Data-Driven Machine Learning and Optimization Pipelines for Real-World Applications

Proefschrift

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van Rector Magnificus prof.mr. C.J.J.M. Stolker,
volgens besluit van het College voor Promoties
te verdedigen op dinsdag 1 september 2020
klokke 10.00 uur

door

Milan Koch

geboren te Ostercappeln, Duitsland
in 1990

Figures and diagrams are generated using PGF/TIKZ, INKSCAPE, MATPLOTLIB and NETRON.

Contents

List of Symbols

$\mathbf{x}$	Data points
N	Number of data points
$\mathbf{X}$	Time series
L	Length of time series
d	Dimension of time series
$\boldsymbol{\Phi}$	Features extracted from time series
$\mathcal{F}$	Feature function
$\mathcal{A}$	Machine learning algorithm
Θ	Configuration space
$\mathcal{H}$	Algorithm configurator
f	Performance metric
N_{cv}	Number of folds in cross-validation
Y	True target label
$\hat{Y}$	Predicted target label
$\mathcal{C}$	Classes
q	Number of classes
$\mathbf{S}$	EEG measurement
w	Weight
CDF	Cumulative distribution function

α	Significance level
Ω	Parse tree
ψ	Genetic operator
ζ	Arithmetic operator
$p(\cdot)$	Probability function
$\mathbb{E}$	Expectation
$\Pr(\cdot\|\cdot)$	Probability distribution
$f(\cdot)$	Function
G	Gini impurity
E	Entropy
b	Bias

CHAPTER 1

Introduction

1.1 Background

The recent progress on machine learning topics has sparked many research projects in science and industry. Many science projects focus on the development of new algorithms or methods and the industry often applies those but tries to improve and automatize them in order to save resources and to create additional business values. Examples are the use of historical car sharing customer data to place shared cars in cities in more efficient ways, or quality assessments with image recognition methods in assembly lines. Numerous industrial applications are imaginable, however, a limiting factor can be a lack of the necessary data and the huge effort to generate it. Nevertheless, many industrial machines or devices already generate data with their equipped sensors which are usually implemented for, e.g., machine monitoring. The data of such sensors are often recorded over time as a so-called time series. Using time series for machine learning can be a tedious and time consuming procedure as they can be and often are of highly complex nature. Next to time series, machine learning models can be used with very different types of data like images, videos, audio, writing, categorical features and numerical features. This results in a huge variety of required knowledge needed for such projects. Due to this variety of knowledge combined with the lack of data experts, industry often calls for simple but good performing modeling techniques which can also be applied by non-experts. In other words, industry requires simple, comprehensible and reliable techniques which solve real-world problems with good performances in an automated manner. Such automated methods belong to the group of so-called Automated Machine Learning (AutoML).

When developing a machine learning model in an industrial context, it is important to consider that the machine learning model itself is only one part of a probably long process chain. Therefore, it needs to be implemented seamlessly in the end-to-end process to be able to exploit all its potentials. This can be challenging because it often requires an adaption or recreation of existing processes.

1.2 Objectives

The industry often demands automated approaches which contain sophisticated machine learning techniques for data-driven projects. Automated methods which are easy-to-use and applicable to real world problems are rare, especially for processing time series. Since devices or machines are often equipped with sensors, time series are being recorded already. The recordings of more than one time series from, e.g., one acceleration sensor and one pressure sensor, is a called multivariate time series. Whereas the data of a single sensor recording is called univariate time series.

This work is mainly devoted to multivariate time series. Therefore, an extensive study of methods for multivariate time series is conducted. Based on this, novel time series classification pipelines with more or less complex methods are developed and verified, as well as other state-of-the-art methods like using AutoML methods as part of the pipeline. Pipelines can build optimized models in an automated approach and enable dealing with machine learning topics without time-consuming modeling, even for non-experts. Beyond that, this work proposes novel techniques based on genetic programming and neural networks. The algorithms are tested on both academic and real-world problems.

Another main objective of this work is to develop an end-to-end methodology to build data-driven services in an automotive environment. This methodology earmarks the implementation of earlier mentioned pipelines. As a real-world problem in the automotive industry, we investigate a data-driven service to assess the damage caused by an accident, i.e. based on vehicle on-board data this model can compute the damaged parts of a vehicle involved in a crash. Amongst others, in the development of the data-driven model of this service the proposed time series modeling algorithms are tested on this real-world problem. The relevant research questions (RQ) of this work are stated in the following:

RQ1 Can we develop efficient time series classification approaches by using feature-based techniques? (in terms of the trade-off between classification quality and computation time)

RQ2 Can AutoML methods be applied to solve time series classification tasks?

RQ3 Are state-of-the-art feature selection methods suitable for time series classification tasks based on a massive number of extracted features?

RQ4 Can the classification quality be increased when combining features with genetic programming?

RQ5 Can we develop a systematic approach for efficiently deploying data-driven services in the industry?

These questions are answered in this dissertation with algorithmic contributions and the linking of most promising existing methods. All methods are applied on academic data sets for a solid comparison. In order to evaluate the industrial use, the methods are also applied on industrial data sets.

1.3 Outline of the Dissertation

In this section, the outline of this dissertation with its motivation and content is briefly described. After each chapter the corresponding publications are listed.

Chapter 2 provides an overview of an industrial project called automated damage assessment. The objective of this project is to estimate the damaged parts caused by a low speed vehicle crash without any additional sensors, i.e. it is only based on already existing sensor data like acceleration signals. Such a digital damage assessment service would support the vehicle customer in the settlement of an accident. The business motivation and the technical challenges are presented.

As part of this dissertation ideas of the invented damage assessment system are summarized in the following patent application:

> Koch, M. and Hundt, W. and Malotta, J. and Godau, R. and Geiger, M. and Krieger, J. (13.06.2019). A Method of Determining Damage Occurring in an Accident between a Vehicle and a Collision Partner on the Vehicle. *World Patent Application WO2019110434A1.*

Chapter 3 introduces the used methods in this dissertation. Traditional machine learning models, neural networks, state-of-the-art AutoML methods, as well as evolutionary algorithms are described. Furthermore, the challenges of feature engineering when dealing with time series, its methods and hyperparameter optimization are discussed. Parts of this chapter are published in the following article:

> Koch, M. and Wang H. and Bürgel R. and Bäck, T. (2018). A Comparison of Hand-Crafted and Automated Machine Learning Approaches for Multivariate Time Series Classification. submitted.

Chapter 4 proposes novel methods for time series classifications. A first method called plain hand-crafted pipeline represents a practical approach which aims at creating good performing models with high efficiencies. Another pipeline method is based on genetic programming with the objective to create complex high level features from many low level features. Furthermore, hand-crafted and state-of-the-art neural network designs are introduced for this purpose. Parts of this chapter are published in the following article:

> Koch, M. and Wang H. and Bürgel R. and Bäck, T. (2018). A Comparison of Hand-Crafted and Automated Machine Learning Approaches for Multivariate Time Series Classification. submitted.

Chapter 5 shows three applications of the plain hand-crafted pipeline on real-world data sets, namely on two automotive on-board data sets and one medical data set. The data set of the first application contains recorded vehicle on-board data from crash tests with the locational crash information on the car. The objective is to estimate the impacted location. In the second application, based on recorded vehicle on-board data from crash tests which were carried out within this work, damaged parts caused by crash events are estimated. In the third application, data from Parkinson's disease patients is used to identify the usefulness of a certain surgery to enhance the patients well-being.

These three applications of the plain hand-crafted pipeline were previously published in:

> Koch, M. and Bäck, T. (2018). Machine Learning for Predicting the Impact Point of a Low Speed Vehicle Crash. In *Proceedings of the 17th IEEE International Conference on Machine Learning and Applications*, ICMLA '18, Orlando, USA, pp. 1432–1437.

Koch, M. and Wang, H. and Bäck, T. (2018). Machine Learning for Predicting the Damaged Parts of a Low Speed Vehicle Crash. In *Proceedings of the 13th International Conference on Digital Information Management*, ICDIM '18, Berlin, Germany, pp. 179–184.

Koch, M. and Geraedts V. and Wang H. and Tannemaat M. and Bäck, T. (2018). Automated Machine Learning for EEG-Based Classification of Parkinson's Disease Patients. IEEE Big Data '19, Los Angeles, USA, pp. 4845–4852 (2019).

Chapter 6 shows the comparison of all proposed methods for multivariate time series classification, as well as promising state-of-the-art AutoML approaches implemented in the pipeline. As basis for the comparison, 18 publicly available multivariate time series data sets and the data set containing on-board data from crash events with its damaged parts are used. Especially with the latter mentioned data set the industrial uses of the methods are investigated. The methods are compared based on the performance and the efficiency (computation time). This extensive comparison is based on the following publication:

Koch, M. and Wang H. and Bürgel R. and Bäck, T. (2018). A Comparison of Hand-Crafted and Automated Machine Learning Approaches for Multivariate Time Series Classification. submitted.

Chapter 7 proposes a novel method to develop data-driven services in the automotive industry. The damage assessment system is used as an exemplary data-driven service. The proposed methodology aims at giving an overview of key points regarding business motivations, as well as the technical challenges when creating successful services. This work has been published in this article:

Koch, M. and Wang, H. and Bürgel and Bäck, T. (2020). Towards Data-driven Services in Vehicles. In *Proceedings of the 6th International Conference on Vehicle Technology and Intelligent Transport Systems*, VEHITS '20.

Next to the mentioned publications above, another publication of the author regarding this topic is:

Geraedts V. and Koch M. and Contarino M. and et al. (2020). Automated EEG-based Machine Learning for Cognitive Profiling during the DBS Screening in Parkinson's Disease Patients. submitted.

CHAPTER 2

Automated Damage Assessment

2.1 Objectives

Industrial applications suffer wear. For example, in assembly lines parts are often exchanged at an early stage to avoid fatal failures and longer downtime's, even when the parts are not roughly worn out. Exchanging such parts later would be beneficial due to longer part operation times and lower material costs. The objective is to exchange parts very shortly before the critical condition of the part is reached. Assembly lines are often equipped with sensors which record conditions. Based on machine learning techniques and the data recorded by the latter sensors, predictive models can be created to estimate failures more precisely without sophisticated process monitoring systems. Using such predictive models for failure detection is called predictive maintenance. Predictive maintenance can not only be realized in assembly lines but also in, e.g., passenger cars, trucks, construction machinery or any other kind of devices or machines which have useful data.

Though, some applications suffer other kinds of losses which are often caused by the human interaction due to incorrect handling or undesirable incidents like vehicle accidents. Such cases often require experts to inspect the damage, evaluate the damaged parts and carry out the repair. Depending on the damage and the machine this can be a time-consuming procedure. Using predictive models for such incidents would allow estimating the loss and the damaged parts directly after a vehicle accident. This estimation would decrease the time needed for the repair, because damaged parts can be ordered and painted even before the car arrives at a workshop. Furthermore, the owner of the vehicle would receive a very transparent overview of the required repair, as well as the precise time needed for the repair.

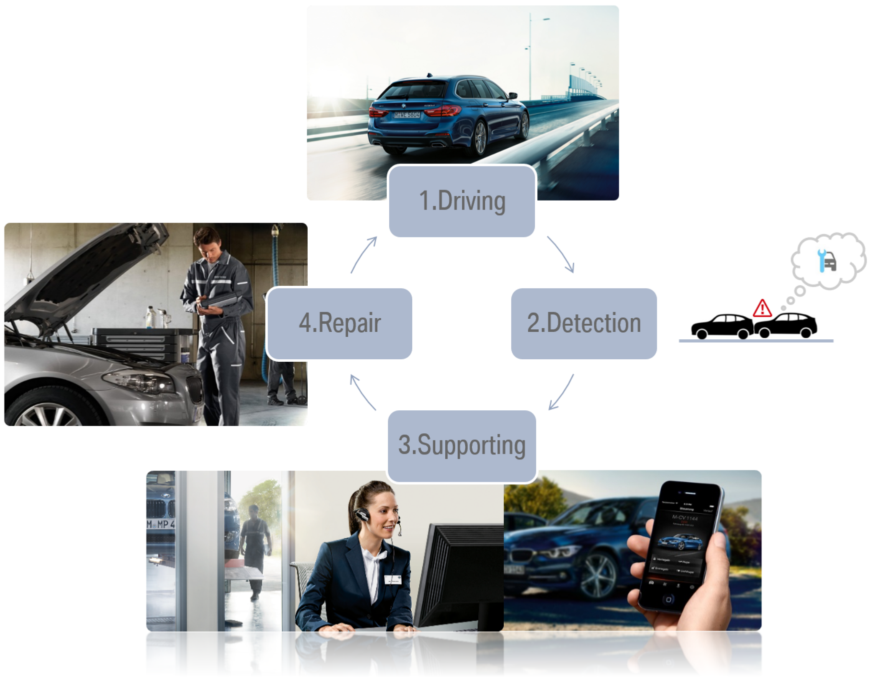

Figure 2.1: A vision of a customer journey of a low speed crash settlement.

The feasibility study to estimate the damaged parts caused by a vehicle crash is carried out in the automated damage assessment project. This project focuses on low speed crashes with a maximum change in velocity of approximately 16 km/h (RCAR, 2018). Low speed crashes usually do not cause injuries to the occupants and are considered as the most occurring accidents. Though, such events are stressful for occupants and usually associated with longer settlement times. An automatic assessment would primarily support the customer shortly after the accident and secondly improve the whole damage settlement.

Figure 2.1 illustrates a vision of an ideal customer journey of a low speed crash settlement.

The journey is separated in four phases:

1. Driving,
2. Detection,
3. Supporting and
4. Repair.

The vision is arranged in a circle for clockwise reading and the starting point is called driving. The driving represents the main purpose of using a car. In the unlikely and unfortunate case of a low speed accident the car is able to detect such an event (phase 2). This detection is based on machine learning techniques (de Silva et al., 2017). A message is shown on the HMI (Human-Machine Interface) in the car with the request to call the accident call center, which supports (phase 3) the customer in all aspects, i.e. calming down, calling the ambulance and police if necessary, helping to proceed in the correct way and finally take over the whole settlement with, e.g., insurance companies and workshops.

Shortly after the customer has granted the request a connection to the accident call center is established. Furthermore, a small data package is transferred to a back-end system which is used by the machine learning assessment model to estimate the damaged parts (de Silva et al., 2017; Koch et al., 2019). This estimated assessment information is provided to the customer which allows the customer to settle the whole accident shortly after the crash via call or app. In the last phase of Figure 2.1, the repair, the car is picked up and delivered to the workshop, repaired and brought back to the customer. In all steps, the customer can access the current car status.

One objective of this project is to increase the time the car is available for driving, i.e. the time of workshop visits should be shortened. Furthermore, this automated assessment should provide a convenient and transparent way to settle such accidents.

Amongst many other things, within this dissertation, we evaluate time series classification methods to assess the damages caused by low speed crashes. Furthermore, along this service development, we have evolved a method to run such data-driven projects in the industry.

2.2 Data

The damage assessment project is targeted to use only vehicle on-board data which is available in all modern passenger cars, i.e., no additional sensor is needed for this purpose. Nowadays, not all cars are equipped with sophisticated camera systems, radar and so on. However, nearly all cars have some sensors, e.g. acceleration sensors for airbag deployment. This project focuses primarily on such sensor systems which are available in most of the vehicles, because it is very likely that an assessment model based on those data can already deliver precise results. And equally important, the number of cars equipped with serial sensors is much higher than cars equipped with more sophisticated sensors. Therefore, using only serial sensor data allows to equip as many cars as possible with such a service. Furthermore, it keeps the data packages to a minimum which is important to limit needed resources like bandwidth, memory and so on. Obviously, for cars with a sophisticated sensors equipment additional data can be used if needed and for new car designs additional sensor could be added.

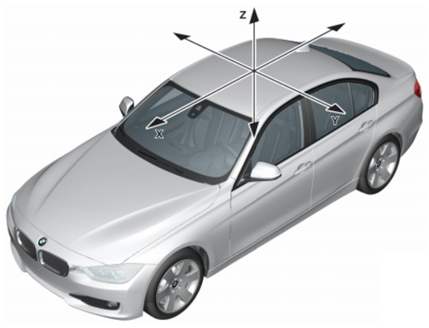

Figure 2.2: The vehicle coordinate system. (Original photo from `BMW Group Technical Qualification Media`)

To develop such an assessment model, sensor data from crash events, as well as its damages are needed. Within this work first data was generated in a tedious manual process. In the following our crash test setting is described: 12 similar cars were used and crashed in a driving-car-to-standing-car configuration in 50 different combinations. Each of those cars was equipped with recording devices to track the sensor data while the accident had happened. Due to the 50 different combinations, our final data set contains 100 cases: 50 from standing vehicles and

50 from driving vehicles. Each case contains the tracked sensor data of the crash events. We have built our assessment models with the following sensor data:

- Acceleration X,
- Acceleration Y,
- Yaw rate,
- Resulting velocity.

Figure 2.2 shows the vehicle coordinate system. It indicates that X describes the longitudinal direction and Y the lateral direction. The used sensors are located close to the vehicle center of gravity.

After each crash combination both cars were partially disassembled to investigate the damage in detail. A low speed crash usually does not cause any damage on structural parts. Therefore, this damage investigation only requires removing the outer, mounted parts. Some representative damaged parts of low speed front crashes are shown in Figure 2.3.

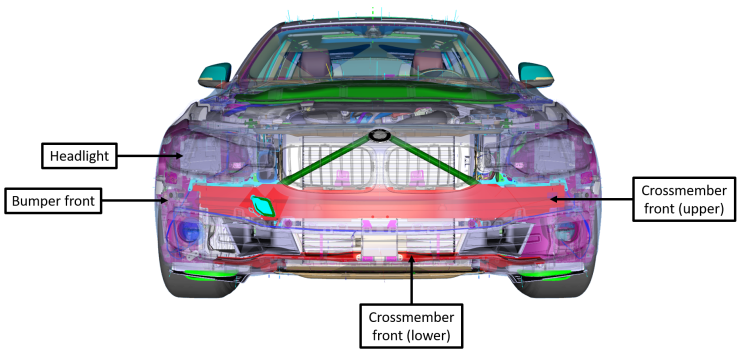

Figure 2.3: Exemplary damaged parts of a low speed crash.

CHAPTER 3

Methods

This chapter introduces the methods used in this dissertation. Machine learning and time series classification are described in sections 3.1 and 3.2. In the following sections, the learning models, decision trees and random forests, as well as feature engineering techniques are discussed. Afterwards, hyperparameter optimization and automated machine learning are described. Finally, neural networks which can build models without explicit features, and evolutionary algorithms are introduced.

3.1 Machine Learning

Machine learning describes the process of computers learning patterns of data. This learning is done without any requirement of explicit programming. Such learning algorithms can be separated in:

- Supervised learning: a model is trained with input data x_i and output data y_i. Based on new input data x, a learned model can predict the output y.
- Unsupervised learning: a model is only trained with input data to, e.g., cluster the data into a number of desired classes.

This work is devoted to supervised learning techniques. Supervised learning includes most of the methods used in machine learning. While in traditional programming observations are explicitly programmed into decision rules, supervised learning techniques develop such rules based on input and output (also called target) data without explicit programming. Famous examples of applied supervised techniques are house price predictions based on attributes like capita crime rate by town and full-value property-tax rate of a region (Harrison and Rubinfeld, 1978). Based on these attributes a supervised learning model can predict the house price. The prediction of such numeric values belongs to the regression tasks. The most simple regression model is called linear regression. Based on the linear combination of the input variables $x_1, x_2, \ldots, x_K$,

$$Y_i = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_K x_K, \tag{3.1}$$

a linear regression model derives the optimal weights $w_0, w_1, \ldots, w_K$ by minimizing the loss function on the training set. A popular loss function is called mean squared error (MSE). It is defined as

$$\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2, \tag{3.2}$$

with the predicted values $\hat{Y}_i$ and the real values Y_i. Another popular loss function is called root mean square error (RMSE):

$$\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2}. \tag{3.3}$$

Such a regression model is often of limited use due to its linearity, i.e. it can only describe simple data relations.

Most of the regression algorithms can also be used for classification tasks. A classification of time series describes a mapping of the time series onto discrete class labels.

When creating prediction models, the number N of available observations, called data points, varies from one problem to another. In some cases, plenty of data is already available, in other cases data has to be generated by running tedious experiments or simulations. Especially when dealing with time series and using simulations, the quality of the data is often difficult to estimate. Therefore, larger numbers of experimental data is often key when developing realistic models.

The goodness of a model, i.e. the difference between prediction $\hat{Y}_i$ and the real labels Y_i, can be measure with different metrics. The most known metric is called accuracy which is defined as follows:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}} \cdot 100\% \,. \tag{3.4}$$

It provides a percentage score of correct predictions. Especially for data sets with an imbalanced number of data points per classes (class-imbalance) accuracy is not a suitable score. For example, a data set contains 100 data points and 2 classes with 95 data points representing class 0 and 5 data points representing class 1. A model which classifies all data points as class 0 would result in an accuracy score of 95%, since 95% of the predictions are right. However, the model is not able to separate between the classes even when the accuracy metric shows good scores. Therefore it is important to consider other, more suitable performance measures like the so-called $F1$ score. In multi-class classification, it can be calculated for each class/label separately. Given the number of classes q, for a class $i \in \{1, \dots, q\}$, $F1$ is defined as the harmonic mean of precision and recall in this class:

$$F1(i) = 2 \cdot \frac{\text{precision}(i) \cdot \text{recall}(i)}{\text{precision}(i) + \text{recall}(i)}, \tag{3.5}$$

where precision(i) is the ratio of correctly classified data points over all data points being classified to class i, and recall(i) measures the ratio of correctly classified data points across all data points belonging to class i. Especially for imbalanced data sets the so-called *macro* $F1$ score can be considered:

$$F1_{\text{macro}} = \frac{1}{q} \sum_{i=1}^{q} F1(i). \tag{3.6}$$

An often used technique to display the performance over different classes is a so-called confusion matrix. A confusion matrix describes the frequency of cases that are correctly or incorrectly classified (Hofmann and Chisholm, 2016) and provides a useful illustration of classification quality for all classes.

Furthermore, visualization methods like, e.g., learning curves, the Receiver Operating Characteristic (ROC) and the precision / recall curve provide further performance insights. A learning curve is a graphical representation of how an increase in a performance measure is achieved as more data (experience) becomes available (Geron, 2017). The ROC curve describes the trade-off between the true positive rate[1] and the false positive rate[2] while the Area Under the Curve (AUC) quantifies such a trade-off. Depending on the data, the ROC and / or the precision / recall trade-off provides additional understanding of the performance of the model. The precision / recall curve should be especially considered when having a small number of observations belonging to the positive class (Geron, 2017).

[1]The rate of correctly predicted positive labels to all positive labels.
[2]The rate of correctly predicted false labels to all false labels.

The main objective when building machine learning models is to create so-called generalizing models which work well on unseen data. To achieve this, different sampling strategies are used to evaluate the model performance on unseen data. The probably most popular method is called holdout which means that all available data is randomly split into two parts: training data and test data.

A common split for training and test data is 75%/25%. Using a train-test split works well when both train and test data contain similar data characteristics. This can be achieved with a large number of data points and partly with a stratified split. The latter describes a split which distributes the data regarding the labels and the percentage equally in train and test parts. Especially when dealing with small data sets, the goodness of the model depends strongly on the certain split due to usually very in-homogeneous data, i.e., a simple train-test split cannot give a realistic impression of the goodness. Not only in such cases, so-called cross validation can give a better impression of the generalization of the model. In a cross validation the data is randomly split into N_{CV} (with e.g. $N_{CV} = 10$), trained on $N_{CV} - 1$ folds and tested on the remaining fold. This process is repeated until each fold has served as test set. The average of all test scores of the N_{CV}-computations represents the final score. This reveals a realistic score of the generalization possibilities of the models. When setting $N_{CV} = N$, all but one data points are used for training a model. This trained model is used to predict the remaining data point. This is repeatedly done for all data points. The average of the goodness of all predictions describes the generalization potential. This strategy is called leave-one out cross validation. Of course, this method is comparably expensive regarding computation time but gives probably the most realistic picture of the model's generalization potential, because it is less biased than, e.g., a simple train-test split. Throughout this work, $N_{CV} = 10$ is used whenever a different approach is not mentioned explicitly.

In some cases, learned models perform reasonably well on training data but when it comes to unseen data the predictions are very poor. This is called overfitting. The model memorizes the training data without generalization. Especially machine learning models based on small data sets tend to overfit the data, because uninformative patterns like noise are often interpreted as relevant information. Overfitting can be reduced by lowering the model complexity with, e.g., changing the model from polynomial to linear, removing outliers and irregularities in the data.

The opposite problem to overfitting is called underfitting. Simpler models like linear regression tend to underfit, i.e. these cannot learn all data patterns because their intrinsic complexity exceeds what the model is able to describe.

The next section introduces the problem of time series classification.

3.2 Time Series Classification

The attention towards artificial intelligence has sparked an intense interest in studying time series classification and time series prediction tasks, since they play an important role in major domains like engineering, medicine, biology or finance (Fawaz et al., 2018; Mikalsen et al., 2018). Time Series Classification (TSC) belongs to the group of sequence classification problems (Xing et al., 2010). Sequences containing alphabetic characters are so-called symbolic sequences, whereas a time series describes a sequence of numerical values observed at consecutive equally spaced points in time (Brockwell and Davis, 1991; Xing et al., 2010). The reciprocal of such a space between points in time defines the so-called sampling rate (Brockwell and Davis, 1991), namely the number of data points per second in Hz.

A time series is a sequence of data points: $\{\mathbf{x}_i \colon i \in [1..L]\}$, where L is the length of the series. For multivariate time series, each data point is a vector of d components:

$$\forall i \in [1..L],\ \mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{id})\,.$$

In this work, the data point is denoted as a row vector and is restricted to real values, namely, $\mathbf{x}_i \in \mathbb{R}^d$. When $d = 1$, it is referred to as a univariate time series and naturally when $d > 1$, it is called a multivariate time series. Practically, the multivariate time series represent recordings on a set of physical signals / channels that is measured with the same frequency and duration for each channel. When discussing modeling procedures, the input time series is typically written as a design matrix $\mathbf{X} = (\mathbf{x}_1^\top, \dots, \mathbf{x}_L^\top) \in \mathbb{R}^{L \times d}$ (by stacking all data points together).

The classification labels are either assigned to each time point $\mathbf{x}_i$, or to a whole time series $\mathbf{X}$ (Aggarwal, 2015b,a; Mitsa, 2010; Santos and Kern, 2016). The latter case is commonly seen in applications such as using Electrocardiogram (ECG) signals to predict the arrhythmia class label which is determined for the entire ECG recordings of a patient (Li et al., 2014). In this scenario, to generate a training data set, multiple recordings of the same multivariate time series are measured with the same length, from either different subjects of the study (e.g., different patients) or different time periods. The resulting data set is a so-called *repeated measures* data set $\mathcal{X} = \{\mathbf{X}^{(1)}, \mathbf{X}^{(2)}, \ldots, \mathbf{X}^{(N)}\}$, where N denotes the number of recordings and $\mathbf{X}^{(i)}$ is a design matrix as defined above. Given the classification labels $\mathcal{C} = \{c_1, c_2, \ldots, c_q\}$, the classification target is denoted as $Y = \{y^{(1)}, y^{(2)}, \ldots, y^{(N)}\} \in \mathcal{C}^N$. This work focuses on solving this scenario.

Learning from data with machines require powerful algorithms. In the following section, the classic algorithms decision tree and random forest are described.

3.3 Decision Trees and Random Forests

Decision trees consist of a root, leaves, nodes and branches. Usually decision trees are drawn so that the root is at the top. The branches below the root or rather the nodes are activated when the corresponding condition is true. Leaves represent the end of a branch and the final decision. Decision trees are hierarchical models for classification and regression problems (Guyon et al., 2008). Such trees are especially useful because their visualisation is comparably comprehensible and therefore decisions are interpretable.

Based on Hastie et al. (2009), Figure 3.1 shows a cutout of a decision tree to classify email spam. The root "email (600/1536)" indicates the starting point with 600 normal emails and 1536 spam emails. This tree is trained to classify an incoming email into normal or spam email. Amongst other character combinations higher occurrences of *!*, *\$* or *remove* indicates spam.

The complexity of a decision tree is mainly influenced by the number of nodes, leaves and the depth of the tree (Rokach and Maimon, 2014; Hastie et al., 2009). A decision tree algorithm primarily creates those splitting conditions which can be based on a so-called impurity (Hastie et al., 2009). Impurity, or rather gini impurity G describes the goodness of the split of the data at a node:

$$G = \sum_{i=1}^{C} p(i) \cdot (1 - p(i)). \tag{3.7}$$

For classification problems, C represents the number of classes and $p(i)$ the probability of selecting data of class i. When considering Figure 3.1, the node in the second row on the right side, spam (48/359), shows that all data, arriving

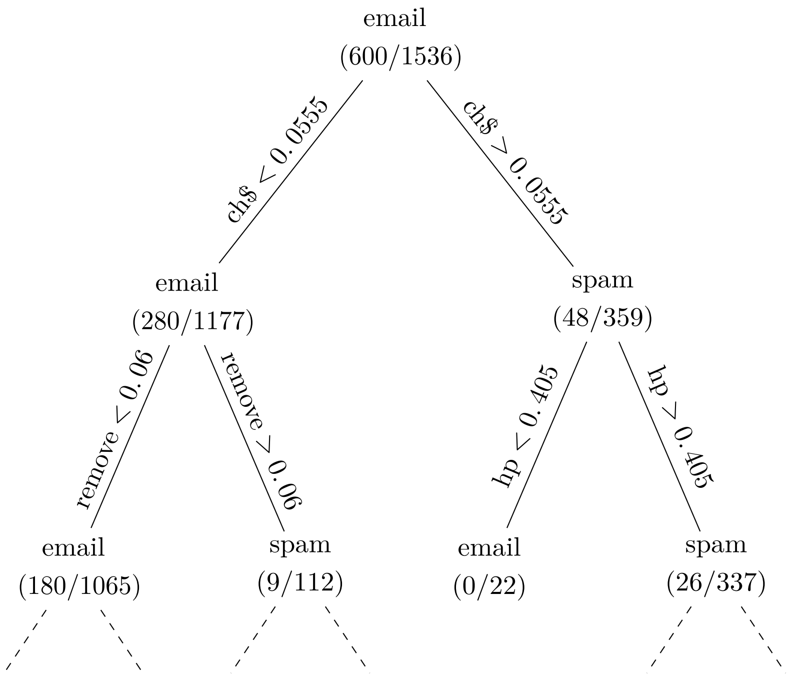

Figure 3.1: A cutout of a decision tree for spam email classification. Each node represents a classification. The split conditions are shown along the branches. The ratio in the nodes illustrate misclassification rates, based on Hastie et al. (2009).

at this point (407 emails) are classified as spam. However, 48 of 407 are normal emails. With $C = 2$, this results in a gini impurity of

$$G = \sum_{i=1}^{C} p(i) \cdot (1 - p(i)) = \frac{48}{407} \cdot \left(1 - \frac{48}{407}\right) + \frac{359}{407} \cdot \left(1 - \frac{359}{407}\right) = 0.2. \tag{3.8}$$

A perfect purity with $G = 0$ is reached when a node contains only members of one class.

