Expert Systems With Applications 213 (2023) 119023

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Towards understanding the importance of time-series features in automated algorithm performance prediction

Gašper Petelin [a,b,*], Gjorgjina Cenikj [a,b], Tome Eftimov [a]

[a] Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Jamova cesta 39, 1000, Ljubljana, Slovenia

ARTICLE INFO

Keywords:
Time-series forecasting algorithm performance prediction
Time-series analysis
Landscape analysis

ABSTRACT

Accurate and reliable forecasting is a crucial task in many different domains. The selection of a forecasting algorithm that is suitable for a specific time series can be a challenging task, since the algorithms' performance depends on the time-series properties, as well as the properties of the forecasting algorithms. The methodology and analysis presented in this paper are contributing towards understanding the performance of time-series forecasting methods. Instead of using time-series meta-features only to obtain a good meta-model that can predict the performance of a forecasting algorithm, the methodology can link which features are important for which forecasting methods. We used time-series meta-features extracted using the *tsfresh* and *catch22* libraries. We also found that the importance of the meta-features changes depending on the meta-model that is used. There are only a few meta-features that always appear important for a given forecasting method no matter which meta-model will be used for learning, which further provides opportunities to select a model-agnostic feature portfolio. In addition, different feature importance techniques can provide different results that are related to the methodology that is used by the meta-model. By using the feature importance obtained by a meta-model and a specified feature importance technique, we can define a representation of a forecasting method behavior, which can further provide an insight into which forecasting methods have similar behavior.

1. Introduction

Nowadays, a lot of industrial real-world problems involve the analysis of time-series, i.e. chronologically collected data points. Tracking the price fluctuations and price of a security over time in the financial, investment, and business domains, assessing disease risk using longitudinal patient history data in the medical domain, and weather forecasting are only a few examples to be mentioned. Due to their ubiquitous presence in various domains, numerous algorithms have been proposed for time-series forecasting and analysis (Dama & Sinoquet, 2021). Supporting research in this direction, the Makridakis Competitions have been largely responsible for pushing forward the development and evaluation of novel time-series forecasting models by evaluating their strengths, weaknesses, and suitability to different time-series instances, as well as releasing diverse time-series datasets from various fields. These competitions published a dataset of time-series data instances, where different researchers test time-series forecasting algorithms whose performances are further stored. In the end, the algorithm that is the best on average across the test data instances is selected as the winner.

Despite a lot of studies that have been already conducted in time-series forecasting (Deng, Karl, Hutter, Bischl, & Lindauer, 2022), the main challenge is which algorithm to select for each dataset, or even more precisely, for each data instance. This is a well-known problem called algorithm selection (AS), which endeavors to identify one or several algorithms which are well-suited for a given task (Salisu, Abdulrahman, Adamu, Ado, & Rilwan, 2017). The ever-increasing number of Machine Learning (ML) algorithms, along with their potentially infinite hyperparameter search spaces and the choice of constituents of composite methods such as ensembles, produce an exponential number of configuration combinations, which makes AS a challenging problem with high computational costs (Cohen-Shapira & Rokach, 2021).

One way to perform AS is the use of meta-learning (Brazdil, Giraud-Carrier, Soares, & Vilalta, 2009; Vanschoren, 2019), which involves training a ML model to automatically predict the best performing algorithm(s) for an unseen data instance, based on its (meta-) features. The performance of an AS model is related to reliable performance prediction, so automated algorithm performance prediction is a crucial step in an AS pipeline. For this purpose, in this paper, we are focusing on the automated algorithm performance prediction task.

* Corresponding author at: Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, 1000 Ljubljana, Slovenia.
E-mail addresses: gasper.petelin@ijs.si (G. Petelin), gjorgjina.cenikj@ijs.si (G. Cenikj), tome.eftimov@ijs.si (T. Eftimov).

https://doi.org/10.1016/j.eswa.2022.119023
Received 18 June 2022; Received in revised form 1 September 2022; Accepted 9 October 2022
Available online 15 October 2022

Algorithm performance prediction can be performed in different learning scenarios. One way is to merge it with the AS step and treat it as a multi-class classification task, where the goal is to predict a single best algorithm, or treat it as a multi-label classification task, where several algorithms can be selected. The other learning scenarios involve solving a regression task, where the ML model predicts the performance achieved by each algorithm (Talagala, Li, & Kang, 2021), or ranking, where the aim is to rank the algorithms according to some evaluation metric (Cohen-Shapira & Rokach, 2021; Tyrrell, 2020). In the latter scenario, the AS step is performed on top of the regression results. The advantage of developing regression models for automated algorithm performance prediction over classification ones is to track the magnitude of differences between performances of different algorithms, as they predict numerical performance values.

The success and reliability of an automated performance prediction task are conditioned on several aspects, which should be addressed with great care: (i) the quality of data used to train the ML model, (ii) the representation learning method used to extract features to represent the time-series data instances, and (iii) the selection of the ML model that will be used for learning. In this study, the main focus is on the intersection between the second and the third step or providing explanations of which features are the most important for reliable algorithm performance prediction.

Transforming time-series data into informative and interpretable features can be a computationally expensive and challenging task. Extracted features have to capture different dynamic time-series properties that are informative enough to be used in further pipelines like regression, classification (Ruiz, Flynn, Large, Middlehurst, & Bagnall, 2021) or clustering (Eftimov et al., 2022). Multiple such sets of features that define the data instance representation have been proposed, which extract informative features across different domains and applications, the most notable ones being the Canonical Time Series Characteristics (*catch22*) and the representations provided by the Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (*tsfresh*) library. Recently, many representations are proposed that are based on deep neural network representations (i.e. ROCKET (Dempster, Petitjean, & Webb, 2020), MiniROCKET (Dempster, Schmidt, & Webb, 2021)), with the main drawback that they are black-box and limit the opportunity for learning explainable ML models. In Henderson and Fulcher (2021), different techniques and libraries for extracting time-series meta-features are explored and the trade-offs between them are analyzed. The limitation of most of the approaches where time-series meta-features are extracted and used for AS is that the features cannot capture all of the necessary information.

Our contribution: To go beyond this, in this paper, we analyze the use of time-series features on the performance of ML models predicting the performance of time-series forecasting algorithms. In particular, we use the *catch22* and *tsfresh* libraries to generate feature representations of the time-series in the M4 dataset (Makridakis, Spiliotis, & Assimakopoulos, 2018, 2020), which are then used to train ML regression models to predict the performance of 61 time-series forecasting algorithms. The main motivation is therefore not to create a meta-learning approach that selects the best algorithm or a set of algorithms and combines them to get good forecasting results, but to explain what features are being used when predicting performance and to quantify their importance. We also explore how feature importances change with different meta-learning models and how they change when predicting performance for a specific forecasting algorithm. To provide a robust and fair analysis of the importance of each feature, we employ several methods for determining feature importance, including permutation feature importance, the SHapley Additive exPlanation (SHAP) (Lundberg & Lee, 2017) framework, as well as the feature importance derived from the intrinsic interpretability of models based on decision trees. The obtained feature importance values are analyzed to get a better insight into how they are related to predicting forecasting algorithms' performance. We provide a detailed exploration of which meta-features are important for different forecasting algorithms and meta-models used to predict performance.

The paper is organized as follows: Section 2 describes the existing approaches for algorithm selection related to the domain of forecasting. In Section 3 we introduce the methodology used to obtain feature importances, followed by the results in Section 4 and limitations of the study in Section 5. Finally, in Section 6 we conclude our work and put forward some future research directions.

2. Related work

A large amount of research on diverse sets of time-series data has been conducted (Makridakis & Hibon, 1979; Newbold & Granger, 1974) to determine if and how different time-series properties affect the accuracy of forecasting methods. Analysis has shown that different properties can have a large influence on the performance of forecasting methods, although being able to capture more complex time dependent relations, does not necessarily produce better results. The link between forecasting accuracy and time-series features was explored in Meade (2000) where authors trained a regression model to predict the performance of nine forecasting methods based on 25 extracted times-series features. At that time, methods for feature creation together with datasets for forecasting were fairly limited.

Several research works have already explored the meta-learning paradigm for automated AS for time-series forecasting. The Feature-based FORecast Model Selection framework (Talagala, Hyndman, & Athanasopoulos, 2018) applies a Random Forest(RF) model with time-series features as inputs, for the selection of a single forecast model. The Feature-based Forecast Model Averaging framework (Montero-Manso, Athanasopoulos, Hyndman, & Talagala, 2020) applies a gradient boosting model to select the weights for a weighted forecast combination, using 41 features extracted by the tsfeatures (Hyndman, Kang, Montero-Manso, Talagala, Wang, Yang, & O'Hara-Wild, 2020) library. The Feature-based Forecast Model Performance Prediction framework (Talagala et al., 2021) treats the forecasting AS problem as a multi-class ranking problem, i.e. meta-learning is used to rank forecast models by simultaneously predicting forecast errors, allowing the user to identify a subset of forecasting models.

The benefits of ensembles in forecasting have been pointed out in Gastinger, Nicolas, Stepić, Schmidt, and Schülke (2021), where it was demonstrated that there is no single best ensemble model and hyperparameter configuration which is well suited for all time-series instances, introducing the need for meta-learning. The meta-learning in this case is implemented by training a binary classifier for each combination of ensemble model and hyperparameter configuration, which indicates whether the model should be used for a specific time-series instance. The instances are represented by the meta-features extracted using the tsfeatures (Hyndman et al., 2020) library and seven general meta-learning features, which are not specific to time-series.

A two-step meta-learning approach has also been proposed for time-series forecasting ensembles (Vaiciukynas, Danenas, Kontrimas, & Butleris, 2021), which first involves ranking the forecasting models by predicting the error they would achieve on the testing instance, and then determining the number of models that should be used in the forecasting ensemble. Both tasks are performed using RF regression models. The time-series instances are represented using 390 meta-features, obtained by generating 130 different features on the original time-series instances, as well as two transformations of each instance. Permutation feature importance is used to analyze the contribution of the meta-features to the meta-models' performance. However, the features are first grouped into several sets, and the importance is reported on a set level, and not on the level of individual features. In contrast, we are interested in how individual features contribute to the meta-model's performance.

We also need to emphasize that meta-learning studies that select the most appropriate algorithms or combine them into an ensemble use

an apriori selected algorithm portfolio (i.e., set of forecasting methods that are used in the study). Nowadays, one of the biggest open research questions is how to select which algorithms should be used in order to generalize the meta-learning approach and to decrease the bias presented in the selection of the set of methods. We are not addressing this question in our study, since our goal is to provide a general methodology to see how the time-series features are linked to the performance of any forecasting method. We are using a set of forecasting algorithms that entered the M4 competition. However, we would like to point out that the field of forecasting methods is increasingly moving towards deep learning based models that can be trained on many instances (sometimes referred to as global models (Hewamalage, Bergmeir, & Bandara, 2022)) and are able to generalize the learned knowledge to previously unseen instances. Such models include recurrent neural networks (Werbos, 1990) with further improvements (Chung, Gulcehre, Cho, & Bengio, 2014; Salinas, Flunkert, Gasthaus, & Januschowski, 2020), convolutional neural networks (Borovykh, Bohte, & Oosterlee, 2018; Chen, Kang, Chen, & Wang, 2020; Li et al., 2021) and transformers (Vaswani et al., 2017). Focus is also shifting towards neural networks that try to combine approaches from deep learning and interpretability from classical statistical modes such as N-BEATS: Neural basis expansion analysis for interpretable time series forecasting (Oreshkin, Carpov, Chapados, & Bengio, 2019) and Temporal fusion transformers (Lim, Arık, Loeff, & Pfister, 2021).

3. Methodology

Fig. 1 gives an overview of the proposed methodology for explaining the importance of features used for representing time-series instances for predicting the performance of ML algorithms achieved on them. The methodology involves three main steps:

- *Data collection*, which involves extracting time-series features of the data instances and collecting forecasting performance data achieved on them. Here, features of individual time-series instances are extracted and the performance of forecasting algorithms is collected from a publicly available repository;
- *Building a diverse portfolio of performance prediction models*, where forecasting performance of time-series algorithms is predicted based on the extracted time-series features using different ML models;
- *Feature importance analysis*, where multiple feature importance approaches are used to determine which features are the most influential ones for predicting forecasting performance and how feature importance changes with different ML models.

3.1. M4 dataset

The M4 time-series forecasting dataset (Makridakis et al., 2018, 2020) was used for the fourth edition of the Makridakis Forecasting Competition. The dataset consists of 100,000 time-series collected either yearly, quarterly, monthly, weekly, daily or hourly. All the time-series instances are divided into a training set for training the forecasting models and a test set designed to evaluate forecasts. Training instances can also greatly differ in length. The shortest time-series consists of only 13 yearly observations while the longest time-series collected daily consists of 9,919 observations.

At the end of the competition, 61 forecasting algorithms were submitted together with forecasts for all 100,000 time-series. Submitted algorithms can be broadly categorized as statistical forecasting algorithms, forecasting algorithms based on ML models or hybrid approaches. The best performing algorithm was a hybrid approach (Smyl, 2020) which combined exponential smoothing with a recurrent neural network. Other top performing approaches mostly relied on combining forecasts of different statistical approaches to achieve good results.

