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

Wang, C.

Chapter 6

AutoML for multi-step time-series forecasting

6.1 Introduction

In the previous chapter, we investigated the use of AutoML for constructing machine learning models for single-step time-series forecasting tasks. Different from the previous chapter, in this chapter we aim to answer the research question: How can AutoML be used for multi-step time-series forecasting? Feature engineering is an essential step in the pipelines used for many machine learning tasks, including time-series forecasting. Although existing AutoML approaches partly automate feature engineering, they do not support specialised approaches for applications on time-series data such as multi-step forecasting. Multi-step forecasting is the task of predicting a sequence of values in a time-series. Two kinds of approaches are commonly used for multi-step forecasting. A typical approach is to apply one model to predict the value for the next time step. Then the model uses this predicted value as an input to forecast the value for the next time step. Another approach is to use multi-output models to make the predictions for multiple time steps of each time-series directly.

This chapter presents a study of AutoML for time-series multi-step time-series forecasting implemented through (i) multi-output modeling, and (ii) recursive modeling. We combine the state-of-the-art AutoML system of auto-sklearn with tsfresh, a well-known library for feature extraction from time-series. We implement three AutoML variants and state-of-the-art baseline models, including auto-sklearn, support vector

machines (SVMs), gradient boosting machine (GBM), N-BEATS, and Auto-Keras. More specifically, our contributions are as follows:

- We adapt the auto-sklearn AutoML system to the task of multi-step forecasting and introduce three AutoML forecasting variants for multi-step-ahead time-series forecasting.
- We demonstrate the importance of feature selection and window size selection in multi-step forecasting problems and further show that by incorporating such approaches, our time-series AutoML techniques outperform available AutoML methods.
- We evaluate our methods on 20 benchmarking data sets from 20 different categories and against the baselines. We found that our proposed AutoML method outperformed the traditional machine learning baseline in 14 out of 20 data sets and N-BEATS on 15 out of 20 data sets.

The remainder of this chapter is structured as follows: Section 6.2 covers related work on time-series forecasting, time-series feature engineering and AutoML. Section 6.3 introduces the problem statement of AutoML for multi-step time-series forecasting. In Section 6.4, the methodology of our newly proposed models is explained. Section 6.5 presents the results of the empirical performance comparison of different machine learning models for multi-step time-series forecasting. A summary of our work and directions for future work are given in Section 6.6

6.2 Related work

Multi-step time-series Forecasting: different machine learning methods and statistical methods have been used for both single-step and multi-step time-series analysis in the past few decades (Hong and Fan, 2016). These include artificial neural networks (Chen et al., 2018b), support vector machines (Nie et al., 2012), gradient boosting machine (Li et al., 2020b), random forest (Candanedo et al., 2017), auto-regressive moving average models (Box et al., 2015), and exponential smoothing models (Nedellec et al., 2014). To use these models for multi-step forecasting, two major approaches are used: direct and recursive strategies (Taieb et al., 2012). The first of these uses multi-output regression models to predict multiple time-series steps into the future directly or use multiple models (one for each time step) to make multi-step forecasting. Multi-output

models show good performance and require less computational resources than training multiple models to realise multi-step forecasting (Ferreira and da Cunha, 2020). The second approach uses a single model recursively, using the predicted values as additional input to forecast the next step. In this case, the error in the prediction may be accumulated (Taieb et al., 2012). Recursive strategies only require one model, which saves significant computational time (Taieb et al., 2012). Other methods based on these two approaches are also used, such as the DirRec strategy (Sorjamaa and Lendasse, 2006), which combines the recursive strategies and direct strategies.

Time-series feature engineering: We have previously reviewed related work in time-series feature engineering in Chapter 5.4.2. We use tsfresh in our experiments in this chapter.

