FA590. Assignment #3.

2021-11-21

I pledge on my honor that I have not given or received any unauthorized assistance on this assignment/examination. I further pledge that I have not copied any material from a book, article, the Internet or any other source except where I have expressly cited the source.

By filling out the following fields, you are signing this pledge. No assignment will get credit without being pledged.

Name:Muhammet Furkan Isik

CWID: 10472193

Date: 11/20/2021

Instructions

In this assignment, you should use R markdown to answer the questions below. Simply type your R code into embedded chunks as shown above. When you have completed the assignment, knit the document into a PDF file, and upload both the .pdf and .Rmd files to Canvas.

```
CWID = 10472193#Place here your Campus wide ID number, this will personalize
#your results, but still maintain the reproduceable nature of using seeds.
#If you ever need to reset the seed in this assignment, use this as your seed
#Papers that use -1 as this CWID variable will earn 0's so make sure you
change
#this value before you submit your work.
personal = CWID %% 10000
set.seed(personal)#You can reset the seed at any time in your code,
#but please always set it to this seed.
```

1 point for every item of every question. Total = 22

Question 1

You have to build a predictive model for targeting offers to consumers, and conduct some model performance analytics on the result.

class: A factor with levels CH and MM indicating whether the customer purchased Citrus Hill or Minute Maid Orange Juice WeekofPurchase: Week of purchase StoreID: Store ID PriceCH: Price charged for CH PriceMM: Price charged for MM DiscCH: Discount offered for CH DiscMM: Discount offered for MM SpecialCH: Indicator of special on CH SpecialMM:

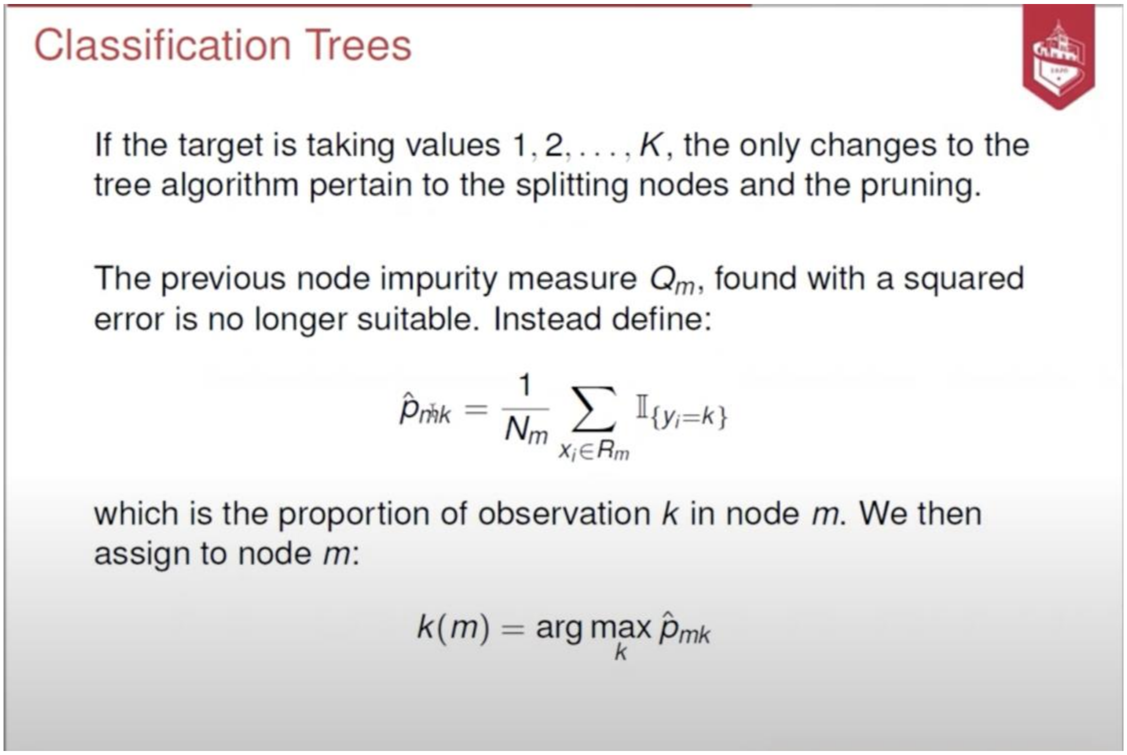
Indicator of special on MM LoyalCH: Customer brand loyalty for CH SalePriceMM: Sale price for MM SalePriceCH: Sale price for CH PriceDiff: Sale price of MM less sale price of CH Store7: A factor with levels No and Yes indicating whether the sale is at Store 7 PctDiscMM: Percentage discount for MM PctDiscCH: Percentage discount for CH ListPriceDiff: List price of MM less list price of CH STORE: Which of 5 possible stores the sale occured at

We will use historical data on past customer responses (contained in the file marketing1.csv) in order to build a classification model to forecast the customers' decision to purchase Citrus Hill or Minute Maid.

(a) You must randomly split your data set using 70% and 30% of the observations for the training and test data set respectively.

```
marketing=  read.csv(file="/Users/metuhead/Desktop/FA590/HW3/marketing1.csv")

# Randomly selecting 70% data for training, 30% for testing
train= sample(x= (1: nrow(marketing)) ,   (0.7* nrow(marketing))  )
head(marketing)

##   class WeekofPurchase StoreID PriceCH PriceMM DiscCH DiscMM SpecialCH
## 1    CH            237       1    1.75    1.99   0.00    0.0         0
## 2    CH            239       1    1.75    1.99   0.00    0.3         0
## 3    CH            245       1    1.86    2.09   0.17    0.0         0
## 4    MM            227       1    1.69    1.69   0.00    0.0         0
## 5    CH            228       7    1.69    1.69   0.00    0.0         0
## 6    CH            230       7    1.69    1.99   0.00    0.0         0
##   SpecialMM  LoyalCH SalePriceMM SalePriceCH PriceDiff Store7 PctDiscMM
## 1         0 0.500000        1.99        1.75      0.24     No  0.000000
## 2         1 0.600000        1.69        1.75     -0.06     No  0.150754
## 3         0 0.680000        2.09        1.69      0.40     No  0.000000
## 4         0 0.400000        1.69        1.69      0.00     No  0.000000
## 5         0 0.956535        1.69        1.69      0.00    Yes  0.000000
## 6         1 0.965228        1.99        1.69      0.30    Yes  0.000000
##   PctDiscCH ListPriceDiff STORE
## 1  0.000000          0.24     1
## 2  0.000000          0.24     1
## 3  0.091398          0.23     1
## 4  0.000000          0.00     1
## 5  0.000000          0.00     0
## 6  0.000000          0.30     0

# Creating training data set
df_train= marketing[train, ]

# Creating testing data set
df_test= marketing[-train, ]
```

(b) Fit a tree to the training data, with "class" as the response and the other variables as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?



Classification Trees

Measures of Node Impurity

MisClassification error:

$$\frac{1}{N_m} \sum_{i \in R_m} \mathbb{I}_{\{y_i \neq k(m)\}} = 1 - \hat{p}_{mk(m)}$$

Gini index:

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk})$$

Cross-entropy or deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

Node Impurity

8.1.4 Advantages and Disadvantages of Trees

Decision trees for regression and classification have a number of advantages over the more classical approaches seen in Chapters 3 and 4:

- ▲ Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- ▲ Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous chapters.
- ▲ Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- ▲ Trees can easily handle qualitative predictors without the need to create dummy variables.

