Introduction to Machine Learning - Final Project

Bitcoin and Kickstarter Projects Data Set Analysis

Author: Thomas Asuncion

Part 1: Bitcoin Data Set

Objective

I aim to create 3 different regression models on our historical Bitcoin dataset that may be used for **price prediction.** Inspiration for this project comes from patterns and models that have been seen on cryptocurrency forums such as the Triangle Pattern Stock. The goal of this project is to see if it truly is possible to create a model that can predict the price of Bitcoin by simply relying on previous price information. Ideally, I would like to see if the price prediction models that I have observed on forums and other websites are accurate or inaccurate. From previous experience, the models I have seen online are usually wrong most of the times, but if we can create a model of our own that predicts relatively well then, we may have something worthwhile!

Please note: *This dataset does not contain ANY factors that may have influenced the price on some given date such as political news, media coverage or attacks. It is without a doubt that those excluded factors have a strong correlation on price and should be noted when evaluating the accuracy of our price prediction models.*

Link to dataset: https://www.kaggle.com/mczielinski/bitcoin-historical-data (https://www.kaggle.com/mczielinski/bitcoin-historical-data)

Importing Our Bitcoin Dataset

The following Bitcoin dataset (Bitstamp-USD) has been downloaded from Kaggle. A description of what each column represents can be found here:

- **Time Stamp** it gives the information of Bitcoin, that is how many times it is spent.
- **Open** gives the cost of Bitcoin, when the day have started or when the stock have started.
- **Close** gives the value of the Bitcoin when the stock have closed. Closing value should be more than the open value, it indicates the profit.
- **High** gives the highest cost of Bitcoin in a day.
- **Low** gives the lowest cost of the Bitcoin in a day.
- **Volume** gives the, total amount of block transaction per 24 hrs.
- **Volume currency** is the cost of the volume.
- **Weighted price** gives the accurate value of Bitcoin in market price, by dynamic calculation for each block in the block chain.

The following data spans from 01/01/2012 to 06/27/2018; roughly 6 years of data (each row represents 1 minute). All of the following columns are numeric values, none of the columns are considered factors.

```
# Importing Bitcoin data and loading into memory
BitcoinData <- read.csv("C:/Users/avaen/Desktop/bitcoindata.csv", header=TRUE)
```

The following dataset will be stored into “BitcoinData” as a dataframe which can then be operated on.

Exploring the Bitcoin Dataset

Now that our data has been imported let’s perform some R operations on it to pull out some useful information.

```
# Return the column names of our BitcoinData.
names(BitcoinData)
```

```
## [1] "Timestamp"         "Open"              "High"
## [4] "Low"               "Close"             "Volume_.BTC."
## [7] "Volume_.Currency." "Weighted_Price"
```

```
# Return a statistical summary of our BitcoinData.
summary(BitcoinData)
```

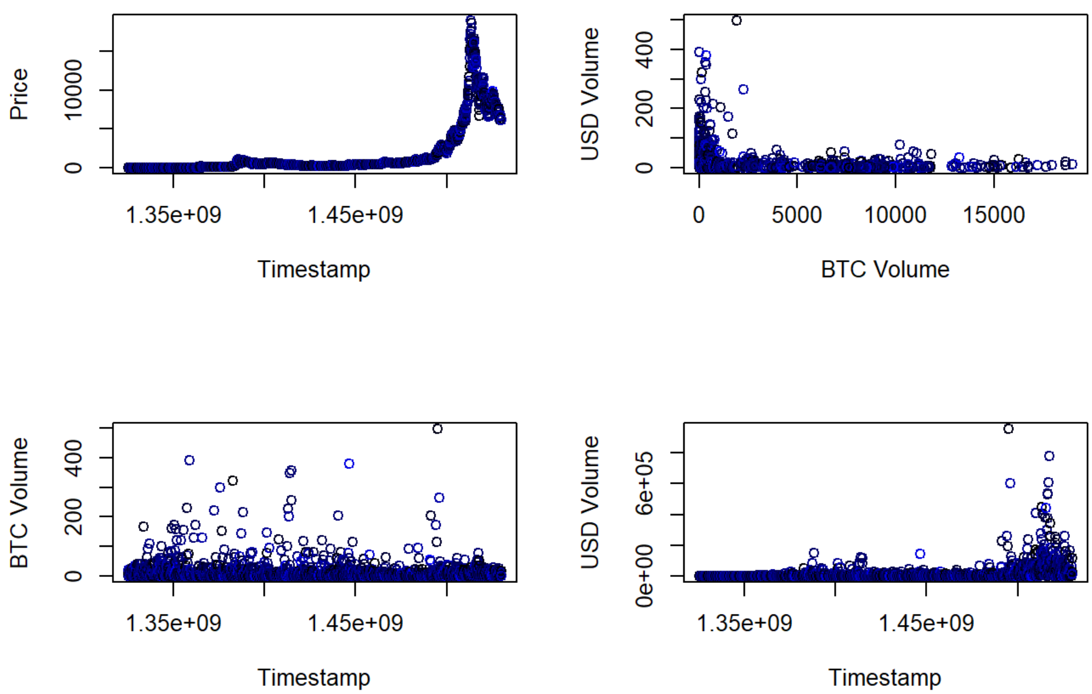
```
##    Timestamp              Open              High               Low
##  Min.   :1.325e+09   Min.   :    3.8   Min.   :    3.8   Min.   :    1.5
##  1st Qu.:1.376e+09   1st Qu.:  122.8   1st Qu.:  122.9   1st Qu.:  122.8
##  Median :1.428e+09   Median :  416.7   Median :  416.9   Median :  416.6
##  Mean   :1.428e+09   Mean   : 1535.8   Mean   : 1537.2   Mean   : 1534.2
##  3rd Qu.:1.479e+09   3rd Qu.:  809.4   3rd Qu.:  810.0   3rd Qu.:  809.0
##  Max.   :1.530e+09   Max.   :19665.8   Max.   :19666.0   Max.   :19650.0
##      Close          Volume_.BTC.      Volume_.Currency. Weighted_Price
##  Min.   :    1.5   Min.   :   0.000   Min.   :      0   Min.   :    3.8
##  1st Qu.:  122.8   1st Qu.:   0.477   1st Qu.:     77   1st Qu.:  122.8
##  Median :  416.7   Median :   2.157   Median :    474   Median :  416.7
##  Mean   : 1535.7   Mean   :  11.019   Mean   :  15335   Mean   : 1535.6
##  3rd Qu.:  809.5   3rd Qu.:   8.932   3rd Qu.:   4197   3rd Qu.:  809.4
##  Max.   :19665.8   Max.   :5853.852   Max.   :5483271   Max.   :19663.3
```

