Decision Trees for Time-Series Forecasting

Article · January 2022

CITATIONS
14

READS
1,859

1 author:

Evangelos Spiliotis
National Technical University of Athens
109 PUBLICATIONS **4,757** CITATIONS

SEE PROFILE

Decision Trees for Time-Series Forecasting

EVANGELOS SPILIOTIS

PREVIEW *In this latest* Foresight *tutorial on forecasting methods, Evangelos Spiliotis takes us into the world of machine learning, introducing the decision-tree methods that have become a frequent and successful foundation of ML approaches to forecasting. He explains how these methods work and illustrates how they can be implemented for time-series forecasting.*

Previous tutorials have been compiled into the Foresight *Guidebook entitled* Forecasting Methods Tutorials*, available online at* **forecasters.org/foresight/bookstore/**.

INTRODUCTION

For many years, forecasters have employed rules to select the most appropriate forecasting model(s) for the time series of interest. Commonly associated with "expert" and "rule-based forecasting" (RBF) systems (Armstrong and colleagues, 2001), such rules have been built on forecasting expertise and domain knowledge that has emerged from work on particular data sets and applications. As a result, however, we cannot always consider them capable of being generalized to other applications.

The rise of machine learning (ML) has enabled new, data-driven approaches for determining rules in a more structured way. ML methods can automatically create explicit rules to minimize forecast error based on the metric the forecaster chooses to evaluate forecasting performance (e.g., mean squared error or mean absolute error). In addition, ML can produce forecasts that directly link time series of interest with relevant external information, such as promotions, holidays, and other special events. These ML methods are nonlinear in nature, making them more flexible than linear counterparts for modeling complex time-series patterns.

Currently the most popular of the rule-based ML methods for forecasting are based on *decision trees* (DTs). Recent forecasting competitions have shown that DT methods provide impressive accuracy in sales forecasting and other applications (Bojer and Meldgaard, 2021; Makridakis and colleagues, 2021a). This tutorial will explain how DTs work and how they can be implemented for time-series forecasting.

THE BASICS

DTs employ explanatory variables (called *features* in the ML literature) to predict a dependent variable (or *target*). The predictions are formed from a set of decision rules that determine how the data can be categorized.

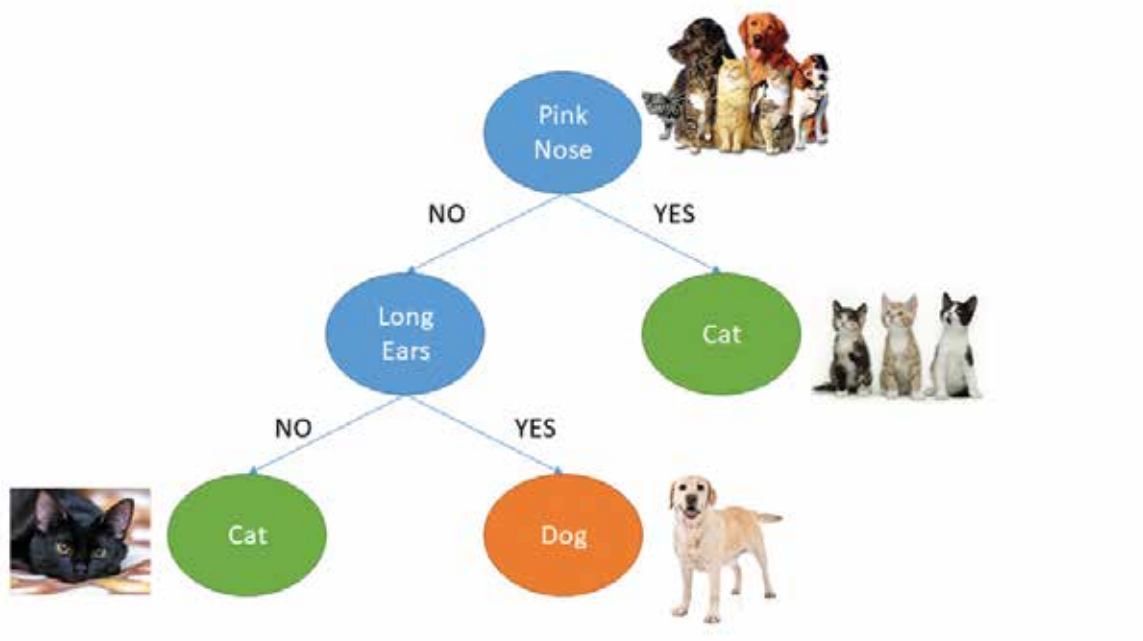
For example, consider a classification task where we are interested in distinguishing *cats* from *dogs*. The target variable being predicted is a label that can be either cat or dog. The features will describe the facial identifiers such as the color of the nose, the size of the ears, and so forth. Although some dogs have pink noses, this characteristic is more common among cats. Therefore, the existence of a *pink nose* could be the first rule considered for classifying the animal. Because some cats have black noses, however, the existence of *short ears* could be a secondary rule that would lead to recognition as a cat.

Figure 1 illustrates a classification tree for distinguishing dogs from cats.

We can think of many examples where this DT would fail to accurately distinguish a dog from a cat, so more features could be

Figure 1. Species Classification Tree

$$\text{Species} = \begin{cases} \text{Cat}, & \text{Nose color} = \text{pink} \ \text{OR} \ \text{Ears Size} \neq \text{Long} \\ \text{Dog}, & \text{Nose color} \neq \text{pink} \ \text{AND} \ \text{Ears Size} = \text{Long} \end{cases}$$



proposed to expand the set of rules. The type and number of rules employed are determined by an algorithm that is used for growing the tree, subject to the objectives and limitations defined by the user and, as with all data-driven methods, the data set available for training the model.

Roots, Branches, and Leaves

A DT consists of a root, branches, and leaves. The *root* is the node that starts the graph and is normally the variable that best splits the data – "best" with respect to the accuracy measure we are using. The root is typically split into two *branches* (a "binary" DT) based on a condition or rule that is automatically determined by the tree-building algorithm. Data points that fulfill the condition are assigned to one of the branches while the rest of the observations are assigned to the other branch.

Each of the initial branches can be further split into two new branches or can end in a terminal node called a *leaf*, which is not further split. The process is repeated until a predefined condition for terminating the growth of the DT is met. In Figure 1, the *pink nose* node on the top is the root of the tree, the *long ears* node is a branch, while the rest of the nodes are leaves. The values in these leaf nodes represent the outputs of the DT model—in this example, whether it's a dog or a cat.

