Karlsruhe Institute of Technology

Institut für Telematik
Pervasive Computing Systems / TECO
Prof. Dr.-Ing. Michael Beigl

Automatic selection of learning methods to predict time series data

Automatische Selektion von Lernverfahren zur Vorhersage von Zeitreihendaten

Master Thesis
by

Yexu Zhou

Department of Informatics

Responsible Supervisor: Prof. Dr. Michael Beigl

Supervising Staff: Dr. Till Riedel and Ms.Ploylearn Ravivanpong

Project Period: 10/01/2018 – 10/08/2018

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, den 10. 08. 2018

Acknowledgements

I would like to take this opportunity to thank all the people who contributed to this master thesis.

Firstly, I would like to thank my two supervisors Dr. Till Riedel and Ms.Ployplearn Ravivanpong, who always provided helpful advice and guidance throughout my thesis. Whenever I needed assistance of any kind, they never hesitated to help me out. For that, I am truly grateful. I would also like to thank my professor Prof.Dr. Michael Beigl.

Finally, I would like to give my special thanks to my family and friends, who lent me inspiration, moral support and affection.

Kurzfassung

Die Forschung zur Vorhersage von Zeitreihendaten ist inzwischen in beinahe allen wissenschaftlichen Zweigen vertreten. Die Auswahl des Modells ist für alle Vorhersagen ein wichtiger und notwendiger Schritt. Allerdings ist die Bestimmung des adequatesten Modells, aus einem zur Verfügung stehenden Satz von Modellen, für einen gegebenen Datensatz, immer noch eine schwierige Aufgabe. Würde man die Hyper-Parameter von Modellkandidaten in Betracht ziehen, hätte man eine riesige Anzahl an möglichen Alternativen. Die Vielzahl an Anwendungen, durch solch eine Vorhersage, führt zu einer wachsenden Nachfrage für ein System, welches automatisch ein gutes Modell mit gleichzeitig guten Parametern für eine neue Aufgabe wählen kann.

Das "combined algorithm selection and hyper-parameter optimization"-Problem wird als CASH abgekürzt. Neuerdings haben automatische Ansätze zum Lösen dieses Problems zu bedeutenden Fortschritten geführt. Die Sequential Model-based Algorithm Configuration (SMAC) Methode zeigte erfolgreiche Leistungen in einem hochdimensionalen, strukturierten, kontinuierlichen und diskreten (kategorischen) Hybridparameterraum.

In dieser Arbeit stelle ich ein Framework vor, welches durch die Anwendung von SMAC, dieses Problem in der Vorhersage von Zeitreihendaten löst. Im Gegensatz zu vorhergegangenen Arbeiten zur Vorhersage von Zeitreihendaten sind die Modellkandidaten, in meiner Arbeit, aus dem Scikit-learn regression package entnommene machine learning (ML) Modelle. Um die Leistungen der ML Modelle zu steigern, habe ich das Paket TSFRESH in meinem Framework integriert, welches die kohärenten und bekannten Eigenschaften der Datensätze extrahiert. Damit die Effizienz des Optimierungsprozesses gesteigert wird, habe ich drei Meta-learning Methoden verwendet. Diese dienen dazu Konfigurationen zu generieren, welche die Wahrscheinlichkeit besitzen, für eine neue Aufgabe am besten geeignet zu sein. Ich habe dafür ein umfangreiches Experiment durchgeführt, um den Effekt von Meta-learning zu validieren. Diese Initialisierungsmethode erzielt leichte Verbesserungen zur Beschleunigung des Optimierungsprozesses.

Abstract

Time-series forecasting is studied in nearly all fields of science and engineering. For all forecasting tasks, model selection is a necessary step. However, from a set of available models, selecting the most adequate model for a given dataset is still a difficult task. If the hyper-parameters of candidate models are taken into consideration, there would be an enormous number of possible alternatives overall. The wide range of forecasting applications led to growing demand for a system which can automatically select a good model and simultaneously set good parameters for a new task.

The combined algorithm selection and hyper-parameter optimization problem was dubbed CASH. Recently, automated approaches for solving this problem have led to substantial improvements. The Sequential Model-based Algorithm Configuration (SMAC) method showed a successful performance in a high-dimensional, structured and continuous and discrete (Categorical) hybrid parameters space.

In this thesis I am going to introduce a framework which, through the application of SMAC, deals with this problem in the context of forecasting. In contrast to previous works on time-series forecasting, the candidate models in my work are machine learning (ML) models from the Scikit-learn regression package. In order to improve the performance of ML models, I integrated the package TSFRESH in my framework, which extracts comprehensive and well-established features from time series. In order to improve the efficiency of the optimization process, I applied three meta-learning methods to generate configurations with the likelihood of performing for a new task. I conducted an extensive experiment to validate the effect of meta-learning. This initialization method yields slight improvements to expedite the optimization.

Keywords: Time-series forecasting, Meta-learning, Automated machine learning

abbreviations

ACF	autocorrelation function
AIC	Akaike's information criterion
ANN	artificial neural networks
ARIMA	Autoregressive Integrated Moving Average
AutoML	automated machine learning
CASH	Combined Algorithm Selection and Hyper-parameters optimization
EEG	Electroencephalogram
ETS	Exponential Smoothing
GARCH	Generalized Autoregressive Conditional Heteroskedasticity
ICA	independent component analysis
IHS	iterated distributed hypercube sampling
KNN	k nearest neighbours
LDA	Linear Discriminant Analysis
LHD	random Latin hypercube design
LOCF	last observation carried forward
LSTM	Long short-term memory
MAE	mean absolute error
MASE	mean absolute scaled error
MKNN	modified k nearest neighbours
ML	Machine Learning
NAN	not a number
NN	neural network
NOCB	next observation carried backward
PACF	partial autocorrelation function
PCA	principal components analysis
Prophet	Automatic Forecasting Procedure
RF	Random Forest
RID	Ridge Regression

RMH relationship between meta-features and hyper-parameters

RMSE root mean squared error

RNN recurrent neural networks

SMAC Sequential Model-based Algorithm Configuration

SMAPE symmetric means absolute percentage error

SMBO Sequential Model-based Bayesian Optimization

SVR Support Vector Regression

TSFRESH Time Series Feature extraction based on scalable hypothesis tests

XGB Extreme Gradient Boosting Regression

Contents

1. Introduction

Time-series forecasting is studied and used for several real-world problems in nearly all fields of science and engineering, such as finance, economics, meteorology, business intelligence and telecommunication [69]. Predicting future values of an observed time-series plays an important role in eliminating losses resultant from uncertainty, as well as supporting the decision-making process [67].

There are many models which can be used to forecast a time-series, including traditional statistical models and data mining models, providing many options for forecasters. However, from a set of available models, selecting the most adequate model for a given time-series is still a difficult task. If each model's hyper-parameters are considered, there would be a staggeringly large number of possible alternatives overall. Although there are now a lot of software and open sources to help people analyse and predict time-series data. Users will face two choices when using them, selecting methods and setting their parameters. In the face of these degrees of freedom, making the right choice can be vary challenging. Naturally, this poses a challenge for the data analysis community: given a dataset, the learning algorithm is automatically selected and its hyper-parameters are simultaneously set to optimize empirical performance.

Frank Hutter et al. dubbed this problem as Combined Algorithm Selection and Hyper-parameters optimization (CASH) [59] problem. They have built an automated machine learning (AutoML) framework called Auto-WEKA [58] based on Bayesian optimization method [46] for selecting a good instantiation of WEKA [17] for a given dataset. Then Matthias Feurer et al. extended this approach to a new, highly parametrized machine learning framework called Auto-Sklearn [20], which applies meta-learning [61] as warm-start to considerably improve the efficiency of the optimization process.

In this thesis, I will discuss an AutoML framework in the context of the time-series forecasting problem. I applied a variant of the Sequential Model-based Bayesian Optimization (SMBO) [28] method to select a forecasting model and simultaneously set the hyper-parameters.

In my work, the term model selection has a broad meaning. My framework will select models for all steps involved in forecasting such as data preparation, feature-extraction and pre-processing. In contrast to previous works on model selection of forecasting models, I am not going to focus on traditional statistical prediction methods. Instead, I concentrate on Machine Learning (ML) methods. Furthermore, I used all the regression models, from the open source scikit-learn [45], as the candidate models. The consideration of a large number of candidate models results in a huge hyper-parameters space. To speed up the optimization process, I tried different meta-learning methods [62] to generate good initial configurations. Lastly, I empirically studied the impact of the meta-learning as initialization for SMBO method and the forecasting performance of my framework.

In the following paragraph, I will briefly review the previous related works, then introduce the objectives, the contribution and the outline of this thesis.

1.1 Background & related work

“ Essentially, all models are wrong, but some are useful ” [36]. These famous words from George Edward Pelham Box, a British statistician, summarize the essence of model selection. There are numerous factors that affect reality. Due to the limited cognitive ability of humans or the difficulty of obtaining some data and the measurement errors of data (missing variables or inaccurate variables), the construction of the model cannot reach a degree of ”God perspective”. All models are only approximations of reality through different assumptions. The result is that there are no models that generally perform better or worse than other models when looking at all possible (time-series) datasets [51]. When facing with a new time-series data, the most practical way is to choose the most appropriate model based on the characteristics of the data, rather than developing a new model.

The force-brute and straightforward solution is to try each model with time-series data and choose the model which gets the best performance. Although simple, this solution can be very computation expensive and time consuming if there are a large number of candidate models or series to predict.

The most effective way to find the most promising model for the best performance is through the use of the experience and expert’s knowledge to detect the relation between models and time-series data that are to be predicted [65]. In other words, the idea is to use meta-features [13] of time-series to select an appropriate forecasting model.

This idea has been pursued since the 1990's. To select a candidate forecasting method, Collopy et al. [15] implemented an expert system with 99 experience rules. The rules were generated based on the time series meta-features and guidelines provided by five human experts through the analysis of real problems. Their results demonstrated that using explicit experience rules can provide more accurate forecasts. Later, Vokurka et al. [64] extended this expert system approach by extracting features automatically, yet manual review of the outputs was still required. Finally, a fully automatic expert system have been proposed [4]. They used Linear Discriminant Analysis (LDA) to select from three forecasting methods, based on 26 features. Although expert systems can supply more accurate results through developing the selection rules, the drawback is that the analysis needs to be performed by human experts, who are not efficient in analysing a large number of hypotheses that can explain results. Expert systems are also very expensive and difficult to implement.

To minimize the difficulty, ML algorithms may be a promising way to automatically acquire knowledge for model selection. "Automatic knowledge acquisition mode " [55] is dubbed as meta-learning. Prudencio et al. [47] was the first to introduce meta-learning into time-series forecasting. They applied a C4.5 Decision Tree [49] to relate 6 meta-features to the performance of 2 forecasting models. They have also used the NOEMON [34] approach to rank 3 methods. A more comprehensive framework was presented by Wang et al. [65], where time-series are clustered according 16 meta features and rules generated using a decision tree. Recently, Rob J Hyndman et al. [54] have presented a general framework where a Random Forest (RF) is used to identify the best forecasting method using only time-series features.

Table 1.1 summarizes some facts about these past works. It can be seen that all previous works treated meta-learning as a classification problem. They separately tackled the problems of model selection and hyper-parameters setting. And most candidate models are statistical models, such as Autoregressive Integrated Moving Average (ARIMA) model and Exponential Smoothing (ETS) model.

1.2 Objectives

The purpose of my work is to discuss various aspects of the AutoML process that should be considered when addressing the time-series forecasting problem in particular. In summary, after reviewing prior works, I have summarized the following ideas which I considered worth trying.

(1) Time-series are studied in various fields, such as economics, medicine and biochemistry. Across different scientific fields, there are many discipline-specific time-series analysis methods. Which meta-features should be used? If a more comprehensive set of meta-features is used, can it improve the learning of the relationship between datasets and forecasting models or datasets and hyper-parameters?

Year	Authous	Features	Meta-Learning Methods	Time Series	Model pool
1992	Collopy et al. [15]	18	Rule based Expert System	126 (M1)	exp. smoothing (Holt and Brown), random walk, linear regression
1996	Vokurka et al. [64]	5	Rule based Expert System (partly automatic)	126 (M1)	exp. Smoothing (single and Gardner), structural and a combination of the three
1997	Shah [4]	26	Discriminant Analysis	203 (M1)	exp. smoothing (single and Holt-Winter), structural
2004	Prudencio et al. [47]	6/7	Decision Tree / NOEMON	99/645 (M3)	exp. smoothing, neural network / random walk, Holt's smoothing, auto-regressive
2009	Wang et al. [65]	9	Decision Tree	315	random walk, smoothing, ARIMA, neural network
2018	Hyndman et al. [54]	33	Random Forest	M1, M3	Random walk, theta model, ETS and ETS's variant, ARIMA and ARIMA's variant
2018	Yexu Zhou	794	SMBO with help of meta-learning methods	Modified datasets	All Machine Learning models in Scikit Regression Package

Table 1.1: overview of past works of time-series model selection. The last row of the blue font in this table represents my work. Compared to previous work, I used more meta-features to facilitate the selection of algorithms. The modified datasets used and the framework with different meta-learning methods will be introduced in chapter 3.

(2) Previous works basically focused on selecting models between statistical forecasting models. Nowadays, ML models have drawn more attention in the forecasting community [44]. Can the forecasting performance, by selecting between ML regression models, be as good or even better than the forecasting performance of statistical forecasting models?

(3) Previous works separately tackled the problems of model selection and hyper-parameters configuration. They functioned like a classifier, choosing the most appropriate model based on the meta-features. This process is not suitable for the selection between ML models, because it has been established, that the performance of some ML models, such as non-linear Support Vector Regression (SVR), relies heavily on hyper-parameter optimization [31]. The selection between ML models, merely by means of their respective default hyper-parameters, is unfair. If the problem is addressed simultaneously, can it provide accurate predictions more effectively?

(4) Taking all hyper-parameters of all candidate models into consideration, the parameter space, which is going to be optimized, is staggeringly huge. Is meta-learning an effective way to accelerate the optimization process?

1.3 Contribution

In this section I want to explain the contribution of my work. It can be divided into the following four points.

First, I integrated the Time Series Feature extraction based on scalable hypothesis tests (TSFRESH) package [14] in my framework. TSFRESH is a package which extracts comprehensive characteristics from time-series. These features are used in two tasks: as meta-features for meta-learning and as the input for forecasting models. I labelled the property of each characteristic of TSFRESH, so that the appropriate set of features could be easily filtered, depending on a given task. The results from the experiments, which are described in section 4.1 and 4.2, indicate that the features from TSFRESH can effectively capture the underlying dynamic mechanisms of time-series.

Second, the candidate forecasting models are different. My work focused on the selection between ML models. All the ML models I used are taken from the scikit-learn regression package. I have adapted these ML models to time-series forecasting by applying the sliding window method. In section 4.4, I compared the forecasting performance of my framework with other classic forecasting methods, such as ARIMA and Neural Networks. The result showed, that applying ML models seems to be a promising way for forecasting problems.

Third, by applying the Sequential Model-based Algorithm Configuration (SMAC) model [26], the model selection problem and the model configuration problem are simultaneously tackled. I extended the Auto-Sklearn [20] framework to the time-series forecasting problem. For every step involved in the data mining pipeline, in the context of time-series, I included time-series specific processing methods. In section 3.2, the structure of my framework is explained in detail.

Fourth, I tried three different meta-learning methods to accelerate the optimization process. Regarding meta-learning, I used more comprehensive meta-features than previous works. The results from the experiments in section 4.2 showed, that meta-learning is a good complement to the Bayesian optimization framework. It can effectively accelerate the optimization process.

1.4 Outline

The remainder of this thesis is organized as follows:

Chapter two starts with the theoretical basis for my thesis. Moreover, I conducted an experiment on the "other" datasets of M3-competition [38] to pre-study the forecasting performance of the combination of ML models and SMAC model. With the help of this experiment, the fundamentals for my work can be presented and understood in an easier way. This experiment also helps to explain the ideas which are presented in the next chapter.

In chapter three, I am going to discuss the structure and process of my automated ML framework for the forecasting. Inspired by previous experiments, I tried to use meta-learning to accelerate the optimization and to extract more features as inputs for forecasting models. The configuration space for the optimization is detailed at the end of the chapter.

In chapter four, I am going to introduce the experiments, which were conducted in order to study the performance of applied meta-features, as well as meta-learning methods as a warm-start for SMAC model and the forecasting of my framework.

In chapter five, I am going to draw conclusions for my thesis and I will be proposing some suggestions for future works.

2. Theoretical foundation

This chapter introduces the theoretical foundation for my work. Section 2.1 describes how to formalize a forecasting task as a supervised learning problem. Section 2.2 briefly introduces the CASH problem and the tree-based SMAC model for tackling it. In section 2.3 I demonstrate the result of an empirical experiment on M3-competition to pre-study the forecasting performance of the combination of ML models and the SMAC model. Based on this experiment, I have summarized two ideas for improving the performance of ML models in forecasting problem. The first is that because of the flexibility of the ML models, more different features can be added to help to improve the performance. The second is that using meta-learning is a promising way to avoid unnecessary exploration process in the optimization. The implementation of these two ideas are going to be introduced in detail in next chapter. In section 2.4 the foundation of meta-learning is introduced.

2.1 Time-series as supervised learning

Classical statistical prediction models have been widely used for a long time [10], such as ETS models and ARIMA models. But it is becoming increasingly clear that linear models are not suitable for many real applications [16]. Although many non-linear forecasting models were proposed, such as Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model [18], the analytical study of non-linear forecasting models compared to linear forecasting models is still in its infancy [16]. Recently, ML models have attracted more attention and have become important competitors to classical statistical models in the forecasting community [2].

ML models which are also called data-driven models use only historical data to learn the dependency between past and future. Compared to traditional statistical models, ML models have the following advantages. They make fewer assumptions

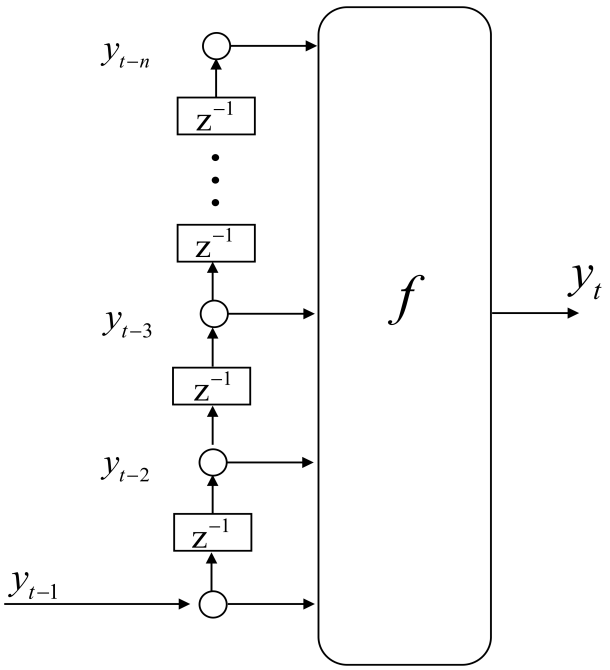


Figure 2.1: f is a function of w previous values. It returns the prediction value of the time-series at the time $t+1$. Here z^{-1} is a lag operator, i.e. $y_{t-1} = z^{-1}(y_t)$.

and have on-line learning capability. They can also deal with time-series dataset where the stochastic process underlying the data is non-stationary. That is the reason why I focused on selection between ML models in my work. The next step introduces the way to apply the ML methods to the time-series forecasting problem.

Before ML models can be used, time-series forecasting problems must be re-framed as supervised learning problems. Supervised learning is a machine learning task of learning a function that maps an input to an output based on example input-output pairs, which are typically referred to as training set. But time-series data is observations with a sequential structure. What needs to be done now, is to convert the sequence data to pairs of input and output sequences by using previous w time steps as input variables and using the next time step value as the output variable. This mechanism is illustrated in figure 2.1.

The use of values from previous time steps to predict the value of the next time step is generally called sliding window method. In time-series analysis and statistics, this method can also be called lag method [29]. The number of prior time steps w is called window width or the size of lag. This sliding window method is the basis for how we can turn a time-series into a supervised learning problem. If there is a time-series with N observations, a training set can be got as equation 2.1 by sliding the window with size w along the time axis. This re-framing of time-series data gives access to the suite of standard linear and non-linear ML models.

$$\mathrm{Y} = \begin{bmatrix} y_N \\ y_{N-1} \\ \vdots \\ y_{n+1} \end{bmatrix} \quad \mathrm{X} = \begin{bmatrix} y_{N-1} & y_{N-2} & \cdots & y_{N-n-1} \\ y_{N-2} & y_{N-1} & \cdots & y_{N-1} \\ \vdots & \vdots & \vdots & \vdots \\ y_n & y_{n-1} & \cdots & y_1 \end{bmatrix} \tag{2.1}$$

But the sliding window method re-frames only one-step forecasting problem as supervised learning problems. Another important aspect of the forecasting problem is multi-steps time series forecasting, where the size of the forecasting horizon is greater than 1. It means that the forecasting task consists of predicting next h values of a time-series. To tackle this problem, there are three main strategies, namely recursive, direct and multi-output strategies [12].

The direct strategy involves learning a separate model for each forecast time step. Learning a model for each time step is very computational expensive and a maintenance burden, especially when the forecasting horizon is very wide. Moreover, another disadvantage of using the direct strategy is that there is no opportunity to learn the relationship between predictions. Because separate models are used.

Multiple output strategy involves developing only one model that is capable of predicting the entire forecast horizontal sequence. This strategy adopts multi-output techniques, where the output value is not only a scalar but a vector of future values of time series. Multiple output models are complex because the model should learn the dependence structure between historical values and future predicted values as well as the dependence structure between future predicted values.

The recursive strategy consists of using a one-step model multiple times where the prediction for the prior time step is used as an input for making a prediction on the following time step. It is important to mention that predictions are used instead of observations, so that this recursive strategy allows prediction errors to accumulate. This may cause the performance to quickly degrade as the prediction time horizon increases.

2.2 CASH problem and SMAC model

In this section I would like to briefly review that the task of AutoML is considered the same as the CASH problem [58]. Later on, I am going to elaborate on existing mechanisms to deal with the CASH problem.

There are two important main problems in AutoML:

(1) model selection

(2) hyper-parameter optimization.

The goal of model selection is to find the best model A^* from a set of candidate models A. A^* is the model which gets an optimal generalization performance over a training set D. Generalization performance is usually estimated by using k-fold cross-validation. In k-fold cross-validation, the training set D is randomly split into k equal sized partitions $(D^{(1)}, D^{(2)}, \cdots, D^{(k)})$. Out of the k partitions, a single partition $D^{(i)}_{valid}$ is retained as a validation data for testing the model and the remaining $k-1$ partitions $D^{(i)}_{train} = D/\, D^{(i)}_{train} \quad (i = 1, 2 \cdots, k)$ are used as training data to train the model. The cross-validation process is then repeated k times, with each of the k partitions used exactly once as the validation data. At last, the k results can be averaged to produce a simple estimation for generalization performance. Formally, model selection can be written as equation 2.2 [58].

$$A^* = \arg\min_{a \in A} \frac{1}{k} \sum_{i=1}^{k} L(a, D^{(i)}_{train}, D^{(i)}_{valid}) \tag{2.2}$$

Where L is a loss function achieved when model a is trained on $D^{(i)}_{train}$ and evaluated on $D^{(i)}_{valid}$.

The concept of the hyper-parameter optimization is similar to the concept of model selection. Given a model a, the purpose of hyper-parameter optimization is to find a set hyper-parameter $\lambda \in \Lambda$ (hyper-parameters space) that causes the model to get the best generalization performance over the training set D. The generalization performance is also estimated using k-fold cross-validation. It can be written as in equation 2.2 [58].

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \frac{1}{k} \sum_{i=1}^{k} L(a_\lambda, D^{(i)}_{train}, D^{(i)}_{valid}) \tag{2.3}$$

The hyper-parameter optimization problem is more complicated, because the optimization space is continuous and high dimensional. Moreover, there is dependency structure between the hyper-parameters. For example when using SVR method, the parameter *degree* is valid only when the *kernel* parameter takes value "poly". This situation can be considered as hyper-parameter i being conditional on another hyper-parameter j.