By running the summary function, we can see some rather interesting information such as the minimum and maximum weighted price of bitcoin, and the increase in volume over time. Due to our data being huge, let’s create a temporary data frame simply for plotting and making observations on a daily, monthly, and yearly basis rather than a minute basis.

```
# Establish a daily bitcoin data frame (1440 minutes in 1 day).
BitcoinDataDaily <- BitcoinData[seq(1, nrow(BitcoinData), 1440),]

# Establish a monthly bitcoin data frame (43800 minutes in 1 month).
BitcoinDataMonthly <- BitcoinData[seq(1, nrow(BitcoinData), 43800),]

# Establish a yearly bitcoin data frame (525600 minutes in 1 year).
BitcoinDataYearly <- BitcoinData[seq(1, nrow(BitcoinData), 525600),]

# Drop "giveaway" columns.
BitcoinDataAdjusted <- BitcoinData[,c(-2,-3,-4,-5)]

# Establish a 5 min interval bitcoin data frame.
BitcoinDataAdjusted <- BitcoinDataAdjusted[seq(1, nrow(BitcoinDataAdjusted), 15),]
```

With our new compressed dataframe let’s make some plots. Plotting this information is a great way to see some usefulness in our data over time.

```
# Setup a 2x2 display.
par(mfrow=c(2,2))

# Used for coloring.
set.seed(100)
z <- sample(1:4, 100, TRUE)
x <- rnorm(100)
y <- rnorm(100)
val <- x + y
valcol <- (val + abs(min(val)))/max(val + abs(min(val)))

# Plot Timestamp vs. Weighted Price.
plot(BitcoinDataDaily$Weighted_Price~BitcoinDataDaily$Timestamp, xlab="Timestamp", ylab="Price", col = rgb(0, 0, valcol))

# Plot Volume BTC vs. Weighted Price.
plot(BitcoinDataDaily$Volume_.BTC.~BitcoinDataDaily$Weighted_Price, xlab="BTC Volume", ylab="USD Volume", col = rgb(0, 0, valcol))

# Plot Timestamp vs. Volume.
plot(BitcoinDataDaily$Volume_.BTC.~BitcoinDataDaily$Timestamp, xlab="Timestamp", ylab="BTC Volume", col = rgb(0, 0, valcol))

# Plot Timestamp vs. Volume.
plot(BitcoinDataDaily$Volume_.Currency.~BitcoinDataDaily$Timestamp, xlab="Timestamp", ylab="USD Volume", col = rgb(0, 0, valcol))
```




```
# Remove the other unused bitcoin datasets to save space
rm(BitcoinDataDaily)
rm(BitcoinDataMonthly)
rm(BitcoinDataYearly)
gc()
```

```
##           used  (Mb) gc trigger  (Mb) max used  (Mb)
## Ncells   529861  28.3   13021041 695.4 10211594 545.4
## Vcells 31685317 241.8   79866034 609.4 78363318 597.9
```

The 1st plot shows how price increases over time, the 2nd plot shows how BTC volume, relates with the price of BTC, the 3rd plot shows the volume of BTC over time, the 4th plot of BTC shows the volume of USD over time.

With our data explored let's divide the data into **testing** and **training** sets to use for our regression algorithms. It is important to use these same sets to evaluate the performance of each model fairly.

```
# Creating test and train sets. Seed set to '1234' for reproducibility.
set.seed(1234)
i <- sample(1:nrow(BitcoinDataAdjusted), nrow(BitcoinDataAdjusted)*0.80, replace=FALSE)
test <- BitcoinDataAdjusted[i,]
train <- BitcoinDataAdjusted[-i,]
```

Our testing and training sets have been created. Let's proceed with the regression algorithms.

Regression Algorithm #1: Linear Regression

The first algorithm that will be used to create our price prediction model will be the classic linear regression. We want our target to be the price of BTC in USD and our predictors to be everything else.

You will notice that I have dropped columns 2, 3, 4 and 5 in preparation for our regression models. Columns 2, 3, 4, and 5 both have the weighted price in them which will cause our model to overfit and memorize rather than making actual predictions.

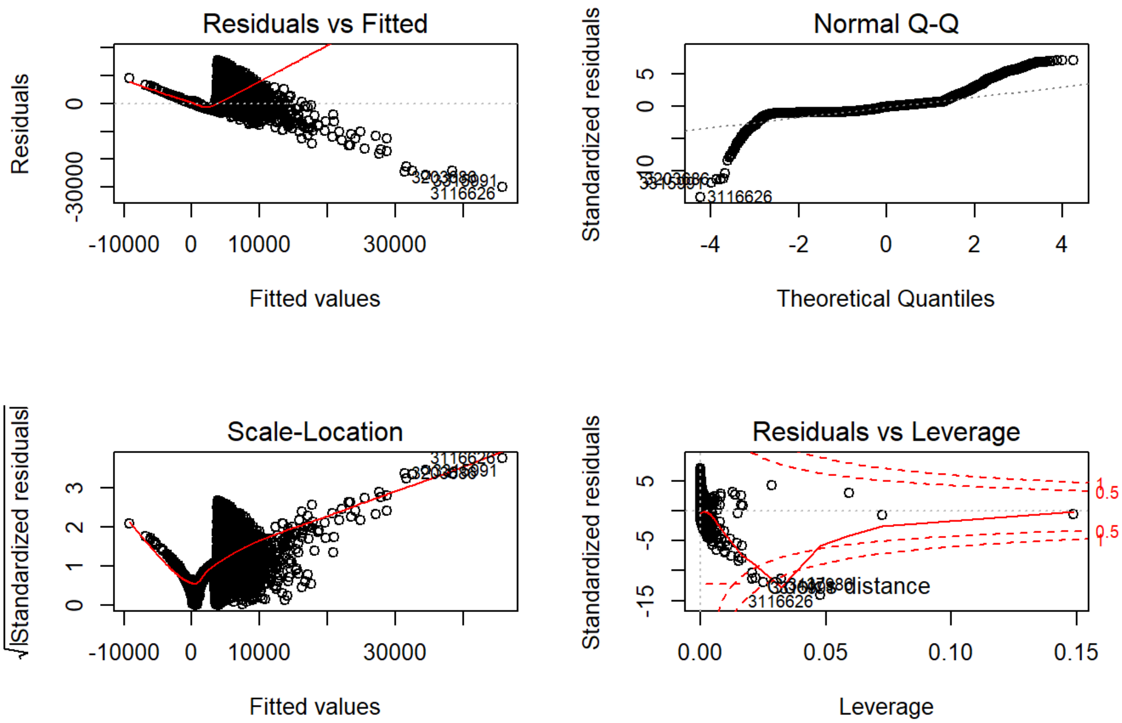
```
# Form the linear regression and store it in 'lm1'.
lm1 <- lm(Weighted_Price~., data=train)

# Get model info.
summary(lm1)
```

```
##
## Call:
## lm(formula = Weighted_Price ~ ., data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -30016.7  -1570.8   -147.4    742.7  15526.8
##
## Coefficients:
##                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)       -3.793e+04  2.600e+02 -145.89   <2e-16 ***
## Timestamp          2.755e-05  1.821e-07  151.31   <2e-16 ***
## Volume_.BTC.      -7.177e+00  2.889e-01  -24.84   <2e-16 ***
## Volume_.Currency.  1.429e-02  1.648e-04   86.72   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2184 on 45408 degrees of freedom
## Multiple R-squared:  0.479,  Adjusted R-squared:  0.479
## F-statistic: 1.392e+04 on 3 and 45408 DF,  p-value: < 2.2e-16
```

Immediately we can see there are signs that we have some good predictors that could be used to predict the weighted price of BTC. However, our **R-squared value** doesn't seem to be very good which may indicate that we could potentially have an inaccurate model.

