MAS Thesis Draft
Chris Meade

Ensemble Methods to Improve Automated Time Series Forecasting

Abstract

There currently exist several “black box” software libraries for the automatic forecasting of time series. Popular among these are the ‘forecast’ and ‘bsts’ packages for R, which have functions to automatically fit several common classes of time series models, such as the autoregressive integrated moving average (ARIMA) and the family of exponential smoothing models, among others. It is often the case that what one gains from the ease in fitting these automatic methods comes at the cost of predictive performance. In this paper, we propose several methods to improve the prediction accuracy of automatic time series forecasting, all of which relate to creating ensembles of models automatically fit from these packages. We explore different ways that one can construct these ensembles and evaluate each on a benchmark time series dataset. In addition, we release an R software library that implements the methods discussed within this paper.

Introduction

Several scenarios exist that necessitate automatic time series forecasting. A manufacturing firm may need to generate monthly production forecasts for each of its products, of which there could be thousands. A hedge fund manager may need to forecast the price of a security every 30 seconds using the latest available stock data. In either case, the high volume of time series datasets or the frequency at which forecasts must be made represent too high a cost for a statistician to manually apply the classical supervised methodology to fit an appropriate, causal, and invertible ARIMA model to each time series. In the first scenario, using such a methodology would consume too many man-hours of work, while in the second, a human could simply not keep up with the data.

In such situations, automatic ‘black box’ time series forecasting methods, like those implemented in the popular “forecast” R software library by Rob J. Hyndman and the ‘bsts’ package by Steven L. Scott, deliver a compelling value proposition — adequate forecasts can be made almost instantly by anyone. Even if these automatic methods do not perform as well as the ideal, manually fitted ARIMA model, the value they create by reducing forecast costs (in hours of work or in time to model) can easily surpass the cost associated with implementing a less-than-ideal forecast. In the first scenario given above, 1000 adequate production-ready forecasts are better than 100 excellent forecasts and 900 missing values because the statistician ran out of time.

The work presented in this paper deals with how these ‘black box’ automatic methods can improved, so that a forecaster need not settle with adequate performance while still enjoying the benefits that black box methods provide. To that end, we explore several ways to create ensembles of four of the popular black box models included in the ‘forecast’ and ‘bsts’ packages to increase prediction accuracy over any single automatically fit model alone. To evaluate the performance of the ensembles, they will be tested with the M3-Competition dataset, which contains 3,003 univariate time series with mostly monthly, quarterly, and yearly periods. Each series in the dataset is split into a training and test set, the latter of which will be used to evaluate loss using normalized root mean squared error (nRMSE), mean absolute percent error (MAPE), and normalized mean absolute error (nMAE).

Members of the Ensemble

The time series ensembles proposed in this paper will be composed of combinations of four classes of models that can be fit using the `forecast` and `bsts` packages in R. These four methods are chosen because they are prominently featured in time series literature, are commonly used by practitioners, and because each can be fit automatically. They are as follows:

Auto.Arima

Proposed by Hyndman and Khandakar (2007), Auto.Arima was designed to automatically select the best Autoregressive Integrated Moving Average (ARIMA) model for forecasting. Given a nonseasonal ARIMA(p,d,q) process $(1-B^d)y_t=c+\phi(B)y_t+\theta(B)\epsilon_t or seasonal ARIMA(p,d,q,P,D,Q)_m process $(1-B^m)(1-B)^dy_t=c+\PHI(B^m)\phi(B)y_t+ \THETA(B^m)\theta(B)\epsilon_t$, ϕ and θ are polynomials with orders $p $ and q respectively, $\PHI$ and $\THETA$ are polynomials with orders P and Q respectively, d and D are the respective numbers of nonseasonal and seasonal differences, B is the backshift operator, ϵ is gaussian white noise with mean 0 and variance σ^2, and c is a constant.

The goal of ARIMA modeling is to choose the parameters p, d, q, P, D, and Q which best fit the data, which can be evaluated by minimizing an information criterion, such as AIC or BIC.

