Predicting U.S. Monthly Inflation with Random Forest

Submitted in partial fulfillment of the requirements for the
Bachelor of Arts degree in the Program of Liberal Studies

Alex Moran

Class of

2021

Directed by _____

(Drew Creal, Associate Professor)

ALEX MORAN

Predicting U.S. Monthly Inflation with Random Forest

I create and tune a random forest model that can compete with classical time-series models. The univariate model I create can outperform a standard ARIMA model when predicting on US monthly inflation data at a one-month forecast horizon. To create the model, I modify a standard random forest with consideration of the unique features of time-series data. Additionally, I begin to experiment with the random forest model at different forecast horizons, with time series other than US monthly inflation, and finally in a multivariate form.

Keywords: inflation, machine learning, random forest, time series.

PREDICTING THE FUTURE is hard. It makes sense, then, to explore all of the resources at our disposal when we set out to forecast a time series. The typical autoregressive (AR) and autoregressive integrated moving average (ARIMA) models are useful and valuable, both for their forecasting ability and for their specification of a time series' underlying process. However, methods from machine learning such as the random forest offer an attractive alternative also to generate good forecasts.

The purpose of this project is not to unseat valuable time-series models such as the ARIMA and the VAR, but rather to highlight the potential of machine learning in a time-series context. This potential is twofold: as I will describe, the random forest can both forecast well *and* offer insight into the process which generates a time series.

Accordingly, this paper explores 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. A random forest is typically meant for use with cross-sectional data, so this paper proposes a modification of the random forest in order to account for the peculiarities of time-series data as opposed to cross-sectional data. In the case of US monthly inflation, the modified random forest presents a sizable improvement over the standard random forest, both in terms of fit and in terms of forecast performance, as measured by root mean squared error (RMSE). Furthermore, the modified random forest presents fit and forecast results which outperform those of an ARIMA model optimized by Akaike Information Criterion (AIC); this model is referred to as an ARIMA(AIC).

One drawback of the random forest is that it does not traditionally yield theoretical insight about data. For example, the random forest would not be expected to shed light on the motivating process for a time series. This paper challenges that convention by proposing a way to interpret the internal mechanism of the random forest in terms which speak to the changing state of the dataset over time.

The modified random forest described in this paper is univariate: it takes as input only lags of inflation. This random forest was designed with US monthly inflation at the one-month prediction horizon in mind, but I also consider its applications in a variety of

contexts: as a univariate model for US monthly inflation across different prediction horizons, as a univariate model for other types of time series, and as a multivariate model for US monthly inflation at the one-month horizon.

The rest of the paper is organized in the following way. Section 1 describes the data used for the bulk of the paper, Section 2 describes the random forest as a concept in more detail, Section 3 describes the modifications made to the random forest in order to make it useful in time-series applications, Section 4 describes some results, and Section 5 looks ahead to further applications. Section 6 concludes.

1. THE DATA

The data which I consider for the bulk of this paper is a time series: United States Consumer Price Index (CPI) monthly inflation, seasonally adjusted, January 1959 – January 2020. I use the Consumer Price Index for All Urban Consumers dataset from the St. Louis Fed. Data is given at the monthly level. I produce inflation data by taking the log difference of monthly CPI data.

Since the random forest is not designed to handle the seasonality, trends, and conditional heteroskedasticity of a time series, I am especially careful to mitigate the challenging characteristics of time-series data. Accordingly, the data I use is already seasonally adjusted. There is strong evidence that the series is stationary with a negative time trend. When I forecast, I will use data from January 1989 – January 2020. In this time window, there is no evidence of a structural break, but when the data is considered at 10-

year intervals, there is limited evidence of a few possible structural breaks. See Appendix 1 for more details on the data.

For the random forest model I describe below, I use a dataset that includes the current month's inflation π_t, along with eleven lags on the current month's inflation π_{t-1} through π_{t-11}, and a *trend* term, which has a value of 1 in January 1959 and increases with each month, finishing with a value of 733 in January 2020.

2. THE RANDOM FOREST

The "random forest" refers to a collection of regression trees, which are each trained on a randomly selected subsample of features (i.e., variables). As the "forest" (the collection of trees) grows in size, it becomes increasingly accurate while also remaining resistant to overfitting. This innovation represents a major development in statistical modelling: "the accuracy increases with the addition of new trees" and yet "an increase in classifier complexity" does not lead to "overtraining" (Ho 1995). Thus, this method has the demonstrated potential to improve accuracy without the risk of overfitting—an ability which classical time-series models lack.

In Section 3, I customize the random forest for time-series applications. In this section, I merely describe the "base" random forest that I will later customize. This base random forest is not necessarily characteristic of a "standard" random forest, if such a thing exists. Rather, the "base" forest represents my idea of a random forest that might be useful

for fitting or predicting on cross-sectional data. It is my starting point as I set out to create a random forest which works better on time-series data.

Despite its title, this section will focus primarily on the concept of a regression tree. The random forest is merely a collection of these trees, each formed by randomized inputs to an identical pattern.

2.1 The Regression Tree

The purpose of the regression tree is to sort a set of dissimilar data into more-similar subsets.

Splitting as a concept. Imagine that the regression tree model that I am about to describe is faced with a large dataset. Some observations will be very similar to other observations. Some different observations will be dissimilar to those observations, but similar to each other. There may be many different clusters of data which are fairly similar to each other, but dissimilar to the data found in other clusters. But the model doesn't know any of this; instead, it is faced with a dataset which contains all of the observations together, with no distinction between observations which are similar and dissimilar to each other. The task of the model, then, is to sort the data into groups of similar observations.

It will do this sorting by reference to the conditional mean. Considering the dataset as a whole, the expected value of each observation is the mean value of observations in the full dataset. But if the data were split into two different subsets, then there could be two

different conditional means. Observations in one subset would have one expected value, the sample mean of that subset; observations in the other subset would have a different expected value, the sample mean of the second group. In effect, the model can create an expected value which is allowed to vary between observations which occupy different subsets.

Within each of these subsets, the model can split again, by the same logic as before. Or, if the model deems that a certain subset contains data which is pretty similar, it may decide not to split that subset. The model stops splitting altogether when it has produced only subsets which it regards as "good enough." Between them, these subsets will contain all of the observations of the original dataset, and none of them will share observations with any of the other subsets; they will be collectively exhaustive and mutually exclusive.

The objective function. In order to generate these ideal subsets, the tree splits the data in such a way as to minimize the sum of squared errors, as shown in the equation below.

$$min \sum_{i=1}^{n} (y_i - \hat{y})^2 \qquad \textit{where} \quad \hat{y} \equiv \text{the sample mean of the relevant subset} \tag{1}$$

The equation above is the "objective function"; it stipulates the objective which, when achieved, will perform the optimal split of the data. In effect, the model tries out different configurations of two collectively exhaustive and mutually exclusive subsets, calculates the sum of squared errors associated with each, and selects the configuration of subsets which minimize that value. Note that $\hat{y}$ in the equation is the conditional mean for the associated y_i, given a particular subset configuration. It is not the mean of the entire dataset.

The configuration of two collectively exhaustive, mutually exclusive subsets which satisfies the objective function is the ideal splitting choice for the model to make. It will specify how the data can be divided to achieve the minimum possible dissimilarity.

The model does not consider literally every possible configuration of subsets. Instead, the model considers subset configurations which it identifies in the following way.

For each feature (i.e., independent variable) in a dataset, the model arranges the data in ascending order by that feature. Once the dataset is sorted, the model treats each observation as a possible splitting point: one possible subset would contain all data above the splitting point, and the other possible subset would contain all data below that point. The model will only evaluate subset configurations which meet this condition; it will evaluate them by reference to the objective function. The goal is to divide the dataset into two complementary subsets, each of which contains only consecutive values of the given feature. This way, the split can be characterized by reference to that feature: for example, "The first subset contains all observations at which the value of independent variable *xyz* is greater than or equal to 10.05; the second subset contains all observations at which the value of *xyz* is less than 10.05." The feature selected in this scenario would be *xyz*, and the splitting point would be 10.05.

The model considers possible subset configurations in light of *each feature*. Thus, it first sorts the data by feature *xyz* and identifies the subset configuration which satisfies the objective function, should the data be sorted by *xyz*. But then the model continues: now it sorts the data by *abc* and identifies the subset configuration which satisfies the objective

function, should the data be sorted by *abc*. In effect, the model identifies one possible subset configuration for each feature in the dataset.

Then, the model considers each of these configurations once more. Of the possible configurations, it selects the one which satisfies the objective function; this configuration can be characterized by reference to one particular feature and one particular value of that feature, as mentioned above. The two complementary subsets which the model produces are called "nodes."

To recap: from the original dataset, two nodes have now been produced. These nodes were configured in such a way as to satisfy the objective function. In the process, the model selected one feature to refer to when performing this split. Now, within each resulting node, the model starts the same process over, splitting each node into two smaller nodes.

