Machine Learning: Decision Trees in Retail

Victor Wright

November 2019

Abstract

In this paper, we first give a brief reasoning why we should use decision trees versus linear models such as linear regression or logistic regression. This reasoning is followed by a "toy" example where we grow a regression tree used to predict *Salary* contained in the *Hitters* data set, which is part of the *Introduction to Statistical Learning* package called *ISLR*, where a natural logarithm transformation has been applied to the variable and explain to the reader how to use the tree for prediction. We do this because the decision trees grown on Walmart stores data contain many more internal and terminal nodes. We include the toy example to help ensure that all readers will be able to understand the decision rules in the larger more complicated trees and use them to make predictions for Walmart store sales. In the next section, we explain how binary recursive splitting is used to grow regression and classification trees which is followed by an explanation of the data source and variables. After this section, we proceed to load the data and organize it by writing SQL queries. The final data matrix we obtain contains five *MarkDown* variables which happen to contain some null values. To deal with the missing data, we make use of the multivariate imputation by chained equations (MICE) algorithm and predictive mean matching to estimate the missing information in these variables. Once our data set is completed, we conduct data visualization, grow a regression tree, engineer a categorical response containing two levels, grow a classification tree, and make predictions for a new observation *X*. The paper is concluded with a section called *Regression Tree Improvement Approaches: Bagging and Random Forests* where bagging and random forests are discussed and applied to *only* our regression tree.

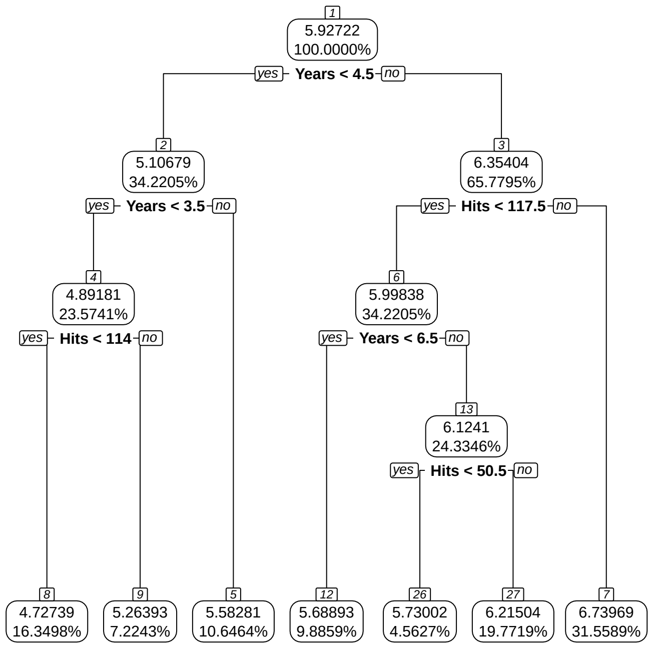
A Discussion of Decision Trees

In *Machine Learning: Learning from Data Using Linear Regression* and *Machine Learning: Using the Logistic Regression Model to Predict Coronary Heart Disease*, we provided details about regression and classification problems. When faced with a regression problem, our objective is to model a continuous dependent variable. On the other hand, in classification problems, our objective is to model a response variable whose values represent categories with one or more independent variables. However, the models used in each project previously mentioned should be applied to data that behave in a certain way. Mainly, linear dependency between the dependent variable and independent variable(s) is required for both linear and logistic regression. So what happens if there is nonlinearity between the dependent variable and independent variables in either a regression or classification problem? What learning approach should we use?

Decision trees are *non-parametric* models that will often be outshined by parametric models such as logistic or linear regression models if it appears that there is a linear or other functional relationship between the dependent and inpendent variable(s) because the functional form of the these parametric models are able to capture most of the linear trend or functional form that is clearly seen in the data [1, 2]. On the other hand, models like linear or logistic regression will be outshined by decision trees if the relationship between the dependent variable and independent variable(s) is *not* linear but exhibits complex patterns [1]. Additionally, decision trees are thought to have an advantage over other learning methods when applied to classification or regression problems because they can always be displayed graphically, better reflect decision-making, and are much easier to interpret compared to a linear model [1]. In fact, we think they are very similar to a *flowchart.* Before we build our own trees in this project, we grow a regression tree using a data set from the *R* package, *ISLR*, which contains data sets for the text *An Introduction to Statistical Learning with Applications in R* by Gareth *et al.*

The data set *Hitters* contains 322 observations of major league baseball players from the 1986 and 1987 seasons. The data set also contains 20 variables.

```
> library(ISLR)
> library(rpart)
> library(rpart.plot)
> attach(Hitters)
> a <- na.omit(Hitters)
> Salary.tree <- rpart(log(Salary, base = exp(1)) ~ Years +
+                      Hits, data = a)
> rpart.plot(Salary.tree, extra = "auto", type = 2, nn = TRUE,
+            varlen = 0, branch = 1, box.palette = 0,
+            yesno = 2, roundint = FALSE, digits = 6)
> detach(Hitters)
```



Above is the code and plot of the regression tree. In general, regression trees are fit using information contained in a data set where the *predictor space* is divided recursively and the response, Y, is typically modeled with a *unique prediction* constant for each division of the predictor space [1, 3, 4]. Additionally, the *mode* or *sample mean* is commonly used to for predicting the response value and the sample statistics are calculated from the training data that lie in the division of the predictor space the observations belongs to [1]. Finally, decision trees are also a graphical representation of *recursive binary splitting*, which is also called *binary recursive partioning* used to segment the predictor space, that applies to regression and classification trees when fitting either type of model [1, 4, 5]. We next explain the major components of the regression tree.

The “bottom” of the tree is where the decision tree’s *leaves* are found, which are also called *terminal nodes*, and when a decision tree’s leaves are found at it’s bottom, the decision tree is drawn upside down [1]. Decision trees also have *internal nodes* which are locations within the tree that indicate how the predictor space has been divided [1]. Finally, the solid black lines are the decision tree’s *branches* which the join the the trees nodes together [1]. We next describe how to use the regression tree.

The regression tree we produced has *seven terminal nodes*. Remember, we have applied a natural logarithm transform to the values in *Salary*. The predictor space has been divided into *prediction regions* in the following way. A $\rightarrow$

means *go to* that node. The node number is found in a rectagular box at the top of the internal and terminal nodes, the percentage indicates the proportion of training observations that meet the conditions of the node, the last number in the node is the prediction constant:

