IST 707 Final Project
Market Movement Prediction

Group 2
Barry Murthy, Tamilselvan Tamilmani, Winnie (Qing) Cai

1 INTRODUCTION

An informationally efficient capital market is one in which the current security price fully, quickly, and rationally reflects all available information about that security. An academic might say, "Given all available information, current securities prices are unbiased estimates of their values, so that the expected return on any security is just the equilibrium return necessary to compensate investors for the risk regarding its future cash flows." Simply put, "You can't beat the market." Investors should use a passive investment strategy in a perfectly efficient market because active investment strategies will underperform due to transaction costs and management fees. So here are our questions: How efficient is the market? Can we beat the market?

Professor Eugene Fama originally developed the market efficiency concept and identified three market efficiency forms: weak-form, semi-strong form, and strong-form market efficiency. The weak form of the efficient market hypothesis (EMH) states that the current security price fully reflects all currently available security market data. Thus, past price and volume (market) information will have no predictive power about security prices' future direction. Price changes will be independent of one period to the next. An investor cannot achieve positive risk-adjusted returns on average in a weak-form efficient market using technical analysis. We attempted to explore and test the weak form of the efficient market hypothesis through this project by applying two different approaches and a series of machine learning algorithms.

2 Analysis & Models

2.1 About the Data

We sourced our data directly from Yahoo Finance via the 'quantmod' package due to its easy accessibility and data accuracy. We selected 'SPY,' which is the ticker of 'SPDR S&P 500 ETF Trust'. S&P 500 index is commonly used as the proxy for the U.S. equity market, which consists of the 500 largest companies in the U.S.

Our data timeline ranges from January 1st, 2001 to January 1st, 2021, which consists of 20 years' worth of daily market data, including historical pricing quotes and trading volume. We selected this timeline to cover the minimum of one complete economic cycle, typically lasting seven to ten years. However, since the last financial crisis when the market touched the bottom in 2009, we have experienced the longest bull market in the past twelve years. To avoid skewness, we extended our timeline to cover the previous economic downturn, driven by the dot-com bubble around 2002.

The initial data set includes 5032 observations and seven variables, some of which will be used to calculate the independent variables and the dependent variables.

	Open	High	Low	Close	Volume	Adjusted	date
2001-01-02	132.0000	132.1562	127.5625	128.8125	8737500	87.55441	2001-01-02
2001-01-03	128.3125	136.0000	127.6562	135.0000	19431600	91.76007	2001-01-03
2001-01-04	134.9375	135.4688	133.0000	133.5469	9219000	90.77238	2001-01-04
2001-01-05	133.4688	133.6250	129.1875	129.1875	12911400	87.80931	2001-01-05
2001-01-08	129.8750	130.1875	127.6875	130.1875	6625300	88.48900	2001-01-08
2001-01-09	131.0469	131.5000	129.4219	129.8438	5702400	88.25532	2001-01-09
2001-01-10	129.0000	132.1250	128.8125	132.1250	8746100	89.80594	2001-01-10
2001-01-11	131.0938	133.4844	131.0938	132.2500	7245100	89.89088	2001-01-11
2001-01-12	132.6875	133.7188	131.2812	132.0000	7244000	89.72096	2001-01-12
2001-01-16	132.0000	133.1875	131.5156	132.8438	8542200	90.29446	2001-01-16
2001-01-17	134.8438	135.0469	132.6406	133.4531	7851400	90.70870	2001-01-17

=

FIGURE 1

2.2 Exploratory Data Analysis

We first plotted the price chart for 'SPY' based on 20 years' worth of daily close price. The chart shows that the market touched the bottom in 2002, 2009, and early 2020, reflecting three economic downturns in history – Dot-com Bubble, Subprime Mortgage Crisis, and the Covid-19 Pandemic, respectively. The volume spikes further support the market trends in the 'Daily Volume' chart.

FIGURE 2

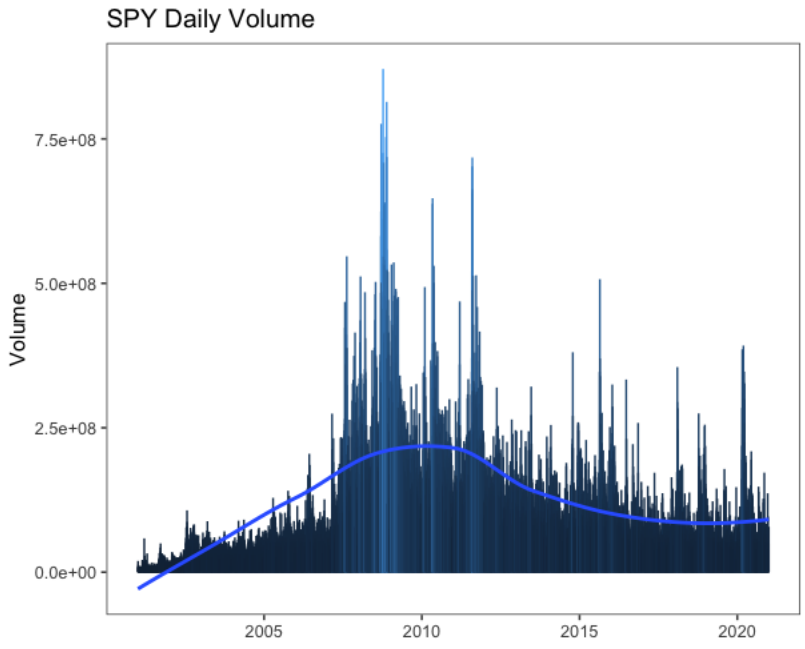


FIGURE 3

We aggregated and plotted ups and downs for pre-defined periods to spot patterns in the market movements. The only clear pattern shown in the below graphs is that there are

significantly more ups than downs which the extended bull market can explain since 2009. Anecdotally, even though we see more ups on Wednesdays, do we feel confident to only trade on Wednesdays, or should we take a closer look at the market?

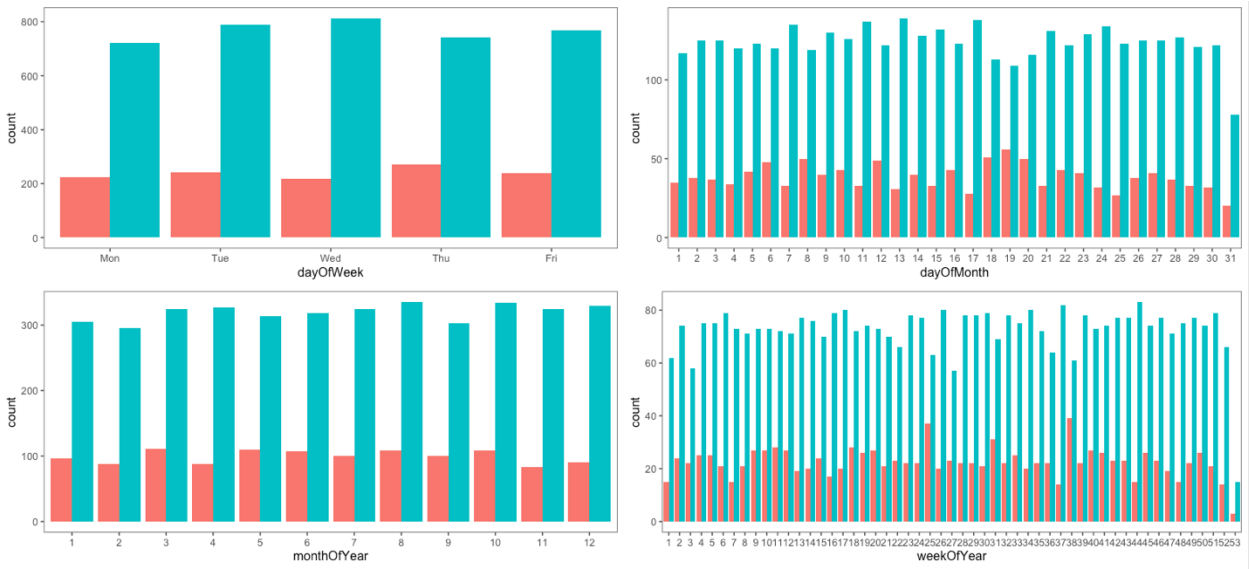


FIGURE 4

2.2.1 *APPROACH #1:* LAGGED RETURNS

We calculated a 1-day percentage return based on the daily close price. If the 1-day return is positive, we label it 'Up'; otherwise, 'down.' Based on the 'Up' or 'Down,' we created 'Class,' which will be used as the dependent variable.

We then calculated the percentage returns for 1-day, 2-day, 3-day, 4-day, and 5-day periods and lagged them by one day to avoid look-ahead bias. The lagged returns will be used as independent variables to predict 'Class.'

We also labeled day of the week, day of the month, day of the year, and week of the year for exploratory analysis purposes.

We graphed a confusion matrix based on the 1-day return and lagged returns. There seem to be clear positive correlations between lagged returns which market momentums can explain. The correlations between lagged returns and 1-day returns seem neutral or no clear correlations.