The stopping condition. The model will continue to split nodes into smaller and smaller nodes until it reaches the point where the nodes it has produced are "good enough." In the case of the base forest, how does the model decide what is good enough?

It is equipped with some stopping condition, some externally imposed parameter that dictates when the tree ought to stop splitting. I envision the "base" regression tree having a minimum node size parameter of 10 observations. This parameter requires that if a split at a particular node would produce one or more nodes which contain less than 10 observations, the model would not split at that node and it would become a "terminal node," also known as a "leaf" in the regression tree.

The parameter which imposes a stopping condition need not be a minimum node size restriction. In the "base" regression tree model I am envisioning, the minimum node size of 10 observations operates in conjunction with a penalty term. Each split in the tree is performed in order to minimize the sum of squared errors (i.e., to satisfy the objective function). A penalty requires that each split reduce the sum of squared errors *by a certain amount*, either by an absolute amount or by an amount relative to the previous sum of squared errors at the parent node (i.e., a fractional penalty). If the optimal split at a node does not satisfy the penalty condition, that node will become a terminal node.

The penalty I'm envisioning is fractional, with a value of 0.9. The penalty requires that the combined sum of squared errors within the two nodes that result from a split has a value of at most 90% of the original sum of squared errors within the parent node. The penalty prevents splits that are not meaningful while allowing for flexibility depending on the dataset.

Fitting and predicting. Each node, including each leaf, is characterized by some splitting filter or series of splitting filters. If π_t is the dependent variable, and if the independent variables are a *trend* term and π_{t-1} through π_{t-11}, then an example of such a series of filters may be

$$\pi_{t-1} \geq 0.0072377 \,\&\, \pi_{t-6} \geq 0.0059084 \,\&\, trend \geq 222 \,\&\, \pi_{t-1} < 0.0127390.$$

This is a node at the fourth level; it is the result of four prior splits. First, the tree split at the π_{t-1} value of 0.00723…. Then, the resulting node which contained observations with values of π_{t-1} greater than or equal to 0.00723… was split at the π_{t-6} value of 0.00590…,

producing two smaller nodes. Of these, the node which contained values of π_{t-6} greater than or equal to 0.00590… was split based on the *trend* value, creating two yet smaller nodes. Then, one of those was split at the π_{t-1} value, creating the node referenced above.

All observations in the dataset which pass those filters are assigned to the same node, and that node will contain only those observations which satisfy the condition that characterizes the node. Note that π_{t-1} is referred to twice in this sequence. This will often happen; each filter in the sequence is generated based on the data in the given node, independently of previous filters.

Within each terminal node, the tree will compute the sample mean of all π_t values. This sample mean will be the model's fitted value for each observation in that node, and therefore the predicted value for any out-of-sample observation which happens to satisfy the filters above. In the case of US monthly inflation, each tree's prediction for the next period's observation will be equal to the mean of the π_t values of observations at the leaf which the next period's observation *would* occupy (if the current period's values were lagged, so that the current π_t becomes π_{t-1} in the next period, etc.).

2.2 The Random Forest

The random forest is a collection of regression trees, each built according to the pattern described above. The "base" random forest consists of "base" regression trees, and I am envisioning it with two additional parameters. First, the parameter of forest size: the

base forest will contain 50 trees. Second, the feature fraction. Before I describe what the feature fraction is, let me explain what it is that makes the "random" forest so random.

A regression tree as a standalone model will consider an entire dataset, with every observation and every feature. A regression tree in the base random forest will be presented a *modified* dataset. The dataset will be modified in two ways: first, it will not contain every observation; and second, it will not contain every feature.

The base random forest will randomly sample, with replacement, from the original dataset. A regression tree in the base random forest will receive a dataset which contains just as many observations as the original dataset, but some of these observations will be duplicated; meanwhile, some observations which were in the original dataset will not be present in the modified dataset. This modification occurs on a random basis. The dataset is then additionally modified by randomly sampling the *features* as well as the observations. Only a certain fraction of features will be included in the modified dataset. Thus, the modified dataset will have as many observations as the original, but it will have less features.

The parameter which regulates how many features the modified dataset will include is the feature fraction. In the base random forest, its value is 0.7; only 70% of features will be included in the modified dataset, and these features will be selected randomly.

Each tree in the forest receives a dataset which is randomly modified as described above, and each tree receives a *different* such dataset. This randomness is the key to the impressive effect that Ho describes: "an increase in classifier complexity" does not lead to "overtraining" (Ho 1995). Randomness prevents overfitting.

Since each tree is generated from a different random dataset, each tree will have a different predicted value for a given out-of-sample observation. In the case of US monthly inflation, each tree will generate a single prediction for the next period π_t value. The base forest's prediction will be the mean average of those predictions.

3 REIMAGINING THE RANDOM FOREST

The random forest I've just described is designed for cross-sectional datasets. Here, I modify it for time-series data. The resulting model outperforms an ARIMA(AIC) model in terms of forecast accuracy.

3.1 The Data-Generating Process

The base random forest is agnostic about the process which generates its data. However, a time series like inflation is likely generated by some sort of AR process. This intuition can be harnessed to make the random forest better at predicting the future.

But I do not proceed on intuition alone: consider the evidence. On a one-month forward prediction of US monthly inflation, from January 1999 to January 2020, the ARIMA(AIC) model yields a root mean squared error (RMSE) of 0.0027, while a naïve model (in this case, the assumption that the value of inflation in a given month will be equal to that of the previous month's inflation) yields an RMSE of 0.0031. This indicates that the

ARIMA(AIC) model does a better job of modelling the data than the naïve model does, and that the data may be generated by the process which the ARIMA model describes.

Thus, there is good evidence that US monthly inflation data is generated by an AR or ARIMA process. Now as I construct the random forest model, I can assume that the data is at least partially generated by an ARIMA process. In practice, it will be difficult to generate residuals in the random forest, so I will focus on the AR part of the ARIMA model. In effect, I assume an AR process for the data.

The most basic AR process is an AR(1): an autoregression on only the first lag of a time series. Each observation in time is generated by some weight of the previous observation, plus some independent and identically distributed (i.i.d.) error. Before I turn to the real US monthly inflation data, I want to think about how the random forest would handle AR(1) data in general. If I were presented with data which I knew with certainty was generated by an AR(1) process, how could I adjust the random forest to better model this data?

3.2 Assessing Results with Simulated Data

To answer this question, I first simulate a time series based on an AR(1) process with a coefficient of 0.9. The time series I generate contains 511 observations.

Performance of the base tree. I then assess the fit given by an ARIMA(AIC) model with the fit given by the base tree described in Section 2. The fits are roughly similar; the ARIMA(AIC) has an RMSE of 1.016, while the base tree has an RMSE of 1.029.

However, I ultimately want a random forest model which will forecast well, not just one that will fit well. Therefore, using first the ARIMA(AIC), and then the base tree, I predict one period ahead for each of the last 100 observations. The ARIMA(AIC) model yields an RMSE of 1.053, while the base tree yields an RMSE of 1.166.

This result offers a baseline: any theoretically sound adjustments to the tree which yield an RMSE lower than 1.166 could indicate an improvement in the model; any adjustments which bring the RMSE below 1.053 would suggest that the model has the potential to outperform an ARIMA(AIC) model.

Improvements to the base tree. Although the base tree can fit existing time-series data with reasonable accuracy, it fails to forecast well. When it forecasts, the regression tree performs two tasks: first, it fits itself to a training set as described in Section 2. Second, once the tree is fit, it predicts based on some additional data. The prediction for the base tree is simply the mean value of the leaf which the next-period observation would have occupied.

In the base random forest model, the objective function is to minimize the sum of squared errors in a given node, based on an expected value of π_t that was merely equal to the sample mean of a possible subset. Thus, in the base tree, the nodes are designed to contain observations with similar values of π_t. Getting similar values of π_t is one

straightforward way to categorize a set of dissimilar data into more-similar subsets. But if I know that a particular process generates the data, then there is a better approach.

If I know what process generates the data in general, then what I want are not subsets of data in which the π_t term has a similar value, but subsets of data which are generated by similar particular implementations of the general process. If the process is an AR(1) process, then the objective function can become to minimize the sum of squared residuals on an AR(1) regression, rather than to minimize the sum of squared errors based on a mean expectation.

Additionally, I allow fitted values for each point in the dataset to be based on an AR(1) equation, fitted separately at each leaf. Each leaf will have a constant value and a coefficient value for the AR(1), and the fitted value of each observation in that leaf will be calculated accordingly. This creates a more flexible fit; the base tree had only as many unique fitted values as it had leaves, but this tree has a unique fitted value for each point in its training set.

