Overview of Decision Tree Modeling Methods

Stuart McMenamin, Ph.D., Itron, Inc.

WHITE PAPER

Decision Trees methods date back to the 1960's and are an important part of the data-science playbook. The early work dealt mostly with classification problems (Decision Tree Classifiers or Classification Trees). But the methods can also be used to model continuous variable outcomes (Decision Tree Regressors or Regression Trees). There are several variations, and the focus here will be on three:

1. Decision Tree Regressor (DTR)
2. Gradient Boosting Regressor (GBR)
3. Random Forest Regressor (RFR)

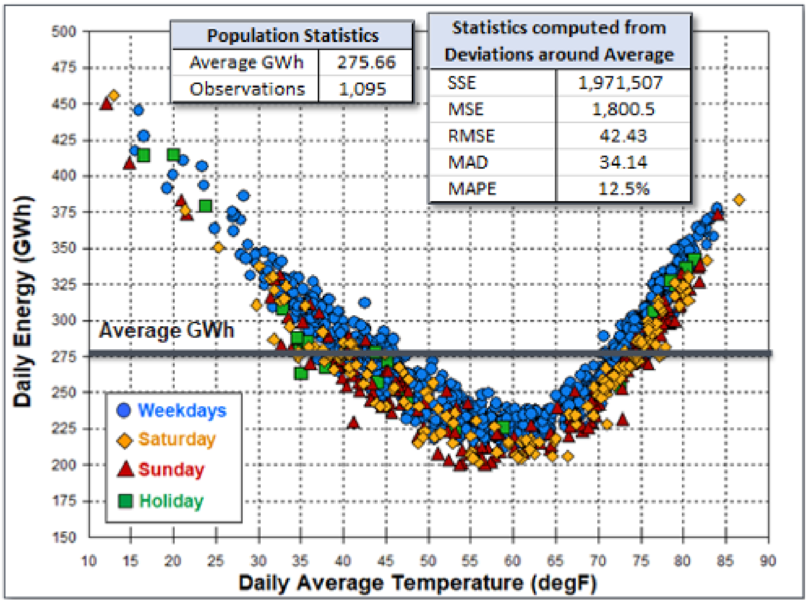
This paper provides details on how these methods work and how they perform when applied to the prediction of daily energy for an electric load zone. The next section provides a summary of the data that are used and a brief discussion of an OLS regression model that is used as a performance benchmark.

If you are new to decision trees, you will learn a lot about how they work and how they perform for daily energy prediction. If you are an experienced data-science type, you are likely to see a few new slants on how these models work and some new thoughts on feature engineering for daily energy forecasting.

DATA AND OLS BENCHMARK MODEL

Figure 1 shows daily energy in GWh on the Y axis and daily average drybulb temperature on the X axis. Each point in the figure represents one day. The points are color-coded by day type to show weekdays, weekend days, and holidays.

Figure 1: Daily Energy vs Daily Average Temperature



Population Statistics	
Average GWh	275.66
Observations	1,095

Statistics computed from Deviations around Average	
SSE	1,971,507
MSE	1,800.5
RMSE	42.43
MAD	34.14
MAPE	12.5%

Daily energy is computed from hourly system loads for Dominion Virginia Power. Daily temperature is computed from hourly values provided by AccuWeather for several weather stations in Virginia.

The population statistics block in the figure shows the number of observations (days) in the data and the average daily energy value. The second block shows statistics computed from the deviations of daily energy around the average value. The statistics are computed as though the deviations are model errors, which they are if you have an extremely naïve model that uses the average value as the model predicted value. With this interpretation, the statistics are:

- » SSE – Sum of squared errors
- » MSE – Mean squared error (SSE/N)
- » RMSE – Root mean squared error (Square root of MSE)
- » MAD – Mean absolute deviation
- » MAPE – Mean absolute percentage error

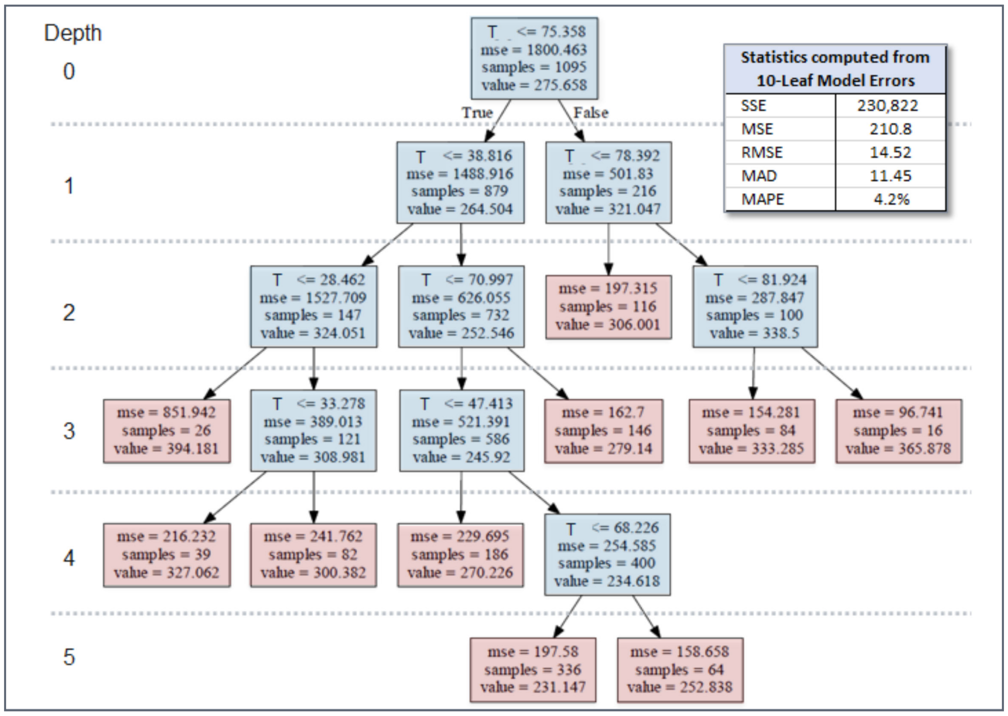
As you will see later, the SSE and MSE values are relevant to decision tree mechanics. The other statistics are included because they are easier to look at and understand. That is, for most people, it is much easier to relate to the MAD (currently =at 34.1 MW) or the MAPE (currently at 12.5%) than to the SSE (currently at 1,971,507).

To judge the accuracy of the decision tree methods, an OLS regression model is used as a performance benchmark. To capture the obvious nonlinear relationship between energy and temperature, the regression model uses a series of Heating Degree and Cooling Degree variables. Additional factors are binary variables for month and day of the week, holiday variables, a time-trend variable, and cloud cover and wind speed variables that interact with temperature. The specification is a solid and proven one for modeling daily energy, and we will see how decision tree models stack up against this benchmark. For reference purposes, the estimation MAPE for the benchmark regression model is 1.43%, so we are working toward that.