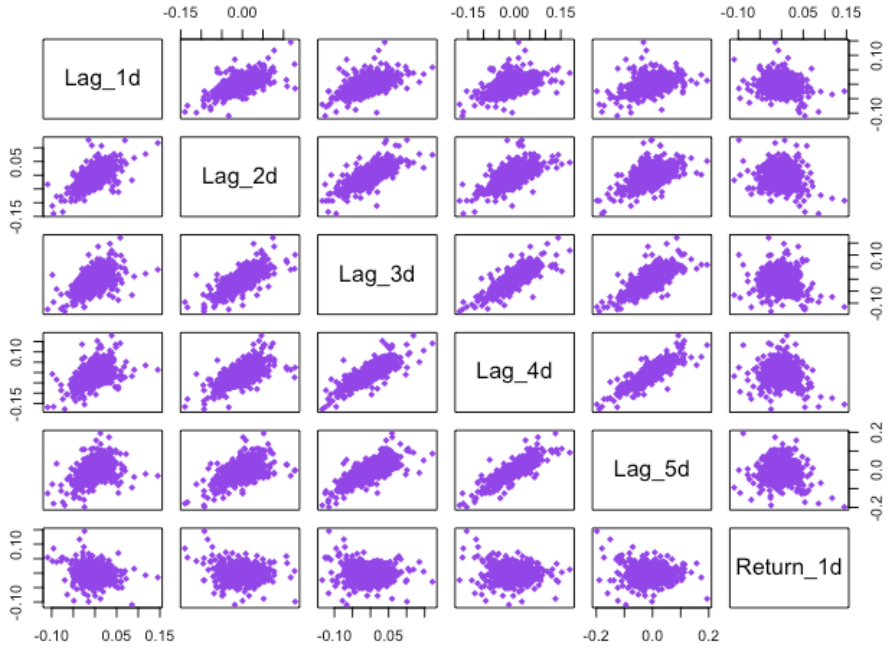


FIGURE 5

2.2.2 *APPROACH #2:* TECHNICAL INDICATORS

Using 'quantmod,' we calculated relative strength index (RSI), 5-day moving average (SMA), 50-day moving average (LMA), Welles Wilder's Directional Movement Index (ADX), which are all metrics that indicate market trends or the strength of the trends (either up or down). These four indicators are used together to predict 'Class.'

- *Moving average*: calculates the arithmetic mean of the series over the past n observations.

- *RSI*: ratio of the recent upward price movement to the absolute price movement. It measures the magnitude of recent price changes to evaluate overbought or oversold conditions in a stock or other asset price.

- *ADX*: a combination of two other indicators developed by Wilder, the positive directional indicator (+DI) and negative directional indicator (-DI). The ADX combines them and smooths the result with a smoothed moving average.

  To calculate +DI and -DI, one needs price data consisting of high, low, and closing prices each period (typically each day). One first calculates the directional movement (+DM and -D.M.):

  UpMove = today's high − yesterday's high

DownMove = yesterday's low − today's low

if UpMove > DownMove and UpMove > 0, then +DM = UpMove, else +DM = 0
if DownMove > UpMove and DownMove > 0, then -DM = DownMove, else -DM = 0

After selecting the number of periods (Wilder used 14 days initially), +DI and -DI are:
+DI = 100 times the smoothed moving average of (+DM) divided by the true average range
-DI = 100 times the smoothed moving average of (-D.M.) divided by the true average range

The smoothed moving average is calculated over the number of periods selected, and the true average range is a smoothed average of the true ranges.

Then: ADX = 100 times the smoothed moving average of the absolute value of (+DI − -DI) divided by (+DI + -DI)

A buy/sell signal is generated when the +/-DI crosses up over the -/+DI when the DX/ADX signals a strong trend. A high/low DX signals a strong/weak trend.

We graphed a confusion matrix based on the 1-day return and the technical indicators. There seem to be only clear positive correlations between short-term and long-term moving average which can be explained by the similar fashion in which the metrics are calculated and the market momentum. There seem to be no clear correlations between the indicators and 1-day return.

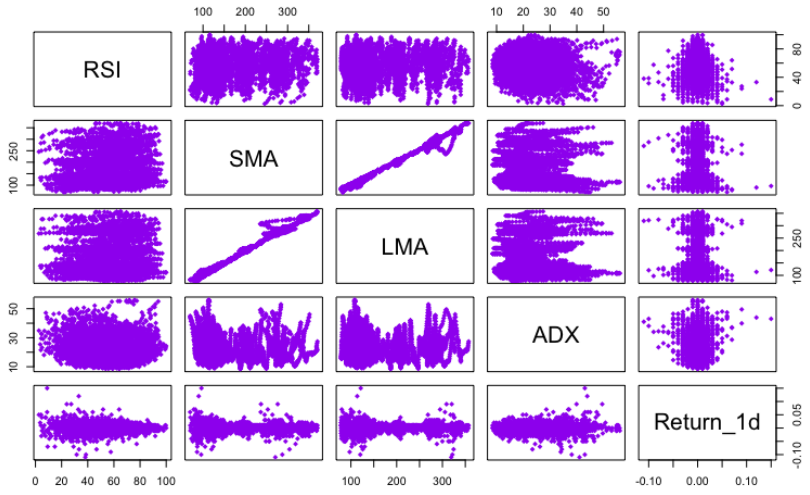


FIGURE 6

The below scatterplot and boxplot imply that one or two specific indicators may not help separate 'Up' from 'Down.'

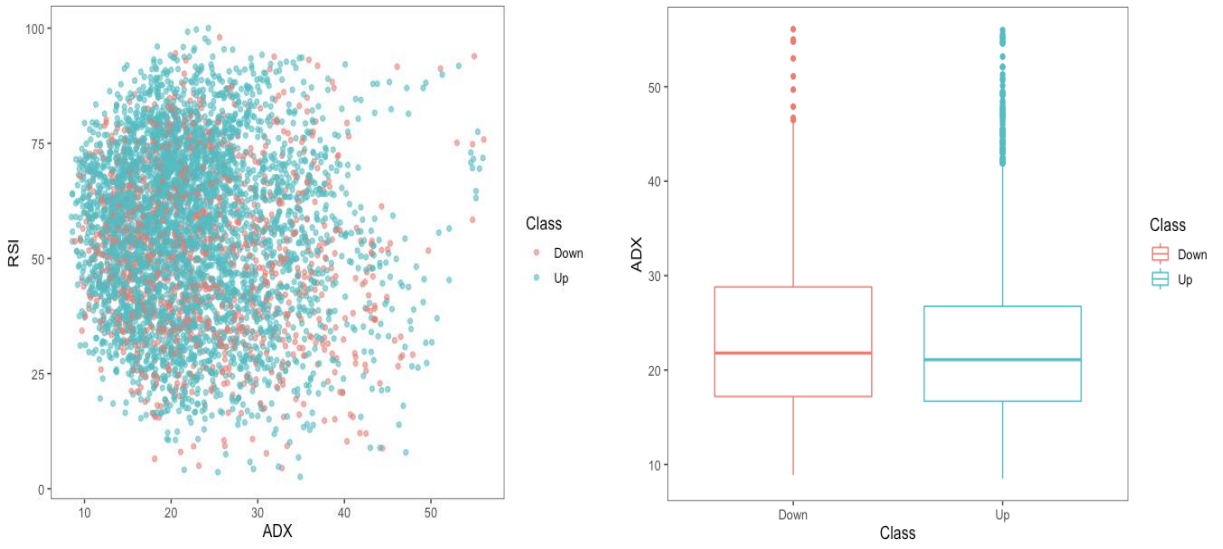


FIGURE 7

2.3 DATA PROCESSING

```
## Calculate lagged % returns
delta<-Delt(data$Close,k=1:5)


# Calculate lagged technical indicators
#RSI, SMA, LMA and ADX (Welles Wilder's Directional Movement Index)
closeprice <- data$Close

rsi <- round(RSI(closeprice, n = 14, maType="WMA"),1)
rsi <- c(NA,head(rsi,-1))   #lagged data

SMA<- 5
LMA <- 50

sma <- round(SMA(closeprice, SMA),1)
sma <- c(NA,head(sma,-1))

lma <- round(SMA(closeprice, LMA),1)
lma <- c(NA,head(lma,-1))


data_ADX <- data.frame (ADX(data[,c("High","Low","Close")]))
adx <- round(data_ADX$ADX,1)
adx <- c(NA,head(adx,-1))
```

After the dataset is divided into the training and testing dataset, we realized that the training data set was off-balance with the number of 'Up' outweighing 'Down.' To balance the data set, we applied the Adaptive Synthetic Sampling Approach, which is the process of generating minority data samples according to their distributions. See below graphs for before vs. after.

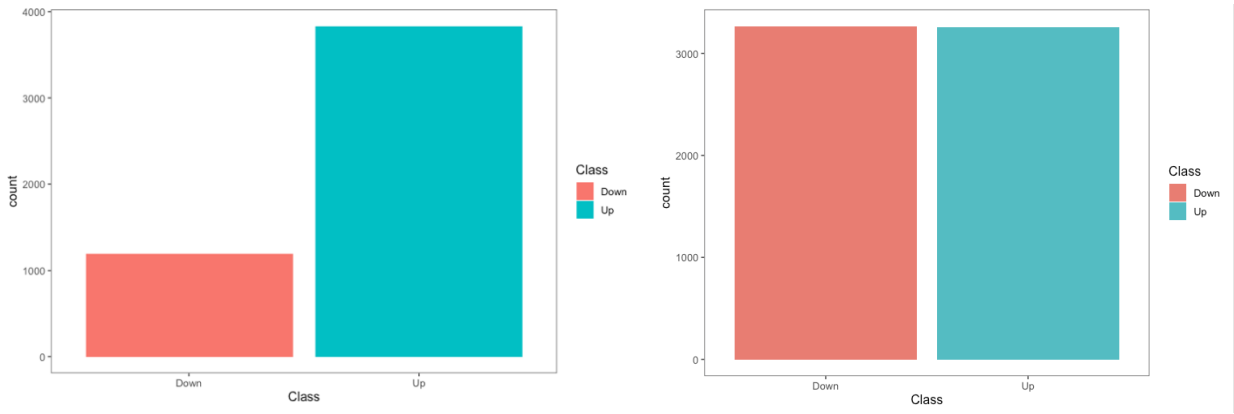


FIGURE 8

2.4 MODELS

2.4.1 MODEL TRAINING AND TESTING STRATEGY

For all of our models, we will be doing a holdout test and a k-fold cross-validation test. We will split the data before 2018 using the time for the holdout.

```
trainDF <-df1%>% filter(Date <= "2017-12-31")  #take until 2017 data
trainDF<-na.omit(trainDF)
trainDF<- select(trainDF,-c(Date))
testDF<- df1%>% filter(Date > "2017-12-31")  # take after 2017
testDF<-na.omit(testDF)
```

For the k-fold cross-validation, we will use 5-fold cross-validation and measure the average accuracy.

