Technische Universität Braunschweig

Study Thesis

Application of Machine Learning Algorithms for Operating State Classification and Fault Detection of a CNC Milling Machine

by Lukas Bommes

Matriculation Number: 4367361

Brunswick University of Technology
Faculty of Mechanical Engineering
Institute of Machine Tools and Production Technology

and

Agency for Science, Technology and Research
Singapore Institute of Manufacturing Technology
Department of Manufacturing Execution and Control

Examiner: Prof. Dr.-Ing. Christoph Herrmann
Supervisors: Dipl.-Ing. Benjamin Neef, Dr. Lee Kee Jin

Submitted on 19th October 2017

Abstract

Aim of the following work is the automated detection of different states of a CNC milling machine based on measurement of the overall current consumption. The machine states hereby refer to different switching combinations of the internal components of the machine, such as motors, pumps and axes drives. The second goal is automated detection of machine faults resolved to the level of individual machine components. To achieve those two objectives a machine program is developed which enables the machine to be operated in a well-defined manner with isolated movement of the individual machine axes and isolated activation of additional components. This program is executed and the overall current consumption of the machine is measured to generate training and test datasets. This data is then automatically cut into smaller segments, each representing an individual movement of any of the machine axes. Moreover, these segments are automatically labeled. Based on the thus processed data different classifiers utilizing classical signal processing, such as dynamic time warping and cross-correlation methods as well as modern machine learning algorithms like k-nearest-neighbour, decision tree, random forest, AdaBoost, support vector machine, stochastic gradient descent and naive bayes, are trained and evaluated. Aim of these classifiers is offline classifiaction of segments of the current measurement data, which is equivalent to prediction of switching states of the machine components within the measurement data. Systematic machine faults on the other hand are detected by comparing extracted features of newly acquired test data with already present training data by simple distance computation of according clusters in feature space. Moreover, random faults are detected by an outlier search based on the local outlier factor. Important conclusions of this work are that all considered classification algorithms are able to predict the machine states within the test data with a classification accuracy of up to 97.7 %, which is achieved by the random forest classifier. Moreover, the proposed algorithms for machine fault detection perform as expected and are able to detect even minor changes on the milling machine, such as removal of the tool from the main spindle. The following work is dedicated to people interested in machine learning and computer science in general. It is especially interesting for those who want to see how typical machine learning algorithms can be applied to a real-world problem. Apart from this the work is valuable for everyone who wants to design a cheap and flexible system for predictive maintenance of machine tools without having to tap the internal machine controller.

Keywords: Machine Learning, Pattern Recognition, Classification, Predictive Health Monitoring, Fault Detection, Anomaly Detection, Cross-Correlation, Dynamic Time Warping, k-Nearest-Neighbour, Decision Tree, Random Forest, AdaBoost, Support Vector Machine, Stochastic Gradient Descent, Naive Bayes, Feature Space, Feature Selection, Signal Processing, Machine Tool, Manufacturing Technology, Python, Scikit-Learn, Matlab

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

KDD	Knowledge Discovery in Databases
ML	Machine Learning
PR	Pattern Recognition
AD	Anomaly Detection
AC	Alternating Current
DC	Direct Current
D.C.	Duty Cycle
N.A.	Not Available
NC	Numeric Control
CNC	Computer Numeric Control
DMU	DMU 100 Monoblock®, NC Milling Machine
RMS	Root Mean Square
L1, L2, L3	Three phases of the power supply
C	Programming Language
SQL	Structured Query Language, Database language
SSH	Secure Shell, cryptographic network protocol
GUI	Software with Graphical User Interface
ASA	Accurate Shape Averaging
DTW	Dynamic Time Warping
DDTW	Derivative Dynamic Time Warping
HDTW	Hybrid Dynamic Time Warping
SDTW	Scaled Dynamic Time Warping
CDTW	Cubic-Spline Dynamic Time Warping
ICDTW	Iterative Cubic-Spline Dynamic Time Warping
PSA	Prioritized Shape Averaging
AWARD	Adaptive Warping Window
DBA	Dynamic time warping Barycenter Averaging
HMM	Hidden Markov Model
DFT	Discrete Fourier Transform
XCOR	Cross-Correlation
SpADe	Spatial Assembling Distance
KNN	k-Nearest-Neighbour, Classifier
DT	Decision Tree, Classifier
CART	Classification and Regression Tree, Decision Tree Classifier
CHAID	Chi-Square Automatic Interaction Detection, Decision Tree Classifier
ID3	Iterative Dichotomiser, Decision Tree Classifier
AdaBoost	Adaptive Boosting, Meta-Algorithm for classification
SVM	Support Vector Machine, Classifier

SGD	Stochastic Gradient Descent, Optimization Method
FFT	Fast Fourier Transform, Integral transformation between time and frequency domain
AUC	Area Under Curve, Performance Measure
TP	True Positives
TN	True Negatives
FP	False Positives
FN	False Negatives
FPR	False Positive Rate
LOF	Local Outlier Factor
API	Application Programming Interface

1 Introduction

Computer numerical controlled machine tools, such as milling machines, injection moulding machines, grinding machines, honing machines, welding machines, lathes and many more, play a major role in the landscape of today's manufacturing facilities for a wide range of products like cars and aircrafts. These machines are highly complex mechatronical structures involving a large amount of different components, such as drives, gear boxes, electrical controllers, motors and pumps. Each individual component is prone to error, wear or even complete breakdown leading to standstill of the entire machine or even worse the entire production line which the machine is integrated in. This is accompanied by enormous cost and production delay of the product. Even if the machine does not break down completely a faulty component or worn tool could decrease energy efficiency of the machine and lower qualitiy of the machined product. Out of this arises the need for continuous monitoring of the machine health condition. Of course, such monitoring should occur in-process when the machine is running to prevent unnecessary downtimes for inspection or repair. The machine operater can then always be informed about the current health state of the machine and warned whenever the machine enters a critical state and is likely to break down in the near future. To build according predictive maintenance systems additional information about the machine, for example the operating states of internal components, is needed. The first idea arising to gather this information is interfacing the machine controller and retrieving the necessary information from it. However, current machine controllers very often offer no easy way to access internal machine data and if they do so, interfaces are usually proprietary which makes development of highly specialized software for every single type of machine controller neccessary. Moreover, the data accessible by the internal machine controller might not be sufficient for proper health condition monitoring of the machine as sometimes more data is needed which requires application of additional sensors within the machine. Due to the issues arising from direct access of the machine controller another more flexible solution has to be found. The solution proposed in this work relies on the main power line of the machine to gather information about its health condition. Instead of having the need for elaborate hardware and software changes on the machine controller it is sufficient to connect a current measurement system to the three main power lines of the machine tool to be able to predict its current health state as well as detect possible malfunctions. This solution is microinvasive, requires no further sensors for later health condition monitoring and, what is most important, it is a very cheap and flexible solution. Under the assumption of having already trained a machine learning model on a specific machine type this solution is applicable to any type of machine regardless of its overall purpose or the present machine controller. Such a solution is especially intersting for small companies owning only a few machines and searching for a cheap and easy way to apply predicitve health monitoring solution.

Because of the great benefits of an easy-to-apply predicitive health monitoring system, this work focuses on providing methods which help developing such a monitoring system. The developed

methods comprise different kinds of pattern recognition models for supervised classification of different operational modes of a machine tool as well as methods for unsupervised detection of different possible machine faults. The topic of this work is a typical pattern recognition task as specific patterns within a large amount of measurement data have to be found and classsified. Pattern recognition itself is a subfield of machine learning which is a topic of computer science dealing with algorithms that are able to learn from data. Learning means internal model parameters are adopted to the data to model the underlying structure of the data. Such a model can then be applied to a new dataset to automatically gain knwoledge from it.

Current research in the field of predicitve health monitoring focuses especially on detecting the health state of a single component, such as a motor or a bearing. For instance [1–4] introduce several methods for conditional health monitoring of bearing based on measurement of acoustic emission. In [5] mechanical vibration is measured to predict bearing damage of a motor. Many papers such as [6–9] focus on detection of faults in motors as these are the most common component in manufacturing machines and robots. Detected faults are for instance broken rotor bars, stator winding faults and motor bearing faults. Apart from machine tools condition monitoring is conducted in other fields as well. For example [10, 11] develop condition monitoring systems for wind turbine generators which are prone to similar errors like those occurring in smaller electrical motors. Apart from predicting whether a component has a specific fault or not a lot of papers in the field of predicitive maintenance focus on prediction of tool wear. For instance [12–16] utilize different machine learning methods to predict continuous tool wear of a cutting machine. Moreover, [17–23] offer only a small collection of papers covering this topic as well. Searching for papers that focus on larger and more complex systems, such as the entire machine tool, makes clear that most research of the past years has focused only on single components which are easier to analyze, whereas nearly no research regarding entire machine tools has been conducted. Moreover most of the research projects aim on detecting machine faults rather than operational states of the machine. It should be pointed out that fault prediction is mostly carried out as a supervised learning task meaning that possible fault states are exactly known and data containing labeled faults is existent. Predicting machine faults by unsupervised methods is less often found in literature. Finally, it becomes explicit that the idea of using the current consumption of the machine as measurement quantity for determining the operational and health state of a machine is not really wide-spread in the research community. Instead, many papers focus on different measurement quantities, such as mechanic vibration, acoustic emission, temperature or surface properties of the workpice, which requires installation of additional sensors on the oberserved object. The only paper found during literature research covering a similar topic to the one represented in this work is [24]. Machine learning methods are utilized there to detect the current operational state of an injection molding machine based on measurement of the current consumption of the machine. However, the operational modes considered in this paper do not resolve single components within the machine, but only distiguish between 7 more general operational modes of the machine. This shows the neccessity of research focusing on prediction of operational states of indivudal components in a larger system, such as a machine tool, and on resolving detected machine faults to the component level.

To close this gap in current research this work aims at developing classifier models for prediciting which components within the considered milling machine of type DMU 100 Monoblock® are currently activated. Instead of only deciding for one specific kind of algorithm the work aims

at implementing a variety of different algorithms and comparing their performance on the given classification problem. So, on the one hand classical signal processing methods like cross-correlation and dynamic time warping are utilized and on the other hand modern machine learning algorithms like support vector machine, decision tree and many more are applied to the problem. Desired outcome of this part of the work is to develop at least one algorithm that is able to detect which of the components contained in the machine are activated at any given point in time. Apart from this, the second goal of this work is the development of an algorithm that is able to detect faulty behaviour of the machine which differs from the machine behaviour during the training phase of the algorithm. Rather than training an algorithm on different known machine faults the detection shall be done in an unsupervised manner, which means that no specific faults are known. The algorithm shall only detect deviations from nominal beviour and inform the machine operator about this fault. For this purpose outlier detection as a common method for anomaly detection as well as a newly proposed algorithm are used. This new algorithm determines the machine health condition based on distances between clusters in feature space. Aim of this work is not to provide a production-ready health condition monitoring product for the considered milling machine. Instead fundamental research which provides an insight into detection of operational states of individual components within a more complex structure only based on information about the overall current consumption is conducted. Additionally, valuable insights into the detection principle of different kinds of machine faults are given.

The work is organized as illustrated in fig. 1.1. After this introduction follows a brief overview of the theoretical concepts of knowledge discovery in databases, machine learning, pattern recognition and anomaly detection in chapter 2. This chapter focuses only on the most basic concepts as more detailed information about the theory behind algorithms and methods used in the work is given in the first section of each chapter. This ensures a clean structure of the work and enables the reader to read individual chapters without missing out the necessary fundamentals for understanding a chapter. In chapter 3 follows a detailed analysis of the milling machine which is the object of research in this work. The internal structure of the machine is examined and individual components which influence current consumption on the main power line are analyzed. In chapter 4 measurements on the main current line are conducted providing the training and test datasets which are needed for the later classsification of machine states. Those states are also defined in chapter 4 as well as a special machine program which enables the creation of a measurement signal that optimally isolates the impact of different internal components of the machine onto the main current consumption. Subsequently, chapter 5 covers the problem of segmenting the measurement dataset. As the measurement data is one single time sequence containing all information about the machine behavior this information needs to be extracted by appropiately segmenting the time sequence into smaller subsequences. Moreover these segments need to be labeled. For this purpose a Matlab GUI tool which enables the user to manually segment and label the measurement data is initially developed. Second, a fully automated approach for this issue which is much faster and more reliable is developed. After successfully preparing the measurement data chapter 6 and chapter 7 focus on the actual classification task. Both chapters deliberately have a very similar structure which makes clear that both chapters cover the same problem only featuring different methods. Chapter 6 classifies different segments of the measurement data directly in the time domain by comparing each

segment with a previously generated template. In contrast, chapter 7 classifies segments by means of modern machine learning algorithms such as k-nearest-neighbor, decision tree, random forest, support vector machine and so on. After developing and evaluating classsifiers for machine state prediction the problem of machine fault detection is covered in chapter 8. Datasets representing different machine faults are synthetically generated and acquired on the slightly modified milling machine and algorihtms are developed to detect these faults.

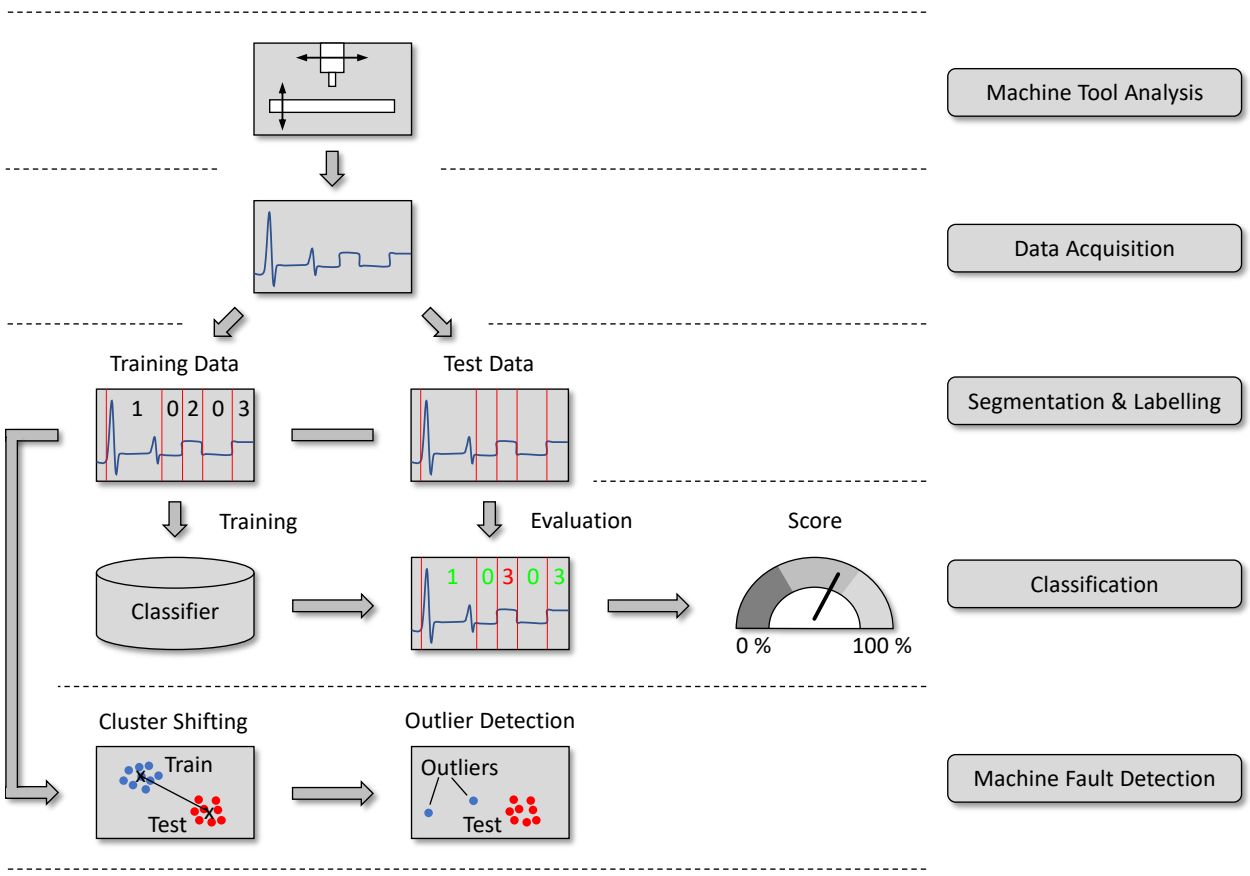


Figure 1.1: Overview of the workflow of the subsequent work.

2 Theoretical Concepts

This chapter gives a brief introduction into the concepts of knowledge discovery in databases, machine learning, pattern recognition and anomaly detection. Aim of this introduction is to clarifiy the differences between those disciplines – as they are often confused – and the terms used analogously and to provide a useful basis for understanding the analysis following in this work. The topics are only covered briefly as a complete introdution would burst the limits of this chapter. More detailed foundations to the individual disciplines are presented in the individual theory sections of the following chapters.

2.1 Knowledge Discovery in Databases

The concept of *Knowledge Discovery in Databases* (KDD) was first introduced in [25] and describes a field of computer sciences which is focused on extraction and generation of new knowledge from large datasets. Such knowledge can be patterns which are new, universal, useful and comprehensible. Given for instance a dataset containing the electrical energy consumption of a city over a period of a month, the KDD process can be used to gather knowledge from that data. It might be found that energy consumption is usually higher during daytime than during the night and that there are times during the day, usually in the morning and during lunch time, when the consumption peaks. These insights are the actual knowledge gathered through the application of the KDD process. Because of the large growth of data stored in online databases over the last decades KDD is one of the ermerging technologies nowadays and yields large benefits for multiple different application fields such as business [26, 27], health care [28, 29], manufacturing technology [30–32], fraud detection [33–35], criminal investigation [36, 37], bio informatics [38, 39] and many more. The term KDD is often used analogously to the term *Data Mining*. However KDD comprises additional steps of data acquisition, data selection, data preprocessing and later interpretation of results while Data Mining only refers to the analysis of already preprocessed datasets. Figure 2.1 shows the typical steps of a KDD process. This process starts off with a dataset which might contain a large amount of data. To focus the knowledge extraction process often only a subset of this dataset is selected. For instance, the dataset could contain the energy consumption of a city over a period of ten years, however to generate a report for the recent year only data of this year is needed and is therefore selected to go into the next process step of preprocessing. Here, the data is cleaned meaning noise is removed from the data and missing values are handled. The next step comprises the transformation of the preprocessed data. This involves for instance rescaling of the data, data reduction and projection into lower dimensions. Data reduction can be done by feature extraction which means alternative quantities (features) are created, which represent the underlying data while having a much lower complexity than the data itself. From the set of features a selection can be made to reduce the amount of data further. After this, the actual knowledge generation or extraction by

means of Data Mining takes place. Here, different algorithms, such as decision trees or neural networks can be applied to the reduced data to find patterns within the data or understand the underlying structure of the data. The found knowledge is then evaluated and interpreted. The last step of the KDD process is acting on knowledge, which means further actions are carried out based on the findings of the prior process steps such as reporting to intersted parties, documentation or comparison to prior knowledge gathered from the dataset. KDD is an iterative process and the mentioned process steps can be repeatedly conducted in a loop. For instance, it can be neccessary to tweak model parameters in the Data Mining step based on results of the evaluation step. [25, 40]

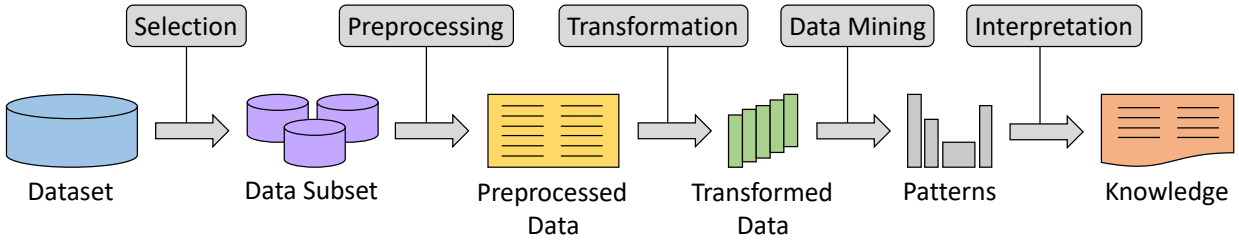


Figure 2.1: Overview of steps composing the KDD process. [25]

2.2 Machine Learning

Machine Learning (ML) is another term for the above mentioned Data Mining step in the KDD process. It was introduced first in [41] and is a subfield of computer sciences dealing with algorithms that learn from data, meaning they are able to conduct data-driven predictions or decisions without the need for explicit programming. This is done through building a model containing a set of parameters which are fit to input or training data. The model is then evaluated on a different dataset and can be used for making predictions or decisions based on new data, the so called test data. So the main purpose of machine learning within the KDD process is discovery of patterns or describing the underlying structure of the training data which both can be seen as gaining knowledge from the dataset. [25, 42]

For the purpose of discovering new patterns and describing the structure of data multiple different machine learning methods have been developed, such as classification, regression and clustering. *Classification* is a task where the machine learning algorithm has to built a model which maps input data to a specific category or class within a finite set of possible classes. Regarding the example above classification could involve categorizing individual consumers in a city based on their average monthly energy consumptions into the three categories *single household*, *family household* and *factory*. The model learns to distiguish between these classes based on training data which consist of labeled examples. In this case the training data could be a table containing the average monthly energy consumption of a consumer and the type or class of the corresponding consumer. The model tries to find a correlation between the type of consumer and its average monthly energy consumption. This model describes the structure of the training data and is able to predict the unkown type of consumer within a new unlabeled test dataset based on only presenting a monthly energy consumption to the classification model.

Regression describes a task where the machine learning model has to map the input data to a real valued (continouus) output variable. A regression model fit to an input dataset can be used to predict the values of the outcome variable for a set of unkown test data. Considering the energy consumption example regression could be used to map the number of people within a household to its average monthly energy consumption. So by creating a regression model for this case the correlation between the feature of number of people in the household and the monthly energy consumption as continouus outcome variable is estimated. If now a new household subscribes to the energy provider the provider can predict its monthly energy consumption by only asking for the number of people living in the household. *Clustering* is another machine learning task which primarily aims at discovering the underlying structure of the input data by means of aggregation. The clustering process groups input data into different categories or clusters. Important difference to classification is the fact that the input data is not labeled. So the cluster algorithm does not know which classes exist in the input dataset, but tries to determine this on its own. Given the energy consumption example a cluster algorithm could look at the average monthly energy consumption of each household or factory in a city. Without knowing whether a household is a single household, a family household or a factory the cluster algorithm will find that there exist three categories in the training data which are characterized by very similar energy consumptions within each category but very different energy consumption between the categories. It is now task of the human operator to interpret the meaning of the found clusters. Apart from the three described methods of machine learning there exist more like dependency modeling, summarization or change detection which have different goals. However one can find that the majority of machine learning problems can either be solved by classification, regression or clustering. [25, 40]

Above another criterion for categoirzing different machine learning methods is implicitly introduced. It can be distiguished between *supervised* and *unsupervised* machine learning methods. Supervised learning refers to all methods involving a labeled training dataset, where desired outcomes (labels) are assigned to each sample of the training dataset. The machine learning model learns during training the correlation between the input data and the desired output label and can then apply a new label to unseen test data in a later prediction step. Examples for supervised learning are classification and regression. On the opposite to that unsupervised learning is based on unlabeled training data. So it is not known, which desired outcome belongs to the samples of the training dataset. Therefore unsupervised machine learning models usually do not aim at prediciting any outcome for testdata, but rather aim at describing the structure of the input data as it is done for instance during clustering. These methods are called unsupervised, because there is no way to verify if the machine learning model represents the input data appropiately. Besides supervised and unsupervised learning there is a third category called *semi-supervised* learning. Semi-supervised learning regards to datasets where only a few samples are labeled while the majority of samples is unlabeled. Usually one would first determine the class labels of the unlabeled samples based on the labeled samples and then feed the now fully labaled dataset into any supervised learning algorithm. [40, 43, 44]

As machine learning is the most important component of the KDD process research of the past decades has focused on developing a large number of different algorithms which implement the above described functionality for knwoledge extraction from datasets. Common algorithms are the k-nearest-neighbour algorithm, decision trees, support vector machines and neural networks.

Despite haveing the same principle goal they operate differently and are very diverse by means of computational expense, execution time and how well they are suited to a given dataset. A more detailed description of different machine learning algorithms is given in the theory section of chapter 7. [45]

2.3 Patterns and Pattern Recognition

To be able to define the field of *Pattern Recognition* (PR) it is necceassary to first understand what a *pattern* is. A pattern can be defined as abstraction describing a physical object by means of a set of measurements. These measurements are also called features. Typical patterns are for example the scan of a fingerprint, a human face within an image, a handwritten letter, a gene sequence within the human DNA, a recorded spoken word, financial transaction data representing a normal transaction or a fraud, a word as sequence of characters within an email or a set of quality metrics of a machined workpiece. Besides that there are many more examples for patterns, as nearly every object in the phyisical world can be seen as pattern, when appropiate measures for describing the object are found. [44, 46, 47]

The term pattern recognition is sometimes used in exchange to machine learning [45], however it is more appropiate to define pattern recognition as an independent research field as it comprises other methods which are not related to machine learning. The goal of pattern recognition is either discovery of new patterns or classification of known patterns in an input dataset. This can be achieved by utilizing machine learning algorithms such as clustering, classification or regression algorithms. However, and this is the major difference between machine learning and pattern recognition, detection and classification of patterns can also be achieved by applying more classical statistical or signal processing algorithms. So pattern recognition does not neccessarily involve a machine learning component, however machine learning is provides great algorithms for pattern recognition and can significantly boost performance of a pattern recognition system. [48]

Possible application for pattern recognition are numerous. They are found for example in the healthcare industry for computer-aided diagnosis like cancer screening, in face recognition systems, fingerprint sensors in mobile devices, handwriting recognition, licencse plate recognition, speech recognition and email spam filtering. [49, 50]

2.4 Anomaly Detection

Anomaly Detection (AD) is a special classification problem that tries to find abnormal behaviour, so called anomalies or outliers in a dataset or datastream. Such anomalies are patterns in the dataset which deviate from the patterns on which the classifier model is trained. The goal of anomaly detection is to find those anomalies with a minimum error rate, meaning all occuring anomalies should be detected, whereas no nominal patterns should be classified as anomalies. Examples for anomaly detection are for instance fraud detection, network intrusion detection [51] or health monitoring of technical systems. Detection of anomalies in any kind of system,

whether technical or not, has a high relevance as anomalies potentially carry a risk and could lead to further damage in the system as well as additional cost due to system breakdown.

For finding anomalies multiple different methods have been proposed. The first one uses a measure to calculate the similarity of a data sample to the other samples within the dataset. This is a density based approach for outlier detection. As outliers deviate from the other patterns in the dataset, they usually lie far away from any of their neighbouring samples within feature space. So their surrounding has a low density while the surrounding of any nominal data sample is characterized by a high density. By measuring this density and comparing the measure to a threshold value the anomaly detection algorithm can distiguished between outliers and nominal samples. Measures are for instance the distance to the k-nearest-neighbours [52] or the *local outlier factor* (LOF) which takes the concept further and calculates a densitiy measure which takes the average density in the environment of a data sample into account when computing the density measure for that sample [53]. Other anomaly detection techniques make use of classifier models such as slightly modified support vector machines which distiguish between nominal data samples and abnormal samples [54]. All these methods only detect individual outliers that represent rare events which only occur once. However another class of anomaly creates multiple abnormal data samples which are similar to each other, but do not refer to any known pattern. As example for this kind of anomaly one could imagine a temperature sensor which breaks down. After break down all subsequent measurement values differ by the same amount from values measured by the fully functional sensor. So when analyzing the dataset one could see two clusters in the dataset. One containing all datasamples of the operational sensor and one all samples which were measured by the broken down sensor. By utilizing clustering algorithms like k-means such clusters can be identified and labeled as anomaleous [55, 56]. Besides these techniques there exist many more such as statistical outlier detection, correlation based outlier detection, fuzzy based outlier detection or replicator neural networks. [57, 58]

3 Machine Tool Analysis

The following chapter aims to give a brief overview of the DMU 100 Monoblock®, which is subject of the following studies. To this first the most important performance parameters of the machine are being described and afterwards a deatailed summary of all contained electrical power consumers within the machine is given. The main purpose of this chapter is to gain domain specific knowledge, which is essential for later data analysis and classiciation. This is an important step, because without comprehensive understanding of the machine's internal structure no expedient conduct of the subsequent analysis tasks is possible.

3.1 Key Features of the Machine

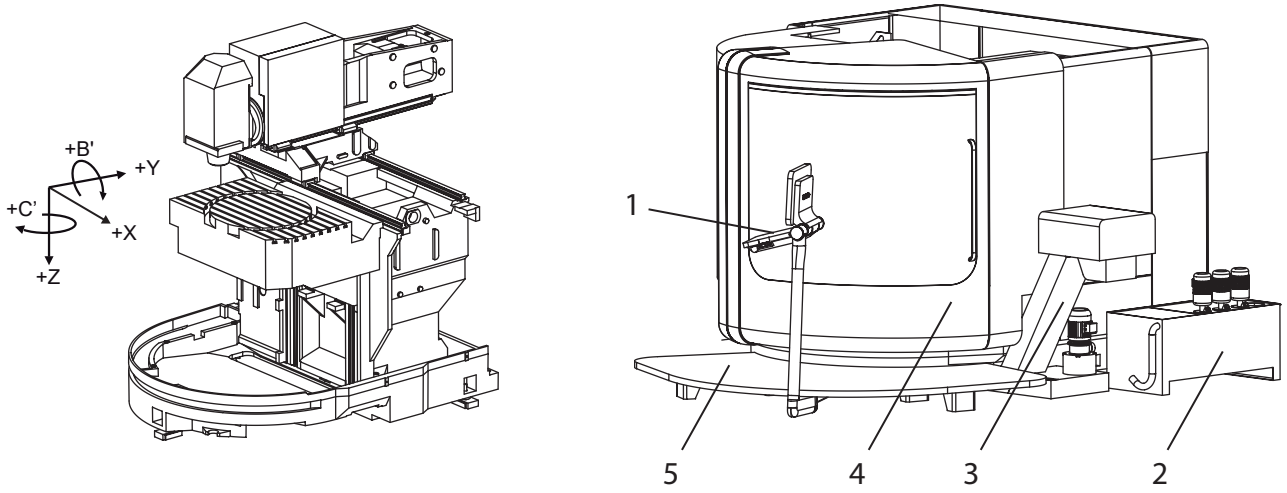
The DMU 100 Monoblock® standing in the Institute of Machine Tools and Production Technology at the TU Braunschweig is a five-axis CNC machining center having a 3D contouring control of type Heidenhain iTNC 530, an automatic tool changer for up to 32 different tools as well as miscellaneous additional features like a laser measurement and tool break detection system by Blum-Novotest, a fume separator, an overhead shower coolant system and a chip conveyor. Figure 3.1a shows the machine without it's casing and any external machine components. Moreover the definition of the machine's coordinate axes is being shown, whereby the arrow tips point in positive axis directions. In contrast fig. 3.1b depicts the machine including it's casing, the machine controller and additional external components like the chip conveyor or the coolant tank with integrated coolant pumps. A detailed summary of the machine's technical performce parameters can be found in table 3.1. [59]

3.2 Functional Modules of the Machine

The sole knowlegde of the performance parameters shown in table 3.1 is not sufficient for conducting any of the following analysis tasks. Therfore further examination of the machine's internal structure is necessary. Because of the later classification of the machine states being based on measured electrical quantities of the machine's main power line, it is essential to know which individual electrical power consumers are built into the machine and what rated electrical parameters they have. This includes rated voltage, rated current and rated power consumption. In order to conduct this analysis systematically, the machine is being divided into individual functional modules. These modules are the media supply unit, the coolant system, the control cabinet heat exchanger, the chip conveyor, the cooling unit, the fume separator, the tool changer, the main spindle and axes drives, the NC dividing attachment as well as any electrical consumers hooked up to the 24 V-low-voltage supply. The arrangement of those components within the

Table 3.1: Technical Properties of the DMU 100 Monoblock®. [59, 60]

Component	Property	Unit	Value
X-/Y-/Z-Axis	Max. Travel	mm	1150/710/710
	Rapid Traverse / Max. Feed Rate	mm/ min	30000
	Max. Acceleration	m/s^2	5/5/4
	Max. Feed Force (100 % D.C.)	kN	10/10/15
	Position Sensor Resolution	mm	0.001
Main Spindle	Max. Power (40 %/100 % D.C.)	kW	35/25
	Max. Torque (40 %/100 % D.C.)	N m	119/85
	Tool Fixture	Type	SK 40
Swivel Head (B-Axis)	Max. Travel	°	150 ($-120 \ldots +30$)
	Rapid Traverse Rate	min^{-1}	35
	Max. Acceleration	$°/s^2$	2300
	Min. Swivel Time	s	1.5
	Max. Holding Torque with Clamping	N m	3500
	Nomimal Torque	N m	1244
Rotary Table (C-Axis)	Max. Travel	°	infinite
	Rapid Traverse / Max. Feed Rate	min^{-1}	30
	Max. Acceleration	$°/s^2$	1200
	Max. Torque (40 % D.C.)	N m	2600
	Diameter	mm	800
	Dimensions Rigid Table	mm	1500×800
	Max. Workpiece Weight	kg	800
Tool Changer	Tool Fixture	Type	SK 40
	Tool Magazine	Type	Disc Magazine
	Number of Magazine Places	–	32
	Clamping-to-clamping Time	s	10
NC Dividing Attachment	Centre Hight	mm	160
	Rapid Traverse / Max. Feed Rate	min^{-1}	80
	Max. Holding Torque with Clamping	N m	500
	Holding Torque	N m	120
Others	Machine Weight	kg	10700
	Coolant Tank Volume	L	250
	Controller	Type	Heidenhain iTNC 530
	Chip Conveyor	–	✓
	Mist and Fume Separator	–	✓
	BLUM Laser Tool Measurement System	–	✓



(a) Front view without casing. Axes definitions are shown.

(b) Front view with casing. Control Stand (1), Coolant System (2), Chip Conveyor (3), Cabin (4), Tread Plate (5).

Figure 3.1: Front view of the DMU 100 Monoblock® with and without casing. [59]

machine can be seen in fig. 3.2. In addition fig. 3.3 shows deatailed views of the media supply unit, the coolant system, the chip conveyor as well as the cooling unit.

As can be seen in fig. 3.3a, the media supply unit contains one hydraulic pump and three lubrication pumps as electrical power consumers. Purpose of the lubrication pumps is the distribution of lubricant within the machine. For instance the axis guidances and bearings are being lubricated by the machine lubrication pump in regular intervals. The oil-air lubrication pump serves the lubrication of air within the machine's pneumatic system and the blower air lubrication pump lubricates blower air, which can be alternatively activated for tool and workpiece cooling instead of liquid coolant. The hydraulic pump being built into the media supply unit as well serves the purpose of maintaing hydraulic pressure within the machine's hydraulic system. Whenever the presssure in the diaphragm accumulator drops below a specific threshold the hydraulic pump is being switched on in order to increase hydraulic pressure. [59]

The coolant system depicted in fig. 3.3b contains multiple powerful electrical pumps that transport coolant from the coolant tank into the machine. Coolant pump 1 provides coolant for the internal cooling of suitable tools with channels for inner coolant supply. This pump can be activated via the M7-command and stopped by a M9-command. By contrast coolant pump 2 transports coolant from the coolant tank into the outer coolant cycle. Here coolant is being sprayed onto the tool and workpiece through nozzles at the swivel head. This pump can be switched on by the M8-command and similar to the inner coolant pump stopped by a M9-command. Another pump built into the coolant system ist the coolant spray gun pump which provides coolant for the manual coolant spray gun. Whenever the coolant spray gun is being used by the machine operator, the coolant spray gun pump is automatically switched on. The last pump depicted in fig. 3.3b is the pump for the overhead shower coolant system, which serves the removal of chips from the tool changer door. This pump is optional and not contained in the present version of the machine. Instead the overhead shower coolant system is

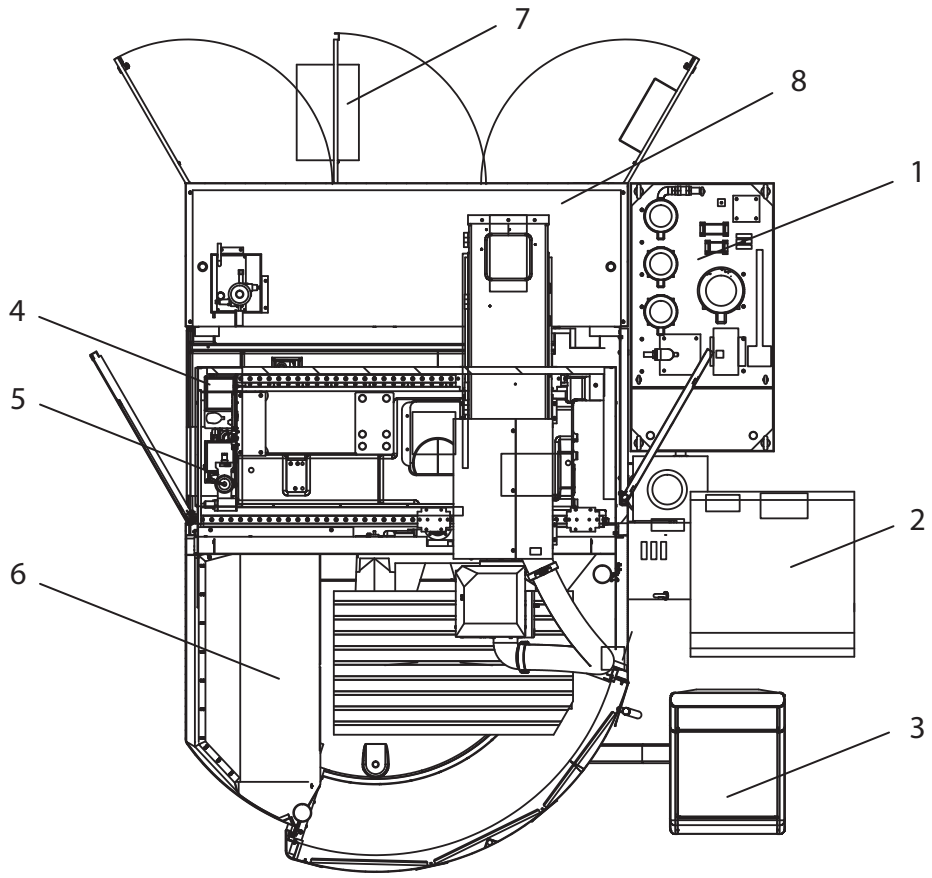


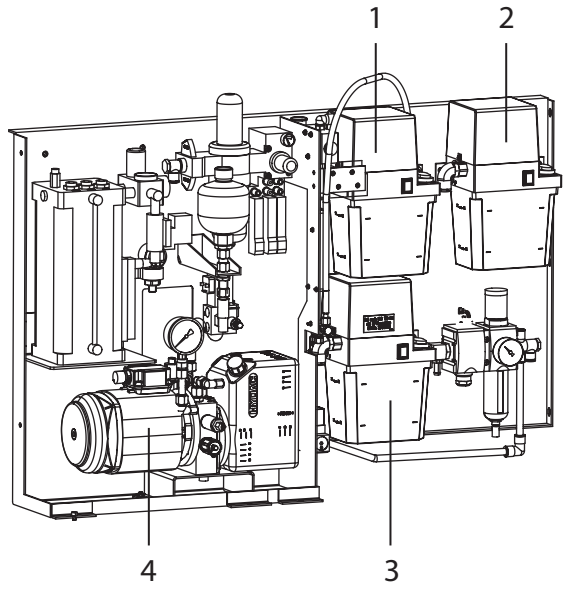
Figure 3.2: Top view of the DMU 100 Monoblock®. Coolant System (1), Chip Conveyor (2), Control Stand (3), Media Supply Unit (4), Cooling Unit (5), Tool Changer (6), Control Cabinet Heat Exchanger (7), Control Cabinet (8). [61]

being supplied with coolant by both coolant pump 1 and coolant pump 2. [59]

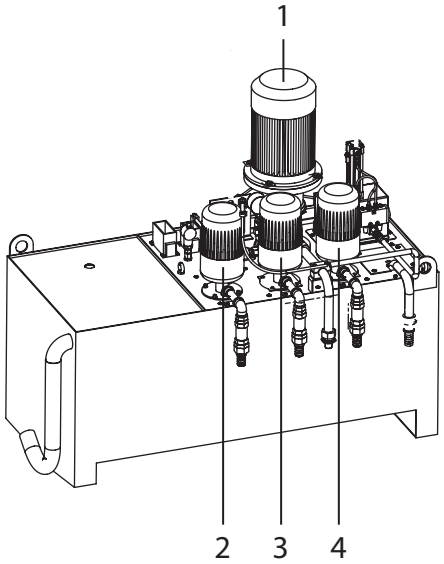
Another important functional module of the machine is the chip conveyor shown in fig. 3.3c, which has the purpose of removing chips from the work area within the machine and transporting them into a container. The chip conveyor contains primarily two large electrical power consumers, first the drive motor for the conveyor belt and second the chip conveyor lifting pump which pumps coolant from the machine cabin into the coolant tank on the outside of the machine. The chip conveyor drive motor can be explicitely activated via a M70-coammand and stopped via a M71-command. However the chip conveyor lifting pump is being activated automatically whenever the coolant level within the chip conveyor's floor tub rises above a specific threshold value. [59]

Figure 3.3 shows the machine's cooling unit that serves the cooling of the main spindle. The unit contains a compressor, a cooling fan and a feeding pump for the coolant. A direct control of the cooling unit is not possible as it has it's own independent temperature controller. [59]

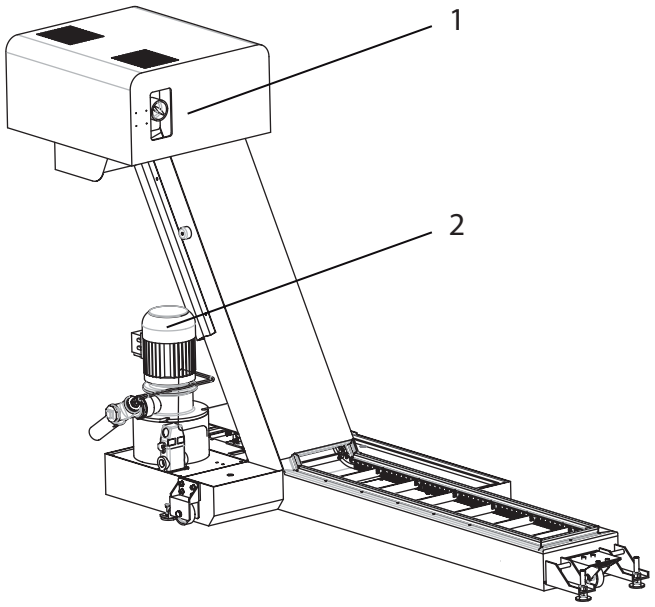
Further important, but not shown machine components are the fume separator, the control cabinet heat exchanger, the tool changer, the NC dividing attachment and other low-voltage



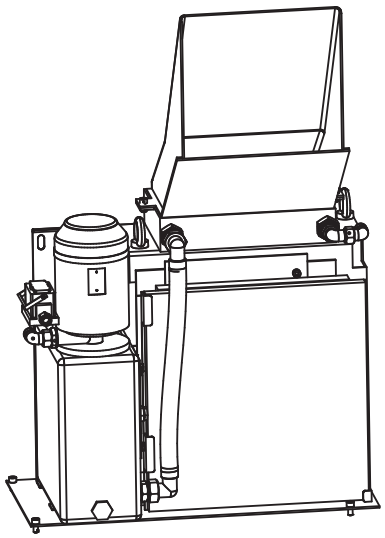
(a) Media Supply Unit. Machine Lubrication Pump (1), Blower Air Lubrication Pump (2), Oil-Air Lubrication Pump (3), Hydraulic Pump (4).



(b) Coolant System. Coolant Pump 1 (for inner cooling) (1), Coolant Pump 2 (for outer cooling) (2), Coolant Spray Gun Pump (3), Chip Rinsing Pump (N.A.) (4).



(c) Chip Conveyor. Chip Conveyor Drive Motor (1), Chip Conveyor Lifting Pump (2).



(d) Cooling Unit. Contains Compressor, Cooling Fan and Feed Pump.

Figure 3.3: Detailed views of the most important subsystems of the DMU 100 Monoblock®. [59, 61]

consumers. The fume seperator transports coolant vapour and other gasses that are generated under certain process conditions and might have negative effects on the machine operator out of the machine cabin and filters those. The fume separator can be activated by a M324-command and is being stopped by a M325-command. The control cabinet heat exchanger is being used for air conditioning of the control cabinet that can be found on the machine's rear side. Similar to the cooling unit for the main spindle the control cabinet heat exchanger works self-sufficient and can not be controlled directly via the machine controller. The tool changer is being activated whenever the controller asks for exchange of the tool clamped in the main spindle. In this case the tool changer door within the machine's cabin is being opened by hydraulic actuators and one of the two turn table magazines, depending on the location of the tool to be exchanged, extends. The tool curently clamped in the main spindle is being loosened and taken from the spindle into the magazine. Subsequently the new tool is being taken from the magazine and clamped in the main spindle. Another external component that can be built into the machine is the NC dividing attachment which allows for horizontal clamping and rotational movement of workpieces. This makes possible combined turning and milling of the same workpiece in one clamping. The NC dividing attachment is being screwed on the machine's table and is an optional module that is not available at the present version of the machine. Beyond the components listed above any electrical power consumers which are hooked up to the 24 V-low-voltage supply need to be considered. Examples for such consumers are the machine lights, the LCD panel of the machine controller as well as the rotating inspection window in the machine's cabin door. [59]

Finally the main spindle and axes drives have to be taken into account. These are the most important components refered to the analysis of elelectrical quantities within the machine's main power line as they are very powerful and therefore have the most impact on the current signal on the main power line. The spindle and axes are being controlled by the machine controller via downstream connected drive regulators and can therefore be moved by explicit machine controller commands. [59]

3.3 Electrical Power Consumers within the Machine

After the internal structure of the machine and the functionality of any external components is now clear, the following section gives a more deatailed insight into the electrical characteristics of the individual components within the machine. Table 3.2 contains any of the electrical power consumers described in section 3.2 as well as the part ID of the corresponding power consumer in the circuit diagram in the machines's documentation and the ID of the circuit diagram sheet on which the according consumer can be found. Moreover the consumers are given unambiguous names that are used throughout the whole work. Beyond that the type of the consumer and if available M-codes for both activation and stopping of the corresponding component are specified. The last column is important for data collection in chapter 4 and describes whether a component is explicitely controllable via controller commands (*controllable*) or being controlled independently of the machine controller and therefore it's state is unknown. To retrieve the current operating state of self-sufficient components a measurement of their electrical quanitites is necessary. Therefore the test-mode for this components is being specified as *measured*. Furthermore there are components built into the machine that have a constant

operating state. For instance the machine lights or the controller's LCD panel are normally always switched on. These components are specified as *constant on* respectively *constant off* in case of the tool magazine drives, which are not being used as will be explained later.

The electrical operating parameters of the power consumers listed in table 3.2 can be found in table 3.3. In case of the consumer being supplied by an alternating current (AC) the number of phases, the permissible frequency of the alternating current as well as the circuit configuration is being specified. Circuit configuration means whether a three-phase AC consumer is being connected to the supply in a star or delta circuit. Apart from that table 3.3 contains the rated voltage, rated current and rated power for any of the consumers within the machine. In addition to that a rotational speed is given, if the consumer is a motor or pump.

Conclusively the electrical power consumers contained in the DMU 100 Monoblock® are being depicted in fig. 3.4 assigned to their superordinate functional modules. The labels to the left of some of the components indicate how the corresponding component can be controlled. In case of the five axes this can be achieved by executing a command for a linear interpolated movement of any of the axis. The main spindle drive, the coolant pumps 1 and 2, the fume separator and the chip conveyor drive motor can be switched on explicitely as well by according M-commands. If a component within the scheme in fig. 3.4 has no such label the component is self-sufficient and can not be controlled directly. As mentioned it's operating state therefore needs to be measured by a current sensor. Components which are not being examinde in the further analysis are being left in a constant operating state indicated by a label stating either *on* or *off*.

Table 3.2: Main power consumers within the DMU 100 Monoblock®. Sheet and Part ID refer to the DMU circuit diagram [61].

ID	Sheet ID	Part ID	Part Name	Type	M-Code	Test-Mode
0	A0-09/001	-A0-09E51	Machine Light Roof	Light		constant on
1	A0-09/001	-A0-09E61	Machine Light Side	Light		constant on
2	A4-01/001	-A4-03P21	Control LCD Panel	LCD		constant on
3	B0-01/001	-B0-01M21	Hydraulic Pump	Motor		measured
4	B1-01/001	-B1-01A21	Machine Lubrication Pump	Motor		measured
5	B1-02/001	-B1-02A21	Oil-Air Lubrication Pump	Motor		measured
6	B1-10/001	-B1-10A21	Blower Air Lubrication Pump	Motor		measured
7	B2-01/001	-B2-01M21	Coolant Pump 1 (for inner cooling)	Motor	M08 / M09	controllable
8	B2-01/001	-B2-01M41	Coolant Pump 2 (for outer cooling)	Motor	M07 / M09	controllable
9	B2-08/001	-B2-08M21	Coolant Spray Gun Pump	Motor		constant off
10	B4-01/001	-B4-01M21	Fume Seperator Motor	Motor	M324 / M325	controllable
11	B5-01/001	-B5-01E21	Control Cabinet Heat Exchanger	Motor		measured
12	B5-10/001	-B5-10M21	Cooling Unit: Compressor	Motor		measured
13	B5-10/001	-B5-10M41	Cooling Unit: Cooling Fan	Motor		measured
14	B5-10/001	-B5-10M61	Cooling Unit: Feed Pump	Motor		measured
15	B6-04/001	-B6-04M51	Chip Conveyor Drive Motor	Motor	M70 / M71	controllable
16	B6-06/001	-B6-06M21	Chip Conveyor Lifting Pump	Motor		measured
17	B8-01/001	-B8-01M41	Rotating Inspection Window	Motor		constant on
18	C1-02/001	-C1-02M31	Main Spindle Drive	Motor	M03 / M05	controllable
19	D1-01/001	-D1-01M21	X-Axis Drive	Motor		controllable
20	D1-11/001	-D1-11M21	Z-Axis Drive	Motor		controllable
21	D1-21/001	-D1-21M21	Y-Axis Drive	Motor		controllable
22	D2-01/001	-D2-01M21	C-Axis Drive	Motor		controllable
23	D3-01/001	-D3-01M21	B-Axis Drive	Motor		controllable
24	D5-01/001	-D5-01M21	NC Dividing Attachment	Motor		constant off
25	F8-11/001	-F8-11M71	Tool Magazine Drive 1	Motor		constant off
26	F8-15/001	-F8-15M71	Tool Magazine Drive 2	Motor		constant off

Table 3.3: Rated electrical characteristics of the main power consumers as documented in the DMU 100 Monoblock® circuit diagram [61].

ID	Current	Phases	Circuit	Voltage	Frequency	Current	Power	Speed
0	DC	–	–	24 V	–	N.A.	65 W	–
1	DC	–	–	24 V	–	N.A.	65 W	–
2	DC	–	–	24 V	–	0.78 A	16.8 W	–
3	AC	3	star	400 V	50/60 Hz	3.70 A	750 W	1380 min^{-1}
4	AC	1	–	230 V	50/60 Hz	0.46 A	105 W	N.A.
5	AC	1	–	230 V	50/60 Hz	N.A.	N.A.	N.A.
6	AC	1	–	230 V	50/60 Hz	0.65 A	144 W	N.A.
7	AC	3	star	400 V	50/60 Hz	1.57 A	750 W	2753 min^{-1}
8	AC	3	star	400 V	50/60 Hz	7.00 A	3000 W	N.A.
9	AC	3	star	400 V	50/60 Hz	1.57 A	750 W	2753 min^{-1}
10	AC	3	star	400 V	50/60 Hz	2.10 A	N.A.	N.A.
11	AC	1	–	230 V	50/60 Hz	4.20 A	580 W	N.A.
12	AC	3	star	400 V	50/60 Hz	2.20 A	1280 W	N.A.
13	AC	3	star	400 V	50/60 Hz	0.26 A	110 W	N.A.
14	AC	3	star	400 V	50/60 Hz	2.20 A	850 W	N.A.
15	AC	3	star	400 V	50 Hz	0.88 A	250 W	13.5 min^{-1}
16	AC	3	star	400 V	50 Hz	4.60 A	2200 W	2850 min^{-1}
17	DC	–	–	24 V	–	0.66 A	16 W	2300 min^{-1}
18	AC	3	star/delta	N.A.	N.A.	N.A.	N.A.	N.A.
19	AC	3	N.A.	N.A.	N.A.	9.10 A	5700 W	3000 min^{-1}
20	AC	3	N.A.	N.A.	N.A.	15.00 A	9900 W	3000 min^{-1}
21	AC	3	N.A.	N.A.	N.A.	9.10 A	5700 W	3000 min^{-1}
22	AC	3	N.A.	N.A.	N.A.	6.10 A	2090 W	4500 min^{-1}
23	AC	3	N.A.	N.A.	N.A.	6.10 A	2090 W	4500 min^{-1}
24	AC	3	N.A.	N.A.	N.A.	4.00 A	1900 W	3000 min^{-1}
25	DC	–	–	24 V	–	1.60 A	42 W	52 min^{-1}
26	DC	–	–	24 V	–	1.60 A	42 W	52 min^{-1}

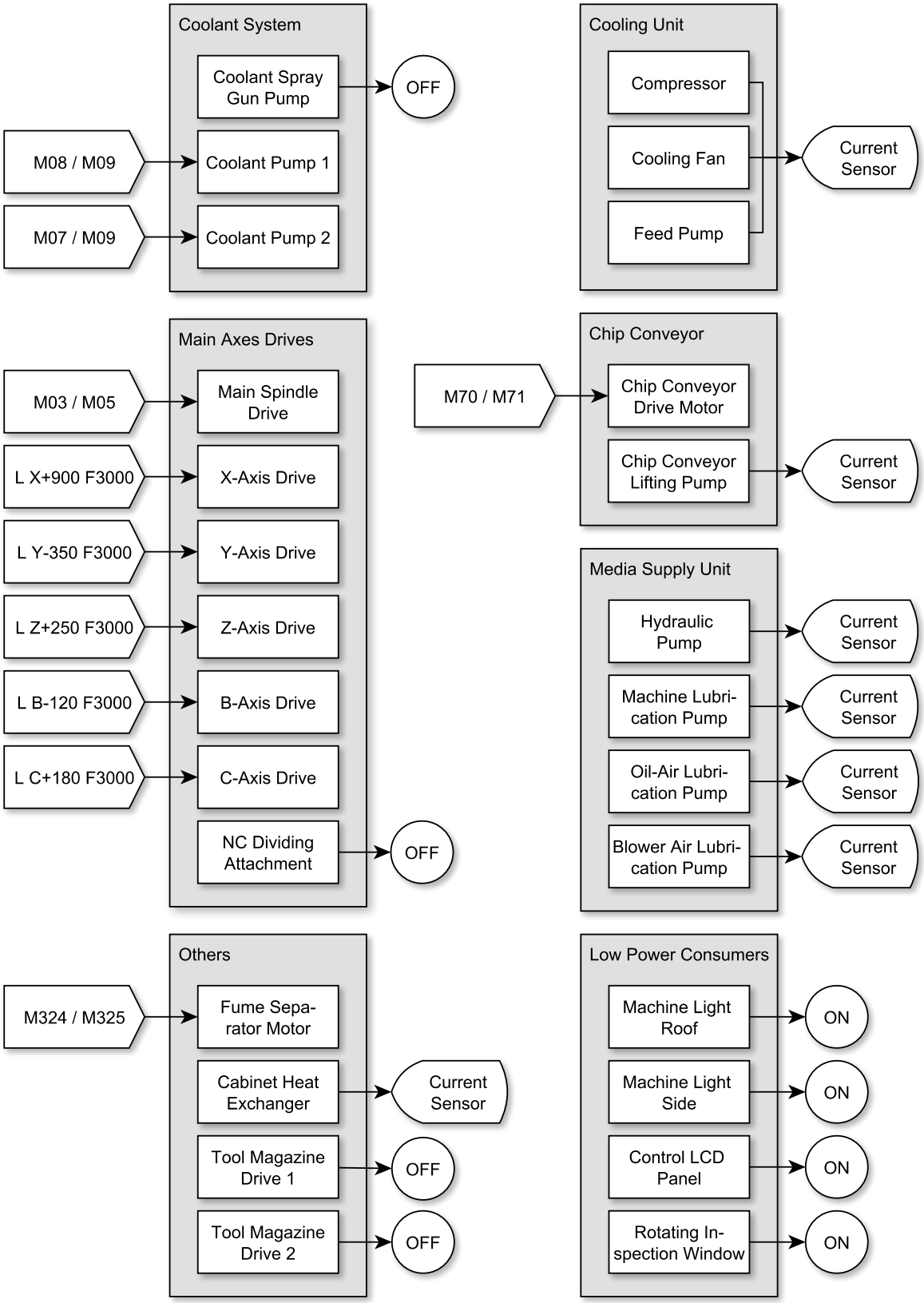


Figure 3.4: Functional modules and main power consumers of the DMU 100 Monoblock®. Inputs and outputs show whether a module is actively controllable during testing, is staying in a constant state or is controlled automatically and its state being measured. [61]

4 Data Acquisition

The aim of the following chapter is the acquisition of measurement data which is the necessary basis for creation of the dataset for classifier design in the forthcoming chapters of this work. Falling back upon the now fully understood milling machine first a brief overview of the data that's required to being measured is given. The available measurement quantites are being described and a choice of quantities needed for further analysis steps is being made. Afterwards the data acquisition toolchain is being explained. This includes a deatailed description of the measurement hardware being used in this work as well as bus and network systems required for data transfer and storage. In the following section the possible different switching states of the machine components are being examined and ordered systematically. In this essential step the possible classes as output variables of the later classification task are being defined. After definition of the classes a test program for the milling machine is being developed that controls the machine and allows for acquisition of training data for the different machine states defined before. In the last section of this chapter the test cycle is being run and the measurements are being conducted. Results of the data measured during the test cycle are being presented and descriped in detail.

4.1 Overview of the Measurement Data

In order to fully understand and model the machine behaviour which is crucial for designing and evaluating a classifier in the subsequent project states a datatset containing all the information about the machine behaviour needs to be created. The first step to achieve this is to program the machine in a particular way so that switching states of all the different machine components follows a well known sequence. This allows to model the correlation between machine component states and patterns wihtin the measured current flow in the machine's main power line. One problem with this approach is the missing contrallability of some of the machine components via the machine controller. For example the control cabinet heat exchanger has it's own temperature controller and therefore works indepentend of the main machine controller. This makes it impossible to know the exact component state from only programming the machine via the machine controller. Apart from the control cabinet heat exchanger the chip conveyor lifting pump, the cooling unit, the hydraulic pump and the three lubrication pumps work self-sufficient so their states need to be measured as well. Moreover the eletrical quantities on the machine's main power line are being monitored in order to find out the correlation between component states and patterns within the machine's current signal. Figure 4.1 shows, which measurands are being acquired during these measurements in order to determine the current component's state. These are current $i(t)$ and power factor $\cos(\varphi)$ for any of the monitored machine components and voltage $u(t)$ as well as active power P for the machine's main power line. Because of the fact, that the machine works on three-phase AC current, the measurements are conducted on

the three individual phases L1, L2 and L3. Only some of the components are connected to just one phase, so that the eletrical quantities for this component need to be measured only on that particular phase.

The three different colors of the measurands in fig. 4.1 illustrate whether the measurand is being used in further analysis steps (green) or not (red). Because it is only necessary to know, wheter the component is currently working or stopped, it is suffiecient to only look at the actual component's current consumption. In order to simplify the classification task, the chip conveyor lifting pump, the hydraulic pump and the blower air lubrication pump are not taken into account in the further classification problem. Ignoring these components though not being controllable directly via the machine controller is possible because their state's only depend on machine actions that are directly controllabe via the machine controller. For example the chip conveyor lifting pump is only activated, when the fluid level within the bottom coolant tank rises above a specific value. If no collant is used, the chip conveyor lifting pump will never be activated. The same applies for the hydraulic pump which is activated only on pressure drop within the machine's hydraulic system. If no hydraulic drives are being used, that means no tool change is being issued via the controller, the hydraulic pump will not be activated. Finally the state of the blower air lubrication pump directly relies on the usage of the compressed air gun. So if the air gun is not used, the pumps's state does not need to be taken into account in the furhter analysis. This simplyfies the classifiaction task enormously as will be explained later in this chapter. A special position is being held by the cooling unit because it's state can be assumed to be constant over time as will turn out in section 4.5. This allows for complete removal of the cooling unit from the classification task because it's state will always be the same, so that a prediction of the component state is not necessary.

4.2 Data Acquisition Toolchain

The following section gives an overview of the toolchain for acquisition of the data being described in section 4.1. This includes a brief description of the measuring hardware as well as the the underlying network architecture for data transfer and storage. A flowchart of the toolchain can be seen in fig. 4.2.

For monitoring the measurands descriped above the current and voltage of the different machine components need to be measured. From these two measurands any other quantity like the power or power factor can be derived. Current measurements on the main power line are being conducted via current transformers of type GMW ASK 412.4 with a conversion ratio of 100 to 5, which means that a measurement current of 100 A leads to an output current of 5 A. Further information of the current transformer can be found in the documentation [62]. In contrast the current measurements of any other of the components listed in fig. 4.1 are being performed with current transformers of type HOBUT 13 Series with a conversion ratio of 60 to 5. Particulars about the current transformer can be found within the documentation [63]. Every current transformer is conected to 3-phase power measuring terminals of type Beckhoff EL3403-0010. These terminals allow for measurement of RMS-values of voltage $u(t)$ and current $i(t)$, active power P, apparent power S, reactive power Q and power factor $\cos(\varphi)$ on all three phases of

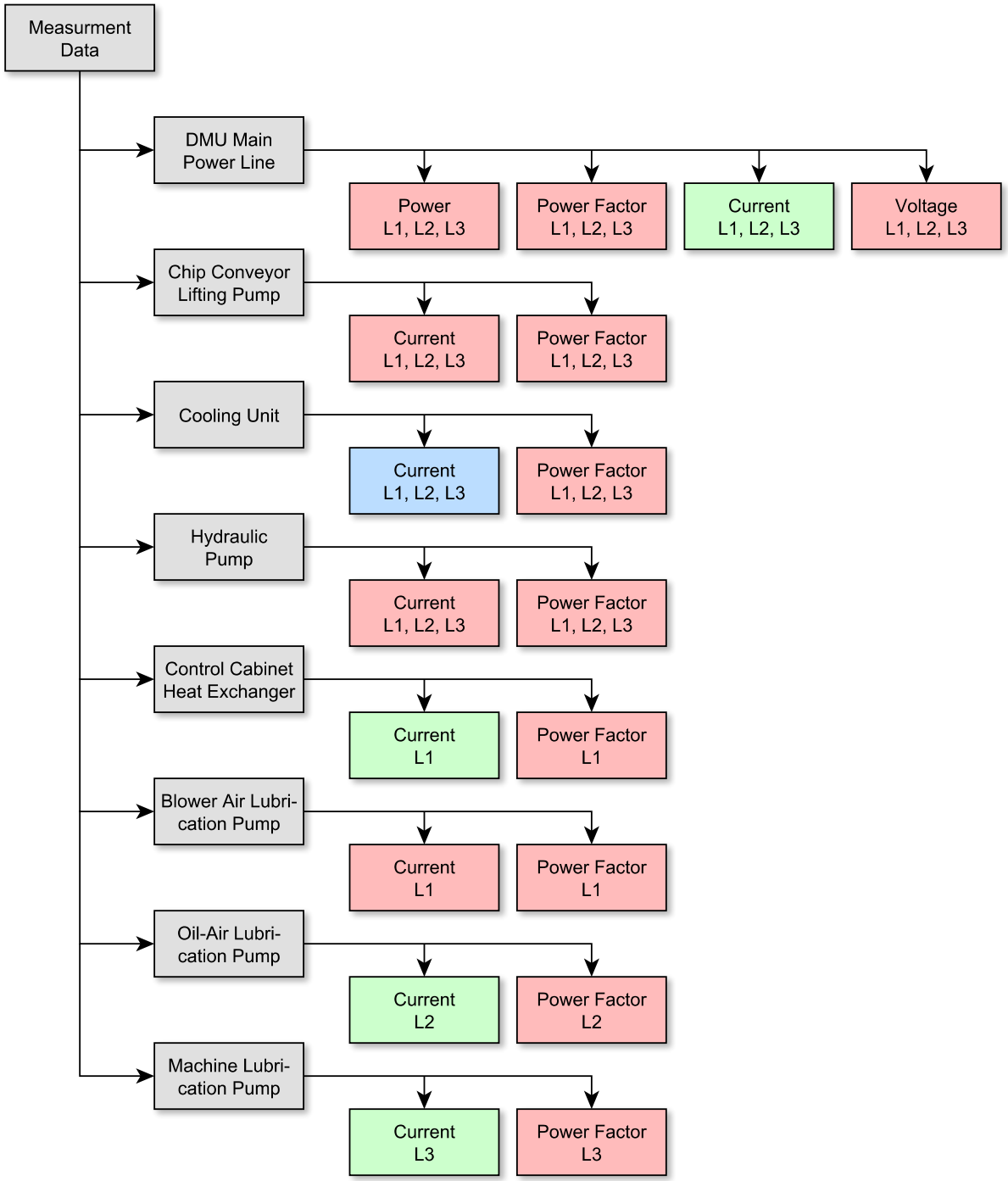


Figure 4.1: Overview of different measurands within the acquired data. Red: Data is being stored in the database but not used in further analysis. Green: Data is being stored and used in furhter analysis. Blue: Data is being stored and used but component state is assumed to being constant. Phase names refer to the input ports of the measurement terminals rather than the actual machine phase which the components are connected to.

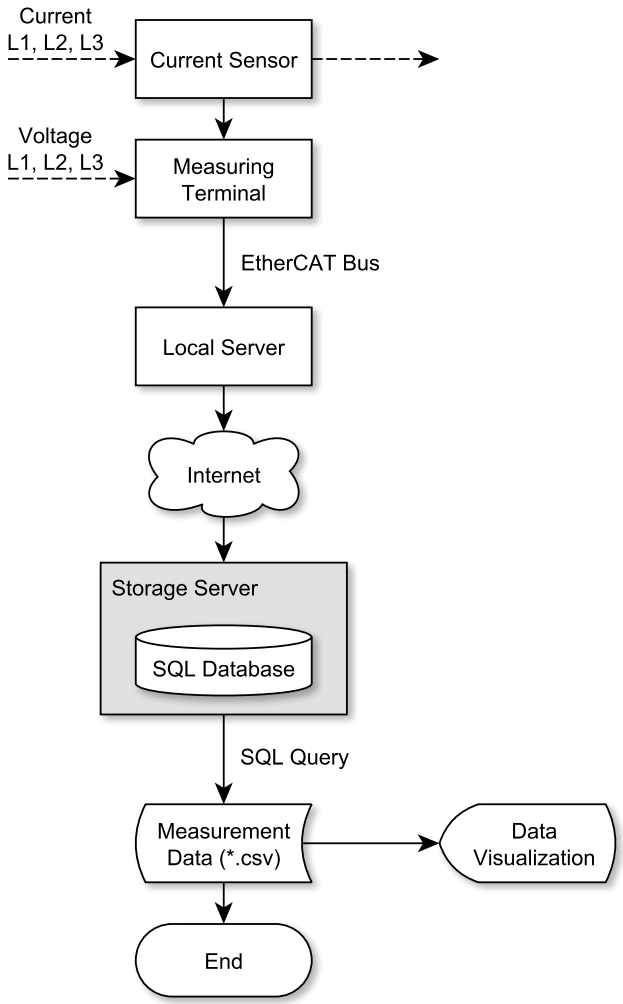


Figure 4.2: Flowchart of the data acquisition process as first step of the classification problem.

the three-phase supply with a resolution of 12 bit. The measuring cycle time is dependand on the frequency of the measured signal, because a measurement value is being determined on each period of the measurement signal. Given a supply frequency of approximately 50 Hz this results in a measurement cycle time of 20 ms. Beyond that the mesurement terminals utilize additonal low-pass and high-pass filters that are activated during the measurement. [64]

Figure 4.3 illustrates all different monitoring terminals built into the machine. It also shows the terminal addresses on the bus and how voltage and current of the three different phases L1, L2 and L3 as well as the neutral conductor are connected to the terminals. The three terminals with the addresses 3:1, 3:2 and 3:3 are used for other purposes and not considered any further. The module to the left with address 3:0 is the EtherCAT-Coupler, that communicates with the measurement software on the local server via the EtherCAT-Bus which is an ethernet like serial communication bus for sensor and actor communication purposes in industrial applications. Table 4.1 contains a more detailed description of the properties of the different measuring terminals connected to the bus. Further information of the terminals can be found in the

corresponding datasheets [64–66].

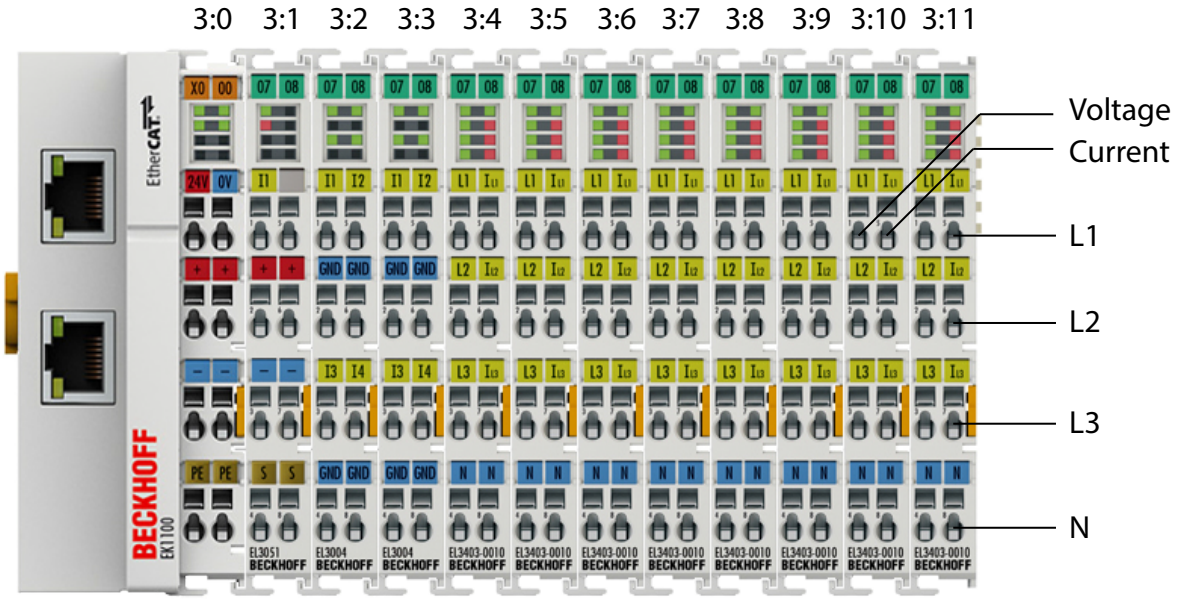


Figure 4.3: Different monitoring terminals on the EtherCAT-Bus. The eight terminals to the right are 3-phase power measuring terminals used for monitoring the electrical current consumption of some of the components of the CNC milling machine.

Apart from the measuring terminal's specification it is important to know, how the different machine components listed in fig. 4.1 are connected to the terminals. This information is contained in table 4.2. The main power line, chip conveyor lifting pump, cooling unit and hydraulic pump are three-phase consumers and require three current sensors and a complete measuring terminal for monitoring each. On the contrary the control cabinet heat exchanger and the three lubrication pumps are connected to only one phase of the power supply so that in theory three single-phase consumers could be connected to a single measurement terminal. To avoid any problems due to signal interference all of the single-phase consumers are connected to an individual measurement terminal, though. This leads to a higher realibality of the conducted measurements. Figure 4.4 shows how to connect the current transformers for a three-phase AC consumer to the three phases L1, L2 and L3 and the neutral conductor N of the power measuring terminals. As can be seen a common neutral conductor is necessary and used as reference point for any current and voltage measurements.

Returning to the data acquisition flowchart in fig. 4.2 the measurement data collected via the power monitoring terminals is transfered over the EtherCAT-Bus to a local machine, which acts as data collection server not only for the considered milling machine, but for terminals in other machines and experiments connected to the EtherCAT-Bus as well. On this server a C-script that communicates with the measuring terminals over the EtherCAT-Bus is being executed via a remote procedure call within the transmission software implemented in Node.js. An editable configuration file on that server allows the user to configure the collection script to new measurement terminals on the EtherCAT-Bus. Among others this script contains information about the terminal ID and address on the bus, the name of the measured quantity and the

Table 4.1: Overview of the measuring terminals on the EtherCAT-Bus.

Address	Type No.	Serial No.	Description	Measurement Range
3:0	EK1100	–	EtherCAT-Coupler (2A E-Bus)	–
3:1	EL3051	–	1-Channel-Analog-Input Terminal	4 . . . 20 mA
3:2	EL3004	–	4-Channel Analog Input Terminal	−10 . . . 10 V
3:3	EL3004	–	4-Channel Analog Input Terminal	−10 . . . 10 V
3:4	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:5	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:6	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:7	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:8	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:9	EL3403-0010	38111106	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:10	EL3403-0010	37141511	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A
3:11	EL3403-0010	37141511	3-Phase Power Measuring Terminal	0 . . . 500 V, 0 . . . 5 A

Listing 4.1: SQL query for retrieval of measurement data from the database on the storage server.

```
SELECT Time, Value FROM Data WHERE Measurement_id = <id> AND Time BETWEEN <t1> AND <t2>
```

calibration factors for the current sensors. Any retrieved data is then written over the Internet into a SQL-database that lies on a data storage server on a different physical machine.

From that database any measured data can be retrieved via an SQL query. For instance listing 4.1 shows how to query all measured data samples consisting of a unix timestamp and a measurement value that lie in between of the two unix timestamps <t1> and <t2>. The parameter <id> indicates what column in the database, that means which measurement quantity, shell be queried. A different table within the same database contains the linkage between measurement ID and the measured quantiy. This allows for faster queries as the processing of the integer value *measurement ID* takes less time to query than the string containing the name of the measured quantity.

The SQL query can be performed directly on the storage server over a SSH connection or remotely via any suitable SQL client software like HeidiSQL [67] or the phpMyAdmin web client [68]. Queried data samples can then be exported into a comma separated text file (CSV) which can be loaded easily into MATLAB or Python for further processing like data visualization, feature extraction and selection as well as designing a suitable classifier.

Table 4.2: Mapping between connected components and the measuring terminals.

Address	Connected Phase	Attached Component	
		Part ID	Part Name
3:4	L1, L2, L3	–	DMU Main Power Line
3:5	L1, L2, L3	12, 13, 14	Cooling Unit (complete unit measured)
3:6	L1	11	Control Cabinet Heat Exchanger
3:7	L1	4	Machine Lubrication Pump
3:8	L1	6	Blower Air Lubrication Pump
3:9	L1	5	Oil-Air Lubrication Pump
3:10	L1, L2, L3	3	Hydraulic Pump
3:11	L1, L2, L3	16	Chip Conveyor Lifting Pump

4.3 Definition of Machine States

After succesful retrival of the measurement data the next important step that needs to be considered in the classfication task is the definition of the classes which are discrete states of the output variable of the classification algorithm. These classes represent different combinations of states of any of the machine's components. As mentioned in section 4.1 some of the components are being neglected in the definition of possible machine states respectively classes in order to reduce the number of classes and hereby simplify the design process of a suitable classification algorithm. The class is the target variable that needs to be estimated by the classifier in the later prediction phase. With regard to the given problem the class represents a segment in the measured data in which the machine state is constant, that means the machine's components are in a specific constant state. Estimating the class of a given time segment within the measured test data means the state of any of the machine components is being estimated by the classifier. If for instance class 1 as defined in table 4.3 is being estimated by the classifier one can assume the control cabinet heat exchanger to be the only machine component being switched on while any other component is switched off.

Table 4.3 contains a scheme showing how the machine states or classes are defined. The full definition of classes can be found in appendix A.1. In total there are 96 different machine states that are theoretically possible, because there are 11 machine components which can be controlled directly via the machine's controller and three machine components that work self-sufficient and therefore are not directly controllable. This gives $2^3 = 8$ possible combinations of states for the non-controllable components and 12 possible switching states for the 11 controllable components, where the first state means that none of the controllable components is switched on. In any subsequent states one of the controllable components is switched on. For each of the 12 states of the controllable components every eight possible states of the non-controllable components need to be taken into account hence giving a total number of $12 \cdot 8 = 96$ different machine states or classses. If any of the 15 controllable and 7 non-controllable components was considered, this would give $16 \cdot 2^7 = 2048$ different states, which would lead to a much more

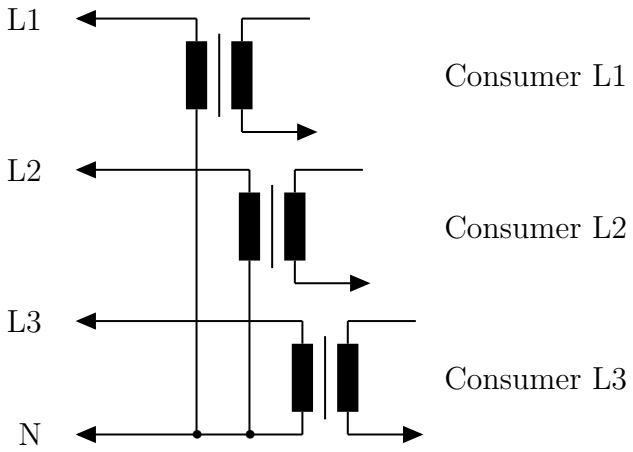


Figure 4.4: Current sensor wiring diagram after [64]. The current on the three consumer phases is measured with current transformers. Those are connected to the three phases and neutral conductor inputs of the measurement terminal.

complex clasification task.

As can be seen in table 4.3 as well, there are only two possible switching states for the machine lubrication pump, the oil-air lubrication pump and the control cabinet heat exchanger, where 0 stands for the component being switched off and 1 for the component running. This is sufficient as these components are either on or off. The situation is different with the main spindle and the five different axis drives, because these components have continuous states, for example the main spindle can run with any speed within the range of $0\,\mathrm{min}^{-1}$ to $18000\,\mathrm{min}^{-1}$. This would result in an infinite number of possible machine states and would therefore be a regression problem rather than a classsification problem. In order to reduce the number of possible states, the main spindle's rotational speed as well as the feed rates of the X-, Y-, Z-, B- and C-axis are being discretized to two respectively three different states. For the main spindle state 0 means the spindle is stopped, whereas state 1 means the main spindle is rotating with a speed of $10000\,\mathrm{min}^{-1}$. Any of the five machine axes has three possible switching states. In order to represent these three states with binary input variables a technique called *one-hot-encoding* is being utilized. Instead of allowing the categorial input variables 0, 1 and 2 the component's switching state is described with two binary input variables. Therefore any of the five machine axes in table 4.3 has two corresponding columns representing the axis moving in either positive or negative direction. If the input variables in both columns are 0 the corresponding axis is not moving. If the value in the *forward*-column is 1 this means, that the axis is moving at a constant feed rate of 12000 mm/min in the positive direction. Accordingly a 1 in the *backward*-column of the axis stands for a motion with the same speed in the negative direction. A 1 in both columns at the same times does not occur as it is obvious that the axis can not move forward and backward simultaneously.

Table 4.3: Definition of possible machine states. Every machine state is a specific combination of operating states of the different machine components and can be seen as individual class.

Controllable											**Non-Controllable**			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	1	0	0	0	0	0	0	0	0	0	0	0	3
0	0	0	1	0	0	0	0	0	0	0	0	0	0	4
0	0	0	0	1	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	1	0	0	0	0	0	0	0	0	6
0	0	0	0	0	0	1	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	1	0	0	0	0	0	0	8
0	0	0	0	0	0	0	0	1	0	0	0	0	0	9
0	0	0	0	0	0	0	0	0	1	0	0	0	0	10
0	0	0	0	0	0	0	0	0	0	1	0	0	0	11
0	0	0	0	0	0	0	0	0	0	0	0	0	1	12
1	0	0	0	0	0	0	0	0	0	0	0	0	1	13
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	0	0	1	23
0	0	0	0	0	0	0	0	0	0	0	0	1	0	24
1	0	0	0	0	0	0	0	0	0	0	0	1	0	25
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	0	1	0	35
0	0	0	0	0	0	0	0	0	0	0	0	1	1	36
1	0	0	0	0	0	0	0	0	0	0	0	1	1	37
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	0	1	1	47
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 4.3 (continuation) : Definition of possible machine states. Every machine state is a specific combination of operating states of the different machine components and can be seen as individual class.

Controllable											**Non-Controllable**			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	0	1	0	0	48
1	0	0	0	0	0	0	0	0	0	0	1	0	0	49
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	1	0	0	59
0	0	0	0	0	0	0	0	0	0	0	1	0	1	60
1	0	0	0	0	0	0	0	0	0	0	1	0	1	61
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	1	0	1	71
0	0	0	0	0	0	0	0	0	0	0	1	1	0	72
1	0	0	0	0	0	0	0	0	0	0	1	1	0	73
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	1	1	0	83
0	0	0	0	0	0	0	0	0	0	0	1	1	1	84
1	0	0	0	0	0	0	0	0	0	0	1	1	1	85
⋮	⋱	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
0	0	0	0	0	0	0	0	0	0	1	1	1	1	95

4.4 Test Cycle Definition

In order to create a dataset of sufficient size for training of a classifier, the machine needs to take on every machine state defined in section 4.3 several times and the corresponding current flow on the three phases of the main power line needs to be measured. For achieving a high predicition accuracy of the classifier, the training set should contain a specific amount of samples, that means every state should be taken on multiple times. How many samples are needed is not known at this point of the analysis and could only be examined in the later classifier design and validation with the so called *learning curve* which plots the classification score against the

number of samples in the training set. Because a classifier is not available at this point, the number of samples needs to be guessed and is therefore chosen to be a very high number so that the training set is very unlikely to contain too less samples for designing a highly performant classifier. Another reason for a very high number of training samples is the fact that the machine can not be put in any of the states deliberately because of the non-controllable component states of the lubrication pumps and the control cabinet heat exchanger. As every state is taken on only with a specific propability, it is necessary to have a long measurement time to ensure that as many of the different machine states is being observed within the training set. Apart from choosing a long measurement time for one test cycle, the test cycle can be repeated multiple times in the later execution phase to gather even more training data as well. In this case the test cycle is repeated three times. Two test cycles are used to get trainings data and one test cycle is being used in the later validation phase for evaluation of the classifier performance. Because the number of samples is only an assumption it needs to be validated in the later course of the project.

To put the machine in as many different states as possible a special program, the *test cycle*, is being written and executed on the machine controller. The overall program structure is being shown in fig. 4.5 and the corresponding subfunctions can be found in fig. 4.6. The flow chart is only an abstraction of the real machine code, which is being listed in appendix A.2. At the beginning of the program the setup code illustrated in fig. 4.6a is executed once. Within this setup procedure the block form of the workpiece is being set to zero as there is no workpiece to be machined. After this the tool is changed so a specific tool is being used in the test cycle. In this case tool number 70 is being used. After the tool has been successfully changed and is clamped into the main spindle all axes are moved to their zero positions. This ensures consistent test conditions as the axes will always start from the same position, which is especially important if the axes are not being located in their zero positions before execution of the test cycle. Otherwise the first axis motions could be different from any proceeding movements and therefore lead to inconsistent measurement results.

The rotational speed of the main spindle is being set to $10000\,\mathrm{min}^{-1}$ and the feed rate to a constant value of 12000 mm/min. These are large values that lead to high current consumption of the drives and therefore ensure clearly distinguishable patterns within the current signal on the DMU main power line. In the following program sentence the feed override is disabled with the M47-command to ensure constant feed rates during the execution of the test cycle. Prior to execution of the program the machine zero needs to be set to $\mathrm{X} = +554.042\,\mathrm{mm}$, $\mathrm{Y} = -280.622\,\mathrm{mm}$, $\mathrm{Z} = +0\,\mathrm{mm}$, $\mathrm{B} = +0°$ and $\mathrm{C} = +0°$. This machine zero is being stored in the custom machine zeros table in the machine controller as point number 35. The user has to manually set the machine zero to this specific point to ensure proper execution of the test cycle. Besides that the oparator needs to ensure that no clamping device is bolted to the machine's table in order to gurantee constant testing conditions.

After execution of the setup code the program enters a loop in which the main spindle as well as any of the five machine axes is being moved several times. First action is the repeated wind up and stopping of the main spindle as depicted in fig. 4.6b. When the programme enters this subfunction the main spindle is wound up by issuing the M03-command after a short delay of 2 s. After reaching the desired speed the spindle is being run for 20 s and afterwards stopped

by a M05-command. This procedure is repeated five times. Hereinafter the machine axes are moved following the scheme that can be seen in fig. 4.6c. After a 2 s-delay the relevant machine axis is moved forward with a constant feed rate of 12000 mm/s. When the desired position is reached the machine pauses for 2 s and then moves back the axis to the machine zero with the same speed. This procedure is repeated five times for each of the five machine axes. The loop itself is repeated 20 times, so that each movement is repeated 100 times in total. This creates the necessary high number of training samples within the training set. After completion of the loop the feed override is switched back on via the M48-command and the program terminated by an M30-command.

Table 4.4 shows the desired positions and travels for the axis movements within the test cycle. End point and zero point coordinates refer to the machine zero set before first execution of the test cycle. Z-1 M91 in the zero point of the Z-axis means the Z-axis is moved downward until it reaches the end stop switch. This ensures maximum distance between the tool and table for following safe rotation of the B-axis. This distance and therfore the travel of the Z-axis depent on the length of the tool used for the test cycle. The desired zero and end points of the axes are chosen so that the travels are maximized. This helps determining wear of the axes' bearings or guidances as well as the ball screws of the translational axes. If the axis moved only in a small range of the maximum travel such failures would hardly be detected.

Table 4.4: Desired positions and travels of the machine axes movements in the test cycle. Zero point and end point coordinates refer to the machine zero.

Axis	Machine Zero	Zero Point	End Point	Travel
X-Axis	554.042 mm	0.000 mm	900.000 mm	900.000 mm
Y-Axis	−280.622 mm	0.000 mm	−600.000 mm	600.000 mm
Z-Axis	0.000 mm	~ 647.763 mm (Z-1 M91)	250.000 mm	~ 397.763 mm (tool dependent)
B-Axis	0.000°	0.000°	−120.000°	120.000°
C-Axis	0.000°	0.000°	180.000°	180.000°

Figure 4.5 also contains procedures for controlling the switching states of the coolant pumps 1 and 2, the chip conveyor drive motor and the fume seperator motor. The corresponding program code for the Heidenhain iTNC 530 can be found in listing A.2. These enhancement of the testcylce needs to be understood as a suggestion for further improvement of the classification task and are not taken into account for the reasons stated in section 4.1.

4.5 Test Cycle Results

After its successful definition and implementation in section 4.4 the test cycle is being executed and the corresponding measurement results are being examined in this section. As mentioned before three test cycle runs are conducted to gather a great amount of training data and an independend test dataset for later evaluation of classifier performance. Table 4.5 contains information about the time of the test cycle execution for each of the three runs. This information

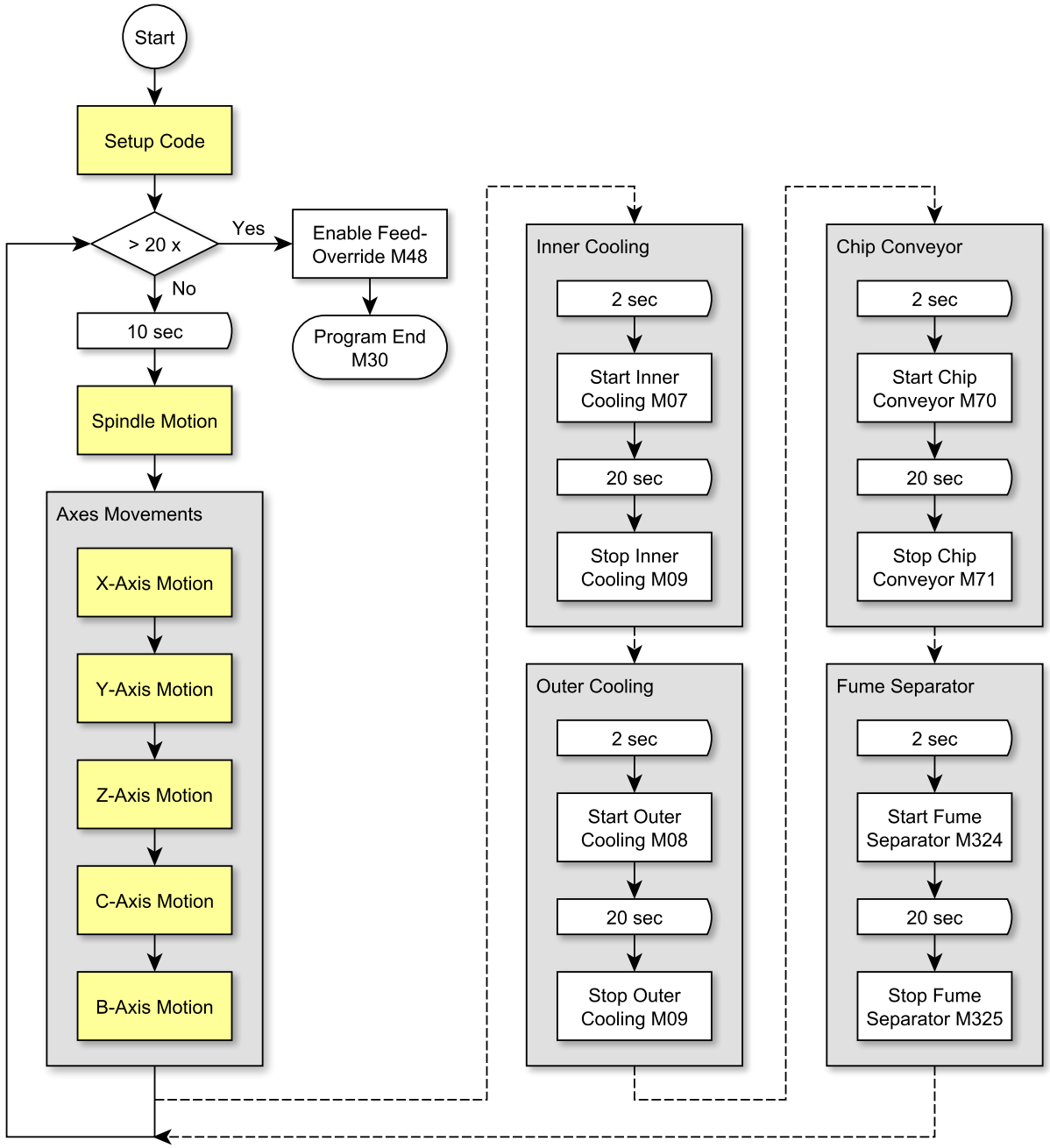
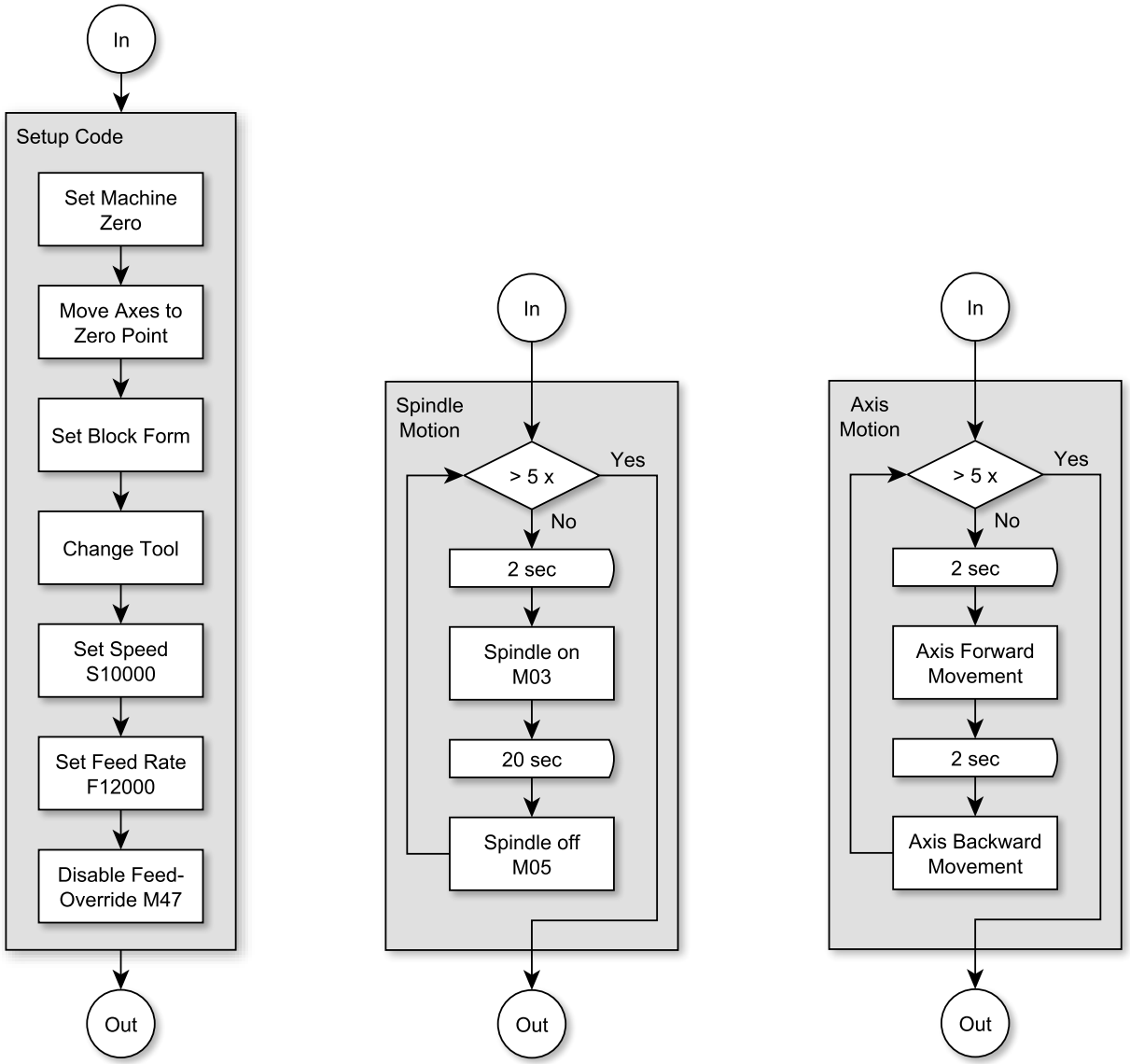


Figure 4.5: Flow chart of the program structure of the test cycle. The test cycle is designed to let the milling machine take on the previously defined states that are used for model training as well as recognition of any machine component failures or wear. Any motions and component activations are repeated 20 times in order to create more data samples for model training. Cooling, chip conveyor and fume separator are not implemented in the test cycle and might be considered in future studies. Yellowish colored boxes are subfunctions containing multiple commands as defined in fig. 4.6.



(a) Subfunction for prior setting up of the machine.

(b) Subfunction for wind up and stopping of the main spindle drive.

(c) Subfunction for movement of any of the five machine axis.

Figure 4.6: Subfunctions for machine setup as well as spindle and axis motions within the above flow chart of the test cycle. Both motion sequences are repeated five times in order to create a bigger amount of data samples.

is needed for read out of data from the database. In the following section a detailed description of the measurement data of the third test cycle run is given. Corresponding results for the 1st and 2nd test cycle runs can be found in appendix A.3. They are not explicitly examined in this section, because the resulting signals look very similar to the depicted ones of the third test cycle run. All plots are created using a MATLAB plotting script which consists of a main script shown in listing A.5 and a subroutine for reading CSV files containing the measurement data which is shown in listing A.6.

One complete test cycle run takes 7200 s for completion resulting in a duration of approximately 360 s for every of the 20 subcycles, in which the five machine axes are moved back and forth five times. This can be seen in fig. 4.7, which depicts the current consumption of the machine on the three phases of the the main power line and shows a very specific regular pattern corresponding to the different motions of the axes. This high correlation between the axis movement and the current consumption on the main power line can be seen even better in the close-up in fig. 4.8 that contains the current flow on phase L2 for only the first of the 20 subcycles. There any axis movements are clearly distinguishable. First the five consecutive spindle wind ups and stops can be perceived, revealing a highly dynamic behaviour of the spindle as the spindle drive consumes over 55 A on wind up and more than 30 A for stopping. In between of the wind up and stopping the current consumption appears to be nearly constant. This can be seen in the detail view of one single wind up and stopping of the spindle in fig. 4.12a. After the spindle movements follow the back and forth movements of the X-, Y- and Z-axis as well as the clockwise and anti-clockwise rotations of the B- and C-axis. Figure 4.12 contains detailed views of the resulting patterns in the current signal in the DMU main power line L2-phase respectively for the forward movement and the backward movement, which differ especially on the Z-axis where gravity has to be overcome on the upward movement. The same applies for the B-axis, which rotates in the vertical plane and therefore is exposed to gravitational pull as well. In contrast the backward and forward movements of the X-, Y- and C-axis do not show this behavior as motion takes place perpendicular to ground.

Table 4.5: Location of the three conducted test cycle runs within the measurement database.

Run	**Start**			**End**			**Type**
	Date	Time	Timestamp	Date	Time	Timestamp	
1	03/05/2017	20:06:00 CEST	1493834880	03/05/2017	21:30:00 CEST	1493842080	Testset
2	03/05/2017	22:43:00 CEST	1493844200	04/05/2017	00:43:00 CEST	1493851400	Trainingset
3	04/05/2017	11:47:00 CEST	1493891290	04/05/2017	13:00:47 CEST	1493898490	Trainingset

Apart from the current signal on the main power line and according to section 4.1, the current consumption of the cooling unit, the control cabinet heat exchanger, the machine lubrication pump and the oil-air lubrication pump have to be taken into account on creation of the dataset as they determine, which of the states defined in section 4.3 is present at a given time segment within the measurement results. This knowledge is essential for later labelling of the acquired datatset. Therefore the current consumptions of these components are being measured as well. The measurement results are shown in fig. 4.11a, fig. 4.11b and fig. 4.11c.

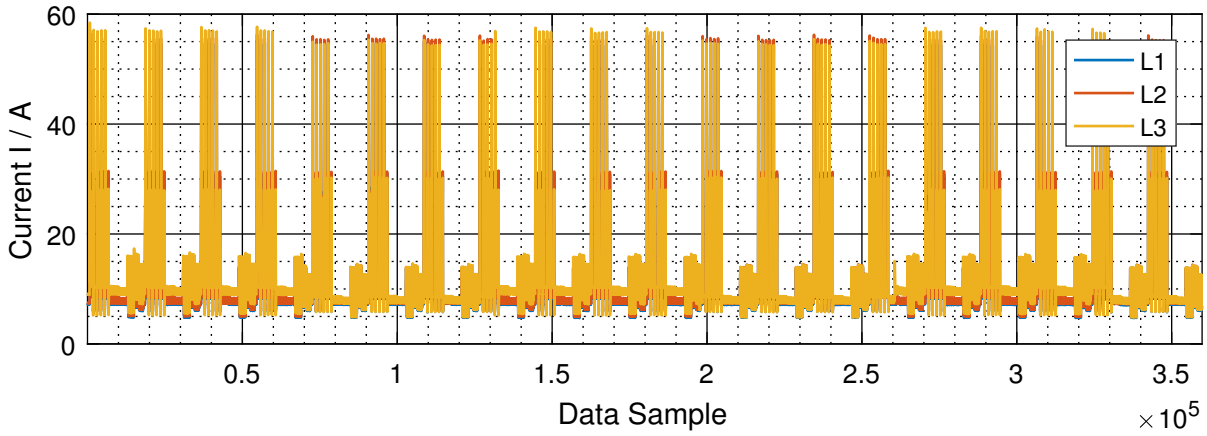


Figure 4.7: Measured signal of the current flow within the three phases of the DMU main power line during the complete third test cycle run.