Hyper-Parameters

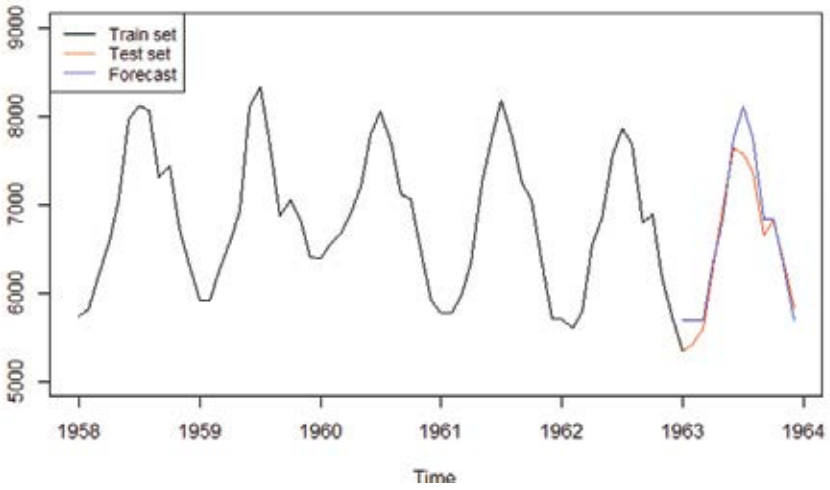
- The conditions for structuring the tree are called *hyper-parameters*. These include the maximum number of leaves (*max_leaves*);
- a maximum tree depth (*max_depth*) – how many splits a tree can make from the root to the leaf;
- a minimum number of observations required for a leaf to be created (*min_bucket*); or
- a minimum number of observations required in a node for a split to be attempted (*min_split*).

The tree in Figure 1 contains only two splits – essentially one for each feature – so we say it has a tree depth of two.

Classification Trees vs. Regression Trees

The principal aim of this tutorial is to

Figure 2. The "N2754" Monthly Series from the M3 Competition



illustrate how DTs can be used for time-series forecasting rather than classification, so here we shift our focus to *regression trees* (RTs). RTs can be considered a nonlinear alternative to the linear regression methods widely employed for time-series forecasting. Instead of providing a proper classification (e.g., cat or dog) at the leaf node, RT leaf nodes yield a predicted value, or forecast, for the observations that arrive at that leaf.

RTs are grown in a similar way to DTs; the difference lies in the fact that the target variable is not categorical but numerical. Whereas our DT for classification splits nodes to make the separation between cats and dogs as clean as possible, an RT splits nodes such that *child nodes* (nodes extending from another node) contain observations that are as similar as possible on this numerical target value. To put it another way, splits are determined by looking at all features and deciding which most cleanly separate the observations in terms of the target variable.

Technically, RTs provide a piecewise approximation to the single continuous function in standard regression, a function that is used to estimate the impact of explanatory variables on the target. A forecast from an RT – essentially the value of the leaf that will be used as a forecast – will be equal to the mean value of the historical observations (training set) in this leaf. Since the leaf we end up in is determined by the features, we can think of the forecast as a *local mean* of those training observations whose features are similar to those of the observation being forecast.

Since DTs are built both for classification and forecasting, they are often referred to with the acronym CART, for Classification and Regression Tree.

REGRESSION TREES FOR TIME SERIES

Figure 2 displays one of the monthly time series used in the M3 forecasting competition (Makridakis and Hibon, 2000). You can see that the series lacks significant trend, but is seasonal, with peaks in the warm-weather months. I'll use this example to show how to forecast the final 12 months of the series based on the earlier historical observations. All the data up to the final 12 months form what is termed the *training set*, and the final 12 months the test set. In Figure 2, the training set is displayed in black while the test data are shown in red. The ultimate forecasts of the RT are in blue.

To generate these forecasts, you need to "engineer features"; that is, create input variables that help explain the variation in the target time series. For simplicity, let's consider only three features:

- A rolling mean of order 12 (*RM12*) – which is the average of the 12 observations just prior to the month being predicted.

- A categorical (i.e., dummy) variable that indicates the month being predicted (*Month*); for example Month = 1 denotes a January.
- The prior year's value of the same month being predicted (*Lag12*).

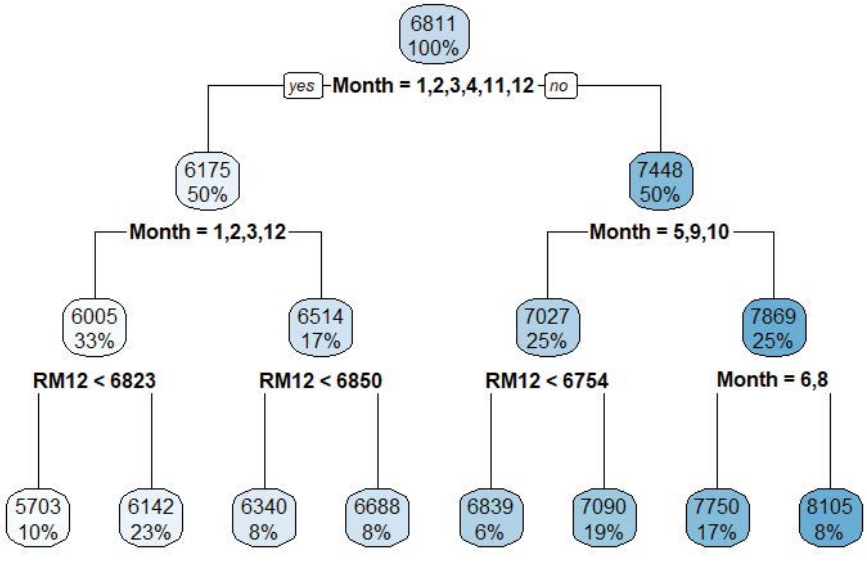
The first feature is designed to represent the level of the series, while the second and third account for seasonality. Note that the M3 competition did not introduce any exogenous explanatory variables (e.g. a measure of economic activity); in principle, however, these could also have been input as features.

Since the data available for training are somewhat limited, we will train the RT by setting *min_bucket* to 2 – meaning that at least two observations are needed to create a leaf – and *min_split* to 5, requiring at least five observations for a node to be split. Lower values for these hyper-parameters could lead to a tree with an excessive number of branches. To further simplify the RT, we'll set *max_leaves* to 8 and *max_depth* to 3. The tree grown based on these hyper-parameters and the training data is shown in **Figure 3**. The *rpart*() function from the rpart package for R (***https://www.r-project.org/about.html***) was used to build the tree, while the *rpart.plot*() function from the package of the same name was used for visualizing it.

In Figure 3, each node embeds a rule for splitting the training data, a rule to move to the left branch if the condition is met (yes) or to the right otherwise (no). The root node (at the top) is the dummy variable for month. This feature was chosen by the algorithm because it provides the best-fitting tree (based on the error metric chosen to evaluate in-sample forecasts, in this case the mean squared error). If the month is one of months 1-4 (January to April) or 11-12 (November to December), we are directed to the left branch; months 5 (May) to 10 (October) move us to the branch on the right.