- Start with the variable *Years*. If *Years* < 4.5, $\rightarrow$ node 2. If *Years* < 3.5 $\rightarrow$ node 4. If *Hits* < 114, $\rightarrow$ node 8 and predict 4.7. This is prediction region P_1
- Start with the variable *Years*. If *Years* < 4.5, $\rightarrow$ node 2. If *Years*, < 3.5 $\rightarrow$ node 4. If *Hits* ≥ 114, $\rightarrow$ node 9 and predict 5.3. This is prediction region P_2
- Start with the variable *Years*. If *Years* < 4.5, $\rightarrow$ node 2. If *Years* ≥ 3.5, $\rightarrow$ node 5 and predict 5.6. This is prediction region P_3
- Start with the variable *Years*. If *Years* ≥ 4.5, $\rightarrow$ node 3. If *Hits* < 117.5, $\rightarrow$ node 6. If *Years* < 6.5 $\rightarrow$ node 12 and predict 5.7. This is prediction region P_4
- Start with the variable *Years*. If *Years* ≥ 4.5, $\rightarrow$ node 3. If *Hits* < 117.5, $\rightarrow$ node 6. If *Years* ≥ 6.5, $\rightarrow$ node 13. If *Hits* < 50.5, $\rightarrow$ node 26 and predict 5.7. This is prediction region P_5
- Start with the variable *Years*. If *Years* ≥ 4.5, $\rightarrow$ node 3. If *Hits* < 117.5, $\rightarrow$ node 6. If *Years* ≥ 6.5 $\rightarrow$ node 13. If *Hits* ≥ 50.5,$\rightarrow$ node 27 and predict 6.2. This is prediction region P_6
- Start with the variable *Years*. If *Years* ≥ 4.5, $\rightarrow$ node 3. If*Hits* ≥ 117.5 $\rightarrow$ node 7 and predict 6.7. This is prediction region P_7

. Note, *Years* and *Hits* are discrete values. The tree sees them as continuous. A function that represents the regression tree is

$$\begin{aligned}\hat{f}(X) = \quad & 4.7 * I(X \in P_1) \;+\; 5.3 * I(X \in P_2) \;+\; 5.6 * I(X \in P_3) \\ & + 5.7 * I(X \in P_4) \;+\; 5.7 * I(X \in P_5) \;+\; 6.2 * I(X \in P_6) \\ & + 6.7 * I(X \in P_7)\end{aligned} \tag{1}$$

where I is an indicator variable that takes on values of either 0 or 1. $I = 1$ when an observation X belongs to that prediction region [1, 4]. $I = 0$ if it doesn't.

Recursive Binary Spliting for Decision Trees

When "growing" regression trees, the goal is to minimize

$$RSS = \sum_{K=1}^{J} \sum_{i \in P_K} (y_i - \hat{y}_{P_K})^2 \tag{2}$$

where *RSS* is the *residual sum of squares* and the observations that fall into prediction region P_K are used to compute the statistic $\hat{y}_{P_K}$ which is typically either the *mean* or *mode* of the training observations that lie in P_K [1, 4]. However, dividing the predictor space into J *disjoint prediction regions* that would minimize (2) is not practical [1]. The points where the variables are divided, as well as the variable used in the division of the predictor space, need to be identified by the algorithm as well as the structure of the tree [4].

In order to minimize (2), we use *recursive binary splitting.* In the first step of recursive binary splitting, a *half-plane* is created in the predictor space which is given by

$$P_1 = \{X \mid X_r \leq t\} \quad \text{and} \quad P_2 = \{X \mid X_r > t\} \tag{3}$$

where all of the data is used for this decision and t is the division point of any available predictor variable $X_r \in X$ where $r = 1, 2, ..., p$ [1, 4]. Moreover, the function

$$\sum_{i: x_i \in P_1(r,t)} (y_i - \hat{y}_{P1})^2 \quad + \quad \sum_{i: x_i \in P_2(r,t)} (y_i - \hat{y}_{P2})^2 \tag{4}$$

is minimized by the choice of r and t [1]. After r and t are identified in this iteration of recursive binary splitting, the next iteration attempts to further divide the predictor space into further prediction regions by creating another half-plane using only *one* of the prediction regions that was obtained in the first iteration [1]. This occurs if the RSS would attain a smaller value by choosing a new r and t that define the new half-plane [1]. Different *ending criteria* may be selected when building a regression tree with recursive binary splitting, and when those criteria are reached, the iterative process ceases yielding a regression tree whose $\hat{y}_{P_u}$, for $u = 1, 2, ..., v$ prediction regions, minimize (2) [1].

If it happens to be the case that the elements of the dependent variable are discrete values, $u = 1, 2, ..., U$, each integer representing a unique category, then the dependent variable contains *classification outcomes* and that can be predicted with a *classification tree* rather than a regression tree [1, 4]. Additionally, recursive binary splitting is also used to grow classification trees. Let P_m be a division of the predictor space for $m = 1, 2, ..., M$ possible divisions containing $i = 1, 2, ..., T_m$ observations of class u. The *proportion* of class u observations in P_m is given by

$$\hat{p}_{P_m,u} = \frac{1}{T_m} \sum_{i=1}^{T_m} I(y_i = u) \tag{5}$$

[4]. Then, for any $X \in P_m$, we compute (5) for all classes u and label X as class u^* after identifying the largest computed $\hat{p}_{P_m,u} \in P_m$ [4]. Note, the *majority class* in P_m is u^* [4]. The splitting criteria is based on the meausrement of

pure the nodes in the classification tree [1]. One way to assess node purity is by computing the function

$$G = \sum_{i=1}^{T_m} \hat{p}_{P_m,u} * (1 - \hat{p}_{P_m,u}) \tag{6}$$

which is called the Gini index [1]. In classification trees, we seek a predictor and cut point in each split that minimizes (6) [1].

Walmart Stores Data: The Data Source, an Explanation of Variables, SQL Joins, and Multivariate Imputation by Chained Equations, and Data Visualization

The Data Source and an Explanation of Variables

We obtained three data sets from https://www.kaggle.com/manjeetsingh/retaildataset. They are *sales*, *features*, and *stores*. Each data set contains a number of features/variables but, they exhibit varying dimensions. $\dim(\textit{sales}) = 421,570 \times 5$, $\dim(features) = 8,190 \times 12$, and $\dim(\textit{stores}) = 45 \times 3$. The *sales* data set contains the features *Store*, *Type*, and *Size*. Furthermore, the data in *stores* is *anonymized* for the forty-five stores where the feature *Store* indicates the store number, the feature *Size* is the store size (unit of measurement assumed to be in square feet), and *Type* represents the type of store [6].

The data set *sales* contains observations taken from February 5th, 2010 to November 1st, 2012 [6]. The features this data set has are *Store*, which is common to the stores data set, *Dept* which is a identifier for the deparment in each *Store*, *Date* is the date the observation was made, $Weekly_Sales$ is the weekly sales in a certain department in a certain store, and *IsHoliday* indicates if the week the observation was made was a holiday week [6].

The *features* data set contains the variables *Store*, *Dept*, *IsHoliday*. It also contains the variables *Temperature* which is the average temperature of the region where the store is located at the time of observation (assuming the average is computed over a seven-day forecast), $Fuel_Price$ which is the price of fuel in the region where the store is located at the time of the observation, *CPI* is the consumer price index, and *Unemployment* is the unemployment rate [6]. The features data set also contains *MarkDown* variables 1 - 5 that indicate markdown sales in dollars. However, the data in these variables is also anonymized and only available after November 2011 [6]. Finally, these data sets also appear at https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data and are part of the *Walmart Recruiting - Sales Forecasting* Kaggle Data competition that we did not participate in; we only utilize the Walmart data sets in this project [7]. In this section, we join the available information with the help of

SQL and R. Finally, the target variable we want to model is *TOTAL* which is *total weekly sales* for Walmart stores. *NOT* weekly sales for the individual departments. We provide code below and comment when necessary.

