FFORMA: Feature-based FORecast-model Averaging

Abstract

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: FFORMA (Feature-based FORecast-model Averaging), FFORMS (Feature-based FORecast-model Selection), Time series features, Forecast combination, XGBoost, M4 Competition

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, **clemen1989combining**,). 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, Timmermann (2006)) attempting to overcome the so called "forecast combination puzzle".

maybe add more here

Recently, Talagala, Hyndman & Athanasopoulos (2018) have explored the idea of using time series features combined with meta-learning for forecast-model selection; a framework labelled as FFORMS (Feature-based FORecast Model Selection). In this paper we build on this framework

Pablo to add on the literature on ensemble meth-

posing the learning problem in a way that includes information about the forecast errors produced by the set of forecasting methods considered 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). The proposed FFORMA framework has been used over the course of the M4 competition and resulted in the second most accurate point forecasts and prediction intervals amongst all competitors.

Pablo please review this

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 Overview / Intuition 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, ETS, 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 error measure* is computed for each method. These form the labels?outputs? for the training of the meta-learning model.

The meta-learning model learns to produce weights for each method in the pool as a function of the features of the series. Once the model is trained, the inductive step consist of assuming that a new series for which we require forecasts 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, enabling the use of existing classification algorithms. This is what was implemented by Talagala, Hyndman & Athanasopoulos (2018) and formed the basis of the FFORMS approaph.

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 more *difficult* than others. In this paper we advance the FFORMS framework by not applying this simplification and taking into account the exact error that each method produces in each series, instead of just considering the one that produces the minimum error.

This information is introduced into the model by posing the problem as finding a function that assigns *probabilities* to each forecasting method for each series, with the objective of minimizing the expected error that is produced if the methods were picked at random following these probabilities.

Our approach can be easily transformed into a classification exercise by selecting the method corresponding to the largest probability produced by our model. If we allow a sufficiently rich hypothesis space to fit freely, we also end up assigning probability one to the method with the smallest error. Hence our approach can also be seen as a classification exercise, but with *per class weights* (the forecasting errors) that vary per instance, combined with *per instance weights* that assign more importance to some series.

Not sure what difficult means here and how it is used - difficult to forecast? and then?

I think a lot of this - even all of it - can go if we need more space.

2.2 Algorithmic description

The operation of the FFORMA framework comprises two phases: (i) offline phase, in which we train a meta-learner and (ii) 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

Given:

$\{X_1, X_2, \ldots, X_N\}$: N observed time series forming the reference set.
P : the pool of K forecasting methods, e.g., naïve, ETS, ARIMA, etc.
F : the set of functions for calculating time series features.
E : A forecast error measure, e.g., MASE, sMAPE, etc.
L : Loss function.
M : number of iterations in the xgboost algorithm.

Output:

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

Prepare the meta-data

For $j = 1$ to N:

1: Split X_j into a training period and test period.
2: Calculate the set of features $\boldsymbol{f}_j$ over the training period by applying F.
3: Fit each method in P over the training period and generate forecasts over the test period.
4: Calcuate forecast error measure e_{jk} over the test period for each method $k = 1, \ldots, K$ in P.
5: Meta-data: input features $\boldsymbol{f}_j$ (step 2), output errors: $\boldsymbol{e}_{jk}$ (step 5).

Train the meta-learner

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

$$argmin_f \sum_{j=1}^{N} \sum_{k=1}^{K} f(x_j)_k e_{jk}$$

7: Meta-learner.

Online phase - forecast a new time series

Given:

FFORMA classifier from step 7 .

Output:

Forecast for the new time series X_{new}.

8: For X_{new} calculate features $\boldsymbol{f}_{new}$ by applying F.
9: From the features, use the meta-learner to produce $\mathbf{w}$ the vector of probabilities.
10: Compute the individual forecasts of the methods in P for X_{new}
11: Combine individual forecasts using $\boldsymbol{w}$ to generate final forecasts for X_{new}.

