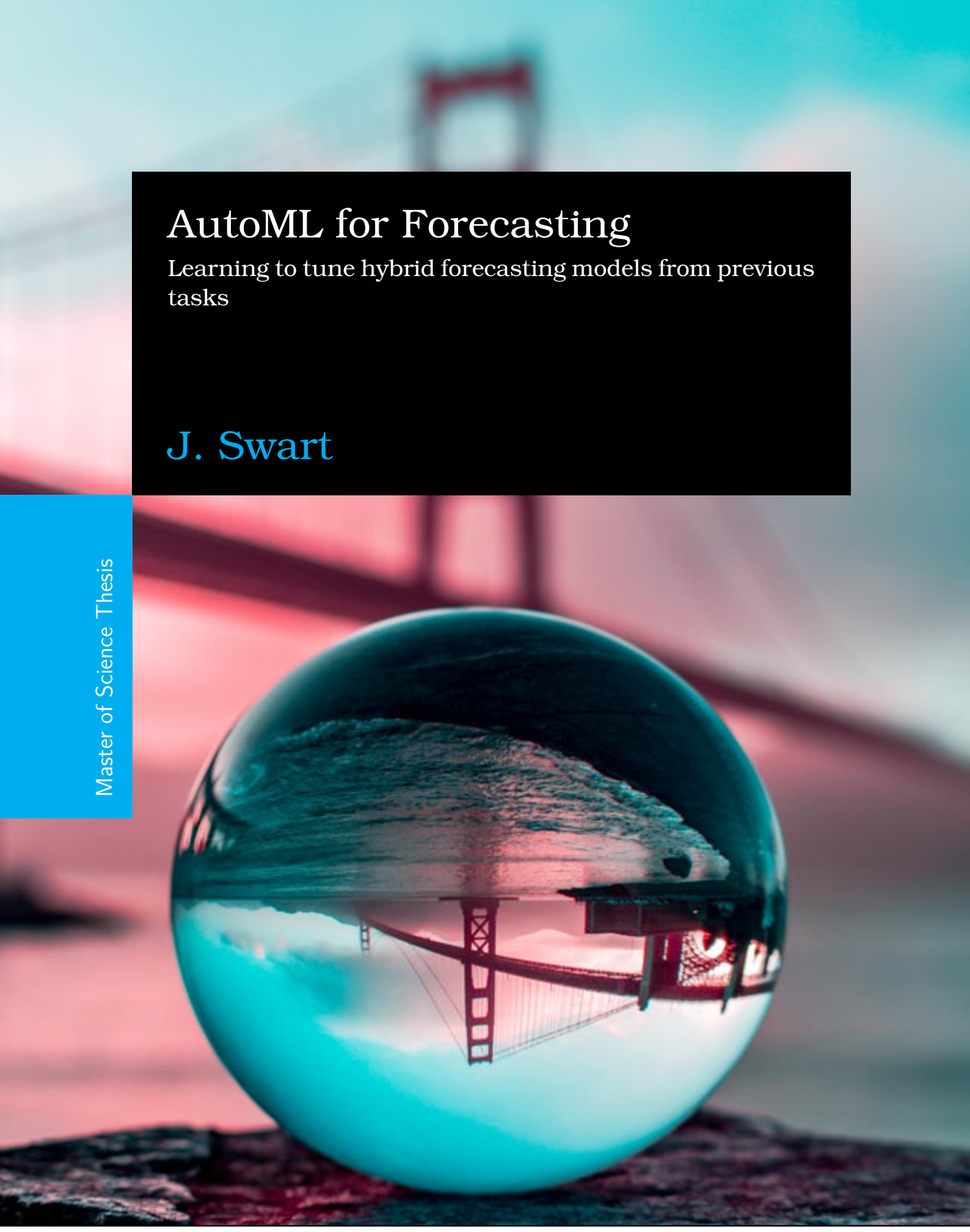
AutoML for Forecasting

Learning to tune hybrid forecasting models from previous tasks

J. Swart

Master of Science Thesis



TU Delft Delft University of Technology

Delft Center for Systems and Control

AutoML for Forecasting

Learning to tune hybrid forecasting models from previous tasks

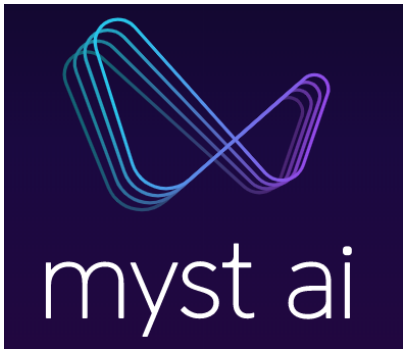
Master of Science Thesis

For the degree of Master of Science in Systems and Control at Delft University of Technology

J. Swart

August 18, 2020

Faculty of Mechanical, Maritime and Materials Engineering (3mE) · Delft University of Technology



The work in this thesis was supported by Myst AI. Their cooperation is hereby gratefully acknowledged.

Delft University of Technology
Department of
Delft Center for Systems and Control (dcsc)

The undersigned hereby certify that they have read and recommend to the Faculty of Mechanical, Maritime and Materials Engineering (3mE) for acceptance a thesis entitled

AutoML for Forecasting

by

J. Swart

in partial fulfillment of the requirements for the degree of

Master of Science Systems and Control

Dated: August 18, 2020

Supervisor(s):

Dr.ing. P. Mohajerin Esfahani

Reader(s):

Dr.ing. J. Kober

Ing. J. Lago

Abstract

A time series is a series of data points indexed in time order. It can represent real world processes, such as demand for groceries, electricity usage and stock prices. Machine Learning (ML) models that accurately forecast these processes enable improved decision-making for reducing waste and increasing efficiency. Previous research has produced an enormous number of ML model classes, each well-performing on a different forecasting task domain, and each written in their paradigm's mathematical language.

For a new forecasting task in business, the job of data scientists is to select, tune and evaluate some existing ML models. Because data scientists are scarce and expensive, many resources are spent on replacing this human job with an automated approach, referred to as AutoML. In current practice, the many existing ML models are used by tuning some of them and combining their separate forecasts. An alternative is using them by merging their intrinsic components, and tuning them all together to find a single hybrid ML model with better performance. This is possible if the ambiguous language between forecasting paradigms is consolidated into a unifying framework.

The first aim of this research is to introduce this framework, and thereby replacing the human job with a computational job. The complete list of instructions to create a hybrid ML model - data cleaning excluded - is presented in parameter format: a superparameter configuration. Its components are feature engineering, training set formation and hypothesis training. An example shows how superparameters from different paradigms can constitute a hybrid model. The computational job is presented as superoptimization: optimizing the superparameters for performance, applied to the task at hand. The problem of superoptimization is that it requires too much runtime on a computer.

The second aim of this research is to reduce the runtime of the computational job, by learning from previous tasks. This research proposes metafeatures for warmstarted Bayesian optimization. It suggests promising hypothesis training superparameters (complexity and overfitting), from previous tasks similar in size and input richness. The computational complexity reduction by 50% in the experiment, with respect to both a naïve and (proposed) coldstart benchmark method, provides evidence for the potential of the proposed method. The weight of evidence for the metafeatures is increased, by maintained performance improvement when

the method is badly tuned. The open-source Python package **warmstart** is published as a foundation for future experiments that focus on the other superparameter components, in the pursuit of an AutoML for hybrid forecasting models.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This thesis report is written as a part of the Master Program in Systems and Control at the Faculty of Mechanical, Maritime and Materials Engineering of Delft University of Technology. It forms the final step in completing this study and thereby also the end of my time as a student in Delft.

Looking back at my internship at Myst AI in San Francisco and putting together a thesis report in The Hague, Holland, I would like to thank a number of people for their help.

First of all, I would like to express my gratitude towards my supervisor, dr. ir. Peyman Mohajerin Esfahani, for his guidance during the project, and for his structured advice on composing a scientific report. I would also like to thank dr. ir. Jens Kober and ir. Jesus Lago, for completing my graduation committee. Furthermore, I would like to thank ir. Pieter Verhoeven, for taking me under his wing at his promising startup, and giving me the opportunity to view the world from a Silicon Valley perspective. I would like thank ir. Patrick Sheehan, for being an excellent, structured and result-oriented mentor, for dancing to National Players Anthem whenever we discovered a better forecasting model, and for being my best friend in San Francisco. Thanks to all the colleagues at Myst, who inspired me every day.

Many thanks to all the people that helped make my time as a student absolutely terrific. I would like thank my mother, who supports me in everything I do. I would like to address a special person, my girlfriend Claire, who has never failed to support me through tough times.

I don't know how to thank my Dad. He has inspired me every day in my life. His guidance has been infinite, an it will stay like that, now that his time has come. Thank you.

Delft, University of Technology J. Swart
August 18, 2020

Chapter 1

Introduction

1-1 Context

Who benefits most from Machine Learning (ML) forecasting algorithms? There are countless organizations that want to know the future value of a process for their decisions. How much should the intensive care of a hospital scale up their capacity, to meet the future number of COVID-19 patients and save lives? In which stocks should a trading agency invest, to profit from the prospective increase of stock prices? How much fruit should a grocery store buy, to match the future demand and prevent wastage? These questions form only the tip of the iceberg of the applications of the forecasting research domain, that focuses on predicting the future value of a process.

1-1-1 How a forecasting model is made

If the historical values of the process are stored in a time series, a ML forecasting model can learn from the past to predict the future value of a process. The shared setting for forecasting tasks is that there is access to the predicted time series (endogenous series), and optionally, time series that are related to the predicted signal (exogenous series). An example of this setting is visualized in Figure 1-1.

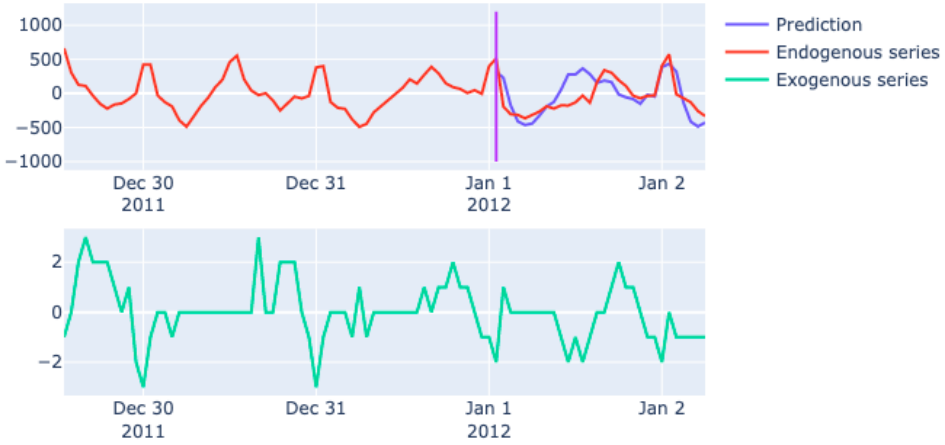


Figure 1-1: An example forecast setting with endogenous and exogenous data. The forecast is made at the first hour of January 1^{st}.

The research community has produced an enormous set of forecasting models. The performance of a forecasting model is measured from the forecast errors over a historical test period. As it depends on the forecasting task at hand which forecasting model yields good forecasting performance, the job is to find the right forecasting model. This job, currently performed by humans, is an iterative process of choosing a model, evaluating its performance and choosing a new one, until the human is satisfied with the best found performance. An example of this iterative job is visualized in Figure 1-2.

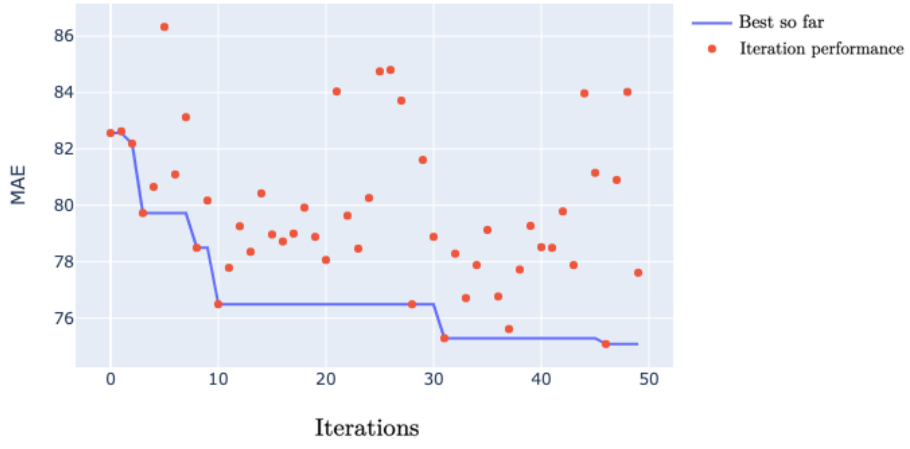


Figure 1-2: Performance visualization of iteratively suggesting model options for a forecasting task. The forecasting accuracy per iteration is measured in Mean Absolute Error (MAE).

1-1-2 Automated Machine Learning

In the current state of practice, the job of choosing and evaluating a forecasting model for new unseen forecasting tasks requires humans with education in programming, forecasting, ML and statistics, also referred to as forecasting data scientists. Because the salaries of these gatekeepers of the technology are high, obtaining a forecasting model that creates value in an application is expensive. The knowledge produced by the forecasting research community therefore tends to be used for economically profitable forecasting applications of big businesses. This tendency is enforced by the increasing complexity of the job of choosing a model, because the ML research community is expanding its model set at a rapid pace.

In the ML research domain of classification and regression tasks, a new emerging field is replacing the human with a computer, referred to as Automated Machine Learning (AutoML). It defines the creation and evaluation of an ML model as an objective function, to be optimized by an algorithm. Algorithms are proposed to reduce the computational complexity and improve best-found model performance of the job. How the model's performance is evaluated can be seen in Figure 1-2. Black-box optimization algorithms, for example Bayesian optimization [1] and genetic optimization [2] learn from previous evaluations of the objective function for the task at hand. The newest subdomain in AutoML, metalearning, focuses on optimization algorithms that learn from the completed job on previous tasks[3, 4, 5].

1-1-3 State of research in forecasting

The M competitions[6, 7, 8, 9] aim to test the performance of forecasting models in an objective and unbiased way. Despite their extensive ability to learn from history, new sophisticated ML forecasting models have not outperformed simple classical models [8]. The most recent M competition [9] however, was won by a forecasting model[10] that incorporates the intrinsic components of a classical and ML model. Founder of the M competitions, Makridakis, argues that "the logical way forward is the utilization of hybrid and combination methods"[9].

1-1-4 Ideological goal

As there are many forecasting subdomains, their respective models are written in different mathematical languages. What is missing, is a unifying parametrization of hybrid models, applicable to the AutoML framework. This research pursuits the ideological goal

to democratize machine learning for any forecasting task, and converge models across paradigms, by replacing expensive data scientists with a computer, that optimizes over cross-paradigm forecasting model components.

This goal is relevant, because hybrid models can improve the forecasting performance for many applications, and an automated approach make well-performing forecasting models accessible to less affluent organizations.

1-2 Objectives, scope and structure

1-2-1 Thesis objectives and scope

To address the ideological goal, this research indentifies three main objectives. Each objective seeks to solve a corresponding problem. The first problem is that combining forecasting model components of classical forecasting and supervised ML is difficult, because the paradigms speak a different mathematical language. The objective is to:

Objective (1): Merge the classical time series forecasting and supervised ML framework into a parametrization of creating and evaluating hybrids.

This parametrization, referred to as **superparameter evaluation** in this research, makes it easier for humans to understand how to define hybrid forecasting models. It also enables the AutoML framework to be applied to a hybrid forecasting model domain.

The second problem is that the complexity and size of a cross-paradigm forecasting model component set, represented by the parametrization, is too big for a human to consider. The objective is to:

Objective (2): Replace the human job of finding a well-performing modeling option with a computational job, by applying the AutoML framework to the parametrization.

As opposed to the limited human capacity, computational capacity is scalable and research is constantly improving computational efficiency. The impact of replacing a human with a computer is that the job becomes more feasible. It also makes the job less expensive, as salaries for data scientists are high. The objective is limited to black-box optimization algorithms and warmstarting. The computational job, which is the AutoML framework applied to the parametrization (superparameter evaluation), is referred to as **superoptimization** in this research. It is limited to black-box optimization and warmstarting.

The third problem is that existing optimization algorithms that optimize over the parametrization of cross-paradigm forecasting model components require too much runtime. The objective is to:

Objective (3): Reduce the runtime of the computational job, by learning from previous tasks.

The impact of achieving this objective is that performing the job becomes more feasible. This research focuses on warmstarting[11], a metalearning method developed in the context of classification tasks. The method imitates the approach of a human: when a new task is at hand, the method finds the most similar previous tasks, uses their best modeling options as starting point, and continues the job from there. Similarity is measured according to metafeatures, a mathematical characterizations of a task, for example the size of the task's dataset. The approach only works if the defined metafeatures correlate with forecasting models which yield good performance. This research generates new knowledge by synthesizing two metafeatures, and providing empirical evidence that they increase the performance of the warmstarting method.

1-2-2 Thesis structure

Chapter 2 presents superparameter evaluation, the parametrization that merges classical time series forecasting and supervised ML models. In chapter 3, the job of a human forecaster is replaced with a computational job, referred to as superoptimization, by applying the AutoML framework to the parametrization. The proposed method to reduce the runtime of the computational job by learning from previous tasks, is explained in chapter 4. Chapter 5 discusses the design choices for the experiment. The results of the experiment are presented in chapter 6. Finally, chapter 7 and 8 present the discussion, the conclusions, and suggestions for future research.

Chapter 2

Superparameter evaluation

This research presents a framework for parametrizing hybrid models from the classical forecasting and supervised Machine Learning (ML) paradigm, for the objective to:

Objective (1): Merge the classical time series forecasting and supervised ML framework into a parametrization of creating and evaluating hybrids.

The parametrization is referred to as superparameter evaluation $I(p, t)$. The list of instructions p, produces a forecasting model f, for the forecasting task at hand t. The forecasting model f is then evaluated, to return the performance index l_{te}. The inputs and outputs of this process are illustrated in Figure 2-1. In the human job of finding a p with a small l_{te} (Figure 3-1). It is used as the objective function, to be minimized.

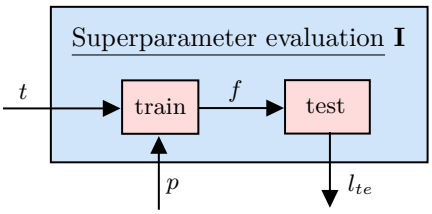


Figure 2-1: The process of superparameter evaluation. Calculating the performance index l_{te} of the suggested superparameter configuration p for task t, consists of training a forecasting model f and evaluating it.

Firstly, section 2-1 explains the function of a forecasting model f, with a simple example. Then, section 2-2 formalizes the forecasting task t. Section 2-3 explains what a superparameter configuration p is, the superparameter space it is drawn from, and how it produces f. Section 2-4 explains how the performance index of f for t is measured, and presents a summary of the entire process. Finally, section 2-5 discusses the improvements of the proposed standardization of forecasting model production.

2-1 Forecasting model

A forecasting model uses time series data. A time series is a set of repeated observations through time of the same variable[12]. An example is the daily number of apples sold by your local grocery store. We write a time series as

$$\mathbf{x} = \{x_1, x_2, ...x_T\} \quad \text{or} \quad \mathbf{x} = \{x_t\}, \quad t = 1, 2, ...T, \tag{2-1}$$

where x_t is the variable at time t and the current time is T. Business decisions can be improved by knowing what the future values of certain time series are, e.g. shortage of apples in a grocery store can be prevented by buying the right amount on forehand. A dataset m can consist of multiple time series. We define the forecasted time series as the endogenous series $\mathbf{y}$, for example the daily apple sales of your local grocery store. We define other time series in m, related to $\mathbf{y}$ as the exogenous series $\mathbf{v}$, for example the precipitation forecast. A forecasting model f

$$f : \mathbb{X} \rightarrow \mathbb{Y}, \tag{2-2}$$

predicts a future value of the endogenous series $\mathbf{y} \in \mathbb{Y}$, by analyzing a selection of the currently available observations in $\mathbf{y}$ and $\mathbf{v}$, denoted as $\mathbf{x}_T \in \mathbb{X}$. We define the number of time steps to predict into the future as forecast horizon c. A prediction $\hat{y}_T$ at prediction base time T for forecast horizon c, is denoted by

$$\hat{y}_T = f(\mathbf{x}_T) \approx y_{T+c}, \tag{2-3}$$

the actual value of the predicted observation at time step T is denoted by

$$\bar{y}_T = y_{T+c}, \tag{2-4}$$

and the forecast error,

$$\epsilon_T = \hat{y}_T - \bar{y}_T, \tag{2-5}$$

is the difference between the prediction and the actual value. An example forecasting task t is forecasting the next day's apple sales $\mathbf{y}$, illustrated in Figure 2-2. For prediction, we use historical values of $\mathbf{y}$ and precipitation forecasts $\mathbf{v}$, because people may postpone their grocery store trip to the next day when it's raining. Every day a prediction for the next day is made, with a one-step shifted prediction base time T and input configuration x_T. The example forecasting model f is

$$f(x_T) = \hat{y}_T = 0.2 \cdot y_{T-1} + 0.9 \cdot y_T - 0.4 \cdot v_T - 0.1 \cdot v_{T+1} + 0.1. \tag{2-6}$$

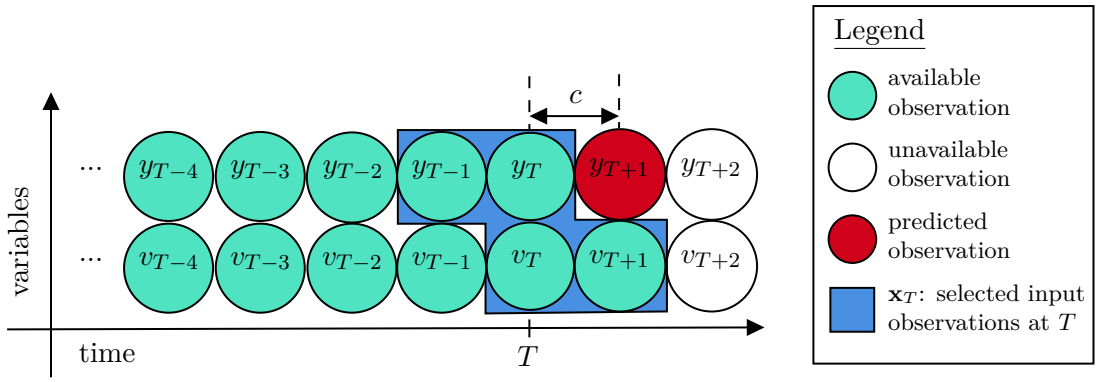


Figure 2-2: The input/output setting at prediction time T of an example one-step-ahead ($c = 1$) forecasting model. The input data $\mathbf{x}_T$ in blue is a selection of the available data points in green, which consists of the endogenous series **y** and one exogenous series **v**. The unavailable observation of interest y_{T+1}, illustrated in red, is predicted by the forecasting model's output $\hat{y}_T$.

2-2 Forecasting task

A forecasting task t is described by its dataset and forecasting scenario. The dataset m, which is provided by the client, is the stored history of the endogenous (and optionally exogenous) series, respectively **y** and **v**. The forecasting scenario, to be defined by the client, has the following components:

- The **data availability**, a declaration of the data points available at time T, as illustrated in an example in Figure 2-4. This is a constraint for input selection.
- The **forecast horizon** $c \in \mathbb{Z}$, which is the number of time steps, relative to T, to predict into the future.
- A **test set size** $n_{te} \in \mathbb{Z}$, the number of prediction base times T to use for reliable testing[1] of f. The data corresponding to these steps is taken apart as test set, before training.
- The **test loss function** $l_{te} \in \mathcal{L}_{te} : \mathbb{Y}^{n_{te}} \times \mathbb{Y}^{n_{te}} \to \mathbb{R}^+$, a way of calculating the performance index, given the actuals and predictions for the test set.