DECISION TREE REGRESSORS (DTR)

Before discussing terminology, let's build a simple Decision Tree Regressor for the daily energy data using daily average temperature as the only explanatory factor. Applying this method in Python with a single parameter setting (maximum leaf nodes = 10) produces the decision tree shown in Figure 2.

Figure 2: Simple Decision Tree With 10 Leaf Nodes


Depth
0
1
2
3
4
5
T <= 75.358 mse = 1800.463 samples = 1095 value = 275.658
True
False
T <= 38.816 mse = 1488.916 samples = 879 value = 264.504
T <= 78.392 mse = 501.83 samples = 216 value = 321.047
T <= 28.462 mse = 1527.709 samples = 147 value = 324.051
T <= 70.997 mse = 626.055 samples = 732 value = 252.546
mse = 197.315 samples = 116 value = 306.001
T <= 81.924 mse = 287.847 samples = 100 value = 338.5
mse = 851.942 samples = 26 value = 394.181
T <= 33.278 mse = 389.013 samples = 121 value = 308.981
T <= 47.413 mse = 521.391 samples = 586 value = 245.92
mse = 162.7 samples = 146 value = 279.14
mse = 154.281 samples = 84 value = 333.285
mse = 96.741 samples = 16 value = 365.878
mse = 216.232 samples = 39 value = 327.062
mse = 241.762 samples = 82 value = 300.382
mse = 229.695 samples = 186 value = 270.226
T <= 68.226 mse = 254.585 samples = 400 value = 234.618
mse = 197.58 samples = 336 value = 231.147
mse = 158.658 samples = 64 value = 252.838

Statistics computed from 10-Leaf Model Errors	
SSE	230,822
MSE	210.8
RMSE	14.52
MAD	11.45
MAPE	4.2%

Focus first on the top box (the Root node). All the observations are there, and the count is the same as in Figure 1 (1,095 cases). The MSE is the same as in Figure 1 (1,800). The average value is the same as in Figure 1 (275.66). So, Figure 1 is a good representation of what is happening in the Root node. We are just computing the average value of all the data and computing statistics from the deviations around this average.

The blue boxes, including the root node, represent decision nodes and the first line in each blue box shows the split rule. The split rule for the root node is T <= 75.358, where T is the daily average temperature. The branch to the left is for cases where the daily average temperature is at or below 75.358 and the branch to the right is for cases where the daily average temperature is above 75.358. We will discuss later how this split value is determined.

Starting at the root node, any temperature value that you pick will trace down through the decision nodes and end up in one of the red boxes, which are the terminal nodes or leaf nodes. We specified that we did not want more than 10 leaf nodes, and we got exactly 10. The splitting would have gone on much further had we not imposed this or some other limitation as a hyperparameter of the DTR model.

In each box, three statistics are computed from the data in that box. Samples is the number of data points in the box. Value is the average value of data points in the box. MSE is the mean square error, computed from deviations around the average value in each box.

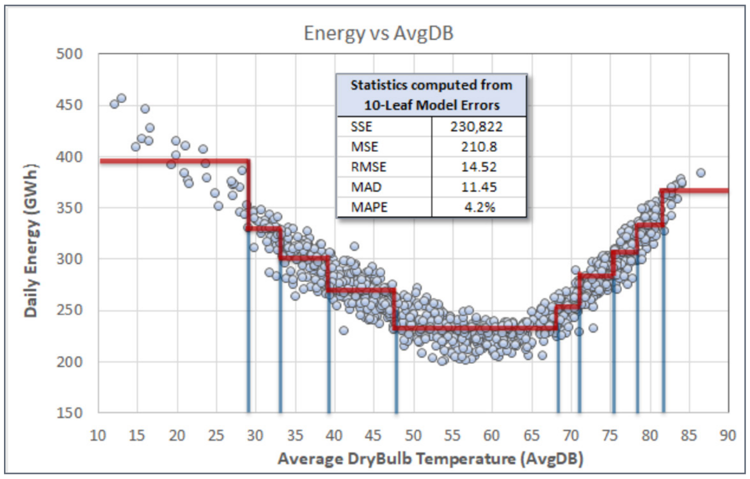
In Figure 2, the MSE values in the leaf nodes range between 96 and 327. This is a large improvement relative to the root node MSE value of 1,800. Model summary statistics are shown in the upper right-hand corner of Figure 2. With this simple 10-leaf decision tree, we have reduced the model MSE to 210.8 and the model MAPE to 4.2%. This is a big improvement from where we started in the root node, which had a MAPE of 12.5%.

If you look at the first red box on the far left, this contains the coldest days (at or below 28.462 degrees). You can see that the box contains data for 26 days and the average energy value is 394 GWh. If you look at the red box on the far right, this contains the hottest days (above 81.9 degrees). This box contains data for 16 days and the average energy value is 365.9 GWh.

It's a bit difficult to see because of all the split criteria, but as you move from left to right the red boxes move from colder days to warmer days. If you look closely, you will see that the split temperatures are not evenly spaced. A good way to understand this is to overlay the split temperature values on the scatter plot of daily energy versus temperature, as is shown below in Figure 3.

In Figure 3, each of the vertical blue lines represents one of the blue-box split temperatures from Figure 2. Each of the horizontal red lines represents the average daily energy use value in the corresponding leaf node. So, the red line in Figure 3 is the estimated DTR model shown as a step function. For any daily temperature value, the horizontal red line gives the corresponding daily energy prediction from the DTR model.

Figure 3: Depiction of 10-Leaf Decision Tree Predictions



Statistics computed from 10-Leaf Model Errors	
SSE	230,822
MSE	210.8
RMSE	14.52
MAD	11.45
MAPE	4.2%

Decision tree models are called "nonparametric," meaning that there is not a specific functional form such as a linear equation with slope parameters. Instead, the estimated model is defined by the split values that guide a temperature input to a leaf node. Once you get to a leaf node, you find the predicted value for that node. That is all there is. One of the advantages often mentioned for DTR models is that they are easy to understand, and in this case, with a single explanatory variable, that is the case. We will see that it gets a bit harder to understand when more explanatory factors are added to the model.

We could build the same model with a regression by defining binary variables that are "on" when the temperature falls in one of the leaf node boundaries and "off" otherwise. The estimated coefficients on these binary variables would be exactly equal to the averages within each leaf. This would be a nonparametric regression in the sense that there are no "temperature slope" parameters. But we would have to know the split values in advance to do this. The DTR model finds the split temperatures for us.

Decision Tree Terminology

As we all know, real trees grow upward. But with decision trees, it is customary to show them starting with the root node at the top and growing downward. With this exception, the terminology is relatively intuitive.

