technische universität
dortmund

Masterthesis

Dynamic Model Selection for Automated Machine Learning in Time Series

Florian Priebe

March 2019

Gutachter:
Prof. Katharina Morik
M.Sc Alexey Egorov

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl für künstliche Intelligenz (LS VIII)
http://www-ai.cs.uni-dortmund.de

Contents

Chapter 1

Introduction

Time series forecasting is important in many domains and basically every scientific field records measurements over time that results in time series data. Understanding and forecasting time series data has become a critical task in fields from industry to research.

The changing nature of time series makes forecasting a difficult and challenging task. Models have different performance across the input space and their performance may change over time[1, 17]. To get the best performance for the whole input space, we want to use of many models that have different areas of expertise.

1.1 Motivation

Ensembles are a very powerful method that make use of multiple models. Ensembles have proven their performance in many real life tasks and are considered a robust and powerful method. They can have a high demand in computational resources. Recent advances in powerful devices make computational constraints less concerning, so that ensembles are a viable solution nowadays.

This thesis proposes a method that uses dynamic selection to construct ensembles for time series forecasting. Dynamic selection is used to construct the ensemble for each prediction depending on the given example. The idea behind using the dynamic selection is to make the ensemble adaptable to the changes of the time series and the changes of each models performances. Besides selecting an ensemble, dynamic selection can be used to select a single model for each prediction. Selecting only a single model would neglect the improvements that ensemble methods can deliver in terms of making the most of multiple models. Dynamic selection methods have been developed and used for classification tasks. Time series forecasting can be viewed as a regression task and adapting dynamic selection methods to regression tasks poses another challenge. To make the selection of models, we want to compare patterns in the time series to patterns in already evaluated validation data. If we can find similar examples in the validation data, the performance recorded for the validation data can be a good estimate for the future performance of each model on the current example. The search for similarity in time series is often guided by distance metrics that are specifically designed for time series.

Regression tasks itself pose more challenges than classification task, especially in terms of outliers. We do not only have to consider outliers in the features and in the concept, but also in the observed target values[56]. To prepare for these challenges, the integration of the ensemble members is a crucial task as well. Common techniques include voting, cascading and stacking. Cascading combines the predictions of the ensemble members in the feature set in an iterative way. Stacking learns a meta-model that combines the outputs of the base models to a prediction. Finally, voting uses a weighted average of outputs of the base models.

Other research methods specifically focus on the combination of forecasting models by using different integration methods. Voting[77] and stacking[23] have been used to integrate ensembles for time series forecasting. The selection step is always neglected and often treated with a selection of an arbitrary number of ensemble members based on recent performance.

Our method can be seen as combination of meta learning and voting. The dynamic selection, which finds similar examples in validation data to estimate model performance, can be seen as a meta learner. After selecting the ensemble members, we will combine their predictions with a weighting method using each ensemble members errors on past predictions.

To evaluate the proposed ideas and models, we test them on multiple real-life time series with a fixed pool of models. This will give an empirical insight into the performance of the ensemble. In these experiments we compare our method against state-of-the-art forecasting methods. We also consider different parameters and variations of our method, to study their share of the overall performance.

1.2 Structure of the thesis

First, we look in chapter 2 at the notations and definitions that are needed in the domain of the time series. After introducing the notations and typical tasks in the domain, we define the evaluation metrics that are used to measure the quality of predictions in the tasks. Then, we define the specific task that is the main target in this thesis.

With the task defined, we look at related work and common approaches in chapter 3. This includes presenting commonly used models and ensemble learning. We also discuss a common theoretical approach for the decomposition of model errors. This decomposition can be expanded for ensembles and gives more insight to what makes ensembles successful.

Chapter 4 presents the approach of dynamic model selection that is used for developing a new ensemble forecasting algorithm. The origins of dynamic model selection, existing terms and methods are presented.

We present our own ensemble forecasting algorithm in chapter 5. We discuss the different methods that are combined in the algorithm, to make the prediction of the ensemble. Finally, we evaluate the new algorithm against standard baseline methods for ensembles and state-of-the-art methods for forecasting in chapter 6.

Chapter 2

Definitions and Problem Formulation

A time series is a sequence of observations y_t, where each observation is recorded at a specific time t. There are two general types of time series. Discrete-time time series, where the set of times is discrete, which includes all examples where a value is recorded in fixed intervals of time. The other type are continuous-time time series, where the values are recorded continuously over a time interval [15]. Continuous-time time series are for example recordings of analog signals.

Recordings of continuous-time time series are made by sampling the series at small and equal intervals of time. The process of discretization of continuous signals is very common in computer science. If the sampling interval is high enough, the loss of information is very small [25]. So in the following we focus on discrete-time time series.

2.1 Time Series Definitions

When defining time series, we want to go beyond the most simple definition, but include context as well. Describing a time series as a sequence of observations does not provide much context and does not help to clearly describe target tasks on this special data type. A more formal and general definition was formalized by Mierswa and Morik [65], which deals with value series in general to cover time series in particular. Each element x_i of the series consists of an index and a value vector. The value vector is a m-dimensional vector and specifies a point in the value space.

2.1.1 Definition (Value Series). A value series is a mapping $x : \mathbb{N} \to \mathbb{R} \times \mathbb{C}^m$. The point of the series with index n is noted as x_n. $x_{1,\dots k}$ is the sequence of points starting at index 1 and ending at index k with length k.

The definition covers time series, but also for example all common transformations for time series like Fourier- or Wavelet transformations. In the following, we will focus on time series. For these the index component is a measurement of time.

2.1.2 Definition (Time Series). A time series is a temporal sequence of values $Y = \{y_1, y_2, \ldots, y_t\}$, where y_i is the value vector of Y at time i. In this thesis we only consider numeric time series with $y_i \in \mathbb{R}^m, \forall i \in \{1, 2, \ldots, t\}$.

Since our main target is forecasting the values of the time series, we changed the notation to Y, because it is commonly used. We will introduce X again as the feature space in the concrete problem definition.

Time series with dimension $m > 1$ are called multivariate time series. For $m = 1$ the time series is called univariate. In the following, we will focus on univariate time series and discuss possible extensions for multivariate time series later.

2.2 Time Series Characteristics

Tim series analysis is about learning something unknown from the known. This is also true for statistical inference in general. The known in terms of data is generated by an underlying distribution. It is necessary that this distribution or at least some features of it do not change over time. Otherwise, it is not possible to use the known to learn about the unknown. Different types of stationarity are assumed to enable learning. One of the main difference between time series analysis and classical statistical analysis is the dependency in the data. This dependency can be measured in different ways. In this section we will describe types of stationarity and dependency measures that are required for learning.

A model that describes the stochastic or random behavior of the time series is called a stochastic process. This process can be described as a set of random variables $\{X_t, t \in T\}$, where T is the index set on which the process is defined and a probability distribution over the set of random variables $p(X_t, t \in T)$.

2.2.1 Definition (Stochastic Process [24]). A stochastic process can be defined as a set of random variables $\{X_t, t \in T\}$. The process is usually described by its *moments*:

- Mean $\mu(t) = E[X(t)]$
- Variance $\sigma^2(t) = VAR[X(t)]$
- Autocovariance $\gamma(t_1, t_2) = E\{[X(t_1) - \mu(t_1)][X(t_2) - \mu(t_2)]\}$

For a given time series $X = \{x_1, x_2, \ldots, x_t\}$, a single value x_t is an observation of the underlying random variable X_t. The time series itself is a realization of the stochastic process. A time series is stationary if the properties of the series' stochastic process are not changing through time. The properties most considered here are the mean and the variance of the series.

2.2.2 Definition (Strictly Stationarity). A stochastic process is *strictly stationary*, if its properties are unaffected by a change of time origin. That is, if the joint probability distribution associated with m observations $x_{t_1}, x_{t_2}, \ldots, x_{t_m}$, made at any set of times

$t_1, t_2, \ldots, t_m$, is the same as that associated with m observations $x_{t_{1+k}}, x_{t_{2+k}}, \ldots, x_{t_{m+k}}$, made at times $t_{1+k}, t_{2+k}, \ldots, t_{m+k}$ [9]. This means that the mean $\mu(t) = \mu$ and the variance $\sigma^2(t) = \sigma^2$ of the process are constant and do not depend on the point in time t.

The autocovariance in this case depends only on the lag $\tau = (t_2 - t_1)$.

$$\begin{aligned}\gamma(\tau) &= E\left\{[X(t) - \mu][X(t+\tau) - \mu]\right\} \\ &= Cov[X(t), X(t+\tau)]\end{aligned}$$

$\gamma(\tau)$ is called the autocovariance coefficient at lag τ. Because it is dependent on the value ranges of X_t, it is standardized to the autocorrelation coefficient $\rho(\tau) = \gamma(\tau)/\gamma(0)$. Calculating the autocorrelation coefficients is a very common way to check if a given time series is stationary.

A less strict way of stationarity is the *second-order stationarity.*

2.2.3 Definition (Second-Order Stationarity [24]). A stochastic process is *second-order stationary*, if its mean is constant and the autocovariance depends only on the lag.

- Mean $\mu = E[X(t)]$
- Autocovariance $\gamma(\tau) = Cov[X(t), X(t+\tau)]$

There are some simple examples for stochastic processes [25]. Take for example a purely random process, where the process is a sequence of uncorrelated, identically distributed random variables with zero mean and constant variance. This process produces, what is often named *white noise* and is a stationary process. Another example is a random walk, where each X_t is given by $X_t = X_{t-1} + Z_t$, with Z_t being white noise. This process is not stationary as its variance is increasing, but the series of differences of a random walk $(X_t - X_{t-1})$ forms a stationary series. Differencing a time series is a common method for transforming a non-stationary series into a stationary one [25]. We will use this later in preprocessing the time series data for our task.

Other stochastic processes are *autoregressive* processes and *moving average* processes. These are also very practical models for solving time series tasks, that will be presented in subsection 3.1.2.

Real time series data is often non stationary. A common assumption is that a non stationary time series can be decomposed and one of the parts is a stochastic component that is stationary. We decompose the time series $\{Y_t\}$ as

$$Y_t = f_t + s_t + X_t, \tag{2.1}$$

with X_t as the stochastic component, s_t the seasonal component and f_t the trend component.

The seasonal component s_t is a periodic function. The trend component f_t is a slowly changing function. Estimating trend and seasonality are main tasks of time series analysis.

Often the seasonal and trend component are removed to focus on the remaining stochastic component. Being aware of trend and seasonality is important for understanding a time series. Handling them for tasks can be done by preprocessing or usage of stochastic models that take both into account.

2.3 Time Series Tasks

Several machine learning tasks can be applied to time series data. In the following, we present two main tasks that are used in this work.

2.3.1 Classification

Classification tasks are centered around predicting a class label for a given series or sequence. So the data consists of pairs (x_i, y_i), where x_i is a time series and y_i is the label. The points of the time series are treated as features and the task is technically handled as a standard classification problem.

Any machine learning classification model can be used for the task. Due to the specific nature of time series, some approaches are more suitable than others. One simple approach, that works particularly well for this task, is the Nearest-Neighbor approach. The distance between the new example and each of the training examples is computed and the label is chosen based on a voting strategy among the label of the closest training instances. The main idea of this approach is to compute the similarity between two time series by using a distance metric. The use of an appropriate distance metric is very important for time series, because a simple metric might not be expressive enough to compare patterns in two series. A metric, specific to time series, will be discussed in subsection 5.1.3.

State-of-the-Art approaches in time series classification use various distance measures, transformations of the series and feature extractions [4, 79, 8, 64]. A big comparison of classification approaches was done in [3]. The comparison showed that the use of the Nearest Neighbor algorithm with a time series specific distance measure is a reasonable comparison benchmark, with a good ratio of performance and computational cost.

2.3.2 Forecasting

Given a time series $Y = \{y_1, y_2, \ldots, y_t\}$ the task of forecasting consists of predicting the next values of the series on a well-defined horizon h. There are variations of the forecasting task, which differ in the size of the prediction horizon. The simplest version with $h = 1$ is step-wise forecasting. If $h > 1$, then it is required to predict multiple steps at once. This is called multi-step forecasting.

The theory for time series, explained in section 2.2, assumes that a given time series is a realization of a stochastic process. This stochastic process f can be regarded as a function that takes past values and generates the next value.

$$y_{t+1} = f(y_1, y_2, \ldots, y_t) \tag{2.2}$$

If we want to predict the next value of the time series, we have to construct a model $\hat{f}_\theta$ that approximates the true process f. It has been shown [54, 26] that the forecasting error increases, when the horizon itself does. Therefore, predicting multiple steps at once is harder than doing step-wise forecasting. For this thesis we will focus on step-wise forecasting.

2.3.1 Definition (Forecasting). Given a time series $Y = \{y_1, y_2, \ldots, y_t\}$, the task is to predict the value of y_{t+1}. A window of size h of the past values is used to make the prediction. The model $\hat{f}$ computes

$$y_{t+1} = \hat{f}(y_{t-h}, y_{t-h+1}, \ldots, y_t) \tag{2.3}$$

Forecasting is a problem heavily covered in statistics. There are statistical models that are stochastic processes with their parameters defined by the given data. Since there is usually just one realization, the given time series, of the true stochastic process, the models can only estimate the real parameters even if the statistical model is of the same type as the true stochastic process.

From a machine learning perspective, a forecasting problem can be viewed as a regression task. Regression tasks present some extra difficulties for predicting. In contrast to classification tasks, a regression task has a target with an undefined value range. This raises some practical problems, due to the fact that some supervise learning algorithm can only predict values inside the value range of the given training set data. Another problem of regression tasks are outliers. A regression task has three probability distributions. The distribution of features $p(x)$, of target values $p(y)$ and the underlying concept $p(y|x)$. For a regression task we can observe outliers in all three of them. Especially target outliers are difficult to handle, because of the unknown value range. These problems make dealing with outliers one of the key elements of a successful forecasting algorithm.

To turn forecasting into a problem of supervised learning, we need to construct a feature space of the time series, which we use to make predictions of the target. One way to transform the time series into a feature set is time delay embedding [84]. The time series gets embedded in a euclidean space with embedding dimension K. This constructs a set of observations X and each observation is a feature vector $x_i \in \mathbb{X} \subset \mathbb{R}^K$, that consists of the previous K values of the time series. To each observation x_i belongs a target value $y_i \in \mathbb{Y} \subset \mathbb{R}$ and the set of all pairs (x_i, y_i) is a dataset suitable for a supervised learning task. In the case of numeric time series, this task is a regression task, where we want to construct a model $f : \mathbb{X} \rightarrow \mathbb{Y}$.

2.3.2 Definition (Regression task). Given a time series $Y = \{y_1, y_2, \dots, y_t\}$, the task is to predict the value of y_{t+1}. Find a model $\hat{f} : \mathbb{X} \to \mathbb{Y}$, where $\mathbb{X}$ is the feature space with $x_i \in \mathbb{X} \subset \mathbb{R}^K$ and $x_i = (y_{i-K}, y_{i-K+1}, \dots, y_i)$, so that

$$y_{t+1} = \hat{f}(x_t) \tag{2.4}$$

It can be assumed that there is a true model f that generates the time series. Our goal is to find a model that $\hat{f}$ makes predictions that are as close as possible to the true model.

2.3.3 Forecasting evaluation

Different distance measures can be used to measure the quality of a model's predictions. Given a set of predictions $\hat{Y} = \{\hat{y}_{t+1}, \hat{y}_{t+2}, \dots, \hat{y}_{t+n}\}$ and the respective truth values $Y = \{y_{t+1}, y_{t+2}, \dots, y_{t+n}\}$, we can compute different distance measures. The most common metrics for regression task are Mean Absolute Error (MAE) and Root mean squared error (RMSE).

2.3.3 Definition (Mean absolute error).

$$MAE = \frac{1}{n} \sum_{t=1}^{n} |y_t - \hat{y}_t| \tag{2.5}$$

2.3.4 Definition (Root mean squared error).

$$RMSE = \frac{1}{n} \sqrt{\sum_{t=1}^{n} (y_t - \hat{y}_t)^2} \tag{2.6}$$

RMSE is more sensitive to outliers than MAE, because extreme differences between prediction and truth value count more to the total value. Depending on the time series this can be a good characteristic, because the peformance measure focuses on making at most small errors. The MAE on the other hand works opposite. If the models tends to make big errors, these can compensated by small errors everywhere else. Hence, choosing the right evaluation metric is important measuring the performance of a model.

Both error metrics are dependent on the value range of the time series. This becomes an issue when the performance of a model is compared over different time series. A MAE of 10 on a time series with values between 1000 and 5000 is not bad, but making such an error on values between 10 and 50 is much worse. A solution to this problem is normalizing the error metrics relative to value ranges. For the RMSE this is done by dividing by the value range.

2.3.5 Definition (Normalized root mean squared error).

$$RMSE = \frac{1}{n} \sqrt{\sum_{t=1}^{n} (y_t - \hat{y}_t)^2 / \text{range}(Y)} \tag{2.7}$$

For the MAE, the normalization works a little different. The most common calculation here is the symmetric mean absolute error percentage.

2.3.6 Definition (Symmetric mean absolute error percentage).

$$SMAPE = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|y_t - \hat{y}_t|}{|y_t| + |\hat{y}_t|} \tag{2.8}$$

This measure has the advantage that the values can be directly interpreted as percentage errors. SMAPE has a restriction: All values have to be positive.

2.3.4 Validation Strategies for Time Series Data

As discussed above we can calculate a models performance by the distance between a models predictions and the true values of the time series. The main goal for our model $\hat{f}_\theta$ is to perform well on new, unseen data. To formalize this, we define an objective for the regression task (Definition 2.3.2).

2.3.7 Definition (Regression Objective). With f as the true model, we want to find

$$\hat{f}^*_\theta = \arg\min_\theta E\left[(f(x) - \hat{f}_\theta(x))^2\right] \quad \forall x \in \mathbb{X} \tag{2.9}$$

We can only estimate this expected error, because in practical machine learning we have only access to a sample of $\mathbb{X}$. To make the most out of this sample and to estimate the error for unseen data, the data is often split in different parts for training and evaluation.