SQL Joins

```
> # It is our objective to clean and manipulate the data we have
> # obtained from Kaggle and produce a data set containing relevant
> # features of Walmart stores. We know that the data sets
> # obtained contain information on forty-five Walmart stores.
>
> stores <- read.csv("stores data-set.csv", header = TRUE, sep = ",")
> sales <- read.csv("sales data-set.csv", header = TRUE, sep = ",")
> features <- read.csv("Features data set.csv", header = TRUE, sep = ",")
> #
> # The data has been loaded into R. We next use the sqldf() function,
> # which is part of the sqldf package, to create a data set containing
> # relevant information we wish to use in the construction of our
> # decision trees in this project. SQL script is shown below.
> #
>
> library(sqldf)
> join_1 <- sqldf(x = "SELECT DISTINCT sales.*,
+                      features.Temperature, features.Fuel_Price,
+                      features.MarkDown1, features.MarkDown2,
+                      features.MarkDown3, features.MarkDown4,
+                      features.MarkDown5, features.CPI,
+                      features.Unemployment FROM sales
+                      INNER JOIN features ON
+                      sales.Date = features.Date
+                      AND sales.Store = features.Store
+                      AND sales.IsHoliday = features.IsHoliday")
> #
> # The above query eliminates duplicates and joins the sales and features
> # tables together. We next join the stores table to the table created
> # above.
> #
>
> join_2  <- sqldf(x = "SELECT join_1.*,
+                       stores.Type, stores.Size FROM join_1
+                       stores.Store = join_1.Store")
> #
> # The final query to create our data set is seen below
> #
>
> final.join <- sqldf(x = "SELECT SUM(Weekly_Sales) AS TOTAL,
+                              Store, Date, IsHoliday AS Holiday, Temperature,
+                              Fuel_Price, MarkDown1, MarkDown2, MarkDown3,
+                              MarkDown4, MarkDown5, CPI, Unemployment, Type,
+                              Size FROM join_2 GROUP BY Date, Store")
>
> #
> # where TOTAL is a conditional sum that sums the Weekly_Sales of each
> # department for Store i, i = 1, 2, ..., 45, in a single week. Thus,
> # creating the target variable TOTAL.
> #
>
>
```

Multivariate Imputation by Chained Equations

The *final.join* data set contains all of the available variables contained in the data sets that we obtained from Kaggle. However, the *MarkDown* variables are still incomplete and contain missing values. In this section, we discuss how we deal with these null values before modeling *TOTAL* with a regression tree. We estimate these missing values by the method of *multivariate imputation by chained equations*.

A popular approach when dealing with variables that contain null elements is multiple imputation [8]. Multivariate imputation by chained equations, also known as *fully conditional specification*, can be used to impute data that are multivariate and contain null values [8]. However, when imputing missing multivariate, some problems that may be faced are:

- High correlation is present among variables
- The variables may be of different types. *E.g.*, continuous versus binary
- Null values, as well as collinearity, may be introduced when the data set contains few observations and a large number of variables

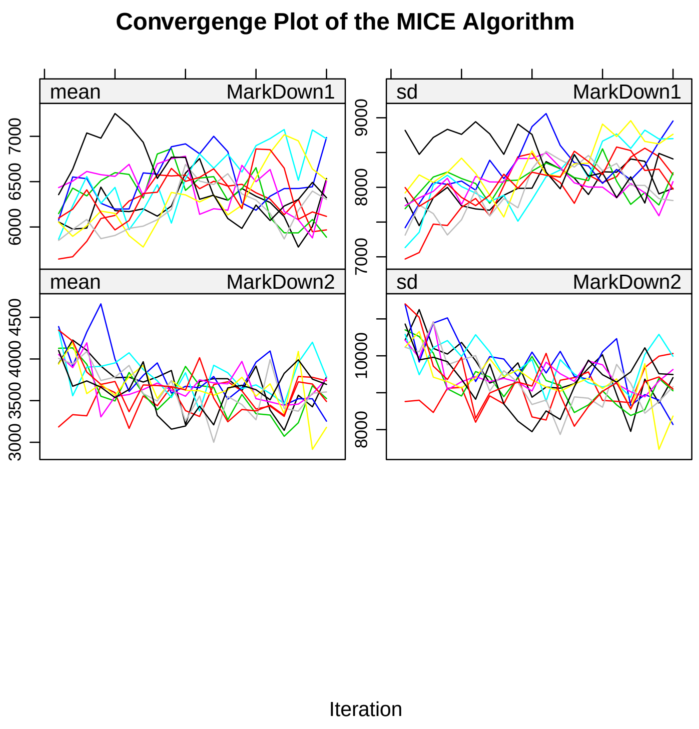
and more [8]. On the other hand, when carrying out multiple imputation to fill in null values of a variable, it is wise to analyze the obtained imputed values and conclude wether or not they make sense [8]. In other words, the known values of the variable and imputed values in the variable should not vary wildly [8]. We impute the missing values for *MarkDown* variables 1-5 with *R* below. We comment in the code chunks when necessary.

```
> cor(na.omit(final.join[, -c(1,2,3,4,14,15)]))

             Temperature  Fuel_Price     MarkDown1     MarkDown2   MarkDown3
Temperature   1.00000000  0.24721805  1.700516e-02 -3.332429e-01 -0.08860119
Fuel_Price    0.24721805  1.00000000  8.492295e-02 -2.424897e-01 -0.09261354
MarkDown1     0.01700516  0.08492295  1.000000e+00  8.122599e-05 -0.12801495
```

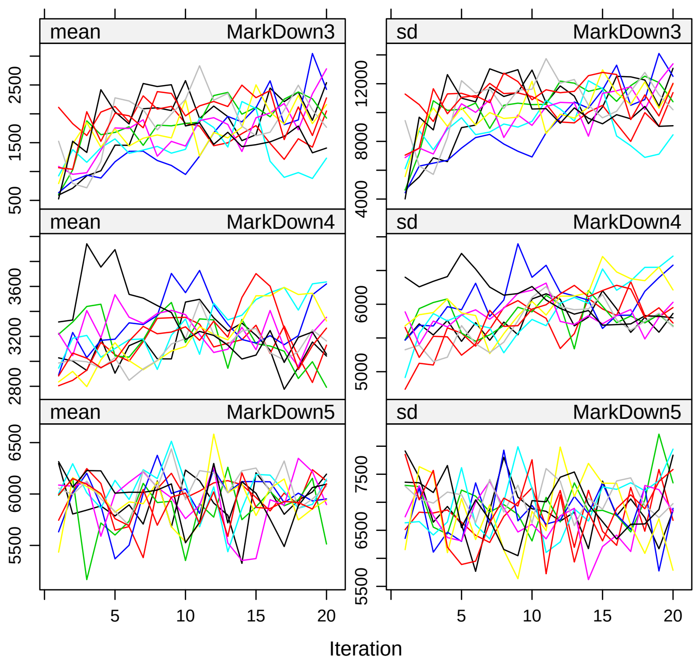
```
MarkDown2    -0.33324293 -0.24248973  8.122599e-05  1.000000e+00 -0.05020980
MarkDown3    -0.08860119 -0.09261354 -1.280150e-01 -5.020980e-02  1.00000000
MarkDown4    -0.05837823 -0.02478564  8.289797e-01 -1.626476e-02 -0.07858316
MarkDown5     0.02370073 -0.13606993  1.091244e-01 -2.129169e-02 -0.04174025
CPI           0.20313773 -0.36955424 -4.889678e-02 -3.813600e-02 -0.02504159
Unemployment -0.01189622  0.28038892  6.451228e-02  1.899580e-02  0.01418730
               MarkDown4   MarkDown5         CPI Unemployment
Temperature  -0.05837823  0.02370073  0.20313773  -0.01189622
Fuel_Price   -0.02478564 -0.13606993 -0.36955424   0.28038892
MarkDown1     0.82897973  0.10912443 -0.04889678   0.06451228
MarkDown2    -0.01626476 -0.02129169 -0.03813600   0.01899580
MarkDown3    -0.07858316 -0.04174025 -0.02504159   0.01418730
MarkDown4     1.00000000  0.10235131 -0.04314642   0.01860101
MarkDown5     0.10235131  1.00000000  0.07580175   0.00244394
CPI          -0.04314642  0.07580175  1.00000000  -0.30020470
Unemployment  0.01860101  0.00244394 -0.30020470   1.00000000
```

