Method Description

General Information

Type of Entry (*Academic, Practitioner, Researcher, Student*)	**Student**
First Name	**Pablo**
Last Name	**Montero-Manso**
Country	**Spain**
Type of Affiliation (*University, Company-Organization, Individual*)	**University**
Affiliation	**University of A Coruña**

Team Members (*if applicable*):

1st Member	
First Name	Pablo
Last Name	Montero-Manso
Country	Spain
Affiliation	University of A Coruña
2nd Member	
First Name	Thiyanga
Last Name	Talagala
Country	Sri Lanka
Affiliation	Monash University
3rd Member	
First Name	Rob
Last Name	Hyndman
Country	Australia
Affiliation	Monash University
4rd Member	
First Name	George
Last Name	Athanasopoulos
Country	Australia
Affiliation	Monash University

Information about the method utilized

Name of Method	**M4metalearning**
Type of Method (*Statistical, Machine Learning, Combination, Other*)	**Combination**
Short Description (up to 200 words)	**The forecasts of 9 standard forecasting methods are linearly combined. The weights of the combinations are calculated per series using a learning model based on gradient tree boosting. This model uses features extracted from the series as**

	input. For the intervals, one set of weights per horizon is calculated.

Extended Description:

NOTE 1: As part of the reproducibility submission, we would like to include the M4comp2018 R package, available online since mid March 2018.

NOTE 2: that the preferred way of reading the description is through a web browser in the webpage of our project

https://github.com/robjhyndman/M4metalearning

where the source code and documentation can be found.

The following is a rendering to standard document of this documentation, so it may be not as friendly to read.

Methodology: Combination of Forecast Methods by Feature-based Learning

Pablo Montero-Manso

2018-06-10

This page explains the methodoly behind the sumbission. Is a part of the reproducibility submission. The second part is the technical description and how to reproduce the results. The reproducibility part can be seen here.

Authorship

- Pablo Montero-Manso
- Thiyanga Talagala
- Rob J Hyndman
- George Athanasopoulos

Overview

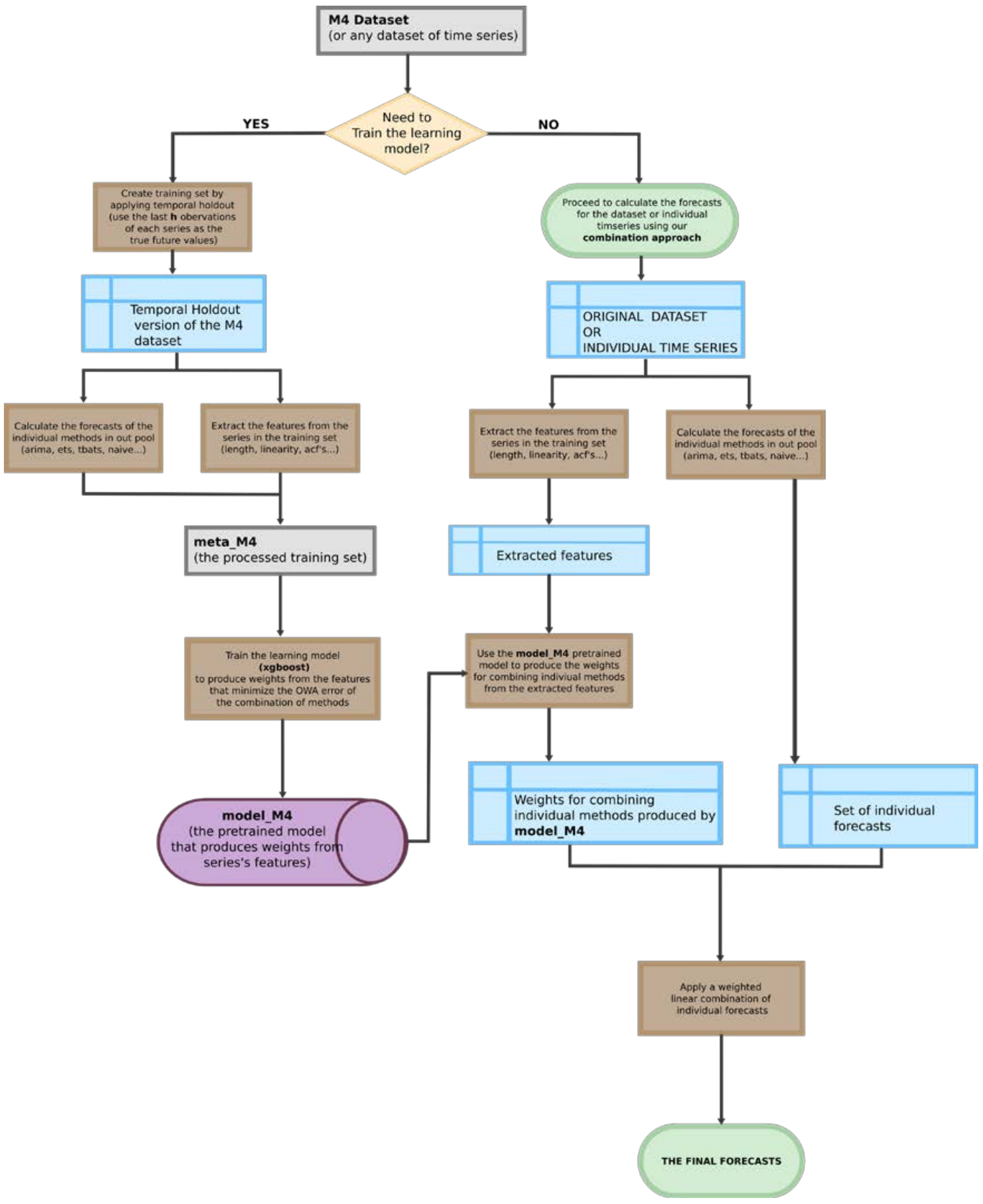
This approach can be described as a linear combination of statistical forecasting methods, where the weights of the combination are calculated by a learning model based on gradient tree boosting over a set of features extracted from the individual series.