Plotting the residuals to observe our model some more:

```
# Display in a 2x2
par(mfrow=c(2,2))
plot(lm1)
```



As we can see from our residual plots our model may not perform as good as we want it to be. This isn't too surprisingly considering we dropped the columns that gave away the opening, closing, low and high prices in our model. Let's make some predictions to see what our model returns and compare that to the actual data.

```
# Form the prediction
pred1 <- predict(lm1, newdata=test, na.rm=TRUE)

# Obtain correlation, ignore NAs, and print.
corLm <- cor(pred1, test$Weighted_Price, use = "complete.obs")
print(paste("Correlation = ", corLm))
```

```
## [1] "Correlation =  0.694960131444092"
```

```
# Calculate the MSE & RMSE, remove NAs and print.
mseLm <- mean((pred1-test$Weighted_Price)^2, na.rm = TRUE)
rmseLm <- sqrt(mseLm)
print(paste("Root Mean Squared Error (in USD): ", rmseLm))
```

```
## [1] "Root Mean Squared Error (in USD):  2186.04893741553"
```

```
# Print our prediction
print(cbind(Predicted = head(pred1, n=9), Actual = head(test$Weighted_Price,n=9)))
```

```
##          Predicted    Actual
## 387256   -785.1671   11.8600
## 2119456  2080.0449  393.2895
## 2075086  2005.8042  458.4548
## 2123116  2094.0711  379.5870
## 2932111  3437.1459 2732.4900
## 2180761  2190.4045  431.5300
## 32341   -1383.5677    6.8300
## 792001   -163.5964   86.7000
## 2268511  2332.2885  437.8985
```

Interestingly enough our model seems to have a **correlation value of 0.69** with our actual test data. It also appears to be off by roughly $2186 which isn't terrible but isn't the best considering that our data ranges from $3.80 to $19663.30.

Regression Algorithm #2: Decision Trees

The next regression algorithm that I would like to try is Decision Tree. Perhaps decision tree will perform better and reduce the likelihood of our model overfitting, not that our model was overfitting to begin with but I could be wrong. Before we can use Decision Trees we need to import and load in the appropriate library.

```
if (!require("tree")){
  install.packages("tree")
}
```

```
## Loading required package: tree
```

```
## Warning: package 'tree' was built under R version 3.5.1
```

```
library(tree)
```

Now that our required library is installed let us perform Decision Tree on our data to form a model.

Firstly, we want to store our tree values in a variable "tree1" and make everything else a predictor similar to our regression model up above. Because we want to see how our model performs let's find the correlation values between our tree and the values in our test sets. The hopes is that the correlation value should be relatively high.

```
# Form a regression tree.
tree1 <- tree(Weighted_Price~., data=train)

# Obtain a summary report.
summary(tree1)
```

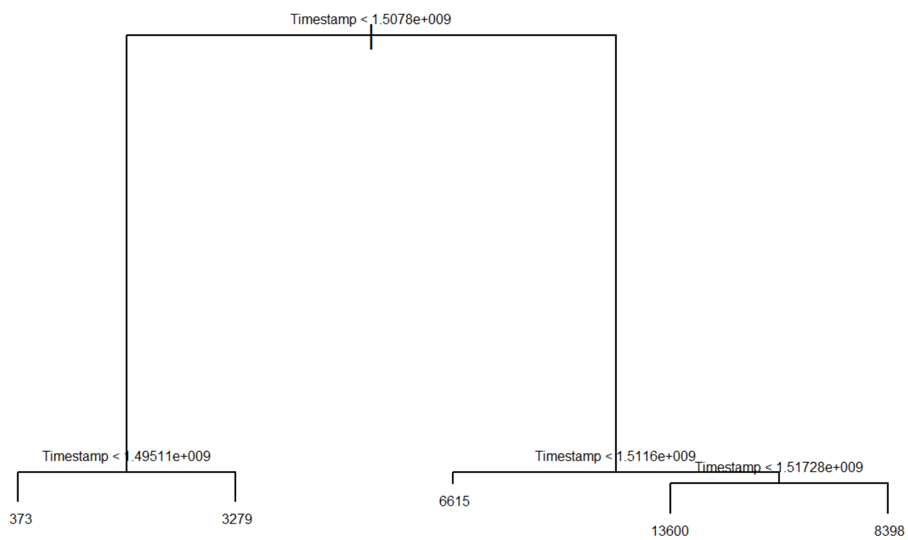
```
##
## Regression tree:
## tree(formula = Weighted_Price ~ ., data = train)
## Variables actually used in tree construction:
## [1] "Timestamp"
## Number of terminal nodes:  5
## Residual mean deviance:  433500 = 1.968e+10 / 45410
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -5245.00  -347.20   -57.34     0.00   243.40  6031.00
```

```
# Make some predictions.
pred2 <- predict(tree1, newdata=test)

# Return the correlation between our predictions and test data.
paste("Correlation = ", cor(pred2, test$Weighted_Price))
```

```
## [1] "Correlation =  0.975882339901067"
```

We should see how our tree turned out, let us return the root mean squared error and plot the tree for visualization.

```
# Return root mean squared error and plot tree.
rmseTree <- sqrt(mean((pred2-test$Weighted_Price)^2))
plot(tree1)
text(tree1, cex=0.5, pretty=1)
```




```
tree1
```

