The Precarious Adventure of Predicting the Stock Market with Statistical Methods

Garrett Peuse

Abstract

It is estimated more than 80% of trades on the New York Stock Exchange(NYSE) are from "robots."[1] Many traders are moving towards algorithmic trading to gain an edge. Algorithmic trading is exactly as it sounds: write an algorithm(technically a program) to buy or sell stocks when the defined conditions are met. The advantage is these programs can compute, analyze and trade at a speed and frequency that is impossible for a mortal. More recently with the recent increase in processing power of PCs, statical methods have grown in popularity from hobbyists to professional traders(rather than high finance institutions using multimillion dollar servers run by mathematicians that characterized algorithmic trading in the late 2000s)[2]. The goal of this project was to make the best predictive model of the stock market using free market data while finding relationships in the data that otherwise was not known. To clarify, the aim was to predict an increase or decrease of the S&P 500 for a given day, by using numerous currencies, electronically traded funds, other foreign stock markets and commodities as variables.Using more than 15 predictive models it was found they all fell in similar range of prediction accuracy on test data with the difference between the best of that 15 and worst of the 15 to be 0.0528. Using a hybrid of some of those models, an ensemble method was used to achieve the highest prediction rate of .7528 on the test data. To achieve the best prediction results it was important to choose the best features from a list of 49 features as many were highly correlated with each other, while some had little correlation with the response variable. Through the feature selection process, it became apparent that the importance of the bond market and specifically the U.S. treasury market on NYSE conditions was the most important predictor.

Table of Contents

Abstract……….1
The Data
Data Attainment, Background and Cleaning…………………………………………………………………………....2
Splitting the Data: Train Set, Test Set……………………………………………………………………………………3
Feature Selection: Financial Knowledge VS Subset Selection VS Principal Component Analysis ……………….3
Model Performance
Cross Validation Model Performance…………………………………………………………………………………...5
Deeper Look into the Models…………………………………………………………………………………………….6
Overall Model Performance on Test Data……………………………………………………………………………...7
Absolute best predictive model…………………………………………………………………………………………..8
Discussion/Conclusion…………. ……………………………………………………………………………………………10
References………...10
Appendix A: Data Cleaning Code…………………………………………………………………………………………11
Appendix B: Feature Selection Code……………………………………………………………………………………...14
Appendix C: Model Building and Testing Code…………………………………………………………………………..19

THE DATA

Data Attainment, Background and Cleaning

The data obtained for this project was through Yahoo Finance's free historical records offering data on specific stocks, currencies and commodities. The specific data sets chosen from this was:

- Japan's stock market(known as the Neikie Index),
- Bitcoin worth (based in U.S. dollar),
- Crude oil price(based on the Chicago Exchange),
- Gold price(based in U.S. dollar)
- Eutherian price based in U.S. dollar,
- the ETF(Electronically traded fund) with ticker symbol TLT(20+ Year Treasury Bond ETF seeks to track the investment results of an index composed of U.S. Treasury bonds with remaining maturities greater than twenty years)[3],
- the ETF with LQD(The iShares iBoxx $ Investment Grade Corporate Bond ETF seeks to track the investment results of an index composed of U.S. dollar-denominated, investment grade corporate bonds)[4]
- the ETF with ticker symbol SPDR know in this report as SPY(This ETF is composed of five hundred selected stocks, all of which are listed on national stock exchanges and spans over approximately 24 separate industry groups.)[5]

Each of these data sets had 7 features: Volume(Amount of shares being sold/bought), high and low price of the day, open price, close price, adjusted close price(this is the close price adjusted for stock splits, dividends / distributions and rights offerings, which affect a stock's price and adjustments are needed to arrive at a technically accurate reflection of the true value of that stock)[6] and date--- starting from January 1st, 2015 to December 31st 2019.

From there each data set was merged into one data set, aligned with matching dates.

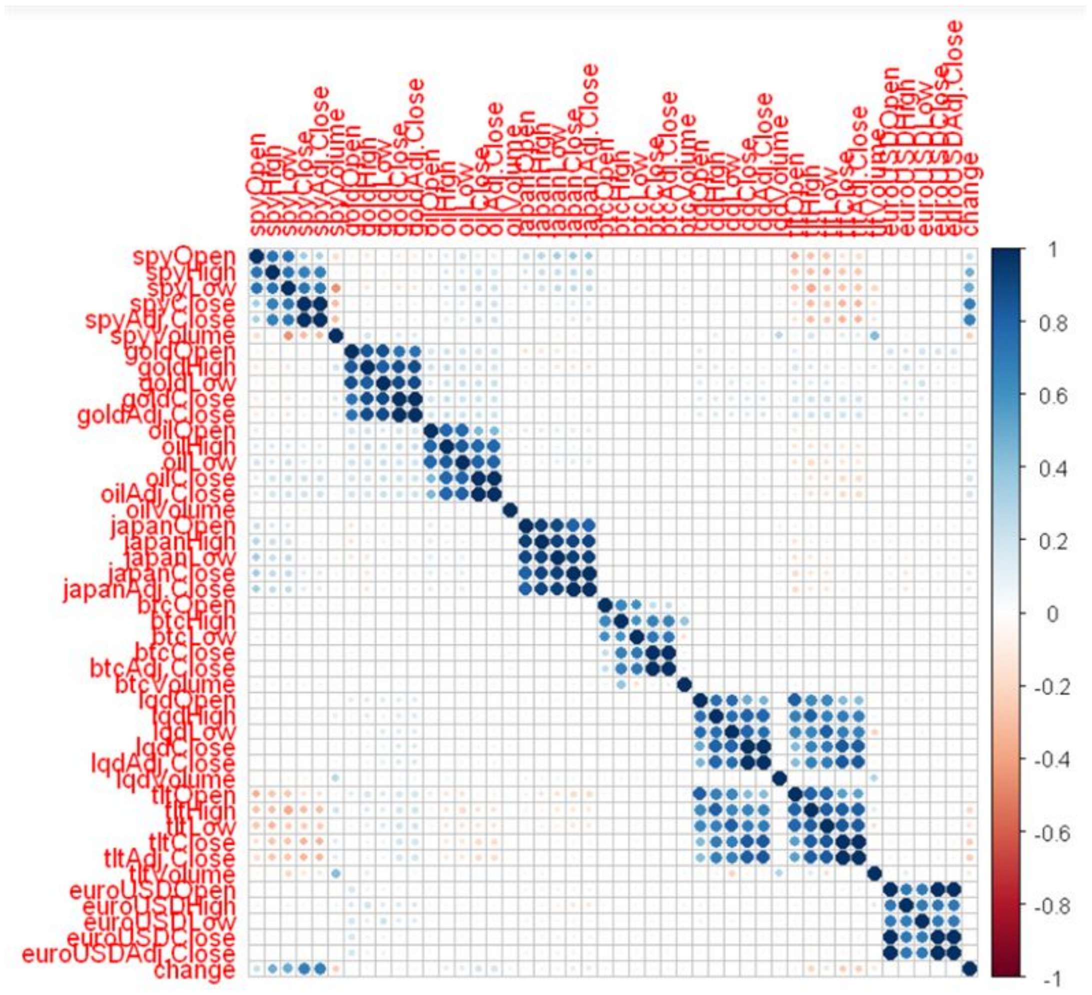
The aim was to predict change in the S&P 500. Another column was made called Change that signified a categorical value if there was a positive percentage change in that day for the S&P 500. This is the category being predicted.