Auto.Arima provides a framework to automate the ARIMA parameter optimization process. If the data are nonseasonal, the algorithm first chooses d on the basis of KPSS unit-root tests, by which the data are checked for a unit root (Kwiatkowski et al., 1992). If a root is present, it is differenced and checked again. The process repeats until no unit root is detected.

For seasonal data, first D is chosen to equal 0 or 1 on the basis of the Canova-Hansen test (Canova and Hansen, 1995). If $D=1$, seasonal differencing is applied. Then the KPSS unit-root test process described previously is applied to select d.

Auto.Arima then considers ARIMA models where p and q can take values ranging from 0 to 3 and, if applicable, P and Q can be set equal to 0 or 1. If $d+D\leq 1$, the constant c is fit. Otherwise it is set equal to 0. For the sake of efficiency, the model space is explored using a novel step-wise algorithms. Finally, the model which minimizes AIC is returned at the “best” model.

ETS

Short for both “Error, Trend, Seasonality” and “ExponenTial Smoothing”, the ets function from R’s forecast packages fits 30 different classes of exponential smoothing models to a given time series and chooses best on basis of which minimizes AIC.

As the function’s acronym suggests, an exponential smoothing model has three components: error, trend, and seasonality. Each of these three components can be further classified. Hyndman and Khandakar (2007) explain that error can be either “additive” or “multiplicative”. Trend can be classified as “none”, “additive”, “additive damped”, “multiplicative”, or “multiplicative damped”. Finally, seasonality can be classified as “none”, “additive”, or “multiplicative”.

These 30 different combinations or error, trend, and seasonality comprise the 30 exponential smoothing models fit by the ets function. Hyndman and Khandakar (2007) provide a taxonomy of each of these 30 methods, including formulas for parameters optimization and point forecasts.

Theta:

Proposed by Assimakopoulos and Nikolopoulos (2000), the Theta method applies a coefficient, θ, to a twice differenced time series in order to change its local curvature. For example, setting $\theta=0$ reduces the time series to a simple linear regression. At the other extreme, setting $\theta = 2$, Assimakopoulos and Nikolopoulos explain, doubles the the local curvature, thereby magnifying the series' short term behavior.

The first Theta-line, given by $\theta=0$, is extrapolated via its simple linear trend. The second line, when $\theta=2$, is extrapolated by simple exponential smoothing. Forecasts are made by combining these two extrapolations.

Assimakopoulos and Nikolopoulos have developed a six-step algorithm to automate forecasting using their Theta method. First, a time series is tested for seasonality by examining the autocorrelation function at the lag equal to the series' periodicity. For example, for monthly data one would check the autocorrelation at lag 12 or lag 4 for quarterly data. Next, the data is deseasonalised if the seasonality was determined to be significant. Then two Theta-lines are generated, corresponding to $\theta=0$ and $\theta=2$. Next, these two lines are extrapolated via linear trend and simple exponential smoothing, respectively. Then these two lines are averaged with equal weights at each point in the forecast horizon. Finally, seasonality is reintroduced to the series.

This algorithm is provided in R's forecast package as the function named 'thetaf' (Hyndman, 2019).

BSTS:

The BSTS framework, short for Bayesian Structural Time Series, was developed by Scott and Varian (2013) at Google to improve automated time series forecasting. According to Scott and

Varian, his approach to time series forecasting combines three methods from Bayesian statistics — Kalman filtering, spike-and-slab regression, and model averaging.

The Case for Ensembles

The use of ensembles to improve prediction accuracy is nothing new in the realm of statistical modeling. In 1996, Leo Breiman introduced the concept of bootstrap aggregation, shortened to 'bagging' (Breiman, 1996). Given a training set L, Bagging uses the bootstrap to create B\in\mathbb{N} new training sets L_i, i = 1,…, B. A given machine learning model model is then fit to each of the new training sets. The final decision is made by taking the average of the B models in the case of regression, or by majority vote for classification. The method, one of the first implementations of ensemble learning, was successful, with Breiman concluding "What one loses […] is a simple and interpretable structure. What one gains is increased accuracy."

