Automated machine learning for dynamic energy management using time-series data

Wang, C.

Chapter 5

AutoML for single-step forecasting

5.1 Introduction

In the previous chapter, we investigated the use of AutoML methods in performing load forecasting tasks, which are the special cases of time-series forecasting. We showed that the current AutoML methods show promising results in the short term load forecasting domain. In this chapter, we investigate the use of AutoML for constructing machine learning models for time-series forecasting tasks. Different from the previous chapter, we design an AutoML framework specifically for time-series forecasting. Feature engineering has been proven to be necessary in many machine learning tasks, including time-series forecasting (Coyle et al., 2005; Phinyomark et al., 2014). The biggest challenge in AutoML for time-series forecasting is to develop advanced ML pipelines that include feature extractors and machine learning models that are well-suited to time-series tasks. Another challenge arises in the selection of the size of the time window that determines how much of the historical information is used as the basis for forecasting future data points. The best window size depends on the properties of the time-series and differs between data sets. It also depends on the ML method used for forecasting.

We compare how different combinations of feature-extraction and window size selection techniques can be used to obtain highly accurate models fully automatically, without the need for making design choices by human experts. Specifically, the con-

tributions made in this chapter are:

- We propose to combine automated time-series feature engineering and window size selection, with automated algorithm selection and hyperparameter optimization to generate machine learning pipelines for time-series forecasting. The focus is on time-series forecasting which is implemented as a regression task.

- We compare the performance of the pipelines generated using our proposed technique against 18 different machine learning models configured with fixed windows sizes across 20 data sets. Our proposed new method aims to be generally applicable for time-series forecasting on arbitrary data sets. We, therefore, assess the performance of the proposed method on a diverse set of problems. We demonstrate the importance of feature selection and window size selection in forecasting problems. We further show the strong performance of our proposed technique in creating specialized AutoML systems for time-series data building upon the available AutoML system of auto-sklearn (Feurer et al., 2015) and a well-known time-series feature extraction library tsfresh (Christ et al., 2018). We propose three new approaches for AutoML for time-series forecasting: (i) auto-sklearn with automated window size selection; (ii) auto-sklearn with tsfresh features; and (iii) auto-sklearn with automated window size selection as well as tsfresh features. Among these, we demonstrate that (i) performs better compared to two other AutoML frameworks: i.e., AutoKeras (Jin et al., 2019), auto-sklearn.

The remainder of this chapter is structured as follows: in Section 5.2, we give a brief overview of the existing work in time-series forecasting and AutoML. Section 5.3 provides the formal problem definition of automated time-series forecasting. Section 5.4 introduces the AutoML systems and feature extraction methods studied in this chapter along with our proposed approaches. Section 5.5 presents our experimental setup, results and discussion. Finally, conclusions and future work directions are provided in Section 5.6.

5.2 Related work

Time-series forecasting: Existing time-series forecasting methods can be mostly divided into two categories: (i) *statistical*, and (ii) *machine learning* methods as we

have mentioned in Section 4.2 Both of these categories aim at making accurate predictions. Their differences lie in the fact that statistical algorithms are typically limited as they assume linear dependence on past observations, while machine learning models are non-linear (Makridakis et al., 2018). Furthermore, machine learning algorithms require the training of predictive models using substantially more computational resources than statistical algorithms.

Feature extraction: In order to apply methods from either of these two categories, it is typically necessary to perform some form of preprocessing on the raw data. Feature extraction is an important preprocessing step used in many cases. So far, many different feature extraction methods for time-series have been used in the context of different applications (e.g., see (Coyle et al., 2005; Phinyomark et al., 2014)). Feature Extraction based on Scalable Hypothesis tests (FRESH) (Christ et al., 2018) is a generic framework for time-series feature extraction. FRESH includes various categories of features such as statistical features, features of sample distributions and features of observed dynamics. A Python implementation of FRESH is available in the tsfresh package (Christ et al., 2018). Tsfresh has been widely used in time-series analysis (e.g., (Dempsey et al., 2020; Yuan et al., 2019)). Catch22 (Lubba et al., 2019) and hctsa (Fulcher and Jones, 2017) are two other feature extraction libraries. Hctsa extracts over 7,700 features selected based on the time-series analysis literature. Such a large number of features make the process of feature extraction computationally expensive. Catch22 (CAnonical Time-series CHaracteristics) presents a set of 22 of the hctsa features, selected on a time-series classification benchmark set. Compared to hctsa, using Catch22 provides an approximately 1000-fold reduction in computation time with only 7% reduction in classification accuracy. However, being selected for classification tasks, these features are not guaranteed to perform well on forecasting problems considered in this chapter. Franceschi et al. (Franceschi et al., 2019) proposed feature extraction using unsupervised representation learning for multivariate time-series classification and regression problems. However, using a deep convolutional neural network also generates high additional computational burdens for preprocessing. Being focused on time-series forecasting problems and opting for lower computational burdens, in our experiments we chose to use tsfresh, which computes a total of 794 time-series features.

AutoML: Another line of related work comes from the area of automated machine learning (AutoML). We comprehensively reviewed AutoML systems in Chapter 3. AutoML has been used in many domains for constructing high-performance machine learning models for a given data set, without the need for machine learning

experts to choose learning algorithms, model classes or hyperparameter settings. For example, AutoML systems have been used for creating models for image classification (Zoph et al., 2018), prediction of the fuel consumption of ships (Ahlgren and Thern, 2018), as well as prediction of biological ecosystem networks (Barreiro et al., 2018). These works mainly involve automating classification and regression tasks. However, they do not consider AutoML for time-series forecasting or the use of time-series features in AutoML. In our earlier work (Wang et al., 2019) we showed the potential of AutoML systems for energy load forecasting by defining a regression problem using additional features (not extracted from the time-series). Kefalas et al. (Kefalas et al., 2021) looked at the problem of remaining useful life estimation of aircraft engines by specifying a specific form of regression problem on time-series data.

In this chapter, we aim at studying how AutoML systems can be enhanced to perform in general time-series forecasting. Auto-sklearn (Feurer et al., 2015), AutoGluon (Shi et al., 2021) and Auto-Keras (Jin et al., 2019) are examples of available AutoML systems. Auto-keras is a deep-learning-based AutoML system. AutoGluon focuses on automated stack ensembling and deep learning techniques, while auto-sklearn is built upon a search space composed of classic machine learning algorithms. In this chapter, we use auto-sklearn for two reasons. Firstly, based on the time-series forecasting literature the feature extraction and pre-processing methods studied in this chapter can complement various algorithms in auto-sklearn (e.g., Support vector machines) that are not available in the other systems. Furthermore, our initial investigation showed that it is much easier to extend auto-sklearn to acquire reliable results. We also use Auto-Keras as a deep learning baseline to compare the performance achieved using our proposed pipelines with classic machine learning models against automatically configured deep learning models that do not require feature extraction.

On the one hand, the aforementioned statistical and the machine learning models for time-series forecasting all need human experts to manually select parameters, such as the windows size (over which features are extracted), the model to use, and the model hyperparameters. On the other hand, current AutoML systems, while supporting model selection and hyperparameter optimisation, do not support the automated prepropessing required for many time-series forecasting techniques. For applicability on time-series data, AutoML systems need to support automated window size selection and time-series feature generation. Here, we integrate time-series feature extraction into AutoML systems. We propose variants of recent AutoML approaches that automatically choose a window size, select one or more machine learning models, optimise hyperparameters and generate time-series features. We benchmark our proposed Au-

toML approaches on a broad range of time-series forecasting problems.

5.3 Problem definition

We have defined the time-series forecasting problem in Chapter 2. Current AutoML techniques cannot optimise the window size. When using an available AutoML system to do time-series forecasting, one needs to select a value for w and preprocess the data manually into pairs of feature vectors and target values. Current AutoML systems do not support time-series feature extraction either. In automated time-series forecasting, however, both window size selection and feature extraction should be performed automatically.

The input raw data are extracted over a window of size w. Given a time-series segment $\hat{\mathbf{x}} = [x_{i-w}, \ldots, x_{i-1}]$, a feature extraction function g and a machine learning model A, we are interested in forecasting x_i. With feature engineering, we can write this as $x_i = A(g(\hat{\mathbf{x}}))$, and without it as $x_i = A(\hat{\mathbf{x}})$.

Given a time-series data set $\mathbf{x} = [x_1, \ldots, x_n]$ that is split into $\mathbf{x}_{train}$ and $\mathbf{x}_{valid}$, we are interested in building an optimised model using $\mathbf{x}_{train}$ by minimising a loss on $\mathbf{x}_{valid}$. *Let $\mathcal{A} = \{A^{(1)}, \ldots, A^{(k)}\}$ be a set of algorithms with associated hyperparameter spaces $\Lambda^{(1)}, \ldots, \Lambda^{(k)}$. Let $\mathbf{w} = \{w^{(1)}, \ldots, w^{(l)}\}$ be the set of the possible window sizes. Further, let $\mathbf{x}_{train}$ be a training set, and $\mathbf{x}_{valid}$ be a validation set. Finally, let $\mathcal{L}(A^{(i)}_{\lambda}, w^{(j)}, \mathbf{x}_{train}, \mathbf{x}_{valid})$ denote the loss that algorithm $A^{(i)}$ achieves on $\mathbf{x}_{valid}$ when trained on $\mathbf{x}_{train}$ with hyperparameters $\lambda \in \Lambda^{(i)}$ and window size $w^{(j)}$. Then the automated time-series forecasting problem is to find the window size, algorithm, and hyperparameter setting that minimises this loss:*

$$(A^*, \lambda^*, w^*) \in \underset{A^{(j)} \in \mathcal{A}, \lambda \in \Lambda^{(j)}, w \in \mathbf{w}}{\arg\min} \mathcal{L}(A^{(i)}_{\lambda}, w^{(j)}, \mathbf{x}_{train}, \mathbf{x}_{valid}) \tag{5.1}$$

5.4 Methods

In our study, we use auto-sklearn (Feurer et al., 2015) as the main AutoML system, and tsfresh (Christ et al., 2018) for time-series feature extraction; these were chosen because they are state-of-the-art, prominent, and freely available. Auto-sklearn can optimise machine learning pipelines, which corresponds to selecting A^* and λ^* in Equation 5.1. Tsfresh realises the feature extraction function g from Section 5.3.