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 45412 4.157e+11  1531.0
##    2) Timestamp < 1.5078e+009 40539 2.897e+10   579.9
##      4) Timestamp < 1.49511e+009 37652 4.139e+09   373.0 *
##      5) Timestamp > 1.49511e+009 2887 2.186e+09  3279.0 *
##    3) Timestamp > 1.5078e+009 4873 4.513e+10  9441.0
##      6) Timestamp < 1.5116e+009 854 7.003e+08  6615.0 *
##      7) Timestamp > 1.5116e+009 4019 3.616e+10 10040.0
##       14) Timestamp < 1.51728e+009 1271 7.857e+09 13600.0 *
##       15) Timestamp > 1.51728e+009 2748 4.802e+09  8398.0 *
```

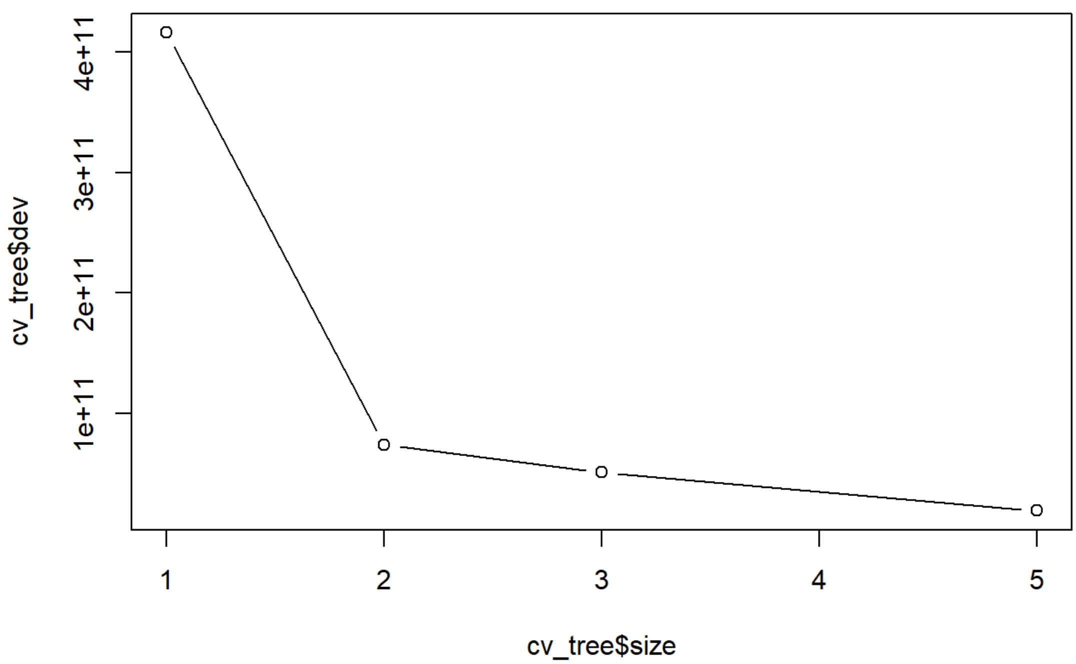
```
print(paste("Root Mean Squared Error (in USD): ", rmseTree))
```

```
## [1] "Root Mean Squared Error (in USD):  663.675729274212"
```

As we can see our decision tree model seems to be off by $663 in comparison to our linear regression model which was off by $2000+. In order to ensure we're not overfitting let's cross validate and prune.

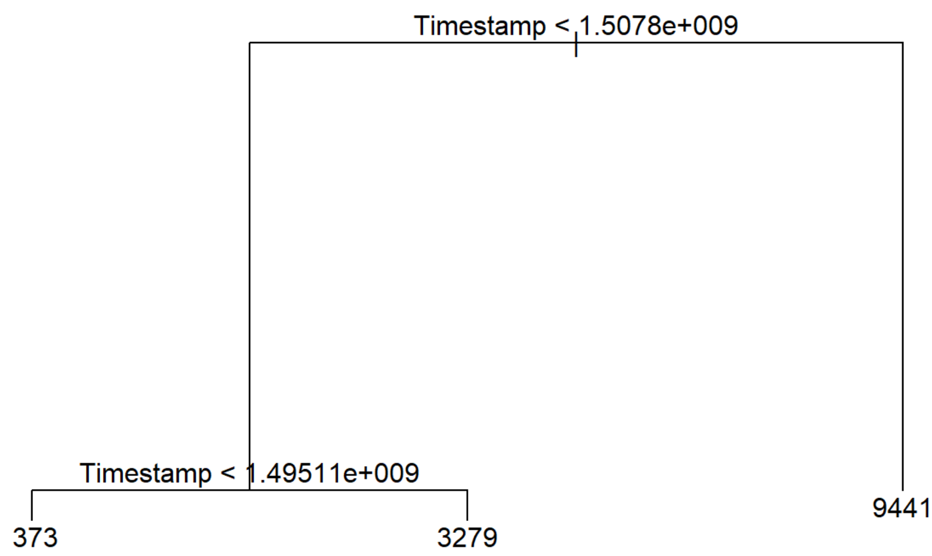
Next let us cross validate our decision tree to help qualify our model.

```
# Perform cross validation
cv_tree <- cv.tree(tree1)
plot(cv_tree$size, cv_tree$dev, type='b')
```



In order to reduce overfitting with our model we can actually **prune our tree** which will reduce the size of our tree by removing sections of the tree that provide noise. This reduction will also reduce the complexity of our model which can reduce overfitting in our model.

```
# Prune the tree and store it in "tree_pruned"
tree_pruned <- prune.tree(tree1, best=3)
plot(tree_pruned)
text(tree_pruned, pretty=0)
```



Now we can test to see if our correlation values have gotten any better (or worst).

```
# Find the correlation.
pred_pruned <- predict(tree_pruned, newdata=test)
paste("Correlation = ", cor(pred_pruned, test$Weighted_Price))
```

```
## [1] "Correlation =  0.936505666806498"
```

```
# Return root mean squared error and plot tree.
rmseTreePruned <- sqrt(mean((pred_pruned-test$Weighted_Price)^2))
print(paste("Root Mean Squared Error (in USD): ", rmseTreePruned))
```

```
## [1] "Root Mean Squared Error (in USD):  1066.08409528188"
```

Here we can see our correlation has reduced from 98 to 93, but not so much so that our model has become completely inaccurate. In addition, our RMSE seems to have almost doubled with our model being off by roughly $1066.

Regression Algorithm #3: kNN Regression

The last regression algorithm I would like to try on our bitcoin data set is the kNN regression algorithm. Ideally, I would have loved to tried neural networks for regression but with a dataset this large I simply **do not have the computational power** to analyze my results.

Let's install and load in the correct library to use kNN regression.

```
# Check for library caret.
if (!require("caret")){
  install.packages("caret")
}
```

```
## Loading required package: caret
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
library(caret)

# Check for library DMwR
if (!require("DMwR")){
  install.packages("DMwR")
}
```

```
## Loading required package: DMwR
```

```
## Warning: package 'DMwR' was built under R version 3.5.1
```

```
## Loading required package: grid
```

```
library(DMwR)
```

Now we can establish a kNN regression model with our training and testing data created above.

```
# Establish a kNN reg model.
fit <- knnreg(train[,1:3],train[,4],k=3)
predictions <- predict(fit, test[,1:3])
corkNN <- cor(predictions, test$Weighted_Price)
paste("Correlation = ", corkNN)
```

```
## [1] "Correlation =  0.999554979893091"
```

With a correlation value of 0.9997 this appears that **our data is overfitting**. This seems rather sketchy, let's see what happens when we scale our data and run the kNN regression algorithm again.

```
# Scale the data and run again.
scaled_data <- scale(BitcoinDataAdjusted[,c(1,2,3,4)])
df <- data.frame(scale(BitcoinDataAdjusted[,c(1,2,3,4)]))
train <- df[i,]
test <- df[-i,]
fit <- knnreg(train[,1:3],train[,4],k=3)
predictions <- predict(fit, test[,1:3])
corkNNScaled <- cor(predictions, test$Weighted_Price)
rmsekNNScaled <- sqrt(mean((predictions - test$Weighted_Price)^2))
paste("Correlation = ", corkNNScaled)
```

```
## [1] "Correlation =  0.999283348424215"
```

```
print(paste("Root Mean Squared Error (in USD): ", rmsekNNScaled))
```

```
## [1] "Root Mean Squared Error (in USD):  0.0377227796432143"
```

Despite the scaling our data still seems to have a 99% correlation with out test data. Let's try different values of k and see what that may return.