There is a long history of methods splitting the data in machine learning and dealing with estimating the expected error. When there are problems with the estimation, two problems that commonly occur are *overfitting* and *underfitting.* When a model is overfitted, then it's parameters are too tightly tuned to the training data. A typical symptom for this problem is a very low error on the training part. This might go unnoticed if the model is not tested against unseen data from an evaluation set. On the unseen data the error of an overfitted model is a lot higher than on the training set. The problem of underfitting is the reverse situation. The model has not learned enough of the patterns in the data and can not make good predictions. In this case the difference between error on the evaluation set and the training set is not big, but the predictions are often just bad on both sets. Over- and underfitting are tightly bound to the model's number of parameters. For models that use stochastic optimization procedures like Neural Networks, the number of iterations spent on training the model is bound to over- and underfitting as well. In subsection 3.3.1 this problem is explained in detail resulting in a discussion about *bias* and *variance* of models.

For now, we want to focus on strategies, which split the data, to estimate the expected error. Out-of-sample methods and cross-validation are good ways for estimating the expected error. Cross-validation is particularly efficient in terms of maximizing the usage of the data [2]. For most datasets one can assume that each example x_i from the sample $X \subset \mathbb{X}$ is independent and identically distributed ($i.i.d$). This does not hold for time series data, because there is a dependency in time for the examples. The most common strategies for splitting the data use the assumption that the examples are *i.i.d.* This is especially true for cross-validation.

For time series data there are concerns for using cross-validation, because it does not take the dependencies between the examples into account. A point of emphasis for validation strategies is to keep the order of examples in time series data intact. This way a model is tested on examples that are in the future, in terms of the index of the series, in comparison to the training data. Cerqueira et al. presented a study [21], where they compared different estimation methods. In contrast to former studies[5, 6, 7] that tried to prove the usefulness of cross-validation for time series data, Cerqueira et al. show that cross-validation requires additional assumptions like stationarity. The authors conclude that strategies, which maintain the temporal order of the data, provide better error estimations. These strategies are Out-of-sample methods that split the data into contiguous blocks and take the temporal order into account.

2.4 Problem Definition

The problem covered in this thesis is dynamic time series forecasting. Given a time series $Y = \{y_1, y_2, \ldots, y_t\}$, the task is to predict the value of y_{t+1}. The predicting model or ensemble should be selected dynamically for each prediction. The predictions are evaluated using SMAPE and the goal is to minimize the expected error on unseen data. We estimate the expected error with a validation strategy.

Typical solutions to forecasting use traditional regression and time series analysis models. The models are used individually or combined as an ensemble. Except for retraining the parameters, which is used to handle changes in the concept of the time series, the models are kept static. This static use is in strict contrast to the dynamic nature of time series forecasting. The performance of forecasting models changes over time[1] and across the input space[17]. Hence, there exists a need for dynamic changes in the selected models and the way they are combined. Recent works in field of classification tasks have made the model selection dynamic [14, 29]. The dynamic model selection changes the selected models for each prediction. Dynamic selection has not been used directly for forecasting tasks. Only dynamic combination methods for forecasting have been presented [81, 23]. In this thesis we present dynamic ensemble selection forecasting (DES-Forecasting), which uses dynamic model selection for constructing an ensemble and dynamic combination methods for the ensemble prediction.

Chapter 3

Common Approaches and Related Work

In this chapter, we want to present some common approaches and related work that is used to solve time series forecasting. We first present a series of models that are suitable for time series forecasting. These models will be later used as model pool, for more advanced ensemble models, that make use of all available models.

3.1 Candidate Models

There are different models that are widely used in time series forecasting. These models can be split in two major groups with regard to different backgrounds.

The first group consists of models originated from statistics that work directly on the time series. These models have a close connection to stochastic processes and the time series theory. The other group originates in the machine learning research. The important aspect for this group is that the forecasting task is turned into a task of supervised learning, with a feature set and a target vector. The feature set is constructed by time delay embedding[84] as already presented in subsection 2.3.2. Since we focus on numeric time series, the target vector contains real values. This makes the supervised learning task a regression task. There exists a variety of machine learning models for regression. We will present some models that have been successful in forecasting time series.

This list of candidate models is not exhaustive for both groups and the choices are arbitrary. These are the candidate models that will later build the model pool for the ensemble methods. At first, we look at statistical models namely Exponential Smoothing and ARIMA, then we continue with the models from the machine learning field.

3.1.1 Exponential Smoothing

A very naive way of forecasting is to predict the next value by using the last one.

$$\hat{x}_t = x_{t-1} \tag{3.1}$$

This naive way can often be used as a guideline, because any serious model should perform better. Another way of predicting is to use the average of all seen values.

$$\hat{x}_t = \frac{1}{t}\sum_{i=1}^{t} x_i \tag{3.2}$$

This average prediction can also be a guideline. Both of these methods are extreme ways of forecasting. The first assumes that all values of the past are irrelevant, except for the last, while the other one assumes that all past values are equally important. It is intuitive to assume that the importance of a value decrease with distance to the current point in time. Therefore, we want to decay the weight of the past values. The further away a previous value is, the less important it is for the prediction. Exponential smoothing uses such decaying weights by exponentially diminishing the weights of past values.

$$\hat{x}_t = \alpha x_{t-1} + \alpha(1-\alpha)x_{t-2} + \alpha(1-\alpha)^2 x_{t-2} + \dots \tag{3.3}$$

This equation can also be expressed recursively. The next value is predicted smoothing the current value and the smoothed prediction for the current value.

$$\hat{x}_t = \alpha x_{t-1} + (1-\alpha)\hat{x}_{t-1} \tag{3.4}$$

If the recurrence is unfolded we get Equation 3.3.

This simple exponential smoothing is suitable for forecasting data with no clear trend and seasonal pattern. More complex versions of exponential smoothing can account for trends in the time series. The prediction is then a combination of level and trend of the series. The first term l_t is the level of the series computed recursively as seen in Equation 3.4. A second term b_t that is a smoothing of the trend or slope in the level of the series, is added to the level for the prediction.

$$\hat{x}_t = l_t + b_t \tag{3.5}$$

$$l_t = \alpha x_{t-1} + (1-\alpha)\hat{x}_{t-1} \tag{3.6}$$

$$b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1} \tag{3.7}$$

If we want to include seasonality in the data, then a third term that models the seasonality is added in the same way. ETS [53] combines all these ideas in a model. A more complex version is TBATS [31], which also includes transformation and some parts of ARMA models.

3.1.2 Autoregressive Models

As discussed in section 2.2, there is the assumption that a time series is a realization of a stochastic process. In an autoregressive integrated moving average model (ARIMA), the future value of a variable is assumed to be a linear function of several past observations and random errors. An ARIMA models is composed of combining an autoregressive and a moving average process with additional differencing.

A stochastic process is an autoregressive process of order p, AR(p) if

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + Z_t \tag{3.8}$$

So the next value in an autoregressive process of order p is a linear combination of the last p values.

A stochastics process is a moving average process of order q, MA(p) if

$$X_t = Z_t + \theta_1 Z_{t-1} + \cdots + \theta_q Z_{t-q} \tag{3.9}$$

where $\{Z_t\}$ is a purely random process with zero mean and constant variance.

A combination of both of these processes are ARMA processes. They have parameters p, q and compute the next value by

$$X_t = Z_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{j=1}^{p} \theta_j Z_{t-j} \tag{3.10}$$

Both of these processes expect the time series to be stationary. A technique to make a time series stationary is differencing. Here a new series is computed by taking the difference between each observation and it's successor. This removes trends from the series. An ARMA(p, q) process that differences the time series d times before fitting the model, is called ARIMA(p, d, q). More details and insights to ARIMA models are covered in [9]. ARIMA models are very popular and successful models for time series forecasting [66, 76].

3.1.3 Generalized Linear Models

For this and the following sections we will use the formulation of time series forecasting as a regression task. Like described in subsection 2.3.2, a time series can be embedded to transform the forecasting into a regression task. This results in a dataset (X, Y) with $X = (x_1, \cdots, x_n)$ and $Y = (y_1, \cdots, y_n)$. Each x_i is a vector of K steps of the time series, created by the time series embedding.

A simple regression model that is often used in machine learning is linear regression. Predictions are made using a linear function $\hat{y} = wx + b$. Fitting the parameters w is done by ordinary least squares, which solves

$$\min_w \|Xw - Y\|_2^2. \tag{3.11}$$

Regularization is often used to control the size of the coefficients. The norm of the parameter vector, $\|w\|_2 = \sqrt{\sum_{i=1}^{n} w_i^2}$ for rigde regression and $\|w\|_1 = \sum_{i=1}^{n} |w_i|$ for lasso regression, is added to the objective function of the optimization problem. An extra parameter λ controls impact of the penalizing term. Combinations of both regularizations are possible. One method using both norms is elastic net regularization.

Linear regression models have structural limitations, because they are always predicting a linear combination of the given features. Two extensions that introduce nonlinearities will be discussed in the following.

Projection Pursuit Regression (PPR)[43] is a model that consists of non-linear transformations of linear combinations of features. The prediction is calculated by

$$\hat{y}_i = \beta_0 \sum_{j=1}^{k} f_j(\beta_j^T x_i), \quad c_i \in \mathbb{R} \tag{3.12}$$

where x_i is the feature vector of an example and $\{f_i\}$ are smoothing functions from $\mathbb{R} \to \mathbb{R}$. The number of terms r is a hyperparamater of the model. The model is fitted using an alternating optimization approach that fits pairs of (f_j, β_j) at each iteration to minimize the mean squared error over the training examples.

Multivariate adaptive regression splines(MARS)[42] can be seen as an extension to linear models that try to capture interactions between variables and nonlinearities. MARS predicts by using the following equation:

$$\hat{y} = \sum_{i=1}^{k} c_i B_i(x), \quad c_i \in \mathbb{R} \tag{3.13}$$

$B_i(x)$ are called basis functions that can be a constant 1 to add an intercept to the model, a hinge function of form $\max(0, x_i - \text{const})$, $\max(0, \text{const} - x_i)$ or a product of two or more hinge functions. The prediction model is build in a forward pass, which iteratively adds basis functions to the model using a greedy search based on the resulting residual error of an addition. Each time a hinge function is added, its mirror is added as well, so that the range of feature x_i is split in two parts and treated differently. The forward pass ends when a max number of terms k is reached or the change in residual error is too small. A backward pass is needed for the model to reduce the overfitting. In this pass the terms get pruned.

3.1.4 Support Vector Regression

Support vector regression[35] is a regression method based on the concept of support vectors, which were introduced for support vector machines for classifcation.

The goal is to find a function $f(x)$ that has at most ϵ deviation from the targets y_i. This function in the linear case is of the form

$$f(x) = \langle w, x \rangle + b \quad b \in \mathbb{R}, w \in \mathbb{R}^d. \tag{3.14}$$

$\langle w, x \rangle$ notes the dot product of w and x. The deviation of ϵ can be visualized as a boundary on both sides of a hyperplane. All training points should lie within the boundaries. We also want to have a *flat* function, which means that a small w is wanted. This is achieved by minimizing the norm of the vector $\|w\|^2$.

With the introduction of slack variables ξ_i, ξ_i^*, this is the constrained optimization problem that is solved to get the model parameters:

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \tag{3.15}$$

$$\text{subject to} \quad \begin{cases} y_i - \langle w, x_i \rangle - b & \leq \epsilon + \xi_i \\ \langle w, x_i \rangle + b - y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases} \tag{3.16}$$

The slack variables ξ_i, ξ_i^* can be interpreted as helpers for outlier targets. The trade off between minimizing the norm of the parameter vector and using few of the slack variables is controlled by $C > 0$. Small values of C push the function to generalize, while a high value of C allows the function to be tightly fitted to all data points.

One of the main factors for the success of support vector methods, is the usage of kernels. Kernel functions allow to implicitly map examples to another feature space. This can be used to replace the dot product $\langle w, x \rangle$ with a kernel function $K(w, x)$ and calculate the *flat* function in the feature space rather than the input space. Using kernel functions allows the model to be nonlinear and have more predictive power than the linear version. Some common kernel functions are

$$K(w, x) = \begin{cases} \langle w, x \rangle & \text{linear} \\ (\gamma \langle w, x \rangle + r)^d & \text{polynomial} \\ \exp(-\gamma\|x_i - w\|^2) & \text{rbf} \end{cases} \tag{3.17}$$

The kernel function itself and the parameter γ are hyperparameters of the SVR. Support vector regression has been successfully used for time series forecasting [20, 38].

3.1.5 Neural Networks

Neural networks are machine learning models that consist of layers of so called neurons. The first layer of a neural network represents the input, which is given in form of a vector. The last layer is the output layer that can consist of a single or multiple neurons. In case of regression tasks a single neuron is used. The rest of the layers are so called hidden layers. The neurons in each layer are connected with neurons in other layers. In Figure 3.1, we see an example of a neural network, where each neuron is connected with all neurons of the next layer. A layer that is connected this way is a fully-connected layer. A neural network that uses only fully-connected layers is called a multilayer perceptron (MLP). The parameters of an MLP are the number of hidden layers and the number of neurons in the layers.

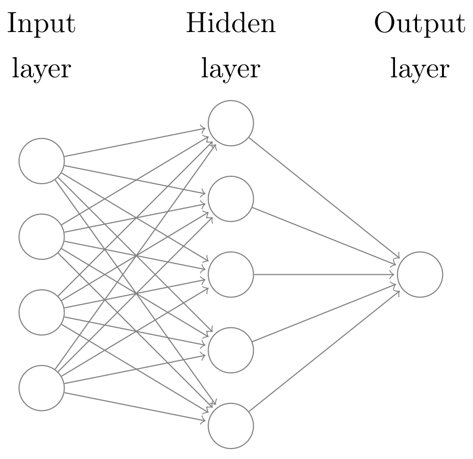

Figure 3.1: An MLP with one hidden layer, consisting of 5 neurons.

Each neuron computes an output y, using an activation function φ, a set of inputs x_i, weights w_i and a bias b:

$$y = \varphi(\sum_{i=1}^{n} w_i x_i + b) \tag{3.18}$$

The input values are the values computed by the connected neurons in previous layers. The bias and weights are the parameters of the model. Without the activation function, a neural network would compute a series of linear regression. The activation function, which is normally a nonlinear function like sigmoid, introduces nonlinearities to neural networks. The parameters of the network are learned by a gradient descent procedure, which uses the gradient of the error of the neural network with respect to the weights of each neuron for optimization. Further insights into neural networks can be found in [49, 91]. MLPs have been successfully used for time series prediction[88].

3.1.6 Boosting & Bagging

A main idea of ensembles was to make a strong model by using multiple models that are limited in their predictive power. This idea is the foundation for Bagging [11] and Gradient Boosting Algorithms[41]. Both algorithms are examples for ensemble learning with models using the same base-line model. Decision trees are often used as such weak models. They are prone to overfitting and sensitive to small changes in the input data. Popular methods of boosting and bagging use decision tree ensembles.

Random Forest[12] consist of multiple decision trees that are trained using subsets of the dataset. The main idea is to have a set of decision trees that are specialized on different patterns of the data and make errors on different examples. Each decision tree is trained on its own subset and each subset is drawn with replacement from the dataset. To introduce even more diversity in the predictions of the decision trees, they use a random subset of all features for each split they make. Since decision trees can be used to predict either discrete or continuous values, they can be used for classification and regression tasks. This holds for random forests as well. To make the final prediction the predictions of all trees of the model are averaged.

Where the idea of bagging is to generalize the prediction and reduce the variance of a model, boosting tries to iteratively improve the prediction. Given a training set (X, Y) with $X = (x_1, \cdots, x_n)$ and $Y = (y_1, \cdots, y_n)$ and a loss function $L(y, F(x)))$, gradient boosting computes a series of models, that are added up to get the final prediction.

$$F(x) = \sum_{m=1}^{M} \gamma_m f_m(x) \tag{3.19}$$

M is the number of iterations and each model f_m is trained on a different set. The first model is trained on the training set. With each iteration the prediction uses more models to predict. The loss of the full model is now reduced using an iterative procedure that performs gradient descent. Each new model is used to make a gradient step on the loss of the full model and the coefficient γ_m is the step size. To really optimize the models loss in this way, the targets that are used to train a new model are the negative gradient of the current model, with respect to the model.

$$g_{im}(x_i) = - \left[\frac{\partial L(y_i, F(x_i)))}{\partial F(x_i)} \right]_{F(x)=F_{m-1}(x)} \tag{3.20}$$

Here $F_{m-1}(x)$ is the model consisting of $m-1$ already fitted models. The next model f_m is then fitted using a dataset of $\{(x_i, g_{im}(x_i))\}$. Also the step size for the model is calculated.

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{N} L(y_i, F_{m-1}(xi) + \gamma f_m(x_i)) \tag{3.21}$$

Both, step size and model, are added to the prediction of $F(x)$. Typical loss functions for regression tasks are the least squares $(y - F(x))^2$ and least absolute deviation $|y - F(x)|$. This method can be used for different base models, but the most common version uses decision trees.

3.2 Ensemble Learning

Ensemble learning refers to methods, that generate a pool of models and combines them to make predictions. There are many examples for the success of ensembles [12, 33, 40].

The field of ensemble learning offers many approaches that lead to an algorithm making predictions. The main separator between ensemble algorithms is the type of models they use. A Random Forest for example uses just one type of model and builds a powerful ensemble by learning variants of the same model. This type of ensembles use a homogeneous model pool. Two very well-known methods with homogeneous model pools have been presented in subsection 3.1.6. Instead of just using one type of model, it is also possible to use multiple types and combine them. This is a heterogeneous model pool and it has an advantage in terms of diversity on homogeneous model pools. As discussed in subsection 3.3.2, the diversity of the ensemble has an important role for its performance. The other important part is accuracy, which is why we try to use models that are proven for time series forecasting. For the rest of this thesis we focus on ensembles constructed from a heterogeneous model pool to use the inherent diversity that comes with different models.