» **Root** – The starting point that includes all data observations

» **Node** – The Root node or a subsequent node created by splits

» **Branch** – A split from a node to a subsequent node

» **Leaf** – A terminal node that is not split further

» **Depth** – The number of node layers

» **Impurity** – The outcome variance within a node (MSE in the example)

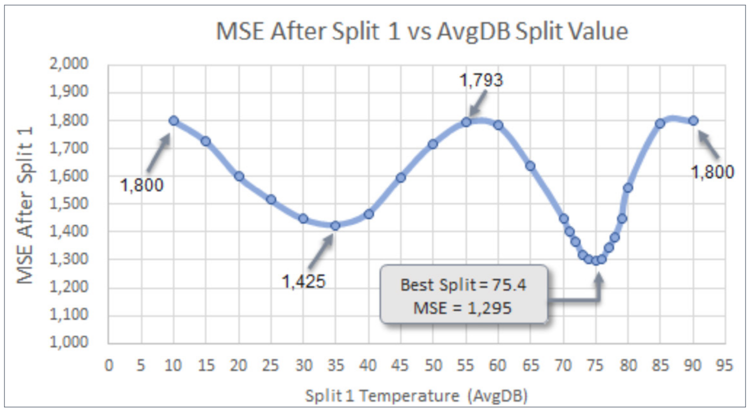
All of these concepts are clear in Figure 2, except for the use of the term “impurity.” The term impurity comes from classification problems, where the goal is to create leaf nodes that are homogeneous or pure. In the case of continuous outcome data, impurity is measured by the variation of outcome values around the average value in each leaf node. In the extreme, if we can create a leaf node where the outcome values are all the same, then the node will be absolutely pure (MSE of zero).

How Splits Work

All Decision Tree methods are based on the idea of splitting, and it is useful to understand how the split criteria is selected. You might be thinking that there is some grand strategic scheme here, but in most cases there is not. Let’s focus on the first split in Figure 2 to understand how the value of 75.4 degrees was determined. Looking at Figure 1, it is not clear what to expect. In fact, 75.4 degrees does not seem to be an obvious outcome. But when you think about it, the goal of the split is to find two groupings that have different average values. If the averages in the two splits are similar, there will not be much gain in terms of reduced SSE values.

To check this out, an Excel spreadsheet was set up to look at all of the possible split values. There are many possibilities since daily average temperature (the average of the hourly temperatures) is unique for most days. Figure 4 shows a plot of the results, with annotation for the extreme calculations.

Figure 4: MSE Values for Selected Split Temperatures…



Recall that the MSE around the overall average in the root node was 1,800. As Figure 4 shows, we get the same value if we split at the extreme cold end (10 degrees) or the extreme hot end (90 degrees), and this happens because there are no observations outside these boundaries. Splits near these extreme temperatures also have high MSE values because almost all the observations go to the colder node when we split near 90 degrees, and almost all observations go to the warmer node when we split near 10 degrees. We get a similarly high SSE value if we split in the middle (55 to 60 degrees). For splits in this range, the average values in the warmer node and cooler node are about the same as the overall average, and the SSE reduction is very small.

To reduce the SSE value, the split needs to result in average values that are different and that have counts big enough on each side for the difference in the average to matter. For the problem here, this means that we are either going to split on the cold side (around 35 degrees) or on the hot side (around 75 degrees). It turns out with these data that splitting on the hot side works better. But there is no way to know this without calculating the results on both sides. And that is what decision trees do through brute force calculation of all possible splits to find the one that works best.

The split logic is “greedy.” There is no multi-step strategy. When you are splitting a node, there is no looking backward. And there is no looking forward to subsequent splits. There is just the opportunity to split myopically the current node, and the split that is chosen will be the one that produces the greatest reduction for the node that is being split.

As shown in Figure 5, the first split has reduced the MSE from 1,800 to 1,295. Referring to the top two blue boxes in Figure 2 you can see that the cold side has most of the data (879 days) and an average value of 264 GWh. The warm side has the remaining data points (216 days) and an average value of 321 GWh. The process continues from there with the same greedy logic to split each of these two nodes.

If we let this process continue with no limits, we potentially end up with one observation in each leaf node. While this would reduce the SSE value to zero, it turns out that the model would not be especially useful for interpolation or extrapolation. This would be an example of overfitting. The potential for these types of methods to overfit in this way drives the keen focus of data science on out-of-sample testing to make sure that estimated models are useful for prediction and forecasting.

Deeper Models with Temperature Only

Next, let's look at more complex (deeper) DTR models with AvgDB temperature as the only variable. Experiments with different ways to constrain the complexity of the model suggested focusing on two: maximum tree depth and minimum leaf size. Using 10-fold Train/Test cross-validation the best out-of-sample results were obtained with the following settings (hyper parameters for the model):

» Maximum tree depth = 8

» Minimum leaf size = 9

With these settings, the estimated model ends up with a total of 115 nodes, 57 or which are decision nodes and 58 of which are leaf nodes. In some parts of the data, the leaf nodes are extremely narrow (less than one degree wide). The predicted values from model estimation using the full population data set are shown on the left-hand side of Figure 5.

On the right-hand side is a chart showing the predicted values for a comparable OLS regression model using a set of heating-degree variables (HD60, HD55, HD50, HD45, HD35) and cooling-degree variables (CD60, CD65, CD70, CD75, CD80) all of which are computed from the same average drybulb temperature that is used in the DTR model.

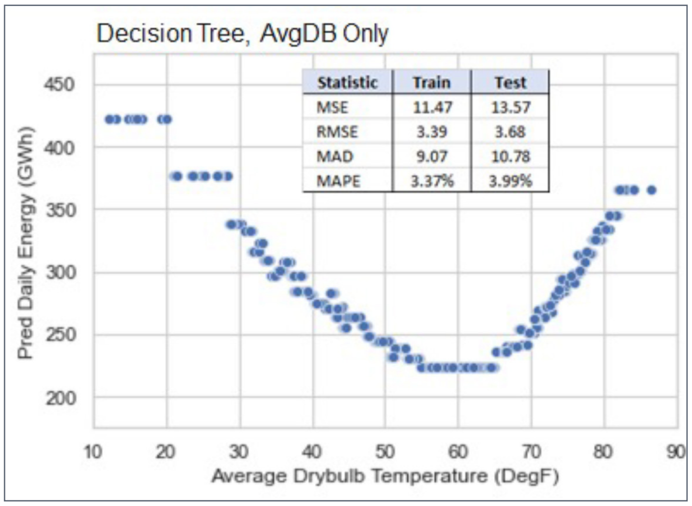
By construction, the OLS regression model generates a spline-like curve. In essence, the model is a series of linked slopes, and the slope is constant within each degree range. For example, there is one slope for degrees above 65, and then the slope changes for degrees above 70. The final slope on both sides (degrees below 35 and degrees above 80) will apply to extreme temperature inputs, even if the temperatures are outside the range of the estimation data set.

Looking at the middle temperatures (55 to 65 degrees), both models find the same "flat" range in the data. For the DTR model, the middle leaf node extends from 54.8 degrees to 65.0 degrees and includes 135 observations with an average value of 223.8. For the regression model, the slope is small for the first heating variable (degrees below 60) and for the first cooling variable (degrees above 60), and the predicted values on both sides are all in a narrow range centered at 225.