[1] testing is explained in 2-4

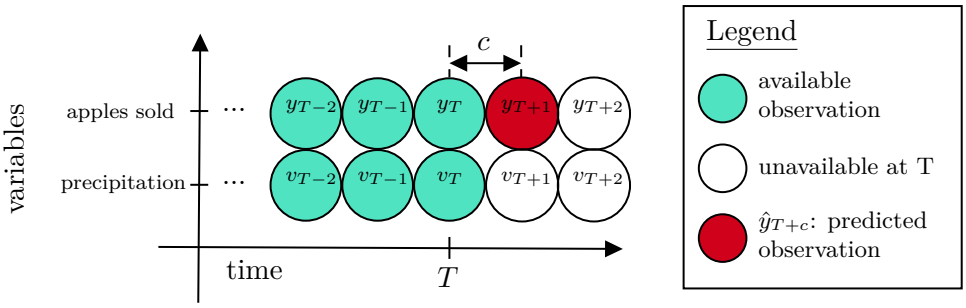


Figure 2-3: Simple example forecasting task: apple sales. This is the basic intuitive example that further sections will refer to. A grocery store wants to know how many apples will be sold tomorrow (forecast horizon $c = 1$), to determine how many apples to buy from its supplier. At every prediction time T, the available inputs for the forecast model are the apple sales numbers per day **y** and the millimeters of precipitation up until the day of forecasting **v**. The grocery store is convinced of the accuracy of the forecasting model, if its test loss function (mean absolute error), yields a low score over a test period of one year ($n_{te} = 365$). The grocery provides a dataset m, consisting of four years of daily measurements of the apple sales numbers and precipitation measurements.

The data availability and forecast horizon of a more complex forecasting task example is given in Figure 2-4.

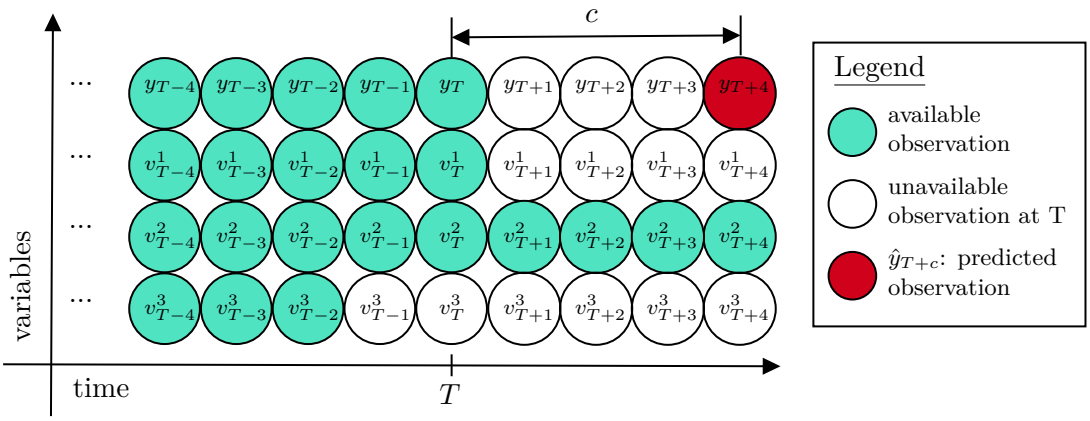


Figure 2-4: Complex example forecasting task. In this example the forecast horizon is four time steps, there is an exogenous series v^1 of which the observations until prediction base time T are available, another exogenous series v^2 of which observations are available four time steps into the future (for example a forecast of millimeters precipitation), and a third exogenous series v^3 of which the observations become available with a lag of two (for example due to database latency).

A task t comes from a structured space, the task space T, as summarized in Figure 2-5. This comprehensive mathematical characterization enables measuring the similarity between two forecasting tasks.

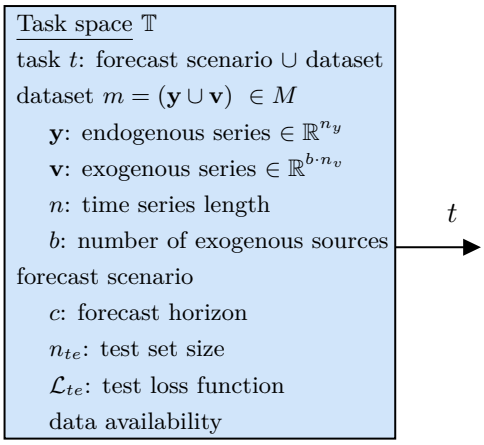


Figure 2-5: The elements of the task space. This is the first building block for the proposed framework in Figure 3-5.

2-3 Superparameters

A superparameter configuration p is the complete hierarchic list of (human) instructions, denoted as parameters of the process $\mathcal{I}_{f(\mathbf{x}_T)}(p)$ of producing a forecasting model $f(\mathbf{x}_T)$,

$$\{\mathcal{I}_{f(\mathbf{x}_T)}(p) : \mathbb{T} \rightarrow f(\mathbf{x}_T), \quad \forall p\}, \tag{2-7}$$

and p,

$$p = \begin{cases} p_{\mathbf{x}_T} \\ p_{tr} \\ p_{\mathcal{H}} \end{cases}, \tag{2-8}$$

is divided in instructions for feature engineering $p_{\mathbf{x}_T}$, composing a training set p_{tr} and hypothesis training $p_{\mathcal{H}}$. Formalizing the human instructions as a parameter type in stead of a functional, enables leveraging hierarchy during optimization over p's domain. Figure 2-6 illustrates how p and t are used to create f.

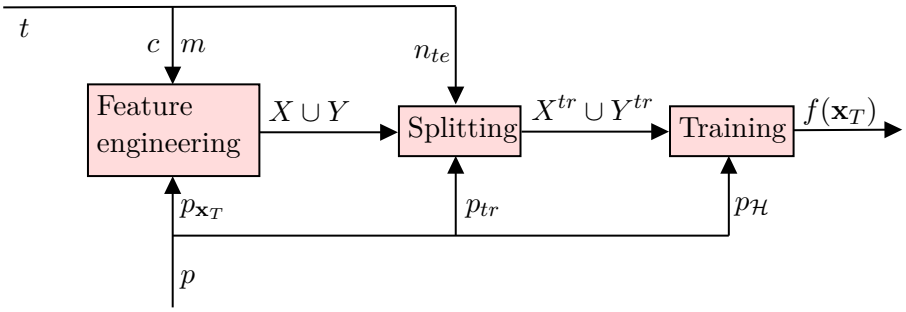


Figure 2-6: The process $\mathcal{I}_{f(\mathbf{x}_T)}$ of using the instructions in p to make forecasting model $f(\mathbf{x}_T)$. The instructions in feature engineering $p_{\mathbf{x}_T}$ are used to define the model input $\mathbf{x}_T$, relative to T. The instructions for forming a training set are denoted as p_{tr}. The instructions for the hypothesis class choice and the corresponding instructions for retrieving its optimized parameters are denoted as $p_{\mathcal{H}}$, and the process results in a forecasting model $f(\mathbf{x}_T)$.

Firstly $p_{\mathbf{x}_T}$, p_{tr} and $p_{\mathcal{H}}$ and their part of $\mathcal{I}_{f(\mathbf{x}_T)}(p)$ are explained. Then the space from which p is drawn, the superparameter space $\mathbb{P}$, is further explained. Finally, the improvement of this formulation is argumented.

2-3-1 Feature engineering

Feature engineering is the process $\mathcal{I}_{\mathbf{x}_T}$,

$$\mathcal{I}_{\mathbf{x}_T}(p_{\mathbf{x}_T}) : \mathbb{M} \times \mathbb{Z} \to \mathbb{X}^{n_s} \cup \mathbb{Y}^{n_s}, \quad \forall p_{\mathbf{x}_T}, \tag{2-9}$$

which retrieves input and output samples $X \cup Y$ from a given dataset m and forecast horizon c, by following the feature engineering instructions in $p_{\mathbf{x}_T}$. Every sample corresponds with a prediction base time T, and n_s is the number of samples. The process consists of series transformation, input selection and sample collection. Series transformation is the transformation of the available observations in the endogenous and exogenous series to create additional predictive exogenous series (hidden states). From all available observations and series transformations at T, a selection is made for input vector $\mathbf{x}_T$. The final feature engineering step is to create the sample set $X \cup Y$, a collection of $\mathbf{x}_T \cup \bar{y}_T$ for every T in m. The process is illustrated in Figure 2-7.

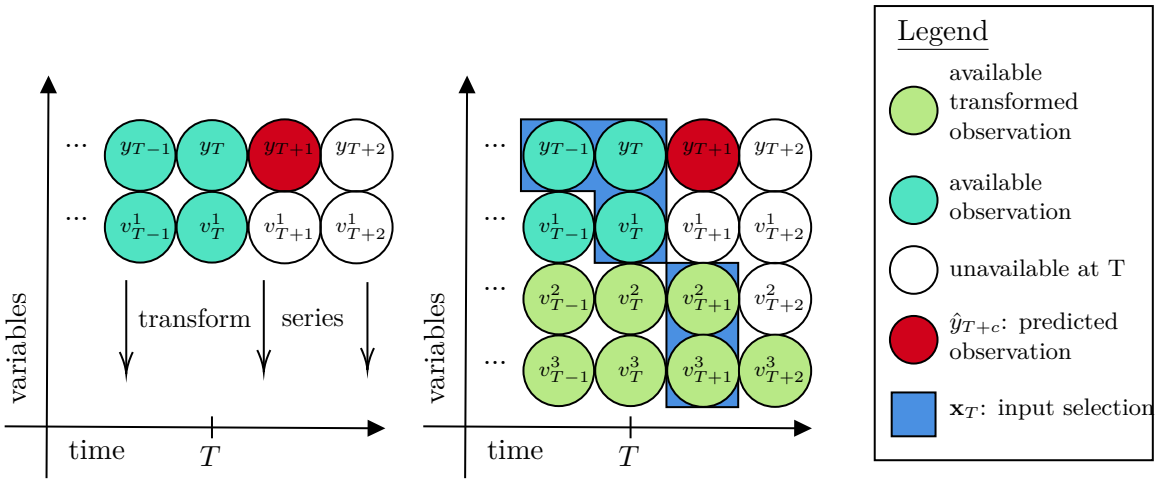


Figure 2-7: The feature engineering process $\mathcal{I}_{\mathbf{x}_T}$, for a single T, creating an example input vector $\mathbf{x}_T$. The process starts in the left figure, with the endogenous and exogenous series, for example the series in the simple example in figure 2-3. The series can be transformed to obtain additional predictive inputs. For example, v^2 can be the one-day-ahead precipitation forecast, and v^3 can be the day number of the week. Finally, a subset of the available observations is selected as input vector $\mathbf{x}_T$. Promising example transformations stem from the classical time series forecasting paradigm, that adds hidden states representing the time series' level, seasonality or forecasting error.

Many model components from the classical forecasting paradigm can be incorporated as transformations. Some examples are the forecast error of the previous time step, the seasonality ('MA' and 'S' in SARIMA[13]), and the modes of a CEEMDAN decomposition[14][15]. Examples of domain specific transformations are the 20-day moving average of a stock price for a stock price forecasting task [16], and the one-day-ahead precipitation forecasts for the apple sales task. An example of $p_{\mathbf{x}_T}$, corresponding to Figure 2-7, is

$$p_{\mathbf{x}_T} : \begin{cases} \mathbf{y} : & \{-1, 0\} \\ \mathbf{v}^1 : & \{0\} \\ \text{predict}(\mathbf{v}^1) : & \{1\} \\ \text{day of week}(\mathbf{y}) : & \{1\} \end{cases},$$

where the first column represents the (transformed) time series, and the second column represents the selected subset of input lags. This example results in the following samples,

$$\mathbf{x}_T = \begin{bmatrix} y_{T-1} \\ y_T \\ v_0^1 \\ \text{predict}(v_{T+1}^1) \\ \text{day\_of\_week}(y_{T+1}) \end{bmatrix}, \quad \bar{y}_T = y_{T+c},$$

and the sample set $X \cup Y$, consisting of a collection of the samples for every T, is illustrated in Figure 2-8.

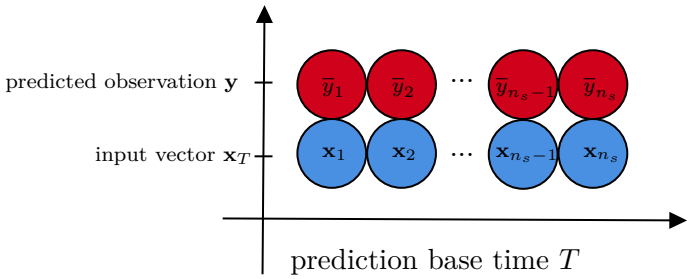


Figure 2-8: The output of feature engineering process: the sample series $X \cup Y$. The sample series contains the input vector $\mathbf{x}_T$ and the actual value of the predicted observation $\bar{y}_T$ (as label) for every prediction base time T in dataset m. For the apple sales task of Figure 2-3, an example $\mathbf{x}_T$ is marked in blue in Figure 2-7.

2-3-2 Splitting

Splitting is the process $\mathcal{I}_{tr}$,

$$\mathcal{I}_{tr}(p_{tr}) : \mathbb{X}^{n_s} \cup \mathbb{Y}^{n_s} \times \mathbb{Z} \rightarrow (\mathbb{X}^{n_{tr}} \cup \mathbb{Y}^{n_{tr}}), (\mathbb{X}^{n_{te}} \cup \mathbb{Y}^{n_{te}}), \quad \forall p_{tr} \tag{2-10}$$

which uses the sample set $X \cup Y$ to retrieve a training set $X_{tr} \cup Y_{tr}$ and test set $X_{te} \cup Y_{te}$, by using the test set size n_{te} (defined in task t, see section 2-5) and the training set instructions p_{tr}. Firstly, the samples corresponding to the n_{te} last prediction base times are removed from $X \cup Y$ to form the test set. The remaining samples are used to form the training set, following the instructions of p_{tr}. An example for p_{tr} is

$$p_{tr} : \quad \text{all}, \tag{2-11}$$

of which the splitting process is illustrated in Figure 2-9.

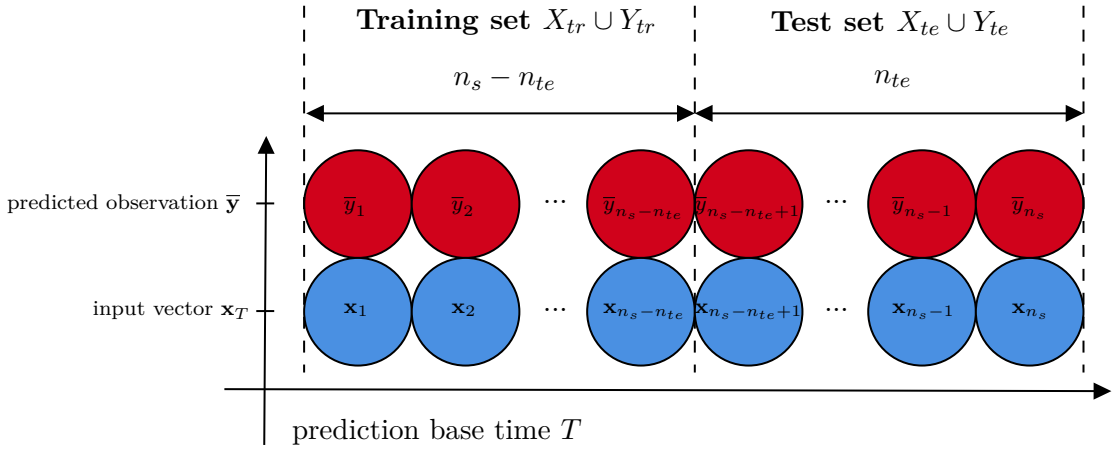


Figure 2-9: Splitting process example $\mathcal{I}_{tr}$, creating a test set and training set from the sample series. Firstly the test set $X_{te} \cup Y_{te}$ of size n_{te} is removed from the sample series $X \cup Y$. Given the remaining samples, p_{tr} instructs how to form the training set. For the apple sales example, $n_{te} = 365$ samples form the test set. When p_{tr} of equation 2-11 is applied (using the example $p_{\mathbf{x}_T}$ in Figure 2-7), the training set consists of 728 samples.

Note that the abstract formulation in equation 2-10 allows for more complex training set instructions. Examples are selecting a subset of the remaining samples and extending the samples with samples from similar datasets[17] or samples with added noise[18].

2-3-3 Hypothesis training

We define hypothesis training as the process $\mathcal{I}_{\mathcal{H}}$,

$$\mathcal{I}_{\mathcal{H}}(p_{\mathcal{H}}) : \mathbb{X}^{n_{tr}} \cup \mathbb{Y}^{n_{tr}} \rightarrow f(\mathbf{x}_T), \quad \forall p_{\mathcal{H}}, \tag{2-12}$$

which retrieves a forecasting model $f(\mathbf{x}_T)$ from training set $X_{tr} \cup Y_{tr}$, by using the hypothesis instructions $p_{\mathcal{H}}$. The process consists of hypothesis formation and training. A subset of the superparameters in $p_{\mathcal{H}}$ represents the type of relation to be learned between $\mathbf{x}_T$ and $\bar{y}_T$. This is the hypothesis class $\mathcal{H}$,

$$\mathcal{H} : \{h(\mathbf{w}) : \mathbb{X} \rightarrow \mathbb{Y}, \text{ for some } \mathbf{w}\}, \tag{2-13}$$

where $\mathbf{w}$ is the parameter vector to be optimized during training. The better the hypothesis class corresponds with the type of relation between the input and predicted variable, the higher the forecasting model's performance can get. An example $\mathcal{H}$ is linear regression[19], which for equation 2-6 corresponds to

$$h(\mathbf{w}) = w_1 \cdot y_{T-1} + w_2 \cdot y_T + w_3 \cdot v_T + w_4 \cdot v_{T+1} + w_5, \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix}, \mathbf{x}_T = \begin{bmatrix} y_{T-1} \\ y_T \\ v_T \\ v_{T+1} \\ w_5 \end{bmatrix}$$

The remaining subset of superparameters in $p_{\mathcal{H}}$ are the training instructions. The objective of training,

$$\min_{h \in \mathcal{H}} \sum_{i=1}^{n_{tr}} \mathcal{L}_{tr}(h(\mathbf{x}_i^{tr}), y_i^{tr}), \tag{2-14}$$

is to optimize parameter vector $\mathbf{w}$, to obtain $h(w) \in \mathcal{H}$, which predicts the training set's samples most accurately. The training loss function $\mathcal{L}_{tr}$,

$$\mathcal{L}_{tr} : \mathbb{Y}^{n_{tr}} \times \mathbb{Y}^{n_{tr}} \rightarrow \mathbb{R}^+, \tag{2-15}$$

is a training instruction in $p_{\mathcal{H}}$ defining how to calculate the performance from a set of predictions $\hat{Y}$ and actuals Y during training. The training optimization algorithm $\mathcal{A}_{tr}$ is a training instruction in $p_{\mathcal{H}}$, defining the sequential process for optimizing equation 2-14, for example Stochastic Gradient Descent (SGD)[20] algorithms. An example $\mathcal{L}_{tr}$ is the Mean Squared Error (MSE)[21],

$$\mathcal{L}_{tr}(\hat{Y}, Y) = \frac{1}{n^{tr}} \sum_{i=1}^{n^{tr}} (\hat{y}_i - y_i)^2, \tag{2-16}$$

a loss function that penalizes large errors heavily. The training process results in a forecasting model

$$f(\mathbf{x}_T) = h(\mathbf{w}^*). \tag{2-17}$$

An example $p_{\mathcal{H}}$ is

$$p_{\mathcal{H}}: \text{ linear}: \begin{cases} \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{SGD} \end{cases}, \tag{2-18}$$

which is the conjunction of the simple examples in this section. More complex examples can be drawn from the neural networks (NN)[22] domain. An example $p_{\mathcal{H}}$ for a NN is

$$p_{\mathcal{H}}: \text{NN}: \begin{cases} \mathcal{H}: & \left\{ \begin{array}{l} \text{width}: 4 \\ \text{activation}: \text{ReLU} \end{array} \right\}, \left\{ \begin{array}{l} \text{width}: 6 \\ \text{activation}: \text{ReLU} \end{array} \right\} \\ \mathcal{L}_{te}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{RMSprop}: \begin{cases} \text{learning rate}: & 0.001 \\ \text{rho}: & 0.9 \\ \text{epsilon}: & \text{None} \\ \text{decay}: & 0.0 \\ \text{batch size}: & 50 \\ \text{epochs}: & 100 \\ \text{kernel initializer}: & \{\text{uniform}, \text{uniform}\} \end{cases} \end{cases}, \tag{2-19}$$

which illustrates a $p_{\mathcal{H}}$ in a hypothesis training domain[2] where lower level parameters have to be set to satisfy equation 2-12. In current literature, these hypothesis training superparameters are referred to as hyperparameters.

Models from the classical forecasting paradigm use the simple method of linear regression for this hypothesis training component. The ML community has recently produced many sophisticated models that can replace linear regression, using this framework.

[2] the example domain includes mini-batch gradient descent (RMSprop)[23] as training optimization algorithm $\mathcal{A}_{tr}$

2-3-4 Superparameter space

The superparameter space $\mathbb{P}$ is the set from which p is drawn. This research attempts to structure this space, to enable optimizing over it. An example for p,

$$p = \begin{cases} p_{\mathbf{x}_T}: & \begin{cases} \mathbf{y}: & \{-1, 0\} \\ \mathbf{v}^1: & \{0\} \\ \text{predict}(\mathbf{v}^1): & \{1\} \\ \text{day of week}(\mathbf{y}): & \{1\} \end{cases} \\ p_{tr}: & \text{all} \\ p_{\mathcal{H}}: & \text{linear}: \begin{cases} \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{GD} \end{cases} \end{cases},$$

is the merge of the partial examples in previous sections. It is a member of a search space $\mathbb{S}$, the set of considered p's. For a human this is the set of p's in its knowledge, for a computer this is the set of p's in the implementation. An example $\mathbb{S}$, which includes a feature engineering search space in the SARIMAX model domain, and a hypothesis training search space in the Random Forest[3][24] domain, is

$$\mathbb{S} = \begin{cases} p_{\mathbf{x}_T}: & \begin{cases} \mathbf{y}: & \{\text{lags} \subset \{\text{lag} \in \mathbb{Z}, \quad -2 \leq \text{lag} \leq 0\}\} \\ \mathbf{v}^1: & \{\text{lags} \subset \{\text{lag} \in \mathbb{Z}, \quad -2 \leq \text{lag} \leq 0\}\} \\ \text{predict}(\mathbf{v}^1): & \{\text{lags} \subset \{\text{lag} \in \mathbb{Z}, \quad -1 \leq \text{lag} \leq 1\}\} \\ \text{day of week}(\mathbf{y}): & \{\text{lags} \subset \{\text{lag} \in \mathbb{Z}, \quad -1 \leq \text{lag} \leq 1\}\} \end{cases} \\ p_{tr}: & \{n_{tr} = 1000 \cdot d_{n_{tr}}, \quad d_{n_{tr}} \in \mathbb{Z}, \quad 10 \leq d_{n_{tr}} \leq 15\} \\ p_{\mathcal{H}}: & \begin{cases} \bigcirc \text{ linear}: & \begin{cases} \mathcal{L}_{tr}: & \{\text{MSE}, \text{MAE}\} \\ \mathcal{A}_{tr}: & \text{GD} \end{cases} \\ \bigcirc \text{ DTA}: & \begin{cases} \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{RF}: \begin{cases} \{m_{depth} \in \mathbb{Z}, \quad 5 \leq m_{depth} \leq 10\} \\ \{n_{est} = 10^{d_{n_{est}}}, \quad d_{n_{est}} \in \mathbb{Z}, \quad 1 \leq d_{n_{est}} \leq 3\} \end{cases} \end{cases} \end{cases} \end{cases}, \tag{2-19}$$

in which all available choices are represented as sets of variables in a multidimensional space. This space consists of integer and categorical variables. The $\bigcirc$-symbol represents the conditional nature of a choice, making the lower level variables redundant when the variable is

[3] DTA stands for Decision Tree Average, the hypothesis class applicable to the Random Forest optimization algorithm

not selected. Note that in the feature engineering part, not one but a subset of lags can be selected. The search space $\mathbb{S}$ is a subset of the superparameter space $\mathbb{P}$,

$$\mathbb{P} = \begin{cases} \mathbb{P}_{\mathbf{x}_T}, & \mathcal{I}_{\mathbf{x}_T}(p_{\mathbf{x}_T}) : \mathbb{M} \times \mathbb{Z} \rightarrow \mathbb{X}^{n_s} \cup \mathbb{Y}^{n_s}, & \forall p_{\mathbf{x}_T} \subset \mathbb{P}_{\mathbf{x}_T} \\ \mathbb{P}_{tr}, & \mathcal{I}_{tr}(p_{tr}) : \mathbb{X}^{n_s} \cup \mathbb{Y}^{n_s} \times \mathbb{Z} \rightarrow (\mathbb{X}^{n_{tr}} \cup \mathbb{Y}^{n_{tr}}), (\mathbb{X}^{n_{te}} \cup \mathbb{Y}^{n_{te}}), & \forall p_{tr} \in \mathbb{P}_{tr} \\ \mathbb{P}_{\mathcal{H}}, & \mathcal{I}_{\mathcal{H}}(p_{\mathcal{H}}) : (\mathbb{X}^{n_{tr}} \cup \mathbb{Y}^{n_{tr}}) \rightarrow h(\mathbf{w}^*), & \forall p_{\mathcal{H}} \in \mathbb{P}_{\mathcal{H}} \end{cases} \tag{2-20}$$

the space of all p's that can possibly be suggested. In words, it is limited to fit parameters of functionals that

- transform a time series (in $\mathcal{I}_{\mathbf{x}_T}$),
- select a time series element, given a reference index (in $\mathcal{I}_{\mathbf{x}_T}$),
- select and/or create labeled samples ($\mathcal{I}_{tr}$),
- are a supervised ML model ($\mathcal{I}$).