Pros of Trees

- Unfortunately, trees generally do not have the same level of predictive accuracy as some of the other regression and classification approaches seen in this book.

- Additionally, trees can be very non-robust. In other words, a small change in the data can cause a large change in the final estimated tree.|

However, by aggregating many decision trees, using methods like *bagging*, *random forests*, and *boosting*, the predictive performance of trees can be substantially improved. We introduce these concepts in the next section.

Cons of Trees

- Two variables which are LoyalCH" , "PriceDiff" used in tree costruction
- Training error rate : 0.1268
- Number of terminal nodes: 10

```
set.seed(22)
library(tree)
tree.opt= tree(class~., data= df_train)
summary(tree.opt)

##
## Classification tree:
## tree(formula = class ~ ., data = df_train)
## Variables actually used in tree construction:
## [1] "LoyalCH"   "PriceDiff"
## Number of terminal nodes:  10
## Residual mean deviance:  0.6508 = 480.9 / 739
## Misclassification error rate: 0.1362 = 102 / 749
```

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

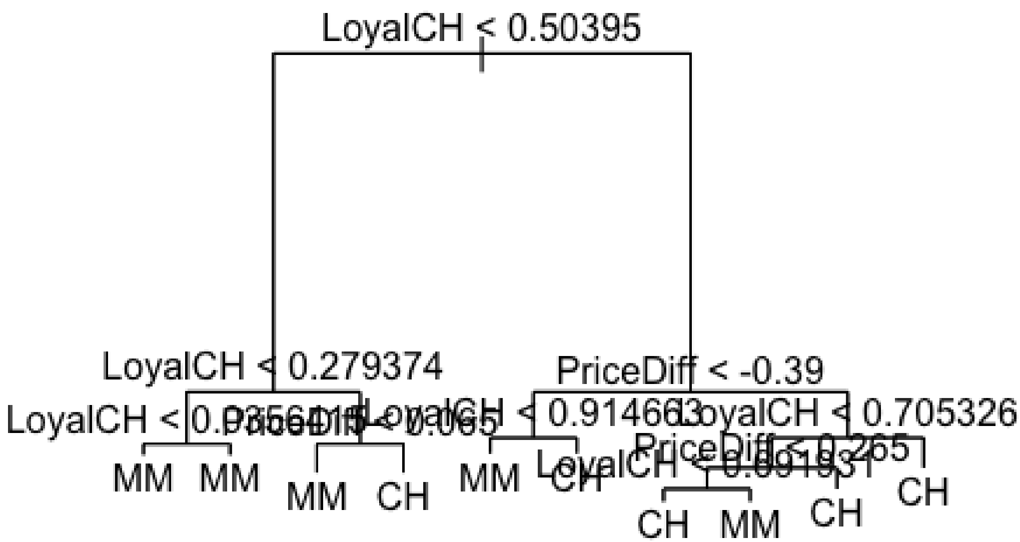
- Chosen Termina node is 6: Data classification starts with LoyalCh, if LoyalCH bigger than 0.5036, then data split again acccording to LoyalCH,and if it is between 0.5036 and 0.705699, PriceDiff comes into play and if PriceDiff less than 0.265 it gives us CH class

```r
# stars are the terminal nodes
tree.opt

## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 749 990.100 CH ( 0.62617 0.37383 )
##    2) LoyalCH < 0.50395 311 360.800 MM ( 0.26688 0.73312 )
##      4) LoyalCH < 0.279374 141  82.080 MM ( 0.08511 0.91489 )
##        8) LoyalCH < 0.0356415 46   0.000 MM ( 0.00000 1.00000 ) *
##        9) LoyalCH > 0.0356415 95  72.070 MM ( 0.12632 0.87368 ) *
##      5) LoyalCH > 0.279374 170 231.000 MM ( 0.41765 0.58235 )
##       10) PriceDiff < 0.065 69  66.780 MM ( 0.18841 0.81159 ) *
##       11) PriceDiff > 0.065 101 137.800 CH ( 0.57426 0.42574 ) *
##    3) LoyalCH > 0.50395 438 319.200 CH ( 0.88128 0.11872 )
##      6) PriceDiff < -0.39 25  32.670 MM ( 0.36000 0.64000 )
##       12) LoyalCH < 0.914663 17  12.320 MM ( 0.11765 0.88235 ) *
##       13) LoyalCH > 0.914663 8   6.028 CH ( 0.87500 0.12500 ) *
##      7) PriceDiff > -0.39 413 244.400 CH ( 0.91283 0.08717 )
##       14) LoyalCH < 0.705326 131 130.500 CH ( 0.80153 0.19847 )
##         28) PriceDiff < 0.265 68  87.020 CH ( 0.66176 0.33824 )
##           56) LoyalCH < 0.691931 63  75.380 CH ( 0.71429 0.28571 ) *
##           57) LoyalCH > 0.691931 5   0.000 MM ( 0.00000 1.00000 ) *
##         29) PriceDiff > 0.265 63  24.120 CH ( 0.95238 0.04762 ) *
##       15) LoyalCH > 0.705326 282  86.430 CH ( 0.96454 0.03546 ) *
```

(d) Create a plot of the tree, and interpret the results.

- Two variables used to construct decision tree, LoyalCH" , “PriceDiff” ,
- Separation starts with LoyalCH Value below and above 0.50395, then LoyalCh value used again for the next nodes to separate the data between 0.2793 and 0.5039, and Price differencec, then for the next nodes LoyalCH, priceDiff used

```r
par(mfrow=c(1,1))
plot(tree.opt)
text(tree.opt,pretty=30)
```



- Other way to create the plot

```
#set.seed(21)

#library(rpart)
#library(rattle)


#rpart.tree= rpart(class~., data= df_train)
#fancyRpartPlot(rpart.tree)
```

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

- Test error rate 0.2180685
- Accuracy rate is 0.7819315

```
# Predicting response using test data
tree.pred= predict(tree.opt, df_test,type="class")
#tree.pred
# Creating table for confusion matrix
table(tree.pred, df_test[,1] )
```

```
##
## tree.pred  CH  MM
##        CH 157  43
##        MM  27  94

# Accuracy rate
mean(tree.pred== df_test[,1])

## [1] 0.7819315

# Test error rate
mean(tree.pred != df_test[,1])

## [1] 0.2180685
```