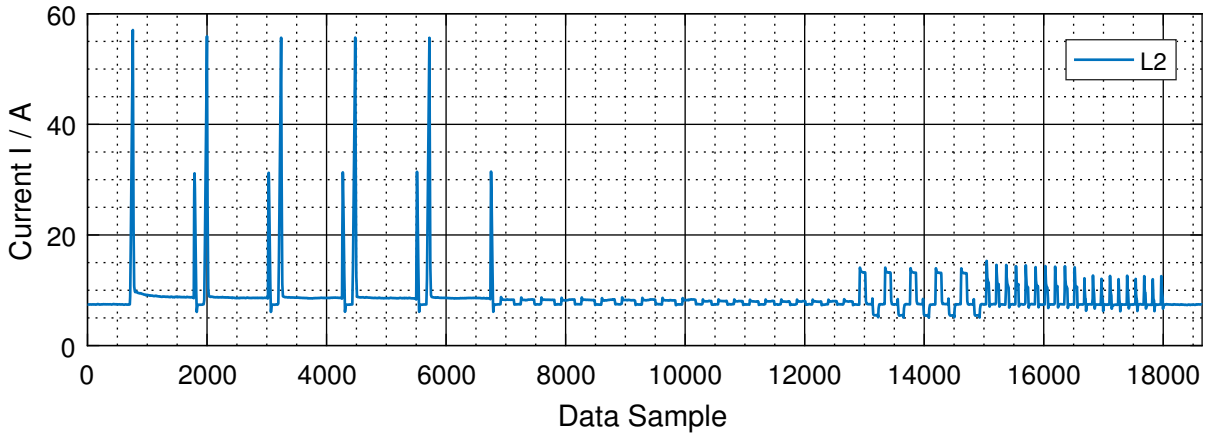


Figure 4.8: Close-up of the measured current in the second phase (L2) of the DMU main power line during a single iteration of the third test cycle. Motions of different axes are clearly distinguishable.

As said in section 4.1 the current consumption of the cooling unit appears to be nearly constant over time neglecting slight fluctuations. Noteworthy is the difference of the current consumptions on the three different phases. As can be seen the current on the L1-phase lies around a value of 4.33 A, the current consumption on the L2-phase is around 4.21 A and on the L3-phase around 4.47 A. Apparently this is due to the internal wiring of the contained compressor, cooling fan and feed pump (see table 3.2). The difference in the current consumptions on the three phases can be seen in the main power line signal, where the current on phase L3 is on average higher than on phases L1 and L2. In turn, the current consumption on L1 is higher than the one on L2.

At this point attentention should be paid to the nomenclature of the phases, because the phase names are being chosen from the measuring terminals viewpoint rather than accoring to internal wiring of the machine. So phase L1 means the current sensor for a specific component power line phase is being connected to the L1-input on the terminal. This does not imply the measured component phase to be connected to the L1-phase of the machine main power line. This internal

connection of the considered non-controllable components to the three power phases on the machine power line can be found in fig. 4.9. As can be determined from the measurement data in fig. 4.7 none of the non-controllable components is connected to phase L2 of the power line. This becomes clear because there is no correlation between the curent flow on phase L2 and the current consumption of any of the non-controllable components. Opposite to that the current flow on phases L1 and L3 changes in a particular manner whenever one of the non-controllable components is swichted on or off. From this can be learned that the control cabinet heat exchanger is connected to phase L3 of the main power line and the machine as well as the oil-air lubrication pump are connected to phase L1 of the main power line. This allows for a simplified approach on solving the classification problem as under consideration of only phase L2 the influence of the non-controllable components can be neglected completely. Neglection of these components reduces the number of different machine states as defined in section 4.3 to only 12 rather than 96. Use of this will be made in the first part of the following classification task. In a second step the current signal on phases L1 and L3 of the main power line and thus all non-controllable components are considered as well.

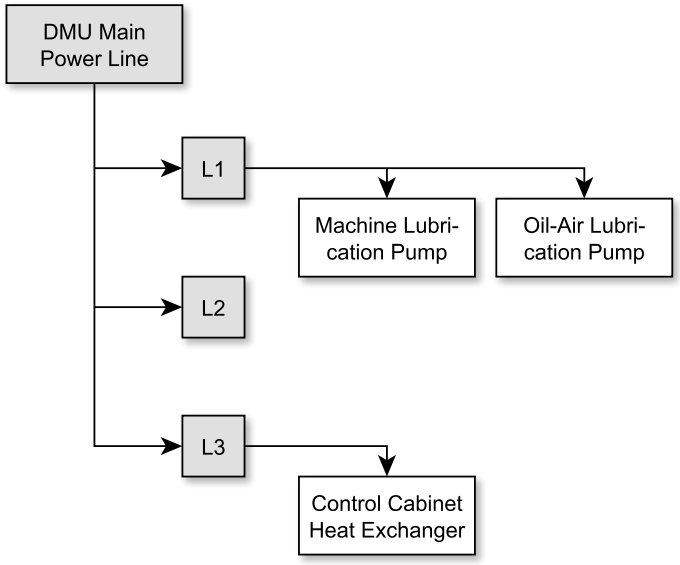
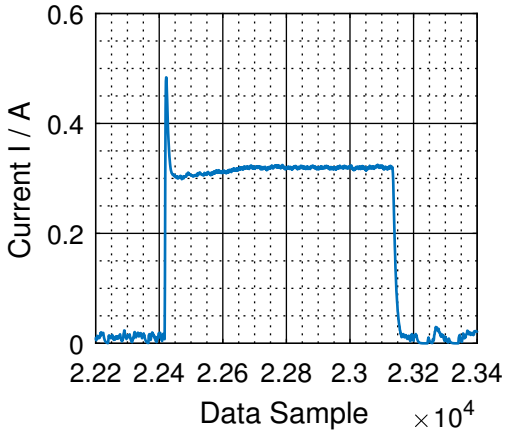


Figure 4.9: Internal connections of the three considered non-controllable componentes to the three phases of the machine main power line.

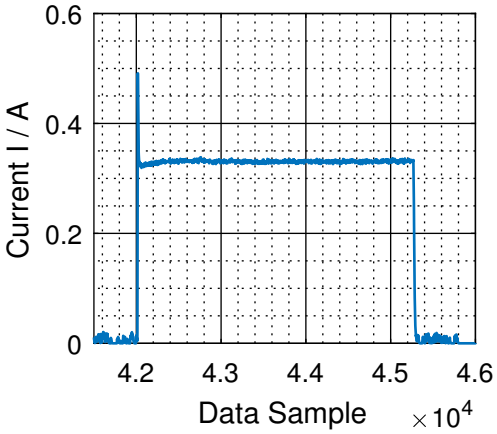
Proceeding with the analysis of the control cabinet heat exchanger's current signal depicted in fig. 4.11b a very different behviour from that of the cooling unit can be seen. Rather than being on an approximately constant level the current consumption of this component switches between to distinct values of 0.32 A and 2.80 A, whereby a current peak occurs on the rising edge of the signal due to a slow dynamic behaviour of the contained compressor. The lower value refers to the control cabinet heat exchanger idling with just the cooling fan being switched on, whereas the higher current value means the contained compressor is being turned on in addition to the cooling fan. The effect of the control cabinet heat exchanger on the current signal of the main power line is again clearly visble as the current on phase L3 of the main power line is increasing accordingly. This makes clear the control cabinet heat exchanger as a single-phase consumer is

internally connected to the L3-phase of the main power line of the machine.

At last the current signal of the three lubrication pumps shown in fig. 4.11c is considered. As said before the blower air lubrication pump is just depicted for the sake of completeness and not being activated resulting in no current consumption on the measuring-input of the terminal. In contrast the oil-air lubrication pump and the machine lubrication pump are being switched on from time to time for a specific duration. Moreover the switching interval for the oil-air lubrication pump is regular. For instance the oil-air lubrication pump is being activated approximately every 434 s for a duration of 14.5 s consuming around 0.32 A. The machine lubrication pump is being switched on in a non-regular pattern for a period of 65.5 s consuming around 0.33 A. The activations occur in intervals ranging from 1380 s to 1500 s. This irregularity in the activation intervals of the machine lubrication pump can be explained by the fact, that activation does not occur based on time, but rather than that based on other factors, for example traveled axis range. This becomes clear, when the machine is operated with another program instead of the test cycle and the activation intervals change completely and become even more irregular. This does not affect the oil-air lubrication pump, which seems to be actived in a time based manner. Close-ups of the current signal of an individual activation can be found in fig. 4.10a for the oil-air lubrication pump and in fig. 4.10b for the machine lubrication pump. As can be seen from those signals, the lubrication pumps have a slow dynamic behaviour as well, which leads to a spike on the rising edge of the current signal. After the current has settled it stays on a constant level until deactivation of the pump.

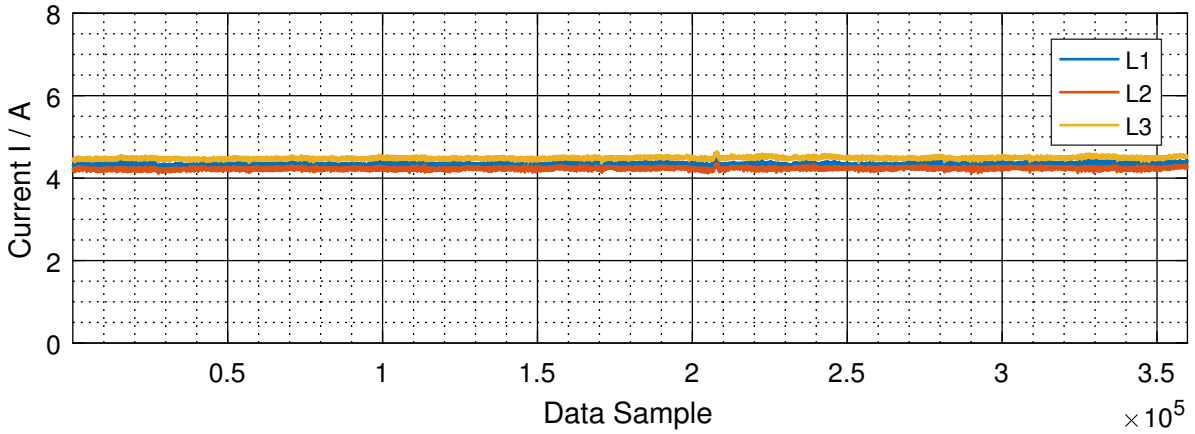


(a) Current signal of an individual activation of the oil-air lubrication pump.

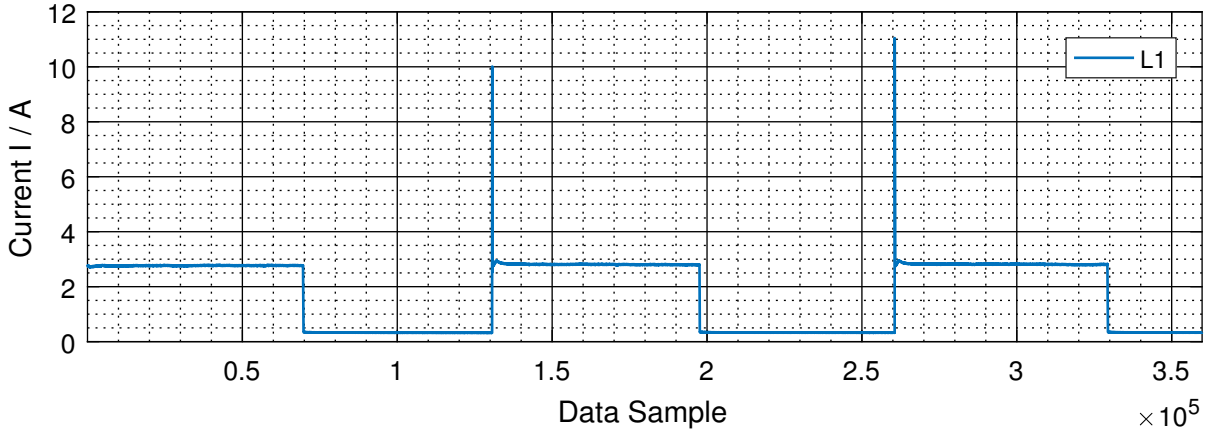


(b) Current signal of an individual activation of the machine lubrication pump.

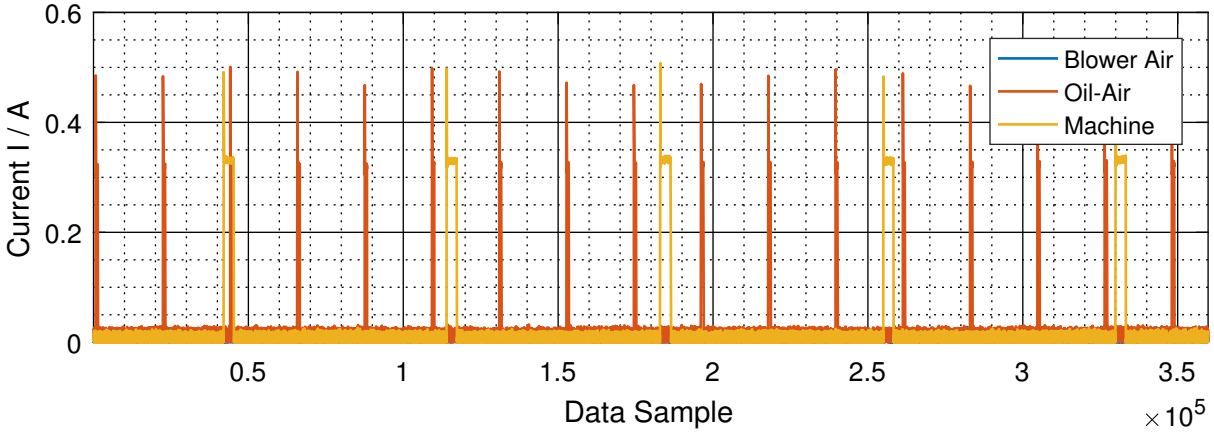
Figure 4.10: Current signals of individual activations of the oil-air and machine lubrication pump in the third test cycle run.



(a) Current consumption of the cooling unit.

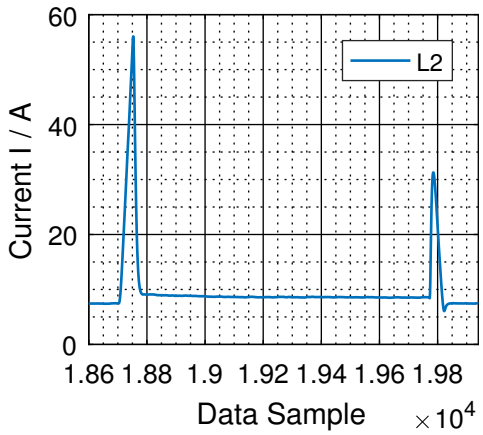


(b) Current consumption of the control cabinet heat exchanger.

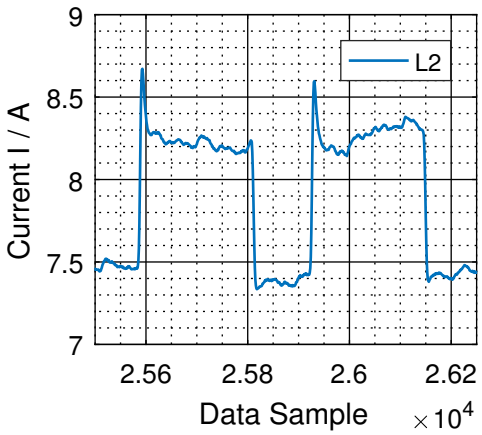


(c) Current consumption of the blower air lubraction pump (L1), the oil-air lubrication pump (L2) and the machine lubrication pump (L3).

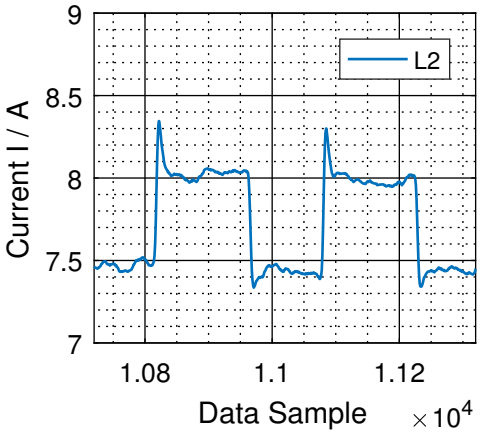
Figure 4.11: Measured current consumption of the different machine components during the third test cycle run.



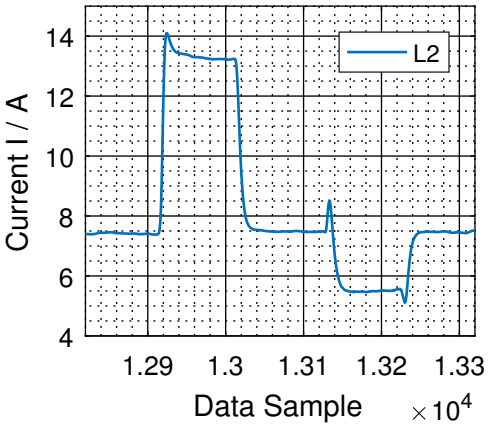
(a) Pattern of spindle runup and stopping.



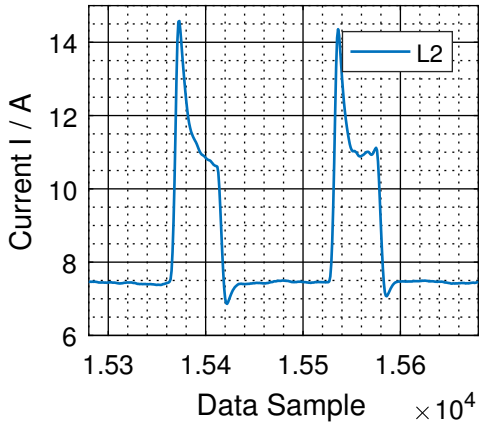
(b) Pattern of X-axis moving back and forth.



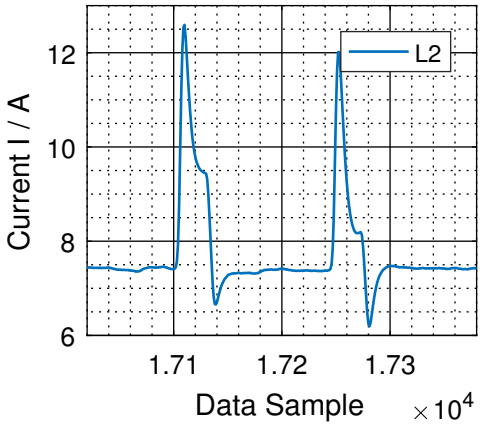
(c) Pattern of Y-axis moving back and forth.



(d) Pattern of Z-axis moving up and down.



(e) Pattern of C-axis turning back and forth.



(f) Pattern of B-axis turning back and forth.

Figure 4.12: Different patterns occuring in the current signal on the first phase of the DMU main power line according to different axes moving either forward or backward.

5 Data Segmentation and Labelling

Purpose of this chapter is the implementation of two different approaches for segmentation and labelling of the previously acquired measurement data. As the measruement dataset only exist in form of timeseries data preprocessing has to be conducted to transform the timeseries data into a set of discrete data samples. Therefore it is necessary to break down measured timeseries data into smaller segments, which are later used as individual samples within the classification task. Moreover it is necessary to assign one of the previously defined class labels to each segment to create a training dataset for later training of classifiers. In this section neither the discretization nor the classifier training takes place, but only an output file will be created, that contains information about the segmentation and classes of each segment of the measurement data. The first section in this chapter gives a brief introdution to the task of segmenting and labelling the measuremend data. Afterwards a manual approach for fulfilling this task is being presented in detail. Here a software with graphical user interface, which assists the user in segmenting an labelling of the measurement data is designed. In the subsequent section a fully automated approach utilizing a template matching algorithm for segmentation is developed. Labelling is split up into two parts of different complexity in this section. Finally results of the automated segmentation and labelling algorithm will be presented.

5.1 Segmentation and Labelling Task Outline

An essential step in generating the dataset for the present machine learning problem is the preparation of the time series measurement data for later processing like feature extraction or template generation. This includes cutting of the complete 7200 s long traing measurement into individual segments or samples as illustrated in fig. 5.1 and table 5.1. Each cut is placed on a state transition of the measurement signal, so that it separates two different machine states as defined in section 4.3. The first cut is placed on the first data value of the measurement data so the corresponding segment or sample lies to the right of each cut. For instance segment 5 lies to right of cut 5, whose position is described by the *cut index* in table 5.1. For cut 5 the cut index equals 60, meaning that the cut lies on the 60th data point of the measurement data. Each segment ends at the data point prior to the next cut. In this case cut 6 at data point 80, which gives the upper boundary for segment 5 as data point 79. This definition ensures nonrecurring inclusion of each data point within the measurement data. Important to notice is that for later analysis a cut at the end of the signal should be inserted. This restriction results from the specific implementation of the feature extraction and template creation algorithms. Otherwise the segment to the right of the last cut, like cut 7 in the schematic, is not considered in the analysis. To include this segment as well, another cut had to be placed at the last point of the measurement data. Besides the description of the sample boundaries by the cut index, another value enables distiguishing between nominal cuts and non-nominal cuts. A nominal

cut, like cut 5, marks a transition between controlled machine states, for example a transition between the idle state and the movement of any of the axis or the main spindle. In contrast to that a non-nominal cut, like cut 3 and 7, marks a transition in the measurement signal, which corresponds to activation or deactivation of one of the three non-controllable machine components, namely the machine lubrication pump, the oil-air lubriaction pump and the control cabinet heat exchanger. The nominality of each cut is represented by a boolean value, where a nominal cut is described by the value 1 and a non-nominal cut by the value 0. After cutting the segments each segment is labelled with the corresponding class, which is retrieved by a table-lookup of the machine state definitions in table 4.3. This allows for determination of the class of each segment by knowlegde of the components being active in that particular segment. For instance in segment 5 in the schematic in fig. 5.1 the x-axis is being moved backward with the control cabinet heat exchanger being active at the same time. As can be learned from table 4.3 this corresponds to machine state 15, wherefore segment 5 is labeled with class 15. This form of labelling allows for a very easy description of each individual segment by only three integer numbers, which can be efficiently stored in a CSV-file. Furthermore this allows for high portability and easy to implement later processing like feature calculation or template creation.

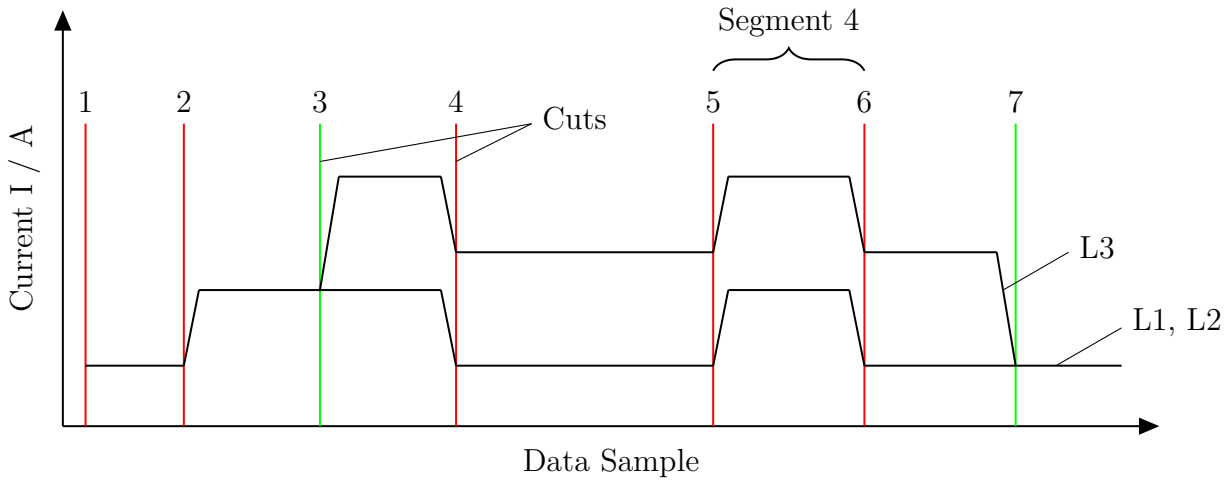


Figure 5.1: Desired outcome of the segmentation and labeling process. Two different kinds of cuts delimiting segments within the measurement signal shall be placed based and each segment shall be labeled with its corresponding class. It needs to be distinguished between nominal cuts (red) and non-nominal cuts (green). Result of the segmentation and labelling process is a CSV file containing index and nominality of each cut and the class of each corresponding segment to the right of the cut.

In the following two different approaches of creating the labeled dataset are taken. The first one is a manual approach, in which the user has to manually place every individual cut and determine the class of each segment semi-automatically by table-lookup of the class based on manual description of the components actived in the current segment. The second one is a fully automated approach, which determines positions of cuts by looking at the characteristics of the measurement signal and additional measurements of the three non-controllable components. Classes for the segments are determined automatically as well based on the periodicity of the

Table 5.1: Example for the segmentation and labeling of the measurement data section in fig. 5.1.

Cut No.	Cut Index	Nominality	Class
1	1	1	0
2	10	1	2
3	25	0	14
4	30	1	12
5	60	1	15
6	80	1	12
7	95	0	0

test cycle and the information of the switching states of the non-controllable components. The two following sections describe both methods in detail.

5.2 Manual Segmentation and Labelling

The first approach for segmentation of the timeseries data is the manual approach. Labeling the data manually is a very easy approach in preprocessing the timeseries data for later analysis. It allows for good precision on the cutting process and exact labelling of all classes. Most important disadvantage of this approach is the long time needed for labelling the whole training data set. Moreover each time, a new dataset is retrieved, the labelling process has to be repeated manually, which is very time consuming. Nevertheless this section addresses the manual labelling of the dataset as it is useful for getting an subjective impression of the structure of the dataset and the contained classes. To reduce the time needed for labelling the dataset, a software with a graphical user interface is built which assists in cutting and labeling of the dataset. Figure 5.2 shows the main functions of this label assistant software, whose source code can be found in listing A.3. As can be seen, first the previously acquired and locally stored datasets are loaded and visualized as timeseries on different plots. This helps the user to pick the locations of the cuts with a specific data cursor tool on the plots. Moreover the user has to put the switching state of every controllable and non-controllable component of the machine for the current time segment in the software. This allows for automatic determination of the class for the current sample via table-lookup. Moreover the user needs to select for each cut, if it is a nominal cut or a non-nominal cut. Input of these values allows for creation of the previously mentioned CSV-file that contains the labelling information for the given dataset.

Figure 5.3 shows the graphical user interface of the label assistant and explains the main modules within the interface. Important part of the GUI is the data visualization in the three plots to the left. On top the main power line currents for all three phases are depicted. In the middle the signal of the control cabinet heat exchanger can be found an on the bottom the signal of the three lubrication pumps is shown. On top of the plots the toolbar menu can be found. As can be seen in the close-up in fig. 5.4 it contains tools for opening an existing labelling file, saving a created labelling file, zooming and panning within the plots as well as the data cursor tool,

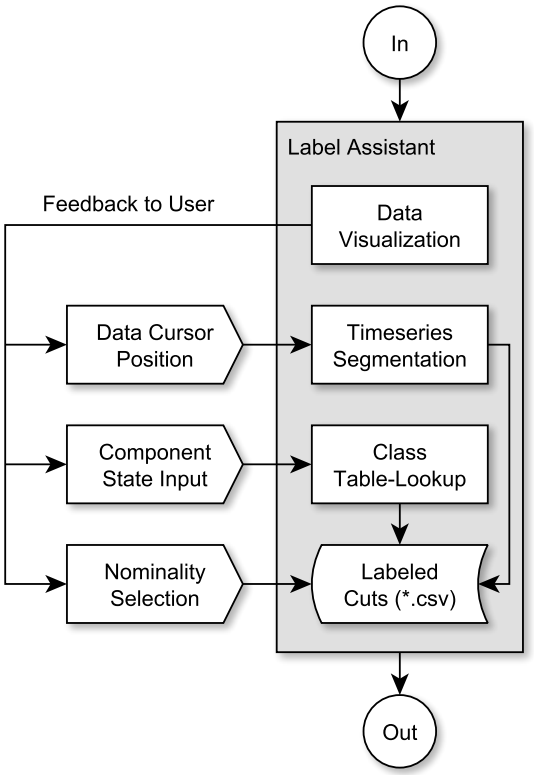


Figure 5.2: Process steps of the manual timeseries segmentation and labeling with the label assistant GUI.

which is used to locate cut positions. On the right side of the GUI window widgets for input of component states and cut nominality as well as an output of the current labelling file can be found. Moreover buttons for cut creation and manipulation of the current labelling file, like removal of lines within the labelling file or the complete file, are contained in this section of the GUI window.

Figure 5.5 shows a detailed view of the control panel to the right side of the label assistant GUI. The first three check boxes to the top allow for input of any combination of switching states of the three non-controllable components. Below a radio button group can be found, which is needed for input of the controlled state, like for example forward movement of the x-axis, spindle movement or machine idle (none). As there is only one controlled state per segment possible, this input is designed as radio button group, in which only one element can be 1 at a time, whereas all others have the value 0. Below the component state input a text box can be found which contains a hisotry of the lasst three selected controlled states. This helps for later labelling as the user easily can see what the last selected state was. Under consideration of the knowledge of the fixed sequence of controlled states in the test cycle the user can easily figure out, which controlled state is present in the current time segment.

In the middle of the control panel a checkbox for input of the current cut nominality (see section 5.1) can be found. This checkbox is reset to value 1 after each processed segment to prevent continous input of non-nominal cuts, which is usually not reasonable for the existing

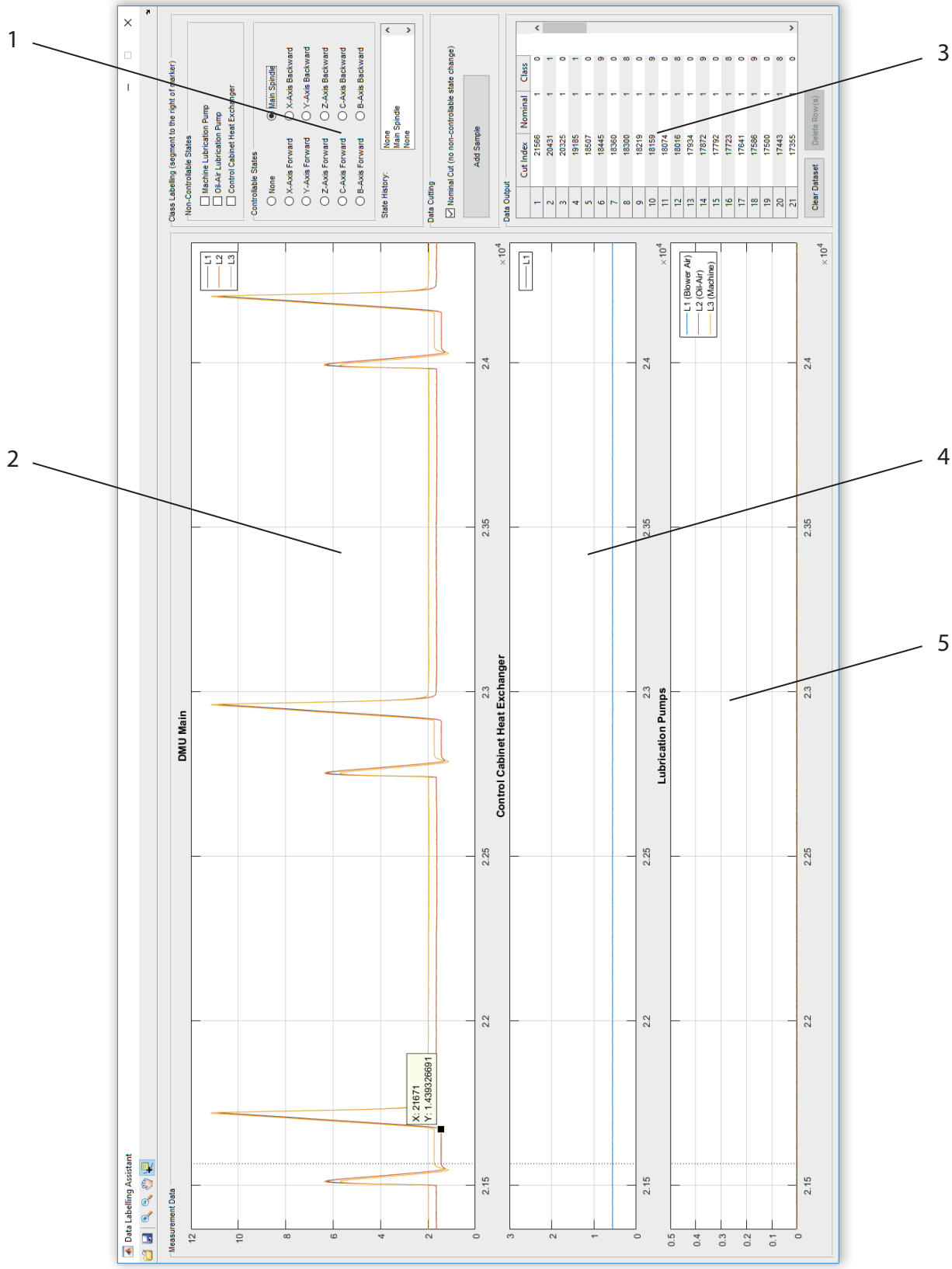


Figure 5.3: Graphical user interface of the data label assistant. (1) Machine State Selection, (2) DMU Main Power Line Current, (3) Segmentation Cuts and Data Labels, (4) Control Cabinet Heat Exchanger Current, (5) Lubrication Pumps Current.

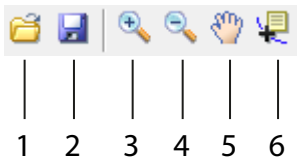


Figure 5.4: Toolbar icons in the data label assistant. (1) Open existing dataset from file, (2) Save current dataset to file, (3) zoom in tool for plots, (4) zoom out tool for plots, (5) pan tool for plots, (6) data tooltip for picking cut locations from plots.

measurement data as on every non-nominal cut a tmaximum on other non-nominal cut should follow. Furthermore a button for adding the current time segment to the labelling dataset can be found in this area of the GUI window. Each time the user has selected a data point at which a cut shall be located and a selection of component states has been made, the button is activated and adds a new line containing the data point index and nominaltiy of the cut as well as the class of the corresponding time segment to the labelling dataset.

On the bottom of the control panel a table view is located. This table view contains every line of the currently loaded and processed labelling dataset consisting of cut index, nominality and class. The view is sorted automatically, so that the highest cut indices are on the top. This allows for selection of any data point on the plot and automatic positioning of the corresponding label entry on the correct position within the labelling dataset. Below the table view two buttons can be found. The left deletes the complete labelling dataset, however it shows a prompt window asking for confiirmation first. The button to the right is activated whenever the user makes a selection of one or more rows in the table view. After clicking on the button and confirming the prompt the selected rows will be deleted from the labelling dataset.

To give a short introduction in how the label assistant is used the main steps of the labelling process are depicted in fig. 5.6. If a labelling dataset already exists, it can be loaded via clicking on the load button in the toolbar menu and browsing the corresponding CSV-file. This step is optional and may be skipped if there is no labelling dataset available and needs to be created from scratch. After this step the user has to select the data cursor tool from the toolbar menu. With activated data cursor tool the user clicks into any of the three measurement data plots on that data point where a cut needs to be made. If needed the cursor can be moved via clicking and dragging after it has been placed on the plot. Zooming and panning tools help to exactly position the data cursor. Once the data cursor is on the desired position, first the states of the three non-controllable components and afterwards the states of the controlled components need to be selected by checking the corresponding checkboxes and radiobuttons. Hereafter the nominality of the cut needs to be selected by checking or unchecking the checkbox. Now the cut can be placed by clicking on the add sample button. A new line contianing cut index, nominality and sgement class is added to the table view. Moreover a vertical dotted line is added to all three plots marking the saved cut position. This steps are repeated for every desired cut until the labelling of the complete dataset is finished. Finally the labelling dataset has to be saved to a CSV-file by clicking the save button in the toolbar menu. It can be openened and edited again at a later time.

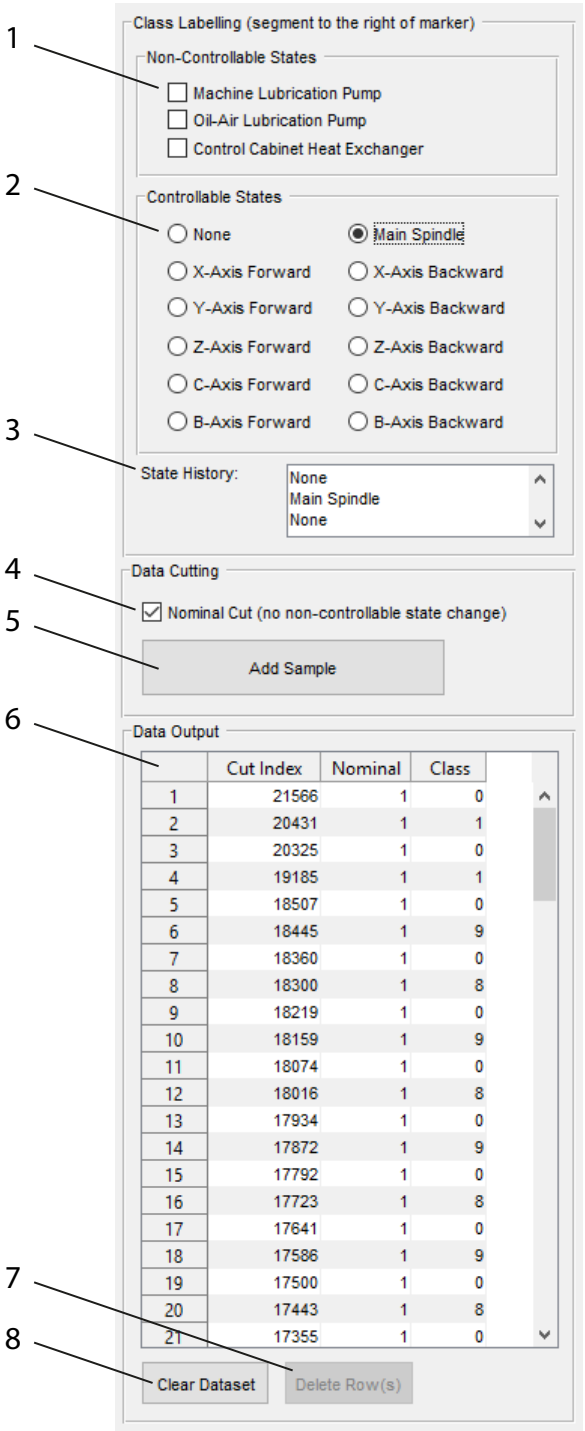


Figure 5.5: Control Panel of the label assistant GUI. (1) State selection of non-controllabe components, (2) State selection of controllable components, (3) History of latest three states, (4) Selection of nominal or non-nominal cut, (5) Button for cutting, (6) Labeled cuts as output data, (7) Button for deletion of selected rows, (8) Button for clearing complete dataset.

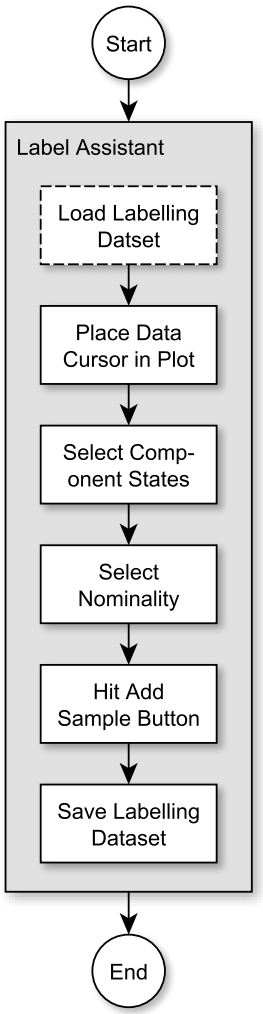


Figure 5.6: Workflow for creating a new segment labeled in the mesaurement dataset and adding it to the labeling dataset in the label assistant GUI.

5.3 Automatic Segmentation and Labelling

Manual labelling has the significant drawback of being very time consuming. Moreover for every new training dataset the complete procedure has to be repeated. Besides that, although the previously presented label assistant GUI significantly reduces the risks for human errors during the labelling process, the procedure is still not as reliable and consistent as an automatic segmentation and labelling approach. This is why this section addresses the problem of designing an fully automatic approach for segmentation and labelling of the given timeseries measurement data. First an algorithm is presented, which automatically segments and labels only the measurement signal of phase L2 of the machine main power line, which is not influenced by any non-controllable component. According to section 4.5 this simplifies labelling of the dataset as only 12 different classes occur in the data. Later in this section the algorithm will be extended to also automatically detect state changes of non-controllable components and label the created

segments accordingly with one of the 96 possible classes. Output of this algorithm will be two CSV files. Both contain according to fig. 5.1 the data sample indices at which a segment begins, the class of the segment and the nominality of the cut, which contains the information about the cut either corresponding to a state change of one of the controllable 12 states or a state change of one of the three non-controllable components. Difference between both output CSV files of the algorithm is that the first CSV file only contains nominal cuts and base classes 0 to 11, whereas the second CSV file contains all cuts and all 96 possible classes.

Automatic segmentation and labelling is conducted in a similar manner as the manual labelling within the GUI presented in the previous section and contains the same important process steps. First the measurement data needs to be segmented. That means at every state change a new segment is introduced, in which the machine state stays constant until the next component state change. Second the switching states of any of the considered machine components need to be retrieved. From this information the class according to section 4.3 can be determined for each segment. This last step is divided into two parts, whereby the algorithm first determines only the classes of the controllable components and considers the non-controllable components in a subsequent step.

An overall structure of the main functional steps within the automatic labelling algorithm can be found in the flowchart in fig. 5.7. This flowchart is divided into the described three main process steps of segmentation, basic labelling and labelling of non-controllable states as well. In the following a detailed explanantion of each of these three subprocesses is given. The Matlab source code of the complete algorithm can be found in listing A.7.

5.3.1 Segmentation

First the automatic segmentation is considered. As can be seen in fig. 5.7 a template matching approach is used to find out at which locations within the measurement data a state change occurs and a cut should be placed to introduce a new segment. In this case template matching basically means for each of the 12 possible kinds of patterns within the signal there is template that is being compared to the measurement signal at different locations. For each location a distant measure is calculated that quantifies the similarity between template and measurment signal. At minimum distance between template and measurement signal the template matches the measurement signal optimally meaning a pattern very similar to the template has been found in the measurement signal. The starting point of this subsequence in the measurement signal is then marked with a cut and a new segment is introduced at this data point.

Figure 5.8 shows the 12 templates for the different kinds of patterns in the current signal in the machine main power line. Each of these patterns refer to a specific controllable state (see section 4.3) of the milling machine, for example there is a specific template for the X-axis forward movement or the main spindle movement. Creation of these patterns is very simple as they are only extracted from the measurement data from the first of the 20 subcycles of the second test cycle run. Hereby the templates correspond respectively to the first occurance of the specific pattern in this subcycle. Calculation of average patterns from a larger set of patterns extracted

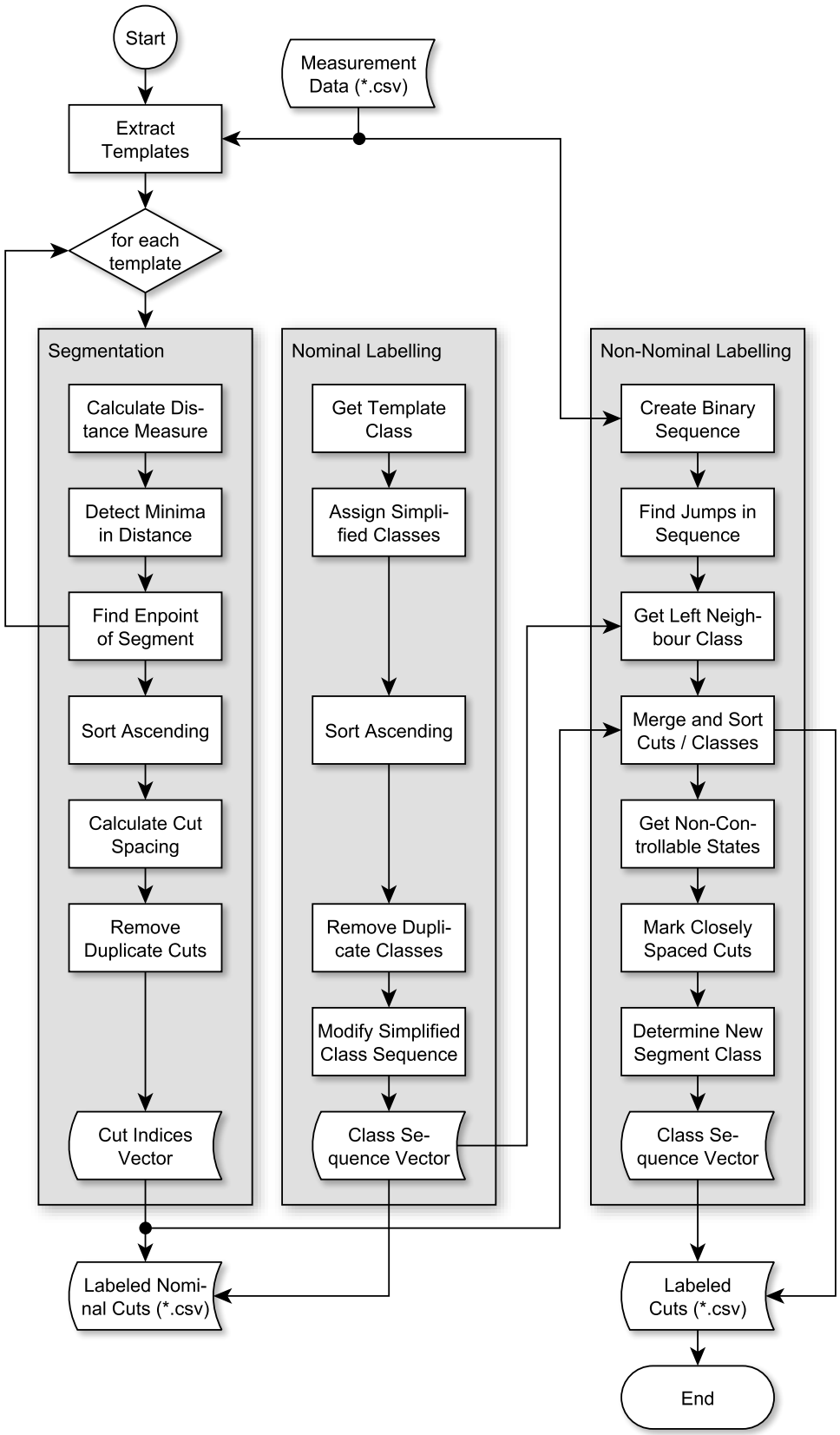


Figure 5.7: Schematic of the full automatic segmentation and labeling algorithm, which is based on a time domain shape-based similarity measurement approach.

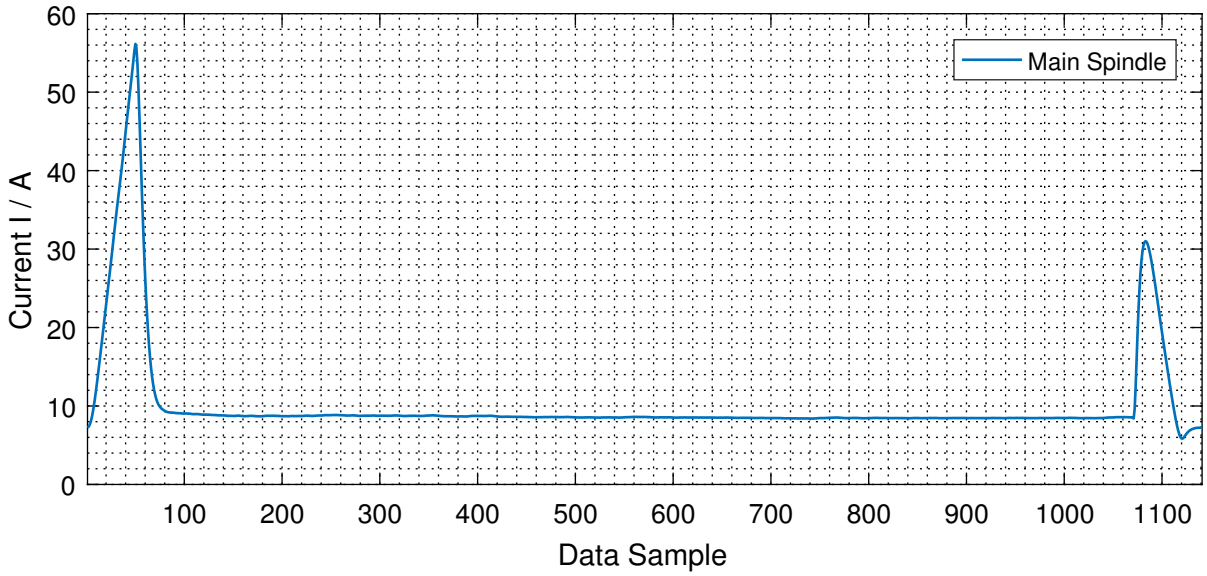
from the training measurement data is not necessary as the templates corresponding to extracted individual patterns already yield very good results of automatic segmentation.

The priciple of determination of the actual cut positions is shown in fig. 5.9. The brown line in this diagram represents a section of the measurement data from the second test cycle run corresponding to the first forward and backward movement of the X-axis in the second subcycle of the test cycle. The blue segment is the template for the X-axis forward movement from fig. 5.8b. Starting at the leftmost sample of the measurement data the algorithm iteratively shifts the template one data sample rightwards in each iteration of a loop. For each iteration a scalar distance measure is calculated which quantifies the similarity between template and measurement signal. From a set of different distance measures (see [40, 69]) the infinitiy norm

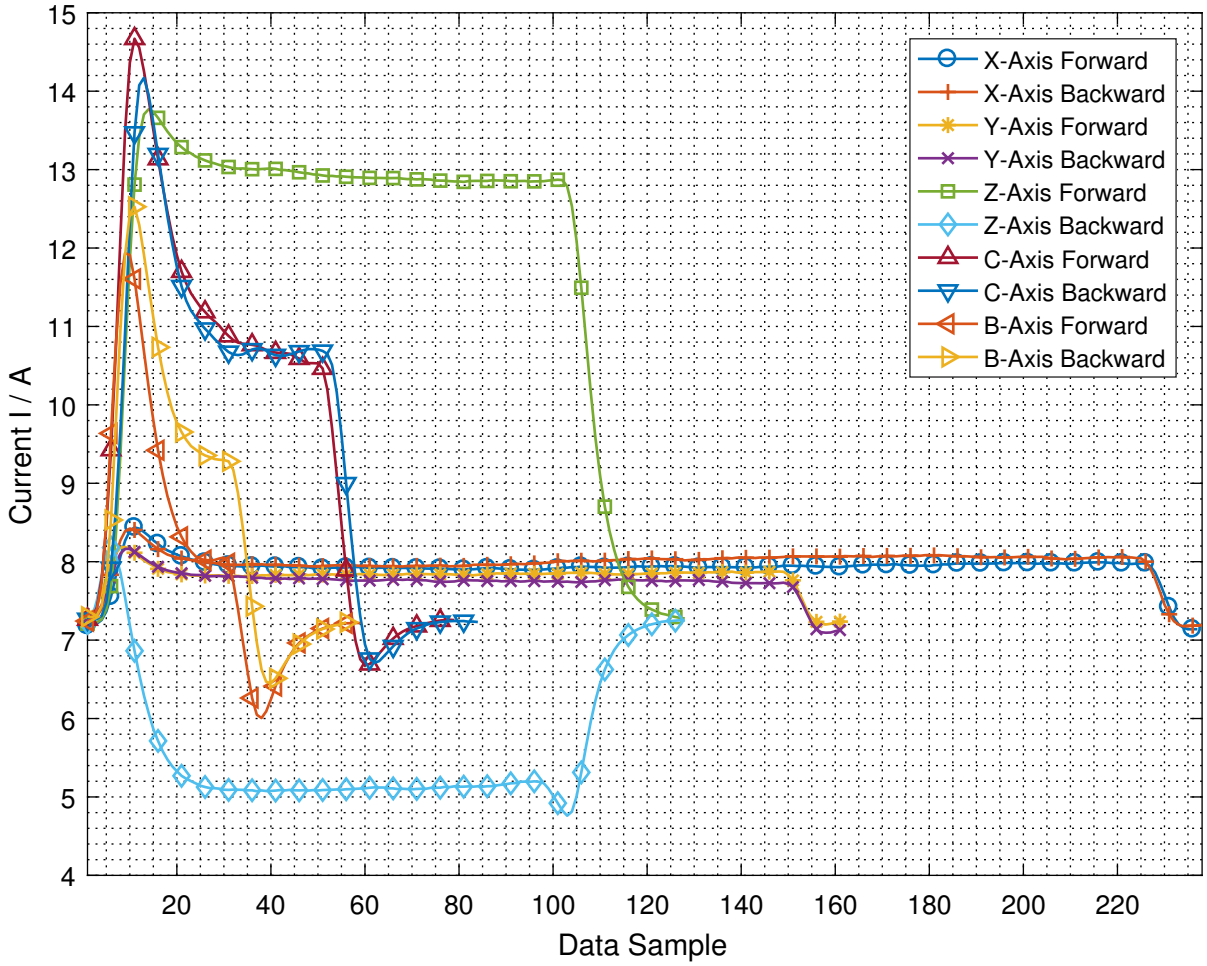
$$\|L\|_\infty = \max\{|x_i - y_i| : i = 1,2,\ldots,m\} \tag{5.1}$$

with measurement data $\boldsymbol{x} \in \mathbb{R}^n$ and template $\boldsymbol{y} \in \mathbb{R}^m$ has proofen to be best suitable for the given task. The value of this norm is depicted as the black line in fig. 5.9. As can be seen, the distance measure becomes minimal at specific points. These points refer to the template shifting positions at which the template matches the measurement signal optimally. Exactly this situation of a template match is schown in fig. 5.9. At this position a cut, represented by the red vertical line, is set to start a new segment and end the prior one. Afterwards the template is slid forward until the next match occurs and a new cut can be set. This process is repeatd until the end of the measurement signal is reached. Due to this template matching approach the whole segmentation process can be reduced to a problem of finding minima in the distance measure, which can easily be implemented in Matlab using the *findpeaks* function. This function is called with two additional arguments. The first one is the number of peaks that should be returned by the function. As the function usually returns every peak within the signal, it can not be ensured, that the number of peaks meets exctly the desired amount of 5 peaks per subcycle per template. So this value is fixed to the desired value and the function always returns the right amount of peaks. The second parameter causes the *findpeaks* function to return peaks in descending order based on the peak height rather than chronologgically after the peak occurance in the signal. This ensures in combination with the restricted number of peaks that only the most significant peaks within the signals are found.

In order to not only find state changes of the X-axis the template matching is repeated for any of the 12 templates from fig. 5.8 giving a subset of cut positions in each iteration. This can be seen in fig. 5.10 and fig. 5.11 as well as the diagrams in appendix A.4 which show each one complete subcycle of the second test cycle run and cut positions found automatically via template matching. For instance in fig. 5.10 the main spindle template is being matched to the measurement signal giving five cut positions, marked with red vertical lines. Actually the algorithm does not only consider the first subcycle, but rather all 20 subcycles in the measurement data. This results in a total amount of 100 cuts that are found due to template matching with the main spindle template. Matching of the X-axis forward template results in the cut positions shown in fig. 5.11. As can be seen the algorithm is not able to distinguish between the X-axis forward and backward movements as the two coresponding patterns in the current signal are very similar (see fig. 4.12b). This results in not only 5 cuts, but 10 cuts. This means comparison of the X-axis forward template helps to find the patterns of the X-axis



(a) Shape template of the main spindle.



(b) Shape template of all other controlled movements occuring in the test cycle.

Figure 5.8: Shape templates used for automatic segmentation of the measurement data. Each template is shifted over the entire measurement data and a distance measure as later criterion for an optimal cutting position is calculated.

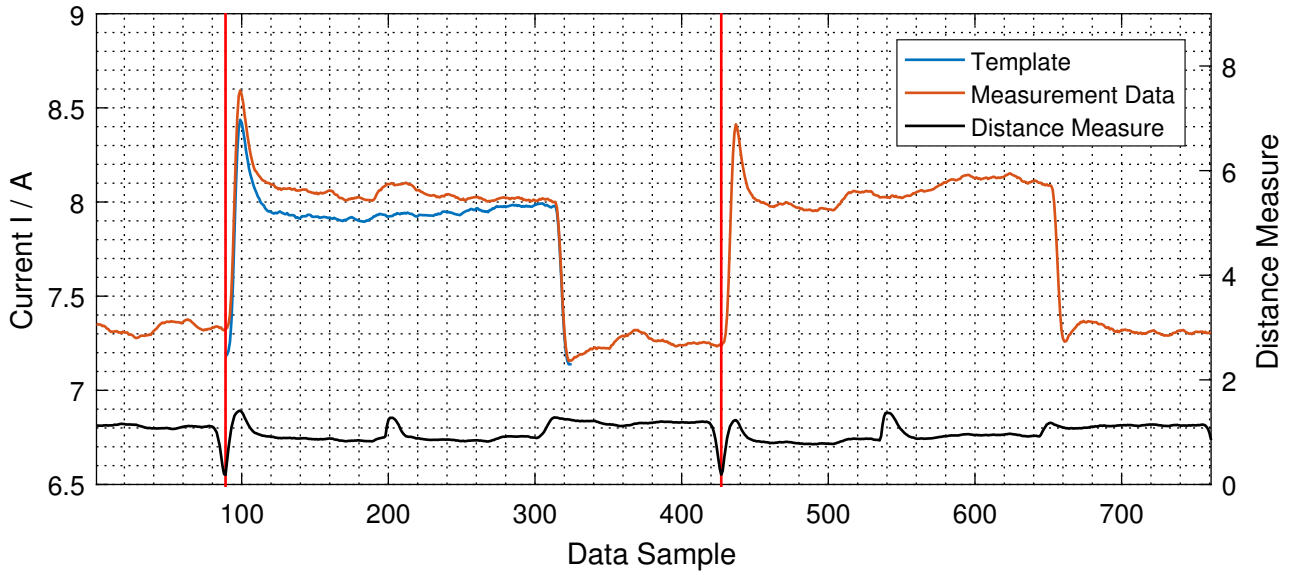


Figure 5.9: Principle of the automatic calculation of cut positions by means of minimizing a distance measure. The template (blue) is slided over the measurement data (brown) and a scalar distance measure (black) is calculated as function of the lag, that means shifting distance. Local minima in the distance measure correspond to a shifting position with maximum correlation between template and measurement data. Cuts (red) are placed at these positions.

forward and the X-axis backward movement in the measurement data at once. Matching of the X-Axis backward template in the next iteration of the superordinate loop leads to approximately the same cut indices. The algorithm handles this problem by calculating cut positions for both templates and discarding one of the two cuts. This is done in a post processing step, which requires first ascending sorting of the found cut indices as the *findpeaks* function does not return consecutive cut positions, but instead returns the cuts sorted by the depth of the corresponding minimum in the distance measure. The sorting allows for easy calculation of the spacing between cuts by differention of the cut position vector by means of the Matlab *diff* function. The differentiation result is then compared to a specific threshold to determine cuts which are located closely to each other. In this case the threshold value is chosen to be 50 data samples. If two cuts lie closer to each other than this threshold value, the cut lying on the lower data index is removed from the vector containing the cut positions. This is illustrated in fig. 5.12. Another important step in determining the cut positions is the locating of the end positions of each segment. The template matching approach only finds the starting position of each state change, but can not easily detect the endpoint of a segment. In this case endpoint means the same as starting point of the machine idle state or class 0, in between of each axis movement. In order to find these cut positions as well, a template matching with the idle state had to be performed. As this leads to a much higher number of minima in the distance measure the computational effort increases by a significant amount. To achieve a lower computational expense as well as an easier implementation of the algorithm the endpoints are determined in an easier way by adding the template length to the starting point of the corresponding segment.

As the pattern lengths can be assumed to be constant over time, this approach gives precise segment endpoint locations and maintains reasonable computational effort.

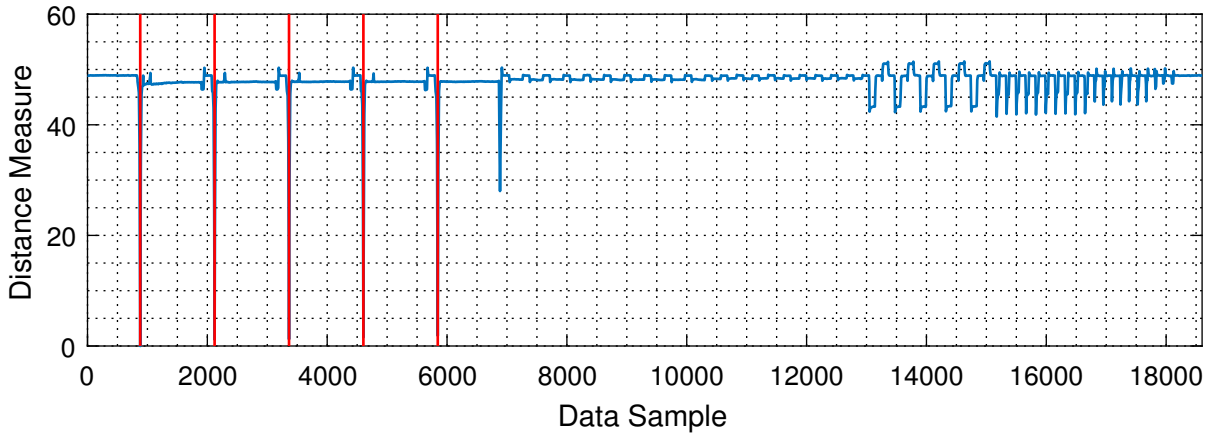


Figure 5.10: Distance measure for the main spindle template as function of the shifting lag. The lag is expressed in data samples of the measurement data. At minimum distance the measurement data looks most similar to the template and therefore a cut can be made at this position. Found cut positions are marked with red vertical lines.

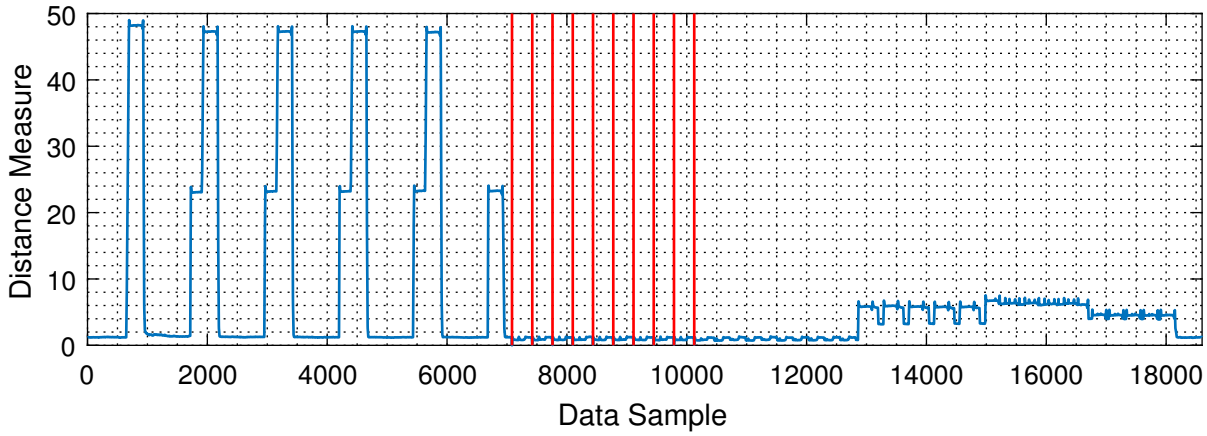


Figure 5.11: Distance measure as function of lag for the X-axis forward template. Due to the similarity of the X-axis forward and backward template the algorithm finds 10 local minima rather than only 5.

Result of the automatic segmentation can be seen in fig. 5.13, which shows sections of the measurement data with automatically inserted cuts. For better visibility only parts of the signal are shown representing the segmentation of the main spindle states, the X-axis states and the Z-axis states. Validate of performance of the segmentation algorithm can be done manually by looking at all cuts placed by the algorithm. This way it can be found that the algorithm performs very well and is able to determine every single state change precisely without missing individual state changes or inserting additional cuts at wrong positions. To further benchmark the algorithm not only the measurement data from the second test cycle run, but also measurement data from the first and third test cycle run are segmented by the algorithm. Here the algorithm is able to place every cut at the desired position automatically without any

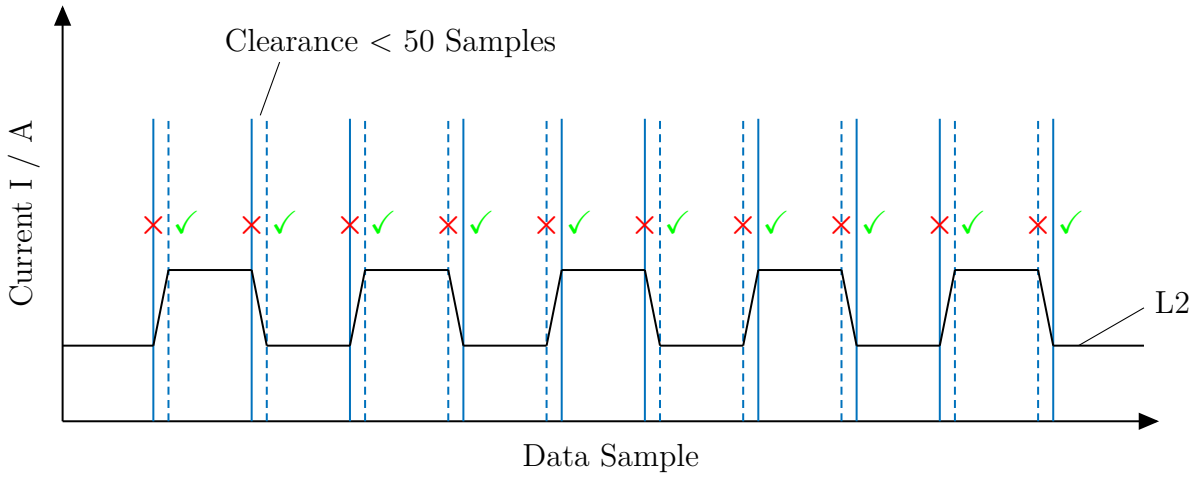
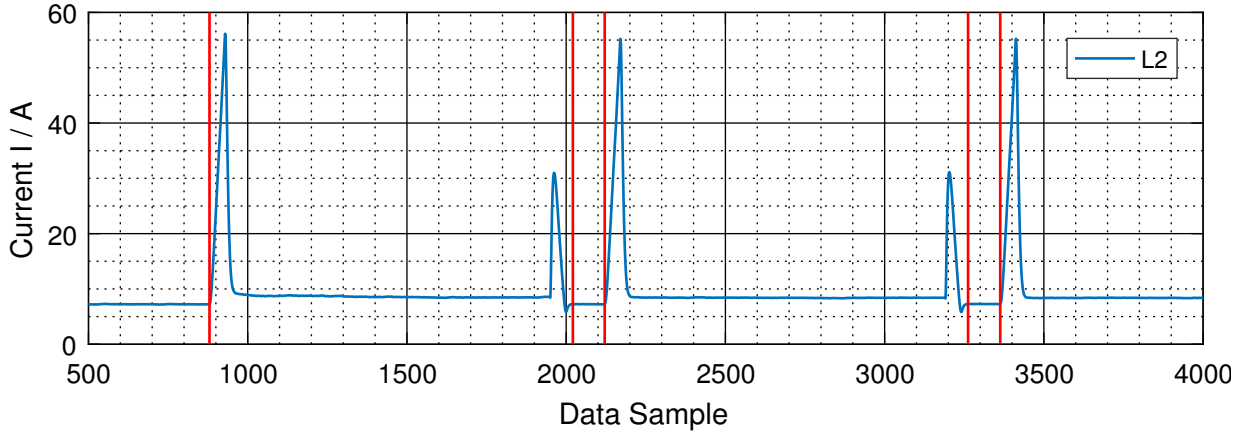


Figure 5.12: Removal of double indicated cut positions. Cut positions for some classes are determined twice, because the algorithm can not determine between forward (solid lines) and backward (dashed lines) movement of the corresponding axes. This results in twice the amount of cut positions. To reduce the number to the desired value, pairs of close cuts are recognized and the left cut is removed.

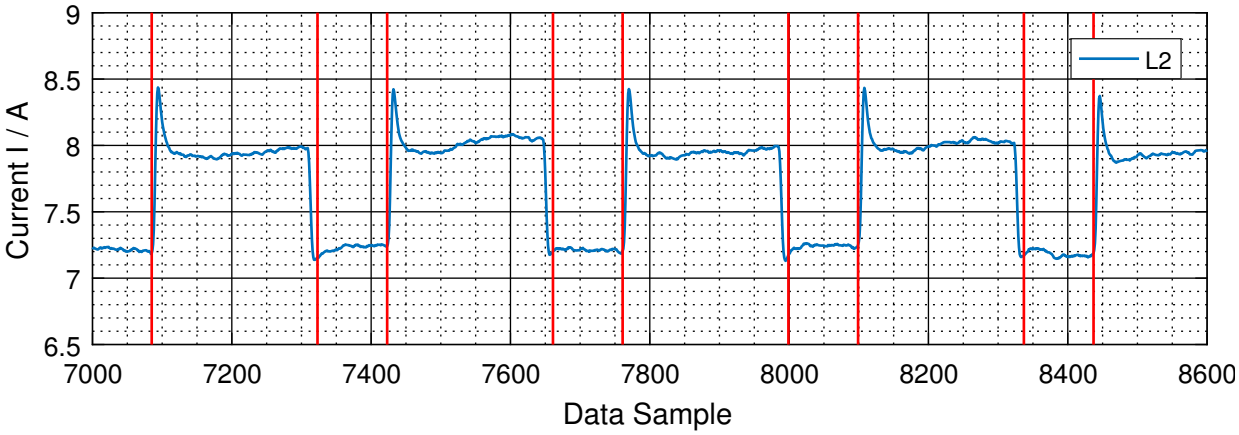
faults as well. Noteworthy is that the templates are not redefined for these two segmentation runs. This means even with templates from the second test cycle run data a proper segmentation of the first and third test cycle run data is possible. This is due to the high similarity of the measurement data of all three experiments.

5.3.2 Labeling of Controllable State Changes

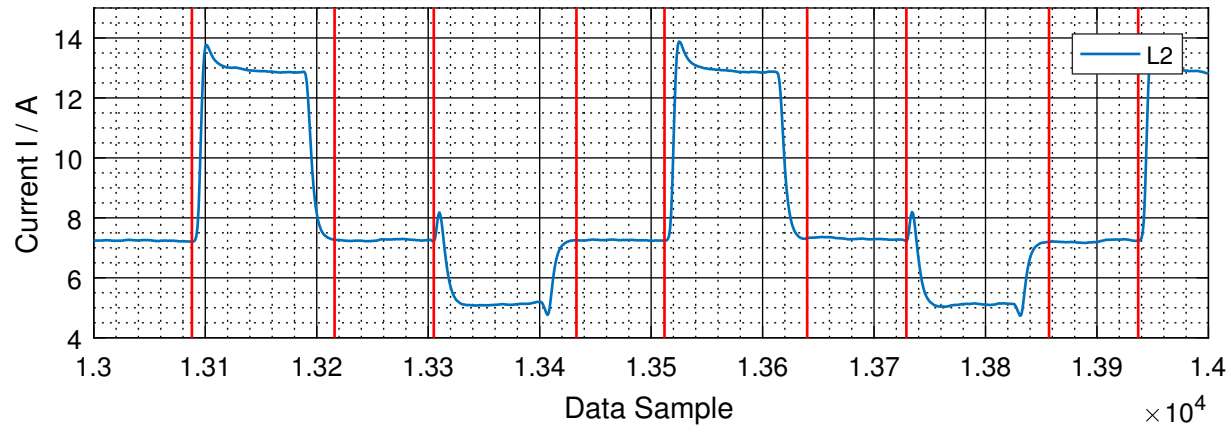
After the succesful segmentation of the measurement data class labels need to be applied to each segment. This means for each cut the segment right to this cut needs to be given an integer number according to the class definitions in section 4.3. As in this stage only the current on phase L2 of the main power line and therefore only controllable states are considered, it is sufficient to label the segments with class numbers from 0 to 11. The whole classification task is as easy as assigning the class of the current template in the matching process described above to each cut found in that iteration of the superordinate loop. To make it clear consider fig. 5.10 and fig. 5.11 again. These two diagrams show the resulting cut positions for different kind of templates. For instance fig. 5.10 shows the cut positions which result from the template matching with the main spindle template. As it is exactly known, that these cut positions belong to the main spindle template, it is possible to assign the corresponding class 1 for the main spindle movement to each of the segments demarkated by the found cuts. As the algorithm proceeds every template once and retrieves corresponding cuts for all templates, every set of cuts can be labeled with the corresponding class number. Advantage of this approach is a high reliability even in the rare case of a faulty located cut. If only the known fixed class sequence would be applied to the segmented measurement data, a single missing cut would lead to missclassification of all following segments. However the presented approach is robust against this kind of failure



(a) Automatically determined cut position during the spindle movement.



(b) Automatically determined cut position during the X-axis movement.



(c) Automatically determined cut position during the Z-axis movement.

Figure 5.13: Evaluation of automatic segmentation of measurement data considering only phase L2 of the current on the main power line. Red lines mark the cut position that were determined automatically via the template matching algorithm.

as it determines the class for each segment individually without relying on the application of the known fixed class sequence.

The only difficulty arises from the aforementioned problem of not being able to distiguish between for example the X-axis forward and backward movement which makes additional postprocessing in form of removal of desely lying cuts necessary. As mentioned the algorithm calculates distances between adjacent cuts and discards the cut with lower index in case of two cuts lying very close to each other. It can not be determined, whether the class of the deleted cut coressponds to the forward or backward movement of the axis. Therefore another postprocessing step of modifying the class sequence is necessary. In order to get the desired class sequence, first the distinction between forward and backward movement of the X-axis, Y-axis and C-axis is removed as these are the states the algorithm can not distiguish. This results in the same classification of forward and backward movement of the X-axis with class 2, respectively class 4 for the Y-axis forward and back movement and class 8 for both types of C-axis movement. Z-Axis and B-axis forward and backward movements are more distinct and can therefore be discrimated by the template matching approach. Therefore the different classes for forward and backward movement of these axes can be applied directly. The resulting class sequence for one subcycle of the test cycle data looks as follows

$$
\begin{aligned}
c = [&1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 2\ 0\ 4\ 0\ 4\ 0\ 4\ 0\ 4\ 0\ 4\\
&0\ 4\ 0\ 4\ 0\ 4\ 0\ 4\ 0\ 4\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 8\ 0\ 8\ 0\ 8\ 0\ 8\ 0\\
&8\ 0\ 8\ 0\ 8\ 0\ 8\ 0\ 8\ 0\ 8\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0] .
\end{aligned}
$$

As can be seen classes 3, 5 and 9 do no occur in the sequence due to the described problem. In order to get the correct class labels from this sequence, the subsequences $c_1 = [2\ 0\ 2]$, $c_2 = [4\ 0\ 4]$ and $c_3 = [8\ 0\ 8]$ need to be found and replaced by the subsequences $\hat{c}_1 = [2\ 0\ 3]$, $\hat{c}_2 = [4\ 0\ 5]$ and $\hat{c}_3 = [8\ 0\ 9]$. It is important to only replace every second occurance of the found subsequences to retrieve the desired class sequence. The output sequence for one individual subcycle then looks like following

$$
\begin{aligned}
\hat{c} = [&1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 2\ 0\ 3\ 0\ 2\ 0\ 3\ 0\ 2\ 0\ 3\ 0\ 2\ 0\ 3\ 0\ 2\ 0\ 3\ 0\ 4\ 0\ 5\ 0\ 4\ 0\ 5\ 0\ 4\\
&0\ 5\ 0\ 4\ 0\ 5\ 0\ 4\ 0\ 5\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 6\ 0\ 7\ 0\ 8\ 0\ 9\ 0\ 8\ 0\ 9\ 0\\
&8\ 0\ 9\ 0\ 8\ 0\ 9\ 0\ 8\ 0\ 9\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0\ 10\ 0\ 11\ 0] .
\end{aligned}
$$

Now the sequence contains the desired classes and distiguishes correctly not only between forward and backward movement of the Z-Axis and B-axis, but between the forward and backward movement of X-axis, Y-axis and C-axis as well. Problem of this solution is the lower reliability as it depends on the correct outlook of the prior class sequence c. If an error occurs within this sequence, the subsequence replacing might fail as well. Though only a few samples would be affected, rather than the complete sequence, so this disadvantage can be neglected and the solution accepted for automatic labelling of the segmented measurement data.

To validate the proper function of the automatic labelling, the output sequence of the algorithm is compared to the desired class sequence c. It is found that the algorithm performs well in labelling the segments of any of the three measurement test cycles correctly. Only in case of the measurement signal from the second test cycle run, the last segment, corresponding to a

backward movement of the B-axis is not cut and labelled properly. However this is due to the fact, that the signal has been read out from the database with too less margin at the end of the signal. This prevents the algorithm from calculating the distance measure for the last segment as the distance measure is always shorter by the number of samples of the applied template. This fault only means, there is one training segment less for the B-axis backward movement. As there are 299 more training segments of this kind, the problem can easily be neglected.

One last step has to be performed to finalize the automatic segmentation and labelling of the controllable states before the non-controllable states can be considered in the next section. In order to classify the first segment of the measurement data correctly, a cut at the first data sample has to be inserted, to mark the machine idle state, respecitvely class 0, at the beginning of the measurement dataset. The class sequence $\hat{c}$ changes slightly as a class label 0 is inserted at the first position in the sequence. Finally the previosly found cut positions and determined base classes for each segment are stored in the aforementioned CSV file.

5.3.3 Labeling of Non-Controllable State Changes

The algorithm presented in the prior section only deals with cutting and labelling of the controllable state changes. This means state changes of the main spindle as well as all five machine axes. Negotiation of the three non-controllable components is possible by considering only the current signal on phase L2 of the machine main power line, because none of the controlled components is connected to this phase. In the following section the automatic cutting and labelling algorithm shall be extended to enable automatic cutting and labelling of both kind of state changes under consideration of all three power line phases. A flowchart of this part of the algorithm can be found to the right side of the overall functional diagram of the cutting and labelling algorithm in fig. 5.7. As can be seen, the non-nominal labelling relies on the outputs of the first two functional segments of the algorithm, the segmentation and the nominal labelling. In the non-nominal cutting and labelling process additional cut positions of state changes of non-controllable components are found and merged with the already estimated cut positions of the nominal state changes. The previously determined class sequence containing only the 12 controlled state is used as well to calculate the class sequence of all 96 classes, which are possible under consideration of the non-controllable components. Output of this part of the algorithm is the previously described CSV file introduced in fig. 5.1 that contains all cut positions, the class of the corresponding segment and the nominality of each cut. This section will describe in detail how the automatic cutting and labelling of the non-nominal state changes is conducted.

The first step of detecting non-nominal state changes is the measurement signal analysis of all three non-controllable components, the machine lubrication pump, the oil-air lubrication pump and the control cabinet heat exchanger. From these three signal can the switching states of the three non-controllable components be retrieved via comparison with a fixed threshold value. Knwoledge of the switching states is important for later determination of the segment class. Figure 5.14 shows how the switching states of the control cabinet heat exchanger within the second test cycle run can be determined. As can be seen, the measured current value is compared to a threshold value, in this case 0.6 A. Whenever the measured current consumption of the control cabinet heat exchanger exceeds the threshold value the binary sequence takes on

the value 1. Accordingly the sequence takes on value 0 when the measured current lies below the threshold value. Output of this processing step is a binary sequence with the same length as the original discrete measurement data vector. Though, the example only shows the binarizing of the control cabinet heat exchanger, the same approach can be used for determination of the binary sequence of the measurement signal of the lubrication pumps. Only the threshold value is changed to a value of 0.1 A for the pumps.

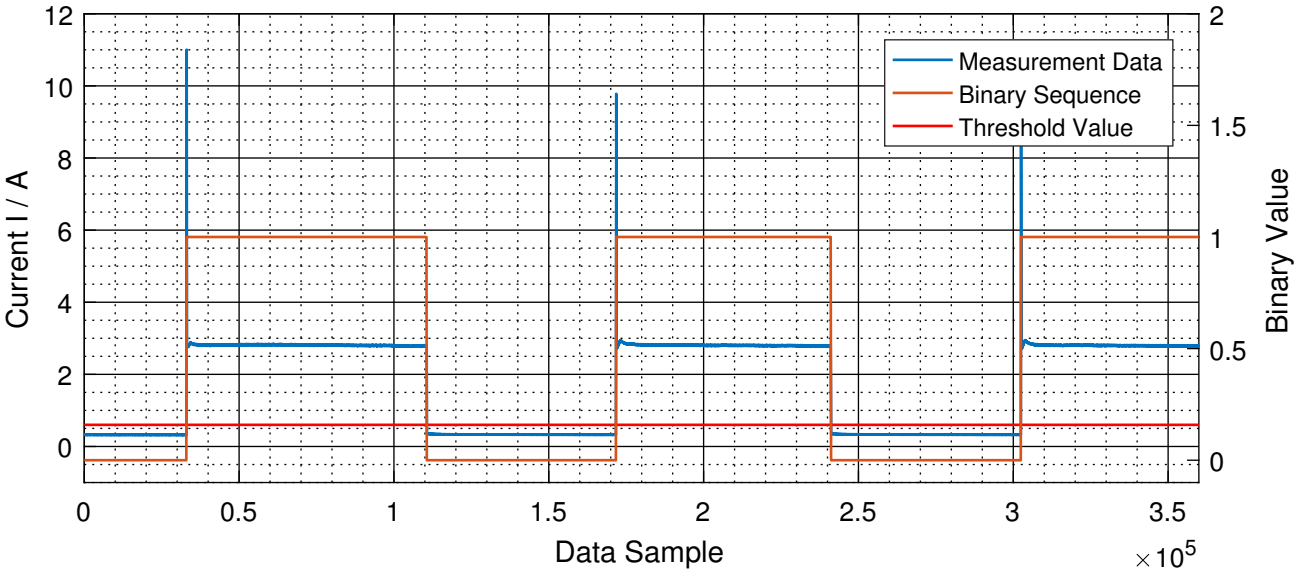


Figure 5.14: Functional priciple of converting measurement data of non-controllable components into a binary sequence. Whenever the measured current increases over the threshold value the binary sequence takes on the value 1, otherwise the sequence takes on the value 0.

To find the index values of the state changes of the non-controllable components the binary sequences need to be differentiated, which is conducted by means of the Matlab *diff* function. This function calculates differences between adjacent samples within the binary sequence. If no state change occurs, the difference is zero. Only at the state change from 0 to 1 the difference is 1 and on at the state change from 1 to 0 the difference takes on the value -1. Take for instance the binary sequence

$$s = [0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\ 0\ 0\ 0]\,.$$

For this sequence the *diff* function will output

$$\hat{s} = [0\ 0\ 0\ 1\ 0\ 0\ 0\ -1\ 0\ 0\ 0]\,.$$

Finding the positions of the state changes is now as simply as finding the non-zero elements within the sequence $\hat{s}$. This can be done by means of the *find* function in Matlab. This function returns the index values of all non-zero elements in a vector. For the example sequence the function will return the state change positions 4 and 8. Under consideration of measurement data of all three non-controllable components all non-controllable state changes can be found via this method. An example for a found non-nominal cut is shown in fig. 5.15. Here the control cabinet heat exchanger is activated and the corresponding cut position, which is marked with a

green vertical line, is determined via binarization and differentiation of the measurement signal of the control cabinet heat exchanger. Nominal cuts, which are found via application of the earlier described template matching approach are depicted as well as red vertical lines.

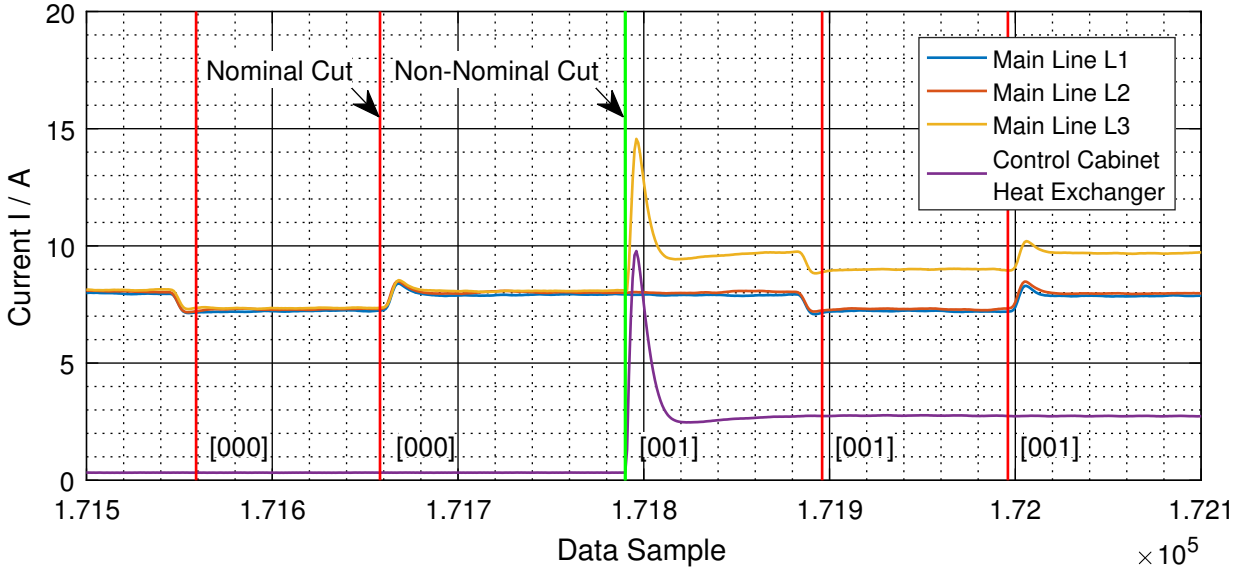


Figure 5.15: Example of nominal cuts (red) and a non-nominal cut (green) within the measurement data. The non-nominal cut is set due to activation of the control cabinet heat exchanger. Triplets to the right of each cut indicate switching states of the three non-controllable components within the current segment.

After succesful determination of non-nominal cut positions classes of the new created segments must be determined and class labels of existing segments must be updated in order to create the final class sequence containing all 96 possible classes as defined in section 4.3. To achieve this the algorithm first finds the nominal cut to the left of each non-nominal cut and assigns its class to the coresponding non-nominal cut. An example of this process is shown in fig. 5.16. As can be seen, the second and the last cut, coloured green, are non-nominal. All other cuts, which are coloured red, are nominal cuts. Classes of these segments are previously determined as described in section 5.3.2. So the nominal class sequence for the depicted section of the measurement signal is $c_{\text{nom}} = [2\ 0\ 3\ 0]$. Now the left neighbour of each non-nominal cut has to be found and its class assigned to the non-nominal cut. This gives the sequence of copied classes $c_{\text{copy}} = [2\ 0]$, which can be merged with the nominal class sequence c_{nom}. After merging both sequences the resulting sequence is sorted in an ascending order based on the position index of the cuts. This process gives the final class sequence $c_{\text{res}} = [2\ 2\ 0\ 3\ 0\ 0]$, which contains six elements for all six cuts rather than only four cuts like the previously determined nominal class sequence. So the length of the class vector is increased by the number of non-nominal cuts.

Parallel to updating the classes of each segment, the algorithm assigns the cut nominality. As defined earlier, a nominal cut is a cut that marks a state change of one of the controllable components, whereas a non-nominal cut marks a state change of one of the three non-controllable machine components. So prior to the aforementioned sorting of the classes vector another vector containing the nominality of both nominal and non-nominal cuts is created. This is possible because after merging nominal and non-nominal classes the nominal cuts are the first elements

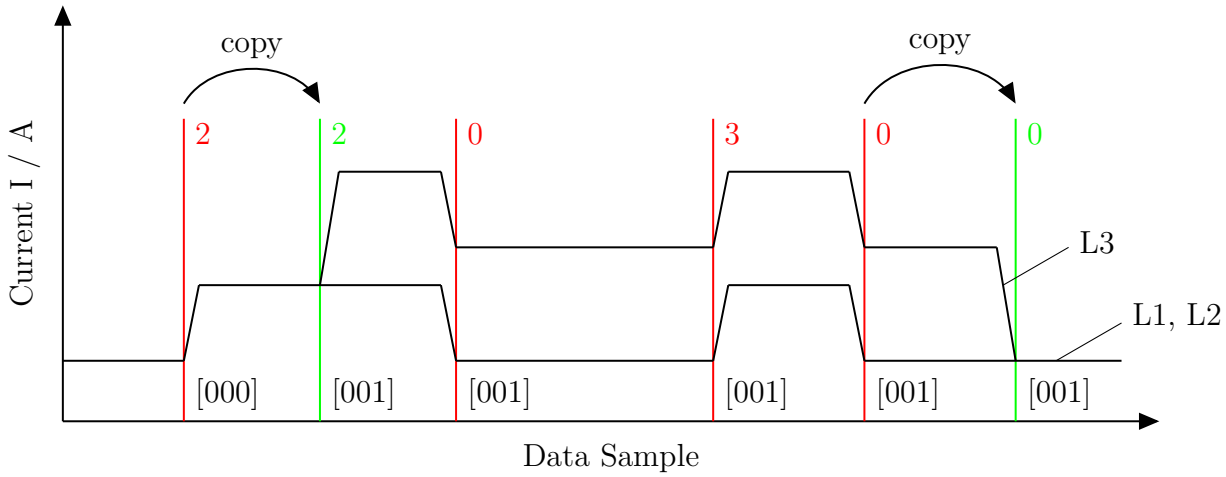


Figure 5.16: Schematic showing the lenghtening of the class sequence by copying the nominal class of the left neighbour of each non-nominal cut. The resulting class sequence needs to be updated by consideration of the switching states of non-controllable components shown in numerical triplets to the right of each cut.

of the vector and the non-nominal cuts follow after the nominal cuts without being mixed. So the first elements can be given the nominality 1 as they are nominal cuts and the remaining cuts are marked with nominality 0 as they are non-nominal cuts. During the mentioned sorting of the cuts based on their positions not only the coresponding classes, but also the cut nominality are sorted in the same manner.

Before the class labels of all segments can be recalculated another processing step is conducted. This steps aims to mark very short segments in the measurement data, which occur every time a non-nominal cut is very close to a nominal cut. In this case the resulting segments only contain a few data samples and are therefore not usable as samples for the later classification task. As the shortest template, the template for the B-axis backward movement, has a length of 56 data samples, the threshold value for marking a segment as too short is set to 20. So every resulting segment that contains less than 20 data samples is marked. The demarkation of those elements is done by setting the segments nominality to a value of -1. So it is possible to decide in later process steps, whether to use these segments or discard them.

Finally the algorithm needs to update the lengthened class sequence, because it does only contain the 12 base classes rather than a set of all possible 96 classes. In order to find out the actual class of every segment, it is necessary to know, which switching states of the three non-controllable components are taken on in each segment. For determining these switching states the previously created binary sequences are used again. Finding out the swithcing state of the machine lubrication pump, the oil-air lubrication pump and the control cabinet heat exchanger is done by looking up the binary value to the right of each cut position. If for example a cut has been placed on data sample 880, the switching states can be determined by looking up the binary values at index 881 in the three binarized sequences for the coresponding components. The results of this lookup operation are shown in fig. 5.15 and fig. 5.16 as numerical triplets to the right of each cut. In these triplets the first number refers to the state of the machine

lubrication pump, the second nuber to the oil-air lubrication pump and the third to the control cabinet heat exchanger, whereby value 0 means the component is not activated and 1 means the coresponding component is activated within the current segment. Now for every segment the switching states of the three non-controllable components are known. In combination with the already known base class in the range of 0 to 11 it is possible to determine the actual class of each segment by a table lookup within the class definition table (see table 4.3). Considering the structure of the class definition table the classes can easily be calculated via adding a scalar value, which depends on the switching states of the three non-controllable components. If for instance the control cabinet heat exchanger is acitvated while both lubrication pumps are deactivated, that means the triplet would be $k = [001]$, the desired class can be found by adding the scalar value 12 to the previously determined base class. For example given a segment in which the main spindle and the control cabinet heat exchanger are activated, the previously determined base class, which does not consider the non-controllable components, would be 1. Adding the scalar 12 gives class 13, which is the correct classs for a segment in which both main spindle and control cabinet heat exchanger are activated. This can be checked by looking at the class definitions in table 4.3. Table 5.2 contains the scalar values that need to be added for all eight possible switching combinations of the three non-controllable components.

Table 5.2: Scalar values for recalculation of class labels based on switching states of the three non-controllable components. These values need to be added to previously assigned base class labels to get proper class labels under consideration of the non-controllable components.

Machine Lubrication Pump	**Oil-Air Lubrication Pump**	**Control Cabinet Heat Exchanger**	**Scalar**
0	0	0	0
0	0	1	12
0	1	0	24
0	1	1	36
1	0	0	48
1	0	1	60
1	1	0	72
1	1	1	84

Big advantage of this approach compared to a normal table lookup is the low computational effort required for updating the class label. On a normal table lookup for each segment all 96 classes had to be compared to the configuration of base classes and switching states of non-controllable components. This comparison is computationally much more expensive than a simple addition of a scalar value for each segment.

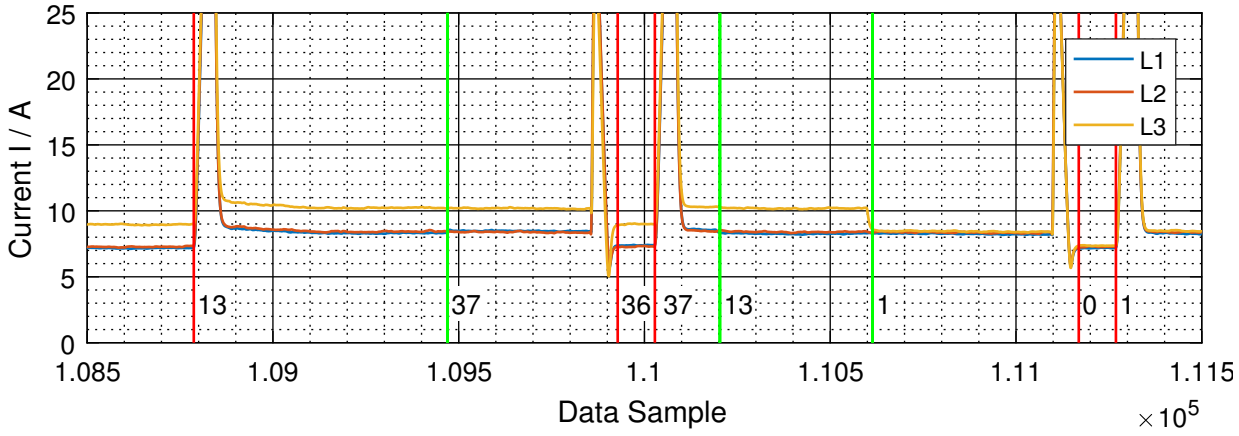
5.3.4 Results of Automatic Segmentation and Labeling

A functional demonstration of the whole algorithm can be seen in fig. 5.17. In fig. 5.17a the current signal of the main power line, in fig. 5.17b the signal of the control cabinet heat

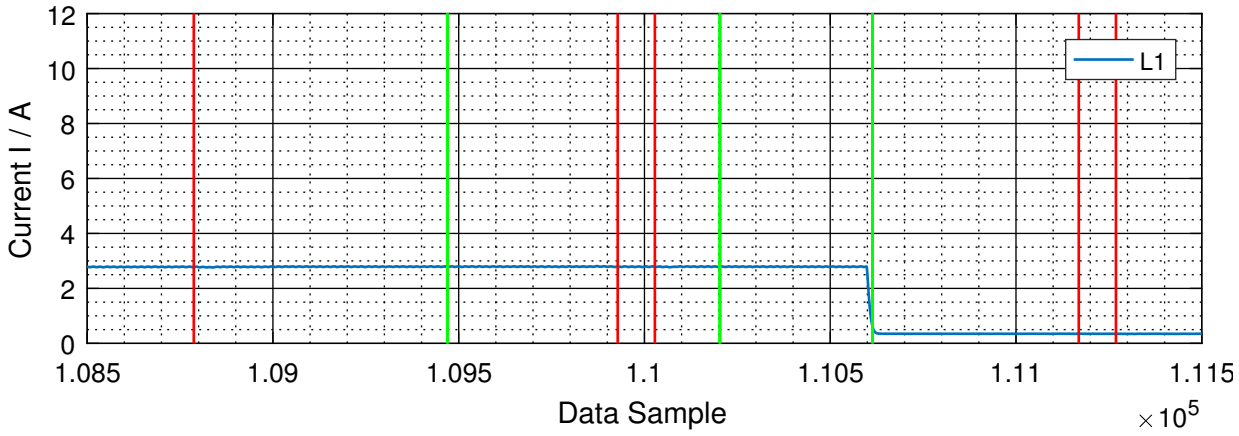
exchanger and in fig. 5.17c the signal of the lubrication pumps is depicted. Cuts have been placed automatically by the algorithm, where nominal cuts are marked by red vertical lines and non-nominal cuts are represented by green vertical lines in every of the three subplots. As can bee seen, the algorithm properly detects every controllable state change as well as any non-controllable state change. Moreover the assigned class labels for each segment are shown in the plot of the main power line signal. Having a look at the full class definition table in appendix A.1, it becomes clear that the assigned classes are correct. After the first nominal state change, the main spindle is activated and simultaneously the control cabinet heat exchanger runs. This gives class 13, which has been assigned properly. The next cut is a non-nominal state change, because the oil-air lubrication pump is activated. Therefore the class number of the next segment is 37. When the main spindle movement stops and the machine falls into idle state the class label changes to 36 and back to 37 after the subsequent activation of the main spindle. Deactivation of the oil-air lubrication pump leads to another non-nominal state change and change of the class label back to 13. Next the control cabinet heat exchanger is switched off and therefore only the the main spindle is active, while any off the non-controllable components is deactivated. This results in assignment of class 1 to this segment. The following idle state is naturally 0 and after another activation of the main spindle the class label is again 1. Having a look at the labelling of the complete measurement test cycle it becomes clear, that the algorithm works properly and classifies all segments correctly. This works for any of the three measured test cycle runs.

Another important feature to test is the detection and demarkation of cuts, which are very close to each other. As mentioned earlier, the left cut of a pair of closely spaced cuts is marked with a nominality value of -1. Such a situation can be found in fig. 5.18. Here a section of the measurement signal from the second test cycle run is shown. As before nominal cuts are marked with red vertical lines and the only occuring non-nominal cut is marked with a green vertical line. In this section the special case of two closely lying cuts occurs. The dashed blue line and the green line lie very close to each other, only seprated by 12 measurement data samples. As this value falls below the threshold value of 20 Samples, the nominality of the cut is marked with value -1 and the corresponding cut is show as dashed blue line. In any further analysis the segment between the blue and the green line can now be discarded easily by checking if nominality equals -1. It becomes clear that this approach is useful as the segment between both lines does not contain any specific kind of pattern, but rather only a rising edge of the current signal, which does not provide enough information for the later classification task.

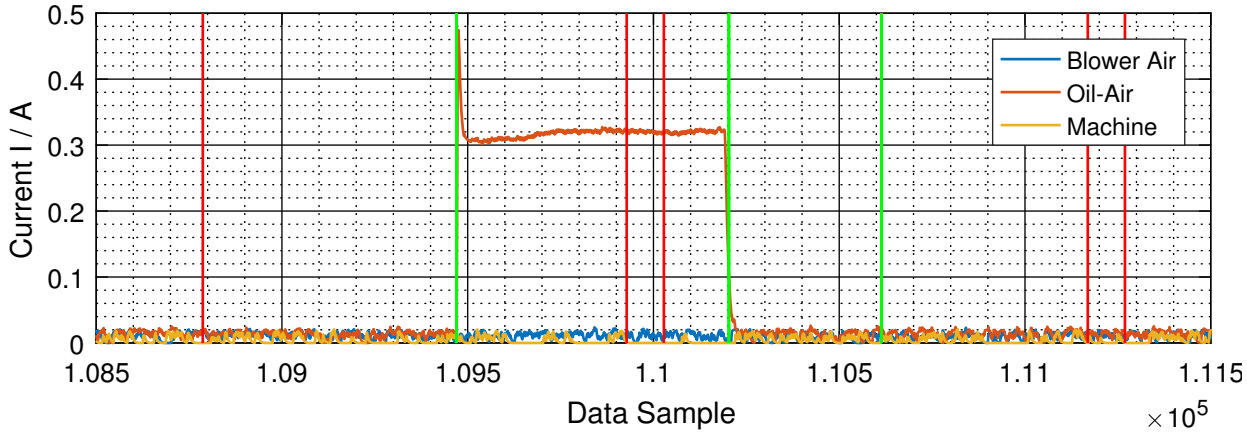
After running the automatic segmentation and labelling algorithm for every three measurement dataset, the statistics depicted in table 5.3 for the distribution of different kinds of cuts can be created. The first three rows consider only the nominal state changes and 12 base classes. The subsequent three rows regard to the complete labelset containing different kind of cuts and all 96 possible classes. As can be seen the number of nominal cuts varies in a specific range. For the first three rows this can be explained by the fact that the last segment within the second test cycle run is not segmented correctly as mentioned earlier. Usually this number should be equal for every test cycle run. The variations in the number of nominal cuts within the labelset which considers every type of cut is comprehensible as some of the nominal cuts might be marked as closely lying or dense cuts. Moreover the fluctuations in the total amount of segments can be explained by the varying number of non-nominal cuts, which is usually not equal for different test



(a) Automatically determined cuts in the main power line signal.