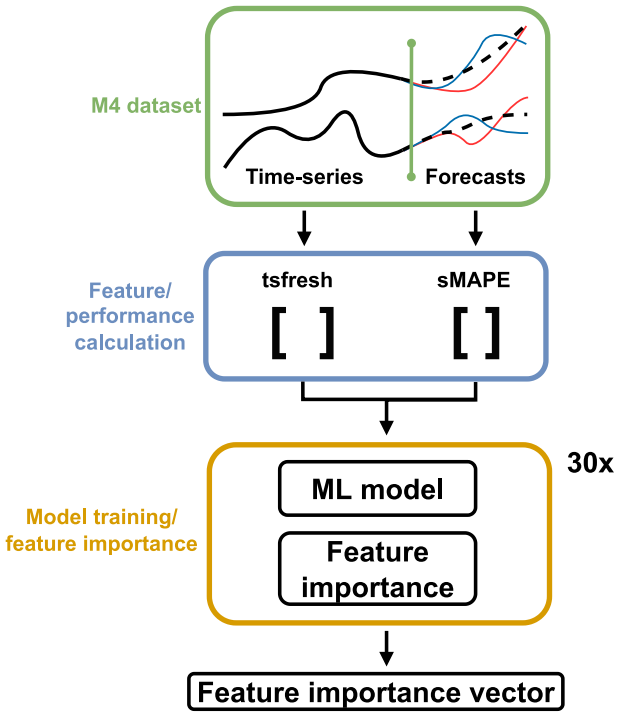

Fig. 1. Pipeline for calculating feature importance. Features and forecasting performance is first extracted from M4 dataset. Extracted features and forecasting performances are repeatedly split into 30 train/test sets on which performance prediction algorithms are trained and feature values are calculated.

Table 1
Hyperparameters for the meta-models used to predict performance.

Model	Parameter	Value
Neural network	activation	ReLU
	number of hidden layers	3
	hidden layer size	500
	optimizer	Adam
RF [MT]	number of trees	100
	max depth	No limit
	min samples in a leaf node	No limit
RF [ST]	number of trees	25
	max depth	No limit
	min samples in a leaf node	2
XGBoost	learning rate	0.3
	maximum tree depth	6
KNN [Cosine]	neighbors	5
KNN [Euclidean]	neighbors	5
Mean	–	–

3.2. Extracting time-series features and collecting the performance data

The time-series data instances were used in the feature extraction process to obtain tabular data that is further linked to the algorithm performance.

3.2.1. Feature extraction

First, a set of time-series features are extracted from the training part of each time-series instance in the dataset using the methods described later in this section. When using forecasting algorithms to predict the future behavior of time-series, a common practice is to first transform time-series to achieve/improve stationarity or to stabilize the variance. This is especially important for some forecasting algorithms (e.g. MA, AR or ARMA) that can only operate on time-series that are strictly/weakly stationary process (Patterson, 2011; Van Greunen,

Heymans, Van Heerden, & Van Vuuren, 2014). One transformation that makes time-series stationary is to compute the differences between consecutive observations, also known as differencing. Differencing can remove trends and seasonality which helps with stabilizing the mean of a time-series. The other property that is desirable when forecasting is to stabilize the variance of a time-series, which can be achieved by applying a logarithmic transformation. Before extracting features, we also apply differencing and logarithmic transformation to the time-series. Due to the specific structure and properties of time-series instances contained in the M4 dataset, feature extraction methods may sometimes produce invalid features (i.e. features with missing values) of features with the same value for all time-series. Features with missing values or identical values are therefore removed.

In our case, we used two well-known features sets for extracting features for time-series data instances, Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (*tsfresh*) (Christ, Braun, Neuffer, & Kempa-Liehr, 2018) and The Canonical Time Series Characteristics (*catch22*) (Lubba et al., 2019), which are most commonly used in time-series forecasting tasks. These methods were used three times depending on which data were applied: (i) raw original time-series data, (ii) its differencing transformation, and (iii) its logarithmic transformation.

tsfresh is a library for extracting a diverse set of features that proved to be effective in capturing and extracting quality information in different scientific areas (financial, biological, industrial applications, etc.). The library can extract just under 800 features from the different feature groups (i.e. statistical, information based, model-based, stationarity, frequency-based, domain-specific features, etc.).

catch22 is a subset of features that were selected as being the most informative/discriminatory features from the set of over 7,700 features from Highly Comparative Time Series Analysis (*hctsa*) (Fulcher, Little, & Jones, 2013) evaluated on the University of California, Riverside (UCR) dataset (Bagnall, Lines, Bostrom, Large, & Keogh, 2017). A subset of 22 features was determined in three steps: (i) remove the features that cannot be calculated or are sensitive to the mean and variance, (ii) evaluate the predictive performance of features and discard features that are below some preselected threshold, (iii) group features into 22 clusters and select the feature that is interpretable, achieves good performance and can be efficiently computed.

We also need to point out here that a common approach for extracting features from time-series is also by applying random convolutional kernels to the existing time-series dataset as done in ROCKET (Dempster et al., 2020) and later in MiniROCKET (Dempster et al., 2021). Similarly, feature generation can also be performed by representing time-series in a frequency domain (Chen et al., 2019; Srinivasan, Eswaran, et al., 2005) or by using a discrete wavelet transformation (Chaovalit, Gangopadhyay, Karabatis, & Chen, 2011). While such methods are extremely powerful when classifying time-series instances, the downside of such approaches is mainly that extracted features are not meaningful to humans and are hard to interpret. Since they provide black-box representations, they were omitted from our analysis.

3.2.2. Performance data

When forecasts are performed, one has to measure the error or forecast compared to the actual observations. The symmetric mean absolute percentage error (sMAPE) (Hyndman & Koehler, 2006) is one such measure that quantifies the prediction accuracy of a forecasting method. The forecasted values are first subtracted from the actual observed values and the absolute value is taken. The obtained difference is further normalized by the sum of absolute values of the forecasted and true predictions. Calculated errors are averaged over the whole forecasting horizon. The following equation is used to calculate the sMAPE metric:

$$sMAPE = \frac{2}{h}\sum_{t=n+1}^{n+h}\frac{\left|Y_t - \hat{Y}_t\right|}{|Y_t| + \left|\hat{Y}_t\right|}$$

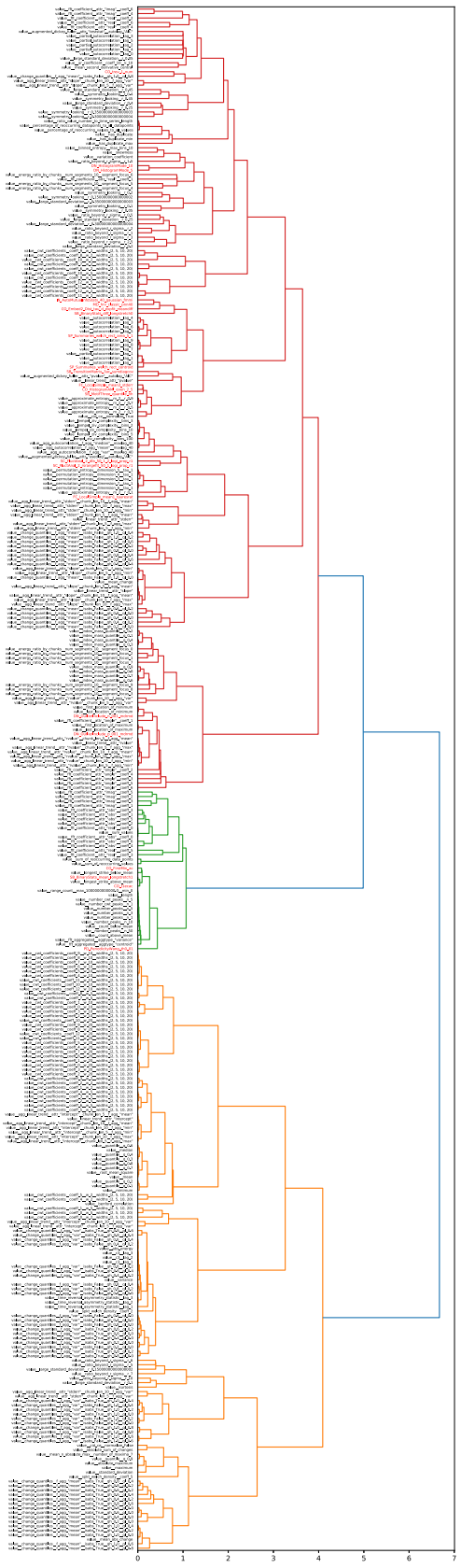

Fig. 2. Dendrogram of *tsfresh* (black) and *catch22* (red) features based on the Pearson correlation between features. The similarity measure is transformed into a distance measure with $1 - abs(pearson(F_i, F_j))$ where F_i and F_j are extracted feature vectors spanning all 100,000 time-series instances. Clusters are merged with the Ward linkage function.

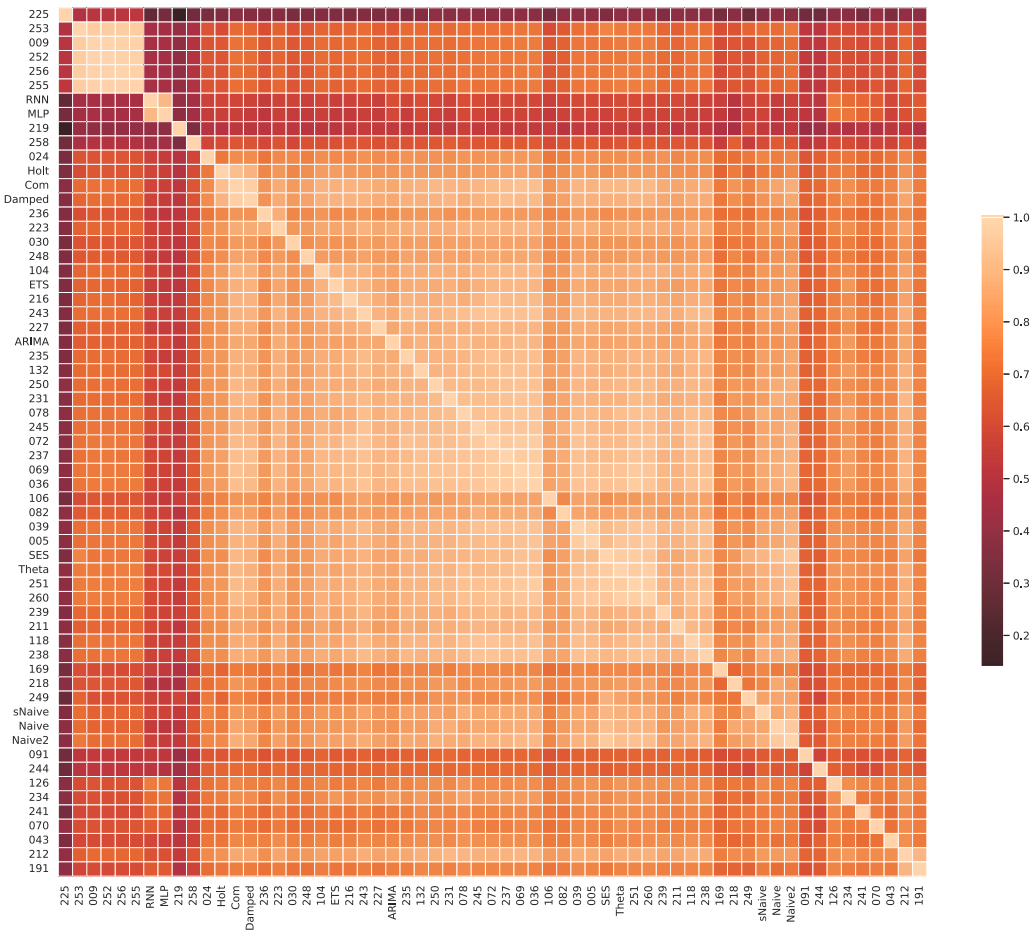

Fig. 3. Pearson correlation between forecasting performance calculated using sMAPE on 100,000 time-series instances.