```
cor_k <- rep(0, 20)
rmse_k <- rep(0, 20)
i <- 1
for (k in seq(1, 10, 2)){
  fit_k <- knnreg(train[,1:3],train[,4], k=k)
  pred_k <- predict(fit_k, test[,1:3])
  cor_k[i] <- cor(pred_k, test$Weighted_Price)
  rmse_k[i] <- sqrt(mean((pred_k - test$Weighted_Price)^2))
  print(paste("k=", k, cor_k[i], rmse_k[i]))
  i <- i + 1
}
```

```
## [1] "k= 1 0.999181230873789 0.0404077649502589"
## [1] "k= 3 0.999283348424215 0.0377227796432143"
## [1] "k= 5 0.999222263654484 0.0392841646197354"
## [1] "k= 7 0.999124170170663 0.0416863633967905"
## [1] "k= 9 0.999025177630885 0.0439785473945799"
```

As it turns out despite what k value we used from 0 to 40 we still seem to have an unreasonably high correlation value. Since it does not matter which k value we choose, we will stick with 3 and deem kNN regression unsuccessful with our data.

Regression Algorithm Analysis: Results

The results were as followed:

- Linear Regression: Correlation of 0.695, RMSE of $2186.05
- Decision Tree: Correlation of 0.976, RMSE of $663.68
- Random Forest: Correlation of 0.937, RMSE of $1066.084
- kNN (k = 3): Correlation of 0.999, RMSE of $0.04

Linear regression had the most believable results, the decision tree was leaning towards overfitting, but luckily random forest reduced it’s overfitting tendency. kNN Regression simply overfitted as a model that was only off by $0.04 appears too good to be true.

Part 2: Kickstarter Projects Data Set

Objective

I am to create 3 different classification models on our Kickstarter projects dataset that may be used for identifying the category of a Kickstarter project based on its respective fields. The hopes are to create a model that could predict what the best Kickstarter category is that will return the best results.

Link to dataset: https://www.kaggle.com/kemical/kickstarter-projects (https://www.kaggle.com/kemical/kickstarter-projects)

Importing Our Kickstarter Dataset

The following Kickstarter Projects Data (2018) has been downloaded from Kaggle. A description of what each column represents can be found here:

- **ID** Internal Kickstarter ID.
- **Name** Name of project - A project is a finite work with a clear goal that you’d like to bring to life. Think albums, books, or films.
- **Category** Category.
- **Main Category** Category of campaign.
- **Currency** Currency used to support.
- **Deadline** Deadline for crowdfunding.
- **Goal** fundraising goal - The funding goal is the amount of money that a creator needs to complete their project.
- **Launched** Date launched.
- **Pledged** Amount pledged by crowd.
- **State** Whether or not the project met its pledge goal.

- **Backers** Number of people who pledged.
- **Country** Country pledged from.
- **USD Pledged** Amount of money pledged.

The following data are Kickstarter projects from 2018.

```
# Importing Kickstarter data and loading into memory
KickstarterData <- read.csv("C:/Users/avaen/Desktop/kickstarterdata.csv", header=TRUE)

# Drop unneeded columns.
KickstarterDataAdjusted <- KickstarterData[,c(-1, -2, -3, -5, -6, -8, -9, -12, -13)]
```

The following dataset will be stored into "KickstarterData" as a dataframe which can then be operated on.

Exploring the Kickstarter Projects Dataset

Now that our data has been imported let's perform some R operations on it to pull out some useful information.

```
# Return the column names of our KickstarterData.
names(KickstarterData)
```

```
##  [1] "ID"               "name"             "category"
##  [4] "main_category"    "currency"         "deadline"
##  [7] "goal"             "launched"         "pledged"
## [10] "state"            "backers"          "country"
## [13] "usd.pledged"      "usd_pledged_real" "usd_goal_real"
```

```
# Return the structure of our KickstarterData
str(KickstarterData)
```

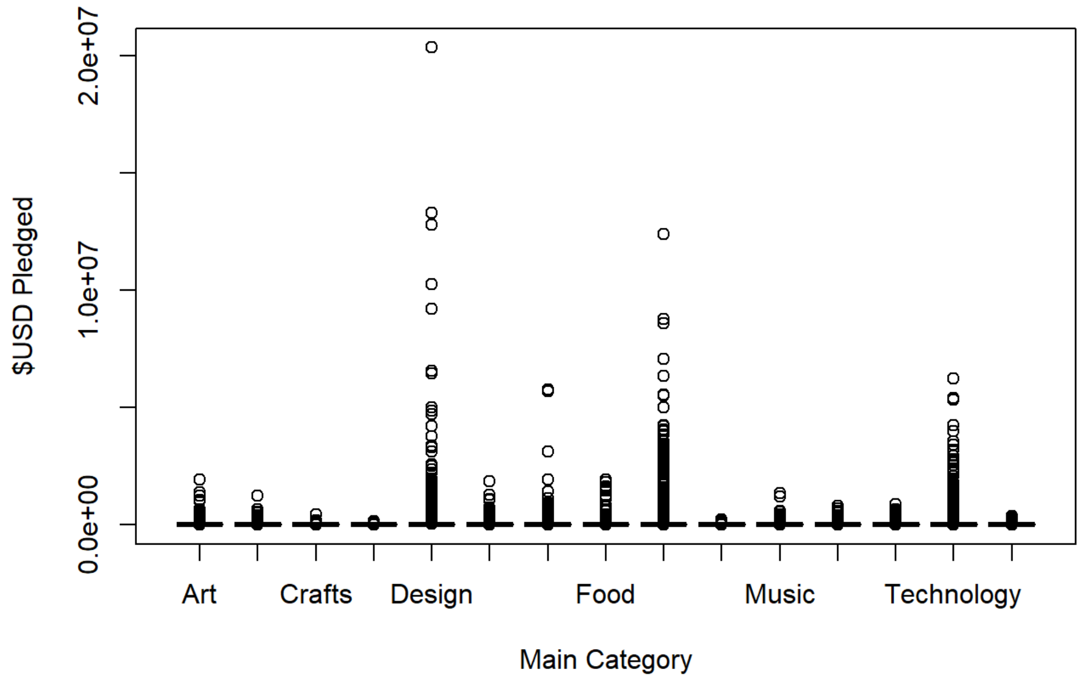
```
## 'data.frame':    378661 obs. of  15 variables:
##  $ ID              : int  1000002330 1000003930 1000004038 1000007540 1000011046 1000014025 1000023410 10000305
81 1000034518 100004195 ...
##  $ name            : Factor w/ 375765 levels "","\177Not Twins - New EP! \"The View from Down Here\"",..: 33254
1 135689 365010 344805 77349 206130 293462 69360 284139 290718 ...
##  $ category        : Factor w/ 159 levels "3D Printing",..: 109 94 94 91 56 124 59 42 114 40 ...
##  $ main_category   : Factor w/ 15 levels "Art","Comics",..: 13 7 7 11 7 8 8 8 5 7 ...
##  $ currency        : Factor w/ 14 levels "AUD","CAD","CHF",..: 6 14 14 14 14 14 14 14 14 14 ...
##  $ deadline        : Factor w/ 3164 levels "2009-05-03","2009-05-16",..: 2288 3042 1333 1017 2247 2463 1996 244
8 1790 1863 ...
##  $ goal            : num  1000 30000 45000 5000 19500 50000 1000 25000 125000 65000 ...
##  $ launched        : Factor w/ 378089 levels "1970-01-01 01:00:00",..: 243292 361975 80409 46557 235943 278600
187500 274014 139367 153766 ...
##  $ pledged         : num  0 2421 220 1 1283 ...
##  $ state           : Factor w/ 6 levels "canceled","failed",..: 2 2 2 2 1 4 4 2 1 1 ...
##  $ backers         : int  0 15 3 1 14 224 16 40 58 43 ...
##  $ country         : Factor w/ 23 levels "AT","AU","BE",..: 10 23 23 23 23 23 23 23 23 23 ...
##  $ usd.pledged     : num  0 100 220 1 1283 ...
##  $ usd_pledged_real: num  0 2421 220 1 1283 ...
##  $ usd_goal_real   : num  1534 30000 45000 5000 19500 ...
```

