Time Series Forecasts with Random Forest

The questions:

* How to use random forest with time series?
  + Is it possible to overfit?
* How to tune the parameters?
  + Feature fraction
  + Minimum size vs. other criteria (for leafs)
* Applications
  + Significant variables (and how these change with horizon)

For tuning:

* Pick a few hyperparameters
* Select values for those hyperparameters
* Use cross-validation to choose best values

1. **Introduction**

Predicting the future is hard. It makes sense, then, to explore all of the resources at our disposal when we set out to forecast. We should not limit ourselves to a certain class of models, as if only these could reasonably be expected to forecast well. Indeed, some classical models do a fair job, but no model is perfect, and many models even fare worse than a naïve forecast.

Given the rise of machine learning methods and their success in modeling cross-sectional data, and given the difficulty of forecasting time series data even with sophisticated classical models, it makes sense to at least consider applying some machine learning techniques to time series forecasting. Classical models such as the ARIMA and the VAR are valuable, and they often perform significantly better than a naïve forecast. But machine learning is a valuable and underexplored resource when it comes to time series data.

With this in mind, this paper offers a brief exploration of one particular machine learning technique, the random forest, and its success in forecasting a particular set of time series data, the US seasonally-adjusted monthly inflation rate.

1. **The Random Forest**

The random forest was first developed by Tin Kam Ho in 1995. It refers to a collection of regression trees, which are each trained on a randomly selected subsample of features. This method allows the “forest” (the collection of trees) to grow increasingly accurate while also remaining resistant to overfitting as it grows in size. Ho’s work represented a major development in statistical modelling: “the accuracy increases with the addition of new trees” and yet “an increase in classifier complexity” did not lead to “overtraining” (Ho 1995). Thus, this method has the demonstrated potential to improve accuracy without risking overfitting—an ability which classical time series models lack.

The random forest we will use is more simplistic than Ho’s. For each regression tree, the splitting criterion is as follows. For each feature (i.e., variable) in our data set, the model arranges the data in ascending order. Then, it identifies the observation at which a split will minimize the sum of squared errors for the data. It records this as a possible splitting point. It repeats this process for each feature in the data. Each feature is now associated with one possible splitting point; of these, the model selects the one that minimizes the sum of squared errors and then performs the split. Within each resulting node, the model starts the same process over, and so on, until a complete tree is formed.

The splitting criterion is optimized by some fractional penalty term lambda, which ranges from 0.75 to 0.99. The penalty requires that splits bring the sum of squared errors *within the current node* (not overall) to a value of at most 0.75 (for example) of the original sum of squared errors *within that node*. The choice of lambda will depend on the data and the model.

The other parameter is the feature fraction. This parameter is fixed for the entire forest, whereas the penalty term may vary with each tree. The feature fraction refers to the fraction of features (variables) that are used in creating each tree. In our model, the feature fraction ranges from 0.3 to 0.9. Each tree within the forest will be created using a random sample of 0.3 (for example) of features in the data set. The feature fraction is essential to prevent overtraining, which is a significant strength of the random forest method. If it is set too high, the gains in accuracy may come at the cost of overtraining. If it is set too low, the model may fail to achieve its full possible accuracy.

Each tree in the forest is trained on a random sample of features, the size of which is determined by the feature fraction. It is also trained on a random sample of the available data, which is sampled with replacement from the original data and contains as many observations as the original data. Both these elements of randomness are supposed to prevent overfitting while preserving or improving accuracy.

The model we use will contain a fixed number of trees: 50. Theoretically, there is no downside to adding more trees (although there may be diminishing returns in accuracy). But the random forest method is computationally heavy, and our resources are limited. Fifty trees should be enough to guarantee a reasonable level of accuracy.

1. **The Data**

Our data is a time series: United States CPI monthly inflation, seasonally adjusted, 1959-2020. Time series data is notoriously tricky. It suffers from seasonality, trends, covariance, and a myriad of other maladies. Many of the classical models are explicitly designed to handle those difficulties. The random forest method is not. It is designed for classification of cross-sectional data.

With this in mind, the data we use is seasonally-adjusted. This should account for seasonality issues. It is also stationary, according to a unit root test. There is no evidence of a structural break or a time trend.

To make the data compatible with the random forest, we use the embed() function in R to convert the time series into a matrix of values. The first column is a vector of inflation observations, from January 1960 to September 2020. The second column is the same vector, from the prior month: December 1959 to August 2020. The third column is one month prior again, November 1959 to July 2020, and so on. The first column represents the *t* values, the second column *t – 1*, the third *t – 2,* and so on, for each of more than 750 observations. Each lag on an observation then, is treated as a feature in the data which is used to train the random forest. We use 11 lags. We also append a time trend term, which contains a value of 1 for the first observation, 2 for the second, etc.