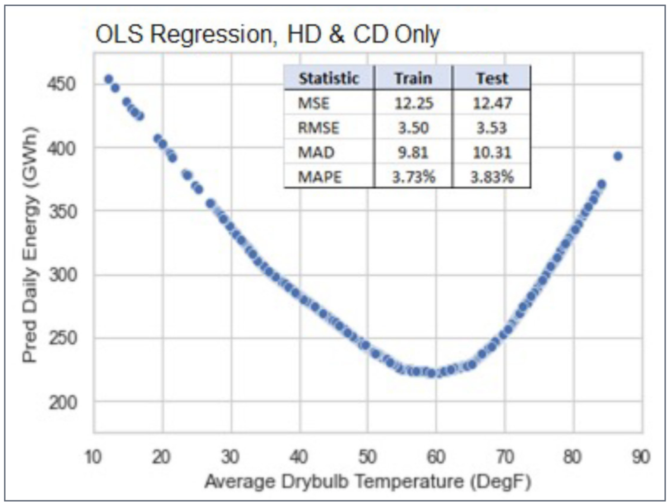
The main point where the models disagree is in the extreme left and right nodes, where the DTR model gives an equal forecast for all qualifying temperatures and the OLS model follows the slopes for the extreme heating-degree variable (HD35) and the extreme cooling-degree variable (CD80).

To test these two models based on daily average temperature, we ran 100 cross-validation tests. Each test estimated the models with 90% of the data and withheld a random subset of 10% for testing. The resulting statistics are shown in Figure 5. From the Train columns, you can see that the in-sample statistics for the DTR models are better than the in-sample statistics for the OLS models. For example, the in-sample or training MAPE is 3.37% for DTR and 3.73% for OLS. But the OLS models do slightly better out of sample, with a test MAPE of 3.83% compared to 3.99% for the DTR model.

Figure 5: Decision Tree and Regression Models using Average Temperature…



Statistic	Train	Test
MSE	11.47	13.57
RMSE	3.39	3.68
MAD	9.07	10.78
MAPE	3.37%	3.99%



Statistic	Train	Test
MSE	12.25	12.47
RMSE	3.50	3.53
MAD	9.81	10.31
MAPE	3.73%	3.83%

Deeper Models with Additional Features

It all gets more complicated with multiple explanatory factors. On the regression side, additional variables can be added by themselves or in structured interactions. For example, the wind variable is interacted with heating degrees (expecting a positive effect – on a cold day more wind makes it feel colder). The wind variable is also interacted with cooling degrees (expecting a negative effect – more wind on a hot day makes it feel less hot). We also interact day-time clouds with heating degrees (expecting a positive effect) and with cooling degrees (expecting a negative effect). These expectations reflect the fact that clouds block the direct warming influence of the sun and therefore increase heating loads and reduce cooling loads. Also, day-time clouds are included independently, and we expect a positive effect since more clouds in the day means less behind-the-meter solar production and therefore greater utility loads.

For the DTR models, we also need to define the factors to be included in the model. As with regression, the factors can be interaction variables. such as a windchill variable that interacts HD and wind. Or we can let the DTR model sort out the interactions, with wind coming into play as a split variable when that reduces the errors. The latter approach was taken here and the variables that are included are listed below in order of split-1 power. That is, the better a variable does as the split-1 variable, the higher on the list it goes. The list of variables is followed by Figure 6, which provides first-split statistics for each variable.

» **AvgDB**. This is the average drybulb temperature for each day. It is the most powerful variable and it is selected by the model for making the first split.

» **LagDB**. This is the average temperature on the day before the current day. It is highly correlated with the current day AvgDB variable but is not as powerful and therefore is not selected for the first split.

» **Lag2DB**. This is the average temperature two days prior to the current day. It is still highly correlated with the current day AvgDB. But for making the first split, it is a bit less powerful than the one-day lag variable.

» **Month**. This is a categorical variable with values 1 to 12. If this variable was used for the first split, the split value would be at 1.5 which cuts out January to one side and all other months to the other side. If we ran this variable deeper, it could split out all 12 months in the end.

» **DOW**. This is the day of the week with Monday as 1 through Sunday as 7. The first split using this variable would be at 5.5, which sends Monday (1) through Friday (5) to the left and Saturday (6) and Sunday (7) to the right. We will dig into that in a minute.

» **BankHols**. This variable has a value of 1.0 for holidays where banks and most office and government establishments are closed, but retail establishments are open. The first split using this variable would be at .5. This is the only thing that can happen, since this is a binary variable. It is very "unbalanced" and is not used much in the models.

» **RetailHols**. This variable has a value of 1.0 for the remaining holidays where most retail establishments are closed (in the US Christmas, New Years, and Thanksgiving are in this list). Again, the first split for this variable would be at .5 which is the only possible split with a binary variable.

» **Trend**. This is a trend variable that is 0 in 2010 and gains about 1/365 every day. The first split using this variable would be at 7.44 which turns out to be in early June of 2017.

» **AvgClouds**. This is the average of day-time cloud cover values, where clouds are measured on an octa (0 to 8) scale. The first split for this variable would be at 4.98 which puts partly cloudy days on the left and cloudy days on the right. This does not do much as a stand-alone variable because it needs to interact with temperature. In the DTR model, the early splits will be on temperature, and later splits based on clouds will occur independently for hot days and for cold days.

» **AvgWind**. This is the daily average of hourly wind speed in miles per hour. The first split for this variable would be at 5.01 which puts relatively calm days on the left, and windier days on the right. Like clouds, wind does not do much as a stand-alone variable because it needs to interact with temperature. Wind on a hot day lowers energy use. Wind on a cold (wind chill) increases energy use.

Figure 6: Additional Features and First Splits as Stand-Alone Variables

Variable	Split	Meaning	Meaning	Left Avg	Right Avg	MSE
	None	All the Data		275.66		1,800.5
		<=Split	>Split			
AvgDB	75.4	Not Hot (T<=75.4)	Hot (T>75.4)	265.40	321.05	1,294.2
LagDB	74.2	Not Hot (T<=75.4)	Hot (T>75.4)	264.34	312.01	1,389.0
Lag2DB	74.3	Not Hot (T<=75.4)	Hot (T>75.4)	266.34	305.89	1,518.8
Month	1.5	Jan	Feb-Dec	309.13	272.55	1,696.5
DOW	5.5	Mon-Fri	Sat-Sun	278.88	265.1	1,755.9
BankHols	0.5	No	Yes	275.43	292.14	1,796.7
RetailHols	0.5	No	Yes	275.37	299.27	1,793.8
Trend	7.44	Before 6/10/17	After 6/10/17	251.72	279.78	1,701.7
AvgClouds	4.98	Clear (0 to 5)	Cloudy (5 to 8)	280.41	263.95	1,744.8
AvgWind	5.01	Calm (<=5 mph)	Windy (>5 mph)	285.01	269.52	1,743.1