2.4.2 LAGGED RETURNS MODELS

2.4.2.1 ASSOCIATION RULES

We will discretize the continuous variables; all the lag returns so that we can group them.

```
edaDF1<-subset(df1, select=-(Date))
edaDF1$Lag_1d <- discretizeDF.supervised(Class ~ ., edaDF1[, c('Class', 'Lag_1d')], method='mdlp')$Lag_1d
edaDF1$Lag_2d <- discretizeDF.supervised(Class ~ ., edaDF1[, c('Class', 'Lag_2d')], method='mdlp')$Lag_2d
edaDF1$Lag_3d <- discretizeDF.supervised(Class ~ ., edaDF1[, c('Class', 'Lag_3d')], method='mdlp')$Lag_3d
edaDF1$Lag_4d <- discretizeDF.supervised(Class ~ ., edaDF1[, c('Class', 'Lag_4d')], method='mdlp')$Lag_4d
edaDF1$Lag_5d <- discretizeDF.supervised(Class ~ ., edaDF1[, c('Class', 'Lag_5d')], method='mdlp')$Lag_5d
edaDF1$dayOfWeek<-lubridate::wday(df1$Date,label = TRUE, abbr = TRUE)
edaDF1$monthOfYear<-lubridate::month(df1$Date,label = TRUE, abbr = TRUE)
```

```
transactionData1<- as(edaDF1,"transactions")
inspect(transactionData1)
itemFrequencyPlot(transactionData1,support=0.5)
arules_model1<-apriori(transactionData1,parameter = list(support=0.01,confidence=0.2))
summary(arules_model1)
quality(arules_model1)
#inspect(arules_model1)
plot(arules_model1,method = "grouped")
goodrules1 <- arules_model1[quality(arules_model1)$lift > 10]
plot(goodrules1,method="graph",engine="interactive")
#plot(goodrules, method = "grouped", interactive = TRUE)
plot(goodrules1, method = "grouped")
inspect(goodrules1)
```

2.4.2.2 K-Means

We will find out how many clusters to be divided from the data by using an unsupervised algorithm by their silhouette distance. We get around two distinct clusters, and we run the K-means clustering with two centers, and we look at how each is grouped

```
fviz_nbclust(select(kmeansDF,-c(Date,Class)),FUN=hcut, method="silhouette")

kmeans1 <- kmeans(select(kmeansDF,-c(Date,Class)),centers = 2,nstart=25,iter.max = 20)
summary(kmeans1)
plot(kmeansDF$Lag1d,kmeansDF$Lag5d,
     xlab = "Lagged 1 day",
     ylab = "Lagged 5 day",
     col = kmeans1$cluster)
points(kmeans1$centers,col= 1:200 , pch=8,cex=2)
```

2.4.2.3 Decision Tree Models

We will start with building a decision tree model with a holdout test. We will begin with the default complexity parameter zero and test out until one.

Then we will prune the tree to generalize based on the least cross-validated error. And we will finish with a 5-fold cross-validation test to measure the accuracy.

```
#Tree model
train.control <- trainControl(method = "cv", number=5,allowParallel = T)
tree_model1<-train(Class~.,
                   data=trainDF,
                   method="rpart",
                   trControl = train.control,
                   tuneGrid = expand.grid(cp = seq(0, 1, length = 10)))
plot(tree_model1)
print(tree_model1)
plotTree(tree_model1$finalModel)
fancyRpartPlot(tree_model1$finalModel,type=1)
tree_cm1<-predictClassModel(tree_model1,testDF,"raw")
summary(tree_model1$finalModel)
```

2.4.2.4 Naïve Bayes

We will then train a Naïve Bayes model for the train data set and look at the accuracy. We will follow the same text train split we did for the decision tree model

```
nb_model1 =  train(Class~., data = trainDF1, method = "nb",
                   trControl = train.control,
                   tuneLength=10)
#summary(model_nb1)
plot(nb_model1)
cm_nb1<-predictClassModel(nb_model1,testDF1,"raw")
```

2.4.2.5 K Nearest Neighbor Model

Then we will start training a kNN model, it's a slow model, will take time to learn, and the prediction will again take time. We will find out the optimal K neighbors. We will also preprocess the data to scale it.

```
knn_model1<- train(Class~., data = trainDF, method = "knn",
                   trControl = train.control,  preProcess = c("center","scale"),
                   tuneLength=10)
plot(knn_model1)
print(knn_model1)
knnPredict1 <- predict(knn_model1,newdata = subset(testDF, select=-c(Class,Date)) )
cm<-predictClassModel(svm_model_linear1,testDF,"raw")
cm$byClass
```

2.4.2.6 SVM Models

We will then try SVM models with three different kernels, linear, poly nominal, and radial. For all of them, we will train 20 models with the cost varying from zero to twenty. All of the models will use the normalized data by scaling.

For poly nominal kernel, we will fix the scale and degree to one and, three respectively. For the radial kernel, we will change the sigma from zero to one.

```
svm_model_linear1 <- train(Class~., data = trainDF, method = "svmLinear",
                           trControl = train.control,  preProcess = c("center","scale"),
                           tuneGrid = expand.grid(C = seq(0, 2, length = 20)))
plot(svm_model_ploy1)
print(svm_model_ploy1)
summary(svm_model_linear1)

svm_model_ploy1 <- train(Class~., data = trainDF, method = "svmPoly",
                         trControl = train.control,  preProcess = c("center","scale"),
                         tuneGrid = expand.grid(C = seq(0, 2, length = 20),scale=1,degree=3))
plot(svm_model_ploy1)
print(svm_model_ploy1)
predictClassModel(svm_model_ploy1,testDF,"raw")


svm_model_radial1 <- train(Class~., data = trainDF, method = "svmRadial",
                           trControl = train.control,  preProcess = c("center","scale"),
                           tuneGrid = expand.grid(C = seq(0, 2, length = 10),sigma=seq(0, 1, length = 10)))
plot(svm_model_radial1)
print(svm_model_radial1)
predictClassModel(svm_model_radial1,testDF,"raw")
```

2.4.2.7 Random Forest Models

Finally, we will train a random forest model. We will first try with the standard mtry we calculate using the square root function, and we will train a set of rf models by changing the mtry.

The mtry regulated the number of predictor variables used for split, so we will try the one we arrived at manually and try out different values to see the behavior.

```
rf_model1<- train(Class~., data = trainDF, method = "rf",
                  trControl = train.control,  preProcess = c("center","scale"),
                  tuneLength=10)

plot(rf_model1)
print(rf_model1)
cm<-predictClassModel(rf_model1,testDF,"raw")
cm$byClass
```

2.4.3 Technical Indicator Models

2.4.3.1 Association Rules

For association rules, we will discretize the continuous variables, all the technical indicators so that we can group them together

```
edaDF<-subset(tempDF, select=-(Date))
edaDF$RSI <- discretizeDF.supervised(Class ~ ., edaDF[, c('Class', 'RSI')], method='mdlp')$RSI
edaDF$SMA <- discretizeDF.supervised(Class ~ ., edaDF[, c('Class', 'SMA')], method='mdlp')$SMA
edaDF$LMA <- discretizeDF.supervised(Class ~ ., edaDF[, c('Class', 'LMA')], method='mdlp')$LMA
edaDF$ADX <- discretizeDF.supervised(Class ~ ., edaDF[, c('Class', 'ADX')], method='mdlp')$ADX
edaDF$dayOfWeek<-lubridate::wday(tempDF$Date,label = TRUE, abbr = TRUE)
edaDF$monthOfYear<-lubridate::month(tempDF$Date,label = TRUE, abbr = TRUE)

transactionData<- as(edaDF,"transactions")
inspect(transactionData)
itemFrequencyPlot(transactionData,support=0.5)

arules_model<-apriori(transactionData,parameter = list(support=0.01,confidence=0.5))

summary(arules_model)

quality(arules_model)
#inspect(arules_model)

plot(arules_model,method = "grouped")

goodrules <- arules_model[quality(arules_model)$lift > 2.5]

plot(goodrules,method="graph",engine="interactive")

#plot(goodrules, method = "grouped", interactive = TRUE)

plot(goodrules, method = "grouped")

inspect(goodrules)
```

2.4.3.2 Hierarchical Clustering Model

Then we will be using hierarchical Clustering to group our data set; we will be using only the selected features we arrived at k-means. To compute the hierarchical cluster, we will first calculate three distance measures Euclidean, Cosine, and Manhattan matric. We will use hcluster to group them; we will do it over both normalized and regular term matrix.

```
distMatrix_E <- dist(hclustDF2, method="euclidean")
distMatrix_C <- dist(hclustDF2, method="cosine")
distMatrix_M <- dist(hclustDF2, method="manhattan")
distMatrix_Mi <- dist(hclustDF2, method="minkowski", p=5)

groups_E <- hclust(distMatrix_E,method="ward.D")
plot(groups_E, cex=0.9, hang=-1)
rect.hclust(groups_E, k=2)
summary(groups_E$order)

## Cosine Similarity
groups_C <- hclust(distMatrix_C,method="ward.D")
plot(groups_C, cex=0.9, hang=-1)
rect.hclust(groups_C, k=2)

##  Manhattan
groups_M_n <- hclust(distMatrix_M,method="ward.D")
plot(groups_M_n, cex=0.9, hang=-1)
rect.hclust(groups_M_n, k=2)

##  Minkowski
groups_Mi <- hclust(distMatrix_Mi,method="ward.D")
plot(groups_Mi_n, cex=0.9, hang=-1)
rect.hclust(groups_Mi_n, k=2)
```