A basic flowchart of the process is included to support the detailed description.

This methodological description page is divided in 3 sections, the Forecasting methods, where the individual forecasting methods are described, the Learning model, where the method for generating the weights is explained, for both Mean and Interval predictions, and the Features, where the features used in the learning model are described.

Table of Contents

Basic Flowchart


M4 Dataset
(or any dataset of time series)
Need to Train the learning model?
YES
NO
Create training set by applying temporal holdout (use the last h obervations of each series as the true future values)
Temporal Holdout version of the M4 dataset
Calculate the forecasts of the individual methods in out pool (arima, ets, tbats, naive...)
Extract the features from the series in the training set (length, linearity, acf's...)
meta_M4
(the processed training set)
Train the learning model (xgboost) to produce weights from the features that minimize the OWA error of the combination of methods
model_M4
(the pretrained model that produces weights from series's features)
Proceed to calculate the forecasts for the dataset or individual timseries using our combination approach
ORIGINAL DATASET OR INDIVIDUAL TIME SERIES
Extract the features from the series in the training set (length, linearity, acf's...)
Calculate the forecasts of the individual methods in out pool (arima, ets, tbats, naive...)
Extracted features
Use the model_M4 pretrained model to produce the weights for combining indivual methods from the extracted features
Weights for combining individual methods produced by model_M4
Set of individual forecasts
Apply a weighted linear combination of individual forecasts
THE FINAL FORECASTS

Forecasting methods

A set of 9 forecasting methods is used. These methods are fitted on each individual series and produce the forecasts at the asked horizon. These forecasts are linearly combined to produce the final point forecast. The rationale behind the pool of methods is to use easily available forecasting methods that can be fitted automatically. All the forecasting methods are well known forecasting methods, and are available in the `forecast` R package. For clarity, the specific R calls for fitting the models are given.

