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 combining their individual forecasts using a weighted average. The inputs to this 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 to forecast a time series: i) use of single model and ii) combination forecast or forecast model averaging. There is a growing consensus that the combination forecast is a way of improving forecast accuracy. Empirical work often provides evidence that the combination of forecasts resulting from several candidate forecasting models are often superior to their individual forecasts. Combining forecasts across different models is considered as a way of reducing the risk of selecting an inappropriate model. However, the main challenge in forecast combination is the selection of appropriate set of weights.

Granger (1969), was the first to forward the idea of "the combination of forecasts". Since then many approaches have been proposed to derive weights for forecast combination (Timmerman, 2006). Simple averages(ref_clemen1989), regression-based approaches, ... to name a few.

Recently, a few researchers have explored the use of time series features in selecting the most appropriate forecasting method. However, the use of time series features to derive weights for forecast combination has been rarely addressed in the literature. In this paper we propose a general framework to obtain weights for forecast combination based on features computed from the time series. We call this framework FFORMA(Feature-based FORecast Model Averaging). The proposed FFORMA framework has been used over the course of M4 competition and placed in second in forecast accuracy in both point forecasts and prediction intervals.

This approach has been influenced by the work of Talagala, Hyndman & Athanasopoulos (2018) and is related to the previous work by ref_prudencio in which they use machine learning techniques to define weights for the linear combination of forecasts.

The rest of the paper is organized as follows:

In Section 2 we describe the FFORMA framework in a general sense, without specifying the set of features, forecasting methods and learning model implementation. This section contains the main contribution: posing the learning problem in a way that includes the information about the errors produced by each forecasting method and combines it with the features extracted from the series.

Section 3 gives the details of our implementation of FFORMA which achieved second place in the M4 Competition. The required preprocessing steps, the sets of features and forecast methods, as well as the specific implementation of the meta-learning model. *The method to produce the prediction intervals for the M4 competition is also explained in this section.*

We show empirical evidence on the good performance of the approach in Section 4 by quantifiying the difference between our proposal and a traditional classifier approach using the same features and underlying learning implementation.

2 Methodology

2.1 Overview / Intuition

The objective of our meta-learning approach is to combine a set of individidual forecasting methods (e.g. ARIMA, exponential smoothing) to produce more accurate forecasts. This combination is a weighted average of the forecasts of individual methods. Our framework requires a dataset of time series, called the *reference set*, from which we learn to produce these weights for any

given time series.

We start from the reference set and a fixed set of forecasting methods, called the *pool of methods*. The forecasting errors of each method in the pool are calculated for each time series in the reference set, using the true future values of the series if available or by dividing the series into a training period and test period by applying temporal holdout.

The inputs for training the meta-learning model are:

- features (e.g. length, autocorrelation coefficients,...) extracted from the training period of the series.
- The errors produced by the forecasting methods.

The model will learn to produce weights for each method in the pool as a function of the features of the series. Once the meta-learning model is trained, the inductive step consist of assuming that a new series we want to forecast comes from a *generating process* similar to the one that generated the reference set.

One common meta-learning approach is to learn to select the best method in the pool for each series, the one that produces the least forecasting error. This approach transforms the problem into a traditional classification one 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. However, this simplification removes relevant information which may be useful in the metalearning process, such as which methods produce similar errors to the best one for a series, or which series are more *difficult* than others. We do not apply this simplification and take into account the exact error that each method produces in each series, instead of just considering which one 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 just by picking the method with the largest probability produced by our model. If we allow a sufficiently rich hyphotesis space to fit freely, we also end up assigning probability one to the methods with the least amount of error. 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.

2.2 Algorithmic description

The FFORMA framework has four main components:

1. The set of forecasting methods, the pool of methods.
2. The set of features to be extracted from the time series
3. A training set of time series, the reference set.
4. The forecasting error measure, such as mean squared error.

The FFORMA framework works in two phases: an **offline** phase, when the meta-learned is trained and an **online** phase, when forecasts are produced. Algorithm 1 describes the two phases.

3 Implementation and Application to M4 Competition

3.1 Data preprocessing

For the M4 competition, we used the whole M4 dataset as the reference set. The M4 competition database consists of 100,000 real-world time series of yearly, quarterly, monthly, weekly, daily and hourly data. To divide the series into training and testing period, we used the forecast horizon specified in the database as the size of the temporal holdout. When the training period resulted too short (less than 2 periods), the size of the temporal holdout was reduced. Some series were removed because they were constant after removing the holdout part.