. It can be seen from the correlation matrix that correlation between the majority of variables is low. However, there are some exceptions of moderate and high correlation.

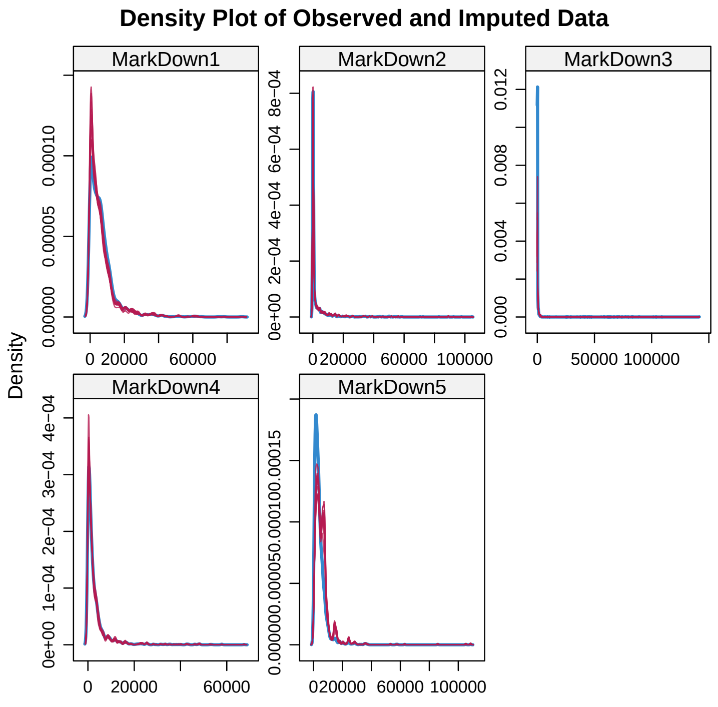
```
> #
> # The initial iteration for the MICE Algorithm is
> #
>
> library(mice)
> ini <- mice(data = final.join, print = FALSE, maxit = 0)
> preds <- ini$predictorMatrix
> preds[, "Store"] <- 0
> preds[,"TOTAL"] <- 0
> preds[,"Date"] <- 0
> #
> # which tells R not to use TOTAL, Store, and Date for imputations because #
> # we have no reason to believe that TOTAL, Store that represents the
> # store number, and Date have anything to do with how the stores
> # determine their markdown prices. Imputation is obtained with
> #
>
> imps <- mice(data = final.join, seed = 1203, print = FALSE,
+              method = c("","","","","","","pmm","pmm","pmm","pmm",
+                         "pmm","","","",""), m = 10, maxit = 20,
+              predictorMatrix  = preds)
>
> #
> # . In the method vector, "" entries mean that no imputation is needed
> # for that variable because it is complete. "pmm" means that the
> # imputation model we want to use is predictive mean matching with
> # maxit = twenty iterations and m = ten imputed data sets.
> #

> #
> # The null values in the final.joing data matrix are then filled by
> # the excuting the commands seen below.
> #
>
> library(sjmisc)
> Walmart_Stores <- merge_imputations(dat = final.join, imp = imps,
+                                     ori = final.join)
> Walmart_Stores <- Walmart_Stores[,-c(7,8,9,10,11)]
> colnames(Walmart_Stores) <- c("TOTAL","Store","Date","Holiday",
+                               "Temperature","Fuel_Price", "CPI",
+                               "Unemployment","Type","Size",
+                               "MarkDown1","MarkDown2","MarkDown3",
+                               "MarkDown4","MarkDown5")
> attach(Walmart_Stores)
> plot(imps, c("MarkDown1", "MarkDown2"),
+       main = "Convergenge Plot of the MICE Algorithm")
>
>
>
>
```



Convergenge Plot of the MICE Algorithm



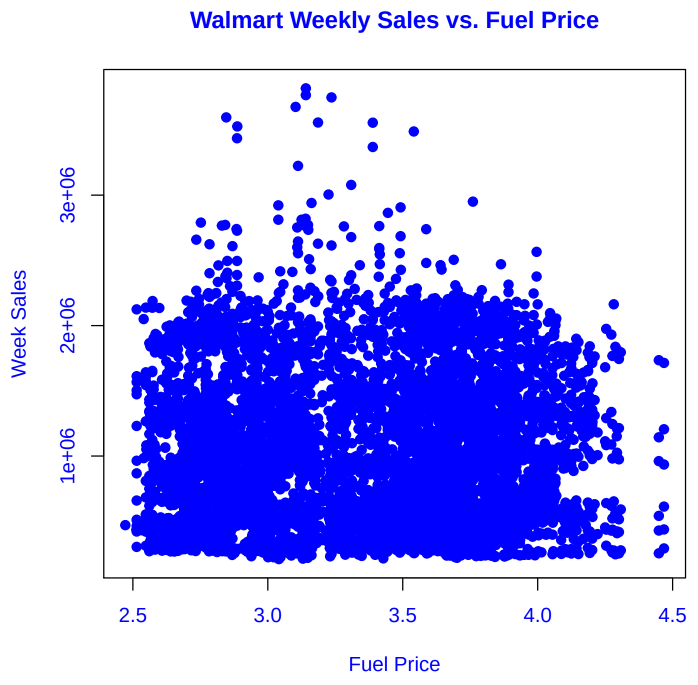
```
> library(lattice)
> densityplot(imps, main = "Density Plot of Observed and Imputed Data")
```



The convergence plots show that the MICE algorithm has converged. The density plot of observed and imputed data for the *MarkDown* variables shows the closeness of the imputed data to the known data in the *MarkDown* variables. The blue line is the density curve of the known *MarkDown* values and the red curve is the density curve of the imputations.

Data Visualization

The *Walmart Stores* data set is a complete data set where the null values of the *MarkDown* variables have been imputed by the MICE algorithm and predictive mean matching model. $\dim(WalmartStores) = 6435 \times 15$. Before fitting the regression tree to the data, we produce a *scatter plot* to examine the relationship between *TOTAL* and *Fuel Price*.