It is an infinitely large space, it contains different types of dimensions and has infinitely many conditional subspaces. Understanding the hierarchy of $\mathbb{P}$ is important for research in optimization algorithms[4] in its space. The definition of $\mathbb{P}$ divides the job of a human into implementation of parametrized functionals, and tuning their parameters for a specific forecasting task.

Applied in practice, the search space $\mathbb{S}$ represents the superparameter configuration set applicable to the implemented functionals, to be optimized by a human or optimization algorithm. In a business, specialized in forecasting, the implementation design of superparameter evaluation $\mathcal{I}$ and corresponding search space $\mathbb{S}$ should contain all well-performing superparameter configurations p for the task set within the real world application domain of its focus. As new well-performing forecasting models are discovered by research, the business absorbs the new knowledge by implementing their intrinsic components (the explained functionals) in $\mathcal{I}$.

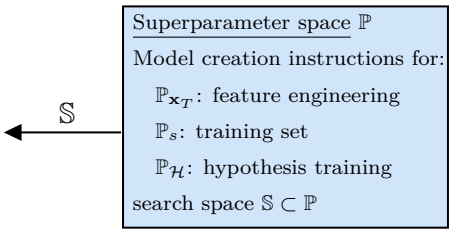


Figure 2-10: The elements of the superparameter space.

[4]optimization algorithms applied to superparameter evaluation are referred to as superoptimization algorithms in this research, as explained in chapter 3

2-4 Performance index

The performance index l_{te} is a scalar accuracy score of the produced forecasting model $f(x) = h(w^*)$, which is retrieved using the task's test set $X_{te} \cup Y_{te}$ and test loss function $\mathcal{L}_{te}$. Firstly the model makes predictions $\hat{Y}_{te} \in \mathbb{Y}^{n_{te}}$ for the sample inputs in the test set X_{te}. The test loss function $\mathcal{L}_{te} : \mathbb{Y}^{n_{te}} \times \mathbb{Y}^{n_{te}} \to \mathbb{R}^+$ then compares $\hat{Y}_{te}$ with the actuals Y_{te} and maps them to a positive scalar, which is the final output of the superparameter evaluation process. An example $\mathcal{L}_{te}$ is the Mean Absolute Percentage Error (MAPE)[25],

$$\mathcal{L}_{te}(\hat{Y}_{te}, Y_{te}) = \frac{1}{n_{te}} \sum_{i=1}^{n_{te}} \left| \frac{Y_{te,i} - \hat{Y}_{te,i}}{Y_{te,i}} \right|, \tag{2-21}$$

where every step i is a shift in the prediction base time. There are countless metrics proposed in literature[26, 27, 28, 21], all of which apply to different risk scenario's of a task, making it a characteristic of the task.

The superparameter evaluation process I can be concisely presented as

$$\mathcal{I} : \mathbb{T} \times \mathbb{S} \to \mathbb{R}^+, \tag{2-22}$$

and the complete process is illustrated in detail in Figure 2-11.

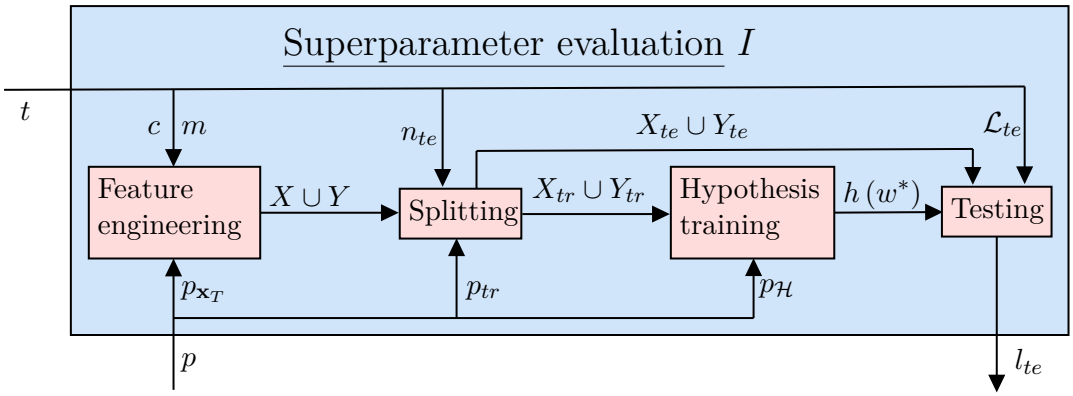


Figure 2-11: The superparameter evaluation process in detail. Given a superparameter configuration $p \in \mathbb{S}$ and forecasting task t, this evaluation process produces a forecasting model $f = h(w^*)$ and returns its performance index l_{te}. The process consists of four steps. Step (1), section 2-3-1: feature engineering is performed to obtain sample set $X \cup Y$ (). Step (2), section 2-3-2: splitting divides $X \cup Y$ up in test set $X_{te} \cup Y_{te}$ and training set $X_{tr} \cup Y_{tr}$. Step (3), section 2-3-3: hypothesis training uses the training set to optimize the parameters of a hypothesis class, to obtain forecasting model $f = h(w^*)$. Step (4): testing calculates the performance index l_{te} from the forecasting errors on the test set. This performance index is the output of the process.

The simple building block of superparameter evaluation is illustrated in Figure 2-12.

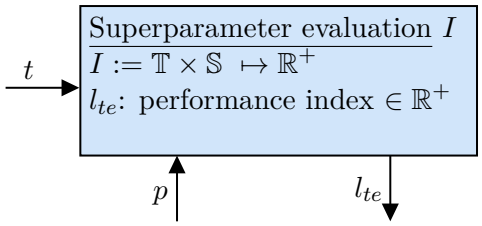


Figure 2-12: Superparameter evaluation building block.

2-5 Comparison with current practice

The first objective of this research,

Objective (1): Merge the classical time series forecasting and supervised ML framework into a parametrization of creating and evaluating hybrids,

is achieved by presenting a framework for parametrizing hybrid models from the classical forecasting and supervised ML paradigm. This section discusses its difference with the state of practice.

In current research, the job of model creation is explained using the term 'hyperparameters'[29]. They are defined as the parameters of a learning algorithm, which uses a dataset to create a model, and they have to be tuned to minimize a loss function. The format of a hyperparameter configuration is defined as an unstructured tuple, and no effort is made to further define the contents of a hyperparameter configuration. No notion is made on whether the term 'hyperparameter' includes or excludes feature engineering processes. This research rejects the vague term 'hyperparameter' that often causes ambiguity, and introduces the terms superparameter p, and forecasting task t. The proposed surrounding theory is a compact writing and implementation standardization for creating cross-paradigm models, applied to forecasting[5].

2-5-1 Completeness for full automation

The first difference of the term superparameter with respect to the term hyperparameter, is that it is explicitly defined as the complete set of instructions necessary for model creation. Likewise, the forecasting task t is defined in a complete sense. Because of this completeness, full automation of the human forecaster's job is enabled.

2-5-2 Cross-paradigm forecasting models

The second difference between the terms superparameter and hyperparameter, is that the dimensions in the superparameter space are hierarchized into feature engineering, training set formation an hypothesis training. This hierarchy enables creating search spaces that include cross-paradigm superparameter configurations, including the classical time series forecasting

[5]using some simplifications in the definitions (excluding the time component of the framework), the theory can be applied to the supervised ML framework

and ML regression paradigm. The hierarchy also reduces the number of dimensions of $\mathbb{S}$, and therefore the computational complexity of finding $p^*(t)$.

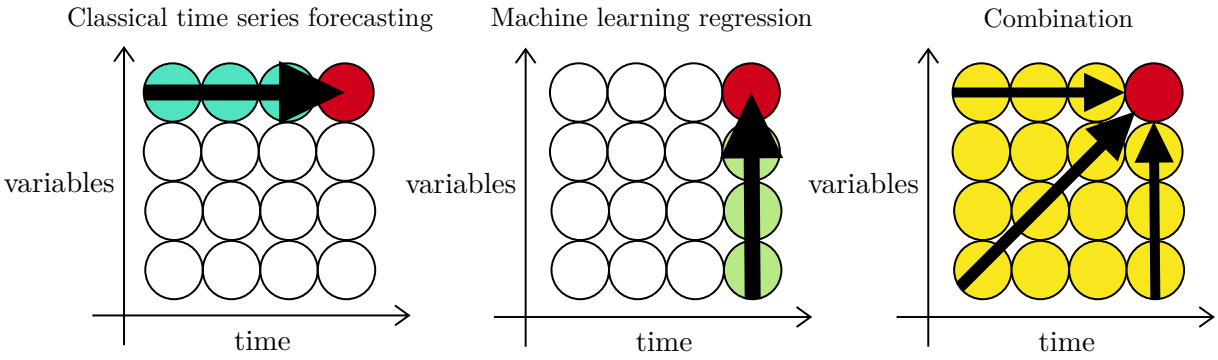


Figure 2-13: Principal forecast direction of classical time series forecasting (left), machine learning regression (center) and a combination of the two (right)

Classical time series forecasting mainly focuses on learning autoregressive patterns of univariate time series[30, 26, 31, 12, 32, 33, 34, 35]. The principal learning direction is illustrated in the left of Figure 2-13. In sophisticated classical models exogenous input time series are sometimes also leveraged[13], but they are always extensions on univariate forecasting models. The principal learning direction of ML regression is illustrated in the middle of Figure 2-13. It focuses on learning the relation between a target value and other variables, without a focus on autoregressiveness. This approach has been highly successful for learning time invariant tasks. This core difference in the approach to creating a model, lends itself for the classical forecasting community to take over the term hyperparameter as 'a parameter that needs to be set before training'. An example result of this way of thinking is the following notation of an example cross-paradigm $\mathbb{S}$,

$$\mathbb{S} = \begin{cases} \bigcirc \text{ ARIMA}: & \begin{cases} \text{order}: & \begin{cases} \{p \in \mathbb{Z}, \quad 1 \leq p \leq 5\} \\ \text{d}: 0 \\ \{q \in \mathbb{Z}, \quad 0 \leq q \leq 5\} \end{cases} \end{cases} \\ \bigcirc \text{ RF}: & \begin{cases} p_{\mathbf{x}_T}: & \begin{cases} \mathbf{y}: & \{\text{lag order}: d_y \in \mathbb{Z}, \quad -4 \leq d_y \leq 0\} \\ \mathbf{y} - \text{pred}(\mathbf{y}): & \{\text{lag order}: d_{\mathbf{y}-pred(\mathbf{y})} \in \mathbb{Z}, \quad -4 \leq d_{\mathbf{y}-pred(\mathbf{y})} \leq 0\} \end{cases} \\ p_{tr}: & \text{all} \\ \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{RF}: \begin{cases} \{m_{depth} \in \mathbb{Z}, \quad 5 \leq m_{depth} \leq 10\} \\ \{n_{est} = 10^{d_{n_{est}}}, \quad d_{n_{est}} \in \mathbb{Z}, \quad 1 \leq d_{n_{est}} \leq 3\} \end{cases} \end{cases} \end{cases}, \tag{2-23}$$

where the ARMA model originates from classical time series forecasting, and Random Forest (RF) from ML regression. The domain $\mathbb{S}$ contains six dimensions. Note that so-called 'hyperparameters' of an ARMA model actually instruct which lags of which (transformed) series

are included for a linear regression model. If we hierarchize $\mathbb{S}$ according to the proposed definition of a superparameter configuration, we are forced to mathematically unpack the ARMA model. The correct way to write $\mathbb{S}$ of the example in equation 2-23 is

$$\mathbb{S} = \begin{cases} p_{\mathbf{x}_T}: & \begin{cases} \mathbf{y}: & \{\text{lag order}: d_y \in \mathbb{Z}, \quad -4 \leq d_y \leq 0\} \\ \mathbf{y} - \text{pred}(\mathbf{y}): & \{\text{lag order}: d_{\mathbf{y}-pred(\mathbf{y})} \in \mathbb{Z}, \quad -4 \leq d_{\mathbf{y}-pred(\mathbf{y})} \leq 0\} \end{cases} \\ p_{tr}: & \text{all} \\ p_{\mathcal{H}}: & \begin{cases} \bigcirc \text{ linear}: & \begin{cases} \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{GD} \end{cases} \\ \bigcirc \text{ DTA}: & \begin{cases} \mathcal{L}_{tr}: & \text{MSE} \\ \mathcal{A}_{tr}: & \text{RF}: \begin{cases} \{m_{depth} \in \mathbb{Z}, \quad 5 \leq m_{depth} \leq 10\} \\ \{n_{est} = 10^{d_{n_{est}}}, \quad d_{n_{est}} \in \mathbb{Z}, \quad 1 \leq d_{n_{est}} \leq 3\} \end{cases} \end{cases} \end{cases} \end{cases} , \tag{2-24}$$

which contains the same exact model configurations, but has only four dimensions. This reduces the computational complexity of finding p^* in $\mathbb{S}$. This example shows the possible combination of the carefully selected feature engineering instructions from the classical forecasting paradigm, with the more sophisticated hypothesis classes and training optimisation algorithms of the ML regression paradigm, which is very promising. This is only an example. We can put many forecasting model components into this hierarchy. The classical forecasting paradigm's models (e.g. SARIMAX[13]) can be rewritten as a set in p_{x_T}. Neural networks and decision trees can be included in $p_{\mathcal{H}}$. Time series decomposition methods, for example CEEMDAN decomposition[15], can be included as feature transformation in p_{x_T}. Which models fit in the standardization is a matter of future exploration. The contributed theoretical standardization acts as an enabler to merge the learnings in forecasting from different paradigms.

2-5-3 Basis for learning from previous tasks

Learning from previous tasks is predicting p^*, using the characteristics of t. This is an ambitious goal, because collecting a sample set of previous tasks is very costly, the relation between t and p^* is highly complex, and the range $\mathbb{S}$ is complex and high dimensional. The first improvement of the theory is that it reduces the number of dimensions of $\mathbb{S}$, as explained in equation 2-24, which is a simplification.

Current researches in AutoML[11, 29, 36, 37] define 'some permutations in modeling' as a hyperparameter search space, and show the performance of their proposed algorithms over this domain. The second improvement of the proposed theory for learning from previous tasks, is hierarchy in t and $p^*(t)$ that divides up their complex relation into more manageable chunks. Generalizable relations can be found between between specific parts of the hierarchy in t and $p^*(t)$. This improves the degree to which AutoML researches can be built on each

other. Example hypothetical relations are the correlation of input measurements and the predicted measurement, in relation to feature selection superparameters in $\mathbb{S}_{\mathbf{x}_T}$, and the size of the dataset m in relation to the number of parameters in the hypothesis class of $\mathbb{S}_{\mathcal{H}}$.

2-5-4 Extension to multivariate inputs

Research in forecasting focuses mainly on univariate models[38, 6, 7, 8], or extensions of these. The increasing amount of data in the real world has led to more exogenous time series in the scenario of forecasting tasks, but a formalization of the problem setting is missing in literature. The formalization of superparameter evaluation fills this gap, by being applicable to multivariate time series. Exogenous time series can simply be included in the 'available observations'. In the feature engineering search space component, the transformations from the classical forecasting paradigm can be applied to the exogenous series. An increased length of input vector $\mathbf{x}_T$ (and a larger length of time series in the dataset m) increases the accuracy potential of ML models that are complex. They can be incorporated in the hypothesis training search space component.

Combining model components from different paradigms and including exogenous series is promising for forecasting tasks with big datasets. However, the superparameter evaluation function is very expensive in these settings, and the size of the search space is very big. How can one find a superparameter configuration that yields good performance on a forecasting task at hand?

Chapter 3

Superoptimization

The previous chapter, a framework for parametrizing the creation and evaluation process of hybrid forecasting models is presented as superparameter evaluation. The next objective is to:

Objective (2): Replace the human job of finding a well-performing modeling option with a computational job, by applying the AutoML framework to the parametrization.

In current practice, a human has the job to choose superparameter configuration $p \in \mathbb{S}$, the complete list of instructions for creating a forecasting model, for the task at hand. Search space $\mathbb{S}$ is the set of p's considered. The objective in this choice is to minimize the superparameter evaluation $I(p,t)$ for the task at hand t. The human approach is to start evaluating a set of promising p's, denoted as $\mathbf{p}_0 \subset \mathbb{S}$, inferred from its experience in previous tasks $\mathbb{T}^*$. Then the human performs an iterative job of suggesting promising p's for evaluation, based on previous evaluations, until satisfaction. Finally, the best-performing p for t, denoted as $p^*(t)$, is used for production. The human job is illustrated in Figure 3-1.

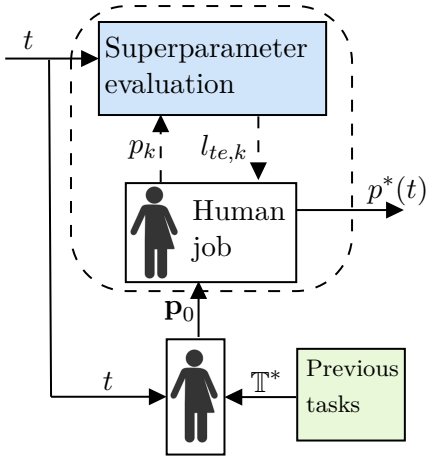


Figure 3-1: The human job of finding a well-performing superparameter configuration $p^*(t)$ for the forecasting task t at hand. Firstly note that the model options in the knowledge of a human can be expressed as a search space $\mathbb{S}$, containing p's. When a new task is at hand, for example the apple sales example in Figure 2-3, the human firstly looks at the previous task set $\mathbb{T}^*$ to find similar tasks. Example similar tasks are apple sales tasks from other stores, or other fruit sales tasks. Their well-performing superparameter configurations $p^*(t_{prev})$ are used as (promising) initial suggestions $\mathbf{p}_{k=0}$ for superparameter evaluation on the task at hand. Using the performance index $l_{te,k}$ of previous iterations k, the human gets an idea of where the high performing regions of p are. Then an iterative process starts, of which the current iteration is denoted as k. During iteration k, the human suggests p_k and updates its ideas on high performing regions using the corresponding $l_{te,k}$. Iterations are repeated until the human is satisfied, and the best-so-far superparameter configuration $p^*(t)$ is used for production.

This job is mathematically represented as

$$\begin{aligned} \min_{p} \quad & I(p,t) \\ \text{s.t.} \quad & p \in \mathbb{S} \subset \mathbb{P}, \end{aligned} \tag{3-1}$$

and we define it as superoptimization. This research replaces the human job with a superoptimization algorithm $\mathcal{A}_s$. It iteratively suggests $p \in \mathbb{S}$ for superparameter evaluation. The search space $\mathbb{S}$ is the set of considered superparameter configurations. In business $\mathbb{S}$ consists of the forecast model components incorporated in the implementation. The set $\mathbb{S}$ is a subset of $\mathbb{P}$, a nonconvex, multidimensional, mixed-type and conditional set of all possible superparameter configurations, as defined in equation 2-20. The evaluation function $I(p,t)$ is computationally expensive (it contains an optimization), and has no derivative available. The goal of superoptimization is two-fold:

- Find a $p \in \mathbb{S}$ that minimizes $I(p,t)$ for a given t, denoted as $p^*(t)$.
- Minimize the computational resources for finding $p^*(t)$.

The performance of a superoptimization algorithm is displayed by plotting the best-so-far found performance index l_{te}, against the computational resources, as illustrated in Figure 1-2. Section 3-1 explains $\mathcal{A}_s$, and how to learn from previous evaluations. Section 3-2 explains how to learn from previous tasks.

3-1 Superoptimization algorithm

The superoptimization algorithm $\mathcal{A}_s$ replaces the human job of iteratively suggesting promising p's, by looking at which p's yielded a good or bad performance index in the evaluation history $\mathbb{E}$. The job of $\mathcal{A}_s$ is illustrated in Figure 3-2.

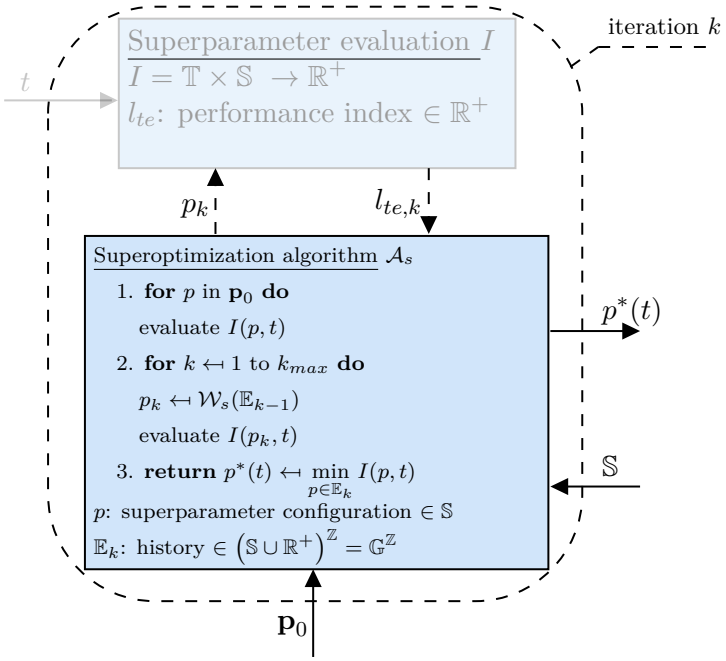


Figure 3-2: Superoptimization algorithm. When a new forecasting task t is at hand, for example the apple sales task in Figure 2-3, the superoptimization algorithm $\mathcal{A}_s$ searches for a well-performing superparameter configuration $p^*(t)$ in the search space $\mathbb{S}$. Note that after every evaluation the result $\{p_k \cup l_{te,k}\}$ of iteration k is stored in the evaluation history $\mathbb{E}$, so we have version $\mathbb{E}_k$ for iteration k. The superoptimization algorithm starts out by evaluating an initial set of p's, denoted as $\mathbf{p}_0$. In the case that $\mathbf{p}_0$ is not inferred from previous tasks, $\mathbf{p}_0$ is randomly drawn from $\mathbb{S}$. After this initialization, the function $\mathcal{W}_s$ suggests a promising $p \in \mathbb{S}$, based on $\mathbb{E}_{k-1}$. When the maximum number of iterations k_{max} is reached, the best-so-far superparameter configuration is outputted as $p^*(t)$.

As explained in section 1, the increase of the amount of data has gone hand in hand with popularity of more complex regression models in the Machine Learning (ML) regression paradigm. These models commonly have an increased amount of superparameters in $p_{\mathcal{H}}$ in combination with computationally complex evaluation functions[29], making the reduction of CPU cycles by $\mathcal{A}_s$ a popular research topic[36]. The algorithms that tackle this problem differ in suggestion function $\mathcal{W}_s$. This section discusses the two main classes of algorithms in section 3-1-1 and section 3-1-2.