Another impurity measure is called entropy:

$$\mathrm{E} = -\sum_{i=1}^{C} p(i) \cdot \log_2(p(i)). \tag{3.9}$$

Entropy is a measure of disorder and describes the degree of randomness. In case of homogeneous samples the entropy is 0, in case of similarly divided samples the entropy is 1. Considering the node spam(48/359) results in an entropy of

$$\mathrm{E} = -\sum_{i=1}^{C} p(i) \cdot \log(p(i)) = -\frac{48}{407} \cdot \log_2\left(\frac{48}{407}\right) - \frac{359}{407} \cdot \log_2\left(\frac{359}{407}\right) = 0.52. \quad (3.10)$$

Based on the entropy, in a next step, the information gain I_G is computed. The information gain describes the effectiveness of linked conditions (C_i). A higher information gain indicates a decrease of entropy with

$$I_G(C_1, C_2) = \mathrm{E}(C_1) \text{ - } \mathrm{E}(C_2). \quad (3.11)$$

A decision tree algorithm based on entropy tries to find conditions which maximize the information gain.

A decision tree algorithm builds one tree. A random forest contains a forest of many single decision trees. The results of each individual tree are aggregated into the final decision, i.e. in a classification problem the most frequently chosen class of all individual trees represents the aggregated result of the random forest. Random forest is an ensemble learning method which is considered to achieve good results in different domains. Random forests learn deeper patterns than a single decision tree and the aggregation of many different decision trees limits the overfitting (Hastie et al., 2009).

Decision trees, random forests and many other classical machine learning models are based on features which are often numeric or categorical values. To create good performing models, it is important to find and select the most promising features in the process called feature engineering.

3.4 Feature Engineering

The most traditional way to optimize the performance of machine learning models is by enhancing features and model hyperparameters in a manual approach. This is very time-consuming due to the large number of possibilities and combinations. Nowadays, already with default model parameters good performances can be achieved. Nevertheless, intense optimizations of hyperparameter and especially the features still improve the performance in a significant way. In general, using feature-based methods enables creating transparent and interpretable models particularly due to the descriptive nature of features.

When dealing with time series instead of given features, optimization becomes even more important due to the needed extraction of representations (features) from the series. The extraction and selection of significant representations are of key importance for successful modeling.

3.4.1 Feature Extraction from Time Series

Classical machine learning models like decision tree or random forest require stationary features. When dealing with time series those features need to be extracted. This means a time series has to be transformed into a one dimensional vector of descriptive features like the maxima, minima, number of peaks and so on. The relevant features differ from one problem to another, i.e. each problem requires another set of suitable representations. For good performing models it is essential to find the most relevant time series representations. Promising feature extraction methods are described in the following.

Time series classification can be realized by means of meta-properties which are typically only able to describe a very limited set of time series characteristics (Kadous and Sammut, 2005). However, if the relevant characteristics or the majority is among them, this technique can be an appropriate and computationally cheap choice. Ben D. Fulcher et al. (2013) proposed an algorithm for sorting time series data sets regarding their properties on the basis of extracted representations. A similar approach, but more all-encompassing is called Highly Comparative Time-Series Analysis (HCTSA) which extracts 7749 features in an automated approach (Fulcher et al., 2012; Ben D. Fulcher et al., 2013; Fulcher and Jones, 2014, 2017). A related approach is called Time Series Feature Extraction based on Scalable Hypothesis Tests (tsfresh), proposed in (Christ et al., 2016, 2018). From each time series tsfresh computes 794 features based on 63 time series characterization methods. It is designed for industrial applications (Christ et al., 2016). Feature extraction methods like tsfresh allow finding significant features without a deep domain knowledge. Furthermore, for automated time series modeling including feature extraction it is typically indispensable to compute a very large number of properties (features) in order to ensure that the relevant characteristics for the problem in question have been generated. Such a large feature space contains noisy and irrelevant features but also features with significant information regarding the modeling problem. It is important to find those significant features and to exclude the other ones. Especially excluding irrelevant features is important in order to enhance the general model performance. This procedure results in a lower dimensional feature space which can provide more transparent and comprehensible models.

In conclusion, to first extract a massive number of features and second, to select the most important features for a certain problem is a promising approach because it is automated and can solve a vast variety of problems.

However, extracting features from time series requires a computational effort. tsfresh provides with 794 features a smaller number of features compared to HCTSA, which extracts 7749 features per time series. Due to the smaller number of extracted features, tsfresh, which is especially designed for (industrial) applications, requires way less computational effort compared to HCTSA. Furthermore, tsfresh is a Python based library which can be seamlessly implemented into Python based machine learning pipelines as developed in this work. In contrast, HCTSA is a Matlab based package (Fulcher et al., 2012; Ben D. Fulcher et al., 2013; Fulcher and Jones, 2014, 2017). For the reasons given above in this work tsfresh has been used to extract the features.

A selection of important features is associated with a reduction of the dimensions of the initial feature space. This can be done with, e.g., methods belonging to the area of feature generation, which transforms high dimensional low level features to so-called low dimensional high level features.

3.4.2 Dimensionality Reduction with Feature Generation

Feature generation or feature construction describes the process of creating new features from existing ones. Guo et al. (2005) applied genetic programming on raw vibration time series data to classify bearing faults. They stated the use of genetic programming for feature extraction / generation as efficient and powerful.

The general purpose of feature generation is to reduce the dimensions without dropping important information (Guyon et al., 2008). This can be done by applying, e.g., Principal Component Analysis (PCA) or neural networks (Jain et al., 2000). PCA reduces the dimensions by combining variables in a linear transformation and dropping unimportant ones (Hastie et al., 2009). Neural networks (see Chapter 3.6) can reduce the dimensions due to smaller sizes of the subsequent layers (Hinton and Salakhutdinov, 2006).

Another method for generating low dimensional high level features from high dimensional low level features is called particle swarm optimization (PSO). PSO is similar to evolutionary algorithms and based on a population iterated over generations. A swarm of particles is *flying* with a certain *velocity* in a solution space. Within the generations each particle converges towards its best location in the solution space (Kennedy and Eberhart, 1995). Xue et al. (2013) proposed a method to use swarm optimization for feature generation: one high level feature is computed from the multivariate feature space. The performance of the classification is comparable or better than using the original features. Beyond, Swesi and Bakar (2020) show an overview of other recently proposed techniques for feature generation.

The probably most traditional and popular technique for dimensional reduction without changing the features itself is called feature selection. Feature selection methods choose the most significant features from a given feature space.

3.4.3 Feature Selection

Feature selection describes the process of selecting the most significant features from a larger features space without changing individual features (Guyon et al., 2008).

The most common selection algorithms for features can be divided into filter-, wrapper- and embedded methods. Filter methods rank features independently from the machine learning model by means of statistical tests, e.g., t-test or χ^2 test. These methods are considered to be computationally effective and generally robust to overfitting (Guyon et al., 2008).

Wrapper- and embedded methods contain a machine learning algorithm. Both methods are more expensive than filter techniques, but more powerful towards higher accuracy models. Wrappers are mainly independent from the training process itself, because the selection of features is done in a first step and the selected features are afterwards used in the main training process. Wrapper algorithms evaluate subsets of features in, e.g., a forward or backward approach. In forward selection, features are added iteratively until a performance measure stagnates or deteriorates. Alternatively, the backward selection method (a.k.a. backward elimination) starts with all features and gradually removes less important features. A widely known algorithm for feature selection is Recursive Feature Elimination with Cross-Validation (RFECV), which repeatedly constructs cross-validated models (in this case a random forest) with a defined number of features. For each calculation the most important features are identified. Loops of RFECV are run until all features are tested (Guyon et al., 2006). More computationally expensive but more powerful wrapper approaches are based on evolutionary algorithms, i.e. feature subsets (population) are modified based on the evolutionary theory in each generation iteratively. This approach is especially challenging for large feature spaces, because of the huge number of possibilities. In each generation best features are conserved based on a fitness function. Harvey (2014); Harvey and Todd (2015) suggested a method for feature selection and some generation for time series classification with the use of genetic programming in the area of structural engineering. Tran et al. (2016) proposed the construction as well as selection of high-dimensional features by using genetic programming. With only 4% of the initial data set, this method outperforms 15 of the 18 cases (Tran et al., 2016). Using evolutionary algorithms for feature selection is an often scientifically discussed topic. Evolutionary algorithm approaches are often pareto-based on a multi-objective setting, which achieve mostly high performances with a small number of features (Castelli et al., 2011; Cano et al., 2017). An overview of recent publications can be found in Chandrashekar and Sahin (2014); Xue et al. (2016)

In embedded approaches the feature selection is implemented in the main training process (Guyon et al., 2008). Popular embedded methods are L1 Regularization (Lasso) or Ridge Regression. Embedded methods are not relevant for this work but mentioned for the sake of completeness. Embedded methods are further discussed in Guyon et al. (2008).

As stated previously feature engineering methods improve the performance of machine learning models and another technique to enhance the outcome will be introduced in the next section. It is called hyperparameter optimization and deals with the optimization of machine learning algorithm parameters.

3.5 Hyperparameter Optimization

Hyperparameters control machine learning processes. To boost up the performance of a machine learning model, it is necessary to conduct a hyperparameter optimization. Well-known methods for parameter optimization are grid search, random search, and Bayesian optimization (Ramasubramanian and Singh, 2017; Geron, 2017; Florea and Andonie, 2018). For all methods, in a first step, the ranges of the parameters have to be assigned to a search space. While grid search systematically tries all possible combinations of the defined feature space, a random search approach randomly creates combinations until a stopping criterion is reached. In each case the parameter combination with the best performance is used. Random search is considered to work more efficiently than grid search (Bergstra and Bengio, 2012). Bayesian optimization belongs to the field of advanced global optimization techniques and is considered to be very efficient for finding an optimum of expensive objective functions (Mockus, 1994; Jones et al., 1998). The objective function is a black box function with boundaries defined by the search space. In each step of the iteration, prior distributions are used to approximate the objective function with a so-called surrogate function within the parameter space (Brochu et al., 2010). Prominent surrogate models are so-called Gaussian processes or random forests. Random forests have the advantage to operate natively with mixed integer data.

Instead of extracting explicit features from the time series to develop machine learning model, neural networks can be used to learn features from the initial time series without explicit programming. Neural networks are considered as one of the most powerful machine learning methods. These are described in the following section.

3.6 Neural Networks

A neural network is inspired by the human brain. In the human brain cells, so-called neurons, are connected to each other. In a neural network, layers with neurons are located next to each other, in between an input and an output layer. The connections between neurons of the previous and the following layer are modelled as weight functions with two different states: stimulation and inhibition (Hastie et al., 2009; Zhang, 2010; Aggarwal, 2018). Learning patterns with such a combination of successive layers is considered as Deep Learning which is a subfield of Machine Learning (Chollet, 2018) (see Chapter 3.1).

The most simple neural network is a so-called perceptron with only an input- and an output layer.

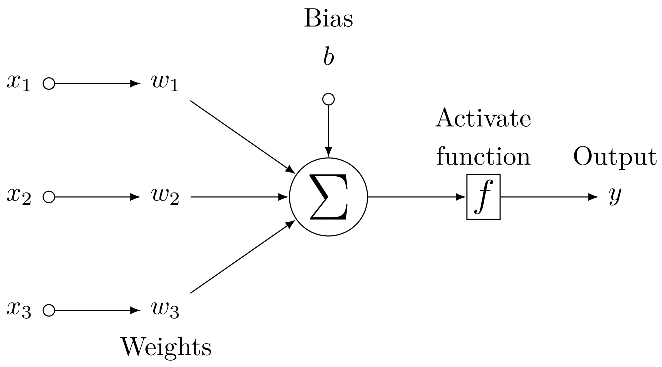

Figure 3.2: The perceptron is a single-layer neural network architecture, based on (Rosenblatt, 1958; Aggarwal, 2018).

Figure 3.2 shows such a perceptron with I inputs x_i (here $I = 3$) and the output y. Each input is multiplied by the certain weight, summed and added up with the bias:

$$\text{Weighted sum} = \sum_{i=1}^{I} x_i \cdot w_i + b. \tag{3.12}$$

The bias b can be used to shift the data. In the next step, an activation function is applied on the weighted sum. An activation function is, e.g., an unit step function like

$$f(x) = \begin{cases} 0, \text{ if weighted sum} < 0 \\ 1, \text{ if weighted sum} >= 0. \end{cases} \tag{3.13}$$

Other popular activation functions are called Sigmoid, Logistic, Tanh (Hyperbolic tangent) or ReLu (Rectified Linear Unit).

A perceptron is a single layer neural network, whereas multi layer networks contain additional so-called hidden layers in between the input and the output layer.

Figure 3.3 shows a typical multi layer neural network architecture with input layer, output layer and one hidden layer in between. Layers which are fed into the next following layer are considered as feed-forward layer.

When training such a neural network an objective function or rather loss function like cross-entropy is used to update the weights. In the beginning, the weights of a neural network are usually initialized at random. Then the input data is used to compute a first output. The difference between the computed output and the true output is expressed by the loss function. Note that at this point the result of this function is only based on a random initialization and therefore, in the next steps the weights will be optimized. For this, different optimization techniques can be considered. A prominent and effective one is based on differentiation. In this method the derivative of the loss function is computed. Based on the depending gradients the objective function is used to enhance the weights. When thinking about neural networks with a large number of layers and neurons it is a tedious process to find the balance of the best suitable weights based on the full loss function. To face this problem, a technique called backpropagation can be used which derives the loss function neuron-wise starting from the end layer towards the input layer. In this manner, each weight is optimized individually (Feldman and Rojas, 2013; Aggarwal, 2018).

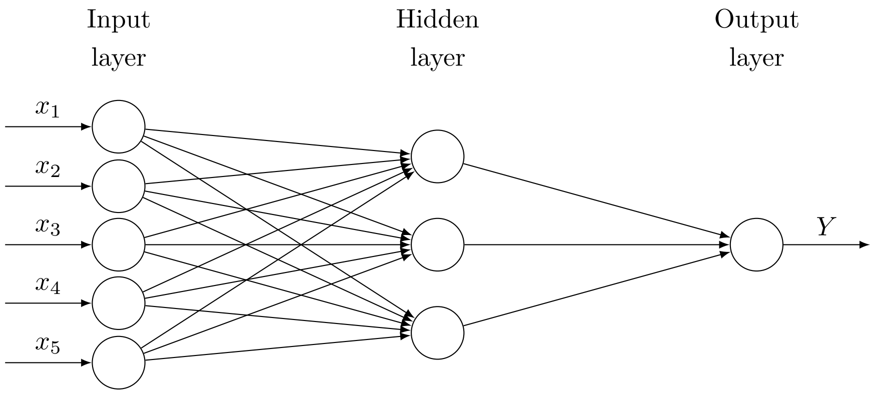

Figure 3.3: A multilayer perceptron with one hidden layer, based on (Aggarwal, 2018).

Using neural networks usually means a computationally expensive training process. Therefore, depending on the data and the problem, the most suitable architecture regarding performance and efficiency should be chosen. Popular neural network architectures are perceptrons, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM). CNNs were developed for image recognition problems and considered as the most successful neural network (Aggarwal, 2018; Lecun et al., 1998). In CNNs a filter is applied on a layer to generate a feature map. A feature map of one of the first layers represents the data in a lower dimensional way. Based on this feature map, an activation function decides in what way to proceed with the features. Usually a first layer can learn basic features like corners or edges. The following layers can learn parts of the image like, considering a picture of a face, eyes or noses. The last layers can learn full objects. This indicates that first layers extract basic features (low dimensional) and posterior layers contain more and more information of the whole object (Aggarwal, 2018).

A Long Short-Term Memory has, as its name suggests, a memory. It describes a variation of Recurrent Neural Networks, which iterate over the data points and compute a state containing information of the data it has seen before. Compared to RNNs, LSTMs can additionally save information for long time periods. LSTMs decide which information should be kept and which should be dropped (Chung et al., 2014; Chollet, 2018).

3.7 Evolutionary Algorithms and Genetic Programming

Genetic programming is an evolutionary algorithm technique (Koza and Rice, 1992). Evolutionary algorithms are based on biological evolution, i.e., on genetic changes in a population, which are inherited over generations and are subject to mutation, recombination and selection based on the objective function values (also called fitness) assigned to solution candidates (Bäck, 1996). Genetic programming aims at applying this principle to the domain of optimizing computer programs. In this work, it is used to find a single *best* feature by combining several other features into one feature.

The solution space of solving Machine Learning problems is wide due to the combinations of hyperparameters, features, models and so on. Finding the best working combination by hand is nearly impossible. Therefore, automated machine learning methods aim at selecting the modeling algorithms, applying feature engineering and optimizing the hyperparameters in an automated approach. It is introduced in the next section.

3.8 Automated Machine Learning

Based on extracted features from time series so-called automated machine learning methods can be applied. AutoML aims at generating machine learning pipelines automatically, e.g., by using preprocessing methods and various feature selection and prediction models, to create a pipeline with high accuracy predictions or classifications. Such AutoML approaches are often developed for use by non-experts (Feurer et al., 2015). In the following, the most promising AutoML methods are described.

Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2017) uses a tree-based Bayesian optimization procedure to find a strong configuration of the machine learning framework WEKA (Witten et al., 2011) for a certain data set. Auto-Sklearn can be considered as an extension of Auto-WEKA (Feurer et al., 2015, 2018), whereas Auto-Sklearn is based on scikit-learn (Pedregosa et al., 2011) instead of WEKA. The main differences are that Auto-Sklearn includes a lower number of classifiers, a smaller set of hyperparameters, and its ensembles are builtin a second step. The smaller search space of Auto-Sklearn increases its efficiency and in total, it results in better performances compared to Auto-WEKA (Feurer et al., 2015).

Based on genetic programming, the Tree-based Pipeline Optimization Tool (TPOT) (Olson et al., 2016) automatically develops and optimizes machine learning pipelines. The pipeline operators (preprocessor, feature selection, decomposition and models) with the respective hyperparameters are combined in a tree-based pipeline. Based on the genetic programming algorithm the whole sequence and each operator are evolved towards maximizing a performance metric. TPOT is based on the libraries scikit-learn (Pedregosa et al., 2011) and deap (Fortin et al., 2012). It contains a number of preprocessing methods like, e.g., feature selection, feature construction with polynomial features, model selection and parameter optimization. Furthermore, TPOT is designed for Pareto optimization to find a Pareto-front of pipelines by maximizing a performance measure like accuracy and minimizing the complexity of the pipeline. TPOT is considered to be efficient and competitive when compared to basic machine learning approaches (Olson et al., 2016). As an extension of TPOT, Layered TPOT was proposed for larger data sets (Gijsbers et al., 2018). It evaluates pipelines only on subsets of the data. The well performing pipelines are tested on all data afterwards. This approach is supposed to yield good pipelines much faster than TPOT.

Another recent method using genetic programming is called Genetic Automated Machine Learning Assistant (GAMA) (Gijsbers and Vanschoren, 2019). It is based on the library scikit-learn and uses its preprocessing and modeling methods. It is closely related to TPOT and builds pipelines with genetic programming, whereas it uses asynchronous evolutionary algorithms instead of synchronous algorithms (Gijsbers and Vanschoren, 2019).

Another AutoML method is called Automated Machine Learning for Production and Analytics (AMLPA) (Parry, 2019). Next to preprocessing, feature selection, model selection and hyperparameter optimization algorithms it also contains a feature learning method. Based on the initial feature space a deep neural network can be used to extract significant features. These features are used for the modeling algorithm. The author of this AutoML library states, that feature learning can increase the accuracy by up to 5%. Furthermore, libraries like TensorFlow, Keras, XGBoost, LightGBM and CatBoost can be embedded (Parry, 2019).

The AutoML method of the H2O library is designed to, e.g., build stacked ensembles, i.e., a combination of different learning algorithms which achieves a higher performance when used together (The H2O.ai team, 2015a,b). Based on the H2O standard library it uses different preprocessing, feature selection and modeling algorithms. Furthermore, a random grid search to optimize the hyperparameters can be applied. Next to, e.g., random forests, extremely randomized trees and XGBoost, also the H2O implementation of neural networks with fully-connected multi-layers is available (The H2O.ai team, 2015a,b).

Another AutoML method is called MLBOX (de Romblay, 2019). MLBOX is based on the optimization library hyperopt (Bergstra et al., 2013) which is used to find the best combination of preprocessing algorithms, feature selection methods, modeling algorithms and its hyperparameters. One key feature of MLBOX is drift identification which describes the process to improve the similarity of training and test set. Especially in real-world problems, both data parts are often differently distributed. MLBOX can perform a so-called *covariate shift* which means it shifts the independent features towards a similar distribution of both training and test set (de Romblay, 2019).

Auto Tune Models (ATM) (Swearingen et al., 2017) is a distributed AutoML system which contains a hierarchical algorithm to automatically select models and optimize the hyperparameters. The selection of the hyperpartition (set of hyperparameters of one path of the hierarchy) is based on a bandit method and the hyperparameter optimization on a Bayesian strategy (Swearingen et al., 2017).

The Resilient Classification Pipeline Evolution (RECIPE) (de Sá et al., 2017) method uses a grammar-based genetic programming approach to generate classification pipelines. The first version of RECIPE includes preprocessing and processing steps. Preprocessing includes, e.g., data normalization techniques and feature selection algorithms. The classification algorithm with its parameters is chosen in the processing step (de Sá et al., 2017).

In the next chapter, the described techniques are used to form approaches to solve multivariate time series problems.

CHAPTER 4

Multivariate Time Series Classification

In recent publications TSC plays an important and challenging role. Most of the proposed approaches for TSC can be categorized into univariate or multivariate techniques. In many cases, only results on univariate data sets are shown due to the limited public availability of multivariate data (Fawaz et al., 2018; Fu, 2011; Esmael et al., 2012; Aggarwal, 2015b). However, many applications, e.g., in engineering or medicine, produce more than one time series. In such cases, techniques dealing with multivariate data sets can significantly improve the performance. Some multivariate data sets were recently published, which has enabled the scientific study of multivariate techniques for TSC at a large scale (Bagnall et al., 2018). In the following section 4.1, the most relevant earlier contributions in the field of Multivariate Time Series Classification (MTSC) are summarized. Then several approaches for MTSC are introduced.

4.1 Related Work

Hidden Markov models showed improved results on two multivariate data sets when compared to an approach proposed in Moghaddam and Pentland (1997) and to neural networks (Bashir et al., 2005). A two kernel approach with a vector autoregressive model outperformed other kernel-based approaches in three out of five tested data sets (Cuturi and Doucet, 2011). Based on industrial time series from drilling sensors a trend- and value approach was proposed in Esmael et al. (2012). This approach relies on and extends the symbolic aggregate approximation (SAX) (Lin et al., 2003), i.e., a segment of the time series is represented by (v, t) where $v \in \{low, normal, high\}$ describes the relative magnitude and $t \in \{up, down, straight\}$ identifies the so-called slope. These extracted symbolic sequences are used as a feature matrix for a classification with, e.g, k-nearest neighbors or decision trees. As an extension of dynamic time warping, the so-called correlation-based dynamic time warping was developed for multivariate time series (Bankó and Abonyi, 2012). These authors combined similarity measures from dynamic time warping and principal component analysis. In 2014, based on structural engineering applications the use of genetic programming for multivariate time series classification was investigated (Harvey, 2014). The author computed 47 features based on using a sliding window. These features were processed with genetic programming techniques. In this benchmark study, however, the methods were tested mainly on univariate datasets. In 2009, shapelets were proposed, which are considered to be subsequences of the time series (Ye and Keogh, 2009). Most discriminating shapelets of different classes are identified and conglomerated in a decision tree. Therefore, the final decision of the tree is based on the comparison of those shapelets. Next to this, a so-called fast classification approach with shapelets was used on multivariate data sets resulting in enhanced accuracy and reduced computation time compared to previous shapelet methods (Grabocka et al., 2016). An autoregressive tree-based ensemble approach was proposed in Tuncel and Baydogan (2018) to model nonlinear behavior of multivariate time series. A novel approach considered different dimensions by feature interplay (Schäfer and Leser, 2017). The authors used a sliding window on each dimension in order to extract features. Based on similarity measures (learned pattern similarity) a generalized autoregression method was proposed to find local patterns in time series (Baydogan and Runger, 2016). A recent novel similarity-based method uses a time series cluster kernel with a Gaussian mixture model (Mikalsen et al., 2018). This approach can deal with missing data and avoids time consuming hyperparameter tuning. Recent contributions show the successful use of neural networks (NN) for MTSC. In this context, a Fully Convolutional Network (FCN) and a Residual Network (ResNet) are considered as baseline results (Wang et al., 2016; Geng and Luo, 2018). FCNs as well as ResNets are developed for object detection in images. ResNets consist, like FCNs, of convolutional layers but with a deeper structure. An overview of other deep learning methods for TSC can be found in Fawaz et al. (2018).

The methods presented above, however, are highly specialized and use a sophisticated modeling approach. They are neither developed for automated modeling nor tested on a significant number of data sets.

4.2 Overview of Approaches for MTSC

In section 4.3, a Plain Hand-Crafted Pipeline (PHCP) is proposed, which aims to be a practical approach using basic methods. Then, in section 4.4, parts of the pipeline are replaced by tree-based genetic programming components to additionally consider the combinations of features from different time series. In section 4.5, the embedding of state-of-the-art automated machine learning approaches in the pipeline is shown. Furthermore, in section 4.6, state-of-the-art neural networks for time series classification, as well as a hand-crafted neural network design are introduced.

4.3 General Approach and Plain Hand-Crafted Pipeline

Most of the approaches to classify time series are based on computationally expensive methods which are nontransparent in the choice of the features. In contrast, one of our hand-crafted approaches is considered to be a plain approach with using basic methods. It aims at being numerically efficient in order to solve classification problems with good performance and acceptable computational effort. The PHCP contains elements of feature-based techniques which have the effect of creating interpretable features.

The approach consists of seven steps, which are each explained in more detail in the following subsections. The first two steps, namely

1. Preprocessing,
2. Exploratory Data Analysis,

require some hand work due to the variety of raw time series characteristics from different domains. This indicates that different data sets require individual preprocessing methods, which have to be identified in a manual process once for each data set. To get an understanding of the data set, it is strongly advised to also execute the exploratory data analysis phase. The next four phases, namely

3. Feature Extraction from Time Series,
4. Feature Selection,

5. Training of a Classifier,
6. Hyperparameter Optimization

form our so-called pipeline. These four steps are completely automated with the input consisting of the preprocessed time series data and the output being the performance scores after the hyperparameter optimization. Our plain pipeline is presented in Algorithm 1: It takes as input repeated measures $\mathcal{X}$ containing N recordings and corresponding target labels Y and starts by extracting time series features (line 3, using the tsfresh package, see Algorithm 2) for each recording in $\mathcal{X}$. Given d-dimensional time series and p predefined feature functions, $d \cdot p$ features are generated for each recording, resulting in a feature matrix $\mathbf{\Phi}^0 \in \mathbb{R}^{N \times dp}$.

The feature matrix then undergoes the feature selection procedure (line 6), where the so-called Boruta algorithm is adopted (see Algorithm 3). Prior to the model training, the feature matrix is then split into the training and test matrices $\mathbf{\Phi}_{\text{train}}, \mathbf{\Phi}_{\text{test}}$ respectively (the split of the matrix is done row-wise). To tune the hyperparameters of the modeling algorithm $\mathcal{A}$, the empirical performance of a candidate hyperparameter vector $\theta \in \Theta$ is assessed through a N_{CV}-fold cross validation (CV) that is conducted on the training matrix (lines 10-15), based on a performance metric f, e.g., accuracy or F1 (see chapter 3.1).

The performance values are obtained and averaged over all validation data sets $(\mathbf{\Phi}_{\text{val}}, Y_{\text{val}})$, which is then passed into a hyperparameter optimizer $\mathcal{H}$ for proposing new candidate hyperparameter settings (line 18). Finally, the best hyperparameter vector θ_{best} is used to train the algorithm $\mathcal{A}$ on the entire training data (line 20) and the generalization ability of the model is measured on the test set (line 22).

The last phase, namely

7. Evaluation of Results

is the manual assessment of the results to evaluate whether the results match the hypotheses.

The seven phases mentioned above are described in detail below. Note that the emphasis of this work is the pipeline itself. Therefore, the preprocessing and the exploratory data analysis step are only discussed from a general perspective.

Algorithm 1 Plain hand-crafted pipeline

```
Require: $\mathcal{X} = \{\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N)}\} \subseteq \mathbb{R}^{L \times d}$ input time series, $Y \in \mathcal{C}^N$ target labels,
    $\mathcal{F} = \{F_i\}_{i=1}^p$ feature functions, performance metric $f$, a ML algorithm $\mathcal{A}$, its
    configuration space $\Theta$ and an algorithm configurator $\mathcal{H}$.
 1: procedure PLAIN-PIPELINE
 2:     for $i = 1$ to $N$ do
 3:         $\phi^{(i)} \leftarrow$ FEATURE-EXTRACTION $\left(\mathbf{X}^{(i)}, \mathcal{F}\right)$        ▷ Alg. 2
 4:     end for
 5:     $\mathbf{\Phi}^0 \leftarrow \left(\phi^{(1)}, \phi^{(1)}, \dots, \phi^{(N)}\right)^\top \in \mathbb{R}^{N \times dp}$
 6:     $\mathbf{\Phi} \leftarrow$ FEATURE-SELECTION$(\mathbf{\Phi}^0, Y)$        ▷ Alg. 3
 7:     $(\mathbf{\Phi}_{\text{train}}, Y_{\text{train}}), (\mathbf{\Phi}_{\text{test}}, Y_{\text{test}}) \leftarrow (\mathbf{\Phi}, Y)$        ▷ train-test split
 8:     Sample $\theta \in \Theta$ randomly
 9:     while stopping criteria are not fulfilled do
10:         $\{(\mathbf{\Phi}^{(i)}, Y^{(i)})\}_{i=1}^{N_{\text{CV}}} \leftarrow (\mathbf{\Phi}_{\text{train}}, Y_{\text{train}})$        ▷ cross validation split
11:         for $i = 1$ to $N_{\text{CV}}$ do
12:             $\mathbf{\Phi}_{\text{val}} \leftarrow \mathbf{\Phi} \setminus \mathbf{\Phi}^{(i)}, Y_{\text{val}} \leftarrow Y \setminus Y^{(i)}$
13:             $\mathcal{M} \leftarrow \mathcal{A}.\texttt{train}(\mathbf{\Phi}^{(i)}, Y^{(i)}, \theta)$
14:             $\widehat{Y} \leftarrow \mathcal{M}.\texttt{predict}(\mathbf{\Phi}_{\text{val}})$
15:             $f_i \leftarrow f(Y_{\text{val}}, \widehat{Y})$
16:         end for
17:         $\bar{f} \leftarrow \sum_i f_i / N_{\text{CV}}$
18:         $\theta \leftarrow \mathcal{H}(\theta, \bar{f})$        ▷ Learn new hyperparameters
19:     end while
20:     $\mathcal{M} \leftarrow \mathcal{A}.\texttt{train}(\mathbf{\Phi}_{\text{train}}, Y_{\text{train}}, \theta_{\text{best}})$
21:     $\widehat{Y} \leftarrow \mathcal{M}.\texttt{predict}(\mathbf{\Phi}_{\text{test}})$
22:     $f_{\text{test}} \leftarrow f(Y_{\text{test}}, \widehat{Y})$
23:     return $\mathbf{\Phi}, \mathcal{M}, \theta_{\text{best}}, f_{\text{test}}$
24: end procedure
```

4.3.1 Preprocessing

Dealing with time series usually implies extensive data preparation. Such raw time series often contain noise and have different lengths, formats and units. Especially time series data from sensors requires certain signal processing techniques like, e.g., filters or transformations (time domain to frequency domain). However, the individual techniques are strongly depending on the domain and the data set at

Algorithm 2 Feature extraction with tsfresh package

```
Require: $\mathbf{X} \in \mathbb{R}^{L \times d}$ an input time series and $\{F\}_{i=1}^{p}$ feature functions.
1: procedure FEATURE-EXTRACTION($\mathbf{X}, \{F_i\}_{i=1}^{p}$)
2:     for $i = 1$ to $d$ do
3:         for $k = 1$ to $p$ do
4:             $n \leftarrow (i-1)d + k$
5:             $\phi_n \leftarrow F_k(\mathbf{X}_{\cdot j})$                    ▷ $j$-th column of $\mathbf{X}$
6:         end for
7:     end for
8:     return $(\phi_1, \phi_2, \dots, \phi_{dp})^\top$
9: end procedure
```

hand. After having processed the data, the exploration phase starts to gain an understanding of the data.

