FFORMA: Feature-based FORecast-model Averaging

Abstract

Needs to be re-written To produce good forecasts of time series, we must first solve the problem of which model to use. Moreover, forecasting accuracy can even be improved by combining different models. We present an automated method for combining forecasting models that outperforms any individual method in a set of the most popular ones in the time series forecasting literature, achieving second position in the M4 Competition. Our approach works in two phases. In the first, we use a dataset of time series to train a meta-model to assign probabilities to the forecasting methods with the goal of minimizing the average forecasting error. In the second phase, we forecast new series by assigning probabilities to methods using our previously trained meta-model, and then combine their individual forecasts using a weighted average with these probabilities. The inputs to the meta-model are features extracted from the series.

Keywords: FFORMS (Feature-based FORecast-model Selection), Time series features, Forecast combination, XGBoost, M4 Competition, Meta-learning

1 Introduction

There are essentially two general approaches for forecasting a time series: (i) generating forecasts from a single model, and (ii) combining forecasts from many models or forecast model averaging. There has been a vast literature on the latter motivated by the seminal work of Bates & Granger (1969) and followed by a plethora of empirical applications showing that combination forecasts are often superior to their individual counterparts (see among others, Clemen 1989, maybe add more here). Combining forecasts is considered a successful way of hedging against the risk of selecting a misspecified model. Of course the main challenge is in selecting an appropriate set of weights with many contributions (see for example, Timmermann 2006) attempting to overcome the so called "forecast combination puzzle".

There has have been various efforts in the literature using time series features combined with meta-learning for forecasting (see for example Prudêncio & Ludermir 2004; Lemke & Gabrys 2010; Kück, Crone & Freitag 2016). (Leave this for Rob as a reminder: not sure how to include

Kang et a. 2017 here). Recently Talagala, Hyndman & Athanasopoulos (2018) proposed FFORMS (Feature-based FORecast Model Selection) a framework that focuses on using time series features combined with meta-learning for forecast-model selection. In this paper we build on this framework posing the learning problem in a way that includes information about the forecast errors produced by the set of candidate forecasting methods and combines these with features extracted from the time series in order to derive a set of weights for forecast combinations. We label this framework, FFORMA (Feature-based FORecast Model Averaging). FFORMA resulted in the second most accurate point forecasts and prediction intervals amongst all competitors in the M4 competition.

The rest of the paper is organized as follows: In Section 2 we describe the FFORMA framework in a very general sense. Section 3 gives the details of our implementation of FFORMA in the M4 Competition for generating both point and interval forecasts. This includes the required preprocessing steps, the set of features and forecast methods, as well as the specific implementation of the meta-learning model. We show empirical evidence on the performance of the approach in Section 4 by quantifying the difference between our proposed learning model and a traditional classifier approach. Section 5 provides some final remarks and conclusions.

2 Methodology

2.1 Intuition and overview of FFORMA

The objective of our meta-learning approach is to derive a set of weights to combine forecasts generated from a *pool of methods* (e.g., naïve, exponential smoothing, ARIMA, etc.). The FFORMA framework requires a set of time series we refer to as the *reference set*. From these the algorithm learns in order to produce forecast combination weights for any given time series. Each time series in the reference set is divided into a training period and a test period. From the training period a set of *time series features* are calculated (e.g., length of time series, strength of trend, etc.). These form the inputs for training the meta-learning model. Moreover, each method in the pool is fitted to the training period and forecasts are generated over the test period. From these a *forecast errors* are computed for each method which form the supervised part for the training of the meta-learning model. These are subsequently summarised using a forecast error measure.

The meta-learning model learns to produce weights for each method in the pool as a function of the features of the series to be forecasted. Once the model is trained, the inductive step consists of assuming that a new series for which forecasts are required comes from a *generating process*

similar to the one that generated the reference set.

A common meta-learning approach is to learn to select the best method in the pool of methods for each series, i.e., the one that produces the smallest forecast error. This approach transforms the problem into a traditional classification problem by setting the individual forecasting methods as the classes and the best method as the target class for each time series. However, this simplification removes relevant information which may be useful in the meta-learning process, such as, which methods produce similar errors to the most accurate one, or which series are the most difficult to forecast and hence have more impact in the total forecast error. In this paper we do not apply this simplification but instead take into account the exact error that each method produces. This information is introduced into the model by posing the problem as finding a function that assigns *probabilities* to each forecasting method, with the objective of minimizing the expected error that would have been produced if the methods were picked at random following these probabilities. This approach is more general than classification, and can be thought of as classification with *per class weights* (the forecast errors) that vary per instance, combined with *per instance weights* that assign more importance to some series.