2.4.3.3 K-Means

```
fviz_nbclust(select(df2,-c(Date,Class)),FUN=hcut, method="silhouette")


kmeansDF2<-df2
kmeansDF2$dayOfWeek<-lubridate::wday(df2$Date)
kmeansDF2$dayOfMonth<-(lubridate::mday(df2$Date))
kmeansDF2$monthOfYear<-(lubridate::month(df2$Date))
kmeansDF2$weekOfYear<-(lubridate::week(df2$Date))
kmeansDF2<-na.omit(kmeansDF2)
kmeans2 <- kmeans(select(kmeansDF2,-c(Date,Class)),centers = 2,nstart=25,iter.max = 20)
summary(kmeans2)

kmeansDF2$kmeans<- as.factor(kmeans2$cluster)
ggplot(kmeansDF2, aes(x=LMA,fill=kmeans,group=kmeans,color=kmeans))+geom_bar(stat="count") + theme_few()

plot(kmeansDF2$LMA,kmeansDF2$ADX,
     xlab = "LMA",
     ylab = "ADX",
     col = kmeans2$cluster)
points(kmeans2$centers,col= 1:2 , pch=8,cex=2)
```

2.4.3.4 Decision Tree Models

We will start with building a decision tree model with a holdout test. We will begin with the default complexity parameter zero and test out until one.

Then we will prune the tree to generalize based on the least cross-validated error. And we will finish with a 5-fold cross-validation test to measure the accuracy.

```
tree_model2<-train(Class~.,
      data=trainDF_balanced1,
      method="rpart",
      trControl = train.control,
      tuneGrid = expand.grid(cp = seq(0, 1, length = 10)))

plot(tree_model2)
print(tree_model2)
#tree_model_1<- rpart (Class~., data=trainDF_balanced1, control=rpart.control(cp=.0001))
plotTree(tree_model2$finalModel)
summary(tree_model2$finalModel$terms)
tree_model2$finalModel$cptable
rpart.rules(tree_model2$finalModel)
#fancyRpartPlot(tree_model$finalModel)
tree_cf_unpruned<-predictClassModel(tree_model2$finalModel,testDF,"class")
tree_finalModel<-tree_model2$finalModel
ptree1<- prune(tree_model_1,
               cp=tree_finalModel$cptable[which.max(tree_finalModel$cptable[,"x error"]),"CP"])
```

2.4.3.5 Naïve Bayes

We will then train a Naïve Bayes model for the train data set and look at the accuracy. We will follow the same text train split we did for the decision tree model

```
nb_model2 =  train(Class~., data = trainDF_balanced2, method = "nb",
                   trControl = train.control,
                   tuneLength=10)
#summary(model_nb1)
plot(nb_model2)
cm_nb2<-predictClassModel(nb_model2,testDF2,"raw")
```

2.4.3.6 K Nearest Neighbor Model

Then we will start training a kNN model, it's a slow model, will take time to learn, and the prediction will again take time. We will find out the optimal K neighbors. We will also preprocess the data to scale it.

```
knn_model<- train(Class~., data = trainDF_balanced1, method = "knn",
                  trControl = train.control,  preProcess = c("center","scale"),
                  tuneLength=10)

plot(knn_model)
print(knn_model)
knnPredict <- predict(knn_model,newdata = subset(testDF, select=-c(Class,Date)) )
plotConfusionMatrix(knnPredict,testDF$Class)
```

2.4.3.7 SVM Models

We will then try SVM models with three different kernels, linear, poly nominal, and radial. For all of them, we will train 20 models with the cost varying from zero to twenty. All of the models will use the normalized data by scaling.

For poly nominal kernel, we will fix the scale and degree to one and three, respectively.

For the radial kernel, we will change the sigma from zero to one.

```
svm_model_linear2 <- train(Class~., data = trainDF_balanced1, method = "svmLinear",
                           trControl = train.control,  preProcess = c("center","scale"),
                           tuneGrid = expand.grid(C = seq(0, 2, length = 20)))

plot(svm_model_linear2)
print(svm_model_linear2)
cm<-predictClassModel(svm_model_linear2,testDF,"raw")
cm$byClass

svm_model_ploy <- train(Class~., data = trainDF_balanced1, method = "svmPoly",
                         trControl = train.control,  preProcess = c("center","scale"),
                         tuneGrid = expand.grid(C = seq(0, 2, length = 20),scale=1,degree=3))
plot(svm_model_ploy)
print(svm_model_ploy)
predictClassModel(svm_model_ploy,testDF,"raw")


svm_model_radial <- train(Class~., data = trainDF_balanced1, method = "svmRadial",
                           trControl = train.control,  preProcess = c("center","scale"),
                           tuneGrid = expand.grid(C = seq(0, 2, length = 10),sigma=seq(0, 1, length = 10)))
plot(svm_model_radial)
print(svm_model_radial)
predictClassModel(svm_model_radial,"raw")
```

2.4.3.8 Random Forest Models

Finally, we will train a random forest model. We will first try with the standard mtry we calculate using the square root function, and we will train a set of rf models by changing the mtry.

The mtry regulated the number of predictor variables used for split, so we will try the one we arrived at manually and try out different values to see the behavior.

```
rf_model2<- train(Class~., data = trainDF_balanced1, method = "rf",
                  trControl = train.control,  preProcess = c("center","scale"),
                  tuneLength=10)

plot(rf_model2)
print(rf_model2)
cm<-predictClassModel(rf_model2,testDF,"raw")
cm$byClass
```

3 Results

3.1.1 Association Rule Mining

Figure 9 shows the high lift items, lift greater than 2.5, of the lagged returns model. It offers exciting trends where we have good support and lift for February and a 3-day lagged return.

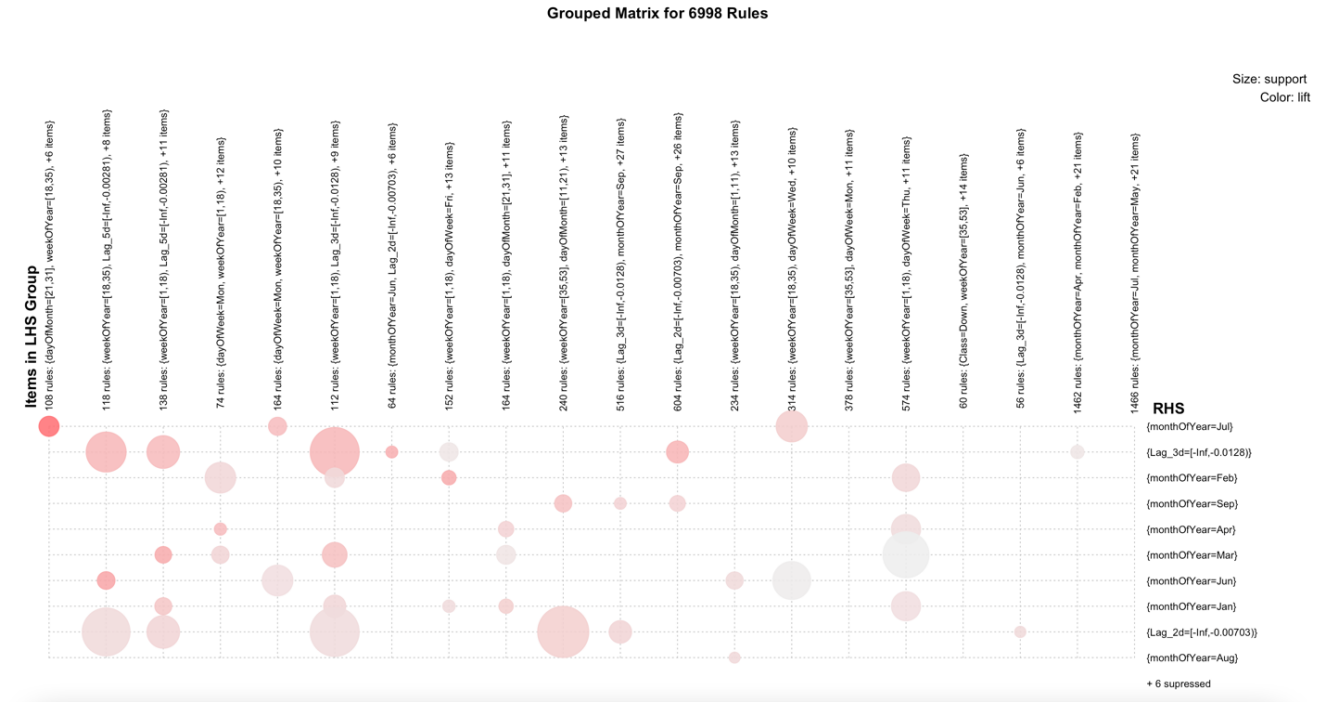


Figure 9

Figure 10 show the high lift items, lift greater than 2.5, of the technical indicator model. Week 1 and 18 have a high correlation with the lower bound RSI index.