(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

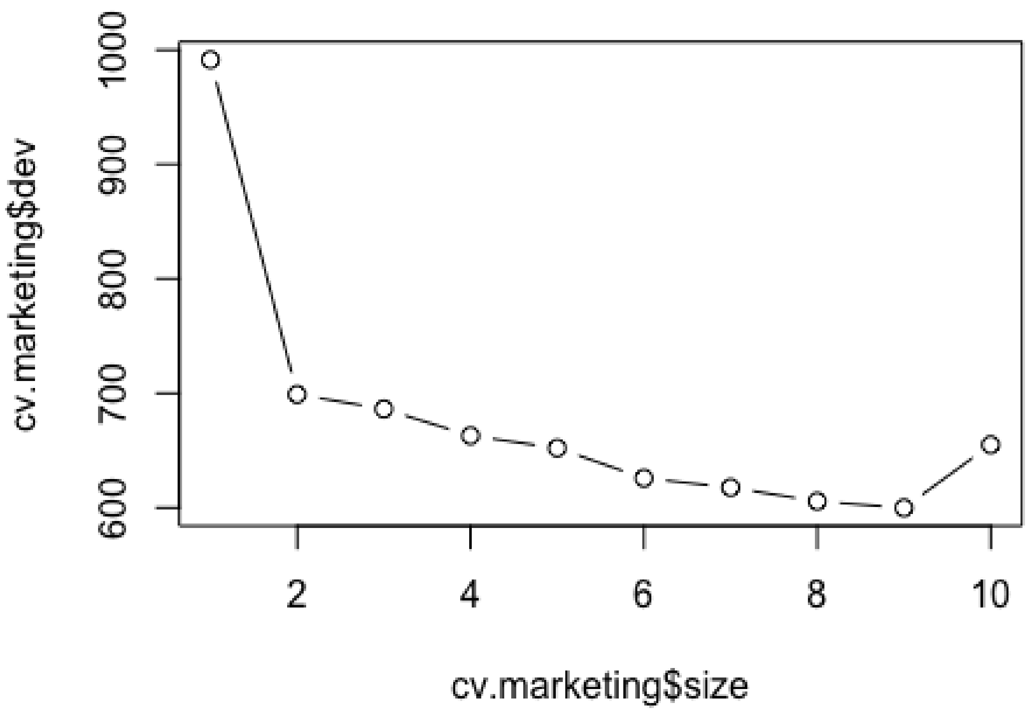
- Size 9 gives the lowest deviance value

```
# Cross Validation approach on tree method
set.seed(21)
cv.marketing=cv.tree(tree.opt)
names(cv.marketing)

## [1] "size"   "dev"    "k"      "method"

which.min ( cv.marketing$dev )

## [1] 2

cv.marketing

## $size
##  [1] 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 654.9940 599.8212 605.6181 617.6215 625.8245 652.1139 662.9501
686.3224
##  [9] 699.0486 991.3861
##
## $k
##  [1]      -Inf  10.00962  11.63872  14.32743  19.40636  26.47883  27.46491
##  [8]  42.07558  47.72392 310.09341
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

- number of terminal nodes of each tree considered (size)
- dev corresponds to the number of cross-validation errors.

```
plot(cv.marketing$size , cv.marketing$dev, type = "b")
```



(h) Which tree size corresponds to the lowest cross-validated classification error rate?

- Tree size 9 corresponding to the lowest cross- validated calssiffication error

```
cv.marketing

## $size
##  [1] 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 654.9940 599.8212 605.6181 617.6215 625.8245 652.1139 662.9501
686.3224
##  [9] 699.0486 991.3861
##
```

```
## $k
##  [1]      -Inf  10.00962  11.63872  14.32743  19.40636  26.47883  27.46491
##  [8]  42.07558  47.72392 310.09341
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to a selection of a pruned tree, then create a pruned tree with five terminal nodes.

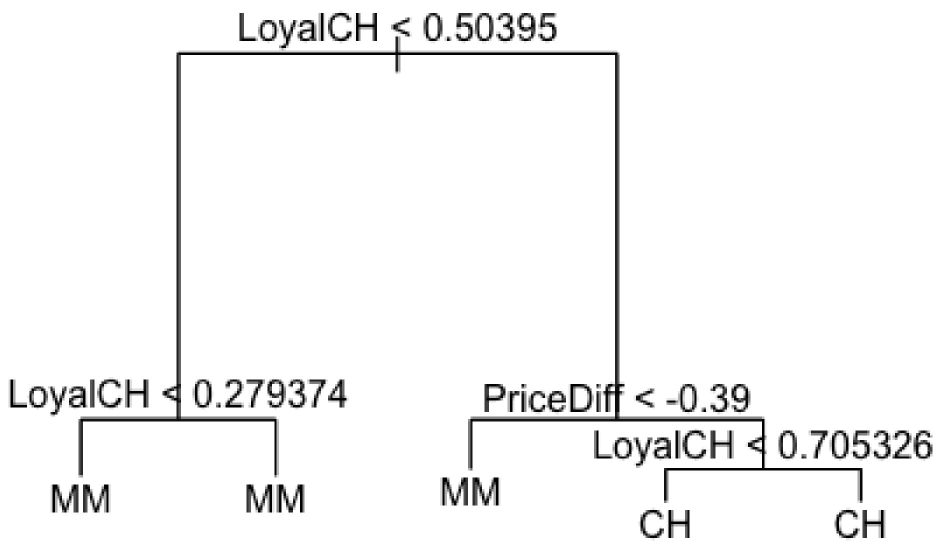
Cost complexity pruning—also known as *weakest link pruning*—gives us a way to do just this. Rather than considering every possible subtree, we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha|T| \qquad (8.4)$$

Tree Pruning

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data
- Since 9 nodes gives the similar results as 10 nodes, I prefered to use 5 nodes

```r
set.seed(21)
prune.marketing= prune.tree(tree.opt,best=5)
plot(prune.marketing)
text(prune.marketing)
```



(j) Compare the training and test error rates between the pruned and unpruned trees. Which is higher?

- Test error is larger then train error, for both methods
- For training error, UnPruned tree gives the lower MSE
- For test error, UnPruned tree gives the lowe MSE

```
# Train error
prune.train=1-mean( df_train[,1]==predict(prune.marketing, df_train,
type="class" )  )
nonprune.train=1- mean(df_train[,1]==predict(tree.opt, df_train, type="class"
))

# Test error
prune.test= 1- mean( df_test[,1]==predict(prune.marketing, df_test,
type="class" )  )
nonprune.test= 1- mean(df_test[,1]==predict(tree.opt, df_test, type="class"
))


comp.tree= data.frame(c(prune.train,nonprune.train, prune.test,
nonprune.test))
row.names(comp.tree)= c( "Pruned Train", "Unpruned Train", "Pruned Test", "
```

```
Unpruned Test")
colnames(comp.tree)= "MSE"

comp.tree
```

```
##                        MSE
## Pruned Train   0.1708945
## Unpruned Train 0.1361816
## Pruned Test    0.2398754
##  Unpruned Test 0.2180685
```

Question 2

You have to predict the daily return of the bitcoin for the next period. The file BTCreturns.csv includes the following variables:

Daily return based on adjusted daily closing prices: Core ETFs that represent complete market: VTI: Vanguard Total Stock Market ETF return VXUS: Vanguard Total International Stock ETF return BND: Vanguard Total Bond Market ETF return BNDX: Vanguard Total International Bond ETF return

Investment style: VUG Vanguard Growth ETF return VTV Vanguard Value ETF return

Sectors: Technology (growth) and energy (value) QQQ Invesco Nasdaq return XLE Energy ETF return

Cryptocurrencies: ETH Ethereum return ETH_V Ethereum trading volume BTC Bitcoin return BTC_V Bitcoin trading volume

Additional market factors from 5 factors Fama French model: RM-Rf : market return minus risk free rate (market risk premium) SMB: Small Minus Big (firm size): difference of average return on 9 small and 9 big stock portfolios HML: High Minus Low (value): difference of average return on 2 value and 2 growth portfolios RMW (Robust Minus Weak): difference of average return on 2 robust and 2 weak operating profitability portfolios CMA (Conservative Minus Aggressive): difference of average return on 2 conservative and 2 aggressive investment portfolios

a) Generate a new variable BTC1 which is the BTC return of the next day. Make sure that you sort the dataset according to “date.” After sorting, you can remove “date” from your data set.