1. **ARIMA** The ARIMA model. Parameters like the order of differencing, *p*, *q* are obtained through an exhaustive search. The code for fitting the model is `forecast::auto.arima(x, stepwise=FALSE, approximation=FALSE)`
2. **ETS** Exponential smoothing state space model with parameters fitted automatically. The code for fitting the model is`forecast::ets(x)`
3. **NNETAR** A feed-forward neural network with a single hidden layer is fitted to the lags. The number of lags is automatically selected. The code for fitting the model is `forecast::nnetar(x)`
4. **TBATS** The Exponential smoothing state space Trigonometric, Box-Cox transformation, ARMA errors, Trend and Seasonal components model. Parameters like the application of a Box-Cox transformation, to include trend, etc. are automatically fitted. The code for fitting the model is `forecast::tbats(x)`
5. **STLM-AR** Seasonal and Trend decomposition using Loess with AR modeling of the seasonally adjusted series. The code for fitting the model is `forecast::stlm(x, modelfunction = stats::ar)`
6. **RW-DRIFT** Random Walk with Drift. The code for fitting the model is `forecast::rwf(x, drift=TRUE)`
7. **THETAF** The theta method of Assimakopoulos and Nikolopoulos (2000). The code for fitting the model is `forecast::thetaf(x)`
8. **NAIVE** The naive method, using the last observation of the series as the forecasts. The code for fitting the model is `forecast::naive(x)`
9. **SEASONAL NAIVE** The forecast are the last observed value of the same season. The code for fitting the model is `forecast::snaive(x)`

In the case of an error when fitting the series (e.g. a series is constant), the SNAIVE forecast method is used instead.

The Learning model

This section explains the learning model that produces the weights used to combine the individual forecast methods. We can distingush two methods:

1. **Model for producing the mean forecasts:** In essence, it is a feature-based gradient tree boosting approach where the loss or error function to minimize is tailored to the OWI error used in the M4 competition. The implementation of gradient tree boosting is `xgboost`, a tool that is computationally efficient and allows a high degree of customization. The set of features is explained in the next section.

2. **Model for producing the prediction intervals:** This method uses the mean forecasts (the result of our mean approach) as the center of the interval, and the finds a linear combination of the 95% prediction intervals of 3 forecasting methods in our original set, THETAF, NAIVE and SEASONAL NAIVE. One set of weights to combine this three intervals is produced per horizon, and is the same for all series, differing from the mean approach, where one set of weights is produced per series. The weights are calculated by minimizing the MSIS error of the linear combination of these three intervals in the training set.

Creating the training set

In order to train the learning model, a dataset where the OWI errors of the methods is required. In our case, we applied temporal holdout to the available M4 dataset to create this training set. The specific temporal holdout procedure is as follows:

- For each series in M4, we know its values, x and the required forecasting horizon, h
- We take the last h observations from x and use them as the true future values, xx of the remaining observations of x, r.
- We add r with its correspoding true future values xx to the training set.
- If r has less than 7 observations, a smaller horizon is used.

In this new training set we can apply the forecasting methods and measure their OWI errors in a scenario that will be close to the real one. We call this training set `meta_M4`

Mean forecasts

Let a be the features extraced for a time series x, f the set of M vectors with the forecasts for each method applied to x and xx the true future values of x. The gradient boosting method produces a vector of length M, real numbers, as a function of the features, $y(a)$.

We transform the output of the trees $y(a)$ from real numbers to probabilities by appling the $softmax$ transform. For a given series error function for the given features, a, the forecasts f and the true future values xx is

$$L_{OWA}(y, a, f, xx) = \sum_{i=1}^{M} \frac{e^{y(a)_i}}{\sum e^{y(a)}} OWA(f_i, xx)$$

This error function has three benefits:

1. It has interpretability as probabilities of the i-eth forecasting method being in our pool for the given features extracted from the series.
2. We are minimizing the OWI that the produced probabilities would make on average.
3. If we use the probabilities as weights in a linear combination of the forecasts, it is less prone to overfitting than a unrestricted linear combination, it can be though as a kind of regularization.

The gradient tree boosting approach implemented in `xgboost` works by additively approximating the function y so that the overall error is minimized:

$$argmin_y \sum_{j}^{N} L_{OWA}\,(y, A_j, F_j, XX_j)$$

with A_j, F_j, XX_j are the features, forecasts and future values oth the j-eth series training set. Gradient and Hessian of L_{OWA} with respect to y must be provided to `xbgoost` to proceed with the minimization.

Hyperparameter search

`xgboost` tree models have a range of hyperparameters that must be tuned to produce good results.