AutoML: AutoML systems have been used in many domains, such as image classification (Zoph et al., 2018), language processing (Bisong, 2019) and energy consumption forecasting (Wang et al., 2019). However, time-series features are not well-studied in AutoML systems. In our earlier work (Wang et al., 2019), we studied AutoML for short term load forecasting demonstrating the competitive performance of AutoML systems. However, we did not investigate the use of time-series features. In another work (Wang et al., 2023), we studied AutoML with time-series features for single-step time-series forecasting. Our Empirical results indicate that AutoML with time-series features can further improve the accuracy of AutoML systems. In this chapter, we focus on studying how to improve the performance of AutoML systems on multi-step forecasting tasks by using time-series features. Many AutoML systems have been recently developed, including Auto-sklearn (Feurer et al., 2015), AutoGluon (Shi et al., 2021) and Auto-Keras (Jin et al., 2019) . Auto-sklearn is used in our experiments since it supports various algorithms (e.g., SVMs) that are not available in the other systems, and it is easy to extend. Auto-Keras is used to create a deep learning baseline in our experiments.

6.3 Problem statement

Given a univariate time-series $\mathbf{x} = [x_1, \ldots, x_i]$ composed of i observations. We are interested in predicting the next k values $\mathbf{x} = [x_{i+1}, \ldots, x_{i+k}]$, where $k > 1$ denotes the forecasting window. Usually, not all the data points show the same influence on the predictions of $[x_{i+1}, \ldots, x_{i+k}]$. The more recent data points tend to be more important. Specifically, given a time-series segment $[x_{i-w+1}, \ldots, x_i]$, we are interested in forecasting of $[x_{i+1}, \ldots, x_{i+k}]$; the window size w indicates how much historical data

are used to make the prediction. In the previous chapter, we showed that the window size w plays an essential role in single-step time-series forecasting; specifically, if a machine learning model gets too much or too little information, this may reduce the model's performance. Here, we extend the automated window size selection technique to multi-step forecasting. Besides this, we use tsfresh to automatically extract features from time-series data within these windows.

6.3.1 AutoML for multi-step time-series forecasting

In the previous chapter, we define the *Combined Algorithm Selection and Hyperparameter optimisation (CASH)* problem (Thornton et al., 2013) for single-output time-series forecasting as the joint optimisation problem 5.1. In this chapter, we focus on multi-step time-series forecasting tasks. The problem definition stays the same but the loss is different, instead of calculating the loss based on the predicted value of every time step, in the multi-step time-series forecasting tasks, we have to calculated it by the whole predicted vector, which contains the predicted value from the next 1 to k time steps.

6.4 Methodology

This section presents the two multi-step forecasting techniques and the AutoML technique enhanced with time-series features we use later in our experiments.

6.4.1 Multi-step forecasting

Recursive strategy: In this strategy, given a univariate time-series $\mathbf{x} = [x_1, \ldots, x_n]$ composed of n observations, a model f is trained to perform a single-step ahead forecast: $\hat{x}_{i+1} = f(x_{i-w+1}, ..., x_i)$ with $i \in \{w, ..., n-1\}$. Then we use $\hat{x}_{i+1}$ as an input to predict x_{i+2}. $\hat{x}_{i+2} = f(x_{i-w+2}, ..., x_i, \hat{x}_{i+1})$ with $i \in \{w, ..., n-1\}$. We continue recursively, making new predictions in this manner until we forecast x_{i+k}.

Direct Multi-Output strategy: A multi-output strategy has been proposed by (Taieb and Bontempi, 2011) to solve multi-step time-series forecasting tasks. In this strategy, one multi-output model f is learned, i.e., $[\hat{x}_{i+1}, ..., \hat{x}_{i+k}] = f(x_{i-w+1}, ..., x_i)$ with $i \in \{w, ..., n-k\}$.

6.4.2 Auto-Sklearn with time-series feature engineering

In this chapter, we study how we can extend auto-sklearn (Feurer et al., 2015) to perform automatic feature extraction on time-series data. Originally, the pipelines constructed by auto-sklearn include a preprocessor, feature preprocessor, and machine learning components. The ensemble construction used in auto-sklearn uses a greedy algorithm to build the ensembles. The workflow of auto-sklearn is illustrated in Figure 5.1. Auto-sklearn has a powerful feature preprocessor component. However, it does not support any specialised time-series feature extractors. In our work, we use our newly proposed time-series feature extractors in the search space instead of the feature extractor in auto-sklearn. Automated feature extraction in this case considers both the selection of the window size and the extraction of relevant features. Therefore, we propose three variants of auto-sklearn that are specially designed for time-series forecasting tasks by replacing the feature extractors of auto-sklearn with one of the following.