4.3.2 Exploratory Data Analysis

The exploration phase serves the purpose of becoming familiar with the data. This phase is mostly done manually and essential for building comprehensible prediction models, because a good data understanding helps building high performing models. Based on visualization and statistical methods the data set is analyzed in order to find, e.g., outliers and to handle missing values. Especially on small data sets, outliers and missing values can have a significant impact. Therefore, treating those is important for enhancing the model performances (Lissandrini et al., 2018; Zuur et al., 2010; Kantardzic, 2011). In this phase many different techniques can be important to ideally identify first patterns in the data.

4.3.3 Feature Extraction from Time Series

This work aims at creating comprehensible and computationally efficient classification models by automatically generating and selecting time series features. Depending on the domain and the data set, very different features are significant. Traditionally, such features are constructed by hand from the time series, which requires good knowledge from experts. Constructing such significant features in a manual process, however, is usually very time consuming. In order to create a

Algorithm 3 Feature selection with Boruta algorithm

```
Require: $\mathbf{\Phi} \in \mathbb{R}^{N \times m}$ feature matrix with $N$ samples and $m$ features, $Y \in \mathcal{C}^N$
    target, $B$ maximal iterations, $\mathrm{CDF}_{B(k,1/2)}$ cumulative distribution function of
    a binomial random variable $B(k, 1/2)$ with $k$ trials and $1/2$ success probability,
    a significance level $\alpha$ and a random forest algorithm RF with its hyperparameter
    $\theta$
 1: procedure FEATURE-SELECTION($\mathbf{\Phi}, Y$)
 2:     $\{c_1, c_2, \dots, c_m\} \leftarrow \{0, 0, \dots, 0\}$                ▷ counters
 3:     $S \leftarrow \emptyset, R \leftarrow \emptyset$                        ▷ selections and rejections
 4:     for $k = 1$ to $B$ and $S \cup R \neq [1..m]$ do
 5:         $I \leftarrow [1..m] \setminus R$                                  ▷ remaining feature indices
 6:         $\mathbf{\Phi}' \leftarrow (\mathbf{\Phi}_{\cdot i})_{i \in I}$
 7:         for $i \in I$ do
 8:             $\mathbf{\Phi}^{\text{shadow}}_{\cdot i} \leftarrow$ RANDOM-SHUFFLE($\mathbf{\Phi}'_{\cdot i}$)
 9:         end for
10:         $\mathcal{M} \leftarrow$ RF.train$\big((\mathbf{\Phi}', \mathbf{\Phi}^{\text{shadow}}), Y, \theta\big)$
11:         $(\mathbf{z}, \mathbf{z}^{\text{shadow}}) \leftarrow$ FEATURE-IMPORTANCE($\mathcal{M}$)
12:         $\tau \leftarrow \max \mathbf{z}^{\text{shadow}}$                    ▷ threshold
13:         $\alpha' \leftarrow \alpha / k$                                     ▷ Bonferroni correction
14:         for $i \in I$ do
15:             if $z_i > \tau$ then
16:                 $c_i \leftarrow c_i + 1$
17:             end if
18:             $p_i \leftarrow 1 - \mathrm{CDF}_{B(k,1/2)}(c_i - 1)$          ▷ right tail
19:             $p'_i \leftarrow \mathrm{CDF}_{B(k,1/2)}(c_i)$                  ▷ left tail
20:             if $p_i < \alpha'$ then
21:                 $S \leftarrow S \cup \{i\}$                               ▷ select feature $i$
22:             else if $p'_i < \alpha'$ then
23:                 $R \leftarrow R \cup \{i\}$                               ▷ reject feature $i$
24:             end if
25:         end for
26:     end for
27:     return $(\mathbf{\Phi}_{\cdot i})_{i \in S}$
28: end procedure
```

generic and automated approach to generate features for time series, a massive number of time series features can be computed in order to subsequently select the most relevant features for the overall classification task.

For the feature extraction phase, we adopt 63 parametric feature functions[1] (e.g., autocorrelation, kurtosis or skewness) that are pre-defined in the tsfresh package (Christ et al., 2016, 2018). In total, by taking multiple different parameterizations (e.g., different time lags when calculating the autocorrelation) for each feature function, 794 features (tsfresh features) are computed by tsfresh for each time series. In this work, tsfresh was used with its default settings (see Chapter 3.4.1).

From this automatically generated feature space, the most significant tsfresh features are selected in the next step.

4.3.4 Feature Selection

Various feature selection methods have been proposed and applied, e.g., forward, backward and recursive selection (Witten et al., 2011). Importantly, all those feature selection methods require a measure of the feature importance with respect to the modeling task at hand. For instance, in random forests, the importance of a particular feature can be defined as the mean decrease of impurity (e.g., Gini impurity) over all the nodes that split this feature (Breiman, 2001). In the PHCP, a recent feature selection algorithm called Boruta (Kursa and Rudnicki, 2010) is adopted due to the fact that: 1) it is computationally relatively cheap and 2) it has shown best performances when compared to other alternatives (Koch et al., 2018). The Boruta algorithm is summarized in Algorithm 3: in each iteration, it starts with creating so-called “shadow” features by randomly shuffling the values of each feature (line 7-9). A random forests model is trained on the combination of original and shadow features, from which the feature importance is determined (line 11). A feature is considered for selection if its importance dominates the maximum importance of all shadow features (line $12, 15$). The number of times such a dominance occurs is counted for each feature (line 16), on which a binomial test is performed to determine the statistical significance (line 18 - 23).

[1]Please see `https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html` for the detailed list of features.

4.3.5 Training of a Classifier

In this phase the selected Boruta features are used to train a random forest classifier. Of course, any other classifier can be implemented here but due to its simplicity and its efficiency we have used a random forest in our pipeline. Random forests belong to the class of ensemble learning methods and are considered to achieve good performance in different problem domains. A random forest is the conglomeration of many decision trees and the resulting decision is the average outcome of all those decision trees (Hastie et al., 2009).

4.3.6 Hyperparameter Optimization

Setting up hyperparameters properly is vital to the performance of a machine learning model. To boost up the performance of the random forest model on the data, it is necessary to conduct a hyperparameter optimization. The list of hyperparameters under consideration and their ranges are shown in Table 4.1. Note that the resulting search space for hyperparameter optimization contains integer variables as well as categorical ones. There are a variety of well-established algorithms for this task, including grid search, evolutionary algorithms and Bayesian optimization (Geron, 2017).

Table 4.1: Hyperparameter search space for optimizing the random forest classifier.

Parameter	**Range**
Max depth of each tree	$\{1, 2, \ldots, 100\}$
Number of trees	$\{1, 2, \ldots, 100\}$
Max number of features when splitting a node	$\{\texttt{auto}, \texttt{sqrt}\}$
Min number of samples required to split a node	$\{2, 3, \ldots, 20\}$
Min number of samples required in the leaf node	$\{1, 2, \ldots, 10\}$
Use bootstrap training samples?	$\{\texttt{True}, \texttt{False}\}$

In the PHCP, the Bayesian optimization algorithm is chosen due to its efficiency when optimizing expensive problems. In detail, a recent variant of it, called Mixed-integer Parallel Efficient Global Optimization (MIP-EGO) (Wang et al., 2017, 2018) is adopted as it handles the mixed-integer categorical variables in an efficient way. The MIP-EGO algorithm is executed for 200 iterations, where in each iteration a candidate hyperparameter setting is suggested and its goodness is determined by the quality of the corresponding model on a validation set. Using N_{CV}-fold cross validation on the training data, the hyperparameter optimizer uses the average classification quality measure $\bar{f}$ across the N_{CV} validation sets as the overall quality measure for each configuration θ under consideration.

4.3.7 Evaluation of the Results

In the last step of this approach the outcome has to be evaluated. In this context, it is essential to run the computation of each method multiple times and use its average to reduce noise. Sometimes machine learning models tend to overfit / underfit, therefore it is important to evaluate the training performance and the validation performance in detail. Often used evaluation metrics are accuracy and the $F1$-score for class-imbalanced data sets (see chapter 3.1).

Furthermore, visualization methods like, e.g., learning curves, the ROC or the precision / recall curve can provide additional performance insights (see Chapter 3.1).

4.4 Hand-Crafted Approach with Genetic Programming

In the Hand-Crafted Pipeline with Genetic Programming (HCPGP) a feature generation module is implemented in the pipeline. This feature generation is a tree-based genetic programming approach which conglomerates many so-called less significant features (low level features) into several significant features (high level features). In detail, the search domain is defined by symbolic expressions for function approximation (Koza and Rice, 1992; Poli et al., 2008). These symbolic expressions are typically represented as parse trees, such as illustrated in Figure 4.1, where $x_0, x_1, \ldots$ describe the features. The genetic operators are applied to those trees, for example by subtree exchanges between different trees in a population (crossover), or by mutation of subtrees or terminal nodes in the tree.

However, this requires iterations over many generations to find combinations with adequate classification qualities. The large tsfresh feature space limits the efficiency of the computation. As mentioned, this work focuses on competitive

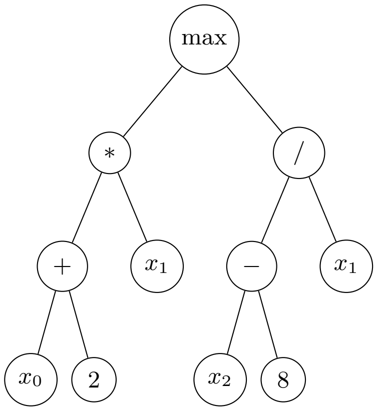

Figure 4.1: An example of a tree-based representation of symbolic expressions, as used in Genetic Programming.

methods regarding performance and computation times. Testing large numbers of combinations is not very efficient here.

Therefore, the dimensionality in terms of the number of initial low level features has to be reduced either way. Promising ways to obtain a smaller set of new features without removing any information would be by using CNNs or a PCA. It is key not to drop information at this point, which is more or less granted in both methods. In this work we are using PCAs mainly due to its lower computation time when compared to CNNs. PCA represents the data in a lower dimension with linear principal components whereas the so-called kernel Principal Component Analysis (kPCA) transforms the data into a higher dimensional feature space and applies PCA on this feature space. Compared to standard PCA, kPCA allows to describe also non-linear behavior (Bishop, 2009).

With an increased prediction scores of approximately 2.5% over all data sets our first experiments showed a slight increase in performance when using kPCA instead of PCA. Therefore, we used kPCA in this work.

Furthermore, in this study we set the number of components of the kPCA to a relatively small number (here 15). This means that the initial tsfresh feature space is reduced to 15 components. Setting the number of components to such a small number enormously improves the efficiency of the genetic algorithm regarding computation time. Especially this boost in efficiency allowed us to investigate the genetic algorithm and its parameters in this study in a computationally feasible way.

When dealing with these algorithms we were facing issues with overfitting, because from our experience such genetic programming algorithms tend to find perfect solutions on the training data by overfitting the data. Therefore, the fitness scores are based on a cross validation approach (see chapter 3.1). Furthermore, as fitness function of our HCPGP approach a performance metric like accuracy is used to maximize the overall performance. This is used together with a minimization of the difference between validation score and training score of the cross validation. This setting aims at reducing the overfitting tendency of the algorithm on the training set of the HCPGP approach.

Table 4.2 shows the parameters of the genetic programming approach. This parameter setting aims at generating a single complex feature based on 15 basic features which can be used as variables in the symbolic expressions generated by genetic programming.

Table 4.2: Parameter of the genetic feature generation algorithm.

Parameter	**Value**
Population	10
Generation	1500
Crossover	0.5
Mutation	0.1
Tree depth	1-10
Number of features	15

The basic features are obtained from applying the tsfresh and kPCA preprocessing algorithms. The performance of the complex features generated by the genetic programming approach is measured by using a random forest and cross validation (see above).

The whole pipeline with genetic programming is shown below:

3. Feature Extraction from Time Series,
4. Kernel PCA,
5. Feature Generation with Genetic Programming,

6. Training of a Classifier and
7. Hyperparameter Optimization.

The pipeline based on genetic programming is shown in Algorithm 4: the input is similar to Algorithm 1 except that this algorithm requires genetic operators ψ and arithmetic operators ζ which are used to construct a parse tree Ω. The feature extraction is executed as presented in Algorithm 1 (line 2-5). The kPCA is applied on the feature matrix (line 3). Then, the feature matrix is split into the training and test matrices $\mathbf{\Phi}_{\text{train}}, \mathbf{\Phi}_{\text{test}}$ respectively (the split of the matrix is done row-wisely). Based on the genetic operator ψ and the arithmetic operators ζ an initial parse tree is constructed for the given population size (here 10).

From line 6-19 the parse tree, which achieves the best cross-validated score of the highest validation performance subtracted by the absolute difference of validation score and training score, is optimized over generations (here 1500). Then, this optimized parse tree Ω_{best} is used to compute its depending features of the split (line 20). These features are considered in the same hyperparameter optimization procedure and validation method as shown in Algorithm 1 (line 8-22).

4.5 AutoML

AutoML generates machine learning pipelines in an automated approach. In this work the following AutoML methods are considered: Auto-Sklearn, TPOT, AMLPA, GAMA, H2O, MLBOX, ATM and RECIPE (see Chapter 3.8). All of these AutoML approaches require stationary features. Therefore, it is implemented after phase 3 (feature extraction):

3. Feature Extraction from Time Series,
4. AutoML,

where phase 4 now replaces the steps 4 - 7 in the pipeline definition given before.

Algorithm 4 Hand-crafted pipeline with genetic programming

```
Require: $\mathcal{X} = \{\mathbf{X}^{(1)}, \dots, \mathbf{X}^{(N)}\} \subseteq \mathbb{R}^{L \times d}$ input time series, $Y \in \mathcal{C}^N$ target labels, $\mathcal{F} = \{F_i\}_{i=1}^p$ feature functions, parse tree $\Omega$, its genetic operator $\psi$, its arithmetic operators $\zeta$, performance metric $f$, a ML algorithm $\mathcal{A}$, its configuration space $\Theta$, an algorithm configurator $\mathcal{H}$ and a genetic programming algorithm GP.
 1: procedure GENETIC-PIPELINE
 2:     Execute lines 2-5 of Algorithm 1
 3:     $\mathbf{\Phi} \leftarrow \text{KPCA}(\mathbf{\Phi}^0, Y)$
 4:     $(\mathbf{\Phi}_{\text{train}}, Y_{\text{train}}), (\mathbf{\Phi}_{\text{test}}, Y_{\text{test}}) \leftarrow (\mathbf{\Phi}, Y)$          ▷ train-test split
 5:     Sample initial parse tree $\Omega$ with $\psi$ and $\zeta$
 6:     $\Omega_{\text{best}} \leftarrow \Omega,\ \bar{f}_{\text{best}} \leftarrow -\infty$
 7:     while the stop criteria are not fulfilled do
 8:         $\{(\mathbf{\Phi}^{(i)}, Y^{(i)})\}_{i=1}^{N_{\text{CV}}} \leftarrow (\Omega(\mathbf{\Phi}_{\text{train}}), Y_{\text{train}})$          ▷ cross validation split
 9:         for $i = 1$ to $N_{\text{CV}}$ do
10:             $\mathbf{\Phi}_{\text{val}} \leftarrow \mathbf{\Phi} \setminus \mathbf{\Phi}^{(i)}, Y_{\text{val}} \leftarrow Y \setminus Y^{(i)}$
11:             $\mathcal{M} \leftarrow \mathcal{A}.\texttt{train}(\mathbf{\Phi}^{(i)}, Y^{(i)}, \theta)$
12:             $\widehat{Y}_{\text{train}} \leftarrow \mathcal{M}.\texttt{predict}(\mathbf{\Phi}^{(i)})$
13:             $\widehat{Y}_{\text{val}} \leftarrow \mathcal{M}.\texttt{predict}(\mathbf{\Phi}_{\text{val}})$
14:             $f_{\text{train,i}} \leftarrow f(\mathbf{Y}^{(i)}, \hat{\mathbf{Y}}_{\text{train}})$
15:             $f_{\text{val,i}} \leftarrow f(\mathbf{Y}_{\text{val}}, \hat{\mathbf{Y}}_{\text{val}})$
16:         end for
17:         $\bar{f}_{\text{train}} \leftarrow \sum_i f_{\text{train,i}}/N_{\text{CV}}$
18:         $\bar{f}_{\text{val}} \leftarrow \sum_i f_{\text{val,i}}/N_{\text{CV}}$
19:         $\bar{f} \leftarrow \bar{f}_{\text{train}} - |\bar{f}_{\text{train}} - \bar{f}_{\text{val}}|$
20:         if $\bar{f} > \bar{f}_{\text{best}}$ then
21:             $\bar{f}_{\text{best}} \leftarrow \bar{f},\ \Omega_{\text{best}} \leftarrow \Omega$
22:         end if
23:         $\Omega \leftarrow \text{GP}(\Omega, \psi, \bar{f})$          ▷ Learn a new parse tree from GP.
24:     end while
25:     $(\mathbf{\Phi}_{\text{train}}, Y_{\text{train}}), (\mathbf{\Phi}_{\text{test}}, Y_{\text{test}}) \leftarrow (\Omega_{\text{best}}(\mathbf{\Phi}), Y)$          ▷ train-test split
26:     Execute lines 8-22 of Algorithm 1
27:     return $\mathbf{\Phi}, \mathcal{M}, \Omega_{\text{best}}, \theta_{\text{best}}, f_{\text{test}}$
28: end procedure
```

4.6 Neural Network Architectures

In this work three neural network architectures for MTSC are considered, namely the hand-crafted architecture Convolutional Neural Network + Long Short-Term Memory (CNN+LSTM), the ResNet (He et al., 2015) and the FCN (Long et al., 2014). Figure 4.2 illustrates these architectures.

The FCN and the ResNet architectures were proposed for time series classification with 2000 epochs (Wang et al., 2016). Initially, both architectures were developed for object detection on images and modified for time series classification with the result of being very competitive regarding other state-of-the-art approaches. Compared to FCNs, a ResNet has a deeper structure by using so-called shortcut connections between layers. ResNets are considered as state-of-the-art in image detection. In a recent publication the time series modification of FCNs showed better results on time series classification tasks compared to ResNets and other architectures (Wang et al., 2016).

Next to FCNs and ResNets, based on mainly real-world time series data (automotive) within this work a hand-crafted neural network has been developed for MTSC by combining CNNs and LSTMs. CNNs can extract important information from the time series and exclude unimportant parts, i.e. CNNs transform the time series into a shorter, high level representation. Furthermore, CNNs learn local patterns, i.e., after learning a pattern this can be found independently of its location on the time axis (see Chapter 3.6). In this setting we are using 2 times 2 CNNs with 128 units or neurons, each followed by a Max Pooling. The number of units can be adapted to the need of the certain data set. However, our real-world data set has 701 time steps and in the optimization phase 128 units showed favorable results. As activation function the Rectified Linear Unit (ReLU) function is used. In detail, the most informative sections are extracted by the CNNs and then, the Max Pooling extracts the maximum value of every 3 successive values. Hence Pooling removes unnecessary information like the exact position of a certain pattern, but the information of a pattern itself still exists, i.e. Pooling thins the data out. This again results in a higher level representation of the data.

However, CNNs do not consider the order of appearance of certain patterns in the data. Therefore, in this setting CNNs are only used to shorten the sequence and to provide this shortened data into a LSTM layer. LSTMs are computationally more

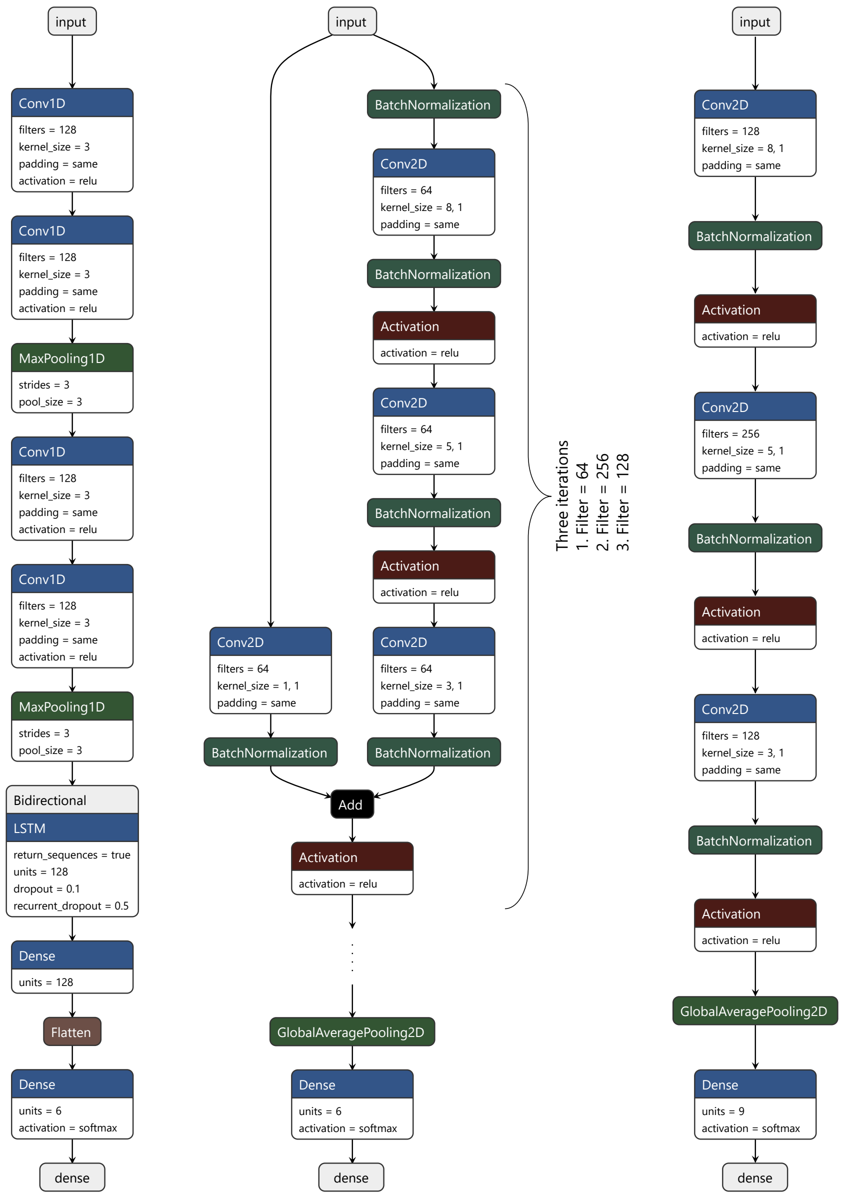

Figure 4.2: In this study three different neural network architectures are used, namely CNN+LSTM (left), ResNet (center) and FCN (right).

expensive than CNNs, but LSTMs do consider the order of values. Therefore, the CNN+LSTM uses CNNs as preprocessor to reduce the long sequence with keeping only high-level features. In this way, the sequence can be shortened into a highly informative sequence, which is processed by the LSTM. The LSTM returns the extracted sequence to a dense layer which is a fully connected layer. The output of the dense layer is flattened in order to be processed in the last dense layer. This last layer contains a softmax activation function. For each class in the data set exists one unit. The output of all units must add up to 1.0, i.e. the output of each individual class unit shows the probability of this class to be the target.

The hyperparameter of this Neural Network has been tuned for the real-world automotive data set. The LSTM has 128 units, a dropout of 0.1 and a recurrent dropout of 0.5. Dropout is a so-called regularization method. It randomly drops units to prevent overfitting and the lower number of units allows faster model training. The *normal* dropout regularizes the inputs between layers, whereas the recurrent dropout drops units of the recurrent state, i.e. it regularizes between different time steps. Furthermore, in this setting 16 data points are used at once to update the model parameters, i.e. the batch size is 16. All data are passed up to 2000 times through the network. This are the so-called epochs. The learning rate is set initially to 0.001 and when the objective function stagnates it is reduced down to 0.0001. In order to reduce computation time, an early stopping criteria is defined to stop training when experiencing no further improvement in the objective function.

CHAPTER 5

Solving Real-World Problems

This chapter presents the first application of the plain hand-crafted pipeline on three real world data sets, indicated as case A, B and C. The objective of these applications was the development, verification and optimization of our basic pipeline (PHCP), as well as the investigation of its usefulness for solving real world problems. The data set of case A contains on-board car data of 417 crash tests. This is used to develop a model which predicts the crash impact point around the vehicle. Case B aims at estimating the damaged parts caused by a low speed crash event. The data set contains on-board data of 100 crash tests carried out within this work. Based on electroencephalography data from 40 Parkinson disease patients, in case C, a model is developed to predict the usefulness of a Deep Brain Stimulation to improve the well-being of the patients.

5.1 Case A: Predicting the Vehicle Crash Impact Point

5.1.1 Introduction

A big share of traffic accidents takes place in a difference in velocity of less than 16 km/h (RCAR, 2018), the so-called low-speed crash. Nowadays, in the car insurance industry, there is a growing demand for prompt transparency of such low speed crash events. The main reasons are to increase the speed of insurance claim processing and to reduce claim frauds. Many companies and some researchers are developing approaches to generate valid information from crash events (Carroll, 2018; Patil et al., 2017). Recent developments are e.g. chatbots, which question the customer about a crash and initiate the claim settlement, or image recognition methods, which estimate the costs based on pictures of the broken parts (Carroll, 2018). Some companies use additional devices, which are communicating with the car via e.g. OBD[1]. Those devices can send information to an insurance company after a crash event (Welt, 2018).

In this application on-board car data from crash events is used to predict the point of impact. Such an application could be beneficial for insurance claims due to the provided prompt transparency of the impact location. Therefore, not only driving vehicles but also parked vehicles which are involved in a crash event can provide more detailed crash information to the customer and, if desired, to the insurance company. Furthermore, such a system can avoid claim fraud and autonomously initiate repair processes.

Based on 417 crash tests, 8 impact points of the car have been defined (see Figure 5.1). In the research presented here, a classifier is developed to predict the impact points. Features from on-board time series data are automatically extracted, selected and used for learning a decision tree classifier.

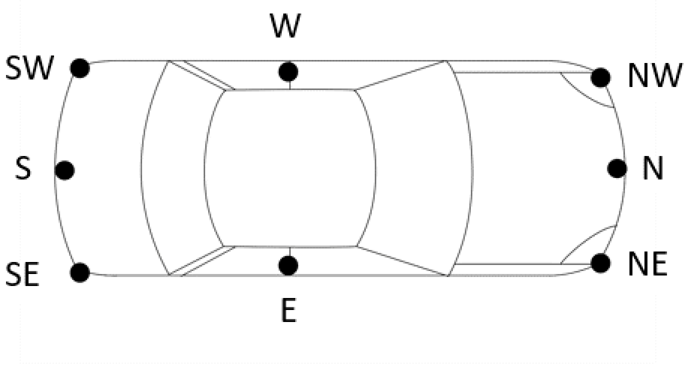

Figure 5.1: The 8 impact points of the car.

5.1.2 Related Work

Using on-board car data, without adding any additional sensors, to generate more transparency of low-speed crash events has not been a major topic of scientific

[1]On-board diagnostics (OBD) is developed as an interface to give the repair technician in a workshop access to the status of the car systems.

research yet. Existing research uses additional sensors for deriving information about damages from structure-borne sounds (Gontscharov et al., 2014). Other researchers discuss the use of image recognition methods with deep learning techniques in order to generate information based on images of crashed vehicles (Patil et al., 2017; Harshani and Vidanage, 2017).

5.1.3 Data Set Description

The data set used in this research contains time series data (e.g. acceleration) from 421 car-to-car crash events with many different car types. Figure 5.2 shows such a typical longitudinal acceleration signal of a crash event.

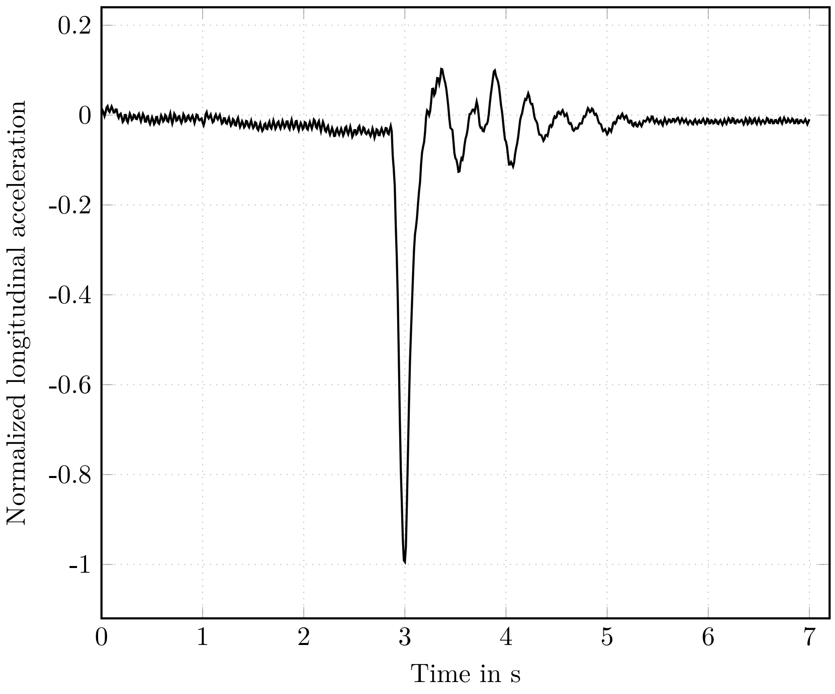

Figure 5.2: A typical, longitudinal acceleration signal (measured at the center of gravity) from a low speed crash.

Each event is manually labeled with the corresponding point of impact: Figure 5.1 illustrates those 8 impact locations. Furthermore, the number of cases per location is shown as high-level description in Table 5.1 which indicates that the number of cases is not balanced over the locations, i.e. the data set is class imbalanced. This is important to have in mind when exploring and analyzing the data.

Table 5.1: Description of the data set.

Class	Location	Cases
0	NW	71
1	N	112
2	NE	87
3	E	20
4	SE	22
5	S	72
6	SW	22
7	W	15

5.1.4 Modeling Approach

This study is the first application on a real-world data set of the methodological approach shown in Chapter 4.3. Therefore components of this pipeline differ from the final PHCP, e.g., a grid search algorithm is used for hyperparameter optimization instead of the Bayesian algorithm. The particular pipeline of this case is described in detail below.

First of all, the recorded raw data from the car has to be preprocessed for further steps. The data is cleaned from corrupted cases and, if necessary, the units are converted. Usually, the traces are capturing much longer time lines than needed for the algorithm. Therefore, the time series are cut into time windows, which catch the crash events only. One of the most time consuming steps, within the preprocessing, is the labeling process. As shown in Table 5.1 each case is assigned to one location (label). In order to generate a valid database, the assignment of the location to each case needs to be done manually.