Likewise, when the tree is used to predict the next-period value, that prediction will no longer be equal to the mean value of the leaf which that observation would have occupied. Instead, it will be based on what the fitted value would have been, given the constant and coefficient values associated with the leaf it would have occupied.

TABLE 1

FIT AND FORECAST RESULTS ON SIMULATED AR(1) TIME SERIES, RELATIVE TO NAÏVE MODEL

	Model Type				
	ARIMA	Base Tree	Hybrid Tree	Modified Tree	Naïve
Fit RMSE	0.991	1.003	**0.947**	0.990	1.000
Forecast RMSE	0.990	1.097	1.027	**0.975**	1.000

NOTES: Numbers refer to the RMSE generated from each model, divided by the RMSE provided by the naïve model. The lowest RMSE in each row is bolded. The first column refers to the ARIMA function optimized by AIC; the last three columns refer to different types of trees: the base tree defined in Section 1, the hybrid tree which borrows the objective function from the base tree but predicts based on an AR(1) assumption, and the modified tree which uses an AR(1) objective function as well as an AR(1) prediction. The first row refers to the in-sample fit provided by each model; the last row refers to an out-of-sample one-period ahead forecast on the last 100 observations generated, in a time series of 511 observations. For the actual RMSEs, see Appendix 4.

Results on simulated data. Thus, both the objective function used to fit the tree and the process used to forecast from the tree can be modified to accommodate the intuition that inflation data is generated by an AR(1) process. The table below shows the RMSEs that result from using four different models to generate fitted values and forecast values on the simulated time series. The first such model is the ARIMA(AIC); the second is the base tree; the third is a tree which uses the same objective function as the base tree, but which uses an AR(1) formula to forecast ("hybrid tree"); and the fourth is the tree described above, which uses AR(1) as an objective function as well to predict ("modified tree").

The model which provides the best fit, by a fair margin, is the hybrid tree. When performing splits, this tree uses the mean-based objective function from the base tree as described in Section 2, but it uses an AR(1) expression for fitting and forecasting.

However, in the pseudo out-of-sample forecast on the last 100 values of the simulated time series, the best model is the modified tree described above. This modified tree even outperforms the ARIMA(AIC) model, which is a promising result for the random forest I intend to construct. If the modified regression tree alone can outperform an ARIMA(AIC) model even on a simulated AR(1) series, it's reasonable to suppose that a random forest built from many of those same trees will be able to outperform an ARIMA(AIC) model on real-world data.

3.3 Modifying the Regression Tree

Table 1 demonstrates that the modified regression tree outperforms an ARIMA(AIC) model when forecasting on simulated AR(1) data. This subsection describes the mechanisms of that modified tree in greater detail.

The objective function. Within the initial node of the tree (i.e., the full dataset), the tree must make a decision which splits the data into two subsets. To optimize this choice, the tree refers to the AR(1) objective function. The function identifies one variable as the dependent variable (i.e., current period inflation π_t) and one variable as the first lag of that

dependent variable (i.e., π_{t-1}). Then, for each of the remaining variables, the function seeks to satisfy the following objective function.

$$min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad where \qquad \hat{y} \equiv \text{the AR(1) OLS fitted value} \tag{2}$$

Or, expressed in terms of an ordinary least squares (OLS) regression,

$$min \sum_{i=1}^{n} \left((y_i - \bar{y}) + \frac{\sum (y_i - \bar{y})(x_i - \bar{x})}{\sum (x_i - \bar{x})^2} (x_i - \bar{x}) \right) \tag{3}$$

In practice, the objective function takes all available observations and considers only three variables: one independent variable, the dependent variable, and the first lag. It sorts these observations in ascending order by the independent variable. Then, just like the base tree, it identifies many configurations of subsets and selects the one which satisfies the objective function (3). The objective is to minimize, by choice of subset configuration, the sum of squared residuals which results from this regression.

The time-series tree, like the base tree, performs this process for each feature in the dataset (i.e., for π_{t-1} through π_{t-11} and the trend term), so that each feature is associated with a sum of squared residuals value that would result from the optimal split at that feature. The tree then selects the variable and the split that satisfy the objective function and performs the split at that point.

The two resulting nodes are now each characterized by the AR(1) regression that best fits them; the goal is to reduce the imprecise fit of the standard AR function by splitting the data into subsets that are characterized by distinct AR(1) processes, thus improving the fit within each node and therefore the fit of the tree overall.

The objective function could run as many as one million times during the fitting process for each random forest. Therefore, it is worthwhile to code the function as efficiently as possible. The practical details of how I coded the AR(1) objective function are given in Appendix 2.

Tuning the penalty term. As before, the tree will need to know when to stop splitting. This section describes that process. The optimal value of the penalty term parameter will vary depending on the sample of data and features which the model receives. Proper specification of the penalty value must depend on the inputs which the tree receives; the penalty value cannot be specified externally. Instead, the penalty value must be specified *internally*, by the tree itself. The tree must be able to "self-tune."

A self-tuning model is able to train itself; it can select its own optimal parameter values. It performs this selection by training itself on a portion of the data it receives (the "training set"), and then predicting on the remaining data (the "test set"). It trains and tests many times, with different parameter values each time. The parameter values which yield the best forecasts on the test set are designated as optimal. This process of dividing the data into a training and a test set in order to select optimal parameters is known as "cross-validation."

A model which forecasts on time-series data demands a cross-validation method which is sensitive to the peculiarities of time-series data. Not only is time-series data serially correlated, but real-world time-series data is not generated by the same process

across all points in time. In the case of US inflation data, for instance, Stock and Watson (2007) argue that the pre-1984 data is fundamentally different from the post-1984 data.

Therefore, I use the following cross-validation technique. Rather than randomly select the training set and the test set, I designate the most recent 11 observations that a tree receives as a test set for the tree, and I designate the remaining, previous observations as the training set. Then I repeatedly train the tree on the training set and forecast on the test set, using a different penalty value each time. The goal is to find the penalty value which minimizes the RMSE of this forecast; this is the "optimization function."

To decide which values of the penalty term to consider at all, I specify a "parameter space": in this case, a series of values from 0.70 to 0.99, increasing by increments of 0.005. The model must select the optimal parameter value within this parameter space, and it will do so by reference to the optimization function described above. The parameter space and the optimization function have now been specified; all that remains is to specify the "search method."

The search method dictates how the model will search the parameter space. The most straightforward option is a grid-based search, where the model validates on each parameter value and selects the one that is optimal. However, this method is inefficient. The parameter space I've specified for the penalty term contains 59 values. Thus, the tree model would have to fit 59 different trees on the training set and validate each of those 59 trees on the test set of eleven observations. Then, the model would identify the optimal penalty value and generate an additional tree on the complete dataset; in all, a grid-based

search method would need to fit 60 trees for each tree which actually appears in the forest. This is needlessly inefficient.

Instead, consider that it is extremely unlikely for forecast RMSEs to be distributed randomly across penalty values. It is far more likely that there are certain regions of the parameter space which tend to produce forecasts with low RMSE values, and other regions which tend to produce forecasts high RMSE values. Therefore, it would be more efficient to identify promising regions of the parameter space and to explore these regions thoroughly, while dedicating less attention to less-promising regions.

Accordingly, rather than specify a grid-based search method, I specify a Bayesian one which can update its focus to spend more resources exploring promising regions of the parameter space, and less resources exploring less-promising regions. The specific search method I specify is a Tree-structured Parzen Estimator (TPE) as described by Bergstra et al. in 2011. For details, see Appendix 3.

3.4 Specifying the Random Forest

I have now described the modifications I make to the regression tree in order to allow it to outperform the ARIMA(AIC) model on the simulated data as shown in Table 1. Now I describe the modifications I make to the forest itself.

Data sampling. In the case of the base forest, data is sampled with replacement from the full dataset. Given that a time series is serially correlated, this method of data sampling

seems unlikely to yield trees which forecast well. Instead, using a technique called a "block bootstrap," I randomly sample the *length* of the dataset, choosing a value between 50 and 100 from a uniform distribution. Each tree receives a dataset of a randomly selected length, whose last value is fixed at the most recent value in the series.

For example, if I want to forecast inflation for January 2000, the tree would receive the most recent *x* observations from the dataset, ending with data from December 1999, where *x* is a random number from 50 to 100. This method of sampling preserves the integrity of the time series, while still allowing the model to benefit from random sampling. Because this approach yields a smaller dataset, I reduce the minimum node size parameter from 10 to 5 observations.

Feature sampling. The data which each tree receives is randomly chosen as I've just described, and so too are the features which each tree considers. The variables in the base forest are π_t through π_{t-11}, plus a time trend. The purpose of feature sampling is to mitigate the impact of correlation between variables. For example, suppose that π_{t-2} and π_{t-3} are highly correlated (indeed, this is expected). In a model which always considers π_{t-3} only when it also considers π_{t-2}, the effect of the π_{t-3} term is likely to be understated, since much of the impact is likely already accounted for in the π_{t-2} term. Thus, the effect of π_{t-2} may be overstated, while the effect of π_{t-3} may be understated. Feature sampling mitigates this effect by allowing the model to sometimes consider both π_{t-2} and π_{t-3} together, to sometimes consider only π_{t-2} and sometimes only π_{t-3}, and sometimes to consider neither of them as it considers other variables instead.