Within each node we see the mean value of the training observations that fall in the node – a mean value of 6811 for the root node at the top – while the percentage figure (100% in the root node) is the percentage of all training observations used to form that average. For the first branch node on the right, we see that 50% of the observations available in the training set were averaged to find the mean value of 7448. The blue shading reflects the value of these node averages, with deeper blue indicating higher values.

Figure 3. The RT Grown for Forecasting the "N2754" Series



The leaf mean values become the forecasts (in Figure 2) to be evaluated across the test data. Consequently, if the month being forecast was month 6 (June), the first rule in the root would move us to the right branch, the following rule to the right again, and the next rule to the left leaf, thus resulting in a prediction of 7750. As another example, if we are forecasting month 3 (March), the first rule moves us to the left branch and again to the left from the second split, and the final split to one of two leaf nodes depending upon whether the rolling mean is less than or greater than 6823.

The RT of Figure 3 implies that the forecasts for the summer months – months 6, 7, and 8 – are based only upon the *month* dummy variable feature. For the other months, the forecasts are also based on the level of the series (*RM12*). Interestingly, *Lag12* is not considered by any of the branches, although available as information, revealing that the categorical value itself is sufficient for modeling the seasonality of the series.

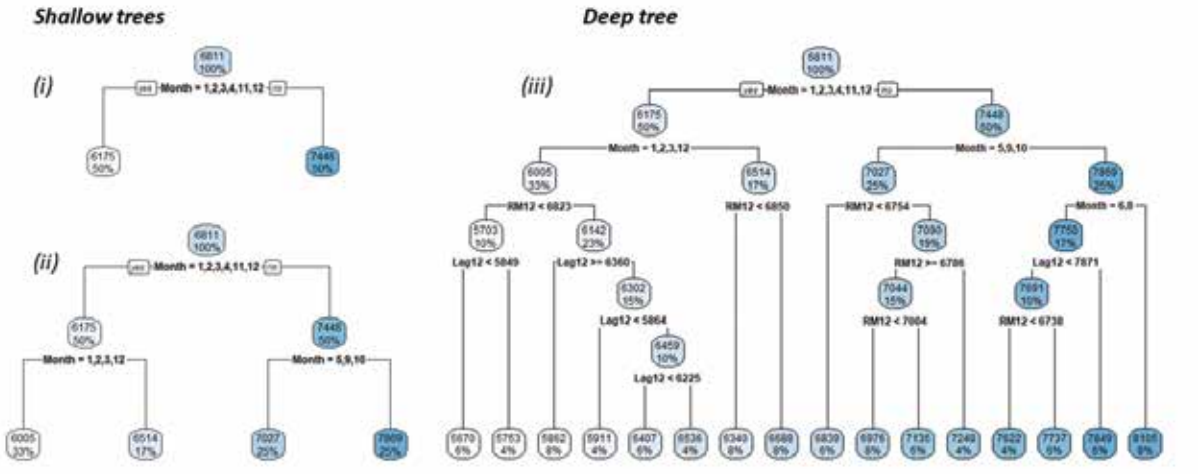
It also illustrates the ability of the tree model to selectively use features. We may provide an RT with as many features as we wish. But whether the tree will decide to use them depends on the ability of each feature to provide a split that reduces forecast error. The hyper-parameters chosen, such as tree depths and number of leaves, can force the tree to consider fewer features. In this example, had we decided to refit the tree with the same hyper-parameters but without *Lag12* as a feature, the resulting tree would be identical.

Referring back to Figure 2, the final forecasts appear to be quite accurate, with the mean absolute percentage error (MAPE) for the test set being under 3%, a remarkable result given the simplicity of the rules constructed and the limited features considered.

Accuracy improvements are still possible through incorporation of new features or an expanded tree structure that allows the RT to analyze the train set in more detail. Trees that are too shallow (few splits and leaves) may underfit the data and fail to model key data relationships, while trees that are too deep (multiple splits and leaves) can overfit the data, making the forecasts sensitive to extreme values and erroneously modeling randomness. Excessively deep trees should be pruned as I discuss below.

Figure 4 illustrates alternative RTs for our target variable. The two trees on the left (*i* – incidentally, a tree that has a depth of 1 like this one is frequently called a *stump* – and *ii*) can be considered too shallow in the sense that their depth (1 and 2, respectively) does not allow for consideration of all features provided as input. In contrast, the tree on the right

Figure 4. Shallow (left) and Deep (right) RTs



(*iii* with a depth of 6) is possibly too deep in the sense that it constructs overly specialized rules that capture even minor variations of the series level (*RM12*) and seasonal behavior (*Lag12*). Note that apart from the tree depth, the rest of the hyper-parameters used for building the RTs are the same as before.

Figure 5 adds the forecasts produced by these shallow and deep trees to Figure 2. Both shallow trees *i* and *ii* of Figure 4 fail to capture seasonality in detail, resulting in less accurate forecasts with MAPEs of 7% and 5.3%, higher than the 3% MAPE of the RT in Figure 2. The deep RT has a MAPE of 2.7%, only slightly below the 3% for the illustrative tree despite much greater complexity.

Proper selection of hyper-parameters can optimize the growth of the tree, maximizing post-sample accuracy from the features selected. If *max leaves* and *max depth* are set to infinity, *min_bucket* to 1 and *min_split* to 2, the RT will grow to its full potential, minimizing in-sample forecast error. However, such a tree may lead to overfitting and hence poor post-sample forecasting performance. In practice, we strive for shallower RTs, often using the default selections: max_leaves=16, max_depth=7, min_bucket=5, and min_split=20.

The Algorithm

Given that RTs are a rule-based forecasting method, their training requires decisions on

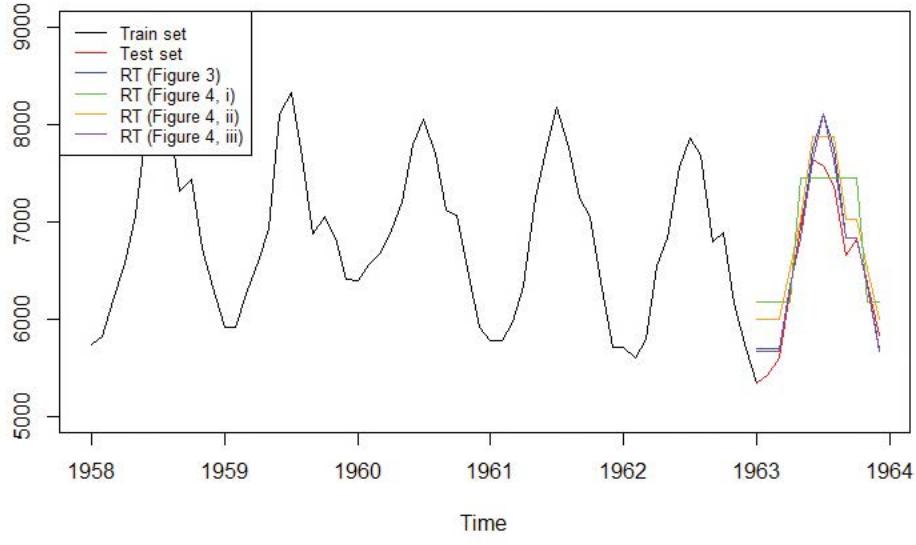
- which feature to choose for splitting the root and each branch;
- what rule (cut-point value) to use on each selected feature for performing the splitting; and
- when to stop splitting.