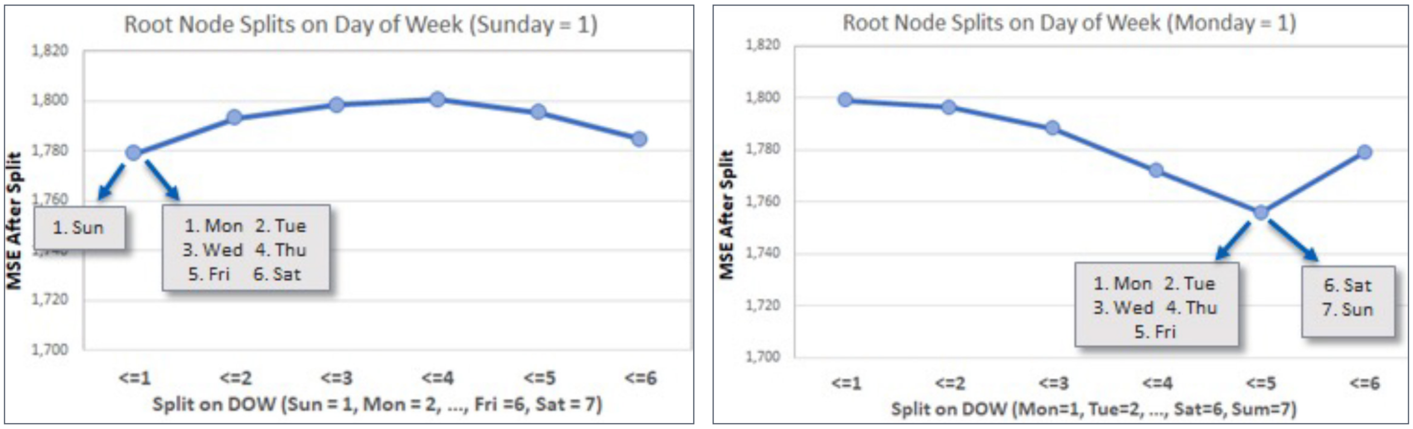
Figure 6 is useful for understanding how multi variable DTR models work. To get the numbers above, we had to find the best split in the root node for each of the X variables. The one that does the best (AvgDB) is selected and the first split is made. Looking at the estimated decision tree, the first few splits are made using this variable and then the other factors start to come into play. At that point, we are usually in a narrow temperature range, and we are making further splits on other factors, like day of the week, or month, or clouds.

Feature Engineering

You won't hear much about feature engineering in Decision Tree modeling. Often the attitude is to give the model the data and let it decide what to do. Of course, it is not actually "deciding" anything. It is merely executing a series of greedy splits using the features that are provided. However, as the following example illustrates, in some cases feature engineering can make a variable more powerful in the split process, and therefore more likely to be used in the model.

The example uses the day of the week (DOW) variable. Initially, the standard mapping was used where Sunday is 1 and Saturday is 7, as is the case in Excel. The results for all possible splits, in this case, are shown on the left-hand side of Figure 7. The best split is DOW<=1.5. This splits all observations for Sunday to the left and all other days of the week to the right. Next, the mapping was shifted so that Monday is 1 and Sunday is 7. The results for all possible splits with this definition are shown on the right-hand side of the figure. The best split is DOW<=5.5, which splits all the weekdays to the left and the weekend days to the right.

Figure 7: Root Node Splits with Alternative DOW Definitions



The reduction in MSE is twice as big using the Monday = 1 definition, making this a more powerful variable that is more likely to be used. With the Sunday = 1 definition, you can split Sundays off to the left and at some other point split Saturdays off to the right, but you can't split them off together.

We did not get into further feature engineering, although it seems like there could be similar benefit to alternative structuring of some of the other explanatory variables.

Multi Variable Decision Tree Regressor (DTR)

With the full set of factors, the potential complexity of the decision tree is greatly increased. As before 10-fold Train/Test cross validation was used to determine the best way to constrain complexity. This process was applied to each step of a model cascade, starting with AvgDB as the only variable and sequentially adding additional variable groups. As shown in Figure 8, as additional features are added, the optimal depth of the DTR model tends to increase and the optimal minimum leaf size tends to decrease. With the full set of features included, the best results were obtained with the following settings:

» Maximum tree depth = 12

» Minimum leaf size = 4

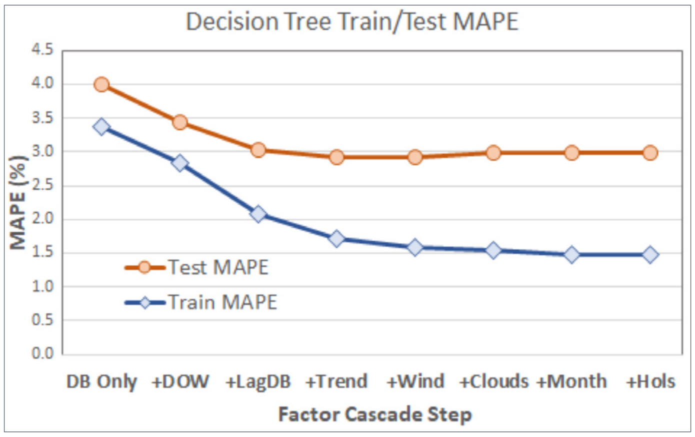
With these settings, the estimated model ends up with a total of 381 nodes, 190 of which are decision nodes and 191 of which are leaf nodes.

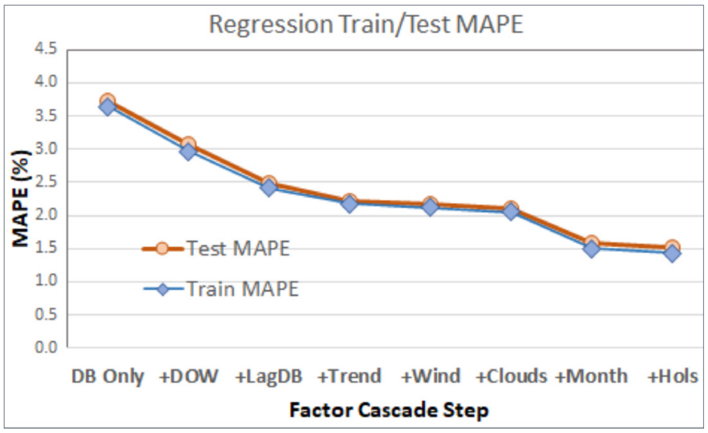
Figure 8 shows the optimal settings for each step of the DTR cascade on the left-hand side. On the right-hand table, the corresponding train and test MAPE values are shown for the DTR model and also for the comparable cascade of regression models. In both cases the train/test results are from 100 cross validation tests using shuffle and split with a 10% test fraction. The Train columns show that the DTR model does better in-sample for all rows except the very last row. The Test columns show the statistics for out-of-sample observations, and the Regression model does better for all rows. The advantage is strongest for the fully specified model, and the DTR MAPE value (2.98%) is almost twice as large as the Regression MAPE value (1.51%).