3.2 Time series features

We used a set of 42 features in for the model. The functions to calculate these features are implemented in `tsfeatures` R package by xxx. These features have been previously used by Talagala, Hyndman & Athanasopoulos (2018) and Hyndman, Wang & Laptev (2015). Table xxx provides a brief description of features used in this experiment. A detailed description of these features is provided in Talagala, Hyndman & Athanasopoulos (2018).

Question: calculation of features for time series with multiple seasonality, and short time series?

1. *x_acf* The first autocorrelation coefficient of the series.
2. *x_acf10* The sum of the squared first ten autocorrelation coefficients of the series.
3. *diff1_acf1* The first autocorrelation coefficient of the first differenced series

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

Offline phase - train the learning model

Given:

$O = \{X_1, X_2, \ldots, X_n\}$: the collection of N observed time series, the reference set.

P : Set of K forecasting algorithms such as ARIMA, ETS, SNAIVE, etc.

F : the set of functions to calculate time series features.

E : A forecasting error measure such as Mean Squared Error.

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 for the training period by applying F.
3: Fit the models in P to the training period.
4: Calculate forecasts for the test period from each model.
5: Calculate forecast error measure E over the test period for all models in P.
6: Meta-data: input features x_j (step 2), output errors: e_j (step 5).

Train the meta-learner

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

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

8: meta-learner.

Online phase - forecast a new time series

Given:

FFORMA classifier from step 8 .

Output:

Forecast for the new time series X_{new}.

9: For X_{new} calculate features x_{new} by applying F.
10: From the features, use the meta-learner to produce w the vector of probabilities.
11: Compute the individual forecasts of the methods in P for X_{new}
12: Compute the final forecast by weighted average using w and the individual forecasts.

4. *diff1_acf10* The sum of the squared first ten autocorrelation coefficients of the first differenced series.
5. *diff2_acf1* The first autocorelation coefficient of the twice-differenced series.
6. *diff2_acf10* The sum of squared fist ten autocorrelation coefficients of the original series.
7. *seas_acf1* The autocorrelation coefficient at the first seasonal lag. If the series is non seasonal, this feature is set to 0.
8. *ARCH.LM* A statistic based on the Lagrange Multiplier test of Engle (1982) for autoregressive conditional heteroscedasticity.The R^2 of an autoregressive model of 12 lags applied to

x^2 after the its mean has been subtracted.

9. *crossing_point* The number of times the time series crosses the median.
10. *entropy* The spectral entropy of the series. $H_s(x_t) = -\int_{-\Pi}^{\Pi} f_x(\lambda) \log f_x(\lambda) d\lambda$ where the density is normalized so $\int_{-\pi}^{\pi} f_x(\lambda) d\lambda = 1$
11. *flat_spots* The number of flat spots in the series, calculated by discretizing the series into 10 equal sized intervals and counting the maximung run length within any single interval.
12. *arch_acf* After the series is pre-whitened using an AR model and squared, the sum of squares of the first 12 autocorrelations.
13. *garch_acf* After the series is pre-whitened using an AR model, a GARCH(1,1) model is fitted to it and the residuals are calculated. The sum of squares of the first 12 autocorrelations of the squared residuals.
14. *arch_r2* After the series is pre-whitened using an AR model and squared, the R^2 value of an AR model applied to it.
15. *garch_r2* After the series is pre-whitened using an AR model, a GARCH(1,1) model is fitted to it and the residuals are calculated. The sum of squares of the first 12 autocorrelations of the squared residuals.
16. *alpha* α The smoothing parameter for the level in a ets(A,A,N) model fitted to the series.
17. *beta* β The smoothing parameter for the trend in a ets(A,A,N) model fitted to de series.
18. *hurst* The hurst coefficient indicating the level of fractional differencing of a time series.
19. *lumpiness* The variance of the variances based on a division of the series in non-overlapping portions. The size of the portions if the frequency of the series, or 10 is the series has frequency 1.
20. *nonlinearity* A nonlinearity statistic based on Terasvirta's nonlinearity test of a time series.
21. *x_pacf5* The sum of squared first 5 partial autocorrelation coefficients of the series.
22. *diff1x_pacf5* The sum of squared first 5 partial autocorrelation coefficients of the first differenced series.
23. *diff2x_pacf5* The sum of squared first 5 partial autocorrelation coefficients of the twice differenced series.
24. *seas_pacf* The partial autocorrelation coefficient at the first seasonal lag. 0 if the series is non seasonal.
25. *nperiods* The number of seasonal periods in the series.
26. *seasonal_period* The length of the seasonal period.

