Demand Forecasting using Tree-based Regression: Random Forest

UC Davis | MSBA | 442 – Advance Statistics | Team Paper

March 9th, 2018

By Team Nugget – Crystal Liu, Daria Zhan, Mei Yang, Pallavi Sama, Sarah Gustafson

Table of Contents

Overview

In this paper, we introduce the concept of Tree-Based Regression using Random Forest and use it to describe and create causal models for demand forecasting purposes. We begin by defining regression and by giving a brief introduction to Tree-Based Regression modeling and the specific technique that we will be employing: Random Forest. We then move on to explain the relationship between the Tree-Based Modeling and Random Forest and demonstrate how Random Forest can be used to perform regression analyses. This paper intends to discuss these concepts at a high level with simple English definitions supplemented with basic mathematical and statistical notations to connect the dots between the model's logic and implementation.

In the implementation section we demonstrate how to use Random Forest to perform demand forecasting using a drug store chain sales dataset. First, we give an overview of the dataset that is used to create the model. We identify the factors or drivers of demand, called independent, causal or explanatory variables which influence the variable to be forecasted or the dependent or response variable. We perform some exploratory analyses on the data to demonstrate the relationship between these variables.

We build the model using off-the-shelf machine learning algorithms in the randomForest package provided in R (open source software platform for statistical modeling and graphics). Once the model is trained, we validate the results using the held out portion of the dataset (cross validation). We assess the performance of the model using in-built functions and basic statistical metrics and conclude the analysis by presenting the results of the same. Finally, we discuss some aspects of the model that can be improved further to increase the accuracy of the regression algorithm.

Introduction to Tree-based Regression

Regression analysis is used to determine the relationship between two or more variables. It is primarily used for prediction and causal inference. It uses statistical methods that attempt to fit a model to data in order to quantify the relationship between the response (dependent) variable and the predictor (independent) variable(s).

Tree-Based Regression or Decision Tree models are known for their simplicity and efficiency when dealing with domains with large number of variables and cases. Regression Trees are obtained using a fast divide and conquer greedy algorithm that recursively partitions the given training data into smaller subsets. The use of this algorithm is the cause of the efficiency of these methods.

Below is a high-level description of the Recursive Partitioning (RP) algorithm employed in building a Regression tree. The algorithm has three main components:

- Splitting rule – A way to select a split test
- Termination criterion – A rule to determine when a tree node is a terminal
- Value Assignment – A rule for assigning a value to each terminal node

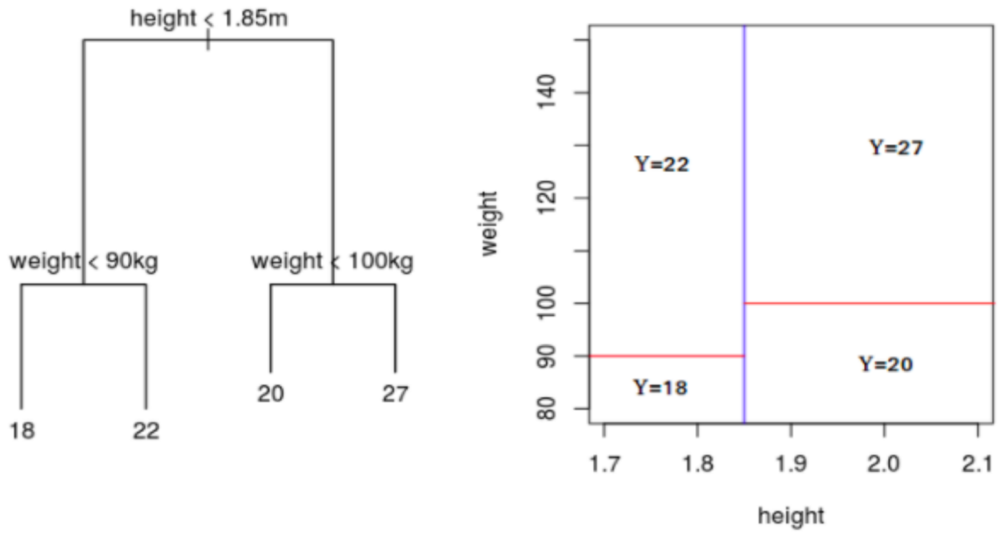
Recursive Partitioning repeatedly splits the dataset into two parts so as to achieve maximum homogeneity within the new parts. Following are the steps at a high-level:

1. *Pick one of predictor variables, Xi*
2. *Pick a value from Xi, say Si, which divides training data into two portions*
3. *Measure the purity (containing records of mostly one class) of resulting portions*
4. *Try different values of Xi and Si to maximize purity in initial split*
5. *Once you get a "maximum purity" split, repeat the process for a following split*

Purity is measured in terms of Information gain (IG) which in turn is based on Entropy values. Entropy is proportional to the homogeneity of data and is zero for completely homogenous dataset. IG is calculated as:

Information Gain = Entropy(Parent Node) - Weighted Sum Entropy(Child Node)

Following example shows a simple Tree-Based Regression model to predict the number of points scored by a set of basketball players based on their heights and weights. The figure below shows two different representations: 1) In the left we have the tree itself and 2) in the right, it shows how the space is partitioned (the blue line shows the first partition and the red lines the following ones). The numbers in the end of the tree (and the partition) represent the value of the response variable. Therefore, if a basketball player is higher than 1.85 meters and weighs more than 100kg, they would be expected to score 27 points.



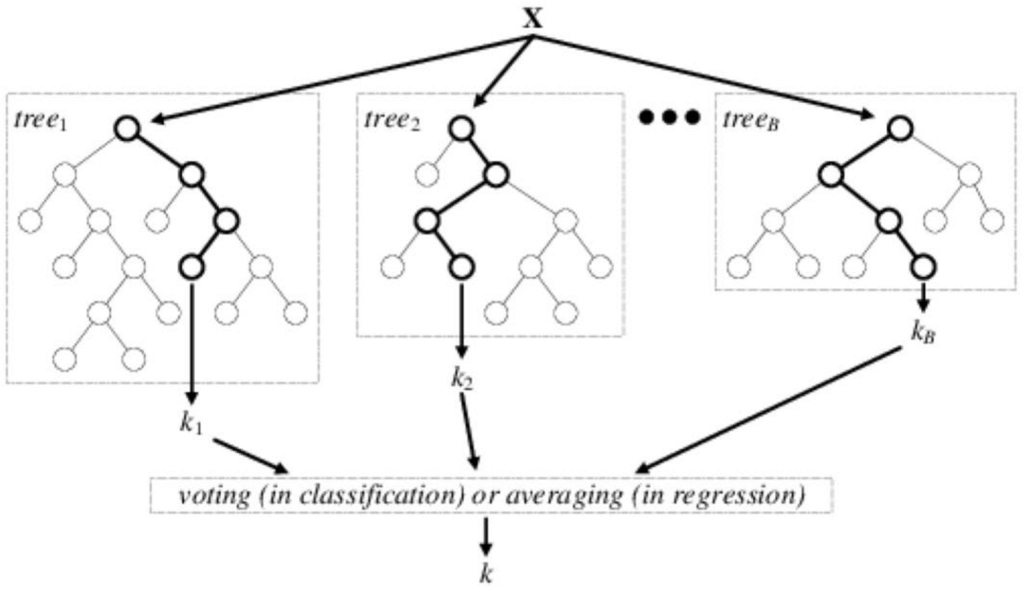
Now that we are aware of how the tree-based regression models are created, let us move on to the next section to see how the Random Forest technique further enhances these models by overcoming some of their shortcomings.

What is Random Forest?

The Random Forest technique is an ensemble learning method typically used for classification or regression tasks. It operates by constructing a multitude of Decision Trees during model training and outputting the class that is the mode of the classes (Classification) or mean prediction (Regression) of the individual trees. Decision Trees are known to be very unstable and suffer from the high probability of overfitting to the training dataset. The Random Forest uses this instability as an advantage through bagging which results in a very stable model.