We have performed a search in a subset of the hyperparameter space by using Bayesian Optimization, measuring the OWA via a 10-fold crossvalidation of the training set `meta_M4`. The range of hyperparameters is as follows:

- `max_depth` The maximum depth of a tree. From 6 to 14.
- `eta` The learning rate, the scale of contribution of each tree. From 0.001 to 1.
- `subsample` The proportion of the training set used to calculate the trees each iteration. From 0.5 to 1.
- `colsample_bytree` The proportion of the features used to calculate the trees each iteration. From 0.5 to 1.
- `nrounds` The number of iterations of the algorithm. From 1 to 250.

The final set of hyperparameters was: `max_depth = 14`, `eta=0.58`, `subsample=0.92`, `colsample_bytree=0.77` and `nrounds = 94`. This set of features was not the best performing on the crossvalidation, but it produced very similar OWA error while being more regularized, less prone to overfitting.

Producing the final combination forecast

The learning model, (that is the function y), is trained in `meta_M4`.

For a series x in the original M4 dataset, the vector of features, a is extracted from it and the individual methods' forecasts f are calculated. The model the produces its output $y(a)$ from and is transformed to probabilties, then the individual method's forecasts are combined by linear combination using these probabilities to produce the **final submitted forecast**.

$$combi\_forecast(x) = \sum_{i}^{M} \frac{e^{y(a)_i}}{e^{y(a)}} f_i$$

Note that we omit the forecasting horizon h for simplicity. It would be an extra input parameter.

Prediction Intervals

The prediction intervals are also a linear combination of intervals of single methods. The weights of the combination is calculated the following way.

In the training set we calculate the 95% prediction intervals of the methods THETA, NAIVE and SEASONAL NAIVE, produced by their implementation in the `forecast` R package. For each of these methods, we get 'radius' of the interval, the difference from the upper bound to the mean forecast (we consider symmetric intervals). These radius are also vectors of length h, the forecasting horizon. Using the mean forecast produced by our approach described in the previous section, the MSIS is minimized.

A set of weights w will be calculated per forecasting horizon. For a given series x we produce the mean combination forecast c, and the radius of the intervals of the $M = 3$ methods, r. *NOTE: This combination forecast c used for training the interval is based on a model trained on a disjoint subset of the training set, to avoid overfitting, the final combination c is based on all the training set.* We start by calculaing the denomitator of the MSIS error formula, common to al horizons:

$$D = \sum_{t=1+m}^{n} x_t - x_{t-m}$$

Since we will calculate one set of weights per horizon, we will simplify the MSIS formula and show the minimization of one horizon. This means that if we w from now are the forecast for one specific horizon, eg. 13, xx would therefore 13th true future value of x in the training set. The weights are calculated by minimizing the following formula:

$$L(w, x) = \sum_{i=1}^{M} c - w_i r_i$$

$$U(w, x) = \sum_{i=1}^{M} c + w_i r_i$$

$$MSIS(w, x, xx) = \frac{2\sum_1^M r_i\, w_i + \frac{2}{\alpha}(L - xx)I(L < xx) + \frac{2}{\alpha}(xx - U)I(xx > U)}{D}$$

$$argmin_w \sum_{j=1}^{N} M\,SIS(w, X_j, XX_j)$$

with X and XX the set of series and true future values in the training set `meta_M4` respectively.

This minimization is repeated for all forecasting horizon, 48 in the M4 competitions, to produce 48 sets of 4 weights. The optimization algorithm used is the standard Conjugate Gradient implemented in the `optim` R function.

Computing the prediction intervals

For a given series x we calculate the radius r based on the three methods and the mean forecast c. The precalculated set of weights W is a $3 \times h$ matrix, just like r.

The prediction intervals for x are:

$$U = c + diag(W'r)$$

and

$$L = c - diag(W'r)$$

With W' denoting the transpose of W and diag the diagonal elements of the matrix.

Final postprocessing

A final step, common to both mean and prediction interval, is setting negative values to 0. This is due to no obervations in the dataset being less than 0.

The Features

A set of 42 features is used in the metalearning model. For each series the following 42 features are calculated.

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
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-alphaâ€ � version of Ph
test with constant trend and lag 1.
42. *series_length* The length of the series.