The feature fraction in the modified forest is 0.7, as in the base forest. However, the trend term and the π_{t-1} term are exempt from feature sampling; they are always considered, in every tree. The trend is always included because it is unlikely to be correlated with any of the π terms and may include information that the π terms are unable to express. The π_{t-1} term is always included because it is necessary in order to calculate the objective function. Of the remaining ten variables (i.e., π_{t-2} through π_{t-11}), only seven are randomly selected for each tree.

Miscellaneous specifications. I specify that this forest consists of 50 trees. The penalty parameter within each tree is self-tuning as described above and is supplemented by the imposed stipulation that no node contain fewer than five observations. Each tree considers slightly different data and slightly different features. Each tree will therefore generate slightly different fits and predictions for each observation. The forecast of the forest overall will be the mean average value of the predictions from each tree. The efficacy of this approach is described in greater detail by Leo Breiman (1996).

For the rest of the paper, I will refer to the random forest I have just described as the "modified" random forest.

4 RESULTS

The primary goal of this project is to construct a random forest that outperforms an ARIMA(AIC) model. Secondary goals are to outperform an AR(1) model, the "base"

random forest described in Section 2, and a naïve one-month-ahead forecast. These goals are narrowly construed: the context is a one-month forward forecast of US monthly inflation data, from January 1999 to January 2020. Performance is gauged by RMSE.

In this context, the modified random forest outperforms all four other models. This result and its implications are discussed below; the broad potential of this approach in contexts other than US monthly inflation is discussed in Section 5.

4.1 Performance

TABLE 2

RMSE FORECAST RESULTS ON US MONTHLY INFLATION DATA, JANUARY 1999 – JANUARY 2020, RELATIVE TO NAÏVE FORECAST

Model Type				
Modified Forest	ARIMA	AR(1)	Base Forest	Naïve
0.866	0.880	0.887	0.951	1.000

NOTES: Numbers refer to the RMSE generated from each model, divided by the RMSE provided by the naïve model. Forecast RMSEs are on a one-month forward forecast from January 1999 to January 2020, performed by five different models. The first column refers to the modified random forest I describe in Section 3. The ARIMA model is optimized by AIC. The "base" random forest is as described in Section 2. The naïve forecast simply predicts that next month inflation will be the same as current month inflation. RMSEs for the random forest and the "base" random forest are not replicable. The lowest RMSE is bolded. For the actual RMSEs, see Appendix 4.

Table 2 shows the RMSE values for the forecasts produced by the modified random forest model and four competing models. The data is US monthly inflation; the forecast period is January 1999 to January 2020. The table demonstrates that the random forest model described in Section 3 actually accomplishes the immediate goals outlined above; it outperforms every model listed.

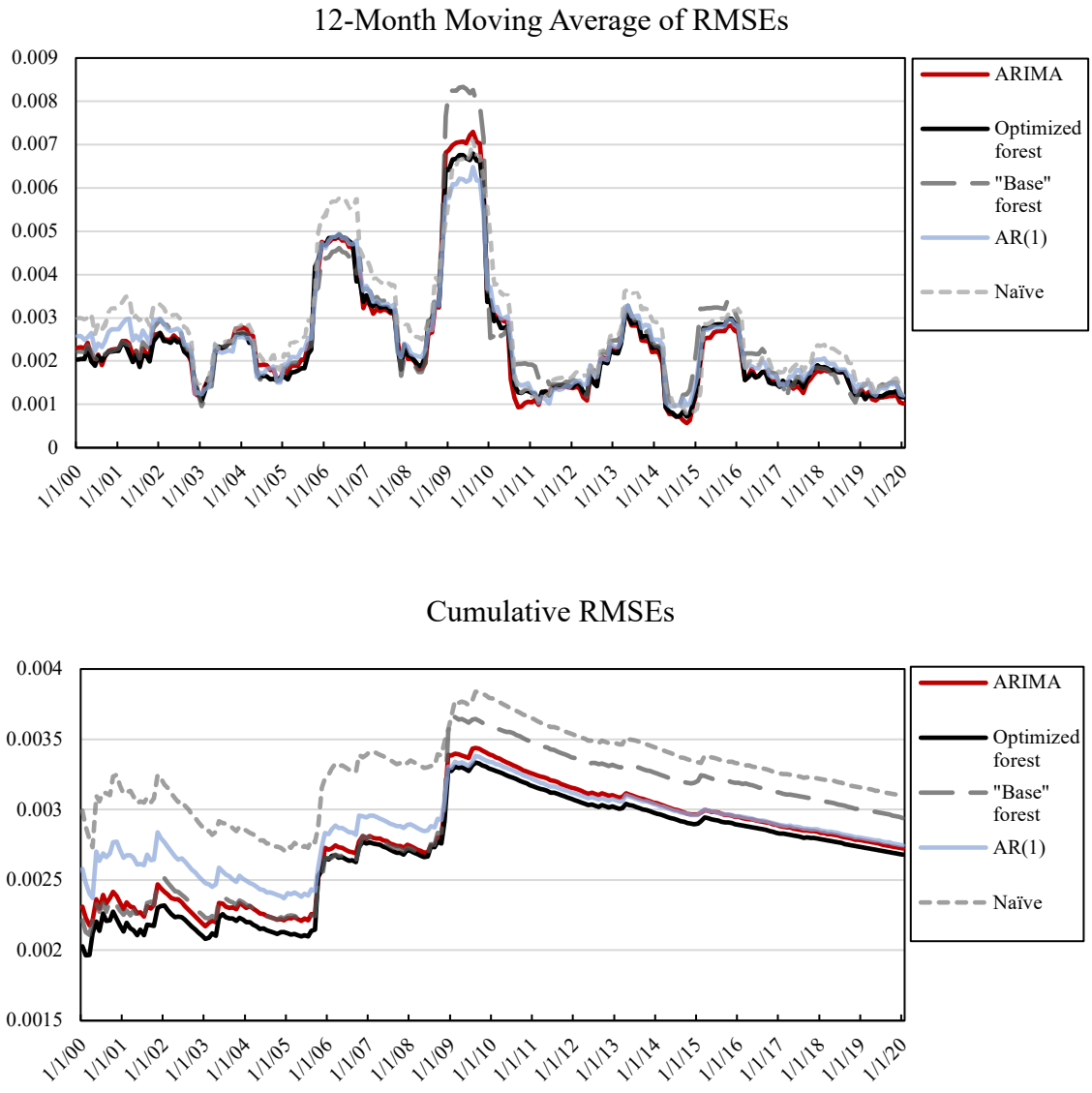
Due to the random nature of the forest, these results are not completely replicable: the RMSE associated with the random forest's forecast will be slightly different each time the forest is run. However, the performance will tend to form a distribution of RMSEs that includes the RMSE listed above.

Over the full 20 years of the forecast period, the random forest outperforms the other models. Additionally, the random forest consistently outperforms the other models *within* the forecast period, as Graph 1 shows. Lower RMSE values indicate a better model.

Graph 1 also shows that all five models perform well at the same time and perform poorly at the same time as the others. For some models this makes sense: the AR(1) model, the ARIMA model, and the modified random forest all have an AR assumption at their core. More surprising is that the "base" forest follows the same trend: it performs well and poorly when the other models perform well and poorly. This implies some sort of inherent "predictability" in the data; at times it is simply easier or harder to predict, regardless of the model, as described by Stock and Watson (2007).

GRAPH 1

12-MONTH MOVING AVERAGE AND CUMULATIVE RMSES FROM FIVE MODELS



NOTES: These charts refer to the RMSEs produced by the inflation forecasts of five different models: the ARIMA model optimized by AIC, which appears in red; the modified forest, which appears in black; the "base" forest, the simple AR(1) model, and a naïve one-month ahead shift, which each appear in a different shade of gray. Lower RMSE values indicate a better model.

4.2 Interpretation

Section 3 explained at length the differences between the modified random forest and the base random forest; below, I revisit those differences in light of the results. The question is, what accounts for the improved performance?

Table 3 highlights the significant improvement made, both in terms of RMSE and in terms of frequency of good fits as opposed to a baseline offered by the ARIMA(AIC) model. In Section 3, I outlined my expectations of how the changes I made from the base forest would improve forecasting performance. Now, I dig into that improved performance and examine what accounts for it.