5.4.1 Auto-sklearn

We use auto-sklearn in our experiments. We previously provided information about auto-sklearn Chapter 4. The feature engineering component of auto-sklearn is powerful and widely used in classification and regression tasks. However, it has not been designed for time-series analysis tasks, nor does it offer native support for such tasks. In particular, it does not support automatic window size selection, nor the specific time-series features offered, for example, by tsfresh (Christ et al., 2018).

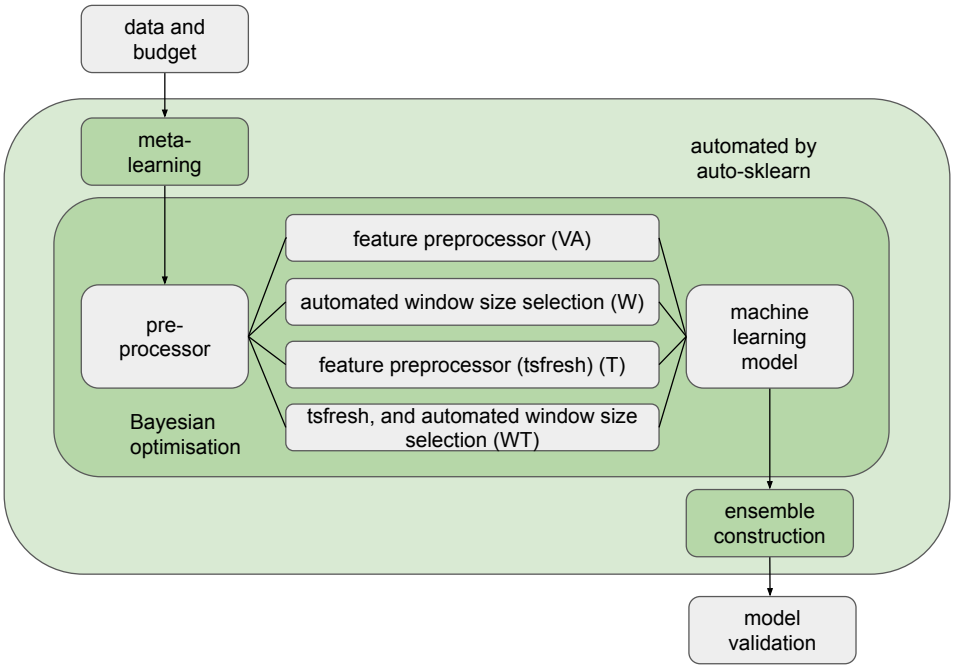

Figure 5.1: Workflow of vanilla auto-sklearn, and three variants of auto-sklearn are specially designed for time-series forecasting tasks. VA represents the vanilla auto-sklearn with its default feature preprocessor. W is the simplest variant of auto-sklearn for time-series forecasting and considers only automated window size selection without any feature engineering. T extracts tsfresh time-series features from the input data, and the window size is predefined and fixed in this case. WT optimises the pipeline with window size selection and time-series feature engineering techniques. Each method is different from the others, only concerning the middle block presented in the figure.

5.4.2 Tsfresh

Feature engineering is an essential component in ML pipelines. Feature extraction methods have also been used for time-series analysis tasks (Coyle et al., 2005; Phinyomark et al., 2014). There are a number of time-series feature extraction libraries that are widely used for time-series analysis, including tsfresh (Christ et al., 2018), Catch22 (Lubba et al., 2019), and hctsa (Fulcher and Jones, 2017). Catch22 extracts a selected list of the 22 most useful features of the 4791 features of hctsa from a time-series. Extracting features with Catch22 is more computationally efficient than hctsa, with only a 7% reduction in accuracy on average. In the following, we use tsfresh, which extracts more than 700 time-series features in parallel and has previously shown strong performance (Christ et al., 2018), since it does not require huge computational resources like hctsa, or suffers from an accuracy reduction like Catch22. Tsfresh covers the feature extraction methods including features from summary statistics (e.g., maximum, minimum and mean); additional characteristics of the sample distribution (e.g., number of data points above-median); and features derived from observed dynamics (e.g., mean absolute change). In total, tsfresh provides 63 time-series feature extractors, which can extract 794 time-series features. The full list of these features can be found in (Christ et al., 2018).

5.4.3 AutoML for time-series forecasting

In this section, we introduce our newly proposed approaches: three variants of auto-sklearn that are specially designed for time-series forecasting tasks.

Auto-sklearn with automated window size selection (W): the simplest variant of AutoML for time-series forecasting considers only automated window size selection by searching for the optimal window size w. No extra feature extractors are used, and only the value of the time-series $\hat{\mathbf{x}} = [x_{i-w+1}, \ldots, x_i]$ are used to train a model that predicts x_{i+w}. Figure 5.1 shows how the workflow of auto-sklearn is modified for automated window size selection. The original feature preprocessor is replaced with an automated window size selection component. The window size w is optimised as an additional hyperparameter using the core Bayesian optimisation procedure of the AutoML framework.

Auto-sklearn with tsfresh features (T): Figure 5.1 shows the workflow of auto-sklearn with automated tsfresh feature extractor. In this case, tsfresh extracts time-series features from the input data and the window size is predefined and fixed. The time-series features $g(\hat{\mathbf{x}}) = g([x_i, \ldots, x_{i+w-1}])$ are used to train a model that

predicts x_{i+w}. The original features are not contained. Since there are many useful methods to evaluate feature importance, feature selection does not have to be done by AutoML. The feature importance is assessed by univariate tests. (Christ et al., 2018) performed a comparative study of different feature selection algorithms (Benjamini and Hochberg, 1995; Kursa and Rudnicki, 2011; Wang et al., 2013) which demonstrated the outperforming performance of the approach proposed in (Benjamini and Hochberg, 1995). Therefore, we selected this approach for feature selection.

Auto-sklearn with automated window size selection and tsfresh features (WT): the two previously mentioned approaches (auto-sklearn with automated window size selection and auto-sklearn with tsfresh features) are combined in this approach by optimising the window size w and extracting the time-series features from the resulting window. The time-series features $g(\hat{\mathbf{x}}) = g([x_i, \ldots, x_{i+w-1}])$ are used to train a model that predicts x_{i+w}. Figure 5.1 shows the workflow of auto-sklearn with automated window size selection and tsfresh feature preprocessor. The additional hyperparameter w is optimised in the AutoML framework using the core Bayesian optimisation procedure. Changes in window size cause different features to be extracted by tsfresh.

5.5 Empirical results

Setup of computational experiments: In our experiments[1], we used version 0.8.0 of auto-sklearn and version 0.16.1 of tsfresh. All experiments were run on 8 cores of an Intel Xeon E5-2683 CPU (2.10 GHz) with 10 GB RAM. In Equation 5.1, we have formalised our goal to minimise a given loss function. In our experiments, we used *RMSE* (root mean squared error) as a performance metric in the optimisation to evaluate the quality of machine learning pipelines. *RMSE* is one of the performance metrics that has been used widely on regression tasks (Tan et al., 2021). Any other performance metric can also be used to calculate the loss.

For this error metric, we define $x'_t = A(\gamma(x_{t-w}, \ldots, x_{t-1}))$ as the predicted value at timestamp t, with $\gamma = \mathrm{id}$ (where $\mathrm{id}(x_i) = x_i$) when time-series features are not used, and $\gamma = g$, otherwise. A is a machine learning model. Then, with $e_t = x_t - x'_t$ defined as the error between x_t and x'_t, the root mean square error is given by $RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$. n represents the length of the time-series. The time limit for each evaluation was set to 20 minutes, and the time limit for each run of auto-sklearn to

[1]The data sets and source codes used in our experiments are available in https://github.com/wangcan04/AutoML-timeseries

2 hours. In these experiments, we used hold-out validation (training:testing = 67:33), the default validation technique in auto-sklearn. We did not shuffle the data set, in order to preserve the sequential nature of the time-series data. All other settings of auto-sklearn were left at their defaults. Since experiments are time-consuming, we used bootstrapping to create a distribution of the errors to study performance variability over multiple independent runs of auto-sklearn. For this, every experiment was executed 25 times. Then we randomly sampled 5 out of the 25 results. We selected the model with the lowest training *RMSE* out of these five models. We repeated the sampling 100 times per model and data set. The solution quality distributions reported in our results were thus obtained over the 100 samples. When we performed automated window size selection, the maximum window size was set to 200. To obtain a fair comparison, we skipped the first 200 points of the time-series before making the train-test split; this helped ensure that the training set and test set were the same for all the evaluated approaches.

data sets used: We used CompEngine (Fulcher et al., 2019), a comparison time-series data engine containing 29 498 data sets in 197 categories (both real-world and synthetic). Other popular time-series data libraries, such as the UCR time-series archive (Dau et al., 2018) (only classification data sets) and the UCI machine learning repository (only 39 data sets, mostly multivariate) were found to be unsuitable for our purposes since we focus on univariate tasks. We used 20 data sets from CompEngine[2] as our benchmark, of which ten are real-world data sets[3], and ten synthetic data sets[4]. We chose these by selecting the first data set from every category. As explained earlier, we split each set into 67% training set and 33% test set, based on temporal order.