```
# Return a statistical summary of our KickstarterData.
summary(KickstarterData)
```

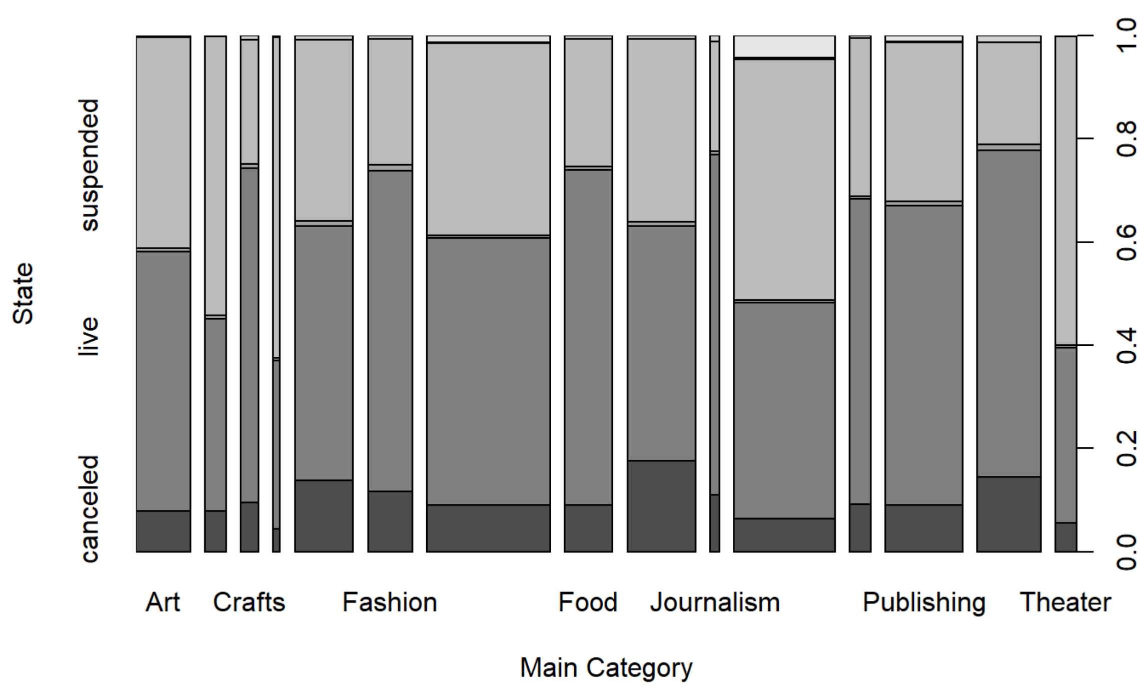
```
##       ID                                    name       
##  Min.   :5.971e+03   New EP/Music Development:    41  
##  1st Qu.:5.383e+08   Canceled (Canceled)     :    13  
##  Median :1.075e+09   Music Video             :    11  
##  Mean   :1.075e+09   N/A (Canceled)          :    11  
##  3rd Qu.:1.610e+09   Cancelled (Canceled)    :    10  
##  Max.   :2.147e+09   Debut Album             :    10  
##                      (Other)                 :378565  
##            category           main_category       currency    
##  Product Design: 22314   Film & Video: 63585   USD    :295365  
##  Documentary   : 16139   Music       : 51918   GBP    : 34132  
##  Music         : 15727   Publishing  : 39874   EUR    : 17405  
##  Tabletop Games: 14180   Games       : 35231   CAD    : 14962  
##  Shorts        : 12357   Technology  : 32569   AUD    :  7950  
##  Video Games   : 11830   Design      : 30070   SEK    :  1788  
##  (Other)       :286114   (Other)     :125414   (Other):  7059  
##       deadline           goal                          launched     
##  2014-08-08:   705   Min.   :        0   1970-01-01 01:00:00:     7  
##  2014-08-10:   558   1st Qu.:     2000   2009-09-15 05:56:28:     2  
##  2014-08-07:   541   Median :     5200   2010-06-30 17:29:43:     2  
##  2015-05-01:   489   Mean   :    49081   2011-02-08 04:29:48:     2  
##  2014-08-09:   477   3rd Qu.:    16000   2011-02-25 09:58:36:     2  
##  2015-07-01:   449   Max.   :100000000   2011-03-03 17:55:38:     2  
##  (Other)   :375442                       (Other)            :378644  
##     pledged                  state           backers        
##  Min.   :       0   canceled  : 38779   Min.   :     0.0  
##  1st Qu.:      30   failed    :197719   1st Qu.:     2.0  
##  Median :     620   live      :  2799   Median :    12.0  
##  Mean   :    9683   successful:133956   Mean   :   105.6  
##  3rd Qu.:    4076   suspended :  1846   3rd Qu.:    56.0  
##  Max.   :20338986   undefined :  3562   Max.   :219382.0  
##                                                           
##     country        usd.pledged       usd_pledged_real  
##  US     :292627   Min.   :       0   Min.   :       0  
##  GB     : 33672   1st Qu.:      17   1st Qu.:      31  
##  CA     : 14756   Median :     395   Median :     624  
##  AU     :  7839   Mean   :    7037   Mean   :    9059  
##  DE     :  4171   3rd Qu.:    3034   3rd Qu.:    4050  
##  N,0"   :  3797   Max.   :20338986   Max.   :20338986  
##  (Other): 21799   NA's   :3797                         
##  usd_goal_real      
##  Min.   :        0  
##  1st Qu.:     2000  
##  Median :     5500  
##  Mean   :    45454  
##  3rd Qu.:    15500  
##  Max.   :166361391  
## 
```

Given our data is now loaded in R's memory. Let's make some plots so that we can see the relationship between some values.