Using and combining multiple models has been discussed as a three step system, called Multiple Classifier System [14]. The first step is generating a model pool. Choosing between a homogeneous and a heterogeneous pool affects the next steps. The second step is selecting models from the pool. One of the critical choices here is, whether a single or multiple models are selected. By selecting a single model the third step of combining the models is redundant. If multiple models are selected then combining them to an ensemble is a task with multiple approaches and often discussed in the research community.

We have seen that there is a variety of different models that are suitable for time series forecasting. A normal approach to any supervised learning task is to compare the performance of different models and select the best. It is also possible to build a heterogeneous model pool of these models. The main questions with this approach are how to select members for the ensemble and how to combine them. The selection step is often solved by using an arbitrary fraction of the model pool that is selected by some performance measure. Chapter 4 shows a more sophisticated selection approach.

Combining different models to an ensemble is compared to selection rarely discussed. Simple methods often work very well. The research is often focused on generating the model pool and selecting the models. We will now present some of these methods for combining predictions. Sometimes the boundaries between selection and combination can be fluent, because weighting an ensemble member with 0 equals a member selection. There are three common combination approaches: voting, e.g. bagging [12], cascading [47] and stacking [93].

3.2.1 Voting

Combining multiple experts with a voting approach, means computing a weighted average of the base models predictions. Classification ensembles often use majority voting, which weights every vote evenly and selects the prediction with the most votes. Regression ensembles need their own combining methods and an equivalent approach to majority voting is to use the average of all ensemble members predictions.

$$\hat{y}_{ens} = \frac{1}{M} \sum_{i}^{M} \hat{y}_i \tag{3.22}$$

The ensemble consists of M models and $\hat{y}_i$ is the prediction of model i for the current step. This approach has been shown to be a robust combination method for forecasting [27, 68, 86]. In general, any convex combination of the ensemble members predictions is a voting combination method.

$$\hat{y}_{ens} = \sum_{i}^{M} c_i \hat{y}_i, \quad c_i \geq 0, \sum_{i}^{M} c_i = 1 \tag{3.23}$$

Making the weights convex is important, because otherwise the combination would add an bias to the ensemble. Generating the weights for the combination can be done in various ways. Forecasting ensembles deal with the special characteristics of time series and have their own combination methods. Sliding window methods have been very successful for combining forecasting predictions.

A sliding window of a time series is a window of certain size h, that is moved along the time series in steps of size s. When starting at index 1, the first window is a sequence of length h and contains the values of the time series from 1 to h. The next window is moved by a step of s and contains now the values from $1 + s$ to $h + s$. We have already seen a sliding window approach, when we converted the forecasting task to a regression task (Definition 2.3.2). The used time series embedding is a sliding window with size K and step size 1.

For the combination of predictions, the sliding window we look at are the predictions made in the past h steps. It has been shown that the performance of forecasting models changes over time [1]. To use the performance of recent predictions for weighting the ensemble members seems intuitive. A sliding window ensemble [66] uses the errors of the recent predictions as weights for combination. We note the error made by model i in the window of the last h steps as e_{ih} and assume that the error is measured in $[0, 1]$. Then the prediction of the sliding window ensemble is:

$$\hat{y}_{ens} = \sum_{i}^{M} \frac{\hat{y}_i (1 - e_{ih})}{N}, \quad N = \sum_{i}^{M} (1 - e_{ih}) \tag{3.24}$$

One method that uses past performance to generate combination weights is AEC [77]. It uses an exponential re-weighting strategy to generate convex weights and includes a forgetting factor to give more importance to recent values. Another method was proposed by Timmermann [85], which originated from the domain of stock-prediction. In this domain the models have only a short period, in which their predictions are suitable. The method uses the R^2 measure of explained variance for weighting and rejects the ensemble members predictions, if their R^2 measure is too low.

We have already presented Random Forests as an ensemble method using voting in subsection 3.1.6.

3.2.2 Cascading

Cascading [47] is a method that is rarely used, because it trains several models in sequential fashion, which is very time-consuming. The idea of this method is to combine the outputs of the base models, one at a time, with the feature set. This way the outputs are used again as features in a special iterative way.

3.2.3 Stacking

The concept of stacking [93] is widely used for combining any kind of predictive models. It is a method that uses a model to combine the predictions of the base models. Stacking approaches can be further separated into two types: parametric and non-parametric. Parametric approaches assume a functional form for the combination. The parameters of this function are estimated or learned. Such an approach can be a linear regression that combines the predictions of the base models [13]. We have already presented Gradient Boosting Machines as an ensemble method using stacking in subsection 3.1.6. Non-parametric approaches can be seen as metalearning approaches, which try to model the learning process of learning algorithms [10]. Typical approaches use for example decision trees[56, 87] to select and combine members for an ensemble prediction. Several metalearning methods have been used for time series forecasting to improve the combination and selection of models [28, 72, 74, 63].

A rather new method for the combination of forecasting models is arbitrating [23], which was originally introduced for the combination of classifiers [69]. In this case, the metalearning is used to estimate the future performance of the candidate models. The estimated performances are then normalized and used as weights for the combination of the ensemble members. For each candidate model a meta-model is learned. This meta-model can learn the candidate's prediction behavior, because it gets the features and the candidate's loss for each example. The meta-models estimate the performance for each candidate. With these estimations the ensemble member are weighted.

3.3 Ensemble Theory

There is a long history in machine learning theory of trying to explain and understand the theoretical foundations of learning. Especially the complexity of models and the tuning of the model's parameters to data are well studied problems. It has been shown that ensemble approaches can theoretically and empirically outperform single predictors. First we need to understand the learning problem and the error decomposition for a single model. Then we can extend this to ensembles.

3.3.1 Bias-Variance Decomposition

Given a data set of input vectors and output scalars $z = \{(x_1, t_1), (x_2, t_2), \ldots (x_N, t_N)\}$, where each element is drawn from a random variable Z with an unknown distribution $p(x, t)$, the learning problem in machine learning is often defined as approximating the mapping between input and output using the given data z. The mapping is a function f, with a set of parameters w. We need to find the right parameter set, to minimize the expected mean squared error,

$$e(f) = \int (f(x; w) - t)^2 p(x, t)\ d(x, t). \tag{3.25}$$

Since there is no access to the full distribution $p(x, t)$, we have to approximate by using the given data set z.

$$e(f) \approx \frac{1}{N} \sum_{n=1}^{N} (f(x_n; w) - t_n)^2 \quad , \quad (x_n, t_n) \in z. \tag{3.26}$$

Common problems in fitting the parameters are *overfitting* and *underfitting*. If the parameters are exactly tuned on z, maybe even reaching an error of zero on z, we overfit the model and might not perform well on future data. This is because z is just a sample of the true distribution. If the parameters are not fitted enough on the data, they might as well have problems to predict future data, because the characteristics of the distribution are not well enough reflected in the parameters of the estimator. This second case is called underfitting.

Geman et al. [48] showed a decomposition of the error into *bias* and *variance*. Over- and underfitting are tightly bound to bias and variance. With replacing the integral notation of Equation 3.25 by $E\{\cdot\}$ and shortening $f(x_n; w)$ to f we get the bias-variance decomposition as

$$E\left\{(t - f)^2\right\} = (E\{f\} - t)^2 + E(f - E\{f\})^2 \tag{3.27}$$

$$= bias(f)^2 + variance(f) \tag{3.28}$$

Bias and variance of a model are tightly bound to each other. Reducing the variance leads to a higher bias and vice versa. Typical for overfitting is a low bias and a high variance. Underfitting is exactly the opposite: A high bias but a low variance. To avoid both over- and underfitting, bias and variance have to be balanced against each other. This leads to the best performance.

3.3.2 Bias-Variance Decomposition for Ensembles

For this thesis we are not only looking for a single model f, but at an ensemble of multiple estimators $f_1, \dots f_M$, where each f_i has its own set of parameters w_i. Each estimator is trained according to Equation 3.26 and then the predictions are combined to get the ensemble prediction. There are many ways to combine the estimator predictions but for now we assume that each estimator gets a weight c_i with $\sum_i^M c_i = 1$ and $c_i \geq 0$. This makes the ensemble prediction a convex combination of the estimator predictions:

$$f_{ens}(x; w_1, \dots, w_M) = \sum_i^M c_i f_i(x; w_i) \tag{3.29}$$

This ensemble can be analyzed using the same bias-variance decomposition as in Equation 3.27. Further investigations of the error decomposition for regression ensembles provided more insights in the dynamics between the different estimators that build the ensemble. A very promising result was the decomposition provided by Krogh and Vedelsby[59]:

$$(f_{ens} - t)^2 = \sum_i^M c_i (f_i - t)^2 - \sum_i^M c_i (f_i - f_{ens})^2 \tag{3.30}$$

The first term, $\sum_i^M c_i(f_i - t)^2$, is the weighted average error of the individual models, which tells us that the error of the ensemble depends on the errors of the individual models. This is completely expected. The second term, $\sum_i^M c_i(f_i - f_{ens})^2$, which is named the *Ambiguity*, measures the variability between the members of the ensemble. Since the Ambiguity is strictly positive, the decomposition shows, *"that the generalization error of the ensemble is always smaller than the (weighted) average of the ensemble errors"*[59]. The first thought that comes to mind is to maximize the Ambiguity to reduce the model error, but as we will see this is not possible without raising the weighted average error.

Another decomposition was provided by Ueda and Nakano [89]. For their decomposition a uniformly weighted ensemble is assumed. This means that $\forall i.\ c_i = \frac{1}{M}$. They decomposed the error in three parts: Bias, Variance and Covariance.

$$E\left\{(f_{ens} - t)^2\right\} = \overline{bias}^2 + \frac{1}{M}\overline{var} + (1 - \frac{1}{M})\overline{covar} \tag{3.31}$$

The three new concepts here are the averaged bias of the ensemble members,

$$\overline{bias} = \frac{1}{M} \sum_i (E\{f_i\} - t) \tag{3.32}$$

the averaged variance of the ensemble members,

$$\overline{var} = \frac{1}{M} \sum_i E\left\{(f_i - E\{f_i\})^2\right\} \tag{3.33}$$

and the averaged covariance of the ensemble members,

$$\overline{cover} = \frac{1}{M(M-1)} \sum_i \sum_{j \neq i} E\left\{(f_i - E\{f_i\})(f_j - E\{f_j\})\right\} \tag{3.34}$$

This decomposition tells us more about the importance of the interaction between the models in an ensemble. We have seen that for single models the variance is an important factor and now we see that for ensemble the covariance between the ensemble members plays an important role as well. This theory motivates a special focus on the covariance between the ensemble members.

Brown [16, 18] showed that both ensemble error decompositions can be reduced to the Bias-Variance decomposition. It is also possible to examine, which part of the bias-variance-covariance decomposition corresponds to the ambiguity term. In this process Brown stated the important fact that both new decompositions do not show ways of decomposing the error, so that one of the parts could be optimized without the others. In both decompositions it is a tradeoff: A two-way tradeoff between bias and ambiguity in the first and a three-way tradeoff between bias, variance and covariance in the second decomposition.

It is a necessary and sufficient condition for an ensemble of classifiers to be more accurate than any of its individual members that the classifiers are accurate and diverse[50]. The three-way tradeoff is a theoretical justification for this statement for regression ensembles. The accuracy of classification ensembles from the statement is the mean squared error for regression ensembles. Measuring diversity is often discussed for classification tasks [61] and for regression tasks as well [37, 19]. Both, diversity and accuracy, play an important role in the ensemble approach presented in this thesis.

Chapter 4

Dynamic Model Selection

Selecting the right model for machine learning is important. At first a set of models are trained and tested. Different models with different parameter sets can create a big number of possible models. Intuition and experience can lead to good results but without a structured approach, selecting the right models from the pool of models can become very time-consuming. If we want to select an ensemble, the number of possible solution grows exponentially with the size of the ensemble. This, and the interactions between the members of an ensemble that have been discussed in subsection 3.3.2, make selecting the best set of models for an ensemble even more complicated and computationally expensive.

The common approach to model selection is the static model selection, where the step of selecting happens ones before applying the model. An important part of this static selection task is to evaluate all trained models. This is often done using a cross-validation scheme [2]. A cross-validation scheme can be used as a good estimator of a model's performance on unseen data. The selection step then uses the estimated performances and chooses among the models with the best performances. A simple extension for ensembles is to pick the n-best performing models for the ensemble. An ensemble of the n-best performing models might produce accurate predictions. But as we already discussed it is also important that an ensemble consists of diverse members. The number of possible ensembles grows exponentially with the number of available candidates. There exist more complex search algorithms for ensemble selection, that try to handle the number of combinations by using for example greedy search[70, 34] or evolutionary approaches[94, 78, 46].

A different and more complex approach to model selection is dynamic model selection. As the name suggest the predicting model is dynamically selected for every new example. The same idea can be transferred to ensembles, where the composition of the ensemble is dynamic and the members get selected for every new example. This approach originated in the selection of classifiers. A wide range of concepts and methods have been presented for the selection of classifiers. Although some of the combination approaches mentioned in subsection 3.2.1 and subsection 3.2.3 implicitly perform a selection, there have been no results with explicit focus on dynamic model selection for regression tasks, let alone forecasting.

Common to all approaches for dynamic selection of classifiers is that they are solutions to selection; the second step of the Multiple Classifier System. To select from the model pool, we need to gather information about the models in the pool. The most important information is the performance of the models. Static selection schemes estimate the performance using validation methods like cross-validation. These methods aim to estimate the models performance on unseen data for the full distribution and find the model or ensemble that is expected to perform best on any data of the distribution. Dynamic Model Selection has a more specific approach and tries to find the model or ensemble that is expected to perform best on the current example x_i. To make the selection specifically tailored for the given example, we have to use the information given by this example. Therefore, we try to get an estimate of how each model would perform on data that is like the current example. With these estimates we can than make a selection.

Cruz et al.[29] present a taxonomy(see Figure 4.1) of the steps and structure of dynamic selection, naming various solutions for the different steps.

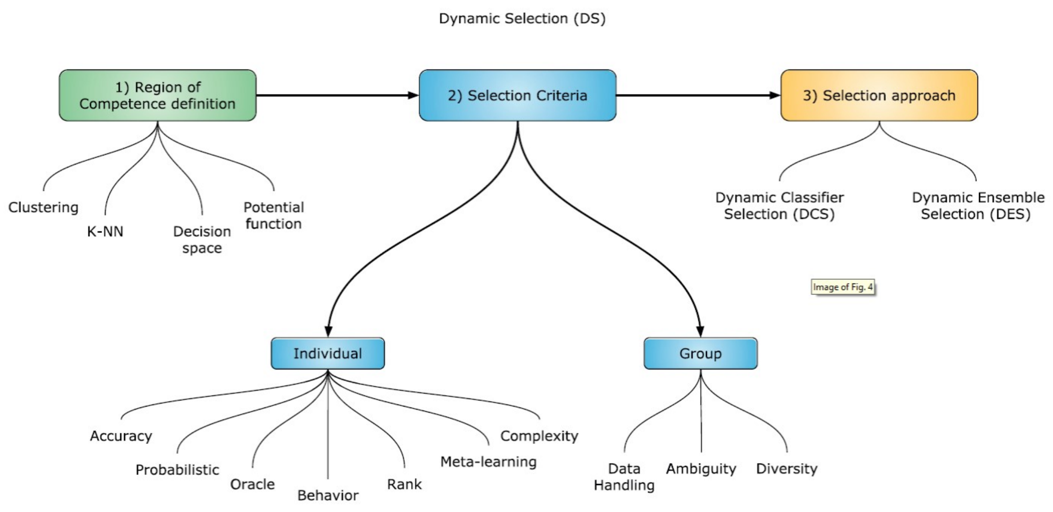

Figure 4.1: Taxonomy of Dynamic Selection[29]

The taxonomy shows three different steps of dynamic selection, of which the selection approach is more of an overall choice then a real part of a typical dynamic selection algorithm. The region of competence and the selection criteria are the main parts of dynamic selection algorithms. First we look at the region of competence, which explains how we get an estimate for each models performance on the current example. Afterwards we look at possible selection criteria that make use of the local regions.

4.1 Region of Competence

As already stated, the dynamic model selection tries to make a specific selection for each example x_i. The selection gets more specific if we estimate each models performance for the current example x_i. Since we do not have the information about the true target value belonging to x_i, we need to base our estimate on similar examples. To have similar examples the method has to use some validation data, that is hold out during the training of the models. We call this holdout data the dynamic selection set(DSEL). We measure similarity as a distance function. Therefore, examples that have a small distance to x_i are similar. We call x_i and all similar examples from DSEL a local region. Since we define the region of x_i based on distance measurements in the feature space, we speak of a local region of the feature space. For each example from the DSEL, we measure the performance for the candidate models. Static selection approaches would use the information from all examples from the dynamic selection set and select based on the average performance. In dynamic selection we only use a subset of the DSEL, the local region of the current example. The performance of each candidate on the examples from the local region gives us the estimate. Because we estimate the competence level for each learner based on the local region, these regions are called region of competence.

There are many ways to define local regions. An overview of methods is presented in [29], here we present K-NN and Clustering.

4.1.1 Clustering

Clustering techniques provide a natural definition of regions. Each cluster is a region in the feature space. The local region belonging to a new example x' is the cluster that x' gets assigned to. An important characteristic of local regions based on clustering is that they are not dependent on the new example x'. The feature space is partitioned with the clustering and the local region subsets itself are static.