Fortunately, the forecaster need not be burdened with these decisions; rather they are automatically made by the algorithm utilized to train the model.

The most popular algorithm for training a tree is a "greedy" one in which the feature selected for each split and the condition used for splitting are determined so that the in-sample error is minimized. Assuming we have N features, the algorithm will consider N candidate splits, one for each feature. Each one will use the optimal cut-point for the feature and then select the split that maximizes in-sample accuracy.

This greedy algorithm does not guarantee that the tree created will provide the best possible fit to the training data (globally optimal), but only that it offers the best possible split at each step in the tree. This is because splitting is performed sequentially and, as a result, the best feature for performing each split, along with the respective optimal cut-point, are decided separately at each step without taking into consideration their impact on the final fit of the model. The reasoning behind this step-wise optimization is linked to computational requirements; it

Figure 5. Forecasting Performance Comparison of Shallow and Deep RTs



is practically impossible to test all possible combinations of splits beforehand to decide which combination provides the best global fit, especially when numerous features are available for training.

Moreover, as is the case with all optimization algorithms that build on in-sample evaluations, this algorithm does not ensure that the tree created will maximize *out-of-sample* accuracy.

The criterion typically used for training RTs is the sum of the squared errors, but other loss functions, such as the sum of the absolute errors, can be used instead to better match the requirements of the forecasting task at hand.

Pruning the Tree

As noted, RTs can grow out of control, using up all the data available in training and leading to overfitting, thus deteriorating forecasting performance. As a result, it is valuable to be able to set a condition to terminate further splitting of the branches.

This can be done in various ways, one of which is specifying the maximum tree depth. But because the hyper-parameters for training the model are defined a priori, we have no guarantee they will be optimally set; that is, set to avoid over- or underfitting. Depending on the number of the features available, the length of the time series, the complexity of the relationships between the features and the target variables, and the patterns the target variable displays, different hyper-parameter values may be warranted and so alternative sets of hyper-parameters should be tested.

A good practice is to allow the tree to grow to its full extent (or close to it) and then apply pruning to remove the branches and leaves that produce only minor improvements in accuracy compared to the complexity they introduce. Since it is possible for a split of minor accuracy gains to be followed by a split with major accuracy improvements, this practice has the further advantage of retaining leaves that reduce in-sample error no matter how helpful their branches initially were. As a result, splitting rules that refer to a limited sample of observations, such as unusual events and outliers, are effectively removed, while rules that significantly improve forecasting accuracy are preserved. In the example of Figure 4, the leaves of the deep tree that consider the *Lag12* feature can be pruned since inclusion of this feature contributes insignificantly to the accuracy improvement of the model.

The algorithm used for training a single RT has several limitations: even if the RT is pruned and trimmed, the model may be overly sensitive to randomness and extreme values when the number of features is large relative to the number of data points available for training.

In order to desensitize the results to randomness and extreme values, but also potentially improve accuracy further, methods have been proposed to create ensembles (combinations) of multiple RTs. Two widely used methods are Random Forests (RFs) and Gradient Boosted (GB) trees.

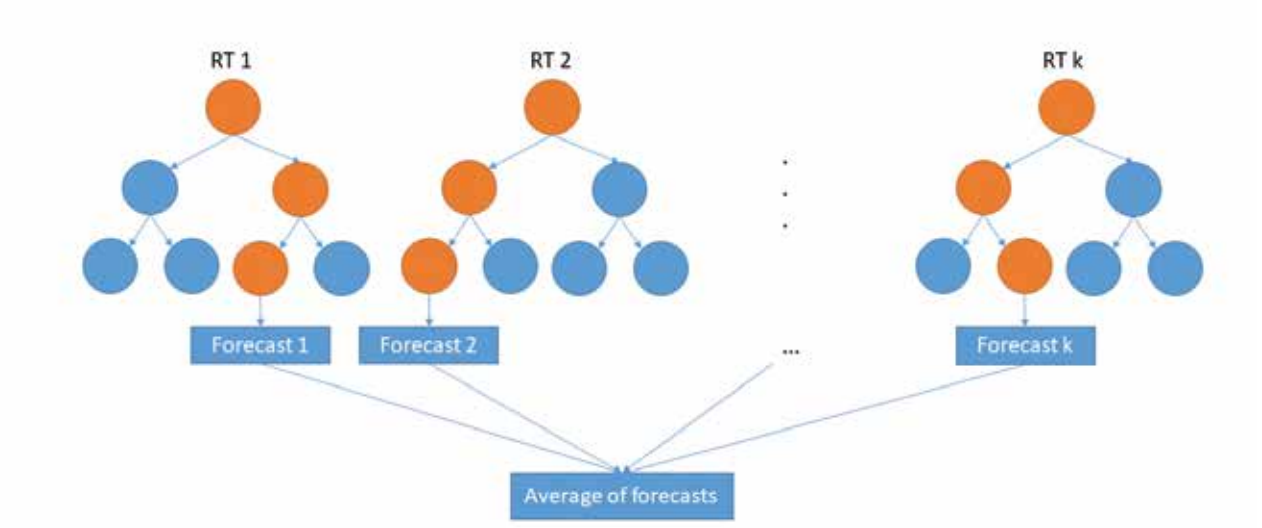
RANDOM FORESTS

Combining forecasts has long been shown to improve forecast accuracy. Because each model whose forecasts are to be combined may consider different variables and make different assumptions about data relationships and distributions, averaging the individual forecasts broadens the information embedded and may offset individual model biases as well.

This same idea can be applied to RT models through a technique known as *bagging* or *bootstrap aggregating*. The original data set is randomly sampled so that multiple new data sets of the same size as the original are created. Because the sampling is usually done with replacement, some of the data sets created will overlap, some data could be included several times within each data set, and other data may wind up not being used at all. In addition, each data set can include a randomly selected subset of features.

An individual RT is then trained on each set of data. Since each tree has access to different information (both in terms of

Figure 6. A Random Forest of k Individual RTs



features and observations), the forecasts are effectively diversified and an average of the forecasts from the individual tree models will be less sensitive to extreme values and can neutralize the potential biases of individual models.

A commonly used algorithm that exploits this method of ensembling is called a "random forest" (Breiman, 2001). **Figure 6** illustrates its structure.

When using RFs, the forecaster must specify

- The total number of trees (*num_trees*) whose forecasts will be averaged. The typical range is between 30 and 500, with higher numbers selected for more complex models (more features, more rules, and more forecast combinations).
- The number of features (*num_features*) that will be randomly considered by each tree (typically being 1/3 of the features originally available).
- The values of the hyper-parameters controlling the growth of the individual trees.