Also, since the S&P 500 is only traded 9:30am and 4:00pm Eastern Time, and other commodities close long after S&P's close time. Thus the close price of some of these commodities would not help in the prediction of change in the U.S. stock market. Therefore, the features where this was the case the information of these prices were shifted back on row to better be able predict the change in the S&P 500. For example, crude oil price close is not helpful to predict the change in S&P(if used in live data) because the crude oil close price is reported 6hrs after the S&P500 has closed and thus the change for that day is already known. Therefore crude oil, Gold, the ETFs and currency were moved back a row in the data, essentially moving it back a day in the data set. Note that Japan's stock market opens and closes before the US stock market opens and thus do not need to be shifted. Thus the use of data in the data set with these changes uses "yesterday's" commodities price in an attempt to predict today's change in the S&P 500.

Data Splitting: Train and Test Sets

After the data cleaning, the left over by product was 1200 data observations and 49 features. Of this 1200, 70% were put into a training data set and the rest(30%) was turned into the test data set. This created a train set consisting of randomly choosing 420 positive change days and 420 S&P decline days to make the training set. The rest of the data was made into a test data set.

Feature Selection



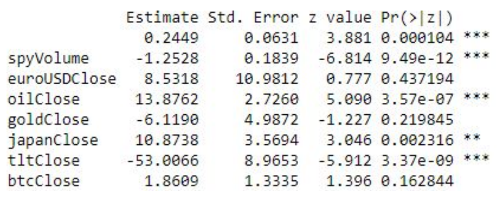
The issue of pockets of correlated data:

The above is a correlation matrix of the data frame used when combining all the data sets from Yahoo Finance into one. The most bottom row and the column on the most right is the Y being predicted, i.e. change in S&P 500. The merging of the data sets can be seen, with the titles having the same beginning, also they are usually highly correlated, this is why there are blocks, "pockets", of correlation in this chart. Variables that came from the same original data set are inherently correlated with each other.

Ideally, using prediction models with features that are not highly correlated is most ideal because features with high correlation are more linearly dependent and hence have almost the same effect on the dependent variable.

There are three approaches that were attempted to deal with the "pockets" of highly correlated features: (1) choosing the features that make the most sense according to sound macroeconomic principles,(2) fitting a least squares regression for each possible combination of predictors and selecting the subset from those results of that analysis, (3) reducing dimensions of the data set with PCA, i.e. creating feature(s) from multiple other features.

The results were quite profound. For example when applying sound economic principles and common sense with logistic regression it was found not all have worth while features.

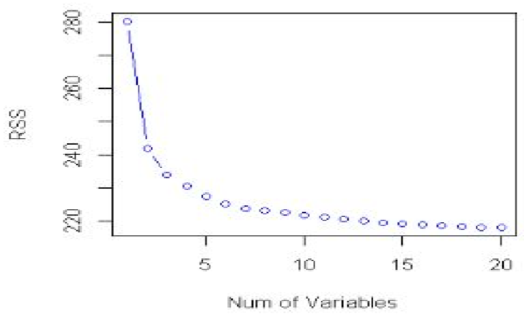


```
             Estimate Std. Error z value Pr(>|z|)    
               0.2449     0.0631   3.881 0.000104 ***
spyVolume     -1.2528     0.1839  -6.814 9.49e-12 ***
euroUSDClose   8.5318    10.9812   0.777 0.437194    
oilClose      13.8762     2.7260   5.090 3.57e-07 ***
goldClose     -6.1190     4.9872  -1.227 0.219845    
japanClose    10.8738     3.5694   3.046 0.002316 ** 
tltClose     -53.0066     8.9653  -5.912 3.37e-09 ***
btcClose       1.8609     1.3335   1.396 0.162844    
```

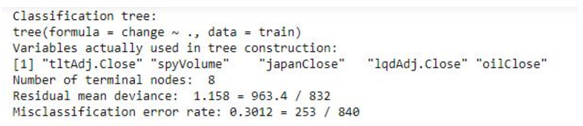
Eliminating all the features associated with high P values, using logistic regression attains a 0.64 on the test data, using 10 fold cross validation. Using the best 8 variables from subset Selection using Logistic regression attains .68 Level of accuracy. While applying PCA on subsets of features highly correlated(meaning making a new vector from the correlated features that have a high effect in Response Y, then using a new vector instead of the features made to create that vector) leads to a .65 level of accuracy.

Subset selection helped finding the best variables to create, but also was most helpful by finding relationships of that data that would not have otherwise been found by "practical" or intuitive knowledge of the market.

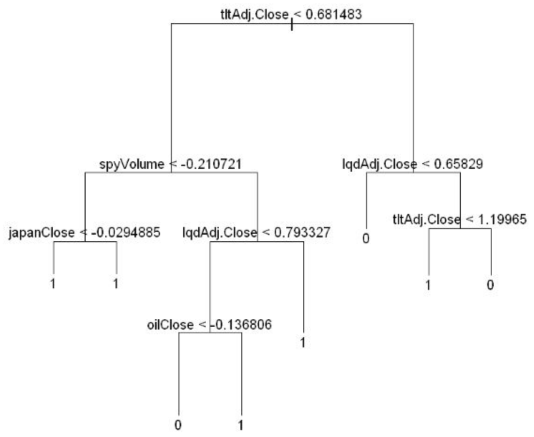
Initially it made sense if close prices were used but not the open. Through subselection it was found out with 9 variables adding in TLT open would increase accuracy and oil volume is more important than oil close.



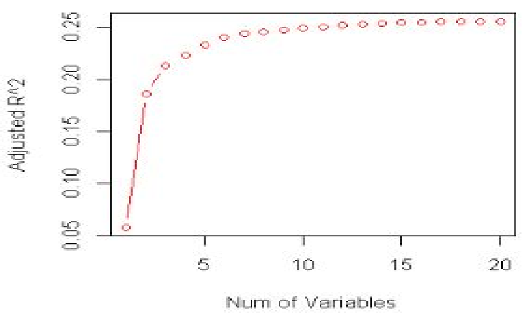
To gain another perspective, comparison or contrast to regression/linear combination based model, in a decision tree, its not really necessary that all the variables are used. Only those which reduce the entropy will be used as a split to nodes. Here splitting is happening based on these variables only, the rest of the variables did not help to reduce the entropy in the data, giving us the value of variables in another sense. Again seeing, TLT at the top(this time adjusted TLT)(and this should be thought as most influential in reducing entropy) with SPY volume and LQD being second most influential variables(depending on the behavior of TLT).



```
Classification tree:
tree(formula = change ~ ., data = train)
Variables actually used in tree construction:
[1] "tltAdj.Close" "spyVolume"    "japanClose"   "lqdAdj.Close" "oilClose"
Number of terminal nodes:  8
Residual mean deviance:  1.158 = 963.4 / 832
Misclassification error rate: 0.3012 = 253 / 840
```