There have been a lot of past works addressing these two problems separately [1, 11, 68, 23, 8]. But as already mentioned, the performance of some ML algorithms strongly depends on the hyper-parameters optimization. If dealing with these two issues separately, the extent of automatic machine learning could be limited. That is why it is necessary to consider and deal with both problems at same time. Hutter et al. defined it as CASH [59] as written in equation 2.2.

$$\lambda^* A^* = \arg\min_{a^{(j)} \in A, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^{k} L(a^{(j)}_\lambda, D^{(i)}_{train}, D^{(i)}_{valid}) \tag{2.4}$$

where $a^{(j)}$ is a model of the set of candidates models A with its respective hyper-parameters spaces $\Lambda^{(j)}$.

The CASH problem can be viewed as a hierarchical hyper-parameters optimization problem when the choice of models itself is considered as a hyper-parameter. After the reformulation of the hyper-parameters, space is extended to $\Lambda = \Lambda^{(1)} \cup \Lambda^{(2)} \cup \cdots \cup \Lambda^{(n)} \cup \{\lambda_r\}$, where λ_r is a root level hyper-parameter that chooses between candidate models. The respective hyper-parameter space $\Lambda^{(j)}$ is conditional on the root level hyper-parameter λ_r.

Now the CASH problem turns back to be a hyper-parameter optimization problem. A promising approach to tackle this problem is the SMBO model. SMBO is a stochastic optimization framework which can work with continuous and categorical hyper-parameters as well as conditional structure of parameters. The general process of the SMBO is outlined in algorithm 2.2.

Algorithm 1 SMBO

1: initialise model M_L;
2: **while** time budget or evaluation budget for optimization has not been exhausted **do**
3: λ_{t+1}: candidate configuration from M_L. (Intensification)
4: Compute $c = L(A_{\lambda_{t+1}}, D^{(i)}_{train}, D^{(i)}_{valid})$. (Evaluation)
5: Update M_L with (λ_{t+1}, c). (Refitting)
6: **end while**
7: **return** λ with optimal c;

Suppose there is a minimizing problem. First of all, at initialization step, configurations are randomly sampled and a response surface model M_L is built to understand the relationship between hyper-parameters and the measured performance (Loss function value). Then it iterates in the following three steps:(1) Intensification. Maximizing an acquisition function based on model M_L to guide the next location $\lambda_{t+1} \in \Lambda$ to sample that is the most promising to be the optimum. Acquisition functions are typically defined as followed: High acquisition corresponds to potentially optimal values of the loss function, either due to the prediction being low, or the uncertainty being great, or both. Maximizing an acquisition function represents an automatic trade-off between exploration (where the loss function is very uncertain) and exploitation (where the loss function is expected to be high). (2) Evaluation. Evaluating the loss value c of λ_{t+1}. (3) Refitting. Updating the model M_L with the new configuration (λ, c). This technique has the advantageous property that it aims to minimize the number of objective function evaluations [9].

There are several existing SMBO model variants. In this thesis, I used the SMAC model to solve the CASH problem in the time-series forecasting problem. The main difference between existing SMBO models is the model class they use, e.g. Gaussian

process model, random forest model, etc. SMAC, a more sophisticated instantiation of the general SMBO framework, is based on random forest. Many experiments have produced evidence that this tree-based SMAC performs with a lot of success in regard to problems with high-dimensional, structured, continuous and discrete (categorical) hyper-parameter space [19].

In case of inquiry for more details of the SMAC model and the CASH problem, I would like to refer to [27, 19, 58, 28, 26].

2.3 The pre-study of the forecasting performance of the combination of ML models and SMAC optimization

In this section, I present a simple experimental study I have conducted on the datasets of the M3-Competition, to pre-study the forecasting performance of ML models cooperated with the SMAC optimization model. This experiment gave me some new ideas to improve the performance. The set-up of this experiment is the initial version of my more sophisticated follow-up experiment, which makes the following work more understandable.

2.3.1 Experiment set-up

I applied the SMAC method in order to choose between three regression models from the Scikit-learn package on the "other" time-series of the M3-Competition. The time-series of the M3-competition is often used as a benchmark for evaluating strategies of model selection [47, 15, 52]. The included three regression models are SVR model, Extreme Gradient Boosting Regression (XGB) model and Ridge Regression (RID) model. The corresponding hyper-parameters space is illustrated in figure 2.2.

By using the slide window method, ML models can be adapted to the forecasting problem. The window size is therefore also considered as a hyper-parameter. There are 17 hyper-parameters in total. All hyper-parameters and respective range are detailed in table 2.1.

In the early stages of a forecasting project, four decisions need to be made: forecasting datasets, forecasting horizon, forecasting strategy and evaluation metrics. As early mentioned, I experimented on the "other" time-series data of the M3-Competition, which consists of 174 time-series derived from different branches. To compare my results with the benchmark, I used the symmetric means absolute percentage error (SMAPE) as evaluation metric, which is also utilized in the M3-Competition. The SMAPE is written as in equation 2.3.1.

$$\sum_{i=1}^{h} \frac{|x_i - f_i|}{(x_i + f_i)/2} * 100 \tag{2.5}$$

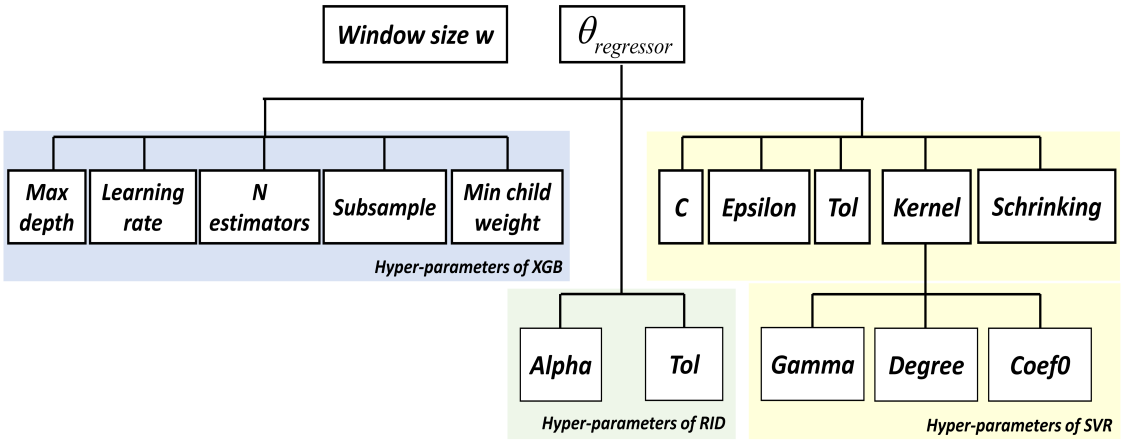


Figure 2.2: Two top-level hyper-parameters: window size and $\theta_{regressor}$. Top-level hyper-parameter $\theta_{regressor}$ is for choosing between 3 regression models. All hyper-parameters except these two are conditional. Hyper-parameters in leaf node are conditional on hyper-parameters in root node. For the specific meaning of these parameters, please refer to the Scikit package.

Component	Hyper-parameter	Values	Type
Root	$\theta_{regressor}$	"SVR", "XGB", "RID"	categorical
Root	window size	[1, 30]	int
SVR	C	[0.01, 32768]	float
SVR	epsilon	[0.001, 1]	float
SVR	kernel	"linear", "poly", "rbf", "sigmoid"	categorical
SVR	degree	[2, 5]	int
SVR	gamma	[0.0001, 8]	float
SVR	coef0	[-1, 1]	float
SVR	shrinking	"True", "False"	categorical
SVR	tol	[0.00001, 0.1]	float
RID	alpha	[0.00001, 10]	float
RID	tol	[0.00001, 0.1]	float
XGB	max depth	[1, 10]	int
XGB	learning rate	[0.01, 1]	float
XGB	n estimators	[50, 500]	int
XGB	min child weight	[1, 20]	int
XGB	subsample	[0.01, 1]	float

Table 2.1: Hyper-parameters space in this experiment. The value column describes the range of each parameter. For the specific meaning of these parameters, please refer to the Scikit package.

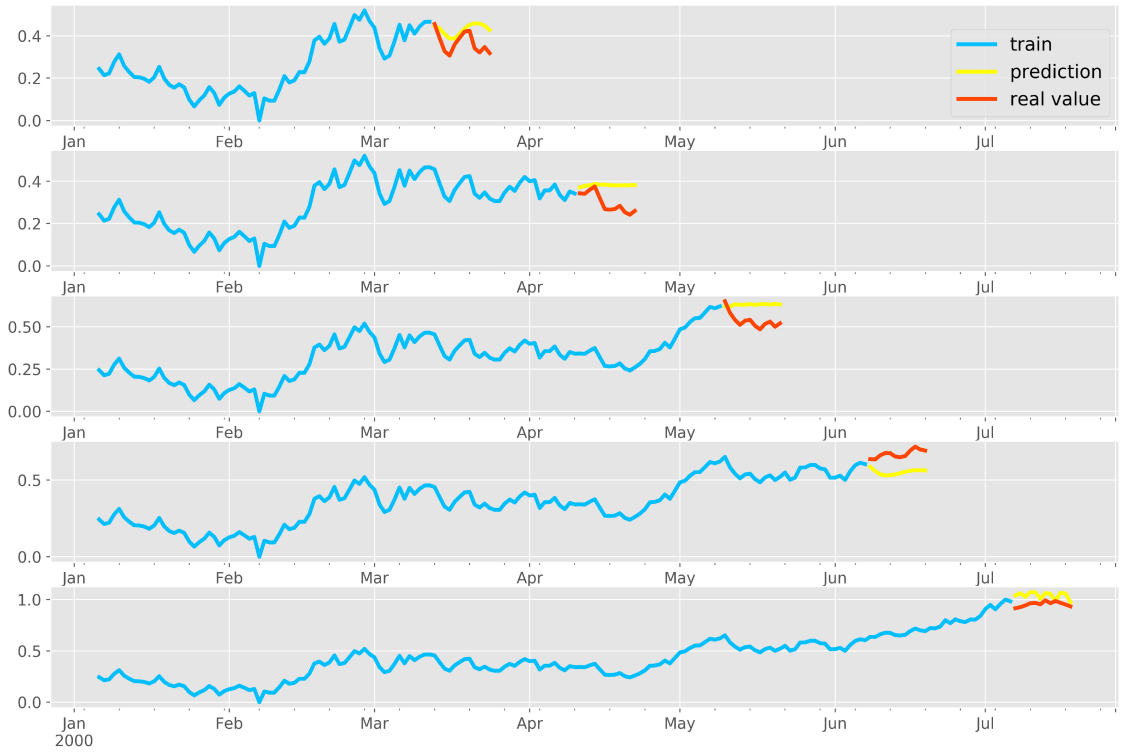


Figure 2.3: This figure illustrates the procedure of the time-series k-fold walk-forward cross-validation, where k=5.

where h is the forecasting horizon, x_i the real value and f_i the forecast. M3-Competition stipulated that the forecasting horizon is eight steps. I used this forecasting horizon for training and test split. The last eight observations as a test set and the previous observations as a training set. The test set can never be seen within the optimization process. It was used only once after the training to evaluate the models found. For Multi-Step forecasting, direct strategy was applied.

As we know, training residuals are not a reliable indication of how large true forecast errors are likely to be. So I applied 10-fold cross-validation to estimate the generalization forecasting performance. But the traditional cross-validation method cannot be directly used with time-series data, because it assumes that there is no relationship between observations and that each observation is independent. The time dependency structure of time-series must not be randomly split into groups. I applied walk-forward 10-folds cross-validation in this experiment. The corresponding training set only consists of observations that occurred prior to the observation that forms the test set. Thus, no future observations can be used in constructing the forecast. Since it is not possible to obtain a reliable forecast based on a small training set, the earliest observations are not considered as test sets. Figure 2.3 illustrates the process of walk-forward cross-validation , where the blue observations form the training sets, the red observations form the test sets and yellow points represent the forecasting values.

In the experiment I executed the SMAC optimization procedure once for each time-series. For each optimization procedure it performed 200 evaluations. The result for each time-series is the evaluation performance on the test set.

2.3.2 Experiment result

Method	F-horizon	SMAPE	Method	F-horizon	SMAPE
ARARMA	8	4.38	Winter	8	4.81
Autobox2	8	4.41	Holt	8	4.81
Theta	8	4.41	SmartFcs	8	4.86
CASH-TS	8	**4.49**	Flores/Pearce2	8	4.89
Comb S-H-D	8	4.56	Autobox1	8	4.93
Robust-Trend	8	4.58	Theta-sm	8	4.93
ForecastPro	8	4.6	B–J automatic	8	5.06
Dampen	8	4.61	Flores/Pearce1	8	5.09
PP-autocast	8	4.62	**SMAC-XGB**	8	**5.52**
ForecastX	8	4.64	RBF	8	5.6
Autobox3	8	4.71	**SMAC-SVR**	8	**6.1**
SMAC-RID	8	**4.72**	Single	8	6.29
Automat ANN	8	4.8	Naıve2	8	6.3

Table 2.2: Average SMAPE on the ”other” dataset of M3-competition. The results with blue font are from my experiment. The rest of the results were published by M3-competition [38].

Table 2.2 shows the results. I referred this experiment as CASH-TS. As one can derive from the table, compared with the classical statistical forecasting model, CASH-TS has also achieved a good performance. It was the fourth best performance. The purpose of this experiment was not to try to prove that the ML model is better than the statistical forecasting model, but to show the potential of the ML models cooperated with the SMAC model on the time-series forecasting problem. The ML model has a lot of room for improvement in forecasting.

In order to verify whether the three algorithms have been effectively selected, I have applied SMAC separately to optimize these three methods, which are referred to as SMAC-SVR, SMAC-RID and SMAC-XGB in table 2.2. If these three algorithms have been effectively selected, the performance of CASH-TS should be better than the performance of the three algorithms. The result is consistent with this assumption as shown in table 2.2. The performance of CASH-TS is indeed better than the three algorithms.

To make this more apparent, I screened several time-series in which the performance of the three regression models varied widely. One line in figure 2.4 represents the performance of a regression model on the selected time series. The black circles represent models chosen by CASH-TS. In most cases, CASH-TS has made a appropriate choice.

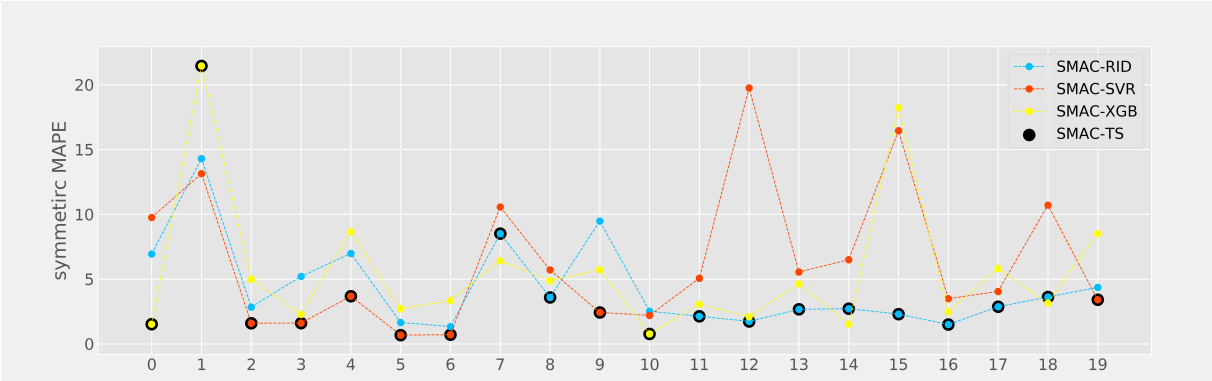


Figure 2.4: The y-axis represent the evaluation performance on test set. The black dots are the selection of SMAC-TS.

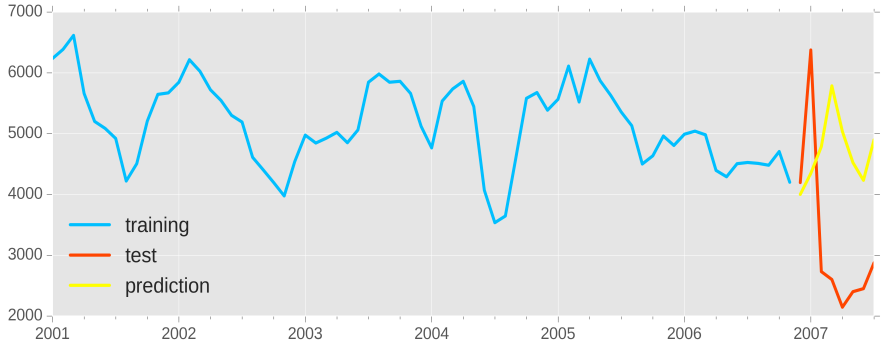


Figure 2.5: In this time-series, CASH-TS has chosen the model which performed best on the training set but worst on the test set. Because curve of last eight steps is completely different from the previous observations.

I labelled each time-series with the best model, meaning the model with the best performance on the test set, from the three regression models. In this experiment, I observed a number of 108 correctly chosen examples from the set of 174 examples, which means that CASH-TS correctly chose 62% of all datasets. One reason for the selection error is that SMAC makes its choice based on the performance on the training set. The cross-validation error is only an estimate of the generalization performance. Another reason is that for each optimization procedure, only 200 configurations are evaluated, which is not enough for a large hyper-parameters space. SMAC needs more evaluations to find the high-performance area.

There were some situations where CASH-TS chose the worst model, such as the time-series shown in figure 2.5. In the last eight steps, its curve is completely different from the previous data. This selection error is understandable, due to the randomness of the time-series being large.

2.3.3 Summary

Based on the experiments from the previous section, we summarized the following two ideas with the ability to improve the ML model's performance in forecasting:

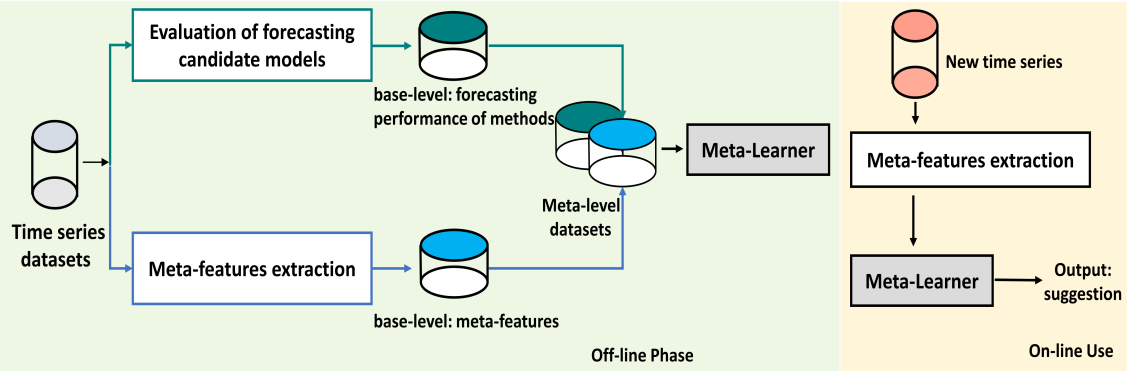


Figure 2.6: The framework of meta-learning.

(1) The ML model is highly flexible. It can handle high-dimensional, different types of features, such as discrete, continuous and categorical. In the traditional statistical methods and in the experiments of previous section, only past values were used as features. it is worth trying to extract a set of comprehensive features as input.

(2) One drawback of SMAC is its slow start when dealing with a large hyper-parameters space. It needs a number of evaluations to find the optimum region. Meta-learning, which acquires knowledge from past experience, can suggest instantiations that are likely to perform well on a new dataset. Using meta-learning is likely to accelerate the optimization process. Meta-learning can also make full use of the test performance. It may be a perfect complementary to the Bayesian optimization framework.

The realization of these two ideas will be introduced in the chapter 3. The following section discusses the basic foundation of meta-learning.

2.4 Meta-learning

Meta-learning was proposed to support data mining tasks and focuses on discovering the relationship between datasets and learning models [63]. Figure 2.6 presents a general type of architecture of meta-learning adapted for the selection of forecasting methods. The architecture has two phases: off-line training and on-line use. Next I will explain these two phases in the context of time-series forecasting problem.

The task of the off-line training phase is to build a meta-learner by acquiring knowledge from the meta-level dataset. This knowledge from the meta-level dataset associates time-series meta-features to the performance of candidate models. More specifically, an example of time-series is used as inputs for two processes, namely evaluation of forecasting candidate models and meta-features extraction. From these two processes, two sets of data are obtained. One is "base-level: forecasting performance of methods" and the other "base-level: meta-features", which are then combined to form the next dataset called "meta-level dataset". Subsequently, a

meta-leaner is built on this meta-level dataset to identify the relationships between forecasting methods and time series meta-features.

In the on-line use phase, given a new time-series, the meta-features extraction process calculates the values of the time-series meta-features. According to these values, the meta-learner can suggest a suitable candidate model.

In order to build a meta-learning system, according to the aforementioned architecture, four important issues should be considered:

(1) The time-series datasets should have enough diversity, because users may be interested in different domains, such as finance, traffic, weather, manufacture, etc.. To help meta-leaner better understand the relationship between meta-features and the corresponding performance of candidate models, the diversity of the time-series example is a key component.

(2) The question of which candidate models should be considered. The models should be well-established and widely used in practice. Models should contain representatives of different families of models. Unlike previous works, which addressed the model selection of time-series forecasting, my work focuses on ML models.

(3) What operations for meta-features extraction should be used to describe time-series. An operation is an algorithm that summarizes a time-series into a single real number or multiple real numbers. Operations should focus on the dynamical characteristics of time series. There are many diverse operations which quantify various time-series properties including basic statistic of distribution, linear correlations, information theoretic, non-linear analysis, frequent domain analysis, etc..

(4) This is the most important issue: which meta-learning approach should be used. This choice depends on the user's requirement. Usually, it can be considered a classification problem, which means that the meta-learning approach can be a classifier. It only suggests the best model for the time-series. Alternatively, users may be interested in a more informative solution, in which case zoomed-ranking ML models can be used, to provide a ranking of candidate models.

3. Design and Implementation

Based on the experience from the experiment in section 2.3, I have built a more complicated and comprehensive automated framework for time-series forecasting, in which only ML regression models are being considered. The general structure is illustrated in figure 3.1. The structure consists of three levels: Top level being meta-learning, the middle level the SMAC model and the bottom level the data mining pipeline for forecasting.

First of all, based on the meta-features of the input time-series, the top level generates good configurations for the middle level. Then the middle level SMAC builds a surface model based on the generated configurations and gives model and parameter suggestions to the bottom level. Receiving the suggestions from the middle level, datasets will be analysed in the bottom level. After the analysis, the bottom level gives the performance back to the middle level. Then the process will be iterative between the middle level and the bottom level.

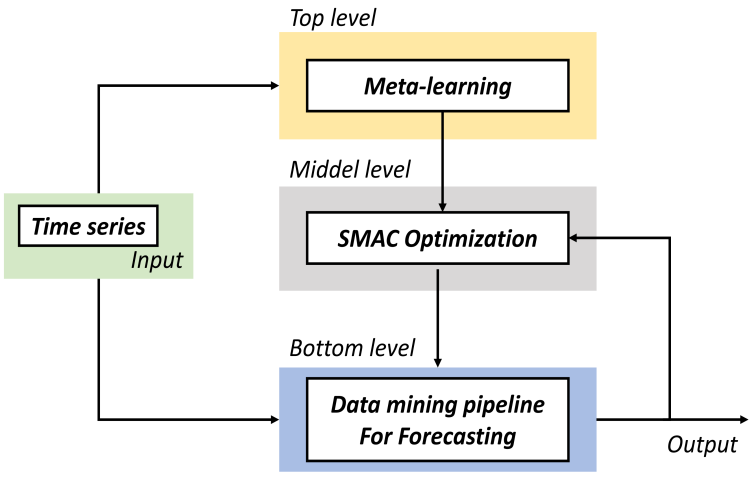


Figure 3.1: The structure of my automated framework.

In this chapter I will introduce my framework from the following two parts.

Section 3.1 introduces the meta-learning system which are built in my framework in order to accelerate the optimization process.