Computation time and cost become a factor with the number of individual trees to be trained, although training an RF is easily parallelized, since each tree can be grown separately. Still, a common way to control cost is to create relatively shallow trees. Such trees are often called *weak learners* because their shallow depth limits its learning capacity and specialization.

As an illustration, I've applied the RF process to the example time series from the M3 competition. I have set *num_trees*=3 of *max_depth*=2 (so these are weak learners), each using *num_features*=2 of the features originally available. Using two features per RT is preferable in this example to using a single explanatory variable, since features that account for seasonality are critical for producing reasonable forecasts. On the other hand, using three features per RT is not recommended since, if we do so, all trees created will have access to the same information in terms of features and, as a result, we lose some of the diversity for the forecasts being combined. Feature sampling is a key asset of the RF method and should be exploited. For more detail on how RFs can be set up, see Fawagreh and colleagues (2014).

Figure 7 presents an RF of three regression trees. RT 1, with inputs *Lag12* and *RM12*, selected only Lag12 as a feature to account for seasonality. RT 3, with inputs *Lag12* and *Month*, exploited both seasonal variables for producing forecasts. Lastly, RT 2 used the Month feature as a primary predictor but also specialized its forecasts based on the level of the series (*RM12*). Because each RT has access to different information and focuses on different data

Figure 7. Illustrative RF of the M3 Series

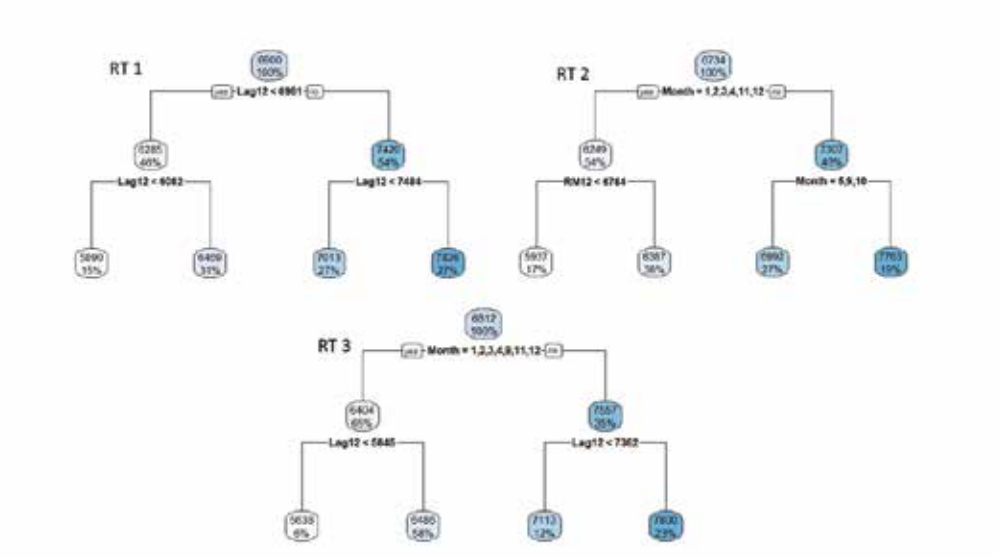
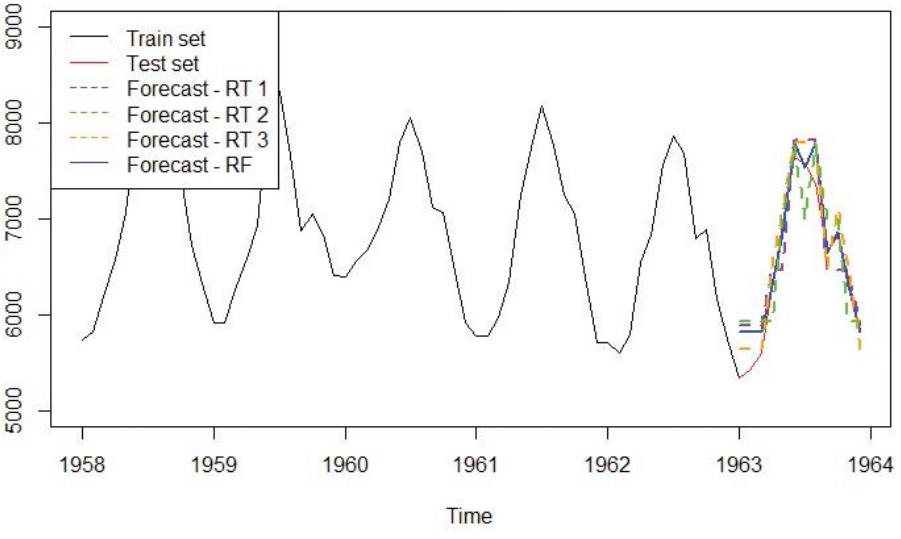


Figure 8. Forecasts by the RF and Its Three Components



relationships, we arrive at different forecasts that, when combined, can improve overall accuracy.

Figure 8 shows the forecasts produced by the three trees individually along with their ensemble. The dashed lines correspond to the individual forecasts of the RTs, and the continuous blue one to their ensemble. As seen, the forecasts from the RTs are very similar, yet provide different estimates for each month depending on the data sample and the features provided for training.

Overall, it appears that the ensemble provides more accurate forecasts of the test data. In particular, although RT 1, RT 2, and RT 3 report MAPE values of 4.7%, 5.3%, and 3.2% respectively, their ensemble has an error of 2.6%, being slightly more accurate than the single RTs trained in Figures 3 and 4.

GRADIENT BOOSTING

Gradient boosting (GB), devised by Jerome Friedman (2002), is similar to RF in that it is an ensemble method built upon multiple trees. It is commonly

applied to combine *weak learners* so as to create a more powerful forecasting model (*strong learner*). Like RF, GB averages forecasts across individual trees but first generates the trees sequentially, so that each new tree improves on the forecast accuracy of the existing trees.

As a result, GB results in more specialized solutions than RF; that is, when both methods use the same number of weak learners (e.g. 2 stumps each), the forecasts from GB can consider more features and more complex data relationships. For example, an RF will average the forecasts of two trees, each of depth 1, while GB will provide a single forecast, resulting from a tree of depth 2.