Since our data sets are all from different categories, the models are trained per time-series. This means that we use a local method in contrast to a global method, which trains one model on all time-series simultaneously (see (Januschowski et al., 2020)).

Research questions: In our experiments, we addressed the following research questions:

- (Q1): To what extent can AutoML techniques beat simple baselines?

[2]https://www.comp-engine.org/

[3]Audio (Animal sounds, Human speech, Music), Ecology (Zooplankton growth), Economics (Macroeconomics, Microeconomics), Finance (Crude oil prices, Exchange rate, Gas prices), Medical (Electrocardiography ECG).

[4]Flow (Driven pendulum with dissipation, Duffing-van der Pol Oscillator, Driven van der Pol oscillator, Duffing two-well oscillator, Diffusionless Lorenz Attractor), Stochastic Process (Autoregressive with noise, Correlated noise, Moving average process, Nonstationary autoregressive, Random walk).

- (Q2): To what extent can current AutoML approaches beat deep neural network baselines?
- (Q3): To what extent can our newly proposed approaches beat current AutoML approaches?
- (Q4): Which approach generalises best from training to test data?
- (Q5): Can optimal window sizes for time-series forecasting be found by the AutoML system?

To address these research questions, we compare the approaches we mentioned in Section 5.4 with simple statistical and machine learning baselines and AutoML approaches. The first nine following approaches are our baselines, and the rest are the AutoML variants proposed by us:

- **Moving average (MV1, MV10, MV200):** This is a simple and commonly used method for time-series forecasting. We use this approach to create three models considering the mean of the previous 1, 10, 200 data points to predict the next point. This method does not have any hyperparameters. As this is a naïve baseline, any approach should perform better in comparison to it.
- **Auto Regressive Integrated Moving Average (ARIMA):** ARIMA is a widely used statistical model for time-series forecasting. In this chapter, we used the auto.arima function from the Python library pmdarima (Smith et al., 2017) that selects the best hyperparameters for ARIMA models for a given data set. We used auto_arima to optimise three important hyperparameters p, d, q, that control the auto-regressive, trend and moving average components. We ran experiments with the default setting of auto.arima that evaluated 50 models for hyperparameter tuning. Since the default search space is small, 50 iterations are enough to find an optimal order for a data set (Smith et al., 2017).
- **Support vector machine (SVM10, SVM200):** SVM is a classical machine learning model that has been commonly used for time-series forecasting (e.g., (Candanedo et al., 2017)). Since we aim to perform single-step forecasting, we assume that a window size of 10 is a reasonable value that captures the most important temporal correlations. To investigate the influence of larger window sizes, we also performed experiments with a window size of 200 (the maximum window size for all experiments). We used time-series values in these window sizes and searched for an optimised machine learning pipeline based on SVMs

for forecasting the next time-step. Next to optimising the hyperparameters, we also searched for the best machine learning pipeline composed of the feature extractors and preprocessors available in auto-sklearn. For optimising hyperparameters of single algorithms, it is common to use simple approaches such as random search and grid search. Here, we used random search, since it is known to often be more efficient than grid search (Bergstra and Bengio, 2012). For implementing this approach, we used auto-sklearn, setting the regressor to SVM and selecting the use of random search instead of SMAC (the default optimiser in auto-sklearn). All remaining settings were left at their default values. We used a 2-hour time limit for optimising the pipeline (the same as the limit used for our proposed AutoML variants).

- **Random forest (RF10, RF200):** Random forest is another classical machine learning model that can be used for time-series forecasting (Candanedo et al., 2017; Wang et al., 2019). We trained RF10 and RF200, following the same process used for training SVM10 and SVM200.

- **Gradient Boosting Machine (GBM10, GBM200):** Gradient Boosting Machine is another classical machine learning model that has been widely used in the past for time-series forecasting tasks (Li et al., 2020b; Srivastava et al., 2020). GBM uses an ensemble of weak prediction models, which means a group of models that are slightly better than random chance. We trained GBM10 and GBM200, following the same process used for training SVM10 and SVM200.

- **eXtreme Gradient Boosting (XGBoost10, XGBoost200):** XGBoost (Chen and Guestrin, 2016) is a method based on gradient boosting. Compared to GBM, XGBoost uses a more regularized model formalization to prevent overfitting. In the M5 forecasting competition, the top 50 performing methods almost all involve Light-GBM, XGBoost and NNs (mostly LSTM) (Makridakis et al., 2021). XGBoost using default parameterization has shown competitive performance in this competition. The process of training XGBoost10 and XGBoost100 is similar to that of SVM experiments. However, instead of using auto-sklearn for a random search, we used h2o AutoML (LeDell and Poirier, 2020) since auto-sklearn does not support XGBoost.

- **Auto-Keras (Auto-Keras10, Auto-Keras200):** Auto-Keras (Jin et al., 2019) is an AutoML system for neural architecture search. It uses network morphisms and Bayesian optimisation to efficiently search for high-performance

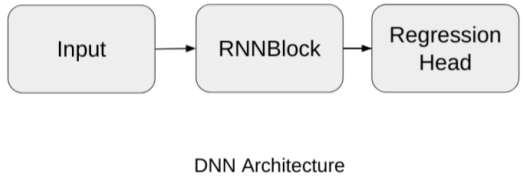

Figure 5.2: Workflow of deep neural network for time-series forecasting. The input data goes through RNNBlock, the critical component of a deep neural network. The hyperparameters will be optimised during the search, such as the number of layers and learning rate. The Regression Head then generates output based on the information from RNNBlock.

neural network architectures. Auto-Keras does not have a time-series forecasting component, but different neural network units available in its search space (e.g., LSTM, GRU) have been previously used for time-series forecasting (e.g., (Siami-Namini et al., 2018; Zhang et al., 2017)). We similarly used values within windows of size 10 and 200 to train a neural network that forecasts the next point in the time-series using the *StructuredDataRegressor* function in Auto-Keras. Auto-Keras allows to set of the maximum number of models evaluated, but it does not allow specifying the time budget as auto-sklearn does. To ensure the fairness of our comparisons, we calculated the average running time of 10 models on each data set and used this to estimate, for each data set, the number of models that can be evaluated in 2 hours. We used hold-out validation (training: testing = 80:20), which is the default setting in Auto-Keras.

- **Deep neural network baseline (LSTM10, LSTM200, GRU10, GRU200):** We use LSTM and GRU as baselines, as these have been recently studied in the literature on time-series forecasting (see, e.g., (Siami-Namini et al., 2018; Yamak et al., 2019)). When using deep neural networks, it is common practice to design the architecture (e.g., by selecting the number of layers) and set the hyperparameters for training (e.g., number of epochs, learning rate) manually, as done in the work of (Siami-Namini et al., 2018) and (Yamak et al., 2019). We believe that it is more effective to automate this process and to carry it out separately for each given data set. To find suitable architectures based on these types of networks for each of our data sets, we adapted the search space of Auto-Keras to automatically design the architecture. Specifically, to deal with time-series data, we designed our new search space using three types of blocks available in Auto-Keras: Input block, RNNBlock and Regression Head (see Fig-

ure 5.2). The RNNBlock is the key component in our networks. A number of hyperparameters need to be considered for this block, including bidirectionality, the number of layers and layer type (LSTM or GRU). Since we performed the experiments for testing the LSTM and GRU architectures separately, we did not optimise layer type; instead, we set the value of this hyperparameter manually in each experiment. We used Auto-Keras to optimise the architecture based on the customised search space. Since Auto-Keras cannot choose the window size, we selected window sizes of 10 and 200, and manually preprocessed the data. We fed the values of the time-series within these windows to Auto-Keras in order to obtain a model, once again using a time budget of 2 hours and a memory budget of 80GB. We used a random search to find the model and left the remainder of the setup of Auto-Keras at its default settings (number of epochs, validation techniques, etc.).

- **N-BEATS:** N-BEATS (Oreshkin et al., 2019) uses deep learning involving residual neural networks and fully-connected layers to deal with univariate time-series forecasting tasks. N-BEATS has shown good performance on several well-known data sets. We executed every experiment 25 times. The distribution reported in our results was obtained by the bootstrapping approach, explained in Section 5.5 In our experiments, we train N-BEATS using the default hyperparameters in the implementation provided by the paper. The number of epochs is 500.

- **Vanilla auto-sklearn (VA10, VA200):** For VA (depicted in Figure 5.1), we optimised the machine learning pipelines for time-series forecasting tasks using auto-sklearn. Since auto-sklearn cannot choose the window size, we selected window sizes equal to 10 and 200 and subsequently fed the manually preprocessed values of the time-series within these windows to auto-sklearn in order to obtain a forecasting model, once again using a time budget of 2 hours.