Section 3.2 introduces the included options for each step involved in the data mining process of time-series forecasting tasks. The blue part of figure 3.1 stands for a general process of data mining. For each step, there are many different approaches. My automated framework considers not only the selection of forecasting models, but also the selection for every other steps. For example, during the data preparation step, missing values should be handled. If there are missing values in time-series, optional approaches are mean imputation, last observation carried forward (LOCF) imputation, linear or spine interpolation, moving average etc. [42].

3.1 Meta-learning as warm starting

When dealing with ML problems, flexibility is very important. Since each learning algorithm is based on a set of assumptions on the dataset, it is inductive biased. This means that it will only learn well if the bias matches the learning assumptions. A learning algorithm may perform very well in one domain, but not in another. Low flexibility poses strong restrictions on the performance of the ML techniques. Meta-learning was proposed to support ML tasks and to improve the performance and flexibility of existing learning algorithms. By relating meta-features of datasets to the performance of candidate learning algorithms, meta-learning tries to understand the relationship between the learning problem (datasets) and the effectiveness of candidate learning algorithms.

To build a meta-learning system in context of time-series forecasting task, four important factors should be considered:

(1) Meta-features, used to describe time-series

(2) Time-series (datasets), used for building the meta-learning system

(3) meta-learner, used to learn the relationship

(4) models, considered as candidates.

As mentioned earlier, candidate models are all regression models from the scikit-learn package. Next, (1) (2) (3) will be discussed in detail.

3.1.1 Meta-features for meta-learning

Meta-features extraction is a critical step for meta-learning. It's performance directly affects whether meta-learner can capture the relationship between datasets

and algorithms in a more effective manner. The meta-features should efficiently capture the structure and property of the dataset.

There are many previous works illustrated in table 1.1, addressing meta-learning for time-series. While these approaches achieved great results, they are extremely limited to very few meta-features. Those few meta-features cannot deal with the high dimensional configuration spaces faced in AutoML [22]. To find a more comprehensive and effective set of meta-features, I did a lot of research work.

Time-series are fundamental data objects studied across the scientific disciplines such as measurements of vibration in industry science, stock prices in finance and air temperature in climate science [24]. To understand the underlying mechanism of time-series datasets, scientists have developed many different techniques: from various summary statistics to time-series models (as landmarking feature). Fulcher and Jones et al.[22] have constructed a reduced representation of time-series that efficiently capture time-series' dynamical properties.

They collected and analysed 9316 interdisciplinary analysis algorithms to form this reduced representation. These 9316 different algorithms from all domains cover almost all analysis features of time-series. This set of features is accurate enough to represent time-series. From the perspective of calculation time and practicality, it is not necessary to calculate all features. There exists redundancy in all analysis algorithms where outputs of algorithms are highly correlated over a set of time-series. To answer the question of how many analysis algorithms are required to summarize the dynamic structure and properties of time-series effectively, they addressed this question by clustering the 9613 interdisciplinary algorithms. In their work, each analysis algorithm was implemented as an operation that summarizes a time-series as a single real number. And each operation was represented as a feature vector containing its outputs across a diverse time-series dataset, as shown in figure 3.2.

Operations should be classified as similar when they are highly correlated or anti-correlated. To quantify distances between operations, an absolute linear correlation distance metric was used, which determines operations to be similar when outputs are either highly correlated or highly anti-correlated across a time-series dataset.

After all operations were presented by vectors, k-medoids clustering was applied to isolate a reduced set of k operations. The quality of the reduced set of operations is quantified with the residual variance measure $\mathrm{V}_{residual}$ [57] written in equation 3.1.1

$$\mathrm{V}_{residual} = 1 - R(S_{full}, S_{reduction}) \tag{3.1}$$

where S_{full} is a matrix of all pairwise distances between time-series characterized in full feature (all operations) space. $S_{reduction}$ is the same matrix, but characterized in the reduced feature space. R is a correlation coefficient of these two distances

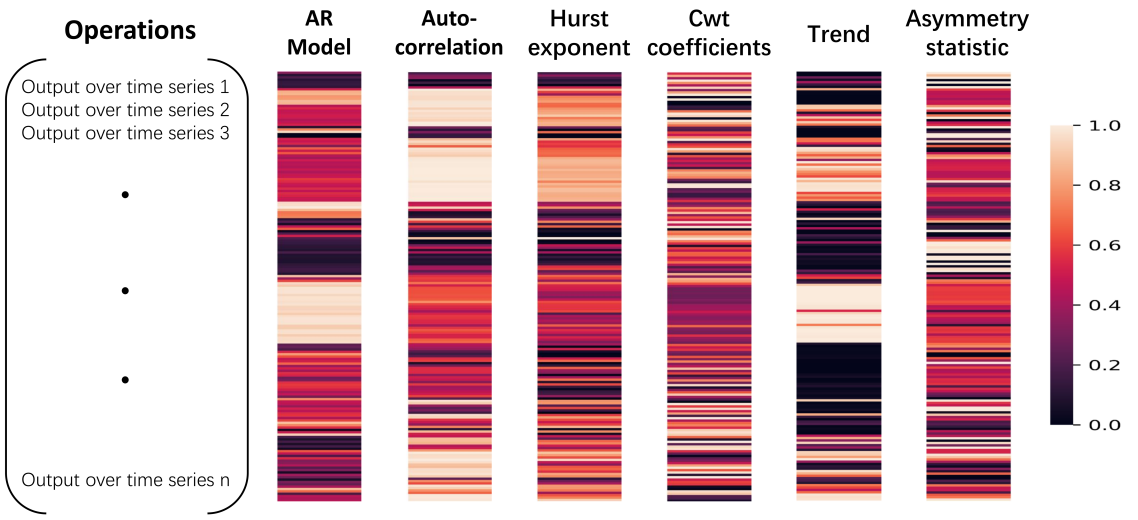


Figure 3.2: An operation is represented by feature vectors consisting of its outputs across time-series datasets. The value of a output is visualized by using color: from dark (0.0) to bright (1.0). I used a set of 187 time-series from interdisciplinary domains to illustrate this.

matrix. Perfect linear correlation results in $R \approx 1$ and $V \approx 0$, which also means an effective dimensionality reduction.

By applying k-medoids clustering from a range of 2 to 2000 clusters, Fulcher and Jones et al. found a reduced set of 200 operations which is a good approximation to all 9613 operations with $V_{residual}$ of just 0.05. This set of operations is developed from different scientific disciplines, such as stationary, correlation dimension, autocorrelation, auto mutual information and wavelet transforms etc.

Later, motivated by industrial applications for ML models, Christ et al. extended the approach of Fulcher et al. and proposed a framework called TSFRESH [14], which characterizes time-series with comprehensive and well-established features describing meta-information.

In my work, I used the TSFRESH package to extract meta-features for time-series directly. Meta-features should focus on the dynamical properties of time-series and not their length or location. I studied the properties of each feature in TSFRESH. I labelled operations which are length-dependent or location-dependent, so that an appropriate set of operations can be quickly filtered, given a specific time-series analysis task such as clustering. These labels were determined by evaluating each operation on a set of time-series. By applying each operation to a set of time-series with different length, operations whose outputs were correlated with length of time-series were labelled as length-dependent. The test for location-dependent was similar. I evaluated each operations on a set of time-series with different mean. Operations whose outputs varied strongly, were labelled as location-dependent.

In addition, after summarizing the previous work of meta-learning for time-series datasets, I added several custom features to TSFRESH. All operations used for meta-features extraction are described in appendix A.

3.1.2 Time-series datasets for meta-learning

Datasets is another basic and critical part for meta-learning. To help meta-learner better understand the relationship between datasets and candidate ML models, datasets should have enough diversity. Most of the previous work of meta-learning on time-series did not focus on this point. They did not check the diversity of datasets. As demonstrated in table 1.1, they applied the datasets directly from a forecasting competition to build a meta-learning system.

The problem here is that most time-series of a competition datasets are measured from a few sources only, where many time-series look very similar (have similar structure). Similar datasets can only help to slightly improve the understanding of local relationships between datasets and candidate models. They contribute little on capturing the overall relationship.

In my work, I collected over 7000 time-series from publicly available databases as described in appendix B. These time-series are of different length, sampling rate and furthermore are sampled from different scientific fields, such as manufacturing, finance, meteorology (e.g., temperature, rainfall), traffic and environmental studies (e.g. co2 concentration, pm 2.5 concentration).

Considering the computational expense and redundancy (e.g. time-series with similar structure) in the collected datasets, I will not use all collected time-series to build the meta-leaner. To have a representative set of all collected time-series datasets, while maintaining the diversity, I addressed the problem by clustering all collected time-series [32].

As illustrated in figure 3.3, each time-series is characterized as a feature vector by the outputs of all operations applied to it. Unlike the operations, time-series should be classified as similar when many operations produce similar outputs for them. Euclidean distance metric can illustrate this property well.

Before clustering, feature vectors were pre-processed. If the application is incorrect, operations sometimes don't return a real value but not a number (NAN). There were missing values in feature vectors. I simply filtered the operations, which have missing values, out, because the number of missing values is very small. Naturally, the remaining number of operations after filtration depends on the dataset used. When the percentage of missing values is larger, a special processing approach is required.

Furthermore, different operations have a different range of outputs, which poses another challenge. When computing the distances between feature vectors, the

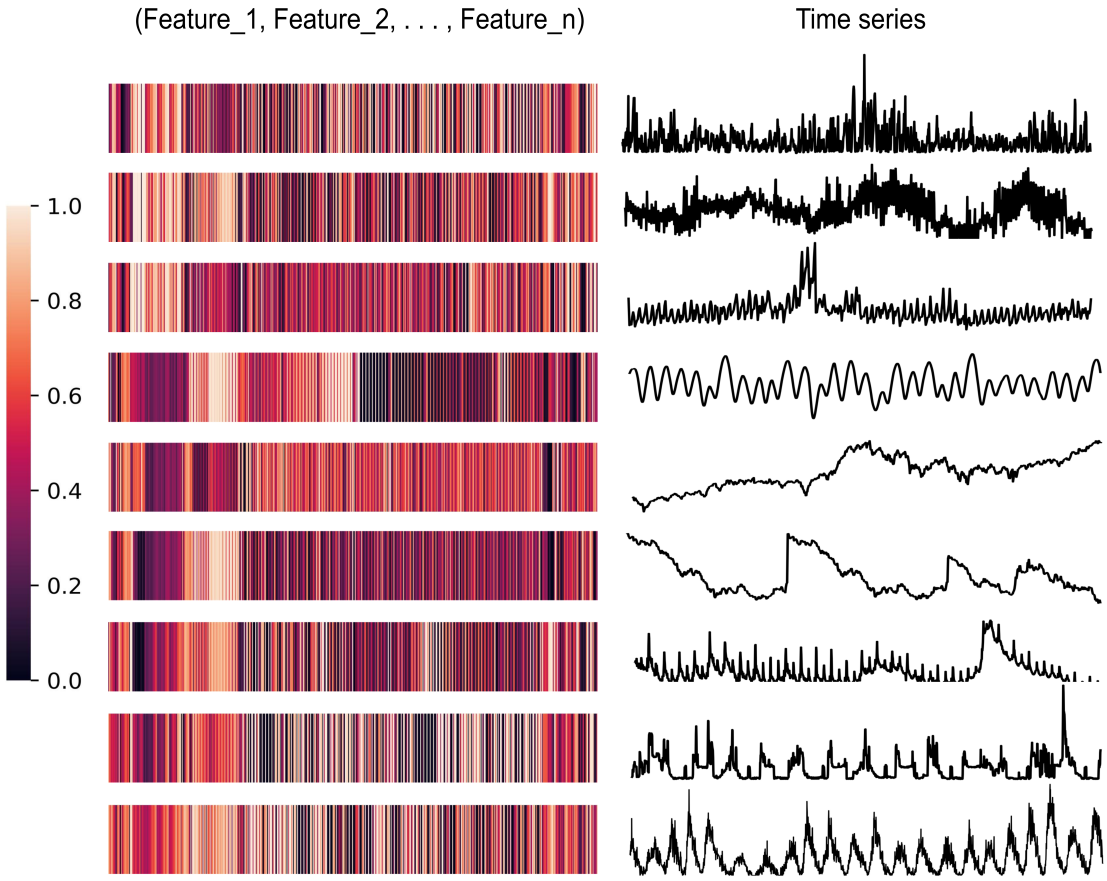


Figure 3.3: The time-series are represented by feature vectors consisting of their properties. Each property of the vectors is the output of an operation applied to the time-series. The value of the property is visualized by using color: from dark (0.0) to bright (1.0). I used 254 operations to illustrate this. All the operations are implemented in TSFRESH package

range of outputs of all operations should be the same, so that all dimensions are equally weighted. Normalization is a reliable method to tackle the aforementioned problem. In addition to missing values, there may be outliers in feature vectors. The normalization mechanisms should be robust to outliers. To deal with this problem, I applied an outlier-robust sigmoid-al transform devised by Fulcher et al.:

$$\hat{f} = \left(1 + \exp(-\frac{f - median(f)}{1.35 \times iqr(f)})\right)^{-1} \tag{3.2}$$

where f are the raw outputs of a given operation across all considered time series, $\hat{f}$ the normalized outputs with range 0 to 1, $median(f)$ the median value of f, $iqr(f)$ the interquartile range of f.

For clustering, I used the hierarchical linkage clustering model [43] implemented in the open source package SciPy [33]. In the hierarchical linkage clustering process, clusters are formed in an agglomerative scheme. It iteratively groups objects, starting with the most similar pair, until all objects are grouped into a single group. This type of clustering requires two choices: (1) Defining distance metric between objects and (2) Defining the linkage method which computes the distance between two clusters. As a reminder, if two time-series are judged as similar, the values of corresponding features should be same. Euclidean distance reflects this property well. Therefore I defined the distance metric as Euclidean distance to measure the dissimilarity between time-series. There are many choices for a linkage method, such as "single", "average", "complete" and "ward" [66]. I applied the "ward" linkage method for the hierarchical clustering.

The output of hierarchical clustering results can be determined either by number of clusters or by a distance threshold to cut the tree at that threshold. I simply defined the distance threshold as 21, as illustrated in figure 3.4. After that, I got 52 clusters of time-series.

To get the representative set of time-series, in addition to the time-series closest to each cluster center, I randomly sampled 2 time-series from each cluster into the representative set. Through that, I got in total of 156 time-series as the representative set of the collected 7000 datasets.

These 156 time-series were named as reduced time-series datasets. In the following work, They were used to build the meta-learning system and to study the forecasting performance and the impact of meta-learning.

In the following section, I'm going to introduce the way to build the meta-level dataset for training meta-learner.

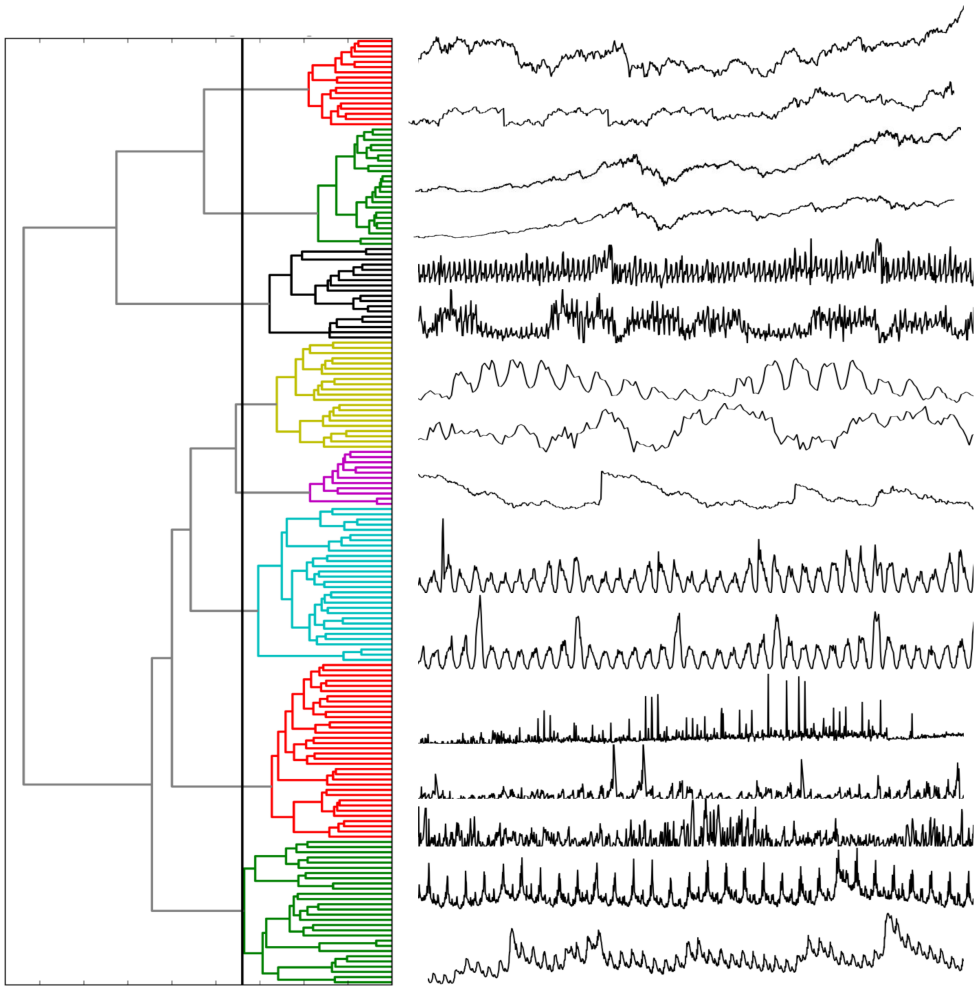


Figure 3.4: This figure is a part of the clustering dendrogram. One color represents a cluster. The vertical thick line in the figure is the distance threshold. The time-series on the right are examples of each cluster. It shows that using hierarchical clustering organizes time-series into meaningful categories. Time-series with similar structures are clustered together.

3.1.3 Meta-level dataset

After training datasets are selected and meta-features defined, a meta-level dataset can be formed. A meta-level dataset consists of two parts:

(1) meta-features of datasets as input

(2) Performance of candidate models as output.

The procedure of forming meta-level data is illustrated in figure 2.6.

First, I split each time-series dataset into a training set and a test set. The size of the test set depends on the horizon of the forecasting task. Then I evaluated the meta-features of the training set of each time-series. It needs to be mentioned that meta-features are exclusively computed from the training sets. After the feature-extraction process, each times-series was represented by a meta-features vector.

To get the performance of each candidate model, I ran SMAC for 10 hours with 10-fold walk-forward cross-validation on the training set of each time-series. Subsequently, I stored the ML regression model instantiation, which has with the best performance on the test set, as a label for each time-series. To be more specific, I applied SMAC to optimize all candidate models separately with the 10-fold cross-validation for each time-series. Finally, in the style of a "winner takes it all" strategy, I labelled each time-series with the best model.

3.1.4 Meta-learner

When faced with a new dataset, choosing the right algorithm and its respective hyper-parameters are a critical step at the beginning of applying ML models. Handling these two problems at the same time is dubbed the CASH problem. The SMAC method has demonstrated significant successes in tackling the CASH problem [19]. However, when started with a new optimization task, it requires a number of initial evaluations to build the initial surface model M_L. For tasks that are evaluation intensive, inappropriate initialization is computationally infeasible. And the quality of the final model depends on the quality of the initial model M_L. An effective initialization helps SMAC to build a better M_L, which tries to accelerate the detection of the high-performance region.

There are many approaches for initialization, such as uniform random sampling, random Latin hypercube design (LHD) and iterated distributed hypercube sampling (IHS) [6]. Frank Hutter et al. studied the performance of four different initialization approaches for the SMBO framework [28]. Unfortunately, their results showed that no initial design is superior to any designs other than simple random sampling.

To conquer this problem, meta-learning is a promising solution, which can suggest good configurations for a new dataset that performed well on previous similar datasets. Using this knowledge, I have applied three different meta-learning

methods. They are k nearest neighbours (KNN) method, modified k nearest neighbours (MKNN) method and the relationship between meta-features and hyper-parameters (RMH) method.

3.1.4.1 KNN method

The first method is the KNN method which was used by Matthias Feurer et al. in Auto-Sklearn [19]. The core idea of this method is to imitate the way, how experts choose models when facing a new dataset.

First, experts will study the characteristics of the new dataset. Then based on these characteristics they associate the new dataset to a previously experienced dataset. At last, they choose the model which performed well on the associated experienced dataset.

The procedure of KNN is outlined in Algorithm 2. $(D_1, D_2, \cdots, D_N)$ are training datasets for building KNN models in the off-line phase. $(\theta_1, \theta_2, \cdots \theta_N)$ denote the model and parameter configuration which performs best on the corresponding dataset. When dealing with a new dataset D_{new}, the distances $d_i = \text{distance}(D_i, D_{new})$ between the new dataset and all previous training dataset are calculated. The distance measure d_i I used is $L1$ distance:

$$d_i = \text{distance}(D_i, D_{new}) = \left\| \mathbf{f}^i - \mathbf{f}^{new} \right\| \tag{3.3}$$

where $\mathbf{f}$ is the meta-features vector of a dataset. A short distance between two datasets indicates similarity. Then select $(\theta_1, \theta_2, \cdots \theta_k)$ of the nearest k datasets based on the calculated distances. These k selected configurations are the initial design for SMAC optimization.

Algorithm 2 KNN meta-learning initialization for SMAC

Require: k number of initial configurations; $(D_1, D_2, \cdots, D_N)$ training datasets; $(\theta_1, \theta_2, \cdots \theta_N)$ best configurations for training dataset; d L1 distance measure;
Ensure: k number of configurations $(\theta_1, \theta_2, \cdots \theta_k)$
1: Calculate distances $d_i = \text{distance}(D_i, D_{new})$ for all training datasets $(i = 1, 2, \cdots, N)$;
2: Sort training datasets by increasing distance to D_{new}
3: Select configurations of the k nearest datasets
4: **return** $(\theta_1, \theta_2, \cdots \theta_k)$

3.1.4.2 MKNN method

The second meta-learning method is MKNN method. Matthias Feurer et at. studied the performance of initial configurations, with different numbers for k. The number k varied in range $\{5, 10, 15, 20, 25\}$. The results of their experiment showed, that more initial configurations perform better. This is reasonable. But I found that

there is not much improvement in the performance between 20 initial configurations and 25 initial configurations. This means that if k is beyond a certain range, the improvement of different k initial configurations is almost stagnant. I believe the reason for that is that the structure of the last few selected datasets might be only remotely similar to the new dataset or even completely different. The configurations from the last few datasets with the least nearest distances contribute only minimally.

I have also noticed that it is very wasteful when only the best model and its configurations is related to each training time-series. Usually, some other models perform as well as the best model.

In order to make the most use of the nearest datasets and the result from the offline phase, the procedure of previous KNN is modified. For each training dataset D_i, two configuration $(\theta_{i1}, \theta_{i2})$ are stored. $(\theta_{i1}, \theta_{i2})$ are different ML models whose performance are the first and second best over D_i.

If the number of initial configurations k is an even number, by detecting the $frack2$ nearest datasets, k initial configurations can be received. If the the number of initial configurations k is an odd number, by detecting the $\frac{k-1}{2}$ nearest datasets, $k-1$ initial configurations can be received. In order to get k initial configurations, the best configuration of the $\frac{k+1}{2}th$ nearest dataset should be added.

3.1.4.3 RMH method

The third method is the RMH method, which tries to learn the relationship between meta-features and hyper-parameters. The core idea is to mimic the expert's strategy of tuning parameters. For example, when using support vector machines to deal with classification problems, if the dataset is found to be too noisy through analysis, parameter C is usually set a little bit larger to prevent over-fitting. This phenomenon shows that there exists relationship between meta-information and hyper-parameters.

I trained a model separately for each hyper-parameter of the candidate models. The input to the model is the meta-features vector of a dataset. And the output is a hyper-parameter. If the hyper-parameter is continuous, the model is a regression model. If the hyper-parameter is categorical, the model is a classification model. I applied RF model for the regression tasks or classification tasks.

3.2 Forecasting pipeline

In my work, forecasting problems are considered as supervised learning problems. I solved such ML problems following the commonly used process, which is known as data-mining process [35] or ML process [40]. The process consists of the following five steps:

(1) Data preparation

(2) Feature extraction and selection

(3) Feature processing

(4) Algorithm selection and hyper-parameters configuration

(5) Evaluation

For each step, there are a number of methods to choose from. As illustrated in figure 3.5, each step takes the suggestions from the SMAC optimization model. Every steps involved in the process are optimized at the same time. So in the context of my work, model selection means to select the best model for every steps. In the next five sections, I introduce these five steps and the included options for each step in chronological order.