Leaf-Wise Growth vs. Level-Wise Growth

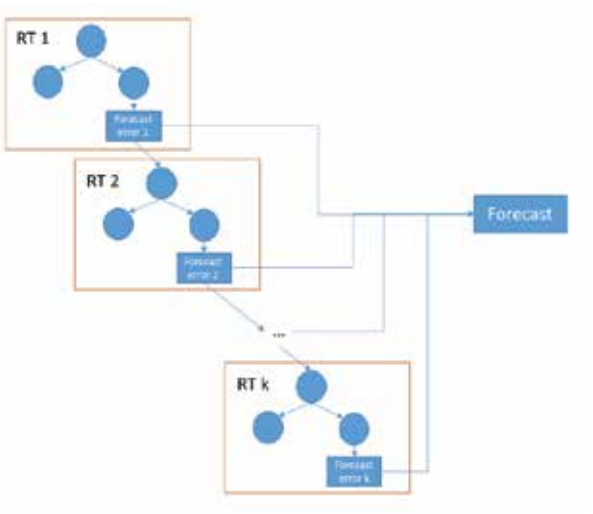
GB algorithms follow one of two approaches: *leaf-wise growth* or *level-wise growth*.

Leaf-wise growth (also called best *first growth*) begins with the training of a single tree. The GB algorithm evaluates the accuracy of this tree and identifies which of its leaves is most responsible for the overall forecast error. It then focuses on the problematic leaf and expands its solution by modeling the data points in this leaf with another tree. More specifically, the second tree employs the data points the problematic leaf failed to forecast precisely in the previous step. The algorithm grows the tree at each step in the direction of the leaf whose improvement maximizes global accuracy gains.

Rather than trying to predict the target variable directly, each new tree attempts to model the residual error of the forecasts from the previous trees. In this sense, GB is a "divide and conquer" strategy, decomposing the data-generation process of the target variable into multiple simpler subprocesses, thus making estimation more manageable. In contrast to single RTs, the existing trees are expanded by other trees that may include multiple branches and leaves instead of being split into two leaves at each step.

At each step, the GB forecasts will be a weighted average of the forecasts

Figure 9. Leaf-Wise Gradient Boosting



provided by the initial tree and the following tree introduced, a process called *boosting*. Boosting continues with each new tree designed to improve the previous model. The training process can be terminated in a manner similar to pruning: if accuracy gains are minor compared to the complexity introduced when expanding the existing tree, no further RT is added.

Figure 9 illustrates the leaf-wise growth approach to boosting. As seen, the final forecast is the weighted average of the forecasts provided by k RTs, each being responsible for modeling the forecast error of the previous tree.

Boosting can also be applied in a *level-wise* (or *depth first*) fashion; that is, *all* the leaves of the previous level are boosted without prioritizing individual leaves. The level-wise approach can produce forecasts that are less prone to overfitting and work better with small data sets. On the other hand, leaf-wise growth provides a more specialized approach, yielding forecasts faster than the level-wise approach from the same number of leaves.

The main conditions the forecaster must specify to implement GB are the total number of trees that will be used to correct the residual errors and the *learning rate* (or shrinkage factor), which is how quickly the error should be corrected from each tree to the next. The learning

rate controls the magnitude of change that each added tree contributes to the final forecasts. Lower rates suggest slower learning and more precise adjustments at each step, but also higher computational cost (more trees may be required to effectively train the model). The training process of the individual trees remains essentially as before.

Variants of Gradient Boosting

There are several variants of the GB method in common use. These share the same principles but differ in the way they grow or aggregate the individual trees.

- **LightGBM** (Light Gradient Boosting Machine) recently earned top positions in the M5 accuracy competition (Makridakis and colleagues, 2021a), and has been praised in the forecasting community for its ability to efficiently handle large data sets with less memory and faster training. LightGBM was used by most of the winning teams in the M5, showcasing the method's ability to forecast multiple time series of diverse patterns and features. The method employs leaf-wise tree growth and balances accuracy and speed through a random-sampling technique that selects the optimal split in each node. For more details on LightGBM and how it was used by the winner of the M5 accuracy competition, see In and Jung (2021).
- **XGBoost** (Extreme GB) puts bounds on the training process so as to restrict overfitting. The method also accelerates training by exploiting parallel computing and reduces memory requirements through improved data structures. Traditionally, XGBoost employs level-wise tree growth, but also supports leaf-wise growth. In contrast to the greedy algorithm behind conventional GB, XGBoost uses a histogram-based algorithm that buckets continuous features into discrete bins and uses these bins to select the optimal split in each node, thus reducing computational cost (Chen and Guestrin, 2016).
- **AdaBoost** (Adaptive Boosting), like other GB techniques, learns from prior mistakes. It sequentially grows multiple weak learners and uses combination weights at each modeling step to "punish" inaccurate forecasts (Cao and colleagues, 2013). The weights are continuously updated till the global forecast error is minimized. The main distinction from the other forms of gradient boosting is that AdaBoost adjusts the combination weights based on those observations that are inaccurately predicted, rather than the residuals from the forecasts produced for these observations. Moreover, AdaBoost readjusts the weights in each iteration, whereas GB adjusts the predictions of the previous steps based on the residuals.

MODEL VALIDATION AND FEATURE ENGINEERING

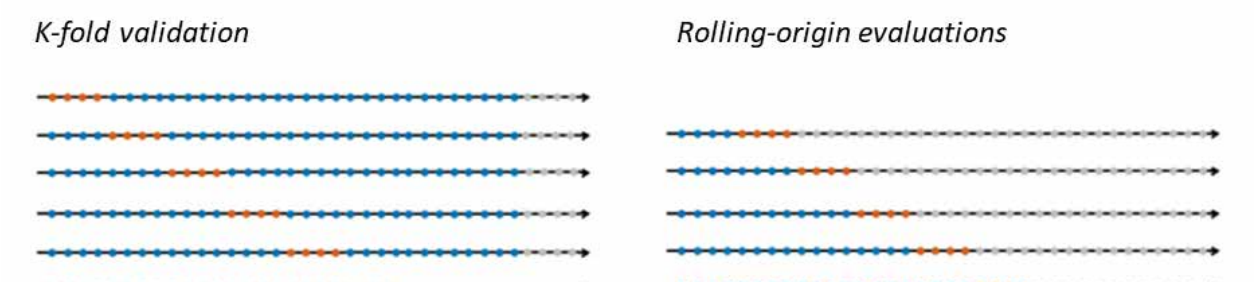
Compared to other ML methods, RT methods require the specification of a relatively small set of hyper-parameters. Nevertheless, these must still be optimized for each forecasting task so as to ensure that post-sample accuracy is maximized. This can be done by employing validation techniques widely used in the forecasting and ML community.

Cross-validation

A robust way to optimize the hyper-parameters of an RT to ensure maximum out-of-sample accuracy is to employ some form of *cross-validation*, such as *k-fold* validation or *rolling origin* evaluations (Bergmeir and colleagues, 2018).

In *k-fold* validation, the data points are randomly shuffled and split into k groups (typically 10). For each group, the other *k-1* groups of data are used for training a model of particular hyper-parameters while the residual group is the test set. Forecasting accuracy metrics are calculated across the *k* holdout groups. By repeating this process for different sets of hyper-parameters, we can identify the most accurate model for the forecasting task at hand. However, there are problems with k-fold validation on time-series data because some of the holdout test data will end up before training data, so that the procedure does not completely reflect forecasting into the future; see ***https://towards-datascience.com/dont-use-k-fold-validation-for-time-series-forecasting-30b724aaea64.***

Figure 10. Training (blue) and Validation (orange) Sets



For time series, then, rolling origin evaluations are a better form of cross-validation. Only the more current observations are held out of the training process to serve as test (validation) data, as shown in **Figure 10**. Tashman (2000) discusses the various design options, including how to split the data between training and validation.