4.1.1 Definition (Local Region Clustering). Given a clustering function C that assigns a cluster to a given example, the local region for a new example x_i is

$$N_{x'} = \{x_j \in \text{DSEL} \mid C(x') = C(x_j)\} \tag{4.1}$$

To compute the local region of x', we only need to find the cluster the example belongs to. Because the local regions are static, we can compute the selection that is made based on the local region during the training phase of the algorithm. This is a big advantage in terms of required computation, because all rankings and selections are computed during the training phase. The algorithm has to compute only distances between the new sample and each cluster center, rather than the full DSEL.

Because the local regions are static, clustering is limited to a rather small number of different selections. At most, each cluster has a different selection. Therefore, the selections are not as specific as with the next method we present.

4.1.2 K-Nearest-Neighbors

K-Nearest-Neighbors (K-NN) computes a neighborhood of examples that are closest to the given test sample x'. The examples in the neighborhood are computed based on the distance between each example in the DSEL and x'. The local region for x' is the neighborhood of x' defined by K-NN.

4.1.2 Definition (Local Region K-NN). Given a distance function d and a new example x', the examples $x_j \in \text{DSEL}$ are sorted by their distance $d(x', x_j)$. This results in a set $\text{DSEL}_d = \{x_{D_1}, \cdots, x_{D_n}\}$ with $d(x', x_{D_i}) \leq d(x', x_{D_j})\ \forall j > i$. The neighborhood of x' is then

$$N_{x'} = \{x_i \in B_d | \forall i \in \{1, \dots, k\}\} \tag{4.2}$$

with $|N_{x'}| = k$.

In contrast to clustering techniques, the local regions when using K-Nearest-Neighbors are dependent on the given test sample x'. The distance between each example in the DSEL and the test sample x' is computed. The examples of the local region are then used to estimate the performance of each candidate model. Therefore, the ranking and classifier selection has to be computed for each new sample during the test phase. This procedure produces many configurations for the selection and is much more adaptable to small changes in the structure of the data.

The performance of K-NN depends mostly on choosing a good distance measure. The most common used metric for K-NN is the euclidean distance, which can be used for many different types of data. A more specialized distance metric for time series is Dynamic Time Warping. We will present an adaption of K-NN with dynamic time warping in subsection 5.1.3.

4.2 Selection Criteria

As we already saw in Figure 4.1 the selection criterion is an important part of dynamic selection and there are different criteria to use. They all have in common that they use the available information, the current example, the local region, and the candidate models, to estimate each models competence. The criteria shown in the figure are all for classifications tasks and transfering them to regression tasks is not always possible. The main difference between regression and classification tasks in terms of evaluation is the category of the target. Classification tasks have a categorical target variable, where regression tasks have a real target variable. It is always possible to use the error metric of the task as selection criteria. The taxonomy lists accuracy for the classification task. We already discussed the mean squared error and mean absolute error for regression tasks. Both can be used instead of accuracy as selection criterion for regression tasks. Selecting by the calculated errors is very intuitive and close to the normal model selection process. Some selection criteria presented in [29] measure the performance of a candidate model per class. This is not possible for a regression task. Therefore, it is not possible to select models that have a high performance at least at one class.

Another group of selection criteria looks directly at a group of ensemble candidates. As discussed in subsection 3.3.2 it is important that the members of an ensemble are accurate and diverse. To make sure that the members' results are accurate we can just look at the individuals, but for making sure that the members of an ensemble are diverse, we have to look at a group of candidates and estimate their diversity. Calculating diversity for different groups of candidates can amount in a big number of computations and is it not always practical for selection.

4.3 Oracle

One theoretical concept for dynamic model selection is the Oracle. It was defined by Kuncheva[60] and is an abstract model that has information about the true label. This does not mean that it always predicts the true label and therefore makes no error, but the Oracle chooses the best predicting model from the model pool. Since the Oracle has information about the true label, it can select the best predicting model and does make an estimated selection.

For classification tasks the Oracle chooses the correct label if there is a model in the pool predicting the correct label. Since there was no definition of the Oracle for regression tasks, we defined it as using the prediction with the smallest absolute difference to the true label.

The Oracle is not a practical machine learning model, but can be seen as an upper limit for dynamic selection techniques. It is possible that in some rare cases a weighted ensemble can perform better on a regression task then the oracle, but overall it provides a useful comparison and upper limit.

Chapter 5

DES Forecasting

The goal of this thesis is to develop a forecasting algorithm that uses a heterogeneous model pool with dynamic selection. The two main motivations behind this approach are, first, to make the best use of all the different models that exist for time series forecasting and, second, being adaptable to changes in the time series. Both motivations originate from the fact that the performance of forecasting models changes over time[1] and across the input space[17]. We also mentioned that models have different assumptions regarding linear or nonlinear dependencies between the features and the target values. So two models with different assumptions can be both useful, if the time series behaves linear in some regions and nonlinear in others. Therefore, we have to recognize and understand the different regions in the time series and the model's for each region, to adjust the selected models accordingly.

To get an understanding for a model's performance on a given example, we try to compare the patterns of the example to other patterns we have seen during the training phase. The performance of each model on similar examples from the training phase is evaluated and used as an estimate for the performance on the new example. With the estimates we can now select the best members for an ensemble.

The first versions of the algorithm only included simple selection procedures and weighted all selected models equally. The selection used clustering and nearest neighbor local regions. The nearest neighbor approach outperformed the clustering approach already in small experiments. Without weighting models individually, simple methods like the sliding window ensemble could outperform dynamic selection. Therefore, we added additional weighting for the ensemble members. Including models for their diversity in contrast to other models resulted in worse results and was therefore ignored.

5.1 Training

There are two parts in training the Dynamic Ensemble Selection Forecasting (DES-Forecasting) algorithm. First we need to train all the candidate models to get a model pool. During training, we want to gather as much information about the performance of each candidate. After training the model, all the acquired information and the training data is used to learn a region of competence model, that provides us with performance estimates on local regions for each candidate model.

5.1.1 Generating the model pool

For the model pool, we need to train all the candidate models. Our algorithms can use any set of candidate models that is suitable for time series forecasting. The candidate models should use different methods, assumptions and parameters to be suitable for different situations. We need to hold back some data, to gain a dynamic selection set that consists of validation data, where each model's predictions is evaluated against the real targets. In domains other than time series, the samples of the data are independent of each other. A cross validation procedure could be used to gain out-of-bag predictions for every sample in the training set. Time series data is not independent. A cross validation procedure might work. But the main assumption for such a procedure is that each sample from the data has to be independent and identically distributed. This assumption is violated for time series data. Therefore, we can not use a cross validation procedure. A simple solution to this problem is to split the data into a train and a holdout set, with respect to the time series index. The holdout set is then a fraction of the time series directly following the train set. Data that is used for training cannot be part of the holdout set and vice versa. This seems problematic because we would like both sets to be as big as possible. If we many examples for training the candidates, we can expect their individual performance to increase. If we have a big holdout set, then we learn more about the performance of the candidate models on new data.

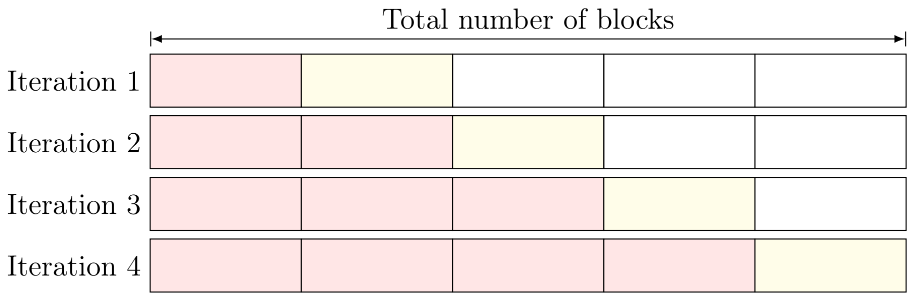

Figure 5.1: Blocked prequential procedure. The data is split into continuous blocks. Red blocks are used for training in the iteration and the yellow block is evaluated. Four out of five blocks of all the data can be evaluated as out-of-bag samples this way.

To make the most out of our limited training data, we would like to maximize both set sizes. This can be achieved by using a blocked prequential procedure [30]. The procedure is visualized in Figure 5.1. The available data gets split into b continuous blocks of the time series. Then the training and evaluation starts with the first two blocks. The first gets used for training and the second is evaluated. This way the sets are still in the correct order of time, and we are testing the candidates' performance on unseen data. All examples of the evaluation block are out-of-bag samples and therefore useful validation data. The next iteration uses the first two blocks as training set and the third block for evaluation. This process is repeated until all the blocks are used. The only block not evaluated is the first one, which is a small cost for keeping up the correct order and dependencies of the data. After iterating through all blocks, we can train the models on the full dataset and use them later for building the ensembles. Therefore, they have used all the training data and the estimated performances are still valid, because they were gained on holdout data.

The out-of-bag samples of the training phase, whether produced by a holdout set or the blocked prequential procedure are saved in the model as B. All candidates predictions and the respective errors on the examples from B are collected. For each candidate i there is a set of predictions $\hat{Y}_B^i = \left\{\hat{y}_b^i\right\}, \forall b \in B$ and the respective errors $E_B^i = \left\{e_b^i\right\}, \forall b \in B$. We implement an option to use a blocked prequential procedure in the training of DES-Forecasting. We will study the advantages and disadvantages in section 6.3.

5.1.2 Region Of Competence Model

As described in section 4.1 a main approach for dynamic selection is to calculate local regions surrounding an example. If we know the region of an example, we can compute the error scores in the region for each model. This error scores indicate if a model is suitable for predicting models from the current local region. For the prediction, the best models are selected based on their performance in the local region of current example.

For the DES Forecasting Algorithm we use a K-Nearest Neighbour algorithm, to compute the local regions. This algorithm has a higher cost in computing the estimates, because the local region depends on the given example and cannot be computed during training. Compared to a clustering algorithm, the K-NN can provide more specialized ensemble combinations. A clustering algorithm would generate at most a different combination for each cluster, where the K-NN might produce different combinations for each example. If we have a lot of combinations, the combinations can be specialized to more specific patterns of the time series. To make the K-NN more specific for time series, we use a specific time series distance metric: dynamic time warping.

5.1.3 K-NN with Dynamic Time Warping

Nearest Neighbor computations are highly dependent on the chosen distance metric. To make K-NN specific for time series, we will use dynamic time warping[67] as our distance metric. Dynamic time warping tries to align two time series in an optimal way, so that a pattern that may be shifted in position in one of the time series, becomes aligned with the other time series. Finding the optimal alignment between two series of length n is solved using dynamic programming and has a complexity of $O(n^2)$. Given the alignment, the distance is calculated as the squared distance between the aligned points of each series. The cheaper alternative to dynamic time warping is the euclidean distance, which has a complexity of $O(n)$ and aligns two points with the same index of each series.

5.1.1 Definition (Euclidean distance). The euclidean distance between two time series Q, C of equal length n is

$$d_{ed}(Q, C) = \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2} \tag{5.1}$$

A comparison of euclidean distance and dynamic time warping is shown on the left side in Figure 5.2. The alignments between the two series are shown as the connections between the series.

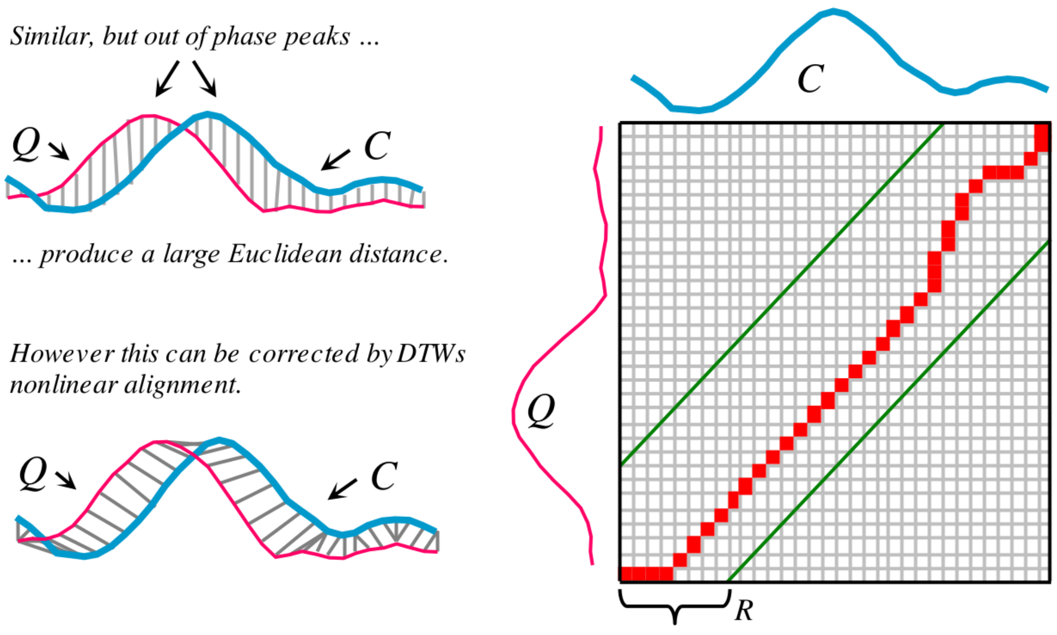
Similar, but out of phase peaks ...
Q
C
... produce a large Euclidean distance.
However this can be corrected by DTWs nonlinear alignment.
Q
C
C
Q
R

Figure 5.2: The left side shows two time series Q and C that have similar patterns. The peaks and the general patterns in the two series seem shifted. The euclidean distance between Q and C is large, while the DTW alignment produces a lower distance. The right side shows the warping path between the two series on the distance matrix. The bounds with distance R are the Sakoe-Chiba-Band, that is used to constrain the warping path. Graphic from [73]

An alignment between two series is a list of pairs $[p_1, \ldots, p_L]$ that matches indices of one time series Q to another time series C. We assume that both time series are of the same length n. Then each pair $p_i \in [1, n] \times [1, n]$. Dynamic time warping can also be used for time series of different lengths, but we only relevant case for us is equal length. The alignment is also called a warping path P and there are some constraints for the possible combinations in each pair.

5.1.2 Definition (Warping path [67]). Given two time series Q and C with equal length n, a warping path $P_{Q,C}$ is a list of pairs of indices $[p_1, \ldots, p_L]$ with $p_i = (a_i, b_i) \in [1, n] \times [1, n]$ with three conditions:

- Boundary condition: $p_1 = (1, 1)$ and $p_n = (n, n)$
- Monotonicity condition: $a_1 \leq a_2 \leq \cdots \leq a_n$ and $b_1 \leq b_2 \leq \cdots \leq b_n$
- Step size condition: $p_{l+1} - p_l \in \{(1, 0), (0, 1), (1, 1)\}$ $\forall l \in [1, n-1]$

Such a warping path can have at most length $L \leq 2n$.

The optimal warping path can be computed with dynamic programming. The matrix on the right of Figure 5.2 shows the optimal warping path on the distance matrix of the two time series. All warping path start at the lower left corner and go to the upper right corner. The optimal warping path has the lowest cost on this way. The cost of a matrix cell (i, j) is $(q_i - c_j)^2$ and the cost of a path is the sum of all matrix cells it uses. Solving this dynamic programming problem has a complexity of $O(n^2)$ and makes dynamic time warping expensive. A simple constraint that reduces the computational cost is the Sakoe-Chiba-Band. A Sakoe-Chiba-Band of size R adds a constraint for the warping paths. The elements of the warping path are not allowed to be further away then R from the diagonal of the matrix. We see this constraint as well in the matrix on the right of Figure 5.2. The dynamic time warping distance is the distance between two time series using the warping path with minimal cost.

5.1.3 Definition (Dynamic time warping distance). Given two time series Q and C with equal length n, the dynamic time warping distance is

$$d_{dtw}(Q, C) = \sqrt{\min \left\{ \sum_{(i,j) \in P_{QC}}^{n} (q_i - c_j)^2 | P_{QC} \text{ is a warping path} \right\}} \tag{5.2}$$

The warping path with minimal cost is the optimal time warping path P^*_{QC}.

In comparison to the euclidean distance, dynamic time warping is computationally expensive. With n as length of the time series, the complexity of DTW is $O(n^2)$, which is significantly worse than euclidean distance with $O(n)$. There has been a lot of research revolving around dynamic time warping and its uses for time series classification, most importantly by Eamonn Keogh. Their state-of-the-art implementation for subsequence search with dynamic time warping is called the UCR Suite [73]. The implementation uses multiple optimizations that compute lower bounds of the dynamic time warping distance. If the lower bound value is already higher than the current smallest distance, there is no need to calculate the real dynamic time warping distance. A set of lower bounds with growing complexity is computed and only in the worst case the full dynamic time warping computation is needed. The computed subsequence match remains exact, while the computations are reduced immensely. These optimizations allow the extensive usage of similarity search under dynamic time warping. An existing python extension [57], that uses the UCR Suite implementation for subsequence search, was adapted to support searching for the k most similar subsequence matches. These k subsequence matches are the nearest neighbors of the given example. We will use this for the local region computation in DES-Forecasting.

5.2 Prediction

Predicting the next value of a time series with our method DES-Forecasting consists of multiple steps. The first step is selecting the ensemble members for the current prediction. This relies on the region of competence model.

The second step handles weighting the ensemble members. Making ensemble predictions for regression task is not as simple as for classification tasks. Where classification ensembles usually work well with a simple voting procedure, a regression ensemble using a simple mean of all models predictions does not perform so well.

The third step is optional and specifically for handling diversity within the ensemble. When the predictions of two models are very similar their use for the ensemble is not that high. Reducing one of the models weight can be useful in this case. If the predictions are very different, we can increase the weight.

To predict the value y_t, which is the value of the time series at point t, the algorithm has a set of data to use for this prediction:

- The feature set x_t. This set includes the last K values of the time series, where K is the dimension of the time series embedding. Also the algorithm can use additional pre-calculated features
- The last predictions $\hat{y}^i_{\{t-h,t-1\}}$ and the errors on these predictions $e^i_{\{t-h,t-1\}}$ for each candidate i
- The out-of-bag predictions $\hat{Y}^i_B = \left\{\hat{y}^i_b\right\}$ and the respective errors $E^i_B = \left\{e^i_b\right\}$ for each candidate i and each example $b \in B$.