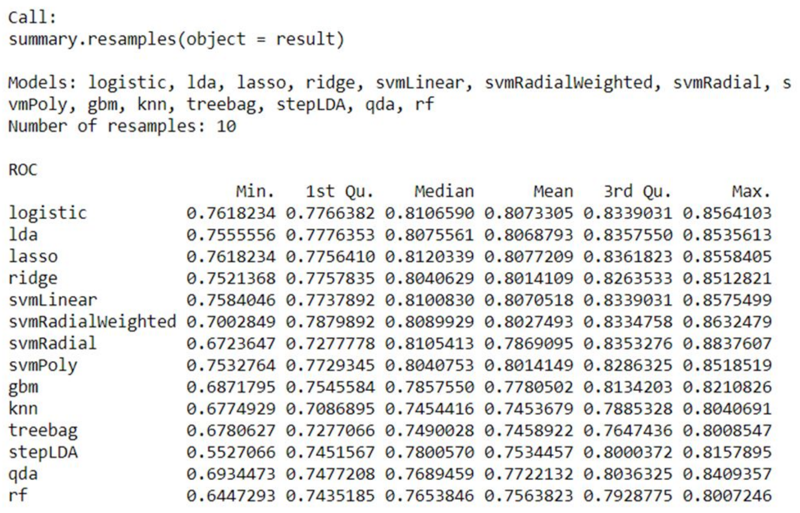


Below is the output from the subset selection which is simply showing the adding of the best choice variable how it reduces the RSS(right). After a certain point it just saturates, meaning R^2 for 8 variables is around 24%and from there (i.e 8 to 20 variables if you go, R^2 goes up by just 1% i.e. from 24 to 25%). This and not being able to create a better PCA features that improve prediction results, is why only eight variables were decided to be used in the creation of all models.



Cross Validation Model Performance

Since the objective was classification an ROC curve (receiver operating characteristic curve)(and AUC) is used to evaluate the performance of classification at all classification thresholds. This was done with cross validation with a re-sample of 10. Thus the below chart is the performance of AUC value for the 10-k fold splits.

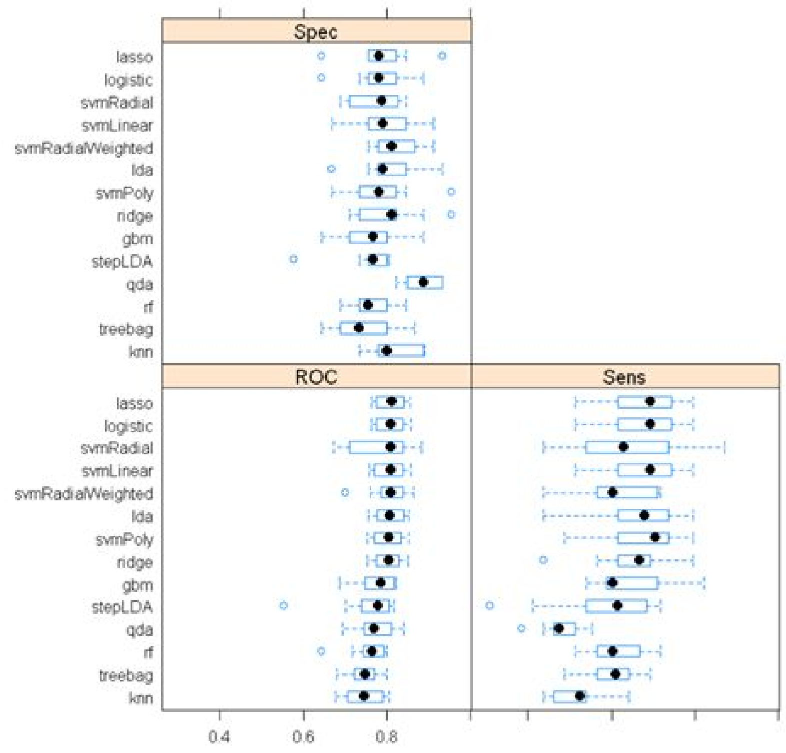


```
Call:
summary.resamples(object = result)

Models: logistic, lda, lasso, ridge, svmLinear, svmRadialWeighted, svmRadial, s
vmPoly, gbm, knn, treebag, stepLDA, qda, rf
Number of resamples: 10

ROC
                       Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
logistic          0.7618234 0.7766382 0.8106590 0.8073305 0.8339031 0.8564103
lda               0.7555556 0.7776353 0.8075561 0.8068793 0.8357550 0.8535613
lasso             0.7618234 0.7756410 0.8120339 0.8077209 0.8361823 0.8558405
ridge             0.7521368 0.7757835 0.8040629 0.8014109 0.8263533 0.8512821
svmLinear         0.7584046 0.7737892 0.8100830 0.8070518 0.8339031 0.8575499
svmRadialWeighted 0.7002849 0.7879892 0.8089929 0.8027493 0.8334758 0.8632479
svmRadial         0.6723647 0.7277778 0.8105413 0.7869095 0.8353276 0.8837607
svmPoly           0.7532764 0.7729345 0.8040753 0.8014149 0.8286325 0.8518519
gbm               0.6871795 0.7545584 0.7857550 0.7780502 0.8134203 0.8210826
knn               0.6774929 0.7086895 0.7454416 0.7453679 0.7885328 0.8040691
treebag           0.6780627 0.7277066 0.7490028 0.7458922 0.7647436 0.8008547
stepLDA           0.5527066 0.7451567 0.7800570 0.7534457 0.8000372 0.8157895
qda               0.6934473 0.7477208 0.7689459 0.7722132 0.8036325 0.8409357
rf                0.6447293 0.7435185 0.7653846 0.7563823 0.7928775 0.8007246
```

Here we see the (mean)average performance is near 80% with all models.

Note, most of the top preformining, in terms of mean, were models that are some form of linear regression or some for of linear relation to logit(y).

Below is the results of the ROC cross validation 10-fold plotted with three bloxplots of ROC curve, specificity and sensitivity. Note the ROC boxplots for all models have a tight spread, except for the Support Machine Radial model which has a wide range of model performance with the highest 3-rd quartile range.

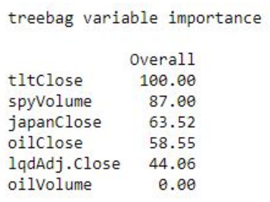


Also note that the medium Specificity is highest with QDA, but also has lowest median sensitivity. All models appear to better predict a market decline day rather than a positive change day. The sensitivity boxplot spreads suggest there is much more variability for all models in predicting a day change positive.