In the case of bagging, it was found that an ensemble of models performed better than any single learner. Dietterich (2002) gives three possible explanations of the improved predictive performance of ensemble learning. For example, in the event of insufficient training data to establish a "best fit" model, it may be the case that several different learners provide an equally accurate but vastly different fit to the training data. Making a future prediction with only one of these models can be risky, due to the high variance of the predictions. However, an ensemble can reduce this prediction variance and thus reduce risk with a simple majority vote or average over all predictions.

It can also be the case that finding the best model to fit to a training data set can be too computationally expensive. This is especially true with gradient based methods, as a learner may become trapped at a local minimum and fail to reach the global minimum. In such a scenario, an ensemble reduces the risk of choosing the wrong minimum.

Finally, it is possible that no single learner can accurately model a given data. Creating an ensemble allows one to explore more functional relationships between between data and model, which comprise what Dietterich calls the representation space. One of these previously untested members of the representation space may perform better than any of its component learners.

In each of the three explanations provided by Dietterich, the advantages attributed to ensemble learning directly relate to the diversity of its member learners (Oliveira 2014). That is, the predictions made by the member learners are relatively uncorrelated. In bagging, uncorrelated learners are created with bootstrap sampling of the dataset. The random forest, in addition to bootstrap sampling, uses a random subset of predictor variables to construct uncorrelated classification and regression trees (Breiman 2001). The result is an even more diverse set a learners and often an even greater prediction accuracy.

Application to Time Series

How, then, can an ensemble of uncorrelated time series models be constructed? The predictions from two reasonable ARIMA models fit to the same data will likely have a correlation coefficient close to one. An ensemble created from these two models will likely look very similar to both of the original forecasts and would therefore likely not reap the benefits of ensembling.

In time series forecasting, a natural way to create diverse forecasts is to combine the forecasts of different classes of time series models, such as the ARIMA, Theta, Exponential Smoothing, and Bayesian Structural Time Series, all of which were discussed in the previous section. Because forecasts created by these different classes of models will naturally vary, an ensemble of these learners should enjoy the same benefits as those ensemble methods mentioned above, namely a increase in prediction accuracy from a reduction in bias and variance and a robustness to training noise and outliers.

Creating the Ensembles

We propose the following ensemble methods:

Naive Ensemble

Perhaps the simplest way to create an ensemble of multiple predictions is to take the average of all the predictors for each time point on the prediction horizon. Specifically, let X be a h by p matrix, where h is the prediction horizon (the number of time points into the future to be forecasted) and p is the number of models from which predictions were made. In this design, each column of the forecast matrix X corresponds to a prediction of horizon h made by one of the p models. Let W be a 1 by h weight matrix with every element given by 1/p. Then a final forecast is given by XW.

We are specifically interested in Naive Ensembles made with the four classes of automatic time series models we previously discussed. We will refer to this model as the **BEAT** (**B**sts, **E**ts, **A**uto Arima, **T**heta) Ensemble. Also of interest are the four Naive Ensembles made using three of the four component models, called **BEA**, **EAT**, **BAT**, and **BET**, following the same naming convention.

Median Ensembles

The Naive Ensembles proposed in the previous section simply take the average of component forecasts at each time point in the forecast horizon. If one of these component forecasts predicts extreme or unreasonable values, it can have a huge impact on the accuracy of the Naive Ensemble on the whole.

One solution to combat the influence of an extreme forecast is to take the median, rather than the mean, of the p predictors at each time point in the forecast horizon. If X is the same h by p forecast matrix, let M = [m_1, … , m_h] be the final forecast, where each m_i is the median i'th row of X, i = 1,…,h. We will refer to this method as the Median BEAT Ensemble.

Bagged Ensemble

Just how Breiman introduced bootstrap aggregation (bagging) to improve the accuracy of classification and regression tree ensembles, so too can bagging be used in time series analysis to improve forecasts. Because of the order-dependent nature of time series, the standard methodology of resampling data points with replacement is ill-suited to create bootstrapped series. Bergmeir et. al. (2016) proposes an adaptation specifically suited for time series analysis in such a way that the trend and seasonal structure of the time series is preserved.