In the next step, an exploratory data analysis step is used to obtain some data understanding and to choose the right approach for modeling. As mentioned, a typical longitudinal acceleration of the investigated crash events is shown in Figure 5.2: the first contact of both cars happened slightly before the largest acceleration value at $t \approx 2.9s$. This acceleration signal quickly reaches a maximum ($t \approx 3.0s$) followed by a damped oscillation towards zero. However, all curves of the crash events are analyzed in this manner and typical behaviors are derived. As an example, Figure 5.3 shows a pattern that has been found through the exploration. It illustrates the normalized maximum acceleration values, extracted from the time series, of the vertical direction (y-axis) as a function of the normalized longitudinal direction (x-axis). The different colors indicate the different impact locations. The rough pattern N/E/S/W can be easily observed. However, exact locations, e.g. NW/N/NE, cannot be distinguished by those maximum acceleration values. Therefore, better representations than maximum acceleration values have to be extracted from the time series.

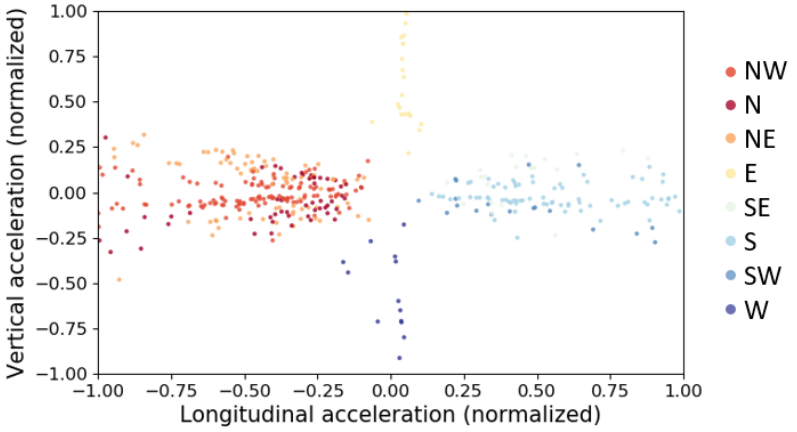

Figure 5.3: Normalized maximum values of longitudinal- and vertical acceleration with a color indication of the crash location.

5.1.4.1 Feature Extraction from Time Series

The feature extraction is done by the library tsfresh (Christ et al., 2018). As Chapter 4.3.3 states tsfresh computes a wide range of different features. It deduces 794 features per time series. For this approach, 4 time series are used per event, i.e. a total of 3,176 features are created. Some of these features can possibly describe the relation to the impact location better than others. Therefore, in the next step the most important features need to be selected.

5.1.4.2 Feature Selection

Most important features are selected with the Boruta algorithm. For a detailed description of this algorithm please refer to Chapter 4.3.4.

5.1.4.3 Training a Decision Tree

A decision tree classifier is used in this application for realizing a fast and comprehensible approach. To find the best hyperparameters, a simple random sampling approach with 50,000 samples is employed, using the 10-fold cross-validated prediction accuracy as performance score.

Table 5.2: The optimized hyperparameters of the decision tree and the range of values used for randomized search.

Parameter	Optimized values	Range
Criterion	entropy	$\{\text{gini}, \text{entropy}\}$
Max depth	6	$\{1, 2, \ldots, 40\}$
Max features	9	$\{\text{None}, \text{auto}, \text{sqrt}, 1, 2, \ldots, 25\}$
Max leaf nodes	23	$\{2, 3, \ldots, 50\}$
Min samples leaf	2	$\{1, 2, \ldots, 10\}$

The investigated range of hyperparameters and the best setting are presented in Table 5.2[1]. Figure 5.4 shows the feature importance or variable importance. Feature importance describes the relative importance of each feature. In detail, for each feature the improvement of the split-criterion over all splits is added up. These values are scaled that the sum of all importances is equal to 1. This describes the relative importance of the features (Hastie et al., 2009).

The Boruta algorithm selected 25 features. By means of the randomized search with the decision tree as classifier, from those 25 features 9 are found to be essential. The range of the relative importance is between 3-30%. The features are described in Table 5.3 according to the definitions provided for tsfresh (see tsfresh (2018)). In general, 4 different types of features are selected: 6 times fourier coefficients calculated by fast fourier transformation, the location of the first maximum, the standard deviation and the approximated entropy. It should be noted, that the

[1]Table 5.2 shows the most important hyperparameters for this application. Hyperparameters with a low influence on the result have been tested but are not shown in this study.

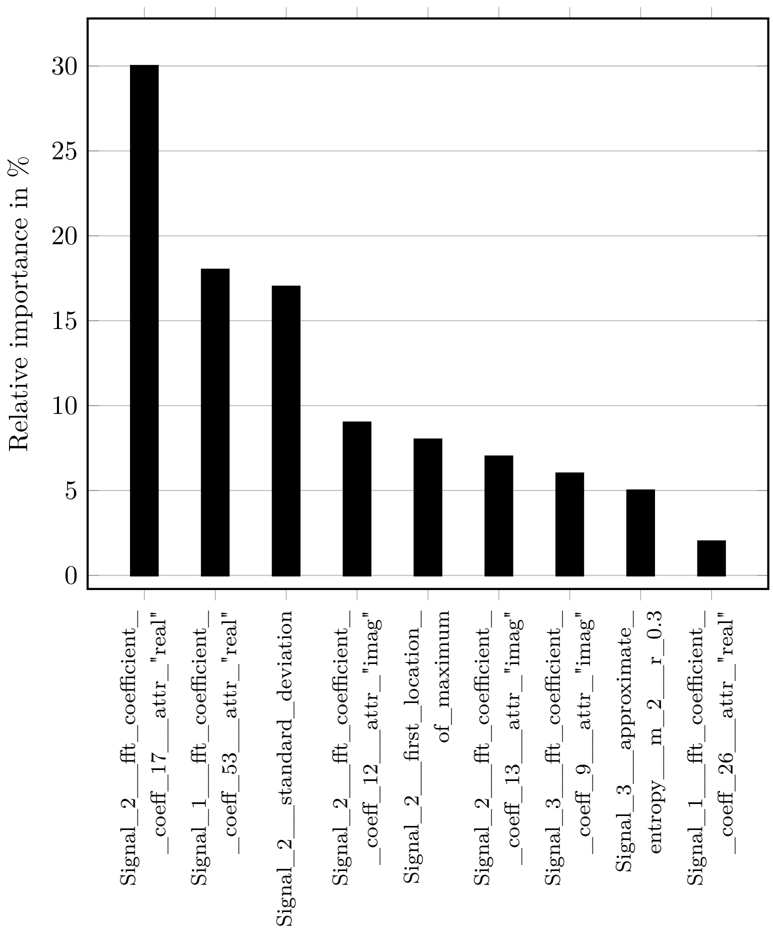

Figure 5.4: The relative importance of the selected features.

automatic approach of this study provides features, which would have been virtually impossible to find by manual selection methods.

The average of the 10-fold cross-validated accuracy score is 76%. Due to the large size of the decision tree only a cutout of the first nodes is illustrated in Figure 5.5. The basic principle of decision trees is explained in Chapter 3.3. The decision tree is drawn from upside down, which means that the root node is at the top. Each horizontal layer of nodes represents the depth of the tree, i.e. the root node is at depth 0 and the horizontal layer below is depth 1 and so on. The root node contains a condition in its first line. If this condition is true, the branch down to

Table 5.3: The description of the selected features.

Name	Description
Fft coefficient	"The fourier coefficients of the one-dimensional discrete Fourier Transform for real input by fast fourier transformation algorithm." (tsfresh, 2018) This feature appears 4 times with the following coefficients: 9, 12, 13, 17, 26, 53
First location of maximum	"The first location of the maximum value of the time series x. The position is calculated relatively to the length of x." (tsfresh, 2018)
Standard deviation	"The standard deviation of x." (tsfresh, 2018)
Approximate entropy	"A vectorized approximate entropy algorithm." (tsfresh, 2018)

the left is active – if false, the branch to the right. In the next step the condition of the node, connected to the active branch, is checked, and so on. In the second line of each node the entropy is shown. The entropy describes the homogeneity of the samples, which means when having completely homogeneous samples the entropy is zero. The more different samples are existing in a node the higher the entropy value (Geron, 2017). The third line of each node shows the number of samples which reach the node. The class affiliation of those samples is displayed in the fourth line. The last line indicates the predicted class of this node. The end of a branch, represented here with the blue and the red rectangle, is called leaf or decision. In this case, it is the indication at which point of the car the crash happened (Geron, 2017; Pedregosa et al., 2011). The whole decision tree is shown in the Appendix A.

In the following section 5.1.5 the performances of the decision tree are discussed.

5.1.5 Results

A first representation to evaluate the performance of the trained decision tree is shown in Figure 5.6, the so-called learning curve (see Chapter 3.1). Each

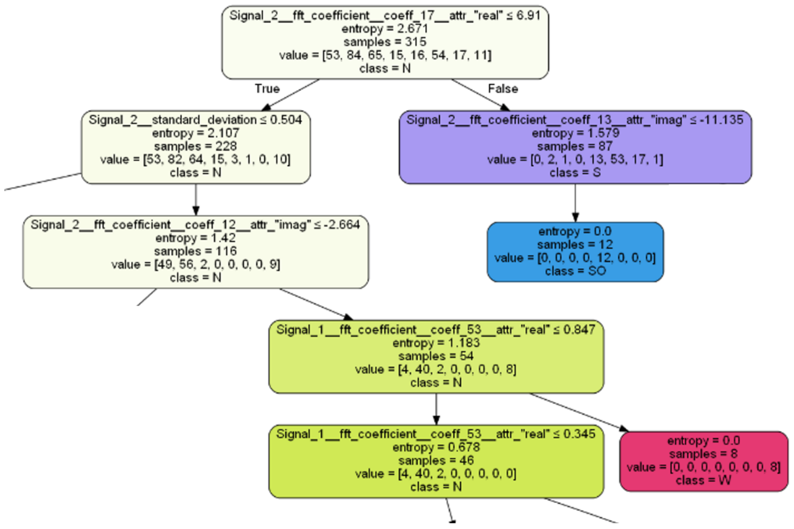
Signal_2__fft_coefficient__coeff_17__attr_"real" ≤ 6.91
entropy = 2.671
samples = 315
value = [53, 84, 65, 15, 16, 54, 17, 11]
class = N
True
False
Signal_2__standard_deviation ≤ 0.504
entropy = 2.107
samples = 228
value = [53, 82, 64, 15, 3, 1, 0, 10]
class = N
Signal_2__fft_coefficient__coeff_13__attr_"imag" ≤ -11.135
entropy = 1.579
samples = 87
value = [0, 2, 1, 0, 13, 53, 17, 1]
class = S
Signal_2__fft_coefficient__coeff_12__attr_"imag" ≤ -2.664
entropy = 1.42
samples = 116
value = [49, 56, 2, 0, 0, 0, 0, 9]
class = N
entropy = 0.0
samples = 12
value = [0, 0, 0, 0, 12, 0, 0, 0]
class = SO
Signal_1__fft_coefficient__coeff_53__attr_"real" ≤ 0.847
entropy = 1.183
samples = 54
value = [4, 40, 2, 0, 0, 0, 0, 8]
class = N
Signal_1__fft_coefficient__coeff_53__attr_"real" ≤ 0.345
entropy = 0.678
samples = 46
value = [4, 40, 2, 0, 0, 0, 0, 0]
class = N
entropy = 0.0
samples = 8
value = [0, 0, 0, 0, 0, 0, 0, 8]
class = W

Figure 5.5: A cutout of the first leafs of the decision tree.

prediction is 10-fold cross-validated. This training curve increases in the beginning and then declines smoothly to 89%. The cross-validated curve is quickly rising in the beginning and then slowly growing until 76%. The gap between the curves indicates that the model, when trained on all data records, is overfitting. However, even with more than 350 training examples the gradient of the cross-validated curve is still positive, which leads to the conclusion that more data would increase the validation score and therefore reduce overfitting (Pedregosa et al., 2011).

Figure 5.7 shows the confusion matrix (see Chapter 3.1)[1]. It is obvious that some misclassifications are observed for neighboring locations e.g. 0,1,2. Figure 5.3 indicated already that close locations are difficult to distinguish. Furthermore, the data set represents different car types, e.g. all-terrain vehicles have different characteristics in the time series than coupés. Therefore, features have to be found, which describe the generalized characteristics of all used car types. Another reason for misclassification results from the class imbalance, with four of the eight classes being represented by only about 5% of the data points, each. This can especially be seen in classes 4 and 6. While in general the results are already quite acceptable,

[1]The location of each class is shown in Table 5.1.

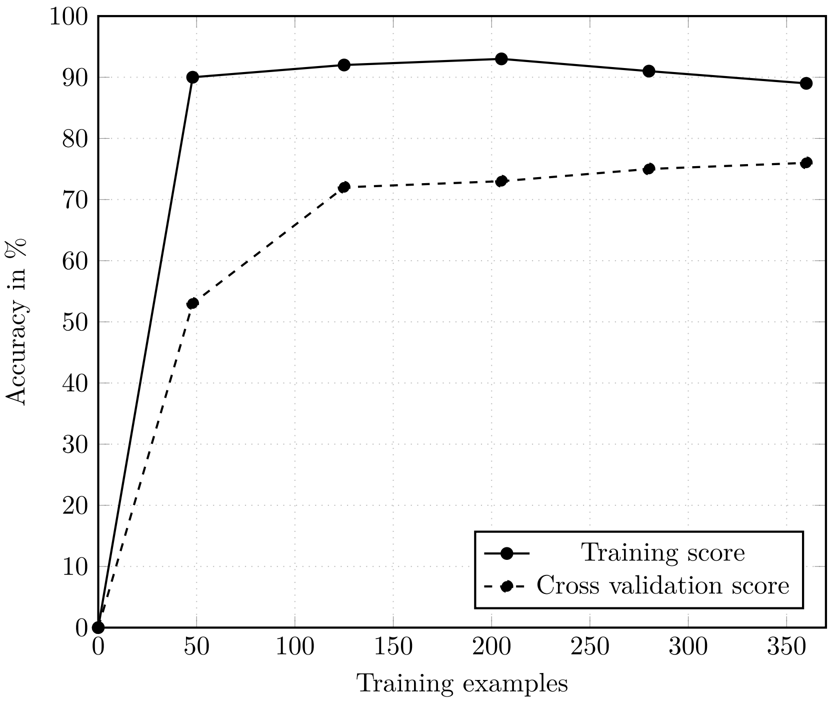

Figure 5.6: The learning curve shows the increase in accuracy with more experience of the modeled decision tree.

separating close locations accurately will likely require more training data and better features (Pedregosa et al., 2011).

5.1.6 Summary and Conclusions for Case A

The evaluation of the described approach demonstrates that 9 automatically computed and selected features (out of a total of 3,176) are sufficient to obtain a decision tree with prediction accuracy of 89% for predicting the location of a low speed vehicle crash, given the sensor signal time series data available in modern passenger cars. This study presents an automatic approach to use time series for machine learning with good performance and promising findings, even on such a small and class imbalanced data set. To further improve the results, the feature set could be enhanced by manual feature selection, because, e.g., the relation between features of different time series is not considered in this approach. Using techniques

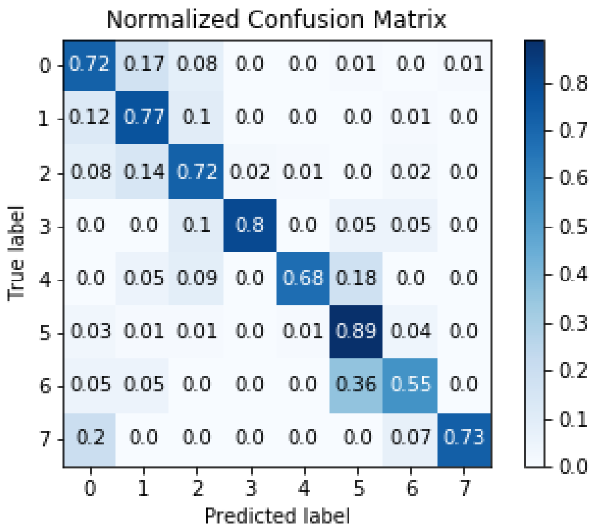

Figure 5.7: The normalized confusion matrix illustrates true class vs. predicted class of each label. Deviations from the highlighted diagonal represent misclassifications.

to decrease class imbalances, e.g., generating more test data, adding copies of instances from the under-represented classes (resampling) or producing synthetic samples from the existing data, provide space for further improvement. More sophisticated modeling algorithms could also potentially improve the performance, however, most likely at the cost of losing the explicative model properties provided by decision trees. To accelerate the search for optimal hyperparameters, as a next step the random search approach will be replaced by an advanced search algorithm, e.g., efficient global optimization for mixed-integer categorical search spaces (Wang et al., 2017, 2018).

5.2 Case B: Predicting the Damaged Parts of a Vehicle Crash

5.2.1 Introduction

In this case, we focus on predicting the damaged parts of a low speed vehicle crash. Such an application could provide prompt transparency of the damaged parts of vehicles involved in a low speed crash (parking or driving). If requested, detailed crash information can be provided to e.g. initiate repair processes efficiently and to avoid claim frauds.

In this study, only the data from installed sensors of serial cars is used. In order to evaluate the possibilities of such a system, crash tests are carried out and the on-board sensor data (e.g. acceleration) is recorded. After each test, the damaged parts of the vehicles are identified. The data set contains 100 cases. Based on 5 time series from 5 sensors, the PHCP approach is further developed to use time series data for Machine Learning in an optimized way.

5.2.2 Related Work

The use of on-board car data, without adding any additional sensors, to generate more transparency of crash events promptly has not been a major topic of scientific research yet. In order to derive the damages, Gontscharov et al. (2014) apply additional sensors for using structure-borne sounds. Other recent publications discuss the use of image recognition methods to generate information from the crashed vehicle after the accident with deep learning techniques (Patil et al., 2017; Harshani and Vidanage, 2017).

5.2.3 Data Set Description

Table 5.4 shows a high-level description of the data set. It consists of a total of 100 cases from vehicle to vehicle low speed crash tests with almost identical vehicles. The cases are distinguished by positions (e.g. front / rear / side / ...), the angle between the vehicles and the velocities. Each data set includes recorded time series sensor data from the crash events (e.g. acceleration) and the damaged parts of the crash. In this investigation, 14 car parts are used for modeling to show the developed method. Table 5.4 indicates those parts and their corresponding number of damages. The cases are not balanced between undamaged and damaged due to the very different crash scenarios (positions, angles, velocities, ...). This class-imbalance and especially the parts with a minor number of damages should be kept in mind when creating the models and evaluating the results.

Table 5.4: Description of the data set: the number of damages per part are shown. The data of 100 crash tests is used.

No.	Part	Undamaged	Damaged
1	Bumper front	57	43
2	Crossmember front (upper)	80	20
3	Crossmember front (lower)	89	11
4	Impact absorber front	77	23
5	Sidewall front (left)	93	7
6	Sidewall front (right)	90	10
7	Bonnet	92	8
8	Headlight (left)	86	14
9	Headlight (right)	83	17
10	Bumper rear	59	41
11	Crossmember rear	86	14
12	Tailgate	92	8
13	Rocker panels (left)	92	8
14	Rocker panels (right)	94	6

5.2.4 Modeling Approach

The modeling approach of this application follows the methodological steps introduced in Chapter 4.3. Based on the pipeline approach of case A, the PHCP is developed further, e.g., a Bayesian hyperparameter optimization technique is introduced, a random forest is used instead of a decision tree and compared to the grid search algorithm, as well as different feature selection methods are compared.

The data set contains time series and a matrix with the damaged parts per crash test. The information containing the time series has to be revealed by extracting characteristic features from the time series. The damaged parts could be treated as vector for each crash test, with $\vec{y} = (y_1, ..., y_k)^T$. This binary vector $\vec{y}$ represents the status, with part i being undamaged (damaged) if $y_i = 0$ ($y_i = 1$). For this problem representation, a so-called multi-label classification approach could be used, which allows for learning a vector of labels by learning one classification model for all parts with one set of features. Hence, the damages for all parts are predicted by one model. This could be very promising for some cases, but for our damage prediction model with this small data set, it does not yield good results. Moreover, there is little room for improving the model through hyperparameter optimization. Therefore, we developed the so-called part-wise approach: for each car part one individual model with specific features and hyperparameters is generated. This transforms the multi-label classification into a regular single binary label classification. It also provides opportunities for optimization of every single model, in order to create best possible models with a small data set.

In the following sections, the individual steps of the methodological approach of this case are discussed in detail.

5.2.4.1 Preprocessing

First of all, the recorded raw time series data from the crash events has to be preprocessed. Generally, vehicle sensors shake slightly in the mounted position depending on, e.g., the road conditions. It means, that the positions of the axes change relatively to the defined vehicle coordination system of longitudinal, transversal and vertical axes. Therefore, the data has to be corrected in order to relate the sensor data to the vehicle coordination system. Furthermore, the sampling rate of the data is very high, which results in a noisy signal. The signals are filtered to remove such noisy components. In our case, the correction and the filtering is done within the control units of the car. In the next step, the recorded signals are verified visually and corrupted cases are deleted. Furthermore, the signals are recorded with longer timelines due to the experimental conditions: therefore, the time windows of all time series are shortened and adjusted to the same length. If necessary, units are converted. In the last step of the preprocessing, the labels (damaged/undamaged) of each part and crash test are collected and digitalized for further processing.

5.2.4.2 Exploratory Data Analysis

In the exploration step, the data set is manually explored to comprehend the data and to find the best approaches for modeling. Figure 5.2 shows a typical acceleration time series from a crash event. Some characteristic features, e.g., minimum, maximum or the number of minima and maxima can be found quickly by visual analysis. However, there could be many other important features that cannot be discovered by manual inspection.

5.2.4.3 Feature Extraction from Time Series

The feature extraction is done by tsfresh (Christ et al., 2018) (see Chapter 4.3.3). In this study we use 5 recorded time series from the crash event, which leads to 3970 features. Additionally, we consider 12 manually created features, namely the obvious ones that were found through the exploration process. Table 5.5 shows these manual selected features.

Table 5.5: The list of the manually selected features.

No.	Name
1-3	Min. of time series 1, 2, 3
4-7	Max. of time series 1, 2, 3, 4
8-9	Absolute (Abs.) max. of time series 1, 2
10	$\sqrt{(\text{Abs. max. of time series 1})^2 + (\text{Abs. max. of time series 2})^2}$
11	$\frac{\text{Abs. max. of time series 1}}{\sqrt{(\text{Abs. max. of time series 1})^2+(\text{Abs. max. of time series 2})^2}}$
12	$\frac{\text{Abs. max. of time series 2}}{\sqrt{(\text{Abs. max. of time series 1})^2+(\text{Abs. max. of time series 2})^2}}$

5.2.4.4 Feature Selection

As indicated above, a total of 3982 features are generated. In order to find the most important ones, which are suitable for modeling, for each part, feature selection methods or subset selection algorithms are used. There are different common methods and algorithms to select the significant features. In this study, two very promising wrapper algorithms, called Boruta (Kursa and Rudnicki, 2010) and RFECV (Guyon et al., 2008) are used and compared (see Chapter 3.4.3).

In this study, the Boruta algorithm is run with $n_{\text{estimators}} = 1000$ and $max_{\text{iterations}} = 100$. The hyperparameters of RFECV are $step = 100$, $cv = 10$ and $scoring = F_1$. Both algorithms use a random forest classifier. As previously described, each car part has its own features, which are selected by these algorithms and then used for classification (damaged/undamaged). Table 5.6 shows the comparison of the feature selection algorithms Boruta and RFECV on this data set without

Table 5.6: Comparison of the feature selection methods Boruta and RFECV.

	Boruta			RFECV		
Part	**Features**	**Accuracy**	**F1 score**	**Features**	**Accuracy**	**F1 score**
Bumper front	5	0.91	0.91	82	0.95	0.94
Crossmember front (upper)	4	0.85	0.63	1	0.83	0.60
Crossmember front (lower)	1	0.85	0.29	1	0.82	0.25
Impact absorber front	4	0.89	0.72	1	0.82	0.60
Sidewall front (left)	1	0.89	0.27	1735	0.93	0.00
Sidewall front (right)	0	0.00	0.00	1	0.83	0.19
Bonnet	6	0.94	0.50	1784	0.92	0.00
Headlight (left)	0	0.00	0.00	1	0.81	0.30
Headlight (right)	4	0.88	0.61	1	0.88	0.67
Bumper rear	4	0.88	0.85	82	0.91	0.88
Crossmember rear	4	0.88	0.55	1	0.86	0.59
Tailgate	1	0.88	0.14	1861	0.92	0.00
Rocker panels (left)	7	0.96	0.75	1	0.91	0.47
Rocker panels (right)	4	0.93	0.46	1	0.92	0.43
Computation time	20min			4h 15min		

hyperparameter optimization. Individual random forest classifiers are trained per part (training a random forest classifier). Table 5.6 illustrates the number of features, the accuracy and the F_1 score of both selection algorithms (see Chapter 3.1). The accuracy for most of the parts is higher than 80%. This could lead to the conclusion that the performance is quite good. However, the accuracy of e.g. the sidewall front (left) is 89% and 93% and there are 7 damages of this part in total, only. In this case a possible scenario could be, that just all of the 93 undamaged cases are predicted correctly, but the 7 damaged cases are predicted wrongly. As in Chapter 3.1 introduced, when dealing with such a class-imbalanced data set, another performance measure other than accuracy is needed, e.g., the so-called F_1 score.

The comparison of the F_1 score of Boruta and RFECV (see Table 5.6) shows mostly similar results for both algorithms but in total Boruta yields better results. The scores vary between 0% and 94%. Obviously, the performance of the models

Table 5.7: Computer Specifications.

Feature	**Specification**
Processor	Intel Core i7-6600U CPU @ 2.60GHz
Operation System	Windows 10 64bit
Graphics	AMD Radeon R7 M365x
RAM Memory	8 GB
Storage	SSD 256 GB

depends strongly on the part. Furthermore, the number of selected features is in most of the cases very different: especially for parts with a low performance like bonnet or sidewall front (left), the number of features varies significantly between the two algorithms. In these cases, RFECV finds a much larger number of features than Boruta with a worse model performance. Therefore, Boruta seems to select features in a more efficient way. Another important criterion for comparison is the computational effort. Table 5.7 shows the specifications of the used computer in this work. We notice that Boruta is about 3h 55min faster than RFECV. This leads to the conclusion, that for such an application Boruta is a better choice, mainly due to its performance and efficiency. Therefore, we are using Boruta in the following.

5.2.4.5 Hyperparameter Optimization

In the next step, we consider two different algorithms for optimizing the hyperparameters of the random forest classifiers: a randomized search (Geron, 2017), and a package named MIP-EGO (Wang et al., 2017, 2018). Randomized search is a simple approach for optimizing hyperparameters. This kind of algorithm picks features randomly from a defined feature space (Geron, 2017). Therefore, it is not always very efficient due to the randomness. In order to increase the efficiency of this approach, we are running a Bayesian optimization with the MIP-EGO (Wang et al., 2017, 2018) for mixed-integer categorical search spaces on our data set and compare the results. MIP-EGO is developed to solve expensive global optimization problems.

Table 5.8 illustrates the results of MIP-EGO with 200 iterations and randomized search with 200 iterations. Both algorithms use the cross-validated F_1 score as the objective function. Table 5.9 shows the random forest hyperparameters and

Table 5.8: Comparison of the hyperparameter optimization methods MIP-EGO and Randomized Search.

		MIP-EGO		Randomized	
Part	**Features**	**Accuracy**	**F1 score**	**Accuracy**	**F1 score**
Bumper front	5	0.93	0.94	0.94	0.93
Crossmember front (upper)	4	0.89	0.74	0.90	0.74
Crossmember front (lower)	1	0.91	0.40	0.90	0.44
Impact absorber front	4	0.88	0.70	0.89	0.74
Sidewall front (left)	1	0.95	0.40	0.91	0.31
Sidewall front (right)	0	0.00	0.00	0.00	0.00
Bonnet	6	0.95	0.53	0.90	0.36
Headlight (left)	0	0.00	0.00	0.00	0.00
Headlight (right)	4	0.84	0.59	0.90	0.67
Bumper rear	3	0.88	0.81	0.89	0.85
Crossmember rear	3	0.89	0.71	0.90	0.71
Tailgate	1	0.93	0.36	0.93	0.36
Rocker panels (left)	7	0.96	0.80	0.96	0.57
Rocker panels (right)	1	0.92	0.62	0.93	0.53
Computation time		1h 7min		48min	

their possible range or categorical parameter choices, respectively. Table 5.8 shows that especially the higher F_1 scores are in a similar range, i.e., this outcome of both algorithms is roughly comparable. However, the MIP-EGO algorithm is achieving considerably better F_1 scores for most of the lower performing models (e.g. bonnet or rocker panels (right)). Such behavior is especially essential on small data sets: it approximates the model performance to the maximum scores, which displays more clearly the whole potential of the data set. This shows, that MIP-EGO is capable of optimizing in far less iterations than a random search. In conclusion, the MIP-EGO algorithm is slightly more expensive (19 min) but yields far better results for this optimization problem than a randomized search approach. Therefore, the MIP-EGO algorithm is used in the following. The evaluation of the results is discussed in the following section.

Table 5.9: Hyperparameter space for optimizing the random forest classifier.

Parameter	**Range**
Max depth	$\{1, 2, \ldots, 40\}$
Estimators	$\{1, 2, \ldots, 100\}$
Max features	$\{\text{auto}, \text{sqrt}\}$
Max samples split	$\{2, 3, \ldots, 20\}$
Min samples leaf	$\{1, 2, \ldots, 10\}$
Bootstrap	$\{\text{True}, \text{False}\}$

5.2.5 Results

Table 5.10 shows the overall results of the final approach. Out of the 3970 generated features and the 12 manually created features, up to 7 features are selected for the part-wise models. The hyperparameters of the random forest classifiers have been optimized using efficient global optimization with 200 objective function evaluations.

Based on the small data set, the F_1 score of 7 parts is quite acceptable ($F_1 > 61\%$). Most of the other parts show a tendency to higher scores: Figure 5.8 illustrates the learning curve of the crossmember front (lower). The significant gradient in the end of the curves shows, that more data would lead to even higher scores. This observation is made for almost all low performing parts. In general, this optimized approach delivers good results for many parts, but for some parts the number of damages is just too low: no relevant features are found for the sidewall front (right) and headlight (left). This means that none of the 3982 features is able to describe the relation to the damage. It is very likely, that the prediction of damages of some parts is difficult, e.g. due to the fact that these are small components. In addition, the mounted location and low connection stiffness can result in not tracking relevant information in the sensor signals. For these parts, feature representations from other sensors could be beneficial for a better prediction. However, the low performance is mainly based on the small data set and especially on the small number of damages of parts (class-imbalance). A lower class-imbalance, especially for parts with more than 20 damages, leads to better performing models like for the bumper (front/rear), crossmember front (upper) or impact absorber (front).

Table 5.10: The final result of the part-wise approach.

Part	**Features**	**Accuracy**	**F1 score**	**Precision**	**Recall**	**ROC**
Bumper front	5	0.93	0.94	0.93	0.95	0.95
Crossmember front (upper)	4	0.89	0.74	0.70	0.80	0.86
Crossmember front (lower)	1	0.91	0.40	0.75	0.27	0.63
Impact absorber front	4	0.88	0.70	0.70	0.70	0.80
Sidewall front (left)	1	0.95	0.40	0.67	0.29	0.64
Sidewall front (right)	0	0.00	0.00	0.00	0.00	0.00
Bonnet	6	0.95	0.53	0.57	0.50	0.73
Headlight (left)	0	0.00	0.00	0.00	0.00	0.00
Headlight (right)	4	0.84	0.59	0.59	0.59	0.75
Bumper rear	3	0.88	0.81	0.82	0.80	0.84
Crossmember rear	3	0.89	0.71	0.60	0.86	0.88
Tailgate	1	0.93	0.36	0.67	0.25	0.62
Rocker panels (left)	7	0.96	0.80	0.86	0.75	0.87
Rocker panels (right)	1	0.92	0.62	0.50	0.83	0.89
Time				1h 7min		

5.2.6 Summary and Conclusions for Case B

This study proposes an automatic and efficient way to use time series data for machine learning. The tsfresh package calculates a wide range of features, the Boruta algorithm helps finding the most valuable features and MIP-EGO optimizes the hyperparameters of the machine learning algorithm (random forests in this case) efficiently. The approach is especially useful for small and difficult data sets, because even on the presented small data set with a significant class-imbalance, the resulting scores are quite promising. Part models with a lower performance show a large potential indicating that more data would improve the results. The part-wise approach proposed here could also be combined with methods like frequent pattern mining in order to identify which parts are likely damaged together within one crash.