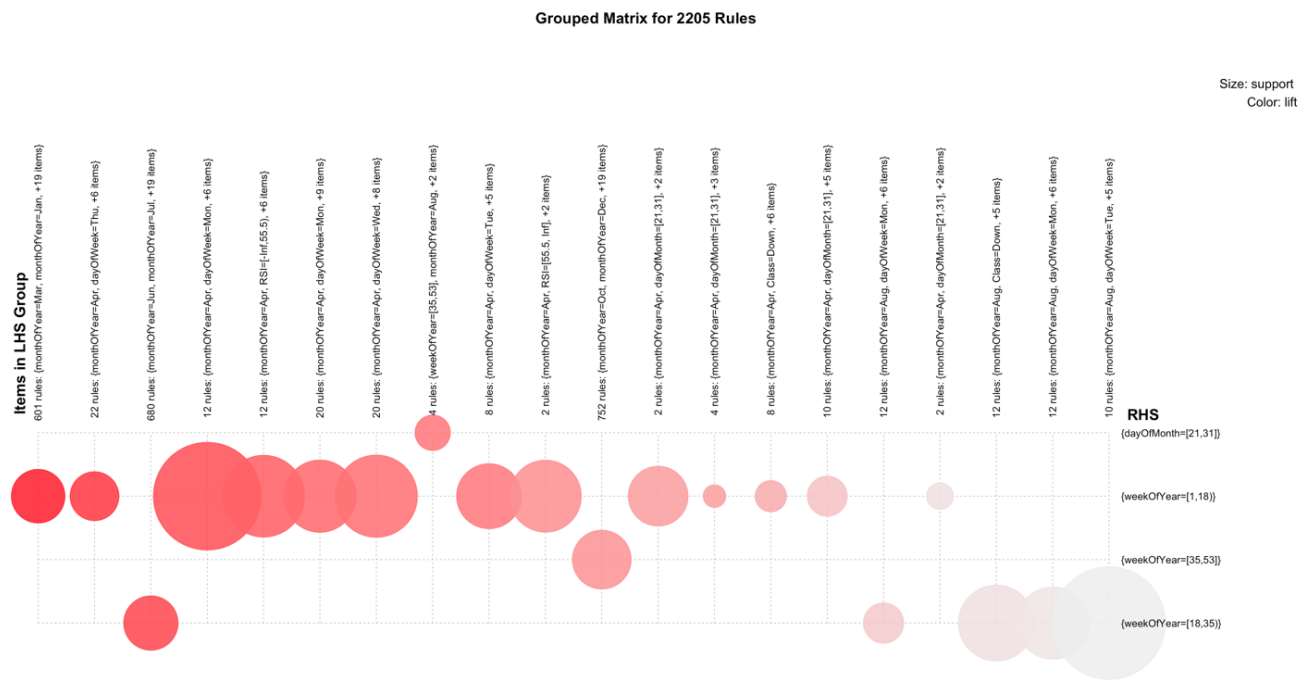


FIGURE 10

3.1.1.1 K-MEANS

The first iteration of cluster estimation recommended 2 clusters for both the approach. And the resulting k means cluster with two groups ended up classifying the data to an equal split without any definitive center. Figure 11 shows the silhouette coefficient for approaches 1 and 2.

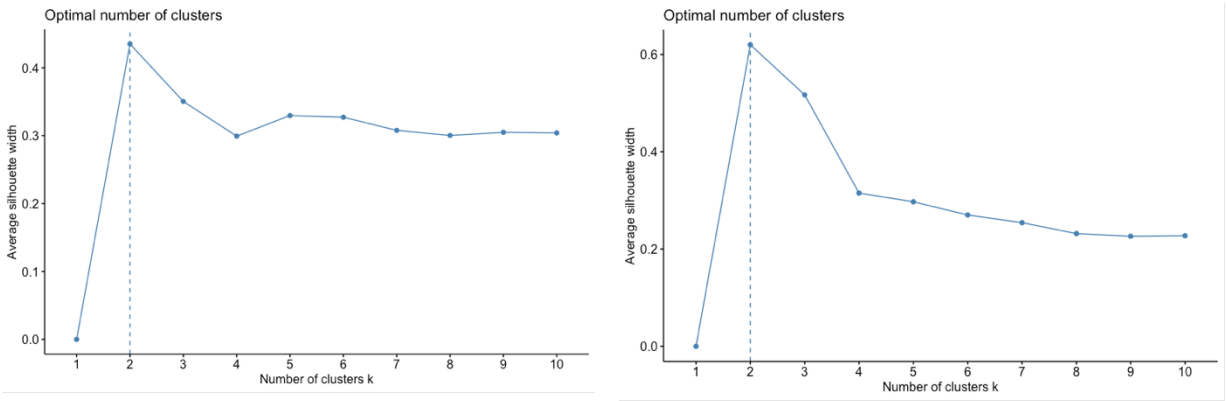


FIGURE 11

Figure 12 shows the k means plot for two variables in the second technical indicators approach. It cannot split the data set into two, so just split it into two equal parts.

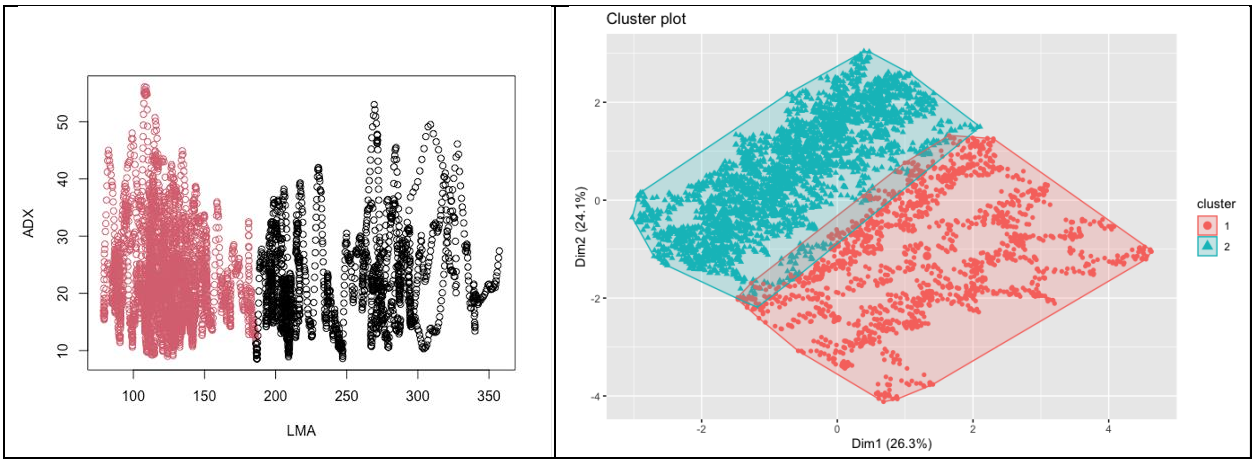
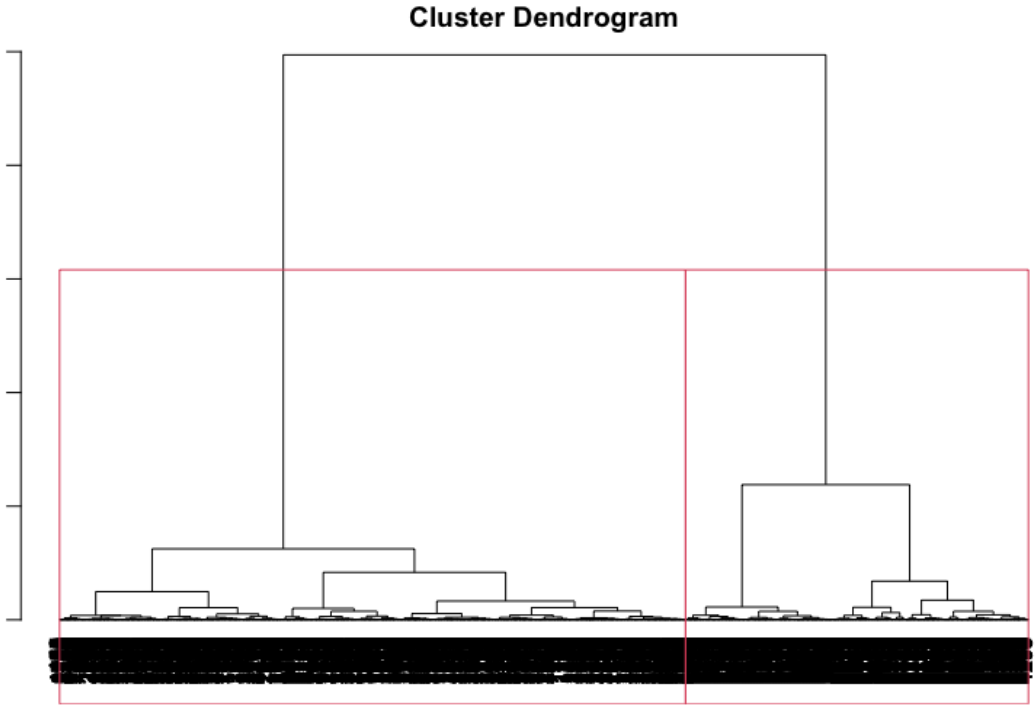


FIGURE 12

3.1.1.2 HIERARCHICAL CLUSTERING

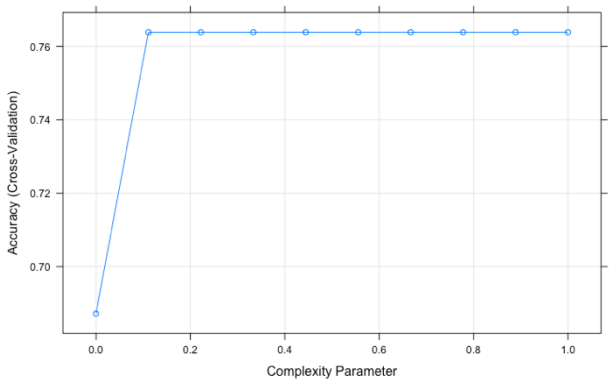
The hierarchical Clustering we tried didn't yield any meaningful result with three different distance measures.