3-1-1 Sequential model-based optimization algorithms

Recently a lot of studies have focused on the traditional black-box optimization method Sequential Model Based Optimization (SMBO)[36]. The intuition behind this approach is that

suggestion function $\mathcal{W}_s$ makes an informed decision, based upon the entire set of previous evaluations $\mathbb{E}_k$, by exploiting area's of good performance and exploring area's where few samples have been evaluated. In SMBO, $\mathcal{W}_s$ consists of training a surrogate model M and retrieving a suggested p_k from this M. The model is trained on $\mathbb{E}_{k-1}$, and functions as a cheap surrogate of the expensive objective function I. The model M then suggests some p's and selects the p that optimizes an acquisition function. SMBO approaches differ in hypothesis class for M and acquisition function. An example SMBO algorithm is SMAC[39], which uses a Random Forest as surrogate model.

3-1-2 Population-based optimization algorithms

Another class of algorithms for superoptimization is the Population-based Optimization Algorithm (POA)[40]. The core difference with SMBO is that $\mathcal{W}_s$ only uses the part of $\mathbb{E}_{k-1}$ with a good performance index l_{te}, referred to as the population. The suggestion function uses the current population to infer a new suggested batch of p's from for evaluation. An example is the evolutionary algorithm[41], which uses selection, recombination and mutation to evolve the search towards superparameter regions of high accuracy.

The bottleneck for these algorithms, is that for every new task, superoptimization starts from scratch, with a randomly drawn $\mathbf{p}_0$.

3-2 Metalearning

In the human process of superoptimization, as illustrated in Figure 3-1, the forecaster finds a previous task t that is highly similar and uses its best performing superparameter configuration(s) $p^*(t_{prev})$ as a promising starting point. This job is replaced by a computational process called metalearning, as illustrated in Figure 3-3. It uses the set of previous tasks $\mathbb{T}^*$ to retrieve a promising initialization batch, also referred to as a warmstart[3] $\mathbf{p}_0$. The warmstart $\mathbf{p}_0$ is fed into superoptimization algorithm $\mathcal{A}_s$, to improve the optimization of equation 3-1. The components of the metalearning process are the metafeature encoder $\mathcal{W}_e$, the set of completed tasks $\mathbb{T}^*$, and the metalearner $\mathcal{W}_m$.

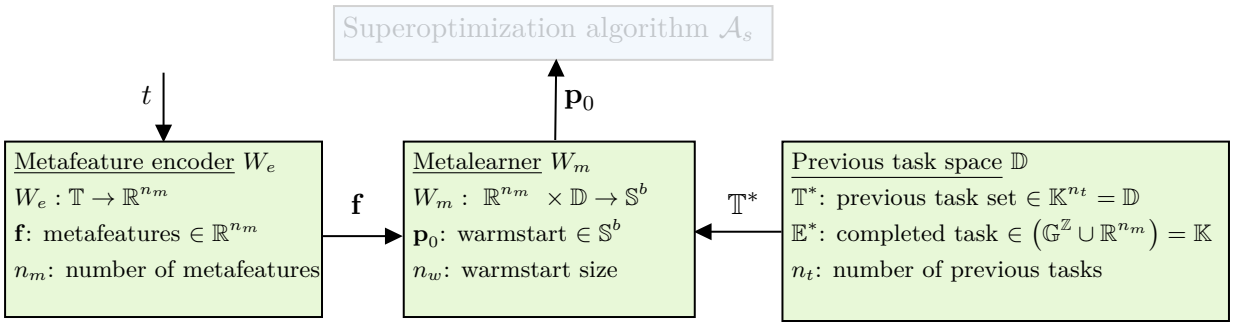


Figure 3-3: Metalearning. When a new forecasting task t is at hand, for example the apple sales task in Figure 2-3, $\mathcal{W}_e$ computes a metafeature vector $\mathbf{f}$ from the dataset, as a low-dimensional characteristic representation. The metalearner $\mathcal{W}_m$ then uses $\mathbf{f}$ to find similar previous tasks in the previous task set $\mathbb{T}^*$, to suggests a promising set of p's to initialize $\mathcal{A}_s$ with, also referred to as the warmstart $\mathbf{p}_0$. As explained in Figure 3-2, the superoptimization algorithm $\mathcal{A}_s$ evaluates $\mathbf{p}_0$, and then proceeds with its own iterative process.

The metafeature encoder $\mathcal{W}_e$,

$$\mathcal{W}_e : \mathbb{T} \to \mathbb{R}^{n_m}, \tag{3-2}$$

maps the task at hand $t \in \mathbb{T}$, to a vector $\mathbf{f} \in \mathbb{R}^{n_m}$ of metafeatures, where n_m is the number of metafeatures. A metafeature is a measure that characterizes t. Complementary to the forecast scenario characteristics explained in section 2-2, statistical measures[4] can be calculated from m, to obtain a lower dimensional representation of m for in $\mathbf{f}$. Some examples are the length of the time series in m and the correlation between lags of (transformed) time series and the target.

Completed tasks are stored in the previous task set $\mathbb{T}^*$,

$$\mathbb{T}^* \in \mathbb{K}^{n_t} = \mathbb{D}, \tag{3-3}$$

which is a set of completed tasks, where a completed task $\mathbb{E}^*$,

$$\mathbb{E}^* \in (\mathbb{G}^{\mathbb{Z}} \cup \mathbb{R}^{n_m}) = \mathbb{K}, \tag{3-4}$$

is defined as a conjunction of the evaluation history $\mathbb{E}_{k_{max}} \in \mathbb{G}^{\mathbb{Z}}$ and metafeatures $\mathbf{f} \in \mathbb{R}^{n_m}$ of a previous task.

The metalearner $\mathcal{W}_m$,

$$\mathcal{W}_m : \mathbb{R}^{n_m} \times \mathbb{D} \to \mathbb{S}^b, \tag{3-5}$$

retrieves a warmstart $\mathbf{p}_0 \subset \mathbb{S}$ for the superoptimization of t, using $\mathbf{f}$ and $\mathbb{T}^*$. The number of p's in $\mathbf{p}_0$ is denoted by b. This approach naturally imitates a data scientist, who starts superoptimization using knowledge from previous tasks. The goal is to suggest a warmstart with well-performing superparameter configurations, to start out as close as possible to the optimal

solution and thereby spare computational resources. Two examples for the metalearner $\mathcal{W}_m$ are demonstrated in [11]. The K-Nearest Neighbors algorithm and the Random Forest (RF) algorithm are used to find the task in previous task set $\mathbb{T}^*$ most similar to t, and their b best performing p's are fed into $\mathcal{A}_s$ as warmstart $\mathbf{p}_0$. Experiments in metalearning have some big challenges[**?**]:

- defining metafeatures that correlate with $p^* \in \mathbb{S}$ for any t,
- composing a previous task set $\mathbb{T}^*$ for an experiment. If the tasks in $\mathbb{T}^*$ are too similar, superoptimization becomes a trivial process, as they share the same p^*. If the tasks in $\mathbb{T}^*$ are too dissimilar, cross-learning is ineffective. It is impossible to know the degree of similarity, before implementation and execution of an experiment. Experiments in metalearning characterize as both implementationally and computationally expensive.

3-3 AutoML framework applied to superparameter evaluation

The second aim of this research is to apply the AutoML framework to superparameter evaluation, to replace the human job of finding an well-performing forecasting modeling option, with a computational job. The task is defined as superoptimization, as presented in equation 3-1. It is performed by the superoptimization algorithm $\mathcal{A}_s$. In Figure 3-4, the automatic approach in (b) illustrates how the superoptimization algorithm replaces the job of the human forecaster (also see Figure 3-1) in (a), which learns from previous evaluations on the task at hand. The bottleneck in the automatic approach is that its naïve approach to a new task is computationally complex, because it does not learn from previous tasks. This research proposes learning from previous forecasting tasks $\mathbb{T}^*$. The metalearner suggests a warmstart $\mathbf{p}_0$, consisting of p's that performed well on similar previous tasks.

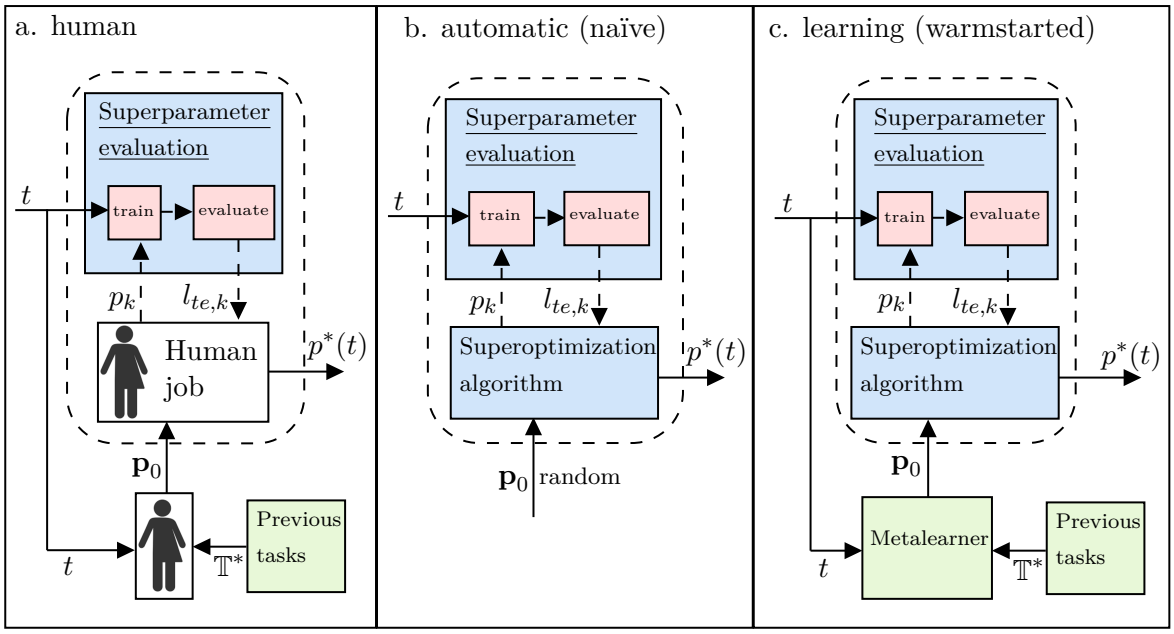


Figure 3-4: The human approach, versus the automatic approach, versus the AutoML framework that learns from previous tasks. The human approach (a) is explained in Figure 3-1. In the automatic approach (b), the human job of learning from previous evaluations on the task at hand is replaced by the superoptimization algorithm. In the approach that learns from previous tasks (c), the human job of learning from previous tasks $\mathbb{T}^*$ to suggest a warmstart $\mathbf{p}_0$, is replaced by the metalearner.

In Figure 3-5, the proposed framework for learning from previous forecasting tasks is illustrated in detail. At the top of each block in the diagram a referral is made to the section it is introduced in. The approach contains five steps:

1. The task at hand t is fed into the metafeature encoder $\mathcal{W}_e$, which maps it to the feature vector $\mathbf{f}$, a lower dimensional representation of t.
2. The task at hand is compared to the previous tasks $\mathbb{T}^*$, to produce a promising warmstart $\mathbf{p}_0 \subset \mathbb{S}$.
3. The superoptimization algorithm $\mathcal{A}_s$ is initialized with $\mathbf{p}_0$ and search space $\mathbb{S}$. The superparameter evaluation function I is initialized with t.
4. Running $\mathcal{A}_s$, which firstly evaluates $\mathbf{p}_0$ and then iteratively evaluates promising p's, by learning from evaluation history $\mathbb{E}_k$.
5. When the maximum number of iterations k_{max} is reached, the best performing superparameter configuration $p^*(t)$ is selected for production.

In the real world, completed tasks can be stored to gradually build out the previous task set, to improve the performance of $\mathcal{A}_s$.

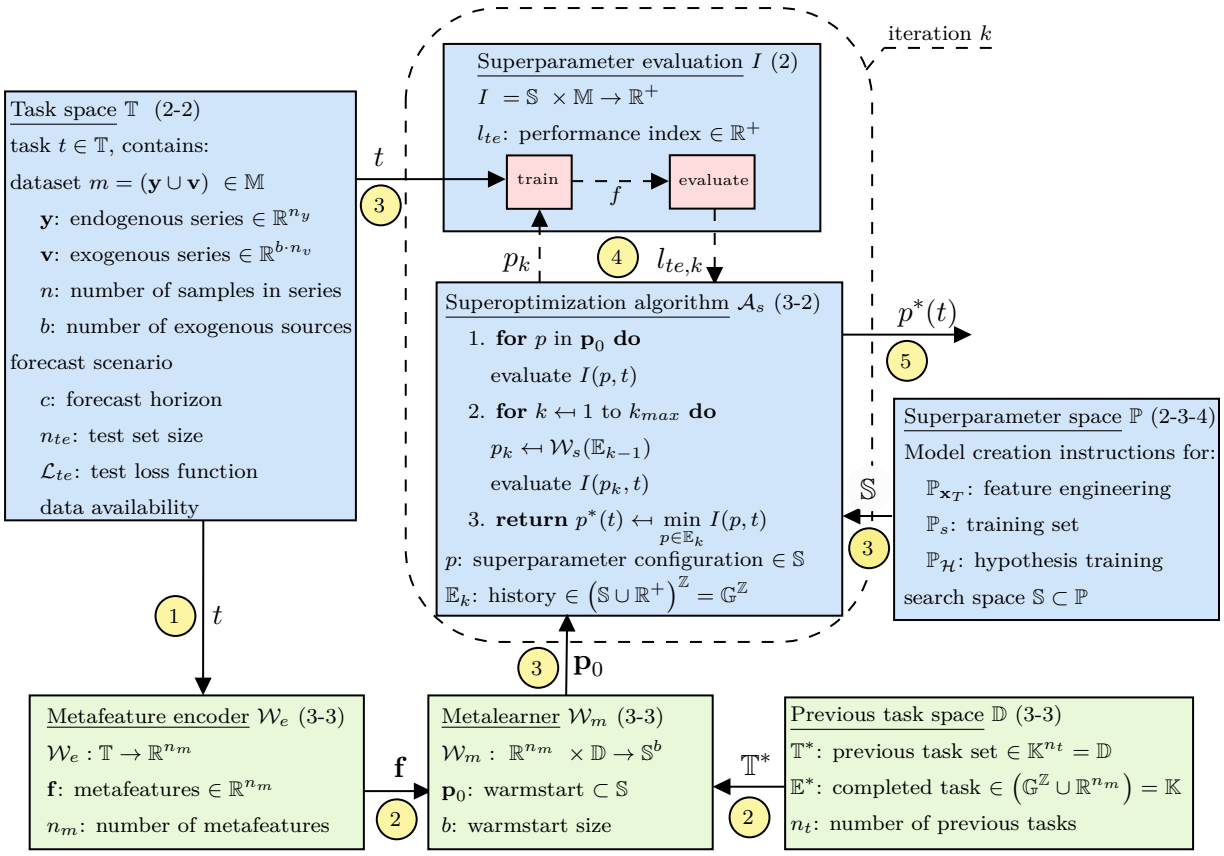


Figure 3-5: AutoML framework applied to superparameter evaluation, in detail. Note that at the top of each block, there is a reference to the section with a detailed explanation. A superparameter configuration $p \in \mathbb{P}$ is the complete list of instructions to produce a forecasting model $f(x)$. This explanation follows the steps, indicated by the yellow circles. In (1) the task t at hand, for example the apple sales task in Figure 2-3, is fed into the metafeature encoder $\mathcal{W}_e$, which maps it to a metafeature vector $\mathbf{f}$, which characterizes it. Then in (2), $\mathbf{f}$ is compared with the tasks in the previous task set $\mathbb{T}^*$, to obtain superparameter configurations that have yielded a good performance index $l_{te,k}$ for similar previous tasks. We refer to this output as the warmstart $\mathbf{p}_0 \subset \mathbb{S} \subset \mathbb{P}$. Step (3) is the initialization of the superoptimization process, performed by the superoptimization algorithm $\mathcal{A}_s$. The task is fed into the superparameter evaluation function I, which contains the training and evaluation process, given $p \in \mathbb{S}$. The search space $\mathbb{S}$ is the domain of forecasting model creation instructions, for example the one in equation 2-19, and initializes $\mathcal{A}_s$. The warmstart $\mathbf{p}_0$ is also fed into $\mathcal{A}_s$. Then in (4) the superoptimization is performed, to obtain a p with a minimized l_{te}: firstly the p's in $\mathbf{p}_0$ are evaluated, and then the iterative process of $\mathcal{A}_s$ starts. The dashed lines represent iteration k of this process, in which the history of evaluations $\mathbb{E}_{k-1}$ is used by a suggestion function $\mathcal{W}_s$ to suggest a promising p_k for evaluation. This process ends when the maximum number of iterations k_{max} is reached. Finally, in (5) the best-found superparameter configuration $p^*(t)$ is outputted to use in production.

Chapter 4

Proposed method

The creation and evaluation process, applicable to a large set of hybrid forecasting models for a new task at hand, is standardized to superparameter evaluation $I(p, t)$ (chapter 2). Given a task t, it takes a superparameter configuration[1] p as input, and retrieves its performance index l_{te} as output. The AutoML framework is incorporated around this process (chapter 3) to define the superoptimization job of obtaining a well-performing forecasting model $p^*(t)$. The job can be performed by any algorithm that minimizes $I(p, t)$ with respect to $p \in \mathbb{S}^2$. As existing algorithms require too much runtime for this computational job, this research proposes a method for the objective to

(Objective 3) Reduce the runtime of the computational job, by learning from previous tasks.

4-1 Motivation

Existing algorithms learn from previous evaluations on the task at hand, as illustrated in Figure 3-4(b). A new method[11], referred to as warmstarting, also incorporates learnings from previous tasks, as illustrated in Figure 3-4(c). When a new task is at hand, it finds the previous task most similar (w.r.t. their metafeatures) in the previous task set $\mathbb{T}^*$, and suggest its best performing superparameter configuration as an initialization batch to the superoptimization algorithm. The research applies the method to classification tasks, and it uses 46 metafeatures on a previous task set of 57 tasks. The method has proven to work from a functional perspective. The paper however leaves the process of finding the metafeatures unexplained, and therefore its implications on overfitting, which is concerning in a context of only 57 tasks. On top of that, the method should have been better benchmarked, as explained in 5-1. This means that the metalearner might not work on tasks outside of the included previous task set. The generalizability of the conclusion on whether the right metafeatures are used is therefore questionable. Without an experiment that maximizes the weight of the

[1] the complete list of instructions for a hybrid model to be set before training and evaluation

[2] search space $\mathbb{S}$ is the considered set of superparameter configurations, for example equation 2-24

evidence that a set of metafeatures has a relation with good performing parts in the search space $\mathbb{S}$, the method is just a complex functionality, inapplicable to the real world.

This research aims to provide evidence for which metafeatures should be used in warmstarting, for the above mentioned objective. The motivations in formulation of the proposed method are therefore:

- propose few metafeatures,
- that have a justifiable relation with a demarcated subset of superparameters,
- apply a simple metalearner.

4-2 Hypothesis

The proposed metafeatures (in metafeature encoder $\mathcal{W}_e$) are size and feature richness. The demarcated subset of the superparameters is the hypothesis training superparameters $p_{\mathcal{H}}$. The metalearner $\mathcal{W}_m$ calculates similarity using the L1-norm. They are explained in section 4-3 and 4-4. The idea is that the size and feature richness in the training set of a task t, have a relation with the good performing superparameters for hypothesis training $p^*_{tr}(t)$. This gives rise to the hypothesis that:

(Hypothesis) Runtime of superoptimizing hypothesis training superparameters reduces, if size and input richness are metafeatures in warmstarting.

The demarcation of the hypothesis is illustrated in Figure 4-1.

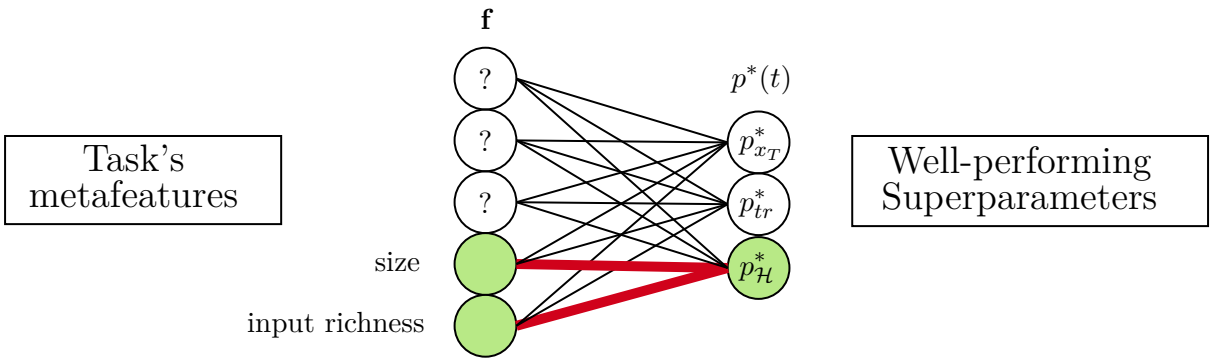


Figure 4-1: Experiment hypothesis demarcation. This research demarcates to finding metafeatures that have a relation with well-performing hypothesis training superparameter configurations $p_{\mathcal{H}}$. Future research is to be done on finding metafeatures that have a relation with the other components of a superparameter configuration.

4-3 Metafeature encoder

The metafeature encoder $\mathcal{W}_e$,

$$\mathcal{W}_e : \mathbb{T} \to \mathbb{R}^{n_m}, \tag{4-1}$$

maps the task t to a vector of metafeatures $\mathbf{f}$, as explained in section 3-2. This experiment's metafeatures and their substantiated hypothetical relations with well performing parts of the search space are:

- **Size**: the number of samples in the training set n_{tr} is hypothesized to indicate well-performing regions of complexity superparameters. When a model has a higher degree of complexity, it has a larger parameter vector $\mathbf{w}$ to train (the training process is explained in section 2-12). The metafeature hypothesis is that more samples are needed when more parameters have to be trained. An example is the number of trees superparameter, as explained in Appendix A, which has to be set higher when more training samples are used.
- **Input richness**: an estimation of how predictive the inputs in $\mathbf{x}_T$ are for the predicted variable y, is hypothesized to indicate well-performing regions of overfitting superparameters. When the input richness is low, there is a high amount of noise, which means that the forecasting model benefits from a higher degree of protective measures against overfitting. An example is the minimum child weight superparameter, as explained in Appendix A, which would have to be set higher when inputs contain more noise, in order to average a prediction over a higher number of input samples.

Other hypothesis training classes from the Machine Learning (ML) paradigm (for example neural networks) also have superparameters that either set the model complexity or the degree of overfitting, just like XGBoost. It is therefore plausible that conclusions on these hypotheses generalize outside of this experiment's search space. The hypothesized relation between metafeatures and well-performing regions in the search space is illustrated in Figure 4-2.

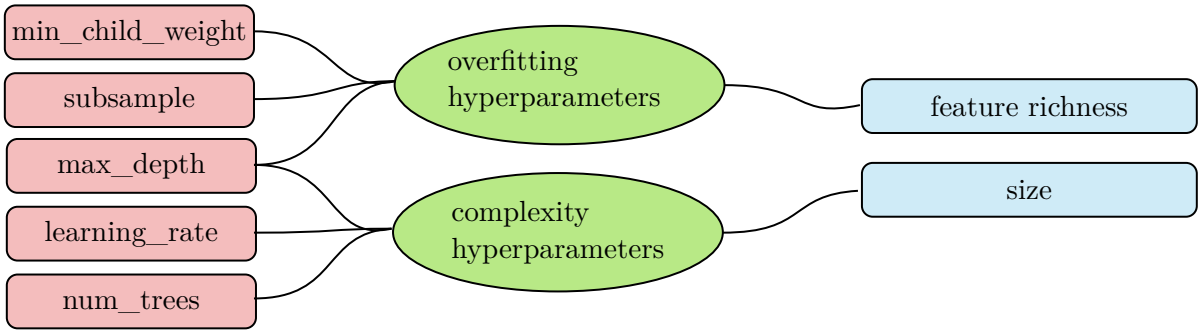


Figure 4-2: Metafeature relations

Input richness is measured by decorrelating the inputs, and accumulating their univariate feature importance with respect to the target. The way of measuring univariate feature importance is Spearman's rank correlation[3] [42],

$$r_{s,j} = 1 - \frac{6 \sum d_{i,j}^2}{n^3 - n} \quad \text{and} \quad f = \sum_j r_{s,j} \tag{4-2}$$

[3]it corresponds best with XGBoost's binning nature, and XGBoost is the search space of the experiment setting.

where $d_{i,j}$ is the difference of the rank between the target and feature j of observation i and n is the number of observations. The rank correlation of all features are summated to obtain a scalar, characterizing feature richness. Note that this method has some limitations. Firstly, the feature interdependencies are not incorporated. Secondly, the interactions with the hypothesis class are not included in this measure.