Reproducibility: Combination of Forecast Methods by Feature-based Learning

Pablo Montero-Manso 2018-06-10

This page explains how to reproduce the results for our submission. The first part is the methodological description of our methods. The methodology can be seen here

Authorship

- Pablo Montero-Manso
- Thiyanga Talagala
- Rob J Hyndman
- George Athanasopoulos

Overview

The technical part of the reproducibility is divided in two sections. First, the tools for producing the forecast of our method and second, a way of replicating the training of our model. It is divided in two parts because replicating the training is a very computationaly expensive process, so a pretrained model is provided. Also for very long time series, the actual calculation of there forecast can also computationally expensive, because 9 different forecasting methods are calculated independently, and some of these methods do exhaustive search on its parameters, which is time consuming.

The model

Two R packages are provided as part of our submission, the `M4metalearning` package contains the functions for training new models and making the predictions.

The `M4metaresults` package is just a 'data' package containing the trained model used in ou submission, as well as the training set, explained in our methodology page as well as the 'test' or 'final' dataset, which is the original M4 dataset but with the forecasts of the individual methods and the features of the series already extracted.

We first show how to reproduce the exact results of our submission and also how to easily use our trained model and provided tools to produce new forecast for any time series.

Simple forecasting

The function `forecas_M4` is included in the M4metalearning package to simplify forecasting. `forecast_M4` uses the pretrained model of the`M4metaresults` package. This takes as input a time series and its required forecasting horizon and outputs the mean and interval forecasts. It may be computationally expensive, especially for long time series (length>1000).

**NOTE: The results may slightly vary since one of the individual forecasting methods, `nnetar` uses random initialization. Nevertheless, the results should vary less than 0.1%.*

The following example shows how to produce forecasts for a example ‘new’ time series and serveral of the ones in the M4 dataset.

```r
#install the package if not already installed
devtools::install_github("robjhyndman/M4metalearning")


devtools::install_github("carlanetto/M4comp2018")


#the M4metaresults package must be installed from sources due to its large size
#note that this package is about 1GB in size
install.packages("https://github.com/pmontman/M4metaresults/releases/download/v0.0.0.9000/M4metaresults_0.0.0.9000.tar.gz",
                 repos = NULL, type="source")
library(M4metalearning)
library(M4metaresults)
#we will showcase how to forecast a syntethic time series
#first, we generate the full series
set.seed(10-06-2018)
truex = (rnorm(60)) + seq(60)/10

#we subtract the last 10 observations to use it as 'true future' values
#and keep the rest as the input series in our method
h = 10
x <- head(truex, -h)
x <- ts(x, frequency = 1)

#forecasting with our method using our pretrained model in one line of code
#just the input series and the desired forecasting horizon
forec_result <- forecast_meta_M4(model_M4, x, h=h)
```

```
#> Loading required package: tsfeatures
#show the output, the mean, upper and lower interval forecasts
print(forec_result, digits=2)
#> $mean
#>  [1] 5.3 5.3 5.4 5.5 5.5 5.6 5.6 5.7 5.8 5.8
#>
#> $upper
#>  [1]  8.1  8.6  8.9  9.5  9.7 10.7  9.8  9.9 10.0 10.2
#>
#> $lower
#>  [1] 2.37 2.01 1.84 1.39 1.37 0.48 1.51 1.53 1.52 1.47
plot(truex, ylim=range(c(truex,unlist(forec_result))), type="l")
lines(c(x,forec_result$mean), col="blue")
lines(c(x,forec_result$upper), col="blue")
lines(c(x,forec_result$lower), col="blue")
lines(truex)
```

example forecast

Any time series of the M4 dataset may be easily forecasted using our `forecast_meta_M4` and the `M4comp2018` package containing the original M4 series.

```
forec_M4_2018 <- forecast_meta_M4(model_M4,
                                  M4comp2018::M4[[2018]]$x,
                                  M4comp2018::M4[[2018]]$h)
print(forec_M4_2018, digits=2)
#> $mean
#>  [1] 12287 12279 12273 12267 12261 12255 12249 12244 12239 12234 12229
#> [12] 12225 12221 12217
#>
#> $upper
#>  [1] 12623 12838 13033 13216 13403 13517 13312 13429 13450 13513 13541
#> [12] 13612 13662 13743
#>
#> $lower
#>  [1] 11950 11721 11513 11317 11118 10993 11187 11059 11028 10955 10918
#> [12] 10837 10779 10690
```

Exact reproduction of the submitted forecasts

Calculating the individual forecast methods for the 100000 time series in the M4 dataset is a very time consuming process (hours). One of the individual methods `nnetar` in the `forecast` R package produces random results which difficult reproducibiliy, for instance when running in a cluster. Training the learning model is also time consuming.