Bagging solves the instability issue by giving each variable several chances to be in the model. It uses the idea of **B**ootstrap **Agg**regat**ing** in the sense that the procedure is repeated in many bootstrap samples and the final outcome is the average outcome across these samples. In other words, it uses the combinations with repetitions to produce multisets of the same size as the original data. By increasing the size of the training set, it doesn't improve the model predictive force, but decreases the variance, thereby narrowly tuning the prediction to the expected outcome.

Random Forests provide an improvement over bagged trees in one way: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable, these features will be selected in many of the trees, causing them to become correlated. Below is simple flowchart illustration.



How does Random Forest work for regression?

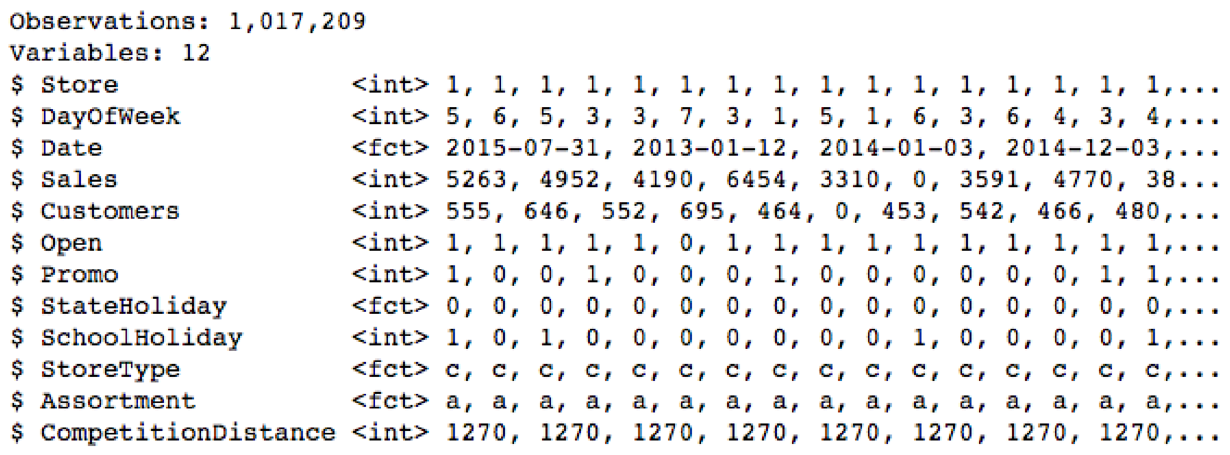
As mentioned before, Random Forest uses the mean prediction of the individual trees to output the final prediction in a Regression Model. The steps for a high-level algorithm are given below:

1. *Choose the number of trees, ntree (default in R is 500)*
2. *Variables randomly sampled as candidates at each split, mtry (default p/3 for regression)*
3. *Draw ntree bootstrap samples*
4. *For each bootstrapped sample, grow un-pruned tree by choosing the best split based on a random sample of mtry predictors at each node (splitting process is same as any tree-based method)*
5. *Calculate the final prediction as the average of all trees' leaf nodes*

Once the forest is built, cross-validation is used to assess the performance. Cross-validation splits current data into train and test datasets. It is a better model evaluation method than using residuals. Although, the data scientists generally perform cross-validation as a separate exercise, Random Forest has an internal mechanism called **Out-of-Bag (OOB)** estimates to do the same at runtime. In Random Forest, since each tree is parallel on a unique bootstrap sample of data, each bootstrap sample should contain approximately 63% of data. The rest 37% is used for testing, and these are called the OOB samples.

Implementation

We used R to implement the Random Forest model by leveraging the off-the-shelf algorithm from randomForest package. To demonstrate the usage of Random Forest, we have used a drug store sales dataset available online. The dataset contains historical sales from over 1000 stores of a drug store chain. The timeframe of the data is close to three years (Jan 2013 - July 2015). It contains many factors which could possibly affect the sales like store type, presence of promotions, holidays, distance from competitors, etc. Our objective is to perform a regression analysis on Sales using the other variables. Below is a snapshot of the dataset:

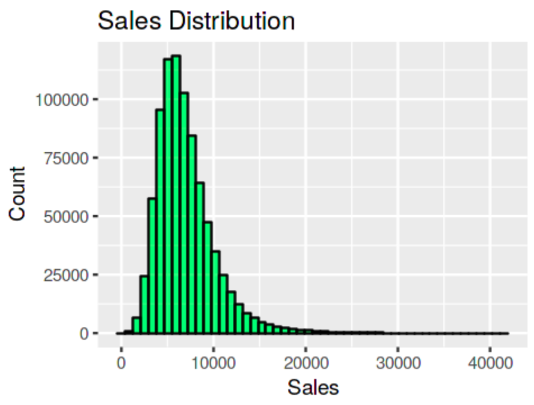


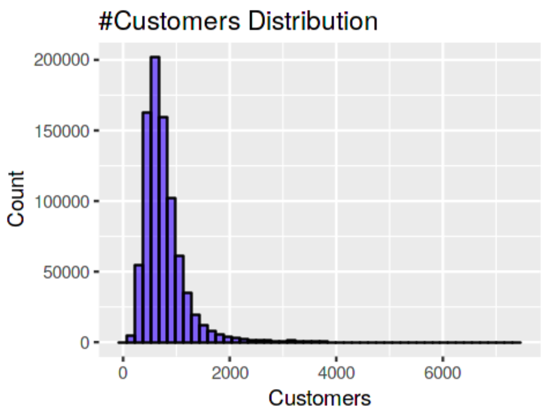
```
Observations: 1,017,209
Variables: 12
$ Store               <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ DayOfWeek           <int> 5, 6, 5, 3, 3, 7, 3, 1, 5, 1, 6, 3, 6, 4, 3, 4,...
$ Date                <fct> 2015-07-31, 2013-01-12, 2014-01-03, 2014-12-03,...
$ Sales               <int> 5263, 4952, 4190, 6454, 3310, 0, 3591, 4770, 38...
$ Customers           <int> 555, 646, 552, 695, 464, 0, 453, 542, 466, 480,...
$ Open                <int> 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,...
$ Promo               <int> 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,...
$ StateHoliday        <fct> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
$ SchoolHoliday       <int> 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,...
$ StoreType           <fct> c, c, c, c, c, c, c, c, c, c, c, c, c, c, c, c,...
$ Assortment          <fct> a, a, a, a, a, a, a, a, a, a, a, a, a, a, a, a,...
$ CompetitionDistance <int> 1270, 1270, 1270, 1270, 1270, 1270, 1270, 1270,...
```

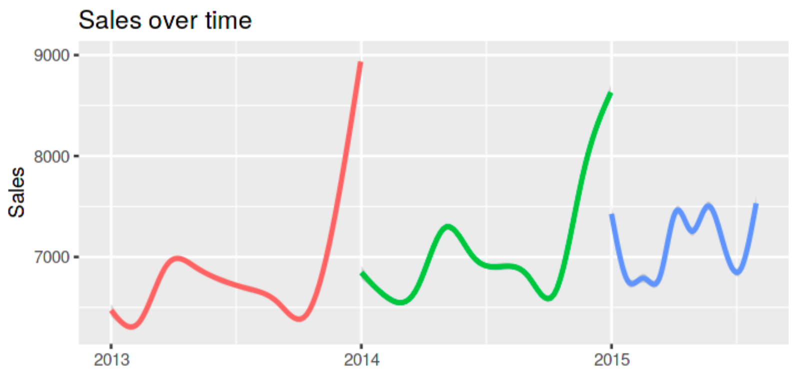
To build the model, we followed a basic set of operations, listed below (detailed R code available with the project submission. Also available on github):