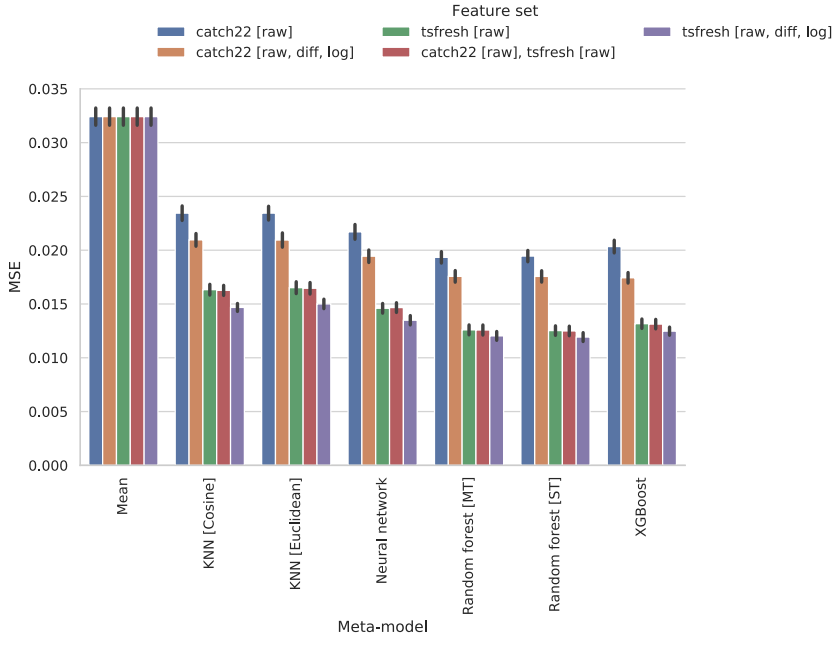

Fig. 4. MSE achieved by different ML models and different sets of features when predicting forecasting errors for 61 time-series algorithms.

where n is the number of available datapoints in the training part of each time-series, h is the forecasting horizon and Y_t and $\hat{Y}_t$ are the true and the forecasted value of time-series and time t. Lower sMAPE values indicate lower prediction errors, so the goal is to minimize this measure. Note that sMAPE is a modification of the MAPE metric and improves on some shortcomings of MAPE such as having a lower bound

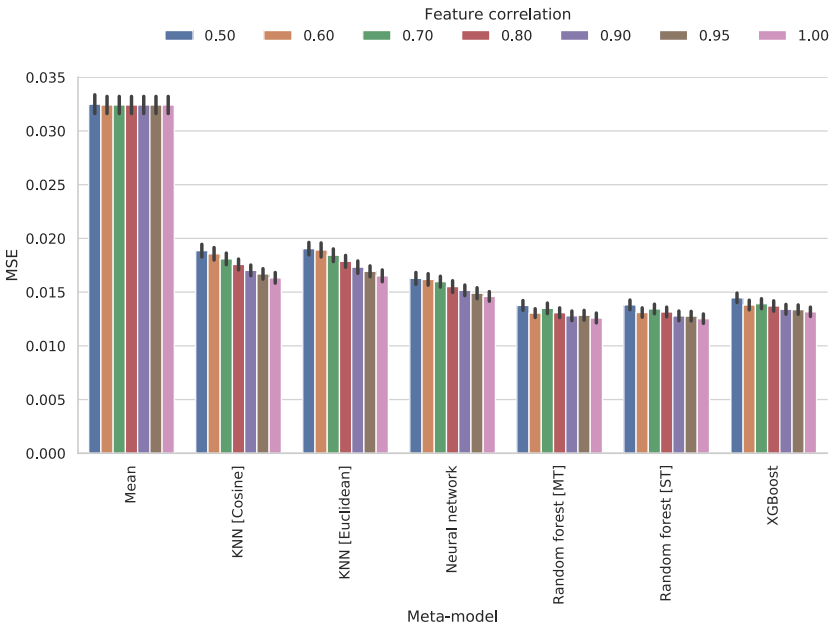

Fig. 5. MSE for different ML models with *tsfresh* features and different thresholds for removing correlated features.

of zero and upper bound of two. Even with this modification, sMAPE can have problems if both the forecast and the actual values are equal to zero, making the denominator also zero. The M4 dataset shifts the time-series in such way to avoid this problem.

Based on the known test parts of the time-series and forecasts made with each of the 61 models, we do the performance extraction by computing the forecasting error between all pairs of forecasting algorithms and time-series instances, expressed using the sMAPE.

3.3. Building a diverse portfolio of performance prediction models

To predict the performance of the 61 forecasting algorithms on the time-series data instances represented by the extracted features, we build a diverse set of performance prediction models, which we refer to as meta-models. The analysis of the meta-models is further used to provide explanations about the importance of the time-series features. For this purpose, the obtained time-series feature data and performance data is first split into training and testing portions, making up 90% and 10% of the original data, respectively. The procedure is repeated 30 times using random sampling, to obtain 30 splits of the original data. Further, we used different ML algorithms including *Neural Network*, *Random Forest (RF)*, *XGBoost*, *K Nearest Neighbors (KNN)*, and *Mean baseline* to train the meta-models on each of the train splits and evaluated on the corresponding test sets. More details about the algorithms used to train the meta-models are presented in Appendix A. Repeating the model training on different train/test splits guarantees that we get a robust estimation of the expected error when predicting forecasting performance and subsequently what features are used for making forecasts and how features differ with different data splits handling the inherent bias that originates from the data quality in different splits.

The meta-models were either trained as multi-target regression (MTR) models, where one model predicts performance for all forecasting algorithms, or as multiple single-target regression (STR) models, where one model is trained for each forecasting algorithm. Due to the computational complexity involved in training the models and calculating feature importance, *Neural Network* and *KNN* were trained only as MTR models. *RF* was trained twice, once as a MTR model and once as a collection of multiple STR models. Lastly, *XGBoost* is designed to only work as a STR model therefore one is trained for each of the 61 forecasting algorithms.

When training the meta-models used to predict forecasting algorithms' performance, an extensive hyperparameter search is desirable or even required for obtaining models with state-of-the-art performance. Due to the high computational costs, we did not extensively tune the hyperparameters to obtain the best possible performance for the meta-learning models used to predict performance. The goal was primarily to ensure that selected models are diverse (use different learning principles) to see how this influences the feature importance values and the explanations provided for each model.

3.4. Feature importance for providing explanations

For each of the 30 train/test splits and corresponding performance predicting meta-model, we evaluate what features are deemed to be important. Feature importance can differ depending on several aspects: *(i) the meta-model* that predicts the performance based on the extracted features, *(ii) the forecasting algorithm* whose performance is being predicted and *(iii) the method for explaining feature importance*. We analyze the feature importance considering each of the above-mentioned aspects. To do this, we explore the following combinations of feature importance measures:

*(i) Permutation feature importance evaluated on neural network (**P/NN**), multi-task random forest (**P/RF**), singletask random forest (**P/SRF**), XGBoost (**P/XGB**) and Mean predictor (**P/Mean**)*

*(ii) SHAP feature importance evaluated with neural network (**S/NN**), multi-task random forest (**S/RF**), singletask random forest (**S/SRF**), XGBoost (**S/XGB**) and Mean predictor (**S/Mean**)*

*(iii) Feature importance obtained during XGBoost training with different feature importance metrics (**C/XGB, G/XGB, TC/XGB, TG/XGB, W/XGB**)*

*(iv) Feature importance obtained during random singletarget forest training (**RF/SRF**).*

More details about the feature importance models are presented in Appendix B.

4. Results

Here, the results from our analysis are presented, which are split in three parts. First, we provide a separate analysis of the time-series feature space and the performance space, treating them independently. Next, we explore the time-series feature importance in combination with different performance predicting models and time-series forecasting models.

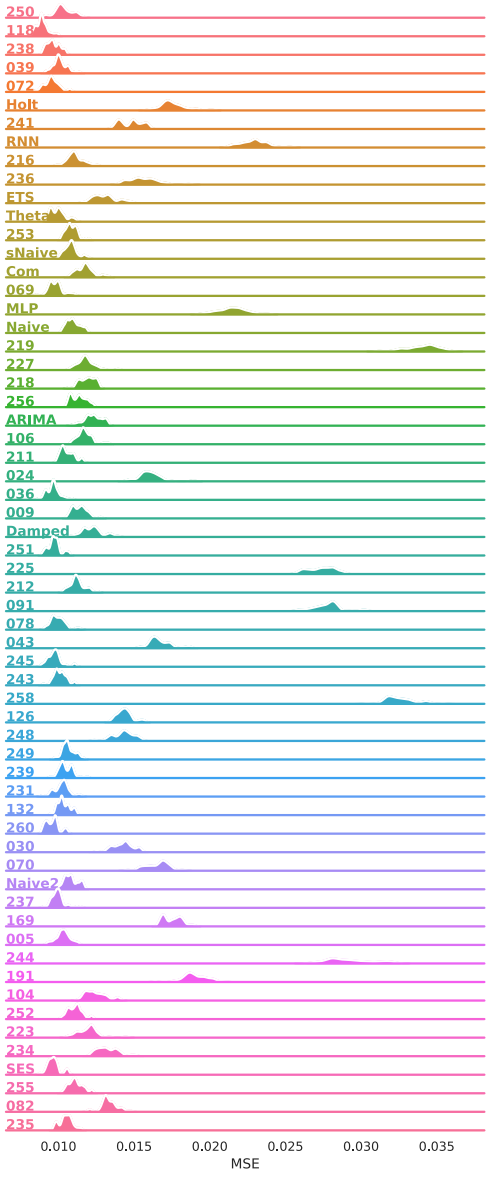

Fig. 6. Ridgeline plot representing the MSE between the true forecasting performance evaluated with MAPE and the performance predicted with RF for each of the forecasting algorithms over 30 train/test splits. The row labels contain the feature names, while the column labels contain the names of the meta-models.

4.1. Experimental setup

For the implementation of the meta-models, we used the *scikit-learn* (Pedregosa et al., 2011), *keras* (Chollet et al., 2015) and *XGBoost* (Chen & Guestrin, 2016) python libraries. The workflow and analysis are performed using Snakemake (Mölder et al., 2021), ensuring reproducibility. The experiments were executed on a system using the Ubuntu operating system, an Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50 GHz, and 1 TB of RAM. The relevant code can be found at the Gitlab repository https://repo.ijs.si/gpetelin/m4-feature-importance. Associated data is available at https://zenodo.org/record/6637637. Reasonable default hyperparameters were selected to balance model performance during train and inference time. Table 1 describes the hyperparameters of the meta-models. Apart from this, feature importance methods produce importance values that can differ in scale. For that purpose, comparison of raw values between different feature importance methods is not always possible. In such cases, feature importance is determined based on the rank of the feature compared to all other features.

4.2. Feature space analysis

Applying the features extraction techniques described in Section 3.2.1, we obtain six different sets of features for each time-series instance. Each one was extracted using the *tsfresh* or *catch22* libraries, in combination with the raw time-series, the time-series that were transformed with the first difference, and the time-series with a logarithmic transformation applied.

Fig. 2 shows a hierarchical clustering based on Pearson correlations between the *tsfresh* and *catch22* features obtained from the time-series where no transformation was applied. Using this figure, we can observe that a lot of extracted features are linearly correlated. In addition, high correlations between *tsfresh* features are also presented, which is expected, since some features are extracted using the same feature extraction method, only with different parameters. Example of two such features would be feature *cwt_coefficients_coeff_1_w_10_widths_(2, 5, 10, 20)* being strongly correlated with feature *cwt_coefficients_coeff_0_w_10_widths_(2, 5, 10, 20)*, with correlation of 0.96.

Comparing *tsfresh* (black) and *catch22* (red) features in Fig. 2, we can observe that there are a lot of features that are strongly linearly correlated. The second more interesting observation is that *catch22* features do not capture large parts of the time-series landscape. This means that there is a large group of *tsfresh* features that are not correlated with any *catch22* features.

To further explore the correlations between raw and transformed time-series feature, one can compare the Pearson correlation between the raw features and the features obtained from the transformed time-series. Focusing on *catch22*, the mean correlation between raw features and their corresponding feature from log transformed time-series is 0.88. This signals that on average, features from log transformed time-series contain information correlated with raw features. For the differenced time-series, corresponding features are not highly correlated, with the average correlation being 0.24. A similar pattern can also be observed with the *tsfresh* features. In that case, the features extracted from log transformed and differenced time-series have Person correlations with the raw features of 0.66 and 0.35, respectively. It is also the case in both feature extraction methods that some log transformed and differenced features have correlation of 1.0 with their raw counterparts. An example of how extracting features from transformed time-series does not always bring additional benefits is the feature *first_location_of_maximum*, where raw and log transformation produce the same values and therefore do not offer any additional value.

4.3. Performance space analysis

The M4 dataset consists of 61 forecasting algorithms and their predictions for 100,000 time-series instances. Fig. 3 shows the Pearson correlations between sMAPE forecasting performance on all 100,000 time-series instances that exist in the M4 dataset. We can observe that performances of forecasting algorithms are usually strongly correlated, meaning that if one algorithm performs well on a given time-series instance, there is a high probability that another algorithm will also perform well. Based on the performance we can also observe clusters of similarly performing forecasting algorithms. One such example are the algorithms *MLP* (perceptron with pre-applied detrending and deseasonalization) and *RNN* (recurrent network with pre-applied detrending and deseasonalization), both ML-based forecasting algorithms with highly correlated performance. Similarly, algorithms *sNaive*, *Naive* and *Naive2* form another cluster of algorithms where forecasting performance is highly correlated. This is a consequence of the algorithms being similar in nature (random walk models), with small modifications. One forecasting algorithm that is not correlated with other algorithms is the forecasting algorithm *225* that achieves poor forecasting performance evaluated with sMAPE on most of the time-series in the dataset.