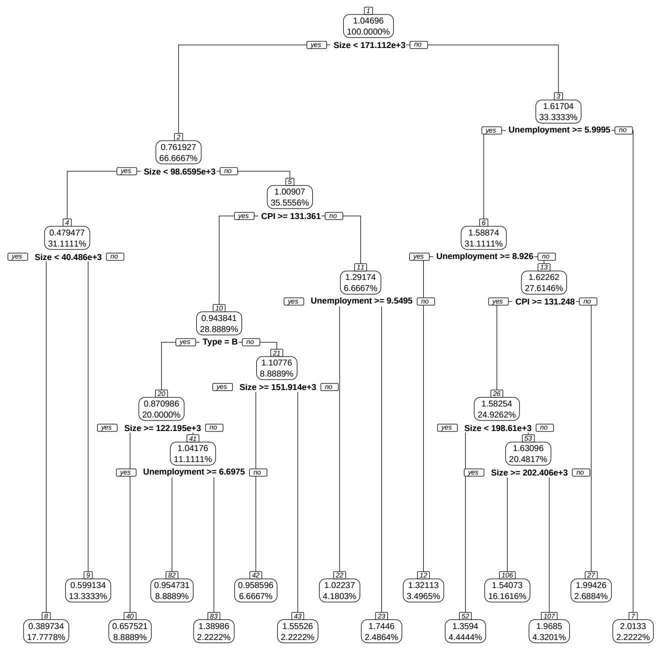
No functional relationship between $TOTAL$ and $Fuel\ Price$ appears in the data. However, it does appear that $TOTAL$ becomes more variable when $Fuel\ Price$ $\in (2.75, 3.75)$ for only a few cases. In fact, the relationship between $TOTAL$ and all of the available continuous independent variables in the *Walmart Stores* data set is highly complex. Additionally, if one were to make scatterplots to visualize the dependence between $TOTAL$ and the rest of the independent variables, you would find that there is no functional form in the data. Thus, it is makes sense to grow a regression tree to model $TOTAL$.

Modeling Walmart Weekly Sales with Regression Trees

In this section we grow two decision trees. One will model $TOTAL$ for the population of Walmart stores which the sample was taken from. The second will model a binary response we create later.

The regression tree that models $TOTAL$ for all forty-five Walmart stores is grown with the R code below.

```
> library(rpart)
> library(rpart.plot)
> control <- rpart.control(minsplit = 45, minbucket = 143, maxdepth = 29)
> Walmart.tree <- rpart((TOTAL/1000000) ~.,
+                       data = Walmart_Stores[,-c(2,3)],
+                       control = control, method = "anova")
> rpart.plot(left = TRUE , x = Walmart.tree, type = 2, extra = "auto", digits = 6, varlen =
+            yesno = 2, fallen.leaves = TRUE,compress = TRUE,
+            ycompress = TRUE, roundint = FALSE)
```



We chose the *minsplit* argument to take on a value of forty-five so that in each splitting rule it is possible that information from each store is considered in the split. Similarly, we chose the *minbucket* argument to take on a value of one hundred and forty-three so that it is possible that information from each observed week is used to calculate the prediction constant. We see that the algorithm has included *Size* in the regression tree. The $e + 3$ value in the decision rules mean to multiply the constant to the left of $e + 3$ by 10^3. Next, we estimate the training MSE and test MSE for this regression tree. The test MSE will be estimated using a 50-50 split.

```
> training.predictions <- predict(Walmart.tree,
+                                 newdata = Walmart_Stores[,-c(2,3)])
> trai.MSE <- mean(((Walmart_Stores[,1]/1000000) - training.predictions)^2)
> trai.MSE

[1] 0.05175588
```

```
> set.seed(31)
> train <- sample(x = 6435, size = 3218, replace = FALSE)
> test <- Walmart_Stores[-train,]
> test.tree <- rpart((TOTAL/1000000) ~., data = Walmart_Stores[,-c(2,3)],
+                    subset = train, control = control, method = "anova")
> test.tree.preds <- rpart.predict(object = test.tree, newdata = test,
+                                  type = "vector")
> test.MSE <- mean(((test[,1]/1000000) - test.tree.preds)^2)
> test.MSE

[1] 0.09027282

>
```

So the average error the regression tree makes on the entire data set is $\$1,000,000*\sqrt{0.05175588} = \$227,499$ and the average error on the test observations, using a 50-50 split, is $\$1,000,000 * \sqrt{0.09027282} = \$300,454$.

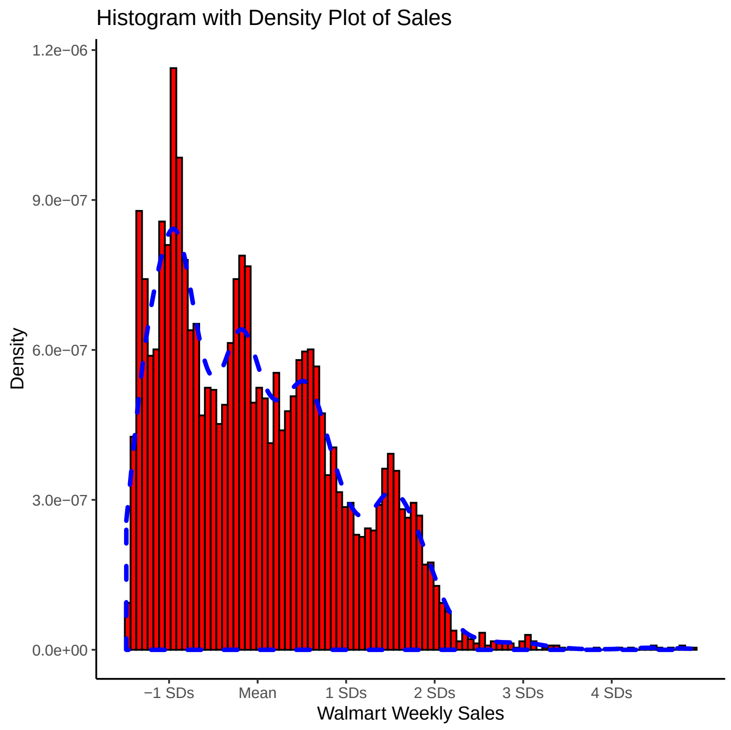
An Application of Classfication Trees

In this section, we examine the distribution of $TOTAL$ across all stores and apply *classification trees* to model a business outcome using the available independent variables. Similar to the target variable $TOTAL$ we created in the subsection of this paper called *SQL Joins*, we create a target variable based on the distribution of $TOTAL$. In order to gain an understanding of the distribution of $TOTAL$ accross all Walmart stores in the population, we create a histogram of the data in this variable with a density curve. First, the summary statistics of the variable, as well as its standard deviation, are found below.

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 209986  553350  960746 1046965 1420159 3818686

[1] 564366.6
```

. The mean of $TOTAL$ is $\$1,046,065$ and its standard deviation is $\pm\$564,367$ per week. It may be of value to classify what values influence $TOTAL$ to lie in certain subsets of its *known* domain. For example, given a case $X = (X_1 = x_1, X_2 = x_2, ..., X_n = x_n)$ will $TOTAL$ fall within one *sample standard deviation* of the *sample mean*? To get an idea of *TOTAL's* distribution, the reader can view the distribution of the variable in a histogram below and code used to produce the image.