27. *trend* In a STL decomposition of the series with r_t the remainder series and z_t the deseasonalized series: $max[0, 1 - Var(r_t)/Var(z_t)]$
28. *spike* In a STL decomposition of the series with r_t the remainder series, the variance of the leave one out variances of r_t
29. *linearity* In a STL decomposition of the series with T_t the trend component, a quadratic model depending on time is fitted: $T_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \epsilon_t$. *linearity* is β_1.
30. *curvature* In a STL decomposition of the series with T_t the trend component, a quadratic model depending on time is fitted: $T_t = \beta_0 + \beta_1 t + \beta_2 t^2 + \epsilon_t$. *curvature* is β_2.
31. *e_acf1* The first autocorrelation coefficient of the remainder series in an STL decomposition of the series.
32. *e_acf10* The sum of the first 10 squared autocorrelation coefficients of the remainder series in an STL decomposition of the series.
33. *seasonal_strength* In a STL decomposition of the series with r_t the remainder series and x_t the detrended series: $max[0, 1 - Var(r_t)/Var(x_t)]$.
34. *peak* The location of the peak (maximum value) in the seasonal component of and STL decomposition of the series.
35. *trough* The location of the trough (minimum value) in the seasonal component of and STL decomposition of the series.
36. *stability* The variance of the means based on a division of the series in non-overlapping portions. The size of the portions is the frequency of the series, or 10 is the series has frequency 1.
37. *hw_alpha* α parameter of an ets(A,A,A) model fitted on the series.
38. *hw_beta* β parameter of an ets(A,A,A) model fitted on the series.
39. *hw_gamma* γ parameter of an ets(A,A,A) model fitted on the series.
40. *unitroot_kpss* The statistic for the Kwiatkowski et al. unit root test with linear trend and lag 1.
41. *unitroot_pp* The statistic for the "Z-alpha" version of Phillips & Perron unit root test with constant trend and lag 1.
42. *series_length* The length of the series.

No exogenous features were used, even though they were available in the M4 dataset, such as which domain the series belongs to (e.g. macroeconomics, finance, tourism...).

3.3 Forecasting methods

We considered nine forecasting algorithms implemented in the `forecast` 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 is fitted to the lags. The number of lags is automatically selected (`nnetar`)
- iv) random walk with drift (`rwf` with drift=TRUE)
- v) TBATS model (`tbats`)
- vi) Theta method forecast (`thetaf`)
- vii) naive forecasts (`naive`)
- viii) STLM-AR Seasonal and Trend decomposition using Loess with AR modeling of the seasonally adjusted series(`stlm` with modelfunction `ar`)
- ix) seasonal naive forecasts (`snaive`).

It is worthy to emphasize that we used the default parameters in most methods without any hand tunning. Only in `auto.arima` we specified a more thorough search for hyperparameters than its default version. In the case of an error when fitting the series (e.g. a series is constant), the SNAIVE forecast method is used instead.

Question: i) Calculation of ets models to daily, hourly and weekly series

3.4 Metal-learning model

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

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

3.4.1 The xgboost objective function

The vanilla 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 shown in Algorithm 1 step 7, implemented as a custom objective function.

xgboost 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 xgboost to converge. This is a relative common problem and one simple fix is to use an upper bound of the hessian by clampling 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 hyperparameters such as learning rate, number of boosting steps, maximum complexity allowed for the trees or subsampling sizes. In our case we limit the hyperparameter search space based on some initial results and rules of thumb and explore it using bayesian optimization, (implemented in the `rBayesianOptimization` R package) measuring performance on a 10% holdout version of the reference set. We picked the most simple hyperparameter set from the top solutions of the exploration.

3.5 Prediction Intervals

The M4 Competition also featured a subcompetition 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 centre 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:

1. For the training period each series in the dataset:
    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 dataset:
    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.

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. *These coefficients are not restricted to be probabilities, the optimization is unrestricted.* In order to prevent overfitting for the center of the intervals, the M4 dataset was divided in two halfs, and our metalearning approach was trained in one half and applied to the other.

4 Results

We quantify the improvement in accuracy produced by our approach, compared to a classification implementation using the same underlying implementation, xgboost.

References

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.

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