While it would be onerous to evaluate all possible combinations of hyper-parameter values, we can utilize *Bayesian optimization* to approximate the optimal values at reduced computational cost. This type of optimization algorithm employs greedy sequential methods that limit the search space dynamically based on the expected improvements (Bergstra and colleagues, 2011).

Feature Engineering

The forecasts produced by RT methods are derived from the features created to train the tree models, so forecast accuracy will be only as good as the effectiveness of the features in explaining the variation in the target time series. As a result, selecting and potentially transforming useful features is critical, perhaps even more so than optimizing the hyper-parameters. This process, typically called *feature engineering*, involves the identification of ready-to-use explanatory variables that describe the attributes and characteristics of the data as well as the extraction of new variables, such as seasonal terms that capture the properties of the time series.

In many cases, features emerge naturally based on the forecasting task. In retail-sales forecasting, for example, features that drive sales, such as promotions, advertising, and product prices, should be obvious choices. In addition, dummy (categorical) variables for the type and location of the product (store, ZIP/postal code, district, city, municipality, or country) are commonly included. Weather forecasts can also be taken into consideration. Dummy variables can also be introduced to capture special events, holidays, structural changes, and anomalies (e.g., COVID-19 effects).

To capture seasonal variation, we typically use dummy variables to identify the year, month, week, day, or hour that sales take place, or lagged features such as "sales of same day last week/year." Alternatively, one may consider Fourier-type models that use harmonics as features, an approach that will allow an RT to produce smoother and, therefore, reasonable forecasts.

Capturing trends in the data is increasingly important as the forecast horizon lengthens. RT forecasts, however, are constants that represent the mean value of the observations in each leaf; they cannot naturally account for trends unless special data preprocessing (e.g. scaling) is applied. For example, the forecaster may transform a trended target variable into percentage changes, as done by Smyl (2020) in the M4 competition. More advanced techniques apply regression on the observations contained in each leaf instead of computing their mean, although they are less common in mainstream software implementations. Still

other options are to express a trended target variable as a running counter of the time-series observations, a cumulative metric that dampens trends, or to externally de-trend the data by assuming a particular type of trend (e.g. linear, exponential, or polynomial).

Accounting for the level of the data is also critical, especially when forecasting series that experience notable fluctuations across time or level shifts. To capture these shifts, summary statistics (e.g. min, max, mean, median, or standard deviation) of the data can be included as features. The most appropriate look-back window (e.g. last week or month) for computing these summary statistics is specific to the forecasting task.

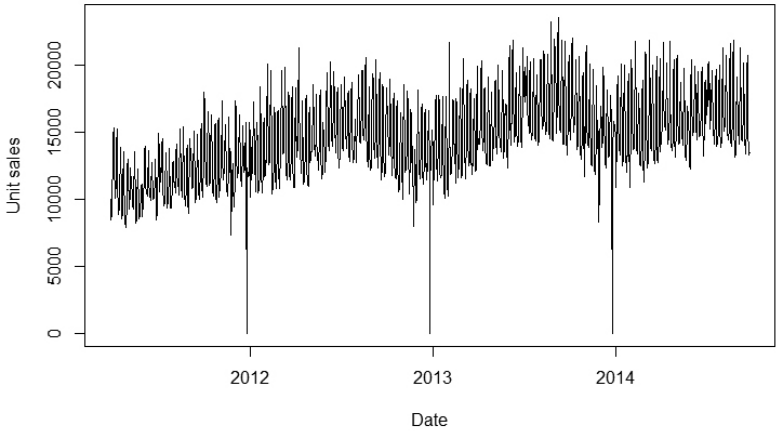
As yet, we lack a fully automated process for identifying and engineering features. Modeling expertise and domain knowledge are needed to conceptualize and create appropriate features, as well as to test and understand their impact, which adds a potentially time-consuming component to the development of RTs and other ML models.

Cross-learning

Cross-learning can be used to mitigate the data requirement issues of tree methods and other ML models, while also improving model performance. Instead of separately training a tree for each time series (*series-by-series training*), we can train a single RT to simultaneously predict multiple series. Doing so allows the model to learn from larger data sets and to capture relationships observed across series of similar characteristics.

Cross-learning has proven particularly effective for forecasting data sets that include relatively short or hierarchical series (e.g. product sales of a retail firm) because it captures the patterns in related time series that could not be seen in overly short series. Walmart, which requires sales forecasts for millions of products on a daily basis, has been using RT methods in a cross-learning fashion to produce cost-efficient but sufficiently accurate forecasts (Seaman and Bowman, 2021). Similarly, cross-learning was an important component of winning submissions in the most recent M competitions for both its forecast accuracy and computational efficiency compared to its series-by-series counterpart (Semenoglou and colleagues, 2021).

Figure 11. Unit Sales in California (M5 data set)



PERFORMANCE IN THE M5 COMPETITION

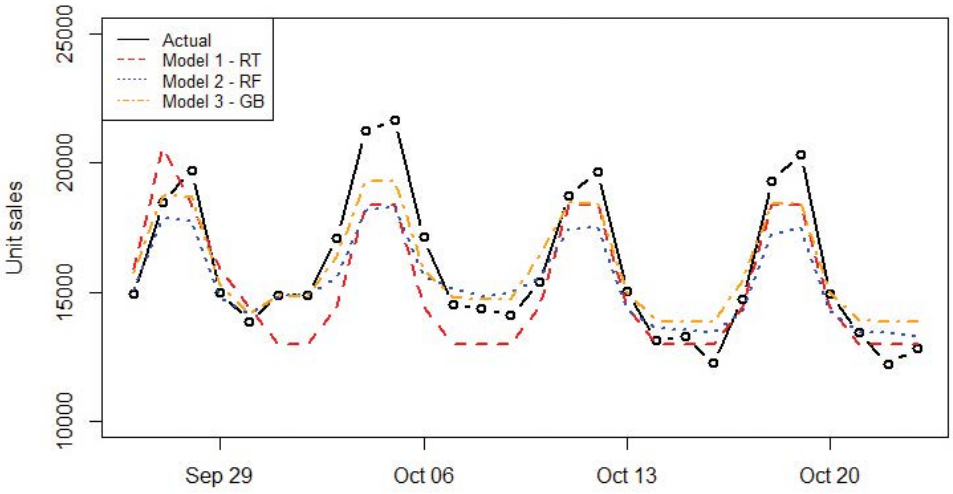
In order to compare the single regression tree, random forest, and gradient-boosting methods, I considered a time series from the M5 competition that was based on a full hierarchy of retail-sales data provided by the Walmart Corporation. The data set contained 30,490 time series of sales of 3,490 products sold by Walmart in 10 of its stores, located in three different states.