In order to facilitate the exact reproduction of the process, the individual forecast are already calculated in the (`M4metaresults`package)

<https://github.com/pmontman/M4metaresults><https://github.com/pmontman/M4metaresults>
and we show here how to apply our approach to the dataset of precalculated individual forecasts.

```
library(M4metalearning)
library(M4metaresults)
data <- create_feat_classif_problem(submission_M4)

preds <- predict_selection_ensemble(model_M4, data$data))

replication_M4 <- ensemble_forecast(preds, submission_M4)


interval_M4 <- predict_interval(submission_M4,
                                get_M4_interval_weights())
```

By examining the `y_hat` elements of the `replication_M4` list we can check the actual mean forecasts. By examining the `upper` and `lower`elements of the interval_M4 list we can check the upper and lower prediction interval bounds. **NOTE that these are the exact results submitted to the M4 competition, in the same series if forecats using the `forecats_meta_M4` approach of the previous subsection, the resulst will be slightly different due to the randomness of the `nnetar` method.

Reproducing the Training of the Model

This section explains how to arrive to the trained metalearning model, for producing the mean forecast and the prediction intervals. Note that this is a very time consuming process, the pretrained models have been provided in the previous section.

```

#training
library(M4metalearning)

```

```r
#create the training set using temporal holdout
set.seed(10-06-2018)
meta_M4 <- temp_holdout(M4comp2018::M4)



#calculate the forecasts of each method in the pool
#THIS WILL TAKE A LOT OF TIME (hours...)
meta_M4 <- calc_forecasts(meta_M4, forec_methods(), n.cores=3)
#calculate the OWA errors
meta_M4 <- calc_errors(meta_M4)
#extract the features
meta_M4 <- THA_features(meta_M4, n.cores=3)

#search for hyperparameters
hyperparameter_search(meta_M4, filename = "M4_hyper.RData", n_iter=150)

#get the best hyperparameter found
load("M4_hyper.RData")
best_hyper <- bay_results[ which.min(bay_results$combi_OWA), ]

#Train the metalearning model with the best hyperparameters found

train_data <- create_feat_classif_problem(meta_M4)
param <- list(max_depth=best_hyper$max_depth,
              eta=best_hyper$eta,
              nthread = 3,
              silent=1,
              objective=error_softmax_obj,
              num_class=ncol(train_data$errors), #the number of forecast me
thods used
              subsample=bay_results$subsample,
              colsample_bytree=bay_results$colsample_bytree)

meta_model <- train_selection_ensemble(train_data$data,
                                       train_data$errors,
                                       param=param)
```

```r
## Now the model is trained, lest produce the predictions

final_M4 <- M4comp2018::M4

#just calculate the forecast and features
final_M4 <- calc_forecasts(final_M4, forec_methods())
final_M4 <- THA_features(final_M4)

#get the feature matrix
final_data <- create_feat_classif_problem(final_M4)
#calculate the predictions using our model
preds <- predict_selection_ensemble(meta_model, final_data$data)
#calculate the final mean forecasts
final_M4 <- ensemble_forecast(preds, final_M4)
#the combination predictions are in the field y_hat of each element in the
list
#lets check one
final_M4[[1]]$y_hat
}
```

For the interval, a similar procedure is required.

```r

#we need the mean predictions on the training as the centers of the interva
ls
#we will calculate them
preds <- predict_selection_ensemble(meta_model, train_data$data)
#this will add the mean forecasts to the dataset, not called interval_M4
interval_M4 <- ensemble_forecast(preds, meta_M4)
#we calculate the radius of the individual forecasts methods
interval_M4 <- calc_radius_interval(interval_M4)
#prepare the data to pose it as a minimization problem
info <- prepare_radius_info(interval_M4)

set.seed("10-06-2018")
#calculate the weights for the combinations of individual methods radius
res_opt <- train_interval_weights(interval_M4, 48)
```

```r
#transform the results of the optimization process to the format used by pr
edict interval

weights <- lapply(res_opt, function (lentry) lentry$opt$par)


#in interval_M4 we have the  the prediction intervals

Interval_M4 <- predict_interval(interval_M4, weights, TRUE)
```