TABLE 3

COMPARING BASE FOREST TO MODIFIED FOREST

	Model Type		**Improvement**
	Base Forest	Modified Forest	
Forecast RMSE	0.0029361	0.0026732	-8.96%
Better fit than ARIMA: count	114	128	14
Better fit than ARIMA: frequency	0.450593	0.505929	12.28%

NOTES: This chart compares the "base" forest from Section 2 with the modified forest from Section 3. In the second and third rows, both models are compared to an ARIMA(AIC) baseline. The second row describes how many predictions, out of 253 in the forecast period, were more accurate than those of an ARIMA(AIC) model. The third row describes this as a decimal value.

Infrequent splits. Consider the number of "real" trees in each forest across the prediction period. By "real" trees, I mean trees which actually divide the data, i.e., trees which contain more than one node. In the base forest model, out of 50 typical trees, nearly all 50 of them will be real. By contrast, in the modified forest model, only about 4.6 trees per 50 trees are real. In other words, a typical tree is less than 10% likely to perform any splits whatsoever.

Intuitively, this makes sense. If the data is actually generated by an AR(1) process and the default fit suggested by the tree is an AR(1) process, then the tree will not *need* to split very often. However, if, as in the base forest, the default fit suggested by a tree is simply the mean of all observations, then the tree is extremely likely to suggest splits when faced with a dataset generated by an AR(1) process. Indeed, this proves to be the case: the base forest fits a tree over 99% of the time. Despite their reasonably similar fits (the forecast RMSEs are within 10% of each other), the two models are behaving in very different ways.

Specifically, the modified forest isn't doing much to change the AR(1) model which underlies it. For the more than 90% of trees which do not split, each tree assumes a single AR(1) process for the entire dataset; in other words, more than 90% of the time, the trees could be replaced by a simple AR(1) model.

Recall from Table 2 that the AR(1) model produces a worse forecast than the ARIMA(AIC) model. Yet the modified forest which is based on the AR(1) model outperforms the ARIMA(AIC) model. This performance boost is due to the 9.19% of trees which actually differ from a straight AR(1) model, and to the randomness of the features and data which the trees are fed.

By judiciously choosing when to alter the AR(1) model and when to let it alone, the forest outperforms, not just the AR(1) model itself, but also the ARIMA(AIC) model. Small adjustments to a good model are all that is needed to improve it.

Feature importance. Another difference between the base forest and the modified forest is more-intuitive feature selection. A brief review: at each splitting point, a tree selects only one value of one feature to split by. The model deems the feature in question more influential than the other features when it comes to motivating the data at the particular node. In the random forest, the frequency with which each feature is selected offers an approximation of feature importance.

By "feature importance" I mean the frequency with which each feature is referred to within the *leaves* of a given tree or a given forest. Each leaf is created according to a sequence of splitting conditions, which track every split that was necessary for the creation of that leaf. Different leaves, therefore, may be characterized by sequences which are identical in the first few terms and only diverge in the later terms.

Accordingly, splits which occur early in the formation of the tree are automatically given greater weight by this metric: if one node is ultimately the ancestor of five different leaves, its sequence will appear five times when feature importance is computed.

Table 4 shows the absolute and relative importance of each feature in both the base tree and the time-series tree. By "absolute importance" I mean the number of times a feature is mentioned within the leaves of a tree. By "relative importance" I mean the number of

times a feature is mentioned within the leaves of a tree, relative to the number of times any feature is mentioned within the leaves of that tree.

As Table 4 shows, the typical base tree and the typical time-series tree prioritize different features. For example, the average "real" base tree assigns the highest importance to the feature π_{t-1}, selecting this feature at nearly 2.5x the rate at which the time-series

TABLE 4

COMPARING THE AVERAGE "REAL" BASE TREE AND THE AVERAGE "REAL" TIME-SERIES TREE

	Base Tree		**Time-Series Tree**		
	Absolute Importance	Relative Importance	Absolute Importance	Relative Importance	Relative Importance vs. Base Tree*
trend	1.551	6.44%	2.201	27.27%	423.63%
π_{t-1}	8.405	34.88%	1.178	14.59%	41.84%
π_{t-2}	3.019	12.53%	1.567	19.42%	155.01%
π_{t-3}	0.787	3.26%	0.278	3.44%	105.41%
π_{t-4}	0.692	2.87%	0.368	4.56%	158.69%
π_{t-5}	0.905	3.76%	0.185	2.29%	60.97%
π_{t-6}	2.064	8.56%	0.273	3.38%	39.43%
π_{t-7}	1.172	4.86%	0.105	1.30%	26.72%
π_{t-8}	0.912	3.78%	0.560	6.93%	183.33%
π_{t-9}	2.552	10.59%	0.710	8.80%	83.08%
π_{t-10}	0.757	3.14%	0.155	1.92%	61.04%
π_{t-11}	1.285	5.33%	0.494	6.11%	114.63%
Total	**24.102**	**(~100%)**	**8.073**	**(~100%)**	**---**

NOTES: The absolute importance refers to the number of times a feature is referred to across all *leaves*. So, if a feature forms the basis for the first split, it will be referred to at least once in each leaf of that tree. Calculating importance in this way effectively weights for feature importance; a feature which forms the basis for the first split is more important to the model than a feature which is selected for a split at a depth of ten or twelve nodes.

*The relative importance term of the average time-series tree as opposed to the relative importance listed for the average base tree. Normalizes for the fact that the base tree tends to be deeper.

tree selects it. The time-series tree, by contrast, selects the *trend* term most often, at over 4x the rate at which the base tree selects it.

Both discrepancies can be explained. The base tree, compared to the time-series tree, favors the π_{t-1} term for two reasons. The first reason is that the time-series tree tends not to select the π_{t-1} term very often. The time-series tree has an objective function and a forecast mechanism which *already* assume an AR(1) process for the data; the fit is unlikely to improve very much by splitting the data based on a π_{t-1} value. The second reason is that the base tree has an objective function and a forecast mechanism which assume a mean-based fit. When faced with data generated from an AR(1) process, the base tree will tend to grab at the first lag term as the most important feature.

For the second discrepancy, there are probably a few explanations, but the most plausible is that the time-series tree selects the *trend* term so often because it uses an AR(1) objective function. If the data is generated by an AR(1) process, then a tree which performs splits by minimizing errors on an AR(1) equation is not likely to split the data according to the first lag term. In an AR(1) process, all the lag terms are going to be highly correlated with each other; thus, if the model is unlikely to split according to the first lag term, it is also unlikely to split according to any other of the individual lag terms. The *trend* term,

however, may not be so tightly correlated, and by default it will be the most likely to be selected.

The modified random forest forecasts US inflation much better than the base forest does; thus, my intuition is that the features the modified forest selects are more likely to reveal the process which actually generates the data. Recall from Section 1 that the data seems to contain a negative time trend; thus, when it selects the *trend* term at such a high rate, the time-series tree is behaving in a reasonable way. This evidence reinforces my intuition: the features which the modified forest tends to select are probably the features which play the largest role in generating the US monthly inflation time series.

Evidence about the data process. If my intuition holds, feature importance could be an extremely valuable tool for analysis. An analysis of feature importance is easy to perform, and observable trends in feature importance over time offer clues to the changing process which the model suggests.

For instance, if the *trend* term has a relative importance of 50% for the first several years of the forecast period, but then falls to a relative importance of only 15% in the later years, that would suggest a change in the underlying structure of the time series. Specifically, it would indicate that short-run structural changes are more significant than autoregressive relationships for the first few years of the forecast period, but then in the later years, the short-run structure is similar enough that the autoregressive trends begin to become more reliable on their face, without requiring a break based on the trend term.

As it turns out, a structural change of this kind is actually observable in Graph 2. Graph 2 displays a relative importance analysis of each feature from a non-sampled random forest during the forecast period. Although the forest I described in Section 3 randomly sampled the length of the dataset, the forest I use for Graph 2 does not sample the dataset at all; instead, it uses all available data, right up to the month which precedes the forecast month. I tried to do a similar analysis with the forest I described in Section 3, but the typical forest simply did not have enough "real" trees to generate meaningful data for this analysis.

The graph shows a 12-month moving average of feature importance, displaying the top three features at every month. There are clearly two different periods in the graph: in the pre-2007 period, the *trend* term at first has a relative importance hovering between 40 and 50%, while π_{t-1} and π_{t-10} round out the top three features. The π_{t-1} term is more important than the π_{t-10} term, but not by much.