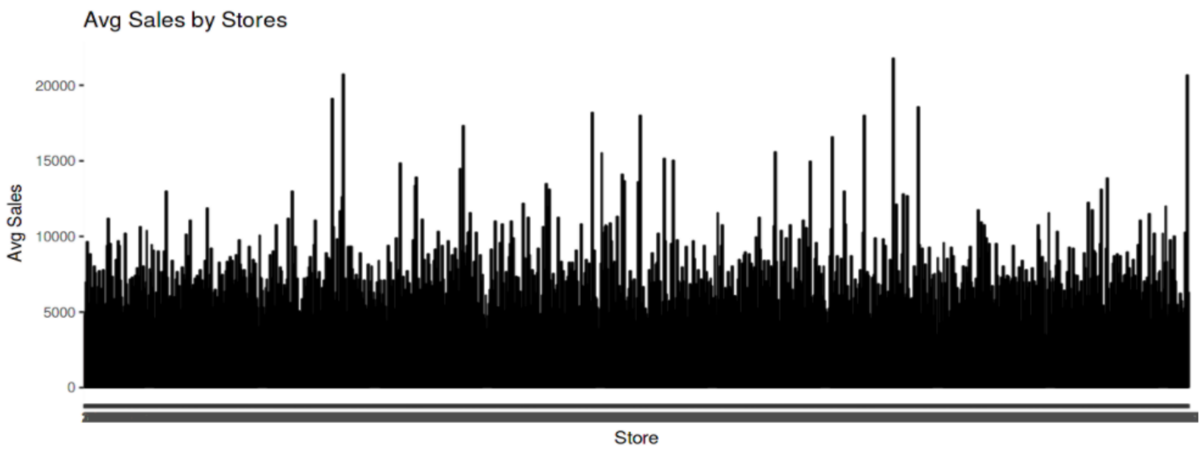
- Install and load the required packages in R (Use Tools -> Install on Rstudio, or run install.packages() in console)
- Read the dataset and get a quick summary statistics of the variables – Frequency of categorical variables, Mean/Median on numerical variables
- Clean the dataset – treat for missing values, drop the irrelevant rows/columns – We imputed missing values in competition distance column with median. We filtered out for rows with non-zero sales and open days.

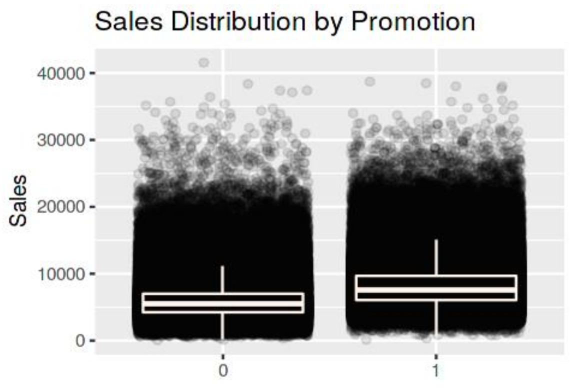
- Perform exploratory analyses to check the relationship among the variables (esp. the response and predictor variables) – We created histograms for continuous variables like sales, #customers, competition distance, and boxplots for categorical features like store type, day of the week, months, etc., and assessed their spread. Some of the self-explanatory graphs created using ggplot package are given below:

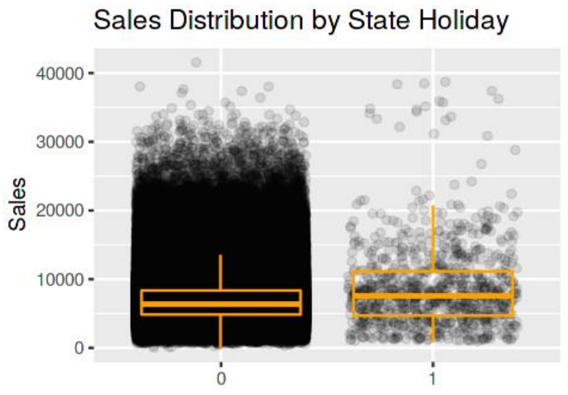


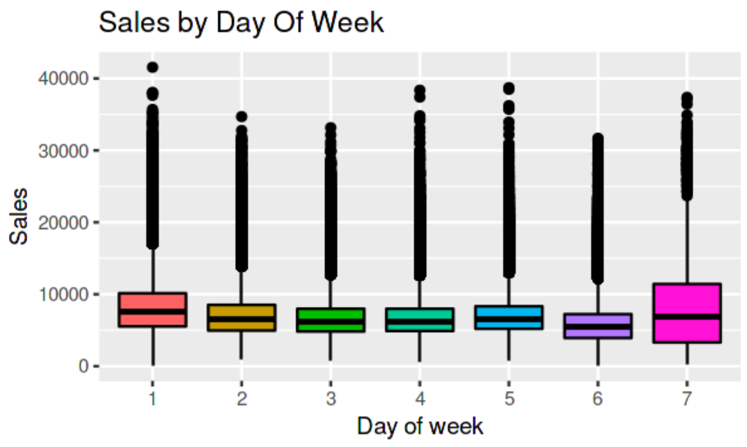






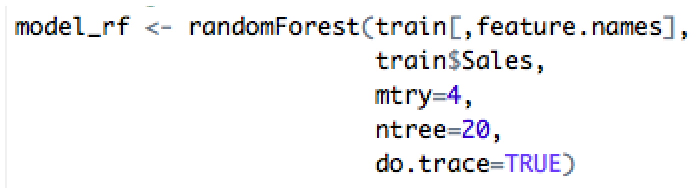






- Feature engineering - Create some new variables, if required - There were over 1000 stores and even though sales varies across them, R cannot handle categories with more than 53 levels. So we created a new variable as average sales per month for each store. Number of Customers on a given day has a very high correlation, which is obvious, but that itself could be a response variable to predict, so we created a new variable as average number of customers per month for each store.

- Split the dataset into train and test set for cross validation - We split the dataset into train and test sets using the 70:30 rule (70% used to train the model and rest 30% held out for cross validation). We retain only the "candidate" variables based on the EDA and feature engineering as the predictors and sales as the response variable (Keeping the open indicator or year did not make much sense here).
- Build the model & assess the performance - Once the dataset is ready, we build the model using the randomForest package in R. Below is a snapshot of the code snippet:



```
model_rf <- randomForest(train[,feature.names],
                         train$Sales,
                         mtry=4,
                         ntree=20,
                         do.trace=TRUE)
```

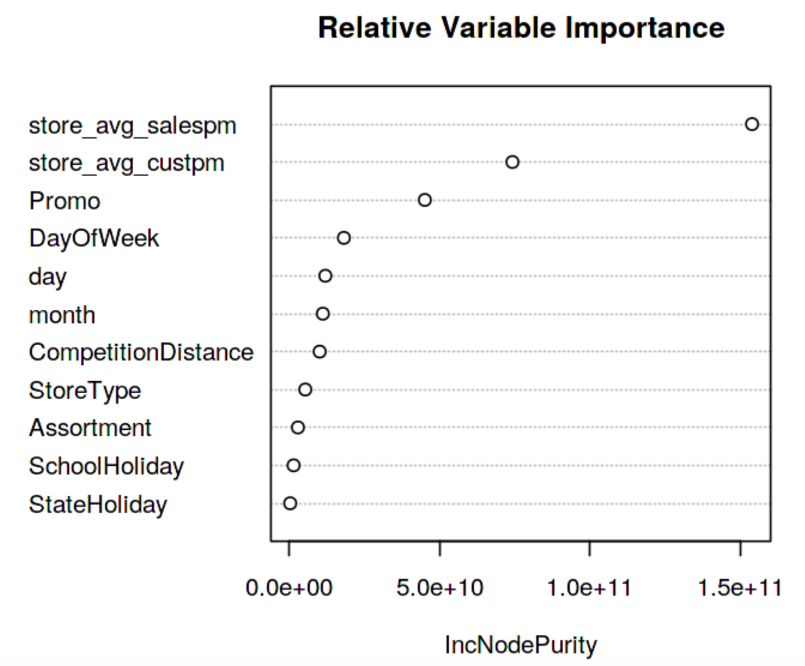
Feature.names contains the list of retained "candidate" variables. "Mtry" is the number of features used to build each tree (as mentioned in previous sections, it should ideally be p/3 where p is the total number of features in the dataset). "Ntree" is the total number of trees to be grown. "Do.trace" is for making the output appear on the screen during runtime. It outputs the MSE and %y explained by each tree. There are many other parameters which can be specified. You can read about them here. Once the model is built, you can check the model statistics like MSE, %variance explained, variable importance by simply printing out the model or using in-built functions. We will discuss this in detail for our dataset in the next section.