- **Auto-sklearn with automated window size selection (W_ens, W_single):** For the W variant (depicted in Figure 5.1), we optimised the machine learning pipeline as well as the window size. Therefore, the time-series can be used without additional preprocessing. In our experiments, the window size varies from 2 to 200. In auto-sklearn, the final model produced is by default an ensemble model that can be composed of models configured with different window sizes. We performed two separate experiments to study the optimal window size, both with ensembling (denoted by W_ens) and without ensembling

(W_single). In W_single, only one machine learning model configured with a single window is selected.

- **Auto-sklearn with tsfresh features (T10, T200)**: In the T variant (depicted in Figure 5.1), we use tsfresh as an internal component of our AutoML system. This component extracts features within a window, the size of which has been set to 10 and 200. Without additional manual preprocessing, the tsfresh component extracts the time-series features from the training data, and auto-sklearn subsequently uses these features as input data and finds a configuration that performs well on them.

- **Auto-sklearn with automated window size selection and tsfresh features (WT)**: In WT (depicted in Figure 5.1), by having tsfresh and the automated window size selection component, we do not need to perform any manual preprocessing. In these experiments, we set the window size range from 2 to 200. The time-series can be used without additional preprocessing. In this experiment, we always obtain an ensemble of forecasting models.

The results of our experiments are summarised in Tables 5.1, 5.2, 5.3, 5.4, 5.5 and 5.6 which compare performances in terms of *RMSE*. In order to determine the statistical significance of the results, we initially performed an Anderson-Darling test (Anderson and Darling, 1952) and found that the error distributions created by the bootstrapping approach, explained in Section 5.5, are not Gaussian. Therefore, we used the non-parametric Mann–Whitney U-test (Mann and Whitney, 1947) with a standard significance level set to 0.05. In the tables, bold-faced entries indicate the lowest mean error achieved on a given data set, and all results statistically tied to the best are marked with $*$. Based on these results, in the remainder of this section, we provide answers to the questions proposed earlier in Section 5.5.

Q1: To what extent can AutoML techniques beat simple baselines (ARIMA, Moving average, SVM, RF, GBM and XGBoost baselines)?

Tables 5.1, 5.2 and 5.3 compare the performance achieved by different methods in terms of *RMSE* on the test set. Table 5.1, 5.2 shows the results for simple baseline models while Table 5.3 presents the results for baseline AutoML techniques. In Table 5.5, the lowest *RMSE* obtained for each data set using simple baseline models and baseline AutoML techniques are presented along with those achieved by our AutoML variants.

Table 5.1: *RMSE* of simple baselines (ARIMA, and Moving average, SVM, RF, GBM, XGBoost with window size = 10) on the test set. The bold-faced entries indicate the lowest mean error achieved on a given data set.

Dataset \ Model	ARIMA	MV1	MV10	SVM10	RF10	GBM10	XGBoost10
Autoregre	**0.400432**	0.610580	0.512732	0.400718	0.404424	0.406328	0.402833
Correlated noise	**1.022264**	4.385376	2.336120	1.027496	1.036663	1.039973	1.034562
Lorenz Attractor	**0.001325**	0.152307	0.734693	0.015520	0.044260	0.031324	0.051661
Pendulum	**0.000620**	2.874323	1.488101	0.002337	0.017249	0.015552	0.045503
Driven oscillator	0.022562	0.208443	0.952037	**0.005747**	0.023396	0.014827	0.021946
Two-well oscillator	0.000258	0.020033	0.109077	**0.000228**	0.004575	0.004209	0.005556
Duffing oscillator	**0.000261**	0.046562	0.248152	0.000664	0.006194	0.005080	0.006358
Moving average	0.439993	1.021399	0.662487	0.441369	0.505118	0.482859	0.497822
Nonstationary	0.994813	3.605841	14.108665	**0.990199**	1.319040	1.264516	1.345824
Random walk	1.006695	**1.005863**	1.958452	1.098413	11.998150	12.387875	12.479568
Crude oil prices	3.954218	**3.944808**	11.855737	5.469766	26.033508	25.818681	26.544268
ECG	9.823127	14.658113	62.507000	**9.214228**	20.4555106	23.087870	22.727534
Exchange rate	0.000668	**0.000662**	0.001098	0.000663	0.005330	0.005837	0.004762
Gas prices	**6.843051**	7.984512	28.207355	7.826223	98.605229	92.366392	93.547291
Human speech	0.0316651	0.055792	0.076618	**0.031606**	0.034001	0.0307520	0.0329440
Macroeconomics	**512.713152**	625.923211	543.892692	557.474601	642.566387	611.218628	637.913982
Microeconomics	170.389959	**170.059750**	330.591176	171.446322	385.804129	445.430152	458.170203
Music	0.039373	0.060195	0.106724	0.044851	0.035303	0.034408	0.037114
Tropical sound	0.006598	0.069521	0.035080	0.006277	**0.006212**	0.006227	0.006331
Zooplankton	266.047819	312.605555	288.962126	263.25297	**257.486771**	290.024830	272.528711

Table 5.2: *RMSE* of simple baselines (Moving average, SVM, RF, GBM, XGBoost with window size = 200) on the test set. The bold-faced entries indicate the lowest mean error achieved on a given data set.

Dataset \ Model	MV200	SVM200	RF200	GBM200	XGBoost200
Autoregre	0.482531	0.411491	0.406045	0.409519	0.410630
Correlated noise	2.261390	1.044179	1.036931	1.042252	1.040713
Lorenz Attractor	1.292518	0.004888	0.062212	0.038435	0.047212
Pendulum	1.443494	0.015913	0.081669	0.088391	0.060481
Driven oscillator	1.277660	0.030615	0.028647	0.017101	0.031403
Two-well oscillator	0.616822	0.001250	0.007276	0.005792	0.007372
Duffing oscillator	0.479744	0.001043	0.006816	0.004251	0.006769
Moving average	0.632330	**0.437374**	0.524114	0.507059	0.516003
Nonstationary	11.680654	1.004374	1.489324	1.419968	1.486933
Random walk	8.348509	1.078787	12.549192	12.428874	12.966513
Crude oil prices	34.56473	51.380480	26.983431	33.805812	28.337624
ECG	149.945466	11.025114	26.620362	31.597675	31.088642
Exchange rate	0.005441	0.000668	0.004816	0.005853	0.002782
Gas prices	94.097732	38.233882	101.034967	90.659451	90.176212
Human speech	0.066414	0.032727	0.037128	0.0348973	0.0375481
Macroeconomics	788.595638	790.718110	840.012363	816.094878	840.274512
Microeconomics	1481.184724	183.942596	451.909546	589.631221	590.172534
Music	0.095336	**0.028995**	0.054897	0.039644	0.038725
Tropical sound	0.034953	0.006610	0.006329	0.006339	0.006342
Zooplankton	315.773444	317.817582	262.21993	270.214120	260.286121

Table 5.3: *RMSE* of AutoML baselines (Auto-keras, vanilla auto-sklearn) on test set. The bold-faced entries indicate the lowest mean error achieved on a given data set, and $*$ indicates the results are statistically tied to the best.

Dataset \ Model	Best baseline	Auto-Keras10	Auto-Keras200	VA10	VA200
Autoregre	0.400718	0.402549	0.412751	**0.400603 ± 0.000086***	0.404322 ± 0.001474
Correlated noise	1.027496	1.029839	1.063033	**1.021476 ± 0.000938***	1.025509 ± 0.001547
Lorenz Attractor	0.004888	0.013126	0.111397	0.003703 ± 0.001902	**0.000646 ± 0.000007***
Pendulum	0.002337	0.009459	0.141597	0.064857 ± 0.244416	**0.001574 ± 0.000106***
Driven oscillator	**0.005747**	0.024633	0.082339	0.012259 ± 0.004473	0.015636 ± 0.001444
Two-well oscillator	**0.000228**	0.045387	0.077271	0.000797 ± 0.000917	0.001475 ± 0.001842
Duffing oscillator	**0.000664**	0.006247	0.023453	0.001509 ± 0.001812	0.001312 ± 0.00197
Moving average	0.437374	0.445282	0.517913	0.441905 ± 0.000381	**0.429657 ± 0.001857***
Nonstationary	0.990199	1.005357	1.473375	0.988265 ± 0.001298	**0.987444 ± 0.000661***
Random walk	**1.005863**	1.307905	11.782249	1.048836 ± 0.016573	1.035739 ± 0.016835
Crude oil prices	**3.944808**	20.946781	26.682971	5.315421 ± 2.542140	32.828312 ± 1.593694
ECG	9.214228	32.7111615	55.142157	**8.297350 ± 1.065379***	8.3175328 ± 0.138348
Exchange rate	0.000662	0.001952	0.011507	**0.000661 ± 0.000001***	0.000665 ± 0.000004
Gas prices	7.826223	6.961940	66.702707	**6.904599 ± 0.033701***	8.648501 ± 0.181225
Human speech	0.031606	0.035305	0.047310	**0.031522 ± 0.000969***	0.036217 ± 0.002604
Macroeconomics	**543.892692**	574.948945	712.831468	579.206474 ± 12.37575	781.995492 ± 25.03351
Microeconomics	**170.059750**	647.763369	10241.14926	170.724059 ± 0.218171	173.318589 ± 1.114270
Music	0.028995	0.037084	0.034293	0.032710 ± 0.000346	**0.027476 ± 0.001178***
Tropical sound	0.006212	0.008190	0.008366	**0.006211 ± 0.000013***	0.006270 ± 0.000018
Zooplankton	**257.486771**	280.185625	301.396218	264.522817 ± 0.686007	264.961660 ± 4.076801

Looking at Table 5.5, we generally observe that, aside from the Microeconomics and Random walk data sets, the best AutoML results (acquired from any of these methods: Auto-Keras10, Auto-Keras200, VA10, VA200, W_ens, W_single, T10, T200, WT) are significantly better than the best baseline results (acquired from any of these methods: ARIMA, MV1, MV10, MV200, SVM10, SVM200, RF10, RF200, GBM10, GBM200, XGBoost10, XGBoost200) in terms of *RMSE*.