(b) Cut positions in the signal of the control cabinet heat exchanger.



(c) Cut positions in the signal of the lubrications pumps.

Figure 5.17: Evaluation of automatic segmentation of measurement data considering all three phases of the main power line as well as the three non-controllable components. Red lines mark cut positions of nominal state changes, green lines non-nominal state changes. In the main power line signal additionally class labels for the segment to the right of each cut are shown.

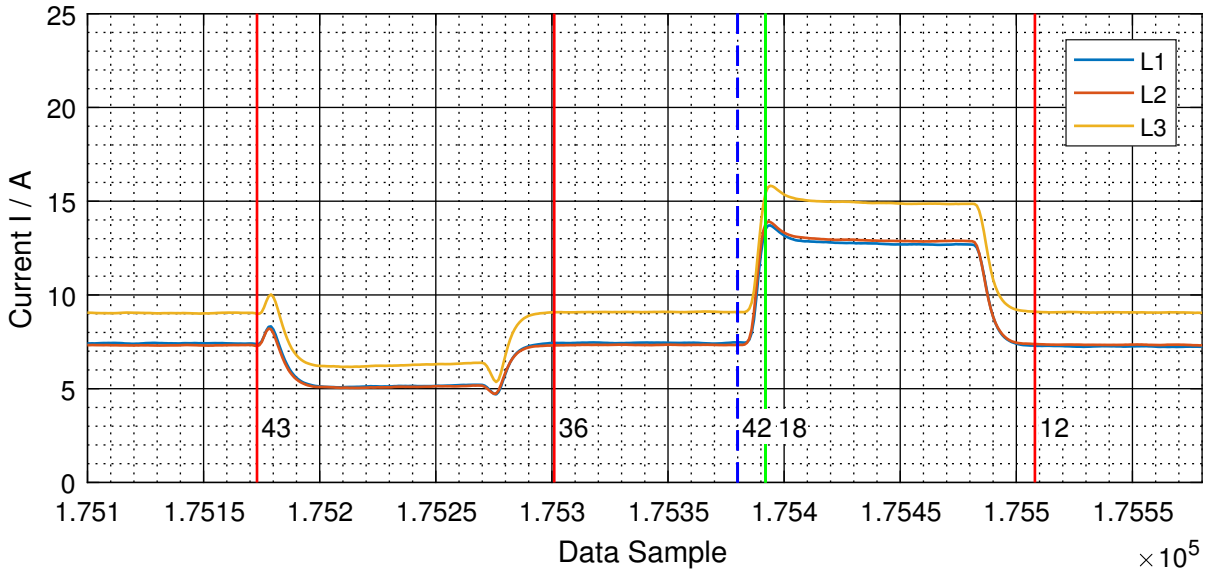


Figure 5.18: Example for detection of closely lying cuts. The dashed blue line represents a cut that lies too close to the non-nominal green cut and is therefore marked with a nominality value of −1.

cycle runs as the non-controllable components can be switched on or off irregularly introducing a new segment on each state change. The same counts for the number of dense cuts, which is only influenced by the random distribution of cut positions. Because the positions of the non-nominal cuts vary in each test cycle run, the pairs of closely lying cuts differ as well.

Table 5.3: Statistics of the segmented and labeled dataset. The number of segments corresponds to the total number of cuts in the signal. Dense cuts represents the number of cuts marked with nominality −1 due to lying too closely to another cut. The first three rows refer to the labelling of only nominal state changes and base classes, the following three rows consider all types of cuts and all 96 classes.

Test Cycle Run	Segments	Nominal Cuts	Non-Nominal Cuts	Dense Cuts
1	2201	2201	–	–
2	2199	2199	–	–
3	2201	2201	–	–
1	2250	2194	38	18
2	2249	2193	43	13
3	2251	2191	43	17

Another important information for later development of a classifier is the distribution of classes within the labeled datasets. These distributions can be illustrated with help of histograms as shown in fig. 5.19. This histogram contains the class label distribution for all three labeled measurement datatsets from the different test cycle runs considering only nominal state changes and therefore only the 12 possible base classes. As can be seen, class distributions within all

test cycle runs are very similar to each other. The majority of samples belongs to class 0, that means the machine idle state. This is reasoned by the idle phases, which occur after each axis movement in the test cycle. Any other class occurs 100 times in all three test cycles except for the second test cycle run, in which the last segment is not detected correctly as mentioned earlier. Even distribution of class labels can be explained by the fixed number for occurance of each of the different movements within the test cycle. Class distributions change dramatically, when non-nominal state changes are considered beneath the nominal state changes. The resulting distributions can be found in the histograms in fig. 5.20 for the three different testcyce runs. Here the numberof possible classes is 96. As can be seen, the amount of samples per class is heavily imbalanced, there are around 400 Samples in class 0, the machine idle state and around 600 samples in class 12, the idle state with simultaneously activated control cabinet heat exchanger. The number of samples in classes 1 to 11, which correspond to the base classes, lie around 40 samples. In classes 13 to 23, which refer to the different axes movements with simultaneously activated control cabinet heat exchanger, this number is between 50 and 60 samples per class. Much lower frequencies occur for the higher classes, which correspond to axes movements with concurrent activation of the machine or oil-air lubrication pump, combinations of those two and combinations of the two lubrications pumps with the control cabinet heat exchanger. As these machine states are much more uncommon than the base classes with eiter activated or deactivated control cabinet heat exchanger, the number of samples per class is very low for the higher classes. Hence there are classes, which only contain a single sample or even no sample. Moreover there are classes, which have a single sample in only one of the three test cycle runs. These imbalanced class distribution will have a large impact on the further classifier development. Therefore consequences of this unique characteristic of the present dataset will be analysed in further course of this work.

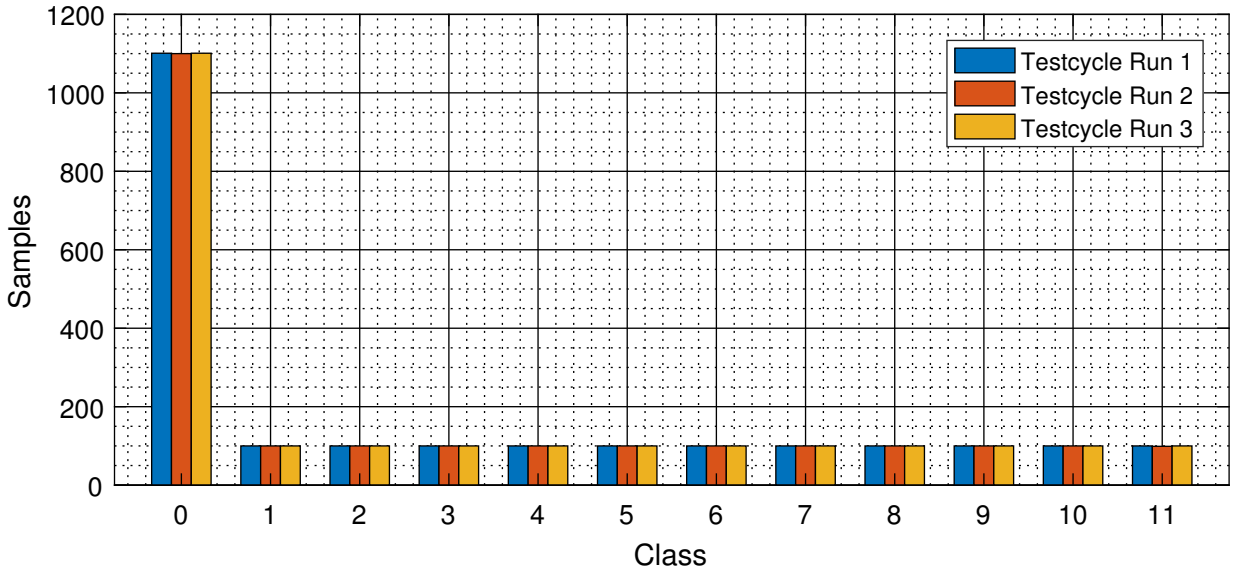
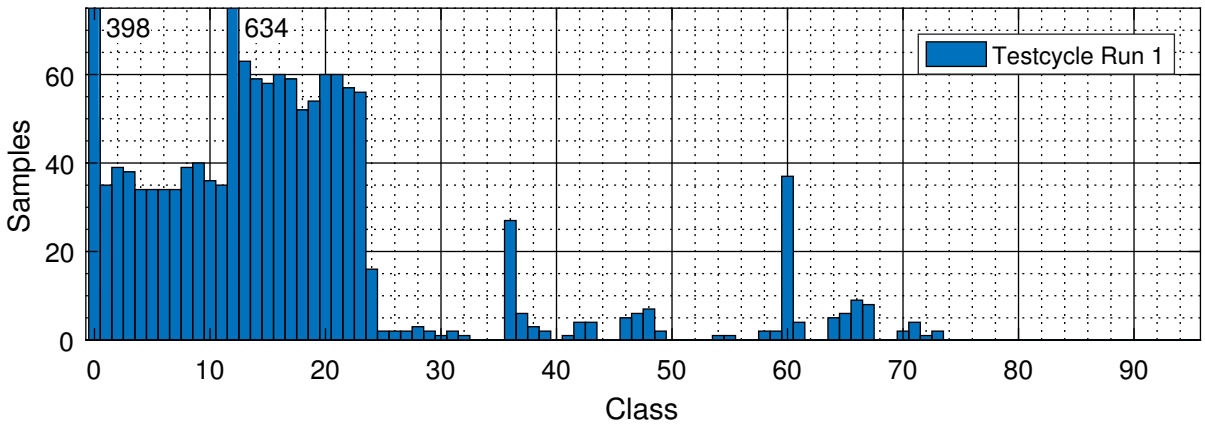
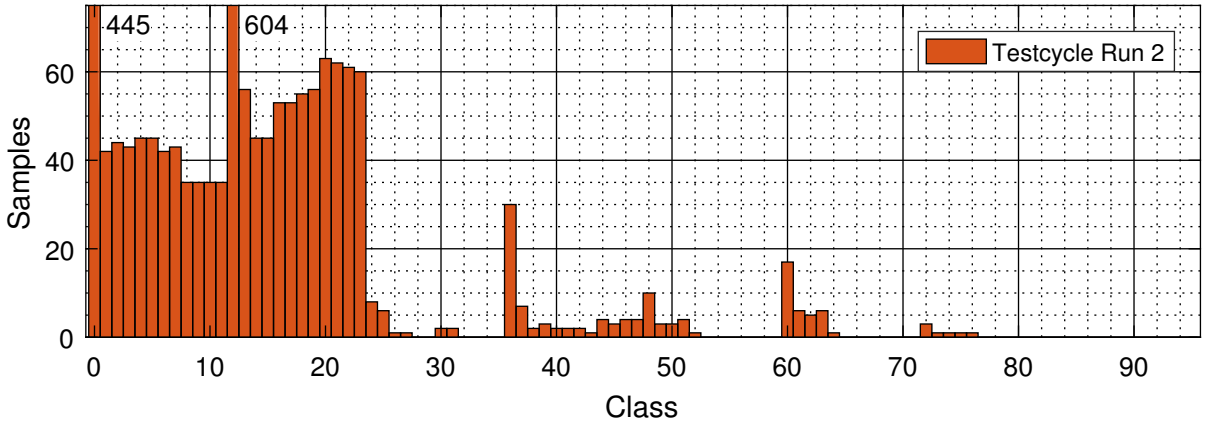


Figure 5.19: Histogram of class distribution within created labelsets of the three test cycle runs considering only nominal state changes and therefore only the 12 base classes.

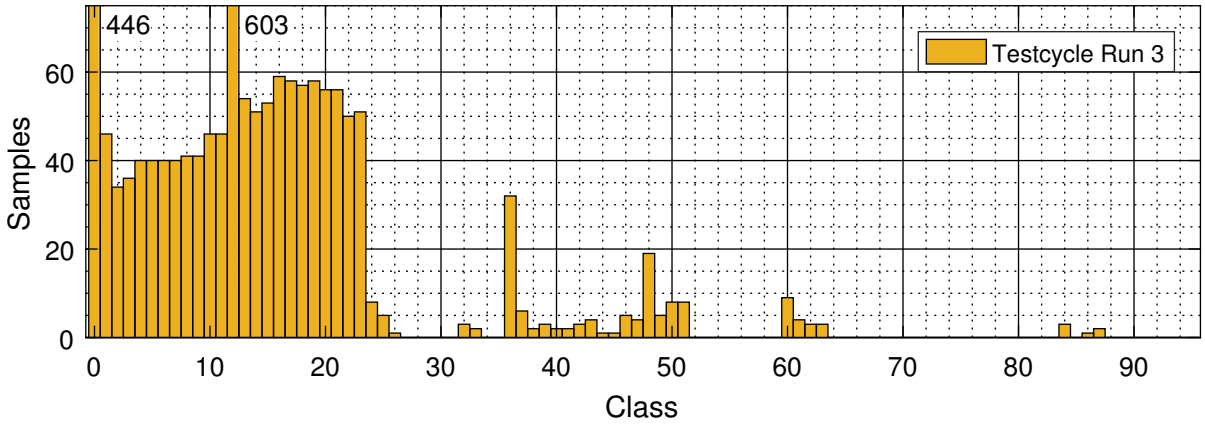
Summarizing the previous chapter two methods for segmentation and labelling of the given measurement dataset have been developed. One manual approach with help of a graphical



(a) Class distribution within first test cycle run.



(b) Class distribution within second test cycle run.



(c) Class distribution within third test cycle run.

Figure 5.20: Histograms of class distributions within created label sets of the three test cycle runs. Considered were all kind of state changes and therefore all 96 possible classes.

user interface and a fully automated approach. The fully automated approach uses templates of known patterns to determine at which positions the individual segments start and end. Segmentation of non-controllable state changes is done by thresholding the measurement signals of the three non-controllable components and finding state changes in these binary signals. Each segment is then labelled automatically with one of the 96 possible class labels. Moreover the algorithms determines, whether a cut is nominal, non-nominal or if the segment contains too less elements. Finally two CSV files are created. The first one contains indices of every cut position, the nominality of each cut and the class of each coresponding segment. The second one contains similar information, however comprises only nominal cuts and the 12 base classes. All three in this way segmented and labeled measurement signals can now be further processed and used for development and evaluation of a classifier.

6 Shape Based Classification

In the following chapter two different approaches for classifying time sequences of the previously segmented measurement data are proposed, implemented and evaluated. Different from the usual machine learning approach, which will be conducted in the subsequent chapter, both classifiers developed in the course of the following chapter will operate directly on the time domain data without any need for transforming the time sequences into feature space. The proposed classifiers utilize *Dyamic Time Warping* (DTW) and cross-correlation to calculate similarity between test sequences and reference sequences for each class, so called class templates. Class labels are then assigned according to the most similar class template for each test sequence. Similarity in this case refers to the signal shape in the time domain, which is why this classification approach is called *shape based template machting*. Prior to development of the classifiers foundations of shape based template matching are explained. First a short introduction in the overall process of classifying time sequences based on template matching is given, followed by two sections introducing Dynamic Time Warping and cross-correlation as typical measures for determining class membership of time sequences by comparison with class templates. After explanation of the basic concepts class templates for each of the 12 possible base classes the milling machine can take on are created. For this purpose a literature review is coducted and an overview over different existing shape averaging methods is given. Afterwards foundations of creating templates by averaging over multiple training sequences and simultaneously maintaining basic shape information of the signal are presented. Finally one specific algorithm for sequence averaging, the *Accurate Shape Averaging* (ASA) algorithm is picked out and the major steps of the algorithm are described in detail. Afterwards the ASA algorithm is implemented in Matlab and equipped with some further functionality to create and store class templates for all 12 base classes in one execution of the algorithm. Results of this algorithm are then presented. After class template creation the actual classifier algorithms are developed and the algorithm is explained in detail. As both algorithms have multiple freely selectable parameters the classifier algorithm is adpoted to conduct an exhaustive search for the optimal set of parameters. In order to find the optimal parameter set classification accuracy is evaluated and compared for different sets of parameters. In the end of the chapter some conclusions are drawn and proposals for further improvements of the developed algorithms are made.

6.1 Foundations of Shape Based Classification

The following section gives a detailed overview of the foundations needed for understanding of the template creation and template matching algorithms which are implemented in the later course of this chapter. First some basic concepts of the overall process of shape based classification via template matching are introduced. Afterwards *Dynamic Time Warping* (DTW) and cross-correlation are explained in detail. Both are important measures for determining

similarity of test and reference sequence. DTW is furthermore used for shape based non-linear alignment of time sequences which provides a distance measure that is unsusceptible to phase shift or non-linear distortion of the time base of both test and reference sequence.

6.1.1 Principle of Shape Based Classification

In the following section foundations of time series classification shall be briefly introduced. Finding and identifying subsequences in larger time series databases is a very common problem in the research community and many different approaches to solve this problem have been proposed. For instance [70] proposes a method of dismantling a larger time series into smaller linear pieces. Afterwards he calculates a distance measure to compare the query sequence with the database and find possible locations of the subsequences. A similar method is developed in [71], where a similarity measure is used to directly compare two time sequences as well. An approach that is robust to noise, scaling and translation of patterns to be matched is proposed in [72]. Another method presented in [73] utilizes *Dynamic Time Warping* (DTW), which is introduced in section 6.1.2, to compare sequences that are even misaligned and distorted in amplitude and time. Advantage of this property of the DTW is also taken in [74], where a method called *Spatial Assembling Distance* (SpADe) is proposed. This method is capable of finding subsequences and patterns not only in static but also in streamed time series data. Besides these more typical methods that calculate a similarity or distance measure between the sequences, there exist completely different approaches. For example in [75] a *Hidden Markov Model* (HMM) is used to locate and identify subsequences within a larger time series database. Apart from that [76] utilizes the *Discrete Fourier Transform* (DFT) to transform the sequences from the time domain into the frequency domain and compare both sequences based on the first few frequencies, which are most representative for the sequence. A further concept introduced in [77] uses a sliding window to transform a subsequence of the time series database into a trace in the feature space. This trace is then compared to the trace of the subsequence and similarity between both sequences is measured. Finally [78] proposes an algorithm for calculating cross-correlation and applies it to a template matching problem in an image database rather than time series database. However as cross-correlation will be used in the later classification task as well this paper provides valuable information about determining pattern location via cross-correlation. More general information about time series classification can be found in [79] and [80].

General task of the subsequent template matching classification is to identify different types of segments of a larger time series. This time series as well as the smaller segments are univariate, real-valued and uniformly sampled time sequences. Segments of the larger time series are created by automatic segmentation as described in section 5.3. The classifier has to assign a specific class label to each of the segments in the time series as shown in fig. 6.1. Here the time series contains four different kinds of patterns, which correspond to class labels 0 to 3. The class for each segment is predicted based on a template matching approach. This means there are four different templates corresponding to each of the four classes respectively. Templates for the example patterns in fig. 6.1 are shown in fig. 6.2. Template matching means, the classifier algorithm compares each of the segments that need to be classified with the four templates

and calculates some similarity measure for each of the four templates. Then the algorithm decides for the most similar template and assigns the template class to the current segment. This comparison is repeated for every segment in the larger time series until all segments are labeled.

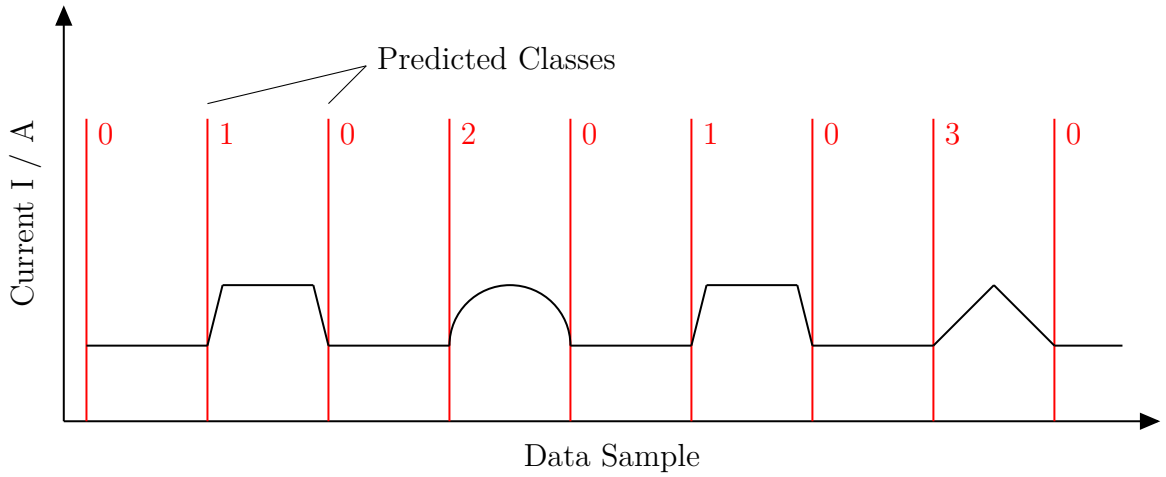


Figure 6.1: Example of a larger time series containing multiple instances of the four example patterns. Task of the classification algorithm is to assign one of the four possible classes $\{0, 1, 2, 3\}$ to each of the time segments within the larger time series.

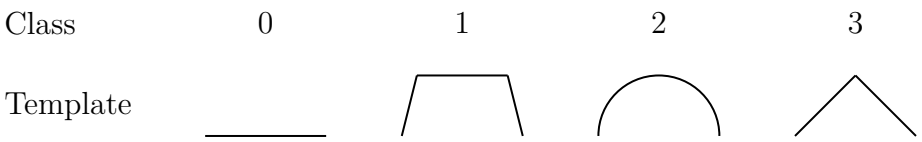


Figure 6.2: Four different kinds of example patterns or templates that need to be classified within a larger time series by template matching.

Successful implementation of a template matching approach comprises multiple steps. First the time series needs to be segmented, which is done either automatically as described in section 5.3 or manually as explained in section 5.2. After this templates for the occuring patterns, representing a specific class each, need to be created. This is done in section 6.2 by averaging multiple instances of training patterns from the training data sets in a particular way. After this an algorithm which sequentially compares every segment with every template and decides which template is most similar to the current segment needs to be developed. This is done in section 6.3. As this algorithm has multiple parameters, which can be chosen freely, it is necessary to conduct a parameter studies to find out optimal values of the paramters. Results of these parameter studies can be found in section 6.4. When the optimal paramters are identified performance of the classifier can be evaulated. Evaluation is necessary, as the classifier usually does not predict the correct class for every single segment in the larger time series. As there is a high number of segments in the test set it is possible that wrong class labels are assigned to some segments. The goal of designing a good classifier is to minimize this prediction error and predict as many classes as possible in the correct way. The number of correct predictions can be

used to define a measure for the classifier performance, the so called *accuracy*

$$\eta = \frac{\mathrm{TP}}{N} \tag{6.1}$$

with the number TP of segments, which were classified correctly and the total number N of segments. If every segment is labeled with the correct class accuracy would be 1. The other extreme is an accuracy of 0 which corresponds to incorrect labelling of each segment. The evaluation metric is also used for determining optimal parameters in the previously conducted paramter studies.

Core of the template matching classification is calculation of similarity between template and current segment of the larger time series, the test sequence. As already mentioned in the literature review, there exist multiple different approaches of determining similarity between template and test sequence. In the following two different approaches are implemented. First a distance measure based on *Dynamic Time Warping* (DTW) is used. It is explained in detail in section 6.1.2. The second approach, which is implemented and evaluated in the later course of this chapter is a similarity measure based on the cross-correlation between template and test sequence. Fundamentals of cross-correlation are presented in section 6.1.3. Idea of both approaches is to calculate a real-valued scalar quantifying similarity between template and test sequence. Different from simple distance or similarity measures like euclidean norm, manhattan norm, infinity norm (see [40]) shape of template and test sequence is considered in calculation of the distance measure. This non-linear alignment based on amplitude values is explained in the fundamentals section of the DTW. Cross-correlation usually does not provide such a shape based alignment, in this case however, template and test sequence are first non-linearly warped by means of DTW and afterwards the cross-correlation is calculated. Both template matching classifiers will be implemented and parameterized in the subsequent sections and performance will be evaluated and compared.

6.1.2 Dynamic Time Warping

An important distance measure for discrete signals in the time domain can be computed via *Dynamic Time Warping* (DTW), first introduced in [81]. It is extensively used in the following classification utilizing template averaging and template matching and therefore a brief overview of the theoretical foundations of DTW are presented in this section.

DTW provides an accurate similarity measure for discrete time series data. Other than for example euclidian distance, which computes the similarity of two signals by linear matching of the signals indices, DTW first aligns the signals optimally by non-linear distortion of both signals and calculates a distance measure on the so aligned signals. This leads to much higher accuracy than other distance measures and is often used for pattern detection in time series data [82]. Consider the two example signals $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$ with

$$\boldsymbol{x} = [1\ 1\ 1\ 2\ 2\ 3\ 3\ 2\ 1\ 1] \tag{6.2}$$

and

$$\boldsymbol{y} = [1\ 1\ 2\ 2\ 2\ 2\ 3\ 3\ 2\ 1\ 1] \tag{6.3}$$

as shown in fig. 6.3. Both signals have roughly the same outline, though signal $\boldsymbol{y}$ has a length of $n = 11$ and is therefore one sample longer than signal $\boldsymbol{x}$ with length $m = 10$. As can be seen in fig. 6.3a computation of euclidian distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ happens by linear matching of the signal indices. This means each sample x_i of the first signal is matched with the corresponding sample y_i of the second signal. This is illustrated by the black lines between corresponding samples. In this case calculation of the euclidian distance is even impossible, because the signals have different lengths. The last sample of signal $\boldsymbol{y}$ had to be removed to be able to calculate euclidian distance of these two signals. [83]

Different from that DTW matches corresponding samples within both signals so that a cumulative distance measure between both signals is minimzed. This results in the index matching shown in fig. 6.3b. So DTW calculates a distance measure between two signals not sample by sample, but additionally considers an optimal matching between the samples of both signals for calculation of the distance measure. For instance in the two example signals the first two samples of signal $\boldsymbol{x}$ are matched up with the first sample of signal $\boldsymbol{y}$ this means, the distance measure is calculated first for the pair of samples (1,1) and afterwards for the pair of samples (2,1). The optimal matching of both signals is stored in the so called *warping path* $\boldsymbol{w}$. In this example the warping path would be $\boldsymbol{w} = \{(1,1), (2,1), (3,2), (4,3), (4,4), (4,5), (5,6), (6,7), (7,8), (8,9), (9,10), (10,11)\}$. Here the first value of each pair contains the sample index of signal $\boldsymbol{x}$ and the second value the sample index of signal $\boldsymbol{y}$. [83]

An illustration of the warping path can also be found in the distance matrix in fig. 6.4 as yellow path through the matrix. However before the warping path can be determined, the distance matrix $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ has to be calculated as follows

$$d_{i,j} = (x_i - y_j)^2, \tag{6.4}$$

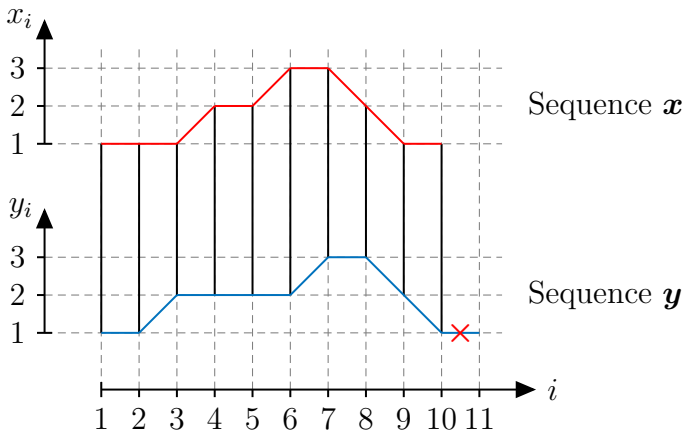
where x_i and y_j are the ith and jth samples in the signals $\boldsymbol{x}$ and $\boldsymbol{y}$. The DTW algorithm now searches for the path $\boldsymbol{w}$ through the distance matrix that minimzes the warping cost

$$DTW(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{k=1}^{K} \boldsymbol{D}\,(w_k)}. \tag{6.5}$$

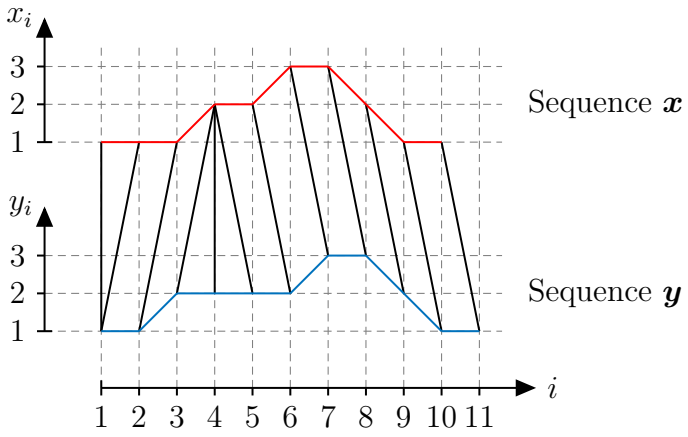
Here k describes all tuples $w_k = (i,j)$ of the warping path $\boldsymbol{w}$, where $\max\{m,n\} < K < m+n-1$ is satisfied. The warping path has to start at the lower left corner of the matrix and end at the upper right corner, so it must be $w_1 = (1,1)$ and $w_K = (m,n)$. Morover the indices i and j must monotonically increase by rather 0 or 1. This means the warping path can only move to the top, to the right or to the left within the distance matrix. Determination of the warping path is usually done via dynamic programming. Hereby the cumulative distance

$$\gamma_{i,j} = d_{i,j} + \min\{\gamma_{i-1,j-1}, \gamma_{i-1,j}, \gamma_{i,j-1}\} \tag{6.6}$$

of the current distance $d_{i,j}$ of the warping path within the distance matrix $\boldsymbol{D}$ and the three possible adjacent elements is evaluated. The path proceeds in the direction, which leads to the smallest increase of the cumulative distance giving the warping path with minimum cumulative distance. [83, 84]



(a) Euclidian distance between two time series. This does not allow for signals of different length.



(b) DTW distance between two time series.

Figure 6.3: Calculation of euclidian distance and DTW distance of two time series. DTW calculates a distance measure between the two signals providing additional optimal alignment of the series. Euclidian distance on the opposite compares the series index by index and is less suitable for pattern recognition applications.

The optimal warping path found by the DTW algorithm can deviate significantly from the diagonal in the distance matrix. This is shown in fig. 6.5a. Here the maximum deviation of the warping path from the matrix diagonal is 7 samples. So the signals, which have in this case both the same length of $n = m = 16$ samples are warped by an amount of 43.75 %. This high distortion of the signals can lead to overfitting of the warping path to outliers in the signal. To prevent the warping path from deviating too far away from the matrix diagonal different kind of global boundary conditions are suggested in the literature [85, 86]. Figure 6.5b shows for instance a Sakoe-Chiba band of with $b = 2$ samples. The warping path can only lie within the dark grey area between the limits parallel to the matrix diagonal. This means any sample x_i of the first signal can only be matched to neighbouring samples within a range of 2 samples, thus samples y_j with $j = i - 2, i - 1, i, i + 1, i + 2$. Further warping of the signal is not possible. This leads to a cumulative distance larger than the optimum value, which is found without

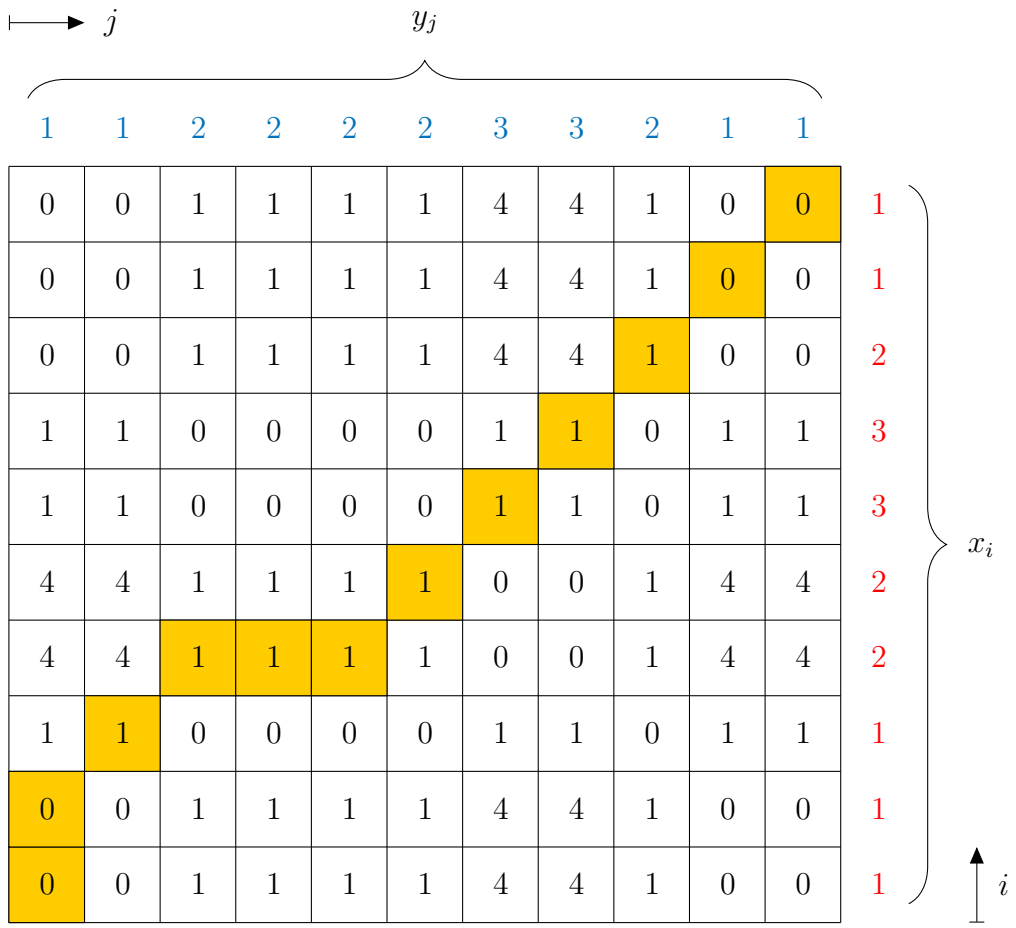
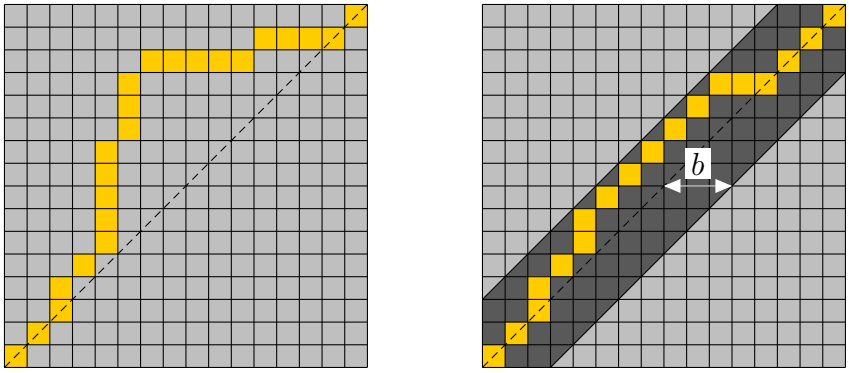


Figure 6.4: Calculated distance matrix between two different time series $\boldsymbol{x}$ and $\boldsymbol{y}$. The calculated warping path $\boldsymbol{w}$, which results in lowest cumulative distance is marked yellow.

global boundary conditions, however prevents excessive warping of the signals. In practice often a width b of 10 % of the signal length is chosen, though in [87] smaller warping windows of 4 % in average lead to highest accuracy. Apart from the Sakoe-Chiba band other global boundary conditions like the Itakura parallelogram have been developed. [86–89]

Though DTW provides a very accurate similarity measure for timeseries classification, even on distorted time sequences, it has the major drawback of high computational cost due to its high time complexity of $\mathcal{O}(n^2)$ [87]. To speed up DTW several lower bounding methods have been introduced to speed up DTW query operations on large datasets [85, 86]. Before lower bounding can be applied, all sequences must be reinterpolated to the same length and a global constraint on the warping window, like the described Sakoe-Chiba band must be applied. Aim of lower bounding is to search through the available set of candidate time sequences not by calculating DTW distance, but instead calculating another distance measure with linear time complexity, like the euclidian distance. Only if this distance is smaller than a specific threshold, the DTW distance of this candidate sequence with the query sequence is calculated. By choosing an appropiate threshold for the lower bouding function, the amount of sequences, which need to be fed into the DTW calculation can be reduced. Listing 6.1 shows an example algorithm



(a) Warping path without bounding conditions.

(b) Warping path with additional Sakoe-Chiba band with width b.

Figure 6.5: Effect of bounding conditions on the warping path. Without any bounding condition the warping cost is minimal, but the warping path can be far away from the matrix diagonal. With an additional linear bounding condition the maximum distance of the bounding path from the matrix diagonal can be limited. This does not result in minimum distance between warped signals, but gives less warped and distorted signals.

that speeds up a sequential search by utilizing a lower bouding function to preselect suitable sequences for the DTW distance calculation. [85]

Another approach to speed up DTW is presented in [90]. Here instruction set extensions for an application-specific embedded processor for calculation of the DTW distance between two signals are developed. To increase performance and reduce energy consumption of the processor logarithmic arithmetic is used instead of floating-point arithmetic.

Beneath the described original DTW algorithm, further modified versions of DTW, like *Derivative Dynamic Time Warping* (DDTW) and *Hybrid Dynamic Time Warping* (HDTW) have been

Listing 6.1: Pseudocode for a sequential scan search algorithm of query time series Q. The search is sped up by utilizing a lower bouding function to preselect suitable sequences. [85]

```
best_so_far = infinity;
for all sequences in database
        LB_dist = lower_bound_distance(Ci,Q);
        if LB_dist < best_so_far
                true_dist = DTW(Ci,Q);
                if true_dist < best_so_far
                        best_so_far = true_dist;
                        index_of_best_match = i;
                endif
        endif
endfor
```

developed. DDTW caclulates another distance matrix than DTW. Here the distance measure is not based on the amplitude values of the two signals $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$, but instead the first order derivatives $\boldsymbol{x'}$ and $\boldsymbol{y'}$ of both signals are used. So the elements of the distance matrix $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ are calculated as follows

$$d_{i,j} = (x'_i - y'_j)^2, \tag{6.7}$$

where x'_i and y'_i are the ith and jth elements of the first order derivatives of the sequences $\boldsymbol{x}$ and $\boldsymbol{y}$. Because of the distance measure based on the first order derivative DDTW only considers the slope of both sequences instead of amplitude values when matching up corresponding samples. In some cases DDTW leads to higher accuracy in signal aligment than normal DTW. For example DDTW performs well on signals with offsets and low frequency noise. Problems arise from high susceptibility to high frequency noise. [83]

HDTW overcomes the drawback of attentiveness of DTW and DDTW to only one shape information. DTW is only susceptible to the amplitude value of both signals and DDTW only to the slope of the signals. Therefore both methods capture only a small part of shape information of the signal. However HDTW considers both amplitude values and slope of the signals when matching up data samples of the sequences. To achieve this the distance matrix $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ is computed as follows

$$d_{i,j} = \left(\frac{d^{(0)}_{i,j} - \mu_0}{\sigma_0} \right) + \left(\frac{d^{(1)}_{i,j} - \mu_1}{\sigma_1} \right), \tag{6.8}$$

where $d^{(0)}_{i,j}$ is the distance computed from the amplitude values and $d^{(1)}_{i,j}$ the distance computed from the signal slopes. Moreover μ_0 and μ_1 are the means of the distances $d^{(0)}_{i,j}$ and $d^{(1)}_{i,j}$ and σ_0 as well as σ_1 the according standard deviations. This normalization is necessary due to the different scaling of the distances of signal amplitudes and slopes. [83]

6.1.3 Cross-Correlation

Cross-correlation is a measure used for determining similarity of two time sequences. It is often used to find a known subsequence within a larger sequence and is therefore a useful tool for the problem of recognizing patterns in a measurement time sequence. Cross-correlation of two continous functions $f(t)$ and $g(t)$ with $f \neq g$ is defined as the integral

$$R_{fg}(\tau) = (f \star g)(\tau) = \int_{-\infty}^{\infty} f^*(t) g(t + \tau)\, dt \tag{6.9}$$

where the star stands for the complex conjugate of the function $f(t)$ and τ means the lag or displacement on the time axis between both functions.

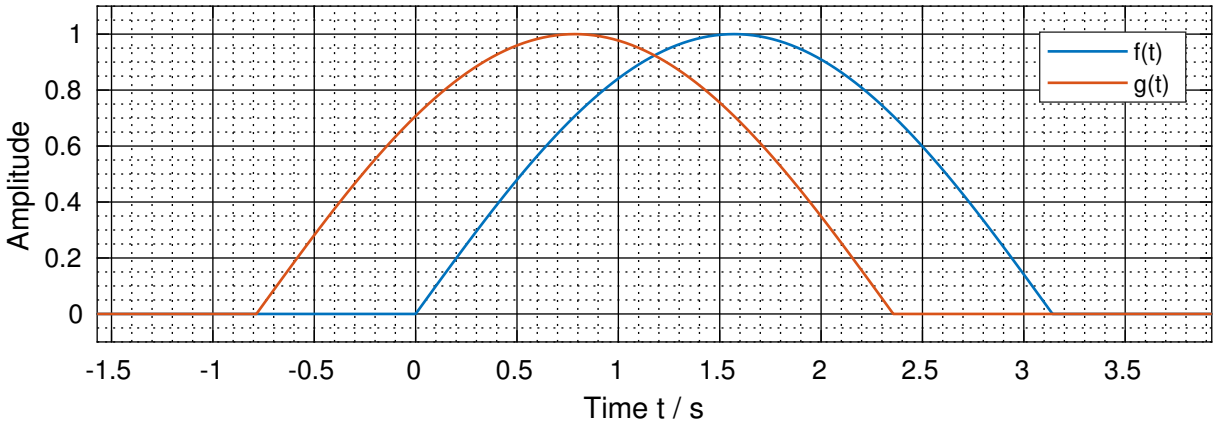
Consider the two continous functions

$$f(t) = \begin{cases} \sin(t), & -\frac{pi}{4} \leq t \leq \frac{3\pi}{4} \\ 0, & \text{otherwise} \end{cases} \tag{6.10}$$

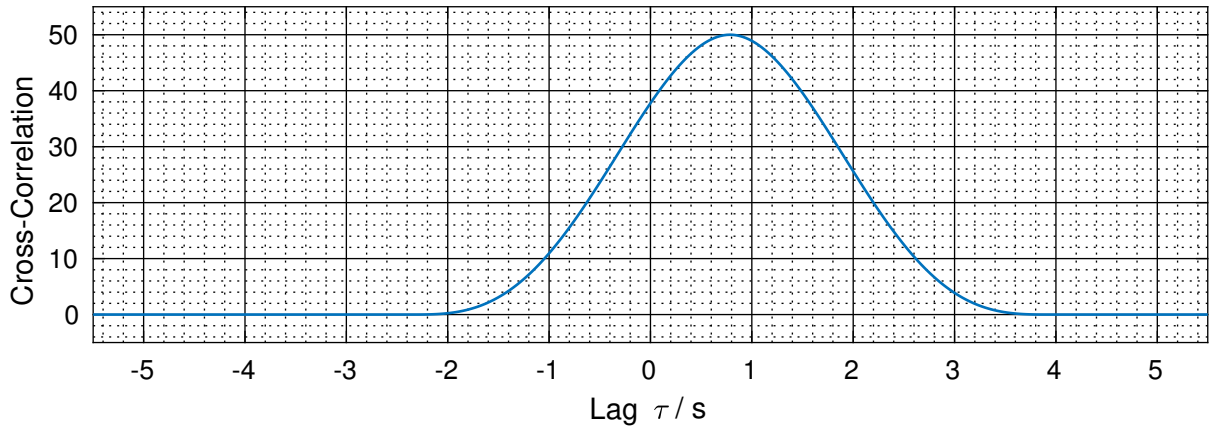
and

$$g(t) = \begin{cases} \sin\left(t + \frac{\pi}{4}\right), & 0 \leq t \leq \pi \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

which are plotted in fig. 6.6a for the time range $-\frac{\pi}{2} \leq t \leq \frac{5\pi}{4}$. As can be seen these functions correspond to half sine waves, where $f(t)$ starts in the coordinate origin and $g(t)$ is phase shifted by $\tau = \frac{\pi}{4}$ to the left. The corresponding cross-correlation function $R_{fg}(\tau)$ as function of the lag τ is depicted in fig. 6.6b. As can be seen the value of the cross-correlation function reaches its maximum at lag $\tau = \frac{\pi}{4}$, which means both functions $f(t)$ and $g(t)$ are most similar, when $g(t)$ is shifted to the right by $\tau = \frac{\pi}{4}$. In this case most similar means the functions are overlapping and therefore are completely identical.



(a) Continuous example functions $f(t)$ and $g(t)$ for which the cross-correlation is computed.



(b) Cross-correlation function $R_{fg}(\tau)$ for the two example functions $f(t)$ and $g(t)$.

Figure 6.6: Example showing the cross-correlation function $R_{fg}(\tau)$ of the two continuous functions $f(t)$ and $g(t)$. The cross-correlation function reaches its maximum value for a lag of $\tau = \frac{\pi}{4}$ s, which means that both functions are most similar when $g(t)$ is shifted right by exactly this lag.

Furthermore it is possible to define cross-correlation for two time discrete functions $f(m)$ and

$g(m)$ by the infinite sum

$$R_{fg}(n) = (f \star g)(n) = \sum_{m=-\infty}^{\infty} f^*(m)g(m+n) \tag{6.12}$$

with discrete timestep m and lag n between both functions.

In terms of a stochastic process cross-correlation has a slightly different meaning. Here cross-correlation is a measure for the linear correlation of two stochastic variables A and B with $A \neq B$. It can be expressed by the *Pearson correlation coefficient*

$$\rho(A,B) = \frac{\mathrm{cov}(A,B)}{\sigma_A \sigma_B} \tag{6.13}$$

with standard deviations σ_A and σ_B and covariance

$$\mathrm{cov}(A,B) = E\left[(A - \mu_A)\,(B - \mu_B)\right] \tag{6.14}$$

of both stochastic variables A and B. μ_A and μ_B are mean values of those variables and $E[*]$ is the expected value of the distribution. Covariance can be expressed differently for time discrete measurement data as it is acquired in the present case. This data can be expressed as random vector $A = (A_1, A_2, \ldots, A_i, \ldots, A_N)^\intercal$ containing N elements which take on specificely distributed values. In this case covariance can be written as

$$\mathrm{cov}(A,B) = \frac{1}{N-1} \sum_{i=1}^{N} (A_i - \mu_A)^* (B_i - \mu_B) \tag{6.15}$$

where the star again marks the complex conjugate of the random variable. Because here A and B are real-valued random vectors considering the complex conjugate has no effect on the covariance. μ_A and μ_B are again mean values of the random vectors A and B and are defined as

$$\mu_A = \frac{1}{N} \sum_{i=1}^{N} A_i. \tag{6.16}$$

The Pearson correlation coefficient is similar to the covariance of the random variables, but is normalized and therefore takes on values between -1 and 1, were 1 means positive linear correlation and -1 negative linear correlation. If the correlation coefficient is 0 there is no linear correlation between both variables. Speaking of cross-correlation in the statistical meaning refers to the Pearson correlation coefficient of two different variables $A \neq B$, whereas auto-correlation refers to the correlation coefficient of one variables at two different points $t_1 \neq t_2$ in time $A(t_1)$ and $A(t_2)$. In this case autocorrelation can be computed according to eq. (6.13) with subtitution $A = A(t_1)$ and $B = A(t_2)$. Computation of the Pearson correlation coefficient for the same variable $\rho(A,A)$ however gives a correlation of 1 as the two time points t_1 and t_2 are identical for calculation of the Pearson correlation coefficient.

Utilizing the Matlab function *corrcoef* for two random vectors A and B gives the correlation coefficient matrix

$$R = \begin{bmatrix} \rho(A,A) & \rho(A,B) \\ \rho(B,A) & \rho(B,B) \end{bmatrix} \tag{6.17}$$

which contains cross-correlation and auto-correlation for both variables. Here $\rho(A,B) = \rho(B,A)$ and as already mentioned $\rho(A,A) = \rho(B,B) = 1$ apply. Therefore it is sufficient to only look at $\rho(A,B)$ or $\rho(B,A)$ for determining correlation of the two random vectors A and B. As these two vectors represent test sequence and class template the Pearson correlation coefficient is a measure for the linear correlation between test sequence and class template. Whenever this correlation is very high, the current test sequence and the class template are very similar to each other and the class of the matched template can be assigned to the test sequence. [91–96]

6.2 Template Creation via Accurate Shape Averaging

In the following section creation of averaged time sequences shall be conducted. This process step is euqivalent to training of the time series classifier which is developed in the subsequent section. To understand the need for template averaging the results of automatic segmentation and labelling in section 5.3.4 are regarded. There measurement data of the second and third test cycle run is segmented into smaller segments representing one of the 12 base classes as defined in section 4.3. These segments are used as training samples for the classifier. As mentioned earlier goal is to predict individual classes of the time segments within the data of the first test cycle run. In the following classification process a template matching classifier will be used. Because a template matching classifier is a very simple parameter-free model, it is not possible to tune any parameters based on training data. This means, no dedicated training step can be conducted on the training data. Instead during classification it is necessary to compare every sample of the test dataset with every segment of the training set belonging to a specific class. For instance, if a given test sample shall be classified, first a comparison with each of the 200 training samples of class 0 is conducted. Afterwards a comparison with the 200 training samples of class 1 is performed and so on. This results in $200 \cdot 12 = 2400$ samples, which need to be compared to the test sample. For all 2201 samples in the test dataset this would lead to a number of $2201 \cdot 2400 = 5282400$ comparative operations. Comparison in this case means calculation of the DTW distance between the test sample and each training sample. As mentioned in section 6.1.2 DTW is computationally expensive, which is why this classification approach is pratically not feasable as calculation time would be too high. Instead prior to classification averaging of all samples in the training dataset belonging to a specific class is conducted, so that only 12 representative samples, so called *templates* remain. Each template is an averaged time sequence representing all training samples of the training dataset. So for each sample in the test dataset a comparison with only 12 templates instead of 2400 training samples is needed. This sums up to $2201 \cdot 12 = 26.412$ necessary DTW distance caulculations needed for classifying the whole test dataset. This is only 0.5 % of the prior aount of needed DTW operations. So averaging of training samples to representative class templates leads to a large benefit in computational expense of the later classification task. Of course averaging itself is computationally complex and needs to be performed additionally, though this step has to be conducted only once on the training data. After averaging is finished, templates can be stored in for example CSV files and are this way easily reusable for the later classification task. This approach of reducing computational expenses of the classification task is well known as *numersity reduction* in literature [97–99].

As creation of class templates from the training dataset s such an important step in classifying patterns within time series data, this section is completely dedicated to the averaging process. First an overview of different methods for time series averging described in literature is given. After a brief description of the foundations of time series averaging, the *Accurate Shape Averaging* (ASA) technique proposed in [83], which is one possible method of averaging two discrete time sequences under retention of shape information via application of DTW, is presented. Afterwards ASA is implemented in Matlab and wrapped with additional functionality to process the complete test dataset in a single execution and to create averaged templates for each of the 12 base classes. This algorithm is presented in detail in the last section.

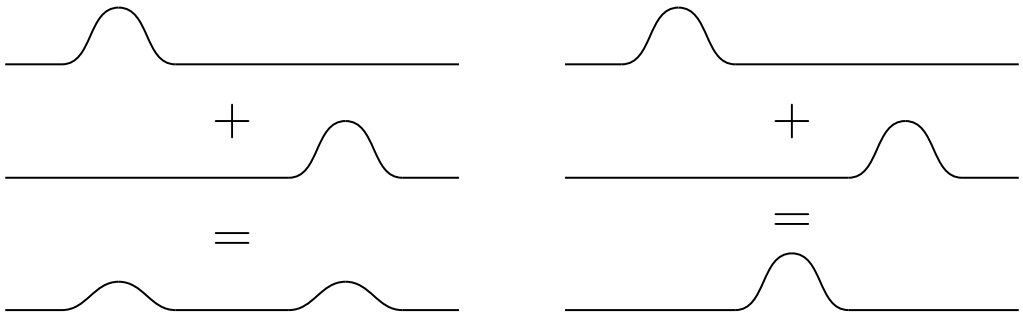
6.2.1 Overview Over Existing Shape Averaging Methods

Time series averaging is a very common problem in signal processing and therefore multiple different solutions to the problem have been proposed in the recent years. General shape averaging algorithms beyond the context of machine learning can be found in [100–103]. Here different methods for non-linear aligment and averging of multiple time series are proposed. Further research in application of DTW for template averging can be found in [104, 105]. Well established methods for time series averaging are for example *Prioritized Shape Averaging* (PSA) [106], *Random Sampling* [107], *Ranking* [108] and *Adaptive Warping Window* (AWARD) [109]. A derivate of the DTW algorithm called *Scaled Dynamic Time Warping* (SDTW) is introduced in [106] as well. This algorithm provides averging of two sequences with additional weighting of both sequences. Further improvements of the DTW algorithm for shape averaging called *Cubic-Spline Dynamic Time Warping* (CDTW) and *Iterative Cubic-Spline Dynamic Time Warping* (ICDTW) are proposed in [110]. Here not only the averaged amplitude, but also the proper placement of data points within the averaged sequence is considered. Another very powerful time series averaging method is *Dynamic Time Warping Barycenter Averaging* (DBA) introduced in [111]. This approach computes the average of all time sequences in one single step instead of computing the pairwise average of two sequences in each iteration of a loop. DBA is the newest algorithm for sequence averaging and outperforms every existing algorithm, though it is more complex to implement than other methods like ASA.

6.2.2 Foundations of Shape Averaging

At this point a short introduction to shape averaging shall be given, because shape averaging is different from usual amplitude averaging of signals. This is shown in fig. 6.7. Here two time sequences containing the same shape feature, but at two different temporal positions, shall be merged into one averaged sequence. If amplitude averaging is performed, like in fig. 6.7a, the merged sequence is constructed by calculating the mean of both original sequences sample-by-sample without considering any information about the shape feature and its temporal position. Shape averaging by means of DTW is different from that by first non-linear aligning both signal based on the contained shape information and then calculating the mean for both aligned sequences. Result is a sequence, containing an averaged version of the original shape feature that was contained in both original sequences. Consequently this form of averaging is the desired one

for the present problem of class template creation by averging of multiple training sequences. If shape averaging is applied to all training sequences it is possible to retrieve only one final averaged sequence containing all shape information of the training sequences neglecting any phase shifts or misalignments between the training sequences.



(a) Arithmetic averaging of two sequences.

(b) Shape averaging of two sequences.

Figure 6.7: Difference between arithmetic averaging and shape averaging of two sequences. Only shape averaging maintains the shape information of the original sequences, while this information is lost in general when arithmetic averaging is conducted. [83]

Before the ASA algorithm for shape averaging of training sequences is explained in detail, it is helpful to understand how the merging process of multiple different sequences is performed in general. This pairwise sequence averaging is illustrated in fig. 6.8. Consider the initial vector

$$\boldsymbol{t} = (\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3, \ldots, \boldsymbol{t}_r) \tag{6.18}$$

containing all r training sequences $\boldsymbol{t}_1, \boldsymbol{t}_2, \boldsymbol{t}_3, \ldots, \boldsymbol{t}_r$. All of these training sequences belong to the same base class $\{0 \ldots 11\}$ and therefore represent this class. In this case r equals 100 as for each class 100 training segments exist within the second test cycle run, which is chosen as basis for creation of the averaged class templates. Though there are twice as much training segments in the acquired training data, in the following training segments from the third test cycle run are not considered. This is due to the high computational complexity of the DTW averaging, which results in long computation time. In order to reduce the time needed for averaging, only the training segments from the second test cycle run are used for template creation. To create a single representative averaged class template from these $r = 100$ different sequences, pairwise avergaging needs to be conducted as the ASA algorithm can only handle two sequences at a time. Therefore, iteratively two sequences, say sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$, are picked from the sequence vector $\boldsymbol{t}$ and averaged by means of DTW. Result is the averaged sequence $\boldsymbol{t}_{12}$. In the same iteration both original sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are removed from the sequence vector and the averaged sequence is inserted into the vector. So in each iteration the length r of the sequence vector $\boldsymbol{t}$ decreases by 1. If this process is repeated $r - 1$ times, only one averaged sequence is left. This sequence contains shape information of all the different training segements initially stored in the sequence vector $\boldsymbol{t}$. Which sequences are merged in each iteration can be chosen in different ways. First it is possible to just merge the first and second segment each time, second

the two sequences can be chosen randomly and third segments can be chosen based on some criteria. In this case the similarity of each pair of sequences is calculated and then the most similar sequences are averaged in each iteration. This will be explained in more detail below.

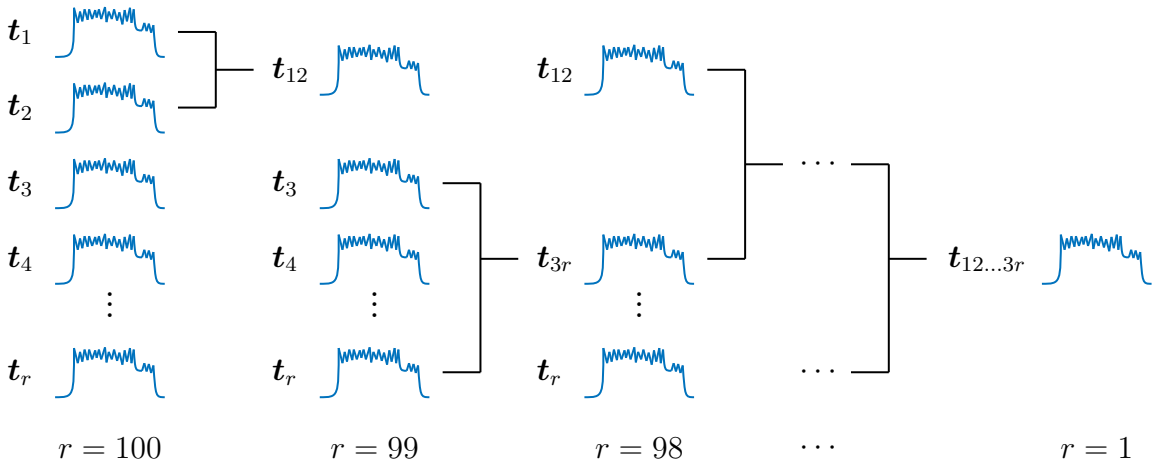


Figure 6.8: Schematic showing pairwise averaging of two sequences in each iteration. As the ASA algorithm can only perform averaging of two sequences at a time, sequencing of averaging operation is necessary. In each iteration the two most similar sequences are identified and averaged by means of DTW. Both original sequences are removed from the sequence vector and the averaged sequence is inserted. If initial number of sequences is r the algorithm terminates after $r - 1$ iterations when only one averaged sequence is left.

6.2.3 Operational Principle of the ASA Algorithm

Now that the general operating principle of pairwise averaging is understood, the ASA algorithm is explained in more detail. This algorithm, first introduced in [83] comprises four major steps, which are

1) Localizing pair of most similar sequences,
2) Alignment of both sequences by means of DTW,
3) Calculation of the shape averaged sequence,
4) Resampling of the averaged sequence.

First these four steps are examined in a more general manner and afterwards the actual implementation of the ASA algorithm including any additional functionality for iterative pairwise averaging and class template creation for every 12 base classes is presented. It is important to understand that the following four steps are conducted every time two sequences are averaged.

6.2.3.1 Localizing Pair of Most Similar Sequences

First step in calculating an average of two sequences is localization of the two most similar sequences within the whole sequence vector $\boldsymbol{t}$. Calculating the average of the two most similar

sequences results in higher accuracy of the averaging result than choosing the sequences randomly as the most similar sequences need to be time warped by the smallest possible amount. To find the most similar pair of sequences a distance matrix $\boldsymbol{D} \in \mathbb{R}^{r \times r}$ containing the DTW distances $d_{i,j}$ between each pair of the r sequences in the sequence vector $\boldsymbol{t}$ is calculated. Result is the following matrix

$$\boldsymbol{D} = \begin{array}{c} \begin{matrix} 1 & 2 & 3 & & j & & r \end{matrix} \\ \begin{bmatrix} d_{1,1} & \boldsymbol{d_{1,2}} & d_{1,3} & \cdots & d_{1,j} & \cdots & d_{1,r} \\ 0 & d_{2,2} & d_{2,3} & \cdots & d_{2,i} & \cdots & d_{2,r} \\ 0 & 0 & d_{3,3} & \cdots & d_{3,j} & \cdots & d_{3,r} \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & d_{i,j} & \cdots & d_{i,r} \\ \vdots & \vdots & \vdots & & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & \cdots & d_{r,r} \end{bmatrix} \end{array} \begin{matrix} 1 \\ 2 \\ 3 \\ \\ i \\ \\ r \end{matrix} . \tag{6.19}$$

Because the distance matrix is symmetrical, which means $d_{i,j} = d_{j,i}$, it is sufficient to calculate only the top right triangular matrix. Every element below the matrix diagonal is set to 0. Obviously the diagonal elements are $d_{i,i} = 0$ as well as the distance of a sequence from itself is 0. Because the most similar sequences need to be found, the minimum non-zero element within the distance matrix $\boldsymbol{D}$ has to be found. To achieve this, every matrix element $d_{i,j} = 0$ is found and replaced by $d_{i,j} = \infty$. Now the minimum element as well as its row and column index is found by means of the Matlab *min* function. In this example the bold printed element $d_{1,2}$ in matrix $\boldsymbol{D}$ is the smallest element. As this element is the distance between the first and second sequence $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ these two sequences are the most similar ones in the current iteration of the algorithm. Because of the high number of necessary DTW distance calculations the step of finding the most similar pair of sequences is very slow. Acceleration is possible by using a lower bounded DTW as described in section 6.1.2. [83]

6.2.3.2 Alignment of Both Sequences by Means of DTW

After the pair of most similar sequences is found, both sequences need to be aligned correctly in order to calculate the shape average in the next step. Alignment is conducted by DTW of both sequences as described in section 6.1.2. Having a look back at fig. 6.3b makes clear, what happens in this stage of the ASA algorithm. The warping path $\boldsymbol{w}$ is determined so that the warping cost is minimized. Result is a non-linear alignment based on the amplitude values of both sequences as illustrated in fig. 6.3b. This steps ensures matching of the charactersitic shape features of the sequences, which is essential for calculating a shape average instead of only an arithmetic average. As the DTW distance is calculated independently from the DTW distance used for determining similarity of the sequences, it is possible to change the distance metric of the DTW function or apply a global constraint like for instance a Sakoe-Chiba band on the warping path. [83]

6.2.3.3 Calculation of the Shape Averaged Sequence

After the sequences are now properly aligned, the actual averaging can be conducted. Both sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are merged and the average sequence $\boldsymbol{t}_{12}$ is calculated. The elements of this sequence are determined by

$$t_{12,k}(\boldsymbol{t}_1, \boldsymbol{t}_2, w_{k,1}, w_{k,2}) = \left(\frac{w_{k,1} + w_{k,2}}{2}, \frac{\boldsymbol{t}_1 \left(w_{k,1}\right) + \boldsymbol{t}_2 \left(w_{k,2}\right)}{2} \right). \tag{6.20}$$

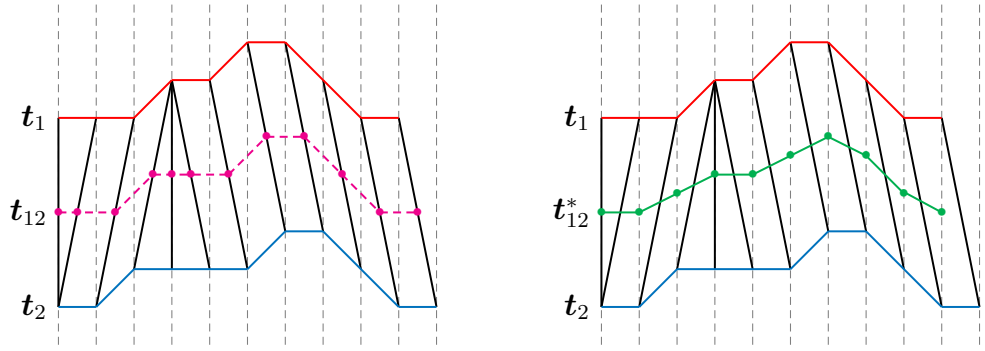
Here $k = 1,2,\ldots,K$ with $\max\{m,n\} < K < m+n-1$. n and m are the number of samples within $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$. This means the length of the averaged sequence $\boldsymbol{t}_{12}$ is equal to the length of the warping path $\boldsymbol{w}$. $w_{k,1}$ and $w_{k,2}$ are the kth tuple $w_k = (w_{k,1},w_{k,2}) = (i,j)$ of the warping path. Consider the example sequences $\boldsymbol{t}_1 \in \mathbb{R}^m$ and $\boldsymbol{t}_2 \in \mathbb{R}^n$ with $\boldsymbol{t}_1 = [1\ 1\ 1\ 2\ 2\ 3\ 3\ 2\ 1\ 1]$ and $\boldsymbol{t}_2 = [1\ 1\ 2\ 2\ 2\ 2\ 3\ 3\ 2\ 1\ 1]$ from section 6.1.2 again. For this sequences the warping path was $\boldsymbol{w} = \{(1,1),(2,1),(3,2),(4,3),(4,4),(4,5),(5,6),(6,7),(7,8),(8,9),(9,10),(10,11)\}$. The resulting averaged sequence $\boldsymbol{t}_{12}$ calculated after the formula above would be $\boldsymbol{t}_{12} = \{(1,1),(1.5,1),(2.5,1),(3.5,2),(4,2),(4.5,2),(5.5,2),(6.5,3),(7.5,3),(8.5,2),(9.5,1),(10.5,1)\}$, which is illustrated in fig. 6.9a as the pink dashed line. [83]

6.2.3.4 Resampling of the Averaged Sequence

Last step of the ASA algorithm is resampling of the averaged sequence. Due to non-linear alingment of both original sequences the averaged sequence contains more data points than the original sequences. For instance the calculated averaged sequence $\boldsymbol{t}_{12}$ contains 12 samples, whereas the length of the original sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ was 10 and 11 respectively. Each time an individual data point of one of the original sequences is matched with more than one data point of the other sequence a new data point in between of those two matched up data points occurs in the averaged sequence. This leads to additional data points in the averaged sequence, which might be not properly aligned with the sampling grid of the original series. The problem becomes even worse, when averaging is repeated multiple time as happens in this case. On each averaging iteration new data points are inserted in between of the data points of the original sequences. So in each averaging iteration the averaged sequence becomes longer. To overcome this issue, resampling by means of cubic spline interpolation is conducted after each averaging. In the implementation the Matlab function *spline* is used to calculate an interpolated sequence. Interpolation reduces the number of data points to the number of samples in the shorter original sequence. In this case the number of data points is reduced to 10. Moreover the data points in the interpolated sequence are exactly aligned to the sampling grid of the original sequence. This can be seen in fig. 6.9b, where the green line represents the cubic spline interpolated averaged sequence $\boldsymbol{t}_{12}^*$. [83, 112]

6.2.4 Description of the Template Creation Algorithm

After the functional principle of the ASA algorithm for shape averaging of sequences is now fully understood the overall algorithm for iterative pairwise averaging and template creation for all 12



(a) Accurate shape averaged sequence of two discrete signals.

(b) Cubic spline resampled averaged sequence.

Figure 6.9: Accurate shape averaged sequence (pink) of two original signals $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ and same sequence after cubic spline interpolation (green). The averaged output sequence contains more samples than both of the original sequences, which is why cubic resampling is needed to reduce the number of samples of the averages sequence to the number of samples within the original signals. [83]

base classes is presented in the following section. The full Matlab sourcecode of this algorithm can be found in listing A.8. A schematic of the functional principle of the whole algorithm not only consisting of the ASA algorithm, but also including loading of labeled training dataset and iterative pairwise averaging of two sequences as described in section 6.2.2 is shown in fig. 6.10.

First the labeled measurement data from the second test cycle run is loaded. As mentioned in section 6.2.2 it is sufficient to create the class templates based only on the training samples within the measurement data from the second test cycle run. Using the samples from the third test cycle run as well would not lead to much improvement of the averaging result as there are already 100 training sequences per class. Instead computation time would increase drastically, as the number of DTW operations per iteration needed for determination of the pair of most similar sequences scales with the order $\mathcal{O}(n^2)$.

After loading of the labeled dataset the algorithm enters the first loop, which iterates over the 12 different base classes. Result of each iteration of this loop is an averaged class template for the corresponding base class. To achieve this first all training segments belonging to the current base class are retrieved and stored in the sequence vector $\boldsymbol{t}$. A special case is the idle class 0, because the training set does not contain only 100 segments of this type, but 1101. So in order to reduce the number of segments of class type 0, only the first 100 segments occuring in the labeled training set are retrieved. The first and last segments as well as the segment in between of the first two subcycles are discarded as they are longer than the usual 2 s long idle phases in between of each axis movement. Another problem occuring with the idle class 0 is length of the training segments, which varies for each segment. For each other base class segments have the same lengths as the end cut is placed after a constant amount of samples (see section 5.3.1). But for the idle class the endpoint is the next starting point and this position flucutates by some samples. In order to create a proper class template for the idle class, it is necessary to bring all training segments to the same length. To do this the length of each training segment of classs

0 is determined and the minimum is calculated. Then every training segment of this class is shortened to this minimum length.

Now that all preprocessing is done the actual pairwise shape averaging as already described in section 6.2.2 is conducted. As it is necessary to calculate the shape average pairwise another loop is needed that iterates over all $r = 100$ training segments that are stored in the sequence vector $\boldsymbol{t}$ for the current base class. As length r of the sequence vector $\boldsymbol{t}$ decreases in each iteration of this loop by 1, the loop is run as long as the length of the sequence vector is larger than 1. If it is 1 the loop terminates and the sequence vector only contains the final averaged sequence for the current base class. On execution of this loop first the triangular distance matrix $\boldsymbol{D}$ as described in section 6.2.3.1 is calculated to determine which two sequences within the sequence vector are most similar. Then the minimum non-zero element and its position within the distance matrix is retrieved. Row and column indices of this element provide the indices of the two most similar sequences in the sequence vector $\boldsymbol{t}$.

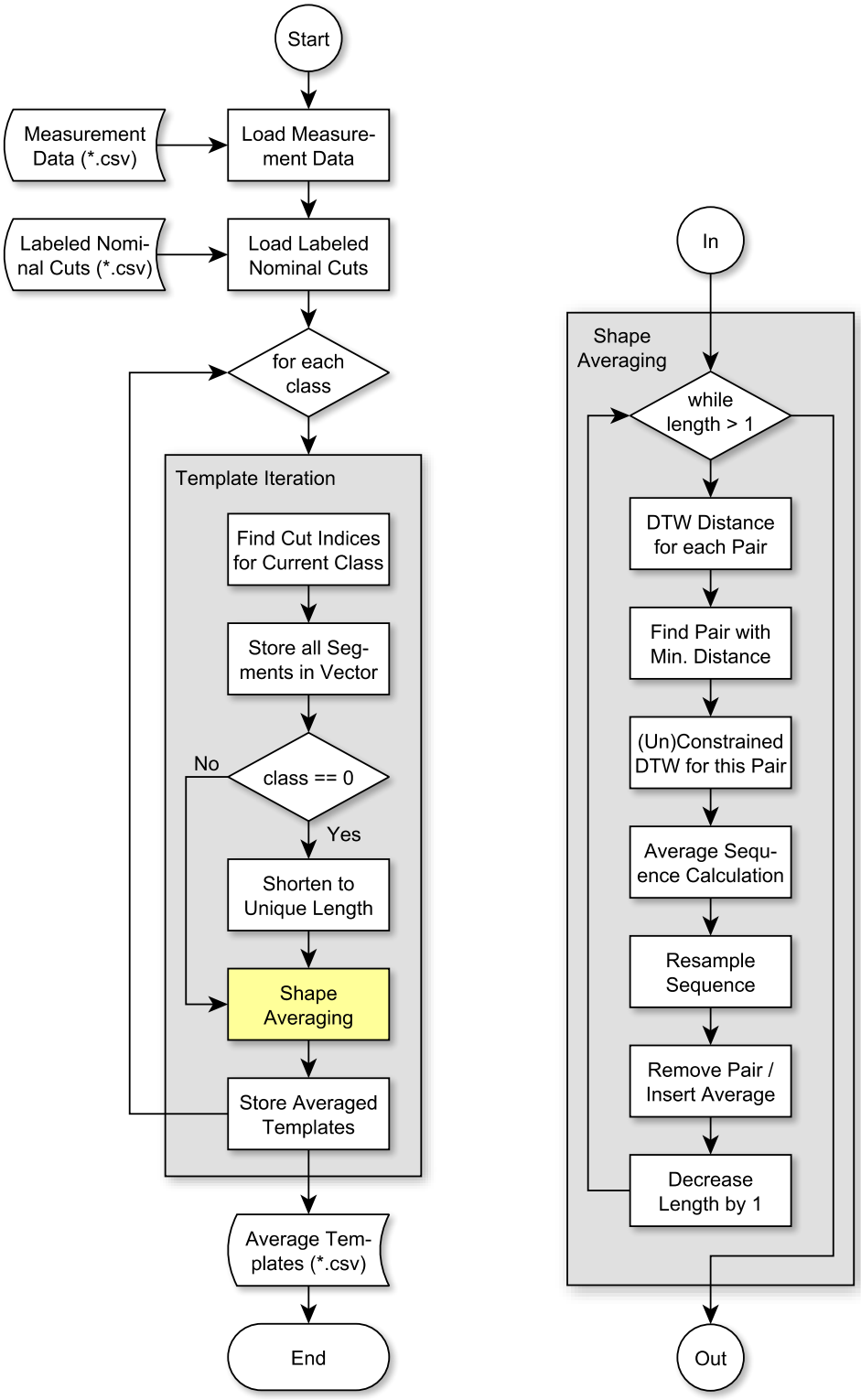
After the two most similar sequences have been found according to section 6.2.3.2 another DTW is performed to non-linearly align both sequences based on their amplitude values. At this point it is possible to select, whether the warping path shall be unconstrained as in fig. 6.5a or a Sakoe-Chiba band as in fig. 6.5b shall be applied to restrict the deviation of the warping path from the matrix diagonal in the cumulative distance matrix calculated during the DTW (see fig. 6.4). In this case the width of the Sakoe-Chiba band is set to $b = 10$ samples.

Now shape averaging of the aligned sequences as described in section 6.2.3.3 needs to be performed. The retrieved sequence is stored and afterwards resampled as explained in section 6.2.3.4. This is done by means of the Matlab *spline* function [91], which takes the averaged sequence as well as a vector containing the query points at which the amplitude value of the sequence shall be evaluated, as input arguments. Subsequently both original sequences are removed from the sequence vector $\boldsymbol{t}$ and the averaged and interpolated sequence is inserted instead of the two sequences. The position index within the squence vector $\boldsymbol{t}$ at which the averaged sequence is inserted is given by the row index of the minimum element in the distance matrix. So if for instance sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are identified to be most similar the index values of the minimum element are $(i,j) = (1,2)$. Therefore both sequences $\boldsymbol{t}_1$ and $\boldsymbol{t}_2$ are removed from the sequence vector $\boldsymbol{t}$ and the averaged sequence $\boldsymbol{t}_{12}$ is inserted at index position $i = 1$ in the sequence vector giving the updated sequence vector

$$\boldsymbol{t} = (\boldsymbol{t}_{12}, \boldsymbol{t}_3, \boldsymbol{t}_4 \ldots, \boldsymbol{t}_r) \tag{6.21}$$

with updated length $r = r - 1$. Now the whole process starts all over with the updated sequence vector $\boldsymbol{t}$ until the termination criterion of $r = 1$ is met.

The whole process described above is repeated 12 times to create averaged templates for every base class from all training sequences. Finally the resulting 12 templates are stored to a CSV file and variable values are exported to a Matlab file for later usage. Moreover the resulting templates are visualized amongst all 100 training segments for each of the templates.



(a) Overall structure of the template averaging algorithm.

(b) Subroutine for shape averaging by means of DTW.

Figure 6.10: Schematic of the template averaging algorithm. In each iteration the two most similar sequences are found and merged in an average sequence by means of dynamic time warping. Averaging is repeated until only one final average sequence is left. This is repeated for all 12 different kinds of templates, each corresponding to one of the 12 base classes.

6.2.5 Results of the Template Creation Algorithm

Results of the template creation algorithm can be found in fig. 6.11 and fig. 6.12, which show resulting averaged templates for class 0, 4 and 5 respectively. Here the gray lines correspond to the 100 training sequences within the segmented and labeled measurement data from the second test cycle run. The green line corresponds to the averaged sequence calculated via unconstrained DTW and the red line on the contrary is the averaged template resulting from the DTW contrained with a Sakoe-Chiba band of width $b = 10$. As can be seen escpecially in fig. 6.12 the constrained DTW provides better averaging results, which complies with the results from [87] that state narrow constrained DTW performs in general better than unconstrained DTW. Though the unconstrained DTW achieves minimum distance between each pair of sequences during the averaging step, it leads to overfitting of the averaging result to outliers within the sequence vector. This can be seen in the multiple steps the green line shows. On the opposite the red line as result of a constrained DTW is much smoother and does not show any steps. This behaviour can also be seen in the class templates for all other base classes that can be found in appendix A.5. Only the class templates for classes 8, 9, 10 and 11 differ as here the results from the constrained and the unconstrained DTW are nearly the same. This might be explained by the fact that the training sequences differ much less from another than the training sequences for the other base classes.

Regarding execution time the algorithm needs approximately 1110 s for calculation of all class templates excluding time needed for loading the labeled training data, storing created templates in CSV files and plotting of results. Here it becomes apparent, why the templates were calculated based on only the trainign samples from the second test cycle run instead of all available training segments form second and third test cycle run. As already said, execution time scales with $\mathcal{O}(n^2)$ using all training templates would increase execution time by manifolds. As the averging needs to be performes only once and the averaged templates are then available in form of CSV files for later usage, the long execution time of the averaging is no problem and the algorithm does not need to be further improved. However some improvement could be performed in future work as the distance matrix $\boldsymbol{D}$ in section 6.2.3.1 is recalculated completely in each iteration of the pairwise averaging to find the most similar pair of sequences. As the sequence vector $\boldsymbol{t}$ only changes by the two orginal sequences that are removed and the averaged sequence that is inserted it would be sufficient to only recalculate some of the distances in the distance matrix in each iteration as all other distances are constant. This would drastically reduce the number of necessary DTW distance calculations and speed up execution of the template creation algorithm significantly. As this would only affect execution time of the algorithm, rather than the averaging result this optimization is not performed at this point because as already mentioned template creation needs to be performed only once.

6.3 Development of Shape Based Classifiers

In the following section algorithm and implementation of the template matching classifier are presented. Refering back to the fundamentals of shape based time series classification in section 6.1.1 the classifier algorithm introduced in this section has the overall task to determine

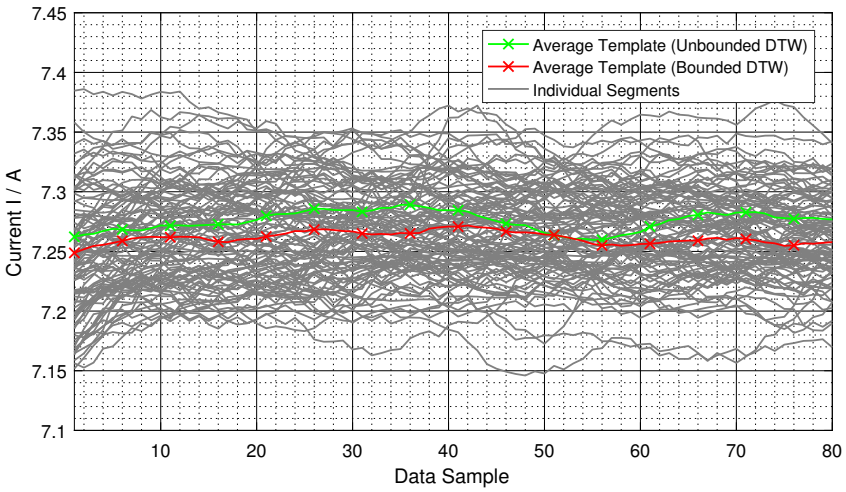
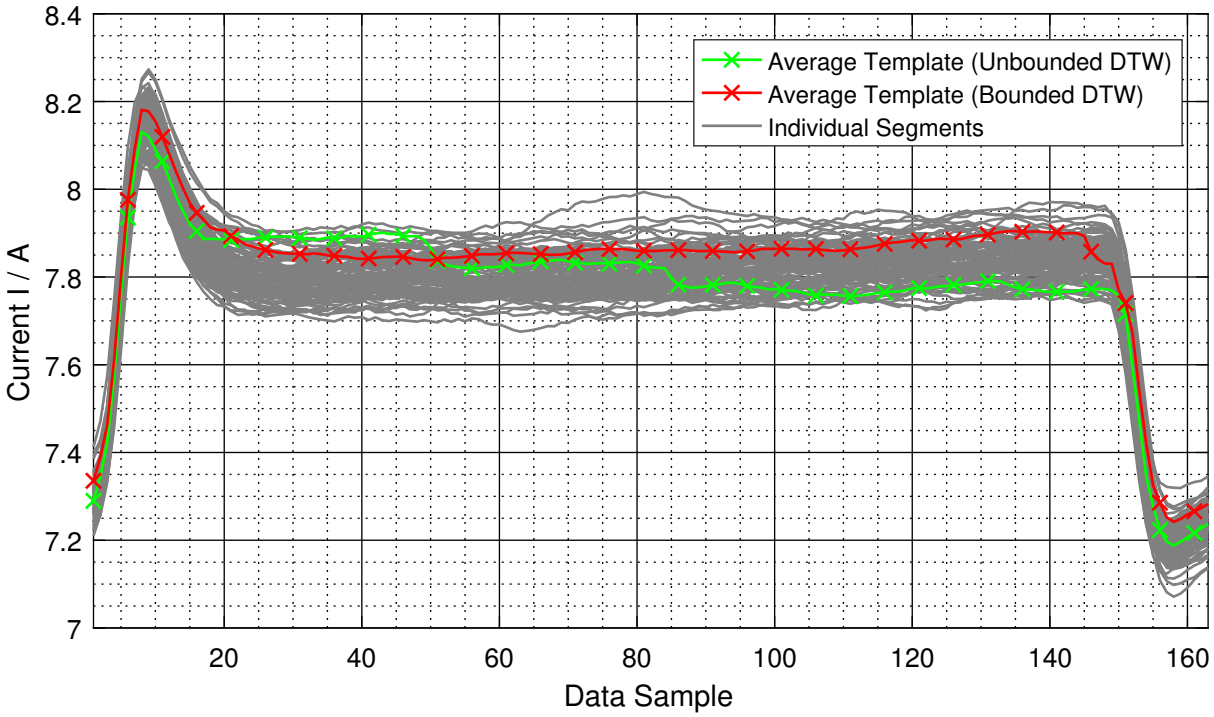


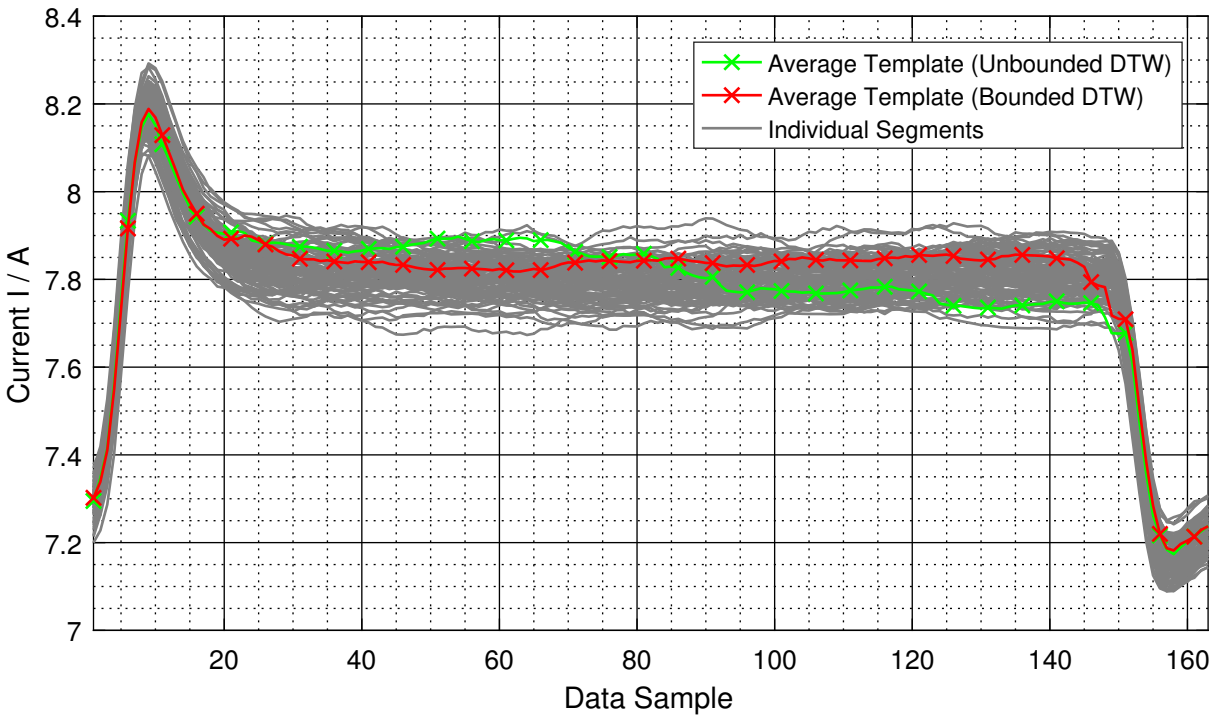
Figure 6.11: Resulting averaged template of the accurate shape averaging algorithm for sequences of class 0, which is the machine idle state. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.

classes of all segments in the previously segmented measurement data which was gathered during the first test cycle run. Prediction of these classes shall be as accurate as possible, this means as many class labels as possible shall be estimated according the true classes of the segments. The proposed algorithm includes two different classifiers. The first classifier is based on DTW distance and the second one calculates cross-correlation coefficients of the test sequence and each template to determine which template is most similar to the segment and assign a class label accordingly. Results of both classifiers are not combined, however performance of both classifiers is measured and compared in the next section. Now the operational principle of the algorithm containing both classifiers is described in detail. The full Matlab sourcecode can be found in listing A.9. A flowchart containing the main process steps of the template matching algorithm is depicted in fig. 6.13. In fig. 6.13a the overall structure of the algorithm is illustrated and fig. 6.13b shows the core part of the algorithm containing both classifiers that perform class prediction on each segment in the test set.

Before any classification can be conducted necessary data needs to be loaded. This includes the averaged class templates created in section 6.2, the measurement data from the first test cycle run which is used as test data and the labelset created in section 5.3 containing only nominal segments. The labelset contains information about segment boundaries within the measurement data, which is important because the classifier shall determine a class label for each individual segment. So it is neccessary to know start and end point of each segment. After this initial information is loaded the algorithm enters the main loop which iterates over all segments within the test set. In this loop first the current segment is retrieved from the measurement data based on the cut positions in the labelset. Now the algorithm needs to calculate distance or similarity measures for the current test segment with each of the class templates to determine which class template is most similar to the sequence. The class of this template is then assigned to the



(a) Averaged template for class 4, the Y-axis forward movement.



(b) Averaged template for class 5, the Y-axis backward movement.

Figure 6.12: Resulting averaged templates of the accurate shape averaging algorithm for sequences of classes 4 and 5, in which the Y-axis moves forward and backward. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.

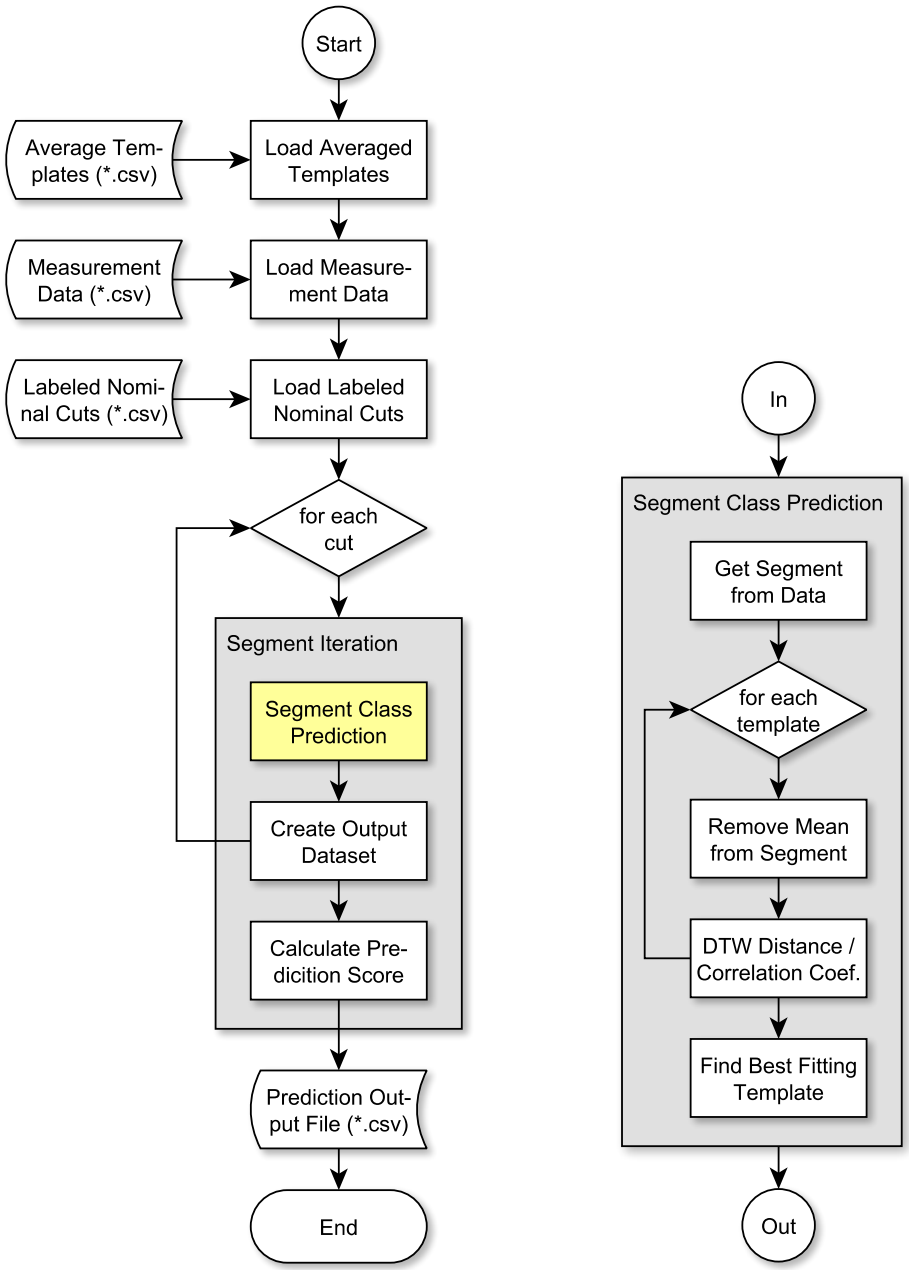
current segment. To caluclate similarity of the test segment with each class template another loop within the main loop iterates over all 12 class templates for each segment. In this loop first the signal mean is removed to prevent the classifiers from being obstructed by any signal offsets which might vary over long time periods, for instance because of the power consumption of the cooling, which is assumed to be constant, however this is not exactly true. After this the actual distance or similarity measure is calculated. First the DTW distance between the current class template and the current segment is calculated and the distance stored in a vector. Moreover the warping path is stored and used to realign template and test segment. For these realigned sequences then the cross correlation coefficient is computed and stored in another vector. After all 12 iterations of the inner loop the DTW distance vector and the correlation coefficient vector contain 12 elements each for all 12 class templates. Now the minimum DTW distance respectively the maximum correlation coefficient is found by means of the Matlab *min* respetively *max* functions and the class label of the corresponding template is retrieved and stored in the ouptut dataset matrix. This matrix has seven columns as shown in table 6.1 which contain an integer number as index for the current segment, the segment startpoint and endpoint, the maximum correlation coefficient and via cross-correlation predicted class, the minimum DTW distance and the via DTW predicted class. This information is stored for every segment within the test set and finally stored in a CSV file. Based on this results moreover the classification accuracy for both classifiers according to eq. (6.1) is calculated and displayed.

Table 6.1: Extract from the output dataset created by the template matching classifier algorithm.

No.	**Cut Index**		**Cross-Correlation Classifier**		**DTW Classifier**	
	Start	**End**	**Max. Coefficient**	**Predicted Class**	**Min. Distance**	**Predicted Class**
1	1	470	0.72419	7	6.2234	0
2	470	1611	0.99972	1	61.311	1
3	1611	1710	0.79537	1	0.62404	0
4	1710	2851	0.99994	1	43.024	1
5	2851	2950	0.8524	8	0.68651	0
6	2950	4091	0.99983	1	57.703	1
⋮	⋮	⋮	⋮	⋮	⋮	⋮
2201	359405	359461	0.9996	11	1.9179	11

As mentioned in section 6.1.1 both classifiers have multiple freely selectable parameters, which need to be chosen properly to achieve optimal prediction accuracy. An overview of all parameters is given in fig. 6.14. As can be seen for both classifiers there are four parameters which can take on multiple different values. For instance during computation of the DTW distance in the classifier algorithm the DTW can use different kinds of distance measures for determining the distance matrix $\boldsymbol{D} \in \mathbb{R}^{m \times n}$ between the signals $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$. According to [91] there are three different distance metrics are available. The euclidean distance

$$d_{i,j} = \sqrt{(x_i - y_j)^2}, \tag{6.22}$$



(a) Overall structure of the template matching classifier algorithm.

(b) Subroutine for calculation of similarity between current segment and each template.

Figure 6.13: Schematic of the template matching classifier algorithm. Each time series segment of the previously segmented measurement data is compared to the 12 averaged class templates and the most similar class is assigned to the segment. This is repeated for every segment within the first test cycle run and afterwards the accuracy score of the classifier is calculated. Similarity of templates and segments is calculated by means of the lowest DTW distance or highest cross correlation coefficients.

the squared distance

$$d_{i,j} = (x_i - y_j)^2 \tag{6.23}$$

and finally the symmetric Kullback-Leibler distance

$$d_{i,j} = (x_i - y_j)(\log x_i - \log y_j). \tag{6.24}$$

Besides the distance measure another parameter that can be varied to change performance of the template matching classifier is restriction of the warping path $\boldsymbol{w}$ by some global constraint like a Sakoe-Chiba band as illustrated in fig. 6.5b. As the width b of this band is an arbitrary positive integer value it is neccessary to choose reasonable limits for the width. In this parameter studies the band width is varied in a range from 1 to 60, which is a sensible choice as the smallest template, which is the one for class 11, contains only 57 samples. So limiting the deviation of the warping path far beyond the number of 57 samples makes no sense as this is identical with not using a constraint on the warping path at least for this class at all. Another classsification run is executed without using a global constraint on the DTW warping path, which is synonymous with an infinite band width $b \to \infty$. Next parameter which can be choosen freely is the kind of averaged class template used for determining class labels. As mentioned in section 6.2.5 two different kinds of averaged templates were created. One utilizing a Sakoe-Chiba band of width 10 for restricting the DTW warping path during the averaging and another one without any restriction of the warping path. Averaged class templates for both variants looked different from each other and were stored in separate CSV files. At this point classification accuracy is compared for each of the two different sets of class templates. The last parameter which is changed in the parameter studies is the removal of the offset from each segment and template. As mentioned earlier in this section removal of the mean makes the classification invariant against varying offsets of the test sequence and therefore affects classification accuracy. To examine this, one half of the classifications is performed with removed offsets of test sequence and template and the other half without removed offset. There is one constraint regarding calculation of the symmetric Kullback-Leibler distance metric. As this metric calculates logarithmic values of the signal amplitudes x_i and y_j it is not possible to calculate this measure for negative amplitude values. This becomes a problem when the mean value is subtracted from the test sequence as well as the template as then negative amplitude values occur. So whenever the distance metric is symmetric Kullback-Leibler distance the classification accuracy is only measured for the test sequence and templates without removal of the mean value. All those parameters apply for both the DTW classifier as well as the classifier based on maximum cross-correlation between test sequence and class templates, because the cross correlation coefficients are calculated for the DTW aligned test segment and class templates.

The described four parameters and their possible values span a parameter grid of $3 \times 60 \times 2 \times 2 = 720$ different classsification tasks for which prediction accuracy has to be calculated each. As one classification involves prediction of classes for all 2201 segments in the test set, evaluation of calculation accuracy of all 720 different combinations of the parameter values is computationally very expensive and takes a long computation time. However this process needs to be carried out only once to find the optimal combination of parameter values. Therefore long computation time is not a major issue.

Evaluation of the parameter grid is conducted via a modified version of the template matching algorithm presented in fig. 6.13. Main difference of this algorithm are the three loops that are

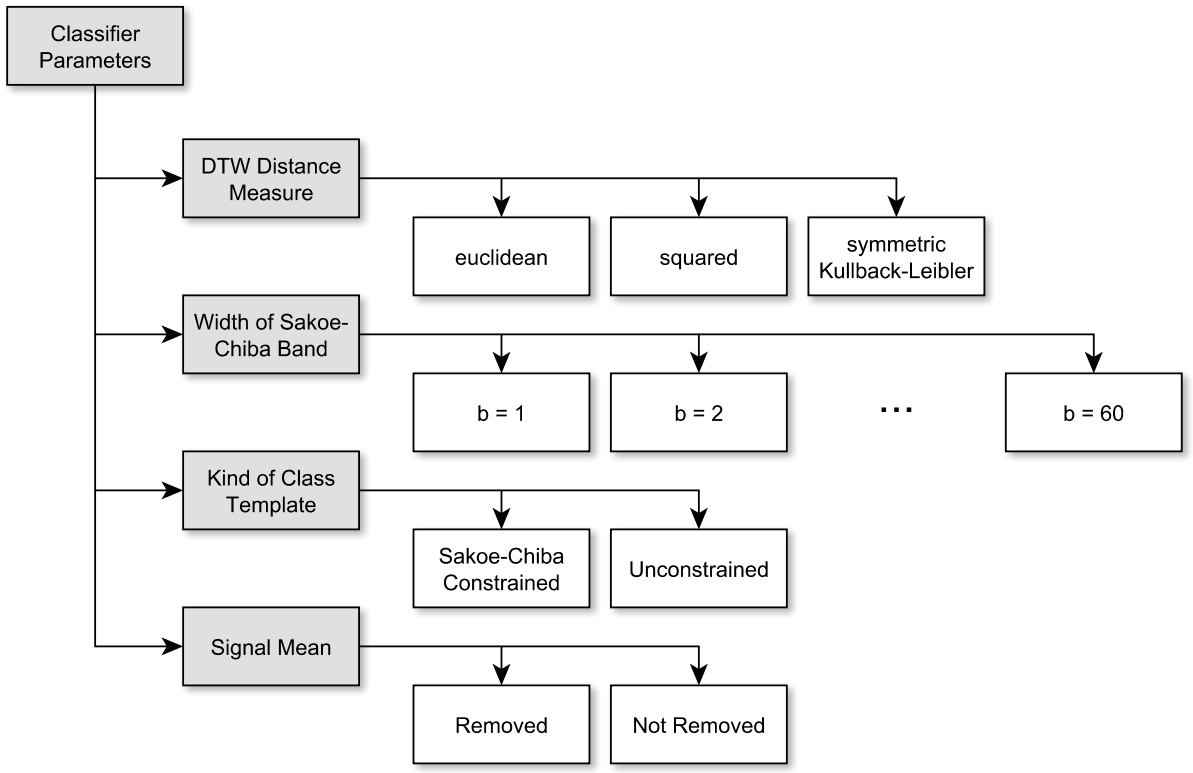


Figure 6.14: Overview of freely selectable parameters and possible parameter values for both DTW classifier and cross-correlation classifier. A paramter studies is conducted to find the optimal set of parameters that leads to highest classification accuracy.

wrapped around the whole classification algorithm except for the measurement data, labelset and class template import. The inserted three loop iterate over the possible values of the used DTW measure, width b of Sakoe-Chiba band and options for removal of the sequence mean values. For each of the resulting 240 iterations the achieved classification accuracy is stored along the setting of the three mentioned parameters. The last of the four parameters, the kind of class templates used for classification is not changed in a loop, but needs to be changed manually by modifying the class template import filename. After calculation of the according classifier accuracies results are plotted. Another run of the classification algorithm has to be performed manually for retrieval of prediction accuracy for a DTW without global constraint of the warping path. The according Matlab sourcecode for the modified algorithm for the parameter studies can be found in listing A.10. Results of the parameter studies are examined in the next section.