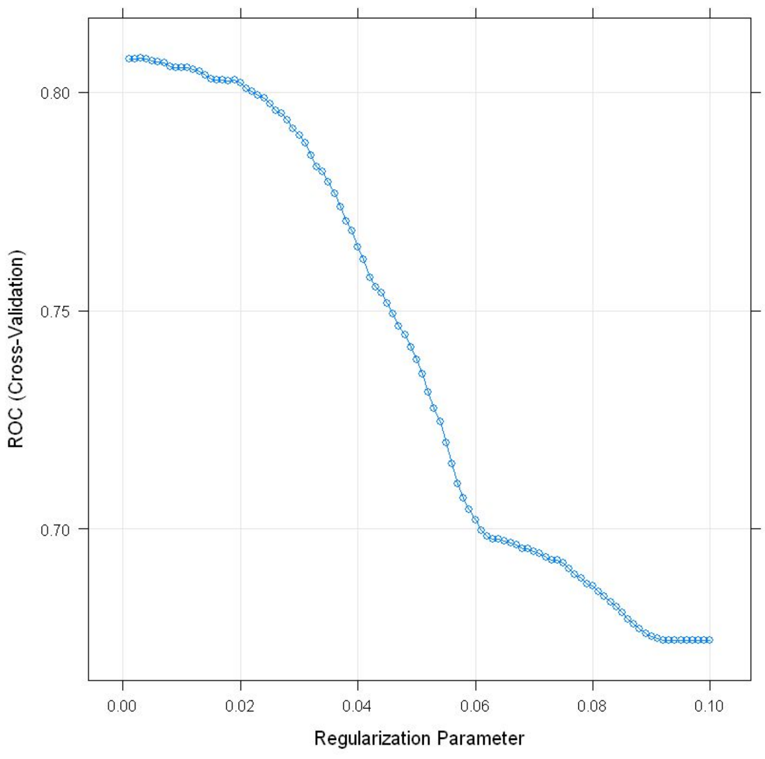
Deeper Look Into The Models

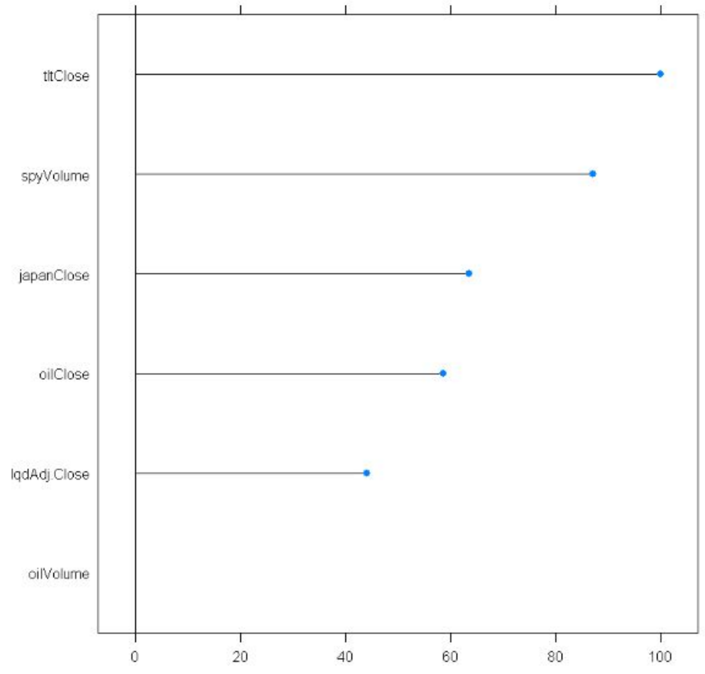
Many of these models were non-parametric, tuned or boosted that gave valuable information about the model and the insights to the data. Below is a brief overview:

- KNN It was KNN was on average, best with a tuning parameter of 15. Meaning nearest neighbor of 15 neighbors gave the best consistent results.
- The stepwise LDA model final only consisted of two steps and model of: y ~ lqdAdj.Close + tltClose, with little to no improvement. This is another indication of the Bond market(TLT reflecting U.S. government bonds and LQD reflecting corporate bonds).
- For Support vector machine Linear the best cost 1.157895.
- For lasso, the final value for lambda = 0.003, for ridge lambda = 0.012. For both cases lambda is rather small, which is essentially the least squares estimate.Below to the left is the results of cross validation ROC curve on the Y-axis the with regularization parameter on the X-axis.
- The below to the right is the feature importance of the bagged tree model. It is interesting to note that the lqdAdj.close is the second lowest on the list, compared to other models that valued it as the second important.



```
treebag variable importance

             Overall
tltClose      100.00
spyVolume      87.00
japanClose     63.52
oilClose       58.55
lqdAdj.Close   44.06
oilVolume       0.00
```





Overall Model Performance

Below are the performance results of the models used on the test data. Note that Positive Predictive is the number the model predicted correct as the numerator and the number of guesses the model predicted as positive as the denominator.

MODEL	Positive Predictive	Negative Predictive	Sensitivity	Specificity	TEST RESULTS
Generalized Boosted Regression	0.6603	0.7304	0.6519	0.7376	0.7
K-Nearest Neighbor	0.6739	0.7072	0.5886	0.7772	0.6944
Lasso Regression	0.6447	0.7115	0.6203	0.7327	0.6833
Logistic Regression	0.6471	0.7150	0.6266	0.7327	0.6861
Ridge Regression	0.6490	0.7129	0.6203	0.7376	0.6861
Support Vector Machine Linear	0.6452	0.7171	0.6329	0.7277	0.6861
Support Vector Machine Polynomial	0.6398	0.7236	0.6519	0.7129	0.6861
Support Vector Machine Radial (Weighted)	0.6552	0.7070	0.6013	0.7525	0.6861
Linear Discriminant Analysis	0.6414	0.6977	0.5886	0.7426	0.675
Random Forest	0.6351	0.6981	0.5949	0.7327	0.6722
Quadratic Discriminant Analysis	0.6026	0.6794	0.4810	0.8218	0.6722
Tree (Bagged)	0.6122	0.6808	0.5696	0.7178	0.6528
Step-Wise Linear Discriminant Analysis	0.6786	0.6694	0.5759	0.7030	0.6472

It should be noted that all models fell in a similar range. This is highly interesting as usually some models perform much better than others(parametric vs non-parametric, linear models VS non linear models). Here everything is following into a similar range.

Comparing this to the k-fold cross validation results, there are also some unexpected results. KNN which was one of the poorest performers in the cross validation results was almost the best performer on the test data. While the stepwise LDA was the poorest performer.

Though the test results average all fell into a similar range, there is a larger range in the Sensitivity and Specificity of these models. Most notably QDA had the highest prediction accuracy in terms of Specificity, but the worst in terms of Sensitivity.

It is also interesting to note that the cross validation data had a much higher performance profile and possibly was a little overfitted to the data. Maybe it would have been better to choose a k-fold of 8 or lower.

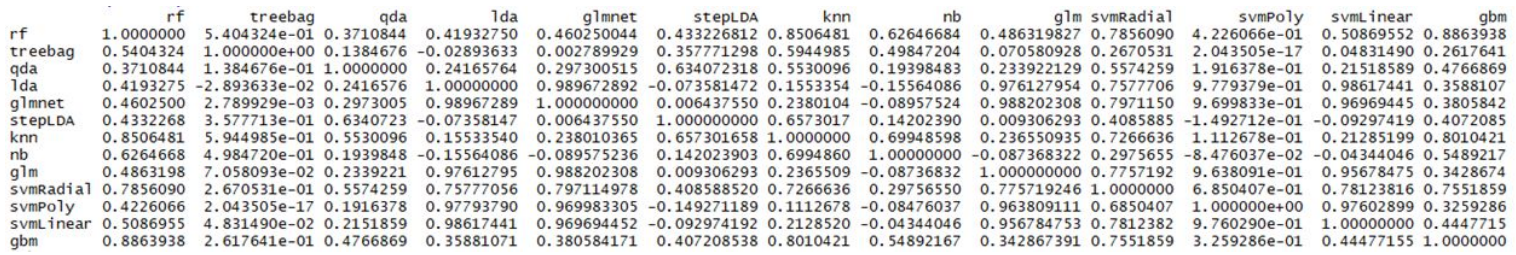
The best performing model was a gradient boosted regression tree. One of the largest advantages to this model is its predictive accuracy, hyper flexible and can optimize on different loss functions, but a problematic disadvantage to this model is it is less interpretable than others.