Comparing the *RMSE* values in the first two columns of Table 5.5, we observe that in 9 out of 20 data sets, the best AutoML baseline significantly outperforms the best baseline. Auto-Keras10, Auto-Keras200, VA10, VA200 outperform the best baselines on 0, 0, 6, 5 data sets, respectively. Overall, the best AutoML baseline on each data set achieves between 0.02 % and 105.11 % higher accuracy. Comparing the *RMSE* values of the best baseline and our AutoML variants (W_ens, W_single, T10, T200, WT) from Table 5.5 indicates that for 18 out of 20 data sets, at least one of our AutoML variants achieves lower error. The only exceptions are the Microeconomics and Random walk data sets, where no AutoML approach was able to perform better than the best baseline. For the random walk data set, this can be explained by the lack of exploitable structure. W_ens, W_single, T10, T200, WT outperform the best baselines on 10, 10, 4, 4, 5 data sets, respectively.

Q2: To what extent can current AutoML approaches (vanilla auto-sklearn: VA10, VA200, and Auto-Keras: Auto-Keras10, Auto-Keras200) beat deep neural network baselines (LSTM10, LSTM200, GRU10, GRU200, N-

Table 5.4: *RMSE* of deep neural network baselines (LSTM, GRU, N-BEATS) on the test set. In the table, the term 'out of memory' denotes that when evaluating the model, the maximum memory budget of 80 GB was reached. In this case, Auto-Keras did not return a model). The bold-faced entries indicate the lowest mean error achieved on a given data set, and $*$ indicates the results are statistically tied to the best.

Dataset \ Model	Best baseline	Best AutoML	LSTM10	LSTM200	GRU10	GRU200	N-BEATS10	N-BEATS200
Autoregre noise	0.400718	**0.400603 ± 0.000086**	0.457445	0.481179	0.511867	0.480812	0.422572	0.437813
Correlated noise	1.027496	**1.021476 ± 0.000938**	1.347711	1.680885	1.187226	2.256934	1.671850	1.219176
Lorenz Attractor	0.004888	**0.000646 ± 0.000007**	1.270105	1.071072	1.284036	0.559788	0.240261	0.213180
Pendulum	0.002337	**0.001574 ± 0.000106**	0.434611	1.438451	0.446650	0.839472	0.124997	0.211660
Driven oscillator	**0.005747**	0.012259 ± 0.004473	0.487008	1.310639	0.392814	out of memory	0.237405	0.130819
Two-well oscillator	**0.000228**	0.000797 ± 0.000917	0.143589	0.691488	0.069024	0.922103	0.026983	0.044469
Duffing oscillator	**0.000664**	0.001312 ± 0.001979	0.334476	0.035095	0.060617	0.969042	0.018989	0.025465
Moving average	0.437374	**0.429657 ± 0.001857**	0.576228	out of memory	0.623362	0.639473	0.621087	0.549476
Nonstationary	0.990199	**0.987444 ± 0.000661**	9.635930	11.436295	11.922808	6.217075	3.849478	2.985249
Random walk	**1.005863**	1.035739 ± 0.016835	85.055414	85.62627611	84.276265	18.526130	2.652366	2.975689
Crude oil prices	**3.944808**	5.315421 ± 2.542140	82.815470	83.935678	34.773389	42.148958	24.807509	19.414582
ECG	9.214228	**8.297350 ± 1.065379**	171.044222	171.683952	155.940562	170.422602	22.290874	79.268606
Exchange rate	0.000662	**0.000661 ± 0.000001**	0.873554	out of memory	1.032494	0.123883	0.003196	0.085444
Gas prices	7.826223	**6.904599 ± 0.033701**	132.514348	308.332609	126.874242	302.017050	47.653068	50.064554
Human speech	0.031606	**0.031522 ± 0.000969**	0.066097	0.066287	0.066290	0.066231	0.052148	0.046229
Macroeconomics	**543.892692**	574.948945	822.114732	1755.364204	1752.439804	1754.180294	783.813518	890.018011
Microeconomics	**170.059750**	170.724059 ± 0.218171	9485.948473	9756.847416	4086.551203	out of memory	4877.916917	430.323863
Music	0.028995	**0.027476 ± 0.001178**	0.102462	0.120091	0.100731	0.095091	0.040063	0.038179
Tropical sound	0.006212	**0.006211 ± 0.000013**	0.031290	0.034240	0.031616	0.032349	0.007544	0.008509
Zooplankton	**257.486771**	264.522817 ± 0.686007	523.959632	315.212572	523.023167	526.726338	393.186004	274.289924

Table 5.5: *RMSE* of time-series AutoML approaches (W_ens, W_single, T10, T200 and WT) on test set. The bold-faced entries indicate the lowest mean error achieved on a given data set, and ∗ indicates the results are statistically tied to the best.

Dataset \ Model	Best Baseline	Best AutoML	Best DNN	W_ens	W_single	T10	T200	WT
Autoregre	0.400718	0.400603 ± 0.000086	0.457445	0.400442 ± 0.000116	**0.400326 ± 0.000216***	0.400967 ± 0.000209	0.416753 ± 0.000453	0.405852 ± 0.001713
Correlated noise	1.027496	1.021476 ± 0.000938	1.187226	**1.021233 ± 0.000232***	1.021779 ± 0.000907	1.021374 ± 0.001513	1.039157 ± 0.007508	1.029969 ± 0.004839
Lorenz Attractor	0.004888	0.000646 ± 0.000007*	0.559788	0.006572 ± 0.003488	**0.000645 ± 0.000011***	0.009944 ± 0.003923	0.003098 ± 0.013171	0.004539 ± 0.012600
Pendulum	0.002337	0.001574 ± 0.000106	0.434611	0.001873 ± 0.001653	0.000331 ± 0.000128	0.003196 ± 0.000981	0.000564 ± 0.000780	**0.000325 ± 0.001430***
Driven oscillator	0.005747	0.012259 ± 0.004473	0.392814	0.011464 ± 0.001179	0.009696 ± 0.005196	**0.000009 ± 0.000000***	0.018824 ± 0.001845	0.017635 ± 0.000000
Two-well oscillator	0.000228	0.000797 ± 0.000917	0.069024	0.000599 ± 0.000520	**0.000024 ± 0.000020***	0.001699 ± 0.000173	0.003017 ± 0.009952	0.002817 ± 0.000000
Duffing oscillator	0.000664	0.001312 ± 0.001979	0.035095	0.001016 ± 0.000796	**0.000024 ± 0.000001***	0.001667 ± 0.000239	0.005959 ± 0.006028	0.003995 ± 0.000000
Moving average	0.437374	0.429657 ± 0.001857	0.576228	0.426285 ± 0.000820	**0.424858 ± 0.000513***	0.442479 ± 0.000398	0.437258 ± 0.000875	0.431531 ∓ 0.001732
Nonstationary	0.990199	0.987444 ± 0.000661	6.217075	0.986342 ± 0.000161	**0.986101 ± 0.000155***	0.987948 ± 0.004381	0.998988 ± 0.007488	1.098608 ± 0.125839
Random walk	**1.005863**	1.035739 ± 0.016835	18.526130	1.052511 ± 0.030777	1.008023 ± 0.000529	1.063534 ± 0.102310	1.078497 ± 0.153690	1.950614 ± 0.974333
Crude oil prices	3.944808	5.315421 ± 2.542140	42.148958	**3.886308 ± 0.031651***	3.983932 ± 0.057481	11.933065 ± 2.464177	15.633299 ± 0.918677	10.546556 ± 0.855509
ECG	9.214228	**8.297350 ± 1.065379***	155.940562	9.004173 ± 1.221086	28.534455 ± 28.28127	11.331525 ± 1.728555	118.053223 ± 25.044242	13.937387 ± 0.000000
Exchange rate	0.000662	0.000661 ± 0.000001	0.123883	**0.000660 ± 0.000001***	0.000661 ± 0.000001	0.000672 ± 0.000012	0.000673 ± 0.000001	0.000668 ± 0.000001
Gas prices	7.826223	6.904599 ± 0.033701	126.874242	**6.762940 ± 0.142239***	7.064739 ± 0.021371	14.596149 ± 4.549673	45.312210 ± 43.414354	28.657329 ± 54.972346
Human speech	0.031606	0.031522 ± 0.000969	0.066097	**0.030287 ± 0.000775***	0.032386 ± 0.001638	0.031206 ± 0.000338	0.030965 ± 0.000266	0.035256 ± 0.000000
Macroeconomics	543.892692	574.948945	822.114732	707.232328 ± 37.759255	731.960238 ± 55.241527	605.267702 ± 5.352145	709.229888 ± 19.251710	**319.126603 ± 294.645363***
Microeconomics	**170.059750**	170.724059 ± 0.218171	4086.551203	170.923711 ± 0.436128	171.002163 ± 0.292934	395.319842 ± 130.523542	395.319842 ± 130.523542	171.298118 ± 0.324284
Music	0.028995	0.027476 ± 0.001178	0.095091	**0.027033 ± 0.000462***	0.027299 ± 0.001342	0.032779 ± 0.000344	0.028302 ± 0.001138	0.029387 ∓ 0.001359
Tropical sound	0.006212	0.006211 ± 0.000013	0.031616	0.006171 ± 0.000025	**0.006166 ± 0.000019***	0.006281 ± 0.000013	0.006465 ± 0.000023	0.006162 ± 0.000009
Zooplankton	257.486771	264.522817 ± 0.686007*	315.212572	263.129431 ± 3.427387	267.354421 ± 0.324754	271.026251 ± 6.203163	273.320260 ± 1.963367	**255.726081 ± 2.978330***

BEATS10, N-BEATS200)?