```
> library(ggplot2)
> standard.deviations <- seq(from = 482598.4, to=max(Walmart_Stores$TOTAL),
> ggplot(data = Walmart_Stores, mapping = aes(x = TOTAL)) +
+ geom_histogram(mapping = aes(y = ..density..), bins = 100,
+                fill = "red", color = "black") +
+ geom_density(mapping = aes(x = TOTAL), col = "blue", lwd = 1.25,
+              lty = "dashed") +
+ labs(x = "Walmart Weekly Sales", y = "Density",
+      title = "Histogram with Density Plot of Sales") +
+ theme_classic() +
+ theme(panel.grid.major.x = element_line(colour = "white"),
+       panel.grid.major.y = element_line(colour = "white")) +
+ scale_x_continuous(breaks = standard.deviations,
+                    labels = c("-1 SDs", "Mean", "1 SDs", "2 SDs",
+                               "3 SDs", "4 SDs"))
```



where the blue dashed line is the population density curve of $TOTAL$ and the x - *axis* is labeled in terms of *standard deviations from the mean* of $TOTAL$. For example, the label *-1 SDs* indicates that weekly sales are *one* sample standard deviation below the sample mean of $TOTAL$ and the label *3 SDs* indicates weekly sales that are *three* sample standard deviations above the sample mean of $TOTAL$ and so on. Next, we define our target variable to be $Class.TOTAL$ with the following levels:

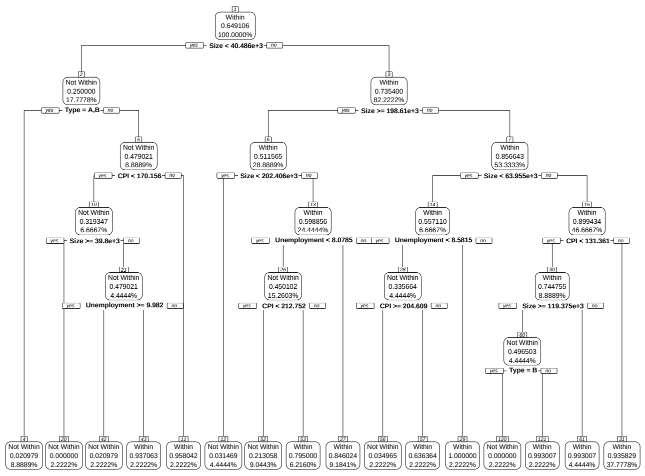
- $Class.TOTAL = Within$ when $TOTAL \in [\$482,598,\$1,611,332]$
- $Class.TOTAL = Not\ Within$ when $TOTAL \notin [\$482,598,\$1,611,332]$

. *Class.TOTAL* is created with the following R code.

```
> Class.TOTAL <- ifelse( TOTAL >= mean(TOTAL) - sqrt(var(TOTAL))
+                        & TOTAL <= mean(TOTAL) + sqrt(var(TOTAL)),
+                        "Within", "Not Within")
> Class.Walmart <- cbind(Class.TOTAL = Class.TOTAL, Walmart_Stores)
```

. The classification tree can be found below.

```
> control <- rpart.control(minsplit = 45, minbucket = 143, maxdepth = 29)
> Class.tree <- rpart( Class.TOTAL ~.,
+                       data = Class.Walmart[,-c(2,3,4)],
+                       control = control, method = "class")
> rpart.plot(x = Class.tree, left = TRUE, type = 2, extra = "auto",
+            digits = 6, varlen = 0, box.palette = 0, branch = 1,
+            nn = TRUE, yesno = 2, fallen.leaves = TRUE,compress = TRUE,
+            ycompress = TRUE, roundint = FALSE)
```


1 Within 0.649106 100.0000%
yes Size < 40.486e+3 no
2 Not Within 0.250000 17.7778%
yes Type = A,B no
3 Within 0.735400 82.2222%
yes Size >= 198.61e+3 no
5 Not Within 0.479021 8.8889%
yes CPI < 170.156 no
6 Within 0.511565 28.8889%
yes Size < 202.406e+3 no
7 Within 0.856643 53.3333%
yes Size < 63.955e+3 no
10 Not Within 0.319347 6.6667%
yes Size >= 39.8e+3 no
13 Within 0.598856 24.4444%
yes Unemployment < 8.0785 no
14 Within 0.557110 6.6667%
yes Unemployment < 8.5815 no
15 Within 0.899434 46.6667%
yes CPI < 131.361 no
21 Not Within 0.479021 4.4444%
yes Unemployment >= 9.982 no
26 Not Within 0.450102 15.2603%
yes CPI < 212.752 no
28 Not Within 0.335664 4.4444%
yes CPI >= 204.609 no
30 Within 0.744755 8.8889%
yes Size >= 119.375e+3 no
60 Not Within 0.496503 4.4444%
yes Type = B no
4 Not Within 0.020979 8.8889%
20 Not Within 0.000000 2.2222%
42 Not Within 0.020979 2.2222%
43 Within 0.937063 2.2222%
11 Within 0.958042 2.2222%
12 Not Within 0.031469 4.4444%
52 Not Within 0.213058 9.0443%
53 Within 0.795000 6.2160%
27 Within 0.846024 9.1841%
56 Not Within 0.034965 2.2222%
57 Within 0.636364 2.2222%
29 Within 1.000000 2.2222%
120 Not Within 0.000000 2.2222%
121 Within 0.993007 2.2222%
61 Within 0.993007 4.4444%
31 Within 0.935829 37.7778%

We next produce confusion matrices for this classification tree in order to gain an understanding of the learning method's predictive power on the training and test sets.

```
> #
> # The code below is used to produce a confusion matrix using all of
> # the data. That is, the full training set.
> #
>
> class.training.preds <- predict(object = Class.tree,
+                                 newdata = Class.Walmart[,-c(2,3,4)],
+                                 type = "class")
> confusion.matrix.1 <- table(class.training.preds, Class.TOTAL)
> confusion.matrix.1

                    Class.TOTAL
class.training.preds Not Within Within
          Not Within       1859    153
          Within            399   4024

> #
> # Next, we produce a confusion matrix for this decision tree using the
> # validation set approach. The split is 50-50.
> #
>
> set.seed(36)
> validation.train <- sample(x = 6435, size = 3218, replace = F)
> validation.test <- Class.Walmart[-validation.train,]
> validation.tree <- rpart(Class.TOTAL ~.,
+                        data = Class.Walmart[,-c(2,3,4)],
+                        control = control, method = "class",
+                        subset = validation.train)
> validation.preds <- predict(object = Class.tree,
+                              newdata = validation.test,
+                              type = "class")
> confusion.matrix.2 <- table(validation.preds, validation.test[,1])
> confusion.matrix.2

validation.preds Not Within Within
      Not Within        929     80
      Within            190   2018