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 tahn 2 observations), 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 42 features used in this experiment. The functions to calculate these are implemented in the `tsfeatures` R package by Hyndman et al. (2018b). Most of the features have been previously used by Hyndman, Wang & Laptev (2015) and Talagala, Hyndman & Athanasopoulos (2018). The last five heterogeneity features are calculated as follows. First, the time series is 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, features 39-42 are computed. The ARCH.LM statistic is calculated based on Lagrange Multiplier test of **engle1982autoregressive** for autoregressive conditional heteroscedasticity (ARCH). A detailed description of the other features is provided in Talagala, Hyndman & Athanasopoulos (2018). The features corresponding to seasonal time series only, are set to zero for the case of non-seasonal time series. Note that, 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.

Should we add here in an effort to be as general as possible.

3.3 Pool of forecasting methods

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

i) automated ARIMA algorithm (`auto.arima`),

ii) automated ETS algorithm (`ets`),

iii) feed-forward neural network with a single hidden layer fitted to the lags. The number of lags is automatically selected (`nnetar`), iv) random walk with drift (`rwf` with drift=TRUE),

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	✓	✓

iv) TBATS model (`tbats`),

v) Theta method (`thetaf`),

vi) naive (`naive`),

vii) STLM-AR Seasonal and Trend decomposition using Loess with AR modeling of the seasonally adjusted series (`stlm` with model function `ar`) and

viii) seasonal naive (`snaive`).

In all cases, the default setting is 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.

Forecast-error measure calculation: OWA

3.4 Metal-learning model implementation

We choose the gradient tree boosting model of **xgboost** (Chen & Guestrin (2016)) as the underlying implementation of the learning model due to following reasons:

1. Ability to customize the model to fit our objective function
2. Computational efficiency: The size of the M4 data set requires a efficient model
3. Good performance in structure based problems

again, find a better name?

3.4.1 The xgboost objective function

The basic boost algorithm produces numeric values from the features, one for each forecasting method in our pool. We apply the soft max transform to these values prior to computing the the objective function shown in Algorithm **??** step 7, implemented as a custom objective function.

boost fits the model by gradient boosting, for which requires a gradient and hessian of the objective function. In our case, the gradient is the direct gradient of the objective, but the *correct* hessian is prone to numerical problems that need to be fixed for boost to converge. This is a relative common problem and one simple fix is to use an upper bound of the hessian by clamping small values to a larger one. In our case, we compute an alternate upper bound of the hessian by removing some terms from the *correct* hessian. Although both alternatives converge, our approach 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.

The functions involved in computing the objective are the following:

- $y(x)$ is the output of the xgboost algorithm
- $p_j = \frac{e^{y(x)_j}}{\sum_k e^{y(x)_k}}$ is the transformation to probabilities by applying the softmax transform.
- $L = \sum p_j e_j$ is the loss of the function p.

- $G_j = \frac{\partial L}{\partial p_j} = p_j(e_j - \sum_k e_k p_k)$

$$H_j = \frac{\partial G_j}{\partial p_j} \approx \hat{H}_j = p_j(e_j(1 - p_j) - G_j)$$

3.4.2 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. In our case 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 most regularized hyper-parameter set from the top solutions of the exploration.

3.5 Prediction Intervals

The M4 Competition also featured a sub competition over the accuracy of prediction intervals. For this part, we used a different approach. In order to compute the intervals we used the point forecast produced by our meta-learner as the center of the interval and computed the 95% bounds of the interval by a linear combination of the bounds of three forecasting methods: Thetaf, SNAIVE and NAIVE methods. The coefficients for the linear combination were calculated in a data driven way also over the M4 database. The procedure is as follows:

maybe too verbose/redun

1. For the training period of each series in the data set:
    1. Compute the point forecast of the meta-learning.
    2. 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 horizon.
2. For each forecasting horizon required in the data set:
    1. Find the coefficients that minimize the MSIS error of the interval with the meta-learning 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 at 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, the optimization is unrestricted.* In order to prevent overfitting for the center of the intervals, the training part of the M4 dataset

was divided in two halves, and our metalearning approach was trained in one half and applied to the other.

4 Results

We measure the impact of our main contribution by comparing the average forecasting error produced by our proposed combination model to a reference implementation of a traditional classification approach, in which a model is trained to pick the best method, which will then be used for 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 way, we isolate the model differences. In the M4 competition dataset for point forecast, our approach produces a **relative improvement of 10%** over the reference, measured as $\frac{classifier\ error}{proposed\ method\ error}$.

5 Discussion and Conclusions

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.

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.

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.