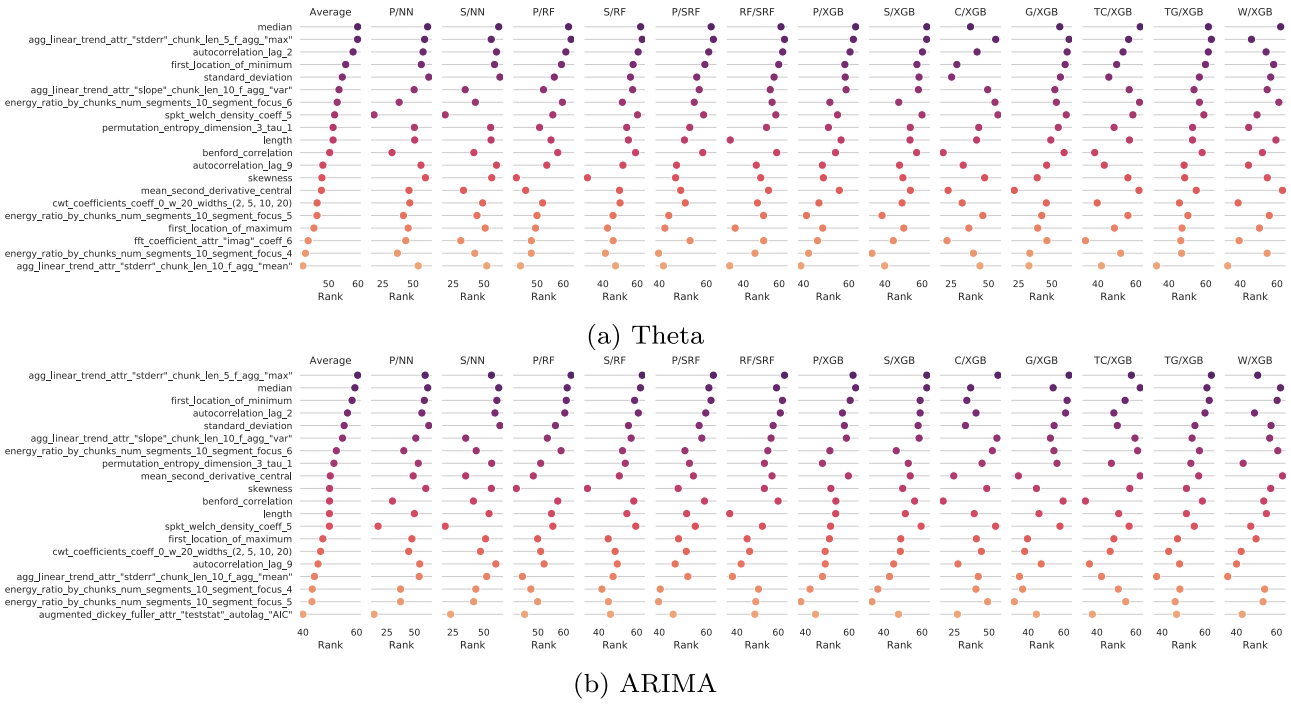

Fig. 7. Feature ranking for Theta (a) and ARIMA (b) forecasting algorithms based on the ranking averaged over 30 train/test splits. The row labels contain the feature names, while the column labels contain the feature importance and meta-model names.

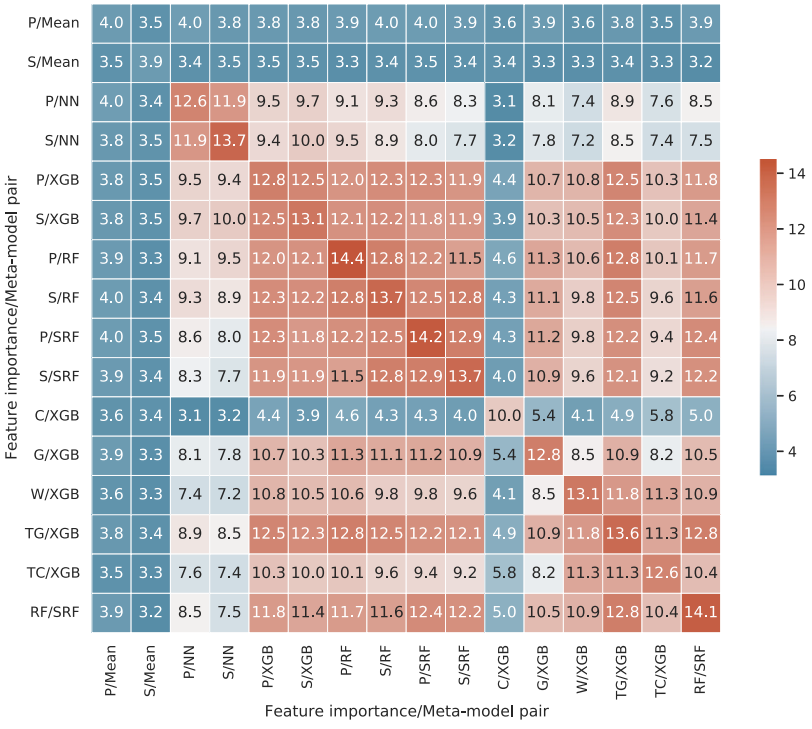

Fig. 8. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for Theta forecasting algorithm, averaged over 30 train/test splits.

4.4. Meta-model performance analysis

To provide a reliable analysis of the importance of each feature when predicting the performance of a time-series forecasting algorithm, we build diverse meta-models to predict the performance of different forecasting algorithms from the obtained time-series features. Fig. 4 shows the performance of different meta-models when using different sets of features averaged over 30 splits. For additional information, black lines on top of each show the standard deviation in MSE. Each meta-model (i.e., KNN with cosine similarity, KNN with Euclidean similarity, RF, XGBoost, Neural Network, and Mean as a baseline) was evaluated with five different feature sets.

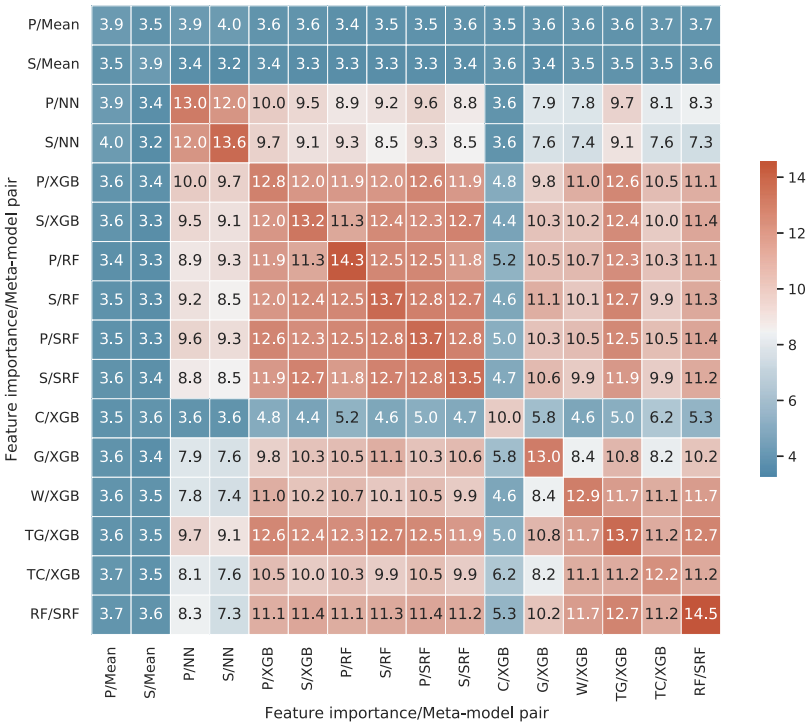

Fig. 9. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for ARIMA forecasting algorithm, averaged over 30 train/test splits.

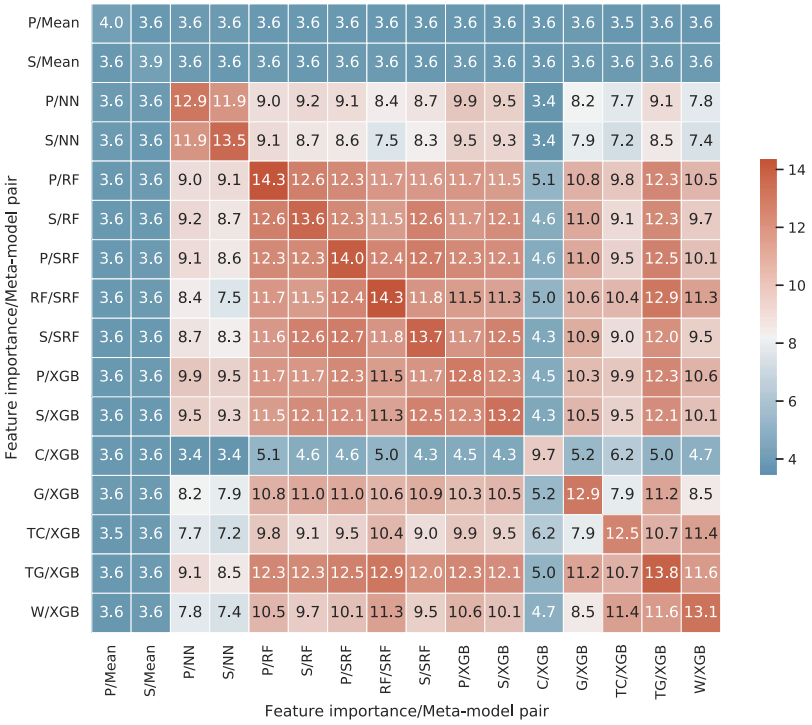

Fig. 10. Agreement on what 15 features are the most important ones as determined by different meta-model/feature importance methods for all forecasting algorithms, averaged over 30 train/test splits.

As feature sets we selected the *tsfresh* features calculated using the raw time-series (*tsfresh [raw]*); *catch22* features calculated using the raw time-series (*catch22 [raw]*); fusion of all *tsfresh* features (*tsfresh [raw, diff, log]*) calculated from the raw time-series, the transformed time-series using differencing, and the transformed time-series using logarithmic transformation; the same fusion using the *catch22* features (*catch [raw, diff, log]*) that fuse raw, differenced and log transformed time-series features; and a fusion of *tsfresh* and *catch22* features calculated from the raw time-series data (*catch22 [raw], tsfresh [raw]*).

The meta-models are evaluated using a Mean Squared Error (MSE) between the ground truth sMAPE performance and the predicted

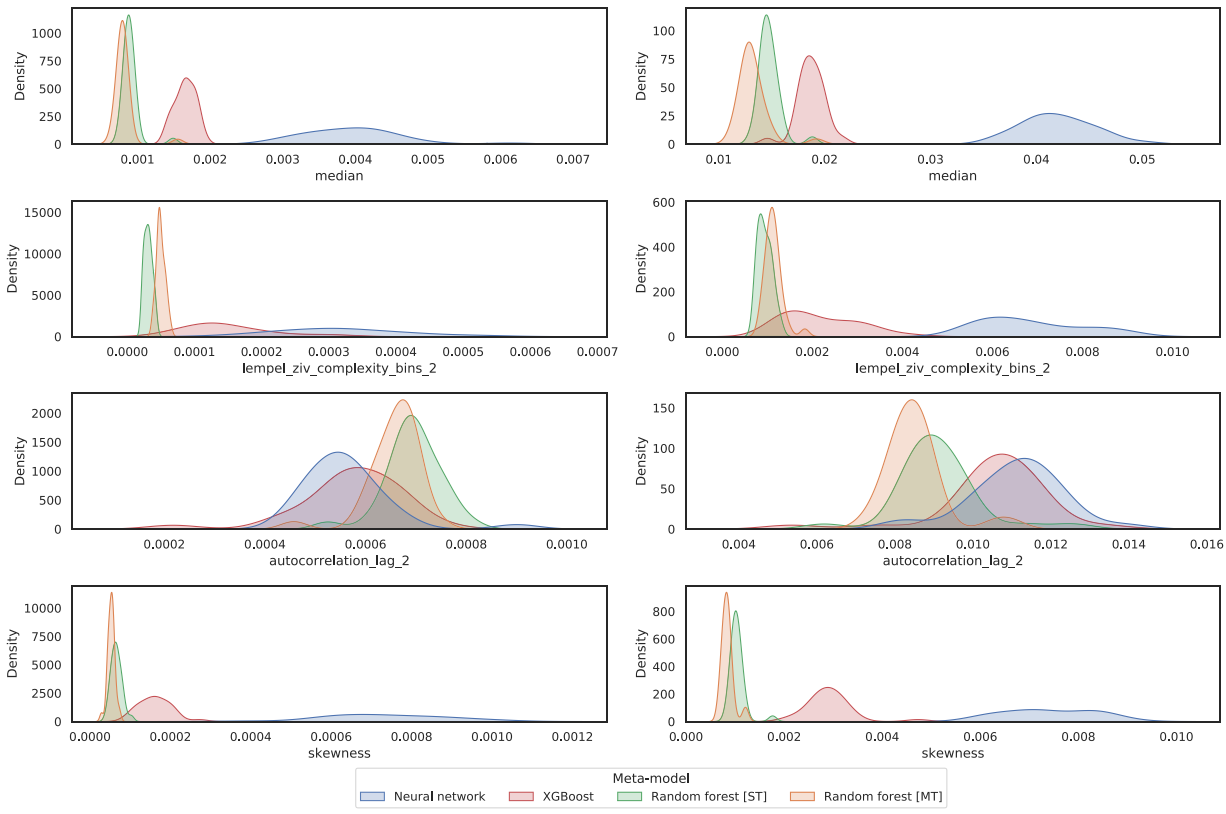

Fig. 11. Kernel density estimation feature importance of features *median*, *lempel_ziv_complexity_bins_2*, *autocorrelation_lag_2* and *skewness* for Theta forecasting algorithms as obtained with Permutation feature importance (left) and SHAP (right). Note that axis use different scales.

sMAPE performance, averaged across all 61 targets. We can observe that the baseline model which always predicts the mean performance for each forecasting algorithm, independent of the feature set used to represent the time-series data instances, achieves a MSE of 0.032. The other meta-models, which rely on the above-mentioned feature sets achieve much better estimates of the performance of the 61 forecasting algorithms.