Figure 8: Root Node Splits with Alternative DOW Definitions

Optimal Settings		
Factors in Model	Max Depth	Min LeafSize
AvgDB Only	8	9
Add DOW	9	8
Add Lag DB	9	5
Add Trend	12	5
Add Wind	11	4
Add Clouds	11	4
Add Month	12	4
Add Hols	12	4

Factors in	Decision Tree MAPE		OLS Regression MAPE	
Model	Train	Test	Train	Test
AvgDB Only	3.373	3.990	3.648	3.729
Add DOW	2.839	3.437	2.968	3.081
Add Lag DB	2.074	3.037	2.413	2.487
Add Trend	1.711	2.911	2.182	2.225
Add Wind	1.584	2.930	2.121	2.172
Add Clouds	1.534	2.978	2.052	2.098
Add Month	1.470	2.978	1.501	1.580
Add Hols	1.470	2.980	1.429	1.514





The graphs on the bottom of Figure 8 show how the Train and Test statistics evolve through the feature cascade. The first thing that stands out is that regression models appear to generalize to out-of-sample observations much better than the DTR models. For all feature sets, the Test statistics for the Regression model are only a bit above the Train statistics, with a difference less than .1% in most cases. In contrast, for the DTR model, Test statistics are all well above (worse than) the Train statistics, and the gap widens as more features are used.

The second thing that stands out is that the regression model becomes more accurate as additional feature groups are added beyond DryBulb temperature, Day of Week, and Lag temperature. The Test MAPE at step 3 is about 2.5%, and it finishes at about 1.5% with all the features. In contrast, the test MAPE for the DTR models does not improve significantly beyond the initial feature groups, stalling at about 3%.

This reflects the fact that the regression model is able to take advantage of secondary variables that interact with heating and cooling variables across broad temperature ranges. In contrast, the DTR model is not able to get much power out of the later factors after the observations are split into narrow temperature ranges. For example, when you look at the detailed DTR model, variables like Clouds, Wind, and Month do appear sporadically as a split variable, but they are not used in the path to most of the leaf nodes.

GRADIENT BOOSTING REGRESSION (GBR)

The second decision tree method explored here is Gradient Boosting Regression (GBR). GBR uses the same splitting idea but applies it in a very different way. The following is a simple outline of the method:

» Compute the deviations around the overall mean value (Step 0 errors)

» Apply decision tree logic to Step 0 errors and apply learning rate to get Step 1 errors

» Apply decision tree logic to Step 1 errors and apply learning rate to get Step 2 errors

» Repeat to get updated errors for Step 3, 4, 5, …

This is a very different idea, and at first glance, it is not clear how this will work or why it is useful. But it does work, and it is useful, and it therefore deserves a deeper dive.

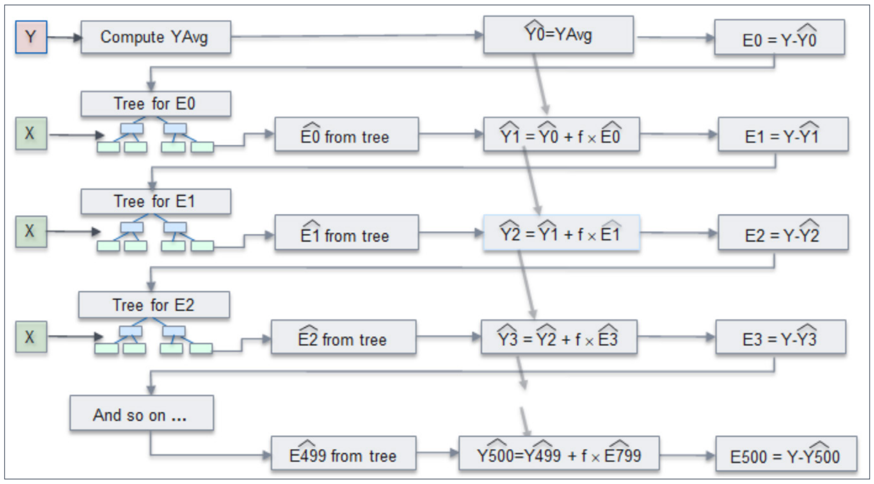
GBR Logic

The following provides a more complete outline of the method. Feel free to skip this unless you like flow charts and equations. The method starts with the calculation of the average Y value and then proceeds with a stack of sequential models. At each step, the DTR method is applied to the cumulative residuals from the prior step. The predicted residual values at each step are only partially used, based on a learning-rate fraction (f). In outline form, the method is as follows:

Step 0:	Compute average Y value (YAvg).
	Use YAvg as the predicted value for all observations (Pred0 = YAvg)
	Compute Step 0 "errors" as E0 = Y – Pred0
Step 1:	Apply DTR to E0 values using explanatory factors
	Compute predicted E0 values (E0_Pred) as the average error in each leaf node
	Update predicted value: Pred1 = Pred0 + f * E0_Pred
	Compute Step 1 errors as E1 = Y – Pred1
Step 2:	Apply DTR to E1 values using explanatory factors
	Compute predicted E1 values (E1_Pred) as the average error in each leaf node
	Update predicted value: Pred2 = Pred1 + f * E1_Pred
	Compute Step 2 errors as E2 = Y – Pred2
Step 3 to N:	Repeat the process until complexity limits are reached.

This process is represented in Figure 9 with the total number of steps set to 500.

Figure 9: Gradient Boosting Estimation Process



In addition to the learning-rate fraction (f), GBR, like DTR, has hyperparameters that limit the complexity of the model. Python has two versions of the gradient boosting method (Gradient Boosting Regression (GBR) and Extreme Gradient Boosting Regression (XGBR). The following uses XGBR which performed slightly better out of sample. The following settings are available for specifying the gradient boosting regression model.

- n_estimators: the number of estimation steps
- learning_rate: the fraction of the predicted residuals added to predicted Y at each step
- criterion: the formula used to make decisions about split factors and values
- complexity settings: settings that limit the depth or width of the trees at each step
  - -- max_depth – the maximum number of decision tree levels at each step
  - -- max_features – the maximum number of features to use at each step
  - -- min_samples_split – do not split nodes below this size
  - -- min_samples_leaf – do not allow leaf nodes smaller than this
  - -- max_leaf_nodes – do not allow more than this number of leaf nodes
- stochastic parameters – settings that introduce randomness into the process
  - -- subsample – the number or fraction of the training data to use at each model step
  - -- max_features – the number or fraction of features to use at each split

If you read up on this, you will find that applications usually end up with the number of estimation steps in the hundreds, with the learning rate set relatively low (.10 is the default), and with the complexity settings like max_depth set very tight. It is not at all obvious that this should be the best approach, but this configuration often works best in out-of-sample tests. Results for the model of daily energy use follow this pattern. Based on a grid search with 10-fold cross validation, the optimal settings are:

» n_estimators = 500

» learning_rate = .11

» max_depth = 2

» subsample = .8

» max_columns = 10 (all)

The stochastic parameters (subsample and max_columns) are introduced to provide some random elements to the process. The subsample parameter (at .8) means that a random sample of 80% of the training data will be used at each model step. The max_columns parameter (if it is set to less than 10) means that a random subset of the 10 features will be used at each split. The idea is to prevent the model from over fitting the training data. In many cases, this will improve the out-of-sample accuracy of the model.