- Downloading the data
- Sorting by date

```
btc=read.csv(file="/Users/metuhead/Desktop/FA590/HW3/BTCreturns.csv")
head(btc)
```

```
##        Date        VTI      VXUS         BND        BNDX         VUG
## 1 8/10/2015  1.2396115  1.2769629 -0.20849333 -0.17027426  1.11425700
```

```
## 2 8/11/2015 -0.9050782 -1.7196933  0.37982377  0.37795630 -0.87576974
## 3 8/12/2015  0.1020012 -0.6475137 -0.03668650  0.01884436  0.10075482
## 4 8/13/2015 -0.1391310 -0.2642638 -0.20820241 -0.05654372 -0.05495894
## 5 8/14/2015  0.3798018  0.1017209 -0.03672168 -0.13217826  0.34743103
## 6 8/17/2015  0.6359180 -0.1831708  0.18375144  0.16987763  0.75464517
##            VTV        QQQ        XLE RM.Rf   SMB   HML   RMW   CMA
ETH
## 1  1.35750455  1.1357729  3.1426945  1.31  0.19  0.69  0.17 -0.01 -
136.4290987
## 2 -0.87908242 -1.2899844  0.1877503 -0.98 -0.09  0.43  0.15  0.05
41.0335263
## 3  0.03579989  0.3443992  1.8018853  0.07 -0.16 -0.31 -0.10 -0.08
13.1093648
## 4 -0.14323289 -0.1629956 -1.5280693 -0.14 -0.43  0.08  0.14 -0.08
40.6291638
## 5  0.44096423  0.1539465 -0.2161049  0.43  0.14  0.43 -0.08  0.01
0.0109423
## 6  0.40352193  0.8290502  0.2161049  0.60  0.36 -0.85 -0.22 -0.37  -
41.7825980
##     ETH_V        BTC    BTC_V
## 1  405283 -5.5578502 20979400
## 2 1463100  2.2122689 25433900
## 3 2150620 -1.4941645 26815400
## 4 4068680 -0.8656832 27685500
## 5 4637030  0.6040512 27091200
## 6 1942830 -2.9425943 21617900

BTC_N= btc$BTC[2:nrow(btc)]
btc= btc[1:nrow(btc)-1, ]
btc$BTC_N=BTC_N

head(btc)

##        Date        VTI       VXUS         BND        BNDX         VUG
## 1 8/10/2015  1.2396115  1.2769629 -0.20849333 -0.17027426  1.11425700
## 2 8/11/2015 -0.9050782 -1.7196933  0.37982377  0.37795630 -0.87576974
## 3 8/12/2015  0.1020012 -0.6475137 -0.03668650  0.01884436  0.10075482
## 4 8/13/2015 -0.1391310 -0.2642638 -0.20820241 -0.05654372 -0.05495894
## 5 8/14/2015  0.3798018  0.1017209 -0.03672168 -0.13217826  0.34743103
## 6 8/17/2015  0.6359180 -0.1831708  0.18375144  0.16987763  0.75464517
##            VTV        QQQ        XLE RM.Rf   SMB   HML   RMW   CMA
ETH
## 1  1.35750455  1.1357729  3.1426945  1.31  0.19  0.69  0.17 -0.01 -
136.4290987
## 2 -0.87908242 -1.2899844  0.1877503 -0.98 -0.09  0.43  0.15  0.05
41.0335263
## 3  0.03579989  0.3443992  1.8018853  0.07 -0.16 -0.31 -0.10 -0.08
13.1093648
## 4 -0.14323289 -0.1629956 -1.5280693 -0.14 -0.43  0.08  0.14 -0.08
40.6291638
```

```
## 5  0.44096423  0.1539465 -0.2161049  0.43  0.14  0.43 -0.08  0.01
0.0109423
## 6  0.40352193  0.8290502  0.2161049  0.60  0.36 -0.85 -0.22 -0.37  -
41.7825980
##      ETH_V         BTC     BTC_V        BTC_N
## 1  405283 -5.5578502 20979400   2.2122689
## 2 1463100  2.2122689 25433900  -1.4941645
## 3 2150620 -1.4941645 26815400  -0.8656832
## 4 4068680 -0.8656832 27685500   0.6040512
## 5 4637030  0.6040512 27091200  -2.9425943
## 6 1942830 -2.9425943 21617900 -20.0634160
```

```
# Sorting the data by date
btc$Date= as.Date(btc$Date, format= "%m/%d/%Y")
head(btc[order(btc$Date), ])
```

```
##          Date        VTI       VXUS         BND        BNDX         VUG
## 1 2015-08-10  1.2396115  1.2769629 -0.20849333 -0.17027426  1.11425700
## 2 2015-08-11 -0.9050782 -1.7196933  0.37982377  0.37795630 -0.87576974
## 3 2015-08-12  0.1020012 -0.6475137 -0.03668650  0.01884436  0.10075482
## 4 2015-08-13 -0.1391310 -0.2642638 -0.20820241 -0.05654372 -0.05495894
## 5 2015-08-14  0.3798018  0.1017209 -0.03672168 -0.13217826  0.34743103
## 6 2015-08-17  0.6359180 -0.1831708  0.18375144  0.16987763  0.75464517
##           VTV        QQQ        XLE RM.Rf   SMB   HML   RMW   CMA
ETH
## 1  1.35750455  1.1357729  3.1426945  1.31  0.19  0.69  0.17 -0.01 -
136.4290987
## 2 -0.87908242 -1.2899844  0.1877503 -0.98 -0.09  0.43  0.15  0.05
41.0335263
## 3  0.03579989  0.3443992  1.8018853  0.07 -0.16 -0.31 -0.10 -0.08
13.1093648
## 4 -0.14323289 -0.1629956 -1.5280693 -0.14 -0.43  0.08  0.14 -0.08
40.6291638
## 5  0.44096423  0.1539465 -0.2161049  0.43  0.14  0.43 -0.08  0.01
0.0109423
## 6  0.40352193  0.8290502  0.2161049  0.60  0.36 -0.85 -0.22 -0.37  -
41.7825980
##      ETH_V         BTC     BTC_V        BTC_N
## 1  405283 -5.5578502 20979400   2.2122689
## 2 1463100  2.2122689 25433900  -1.4941645
## 3 2150620 -1.4941645 26815400  -0.8656832
## 4 4068680 -0.8656832 27685500   0.6040512
## 5 4637030  0.6040512 27091200  -2.9425943
## 6 1942830 -2.9425943 21617900 -20.0634160
```

```
# Excluding Date from the dataset
df_btc= btc[, -1]



head(df_btc)
```

```
##            VTI       VXUS         BND        BNDX         VUG         VTV
## 1  1.2396115  1.2769629 -0.20849333 -0.17027426  1.11425700  1.35750455
## 2 -0.9050782 -1.7196933  0.37982377  0.37795630 -0.87576974 -0.87908242
## 3  0.1020012 -0.6475137 -0.03668650  0.01884436  0.10075482  0.03579989
## 4 -0.1391310 -0.2642638 -0.20820241 -0.05654372 -0.05495894 -0.14323289
## 5  0.3798018  0.1017209 -0.03672168 -0.13217826  0.34743103  0.44096423
## 6  0.6359180 -0.1831708  0.18375144  0.16987763  0.75464517  0.40352193
##            QQQ        XLE RM.Rf   SMB   HML   RMW   CMA          ETH   ETH_V
## 1  1.1357729  3.1426945  1.31  0.19  0.69  0.17 -0.01 -136.4290987  405283
## 2 -1.2899844  0.1877503 -0.98 -0.09  0.43  0.15  0.05   41.0335263 1463100
## 3  0.3443992  1.8018853  0.07 -0.16 -0.31 -0.10 -0.08   13.1093648 2150620
## 4 -0.1629956 -1.5280693 -0.14 -0.43  0.08  0.14 -0.08   40.6291638 4068680
## 5  0.1539465 -0.2161049  0.43  0.14  0.43 -0.08  0.01    0.0109423 4637030
## 6  0.8290502  0.2161049  0.60  0.36 -0.85 -0.22 -0.37  -41.7825980 1942830
##            BTC    BTC_V       BTC_N
## 1 -5.5578502 20979400   2.2122689
## 2  2.2122689 25433900  -1.4941645
## 3 -1.4941645 26815400  -0.8656832
## 4 -0.8656832 27685500   0.6040512
## 5  0.6040512 27091200  -2.9425943
## 6 -2.9425943 21617900 -20.0634160
```

Split your data set into 70% and 30% training and testing datasets respectively.