Figure 11 plots the aggregate daily sales reported for the state of California. You can see that the series is characterized by trend and strong seasonal patterns, both at the daily and monthly levels. The M5 data set also includes information about Supplemental Nutrition Assistance Program (SNAP) activities that affect food products sales (by about 15%, according to Makridakis and colleagues, 2021b).

I used the first 1,281 observations (3.5 years * 365 days of sales) to train the RT methods introduced earlier, and generated forecasts from them for the following 28 days.

Model 1: A single RT whose features are the month and day of the week (dummy

Figure 12. Forecasting Performance Comparison of RT, RF, and GB for the Walmart Series



variables), the existence of SNAP activities (binary variable that may be either 0 when no activity takes place or 1 otherwise), and the average daily sales reported in the previous 28 days (rolling mean). I implemented the model using the *rpart*() function in the R program.

Model 2: A random forest model of 100 trees created from the same features as Model 1. The model was implemented using the *randomForest*() function in the package of the same name for R.

Model 3: A GB model also based on these same features as Model 1 and again consisting of 100 trees. The model was implemented using the *gbm*() function for R, again taken from the package of the same name, thus being a standard GB model that grows in a level-wise fashion.

Figure 12 presents the results.

Model 1 appears as the least accurate (7.5% MAPE), which is to be expected since its forecasts are produced by a single RT. The RF, Model 2, by exploiting the benefits of bagging and ensembling, reduces the mean absolute error to 5.9%, an improvement largely driven by the model's smaller underestimate of sales at the beginning of October compared to Model 1. Lastly, Model 3, which constructs the most specialized rules to produce forecasts, provides the most accurate results with a MAPE down to 5%. This additional improvement can be attributed to the more accurate predictions Model 3 makes for the weekend days, although all three models still underestimate weekend sales.

DTs vs. NNs

DTs are powerful forecasting solutions that have become an important presence in the data scientist's tool kit. Compared to the other principal type of ML model – neural networks (*NNs*) – tree models are interpretable because their rules can be visualized, thus giving decision makers the ability to understand the factors that drive the forecasts. Note, however, that this property is more applicable to single DTs: ensembles of multiple DTs, as with random forests, obfuscate the exact rules and their impacts.

Tree models are also less computationally intensive than NNs, because they tend to require less parameter optimization, fewer data points for training, and less data preprocessing such as scaling and normalization. Because each split is made looking at one feature at a time, the features don't usually require normalization to be placed on the same scale. In contrast, normalization is a standard requirement in NNs and many statistical models as well.

Tree methods match the capabilities of NNs to exploit numerical, categorical, and binary features, to selectively include features, and process multiple features in a nonlinear way. However, their selectivity works more directly: instead of shrinking the weights assigned to insignificant

features as done in NNs, RT rules can simply exclude insignificant features.

All forecasting methods have drawbacks, and tree methods are no exception. In the absence of limitations imposed for controlling their growth or mechanisms for boosting or bagging, these methods tend to overfit the data. On the other hand, restrictions can be overdone, resulting in overly shallow trees which do not adequately explain the behavior of the time series.

Because tree methods provide piecewise approximations, their forecasts can be choppy, rather than smooth or continuous, thus becoming problematic when used for production planning or inventory management. In addition, their performance can be highly sensitive to small changes in the training data. While the training of tree models requires fewer data points than NNs, the data requirements are still significantly larger than those for standard time-series forecasting methods, especially exponential smoothing.

REFERENCES

Armstrong, J.S., Adya, M. & Collopy, F. (2001). Rule-Based Forecasting: Using Judgment in Time-Series Extrapolation, *Principles of Forecasting: A Handbook for Researchers and Practitioners* (Ed. J. Scott Armstrong). Kluwer, 2001. ***https://repository.upenn.edu/marketing_papers/149/***

Bergmeir, C., Hyndman, R.J. & Koo, B. (2018). A Note on the Validity of Cross-Validation for Evaluating Autoregressive Time Series Prediction, *Computational Statistics & Data Analysis*, 120, 70-83.

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011). Algorithms for Hyper-Parameter Optimization, *Advances in Neural Information Processing Systems*, 24. Curran Associates, Inc.

Bojer, C.S. & Meldgaard, J.P. (2021). Kaggle Forecasting Competitions: An Overlooked Learning Opportunity, *International Journal of Forecasting*, 37(2), 587-603.

Breiman, L. (2001). Random Forests, *Machine Learning*, 45(1), 5-32.

Cao, Y., Miao, Q.G., Liu, J.C. & Gao, L. (2013). Advance and Prospects of AdaBoost Algorithm, *Acta Automatica Sinica*, 39(6), 745-758.

Chen, T. & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM.

Fawagreh, K., Gaber, M.M. & Elyan, E. (2014). Random Forests: From Early Developments to Recent Advancements, *Systems Science & Control Engineering*, 2(1), 602-609.

Friedman, J.H. (2002). Stochastic Gradient Boosting, *Computational Statistics & Data Analysis*, 38(4), 367-378C.

In, Y. & Jung, J.Y. (2021). Simple Averaging of Direct and Recursive Forecasts via Partial Pooling Using Machine Learning, *International Journal of Forecasting*.

Makridakis, S. & Hibon, M. (2000). The M3-Competition: Results, Conclusions and Implications, *International Journal of Forecasting*, 16(4), 451-476.

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2021a). The M5 Accuracy Competition: Results, Findings and Conclusions, *International Journal of Forecasting* (forthcoming).

Makridakis, S., Spiliotis, E. & Assimakopoulos, V. (2021b). The M5 Competition: Background, Organization, and Implementation, *International Journal of Forecasting* (forthcoming).

Seaman, B. & Bowman, J. (2021). Applicability of the M5 to Forecasting at Walmart, *International Journal of Forecasting* (forthcoming).

Semenoglou, A.A., Spiliotis, E., Makridakis, S. & Assimakopoulos, V. (2021). Investigating the Accuracy of Cross-learning Time Series Forecasting Methods, *International Journal of Forecasting*, 37(3), 1072-1084.

Smyl, S. (2020). A Hybrid Method of Exponential Smoothing and Recurrent Neural Networks for Time Series Forecasting, *International Journal of Forecasting*, 36(1), 75-85.

Tashman, L.J. (2000). Out-of-sample Tests of Forecasting Accuracy: An Analysis and Review, *International Journal of Forecasting*, 16(4), 437-450.

Evangelos Spiliotis is a Research Fellow at the Forecasting & Strategy Unit, National Technical University of Athens, where he also serves as Coordinator. His research interests include time-series forecasting, decision support systems, machine learning, and optimization. He is the co-organizer of the M4 and M5 forecasting competitions.

spiliotis@fsu.gr