2.2 Algorithmic description

The operation of the FFORMA framework comprises two phases: (i) the offline phase, in which we train a meta-learner and (ii) the online phase, in which we use the pre-trained meta-learner to identify forecast combination weights. Algorithm 1 presents in detail the Pseudo code of the proposed framework.

Algorithm 1 The FFORMA framework - Forecast combination based on meta-learning.

Offline phase - train the learning model

Inputs:

$\{x_1, x_2, \ldots, x_N\}$: N observed time series forming the reference set.
F : a set of functions for calculating time series features.
M : a set of forecasting methods in the pool, e.g., naïve, ETS, ARIMA, etc.
K : number of iterations in the xgboost algorithm.

Output:

FFORMA meta-learner: A function from the extracted features to a set of M probabilities, one for each forecasting method.

Prepare the meta-data

For $n = 1$ to N:

1: Split x_n into a training period and test period.
2: Calculate the set of features $\boldsymbol{f}_n \in F$ over the training period.
3: Fit each forecasting method $m \in M$ over the training period and generate forecasts over the test period.
4: Calcuate forecast losses L_{nm} over the test period.
5: Meta-data: input features $\boldsymbol{f}_n$ (Step 2), output loss L_{nm} (Step 4).

Train the meta-learner, w

6: Train a learning model based on the meta-data and errors, by minimizing:

$$argmin_w \sum_{n=1}^{N} \sum_{m=1}^{M} w(x_n)_m L_{nm}$$

were L is the loss function.

Online phase - forecast a new time series

Input:

FFORMA classifier from Step 6.

Output:

Forecast the new time series x_{new}.

For x_{new}:

7: Calculate features $\boldsymbol{f}_{new}$ by applying F.
8: Use the meta-learner to produce $\boldsymbol{w}(\boldsymbol{f}_{new})$ an M-vector of probabilities.
9: Compute the individual forecasts of the M forecasting methods in the pool.
10: Combine individual forecasts using $\boldsymbol{w}$ to generate final forecasts.

3 Implementation and Application to M4 Competition

3.1 Reference set

The M4 dataset includes 100,000 real-world time series of yearly, quarterly, monthly, weekly, daily and hourly data. All 100,000 series form the reference set. Each series is split into training period and a test period. The length of the test period for each time series was set to be equal to the forecast horizon set by the competition. When the resulting training period was too short (fewer than two observations I have left this as observations to avoid confusion with train and test period), the size of the test period was reduced. The series that were simply a constant over the training period were eliminated from any further analysis.

3.2 Time series features

Table 1 provides a brief description of the features used in this experiment, F in Algorithm 1. The functions to calculate these are implemented in the `tsfeatures` R package by Hyndman et al. (2018b). Most of the features (or variations of these) have been previously used in a forecasting context by Hyndman, Wang & Laptev (2015) and Talagala, Hyndman & Athanasopoulos (2018) (please refer to these papers for detailed descriptions). The ARCH.LM statistic is calculated based on the Lagrange Multiplier test of Engle (1982) for autoregressive conditional heteroscedasticity (ARCH). For the heterogeneity features 39-42, the time series is first pre-whitened using an AR model resulting to a new series z. A GARCH(1, 1) model is then fitted to the pre-whitened series to obtain the residual series, r. Subsequently, the features are computed.

Features corresponding to seasonal time series only, are set to zero for the case of non-seasonal time series. Note that for the sake of generality, we have not used any of the domain-specific features such as macro, micro, finance, etc., even though this information was available in the M4 data set.