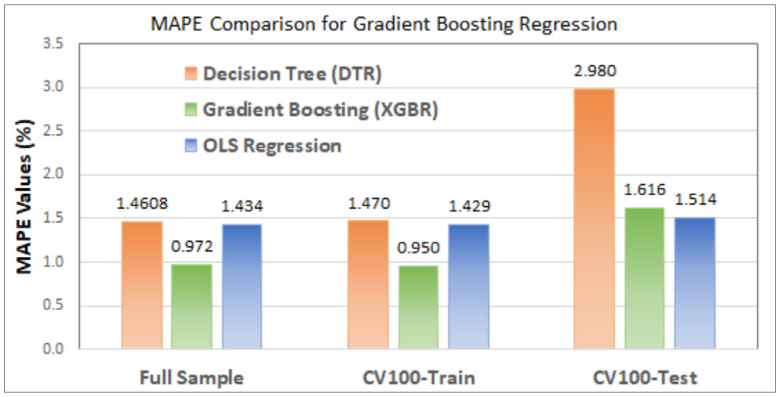
The number of trees in the vertical stack of trees showed strong improvement up to 300 levels and modest improvement beyond that. In the parameter grid search with 10-fold cross validation, there was no advantage to going beyond 500. The best value for the maximum depth setting is 2. (The best value for maximum leaf nodes was 3, but results were slightly better using max_depth).

So, this is a tall stack of really small decision trees. Maximum depth = 2 means that, for each step, we split the top node once and then split each side again, giving four leaf nodes. That seems abrupt, but it works best (that is, it gives the best out-of-sample results). These settings allow the model to do some useful things. For example, the tree for a step can split the residuals on temperature into a hot side and a cold side. Then split each side based on windspeed or cloud cover. This allows wind and cloud impacts to be calculated and applied to a broad range of temperatures, which is something the DTR model (with a single wide and deep tree) did not seem to be able to do.

GBR Results

Figure 10 shows comparison of the average value of MAPE statistics for 100 Train/Test model estimations. As shown, the GBR models do much better than the DTR models in the out-of- sample tests. In fact, the GBR average MAPE of 1.62 is close to the OLS average MAPE of 1.51. It is somewhat of a surprise that this tall stack of weak learners with a small learning rate provides such a powerful result, providing one more example of why gradient boosting is sometimes called the Swiss army knife of data science.

Figure 10: Comparison of Train and Test Accuracy Statistics



While it was possible to look at the individual trees in the estimated GBR model, it is very difficult to understand how the model is actually working. With 500 trees in the stack, each with 4 leaf nodes, there are 2,000 leaf nodes. And, because of the small learning rate, each 4-node tree is making a very small contribution to the overall predicted value. But it is apparent that this method works quite well, as witnessed by the strong out-of-sample performance.

A Note on Splitting Calculations

For DTR, splits were made by minimizing the sum of squared errors, the default option. Other options are available, and GBR maximizes the Friedman MSE formula as the default. Other boosting methods, such as Extreme Gradient Boosting, use a similarity gain metric which can be extended to include regularization terms. For the problem here, the results did not seem to be sensitive to the choice of the splitting metric. Although the formulas look different, it can be shown that (regularization aside) minimizing the SSE, maximizing the Friedman MSE and maximizing the similarity gain are equivalent in terms of the split results. Algorithmically it seems like the Friedman MSE may be more efficient, but that was not an important consideration for a small data set like the one used here.

RANDOM FOREST REGRESSOR (RFR)

So far, we have looked at Decision Tree Regression (DTR), which is a single deep and wide tree, and Gradient Boosting regression (GBR), which is a tall stack of sequential small trees. Random Forests take the idea in a different direction. Instead of a single tree, there are many trees. Each is estimated separately, and each is a bit different by design.

The first reason each tree is different is that estimation is performed using a bootstrap sample of the training dataset. Bootstrap samples are constructed using sampling with replacement, and in Python the default sample size is the same as the size of the training data set (it can also be set to be smaller if that is useful). In bootstrap samples, some data points will be included multiple times, and some will not be included at all. The result is that each tree in the forest is estimated with a different subset of the data and with different weights on the observations that are included.

The second reason each tree is different is that each split calculation is made using a random subset of the explanatory features in the model. The size of the subset is controlled by the max_features setting.

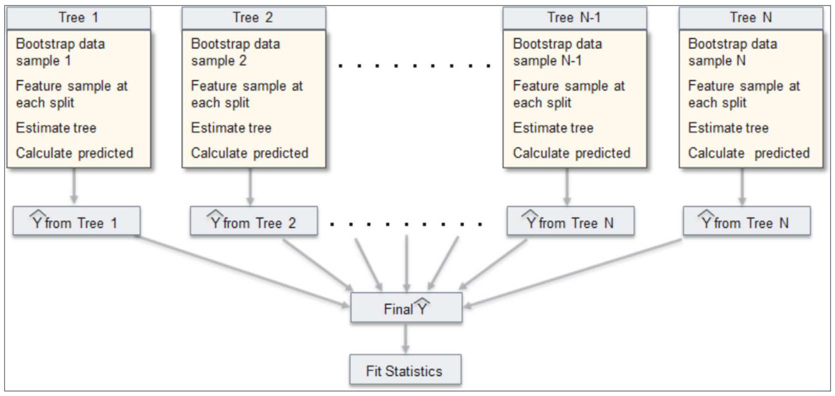
By introducing this randomness, each tree will have a unique decision structure. Once all trees are estimated, the predicted values are averaged or weighted together to get the overall prediction from the random forest model.

Random Forest Regressor Logic

This RFR process is depicted in Figure 11. The number of independent decision trees is a hyper parameter for the model and in Python it is controlled by the n_estimators parameter. The complexity of all trees in the forest is determined by a set of depth parameters (like max_depth, max_leaf_nodes, and min_node_size).

The degree of randomness is controlled by the additional parameters discussed above that control the size of bootstrap samples and the limit on the number of features included in split calculations.

Figure 11: Depiction of Random Forrest Regressor (RFR) Estimation



Based on grid search calculations with 10-fold cross validation, the following hyper parameters were selected:

» n_estimators = 100, the number of trees in the forest

» max_features = 8, the maximum number of features to use in split calculations

» max_samples = 1.0, use all the training data to develop the bootstrap samples

» max_depth = 13, the maximum depth of each tree in the forest

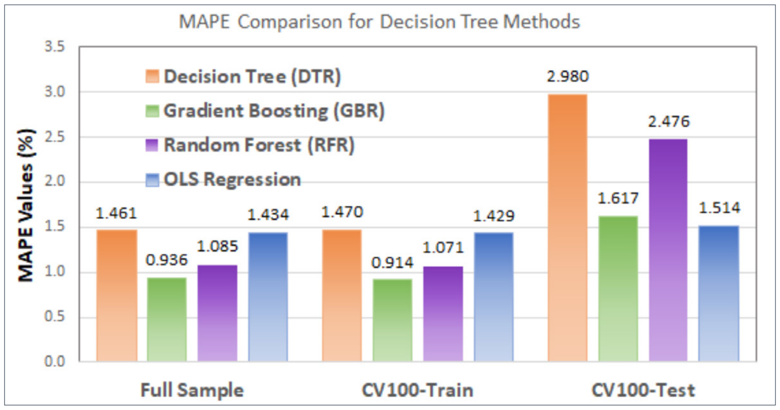
» max_leaf_nodes = 250, the maximum number of leaf nodes in each tree.