5.2.1 Ensemble Member Selection

To select models for the ensemble we want to estimate how good their performance will be for the current example. Therefore, we use the information gathered during the training.

First, the local region of the current example is calculated. This uses the RoC-Model. The local region or in case of the used K-NN model the neighborhood of x_t are the closest k examples from the out-of-bag samples B. Here, closest is defined by the distance metric d used in the K-NN model. So we order the examples $x_i \in B$ by their distance $d(x_t, x_i)$. This results in a set $B_d = \{x_{b_1}, \cdots, x_{b_i}\}$ with $d(x_t, x_{b_i}) \leq d(x_t, x_{b_j})\ \forall j > i$. From the ordered set we select the first k elements. This yields a neighborhood N_{x_t}:

$$N_{x_t} = \{x_i \in B_d | \forall i \in \{1, \dots, k\}\} \tag{5.3}$$

with $|N_{x_t}| = k$. Accordingly, there are the predictions of $\hat{Y}^i_{N_{x_t}} \subset \hat{Y}^i_B$ and the errors for each prediction $E^i_{N_{x_t}} \subset E^i_B$.

For each candidate model we calculate an error estimate $\hat{e}^i_{x_t}$. For this, we use the average errors $\bar{e}^i_{N_{x_t}}$ of the neighborhood and the average error of the last predictions $\bar{e}^i_{\{t-h,t-1\}}$.

$$\hat{e}^i_{x_t} = \alpha \cdot \bar{e}^i_{N_{x_t}} + (1 - \alpha) \cdot \bar{e}^i_{\{t-h,t-1\}} \tag{5.4}$$

where α is a model parameter that balances the estimate between recent performance and local region performance. The default value is $\alpha = 0.1$. Intuitively we want to rely mostly on the error estimates from the local region. To cover extreme cases, where an expert is performing really bad in the last observations, we are adding a small amount of the recent average error. When all errors are estimated, we can sort the candidates by the error estimate and select the best of them for our ensemble.

The number of models in an ensemble is often selected arbitrarily. We make a proposal to set the number of models dynamically as well. As we discussed in section 3.2, an ensembles' performance depends on accurate and diverse members. Since there is no clear answer what an accurate member is, we try to use an estimate from the data gathered during training. After evaluating all the out-of-bag-samples during training, we compute a threshold of what we deem to be accurate on the given data. This threshold is the p-quantile of the prediction errors on the out-of-bag-samples. The strictness of this threshold is controlled by p, which is in $[0, 1]$. The 0.5-quantile is the median and used as default value. We evaluate the threshold against the estimated error $\hat{e}^i_{x_t}$ of each model and select the ones, below the threshold. The number of models is dynamic and can change for each example. If we estimate that there are many accurate models for the current example then we select many models. If there are only few accurate models, then our ensemble is smaller. This controls the number of models and adapts to the current example. It also takes away the arbitrary choice, that is made when selecting the number of models in the ensemble.

5.2.2 Ensemble Weighting

After composing the ensemble we have to think about weighting the members. Simply averaging the predictions of the selected candidates is a robust strategy for regression ensembles. A more sophisticated approach to weighting a regression ensemble is using a convex combination of the candidate predictions:

$$\hat{y} = \sum_i w_i \hat{y}_i, \qquad \sum_i w_i = 1, w_i \geq 0 \tag{5.5}$$

Our approach is to use the average errors $\bar{e}^i_{N_{x_t}}$ or $\bar{e}^i_{\{t-h,t-1\}}$, that were already calculated for the selection, as weights for the candidates. We call the variant using the errors from the local region, validation weighting and the other variant using the recent errors, sliding window weighting. Both variants are possible and there is a parameter for the model to select between the two variants. The default variant is to weight using the errors of the past predictions. Either way the weights for all candidates not selected for the ensemble are set to 0 and the weights get normalized, so that they sum up to 1. Consequently, the actual prediction calculation is in the default case

$$\hat{y} = \sum_i \frac{\bar{e}^i_{\{t-h,t-1\}}}{\sum_i \bar{e}^i_{\{t-h,t-1\}}} \hat{y}_i. \tag{5.6}$$

5.2.3 Sequential Reweighting

The importance of diversity in ensembles has already been discussed in subsection 3.3.2. An important insight was revealed in the Bias-Variance-Covariance decomposition, which showed that the covariance between the models is a factor for the expected error of the ensemble. For ensembles, there is more than the typical two-way trade-off of bias and variance. Ensembles need to balance the bias-variance-covariance three-way trade-off. There is not a clear way of transferring this knowledge into an objective function for the ensemble, but we can try to control the covariance in the ensemble.

Cerqueira et al.[22] presented a method called Sequential Reweighting (see Algorithm 5.2.1) to manage the diversity in a regression ensemble. The main idea of sequential reweighting is to adapt the weights of the ensemble, so that models are penalized if they are too similar to other, better performing models. They measure diversity as the correlation between the two models' predictions.

First Sequential Reweighting sorts the experts descending by their weight. Because the weights are our expectation of how good the model will perform, this means we sort by expected performance of the models. Next, a penalty η_{ij} based on the correlation of the two experts i, j is computed. If the experts predictions have a high correlation, then the penalty is high. Because of the sorting order, we know that expert i is expected to perform better than expert j. If their correlation is high, we want to decrease the weight of the lesser performing expert. The better performing experts weight is increased by the penalty, so that the weights still sum up to one.

Our version of the algorithm changes the set of predictions. Originally, only the predictions in the last h observations were used. We want to use the additional predictions

```
Input: weight of experts for $t$: $W$
Input: predictions for each expert $i$ in the last $h$ observations: $\hat{y}^i_{\{t-h,t-1\}}$
Input: predictions from the local region for each expert $i$: $\hat{Y}^i_{N_{x_t}}$
Output: adapted weights $W'$
1: $W \leftarrow Sort(W, \text{decreasing})$
2: $W'_i \leftarrow \{\}$
3: $W'_1 \leftarrow W_1$
4: for remaining expert $i$ in $W$ do
5:     $W'_i \leftarrow W_i$
6:     for expert $j$ in $W'$ do
7:         $p_i \leftarrow (\hat{y}^i_{\{t-h,t-1\}}, \hat{Y}^i_{N_{x_t}})$
8:         $p_j \leftarrow (\hat{y}^j_{\{t-h,t-1\}}, \hat{Y}^j_{N_{x_t}})$
9:         $cor_{ij} \leftarrow Cor(p_i, p_j)\text{L}$
10:        $\eta_{ij} \leftarrow cor_{ij} \cdot W'_j \cdot W'_i$
11:        $W'_j \leftarrow W'_j + \eta_{ij}$
12:        $W'_i \leftarrow W'_i - \eta_{ij}$
13:    end for
14: end for
15: return $W'$
```

Algorithm 5.2.1: Sequential Reweighting

from the local region. We calculate the correlation of the predictions from the last h observations $\hat{y}^i_{\{t-h,t-1\}}$ and the predictions of the examples in the local region $\hat{Y}^i_{N_{x_t}}$. The use of sequential reweighting is computationally expensive. Therefore, we deactivate this method in the default setting.

5.3 Overview of model parameter

DES-Forecasting tries to reduce the number of parameters the user has to set for the model selection. Still, there are some left to control the algorithm's behavior. These parameters are shown in Figure 5.3. Some parameters are mutually exclusive: If we set a value for the p-quantile, we do not use an arbitrary integer for the number of models. Some default choices have been explained in this chapter. For the error horizon, that is used for the size of the sliding window of past observations, the default value uses the embedding dimension of the time series. The embedding dimension is so chosen, that the time series is embedded in an euclidean space preserving the series. We deem the dimension of this space a good approach for how many recent observations matter for the current performance. In the experimental evaluation different combinations of these parameters are tested. A discussion over their importance and explanation for the default values is in section 6.3.

Parameter	Parameter Range	Default
Recency bias	$\alpha \in [0, 1]$	0.1
Weighting method	Validation data or sliding window	sliding
Error horizon	Steps for the sliding window	Emb. Dim K
Distance metric	Euclidean or DTW	DTW
Local Region Neighbors	Integer k	10
Use sequential reweighting	True, False	False
Use blocked prequential	True, False	False
Number of models in the ensemble	Integer	None
p-quantile for accuracy threshold	$p \in [0, 1]$	0.5

Figure 5.3: Overview of model parameters

Chapter 6

Experiments

In this chapter the experimental evaluation of our method DES-Forecasting is shown. We discuss the setup of the experiments and then take a look at the performance of the algorithm compared to single models and other ensemble algorithms. By comparing different versions of the algorithm, we take a closer look at the influence of the algorithm's parameters.

6.1 Experimental Setup

To evaluate the performance of our algorithm, we evaluate it on a reference task defined in [21]. The experiments use 57 real world univariate time series from different domains.[1] (for a list of all datasets see in the appendix Figure A.2)

Before using the data for predicting, the time series are preprocessed. The preprocessing steps handle some common tasks of preprocessing time series for forecasting. This includes the removal of trend, which is done by detecting trend using the KPSS statistical test[62] and removing it by differencing the time series. If a trend is detected, the time series is differenced until the test is passed. A second important step in preprocessing for time series forecasting is embedding the time series. It is important to find a good embedding dimension K, which is the number of past time steps that are used as features to predict the next time step. False Nearest Neighbors [55] is a method to estimate a good embedding dimension. The idea of the method is to calculate the neighbors of a window of size K and check whether they are real or false neighbors. By increasing K the number of false neighbors should decrease and when a certain tolerance is reached, a good estimate for K is found.

Additionally, the features set contains a set of common characteristics of time series that are computed in each embedding vector:

[1]The dataset is available at `https://github.com/vcerqueira/forecasting_experiments`

- Local trend, estimated according to the ratio between the standard deviation of the embedding vector and the standard deviation of the differenced embedding vectors
- Skewness, for measuring the symmetry of the distribution of the embedding vectors
- Mean, as a measure of centrality of the embedding vectors
- Standard deviation, as a dispersion metric
- Serial correlation, estimated using a Box-Pierce test statistic
- Long-range dependence, using a Hurst exponent estimation with wavelet transformation
- Chaos, using the maximum Lyapunov exponent to measure the level of chaos in the system

These features were described as statistics to summarize the overall structure of time series [90]. This results in a feature set $X_{[n,K]}$ and the target vector $y_{[K]}$.

$$X_{[n,K]} = \begin{bmatrix} y_1 & y_2 & \cdots & y_{K-1} & y_K & S_{trend_1} & \cdots & S_{chaos_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{i-K+1} & y_{i-K+2} & \cdots & y_{i-1} & y_i & S_{trend_i} & \cdots & S_{chaos_i} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ y_{n-K+1} & y_{n-K+2} & \cdots & y_{n-1} & y_n & S_{trend_n} & \cdots & S_{chaos_n} \end{bmatrix} \quad y_{[K]} = \begin{bmatrix} y_{K+1} \\ \vdots \\ y_{i+1} \\ \vdots \\ y_{n+1} \end{bmatrix}$$

To get a fair comparison between all ensemble methods, the candidate models were trained once on the dataset and then used in all ensembles. The model pool included the same set of models and parameters that were used in [21]. A full list of models and the parameters used can be found in the appendix. (See Figure A.4)

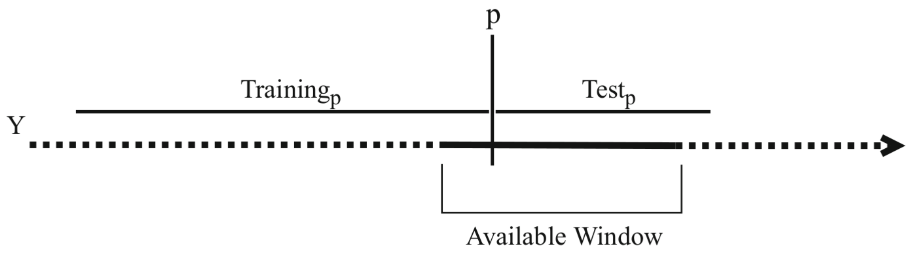

Figure 6.1: One iteration of the repeated holdout procedure [22]. The dashed line represents the time series Y. The point p is chosen from the available window. The training set are 50% of all observations directly before p and the subsequent 25% observations are used for testing

One experiments consists of two steps: Training model pool and ensembles on the train set. Evaluating model pool and ensembles on the test set. To get a good estimate of the expected performance, we use repeated holdout, which is a robust performance estimation method for time series data [21]. The method is illustrated in Figure 6.1. For the experiments each iteration uses 50% of all observations of the time series for training and 25% for testing. A random point p on the index of the time series is chosen from the available window. This window is constrained by the size for train and test set, so that for each point in the window not less than 50% of all observations lie before the point and not less than 25% after. After the point p is chosen the 50% of all observations directly before p form the training set and the subsequent 25% observations are used for testing. We calculate SMAPE and RMSE on the predictions of the test set. For comparing the RMSE predictions over different datasets the scores were normalized by the value range of the target. Each experiment is repeated 15 times for each dataset, so that different but overlapping observations of the time series are used. The final performance estimate for one dataset is the average over the 15 repetitions.

Visualizing and comparing model results over a set of datasets is complicated. It starts with the need to normalize performance measures and goes on with question about the significance of differences in the results. Is a model really significantly better than another model, because it performs better on some datasets? Is a model better when it performs better on average? How do we measure a significant difference?

There are numerous statistical tests that could be used for answering the questions. A study by Demšar [32] discussed the right usage of statistical tests for comparing models and their performance on different datasets. The main recommendation is to use the Wilcoxon-Signed-Ranked Test [92] for the pairwise comparison of models and the Friedman Test [45, 44] for the comparison of multiple models on multiple datasets. Both tests are more suitable than others because they make few assumptions about the data and are still powerful. Another issue is the visualization of the results. Demšar [32] proposed a diagram that presents the average ranks of the models over all datasets and additionally marks the critical differences between them. The ranks of the models are computed for each dataset by sorting the models by their performance and assigning rank 1 to the best performing model. The next best performing model gets rank 2 until all models are ranked.

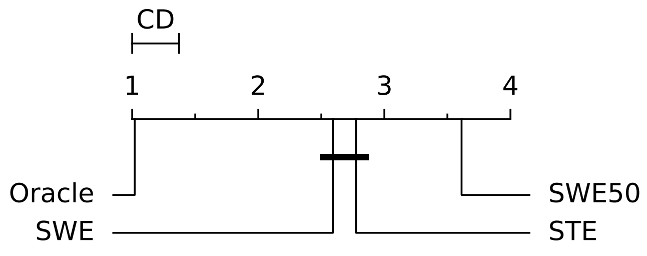

Figure 6.2: CD Diagram for the baseline models. SlidingWindowEnsemble is abbreviated as SWE

For demonstration, we show in Figure 6.2 a comparison of the four baseline models we use for our experiments. The models are, a static ensemble (STE) using the average over all predictions as in Equation 3.22, a sliding window ensemble (as in Equation 3.24) with horizon $h = 5$ (SWE) and one with horizon $h = 50$ (SWE50). We also include the oracle, that was presented in section 4.3. The top line in the diagram is the possible range for the average ranks of the models. Since there are only four models used in this comparison, the ranks go from 1 to 4. As we can see the oracle has an average rank of 1, which means it was the best performing model on all datasets. This is by definition of the oracle no surprise, because the oracle is defined as an upper limit for dynamic selection. The sliding window ensemble with the big horizon, has a big gap to the other two ensemble methods. The results of the statistical tests are visualized using the blacks bars connecting the results lines of models. If two models are connected with a black bar, they are not significantly different. In this case we see that the static ensemble and the sliding window ensemble with small horizon are not significantly different. The worst model is the sliding window ensemble with big horizon and the difference to the other models is significant. This is an interesting result in itself, as it shows the importants of recent errors. We will see these models again in the evaluation of our method.

6.2 Results

In this section we discuss the results of the experiments presented above. The main interest is to see how the DES-Forecasting will compete against static and sliding window ensembles. To get a good understanding of the results, we will briefly take a look at the results of the candidate models.

The created model pool is a set of models that show varying performance on different datasets. In Figure 6.3 we see the distribution of SMAPE scores across the datasets. The exact settings for each model shown in the figure are explained in Figure A.5. The names of the candidate models and the associated parameter names do not change in this chapter. Each shown score is the performance estimation of the repeated holdout procedure. It is no surprise that the scores cover a big value range, as we expect some time series to be harder to predict than others. We also see that the models TBATS and PPR3 contain a number of extreme outliers. We expect our dynamic selection algorithm to control these outliers. For a better understanding of the models performance compared to each other, we take a look at the rank distribution of each model accross the datasets in Figure 6.4.