```
# Plot Main Category vs. $USD Pledged.
plot(KickstarterDataAdjusted$usd_pledged_real~KickstarterDataAdjusted$main_category, xlab="Main Category", ylab=
"$USD Pledged", col="red")
```




```
# Plot Main Category vs. State.
plot(KickstarterDataAdjusted$state~KickstarterDataAdjusted$main_category, xlab="Main Category", ylab="State")
```



The 1st plot shows the category and how much money was pledged, the 2nd plot shows the category and whether or not it was successful in meeting the goal, canceled, suspended, or failed.

With our data explored let's divide the data into **testing** and **training** sets to use for our classification algorithms. It is important to use these same sets to evaluate the performance of each model fairly.

```
# Creating test and train sets. Seed set to '0000' for reproducibility.
set.seed(0000)
i <- sample(1:nrow(KickstarterDataAdjusted), nrow(KickstarterDataAdjusted)*0.80, replace=FALSE)
test <- KickstarterDataAdjusted[i,]
train <- KickstarterDataAdjusted[-i,]
```

Our testing and training sets have been created. Let’s proceed with the classification algorithms.

Classification Algorithm #1: Logistic Regression

The first classification algorithm I would like to try on our data is logistic regression. Notice that I have removed some columns that may not be helpful towards our model. In order to ensure our model is accurate it is best to perform some data cleaning before proceeding. While I do not believe there are NAs in our data it doesn’t hurt to check.

```
sapply(KickstarterDataAdjusted, function(x) sum(is.na(x)==TRUE))
```

```
##    main_category             goal            state          backers
##                0                0                0                0
## usd_pledged_real    usd_goal_real
##                0                0
```

Now that our data is clean let’s create the logistic model.

```
# Create a log model where main_category is the target and everything else minus the category are the predictors.
glm1 <- glm(main_category~., data=train, family =binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm1)
```

```
##
## Call:
## glm(formula = main_category ~ ., family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904   0.3652   0.3921   0.3935   0.4978
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       2.673e+00  4.734e-02  56.470  < 2e-16 ***
## goal             -1.387e-09  3.407e-08  -0.041  0.96751
## statefailed      -1.533e-01  5.108e-02  -3.001  0.00269 **
## statelive        -1.626e-01  1.652e-01  -0.984  0.32499
## statesuccessful  -5.760e-01  5.392e-02 -10.682  < 2e-16 ***
## statesuspended    8.591e-02  2.473e-01   0.347  0.72830
## stateundefined    1.386e+01  8.826e+01   0.157  0.87524
## backers           1.145e-03  2.285e-04   5.012 5.40e-07 ***
## usd_pledged_real  1.712e-05  2.839e-06   6.031 1.62e-09 ***
## usd_goal_real    -3.335e-09  3.708e-08  -0.090  0.92833
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 39945  on 75732  degrees of freedom
## Residual deviance: 39336  on 75723  degrees of freedom
## AIC: 39356
##
## Number of Fisher Scoring iterations: 15
```

As it turns out the total amount of pledged money and the state are good variables on predicting Kickstarter categories.

Now let’s make some predictions and see what our model returns.

```
# Make some predictions.
probs <- predict(glm1, newdata=test, type="response")
pred <- ifelse(probs>0.5, 1, 0.5)
acc <- mean(pred==as.integer(test$main_category))
print(paste("accuracy = ", acc))
```

```
## [1] "accuracy =  0.0744500343315904"
```

As it turns out, our model **has low accuracy**. Perhaps we can run logistic regression again just 2 predictors and see if that will improve our chances of obtaining a better model.

```
# Redo the log model for better results.
glm2 <- glm(main_category~.-goal-usd_goal_real-goal-backers, data=train, family =binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
summary(glm2)
```

```
##
## Call:
## glm(formula = main_category ~ . - goal - usd_goal_real - goal -
##     backers, family = binomial, data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -8.4904   0.3648   0.3921   0.3928   0.4740
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)       2.676e+00  4.731e-02  56.566  < 2e-16 ***
## statefailed      -1.530e-01  5.108e-02  -2.995  0.00274 **
## statelive        -1.603e-01  1.652e-01  -0.970  0.33187
## statesuccessful  -5.470e-01  5.361e-02 -10.202  < 2e-16 ***
## statesuspended    8.866e-02  2.473e-01   0.359  0.71995
## stateundefined    1.383e+01  8.824e+01   0.157  0.87545
## usd_pledged_real  2.878e-05  2.099e-06  13.711  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 39945  on 75732  degrees of freedom
## Residual deviance: 39392  on 75726  degrees of freedom
## AIC: 39406
##
## Number of Fisher Scoring iterations: 15
```

Now let’s make some predictions and see what our model returns.

```
# Make some predictions.
probs2 <- predict(glm2, newdata=test, type="response")
pred2 <- ifelse(probs2>0.5, 1, 0.5)
acc2 <- mean(pred2==as.integer(test$main_category))
print(paste("Accuracy = ", acc2))
```

```
## [1] "Accuracy =  0.0744500343315904"
```

There was little to no difference with our 2 different log models. Let’s move on to another classification algorithm and see if we have any luck there.

Classification Algorithm #2: Naive Bayes

The next classification algorithm that I would like to run on our dataset is the Naïve Bayes algorithm. Let’s prepare our data for Naïve Bayes. Naïve Bayes is one of the easier algorithms to setup, my assumptions are that this algorithm may not perform as well as the logistic regression model due to it’s simplistic nature and the complexity of this data set.

```
# Check for package e1071.
if (!require("e1071")){
  install.packages("e1071")
}
```

```
## Loading required package: e1071
```

```
## Warning: package 'e1071' was built under R version 3.5.1
```

```r
library(e1071)
```

Now that we have the required package let's proceed with the model.

```r
nb1 <- naiveBayes(main_category~., data=train)
```

From the information above we can see where we were off. Let's get a value to grade the performance of our model.

```r
# Make some predictions.
p2 <- predict(nb1, newdata=test, type="class")
table(p2, test$main_category)
```

```
##
## p2              Art Comics Crafts Dance Design Fashion Film & Video  Food
##   Art             0      0      0     0      0       0            0     0
##   Comics          0      0      0     0      0       0            0     0
##   Crafts          2      3      2     2     17       4           20    13
##   Dance       20450   6829   6601  2719  15839   15593        40338 15516
##   Design          1      0      0     0      2       0            3     0
##   Fashion       419    713     92    24   2710     594         2022   719
##   Film & Video    6      0      0     0     18       6           60    14
##   Food            7      2      0     0     81      10           81    13
##   Games          36     98      6     1    730     102          265    53
##   Journalism      1      0      0     0      1       1            2     0
##   Music         518     76    130    41   1571     581         3816  1532
##   Photography  1055    969    185   191   2778    1298         3856  1752
##   Publishing      3      3      1     0     68       5           49     8
##   Technology      7      1      0     0    111      11           85    11
##   Theater        48      6      3     6     74      25          238    91
##
## p2            Games Journalism Music Photography Publishing Technology
##   Art             0          0     0           0          0          0
##   Comics          0          0     0           0          0          0
##   Crafts         29          4     6           6          8         52
##   Dance       18692       3297 35013        7739      27978      17394
##   Design          0          0     0           0          0          5
##   Fashion      3533        101   844         197       1001       1935
##   Film & Video   54          3     3           1          2         66
##   Food           41          1     4           2          7        105
##   Games         910         12    76          11         97        642
##   Journalism      1          0     0           0          0          3
##   Music        1678        177  2421         135        973       3725
##   Photography  3042        131  3093         512       1760       1639
##   Publishing     83          4     3           2         14         79
##   Technology     84          0     6           6         12        198
##   Theater       116         15    50           2         27        274
##
## p2            Theater
##   Art               0
##   Comics            0
##   Crafts            4
##   Dance          7868
##   Design            1
##   Fashion         154
##   Film & Video      3
##   Food              6
##   Games            14
##   Journalism        0
##   Music           193
##   Photography     483
##   Publishing        0
##   Technology        5
##   Theater          17
```