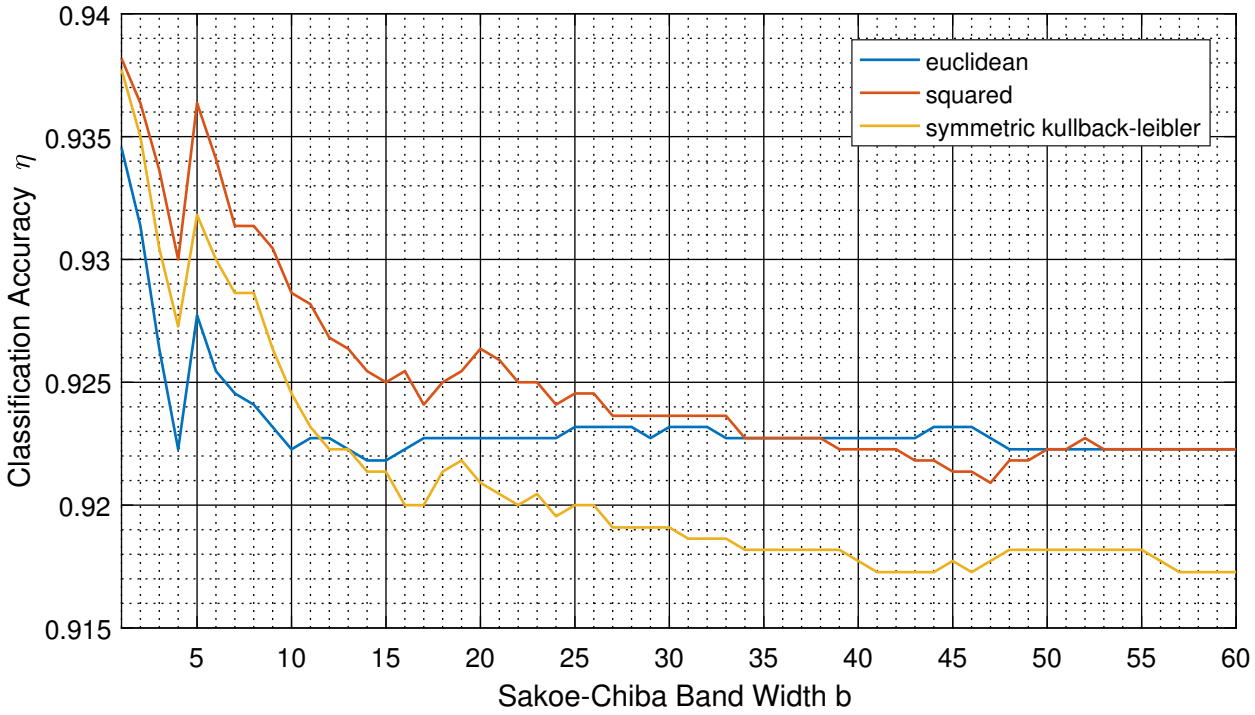
6.4 Evaluation of Classification Result

In this section experimental results of the exhaustive grid search for optimal parameters of both DTW as well as cross-correlation template matching classifiers are analysed and the optimal set of parameters for both classifiers is determined. In this case the optimal set of parameters is the

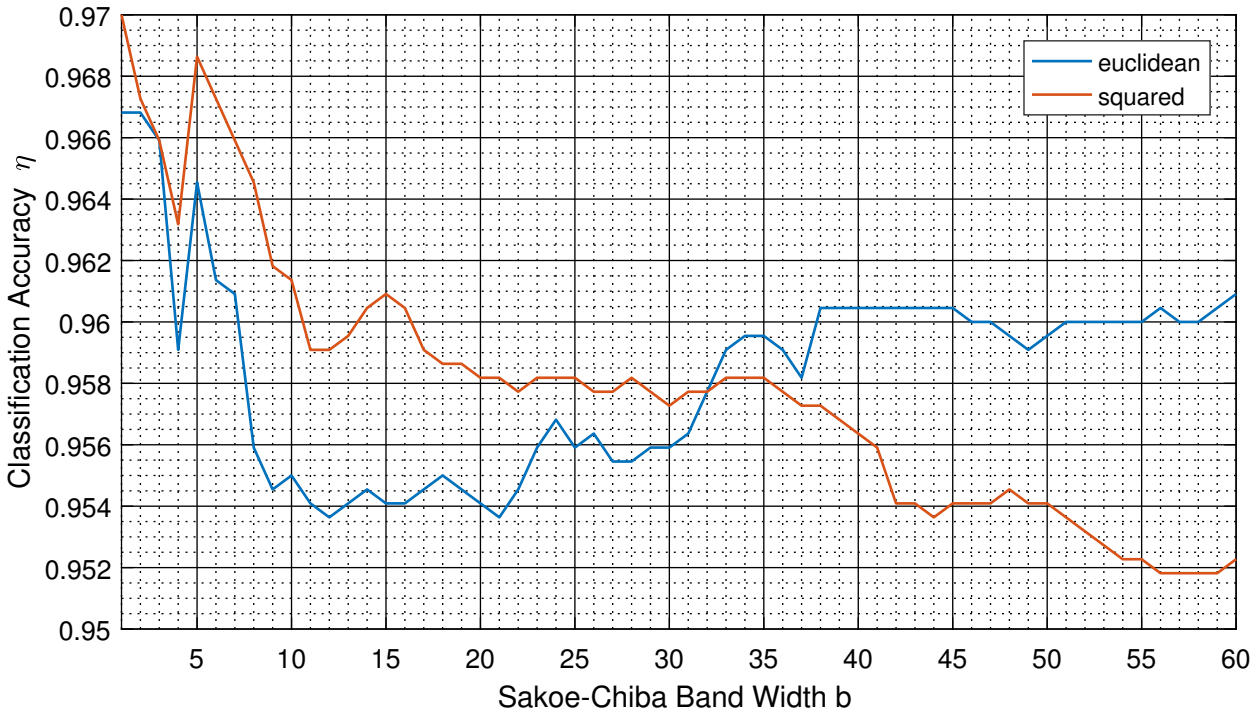
combination of parameter values which lead to highest classification accuracy η as defined in eq. (6.1). In this case accuracy can be measured easily by comparing the prediction result with the labelset. The labelset created in section 5.3 contains true class labels for each of the 2201 segments of the measurement data of the first test cycle run. So calculating accuracy is as easy as comparing each predicted class label with the correpsonding true class label in the labelset. Whenever the predicted class is identical to the true class of the segment, accuracy is increased by 1/2201. Therefore an optimal classifier would reach accuracy of 1.

In the following results of the parameter studies described in section 6.3 are presented in different diagrams showing accuracy score of both the DTW and the cross-correlation classifier for multiple different combinations of parameters. First fig. 6.15 shows accuracy of the the DTW classifier for different DTW distance metrics. Moreover width of the Sakoe-Chiba band restricting the DTW warping path globally is changed in the range of 1 to 60 with a step size of 1 and accuracy score is shown for these different widths. In fig. 6.15a the mean value is removed from test segment as well as class template. Different from that fig. 6.15b shows accuracy score without prior removal of signal mean. As mentioned before symmetric Kullback-Leibler distance can not be evaluated for negative signal amplitudes. Therefore this measure was not evaluated for the sequences with prior removal of the mean value as here negative amplitude values occur. Accuracy score for the cross-correlation classifier und the same parameter variations is depicted in fig. 6.16. Both classifiers use class templates that are created utilizing DTW with a global contraint on the warping path, in this case a Sakoe-Chiba band of width $b = 10$ as explained in section 6.2.4. In contrast to this fig. 6.17 and fig. 6.18 show accuracy for the DTW and the cross-correlation classifier respectively. Again the first plot shows the accuracy score for different DTW distance measures witout prior removal of the test sequence and class template mean values. Unlike this the lower plot again shows accuracy score for sigals without removal of the mean value.

Finding the optimal set of parameters for both classifiers is now as easy as looking at the combination of parameters that achieve highest classification accuracy. Peak accuracy scores for every tested combination of parameters are shown in table 6.2. Here the accuracy scores for both classifiers as well as the correpsonding width of the Sakoe-Chiba band at which highest accuracy is reached are listed. Maximum accuracy scores, printed in bold, are 97.0000 % for the DTW classsifier and 77.9545 % of the cross-correlation classifier. The DTW classifier achieves this accuracy optimum utilizing a class template that was created with globally restricted DTW warping path, removed mean value from test sequence and class template, squared distance measure of the DTW and a width of $b = 1$ for the Sakoe-Chiba band restricting the warping path during classification. Opposed to that the cross-correlation classifier reaches maximum accuracy for a Sakoe-Chiba band width $b = 37$ and euclidean distance measure for the DTW. Again the used class template is one created with restricted DTW warping path and signal mean of test sequence and class template are removed. Looking at the results for the classification accuracy in table 6.2 it becomes clear that the DTW classifier performs generally much better than the cross-correlation classifier. How big the difference between both classsifer accuracies is depends on the combination of parameters. For the optimum case of the DTW classifier the cross-correlation accuracy is 20.64 % lower than accuracy of the DTW classifier. For the optimum case of the cross-correlation classifier the difference is 18.73 %. If the signal mean is not removed from test sequence and class template performance of the cross-correlation classifier

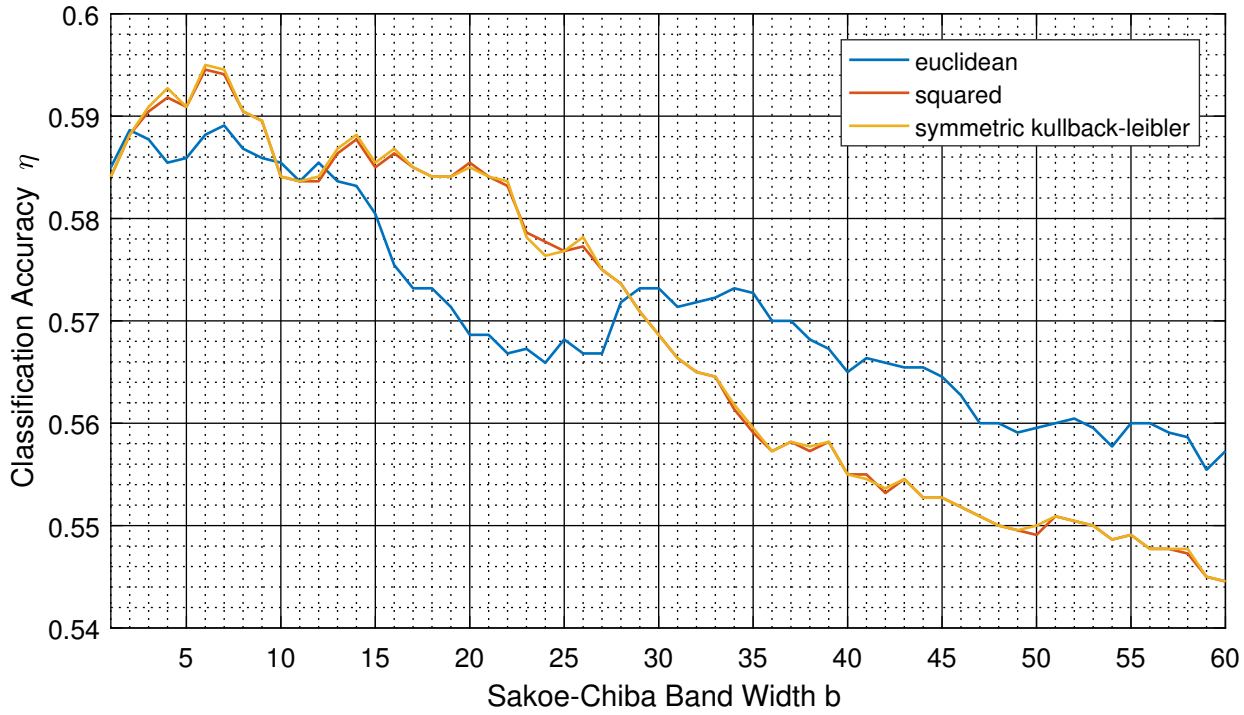


(a) Accuracy of the DTW classifier on the test set without previous subtraction of the mean value.

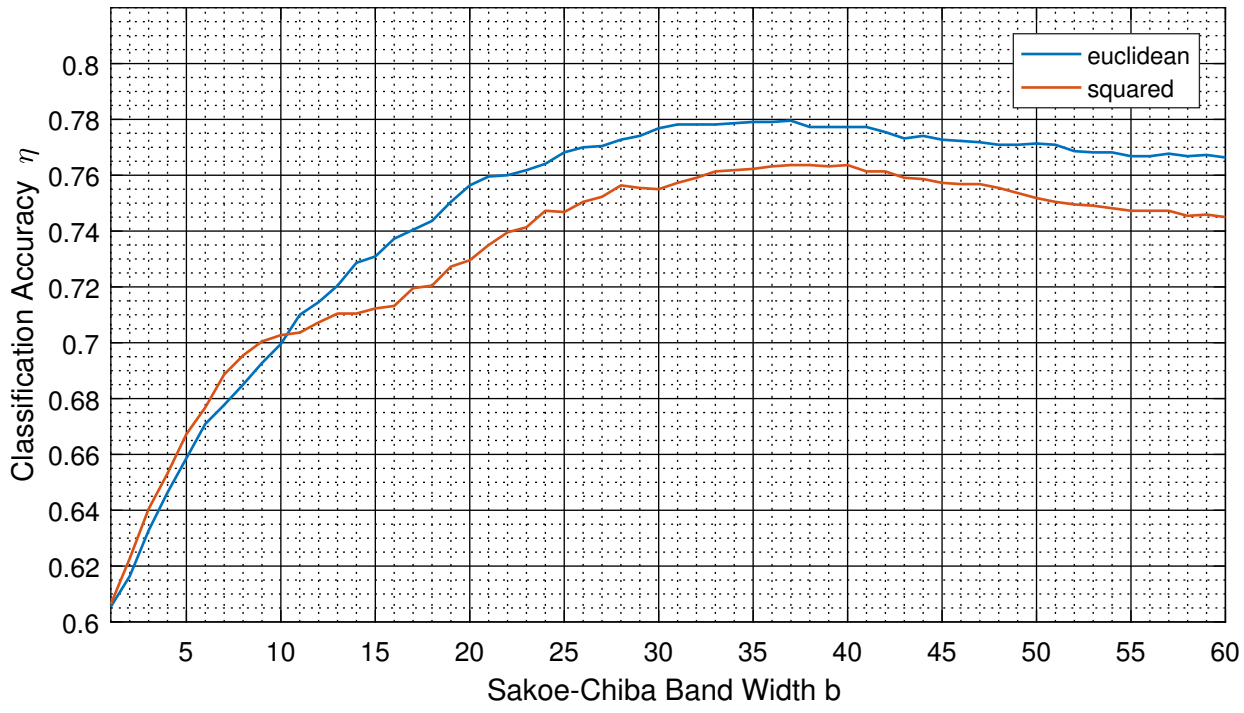


(b) Accuracy of the DTW classifier on the test set with previous subtraction of the mean value.

Figure 6.15: Classification accuray η of the classification based on DTW distance with globally contrained warping path. Constraint is a Sakoe-Chiba band with width b that is varied during the parameter studies. Different colored lines refer to different distance measures for calculating the DTW warping cost. Accuracy is estimated both for the test set with and without previous subtraction of the mean value of each segment. Here class templates which were averaged with a Sakoe-Chiba band of width $b = 10$ as global constraint on the warping path were used as reference.

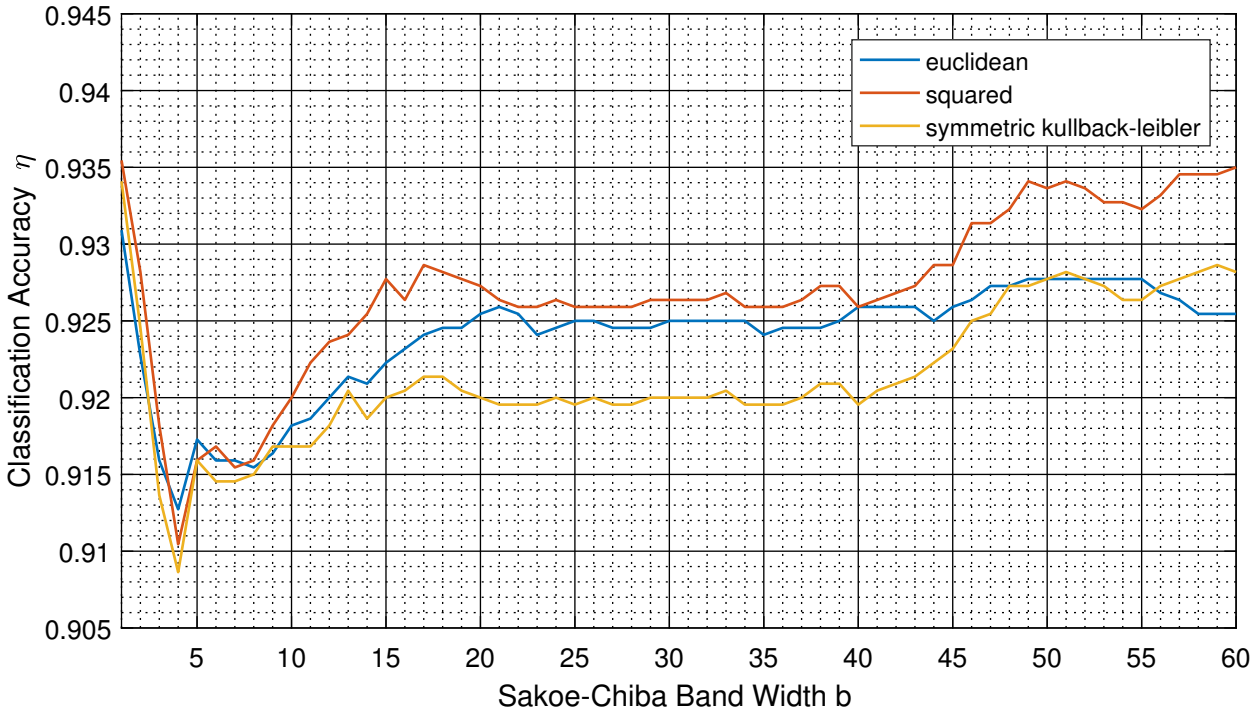


(a) Accuracy of the cross-correlation classifier on the test set without previous subtraction of the mean value.

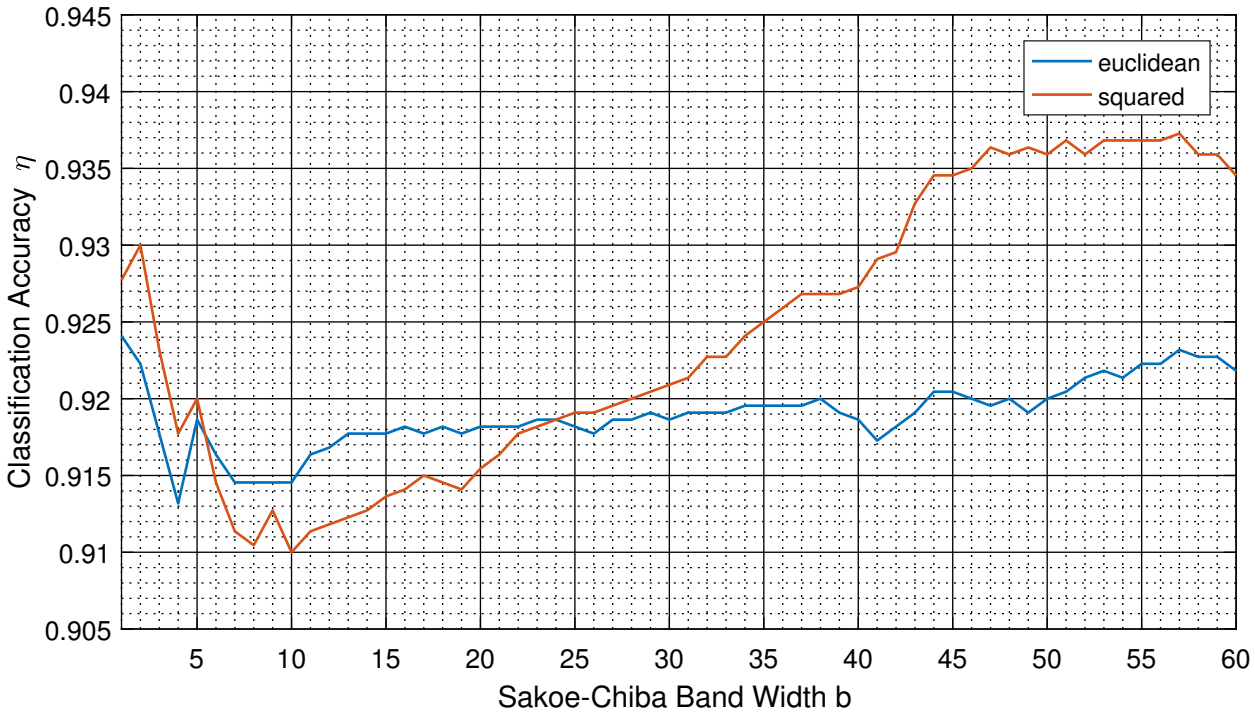


(b) Accuracy of the cross-correlation classifier on the test set with previous subtraction of the mean value.

Figure 6.16: Classification accuracy η of the classification based on cross-correlation of the templates with the subsequences of the test set. As template and test sequence are aligned via DTW accuracy depends on the Sakoe-Chiba band width b and the DTW distance measure as well. Again the accuracy is compared for the test set with and without previous subtraction of the mean value of each segment. Here class templates which were averaged with a Sakoe-Chiba band of width $b = 10$ as global constraint on the warping path were used as reference.

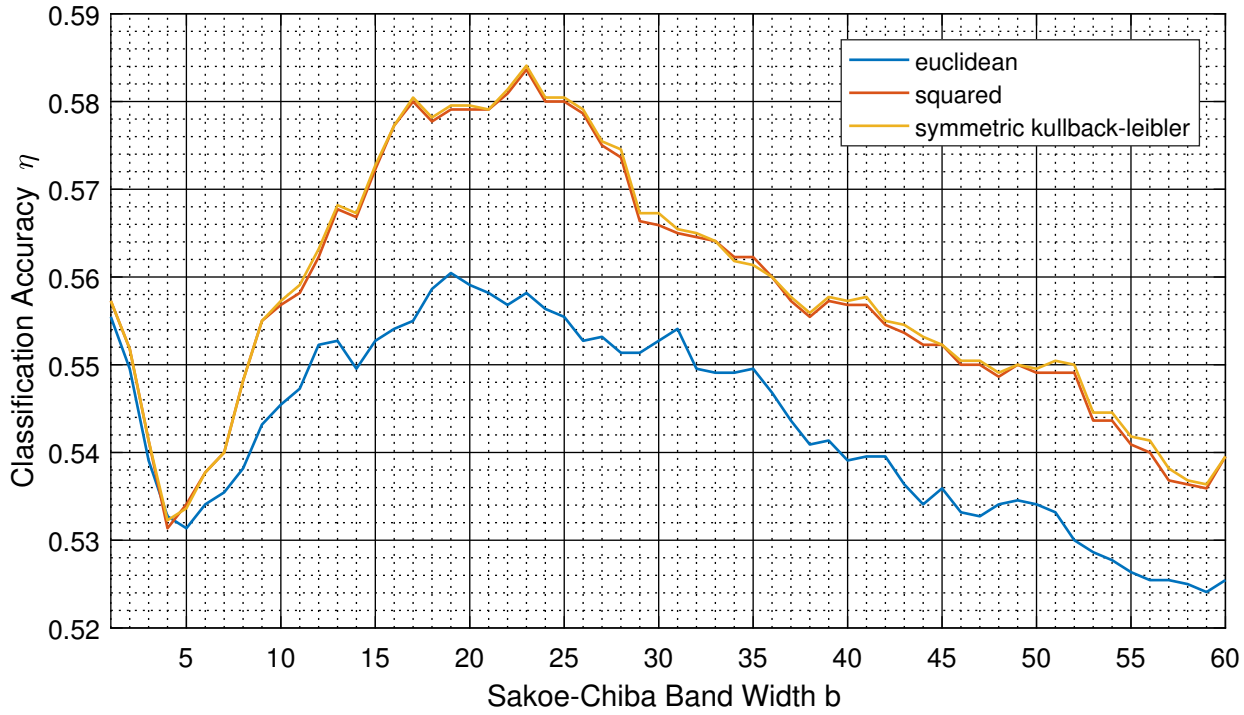


(a) Accuracy of the DTW classifier on the test set without previous subtraction of the mean value.

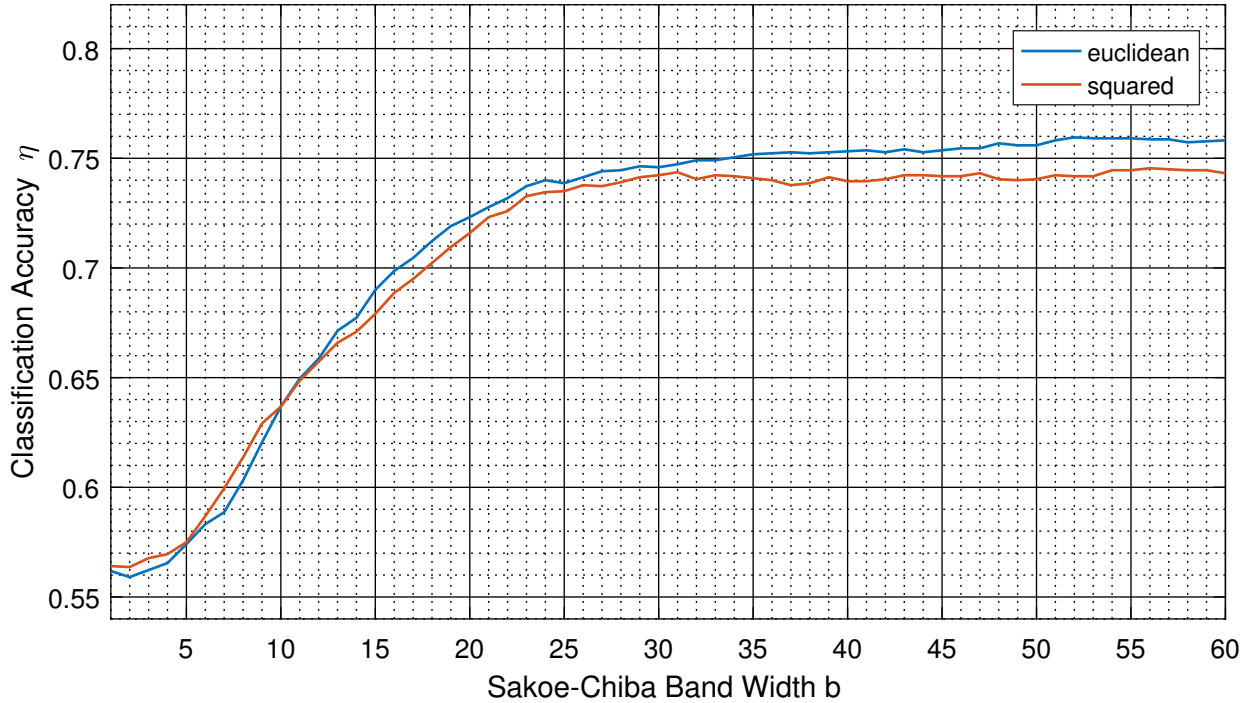


(b) Accuracy of the DTW classifier on the test set with previous subtraction of the mean value.

Figure 6.17: Classification accuray η of the classification based on DTW distance with globally contrained warping path. Constraint is a Sakoe-Chiba band with width b that is varied during the parameter studies. Different colored lines refer to different distance measures for calculating the DTW warping cost. Accuracy is estimated both for the test set with and without previous subtraction of the mean value of each segment. Here class templates which were averaged without a global constraint on the warping path were used as reference.



(a) Accuracy of the cross-correlation classifier on the test set without previous subtraction of the mean value.



(b) Accuracy of the cross-correlation classifier on the test set with previous subtraction of the mean value.

Figure 6.18: Classification accuracy η of the classification based on cross-correlation of the templates with the subsequences of the test set. As template and test sequence are aligned via DTW accuracy depends on the Sakoe-Chiba band width b and the DTW distance measure as well. Again the accuracy is compared for the test set with and without previous subtraction of the mean value of each segment. Here class templates which were averaged without a global constraint on the warping path were used as reference.

compared to the DTW classifier even goes down. Here the DTW classifier accuracy is up to 37.05 % better than accuracy of the cross-correlation classsifier.

Table 6.2: Peak accuracies η and optimal width b of Sakoe-Chiba band for DTW and cross-correlation classifiers for different combinations of parameter values.

Kind of Class Template	**Signal Mean**	**DTW Distance Measure**	**Sakoe-Chiba Band Width b**		**Accuracy η / %**	
			DTW	XCOR	DTW	XCOR
with constraint	removed	euclidean	1	37	96.6818	**77.9545**
with constraint	removed	squared	1	37	**97.0000**	76.3636
with constraint	not removed	euclidean	1	7	93.4545	58.9091
with constraint	not removed	squared	1	6	93.8182	59.4545
with constraint	not removed	symmetric KL	1	6	93.7727	59.5000
without constraint	removed	euclidean	1	52	92.4091	75.9545
without constraint	removed	squared	57	56	93.7273	74.5455
without constraint	not removed	euclidean	1	19	93.0909	56.0455
without constraint	not removed	squared	1	23	93.5455	58.3636
without constraint	not removed	symmetric KL	1	23	93.4091	58.4091

The last parameter option that has to be tested is usage of an globally unrestricted DTW warping path for calculation of the distance matrix during the DTW. Regarding to the experiments above this complies to an infinite Sakoe-Chiba band width $b \to \infty$, meaning the warping path can reach its global minimum, however is susceptible to outliers and therefore can lead to a high amount of signal warping. Again accuracies of both classifiers under different combinations of all other parameters except for the band width b of the Sakoe-Chiba band are evaulated. According results are shown in table 6.3. Once more maximum accuracy vealues reached for both classifier are printed in bold. For the DTW classifier the maximum accuracy reached is 96.3182 % and for the cross-correlation classifier it is 74.0000 %. Both values are slightly lower than the peak values reached with globally restricted warping path in the DTW process. However the differences in accuracies reached with the DTW and cross-correlation classifier are still similar to the ones in table 6.2. In the cases with removed mean value from all test sequences and class templates DTW accuracy is approximately 20 % better than cross-correlation accuracy. If the mean value is not removed from test sequence and class template the differences between both classifier accuracies increase up to 46 %, which is even more than in the above case of globally restricted DTW. Apart from this accuracies for the DTW classifier are ver similar to the ones shown in table 6.2. Differences are larger for the cross-correlation classifier, which is reasoned by the higher distortion of the signals by the unrestricted DTW, which affects the cross-correlation of test sequences and class templates significantly.

Summarizing the above results, it can be said that the DTW classifier outperforms the cross-correlation classifier by means of classification accuracy by at least 18 % in every test case.

Table 6.3: Classification accuracy η of both classifiers for unconstrained DTW for different combinations of parameters. Unconstrained means no Sakoe-Chiba band is applied to the warping path, so $b \to \infty$.

Kind of Class Template	**Signal Mean**	**DTW Distance Measure**	**Accuracy η / %**	
			DTW	XCOR
with constraint	removed	euclidean	**96.3182**	73.8636
with constraint	removed	squared	95.3182	73.1364
with constraint	not removed	euclidean	92.7727	48.8636
with constraint	not removed	squared	92.9545	48.7727
with constraint	not removed	symmetric KL	92.4545	48.7727
without constraint	removed	euclidean	92.0000	73.3182
without constraint	removed	squared	93.3636	**74.0000**
without constraint	not removed	euclidean	92.7273	46.6364
without constraint	not removed	squared	93.8636	47.6818
without constraint	not removed	symmetric KL	93.1364	47.6818

Generally performance of the DTW classifier is best for a combination of removed mean value from the signals and utilization of class templates created with globally restricted DTW. There is no significant difference in accuracy of the DTW classifier when restricting the warping path during the classification. Removing signal mean values from test sequences and class templates does not affect accuracy of the DTW classifier much, however accuracy of the cross-correlation classifier is significantly increased by removing the mean values. The best classifier found by the exhaustive grid search of the defined parameter grid is the DTW classifier using a Sakoe-Chiba band of width $b = 1$, squared distance metric, removed mean values and class templates created with globally restricted DTW. Reason for this combination of parameters being optimal is comprehensible as for instance the class templates created with globally restricted DTW warping path are much smoother than the templates created without restriction (see section 6.2.5). Moreover removal of the signal means increases classification accuracy as the signal offset is ignored. This is important as the measurement data has a sligthly varying offset due to neglection of the influcence of the cooling unit. When the offset is removed only signal shape is taken into account on classifying a pattern rather than looking on the absolute amplitude values. This increases especially performance of the cross-correlation classifier, whereas the DTW classifier is not affected that much by removal of the signal offset. Regarding the distance metric of the DTW the squared metric performs in most casses similar to symmetric Kullback-Leibler distance and slightly bettern than euclidian distance. This might be due to the fact that squared and symmetric Kullback-Leibler distance weight larger distances between test sequence and class template higher than smaller distances, whereas euclidean distance applies same weights regardless of how large deviation between both sequences is.

Apart from the simple accuracy score used for finding optimal classifier parameters it is interesting to examine how good the classifiers can distinguish between individual classes. For this purpose a *confusion matrix* is a very helpful tool. Such a matrix is calculated for both optimally adjusted classifiers. The confusion matrix for the optimal DTW classifier is shown in fig. 6.19 and the confusion matrix for the optimal cross-correlation classifier is depicted in fig. 6.20. The confusion matrix is a square matrix with columns referring to the predicted class labels $0, 1, \ldots, 11$ occuring in the test set and rows referring to the true class labels $0, 1, \ldots, 11$. Matrix elements (i,j) contain the number of samples of a specific kind in the test set that are labeled with a specific class label according to the column of the element. For instance element (2,3) = 3 in the confusion matrix for the DTW classifier refers to the number of segments of true class 2 (row $i = 2$) in the test set that are misclassified as class 3 according to the column index $j = 3$. So elements that do not lie on the matrix diagonal from the top left to the bottom right corner refer to misclassified segments, whereas elements on this matrix diagonal are correctly classified elements. So for an ideal classifier with accuracy of 1 all elements except for those on the diagonal are zero. In opposite to the accuracy score the confusion matrix not only shows how many classes are misclassified, but also illustrates how classes are mixed up with other classes. This allows for detection of classes, which can not be distiguished properly by the classifier.

Regarding the optimal DTW classifier the overall accuracy is very good and only a few classes are misclassified, meaning only a few elements except those on the matrix diagonal differ from zero. Only classes 4 and 5, which refer to the forward and backward movement of the Y-axis are mixed up many times. Class 4 is mistakenly classified as class 5 exactly 13 times, which equals 13 % of all 100 segments of true class 4. Class 5 is falsely classified as class 4 precisely 23 times, equivalent to 23 % of all segments of true class 5. This shows, the classifier is weak in distiguishing especially those two classes, however performs very well on all other classes, which is why the resulting overall accuracy is as high as 97 %.

The confusion matrix of the cross-correlation classifier shown in fig. 6.20 looks similar to the one of the DTW classifier, however there are more elements which do not lie on the matrix diagonal and differ from zero, meaning there are more misclassified elements. In total 485 of the 2201 segments are classified with wrong class labels giving the overall accuracy of 77.9545 % which is equal to the accuracy found earlier. The confusion matrix now shows which of the classes are most problematic and get misclassified most frequently by the cross-correlation classifier. As can be seen classes 4 and 5 are misclassified most often similar as with the DTW classifier. 31 % of segments with true class 4 are falsely classified as class 5 and 20 % of segments with true class 5 are misclassified as class 4. Different from the DTW classifier the cross-correlation classifier has also problems with distiguishing between classes 8 and 9 which refer to the forward and backward movement of the C-axis. Moreover classes 2 and 3 are mixed up, but only by 6 % respectively 9 %, which is higher than the misclassification ratio of the DTW classifier for those both classes, but still on an acceptable level. The biggest isue with the cross-correlation classifier in comparison to the DTW classifier is recognition of class 0 which corresponds to the idle state of the machine. Only 726 of the 1101 segments with true class 0 are classified correctly as class 0, which euqals only 65.94 %. Instead segments with true class 0 are misclassified in 17.08 % of all cases as class 7, in 13.90 % of all cases as class 8 and in 1.81 % of the cases as class 6. A few segments are even misclassified as classes 1, 3, 4, 9, 10 and 11. All in all the cross-correlation

classifier performs not as good as the DTW classifier as it misclassifies classes more frequently and has serious problems labelling segments which refer to the idle state correctly.

So all in all the developed DTW classifier with optimally adjusted parameters achieves best performance and can therefore be used to classify segments in future test data sets.

6.5 Conclusion of Shape Based Classification Approach

In the course of the preceeding chapter first foundations of template matching classification were explained. This included a detailed description of the overall process of classifying time sequences by comparison with other known time sequences, so called templates. Moreover Dynamic Time Warping (DTW) and cross-correlation were introduced as useful distance respectively similarity measures for classification of time sequences. After this a detailed explanation of the process involved in creating class templates by averaging multiple training sequences was given. The Accurate Shape Averaging (ASA) algorithm was implemented in Matlab to create templates for the 12 base classes corresponding to the basic states of the milling machine. After evaluation of the averaging results two template matching classifiers were developed. One based on DTW distance and another one based on cross-correlation of test sequence and class templates. Both classifiers had four parameters with different possible values which needed to be choosen properly to achieve maximum classsification accuracy. This was done by an exhaustive grid search, which led to optimal parameter sets for both classsifiers. Performance of both classifiers was evaluated and compared. Here it turned out that the DTW classifier outperformed the cross-correlation classifier easily and achieved a high classification accuracy of 97 %.

Although the achieved classsification accuracy of the optimized DTW classsifier is already very good and suffiecient for most applications further improvement is possible. For instance classes which are confused by the algorithm frequently as for example classes 4 and 5 could be merged into one virtual class containing all segments from both classes. This would simplify the classification task and increase classification accuracy as there were less misclassifications on those two classes. Moreover segments with true class 0 are misclassified 19 times as class 1, which is due to the fact that in between of the 20 subcycles of the test cycle run there occur 19 longer idle phases, which are misclasified as class 1 which refers to the main spindle activation. In order to prevent this misclassification it would be necessary to create another class template for the longer idle state in between of the subcycles. If this optimizations were conducted accuracy would approach 100 % resulting in an optimal classifier for the given classification problem.

Apart from this during the exhaustive grid search for the optimal set of parameters one parameter that possibly affects classification accuracy was not considered. Meant is the warping path restriction during template creation. In the preceeding chapter only two cases were examined, first class templates were created by DTW averaging with unrestricted warping path and second a Sakoe-Chiba band of width $b = 10$ was implemented to restrict deviation of the warping path during DTW averaging. As the above results show this modification of the template creation process significantly affects classification accuracy of both the DTW and the cross-correlation classifier. To investigate how restriction of the DTW warping path during template creation influences accuracy of the later classification, further research is neccessary. Apart from the

two cases tested above multiple different sets of averaged templates with different settings for the Sakoe-Chiba band width, for instance in the range of 1 to 60 with step size 1, should be created and resulting classsification accuracy of both classifiers should be compared. This way the Sakoe-Chiba band during template creation can be optimized and classification accuracy might be enhanced further. Before conducting this additional parameter studies it is neccessary to speed up the template averaging algorithm presented in section 6.2.4 as the creation of all 12 class templates takes 1100 s. This would lead to a duration of 66000 s for completion of the set of class templates for the parameter studies in the range proposed above. As already mentioned in section 6.2.5 further improvement of the template creation algorithm by means of computational expense of the algorithm is easily possible so that investigation of this parameter is possible.

The last aspect that should be considered in future work is usage of Hybrid DTW instead of normal DTW for both creation of the class templates and calculation of the DTW distance during classification. As explained in section 6.1.2 HDTW considers not only the amplitude values of the signals, but also the slope at each sample position for calculating proper alignment of both sequences. This way more shape information is used to algin both signals and classification accuracy might increase.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1080	19	0	1	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	97	3	0	0	0	0	0	0	0	0
3	0	0	4	96	0	0	0	0	0	0	0	0
4	0	0	0	0	87	13	0	0	0	0	0	0
5	0	0	0	0	23	77	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	97	3	0	0
9	0	0	0	0	0	0	0	0	0	100	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure 6.19: Confusion matrix for the optimal DTW classifier. Rows show true classes and columns predicted classes of each segment within the test data set. Ideally all elements except for the diagonal from the upper left to the lower right would be zero, meaning class labels are predicted correctly for all segments in the test set.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	726	5	0	1	1	0	20	188	153	2	2	2
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	85	9	0	6	0	0	0	0	0	0
3	0	0	6	94	0	0	0	0	0	0	0	0
4	0	0	0	0	69	31	0	0	0	0	0	0
5	0	0	0	0	20	80	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	90	10	0	0
9	0	0	0	0	0	0	0	0	29	71	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure 6.20: Confusion matrix for the optimal cross-correlation classifier. Rows show true classes and columns predicted classes of each segment within the test data set. Ideally all elements except for the diagonal from the upper left to the lower right would be zero, meaning class labels are predicted correctly for all segments in the test set.

7 Feature Based Classification

Aim of the following chapter is the development and evaluation of a classical machine learning approach for classification of the previously acquired and segmented measurement data. Different from the shape based classification approach presented in chapter 6 measurement data is first transformed from the time tomain into feature space which reduces the amount of data to process significantly and makes the data applicable to common machine learning algorithms for classification. Before applying any classification algorithms first fundamentals of feature based classification are explained. This includes an introduction in the common terminology and matehmatical formulation regarding feature based datasets as well as a detailed insight into the general operational principle of feature based classifiers. Thereby focus lies on explanation of the neccessary process steps training, evaluation and testing. Afterwards the classification models used in the later course of this chapter are explained in detail. After clarification of the basics a literature review is conducted to collect possible features for abstraction of the present measurement data. The found statistical features are then extracted from the measurement data and stored in three different 49-dimensional datasets, one for testing and two for training. Besides a detailed overview of the extracted features the extraction algorithm is explained in-depth. Afterwards the actual classification workflow is implemented in the Python based visual machine learning tool Orange3. This workflow comprises standardization of the extracted datasets and selection of the 10 most relevant features by means of a decision tree which ranks features based on a measure called information gain. The need for dimensionality reduction by feature selection is justified by explaining problems occuring during classification in too high dimensional datasets. After feature selection hyperparameters of the utilized classsifiers are tweaked manually to achieve maximum classification performance on the test dataset. Classifiers used are k-nearest-neighbours, decision tree, random forest, AdaBoost, support vector machine, stochastic gradient descent and naive bayes classifier. All classifiers are trained on the preprocessed training data and evaluated on the likewise preprocessed training set. Performance is measured and compared by means of area under curve, classification accuracy, precision, recall and F_1-score. Moreover confusion matrices for all classifiers are created and analyzed. Finally conclusions are drawn and performance of the developed feature based classifiers is compared to performance of the previously developed shape based classifiers.

7.1 Foundations of Feature Based-Classsification

The following section gives an overview of the foundations of feature based classsification. It first introduces common terminology of feature based datasets and explains their basic structure. Afterwards follows a brief introduction into feature space represenation of the dataset and an explanation of the operational principle of feature based classifiers. Moreover the section describes the typical process steps training, evaluation and testing of a classification model and

explains on a general level how model fitting to the training data is achieved and how new data samples can be classified. In this context the very important concept of the *decision boundary* is introduced and examples for a well fit classifier and a badly fit classifier are given. After teaching the fundamentals of feature based classification this section gives a detailed insight into the functional concepts as well as pros and cons of seven popular models used for pattern classification. The presented algorithms are k-nearest-neighbour, decision tree, random forest, AdaBoost, support vector machine, stoachstic gradient descent and the naive bayes classifier. The list is limited to these seven algorithms as they are used in the later course of the analysis.

7.1.1 Dataset Nomenclature

To understand the difference between the feature based classification approach and the shape based classification conducted in chapter 6 this section first explains the basic concept of feature extraction from time series data and introduces fundamental terms for describing data in the context of feature based classification.

Consider the previously acquired measurement data which corresponds to a discrete sequence of equally sampled amplitude values s_j with $j = \{1, 2, \ldots, q\}$. During segmentation as described in chapter 5 the complete measurement time sequence is cut to n shorter time series segments. The kth segment can therefore be described as

$$\boldsymbol{s}_k = \left\{ s_{k,1}, s_{k,2}, \ldots, s_{k,j}, \ldots, s_{k,q} \right\}. \tag{7.1}$$

Length q of each of the n time series segments is not constant, but varies significantly.

While the shape based classification developed in the previous chapter worked directly with these sequences, the feature based classsification aims to reduce amount of data needed for classification by transforming each of the n time series segments to a set of scalar values, the feature vector, which is then fed into the classification algorithm. The step of calculating the feature vector for the time series segment is called *feature extraction*. One of the scalar values $x_k^{(i)} \in \mathbb{R}$ is calculated by applying a linear or non-linear function $f_i : \mathbb{R}^q \to \mathbb{R}$ to the time series segment $\boldsymbol{s}_k$. Such a function could be for instance calculation of the mean value of the time series segment

$$x_k^{(i)} = f_i(\boldsymbol{s}) = \frac{1}{q} \sum_{j=1}^{q} s_{k,j}. \tag{7.2}$$

To get a set containing p different scalar values for description of the time series segment p different functions $f_i(\boldsymbol{s})$ with $i = \{1, 2, \ldots, p\}$ need to be applied to the time series segment $\boldsymbol{s}_k$. This gives the *feature vector*

$$\boldsymbol{x}_k = \left\{ x_k^{(1)}, x_k^{(2)}, \ldots, x_k^{(i)}, \ldots, x_k^{(p)} \right\}. \tag{7.3}$$

This feature vector can also be called *sample* in the context of feature based classification. Caution needs to be exercised as in context of shape based classification as conducted in the previous chapter the whole time series segment was called sample. Here only the abtraction of the segment in form of the feature vector $\boldsymbol{x}_k$ is the sample.

If all n time series segments are considered the feature extraction gives n different samples $\boldsymbol{x}_k$ with $k = \{1, 2, \ldots, n\}$ which can be composed in a data matrix $X \in \mathbb{R}^{n \times p}$ with

$$X = \begin{bmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(i)} & \cdots & x_1^{(p)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(i)} & \cdots & x_2^{(p)} \\ \vdots & \vdots & \ddots & \vdots & & \vdots \\ x_k^{(1)} & x_k^{(2)} & \cdots & x_k^{(i)} & \cdots & x_k^{(p)} \\ \vdots & \vdots & & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(i)} & \cdots & x_n^{(p)} \end{bmatrix}. \tag{7.4}$$

This data matrix is also refered to as *dataset*. Columns $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ with $i = \{1, 2, \ldots, p\}$ are simply called features containing feature values for all n samples. The data matrix can also be written as set

$$X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p \tag{7.5}$$

where the $\boldsymbol{x}_k$ are samples in form of row vectors. In this case the dataset is not labeled. Labelling means that another column vector $\boldsymbol{c} = \{c_1, c_2, \ldots, c_k, \ldots, c_n\}^\intercal$ is added to the data matrix. So each sample can be assigned a single class label c_k which results in the labeled dataset

$$X_L = \left[\begin{array}{cccccc|c} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(i)} & \cdots & x_1^{(p)} & c_1 \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(i)} & \cdots & x_2^{(p)} & c_2 \\ \vdots & \vdots & \ddots & \vdots & & \vdots & \vdots \\ x_k^{(1)} & x_k^{(2)} & \cdots & x_k^{(i)} & \cdots & x_k^{(p)} & c_k \\ \vdots & \vdots & & \vdots & \ddots & \vdots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \cdots & x_n^{(i)} & \cdots & x_n^{(p)} & c_n \end{array}\right]. \tag{7.6}$$

Each of the n class labels can take on one of m different values $c_k \in C$ with $\{C = 1, 2, \ldots, l, \ldots, m\}$. These are nominally scaled values whcih means it is only possible to compare if two class labels are different or equal. It is neither possible to compare if one class label is greater or lower than the other (ordinal scale) nor can be conducted any arithmetic operations on the class label values (interval and ratio scale). The labeled dataset can also be described as set

$$X_L = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p \times \{c_1, c_2, \ldots, c_k, \ldots, c_n\} \tag{7.7}$$

with an additional class label vector. Despite mathematically not being identical, both the set and the matrix representation can be used for description of the dataset. [40, 44]

7.1.2 Principle of Feature Based Classification

Now that fundamentals of feature extraction are fully understood the operational principle of a feature based classification algorithm is explained in this section. Moreover training and testing

of a feature based classifier are described in detail. First Consider the labeled example dataset

$$X_L = \left[\begin{array}{cc|c} 6.3 & 3.8 & 2 \\ 2.2 & 4.2 & 1 \\ 2.0 & 3.3 & 1 \\ 5.3 & 0.9 & 3 \\ 5.5 & 3.9 & 2 \\ 4.7 & 1.3 & 3 \\ 4.2 & 0.8 & 3 \\ 5.3 & 3.4 & 2 \\ 1.3 & 4.0 & 1 \end{array}\right]. \tag{7.8}$$

It consists of $n = 9$ samples and two features $x^{(1)}$ and $x^{(2)}$ corresponding to the first resepctively second column of the matrix, hence dimensionality of the dataset is $p = 2$. The third column represents class labels. As can be seen, the dataset contains three different classes $c_k = \{1,2,3\}$, whereby each class occurs three times. The dataset can be plotted as shown in fig. 7.1. Each sample corresponds to a point $\left(x^{(1)}, x^{(2)}\right)$ in a two-dimensaional plane. If the dataset contained more than two features, for example three, a three-dimensional plot would be neccessary to display the dataset. Accordingly for a p-dimensional dataset p linearly independent base vectors would be needed for display of the data. These base vectors span a vector space called *feature space*. Coming back to the example it becomes clear that samples are accumulating in feature space depending on their class. For instance all samples of class $c = 1$ are in the upper left corner, samples of class $c = 2$ are in the upper right corner and samples of class $c = 3$ are located in the lower area of feature space.

Like in the previously conducted shape based classification feature based classification consists of three major steps *training*, *evaluation* and *prediction*. This is illustrated in fig. 7.2. In the training phase the classifier is fit to a labeled training dataset, which is created by feature extraction from measurement data and manual or automatic labeling (see chapter 5). During this process the model parameters are adjusted based on the training data, so that the classification algorithm adopts to the training data and learns specific properties of the samples belonging to a distinct class within the training dataset. When the classifier training is finished performance of the classifier is evaluated by predicting class labels of another independet test dataset created by feature extraction from an independent experiment. These predictions are then compared to the true class labels of the labeled test dataset and some performance score like classification accuracy is calculated. If the score is sufficient the trained classifier can be used to predict class labels of new data samples with unknown class label. [44, 45, 113]

Returning to the example dataset plotted in feature space in fig. 7.1 training of the classifier can be understood as determination of a so called *decision boundary* which separates feature space into multiple different regions belonging to a specific class each. During the training phase appearance of this boundary is adjusted based on the training dataset. In this case for instance the boundary is placed so that the distance between decision boundary and each sample of the

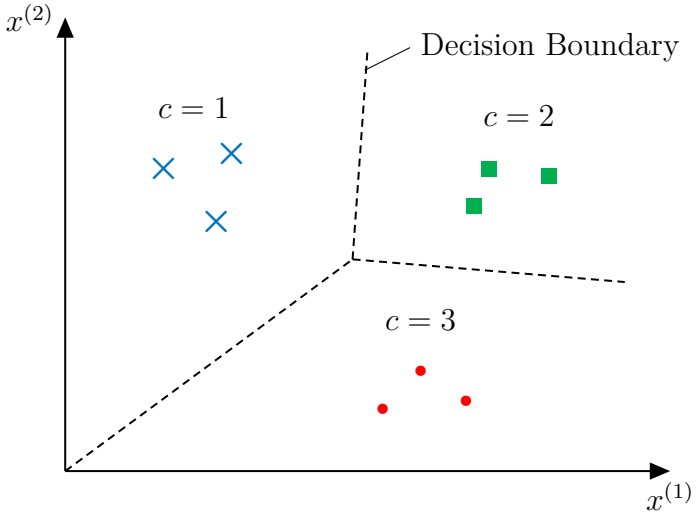


Figure 7.1: Two dimensional feature space with samples from example dataset and decision boundary. Samples belong to three different classes which can be separated by three linear functions, forming a more complex decision boundary which divides feature space into three areas each belonging to a particular class.

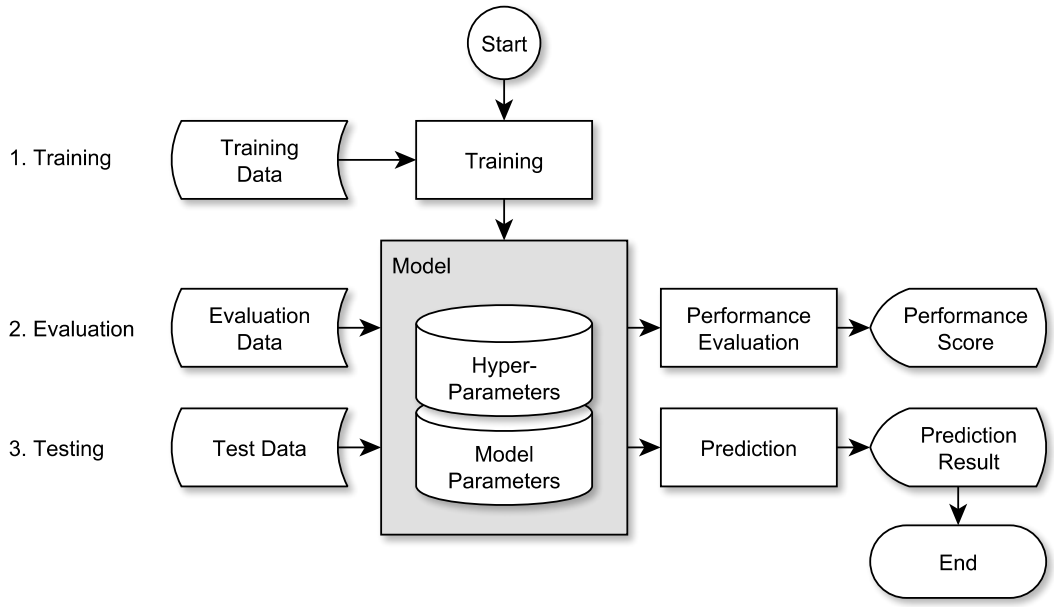
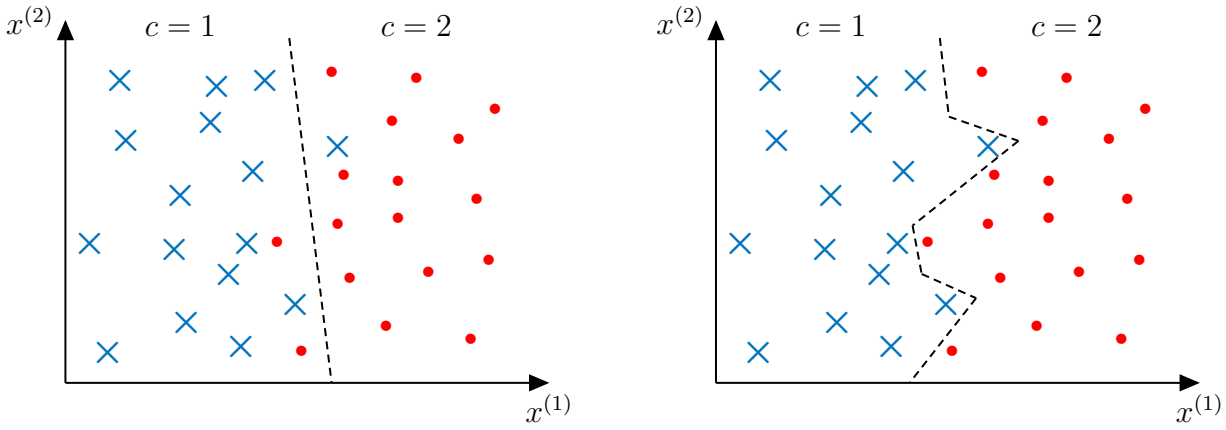


Figure 7.2: Flowchart of the basic process steps for machine learning, model training, evaluation and testing. First the model consisting of a set of model parameters is trained based on the training data. During training model parameters are fit to the data. Afterwards model performance is evaluated on an independent evaluation dataset. When model performance is sufficient it can be used for making predictions on previously unseen test data. Hyper-parameters can be tweaked to change model behaviour during training and testing.

training dataset is maximized. For the sake of simplicity here the decision boundary consist only of linear functions which in general is not neccessary. Depending on the classification algorithm used the decision boundary can be non-linear as well. Moreover the decision boundary is in general not a one-dimensional line, but a $(p-1)$-dimensional hypersphere, depending on the number p of features describing each sample. If for example the dataset contained 3 rather than only 2 features the decision boundary would have been a two-dimensional surface, separating classes in three-dimensional feature space. If the user-adjustable parameters of the algorithm, so called *hyper-parameters*, are choosen well and the ratio between number of feature and number of samples in the training dataset is choosen appropietly a the resulting decision boundary is very smooth separating classes well without overfitting to outliers in the training dataset. This is shown in fig. 7.3a. If on the opposite hyper-parameters are choosen unfavourable the resulting decision boundary can become rough and overfit to the training data as can be seen in fig. 7.3b. This case is unwanted as then later prediction would become less accurate, because the classification algorithm is indeed fit very precisely to the training data, however has not adopted to the general characteristics of the training data. This property of the classification algorithm is refered to as *generalization*. A well trained algorithm is able to generalize to different test dataset very well and achieves high classification performance. When the classifier is overfit to the training data, generalization and therefore classification perforance is usually poor. [44, 114]



(a) Smooth decision boundary as result of well choosen hyper-parameters of the classifier.

(b) Rough decision boundary which overfits to outliers as result of badly choosen hyper-parameters.

Figure 7.3: Difference between a smooth decision boundary and a rough decision boundary as result of model overfitting. If user-adjustable parameters of the classifier are tuned well the resulting decision boundary separates both classes without overfitting to outliers. On the opposite if parameters are chosen insuffieciently the decision boundary becomes rough and overfits to the training data which results in poor classification performance on test datasets which differ from the training dataset.

Once the classifier is fit to the training dataset prediction on a test dataset can be performed. Aim is to predict as many class labels of the test dataset as possible correctly. Operational principle of the prediction step is illustrated in fig. 7.4. As the test data is acquired from an

experiment in form of a time discrete sequence, first transformation into feature space needs to be performed. This is done the same way as on the training data by applying multiple different functions on each segment of the test measurement data (see eq. (7.2)). This results in a set of scalar values for each test segment which can be composed in a matrix (see eq. (7.4)). The three test samples within fig. 7.4 for instance belong to the test dataset $X_t = \{\boldsymbol{x}_{t,1}, \boldsymbol{x}_{t,2}, \boldsymbol{x}_{t,3}\}^\mathsf{T}$ with

$$X_t = \begin{bmatrix} 0.85 & 1.60 \\ 6.50 & 1.00 \\ 3.94 & 4.50 \end{bmatrix}. \tag{7.9}$$

During the prediction phase the algorithm determines, which class label is most likely for each of the three test samples. This is done by determining in which area each of the test samples is located. For instance the first sample $\boldsymbol{x}_{t,1}$ lies in the area which belongs to class $c = 1$ according to the previously created decision boundary. Accordingly the second test sample $\boldsymbol{x}_{t,2}$ is assigned class label $c = 3$, because it lies in the area which was determined as most likely for class 3 during the training phase. Only the third test sample $\boldsymbol{x}_{t,3}$ is a special case as it is located directly on the decision boundary between classes $c = 1$ and $c = 2$. So it is not possible to determine which of both classes is to be assigned to this test sample. However this case is very rare as it is usually not likely to have a test sample located directly on the decision boundary. Moreover to resolve this issue simply either class label 1 or 2 could be assigned randomly to this test sample. [44, 45]

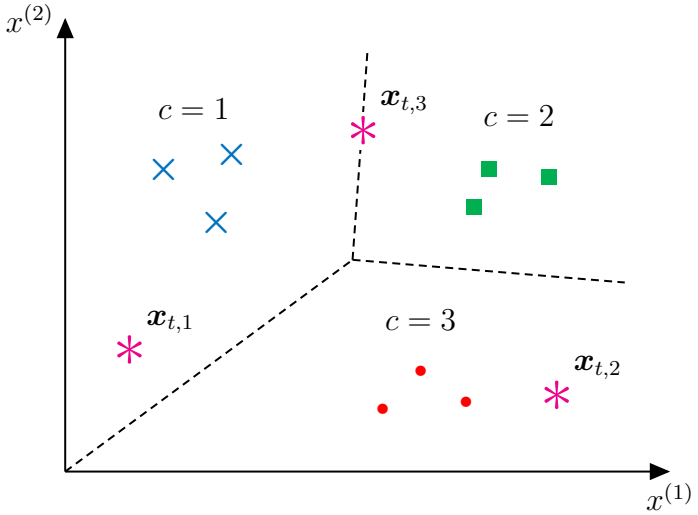


Figure 7.4: Illustration of the prediction phase of a feature-based classifier. Test data is transformed into feature space giving test samples $\boldsymbol{x}_q$ (magenta stars). Position of test samples relative to the decision boundary determines class labels that are assigned to test samples.

The evaluation phase is very similar to the prediction phase. Here a labeled evaluation dataset which is independent from the training dataset is fed into the classification algorithm which assigns class labels to each of the samples within the evaluation dataset. As true classes of the evaluation samples are known it is possible to compute a performance measure of the classification algorithm by comparing true class labels with predicted class labels. [44, 45]

7.1.3 Classification Algorithms

In practice multiple different algorithms have been developed addressing the problem of classifying data based on previously extracted features. The following section gives a brief overview of some of those classification algorithms which are utilized in the later classification of the measurement data of the milling machine. The functional principle of every classifier algorithm is explained and advanatages and disadvanatages of the according algorithm are analyzed. Training and testing of all those algorithms bascially follows the principles described in the previous section, however actual realisation differs in practise based on which kind of algorithm is used. Accordingly complexity and performance of the classifiers are very different. Some of the easier to implement algorithms, like *k-nearest-neighbours* need only very short time for training, but take a long time for testing compared to other more complex to implement algorithms like for instance *support vector machines* which usually need longer time for training, but are very fast in predicting classes of the test dataset. The first algorithm presented is the *k-nearest-neighbours* classifier which classifies test samples based on the distance to neighbouring samples in the training set. Afterwards the *decision tree* classifier as a hierarchical classifier which splits the feature space iteratively in two sub spaces is examined. Based on the basic decision tree an ensemble classsifier called *random forest* utilizing multiple different decision trees for classification is explained. With *AdaBoost* another ensemble method is introduced. AdaBoost combines multiple different weak learners and iteratively trains each weak learner on samples which are misclasssified by the previous weak learner. Afterwards *support vector machines* which try to separate classes in feature space by inserting a hyperplane and maximizing the distance between the hyperplane and neighbouring data samples of both classes are described. As most of the machine learning algorithms include an optimization problem the *stochastic gradient descent* as important method for finding an optimum of a cost function is introduced. Finally the last section covers the *naive bayes* classifier which is based on bayes' theorem and calculates probabilities for the class of a test sample. The following overview over machine learning algorithms is not exhaustive, but rather covers only algorithms which are implemented in the Python based visual programming tool Orange3 [115–117] which is used in the analysis in the later course of this chapter. A more complete list of machine learning algorithms, containing for instance neural networks, deep learning methods and rule induction algorithms can be found in [118].

7.1.3.1 k-Nearest-Neighbours

One of the simplest feature based classification algorithms is the *k-nearest-neighbour* (KNN) classifier which estimates the class label of a test sample by determining class labels of the k closest neighbouring points. Consider the test sample $\boldsymbol{x}_t = (3.8, 3)$ which is plot in fig. 7.5 amongst the training dataset. The training data comprises of samples of two different classes $c = 1$ (blue crosses) and $c = 2$ (red dots). The goal is to predict the most likely class label for the test sample $\boldsymbol{x}_t$ which is done by retrieving class labels of those k training samples that lie closest to the test sample within feature space. In this example $k = 4$ is choosen and as distance measure euclidean distance is computed. As can be seen 3/4 of the nearest neighbours belong to class $c = 2$, whereas only 1/4 of the the neighbouring samples belong to class $c = 1$.

This distribution represents the probability distribution of class labels for the test sample and therefore class label $c_t = 2$ is assigned to the test sample. [40, 119]

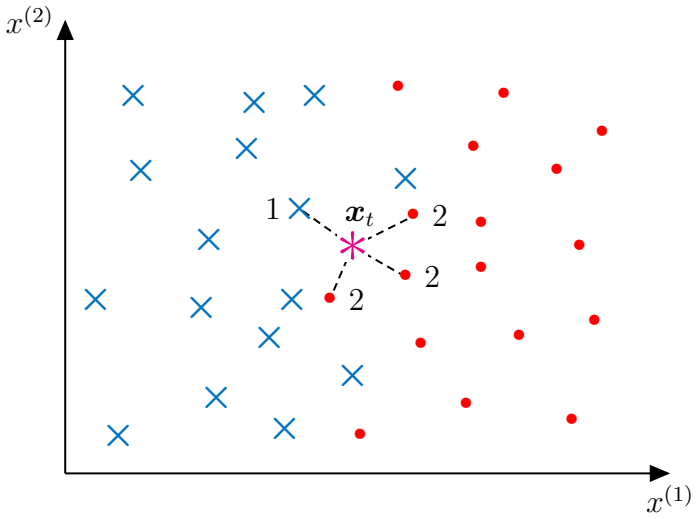


Figure 7.5: Principle of k-nearest-neighbours classification. The algorithm searches within all training samples for the k nearest neighbours of the test sample $\boldsymbol{x}_t$ and assigns that class label to the test sample which occurs most frequently within the k nearest neighbours. In this case 3 of the $k = 4$ nearest neighbours of the test sample belong to class $c = 2$ (red dots), wherefore class label 2 is assigned to the test sample.

Distance to neighbouring training samples of a test sample can be defined by any vector or matrix norm as for instance euclidean norm, manhattan norm, mahalanobis norm or maximum norm (see [40]). Depending on the used metric outlook of the decision boundary changes. Another parameter that influences the decision boundary significantly is the number k of nearest neighbours which are considered during testing. If only the nearest neighbour of a test sample is considered ($k = 1$) the decision boundary can be derived from the Voronoi diagram as shown in fig. 7.6. The Voronoi diagram consist of a net of lines (black lines) which comprise all points in feature space that have an equal distance to two samples in the training set. Again the distance measure can be choosen to be any norm. The decision boundary can now be drawn by looking at the lines which separate samples of different classes. In the simple two-dimensional feature space of the example this gives a decision boundary composed of piecewise linear functions (magenta line). In general the decision boundary fits to the training data very well, but is likely to overfit, if the hyper-parameter k is choosen to small. In this case it is even possible that the decision boundary becomes discontiguous whenever outliers of one class lie within an area in which mainly samples of another class occur. This leads to poor classification performance on test sets differing from the training set. However if k is choosen too big the characteristic structure of the training data might not be recognized well and classification performance goes down as well. So finding the optimal value for the hyper-parameter k is esential for getting good classification results. [40, 44, 120, 121]

A slightly different version of the k-nearest-neighbours classifier utilizes weighting during prediction of the class label of a test sample. Here class labels of neighbouring training samples are weighted according to their distance to the test sample. So training samples which are closer to

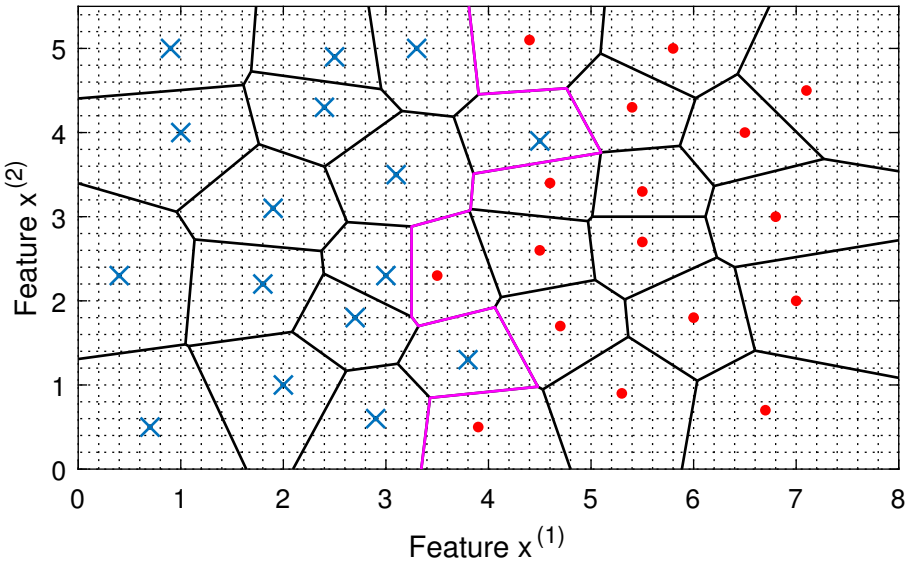


Figure 7.6: Decision boundary (magenta line) created during training of a 1-nearest-neighbour classifier. It can be derived from the Voronoi diagram which consists of all points (black lines) which have equivalent distance to two samples in the feature space.

the test sample have a higher impact on the class of the test sample than neighbouring training samples which are further away from the test sample. [117]

The k-nearest-neighbour classifier is a very simple and easy to implement algorithm, however has the major disadvantage of being computationally expensive during the prediction phase as for each test sample distance measures to every individual training samples needto be computed. Moreover the entire training set needs to be stored which requires a large amount of memory especially for larger datasets. Despite this disadvantage k-nearest-neighbour has the advantage of not needing any time for model training, because in the training phase the training samples are only stored, but no computations are conducted. Moreover the algorithm is capabale of finding even non-linear decision boundaries which fit to the data very well. However this advantages can not compensate for the high computational effort during predition. So k-nearest-neighbour is only appropiate for small datasets. For large datasets other classifiers should be prefered. [40, 45]

7.1.3.2 Decision Tree

Another simple but powerful classifier can be realized utilizing a *decision tree* (DT). Different from most of the other classification algorithms a decision tree only considers one feature at a time rather than classifying a test sample based on all features simulataneously. This is advantageous when operating on high dimensional datasets on which other classifiers are suceptible to overfitting and resulting poor classification performance. The basical operating principle of a decision tree is illustrated in fig. 7.7. As can be seen the decision tree algorithm first determines which of the features separates classes best which is in this case feature $x^{(1)}$, and splits feature space into two subspaces by inserting an axis-parallel line at the threshold

value $\theta_1 = 3.85$. This step is repeated by looking at the second most relevant feature, in this case $x^{(2)}$, and inserting two new splits at $\theta_2 = 3.7$ and $\theta_3 = 2.7$ for the left and right subset of the first split respectively. This process is iterativelly carried on until some stop criterion like a minimum error value is reached. Another stop criterion is size of the tree which is usually measured by either its depth or the number of leaf nodes. Training of such a tree therefore always consists of two major steps, first determination of the next most relevant feature which splits classes optimally and second finding a threshold value θ at which to split the considered sub space of feature space into two new and even smaller subspaces. Determination of the most important feature is done by computing entropy which is a measure of information yield of a feature (see [40]). The better a feature separates two classes the higher is its entropy. To create an entropy optimal tree recursively the feature with highest entropy yield is choosen and a split on this feature is inserted. [40, 45, 120, 122]

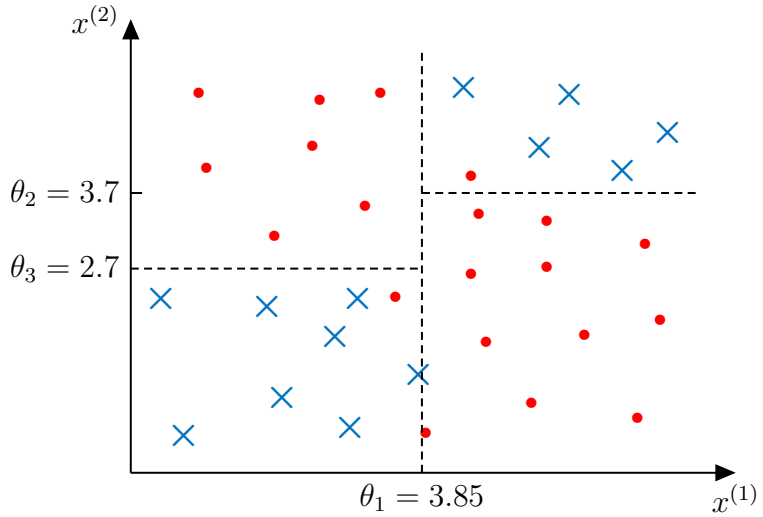


Figure 7.7: Principle of sequential feature space splitting during training of a decision tree classifier. Beginning with the most important feature the feature space is iteratively divided into two subsets dividing classes $c = 1$ (blue crosses) and $c = 2$ (red dots) optimally. This process is repeated until specific stop criterion is reached. The resulting decision boundary (dashed line) is always piecewise parallel to the feature space axes.

The decision tree which results from training on the above example dataset is shown in fig. 7.8. As can be seen the tree contains only two hierachy levels and splits feature space into two subspaces on each split, which leads to four subspaces and therefore four leaf nodes. There are no more splits needed as the decision tree divides the given dataset already very well with low error. Each of the four leaf nodes refer to one of the four subspaces in feature space that result from iteratively splitting of feature space and can be assigned one specific class label $c = 1$ or $c = 2$ depending on the class labels of the majority of samples in each subspace.

During the testing phase the features of the test sample are analyzed in the same order the decision tree has ranked features during training phase and proceeds along one branch based on values of each of the features until one leaf node is reached. The class of this leaf node is then assigned to the test sample. Consider for instance the test sample $\boldsymbol{x}_t = \left(x^{(1)}, x^{(2)}\right) = (4.8, 1)$ which lies in the lower right corner of feature space as shown in fig. 7.9a and therefore should be

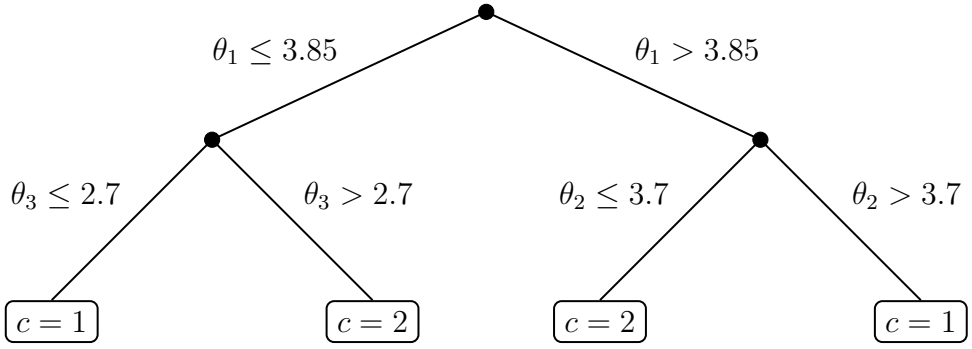


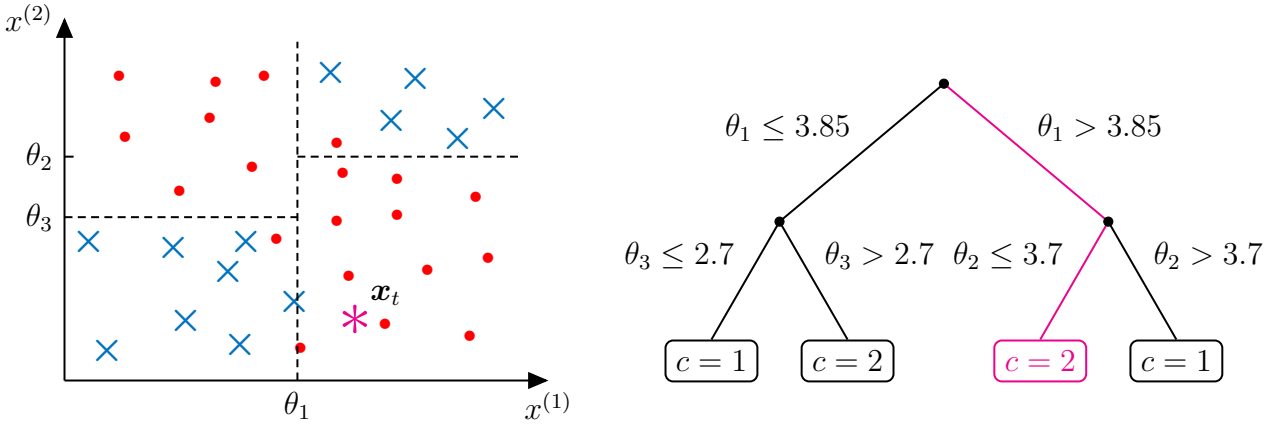
Figure 7.8: Resulting decision tree after training on the above example dataset. The tree consist of two hierarchy levels according to the two considered features. In the first level the tree determines between points to the left and to the right of the vertical split at θ_1. In the next level suspaces above and below two threshold values θ_2 and θ_3 are distguished, whereby the threshold θ_2 accounts for all points to the right of the first split and θ_3 for all points to its left.

classified as class $c = 2$. In the first level the decision tree considers feature $x^{(1)}$ which in this case has the value 4.8 and is therefore bigger than the threshold value $\theta_1 = 3.85$. So the tree has already determined that the test sample lies to the right of the first split. In the subsequent iteration the second feature $x^{(2)}$ is considered, which for the test sample has the value 1. The tree compares this values to the second threshold value $\theta_2 = 3.7$ and recognizes the test sample to be located below the corresponding horizontal split line in the right part of feature space. Therefore the test sample has been located in the lower right corner of feature space, which is equivalent to the third leaf node, and is classified correctly as class $c = 2$. The resulting path in the decision tree is highlighted in fig. 7.9b.

Apart from the described binary classification task where the decision tree had to distiguish between only two classes, it is possible to use a decision tree for a multiclass problem with more than two different classes as well. [123]

In practice there are multiple different implementations of classification trees each dedicated to a specific kind of data. The most common tree algorithm is the *Classification and Regression Tree* (CART), which is able to determine entropy optimal split positions on continuous features as in the example above [124]. Similar to this the *Chi-Square Automatic Interaction Detection* (CHAID) finds entropy optimal split positions by means of a Chi-squared test [125]. Different from CART and CHAID which operate on continous features another tree algorithm, the *Iterative Dichotomiser* (ID3) operates only on discrete features [126]. Extensions of this algorithm are the C4.5 and C5.0 algorithms which allow for additional missing entries in the dataset and remove redundant branches from the tree [127, 128]. [40, 129]

Decision trees have some advantages as they are for instance invariant to different transofmations and especially scaling of the features within the dataset, which makes prior normalization of the dataset unneccessary. Moreover they are easy to interpret by humans and performi well on high dimensional datasets because of the hierarchical model structure which leads to internal feature ranking during training. However decision trees have some disadvantages as well as they



(a) Test sample in feature space with previously trained decision boundary of the tree.

(b) Path the decision tree runs through during prediction of the test samples class label.

Figure 7.9: Illustration of class prediction of a test sample $\boldsymbol{x}_t$ with a decision tree. The tree determines first if the test sample lies right or left of the vertical split at θ_1 and afterwards if it lies below or above the horizontal split at θ_2. Here the test sample is correctly located in the lower right corner and the majority class label $c = 2$ of this region is assigned to the test sample. The corresponding path in the decision tree is highlighted (magenta line).

are for instance sensible to changes in the dataset and therefore unstable. Only small changes in the training dataset can change splitting behaviour of the tree completely which might be unwanted. Moreover the created decision boundary only consists of axis-parallel segments and therefore might not separate classes well which leads to poor clasification performance. In the above example this was no problem as the classes could be separated easily by axis-parallel boundaries. However if the optimal class boundary would be located under some angle relative to the feature space axes the resulting tree would contain many splits resulting in a staircase-shaped decision boundary. Non-linear shaped decision boundaries are therfore not possible to create with decision trees. Regarding computational expense decision trees are fast to train and allow for fast prediction as only a few inequalities need to be solved rather than computing distances between each sample in the training set like the k-nearest-neighbours classifier does. [40, 45, 123, 130]

7.1.3.3 Random Forest

To overcome problems of individual decision trees like easy overfitting to training data and poor classification performance an ensemble method called *random decision forest* has been proposed in [131]. Ensemble methods combine multiple different learners to create one aggregated learning that overcomes performance issues of the individual learners. In this case the random decision forest improves accuracy and reduces susceptibility to overfitting of usual decision trees by fitting multiple different trees to the data and average prediction results of the individual decision trees. Hereby creation of the individual trees is highly randomized to grow trees in the forests as uncorrelated as possible. If no randomization would be performed trees within the forest would look very similar or even identical giving no advantage compared to an individual decision tree.

To grow decision forests of many uncorrleated trees two methods called *bootstrap aggregating* and *random subset selection* are utilized. Bootstrap aggregating or bagging was first proposed in [132] and is an ensemble learning technique which is based on random selection of subsets from the training data. The concept is illustrated in fig. 7.10. Each classifier within the ensemble is trained on a different randomly choosen subset of the training data giving a set of different classifiers for the same training data. While each individual classifier is likely to overfit to the training data the ensemble of different classifiers is not and generalizes much better on unknown test data. The number of individual trees in the ensemble can vary in a wide range and is unknown. Therefore it needs to be optimized by evaluationg classifcation performance of the ensemble with respect to the number of individual trees. [132, 133]

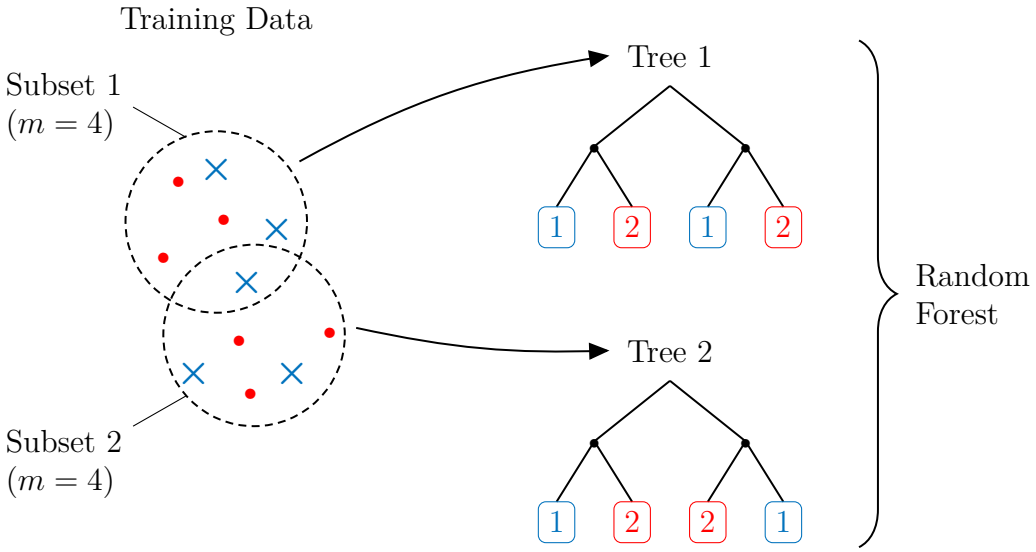


Figure 7.10: Principle of bootstrap aggregating. From the entire dataset multiple subsets (here two subsets of equal size $m = 4$) are derived by random selection of samples. Each subset is used for training of an individual decision tree within the random forest. Because each tree is trained on a different set of training samples correlation between the trees is low and therefore classification performance of the random forest is high.

When growing multiple individual decision trees on different subsets of training data which are created by bootstrap aggregating, performance of the ensmable of trees is much higher than perfomance of the individual tree [131]. However the so created ensemble still has a problem with datasets containing a few samples which are highly important while all other features are less important. In this case individual trees within the esamble are likely to rank the features in the same order and create splits based on the same few important features. This leads to high correlation of the decision trees within the ensemble and results in poor classification performance. A method called *random subset selection* which overcomes this problem was introduced in [134]. To understand the operational principle of this method, it is neccessary to refresh knowledge about feature ranking utilized by individual decision trees when introducing a new split during training. As described in section 7.1.3.2 each split is inserted along the feature which leads to highest information gain. Random subset selection changes this behaviour of individual tree growth by randomly sampling a fixed number of features at each split. The tree

algorithm now maximizes information gain by introducing the split along the most relevant feature within this subspace rahter than the entire feature space. Randomizing the subspaces at each split leads to much less correlated trees within the random forest increasing classification performance. Regarding the number of features [135] suggests $\sqrt{p}$ features of a p-dimensional feature space at each split as a first starting point, however recommends tuning the number of considered features as the optimum value might differ for each problem. [133–136]

Stop criteria for the growth of each individual tree within the random forest can be choosen depending on the problem as well. Usual stop criteria are maximum depth, meaning maximum number of splits of each tree and the minimum size of subsets in the dataset at which no more splits shall be introduced. [117]

Predicting a class of a test sample with help of a trained random forest is as easy as predicting the class label for the test sample with each individual tree in the forest and assigning the class which was predicted by the majority of trees to the test sample. Apart from using random forests for classification problems they are suitable for regression problems as well. [135]

7.1.3.4 AdaBoost

Another ensemble method that combines multiple simple or *weak* classifiers to create one *strong* classifier is the *Adaptive Boosting* (AdaBoost) algorithm first proposed in [137]. Similar to random forests AdaBoost is not an classification algorithm, but a so called *meta-algorithm* which can be applied to nearly every classification algorithm such as decision trees or k-nearest-neighbours classifiers to boost their performance. However AdaBooost should only be applied to weak classifiers and not to more complex ones like state vector machines or neural networks, because here AdaBoost might easily lead to overfitting of the combined classifier to the training data [138]. In practive often used are so called *Decision Stumps* which are single-level decision trees which split the dataset based on one feature and have a prediction accuracy of only slightly more than 50 % [139]. AdaBoost combines a large number of these weak classifiers to generate a strong overall classifier.

In general AdaBoost differs from random forests in the way it applies the weak classifiers to the dataset. While random forests randomly select subsets from the training data by boostrap aggregating (see fig. 7.10) and train multiple different decision trees on that subsets simultaneously, AdaBoost is a sequential algorithm adding only one weak classifier at a time. Based on the classification results of the recently added classifier subsets of the training set are selected and the next weak classifier is trained on this subset. Hereby samples which are misclassified by the previous weak classifier are considered in the training set of the next weak classifier with higher probability than samples that are classified correctly. Therefore AdaBoost can not train all of the contained weak classifiers simultaneosly, but needs to train them sequentially. Besides this sequential determination of a training subset AdaBoost calculates weights for every weak classifier depeding on the classifiers accuracy. The more accurate a classifier is, the more weight is assigned to its classification result in the later prediction phase. [140–142]

Considering T different weak binary classifiers $h_t(x)$ which output 1 for the positive class and -1 for the negative class the combined strong learner created by the AdaBoost algorithm can

be written as

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \tag{7.10}$$

with weights α_t for each of the weak classifiers. So the overall classification result can be obtained by determining the sign of the summed weighted classification results of the weak classifiers. For equal weights α_t this bascially means if the majority of weak classifiers assign label 1 to a test sample the overall output of the AdaBoost algorithm would be 1 and vice versa. Considering different weights prediction reults of the weak classifiers have different impact on the overall predcition output. [140–143]

In the following the training process of an AdaBoost classifier shall be described. Training comprises of the above mentioned steps of choosing a subset of training samples for the weak classifier which is added in the next iteration of the algorithm as well as calculating the weighting factor α_t for the current weak classifier. So it a loop which runs from $t = 1$ to $t = T$ first a weak classifier is added which computes a prediction output considering training subset of size m of all training samples. Selection of this subset is random, but considers different probabilities $D_t(i)$ of each training samples. Here samples with a higher probability are more likely to occur in the training set than samples with low probability. In the first iteration of the loop probabilities are all equal and set to $D_t(i) = 1/m$. As true labels of the training samples are known classification error ε_t for the current classifier can be calculated and used for determining the weight α_t for the current weak classifier as

$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right). \tag{7.11}$$

This definition gives a weight of zero for classifiers with accuracy of 50 %. So whenever a weak classifier is not better than random guessing, its output is not considered during computation of the overall classification output $H(x)$. When classification accuracy of a weak classifier is less than 50 % a negative weight is assigned. This means always the opposite of the prediction result of the weak classifier is considered when calculating the overall classification output. Moreover this definition of the weighting factor α_t weights more accurate classifiers exponentially. This means if the ensemble contains a few good classifiers the overall prediction result is mainly based on these more accurate classifiers, while less accurate classifiers hardly have an impact on the overall prediction outcome. [140–143]

After calculating the weighting factor for the current weak classifier the probability distribution D_t for the training samples needs to be updated. So for each sample within the entire training set the probability for occuring in the training subset for the next weak classifier is updated according to

$$D_{t+1}(i) = \frac{D_t(i)\mathrm{e}^{-\alpha_t y_i h_t(x_i)}}{Z_t}. \tag{7.12}$$

Here $h_t(x_i)$ is the predicted label for the ith sample and y_i the true class of the sample. Z_t is a normalization factor that ensures probabilites of all n training samples sum up to 1. It is computed as

$$Z_t = \sum_{i=1}^{n} D_t(i)\mathrm{e}^{-\alpha_t y_i h_t(x_i)}. \tag{7.13}$$

Illustratively spoken probability of occurance is increased whenever the according sample is misclassified by the current classifier, so whenever the predicted class $h_t(x_i)$ and the true class y_i differ. In this case the exponential term is greater than 1 and probability is increased. In the opposite case of a correctly predicted class label $h_t(x_i)$ and y_i are equal and the exponential term is a fraction smaller than 1 decreasing probability for the according sample. Moreover probabilities of samples are updated considering the weighting factor α_t of the current classifier as well. This means misclassifications of a more accurate classifier are considered more important than misclassifications of a less acurrate classifier. The so calculated probabilities decide which samples are used in the training set of the subsequently added weak classifier. As probabilites are higher for the previously misclassfied samples these are more likely to be part of the training subset for the next classifier and therefore AdaBoost is a method which iteratively tries to detect issues of the current model and takes counter-measures to improve the classifier. [140–143]

The AdaBoost algorithm terminates when the overall classification accuracy reaches 100 % or after a fixed number of iterations [144]. Parameters like the number of iterations and the number T of weak classifiers contained in the ensemble need to be choosen thoroughly to optimize overall classification performance of the ensemble. [138]

Advantages of the AdaBoost algorithm is the increase in classsification performance by combining multiple weak classifiers. Moreover it is not only suitable for binary classification problem, but can be used for multiclass problems as well. AdaBoost is considered one of the most performant out-of-the-box algorithms meaning it yields in very high predcition performance on most machine learning problems without much need for parameter tuning [145]. Major drawback of the AdaBoost algorithm is its high computational complexity. As it involves many computational operations which need to be executed sequentially AdaBoost has a long execution time and needs more memory than most of the other classification algorithms. Apart from this one of the most important disadvantages of AdaBoost is its suceptibility to overfitting to noisy training data. Because the algorithm assigns highest occurance probabilities to outliers it fits subsequent classifiers especially to these training samples which leads to overfitting. Therefore AdaBoost requeires preliminary processing of the dataset. Outliers should be removed and noise reduced as far possible. [138]

7.1.3.5 Support Vector Machine

Support vector machines are state-of-the-art classification models which separate two classes by a linear decision boundary or so called hyperplane in feature space. They were first proposed in [146] and are suitable for both classification and regression problems. Aim of support vector machines is to find an optimal set of two parallel hyperplanes, which are spanned by samples of the training dataset and have maximum distance or *margin* to each other. In between of these hyperplanes no data samples must lie. Samples which span the hyperplanes are called *support vectors* and are sufficient to fully define the decision boundary in form of the two hyperplanes. An example for the operational principle of a support vector machine is illustrated in fig. 7.11. Here two different sets of parallel hyperplanes labeled with H_1 and H_2 separate both classes $c = 1$ (blue crosses) and $c = 2$ (red dots). However the set H_1 of hyperplanes separates classes better than the set H_2 as the distance between the two hyperplanes of H_1 is larger than the

distance of the hyperplanes H_2, so H_1 has a bigger margin and is therefore a better decision boundary for both classes. [45, 147–149]

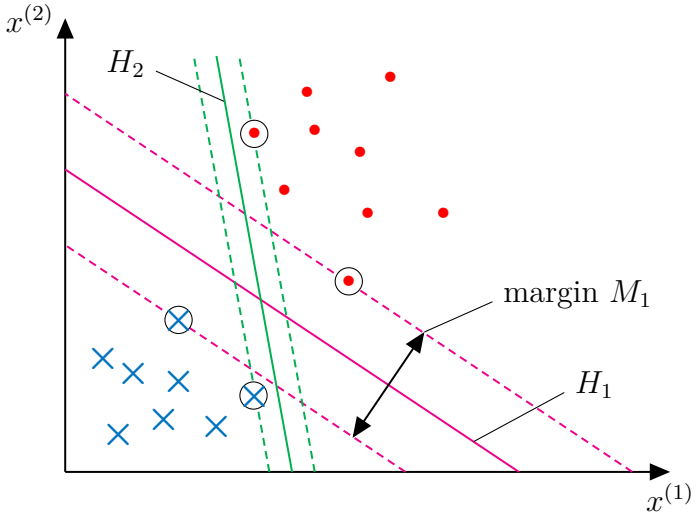


Figure 7.11: Principle of support vector machines on linearly separable classes. Classes $c = 1$ (blue crosses) and $c = 2$ (red dots) are separated by a set of two linear hyperplanes with maximum margin, this means maximum distance between both hyperplanes. Each of the two hyperplanes is spaned by a few data samples of the training dataset, called support vectors (marked with black circles). They are the only samples needed for describing the support vector machine. Here the set H_1 of hyperplanes has maximum margin M_1, while the set H_2 of hyperplanes which linearly separates the two classes as well has a smaller margin and is therfore not the optimal decision boundary.

Training of a support vector machine means finding a set of two hyperplane with maximum distance or margin. In general a single hyperplane $H(\boldsymbol{x})$ can be described as linear function in a p-dimensional feature space by

$$\boldsymbol{w}\boldsymbol{x}^\mathsf{T} + b = 0 \tag{7.14}$$

with normal vector $\boldsymbol{w} \in \mathbb{R}^p$ of the hyperplane and a set of points $\boldsymbol{x} \in \mathbb{R}^p$ in feature space. $b \in \mathbb{R}$ is the constant offset of the hyperplane relative to the coordinate origin. According to this definition of a hyperplane a set of two parallel hyperplanes which lie equidistant and parallel to the original hyperplane and separate both classes can be created by defining an additional boundary condition such as

$$|\boldsymbol{w}\boldsymbol{x}_s^\mathsf{T} + b| = 1. \tag{7.15}$$

This gives a set of two parallel hyperplanes each containing the support vectors $\boldsymbol{x}_s \in \mathbb{R}^p$ for the specific class. These two hyperplanes are illustrated in fig. 7.11 as dashed lines parallel to the hyperplane H_1 and can be expressed by the two equations

$$\boldsymbol{w}\boldsymbol{x}_k^\mathsf{T} + b \leq -1 \quad \text{when } c_k = 1 \tag{7.16}$$

$$\boldsymbol{w}\boldsymbol{x}_k^\mathsf{T} + b \geq 1 \quad \text{when } c_k = 2. \tag{7.17}$$

Again $\boldsymbol{w}$ is the normal vector, $\boldsymbol{x}_k$ is the kth data sample and b is a scalar describing distance between hyperplane and coordinate origin. According to these two equations feature space is

split into three regions by the hyperplanes, one between of both hyperplanes and one below or above of each hyperplane. The regions below or above of the two hyperplanes belong to class $c = 1$ respectively $c = 2$ exclusively which means training data samples lying on or below the hyperplane satisfying the first equation belong to class $c = 1$ and samples on or above the hyperplane satisfying the second equation belong to class $c = 2$. As mentioned no training samples must lie in between of the two hyperplanes which makes it sometimes impossible to fit a set of two hyerplanes to a dataset. However there exists a method to allow for outliers which lie in the region between both hyperplanes or even on the wrong side of the set of planes. This is achieved by introducing *slack variables* which allow for such outliers, but penalize every outlier by an additional term in the later optimization problem [40]. [40, 45, 146, 147, 150]

To derive a mathematical formulation for the margin of the two hyperplanes first the distance of an arbitrary point $\boldsymbol{x}_k$ in feature space to the hyperplane which lies exactly in the middle of the two hyperplanes needs to be evaluated. This distance can be calculated as

$$d = \frac{|\boldsymbol{w}\boldsymbol{x}_k^\intercal + b|}{\|\boldsymbol{w}\|}. \tag{7.18}$$

When calculating distance between a support vector $\boldsymbol{x}_s$ and the hyperplane in the middle the above equation can be simplified according to the boundary condition in eq. (7.15) to the form

$$d_{\mathrm{SV}} = \frac{1}{\|\boldsymbol{w}\|}. \tag{7.19}$$

The margin is exactly twice d_{SV} as this is the distance between a support vector and the hyperplane in the middle. Hence margin can be written as

$$M = \frac{2}{\|\boldsymbol{w}\|}. \tag{7.20}$$

Maximizing the margin can therefore be achieved by minimizing the norm of the normal vector $\|\boldsymbol{w}\|$. To simplify this task the normal vector can be expressed as linear combination of all training samples $\{\boldsymbol{x}_j \in X : j = 1, 2, \ldots, n\}$

$$\boldsymbol{w} = \sum_{c_j=1} \alpha_j \boldsymbol{x}_j - \sum_{c_j=2} \alpha_j \boldsymbol{x}_j. \tag{7.21}$$

Instead of minimizing the norm of the normal vector now the weights $\alpha_1, \alpha_2, \ldots, \alpha_n$ need to be optimized. For this the above expression is inserted in eq. (7.16) giving an updated description for the set of the two hyperplanes

$$\sum_{c_j=1} \alpha_j \boldsymbol{x}_j \boldsymbol{x}_k^\intercal - \sum_{c_j=2} \alpha_j \boldsymbol{x}_j \boldsymbol{x}_k^\intercal + b \leq -1 \quad \text{when } c_k = 1 \tag{7.22}$$

$$\sum_{c_j=1} \alpha_j \boldsymbol{x}_j \boldsymbol{x}_k^\intercal - \sum_{c_j=2} \alpha_j \boldsymbol{x}_j \boldsymbol{x}_k^\intercal + b \geq 1 \quad \text{when } c_k = 2. \tag{7.23}$$

This set of equations is fed into an optimization algorithm such as gradient descent [151], newton's method in optimization [152] or a method utilizing lagrangian multipliers [153]. Output of the optimization algorithm is a set of optimal parameters $\alpha_1, \alpha_2, \ldots, \alpha_n$ and an optimal constant b

giving the two hyperplanes which linearly separate both classes with maximum margin. [40, 45, 146, 147, 150]

Predicting the class label of a test sample $\boldsymbol{x}_t \in \mathbb{R}^p$ with a support vector machine consisting of a set of optimal parameters is as easy as inserting coordinates of the test sample in eq. (7.22) for the two hyperplanes and evaluating the output. This is identical to determining location of the test sample relative to the two hyerplanes which form the decision boundary. Whenever eq. (7.22) gives an output smaller or equal -1 the test sample class is predicted as $c = 1$ and when the output is greater or equal 1 the test sample class is predicted as $c = 2$. One problem occurs, when test samples lie in between of both hyperplanes which form the decision boundary as these can not be classified by the support vector machine. [45, 146]

As already mentioned support vector machines can not only be used for creating linear decision boundaries, but offer a very efficient way of determining non-linear decision boundaries as well. The solution for this problem is called *kernel trick* and was first introduced in [154]. Basic idea of the kernel trick is to transform a training dataset $X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p$ with a non-linear decision boundary into a higher dimensional dataset $X' = \{\boldsymbol{x}'_1, \boldsymbol{x}'_2, \ldots, \boldsymbol{x}'_n\} \in \mathbb{R}^q$ with $q > p$ in which the decision boundary is linear again. So the kernel trick involves first transformation into the higher dimensional feature space and then training of a support vector machine on this transformed dataset. This gives the transformed version of the hyperplanes from eq. (7.22) as

$$\sum_{c_j=1} \alpha_j \varphi(\boldsymbol{x}_j) \cdot \varphi(\boldsymbol{x}_k)^\intercal - \sum_{c_j=2} \alpha_j \varphi(\boldsymbol{x}_j) \cdot \varphi(\boldsymbol{x}_k)^\intercal + b \leq -1 \quad \text{when } c_k = 1 \tag{7.24}$$

$$\sum_{c_j=1} \alpha_j \varphi(\boldsymbol{x}_j) \cdot \varphi(\boldsymbol{x}_k)^\intercal - \sum_{c_j=2} \alpha_j \varphi(\boldsymbol{x}_j) \cdot \varphi(\boldsymbol{x}_k)^\intercal + b \geq 1 \quad \text{when } c_k = 2. \tag{7.25}$$

Whenever a test sample shall be classified it is transformed into the higher dimensional feature space as well and then classified in this space. However in practice an explicit transformation of the dataset into the higher dimensional feature space is not neccessary as the kernel trick allows for implicit transformation of the dataset by replacing dot products in the lower dimensional space X with kernel functions $k : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$. This is according to mercer's theorem as proposed in [155] equivalent to the transformation $\varphi : \mathbb{R}^p \to \mathbb{R}^q$ which transforms the dot product into the higher dimensional feature space X'. So mercer's theorem states

$$\varphi(\boldsymbol{x}_j) \cdot \varphi(\boldsymbol{x}_k)^\intercal = k(\boldsymbol{x}_j, \boldsymbol{x}_k). \tag{7.26}$$

Often used kernel functions are the polynomial kernel

$$k(\boldsymbol{x}_j, \boldsymbol{x}_k) = (\boldsymbol{x}_j \cdot \boldsymbol{x}_k^\intercal)^d, \quad d \in \{2, 3, \ldots\}, \tag{7.27}$$

the Gaussian kernel

$$k(\boldsymbol{x}_j, \boldsymbol{x}_k) = \mathrm{e}^{-\frac{\|\boldsymbol{x}_j - \boldsymbol{x}_k\|^2}{\sigma^2}}, \quad \sigma > 0, \tag{7.28}$$

the hyperbolic tangent kernel

$$k(\boldsymbol{x}_j, \boldsymbol{x}_k) = 1 - \tanh \frac{\|\boldsymbol{x}_j - \boldsymbol{x}_k\|^2}{\sigma^2}, \quad \sigma > 0 \tag{7.29}$$

and the radial basis function kernel (see [156])

$$k(\boldsymbol{x}_j, \boldsymbol{x}_k) = f\left(\|\boldsymbol{x}_j - \boldsymbol{x}_k\|\right). \tag{7.30}$$

Replacing the dot products in eq. (7.24) with one of the above kernel funtions gives the generalized support vector machine

$$\sum_{c_j=1} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k) - \sum_{c_j=2} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k) + b \leq -1 \quad \text{when } c_k = 1 \tag{7.31}$$

$$\sum_{c_j=1} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k) - \sum_{c_j=2} \alpha_j k(\boldsymbol{x}_j, \boldsymbol{x}_k) + b \geq 1 \quad \text{when } c_k = 2. \tag{7.32}$$

which allows for classification of non-linearly separable classes. The simplified case of linearly separable classes can be represented by inserting the linear kernel $k(\boldsymbol{x}_j, \boldsymbol{x}_k) = \boldsymbol{x}_j \cdot \boldsymbol{x}_k^\mathsf{T}$ into this equation. [40, 45, 147]

Support vector machines are charaterized by a very high classification performance on most of the classification tasks as they guarantee a globally optimized solution rather than only a local optimum as it might occur with some other classification algorithms. Moreover support vector machines can find even non-linear decision boundaries efficiently by applying the kernel trick. Regarding computational expense support vector machines enable very fast prediction of class labels by inserting the test sample in the equation of the hyperplane and evaluating the sign of the ouput. However training of a support vector machine requires much more computational ressources as a usually large optimization problem under consideration of multiple boundary conditions needs to be solved. Moreover prediction is not as fast as described when the classification problem is not a binary problem, but rather a multi-class problem. As support vector machines are not able to directly distiguish between more than two classes the multi-class problem first needs to be transformed into multiple binary classification problems which are then fed into support vector machines. Different strategies for reducing a multi-class problem into single-class problems are described in [157] and [158]. Another disadvantage of support vector machines is their black-box charater as it is difficult or sometimes impossible to interpret the parameters of a trained support vector machine. However they are still more easy to interpret than for instance nerual networks. [40, 147]

7.1.3.6 Stocastic Gradient Descent

Stochastic gradient descent (SGD) is another meta-algorithm often used in classification algorithms. It is used to solve optimization problems such as minimizing a cost function. Similar to AdaBoost (see section 7.1.3.4) it is not a dedicated classification algorithm, but can be applied to other classifiers like support vector machines, perceptrons and neural networks (see [45] and [40]). Thinking back to the support vector machines in section 7.1.3.5 classification was achieved by inserting a linear decision boundary in form of two hyperplanes, which needed to be placed optimally by minimizing a cost function. Stochastic gradient descent aims to minimize such a cost function. [117, 159, 160]

Consider a cost function $J : \mathbb{R}^v \to \mathbb{R}$ for the parameter vector $\boldsymbol{w} \in \mathbb{R}^v$ with

$$J(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} J_i(\boldsymbol{w}) \tag{7.33}$$

which evaluates the cost of a set of parameters $\boldsymbol{w}$ by cumulating the cost for each of the n data samples in the training set. An example for such a cost function is given in fig. 7.12. Note that parameter space is only one-dimensional, so there is only one free parameter $w \in \mathbb{R}$ and the cost $J(w)$ can be plotted as function of this parameter. As can be seen the cost function has a global minimum at $w = w^*$. This minimum is found via stochastic gradient descent. [160]

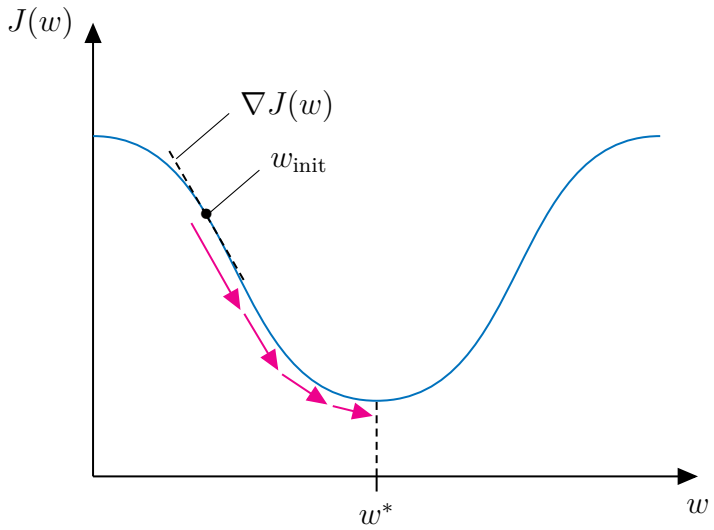


Figure 7.12: Principle of stochastic gradient descent for minimizing a cost function $J(\boldsymbol{w})$. In this case parameter space is one-dimensional and the cost function $J(w)$ has only one global minimum at w^* which is found by iteratively descending along the approximated gradient $\nabla J(w)$ of the cost function for the current parameter w. Descent is controlled by the learning rate η which is decreased after each iteration until the parameter convergences to its optimum value for which the cost function is minimal.