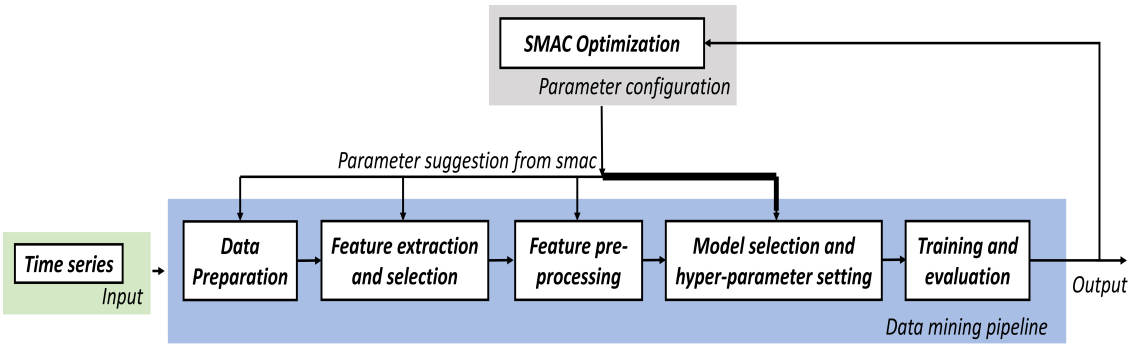


Figure 3.5: The data mining process for forecasting problem.

3.2.1 Data preparation

Data preparation is the process to prepare raw datasets into refined information assets that could be used effectively for analysis purposes [48]. With the prepared dataset, models could be built faster as opposed to models with raw datasets [50]. During the preparation, the natural structure of datasets must not be disturbed. For the preparation of time-series datasets, the following three aspects are considered: normalization, missing value imputing and differencing.

Normalization is a technique to rescale the time-series dataset from original range to range of 0 and 1. The value is normalized as follows:

$$\hat{y} = \frac{y - \min}{\max - \min} \tag{3.4}$$

where $\hat{y}$ is the normalized value and min and max are the minimum and maximum of the dataset. This method is advantageous, as it does not change the structure

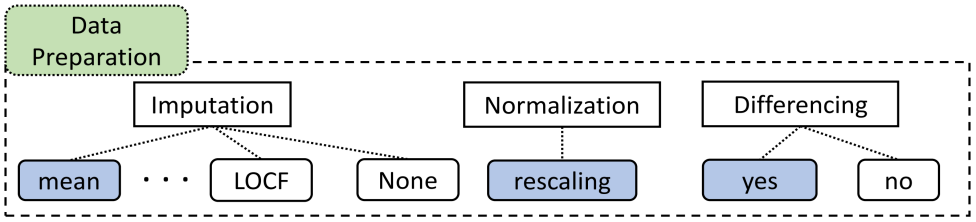


Figure 3.6: Configuration space for data preparation. The blue coloured rectangles represent default setting for this step.

of the time-series and the transformation can be inverted. Considering the undesirable dependence of some operations on location, this normalization is treated as a standard preparation for each time-series dataset.

If there are missing values in the time-series dataset, there are two commonly used methods. One consists of discarding the entire record, the other one consists of imputing the missing information. Discarding the record reduces the information available for modelling. Here, I considered only imputation approaches. The imputation approaches I included are mean imputation, LOCF method, next observation carried backward (NOCB) method, linear interpolation, spline interpolation, cubic interpolation and moving average.

Typically, differencing is applied to time-series to remove the series dependency on times. This is a widely used preparation method in the statistical community [53]. Differencing can help stabilize the mean of the time-series by removing changes in the level of a time-series, and so reducing trend and seasonality [29]. Differencing is performed by subtracting the previous observation from the current observation. It was considered as an option in my framework for data preparation. The range of the differencing order was from 0 to 2.

As illustrated in figure 3.6, during the optimization process, my framework will optimize the method to impute when there are missing values and whether to apply differencing to the time-series. And normalization is a standard preparation process for all input time-series.

3.2.2 Feature extraction and selection

Feature extraction is a process mapping the set of initial measured data into specific properties of raw datasets [25]. It helps the model to understand the dataset more effectively. A feature is the result of a mapping operator which summarizes raw datasets with a real value. One simple example for such an operator is the variance operator. Unlike the experiment introduced in section 2.1, in which I only used past values as features to predict values of the future. During this step, I extended the feature set to a more comprehensive feature set by applying TSFRESH.

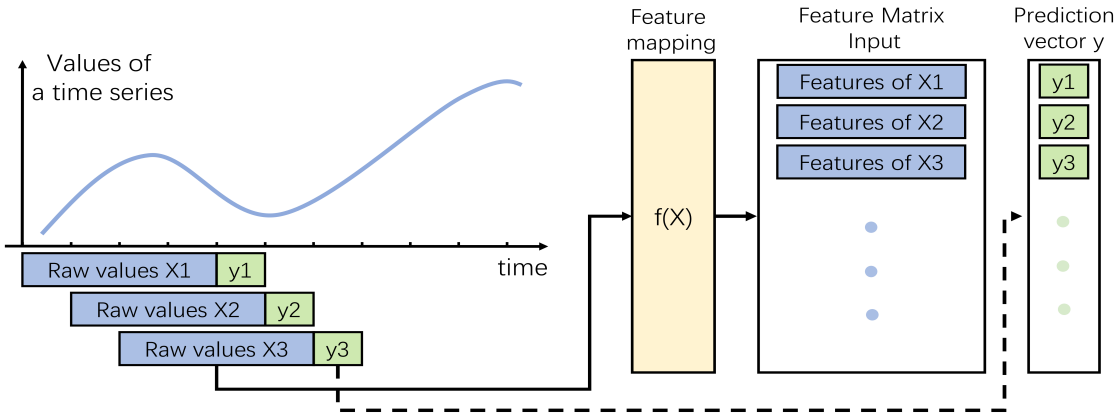


Figure 3.7: This figure visualizes the process of feature extraction. X stands for a sub time-series and Y indicates the corresponding future value. It shows how the blue, rolled sub time-series are used as base for the construction of the feature matrix. Sliding the window along the time axe (from left to right), each sub time-series is the input of feature mapping. After the mapping, feature matrix is formed. The green data points are the values need to be predicted by the model.

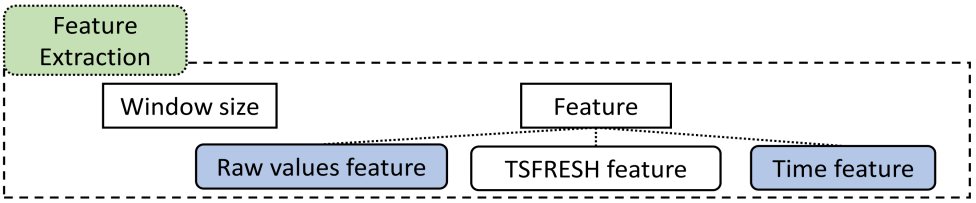


Figure 3.8: Configuration space for feature extraction. The size of window ranges from 5 to 100.

The mechanism of feature extraction in my framework is illustrated in figure 3.7. Trough the rolling mechanism, I got sub time-series of w time steps to calculate the features, where w is the size of the window width. In addition to using raw values as features, I calculated all feature-mappings that are available in the TSFRESH package on the sub time-series. These set features cover analysis properties from different domains. Moreover, I also used the time index of the predicted values as features (e.g. year, month, and minute), if the time index is available.

In summary, there are three types of features that are extracted from the sub time-series, namely the raw value features, the TSFRESH features and the time index features. As illustrated in figure 3.8, during the optimization process, the framework selects which kinds of features to be used, such as using only raw value features or using features of all types.

Typically, extracted features contain some features that are irrelevant or redundant. I applied the selection mechanism in the TSFRESH package directly to select relevant features. Figure 3.9 visualizes the process of feature selection in TSFRESH.

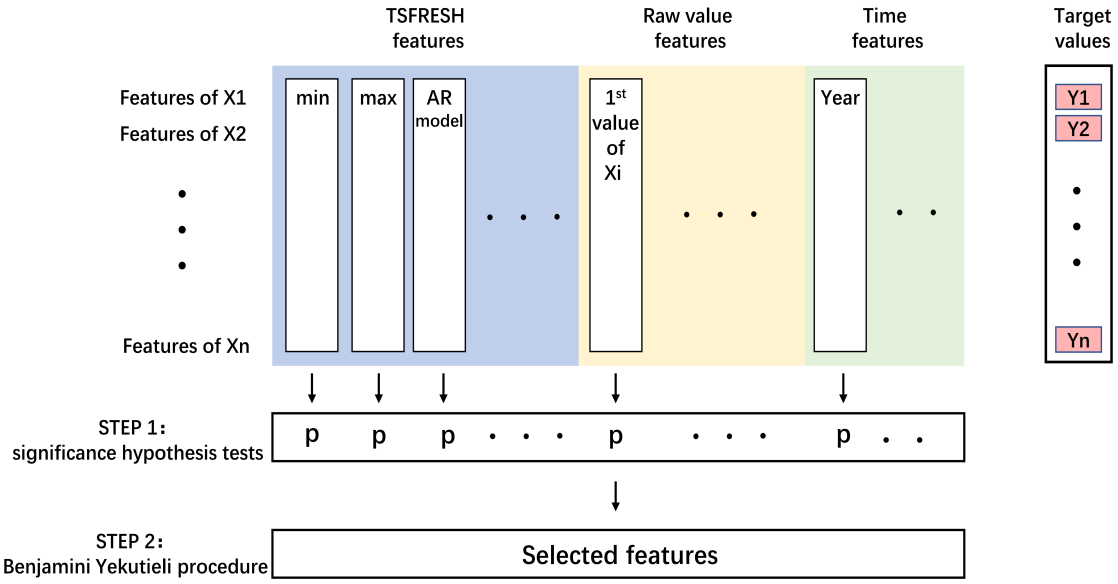


Figure 3.9: The upper half of this figure corresponds to the right part of figure 3.7. After feature extraction we got the feature matrix. The feature matrix consists of three parts, the blue part is the TSFRESH features, the yellow part is the raw value features and the green part is the time features. In the first step of feature selection each feature vector $\mathbf{f}$ (column) is individually and independently evaluated with respect to its significance for predicting the target $\mathbf{Y}$. In the second step, features are selected by performing Benjamini-Yekutieli procedure based on the vector of p-values.

A feature $\mathbf{f}$ is defined as relevant for forecasting $\mathbf{Y}$ only if $\mathbf{f}$ and $\mathbf{Y}$ are statistically dependent [14]. To check whether a given feature satisfies the definition, hypothesis testing is an appropriate approach.

The selection process consists of two steps. During the first step, each extracted feature $\mathbf{f_i}$ is separately deployed a statistical test to check the hypothesis

$$H_0^\phi = \{X_\phi \; is \; irrelevant \; for \; predicting \; Y\},$$

$$H_1^\phi = \{X_\phi \; is \; relevant \; for \; predicting \; Y\}.$$

The result of the statistical test is a probability value called p-value. This p-value describes the probability whether a given feature is relevant. Features with small p-value are relevant to the prediction target.

For the second step, the vector of p-values is evaluated on basis of the Benjamini-Yekutieli procedure [7], a multiple testing procedure, to decide which features to keep. Figure 3.10 demonstrates an example of the Benjamini-Yekutieli procedure.

Feature selection is treated as an obligatory measure post feature extraction. The filtered feature types are saved so that only the filtered features are calculated when making predictions.

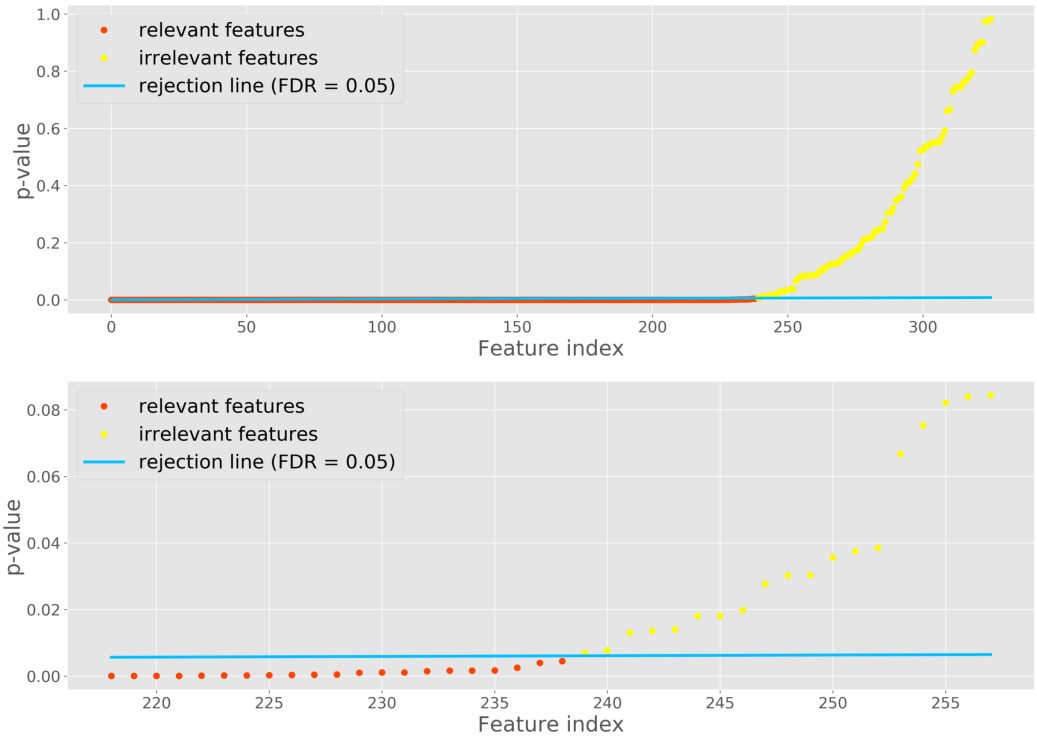


Figure 3.10: During the BY procedure a rejection line will be generated(illustrated as blue line). The p-values are ordered from low to high. According to the intersection of the ordered p-values and the rejection line, features are divided into two groups, relevant and irrelevant. The features to the left of the intersection are relevant. The features to the right of the intersection are irrelevant. Since the upper plot is not clearly visible, a zoomed plot is provided.

	$f_{t,1}$	$f_{t,2}$	$f_{t,3}$	$f_{t,4}$	$f_{t,5}$	$f_{t,6}$
Monday	1	0	0	0	0	0
Tuesday	0	1	0	0	0	0
Wednesday	0	0	1	0	0	0
Thursday	0	0	0	1	0	0
Friday	0	0	0	0	1	0
Saturday	0	0	0	0	0	1
Sunday	0	0	0	0	0	0
Monday	1	0	0	0	0	0
⋮		⋮				⋮

Figure 3.11: Encoding for time features (days of the week).

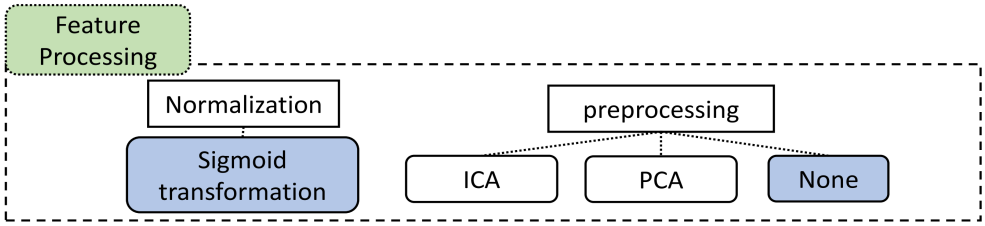


Figure 3.12: Configuration space for feature processing.

3.2.3 Feature processing

After the features have been filtered, there are some subsequent processes that need to be done, namely feature encoding, feature normalization and dimensionality reduction.

Features encoding is applied for time features because they are categorical variables, like minutes and days of the week. In addition, there are cyclic structures in the time feature. During the encoding, this structure needs to be respected. This problem can be handled by using a modified one-hot encoding, where n − 1 variables are needed to encode n categories. Figure 3.11 demonstrates an example of the encoding.

The range of each feature can vary significantly. I applied the outlier-robust sigmoid transformation as used in section 3.1.2 to scale all features into range [0, 1].

Encoding and normalization are treated as a mandatory measure for filtered features.

The dimensionality of the filtered features is usually also high. The principal components analysis (PCA) method [60] and fast independent component analysis (ICA) method [30] are highly effective dimensionality reduction methods, while maintaining the amount of information of features as much as possible.

As illustrated in figure 3.12, during the optimization process, my framework chooses whether to use PCA or ICA. If one of them is used, the number of components will be optimized.

3.2.4 Candidate models and hyper-parameter setting

In this section, I list the hyper-parameters and their respective range of all regression models which are considered in my framework.

In my project, a total of 10 regression models were included. For the hyper-parameters which are not listed in the table below, the default value defined in the Sklearn package will be used directly. For further inquiries, please refer to Sklearn's website for the meaning of the parameters.

AdaBoost regressor		
Hyper-parameter	values	Tyoe
n_estimators	[50, 500]	int
learning rate	[0.01, 2]	int
loss	"linear", "square", "exponential"	categorical
max depth	[1, 10]	int

Table 3.1: Hyper-parameters of adaboost regressor.

Bayesian ARD regressor		
Hyper-parameter	values	Tyoe
tol	[0.00001, 0.1]	float
alpha 1	[1e-10, 1e-3]	float
alpha 2	[1e-10, 1e-3]	float
lambda 1	[1e-10, 1e-3]	float
lambda 2	[1e-10, 1e-3]	float
threshold lambda	[1000, 50000]	float

Table 3.2: Hyper-parameters of Bayesian ARD regressor.

Regression based on k-nearest neighbors		
Hyper-parameter	values	Tyoe
n neighbors	[1, 100]	int
weights	"uniform", "distance"	categorical
p	[1, 2]	int

Table 3.3: Hyper-parameters of KNN regressor.

Random forest regressor		
Hyper-parameter	values	Tyoe
criterion	”mes”, ”friedman_mse”, ”mae”	categorical
max features	[0.1, 1.0]	float
min samples split	[2, 20]	int
min samples leaf	[1,20]	int
bootstrap	”True”, ”False”	bool
n estimators	[10, 200]	int

Table 3.4: Hyper-parameters of random forest regressor.

Ridge regressor		
Hyper-parameter	values	Tyoe
alpha	[10e-5, 10]	float
tol	[1e-5, 1e-1]	float
fit intercept	”Ture”, ”False”	bool

Table 3.5: Hyper-parameters of ridge regressor.

Stochastic Gradient Descent regressor		
Hyper-parameter	values	Tyoe
loss	”squared”, ”huber”, ”epsilon”, ”squared epsilon”	categorical
penalty	”l1”, ”l2”, ”elasticnet”	categorical
alpha	[1e-7, 1e-1]	float
l1 ratio	[1e-9, 1]	float
tol	[1e-4, 1e-1]	float
epsilon	[1e-5, 1e-1]	float
learning rate	”optimal”, ”invscaling”, ”constant”	categorical
eta0	[1e-7, 1e-1]	float
power t	[1e-5, 1]	float
average	”True”, ”False”	bool

Table 3.6: Hyper-parameters of Stochastic Gradient Descent regressor.

eXtreme Gradient Boosting regressor		
Hyper-parameter	values	Tyoe
max depth	[1, 10]	int
learning rate	[0.01, 1]	float
n estimators	[50, 500]	int
subsample	[0.01, 1]	float
min child weight	[1, 20]	int

Table 3.7: Hyper-parameters of eXtreme Gradient Boosting regressor.

extra-trees regressor		
Hyper-parameter	values	Tyoe
n estimators	[10, 200]	int
criterion	”mes”, ”friedman_mse”, ”mae”	categorical
max features	[0.1, 1.0]	float
min samples split	[2, 20]	int
min samples leaf	[1,20]	int
bootstrap	”True”, ”False”	bool

Table 3.8: Hyper-parameters of extra-trees regressor.

Gaussian process regressor		
Hyper-parameter	values	Tyoe
alpha	[1e-10, 1]	float
theta L	[1e-10, 1e-3]	float
theta U	[1, 100000]	float

Table 3.9: Hyper-parameters of Gaussian process regressor.

Epsilon-Support Vector regressor		
Hyper-parameter	values	Tyoe
C	[0.01, 20000]	float
epsilon	[0.001, 1]	float
kernel	”rbf”, ”linear”, ”sigmoid”, poly”	categorical
degree	[2, 5]	int
gamma	[1e-5, 8]	float
shrinking	”True”, ”False”	bool
tol	[1e-5, 1e-1]	float
coef0	[-1, 1]	float

Table 3.10: Hyper-parameters of Support Vector regressor.

3.2.5 Evaluation

The difference between an observed value and its forecasting is called ”error”. There are different ways to evaluate the forecasting accuracy by summarizing the forecast errors across the forecasting horizon. The most commonly used metrics are: mean absolute error (MAE), root mean squared error (RMSE), SMAPE and mean absolute scaled error (MASE).

Different metrics prefer different models. For example: Let's say the future values are (2,4,6,8). The predicted values of model A are (3,5,7,9). The predicted values of model B are (2,4,6,11). The MAE of model A is 1 and the MAE of model B is 0.75. The RMSE of model A is 1 and the RMSE of model B is 1.5. If MAE is used as a metric, model B is better than model A. If RMSE is used as a metric, model A is better than model B. The reason is that the RMSE penalizes large errors heavier than the MAE.

In the experiment described in section 2.3, the metric for the evaluation was defined as SMAPE. If a meta-learner is built during this experiment, it can only suggest configurations which will probably perform well under the SMAPE metric.

Therefore, the metric for the evaluation must be defined before building the meta-learning system. For the experiments in the following chapter, I used SMAPE as the evaluation metric.

3.2.6 Structured configuration space

This section serves as a summary for section 3.2. Figure 4.12 demonstrates all considered components for all processing steps. This figure shows the contents of the figure 3.5 in more detail. Components in a rounded rectangle are conditional to components in a sharp edged rectangle. As an example, blue coloured rectangles represent an active set of components, which forms an example for a ML pipeline.

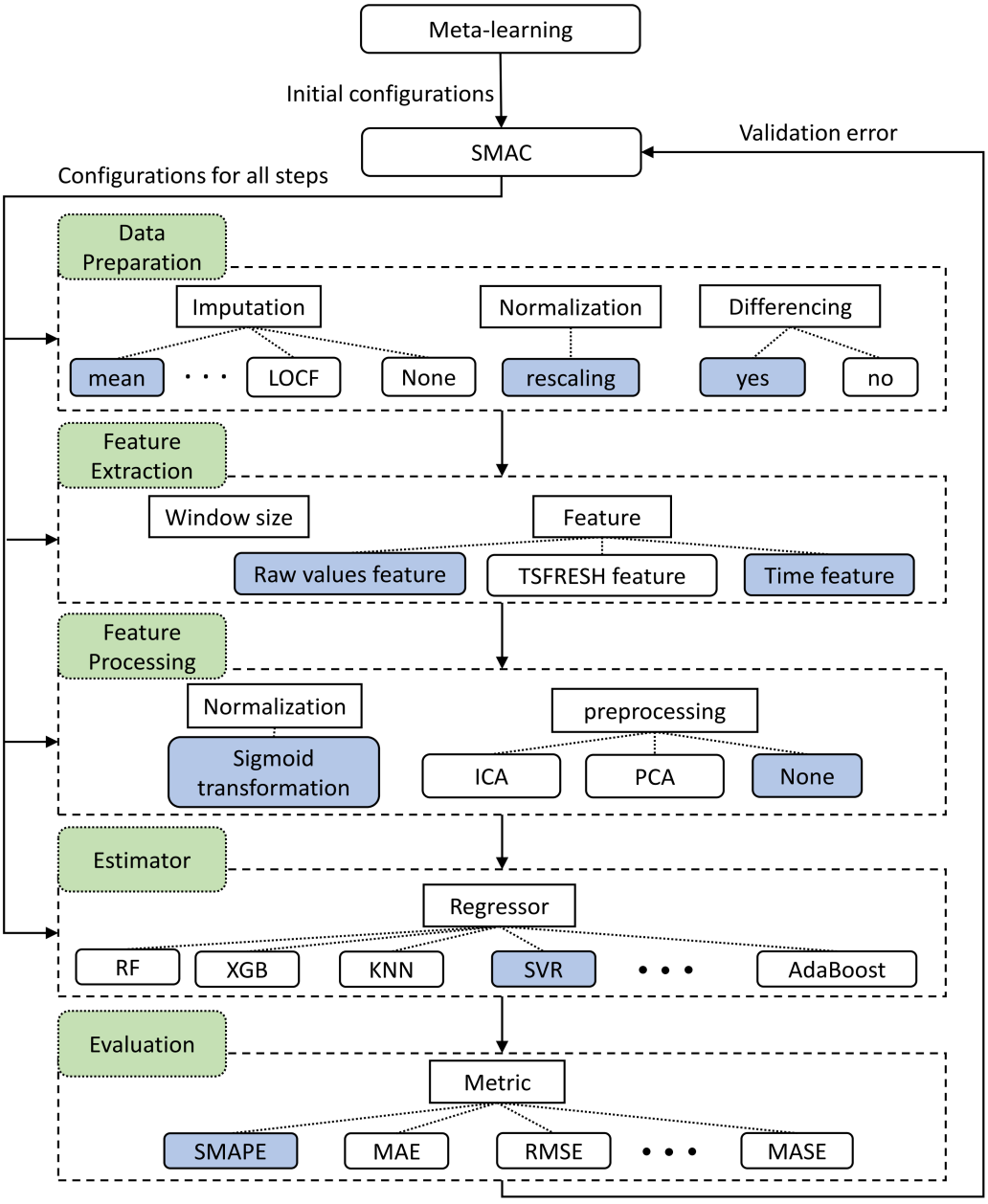


Figure 3.13: Structured configuration space of my framework.