```
acc3 <- mean(p2==test$main_category)
print(paste("Accuracy = ", acc3))
```

```
## [1] "Accuracy =  0.0246329160724661"
```

As assumed, our Naïve Bayes model only has an accuracy of roughly 2%. This is less than our logistic regression model by roughly 0.05 or 5%.

Classification Algorithm #3: Decision Tree Classification

Due to the large amount of data that I am handling I will have to be forced to use Decision Tree Classification. The other 2 algorithms, **Neural Nets and SVM are to computationally expensive** and my computer cannot handle this much data being processed by them. Therefore let us continue with Decision Tree Classification.

```
# Download the required package.
if (!require("tree")){
  install.packages("tree")
}

library(tree)
```

With the correct package installed and loaded let's continue with DT classification.

```
# Setup tree.
tree_kick <- tree(main_category~., data=KickstarterDataAdjusted)

# Display tree.
tree_kick
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 378661 1875000 Film & Video ( 0.074349 0.028572 0.023264 0.009951 0.079411 0.060254 0.167921 0.064971
0.093041 0.012557 0.137109 0.028466 0.105303 0.086011 0.028820 )
##   2) goal < 9891.5 229372 1133000 Music ( 0.095975 0.037005 0.030522 0.013376 0.055473 0.058608 0.157635 0.0473
68 0.080306 0.013162 0.171146 0.034751 0.123890 0.043715 0.037066 ) *
##   3) goal > 9891.5 149289  704700 Film & Video ( 0.041122 0.015614 0.012111 0.004689 0.116191 0.062784 0.183724
0.092016 0.112607 0.011628 0.084815 0.018809 0.076744 0.150996 0.016150 ) *
```

```
# Obtain a summary of our data.
summary(tree_kick)
```

```
##
## Classification tree:
## tree(formula = main_category ~ ., data = KickstarterDataAdjusted)
## Variables actually used in tree construction:
## [1] "goal"
## Number of terminal nodes:  2
## Residual mean deviance:  4.853 = 1838000 / 378700
## Misclassification error rate: 0.8239 = 311977 / 378661
```

With our tree setup let's get a visual of our tree.

```
# Plotting our tree.
plot(tree_kick)
text(tree_kick, cex=0.5, pretty=0)
```

Unfortunately, my tree only used **one variable: goal** which isn't very helpful. Either way let's make some predictions and see what happens.

```
# Form a new tree based on train data.
tree_kick2 <- tree(main_category~., data=train)

# Make some predictions.
predTree2 <- predict(tree_kick2, newdata=test, type="class")
accTree2 <- mean(predTree2==test$main_category)
paste("Accuracy = ", accTree2)
```

```
## [1] "Accuracy =  0.176104552897058"
```

With an accuracy of 0.17 this is indeed better than our Naïve Bayes model and our logistic regression model. So far this yielded the best results, let's try to use random forest and see if that will allow us to produce even better results.

```
# Load in the needed package.
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.5.1
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
# Establish a random forest.
tree_forestclass <- randomForest(main_category~., data=train)

# Call the forest by name.
tree_forestclass
```

```
##
## Call:
##  randomForest(formula = main_category ~ ., data = train)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 80.49%
## Confusion matrix:
##              Art Comics Crafts Dance Design Fashion Film & Video Food
## Art          408     87     57    24    231     160         1680  229
## Comics       124    121     20     6    127      50          353   71
## Crafts       142     21     40     2     61      61          540   91
## Dance         46      9      6     1     19      20          257   23
## Design       210     80     39     6    898     169         1500  266
## Fashion      196     42     38     8    327     171         1564  202
## Film & Video 513    110     73    34    629     287         4827  554
## Food         174     52     33     7    281     140         1728  469
## Games        250    166     54    12    739     186         1367  263
## Journalism    34      9     13     1     35      26          349   59
## Music        462    116     66    25    327     211         3165  330
## Photography  138     34     18     5     90      43          714   82
## Publishing   399    152     47    20    380     192         2394  301
## Technology   187     54     34     5    651     202         2007  331
## Theater      139     22     19     6     55      34          737   53
##              Games Journalism Music Photography Publishing Technology
## Art            333         12  1457          51        559        238
## Comics         403          3   477          26        229         91
## Crafts         128          5   396          21        188         80
## Dance           21          2   271           8         67         18
## Design         982         18   740          31        379        720
## Fashion        329         11   893          29        418        332
## Film & Video   586         28  2997          81        951        924
## Food           315         16   795          23        365        459
## Games         1844         27   850          54        503        615
## Journalism      75          1   216           9        118         60
## Music          497         16  3765          73        883        343
## Photography    126          6   520          31        214        112
## Publishing     563         27  2032          70        944        399
## Technology     661         13   673          27        393       1195
## Theater         78          3   716          22        161         63
##              Theater class.error
## Art               74   0.9271429
## Comics            18   0.9428976
## Crafts            13   0.9776411
## Dance             16   0.9987245
## Design            32   0.8520593
## Fashion           26   0.9627126
## Film & Video     156   0.6214118
## Food              23   0.9038934
## Games             38   0.7353617
## Journalism         5   0.9990099
## Music            120   0.6379460
## Photography       33   0.9856879
## Publishing        75   0.8819262
## Technology        19   0.8147861
## Theater           57   0.9736721
```

Let’s make some predictions.

```
# Make some predictions
pred_forest2 <- predict(tree_forestclass, newdata=test, type="class")

# Obtain the mean
predforestacc <- mean(pred_forest2==test$main_category)
paste("Accuracy = ", predforestacc)
```

```
## [1] "Accuracy =  0.195056250990334"
```

This was defiantly an improvement in model performance. An accuracy of 0.19 isn't a lot but in comparison to the other models this was a lot better!

Classification Algorithm Analysis: Results

The results were as followed:

- Logistic Regression: Accuracy of 0.0744 for all predictors, and Accuracy of 0.0744 for 2 predictors.
- Naive Bayes: Accuracy of 0.025.
- Decision Tree: Accuracy of 0.176.
- Random Forest: Accuracy of 0.195.

Out of all of the models the random forest had the best results. It is no surprised that Naïve Bayes performed the worst as the data provided did not suit the preferences of Naïve Bayes. Logistic Regression had trouble forming a good model, this is due to the perfect seperation issue that wasn't resolved. Despite our decision tree creating a bad tree it was still able to classify our data the best when used in a random forest.