The rank distribution shows many models with a wide range of ranks accross the datasets. Especially Exponential Smoothing (ETS), some versions of Support Vector Regression (SVR) and versions of Project Pursuit Regression (PPR) are standing out. ETS has the lowest median rank of all models but at the same time the biggest quantile range. For many datasets ETS is the best performing model and for other datasets the model

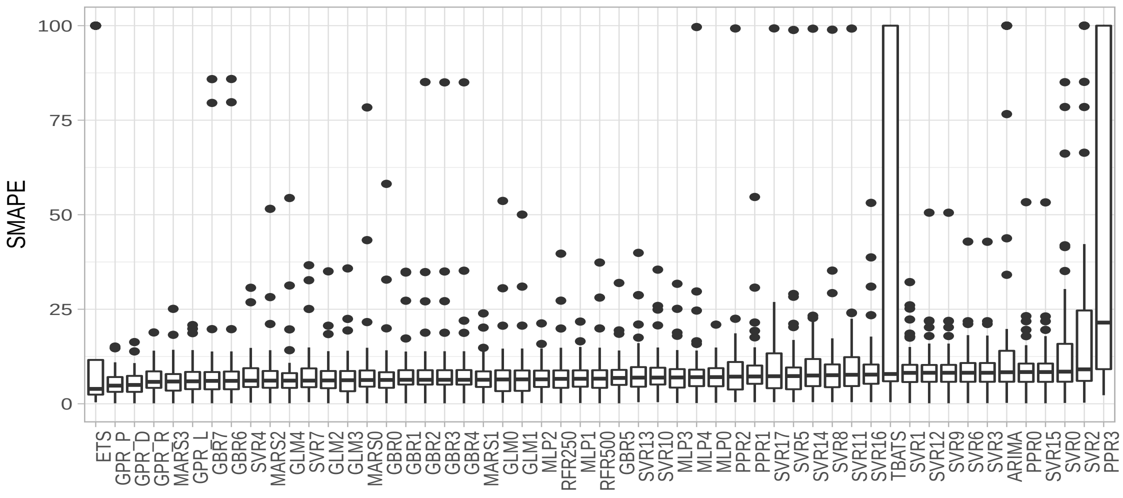

Figure 6.3: Distribution of the SMAPE scores of all candidate models across different time series. Each abbreviation stands for a model and a specific parameter set. The explanation for abbreviations for the models can be found in Figure A.5

completly failes to make useful predictions. Although not in such an extreme way, the same arguments can be made for other models that show a big range of ranks. There are also models that make solid predictions over all datasets, which can be detected by a rather low variance in their rank. These models do not perform at the highest level, but never fail drastically. The wide range of ranks for some models shows that not only are some time series harder to predict in general, but that some models are performing better or worse on specific time series. A single model is not the best predictor for each time series, but has potentially the best performance on only a few specific time series. A deeper analysis of the used time series would propably reveal the characteristics of a time series that make certain models perform better or worse. For our approach we do not want to analyse the time series further, but use an ensemble selection approach that is able to detect a good performing model on its own and select the set of models to perform well together.

We compare it to ensemble methods and single models of the candidate pool, to show the power of the method. Our method DES-Forecasting performs significantly better than ensemble methods like static or sliding window ensemble. The graphics use some abbreviations of the ensemble names, to make more efficient use of space. We compare our method DES-Forecasting (DESF) with a static ensemble (STE) and a sliding window ensemble (SWE). For the sliding window ensemble we use the version with horizon $h = 5$. We also include examples for ensemble methods using Bagging & Boosting in the form of a Random Forest (RFR250) and a Gradient Boosting Regression (GRB7). To compare single candidate models as well, we include Support Vector Regression with a linear kernel (SVR4) and ARIMA as one of the most popular time series models. This way we can compare the ensemble methods with the single models. The NAIVE forecasting is also

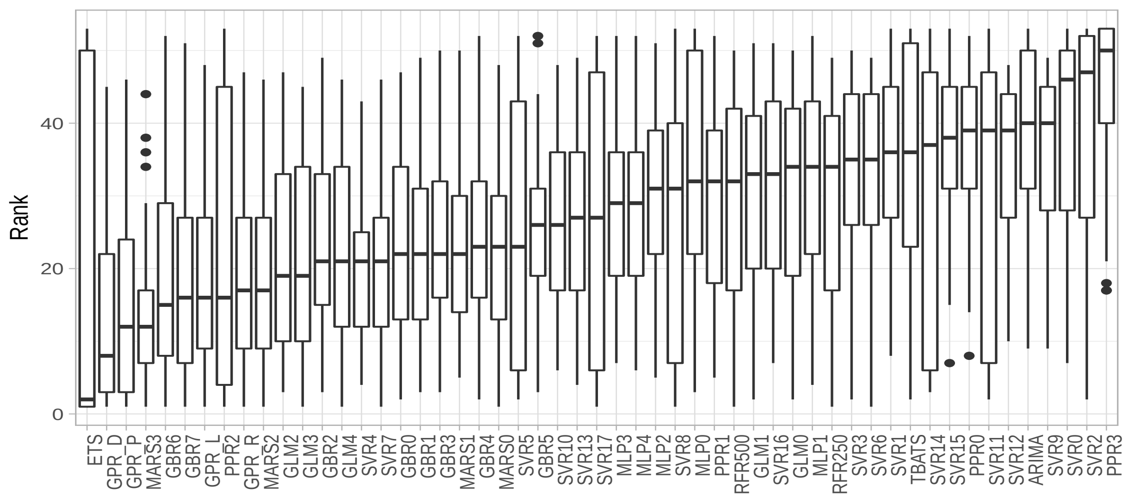

Figure 6.4: Distribution of the ranks of all candidate models across different time series. Each abbreviation stands for a model and a specific parameter set. The explanation for abbreviations for the models can be found in Figure A.5

included. This approach uses the last seen value as prediction for the next. It is a very low baseline each of the candidates should be able to outperform. We use the following parameter setting for DES-Forecasting:

- Recency bias: 0.1
- Weighting method: sliding
- Distance metric: DTW
- Number of models: automatic with 0.5 quantile
- Sequential reweighting: False
- Blocked Prequential: False

This parameters are the suggested default setting. We compare more combinations of parameter settings in section 6.3 and explain our choices for the default setting.

At first, we look at Figure 6.5 and the average ranks and scores of the discussed models. DES-Forecasting clearly outperforms all other models in terms of average rank and average score. The other ensemble methods, except for the random forest are performing similar. The SVR is the best of the three included candidate models and it outperforms the random forest. The very popular ARIMA method does not perform well on average and is close to the level of the NAIVE prediction. The improvement in the average rank of DES-Forecasting could be misleading, but the big difference in average score shows that the algorithm clearly outperforms the competitors.

To further analyze this difference we take a closer look at the ranks and their distribution in Figure 6.6. The plot shows the performance advantage of DES-Forecasting compared to the other methods. It is expected that all methods cover a range of ranks and that none of the methods is the best for every dataset. DES-Forecasting is performing as

Model	Avg. Rank	Std. Rank	Avg. Score	Std. Score
DES-Forecasting	**2.12**	1.4	**4.97**	3.4
Sliding Window Ensemble	3.56	2.1	8.2	10.1
Grad. Boost. Regression	3.66	1.6	8.8	14.5
Static Ensemble	3.86	2.2	8.2	9.5
Supp. Vector Regression	4.15	1.7	7.42	5.3
Random Forest	5.23	1.7	7.34	6.2
ARIMA	6.4	1.3	17	26.2
NAIVE	7	1.4	9.87	8

Figure 6.5: Rank and score results of model, averaged over 57 datasets. Models are sorted based on the average rank. Scores are measured using SMAPE.

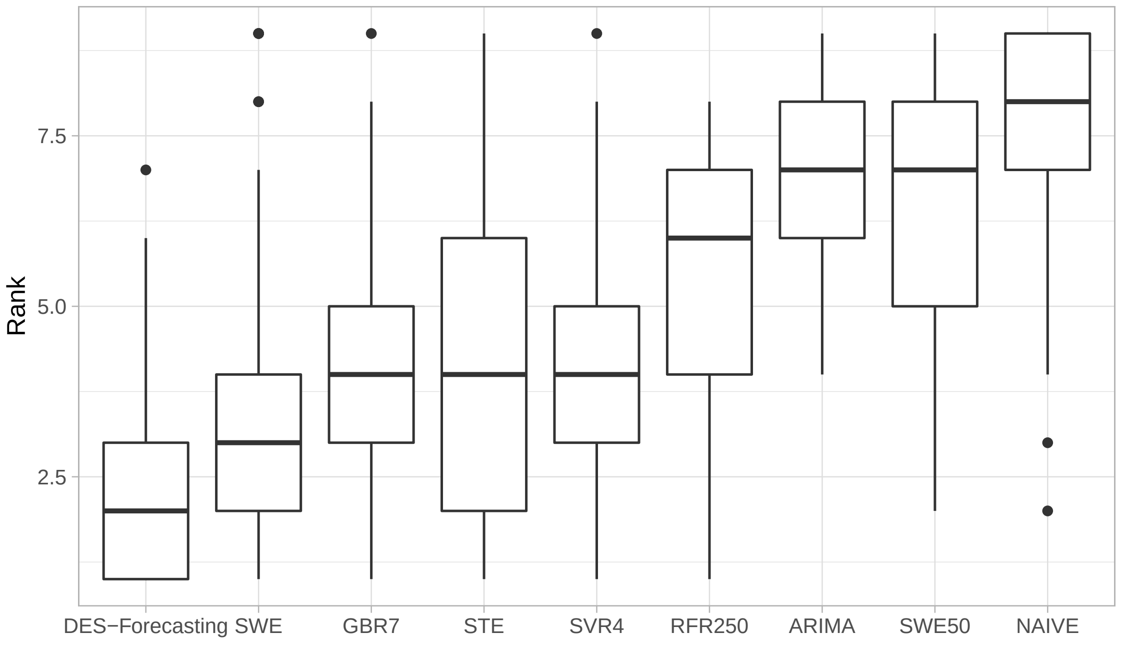

Figure 6.6: Distribution of the ranks of ensemble methods across the different time series

one of the top-3 models for most of the datasets. The other ensemble methods show consistent performance as well, but not on the same high level as DES-Forecasting. The model with the biggest range of performances is the static ensemble, which can be explained by the combination method. A simple average over all candidates predictions is sensitive to outliers. In contrast, a sliding window ensemble will notice the outliers by the errors in the recent predictions. DES-Forecasting will avoid the outlier predictions by detecting their bad performance on the validation data.

We use the friedman test to further determine, whether the difference in the average ranks are significant. For visualization, we use the critical difference diagram. As shown in

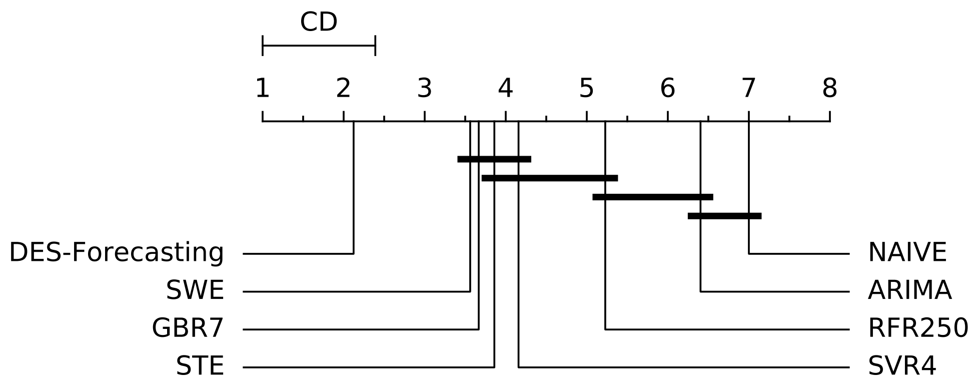

Figure 6.7: Critical difference diagram of ensemble methods

Figure 6.7, DES-Forecasting is the best performing model. We also note that the statistical tests show that the difference in the average rank is significant. It is interesting to note that the other methods are all very similar. From the ensemble methods, only the random forest performs notably worse than the other, and we see that a good single model like the SVR can compete with simple ensemble methods.

6.3 Parameter Study

The DES-Forecasting algorithm has a set of parameters that control its methods and behavior. The parameters in other ensemble learning methods are often set arbitrarily, like for example the number of selected models. One of the motivations behind dynamic selection is to automate the process of model and ensemble selection. If it is sufficient to present a big heterogeneous model pool to an dynamic selection algorithm, all the effort that is typically put into model selection can be used for other parts of the prediction system. To really automate the process, an algorithm without extra hyperparameters would be optimal. We try to reduce the parameters that have to be chosen. Our algorithm offers a few choices to the user and we will argue why the chosen default values are a suitable choice for various problems.

One important parameter for selection is the number of selected models. We argue in the presentation of the method, that this value is often selected arbitrarily for other ensemble methods. We propose a dynamic number that is based on a threshold from the validation data. This threshold is the p-quantile of the errors on the validation data. We compare three different values p-quantile versions, and 5 static values s. The static numbers are 1,5,10,26 and 50. 26 is chosen because it is half the size of the model pool. The other values are chosen arbitrarily. So the p-quantile versions will calculate for each prediction, how many models are expected to be accurate. The versions with static numbers will select the best s models according to the estimated performance in the local region.

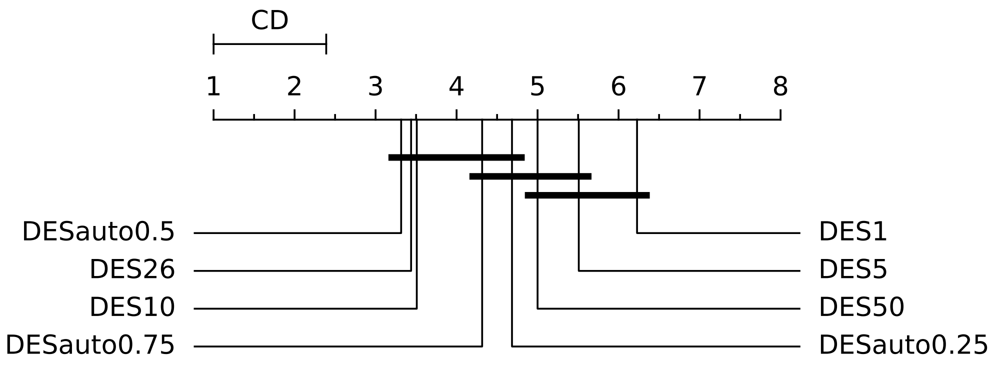

Figure 6.8: Critical difference diagram of DES-Forecasting with different parameter settings for the number of models.

In Figure 6.8 we can see the critical difference diagram of the different parameter settings. The default setting of using the automatic ensemble size selection with the 0.5-quantile has the lowest average rank. There is no critical difference to using one of the other p-values or using 10 or 26 models. Compared to the default choice, there are significant differences when using $1, 5$ or 50 models. In the case of 1 model, the algorithm does a dynamic model selection. We can see that selecting only one model is worse than selecting an ensemble and that there is a significant difference to the default parameter choice. For our experiments the model pool consists of 52 models. Selecting 10 models or half of the model pool for an ensemble are frequently used values because experience with ensembles suggest that they work well. These values are nothing more than a rule of thumb suggestion, that might work. The 0.5-quantile outperforms the static values and does not require an arbitrary choice by the user.

Another important parameter of DES-Forecasting is the so called recency bias. This factor balances the importance of errors in the recent predictions against the errors on the local region in the validation data. A high recency bias puts more focus on the recent predictions then the local region. The extreme values of 0 and 1 are equivalent to only using the local region in case of 0 and only using the recent predictions in case of 1. We exclude the value of 1 from our evaluation because it removes the region of competence computation completely from our algorithm. The importance of recent predictions is specifically useful for time series data, because of the dependency between the values of the series. We compare different values for the recency bias, $\{0, 0.1, 0.5\}$, and combine them with the default choices for the other parameters.

The results in Figure 6.9 show that there is a significant difference between 0.1 and 0.5. The tendency in the averages hints also at using low values for the recency bias. Our default choice is 0.1, because we saw extreme cases of overfitting to the trainings data in our experiments. These outliers could go unnoticed even with validation data. If the

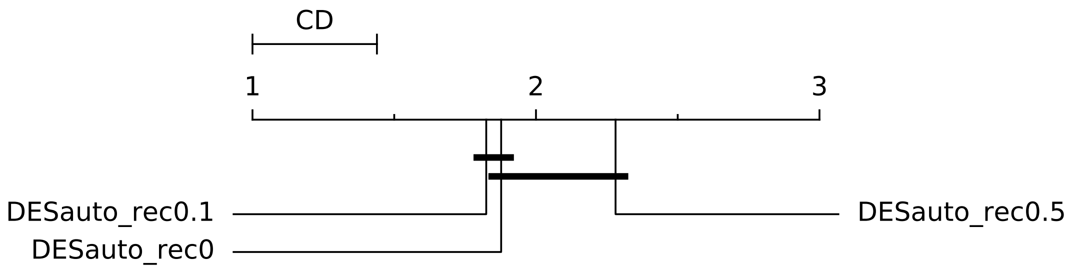

Figure 6.9: Critical difference diagram of DES-Forecasting with settings for the recency bias.

performance on the recent predictions is included to a small degree, then extreme outliers are avoided.

We presented two different methods for generating the combinations weights of the ensemble. Either using the performance estimation from local region, which we call validation, or using the recent errors, which we call sliding window. Since the validation method is clearly dependent on the local region, we also take the used distance measure into account. Dynamic time warping is specific for time series data, but for comparison we include the local region computation with euclidean distance. Combining these options we get four versions of the algorithm, which we compare intensively with the pairwise comparison using the Wilcoxon-Signed-Ranked test. For the test we compare the residuals of the predictions of each algorithm per dataset. The test will show us if the residuals of the two compared models are significantly greater or smaller, or if the difference is insignificant. A model performs better, or wins, if its residuals are significantly smaller than the other model's residuals. Since the sliding window weighting and the DTW distance are our default values, we compare them against the other three methods. We see the test result statistics in Figure 6.10. The distance function does not make a great impact, as 54 of 57 datasets show no significant difference in the prediction residuals if we change only the distance metric to euclidean. The results for the weighting method are much clearer. On more than 20 datasets the sliding window weighting performs significantly better than the validation weighting, regardless of the used distance metric. It is also to note that for none of the datasets the default choices are significantly worse.