4. Evaluation

This chapter presents the main results of my empirical studies. The discoveries of this thesis are drawn from studies of the following experiments.

Section 4.1 introduces the result of a multi-class classification experiment based on five types of Electroencephalogram (EEG) time-series. The experiment's aim was to study the performance of the meta-features, which are applied in my framework. The result indicated that the applied meta-features can efficiently summarize the characteristics of time-series.

Section 4.2 introduces the result of two experiments I conducted. These two experiments aimed to study the performance of the applied meta-learning methods. The results indicated that the suggestions from the KNN method and the MKNN can efficiently accelerate the optimization process. Unfortunately, the method RMH, did not speed up the optimization process.

In section 4.3, I elaborated on the reasons responsible for the bad performance of the RMH method, by visualizing the relationship between hyper-parameters and performance.

In section 4.4, I drew a comparison between my framework and other classic models (the ARIMA model, neural network (NN) model and the Automatic Forecasting Procedure (Prophet) model [56]), in order to evaluate the forecasting performance of my framework.

4.1 Experiment on meta-features

The purpose of this experiment is to study the performance of meta-features. I assumed that if these meta-features can capture the characteristics of time-series well, the classification accuracy based on these meta-features should be high.

In this experiment, I used the publicly available EEG time-series dataset [3]. This dataset consists of five labelled sets, denoted as A, B, C, D and E. These five sets are recorded under different conditions. Time-series of sets A and B were taken from surface EEG recordings that were carried out on healthy volunteers. Volunteers were in a relaxed and awake state with their eyes open (set A) and their eyes closed (set B), respectively. Time-series of sets C, D and E originated from pre-surgical diagnoses of epileptic patients during seizure episodes. Time-series of set C were recorded from the hippocampal formation of the opposite hemisphere of the brain. Time-series of set D were recorded from within the epileptogenic zone. Time-series of set E only contained seizure activity. Each set contains 100 time-series. Each time-series consists of 4096 samples.

I applied the support vector machine classifier implemented in the scikit-learn package. The set of meta-features described in appendix A were used as input of the classifier. After the meta-features extraction, each meta-feature was rescaled to range from 0 to 1 through the outlier-robust sigmoidal transform [22]. Figure 4.1 visualizes a three dimensional principal components projection of the dataset. It shows that the applied meta-features (analysis operations) structure the five sets in a meaningful way.

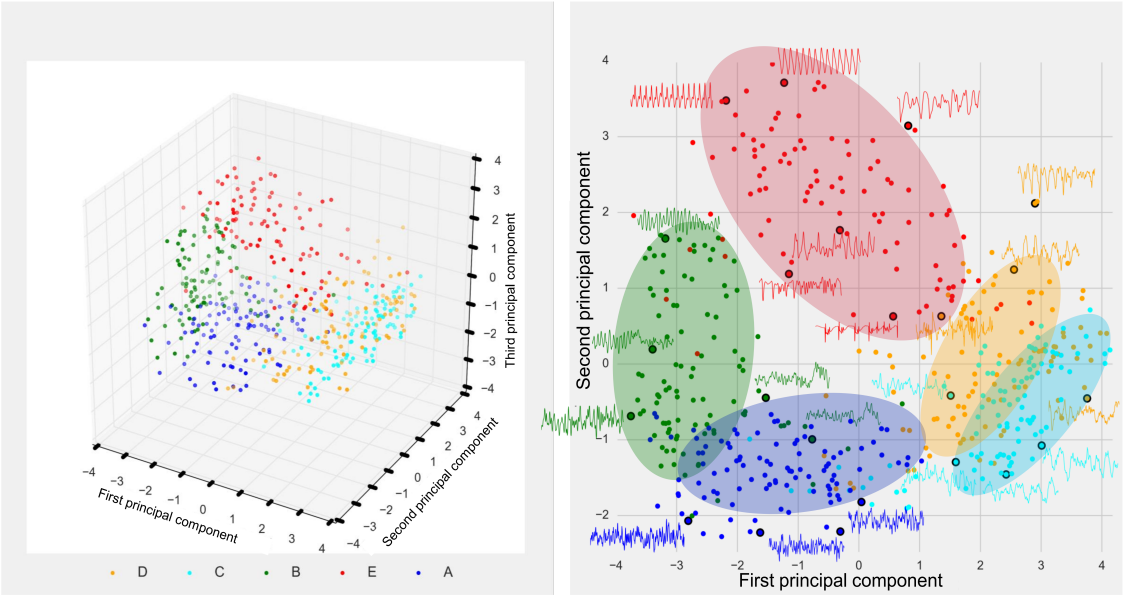


Figure 4.1: Three dimensional principal components projection of the five sets. Background shading is used to help visualize the area of each set. For the points with a black circle in the plot, segments of the corresponding time-series are annotated.

I have investigated a total of three classification task. Firstly, I performed a multi-class classification task between the five sets. Secondly, I investigated a binary classification task between healthy people (A and B) and patient (C, D and E). This database has been previously used for classification, with the purpose to separate sets A (healthy) and E (seizure) of the dataset. Therefore, I performed the third

classification task, which is another binary classification task to distinguish between sets A and E. The procedure of the three experiments can be summarized as follows:

Step 1: Calculating all meta-features.

Step 2: Filtering out the meta-features which are unique or contain missing values. Rescaling the filtered meta-features to range from 0 to 1 through the sigmoidal transform.

Step 3: Carrying out the classification process by using support vector machine classifier.

Because radial basis function kernel was used as the kernel function of support vector machine classifier, there are two parameters related with this kernel: penalty parameter C and kernel coefficient γ. The other parameters remained as their default values. I used 10-fold cross-validation to investigate the appropriate kernel parameter C and γ. Principally, I used grid-search method to try all the pairs of (C, γ) and the one with the best cross-validation accuracy is selected. The result is listed in table 4.1 .

	Accuracy	Precision	Recall	f1
Classification task 1	93.40%	94.03%	93.40%	93.32%
Classification task 2	98.40%	98.82%	98.67%	98.7%
Classification task 3	99.50%	100%	99%	99.47%

Table 4.1: The values of the results for the classifications

These experiments have shown good performance on the EEG time-series classifications. This indicates that the applied meta-features can capture the characteristics of time-series well.

4.2 Experiment on meta-learning

In this section, I introduce two experiments I conducted, to study the performance of the applied meta-learning methods. I verified their acceleration performance during the optimization process.

4.2.1 Experiment on the datasets of M3-competition

In section 2.3, I conducted a basic experiment on datasets of M3-competition, to pre-study the forecasting performance of the combination of ML regression models and SMAC optimization model. It has achieved very good forecasting results and it also reflected a drawback of the SMAC model, that is, when dealing with a large hyper-parameters space, it needs a number of evaluations to find the high performance region. In other words, the start-up of the optimization process is slow. In this experiment, based on the basic experiment, I tried to use meta-learning method to accelerate the optimization process.

Experiment set-up

First, I will briefly present the set-up of this experiment. The data used in this experiment is the "other" dataset from the M3-competition. It contains 174 time-series. The forecasting horizon is defined as 8, which was stipulated by the competition. The first $n - 8$ observations were used for training/validating the models, and the last 8 for testing the forecasting performance.

I applied the recursive strategy to make multi-step forecasts. The evaluation metric used in this experiment is SMAPE. Considering the costliness of calculation and the interpretability of the results, I only included three models: the SVR regression model, the XGB regression model and the RID regression model. Up to this point, this set-up is the same as the experiment set-up in section 2.3.

Using the method described in section 3.1.3, the meta-level dataset was formed. I ran SMAC for one hour with 10-fold walk-forward cross-validation on the training set of each time-series. And each time-series was labelled with the best model and corresponding hyper-parameters, which got the best test performance on the test set.

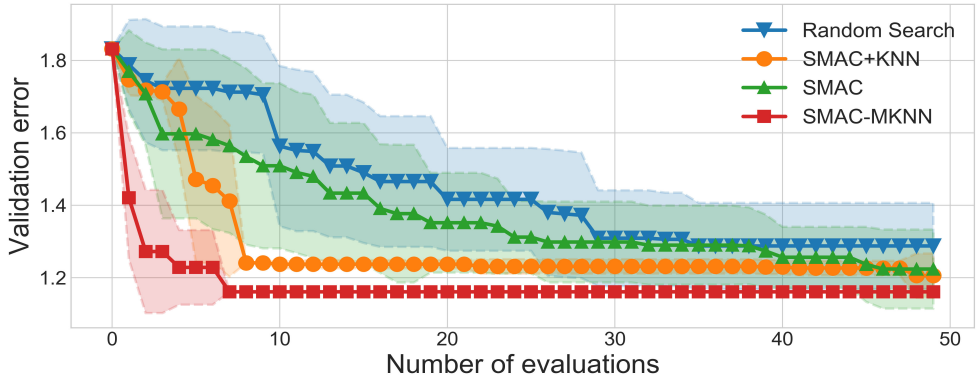
I evaluated the performance of five hyper-parameter optimization procedures: random search, SMAC without meta-learning, SMAC with the KNN meta-learner, SMAC with the MKNN meta-learner and SMAC with the RMH meta-learner. All meta-learning methods are presented in section 3.1.4. The number of generated initial configurations by meta-learner was defined as 8.

In order to evaluate the performance of SMAC with meta-learning methods, I applied the leave-one-dataset-out strategy. When a time-series is being evaluated, the remaining time-series and their best models are known. More specifically, to evaluate the performance of SMAC with meta-learner on one time-series, I assumed that the other 173 time-series and their best hyper-parameter configurations are available. Due to the randomness of SMAC and random search, I repeated each optimization ten times on each time-series.

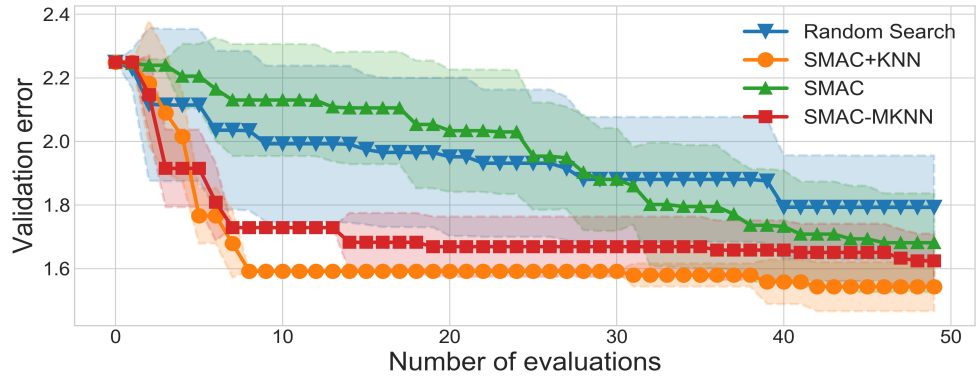
I am going to ignore the result of RMH meta-learner for the time being, due to its bad performance and therefore its interference with the visualization of the result. However, I will elaborate on the reasons for its bad performance in section 4.3.

Result

Moving on to the result of this experiment. Figure 4.2, 4.3 and 4.4 illustrate the qualitative performance of four optimization procedures on representative time-series. The figures show the mean of the best evaluation values obtained from an optimization procedure up to a given number of evaluations. Because the first configuration for all optimizers were defined as the same default configuration, the curves started at the same value point. In total, there are three different situations.



Dataset: M3 time-series number 2946



Dataset: M3 time-series number 2972

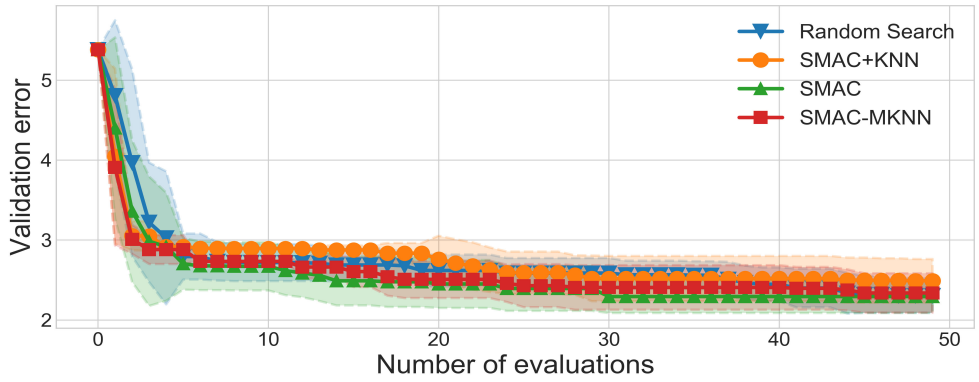
Figure 4.2: The case where the initial configurations suggested by meta-learners were very suitable. There was almost no room left for SMAC to improve. The shadow in the figure represents the standard deviation.

Figure 4.2 exhibits the first case, where the initial configurations, suggested by meta-learners, were very suitable for the unseen new time-series. Suggestions from meta-learner helped the optimizers to quickly find the high-performance region. After the 8th evaluation, there were only slight improvements in the performance. The initial configurations suggested by meta-learner were among the best.

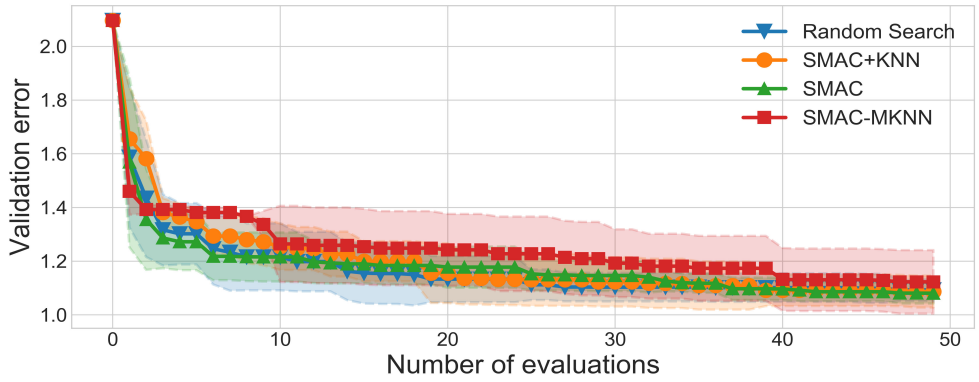
Figure 4.3 illustrates the second case, where the performance of the four optimizers was similar. The suggestions from meta-learner worked like random sampling.

Figure 4.4 shows the third case, where meta-learning could potentially make the performance worse. The initial configurations suggested by meta-learner were unsuitable for the new time-series. The performance of these initial configurations were worse than the performance of default configurations. By applying meta-learning as initialization, the SMAC procedure doesn't start until all configurations suggested by meta-learner have been evaluated. Unsuitable initial configurations delay the start of SMAC.

In order to better demonstrate the optimizers' performance over all time-series, I used a ranking-based evaluation method. I computed the ranks of the four opti-



Dataset: M3 time-series number 2964



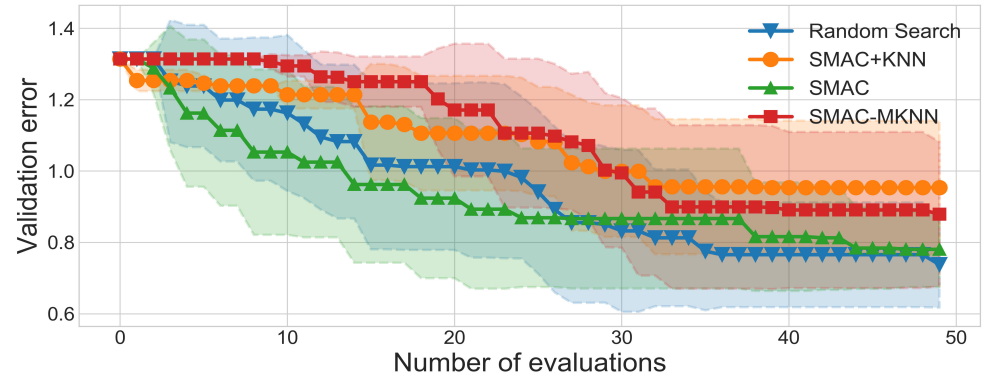
Dataset: M3 time-series number 2839

Figure 4.3: Case where there is no significant difference in the optimization speed of the four optimizers.

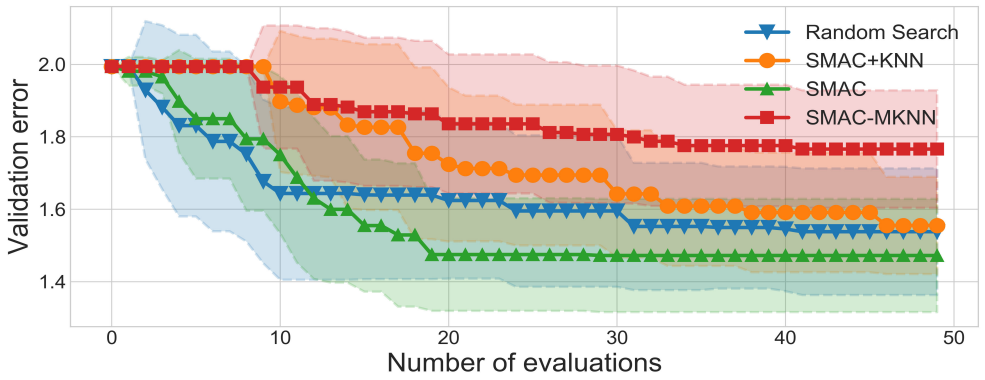
mizers according to the mean of best evaluation values for each time-series at each evaluation step. Then I took the average of the ranks at each evaluation step across all time-series for each optimizer. The result is illustrated in figure 4.5.

From figure 4.5, I observed that the two SMAC variants with meta-learning yielded a rapid performance improvement at the beginning of the optimization and the advantage lasted to the end. The curve of SMAC with MKNN was below the curve of SMAC with KNN, which means that SMAC with MKNN performed better on the M3 "other" dataset. This implies that making full use of the most similar datasets produces a greater chance at generating better initial configurations.

After the 8th evaluation, the ranks of the two SMAC variants slightly decreased over time, which doesn't mean that the two SMAC variants performed worse over time. The reason behind this is that the ranks are a relative measure of performance. As time goes on, the performance of each optimizer is improving. If other optimizers achieve a greater performance improvement, an optimizer's rank can decrease. The ranks of SMAC without meta-learning gradually increased with time and finally approached the ranks of SMAC with KNN. This indicates that SMAC can find



Dataset: M3 time-series number 2885



Dataset: M3 time-series number 2860

Figure 4.4: The case where meta-leaner could delay the optimization procedure of SMAC. The improvements have occurred only after all initial configurations have been evaluated.

good configurations without the help of meta-learning, if there is sufficient time. The only downside is that it requires a higher amount of evaluations.

Previous ranking is based on the mean of best evaluation values. However, in practice, the optimizer will not be executed many times, which means the average won't be used as the result. In order to show a more realistic ranking of the optimizers, I used the ranking-based evaluation proposed by Bardenet et al [5].

There were 10^4 possible combinations of the four optimizers for each time-series, because for each time-series, each optimization procedure has been executed ten times. I applied the bootstrap sampling method to get 500 combinations of the four optimizers for each time-series. In total, I got 87000 combinations. After calculating the ranks of each optimizer at each evaluation step, I took the average of the ranks across all 87000 combinations. The result is depicted in figure 4.6.

In order to verify the forecasting performance improvement, I compared the test performance of SMAC without meta-leaner with the test performance of SMAC with MKNN. During each optimization, the maximum number of configuration-evaluations was defined as 200. In addition, I recorded the test performance of the

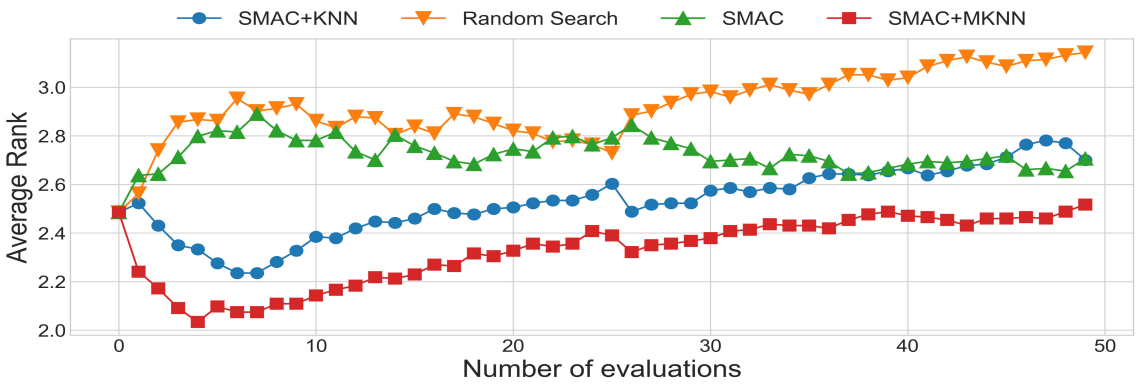


Figure 4.5: The average rank of the four optimization procedures across 174 time-series at every evaluation step. The ranking is based on the mean of the best validation performance.

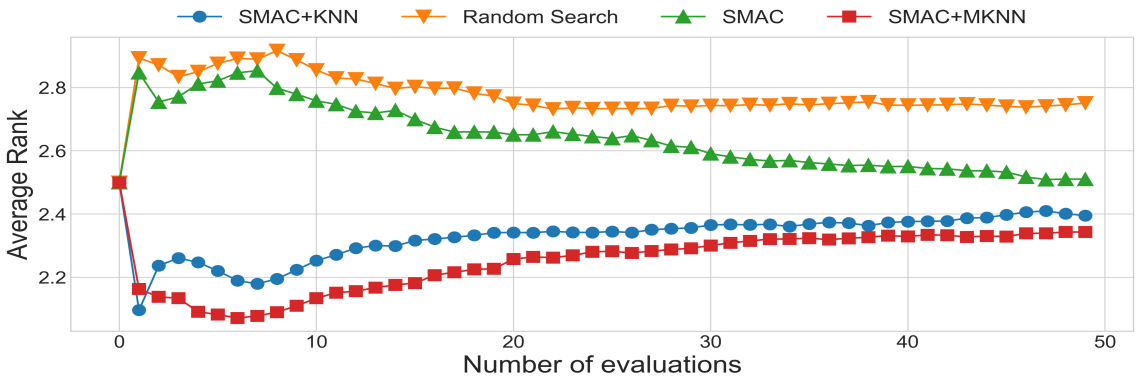


Figure 4.6: Average ranks of the four optimizers over 87000 bootstrap sampled combinations. The curves of this figure are consistent with the curves in figure 4.5. The difference is that the curves of this figure are much smoother, which implies a stabler ranking of these four optimizers.

two optimizers at the 50th evaluation. There result is presented in table 4.2. The numbers in parentheses represent the evaluation step at which the test performance was recorded. The forecasting methods were arranged according to their performance rankings. The results of the methods written in black are published by the M3-competition [38].