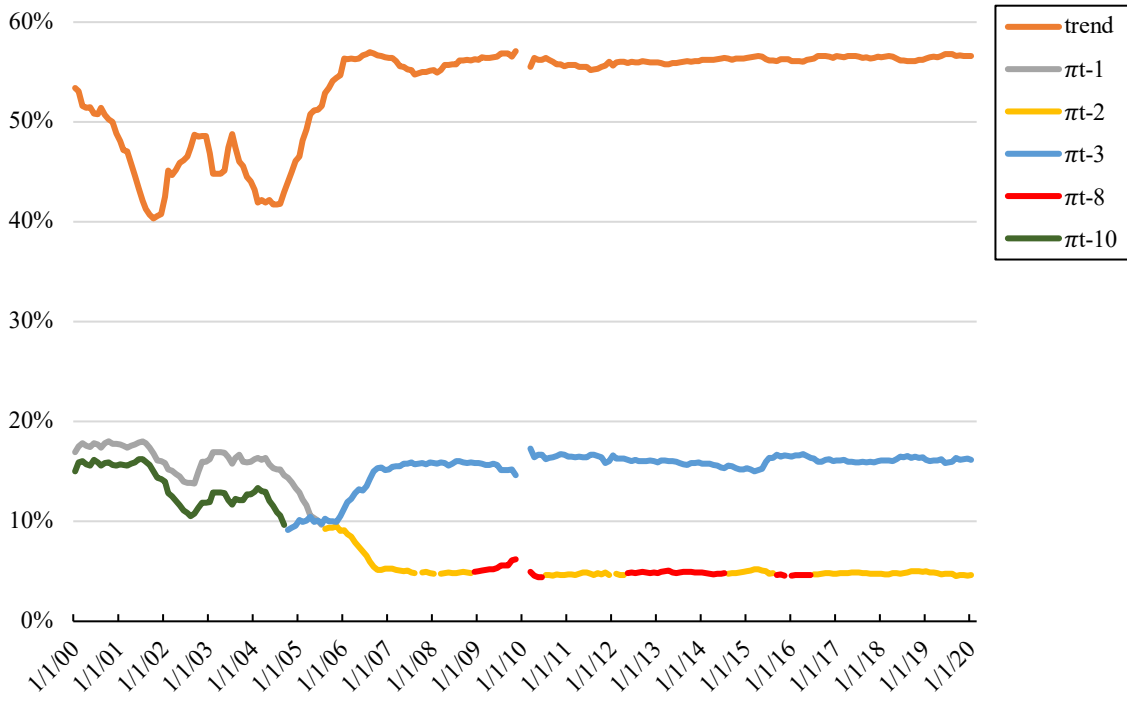
In the post-2007 period, the *trend* importance jumps up to hover around 55%, and the π_{t-1} and π_{t-10} terms fall out of importance in favor of the π_{t-3} term. No third term ever achieves more than 5% relative importance. This analysis actually does suggest some kind of structural change right around 2006 or 2007. It seems to indicate that during the first few years, inflation is a somewhat autoregressive process, while during the last few years, information about which month an observation is from is likely to be more valuable than information about what the previous month's inflation was.

Feature importance analysis is very intuitive, both in its presentation and in its performance. When I say it is intuitive in its presentation, I mean that its interpretation is straightforward; a glance at Graph 2 is sufficient to convey the point. And by intuitive in

its performance, I mean that this type of analysis makes good sense as a valid way to think about the output of the random forest. The random forest does perform splits according to certain features; the features it selects to split by are in fact the features which have the greatest influence on the composition of the data at the time of the split.

GRAPH 2

RELATIVE FEATURE IMPORTANCE FROM JANUARY 2000 TO JANUARY 2020, 12-MONTH MOVING AVERAGE



NOTES: This chart relies on the modified random forest, with no data sampling. It displays the three most important features for each month, with a measure of their relative importance. The data displayed is a 12-month moving average.

Machine learning tends to be thought of as a "black box": a machine learning model takes input and produces output, without making any claims about the economy or justifying itself theoretically. The type of analysis presented in Graph 2 represents a release from that restricting framework: the random forest can actually provide information about the economy. Through this type of analysis, the random forest model moves beyond its "black box" designation and into the realm of economic models.

5 FURTHER APPLICATIONS

The random forest laid out in Section 3 performs well under a very specific set of conditions: it outperforms the ARIMA(AIC) model on predictions of US monthly inflation data at the one-month horizon, where performance is measured by RMSE. Since the model was explicitly designed to function well under the precise conditions just described, its success should not be a surprise. Its performance in other contexts, however, is mixed.

5.1 The Random Forest in Different Contexts

All results are summarized in Table 5. Forecast performance is measured by RMSE, relative to a naïve forecast.

Different horizons. The modified random forest outperforms the ARIMA(AIC) when forecasting US monthly inflation at the 3-, 6-, and 12-month horizons. This result is

encouraging, but expected, given that the model was designed with US monthly inflation in mind: the change in horizon does not affect the underlying process, so it should not affect the relative success of the forest model as compared with the ARIMA(AIC) model.

TABLE 5

RMSE FORECAST RESULTS ON DIFFERENT TYPES OF DATA, RELATIVE TO NAÏVE FORECAST

	Model Type				
US monthly inflation data: horizons	Modified Forest	ARIMA	AR(1)	Base Forest	Naïve
1 month	**0.866**	0.880	0.887	0.951	1.000
3 months	0.720	0.742	0.768	**0.715**	1.000
6 months	**0.699**	0.733	0.768	0.701	1.000
12 months	0.704	0.707	0.749	**0.684**	1.000
One-month horizon: other time series					
US unemployment, Jan 1990 – Jan 2000	1.007	**0.962**	1.017	1.001	1.000
US 3-month Treasury rate, Jan 1985 – Jan 1995	**0.869**	0.928	0.872	0.977	1.000
UK inflation rate, Jan 2015 – Jan 2020	0.618	**0.324**	0.715	0.620	1.000

US monthly inflation: multivariate model	Modified Forest	VAR	Base Forest	Naïve
Predicted from Jan 1999 – Jan 2020	0.871	**0.679**	0.927	1.000

NOTES: Numbers refer to the RMSE generated from each model, divided by the RMSE provided by the naïve model. The lowest RMSE in each row is bolded. For US monthly inflation data at horizons other than one month, the AR(1) model may be thought of as relying only on the most recent available observation of inflation in order to predict 3, 6, or 12 months in advance. The ARIMA is an ARIMA(AIC). The multivariate case substitutes a VAR of up to six lags, optimized by AIC, for the ARIMA(AIC) and excludes the AR(1) model entirely. For the actual RMSEs, see Appendix 4.

Different time series. I also consider the following time series at the one-month horizon: the US monthly unemployment rate from January 1990 to January 2000, the US 3-month Treasury rate from January 1985 to January 1995, and the UK inflation rate from January 2015 to January 2020.

A word about the data here: neither the US unemployment rate series nor the US 3-month Treasury rate series are stationary. To make the data more workable, I de-trended and then differenced both series; the resulting series were stationary. I then predicted on that stationary series, using each of the four models in Table 5, and transformed the forecasts back. For the naïve values, I did not transform the data, but simply moved the existing time series one period forward. The UK monthly inflation data was stationary, so I did not transform it.

At least in these three contexts, the random forest model compares perfectly well with the ARIMA(AIC) or other models. This result is encouraging because it indicates that the random forest performs well with different types of time-series data. The US monthly unemployment rate and the US 3-month Treasury rate are non-stationary and, unlike inflation, rather sticky. The unemployment rate is not going to change very much from one month to the next, nor is the Treasury rate going to move as quickly and seemingly

randomly as inflation does. Although UK inflation is probably similar to US inflation, it is still a fundamentally different time series which may be generated by a different process.

Despite these differences, the random forest outperforms a simple AR(1) model in all three cases. I note the performance relative to the AR(1) model because the random forest I use is built on an AR(1) foundation. These results show that the changes I've made add value to the AR(1) model by judiciously choosing when to split the data into subsets and when to model the entire dataset as a single AR(1) process.

Multivariate model. The univariate model from Section 3 compares favorably with the univariate ARIMA(AIC) model. Here, I consider how well a multivariate forest model of the same description as the one in Section 3 compares to a multivariate VAR model when it comes to predicting US monthly inflation at a one-month horizon. The variables considered are a *trend* term and six lags on each of π_t (i.e., π_{t-1} through π_{t-6}), the 10-year US Treasury rate, the 3-month US Treasury rate, the unemployment rate, and the natural rate of unemployment, as given by the St. Louis Fed. The VAR referenced in Table 5 considers the same variables, each with six lags; so too does the "base" forest. The naïve forecast is simply a one-month ahead shift of the time series, as before.

5.2 Implications

The good news for the model is that the modified random forest outperforms ARIMA(AIC) models in forecasting US inflation, at every horizon. Furthermore, the

random forest almost always outperforms a naïve model, in all contexts, and shows the ability to outperform the ARIMA(AIC) model in the context of the 3-month Treasury rate. The bad news for the model is that it does not outperform the ARIMA(AIC) in every case, and occasionally does worse than the "base" forest, especially at different horizons. Worse yet, as a multivariate model it fails horribly; it is actually worse as a multivariate model than it is as a univariate model. But this is a model specifically optimized for a particular forecasting scenario: US monthly inflation at the one-month horizon. Its success in other conditions is encouraging, and its failure cannot be too disheartening.