In Table 5.4, we show the test set *RMSE* obtained for each data set using our deep neural networks baselines, along with the lowest error achieved by simple baseline models and baseline AutoML techniques. Among LSTM, GRU and N-BEATS baselines, the results indicate that N-BEATS shows lower $RMSE$ than LSTM and GRU on 19 out of 20 data sets. We observe that the deep neural network baselines were outperformed by the best AutoML results, followed by the best simple baseline. In our experiments, we noticed that for longer window sizes, the optimisation process terminated due to memory limitations. This demonstrates the complexity and difficulty of optimising deep neural network architectures for time-series data. We suspect that better results could be obtained with automatically designed LSTM/GRU architectures, but likely at the cost of substantially higher resource usage (computing time and memory). Compared to LSTM10 and GRU10, Auto-Keras10 achieves better performance on all the data sets. Similarly, when compared to LSTM200 and GRU200, Auto-Keras200 performs better on 19 out of 20 data sets. This shows that neural network architectures that are known to be suitable for modelling the dynamics of time-series data, such as LSTMs and GRUs, are too expensive to be effectively optimised automatically. Given the same amount of resources, when using Auto-Keras with its original search space, other types of architectures (e.g., dense layers) can also be selected and may offer better results. These results also show that there is still room for more complex and efficient approaches for neural architecture search for time-series data. Compared to N-BEATS10, Auto-Keras10 achieves higher accuracy on 17 out of 20 data sets; Compared to N-BEATS200, Auto-Keras200 outperforms on 13 out of 20 data sets. N-BEATS has been considered one of the state-of-the-art methods in the time-series forecasting domain. We see a big potential in automatically generated models for time-series forecasting. Current AutoML approaches (vanilla auto-sklearn: VA10, VA200, and Auto-Keras: Auto-Keras10, Auto-Keras200) beat deep neural network baselines (LSTM10, LSTM200, GRU10, GRU200, N-BEATS10, N-BEATS200) on all the 20 data set.

Q3: To what extent can our newly proposed approaches beat current AutoML approaches (vanilla auto-sklearn: VA10, VA200, and Auto-Keras: Auto-Keras10, Auto-Keras200)?

In Table 5.5, we compare the performance achieved by our newly proposed approaches against that of the best AutoML baselines. For 18 out of the 20 data sets, at least one of our newly proposed approaches outperforms the best baseline Au-

toML approach, with improvements ranging between 0.02% and 136 111.11% in terms of *RMSE*. Among the five proposed variants, W_ens and W_single show the lowest *RMSE* on 13 out of 20 data sets, while T10, T200 outperform the rest only on one data set (Driven oscillator). WT outperforms the rest only on 3 data sets (Pendulum, Macroeconomics, Zooplankton). W_ens, W_single, T10, T200, WT outperform the best AutoML baselines (Auto-Keras10, Auto-Keras200, VA10, VA200) on 14, 12, 3, 2 and 4 data sets, respectively.

Overall, next to observing improvements against the best baseline AutoML approaches, a comparison of the performance of our newly proposed approaches with each other suggests that W_ens and W_single tend to perform better than the rest in terms of performance on test data.

Q4: Which approach generalises best from training to test data?

To understand which method generalises best, we compare the training and test performance results (training set results in terms of *RMSE* are presented in Table 5.6). A method that generalises well should show similarly low prediction error in both cases. Comparing the training and test set errors, it is evident that the approaches with tsfresh features (T10, T200) suffer from overfitting on training data: these approaches perform better than VA10 and W_ens, W_single, WT on 13 out of 20 data sets in terms of the *RMSE* on the training set, while they only show better performance on the test set on one data set. The approaches with automated window size selection (W_ens, W_single or WT) show the highest accuracy on 7 out of 20 data sets in terms of the *RMSE* on the training set (mainly outperformed by T10 and T200), while the corresponding results for test set performance is much higher (16 out of 20). This suggests that these approaches are more robust against overfitting. W_ens is most of the time better than VA10 in training but not as good as T variants (T10, T200).

Figure 5.3 and Figure 5.4 show the *RMSE* distributions obtained from bootstrap sampling over multiple independent runs of our proposed AutoML variants for two synthetic and two real-world benchmarks. These data sets have been selected such that the difference between approaches is clearly visible. As seen in Figures 5.3(a), 5.3(c), 5.4(a), among the AutoML variants, W_ens shows the lowest median and the lowest test set error followed by VA10, while WT performs best only in Figure 5.4(c). Looking at training error (Figures 5.3(b), 5.3(d), 5.4(b), 5.4(d)), we find that in 3 out of the 4 cases, T10 shows a clear advantage over other approaches; however, similar performances are not achieved on the corresponding test sets. WT does not perform very well either and shows a low performance on the test set. This could be the result

Table 5.6: *RMSE* on the training set acquired from different AutoML methods including vanilla auto-sklearn VA10 and our proposed variants (W_ens, W_single, T10, T200, WT). The bold-faced entries indicate the lowest mean error achieved on a given data set, and $*$ indicates the results are statistically tied to the best.

Dataset \ Model	VA10	W_ens	W_single	T10	T200	WT
Autoregre noise	0.404725 ± 0.000028	0.357100 ± 0.018618	0.404948 ± 0.002718	0.312280 ± 0.019081	**0.099848 ± 0.059389***	0.309739 ± 0.000000
Correlated noise	0.953938 ± 0.018389	0.821940 ± 0.028382	0.988425 ± 0.001885	0.582574 ± 0.029850	**0.445290 ± 0.021132***	0.725874 ± 0.000000
Lorenz Attractor	0.003643 ± 0.002108	0.004787 ± 0.001795	**0.000687 ± 0.000001***	0.005257 ± 0.000979	0.001684 ± 0.004996	0.033001 ± 0.000000
Pendulum	0.013990 ± 0.022468	0.001807 ± 0.001262	**0.000143 ± 0.000019***	0.001046 ± 0.000127	0.001172 ± 0.003045	0.008570 ± 0.000000
Driven oscillator	0.003357 ± 0.001745	0.003524 ± 0.000898	**0.002885 ± 0.000953***	0.003180 ± 0.000028	0.015826 ± 0.003702	0.013420 ± 0.000000
Two-well oscillator	0.000519 ± 0.000356	**0.000406 ± 0.000252***	0.000725 ± 0.000494	0.000789 ± 0.000015	0.004617 ± 0.005851	0.001773 ± 0.000000
Duffing oscillator	0.000679 ± 0.000463	**0.000575 ± 0.000216***	0.002424 ± 0.000738	0.000691 ± 0.000042	0.001937 ± 0.001342	0.002474 ± 0.000005
Moving average	0.413227 ± 0.003247	0.372342 ± 0.002935	0.397426 ± 0.000732	0.293087 ± 0.036759	**0.261658 ± 0.059058***	0.405050 ± 0.000000
Nonstationary	0.960435 ± 0.006790	0.941951 ± 0.012253	0.994682 ± 0.000214	**0.612182 ± 0.076556***	0.629782 ± 0.128813	1.151766 ± 0.000000
Random walk	0.994218 ± 0.004097	0.993145 ± 0.003799	1.008242 ± 0.000831	**0.544931 ± 0.114882***	0.637171 ± 0.141287	1.692634 ± 0.000000
Crude oil prices	1.862273 ± 0.019859	1.787196 ± 0.039135	1.877526 ± 0.004829	1.486114 ± 0.069461	**0.657657 ± 0.148479***	1.854731 ± 0.000000
ECG	6.345342 ± 0.230522	5.609683 ± 0.217275	**5.555471 ± 0.597214***	12.434153 ± 6.124225	12.598475 ± 3.335467	7.246352 ± 0.000000
Exchange rate	0.000762 ± 0.000014	0.000753 ± 0.000008	0.000761 ± 0.000003	0.000751 ± 0.000018*	**0.000734 ± 0.000036***	0.000797 ± 0.000000
Gas prices	2.109257 ± 0.003033	2.037483 ± 0.019962	2.103643 ± 0.002152	1.936833 ± 0.089270	**1.268692 ± 0.212155***	2.023632 ± 0.010646
Human speech	0.025226 ± 0.002773	0.025127 ± 0.002285	0.028316 ± 0.006866	0.020102 ± 0.002484	**0.006871 ± 0.001208***	0.033939 ± 0.000146
Macroeconomics	332.182737 ± 26.603055	137.694269 ± 29.074179	296.448896 ± 121.080645	161.153074 ± 47.609146	130.189798 ± 24.135087	**112.574887 ± 0.277946***
Microeconomics	112.433023 ± 0.060023	111.805190 ± 0.198384	112.400226 ± 0.170035	457.133087 ± 472.235290	**101.604636 ± 1.238983***	112.605469 ± 0.320563
Music	0.054124 ± 0.002594	0.025809 ± 0.002242	0.027113 ± 0.001405	**0.025053 ± 0.003775***	0.025895 ± 0.000817	0.064743 ± 0.000000
Tropical sound	0.005904 ± 0.000122	0.004962 ± 0.0001514	0.003926 ± 0.000523	0.003967 ± 0.000932	**0.003253 ± 0.000454***	0.005707 ± 0.000004
Zooplankton	312.205839 ± 2.797241	265.446350 ± 18.170189	315.313539 ± 3.728371	**109.159209 ± 9.050114***	173.214940 ± 12.636791	176.475896 ± 9.671782