The Box-Cox transformation, shown below, is applied to the series to ensure that its trend and seasonality components are additive (Petropoulosa et. al., 2018). The Box-Cox lambda parameter is chosen on the basis of maximum likelihood estimation (Box & Cox, 1964). The Box-Cox transformed series is then decomposed into seasonal, trend, and error components using LOESS (STL) (Cleveland et al., 1990). Bootstrapped resampling is then applied to the error components of the series. Bergmeir et. al. utilize moving block bootstrap (Künsch 1989), whereby n data points are assigned to n-b+a overlapping blocks, each with length b. Then n/b blocks are drawn with replacement and assigned in the order that they were drawn, creating a bootstrapped set of errors. This method is utilized in the event that there is any remaining autocorrelation in the LOESS residuals after the STL decomposition.

A new bootstrapped time series is then constructed by performing the inverse STL decomposition, whereby the trend, seasonality, and bootstrapped error terms are added together. The inverse Box-Cox transformation is then applied to return the time series to its original scale. This process then repeated a given number of times to create a set of bootstrapped series.

If a time series is not periodic, or has fewer than two periods, the Box-Cox transformation is applied, after which the series is decomposed into trend and error components using LOESS. Seasonality is not calculated. Then the same procedure as above is followed to create a set of bootstrapped time series.

Bergmeir et. al. then fit an ETS model to each of the bootstrapped time series, make a forecast, then take the median of the component forecasts for each point on the forecast horizon to make a final prediction. This method was found to perform better than the ETS model alone the M3 Competition Dataset.

We propose the following adaptation to this method: given that we expect the BEAT ensemble to perform better than ETS alone under our ensemble hypothesis, we fit a BEAT ensemble to each bootstrapped time series instead of an ETS model. Final forecasts will then be made by taking an equal-weight average of the BEAT point forecasts for each point on the forecast horizon. We also evaluate the median of the BEAT models, as done in the original paper. These methods are referred to as Mean Bagged BEAT and Median Bagged BEAT, respectively. In addition, early results using the EAT (Ets, Auto Arima, Theta) ensemble proved promising, so we evaluate Mean Bagged EAT and Median Bagged EAT as well.

Random Error Resampling Ensemble

We propose the Random Error Resampling ensemble model for time series forecasting. This model is functionally very similar to the bagged model proposed in the previous section. Bagging utilizes bootstrap resampling of residuals after performing STL decomposition of a time series. However, it is common in time series literature to assume that these errors are distributed as gaussian white noise with mean 0 and a finite variance (Schumway 2011).

Therefore, instead of applying bootstrap resampling to STL residuals to create a new set of bootstrapped time series, we create a new set of residuals by sampling from a gaussian distribution with mean 0 and variance equal to the sample variance of the original STL residuals. We repeat this resampling a number of times to create a bootstrapped set of time series.

Except for this step, the process is exactly the same as the bagged BEAT model. We evaluate the performance of this ensemble using both the mean and median of the component BEAT forecasts. They are referred to as Mean Perturbed BEAT and Median Perturbed BEAT models, respectively.

Methodology

Each ensemble method, in addition to the automatic component learners (auto.arima, thetaf, bsts, and ets), will be evaluated on the M3 Competition Dataset (Makridakis and Hibon, 2000). This dataset contains 3003 time series, each of which is divided into a training and a test set. Of the 3,003 series, 645 are yearly data, 756 are quarterly, 1428 are monthly, and 174 have frequencies that are not yearly, quarterly, or monthly. To ensure that enough data is available to make a reasonable forecast, each yearly series has at least 14 observations, quarterly series have at least 16 observations, monthly data have at least 48 observations, and series with frequencies that are not yearly, quarterly, or monthly have at least 60 observations.

Each method will be fit to the training set of each series, and will create a forecast on the same time interval as the test set. In this way, forecasts can be evaluated against the ground truth for each of the 3,003 series.