With some modifications, the modified random forest is likely to be able to challenge univariate models in any context. Some immediate ideas for improvement: eliminate or modify the data sampling procedure; perhaps different time series call for different sampling techniques. Change the feature fraction; possibly different time series call for a different degree of feature randomness in the model. Change the objective function; especially at different horizons, a different objective function may produce better results.

Whether the model can ever compete with multivariate models is an open question. Probably, it will tend to perform better in contexts with more data, but economic time series are not exactly large datasets.

Interestingly, above, it seems like the base forest does not grow worse with a lengthening horizon, while the modified forest and the other models do. I'm not sure why this is, but it once again suggests the forecasting potential latent in the random forest.

6 CONCLUSION

The random forest approach to time-series forecasting is promising. A few theoretically sound modifications to an existing machine learning method yield a model that can outperform an ARIMA(AIC) model. Furthermore, even a random forest customized to achieve a narrow forecasting goal is able to perform well in other time-series applications; this suggests its potential for time-series use in general.

Its value lies not just in its predictive power, but also in its ability to offer economic intuition through its expression of the relative importance of various features at various points in time. Graphs of feature importance such as Graph 2 above can indicate the presence and fluctuation of economic trends. Thus, the random forest should be valued not just for its predictive power, but for its ability to provide economic insight.

The potential of a random forest in a time-series context is relatively untapped, but it has demonstrated the capacity to outperform classical univariate models. With some more adjustments, it is perfectly plausible that it could outperform multivariate models as well and open the door to new possibilities for time-series forecasting.

REFERENCES

Bergstra, James et al., "Algorithms for Hyper-Parameter Optimization," Proceedings of the 25th Annual Conference on Neural Information Processing Systems, 2011, pp. 2546-2554 vol. 24.

Board of Governors of the Federal Reserve System (US), 1-Year Treasury Bill: Secondary Market Rate [TB1YR], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/TB1YR, September 10, 2020.

Board of Governors of the Federal Reserve System (US), 10-Year Treasury Constant Maturity Rate [DGS10], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/DGS10, September 10, 2020.

Board of Governors of the Federal Reserve System (US), 3-Month Treasury Bill: Secondary Market Rate [TB3MS], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/TB3MS, September 10, 2020.

Breiman, Leo. "Bagging Predictors." *Machine Learning*, vol. 24, 1996, pp. 123–140. *Springer*, https://link.springer.com/content/pdf/10.1023/A:1018054314350.pdf. Accessed 20 Mar. 2021.

"Coding Random Forests in 100 Lines of Code." Statworx. June 5, 2019. Accessed October 9, 2020. https://www.statworx.com/blog/coding-random-forests-in-100-lines-of-code/.

"Coding Regression Trees in 150 Lines of Code." Statworx. November 9, 2018. Accessed October 9, 2020. https://www.statworx.com/blog/coding-regression-trees-in-150-lines-of-code.

De Baets, Bernard, et al. "On Estimating Model Accuracy with Repeated Cross-Validation." *BeneLearn 2012: Proceedings of the 21st Belgian-Dutch Conference on Machine Learning*, 2012, pp. 39–44.

Federal Reserve Bank of St. Louis, 10-Year Breakeven Inflation Rate [T10YIE], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/T10YIE, September 10, 2020.

Haider, Adnan, and Muhammad Nadeem Hanif. "Inflation Forecasting in Pakistan Using Artificial Neural Networks." *Pakistan Economic and Social Review*, vol. 47, no. 1, 2009, pp. 123–138. *JSTOR*, www.jstor.org/stable/25825345. Accessed 21 Mar. 2021.

Hansen, Bruce E. "Tests for Parameter Instability in Regressions with I(1) Processes." *Journal of Business & Economic Statistics*, vol. 20, no. 1, 2002, pp. 45–59. *JSTOR*, www.jstor.org/stable/1392149. Accessed 21 Mar. 2021.

Inoue, Atsushi, and Lutz Kilian. "How Useful Is Bagging in Forecasting Economic Time Series? A Case Study of U.S. Consumer Price Inflation." *Journal of the American Statistical Association*, vol. 103, no. 482, 2008, pp. 511–522. *JSTOR*, www.jstor.org/stable/27640075. Accessed 21 Mar. 2021.

Office for National Statistics, CPIH Index 00: All Items 2015 = 100, retrieved from ONS; https://www.ons.gov.uk/economy/inflationandpriceindices/timeseries/l522/mm23, February 20, 2021.

"Package 'randomForest'." CRAN. March 25, 2018. Accessed October 9, 2020. https://cran.r-project.org/web/packages/randomForest/randomForest.pdf.

Picard, Richard R., and R. Dennis Cook. "Cross-Validation of Regression Models." *Journal of the American Statistical Association*, vol. 79, no. 387, 1984, pp. 575–583. *JSTOR*, www.jstor.org/stable/2288403. Accessed 8 Mar. 2021.

"Time series forecasting with random forest." R-bloggers. September 25, 2019. Accessed October 20, 2020. https://www.r-bloggers.com/2019/09/time-series-forecasting-with-random-forest/.

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.

"Tuning Random Forest on Time Series Data." R-bloggers. November 21, 2019. Accessed October 20, 2020. https://www.r-bloggers.com/2019/11/tuning-random-forest-on-time-series-data/.

U.S. Bureau of Labor Statistics, Consumer Price Index for All Urban Consumers: All Items in U.S. City Average [CPIAUCNS], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/CPIAUCNS, September 11, 2020.

U.S. Bureau of Labor Statistics, Consumer Price Index for All Urban Consumers: All Items in U.S. City Average [CPIAUCSL], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/CPIAUCSL, September 11, 2020.

U.S. Bureau of Labor Statistics, Unemployment Rate [UNRATE], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/UNRATE, September 10, 2020.

U.S. Bureau of Labor Statistics, Unemployment Rate [UNRATENSA], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/UNRATENSA, September 10, 2020.

U.S. Congressional Budget Office, Natural Rate of Unemployment (Long-Term) [NROU], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/NROU, September 10, 2020.

U.S. Congressional Budget Office, Natural Rate of Unemployment (Short-Term) [NROUST], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/NROUST, September 10, 2020.

"Using k-fold cross-validation for time-series model selection." Stats Stack Exchange. August 10, 2011. Accessed October 9, 2020. https://stats.stackexchange.com/questions/14099/using-k-fold-cross-validation-for-time-series-model-selection/.

University of Michigan, University of Michigan: Inflation Expectation [MICH], retrieved from FRED, Federal Reserve Bank of St. Louis; https://fred.stlouisfed.org/series/MICH, September 10, 2020.

APPENDIX 1: THE DATA

Taming the data. I use seasonally adjusted data. There is very strong evidence that the series is stationary: an Augmented Dickey-Fuller test rejects the null hypothesis of a unit root at the 5% level. There is likewise evidence of a time trend: a Spearman rank correlation test rejects the null hypothesis of no time trend and suggests the presence of a negative time trend in the data. Furthermore, a least squares regression of the data against a time trend and a constant suggests that time trend is significant at the 5% level.

The forecast period I consider for the bulk of the paper is January 1999 to January 2020. During this period in particular, there is limited evidence of a structural break in the series. Neither in the longer series from January 1959 to January 2020, nor in the shorter series from January 1989 to January 2020 is there evidence of a structural break, according to a supF test as described by Hansen in 1992. (Since the model only considers data from at most 8 years previous, only data from about January 1989 to January 2020 is relevant for the forecast period January 1999 to January 2020.) At 10-year intervals, beginning with the interval January 1989 – January 1999 and increasing by increments of one year, ending with January 2010 – January 2020, there is limited evidence of a few possible structural breaks.

Building the dataset. To make the time-series data usable by the random forest model, I 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 π_t values, the second column π_{t-1}, the third π_{t-2}, and so on, for each observations. Each lag on an observation is treated as a feature in the data. I use 11 lags. I also append a time trend term, which contains a value of 1 for the first observation, 2 for the second, etc. Each row now consists of one observation, containing a π_t term with 11 lags and a time trend.

Eleven lags in an ARIMA model on this data would likely lead to an overfit; the best ARIMA model by AIC is an ARIMA(4,0,1). But since the regression tree self-identifies the most important features and allows only them to determine its fit, there should be no harm to adding more lags than are likely to be necessary.

APPENDIX 2: OPTIMIZING THE AR(1) OBJECTIVE FUNCTION

R has a package to compute the sum of squared residuals (SSR) on any regression, but this is unnecessarily cumbersome for my purposes. Instead, the way the tree computes its optimal split point permits a more efficient shortcut. The tree operates by arranging the data in ascending order by some independent feature. Now the data must be split at each value of that feature in order to generate the SSR for each split. For the first configuration, imagine that the first subset contains 0 observations, and the second subset contains all *n* observations at the node in question. In the second configuration, the first subset will contain 1 observation, and the second subset will contain $n - 1$ observations, and so on. There will be *n* total configurations, with *n* total associated SSR values.