Table 1: *Features used in FFORMA framework.*

	Feature	Description	Non-seasonal	Seasonal
1	T	length of time series	✓	✓
2	trend	strength of trend	✓	✓
3	seasonality	strength of seasonality	-	✓
4	linearity	linearity	✓	✓
5	curvature	curvature	✓	✓
6	spikiness	spikiness	✓	✓
7	e_acf1	first ACF value of remainder series	✓	✓
8	e_acf10	sum of squares of first 10 ACF values of remainder series	✓	✓
9	stability	stability	✓	✓
10	lumpiness	lumpiness	✓	✓
11	entropy	spectral entropy	✓	✓
12	hurst	Hurst exponent	✓	✓
13	nonlinearity	nonlinearity	✓	✓
13	alpha	ETS(A,A,N) $\hat{\alpha}$	✓	✓
14	beta	ETS(A,A,N) $\hat{\beta}$	✓	✓
15	hwalpha	ETS(A,A,A) $\hat{\alpha}$	-	✓
16	hwbeta	ETS(A,A,A) $\hat{\beta}$	-	✓
17	hwgamma	ETS(A,A,A) $\hat{\gamma}$	-	✓
18	ur_pp	test statistic based on Phillips-Perron test	✓	✓
19	ur_kpss	test statistic based on KPSS test	✓	✓
20	y_acf1	first ACF value of the original series	✓	✓
21	diff1y_acf1	first ACF value of the differenced series	✓	✓
22	diff2y_acf1	first ACF value of the twice-differenced series	✓	✓
23	y_acf10	sum of squares of first 10 ACF values of original series	✓	✓
24	diff1y_acf10	sum of squares of first 10 ACF values of differenced series	✓	✓
25	diff2y_acf10	sum of squares of first 10 ACF values of twice-differenced series	✓	✓
26	seas_acf1	autocorrelation coefficient at first seasonal lag	-	✓
27	sediff_acf1	first ACF value of seasonally-differenced series	-	✓
28	y_pacf5	sum of squares of first 5 PACF values of original series	✓	✓
29	diff1y_pacf5	sum of squares of first 5 PACF values of differenced series	✓	✓
30	diff2y_pacf5	sum of squares of first 5 PACF values of twice-differenced series	✓	✓
31	seas_pacf	partial autocorrelation coefficient at first seasonal lag	✓	✓
32	crossing_point	number of times the time series crosses the median	✓	✓
33	flat_spots	number of flat spots, calculated by discretizing the series into 10 equal sized intervals and counting the maximum run length within any single interval	✓	✓
34	nperiods	number of seasonal periods in the series	-	✓
35	seasonal_period	length of seasonal period	-	✓
36	peak	strength of peak	✓	✓
37	trough	strength of trough	✓	✓
38	ARCH.LM	ARCH LM statistic	✓	✓
39	arch_acf	sum of squares of the first 12 autocorrelations of z^2	✓	✓
40	garch_acf	sum of squares of the first 12 autocorrelations of r^2	✓	✓
41	arch_r2	R^2 value of an AR model applied to z^2	✓	✓
42	garch_r2	R^2 value of an AR model applied to r^2	✓	✓

3.3 Pool of forecasting methods

We consider nine methods implemented in the `forecast` package in R (Hyndman et al. 2018a) for the pool of methods, P in Algorithm 1. They are (the specific R calls for fitting the models are given):

i. naïve (`naive`);
ii. random walk with drift (`rwf` with drift=TRUE);
iii. seasonal naïve (`snaive`).
iv. theta method (`thetaf`);
v. automated ARIMA algorithm (`auto.arima`);
vi. automated exponential smoothing algorithm (`ets`);
vii. TBATS model (`tbats`);
viii. STLM-AR Seasonal and Trend decomposition using Loess with AR modeling of the seasonally adjusted series (`stlm` with model function `ar`);

In all cases, the default settings are used. Further, in the case of an error when fitting the series (e.g. a series is constant), the `snaive` forecast method is used instead.

3.4 Forecast error measure

We don't need this at all as we stipulate the L below. Let's remove this.

The forecasting error measure, E in Algorithm 1, is adapted from the Overall Weighted Average error described in the M4 competition guide, which adds together the Mean Absolute Scaled Error and the symmetric Mean Absolute Percentage Error. For each series and method, the Mean Absolute Scaled Error and the symmetric Mean Absolute Percentage Error are divided by the respective error of the Naive 2 method *over all series in the dataset* (i.e. MASE by the average MASE of Naive 2), and then added.

3.5 Metal-learning model implementation

Pablo are you happy with this? We use the gradient tree boosting model of `xgboost` as the underlying implementation of the learning model. This is a state of the art computationally efficient model that has shown good performance in structure based problems (Chen & Guestrin 2016). The great advantage here is that we are able to customise the model to fit our objective function.

again, find a better name?