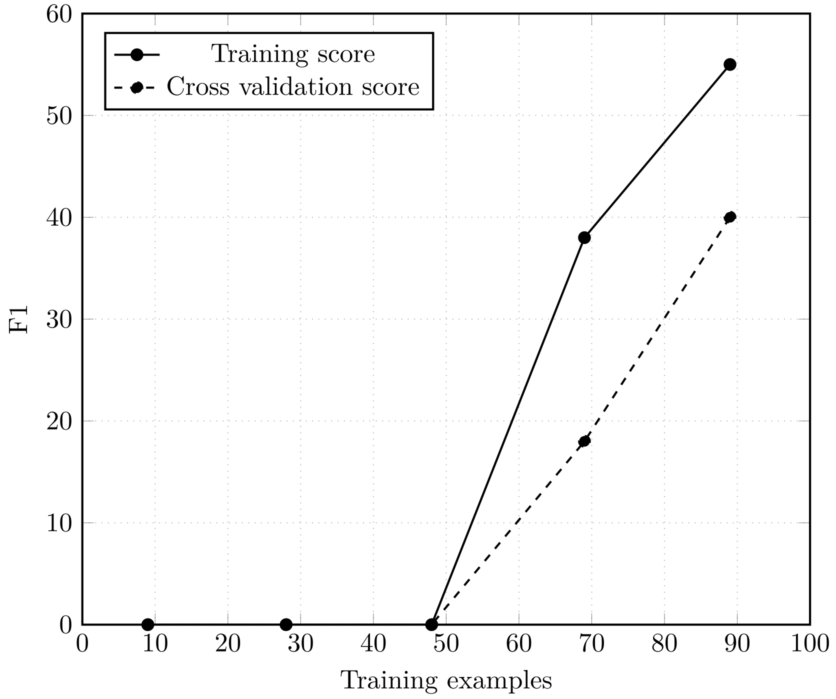

Figure 5.8: The learning curve of the crossmember front (lower). The significant gradient in the end of the curve indicates better performing models when using more data.

5.3 Case C: Automated Machine Learning for Parkinson Disease Patients

5.3.1 Introduction

Based on EEG-profiles and machine learning techniques, the objective of case C is to classify PD patients as having either a *good cognitive performance* or a *poor cognitive performance*. In this context, it is proposed to use and compare both conventional EEG biomarkers (clinical features) and features that are automatically extracted from EEG series (including potentially new biomarkers). The models are trained on patients with either clinically determined good or poor cognitive performance. The evaluation of cognition of the data is based on several clinical scores, including the SENS-PD-COG scores and the MoCA scores which are described below. The data set contains EEGs of 125 patients. For the research reported here, the 40 most extreme patients of each group in terms of *good cognition* and *poor cognition* were selected, i.e., 20 patients with a good cognition and 20 patients with a poor cognition.

In case of promising results, our approach could reveal new biomarkers, as well as provide a machine learning model that may be used to complement the screening procedure on cognitive assessment.

To show the background and the objective in detail, a more medical introduction follows. Parkinson's disease (PD) is considered the second most common neurodegenerative disorder after Alzheimer's disease. PD patients suffer from a lack of the neurotransmitter dopamine, and therefore initially receive oral suppletion of the dopamine-precursor levodopa to treat motor symptoms such as bradykinesia (i.e., slowness of movement), rigidity, or tremor. Although levodopa initially has a good effect on these motor symptoms, patients may develop motor complications such as “OFF“ time during which the medication no longer generates the desired effect, or excessive movements (i.e., dyskinesias) due to a surplus of levodopa (Ahlskog and Muenter, 2001). For these patients, with fluctuations refractory to changes in oral medication, Deep Brain Stimulation (DBS) may provide relief.

DBS is an invasive surgical procedure during which electrodes are inserted into the brain. Contrary to the fluctuations in motor functioning due to intake of oral levodopa, DBS may provide a constant level of functioning. However, an extensive screening procedure precedes the DBS surgery, as several symptoms of the disease may deteriorate postoperatively, mostly through unknown mechanisms (Geraedts et al., 2019). As several patients may suffer from cognitive deterioration after surgery, the screening procedure encompasses amongst others an assessment of neuropsychological functioning.

The tests of neuropsychological functioning are often influenced by factors, such as fatigue, motivation, motor impairments, or language barriers. Therefore, it has been previously suggested to use Electroencephalography (EEG) as additional biomarkers for cognitive impairment. EEG is a noninvasive method to study brain activity directly by means of electrodes attached to the scalp. EEG records the difference in electric potential between pairs of electrodes (Britton et al., 2016). Although typically performed to detect aberrant brain activity for the diagnosis of epilepsy (i.e., qualitative EEG where the clinician identifies abnormal brain activity), the use of quantitative Electroencephalography (qEEG) is being increasingly used for research purposes. During qEEG analyses, metrics relating to EEG-frequency (derived from the EEG power spectrum) or EEG-synchronization (i.e., connectivity) are derived. A limited set of qEEG biomarkers has been previously associated with cognitive dysfunction in PD (Geraedts et al., 2018), distincting cognitively normal from cognitively impaired patients. Most biomarkers relate to the EEG power spectrum (brain activity frequency), however recent advances in neurophysiology have identified connectivity-metrics relating to brain-synchronization to have additional value. The extraction of new biomarkers from the EEG time series is a tedious and time-consuming process and there is no driving hypothesis on which new biomarkers may be important.

5.3.2 Related Work

The study of clinical correlates of qEEG in PD has been a major topic in neurology research. qEEG describes the analysis of EEG signals based on quantitative measures discovered by, e.g., spectral analyses, whereas qualitative EEG relies on descriptions or interpretations of the EEG provided by the clinician, e.g., epileptiform activity. This study is devoted to qEEG in PD. Relevant studies can be separated into cognition, motor function, responsiveness to interventions and others (Geraedts et al., 2018).

Supervised machine learning techniques are used to identify cognitive decline in PD patients. Training on a data set collected over 3 years with 24 hand-crafted features, a Random Forests model is able to achieve an AUC score of 75% (AUC stands for Area under the Receiver Operating Characteristic Curve and represents the trade-off between true positive rate and false positive rate). *Theta power* was identified as the most important feature (Cozac et al., 2016). *Theta power* describes the brain-activity in the *slow activity range* within $4-8$Hz.

In a similar approach, survival analysis and logistic regression were used to predict cognitive worsening of newly diagnosed PD patients, which trained on a data set of 54 PD patients collected over 5 years. In this research, both clinical and neuropsychological features are adopted. The logistic regression model achieved an accuracy of 82% in the leave-one-out cross validation. The survival analysis showed that a high EEG frequency was related to long survival times and low EEG frequency to an early worsening (Arnaldi et al., 2017).

A larger data set with 118 PD patients, containing 5 different severity groups with cognitive impairment, was used to train a Support Vector Machine (SVM) and a k-Nearest-Neighbor (KNN) model. Based on 6 different EEG configurations, the authors selected qEEG features computed from 7 frequency bands, as well as age, gender, duration of formal education and individual peak frequency. A filter method for feature selection was applied. Based on Pearson correlation, only correlated features from different frequency bands and different anatomical regions were selected. These selected features were used with a 5-fold cross validation approach achieving accuracies of about $86\% \pm 3.5$ for SVM and $87\% \pm 2.8$ for KNN. The results represent the average of 5 computation runs. In a second attempt, the data was split into a group of 100 patients as training data and 18 patients as test data resulting in an accuracy of 84% for SVM and 88% for KNN (Betrouni et al., 2019). The evaluation of the model performance based on such a small data set with only one train / test split does not necessarily reflect the real generalization capacity. Furthermore, the number of patients belonging to one of the five classes differs from $5 - 43$. Therefore, we can state an imbalance of the data set (class-imbalance). However, the study reports only the accuracy results, which is no adequate and representative metric for class-imbalanced data sets (Hossin and Sulaiman, 2015).

Based on 70 PD patients including 27 patients with a mild cognitive impairment (MCI) and 43 without a cognitive impairment, the authors of (Chaturvedi et al., 2019) used a Random Forest model to classify MCI and MCI-free patients. The model contained 66 frequency and 330 connectivity features (Phase Lag Index). Based on a 5-fold cross validation the model achieved the maximum AUC score of $73\% \pm 16$ with Phase Lag Index features. The results represent the average of 20 computation runs (Chaturvedi et al., 2019).

The presented work shows relatively straightforward machine learning approaches, but the described algorithms may be difficult to reproduce in independent populations, a problem that is inherent to many machine learning approaches. However, particularly in the medical domain, the transparency of the way how a machine learning approach is designed is key for evaluating its clinical real-world use and to assess its capabilities. Therefore, combining clinical knowledge with mathematics and computer science is key when developing reliable models (Bonanni, 2019).

The approaches presented above offer opportunities for enhancement by using more advanced machine learning techniques. The intense use of feature engineering, particularly in Chaturvedi et al. (2019); Arnaldi et al. (2017); Cozac et al. (2016), in terms of feature selection algorithms would help finding most important features and removing less important ones. This would enhance the model overall performances. Furthermore, the set of chosen features varies, and the features are mainly based on hand-picked representations from earlier publications. This a limitation due to the arbitrary selection of features with focusing only on spectral analysis of connectivity (PLI), rather than trying to find completely new features. An automatic approach to find relevant time series representations would create new features or rather biomarkers which were not found by manual analysis, as well as avoiding time-consuming feature engineering processes. In addition, in none of the presented work model optimization techniques were used. For example, the use of hyperparameter optimization would enhance the model performances, further (Claesen and Moor, 2015).

In this chapter, we address such downsides and use more advanced techniques. These techniques can be still considered as comprehensible and interpretable. We combine advanced techniques with clinical and automatically computed features to find the most promising features and the best performing approach.

Table 5.11: Overview of the approaches for automated EEG assessments with machine learning.

Appr.	Description	Cases	Poor cog.	Good cog.	Dimension	Length
1	Auto. feat. (indi.)	200	100	100	21	5000
2	Auto. feat. (aver.)	40	20	20	21	5000
3	Clin. feat. (aver.)	40	20	20	-	-
4	Clin. feat. + auto. feat. (aver.)	40	20	20	21	5000
5	Clin. feat. (indi.)	200	100	100	-	-
6	Clin. feat. + auto. feat. (indi.)	200	100	100	21	5000
7	Clin. feat. (all) + auto. feat. (aver.)	40	20	20	21	5000
8	Clin. feat. (all) + auto. feat. (indi.)	200	100	100	21	5000

5.3.3 Data Set

The data used in case C contains EEG recordings of 125 patients. Forty patients with clinically confirmed 'good cognition' ($n = 20$) or 'poor cognition' ($n = 20$) were selected for group-classification. From the EEG of each patient, 5 separate epochs (EEG-segments) were extracted.

The data was recorded with a sampling rate of 500 Hz, a 16 bit AD (Analog-to-digital) converter and a band filter of $0.16 - 70.0$ Hz. Given the sampling rate and the duration of 8.192 seconds, each EEG epoch contains 4096 data points.

All patients received routine EEGs (Britton et al., 2016) with 21 electrodes which results in 21 recorded time series. The electrodes were placed according to the international $10 - 20$ system (Jasper, 1958). During recording, patients were supine, eyes-closed, during a state of relaxed wakefulness. All epochs per patient originate from the same recording (total EEG lasts roughly 15-20 minutes), but we selected 5 epochs of 8.192 seconds from the total EEG to ensure all time-series are artefact-free. EEGs are very sensitive to artefacts, including muscle artefacts (i.e., when a patient moves, frowns, swallows, blinks, etc.), drowsiness-artefacts, electrical artefacts from neighboring electrical apparatus, or sweating artefacts. These artefacts are likely to interfere with the quantitative analysis of brain activity. We therefore performed a rigorous selection of artefact-free segments through visual inspection. Mathematically, using indices $p \in [1..40]$ for the patient, $e \in [1..5]$ for the measurement epoch and $i \in [1..21]$ for the channel, the EEG measurement is denoted as follows:

$$\forall(p,e,i), \quad \mathbf{s}_i^{(p,e)} = \left(s_1^{(i)}, s_2^{(i)}, \ldots, s_l^{(i)}\right)^\top \in \mathbb{R}^L,$$

where $L = 5000$ is the number of recordings for each EEG channel. In addition, for each pair of patient and epoch, the corresponding data point is represented as matrix:

$$\forall(p,e), \quad \mathbf{S}^{(p,e)} = \left(\mathbf{s}_1^{(p,e)}, \mathbf{s}_2^{(p,e)}, \ldots, \mathbf{s}_{21}^{(p,e)}\right) \in \mathbb{R}^{L\times 21}.$$

As for the classification target, it is labelled (determined by experts) for each patient and thus is denoted as follows: $\forall p \in [1..40]$,

$$Y^{(p)} \in \{\textsc{good cognition}, \textsc{poor cognition}\}.$$

As EEG-channels reflect upon a difference in electrical potential, i.e., the difference in potential between pairs of channels, all time series were re-referenced towards a single common reference (Cz).

Based on two separate scales the labelling of the data was performed by clinicians after routine neuropsychological assessments. The scales are the disease-specific screening tool *Severity of Non-dopaminergic Symptoms in Parkinson's Disease scale*, items *cognition* (SENS-PD-COG, range $0 - 9$) (van der Heeden et al., 2016), and the generic instrument *Montreal Cognitive Assessment* (MoCA) (range

0 – 30). In the absence of MoCA scores, scores from the Minimal Mental State Examination (MMSE) were used. Criteria for *good cognitive performance* included SENS-PD-COG scores ≤ 2 and MoCA scores ≥ 27 (or MMSE ≥ 29); criteria for *poor cognitive performance* included SENS-PD-COG scores ≥ 5 and MoCA or MMSE scores ≤ 27 (Nasreddine et al., 2005; Folstein et al., 1975).

The 20 patients selected for each group result in 100 available epochs per group if biological intra-individual variability is being disregarded, or 20 available *averaged* EEGs if this variability was accounted for.

In the following section the machine learning pipeline is described.

5.3.4 Machine Learning Pipeline

The pipeline used in this study is proposed in Chapter 4.3 and applied in the automotive industry for time series classification with vehicle on-board data (see Chapter 5.1, 5.2).

The adaption of the pipeline for this study is described below.

5.3.4.1 Feature Extraction from Time Series

When considering all the *EEG* features together, the overall feature mapping is a real-valued function that takes all EEG channels as input (for each pair of patient and epoch) and yields a m-dimensional (m is the number of features. Please see below) real-valued feature vector, $\boldsymbol{\mathcal{F}}\colon \mathbb{R}^{L\times 21} \to \mathbb{R}^m$:

$$\forall (p,e), \quad \mathbf{S}^{(p,e)} \mapsto \boldsymbol{\mathcal{F}}\left(\mathbf{S}^{(p,e)}\right).$$

For each pair (p,e) of patient and measurement epoch, we shall denote the extracted feature vector as $\boldsymbol{\mathcal{F}}^{(p,e)}$, which will undergo the feature selection procedure. For the feature extraction phase tsfresh was used with its default settings. In total, such an extraction procedure gives us $m = 21 \times 794 = 16674$ features (21 electrodes per patient recording result in 21 time series).

In addition to the automatically extracted features, several spectral features derived from Fast Fourier Transformation (FFT) of the raw time series, which are predominantly used for clinical purposes, are considered in this study (Geraedts et al., 2018). Specifically, the following *clinical features* are also considered when training the machine learning model:

- Occipital peak frequency,
- Total Peak frequency,
- FFT delta power (global),
- FFT theta power (global),
- FFT alpha1 power (global),
- FFT alpha2 power (global),
- FFT beta power (global),
- FFT ratio slow-over-fast,
- age,
- gender.

We shall denote those features together as $\boldsymbol{\mathcal{F}}_c^{(p,e)} \in \mathbb{R}^8 \times \mathbb{N}_{>0} \times \{\textsc{male}, \textsc{female}\}$ (c stands for "clinical") for each pair of patient p and epoch e.

From this generated feature space (automatically computed features and clinical features) the most significant ones, which are most suitable for representing the time series, are selected in the next step.

5.3.4.2 Feature Selection

In the feature selection phase, the goal is to choose the set of features that are mostly relevant for the classification task. Note that, due to the massive number of automatic features (from tsfresh), those features always undergo the selection procedure. As for the clinical features, it is undecidable whether those should be included in the feature selection and therefore, two approaches,

- Clinical features are also selected as with automatic features, and
- Clinical features do not undergo feature selection (marked as "all" in Table 5.11),

are both investigated in this chapter. For each pair (p, e) of patient and measurement epoch, we shall use $\boldsymbol{\mathcal{F}}_s^{(p,e)} \in \mathbb{R}^{m'}$ to denote the vector resulted from feature selection (s stands for "selected" and m' is the number of selected features). Note that if clinical features do not undergo the feature selection procedure, those are directly included in $\boldsymbol{\mathcal{F}}_s^{(p,e)}$. As in Chapter 3.4.3 described, the feature selection procedure is carried out with the Boruta algorithm.

5.3.4.3 Modeling

In this phase the selected features are used to train a random forest classifier. Naively, the input to random forest model is those selected feature vectors $\boldsymbol{\mathcal{F}}_s^{(p,e)}$, resulting in $40 \times 5 = 200$ data points as the available data. In this case, each pair (p, e) of patient and measurement epoch is considered as *individual* data point. However, it might also make sense to group the feature vectors by patients as those 5 epochs of EEG are measured on the same patient. In this case, the most straightforward treatment is to take the *average* value of feature vectors over five epochs. In this study, those two modeling approaches are both conducted. We summarize them as follows:

- *individual*: the data set (pairs of input and output) is,

$$\left\{ \left(\boldsymbol{\mathcal{F}}_s^{(p,e)}, Y^{(p)} \right) \right\}_{\substack{p \in [1..40] \\ e \in [1..5]}}$$

- *average*: each input is obtained by averaging feature vectors over five epochs:

$$\boldsymbol{\mathcal{F}}_s^{(p)} = \frac{1}{5} \sum_{e=1}^{5} \boldsymbol{\mathcal{F}}_s^{(p,e)}.$$

  And only 40 data points are available after this data aggregation:

$$\left\{ \left(\boldsymbol{\mathcal{F}}_s^{(p)}, Y^{(p)} \right) \right\}_{p=1}^{40}.$$

Moreover, for both *individual* (indi.) and *average* (aver.) approaches, the input $\boldsymbol{\mathcal{F}}_s^{(p,e)}$ could be selected from 1) automatic (auto.) features (feat.) only 2) clinical features only 3) the union of automatic and clinical features. Table 5.11 shows an overview of the resulting 8 modeling approaches.

5.3.4.4 Hyperparameter Optimization

The hyperparameter of the random forest model are tuned with the Bayesian MIP-EGO algorithm (see Chapter 4.3). The list of hyperparameters under consideration and the range thereof are shown in Table 5.12.

Table 5.12: Hyperparameter search space for optimizing the random forest classifier.

Parameter	**Range**
Max depth of each tree	$\{1, 2, \ldots, 100\}$
Number of trees	$\{1, 2, \ldots, 100\}$
Max number of features when splitting a node	$\{\texttt{auto}, \texttt{sqrt}\}$
Min number of samples required to split a node	$\{2, 3, \ldots, 20\}$
Min number of samples required in the leaf node	$\{1, 2, \ldots, 10\}$
Use bootstrap training samples?	$\{\texttt{True}, \texttt{False}\}$

5.3.5 Performance Evaluation

The data set used in this chapter contains data of 20 patients with a *good cognitive performance* and 20 patients with a *poor cognitive performance.* Since this data set is completely balanced between both classes, the classification accuracy is an appropriate performance measure. In addition, the commonly used performance measures, precision, recall, F_1 score, ROC curve and the Area Under ROC curve (AUC) are also provided in the result (see Chapter 4.3).

The resulting scores are based on a 10-fold cross validation (CV) (see Chapter 4.3). The average of performance scores from all folds represents the final score. In contrast to CV, when trained on all data, the models with optimized hyperparameter settings for EEG assessment achieve a final classification accuracy of 100%. This is a clear indication of model overfitting, i.e., such models would not generalize well for new patients. Note that, in the *individual* modeling approach (where there are five data points for each patient, obtained from five epochs), data points of one patient is **never** separated to the training data and test data. It is important to avoid, that the model is trained on an EEG of a patient and tested on another EEG of the same patient. Each resulting score represents the average of 5 runs of the automatic machine learning pipeline.

Table 5.13: Performance scores of all modeling approaches tested in this chapter. The scores are calculated on the test set and averaged in a 10-fold cross validation. The mean and standard deviation are aggregated from 5 repeated runs of the 10-folder CV. Here m' stands for the number of features selected by Boruta algorithm. A range of m' is shown here as the number of selected features differ from one repeated run to another.

Approach	1	2	3	4
Range of m'	$99-108$	$1-3$	3	$1-3$
Accuracy	66.5%±1.4	83.0%±3.3	67.5%±5.0	84.0%±4.2
F_1	68.2%±1.3	83.8%±3.7	70.5%±2.7	85.6%±3.2
Precision	64.0%±0.9	81.7%±6.1	64.3%±6.2	88.3%±6.9
Recall	73.0%±1.4	86.0%±5.5	78.0%±2.7	83.0%±2.7
AUC	69.2%±1.9	86.0%±6.5	76.5%±3.8	86.8%±6.0

Approach	5	6	7	8
Range of m'	8	$103-108$	$11-14$	$113-118$
Accuracy	66.0%±0.0	67.4%±1.4	77.5%±10.7	67.6%±1.4
F_1	74.3%±0.0	69.8%±1.2	81.6%±11.7	69.8%±1.2
Precision	72.4%±0.0	65.2%±1.4	80.3%±10.8	64.2%±0.9
Recall	76.3%±0.0	75.1%±2.7	83.0%±12.5	76.3%±1.7
AUC	72.1%±0.0	74.1%±1.4	89.5%±11.7	71.3%±1.7

5.3.6 Results

The results of all approaches are shown in Figure 5.13. First of all, the achieved scores indicate that the assessment of PD patients into a *good* or *poor cognition* can potentially be carried out by machine learning techniques. Comparing the *averaged* approaches with the *individual* approaches, it suggests that the common clinical practice to average the features is better suited than using EEG features from a single EEG, which could be attributed to the noise reduction effect from the averaging and more suitable features.

Based on the cross validation, the modeling approach (4) (cf. Table 5.13) shows the best performance with an accuracy of 84.0%. This so-called *averaged* approach contains an initial feature space with automatically extracted features and 10 clinical features. In 5 repeated runs, the Boruta algorithm selects $1-3$ number of features. Moreover, all selected features are from the automatically extracted features, i.e., none of the clinical features were chosen by the algorithm. Hence, approach (4) is similar to approach (2) which is also an *averaged* approach that only uses automatically extracted features. This similarity is also supported by the accuracy scores 83.0% and 84.0% for approach (2) and (4), respectively. The small difference in accuracy, 1.0% is likely caused by random error due to the small number of repeated runs. Since the approaches are identical, we will state (2, 4) as the best modeling approach henceforth.

		Predicted	
		good	poor
Actual	good	16.2	1.8
	poor	4.3	13.7

		Predicted	
		good	poor
Actual	good	1.7	0.3
	poor	0.5	1.5

Table 5.14: For modeling approach (4), the confusion matrices is drawn for the training data (left) and the test data (right). The entries are averaged over all folds of the 10-fold cross validation.

Figure 5.14 illustrates the confusion matrices of modeling approach (4) on the training data (left) and the test data (right). The entries are averaged over all folds of the 10-fold cross validation (see Chapter 3.1), showing that on average only 0.3 false positives and 0.5 false negatives are incurred when testing. Furthermore, figure 5.9 illustrates ROC curves of modeling approach (4). The mean ROC curve as aggregated over all folds is depicted for both training data (blue) and test data (black). The area under the curve (AUC) is an indicator of the quality of ROC curve, i.e., the larger the area under the curve the higher the performance. The AUC score is 91.4% and 81.3% on the training and test data, respectively.

In contrast to modeling approach (4), approach (7) excludes the clinical features from the feature selection procedure, i.e., all 10 clinical features were used with automatically extracted feature. The comparison of approach (4) and (7) shows that the clinical features cause a 6.5% decrease of the accuracy.

Furthermore, the exclusive use of clinical features as in approach (3) and (5) also results in significantly lower accuracies: 67.5% and 66.0%, respectively. This observation is also supported by other types of performance scores. Based on the performance score, the automatically extracted features could facilitate random forest models to achieve a higher performance, compared to the random forest model that only uses clinical features.

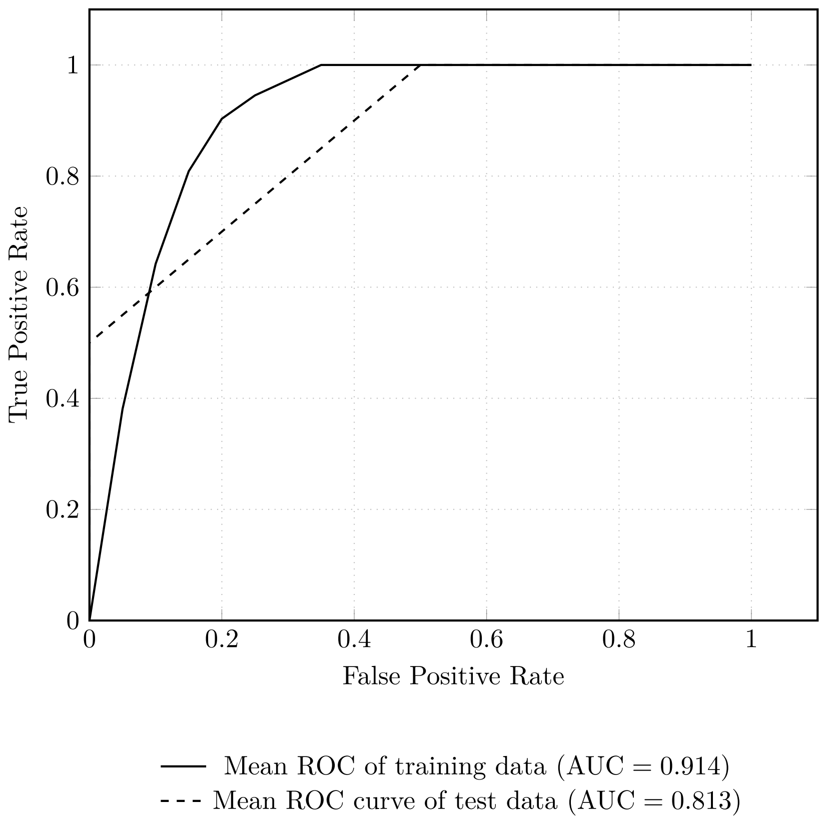

Figure 5.9: The ROC-curve of the train and test data folds. This illustration is based on the modeling approach (4).

The best performing random forest models from approach (2, 4) rely on features that are automatically extracted and selected, which might be unforeseen for the clinical expert. In the following, the most important automatically extracted features of approach (2, 4) are listed:

- ’Fp1-Cz___fft_coefficient___coeff_54___attr_"real"’,
- ’F8-Cz___fft_coefficient___coeff_35___attr_"real"’,
- ’Pz-Cz___fft_coefficient___coeff_33___attr_"real"’,
- ’Pz-Cz___fft_coefficient___coeff_96___attr_"imag"’,
- ’Fp1-Cz___fft_coefficient___coeff_25___attr_"angle"’,

- 'F3-Cz___fft_coefficient___coeff_25___attr_"angle"',
- 'P4-Cz___fft_coefficient___coeff_76___attr_"angle"',
- 'P3-Cz___fft_coefficient___coeff_67___attr_"imag"',
- 'P4-Cz___fft_coefficient___coeff_6___attr_"abs"',
- 'P3-Cz___fft_coefficient___coeff_17___attr_"abs"',
- 'P3-Cz___autocorrelation___lag_1',
- 'Pz-Cz___autocorrelation___lag_5',
- 'Fp1-Cz___cwt_coefficients___widths_(2, 5, 10, 20)___coeff_9___w_2'.

Note that these features are a conglomerate of features from different runs of approach (2, 4). The definition of these features can be found in tsfresh (2018).

The most important clinical features of approach (3) are

- Occipital peak frequency,
- FFT beta power (global),
- Total peak frequency.

The Boruta algorithm, used in the automated approach, yields different features from one repeated run to another due to its internal randomness. Based on this consideration, it is additionally proposed to combine those "best" features from all repeated runs and such a treatment might potentially achieve an even higher accuracy. Therefore, in this *hand-crafted approach*, we combine the selected features from all repeated runs, as well as the best clinical features, which results in 16 features. All of the used features are shown in the two lists above. To test the performance of the combined features, another 10-fold cross validation is performed without any feature selection in the pipeline (obviously not needed here), elevating the accuracy to 91.0%. The results are summarized in Table 5.15.

5.3.7 Summary and Conclusions for Case C

This chapter presents an automated method to classify EEG data. Based on a data set containing 40 PD patients our method extracts and selects most significant features from the time series, trains a random forest model and optimizes its hyperparameters in an automated approach. The results show that, compared

Table 5.15: Test scores of the hand-crafted approach. The scores are calculated on test sets in a 10-fold cross validation. m' stands for the number of manually selected features. The mean and standard deviation are aggregated from 5 repetitions of 10-fold CV.

m'	**16**
Accuracy	91.0% ± 2.2
F_1	92.1% ± 2.1
Precision	90.25% ± 2.8
Recall	94.0% ± 2.2
AUC	98.0% ± 1.1

to only using established clinical features, the automatically computed features significantly increase the model performance. Furthermore, clinicians usually average the features (biomarkers) of 5 EEGs to, e.g., reduce noise. We use this approach, as well as a non-average approach, i.e., a single segment from the EEG per patient is treated as one data point. The results assure that the clinical approach with averaging over 5 EEG per patient is more suitable and achieves higher accuracy. Based on automatically extracted features, the best model achieves an accuracy of 84.0% in the automated approach. In a more hand-crafted approach we used both previously selected features from the automatically extracted features and clinical features resulting in an accuracy of 91.0%.

In a next step the model will be tested on intermediate PD patients, i.e., on patients whose cognitive performance falls between the two extremes used in the present study. If the approach described here can accurately identify patients with a relatively poor cognitive function in this intermediate group, a deep analysis of the newly extracted features may be carried out to investigate the utility of complementary biomarkers for decision making on DBS surgery.

Furthermore, the interaction between the 21 EEG channels can be considered, i.e., the combination of features from different channels. This is a computationally expensive approach but can possibly improve the performance and provide additional knowledge in terms of biomarkers.

In the following section the Boruta algorithm is discussed.

5.4 Discussion of Boruta Algorithm

The feature selection in all three applications described above was mainly performed by the Boruta algorithm. The comparison of Boruta and another promising feature selection method in section 5.2 shows that the model performance of Boruta computed features was higher than with other feature selection methods. However, as this algorithm relies on a multiple testing procedure, it could become too conservative, i.e. Boruta yields too many false negatives. Especially when dealing with a larger feature space like in the examples above, Boruta eliminates possibly important features. This behavior can be observed, for example, in the best working approaches of the EEG study (see section 5.3): the Boruta algorithm tends to select a small number of features ($1-3$) but with different selected features in repeatedly performed runs, i.e. the individual features differ from run to run massively which indicates that important features from one run are not considered as important in another run.