The result showed, that the test performance at the 50th evaluation step of SMAC with MKNN was much better than that of SMAC without meta-learner. With the help of meta-learner, SMAC can efficiently save enormous evaluations. This characteristic is very desirable for real world applications. Suppose there are 100,000 time-series to be predicted and SMAC can save 10 evaluations per optimization procedure. This would result in a reduction of 1,000,000 evaluations.

Compared to SMAC with MKNN, SMAC without meta-learner got a better test performance at the 200th evaluation. The reason may be that 200 configuration evaluations were sufficient for SMAC to find the high-performance region. The performance of SMAC caught up over time.

It is worth mentioning, that the test performance of SMAC with MKNN at the 200th step was worse than it was at the 50th step. This indicates that there is a risk of over-fitting.

Method	F-horizon	SMAPE	Method	F-horizon	SMAPE
ARARMA	8	4.38	Automat ANN	8	4.8
Autobox2	8	4.41	Winter	8	4.81
Theta	8	4.41	Holt	8	4.81
SMAC (200)	8	**4.49**	SmartFcs	8	4.86
Comb S-H-D	8	4.56	Flores/Pearce2	8	4.89
Robust-Trend	8	4.58	Autobox1	8	4.93
ForecastPro	8	4.6	Theta-sm	8	4.93
Dampen	8	4.61	B–J automatic	8	5.06
SMAC+MKNN (50)	8	**4.61**	Flores/Pearce1	8	5.09
PP-autocast	8	4.62	**SMAC (50)**	8	**5.32**
ForecastX	8	4.64	RBF	8	5.6
SMAC+MKNN (200)	8	**4.67**	Single	8	6.29
Autobox3	8	4.71	Naıve2	8	6.3

Table 4.2: Average SMAPE evaluation on ”other” dataset of the M3 competition.

4.2.2 Experiment on more comprehensive datasets

I studied situations in which meta-learning decreased the performance of the SMAC optimization. In those situations, the most similar time-series happened to have a large distance to the optimized time-series. Therefore, the suggestion from the most similar time-series was unsuitable for the optimized time-series. In fact in those situations, there was no time-series in the meta-level dataset that was similar to the optimized time-series. Whenever all suggestions from the meta-level datasets were unsuitable, the start of the SMAC procedure was delayed.

This problem occurs, when the diversity of the meta-level dataset is insufficient. The diversity of meta-level datasets is comparable to the experience of an expert. The richer the diversity, the more experienced the expert. To build a better meta-learner, meta-level datasets with sufficient diversity are necessary. In this experiment, I rebuilt the meta-learning system based on the representative set with sufficient diversity and tested the impact of meta-learning.

Experiment set-up

The experiment set-up is roughly the same as the previous experiment. In the following paragraph, I am going to discuss all differences.

The data used in this experiment is the representative set described in section 3.1.2. It consists of 156 time-series. The lengths of time-series range from 400 to 8000 samples. Compared to the previous experiment, the forecasting horizon was defined as 25 steps. In order to get more robust estimates of the test performance and avoid the situations which are illustrated in figure 2.5, I applied the training/test split mechanism illustrated in Figure 4.7. I defined the last three lengths of the forecasting horizon as the test set. More specifically, the last $3 * 25$ observations

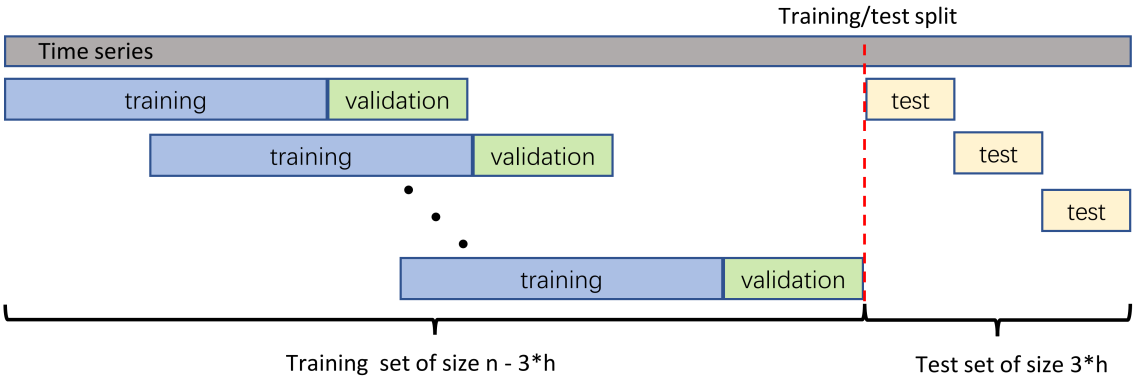


Figure 4.7: Training/test split. n is the length of the time-series. h is the size of forecasting horizon.

were used for testing the forecasting accuracy and the first $n - 3 * 25$ observations for training and validating the models. A single estimate of the test performance was produced by taking the average of the three test results.

To keep the computation bearable and the results interpretable, I included one more regression model. I used a total of four models, namely the SVR regression model, the RID regression model, the XGB regression model and the RF regression model. The range of related hyper-parameters are listed in section 3.2.4. The number of generated initial configurations by meta-learner was defined as 20.

In the off-line phase, I ran SMAC for 10 hours to optimize each candidate regression model. I did that for each time-series of the representative set. Lastly, I stored the instantiation which had the best performance on the test set for each time-series.

During the on-line phase, I applied the leave-one-dataset-out strategy again, to evaluate the performance of SMAC with meta-learner. When a time-series is being evaluated, I assumed that the other 155 time-series and their best hyper-parameter configurations are known. Due to the randomness of SMAC and random search, I repeated each optimization ten times on each time-series.

Result

The result of this experiment is illustrated in figure 4.8 and figure 4.9. I applied the two ranking-based method for the evaluation. Both evaluations showed, that the two meta-learning methods helped SMAC yield a drastic performance improvement at the beginning of the optimization. At the 50th evaluation, I noticed that the SMAC without meta-learning did not catch up to the SMAC with meta-learning. This indicates that meta-learning not only can help accelerate the optimization, but it can also provide a good basis for further optimization.

Figure 4.10 illustrates an example of the qualitative performance of the four optimization procedures on one representative time-series. This time-series comes from

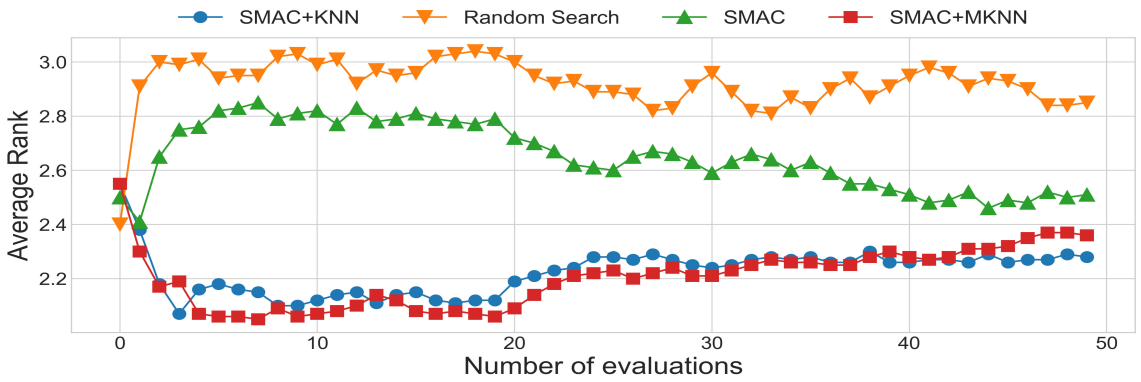


Figure 4.8: Average ranks of the four optimizers across the representative set. Ranking is determined by the mean of the best validation value.

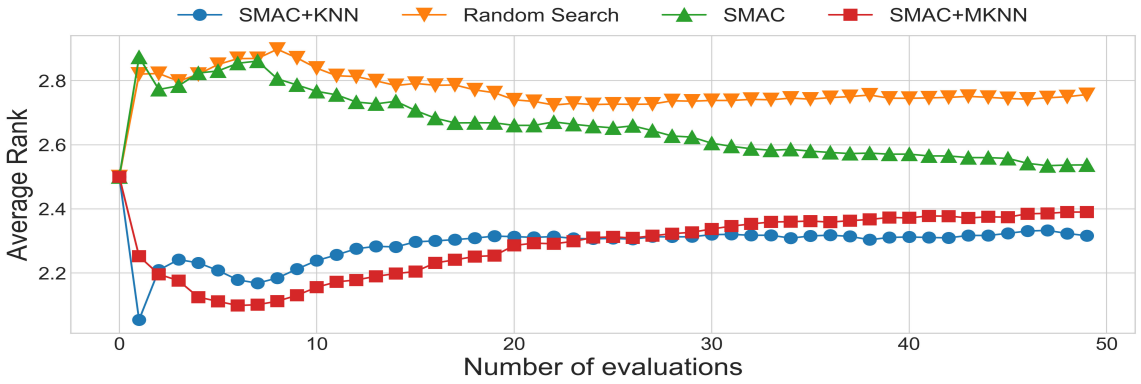


Figure 4.9: Average ranks of the four optimizers over all bootstrap sampled combinations. For each time-series, I bootstrap sampled 1000 combinations.

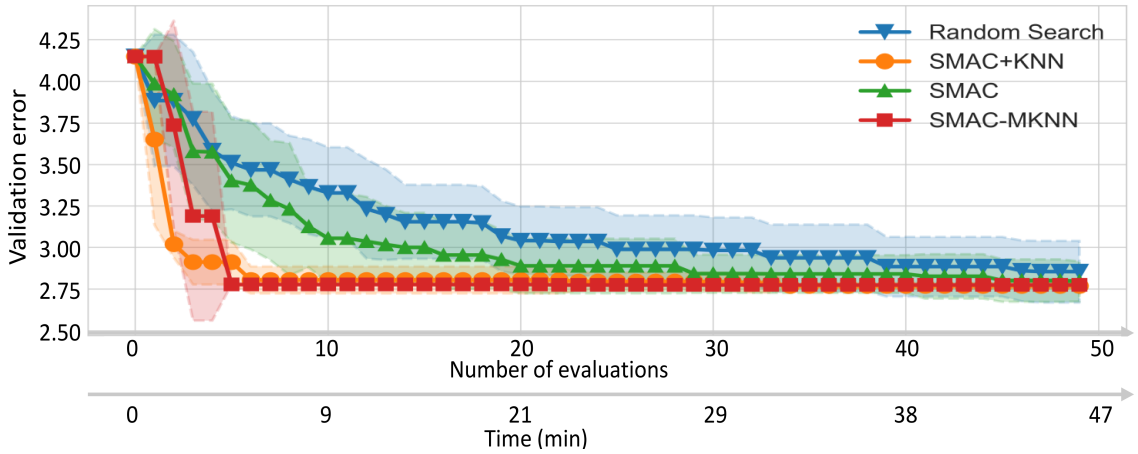
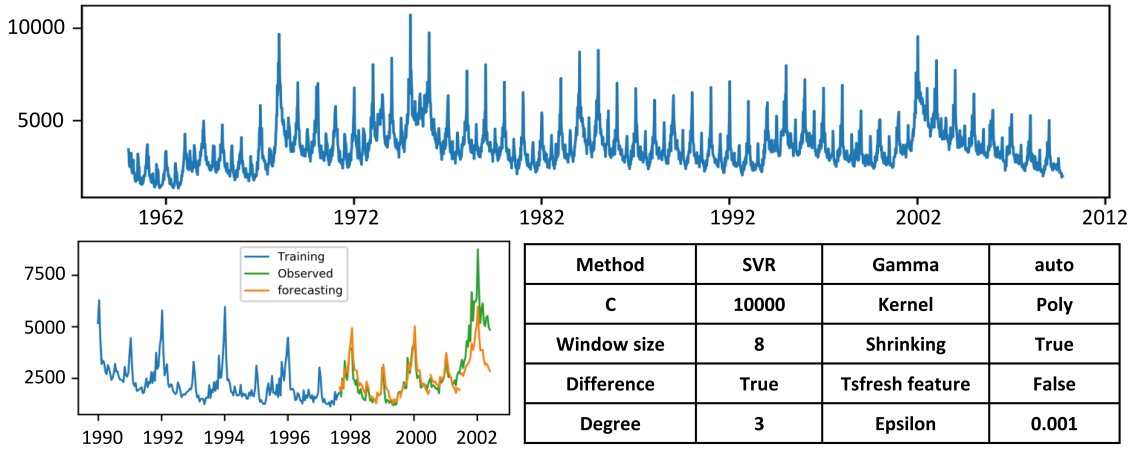


Figure 4.10: Qualitative performance of the four optimization procedures on the time-series "week 3" of M4-competition.



Method	SVR	Gamma	auto
C	10000	Kernel	Poly
Window size	8	Shrinking	True
Difference	True	Tsfresh feature	False
Degree	3	Epsilon	0.001

Figure 4.11: Details of time-series "week 3"

the M4-competition [39]. It contains 2596 weekly samples. I added a time-line in this figure, which represents the average time spent, up to the related evaluation steps. Each evaluation step includes the process of 10-fold walk-forward cross-validation and the process of SMAC optimization.

Figure 4.10 shows that, around the seventh evaluation, SMAC with meta-learner found a good solution, which took around 6 minutes. In contrast, SMAC without meta-learner used about 40 evaluations to find an identical qualitative solution, which took about 40 minutes. In this case, meta-learning is a good complement to SMAC optimization. Figure 4.11 shows the details of this time-series, as well as the prediction results and the final setting of hyper-parameters.

4.2.3 Summary

Regarding these two experiments, it can be summarized that meta-learning turned out to be a good supplementary to the SMAC optimization. It can effectively save

evaluation steps and reduce optimization time. Its excellent performance comes from the following two reasons.

(1) The applied meta-features from the TSFRESH packages represent time-series well.

(2) KNN-based meta-learner makes full use of meta-features to find the relationship between datasets. The suggestion from KNN-based meta-learner is a set which contains the best model and the configuration of all hyper-parameters. It does not need to learn the relationship between hyper-parameters.

4.3 Relationship between hyper-parameters and datasets

As mentioned earlier, I am now going to explain the meta-learning method RMH in more detail. But before I go on, I would like to briefly review this method. I built a model for each hyper-parameter of the candidate models. The type of model depends on the type of hyper-parameter. If the hyper-parameter is continuous, the model is a regression model. If the hyper-parameter is categorical, the model is a classification model. The input of the model is the meta-features vector of the time-series dataset.

However, this method did not achieve good results in the previous experiment. So in this section, I conducted an experiment to test the dependence between hyper-parameters and datasets and I analysed the reason behind the failure of this method.

The dataset used in this experiment was the representative set of 7000 collected time-series, which is described in section 3.1.2. The representative set has more diversity and less redundancy. I assumed that datasets with a large variance lead to a large variance in hyper-parameters. Such properties of datasets could help facilitate the study of the relationship between hyper-parameters and datasets. More precisely, like the concept of the KNN meta-learning method, the best parameter configuration for a dataset could probably perform excellent on another similar dataset. If the dataset has a low level of diversity, the parameters would be located at a small range. The relationship between hyper-parameters and dataset can be easily affected by noise.

I used the set of length independent features described in appendix A as the features vector to represent time-series. After filtering out the features which were unique or contained NAN, each time-series was represented by 324 features.

In this experiment, I took the SVR model as an example. The hyper-parameters of SVR and their respective ranges are listed in table 3.10. For each time-series, I ran SMAC for 24 hours with 10-fold walk-forward cross-validation to optimize the

hyper-parameters of SVR. The forecasting horizon was defined as 25. And I used the SMAPE as evaluation metric.

I applied hypothesis tests to check the statistical dependence between hyper-parameters and meta-features. Depending on the co-domains of the hyper-parameter and the feature, different statistical tests were applied. More specifically, if both the hyper-parameter and the feature were binary, I used Fisher's exact test [21]. If either the hyper-parameter or the feature was continuous, and the other one was binary, the Kolmogorov-Smirnov test [41] was executed. If both were continuous, the Kendal rank test [37] was applied. All used hypothesis tests are available in the TSFRESH package.

It resulted that no dependence existed between any features and any hyper-parameters. The resulting p-values of all hypothesis tests were more than 0.05. Big p-values indicate that related features are irrelevant for the target hyper-parameter.

I used PCA to reduce these features into two dimensions, in order to visualize the relationship between meta-features and hyper-parameters. Figure 4.12 and 4.13 demonstrate the relationship between some representative hyper-parameters and the first two principal components. The two figures indicate that no relationship can be observed.

	C	Kernel	Window_size	Epsilon	Shrinking	Tol	Tsfresh_feature	Difference	gamma
a	0.08	linear	67	0.001	true	0.00076	true	true	\
b	191	linear	57	0.001	true	0.0003	false	true	\
c	13	rbf	91	0.059	true	0.00291	false	true	”auto”
d	107	rbf	59	0.007	true	0.0231	flase	ture	”auto”

Table 4.3: Four configurations for SVR, which have the same validation performance.

Further, I ran SMAC four times for 24 hours on one time-series, which is shown in the top plot in figure 4.14. The optimized validation error of four runs were all roughly 0.993. But the final optimized hyper-parameter configurations of the four runs were entirely different. The four configurations are listed in table 4.3. To visualize the relationship between validation performance and parameters, I take penalty parameter C as an example. I varied the parameter C while keeping the other parameters unchanged. Figure 4.14 illustrates the validation performance curve of the four different configurations. It clearly shows that the parameter C is located in different ranges (sub-optimal) for the same time-series. Like parameter C, all other parameters got different values for the same time-series. The performance surface of all parameters is complicated. It is not convex, not smooth and contains many local optimal solutions.

The complexity of the performance surface stems from a metrical peculiarity. For the task of prediction, it is common to calculate the mean of the error among the

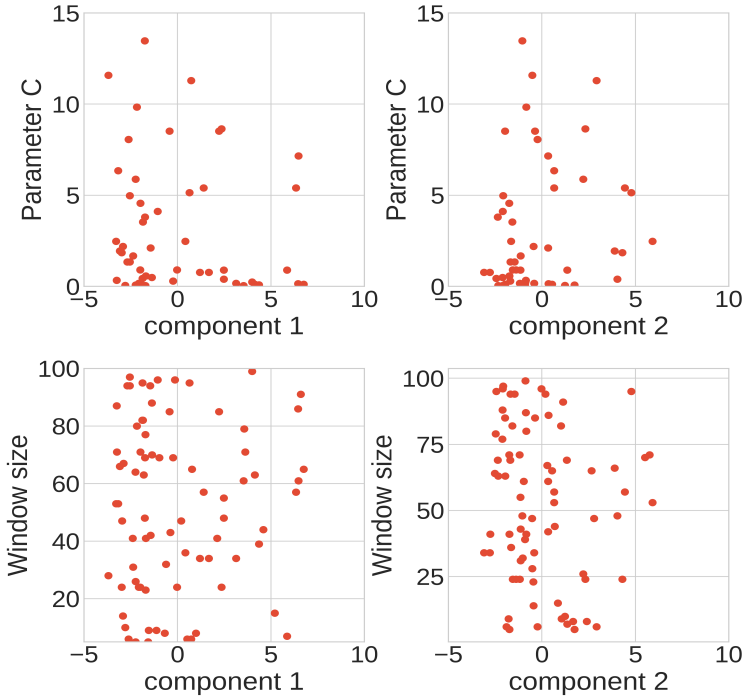


Figure 4.12: Relationship between the first two principal components and two hyper-parameters (penalty parameter C and window size). The C parameter ranges from 0.01 to 20,000. Only part of the entire space of parameter C is displayed in the sub-figures.

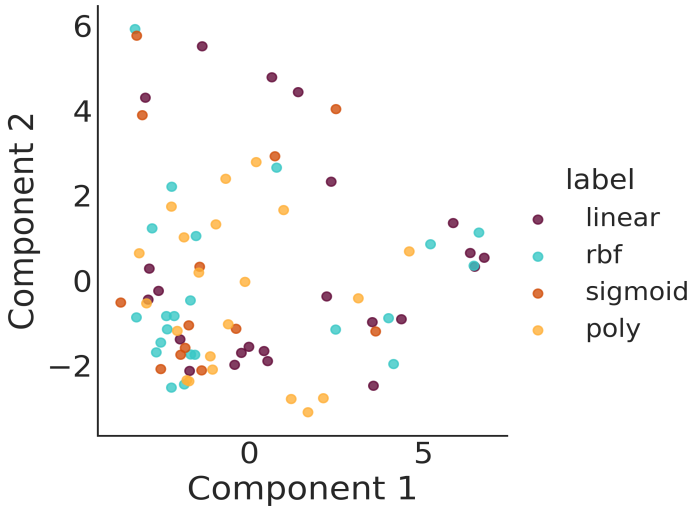


Figure 4.13: Relationship between the first two principal components and the categorical hyper-parameter kernel.

forecasting horizon. Big error values can be balanced by small error values. To clarify what I mean by that, I created figure 4.16. In this illustration, the blue curve represents the real value, whereas the red and green graph represent forecasting

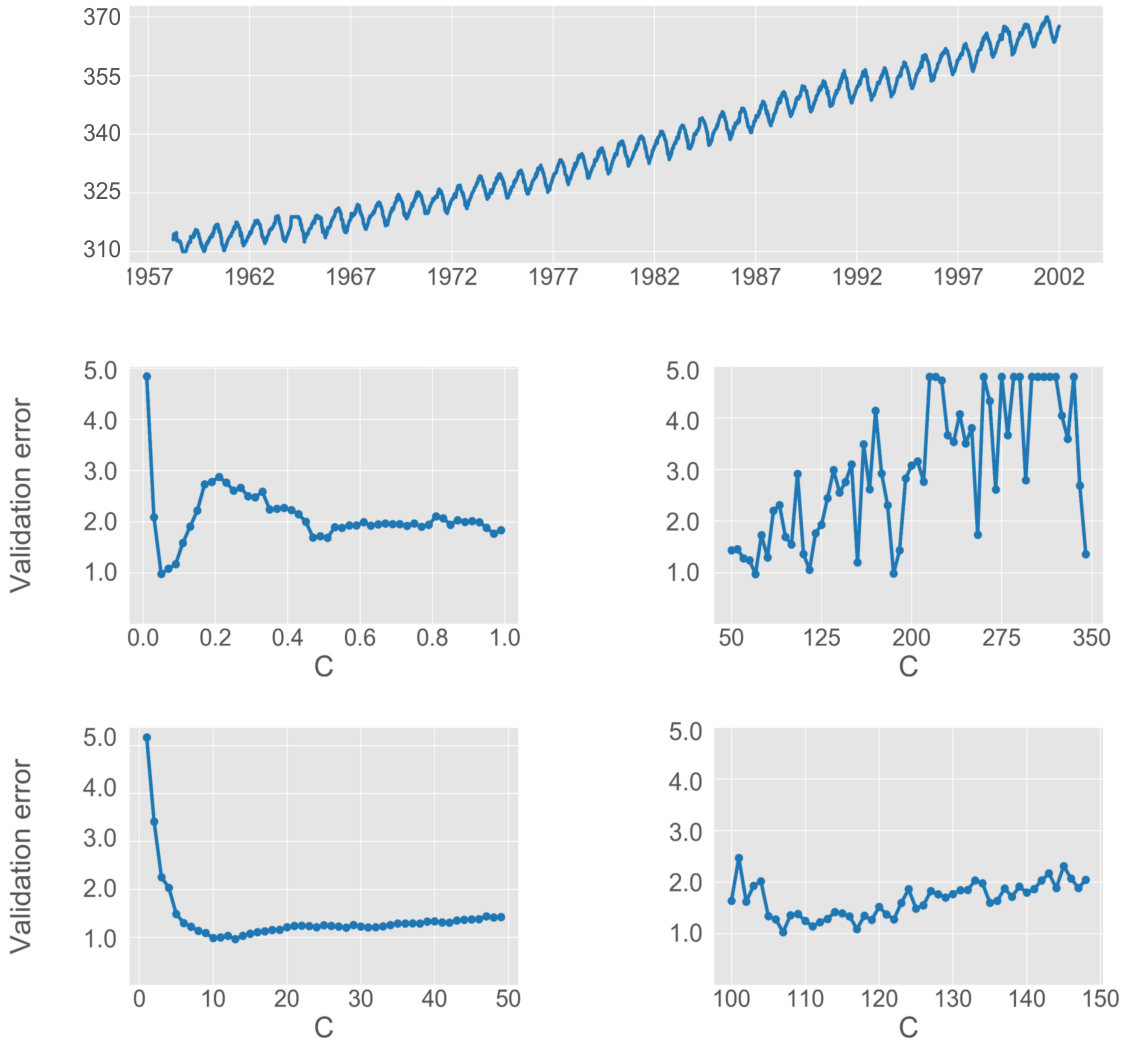


Figure 4.14: Relationship between the penalty parameter C and the validation performance.

horizons. Both forecasting horizons have the same SMAPE, which is 1/3. There are infinite constellations which all result in the same SMAPE. This property of evaluation metric results in different sets of prediction values, potentially having the same evaluation value. Therefore, SMAC landed at different local optimums. Figure 4.16 visualizes the prediction curve of the four different parameter configurations. All four curves are slightly different but have the same SMAPE.