The performance of the methods will be evaluated on the basis of normalized RMSE, normalized MAE, and MAPE. Standard RMSE and MAE are insufficient for this problem, as they are scale dependent, meaning that it does not make sense to compare the RMSE and MAE of the same model on multiple time series. We solve this problem by normalization. We first calculate the mean of training set, then divide RMSE and MAE of the model by this mean to get normalized RMSE (nRMSE) and normalized MAE (nMAE). This normalization process allows us to compare the performance of the models on the 3,003 series in the M3 Competition Dataset. MAPE, mean absolute percent error, is already scale independent.

- nRMSE: sqrt(\sum_{i=1}^T(\hat{y}_t-y_t)^2/T}/\bar{Y}
- nMAE: \sum_{i=1}^T|\hat{y}_t-y_t)|/T}/\bar{Y}
- MAPE: 1/T \sum_{i=1}^T |\hat{y}_t-y_t)|/y_t

A smaller number represents a more accurate forecast for each of the three metrics. In evaluating the nRMSE, nMAE, and MAPE for each method on each of the time series in the competition dataset, we may compare models and draw conclusions about “best” methods by determining which best minimizes these loss functions by better predicting the test set.

Results

The summary of results are given below. The average performance of each method is given for each period of data.

Results Table

Period	Method	Average nRMSE	Average nMAE	Average MAPE
MONTHLY	Auto.Arima	0.179352654057389	0.149074129210316	21.913327284757
MONTHLY	BAT	0.173754239771607	0.144262404511173	20.9273127499491
MONTHLY	BEA	0.174824482368254	0.145248341153883	21.2830583761505
MONTHLY	BEAT	0.171246443701531	0.142007003230992	20.5855742142992
MONTHLY	BET	0.173599536447893	0.144097928523442	20.7709683332572
MONTHLY	BSTS	0.201816945097609	0.169145551562309	25.3932296148741
MONTHLY	EAT	0.167983290120919	0.138782340771624	19.9440327197852
MONTHLY	ETS	0.173704014058845	0.143896812375323	20.6976267143733
MONTHLY	meanBaggedBEAT	0.170395611086606	0.140969606480069	19.4073997545448
MONTHLY	meanBaggedEAT	0.168525386334717	0.138816923588856	18.8409958241792

MONTHLY	meanPertBEAT	0.170928278327231	0.141447433682479	19.8903420970665
MONTHLY	medianBaggedBEAT	0.169487709467157	0.139951764027036	19.1012965578787
MONTHLY	medianBEAT	0.170257210458723	0.141017793703365	20.2113993719478
MONTHLY	medianPertBEAT	0.169168857737462	0.139850847641178	18.893716751178
MONTHLY	THETA	0.171910989531606	0.141943551112754	19.5617701017724
OTHER	Auto.Arima	0.0427273855154894	0.0371179293029453	4.82126267690975
OTHER	BAT	0.0415771746030272	0.0365882650958403	4.67567092912108
OTHER	BEA	0.0419777480582881	0.0367313926212137	4.72970610123745
OTHER	BEAT	0.0414265165036683	0.036404064318952	4.66382155782714
OTHER	BET	0.0421216143019473	0.0370812502377806	4.72400430714189
OTHER	BSTS	0.0461259348786055	0.0402465022304847	5.12701141841804
OTHER	EAT	0.0417826191108427	0.0366916728895651	4.72988151824085
OTHER	ETS	0.0423216146477729	0.0370050471669946	4.79658939132952
OTHER	meanBaggedBEAT	0.0445926407781915	0.0397118071984218	5.10205453144379
OTHER	meanBaggedEAT	0.0432712591543964	0.038247252286897	4.90947187067011
OTHER	meanPertBEAT	0.0438723885427401	0.0388365300862863	5.00787572586024
OTHER	medianBaggedBEAT	0.0446831840210988	0.0397346616295515	5.10639920925383
OTHER	medianBEAT	0.0420405039513031	0.0368350510833602	4.74825362191859
OTHER	medianPertBEAT	0.0435997637356648	0.0384830387652763	4.96367760895141
OTHER	THETA	0.0476432907346682	0.0423545784051314	5.46549073768767
QUARTERLY	Auto.Arima	0.147365470281364	0.126864203765783	13.2291804445789
QUARTERLY	BAT	0.132876657272763	0.113465726039971	11.8921031360842
QUARTERLY	BEA	0.139615554537844	0.119509479261084	12.2900783620751
QUARTERLY	BEAT	0.132344994946118	0.113050153208255	11.8006302948928
QUARTERLY	BET	0.13132946578083	0.111838977796	11.610214884854
QUARTERLY	BSTS	0.155959862148043	0.132776536899446	13.6615727133982
QUARTERLY	EAT	0.131018936321763	0.111977910498826	11.7843579472968
QUARTERLY	ETS	0.139367221227563	0.119415424269755	12.1527000783308
QUARTERLY	meanBaggedBEAT	0.134913568379698	0.115587623437695	12.1555417049004
QUARTERLY	meanBaggedEAT	0.132529342289139	0.113614516489097	11.8963504131255
QUARTERLY	meanPertBEAT	0.134263443172932	0.114697638423861	12.0184780946994
QUARTERLY	medianBaggedBEAT	0.135303584504397	0.116092768502346	12.1720134265916