4-4 Metalearner

The metalearner $\mathcal{W}_m$,

$$\mathcal{W}_m : \mathbb{R}^{n_m} \times \mathbb{D} \rightarrow \mathbb{S}^b, \tag{4-3}$$

maps the metafeatures $\mathbf{f}$ and the previous task set $\mathbb{T}^*$ to a warmstart $\mathbf{p}_0 \subset \mathbb{S}$, a promising set of superparameter configurations p_0 to initialize the Bayesian optimization with. Analogous to the metafeature encoder, this research deviates from previous studies [11, 43] which use a complex metalearner, by choosing a simple one, in order to prevent overfitting on meta level. The chosen simple metalearner calculates the L1-norm between the metafeatures of the task at hand and previous tasks,

$$d(t_i, t_j) = ||\mathbf{f}(t_i), \mathbf{f}(t_j)||_1, \tag{4-4}$$

to rank previous tasks in $\mathbb{D}$ on similarity. The r best performing configurations of the k most similar datasets are collected, of which the b most occurring configurations constitute $\mathbf{p}_0$. The parameters r, k and b, also referred to as metaparameters, are the only parameters that are manually set in the proposed method. They are adjusted only slightly to obtain meaningful results, while preventing overfitting on the previous task set.

Chapter 5

Experiment

In the experiment, the proposed method performs superoptimization, given a task at hand. This job is performed for multiple tasks. The experiment setting is defined to answer the hypothesis:

(Hypothesis) Runtime of superoptimizing hypothesis training superparameters reduces, if size and input richness are metafeatures in warmstarting.

Benchmark methods perform the same job, to compare the performances of the methods. The benchmarks are explained in 5-1, the set of tasks is explained in section 5-2, and the search space is explained in section 5-3. The superoptimization algorithm is defined in **??**. Finally, the experiment execution is explained in section 5-5, including the published open-source Python package **warmstart**, which provides an implementational basis for future research in metalearning.

5-1 Benchmarks

The proposed method is benchmarked against three algorithms.

Benchmark algorithm 1: Random Search

The first benchmark algorithm is Random Search[37], whose suggestion function $\mathcal{W}_s$ suggest random samples from the search space.

Benchmark algorithm 2: naïve

In this benchmark algorithm, the warmstart $\mathbf{p}_0$ of the proposed method is replaced with a random start.

This benchmark is used by [11] to compare their metalearning method with, while it does not provide insight on whether the metalearning method is learning from previous tasks. Even if

the metalearner is well-designed, it could be outperformed by the naive approach, when the previous task set is too small or the tasks in the previous task set are too different. On the other hand, if the metalearner is ill-designed, it could still outperform the naive approach, when the search space is too wide. In other words, a superparameter configuration suggested from ány previous task could be helpful, no matter the similarity of the previous task. This insight is of crucial importance in drawing conclusions from the results of a metalearning experiment, and has gotten no attention in previous work.

Benchmark algorithm 3: Coldstarted

This research proposes a coldstart as a benchmark algorithm for metalearning, as illustrated in Figure 5-1. The coldstart uses the same metalearning method, but as opposed to the most similar previous tasks, the least similar previous tasks are selected to suggest $\mathbf{p}_0 \subset \mathbb{S}$. By comparison with this benchmark, more reliable conclusions on the defined experiment hypothesis can be drawn.

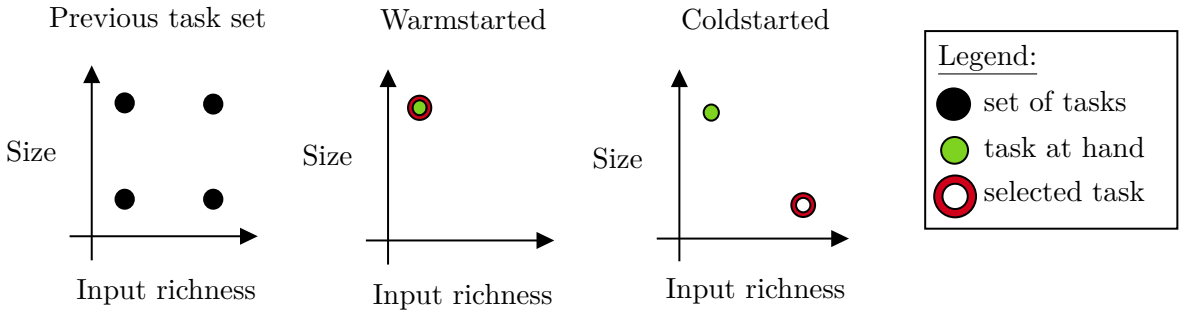


Figure 5-1: Task selected from previous task set $\mathbb{T}^*$ to suggest initialization batch $\mathbf{p}_0$ from (see figure Figure 3-4), for the warmstart method and coldstart benchmark. Selecting the most similar previous task to suggest its best superparameter configuration to the task at hand, referred to as warmstart, should outperform its reversed method of selecting the least similar previous task, referred to as coldstart.

5-2 Forecasting task set

This section explains the set of forecasting tasks, used in the experiment. The forecasting tasks of interest characterize as follows:

- The dataset m contains a large number of indices n for which measurements are stored.
- The dataset m contains exogenous time series $\mathbf{v}$.
- The endogenous series $\mathbf{y}$ of the dataset m represents a complex process, of which the forecasting can benefit from complex forecasting models.

At the time of writing, no public benchmark set of forecasting, matching the criteria is available. The internship company generously provides a dataset of 16 forecasting tasks as a public benchmark dataset. The tasks have the following characteristics:

- The dataset contains measurements of a complex process in the trading industry, of which the forecasting can benefit from complex forecasting models.
- The dataset contains one endogenous series $\mathbf{y}$ and one exogenous series $\mathbf{v}$ with three years of hourly measurements.
- The exogenous measurements are available up until $T + 1$.
- The time series in m have no level, which means they fluctuate around 0.

The tasks are grouped into two different processes that differ in input richness, so 8 tasks per process. Every task is then also divided into one with a big dataset, and one with a small dataset (by cutting a part off the data). This means there are 4 groups of 8 tasks, and a total of 32 tasks, as shown in Figure 5-2. This setting is chosen, because in this task set there are very similar tasks and very dissimilar tasks. If the experiment's hypothesis is true, the chances are high that the results can conclude so.

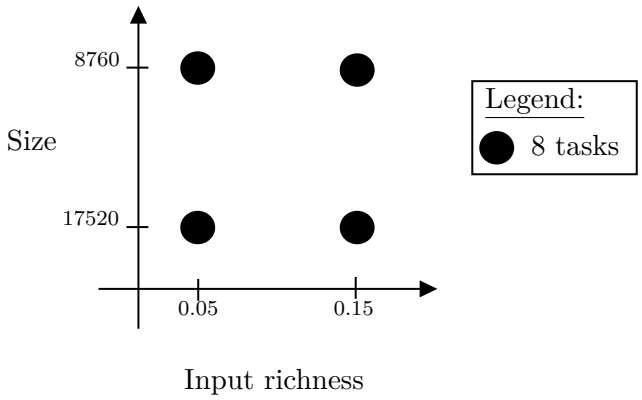


Figure 5-2: The groups of tasks in the experiment are divided into 4 groups of different size and input richness.

The shared forecasting scenario of tasks in the previous task set is illustrated in Figure 5-3.

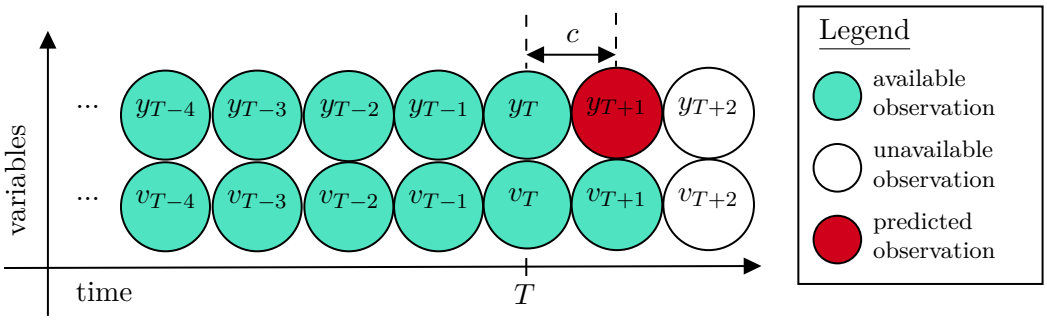


Figure 5-3: The forecast horizon and data availability of the previous task set. The forecast horizon $c = 1$, and the exogenous measurements are available up until $T + 1$.

The test loss function is the mean squared error[21],

$$\mathcal{L}_{te}(\hat{Y}_{te}, Y_{te}) = \frac{1}{n_{te}} \sum_{i=1}^{n_{te}} \left| Y_{te,i} - \hat{Y}_{te,i} \right|, \tag{5-1}$$

on a test set of $\frac{3}{4}$ years ($n_{te} = 6570$). This research uses a simple, intuïtive and general test loss function, because the choice for the right test loss function is specific for the use case of a forecasting task, and this outside the research scope. Superoptimization overfitting is evaluated on a crossvalidation set of $\frac{1}{4}$ years ($n_{cv} = 2190$).

5-3 Search space

This section motivates the demarcation of the search space for the experiment. The considerations in this demarcation process are defined as follows:

1. **relevance**: the search space should result in a well-performing forecasting model for every task in the previous task set.
2. **feasibility**: the experiment resulting from the search space should not demand a too high computational complexity.
3. **generalizability**: conclusions from the experiment on this search space should be plausible to generalize to other search spaces and forecasting tasks, and therefore develop AutoML for forecasting in a general sense.

In subsection 5-3-1 the fixed feature engineering instructions are explained, in subsection 5-3-2 the fixed training set instructions are specified, and subsection 5-3-3 explains the hypothesis training search space on which the experiment focuses. Finally, in 5-3-4 the complete search space is presented.

5-3-1 Feature engineering

The experiment's feature engineering instruction set in $\mathbb{S}_{\mathbf{x}_T}$ contains a single member. The index of $\mathbf{y}$ is transformed to time series, representing the hour of day, the day number of the week, the day number of the year, and a categorical variable representing whether the day is a holiday. The observations of the predicted index are selected to form input vector $\mathbf{x}_T$. A relatively small number of inputs is chosen, taking into account the feasibility of the experiment. While this is a rigorous simplification of the superparameter space, it is plausible that the conclusions on the metalearning specific hypothesis in section [**?**] generalize to other feature engineering instructions. The feature engineering instructions in $\mathbb{S}_{\mathbf{x}_T}$ are written according to section 2-3-1 as

$$\mathbb{S}_{\mathbf{x}_T} = \begin{cases} \mathbf{y}: & \{\} \\ \mathbf{v}^1: & \{1\} \\ \text{hour of day}(\mathbf{y}): & \{1\} \\ \text{day of week}(\mathbf{y}): & \{1\} \\ \text{day of year}(\mathbf{y}): & \{1\} \\ \text{holiday}(\mathbf{y}): & \{1\} \end{cases},$$

and is illustrated in Figure 5-4.

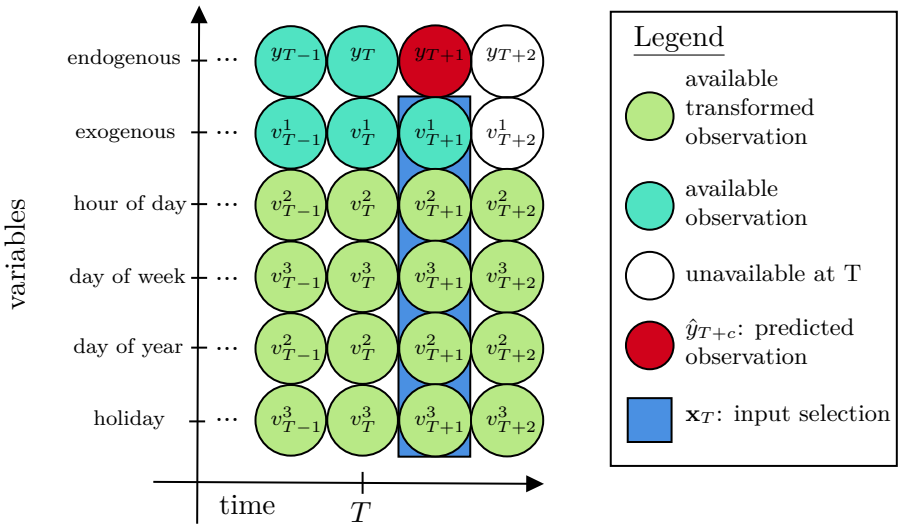


Figure 5-4: The fixed feature engineering process $\mathcal{I}_{\mathbf{x}_T}$ for creating input vector $\mathbf{x}_T$ at T.

5-3-2 Splitting

The training set formation instruction is to, after the reduction of the test and crossvalidation samples, simply select all remaining samples as training set $X_{tr} \cup Y_{tr}$. The single member $\mathbb{S}_{tr}$, writting according to section 2-3-2, is

$$\mathbb{S}_{tr} = \quad \{\text{all}\}.$$

It is also plausible that the conclusions on the metalearning specific hypothesis in section [**?**] generalize to other training set formation instructions.

5-3-3 Hypothesis training

The hypothesis training search space $\mathbb{S}_{\mathcal{H}}$ of the experiment is in the domain of XGBoost, originating from the Machine Learning (ML) paradigm. The motivation for this choice is that:

- it has proven to yield an excellent performance on a very broad range of tasks[**?**], which makes it a relevant and generalizable domain of interest.
- the numerical method and implementation is known for its low computational complexity with respect to the achieved performance[**?**].

XGBoost is the optimization of a Decision Tree Summation (DTS) as hypothesis class $\mathcal{H}$, performed by the Newton Boosting algorithm as training algorithm $\mathcal{A}_{tr}$, with respect to a specific regularized loss function as training loss function $\mathcal{L}_{tr}$. It is explained in detail in Appendix A. XGBoost's superparameters can be divided into complexity and overfitting superparameters. The superparameters that are related to the model complexity are the number of trees, the maximum tree depth, and the learning rate. The superparameters that determine the protection against overfitting in $\mathcal{L}_{tr}$ and $\mathcal{A}_{tr}$ are γ, subsample, column sample per tree and regalizer α. There are no publications on how to set the superparameters. By reading blog posts and manual analysis, this research defines the following $\mathbb{S}_{\mathcal{H}}$ of interest for the task domain, written according the superparameter standardization rules of equation 2-20:

$$\mathbb{S}_{\mathcal{H}} = p_{\mathcal{H}}: \quad \text{DTS}: \quad \begin{cases} \mathcal{H}: & \begin{cases} \text{num\_trees}: & \{d_{nt} \in \mathbb{Z}, \quad 1 \leq d_{nt} \leq 800\} \\ \text{max\_depth}: & \{d_{md} \in \mathbb{Z}, \quad 1 \leq d_{md} \leq 40\} \\ \text{min\_child\_weight}: & \{d_{mcw} \in \mathbb{Z}, \quad 1 \leq d_{mcw} \leq 30\} \end{cases} \\ \mathcal{L}_{tr}: & \text{regularized loss}: \begin{cases} \gamma: & \{d_\gamma = 10^{\lambda_\gamma}, \lambda_\gamma \in \mathbb{R}, 0 \leq \lambda_\gamma \leq 1\} \\ \alpha: & \{d_\alpha = 10^{\lambda_\alpha}, \lambda_\alpha \in \mathbb{R}, -4 \leq \lambda_\alpha \leq 2\} \end{cases} \\ \mathcal{A}_{tr}: & \text{Newton Boosting}: \begin{cases} \text{learning\_rate}: & \{d_{lr} = 10^{\lambda_{lr}}, \lambda_{lr} \in \mathbb{R}, -6 \leq \lambda_{lr} \leq 0\} \\ \text{subsample}: & \{d_s \in \mathbb{R}, 0.5 \leq d_s \leq 1.0\} \\ \text{colsample\_by\_tree}: & \{d_{cbt} \in \mathbb{R}, 0.5 \leq d_{cbt} \leq 1.0\} \end{cases} \end{cases} ,$$

This research is focused on proposing a metalearning method that improves a superoptimization algorithm for this component of the superparameter space.

5-3-4 Complete search space

The complete initial search space is written as follows:

$$
\mathbb{S} = \begin{cases}
p_{\mathbf{x}_T}: & \begin{cases}
\mathbf{y}: & \{\} \\
\mathbf{v}^1: & \{1\} \\
\text{hour of day}(\mathbf{y}): & \{1\} \\
\text{day of week}(\mathbf{y}): & \{1\} \\
\text{day of year}(\mathbf{y}): & \{1\} \\
\text{holiday}(\mathbf{y}): & \{1\}
\end{cases} \\
p_{tr}: & \text{all} \\
p_{\mathcal{H}}: & \text{DTS}: \begin{cases}
\mathcal{H}: & \begin{cases}
\text{num\_trees}: & \{d_{nt} \in \mathbb{Z}, \quad 1 \leq d_{nt} \leq 800\} \\
\text{max\_depth}: & \{d_{md} \in \mathbb{Z}, \quad 1 \leq d_{md} \leq 40\} \\
\text{min\_child\_weight}: & \{d_{mcw} \in \mathbb{Z}, \quad 1 \leq d_{mcw} \leq 30\}
\end{cases} \\
\mathcal{L}_{tr}: & \text{regularized loss}: \begin{cases}
\gamma: & \{d_\gamma = 10^{\lambda_\gamma}, \lambda_\gamma \in \mathbb{R}, 0 \leq \lambda_\gamma \leq 1\} \\
\alpha: & \{d_\alpha = 10^{\lambda_\alpha}, \lambda_\alpha \in \mathbb{R}, -4 \leq \lambda_\alpha \leq 2\}
\end{cases} \\
\mathcal{A}_{tr}: & \text{Newton Boosting}: \begin{cases}
\text{learning\_rate}: & \{d_{lr} = 10^{\lambda_{lr}}, \lambda_{lr} \in \mathbb{R}, -6 \leq \lambda_{lr} \leq 0\} \\
\text{subsample}: & \{d_s \in \mathbb{R}, 0.5 \leq d_s \leq 1.0\} \\
\text{colsample\_by\_tree}: & \{d_{cbt} \in \mathbb{R}, 0.5 \leq d_{cbt} \leq 1.0\}
\end{cases}
\end{cases}
\end{cases}
$$

5-4 Superoptimization algorithm

As superoptimization algorithm $\mathcal{A}_s$ (explained in section 3-1), the experiment uses Bayesian optimization, more specifically the Tree Parzen Estimator (TPE) algorithm[36]. It falls under the Sequential Model Based Optimization (SMBO) domain, as explained in section 3-1-1. The motivations for choosing TPE are that

- the algorithm has recently had a lot of attention from the ML community for 'hyper-parameter' optimization, making it a relevant benchmark.
- the algorithm can benefit from metalearning, because of its tendency to exploit spaces of well performance and to explore unknown spaces.
- the open source implementation **hyperopt** is compatible with conditional search spaces, and it is compatible with warmstarting (see section 3-2), which gives the possibility to feed in $\mathbf{p}_0 \subset \mathbb{S}$ for initial evaluation.

In Appendix B the exact suggestion function $\mathcal{W}_s$ of the algorithm is explained in detail, but the intuïtive explanation is as follows:

1. Make two Gaussian Mixture Models, using evaluation history $\mathbb{E}_k$, for the probability for p of either being in 'the good group' or the 'bad group', in terms of their performance index l_{te}.

2. Retrieve some samples from 'the good group' distribution.

3. For suggestion output, select one of these samples, which has a high probability of being in 'the good group', whilst having a low probability of being in 'the bad group'.

In the default setting of **hyperopt**'s implementation, before suggesting p_k at every iteration k, a 'naïve' initial set $\mathbf{p}_0$ of 20 randomly drawn p's is evaluated. The metafeature encoder and metalearner replaces this with a warmstart $\mathbf{p}_0$.

5-5 Experiment execution

This section explains the execution of the experiment, which is summarized in Figure 5-5. Section 5-5-1 explains how statistical significance of the results is achieved, which raises the issue of a too high computational complexity. In section 5-5-2 and 5-5-3 the method for reducing the computational complexity is discussed. Finally, this research introduces an open-source Python package for metalearning experiments in section 5-5-4.

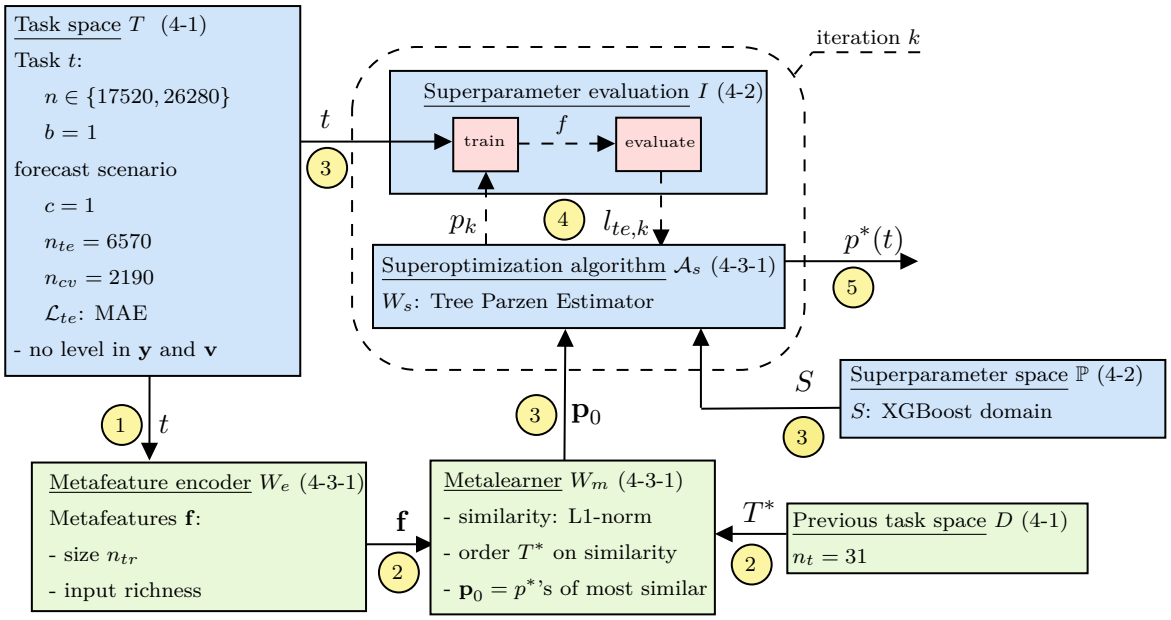


Figure 5-5: Experiment summary, showing the proposed method, the task domain and the search space. This figure summarizes the experiment, in the structure of Figure 3-3. Note that at the top of each block, there is a reference to the section with a detailed explanation and motivation of the design choice. The task contains 2 or 3 years of hourly endogenous time series and $b = 1$ exogenous time series (so $n \in \{17520, 26280\}$). The forecast scenario is predicting $c = 1$ step ahead, and the predictions will be tested by the MAE on a left out test set of $n_{te} = 6570$ samples, and $n_cv = 2190$ samples are held apart as crossvalidation set. In the first step, the task t is fed into the metafeature encoder W_e, which calculates the metafeatures 'size' and 'input richness'. The metalearner $\mathcal{W}_m$ then retrieves the best superparameter configurations for the most similar datasets $\mathbf{p}_0$. The superoptimization process is then initialized. The warmstart $\mathbf{p}_0$ and search space, which is in the XGBoost domain, initializes the superoptimization algorithm $\mathcal{A}_s$. Superparameter evaluation I is initialized with t. In the fourth step, $\mathcal{A}_s$ evaluates $\mathbf{p}_0$, and then the Tree Parzen Estimator iteratively suggests promising p's for evaluation. Finally the process outputs the best found p as $p^*(t)$.