As a conclusion, it is not feasible to learn the relationship between the datasets and the hyper-parameters separately, because it is difficult to find the global optimum and the relationship between the hyper-parameters is neglected.

4.4 Experiment of forecasting performance

In this section, I drew a comparison between my forecasting framework and other three widely used models, to study the forecasting performance of my forecasting

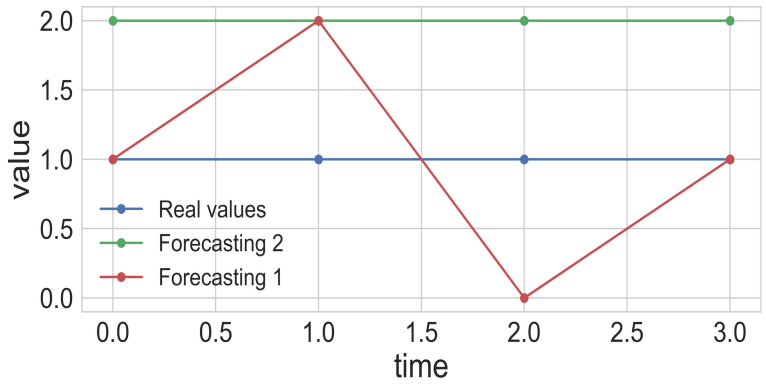


Figure 4.15: Example to describe the cause of complex performance surfaces.

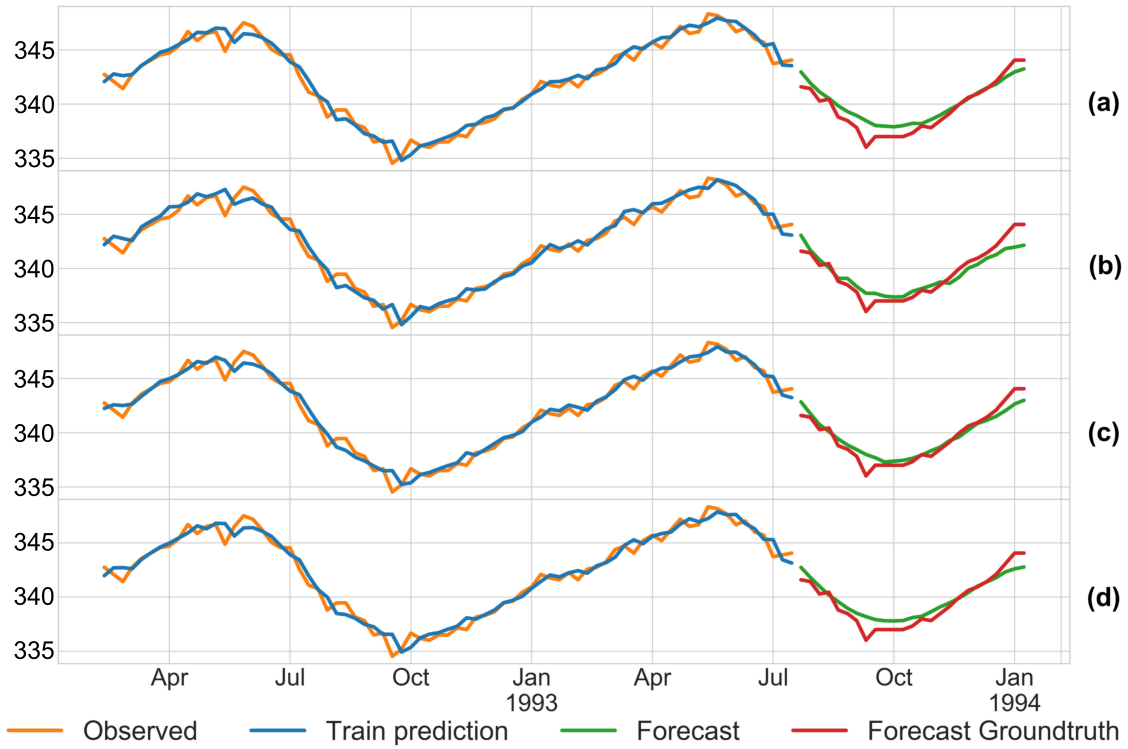


Figure 4.16: Prediction curve of the four parameter configurations. Although each prediction curve is slightly different, the evaluation metric treated the four different curves the same. The letters on the right side correspond to the configurations listed in table 4.3.

framework. The models are: the seasonal ARIMA model, artificial neural networks (ANN) model with Long short-term memory (LSTM) units and the Prophet model.

Experiment set-up

In this experiment, I collected 87 time-series of different lengths and sampling rates from different domains. The maximal periodicity of the collected 87 time-series is 365. The forecasting horizon was defined as 25 steps. The first $n - 75$ observations were used for training and validating the models. The last 75 observations were used for testing their forecasting accuracy. I chose SMAPE as the evaluation metric. The

meta-level dataset of my framework was built based on the representative set. This means when a time-series is being evaluated, the remaining 86 time-series are not available.

Seasonal ARIMA is a classic statistical prediction model. I used a python open source package called statsmodels to implement my ARIMA model. I considered a total of 8 hyper-parameters. These hyper-parameters and their respective range are listed in table 4.4. The common practice is to adjust the hyper-parameters of the ARIMA model by observing the autocorrelation function (ACF) plot and partial autocorrelation function (PACF) plot or using the Akaike's information criterion (AIC). In this experiment, I used grid search method to choose the hyper-parameters of the ARIMA model. Principally, all possible combinations of hyper-parameters were tried and the one with the smallest AIC was selected. After the selection of hyper-parameters, the observations up until the test set were trained to predict the values of 25 future steps. Then the average of the three test results were taken to produce an estimation of the forecasting performance.

	Notation	Range	Type
AR parameter	p	[0, 5]	int
Degree of differencing	d	[0, 2]	int
MA parameter	q	[0, 5]	int
Seasonal AR parameter	P	[0, 1]	int
Degree of seasonal differencing	D	[0, 1]	int
Seasonal MA parameter	Q	[0, 1]	int
Periodicity	m	manual defined	int

Table 4.4: Hyper-parameters of the seasonal ARIMA model. Periodicity parameters were manually defined according to each time-series. Parameter P, D, Q are conditional to the parameter periodicity. They are only valid when the periodicity is unequal zero.

A LSTM network is an artificial neural network that contains LSTM units instead of, or in addition to, other network units. A LSTM unit is a recurrent network unit that is excellent at remembering values for either long or short durations of time. In this experiment, I used a single layer LSTM network with 256 hidden units. The input is 100 past values. This means that the dimension of input is 100. Because I used the recursive strategy to perform the multi-steps prediction, the dimension of the output layer is 1. The default sigmoid activation function is used for the LSTM units. I set the dropout as 0.2 to avoid over-fitting. The network was trained for 4000 steps, with a batch size of 10.

Prophet is an automatic procedure for forecasting time-series data based on an additive model. It is an open source software, released by Facebook's Core Data Science team. It is a tool for producing high quality forecasts for time-series data, that has multiple seasonality with linear or non-linear growth. The use of Prophet

is simple. The only necessary action is to enter the time-series and the size of the forecasting horizon.

For my framework, I ran SMAC with 10-folds cross-validation on the training set of each time-series. The time limit set for each optimization was one hour. The included regression models are SVR model, XGB model, RID model and RF model.

Result

Table 4.5 lists the average performance of these four methods across all 87 time-series. The results show, that LSTM had the best overall performance, whereas my framework performed only slightly worse.

I analysed the results of my framework on every time-series. For 36.78% of all time-series, my framework performed the best. Only for 2.98% of all time-series, it performed the worst. This indicated, that my framework turned out to be very versatile and robust, in that it performs consistently well under various conditions. My framework can create good models for all situations such as seasonal, non-seasonal, with noise, without noise, etc.. To show more details of the forecasting performance, I created the following figures. In the figures, "AutoML" stands for my framework.

	My framework	LSTM	Prophet	ARIMA
Average SMAPE	22.78	21.21	47.2	31.34

Table 4.5: Average SMAPE of 25 steps forecasting horizon

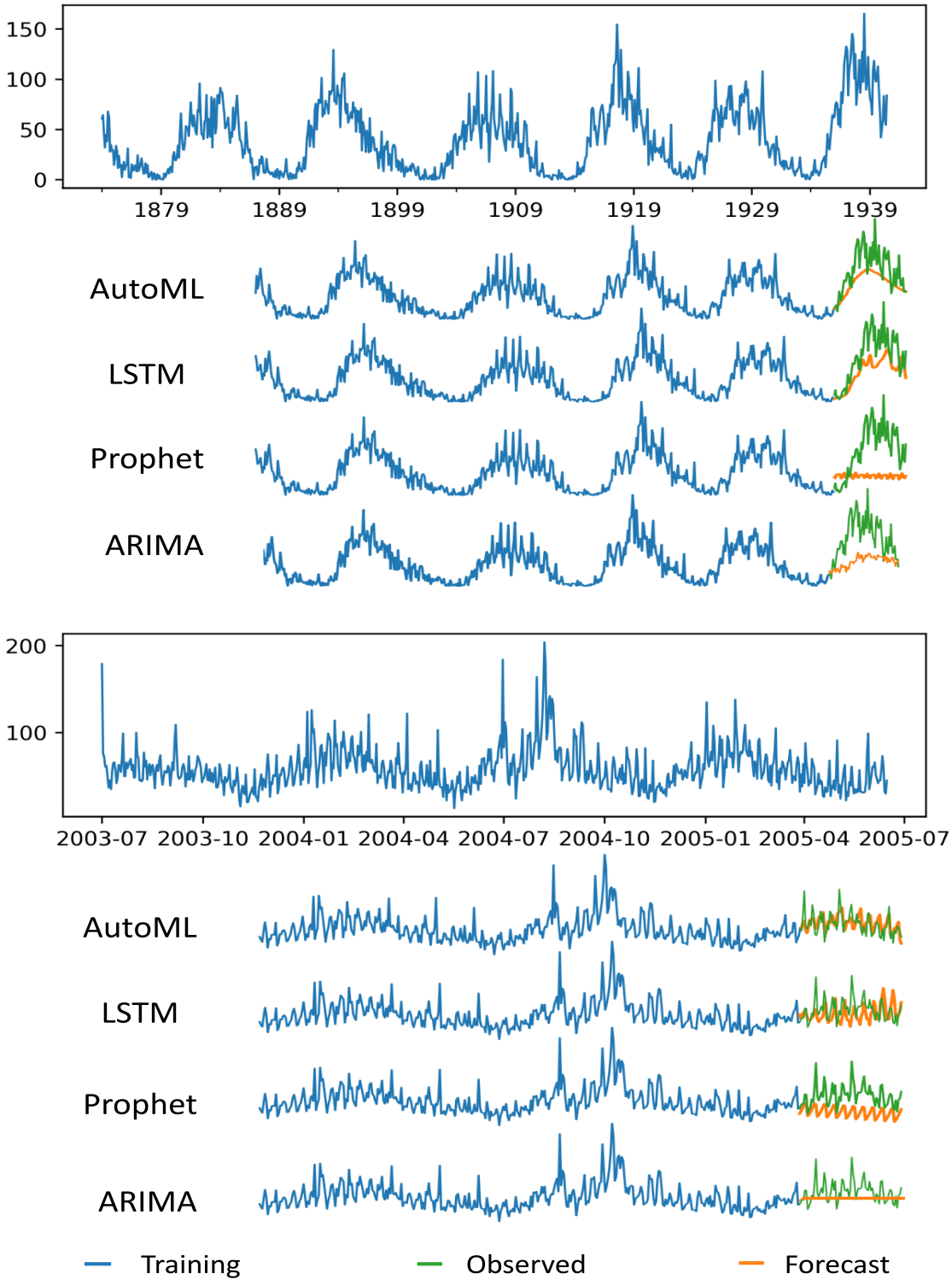


Figure 4.17: Top: Zurich monthly sunspot numbers. Bottom: daily number of vehicles in both directions of tunnel Andermatt-Goeschenen.

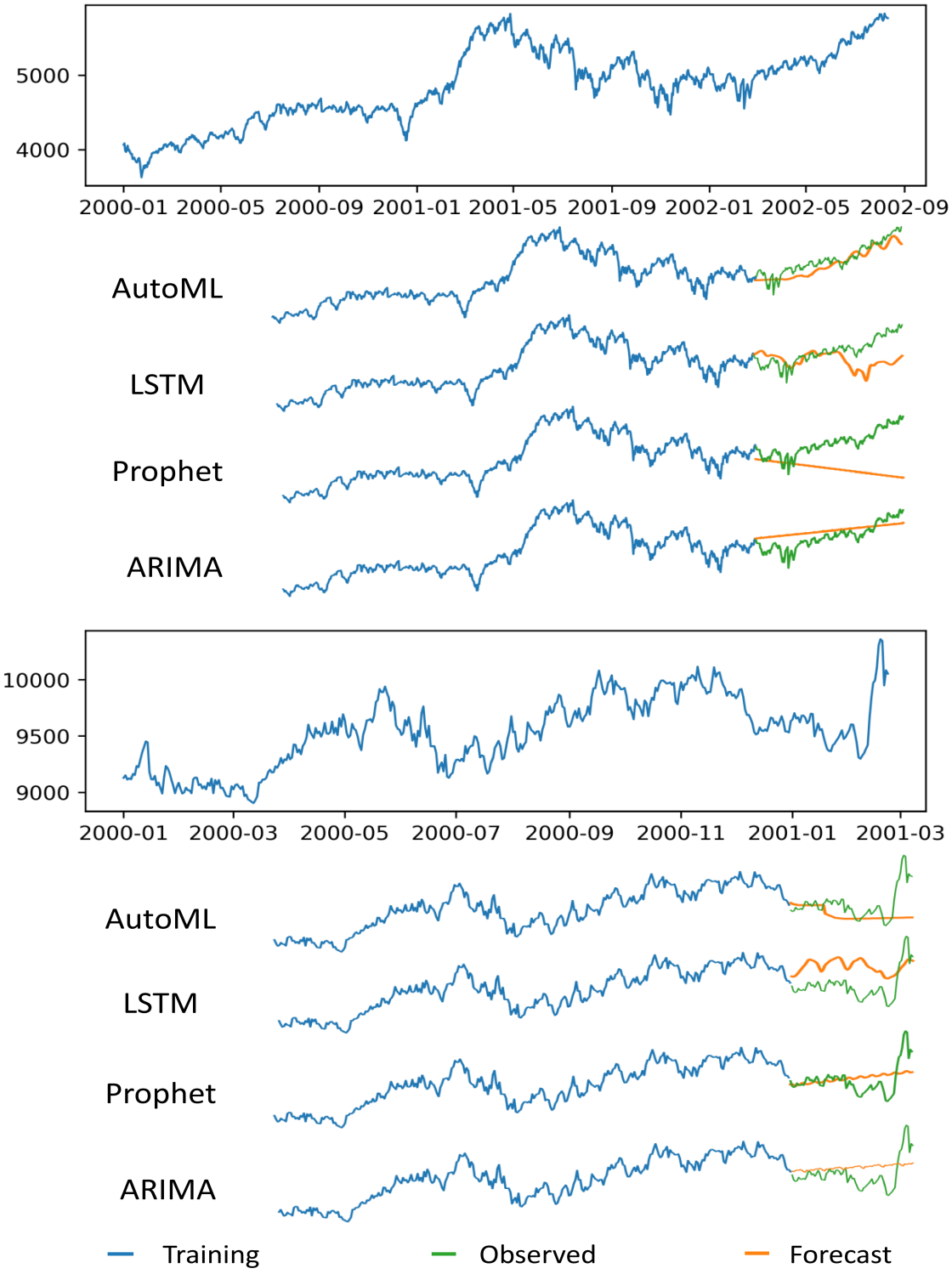


Figure 4.18: Top: daily financial time-series No. D2040 of the M4 competition. Bottom: daily industry time-series No. D2009 of the M4 competition.

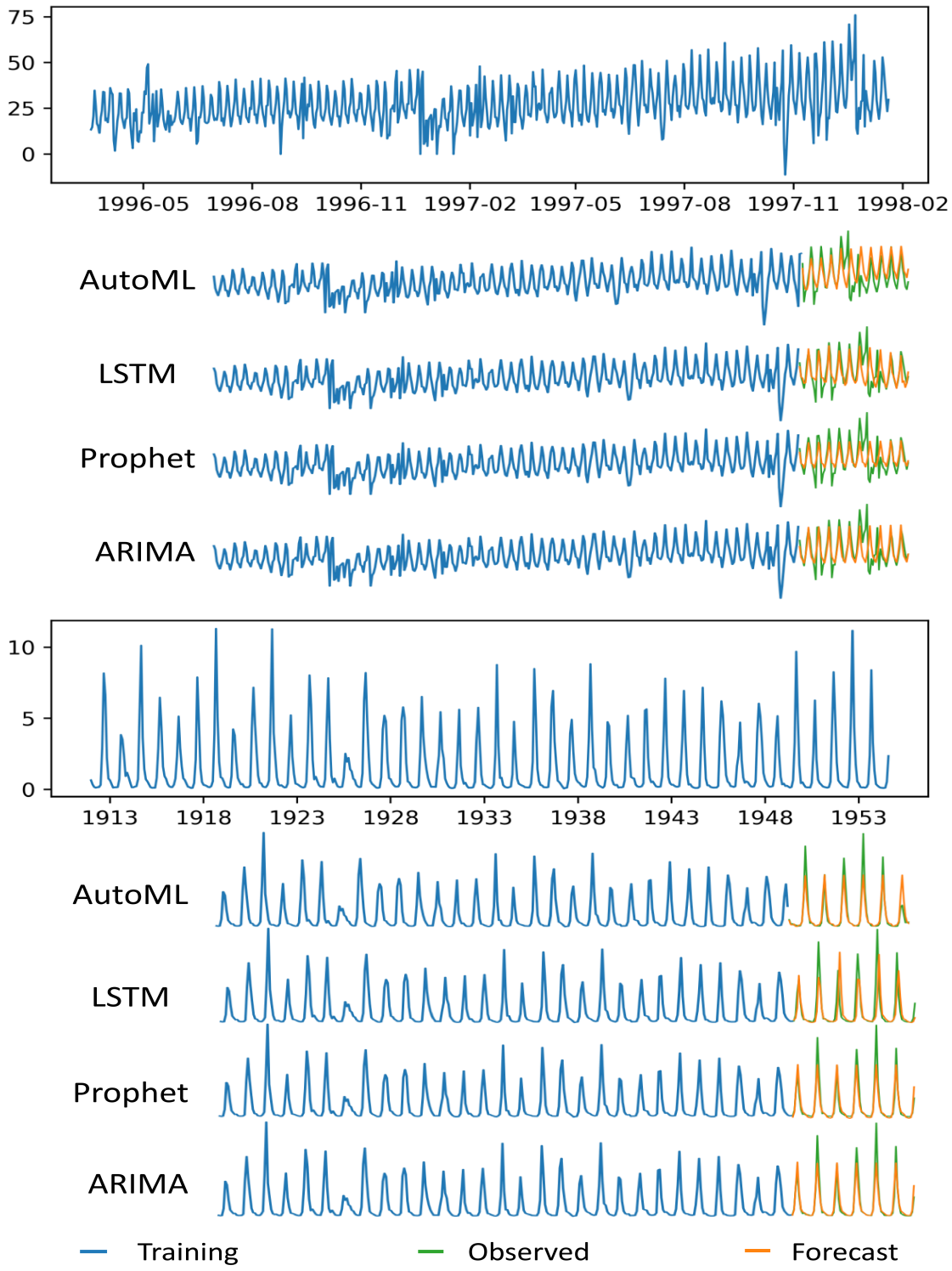


Figure 4.19: Top: daily time-series No.1 of NN5 forecasting competition. Bottom: monthly river flow in Madison river.

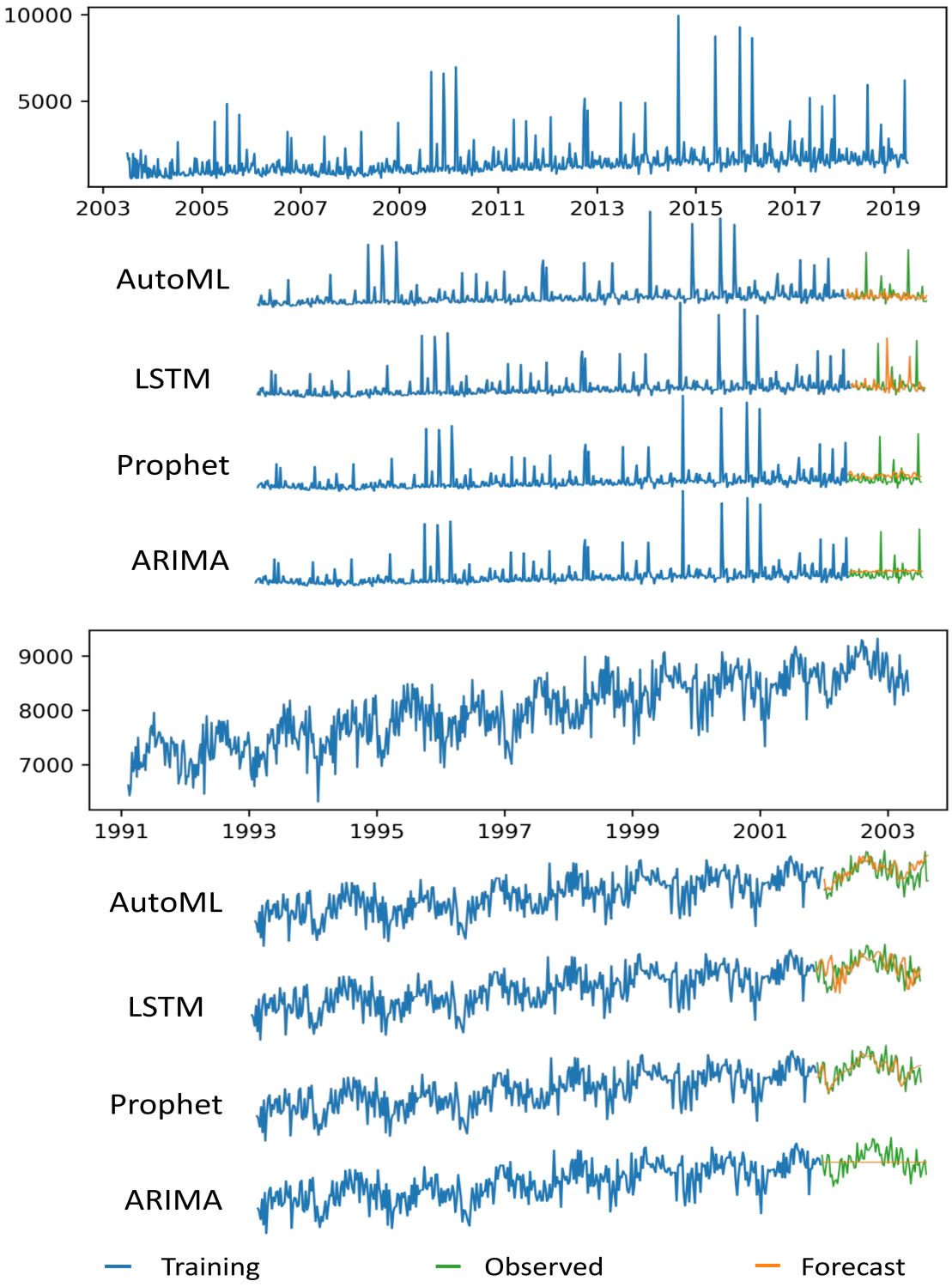


Figure 4.20: Top: weekly time-series No. w284 of the M4 competition. Bottom: weekly U.S. finished motor gasoline product supplied.