- Data splitted as 70% and 30% training and testing datasets respectively

```
train= 1: (0.7 *nrow(df_btc) )
df_btc_train= df_btc[train, ]
head(df_btc_train)
```

```
##            VTI       VXUS         BND        BNDX         VUG         VTV
## 1  1.2396115  1.2769629 -0.20849333 -0.17027426  1.11425700  1.35750455
## 2 -0.9050782 -1.7196933  0.37982377  0.37795630 -0.87576974 -0.87908242
## 3  0.1020012 -0.6475137 -0.03668650  0.01884436  0.10075482  0.03579989
## 4 -0.1391310 -0.2642638 -0.20820241 -0.05654372 -0.05495894 -0.14323289
## 5  0.3798018  0.1017209 -0.03672168 -0.13217826  0.34743103  0.44096423
## 6  0.6359180 -0.1831708  0.18375144  0.16987763  0.75464517  0.40352193
##            QQQ        XLE RM.Rf   SMB   HML   RMW   CMA          ETH   ETH_V
## 1  1.1357729  3.1426945  1.31  0.19  0.69  0.17 -0.01 -136.4290987  405283
## 2 -1.2899844  0.1877503 -0.98 -0.09  0.43  0.15  0.05   41.0335263 1463100
## 3  0.3443992  1.8018853  0.07 -0.16 -0.31 -0.10 -0.08   13.1093648 2150620
## 4 -0.1629956 -1.5280693 -0.14 -0.43  0.08  0.14 -0.08   40.6291638 4068680
## 5  0.1539465 -0.2161049  0.43  0.14  0.43 -0.08  0.01    0.0109423 4637030
## 6  0.8290502  0.2161049  0.60  0.36 -0.85 -0.22 -0.37  -41.7825980 1942830
##            BTC    BTC_V       BTC_N
## 1 -5.5578502 20979400   2.2122689
## 2  2.2122689 25433900  -1.4941645
## 3 -1.4941645 26815400  -0.8656832
## 4 -0.8656832 27685500   0.6040512
## 5  0.6040512 27091200  -2.9425943
## 6 -2.9425943 21617900 -20.0634160
```

```
df_btc_test= df_btc[-train, ]
head(df_btc_test)
```

```
##             VTI       VXUS         BND       BNDX         VUG         VTV
## 1081 -0.2531919 -0.1856595 -0.16621563 -0.1548183 -0.42891654  0.09445382
## 1082  0.2215668  0.2042102  0.05939232  0.1376303  0.04582941  0.35972649
## 1083  0.9440680  0.7390987  0.09497372  0.0171880  0.98617958  0.56271501
## 1084  0.2127651  0.0000000  0.16597801  0.1374157  0.34543317  0.11045972
## 1085  0.4739220  0.2206626 -0.15412842  0.0000000  0.67044128  0.22056840
## 1086 -0.4489261 -0.8855967  0.00000000 -0.1717821 -0.38821343 -0.30552832
##              QQQ        XLE RM.Rf   SMB   HML   RMW   CMA       ETH
ETH_V
## 1081 -0.22285743  1.6290465 -0.14 -0.32  0.13  0.09 -0.10 -8.449430
8546371325
## 1082  0.06442892 -0.3337117  0.24  0.15  0.22  0.18  0.05 -7.185278
12020749863
## 1083  1.17720970  0.1002378  0.92  1.27 -0.39  0.11  0.01 -2.555629
10962753356
## 1084  0.19078799 -0.9225927  0.19 -0.09 -0.90  0.20 -0.25  1.684664
7648516297
## 1085  0.69646190  0.2524608  0.44  0.22 -0.01  0.04 -0.02  2.679506
8778095308
## 1086 -0.45726428 -1.0137037 -0.42 -0.11 -0.31 -0.47  0.00  1.487854
7503898278
##            BTC       BTC_V     BTC_N
## 1081 -4.908677 22514243371 -4.635206
## 1082 -4.635206 34242315785 -2.083394
## 1083 -2.083394 42685231262  1.005784
## 1084  1.005784 21129505542  4.248663
## 1085  4.248663 23991412764  3.002664
## 1086  3.002664 19709695456 -5.826067
```

(b) Perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter λ. Hint: use the gbm package, the gbm() function with the option distribution=”Gaussian” to apply boosting to a regression problem.

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   (a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$
   (c) Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

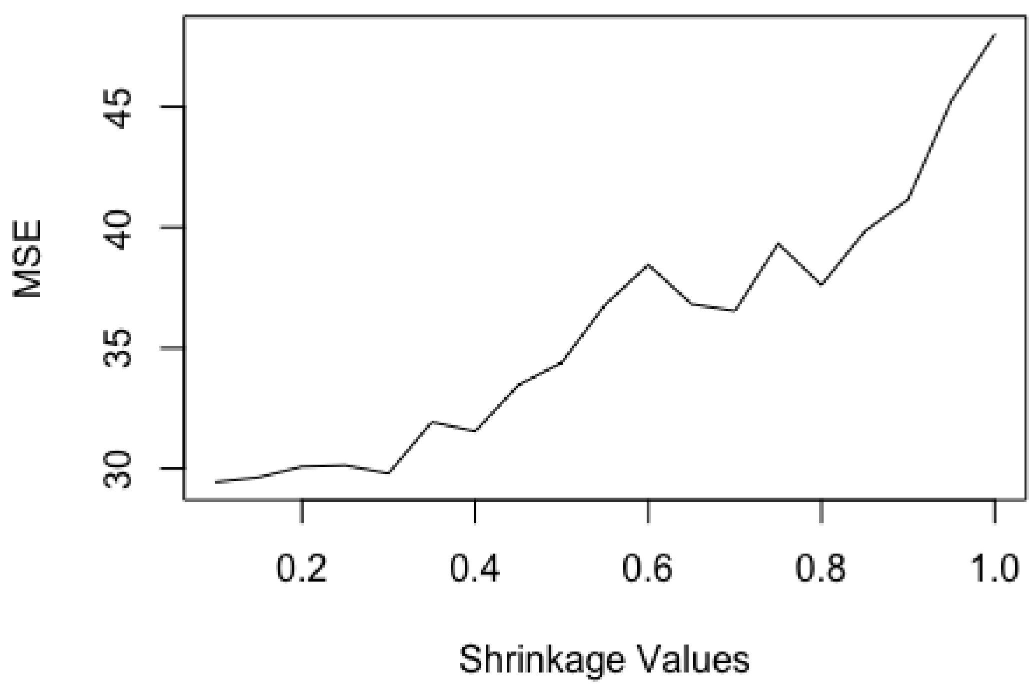
Boosting Regression Trees

```r
library(gbm)

set.seed(708)
MSE= c()

for  ( i in  seq( from= 0.1, to= 1, by=0.05 ) ) {


boost.BTC_N =   gbm(BTC_N~., data= df_btc_train, distribution = "gaussian", 
n.trees = 1000,  interaction.depth = 1,shrinkage= i )


# distribution = "gaussian" since this is a regression problem; if it were a 
binary classification problem, we would use distribution = "bernoulli".

yhat.boost = predict(boost.BTC_N, newdata = df_btc_test, n.trees = 1000 )
curr_mse= mean((yhat.boost- df_btc_test[,"BTC_N"] )^2 )
MSE= append( MSE, curr_mse )


}

MSE
```

```
## [1] 29.43412 29.63082 30.07686 30.12941 29.79273 31.93147 31.53735
33.46241
## [9] 34.39821 36.79729 38.44113 36.82450 36.53723 39.31899 37.60282
39.83419
## [17] 41.15778 45.25297 48.01591
```

(c) Produce a plot with different shrinkage values on the x-axis and the corresponding test set MSE on the y-axis.

```
a=seq( from= 0.1, to= 1, by=0.05 )
plot(x=a, y= MSE, type="l", xlab= "Shrinkage Values")
```



(d) Using the best shrinkage value, retrain your boosting model with the training dataset. What is the test set MSE for this approach?