To find a minimum by gradient descent first either random values $\boldsymbol{w}_{\text{init}}$ for the parameter vector $\boldsymbol{w}$ are choosen or all parameters are set to 0. Now the gradient $\nabla J(\boldsymbol{w})$ of the cost function is evaluated for each training sample and the parameter vector is updated by descending the gradient

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla J(\boldsymbol{w}^{(t)}) = \boldsymbol{w}^{(t)} - \eta \sum_{i=1}^{n} \nabla J_i(\boldsymbol{w}^{(t)})/n. \tag{7.34}$$

This minimizes cost in the next iteration slightly as the gradient always points in direction of the global or local minimum closest to the according training sample. So descending the gradient is equal to finding a set of new parameters $\boldsymbol{w}^{(t+1)}$ which lead to slighly lower cost. Descent along the gradient can be controlled with the *learning rate* η which specifies how much the parameter vector is changed in each iteration of the algorithm. As this algorithm needs to compute the gradient for each of the n training samples in each iteration of the descent this so called *batch gradient descent* is not very efficient regarding computational expense especially

Listing 7.1: Pseudocode of stochastic gradient descent for minimizing a cost function by iterative updating of the parameter vector. [160]

```
w = w_init;
eta = eta_init;
while cost > approximative_minimum_cost
        random_shuffle_training_set();
        for each sample in the shuffled training set
                w = w - eta * gradient(J(w));
        endfor
        eta = eta * decrease_rate;
endwhile
```

when the training dataset is very large. To resolve this major drawback and make gradient descent real-time capable even on larger datasets the true gradient is approximated by the local gradient at one specific training sample giving

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} - \eta \nabla J_i(\boldsymbol{w}^{(t)}). \tag{7.35}$$

In each step of the descent the gradient is evaluated at another training sample. Updating of the parameter vector $\boldsymbol{w}$ is repeated until each sample within the training set has been selected once. The order of this evaluation can be either sequential or random. As the random process might not always directly find a sufficiently good mimumum it is neccessary to run the entire stochastic gradient descent multiple times with different initial values for the learning rate η and the parameter vector $\boldsymbol{w}$. Here the learning rate can be decreased in each of this overall runs to make the algorithm converge to the approximative minimum. Moreover the entire dataset is shuffled before each run to further randomize the gradient descent and increase probability of finding a sufficiently good minimum. The pseudocode for the entire process comprising multiple different stochastic gradient descents is shown in listing 7.1. [45, 159, 160]

Though approximation of the gradient by only one sample is highly efficient, it can sometimes be inaccurate. Therefore another version of the stochastic gradient descent computes an average gradient not based on only one training sample, but a small number of different randomly choosen training samples called *mini-batch*. This increases accuracy of the gradient approximations and results in smoother convergence. [45, 160]

According to [161] and [162] stochastic gradient descent almost always converges to a global minimum if the cost function is convex or pseudoconvex or to a local minimum otherwise. Convergion requires the learning rate η to decrease with an appropiate rate.

Adavantages of stoachstic gradient descent are the easy implementation and the high efficiency of the algorithm even on very large datasets. Usually only a few runs, for instance 1 to 10 are neccessary to find a sufficiently good minimum [163]. Disadvanatages are the high number of hyper-paramaters which need to be tuned be the user. Moreover stocastic gradient descent requires the cost function to be as less distorted in parameter space as possible. This means features should be scaled for instance into a range of $[0, 1]$ to achieve highest performance of the stochastic gradient descent. [123]

7.1.3.7 Naive Bayes

The naive bayes classifier determines class labels of test samples by calculating and comparing conditional probalilities for each possible class under consideration of additional observations. It was first proposed by Thomas Bayes in his *Essay Towards Solving a Problem* [164] which was published in 1763. Given a finite set of nominally scaled class labels $C = \{c_1, c_2, \ldots, c_m\}$ and a test sample $\boldsymbol{x}_t \in \mathbb{R}^p$ propability of the test sample belonging to class $c = j \in C$ can be calculated as

$$P(c = j|\boldsymbol{x}_t) = \frac{p(\boldsymbol{x}_t|c = j)P(c = j)}{\sum_{j \in C} p(\boldsymbol{x}_t|c = j)P(c = j)}. \tag{7.36}$$

This is equation is known as *Bayes Theorem*. Within this formula $P(c = j|\boldsymbol{x}_t)$ is the posterior probability which determines how likely it is that the test sample $\boldsymbol{x}_t$ belongs to class $c = j$. $P(c = j)$ is the a priori probability for the class $c = j$. Consider a training set of size $n = 10$ containing 7 samples of class $c = 1$ and 3 samples of class $c = 2$. Prior probabaility of class 1 would be $P(c = 1) = 7/10$ and prior probability of class $c = 2$ would be $P(c = 2) = 3/10$. So the prior probabaility is derived from the distribution of classes in the training set without considering any properties of the test sample. Another term that needs to be derived from the training set is the likelihood $p(\boldsymbol{x}_t|c = j)$ for the class label $c = j$. The likelihood for class $c = j$ can be calculated from the probability densitiy function of each feature in the test sample. This is illustrated in fig. 7.13. Here two different classes $c \in C = \{1, 2\}$ are considered and the test sample is for the sake of simplicity only one-dimensional and has the measured value $x_t = 5$. The diagram shows two different probability density functions, one for the distribution of x_t in each of the two classes. As can be seen from the distributions the likelihood for class $c = 1$ under the oberservation $x_t = 5$ is $p(c = 1|x_t = 5) = 0.15$ and the corresponding likelihood for class $c = 2$ is $p(c = 2|x_t = 5) = 0.01$. With both likelihood and a priori probability the posterior probability $P(c = j|\boldsymbol{x}_t)$ can be calculated for both classes. The naive bayes classifier now evaluates the a posterior probabilites for all class labels under consideration of the oberservation and assigns the class label c^* with highest posterior probability to the test sample. So the naive bayes classifier can be written as

$$c^* = \arg\max_{j \in C} P(c = j|\boldsymbol{x}_t). \tag{7.37}$$

Whenever the test sample dimension is larger than $p > 1$ likelihoods can be decomposed as

$$p(\boldsymbol{x}_t|c = j) = \prod_{i=1}^{p} p\left(x_t^{(i)}|c = j\right). \tag{7.38}$$

With this the naive bayes formula can be rewritten as

$$P(c = j|\boldsymbol{x}_t) = \frac{P(c = j)\prod_{i=1}^{p} p\left(x_t^{(i)}|c = j\right)}{\sum_{j \in C} P(c = j)\prod_{i=1}^{p} p\left(x_t^{(i)}|c = j\right)} = \frac{P(c = j)\prod_{i=1}^{p} p\left(x_t^{(i)}|c = j\right)}{p(\boldsymbol{x}_t)}. \tag{7.39}$$

The denumerator $p(\boldsymbol{x}_t)$ of this expression is called evidence and is in a classification task usually not computed because it is identical for all classes. So if just the class $c*$ of a test sample shall be

predicted only the numerator determines which class is most likely for the test sample. However if the true posterior probabilities need to be computed, the denumerator has to be considered as well. [40, 44, 45, 165–167]

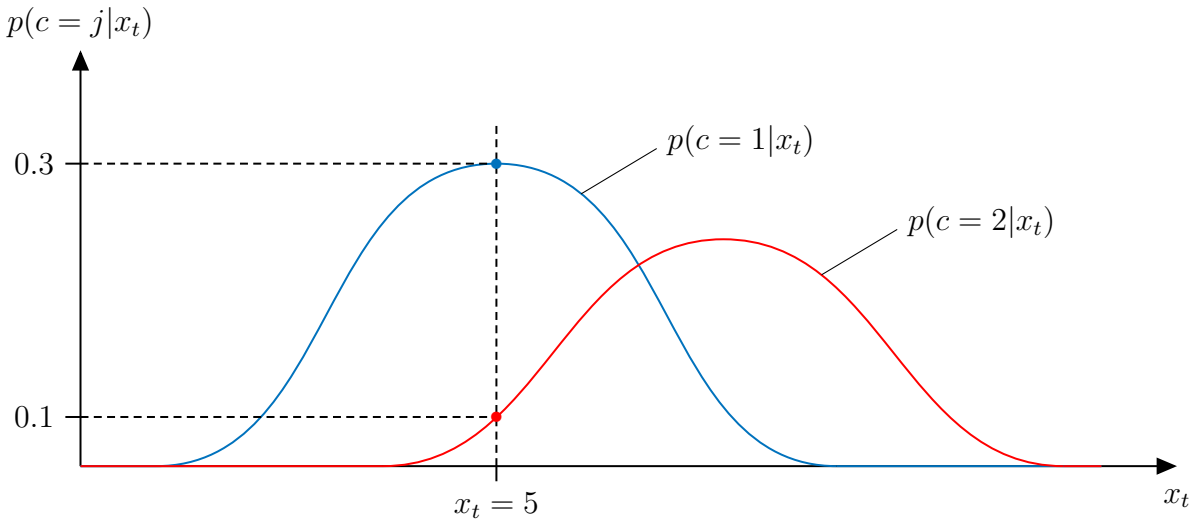


Figure 7.13: Estimation of likelihoods $p(c = j|\boldsymbol{x}_t)$ from the probability densitiy functions of features. Here only one feature and two different classes $c = 1$ and $c = 2$ are considered. likelihoods for a given test sample or observation $x_t = 5$ are $p(c = 1|x_t = 5) = 0.3$ for class 1 and $p(c = 2|x_t = 5) = 0.1$ for class 2. If probability density functions are not available they can be estimated from the training set.

To better understand the naive bayes classifier an example for an easy classification problem shall be given. Consider a group of 1000 people of which 600 are male and 400 are female. All people are asked whether they like cars or not and it was evaluated which person has long hair and which one has short hair. This gives the training set shown in table 7.1. Now the sex of an independent person which likes cars and has short hair shall be determined by calculating the posterior probabilities for this person being male or female. So the test sample can be written as $\boldsymbol{x}_t = (\text{likes cars}, \text{short hair})^\intercal$.

Table 7.1: Example training set derived from a group of 1000 people with 600 males and 400 females. All people are asked if they like cars or not and it is determined if the person has long hair or short hair.

Sex	Likes cars	Dislikes cars	Long hair	Short hair	Total
Male	480	120	30	570	600
Female	160	240	360	40	400

From the training set the prior probrability for the test person to be male can be computed as $P(c = \text{male}) = 600/1000 = 0.6$ and for the test person to be female as $P(c = \text{female}) =$

$400/1000 = 0.4$. After this the likelihoods can be computed as

$$\begin{aligned}
&p(\text{likes cars}|\text{male}) = 480/600 = 0.8 && p(\text{dislikes cars}|\text{male}) = 120/600 = 0.2\\
&p(\text{likes cars}|\text{female}) = 160/400 = 0.4 && p(\text{dislikes cars}|\text{female}) = 240/400 = 0.6\\
&p(\text{long hair}|\text{male}) = 30/600 = 0.05 && p(\text{short hair}|\text{male}) = 570/600 = 0.95\\
&p(\text{long hair}|\text{female}) = 360/400 = 0.9 && p(\text{short hair}|\text{female}) = 40/400 = 0.1.
\end{aligned}$$

Before inserting the according likelihoods and a priori probabilities into eq. (7.39) to retrieve the posterior probabilities for the test person being male or female the evidence is calculated as

$$\begin{aligned}
p(\boldsymbol{x}_t) = {} & P(c = \text{male}) \cdot p(\text{likes cars}|\text{male}) \cdot p(\text{short hair}|\text{male})\\
& + P(c = \text{female}) \cdot p(\text{likes cars}|\text{female}) \cdot p(\text{short hair}|\text{female}) = 0.472.
\end{aligned} \tag{7.40}$$

Now the posterior probabilities can be calculated. The posterior probability for the test person being male is

$$P(c = \text{male}|\boldsymbol{x}_t) = \frac{P(c = \text{male}) \cdot p(\text{likes cars}|\text{male}) \cdot p(\text{short hair}|\text{male})}{p(\boldsymbol{x}_t)} = 0.9661 \tag{7.41}$$

and the posterior probability for the test person being female is

$$P(c = \text{female}|\boldsymbol{x}_t) = \frac{P(c = \text{female}) \cdot p(\text{likes cars}|\text{female}) \cdot p(\text{short hair}|\text{female})}{p(\boldsymbol{x}_t)} = 0.0339. \tag{7.42}$$

So the posterior probability for the test person being male is 96.61 % and for being female is 3.39 %. Therefore the naive bayes classifier would classify the test person which likes cars and has short hair as male.

Conclusively it can be said that the naive bayes classifier is despite its simplicity a very powerful classifier which can outperform modern machine learning techniques in some use cases such as text classification and spam filtering. Other than for instance the support vector machine naive bayes performs especially well on multiclass problems and is easy to understand and implement. Moreover it is heighly efficient as prior probabilities, likelihoods and evidences need to be estimated only once for a giving training set. Following predictions can be made by calculating the posterior probabilities for all classes given a test sample. This is very fast which is why the naive bayes classifier is suitable for real time applications. A major disadvantage of the naive bayes classifier is the fact that features need to be statistically independent which is in practice often not the case. Moreover features need to be discrete. If features are continous they are estimated with a normal distribution which requires according preprocessing of the dataset to ensure a sufficiently accurate approximation when the normal distribution is calculated. Alternatively continuous features can be discretized using histograms or a chi-squared test. Another problem regards feature values which occur in the test set, but are not contained in the training set. As this missing feature value is not contained in the naive bayes model no posterior probability for the test sample can be computed making a classification of this sample impossible. [40, 168]

7.2 Extraction of Statistical Features

As explained in section 7.1 classification is based on features which are extracted from the time domain data. The following section covers feature extraction from the measurement data of the milling machine acquired in section 4.5. Extraction is conducted via a Matlab script that can be found in listing A.11. Within this script a subroutine which conducts the actual feature calculation for each time series segment is called. The source code of this subroutine is shown in listing A.12. The according flowchart of the overall feature extraction algorithm is depicted in fig. 7.14. The algorithm first loads the measurement data from all three test cycle runs as well as the labelsets from section 5.3 containing cut positions and true class labels of each segment within the three measurement datasets. Afterwards it iterates through all segments within the measurement data and calls the feature calculation subroutine from listing A.12 to calculate all 49 different feature values for the current segment. The so calculated row is then appended to the output dataset. This process is repreated for each of the three test cycle runs giving three output datasets containing 49 columns and approximately 2200 rows each. The exact number of rows depends on the number of segments in the measurement data for the corresponding test cycle run and are shown in table 5.3. Important to mention is that feature extraction is only conducted on the current signal for the L2-phase of the DMU main power line. As mentioned in chapter 5 this is the only signal which is not affected by any of the non-controllable components like control cabinet heat exchanger, machine lubrication pump and oil-air lubrication pump. So by extracting features from the L2 current signal only it is possible to simplify the classsification problem to the 12 controllable machine states instead of determining all 96 possible machine states as defined in section 4.3. After calculation of all features the datasets are searched for any non numerical values (NaN) or values that are positive or negative infinity. As these values would lead to problems in the further process they are replaced with zeros. Finally the extracted features of the second and third test cycle run are merged into one larger training dataset and the true class labels known from the previously loaded labelsets are applied to each row of the datasets. This gives two labeled datasets, one for training and one for testing, whereby the testing set makes up one third of the entire amount of samples gathered during data acquistion. Both datasets are then stored as CSV file and in a special tabular format required by the visual programming tool Orange3 which is used in the subsequent steps of the classification task. For clarification of how the datasets are composed table 7.2 shows an extract from the training dataset containing the first few and the last sample as well as the first five and the last feature.

Core of the presented feature extraction algorithm is calculation of features for each of the segments of measurement data. These features are statistical features meaning they abstract each segment by a set of statistical measures which is computed for the entire segment. So each feature corresponds to a scalar value unique for the given segment. One features could be for example the mean value

$$\bar{s} = \frac{1}{q} \sum_{j=1}^{q} s_j \tag{7.43}$$

of the time series data $\boldsymbol{s} = \{s_1, s_2, \ldots, s_j, \ldots, s_q\}$ within the segment. Here q corresponds to the number of data samples in the segment and s_j to the jth measured L2 current value within the segment. Despite the fact that the so extracted values are unique for each sample there is a

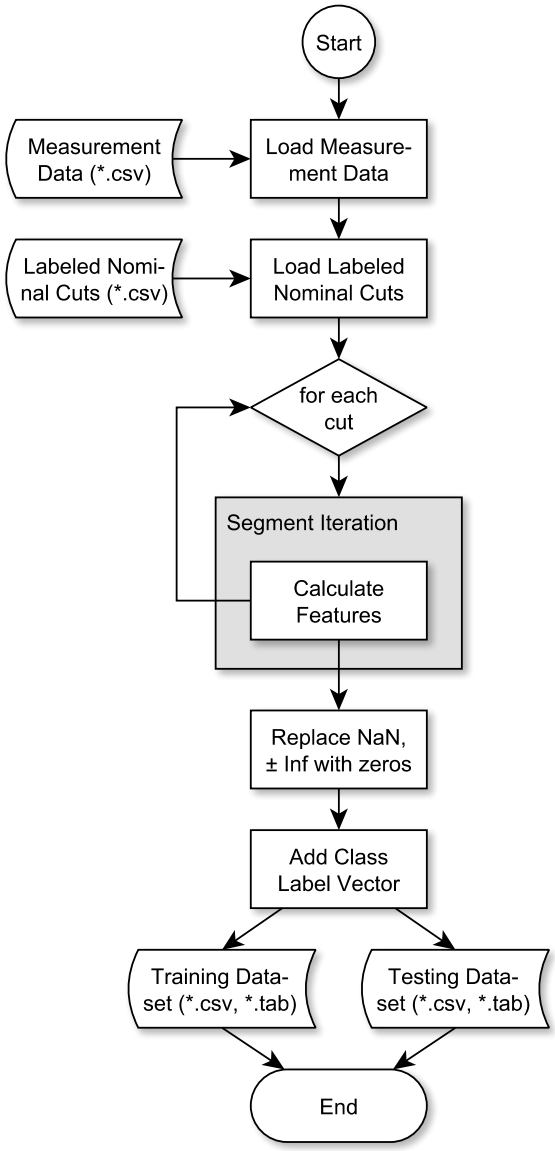


Figure 7.14: Flowchart of the feature extraction algorithm. The algorithm extracts a set of 49 features for each segment within the cut and labeled measurement data from all three test cycle runs. Features and class abels are merged into two different datasets, one for testing which contains features from the first test cycle run and one for training comprising features from the second and third test cycle run.

Table 7.2: Extract from the created training datatset. The entire training dataset contains 4398 rows or samples each referring to one segment in the measurement data. Each row is composed of 49 statistical values or features extracted from the corresponding segment and the true class label of this segment.

Row No.	**Features**							**Class**
	n	x_q	μ_2	μ_3	μ_4	...	x_{THD}	
1	880	7.2192	4.1342×10^{-4}	2.4659×10^{-6}	6.6800×10^{-7}	...	1.7017	0
2	1142	10.3079	45.3944	1.3676×10^{3}	4.9326×10^{4}	...	−4.3473	1
3	101	7.2345	1.3445×10^{-4}	4.0141×10^{-8}	4.8579×10^{-8}	...	−5.5881	0
4	1142	10.1106	44.6239	$1.3220 \times 10^{+3}$	4.6847×10^{4}	...	−13.2185	1
⋮	⋮	⋮	⋮	⋮	⋮		⋮	⋮
4398	57	8.7478	2.9998	3.1976	20.2808	...	−1.1694	11

high correlation between feature values for segments belonging to the same class. So segments belonging to the same class have very similar feature sets which is exploited for later classification of new test samples.

Apart from the already mentioned mean value many other statistical features can be extracted from the measurement data segments. The following two tables table 7.3 and table 7.4 show 49 different features which are extracted from the mesaurement data. Moreover the tables show how the features are mathematically defined and how calculation can be implemented in Matlab. Within the tables the current time segment of measurement data is called $\boldsymbol{s} = \{s_1, s_2, \ldots, s_j, \ldots, s_q\}$. Summations are usually carried out over the entire q data samples within the time segment. Only the sums for features 35 to 39 are different as here the summands are the numbers r of occurances of overshoots, undershoots, rises and falls in the segment. The value r can differe between the five features. Despite not being exhaustive the presented list of features contains all features found in related literature [3, 4, 12–15, 84, 91, 169–173]. Features of the measurement data can be either extracted from the time domain representation of the signal (see table 7.3) or from the frequency domain (see table 7.4). An explicit conversion of the measurement signal from time to frequency domain by means of a fast fourier transform (FFT) is not conducted, but instead special Matlab commands are used which implicitely retrieve frequency domain features from the time domain signal.

Reason for the relatively high number of extracted features is missing prior knowlegde about eligibility of specific features for the given classification problem. Depending on the outlook of the measurement data some combinations of features might abstract the measurement data much better than other sets of features. Finding the optimal set of features is an essential process step and is conducted later in this section. To provide a large enough search space for this task as many features as possible are extracted at this point regardless of any prior assumptions about how well different features might perform on the given measurement data. With the two different datasets created in this section subsequent process steps can be performed in the next sections to fit, evaluate and compare different classification algorithms.

Table 7.3: Overview of extracted features in the time domain. [3, 4, 12–15, 84, 91, 169–173]

No.	Feature Name	Calculation	Matlab Command
0	Number of Samples	$q = \lvert \boldsymbol{s} \rvert$	q = length(s)
1	1st Order Moment (Mean)	$\bar{s} = \frac{1}{q}\sum_{j=1}^{q} s_j$	sq = mean(s)
2	2nd Order Central Moment (Variance)	$\mu_2 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^2$	mu2 = moment(s,2), mu2 = var(x,1)
3	3rd Order Central Moment	$\mu_3 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^3$	mu3 = moment(s,3)
4	4th Order Central Moment	$\mu_4 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^4$	mu4 = moment(s,4)
5	5th Order Central Moment	$\mu_5 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^5$	mu5 = moment(s,5)
6	6th Order Central Moment	$\mu_6 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^6$	mu6 = moment(s,6)
7	7th Order Central Moment	$\mu_7 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^7$	mu7 = moment(s,7)
8	8th Order Central Moment	$\mu_8 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^8$	mu8 = moment(s,8)
9	9th Order Central Moment	$\mu_9 = \frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^9$	mu9 = moment(s,9)
10	4th Order Cumulant	$c_4 = \mu_4 - 3\mu_2^2$	-
11	5th Order Cumulant	$c_5 = \mu_5 - 10\mu_3\mu_2$	-
12	6th Order Cumulant	$c_6 = \mu_6 - 15\mu_4\mu_2 - 10\mu_3^2 + 30\mu_2^3$	-
13	Skewness	$v = \mu_3/\sqrt{\mu_2^3}$	v = skewness(s)
14	Kurtosis	$w = \mu_4/\sqrt{\mu_2^2}$	w = kurtosis(s)
15	Standard Deviation	$\sigma = \sqrt{\frac{1}{q}\sum_{j=1}^{q}(s_j - \bar{s})^2}$	sigma = std(s,1)
16	Relative Standard Deviation	$s_{\mathrm{RSD}} = \sigma/\bar{s}$	srsd = std(s,1)/mean(s)
17	Interquartile Range	$s_{\mathrm{IQR}} = \mathrm{Q}_{.75} - \mathrm{Q}_{.25}$	siqr = iqr(s)
18	Median	$s_{\mathrm{med}} = \begin{cases} s_{(q+1)/2}, & q\,\text{odd} \\ \frac{1}{2}\left(s_{q/2} + s_{q/2+1}\right), & q\,\text{even} \end{cases}$	smed = median(s)
19	Mean Absolute Deviation	$d_{\mathrm{mean}} = \frac{1}{q}\sum_{j=1}^{q}\lvert s_j - \bar{s}\rvert$	dmean = mad(s)
20	Median Absolute Deviation	$d_{\mathrm{med}} = \frac{1}{q}\sum_{j=1}^{q}\lvert s_j - s_{\mathrm{med}}\rvert$	dmed = mad(s,1)
21	Minimum Value	$s_{\mathrm{min}} = \min_{j=1,2,\ldots,q}\{s_j\}$	smin = min(s)
22	Maximum Value	$s_{\mathrm{max}} = \max_{j=1,2,\ldots,q}\{s_j\}$	smax = max(s)
23	Peak to Peak Value	$s_{\mathrm{p2p}} = s_{\mathrm{max}} - s_{\mathrm{min}}$	sp2p = peak2peak(s)
24	Maximum Absolute Value	$s_{\mathrm{peak}} = \max_{j=1,2,\ldots,q}\lvert s_j\rvert$	speak = max(abs(s))
25	Square Root Value	$s_{\mathrm{SRV}} = \left(\frac{1}{q}\sum_{j=1}^{q}\sqrt{\lvert s_j\rvert}\right)^2$	ssrv = (1/length(s)* sum(sqrt(abs(s)))).^2
26	Average Rectified Value	$s_{\mathrm{ARV}} = \frac{1}{q}\sum_{j=1}^{q}\lvert s_j\rvert$	sarv = 1/length(s)*sum(abs(s))
27	Root Mean Square	$s_{\mathrm{RMS}} = \sqrt{\frac{1}{q}\sum_{j=1}^{q}\lvert s_j\rvert^2}$	srms = rms(s)
28	Root Sum of Squares	$s_{\mathrm{RSS}} = \sqrt{\sum_{j=1}^{q}\lvert s_j\rvert^2}$	srss = rssq(s)
29	Crest Factor	$k_s = s_{\mathrm{peak}}/s_{\mathrm{RMS}}$	ks = peak2rms(s)
30	Shape Factor	$k_f = s_{\mathrm{RMS}}/s_{\mathrm{ARV}}$	kf = srms/sarv

Table 7.3 (continuation) : Overview of extracted features in the time domain. [3, 4, 12–15, 84, 91, 169–173]

No.	Feature Name	Calculation	Matlab Command
31	2nd Hjorth Parameter (Mobility)	$\Psi_2 = \sqrt{var\left(\dot{s}\right)/var\left(s\right)}$	[hj2,~] = HjorthParameters(s)
32	3nd Hjorth Parameter (Complexity)	$\Psi_3 = \Psi_2\left(\dot{s}\right)/\Psi_2\left(s\right)$	[~,hj3] = HjorthParameters(s)
33	Area under Curve	$s_{\mathrm{AUC}} = \frac{b-a}{2q}\sum_{j=1}^{q}\left(s_j + s_{j+1}\right)$	auc = trapz(s)
34	Area under Autocorrelation Curve	$s_{\mathrm{AAC}} = \frac{b-a}{2q}\sum_{j=1}^{q}\left(s_{\mathrm{AUC},j} + s_{\mathrm{AUC},j+1}\right)$	aac = trapz(xcorr(s))
35	Sum of Overshoots	$\bar{s}_{\mathrm{OS}} = \sum_{i=1}^{r} s_{\mathrm{OS},i}$	sos = sum(overshoot(s))
36	Sum of Undershoots	$\bar{s}_{\mathrm{OS}} = \sum_{i=1}^{r} s_{\mathrm{US},i}$	sus = sum(undershoot(s))
37	Sum of Rise Times	$\bar{t}_r = \sum_{i=1}^{r} t_{r,i}$	srt = sum(risetime(s))
38	Sum of Fall Times	$\bar{t}_f = \sum_{i=1}^{r} t_{f,i}$	sft = sum(falltime(s))
39	Sum of Slew Rates	$\bar{s}_{\mathrm{SR}} = \sum_{i=1}^{r} s_{\mathrm{SR},i}$	ssr = sum(slewrate(s))

7.3 Development of Feature-Based Classifiers

After extraction of features from the measurement data and composing of the training and testing sets the actual classsification problem can be resolved. All following process steps are conducted in Orange3 a Python based open-source data visualization, machine learning and data mining toolkit [115, 116]. It incorporates a graphical user interface which simplifies data manipulation, model setup and evaluation significantly compared to other machine learning toolkits which require much more complicated manual scripting [174]. Orange3 workflows comprise different widgets for data manupulation, data visualization, model fitting and evaluation which can be dragged to the workspace and modified afterwards. The Orange3 workflow used in the subsequent sections is shown in fig. 7.15. It contains several different widgets which serve different purposes. Widgets labeled with *Training Data* and *Testing Data* load the datasets created by the feature extraction algorithm in section 7.2. The training dataset is then directly fed into a *Column Selection* and afterwards in a *Preprocessor* widget. Here first a manual preselection of features takes places and afterwards features are standardized and an automated selection of features based on a feature importance measure is conducted. This will be explained in detail in section 7.3.2. Apart from this the model contains seven different classification models, more precisely one k-nearest-neighbour model, one tree model, one random forest, one AdaBoost model, one support vector machine, one stochastic gradient descent model and one naive bayes model. All those models are explained in detail in section 7.1.3. They are connected to the *Scoring* widget and are automatically fit to the preprocessed training data. Tuning of the hyper-parameters of each model is explained in section 7.3.3. After model training the test set is loaded from the *Testing Data* widget and fed into the *Scoring* widget as well. Because the preprocessor is connected to the *Scoring* widget as well the test data is processed the same way as the training data which is neccessary to enable classification of test samples. The test data is

Table 7.4: Overview of extracted features in the frequency domain. Any feature is extracted for all three power line phases L1, L2 and L3. [3, 4, 12–15, 84, 91, 169–173]

No.	Feature Name	Matlab Command
0	Average Power	abp = bandpower(s)
1	3 dB Half-Power Bandwidth	hpbw = powerbw(s)
2	99 % Occupied Bandwidth	ocbw = obw(s)
3	Mean Normalized Frequency	fmean = meanfreq(s)
4	Median Normalized Frequency	fmed = meanfreq(s)
5	Spurious Free Dynamic Range	ssfdr = sfdr(s)
6	Signal to Noise and Distortion Ratio	ssinad = sinad(s)
7	Signal to Noise Ratio	ssnr = snr(s)
8	Total Harmonic Distortion	sthd = thd(s)

fed into all seven classifier models and the predicted labels are compared to the true labels of the test data. Based on this different performance scores are measured and confusion matrices are created for all classifiers giving a more detailed insight into the performance of the classifiers. Details of this step are presented in section 7.4. Besides the Orange3 implementation the same classification problem is solved with two Python scripts shown in listing A.18 and listing A.19. They utilize the machine learning library *scikit-learn* (see [123] and [175]) to conduct the same process steps of the Orange3 implementation, however feature an automatic grid search for finding optimal hyperparameters. For the sake of simplicity and because of very similar results of both approaches the following process is described by means of the Orange3 implementation instead of the scikit-learn scripts.

7.3.1 Standardization of Extracted Features

Before any classifier can be fit to the training dataset or any classification can be performed on the test set, it is neccessary to *standardize* both training and testing dataset. Standardization is required for proper operation of many classification algorithms. It comprises centering of the features by removal of the mean or median value and scaling of each feature to either unit variance or unit range. [123]

Consider for instance the two-dimensional dataset in fig. 7.16a. As can be seen both features are significantly different scaled. While the data is broadly distributed along the first feature $x^{(1)}$ it is almost not spread along the second feature $x^{(2)}$. This is due to different scaling of both features and leads to poor performance when fitting most classification algorithms to this kind of misscaled dataset. So standardization of this dataset is needed to increase classification performance. The same dataset after scaling to unit variance and centering by removal of the mean value is shown in fig. 7.16b. [40]

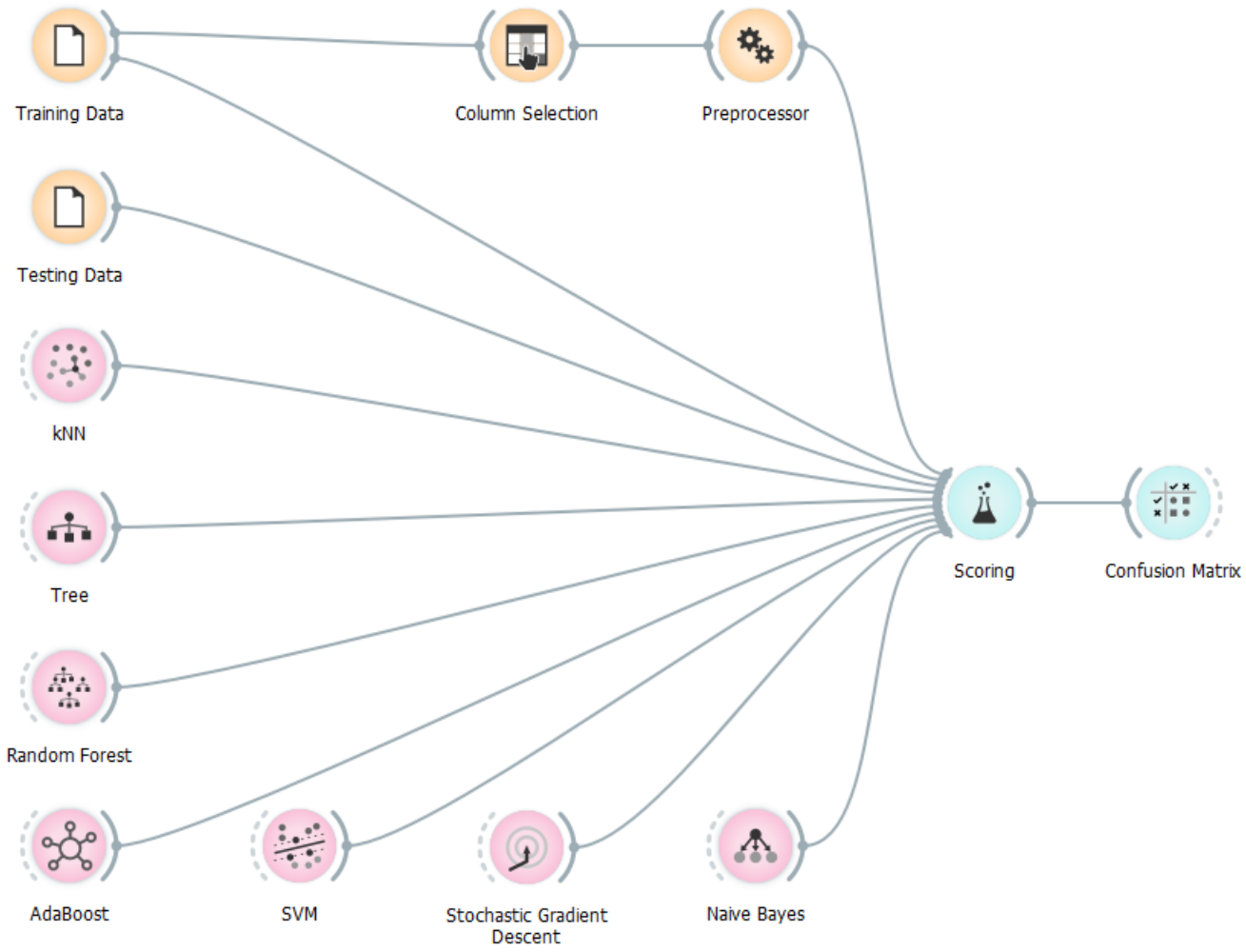


Figure 7.15: Overall workflow of the feature based classification in Orange3. First training data and test datasets are loaded. Both datasets are then standardized and relevant features are selected. Based on this preprocessed training data multiple different classifier models are fit and afterwards their performance is evaluated on the independent test dataset.

To standardize a dataset

$$X = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p \tag{7.44}$$

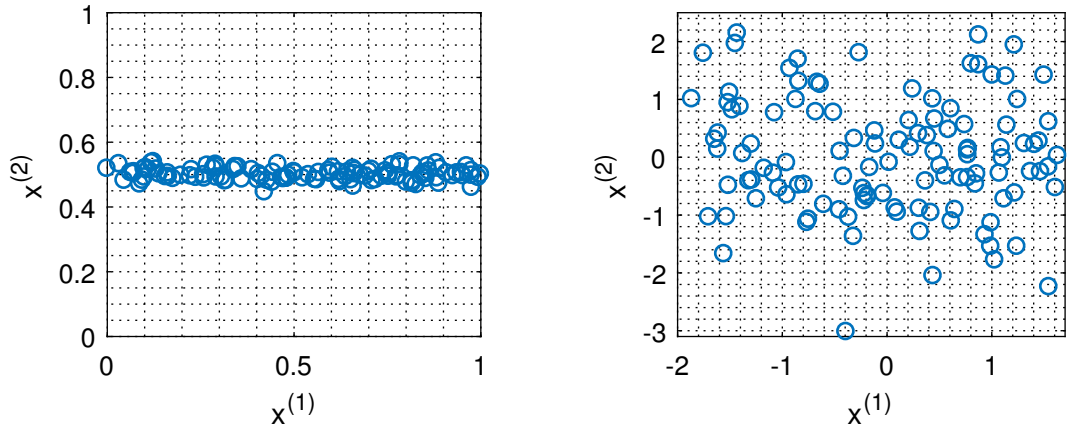
first each feature $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ with $i = \{1, 2, \ldots, p\}$ needs to be centered. This is done by eiter subtracting the mean value

$$\bar{x}^{(i)} = \frac{1}{n} \sum_{k=1}^{n} x_k^{(i)} \tag{7.45}$$

or the median value

$$x_{\text{med}}^{(i)} = \begin{cases} x_{(n+1)/2}, & n \text{ odd} \\ \frac{1}{2}\left(x_{n/2} + x_{n/2+1}\right), & n \text{ even} \end{cases} \tag{7.46}$$

from each column of the data matrix $\boldsymbol{x}^{(i)}$. Now features are centered around zero and scaling to either unit range or unit variance can be performed. If features are scaled to unit range all



(a) Dataset before scaling.

(b) Dataset after scaling to unit variance and centering around mean.

Figure 7.16: Standardization of an unequally scaled dataset. Standardization scales all features to either unit range or unit variance and centers them around mean or median. The left diagram shows a two-dimensional dataset with unequally scaled features. The right diagram shows the dataset after centering around the mean values and scaling to unit variance. Here both features are similarly distributed.

values in the data matrix lie in the range $[0, 1]$ and if they are scaled to unit variance all features within the data matrix have roughly the same variance of $\sigma = 1$. Scaling a feature $\boldsymbol{x}^{(i)} \in \mathbb{R}^n$ to unit range is done by subtracting the minimum value $x^{(i)}_{\text{min}} = \min_{k=1,2,\dots,n}\{x^{(i)}_k\}$ from each data value $x^{(i)}_k$ in that column and divinding by the distance between minimum value and maximum value $x^{(i)}_{\text{max}} = \max_{k=1,2,\dots,n}\{x^{(i)}_k\}$ of that column giving the new data values

$$\hat{x}^{(i)}_k = \frac{x^{(i)}_k - x^{(i)}_{\text{min}}}{x^{(i)}_{\text{max}} - x^{(i)}_{\text{min}}}, \quad k = \{1, 2, \dots, n\}\,. \tag{7.47}$$

for the ith column of the data matrix. Scaling of the ith features to unit variance on the other hand is conducted by subtracting the mean value $\bar{x}^{(i)}$ and dividing by the standard deviation $\sigma^{(i)}$ of the feature giving the new data values

$$\hat{x}^{(i)}_k = \frac{x^{(i)}_k - \bar{x}^{(i)}}{\sigma^{(i)}}, \quad k = \{1, 2, \dots, n\}\,. \tag{7.48}$$

Of course subtraction of the mean value is only neccessary if no previous centering has been conducted. Otherwise subtraction of the mean value does not affect the scaling operation. [40, 123]

Which method for scaling is suitable for a given problem depends on the outlook of the features. If features are approximately equally distributed, meaning they have approximately zero variance, scaling to unit variance is not reasonable as this would lead to very high data values with large fluctuations in the scaled dataset. Instead equally distributed features should always be scaled to unit range. However if features can be approximated by a normal distribution scaling to unit variance is appropiate. In this case distributions of all p features have roughly the same outlook which significantly improves classification performance. [40]

Standardization of the dataset is carried out in the Orange3 workflow by means of the *Preprocessor* widget. Here the subwidget *Normalize Features* which is shown in fig. 7.17 is enabled and the options are set to *Center by Mean* and *Scale by SD* which means each feature is centered around its mean value and scaled to unit variance. [117]

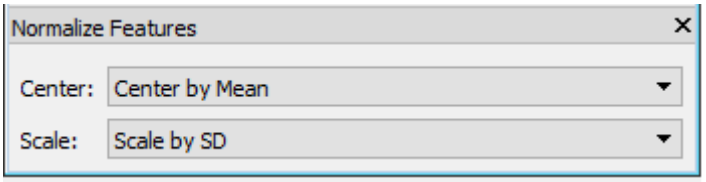


Figure 7.17: Normalization subwidget in the preprocessing widget in Orange3. This subwidget scales each feature in the dataset to unit variance and centers it around its mean value.

7.3.2 Selection of Relevant Features

After extraction of the features listed in section 7.2 it is neccessary to reduce dimensionality of the dataset by selecting only the most relevant features in order to achieve optimal classification performance. This section first explains why dimensionality reduction is neccessary and afterwards explains in detail how feature selection is conducted by means of exploiting internal feature ranking of decision trees. Finally implementation of feature selection within the Orange3 workflow presented in fig. 7.15 is described.

To be able to classify samples of an unseen test set with reasonable performance a decision boundary needs to be fit to given training data (see section 7.1.2). This decision boundary should be as simple as possible to improve classification performance on the test set, but must be complex enough to correctly separate classes of the training set in feature space. For this a sufficient amount of different features which describe training samples is needed. If for instance only one feature would be used it could be that this feature is not able to distiguish between all classes in the training set. Accordingly more features needs to be extracted and considered during classification. If these features are sufficient in separating all classes in the training set, performance on the training set increases as shown in fig. 7.18 (red line). Increasing the number of features, this means dimensionality of the dataset even more creates a very complex decision boundary that is highly fit to the training data giving a very high classification performance on this training set. However performance on the test set is very low, when dimensonality is too high as the highly complex decision boundary is not able to generalize well to unseen training data which might differ slightly from the training data. Accordingly there is an optimal number p^* of features which give a decision boundary that is on one hand able to separate classes of the training set satisfactory and on the other hand generalizes well to unseen training data. The problems arising from a too high dimensional feature space are often refered to as *curse of dimensionality*. [45, 176, 177]

Reason for increasing performance on the training set with increasing dimensionality is the growing sparsity of samples in feature space. Consider a training set with 1000 samples. Given a one-dimensional feature space of length 5 units leads to a densitiy of $1000/5 = 200$ samples per unit. Spreading the same samples in a two dimensional feature space with an area of $5 \times 5 = 25$

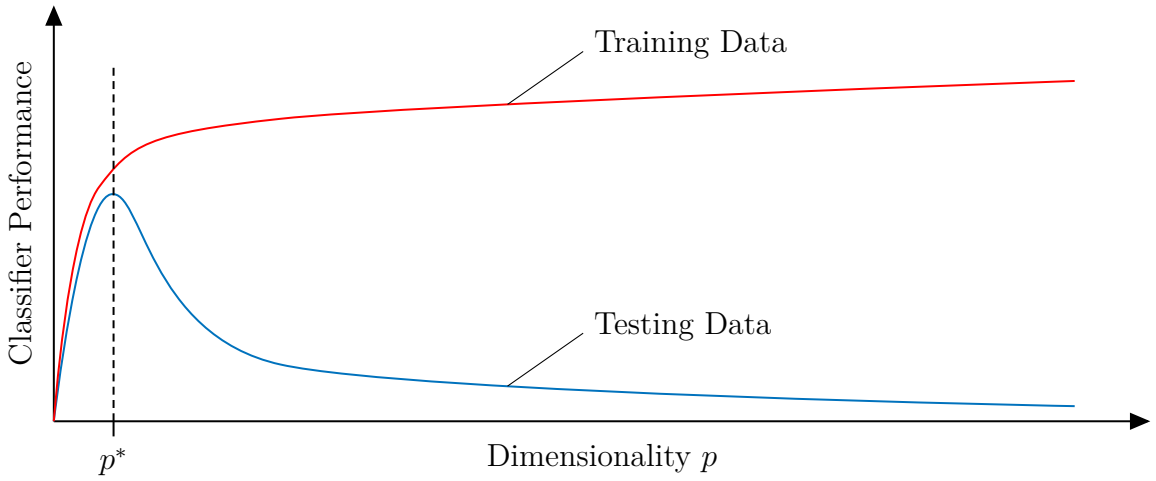


Figure 7.18: Classifier performance on both test and training set versus dimensionality p of the data set clarifies curse of dimensionality. A too low number of features is usually not sufficient for proper classification of unkown test samples. However a too high number of features leads to highly non-linear decision boundaries which overfit to the training data leading to poor classification performance on unknown test data. Depending on the particulars of the datatset there is an optimal number p^* of features leading to highest classification performance on the test set. [176]

units decreases densitiy of the samples to $1000/25 = 40$ per unit. Increasing dimensionality to 3 gives a cube of volume $5 \times 5 \times 5 = 125$ units which reduces density of the training set to only $1000/125 = 8$ samples per unit. This decrease in density of the training set makes fitting of a hyperplane in higher dimensions much easier than in lower dimensions because it is less likely for a training sample to be located on the wrong side of the hyperplane. However transforming the hyperplane into a lower dimensional feature space gives a highly non-linear decision boundary which is overfit to the training data and generalizes not adequately to an unseen test set, which is why performance on the test set decreases with too many features used for classification. Such a casse of a highly overfit decision boundary is illustrated in fig. 7.19. [45, 176]

Another reason for bad performance on the test set is unequal sparsity of the test set in a high dimensional feature space. Here sparsity is much higher in the center of feature space than at its surface, meaning most of the samples lie directly at the surface while the center of feature space is nearly empty. From this arise problems as samples that lie far off from center of feature space are harder to classify than samples that lie close to the center. This is because feature values of these remote samples deviate strongly from the class averages making it harder to assign a specific class to them. Cause for decreasing sample density in the center of feature space is the change in the ratio of hypercube volume to volume of the unit sphere around the center of the hypercube. To clarify this consider the two-dimensional feature space shown in fig. 7.20. Here the unit sphere around the center is a circle with radius $r = 0.5$ units and the hypercube is a square with edge length of $a = 1$ unit. So the volume (area) of the center sphere is $V_S = \pi r^2 \approx 0.785$ and the volume (area) of the hypercube is $V_C = a^2 = 1$. So the sphere volume makes up 78.5 % of the hypercube volume when only two dimensions are considered. Therefore the center sphere contains most of the test samples making classification easy. Repeating the

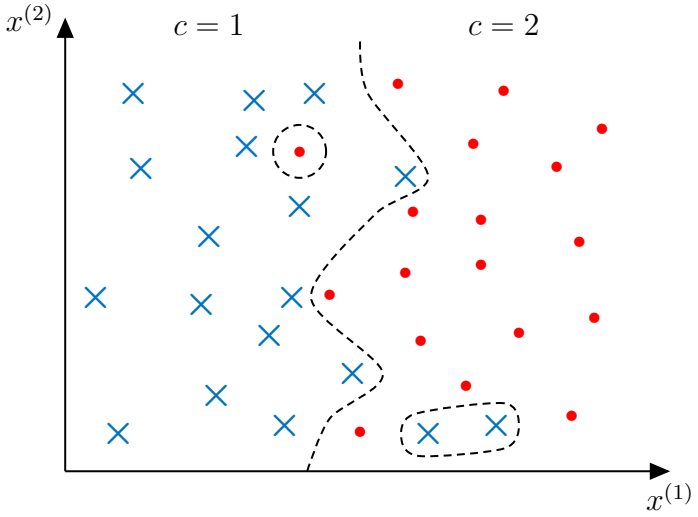


Figure 7.19: Highly non-linear decision boundary as result of overfitting to the training data in a high dimensional feature space. In the higher dimension the decision boundary is linear, however when projected onto a two-dimensional space it becomes highly non-linear and shows strong overfitting to the training data which leads to poor classification performance on unseen test data. [176]

same calculation for three dimensions gives $V_C = 1$ and $V_S = 4/3\pi r^3 \approx 0.524$. So as can be seen volume of the hypercube stays constant while volume of the center sphere and therefore the proportion of samples lying in the center of the hypercube decreases. For an infinite dimensional feature space the volume of the center sphere approaches zero and all samples lie outside of the center and on the hypercube surface making it hard to classify them. [45, 176]

To avoid the problems arising from too high dimensional feature spaces the number of features should be kept low. The optimal number of features p^* which gives maximum performance on the test set and performs reasonable on the training set depends on different aspects and varies for each classification problem. First the amount of training samples determine how many features can be considered before overfitting occurs. In general it can be said that the more training samples are available the higher dimensionality of the feature space can be choosen. Moreover the classifier used for classifying test samples affects optimal number of features. Classifiers that build highly non-linear decision boundaries such as k-nearest-neighbours tend to overfit to training data easily. When such a classifier is used dimensionality of feature space should be lower compared to the case of classifiers which fit linear decision boundaries such as support vector machines or decision trees. Finally the individual outlook of the training data and the decision boundary needed for distiguishing between classes in the training set influences the optimal number p^* of features. [45, 176]

To find the optimal number and combination of features for a given classsification problem for example an exhaustive grid search could be performed. This tests different numbers and combinations of features against the classifier performance and decides for the combination of features which lead to highest performance. These kind of feature selection methods which incorporate the classifier model is called *wrapper-methods*. Another type of feature selection

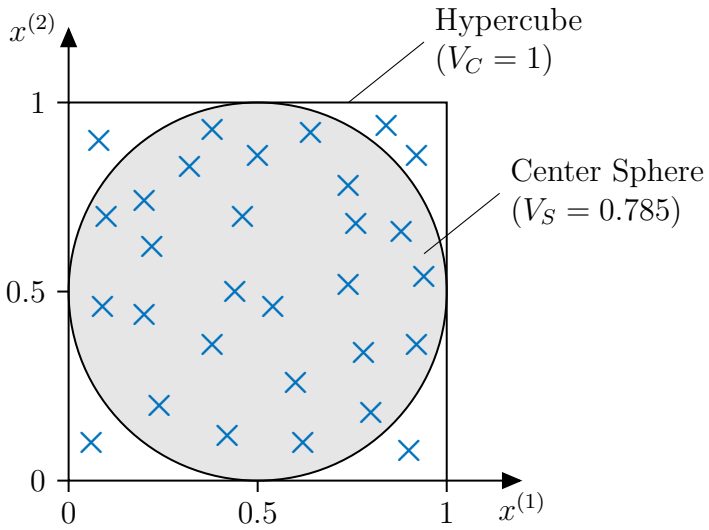


Figure 7.20: Illustration of the number of samples close to the center of feature space compared to samples at the edges of feature space. In a low dimensional feature space like here ($p = 2$) most of the samples lie close to the center of feature space within the unit hypersphere around the center and are easy to classify. Samples outside the unit hypersphere deviate much from the class average and are therefore more unlikely to be classsified correctly. When dimensionality of feature space increases the volume of the hypersphere decreases while the feature space volume V_C stays constant. Accordingly the number of samples in the center region declines and the majority of samples lies outside of the unit hypersphere leading to poor classification performance. [176]

method which operates regardless of the actual classifier is called *filter-method*. Methods of this kind determine quality of features on different measures as for example variance and preselect features which have for instance high variance and therefore might be more suitable for distiguishing between classes than features with low variance. However filter methods are not able to detect dependencies between features which might lead to problems when the dataset contains some features which have high variance but are strongly correlated. Apart from this selection of relevant features can be conducted by utilizing internal feature ranking of the decision tree algorithm. As mentioned in section 7.1.3.2 decision trees internally rank features and split feature space along the most important features by introducing axis-parallel decision boundaries along the selected features. Decision trees can not only be used as classifiers, but also for feature selection by exploiting their internal feature ranking. After training of the decision tree on a given training set the most important feature can be found directly after the root node of the tree, followed be the second most relevant feature in the second tree level and so on. By looking at the first r hierarchical levels of the tree the r most relevant features can be selected. Internal ranking of the features within the decision tree is done by comparing the *information gain* of all features for each split that is introduced to the tree. This measure describes how much information is won by introducing a split along a specific value of a specific feature. By comparing the information gain for split alogn all features the feature giving maximum information gain is selected. [123, 178, 179]

To understand information gain first the concpet of *entropy* needs to be introduced. Entropy is

a measure of impurity of a dataset meaning it describes how much a set of samples containing a specific class is contaminated by samples of other classes. High entropy is equal to high information content of a dataset. Consider a labeled dataset

$$X_L = (X, \boldsymbol{c}) = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p \times \{c_1, c_2, \ldots, c_k, \ldots, c_n\} \tag{7.49}$$

of size n with class labels $c_k \in \{1, 2, \ldots, l, \ldots, m\}$. Entropy for such a dataset can be defined as

$$H\left(X_L\right) = -\sum_{l=1}^{m} p(c_k = l) \log_2 p(c_k = l) \tag{7.50}$$

with the probability

$$p(c_k = l) = \frac{|\{c_k \in \boldsymbol{c} | c_k = l\}|}{n} \tag{7.51}$$

of a sample having the class label $c_k = l \in C$ within the labeled dataset. If only one specific class label $c_k = l$ occurs in the dataset probability for this class label is 1 and probabilities for all other class labels are 0. Hence in this case entropy is 0 bit which means the dataset is completely pure and contains only samples of one class. If all m different classes occur with equal probabilities entropy of the dataset is 1 bit which is the maximum value possible meaning the dataset has maximum impurity. Given this definition of entropy information gain can be defined as amount of information gained by inserting a split along the specific value ν_i of feature $\boldsymbol{x}^{(i)}$ with $i = \{1, 2, \ldots, p\}$. Probability for the feature $\boldsymbol{x}^{(i)}$ having the value $l \in \{1, 2, \ldots, \nu_i\}$ is

$$p(\boldsymbol{x}^{(i)} = l) = \frac{\left|\{\boldsymbol{x}^{(i)} \in X | \boldsymbol{x}^{(i)} = l\}\right|}{n}. \tag{7.52}$$

With this information gain of inserting a split at $\boldsymbol{x}^{(i)} = l$ can finally be computed as

$$g_i = H\left(X_L\right) - \sum_{l=1}^{\nu_i} p(\boldsymbol{x}^{(i)} = l) H\left(X_L | \boldsymbol{x}^{(i)} = l\right). \tag{7.53}$$

[40, 178, 179]

To better understand the concept of information gain an example shall be given. Consider the labeled training datatset X_L shown in fig. 7.21 which contains 16 samples of class $c = 1$ and 17 samples of class $c = 2$. As can be easily seen, the classes can be separated linearly by introducing a vertical split at $x^{(1)} = 4.1$. Splitting the data along another split as for instance, $x^{(2)} = 2.9$ does not help in creating an appropiate decision boundary for the given dataset. Hence the first feature $x^{(1)}$ is more relevant for the given classsification task than the second feature $x^{(2)}$. This intuitive solution can also be found by computing information gain for both cuts. Expected result would be a high information gain for the split along the first feature $x^{(1)} = 4.1$ and a low information gain for the split along $x^{(2)} = 2.9$.

To calculate the information gain first the entropy of the entire dataset need to be computed. It is

$$H\left(X_L\right) = -\frac{16}{33} \log_2 \frac{16}{33} - \frac{17}{33} \log_2 \frac{17}{33} \approx 0.99934 \,\text{bit} \tag{7.54}$$

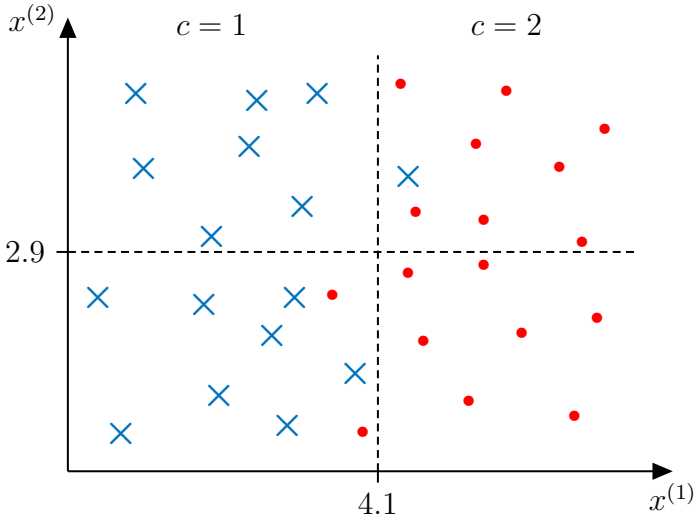


Figure 7.21: Example for feature ranking by a decision tree. The split along feature $x^{(1)} = 4.1$ separates classes much better than the split along $x^{(2)} = 2.9$ which can be found by calculating information gain for both splits. Hence the decision tree splits the data along feature $x^{(1)}$ in its first level and along $x^{(2)}$ in its second level, implicitely ranking both features according to their relevance.

which is close to the maximum entropy of 1 bit as the dataset contains nearly equal amounts of samples for both classes. If a decision tree now splits data along the first feature $x^{(1)} = 4.1$ resulting entropies can be calculated as

$$H\left(X_L|x^{(1)} > 4.1\right) = -\frac{1}{16}\log_2\frac{1}{16} - \frac{15}{16}\log_2\frac{15}{16} \approx 0.33729\,\text{bit} \tag{7.55}$$

for the subset to the right of the split and as

$$H\left(X_L|x^{(1)} < 4.1\right) = -\frac{15}{17}\log_2\frac{15}{17} - \frac{2}{17}\log_2\frac{2}{17} \approx 0.52256\,\text{bit} \tag{7.56}$$

for the subset to the left of the split. This leads to an information gain for the first split of

$$g_1 = H\left(X_L\right) - \frac{16}{33}H\left(X_L|x^{(1)} > 4.1\right) - \frac{17}{33}H\left(X_L|x^{(1)} < 4.1\right) \approx 0.56661\,\text{bit}. \tag{7.57}$$

Accordingly for the second split the information gain can be calculated from the individual entropies of the two subsets above and below the split. The entropy for the above subset is

$$H\left(X_L|x^{(2)} > 2.9\right) = -\frac{8}{16}\log_2\frac{8}{16} - \frac{8}{16}\log_2\frac{8}{16} \approx 1.00000\,\text{bit} \tag{7.58}$$

and the entropy for the subset below the split is

$$H\left(X_L|x^{(2)} < 2.9\right) = -\frac{8}{17}\log_2\frac{8}{17} - \frac{9}{17}\log_2\frac{9}{17} \approx 0.99750\,\text{bit}. \tag{7.59}$$

This gives the information gain

$$g_2 = H\left(X_L\right) - \frac{16}{33}H\left(X_L|x^{(2)} > 2.9\right) - \frac{17}{33}H\left(X_L|x^{(1)} < 2.9\right) \approx -0.99816\,\text{bit} \tag{7.60}$$

for the second split. As expected information gain of the second split is much lower than information gain of the first split. Therefore the tree has ranked feature $x^{(1)}$ as more relevant than feature $x^{(2)}$ and splits the dataset along the first feature. Of course in reality the value $\nu_1 = 4.1$ of the split position is not given, but needs to be determined simultaneously to the feature along which the split needs to be inserted. This is done by discretizing both numeric features $x^{(1)}$ and $x^{(2)}$ and calculating information gain for each possible split position along both discrete features. The resulting information gains are compared and the highest information gain gives the most relevant feature as well as the position where the split needs to be inserted. The entire process is repeated until some stop criterion like depth of the tree or number of selected features is reached. Result of this process is then a ranking of the first r most relevant features in the datatset. [40, 178, 179]

Regarding the classification task for the milling machine feature selection is implemented in the Orange3 workflow shown in fig. 7.15 in form of the *Preprocessor* widget. Here the subwidget *Select Relevant Features* which is depicted in fig. 7.22 is utilized to determine which of the features extracted in section 7.2 are most relevant. Selection is done as described above by means of maximizing information gain. The number of features to select is set to 10 as this number turns out to be a good trade-off between high classification performance on the test set and low dimensionality of the dataset. [117]

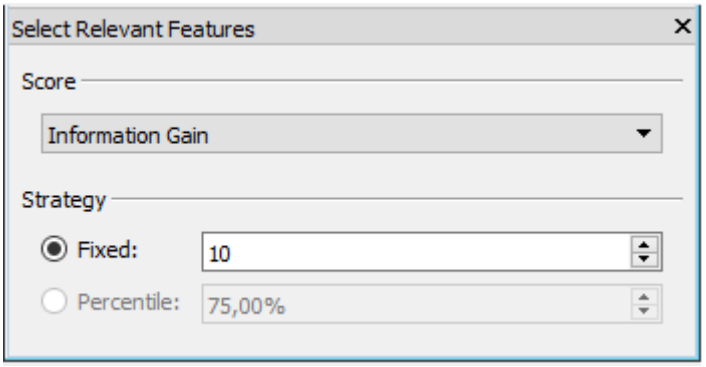


Figure 7.22: Feature Selection subwidget in the preprocessing widget in Orange3. This subwidget selects the 10 most relevant features from the input dataset, whereby features are scored according to their information gain.

However preceeding to the automatic feature selection a manual preselection is conducted as some of the 49 features extracted in section 7.3.2 turn out to be inappropiate for the given classification task. For the manual preselection a *Column Selection* widget that filters out selected columns, this means features from its input dataset and passes the modified dataset to its output. Features that are manually removed from the dataset are the number of samples q and area under curve s_{AUC} as they are highly correlated with the length of the segment which is classified. This is usually unwanted as features should be invariant to changes in the length of the signal that is classified. In this case it would be suffiecient to rely on the length as feature, however if the measurement signal is not cut into segments, but instead continuously analyized as it might be the case in the later course of this project, features should be choosen to be compatible with such an online analysis. Moreover the median normalized frequency f_{med} is removed as it is highly correlated with the mean normalized frequency f_{mean} and therefore the

feature selection algorithm always chooses both features. A strong correlation also exists between the 2nd order central moment (variance) μ_2, the standard deviation σ and the relative standard deviation s_{RSD}. Therefore only variance is kept while the other two features are removed from the dataset. Moreover the minimum value s_{min}, the maximum value s_{max} and the maximum absolute value s_{peak} are sorted out as both minimum and maximum value are contained in the peak to peak value s_{p2p} and the maximum absolute value is identical to the maximum value for the given signal which has only positive amplitudes. After removing these features the remaining dataset comprises 41 different features from which the automatic feature ranking selects the 10 most relevant features. The selected features are shown in table 7.5 together with their according rank, whereby rank 1 stand for the most relevant feature.

Table 7.5: Subset of 10 most relevant features which maximize information gain and are choosen by the Orange3 feature selection algorithm. This subset is drawn from the 41-dimensional dataset of which some features are removed by manual preselection.

Rank	Feature Name	Symbol	Rank	Feature Name	Symbol
1	9th Order Central Moment	μ_9	6	Median Normalized Frequency	f_{mean}
2	7th Order Central Moment	μ_7	7	Mean Absolute Deviation	d_{mean}
3	8th Order Central Moment	μ_8	8	Standard Deviation	σ
4	4th Order Central Moment	μ_4	9	Crest Factor	k_s
5	Peak to Peak Value	s_{p2p}	10	Shape Factor	k_f

Visualizing the 10-dimensional feature space spanned by the slected features if of course not possible. However projections onto two-dimensional planes in the 10-dimensional feature space are possible. These planes are spanned by two features each, so there are 100 possible projections. Orange3 features an automative ranking for those projections according to their information content. The most informative projection found is shown in fig. 7.23 and is spanned by the peak to peak value and the mean absolute deviation. While most of the classes are easily distinguishable as each class corresponds to one cluster of samples, some of the classes like 2, 3, 4 and 5 as well as 6, 8 and 9 lie very close to each other. Zooming closer into the first area as shown in fig. 7.24a makes clear that classes 2 and 3 as well as 4 and 5 are partly overlapping, which makes fitting of a decision boundary in between of those two pairs of classes difficult. Classes 6, 8 and 9 however are not significantly overlapping as can be seen in fig. 7.24b, wherefore classification of those classes is uncomplicated. Taking into account that the scatterplots are only a projection of the 10-dimensional feature space and that there are 8 more features which help distiguishing between the 12 classes it is very likely that classes are easily separable with a relatively simple decision boundary. Therefore a high classification performance even on unseen test data can be expected.

7.3.3 Hyperparameter Setup

Each of the seven classifiers used in the Orange3 workflow (see fig. 7.15) for solving the present classification task comprises a set of hyperparameters which need to be adjusted prior to classifier

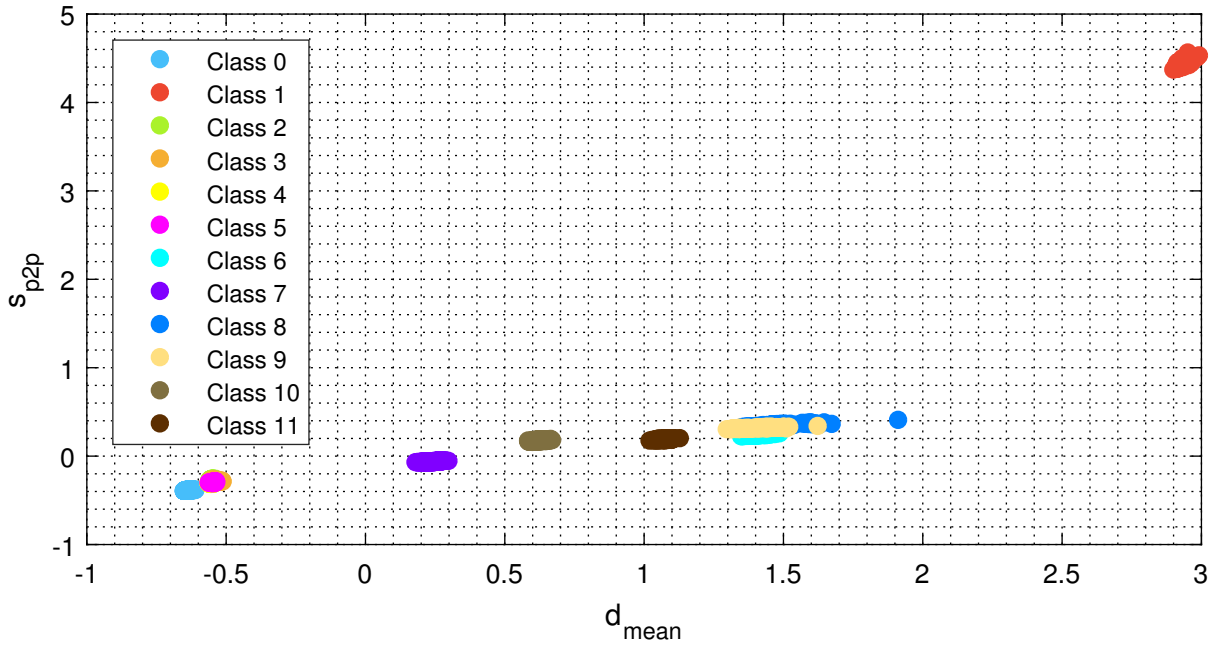
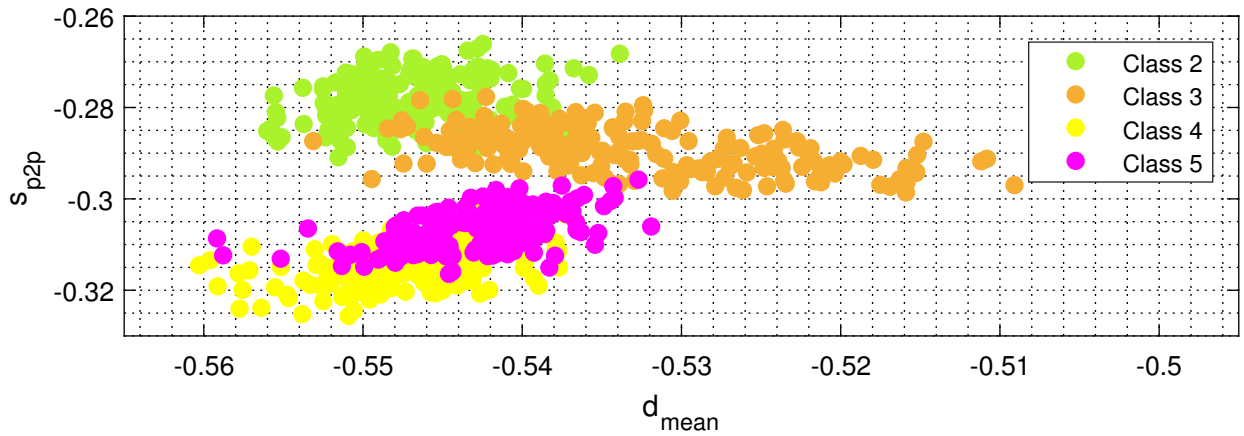


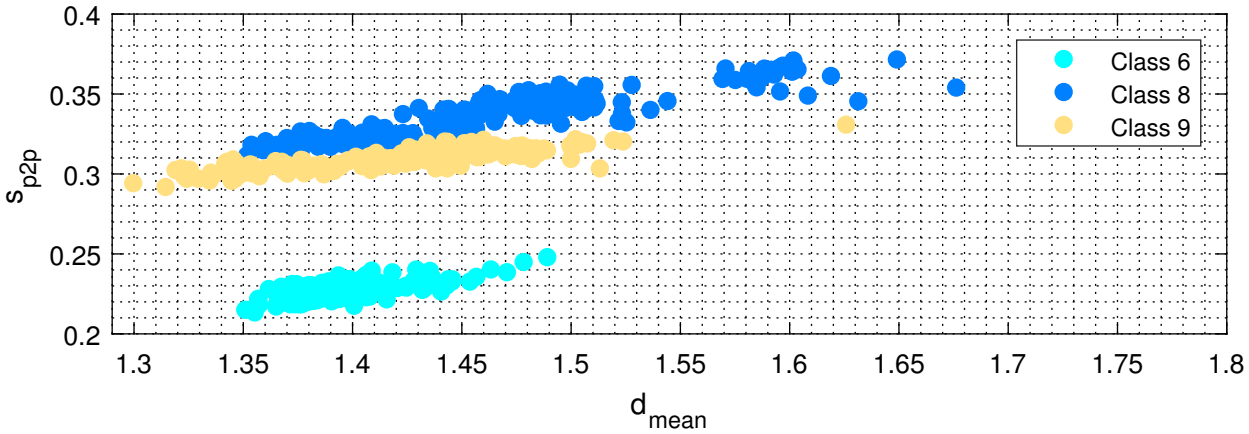
Figure 7.23: Two-dimensional projection of a scatterplot of the training dataset. This projection was ranked as most informative by the Orange3 scatterplot widget and shows all samples in the training set projected to the plane spanned by the peak to peak value and mean absolute deviation.

training. As adjustment of the hyperparameters affect classification performance this task is an optimization problem. Hyperparameters differ from normal model parameters because they need to be estimated before the classifier is trained on the training data. This estimation is then usually scored on an independent test dataset. Therefore hyperparameters play an important role in controlling the training process. During training normal model parameters are adjusted to fit the model to a given training set. As this fit is only scored on the training data itself, it is likely that the model overfits to the training data and loses its ability to generalize to unseen test data. But by introducing hyperparameters which control the training process and which are scored against an unseen test dataset overfitting can be prevented. Finding optimal hyperparameters can be done for instance by an exhaustive grid search. Forthis first different discrete values for each of the hyperparameters that shall be tuned have to be given. They span a parameter grid containing every possible combination of the hyperparameters which are applied to the classifier model. Performance of the so created model is scored against a test set giving a model test score for each combination of hyperparameters. The maximum test score finally leads to the optimal set of hyperparameters within the predefined grid. Such an exhaustive grid search has some problems though, because it requires discretization of numeric hyperparameters and is in most cases computationally expensive. Another method is randomized searching of the hyperparameter space. This method overcomes the problem of high computational expense, however does not evaluate each possible hyperparameter combination. [123, 180, 181]

For the given problem hyperparameters are not optimized automatically, but rather by manual tweaking and evaluating of the classification accuracy on the test dataset. This is reasoned by the fact that the visual interface of Orange3 does not incorporate any tools for automatic



(a) Area in the scatterplot containing classes 2, 3, 4 and 5.



(b) Area in the scatterplot containing classes 6, 8 and 9.

Figure 7.24: Close-up of the two-dimensional projection of the training dataset. The close-ups show two areas in which classes are more dense packed. The first area contains classes 2, 3, 4 and 5 which are relatively densely packed and partly overlapping which makes fitting of a decision boundary complicated. The second area contains classes 6, 8 and 9 which are close to each other however not overlapping significantly, so a decision boundary can easily be fit in between.

hyperparameter optimzation. However manual adjustment allows for determining which hyperparameters are important and which ones are not by means of their influence on the test score. This makes efficient adjustment of the hyperparameters possible and leads to sufficient scores on the testset. Table 7.6 shows the selected hyperparameters for each of the seven classifiers that lead to reasonable classification performance on the test dataset. Important to mention is that the naive bayes classifier is the only classifier not having any hyperparameters and therefore not requiring any hyperparameter tuning. Detailed explanations of the individual parameters for all other classifiers can be found in the Orange3 documentation in [117]. It is not guranteed that the choosen hyperparameters are the optimal ones, however finding globally optimal parameters by means of a grid search is not possible as well, because the search space has to be manually discretized and restricted. Therefore the hyperparameter setting can be seen as reasonable and is used in the later evaluation and application of the classifiers.

Table 7.6: Overview of manually-tuned hyperparameters of classifiers implemented in Orange3 [117]. The found parameters result in a reasonable high classification performance.

Classifier	**Setting**	**Value**	**Type**
k-Nearest-Neighbour	Number of neighbors	4	Integer
	Metric	Euclidean	Categorical
	Weight	Uniform	Categorical
Decision Tree	Induce binary tree	Checked	Checkbox
	Min. number of instances in leaves	2	Integer
	Do not split subsets smaller than	5	Integer
	Limit the maximal tree depth to	100	Integer
	Stop when majority reaches (%)	95	Integer
Random Forest	Number of trees	30	Integer
	Number of attributes considered at each split	Unchecked	Checkbox
	Fixed seed for random generator	0	Integer
	Limit depth of individual trees	Unchecked	Checkbox
	Do not split subsets smaller than	5	Integer
AdaBoost	Number of estimators	100	Integer
	Learning rate	1.00000	Float
	Fixed seed for random generator	0	Integer
	Classification algorithm	SAMME.R	Categorical
	Regression loss function	Linear	Categorical
Support Vector Machine	SVM Type	SVM	Categorical
	Cost (C)	1.00	Float
	Regression loss epsilon (ε)	0.10	Float
	Kernel	Polynomial	Categorical
	Kernel Parameter g	auto	Float
	Kernel Parameter c	0.00	Float
	Kernel Parameter d	5.0	Float
	Numerical tolerance	0.0010	Float
	Iteration limit	10	Integer

Table 7.6 (continuation) : Overview of manually tuned hyperparameters of classifiers implemented in Orange3 [117]. The found parameters result in a reasonable high classification performance.

Classifier	Setting	Value	Type
Stochastic Gradient Descent	Classification loss function	Perceptron	Categorical
	Parameter ε	0.10	Float
	Regression loss function	Squared Loss	Categorical
	Regularization method	Ridge (L2)	Categorical
	Regularization strength (α)	0.00001	Float
	Mixing Parameter	0.15	Float
	Learning rate	Constant	Categorical
	Initial learning rate (η_0)	0.0100	Float
	Inverse scaling exponent (t)	0.2500	Float
	Number of iterations	8	Integer
	Shuffle data after each iteration	Checked	Checkbox
	Fixed seed for random shuffling	0	Integer
Naive Bayes	-	-	-

7.4 Evaluation of Feature-Based Classifiers

After setting up the proper hyperparameters and training on the preprocessed training dataset the seven different classifiers presented in section 7.3 can now be evaluated on the independent and also preprocessed test dataset. For this the Orange3 *Scoring* widget is used in combination with the *Confusion Matrix* widget. Examining the confusion matrix of all classifiers gives a detailed insight in the accuracy of the classifier and shows especially which classes are misclassified most frequently. As an exmaple fig. 7.25 shows the confusion matrix for the k-nearest-neighbour classifier. Corresponding confusion matrices for all other classifiers are depicted in the appendix in appendix A.6. A confusion matrix shows the true classes of the samples within the test set in its rows and the class labels predicted by the classifier in its columns. The matrix entries (i, j) correspong to the number of samples classified as class c_j and having the true class c_i. So ideally all elements apart from the matrix diagonal from the upper left to the lower right would be zero and the diagonal would contain all samples, meaning all class labels are predicted correctly. Whenever an element not lying on the diagonal is non-zero the classifier has misclassified the according amount of samples. This behaviour can easily be expressed by means of scalar values called *precision*, *recall* and F_1*-score* which can be computed from the values in the confusion matrix. The three measures lie in a range between 0 and 1, where 1 is the best score and 0 the worst. Precision is the number of correctly classsified positive samples divided by the number of samples labeled by the classifier as positive [182]. Consider a precision of 0.6 for class 3. This means in 60 % of the cases class 3 was predicted it was predicted correctly. For the given k-nearest-neighbour classifier class 3 was predicted $92 + 15 = 107$ times. In 92 cases this

prediction was correct. This gives a precision of $92/107 \approx 0.8598$ for class 3. Recall on the other hand is the number of correctly classified positive samples divided by the number of positive samples in the entire test dataset [182]. Consider a recall of 0.3 for class 3. This means in all the cases class label 3 should have been predicted only 30 % of the labels are correctly predicted. The F_1-score is a combination of precision and recall and usually lies in between of those two values. Calculating precision, recall and F_1-score from the confusion matrix requires definition of the terms *true positive*, *false positive* and *false negative*. True positives TP_i are all samples of class i that are correctly classified as class i. False positives FP_i are all samples that are predicted as belonging to class i, but actually belong to any other class except for class i. This is the sum of all elements in the ith column except for the element (i,i) on the matrix diagonal. False negatives FN_i finally are all elements that actually belong to class i, but are predicted as any other class except for class i. This is equal to the row sum of the ith row except for the element (i,i) on the matrix diagonal. Given this precision for the ith class can be written as

$$\mathrm{Precision}_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i} \tag{7.61}$$

and accordingly recall can be defined as

$$\mathrm{Recall}_i = \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FN}_i}. \tag{7.62}$$

Based on this F_1-score is defined as

$$F_1 = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}. \tag{7.63}$$

This results in 12 different values for precision, recall and $F1$-score each. To reduce these to a single scalar value the scores are averaged. For precision and recall this can be done by summing the scores over all classes and dividing by the number of classes. For instance the average precision for all classes can be claulcated as

$$\mathrm{Precision}_M = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Precision}_i \tag{7.64}$$

where m is the number of individual classes. Average recall is calculated accordingly. With the two averaged values $F1$-score can be calculated according to the above equation by replacing precision and recall with the averaged values for precision and recall. This form of averaging is called *macro-averaging*. It is implemented in Orange3 and computes averaged values independently of class sizes, meaning it does not consider how many samples belong to each class. Another form of averaging called *micro-averaging* computes the averaged values for precision and recall differently favoring bigger classes. [117, 182, 183]

Besides precision, recall and F_1-score performance of the classifiers is compared utilizing classification accuracy which is the ratio of correctly predicted samples TP out of all samples n as defined in eq. (6.1). The last measure considered is the *area und curve* (AUC) value which is the area under the receiver-operating-curve. This curve plots the recall on the vertical axis over the *false positive rate* (FPR) on the horizontal axis for a specific class i. To generate multiple

different pairs of values for recall and false positive rate any hyperparameter of the classifier is varied. False positive rate for the ith class is defined as

$$\mathrm{FPR}_i = \frac{\mathrm{FP}_i}{\mathrm{FP}_i + \mathrm{TN}_i} \tag{7.65}$$

with *true negatives* (TN) which comprises all samples in the confusion matrix except for the ones in the ith row and column. An optimal classifier has a recall of 1 and a false positive rate of 0 giving an area of 1 under the receiver operating curve. [184, 185]

Table 7.7 shows the AUC, classification accuracy, F_1-score, precision and recall for all seven classifiers. F_1-score, precision and recall are macro-averaged over all 12 classes. As can be seen naive bayes and stochasstic gradient descent perform worst with a total classification accuracy of only 80.1 % respectively 88.1 %. Much better perform AdaBoost and the support vector machine with 94.5 % or rather 95.4 %. Likewise well perform k-nearest neighbhour with an accuracy of 96.2 % and the decision tree with 96.7 %. Highest performance is achieved by the random forest classsifier which reaches an accuracy score of 97.7 %. Despite these reasonable high accuracy scores a closer look on the confusion matrices in appendix A.6 shows that especially the naive bayes and the stochastic gradient descent classifiers have serious problems with distiguishing between classes 2 and 3 as well as 4 and 5 which correspond to the forward and backward movements of x- and y-axis. As the underlying signals look very similar it is clear why both classifiers have problems to differentiate between these two pairs of classes. Additionally the naive bayes classifier misclassifies classes 8 and 9 in most cases as class 1. Reason for this is not obvious as the signal corresponding to class 1, the spindle movement, does not look not very similar to the signals corresponding to classes 8 and 9, the forward and backward movement of the c-axis. However it is possible that the extracted and selected features are similar for those three classes which is why the naive bayes classifier mixem them up. The other five classifiers classify the majority of samples correctly and only show some misclassifications of classes 2 and 3 as well as classes 4 and 5, however only a fraction of all samples is misclasssified and not all samples like with the naive bayes and stochastic gradient descent classifier. In general classes 4 and 5 are mixed up more frequently than classes 2 and 3. Classes 0, 1, 6, 7, 10 and 11 are correctly classified in nearly 100 % of all cases by the other five classifiers. This is reasoned by the highly different characterstics of the underlying signals in the time domain which lead to well separated clusters in feature space (see fig. 7.24). Moreover all classifiers excpect for naive bayes and stochastic gradient descent achieve a very low amount of misclassifications on classes 8 and 9 which correspond to the forward and backward movement of the c-axis. Only the support vector machine incrorrectly classifies class 9 as class 8 in 43 % of the cases.

Based on the evaluation results the conclusion can be draw that at least the five better performing classifiers, k-nearest-neighbours, decision tree, random forest, AdaBoost and support vector machine achieved sufficient classification performance and are therefore usable for predicting class labels on unknown test sets in future experiments.

Table 7.7: Comparison of typical performance measures for the seven different classifiers. All scores are computed on the test dataset which is independent from the training set.

Classifier	**Performance Scores / %**				
	AUC	**Accuracy**	**F_1-Score**	**Precision**	**Recall**
k-Nearest-Neighbour	99.4	96.2	96.2	96.3	96.2
Decision Tree	98.3	96.7	96.7	96.8	96.7
Random Forest	**99.9**	**97.7**	**97.7**	**97.8**	**97.7**
AdaBoost	96.2	94.5	94.5	94.8	94.5
Support Vector Machine	96.8	95.4	95.3	96.0	95.4
Stochastic Gradient Descent	91.8	88.1	85.4	84.8	88.1
Naive Bayes	97.9	80.1	76.4	78.9	80.1

7.5 Conclusion of Feature-Based Classification Approach

The preceeding chapter first introduced foundations of feature based classification. It explained common terminology of feature based datasets and described in general the basic process steps of training, evaluation and testing independent of a specific classification algorithm. Afterwards seven important classification models, k-nearest-neighbour, decision tree, random forest, AdaBoost, support vector machine, stochastic gradient descent and naive bayes were described in detail and pros and cons of those algorithms were analyzed. In the subsequent section 49 different statistical features were extracted from the measurement data, both from time and frequency domain. Besides a mathematical description of all features the feature extraction algorithm which was implemented in Matlab was explained. Hereafter the extracted features were standardized by removal of the mean value and scaling to unit variance. Moreover the curse of dimensionality was introduced and the 10 most relevant features by means of maximum information gain were selected. Prior to that some inappropiate features were manually sorted out. After this hyperparameters of the classifiers were tweaked to achieve maximum performance of the test set. In the Orange3 implementation this step was conducted manually and in the scikit-learn script an exhaustive grid search over a given parameter space was carried out for each classifier. Finally classification performance of the classifiers implemented in Orange3 were evaluated and compared. It turned out that naive bayes and stochastic gradient descent performed unsatisfactory while the other five classsifiers achieved good classification performance, above all the random forest classifier with a classification accuracy of 97.7 %.

Despite the satisfying evaluation results for most of the classifiers there is potential for further optimization. For instance tuning of the hyperparameters was only conducted manually in Orange3, so it is likely that the hyperparameter setup for most of the algorithms is not optimal. However as another implementation of all algorithms in a Python script utilizing scikit-learn with an exhaustive grid search over a given hyperparameter space achieves no better results, it is possible that the best hyperparameter settings are already found. Nevertheless it must be kept in mind that even an automatic grid search is not capable of finding the global optimum

of hyperparaemters as long as the search grid is not set up properly. Because of limited computational ressources not all parameters are examined in the given case. Moreover parameter values for numeric parameters are discretized in a coarse grid. All this indicates there are better hyperparameter combinations leading to higher performance of the classifiers. Apart from this feature selection could be done differently. Instead of utilizing a decision tree which ranks features based on information gain a wrapper method comprising the actual classfier could perform better. To investigate this the classifier should be trained and evaluated using all possible combinations of features and then the set leading to highest classification performance on the test set should be selected. However this approach is computationally much more expensive than the conducted selection by means of a decision tree, but could lead to higher performance of the classifiers. Besides that different kinds of scaling and centering could be evaluated to possibly achieve higher classsification performance. And finally more classification algorithms could be fitted to the dataset. For instance neural networks and rule induction algorithms which are not covered in this chapter.

Referring back to chapter 6 it can be said that the approach considered in this chapter lead to development of classifiers which performed similar well as the DTW classifier which achieved a classsification accuracy of 97 %. This accuracy is exceeded by 0.7 % by the random forest classifier, however all other feature based classifiers perform slightly worse than the DTW classifier. This said both approaches, the shape based and the feature based, seem to be equally useful for the given task of classifying the 12 nominal machine states based on the acquired measurement data. However the approach conducted in this chapter is easier to standardize, as it complies with the common feature based pattern recognition approach. This also allows for easy application of new machine learning algorithms as the data is presented in a very universal form as extracted and normalized feature set. To further improve performance of the feature based classifiers suggestions already made in chapter 6 can be applied as well. For instance merging classes 2 and 3 respectively 4 and 5 in one virtual class each would drastically increase classification performance as most misclassification are made on thos two pairs of classes.

Finally it can be concluded that the classifiers developed both in this chapter as well as in chapter 6 are sufficient for classyfing machine states of the miling machine and can be used to predict segment classes on test datasets of future measurements.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	84	15	0	1	0	0	0	0	0	0
3	0	0	4	92	0	4	0	0	0	0	0	0
4	0	0	0	0	85	15	0	0	0	0	0	0
5	0	0	0	0	35	65	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	99	1	0	0
9	0	0	0	0	0	0	0	0	9	91	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure 7.25: Confusion matrix for the k-nearest-neighbours classifier. Rows show true classes and columns predicted classes of each sample within the test data set. Ideally all elements except for the diagonal from the upper left to the lower right would be zero, meaning class labels are predicted correctly for all samples in the test set.

8 Machine Fault Detection

Aim of this chapter is development of easy to understand and robust algorithms for detecting faults of the milling machine which is subject to this work. A machine fault is hereby any operational state which significantly deviates from the machine state taken on during acquisition of the training data in chapter 4. Here the algorithms are required to not only detect whether the machine is operational or not, but furthermore resolve also in which operational mode a fault occurs. To be able to resolve machine faults to the individual components of the machine the machine is operated in the previously developed test cycle to create a test dataset which contains segments representing movements of individual machine axes. By finding faulty segments in the entire measurement data faulty components can be detected. In general this task can be seen as classsification, where the classifier has to detect for each of the 12 machine classes whether the machine is healthy or not. Such kind of algorithm is developed in this chapter. However rather than utilizing complex machine learning algorithms to classify between faults first a simple similarity measurement approach is conducted to find systematic faults which apply to all segments within the measurement data of the test cycle run and second an outlier detection is performed to obtain faults which affect only a few segments within the entire measurement data. To develop those two kinds of algorithms first the local outlier factor as method for finding outliers based on estimation of the local densitiy wihtin feature space is introduced. Afterwards the experiment layout is described in detail. Four different sets of modified test data are created each simulating a different machine fault. Here the first three are synthetically generated by applying mathematical functiosn to the already present test measurement data and the last one is acquired by measurement on a the slightly modified machine. After generation of modified test datasets two algorithms are developed and implemented in Python to find both systematic and random machine faults. Results of the application of these two algorithms on the previously modified test datasets are then evaluated. It turns out that the proposed algorithms are capable of correctly detecting all of the synthetically introduced machine faults and find all significant changes in the real test data acquired on the modified machine. Finally conclusions for the machien fault detection are drawn and suggestions for improvement and further research are made.

8.1 Local Outlier Factor

To detect outliers in a set of samples within feature space a large variety of different methods has been developed. An overview of these methods can be found in [186] and [187]. In the following experiments a densitiy based approach utilizing the *local outlier factor* (LOF) is used to find outliers in feature space. The local outlier factor is a method originally proposed in [53] which allows for detection of outliers in feature space based on estimation of the local sample density around each sample. Advantage of this method compared to other methods which only

calculate the distance of a point to its nearest neighbours is the fact that samples lying within a cluster of low density corresponding to large distances between samples are not considered as outliers while samples which lie close but outside of a dense cluster are considered as outliers. This is achieved by comparing the sample density of a sample with the sample density of its k nearest neighbours. When the density around a sample is significantly lower than the density around its neighbours the sample is an outlier, otherwise an inlier. This situtation is illustrated in fig. 8.1. Here a dataset containing two clusters Z_1 and Z_2 and five outliers $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_5$ is shown. If outliers were searched based on distance to the surrounding neighbours of a sample only the global outliers $\boldsymbol{o}_3$, $\boldsymbol{o}_4$ and $\boldsymbol{o}_5$ would be found as they have a large distance to their next neighbour. However the local outliers $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$ would not be found as their distance to the nearest neighbour is smaller than the distance between the samples in cluster Z_2. The density based approach however considers the local density in the surrounding region of a sample. Samples $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$ for instance lie in a region of high density and have a distance to their nereast neighbour which is significantly larger than the average distance between samples in that area. Therefore they are considered to be outliers. If the same samples would lie with thesame distance to cluster Z_2 they would not be labeled as outliers as the density in cluster Z_2 is much lower than in Z_1. [53, 188, 189]

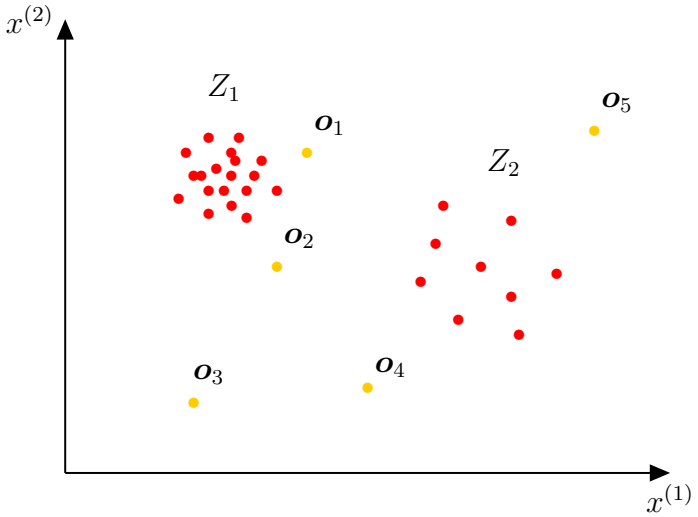


Figure 8.1: Scatterplot of data samples illustrating outlier detection based on local density estimation. The scatterplot contains two clusters Z_1 and Z_2, one of high densitiy and one of lower density. Moreover the set contains five outliers. In areas of high density as around cluster Z_1 samples which lie relatively close to the cluster, however outside of it are already considered to be outliers, such as $\boldsymbol{o}_1$ and $\boldsymbol{o}_2$. In areas of low density outliers have a large distance to neighbouring samples as it is the case for outliers $\boldsymbol{o}_3$, $\boldsymbol{o}_4$ and $\boldsymbol{o}_5$.

Mathematically the local outlier factor of a sample $\boldsymbol{o} \in \mathbb{R}^p$ is computed the following way. First the *k-distance* $\text{dist}_k(\boldsymbol{o})$ is defined as distance between the sample $\boldsymbol{o}$ and its kth nearest neighbour $\tilde{\boldsymbol{o}}$. If for instance the 2-distance $\text{dist}_2(\boldsymbol{o})$ of sample o is demanded the distance can be found by finding the sample $\tilde{\boldsymbol{o}}$ which is not closest, but second closest to sample $\boldsymbol{o}$ and computing an

appropiate distance measure like the manhattan distance

$$\|\boldsymbol{o} - \tilde{\boldsymbol{o}}\| = \sum_{i=1}^{p} \left| o^{(i)} - \tilde{o}^{(i)} \right| \tag{8.1}$$

between those two samples. Furthermore the *k-distance-neighbourhood* $N_k(\boldsymbol{o})$ of sample $\boldsymbol{o}$ has to be defined as

$$N_k(\boldsymbol{o}) = \{\boldsymbol{o}' \,|\, \boldsymbol{o}' \in D, \mathrm{dist}(\boldsymbol{o},\boldsymbol{o}') \leq \mathrm{dist}_k(\boldsymbol{o})\}\,. \tag{8.2}$$

So the k-distance-neighbourhood contains all samples around the sample $\boldsymbol{o}$ which lie within its k-distance. With these two defintions the *reachability distance* from a member $\boldsymbol{o}'$ of the k-distance-neighbourhood to sample $\boldsymbol{o}$ can be computed as

$$\mathrm{reachdist}_k(\boldsymbol{o},\boldsymbol{o}') = \max\{\mathrm{dist}_k(\boldsymbol{o}), \mathrm{dist}(\boldsymbol{o},\boldsymbol{o}')\}. \tag{8.3}$$

So the reachability density is either the true distance between the two samples, but at least the k-distance of $\boldsymbol{o}$. This definition which is not a distance in mathematical sense as it is not symmetric gives more stable results. By caclulating reachability distances from all samples in the k-distance-neighbourhood of sample $\boldsymbol{o}$ to this sample enables calculation of the *local reachability density* of $\boldsymbol{o}$ according to

$$\mathrm{lrd}_k(\boldsymbol{o}) = \frac{\|N_k(\boldsymbol{o})\|}{\sum_{\boldsymbol{o}' \in N_k(\boldsymbol{o})} \mathrm{reachdist}_k(\boldsymbol{o}',\boldsymbol{o})}. \tag{8.4}$$

This is the inverse of the average reachability distance of sample $\boldsymbol{o}$ from its neighbours $\boldsymbol{o}' \in N_k(\boldsymbol{o})$. The local outlier factor of sample $\boldsymbol{o}$ can now be computed by comparing its local reachability density with the local reachability densities of its neighbours according to

$$\mathrm{LOF}_k(\boldsymbol{o}) = \frac{\sum_{\boldsymbol{o}' \in N_k(\boldsymbol{o})} \frac{\mathrm{lrd}_k(\boldsymbol{o}')}{\mathrm{lrd}_k(\boldsymbol{o})}}{\|N_k(\boldsymbol{o})\|} = \sum_{\boldsymbol{o}' \in N_k(\boldsymbol{o})} \mathrm{lrd}_k(\boldsymbol{o}') \cdot \mathrm{reachdist}_k(\boldsymbol{o}',\boldsymbol{o}). \tag{8.5}$$

Whenever the local reachability density of sample $\boldsymbol{o}$ is low and the local reachability densities of its neighbours is high the LOF is high and $\boldsymbol{o}$ considered to be an outlier. On the opposite if local reachability density of the sample is similar to the one of its neighbours the LOF is approximately 1, which means sample $\boldsymbol{o}$ lies within a cluster and is no outlier. So finding outliers in a dataset is as easy and computing the LOF for all samples in the dataset and searching for samples with high LOF values. For best performance of the outlier detection appropiate values for the number k of considered neighbours as well as the threshold value for the LOF above which to take a sample as outlier have to be found. In [53] a method for finding the best value for k is proposed, however experimental tweaking of the parameter on the given dataset is also possible. [53, 188–190]

8.2 Test Dataset Generation

As the aim of this chapter is unsupervised detection of machine faults, it is necceassary to create test data containing faulty machine behaviour. This can be done by either replacing a healthy

component with a defect one or adding of an additional faulty component. Both solutions require substantial modification of the milling machine as well as the entire segmentation and labelling process and are therefore not easily feasible. Instead two different approaches for creation of modified test data are conducted. First the already available test data is modified by mathematical functions to synthetically model faulty machine behaviour and second an additional weight is placed on the machine table which affects power consumption during movement of the z-axis. Moreover the tool clamped into the spindle is removed which also affects current consumption. Adding a weight on the machine table and removing the tool are of course no machine faults, however affect power consumption of the machine similar to possible machine failures like sluggish movement or partial deadlock of the axes and therefore can be seen as method to simulate faulty machine behaviour. The first of the following two sections describes the synthetic generation of modified test data in detail while the second one examines the issue of generating a modified test dataset by adding an additional weight on the machine table.