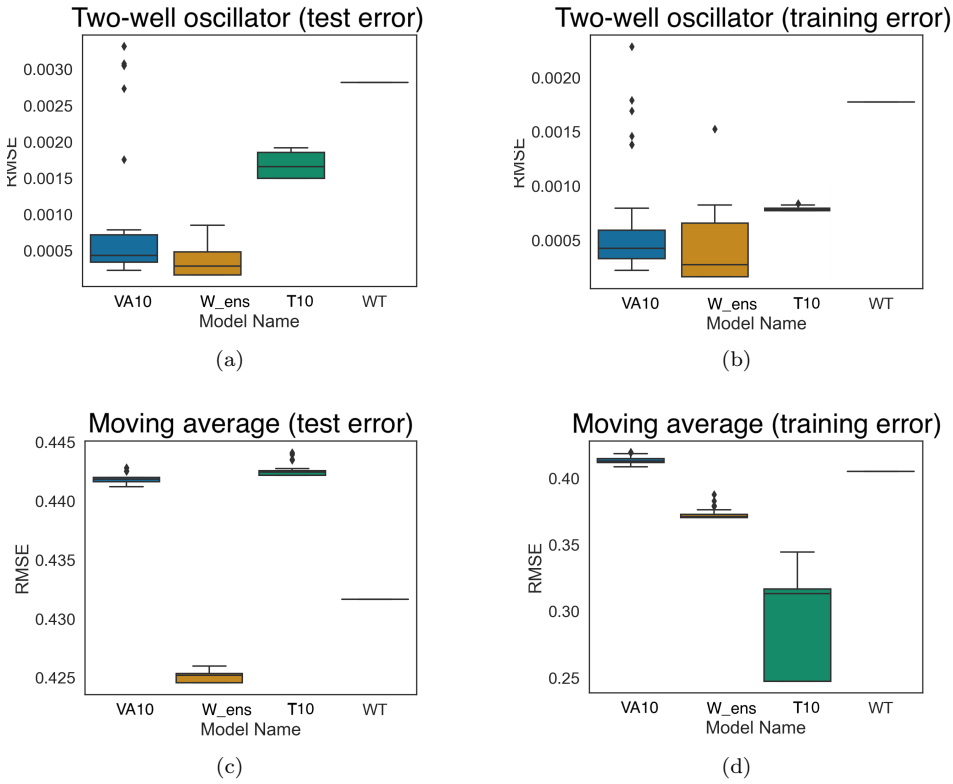

Figure 5.3: *RMSE* distribution of the bootstrapping results of the AutoML method runs on the Two-well oscillator, Moving average data sets with models VA10, W_ens, T10, WT. The distribution is created by the bootstrapping protocol we mentioned in Section 5.5.

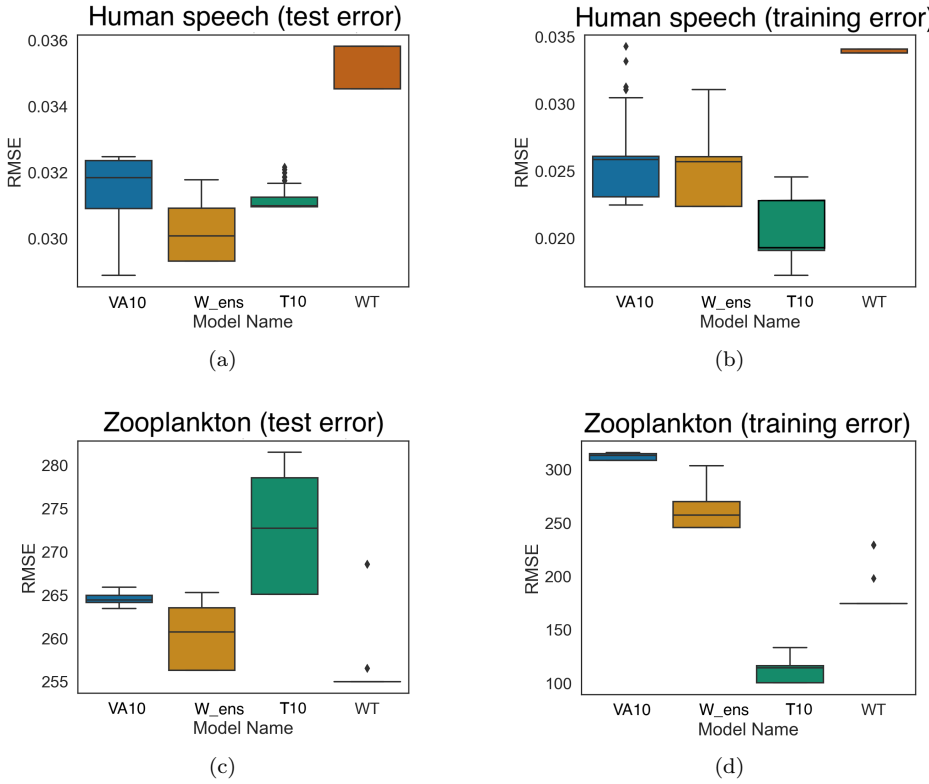

Figure 5.4: *RMSE* distribution of the bootstrapping results of AutoML method runs on the Human speech, and Zooplankton data sets with models VA10, W_ens, T10, WT. The distribution is created by the bootstrapping protocol we mentioned in Section 5.5.

of the fact that the search space is considerably more complex. Using the same time budget as the other variants, WT can evaluate much fewer configurations than other approaches, since significant time has to be spent on extracting features.

These results generally suggest that while the tsfresh features can indeed help in training machine learning models for time-series forecasting, their use can easily lead to overfitting. We believe this is caused by the relatively large number of features (794 time-series features total before performing feature selection) compared to the number of data instances available for training. During training, we used the Benjamini-Hochberg procedure for feature selection (Benjamini and Hochberg, 1995) in both tsfresh and auto-sklearn in order to avoid overfitting. After feature selection, between 26 to 466 features were selected in T10 and T200 experiments (depending on the data set). However, this approach still could not prevent the overfitting issue.

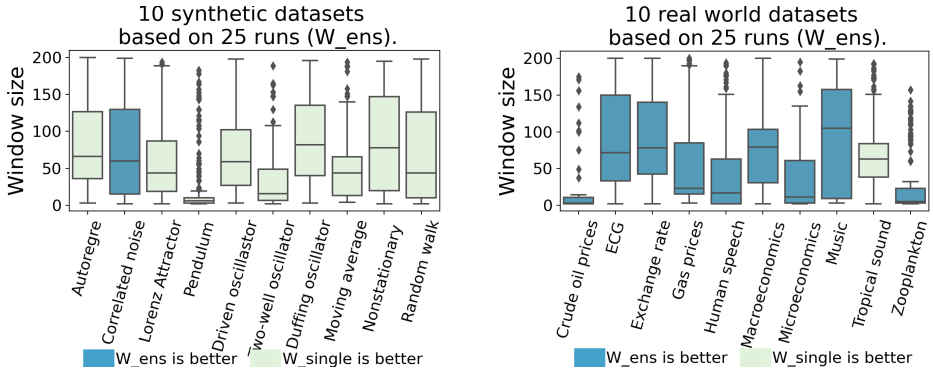

Figure 5.5: The window size distribution of W_ens on (a) 10 synthetic datasests and (b) 10 real data sets. Distributions of window sizes are acquired from 25 runs. Color codes represent comparative performance based on the test set *RMSE* presented in Table 5.5.

Q5: Can optimal window sizes for time-series forecasting be found by the AutoML system?

Figure 5.5 and Figure 5.6 visualise the distribution of window sizes found for synthetic and real-world data sets in 25 runs, with (W_ens) and without ensembling (W_single) and compares their performances based on *RMSE* on test sets presented earlier in Table 5.5. We first investigated whether an optimal window size has been found for each data set. A distribution with low variance will indicate the stability of the range of these window sizes. Figures 5.5 show the window size distributions for

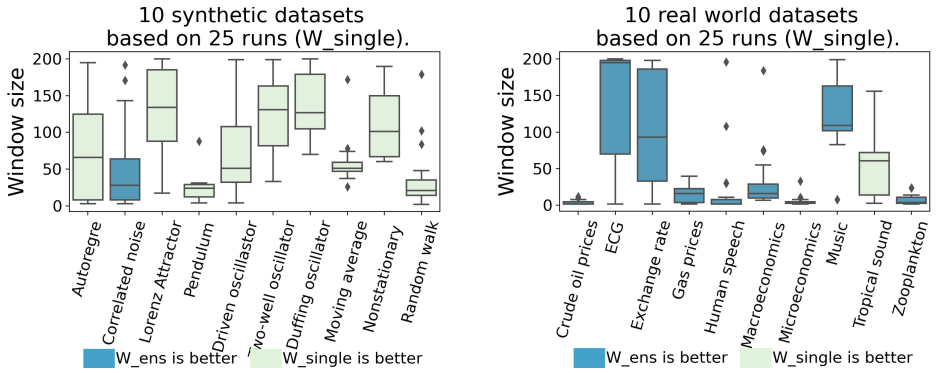

Figure 5.6: The window size distribution of W_single on (a) 10 synthetic data sets and (b) 10 real data sets. Distributions of window sizes are acquired from 25 runs. Color codes represent comparative performance based on the test set *RMSE* presented in Table 5.5.