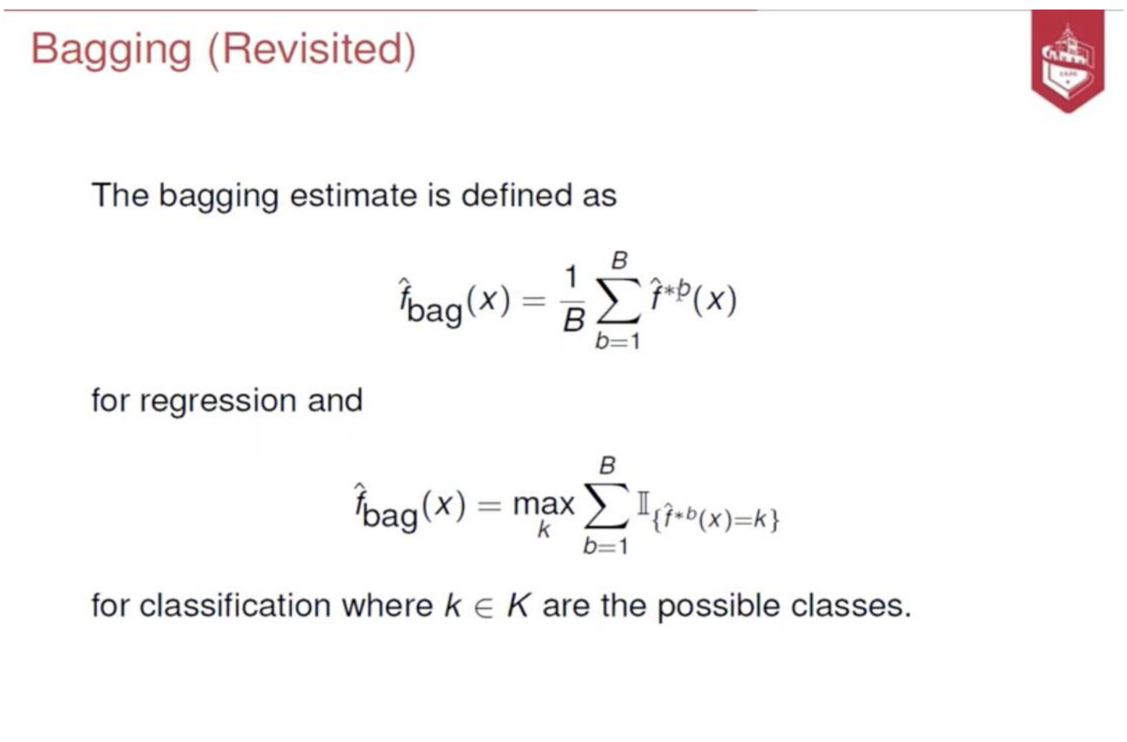
- The test MSE is 30.43285

```
set.seed(708)
which.min(MSE)

## [1] 1

best_shrikage= a[which.min(MSE)]
```

```r
boost.BTC_N =   gbm(BTC_N~., data= df_btc_train, distribution = "gaussian",
n.trees = 1000,  interaction.depth = 2,shrinkage= best_shrikage )

yhat.boost = predict(boost.BTC_N, newdata = df_btc_test, n.trees = 1000 )


MSE_test=  mean((yhat.boost-df_btc_test[,"BTC_N"])^2 )
MSE_test

## [1] 30.43285
```

(e) Apply bagging to the training set. What is the test set MSE for this approach?

- Test MSE is 57.97372
- Bootstrap aggregation called Bagging
- It is a general- purpose procudere for reducing the variance of a statistical learning method.
- Bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.



Bagging

- Obtain an overall summary of the importance of each predictor using the RSS (for bagging regression trees) or the Gini index (for bagging classification trees)
- A large value indicates an important predictor
- Variable Importance : the mean decrease in Gini index for each vari- importance able, relative to the largest.

```
library(randomForest)
set.seed(708)

bag.tree= randomForest(BTC_N~., data= df_btc_train, mtry= 17, importance=
TRUE)
bag.tree

##
## Call:
##  randomForest(formula = BTC_N ~ ., data = df_btc_train, mtry = 17,
importance = TRUE)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 17
##
##           Mean of squared residuals: 22.90828
##                     % Var explained: -7.06

summary(bag.tree)

##                 Length Class  Mode
## call               5   -none- call
## type               1   -none- character
## predicted       1080   -none- numeric
## mse              500   -none- numeric
## rsq              500   -none- numeric
## oob.times       1080   -none- numeric
## importance        34   -none- numeric
## importanceSD      17   -none- numeric
## localImportance    0   -none- NULL
## proximity          0   -none- NULL
## ntree              1   -none- numeric
## mtry               1   -none- numeric
## forest            11   -none- list
## coefs              0   -none- NULL
## y               1080   -none- numeric
## test               0   -none- NULL
## inbag              0   -none- NULL
## terms              3   terms  call

yhat.bag= predict(bag.tree, df_btc_test)
```