CHAPTER 6

Advanced and Optimized Pipelines

This chapter shows the application of all proposed pipelines from chapter 4. The pipelines are tested on a variety of multivariate data sets which are publicly available. The industrial use of all approaches is investigated with a data set containing on-board car data with the objective to estimate the damaged parts (see Chapter 5.2).

6.1 Data Sets

Table 6.1 shows an overview of the data sets. In this chapter, we use data sets from a multivariate time series archive provided by University of East Anglia (UEA) and University of California, Riverside (UCR) (Bagnall et al., 2018). This archive is called Multivariate Time Series Classification archive (abbreviated as UEA).

The data sets of the UEA archive represent a broad range of problems, such as Human Activity Recognition (HAR), Motion Classification, ECG classification, EEG/MEG classification, Audio Spectra Classification and others. Therefore, the data sets have a wide range of characteristics. Table 6.1 shows the data set sizes, dimensions d, length L and number of classes q. Due to the large number of expensive computations this investigation is limited to data sets with less than 62 dimensions and to comparably smaller data sets. In total, the UEA archive contains 30 data sets from which we use 18 in this investigation. The time series of each set have the same length without any missing values. The training and test cases are already split in those data sets, such that there is a good, unique basis for performance comparison. Furthermore, the imbalance of the training and the

Table 6.1: Overview of the data sets. 18 UEA multivariate time series data sets and one real-world industrial data set (AutomotiveCrash) are used.

Data Set	Tr. Cases	Te. Cases	LRID (Tr.)	LRID (Te.)	Dimensions d	Length L	Classes q
ArticularyWordRecognition	275	300	0.00	0.00	9	144	25
AtrialFibrillation	15	15	0.00	0.00	2	640	3
BasicMotions	40	40	0.00	0.00	6	100	4
Cricket	108	72	0.00	0.00	6	1197	12
Epilepsy	137	138	-0.01	-0.01	3	206	4
EthanolConcentration	261	263	-4.39	-4.35	3	1751	4
ERing	30	30	0.00	-0.13	4	65	6
FingerMovements	316	100	-4.01	0.00	28	50	2
HandMovementDirection	320	147	0.0	-0.12	10	400	4
Heartbeat	204	205	-0.20	-0.20	61	405	2
JapaneseVowels	270	370	0.00	-0.17	12	29	9
Libras	180	180	0.00	0.00	2	45	15
NATOPS	180	180	0.00	0.00	24	51	6
RacketSports	151	152	-0.01	-0.01	6	30	4
SelfRegulationSCP1	268	293	-5.57	-1.16	6	896	2
SelfRegulationSCP2	200	180	0.00	0.00	7	1152	2
StandWalkJump	12	15	0.00	0.00	4	2500	3
UWaveGestureLibrary	120	320	0.00	0.00	3	315	8
AutomotiveCrash	100[1]	-	see Table 6.2	see Table 6.2	4	701	2

[1] This data set is evaluated by cross validation on all data.

test data sets are shown with the Likelihood Ratio Imbalance Degree (LRID) (Zhu et al., 2018). Based on a likelihood-ratio, LRID describes the deviation of the class distribution from a balanced class distribution:

$$LRID = -2 \sum_{i=1}^{q} n_i \ln \left(\frac{N}{q n_i} \right). \tag{6.1}$$

A data set contains q classes. The frequency of the labels can be described by a multinomial distribution $Multinomial(N, p)$, with a total number of observations $N = \sum_{i=1}^{q} n_i$ and the probability of a label $\mathbf{p} = [p_1, p_2, \ldots, p_q]$, which is usually estimated as $\hat{p}_i = \frac{n_i}{N}$. This leads to a so-called class distribution vector $\mathbf{p}$, estimated by $\hat{\mathbf{p}} = [\hat{p}_1, \hat{p}_2, \ldots, \hat{p}_q]$. Furthermore, when referring to $\mathbf{b}$ as an exactly balanced class distribution vector with $\mathbf{b} = [\frac{1}{q}, \frac{1}{q}, \ldots, \frac{1}{q}]$, a measure describing the difference between $\hat{\mathbf{p}}$ and $\mathbf{b}$ indicates the extent of the imbalance. LRID considers the likelihood function in order to test the null hypothesis, whether $\mathbf{p}$ can be fitted by $\mathbf{b}$. The LRID value for a balanced data is zero and for an unbalanced set $LRID < 0$ (Zhu et al., 2018; Ortigosa-Hernández et al., 2017).

Table 6.1 shows that most of the data sets are balanced ($LRID = 0$). Some data sets, however, are imbalanced and the ratio of LRID (Train) and LRID (Test) is not always equal. Such imbalance is important to consider when developing the modeling approach and choosing the right evaluation metric.

Next to the UEA data sets, we use a real-world data set called AutomotiveCrash which contains multivariate time series from low speed car crash tests with the label damaged / undamaged of car parts (see Chapter 5.2). The LRID of the parts are shown which identify an imbalance of the data set.

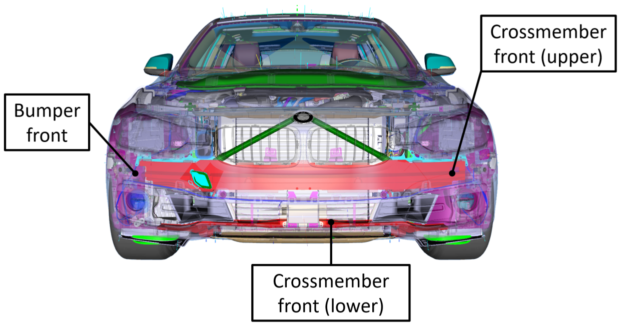

Figure 6.1: Exemplary damaged parts of a low speed frontal crash.

Table 6.2: Overview of the data set AutomotiveCrash. It shows the parts and the LRID. The data of 100 crash tests is used.

Part	LRID
Bumper front	-0.02
Crossmember front (upper)	-0.39
Crossmember front (lower)	-0.69
Impact absorber front	-0.31
Bumper rear	-0.03
Crossmember rear	-0.58
Tailgate	-0.83

6.2 Performance Evaluation Measure

The UEA archive provides a defined split of the data sets into training and test data. Moreover, the archive includes Dynamic Time Warping (DTW) benchmark results based on the key performance score accuracy (Bagnall et al., 2018). Therefore, we apply the proposed data split, i.e. in the training procedure the model only has access to the training data and the final model is trained on all training data and tested on the unseen holdout test set (see Chapter 3.1).

Table 6.3: Overview of the computer specifications. The neural networks (FCN, ResNet, CNN+LSTM) were computed with specification 3. Auto-Sklearn, H2O, ATM and RECIPE were calculated with specification 2. All other methods were computed with specification 1.

Feature	Specification 1	Specification 2	Specification 3
Processor	Xeon Gold @ 2.40 GHz	i7 @ 2.90 GHz	i7 @ 2.20 GHz
Operation System	Windows 7	Ubuntu 19.4	Windows 10
RAM Memory	32 GB	8 GB	24 GB
Graphics	-	-	NVIDIA GeForce GTX 1080
Storage	SSD 300 GB	SSD 256 GB	SSD 500 GB

However, Table 6.1 indicates that the number of classes of some data sets are not nearly balanced (class-imbalance), i.e., accuracy is not a suitable score for all data sets. Therefore, in this experiment the so-called F1 macro score is used as additional performance score (see Chapter 3.1).

Furthermore, each resulting value, based on the UEA data sets, is the average value of three computation runs. Some of the calculations are very time consuming, which has limited the number of runs to three. Our comparison of multiple classification methods on the 18 data sets is based on Friedman's test (Friedman, 1940; Sheskin, 2000). The data for Friedman's test does not have to be normally distributed. Using Friedman's test for such kind of task was proposed in Demšar (2006). It is a non-parametric test, which ranks the algorithms for each data set separately. The lower the ranking the better the algorithm. The rejection of the null hypothesis H_0 indicates a significant difference between at least two of the algorithms. In addition to the comparison of the accuracy and $F1$ performances, we investigate the trade-off between $F1$ and the computation time.

The automotive data set was generated by the author. Furthermore, the data set is small. Splitting the data in the fashion of the other used data sets would influence the results, i.e., the model performance varies depending on the split of training and test data. Therefore, for each part of the vehicle an individual classification model was computed and evaluated using cross validation. The final score of this data set represents the average of all vehicle part model scores.

6.3 Experimental Settings

This section describes the chosen settings of the used methods. In all methods, when possible, the number of CPUs and memory were set to a maximum. In case of methods with a cross validation procedure, we have manually defined its number of splits. This was important, since the default split is usually set to 5 or 10. This caused problems for some of the data sets we used which have less than 10 members in a class.

Besides this, whenever it is not differently mentioned, the methods were run with default settings. In the case of TPOT, we were forced to modify both population size and generations from 100 to 25 due to excessively long computation times. The search space for the hyperparameter optimization of our PHCP is shown in Table 4.1. Table 6.3 shows the specification of the used computers. Due to different hardware requirements of the methods, different computers were used. The neural networks (FCN, ResNet, CNN+LSTM) were computed with specification 3. Auto-Sklearn, H2O, ATM and RECIPE were calculated with specification 2. All other methods were computed with specification 1.

6.4 Results and Analysis

Table 6.4 presents the resulting accuracy scores of the methods. Based on Friedman's test, the ranks in the last row indicate the method's performances over all data sets. At first sight, the rank values are distributed over a wide range. This leads to the assumption, that the methods are significantly different from each other. To verify this, we checked the null hypothesis on the basis of Friedman's test. The chi-square approximation of this distribution is $\chi_r^2 = 51.68$. The tabled value of the depending 99th percentile is $\chi_{.99} = 29.14$ (Sheskin, 2000), i.e., the null hypothesis is rejected at the 0.01 level due to $\chi_r^2 > \chi_{.99}$. This verifies the assumption: according to Friedman's test, a significant difference between the methods exists. Based on the 18 data sets, the AutoML method GAMA shows the best results, followed by the benchmark DTW, Auto-Sklearn, PHCP, CNN+LSTM and TPOT. In this context, it is important to notice that DTW is a distance-based method and considered as a strong method for time series classification. We show that time series classification based on classical features in combination with AutoML or our hand-crafted pipeline can achieve similar or even better performances. Furthermore, the state-of-the-art neural networks (FCNs and ResNets) are in a lower range of performance, whereas our hand-crafted neural network (CNN+LSTM) is showing a significantly higher performance compared to FCNs and ResNets. The HCPGP approach achieves the penultimate rank, outperforming the kPCA+rf.

The comparably poor performance of HCPGP can be caused by many reasons. The population size had to be reduced, potentially requiring additional hyperparameter tuning of the HCPGP parameters, and a larger number of generations would possible also improve the results. For such algorithms it is important to choose wisely the trade-off between performance and runtime. Secondly, and most importantly, the 15 used features are computed by kPCA. Only kPCA has, in combination with a random forest, the lowest performance. Using those 15 feature with our HCPGP approach improves the performance, compared to using the 15 features purely. Based on these findings, we can conclude that the assumption is true that using a combination of features from different time series significantly improves performance.

When taking the imbalance of data sets into account, it is important to consider the $F1$ score. Similar to Table 6.4, Table 6.5 shows the $F1$ scores of all methods with their ranking[1]. The chi-square approximation of this distribution is $\chi^2_r = 54.24$. The tabled value of the depending 99th percentile is $\chi_{.99} = 27.69$ (Sheskin, 2000), i.e. the hypothesis is rejected at the 0.01 level due to $\chi^2_r > \chi_{.99}$. This indicates, that according to Friedman's test a significant difference between the methods exists. Again, GAMA achieves the best performance, followed by PHCP, ATM, TPOT, CNN+LSTM and RECIPE.

Most of the real-world data sets, for example AutomotiveCrash, contain unequally distributed classes and require other measures than accuracy for the evaluation. RECIPE achieves the highest performance for our imbalanced real-world data set AutomotiveCrash.

Next to the accuracy and $F1$ scores, we are investigating the efficiency of the methods. Figure 6.2 shows the performance and computation times of the different methods in one plot. As a performance indicator, we take the ranking of the

[1]The $F1$ scores of the DTW benchmark were not published (Bagnall et al., 2018) and therefore DTW cannot be used in this comparison.

more realistic $F1$ scores into account. The computation time is the average of all runs per method. It is important to consider that the methods Auto-Sklearn, H2O, ATM and RECIPE were computed on a different computer due to different requirements, as well as the neural networks (FCN, ResNet, CNN+LSTM), which were computed on a GPU (Graphics Processing Unit) (see Table 6.3). This limits the comparison regarding computation times of methods which are computed on different computers but we can still gain some insights into the relative performance of the methods. GAMA, the best performing method, requires approximately 60 min to process one data set. Similar performing methods like PHCP and ATM show higher efficiency with computation times below 25 min. PHPC shows a good compromise between performance and computation time. The comparison of the neural networks indicates that the hand-crafted CNN+LSTM achieved higher performance with shorter computation times than the state-of-the-art neural networks, namely FCN and ResNet. Due to many generations of the genetic programming approach, HCPGP needs approximately 14h to process one data set.

As Figure 6.2 clarifies, there is a trade-off between rank and computation time. Four of the tested methods, namely AMLPA, CNN+LSTM, PHCP and GAMA belong, in the sense of the concept of Pareto-optimality (Deb, 2001), to the non-dominated set. All other methods are dominated, i.e., another method that is better in both objectives, rank and computation time, exists.

6.5 Conclusions and Outlook

This chapter presents a comparison of hand-crafted- and state-of-the-art methods for classifying multivariate time series. We have evaluated the methods on 18 publicly available data sets and one real-world data set from the automotive industry. Furthermore, we have investigated the accuracy and $F1$ score as performance measures with the result that, especially for real-world problems, the $F1$ score is a more suitable measure, because $F1$ takes the class-imbalance of data sets into account. The performance comparison shows that feature-based AutoML methods can achieve a higher performance than DTW, which can be considered as a suitable technique for time series processing. Based on the settings used in this chapter, GAMA showed the best performance regarding accuracy and $F1$ score. Other methods like our plain hand-crafted pipeline and our neural network architecture

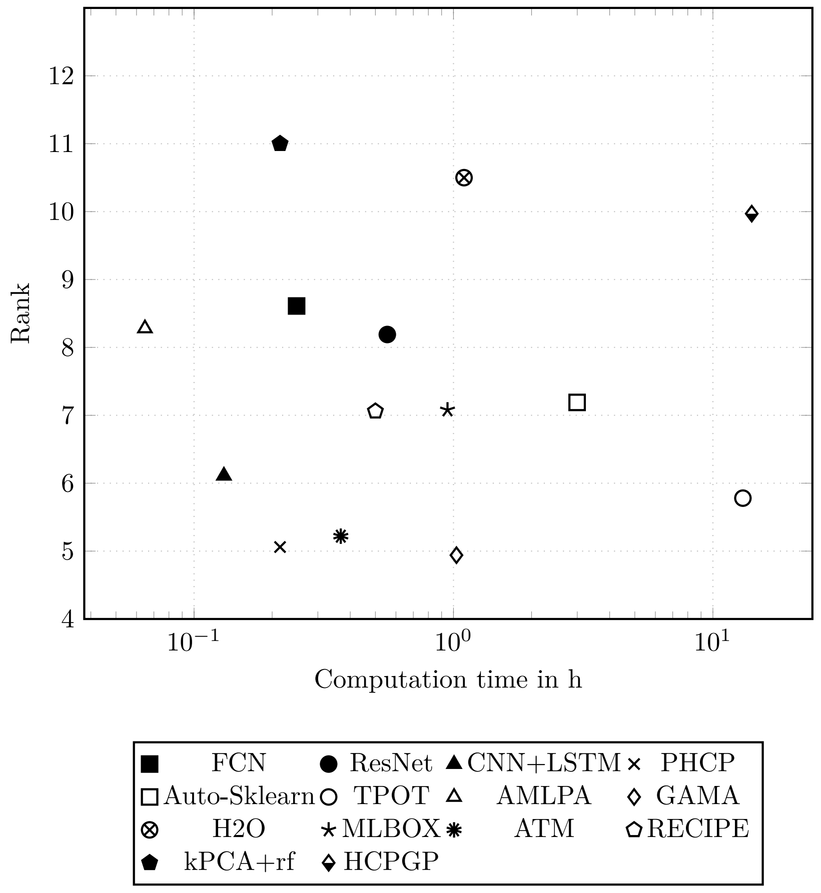

Figure 6.2: The trade-off between performance and computation time. The rank is based on $F1$ score and the computation time are averaged per method over all data sets. The time of feature extraction is not considered.

with CNNs+LSTMS show a similar performance, but these technique are more efficient regarding computation time. State-of-the-art neural networks have a lower performance with longer computation times. Other AutoML methods like ATM or TPOT show also a similar performance. ATM demonstrates a good trade-off between performance and efficiency. The approach with genetic programming achieved a comparably low performance but we could show that the basic idea of automatically combining features from different time series can improve the performance.

As a result, most of the used approaches generate well performing models for time series classification in a more or less efficient way, and they can be used by non-experts.

As a next step, the parameters of the AutoML methods can also be optimized. This would enhance the results once more. In addition, the architecture of especially the CNNs+LSTM could be enhanced further with, e.g., finding a better balance between the number of neurons per layer. In case of the HCPGP, an efficient approach needs to be found to use as many extracted pure features from the time series as possible. The key is not to drop any information from the initial features, because we do not know which combinations of which features can help to classify the problem in a more accurate manner. A promising approach may be using a neural network to reduce the initial feature space.

The following chapter shows a methodology to deal with the implementation of machine learning algorithm in data-driven services in the car industry.

Table 6.4: Average accuracy of the classification methods on 18 UEA multivariate time series data sets and 1 real-world industrial data set (AutomotiveCrash). The boldface highlights the best performing method per data set.

Data set	DTW[1]	FCN	RESNET	CNN+LSTM	PHCP	kPCA+rf	HCPGP	Auto-Skl.	TPOT	AMLPA	GAMA	H2O	MLBOX	ATM	RECIPE
ArticularyWordRecognition	**0.99**	0.92±0.05	0.97±0.01	0.98±0.01	0.98±0.00	0.56±0.01	0.55±0.00	0.97±0.01	0.97±0.04	0.60±0.01	0.92±0.06	0.18±0.02	0.98±0.00	0.97±0.00	0.97±0.00
AtrialFibrilation	0.27	0.27±0.00	0.29±0.04	0.36±0.04	0.25±0.00	0.32±0.01	0.34±0.05	0.38±0.04	0.36±0.06	0.24±0.04	0.29±0.10	0.27±0.00	0.33±0.00	**0.40±0.12**	**0.40±0.00**
BasicMotions	**1.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	0.79±0.21	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	0.96±0.01	**1.00±0.00**	0.94±0.03	**1.00±0.00**	0.98±0.01	0.98±0.00
Cricket	**1.00**	0.97±0.00	0.97±0.00	0.93±0.02	0.97±0.00	0.71±0.10	0.40±0.05	0.96±0.00	0.97±0.02	0.74±0.07	0.99±0.00	0.42±0.06	**1.00±0.00**	0.98±0.01	0.93±0.00
Epilepsy	0.98	**1.00±0.00**	**1.00±0.00**	0.98±0.00	0.97±0.00	0.53±0.13	0.40±0.04	0.98±0.00	0.98±0.01	0.92±0.02	0.99±0.01	0.94±0.01	**1.00±0.00**	0.97±0.01	0.95±0.00
ERing	0.13	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	**0.16±0.02**	0.13±0.00
EthanolConcentration	0.32	0.11±0.16	0.25±0.07	0.29±0.00	0.42±0.00	0.27±0.00	0.28±0.03	0.43±0.03	0.42±0.03	0.40±0.04	0.42±0.01	0.31±0.02	**0.98±0.00**	0.37±0.03	0.96±0.00
FingerMovements	0.55	**0.59±0.04**	0.54±0.01	0.51±0.00	0.58±0.00	0.50±0.01	**0.59±0.01**	0.54±0.00	0.53±0.02	**0.59±0.05**	0.54±0.04	0.55±0.06	0.53±0.00	0.55±0.05	0.57±0.01
HandMovementDirection	0.31	0.18±0.03	0.16±0.01	**0.48±0.12**	0.42±0.01	0.28±0.01	0.29±0.01	0.36±0.03	0.44±0.24	0.30±0.02	0.43±0.03	0.27±0.03	0.24±0.00	0.36±0.07	0.24±0.00
Heartbeat	0.72	0.72±0.00	0.73±0.01	0.69±0.03	0.72±0.00	0.70±0.02	0.70±0.04	0.75±0.05	0.73±0.03	0.74±0.02	0.75±0.01	0.73±0.03	**0.99±0.00**	0.72±0.00	0.95±0.00
JapaneseVowels	0.96	0.91±0.01	0.91±0.01	**0.98±0.00**	0.95±0.00	0.43±0.05	0.29±0.04	0.97±0.03	0.93±0.02	0.79±0.03	0.91±0.05	0.52±0.02	0.69±0.00	0.95±0.01	0.91±0.06
Libras	0.89	0.74±0.00	0.78±0.01	0.78±0.01	0.88±0.00	0.36±0.04	0.26±0.03	0.87±0.01	0.86±0.02	0.63±0.02	0.87±0.07	0.50±0.08	**0.95±0.00**	0.86±0.02	0.94±0.00
NATOPS	0.88	0.85±0.03	0.88±0.00	**0.96±0.01**	0.84±0.00	0.44±0.10	0.32±0.06	0.86±0.01	0.85±0.02	0.71±0.04	0.81±0.07	0.58±0.02	0.83±0.00	0.81±0.06	0.76±0.00
RacketSports	0.87	0.85±0.03	0.91±0.01	0.85±0.02	0.85±0.00	0.69±0.01	0.50±0.00	0.89±0.04	0.83±0.05	0.63±0.20	0.86±0.05	0.82±0.02	**1.00±0.00**	0.84±0.04	0.88±0.00
StandWalkJump	0.33	0.35±0.04	0.29±0.04	**0.47±0.07**	0.44±0.00	0.35±0.07	0.45±0.03	0.20±0.00	0.36±0.04	0.42±0.04	0.46±0.07	0.33±0.20	0.33±0.00	0.31±0.10	0.40±0.00
SelfRegulationSCP1	0.78	0.81±0.08	0.74±0.01	0.83±0.02	0.88±0.00	0.77±0.00	0.77±0.12	0.89±0.02	0.84±0.02	0.88±0.01	0.90±0.04	**0.91±0.02**	0.52±0.00	0.89±0.03	0.51±0.00
SelfRegulationSCP2	**0.54**	0.48±0.04	0.49±0.05	0.49±0.04	0.46±0.00	0.51±0.01	0.53±0.02	0.51±0.01	0.51±0.01	0.52±0.02	0.53±0.03	0.52±0.04	0.46±0.00	**0.54±0.03**	0.46±0.00
UWaveGestureLibrary	**0.90**	0.56±0.01	0.57±0.01	0.85±0.01	0.85±0.00	0.46±0.12	0.45±0.05	0.84±0.03	0.83±0.05	0.66±0.02	0.86±0.02	0.52±0.06	0.78±0.00	0.81±0.04	0.81±0.00
AutomotiveCrash[1]	-	0.84	0.85	0.86	**0.92**	0.82	0.89	**0.92**	0.91	0.77	0.76	0.89	0.90	0.83	0.84
Rank	5.86	8.95	8.67	6.78	6.72	12.22	10.75	5.97	6.83	9.47	5.64	10.67	6.92	6.81	7.75

[2] AutomotiveCrash is not included in the ranking, because it is not a part of the publicly available UEA data sets. These scores are presented as the average of all individual car part prediction models. Due to computationally expensive model computations (one model per car part), each part model was computed only one time. Therefore no standard deviation can be shown for the AutomotiveCrash data set.
[1] DTW results are based on (Bagnall et al., 2018).

Table 6.5: Average $F1$ score of the classification methods on 18 UEA multivariate time series data sets and 1 real-world industrial data set (AutomotiveCrash). The $F1$ score of DTW was not included in the benchmark results and therefore not included in this comparison (Bagnall et al., 2018). The boldface highlights the best performing method per data set.

Data set	FCN	RESNET	CNN+LSTM	PHCP	kPCA+rf	HCPGP	Auto-Skl.	TPOT	AMLPA	GAMA	H2O	MLBOX	ATM	RECIPE
ArticularyWordRecognition	0.93±0.04	**0.98±0.00**	**0.98±0.01**	**0.98±0.00**	0.55±0.01	0.54±0.00	0.93±0.02	0.97±0.04	0.61±0.00	0.92±0.06	0.14±0.01	**0.98±0.00**	0.97±0.00	0.97±0.00
AtrialFibrilation	0.23±0.04	0.15±0.02	0.31±0.06	0.22±0.00	0.25±0.01	0.30±0.08	0.30±0.01	**0.35±0.09**	0.21±0.03	0.26±0.11	0.13±0.04	0.17±0.00	**0.35±0.12**	0.32±0.00
BasicMotions	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	0.77±0.23	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	0.96±0.01	**1.00±0.00**	0.94±0.03	**1.00±0.00**	0.98±0.01	0.97±0.00
Cricket	0.97±0.00	0.97±0.00	0.92±0.02	0.97±0.01	0.71±0.10	0.36±0.08	0.95±0.01	0.97±0.02	0.74±0.07	0.99±0.00	0.40±0.05	**1.00±0.00**	0.98±0.01	0.93±0.00
Epilepsy	**1.00±0.00**	**1.00±0.00**	0.98±0.00	0.97±0.00	0.51±0.13	0.40±0.04	0.97±0.01	0.98±0.02	0.92±0.02	0.99±0.01	0.88±0.11	**1.00±0.00**	0.97±0.01	0.95±0.00
ERing	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.00	0.13±0.01	0.13±0.00	0.13±0.00	0.04±0.00	**0.15±0.02**	0.14±0.00
EthanolConcentration	0.10±0.10	0.24±0.09	0.17±0.00	0.42±0.00	0.26±0.00	0.26±0.01	0.40±0.00	0.43±0.03	0.40±0.04	0.42±0.00	0.25±0.03	**0.98±0.00**	0.37±0.03	0.96±0.00
FingerMovements	0.58±0.04	0.54±0.01	0.34±0.00	0.59±0.00	0.49±0.01	0.55±0.05	0.52±0.02	0.52±0.02	0.60±0.05	0.56±0.02	0.57±0.05	0.53±0.00	**0.63±0.03**	0.54±0.00
HandMovementDirection	0.16±0.01	0.15±0.01	**0.48±0.14**	0.41±0.02	0.26±0.02	0.28±0.01	0.29±0.03	0.43±0.24	0.29±0.04	0.33±0.16	0.16±0.02	0.16±0.00	0.35±0.07	0.15±0.00
Heartbeat	0.42±0.00	0.46±0.04	0.61±0.04	0.81±0.00	0.51±0.07	0.47±0.03	0.69±0.06	0.62±0.01	0.84±0.01	0.84±0.01	0.83±0.03	**0.99±0.00**	0.84±0.00	0.95±0.00
JapaneseVowels	0.91±0.01	0.91±0.02	**0.97±0.00**	0.95±0.00	0.43±0.05	0.29±0.04	0.92±0.01	0.93±0.02	0.79±0.02	0.90±0.05	0.42±0.10	0.66±0.00	0.95±0.01	0.90±0.07
Libras	0.75±0.00	0.78±0.01	0.78±0.01	0.88±0.00	0.36±0.04	0.24±0.02	0.85±0.01	0.86±0.02	0.63±0.02	0.87±0.08	0.45±0.09	**0.95±0.00**	0.85±0.02	0.94±0.00
NATOPS	0.83±0.04	0.89±0.00	**0.96±0.01**	0.84±0.00	0.46±0.00	0.30±0.06	0.84±0.01	0.85±0.02	0.71±0.03	0.81±0.08	0.49±0.01	0.83±0.00	0.80±0.06	0.74±0.00
RacketSports	0.85±0.02	0.92±0.01	0.86±0.02	0.86±0.00	0.70±0.01	0.50±0.00	0.82±0.02	0.84±0.05	0.63±0.23	0.86±0.05	0.63±0.08	**1.00±0.00**	0.85±0.04	0.87±0.00
StandWalkJump	0.28±0.03	0.17±0.03	**0.44±0.06**	0.42±0.00	0.29±0.12	0.41±0.00	0.32±0.00	0.30±0.12	0.36±0.03	0.43±0.10	0.25±0.23	0.17±0.00	0.26±0.12	0.40±0.00
SelfRegulationSCP1	0.80±0.09	0.74±0.02	0.83±0.02	0.89±0.00	0.77±0.00	0.84±0.05	0.84±0.02	0.84±0.02	0.88±0.01	**0.91±0.04**	**0.91±0.02**	0.53±0.00	0.89±0.03	0.51±0.00
SelfRegulationSCP2	0.45±0.06	0.49±0.05	0.49±0.04	0.49±0.00	0.50±0.00	0.53±0.02	0.50±0.01	0.50±0.00	**0.58±0.02**	0.54±0.06	0.52±0.05	0.42±0.00	**0.58±0.07**	0.45±0.00
UWaveGestureLibrary	0.56±0.01	0.59±0.01	**0.86±0.01**	0.85±0.00	0.45±0.13	0.45±0.04	0.78±0.02	0.83±0.05	0.66±0.02	**0.86±0.02**	0.43±0.08	0.76±0.00	0.80±0.04	0.81±0.00
AutomotiveCrash[1]	0.67	0.68	0.72	0.68	0.28	0.55	0.62	0.60	0.00	0.52	0.54	0.54	0.56	**0.85**
Rank	8.61	8.19	6.11	5.06	11.00	9.97	7.19	5.78	8.28	4.94	10.5	7.08	5.22	7.06

[1] AutomotiveCrash is not included in the ranking, because it is not a part of the publicly available UEA data sets. These scores are presented as the average of all individual car part prediction models. Due to computationally expensive model computations (one model per car part), each part model was computed only one time. Therefore no standard deviation can be shown for the AutomotiveCrash data set.

CHAPTER 7

Data-Driven Services in the Car Industry

Numerous recent studies show the prosperous future of data-driven business models. Some key challenges have to be dealt with when moving towards the development of data-driven car services (see Chapter 2). We present a more general approach towards the development of data-driven car services. We point out its main challenges and suggest a method for developing new customer-oriented data-driven services. This approach illustrates key points in developing a practical service, from a technical and business related perspective, which is connected to individualized service examples that would potentially benefit from it. Such data-driven services are developed mostly on a small number of initial test data, which results often in a limited prediction performance. Therefore, based on an optimized Cross Industry Standard Process for Data Mining (CRISP-DM) approach, we propose a methodology for developing initial prediction models with limited test data and stabilizing the models with newly gained data after deployment by online learning. On-board and off-board services are discussed with the result that especially off-board running services offer a large potential for future data-driven business models in a digital ecosystem. The flexibility of such an ecosystem depends on the degree of the integration of the vehicle in the ecosystem - in other words, the car needs to be enabled to deliver data on demand according to General Data Protection Regulation (GDPR) and to any applicable regional law and in cooperation with the customer. The presented method, together with the ecosystem, enables fast developments of various data-driven services.