>
```

The sensitivity of the classification tree on all of the training data is $100 * \frac{4024}{4024+153}\% = 96.34\%$, the specificity is $100*\frac{1859}{1859+399}\% = 81.25\%$, and the overall classification accuracy on the training set is $100*\frac{4024+1859}{6435}\% = 91.42\%$. The estimated test sensitivity of the classification tree is $100*\frac{2018}{2018+80}\% = 96.19\%$, the estimated test specificity is $100*\frac{929}{929+190}\% = 83.02\%$ and the estimated overall test accuracy is $100*\frac{929+2018}{3217} = 91.61\%$. So we can conclude that this classification tree is very effective at labeling the categorical response *Class.TOTAL* correctly. To conclude this section, we predict *TOTAL* for a observation X and then classify whether $Class.TOTAL = Within$ or not.

Suppose for one week, and *Store* 17, the week is not a holiday week, the average temperature in the region is 71.2 degrees farenheit, the fuel price in the region is \$3.32 per gallon, *MarkDown1* dollars are \$27,835, *MarkDown2* dollars are \$34,065, *MarkDown3* dollars are \$31,223, *MarkDown4* dollars are \$51,189, *MarkDown5* dollars are \$83,437, the consumer price index at the time of observation is 215.2328, and the the unemployment rate is 5.14%. So the vector we want to predict and classify is constructed by

```
> pred.vec <- data.frame(Store = 17, Holiday = 0, Temperature = 71.2,
+                        Fuel_Price = 3.32, MarkDown1 = 27835,
+                        MarkDown2 = 34065, MarkDown3 = 31223,
+                        MarkDown4 = 51189, MarkDown5 = 83437,
+                        CPI = 215.2328, Unemployment = 5.14,
+                        Type = "B", Size = 93188)
> #
> # The predicted value of TOTAL is
> #
>
> 1000000 * rpart.predict(object = Walmart.tree,
+                        newdata = pred.vec,
+                        type = "vector")

       1
599133.7

> #
> # . We next classify the vector.
> #
>
> rpart.predict(object = Class.tree, newdata = pred.vec,
+               type = "class")

     1
Within
Levels: Not Within Within
```

. One can also use the decision rules seen in the plots of the regression and classification trees to obtain the same prediction and classification of this particular X.

Regression Tree Improvement Approaches: Bagging and Random Forest

Bagging

In this section, we briefly describe bagging and random forests which are methods that can be used to increase prediction accuracy of decision trees. We then apply these to *only* our regression tree. We do not cover pruning because the regression tree we grew yielded the lowest test error compared to pruned trees. These are methods that can be used to improve prediction accuracy by reducing the variance of the model.

Variances of statistical learning models can be minimized by the technique of *bagging*, also known as *bootstrap aggregation*, which is similar to the bootstrap resampling method we used to simulate linear regression parameter estimates in *Machine Learning: Learning from Data Using Linear Regression* [1]. The idea if bagging for regression trees is to use the original training set that we have and generate a large number of bootstrapped training data sets that are created by resampling from the original training data set [1]. Before applying bagging to our regression tree, we illustrate with a theoretical example how prediction accuracy can be improved by reducing variance.

Suppose we are studying a population that has a population mean μ and variance σ^2 and we are interested in the finding an estimate for the population mean. Next, suppose we collect a sample from that population and the observations are $S_1, S_2, ..., S_n$. Further, S_i are independent random variables with expectation $E[S_i] = \mu$ and variance $Var[S_i] = \sigma^2$. Then, it is well known that

$$E[\bar{S}] = \frac{1}{n}\sum_{i=1}^{n} E[S_i] = \mu$$

and

$$Var[\bar{S}] = \frac{1}{n^2}\sum_{i=1}^{n} Var[S_i] = \frac{\sigma^2}{n}$$

[9, 10]. Then, suppose we draw $T_1, T_2, ..., T_{n'}$ and compute $\bar{T}$. Further, suppose that n and $n^{'}$ are not very large. $10 \leq n \leq 20$ as well as $n^{'}$. It is very possible that $\bar{S}$ and $\bar{T}$ will vary greatly from each other. If it is found $\bar{S}$ and $\bar{T}$ vary greatly, it will make it difficult to decide which sample mean we should use to estimate μ even though we already claimed that $E[\bar{S}] = \mu$ and $E[\bar{T}] = \mu$. The way we could avoid a dilemma like this is by the following: draw $A_1, A_2, ..., A_{n''}$ from the population and let $n^{''} \to \infty$. Then $E[\bar{A}] = \mu$ and $Var[\bar{A}] \approx 0$ making $\bar{A}$ a much better estimator of μ compared to $\bar{S}$ and $\bar{T}$. It follows that *bagging* attempts reduce the variance of each prediction constant contained in a regression tree in a “similar” way. The bagged tree is created with the code seen below

```
> library(randomForest)
> bag.Walmart.tree <- randomForest((TOTAL/1000000) ~.,
+                                  data = Walmart_Stores[,-c(2,3)],
+                                  mtry = 12, ntree = 1000,
+                                  replace = TRUE,
+                                  importance = TRUE)
> bag.Walmart.tree
```

```
Call:
 randomForest(formula = (TOTAL/1e+06) ~ ., data = Walmart_Stores[,      -c(2, 3)], mtry = 12
               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 12

          Mean of squared residuals: 0.02019391
                    % Var explained: 93.66
```

and the form of the bagged regression tree is given by

$$B(X) = \frac{1}{L}\sum_{j=1}^{L} \hat{f}^{L}(X) \tag{7}$$

where the bagged tree we produced was obtained by generating one thousand regression trees, each grown on one of $j = 1, 2, ..., L$ bootstrapped data sets, and then the prediction constants of each tree were averaged as seen in (7). Finally, the test MSE for the bagged model using a 50-50 split is

```
> set.seed(14)
> bag.train <- sample(x = 6345, size = 3217, replace = F)
> bag.test <- Walmart_Stores[-bag.train, -c(2,3)]
> bag.preds <- predict(object = bag.Walmart.tree, newdata = bag.test,
+                     type = "response")
> bag.MSE <- mean(((bag.test[,1]/1000000) - bag.preds)^2)
> 1000000*sqrt(bag.MSE)

[1] 61782.38
```

or $\$61,782$.

Random Forests

In the previous section, we briefly described bagging and applied it to our application. *Random Forests* are an extension of bagging because training sets are still generated by bootstraping to grow the regression trees in a random forest [1]. Additionally, random forests provide an improvement over bagging regression trees [1]. The improvement comes from how many, and *which*, predictors are considered to construct the decision rules of the trees grown on each bootstrapped data set [1]. In the code we have shown for bagging, the *mtry* argument indicates the size of the subset of available predictors to randomly consider at each split in the bagged tree. This information can be found in the *R* documentation center for the random forest function. In the code that created our bagged model, we chose to use a value of $mtry = 12$ for bagging. The value for random forests is different.

Predictor subset size in the random selection of variables used to grow bagged trees and trees in a random forest creates the distinction between bagging and random forests [1]. In random forests of regression trees, the size of the subset of predictors considered at each split is $\frac{p}{3}$ of p possible predictors in each bootstrapped data set. This reduction of the size of the subset of predictors, compared to bagging which grows regression trees by considering all available independent variables at each split, makes the random forest algorithm superior to bagging because the trees in the random forest are not as highly correlated as bagged trees, the grown trees in the random forest are more likely to assume a larger number of different shapes thus yielding trees that are not as correlated as the bagged trees resulting in improved performance and accuracy when trees in the random forest are averaged [1]. Code for the random forest is provided below.