Each successive configuration differs from the last by *only one observation*. Each successive configuration is formed by moving one observation from the second subset of

the previous configuration into the first subset of the current configuration; otherwise, the two subsets are unchanged across successive configurations.

This means that instead of recalculating the SSR at every data/observation pair, I can simply *update* the previous SSR. I retain the previous subset values and sample means, one each for above and below the previous splitting point. I identify the one observation that switches from the "above" vector to the "below" vector, and I update the previous subset vectors and means accordingly. Then I calculate the SSR, given the updated vectors and means. This method is much more efficient than repeatedly using the R function.

In practice, the relevant equation is this one:

$$min \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \qquad where \qquad \hat{y} \equiv \text{the AR(1) OLS fitted value} \tag{2}$$

Specifically, $\hat{y}_i$ is

$$\hat{y}_i = \bar{y} + \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}(x_i - \bar{x}) \tag{4}$$

When the formula is expressed in this way, it is easier to see how it may be updated. The SSR is a sum. The only difference in that sum from one configuration to the next is the addition or removal of one y_i and a corresponding x_i. This in turn only affects the $\bar{y}$ and $\bar{x}$ terms, which may easily be updated. The computation of the SSR is now straightforward.

This update process is much more efficient than the process of computing, over and over again, the SSR for a series of independent regressions. Given that this objective function runs literally hundreds of thousands of times per forest, and therefore tens of millions of times across the forecast period, it is worthwhile to optimize it for computational efficiency.

APPENDIX 3: SEARCHING THE PARAMETER SPACE

Overview. To efficiently search the parameter space for the penalty term, I drew on the Tree-structured Parzen Estimator (TPE) technique described by Bergstra et al. in the 2011 paper "Algorithms for Hyper-Parameter Optimization." For a full description of this process, see Section 4 of that paper, which lays out the necessary steps. For a less rigorous description, read on.

The quest to find optimal parameter values is referred to as "searching the parameter space;" in this case, the only parameter is the penalty term, so that space is one-dimensional. To search, the model trains itself with certain parameter values on a training set, scores its result against a test set, then runs with a different configuration of penalty values on the training set, scores that result against the test set, and so on. The goal is to find the configuration of penalty values that provide the optimal score.

This search may be performed in at least three ways: grid-wise, randomly, or in a Bayesian way. A grid-wise search would run the model or perform the desired task for *every possible combination* of parameter values (or, in the case of parameters which have continuous rather than discrete values, every possible combination of parameter values drawn at fixed intervals). In the case of my one-dimensional parameter space, the grid-wise search would generate one tree for each possible penalty value. In a one-dimensional space, the grid-wise search may not be very expensive, but with each added dimension, the search time has the potential to grow exponentially. On the plus side, this

type of search basically guarantees that the model will identify the optimal combination of parameter values.

A random search would randomly sample parameter values and hope to cover the parameter space in that way. It need not be as exhaustive as the grid-wise approach, which is an advantage, but it may also fail to identify a set of parameter values which is anywhere near optimal.

The Bayesian approach is the most efficient; as it searches, it updates the scope of the parameter space which it finds relevant to search; in other words, its focus narrows more and more as it continues to run, until it identifies a set of parameter values which is sufficiently optimal.

The TPE approach. The TPE approach I use is a Bayesian one, and operates as follows. First, the tree performs a wide, grid-wise search of the parameter space. Possible penalty values range from 0.70 to 0.99, by intervals of 0.005. Thus, there are 31 penalty values. The tree's first search uses the penalty values 0.70, 0.75, 0.80, 0.85, 0.90, and 0.95. It generates a score for each of these parameter values, based on how well they predict on the 11 observations which are withheld.

Once the model has six scores to correspond with the six different parameter values, it sorts the parameters into two groups: an *l* group and a *g* group. The *l* group consists of the 50% of parameter values which correspond with the best scores; the *g* group consists of the remaining 50%. In theory, then, after the first search, there are 3 parameter values in the *l* group and 3 parameter values in the *g* group. The *l* group is used

to create a distribution, with the 3 values which make up the group forming 3 peaks in the distribution; each peak resembles that of a normal distribution, with a certain standard deviation. The same is done for the *g* group.

Then, ten penalty terms are randomly selected from the *l* group's distribution. Each of the ten penalty terms is evaluated on the following basis. It is checked against the distribution of the *l* group, and its value in that distribution is found. This may be called its *l* value. Then it is checked against the distribution of the *g* group, and its value in that distribution is found; this is its *g* value. Its *l* value is divided by its *g* value, and the resulting value becomes its score. Each of the ten penalty terms is scored.

The score is supposed to represent the model's expectation for how well the model would perform with that penalty term. If the *l* distribution consists of the "good" penalty values, and the *g* distribution consists of the "bad" penalty values, then a value's score on the *l* distribution answers the question, "If I were to choose a good parameter value, how likely would I be to choose this particular value?" Likewise, the value's score on the *g* distribution answers the question, "If I were to choose a bad parameter value, how likely would I be to choose this particular value?" To divide the *l* score by the *g* score is to combine both answers into one; the higher the total score, the more likely the particular value is to be a good one.

So, ten values have been sampled from the *l* distribution, and they have been preliminarily evaluated as described. Of these ten, the three values with the highest score are actually tried out; the model runs three times, with these three parameter values, and

the model performance is scored for each of these three parameters, based on how well they handle the 11 observations which are withheld.

Now the model has run 9 times: the first 6 times for the broad, grid-wise search, and 3 additional times as described above. The model now has experience of 9 parameter values, and a corresponding score for each value. Accordingly, the model now sorts those 9 values, just as it had sorted the 6 values before: the 50% of parameter values which correspond with the best scores go to the *l* group, and the rest go to the *g* group. Now the *l* group and the *g* group each have four or five parameters in them, with corresponding performance scores.

Next, the model draws ten parameter values from the *l* distribution, scores them against both distributions as before, and selects the three values to actually run, just as before. This process occurs 5 total times, so that the model runs 21 times: 6 times initially, and then 3 more times on 5 additional occasions. This is already 10 times less than the grid-wise approach, and in practice the model will run even less than that: the parameter values which the model decides to run will often be repeated, and in this case the model will simply duplicate the result from the last time it was run with that parameter value. So, in practice, the model will likely run much less than 21 times.

Finally, the model has run with different parameter values and evaluated its performance on each occasion. Now it selects the parameter value which scored the best and generates a tree from the complete set of data which it is fed: both the erstwhile training set and the erstwhile test set.

APPENDIX 4: REAL RMSE DATA

Many of the tables I've given refer to RMSE values, relative to RMSEs produced by the naïve model. The tables below show the real RMSE values.

TABLE 1A

FIT AND FORECAST RESULTS ON SIMULATED AR(1) TIME SERIES

	Model Type				
	ARIMA	Base Tree	Hybrid Tree	Modified Tree	Naive
Fit RMSE	1.01629	1.029163	**0.9716035**	1.015209	1.025708
Forecast RMSE	1.052803	1.16591	1.0922	**1.036251**	1.063193

TABLE 2A

RMSE FORECAST RESULTS ON US MONTHLY INFLATION DATA, JANUARY 1999 – JANUARY 2020

Model Type				
Modified Forest	ARIMA	AR(1)	Base Forest	Naïve
0.002673159	0.002716793	0.002740606	0.002936135	0.003088432

TABLE 5A

RMSE FORECAST RESULTS ON DIFFERENT TYPES OF DATA

	Model Type				
US monthly inflation data: horizons	Modified Forest	ARIMA	AR(1)	Base Forest	Naïve
1 month	**0.0026732**	0.0027168	0.0027406	0.0029361	0.0030884
3 months	0.0029903	0.0030819	0.0031898	**0.0029699**	0.0041521
6 months	**0.0028885**	0.0030296	0.0031755	0.0028963	0.0041340
12 months	0.0030190	0.0030332	0.0032122	**0.0029328**	0.0042875
One-month horizon: other time series					
US unemployment, Jan 1990 – Jan 2000	0.1412112	**0.134985**	0.1426769	0.1404471	0.1402477
US 3-month Treasury rate, Jan 1985 – Jan 1995	**0.2084125**	0.222669	0.2091016	0.2344242	0.2398949
UK inflation rate, Jan 2015 – Jan 2020	0.0024801	**0.001302**	0.0028711	0.0024909	0.0040153

US monthly inflation: multivariate model	Modified Forest	VAR	Base Forest	Naïve
Predicted from Jan 1999 – Jan 2020	0.0026905	**0.0020979**	0.0028633	0.0030884