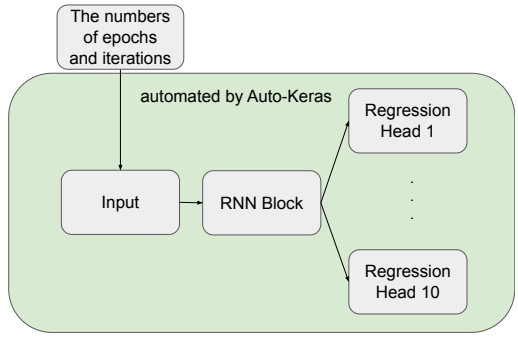

Figure 6.1: Workflow of our customised search space of Auto-Keras for time-series forecasting. The input data flows through the RNN Block. Hyperparameters, such as the number of layers and learning rate, will be optimised during the search. The Regression Head then generates output based on the information from the RNN Block.

1. **Auto-sklearn with automated window size selection (W):** the first variant of auto-sklearn for time-series forecasting optimises the window size w. The time-series $\mathbf{x} = [x_{i-w+1}, \ldots, x_i]$ are used to train a model that predicts $[x_{i+1}, \ldots, x_{i+k}]$.

2. **Auto-sklearn with tsfresh features (T):** the second variant of auto-sklearn extracts tsfresh time-series features from the time-series segment $\mathbf{x} =$

$[x_{i-w+1}, \ldots, x_i]$ to predict $[x_{i+1}, \ldots, x_{i+k}]$. In this case, the window size w is predefined and fixed. The time-series features $g(\mathbf{x}) = g([x_{i-w+1}, \ldots, x_i])$ are calculated using the time-series feature extractor g. Feature importance is calculated using the Benjamini-Hochberg procedure (Benjamini and Hochberg, 1995) to select the important features. The Benjamini-Hochberg procedure selects important features for each step in the time-series separately. We then use the union of all the important features to predict $[x_{i+1}, \ldots, x_{i+k}]$.

3. **Auto-sklearn with automated window size selection and tsfresh features (WT):** this approach combines the two previously mentioned approaches. Both window size w and the time-series extractor g are optimised in this variant.

6.5 Empirical results

Our key empirical results are based on aggregate performance over 20 data sets and 8 models. More detailed descriptions of the data sets and models are described in the following.

6.5.1 Data sets

The open-source data sets from CompEngine (Fulcher et al., 2019) are used in our experiments. CompEngine is a time-series data engine containing 197 types of data sets comprising 29514 in total. These data sets include both real-world and synthetic data sets. We chose ten real-world and ten synthetic data sets from different categories. These are comprised of the following 20 categories: Audio: Animal sounds, Human speech, Music; Ecology: Zooplankton growth; Economics: Macroeconomics, Microeconomics; Finance: Crude oil prices, Exchange rate, Gas prices; Medical: Electrocardiography ECG; 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.

Since there are usually more than one data set in each category, we choose the first one of each category. We split every data set into 67% training and 33% test set, based on temporal order, since the data sets are time-series.

Table 6.1: *RMSE* on test set acquired from traditional machine learning baselines. GBM-recursive, GBM-multiout, SVM-recursive, and SVM multioutput win on 6, 6, 0, and 8 out of 20 data sets respectively.

Dataset	RMSE(GBM -recursive)	RMSE(GBM -multioutput)	RMSE(SVM -recursive)	RMSE(SVM -multioutput)
Autoregre noise	**0.458890**	0.461558	0.484516	0.459410
Correlated noise	1.872176	**1.862137**	2.012916	2.004572
Lorenz Attractor	0.102323	**0.088045**	0.188223	0.152384
Pendulum	0.112041	0.104519	0.172118	**0.035350**
Driven oscillator	**0.121606**	0.124701	0.231661	0.224206
Two-well oscillator	0.033950	0.032462	0.075318	**0.007772**
Duffing oscillator	0.025830	0.021330	0.075308	**0.013762**
Moving average	0.629791	0.627176	0.641453	**0.622803**
Nonstationary	6.049796	**5.987631**	6.796246	6.448516
Random walk	**12.766561**	13.690753	30.594553	25.654821
Crude oil prices	28.215008	32.909490	42.278003	**20.867176**
ECG	**79.209558**	103.881034	128.420743	126.1525026
Exchange rate	0.006880	0.006823	0.028571	**0.005433**
Gas prices	102.819893	**100.612148**	166.021626	172.605827
Human speech	0.059365	**0.054838**	0.085002	0.057631
Macroeconomics	779.515969	806.704035	713.073168	**713.363569**
Microeconomics	**647.432403**	705.051879	3500.094238	3865.235605
Music	0.082864	0.076047	0.068341	**0.052978**
Tropical sound	0.009468	**0.006285**	0.034925	0.008820
Zooplankton	**312.033380**	385.377067	319.839856	320.049399