5-5-1 Statistical significance

The Tree Parzen Estimator is a stochastic procedure, because candidates for the EI-criterium are drawn from $l(p)$, which is a Gaussian Mixture Model. Furthermore, the naive approach has the stochastic component of a set of randomly drawn $\mathbf{p}_0 \subset \mathbb{S}$. We therefore perform 10 duplicates of the Bayesian optimization on a dataset and take the average of the results.

For the warmstarted approach we perform a 32-fold leave-one-out procedure, where one task is treated as the new unseen task, while the other 31 tasks and their best superparameter configurations $p^*(t)$ are included in the previous task set $\mathbb{T}^*$. At every iteration in the superoptimization, the compared approaches are ranked according to their performance index. Finally the ranks are averaged over all experiment duplicates and folds. Rank as a performance metric loses information of the degree to which an approach is better, but this metric is chosen because there are no metrics that correctly scale amongst multiple datasets. This even counts for the mean absolute scaled error (MASE)[25], which is dependent on the

predictability of a time series.

5-5-2 Result recycling

A practical constraint in the experiment is that an iteration in superoptimization contains a very computationally expensive superparameter evaluation. For this reason, trying out only a single set of metafeatures could result in a week of computations, while it is desirable to experiment on multiple permutations of the metalearning method. Result recycling solves this problem, dividing the experiment in an expensive offline and cheap online phase. During the offline phase the search space is discretized into a grid of superparameter configurations (instead of a continuous space). Superparameter evaluation is run for the 138.240 options in the task and search space (32 t's, 4320 p's as explained in 5-5-3), and stored in a lookup table. The grid search yields a walltime of 45 days for a 2.7 GHz i7 processor. An optimally parallellized implementation resulted in a computation time of 4.5 days. During the online phase the performance index l_{te} of a superparameter configuration p is retrieved by means of the lookup table, significantly speeding up the experiment. In this way, the superparameter evaluations are recycled.

5-5-3 Search space cropping

Before performing the experiment, the search space $\mathbb{S}$ defined in 5-3-4 has been cropped to a smaller one. Search space cropping has reduced the computational complexity of the experiment by +/-1000 times[1], while improving relevance of the conclusions. This section explains why $\mathbb{S}$ has been cropped, how it has been cropped, and what the resulting $\mathbb{S}$ is, used in the experiment.

In AutoML research, there is no standard way of demarcating a relevant search space $\mathbb{S}$. If none of the tasks in the previous task set yield good performance on a part of $\mathbb{S}$, we refer to this part as an irrelevant part of $\mathbb{S}$. Including irrelevant parts in $\mathbb{S}$ imposes some problems:

- A naive algorithm will be outperformed by a badly warmstarted algorithm (e.g. cold-started), as illustrated in Figure 5-6. If a naive initialization $\mathbf{p}_0$ draws random configurations from a search space $\mathbb{S}$ including many irrelevant configurations, the approach would even be outperformed by the best configurations of the least similar previous task.
- A bigger search space $\mathbb{S}$ increases the computational complexity of an already expensive experiment.

Cropping the search space therefore makes the experiment more feasible, and makes its conclusions more relevant.

[1] assuming an equal discretization of the superparameters for the uncropped search space

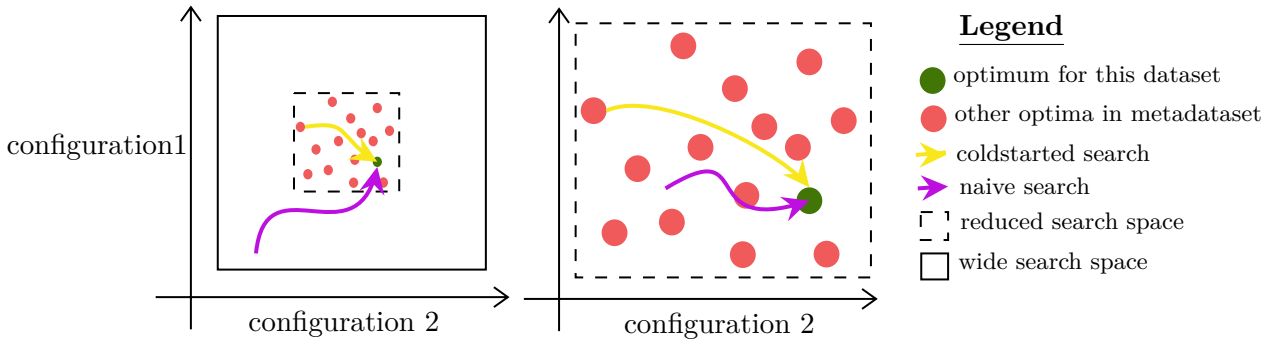


Figure 5-6: Intuïtive illustration of an uncropped (left) and cropped (right) search space $\mathbb{S}$, showing how a badly warmstarted superoptimization algorithm (e.g. coldstarted search) outperforms a naive algorithm, when $\mathbb{S}$ is not cropped. In an uncropped $\mathbb{S}$, the coldstarted search, starting at the optimum of the least similar task, has a shorter trajectory to the optimum than the naive search. The comparison is unfair without cropping $\mathbb{S}$.

The search space is cropped according to the following steps (read Appendix C for the exact process):

1. Exclude superparameter spans in $\mathbb{S}$ that (almost) never yield good performance.
2. Exclude superparameters in $\mathbb{S}$ with negligible sensitivity to the performance index l_{te}.
3. Define the search space discretization, such that it is feasible to perform the offline phase of the experiment, as explained in 5-5-2.

The resulting search space $\mathbb{S}$, used for the experiment is

$$
\mathbb{S} = \begin{cases}
p_{\mathbf{x}_T}: & \begin{cases}
\mathbf{y}: & \{\} \\
\mathbf{v}^1: & \{1\} \\
\text{hour of day}(\mathbf{y}): & \{1\} \\
\text{day of week}(\mathbf{y}): & \{1\} \\
\text{day of year}(\mathbf{y}): & \{1\} \\
\text{holiday}(\mathbf{y}): & \{1\}
\end{cases} \\
p_{tr}: & \text{all} \\
p_{\mathcal{H}}: & \text{DTS}: \begin{cases}
\mathcal{H}: & \begin{cases}
\text{num\_trees}: & \{\text{span}: [100, 800],\ \text{scale}: \text{lin},\ \text{values}: 6\} \\
\text{max\_depth}: & \{\text{span}: [5, 20],\ \text{scale}: \text{lin},\ \text{values}: 8\} \\
\text{min\_child\_weight}: & \{\text{span}: [5, 40],\ \text{scale}: \text{lin},\ \text{values}: 3\}
\end{cases} \\
\mathcal{L}_{tr}: & \text{regularized loss}: \begin{cases} \gamma: 0 \\ \alpha: 0 \end{cases} \\
\mathcal{A}_{tr}: & \text{Newton Boosting}: \begin{cases}
\text{learning\_rate}: \{\text{span}: [-2.5, -0.5], \text{scale}: \log, \text{values}: 10\} \\
\text{subsample}: \{\text{span}: [0.5, 1],\ \text{scale}: \text{lin},\ \text{values}: 3\} \\
\text{colsample\_by\_tree}: 1
\end{cases}
\end{cases}
\end{cases}
$$

5-5-4 Metalearning package

This research introduces a python package for experiments in warmstarting. The implementation provides the infrastructure of Figure 3-5, and is written according to Google's Python documentation style. A user can define their own experiment setting: an objective function, a set of tasks, and a search space. The user can propose their own metafeatures to construct warmstarted Bayesian optimization, and evaluate it together with the explained benchmarks, in a duplicated leave-one-out procedure. The package contains visualizers to compare the methods. The quickstart user example is attached as Appendix D.

Chapter 6

Results

The previous chapters introduced the proposed metafeatures for the warmstarting method (chapter 4), and the experiment (chapter 5) that aims to provide evidence for the hypothesis:

(Hypothesis) Runtime of superoptimizing hypothesis training superparameters reduces, if size and input richness are metafeatures in warmstarting.

The visualization of the results is introduced by showing the superoptimization performance on separate tasks in section 6-1. The heuristic method of setting the metaparameters of the metalearner, and their impact on superoptimization performance is visualized in section 6-2. The results of the central experiment of this thesis are then presented in section 6-3.

6-1 Performance on single tasks

To provide an intuition on how the method is compared to the benchmarks, Figure 6-1 visualizes the results comparing the methods on three separate forecasting tasks. The best Mean Absolute Error so far is shown for random search, naive Bayesian optimization and warmstarted Bayesian optimization. The methods are evaluated for 50 iterations and the results are averaged over five duplicate experiments. The metalearner uses the $r = 5$ best configurations, of the $k = 5$ most similar datasets and selects the $b = 5$ most occurring superparameter configurations.

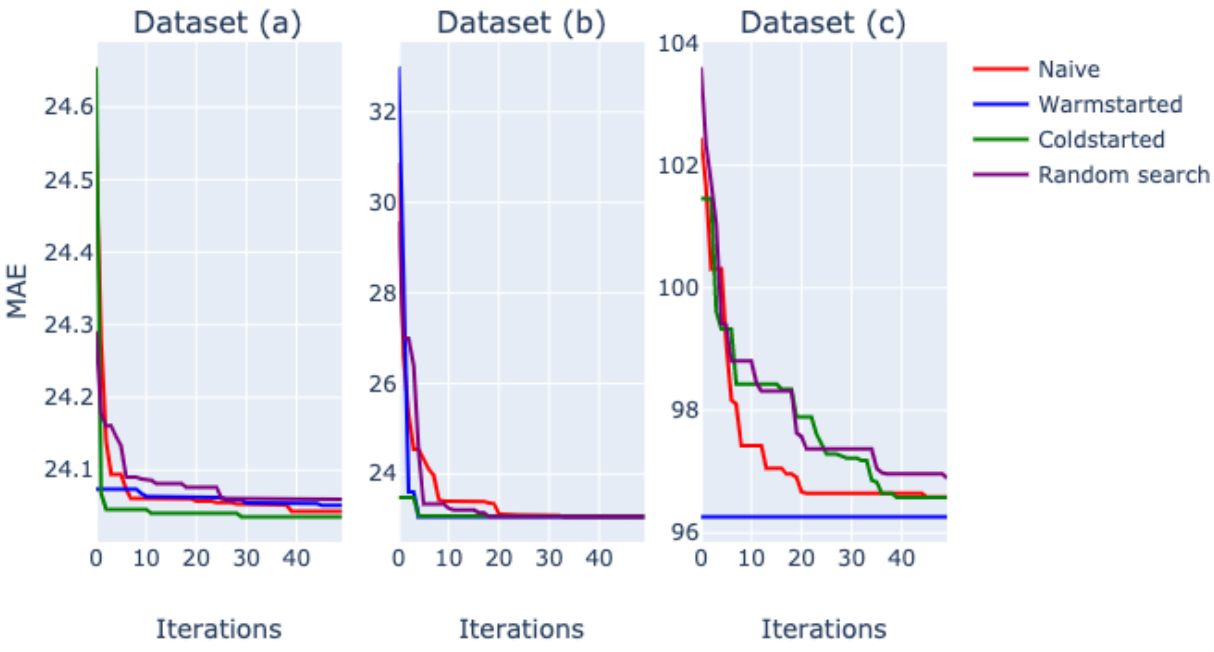


Figure 6-1: Comparison of the methods for three separate forecasting tasks. The best-so-far performance index l_{te} (in Mean Absolute Error) of the four methods is plotted per iteration. The metaparameters of the warmstarted and coldstarted method are $r = 5$, $k = 5$ and $b = 5$.

The visualized results only aim to provide insight on how the methods are compared in section 6-3, and do not meet criteria of statistical significance for conclusions on the hypothesis. It does draw attention to the fact that the relative performance of the methods is highly dependent on the task at hand.

6-2 Metaparameters

Before running the experiment, the following metaparameters of the metalearner have to be set::

- k: number of most similar samples selected.
- r: number of best performing superparameter configurations p selected per selected sample.
- b: number of most occurring superparameter configurations p selected, to constitute $\mathbf{p}_0$.

Figure 6-2 shows the performance of different settings of k and r, with a fixed $k = 5$. It is evaluated according the 32-fold leave-one-out procedure with 5 duplicates, as explained in

section 5-5-1. Using the result recycling procedure as defined in section 5-5-2 it took 6 hours to compute this on a 1,6 GHz Dual-Core Intel Core i5 processor.

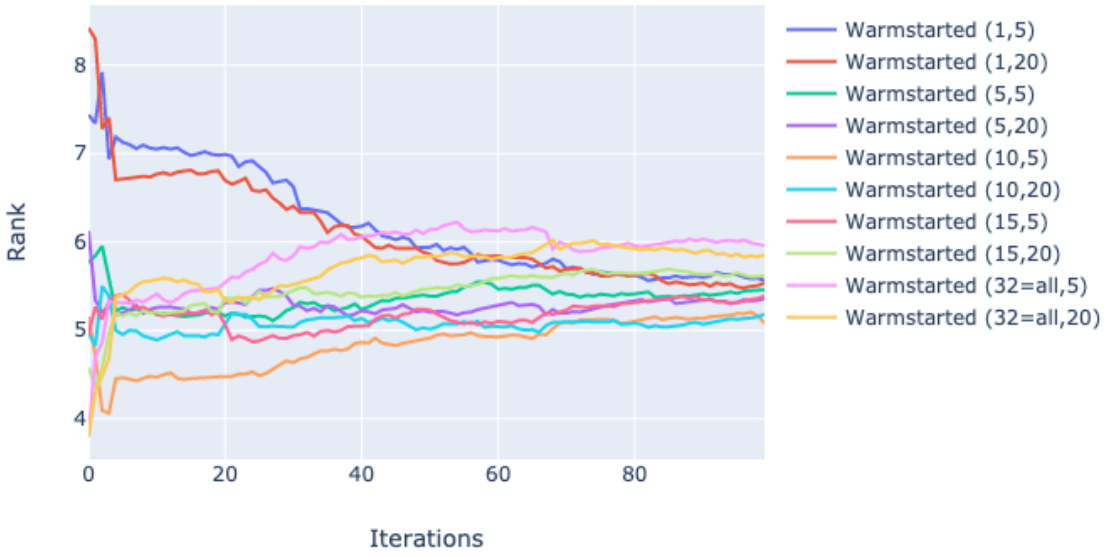


Figure 6-2: Superoptimization performance of the proposed warmstarting method on 32 tasks for different metaparameter settings, denoted as Warmstarted(k,r), using a 5 duplicated 32-fold leave-one-out procedure. The number of most similar samples selected is k, and the number of selected superparameter configurations per sample is r. The number of most occurring superparameter configurations, selected as warmstart, is fixed to $b = 5$. The procedure is explained in section 5-5-1.

The metaparameter settings (k,r) can roughly be divided into three groups, based on performance:

1. good performance: (10,5)
2. average performance: (5,5), (5,20), (10,20), (15,5), (15,20), (32,5), (32,20)
3. bad performance: (10,5)

Warmstarter$(10, 5)$ will be referred to as the well tuned warmstart, and Warmstarter$(5, 5)$ will be referred to as the badly tuned warmstart.

6-3 Experiment results

The results of the experiment are visualized in this section.

6-3-1 Performance versus iterations

The performance of the well-tuned warmstart is compared in Figure 6-3 against the benchmarks, explained in section 5-1. A 32-fold leave-one-out result recycling procedure with ten duplicates (as explained in sections 5-5-1 and 5-5-2) is performed in three hours on a 1,6 GHz i5 processor.

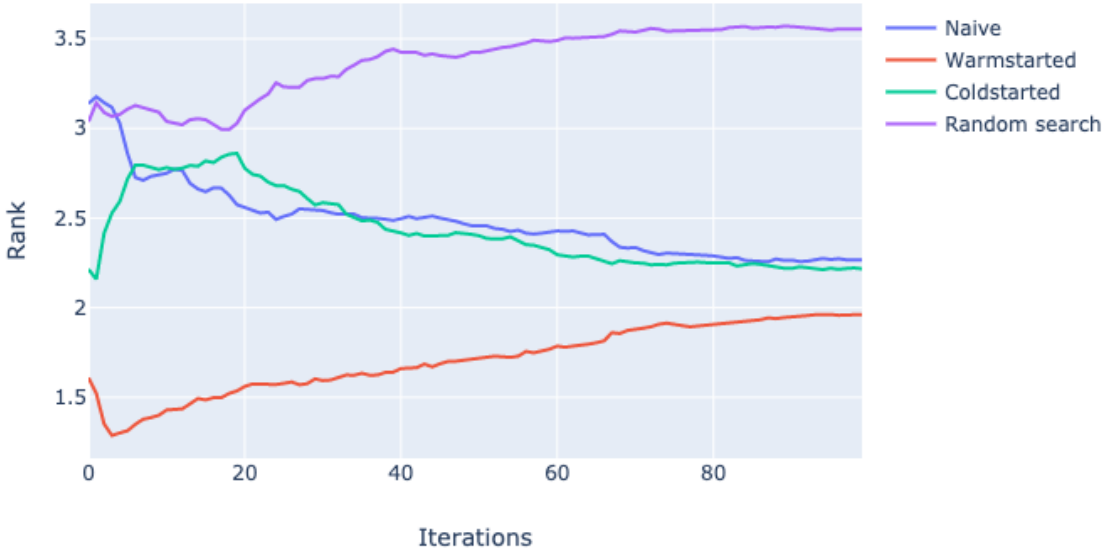


Figure 6-3: The performance of the well-tuned proposed warmstart, compared with the benchmarks. The methods perform superoptimization for a forecasting task at hand. At every iteration they are ranked according to their best-so-far performance index. A lower rank means better performance. The results are averaged over all forecasting tasks and 10 duplicates.

The main observations are that:

Observation (1): The proposed warmstart method outperforms the coldstart benchmark.

Observation (2): The coldstart benchmark outperforms the naive benchmark during the first iterations.

6-3-2 Runtime

Figure 6-4 shows the number of iterations it takes for the compared methods to achieve the same best-so-far performance as random start's 100^{th} iteration.

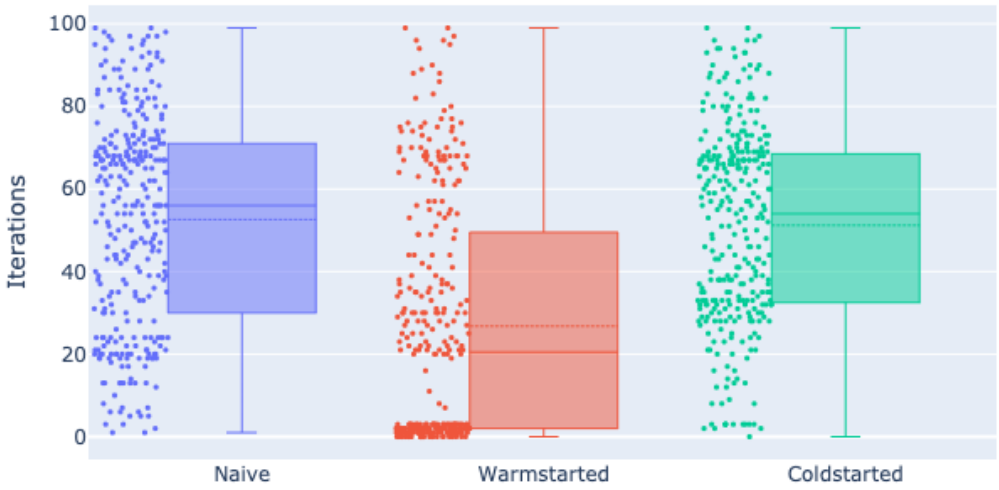


Figure 6-4: The number of iterations until the same performance is reached as random search in the 100th iteration.

In comparison to random search, warmstarted search decreases the number of search iterations by 65% on average. This is an improvement of 50% compared to naïve search. Coldstarted search performs approximately as good as random search from this perspective. Also note the gap after the initialization batch of five iterations of the warm- and coldstart.

Chapter 7

Discussion

The results of the experiment are visualized in the previous section. Given

Observation (1): The proposed warmstart method outperforms the coldstart benchmark,

this section discusses whether the hypothesis

(Hypothesis) runtime of superoptimizing hypothesis training superparameters reduces, if size and input richness are metafeatures in warmstarting,

can be confirmed. In this consideration, the relevance of proposing the coldstart benchmark that led to

Observation (2): The coldstart benchmark outperforms the naive benchmark during the first iterations,

is discussed. The observations are discussed for conclusions for inside the experiment setting, and outside of the setting.

7-1 Inside the experiment setting

Firstly the underlying assumptions for the hypothesis are tested visually. Then the implications of tuning the metaparameters are discussed, and the individial contribution of the two metafeatures.

7-1-1 Underlying assumptions of hypothesis

The underlying assumptions of the hypothesis, explained in section 4-3, are that

- the metafeature size correlates with well-performing regions of complexity superparameters,

- the metafeature input richness correlates with well-performing regions of overfitting superparameters.

In 7-2 the metafeatures of all the task set are plotted against their best-found superparameters, to visually inspect whether the assumed correlations are present.

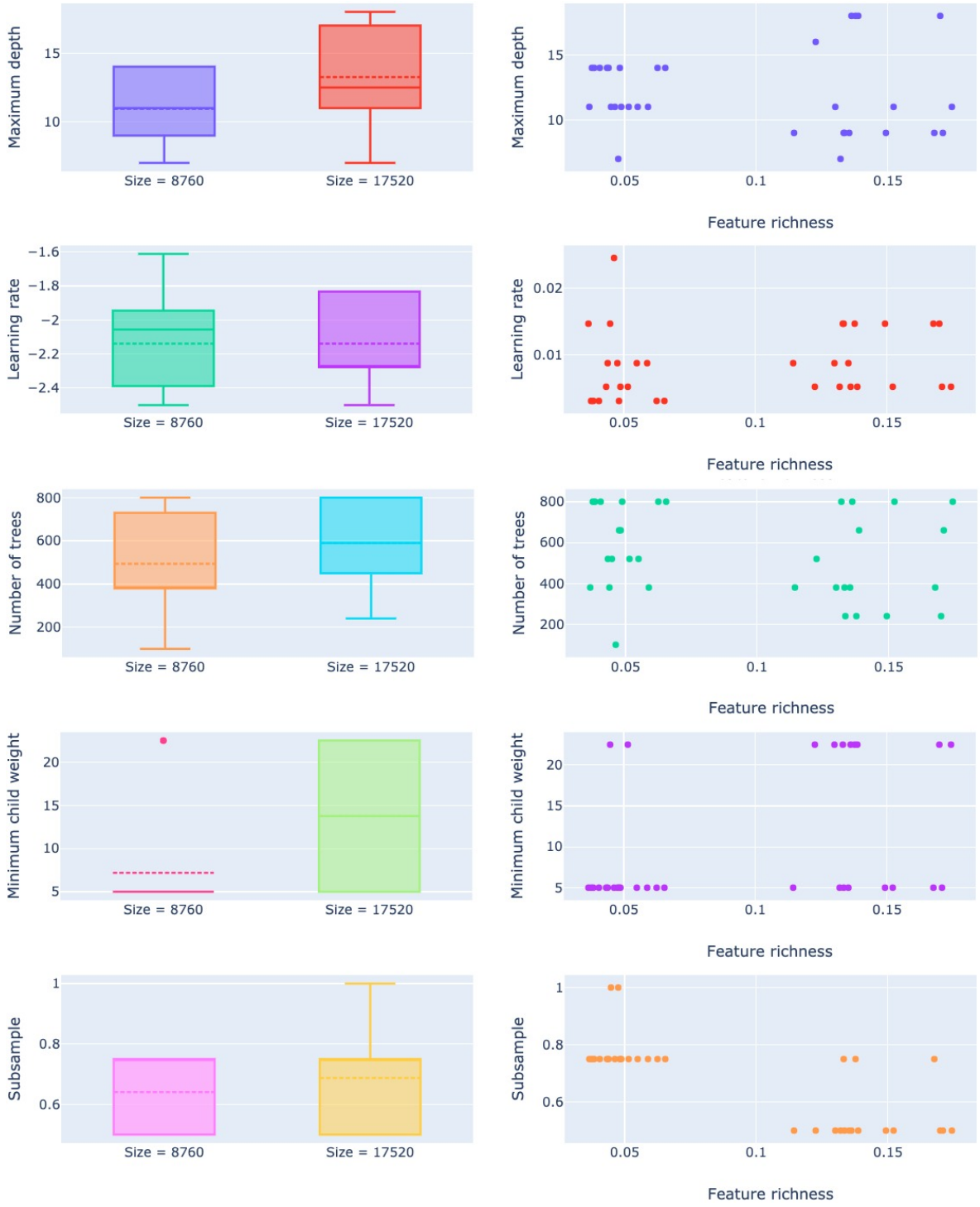


Figure 7-1: The best-found configurations per superparameter, plotted against metafeatures for all 32 tasks. The data comes from the offline phase of the experiment, as explained in section 5-5-2.