7.1 Introduction

Robotics and transportation have been underpinned by artificial intelligence since its early beginning. In 1969, Nilsson (1969) discussed the use of artificial intelligence in integrated robot systems. In the late 1970s pioneering discussions were made on the first autonomous vehicles with artificial intelligence (Tsugawa et al., 1979). Across the end of the 1970s to the 1990s, first prototypes were developed by different scientists and organizations (Schmidhuber, 2018). Such technical progresses continued until 2000 and the autonomous driving was feasible for the first time, sparking major developments in both research and industry (Stone et al., 2016; Huber et al., 2008; Aeberhard et al., 2019; Ardelt et al., 2012). In some reports, it is estimated that as of 2020, 10 million cars featuring self-driving will be on the street (Greenough, 2018). In autonomous driving, data from different sensors are combined by computers deployed in the car (Liu et al., 2017). Using methods of artificial intelligence (specifically deep learning techniques), these computers predict the car actions that are required to handle situations. Due to the large data volume, those artificial intelligence models are mainly deployed on on-board-systems (embedded) in the car (Aeberhard et al., 2019). Beyond autonomous driving functionalities, certain types of car data, especially the one related to self-driving, are combined with a car internet interface and a robust internet connection, offering a new era of data-driven services. Most of these services require no additional car hardware and operate only with the vehicle data that is available. As such services are mainly driven by small data volumes, the data set used can be transferred to a back-end system, complying with data protection regulations and customer's consent. This enables running the data-driven service outside of the car (off-board). Hence, new services do not require any changes in the hardware, which significantly simplifies the service development. It allows for the continuous and faster creation of new services, even within the lifetime of cars. Therefore, off-board running services are much more powerful than in-car computations: A car interface sends data on request to a back-end system, which uses data-driven models for a prediction and sends the output, e. g., back to the car. This is combined with full transparency and involvement of the customer regarding certain data. The interconnection of vehicle and back-end system builds a so-called *digital ecosystem*, which integrates all aforementioned methods to provide car services. It enables faster service development and deployment, even within the lifetime of cars.

The revenue from mobility services and connected car services are projected to reach USD 1,087 billion by 2030 (Seiberth and Gruendinger, 2018). This is not only a large business for OEMs (Original Equipment Manufacturer), but also for suppliers as well as ecosystem developers and other parties involved (Seiberth and Gruendinger, 2018). Data availability, its protection and privacy of an open (e. g. to third parties) digital ecosystem, is of key importance to integrate cars more seamlessly into our lives with more digital services. This study mainly illustrates the technical approach with its challenges for developing new data-driven customer services, going from the idea to a running service.

The remainder of this chapter is organized as follows: First, existing work that is related to our approach is discussed in section 7.2. Second, we present an example of a data-driven service in section 7.3. Our proposed methodology with its six main steps is then introduced in section 7.4. In section 7.5, we provide conclusions and an outlook.

7.2 Related Work

Recent studies illustrate new data-driven business models in the car industry by means of a digital vehicle ecosystem. In this context, Seiberth et al. present a definition of data-driven business models: "data [...] as primary business resource to deliver value to customers and to convert this value into revenue and/or profit" (Seiberth and Gruendinger, 2018, p. 8). They declare that in 2050 car manufacturers will achieve 50 % of the revenue from data-driven services. The growing digitalization with its disruption process destroys many traditional business models (Weill and Woerner, 2014). The authors also picture more general business models and the possibilities of digital ecosystems for different industries. Car manufacturers have different approaches to deal with digital services and many tech start-ups are already developing sustainable business models with digital services. Furthermore, OEMs enter already strategic partnerships and invest into such connected vehicle start-ups (Kaiser et al., 2017).

Seiberth and Gruendinger (2018) discuss the available car data, e. g. from sensors for autonomous driving, and highlight the possible revenues when creating services based on it. In addition, there is a growing need for building trust towards the customers regarding the use of their data for services and therefore for the transparency of the data, its use, and privacy and security (Kilian et al., 2017). Beyond that, they present a figure of *the connectivity ecosystem*, which describes roughly a connectivity platform: it communicates with the data source (car) and receives external data like weather, traffic etc. The connectivity platform is connected to the OEM's back-end system, as well as to third party services and apps.

Many studies present new business models enabled by data (Seiberth and Gruendinger, 2018; Weill and Woerner, 2014; Kilian et al., 2017). In most cases, new service ideas are superficially mentioned and it is only briefly discussed how to really benefit from each individual service. Some of the studies discuss the design of (vehicle) ecosystems, e. g. Immonen et al. (2016, 2018), but a methodology for creating data-driven (customer) vehicle services has not been a major topic of scientific research yet.

7.3 A Data-driven Service for Crash Damage Prediction

The variety of possible data-driven services is large. A data-driven car service often assists the customer (like a car pooling service) or the car (like predictive maintenance services). Based on historical data and with methods of artificial intelligence, models are trained to predict behavior, e. g. in car pooling to predict the best possible route to carry the most passengers or if a certain part of the car needs to be maintained in the nearby future.

Another example of a data-driven service is a crash damage prediction system (see Chapter 7). Based on a machine learning model, such a system predicts the damaged parts of a vehicle in a low speed crash. A low speed crash is an accident with a velocity difference below approximately 16 km/h (RCAR, 2018). The baseline of this service is to use only on-board data. Therefore, data from serial car sensors are used for the prediction (e. g. acceleration). To generate an initial data set, low speed crash tests are performed and certain on-board data are recorded. These recordings are used together with the occurred damage on the vehicle for training first initial models. The benefit of such a data-driven service is

e. g. immediate transparency of the damage, which allows initiating a faster and more convenient repair for the customer (Seiberth and Gruendinger, 2018; Koch et al., 2018; Koch and Bäck, 2018).

However, the machine learning model itself is only one unit of the car service. When striving to create seamless customer experiences with a data-driven service, it is essential to consider the whole customer journey. Such a journey describes the way how a customer experiences the whole service. The overall objective should be creating something which is so-called convenient to the customer at all levels. This can be achieved by designing the end-to-end service with its technical challenges like data transfer or intuitiveness of its handling as a whole picture. Based on this, in the following we propose a general path towards data-driven services to tackle and consider the challenges with the one and only goal to create customer value.

7.4 Methodology towards Data-Driven Services

In this section we propose a methodology for developing data-driven car services. This interdisciplinary method is illustrated in Figure 7.1. The horizontal axis shows the time of the development while the vertical one describes the level of development, i. e. the maturity of the service. The origin presents the time of the initial idea about the service and the beginning of its development. The methodological approach consists of six overlapping phases:

1. Idea.
2. Potentials.
3. Modeling.
4. Deployment.
5. Process.
6. Finalization.

All phases are linked to each other. In order to allow short development times the phases are partially executed in parallel. The phases are described in the following sections.

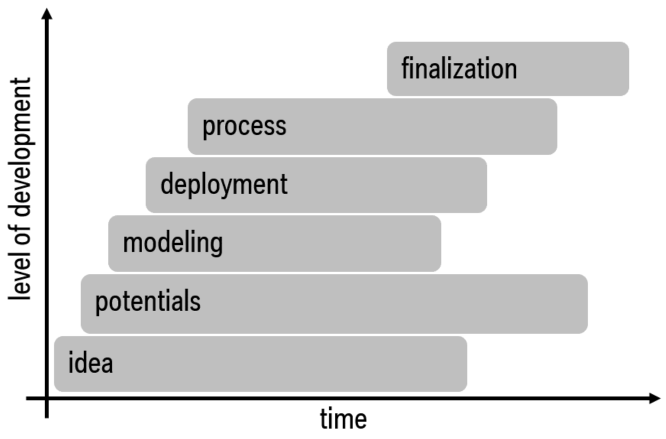

Figure 7.1: The methodological approach: From the idea to a deployed data-driven service.

7.4.1 Idea

The first phase of Figure 7.1 is referred as idea. This pictures the timeline from the first idea about the service to very concrete solution concepts. Principally, there are many motivations or ideas for thinkable services, but for successful and seamless services the business potential and customer benefits have to be evaluated continuously in the next phase, the evaluation of the potential.

7.4.2 Potentials

Seiberth et al. state that new car services follow mainly two objectives: improvement of the brand image or increase of profit (Seiberth and Gruendinger, 2018). This shows that the motivation to create those is based on image or profit reasons or a combination of both. Therefore, data-driven services can have strong impacts on the brand and can be used for strengthening images with creating so-called customer experience by building positive experiences followed by an emotional bond between user and product (Glattes, 2016).

Next to retail customers other stakeholders like, e. g., fleet operators, insurance companies or other parties can strongly support such services with their own advantages (Seiberth and Gruendinger, 2018). Creating a service with many benefiting parties exploits its potentials and is key for a successful and seamless service. Therefore, in case of promising ideas, in phase 2 of Figure 7.1, it is important to analyze and continuously evaluate all aspects of the data-driven service regarding the own objectives and the targets of partners. However, it is mostly very ambitious to evaluate the real potential of a new service in an ad-hoc manner. Therefore, it is important to quickly develop prototypes for experiments, get customer feedback and constantly monitor the need for the service and decide continuously to proceed or cancel the development.

In this context, after revealing an initial potential of the idea, data scientists begin the phase of modeling with collecting data and designing first models.

7.4.3 Modeling

In the beginning of the modeling phase, data scientists have to prove the feasibility of describing the desired relations by the available data with methods from the field of artificial intelligence. A feasibility study helps to quickly assess the practicability of the idea.

To start the modeling phase, an initial data set is crucial. In some cases, the data has been already collected and is available or can be gathered quickly. However, in most cases the data has to be generated manually. When considering the damage prediction system, data from low speed crash tests are required. Performing large numbers of such tests is very tedious and expensive. Therefore, in such cases only small initial data sets are generated in order to evaluate the feasibility. Prediction models based on small data sets are often of poor prediction quality. In order to increase its quality and especially to have informative results for the feasibility study, the use of optimization techniques is key.

Shearer proposed an approach to run data mining projects in industry, the CRISP-DM (Shearer, 2000). This approach has become a very known standard process to perform industrial data science projects. Roughly, it describes the process from the business understanding to data understanding, data preparation, modeling, evaluation until its deployment. We have modified parts of the CRISP-DM and added optimization between modeling and evaluation in order to enhance the model performance. Furthermore, we have separated the data into initial data and field data, as well as the process streams into offline learning (black) and online learning (yellow) (see Figure 7.2). Offline learning describes the process of learning models with an offline generated (initial-) data set (Service Data) to create an (initial-) prediction model. In addition, car information data like the car type, the equipment of the vehicle and geometry information, as well as external data like, e. g. weather or traffic are used as additional data resource, because such data often contain valuable information for the service. After deploying the initial model in an offline learning process, we are updating it by online learning (yellow stream). This means that the initial model is stabilized step by step after deployment with newly generated field data.

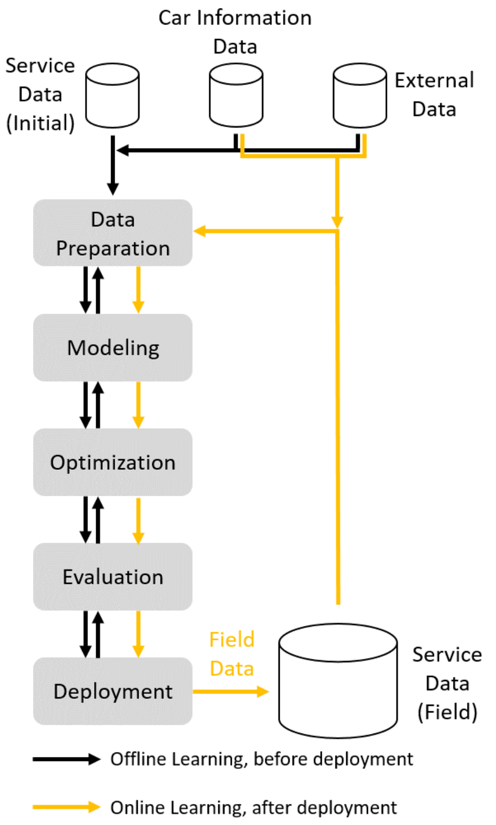

Figure 7.2: The modified CRISP-DM approach with optimization and online learning components. Note that the offline learning part follows the methodology proposed in (Shearer, 2000).

We have developed this modified CRISP-DM approach, when we were dealing with the modeling of the crash damage prediction system. Its required crash data is extremely difficult to generate at large volume, because either crash tests or simulations have to be performed. Therefore, we only created a small test data set containing just enough observations to verify whether it is feasible to use on-board data to predict the damaged parts. We obtained a data set with 100 observations. The number of damages of some parts is less than 5 among the 100 tests. This indicates a very small and class-imbalanced data set. Due to the character of our data set, first results with, e. g., multi-label classification methods were not leading to promising results. More and more we have tailored our approach: we developed a part-wise classification, i. e., we generated individual prediction models for each part of the vehicle. This was very promising, because each model has its own set of characterizing features and its own set of hyperparameters (see Chapter 5.2). However, creating hand-crafted predicting models for each vehicle part is a very time consuming process. As a result, we developed our own automatic approach for time series classification, a so-called machine learning pipeline. The input of our pipeline are time series with the corresponding label. The outputs are predictive model performance measures such as accuracy or $F1$-score, which describe the quality of the prediction.

In Chapter 4.3 our pipeline approach is shown in detail. This pipeline describes the modeling and optimization part of our modified CRISP-DM approach more in detail. Our pipeline is computational relatively cheap and shows promising result (see Chapter 5 and 6). We developed this pipeline to efficiently generate individual models predicting the damage for each part and, more importantly, the pipeline can be used for automatically enhancing and stabilizing the initial model performance after deployment by online learning following the methodology of our modified CRISP-DM approach.

Our initial pipeline models have achieved $F1$-scores between 0% and 94%. This indicates, that based on the small number of data points the predictability depends strongly on the part, i. e., the damage of some parts can be predicted more precisely than of others. Additional methods like frequent pattern mining could help analyzing which parts are likely to be damaged together within one crash. This is especially important for parts with a low prediction quality. Furthermore, by considering, e. g., the learning curves from our results we were able to foresee an improvement of the performance with increasing data (see Chapter 5). Among other things, this helped us to evaluate the feasibility for practical use.

The solution space for such modeling problems is usually large and often the combination of different methods is leading to usable results in practice. In our opinion automatic machine learning methods (AutoML) like our pipeline approach are very promising, because of its practical use and due to its performance and its efficiency (short computation times).

Modern vehicles employ many different sensors and can produce large amounts of data. Sometimes it is not obvious what data would be promising for modeling. Therefore, from a possibly large number of sensors, the most promising ones for the task at hand have to be discovered.

As mentioned, after the data generation a feasibility study shows the practicability of the data-driven service. When receiving results matching the expectations, the models for the serial application can be developed.

In conclusion, the key of the modeling phase is to generate efficient predictive models and to check the validity of the service model approach by assessing the quality of the models that can be learned from the data. After the feasibility is identified and confirmed, the deployment of the service should be prepared.

7.4.4 Ecosystem/Deployment

The deployment of a data-driven service in an automotive environment depends mainly on whether it requires on-board or off-board running services. On-board services are deployed on embedded systems in the car. This needs data storage and computing power on control units of the vehicle. Off-board services are running in back-end systems. In this case, data is transferred via the internet interface from the car to the back-end, provided a sufficient bandwidth is available. In both off-board and on-board running services the car needs to provide the required data. In this regard, in the deployment phase the electronic components of the car have to be enabled to deliver the needed data. Figure 7.3 shows a typical vehicle network of a passenger car. Such architecture consists of different communication systems like Ethernet and more traditional *bus systems* like *FlexRay, CAN* or *LIN*. Ethernet is a local area network (LAN). It is designed to transmit data between computers (Spurgeon, 2000). BES are bridged end stations (switches), which can send and receive transmissions. Bridges communicate to other bridges, to the gateway (router) and to end stations (ES), which is in an automotive environment, e. g., the head unit (Spurgeon, 2000; Spurgeon and Zimmerman, 2013). Bus systems like *FlexRay, CAN* or *LIN* differ in various bandwidths and each system transfers data between components, called Electronic Control Units (ECU). ECUs are embedded systems, which control electrical systems in the vehicle. A car contains many ECUs like the engine control unit, the airbag control unit, the battery management system or the telematic control unit, which sends and receives data via the mobile network. All bus systems are connected via gateways (Robert Bosch GmbH, 2014; Matheus and Königseder, 2015). Sensors are connected to the ECUs, which process the sensor raw data and route it partly to the bus system. This bus data can be used from other ECUs within the connected bus or by the gateway. In some cases, data from one bus is required on another bus. Then, the gateway routes this data from one bus to another. However, mostly data is only available in the ECUs or on the initial bus system.

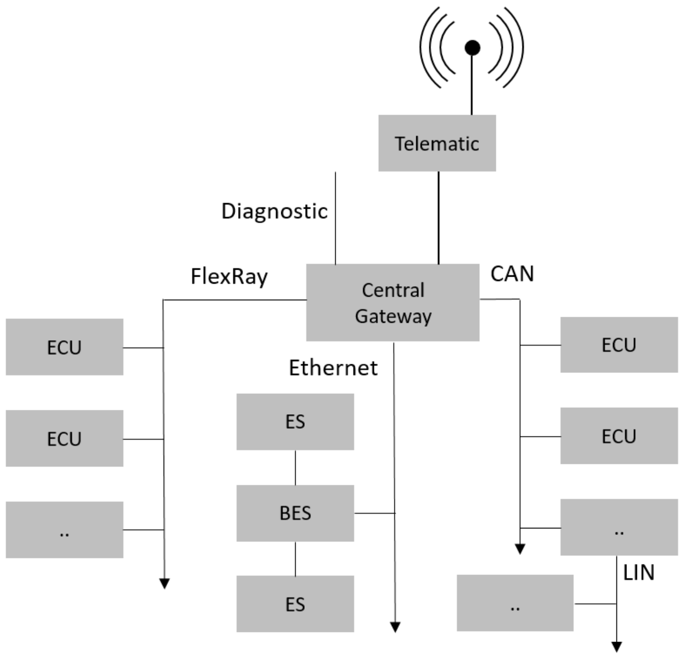

Figure 7.3: A typical schematic in-vehicle network.

Deploying a service on-board in an ECU demands a high effort regarding receiving / delivering the needed data, matching the data quality requirements and the general deployment of the software within the automotive system. This additional software needs to harmonize with all car systems and thus implies very costly technical security. In addition, the capacity of storage within the ECU and its computing power is limited for additional services due to the fact that ECUs serve most likely other essential vehicle functions. Furthermore, it is very challenging to deploy on-board services in the lifetime of cars due to compatibility issues and the additional technical security required.

A more flexible way is established by transmitting the data from the car to a back-end system and running the data-driven system off-board. As soon as the required data is available on a bus system, this data is routed by the gateway to the bus where the sending unit (telematic) is located and it transfers the data from the car to a back-end system. The back-end computes the results and transmits it to the involved systems of the stakeholders. In this regard, the internet interface (telematic) of the car needs to be enabled to transmit different data package sizes in order to provide efficiently the required data. One crucial baseline is, that the bandwidth of the mobile network allows such data transfers.

As mentioned before, the deployment of a service on-board (embedded) is complex, time consuming and not as flexible as a data-driven service is meant to be. Some services need to run on embedded systems like autonomous driving (Liu et al., 2017). The functionality of most of the other services allows running outside of the car like in case of the damage prediction system. Off-board car services provide, next to their simpler deployment, more flexibility regarding faster model updates and adaptations. The key for off-board services is the availability of data: The car has to be able to send the required data on demand, according to GDPR and any applicable regional law and after the confirmation of the customer, i. e., the electronic architecture of the vehicle must be enabled to provide the requested data. This is the foundation of a digital ecosystem, which can collect demanded data and allows deploying new services and interactions with the car and other involved parties quickly.

Figure 7.4 shows the basic principle of a vehicle ecosystem for data-driven services. When developing a new service, the requested data is sent via mobile network from the car to a data layer, called service data, of a protected service cloud (Security Layer). This service cloud is a part of the whole vehicle back-end system. Next to the service data, the back-end system receives external data from third parties like traffic, weather and other service important information. Furthermore, the back-end contains car information like, e. g., the car type, the equipment of the vehicle, the drive technology, geometry information, service information. Such information are often very valuable for a data-driven service. Hence, using these additional data can increase the prediction performances. The data-driven model (AI-Unit) is deployed to the back-end system, as well as other units (Flexible Units) like data processing. This back-end system communicates on demand with the car. Beyond that, the architecture of the ecosystem allows training the models with newly collected data from time to time or automatic (see Figure 7.2). Such a digital ecosystem gives even the possibility to provide partly accesses to third parties to create new valuable services, e. g. BMW Group (2018). Generated customer information from the service cloud are provided to customer devices in the car (control panel) or outside the car, e. g. mobile applications.

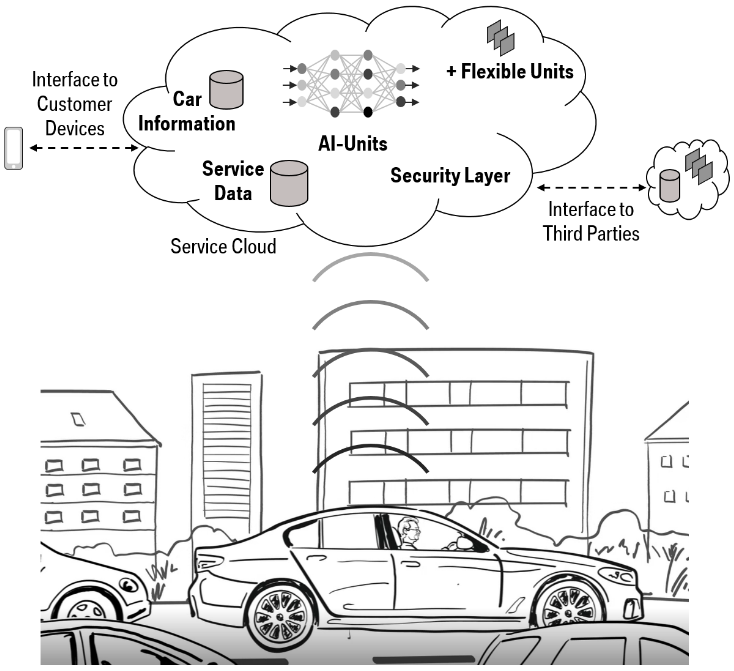

Figure 7.4: A digital ecosystem to run data-driven service. Note that this illustration follows partly the methodology of Kilian et al. (2017).

When considering an example like the damage prediction system, this would in concrete terms imply the following sequence of events: after a low speed crash event and a confirmation of the customer, a small data package is transmitted to the data-driven prediction model in the back-end. With the data package as input the model predicts, based on historical data, the damaged parts and the repair costs. This information can be provided to participants like, e. g., the customer, the insurance companies for seamless claim settlements or the workshops for faster repair (see Chapter 5.2).

In all cases, before any data is transmitted, the customer has to confirm the certain service with an overview of the transmitted data. Furthermore, a general transparency of the service and its intuitiveness in understanding and handling must be provided within seconds to the customer. A confirmation can be canceled anytime. In this context, recent studies show that 94 % of connected car owners are interested in apps and services. Out of those 94 %, 84 % are willing to share personal automotive data for new services (Otonomo, 2018).

An ecosystem with seamlessly operating data-driven services requires data exchange from the ecosystem not only to the car but also to other stakeholders/participants. These processes need to be designed in regards to the business processes. This is described below.

7.4.5 Process

Running a service requires an interconnection of all stakeholders/participants. Without data transfer to all involved parties the potential of the service cannot be exploited. Therefore, shortly after having a rough idea about the deployment, the business process needs to be designed with taking all necessary stakeholders into account.

When looking at the example of the damage prediction system, the data of the damaged parts and the cost for repair are computed in the back-end system. It can be beneficial for the customer to send certain information to other participants like the workshop to order the parts immediately and to prepare the workshop visit. Beyond that, with detailed damage information the insurance company could approve the repair immediately, which simplifies the whole insurance claim settlement and would avoid an interaction of customer and insurance company. Such connections are identified and designed in the process phase. Furthermore, business architects establish customer oriented processes for running the data-driven system with all necessary parties connected. Often, the whole potential can be only reached when all parties are connected in a beneficial way.

7.4.6 Finalization

Figure 7.1 indicates that the finalization phase starts approximately half-way of the process phase. More precisely, when having first working systems, the finalization phase starts with testing, validating and improving the service. In most cases, it is indispensable to test the service with, e. g. defined customer groups to use this feedback for further improvements. In this phase it is key to consider and connect the five previous phases seamlessly with each other in order to create a customer experience.

7.5 Conclusions and Outlook

Nowadays, the expectations regarding data-driven business models in the car industry are massive. This chapter illustrates a track towards an efficient development and deployment approach for data-driven services in vehicles. It presents the important steps, as well as the main challenges. Through the explanation of the methodology, examples are drawn to show precisely the key points. A flexible and various service generation can be reached with a full integration of vehicles in a digital ecosystem, which means that the car delivers data according to GDPR and any applicable regional law and in cooperation with the customer to a back-end system. The main service runs on this back-end system, processes the data and transfers the results to the participants like the customer or other involved parties like, e. g., fleet operators. The shown method enables generating fast data-driven services in order to integrate cars more seamlessly into our lives.

As an outlook, we mention that data enables much more than creating data-driven services: data is transforming car manufacturers from traditional engineering companies to data-driven companies. This indicates that not only service creation but also car development in general is progressively based on data.

CHAPTER 8

Conclusions and Outlooks

8.1 Conclusions

This thesis deals with several important aspects regarding automated machine learning techniques for time series in depth, as well its end-to-end implementation into data-driven industrial application.

To answer the research question (RQ1), *Can we develop efficient time series classification approaches by using feature-based techniques? (in terms of the trade-off between classification quality and computation time)*, in this work hand-crafted machine learning techniques for time series data are developed with the result that feature-based methods can solve such problems in a competitive way. The use of feature-based techniques is sometimes required by the industry because of the need of tamper-proof techniques, i.e. a machine learning model must be explainable. In this context feature-based techniques are very suitable due to the descriptive nature of the features.

Next to hand-crafted approaches, in this work the practical use of state-of-the-art AutoML methods are investigated. This leads to the next research question (RQ2), *Can AutoML methods be applied to solve time series classification tasks?* with the result that, based on an extensive comparison, especially some of those methods are suitable for solving time series classification tasks in an efficient way.

When using generalized and feature-based approaches for time series classification, a massive number of time series features is computed. From this feature space the most significant features are selected in the following step. In order to generate high performing models, it is key to find the most relevant features in the massive feature space. Compared to common machine learning tasks, the feature space for such time series problems is much larger due to the massive extraction which complicates the search problem for significant features. This leads to the next research question (RQ3): *Are state-of-the-art feature selection methods suitable for time series classification tasks based on a massive number of extracted features?*. As result of automatically computed features, especially one method (Boruta) can cope with this problem, even when we believe that some adaptions and optimizations of this methods to this purpose would enhance the performance.

When approaching multivariate time series classification with feature-based methods, it is often beneficial to consider also combinations of features from different time series. This is a challenging problem, because due to the large feature space many different combinations are possible. This issue leads to the next research question (RQ4): *Can the classification quality be increased when combining features with genetic programming?*. To answer this question a tree-based genetic programming approach was developed to combine several features to one complex feature which results in an enhancement of the model performance measures.

Beyond the research questions related to machine learning topics, this work deals with the application of the associated algorithms to the automotive environment. This leads to the next research question (RQ5): *Can we develop a systematic approach for efficiently deploying data-driven services in the industry?* From experience we present the challenges and key points when developing data-driven services and based on this, we developed a methodology to efficiently build such services. This methodology is proposed in this work.

8.2 Outlook

Some of the developed and shown methods show solid results on several academic and real-world data sets. When following the objectives of this work to develop high performing methods with reasonable computation times, parts of each pipeline can be further developed. The neural network CNN+LSTM architecture offers an efficient approach when accepting to sacrifice explainable models. The neural network avoids the whole feature engineering process. The architecture can be further developed, maybe with an even deeper structure when dealing with larger data sets.

The combination of features from different time series is investigated in this work with a genetic approach. Using the massive extracted feature space from the time series is not really feasible due to overwhelming number of feature combinations. Therefore, we reduce the initial feature space by PCA. Even when PCA is an efficient way, information can get lost. Hence, other ways to reduce the dimensions of the initial features without dropping critical information need to be investigated. A promising way could be a neural network.

APPENDIX A

Decision Tree of Case A

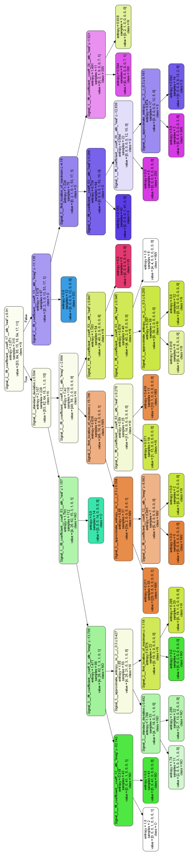

Figure A.1: A decision tree for estimating the vehicle point of impact of a low speed crash event (see Chapter 5.1).

Bibliography

Aeberhard, M., T. Kühbeck, B. Seidl, M. Friedl, J. Thomas, and O. Scheickl (2019). Automated Driving with ROS at BMW. https://roscon.ros.org/2015/presentations/ROSCon-Automated-Driving.pdf. Retrieved May 6, 2019.

Aggarwal, C. (2015a). *Data classification: Algorithms and applications.* Chapman & Hall / CRC data mining and knowledge discovery. Boca Raton, FL, USA: CRC Press.

Aggarwal, C. (2015b). *Data Mining.* Cham, Switzerland: Springer.

Aggarwal, C. C. (2018). An introduction to neural networks. In *Neural Networks and Deep Learning*, Volume 2, pp. 1–52. Cham: Springer.

Ahlskog, J. E. and M. D. Muenter (2001). Frequency of levodopa-related dyskinesias and motor fluctuations as estimated from the cumulative literature. *Journal of the Movement Disorder Society 16*(3), 448–458.

Ardelt, M., C. Coester, and N. Kaempchen (2012). Highly automated driving on freeways in real traffic using a probabilistic framework. *IEEE Transactions on Intelligent Transportation Systems 13*(4), 1576–1585.

Arnaldi, D., F. de Carli, F. Famà, A. Brugnolo, N. Girtler, A. Picco, M. Pardini, J. Accardo, L. Proietti, F. Massa, M. Bauckneht, S. Morbelli, G. Sambuceti, and F. Nobili (2017). Prediction of cognitive worsening in de novo parkinson's disease: Clinical use of biomarkers. *Journal of the Movement Disorder Society 32*(12), 1738–1747.

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms.* Oxford University Press.

Bagnall, A. J., H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. J. Keogh (2018). The UEA multivariate time series classification archive, 2018. *CoRR abs/1811.00075*.

Bankó, Z. and J. Abonyi (2012). Correlation based dynamic time warping of multivariate time series. *Expert Systems with Applications 39*(17), 12814–12823.

Bashir, F., W. Qu, A. Khokhar, and D. Schonfeld (2005). HMM-based motion recognition system using segmented PCA. In *IEEE International Conference on Image Processing 2005*, Genoa, Italy, pp. III–1288. IEEE.

Baydogan, M. G. and G. Runger (2016, March). Time series representation and similarity based on local autopatterns. *Data Mining and Knowledge Discovery 30*(2), 476–509.

Ben D. Fulcher, Max A. Little, and Nick S. Jones (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of The Royal Society Interface 10*(83).

Bergstra, J. and Y. Bengio (2012, February). Random search for hyper-parameter optimization. *The Journal of Machine Learning Research 13*(1), 281–305.

Bergstra, J., D. Yamins, and D. D. Cox (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning*, pp. I–115–I–123. JMLR.

Betrouni, N., A. Delval, L. Chaton, L. Defebvre, A. Duits, A. Moonen, A. F. G. Leentjens, and K. Dujardin (2019). Electroencephalography-based machine learning for cognitive profiling in parkinson's disease: Preliminary results. *Journal of the Movement Disorder Society 34*(2), 210–217.