Comparing the sets of features, we can observe that the *tsfresh* features offer better predictive performance compared to the *catch22* features, independent of the meta-model that is used. By fusing both the *catch22* and the *tsfresh* feature sets, the predictive performance is almost identical to the performance obtained using only the *tsfresh* features. We can therefore conclude that the 323 *tsfresh* features are much more informative than the 22 features extracted with *catch22* when predicting the performance of the 61 forecasting algorithms on the M4 dataset. This is not surprising, since in the feature space analysis (see Section 4.2), we have shown that the *tsfresh* features set is much larger and covers parts of the feature landscape that are not covered by the *catch22* features, i.e. the *catch22* features capture only a subset of the information captured by the *tsfresh* feature set.

The second interesting observation is the predictive performance obtained using the feature sets extracted from the differenced and log transformed time-series. For *catch22*, fusing the features extracted from the raw, differenced, and log transformed data has a large impact on the predictive performance for all used meta-models. With the extended set of *catch22* features, all models achieve practically better MSE compared to the features extracted just from the raw time-series. With the *tsfresh* features, extracting features from the raw time-series and fusing them with features from the differenced and log transformed time-series does improve the meta-learner performance, but the difference in performance obtained using only the raw time-series features, and the fused time-series features is not as large as with the *catch22* feature set. From the results, it follows that all meta-learners achieve good performance with the *tsfresh* features extracted only from the raw time-series data. Therefore, we decide to only use this feature set for further analysis. We need to point out here that if the predictive performance of the meta-learners was the primarily goal, using the *tsfresh* features extracted from the raw time-series and fusing them with the features extracted from differenced and log transformed data would also be beneficial.

Since the feature space analysis shows that the extracted *tsfresh* time-series features can be highly correlated (see Section 4.2), this can sometimes adversely affect the training of certain models and subsequent methods for evaluating feature importance. A way to remove feature correlations is to project data instances to a lower-dimensional space using feature reduction methods, such as Principal Component Analysis or Non-Negative Matrix Factorization. Unfortunately, such an approach makes contributions of individual features difficult to compute. For this purpose, we explore the performance of the meta-models when correlated features are removed, if their correlation exceeds a predefined threshold. Fig. 5 presents the meta-models' performance (MSE averaged across 30 train/test split with standard deviation) when features which have correlations exceeding a certain threshold are removed and thus not used during the training and testing. We have evaluated six different correlation thresholds: 0.50, 0.60, 0.70, 0.80, 0.90, and 0.95. When the threshold is set to 1.00, this means that all *tsfresh* features are kept for the learning process and we did this only for comparing the performance with the meta-models achieved with the reduced feature sets. Removing a subset of highly correlated features slightly degrades the models' performance, indicating that the removed features, despite being linearly correlated, do in fact contain additional information. For the purpose of reducing computational costs associated with computing feature importance for the full set of features and problems that can arise from determining the feature importance of correlated features, we choose to work only with the features that are not linearly correlated with other features above the pre-selected threshold of 0.50.

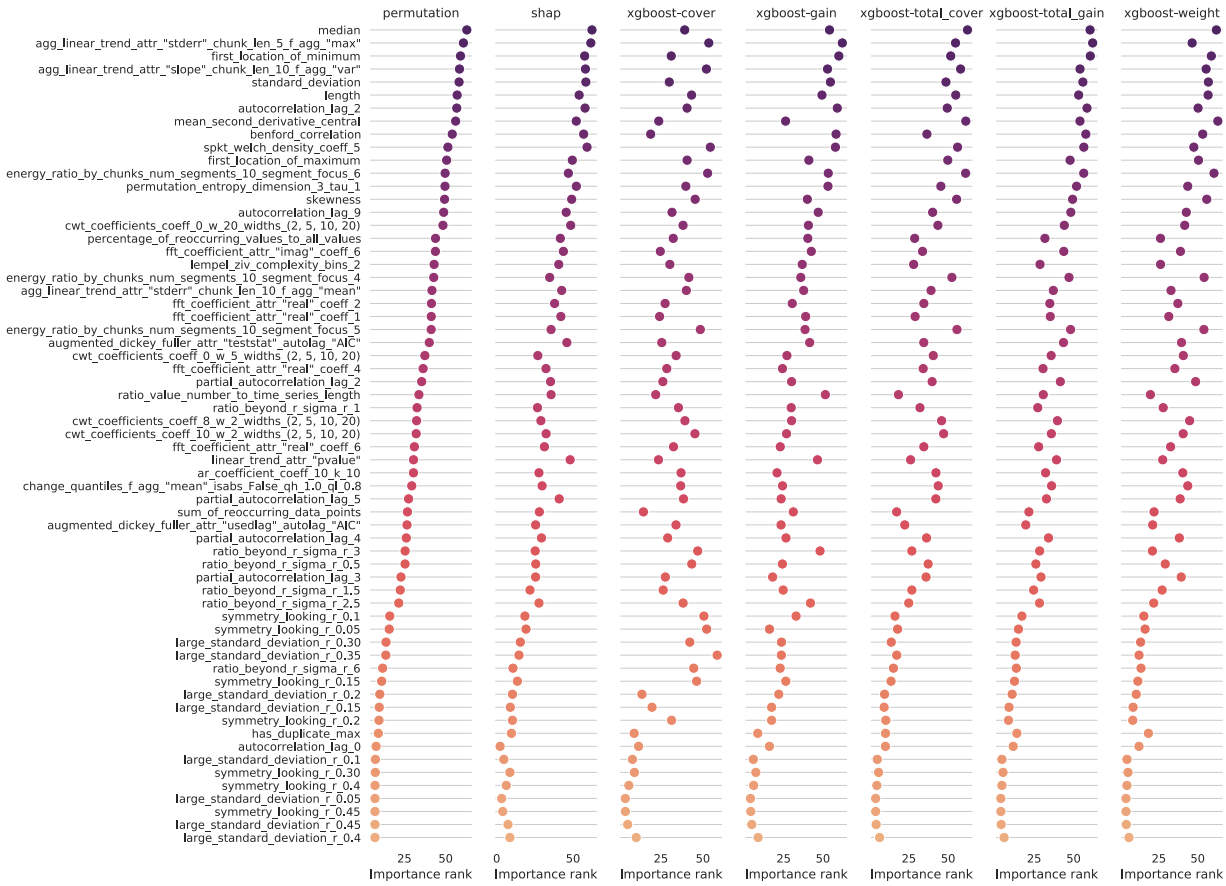

Fig. 12. Feature importance ranking for XGBoost with different feature importance methods aggregated over 30 runs and all forecasting models. For clarity, features are sorted by the permutation feature importance.

Fig. 6 shows the MSE between the actual sMAPE obtained by each forecasting algorithm and the sMAPE predicted with the best performing meta-model RF with the reduced feature set with 0.50 threshold. The MSE is presented for 30 different train and test splits. From the figure, we can see that the performance of the meta-models is generally stable with relatively small deviations in the MSE across different train/test splits.

We can also observe that estimating the sMAPE performance for some of the forecasting algorithms is much more challenging than for others. The MSEs for most of the forecasting algorithms are around 0.01, however for some forecasting algorithms the estimates of the performance are less accurate with MSE above 0.02. The five forecasting algorithms with a larger MSE whose performance is more difficult to predict are algorithms 219, 225, 091, 258 and 244. In general, the algorithms whose performance is hard to predict performed bellow the average in the competition, meaning that their forecasting error evaluated with sMAPE is higher.

4.5. Feature importance for explainability of forecasting algorithms

In this section, we investigate the feature importance obtained for several forecasting algorithms, based on the meta-models and feature importance methods used. In this scenario, a few forecasting algorithms are selected and kept fixed while changing the meta-models and feature importance methods.

Fig. 7 visualizes the top 20 most important features (globally across all forecasting algorithms, meta-models and feature importance models) and how they are ranked with different meta-models/feature importance methods for Theta and ARIMA forecasting algorithms. Note that Theta and ARIMA forecasting algorithms were selected since they are the baseline models with available source code and relatively good forecasting accuracy compared to some other forecasting algorithms. As expected, for the meta-model that only predicts mean values, all features are assigned the same feature importance by both the permutation-based and the SHAP feature importance methods. This confirms that both feature importance methods can discover when features are not used to make a prediction. Because all features have the same importance values, they are omitted in Fig. 7 to improve readability. Other meta-models, however, rely heavily on the features. For those meta-models, large variation in the raking of individual features can be observed based on the meta-model and feature importance methods used. The largest discrepancies in feature importance orders are between model-specific methods (where feature importance is determined while the model is built such as the XGBoost cover feature importance - **C/XGB**) and model agnostic feature importance (where feature importance is determined after the model is built such as SHAP or permutation method). This demonstrates that model-specific features that are built during the models' construction from the train set can substantially differ from other approaches. When comparing feature importance between different meta-models using the same feature importance method, we can observe that tree-based models utilize different features compared to the neural network. Lastly, the feature importance ranking obtained with SHAP and the one obtained with permutation importance is similar when the same meta-model is used. This can be seen when comparing columns pairs such as **P/NN** and **S/NN** or **P/RF** and **S/RF**.

To better demonstrate the similarities between different meta-models and feature importance methods, Figs. 8 and 9 show how many of the top 15 features are shared for different meta-model/feature importance combinations for the Theta and ARIMA forecasting algorithms. This is obtained as an average number of top 15 features

Fig. 13. Feature importance ranks for different meta-models using permutation based feature importance averaged over all forecasting algorithms and 30 runs.

that are shared between all the pairs of 30 runs. We can observe that selecting top 15 most important features can differ depending on what meta-model and feature importance model is used. For example, predicting performance of Theta forecasting algorithm using RF meta-model with permutation feature importance (**P/RF**), top 15 features agree on 14.3 out of 15 features between different train/test splits. Diagonal elements of the figure thus show that even when meta-model and feature importance models are fixed, top features may still differ between different splits. Additionally, when comparing meta-models with different feature importance methods, large disagreements can be observed. For instance, XGBoost with permutation feature importance (**P/XGB**) and gain feature importance (**G/XGB**) achieve average agreement only on 10.8 features for the Theta forecasting algorithm.

Fig. 10 also shows the global top 15 feature agreement between all meta-model/feature importance combinations for 30 runs across all forecasting algorithms. We can again observe that feature importance can vary depending on the meta-model and feature importance method. Therefore, when determining feature importance one is highly dependant on what methods are selected.

Further insight can be obtained when comparing the "raw" feature importance values and not just the rank. Fig. 11 shows the distribution of feature importance values for the Theta forecasting algorithm and four selected features (*median*, *lempel_ziv_complexity_bins_2*, *autocorrelation_lag_2* and *skewness*) as a distribution over 30 train/test splits obtained with the SHAP and permutation feature importance methods. Observation can be made that feature importance distributions differ with different meta-models. For the feature importance of the *median* feature, obtained with permutation feature importance, we can observe that the neural network assigns higher feature importance values than the other methods. It is also interesting to note that the