Highest Possible Prediction:Ensemble

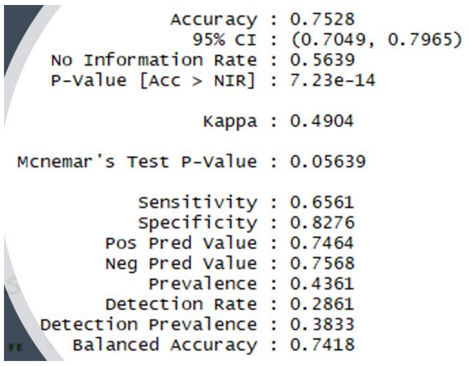
Since one of the main objectives was looking for the highest predictive model on the test data, other methods were explored. Usually to improve a model, tuning may help prediction accuracy, that already has been done for these methods. Three other common approaches involve combining the predictions of other models. Two of these approaches are:bagging (seen in tree bag model, this involves building many models from different subsamples of the training data) and boosting(seen in random forest and gradient boosted regression, which is building multiple models which each model learns aims to reduce the prediction errors of the last model in the gestalt of models). Finally the third technique is called stacking, which uses a base of other models using(usually different models, unlike the last two techniques described that use variations of the same type of predictive model), using the strengths of the base learners, to apply a model to that which can possibly create higher accuracy results.

Originally the idea of creating a hybrid stacked ensemble predictive model was to use models that had good prediction accuracy, that weren't highly correlated. Though this didn't create the highest accuracy in applied to the test data set.

Below is the correlation matrix of 10 fold cross validation results of different models.



```
                 rf       treebag       qda         lda       glmnet      stepLDA       knn          nb          glm svmRadial       svmPoly   svmLinear       gbm
rf        1.0000000  5.404324e-01 0.3710844  0.41932750  0.460250044  0.433226812 0.8506481  0.62646684  0.486319827 0.7856090  4.226066e-01  0.50869552 0.8863938
treebag   0.5404324  1.000000e+00 0.1384676 -0.02893633  0.002789929  0.357771298 0.5944985  0.49847204  0.070580928 0.2670531  2.043505e-17  0.04831490 0.2617641
qda       0.3710844  1.384676e-01 1.0000000  0.24165764  0.297300515  0.634072318 0.5530096  0.19398483  0.233922129 0.5574259  1.916378e-01  0.21518589 0.4766869
lda       0.4193275 -2.893633e-02 0.2416576  1.00000000  0.989672892 -0.073581472 0.1553354 -0.15564086  0.976127954 0.7577706  9.779379e-01  0.98617441 0.3588107
glmnet    0.4602500  2.789929e-03 0.2973005  0.98967289  1.000000000  0.006437550 0.2380104 -0.08957524  0.988202308 0.7971150  9.699833e-01  0.96969445 0.3805842
stepLDA   0.4332268  3.577713e-01 0.6340723 -0.07358147  0.006437550  1.000000000 0.6573017  0.14202390  0.009306293 0.4085885 -1.492712e-01 -0.09297419 0.4072085
knn       0.8506481  5.944985e-01 0.5530096  0.15533540  0.238010365  0.657301658 1.0000000  0.69948598  0.236550935 0.7266636  1.112678e-01  0.21285199 0.8010421
nb        0.6264668  4.984720e-01 0.1939848 -0.15564086 -0.089575236  0.142023903 0.6994860  1.00000000 -0.087368322 0.2975655 -8.476037e-02 -0.04344046 0.5489217
glm       0.4863198  7.058093e-02 0.2339221  0.97612795  0.988202308  0.009306293 0.2365509 -0.08736832  1.000000000 0.7757192  9.638091e-01  0.95678475 0.3428674
svmRadial 0.7856090  2.670531e-01 0.5574259  0.75777056  0.797114978  0.408588520 0.7266636  0.29756550  0.775719246 1.0000000  6.850407e-01  0.78123816 0.7551859
svmPoly   0.4226066  2.043505e-17 0.1916378  0.97793790  0.969983305 -0.149271189 0.1112678 -0.08476037  0.963809111 0.6850407  1.000000e+00  0.97602899 0.3259286
svmLinear 0.5086955  4.831490e-02 0.2151859  0.98617441  0.969694452 -0.092974192 0.2128520 -0.04344046  0.956784753 0.7812382  9.760290e-01  1.00000000 0.4447715
gbm       0.8863938  2.617641e-01 0.4766869  0.35881071  0.380584171  0.407208538 0.8010421  0.54892167  0.342867391 0.7551859  3.259286e-01  0.44477155 1.0000000
```



```
               Accuracy : 0.7528
                 95% CI : (0.7049, 0.7965)
    No Information Rate : 0.5639
    P-Value [Acc > NIR] : 7.23e-14

                  Kappa : 0.4904

 Mcnemar's Test P-Value : 0.05639

            Sensitivity : 0.6561
            Specificity : 0.8276
         Pos Pred Value : 0.7464
         Neg Pred Value : 0.7568
             Prevalence : 0.4361
         Detection Rate : 0.2861
   Detection Prevalence : 0.3833
      Balanced Accuracy : 0.7418
```

The base learners that were used to make the highest predictive ensemble model were knn, stepLDA, random forest, support vector machine with radial kernel, support vector machine with polynomial kernel, support vector machine with linear kernel and gradient boosted regression. Then the model put on top of that base of learners was a support vector machine with a radial kernel. Many different combinations of base learners were used, the highest predictive ensemble of base models consisted of uncorrelated learners in combination of highly accurate models(that were correlated with each other). The highest accuracy achieved was 0.7528 on the test data. It should be noted that its sensitivity was not noticeable higher than any other the methods from above, but the specificity was abnormally higher then the average specificity of models tested before.

Discussion/Conclusion

Some expected and unexpected findings were found. Changes in the bond market were the most influential for predicting changes in the S&P 500. U.S. treasuries as represented by the ETF TLT had the most predictive power. It is also interesting to note that yesterday's S&P 500 volume, yesterday's TLT volume and yesterday's LQD volume are correlated. This might give validity to the idea that when bond prices go down, investors move their money from the bond market to the stock market--while the reverse is also true.

Also another interesting insight to the data's relation to S&P is going up to 9 variables for a model, according to the subset selection method involved, of the 9 features 3 should be associated to TLT and if the model consisted of 6 features two should be TLT open and TLT close. What is most interesting is TLT open is a powerful feature. Something that would have easily been overlooked if one was using common sense rather than data.

The relation to Y with the features are also interesting to look at. When first looking at the way the train data performed with the 10 fold cross validation results, the parametric models with a data form of linearity relation to logit(y)(i.e. logistic regression, LDA and QDA) seemed to perform surprisingly well against non-parametric models that are considered to be highly better predictors and less so in interpreters.

Also highly interesting is the rate of specificity of QDA on the training cross validation(the box plot spread was small and medium higher than all other models) while much higher specificity compared to all other models used on the test data. This does tell us that the S&P 500 decline days are associated with the assumption that the observations are drawn from a Gaussian distribution with each feature having its own covariance matrix, and that the decline days fall well into quadratic decision boundary. Though QDA did not do as well at predicting true positives, it should be noted that its negative predictive value is in range with all other models.

Continuing the examination of the data relation to logit(y), the fact that the performance of lasso regression, ridge regression and logistic performed similarly in terms of accuracy, with the tuning parameters of lasso and ridge being so close to zero, there is a question: what if logistic regression was regularized? In the midst of writing this conclusion, this was tested(and is attached in appendix C). The result was surprising, with the highest model accuracy(not comparing it to the ensemble hybrid method). It performed 0.7111 level of accuracy, a little more than a 1% level of accuracy than the gradient boosted regression. Again very surprising and further supports the idea that the data is linearly related to logit(y).