W_ens. It is generally seen that the ensemble models have selected a wide variation of window sizes. We believe that this is due to the fact that a time-series can be decomposed into multiple signals with different frequency components (according to the Fourier transform). Each frequency component can be represented as values in windows with different sizes. Ensemble models can consider the contribution of each of these components in forecasting. Figures 5.6 show the window size distributions for W_single. As expected, W_single models tend to have less varied window size ranges than W_ens and considerably lower numbers of outliers.

Looking at these figures, we also observed that in most cases W_single tends to yield better performance than W_ens on synthetic data sets, while W_ens yields better performance on real-world data sets. This can be explained by looking at the data sets in Figure 5.7 and Figure 5.8. In Figures 5.7 and 5.8 (a),(c) we present 2 samples of each data set category that had lower variance in Figures 5.6 along with their first 200 samples (Figures 5.7 and 5.8 (b),(d)). Clearly, real-world data sets are more complex. Furthermore, in these data sets, the relation between different data points in terms of slow and fast-changing patterns (i.e., low and high-frequency components) is visible. This explains why W_ens performs best on these data sets. The synthetic data sets we used, on the other hand, are either periodic or do not show any complex and non-stationary behaviour. This explains why a single window size that is large enough to capture the temporal patterns yields better results. Since the goal of the AutoML system should be to perform well on real-world data sets, W_ens should be preferred

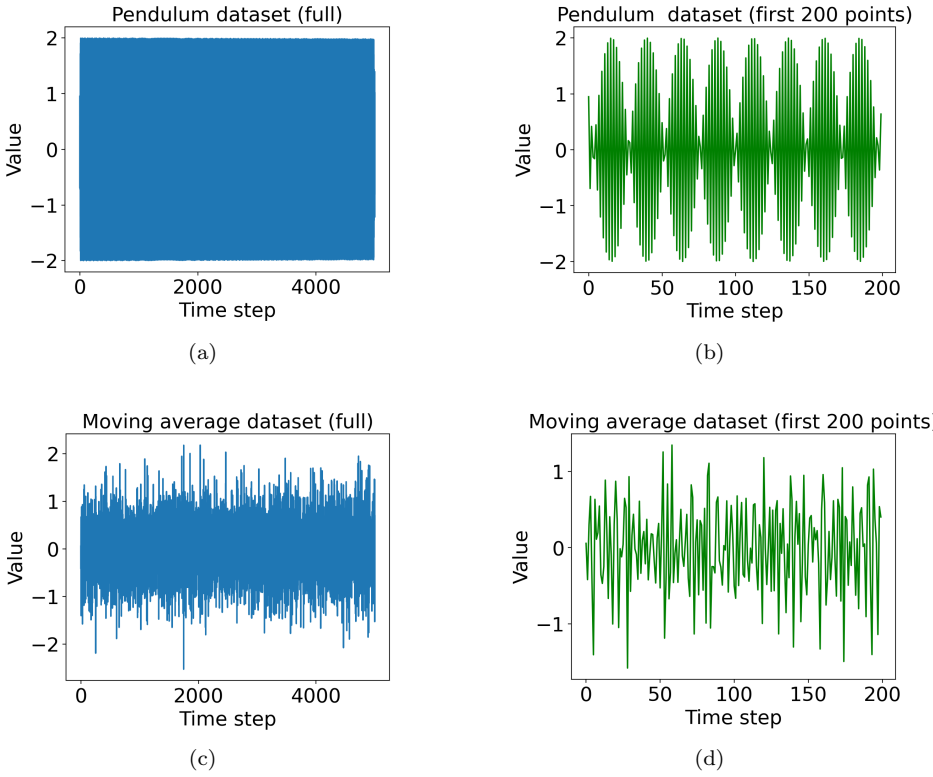

Figure 5.7: Visualization of Pendulum, Moving average, Human speech, and Microeconomics data sets: (a) Pendulum data set (b) the first 200 points of Pendulum data set (c) Moving average data set (d) the first 200 points of Moving average data set.

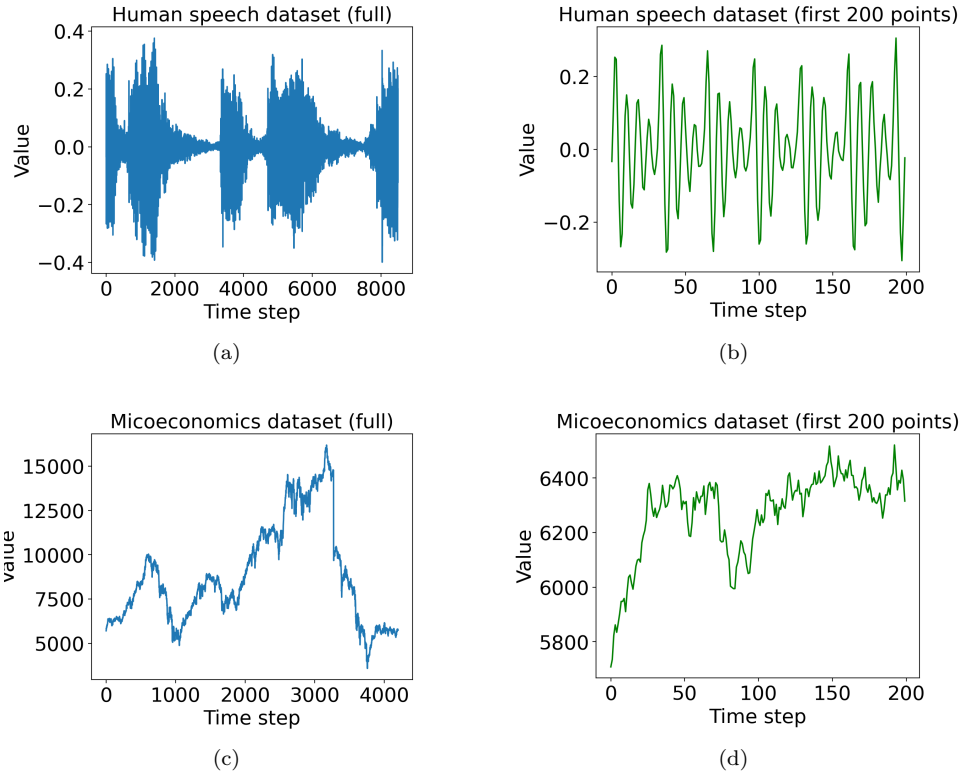

Figure 5.8: Visualization of Pendulum, Moving average, Human speech, and Microeconomics data sets: (a) Human speech data set (b) the first 200 points of Human speech data set (c) Microeconomics data set. (d): the first 200 points of Microeconomics data set.

over W_single as the ensembling method can capture different periodic components of the time-series in different window sizes.

5.6 Conclusion

In this chapter, we investigated the realisation of AutoML systems for time-series forecasting. We proposed three variants of AutoML systems based on auto-sklearn and tsfresh for time-series forecasting and compared them to simple statistical, machine learning baselines, and AutoML baselines. We found that vanilla AutoML can perform very well on time-series forecasting tasks, achieving significantly higher accuracy than the simple statistical and machine learning baselines on 9 out of 20 data sets in terms of *RMSE*. Compared to deep neural network baselines, AutoML baselines and our newly proposed approaches perform better on all the data sets in terms of *RMSE* on test sets. However, it is likely that better results can be obtained using automatically designed LSTM/GRU architectures at the expense of much higher resource usage (computing time and memory). The state-of-the-art model N-BEATS has been used in our experiments. We found that although N-BEATS beats our LSTM/GRU baselines on most of the data sets, Auto-Keras baselines and vanilla auto-sklearn baselines still perform better than it on most of the data sets. With the same computational resources, our enhanced approaches yield better results than vanilla AutoML on most of our data sets (on 18 out of 20 benchmarks in terms of *RMSE* on the test sets). Overall, our methods improved accuracy for 17 out of 20 data sets compared to our simple baselines and current AutoML baselines. In terms of *RMSE* on training sets, our new approaches outperform vanilla auto-sklearn on all 20 data sets.

We also found that the window size has a considerable impact on the accuracy of time-series forecasting and that by optimising the window size, we could achieve significantly higher performance than current AutoML techniques. Our experiments show that, while tsfresh features can be useful in terms of improving forecasting accuracy, their use in combination with an AutoML system often leads to overfitting. Our empirical results further indicate that our enhanced AutoML variant with windows size selection works best in connection with ensembling when tackling forecasting tasks on complex real-world data sets. We found that models with longer input window sizes do not always perform better than those with shorter input window sizes, which further proves that the window size selection is essential in time-series forecasting.

Overall, these results clearly demonstrate that the use of AutoML techniques for time-series forecasting is highly promising. Interesting avenues for future work include

the development of techniques for detecting and preventing overfitting as well as the use of AutoML for other tasks involving time-series data, such as multi-step forecasting and time-series classification.