QUARTERLY	medianBEAT	0.134005548176193	0.114511653163891	11.9114081070998
QUARTERLY	medianPertBEAT	0.13545931305408	0.11570969748332	12.1345472700232
QUARTERLY	THETA	0.134151039587879	0.115137412995583	11.8459117910564
YEARLY	Auto.Arima	0.390432436369911	0.338083897813619	22.0709152591235
YEARLY	BAT	0.351253847058328	0.302848300750172	21.142129906372
YEARLY	BEA	0.366393831695221	0.316011743779804	21.4578126010235
YEARLY	BEAT	0.346991452149822	0.299065576759458	20.808885813614
YEARLY	BET	0.339661684728264	0.29273491896186	20.9104559418198
YEARLY	BSTS	0.399953628936052	0.345245083876453	24.6491353084064
YEARLY	EAT	0.341249588084592	0.29380190477184	20.4067967676234
YEARLY	ETS	0.354166800475176	0.305252479503413	21.0164078095437
YEARLY	meanBaggedBEAT	0.350781762911277	0.302597567368534	21.427229653823
YEARLY	meanBaggedEAT	0.347743340561574	0.300143391119722	20.9026367170306
YEARLY	meanPertBEAT	0.36113547960866	0.312432902825334	21.8182960438853
YEARLY	medianBaggedBEAT	0.350382341800014	0.302471955658046	21.467529633319
YEARLY	medianBEAT	0.354047284270817	0.304779132399154	20.9935730403271
YEARLY	medianPertBEAT	0.360042828359911	0.311655881562283	21.6820141029765
YEARLY	THETA	0.33139553368441	0.285401829648767	20.9113611247437

Also provided are the average results of each method on the dataset as a whole.

Average Results on M3 Data

Method	Average nRMSE	Average nMAE	MAPE
Auto.Arima	0.208720403618032	0.177592350432154	18.7706066688817
BAT	0.193929059948589	0.164332439107129	17.7572005025255
BEA	0.199409547005673	0.169158386461268	18.0974907066757
BEAT	0.191678467630145	0.16233217470551	17.4994048557447
BET	0.191007712859197	0.161701055416925	17.5649304244946
BSTS	0.223808277294684	0.190344615389736	21.1057186553767
EAT	0.188580291106152	0.15941493926727	17.1077045059277
ETS	0.196207958234855	0.166196948487064	17.6935870125176

meanBaggedBEAT	0.192918064235732	0.163427947644911	17.1867056255683
meanBaggedEAT	0.190699330073808	0.16129558791901	16.7282822967806
meanPertBEAT	0.195189785029652	0.165492883284839	17.4603893702329
medianBaggedBEAT	0.192503977184517	0.163045452727726	17.0542003699817
medianBEAT	0.193177168497656	0.163481870957797	17.3939239338441
medianPertBEAT	0.194403713852856	0.164801076950324	16.9838575615042
THETA	0.189459584013935	0.160237479846694	17.0924210233723

The full results will be provided in the appendix.

Conclusion

Experimental results indicate that time series ensemble have the ability to increase forecast accuracy over their component models.