There were strong out-of-sample accuracy gains as the number of trees increased from 1 to 50, and then weak increases as the number of trees was increased further to 100. There was no gain beyond 100 trees. Out-of-sample accuracy improved as the number of features used in each split was increased from 4 to 8, and then decreased beyond 8. The max_samples setting was best at 1.0, meaning that the bootstrap sample for each tree is constructed to be the same size as the original data set (1,095 observations).

Random Forest Regressor Results

Figure 12 shows a comparison of the average value MAPE statistics for the 100 Train/Test model estimations. The results show that averaging multiple independent wide trees (Random Forest) is better than a single large tree (DTR) but not as good as a tall stack of sequential weak learners (GBR).

Figure 12: Comparison of Train and Test Accuracy Statistics



RANDOM FOREST AND GRADIENT BOOSTING COMBINED

The Extreme Gradient Boosting Regressor (XGBRegressor) method in Python is based on the XGBoost library, and it supports a combination of the Gradient Boosting and Random Forest ideas. Visualize multiple trees (as in random forest) where each tree is a tall skinny boosting tree (a stack of sequential weak learners). This method has two size parameters. First, num_parrallel_trees is the number of trees (the width of the forest). Second, n_estimators is the height of the boosting stack for each tree

This method was applied to the daily energy data with the following settings:

» num_parallel_tree = 50, the number of trees in the forest

» n_estimators = 500, the number of levels in each gradient boosting tree

» max_depth = 2, the maximum depth of trees in each boosting step

» learning_rate = .1, the gradient boosting learning rate

» subsample = .7, the fraction of the training set used in estimation

» colsample_bylevel = .9, the fraction of the features used in estimation

» reg_lambda = 11, the regularization parameter

With these settings the average MAPE statistics for the 100 Train/Test estimations came out at 1.559, which is about halfway between the accuracy of the GBR gradient boosting model and OLS regression model.

REGRESSION AND GRADIENT BOOSTING COMBINED

In the first step of Gradient Boosting, the average value of the Y variable is computed and residuals are calculated around that average. The subsequent boosting steps proceed from this truly naïve initial model. In this section we look at what happens if we replace that initial step with the OLS regression model. That is, in step 0, we run the OLS regression and take the residuals from that model as the starting residuals for a GBR boosting model. Once that GBR model is estimated, the predicted residual values are added to the predicted values from the regression model to get the combined prediction.

In this combined method, the boosting component is an error correction process that is focused on modeling any systematic bias. The boosting tree identifies relationships between regression model residuals and the explanatory factors, including possible interactions among the explanatory factors. If this error correction process provides better performance in out-of-sample tests, we can expect that the combined model will provide a more accurate forecast.

As an aside, this is similar to time-series error correction, such as a regression model with ARMA errors (RegARMA). There is a difference, however. There is an argument that RegARMA produces better parameter estimates and more reliable standard error estimates for the regression parameters as well as more accurate short-term forecasts. There is no such argument with the regression/boosting combination, since the regression parameter estimates and estimated standard errors are not impacted by the subsequent boosting model. The main benefit, if the method works, is improved forecast accuracy. (Note, we could apply the boosting step to the squared errors to build an error variance model and use the results to do weighted least squares, but that is a story for another time).

The regression model is discussed briefly in the introduction. The boosting model uses the XGBoost version with the following settings:

» n_estimators = 500, the number of levels in the gradient boosting tree

» max_depth = 2, the number decision tree levels at each step in the boosting tree

» learning_rate = .1, the gradient boosting learning rate

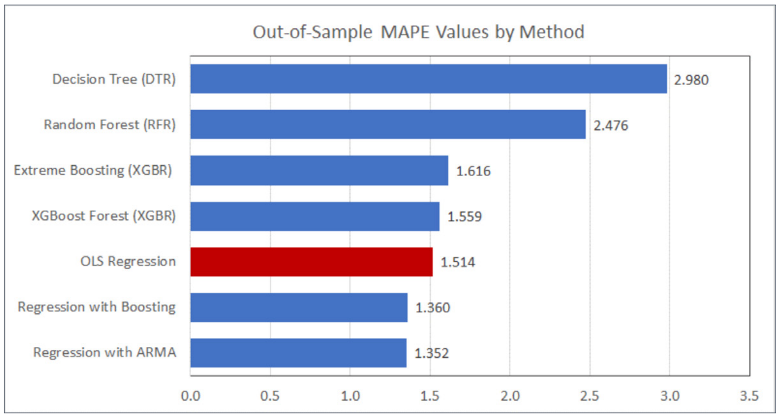
» subsample = .7, the fraction of the training set used in estimation

» colsample_bylevel = .8, the fraction of the features used in estimation

» reg_lambda = 11, the regularization parameter

Figure 13 shows the comparison of out-of-sample MAPE values for all the methods discussed above, with an additional entry for Regression with an AR1 correction.

Figure 13. Comparison of All Methods



CONCLUSIONS AND FINAL THOUGHTS

Decision tree regression methods are nonparametric. There are no slope parameters. Just one or more decision trees which are defined by split rules that assign an observation to a terminal leaf based on explanatory factor values. Any observations that are assigned to a terminal leaf get the same predicted value, which is the average of the training set values in that leaf.

For the problem analyzed here, which is prediction of daily energy values, the ranking of the core methods is clear. Gradient Boosting Regressor is best, followed by Random Forest Regressor, followed by Decision Tree Regressor. Going one step further, a Gradient Boosting Forest is a bit better than Gradient Boosting with a single-boosting tree. The boosting methods come close to OLS Regression, and it is possible that they would do as well with additional feature engineering.

Although decision trees are conceptually simple (just a series of splitting rules), it is hard to see how these models work when there are multiple explanatory factors and many levels. Whereas a regression model can be decomposed to show the role of each individual explanatory variable for each observation, this type of insight is difficult to squeeze out of the decision tree splits. The biggest difference is at the extremes, where regression methods will extend the prediction with a temperature slope, but decision trees will just return the average value for the most extreme leaf.

That said, the Gradient Boosting methods are competitive with regression and the possibility of combining the two methods is promising.

Join us in creating a more **resourceful world**.

CORPORATE HQ
2111 North Molter Road
Liberty Lake, WA 99019 USA
Phone: 1.800.635.5461
Fax: 1.509.891.3355