The idea of using a blocked prequential procedure, was motivated by making the most of the given data. We compared a simple holdout set with a blocked prequential procedure for training the algorithm. The average results are slightly worse with blocked prequential than with a holdout set. This is due to some outlier results in a few datasets. Removing these outliers, the results are on par with the holdout set. Training the algorithm with a blocked prequential procedure took much longer then a simple holdout set. Since the blocked prequential procedure trains each model b times we can expect a factor of up to b more time. In our experiments for $b = 5$ the blocked prequential training took close to 3 times as long as the holdout procedure. Therefore, we choose to use the holdout version

Model	Win	Insignificant	Loss
DESF-validation-euclidean	22	35	0
DESF-validation-DTW	23	34	0
DESF-sliding-euclidean	3	54	0

Figure 6.10: Results of pairwise model comparison using the Wilcoxon-Signed-Ranked test. The models are compared against the default version of DES-Forecasting with DTW distance and sliding window weighting. A win means the default version's residuals are significantly smaller than the residuals of the compared model. If they are insignificant, then the dataset is counted in that column.

as default choice. If the computational costs are not a factor, using a blocked prequential procedure can be useful.

Another important parameter is used for activating sequential reweighting. It seems like a good idea to control the diversity of the ensemble members by changing their weights if they are too similar. We tested models with activated and deactivated sequential reweighting against each other and in all cases, the sequential reweighting did not improve the results significantly. They were on average slightly worse. Another factor is, that the multiple calculations of correlation between the ensemble members prediction have a high computational cost and increase the run time of the prediction by a factor of nearly 3. For our approach of dynamic selection based on local regions, sequential reweighting is not improving our method. This result is only empirical for our method, and we note that others used it successfully [22]. The need for balancing the diversity might not be that high for a method using dynamic selection specific to each example.

6.4 Results comparing with State-of-the-Art

In this section we will compare our algorithm DES-Forecasting with other state-of-the-art methods, especially those presented and shown by Cerqueira et al. [22]. The authors published their experiment code that created their results. As discussed in section 6.1, we constructed our experiments on the same datasets and replicated the model pool. In Figure 6.11, we can see that the rank distribution of their candidate models resembles our own results in Figure 6.4. There are some differences but both models pools show a good diversity and are suitable for ensemble construction. Differences between the implementation of each candidate model may cause some differences but the model pools should be similar enough to compare ensemble methods on them.

The main contribution of Cerqueira et al. [22] was their algorithm ADE. The algorithm uses a combination of selection, base on recent errors, and a meta learning approach for weighting the ensemble members. The meta learning is done with a random forest, that

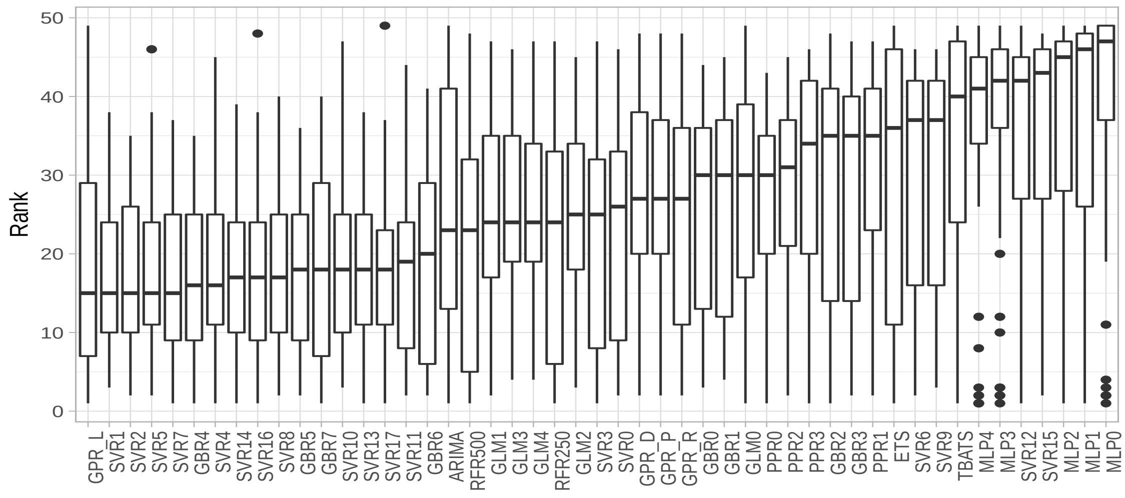

Figure 6.11: Distribution of the ranks of all candidate models across different time series. Each abbreviation stands for a model and a specific parameter set. The explanation for abbreviations for the models can be found in Figure A.5

estimates each candidates loss for the current example. Other methods that are compared are:

- AEC[77]: It uses an exponential re-weighting strategy to generate convex weights and includes a forgetting factor to give more importance to recent values
- OGD[95]: An approach based on online gradient descent that provides theoretical loss bound guarantees
- trimmedSimple: An average ensemble that uses 50% of the experts, based on recent performance
- FixedShare[51]: The fixed share approach from Herbster and Warmuth, which is designed for tracking the best expert across a time series
- Sliding Window Ensemble

First, we look at the average ranks and scores of DES-Forecasting against the other methods. The average rank and especially the average score of our method, seem to indicate that it is competitive with state-of-the-art methods like ADE. If we look closely at the results, we can see high values for the standard deviation of the score of the methods from the other experiments. It seems like there are extreme outliers for some methods, especially the ADE. The average rank of 2.89, which is the very close to our method, but a difference of 2 in the average score, is a clear indicator.

For computing the performances, the predictions for all models were collected and compared to the truth values. The experiments only differ in the implementation language. We see that our method has the best average rank and score compared to the other ensemble methods. The difference between DES-Forecasting and ADE is very small. Even without a statistical test we can be pretty sure that the difference is not significant. The differences

Model	Avg. Rank	Std. Rank	Avg. Score	Std. Score
DES-Forecasting	**2.86**	2.5	**5.16**	3.5
ADE	2.89	1.5	7.1	6
trimmedSimple	3.73	1.5	8.0	10.5
FixedShare	4.26	1.6	6.58	4.1
OGD	4.65	1.7	7.27	6.4
AEC	4.69	2.1	6.56	3.8
SlidingWindowEnsemble	4.89	1.6	8.45	11.1

Figure 6.12: Rank and score results of models, averaged over 57 datasets. Models are sorted based on the average rank. Scores are measured using SMAPE.

to the other models seem more significant, and we will look at the statistical tests later. It is to note that DES-Forecasting has a high deviation in rank, which means that the predictions are not extremely robust compared to the other methods in terms of ranking. On the other hand we should note that deviation and average of the score are the lowest for DES-Forecasting. Even though the other methods might perform better for some datasets, they all have more outliers in their scores.

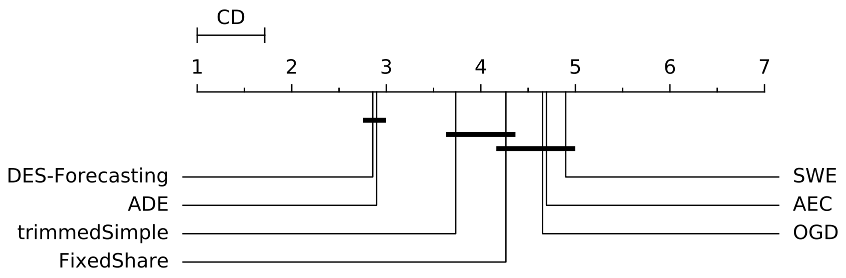

Figure 6.13: Critical difference diagram of comparing DES-Forecasting with ADE and other ensemble methods.

In Figure 6.13, we look at the critical difference diagram for the average ranks of DES-Forecasting, ADE and the other ensemble methods. We can clearly see that ADE and DES-Forecasting are the top performing methods. There is no critical difference between them, but to all the other methods. It is very interesting to the power of using a good selection: The top-3 algorithm perform a selection. TrimmedSimple performs a selection only based on the recent performance and outperforms the other methods with more complex combination methods. The results show that a good selection strategy produces a good forecasting ensemble. If the selection strategy is further combined with a good combination method, then the results are even better.

6.5 Comparison with AutoSK

AutoSK [39] is a framework that provides automated machine learning using Bayesian optimization, meta-learning and ensemble construction. The algorithm treats the full process of supervised learning as a highly parametric problem and optimizes it's parameters to find a good solution. This optimization includes preprocessing, selecting models, tuning model's hyperparameters, and constructing ensembles. Without further user attention a full prediction pipeline is generated. We used the framework on the datasets described above and compared it with our method DES-Forecasting. Since we try to automate the selection step of a prediction pipeline, we think it might be interesting to see our method compared to a fully automated framework.

The defaults parameters of the autoSK regression model are used for the comparison. We note that a subset of the datasets was used, because the default parameters of the framework fail for 12 datasets. We reduce our comparison to these 35 datasets and note that the automatic framework is not very robust. The performance of autoSK is impressive. Across the 37 datasets, it could outperform DES-Forecasting on 16. But there was only a significant difference on six datasets of these 16. We also to note that the autoSK results have a high variance. DES-Forecasting has an average SMAPE score of 4.75 over these 37 datasets, while autoSK only achieved 15.88 with a high standard deviation of 21. There are clear problems in the framework and the predictions still require user attention. The required time to train autoSK is very high. The default constraint for training the model is one hour. With this setting the algorithm will optimize for one hour to find the best possible pipeline. Training a big model pool, like our method does, is also time-consuming. Depending on the size of the dataset, our algorithm finished training in at most 30 minutes.

The DES-Forecasting algorithm needs an extension regarding the model pool generation, to make it an automatic framework. If the selection of models and their hyperparameters for the pool would be automated and not initially chosen by the user, our method could directly compete with automated machine learning methods for the domain of time series.

Chapter 7

Further Discussion

In this thesis we have presented the DES-Forecasting algorithm for dynamic time series forecasting. We have transferred methods of dynamic model selection, which have been only used for classification task so far, to regression tasks and combined them with dynamic ensemble weighting. The ensemble members are selected based on their performance on local regions, which consist of examples from validation data similar to the current example. For combination of the ensemble members, each members is weighted according to its recent performance. The use of recent performance restrain the method to forecasting tasks. The selection of our method could also be used for other regression tasks.

We have shown that the presented method has potential as an ensemble method for time series forecasting. Further extensions of the method could include covering more topics of time series forecasting. The presented algorithm is currently restricted to univariate time series, because of the region of competence computation. All other parts are already able to work with multivariate time series. The similarity measure for the local regions has to be adapted to the multivariate case. This is especially true for the dynamic time warping distance, which is not compatible to multivariate time series in the current implementation. The euclidean distance will already work. Further evaluation have to show, if the current weighting approaches are also suitable for multivariate times series. This is more a concern about quality than about functionality. We only covered step-wise forecasting in this thesis. Extending to multi-step forecasting could be another interesting topic.

In case of a concept the dynamic selection might be able to adapt, by using other models. If the time series changes into a completely new concept, dynamic selection is not a solution. Only introducing new models that are trained on the new concept, or retraining the current models are options. Tracking the loss of the ensemble and retraining, when the performance increases beyond tolerance, might be a possible solution.

We have also shown that dynamic model selection is a more powerful alternative to static model selection. It also eases the users work by integrating the model selection into the algorithm, instead of it being a step of the working pipeline. The current version of the algorithm uses an arbitrarily chosen set of models. To automate even more parts of the algorithm, the model pool generation could be automated. If we measure accuracy and diversity of the model pool, we can estimate its quality and guide the pool generation. How to construct the best model pool, is still an open question.

The experimental evaluation using different time series has shown that our method is competitive with state-of-the-art ensemble methods for forecasting. The experiments were not focused on a single time series, but covered time series from different domains. Additional studies could investigate the impact of the model pool's size and quality on the performance of our method.

Appendix A

Appendix

Time Series	Data Source	Size	K	I
Electricy total load	Hospital energy loads [23]	3000	19	0
Gas energy		3000	10	0
Gas heat energy		3000	13	0
Water heater energy		3000	30	0
Total demand	Australian electricity [58]	2833	6	0
Recommended retail price		2833	19	0
Min. temperature	Porto Weather [23]	1456	8	0
Sea level pressure	Ozone level detection [36]	2534	9	0
Geo-potential height		2534	7	0
K index		2534	7	0
Rotunda AEP	Porto water consumption from different locations in the city of Porto [23]	3000	30	0
Preciosa mar		3000	9	1
Amial		3000	11	0
Total bike rentals	Bike sharing [23]	1338	8	0

Figure A.1: List of Datasets used for the experiments. K is the used embedding dimension. I is the number of times the series was differenced to remove trend.

Time Series	Data Source	Size	K	I
Global horizontal radiation	Solar radiation monitoring [23]	3000	23	1
Direct normal radiation		3000	19	1
Diffuse horizontal radiation		3000	18	1
Average wind speed		3000	10	1
CO.GT	Air quality indicators in an italian city [36]	3000	30	1
PT08.S1.CO		3000	8	1
NMHC.GT		3000	10	1
C6H6.GT		3000	13	1
PT08.S2.NMHC		3000	9	1
NOx.GT		3000	10	1
PT08.S3.NOx		3000	10	1
NO2.GT		3000	30	1
PT08.S4.NO2		3000	8	1
PT08.S5.O3		3000	8	1
Temperature		3000	8	1
RH		3000	23	1
AeroStock1	Stock price values from different aerospace companies [23]	949	6	1
AeroStock2		949	13	1
AeroStock3		949	7	1
AeroStock4		949	8	1
AeroStock5		949	6	1
AeroStock6		949	10	1
AeroStock7		949	8	1
AeroStock8		949	8	1
AeroStock9		949	9	1
AeroStock10		949	8	1

Figure A.2: Continuation list of Datasets used for the experiments. K is the used embedding dimension. I is the number of times the series was differenced to remove trend.

Time Series	Data Source	Size	K	I
Flow of Vatnsdalsa river	Data market [52]	1095	11	0
Rainfall in Melbourne		3000	29	0
Max temperatures in Melbourne		3000	7	0
Min temperatures in Melbourne		3000	6	0
Rainfall in Melbourne		3000	29	0
Foreign exchange rates		3000	6	1
IBM stock		1008	9	0
Internet traffic data 1		1231	10	0
Internet traffic data 2		1657	11	1
Internet traffic data 3		3000	6	1
Flow of Jokulsa river		1096	21	0
Flow of saugeen river 1		1400	6	0
Flow of saugeen river 2		3000	30	0
Flow of oldman river		1461	6	0
Precipitation of oldman river		1461	29	0
Flow of fisher river		1461	6	0
No of births in quebec		3000	6	1

Figure A.3: Continuation list of Datasets used for the experiments. K is the used embedding dimension. I is the number of times the series was differenced to remove trend.

ID	Algorithm	Parameter	Value
SVR	Support Vector Regression[71]	Kernel	{Linear, RBF, Polynomial}
		C	{1, 5, 10}
		ϵ	{0.001, 0.01}
MARS	Multivariate A. R. Splines[75]	max-degree	{1, 3}
		max-terms	{7, 15}
RF	Random Forest[71]	Number Trees	{250, 500}
PPR	Projection Pursuit Regression[43]	Number Terms	{2,5}
		Method	{polyfit, spline}
GBR	Gradient Boosting Regression[71]	Depth	{5, 10}
		Loss	{ls, lad}
		Number Trees	{500, 1000}
		Learning Rate	{0.1}
MLP	Multi-layer perceptron[71]	Hidden Units	{3, 5, 10, 15, 25}
GLM	Elastic Net[71]	l1-ratio	{0, 0.25, 0.5, 0.75, 1}
GPR	Gaussian Process Regression[71]	Kernel	{Linear, RBF}
		Kernel	{Polynomial, Laplacian}
ARIMA	ARIMA[80, 83]	Auto	
ETS	Exponential Smooting[80, 82]	Method	{ETS, TBATS}

Figure A.4: List of Candidate Models. Each parameter combination was used as one model in the model pool of the experiments. The citation references the used implementation. The references to the publication of the methods can be found in section 3.1

ID	Algorithm	Parameter	Value
MLP0	Multi-layer perceptron	hidden-layer-sizes	(25,)
MLP1	Multi-layer perceptron	hidden-layer-sizes	(15,)
MLP2	Multi-layer perceptron	hidden-layer-sizes	(10,)
MLP3	Multi-layer perceptron	hidden-layer-sizes	(5,)
MLP4	Multi-layer perceptron	hidden-layer-sizes	(3,)
GLM0	Elastic Net	l1-ratio	0
GLM1	Elastic Net	l1-ratio	0.25
GLM2	Elastic Net	l1-ratio	0.5
GLM3	Elastic Net	l1-ratio	0.75
GLM4	Elastic Net	l1-ratio	1
GBR0	Gradient Boosting Regression	Depth, Loss, Number Trees	(5, ls, 500)
GBR1	Gradient Boosting Regression	Depth, Loss, Number Trees	(5, ls, 1000)
GBR2	Gradient Boosting Regression	Depth, Loss, Number Trees	(10, ls, 500)
GBR3	Gradient Boosting Regression	Depth, Loss, Number Trees	(10, ls, 1000)
GBR4	Gradient Boosting Regression	Depth, Loss, Number Trees	(5, lad, 500)
GBR5	Gradient Boosting Regression	Depth, Loss, Number Trees	(5, lad, 1000)
GBR6	Gradient Boosting Regression	Depth, Loss, Number Trees	(10, lad, 500)
GBR7	Gradient Boosting Regression	Depth, Loss, Number Trees	(10, lad, 1000)
SVR0	Support Vector Regression	Kernel, C, ϵ	(rbf, 1, 0.001)
SVR1	Support Vector Regression	Kernel, C, ϵ	(linear, 1, 0.001)
SVR2	Support Vector Regression	Kernel, C, ϵ	(poly, 1, 0.001)
SVR3	Support Vector Regression	Kernel, C, ϵ	(rbf, 1, 0.01)
SVR4	Support Vector Regression	Kernel, C, ϵ	(linear, 1, 0.01)
SVR5	Support Vector Regression	Kernel, C, ϵ	(poly, 1, 0.01)
SVR6	Support Vector Regression	Kernel, C, ϵ	(rbf, 5, 0.001)
SVR7	Support Vector Regression	Kernel, C, ϵ	(linear, 5, 0.001)
SVR8	Support Vector Regression	Kernel, C, ϵ	(poly, 5, 0.001)
SVR9	Support Vector Regression	Kernel, C, ϵ	(rbf, 5, 0.01)
SVR10	Support Vector Regression	Kernel, C, ϵ	(linear, 5, 0.01)
SVR11	Support Vector Regression	Kernel, C, ϵ	(poly, 5, 0.01)
SVR12	Support Vector Regression	Kernel, C, ϵ	(rbf, 10, 0.001)
SVR13	Support Vector Regression	Kernel, C, ϵ	(linear, 10, 0.001)
SVR14	Support Vector Regression	Kernel, C, ϵ	(poly, 10, 0.001)
SVR15	Support Vector Regression	Kernel, C, ϵ	(rbf, 10, 0.01)
SVR16	Support Vector Regression	Kernel, C, ϵ	(linear, 10, 0.01)
SVR17	Support Vector Regression	Kernel, C, ϵ	(poly, 10, 0.01)

Figure A.5: List of Candidate Models. Mapping of names used in the thesis and parameter combination.