```
> Forest.trees <- randomForest((TOTAL/1000000) ~.,
+                                  data = Walmart_Stores[,-c(2,3)],
+                                  mtry = 4, ntree = 1000,
+                                  replace = TRUE,
+                                  importance = TRUE)
> Forest.trees

Call:
 randomForest(formula = (TOTAL/1e+06) ~ ., data = Walmart_Stores[,      -c(2, 3)], mtry = 4,
               Type of random forest: regression
                     Number of trees: 1000
No. of variables tried at each split: 4

          Mean of squared residuals: 0.01898609
                    % Var explained: 94.04

>
>
```

The estimated test MSE of the random forest on a 50-50 split is

```
> set.seed(18)
> forest.train <- sample(x = 6435, size = 3217, replace = F)
> forest.test <- Walmart_Stores[-forest.train, -c(2,3)]
> forest.preds <- predict(object = Forest.trees, newdata = forest.test,
+                          type = "response")
> forest.MSE <- mean(((forest.test[,1]/1000000) - forest.preds)^2)
> 1000000*sqrt(forest.MSE)

[1] 62690.89
```

or $62,691 which is a slight improvement compared to the bagged model.

It can be seen that that bagged model and random forest explain a significant amount of variation in the response. We conclude this section by using the bagged model and random forest to predict $TOTAL$ on the vector we previously created in the section called *Applications of Classification Trees.* The predictions from the bagged model and random forest are found below.

```
       1
771671.7
```

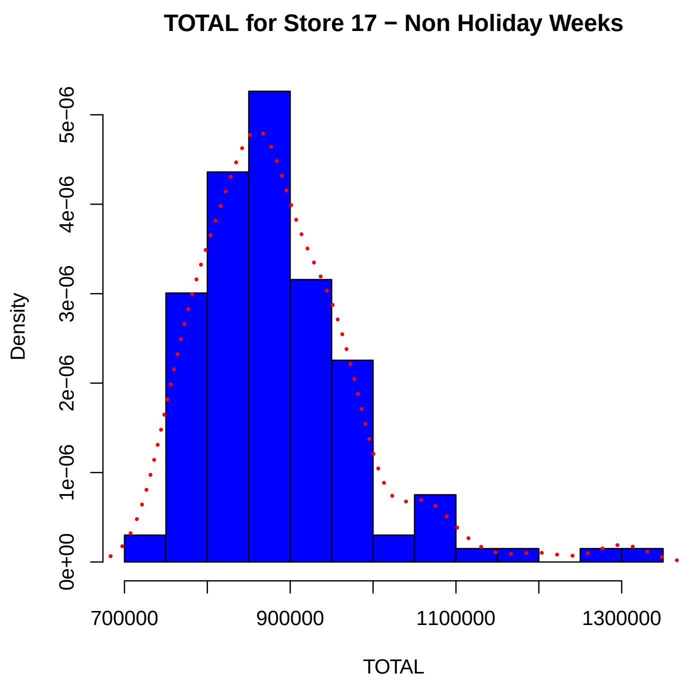
```
      1
1016684
```

It appears that the bagged model and random forest return very different predicted values. Therefore, it might be wise to compare *TOTAL* for some of the training data that has feature values close to *X* before making a decision on what to forecast. The cases can be returned using SQL.

```
> comparison.set <- sqldf(x = "SELECT TOTAL, Holiday, Temperature,
+                              Fuel_Price, MarkDown1, MarkDown2,
+                              MarkDown3, MarkDown4, MarkDown5, CPI,
+                              Unemployment FROM Walmart_Stores
+                              WHERE Store = 17 AND Holiday = 0")
> range(comparison.set$TOTAL)
```

```
[1]  736335.7 1309226.8
```

```
> hist(comparison.set$TOTAL, breaks = 20, col = "blue",
+      probability = TRUE, main = "TOTAL for Store 17 - Non Holiday Weeks"
+      , xlab = "TOTAL")
> lines(density(comparison.set$TOTAL), col = "red", lty = "dotted",
+       lwd = 3)
```



The range of $TOTAL$ is \$736, 336 to \$1, 309, 227 for store 17 on weeks that are not holiday weeks. $TOTAL$ for store 17 on weeks that are not holiday weeks appears to follow a normal distribution. Note, the prediction from the bagged model is \$771, 672 and the mean of $TOTAL$ under these conditions is \$887, 099. Moreover, the standard deviation is less than \$100, 000 and the be tightly clustered around the mean. Therefore, it makes sense to forecast \$771, 672 for weekly since it is closer to the center of the distribution of $TOTAL$ under these conditions. However, we cannot rule out the fact that the observed value of $TOTAL$ may fall near the predicted value from the random forest. One thing is for sure, the predictions of $TOTAL$ for X from the regression tree, bagged model, and random forest all fall within one standard deviation of the mean of $TOTAL$.

Conclusion

In this paper, we briefly explained the benefit of using decision trees over other methods such as linear or logistic regression. Then, we discussed how recursive binary splitting is used for decision trees. After we created our data set, we built a regression tree that modeled a target variable, $TOTAL$ that we engineered, and a classification tree that model a target variable called $Class.TOTAL$ that we engineered as well. Later, we observed that the performance of the decision trees on the training and test sets was very good. Finally, we attempted to improve our regression tree through bagging and growing a random forest. The bagged model and random forest were used to predict the value of $TOTAL$ for a new observation X. One topic that we did not cover was *boosting* for decision trees. We plan on covering this topic in future projects where decision trees are appropriate learning methods. We may even cover boosting for a completely different learning method.

References

[1] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning with applications in R*. Springer, 2017.

[2] Mandy C. Phelps and Edgar C. Merkle. Classification and regression trees as alternatives to regression. *In Proceedings: 4th Annual Symposium: Graduate Research and Scholarly Projects*, pages 77–78, 2008.

[3] Wei-Yin Loh. Classification and regression trees. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(1):14–23, 2011. doi: 10.1002/widm.8.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning, second edition: data mining, inference, and prediction.* Springer, 2009.

[5] G.g. Moisen. Classification and regression trees. *Encyclopedia of Ecology*, page 582–588, 2008. doi: 10.1016/b978-008045405-4.00149-x.

[6] Manjeet Singh. Retail data analytics, Sep 2017. URL `https://www.kaggle.com/manjeetsingh/retaildataset`.

[7] Walmart recruiting - store sales forecasting. URL `https://www.kaggle.com/c/walmart-recruiting-store-sales-forecasting/data`.

[8] Stef Van Buuren and Karin Groothuis-Oudshoorn. mice: Multivariate imputation by chained equations inr. *Journal of Statistical Software*, 45(3), 2011. doi: 10.18637/jss.v045.i03.

[9] Morris H. DeGroot and Mark J. Schervish. *Probability and statistics*. Addison-Wesley, 2012.

[10] Robert V. Hogg and Elliot A. Tanis. *Probability and statistical inference*. Macmillan Publishing Company, 1994.