Data-driven machine learning and optimization pipelines for real-world applications

Koch, M.

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 $i \in \{1, \ldots, q\}$, $F1$ 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:

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

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 $\mathbf{X} = (\mathbf{x}_1^\top, \ldots, \mathbf{x}_L^\top) \in \mathbb{R}^{L \times d}$ (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

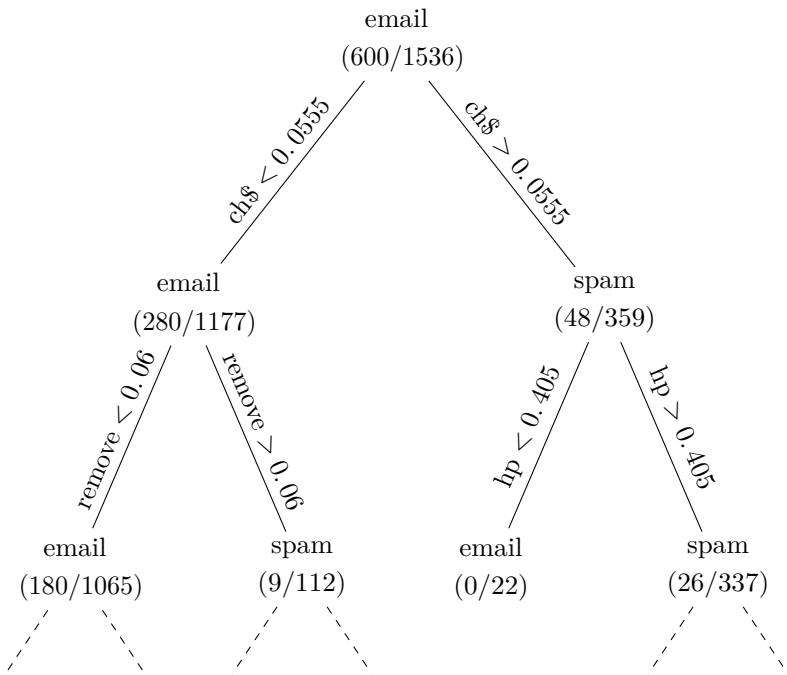

Figure 3.1: A cutout of a decision tree for spam email classification. Each node represents a classification. The split conditions are shown along the branches. The ratio in the nodes illustrate misclassification rates, based on Hastie et al. (2009).

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

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

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

Another impurity measure is called entropy:

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

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.

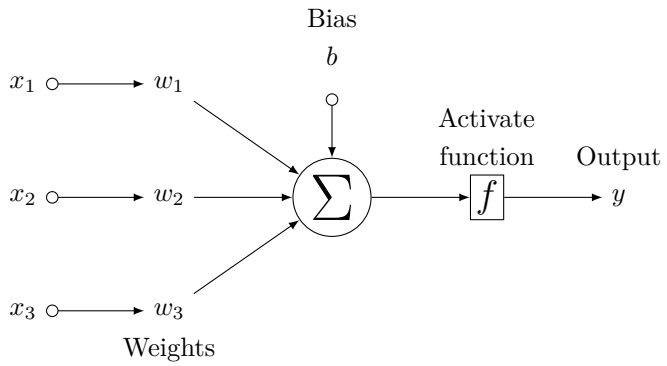

Figure 3.2: The perceptron is a single-layer neural network architecture, based on (Rosenblatt, 1958; Aggarwal, 2018).

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).

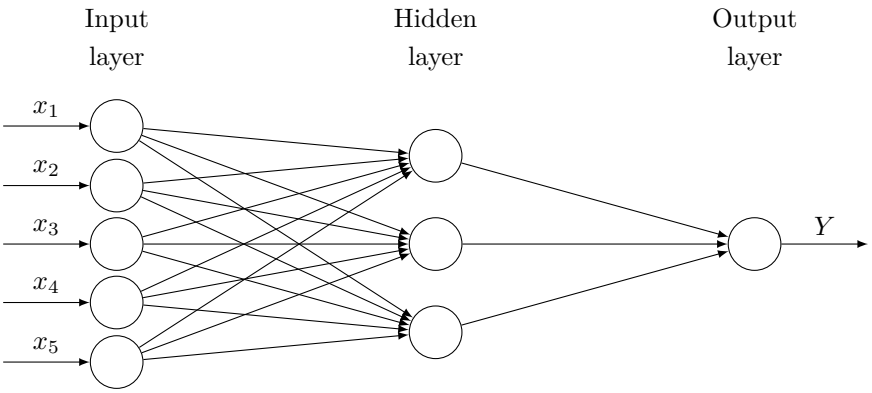

Figure 3.3: A multilayer perceptron with one hidden layer, based on (Aggarwal, 2018).

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.