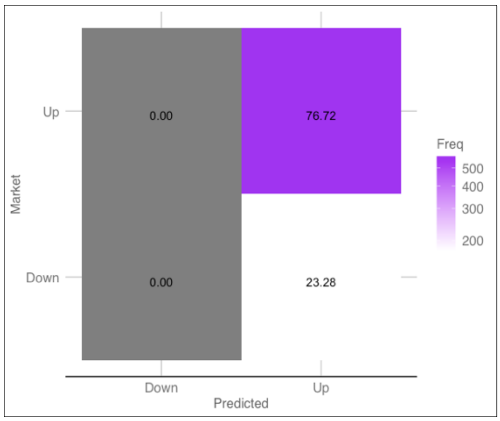


3.1.1.3 DECISION TREE MODELS

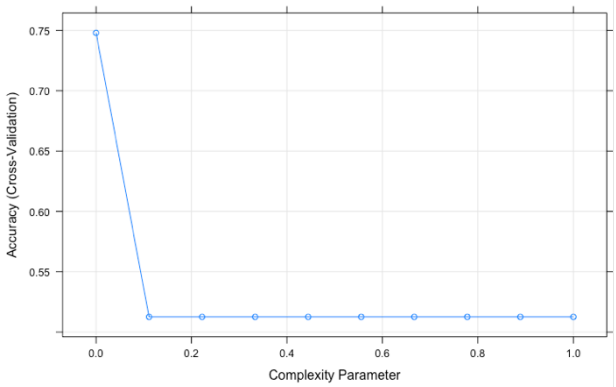
The decision tree model for approach two with technical indicators gave us a better result, where the second model at least predicted some downward movements.

Lagged Returns





Technical Indicators



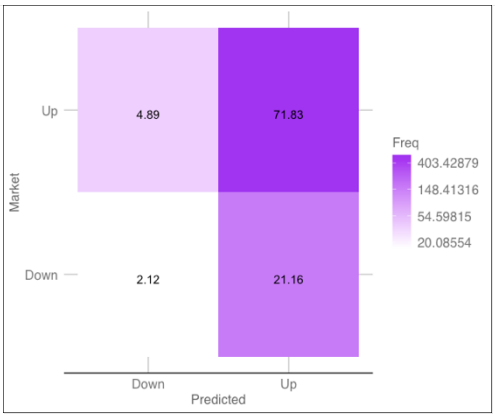


FIGURE 13

Figure 14 shows the actual tree, and it is so deep. And from the layout, we can tell it just learned to classify the 'Up' rather than 'Down.'

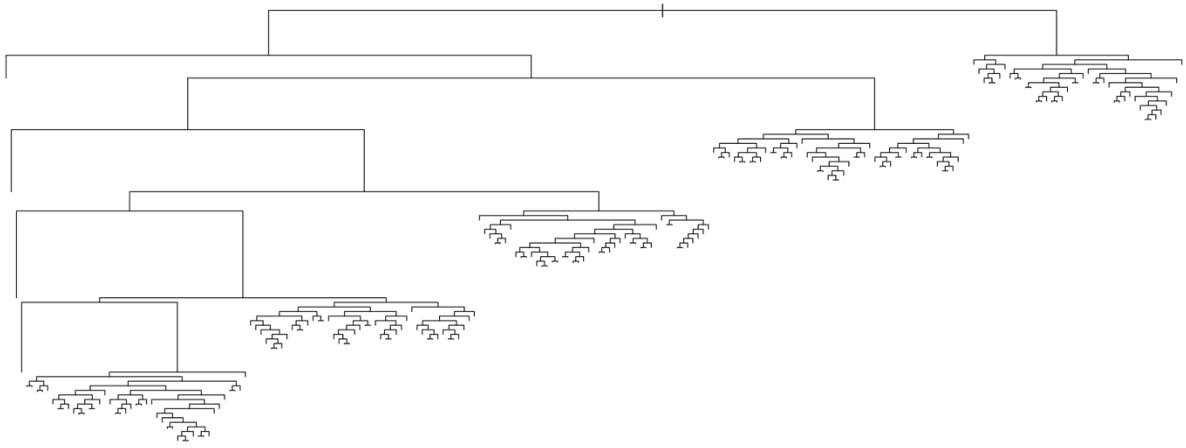
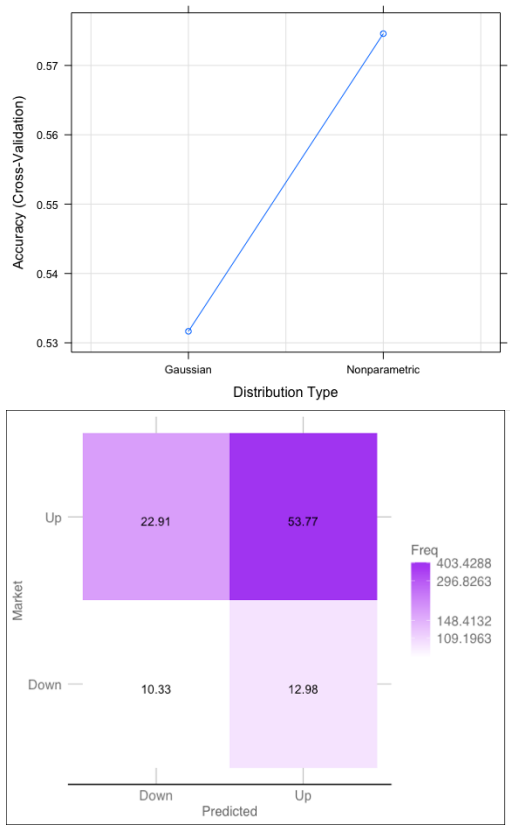


FIGURE 14

3.1.1.4 Naïve Bayes Models

Naïve Bayes models gave us the lowest accuracy of all the tested models but performed the highest in the backtesting. The parametric version is better than the gaussian models. Figure 15 shows the confusion matrix and the Naïve Bayes model's corresponding accuracy with

Lagged Returns



Technical Indicators

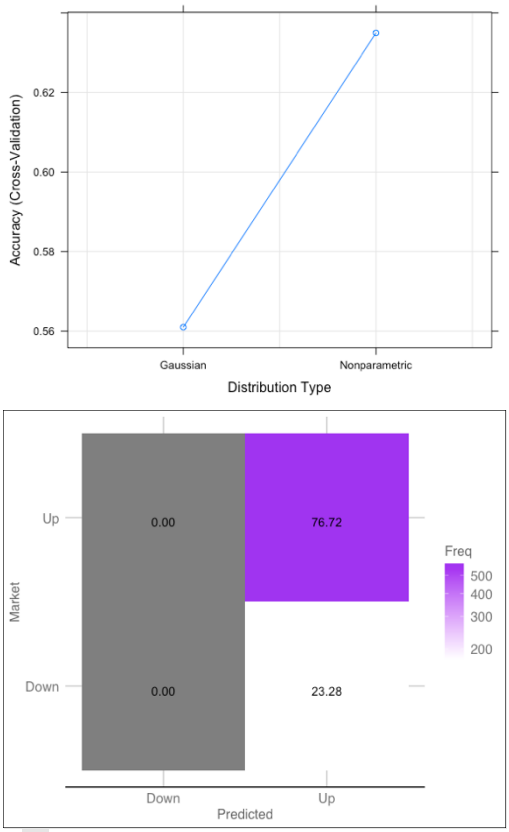
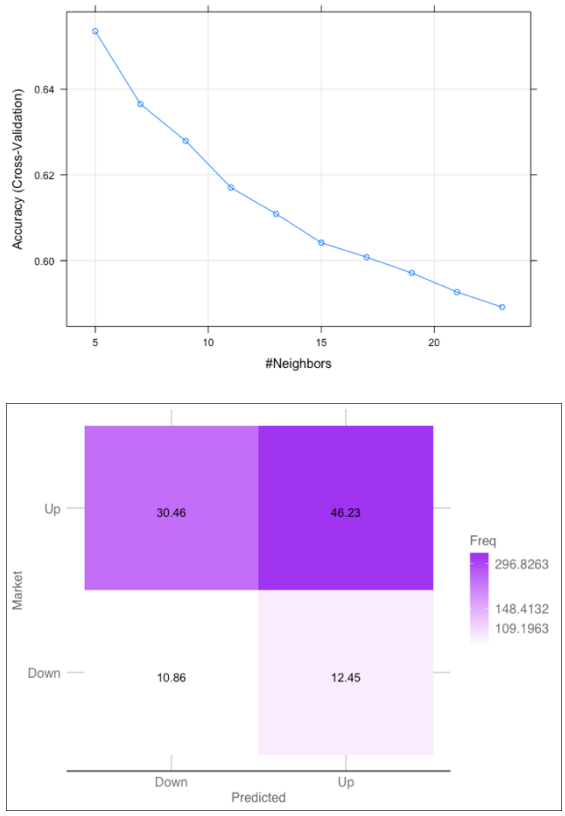


FIGURE 15

3.1.2 KNN Models

The kNN models lost some accuracy when we increased the number of neighbors considered in the count. The highest accuracy we got is 68.7%, with the k value of 5, in approach two. Figure 16 shows the k values, accuracy, and confusion matrix for the best fit model. The kNN model still has a problem classifying the downtrends. The Technical Indicator model performed better than the lagged returns.