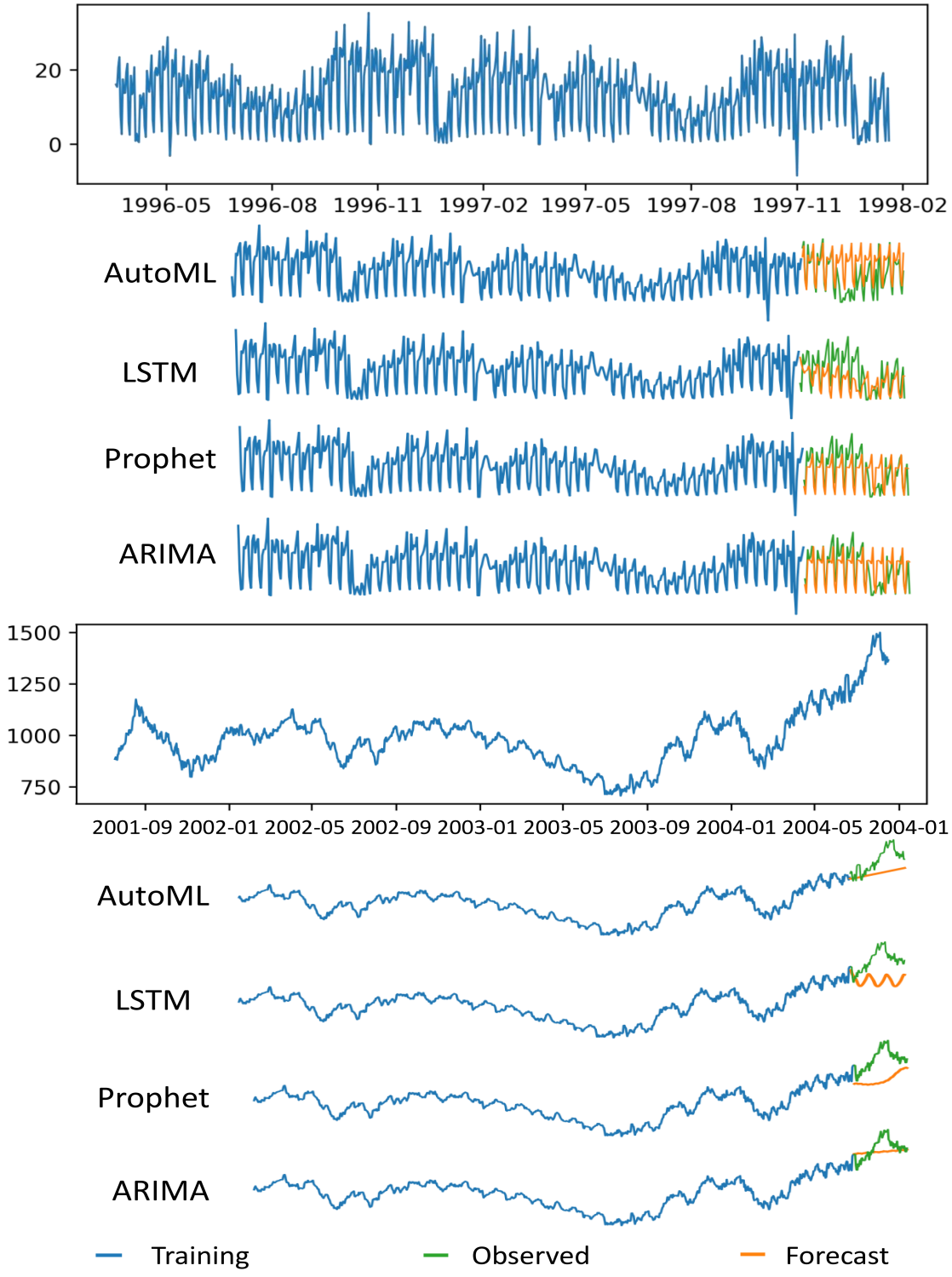


Figure 4.21: Top: daily time-series No.36 of NN5 forecasting competition. Bottom: daily industry time-series No. D2024 of the M4 competition.

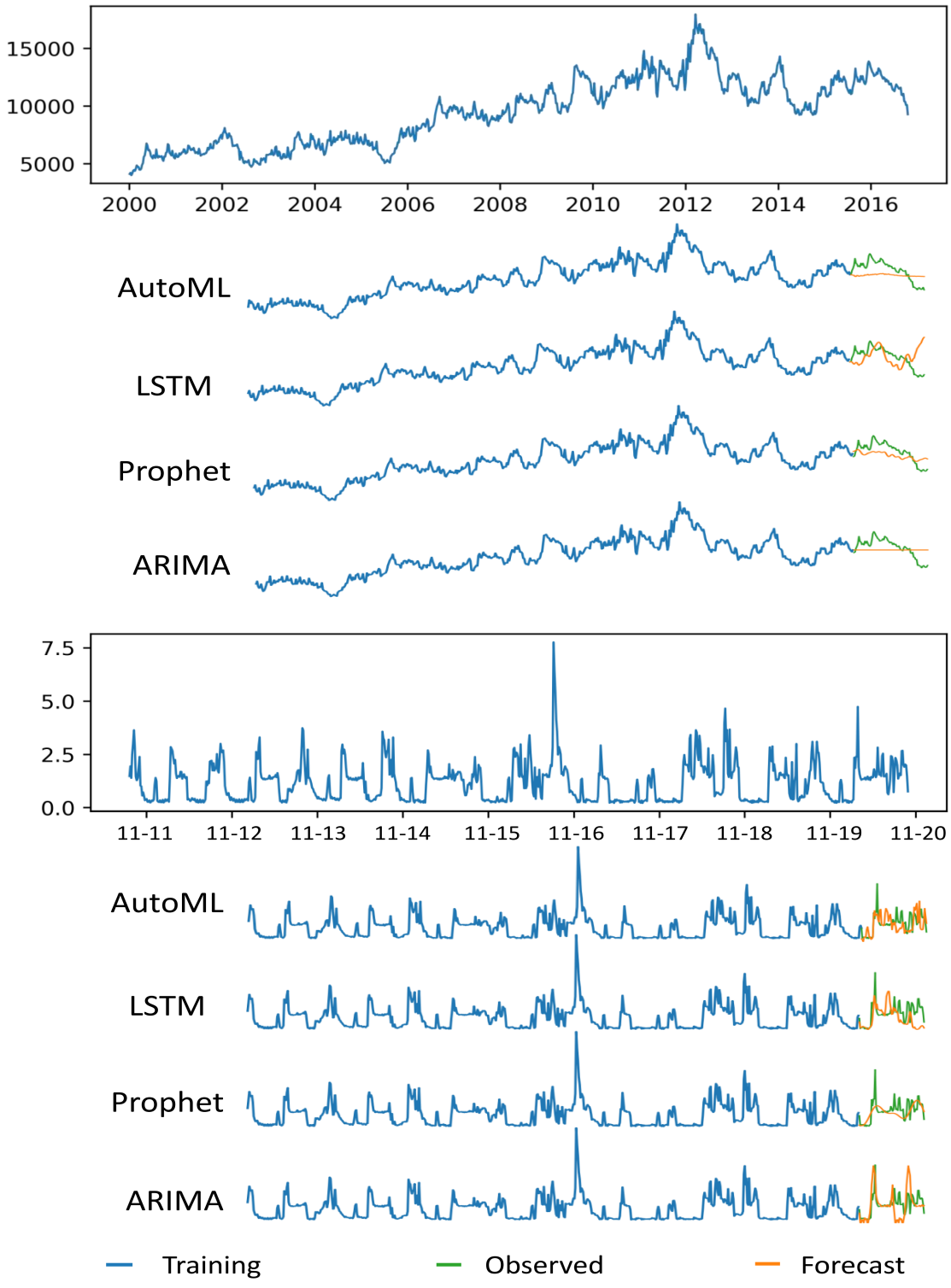


Figure 4.22: Top: weekly industry time-series No. W266 of the M4 competition. Bottom: minutely household electric power consumption.

5. Conclusion and Future Work

This final chapter consists of my conclusion as well as an outlook and suggestions to future projects in this field.

5.1 Conclusion

For this thesis, I have built an automated framework for time-series forecasting. By applying SMAC with the help of meta-learning, my framework produced excellent results for time-series forecasting tasks. I have conducted multiple experiments to demonstrate that SMAC can efficiently solve the CASH problem in the context of forecasting, as well as showing that meta-learning is a perfect complementary to SMAC.

I have used a total of three meta-learning methods. The suggestions from KNN meta-learner and its variant can effectively accelerate the optimization process of SMAC. The method, which learns the relationship between datasets and hyper-parameters, performed poorly.

Furthermore, I integrated the TSFRESH package in my framework, which assists the included ML models in my framework to capture the dynamical structure of time-series much more effectively.

5.2 Outlook

The experience I have gathered, over the course of writing this master thesis, has given me the following ideas that I would like to share for future work:

(1) The forecasting models included in my framework are all regression models from the package Sklearn. I used window sliding methods to adapt them to forecasting problem. There are many specific ML methods for forecasting models. When an

automated framework includes all specific forecasting ML models, the performance of the framework drastically improves.

(2) During my work, I have used a lot of time to gather time-series. The diversity of time-series datasets directly affects the performance of meta-learner. Using sufficient time-series to form the meta-level datasets for meta-learning is much more effective. In addition, the framework should have the ability to automatically update the meta-level datasets. With every completion of a forecasting task, there is a possibility of a good instantiation for meta-level datasets.

(3) The SMAC framework offers several degrees of freedom to be instantiated. Only KNN and its variants performed well in my work. It would be beneficial to extend the data library of different meta-learning method.

(4) For my thesis, I simply inserted all ML models into the optimization process, with the result, that the optimized hyper-parameter space becomes incredibly large. Building a hierarchical framework could prove advantageous. Filtering out the models with a high probability of performing badly on the first level, could reduce the hyper-parameters space. This would result in a faster optimization process.

(5) There are other optimization methods, which also perform very well in a high dimensional parameter space, such as generic programming. I used the Bayesian optimization for my framework. It would be expedient to try other optimization methods.

(6) Considering the computational expense, I used the recursive strategy to perform multi-steps forecasting in my framework. But the use of that strategy can accumulate errors along the forecasting horizon. However, if the computational budget is sufficient, I would suggest using the direct strategy to perform the multi-steps forecasting.

(7) Nowadays, many applications are related to multi-variant time-series. My work focused on uni-variant time-series. The research on the performance of automated machine learning, in the context of multi-variant time-series forecasting, could lead to significant developments in industry 4.0.

Bibliography

[1] Mathias M Adankon and Mohamed Cheriet. "Model selection for the LS-SVM. Application to handwriting recognition". In: *Pattern Recognition* 42.12 (2009), pp. 3264–3270.

[2] Nesreen K Ahmed et al. "An empirical comparison of machine learning models for time series forecasting". In: *Econometric Reviews* 29.5-6 (2010), pp. 594–621.

[3] Ralph G Andrzejak et al. "Indications of nonlinear deterministic and finite-dimensional structures in time series of brain electrical activity: Dependence on recording region and brain state". In: *Physical Review E* 64.6 (2001), p. 061907.

[4] Bay Arinze, Seung-Lae Kim, and Murugan Anandarajan. "Combining and selecting forecasting models using rule based induction". In: *Computers & operations research* 24.5 (1997), pp. 423–433.

[5] Rémi Bardenet et al. "Collaborative hyperparameter tuning". In: *International Conference on Machine Learning*. 2013, pp. 199–207.

[6] Thomas Bartz-Beielstein et al. *Experimental methods for the analysis of optimization algorithms*. Springer, 2010.

[7] Yoav Benjamini and Yosef Hochberg. "Controlling the false discovery rate: a practical and powerful approach to multiple testing". In: *Journal of the royal statistical society. Series B (Methodological)* (1995), pp. 289–300.

[8] James Bergstra and Yoshua Bengio. "Random search for hyper-parameter optimization". In: *Journal of Machine Learning Research* 13.Feb (2012), pp. 281–305.

[9] James S Bergstra et al. "Algorithms for hyper-parameter optimization". In: *Advances in neural information processing systems*. 2011, pp. 2546–2554.

[10] Gianluca Bontempi, Souhaib Ben Taieb, and Yann-Aël Le Borgne. "Machine learning strategies for time series forecasting". In: *European Business Intelligence Summer School*. Springer. 2012, pp. 62–77.

[11] Hamparsum Bozdogan. “Model selection and Akaike’s information criterion (AIC): The general theory and its analytical extensions”. In: *Psychometrika* 52.3 (1987), pp. 345–370.

[12] J Brownlee. “Introduction to time series forecasting with python”. In: *URL: https://machinelearningmastery. com/introduction-to-timeseries-forecasting-with-python* (2017).

[13] Ciro Castiello, Giovanna Castellano, and Anna Maria Fanelli. “Meta-data: Characterization of input features for meta-learning”. In: *International Conference on Modeling Decisions for Artificial Intelligence*. Springer. 2005, pp. 457–468.

[14] Maximilian Christ, Andreas W Kempa-Liehr, and Michael Feindt. “Distributed and parallel time series feature extraction for industrial big data applications”. In: *arXiv preprint arXiv:1610.07717* (2016).

[15] Fred Collopy and J Scott Armstrong. “Rule-based forecasting: Development and validation of an expert systems approach to combining time series extrapolations”. In: *Management Science* 38.10 (1992), pp. 1394–1414.

[16] Jan G De Gooijer and Rob J Hyndman. “25 years of time series forecasting”. In: *International journal of forecasting* 22.3 (2006), pp. 443–473.

[17] Frank Eibe, MA Hall, and IH Witten. “The WEKA Workbench. Online Appendix for” Data Mining: Practical Machine Learning Tools and Techniques”. In: *Morgan Kaufmann* (2016).

[18] Robert F Engle. “Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation”. In: *Econometrica: Journal of the Econometric Society* (1982), pp. 987–1007.

[19] Matthias Feurer, Jost Tobias Springenberg, and Frank Hutter. “Initializing Bayesian Hyperparameter Optimization via Meta-Learning.” In: *AAAI*. 2015, pp. 1128–1135.

[20] Matthias Feurer et al. “Efficient and robust automated machine learning”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2962–2970.

[21] Ronald A Fisher. “On the interpretation of χ 2 from contingency tables, and the calculation of P”. In: *Journal of the Royal Statistical Society* 85.1 (1922), pp. 87–94.

[22] Benjamin D Fulcher. “Highly comparative time-series analysis”. PhD thesis. University of Oxford, 2012.

[23] XC Guo et al. “A novel LS-SVMs hyper-parameter selection based on particle swarm optimization”. In: *Neurocomputing* 71.16-18 (2008), pp. 3211–3215.

[24] James Douglas Hamilton. *Time series analysis*. Vol. 2. Princeton university press Princeton, NJ, 1994.

[25] Zena M Hira and Duncan F Gillies. "A review of feature selection and feature extraction methods applied on microarray data". In: *Advances in bioinformatics* 2015 (2015).

[26] Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "Sequential model-based optimization for general algorithm configuration". In: *International Conference on Learning and Intelligent Optimization*. Springer. 2011, pp. 507–523.

[27] Frank Hutter et al. "An experimental investigation of model-based parameter optimisation: SPO and beyond". In: *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM. 2009, pp. 271–278.

[28] Frank Hutter et al. "Sequential model-based parameter optimization: An experimental investigation of automated and interactive approaches". In: *Experimental Methods for the Analysis of Optimization Algorithms*. Springer, 2010, pp. 363–414.

[29] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice*. OTexts, 2018.

[30] Aapo Hyvärinen and Erkki Oja. "Independent component analysis: algorithms and applications". In: *Neural networks* 13.4-5 (2000), pp. 411–430.

[31] Mingfeng Jiang et al. "Study on parameter optimization for support vector regression in solving the inverse ECG problem". In: *Computational and mathematical methods in medicine* 2013 (2013).

[32] ZQ John Lu. "The elements of statistical learning: data mining, inference, and prediction". In: *Journal of the Royal Statistical Society: Series A (Statistics in Society)* 173.3 (2010), pp. 693–694.

[33] Eric Jones, Travis Oliphant, and Pearu Peterson. "{SciPy}: open source scientific tools for {Python}". In: (2014).

[34] Alexandros Kalousis and Theoharis Theoharis. "Noemon: Design, implementation and performance results of an intelligent assistant for classifier selection". In: *Intelligent Data Analysis* 3.5 (1999), pp. 319–337.

[35] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.

[36] Robert E Kass. "Nonlinear regression analysis and its applications". In: *Journal of the American Statistical Association* 85.410 (1990), pp. 594–596.

[37] Maurice G Kendall. "A new measure of rank correlation". In: *Biometrika* 30.1/2 (1938), pp. 81–93.

[38] Spyros Makridakis and Michele Hibon. “The M3-Competition: results, conclusions and implications”. In: *International journal of forecasting* 16.4 (2000), pp. 451–476.

[39] Spyros Makridakis, Evangelos Spiliotis, and Vassilios Assimakopoulos. “The M4 Competition: Results, findings, conclusion and way forward”. In: *International Journal of Forecasting* (2018).

[40] Stephen Marsland. *Machine learning: an algorithmic perspective*. Chapman and Hall/CRC, 2011.

[41] Frank J Massey Jr. “The Kolmogorov-Smirnov test for goodness of fit”. In: *Journal of the American statistical Association* 46.253 (1951), pp. 68–78.

[42] Steffen Moritz and Thomas Bartz-Beielstein. “imputeTS: time series missing value imputation in R”. In: *The R Journal* 9.1 (2017), pp. 207–218.

[43] Daniel Müllner. “Modern hierarchical, agglomerative clustering algorithms”. In: *arXiv preprint arXiv:1109.2378* (2011).

[44] Ajoy K Palit and Dobrivoje Popovic. *Computational intelligence in time series forecasting: theory and engineering applications*. Springer Science & Business Media, 2006.

[45] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of machine learning research* 12.Oct (2011), pp. 2825–2830.

[46] Martin Pelikan, David E Goldberg, and Erick Cantú-Paz. “BOA: The Bayesian optimization algorithm”. In: *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 1*. Morgan Kaufmann Publishers Inc. 1999, pp. 525–532.

[47] Ricardo BC Prudêncio and Teresa B Ludermir. “Meta-learning approaches to selecting time series models”. In: *Neurocomputing* 61 (2004), pp. 121–137.

[48] Dorian Pyle. *Data preparation for data mining*. morgan kaufmann, 1999.

[49] J Ross Quinlan. *C4. 5: programs for machine learning*. Elsevier, 2014.

[50] Erhard Rahm and Hong Hai Do. “Data cleaning: Problems and current approaches”. In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 3–13.

[51] John R Rice. “The algorithm selection problem”. In: (1975).

[52] Chandra Shah. “Model selection in univariate time series forecasting using discriminant analysis”. In: *International Journal of Forecasting* 13.4 (1997), pp. 489–500.

[53] Robert H Shumway and David S Stoffer. “Time series regression and exploratory data analysis”. In: *Time series analysis and its applications*. Springer, 2011, pp. 47–82.

[54] Priyanga Talagala, Rob Hyndman, George Athanasopoulos, et al. *Meta-learning how to forecast time series*. Tech. rep. Monash University, Department of Econometrics and Business Statistics, 2018.

[55] Yuan Yan Tang, Chang De Yan, and Ching Y. Suen. "Document processing for automatic knowledge acquisition". In: *IEEE transactions on Knowledge and Data Engineering* 6.1 (1994), pp. 3–21.

[56] Sean J Taylor and Benjamin Letham. "Forecasting at scale". In: *The American Statistician* 72.1 (2018), pp. 37–45.

[57] Joshua B Tenenbaum, Vin De Silva, and John C Langford. "A global geometric framework for nonlinear dimensionality reduction". In: *science* 290.5500 (2000), pp. 2319–2323.

[58] Chris Thornton et al. "Auto-WEKA: Automated selection and hyper-parameter optimization of classification algorithms". In: *CoRR, abs/1208.3719* (2012).

[59] Chris Thornton et al. "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2013, pp. 847–855.

[60] Michael E Tipping and Christopher M Bishop. "Probabilistic principal component analysis". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 61.3 (1999), pp. 611–622.

[61] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial Intelligence Review* 18.2 (2002), pp. 77–95.

[62] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial Intelligence Review* 18.2 (2002), pp. 77–95.

[63] Ricardo Vilalta and Youssef Drissi. "A perspective view and survey of meta-learning". In: *Artificial Intelligence Review* 18.2 (2002), pp. 77–95.

[64] Robert J Vokurka, Benito E Flores, and Stephen L Pearce. "Automatic feature identification and graphical support in rule-based forecasting: a comparison". In: *International Journal of Forecasting* 12.4 (1996), pp. 495–512.

[65] Xiaozhe Wang, Kate Smith-Miles, and Rob Hyndman. "Rule induction for forecasting method selection: Meta-learning the characteristics of univariate time series". In: *Neurocomputing* 72.10-12 (2009), pp. 2581–2594.

[66] Rui Xu and Donald Wunsch. "Survey of clustering algorithms". In: *IEEE Transactions on neural networks* 16.3 (2005), pp. 645–678.

[67] Nien Fan Zhang. *Forecasting and time series analysis*. 1992.

[68] Peng Zhao and Bin Yu. "On model selection consistency of Lasso". In: *Journal of Machine learning research* 7.Nov (2006), pp. 2541–2563.

[69] Eric R Ziegel. “Computational Intelligence in Time Series Forecasting”. In: *Technometrics* 48.3 (2006), p. 451.

A. Appendix considered feature mappings

The list of features in TSFRESH package can be found at http://tsfresh.readthedocs.io/en/latest/. Here I only list custom feature mappings.

B. Appendix of included time series

This appendix summarizes the time-series which i used to compose the my dataset.

Neural Networks Forecasting Competition (NN3). NN3 is a forecasting competition for 2006-2007. It contains a complete dataset of 111 monthly time-series drawn from homogeneous population of empirical business time series. Data can be downloaded from the following website:

http://www.neural-forecasting-competition.com/NN3/datasets.htm.

Time-Series Data Library (TSDL). The TSDL was created by Rob Hyndman, Professor of Statistics at Monash University, Australia. There are many time-series available, which are drawn from different categories, such as finance, computing, transport and tourism, micro-economic, production, demography. Data can be downloaded from the following website:

https://datamarket.com/data/list/?q=cat:ecc

U.S. Department of State Data Use Statement. It applies to data available from the Mission China air quality monitoring program, which includes the hourly PM 2.5 observation data from 2009 to 2016 in Beijing. Data can be downloaded from the following website:

website is: http://www.stateair.net/web/historical/1/1.html

M Competition. M competitions have an enormous influence on the field of forecasting. In total, "M" competitions were held four times. In my work, I used

time-series from M3 competition and M4 competition. M3 involves 3003 time series. M4 provides a large collection of 100,000 time-series. The time-series from the competition come mainly from the industry, finance, demographics and economics, while also including data from tourism, trade, labor and wage, transportation and the environment. Data can be downloaded from the following website:

M3: https://forecasters.org/resources/time-series-data/m3-competition/

M4: https://www.m4.unic.ac.cy/the-dataset/

Google trends. Google Trends is a public web facility of Google Inc. Time-series can be obtained with keywords, such as "amazon", "ebay", "travel" and "vegetarian". The website of google trends is

https://trends.google.com/trends/?geo=US

UCI Machine Learning Repository. UCI Machine Learning Repository is a great source of multivariate time-series data. Each column of the datasets can be viewed as a univariate time-series. Data can be downloaded from the following website:

http://archive.ics.uci.edu/ml/index.php

Kaggle. Kaggle is a platform for predictive modelling and analytics competitions. There are many competition datasets available, such as web traffic time-series forecasting competition, household electric power consumption forecasting competition and Global Energy Forecasting Competition.

https://www.kaggle.com/