Note :

- *Make sure to set a seed to ensure reproducibility of the results*
- *We used a sample of 50,000 rows of our dataset due to hardware resource constraints, but it can be easily run on a higher configuration laptop or on cloud.*

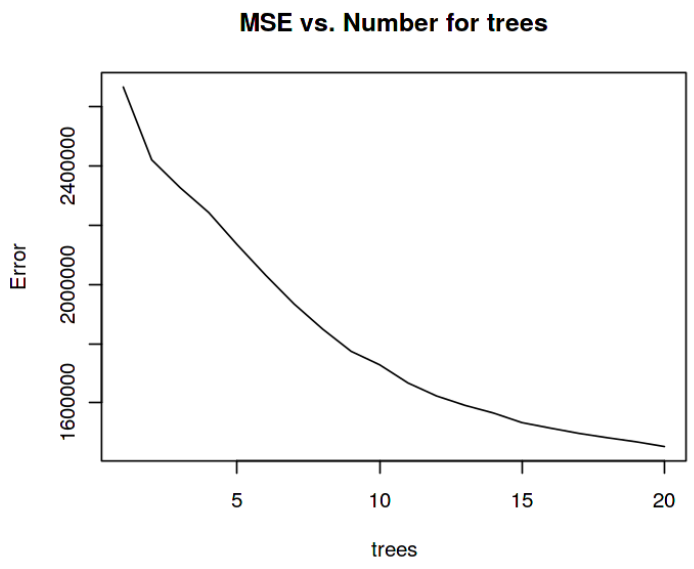
Results

To assess the model performance, we first looked at the percent variance of the response variable which is explained by the model. We achieved a pretty good value of ~85%. R's Random Forest package, as with other tree-based regression models, also provides a way to check the Variable Importance in the model. It is measured in terms of IncNodePurity (the Information Gain concept discussed in previous sections). It implies more useful variables achieve higher increase in Node Purities, that is to find a split which has a high inter-node variance and a small intra-node variance. This can help in building a better model, and allows for easier interpretation of results. Below is a plot showing the Variable Importance:

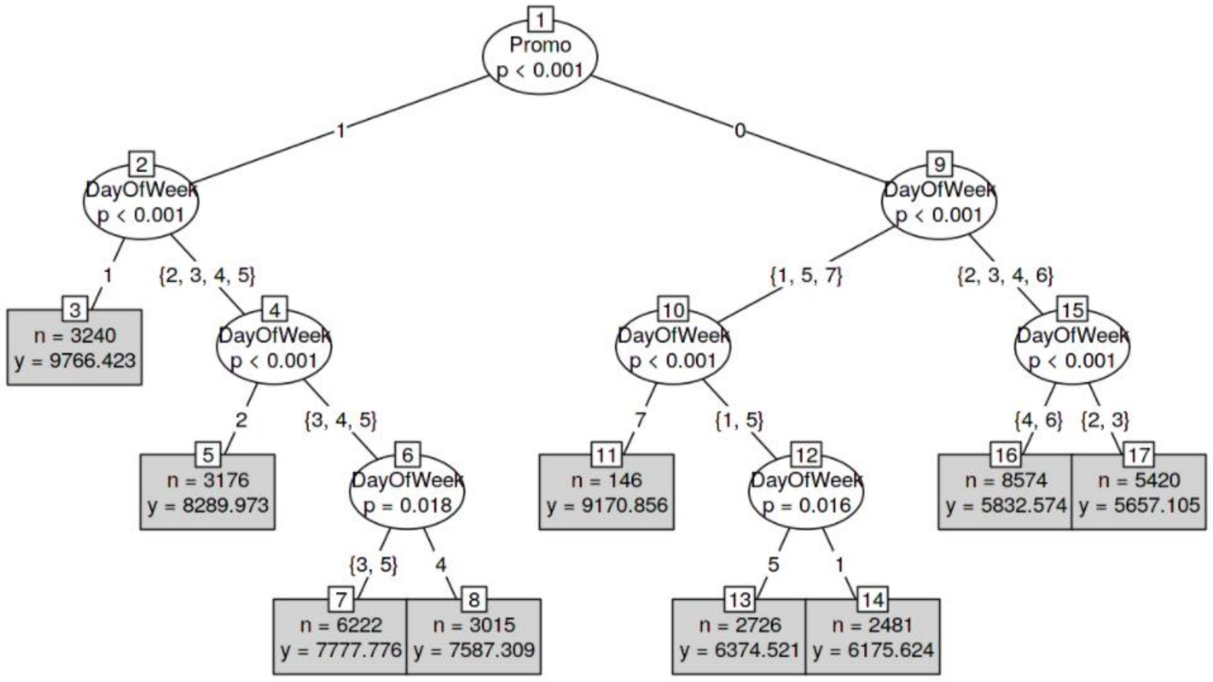


We can clearly see that store average sales has the highest importance among all variables which makes sense as historical sales does play an important part in predicting the future sales. Presence of a promotion is also significant, but variables like School and State holidays do not provide much insight.

Next we look at the Mean Squared Error (MSE) for the incremental forest and see how the MSE decreases exponentially with the increase in the number of trees.



We have now understood conceptually how random forest works. To demonstrate visually how the prediction works with just two features (Promo and Day of the week), here is a sample ensemble of trees:



Let us move on to the most important part of assessing the model's performance: cross-validation. For this model, we held-out 30% of the dataset for hold-out testing. We use the model which we fit on the 70% training sample to predict the sales from our test dataset.

```
pred <- predict(model_rf, test[,feature.names])
print("Test RMSE:");sqrt(mean((test$Sales-pred)^2))
print("MAPE:");mean(abs((test$Sales-pred)/test$Sales) * 100)
```

Once the predict function throws out the final predicted sales, we compare it against the actual sales values from the test dataset and calculate RMSE and MAPE to assess the prediction's accuracy. Both the metrics measure the deviation (or difference) of the predicted values from the actuals. Root Mean Square Error (RMSE) is the square root of the averaged errors or residuals. Mean Absolute Percent Error (MAPE) expresses it using the absolute difference instead of squared errors. For our test set, we achieved an RMSE of 582 and MAPE of 6.08%, which is very close to the training RMSE (578) and MAPE (6.04%). Strong out-of-sample performance indicates that the Random Forest, as expected, did not overfit on the training sample.

Further Improvements

The model has some room for improvement by applying various techniques already implemented in the machine learning packages which are readily available. We will discuss two such techniques (briefly as concepts) - Principal Component Analysis and Hyperparameter Tuning.

- **Principal Component Analysis** (PCA) is a very powerful technique used for dimension reduction purposes. This is very useful in cases where we have a large number of predictor variables to select from, i.e., when the number of predictors is higher than number of observations (e.g., when $p > N$. Note that this is not the case in our dataset). PCA transforms a number of (possibly) correlated variables into a smaller set of uncorrelated variables called principal components. The first principal component accounts for as much of the variability in the data as possible, and each succeeding component accounts for as much of the remaining variability as possible.

There are several function (packages) available in R - prcomp (stats), princomp (stats), PCA (FactoMineR).

- **Hyperparameter tuning** is another way to enhance the performance of the model. It is as simple as tuning a radio station by turning the knobs to get a better signal. While model parameters are learned during training (slope and intercept), hyperparameters must be set before training. In the case of Random Forest, these include the number of Decision Trees, number of features, maximum depth of the trees, etc. For tuning, we performed many iterations of the entire K-fold cross-validation process, each time using different model settings. We then compared all of the models and select the one with the least MSE. This sounds very tedious but R's MLR package provides a way to automate this process.

References

- An introduction to Statistical Learning by Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani - Chapter 8, Tree-Based Methods
- https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
- https://www.hackerearth.com/practice/machine-learning/machine-learning-algorithms/tutorial-random-forest-parameter-tuning-r/tutorial/
- https://www.quora.com/How-does-randomization-in-a-random-forest-work
- http://www.erogol.com/randomness-randomforests/