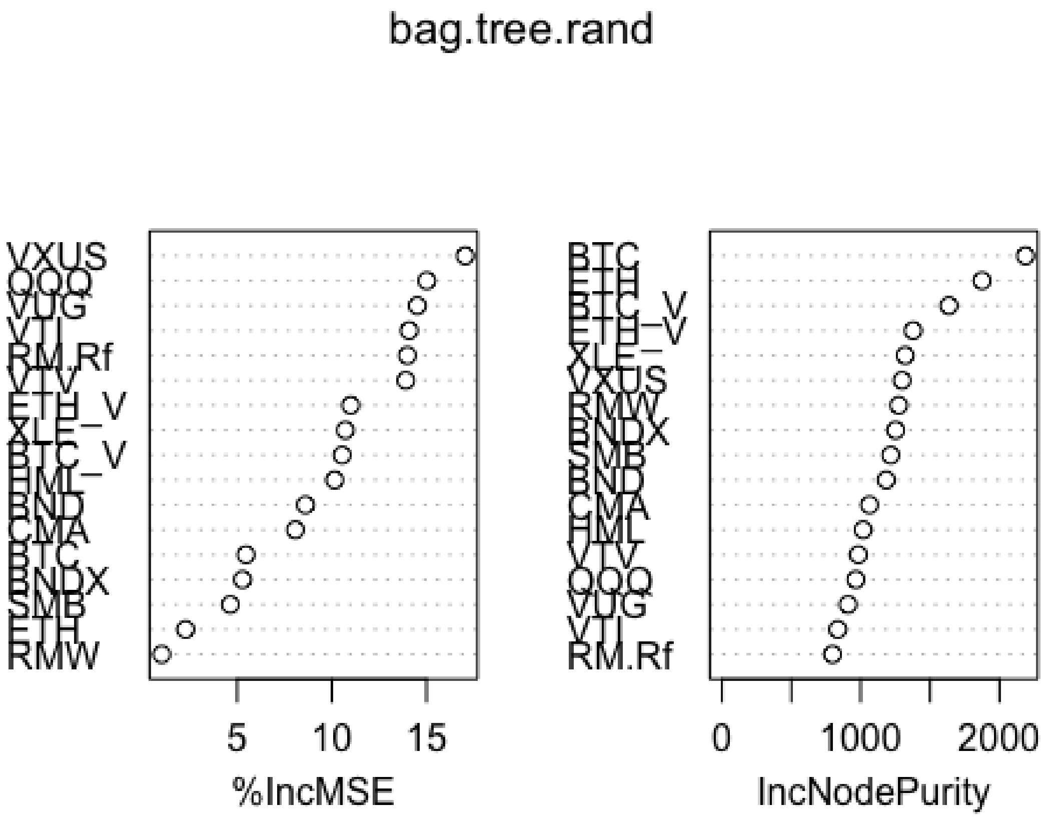
```r
MSE_bag_test= mean( (yhat.bag- df_btc_test[,"BTC_N"] )^2 )
MSE_bag_test

## [1] 57.97372

#varImpPlot(bag.tree)
```

(f) Apply random forest to the training set. What is the test set MSE for this approach? Which variables appear to be the most important predictors in the random forest model?

- Test MSE is 46.48643
- Accordiging to the plot, BTC and ETH seems to be most important variables

- Random forests provide an improvement over bagged trees by way of a random small tweak that decorrelates the trees
- Using random forest for regression
- Looking only subset of predictors

```r
set.seed(21)
library(rpart)

bag.tree.rand= randomForest(BTC_N~., data= df_btc_train, mtry= 5, importance=
TRUE)
yhat.rand= predict(bag.tree.rand, df_btc_test)

MSE_rand_test= mean( (yhat.rand- df_btc_test[,"BTC_N"] )^2 )
MSE_rand_test

## [1] 46.48643

# plot(yhat.rand, df_btc_test[,"BTC_N"])


#abline(0,0)

varImpPlot(bag.tree.rand)
```



(g) Apply support vector machine to the training set. What is the test set MSE for this approach?

- The Test "MSE 30.0219215464119"

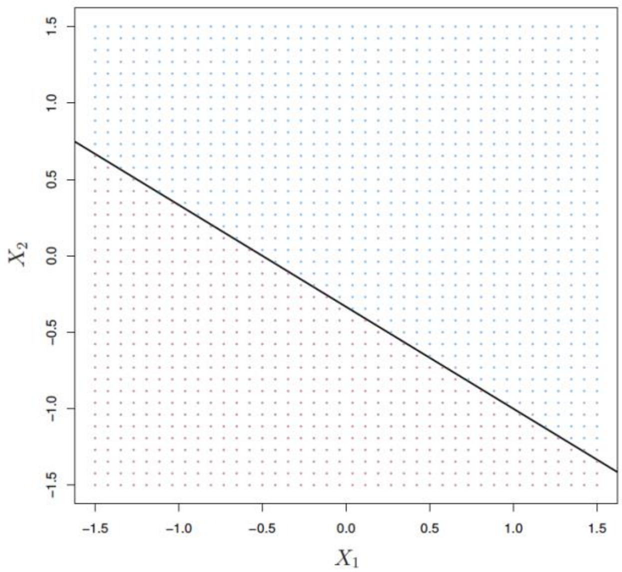


FIGURE 9.1. *The hyperplane $1 + 2X_1 + 3X_2 = 0$ is shown. The blue region is the set of points for which $1 + 2X_1 + 3X_2 > 0$, and the purple region is the set of points for which $1 + 2X_1 + 3X_2 < 0$.*

Hyper Plane in R2

9.1.4 Construction of the Maximal Margin Classifier

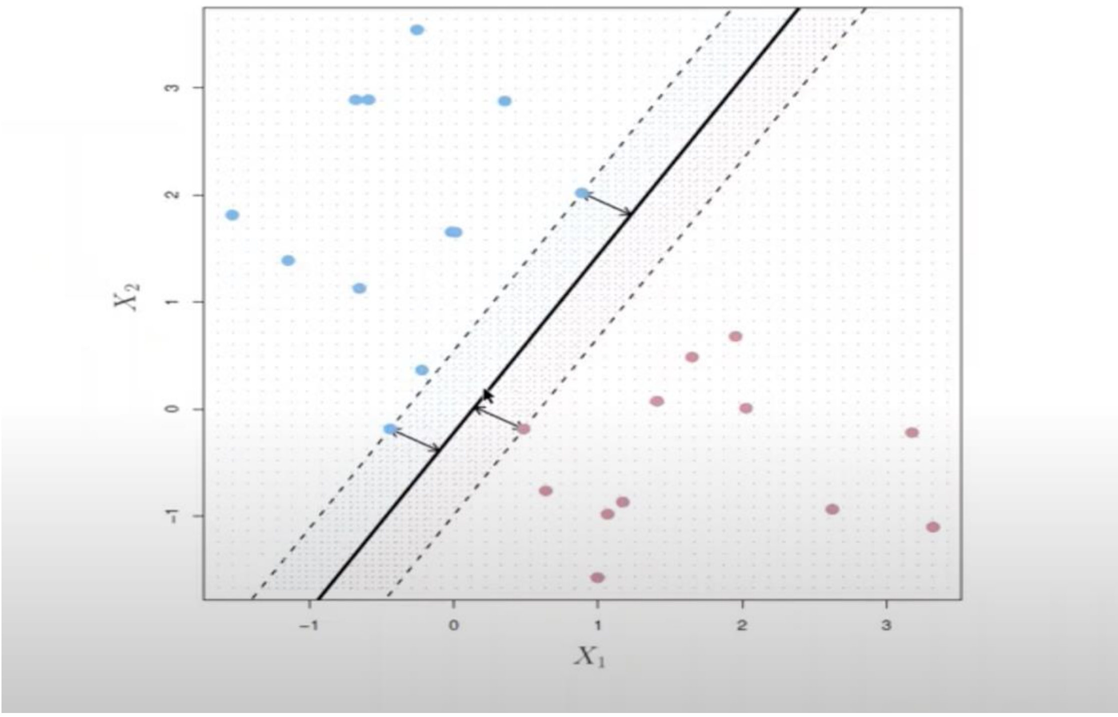
We now consider the task of constructing the maximal margin hyperplane based on a set of n training observations $x_1, \ldots, x_n \in \mathbb{R}^p$ and associated class labels $y_1, \ldots, y_n \in \{-1, 1\}$. Briefly, the maximal margin hyperplane is the solution to the optimization problem

$$\underset{\beta_0,\beta_1,\ldots,\beta_p,M}{\text{maximize}} \; M \tag{9.9}$$

$$\text{subject to} \sum_{j=1}^{p} \beta_j^2 = 1, \tag{9.10}$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip}) \geq M \;\; \forall\, i = 1, \ldots, n. \tag{9.11}$$

Maximum Margin Optimization



Support Vector Classifier

- That is, it could be worthwhile to misclassify a few training observations in order to do a better job in classifying the remaining observations. The support vector classifier, sometimes called a soft margin does exactly this.