Another important point that might be interesting to take into account is that the prediction is of the S&P 500 change from open to close which is 6.5 hrs(9:30 a.m. to 4 p.m. Eastern Standard Time). During this time a lot can and usually does affect the state of the S&P 500. Such breaking news events very often can affect the direction of the S&P 500, such as the President announcement of trade tariffs or the Federal Reserve President hinting at an increase in interest rates. These events are not taken into account here so this is a level of randomness in the data. Getting a prediction rate higher than 0.9 with this data set would be questionable at best. In further exploration it would be more interesting (and profitable) to aim to only predict the first hr movement of the market. This may make the features a better predictor while eliminating noise and randomness.

The best accuracy achieved was 0.7528. It is interesting to consider that the techniques here were primitive compared to a multimillion dollar algorithmic trading hedge fund. The data set was also extremely primitive in comparison(to minute to minute data, live satellite images of commodities, more markets and banking information with better details). Knowing that the financial markets are around 80% of algorithmic trading of these multi million dollar algorithmics(hardware, software, and team of mathematical plus financial academics) brings an interesting light into what the financial stock markets are made of today.

References

1. Mahmoodzadeh, Soheil and Tseng, Michael, Spot Arbitrage in FX Market and Algorithmic Trading: Speed is Not of the Essence (March 25, 2019). Available at SSRN: https://ssrn.com/abstract=3039407 or http://dx.doi.org/10.2139/ssrn.3039407
2. Kirilenko, A. A., & Lo, A. W. (2013). Moore's law versus murphy's law: Algorithmic trading and its discontents. Journal of Economic Perspectives, 27(2), 51-72.
3. https://www.ishares.com/us/products/239454/ishares-20-year-treasury-bond-etf
4. https://www.ishares.com/us/products/239566/ishares-iboxx-investment-grade-corporate-bond-etf
5. https://www.ssga.com/us/en/individual/etfs/funds/spdr-sp-500-etf-trust-spy
6. https://www.investopedia.com/terms/a/adjusted_closing_price.asp

APPENDIX A DATA CLEANING(Note this portion was done in python)

```
import csv
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

spy = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/SPY.csv' ,
parse_dates=["Date"], index_col="Date")
gold = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/GC=F.csv',
parse_dates=["Date"], index_col="Date")
crudeOil = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/CL=F.csv' ,
parse_dates=["Date"], index_col="Date")
london =
pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/0P0000WN79.L.csv',
parse_dates=["Date"], index_col="Date")
japan = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/^N225.csv',
parse_dates=["Date"], index_col="Date")
#London's data isn not complete So I am not going to use



btc = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/BTC-USD.csv' ,
parse_dates=["Date"], index_col="Date")
ethereum =
pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/ETH-USD.csv',
parse_dates=["Date"], index_col="Date")
euroUSD =
pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/EURUSD=X.csv' ,
parse_dates=["Date"], index_col="Date")
tlt = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/TLT.csv',
parse_dates=["Date"], index_col="Date")
#Etheurm is too new of a data set for this to be substantailly helpful

lqd = pd.read_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/LQD.csv',
parse_dates=["Date"], index_col="Date")
```

```
spy.columns = [('spy'+ 'Open'), ('spy'+'High'), ('spy'+'Low'),
            ('spy'+'Close'), ('spy'+'Adj Close'), ('spy'+'Volume')]

def changeName(string, df):
   df.columns = [(string+ 'Open'), (string+'High'), (string+'Low'),
            (string+'Close'), (string+'Adj Close'), (string+'Volume')]
   return df

gold=changeName('gold', gold)
crudeOil=changeName('oil', crudeOil)
japan=changeName('japan', japan)
btc=changeName('btc', btc)
lqd=changeName('lqd', lqd)
tlt=changeName('tlt', tlt)
euroUSD=changeName('euroUSD', euroUSD)


gold.info()#help to see what numbers are there in terms of null and not null
print('Gold above')
ethereum.info()
print('Etherum above')
tlt.info()
euroUSD.info()
crudeOil.info()

###result = pd.merge(spy, gold, right_index=True, left_index=True)#Works
#Need to make this proccess cleaner because this can get very sloppy mixed up and confused

merg1 = pd.merge(spy, gold, right_index=True, left_index=True)
merg2 = pd.merge(crudeOil, japan, right_index=True, left_index=True)

merg3 = pd.merge(btc, lqd, right_index=True, left_index=True)
merg4 = pd.merge(tlt, euroUSD, right_index=True, left_index=True)

result1=pd.merge(merg1, merg2, right_index=True, left_index=True)
result2=pd.merge(merg3, merg4, right_index=True, left_index=True)


result=pd.merge(result1, result2, right_index=True, left_index=True)

sns.heatmap(result.isnull(), cbar=False)

result.to_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/2015to2020NoNews/Na
me.csv')
```

```
precentage=result
precentage=precentage.pct_change(fill_method='ffill')
precentage.to_csv('C:/Users/garre/Desktop/DataPredictionModel2015to2019/2015to2020NoNe
ws/NamePrecent.csv')
```