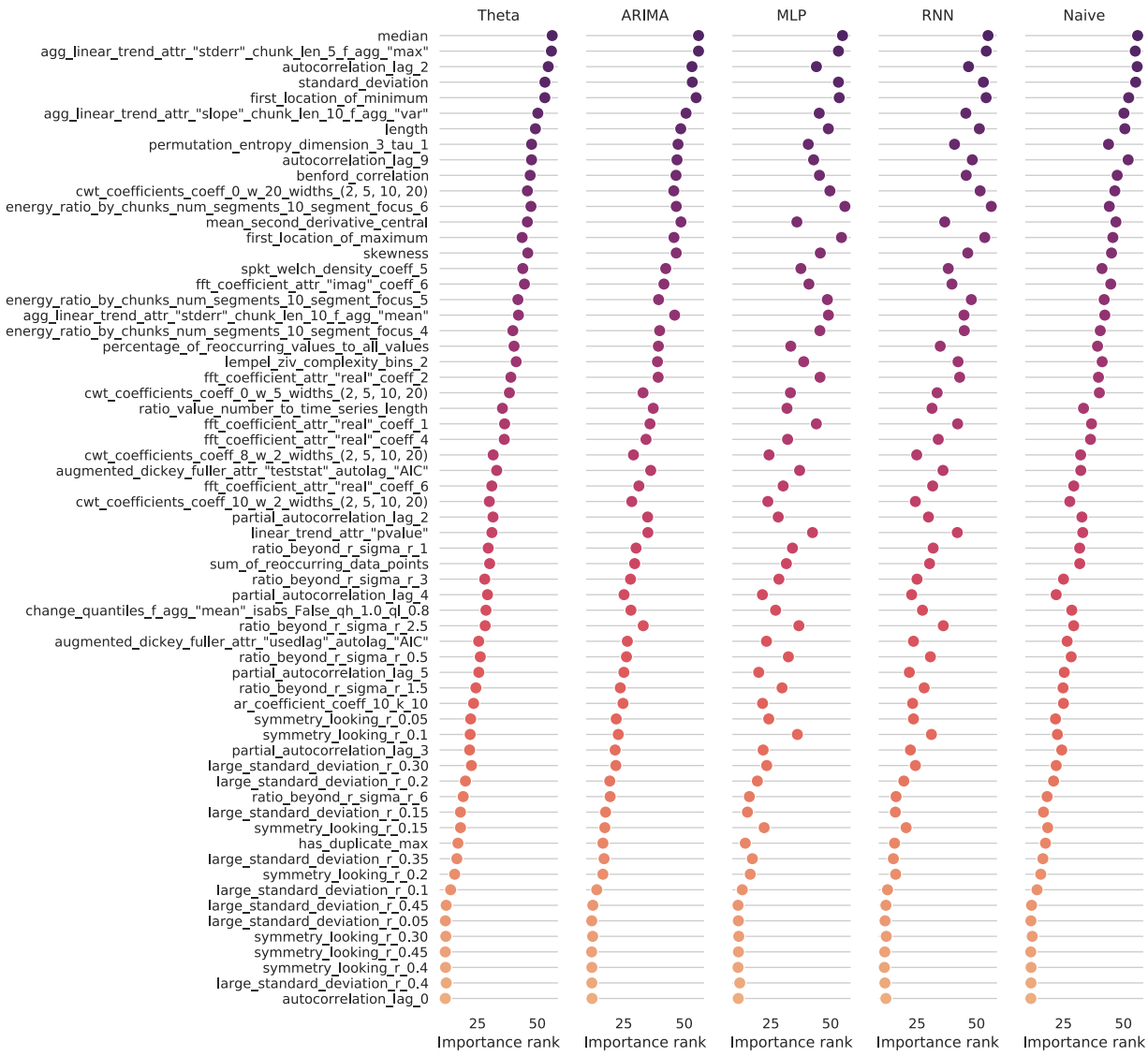

Fig. 14. Feature importance ranks for different forecasting models using permutation based feature importance averaged over all meta-models and 30 runs.

feature importance distributions have different variance evaluated over 30 train/test splits. In this case, the feature importance distributions of the neural network based meta-model has higher variance compared to the tree based models. In this small example, we can see that while feature importance ranking might be similar between SHAP and the permutation based method, the raw values can have prominently different distributions.

4.6. Feature importance for explainability of meta-models

This subsection investigates how feature importance values change when the meta-model is fixed and we change feature importance models and forecasting models. Fig. 12 shows the ranking for all the feature importance methods for XGBoost, aggregated over all 30 splits train/test and all forecasting models. In this case, XGBoost was selected because it has multiple methods determining feature importance. From Fig. 12, we can observe large differences between the importance scores assigned by different feature importance methods for the XGBoost model. In general, SHAP and permutation methods usually produce a similar ordering of the features, based on their importance. On the other hand, the feature importance obtained during the building of the XGBoost model that are constructed from the training data differ. Some features, which are ranked lower with SHAP/permutation feature importance, are assigned high importance by the feature importance methods built during XGBoost construction and vice versa. This signals that even when the same model is used, the feature importance method can have a large impact on how the features are ranked.

4.7. Feature importance with fixed feature importance methods

This subsection describes how keeping the feature importance method fixed but changing the meta-model and forecasting model changes feature importance. Fig. 13 the feature rankings obtained using permutation based feature importance for different meta-models. Ranks are averaged over all forecasting methods and over 30 runs. We can observe that tree-based meta-models assign a similar rank to features. The outlier in this case is the neural network meta-model, where the feature order substantially differs from the rest of the meta-models.

Averaging can also be performed across multiple meta-models with the same feature importance method, to show what foresting algorithms are using what features, independent of the meta-model. Fig. 14 shows feature importance ranks for different forecasting algorithms. Observation can be made that some forecasting algorithms rely on different features than others. One example of this are the MLP and RNN forecasting algorithms where the order of features differs compared to the other three forecasting algorithms. Although we do not

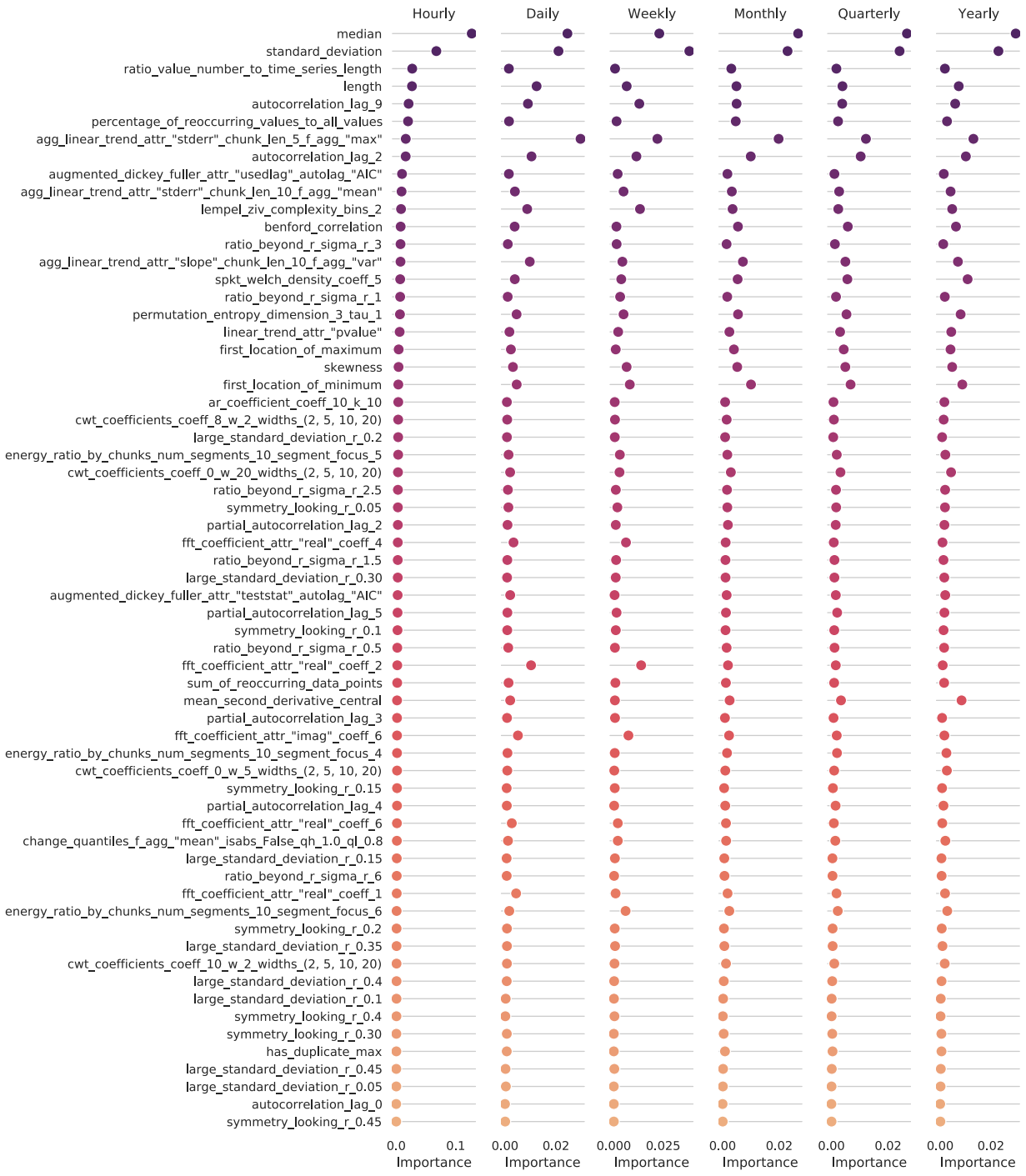

Fig. 15. Feature importance averaged across all forecasting algorithms based on the frequency of the time-series.

go into details about what forecasting algorithm uses what features, one possible explanation for this is that some algorithms can have problems on time-series instances where the mean is not zero or where the standard deviation is not constant over time.

4.8. Time-series and forecasting algorithm types

All the time-series in the dataset also have a frequency with which they were collected (hourly, monthly, yearly, ...). When forecasting models are being built, the frequency of the time-series can have a large effect on model selection and further also on the selection of forecasting model hyperparameters. Therefore, Fig. 15 shows SHAP feature importance based on the frequency of a time-series. The frequency of the individual time-series was available as part of the M4 competition. Note that only SHAP values can be used for this purpose, since they are the only ones that give feature importance on an instance level. We can observe that based on the frequency of the time-series, different features are relevant. If we focus on the feature *agg_linear_trend_attr_"stderr"_chunk_len_5_f_agg_"max"*, we can observe that this feature is not assigned high importance values when dealing with hourly time-series. On the other hand, for the time-series collected daily, this is assigned high importance. A conclusion can be made that when performing algorithm selection, it is beneficial to know the frequency with which the time-series was collected.

Fig. 16. Feature importance ranking based on the type of an forecasting algorithm. Types for each of the forecasting algorithms are available in Appendix C.

Similarly to the previous plot where time-series instances are grouped by frequency of observations, a plot can be made by grouping forecasting algorithms by the type of the model (i.e. statistical, machine-learning, . . .). Fig. 16 shows the feature importance based on the forecasting algorithm type (Statistical, Machine Learning, Hybrid, Combination). Feature importance values are obtained by computing the mean over all feature importance and meta-models. Based on the type of the forecasting algorithm the feature importance values differ slightly, but it is important to note that Hybrid category only has one algorithm while the ML one is an average of three algorithms. Due to this small number of samples in two of the categories, drawing a conclusion is not possible.

Lastly, we visualize the similarities between feature importance values that are produced with different meta-models for different forecasting algorithms. Since feature values are high-dimensional vectors, visualizing them is challenging. One of the techniques for visualizing such vectors is t-SNE embeddings (Van der Maaten & Hinton, 2008) where each high-dimensional vector is projected into two-dimensional space with the goal of preserving distances. We select feature importance vectors from nine different forecasting algorithms obtained with four different meta-models. Since we are repeating this 30 times, we embed all 30 vectors for each combination of meta-model and forecasting model. A few interesting observations can be made from the Fig. 17, showing t-SNE embeddings with cosine metric. Even when the meta-model is fixed, the embeddings calculated based on feature importances can have pronounced differences. For example, comparing feature importance embeddings calculated with the RF meta-model for RNN and ARIMA forecasting algorithms, we can notice a large difference in the dispersions of the embeddings, with embeddings for RNN forecasting models being close together for all 30 splits of the data, while ARIMA embeddings are more dispersed. It is also interesting to note that when tree-based models are used, the embeddings are much more closely grouped together compared to the neural network

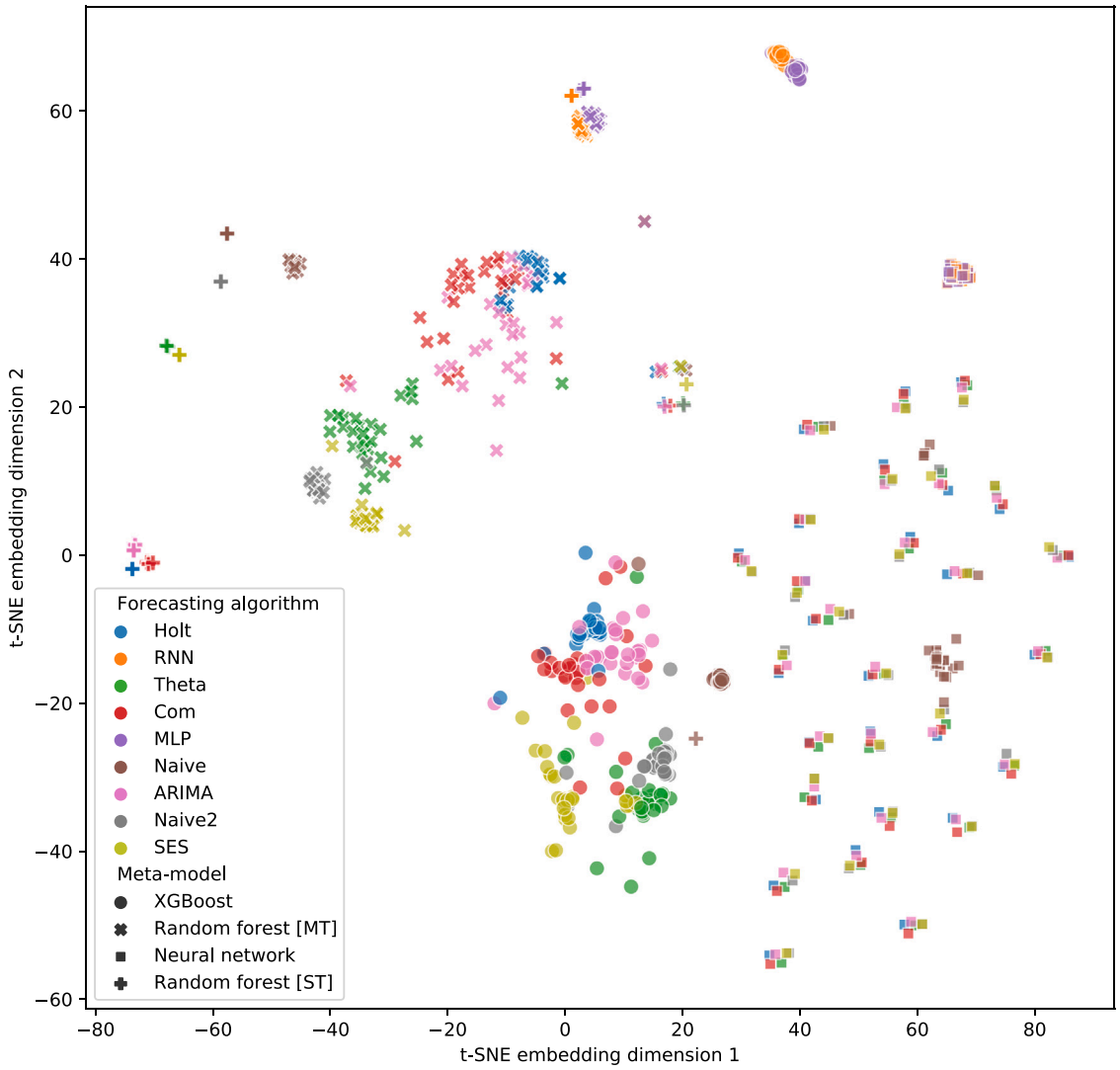

Fig. 17. t-SNE embedding of permutation feature importance values for different forecasting algorithms/meta-learning algorithms.

meta-model. When a neural network is used, the dispersion of embeddings is much larger with clusters of individual forecasting algorithms overlapping.