Just as the random forest’s feature fraction is supposed to prevent overfitting the data, the regression tree’s splitting criterion also prevents overfitting. At each node, only one feature will ultimately determine the split. Thus, for a data set in the form of the matrix described above, it is theoretically impossible to overfit by adding too many features. Even if the matrix had 100 lags, the splits at each node would only be determined by the one feature whose choice minimized the sum of squared errors. It is very unlikely that this feature would be the 100th lag, at any node. Thus, unlike in the standard ARIMA model, the regression tree self-identifies the most important features, and allows only them to determine its fit.

1. **Tuning**

The bulk of this project consists in appropriately tuning the forest and the trees. We anticipate two special difficulties that may arise when using the random forest method on inflation data. First, an especially prominent challenge that results from the time-series nature of our data is the problem of cross-validation. Any tuning technique will require some form of cross-validation; with cross-sectional data, it is straightforward to use a k-fold cross-validation technique. With time series data, however, the process requires more careful consideration. Second, we realize that our tuned trees will only be used to forecast in the context of a random forest that we also tune. Therefore, a perfectly tuned standalone tree may not actually be the best tree to use in the context of the forest. How do we tune the trees and the forest together? This is something we need to consider.

In addition to these two concerns, we realize that there are impossibly many parameters to consider. For example, when we randomly sample the data for each tree in the forest, should we sample with replacement? Randomly sample without replacement from a fixed point in time? From a random point in time? How big would this random sample without replacement be? And none of these parameters can be tuned in a vacuum; each of these questions would have to be answered simultaneously with each other and with dozens of other such questions, regarding for instance the number of trees in the forest, the feature fraction for the forest, the splitting penalty for each tree, etc. Thus, to perfectly tune the forest would take an extraordinary amount of computation, which we do not have access to. So, we will focus our efforts on tuning the parameters outlined in the second section, above.

Instead of the k-fold cross-validation technique favored when training models on cross-sectional data, we will use an alternative that makes more sense for time series data. We will hold the most recent 48 observations as a test set for each tree and train each tree on the previous hundreds of observations. Then, we will test the tree on the last 48 observations in order to “cross-validate” it. Within the forest, each tree contains a random subsample of data and features. This means that the last 48 observations in the data may not (indeed, almost certainly will not) be 48 successive observations. Some will be repeated, some will be skipped. Nevertheless, the tree which results from the training set will at least be tested on later data, rather than earlier data.

The random forest and regression tree methods are explicitly cross-sectional models. They are not designed to handle time trends or seasonal variation or evolving processes. The theory with time-series data is that it is motivated by some underlying process, typically an autoregressive one. This process, however, is not perfectly consistent over time, and may include a time trend or seasonal variation, or may simply evolve over time, so that the process in the 1960s differs from the process in the 1990s. The random forest method is not equipped to handle this sort of data. However, the random forest method can be very effective at predicting cross-sectional data, and it is not impossible that with a few adjustments and with some careful handling, this method could present an improvement over a classical ARIMA model. To realize that potential improvement is the goal of this project.

Random forest and cross-validation are inherently cross-sectional techniques. It doesn’t quite make sense to k-fold the time series data, because you’ll end up validating past data on future data. This seems problematic because the future data theoretically has no effect on the past data. For all we know, the time series is fundamentally different in the 2010s as opposed to the 1960s. Maybe there’s a structural break, maybe there’s some kind of non-linear time trend; at any rate, it would be best to avoid validating past predictions on future data.

To minimize the computational complexity of this project, we will consider two parameters. One of these, the penalty function, relates to the regression tree itself; the other, the feature fraction, relates to the random forest.

When tuning the forest, it’s important to consider the trees and the forest itself as complementary entities. In other words, neither the tree nor the forest should be tuned in a vacuum. Instead, the tree should be tuned with the understanding that it will be used in a forest explicitly designed to accommodate overfitting. Thus, we may allow the tree to fit the data more tightly than we would if we planned to use the tree alone for forecasting.

Tin Kam Ho, "Random decision forests," Proceedings of 3rd International Conference on Document Analysis and Recognition, Montreal, Quebec, Canada, 1995, pp. 278-282 vol.1, doi: 10.1109/ICDAR.1995.598994.