APPENDIX B(the rest of the code is in R)

```
library(mlbench)
library(corrplot)
library(ggvis)
library(randomForest)
library(leaps)
library(caTools)
library(pROC)
library(tree)
library(caret)
library(caretEnsemble)



#Importing CVS, this will need to be change for your own personal computer
#This CVS can be found within the github link on the first page of the report document
realDataSet <- read.csv(file =
'C:/Users/garre/Desktop/DataPredictionModel2015to2019/2015to2020NoNews/Name.csv')
precentage<- read.csv(file =
'C:/Users/garre/Desktop/DataPredictionModel2015to2019/2015to2020NoNews/NamePrecent.cs
v')



#####Creates if there was a change in SPY
###1 above zero, 0 for below zerp
##Used for categeorical Prediction
#precentage$change=ifelse(precentage$spyClose>0,1,0)

size<-nrow(precentage)
precentage<-precentage[2:size,]#getting rid of first row, we are using "yesterday's data", for
tomorrow's spy change
precentage$change=ifelse(precentage$spyClose>0,1,0)
#Scaling data
newPrecentScaled=as.data.frame(scale(precentage[,c(-1,-50)]))
newPrecentScaled$change= precentage$change
#head(newPrecentScaled)


#CORRPLOT FOR EVERYTHING
newdf<-newPrecentScaled[ , !(names(newPrecentScaled) %in% drops)]
CorrelationTEster2<-data.frame(newdf)
```

```
corlChart<-cor(CorrelationTEster2)
corrplot(corlChart, method = "circle")


#SUBSET Selection Analysis
drops <- c('spyOpen', 'spyHigh', 'spyLow' ,'spyClose', 'spyAdj.Close')
subsetting<-newdf[ , !(names(newdf) %in% drops)]
regit.full=regsubsets(change~., subsetting, nvmax=19)
summary(regit.full)

#plot(regit.full)
sumOfRegSub=summary(regit.full)
par(mfrow=c(2,2))
plot(sumOfRegSub$rss,
   xlab= "Num of Variables",
   ylab= "RSS",
   col="blue",
   type="b")


plot(sumOfRegSub$adjr2,
   xlab= "Num of Variables",
   ylab= "Adjusted R^2",
   col="red",
   type="b")



#############
####Comparing Variables
##############




SEPARATING DATA

Up=subset(newPrecentScaled,change==1)
Down=subset(newPrecentScaled,change==0)
rows4fiftyfity1=sample(1:nrow(Up),420,replace=FALSE)
rows4fiftyfity2=sample(1:nrow(Down),420,replace=FALSE)

train=rbind(Up[rows4fiftyfity1,],Down[rows4fiftyfity2,])

####Splitting the rest of validation and training data
```

```
remainingData=rbind(Up[-rows4fiftyfity1,],Down[-rows4fiftyfity2,])

#validation <- remainingData[sample(1:nrow(remainingData), 180, replace=FALSE),]
#testSet=rbind(remainingData[-validation,])


validate_rows = sample(1:nrow(remainingData),180,replace=FALSE)

validation = remainingData[validate_rows,]

test=remainingData[-validate_rows,]



library(caret)


class(newPrecentScaled$change)

newPrecentScaled$change = as.factor(newPrecentScaled$change)


set.seed(123456)

rows= sample(1:nrow(newPrecentScaled),size = 0.7*nrow(newPrecentScaled),
replace=FALSE)

train = newPrecentScaled[rows,]
test = newPrecentScaled[-rows,]
#names(train)
train<-train[,c(6, 16, 18, 22, 35, 40, 49)]
test<-test[,c(6, 16, 18, 22, 35, 40, 49)]
head(train)


#4 Parameter from Subset Information
#train<-train[,c(6, 16, 22, 40, 49)]
#test<-test[,c(6, 16, 22, 40, 49)]

trainX<-train[,-5]
trainY<-train[,5]
head(train)

seed<-7
```

```
#############
######PCA EVALUATION
#############
#names(newPrecentScaled)
prComptestg= prcomp(test[,2:4],scale=TRUE,center=TRUE)
prComp= prcomp(train[,2:4],scale=TRUE,center=TRUE)

prComptestoIL= prcomp(test[,5:8],scale=TRUE,center=TRUE)

prCompOIL= prcomp(train[,5:8],scale=TRUE,center=TRUE)

prComptestTL= prcomp(test[,11:12],scale=TRUE,center=TRUE)

prCompTL= prcomp(train[,11:12],scale=TRUE,center=TRUE)


prComp$sdev^2
cumsum(prComp$sdev^2/sum(prComp$sdev^2))
expl.var <- round(prComp$sdev^2/sum(prComp$sdev^2)*100) # percent explained variance
expl.var
std_dev <- prComp$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
sum(prop_varex[1:5])
#pred <- predict(pca, newdata=newPrecentScaled[,1:4])

#plot(pred)

#pca$stdev^2
plot(cumsum(prop_varex), xlab = "Principal Component",ylab = "Cumulative Proportion of
Variance Explained",type = "b")
abline(h=0.975,col='red',v=30)

test.data<-data.frame(classe = test, prComptestg$x)
train.data<-data.frame(classe = train, prComp$x)

test.data<-data.frame(test.data, prComptestoIL$x)
train.data<-data.frame(train.data, prCompOIL$x)

test.data<-data.frame(test.data, prComptestTL$x)
train.data<-data.frame(train.data, prCompTL$x)
```

```
#prComp
#prCompOIL
#prCompTL


names(train.data)

pcaTrain <- train.data[,c(1, 9,10,13, 18,22)]
pcaTest <- test.data[,c(1, 9,10,13, 18,22)]
names(pcaTrain)



pcaTest$classe.change<-as.factor(pcaTest$classe.change)
pcaTrain$classe.change<-as.factor(pcaTrain$classe.change)

mytrain=trainControl(method="repeatedcv",number=10, repeats=10)
set.seed(7)
model_logistic= train(classe.change~., data=pcaTrain,
                  method="glm",
                  preProcess = c("center", "scale"),
                  trControl=mytrain)

set.seed(7)
model_knn= train(classe.change~., data=pcaTrain,
              method="knn",
              preProcess = c("scale", "center"),
              trControl=mytrain,
              tuneLength = 13)
```