Lagged Returns



Technical Indicators

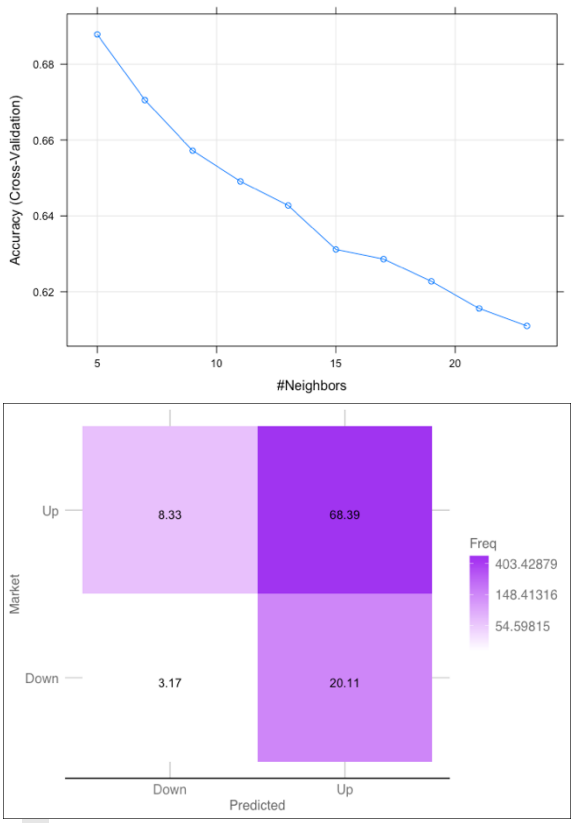


FIGURE 16

3.1.3 SUPPORT VECTOR MACHINES MODELS

For all the SVM models, we normalized our data set using the preprocessing step.

3.1.3.1 LINEAR KERNEL

The linear kernel got the highest accuracy of 56.4% at cost 2. Linear kernel gave us the most generalized model out of all the models tested; the cross-validation accuracy matched the lagged indicators model's backtesting accuracy. Figure 17 shows the confusion matrix, the cost vs. accuracy.

Lagged Returns

Technical Indicators

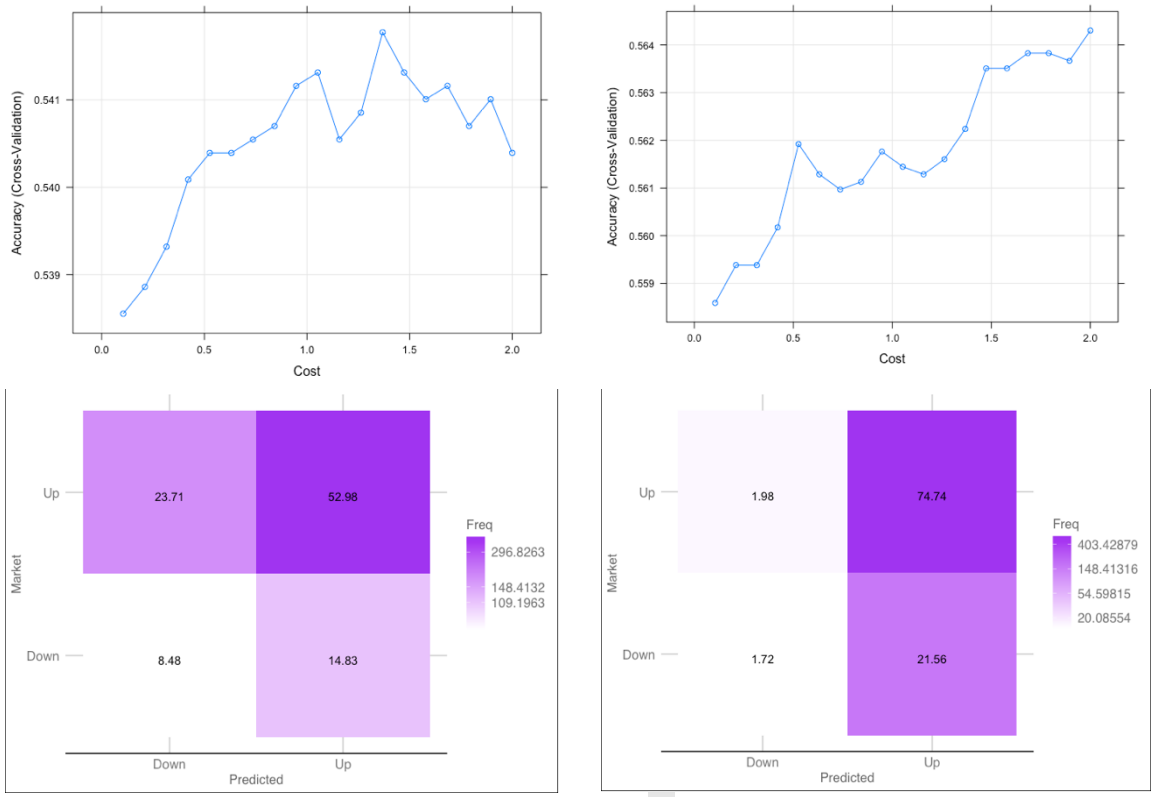
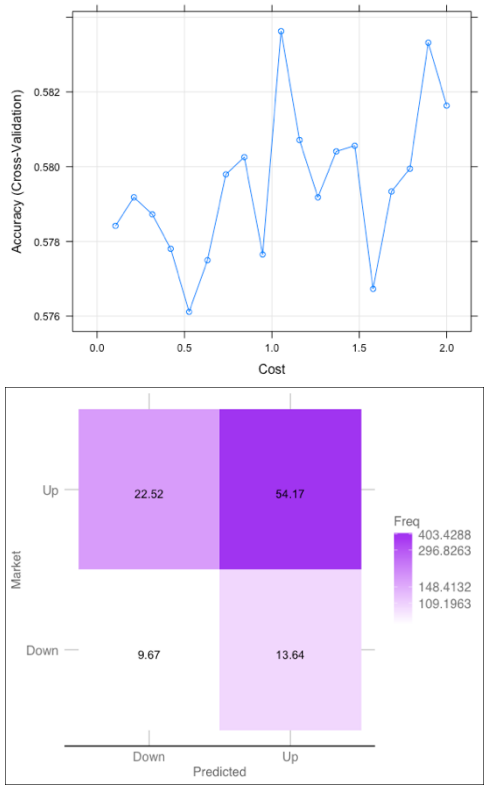


FIGURE 17

3.1.3.2 POLY NOMINAL KERNEL

The Poly Nominal achieved the highest accuracy of 61%. But did as equal as the linear kernel in backtesting. So it overfits the data. Figure 18 shows the confusion matrix, the cost vs. accuracy.

Lagged Returns



Technical Indicators

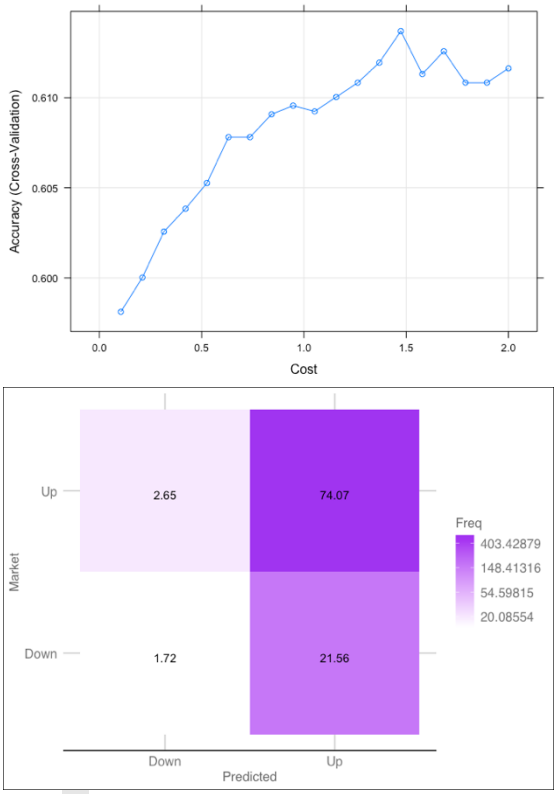


FIGURE 18

3.1.3.3 RADIAL KERNEL

Radian kernel achieved the highest accuracy of 75% at the cost of 1.77 and sigma of 1. But ended up at 50% balanced accuracy in backtesting. It did not learn the downtrend at all.

Figure 19 shows the cost vs. accuracy result and the confusion matrix.