6.5.2 Experimental setup

All the experiments were executed on 8 cores of an Intel Xeon E5-2683 CPU (2.10 GHz) with 10 GB RAM. In the experiments, version 0.8.0 of auto-sklearn and version 0.16.1 of tsfresh were used. To evaluate the quality of a machine learning pipeline, we used quantified error/accuracy. $RMSE$ was used as a performance metric in the optimisation. The maximum evaluation time for one machine learning pipeline was set to 20 min wall-clock time. The time budget for every AutoML optimisation on each data set was set to 3 h wall-clock time. In these experiments, we used hold-out validation (training:validation = 67:33), the default validation technique in auto-sklearn. The split was carried out only on the training data, such that the optimisation process never sees the test data. However, we did not shuffle the data set in order to preserve the temporal structure of the time-series data. All remaining choices were left at their default settings. Since experiments are very time-consuming, we used bootstrapping to create distributions of performance results in order to investigate their variability. Every experiment was run 25 times. We then randomly sampled 5 out of the 25 results and selected the model with the lowest $RMSE$ on the training set out of these five models and reported the $RMSE$ on the test set. We repeated this process 100 times per model and data set. The distributions we showed are based on these 100 values.

We compared the AutoML methods, including Auto-Keras, auto-sklearn and our proposed variants, with traditional machine learning baselines and N-BEATS. Both recursive and multi-output techniques are used in the machine learning baselines (GBM

Table 6.2: *RMSE* on test set acquired from different AutoML methods including vanilla auto-sklearn (VA), our proposed variants (W, T, and WT), Auto-Keras and the state-of-the-art method N-BEATS. The accuracy of N-BEATS, Auto-Keras, VA, W, T, WT are statistically significant on 5, 0, 2, 8, 3, and 3 out of 20 data sets, respectively.

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

and SVM). All other models use the multi-output approach.

6.5.3 Baselines

- **Gradient Boosting Machine (GBM):** Gradient Boosting Machine is a classical machine learning model used for time-series analysis tasks that has shown promising performance in the M3, M4 competitions (Januschowski et al., 2020). For hyperparameter optimisation, we performed a random search on GBM with 30 iterations and window size $w = 100$ (Bergstra and Bengio, 2012). In this case, the search space is the same as the search space of GBM in auto-sklearn. In this experiment, we did not split the training set into the training set and validation sets.

- **Support vector machine (SVM):** SVM is another classical machine learning model that has been used for time-series forecasting (e.g., (Candanedo et al., 2017)). Similar to GBM experiments, we use 30 iterations of random search and window size $w = 100$. The search space is the same as the search space of SVM in auto-sklearn.

- **N-BEATS:** N-BEATS (Oreshkin et al., 2019) uses fully-connected layers with residual links to improve 3% over the winner of the M4 competition, which demonstrates state-of-the-art performance. We used the default hyperparameter settings in the implementation provided by (Oreshkin et al., 2019) and the bootstrapping approach mentioned in Section 6.5 to create distributions of results. The number of epochs was set to 500.

- **Auto-Keras:** Auto-Keras (Jin et al., 2019) is a neural architecture search system that uses Bayesian optimisation to search for high-performance neural network architectures. Some neural network units available in its search space (e.g., LSTM, GRU), have been used for time-series forecasting (see, e.g., (Siami-Namini et al., 2018; Zhang et al., 2017)). Vanilla Auto-Keras does not support multi-output models. To deal with multi-step forecasting tasks, we designed our new search space using three types of blocks available in Auto-Keras: Input block, RNN Block and Regression Head (see Figure 6.1). The RNN Block is a critical component in our networks. We use RNN as a baseline, as it has been recently studied in the literature on time-series forecasting (see, e.g., (Siami-Namini et al., 2018; Yamak et al., 2019)). Several hyperparameters need to be considered for this block, including bidirectionality, the number of layers and

layer type (LSTM or GRU). Auto-Keras cannot choose the window size w automatically. We manually preprocessed the data with window size $w = 100$. We used Bayesian optimisation for architecture search. The number of epochs was set to 100, and we left the remaining settings of Auto-Keras at their default values.