8.2.1 Synthetic Generation of Modified Test Datasets

The first method of generating a test dataset of a faulty machine is synthetic modification of the already available measurement data. For this purpose the measurement data from the first test cycle run (see fig. 4.7 and fig. 4.8), which is acquired in chapter 4 is copied and modified partially to simulate different kinds of faults. In total three different synthetic modifications are conducted corresponding to three different experiments.

In the first experiment all 100 segments belonging to class 6 are modified. As known from section 4.3 class 6 corresponds to the upward movement of the z-axis. To simulate a partial deadlock of this axis during the upward movement current consumption is increased by factor 1.5 compared to the original current signal. Figure 8.2a shows a single segment of class 6 before the modification (brown line) and after the modification (blue line). Modifications are conducted on the already segmented measurement data, so it is possible to apply modifications only to specific segments rather than the entire measurement signal. The modification process comprises first finding of all segments belonging to class 6 which can easily be done by looking at the class label of each segment. Afterwards the algorithm determines for each segment the signal offsets at both endpoints of the segment and creates a linear function connecting both endpoints. This offset function is then subtracted from the segment to retrieve a transformed segment. Both endpoints of this transformed segment lie on the abscissa corresponding to a current value of 0 A. After this the transformed segment is multiplied by factor 1.5 and the linear offset function is added again giving the signal shown in fig. 8.2a. Prior subtraction of the linear offset function is neccessary to ensure that original and modified segment have the same ampltiude values at the endpoints so that the modified measurement signal does not contain any discontinuities.

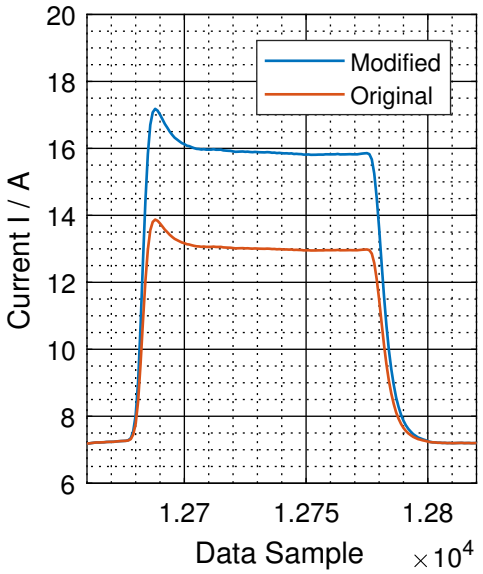
In the second example all 100 segments of class 6 and all 100 segments of class 7 are modified simultaneously. This simulates a fault affecting both upward and downward movement of the z-axis. For instance the additional weight on the machine table has such an effect on the current signal. Computation of the modified segments is conducted according to the process described above. Figure 8.2b compares two modified segments of classes 6 and 7 with the corresponding segments in the original test measurement data.

While the first and second experiment aim to simulate a systematic fault which persists over the entire measurement time of 7200 s the third experiment tries to emulate random machine faults that occur only sporadically and last a short time rather than occuring constantly. This kind of machine fault is simulated by modification of randomly sampled segments. Moreover not all segments are modified the same way, but one individual segment of class 6 is modified differently to emulate noisy measurements. This modified segment is depicted in fig. 8.2d. Such noise should not be detected as machine fault by the detection algorithm, but instead be neglected as noise. To create suitable test data for this experiment 10 segments of class 6, 5 segments of class 7 and 2 segments of class 2 (x-axis forward movement) are randomly selected. One of the modified segments of class 2 is shown in fig. 8.2c. As only two samples of class 2 are modified, the algorithm should not assume any machine fault in this class as all other 98 segments of class 2 do not contain any faulty behaviour. So similar to the noisy segment of class 6 the two faulty segments of class 2 should be taken as noise which can be neglected. Only when the number of segments containing the same kind of modification increases a specific threshold a systematic machine fault shall be assumed. How this detection of machine faults is achieved will be explained in the further course of this chapter.

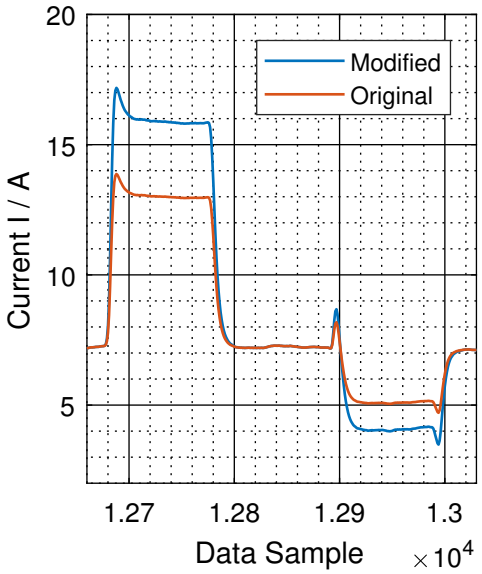
After modification of the test measurement data by means of the Matlab script shown in listing A.13 the three synthetically generated test measurement datasets are fed into the feature extraction algorithm developed in section 7.2 which gives feature based datasets that can be further processed by the machine fault detection algorithm.

8.2.2 Experimental Generation of a Modified Test Dataset

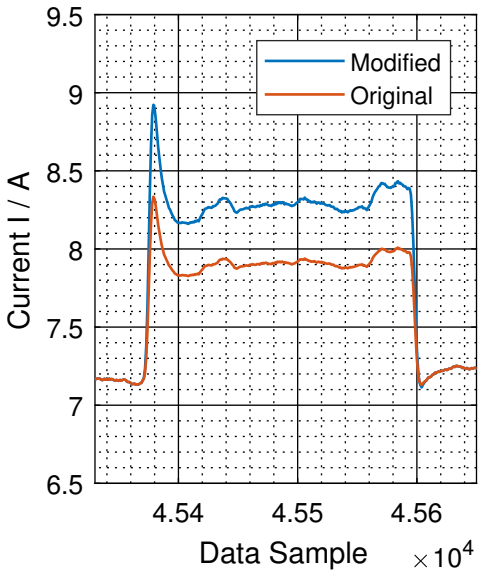
In addition to the synthetic modification of the measurement data a fourth experiment is conducted which aims to experimentally produce a test dataset by partly modifying the milling machine and acquiring a new test dataset. Looking at real measurement data helps to validate if the synthatically modified datasets simulate machine faults appropiately. However as already said it is an elaborate process to replace a healthy component of the milling machine by a defect one as after replacement modifications of the measurement hardware have to be conducted and additionally the machine has to be recalibrated. Therefore no component is exchanged, but instead an additional weight is added to the machine table increasing static and dynamic load of the z-axis which leads to higher current consumptions when moving this axis. This increased current consumption simulates a machine fault. In the experiment the additional weight is realized by bolting a heavy steel clamping unit onto the machine table. Afterwards the test cycle program as defined in section 4.4 is executed to create the specific measurement dataset which can then be further processed by the already developed algorithms. To process this newly created dataset it is first extracted from the database. Subsequently autmatic segmentation and labelling by means of the algorithm developed in section 5.3 is conducted to create specific segments for each individual axis movement within the measurement data of the test cycle. After segementation features can be extracted according to section 7.2 giving a feature space representation of the modified test dataset. This dataset is the basis of the machine fault detection algorithm. For processing the measurement data of the modified machine slightly



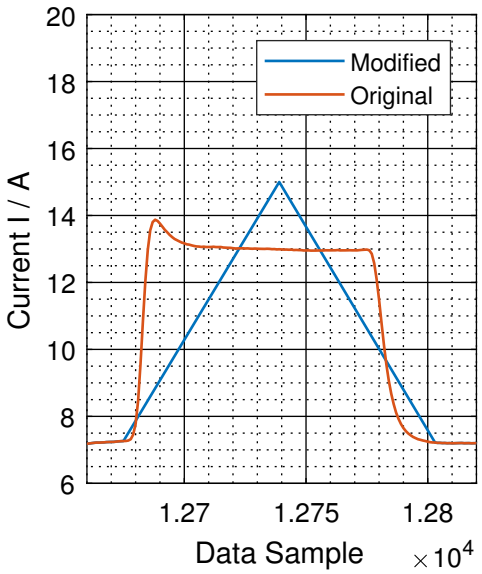
(a) Modified segment of class 6 as occuring in the first experiment.



(b) Simultaneously modified segments of class 6 and 7 as occuring in the second experiment.



(c) Modified segment of class 2 as occuring in the third experiment.



(d) Differently modified segment of class 6 occuring in the third experiment to simulate a random fault (outlier).

Figure 8.2: Different kind of synthetically modified segments in the test cycle measurement data simulating machine faults. In the first experiment all instances of class 6 are modified according to the upper left plot. In the second experiment all instances of classes 6 and 7 are modified as shown in the upper right plot. Finally, in the third experiment 10 randomly choosen segments of class 6, 5 segments of class 7 and 2 segments of class 2 are modified. Additionally the first segment of class 6 is modified differently as shown in the lower right plot.

changed versions of the scripts developed for segmentation and feature extractions are used. These can be found in appendix A.7.9.

Figure 8.3 shows an extract of the main line current consumption on phase L2 of the modified machine (blue line) in comparison to the according current consumption of the non-modified machine (brown line) which is acquired in chapter 4. Important to mention is that in the diagram mean values of both signals are removed to better align the sequences and emphasize the differences between them. As can be seen downward movement of the z-axis (class 7) is significantly influenced by the additional weight on the machine table while the upward movement (class 6) is rarely affected. Interestingly current consumption is decreased during upward movement and increased during downward movement which seems counter intuitive as load on the table is increased by the additional weight. Expected behaviour would be an increase in current consumption during the upward movement and a decrease during downward movement. Why the machine behaves differently is not clear, however important for the experiment is only the fact that the measured current consumption of the modified machine deviates from those of the non-modified machine. Aim of the machine fault detection is to detect this fault. Interpreting the fault and possible reasons for it is still task of the machine operator. Apart from this modification another modification is introduced by using an empty tool clamp instead of the tool used during acquisition of the training set. As the empty tool clamp has a lower moment of inertia compared to the previously used tool the measurement signal of the spindle movement within the test data differs slightly from the training data as shown in fig. 8.4. This is another modification which should be detected by the machine fault detection algorithm. Aside from the modification on segments of class type 1, 6 and 7 the measured current signal of the modified machine is very similar to the non-modified machine. Therefore the algorithm should not predict any machine faults in other classes.

8.3 Fault Detection Algorithm

After successful generation of synthetic test data and acqusition of test data of the modified milling machine, different algorithms for detecting faults introduced in the test datasets are developed. This section is divided into two subsections. The first deals with systematic faults and the second one with random faults. Systematic faults one the one hand refer to the kind of machine fault simulated by experiments 1, 2 and 4. Here each occurance of a specific class is modified in the same way. So the machine fault is static and affects every movement of the same type as for example the additional weight on the machine table influences each of the 100 upward and 100 downward movements in the test cycle. Random faults one the other hand are the kind of fault emulated by experiment 3. Here only a few segments of a specific class are modified while the majoritiy is equivalent to the non-modified testdata. This situtation represents a machine fault which is not static, but occurs randomly and only for a short period of time. Both kind of faults require different approaches for detections.

The source code of the entire detection algorithm covered in the following two sections can be found in listing A.20. It contains a subroutine for calculating the local outlier factor which is shown in listing A.21. A more abstract representaation of the algorithm is depticed in form of a

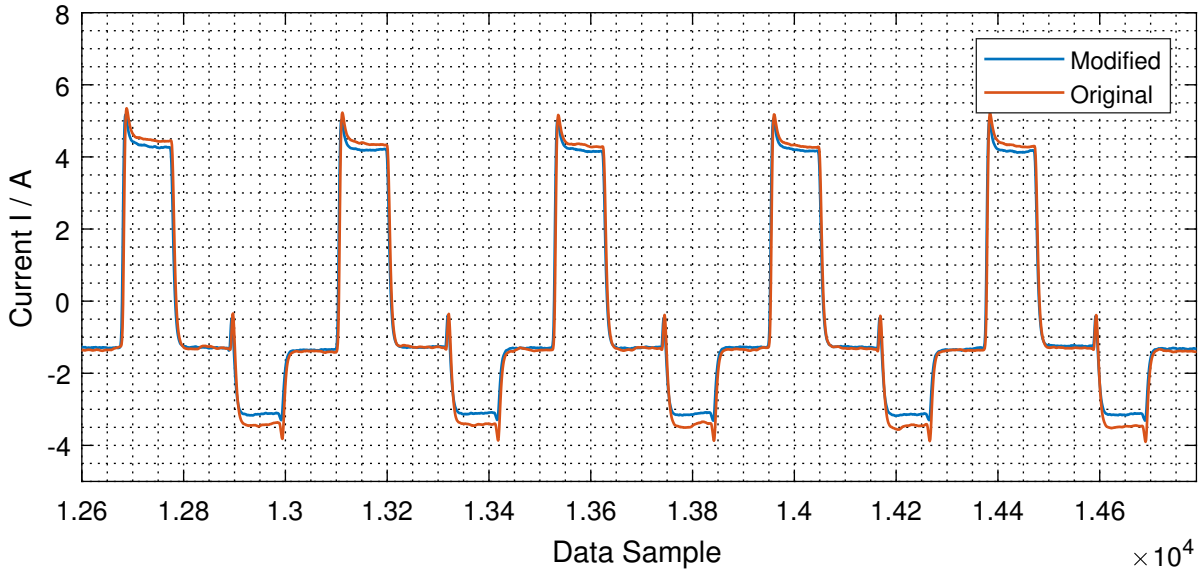


Figure 8.3: Excerpt of the current consumption on phase L2 of the modified machine in comparison to the non-modified machine. Shown are segments of the first subcycle of the test cycle run which belong to classes 6 and 7 respectively upward and downward movement of the z-axis. For better alignment of both the original and the modified sequence the signal mean is removed. Downward movement of the z-axis (class 7) is significantly affected by the additional weight on the machine table while its influence on the upward movement (class 6) is hardly apparent.

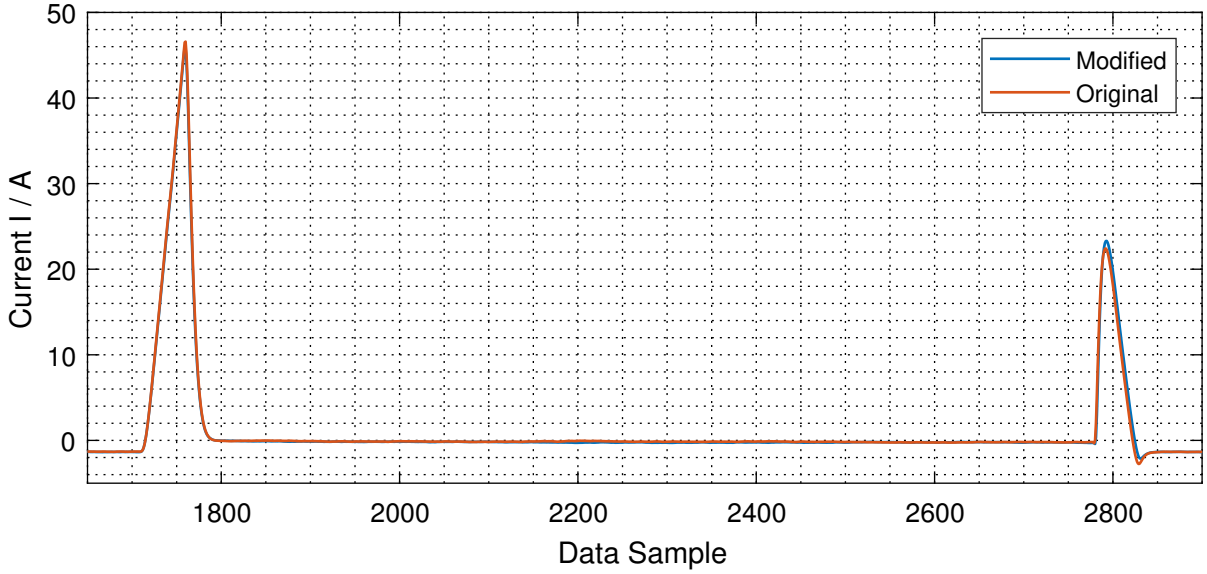


Figure 8.4: Excerpt of the current consumption on phase L2 during spindle movement of the modified machine in comparison to the non-modified machine. During training a drill is clamped into the spindle. This drill is removed during acquisition of test data for the fourth experiment leading to a slight change in the measurement data. The blue line corresponds to the measurement data of the modified machine while the brown line represents data of the non-modified machine. Again signal mean values are removed for better alignment of the sequences.

flowchart in fig. 8.5 and fig. 8.6. Figure 8.5 contains the preprocessing part of the machien fault detection algorithm. For preprocessing first the feature based training dataset as well as the four test datasets belonging to the four different experiments are loaded from their CSV files. As these datasets comprise both the 49 extracted features from section 7.2 as well as the desired class labels of every sample, they are split into the feature matrix and the desired class labels vector. Afterwards features are standardized by removal of the mean value and scaling to unit variance (see section 7.3.1). Subsequently relevant features are selected to reduce the dimensionality of the feature space representation of the measurement data. To simplify the problem of feature selection at this point the 10 features ranked as most important in section 7.3.2 are selected. Afterwards cluster centroids are calculated for each class within the training set as well as the four test datasets. The cluster centroid of a class is equivalent to the mean value of all samples belonging to that class. So given the labeled dataset

$$X_L = (X, \boldsymbol{c}) = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_k, \ldots, \boldsymbol{x}_n\} \in \mathbb{R}^p \times \{c_1, c_2, \ldots, c_k, \ldots, c_n\} \tag{8.6}$$

with $c_k \in C$ and $C = \{1, 2, \ldots, l, \ldots, m\}$ the cluster centroid for class c_l can be calculated as

$$\boldsymbol{z}_l = \frac{1}{n_l} \sum_{k=1}^{n_l} \boldsymbol{x}_k \in c_l \tag{8.7}$$

where n_l is the number of samples with class label c_l. So for each of the experiments 12 different cluster centroids can be calculated for the training dataset and 12 centroids for the according test dataset. After calculation of the cluster centroids the actual fault detection takes place. The according subroutines of the fault detection algorithm shown in fig. 8.6 are explained in detail in the following two subsections.

8.3.1 Detection of Systematic Faults

As already mentioned the first subsections deals with detection of systematic faults. Development goal is an algorithm that is able to predict whether the milling machine operates as desired or whether a machine fault is present. Moreover the algorithm should be able to determine in which class the fault occurs. So rather than only determining if the machine is fully functional or not the algorithm should also be able to determine which component is faulty, whenever a fault occurs.

The operational principle of the algorithm is shown in fig. 8.6a. First it takes the positions of the previously calculated cluster centroids $\boldsymbol{z}_l$ with $l = \{1,2,\ldots,12\}$ for both training and modified test datasets and then calculates the distance between clusters of the training and test datasets for each of the 12 classes. In this case the euclidean distance is used as distance measure, however other distance measures can be used as well. Whenever the cluster distance between training and test cluster exceeds the threshold value $d_{\text{thrs}} = 0.2$ a systematic machine fault is predicted for the specific class. If the clusters lie closer than the threshold value, the machine is assumed to be fully operational in the corresponding class.

To clarify this fig. 8.7 illustrates the process of detecting a systematic machine fault for a single class. Here the red dots represent samples of the training datasets and the blue crosses samples

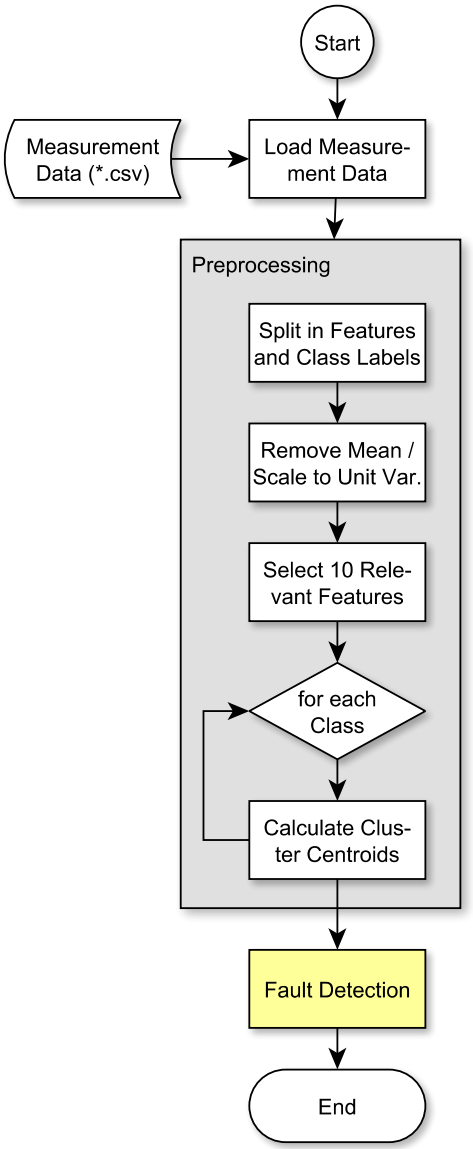
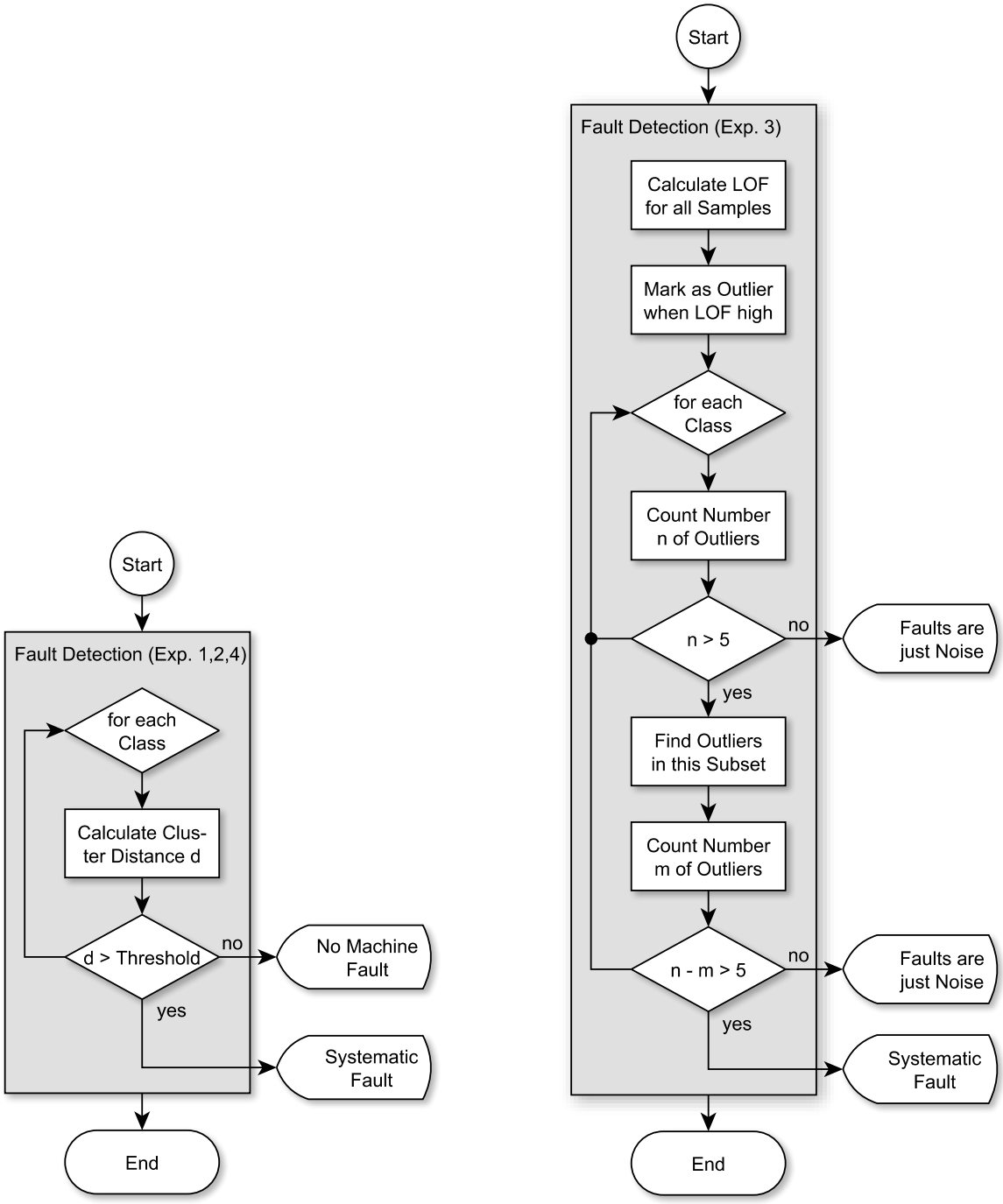


Figure 8.5: Flowchart of the preprocessing part of the machine fault detection algorithm. Training dataset as well as all four modified test datasets are loaded and preprocessed for later fault analysis. The fault detection subroutines can be found in fig. 8.6.

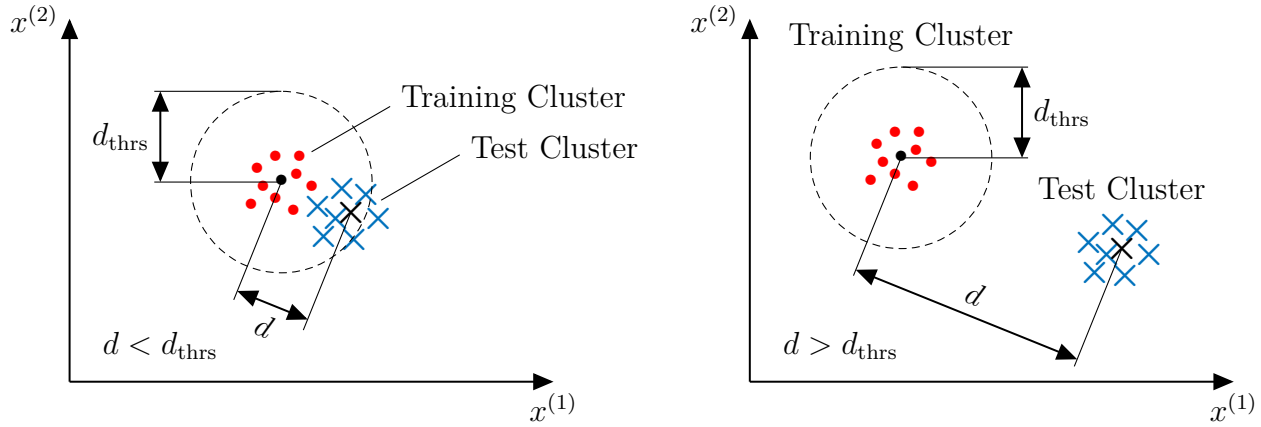


(a) Subroutine for machine fault detection in experiments 1, 2 and 4.

(b) Subroutine for machine fault detection in experiment 3.

Figure 8.6: Flowcharts of the machine fault detection subroutines. The left subroutine on the one hand applies for experiments 1, 2 and 4 and detects machine faults based on evaluation of cluster distances between corresponding classes in training and test datasets. The right subroutine on the other hand counts for experiment 3 and tries to distiguish between random noise and systematic machine faults when only a few individual segments in the measurement data are modified.

of the test dataset. The black dot and cross are the cluster centroids. Whenever the sequences in training and test dataset are very similar both the training and the test cluster lie close to each other as the features are very similar. This situtation is shown in fig. 8.7a, where the test cluster centroids lies close to the training cluster centroid ($d < d_{\text{thrs}}$). So corresponding sequences in the measurement data look similar and no machine fault is present. A different situtation is shown in fig. 8.7b. Here all test samples lie far away from the training samples which means that each segment in the measurement data is different from the corresponding segments in the training data. The distance d between both cluster centroids exceeds the threshold value d_{thrs} which is why the algorithm rightly assumes a machine fault.



(a) Cluster distance d is small. There is no fault in the according class.

(b) Cluster distance d is large. A systematic machine fault can be assumed in this class.

Figure 8.7: Principle of systematic machine fault detection by thresholding cluster shift. Distances d between the cluster centroids of training and testing clusters are calculated for each class separately and compared with a threshold value d_{thrs}. When the test cluster centroid of a specific class deviates from the training cluster centroid by a large distance a systematic fault can be assumed in this class. If the distance is small, the machine operates as desired in the according class.

The only difference between the above illustrative example for fault detection and the real implementation of the fault detection algorithm is the fact, that the real problem comprises not only one set of clusters, but 12 training and 12 test clusters representing one class each. However as class labels of all samples are known it is easy to loop through each of the 12 classes and calculate the distance between the corresponding test and training clusters. This distance is then compared to the threshold and a report for the current class is printed. So output of the entire algorithm is a list stating for each class whether a machine fault is present or not.

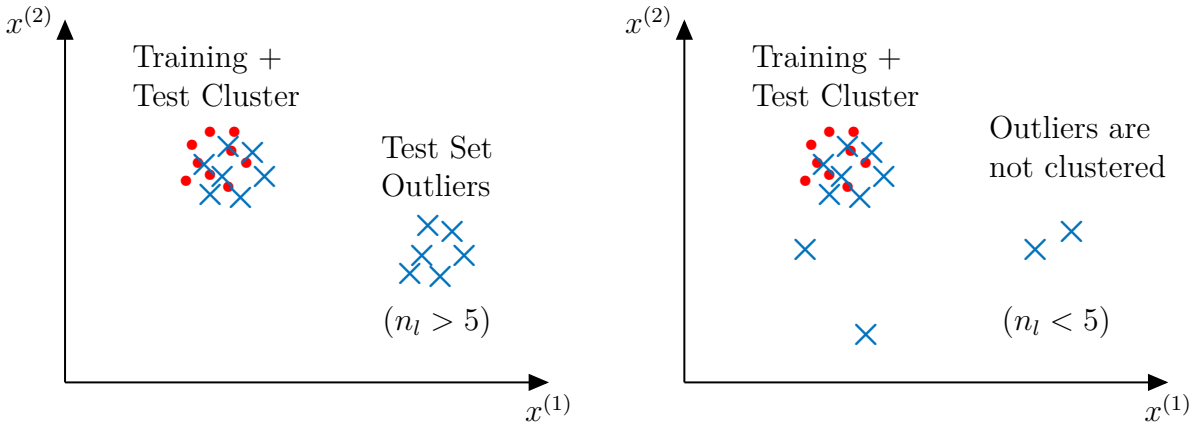
8.3.2 Detection of Random Faults

While the previous algorithm is able to detect systematic faults which are characterized by the fact that every segment in the measurement data belonging to a specific class is modified as it is the case with experiments 1, 2 and 4. Different from that experiment 3 simulates random

faults which affect only a few individual segments rather than every segment in the test cycle run. To detect these kind of faults a different detection approach has to be developed. This algorithm should be able to distiguish whether the randomly occuring faults correspond to a systematic machine fault or whether they are only random noise occuring due to measurement errors or other external or internal factors. This is done by looking at the cluster structure of the test data and counting the number of samples representing a faulty segment. In this case a fault event is considered only random noise whenever the number of segments modified in the same way is smaller than 5 per class. If the number of modified segments is larger than or equal 5 the algorithm assumes, that a systematic machine fault must be present. Despite the plain number of modified segments in the test data the algorithm also considers in which way the segments are modified. If for instance 10 segments are modified, but every segment in a different way, the algorithm does not assume a systematic fault, but rather takes the modified segments as random noise. This is illustrated in fig. 8.8. The left diagram (fig. 8.8a) shows a situtation, where most of the test samples lie close to the training samples, so clusters are overlapping. However there are 6 outliers in the test set, which form a separate cluster. This cluster corresponds to 6 segments which are modified in the same way. Therefore a systematic machine fault is likely. The algorithm first checks, whether more than 5 outliers exist, which is true in this case. Afterwards it evaluates, whether the outliers form a cluster of size 5 or bigger, which is also given. Therefore the fault detection algorithm assumes the machine has a systematic fault in the specific class the outliers belong to. A different situation is shown in the right diagram (fig. 8.8b). Here the number of outliers equals 5, which satisfies the first condition for a systematic fault. However the outliers are distributed randomly in feature space which means each of the outliers corresponds to a segment in the measurement data which is modified in a different way. Therefore the algorithm does not predict a systematic fault, but rather takes the outliers as random noise, produced by for instance a measurement error.

Analysis of the cluster structure of outliers is performed by a second search for outliers of the subset of outliers. Consider the example test dataset shown in fig. 8.9. Here the majority of samples belongs to the main cluster Z_{main} which represents the non-modified segments of this class. Apart from this main cluster the dataset contains a subset $O = \{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_8\}$ of eight outliers. The first six outliers form a separate cluster Z_O representing segments which are modified in the same way. The remaining two outliers $\boldsymbol{o}_7$ and $\boldsymbol{o}_8$ do not form any cluster and represent segments which are modified differently from the segments belonging to the samples in the cluster. For this example the algorithm should predict a systematic machine fault as more than 5 segments are modified in the same way. To achieve this first the entire dataset is searched for outliers by computing the LOF for all samples and labelling samples with high LOF as outlier. In this step the set O of outliers is found. To determine the cluster structure of this subset of samples another outlier search is performed only on this subset. Again the LOF is calculated, however this timeonly for the eight samples in O. The LOF for values lying within the separate cluster Z_O is low, while it is high for the two samples $\boldsymbol{o}_7$ and $\boldsymbol{o}_8$, which are then marked as outliers of the subset O. So $\boldsymbol{o}_7$ and $\boldsymbol{o}_8$ can be seen as outliers of the outliers. By subtracting the amount of outliers of the outliers from the size of the subset O of outliers the cluster size of Z_O can be found which is in this case $8 - 2 = 6$. As it is larger than 5 the algorithm assumes the machine to have a systematic fault.

The programmatic realisation of the described algorithm is illustrated in the flowchart in fig. 8.6b.



(a) Number of outliers in test set is larger than 5 and outliers form a cluster. This corresponds to a systematic machine fault.

(b) Number of outliers in test set is 5. However no cluster of size 5 oor larger is formed, therefore the outliers are considered to be random noise and the machine is assumed to be healthy.

Figure 8.8: Principle of random machine fault detection by outlier detection. For each class outliers are found by comparing the local outlier factor of each test sample with a threshold value. When the numer of outliers per class is below the threshold value of 5 the outliers are directly considered to be random noise and the machine is assumed to be healthy. When the number of outliers is bigger than or equal 5 the algorithm determines whether the outliers are clustered (left diagram) or not (right diagram). If the outliers are clustered a systematic machine fault is assumed. If the outliers do not form any cluster of size 5 or larger they are considered to be random noise and the machine is assumed to be fully operational in the specific class.

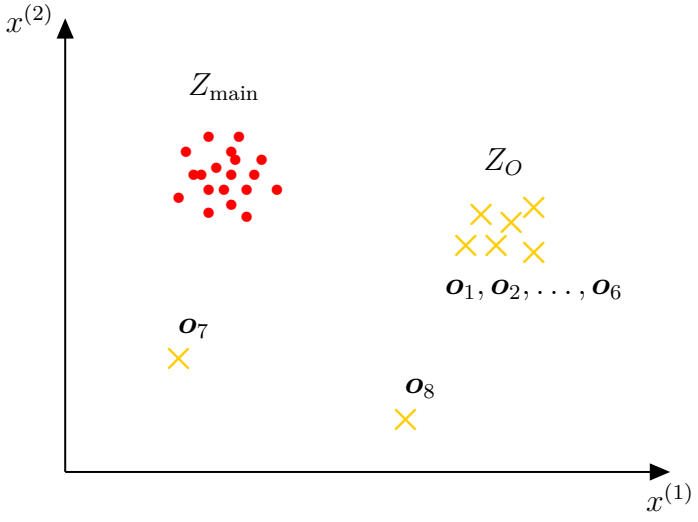


Figure 8.9: Determination of the cluster structure of outliers by finding outliers in the subset of outliers. First the subset of outliers $O = \{\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_8\}$ (orange dots) is found. Afterwards another outlier search is performed on this subset giving samples $\boldsymbol{o}_7$ and $\boldsymbol{o}_8$ as outliers of outliers while the remaining six outliers form a cluster Z_O. As the size of this cluster is larger than 5 a systematic machine fault is assumed.

The displayed subroutine is executed after preprocessing of the training and test dataset as shown in fig. 8.5. The corresponding source code is contained in the fault detection script listed in listing A.20 and listing A.21. Output of this algorithm is a report which specifies the number of outliers per class. When this number is larger than or equal 5 the report states how many of these outliers are clustered and how many are randomly distributed in feature space. When the outliers of a class form a cluster with at least 5 samples the report indicates a systematic fault in this class.

8.4 Experiment Results

After development of suitable algorithms for machine fault detection in the situtations simulated by experiments 1 to 4 (see section 8.2) this sections focuses on evaluation of the detection results. Ideally any faults introduced in experiments 1, 2 and 4 should be detected securely with the first algorithm presented in section 8.3.1 which detects machine faults based on shift of cluster centroids. Analogous the modified test data of experiment 3 is fed into the second algorithm introduced in section 8.3.2 which features density based outlier detection to find random faults. Here the algorithm should predict an according machine fault as well. To evaluate results the training dataset as well as the according modified test dataset are visualized in a scatterplot representing feature space. As the feature space is spaned by 10 features and therefore 10-dimensional a 2-dimensional projection is utilized. According to the results found in section 7.3.2 the projection onto the plane spanned by the peak to peak value s_{p2p} and the mean absolute deviation d_{mean} has highest information gain and is therefore choosen at this point to visualize the datasets. In all scatterplots data samples belonging to the training data are plotted as coloured dots and samples of the corresponding test set are plotted as coloured crosses. In cases of experiment 1, 2 and 4 cluster centroids of the training dataset are illustrated by black dots and cluster centroids of the test datasets as black crosses. Black lines between the centroids show how far corresponding training and testing cluster of a class are shifted.

Starting with the first experiment where all 100 segments of class 6 within the test dataset are modified in the same way, the fault detection algorithm finds the results illustrated in fig. 8.10. Here only a small excerpt of the entire feature space is shown containing only classes 6, 8 and 9. As can be seen training and test clusters of classes 8 and 9 which correspond to forward and backward movement of the c-axis are very similar. So their centroids lie close to each other. However the test cluster for class 6 is significantly shifted compared to the cluster of class 6 in the training set. This is due to the fact that all segments in the test dataset are modified and therefore have different characteristic feature values than the accoridng segments in the training set. Apart from the three depicted classes the datasets contains 9 more classes. They are not shown to be able to zoom further into the relevant area of class 6. Training and test clusters of all other classes lie very close to each other similar as it is the case for classes 8 and 9. This is reasonable as no other segments apart from the segments of classs 6 are modified in the first experiment. So distances between training and testing clusters of all classes except for class 6 are small. Distance between the clusters of class 6 is $d_6 = 1.432$ (manhattan distance) which exceeds the threshold value of $d_{\mathrm{thrs}} = 0.2$. Therefore the algorithm correctly assumes a machine fault in class 6, while it predicts the machine as healthy in all other classes.

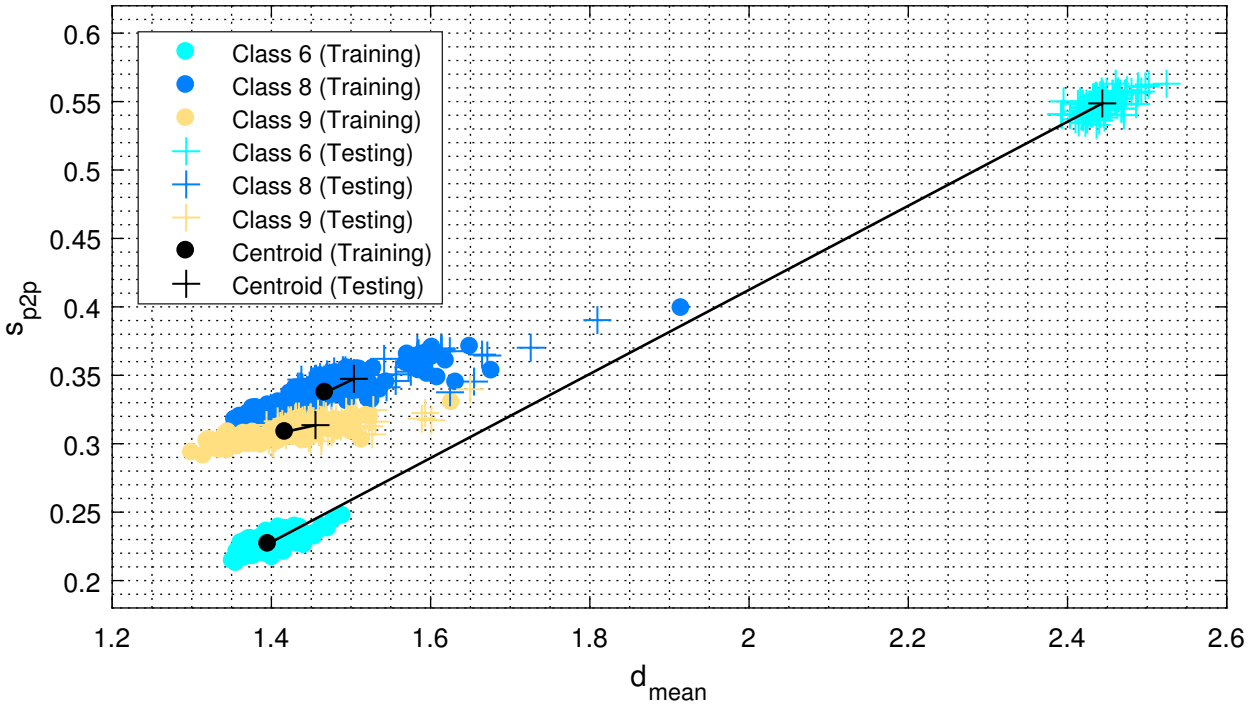


Figure 8.10: Result of the first experiment for machine fault detection. The scatterplot shows training (colored dots) and testing samples (colored crosses) of classes 6, 8 and 9 in a two-dimensional projection of feature space as well as the six centroids (black dots and crosses). Centroids are mean values of all samples in the training and testing subsets of the three classes. When training and testing set are very similar as it is the case for classes 8 and 9 distance between the centroids (black lines) are small and nor fault is detected. However if the distance is large as it is the case for class 6 ($d_6 = 1.432$), the test set has a systematic deviation from the training set and a machine fault can be assumed.

As the second experiment is similar to the first one so are the results of fault detection. While in the first experiment only segments of class 6 are modified additionally all 100 segments of class 7 occuring in the test set are modified in the second experiment. Accordingly training and test clusters of both classes 6 and 7 show a significant shift in feature space as can be seen in fig. 8.11. Here a bigger extract of feature space containing training and test clusters of classes 6 to 11 is shown. As can be seen distances between training and test clusters of classses 8, 9, 10 and 11 are very small which means the corresponding segments in training and test set are very similar which is reasonable as they are not modified during this experiment. However training and test clusters of classes 6 and 7 are clearly separated in feature space. The manhattan distance between the clusters of class 6 is still $d_6 = 1.432$ while distance between the clusters of classs 7 is 1.607. Therefore both exceed the threshold value and the algorithm rightly detects the simulated machine fault in both classes 6 and 7, while it does not assume a machine fault in any other class.

Because the fourth experiment is more similar to experiments 1 and 2 than the third experiment, evaluation of the results for this experiment is antedated. As described in section 8.2.2 modifications are introduced in classes 1, 6 and 7 by removing the tool clampend into the main spindle

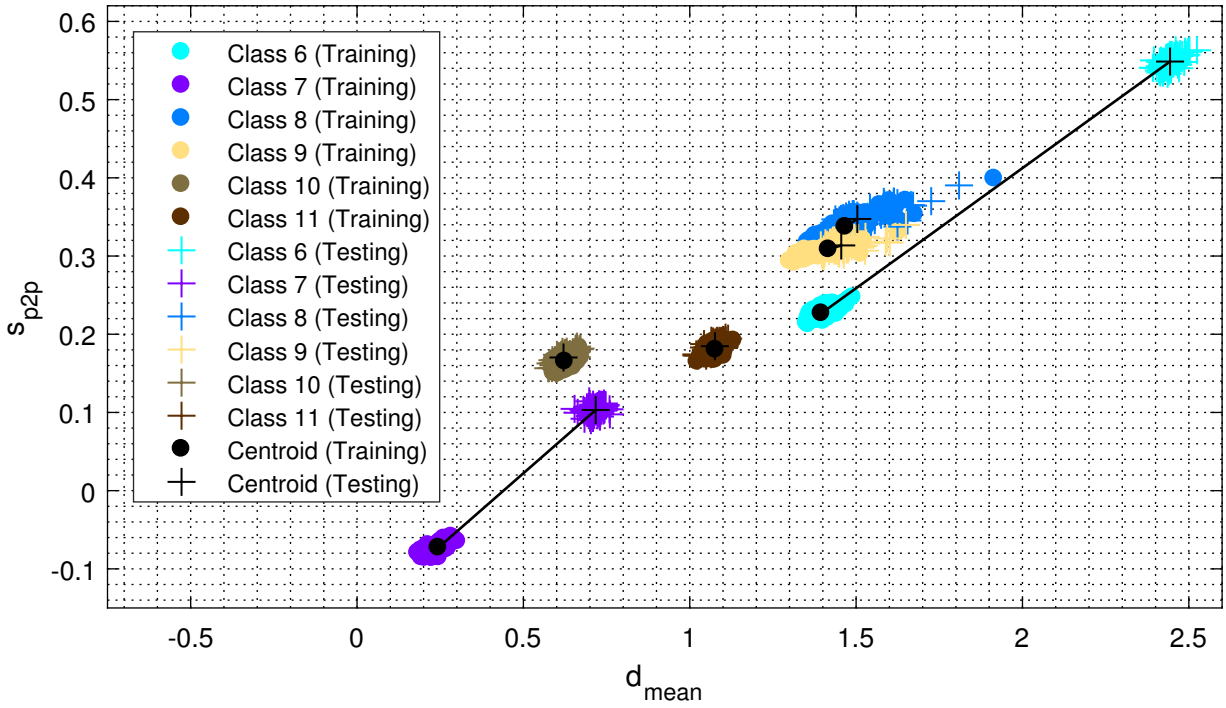


Figure 8.11: Result of the second experiment for machine fault detection. Meaning of the plot elements is identical to fig. 8.10, however additional classes 7, 10 and 11 are shown. Distance between clusters of training and testing data of classes 6 and 7 is $d_6 = 1.432$ and $d_7 = 1.607$. These values exceed the threshold of $d_{\text{thrs}} = 0.2$, so systematic machine fault in classes 6 and 7 is recognized.

and adding an additional weight to the machine table. However as already mentioned in this section the influence of the additional weight on class 6 is minor and therefore could be hard to detect by the fault detection algorithm. The effect on class 7 is significant, though, wherefore the algorithm should detect a systematic machine fault in this class. The influence of the removed tool on the current consumption of the main spindle drive is also small, however should be detected by the algorithm to validate its proper functionality. The according scatterplot of both the training and test dataset for this experiment is shown in fig. 8.12. This time the entire feature space containing all 12 classes is shown. However the training and test cluster of class 1 is shifted from its original position around the point $(2.9, 4.4)$ to the depicted one for more compact plotting of the datasets. As can be seen training and testing clusters of all 12 classes lie much closer to each other as it is the case within the two prior experiments. This is due to the fact that the influence of the machine modifications on the measurement signal is much smaller than the synthetic modifications introduced to the measurement signal within experiments 1 and 2. Despite that distances between each pair of clusters can be calculated and compared with the threshold value $d_{\text{thrs}} = 0.2$. It turns out that the manhattan distance between training and test clusters of classes 1 and 7 exceed that threshold. The distance between cluster of class 1 is $d_1 = 0.559$ and the one between clusters of class 7 is $d_7 = 0.232$. Therefore the algorithm correctly predicts machine faults in those two classes. However as already noticed the change in the segments belonging to class 6 are to small to be detected. The manhattan distance of training and test clusters of class 6 is $d_6 = 0.075$ which is slightly higher than the distance

between the cluster pairs of classes 0, 2, 3, 4 and 5 and slightly lower than the distance between cluster of classes 8, 9, 10 and 11. Therefore there is no chance to isolate a machine fault in class 6 by changing the distance threshold value. Looking back at the modified test measurement data shown in fig. 8.3 it can be said that the difference between the training and the testing set is so small for classs 6 that the missing detection of a fault in class 6 is unproblematic. This is especially true when considering that machine faults in classes 1 and 7 are easily recognized by the algorithm.

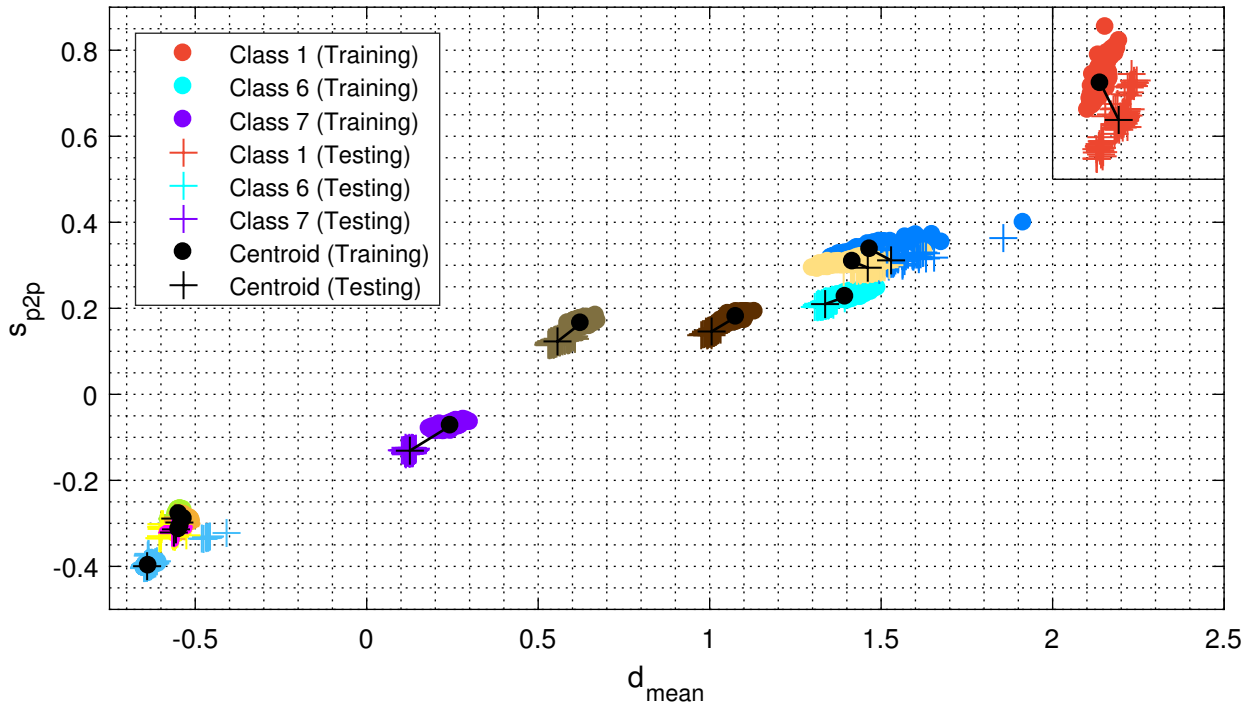


Figure 8.12: Result of the fourth experiment for machine fault detection. Clusters of classes 1 and 7 are significantly shifted and have a distance of $d_1 = 0.559$ respectively $d_7 = 0.232$ which exceeds the distance threshold $d_{\text{thrs}} = 0.2$. Therefore the algorithm detects systematic machine faults in classes 1 and 7. Cluster shifts of all other classes lie below the threshold and therefore no faults are assumed in these classes. For easier plotting cluster 1 is shifted from its original position around the point $(2.9, 4.4)$ to the depicted location.

Finally the third experiment is evaluated. As described in detail in section 8.2.1 this experiment differs from experiments 1, 2 and 4 in the way that not all segments of a specific class are modified, but only a few instances. This simulates random machine faults occuring not always, but only randomly and only for a short period of time. To be precise in this experiment 10 segments of class 6, 5 segments of class 7 and 2 segments of class 2 are randomly picked out and modified. Furthermore one additional segment of class 6 is modified, however differently from the other segments of this class. To detect these random faults the cluster shift algorithm is no longer applicable, but rather the outlier detection algorithm developed in section 8.3.2 is applied to the problem. Expected outcome of using the algorithm on this test dataset would be first detection of all 18 outliers. As class 2 contains only 2 outliers no machine fault should be predicted in this class, but instead the outliers should be taken as random noise. Different from that the algorithm should report systematic machine faults in both classes 6 and 7 as the

number of outliers in this classes is larger than or equal the threshold value 5. The algorithm should also detect the outlier of the outlier in class 6 which means it should detect that one of the segments of class 6 is modified differently compared to the other modified segments of this class. As the scatterplot in fig. 8.13 shows these expectations are fullfilled. The scatterplot shows training and testing clusters of all classes except for class 0 and 1 as these lie outside of the plot range and are not of interest for this experiment. Found outliers in the test dataset are surrounded by a red circle and the only outlier of outliers of class 6 is surrounded by a red square. As can be seen all outliers are found correctly and even the outlier of outliers is detected. Interesting baout this sample is that it lies directly within the training and testing cluster of class 7. However this is no problem for detection of the outlier as outliers are detected separately for each class. So when finding the outliers of class 6 this point is already found, because samples of all other classes are not considered. When now another outlier search is conducted within the subset of 11 outliers of class 6 the according outlier of the outliers is easily found. So the algorithm has no problem with detecting random faults even when classes overlap as it calculates outliers for each class separately. So all in all it can be said that performance of the random fault detection algorithm is sufficient.

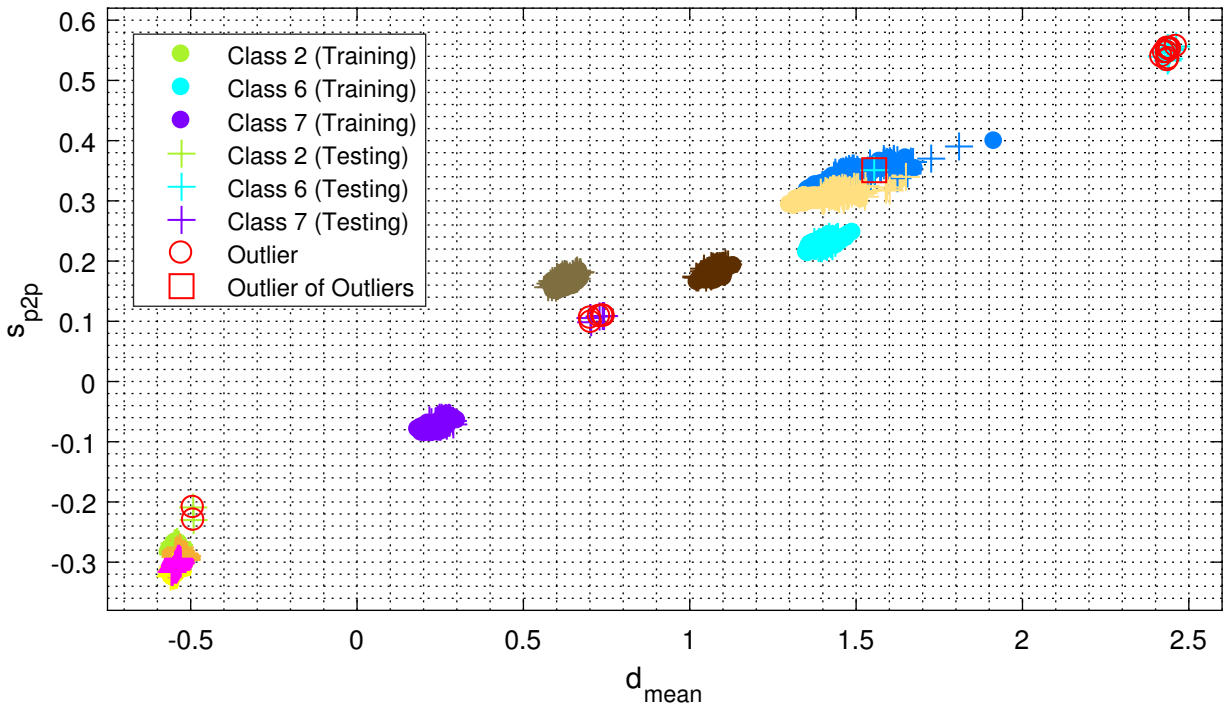


Figure 8.13: Result of the third experiment for machine fault detection. Individual outliers (red circles) of classes 2, 6 and 7 are found by thresholding the local outlier factor of each point. Moreover the algorithm finds an outlier within the subset of outliers of class 6 by performing a second local outlier factor thresholding. This sample is marked with a red square and corresponds to the individual modification of a segment of class 6 according to fig. 8.2d. Only when the subset of outliers contains more than 5 densely lying samples per class a systematic fault in this class is assumed. Otherwise outliers are taken to be noise.

8.5 Conclusion of Machine Fault Detection

In the preceeding chapter two different algorithms were developed which were able to detect systematic and random machine faults. Prior to this the local outlier factor as measure for density based outlier detection was mathematically defined and explained. Afterwards four different experiments were designed to simulate different kind of faults which might occur during machine operation. For each experiment a modified test dataset containing one or multiple exactly specified machine faults was generated. In case of the first three experiments this was done by applying mathematical functions to the already present test measurement data and in case of the last experiment measurement data was acquired directly from the machine. To simulate a machine fault in this expeirment first the tool was removed from the main spindle and and additional weigth was placed on the machine table. After presentation of the resulting test datasets two different fault detection algorithms were developed. The first one was able to detect systematic faults which affected all segments in the test datasets and therefore shifted all samples of the training dataset by the same distance in feature space. This algorithm computed the distance between corresponding class clusters in the training and test datasets and predicted a machine fault in a class whenever this distance exceeded a specific threshold value. This algorithm was applied to the test datasets of experiment 1, 2 and 4, because in these experiments all segments of a specific class type were modified. On the opposite to that another algorithm was developed which was able to cope with random machine faults. This kind of fault affected only a few segments of the same class rather than all segments. The algorithm found outliers for each class in feature space by calculating the previously introduced local outlier factor for each sample within the test dataset and comparing it to a threshold value. The found outliers were then counted and whenever less than five outliers occured in a class they were taken as random noise rather than a machine fault. However when their number exceeded 5 per class another outlier detection was performed on the subset of outliers to analyse the cluster structure of the outliers. If all outliers of a class were found to lie within a cluster, which means all segments were modified in the same way, a machine fault was predicted in that class. Otherwise the outliers were again just neglected as random noise. After implmentation of the developed algorithms in Python the generated test datasets of all four experiments were fed into the according algorithm and detection results were evaluated. It tunred out that the fault detection algorithms found all synthetically modified segments in the test datasets of the first three experiments. Moreover performance on the real measurement data acquired on the modified machine was very good as well. The algorithm detected a machine fault in classs 7 corresponding to the upward movement of the z-axis and a fault in class 1 which was reasoned by the removed tool. Only the sligth differences in class 6 could not be detected by the algorithm.

Despite performing as expected and classifying nearly all machine faults correctly the fault detection algorithms still have potential for improvement. The first improvement relates to the overall structure of the implementation of the proposed algorithms. To evaluate their performance on the generated test datasets a simple script (see listing A.20) was sufficient. However to make them applicable in other use cases as well they should be wrapped in a suitable application programming interface (API) offering methods for fitting the detection algorithm to a training dataset and classifying faults on a corresponding test dataset. This would also comprise methods for automatically determining appropiate values for all free parameters within

the algorithms such as the number of neighbours cosnidered during computation of the LOF or any of the threshold values used to distiguish between faulty and healthy condition.

Apart from this both algorithms could be merged in one bigger algorithm comprising both detection of random faults and systematic faults. In the preceeding analysis the two different kinds of algorithms were applied separately to the generated test datasets. This was possible, because the datasets represetented a very controlled case of machine fault. In praxis however a mixture of both random and systematic fault might occur. Consider a case where nearly all segments of a class except for only a few are modified the same way. The few outliers might for instance be measurement errors. In this case the algorithm threshdoling distances between cluster centroids would usually be applicable. However because of the outliers in one class of the test dataset the cluster centroid of the corresponding class would be erroneously shifted leading to a different distance measure between the training and test cluster of the according class. Depending on the positions of the outliers this directly affects the outcome of the prediction algorithm. Merging both algorithms developed above could prevent this problem. If first an outlier search would be performed and outliers removed from the test dataset the fault detection algorithm would then be applicable again and achieve its full performance. SO in order to make the fault detection algorithm useful for practical application further work has to be done.

A last problem relates to the present implementation of the second fault detection algorithm utilizing outlier detection. Here a systematic fault was predicted whenever the number of outliers exceedeed 5 and when these outliers were clustered. However analysis of the cluster structure is performed by finding outliers in the subset of outliers. This approach works for the given problem, however might lead to problem in different situations. If for example 8 outliers exist in a class, of which 2 are global outliers having no close neighbours and 3 outliers form a cluster each. The proposed algorithm would now find the two global outliers in this subset of outliers and reduce the number of toal outliers to 6 which exceeds 5 leading to the message that a systematic machine fault is present in the particular class. However from the six modified segments each three are modified differently, which is likely to be only random noise. However the algorithm assumes this to be a systematic fault. To improve this behviour a more sophisticated approach for analysis of the cluster structure like k-means has to be utlized.

Apart from the mentioned problem further research should also focus on evaluating the next stage of the fault detection algorithm on a bigger set of real measurement data comprising more different kind of machine faults than the one used above. For instance it should be analyzed whether the expected random faults really occur during operation of the machine or if they are not existent at all. If the latter one would apply detection of machine faults could be done by only considering shift of cluster centroids rather than by considering possible outliers in the test dataset.

9 Conclusion

Summarizing the preceeding work in chapter 2 a brief overview over the theoretical concepts of knowledge discovery in databases, machine learning, pattern recognition and anomly detection was given. Subsequently in chapter 3 the internal structure of the milling machine DMU 100 Monoblock® which was object of this project was analyzed in detailed. Contained components were characterized regarding their electrical characteristics as well as regarding their activation during execution of a typical machine program. Here two different kinds of components were identified, first controllable components such as axis, which can be fully controlled by the machine controller and second non-controllable components which are activated automatically. In chapter 4 measurement hardware and software neccessary for acquiring data from the main power line of the machine was built up. Furthermore machine states were defined as possible switching states of the contained commponents within the machine. Here some components were neglected to reduce the problem complexity and number of different states to 96, respectively 12 when further neglecting any non-controllable components. Afterwards a specific machine program was written which allowed for isolated observation of the impact of activating individual components on the overall current consumptions of the machine. Running this program three times gave two training datasets and one test dataset each comprising 2200 segments which refered to one movement of an axis each. To enable further analysis of this datasets an algorithm for segmentation of the measurement data was developed in chapter 5. This algorithm used shape templates which were slid over the entire measurement sequence and by evaluating a similarity measure between the template and the measurement data appropriate segment cuts could be placed. Utilizing further information about the switching states of the non-controllable components additional segment cuts were inserted at positions where non-controllable aggregats were automatically activated. The so segmented measurement data was then labeled meaning each segment was assigned an integer number representing one of the previously defined machine states. This gave two labelsets for each of the three acquired datasets, one comprising only the 12 nominal state changes of the controlable components and one containing all 96 possible states of conotrollable and non-controllable components. With the help of these labelsets and the segmented measurement data a set of different classifiers utilizing both classical signal processing methods as well as modern machine learning methods could be developed in the two subsequent chapters. In chapter 6 two shape based classification approaches were developed and compared. Fist a classifier based on the Dynamic Time Warping distance as useful distance measure between two time sequences of different length was developed. Second a cross-correlation-based classifier which expressed correlation between two time sequences by computing the Pearson correlation coefficient was devised. Both algorithms utilized a previously created shape template dictionary containing current signal patterns for each of the 12 nominal classes. So the algorithms had to compare each segment in the test datasets with each shape template to determine which template fit best and assign the corresponding class label of that template to the segment. The shape template dictionary was built up utilizing the Accurate Shape Averaging algorithm which

created templates by averaging over all 100 segments of each class within the first training dataset. It turned out that the dynamic time warping classsifier was able to distiguish between classes with an accuracy of 97 % after algorithm parameters had been optimized by an exhaustive grid search. The cross-correlation classifier only achieved an accuracy of 77.95 %. In chapter 7 modern machine learning algorithms were applied to the same problem. More precisely k-nearest-neighbour, decision tree, random forest, AdaBoost, support vector machine, stochastic gradient descent and naive bayes classifiers were used to predict which of the 12 nominal machine states is present in each of the 2200 segments within the test dataset. To apply these algorithms first the training and test data was transformed from time domain into a 49-dimensional feature space by exctraction of statistical features. These 49 features were then ranked by means of a decision tree that maximized information gain and the 10 most relevant features were selected to decrease dimensionality of the datasets. Comparing results of the seven applied classsifier models showed that the random forest classifier performed best with an overall classification accuracy of 97.7 %. After having succesfully developed classifiers to predict machine states based only on measurements of the overall current signal of the milling machine the problem of detecting machine faults was addressed in chapter 8. Here four test datasets containing modified segments were created to simulate different kinds of machine faults. The first three datasets were created synthetically by applying mathematical functions to the already present test dataset in the time domain. The fourth dataset was experimentally acquired after modifying the milling machine slightly by adding an additional weight to the machine table and removing the tool from the main spindle. After generation of the modified test datasets the ten slected features from chapter 7 were extracted and standardized to create a feature space representation of the modified test datasets. These were then compared to the training datasets by computing the manhattan distance between cluster centroids of each class. Whenever this distance exceeded a threshold value a machine fault was assumed in that particular class. So it was possible to detect faults on component level rather than only determining if the machine is operating properly or if there is any fault somewhere in the machine. Another algorithm developed in that chapter addressed the problem of detecting random faults by implementing an outlier detection by means of comparing the local outlier factor of each test sample with a threshold value. Both developed fault detection algorithms performed well and were able to detect faults introduced in the synthetic test sets as well as in the real measurement data of the modified machine.

Referring back to the two main goals of this work of first developing algorithms for classifying machine states and second detecting machine faults, it can be said that the work prevailed. However there are some limitations ragrding the project goals in general as well as their realization in the preceeding work. The first and probably most important issue is the fact that the developed classifiers are only capable of detecting the 12 controllable machine states rather than all previously defined 96 machine states which also include non-controllable components. If a predicitve health monitoring system shall be developed which is able to determine switching states of all contained components it is insufficient to simplify the problem to only controllable components as done here. Rather than that all components contained in the machine tool should be considered, however as this is a very large number a new system for defining machine states has to be developed or at least methods need to be applied to synthetically increase the amount of training data as it is not convenient to acquire the neccessary amount of training data experimentally. Another problem with the current state of the project is that the algorithm

are only applicable to the specific configuration of the DMU 100 Monoblock® as it is present at the Institute of Machine Tools and Production Technology of the TU Brunswick. Making the proposed algorithm work with other machine tools as well involves redefinition of machine states and analysis of the structure of the new machine. Probably the proposed method of isolating different the influence of different components on the current signal on the main power line would also not work as the fact was exploited that no non-controllabel component was connected to phase L2 of the main power line. Future research should consider developing a method which has no need for isolated component impacts on the training data. Another problem regards the fact that all proposed methods operate offline on a bounded datasets which was acquired earlier. To make the algorithms useful for development of a predictive health monitoring system which observes machine state as well as health condition continouusly they should be made online-capable meaning they should base any predictions only on a stream of test data rather than an entire bounded datatset. To support streaming of test data and to generate a continouus rather than a static output of machine satates the segmentation algorithm has to be redesigned as well. Instead of segmenting the measurement data based on context it should be segmented into equidistant portions which are then classified by an appropiate classifiaction model giving a continouus output of the machine state. The same applies to the fault detection algorithm which operates also only offline at this point of the project. Finally all algorithms should be wrapped in an appropiate application programming interface providing well documented methods for conducting machine state predicition and fault detection. Despite those limitations the work can be considered as success as the aim was not to build a marked-ready product for health condition monitoring, but instead should provide a set of algorithms useful for later developing of such a system. Moreover limitations arise from the fact that the milling machine as object of research is a highly complex technical system containing a large amount of different components. So in order to elaborate the basic principles of state predicition and fault detection it was necceassary to reduce the complexity of the problem to get a better understanding of the underlying principle and develop easy to understand methods for addressing the project goals. Finally the project is also an accomplishment as it has shown that it is possible to detect machine states and the machine health condition based only on measurement of the overall current consumption of the machine. This proves feasability of a cheap and flexible predicitve health monitoring system operating only on measurement of the main current signal.

Based on the aforementioned limitations a suggestion for a future work on the topic like a master thesis can be made. This future work could focus on building a practically usable solution for predicitve health monitoring of a broader range of machine tools by not focusing to heavily on machine specific facts, but rather try to implement a more general state recognition system based on measurement of the overall current consumption of a machine. To make this possible at the first place the future work should look at a simpler machine tool than the one considered in the preceeding work. This could be for example a simple grinding machine containing only a few fully controllable internal components affecting current consumption on the main power line. Instead of focusing on a broader range of machine tools future work could also try to build a ready-to-use health condition monitoring solution for the DMU 100 Monoblock®. This solution should be able to predict switching states of every internal component of the machine online by only considering measurement data of the current consumption on the main power line. Moreover component states as well as machine faults should be detected always regardless

of the kind of program being executed on the machine. For this a much more complicated model describing the impact of different machine components on the current signal needs to be developed. To do this future work could redefine machine states by not looking at any possible combination of internal component states, but instead determining states of the components independently. So each component could have a finite number of states which affect overall current consumption in a well known manner. The state detection model than has to predict switching states of the individual components rather than an overall machine state based on the combination of component states. Besides that the solution developed in future worked should also incorporate a new method for detecting machine faults which is capable of predicting the machine health state online.

Finally it can be said that the research conducted in the preceeding work is highly relevant as it provides solutions for predictive health monitoring which is one of the currently most important subfields in manufacturing technology. It provides the opportunity to massively improve machine operation in production lines as well as operation of individual machines in smaller companies as it reduces the risk of sudden machine break down and standstill of prodution lines. Moreover it enables gaterhing information about the machine tools within a copany which could be further used to analyze usage of the machine and frequently occuring problems with specific machines. This information could be used to improve quality of future machine tools and quality of machined products as well as efficiency of the entire production. Moreover the research follows the current trend of utilizing state of the art big data and machine learning methods which belong to the fastest evolving technologies nowadays.

Bibliography

[1] Caesarendra, W., Widodo, A., Thom, P. H., et al. "Combined Probability Approach and Indirect Data-Driven Method for Bearing Degradation Prognostics". In: *IEEE Transactions on Reliability* 60.1 (2011), pp. 14–20. ISSN: 0018-9529. DOI: 10.1109/TR.2011.2104716.

[2] Caesarendra, W., Widodo, A., and Yang, B.-S. "Combination of probability approach and support vector machine towards machine health prognostics". In: *Probabilistic Engineering Mechanics* 26.2 (2011), pp. 165–173. ISSN: 02668920. DOI: 10.1016/j.probengmech.2010.09.008.

[3] Caesarendra, W. "Vibration and acoustic emission-based condition monitoring and prognostic methods for very low speed slew bearing". Dissertation. University of Wollongong, 2015. URL: http://ro.uow.edu.au/cgi/viewcontent.cgi?article=5487&context=theses (visited on 2017-04-19).

[4] Caesarendra, W., Kosasih, B., Tieu, A. K., et al. "Acoustic emission-based condition monitoring methods: Review and application for low speed slew bearing". In: *Mechanical Systems and Signal Processing* 72-73 (2016), pp. 134–159. ISSN: 08883270. DOI: 10.1016/j.ymssp.2015.10.020.

[5] Schoen, R. R., Habetler, T. G., Kamran, F., et al. "Motor bearing damage detection using stator current monitoring". In: *IEEE Transactions on Industry Applications* 31.6 (1995), pp. 1274–1279. ISSN: 00939994. DOI: 10.1109/28.475697.

[6] Youn, Y.-W., Hwang, D.-H., Sun, J.-H., et al. "A Method for Indentifying Broken Rotor Bar and Stator Winding Fault in a Low-voltage Squirrel-cage Induction Motor Using Radial Flux Sensor". In: *Journal of Electrical Engineering and Technology* 6.5 (2011), pp. 666–670. ISSN: 1975-0102. DOI: 10.5370/JEET.2011.6.5.666.

[7] Seera, M., Lim, C. P., Nahavandi, S., et al. "Condition monitoring of induction motors: A review and an application of an ensemble of hybrid intelligent models". In: *Expert Systems with Applications* 41.10 (2014), pp. 4891–4903. ISSN: 09574174. DOI: 10.1016/j.eswa.2014.02.028.

[8] Siddique, A., Yadava, G. S., and Singh, B. "A Review of Stator Fault Monitoring Techniques of Induction Motors". In: *IEEE Transactions on Energy Conversion* 20.1 (2005), pp. 106–114. ISSN: 0885-8969. DOI: 10.1109/TEC.2004.837304.

[9] Singh, S., Kumar, A., and Kumar, N. "Motor Current Signature Analysis for Bearing Fault Detection in Mechanical Systems". In: *Procedia Materials Science* 6 (2014), pp. 171–177. ISSN: 22118128. DOI: 10.1016/j.mspro.2014.07.021.

[10] Ogidi, O. O., Barendse, P. S., and Khan, M. A. "Fault diagnosis and condition monitoring of axial-flux permanent magnet wind generators". In: *Electric Power Systems Research* 136 (2016), pp. 1–7. ISSN: 03787796. DOI: 10.1016/j.epsr.2016.01.018.

[11] Yang, D., Li, H., Hu, Y., et al. "Vibration condition monitoring system for wind turbine bearings based on noise suppression with multi-point data fusion". In: *Renewable Energy* 92 (2016), pp. 104–116. ISSN: 09601481. DOI: 10.1016/j.renene.2016.01.099.

[12] Geramifard, O., Xu, J.-X., Pang, C. K., et al. "Data-driven approaches in health condition monitoring — A comparative study". In: *8th IEEE International Conference on Control and Automation (ICCA), 2010*. Piscataway, NJ: IEEE, 2010, pp. 1618–1622. ISBN: 978-1-4244-5195-1. DOI: 10.1109/ICCA.2010.5524339.

[13] Geramifard, O., Xu, J.-X., Zhou, J. H., et al. "Continuous health assessment using a single hidden Markov model". In: *11th International Conference on Control, Automation, Robotics & Vision (ICARCV), 2010*. Piscataway, NJ: IEEE, 2010, pp. 1347–1352. ISBN: 978-1-4244-7814-9. DOI: 10.1109/ICARCV.2010.5707866.

[14] Geramifard, O., Xu, J.-X., Zhou, J.-H., et al. "A Physically Segmented Hidden Markov Model Approach for Continuous Tool Condition Monitoring: Diagnostics and Prognostics". In: *IEEE Transactions on Industrial Informatics* 8.4 (2012), pp. 964–973. ISSN: 1551-3203. DOI: 10.1109/TII.2012.2205583.

[15] Geramifard, O., Xu, J.-X., Zhou, J.-H., et al. "Multimodal Hidden Markov Model-Based Approach for Tool Wear Monitoring". In: *IEEE Transactions on Industrial Electronics* 61.6 (2014), pp. 2900–2911. ISSN: 0278-0046. DOI: 10.1109/TIE.2013.2274422.

[16] Geramifard, O. and Tung, L. "Multi-model diagnostics for various machining conditions: A similarity-based approach". In: *IECON 2015 - Yokohama*. Ed. by Ohishi, K. and Hashimoto, H. Piscataway, NJ: IEEE, 2015, pp. 000333–000338. ISBN: 978-1-4799-1762-4. DOI: 10.1109/IECON.2015.7392121.

[17] Leem, C. S. and Dornfeld, D. A. "Design and implementation of sensor-based tool-wear monitoring systems". In: *Mechanical Systems and Signal Processing* 10.4 (1996), pp. 439–458. ISSN: 08883270. DOI: 10.1006/mssp.1996.0031.

[18] El-Wardany, T. I., Gao, D., and Elbestawi, M. A. "Tool condition monitoring in drilling using vibration signature analysis". In: *International Journal of Machine Tools and Manufacture* 36.6 (1996), pp. 687–711. ISSN: 08906955. DOI: 10.1016/0890-6955(95)00058-5.

[19] Abu-Mahfouz, I. "Drilling wear detection and classification using vibration signals and artificial neural network". In: *International Journal of Machine Tools and Manufacture* 43.7 (2003), pp. 707–720. ISSN: 08906955. DOI: 10.1016/S0890-6955(03)00023-3.

[20] Li, X., Dong, S., and Venuvinod, P. K. "Hybrid Learning for Tool Wear Monitoring". In: *The International Journal of Advanced Manufacturing Technology* 16.5 (2000), pp. 303–307. ISSN: 0268-3768. DOI: 10.1007/s001700050161.

[21] Balazinski, M., Czogala, E., Jemielniak, K., et al. "Tool condition monitoring using artificial intelligence methods". In: *Engineering Applications of Artificial Intelligence* 15.1 (2002), pp. 73–80. ISSN: 09521976. DOI: 10.1016/S0952-1976(02)00004-0.

[22] Doukas, C., Stavropoulos, P., Papacharalampopoulos, A., et al. "On the Estimation of Tool-wear for Milling Operations based on Multi- Sensorial Data". In: *Procedia CIRP* 8 (2013), pp. 415–420. issn: 22128271. doi: 10.1016/j.procir.2013.06.126.

[23] Kilundu, B., Dehombreux, P., and Chiementin, X. "Tool wear monitoring by machine learning techniques and singular spectrum analysis". In: *Mechanical Systems and Signal Processing* 25.1 (2011), pp. 400–415. issn: 08883270. doi: 10.1016/j.ymssp.2010.07.014.

[24] Geramifard, O., Zhi, Z. Y., Quan, C. Y., et al. "Power-signature-based Bayesian multi-classifier for operation mode identification". In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. Piscataway, NJ: IEEE, 2016, pp. 1–6. isbn: 978-1-5090-1314-2. doi: 10.1109/ETFA.2016.7733530.

[25] Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. "From Data Mining to Knowledge Discovery in Databases". In: *AI Magazine* 17 (1996), pp. 37–54.

[26] Kohavi, R., Rothleder, N. J., and Simoudis, E. "Emerging trends in business analytics". In: *Communications of the ACM* 45.8 (2002). issn: 00010782. doi: 10.1145/545151.545177.

[27] Tyagi, S. "Using data analytics for greater profits". In: *Journal of Business Strategy* 24.3 (2003), pp. 12–14. issn: 0275-6668. doi: 10.1108/02756660310734938.

[28] Koh, H. C. and Tan, G. "Data mining applications in healthcare". In: *Journal of healthcare information management* 19.2 (2011), p. 65.

[29] Kaur, H. and Wasan, S. K. "Empirical Study on Applications of Data Mining Techniques in Healthcare". In: *Journal of Computer Science* 2.2 (2006), pp. 194–200. issn: 15493636. doi: 10.3844/jcssp.2006.194.200.

[30] Wang, K. "Applying data mining to manufacturing: The nature and implications". In: *Journal of Intelligent Manufacturing* 18.4 (2007), pp. 487–495. issn: 0956-5515. doi: 10.1007/s10845-007-0053-5.

[31] Choudhary, A. K., Harding, J. A., and Tiwari, M. K. "Data mining in manufacturing: A review based on the kind of knowledge". In: *Journal of Intelligent Manufacturing* 20.5 (2009), pp. 501–521. issn: 0956-5515. doi: 10.1007/s10845-008-0145-x.

[32] Gröger, C., Niedermann, F., and Mitschang, B. "Data Mining Driven Manufacturing Process Optimization". In: *Proc. of the World Congress on Engineering* 3 (2012), pp. 4–6.

[33] Bhowmik, R. "Data mining techniques in fraud detection". In: *The Journal of Digital Forensics, Security and Law* 35.2 (2008).

[34] Golmohammadi, K. and Zaiane, O. R. "Data Mining Applications for Fraud Detection in Securities Market". In: *European Intelligence and Security Informatics Conference (EISIC), 2012*. Ed. by Memon, N. Piscataway, NJ: IEEE, 2012, pp. 107–114. isbn: 978-1-4673-2358-1. doi: 10.1109/EISIC.2012.51.

[35] Bănărescu, A. "Detecting and Preventing Fraud with Data Analytics". In: *Procedia Economics and Finance* 32 (2015), pp. 1827–1836. ISSN: 22125671. DOI: 10.1016/S2212-5671(15)01485-9.

[36] Chen, H., Chung, W., Xu, J. J., et al. "Crime data mining: A general framework and some examples". In: *Computer* 37.4 (2004), pp. 50–56. ISSN: 0018-9162. DOI: 10.1109/MC.2004.1297301.

[37] Xu, J. and Chen, H. "Criminal network analysis and visualization". In: *Communications of the ACM* 48.6 (2005), pp. 100–107. ISSN: 00010782. DOI: 10.1145/1064830.1064834.

[38] Wang, J. T. L., Zaki, M. J., Toivonen, H. T. T., et al. "Introduction to Data Mining in Bioinformatics". In: *Data Mining in Bioinformatics*. Ed. by Wu, X., Jain, L., Shasha, D., et al. Advanced Information and Knowledge Processing. London: Springer-Verlag London Limited, 2005, pp. 3–8. ISBN: 1-85233-671-4. DOI: 10.1007/1-84628-059-1_1.

[39] Raza, K. "Application of Data Mining in Bioinformatics". In: *Indian Journal of Computer Science and Engineering* 1.2 (2010), pp. 114–118.

[40] Runkler, T. A. *Data Mining: Modelle und Algorithmen intelligenter Datenanalyse*. 2., aktualisierte Auflage. Computational Intelligence. Wiesbaden: Springer Vieweg, 2015. ISBN: 978-3-8348-1694-8. DOI: 10.1007/978-3-8348-2171-3. URL: http://dx.doi.org/10.1007/978-3-8348-2171-3.

[41] Lovell, M. C. "Data Mining". In: *The Review of Economics and Statistics* 65.1 (1983), p. 1. ISSN: 00346535. DOI: 10.2307/1924403.

[42] Wikipedia. *Machine learning - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Machine_learning (visited on 2017-07-07).

[43] Brownlee, J. *Supervised and Unsupervised Machine Learning Algorithms*. Vermont Victoria, Australia, 2016. URL: http://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/ (visited on 2017-07-07).

[44] Fingerscheidt, T. "Lecture Notes on Pattern Recognition". Technical University Brunswick, 2017.

[45] Bishop, C. M. *Pattern recognition and machine learning*. Corrected at 8. printing 2009. Information science and statistics. New York, NY: Springer, 2009. ISBN: 978-0-387-31073-2.

[46] Aksoy, S. *Introduction to Pattern Recognition*. 2016. URL: http://www.cs.bilkent.edu.tr/~saksoy/courses/cs551/slides/cs551_intro.pdf (visited on 2017-07-07).

[47] Martins, L. G. *Introduction to Pattern Recognition*. 2011. URL: https://www.slideshare.net/lgustavomartins/introduction-to-pattern-recognition (visited on 2017-07-07).

[48] Miclet, L. and Cornuéjols, A. *What is the place of Machine Learning between Pattern Recognition and Optimization: Presented at TML 2008 Conference: (Teaching Machine Learning)*. Saint Etienne, FRA, 2008. URL: https://pdfs.semanticscholar.org/fb76/c2b3230f09bc2ffc6a268373528749ad9a9a.pdf (visited on 2017-07-07).

[49] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern Classification*. 2. Aufl. s.l.: Wiley-Interscience, 2012. ISBN: 0-471-05669-3. URL: http://gbv.eblib.com/patron/FullRecord.aspx?p=699526.

[50] Wikipedia. *Pattern recognition - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Pattern_recognition (visited on 2017-07-07).

[51] Dokas, P., Ertoz, L., Kumar, V., et al. "Data Mining for Network Intrusion Detection". In: *Proceedings NSF Workshop on Next Generation Data Mining* (2002).

[52] Knorr, E. M. and Ng, R. T. "Algorithms for Mining Distance-Based Outliers in Large Datasets". In: *Proceedings of the 24rd International Conference on Very Large Data Bases*. Ed. by Morgan Kaufmann Publishers Inc. VLDB '98. San Francisco, CA, USA, 1998, pp. 392–403. ISBN: 1-55860-566-5. URL: http://www.vldb.org/conf/1998/p392.pdf (visited on 2017-07-10).

[53] Breunig, M. M., Kriegel, H.-P., Ng, R. T., et al. "LOF: Identifying Density-Based Local Outliers". In: *ACM SIGMOD Record* 29.2 (2000), pp. 93–104. ISSN: 01635808. DOI: 10.1145/335191.335388.

[54] Schölkopf, B., Platt, J. C., Shawe-Taylor, J., et al. "Estimating the support of a high-dimensional distribution". In: *Neural computation* 13.7 (2001), pp. 1443–1471. ISSN: 0899-7667. DOI: 10.1162/089976601750264965.

[55] He, Z., Xu, X., and Deng, S. "Discovering cluster-based local outliers". In: *Pattern Recognition Letters* 24.9-10 (2003), pp. 1641–1650. ISSN: 01678655. DOI: 10.1016/S0167-8655(03)00003-5.

[56] Campello, R. J. G. B., Moulavi, D., Zimek, A., et al. "Hierarchical Density Estimates for Data Clustering, Visualization, and Outlier Detection". In: *ACM Transactions on Knowledge Discovery from Data* 10.1 (2015), pp. 1–51. ISSN: 15564681. DOI: 10.1145/2733381.

[57] Wikipedia. *Anomaly detection - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Anomaly_detection (visited on 2017-07-10).

[58] Choudhary, P. *Introduction to Anomaly Detection*. 2017. URL: https://www.datascience.com/blog/intro-to-anomaly-detection-learn-data-science-tutorials (visited on 2017-07-10).

[59] Deckel Maho Pfronten GmbH. "User Manual DMU 80/100T2". Pfronten, 2006.

[60] DMG / MORI SEIKI Deutschland GmbH. *DMU 60 / 80 / 100 Monoblock Brochure*. 2011. URL: http://www.bewema.com/sites/default/files/DMU%2080T.pdf (visited on 2017-03-29).

[61] Deckel Maho Pfronten GmbH. "Circuit Diagram DMU 80/100T2". Pfronten, 2006.

[62] Gilgen, Müller & Weigert (GMW) GmbH & Co. KG. *GMW ASK 412.4 Current Transformer Documentation*. 2017. URL: http://www.g-mw.de/fileadmin/PDFs/Nieder-Mittelspannungs-Stromwandler/Stromwandler_DE.pdf (visited on 2017-04-12).

[63] Howard Butler LTD. *HOBUT Molded Case Current Transformers 13 Series Datasheet*. 2017. URL: http://docs-europe.electrocomponents.com/webdocs/151e/0900766b8151e916.pdf (visited on 2017-04-11).

[64] Beckhoff Automation GmbH & Co. KG. *Documentation EL3403 3-Phase power Measuring Terminal*. 2017. URL: https://download.beckhoff.com/download/document/io/ethercat-terminals/el3403de.pdf (visited on 2017-04-03).

[65] Beckhoff Automation GmbH & Co. KG. *Documentation EK110x, EK15xx EtherCAT-Coupler*. 2017. URL: https://download.beckhoff.com/download/document/io/ethercat-terminals/ek110x_ek15xxde.pdf (visited on 2017-04-03).

[66] Beckhoff Automation GmbH & Co. KG. *Documentation EL30xx Analog Input Terminal*. 2017. URL: https://download.beckhoff.com/download/document/io/ethercat-terminals/el30xxde.pdf (visited on 2017-04-03).

[67] Becker, A. *HeidiSQL SQL Client*. 2017. URL: https://www.heidisql.com/ (visited on 2017-04-10).

[68] Čihař, M., Jayaratne, M., Bennetch, I., et al. *phpMyAdmin SQL Webclient*. 2017. URL: https://www.phpmyadmin.net/ (visited on 2017-04-10).

[69] Ron, A. *Introduction to Numerical Analysis*. 2010. URL: http://pages.cs.wisc.edu/~amos/412/lecture-notes/lecture14.pdf (visited on 2017-05-15).

[70] Keogh, E. and Smyth, P. *A Probabilistic Approach to Fast Pattern Matching in Time Series Databases*. Ed. by AAAI Press. Newport Beach, CA, 1997. URL: http://www.aaai.org/Papers/KDD/1997/KDD97-004.pdf (visited on 2017-05-25).

[71] Frank, J., Mannor, S., Pineau, J., et al. "Time Series Analysis Using Geometric Template Matching". In: *IEEE transactions on pattern analysis and machine intelligence* 35.3 (2013), pp. 740–754. ISSN: 0098-5589. DOI: 10.1109/TPAMI.2012.121.

[72] Agrawal, R., Lin, K.-i., Sawhney, H. S., et al. "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases". In: *In VLDB* (1995), pp. 490–501. (Visited on 2017-05-25).

[73] Kocyan, T., Martinovic, J., Drazdilova, P., et al. *Searching Time Series Based On Pattern Extraction Using Dynamic Time Warping*. 2013. URL: http://ceur-ws.org/Vol-971/poster5.pdf (visited on 2017-05-25).

[74] Chen, Y., Nascimento, M. A., Ooi, B. C., et al. "SpADe: On Shape-based Pattern Detection in Streaming Time Series". In: *IEEE 23rd International Conference on Data Engineering, 2007*. Piscataway, NJ: IEEE Service Center, 2007, pp. 786–795. ISBN: 1-4244-0802-4. DOI: 10.1109/ICDE.2007.367924.

[75] Ge, X. and Smyth, P. "Deformable Markov model templates for time-series pattern matching". In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. Ed. by Ramakrishnan, R. New York, NY: ACM, 2000, pp. 81–90. ISBN: 1581132336. DOI: 10.1145/347090.347109.

[76] Agrawal, R., Faloutsos, C., and Swami, A. N. "Efficient Similarity Search In Sequence Databases". In: *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms* (1993), pp. 69–84. (Visited on 2017-05-25).

[77] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. "Fast subsequence matching in time-series databases". In: *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data - SIGMOD '94*. Ed. by Snodgrass, R. T. and Winslett, M. SIGMOD record. New York, NY: ACM Press, 1994, pp. 419–429. isbn: 0897916395. doi: 10.1145/191839.191925.

[78] Briechle, K. and Hanebeck, U. D. "Template matching using fast normalized cross correlation". In: *Proceedings of SPIE - The International Society for Optical Engineering 4387*. Ed. by Casasent, D. P. and Chao, T.-H. SPIE Proceedings. SPIE, 2001, pp. 95–102. doi: 10.1117/12.421129.

[79] Kleist, C. *Time Series Data Mining Methods: A Review*. 2015. url: http://edoc.hu-berlin.de/master/kleist-caroline-2015-03-25/PDF/kleist.pdf (visited on 2017-06-08).

[80] Esling, P. and Agon, C. "Time-series data mining". In: *ACM Comput. Surv.* 45.1 (2012), pp. 1–34. doi: 10.1145/2379776.2379788.

[81] Berndt, D. J. and Clifford, J. "Using Dynamic Time Warping to Find Patterns in Time Series". In: *AAAI Press* 1994 (1994), pp. 359–370.

[82] Meesrikamolkul, W., Niennattrakul, V., and Ratanamahatana, C. A. "Shape-Based Clustering for Time Series Data". In: *Advances in knowledge discovery and data mining*. Ed. by Tan, P.-N., Chawla, S., Ho, C. K., et al. Vol. 7301. Lecture notes in computer science Lecture notes in artificial intelligence. Berlin: Springer, 2012, pp. 530–541. isbn: 978-3-642-30216-9. doi: 10.1007/978-3-642-30217-6_44.

[83] Srisai, D. and Ratanamahatana, C. A. "Efficient Time Series Classification under Template Matching Using Time Warping Alignment". In: *2009 Fourth International Conference on Computer Sciences and Convergence Information Technology*, pp. 685–690. doi: 10.1109/ICCIT.2009.291.

[84] The MathWorks Inc. *MATLAB Signal Processing Toolbox Documentation*. Natik Massachusetts, 2017. url: https://de.mathworks.com/help/matlab/ (visited on 2017-04-19).

[85] Keogh, E. and Ratanamahatana, C. A. "Exact indexing of dynamic time warping". In: *Knowledge and Information Systems* 7.3 (2005), pp. 358–386. issn: 0219-1377. doi: 10.1007/s10115-004-0154-9.

[86] Niennattrakul, V., Ruengronghirunya, P., and Ratanamahatana, C. A. "Exact indexing for massive time series databases under time warping distance". In: *Data Mining and Knowledge Discovery* 21.3 (2010), pp. 509–541. issn: 1384-5810. doi: 10.1007/s10618-010-0165-y.

[87] Ratanamahatana, C. A. and Keogh, E. "Three Myths about Dynamic Time Warping Data Mining". In: *Proceedings of the 2005 SIAM International Conference on Data Mining*. Ed. by Kargupta, H., Srivastava, J., Kamath, C., et al. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2005, pp. 506–510. isbn: 978-0-89871-593-4. doi: 10.1137/1.9781611972757.50.

[88] Müller, M. *Information retrieval for music and motion*. Berlin u.a.: Springer, 2007. ISBN: 978-3-540-74047-6.

[89] Kordik, P. *Lecture Notes on Feature extraction from time series*. 2012. URL: https://edux.fit.cvut.cz/oppa/MI-PDD/prednasky/l8-signal-extraction.pdf (visited on 2017-06-08).

[90] Tarango, J., Keogh, E., and Brisk, P. "Accelerating the dynamic time warping distance measure using logarithmetic arithmetic". In: *2014 48th Asilomar Conference on Signals, Systems and Computers*, pp. 404–408. DOI: 10.1109/ACSSC.2014.7094472.

[91] The MathWorks Inc. *MATLAB Documentation*. Natik Massachusetts, 2017. URL: https://de.mathworks.com/help/matlab/ (visited on 2017-04-19).

[92] Eric W., W. *Cross-Correlation*. 2016. URL: http://mathworld.wolfram.com/Cross-Correlation.html (visited on 2017-06-08).

[93] Wikipedia. *Cross-Correlation - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Cross-correlation (visited on 2017-06-08).

[94] Wikipedia. *Autocorrelation - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Autocorrelation (visited on 2017-06-08).

[95] Wikipedia. *Pearson correlation coefficient - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Pearson_correlation_coefficient (visited on 2017-06-08).

[96] Wikipedia. *Multivariate random variable - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Multivariate_random_variable (visited on 2017-06-08).

[97] Brighton, H. and Mellish, C. "Advances in Instance Selection for Instance-Based Learning Algorithms". In: *Data Mining and Knowledge Discovery* 6.2 (2002), pp. 153–172. ISSN: 13845810. DOI: 10.1023/A:1014043630878.

[98] Wilson, D. R. and Martinez, T. R. "Reduction Techniques for Instance-Based Learning Algorithms". In: *Machine Learning* 38.3 (2000), pp. 257–286. ISSN: 08856125. DOI: 10.1023/A:1007626913721.

[99] Pękalska, E., Duin, R. P., and Paclík, P. "Prototype selection for dissimilarity-based classifiers". In: *Pattern Recognition* 39.2 (2006), pp. 189–208. ISSN: 00313203. DOI: 10.1016/j.patcog.2005.06.012.

[100] Boudaoud, S., Rix, H., and Meste, O. "Integral shape averaging and structural average estimation: A comparative study". In: *IEEE Transactions on Signal Processing* 53.10 (2005), pp. 3644–3650. ISSN: 1053-587X. DOI: 10.1109/TSP.2005.855106.

[101] Boudaoud, S., Rix, H., Meste, O., et al. "Corrected Integral Shape Averaging Applied to Obstructive Sleep Apnea Detection from the Electrocardiogram". In: *EURASIP Journal on Advances in Signal Processing* 2007.1 (2007), p. 032570. ISSN: 1687-6180. DOI: 10.1155/2007/32570.

[102] Gupta, L., Molfese, D. L., Tammana, R., et al. "Nonlinear alignment and averaging for estimating the evoked potential". In: *IEEE transactions on bio-medical engineering* 43.4 (1996), pp. 348–356. ISSN: 0018-9294. DOI: 10.1109/10.486255.

[103] Wang, K., Begleiter, H., and Porjesz, B. "Warp-averaging event-related potentials". In: *Clinical Neurophysiology* 112.10 (2001), pp. 1917–1924. issn: 13882457. doi: 10.1016/S1388-2457(01)00640-X.

[104] Petitjean, F. and Gançarski, P. "Summarizing a set of time series by averaging: From Steiner sequence to compact multiple alignment". In: *Theoretical Computer Science* 414.1 (2012), pp. 76–91. issn: 03043975. doi: 10.1016/j.tcs.2011.09.029.

[105] Soheily-Khah, S., Douzal-Chouakria, A., and Gaussier, E. *Progressive and Iterative Approaches for Time Series Averaging*. Porto, Portugal, 2015. url: https://hal.archives-ouvertes.fr/hal-01208451 (visited on 2017-05-23).

[106] Niennattrakul, V. and Ratanamahatana, C. A. "Shape averaging under Time Warping". In: *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pp. 626–629. doi: 10.1109/ECTICON.2009.5137128.

[107] Skalak, D. B. *Prototype and Feature Selection by Sampling and Random Mutation Hill Climbing Algorithms*. 1994. url: https://pdfs.semanticscholar.org/0df0/13671e9e901a9126deb4957e22e3d937b1a5.pdf (visited on 2017-05-23).

[108] Wilson, D. R. and Martinez, T. R. *Instance Pruning Techniques*. San Francisco, CA, USA, 1997. url: https://pdfs.semanticscholar.org/100d/35e18a98d1714a33b97ecc2df0c94847d9a8.pdf (visited on 2017-05-23).

[109] Xi, X., Keogh, E., Shelton, C., et al. "Fast time series classification using numerosity reduction". In: *Proceedings of the 23rd international conference on Machine learning*. Ed. by Cohen, W. New York, NY: ACM, 2006, pp. 1033–1040. isbn: 1595933832. doi: 10.1145/1143844.1143974.

[110] Niennattrakul, V., Srisai, D., and Ratanamahatana, C. A. "Shape-based template matching for time series data". In: *Knowledge-Based Systems* 26 (2012), pp. 1–8. issn: 09507051. doi: 10.1016/j.knosys.2011.04.015.

[111] Sathianwiriyakhun, P., Janyalikit, T., and Ratanamahatana, C. A. "Fast and accurate template averaging for time series classification". In: *The 2016 - 8th International Conference on Knowledge and Smart Technology (KST)*. Piscataway, NJ: IEEE, 2016, pp. 49–54. isbn: 978-1-4673-8137-6. doi: 10.1109/KST.2016.7440530.