Bishop, C. M. (2009). *Pattern recognition and machine learning* (Corrected at 8. printing 2009 ed.). Information science and statistics. New York, NY: Springer.

BMW Group (2018). BMW group launches BMW cardata: new and innovative services for customers, safely and transparently. https://www.press.bmwgroup.com/global/article/detail/T0271366EN/bmw-group-launches-bmw-cardata:-new-and-innovative-services-for-customers-safely-and-transparently?language=en. Retrieved November 19, 2018.

Bonanni, L. (2019). The democratic aspect of machine learning: Limitations and opportunities for parkinson's disease. *Journal of the Movement Disorder Society 34*(2), 164–166.

Breiman, L. (2001, Oct). Random forests. *Machine Learning 45*(1), 5–32.

Britton, J. W., L. C. Frey, J. L. Hopp, P. Korb, M. Z. Koubeissi, W. E. Lievens, E. M. Pestana-Knight, and E. K. St. Louis (2016). *Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants.* American Epilepsy Society, Chicago.

Brochu, E., V. M. Cora, and N. de Freitas (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR abs/1012.2599*.

Brockwell, P. J. and R. A. Davis (1991). *Time Series: Theory and Methods.* New York, NY, USA: Springer.

Cano, A., S. Ventura, and K. J. Cios (2017). Multi-objective genetic programming for feature extraction and data visualization. *Soft Computing 21*(8), 2069–2089.

Carroll, M. (2018). How artificial intelligence enables smarter claims processing. https://www.capgemini.com/2017/10/how-artificial-intelligence-enables-smarter-claims-processing/. Retrieved June 28, 2018.

Castelli, M., L. Manzoni, and L. Vanneschi (2011). Multi objective genetic programming for feature construction in classification problems. In *Learning and Intelligent Optimization*, Berlin, Germany, pp. 503–506. Springer.

Chandrashekar, G. and F. Sahin (2014). A survey on feature selection methods. *Computers & Electrical Engineering 40*(1), 16–28.

Chaturvedi, M., J. G. Bogaarts, V. V. Cozac, F. Hatz, U. Gschwandtner, A. Meyer, P. Fuhr, and V. Roth (2019). Phase lag index and spectral power as QEEG features for identification of patients with mild cognitive impairment in parkinson's disease. *Clinical Neurophysiology 130*(10), 1937 – 1944.

Chollet, F. (2018). *Deep learning with Python* (1st ed.). Manning Publications Co.

Christ, M., N. Braun, J. Neuffer, and A. W. Kempa-Liehr (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh – a python package). *Neurocomputing 307*, 72–77.

Christ, M., A. W. Kempa-Liehr, and M. Feindt (2016). Distributed and parallel time series feature extraction for industrial big data applications. *CoRR abs/1610.07717*.

Chung, J., Ç. Gülçehre, K. Cho, and Y. Bengio (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR abs/1412.3555*.

Claesen, M. and B. D. Moor (2015). Hyperparameter search in machine learning. *CoRR abs/1502.02127*.

Cozac, V. V., M. Chaturvedi, F. Hatz, A. Meyer, P. Fuhr, and U. Gschwandtner (2016). Increase of EEG spectral theta power indicates higher risk of the development of severe cognitive decline in parkinson's disease after 3 years. *Frontiers in Aging Neuroscience 8*, 284.

Cuturi, M. and A. Doucet (2011). Autoregressive kernels for time series. *arXiv:1101.0673*.

de Romblay, A. (Retrieved June 19, 2019). MLBOX. https://github.com/AxeldeRomblay/MLBox.

de Sá, A. G. C., W. J. G. S. Pinto, L. O. V. B. Oliveira, and G. L. Pappa (2017). RECIPE: A grammar-based framework for automatically evolving classification pipelines. In *Genetic Programming*, Cham, pp. 246–261. Springer.

de Silva, A., W. Hundt, and J. Malotta (2017). Automatische Erkennung und Bewertung von Low-Speed-Cashs. World Patent WO2017009201.

Deb, K. (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. New York, NY, USA: John Wiley & Sons, Inc.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine Learning Research 7*, 1–30.

Esmael, B., A. Arnaout, R. K. Fruhwirth, and G. Thonhauser (2012). Multivariate time series classification by combining trend-based and value-based approximations. In *Computational Science and Its Applications – ICCSA 2012*, pp. 392–403. Springer.

Fawaz, H. I., G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller (2018). Deep learning for time series classification: A review. *CoRR abs/1809.04356*.

Feldman, J. and R. Rojas (2013). *Neural Networks: A Systematic Introduction*. Springer.

Feurer, M., K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter (2018). Practical automated machine learning for the AutoML challenge 2018. In *ICML 2018 AutoML Workshop*, Stockholm, Sweden.

Feurer, M., A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter (2015). Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pp. 2755–2763. Cambridge, MA, USA: MIT Press.

Florea, A. C. and R. Andonie (2018). A dynamic early stopping criterion for random search in SVM hyperparameter optimization. In *Artificial Intelligence Applications and Innovations*, Cham, pp. 168–180. Springer.

Folstein, M. F., S. E. Folstein, and P. R. McHugh (1975). Mini-mental state: A practical method for grading the cognitive state of patients for the clinician. *Journal of Psychiatric Research 12*(3), 189 – 198.

Fortin, F.-A., F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné (2012). DEAP: Evolutionary algorithms made easy. *The Journal of Machine Learning Research 13*, 2171–2175.

Friedman, M. (1940, March). A comparison of alternative tests of significance for the problem of m rankings. *The Annals of Mathematical Statistics 11*(1), 86–92.

Fu, T.-C. (2011). A review on time series data mining. *Engineering Applications of Artificial Intelligence 24*(1), 164–181.

Fulcher, B. D., A. E. Georgieva, C. W. G. Redman, and N. S. Jones (2012, Aug). Highly comparative fetal heart rate analysis. In *2012 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, San Diego, USA, pp. 3135–3138.

Fulcher, B. D. and N. S. Jones (2014, Dec). Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering 26*(12), 3026–3037.

Fulcher, B. D. and N. S. Jones (2017). hctsa: A computational framework for automated time-series phenotyping using massive feature extraction. *Cell Systems 5*(5), 527–531.

Geng, Y. and X. Luo (2018). Cost-sensitive convolution based neural networks for imbalanced time-series classification. *CoRR abs/1801.04396*.

Geraedts, V., M. Kuijf, J. van Hilten, J. Marinus, M. Oosterloo, and M. Contarino (2019). Selecting candidates for Deep Brain Stimulation in Parkinson's disease: the role of patients' expectations. *Parkinsonism & Related Disorders 66*, 207 – 211.

Geraedts, V. J., L. I. Boon, J. Marinus, A. A. Gouw, J. J. van Hilten, C. J. Stam, M. R. Tannemaat, and M. F. Contarino (2018). Clinical correlates of quantitative EEG in parkinson disease. *Neurology 91*(19), 871–883.

Geron, A. (2017). *Hands-on machine learning with scikit-learn & tensorflow* (1 ed.). Sebastopol, CA, USA: O'Reilly Media.

Gijsbers, P. and J. Vanschoren (2019, 01). GAMA: Genetic Automated Machine learning Assistant. *Journal of Open Source Software 4*, 1132.

Gijsbers, P., J. Vanschoren, and R. S. Olson (2018). Layered TPOT: speeding up tree-based pipeline optimization. *CoRR abs/1801.06007*.

Glattes, K. (2016). *Der Konkurrenz ein Kundenerlebnis voraus: Customer Experience Management – 111 Tipps zu Touchpoints, die Kunden begeistern*. Springer.

Gontscharov, S., H. Baumgärtel, A. Kneifel, and K.-L. Krieger (2014). Algorithm development for minor damage identification in vehicle bodies using adaptive sensor data processing. *Procedia Technology 15*, 586–594.

Grabocka, J., M. Wistuba, and L. Schmidt-Thieme (2016). Fast classification of univariate and multivariate time series through shapelet discovery. *Knowledge and Information Systems 49*(2), 429–454.

Greenough, J. (2018). 10 million self-driving cars will be on the road by 2020. https://www.businessinsider.de/report-10-million-self-driving-cars-will-be-on-the-road-by-2020-2015-5-6?r=US&IR=T. Retrieved November 13, 2018.

Guo, H., L. B. Jack, and A. K. Nandi (2005). Feature generation using genetic programming with application to fault classification. *IEEE Transactions on Systems, Man, and Cybernetics 35*(1), 89–99.

Guyon, I., S. Gunn, M. Nikravesh, and L. Zadeh (2008). *Feature Extraction: Foundations and Applications*. Studies in Fuzziness and Soft Computing. Springer.

Guyon, I., S. Gunn, M. Nikravesh, and L. A. Zadeh (2006). *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing).* Berlin, Heidelberg: Springer.

Harrison, D. and D. L. Rubinfeld (1978). Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management 5*(1), 81 – 102.

Harshani, W. A. R. and K. Vidanage (2017). Image processing based severity and cost prediction of damages in the vehicle body: A computational intelligence approach. In *National Information Technology Conference (NITC)*, Colombo, Sri Lanka, pp. 18–21.

Harvey, D. Y. (2014). *Automated Feature Design for Time Series Classification by Genetic Programming.* Dissertation, University of California, San Diego.

Harvey, D. Y. and M. D. Todd (2015). Automated feature design for numeric sequence classification by genetic programming. *IEEE Transactions on Evolutionary Computation 19*(4), 474–489.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning, Second Edition: Data Mining, Inference, and Prediction.* Berlin, Germany: Springer.

He, K., X. Zhang, S. Ren, and J. Sun (2015). Deep residual learning for image recognition. *CoRR abs/1512.03385.*

Hinton, G. E. and R. R. Salakhutdinov (2006). Reducing the dimensionality of data with neural networks. *Science 313*(5786), 504–507.

Hofmann, M. and A. Chisholm (2016). *Text Mining and Visualization: Case Studies Using Open-Source Tools.* Taylor & Francis Group, USA.

Hossin, M. and M. Sulaiman (2015). A review on evaluation metrics for data classification evaluations. *International Journal of Data Mining & Knowledge Management Process 5*(2), 1.

Huber, W., J. Steinle, and M. Marquardt (2008). Der Fahrer steht im Mittelpunkt - Fahrerassistenz Aktive Sicherheit bei der BMW Group. *Integrierte Sicherheit und Fahrerassistenzsysteme 2048*, 123–138.

Immonen, A., E. Ovaska, J. Kalaoja, and D. Pakkala (2016). A service requirements engineering method for a digital service ecosystem. *Service Oriented Computing and Applications 10*(2), 151–172.

Immonen, A., E. Ovaska, and T. Paaso (2018). Towards certified open data in digital service ecosystems. *Software Quality Journal 26*(4), 1257–1297.

Jain, A. K., P. W. Duin, and J. Mao (2000). Statistical pattern recognition: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence 22*(1), 4–37.

Jasper, H. (1958). Report of the committee on methods of clinical examination in electroencephalography. *Electroencephalography and Clinical Neurophysiology 10*(2), 370 – 375.

Jones, D. R., M. Schonlau, and W. J. Welch (1998). Efficient global optimization of expensive black-box functions. *Journal of Global Optimization 13*(4), 455–492.

Kadous, M. and C. Sammut (2005). Classification of multivariate time series and structured data using constructive induction. *Machine Learning 58*, 179–216.

Kaiser, C., A. Stocker, and G. Viscusi (2017). Digital vehicle ecosystems and new business models: An overview of digitalization perspectives. In *Platform Economy & Business Models workshop at i-KNOW '17*.

Kantardzic, M. (2011). *Data mining: Concepts, models, methods, and algorithms* (2nd ed.). Hoboken N.J.: John Wiley and IEEE Press.

Kennedy, J. and R. Eberhart (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, Volume 4, Perth, Australia, pp. 1942–1948 vol.4.

Kilian, R., C. Gauer, J. Stein, and M. Scherer (2017). Connected vehicles and the road to revenue. http://image-src.bcg.com/Images/BCG-Connected-Vehicles-and-the-Road-to-Revenue-Dec-2017_tcm9-179631.pdf. Retrieved November 26, 2017.

Koch, M. and T. Bäck (2018, Dec). Machine learning for predicting the impact point of a low speed vehicle crash. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Orlando, USA, pp. 1432–1437.

Koch, M., W. Hundt, J. Malotta, R. Godau, M. Geiger, and J. Krieger (2019). A method of determining damage occurring in an accident between a vehicle and a collision partner on the vehicle. World Patent Application WO2019110434A1.

Koch, M., H. Wang, and T. Bäck (2018). Machine learning for predicting the damaged parts of a low speed vehicle crash. In *13th International Conference on Digital Information Management*, Berlin, Germany, pp. 179–184.

Kotthoff, L., C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown (2017). Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *The Journal of Machine Learning Research 18*(25), 1–5.

Koza, J. R. and J. P. Rice (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, UK: Bradford.

Kursa, M. and W. Rudnicki (2010). Feature selection with the boruta package. *Journal of Statistical Software 36*(11), 1–13.

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*(11), 2278–2324.

Li, Q., C. Rajagopalan, and G. D. Clifford (2014). A machine learning approach to multi-level ECG signal quality classification. *Computer Methods and Programs in Biomedicine 117*(3), 435 – 447.

Lin, J., E. Keogh, S. Lonardi, and B. Chiu (2003). A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, DMKD '03, New York, USA, pp. 2–11. ACM.

Lissandrini, M., D. Mottin, T. Palpanas, Y. Velegrakis, and H. V. Jagadish (2018). *Data Exploration Using Example-Based Methods.* Morgan & Claypool Publishers.

Liu, S., L. Li, J. Tang, S. Wu, and J. L. Gaudiot (2017). *Creating Autonomous Vehicle Systems.* Synthesis Lectures on Computer Science. Morgan & Claypool Publishers.

Long, J., E. Shelhamer, and T. Darrell (2014). Fully convolutional networks for semantic segmentation. *CoRR abs/1411.4038.*

Matheus, K. and T. Königseder (2015). *Automotive Ethernet.* Cambridge University Press.

Mikalsen, K. Ø., F. M. Bianchi, C. Soguero-Ruiz, and R. Jenssen (2018). Time series cluster kernel for learning similarities between multivariate time series with missing data. *Pattern Recognition 76*, 569–581.

Mitsa, T. (2010). *Temporal Data Mining.* Chapman & Hall / CRC data mining and knowledge discovery series. New York, NY, USA: CRC Press.

Mockus, J. (1994). Application of bayesian approach to numerical methods of global and stochastic optimization. *Journal of Global Optimization 4*(4), 347–365.

Moghaddam, B. and A. Pentland (1997). Probabilistic visual learning for object representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence 19*(7), 696–710.

Nasreddine, Z. S., N. A. Phillips, V. Bédirian, S. Charbonneau, V. Whitehead, I. Collin, J. L. Cummings, and H. Chertkow (2005). The montreal cognitive assessment, moca: A brief screening tool for mild cognitive impairment. *Journal of the American Geriatrics Society 53*(4), 695–699.

Nilsson, N. J. (1969). A mobile automaton: An application of artificial intelligence techniques. In *1st International Conference on Template Production*, Washington, USA, pp. 509–520.

Olson, R. S., N. Bartley, R. J. Urbanowicz, and J. H. Moore (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, Denver, USA, pp. 485–492. ACM.

Ortigosa-Hernández, J., I. Inza, and J. A. Lozano (2017). Measuring the class-imbalance extent of multi-class problems. *Pattern Recognition Letters 98*, 32–38.

Otonomo (2018). Are consumers willing to share connected car data? https://otonomo.io/are-consumers-willing-to-share-connected-car-data/. Retrieved November 19, 2018.

Parry, P. (Retrieved June 19, 2019). Automated machine learning for analytics & production. https://github.com/ClimbsRocks/auto_ml.

Patil, K., M. Kulkarni, A. Sriraman, and S. Karande (2017). Deep learning based car damage classification. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, Cancun, Mexico, pp. 50–54.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research 12*, 2825–2830.

Poli, R., W. B. Langdon, and N. F. McPhee (2008). *A field guide to genetic programming.* Morrisville, NC, USA: Lulu Press.

Ramasubramanian, K. and A. Singh (2017). *Machine Learning Using R.* Springer.

RCAR (2018). Rcar low-speed structural crash test protocol. Retrieved June 28, 2018. https://www.rcar.org/Papers/Procedures/rcar_Low SpeedCrashT-est2_2.pdf.

Robert Bosch GmbH (2014). *Bosch Automotive Electrics and Automotive Electronics: Systems and Components, Networking and Hybrid Drive* (5th ed.). Wiesbaden: Springer.

Rokach, L. and O. Maimon (2014). *Data Mining With Decision Trees: Theory And Applications (2nd Edition).* Series In Machine Perception And Artificial Intelligence. World Scientific Publishing Company.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65–386.

Santos, T. and R. Kern (2016). A literature survey of early time series classification and deep learning. In *Proceedings of the 1st International Workshop on Science, Application and Methods in Industry 4.0 co-located with (i-KNOW 2016), Graz, Austria*, Volume 1793. CEUR-WS.org.

Schäfer, P. and U. Leser (2017). Benchmarking univariate time series classifiers. In *Datenbanksysteme für Business, Technologie und Web*, pp. 289–298. Gesellschaft für Informatik, Bonn, Germany.

Schmidhuber, J. (2018). Prof. Schmidhuber's highlights of robot car history. http://people.idsia.ch/∼juergen/robotcars.html. Retrieved November 30, 2018.

Seiberth, G. and W. Gruendinger (2018). Data-driven business models in connected cars, mobility services & beyond. In *BVDW Research*, Volume 01/2018.

Shearer, C. (2000). The CRISP-DM model: the new blueprint for data mining. *J Data Warehouse 5*, 13–22.

Sheskin, D. J. (2000). *Handbook of parametric and nonparametric statistical procedures* (2. ed.). Boca Raton, FL, USA: Chapman & Hall / CRC.

Spurgeon, C. (2000). *Ethernet: The Definitive Guide.* Definitive Guide Series. O'Reilly.

Spurgeon, C. and J. Zimmerman (2013). *"Ethernet Switches: An Introduction to Network Design with Switches".* O'Reilly Media.

Stone, P., R. Brooks, E. Brynjolfsson, R. Calo, O. Etzioni, G. Hager, J. Hirschberg, S. Kalyanakrishnan, E. Kamar, S. Kraus, K. Leyton-Brown, D. Parkes, W. Press, A. A. Saxenian, J. Shah, M. Tambe, and A. Teller (2016). Artificial Intelligence and Life in 2030. One Hundred Year Study on Artificial Intelligence: Report of the 2015-2016 Study Panel. http://ai100.stanford.edu/2016-report. Retrieved November 5, 2016.

Swearingen, T., W. Drevo, B. Cyphers, A. Cuesta-Infante, A. Ross, and K. Veeramachaneni (2017). ATM: A distributed, collaborative, scalable system for automated machine learning. In *2017 IEEE International Conference on Big Data*, Boston, USA, pp. 151–162.

Swesi, I. M. A. O. and A. A. Bakar (2020). Recent developments on evolutionary computation techniques to feature construction. In M. Huk, M. Maleszka, and E. Szczerbicki (Eds.), *Intelligent Information and Database Systems: Recent Developments*, Volume 830 of *Studies in Computational Intelligence*, pp. 109–122. Cham, Switzerland: Springer.

The H2O.ai team (2015a). *H2O: Python Interface for H2O.* Python package.

The H2O.ai team (2015b). *H2O: Scalable Machine Learning.* Python package.

Thornton, C., F. Hutter, H. H. Hoos, and K. Leyton-Brown (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, New York, NY, USA, pp. 847–855. ACM.

Tran, B., B. Xue, and M. Zhang (2016). Genetic programming for feature construction and selection in classification on high-dimensional data. *Memetic Computing 8*(1), 3–15.

Tran, B., M. Zhang, and B. Xue (2016). Multiple feature construction in classification on high-dimensional data using gp. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, Athens, Greece, pp. 1–8. IEEE.

tsfresh (2018). Documentation of tsfresh. http://tsfresh.readthedocs.io/en/latest. Retrieved June 28, 2018.

Tsugawa, S., T. Yatabe, and et al. (1979). An automobile with artificial intelligence. In *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'79, San Francisco, USA, pp. 893–895. Morgan Kaufmann Publishers Inc.

Tuncel, K. S. and M. G. Baydogan (2018). Autoregressive forests for multivariate time series modeling. *Pattern Recognition 73*, 202 – 215.

van der Heeden, J. F., J. Marinus, P. Martinez-Martin, and J. J. van Hilten (2016). Evaluation of severity of predominantly non-dopaminergic symptoms in parkinson's disease: The SENS-PD scale. *Parkinsonism & Related Disorders 25*, 39 – 44.

Wang, H., M. Emmerich, and T. Bäck (2018). Cooling Strategies for the Moment-Generating Function in Bayesian Global Optimization. In *Evolutionary Computation (CEC), 2018 IEEE Congress on*, Rio de Janeiro, Brasil, pp. to appear. IEEE.

Wang, H., B. van Stein, M. Emmerich, and T. Bäck (2017). A New Acquisition Function for Bayesian Optimization Based on the Moment-Generating Function. In *Systems, Man, and Cybernetics (SMC), 2017 IEEE International Conference on*, Banff, Canada, pp. 507–512. IEEE.

Wang, Z., W. Yan, and T. Oates (2016). Time series classification from scratch with deep neural networks: A strong baseline. *CoRR abs/1611.06455.*

Weill, P. and S. L. Woerner (2014). Thriving in an increasingly digital ecosystem. *MIT sloan management review 56*(4), 27–34.

Welt (2018). Vernetztes Auto zum Selbermachen. https://www.welt.de/motor/news/article158417685/Vernetztes-Auto-zum-Selbermachen.html. Retrieved June 28, 2018.

Witten, I. H., E. Frank, and M. A. Hall (2011). The WEKA Workbench. In *Data Mining: Practical Machine Learning Tools and Techniques* (3. ed.)., pp. 403 – 406. Boston: Morgan Kaufmann.

Xing, Z., J. Pei, and E. Keogh (2010). A brief survey on sequence classification. *SIGKDD Explorations Newsletter 12*(1), 40–48.

Xue, B., M. Zhang, W. N. Browne, and X. Yao (2016). A survey on evolutionary computation approaches to feature selection. *IEEE Transactions on Evolutionary Computation 20*(4), 606–626.

Xue, B., M. Zhang, Y. Dai, and W. N. Browne (2013). PSO for feature construction and binary classification. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, New York, USA, pp. 137–144. ACM.

Ye, L. and E. Keogh (2009). Time series shapelets: A new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, New York, NY, pp. 947–956. ACM.

Zhang, W. (2010). *Computational Ecology: Artificial Neural Networks and Their Applications.* World Scientific.

Zhu, R., Z. Wang, Z. Ma, G. Wang, and J.-H. Xue (2018). LRID: A new metric of multi-class imbalance degree based on likelihood-ratio test. *Pattern Recognition Letters 116*, 36–42.

Zuur, A., E. Ieno, and C. Elphick (2010). A protocol for data exploration to avoid common statistical problems. *Methods in Ecology and Evolution 1*(1), 3–14.

Acronyms

AMLPA Automated Machine Learning for Production and Analytics. 30, 45, 93

ATM Auto Tune Models. 31, 45, 91–94

AUC Area Under the Curve. 15

AutoML Automated Machine Learning. 1–5, 29–31, 45, 91–95, 113

CNN Convolutional Neural Network. 28, 43, 47, 49

CNN+LSTM Convolutional Neural Network + Long Short-Term Memory. 47, 49, 91, 92, 114

CRISP-DM Cross Industry Standard Process for Data Mining. 97, 103–105

DTW Dynamic Time Warping. 89, 91–93

ECU Electronic Control Units. 107, 108

EEG Electroencephalography. 71–77, 79–81, 84–86

FCN Fully Convolutional Network. 34, 47, 91–93

GAMA Genetic Automated Machine Learning Assistant. 30, 45, 91–93

GDPR General Data Protection Regulation. 97, 108, 111

HCPGP Hand-Crafted Pipeline with Genetic Programming. 42, 44, 92, 93, 95

HCTSA Highly Comparative Time-Series Analysis. 21, 22

KNN k-Nearest-Neighbor. 74

kPCA kernel Principal Component Analysis. 43–45, 92

LRID Likelihood Ratio Imbalance Degree. 88, 89

LSTM Long Short-Term Memory. 28, 47, 49

MIP-EGO Mixed-integer Parallel Efficient Global Optimization. 41, 67, 68, 70

MTSC Multivariate Time Series Classification. 33, 34, 47

PCA Principal Component Analysis. 22, 43, 115

PD Parkinson's disease. 71–74, 81, 84, 85

PHCP Plain Hand-Crafted Pipeline. 35, 40, 41, 51, 54, 62, 63, 91–93

qEEG quantitative Electroencephalography. 73, 74

RECIPE Resilient Classification Pipeline Evolution. 31, 45, 91–93

ResNet Residual Network. 34, 47, 91–93

RFECV Recursive Feature Elimination with Cross-Validation. 24, 65–67

RNN Recurrent Neural Network. 28

ROC Receiver Operating Characteristic. 15, 42, 80

SVM Support Vector Machine. 74

TPOT Tree-based Pipeline Optimization Tool. 29, 30, 45, 91, 92, 94

TSC Time Series Classification. 17, 33, 34

tsfresh Time Series Feature Extraction based on Scalable Hypothesis Tests. 21, 22, 36, 40, 42–44, 55, 56, 65, 70, 77, 78

English Summary

Machine Learning is becoming more and more a substantial technology for industry. It allows to learn from previous experiences in an automated way to make choices based on the learned experience. This principle can be used to optimize and / or automatize processes in industry applications. A suitable process from the car industry is for example the quality assessment in assembly lines. Instead of assessing the quality in a time-consuming manual process, machine learning based image recognition methods could be applied to detect failures. Next to enhancing the existent, machine learning enables the development of completely new products like autonomous driving or services which are purely driven by data. The development of such new data-driven products is often a long procedure. Even the application of machine learning algorithms to specific problems is mostly not straightforward. To illustrate this, we introduce a data-driven service from the automotive industry called Automated Damage Assessment. Based on the gained experience from such data-driven service developments, we propose a methodology to develop data-driven services in an accurate and fast manner.

Especially data-driven services in the automotive domain could be based on sensor data, i.e. data which is recorded from on-board sensors over time in a so-called time series. In many cases, time series from more than one sensor can be used which is a so-called multivariate time series. The existent methods to solve multivariate time series classification-problems are often complex and developed to solve a specific problem without being scalable to solve various problems. To overcome this, suitable approaches with different complexities, applied on multiple publicly available data sets, as well as medical and industrial data sets are proposed in this work. As starting point, we have designed a plain feature-based pipeline and applied and enhanced it on several real-world data sets from the automotive and medical domains. Recently, numerous AutoML methods have been proposed. AutoML aims at building optimized models in an automatized way. We have empowered those promising AutoML methods with another method to solve multivariate time series problems with the result that some of those techniques are suitable for this task. We have compared all those approaches on several publicly available data set with the results that especially two AutoML approaches, namely GAMA and ATM, as well as our PHCP approach are most suitable to solve this particular multivariate time series problems.

In order to consider the interaction of features from different time series, an approach is developed based on the evolutionary algorithm technique called genetic programming. It uses the principle of biological evolution to compute combination of features. Based on our results we can state, that the assumption, that combining several features can improve the performance instead of using the pure features, is true.

Samenvatting

Machine Learning wordt steeds meer een belangrijke technologie voor de industrie. Het biedt de mogelijkheid om te leren van voorgaande ervaringen op een geautomatiseerde wijze om zo beslissingen te maken die gebaseerd zijn op aangeleerde ervaring. Dit principe kan worden gebruikt op processen te optimaliseren en / of te automatiseren voor industriële doeleinden. Een dergelijk geschikt proces binnen de auto-industrie is bijvoorbeeld de kwaliteitsbepaling bij de lopende band. In plaats van deze kwaliteitsbepaling te verrichten in een tijdrovend, handmatig proces, kunnen beeldherkenningsmethoden gebaseerd op machine learning toegepast worden om storingen te detecteren.

Naast het verbeteren van de huidige situatie kan machine learning ook bijdragen aan de ontwikkeling van volledige nieuwe producten, zoals autonoom rijden of het aanbieden van diensten die volledig aangestuurd worden door data. De ontwikkeling van deze nieuwe data-gedreven producten is veelal een lange procedure en zelfs de toepassing van machine learning algoritmes voor specifieke problemen is vaak niet eenvoudig. Om dit te illustreren introduceren wij een data-gedreven dienst vanuit de auto-industrie genaamd Automated Damage Assessment. Gebaseerd op reeds vergaarde ervaringen van deze data-gedreven ontwikkeling van diensten, beschrijven wij hier een methodologie om data-gedreven diensten nauwkeurig en snel te ontwikkelen.

Met name data-gedreven diensten binnen het domein van de auto-industrie kunnen gebaseerd worden op sensor data, d.w.z. data welke wordt geregistreerd door sensoren binnenin de auto door de tijd heen, in zogenaamde tijdreeksen. In veel gevallen kunnen tijdreeksen van meer dan één sensor worden gebruikt, resulterende in zogenaamde multivariate tijdreeksen. De bestaande methoden om classificatie-problemen met multivariate tijdreeksen te verhelpen zijn vaak complex en ontwikkeld voor een specifiek probleem, zonder schaalbaar te zijn om

verschillende problemen op te lossen. Om dit te verhelpen demonstreren wij in dit proefschrift benaderingen van verschillende complexiteit, toepast op meerdere algemeen-verkrijgbare datasets, medische- en industriéle datasets. Als startpunt hebben wij een algemene pijplijn ontwikkeld, gebaseerd op tijdreeks-features en toepast en verbeterd binnen verschillende bestaande datasets uit de auto-industrie en medische domeinen.

Recent zijn een aantal AutoML methoden beschreven, welke er naar streven om geoptimaliseerde modellen op een automatische wijze te ontwikkelen. Wij hebben deze veelbelovende AutoML methoden versterkt met een andere methode om multivariate tijdreeks-problemen te verhelpen, met als resultaat dat enkele van deze technieken geschikt blijken te zijn voor dit doeleinde. We hebben al deze technieken toegepast op algemeen-verkrijgbare datasets, waaruit blijkt dat met name twee AutoML methodes, namelijk GAMA en ATM, evenals onze eigen PHCP methode het beste geschikt zijn om deze multivariate tijdreeks-problemen te verhelpen.

Om rekening te houden met de interactie van features uit verschillende tijdreeksen is een methode ontwikkeld, gebaseerd op de evolutionaire algoritmen techniek genaamd genetisch programmeren. Het gebruikt het principe van biologische evolutie om combinaties van features te berekenen. Gebaseerd op onze resultaten kunnen wij stellen dat de aanname dat de combinatie van verschillende features betere prestaties levert dan bij het gebruik van de pure features, waar is.

About the Author

Milan Koch was born 1990 in Ostercappeln, Germany. He started as a Ph.D. student in 2017 at the Leiden University, The Netherlands (promotor: Prof. Dr. Thomas Bäck). During his bachelor's study in Automotive Engineering at the Osnabrück University of Applied Sciences, Osnabrück, Germany, Milan was on an industrial fellowship for outstanding achievements in the study program. In 2014, he received his bachelor's degree. In 2017, Milan received his master‘s degree in Automotive Engineering from the Hamburg University of Applied Sciences, Hamburg, Germany. His research interests are developing, optimizing and comparing machine learning methods for real-world problems, especially for time-series problems. He is investigating methods for using simulated car sensor data in addition to real test data for machine learning applications. Furthermore, he works on the interaction of customer and vehicle to discover new efficient methods to create data-driven services.