Figure 7-2

A couple of assumptions are confirmed, as a higher dataset size correlates with a higher best value for maximum depth and number of trees, as expected. On the other hand, a lower feature richness unexpectedly correlates to a smaller best-found minimum child weight and higher subsample. The expected versus the actual correlation is summarized in table 7-1.

Table 7-1: Expected versus actual relation between good superparameters and metafeatures, concluded visually from Figure 7-2.

Superparameter	**Expected if size is higher**	**Actual relation in experiment**	**Confirmed?**
max depth	higher	higher	yes
learning rate	lower	-	no
number of trees	higher	higher	yes

Superparameter	**Expected if feature richness is higher**	**Actual relation in experiment**	**Confirmed?**
max depth	lower	higher	no
min child weight	lower	-	no
subsample	higher	lower	no

Two of the six assumptions are confirmed, so the evidence of observation (1) is not given weight, by this visual inspection. Visual inspection has limitations, as the eye only sees monotonic and linear correlation, no nonlinear and interdependent correlations. Observation (1) could be attributed to interdependent relations between the well performing superparameter settings, leveraged by the warmstart.

7-1-2 Implications of tuning the metaparameters

The metaparameters of the warmstart have only been tuned slightly. Careful assessment of overfitting the task set is still necessary, as the previous task set is very small, similar to other researches in Automated Machine Learning (AutoML)[11]. For this end, the performance of the badly tuned warmstart, is visualized in Figure 7-3.

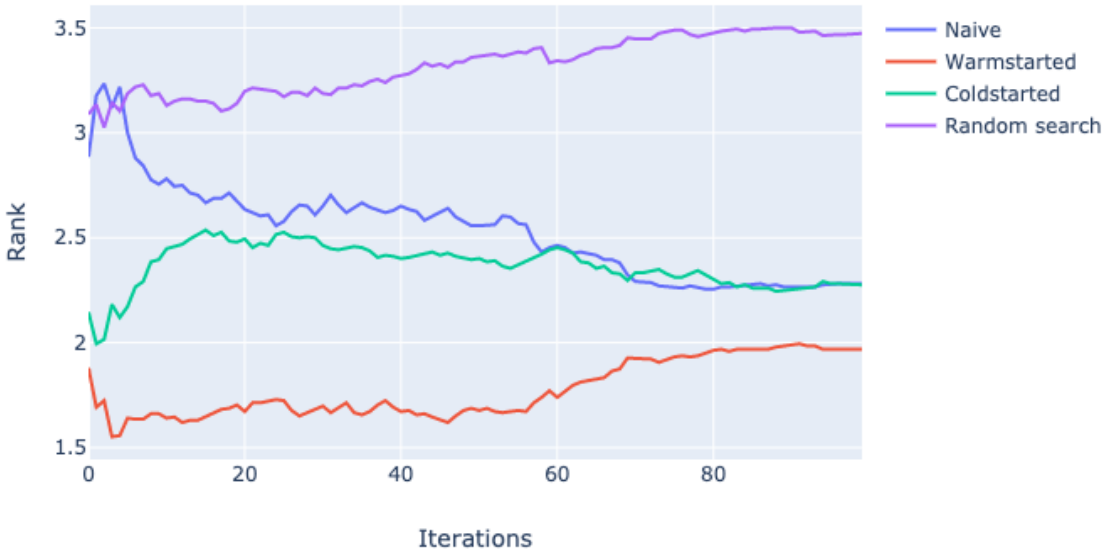


Figure 7-3: The performance of the badly tuned proposed warmstart, compared with the benchmarks. The methods perform superoptimization for a forecasting task at hand. At every iteration they are ranked according to their best-so-far performance index. A lower rank means better performance. The results are averaged over all forecasting tasks and 3 duplicates.

Even if the warmstart method is badly tuned, it outperforms the coldstart benchmark. Appendix E shows that this is consistent with the results for the metaparameter settings with average performance. The relevance of observation (1) is given extra weight. Furthermore, the coldstart benchmark outperforms the naïve benchmark in the first iterations. This gives extra weight to observation (2). Even the worst warmstarting method can outperform the naive benchmark, as explained in section 5-1. It therefore proves that a comparison with a naïve method is insufficient for making the conclusion that a warmstart is learning from previous tasks. This research proposes coldstart benchmarking to fill this gap.

7-1-3 Separate metafeatures

A separate assessment of the two metafeatures is needed, to determine whether they both have a contribution in warmstarting. Figure 7-4 visualizes the results of using the separate metafeatres, using the badly tuned warmstart.

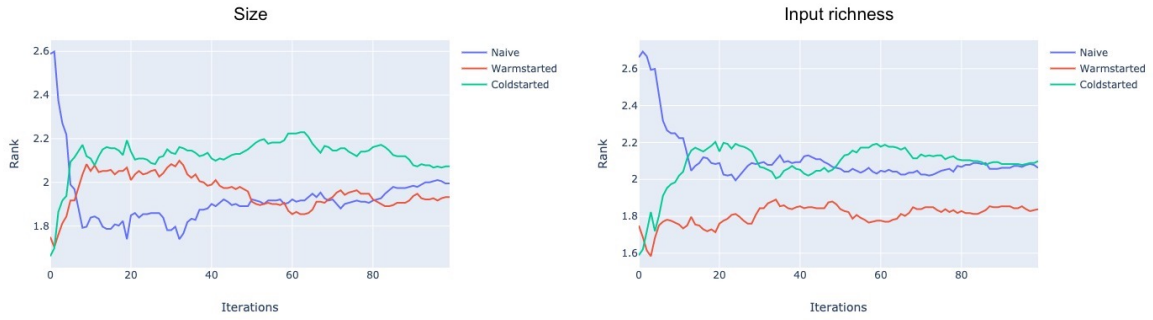


Figure 7-4: The performance of the badly tuned proposed warmstart, for the individual metafeatures, compared with the benchmarks. The methods perform superoptimization for a forecasting task at hand. At every iteration they are ranked according to their best-so-far performance index. A lower rank means better performance. The results are averaged over all forecasting tasks and 3 duplicates.

The warmstart outperforms the coldstart benchmark for both metafeatures. For input richness, the warmstart outperforms the benchmark to a higher degree than for size. The results show that both metafeatures contribute to the performance improvement of observation (1). In the experiment with size as metafeature, it is notable that the naïve benchmark outperforms both methods between iteration 5 and 50.

7-2 Outside the experiment

If the hypothesis is confirmed in the experiment setting, does the conclusion hold outside of the experiment setting? For this question, the quality of the hypothesis and the experiment setting is discussed.

7-2-1 Hypothesis quality

As mentioned, the underlying assumptions of the hypothesis are that:

- the metafeature size correlates with well-performing regions of complexity superparameters,
- the metafeature input richness correlates with well-performing regions of overfitting superparameters.

The overlap in the superparameter categorization is discussed firstly, then the metafeature input richness is discussed, and finally the metafeature size is discussed.

Complexity superparameters determine the number of parameters in the hypothesis class to be optimized. Some examples are the number of trees in XGBoost, or the number of hidden layers in Neural Networks. Overfitting superparameters determine the degree to which a

Machine Learning (ML) model applies protective measures against modeling the noise in a training set. Some examples are subsample in XGBoost, and the regularization rate in Neural Networks. Although existing ML models have superparameters for either complexity or overfitting, there is an overlap between the two. The similarity is that they both affect the number of parameters of a trained model, but they do it in a different way. It is hard to define metafeatures that capture this nuanced different way of affecting the number of parameters. It is therefore arguable that size and input richness are too limited to capture this difference.

The proposed metafeature feature richness is an estimation of the amount of noise in the dataset. Noise is the term for the fluctuations in the input data of the training set, that do not carry useful information for the output. A fluctuation carries useful information, if the assumed hypothesis class fits the type of correlation between the input and output. The type of correlations that fit the ML models of interest (for objective 1) are specific complex nonlinearities. Feature richness only estimates simple nonlinearities (Spearman's correlation), and is therefore a very limited estimation of the noise. Because of this limitation the metafeature might not generalize outside of the experiment setting.

The proposed metafeature size is the number of samples in the training set. In the experiment, the samples have the same length of the input vector. Outside of the experiment, input vectors can have a different length. This could affect what the good regions of the complexity superparameters, as more data points means that more parameters can be optimized. The length of the input vector as metafeature is not investigated in this experiment.

7-2-2 Experiment setting

The construction of an experiment setting is discussed in this section. Firstly the task set is discussed, and then the search space.

Due to computational constraints, the experiment setting includes only few forecasting tasks. A task set of 32 tasks is used, which can be divided into four groups of similar size and feature richness. It is applicable to the warmstart's hypothesis. Selecting the most similar dataset is made easy, and the focus is drawn to the superoptimization performance of the method. The experiment setting is similar to the real world, to the degree that the tasks are datasets in financial trading. The limitation is that in the real world, forecasting tasks are not grouped, but their size and input richness varies largely on a continuous scale. The forecasting tasks also vary on other characteristics, excluded in this experiment. It is interesting to test whether the hypothesis holds in the real world setting.

The search space was limited to the hypothesis training superparameters of XGBoost. More valuable conclusions on the hypothesis could have been applied, if a ML model was chosen that has less overlapping superparameters for complexity and overfitting. Generalizability of the results to other search spaces is substantiable, because many other ML models have superparameters that set complexity and overfitting.

Chapter 8

Conclusion

Exploring hybrid Machine Learning (ML) forecasting models is highly promising for improving forecasting performance in numerous applications. A definition of the hybrid model space, with intrinsically incorporated model components from different paradigms, was missing. As the hybrid model space is very big, the job of exploring its full potential requires too much effort for a data scientist. Replacing the human that performs this job with a computer is a promising approach, especially given the recent developments in Automated Machine Learning (AutoML) research. The approach is even more relevant, because reducing the costs of the job by replacing expensive data scientists, makes the prospective technology accessible to smaller and less economically driven organizations. This research pursuited the ideological goal

to democratize ML for any forecasting task, and converge models across paradigms, by replacing expensive data scientists with a computer, that optimizes over cross-paradigm forecasting model components.

The first objective of this research is reached by *contribution (1): superparameter evaluation, a framework for parametrizing the creation and evaluation process of a hybrid forecasting model.* On top of the examples in the SARIMAX, Neural Networks and Decision Tree model domain, it can merge any intrinsic component into the hybrid model, if it is rewritable to the defined format. The complete list of instructions for creating a hybrid model in parameter format is a superparameter configuration. Given a task at hand, a human can input a superparameter configuration to superparameter evaluation, which returns a corresponding performance index.

Then, this research incorporated existing AutoML literature, to replace the human job by a computational job, to reach the second research objective. The computational job is referred to superoptimization, which treats superparameter evaluation an objective function to be minimized. Given a new unseen forecasting task, the warmstart method chooses the most similar tasks, measured using metafeatures, from a set of previous tasks. The warmstart's metalearner then suggests the best performing superparameter configurations of the most

similar tasks, to be firstly evaluated. After this, a black-box optimization algorithm proceeds optimizing superparameter evaluation, by learning from previous evaluations on the task at hand.

Even for existing algorithms in AutoML, the set of superparameter configurations to consider (the search space), characterizes as too large and complex. The third research objective was to reduce the runtime of the computational job, by learning from previous tasks. Previous work that introduced the warmstart method[11] was analyzed critically. It was argumented that the small size of the previous task set, in combination with the unexplained method of defining a big set of metafeatures, raises the concern of overfitting the previous task set. The benchmarking method was also questioned. In order for the warmstarting method to be more than a function, inapplicable to the real world, it needs evidence that there are metafeatures that make the metalearner learn from previous tasks. This research proposed two simple metafeatures for the warmstart method, size and input richness, and aimed to contribute evidence for the hypothesis that:

(Hypothesis) Runtime of superoptimizing hypothesis training superparameters[1] reduces, when size and input richness are metafeatures in warmstarting.

The underlying assumptions are that the size of a training set correlate with well-performing settings of superparameters related to model complexity, and that input richness correlates with well-performing settings of superparameters related to measures against overfitting. The hypothesis was tested in an experiment setting of 32 forecasting tasks. As the tasks have either a high or low value for size and input richness, the tasks are divided into 4 groups of 8 tasks. The search space of the experiment consists of the hypothesis training superparameters[1] of XGBoost.

The first conclusion from the results applies to the method of benchmarking: *Contribution (2): the knowledge that no conclusion can be made on whether the warmstart is learning from previous tasks, if it is compared with the naïve benchmark. The coldstart benchmark is proposed as the alternative benchmark.* The proof by falsification is as follows: if the warmstart method outperforms the coldstart method, this is evidence that the metalearner is learning to suggest well-performing superparameter configurations from the previous task set. It is also evidence that the reversed method, the coldstart method, is learning to suggest badly-performing superparameter configurations from the previous task set. Results show that the coldstart method can outperform the naïve method, while the warmstart outperforms the coldstart (*Observation (2)*). Any 'warmstart' method proposed in literature, only shown to outperform the naïve benchmark, could therefore be learning to suggest badly-performing superparameters.

The experiment results showed that a reduction of 50% of computational resources is achieved with respect to the coldstart and naïve benchmark. Outperforming the coldstart method maintains, even if the warmstart is badly tuned. Furthermore, the results showed that both metafeatures can contribute separately to the performance improvement of superoptimization. These observations led to *contribution (3): evidence for the reduction of runtime of superoptimizing hypothesis training superparameters, when size and input richness are metafeatures in warmstarting* The evidence is mild, because the previous task set in the experiment was very small. This research does classify the positive results as evidence, instead of positive results due to overfitting the task set, because:

[1] the hypothesis training superparameters correspond with hyperparameters in the supervised ML framework

- the proposed metafeatures were synthesized by using logic,
- the chances of overfitting were reduced as much as possible, by using a simple warmstart method, and testing the results on the badly tuned warmstart,
- reliable benchmarking, due to contribution (2).

This research finally provides *contribution (4): the Python package warmstart*. The package provides the implementation for future research on metafeatures for the warmstart method.

To conclude, the ideological goal is to democratize promising hybrid ML forecasting models. The approach is to replace the data scientist with a computer, that optimizes over cross-paradigm forecasting model components. This research contributed a framework for parametrizing the creation and evaluation process of a hybrid forecasting model. Because the set of hybrid forecasting models is very large, the human job of selecting one, is replaced with an optimization algorithm, by incorporating the AutoML framework. Acknowledging that the optimization also requires too much runtime from a computer, this research has focused on reducing the runtime of the warmstart method. The usefulness of the existing benchmark was falsified, and the coldstart benchmark was proposed as an alternative. The research then proposed metafeatures for hypothesis training superparameters, which corresponds with hyperparameters in the ML framework. The experiment focused on preventing overfitting on the task set, which led to the confirmation of the hypothesis, that the method improves superoptimization performance.

8-1 Future work

This section suggests potential future work, building on the outcomes of this research. It is divided into the future work related to superparameter evaluation, superoptimization, and warmstarting, and applies to both business and research. The suggestions related to superparameter evaluation are:

- Expand the framework to fit models that contain unsupervised temporal layers, such as RNN, DeepAR, and WaveNet.
- Use the framework to more easily design new hybrid models for a given forecasting task.
- For forecasting businesses: use superparameter evaluation as implementation blueprint, to incrementally add model components for performance improvements for customers.

A couple of suggestions apply to the superoptimization job:

- An example superoptimization algorithm specifically for superoptimization: compose a search space with SARIMA in the feature engineering superparameters. Include linear regression, XGBoost and Neural Networks in the hypothesis training superparameters. Train a model that predicts the computational runtime, given a superparameter configuration. Apply Bayesian optimization to the search space, but with a weighted penalty of the predicted computational runtime. During superoptimization, slowly reduce the

weight of this penalty. The assumption is that in this method, Bayesian optimization will use the cheap evaluations to find good feature engineering superparameter configurations, and then after that the other superparameters are set.

- Experiment with a warmstarted Bayesian prior, instead of a warmstarted set of initial configurations.
- For a forecasting business with many customer in the same forecasting task domain: Define a small hybrid search space and perform superoptimization to automatically find a hybrid model for production.
- combine warmstart with a black-box optimization algorithm, that is parallellizable.
- a different approach to the computational complexity, is to implement superoptimization as a quantum optimization.
- in the context of superoptimization, the prevention of overfitting gets more and more important. Investigate

Finally, the suggestions related to warmstarting are:

- find more metafeatures, applicable to the hypothesis training superparameters. An example is the length of the input vector. Different ways of measuring feature richness would also be an interesting approach, to make the metalearner learn the nuanced difference between setting the complexity and overfitting superparemeters.
- collect better evidence for this research's hypothesis, by composing a very big set of tasks with more variation in size and input richness, and testing the proposed warmstart in that setting.
- find metafeatures for the other components of superparameter evaluation, feature engineering and splitting.

Appendix A

XGBoost

XGBoost is a regression model from the machine learning community, using decision trees, a simple model of logical operators (splits), splitting input data into buckets. The predicted value is the leaf weight. An example decision tree for grocery apple sales would be as follows: if it is not weekend and the predicted hour is after 5 pm, the predicted amount of sold apples is 25 during that hour.

A popular method to boost performance of simple models - also referred to as weak learners f - is ensemble learning. Using multiple - K - weak learners for a single prediction. The random forest model is an example where the predicted value is the mean of the decision tree predictions. XGBoost uses another type of ensemble learning, namely adaptive boosting,

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}, \tag{A-1}$$

where the predictions of the weak learners are summated. Every weak learner - in this case tree - is sequentially trained on the residuals of its previous weak learners. First the global process of building trees is explained, then we will zoom in on how splits are made.

The algorithm starts by predicting the mean, after which the tree building process starts to create trees that explain deviations from the mean. A tree is built by greedily making the best splits, according to an objective function, which is explained later. The model stops splitting when one of the stopping criteria is met, defined by the manually determined hyperparameters. Hyperparameter **max depth** determines the maximum amount of logical operators before a prediction. Hyperparameter **min child weight** is an integer value that forces splitting to stop, at a minimum amount of samples in the training set that reaches the leaf. Figure A-1 provides an example, where in tree 1 at data samples I_2, I_5, the tree stops splitting because of reaching the minimum child weight. In tree 2 at data samples I_3, I_5, I_6, the tree stops splitting because of reaching the maximum depth.

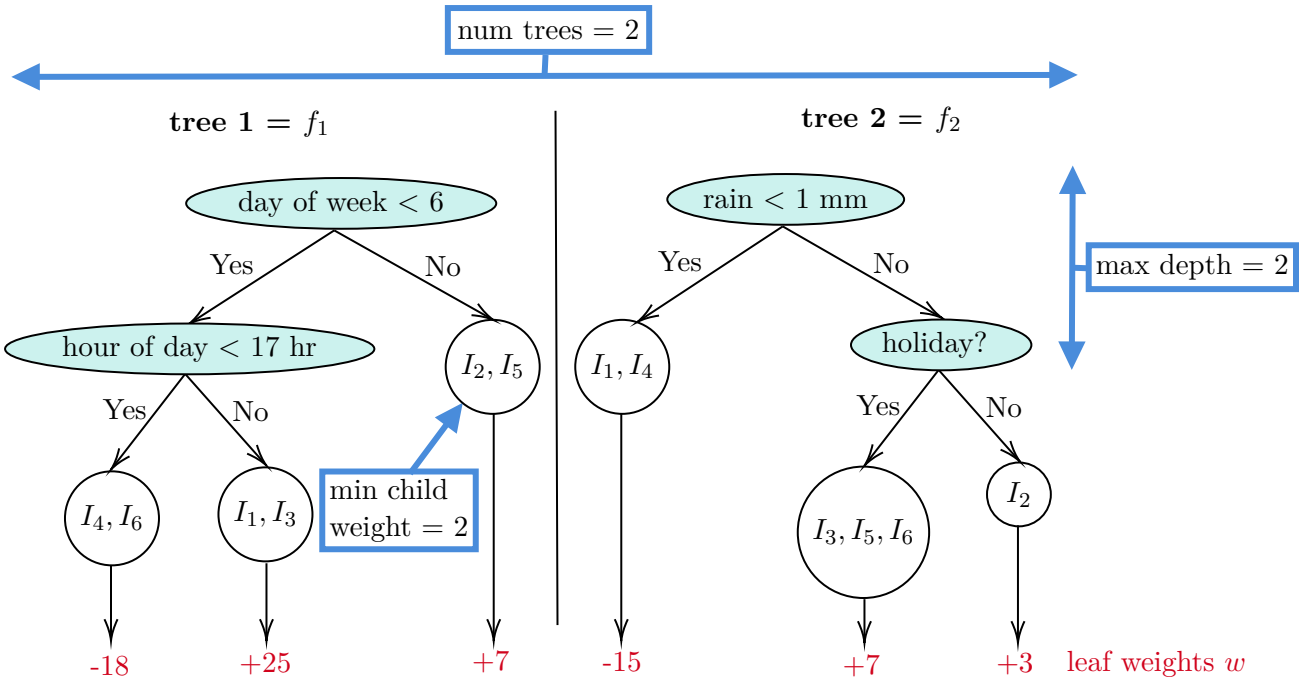


Figure A-1: XGBoost example

After making the splits, the pruning process iteratively erases leafs, whose splits contribute negatively to the objective function.

When a tree is built and pruned, the leaf weight predictions are multiplied by the hyperparameter **learning rate**, a real valued number between zero and one. This shrinkage gives room to future trees to impact the total prediction and slows down the learning process. The next tree(s) are iteratively built using the residuals of its previous tree(s), until the maximum number of trees is reached, manually defined by the hyperparameter **num trees**.

Splits are made by using the following regularized objective function,

$$\begin{aligned} \mathcal{L} &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \text{where} \quad \Omega(f) &= \gamma T + \frac{1}{2}\alpha||w||^2, \end{aligned} \tag{A-2}$$

to find the best feature and best value to split on. As loss function $l(\hat{y}_i, y_i)$ we use squared error $(\hat{y}_i - y_i)^2$. The regularization loss Ω is calculated from the tree structure of the weak learners f. The loss consists of a penalty for the number of leafs T scaled by hyperparameter gamma γ and a penalty for the magnitude of the leaf weights w, scaled by hyperparameter α. Overfitting is furthermore counteracted by hyperparameters **subsample** and **colsample by tree**, scalar values between zero and one, which randomly reduce the instances and features respectively per tree. The optimization algorithm uses Newton boosting to evaluate all possible splits over all possible features to select the best split, as explained in the next section. The XGBoost implementation introduces a cache aware parallellized method for split finding, that leverages all the computational resources to minimize computation time.