[112] McKinley, S. and Levine, M. *Cubic Spline Interpolation*. 2009. url: https://web.archive.org/web/20090408054627/http://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall98/SkyMeg/Proj.PDF (visited on 2017-05-24).

[113] Thearling, K. *An Introduction to Data Mining*. 2006. url: http://www.thearling.com/ (visited on 2017-06-08).

[114] Domingos, P. "A few useful things to know about machine learning". In: *Communications of the ACM* 55.10 (2012), p. 78. issn: 00010782. doi: 10.1145/2347736.2347755.

[115] Demsar, J., Curk, T., Erjavec, A., et al. "Orange: Data Mining Tolbox in Python". In: *Journal of Machine Learning Research* 14 (2013), pp. 2349–2353.

[116] Demsar, J. and Zupan, B. *Orange: Data Mining Fruitful and Fun*. 2012. URL: http://ailab.ijs.si/dunja/TuringSLAIS-2012/Papers/Demsar_Orange.pdf (visited on 2017-06-16).

[117] Shaulsky, G., Borondics, F., and Bellazzi, R. *Documentation of Orange Visual Programming*. 2015. URL: https://docs.orange.biolab.si/3/visual-programming/index.html (visited on 2017-06-09).

[118] Brownlee, J. *A Tour of Machine Learning Algorithms*. Vermont Victoria, Australia, 2013. URL: http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (visited on 2017-06-15).

[119] Kramer, O. *Machine Learning for Evolution Strategies*. Vol. 20. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-33381-6. DOI: 10.1007/978-3-319-33383-0.

[120] Settles, B. H. *Lecture Notes on Feature Spaces*. 2003. URL: http://pages.cs.wisc.edu/~bsettles/cs540/lectures/16_feature_spaces.pdf (visited on 2017-06-08).

[121] Niemann, H. *Klassifikation von Mustern*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1983. ISBN: 978-3-540-12642-3. DOI: 10.1007/978-3-642-47517-7.

[122] Nilsson, N. J. *Introduction to Machine Learning*. Stanford, 1998. URL: https://ai.stanford.edu/~nilsson/MLBOOK.pdf (visited on 2017-06-23).

[123] Pedregosa, F., Varoquaux, G., Gramfort, A., et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830. (Visited on 2017-06-09).

[124] Breiman, L., Friedman, J. H., Olsen, R. A., et al. *Classification and Regression Trees*. New York, 1984.

[125] Kass, G. V. "Significance Testing in Automatic Interaction Detection (A.I.D.)" In: *Applied Statistics* 24.2 (1975), p. 178. ISSN: 00359254. DOI: 10.2307/2346565.

[126] Quinlan, J. R. "Induction of decision trees". In: *Machine Learning* 1.1 (1986), pp. 81–106. ISSN: 08856125. DOI: 10.1007/BF00116251.

[127] Wu, X., Kumar, V., Ross Quinlan, J., et al. "Top 10 algorithms in data mining". In: *Knowledge and Information Systems* 14.1 (2008), pp. 1–37. ISSN: 0219-1377. DOI: 10.1007/s10115-007-0114-2.

[128] Pandya, R. and Pandya, J. "C5. 0 Algorithm to Improved Decision Tree with Feature Selection and Reduced Error Pruning". In: *International Journal of Computer Applications* 117.16 (2015), pp. 18–21. ISSN: 09758887. DOI: 10.5120/20639-3318.

[129] Loh, W.-Y. "Classification and regression trees". In: *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 1.1 (2011), pp. 14–23. ISSN: 19424787. DOI: 10.1002/widm.8.

[130] Markham, K. *Comparing supervised learning algorithms*. 2015. URL: http://www.dataschool.io/comparing-supervised-learning-algorithms/ (visited on 2017-06-09).

[131] Ho, T. K. "Random decision forests". In: *Proceedings of the Third International Conference on Document Analysis and Recognition*. Los Alamitos, Calif.: IEEE Computer Society Press, 1995, pp. 278–282. ISBN: 0-8186-7128-9. DOI: 10.1109/ICDAR.1995.598994.

[132] Breiman, L. "Bagging Predictors". In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 08856125. DOI: 10.1023/A:1018054314350.

[133] Breiman, L. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 08856125. DOI: 10.1023/A:1010933404324.

[134] Ho, T. K. "The random subspace method for constructing decision forests". In: *IEEE transactions on pattern analysis and machine intelligence* 20.8 (1998), pp. 832–844. ISSN: 01628828. DOI: 10.1109/34.709601.

[135] Hastie, T., Tibshirani, R., and Friedman, J. *The elements of statistical learning: Data mining, inference, and prediction*. 10. [print.], (corr. as of 4. print.) Springer series in statistics. New York, NY: Springer, 2008. ISBN: 0-387-95284-5.

[136] Dietterich, T. G. "An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization". In: *Machine Learning* 40.2 (2000), pp. 139–157. ISSN: 08856125. DOI: 10.1023/A:1007607513941.

[137] Freund, Y. and Schapire, R. E. "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". In: *Journal of Computer and System Sciences* 55.1 (1997), pp. 119–139. ISSN: 00220000. DOI: 10.1006/jcss.1997.1504.

[138] Joglekar, S. *A small introduction to Boosting*. 2016. URL: https://codesachin.wordpress.com/tag/adaboost/ (visited on 2017-06-12).

[139] Iba, W. and Langley, P. "Induction of One-Level Decision Trees". In: *Proceedings of the Ninth International Workshop on Machine Learning*. Ed. by Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 1992, pp. 233–240. ISBN: 1-55860-247-X.

[140] Schapire, R. E. "Explaining AdaBoost". In: *Empirical Inference*. Ed. by Schölkopf, B., Luo, Z., and Vovk, V. Berlin, Heidelberg and s.l.: Springer Berlin Heidelberg, 2013, pp. 37–52. ISBN: 978-3-642-41135-9. DOI: 10.1007/978-3-642-41136-6_5.

[141] Schapire, R. E. "A Brief Introduction to Boosting". In: *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. Ed. by Morgan Kaufmann Publishers Inc. Vol. 2. San Francisco, CA, USA, 1999, pp. 1401–1406.

[142] McCormick, C. *AdaBoost Tutorial*. 2013. URL: http://mccormickml.com/2013/12/13/adaboost-tutorial/ (visited on 2017-06-12).

[143] Sochman, J. and Matas, J. *Lecture Notes on AdaBoost*. 2016. URL: http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf (visited on 2017-06-12).

[144] Kadeous, W. *What is AdaBoost?* 2014. URL: https://www.quora.com/What-is-AdaBoost/answer/Waleed-Kadous (visited on 2017-06-12).

[145] Kegl, B. *The return of AdaBoost.MH: multi-class Hamming trees*. 2013. URL: https://arxiv.org/pdf/1312.6086.pdf (visited on 2017-06-12).

[146] Cortes, C. and Vapnik, V. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. ISSN: 08856125. DOI: 10.1007/BF00994018.

[147] James, G. *An introduction to statistical learning: With applications in R*. corr. at 6. print. Springer texts in statistics. New York NY u.a.: Springer, 2015. ISBN: 978-1-4614-7138-7.

[148] Kowalczyk, A. *SVM Tutorial - Understanding the math - the optimal hyperplane*. 2014. URL: https://www.svm-tutorial.com/2015/06/svm-understanding-math-part-3/ (visited on 2017-06-13).

[149] Zielesny, A. *From Curve Fitting to Machine Learning*. Vol. 109. Cham: Springer International Publishing, 2016. ISBN: 978-3-319-32544-6. DOI: 10.1007/978-3-319-32545-3.

[150] Itseez. *The OpenCV Reference Manual - Introduction to SVM*. 2017. URL: http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html (visited on 2017-06-13).

[151] Snyman, J. A. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Vol. 97. Applied Optimization. Boston, MA: Springer Science+Business Media Inc, 2005. ISBN: 9780387243481. DOI: 10.1007/b105200. URL: http://site.ebrary.com/lib/alltitles/docDetail.action?docID=10133687.

[152] Bonnans, J. F., Gilbert, J. C., Lemaréchal, C., et al. *Numerical optimization: Theoretical and practical aspects*. Springer Berlin Heidelberg, 2006. ISBN: 978-3-540-35445-1. DOI: 10.1007/978-3-540-35447-5.

[153] Vapnyarskii, I. B. "Lagrange multipliers". In: *Encyclopaedia of mathematics*. Ed. by Hazewinkel, M. Berlin and New York: Springer-Verlag, 2002. ISBN: 9781402006098.

[154] Aizerman, M. A., Braverman, E. A., and Rozonoer, L. "Theoretical foundations of the potential function method in pattern recognition learning". In: *Automation and Remote Control*. Vol. 25. Automation and Remote Control. 1964, pp. 821–837.

[155] Mercer, J. "Functions of Positive and Negative Type, and their Connection with the Theory of Integral Equations". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 209.441-458 (1909), pp. 415–446. ISSN: 1364-503X. DOI: 10.1098/rsta.1909.0016.

[156] Powell, M. J. D. "Radial Basis Functions for Multivariable Interpolation: A Review". In: *Algorithms for Approximation*. Ed. by Mason, J. C. and Cox, M. G. New York, NY, USA: Clarendon Press, 1987, pp. 143–167. ISBN: 0-19-853612-7.

[157] Duan, K.-B. and Keerthi, S. S. "Which Is the Best Multiclass SVM Method? An Empirical Study". In: *Multiple Classifier Systems*. Ed. by Hutchison, D., Kanade, T., Kittler, J., et al. Vol. 3541. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 278–285. ISBN: 978-3-540-26306-7. DOI: 10.1007/11494683_28.

[158] Hsu, C.-W. and Lin, C.-J. "A comparison of methods for multiclass support vector machines". In: *IEEE transactions on neural networks* 13.2 (2002), pp. 415–425. ISSN: 1045-9227. DOI: 10.1109/72.991427.

[159] Zhang, T. "Solving large scale linear prediction problems using stochastic gradient descent algorithms". In: *Proceedings of the 41st annual Design Automation Conference*. Ed. by Brodley, C. New York, NY: ACM, 2004, p. 116. isbn: 1-58113-828-8. doi: 10.1145/1015330.1015332.

[160] Wikipedia. *Stochastic gradient descent - Wikipedia: The Free Ecyclopedia*. 2017. url: https://en.wikipedia.org/wiki/Stochastic_gradient_descent (visited on 2017-06-14).

[161] Kiwiel, K. C. "Convergence and efficiency of subgradient methods for quasiconvex minimization". In: *Mathematical Programming* 90.1 (2001), pp. 1–25. issn: 0025-5610. doi: 10.1007/PL00011414.

[162] Bottou, L. *Online Algorithms and Stochastic Approximations*. Ed. by Saad, D. Cambridge, UK, 1998. url: https://pdfs.semanticscholar.org/e861/7e0970802fce2506b703877b5459c72611c7.pdf (visited on 2017-06-14).

[163] Brownlee, J. *Gradient Descent For Machine Learning*. 2016. url: http://machinelearningmastery.com/gradient-descent-for-machine-learning/ (visited on 2017-06-14).

[164] Bayes and Price. "An Essay towards Solving a Problem in the Doctrine of Chances. By the Late Rev. Mr. Bayes, F. R. S. Communicated by Mr. Price, in a Letter to John Canton, A. M. F. R. S". In: *Philosophical Transactions of the Royal Society of London* 53.0 (1763), pp. 370–418. issn: 0261-0523. doi: 10.1098/rstl.1763.0053.

[165] Rish, I. *An empirical study of the naive bayes classifier*. Yorktown Heights, NY, 2001. url: http://www.research.ibm.com/people/r/rish/papers/RC22230.pdf (visited on 2017-06-14).

[166] Rennie, J. D. M., Shih, L., Teevan, J., et al. "Tackling the Poor Assumptions of Naive Bayes Text Classifiers". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 616–623. isbn: 1-57735-189-4.

[167] Barber, D. *Bayesian Reasoning and Machine Learning*. 2017. url: http://web4.cs.ucl.ac.uk/staff/D.Barber/textbook/090310.pdf (visited on 2017-06-23).

[168] Ray, S. *6 Easy Steps to Learn Naive Bayes Algorithm (with code in Python)*. 2015. url: https://www.analyticsvidhya.com/blog/2015/09/naive-bayes-explained/ (visited on 2017-06-15).

[169] Fernandez-Temprano, M., Gardel-Sotomayor, P. E., Duque-Perez, O., et al. "Broken bar condition monitoring of an induction motor under different supplies using a linear discriminant analysis". In: *9th IEEE International Symposium on Diagnostics for Electric Machines, Power Electronics and Drives (SDEMPED), 2013*. Piscataway, NJ: IEEE, 2013, pp. 162–168. isbn: 978-1-4799-0025-1. doi: 10.1109/DEMPED.2013.6645712.

[170] Jie, S., Hong, G. S., Rahman, M., et al. "Feature Extraction and Selection in Tool Condition Monitoring System". In: *AI 2002: Advances in Artificial Intelligence*. Ed. by McKay, B. and Slaney, J. Vol. 2557. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, 2002, pp. 487–497. isbn: 978-3-540-00197-3. doi: 10.1007/3-540-36187-1_43.

[171] Zhou, J.-H., Pang, C. K., Lewis, F. L., et al. "Intelligent Diagnosis and Prognosis of Tool Wear Using Dominant Feature Identification". In: *IEEE Transactions on Industrial Informatics* 5.4 (2009), pp. 454–464. issn: 1551-3203. doi: 10.1109/TII.2009.2023318.

[172] Zhou, J.-H., Pang, C. K., Zhong, Z.-W., et al. "Tool Wear Monitoring Using Acoustic Emissions by Dominant-Feature Identification". In: *IEEE Transactions on Instrumentation and Measurement* 60.2 (2011), pp. 547–559. issn: 0018-9456. doi: 10.1109/TIM.2010.2050974.

[173] Zhang, C., Yao, X., Zhang, J., et al. "Tool Condition Monitoring and Remaining Useful Life Prognostic Based on a Wireless Sensor in Dry Milling Operations". In: *Sensors (Basel, Switzerland)* 16.6 (2016). issn: 1424-8220. doi: 10.3390/s16060795.

[174] Wikipedia. *Orange (software) - Wikipedia: The Free Ecyclopedia*. 2017. url: https://en.wikipedia.org/wiki/Orange_(software) (visited on 2017-06-16).

[175] Unpingco, J. *Python for Probability, Statistics, and Machine Learning*. Cham: Springer International Publishing, 2016. isbn: 978-3-319-30715-2. doi: 10.1007/978-3-319-30717-6.

[176] Spruyt, V. *The Curse of Dimensionality in Classification*. 2014. url: http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/ (visited on 2017-06-19).

[177] Jebara, T. *Machine Learning*. Boston, MA: Springer US, 2004. isbn: 978-1-4613-4756-9. doi: 10.1007/978-1-4419-9011-2.

[178] Shapiro, L. *Lecture Notes on Information Gain*. 2010. url: https://courses.cs.washington.edu/courses/cse455/10au/notes/InfoGain.pdf (visited on 2017-06-19).

[179] Amro. *What is entropy and information gain? - Stackoverflow*. 2016. url: https://stackoverflow.com/a/1859910 (visited on 2017-06-19).

[180] Bergstra, J., Bardenet, R., Bengio, Y., et al. "Algorithms for Hyper-parameter Optimization". In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. Ed. by Curran Associates Inc. NIPS'11. Granada, Spain, 2011, pp. 2546–2554. isbn: 978-1-61839-599-3. url: http://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf (visited on 2017-06-22).

[181] Bergstra, J. and Bengio, Y. "Random Search for Hyper-parameter Optimization". In: *Journal of Machine Learning Research* 13 (2012), pp. 281–305. url: http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf (visited on 2017-06-22).

[182] Sokolova, M. and Lapalme, G. "A systematic analysis of performance measures for classification tasks". In: *Information Processing & Management* 45.4 (2009), pp. 427–437. issn: 03064573. doi: 10.1016/j.ipm.2009.03.002.

[183] Ganesan, K. *Computing Precision and Recall for Multi-Class Classification Problems*. 2014. URL: http://text-analytics101.rxnlp.com/2014/10/computing-precision-and-recall-for.html (visited on 2017-06-22).

[184] Fawcett, T. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874. ISSN: 01678655. DOI: 10.1016/j.patrec.2005.10.010.

[185] Powers, D. M. W. "Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation". In: *Journal of Machine Learning Technologies*. Vol. 2. 2011, pp. 37–63. URL: https://csem.flinders.edu.au/research/techreps/SIE07001.pdf (visited on 2017-06-22).

[186] Chandola, V., Banerjee, A., and Kumar, V. "Anomaly detection: A Survey". In: *ACM Comput. Surv.* 41.3 (2009), pp. 1–58. DOI: 10.1145/1541880.1541882.

[187] Hodge, V. J. and Austin, J. "A Survey of Outlier Detection Methodologies". In: *Artificial Intelligence Review* 22.2 (2004), pp. 85–126. ISSN: 0269-2821. DOI: 10.1007/s10462-004-4304-y.

[188] Wikipedia. *Local outlier factor - Wikipedia: The Free Ecyclopedia*. 2017. URL: https://en.wikipedia.org/wiki/Local_outlier_factor (visited on 2017-07-03).

[189] Chen, L. *Lecture Notes on Density-Based Outlier Detection*. 2016. URL: http://www.cse.ust.hk/~leichen/courses/comp5331/lectures/LOF_Example.pdf (visited on 2017-07-03).

[190] Shahram. *My first attempt with Local Outlier Factor(LOF): Identifying Density Based Local Outliers*. 2015. URL: http://shahramabyari.com/2015/12/30/my-first-attempt-with-local-outlier-factorlof-identifying-density-based-local-outliers/ (visited on 2017-07-03).

[191] TU München. *Declaration of Authorship*. 2017. URL: https://www.ent.wi.tum.de/fileadmin/w00bcx/www/Ehrenwoertliche_Erklaerung_deutsch_und_englisch.pdf (visited on 2017-05-11).

A Appendix

A.1 Full Machine State Definition Table

Table A.1: Full set of possible machine states.

Controllable											**Non-Controllable**			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	2
0	0	1	0	0	0	0	0	0	0	0	0	0	0	3
0	0	0	1	0	0	0	0	0	0	0	0	0	0	4
0	0	0	0	1	0	0	0	0	0	0	0	0	0	5
0	0	0	0	0	1	0	0	0	0	0	0	0	0	6
0	0	0	0	0	0	1	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	1	0	0	0	0	0	0	8
0	0	0	0	0	0	0	0	1	0	0	0	0	0	9
0	0	0	0	0	0	0	0	0	1	0	0	0	0	10
0	0	0	0	0	0	0	0	0	0	1	0	0	0	11
0	0	0	0	0	0	0	0	0	0	0	0	0	1	12
1	0	0	0	0	0	0	0	0	0	0	0	0	1	13
0	1	0	0	0	0	0	0	0	0	0	0	0	1	14
0	0	1	0	0	0	0	0	0	0	0	0	0	1	15
0	0	0	1	0	0	0	0	0	0	0	0	0	1	16
0	0	0	0	1	0	0	0	0	0	0	0	0	1	17
0	0	0	0	0	1	0	0	0	0	0	0	0	1	18
0	0	0	0	0	0	1	0	0	0	0	0	0	1	19
0	0	0	0	0	0	0	1	0	0	0	0	0	1	20
0	0	0	0	0	0	0	0	1	0	0	0	0	1	21
0	0	0	0	0	0	0	0	0	1	0	0	0	1	22
0	0	0	0	0	0	0	0	0	0	1	0	0	1	23

Table A.1 (continuation) : Full set of possible machine states.

Controllable											Non-Controllable			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
0	0	0	0	0	0	0	0	0	0	0	0	1	0	24
1	0	0	0	0	0	0	0	0	0	0	0	1	0	25
0	1	0	0	0	0	0	0	0	0	0	0	1	0	26
0	0	1	0	0	0	0	0	0	0	0	0	1	0	27
0	0	0	1	0	0	0	0	0	0	0	0	1	0	28
0	0	0	0	1	0	0	0	0	0	0	0	1	0	29
0	0	0	0	0	1	0	0	0	0	0	0	1	0	30
0	0	0	0	0	0	1	0	0	0	0	0	1	0	31
0	0	0	0	0	0	0	1	0	0	0	0	1	0	32
0	0	0	0	0	0	0	0	1	0	0	0	1	0	33
0	0	0	0	0	0	0	0	0	1	0	0	1	0	34
0	0	0	0	0	0	0	0	0	0	1	0	1	0	35
0	0	0	0	0	0	0	0	0	0	0	0	1	1	36
1	0	0	0	0	0	0	0	0	0	0	0	1	1	37
0	1	0	0	0	0	0	0	0	0	0	0	1	1	38
0	0	1	0	0	0	0	0	0	0	0	0	1	1	39
0	0	0	1	0	0	0	0	0	0	0	0	1	1	40
0	0	0	0	1	0	0	0	0	0	0	0	1	1	41
0	0	0	0	0	1	0	0	0	0	0	0	1	1	42
0	0	0	0	0	0	1	0	0	0	0	0	1	1	43
0	0	0	0	0	0	0	1	0	0	0	0	1	1	44
0	0	0	0	0	0	0	0	1	0	0	0	1	1	45
0	0	0	0	0	0	0	0	0	1	0	0	1	1	46
0	0	0	0	0	0	0	0	0	0	1	0	1	1	47

Table A.1 (continuation) : Full set of possible machine states.

Controllable											Non-Controllable			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
0	0	0	0	0	0	0	0	0	0	0	1	0	0	48
1	0	0	0	0	0	0	0	0	0	0	1	0	0	49
0	1	0	0	0	0	0	0	0	0	0	1	0	0	50
0	0	1	0	0	0	0	0	0	0	0	1	0	0	51
0	0	0	1	0	0	0	0	0	0	0	1	0	0	52
0	0	0	0	1	0	0	0	0	0	0	1	0	0	53
0	0	0	0	0	1	0	0	0	0	0	1	0	0	54
0	0	0	0	0	0	1	0	0	0	0	1	0	0	55
0	0	0	0	0	0	0	1	0	0	0	1	0	0	56
0	0	0	0	0	0	0	0	1	0	0	1	0	0	57
0	0	0	0	0	0	0	0	0	1	0	1	0	0	58
0	0	0	0	0	0	0	0	0	0	1	1	0	0	59
0	0	0	0	0	0	0	0	0	0	0	1	0	1	60
1	0	0	0	0	0	0	0	0	0	0	1	0	1	61
0	1	0	0	0	0	0	0	0	0	0	1	0	1	62
0	0	1	0	0	0	0	0	0	0	0	1	0	1	63
0	0	0	1	0	0	0	0	0	0	0	1	0	1	64
0	0	0	0	1	0	0	0	0	0	0	1	0	1	65
0	0	0	0	0	1	0	0	0	0	0	1	0	1	66
0	0	0	0	0	0	1	0	0	0	0	1	0	1	67
0	0	0	0	0	0	0	1	0	0	0	1	0	1	68
0	0	0	0	0	0	0	0	1	0	0	1	0	1	69
0	0	0	0	0	0	0	0	0	1	0	1	0	1	70
0	0	0	0	0	0	0	0	0	0	1	1	0	1	71

Table A.1 (continuation) : Full set of possible machine states.

Controllable											Non-Controllable			
Main Spindle	X-Axis forward	X-Axis backward	Y-Axis forward	Y-Axis backward	Z-Axis forward	Z-Axis backward	C-Axis forward	C-Axis backward	B-Axis forward	B-Axis backward	Machine Lubrication Pump	Oil-Air Lubrication Pump	Control Cabinet Heat Exchanger	**Machine State**
0	0	0	0	0	0	0	0	0	0	0	1	1	0	72
1	0	0	0	0	0	0	0	0	0	0	1	1	0	73
0	1	0	0	0	0	0	0	0	0	0	1	1	0	74
0	0	1	0	0	0	0	0	0	0	0	1	1	0	75
0	0	0	1	0	0	0	0	0	0	0	1	1	0	76
0	0	0	0	1	0	0	0	0	0	0	1	1	0	77
0	0	0	0	0	1	0	0	0	0	0	1	1	0	78
0	0	0	0	0	0	1	0	0	0	0	1	1	0	79
0	0	0	0	0	0	0	1	0	0	0	1	1	0	80
0	0	0	0	0	0	0	0	1	0	0	1	1	0	81
0	0	0	0	0	0	0	0	0	1	0	1	1	0	82
0	0	0	0	0	0	0	0	0	0	1	1	1	0	83
0	0	0	0	0	0	0	0	0	0	0	1	1	1	84
1	0	0	0	0	0	0	0	0	0	0	1	1	1	85
0	1	0	0	0	0	0	0	0	0	0	1	1	1	86
0	0	1	0	0	0	0	0	0	0	0	1	1	1	87
0	0	0	1	0	0	0	0	0	0	0	1	1	1	88
0	0	0	0	1	0	0	0	0	0	0	1	1	1	89
0	0	0	0	0	1	0	0	0	0	0	1	1	1	90
0	0	0	0	0	0	1	0	0	0	0	1	1	1	91
0	0	0	0	0	0	0	1	0	0	0	1	1	1	92
0	0	0	0	0	0	0	0	1	0	0	1	1	1	93
0	0	0	0	0	0	0	0	0	1	0	1	1	1	94
0	0	0	0	0	0	0	0	0	0	1	1	1	1	95

A.2 Test Cycle Code for Heidenhain iTNC 530

Listing A.1: Program code of the test cycle for the DMU 100 Monoblock® without activation of additional components.

```
0  BEGIN PGM Testzyklus3 MM
1  BLK FORM 0.1 Z  X+0  Y+0  Z+0
2  BLK FORM 0.2  X+0  Y+0  Z+0
3  TOOL CALL 70 Z S10000 F12000
4  L  Z-1 F AUTO M91
5  L X+0 Y+0
6  M47
7  CALL LBL "Gesamt"
8  CALL LBL "Gesamt"
9  CALL LBL "Gesamt"
10 CALL LBL "Gesamt"
11 CALL LBL "Gesamt"
12 CALL LBL "Gesamt"
13 CALL LBL "Gesamt"
14 CALL LBL "Gesamt"
15 CALL LBL "Gesamt"
16 CALL LBL "Gesamt"
17 CALL LBL "Gesamt"
18 CALL LBL "Gesamt"
19 CALL LBL "Gesamt"
20 CALL LBL "Gesamt"
21 CALL LBL "Gesamt"
22 CALL LBL "Gesamt"
23 CALL LBL "Gesamt"
24 CALL LBL "Gesamt"
25 CALL LBL "Gesamt"
26 CALL LBL "Gesamt"
27 M48
28 M30
29 LBL "Gesamt"
30 CYCL DEF 9.0 VERWEILZEIT
31 CYCL DEF 9.1 V.ZEIT10
32 CALL LBL "spindle"
33 CALL LBL "spindle"
34 CALL LBL "spindle"
35 CALL LBL "spindle"
36 CALL LBL "spindle"
37 CALL LBL "xaxis"
38 CALL LBL "xaxis"
39 CALL LBL "xaxis"
40 CALL LBL "xaxis"
41 CALL LBL "xaxis"
42 CALL LBL "yaxis"
43 CALL LBL "yaxis"
44 CALL LBL "yaxis"
45 CALL LBL "yaxis"
46 CALL LBL "yaxis"
47 CALL LBL "zaxis"
```

```
48 CALL LBL "zaxis"
49 CALL LBL "zaxis"
50 CALL LBL "zaxis"
51 CALL LBL "zaxis"
52 CALL LBL "caxis"
53 CALL LBL "caxis"
54 CALL LBL "caxis"
55 CALL LBL "caxis"
56 CALL LBL "caxis"
57 CALL LBL "baxis"
58 CALL LBL "baxis"
59 CALL LBL "baxis"
60 CALL LBL "baxis"
61 CALL LBL "baxis"
62 LBL "w1"
63 CYCL DEF 9.0 VERWEILZEIT
64 CYCL DEF 9.1 V.ZEIT2
65 LBL 0
66 LBL "w2"
67 CYCL DEF 9.0 VERWEILZEIT
68 CYCL DEF 9.1 V.ZEIT20
69 LBL 0
70 LBL "spindle"
71 CALL LBL "w1"
72 M3
73 CALL LBL "w2"
74 M5
75 LBL 0
76 LBL "xaxis"
77 CALL LBL "w1"
78 L  X+900 R0 F AUTO
79 CALL LBL "w1"
80 L  X+0 F AUTO
81 LBL 0
82 LBL "yaxis"
83 CALL LBL "w1"
84 L  Y-600 F AUTO
85 CALL LBL "w1"
86 L  Y+0 F AUTO
87 LBL 0
88 LBL "zaxis"
89 CALL LBL "w1"
90 L  Z+250 F AUTO
91 CALL LBL "w1"
92 L  Z-1 F AUTO M91
93 LBL 0
94 LBL "caxis"
95 CALL LBL "w1"
96 L  C+180 F AUTO
97 CALL LBL "w1"
98 L  C+0 F AUTO
99 LBL 0
100 LBL "baxis"
101 CALL LBL "w1"
```

```
102 L  B-120 F AUTO
103 CALL LBL "w1"
104 L  B+0 F AUTO
105 LBL 0
106 END PGM Testzyklus3 MM
```

Listing A.2: Program code of the test cycle for the DMU 100 Monoblock® with activation of additional components like the Fume Separator Motor, both Coolant Pumps and the Chip Conveyor Drive Motor.

```
0  BEGIN PGM Testzyklus3 MM
1  BLK FORM 0.1 Z  X+0  Y+0  Z+0
2  BLK FORM 0.2  X+0  Y+0  Z+0
3  TOOL CALL 70 Z S10000 F12000
4  L  Z-1 F AUTO M91
5  L X+0 Y+0
6  M47
7  CALL LBL "Gesamt"
8  CALL LBL "Gesamt"
9  CALL LBL "Gesamt"
10 CALL LBL "Gesamt"
11 CALL LBL "Gesamt"
12 CALL LBL "Gesamt"
13 CALL LBL "Gesamt"
14 CALL LBL "Gesamt"
15 CALL LBL "Gesamt"
16 CALL LBL "Gesamt"
17 CALL LBL "Gesamt"
18 CALL LBL "Gesamt"
19 CALL LBL "Gesamt"
20 CALL LBL "Gesamt"
21 CALL LBL "Gesamt"
22 CALL LBL "Gesamt"
23 CALL LBL "Gesamt"
24 CALL LBL "Gesamt"
25 CALL LBL "Gesamt"
26 CALL LBL "Gesamt"
27 M8
28 CALL LBL "w2"
29 M9
30 CALL LBL "w1"
31 M7
32 CALL LBL "w2"
33 M9
34 CALL LBL "w1"
35 M324
36 CALL LBL "w2"
37 M325
38 CALL LBL "w1"
39 M70
40 CALL LBL "w2"
41 M71
42 CALL LBL "w1"
```

```
43 M48
44 M30
45 LBL "Gesamt"
46 CYCL DEF 9.0 VERWEILZEIT
47 CYCL DEF 9.1 V.ZEIT10
48 CALL LBL "spindle"
49 CALL LBL "spindle"
50 CALL LBL "spindle"
51 CALL LBL "spindle"
52 CALL LBL "spindle"
53 CALL LBL "xaxis"
54 CALL LBL "xaxis"
55 CALL LBL "xaxis"
56 CALL LBL "xaxis"
57 CALL LBL "xaxis"
58 CALL LBL "yaxis"
59 CALL LBL "yaxis"
60 CALL LBL "yaxis"
61 CALL LBL "yaxis"
62 CALL LBL "yaxis"
63 CALL LBL "zaxis"
64 CALL LBL "zaxis"
65 CALL LBL "zaxis"
66 CALL LBL "zaxis"
67 CALL LBL "zaxis"
68 CALL LBL "caxis"
69 CALL LBL "caxis"
70 CALL LBL "caxis"
71 CALL LBL "caxis"
72 CALL LBL "caxis"
73 CALL LBL "baxis"
74 CALL LBL "baxis"
75 CALL LBL "baxis"
76 CALL LBL "baxis"
77 CALL LBL "baxis"
78 LBL "w1"
79 CYCL DEF 9.0 VERWEILZEIT
80 CYCL DEF 9.1 V.ZEIT2
81 LBL 0
82 LBL "w2"
83 CYCL DEF 9.0 VERWEILZEIT
84 CYCL DEF 9.1 V.ZEIT20
85 LBL 0
86 LBL "spindle"
87 CALL LBL "w1"
88 M3
89 CALL LBL "w2"
90 M5
91 LBL 0
92 LBL "xaxis"
93 CALL LBL "w1"
94 L  X+900 R0 F AUTO
95 CALL LBL "w1"
96 L  X+0 F AUTO
```

```
97 LBL 0
98 LBL "yaxis"
99 CALL LBL "w1"
100 L  Y-600 F AUTO
101 CALL LBL "w1"
102 L  Y+0 F AUTO
103 LBL 0
104 LBL "zaxis"
105 CALL LBL "w1"
106 L  Z+250 F AUTO
107 CALL LBL "w1"
108 L  Z-1 F AUTO M91
109 LBL 0
110 LBL "caxis"
111 CALL LBL "w1"
112 L  C+180 F AUTO
113 CALL LBL "w1"
114 L  C+0 F AUTO
115 LBL 0
116 LBL "baxis"
117 CALL LBL "w1"
118 L  B-120 F AUTO
119 CALL LBL "w1"
120 L  B+0 F AUTO
121 LBL 0
122 END PGM Testzyklus3 MM
```

A.3 Results of Test Cycle Runs 1 and 2

A.3.1 Test Cycle Run 1

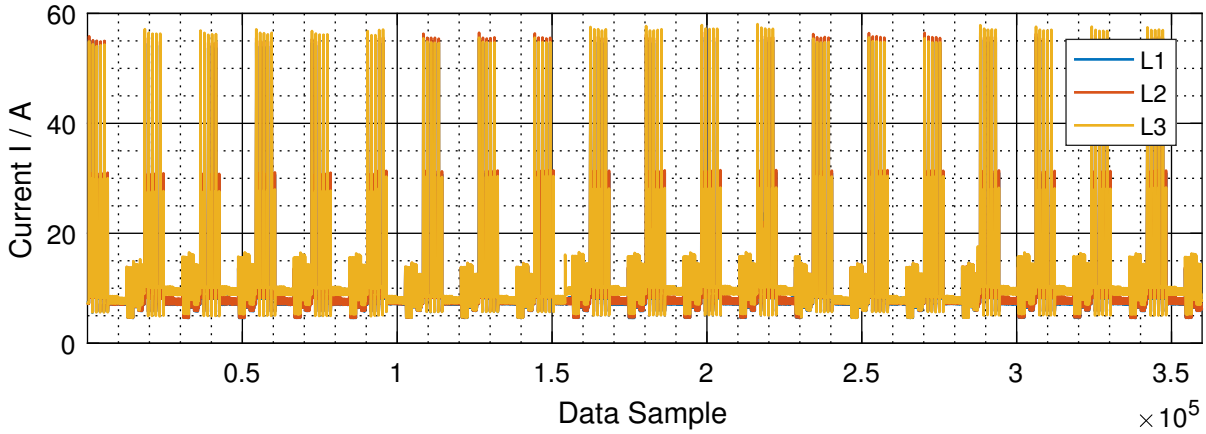
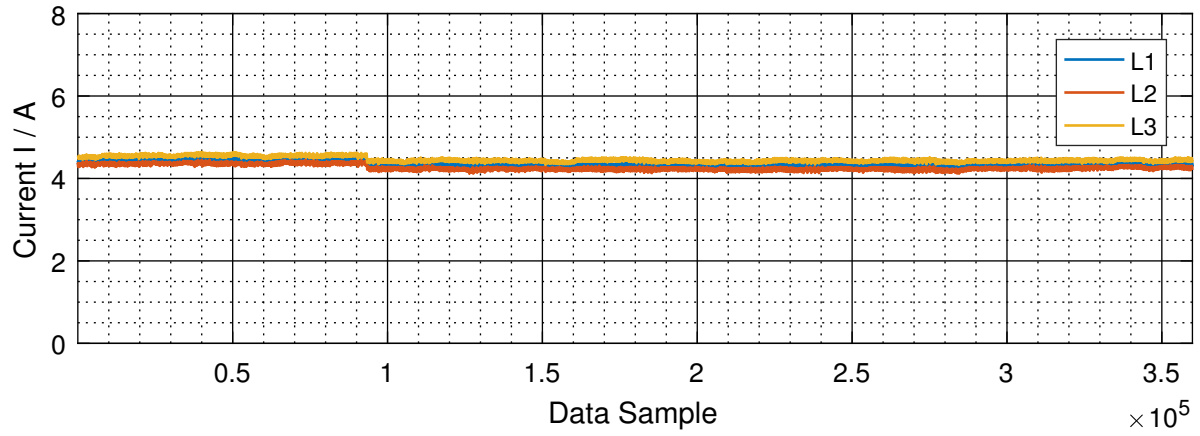
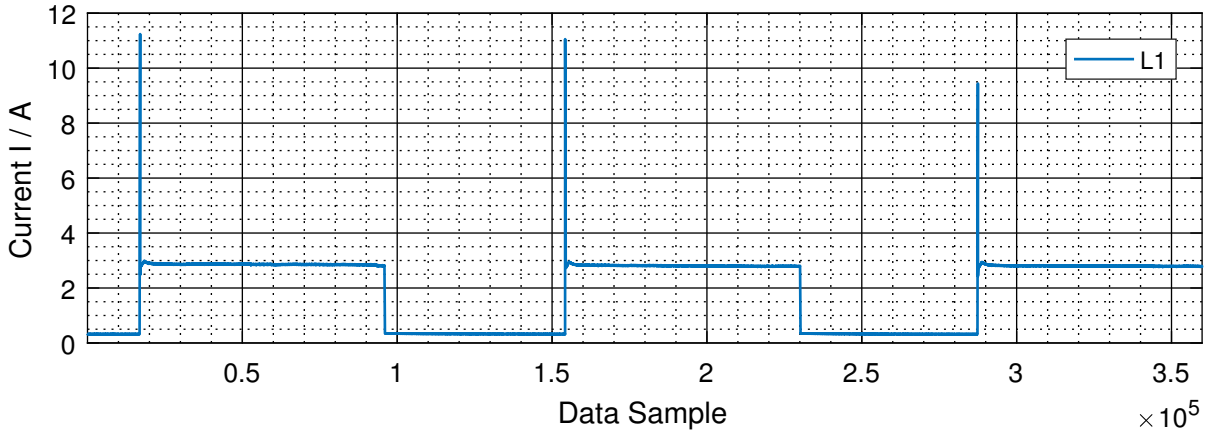


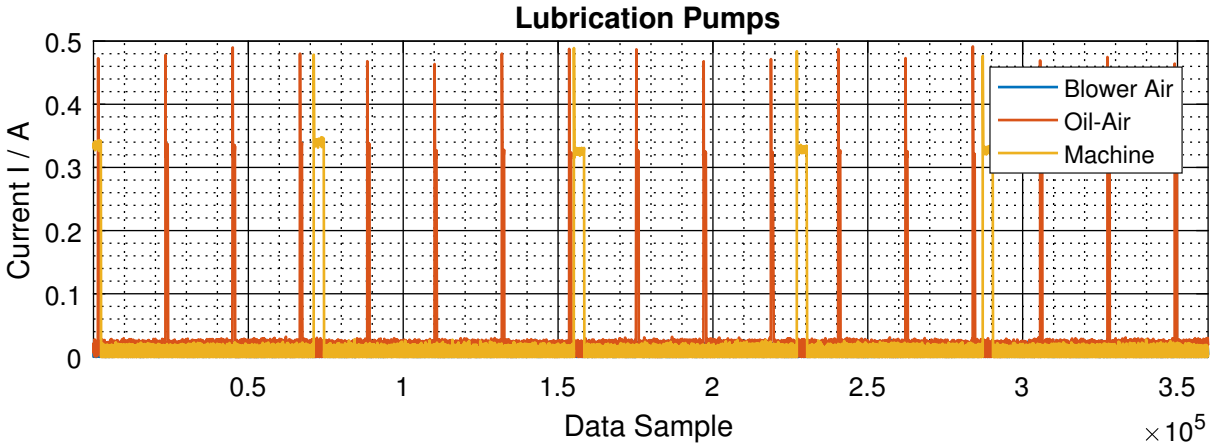
Figure A.1: Measured signal of the current flow within the three phases of the DMU main power line during the complete first test cycle run.



(a) Current consumption of the cooling unit.



(b) Current consumption of the control cabinet heat exchanger.



(c) Current consumption of the blower air lubraction pump (L1), the oil-air lubrication pump (L2) and the machine lubrication pump (L3).

Figure A.2: Measured current consumptions of the different machine components during the first test cycle run.

A.3.2 Test Cycle Run 2

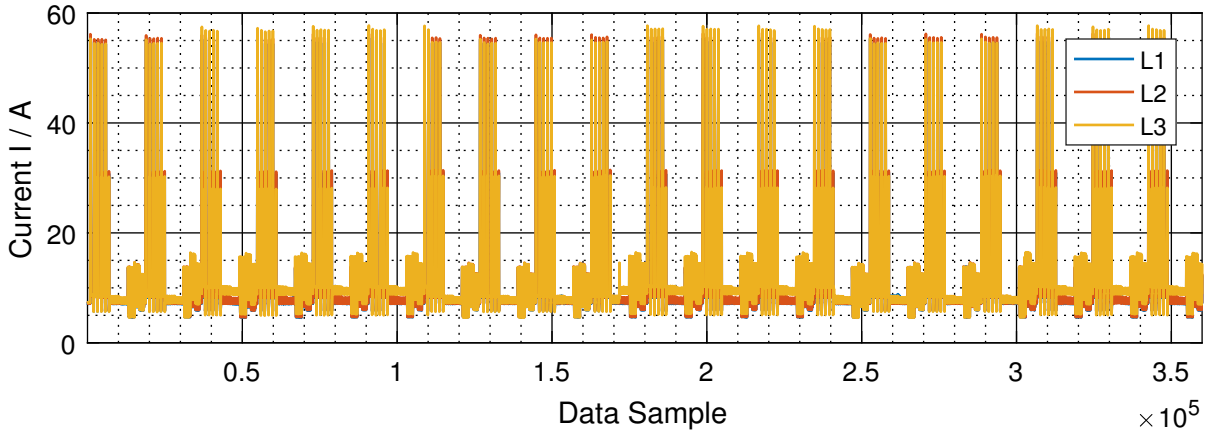
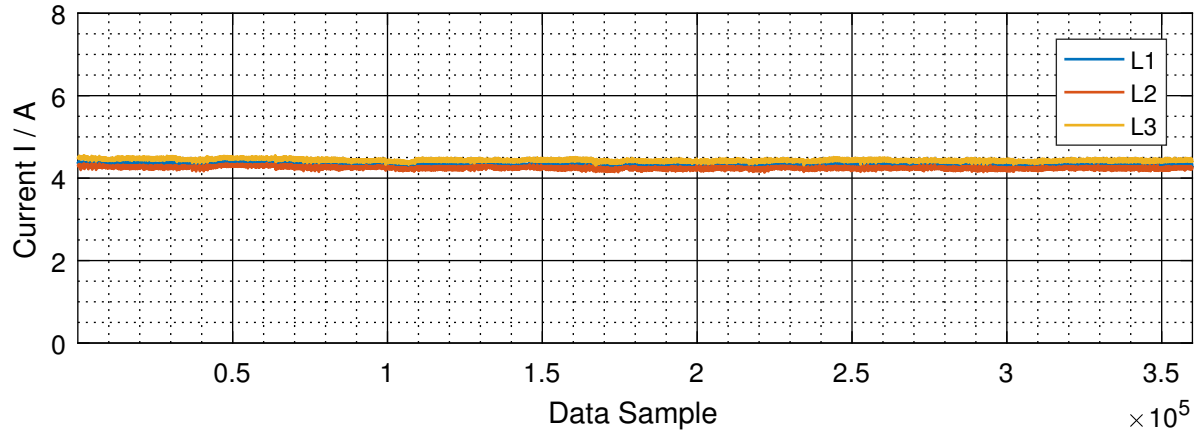
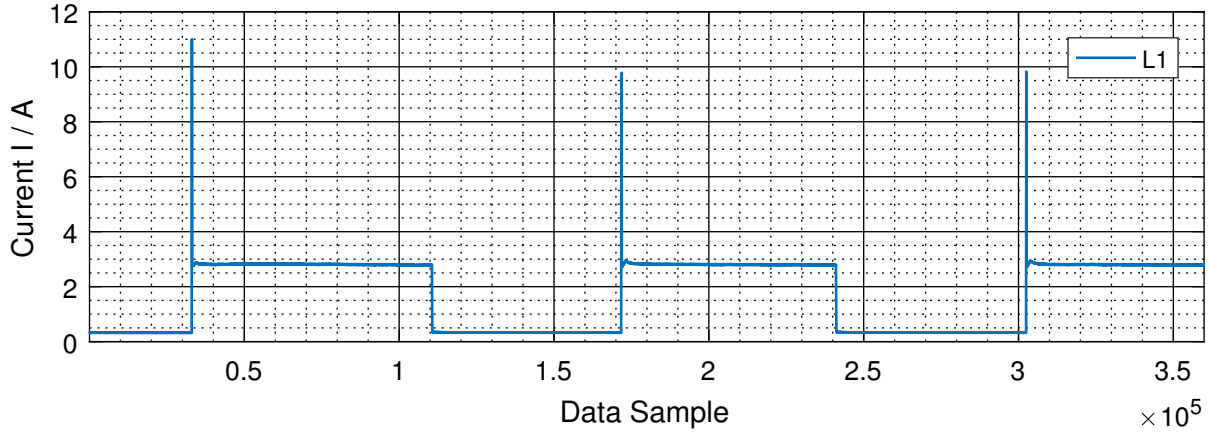


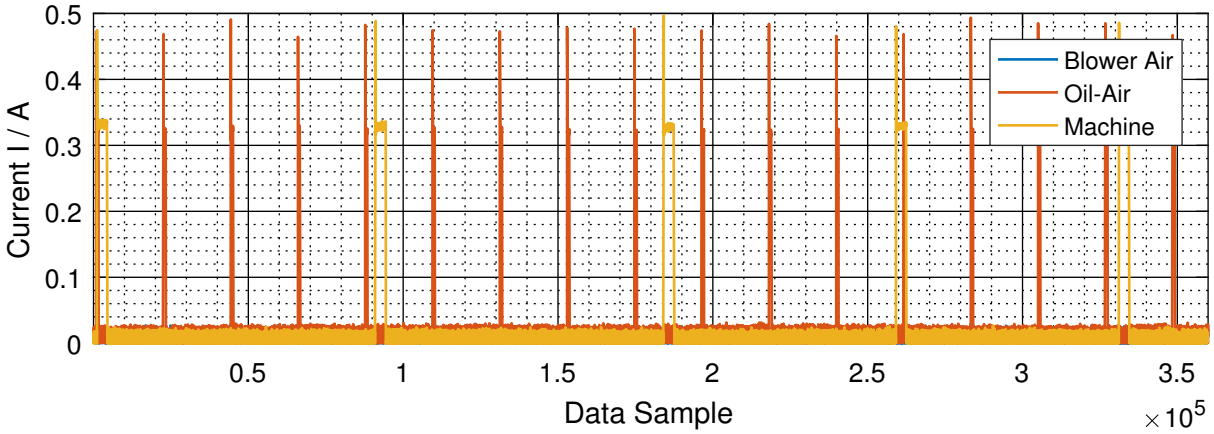
Figure A.3: Measured signal of the current flow within the three phases of the DMU main power line during the complete second test cycle run.



(a) Current consumption of the cooling unit.



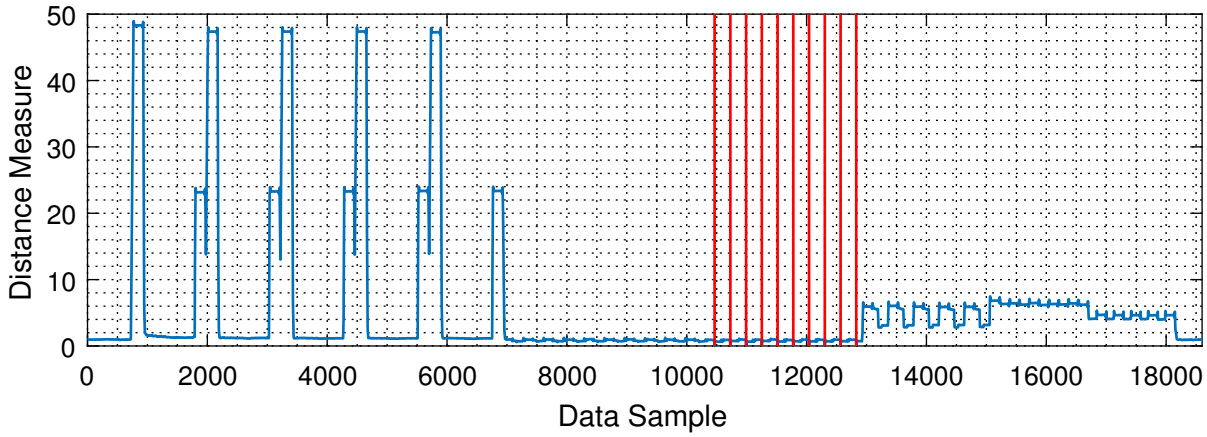
(b) Current consumption of the control cabinet heat exchanger.



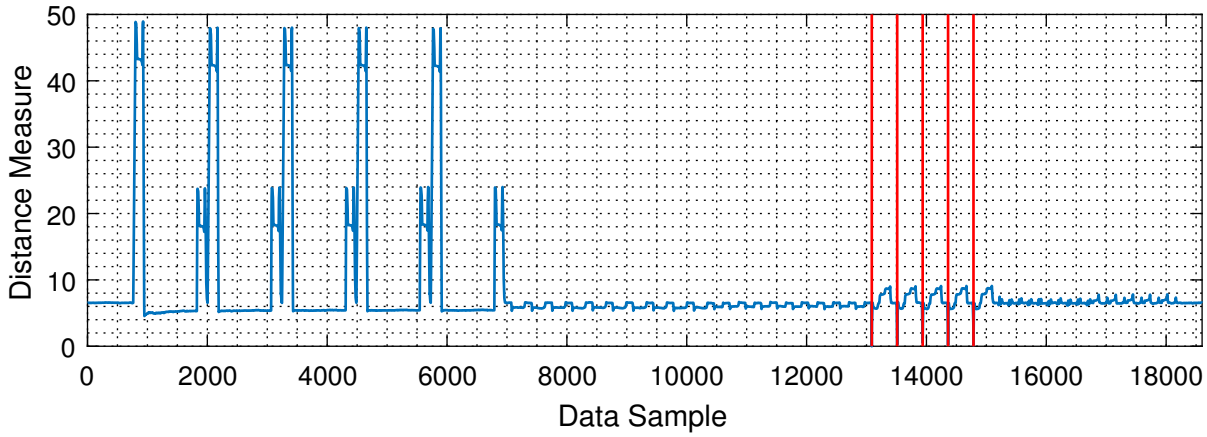
(c) Current consumption of the blower air lubraction pump (L1), the oil-air lubrication pump (L2) and the machine lubrication pump (L3).

Figure A.4: Measured current consumptions of the different machine components during the second test cycle run.

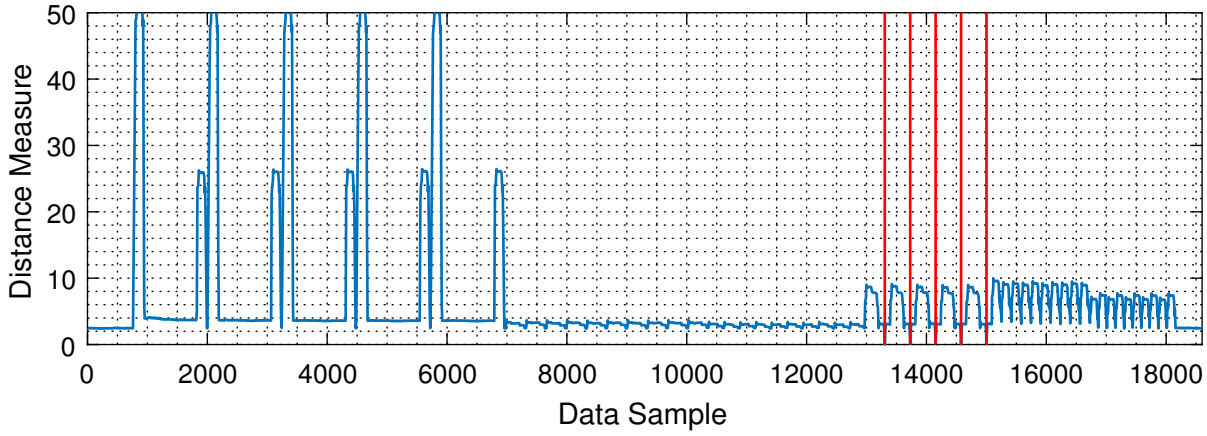
A.4 Distance Measures of Templates for Automatic Segmentation



(a) Distance measure of the Y-axis forward and backward templates.

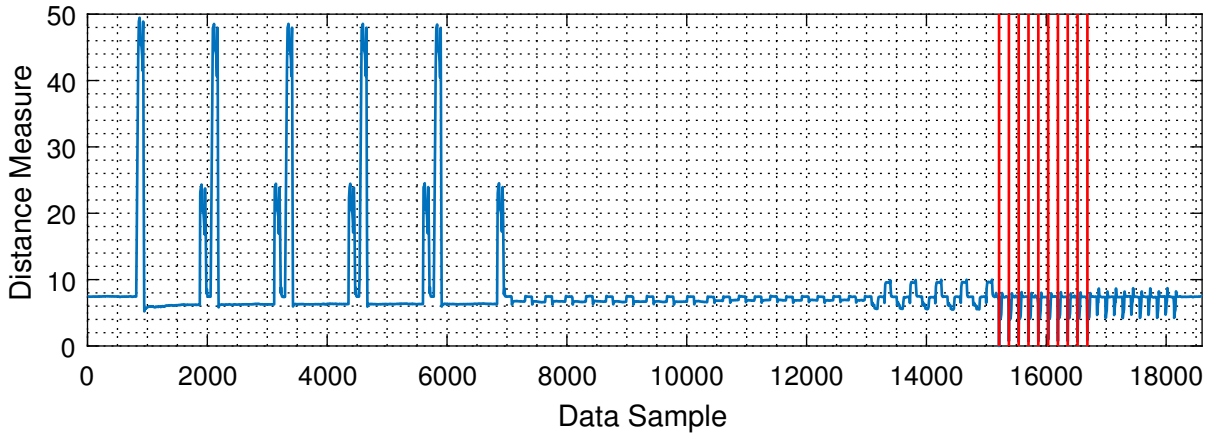


(b) Distance measure of the Z-axis forward template.

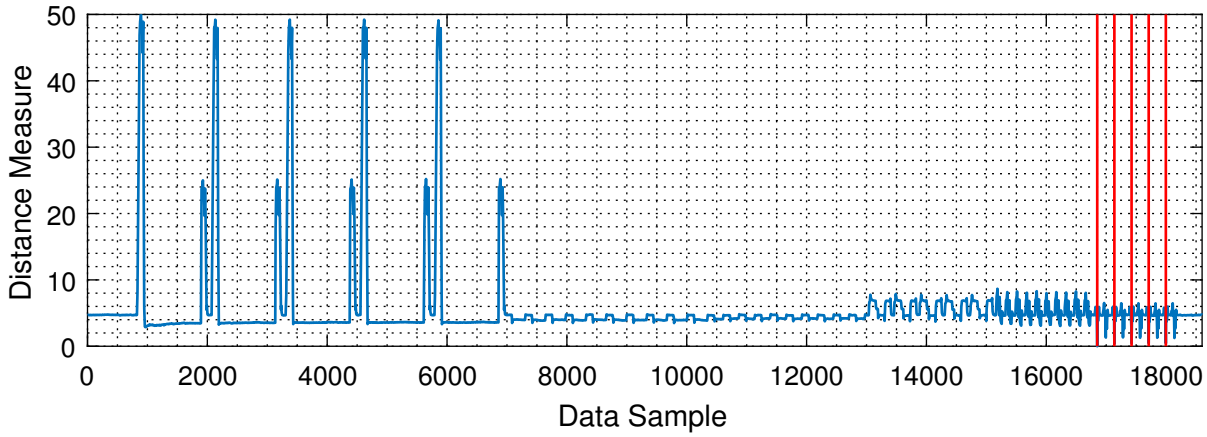


(c) Distance measure of the Z-axis backward template.

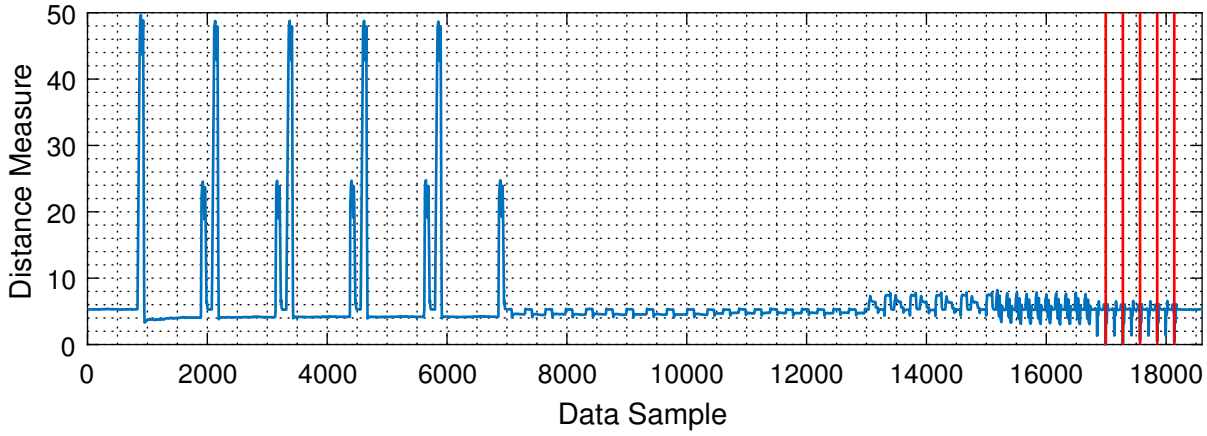
Figure A.5: Distance measures of different patterns as function of shifting lag expressed in data samples of measurement data.



(a) Distance measure of the C-axis forward and backward templates.



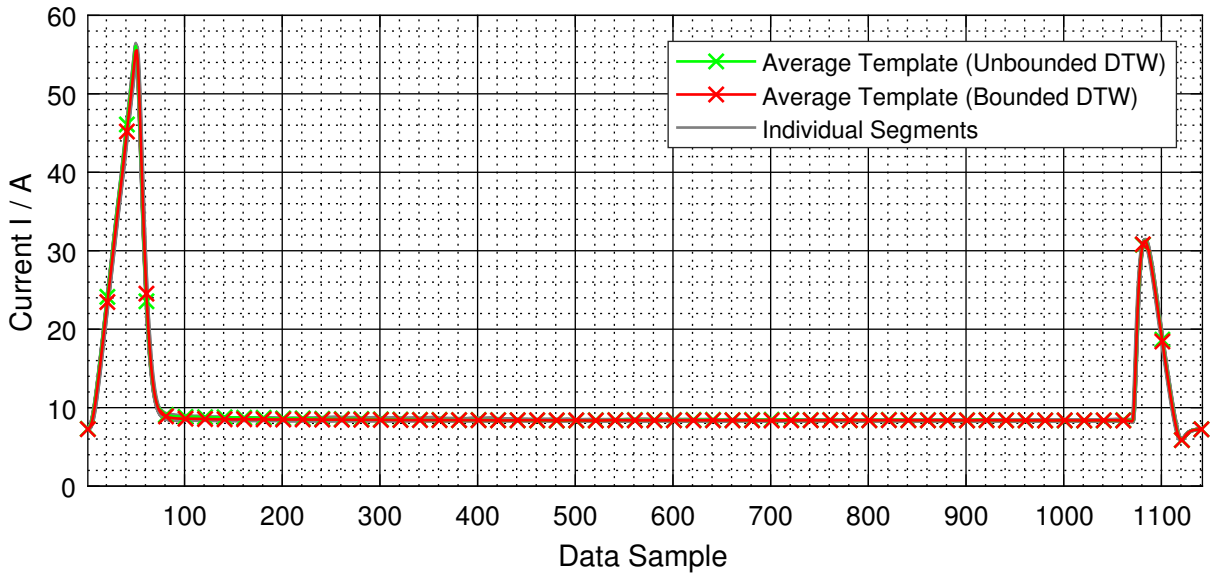
(b) Distance measure of the B-axis forward template.



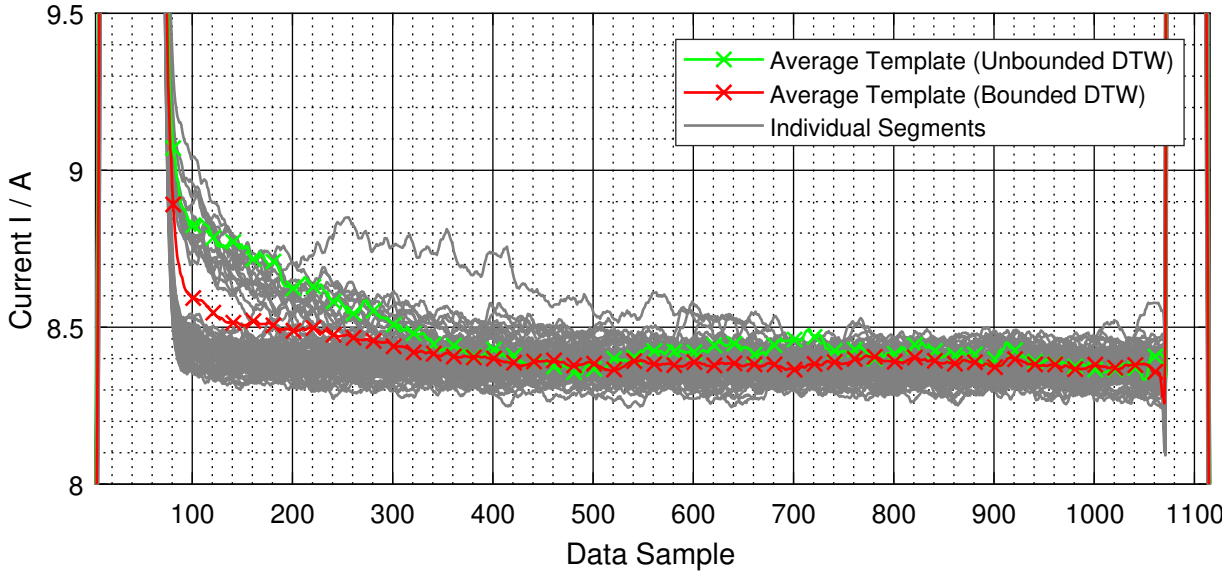
(c) Distance measure of the B-axis backward template.

Figure A.5 (continuation) : Distance measures of different patterns as function of shifting lag expressed in data samples of measurement data.

A.5 Resulting Averaged Templates from Accurate Shape Averaging

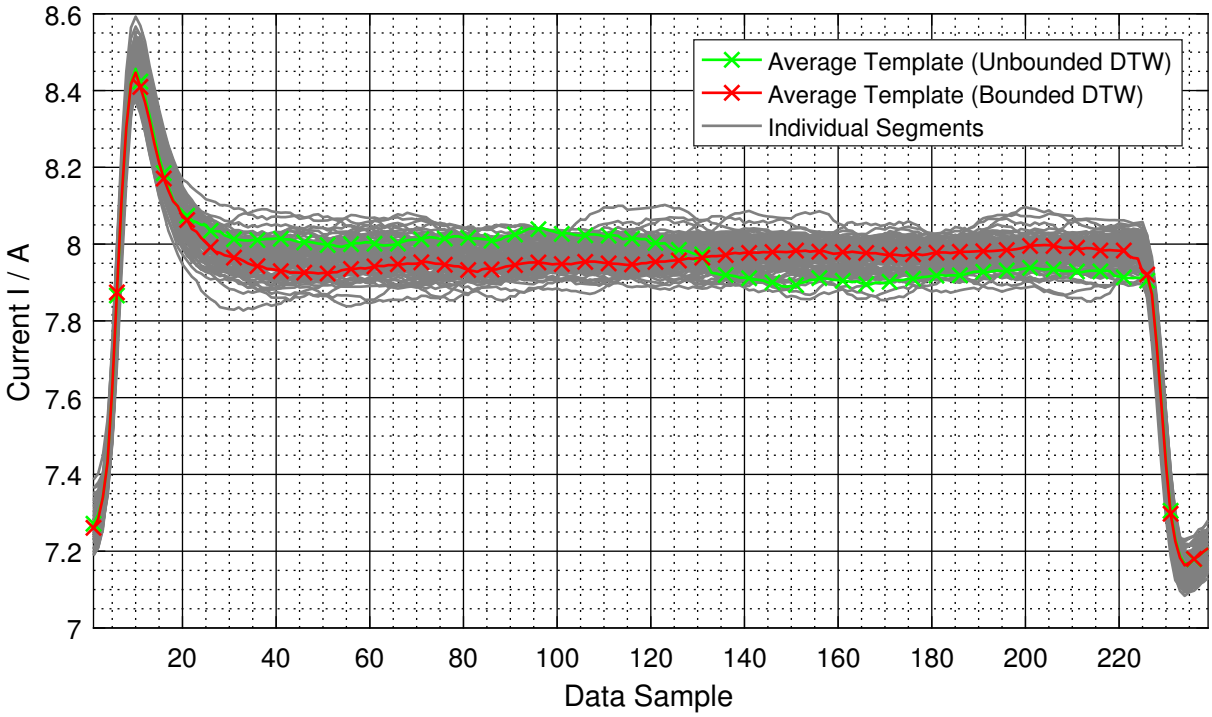


(a) Averaged template for class 1.

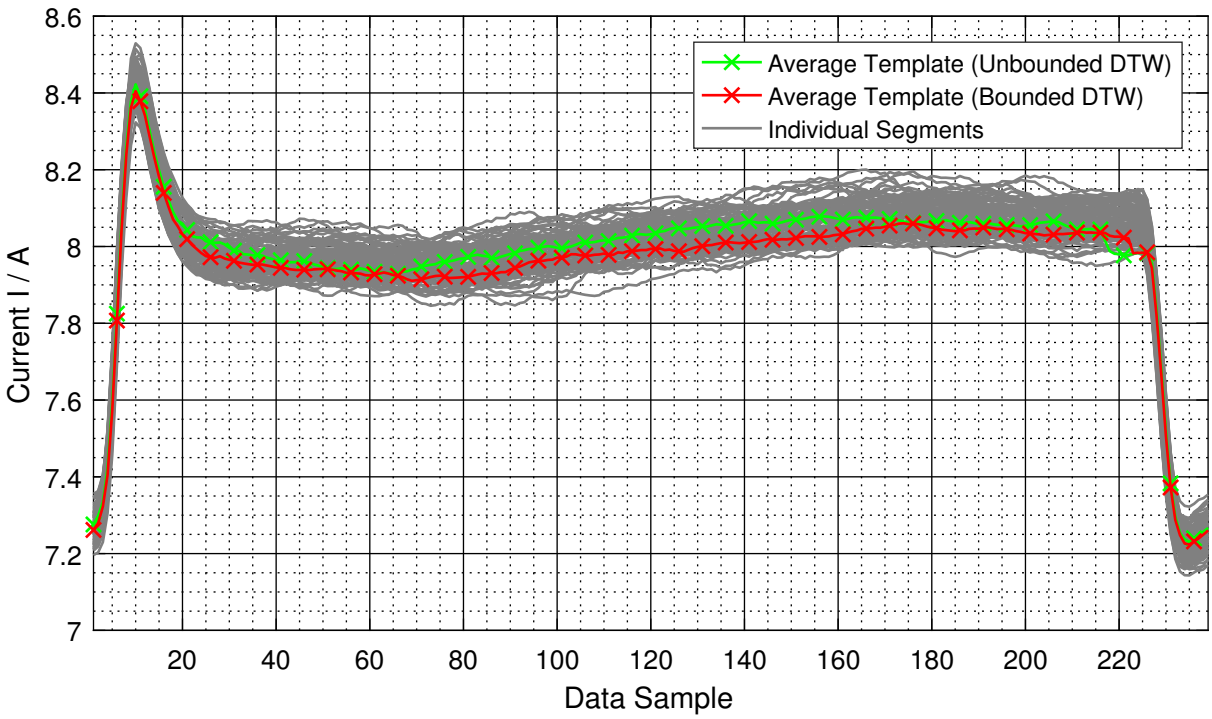


(b) Close up of averaged template for class 1.

Figure A.6: Resulting averaged template of the accurate shape averaging algorithm for sequences of class 1, in which the main spindle is activated. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.

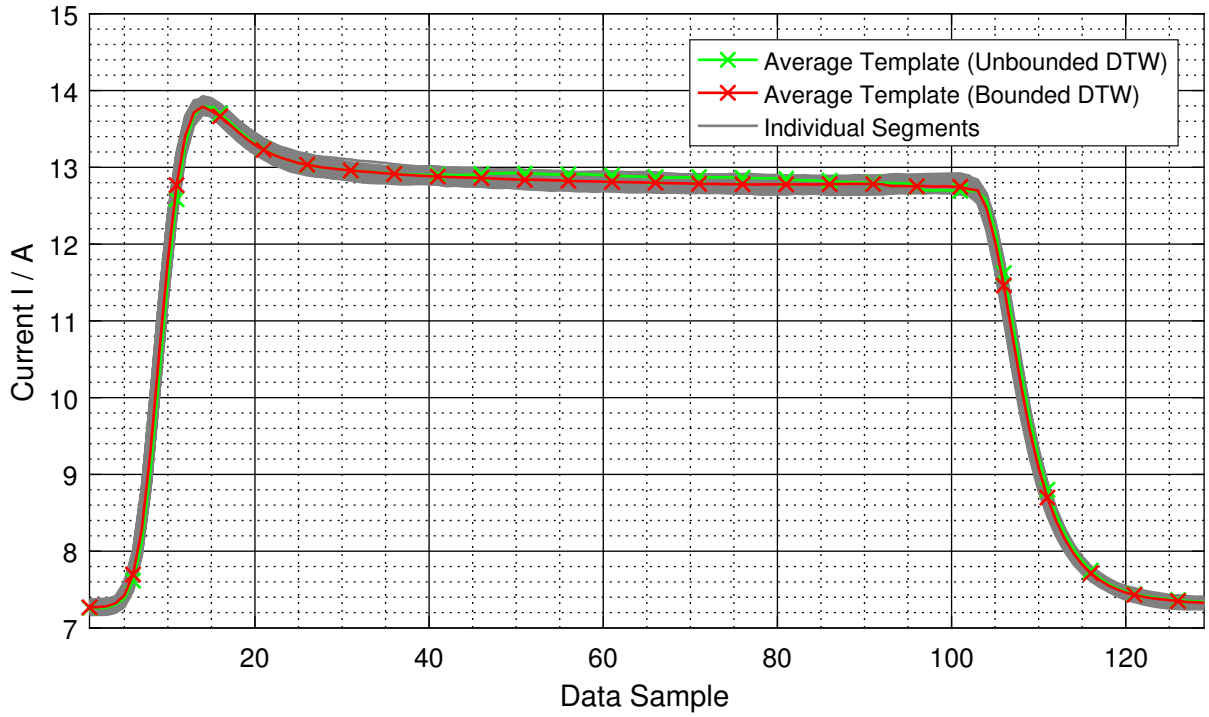


(a) Averaged template for class 2, the X-axis forward movement.

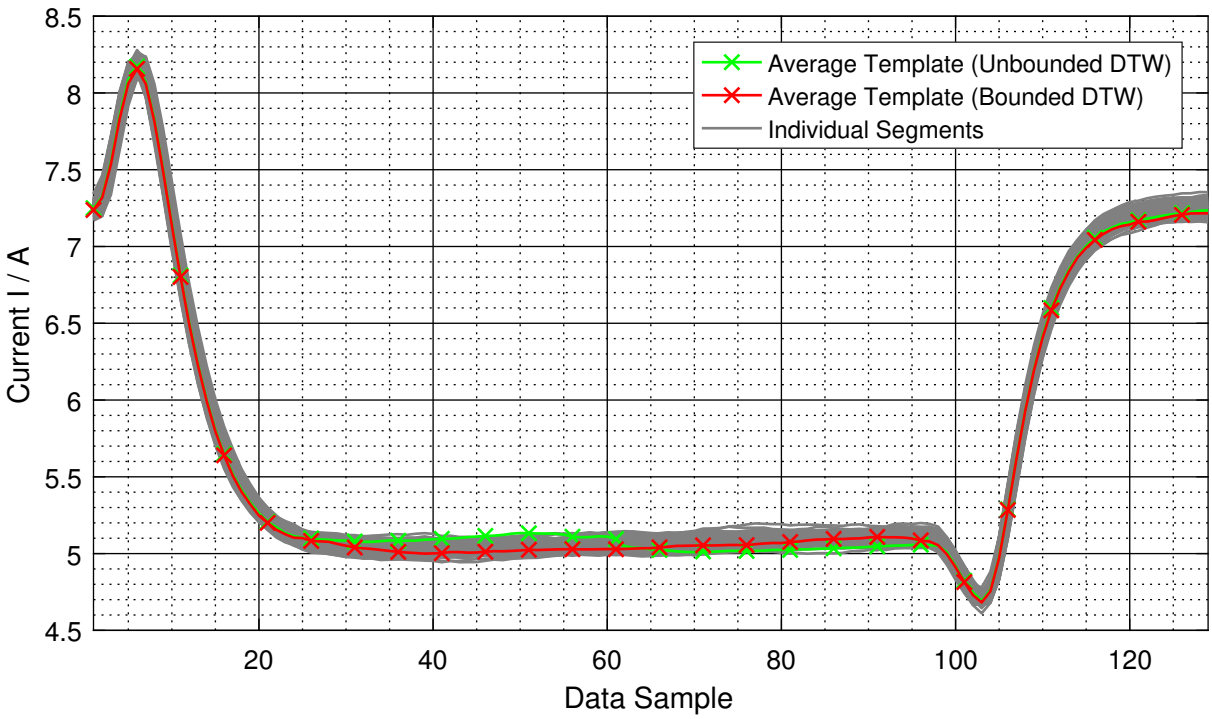


(b) Averaged template for class 3, the X-axis backward movement.

Figure A.7: Resulting averaged templates of the accurate shape averaging algorithm for sequences of classes 2 and 3, in which the X-axis moves forward and backward. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.

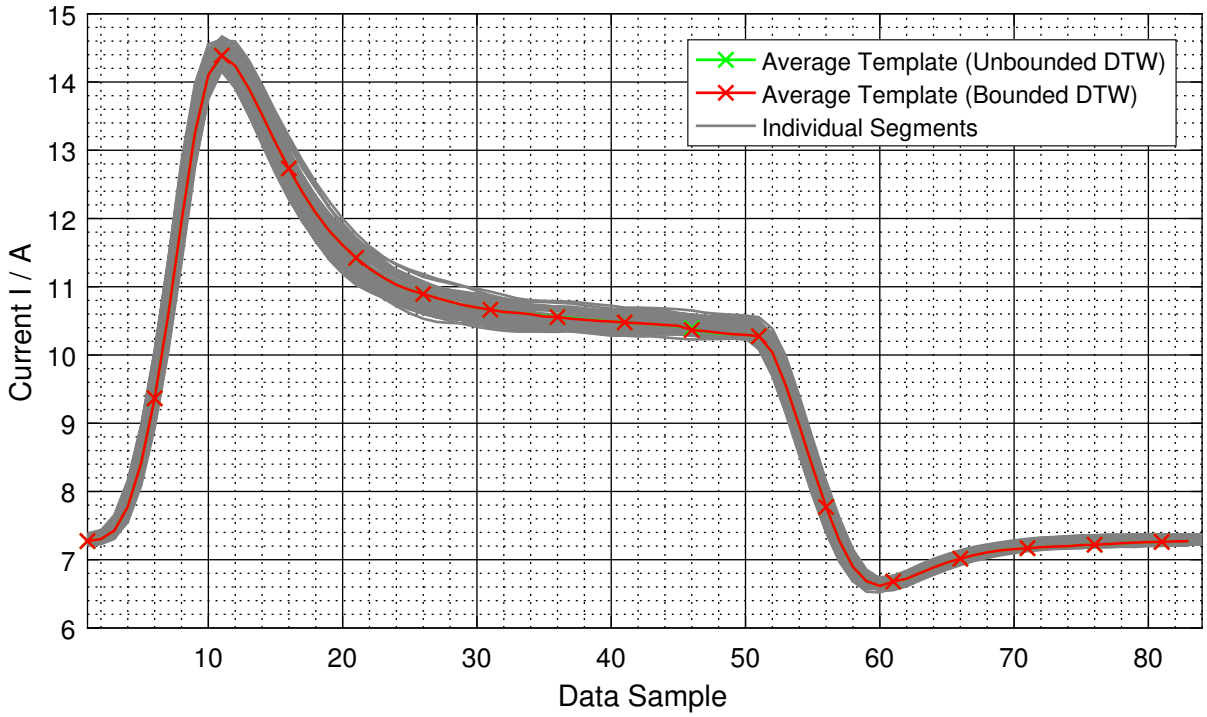


(a) Averaged template for class 6, the Z-axis forward movement.

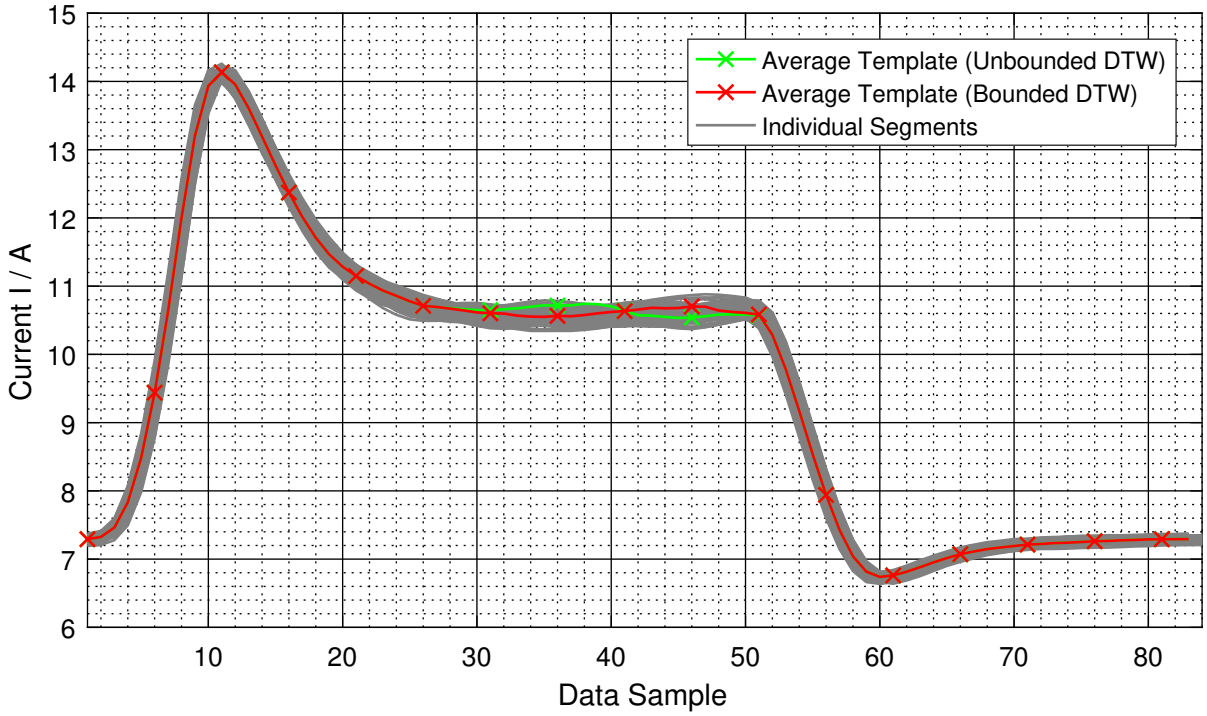


(b) Averaged template for class 7, the Z-axis backward movement.

Figure A.8: Resulting averaged templates of the accurate shape averaging algorithm for sequences of classes 6 and 7, in which the Z-axis moves forward and backward. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.

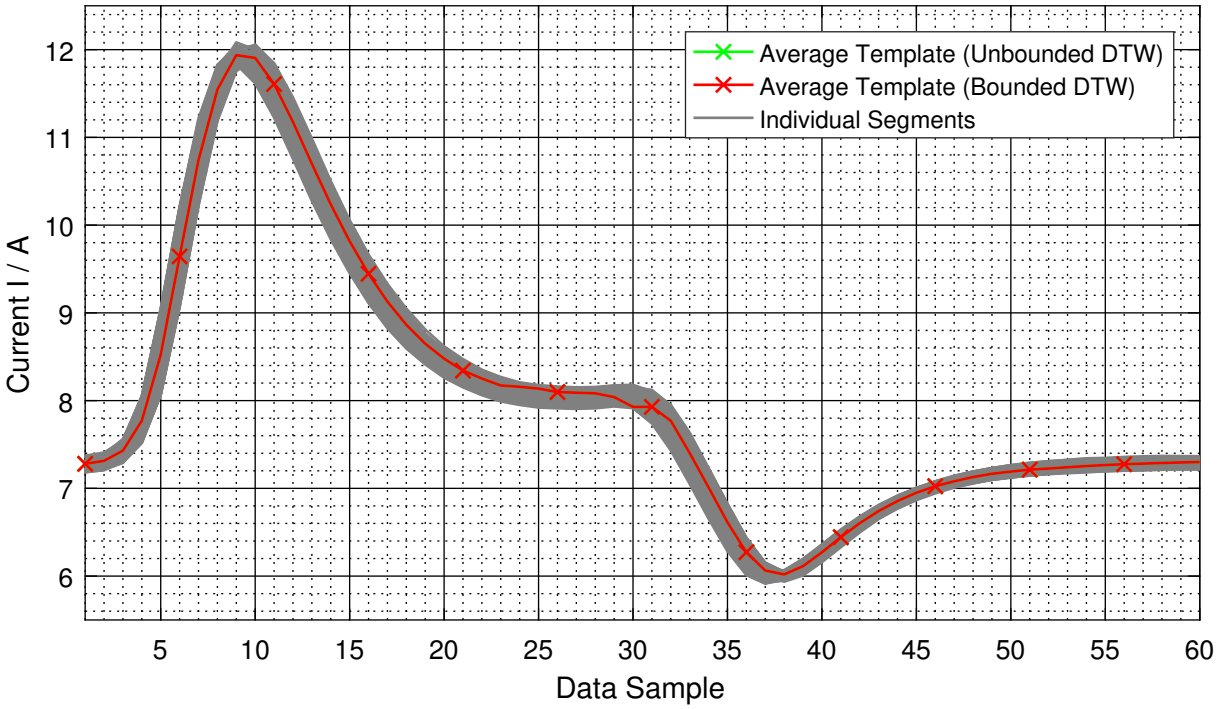


(a) Averaged template for class 8, the C-axis forward movement.

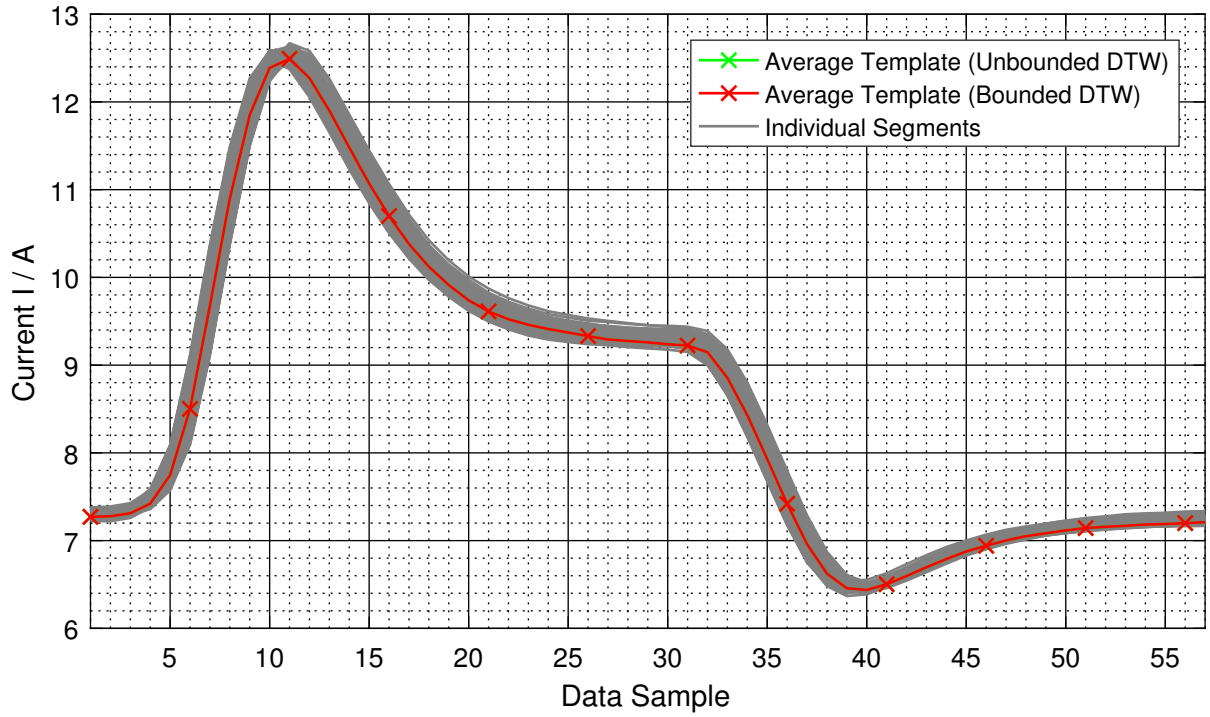


(b) Averaged template for class 9, the C-axis forward movement.

Figure A.9: Resulting averaged templates of the accurate shape averaging algorithm for sequences of classes 8 and 9, in which the C-axis moves forward and backward. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path.



(a) Averaged template for class 10, the B-axis forward movement.



(b) Averaged template for class 11, the B-axis forward movement.

Figure A.10: Resulting averaged templates of the accurate shape averaging algorithm for sequences of classes 10 and 11, in which the B-axis moves forward and backward. Gray lines represent all 100 individual sequences from the second test cycle run, which were used for calculating the averaged sequence, shown as green and red lines with cross markers. The red line corresponds to the averaging result of a DTW with a Sakoe-Chiba band as global constrained for the warping path. Remarkable in this case is the fact that bounding of DTW has no effect on the averaging result.

A.6 Confusion Matrices for Feature-Based Classifiers

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	86	14	0	0	0	0	0	0	0	0
3	0	0	7	93	0	0	0	0	0	0	0	0
4	0	0	0	0	72	28	0	0	0	0	0	0
5	0	0	0	0	14	86	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	98	0	0	2	0
8	0	0	0	0	0	0	0	0	100	0	0	0
9	0	0	0	0	0	0	0	0	8	92	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure A.11: Confusion matrix for the decision tree classifier.

Predicted Classes

True Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	91	8	0	1	0	0	0	0	0	0
3	0	0	2	97	0	1	0	0	0	0	0	0
4	0	0	0	0	84	16	0	0	0	0	0	0
5	0	0	0	0	18	82	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	99	1	0	0
9	0	0	0	0	0	0	0	0	3	97	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure A.12: Confusion matrix for the random forest classifier.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	67	31	2	0	0	0	0	0	0	0
3	0	0	3	90	0	7	0	0	0	0	0	0
4	0	0	0	0	71	29	0	0	0	0	0	0
5	0	0	0	3	34	63	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	95	5	0	0
9	0	0	0	0	0	0	0	0	6	94	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	2	98

Figure A.13: Confusion matrix for the AdaBoost classifier.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	86	13	0	1	0	0	0	0	0	0
3	0	0	3	96	0	1	0	0	0	0	0	0
4	0	0	0	0	82	18	0	0	0	0	0	0
5	0	0	0	4	17	79	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	99	1	0	0
9	0	0	0	0	0	0	0	0	43	57	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure A.14: Confusion matrix for the support vector machine.

Predicted Classes

True Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	95	0	5	0	0	0	0	0	0	0
3	0	0	34	0	66	0	0	0	0	0	0	0
4	0	0	0	0	100	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	0	0	0	0	0	0	0	49	51	0	0
9	0	0	0	0	0	0	0	0	5	95	0	0
10	0	0	0	0	0	0	0	0	0	0	100	0
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure A.15: Confusion matrix for the stochastic gradient descent classifier.

True Classes \ Predicted Classes	0	1	2	3	4	5	6	7	8	9	10	11
0	1100	0	0	0	0	0	0	0	0	0	0	0
1	0	100	0	0	0	0	0	0	0	0	0	0
2	0	0	53	0	47	0	0	0	0	0	0	0
3	0	0	16	0	84	0	0	0	0	0	0	0
4	0	0	0	0	100	0	0	0	0	0	0	0
5	0	0	0	0	100	0	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0
7	0	0	0	0	0	0	0	100	0	0	0	0
8	0	100	0	0	0	0	0	0	0	0	0	0
9	0	84	0	0	0	0	0	0	0	16	0	0
10	0	0	0	0	0	0	0	0	0	0	93	7
11	0	0	0	0	0	0	0	0	0	0	0	100

Figure A.16: Confusion matrix for the naive bayes classifier.