3.5.1 The xgboost objective function

The basic xgboost algorithm produces numeric values from the features, one for each forecasting method in our pool. We apply the softmax transform to these values prior to computing the the objective function which includes the forecasting errors shown in Algorithm 1 Step 6. This implemented as a *custom objective function* in the xgboost framework.

xgboost requires a gradient and hessian of the objective function to fit the model. The *correct* hessian is prone to numerical problems that need to be addressed for the boosting to converge. This is a relative common problem and one simple fix is to use an upper bound of the hessian by clamping its small values to a larger one. We computed a different upper bound of the hessian by removing some terms from the correct hessian. Although both alternatives converge, the later works faster, requiring less boosting steps to converge. This not only increases the computational efficiency, it also generalizes better due to a less complex set of trees produced in the final solution. I am not sure about the above. I leave to Rob to evaluate.

The general paremeters of the meta-learning in Algorithm 1 are set to:

- $p(\boldsymbol{f}_n)_m$ is the output of the xgboost algorithm, one value for each forecasting method m, for the features extracted from series x_n.
- $w(x_n)_m = \frac{exp(p(\boldsymbol{f}_n)_m)}{\sum_m exp(p(\boldsymbol{f}_j)_m)}$ is the transformation to probabilities of the xgboost output by applying the softmax transform.
- L_{nm} is the OWA error measure.
- $\bar{L}_n = \sum_{m=1}^{M} w(x_n)_m L_{nm}$ is the weighted average loss function, Step 6 of Algorithm 1.
- I am not sure at all how the following two are derived. Rob was not sure either. Pablo can you please help us understand here. $G_n = \frac{\partial \bar{L}}{\partial w(x_n)} = w_n(L_n - \sum_m L_m w_m)$ is the gradient of the loss function.
- The hessian H_n is approximated by our upper bound $\hat{H}_n$:

$$H_n = \frac{\partial G_n}{\partial w_n} \approx \hat{H}_n = w_n(L_n(1 - w_n) - G_n)$$

The functions G and $\hat{H}$ are passed to the xgboost algorithm to minimize our objective function L.

3.5.2 Hyperparameters

The results of xgboost are particularly dependent on its hyper-parameters such as learning rate, number of boosting steps, maximum complexity allowed for the trees or subsampling sizes.

We limit the hyper-parameter search space based on some initial results and rules of thumb and explore it using Bayesian optimization (implemented in the `rBayesianOptimization` R package Yan 2016) measuring performance on a 10% holdout version of the reference set. We picked the simplest hyper-parameter set from the top solutions of the exploration.

3.6 Prediction Intervals

For each series x_{new} we use as the center of the interval the point forecast produced by our meta-learner. Then the 95% bounds of the interval are generated by a linear combination of the bounds of three forecasting methods: naïve, theta and seasonal naïve Pablo have we distinguished between seasonal and non-seasonal series. The coefficients for the linear combination are calculated in a data driven way over the M4 database. The complete procedure is as follows:

Pablo I have re-written the following. Can you please check below, edit and delete the repetition from further below.

1. We divide the M4 dataset into two parts: A and B. We train the FFORMA learner using the training periods of the series in part A and produce point forecasts over the series of part B, and vice versa.

2. We compute the 95% *predicion radius* for the naïve, theta, and seasonal naïve methods. This is the difference between the 95% upper bound and the point forecast for each forecast horizon.

3. For each forecast horizon we find the coefficients that minimize the MSIS of the interval, as defined in the M4 Competition guide, with the FFORMA point forecast as the center and a linear combination of the radiuses of naïve, theta, seasonal naïve forecasts as the interval. The minimization is done by gradient descent over the test period of the series.

This procedure produces a set of three coefficients for each prediction horizon in the M4 dataset and these coefficients will be the same independently of the series we want to forecast. Unlike the point forecast, these coefficients are not restricted to be probabilities.

1. Using the training period of each series:
    a. Compute the point forecast using FFORMA. At this stage we do not use the full dataset for training FFORMA, since it would produce too acurate forecasts in the same dataset, leading to very small prediction intervals. we divide the M4 dataset in two parts, A and B. We train FFORMA on part A and use it to produce the point forecasts over the series of part B, and the reverse.

   b. Compute the 95% *predicion radius* for the thetaf, snaive and naive, this is the difference between the 95% upper bound and the point forecast for each forecast horizon.

2. For each forecast horizon. Find the coefficients that minimize the MSIS error, (defined in the M4 Competition guide), of the interval with the FFORMA point forecast as center and a linear combination of the radiuses of thetaf, snaive and naive as radius. The minimization is done by gradient descent over the test period of the series.