ID	Algorithm	Parameter	Value
RF250	Random Forest	Number Trees	250
RF500	Random Forest	Number Trees	500
PPR0	Projection Pursuit Regression	Number Terms, Method	2,polyfit
PPR1	Projection Pursuit Regression	Number Terms, Method	5,polyfit
PPR2	Projection Pursuit Regression	Number Terms, Method	2,spline
PPR3	Projection Pursuit Regression	Number Terms, Method	5,spline
MARS0	Multivariate A. R. Splines	max-degree, max-terms	(1,7)
MARS1	Multivariate A. R. Splines	max-degree, max-terms	(1,15)
MARS2	Multivariate A. R. Splines	max-degree, max-terms	(3,7)
MARS3	Multivariate A. R. Splines	max-degree, max-terms	(3,15)
GPR0	Gaussian Process Regression	Kernel	(Linear)
GPR1	Gaussian Process Regression	Kernel	(RBF)
GPR2	Gaussian Process Regression	Kernel	(Polynomial)
GPR3	Gaussian Process Regression	Kernel	(Laplacian)
ARIMA	ARIMA	Auto	
ETS	Exponential Smooting		
TBATS	TBATS		

Figure A.6: Continuation of list of Candidate Models. Mapping of names used in the thesis and parameter combination.

Bibliography

[1] Marco Aiolfi and Allan Timmermann. “Persistence in forecasting performance and conditional combination strategies”. In: *Journal of Econometrics* 135.1-2 (2006), pp. 31–53.

[2] Sylvain Arlot and Alain Celisse. “A survey of cross-validation procedures for model selection”. In: *Statistics surveys* 4 (2010), pp. 40–79.

[3] Anthony Bagnall et al. “The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances”. In: *Data Mining and Knowledge Discovery* 31.3 (May 2017), pp. 606–660.

[4] A. Bagnall et al. “Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles”. In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (Sept. 2015), pp. 2522–2535.

[5] Christoph Bergmeir and José M. Benítez. “On the use of cross-validation for time series predictor evaluation”. In: *Information Sciences* 191 (2012). Data Mining for Software Trustworthiness, pp. 192–213.

[6] Christoph Bergmeir, Mauro Costantini, and José M. Benítez. “On the usefulness of cross-validation for directional forecast evaluation”. In: *Computational Statistics & Data Analysis* 76 (2014). CFEnetwork: The Annals of Computational and Financial Econometrics, pp. 132–143.

[7] Christoph Bergmeir, Rob J Hyndman, Bonsoo Koo, et al. “A Note on the Validity of Cross-Validation for Evaluating Time Series Prediction”. In: *Monash University Department of Econometrics and Business Statistics Working Paper* 10.April (2015), p. 15.

[8] Aaron Bostrom and Anthony Bagnall. “Binary Shapelet Transform for Multiclass Time Series Classification”. In: *Big Data Analytics and Knowledge Discovery.* Ed. by Sanjay Madria and Takahiro Hara. Cham: Springer International Publishing, 2015, pp. 257–269.

[9] George EP Box et al. *Time series analysis: forecasting and control.* John Wiley & Sons, 2015.

[10] Pavel Brazdil et al. *Metalearning: Applications to data mining*. Springer Science & Business Media, 2008.

[11] Leo Breiman. “Bagging predictors”. In: *Machine learning* 24.2 (1996), pp. 123–140.

[12] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.

[13] Leo Breiman. “Stacked regressions”. In: *Machine learning* 24.1 (1996), pp. 49–64.

[14] Alceu S. Britto, Robert Sabourin, and Luiz E.S. Oliveira. “Dynamic selection of classifiers - A comprehensive review”. In: *Pattern Recognition* 47.11 (2014), pp. 3665–3680.

[15] Peter J Brockwell, Richard A Davis, and Matthew V Calder. *Introduction to time series and forecasting*. Vol. 2. Springer, 2002.

[16] Gavin Brown. “PhD Thesis: Diversity in neural network ensembles”. PhD thesis. 2004, p. 123.

[17] Gavin Brown and Ludmila I. Kuncheva. ““Good” and “Bad” Diversity in Majority Vote Ensembles”. In: *Multiple Classifier Systems*. Ed. by Neamat El Gayar, Josef Kittler, and Fabio Roli. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 124–133.

[18] Gavin Brown, Jeremy L Wyatt, and Peter Tiňo. “Managing Diversity in Regression Ensembles”. In: *Journal of Machine Learning Research* 6.2 (2005), pp. 1621–1650.

[19] Gavin Brown et al. “Diversity creation methods: A survey and categorisation”. In: *Information Fusion* 6.1 (2005), pp. 5–20.

[20] L. J. Cao and F. E. H. Tay. “Support vector machine with adaptive parameters in financial time series forecasting”. In: *IEEE Transactions on Neural Networks* 14.6 (Nov. 2003), pp. 1506–1518.

[21] Vitor Cerqueira et al. “A comparative study of performance estimation methods for time series forecasting”. In: *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. IEEE. 2017, pp. 529–538.

[22] Vitor Cerqueira et al. “Arbitrage of forecasting experts”. In: *Machine Learning* (2018).

[23] Vitor Cerqueira et al. “Arbitrated ensemble for time series forecasting”. In: *Joint European conference on machine learning and knowledge discovery in databases*. Springer. 2017, pp. 478–494.

[24] Chris Chatfield. *The analysis of time series: an introduction*. Chapman and Hall/CRC, 2016.

[25] Chris Chatfield. *Time-series forecasting*. Chapman and Hall/CRC, 2000.

[26] Guillaume Chevillon. “Direct multi-step estimation and forecasting”. In: *Journal of Economic Surveys* 21.4 (2007), pp. 746–785.

[27] Robert T Clemen and Robert L Winkler. “Combining economic forecasts”. In: *Journal of Business & Economic Statistics* 4.1 (1986), pp. 39–46.

[28] Dwight B Crane and James R Crotty. “A two-stage forecasting model: Exponential smoothing and multiple regression”. In: *Management Science* 13.8 (1967), B–501.

[29] Rafael M.O. Cruz, Robert Sabourin, and George D.C. Cavalcanti. “Dynamic classifier selection: Recent advances and perspectives”. In: *Information Fusion* 41.May (2018), pp. 195–216.

[30] A Philip Dawid. “Present position and potential developments: Some personal views: Statistical theory: The prequential approach”. In: *Journal of the Royal Statistical Society. Series A (General)* (1984), pp. 278–292.

[31] Alysha M De Livera, Rob J Hyndman, and Ralph D Snyder. “Forecasting time series with complex seasonal patterns using exponential smoothing”. In: *Journal of the American Statistical Association* 106.496 (2011), pp. 1513–1527.

[32] Janez Demšar. “Statistical comparisons of classifiers over multiple data sets”. In: *Journal of Machine Learning Research* 7 (2006), pp. 1–30.

[33] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: *Proceedings of the First International Workshop on Multiple Classifier Systems* (2000), pp. 1–15.

[34] E. M. dos Santos and R. Sabourin. “Classifier ensembles optimization guided by population oracle”. In: *2011 IEEE Congress of Evolutionary Computation (CEC).* June 2011, pp. 693–698.

[35] Harris Drucker et al. “Support vector regression machines”. In: *Advances in neural information processing systems.* 1997, pp. 155–161.

[36] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository.* 2017.

[37] Haimonti Dutta. “Measuring diversity in regression ensembles”. In: *Iicai* (2009).

[38] Ehab E Elattar, John Goulermas, and Q Henry Wu. “Electric load forecasting based on locally weighted support vector regression”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 40.4 (2010), pp. 438–447.

[39] Matthias Feurer et al. “Efficient and Robust Automated Machine Learning”. In: *Advances in Neural Information Processing Systems 28.* Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 2962–2970.

[40] Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1 (1997), pp. 119–139.

[41] Jerome H Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* (2001), pp. 1189–1232.

[42] Jerome H Friedman et al. “Multivariate adaptive regression splines”. In: *The annals of statistics* 19.1 (1991), pp. 1–67.

[43] Jerome H Friedman and Werner Stuetzle. “Projection pursuit regression”. In: *Journal of the American statistical Association* 76.376 (1981), pp. 817–823.

[44] Milton Friedman. “A comparison of alternative tests of significance for the problem of m rankings”. In: *The Annals of Mathematical Statistics* 11.1 (1940), pp. 86–92.

[45] Milton Friedman. “The use of ranks to avoid the assumption of normality implicit in the analysis of variance”. In: *Journal of the american statistical association* 32.200 (1937), pp. 675–701.

[46] Bogdan Gabrys and Dymitr Ruta. “Genetic algorithms in classifier fusion”. In: *Applied Soft Computing* 6.4 (2006). Application of Soft Computing in Information & Communication Technology (ICT), pp. 337–347.

[47] João Gama and Pavel Brazdil. “Cascade generalization”. In: *Machine learning* 41.3 (2000), pp. 315–343.

[48] Stuart Geman, Elie Bienenstock, and René Doursat. “Neural networks and the bias/variance dilemma”. In: *Neural computation* 4.1 (1992), pp. 1–58.

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. `http://www.deeplearningbook.org`. MIT Press, 2016.

[50] Lars Kai Hansen and Peter Salamon. “Neural Network Ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (1990), pp. 993–1001.

[51] Mark Herbster and Manfred K Warmuth. “Tracking the best expert”. In: *Machine learning* 32.2 (1998), pp. 151–178.

[52] R. Hyndman. *Time series data library*. 2017. URL: `http://data.is/TSDLdemo`.

[53] Rob Hyndman et al. *Forecasting with exponential smoothing: the state space approach*. Springer Science & Business Media, 2008.

[54] HN Johnston. “A note on the estimation and prediction inefficiency of "dynamic" estimators”. In: *International Economic Review* (1974), pp. 251–255.

[55] Matthew B Kennel, Reggie Brown, and Henry DI Abarbanel. “Determining embedding dimension for phase-space reconstruction using a geometrical construction”. In: *Physical review A* 45.6 (1992), p. 3403.

[56] Jihed Khiari et al. “MetaBags: Bagged Meta-Decision Trees for Regression”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2018, pp. 637–652.

[57] Johannes Klemming. *Python Extension for the UCR-Suite*. 2018. URL: `https://github.com/klon/ucrdtw`.

[58] Irena Koprinska, Mashud Rana, and Vassilios G Agelidis. “Yearly and seasonal models for electricity load forecasting”. In: *The 2011 International Joint Conference on Neural Networks*. IEEE. 2011, pp. 1474–1481.

[59] Anders Krogh and Jesper Vedelsby. “Neural network ensembles, cross validation, and active learning”. In: *Advances in neural information processing systems*. 1995, pp. 231–238.

[60] L. I. Kuncheva. “A theoretical study on six classifier fusion strategies”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.2 (Feb. 2002), pp. 281–286.

[61] Ludmila I Kuncheva and Christopher J Whitaker. “Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy”. In: *Machine learning* 51.2 (2003), pp. 181–207.

[62] Denis Kwiatkowski et al. “Testing the null hypothesis of stationarity against the alternative of a unit root How sure are we that economic time series have a unit root ?” In: *Journal of econometrics* 54 (1992), pp. 159–178.

[63] Christiane Lemke and Bogdan Gabrys. “Meta-learning for time series forecasting and forecast combination”. In: *Neurocomputing* 73.10-12 (2010), pp. 2006–2016.

[64] Jason Lines and Anthony Bagnall. “Time series classification with ensembles of elastic distance measures”. In: *Data Mining and Knowledge Discovery* 29.3 (May 2015), pp. 565–592.

[65] Ingo Mierswa and Katharina Morik. “Automatic feature extraction for classifying audio data”. In: *Machine Learning* 58.2-3 (2005), pp. 127–149.

[66] Luis Moreira-Matias et al. “Predicting taxi-passenger demand using streaming data”. In: *IEEE Transactions on Intelligent Transportation Systems* 14.3 (2013), pp. 1393–1402.

[67] Meinard Müller. “Dynamic time warping”. In: *Information retrieval for music and motion* (2007), pp. 69–84.

[68] Mariana Oliveira and Luis Torgo. “Ensembles for Time Series Forecasting”. In: *Asian Conference on Machine Learning*. 2015, pp. 360–370.

[69] Julio Ortega, Moshe Koppel, and Shlomo Argamon. “Arbitrating among competing classifiers using learned referees”. In: *Knowledge and Information Systems* 3.4 (2001), pp. 470–490.

[70] Ioannis Partalas, Grigorios Tsoumakas, and Ioannis P Vlahavas. “Focused Ensemble Selection: A Diversity-Based Method for Greedy Ensemble Selection.” In: *ECAI*. 2008, pp. 117–121.

[71] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[72] Fábio Pinto, Carlos Soares, and Joao Mendes-Moreira. "Chade: Metalearning with classifier chains for dynamic combination of classifiers". In: *Joint european conference on machine learning and knowledge discovery in databases.* Springer. 2016, pp. 410–425.

[73] Thanawin Rakthanmanon et al. "Searching and Mining Trillions of Time Series Subsequences under Dynamic Time Warping". In: *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining.* 2012, pp. 262–270.

[74] André Luis Debiaso Rossi et al. "MetaStream: A meta-learning based method for periodic algorithm selection in time-changing data". In: *Neurocomputing* 127 (2014), pp. 52–64.

[75] Jason Rudy. *Python package for Multivariate Adaptive Regression Splines.* 2017. URL: `https://github.com/scikit-learn-contrib/py-earth`.

[76] Amal Saadallah et al. "BRIGHT-Drift-Aware Demand Predictions for Taxi Networks". In: *IEEE Transactions on Knowledge and Data Engineering* (2018).

[77] Ismael Sánchez. "Adaptive combination of forecasts with application to wind energy". In: *International Journal of Forecasting* 24.4 (2008), pp. 679–693.

[78] Eulanda M. Dos Santos, Robert Sabourin, and Patrick Maupin. "Overfitting cautious selection of classifier ensembles with genetic algorithms". In: *Information Fusion* 10.2 (2009), pp. 150–162.

[79] Patrick Schäfer. "The BOSS is concerned with time series classification in the presence of noise". In: *Data Mining and Knowledge Discovery* 29.6 (Nov. 2015), pp. 1505–1530.

[80] Skipper Seabold and Josef Perktold. "Statsmodels: Econometric and statistical modeling with python". In: *9th Python in Science Conference.* 2010.

[81] Anderson T Sergio, Tiago PF de Lima, and Teresa B Ludermir. "Dynamic selection of forecast combiners". In: *Neurocomputing* 218 (2016), pp. 37–50.

[82] Grzegorz Skorupa. *Python package for BATS and TBATS time series forecasting.* 2018. URL: `https://github.com/intive-DataScience/tbats`.

[83] Taylor G Smith. *Python package for auto-arima.* 2018. URL: `https://github.com/tgsmith61591/pmdarima`.

[84] Floris Takens. "Detecting strange attractors in turbulence". In: *Dynamical systems and turbulence.* Springer, 1981, pp. 366–381.

[85] Allan Timmermann. "Elusive return predictability". In: *International Journal of Forecasting* 24.1 (2008), pp. 1–18.

[86] Allan Timmermann. “Forecast combinations”. In: *Handbook of economic forecasting* 1 (2006), pp. 135–196.

[87] Ljupčo Todorovski and Sašo Džeroski. “Combining classifiers with meta decision trees”. In: *Machine learning* 50.3 (2003), pp. 223–249.

[88] Robert R. Trippi and Efraim Turban, eds. *Neural Networks in Finance and Investing: Using Artificial Intelligence to Improve Real World Performance.* New York, NY, USA: McGraw-Hill, Inc., 1992.

[89] Naonori Ueda and Ryohei Nakano. “Generalization Error of Ensemble Estimators”. In: *Neural Networks, 1996., IEEE International Conference on.* xi. 1996, pp. 90–95.

[90] Xiaozhe Wang, Kate Smith-miles, and Rob Hyndman. “Rule induction for forecasting method selection : Meta-learning the characteristics of univariate time series”. In: *Neurocomputing* 72 (2009), pp. 2581–2594.

[91] Halbert White. *Artificial neural networks: approximation and learning theory.* Blackwell Publishers, Inc., 1992.

[92] Frank Wilcoxon. “Individual comparisons by ranking methods”. In: *Biometrics bulletin* 1.6 (1945), pp. 80–83.

[93] David H Wolpert. “Stacked generalization”. In: *Neural networks* 5.2 (1992), pp. 241–259.

[94] Zhi-Hua Zhou, Jianxin Wu, and Wei Tang. “Ensembling neural networks: Many could be better than all”. In: *Artificial Intelligence* 137.1 (2002), pp. 239–263.

[95] Martin Zinkevich. “Online convex programming and generalized infinitesimal gradient ascent”. In: *Proceedings of the 20th International Conference on Machine Learning (ICML-03).* 2003, pp. 928–936.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst habe und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Dortmund, den 21.03.2019

Florian Priebe