5. General limitations

This section explains some of the limitations of the methodology for exploring feature importance for predicting the performance of forecasting methods.

The first limitation of the proposed methodology is the selection of the forecasting algorithms. The M4 competition was conducted in 2020 and some of the forecasting methods that were developed later outperform the forecasting methods proposed up to the end of the competition. We limited the scope of the study to only methods that were submitted during the competition. Some later methods that obtain better performance do not report performance for every individual time-series instance included in the competition. Evaluating all the models after 2020 that claim to have improved performance on the M4 dataset would involve constructing all the models together with extensive hyperparameter tuning on models that we are not experts on. For that reason, this study should not serve as an overview of the forecasting models that are available, but as a way of determining feature importance when selecting forecasting algorithms.

Additionally, some of the forecasting methods that were submitted during the competition were trained on multiple time-series instances (aggregated by data frequency). Such is also the case for the three winning forecasting algorithms that first extract patterns from all time-series instances and combine those patterns to forecast future points for an unseen instance. Therefore, it is important to note that when we split the time-series instances in train/test sets and use them to predict performance, some of the forecasting algorithms already introduced information from the test set. Although we realize that such algorithms might bias the results, we still use them in the analysis.

There are numerous libraries for extracting time-series features that could be used as features for a meta-model. We focused only on the libraries that are able to generate features that are interpretable and we, therefore, did not use deep learning-based feature representations since a further analysis of comparing features would not be meaningful. Focusing only on interpretable feature sets, we examined the following possibilities that were most commonly used representations to the best of our knowledge: *hctsa*, *catch22*, *tsfeatures*, *feasts*, *Kats*, *tsfresh*, and *TSFEL*. We ruled out of consideration the *hctsa* feature set since the features are extremely expensive to compute and require a Matlab licence. In Henderson and Fulcher (2021) authors show that features from *catch22*, *feasts*, *Kats*, *tsfeatures*, and *TSFEL* are generally correlated with *hctsa*. We therefore selected *catch22* as a subset of *hctsa* since the features are obtained directly from *hctsa* and are cheaper to compute. *tsfresh* was selected because it includes more features from Fourier transform (i.e. real and imaginary parts, magnitudes, phase angles) while other feature sets include only some summary statistics.

An additional limitation of the study is the selection of error metrics such as sMAPE and MSE. Selecting what metric to use for reporting an error introduces biases. The forecasting error metric was selected based on the following criteria: (i) the sMAPE metric was used in the M4 competition where we obtained performance data. There, the goal was to minimize sMAPE and using another metric for which algorithms were not trained/optimized for might introduce a substantial decrease in performance and thus influence feature importance values, (ii) sMAPE penalizes the relative errors and not the absolute differences which can be large for certain time-series. A more detailed look at the limitations of metrics can be found in Kolassa (2020). However, in future the proposed methodology can be used for any performance metric used as a target, and will further provide insights to the users which features are important depending what is their goal (objective) they want to achieve.

When interpreting feature importance values, one has to also be aware of the limitations of existing methods. Methods that calculate importance during model construction such as RF and XGBoost obtain those features from train data and overfitting the models can distort feature importance values (Strobl, Boulesteix, Zeileis, & Hothorn, 2007). Similar limitations are also present in model agnostic approaches where correlations (Fryer, Strümke, & Nguyen, 2021; Kumar, Venkatasubramanian, Scheidegger, & Friedler, 2020) and model loss (König, Molnar, Bischl, & Grosse-Wentrup, 2021) can have an impact on estimating importance values.

It is also important to note that when calculating Shapley feature importance, only the first 512 time-series instances from the test set were used. This limitation is due to the high computational complexity of SHAP. Similarly, while KNN-regressor appears as one of the meta-models, it was not used for feature importance calculations due to its slow inference time. Due to the time complexity needed to train the meta-models, an exhaustive parameter search was not performed.

Performing this analysis was extremely costly and most of the limitations and trade-offs are therefore related to the available computational power.

6. Conclusion

In this paper, we apply the concept of meta-learning for predicting the performance of time-series forecasting methods, accomplished by training various machine-learning models using features describing characteristics of each time series instance, extracted by the *tsfresh* and *catch22* libraries.

We conduct a comprehensive evaluation of the contribution of the features in the meta-models' predictive performance, using several feature importance methods. A few conclusions follow from the performed analysis. Our results indicate that there are a lot of features that are strongly linearly correlated, however, there is a large subset of *tsfresh* features that are not correlated with any *catch22* features, meaning that catch22 features fail to capture some areas of the time-series landscape. This is also evident from the fact that *tsfresh* features offer better predictive performance compared to the *catch22* features, regardless of the meta-model that is being used, and the fact that there is a negligible difference between the predictive performance obtained by fusing the *catch22* and *tsfresh* feature sets and the performance obtained using only the *tsfresh* features. Looking at the features extracted from log transformed and differenced time-series, we observe high correlations between the features of the log transformed and the raw time series, and lower correlations between the features of the differenced and the raw time series. For *catch22*, fusing the features extracted from the raw, differenced, and log transformed data has a large impact on the predictive performance for all used meta-models, while this difference is lower when using the *tsfresh* features. The analysis of the forecasting algorithm performance revealed strong correlations between the algorithms' performance, meaning that if a single algorithm performs well on a given time-series instance, it is likely that another algorithm will also perform well.

We also find that model-specific features built during the models' construction from the train set can substantially differ from other approaches. The comparison of feature importance between different meta-models using the same feature importance method shows that tree-based models use different features compared to the neural network model. This analysis also reveals that similar feature importance rankings are obtained with SHAP and the permutation importance when the same meta-model is used.

We show that assigning importance to the individual features is a challenging task that depends on the selection of meta-model as well as the method to calculate feature importance values. Regardless of this, there are a few features that are consistently marked as high-importance ones for predicting performance. Such features are for example time-series median, standard deviation, length, autocorrelation at different lags and skewness.

One should also note that feature importance values may vary depending on the forecasting method for which performance is being predicted. Some methods assume that time-series have certain properties such as stationarity and homoscedasticity and when those properties are not met, forecasting performance accuracy can decrease. Features that capture such properties can therefore be assigned high importance for estimating the performance of some forecasting algorithms while being uninformative for others.

The immediate future step for this research is to also include hyperparameters for forecasting algorithms and explore how hyperparameters affect the feature importance. A similar approach can also be extended to multivariate time-series forecasting. Especially interesting would also be applying a similar pipeline for predicting the performance of time-series regression or classification algorithms and assessing the similarity of the importance of features in these tasks, compared to the forecasting algorithm performance prediction task.

CRediT authorship contribution statement

Gašper Petelin: Conceptualization, Methodology, Software, Formal analysis, Writing – original draft, Visualization. **Gjorgjina Cenikj:** Methodology, Formal analysis, Writing – original draft, Visualization, Writing – review & editing, Supervision. **Tome Eftimov:** Conceptualization, Methodology, Writing – original draft, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Funding in direct support of this work: Slovenian Research Agency: young researcher grant No. PR-11263 to GP, research core fundings No. P2-0098 and project No. N2-0239 to TE, and scholarship by the Ad Futura grant for postgraduate study to GC.

Table C.2
Forecasting algorithms, their type (Statistical, Machine Learning, Hybrid, Combination) and the average sMAPE obtained in the competition.

Rank	For. algo.	sMAPE	Type
0	118	0.114	Hybr.
1	072	0.117	Comb.
2	245	0.117	Comb.
3	069	0.118	Comb.
4	237	0.118	Comb.
5	036	0.119	Comb.
6	238	0.119	Comb.
7	260	0.120	Stat.
8	078	0.120	Comb.
9	039	0.121	Comb.
10	243	0.121	Comb.
11	005	0.121	Stat.
12	132	0.122	Comb.
13	251	0.122	Stat.
14	235	0.123	Stat.
15	Theta	0.123	Stat.
16	223	0.124	Comb.
17	227	0.125	Comb.
18	250	0.125	Comb.
19	239	0.125	Comb.
20	104	0.126	Comb.
21	231	0.126	Comb.
22	Com	0.126	Comb.
23	216	0.127	Comb.
24	Damped	0.127	Stat.
25	ARIMA	0.127	Stat.
26	ETS	0.127	Stat.
27	212	0.129	Comb.
28	211	0.129	ML
29	082	0.129	Stat.
30	236	0.131	Comb.
31	SES	0.131	Stat.
32	248	0.132	Comb.
33	030	0.133	Stat.
34	106	0.135	Stat.
35	Naive2	0.136	Stat.
36	009	0.136	Stat.
37	255	0.136	Stat.
38	252	0.136	Stat.
39	256	0.136	Stat.
40	024	0.138	Stat.
41	Holt	0.138	Stat.
42	234	0.138	Comb.
43	043	0.140	Comb.
44	Naive	0.142	Stat.
45	218	0.143	Stat.
46	169	0.144	Comb.
47	253	0.145	Stat.
48	sNaive	0.147	Stat.
49	241	0.149	Comb.
50	191	0.154	Comb.
51	249	0.159	Stat.
52	070	0.159	Comb.
53	126	0.165	Comb.
54	244	0.166	ML
55	091	0.185	ML
56	258	0.191	Stat.
57	219	0.196	ML
58	RNN	0.212	ML
59	MLP	0.217	ML
60	225	0.261	Stat.

Appendix A. Meta models

Random forest is a ensemble learning model (Ho, 1995) where multiple simpler trees are built and predictions are made by combining the predictions of individual trees. Trees are trained with bootstrap aggregating also know as bagging (Breiman, 1996) where each individual decision tree is trained on randomly sampled training data with replacement. Such an approach means that a specific tree can learn on a training set that contains one or more identical instances. Using such a training procedure ensures that trees that are built are more diverse.

Neural networks are parametric models composed of a series of interconnected units, i.e. neurons, each associated with a corresponding weight. The mapping between the input and output data is learned by iteratively adjusting the weights of each neuron through a process referred to as backpropagation, which attempts to find the minimum of the error function in the weight space using the method of gradient descent (Rojas, 1996). Neural networks generally require larger quantities of training data compared to traditional ML models, and are well suited for modeling complex, non-linear relations between the input and output data (Lancashire, Lemetre, & Ball, 2008; Tu, 1996).

The **K-Nearest Neighbors Algorithm** (KNN) is a non-parametric algorithm which, in order to predict the value of a target variable for a given instance, performs aggregation of the values of the target variable of the K instances which are most similar to the new instance, based on some similarity measure. The algorithm uses instance-based, lazy learning, meaning that it is not trained to learn any weights, but instead uses entire instances to predict the output and the entire computation is performed at prediction time.

Extreme Gradient Boosting (XGBoost) (Chen et al., 2015) uses a principle of combining multiple weak learners to obtain better predictions. It is essentially a decision-tree-based ensemble that uses a gradient boosting framework (Friedman, Hastie, & Tibshirani, 2000) and is designed to be extremely computationally efficient and scalable for large amounts of data. XGBoost works especially well on tabular data (Shwartz-Ziv & Armon, 2022)

Appendix B. Feature importance

Feature importance and feature selection are closely linked and are one of the most studied topics in the field of ML. Selecting informative features can drastically decrease the computational cost of training ML models, improve models' performance and even provide feedback to the users on what features are important and what features are not important or redundant.

Feature importance/selection methods can be divided into multiple groups depending on the principle of how importance is determined. One class of methods are filter feature importance methods where feature importance is determined based on the various statistical tests for their correlation with the predicted variable. Such methods are independent of any ML model and are generally used as a prepossessing step to remove uninformative features (Pearson's Correlation, LDA, ANOVA, Chi-Square). Feature importance/selection can also be determined with so called wrapper methods (Forward Selection, Backward Elimination, Recursive Elimination) model is trained and evaluated on a subset of features. Features are then added or removed based on the performance of the trained model. Our primary goal is not to select a subset of features to determine the importance of all the used features. For that reason we primarily focus on feature importance methods that can either be obtained when model is already trained or that are calculated during the model construction and are commonly known as embedded feature importance approaches.