APPENDIX C

```
##############
#########TRAINING OF ALL MODELS
##################

seed<-7

mytrain=trainControl(method="repeatedcv",number=10, repeats=10)
set.seed(seed)
model_logistic= train(change~., data=train,
                  method="glm",
                  preProcess = c("center", "scale"),
                  trControl=mytrain)
set.seed(seed)
model_knn= train(change~., data=train,
              method="knn",
              preProcess = c("scale", "center"),
              trControl=mytrain,
              tuneLength = 15)

set.seed(seed)



grid <- expand.grid(sigma = c(.01, .015, 0.2),
                C = c(0.75, 0.9, 1, 1.1, 1.25)
)


svmGrid <- expand.grid(sigma= 2^c(-25, -20, -15,-10, -5, 0), C= 2^c(0:5))

model_svm= train(change~., data=train,
              method="svmRadial",
              trControl=mytrain,
              preProcess = c("center", "scale"),
              tuneLength = 10,
              #tuneGrid=grid
              tuneGrid=svmGrid
             )
```

```
set.seed(seed)
model_lda <- train(change~.,
               data=train,
               method="lda",
               trControl=mytrain)

set.seed(seed)
model_qda <- train(change~.,
               data=train,
               method="qda",
               trControl=mytrain)

set.seed(seed)
gbmGrid <-  expand.grid(interaction.depth = c(1, 5, 9),
                    n.trees = (1:30)*50,
                    shrinkage = 0.1,
                    n.minobsinnode = 20)
model_gbm <- train(change~., data=train,
                method="gbm",
                trControl=mytrain,
                tuneGrid= gbmGrid)


set.seed(seed)
model_rf <- train(change~., data=train, method="rf", trControl=mytrain)

# Random Search
control <- trainControl(method="repeatedcv", number=10, repeats=10, search="random")
set.seed(seed)
model_rf <- train(change~., data=train, method="rf",
               metric="Accuracy",
               tuneLength=8,
               #tuneGrid=grid,
               trControl=control)


set.seed(seed)
model_treebag <- train(change~., data=train, method="treebag", trControl=mytrain)


set.seed(seed)
model_svmPoly <- train(change~., data=train, method="svmPoly", trControl=mytrain)
```

```
set.seed(seed)
#tune.grid.svmLinear<- expand.grid (C=c( 1 ,3 ,5 ,7),
#           sigma = c(0.0005 ,0.001 ,0.005 ,0.01 ,0.05) )

model_svmLinear <-train(change~., data=train, method="svmLinear",  trControl =mytrain,
           preProcess = c("center","scale"),
            tuneGrid = expand.grid(C = seq(0, 2, length = 20)))



set.seed(seed)
model_stepLDA <- train(change~.,
                   data=train,
                   method="stepLDA",
                    tuneGrid= model_stepLDA$bestTune,
                    preProc=c("center", "scale"),
                   trControl=mytrain)

set.seed(seed)
model_stepLDA2 <- train(change~.,
                   data=train,
                   method="stepLDA",
                    tuneGrid= model_stepLDA$bestTune,
                    preProc=c("center", "scale"),
                   trControl=mytrain)



svmGrid <- expand.grid(sigma= 2^c(-25, -20, -15,-10, -5, 0), C= 2^c(0:5))
tune.grid.svmRdialWeights<- expand.grid ( C = c( 1 ,3 ,5 ,10 ,20) ,
          Weight = c(0.1 ,0.5 ,1 ,2 ,3 ,5 ,10) ,
          sigma = c(0.0005 ,0.001 ,0.005 ,0.01 ,0.05) )

model_svm= train(change~., data=train,
              method="svmRadialWeights",
              trControl=mytrain,
              preProcess = c("center", "scale"),
              tuneLength = 10,
              #tuneGrid=grid
              tuneGrid=tune.grid.svmRdialWeights
             )
set.seed(seed)
model_lasso <- train(change~., data=train, method = "glmnet",
                        trControl = mytrain,
                        nlambda = 25,
                        metric="Accuracy",
                        #metric = "ROC",
                        tuneGrid = expand.grid(alpha = 1,
                                           lambda = seq(0.001,0.1,by = 0.001)))

#########

set.seed(seed)
model_ridge <- train(change~., data=train, method = "glmnet",
                        trControl = mytrain,
                        nlambda = 25,
                        metric="Accuracy",
                        #metric = "ROC",
                        tuneGrid = expand.grid(alpha = 0,
                                           lambda = seq(0.001,0.1,by = 0.001)))


set.seed(seed)
model_stepLDA <- train(change~.,
                   data=train,
                   method="stepLDA",
                    #tuneGrid= model_stepLDA$bestTune,
                    preProc=c("center", "scale"),
                   trControl=mytrain)

model_list = list(logistic = model_logistic,
               qda=model_qda,
               lda=model_lda,
               lasso=model_lasso,
               svmLinear=model_svmLinear,
               svmRadial=model_svm,
               svmPoly=model_svmPoly,
               gbm=model_gbm,
               knn= model_knn,
               treebag=model_treebag,
               ridge=model_ridge,
                stepLDA=model_stepLDA,
               fit.rf=fit.rf
               )

summary(result)
bwplot(result)
dotplot(result)
ggplot(result)
```

```
#############
#######ROC plots
##############


predict_gbm<-predict(model_gbm,test,type="prob")[,2]#so first get the predicted probabilities of
the final model
#we use type = "prob" because its the probabilities we need to get an ROC curve
predict_lasso<-predict(model_lasso,test,type="prob")[,2]


predict_ridge<-predict(model_ridge,test,type="prob")[,2]
predict_knn<-predict(model_knn,test,type="prob")[,2]
predict_svm<-predict(model_svm,test,type="prob")[,2]
predict_logistic<-predict(model_logistic,test,type="prob")[,2]
predict_rf<-predict(model_rf,test,type="prob")[,2]
predict_treebag<-predict(model_treebag,test,type="prob")[,2]
colAUC(cbind(predict_gbm,predict_knn,predict_ridge,predict_lasso,predict_logistic,predict_rf,pr
edict_treebag,predict_svm), test$change, plotROC=TRUE)


###########
######ROC confusion matrix of plots
##############

splom(result)


#############
#######Other results from models
#################

set.seed(seed)
model_treebag <- train(change~., data=train, method="treebag", trControl=mytrain)
varImp(model_treebag )
plot(varImp(model_treebag ))


plot(model_svmLinear)
model_svmLinear$bestTune


plot(model_lasso)
```

```
tree.stocks= tree(change~., data=train)
summary(tree.stocks)
plot(tree.stocks)



###############
#######Attaining results of models on test data
###############
predict_logistic  = predict ( model_logistic, test)
confusionMatrix(predict_logistic, test$change,positive = '1' )











############
##########SUPER SPECIAL CASE
##########regularized logistic regression
#########Best performer as mentioned in the Conclusion

set.seed(7)
model_logistic= train(change~., data=train,
                      method="glm",
                      family=binomial(link="logit"),
                      preProcess = c("center", "scale"),
                      trControl=mytrain)


predict_regularizedLOGISITC  = predict ( model_logistic, test)

confusionMatrix(regularizedLOGISITC, test$change )





##################
#######PCA
```

```
##################

set.seed(123456)

rows= sample(1:nrow(newPrecentScaled),size = 0.7*nrow(newPrecentScaled),
replace=FALSE)

train = newPrecentScaled[rows,]
test = newPrecentScaled[-rows,]
#names(train)
train<-train[,c(6, 7, 9, 10, 13, 16,17, 18, 22, 35, 37, 40, 49)]
test<-test[,c(6, 7, 9, 10, 13, 16,17, 18, 22, 35, 37, 40, 49)]
names(train)

#names(newPrecentScaled)
prComptestg= prcomp(test[,2:4],scale=TRUE,center=TRUE)
prComp= prcomp(train[,2:4],scale=TRUE,center=TRUE)

prComptestoIL= prcomp(test[,5:8],scale=TRUE,center=TRUE)

prCompOIL= prcomp(train[,5:8],scale=TRUE,center=TRUE)

prComptestTL= prcomp(test[,11:12],scale=TRUE,center=TRUE)

prCompTL= prcomp(train[,11:12],scale=TRUE,center=TRUE)

prComp$sdev^2
cumsum(prComp$sdev^2/sum(prComp$sdev^2))
expl.var <- round(prComp$sdev^2/sum(prComp$sdev^2)*100) # percent explained variance
expl.var
std_dev <- prComp$sdev
pr_var <- std_dev^2
prop_varex <- pr_var/sum(pr_var)
sum(prop_varex[1:5])
#pred <- predict(pca, newdata=newPrecentScaled[,1:4])

#plot(pred)
```

```
#pca$stdev^2
plot(cumsum(prop_varex), xlab = "Principal Component",ylab = "Cumulative Proportion of
Variance Explained",type = "b")
abline(h=0.975,col='red',v=30)

test.data<-data.frame(classe = test, prComptestg$x)
train.data<-data.frame(classe = train, prComp$x)

train.data<-data.frame(train.data, prCompOIL$x)

#test.data<-data.frame(classe = test, prComptest$x)
#train.data<-data.frame(classe = train, prComp$x)

#prComp
#prCompOIL
#prCompTL

names(train.data)

###########
######Ensemble
#############

mycontrol = trainControl(method="cv", number=10, classProbs=TRUE)

model_list = c("rf", "treebag", "qda", "lda", "glmnet", "stepLDA" , "knn", "nb", "glm", "treebag",
"svmRadial", "svmPoly", "svmLinear", "gbm") # list of algorithms

levels(train$change)=c("Up","Down")
View(train$change)
```

```
set.seed(seed)
models = caretList(change~., data=train, trControl=mycontrol, methodList=model_list)
results = resamples(models)

modelCor(results)


corrplot(cor(summary(results)$values),method="circle")
splom(results) #INTERESTING CORRELATION MATRIX OF ALL MODELS




##########
####Best predictive model
###############
model_list = c("knn", "stepLDA", "rf", "svmRadial", "svmPoly", "svmLinear", "gbm" ) # list of
algorithms
set.seed(seed)
models = caretList(change~., data=train, trControl=mycontrol, methodList=model_list)
results = resamples(models)
modelCor(results)
summary(results)
# stack using glm
stackControl <- trainControl(method="repeatedcv", number=10, repeats=3,
savePredictions=TRUE, classProbs=TRUE)
set.seed(seed)
stack.glm <- caretStack(models, method="svmRadial", metric="Accuracy",
trControl=stackControl)
print(stack.glm)
cf<-predict(stack.glm, test)
levels(cf)=c("Up","Down")
levels(test$change)=c("Up","Down")
confusionMatrix(cf, test$change)
```