A-1 Newton Boosting

XGBoost uses adaptive boosting,

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F}, \tag{A-3}$$

where every consecutive weak learner f predicts the residuals of its former weak learners. The number of weak learners is denoted as K. The set of weak learners $\mathcal{F}$ included are decision trees,

$$\mathcal{F} = \{f(x) = w_{q(x)}\}(q : \mathbb{R} \mapsto T, w \in \mathbb{R}^T), \tag{A-4}$$

where q is the function of operators applied to the features of a data sample x, leading to a prediction, defined as the leaf weight w. Splits in the trees are constructed greedily according to the regularized objective function,

$$\begin{aligned} \mathcal{L} &= \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k) \\ \text{where} \quad \Omega(f) &= \gamma T + \frac{1}{2}\alpha||w||^2, \end{aligned} \tag{A-5}$$

in which the loss function $l(\hat{y}_i, y_i)$ penalizes error between prediction and actuals. The regularization loss Ω penalizes model complexity and thus reduces overfitting, penalizing the amount of leaves, weighted by hyperparameter γ, and penalizing the magnitude of the weights, weighted by hyperparameter α. For regression often the squared error,

$$l(\hat{y}_i, y_i) = (y_i - \hat{y}_i)^2 \tag{A-6}$$

is used as loss function. As multiple trees are consecutively built, we have an objective function specific for every tree,

$$\mathcal{L}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t). \tag{A-7}$$

Using the second-order Taylor expansion,

$$f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2, \tag{A-8}$$

we can rewrite (A-7) to a Taylor approximation,

$$\begin{aligned} \mathcal{L}^{(t)} &\simeq \sum_{i=1}^{n}[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ \text{where} \quad g_i &= \delta_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}) \quad \text{and} \quad h_i = \delta^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \end{aligned} \tag{A-9}$$

and when filling in the squared error as objective function we find that $g_i = 2(y_i - \hat{y}_i^{t-1})$ and $h_i = -2$. When we remove the constant terms in (A-9),

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t), \tag{A-10}$$

and expand Ω as defined in (A-5) and rewrite the equation to have the leaves in the outer loop of the summation

$$\begin{aligned}\mathcal{L}^{(t)} &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\alpha\sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \alpha)w_j^2] + \gamma T,\end{aligned} \tag{A-11}$$

we derive an analytical expression for the optimal weight of a leaf,

$$w_j^* = -\frac{\sum_{i\in I_j} g_i}{\sum_{i\in I_j} h_i + \alpha} \tag{A-12}$$

in the case of squared error objective function this optimal weight is

$$w_j^* = \frac{\sum_{i\in I_j}(y_i - \hat{y}_i^{t-1})}{\sum_{i\in I_j}(1 + \frac{1}{2}\alpha)}. \tag{A-13}$$

The objective function for optimally chosen weights,

$$\mathcal{L}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \alpha} + \gamma T \tag{A-14}$$

is used to choose the best split. During tree pruning the gain of a split,

$$\text{Gain} = \frac{1}{2}\left[\frac{(\sum_{i\in I_{j,L}} g_i)^2}{\sum_{i\in I_{j,L}} h_i + \alpha} + \frac{(\sum_{i\in I_{j,R}} g_i)^2}{\sum_{i\in I_{j,R}} h_i + \alpha} - \frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \alpha}\right] - \gamma \tag{A-15}$$

is computed to derive whether a split has positive effect on the regularized function.

Appendix B

Tree Parzen Estimator

Bayesian optimization makes a model $p(l_{test}|p)$, using priors, the scale of the hyperparameter spaces. Every iteration k it performs Bayesian posterior updating using evaluation history

$$\{p_j \cup l_{test,j}\}, \quad j = 0, 1, ..., k, \tag{B-1}$$

and uses an acquisition function to suggest a promising p_{k+1}. Tree Parzen Estimator (TPE) is a Bayesian optimization algorithm that models $Pr(l_{te}|p)$ indirectly by modeling $Pr(p|l_{te})$ and $Pr(l_{te})$. $Pr(p|l_{te})$ is modeled separately per dimension in the search space as

$$Pr(p_1|l_{te}) = \begin{cases} l(p_1), & \text{if } l_{te} < l^*_{te} \\ g(p_1), & \text{if } l_{te} \geq l^*_{te}, \end{cases} \tag{B-2}$$

where p_1 is the configuration in a single dimension in $\mathbb{S}$. The previous evaluations are thus divided into a set of best p_1's and a set of poor p_1's. Both sets are fitted to a Gaussian Mixture Model, where every evaluation represents the mean of a Gaussian that has a variance equal to the greater of the distance between the left or right neighboring sample. The parameters of the Gaussian Mixture Models are fitted through maximum likelihood estimation. Hyperopt defines the threshold l^*_{te} as some heuristic quantile. The acquisition function is the Expected Improvement,

$$\mathrm{EI}_{l^*_{te}}(p_1) := \int_{-\infty}^{\infty} \max(l^*_{te} - l_{te}, 0) p_M(l_{te}|p_1) dl_{te} \tag{B-3}$$

which, according to [36], can be rewritten to the maximization objective,

$$\max_{p_1} \frac{l(p_1)}{g(p_1)}, \tag{B-4}$$

and the candidates for this maximization are samples drawn from $l(p_1)$. Figure B-1 provides an intuitive example, showing that this procedure suggests $p_1 \in \mathbb{S}$ of which there is a high expectancy of being in the set of good p_1's, while having a low expectancy of being in the set of poor p_1's.

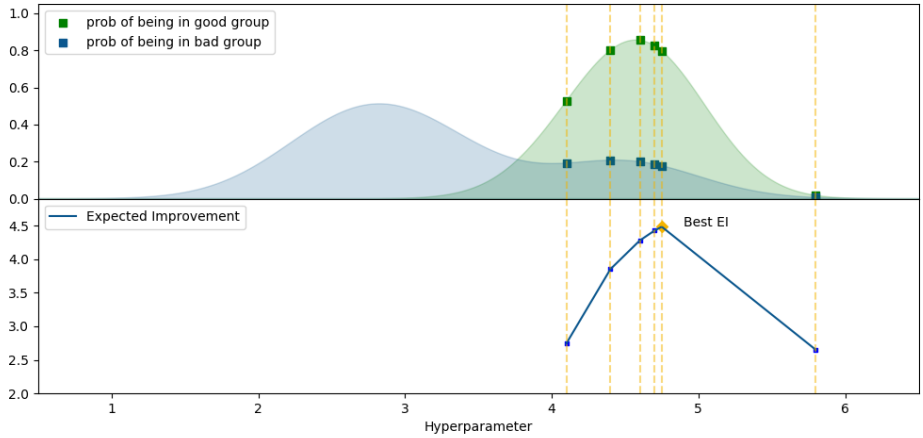


Figure B-1: Calculation of a candidate with best Expected Improvement

Appendix C

Search space cropping

We start out with the broad search space of Table C-1. The following steps will narrow it this space to a relevant and computationally viable search space.

Table C-1: Initial search space

Superparameter	**Span**	**Scale**
num trees	1 - 800	linear
learning rate	(-6) - 0	log
max depth	1 - 40	linear
min child weight	1 - 30	linear
gamma	0 - 1	log
subsample	0.5 - 1	linear
colsample by tree	0.5 - 1	linear
reg alpha	(-4) - 2	log

The first step is to find the tightest relevant superparameter spans. A random search of 500 iterations over a randomly drawn quarter of the 32 datasets is performed. Figure C-1 shows the distribution of the 5 best superparameters on each task. Spans that are never optimal are excluded from the search space.

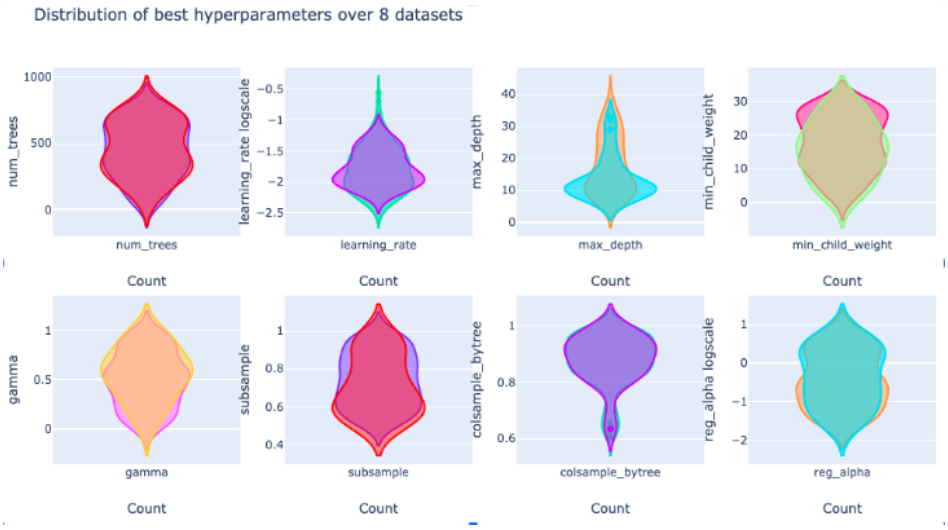


Figure C-1: Superparameter optimality (5 best iterations or 20 best iterations

The next step in cropping the search space is to exclude superparameters that have negligible effect on the forecasting model performance. A one dimensional grid search is performed for the superparameter of interest around the optimal superparameter setting from the previous experiment. In Figure C-2 the ratio of mean absolute errors with respect to the optimal p's mean absolute errors are plotted and we define the ratio as superparameter sensitivity. This is repeated for 8 randomly drawn datasets. Gamma and colsample by tree turn out to have no significant impact on performance and are therefore excluded from the search space.

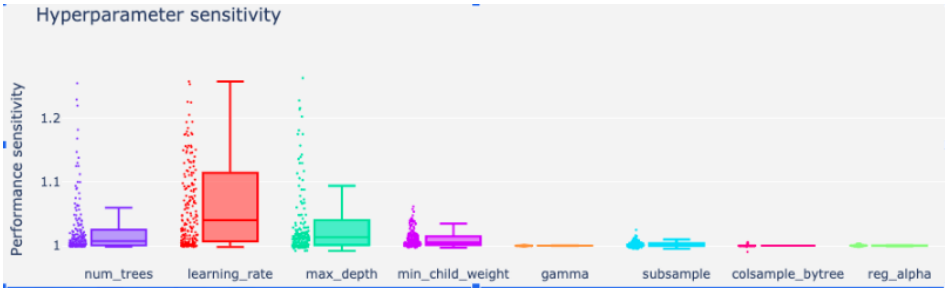


Figure C-2: Superparameter sensitivity, update since the scale are not nice

In order to apply result recycling, as explained in section 5-5-2, the search space is discretized to make it suitable for a grid search. We define a higher granularity for superparameters with higher sensitivity to performance. Using a computational complexity estimator, the granularity per superparameter is defined, resulting in the cropped search space of Table C-2.

Table C-2: Cropped search space

Superparameter	**Span**	**Scale**	**Values**
num trees	100 - 800	linear	6
max depth	5 - 20	linear	8
learning rate	(-2.5) - (-0.5)	log	10
subsample	0.5 - 1	linear	3
min child weight	5 - 40	linear	3

Appendix D

Metalearning Python package

This research introduces a python package for metalearning. It performs experiments comparing search strategies such as warmstarted Bayesian optimization, following the formalization of Figure 3-5. We provide a quick-start user example of the experiment of this specific research.

Environment setup

Start by cloning the Github repository (https://github.com/JeroenSwart/autoxgb) and installing Jupyter Lab, an extensible environment for interactive and reproducible computing. The README file gives a walk-through for setting up the virtual environment and creating an ipython kernel, from which to launch a Jupyter Lab notebook. Firstly the classes are imported from the library.

```
# Import external libraries
import pickle

# Import internal metalearning libraries
from src.experimenting.hopt_experiment import HoptExperiment
from src.metalearning.metadata import MetaDataset
from src.metalearning.warmstarter import Warmstarter
from src.pipeline_optimization.bayesian_hopt import BayesianHopt
from src.utils.metafeature_utils import size, cumac

# Import thesis specific objective and search space
from src.utils.thesis_utils import thesis_lookup_objective,
                                   thesis_search_space
```

Definitions

A note on the definitions used in the code:

- MetaSample = task
- MetaDataset = previous task set $\mathbb{T}^*$
- Pipeline configuration = superparameter configuration
- Pipeline optimization = superoptimization
- Search strategy = superoptimization algorithm

Fixed experiment setting

We then define the fixed setting of the experiment. A metasample is instantiated with an identifier string, the training dataset, the test dataset and results (a pandas dataframe of pipeline configurations and resulting performance indices). In this example pickle loads prestored instances of metasamples. A metadataset is instantiated with a list of metasamples and a list of metafeature functions, mappings from a dataset to a metafeature. Futhermore the identifier of target datasets, the search space and the objective, which is the mapping from pipeline configuration to the performance index, are defined. The maximum number of iterations for the pipeline optimizations is specified in max_evals, the number of duplicates is defined and the size of the initialization batch is defined in n_init_configs.

```
# Initialize metadataset and calculate metafeatures
metadataset_sample_names = !ls ../../data/metadata/interim
metasamples = [pickle.load(open('../../data/metadata/interim/' + sample_name,"
                                rb")) for sample_name in
                                metadataset_sample_names]
metadataset = MetaDataset(metasamples, metafeature_functions=[size, cumac])
objective = thesis_lookup_objective
search_space = thesis_search_space()
target_ids = [metasample.identifier for metasample in metadataset.metasamples]

# Experiment practicalities
max_evals = 50
duplicates = 2
n_init_configs = 5
```

Variable experiment setting

The variable experiment setting defines the compared search strategies. Since the current implementation integrates with the open-source library hyperopt, search strategies are limited to Bayesian optimization (BayesianHopt), which is instantiated with an identifier, search space, objective and a maximum number of iterations. A naive Bayesian hyperoptimization has an initial set of random configurations, defined by nr_random_starts. The search strategy is a random search if nr_random_starts is set equal to max_evals.

```
# initialize search strategies
rand = BayesianHopt(
    identifier='Random search',
    search_space=search_space,
    objective=objective,
    max_evals=max_evals,
    nr_random_starts=max_evals
)
naive = BayesianHopt(
    identifier='Naive Bayesian optimization',
    search_space=search_space,
    objective=objective,
    max_evals=max_evals,
    nr_random_starts=n_init_configs
)
```

The Bayesian hyperoptimization can be given a warmstarter object, instantiated with the defined metadataset, the number of most similar samples and the number of best configurations per sample and the number of suggested initialization configurations.

```
warm = BayesianHopt(
    identifier='Warmstarted Bayesian optimization',
    search_space=search_space,
    objective=objective,
    max_evals=max_evals,
    warmstarter=Warmstarter(metadataset, n_init_configs=n_init_configs,
                                         n_sim_samples=5, n_best_per_sample=
                                         5)
)
cold = BayesianHopt(
    identifier='Coldstarted Bayesian optimization',
    search_space=search_space,
    objective=objective,
    max_evals=max_evals,
    warmstarter=Warmstarter(metadataset, n_init_configs=n_init_configs,
                                         n_sim_samples=5, n_best_per_sample=
                                         5, cold=True),
)
```

The experiment is then instantiated by giving it the Bayesian hyperoptimizations and the number of duplicates to average over.

```
# initialize hyperoptimization experiment
hopt_exp = HoptExperiment(
    hopts=[rand, naive, warm, cold],
    duplicates=duplicates,
    objective=objective,
    metadataset=metadataset
)
```

Results

Experiment results in a dataframe are added as an attribute by calling the run function on the pipeline optimization experiment.

```
hopt_exp.run_hopt_experiment(target_ids)
```

Several visualizers show the experiment results, for example the averaged performance over duplicates so far, with respect to the amount of iterations of the search strategy, as shown in Figure D-1.

```
hopt_exp.visualize_avg_performance(target_ids[12])
```

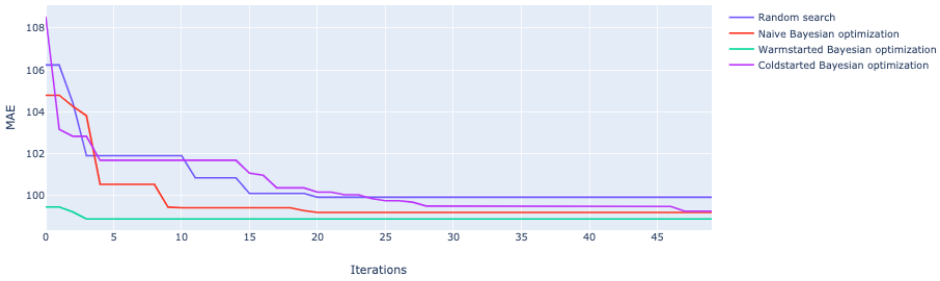


Figure D-1: Example of visualizer: average performance per iteration on one dataset

The library provides a base implementation to build more sophisticated features on as research in metalearning progresses.

Appendix E

Other badly tuned warmstarts

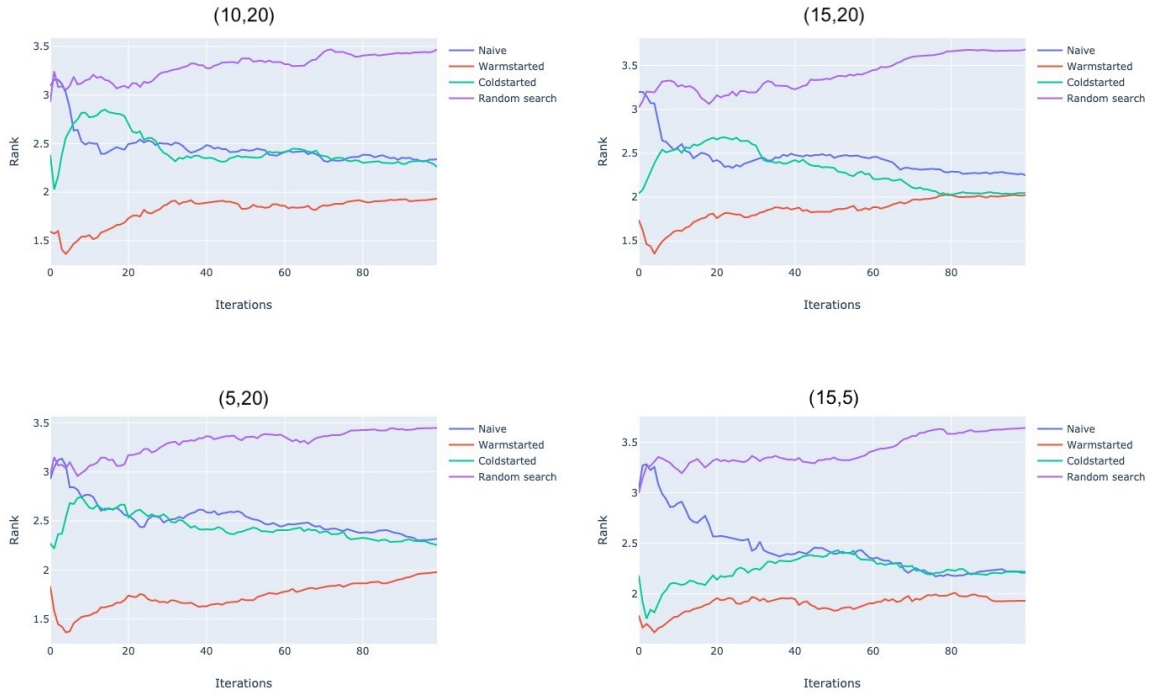


Figure E-1: The performance of the badly tuned proposed warmstart, compared with the benchmarks. The metaparameter settings (k,r) are given in the titles. The methods perform superoptimization for a forecasting task at hand. At every iteration they are ranked according to their best-so-far performance index. A lower rank means better performance. The results are averaged over all forecasting tasks and 3 duplicates.

Bibliography

[1] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, pp. 2951–2959.

[2] M. Suganuma, S. Shirakawa, and T. Nagao, "A genetic programming approach to designing convolutional neural network architectures," in *Proceedings of the genetic and evolutionary computation conference*, pp. 497–504.

[3] C. Lemke, M. Budka, and B. Gabrys, "Metalearning: a survey of trends and technologies," *Artificial intelligence review*, vol. 44, no. 1, pp. 117–130, 2015.

[4] J. Vanschoren, *Meta-learning*, pp. 39–68. Springer, 2019.

[5] T. A. Gomes, R. B. Prudêncio, C. Soares, A. L. Rossi, and A. Carvalho, "Combining meta-learning and search techniques to select parameters for support vector machines," *Neurocomputing*, vol. 75, no. 1, pp. 3–13, 2012.

[6] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and machine learning forecasting methods: Concerns and ways forward," *PloS one*, vol. 13, no. 3, p. e0194889, 2018.

[7] S. Makridakis, C. Chatfield, M. Hibon, M. Lawrence, T. Mills, K. Ord, and L. F. Simmons, "The m2-competition: A real-time judgmentally based forecasting study," *International Journal of forecasting*, vol. 9, no. 1, pp. 5–22, 1993.

[8] S. Makridakis and M. Hibon, "The m3-competition: results, conclusions and implications," *International journal of forecasting*, vol. 16, no. 4, pp. 451–476, 2000.

[9] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "The m4 competition: Results, findings, conclusion and way forward," *International Journal of Forecasting*, 2018.

[10] A. Redd, K. Khin, and A. Marini, "Fast es-rnn: A gpu implementation of the es-rnn algorithm," *arXiv preprint arXiv:1907.03329*, 2019.

[11] M. Feurer, J. T. Springenberg, and F. Hutter, "Initializing bayesian hyperparameter optimization via meta-learning," in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 7.

[12] J. H. Cochrane, "Time series for macroeconomics and finance," *Manuscript, University of Chicago*, 2005.

[13] N. Elamin and M. Fukushige, "Modeling and forecasting hourly electricity demand by sarimax with interactions," *Energy*, vol. 165, pp. 257–268, 2018.

[14] M. E. Torres, M. A. Colominas, G. Schlotthauer, and P. Flandrin, "A complete ensemble empirical mode decomposition with adaptive noise," in *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 4144–4147, IEEE.

[15] Y. Zhou, T. Li, J. Shi, and Z. Qian, "A ceemdan and xgboost-based approach to forecast crude oil prices," *Complexity*, vol. 2019, 2019.

[16] A. Gunasekarage and D. M. Power, "The profitability of moving average trading rules in south asian stock markets," *Emerging Markets Review*, vol. 2, no. 1, pp. 17–33, 2001.

[17] V. Flunkert, D. Salinas, and J. Gasthaus, "Deepar: Probabilistic forecasting with autoregressive recurrent networks," *arXiv preprint arXiv:1704.04110*, 2017.

[18] R. M. Zur, Y. Jiang, L. L. Pesce, and K. Drukker, "Noise injection for training artificial neural networks: A comparison with weight decay and early stopping," *Medical physics*, vol. 36, no. 10, pp. 4810–4818, 2009.

[19] S. Weisberg, *Applied linear regression*, vol. 528. John Wiley Sons, 2005.

[20] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[21] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International journal of forecasting*, vol. 22, no. 4, pp. 679–688, 2006.

[22] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[23] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," *Cited on*, vol. 14, no. 8, 2012.

[24] A. Liaw and M. Wiener, "Classification and regression by randomforest," *R news*, vol. 2, no. 3, pp. 18–22, 2002.

[25] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology," *arXiv preprint arXiv:1809.03006*, 2018.

[26] R. Adhikari and R. Agrawal, "An introductory study on time series modeling and forecasting," *arXiv preprint arXiv:1302.6613*, 2013.

[27] M. H. Ali, I. J. J. o. N. R. Abustan, and Development, "A new novel index for evaluating model performance," vol. 55, no. unknown, pp. 1–9, 2014.

[28] S. Pool, M. Vis, and J. J. H. S. J. Seibert, "Evaluating model performance: towards a non-parametric variant of the kling-gupta efficiency," vol. 63, no. 13-14, pp. 1941–1953, 2018.

[29] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," *arXiv preprint arXiv:1502.02127*, 2015.

[30] G. E. Box and G. M. Jenkins, "Some recent advances in forecasting and control," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 17, no. 2, pp. 91–109, 1968.

[31] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice*. OTexts, 2018.

[32] P. Goodwin, "The holt-winters approach to exponential smoothing: 50 years old and going strong," *Foresight*, vol. 19, pp. 30–33, 2010.

[33] C. C. Holt, "Forecasting seasonals and trends by exponentially weighted moving averages," *International journal of forecasting*, vol. 20, no. 1, pp. 5–10, 2004.

[34] R. Hyndman, A. B. Koehler, J. K. Ord, and R. D. Snyder, *Forecasting with exponential smoothing: the state space approach*. Springer Science Business Media, 2008.

[35] P. R. Winters, "Forecasting sales by exponentially weighted moving averages," *Management science*, vol. 6, no. 3, pp. 324–342, 1960.

[36] J. S. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in neural information processing systems*, pp. 2546–2554.

[37] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[38] K. A. Smith-Miles, "Cross-disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 6, 2009.

[39] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International conference on learning and intelligent optimization*, pp. 507–523, Springer.

[40] M. Feurer and F. Hutter, *Hyperparameter optimization*, pp. 3–33. Springer, Cham, 2019.

[41] M. R. Bonyadi, M. R. Azghadi, and H. Shah-Hosseini, *Population-based optimization algorithms for solving the travelling salesman problem*. IntechOpen, 2008.

[42] C. Spearman, "The proof and measurement of association between two things," 1961.

[43] M. Reif, F. Shafait, and A. Dengel, "Meta-learning for evolutionary parameter optimization of classifiers," *Machine learning*, vol. 87, no. 3, pp. 357–380, 2012.

Glossary

List of Acronyms

SMBO	Sequential Model Based Optimization
TPE	Tree Parzen Estimator
ML	Machine Learning
POA	Population-based Optimization Algorithm
AutoML	Automated Machine Learning