- **Vanilla auto-sklearn (VA):** We manually preprocessed the data with window size $w = 100$ and then fed it to the auto-sklearn. The time budget for the optimisation was set to 3 h.

6.5.4 Our methods

- **Auto-sklearn with automated window size selection (W)**: For the W variant, we did not need to manually preprocess the data, since the window size w is selected automatically. The window size ranges from 50 to 200.

- **Auto-sklearn with tsfresh features (T)**: In the T variant, the time-series feature extractor tsfresh was used as an internal component of auto-sklearn. Auto-sklearn used these time-series features as input data to search over machine learning pipelines. The window size was set to $w = 100$.

- **Auto-sklearn with automated window size selection and tsfresh features (WT)**: In WT, we set the window size w to range from 50 to 200. The time-series features were extracted from these input data.

Tables 6.1 and 6.2 compare the performance achieved by different methods in terms of *RMSE* on the test set. Table 6.1, shows the results for traditional machine learning baseline models, while Table 6.2 presents the results for AutoML techniques and N-BEATS. To present the results in these tables, we calculated the statistical significance of the results by the non-parametric Mann–Whitney U-test (Mann and Whitney, 1947) with a standard significance level set to 0.05. The bold-faced entries show the lowest mean $RMSE$ achieved on a given data set, and the $*$ means the $RMSE$ is statistically best.

6.5.5 Research questions

Q1: How do recursive and multi-output techniques compare in terms of accuracy?

To determine the answer to this question, we compared the recursive and multi-output versions of GBM and SVM algorithms. Among the baselines we consider, N-BEATS as described in the original work is not a recursive model. Therefore, we do not consider it for this analysis. Looking at Table 6.1, we generally observe that GBM-multioutput performs better than GBM-recursive on 12 out of 20 data sets, while SVM-multioutput outperforms SVM-recursive on 17 out of 20 data sets in terms of *RMSE*. As we have observed that multi-output models tend to perform better, which is in line with the results from (Taieb et al., 2012). Therefore, we only use the multi-output technique in our next experiments.

Q2: To what extent can AutoML techniques (Auto-Keras, auto-sklearn, and our variants) beat the traditional baselines (GBM, SVM)?

Looking at Tables 6.1 and 6.2, one can compare the performance achieved by different methods in terms of *RMSE* on the test set. We observe that Auto-Keras beats all the traditional machine learning baseline models (GBM-recursive, GBM-multioutput, SVM-recursive, and SVM-multioutput) on 4 out of 20 data sets. Vanilla auto-sklearn outperforms all the traditional machine learning baselines on 8 out of 20 data sets. Our three variants W, T, WT show lower error than all the traditional machine learning baselines on 10, 5, and 5 out of 20 data sets, respectively. The best AutoML (W) outperforms the best traditional machine learning baseline (SVM-multioutput) on 14 out of 20 data sets.

Q3: To what extent can AutoML techniques beat N-BEATS?

Looking at Table 6.2, we observe that the best AutoML (W) outperforms N-BEATS on 14 out of 20 data sets. For other AutoML techniques, we observe that Auto-Keras, VA, T, and WT beat N-BEATS on 5, 12, 11, and 10 out of 20 data sets, respectively. AutoML methods that are based on standard machine learning are beating this neural networks-based model.

6.6 Conclusion

In this chapter, we extended AutoML for multi-step time-series forecasting with time-series features. We found that AutoML can achieve significantly higher accuracy than the traditional machine learning baselines on 14 out of 20 data sets in terms of *RMSE*. Although N-BEATS performs better than Auto-Keras and vanilla auto-sklearn on

many data sets, our AutoML time-series variants still managed to beat it on 14 out of 20 data sets. We found that the multi-output technique tends to perform better with the same budget than the recursive technique in the multi-step time-series forecasting tasks. Overall, these results clearly demonstrate that the use of AutoML techniques and multi-output strategies for multi-step time-series forecasting is promising.