A.7 Matlab Source Codes

A.7.1 Label Assistant GUI

Listing A.3: Matlab source code of the graphical user interface for the assistant software used for manual measurement data segmentation and labelling.

```
function varargout = label_assistant_gui(varargin)
% LABEL_ASSISTANT_GUI MATLAB code for label_assistant_gui.fig
%      LABEL_ASSISTANT_GUI, by itself, creates a new LABEL_ASSISTANT_GUI or raises the existing
%      singleton*.
%
%      H = LABEL_ASSISTANT_GUI returns the handle to a new LABEL_ASSISTANT_GUI or the handle to
%      the existing singleton*.
%
%      LABEL_ASSISTANT_GUI('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in LABEL_ASSISTANT_GUI.M with the given input arguments.
%
%      LABEL_ASSISTANT_GUI('Property','Value',...) creates a new LABEL_ASSISTANT_GUI or raises
the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before label_assistant_gui_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to label_assistant_gui_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help label_assistant_gui

% Last Modified by GUIDE v2.5 16-Apr-2017 13:52:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @label_assistant_gui_OpeningFcn, ...
                   'gui_OutputFcn',  @label_assistant_gui_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin  ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```

```
46
47 % --- Executes just before label_assistant_gui is made visible.
48 function label_assistant_gui_OpeningFcn(hObject, eventdata, handles, varargin)
49 % This function has no output args, see OutputFcn.
50 % hObject    handle to figure
51 % eventdata  reserved - to be defined in a future version of MATLAB
52 % handles    structure with handles and user data (see GUIDATA)
53 % varargin   command line arguments to label_assistant_gui (see VARARGIN)
54
55 % counter variable
56 handles.numcuts = 1;
57
58 % cuts
59 handles.cuts = [];
60
61 % line object handle for lines indicating cuts
62 handles.cutLines1 = -1;
63
64 % inital content of the click history listbox
65 handles.clickedHistory = {};
66
67 % clear table data
68 handles.dataset = [];
69 set(handles.OutputTable,'Data',handles.dataset);
70
71 % read data from csv files
72 handles.dataset_dmu_strom_l1 = csvread('../dataset_dmu_strom_l1.csv');
73 handles.dataset_dmu_strom_l2 = csvread('../dataset_dmu_strom_l2.csv');
74 handles.dataset_dmu_strom_l3 = csvread('../dataset_dmu_strom_l3.csv');
75 handles.dataset_schaltschrank_kuehlung_strom_l1 = csvread('../
   dataset_schaltschrank_kuehlung_strom_l1.csv');
76 handles.dataset_schmieroelpumpe_blasluft_strom_l1 = csvread('../
   dataset_schmieroelpumpe_blasluft_strom_l1.csv');
77 handles.dataset_schmieroelpumpe_oel_luft_strom_l1 = csvread('../
   dataset_schmieroelpumpe_oel_luft_strom_l1.csv');
78 handles.dataset_schmieroelpumpe_maschine_strom_l1 = csvread('../
   dataset_schmieroelpumpe_maschine_strom_l1.csv');
79
80 % drop last samples (only one or two) of too long vector
81 lmin = min([length(handles.dataset_dmu_strom_l1(:,2)),...
82             length(handles.dataset_dmu_strom_l2(:,2)),...
83             length(handles.dataset_dmu_strom_l3(:,2)),...
84             length(handles.dataset_schaltschrank_kuehlung_strom_l1(:,2)),...
85             length(handles.dataset_schmieroelpumpe_blasluft_strom_l1(:,2)),...
86             length(handles.dataset_schmieroelpumpe_oel_luft_strom_l1(:,2)),...
87             length(handles.dataset_schmieroelpumpe_maschine_strom_l1(:,2))]);
88
89 handles.dataset_dmu_strom_l1 = handles.dataset_dmu_strom_l1(1:lmin,2);
90 handles.dataset_dmu_strom_l2 = handles.dataset_dmu_strom_l2(1:lmin,2);
91 handles.dataset_dmu_strom_l3 = handles.dataset_dmu_strom_l3(1:lmin,2);
92 handles.dataset_schaltschrank_kuehlung_strom_l1 = handles.
   dataset_schaltschrank_kuehlung_strom_l1(1:lmin,2);
93 handles.dataset_schmieroelpumpe_blasluft_strom_l1 = handles.
   dataset_schmieroelpumpe_blasluft_strom_l1(1:lmin,2);
```

```
94 handles.dataset_schmieroelpumpe_oel_luft_strom_l1 = handles.
   dataset_schmieroelpumpe_oel_luft_strom_l1(1:lmin,2);
95 handles.dataset_schmieroelpumpe_maschine_strom_l1 = handles.
   dataset_schmieroelpumpe_maschine_strom_l1(1:lmin,2);
96
97 % plot dmu main
98 axes(handles.axes1);
99 plot(handles.dataset_dmu_strom_l1);
100 hold on;
101 plot(handles.dataset_dmu_strom_l2);
102 plot(handles.dataset_dmu_strom_l3);
103 title('DMU Main');
104 legend('L1','L2','L3');
105 grid on;
106 %ylim([-inf, inf]);
107 ylim([0, 12]);
108
109 % plot control cabinet heat exchanger
110 axes(handles.axes2);
111 plot(handles.dataset_schaltschrank_kuehlung_strom_l1);
112 hold on;
113 title('Control Cabinet Heat Exchanger');
114 legend('L1');
115 grid on;
116 %ylim([-inf, inf]);
117 ylim([0, 3]);
118
119 % plot lubrication pumps
120 axes(handles.axes3);
121 plot(handles.dataset_schmieroelpumpe_blasluft_strom_l1);
122 hold on;
123 plot(handles.dataset_schmieroelpumpe_oel_luft_strom_l1);
124 plot(handles.dataset_schmieroelpumpe_maschine_strom_l1);
125 title('Lubrication Pumps');
126 legend('Blower Air','Oil-Air','Machine');
127 grid on;
128 %ylim([-inf, inf]);
129 ylim([0, 0.5]);
130
131 % link axes
132 linkaxes([handles.axes1,handles.axes2,handles.axes3],'x');
133 xlim([0, 3000]);
134
135 % set text update function for data cursor
136 handles.dcm = datacursormode(handles.figure1);
137 handles.dcm.UpdateFcn = {@TextCursorCallback,handles};
138
139 % define machine states
140 handles.classes = (0:95)';
141 handles.component_states = [...
142 0,0,0,0,0,0,0,0,0,0,0,0,0,0;...
143 1,0,0,0,0,0,0,0,0,0,0,0,0,0;...
144 0,1,0,0,0,0,0,0,0,0,0,0,0,0;...
145 0,0,1,0,0,0,0,0,0,0,0,0,0,0;...
```

```
0,0,0,1,0,0,0,0,0,0,0,0,0,0;...
0,0,0,0,1,0,0,0,0,0,0,0,0,0;...
0,0,0,0,0,1,0,0,0,0,0,0,0,0;...
0,0,0,0,0,0,1,0,0,0,0,0,0,0;...
0,0,0,0,0,0,0,1,0,0,0,0,0,0;...
0,0,0,0,0,0,0,0,1,0,0,0,0,0;...
0,0,0,0,0,0,0,0,0,1,0,0,0,0;...
0,0,0,0,0,0,0,0,0,0,0,0,0,1;...
1,0,0,0,0,0,0,0,0,0,0,0,0,1;...
0,1,0,0,0,0,0,0,0,0,0,0,0,1;...
0,0,1,0,0,0,0,0,0,0,0,0,0,1;...
0,0,0,1,0,0,0,0,0,0,0,0,0,1;...
0,0,0,0,1,0,0,0,0,0,0,0,0,1;...
0,0,0,0,0,1,0,0,0,0,0,0,0,1;...
0,0,0,0,0,0,1,0,0,0,0,0,0,1;...
0,0,0,0,0,0,0,1,0,0,0,0,0,1;...
0,0,0,0,0,0,0,0,1,0,0,0,0,1;...
0,0,0,0,0,0,0,0,0,1,0,0,0,1;...
0,0,0,0,0,0,0,0,0,0,1,0,0,1;...
0,0,0,0,0,0,0,0,0,0,0,1,0,1;...
1,0,0,0,0,0,0,0,0,0,0,1,0,1;...
0,1,0,0,0,0,0,0,0,0,0,1,0,1;...
0,0,1,0,0,0,0,0,0,0,0,1,0,1;...
0,0,0,1,0,0,0,0,0,0,0,1,0,1;...
0,0,0,0,1,0,0,0,0,0,0,1,0,1;...
0,0,0,0,0,1,0,0,0,0,0,1,0,1;...
0,0,0,0,0,0,1,0,0,0,0,1,0,1;...
0,0,0,0,0,0,0,1,0,0,0,1,0,1;...
0,0,0,0,0,0,0,0,1,0,0,1,0,1;...
0,0,0,0,0,0,0,0,0,1,0,1,0,1;...
0,0,0,0,0,0,0,0,0,0,1,1,0,1;...
0,0,0,0,0,0,0,0,0,0,0,1,1,1;...
1,0,0,0,0,0,0,0,0,0,0,1,1,1;...
0,1,0,0,0,0,0,0,0,0,0,1,1,1;...
0,0,1,0,0,0,0,0,0,0,0,1,1,1;...
0,0,0,1,0,0,0,0,0,0,0,1,1,1;...
0,0,0,0,1,0,0,0,0,0,0,1,1,1;...
0,0,0,0,0,1,0,0,0,0,0,1,1,1;...
0,0,0,0,0,0,1,0,0,0,0,1,1,1;...
0,0,0,0,0,0,0,1,0,0,0,1,1,1;...
0,0,0,0,0,0,0,0,1,0,0,1,1,1;...
0,0,0,0,0,0,0,0,0,1,0,1,1,1;...
0,0,0,0,0,0,0,0,0,0,1,1,1,1;...
0,0,0,0,0,0,0,0,0,0,0,1,0,0;...
1,0,0,0,0,0,0,0,0,0,0,1,0,0;...
0,1,0,0,0,0,0,0,0,0,0,1,0,0;...
0,0,1,0,0,0,0,0,0,0,0,1,0,0;...
0,0,0,1,0,0,0,0,0,0,0,1,0,0;...
0,0,0,0,1,0,0,0,0,0,0,1,0,0;...
0,0,0,0,0,1,0,0,0,0,0,1,0,0;...
0,0,0,0,0,0,1,0,0,0,0,1,0,0;...
0,0,0,0,0,0,0,1,0,0,0,1,0,0;...
0,0,0,0,0,0,0,0,1,0,0,1,0,0;...
```

```
0,0,0,0,0,0,0,0,0,1,0,1,0,0;...
0,0,0,0,0,0,0,0,0,0,1,1,0,0;...
0,0,0,0,0,0,0,0,0,0,0,1,0,1;...
1,0,0,0,0,0,0,0,0,0,0,1,0,1;...
0,1,0,0,0,0,0,0,0,0,0,1,0,1;...
0,0,1,0,0,0,0,0,0,0,0,1,0,1;...
0,0,0,1,0,0,0,0,0,0,0,1,0,1;...
0,0,0,0,1,0,0,0,0,0,0,1,0,1;...
0,0,0,0,0,1,0,0,0,0,0,1,0,1;...
0,0,0,0,0,0,1,0,0,0,0,1,0,1;...
0,0,0,0,0,0,0,1,0,0,0,1,0,1;...
0,0,0,0,0,0,0,0,1,0,1,0,0,1;...
0,0,0,0,0,0,0,0,0,1,1,0,0,1;...
0,0,0,0,0,0,0,0,0,0,0,1,1,0;...
1,0,0,0,0,0,0,0,0,0,0,1,1,0;...
0,1,0,0,0,0,0,0,0,0,0,1,1,0;...
0,0,1,0,0,0,0,0,0,0,0,1,1,0;...
0,0,0,1,0,0,0,0,0,0,0,1,1,0;...
0,0,0,0,1,0,0,0,0,0,0,1,1,0;...
0,0,0,0,0,1,0,0,0,0,0,1,1,0;...
0,0,0,0,0,0,1,0,0,0,0,1,1,0;...
0,0,0,0,0,0,0,1,0,0,1,1,0,0;...
0,0,0,0,0,0,0,0,1,0,1,1,0,0;...
0,0,0,0,0,0,0,0,0,1,1,1,0,0;...
0,0,0,0,0,0,0,0,0,0,0,1,1,1;...
1,0,0,0,0,0,0,0,0,0,0,1,1,1;...
0,1,0,0,0,0,0,0,0,0,0,1,1,1;...
0,0,1,0,0,0,0,0,0,0,0,1,1,1;...
0,0,0,1,0,0,0,0,0,0,0,1,1,1;...
0,0,0,0,1,0,0,0,0,0,0,1,1,1;...
0,0,0,0,0,1,0,0,0,0,0,1,1,1;...
0,0,0,0,0,0,1,0,0,0,0,1,1,1;...
0,0,0,0,0,0,0,1,0,0,0,1,1,1;...
0,0,0,0,0,0,0,0,1,0,0,1,1,1;...
0,0,0,0,0,0,0,0,0,1,0,1,1,1;...
0,0,0,0,0,0,0,0,0,0,1,1,1,1
];

% Choose default command line output for label_assistant_gui
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes label_assistant_gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = label_assistant_gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in getCutIndex.
function getCutIndex_Callback(hObject, eventdata, handles)
% hObject    handle to getCutIndex (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% get data from data cursor
handles.infoStruct = getCursorInfo(handles.dcm);

% store data into dataset array
if ~isempty(handles.infoStruct)

    % get data from nominal checkbox
    handles.nominalCut = get(handles.nominalCheckBox,'Value');

    % store data into cuts array
    handles.cuts(handles.numcuts,:) = [handles.infoStruct.DataIndex, handles.nominalCut];

    % reset nominal checkbox after each click
    set(handles.nominalCheckBox,'Value',1);

    % get current class from selection of state selection checkboxes and radios
    labelset = zeros(1,14);
    labelset(1) = get(handles.MainSpindleRadiobutton,'Value');
    labelset(2) = get(handles.XAxisForwardRadiobutton,'Value');
    labelset(3) = get(handles.XAxisBackwardRadiobutton,'Value');
    labelset(4) = get(handles.YAxisForwardRadiobutton,'Value');
    labelset(5) = get(handles.YAxisBackwardRadiobutton,'Value');
    labelset(6) = get(handles.ZAxisForwardRadiobutton,'Value');
    labelset(7) = get(handles.ZAxisBackwardRadiobutton,'Value');
    labelset(8) = get(handles.BAxisForwardRadiobutton,'Value');
    labelset(9) = get(handles.BAxisBackwardRadiobutton,'Value');
    labelset(10) = get(handles.CAxisForwardRadiobutton,'Value');
    labelset(11) = get(handles.CAxisBackwardRadiobutton,'Value');
    labelset(12) = get(handles.MachineLubricationPumpCheckbox,'Value');
    labelset(13) = get(handles.OilAirLubricationPumpCheckbox,'Value');
    labelset(14) = get(handles.ControlCabinetHeatExchangerCheckbox,'Value');

    % caculate classs for each sample via table lookup
    for i = 1:length(handles.classes(:,1))
        if labelset(1,:) == handles.component_states(i,:)
            handles.targets(handles.numcuts,1) = handles.classes(i);
        end
    end

    % increment iteration variable
    handles.numcuts = handles.numcuts + 1;
```

```
308
309     % update dataset
310     handles.dataset = [handles.cuts, handles.targets];
311
312     % sort data automatically
313     [~,ind] = sort(handles.dataset(:,1),'descend');
314     handles.dataset = handles.dataset(ind,:);
315
316     % update table and scroll to table bottom
317     set(handles.OutputTable,'Data',handles.dataset);
318
319     % add lines to plots
320     xl = xlim(handles.axes1);
321     if handles.cutLines1 ~= -1
322         delete(handles.cutLines1);
323         delete(handles.cutLines2);
324         delete(handles.cutLines3);
325     end
326     yl1 = ylim(handles.axes1);
327     yl2 = ylim(handles.axes2);
328     yl3 = ylim(handles.axes3);
329     tx = [handles.cuts(:,1).';handles.cuts(:,1).';nan(1,length(handles.cuts(:,1)))];
330     ty1 = [repmat(yl1(1),length(handles.cuts(:,1)),1).';repmat(yl1(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
331     ty2 = [repmat(yl2(1),length(handles.cuts(:,1)),1).';repmat(yl2(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
332     ty3 = [repmat(yl3(1),length(handles.cuts(:,1)),1).';repmat(yl3(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
333     handles.cutLines1 = plot(handles.axes1,tx(:),ty1(:),'k:');
334     handles.cutLines2 = plot(handles.axes2,tx(:),ty2(:),'k:');
335     handles.cutLines3 = plot(handles.axes3,tx(:),ty3(:),'k:');
336     xlim(handles.axes1,xl);
337     xlim(handles.axes2,xl);
338     xlim(handles.axes3,xl);
339
340     % move viewport to right (shift x axis left) so that latest cursor is on left axes border
341     xl = xlim(handles.axes1);
342     xl_dif = xl(2)-xl(1);
343     xl = [handles.dataset(1,1)-200, handles.dataset(1,1)-200+xl_dif];
344     xlim(handles.axes1,xl);
345     xlim(handles.axes2,xl);
346     xlim(handles.axes3,xl);
347
348     % disable get sample button
349     set(handles.getCutIndex,'Enable','off');
350
351     % update click history
352     clicked_index = find(labelset(1:11));
353     if isempty(clicked_index)
354         clicked_index = -1;
355     end
356     clickHistoryOld = handles.clickedHistory;
357     if length(handles.clickedHistory) > 1
358         handles.clickedHistory{2} = clickHistoryOld{1};
```

```
359         handles.clickedHistory{3} = clickHistoryOld{2};
360     elseif length(handles.clickedHistory) == 1
361         handles.clickedHistory{2} = clickHistoryOld{1};
362     end
363     switch clicked_index
364         case 1
365             handles.clickedHistory{1} = 'Main Spindle';
366         case 2
367             handles.clickedHistory{1} = 'X Axis Forward';
368         case 3
369             handles.clickedHistory{1} = 'X Axis Backward';
370         case 4
371             handles.clickedHistory{1} = 'Y Axis Forward';
372         case 5
373             handles.clickedHistory{1} = 'Y Axis Backward';
374         case 6
375             handles.clickedHistory{1} = 'Z Axis Forward';
376         case 7
377             handles.clickedHistory{1} = 'Z Axis Backward';
378         case 8
379             handles.clickedHistory{1} = 'B Axis Forward';
380         case 9
381             handles.clickedHistory{1} = 'B Axis Backward';
382         case 10
383             handles.clickedHistory{1} = 'C Axis Forward';
384         case 11
385             handles.clickedHistory{1} = 'C Axis Backward';
386         case -1
387             handles.clickedHistory{1} = 'None';
388     end
389     set(handles.listbox3,'String',handles.clickedHistory);
390 
391 end
392 
393 % Update handles structure
394 guidata(hObject, handles);
395 
396 
397 % --- Executes when entered data in editable cell(s) in OutputTable.
398 function OutputTable_CellEditCallback(hObject, eventdata, handles)
399 % hObject    handle to OutputTable (see GCBO)
400 % eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
401 %       Indices: row and column indices of the cell(s) edited
402 %       PreviousData: previous data for the cell(s) edited
403 %       EditData: string(s) entered by the user
404 %       NewData: EditData or its converted form set on the Data property. Empty if Data was not
     changed
405 %       Error: error string when failed to convert EditData to appropriate value for Data
406 % handles    structure with handles and user data (see GUIDATA)
407 
408 % get dataset from table whenever user makes an edit
409 handles.dataset = get(handles.OutputTable,'data');
410 
411 % update individual data vectors
```

```
412 handles.cuts = handles.dataset(:,1:2);
413 handles.targets = handles.dataset(:,3);
414
415 % redraw lines on plots
416 xl = xlim(handles.axes1);
417 if handles.cutLines1 ~= -1
418     delete(handles.cutLines1);
419     delete(handles.cutLines2);
420     delete(handles.cutLines3);
421 end
422 yl1 = ylim(handles.axes1);
423 yl2 = ylim(handles.axes2);
424 yl3 = ylim(handles.axes3);
425 tx = [handles.cuts(:,1).';handles.cuts(:,1).';nan(1,length(handles.cuts(:,1)))];
426 ty1 = [repmat(yl1(1),length(handles.cuts(:,1)),1).';repmat(yl1(2),length(handles.cuts(:,1)),1)
    .';nan(1,length(handles.cuts(:,1)))];
427 ty2 = [repmat(yl2(1),length(handles.cuts(:,1)),1).';repmat(yl2(2),length(handles.cuts(:,1)),1)
    .';nan(1,length(handles.cuts(:,1)))];
428 ty3 = [repmat(yl3(1),length(handles.cuts(:,1)),1).';repmat(yl3(2),length(handles.cuts(:,1)),1)
    .';nan(1,length(handles.cuts(:,1)))];
429 handles.cutLines1 = plot(handles.axes1,tx(:),ty1(:),'k:');
430 handles.cutLines2 = plot(handles.axes2,tx(:),ty2(:),'k:');
431 handles.cutLines3 = plot(handles.axes3,tx(:),ty3(:),'k:');
432 xlim(handles.axes1,xl);
433 xlim(handles.axes2,xl);
434 xlim(handles.axes3,xl);
435
436 % Update handles structure
437 guidata(hObject, handles);
438
439
440 % --- Executes on button press in ClearDatasetButton.
441 function ClearDatasetButton_Callback(hObject, eventdata, handles)
442 % hObject    handle to ClearDatasetButton (see GCBO)
443 % eventdata  reserved - to be defined in a future version of MATLAB
444 % handles    structure with handles and user data (see GUIDATA)
445
446 % show question box for asking the user if he is sure to delete dataset
447 dlgTitle    = 'Delete Dataset';
448 dlgQuestion = 'Do you really want to delete the entire dataset? The dataset can not be restored
    .';
449 choice = questdlg(dlgQuestion,dlgTitle,'Yes','No','No');
450
451 % if choice was yes, delete entire dataset
452 if strcmp(choice,'Yes')
453     handles.dataset = [];
454     handles.cuts = [];
455     handles.targets = [];
456 end
457
458 % counter variable
459 handles.numcuts = 1;
460
461 % clear table
```

```
462 set(handles.OutputTable,'Data',handles.dataset);
463
464 % delete cut lines from plots
465 if handles.cutLines1 ~= -1
466     delete(handles.cutLines1);
467     delete(handles.cutLines2);
468     delete(handles.cutLines3);
469 end
470
471 % delete entries in click history listbox
472 handles.clickedHistory = {};
473 set(handles.listbox3,'String',handles.clickedHistory);
474
475 % Update handles structure
476 guidata(hObject, handles);
477
478
479 % --- Executes on button press in DeleteRowButton.
480 function DeleteRowButton_Callback(hObject, eventdata, handles)
481 % hObject    handle to DeleteRowButton (see GCBO)
482 % eventdata  reserved - to be defined in a future version of MATLAB
483 % handles    structure with handles and user data (see GUIDATA)
484
485 % show question box for asking the user if he is sure to delete selected rows
486 dlgTitle    = 'Delete Selected Rows';
487 dlgQuestion = 'Do you really want to delete any selected rows? The rows can not be restored.';
488 choice = questdlg(dlgQuestion,dlgTitle,'Yes','No','No');
489
490 % if choice was yes, delete selected rows
491 if strcmp(choice,'Yes')
492     % delete rows
493     handles.dataset(handles.rows,:) = [];
494
495     % update counter variable
496     handles.numcuts = handles.numcuts-length(handles.rows);
497
498     % update individual data vectors
499     handles.cuts = handles.dataset(:,1:2);
500     handles.targets = handles.dataset(:,3);
501
502     % update table
503     set(handles.OutputTable,'Data',handles.dataset);
504
505     % redraw lines on plots
506     xl = xlim(handles.axes1);
507     if handles.cutLines1 ~= -1
508         delete(handles.cutLines1);
509         delete(handles.cutLines2);
510         delete(handles.cutLines3);
511     end
512     yl1 = ylim(handles.axes1);
513     yl2 = ylim(handles.axes2);
514     yl3 = ylim(handles.axes3);
515     tx = [handles.cuts(:,1).';handles.cuts(:,1).';nan(1,length(handles.cuts(:,1)))];
```

```
516     ty1 = [repmat(yl1(1),length(handles.cuts(:,1)),1).';repmat(yl1(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
517     ty2 = [repmat(yl2(1),length(handles.cuts(:,1)),1).';repmat(yl2(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
518     ty3 = [repmat(yl3(1),length(handles.cuts(:,1)),1).';repmat(yl3(2),length(handles.cuts(:,1))
        ,1).';nan(1,length(handles.cuts(:,1)))];
519     handles.cutLines1 = plot(handles.axes1,tx(:),ty1(:),'k:');
520     handles.cutLines2 = plot(handles.axes2,tx(:),ty2(:),'k:');
521     handles.cutLines3 = plot(handles.axes3,tx(:),ty3(:),'k:');
522     xlim(handles.axes1,xl);
523     xlim(handles.axes2,xl);
524     xlim(handles.axes3,xl);
525
526     % check if dataset is not empty now
527     if ~isempty(handles.dataset)
528         % move viewport to right (shift x axis left) so that latest cursor is on left axes
            border
529         xl = xlim(handles.axes1);
530         xl_dif = xl(2)-xl(1);
531         xl = [handles.dataset(1,1)-200, handles.dataset(1,1)-200+xl_dif];
532         xlim(handles.axes1,xl);
533         xlim(handles.axes2,xl);
534         xlim(handles.axes3,xl);
535     end
536
537     % disable delete rows button
538     set(handles.DeleteRowButton,'Enable','off');
539 end
540
541 % Update handles structure
542 guidata(hObject, handles);
543
544
545 % --- Executes when selected cell(s) is changed in OutputTable.
546 function OutputTable_CellSelectionCallback(hObject, eventdata, handles)
547 % hObject    handle to OutputTable (see GCBO)
548 % eventdata  structure with the following fields (see MATLAB.UI.CONTROL.TABLE)
549 %       Indices: row and column indices of the cell(s) currently selecteds
550 % handles    structure with handles and user data (see GUIDATA)
551
552 % activate delete rows button
553 set(handles.DeleteRowButton,'Enable','on');
554
555 % get indices of selected rows
556 index = eventdata.Indices;
557 if any(index)             %loop necessary to surpress unimportant errors.
558     handles.rows = index(:,1);
559 end
560
561 % Update handles structure
562 guidata(hObject, handles);
563
564
565 % --- Executes when selected object is changed in uibuttongroup1.
```

```
566 function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)
567 % hObject    handle to the selected object in uibuttongroup1
568 % eventdata  reserved - to be defined in a future version of MATLAB
569 % handles    structure with handles and user data (see GUIDATA)
570
571 % if button group has been changed, enable get sample button
572 set(handles.getCutIndex,'Enable','on');
573
574 % Update handles structure
575 guidata(hObject, handles);
576
577
578 % --------------------------------------------------------------------
579 function uipushtool2_ClickedCallback(hObject, eventdata, handles)
580 % hObject    handle to uipushtool2 (see GCBO)
581 % eventdata  reserved - to be defined in a future version of MATLAB
582 % handles    structure with handles and user data (see GUIDATA)
583
584 % show open dialog
585 [fileName,pathName] = uigetfile('*.csv','Select existing dataset.','dataset.csv');
586 fullPath = strcat(pathName,fileName);
587
588 % try catch block for preventing empty files to be loaded
589 try
590     % read file
591     readFile = csvread(fullPath);
592     readFile = flipud(readFile);
593
594     % check if dataset has three rows
595     if size(readFile,2) == 3
596
597         % update counter variable
598         handles.numcuts = length(readFile(:,1));
599
600         % update dataset
601         handles.dataset = readFile;
602
603         % update individual data vectors
604         handles.cuts = handles.dataset(:,1:2);
605         handles.targets = handles.dataset(:,3);
606
607         % update table
608         set(handles.OutputTable,'Data',handles.dataset);
609
610         % redraw lines on plots
611         xl = xlim(handles.axes1);
612         if handles.cutLines1 ~= -1
613             delete(handles.cutLines1);
614             delete(handles.cutLines2);
615             delete(handles.cutLines3);
616         end
617         yl1 = ylim(handles.axes1);
618         yl2 = ylim(handles.axes2);
619         yl3 = ylim(handles.axes3);
```

```
620         tx = [handles.cuts(:,1).';handles.cuts(:,1).';nan(1,length(handles.cuts(:,1)))];
621         ty1 = [repmat(yl1(1),length(handles.cuts(:,1)),1).';repmat(yl1(2),length(handles.cuts
            (:,1)),1).';nan(1,length(handles.cuts(:,1)))];
622         ty2 = [repmat(yl2(1),length(handles.cuts(:,1)),1).';repmat(yl2(2),length(handles.cuts
            (:,1)),1).';nan(1,length(handles.cuts(:,1)))];
623         ty3 = [repmat(yl3(1),length(handles.cuts(:,1)),1).';repmat(yl3(2),length(handles.cuts
            (:,1)),1).';nan(1,length(handles.cuts(:,1)))];
624         handles.cutLines1 = plot(handles.axes1,tx(:),ty1(:),'k:');
625         handles.cutLines2 = plot(handles.axes2,tx(:),ty2(:),'k:');
626         handles.cutLines3 = plot(handles.axes3,tx(:),ty3(:),'k:');
627         xlim(handles.axes1,xl);
628         xlim(handles.axes2,xl);
629         xlim(handles.axes3,xl);
630 
631         % move viewport to right (shift x axis left) so that latest cursor is on left axes
            border
632         xl = xlim(handles.axes1);
633         xl_dif = xl(2)-xl(1);
634         xl = [handles.dataset(1,1)-200, handles.dataset(1,1)-200+xl_dif];
635         xlim(handles.axes1,xl);
636         xlim(handles.axes2,xl);
637         xlim(handles.axes3,xl);
638     end
639 catch
640     msgTitle = 'Import Error';
641     msgText = 'Imported dataset is empty.';
642     msgbox(msgText,msgTitle,'error');
643 end
644 
645 % Update handles structure
646 guidata(hObject, handles);
647 
648 
649 %--------------------------------------------------------------------------
650 function uipushtool3_ClickedCallback(hObject, eventdata, handles)
651 % hObject    handle to uipushtool3 (see GCBO)
652 % eventdata  reserved - to be defined in a future version of MATLAB
653 % handles    structure with handles and user data (see GUIDATA)
654 
655 % show save dialog
656 [fileName,pathName] = uiputfile('dataset.csv','Save Dataset');
657 fullPath = strcat(pathName,fileName);
658 
659 try
660     if isempty(handles.dataset)
661         % show error message
662         msgTitle = 'Save Error';
663         msgText = 'Dataset is empty. No file has been written';
664         msgbox(msgText,msgTitle,'error');
665     else
666         % flip dataset upside down
667         dataset_flipped = flipud(handles.dataset);
668 
669         % save datatset to csv file
```

```
670         dlmwrite(fullPath,dataset_flipped,'precision',9);
671 
672         % show error message
673         msgTitle = 'Save Dataset';
674         msgText = strcat('Dataset sucessfully saved to file " ',fileName,'".');
675         msgbox(msgText,msgTitle,'warn');
676     end
677 catch
678     msgTitle = 'Save Error';
679     msgText = 'Could not open file.';
680     msgbox(msgText,msgTitle,'error');
681 end
```

Listing A.4: Matlab source code of the text cursor callback function used in the segmentation GUI script.

```
1  function output_txt = myfunction(obj,event_obj,handles)
2  % Display the position of the data cursor
3  % obj          Currently not used (empty)
4  % event_obj    Handle to event object
5  % output_txt   Data cursor text string (string or cell array of strings).
6 
7  pos = get(event_obj,'Position');
8  output_txt = {['X: ',num2str(pos(1),10)],...
9      ['Y: ',num2str(pos(2),10)]};
10 
11 % If there is a Z-coordinate in the position, display it as well
12 if length(pos) > 2
13     output_txt{end+1} = ['Z: ',num2str(pos(3),10)];
14 end
```

A.7.2 Measurement Data Plotting Script

Listing A.5: Matlab source code of the plotting script for all measurement data.

```
%% Script for plotting of measurement data
% Lukas Bommes
% IWF TU Braunschweig
% 28.04.2017
clear all, clc, close all;
format short;

%% select subfolder of dataset to analyse
dataset_subfolder = 'Data_Run_1';

%% read csv files
[main_line_l1,...
main_line_l2,...
main_line_l3,...
chip_conveyor_lifting_pump_l1,...
chip_conveyor_lifting_pump_l2,...
chip_conveyor_lifting_pump_l3,...
hydraulic_pump_l1,...
hydraulic_pump_l2,...
hydraulic_pump_l3,...
cooling_unit_l1,...
cooling_unit_l2,...
cooling_unit_l3,...
control_cabinet_heat_exchanger_l1,...
blower_air_lubrication_pump_l1,...
oil_air_lubrication_pump_l1,...
machine_lubrication_pump_l1] = get_data(dataset_subfolder);

%% plot raw data
% DMU Main
fmain = figure;
clf;
plot(main_line_l1);
hold on;
plot(main_line_l2);
plot(main_line_l3);
title('DMU Main');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1','L2','L3');
grid on;
axis tight;
ylim([0, 60]);

% Chip Conveyor Lifting Pump
figure;
clf;
plot(chip_conveyor_lifting_pump_l1);
hold on;
```

```
plot(chip_conveyor_lifting_pump_l2);
plot(chip_conveyor_lifting_pump_l3);
title('Chip Conveyor Lifting Pump');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1','L2','L3');
grid on;
axis tight;
ylim([0, 0.5]);

% Hydraulic Pump
figure;
clf;
plot(hydraulic_pump_l1);
hold on;
plot(hydraulic_pump_l2);
plot(hydraulic_pump_l3);
title('Hydraulic Pump');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1','L2','L3');
grid on;
axis tight;
ylim([0, 0.5]);

% Cooling Unit
figure;
clf;
plot(cooling_unit_l1);
hold on;
plot(cooling_unit_l2);
plot(cooling_unit_l3);
title('Cooling Unit');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1','L2','L3');
grid on;
axis tight;
ylim([0, 8]);

% Control Cabinet Heat Exchanger
figure;
clf;
plot(control_cabinet_heat_exchanger_l1);
title('Control Cabinet Heat Exchanger');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1');
grid on;
axis tight;
ylim([0, 12]);

% Lubrication Pumps
figure;
```

```
clf;
plot(blower_air_lubrication_pump_l1);
hold on;
plot(oil_air_lubrication_pump_l1);
plot(machine_lubrication_pump_l1);
title('Lubrication Pumps');
xlabel('Data Sample');
ylabel('Current I / A');
legend('Blower Air','Oil–Air','Machine');
grid on;
axis tight;
ylim([0, 0.5]);
```

Listing A.6: Matlab source code of the function used to read out the CSV files containing measurement data.

```
function [main_line_l1,...
          main_line_l2,...
          main_line_l3,...
          chip_conveyor_lifting_pump_l1,...
          chip_conveyor_lifting_pump_l2,...
          chip_conveyor_lifting_pump_l3,...
          hydraulic_pump_l1,...
          hydraulic_pump_l2,...
          hydraulic_pump_l3,...
          cooling_unit_l1,...
          cooling_unit_l2,...
          cooling_unit_l3,...
          control_cabinet_heat_exchanger_l1,...
          blower_air_lubrication_pump_l1,...
          oil_air_lubrication_pump_l1,...
          machine_lubrication_pump_l1] = get_data(dataset_subfolder)
main_line_l1 = csvread([dataset_subfolder,'/main_line_l1.csv']);
main_line_l2 = csvread([dataset_subfolder,'/main_line_l2.csv']);
main_line_l3 = csvread([dataset_subfolder,'/main_line_l3.csv']);
chip_conveyor_lifting_pump_l1 = csvread([dataset_subfolder,'/chip_conveyor_lifting_pump_l1.csv'
]);
chip_conveyor_lifting_pump_l2 = csvread([dataset_subfolder,'/chip_conveyor_lifting_pump_l2.csv'
]);
chip_conveyor_lifting_pump_l3 = csvread([dataset_subfolder,'/chip_conveyor_lifting_pump_l3.csv'
]);
hydraulic_pump_l1 = csvread([dataset_subfolder,'/hydraulic_pump_l1.csv']);
hydraulic_pump_l2 = csvread([dataset_subfolder,'/hydraulic_pump_l2.csv']);
hydraulic_pump_l3 = csvread([dataset_subfolder,'/hydraulic_pump_l3.csv']);
cooling_unit_l1 = csvread([dataset_subfolder,'/cooling_unit_l1.csv']);
cooling_unit_l2 = csvread([dataset_subfolder,'/cooling_unit_l2.csv']);
cooling_unit_l3 = csvread([dataset_subfolder,'/cooling_unit_l3.csv']);
control_cabinet_heat_exchanger_l1 = csvread([dataset_subfolder,'/
control_cabinet_heat_exchanger_l1.csv']);
blower_air_lubrication_pump_l1 = csvread([dataset_subfolder,'/blower_air_lubrication_pump_l1.
csv']);
oil_air_lubrication_pump_l1 = csvread([dataset_subfolder,'/oil_air_lubrication_pump_l1.csv']);
machine_lubrication_pump_l1 = csvread([dataset_subfolder,'/machine_lubrication_pump_l1.csv']);
```

```
33
34 %% drop last samples (only one or two) of too long vector
35 lmin = min([length(main_line_l1(:,2)),...
36            length(main_line_l2(:,2)),...
37            length(main_line_l3(:,2)),...
38            length(chip_conveyor_lifting_pump_l1(:,2)),...
39            length(chip_conveyor_lifting_pump_l2(:,2)),...
40            length(chip_conveyor_lifting_pump_l3(:,2)),...
41            length(hydraulic_pump_l1(:,2)),...
42            length(hydraulic_pump_l2(:,2)),...
43            length(hydraulic_pump_l3(:,2)),...
44            length(cooling_unit_l1(:,2)),...
45            length(cooling_unit_l2(:,2)),...
46            length(cooling_unit_l3(:,2)),...
47            length(control_cabinet_heat_exchanger_l1(:,2)),...
48            length(blower_air_lubrication_pump_l1(:,2)),...
49            length(oil_air_lubrication_pump_l1(:,2)),...
50            length(machine_lubrication_pump_l1(:,2))]);
51
52 main_line_l1 = main_line_l1(1:lmin,2);
53 main_line_l2 = main_line_l2(1:lmin,2);
54 main_line_l3 = main_line_l3(1:lmin,2);
55 chip_conveyor_lifting_pump_l1 = chip_conveyor_lifting_pump_l1(1:lmin,2);
56 chip_conveyor_lifting_pump_l2 = chip_conveyor_lifting_pump_l2(1:lmin,2);
57 chip_conveyor_lifting_pump_l3 = chip_conveyor_lifting_pump_l3(1:lmin,2);
58 hydraulic_pump_l1 = hydraulic_pump_l1(1:lmin,2);
59 hydraulic_pump_l2 = hydraulic_pump_l2(1:lmin,2);
60 hydraulic_pump_l3 = hydraulic_pump_l3(1:lmin,2);
61 cooling_unit_l1 = cooling_unit_l1(1:lmin,2);
62 cooling_unit_l2 = cooling_unit_l2(1:lmin,2);
63 cooling_unit_l3 = cooling_unit_l3(1:lmin,2);
64 control_cabinet_heat_exchanger_l1 = control_cabinet_heat_exchanger_l1(1:lmin,2);
65 blower_air_lubrication_pump_l1 = blower_air_lubrication_pump_l1(1:lmin,2);
66 oil_air_lubrication_pump_l1 = oil_air_lubrication_pump_l1(1:lmin,2);
67 machine_lubrication_pump_l1 = machine_lubrication_pump_l1(1:lmin,2);
68 end
```

A.7.3 Automatic Segmentation and Labeling Algorithm

Listing A.7: Matlab source code of the automatic segmentation and labeling algorithm.

```
%% Script for shape-based automatic cutting
% Lukas Bommes
% IWF TU Braunschweig
% 08.05.2017
% !!Run N01_Plotting.m first!!

%% create shape dicitionary from first subcycle of second testcycle run
[~,main_line_l2_2,~,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_2');
c{1} = main_line_l2_2(880:2020); % spindle
c{2} = main_line_l2_2(7084:7320); % x-axis forward
c{3} = main_line_l2_2(7423:7660); % x-axis backward
c{4} = main_line_l2_2(10465:10626); % y-axis forward
c{5} = main_line_l2_2(10727:10887); % y-axis backward
c{6} = main_line_l2_2(13088:13215); % z-axis forward
c{7} = main_line_l2_2(13305:13432); % z-axis backward
c{8} = main_line_l2_2(15212:15290); % c-axis forward
c{9} = main_line_l2_2(15374:15457); % c-axis backward
c{10} = main_line_l2_2(16850:16908); % b-axis forward
c{11} = main_line_l2_2(16992:17047); % b-axis backward

%% match templates to find positions
% number of subcycles in one testcycle (normally 20)
num_of_runs = 20;
% variable holds positions of cuts
peak_pos = 0;

% repeat for every template
for k = 1:11
    % Determine, how many peaks to find for the current type of template.
    if(ismember(k,[1,6,7,10,11]))
        num_of_peaks = 5*num_of_runs;
    else
        num_of_peaks = 10*num_of_runs;
    end

    % slide template indexwise over dataset_dmu_strom_l2
    % calculate for each window position a distance measure (e.g. inf norm)
    % find local minima in similarity measure over lag
    templ = c{k};
    tmplen = length(templ);
    len = length(main_line_l2)-tmplen;
    dist = zeros(len,1);
    for i = 1:len
        dist(i) = norm(main_line_l2(i:i+tmplen-1)-templ,Inf);
    end

    % invert distance to make it usable for findpeaks function
    dist = -dist;

```

```
50          % find peaks in the signal and append to peak_pos
51      len = nnz(peak_pos);
52          [~, pp] = findpeaks(dist,'SortStr','descend','NPeaks',num_of_peaks);
53
54      % combine cutting points and endpoints of templates to cutting vector
55      peak_pos(len+1:len+2*num_of_peaks,1) = [pp; pp+tmplen];
56
57      % create variable which contains class numbers, but combines classes,
58      % which can't be distiguished by the algorithm, to a single classs
59      switch(k)
60          case {1,6,7,10,11}
61              r = k;
62          case {2,3}
63              r = 2;
64          case {4,5}
65              r = 4;
66          case {8,9}
67              r = 8;
68      end
69
70      % append current class as well as endpoint class (0) to the segment
71      classes(len+1:len+2*num_of_peaks,1) = ...
72          [repmat(r,num_of_peaks,1); zeros(num_of_peaks,1)];
73  end
74
75  % sort peak_pos in ascending order and sort classes accordingly
76  [peak_pos, sortidx] = sort(peak_pos);
77  classes = classes(sortidx);
78
79  % check, if there are approximative duplicates in the peak vector and
80  % remove them. Approximative duplicates are values, that are closer than
81  % 50 samples to each other. Remove from classes vector accordingly.
82  peak_pos_diff = diff(peak_pos);
83  idx = find(peak_pos_diff<50);
84  peak_pos(idx) = [];
85  classes(idx) = [];
86
87  % search for classes 2, 4 and 8 and replace every second one by the
88  % corresponding class 3, 5 or 9 (backward movements of x-, y- and z-axis)
89  for i = 1:length(classes)-2
90      if isequal(classes(i:i+2),[2;0;2])
91          classes(i:i+2) = [2;0;3];
92      elseif isequal(classes(i:i+2),[4;0;4])
93          classes(i:i+2) = [4;0;5];
94      elseif isequal(classes(i:i+2),[8;0;8])
95          classes(i:i+2) = [8;0;9];
96      end
97  end
98
99  % add inital cut on first sample of the dataset, where the state of the
100 % corresponding first segment must be idle (0)
101 peak_pos = [1; peak_pos];
102 classes = [0; classes];
103
```

```
%% check if class sequence is nominal
seq = [1,0,1,0,1,0,1,0,1,0,2,0,3,0,2,0,3,0,2,0,3,0,2,0,3,0,2,0,3,0,...
       4,0,5,0,4,0,5,0,4,0,5,0,4,0,5,0,4,0,5,0,6,0,7,0,6,0,7,0,6,0,...
       7,0,6,0,7,0,6,0,7,0,8,0,9,0,8,0,9,0,8,0,9,0,8,0,9,0,8,0,9,0,...
       10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0];
seq = repmat(seq,1,num_of_runs);
seq = [0, seq];
if isequal(seq',classes)
    disp('Class sequence is valid!');
else
    disp('Class sequence is invalid! Check manually for any faults.');
end

%% create labelset csv file containing nominal cuts and base classes
labelset = [peak_pos ones(size(peak_pos)) classes];
dlmwrite('Labelsets/labelset_nominal.csv',labelset,'precision',10);

%% get state changes under considertation of non-controllable components
% transform signals of non-controllable components in binary sequence by
% thresholding
cche = (control_cabinet_heat_exchanger_l1 >= 0.6);
mlp = (machine_lubrication_pump_l1 >= 0.1);
olp = (oil_air_lubrication_pump_l1 >= 0.1);

% differentiate to find all state changes in the signal
cche_stchg = find(diff(cche));
mlp_stchg = find(diff(mlp));
olp_stchg = find(diff(olp));

% merge all non-controllable state changes into one vector, sort and
% add state change at first sample
nc_stchg = [mlp_stchg; olp_stchg; cche_stchg];
nc_stchg = sort(nc_stchg);
nc_stchg = [1; nc_stchg];

% get class of controlled state prior to each non-nominal cut
prior_class = zeros(length(nc_stchg),1);
for i = 1:length(nc_stchg)
    prior_cut = find(peak_pos<=nc_stchg(i),1,'last');
    prior_class(i) = classes(prior_cut);
end

% merge non_controllable and controllable state changes into one vector
% and merge classes accordingly, additionally add nominality tag
stchg = [peak_pos; nc_stchg];
nominality = [ones(size(classes)); zeros(size(prior_class))];
classes = [classes; prior_class];
[stchg, stchg_idx] = sort(stchg);
nominality = nominality(stchg_idx);
classes = classes(stchg_idx);

% for each state change get states of non-controllable objects
nc_states = [mlp(stchg+1), olp(stchg+1), cche(stchg+1)];

```

```
% mark segments as nominality -1 with very small lengths, which occur due
% to cuts that are very close to each other
stchg_diff = diff(stchg);
idx = find(stchg_diff<20);
nominality(idx) = -1;

% recalculate class for each controllable and non-controllable state change
for i = 1:length(stchg)
    if isequal(nc_states(i,:), [0 0 1])
        classes(i) = classes(i)+12;
    elseif isequal(nc_states(i,:), [0 1 0])
        classes(i) = classes(i)+24;
    elseif isequal(nc_states(i,:), [0 1 1])
        classes(i) = classes(i)+36;
    elseif isequal(nc_states(i,:), [1 0 0])
        classes(i) = classes(i)+48;
    elseif isequal(nc_states(i,:), [1 0 1])
        classes(i) = classes(i)+60;
    elseif isequal(nc_states(i,:), [1 1 0])
        classes(i) = classes(i)+72;
    elseif isequal(nc_states(i,:), [1 1 1])
        classes(i) = classes(i)+84;
    end
end

%% plot found positions into data
if 1
    figure;
    clf;
    ax1 = subplot(3,1,1);
    plot(main_line_l1);
    hold on;
    plot(main_line_l2);
    plot(main_line_l3);
    hold on;
    title('DMU Main');
    xlabel('Data Sample');
    ylabel('Current I / A');
    legend('L2');
    grid on;
    axis tight;
    ylim([0, 60]);

    ax2 = subplot(3,1,2);
    plot(control_cabinet_heat_exchanger_l1);
    hold on;
    title('Control Cabinet Heat Exchanger');
    xlabel('Data Sample');
    ylabel('Current I / A');
    legend('L1');
    grid on;
    axis tight;
    ylim([0, 12]);

```

```
    ax3 = subplot(3,1,3);
    plot(blower_air_lubrication_pump_l1);
    hold on;
    plot(oil_air_lubrication_pump_l1);
    plot(machine_lubrication_pump_l1);
    title('Lubrication Pumps');
    xlabel('Data Sample');
    ylabel('Current I / A');
    legend('Blower Air','Oil-Air','Machine');
    grid on;
    axis tight;
    ylim([0, 0.5]);

    % plot cut positions
    for i = 1:length(stchg)
        if nominality(i) == 1
            plot(ax1,[stchg(i), stchg(i)],[0, 60],'r:');
        elseif nominality(i) == 0
            plot(ax1,[stchg(i), stchg(i)],[0, 60],'g-');
            plot(ax2,[stchg(i), stchg(i)],[0, 60],'g-');
            plot(ax3,[stchg(i), stchg(i)],[0, 60],'g-');
        else
            plot(ax1,[stchg(i), stchg(i)],[0, 60],'k*');
            plot(ax2,[stchg(i), stchg(i)],[0, 60],'k*');
            plot(ax3,[stchg(i), stchg(i)],[0, 60],'k*');
        end
    end

    linkaxes([ax1, ax2, ax3],'x');

    for i = 1:length(stchg)
        text(ax1,stchg(i),10,num2str(classes(i)));
    end
end

%% create labelset csv file containing all cuts
labelset = [stchg nominality classes];
dlmwrite('Labelsets/labelset.csv',labelset,'precision',10);
```

A.7.4 Template Averaging Algorithm

Listing A.8: Matlab source code of the template creation via accurate shape averaging.

```
%% Script for template creation via ASA shape averaging
% Lukas Bommes
% IWF TU Braunschweig
% 18.05.2017

%% select option for lower bounding of dtw
enable_lower_bounding = 1; % 1 - enable, 0 - disable

%% load labeled training data from second testcycle run
[~,main_line_l2_2,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_2');
labelset = csvread('Labelsets/labelset_nominal_run2.csv');
cuts = labelset(:,1);
class_labels = labelset(:,3);

%% loop through all 12 classes of templates
ts_avg = cell(12,1);
xq = cell(12,1);
ts_copy = cell(12,1);
for t_class = 1:12
    %% find all cut indices for specific kind of template
    % idle class
    if t_class == 12
        t_ind = find(class_labels == 0,100);
        t_cuts = cuts(t_ind(1:end-1));
        t_cuts_end = cuts(t_ind(1:end-1)+1);
        % remove sequences at beginning, end and in between of subcycles
        t_cuts(1) = [];
        t_cuts_end(1) = [];
        t_cuts(55) = [];
        t_cuts_end(55) = [];
        t_cuts(end) = [];
        t_cuts_end(end) = [];
    % all other classes
    else
        t_ind = find(class_labels == t_class);
        t_cuts = cuts(t_ind);
        t_cuts_end = cuts(t_ind+1);
    end

    %% store according sequences in cell array
    len = length(t_cuts);
    ts = cell(len, 1);
    for i = 1:len
        ts{i} = main_line_l2_2(t_cuts(i):t_cuts_end(i));
    end

    %% for idle class shorten all segments to length of smallest segment
    if t_class == 12
        len = zeros(length(ts),1);
```

```
50          for i = 1:length(ts)
51              len(i) = min(length(ts{i}));
52          end
53          min_len = min(len);
54          for i = 1:length(ts)
55              ts_new = ts{i};
56              ts{i} = ts_new(1:min_len);
57          end
58      end
59
60      % make a copy for later plotting
61      ts_copy{t_class} = ts;
62
63      % start timer
64      tic
65
66      %% run this loop until there is only one final averaged sequence left
67      len_ts = length(ts);
68      while(len_ts > 1)
69          %% calculate dtw distance matrix between each pair of segments
70          d_prior = inf(len_ts);
71          for i = 1:len_ts
72              for j = 1:i
73                  [d_prior(j,i),~,~] = dtw(ts{i},ts{j});
74              end
75          end
76
77          %% find pair of segments with minimum distance within distance matrix
78          % replace diagonal elements with inf
79          d_prior(d_prior == 0) = inf;
80          % then find location of minimum value
81          [min_dist,idx] = min(d_prior(:));
82          [opt_row,opt_col] = ind2sub(size(d_prior),idx);
83
84          %% calculate average sequence for this pair of segments via dtw
85          ts1 = ts{opt_row};
86          ts2 = ts{opt_col};
87
88          % calculate sequence alignment via dtw
89          if enable_lower_bounding
90              [d,its1,its2] = dtw(ts1,ts2,10);
91          else
92              [d,its1,its2] = dtw(ts1,ts2);
93          end
94
95          % calculate average sequence
96          S = [((its1+its2)./2)', ((ts1(its1)+ts2(its2))./2)];
97
98          % resample averaged sequence by cubic spline interploation
99          xq{t_class} = 1:length(ts1);
100         S_res = spline(S(:,1),S(:,2),xq{t_class});
101
102         %% remove pair of segments from segment array and insert average
103         ts{opt_row} = S_res';
```

```
104         ts(opt_col) = [];
105 
106         % update length of segment array
107         len_ts = len_ts - 1;
108     end
109 
110     %% store calculated average sequence for later analysis
111     ts_avg{t_class} = ts{1};
112 
113     % output time
114     toc
115 end
116 
117 %% store results in csv files
118 for k = 1:length(ts_avg)
119     if enable_lower_bounding
120         filename = ['AverageTemplates/Template_Class_LB_',num2str(k),'.csv'];
121     else
122         filename = ['AverageTemplates/Template_Class_',num2str(k),'.csv'];
123     end
124     csvwrite(filename,ts_avg{k});
125 end
126 
127 %% store sequences for later plotting in mat file
128  if enable_lower_bounding
129     filename = 'AverageTemplates/Variables_LB.mat';
130 else
131     filename = 'AverageTemplates/Variables.mat';
132 end
133 save(filename,'ts_copy','xq','ts_avg');
134 
135 %% plot results
136 for k = 1:length(ts_avg)
137     figure;
138     clf;
139     for i = 1:length(ts_copy{k})
140         plot(ts_copy{k}{i})
141         hold on;
142     end
143     plot(xq{k},ts_avg{k},'Marker','o','MarkerSize',12,'Color','g');
144     title('ASA Averaged Segment');
145     xlabel('Data Sample');
146     ylabel('Current I / A');
147     grid on;
148 end
```

A.7.5 Shape-Based Template Matching Algorithm

Listing A.9: Matlab source code of the shape-based template matching algorithm.

```
%% Script for shape-based template matching
% Lukas Bommes
% IWF TU Braunschweig
% 16.04.2017
% !!Run N01_Plotting.m first!!

%% load averaged templates
c = cell(12,1);
for k = 1:12
    filename = ['AverageTemplates/Template_Class_LB_',num2str(k),'.csv'];
    c{k} = csvread(filename);
end

%% load labelset of test data (testcycle run 1) to get cut positions
labelset = csvread('Labelsets/labelset_nominal_run1.csv');
cuts = labelset(:,1);
desired_classes = labelset(:,3);

%% For each segment decide which class template fits best
len = length(cuts)-1;
%len = 111;
predicted_classes = zeros(len,7);
% loop through samples
for i = 1:len
    % get sample
    ts = main_line_l2(cuts(i):cuts(i+1));
    % loop through class templates
    coef = zeros(length(c),1);
    d = zeros(length(c),1);
    infnorm = zeros(length(c),1);
    for j = 1:length(c)
        ct = c{j};
        % remove signal mean
        ts = ts - mean(ts);
        ct = ct - mean(ct);
        % align signals and calculate distance via DTW
        [d(j),its,ict] = dtw(ts,ct,1,'squared');
        tsw = ts(its);
        ctw = ct(ict);
        % calculate cross correlation coefficient
        coef1 = corrcoef(tsw,ctw);
        coef(j) = coef1(2,1);
    end

    % find template with maximum fit
    [maxcoef,predicted_class_xcor] = max(coef);
    [maxd,predicted_class_dtw] = min(d);
    % change label of idle class from 12 to 0 to meet class definition
    if predicted_class_xcor == 12
```

```
        predicted_class_xcor = 0;
    end
    if predicted_class_dtw == 12
        predicted_class_dtw = 0;
    end
    % create output dataset
    predicted_classes(i,:) = [i, cuts(i), cuts(i+1), maxcoef,...
        predicted_class_xcor, maxd, predicted_class_dtw];
end

%% calculate score
score_dtw = 0;
score_xcor = 0;
for i = 1:length(predicted_classes)
    if predicted_classes(i,7) == desired_classes(i)
        score_dtw = score_dtw+1;
    end
    if predicted_classes(i,5) == desired_classes(i)
        score_xcor = score_xcor+1;
    end
end
score_dtw = score_dtw/length(predicted_classes);
score_xcor = score_xcor/length(predicted_classes);
disp(['Score for DTW classifier is: ',...
    num2str(score_dtw*100), ' %']);
disp(['Score for cross corellation classifier is: ',...
    num2str(score_xcor*100), ' %']);

%% write output to CSV file
filename = 'PredictionOutput/predicted_classes.csv';
dlmwrite(filename,predicted_classes,'precision',10);
```

A.7.6 Parameter Studies for the Template Matching Algorithm

Listing A.10: Matlab source code of the parameter studies for the template matching algorithm.

```
%% Script for shape-based template matching
% Lukas Bommes
% IWF TU Braunschweig
% 16.04.2017
% !!Run N01_Plotting.m first!!

%% load averaged templates
c = cell(12,1);
for k = 1:12
    filename = ['AverageTemplates/Template_Class_LB_',num2str(k),'.csv'];
    c{k} = csvread(filename);
end

%% load labelset of test data (testcycle run 1) to get cut positions
labelset = csvread('Labelsets/labelset_nominal_run1.csv');
cuts = labelset(:,1);
desired_classes = labelset(:,3);

%% Iterate over different kinds of parameters
score_dtw = zeros(3,3,2);
score_xcor = zeros(3,3,2);
for p1 = 1:2 % with/without removal of signal mean
    for p2 = 1:3 % distance measure
        for p3 = 1:60 % width of Sakoe-Chiba band to contraint warping path
            %% For each segment decide which class template fits best
            len = length(cuts)-1;
            %len = 111;
            predicted_classes = zeros(len,7);
            % loop through samples
            for i = 1:len
                % get sample
                ts = main_line_l2(cuts(i):cuts(i+1));
                % loop through class templates
                coef = zeros(length(c),1);
                d = zeros(length(c),1);
                infnorm = zeros(length(c),1);
                for j = 1:length(c)
                    ct = c{j};
                    % remove signal mean
                    if (p1 == 1)  (p2 ~= 3)
                        ts = ts - mean(ts);
                        ct = ct - mean(ct);
                    end
                    % align signals and calculate distance via DTW
                    switch(p2)
                        case 1 % euclidean
                            [d(j),its,ict] = dtw(ts,ct,p3,'euclidean');
                        case 2 % squared
                            [d(j),its,ict] = dtw(ts,ct,p3,'squared');
```

```
                        case 3 % symkl
                            [d(j),its,ict] = dtw(ts,ct,p3,'symmkl');
                    end
                    tsw = ts(its);
                    ctw = ct(ict);
                    % calculate cross correlation coefficient
                    coef1 = corrcoef(tsw,ctw);
                    coef(j) = coef1(2,1);
                end

                % find template with maximum fit
                [maxcoef,predicted_class_xcor] = max(coef);
                [maxd,predicted_class_dtw] = min(d);
                % change label of idle class 12 to 0 to meet class definition
                if predicted_class_xcor == 12
                    predicted_class_xcor = 0;
                end
                if predicted_class_dtw == 12
                    predicted_class_dtw = 0;
                end
                % create output dataset
                predicted_classes(i,:) = [i, cuts(i), cuts(i+1), maxcoef,...
                    predicted_class_xcor, maxd, predicted_class_dtw];
            end

            %% calculate score
            score_dtw(p3,p2,p1) = 0;
            score_xcor(p3,p2,p1) = 0;
            for i = 1:length(predicted_classes)
                if predicted_classes(i,7) == desired_classes(i)
                    score_dtw(p3,p2,p1) = score_dtw(p3,p2,p1)+1;
                end
                if predicted_classes(i,5) == desired_classes(i)
                    score_xcor(p3,p2,p1) = score_xcor(p3,p2,p1)+1;
                end
            end
            score_dtw(p3,p2,p1) = score_dtw(p3,p2,p1)/length(predicted_classes);
            score_xcor(p3,p2,p1) = score_xcor(p3,p2,p1)/length(predicted_classes);
            disp(['mean (p1) = ', num2str(p1),...
                ', measure (p2) = ', num2str(p2),...
                ', constraint (p3) = ', num2str(p3)]);
            disp(['Score for DTW classifier is: ',...
                num2str(score_dtw(p3,p2,p1)*100), ' %']);
            disp(['Score for cross correlation classifier is: ',...
                num2str(score_xcor(p3,p2,p1)*100), ' %']);
        end
    end
end

%% plot results
if 1
    % DTW mean removed
    figure;
    clf;
```

```
104     plot(1:length(score_dtw(:,1,1)),score_dtw(:,1,1));
105     hold on;
106     plot(1:length(score_dtw(:,2,1)),score_dtw(:,2,1));
107     title('DTW Classifier (mean removed)');
108     xlabel('Sakoe-Chiba Band Width b');
109     ylabel('Classification Accuracy \eta');
110     legend('euclidean','squared');
111     grid on;
112     axis tight;
113 
114     % DTW mean not removed
115     figure;
116     clf;
117     plot(1:length(score_dtw(:,1,2)),score_dtw(:,1,2));
118     hold on;
119     plot(1:length(score_dtw(:,2,2)),score_dtw(:,2,2));
120     plot(1:length(score_dtw(:,3,2)),score_dtw(:,3,2));
121     title('DTW Classifier (mean not removed)');
122     xlabel('Sakoe-Chiba Band Width b');
123     ylabel('Classification Accuracy \eta');
124     legend('euclidean','squared','symmetric kullback-leibler');
125     grid on;
126     axis tight;
127 
128     % Cross-Correlation mean removed
129     figure;
130     clf;
131     plot(1:length(score_xcor(:,1,1)),score_xcor(:,1,1));
132     hold on;
133     plot(1:length(score_xcor(:,2,1)),score_xcor(:,2,1));
134     title('Cross-Correlation Classifier (mean removed)');
135     xlabel('Sakoe-Chiba Band Width b');
136     ylabel('Classification Accuracy \eta');
137     legend('euclidean','squared');
138     grid on;
139     axis tight;
140 
141     % Cross-Correlation mean not removed
142     figure;
143     clf;
144     plot(1:length(score_xcor(:,1,2)),score_xcor(:,1,2));
145     hold on;
146     plot(1:length(score_xcor(:,2,2)),score_xcor(:,2,2));
147     plot(1:length(score_xcor(:,3,2)),score_xcor(:,3,2));
148     title('Cross-Correlation Classifier (mean not removed)');
149     xlabel('Sakoe-Chiba Band Width b');
150     ylabel('Classification Accuracy \eta');
151     legend('euclidean','squared','symmetric kullback-leibler');
152     grid on;
153     axis tight;
154 end
155 
156 %% save important variables to workspace
157 filename = 'PredictionOutput/score_vars/scores.mat';
```

```
158 save(filename,'score_dtw','score_xcor');
159
160 %% write output files for parameter studies
161 filename1 = 'PredictionOutput/ps/score_dtw_with_mean.csv';
162 filename2 = 'PredictionOutput/ps/score_xcor_with_mean.csv';
163 filename3 = 'PredictionOutput/ps/score_dtw_without_mean.csv';
164 filename4 = 'PredictionOutput/ps/score_xcor_without_mean.csv';
165
166 csvwrite(filename1,score_dtw(:,:,1));
167 csvwrite(filename2,score_xcor(:,:,1));
168 csvwrite(filename3,score_dtw(:,:,2));
169 csvwrite(filename4,score_xcor(:,:,2));
```

A.7.7 Feature Extraction Algorithm

Listing A.11: Matlab source code of the feature extraction algorithm.

```
%% Script for feature extraction from measurement data (simplified task)
% Lukas Bommes
% IWF TU Braunschweig
% 07.06.2017
clear all, clc, close all;

%% load measurement data
[~,main_line_l2_run1,~,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_1');
[~,main_line_l2_run2,~,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_2');
[~,main_line_l2_run3,~,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_3');

%% load labels and cut positions from labelsets
labelset_run1 = csvread('Labelsets/labelset_nominal_run1.csv');
labelset_run2 = csvread('Labelsets/labelset_nominal_run2.csv');
labelset_run3 = csvread('Labelsets/labelset_nominal_run3.csv');
cuts_run1 = labelset_run1(:,1);
cuts_run2 = labelset_run2(:,1);
cuts_run3 = labelset_run3(:,1);
desired_classes_run1 = labelset_run1(:,3);
desired_classes_run2 = labelset_run2(:,3);
desired_classes_run3 = labelset_run3(:,3);

%% create cell arrays containing all segments for each testcycle run
cuts_main_line_l2_run1 = cell(length(cuts_run1)-1,1);
cuts_main_line_l2_run2 = cell(length(cuts_run2)-1,1);
cuts_main_line_l2_run3 = cell(length(cuts_run3)-1,1);
for i = 1:length(cuts_run1)-1
    cuts_main_line_l2_run1(i,1) = ...
        {main_line_l2_run1(cuts_run1(i):cuts_run1(i+1))'};
end
for i = 1:length(cuts_run2)-1
    cuts_main_line_l2_run2(i,1) = ...
        {main_line_l2_run2(cuts_run2(i):cuts_run2(i+1))'};
end
for i = 1:length(cuts_run3)-1
    cuts_main_line_l2_run3(i,1) = ...
        {main_line_l2_run3(cuts_run3(i):cuts_run3(i+1))'};
end

%% loop through each row of the cell arrays and calculate features
num_of_features = 49;
dataset_run1 = zeros(length(cuts_run1)-1,num_of_features);
dataset_run2 = zeros(length(cuts_run2)-1,num_of_features);
dataset_run3 = zeros(length(cuts_run3)-1,num_of_features);
for i = 1:length(cuts_run1)-1
        dataset_run1(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_run1{i});
end
for i = 1:length(cuts_run2)-1
```

```
        dataset_run2(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_run2{i});
end
for i = 1:length(cuts_run3)-1
        dataset_run3(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_run3{i});
end

%% replace every NaN, inf or -inf with 0
dataset_run1(isnan(dataset_run1)) = 0;
dataset_run1(isinf(dataset_run1)) = 0;
dataset_run2(isnan(dataset_run2)) = 0;
dataset_run2(isinf(dataset_run2)) = 0;
dataset_run3(isnan(dataset_run3)) = 0;
dataset_run3(isinf(dataset_run3)) = 0;

%% merge desired classes vectors into datasets
desired_classes_run1(end) = [];
desired_classes_run2(end) = [];
desired_classes_run3(end) = [];
dataset_run1 = [dataset_run1, desired_classes_run1];
dataset_run2 = [dataset_run2, desired_classes_run2];
dataset_run3 = [dataset_run3, desired_classes_run3];

%% write extracted datasets to csv and orange3 tab files
% csv files
dlmwrite('Extracted_Datasets/dataset_testing.csv',...
    dataset_run1,'precision',10);
dlmwrite('Extracted_Datasets/dataset_training.csv',...
    [dataset_run2; dataset_run3],'precision',10);

% special table format for orange3
header = {'C#q' 'C#sq' 'C#mu2' 'C#mu3' 'C#mu4' 'C#mu5' 'C#mu6'...
'C#mu7' 'C#mu8' 'C#mu9' 'C#c4' 'C#c5' 'C#c6' 'C#v' 'C#w' 'C#sigma'...
'C#srsd' 'C#siqr' 'C#smed' 'C#dmean' 'C#dmed' 'C#smin' 'C#smax'...
'C#sp2p' 'C#speak' 'C#ssrv' 'C#sarv' 'C#srms' 'C#srss' 'C#ks' 'C#kf'...
'C#hj2' 'C#hj3' 'C#auc' 'C#aac' 'C#sos' 'C#sus' 'C#srt' 'C#sft'...
'C#ssr' 'C#abp' 'C#hpbw' 'C#ocbw' 'C#fmean' 'C#fmed' 'C#ssfdr'...
'C#ssinad' 'C#ssnr' 'C#sthd' 'cD#Machine State'};

filename = 'Extracted_Datasets/dataset_testing.tab';
fid = fopen(filename,'w');
fprintf(fid, '%s\t', header{1:end-1});
fprintf(fid, '%s\n', header{end});
fclose(fid);
dlmwrite(filename,dataset_run1,'delimiter','\t','precision',10,'-append');

filename = 'Extracted_Datasets/dataset_training.tab';
fid = fopen(filename,'w');
fprintf(fid, '%s\t', header{1:end-1});
fprintf(fid, '%s\n', header{end});
fclose(fid);
dlmwrite(filename,[dataset_run2; dataset_run3],'delimiter','\t',...
    'precision',10,'-append');
```

Listing A.12: Matlab source code of the subroutine which conducts feature calculation for each segment of time series data.

```
function features = calc_features(time_signal)
% Calculate features from given Signal

%#### time domain ####
    % number of samples
    q = length(time_signal);
    % moment
    sq = mean(time_signal);
    % central moments
    mu2 = moment(time_signal,2);
    mu3 = moment(time_signal,3);
    mu4 = moment(time_signal,4);
    mu5 = moment(time_signal,5);
    mu6 = moment(time_signal,6);
    mu7 = moment(time_signal,7);
    mu8 = moment(time_signal,8);
    mu9 = moment(time_signal,9);
    % cumulants
    c4 = mu4-3*mu2^2;
    c5 = mu5-10*mu3*mu2;
    c6 = mu6-15*mu4*mu2-10*mu3^2+30*mu2^3;
    % skewness
    v = skewness(time_signal);
    % kurtosis
    w = kurtosis(time_signal);
    % standard deviation
    sigma = std(time_signal,1);
    % relative standard deviation
    srsd = std(time_signal,1)/mean(time_signal);
    % interquartile range
    siqr = iqr(time_signal);
    % median
    smed = median(time_signal);
    % mean absoute deviation
    dmean = mad(time_signal);
    % median absolute deviation
    dmed = mad(time_signal,1);
    % minimum value
    smin = min(time_signal);
    % maximum value
    smax = max(time_signal);
    % peak to peak value
    sp2p = peak2peak(time_signal);
    % maximum absolute value
    speak = max(abs(time_signal));
    % square root value
    ssrv = (1/length(time_signal)*sum(sqrt(abs(time_signal)))).^2;
    % average rectified value
    sarv = 1/length(time_signal)*sum(abs(time_signal));
```

```
    % root mean square
    srms = rms(time_signal);
    % root sum of squares
    srss = rssq(time_signal);
    % crest factor
    ks = speak/srms;
    % shape factor
    kf = srms/sarv;
    % 2nd and 3rd Hjorth Parameter
    [hj2, hj3] = HjorthParameters(time_signal');
    % area under curve
    auc = trapz(time_signal);
    % area under autocorrelation curve
    aac = trapz(xcorr(time_signal));
    % sum of overshoots
    sos = sum(overshoot(time_signal));
    % sum of undershoots
    sus = sum(undershoot(time_signal));
    % sum of rise times
    srt = sum(risetime(time_signal));
    % sum of fall time
    sft = sum(falltime(time_signal));
    % sum of slew rates
    ssr = sum(slewrate(time_signal));

%#### frequency domain ####
    % average bandpower
    abp = bandpower(time_signal);
    % 3dB half-power bandwidth
    hpbw = powerbw(time_signal);
    % 99% occupied bandwidth
    ocbw = obw(time_signal);
    % mean normalized frequency
    fmean = meanfreq(time_signal);
    % median normalized frequency
    fmed = meanfreq(time_signal);
    % spurious free dynamic range
    ssfdr = sfdr(time_signal);
    % singal to noise and distortion ratio
    ssinad = sinad(time_signal);
    % signal to noise ratio
    ssnr = snr(time_signal);
    % total harmonic distortion
    sthd = thd(time_signal);

%#### create feature vector ####
    features = [q sq mu2 mu3 mu4 mu5 mu6 mu7 mu8 mu9...
                c4 c5 c6 v w sigma srsd siqr smed dmean...
                dmed smin smax sp2p speak ssrv sarv...
                srms srss ks kf hj2 hj3 auc aac sos sus srt sft ssr...
                abp hpbw ocbw fmean fmed ssfdr ssinad ssnr sthd];
end
```

A.7.8 Test Set Modification Script

Listing A.13: Matlab source code of the test set modification script.

```
%% Script for modifying testsets to simulate machine faults
% Lukas Bommes
% IWF TU Braunschweig
% 27.06.2017
clear all, clc, close all;

%% load original test dataset
[~,main_line_l2_run1,~,~,~,~,~,~,~,~,~,~,~,~,~,~] = get_data('Data_Run_1');

%% load labels and cut positions from labelsets
labelset_run1 = csvread('Labelsets/labelset_nominal_run1.csv');
cuts_run1 = labelset_run1(:,1);
desired_classes_run1 = labelset_run1(:,3);

%% create cell arrays containing all segments of the testcycle run
cuts_main_line_l2_exp1 = cell(length(cuts_run1)-1,1);
for i = 1:length(cuts_run1)-1
    cuts_main_line_l2_exp1(i,1) = ...
        {main_line_l2_run1(cuts_run1(i):cuts_run1(i+1))'};
end
cuts_main_line_l2_exp2 = cuts_main_line_l2_exp1;
cuts_main_line_l2_exp3 = cuts_main_line_l2_exp1;

%% create testdataset 1: all instances of class 6 are modified
% find all instances of class 6
ind = desired_classes_run1==6;
instances_exp1 = cuts_main_line_l2_exp1(ind);
% add weighted offset onto segments
for i = 1:length(instances_exp1)
    % get offsets of first and last datapoint
    offset(1) = instances_exp1{i}(1);
    offset(2) = instances_exp1{i}(end);
    % interpolate between both offsets to create linear offset function
    len = length(instances_exp1{i});
    offset_func = interp1([1 len], offset, 1:len);
    % subtract offset function from original sequence
    instances_exp1{i} = instances_exp1{i} - offset_func;
    % multiply by factor
    instances_exp1{i} = 1.5*instances_exp1{i};
    % add offset function again
    instances_exp1{i} = instances_exp1{i} + offset_func;
end
% merge modified instances into test dataset
cuts_main_line_l2_exp1(ind) = instances_exp1;

%% create testdataset 2: all instances of class 6 and 7 are modified
% find all instances of class 6 and 7
ind = (desired_classes_run1==6) | (desired_classes_run1==7);
instances_exp2 = cuts_main_line_l2_exp2(ind);
```

```
% add weighted offset onto segments
for i = 1:length(instances_exp2)
    % get offsets of first and last datapoint
    offset(1) = instances_exp2{i}(1);
    offset(2) = instances_exp2{i}(end);
    % interpolate between both offsets to create linear offset function
    len = length(instances_exp2{i});
    offset_func = interp1([1 len], offset, 1:len);
    % subtract offset function from original sequence
    instances_exp2{i} = instances_exp2{i} - offset_func;
    % multiply by factor
    instances_exp2{i} = 1.5*instances_exp2{i};
    % add offset function again
    instances_exp2{i} = instances_exp2{i} + offset_func;
end
% merge modified instances into test dataset
cuts_main_line_l2_exp2(ind) = instances_exp2;

%% create testdataset 3: individual segments randomly modified
% set state of random number generator
rng(0);
% randomly select instances
ind = [datasample(find(desired_classes_run1==6),10); ...
    datasample(find(desired_classes_run1==7),5); ...
    datasample(find(desired_classes_run1==2),2)];
instances_exp3 = cuts_main_line_l2_exp3(ind);
% add weighted offset onto segments
for i = 1:length(instances_exp3)
    % get offsets of first and last datapoint
    offset(1) = instances_exp3{i}(1);
    offset(2) = instances_exp3{i}(end);
    % interpolate between both offsets to create linear offset function
    len = length(instances_exp3{i});
    offset_func = interp1([1 len], offset, 1:len);
    % subtract offset function from original sequence
    instances_exp3{i} = instances_exp3{i} - offset_func;
    % multiply by factor
    instances_exp3{i} = 1.5*instances_exp3{i};
    % add offset function again
    instances_exp3{i} = instances_exp3{i} + offset_func;
end
% merge modified instances into test dataset
cuts_main_line_l2_exp3(ind) = instances_exp3;
% modify one segment of class 1 to create an outlier within the outliers
segment_ind = 52;
offset(1) = cuts_main_line_l2_exp3{segment_ind}(1);
offset(2) = 15;
offset(3) = cuts_main_line_l2_exp3{segment_ind}(end);
len = length(cuts_main_line_l2_exp3{segment_ind});
linear_func = interp1([1 round(len/2) len], offset, 1:len);
cuts_main_line_l2_exp3{segment_ind} = linear_func;

%% sequencialize all cuts to view modified test datasets
main_line_l2_exp1 = zeros(length(main_line_l2_run1),1);
```

```
104 main_line_l2_exp2 = zeros(length(main_line_l2_run1),1);
105 main_line_l2_exp3 = zeros(length(main_line_l2_run1),1);
106 for i = 1:length(cuts_main_line_l2_exp1)-1
107     main_line_l2_exp1(cuts_run1(i):cuts_run1(i+1)) = ...
108        cuts_main_line_l2_exp1{i};
109 end
110 for i = 1:length(cuts_main_line_l2_exp2)-1
111     main_line_l2_exp2(cuts_run1(i):cuts_run1(i+1)) = ...
112        cuts_main_line_l2_exp2{i};
113 end
114 for i = 1:length(cuts_main_line_l2_exp3)-1
115     main_line_l2_exp3(cuts_run1(i):cuts_run1(i+1)) = ...
116        cuts_main_line_l2_exp3{i};
117 end
118
119 %% plot modified test dataset
120 % EXP 1
121 figure;
122 clf;
123 plot(main_line_l2_exp1);
124 hold on;
125 plot(main_line_l2_run1);
126 xlabel('Data Sample');
127 ylabel('Current I / A');
128 legend('Modified','Original');
129 title('EXP 1: Modified Test Dataset');
130 grid on;
131 axis tight;
132 % EXP 2
133 figure;
134 clf;
135 plot(main_line_l2_exp2);
136 hold on;
137 plot(main_line_l2_run1);
138 xlabel('Data Sample');
139 ylabel('Current I / A');
140 legend('Modified','Original');
141 title('EXP 2: Modified Test Dataset');
142 grid on;
143 axis tight;
144 % EXP 3
145 figure;
146 clf;
147 plot(main_line_l2_exp3);
148 hold on;
149 plot(main_line_l2_run1);
150 xlabel('Data Sample');
151 ylabel('Current I / A');
152 legend('Modified','Original');
153 title('EXP 3: Modified Test Dataset');
154 grid on;
155 axis tight;
156
157 %% calculate features for modified testcycles
```

```
num_of_features = 49;
dataset_exp1 = zeros(length(cuts_run1)-1,num_of_features);
dataset_exp2 = zeros(length(cuts_run1)-1,num_of_features);
dataset_exp3 = zeros(length(cuts_run1)-1,num_of_features);
for i = 1:length(cuts_run1)-1
        dataset_exp1(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_exp1{i});
    dataset_exp2(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_exp2{i});
    dataset_exp3(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_exp3{i});
end

%% replace every NaN, inf or -inf with 0
dataset_exp1(isnan(dataset_exp1)) = 0;
dataset_exp1(isinf(dataset_exp1)) = 0;
dataset_exp2(isnan(dataset_exp2)) = 0;
dataset_exp2(isinf(dataset_exp2)) = 0;
dataset_exp3(isnan(dataset_exp3)) = 0;
dataset_exp3(isinf(dataset_exp3)) = 0;

%% merge desired classes vectors into datasets
desired_classes_run1(end) = [];
dataset_exp1 = [dataset_exp1, desired_classes_run1];
dataset_exp2 = [dataset_exp2, desired_classes_run1];
dataset_exp3 = [dataset_exp3, desired_classes_run1];

%% write extracted datasets to csv and orange3 tab files
% csv files
dlmwrite('Modified_Testsets/dataset_testing_exp1.csv',...
    dataset_exp1,'precision',10);
dlmwrite('Modified_Testsets/dataset_testing_exp2.csv',...
    dataset_exp2,'precision',10);
dlmwrite('Modified_Testsets/dataset_testing_exp3.csv',...
    dataset_exp3,'precision',10);

% special table format for orange3
header = {'C#q' 'C#sq' 'C#mu2' 'C#mu3' 'C#mu4' 'C#mu5' 'C#mu6'...
'C#mu7' 'C#mu8' 'C#mu9' 'C#c4' 'C#c5' 'C#c6' 'C#v' 'C#w' 'C#sigma'...
'C#srsd' 'C#siqr' 'C#smed' 'C#dmean' 'C#dmed' 'C#smin' 'C#smax'...
'C#sp2p' 'C#speak' 'C#ssrv' 'C#sarv' 'C#srms' 'C#srss' 'C#ks' 'C#kf'...
'C#hj2' 'C#hj3' 'C#auc' 'C#aac' 'C#sos' 'C#sus' 'C#srt' 'C#sft'...
'C#ssr' 'C#abp' 'C#hpbw' 'C#ocbw' 'C#fmean' 'C#fmed' 'C#ssfdr'...
'C#ssinad' 'C#ssnr' 'C#sthd' 'cD#Machine State'};

filename = 'Modified_Testsets/dataset_testing_exp1.tab';
fid = fopen(filename,'w');
fprintf(fid, '%s\t', header{1:end-1});
fprintf(fid, '%s\n', header{end});
fclose(fid);
dlmwrite(filename,dataset_exp1,'delimiter','\t','precision',10,'-append');

filename = 'Modified_Testsets/dataset_testing_exp2.tab';
fid = fopen(filename,'w');
```

```
212 fprintf(fid, '%s\t', header{1:end-1});
213 fprintf(fid, '%s\n', header{end});
214 fclose(fid);
215 dlmwrite(filename,dataset_exp2,'delimiter','\t','precision',10,'-append');
216
217 filename = 'Modified_Testsets/dataset_testing_exp3.tab';
218 fid = fopen(filename,'w');
219 fprintf(fid, '%s\t', header{1:end-1});
220 fprintf(fid, '%s\n', header{end});
221 fclose(fid);
222 dlmwrite(filename,dataset_exp3,'delimiter','\t','precision',10,'-append');
```

A.7.9 Processing of the Experimentally Acquired Modified Test Data

A.7.9.1 Measurement Data Plotting Script

Listing A.14: Matlab source code of the plotting script for the modified test measurement data.

```
%% Script for plotting of measurement data (for modified testset)
% Lukas Bommes
% IWF TU Braunschweig
% 01.07.2017
clear all, clc, close all;
format short;

%% read csv files
[main_line_l1,...
main_line_l2,...
main_line_l3] = get_data_modified('Data_Run_1');

%% plot raw data
% DMU Main
fmain = figure;
clf;
plot(main_line_l1);
hold on;
plot(main_line_l2);
plot(main_line_l3);
title('DMU Main');
xlabel('Data Sample');
ylabel('Current I / A');
legend('L1','L2','L3');
grid on;
axis tight;
ylim([0, 60]);
```

Listing A.15: Matlab source code of the function used to read out the CSV files containing modified test measurement data.

```
function [main_line_l1,...
          main_line_l2,...
          main_line_l3] = get_data(dataset_subfolder)
main_line_l1 = csvread([dataset_subfolder,'/main_line_l1.csv']);
main_line_l2 = csvread([dataset_subfolder,'/main_line_l2.csv']);
main_line_l3 = csvread([dataset_subfolder,'/main_line_l3.csv']);

%% drop last samples (only one or two) of too long vector
lmin = min([length(main_line_l1(:,2)),...
           length(main_line_l2(:,2)),...
           length(main_line_l3(:,2))]);

main_line_l1 = main_line_l1(1:lmin,2);
main_line_l2 = main_line_l2(1:lmin,2);
main_line_l3 = main_line_l3(1:lmin,2);
```

```
16 end
```

A.7.9.2 Automatic Segmentation and Labeling Script

Listing A.16: Matlab source code of the automatic segmentation and labeling algorithm for the modified test measurement data.

```
%% Script for shape-based automatic cutting (for modified testset)
% Lukas Bommes
% IWF TU Braunschweig
% 01.07.2017
% !!Run N01_Plotting.m first!!

%% create shape dicitionary from first subcycle of second testcycle run
[~,main_line_l2_2,~] = get_data_modified('../Data_Run_2');
c{1} = main_line_l2_2(880:2020); % spindle
c{2} = main_line_l2_2(7084:7320); % x-axis forward
c{3} = main_line_l2_2(7423:7660); % x-axis backward
c{4} = main_line_l2_2(10465:10626); % y-axis forward
c{5} = main_line_l2_2(10727:10887); % y-axis backward
c{6} = main_line_l2_2(13088:13215); % z-axis forward
c{7} = main_line_l2_2(13305:13432); % z-axis backward
c{8} = main_line_l2_2(15212:15290); % c-axis forward
c{9} = main_line_l2_2(15374:15457); % c-axis backward
c{10} = main_line_l2_2(16850:16908); % b-axis forward
c{11} = main_line_l2_2(16992:17047); % b-axis backward

%% match templates to find positions
% number of subcycles in one testcycle (normally 20)
num_of_runs = 20;
% variable holds positions of cuts
peak_pos = 0;

% repeat for every template
for k = 1:11
    % Determine, how many peaks to find for the current type of template.
    if(ismember(k,[1,6,7,10,11]))
        num_of_peaks = 5*num_of_runs;
    else
        num_of_peaks = 10*num_of_runs;
    end

    % slide template indexwise over dataset_dmu_strom_l2
    % calculate for each window position a distance measure (e.g. inf norm)
    % find local minima in similarity measure over lag
    templ = c{k};
    tmplen = length(templ);
    len = length(main_line_l2)-tmplen;
    dist = zeros(len,1);
    for i = 1:len
        dist(i) = norm(main_line_l2(i:i+tmplen-1)-templ,Inf);
    end

    % invert distance to make it usable for findpeaks function
    dist = -dist;
```

```
49
50          % find peaks in the signal and append to peak_pos
51      len = nnz(peak_pos);
52          [~, pp] = findpeaks(dist,'SortStr','descend','NPeaks',num_of_peaks);
53
54      % combine cutting points and endpoints of templates to cutting vector
55      peak_pos(len+1:len+2*num_of_peaks,1) = [pp; pp+tmplen];
56
57      % create variable which contains class numbers, but combines classes,
58      % which can't be distiguished by the algorithm, to a single classs
59      switch(k)
60          case {1,6,7,10,11}
61              r = k;
62          case {2,3}
63              r = 2;
64          case {4,5}
65              r = 4;
66          case {8,9}
67              r = 8;
68      end
69
70      % append current class as well as endpoint class (0) to the segment
71      classes(len+1:len+2*num_of_peaks,1) = ...
72          [repmat(r,num_of_peaks,1); zeros(num_of_peaks,1)];
73  end
74
75  % sort peak_pos in ascending order and sort classes accordingly
76  [peak_pos, sortidx] = sort(peak_pos);
77  classes = classes(sortidx);
78
79  % check, if there are approximative duplicates in the peak vector and
80  % remove them. Approximative duplicates are values, that are closer than
81  % 50 samples to each other. Remove from classes vector accordingly.
82  peak_pos_diff = diff(peak_pos);
83  idx = find(peak_pos_diff<50);
84  peak_pos(idx) = [];
85  classes(idx) = [];
86
87  % search for classes 2, 4 and 8 and replace every second one by the
88  % corresponding class 3, 5 or 9 (backward movements of x-, y- and z-axis)
89  for i = 1:length(classes)-2
90      if isequal(classes(i:i+2),[2;0;2])
91          classes(i:i+2) = [2;0;3];
92      elseif isequal(classes(i:i+2),[4;0;4])
93          classes(i:i+2) = [4;0;5];
94      elseif isequal(classes(i:i+2),[8;0;8])
95          classes(i:i+2) = [8;0;9];
96      end
97  end
98
99  % add inital cut on first sample of the dataset, where the state of the
100 % corresponding first segment must be idle (0)
101 peak_pos = [1; peak_pos];
102 classes = [0; classes];
```

```
103
104 %% check if class sequence is nominal
105 seq = [1,0,1,0,1,0,1,0,1,0,2,0,3,0,2,0,3,0,2,0,3,0,2,0,3,0,2,0,3,0,...
106        4,0,5,0,4,0,5,0,4,0,5,0,4,0,5,0,4,0,5,0,6,0,7,0,6,0,7,0,6,0,...
107        7,0,6,0,7,0,6,0,7,0,8,0,9,0,8,0,9,0,8,0,9,0,8,0,9,0,8,0,9,0,...
108        10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0,10,0,11,0];
109 seq = repmat(seq,1,num_of_runs);
110 seq = [0, seq];
111 if isequal(seq',classes)
112     disp('Class sequence is valid!');
113 else
114     disp('Class sequence is invalid! Check manually for any faults.');
115 end
116
117 %% create labelset csv file containing nominal cuts and base classes
118 labelset = [peak_pos ones(size(peak_pos)) classes];
119 dlmwrite('Labelsets/labelset_nominal_run4.csv',labelset,'precision',10);
120
121 %% plot found positions into data
122 if 1
123     figure;
124     clf;
125     plot(main_line_l1);
126     hold on;
127     plot(main_line_l2);
128     plot(main_line_l3);
129     hold on;
130     title('DMU Main');
131     xlabel('Data Sample');
132     ylabel('Current I / A');
133     %legend('L2');
134     grid on;
135     axis tight;
136     ylim([0, 60]);
137
138     % plot cut positions
139     for i = 1:length(peak_pos)
140         plot([peak_pos(i), peak_pos(i)],[0, 60],'r:');
141     end
142
143     for i = 1:length(peak_pos)
144         text(peak_pos(i),10,num2str(classes(i)));
145     end
146 end
```

A.7.9.3 Feature Extraction Script

Listing A.17: Matlab source code of the feature extraction algorithm for the modified test measurement data.

```
%% Script for feature extraction (for modified testset)
% Lukas Bommes
% IWF TU Braunschweig
% 01.07.2017
clear all, clc, close all;

%% load measurement data
[~,main_line_l2_run4,~] = get_data_modified('Data_Run_1');

%% load labels and cut positions from labelsets
labelset_run4 = csvread('Labelsets/labelset_nominal_run4.csv');
cuts_run4 = labelset_run4(:,1);
desired_classes_run4 = labelset_run4(:,3);

%% create cell arrays containing all segments for each testcycle run
cuts_main_line_l2_run4 = cell(length(cuts_run4)-1,1);
for i = 1:length(cuts_run4)-1
    cuts_main_line_l2_run4(i,1) = ...
        {main_line_l2_run4(cuts_run4(i):cuts_run4(i+1))'};
end

%% loop through each row of the cell arrays and calculate features
num_of_features = 49;
dataset_run4 = zeros(length(cuts_run4)-1,num_of_features);
for i = 1:length(cuts_run4)-1
        dataset_run4(i,1:num_of_features) = ...
        calc_features(cuts_main_line_l2_run4{i});
end

%% replace every NaN, inf or -inf with 0
dataset_run4(isnan(dataset_run4)) = 0;
dataset_run4(isinf(dataset_run4)) = 0;

%% merge desired classes vectors into datasets
desired_classes_run4(end) = [];
dataset_run4 = [dataset_run4, desired_classes_run4];

%% write extracted datasets to csv and orange3 tab files
% csv files
dlmwrite('Modified_Testsets/dataset_testing_exp4.csv',...
    dataset_run4,'precision',10);

% special table format for orange3
header = {'C#q' 'C#sq' 'C#mu2' 'C#mu3' 'C#mu4' 'C#mu5' 'C#mu6'...
'C#mu7' 'C#mu8' 'C#mu9' 'C#c4' 'C#c5' 'C#c6' 'C#v' 'C#w' 'C#sigma'...
'C#srsd' 'C#siqr' 'C#smed' 'C#dmean' 'C#dmed' 'C#smin' 'C#smax'...
'C#sp2p' 'C#speak' 'C#ssrv' 'C#sarv' 'C#srms' 'C#srss' 'C#ks' 'C#kf'...
'C#hj2' 'C#hj3' 'C#auc' 'C#aac' 'C#sos' 'C#sus' 'C#srt' 'C#sft'...
```

```
'C#ssr' 'C#abp' 'C#hpbw' 'C#ocbw' 'C#fmean' 'C#fmed' 'C#ssfdr'...
'C#ssinad' 'C#ssnr' 'C#sthd' 'cD#Machine State'};

filename = 'Modified_Testsets/dataset_testing_exp4.tab';
fid = fopen(filename,'w');
fprintf(fid, '%s\t', header{1:end-1});
fprintf(fid, '%s\n', header{end});
fclose(fid);
dlmwrite(filename,dataset_run4,'delimiter','\t','precision',10,'-append');
```

A.8 Python Source Codes

A.8.1 Feature-Based Classification

A.8.1.1 Training and Hyperparameter Optimization

Listing A.18: Python source code of the feature-based classifier training and hyperparameter optimization.

```
"""
Script for Feature Based Classification
Technical University Brunswick
21/06/2017

@author: Lukas Bommes
"""

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectFromModel
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier, RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.externals import joblib

# define feature names
f_names = ['q', 'sq', 'mu2', 'mu3', 'mu4', 'mu5', 'mu6', 'mu7', 'mu8', 'mu9',
           'c4', 'c5', 'c6', 'v', 'w', 'sigma', 'srsd', 'siqr', 'smed',
           'dmean', 'dmed', 'smin', 'smax', 'sp2p', 'speak', 'ssrv', 'sarv',
           'srms', 'srss', 'ks', 'kf', 'hj2', 'hj3', 'auc', 'aac', 'sos',
           'sus', 'srt', 'sft', 'ssr', 'abp', 'hpbw', 'ocbw', 'fmean', 'fmed',
           'ssfdr', 'ssinad', 'ssnr', 'sthd','class']

# import training and testing dataset
dataset_training = pd.read_csv('dataset_training.csv', names=f_names)
dataset_testing = pd.read_csv('dataset_testing.csv', names=f_names)

# split datasets in target class vector and features
X_train = dataset_training.iloc[:,0:49]
X_test = dataset_testing.iloc[:,0:49]
y_train = dataset_training.iloc[:,49]
y_test = dataset_testing.iloc[:,49]

# center data around mean and then scale it to unit variance
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
44
45 # manual removal of inappropiate features
46 f_not_used = [0, 33, 44, 15, 16, 21, 22, 24]
47 X_train = np.delete(X_train, f_not_used, 1)
48 X_test = np.delete(X_test, f_not_used, 1)
49 for i in f_not_used:
50     del f_names[i]
51
52 # select 10 most relevant features by maximizing information gain
53 dt = DecisionTreeClassifier(criterion='entropy',
54                             min_samples_leaf=1,
55                             random_state=0)
56 sfm = SelectFromModel(dt)
57 sfm.fit(X_train, y_train)
58 X_train = sfm.transform(X_train)
59 X_test = sfm.transform(X_test)
60 selected_features = sfm.get_support(indices=True)
61 print('Selected Features: ' + str(list(f_names[i] for i in selected_features)))
62
63 # store preprossed test dataset
64 np.savetxt('X_test_preprocessed.csv', X_test, delimiter=",")
65 np.savetxt('y_test_preprocessed.csv', y_test, delimiter=",")
66
67 # initialise classifiers
68 classifiers = [KNeighborsClassifier(),
69                DecisionTreeClassifier(random_state=0),
70                RandomForestClassifier(random_state=0),
71                AdaBoostClassifier(random_state=0),
72                SVC(random_state=0, decision_function_shape='ovr'),
73                SGDClassifier(random_state=0),
74                GaussianNB(priors=None)]
75
76 # define classifier names
77 names = ['NearestNeighbours',
78          'DecisionTree',
79          'RandomForest',
80          'AdaBoost',
81          'SupportVectorMachine',
82          'StochasticGradientDescent',
83          'NaiveBayes']
84
85 # define parameter grids for classifier hyperparameters
86 parameters = [{'n_neighbors':np.arange(1, 20, 1), # KNN
87                'p':[1,2],
88                'weights':['uniform','distance']},
89               {'criterion':['gini','entropy'], # DT
90                'max_depth':[None] + list(np.arange(1, 100, 1)),
91                'min_samples_leaf':np.arange(1, 5, 1)},
92               {'n_estimators':np.arange(10, 100, 5), # RF
93                'criterion':['gini','entropy']},
94               {'n_estimators':np.arange(10, 100, 5), # AdaBoost
95                'learning_rate':np.arange(0.1, 1.0, 0.1)},
96               {'kernel':['linear','poly','rbf','sigmoid'],  # SVM
97                'C':np.arange(0.1, 1.0, 0.1),
```

```
98                'degree':[2,3,4,5]},
99               {'loss':['hinge','log','modified_huber', # SGD
100                        'squared_hinge', 'perceptron'],
101                'penalty':['none','l2','l1','elasticnet'],
102                'shuffle':[True, False]},
103               {}] # Bayes
104
105 # do an exhaustive grid search over all hyperparameters for each classifiers
106 for i, (name, clf) in enumerate(zip(names, classifiers)):
107
108     # fit to training data and do grid search
109     clf = GridSearchCV(clf, parameters[i], cv=3)
110     clf.fit(X_train, y_train)
111
112     # output score for classifiers with optimized hyperparameters
113     print(name + ': ' + str(clf.best_estimator_.score(X_test, y_test)*100)
114     + ' % (' + str(clf.best_params_) + ')')
115
116     # store classifier to binary files
117     filename = 'Models\\' + name + '.pkl'
118     joblib.dump(clf, filename)
```

A.8.1.2 Classifier Evaluation

Listing A.19: Python source code of the feature based classifier evaluation.

```
"""
Script for Evaluation of Feature Based Classsifiers
Technical University Brunswick
21/06/2017

@author: Lukas Bommes
"""

import numpy as np
from sklearn.externals import joblib
import sklearn.metrics as metric

# print always full arrays instead of truncated versions
np.set_printoptions(threshold=np.inf)

# import preprocessed datasets
X_test = np.genfromtxt('X_test_preprocessed.csv', delimiter=',')
y_test = np.genfromtxt('y_test_preprocessed.csv', delimiter=',')

# define classifier names
names = ['NearestNeighbours',
         'DecisionTree',
         'RandomForest',
         'AdaBoost',
         'SupportVectorMachine',
         'StochasticGradientDescent',
         'NaiveBayes']

# iterate over all classifiers
for i, name in enumerate(names):

    # load trained classsifiers
    filename = 'Models\\' + name + '.pkl'
    clf = joblib.load(filename)

    # calculate predictions on test set
    y_pred = clf.predict(X_test)

    # calculate confusion matrix
    clf_confusion_matrix = metric.confusion_matrix(y_test, y_pred)

    # compute different metrics
    clf_accuracy_score = metric.accuracy_score(y_test, y_pred)

    # create evaluation report file
    filename = 'Evaluation\\' + name + '_report.txt'
    f = open(filename, 'w')
    f.write('Confusion Matrix: \n')
    f.write(str(clf_confusion_matrix) + '\n\n')
```

```
50     f.write('Accuracacy Score: ' + str(clf_accuracy_score))
51     f.close()
```

A.8.2 Machine Fault Detection Algorithm

A.8.2.1 Main Script

Listing A.20: Python source code of the machine fault detection algorithm.

```
"""
Script for Machine Fault Detection
Technical University Brunswick
21/06/2017

@author: Lukas Bommes
"""

################################################################################
#
#    Import packages
#
################################################################################

import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
import LocalOutlierFactor

# print always full arrays instead of truncated versions
np.set_printoptions(threshold=np.inf)

################################################################################
#
#    Load and scale datasets, calculate centroids
#
################################################################################

# define feature names
f_names = ['q', 'sq', 'mu2', 'mu3', 'mu4', 'mu5', 'mu6', 'mu7', 'mu8', 'mu9',
           'c4', 'c5', 'c6', 'v', 'w', 'sigma', 'srsd', 'siqr', 'smed',
           'dmean', 'dmed', 'smin', 'smax', 'sp2p', 'speak', 'ssrv', 'sarv',
           'srms', 'srss', 'ks', 'kf', 'hj2', 'hj3', 'auc', 'aac', 'sos',
           'sus', 'srt', 'sft', 'ssr', 'abp', 'hpbw', 'ocbw', 'fmean', 'fmed',
           'ssfdr', 'ssinad', 'ssnr', 'sthd','class']

# import training and test datasets
dataset_training = pd.read_csv('dataset_training.csv', names=f_names)
dataset_testing_exp1 = pd.read_csv('dataset_testing_exp1.csv', names=f_names)
dataset_testing_exp2 = pd.read_csv('dataset_testing_exp2.csv', names=f_names)
dataset_testing_exp3 = pd.read_csv('dataset_testing_exp3.csv', names=f_names)
dataset_testing_exp4 = pd.read_csv('dataset_testing_exp4.csv', names=f_names)

# split datasets in target class vector and features
X_train = dataset_training.iloc[:,0:49]
y_train = dataset_training.iloc[:,49]
```

```
X_test_exp1 = dataset_testing_exp1.iloc[:,0:49]
y_test_exp1 = dataset_testing_exp1.iloc[:,49]
X_test_exp2 = dataset_testing_exp2.iloc[:,0:49]
y_test_exp2 = dataset_testing_exp2.iloc[:,49]
X_test_exp3 = dataset_testing_exp3.iloc[:,0:49]
y_test_exp3 = dataset_testing_exp3.iloc[:,49]
X_test_exp4 = dataset_testing_exp4.iloc[:,0:49]
y_test_exp4 = dataset_testing_exp4.iloc[:,49]

# center data around mean and then scale it to unit variance
scaler = StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test_exp1 = scaler.transform(X_test_exp1)
X_test_exp2 = scaler.transform(X_test_exp2)
X_test_exp3 = scaler.transform(X_test_exp3)
X_test_exp4 = scaler.transform(X_test_exp4)

# select relevant features
selected_features = np.array([9,7,8,4,23,43,19,15,29,30])
X_train = X_train[:, selected_features]
X_test_exp1 = X_test_exp1[:, selected_features]
X_test_exp2 = X_test_exp2[:, selected_features]
X_test_exp3 = X_test_exp3[:, selected_features]
X_test_exp4 = X_test_exp4[:, selected_features]

# calculate centroid (center) for every class cluster
centroids_train = np.zeros((12,10))
centroids_test_exp1 = np.zeros((12,10))
centroids_test_exp2 = np.zeros((12,10))
centroids_test_exp3 = np.zeros((12,10))
centroids_test_exp4 = np.zeros((12,10))
for i in range(12):
    class_samples_train = X_train[y_train==i,:]
    class_samples_test_exp1 = X_test_exp1[y_test_exp1==i,:]
    class_samples_test_exp2 = X_test_exp2[y_test_exp2==i,:]
    class_samples_test_exp3 = X_test_exp3[y_test_exp3==i,:]
    class_samples_test_exp4 = X_test_exp4[y_test_exp4==i,:]
    centroids_train[i,:] = np.mean(class_samples_train, axis=0)
    centroids_test_exp1[i,:] = np.mean(class_samples_test_exp1, axis=0)
    centroids_test_exp2[i,:] = np.mean(class_samples_test_exp2, axis=0)
    centroids_test_exp3[i,:] = np.mean(class_samples_test_exp3, axis=0)
    centroids_test_exp4[i,:] = np.mean(class_samples_test_exp4, axis=0)

##############################################################################
#
#    EXP 1: Detect shift of cluster centroid of class 6
#
##############################################################################

# calculate distance between centroids of training data and test data
dist = np.zeros((12,1))
for i in range(12):
    dist[i] = np.linalg.norm(centroids_test_exp1[i,:] - centroids_train[i,:])

```

```
# compare distances with threshold value and print result
distance_threshold = 0.2
print('\n----------------------------------------\nReport of experiment 1:')
for i, dist in enumerate(dist):
    if dist > distance_threshold:
        print('\nFault detected in class ' + str(i) + '.' +
              '\nCentroid shifted by ' + str(dist[0]) +
              ' units.\n(threshold = ' +
               str(distance_threshold) + ' units)')
print('----------------------------------------')

###############################################################################
#
#    EXP 2: Detect shift of cluster centroids of classes 6 and 7
#
###############################################################################

# calculate distance between centroids of training data and test data
dist = np.zeros((12,1))
for i in range(12):
    dist[i] = np.linalg.norm(centroids_test_exp2[i,:] - centroids_train[i,:])

# compare distances with threshold value and print result
distance_threshold = 0.2
print('\n----------------------------------------\nReport of experiment 2:')
for i, dist in enumerate(dist):
    if dist > distance_threshold:
        print('\nFault detected in class ' + str(i) + '.' +
              '\nCentroid shifted by ' + str(dist[0]) +
              ' units.\n(threshold = ' +
               str(distance_threshold) + ' units)')
print('----------------------------------------')

###############################################################################
#
#    EXP 3: Detect individual outliers of classes 2, 6 and 7
#
###############################################################################

# parameters for experiment 3
lof_n_neighbours = 20 # neighbours to consider when calculating overall lof
lof_threshold = 2000 # lof value above which point is labeled as outlier
outlier_subset_size = 5 # outliers per class above which fault is assumed
outlier_subset_lof_threshold = 200 # threshold for finding outliers in subset

# function for plotting the report
def print_report(obj):
    if obj == 'header':
        print('\n----------------------------------------' +
              '\nReport of experiment 3:')
    elif obj == 'footer':
        print('----------------------------------------')
    elif obj == 'systematic_fault':
        print('\nFault detected in class ' + str(i) + '.' +
```

```
       '\nThis class contains ' + str(outlier_subset_count) + ' outliers.')
        print('(threshold = ' + str(outlier_subset_size) + ' outliers)')
        print(str(outlier_subset_count_updated) + ' outliers are' +
              ' clustered and ' + str(outlier_subset_count -
              outlier_subset_count_updated) + ' are random noise.')
    elif obj == 'random_fault':
        print('\nClass ' + str(i) + ' contains ' + str(outlier_subset_count) +
          ' outliers.\nThis is a low number, possibly it is just noise.')

# calculate lof for all samples in the dataset
lof = LocalOutlierFactor.lof(X_test_exp3, lof_n_neighbours)

# label outliers (points with lof bigger than threshold)
outliers = np.where(lof > lof_threshold)[0]

# get outlier classes and count outliers per class
outlier_classes = np.array(y_test_exp3[outliers])

# create dict containing numbers and indices of outliers for each class
outlier_index = dict((i, [outliers[np.where(outlier_classes == i)],
                     np.shape(outliers[np.where(outlier_classes == i)])[0]])
                     for i in range(12))

print_report('header')

# loop through all classes and find outliers within the subset of outliers
for i in range(12):
    outlier_subset_count = outlier_index[i][1]
    # check if there are enough outliers per class to preclude a random error
    if outlier_subset_count < outlier_subset_size:
        if outlier_subset_count > 0:
            print_report('random_fault')
    else:
        # identify interclass outliers and check number of outliers again
        outlier_subset_index = outlier_index[i][0]
        outlier_subset_data = X_test_exp3[outlier_subset_index]
        outlier_subset_lof = LocalOutlierFactor.lof(outlier_subset_data, 4)
        interclass_outliers = np.where(outlier_subset_lof >
                                       outlier_subset_lof_threshold)[0]
        # remove number of interclass outliers from total number of outliers
        outlier_subset_count_updated = (outlier_subset_count -
                                        np.shape(interclass_outliers)[0])
        # check again, if enough outliers are left to assume systematic fault
        if outlier_subset_count_updated < outlier_subset_size:
            print_report('random_fault')
        else:
            print_report('systematic_fault')

print_report('footer')

###############################################################################
#
#    EXP 4: Detect faults in real measurement data
#
```

```
210 ##############################################################################
211
212 # calculate distance between centroids of training data and test data
213 dist = np.zeros((12,1))
214 for i in range(12):
215     dist[i] = np.linalg.norm(centroids_test_exp4[i,:] - centroids_train[i,:])
216
217 # compare distances with threshold value and print result
218 distance_threshold = 0.2
219 print('\n----------------------------------------\nReport of experiment 4:')
220 for i, dist in enumerate(dist):
221     if dist > distance_threshold:
222         print('\nFault detected in class ' + str(i) + '.' +
223               '\nCentroid shifted by ' + str(dist[0]) +
224               ' units.\n(threshold = ' +
225                str(distance_threshold) + ' units)')
226 print('----------------------------------------')
227
228 ##############################################################################
229 #
230 #    Create CSV files containing neccessary data for plotting in MATLAB
231 #
232 ##############################################################################
233
234 # Preprocessed Training and test Dataset
235 np.savetxt('plot\\X_train.csv', X_train, delimiter=",")
236 np.savetxt('plot\\X_test_exp1.csv', X_test_exp1, delimiter=",")
237 np.savetxt('plot\\X_test_exp2.csv', X_test_exp2, delimiter=",")
238 np.savetxt('plot\\X_test_exp3.csv', X_test_exp3, delimiter=",")
239 np.savetxt('plot\\X_test_exp4.csv', X_test_exp4, delimiter=",")
240
241 # Centroids
242 np.savetxt('plot\\centroids_train.csv', centroids_train, delimiter=",")
243 np.savetxt('plot\\centroids_exp1.csv', centroids_test_exp1, delimiter=",")
244 np.savetxt('plot\\centroids_exp2.csv', centroids_test_exp2, delimiter=",")
245 np.savetxt('plot\\centroids_exp3.csv', centroids_test_exp3, delimiter=",")
246 np.savetxt('plot\\centroids_exp4.csv', centroids_test_exp4, delimiter=",")
247
248 # Outliers
249 np.savetxt('plot\\outliers.csv', outliers, delimiter=",")
250
251
252 ##############################################################################
253 #
254 #    Plot results
255 #
256 ##############################################################################
257
258 # create axis with three subplots
259 f, ax = plt.subplots(2, 2)
260 ax[0, 0].set_title('Experiment 1')
261 ax[0, 1].set_title('Experiment 2')
262 ax[1, 0].set_title('Experiment 3')
263 ax[1, 1].set_title('Experiment 4')
```

```
264
265 # define 12 different colors for distiguishing classes of the training set
266 colors = np.zeros(np.shape(X_train)[0])
267 for i in range(12):
268     colors[y_train==i] = i/12
269
270 # plot training set
271 ax[0, 0].scatter(X_train[:,6], X_train[:,4], c=colors, cmap='hsv')
272 ax[0, 1].scatter(X_train[:,6], X_train[:,4], c=colors, cmap='hsv')
273 ax[1, 0].scatter(X_train[:,6], X_train[:,4], c=colors, cmap='hsv')
274 ax[1, 1].scatter(X_train[:,6], X_train[:,4], c=colors, cmap='hsv')
275
276 # plot test set
277 ax[0, 0].scatter(X_test_exp1[:,6], X_test_exp1[:,4], c='k', alpha=0.4)
278 ax[0, 1].scatter(X_test_exp2[:,6], X_test_exp2[:,4], c='k', alpha=0.4)
279 ax[1, 0].scatter(X_test_exp3[:,6], X_test_exp3[:,4], c='k', alpha=0.4)
280 ax[1, 1].scatter(X_test_exp4[:,6], X_test_exp4[:,4], c='k', alpha=0.4)
281
282 # plot centroids of experiment 1, 2 and 4 as pink dots
283 ax[0, 0].scatter(centroids_test_exp1[:,6], centroids_test_exp1[:,4], c='pink')
284 ax[0, 1].scatter(centroids_test_exp2[:,6], centroids_test_exp2[:,4], c='pink')
285 ax[1, 1].scatter(centroids_test_exp4[:,6], centroids_test_exp4[:,4], c='pink')
286
287 # plot outliers of experiment 3 as orange points
288 ax[1, 0].scatter(X_test_exp3[outliers,6], X_test_exp3[outliers,4], c='orange')
289 plt.tight_layout()
290 plt.draw()
```

A.8.2.2 Function for Computing Local Outlier Factor

Listing A.21: Python source code of the function computing the local outlier factor.

```
"""
Script for Calculation of Local Outlier Factor
Technical University Brunswick
29/06/2017

@author: Lukas Bommes
"""

# import packages
import numpy as np
from sklearn.neighbors import NearestNeighbors

def lof(data, k):
    # get neighbourhoods of all points
    neigh = NearestNeighbors(n_neighbors=k+1, p=1)
    neigh.fit(data)
    (dist, neighbours) = neigh.kneighbors(data)
    (dist, neighbours) = (dist[:,1:], neighbours[:,1:])
    kdist = np.amax(dist, axis=1)

    # calculate local reachability distances for all points
    rdist = np.zeros([k, np.shape(data[:,0])[0]])
    for o in range(np.shape(data[:,0])[0]):
        neighbours_o = neighbours[o,:]
        distances = np.array([kdist[neighbours_o], dist[o,:]])
        rdist[:,o] = np.amax(distances, axis=0)

    # calculate local reachability densities of all points
    lrd = np.divide(k, np.sum(rdist, axis=0))

    # calculate local outlier factors for all points
    lof = np.multiply(np.sum(lrd[neighbours], axis=1), np.sum(rdist, axis=0))

    return lof
```

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references. I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here. This paper was not previously presented to another examination board and has not been published. [191]

Singapore, 17th Juli 2017

(Lukas Bommes)