Permutation feature importance determines the importance of a feature by observing the difference in the model's prediction error when values of the feature in question are randomly shuffled. The feature is considered important if shuffling the values increases the model's error. This method can produce reliable results for features that are not correlated or correlations between features are low. A problem arises when the model being evaluated is trained on correlated features. In this case, randomly shuffling only one feature might not produce the correct importance of a feature, since the model will still have access to the information present in the feature in question through its correlated feature, resulting in a less pronounced change in performance, which may not be indicative of the feature's importance. Using correlated features can decrease the importance of correlated features by splitting the importance between them.

Shapley value Additive feature importance methods use an explanation model, which is an interpretable approximation of the original model, to provide interpretability for complex models, such as ensembles or deep neural networks, which may not be innately interpretable. The explanation model is a linear function of binary variables indicating whether each feature is used in the model. The explanation model then assigns an effect score to each feature, such that summing the effects of all features approximates the output of the original model.

SHapley Additive exPlanation (SHAP) (Lundberg & Lee, 2017) values are a type of additive feature importance methods derived using game theory, which satisfy the desirable properties of local accuracy, missingness, and consistency. The feature scores are assigned based on the change in the expected model prediction when conditioning on that feature and explain how to get from the base value that would be predicted if none of the features are known, to the current output obtained for a particular observation.

Random forest The importance of a particular feature used by a RF model can be determined by observing the decrease in the impurity after splitting the data by the feature in question, in each internal node in the trees. Since the RF model internally uses several trees, the overall feature importance is computed by averaging its importance over all trees, and normalizing the importance scores. One of the drawbacks of this approach is that the importance scores are computed on the training set and do not reflect the usefulness of the features to the generalization of the model on the test set. High importance can therefore be obtained for features that the model uses to overfit on the training data. Furthermore, the approach tends to assign higher importance to numerical features and categorical features with high cardinality, and neglect the importance of correlated features.

XGBoost Similarly to random forest, feature importance can also be obtained during the construction of XGboost model. (List options) 'weight': the number of times a feature is used to split the data across all trees., 'gain': the average gain across all splits the feature is used in., 'cover': the average coverage across all splits the feature is used in., 'total_gain': the total gain across all splits the feature is used in, 'total_cover': the total coverage across all splits the feature is used in.

Appendix C. Forecasting algorithm type

Table C.2 show that types for all of the forecasting algorithms that were submitted as a part of the M4 competition together with the average performance of the algorithm.

References

Bagnall, A., Lines, J., Bostrom, A., Large, J., & Keogh, E. (2017). The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *31*(3), 606–660.

Borovykh, A., Bohte, S., & Oosterlee, C. W. (2018). Dilated convolutional neural networks for time series forecasting. *Journal of Computational Finance*.

Brazdil, P. B., Giraud-Carrier, C. G., Soares, C., & Vilalta, R. (2009). Metalearning - applications to data mining. In *Cognitive technologies*.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Chaovalit, P., Gangopadhyay, A., Karabatis, G., & Chen, Z. (2011). Discrete wavelet transform-based time series analysis and mining. *ACM Computing Surveys*, *43*(2), 1–37.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 785–794). New York, NY, USA: ACM, http://dx.doi.org/10.1145/2939672.2939785, URL: http://doi.acm.org/10.1145/2939672.2939785.

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., et al. (2015). Xgboost: extreme gradient boosting. *R Package Version 0.4-2*, *1*(4), 1–4.

Chen, Y., Kang, Y., Chen, Y., & Wang, Z. (2020). Probabilistic forecasting with temporal convolutional neural network. *Neurocomputing*, *399*, 491–501.

Chen, J., Li, K., Rong, H., Bilal, K., Li, K., & Philip, S. Y. (2019). A periodicity-based parallel time series prediction algorithm in cloud computing environments. *Information Sciences*, *496*, 506–537.

Chollet, F., et al. (2015). *Keras*. GitHub, URL: https://github.com/fchollet/keras.

Christ, M., Braun, N., Neuffer, J., & Kempa-Liehr, A. W. (2018). Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package). *Neurocomputing*, *307*, 72–77.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. ArXiv preprint arXiv:1412.3555.

Cohen-Shapira, N., & Rokach, L. (2021). Automatic selection of clustering algorithms using supervised graph embedding. *Information Sciences*, *577*, 824–851, https://doi.org/10.1016/j.ins.2021.08.028. URL: https://www.sciencedirect.com/science/article/pii/S0020025521008288.

Dama, F., & Sinoquet, C. (2021). Time series analysis and modeling to forecast: a survey. ArXiv preprint arXiv:2104.00164.

Dempster, A., Petitjean, F., & Webb, G. I. (2020). ROCKET: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, *34*(5), 1454–1495.

Dempster, A., Schmidt, D. F., & Webb, G. I. (2021). Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining* (pp. 248–257).

Deng, D., Karl, F., Hutter, F., Bischl, B., & Lindauer, M. (2022). Efficient automated deep learning for time series forecasting. ArXiv preprint arXiv:2205.05511.

Eftimov, T., Petelin, G., Cenikj, G., Kostovska, A., Ispirova, G., Korošec, P., et al. (2022). Less is more: Selecting the right benchmarking set of data for time series classification. *Expert Systems with Applications*, *198*, Article 116871.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, *28*(2), 337–407.

Fryer, D., Strümke, I., & Nguyen, H. (2021). Shapley values for feature selection: the good, the bad, and the axioms. *IEEE Access*, *9*, 144352–144360.

Fulcher, B. D., Little, M. A., & Jones, N. S. (2013). Highly comparative time-series analysis: the empirical structure of time series and their methods. *Journal of the Royal Society Interface*, *10*(83), Article 20130048.

Gastinger, J., Nicolas, S., Stepić, D., Schmidt, M., & Schülke, A. (2021). A study on ensemble learning for time series forecasting and the need for meta-learning. In *2021 international joint conference on neural networks* (pp. 1–8). IEEE.

Henderson, T., & Fulcher, B. D. (2021). An empirical evaluation of time-series feature sets. In *2021 international conference on data mining workshops* (pp. 1032–1038). IEEE.

Hewamalage, H., Bergmeir, C., & Bandara, K. (2022). Global models for time series forecasting: A simulation study. *Pattern Recognition*, *124*, Article 108441.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition (Vol. 1)* (pp. 278–282). IEEE.

Hyndman, R., Kang, Y., Montero-Manso, P., Talagala, T., Wang, E., Yang, Y., et al. (2020). Tsfeatures. URL: https://pypi.org/project/tsfeatures/. (Accessed 28 February 2022).

Hyndman, R. J., & Koehler, A. B. (2006). Another look at measures of forecast accuracy. *International Journal of Forecasting*, *22*(4), 679–688.

Kolassa, S. (2020). Why the "best" point forecast depends on the error or accuracy measure. *International Journal of Forecasting*, *36*(1), 208–211.

König, G., Molnar, C., Bischl, B., & Grosse-Wentrup, M. (2021). Relative feature importance. In *2020 25th international conference on pattern recognition* (pp. 9318–9325). IEEE.

Kumar, I. E., Venkatasubramanian, S., Scheidegger, C., & Friedler, S. (2020). Problems with Shapley-value-based explanations as feature importance measures. In *International conference on machine learning* (pp. 5491–5500). PMLR.

Lancashire, L. J., Lemetre, C., & Ball, G. R. (2008). An introduction to artificial neural networks in bioinformatics–application to complex microarray and mass spectrometry datasets in cancer studies. *Briefings in Bioinformatics*, *10*(3), 315–329. http://dx.doi.org/10.1093/bib/bbp012, https://doi.org/10.1093/bib/bbp012.

Li, Y., Li, K., Chen, C., Zhou, X., Zeng, Z., & Li, K. (2021). Modeling temporal patterns with dilated convolutions for time-series forecasting. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *16*(1), 1–22.

Lim, B., Arık, S. Ö., Loeff, N., & Pfister, T. (2021). Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, *37*(4), 1748–1764.

Lubba, C. H., Sethi, S. S., Knaute, P., Schultz, S. R., Fulcher, B. D., & Jones, N. S. (2019). catch22: Canonical time-series characteristics. *Data Mining and Knowledge Discovery*, *33*(6), 1821–1852.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. In *Proceedings of the 31st international conference on neural information processing systems* (pp. 4768–4777). Red Hook, NY, USA: Curran Associates Inc.

Makridakis, S., & Hibon, M. (1979). Accuracy of forecasting: An empirical investigation. *Journal of the Royal Statistical Society: Series A (General)*, *142*(2), 97–125.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2018). The M4 competition: Results, findings, conclusion and way forward. *International Journal of Forecasting*, *34*(4), 802–808.

Makridakis, S., Spiliotis, E., & Assimakopoulos, V. (2020). The M4 competition: 100,000 time series and 61 forecasting methods. *International Journal of Forecasting*, *36*(1), 54–74.

Meade, N. (2000). Evidence for the selection of forecasting methods. *Journal of Forecasting*, *19*(6), 515–535.

Mölder, F., Jablonski, K. P., Letcher, B., Hall, M. B., Tomkins-Tinch, C. H., Sochat, V., et al. (2021). Sustainable data analysis with snakemake. *F1000Research*, *10*.

Montero-Manso, P., Athanasopoulos, G., Hyndman, R. J., & Talagala, T. S. (2020). FFORMA: Feature-based forecast model averaging. *International Journal of Forecasting*, *36*(1), 86–92, M4 Competition. https://doi.org/10.1016/j.ijforecast.2019.02.011. URL: https://www.sciencedirect.com/science/article/pii/S0169207019300895.

Newbold, P., & Granger, C. W. (1974). Experience with forecasting univariate time series and the combination of forecasts. *Journal of the Royal Statistical Society: Series A (General)*, *137*(2), 131–146.

Oreshkin, B. N., Carpov, D., Chapados, N., & Bengio, Y. (2019). N-BEATS: Neural basis expansion analysis for interpretable time series forecasting. ArXiv preprint arXiv:1905.10437.

Patterson, K. (2011). An introduction to ARMA models. In *Unit root tests in time series* (pp. 68–122). Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Rojas, R. (1996). *Neural networks: A systematic introduction*. Berlin, Heidelberg: Springer-Verlag.

Ruiz, A. P., Flynn, M., Large, J., Middlehurst, M., & Bagnall, A. (2021). The great multivariate time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Mining and Knowledge Discovery*, *35*(2), 401–449.

Salinas, D., Flunkert, V., Gasthaus, J., & Januschowski, T. (2020). DeepAR: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, *36*(3), 1181–1191.

Salisu, M., Abdulrahman, S., Adamu, A., Ado, Y., & Rilwan, A. (2017). An overview of the algorithm selection problem. *International Journal of Computer (IJC)*.

Shwartz-Ziv, R., & Armon, A. (2022). Tabular data: Deep learning is not all you need. *Information Fusion*, *81*, 84–90.

Smyl, S. (2020). A hybrid method of exponential smoothing and recurrent neural networks for time series forecasting. *International Journal of Forecasting*, *36*(1), 75–85.

Srinivasan, V., Eswaran, C., et al. (2005). Artificial neural network based epileptic detection using time-domain and frequency-domain features. *Journal of Medical Systems*, *29*(6), 647–660.

Strobl, C., Boulesteix, A.-L., Zeileis, A., & Hothorn, T. (2007). Bias in random forest variable importance measures: Illustrations, sources and a solution. *BMC Bioinformatics*, *8*(1), 1–21.

Talagala, T. S., Hyndman, R. J., & Athanasopoulos, G. (2018). *Meta-learning how to forecast time series*: *Monash econometrics and business statistics working papers 6/18*, Monash University, Department of Econometrics and Business Statistics, URL: https://ideas.repec.org/p/msh/ebswps/2018-6.html.

Talagala, T. S., Li, F., & Kang, Y. (2021). FFORMPP: Feature-based forecast model performance prediction. *International Journal of Forecasting*, https://doi.org/10.1016/j.ijforecast.2021.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0169207021001138.

Tu, J. V. (1996). Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, *49*(11), 1225–1231. http://dx.doi.org/10.1016/s0895-4356(96)00002-9, https://doi.org/10.1016/s0895-4356(96)00002-9.

Tyrrell, B. (2020). 'Algorithm-performance personas' for siamese meta-learning and automated algorithm selection.

Vaiciukynas, E., Danenas, P., Kontrimas, V., & Butleris, R. (2021). Two-step meta-learning for time-series forecasting ensemble. *IEEE Access*, *9*, 62687–62696.

Van der Maaten, L., & Hinton, G. (2008). Visualizing data using t-SNE. *Journal of Machine Learning Research*, *9*(11).

Van Greunen, J., Heymans, A., Van Heerden, C., & Van Vuuren, G. (2014). The prominence of stationarity in time series forecasting. *Studies in Economics and Econometrics*, *38*(1), 1–16.

Vanschoren, J. (2019). Meta-learning. In *Automated machine learning* (pp. 35–61). Springer, Cham.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. *Advances in Neural Information Processing Systems*, *30*.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, *78*(10), 1550–1560.