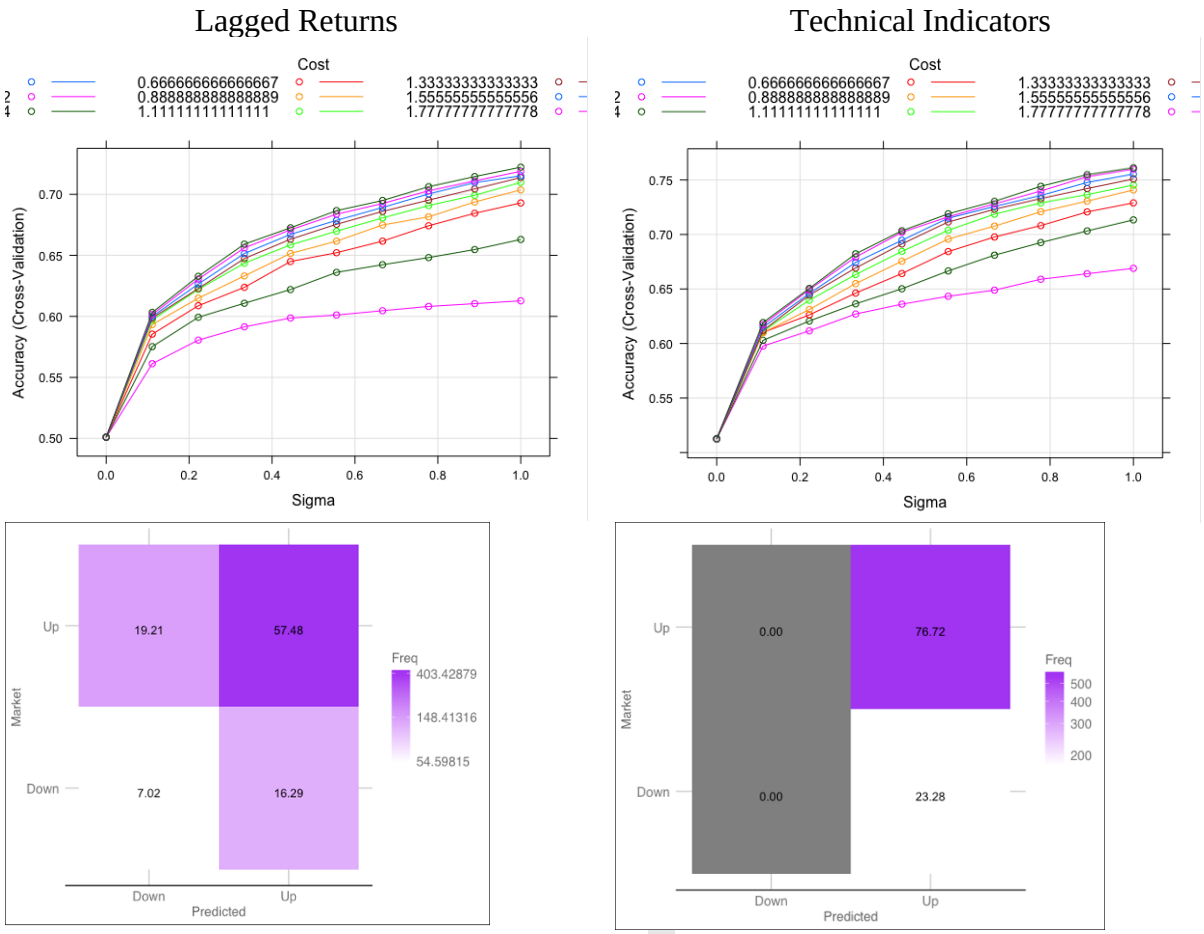
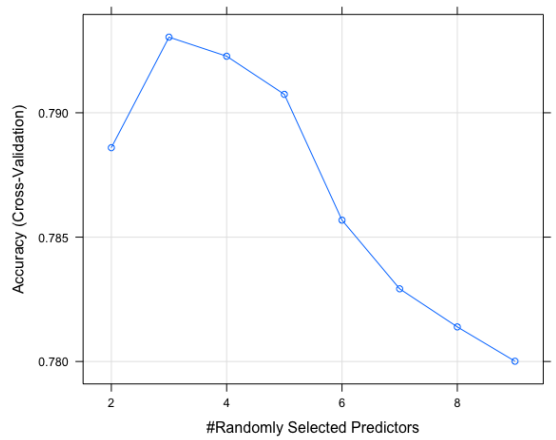


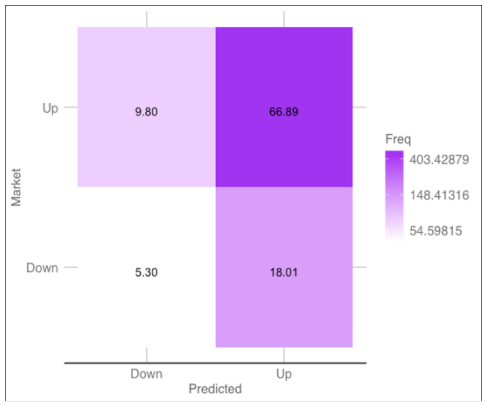
FIGURE 19

3.1.4 RANDOM FOREST MODELS

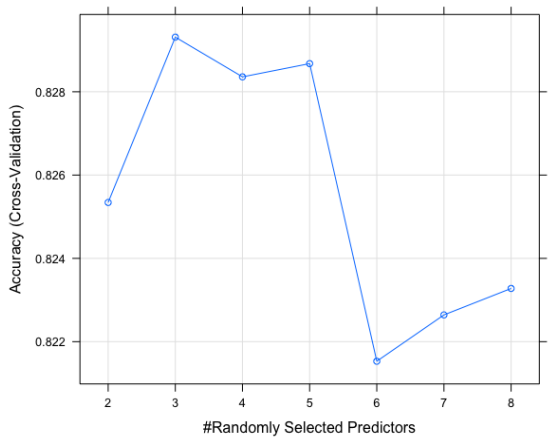
The Random Forest model sprinted and scored quickly than our SVM models. It produces the highest accuracy at mtry 3. Figure 20 shows the confusion matrix, mtry vs. accuracy results. The minimum accuracy it started with 78% is higher than our decision tree's best accuracy of 68%. So we should just start with a random forest for any real-life problems. Also, it shows the strength of the ensemble model like the random forest.

Lagged Returns





Technical Indicators



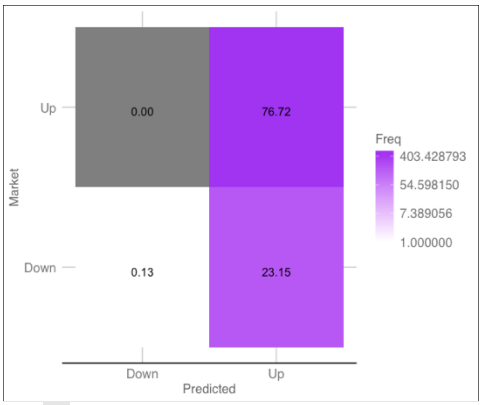


FIGURE 20

3.1.5 MODEL COMPARISON

The decision tree model is more comfortable to develop, and it took less time.

The Naïve Bayes model performed best for our lagged indicators in backtesting.

The KNN Model is the longest to train on and achieved the highest balanced accuracy.

The linear SVM models achieved the highest balanced accuracy out of all the SVM kernels.

The random forest model took as much time as SVM to train, but the prediction speed is par with a decision tree. If the real-time prediction is one of our goals, then the random forest will be our top priority over the SVM with Poly Nominal kernel.

Figure 21 summarizes all the models' training accuracy, back-testing results. Although the Random forest model performed better in cross-validation, it failed spectacularly in unseen real-world data, check the balanced accuracy. The simplest of the Naïve Bayes model performed better in the real world.

Model	K-fold Accuracy	Accuracy	Balanced Accuracy	Precision	Recall	F1

DTREE1	69.94%	63.84%	53.49%	27.65%	34.09%	30.53%
DTREE2	76.20%	76.72%	50.00%	0.00%	0.00%	0.00%
KNN1	65.35%	57.09%	53.43%	26.28%	46.59%	33.61%
KNN2	68.78%	71.56%	51.39%	27.59%	13.64%	18.25%
NB1	57.46%	64.11%	57.22%	31.08%	44.32%	36.53%
NB2	63.50%	76.72%	50.00%	0.00%	0.00%	0.00%
RF1	79.30%	72.19%	54.97%	35.09%	22.73%	27.59%
RF2	82.93%	76.85%	50.28%	100.00%	0.57%	1.13%
SVM_LINEAR1	54.18%	61.46%	52.72%	26.34%	36.36%	30.55%
SVM_LINEAR2	56.43%	76.46%	52.40%	46.43%	7.39%	12.75%
SVM_POLY1	58.36%	63.84%	56.06%	30.04%	41.48%	34.84%
SVM_POLY2	61.37%	75.79%	51.97%	39.39%	7.39%	12.44%
SVM_RADIAL1	72.22%	64.50%	52.54%	26.77%	30.11%	28.34%
SVM_RADIAL2	76.12%	76.72%	50.00%	0.00%	0.00%	0.00%

FIGURE 21

Legend	*Description*
K-fold Accuracy	Highest accuracy during 5-fold cross-validation training
Accuracy	Overall accuracy during back-testing
Balanced Accuracy	Balanced accuracy during back-testing
F1	F1 during back-testing

4 CONCLUSIONS

We explored two different approaches to the historical stock market data using multiple machine learning methods. Although we observed some initial trends, the statistical models

could not back it up with evidence. The stock market analysis is complex than we estimated and might need further investigation with enhanced datasets like real-time and adverse event data

The findings align with the weak form of the efficient market hypothesis, which states that the current security price fully reflects all currently available security market data. Thus, past price and volume (market) information will have no predictive power about security prices' future direction.

In continuation of this topic, as discussed at the beginning of our project, we continue to explore how efficient the market is if other approaches can be applied to predict market movements.

We started by look into real-time data for high-frequency trading. However, we are faced with multiple challenges. For example, the communications technologies (such as low latency fiber-optic connections or microwave relay stations) are cost-prohibitive and inaccessible for our purposes. The importance of high precision tracking in terms of the algorithms we employ is heightened when dealing with fast market data. Despite the obstacles, this could potentially be a task to take on in future projects.

We also considered fundamental analysis based on public information such as earnings, dividends, and various accounting ratios and estimates. The semi-strong form of market efficiency suggests that all public information is already reflected in stock prices. As a result, investors should not be able to earn abnormal profits by trading on this information. We believe there are some truths to it, considering 70% of fund managers underperform the market based on historical evidence. However, given the continuously evolved machine learning techniques we can deploy, we may just be 30% who beat the market.