This procedure produces a set of three coefficients for each prediction horizon in the M4 dataset and these coefficients will be the same independently of the series we want to forecast. *Unlike the point forecast, these coefficients are not restricted to be probabilities.*

4 Results

In order to evaluate the impact of our contribution we compared the average forecast error produced by our proposed combination FFROMA approach to a reference implementation of a traditional classification approach, in which a model is trained to select the "best method" to generate the final forecast. We use the same set of features, the same pool of forecasting methods, the same underlying implementation (xgboost), and in both approaches we perform hyperparameter search prior to fitting the model using bayesian optimization. This ebales us to isolate the model differences. Using the M4 dataset for point forecast, our approach produces a **relative improvement of 10%** over the reference, measured as $\frac{classifier\ error}{proposed\ method\ error}$. Pablo can you elaborate a little here. A couple more sentences shoud be enough. It is not exactly clear to me what this does. What you have on the comments below are very interesting and I think really worth writing up in order to understand/evaluate FFORMA. I think it would really be worth putting all this work in a seprate paper. I cannot see why the IJF would not be interested in it.

5 Discussion and Conclusions

We have presented an algorithm for forecasting using weighted averaging of a set of models. The objective function of the learning model assigns probabilities to forecasting methods in order to minimize the forecasting error that would be produced if we picked the methods at random using these probabilities. This contrasts with how the final forecasts are produced, which is a weighted average, not a selection. These weights can however be used as model selection algorithm, if one picks the methods receiving the largest weight. This can be useful for interpretability or computational reasons, at the cost of forecasting performance.

One advantage of the approach is that its form is not independent of the forecasting error

measure. Forecasting errors enter the model as additional precalculated constants. This allows FFORMA to adapt to arbitrary error measures when models that directly minimize them would be restricted, e.g. our approach can be applied to non-differentiable errors.

We explored minimizing the weighted average of the forecasts, but the results did not improve over the simple version.

The source code for FFORMA is available at https://github.com/robjhyndman/M4metalearning.

References

Bates, JM & CWJ Granger (1969). The Combination of Forecasts. *Journal of the Operational Research Society* **20**(4), 451–468.

Chen, T & C Guestrin (2016). Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. ACM, pp.785–794.

Clemen, R (1989). Combining forecasts: a review and annotated bibliography with discussion. *International Journal of Forecasting* **5**, 559–608.

Engle, RF (1982). Autoregressive conditional heteroscedasticity with estimates of the variance of United Kingdom inflation. *Econometrica: Journal of the Econometric Society*, 987–1007.

Hyndman, RJ, E Wang & N Laptev (2015). Large-scale unusual time series detection. In: *Data Mining Workshop (ICDMW), 2015 IEEE International Conference on*. IEEE, pp.1616–1619.

Hyndman, R, G Athanasopoulos, C Bergmeir, G Caceres, L Chhay, M O'Hara-Wild, F Petropoulos, S Razbash, E Wang & F Yasmeen (2018a). *forecast: Forecasting functions for time series and linear models*. R package version 8.3. `http://pkg.robjhyndman.com/forecast`.

Hyndman, R, E Wang, Y Kang & T Talagala (2018b). *tsfeatures: Time Series Feature Extraction*. R package version 0.1. `https://github.com/robjhyndman/tsfeatures`.

Kück, M, SF Crone & M Freitag (2016). Meta-Learning with Neural Networks and Landmarking for Forecasting Model Selection - An Empirical Evaluation of Different Feature Sets Applied to Industry Data Meta-Learning with Neural Networks and Landmarking for Forecasting Model Selection. *International Joint Conference on Neural Networks*, 1499–1506.

Lemke, C & B Gabrys (2010). Meta-learning for time series forecasting and forecast combination. *Neurocomputing* **73**(10). Subspace Learning / Selected papers from the European Symposium on Time Series Prediction, 2006–2016.

Prudêncio, R & T Ludermir (2004). Using machine learning techniques to combine forecasting methods. In: *Australasian Joint Conference on Artificial Intelligence*. Springer, pp.1122–1127.

Talagala, TS, RJ Hyndman & G Athanasopoulos (2018). Meta-learning how to forecast time series. *Technical Report 6/18, Monash University.*

Timmermann, A (2006). "Forecast Combinations". In: *Handbook of economic forecasting,pp. 135-196. Chapter 4*. Ed. by Graham Elliot and Clive W. J. Granger, Timmermann Allan. Amsterdam, North-Holland.

Yan, Y (2016). *rBayesianOptimization: Bayesian Optimization of Hyperparameters*. R package version 1.1.0. https://cran.r-project.org/web/packages/rBayesianOptimization/index.html.