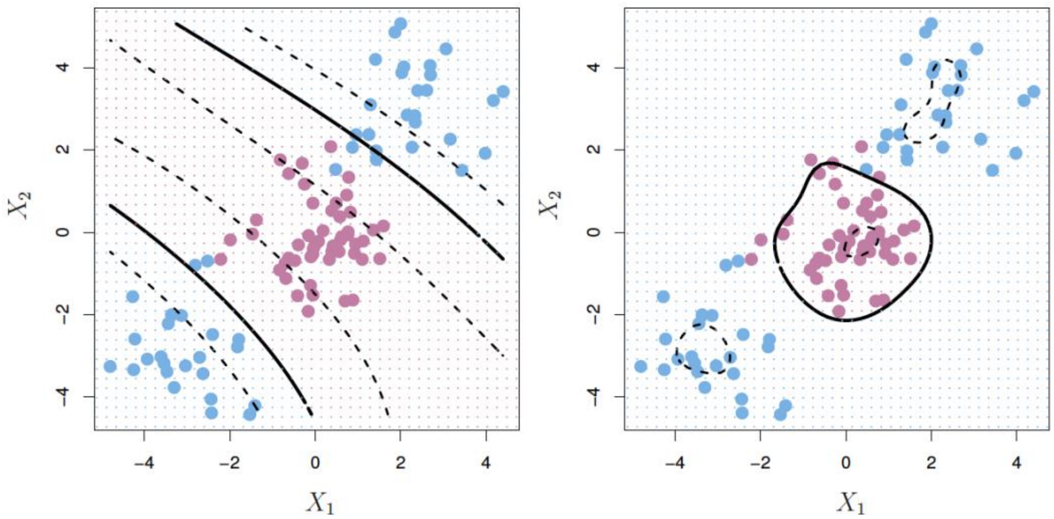


FIGURE 9.9. Left: *An SVM with a polynomial kernel of degree 3 is applied to the non-linear data from Figure 9.8, resulting in a far more appropriate decision rule.* Right: *An SVM with a radial kernel is applied. In this example, either kernel is capable of capturing the decision boundary.*

Support Vector Machines

- General mechanism for converting a linear classifier into one that produces non-linear decision boundaries. Support vector machine, which does this in an automatic way.

```
set.seed(21)

library(e1071)
sup.mod= svm(BTC_N~., df_btc_train,kernel="linear" )
summary(sup.mod)

##
## Call:
## svm(formula = BTC_N ~ ., data = df_btc_train, kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  1
##       gamma:  0.05882353
##     epsilon:  0.1
##
##
## Number of Support Vectors:  895

sup.pred= predict(sup.mod, df_btc_test )
```

```r
# MAE for Test
mean( abs(df_btc_test[,"BTC_N"] - sup.pred  ))
```

```
## [1] 3.766192
```

```r
# MSE for Test
sup.mse= mean( (df_btc_test[,"BTC_N"] - sup.pred ) ^2 )
paste( "MSE" , mean( (df_btc_test[,"BTC_N"] - sup.pred ) ^2 ) )
```

```
## [1] "MSE 30.0219215464119"
```

(h) Apply support vector machine with a nonlinear kernel to the training set. What is the test set MSE for this approach?

- The test“MSE 26.3008844918982”

```r
set.seed(21)
sup.non_mod= svm(BTC_N~., df_btc_train,kernel="radial" )
sup.non_pred= predict(sup.non_mod, df_btc_test)


# MSE for test using nonlinear kernel
sup.non_mse= mean(  (sup.non_pred -  df_btc_test[,"BTC_N"])^2  )
sup.non_mse
```

```
## [1] 26.30088
```

```r
paste ( "MSE" ,       mean(  (sup.non_pred -  df_btc_test[,"BTC_N"])^2   ) )
```

```
## [1] "MSE 26.3008844918982"
```

(i) Perform subset selection (your choice on how) in order to identify a satisfactory model that uses just a subset of the predictors (if your approach suggests using all of the predictors, then follow your results and use them all).I suggest that you use the function stepAIC.

Algorithm 6.1 *Best subset selection*

---

1. Let $\mathcal{M}_0$ denote the *null model*, which contains no predictors. This model simply predicts the sample mean for each observation.
2. For $k = 1, 2, \ldots p$:
   (a) Fit all $\binom{p}{k}$ models that contain exactly k predictors.
   (b) Pick the best among these $\binom{p}{k}$ models, and call it $\mathcal{M}_k$. Here *best* is defined as having the smallest RSS, or equivalently largest R^2.
3. Select a single best model from among $\mathcal{M}_0, \ldots, \mathcal{M}_p$ using cross-validated prediction error, C_p (AIC), BIC, or adjusted R^2.

Best Subset Selection

- According to Mallow's cp, model 2 is chosen, which has predictor as: SMB, ETH_V
- According to AIC cricterian, two variables chosen: SMB, ETH_V

```r
library(MASS)
lm.btc1 <- lm(BTC_N~., data=df_btc_train)
lm.select <- stepAIC(lm.btc1, direction = 'both', trace = FALSE)
summary(lm.select)
```

```
##
## Call:
## lm(formula = BTC_N ~ SMB + ETH_V, data = df_btc_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -23.8209  -1.6552  -0.0247   1.8277  21.9151
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.778e-01  1.738e-01   2.749  0.00608 **
## SMB         -5.297e-01  2.708e-01  -1.956  0.05072 .
```

```
## ETH_V       -7.829e-11  4.660e-11  -1.680  0.09325 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.619 on 1077 degrees of freedom
## Multiple R-squared:  0.005888,   Adjusted R-squared:  0.004042
## F-statistic: 3.189 on 2 and 1077 DF,  p-value: 0.04159
```

```
library(leaps)
sub.mod=  regsubsets(BTC_N~., data=df_btc_train)
t(summary(sub.mod)$which)
```

```
##                 1     2     3     4     5     6     7     8
## (Intercept)  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## VTI         FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE
## VXUS        FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
## BND         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## BNDX        FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## VUG         FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## VTV         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## QQQ         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## XLE         FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
## RM.Rf       FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
## SMB          TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## HML         FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE
## RMW         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## CMA         FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
## ETH         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## ETH_V       FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## BTC         FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## BTC_V       FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
which.min(summary(sub.mod)$cp)
```

```
## [1] 2
```

(j) Fit a GAM on the training data with this reduced dataset, using splines of each feature with 5 degrees of freedom.What is the test set MSE for this approach? What are the relevant nonlinear variables?

- The test MSE 27.53584

7.7.1 GAMs for Regression Problems

A natural way to extend the multiple linear regression model

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i$$

in order to allow for non-linear relationships between each feature and the response is to replace each linear component $\beta_j x_{ij}$ with a (smooth) non-linear function $f_j(x_{ij})$. We would then write the model as

$$\begin{aligned} y_i &= \beta_0 + \sum_{j=1}^{p} f_j(x_{ij}) + \epsilon_i \\ &= \beta_0 + f_1(x_{i1}) + f_2(x_{i2}) + \cdots + f_p(x_{ip}) + \epsilon_i. \end{aligned} \tag{7.15}$$

GAM for Regreesion

```
library(mgcv)
gam.btc1 <- gam(BTC_N~s(SMB,k=5)+s(ETH_V,k=5), data=df_btc_train)
gam.pred <- predict(gam.btc1,df_btc_test)
gam.mse <- mean((gam.pred - df_btc_test$BTC_N)^2)
gam.mse

## [1] 27.53584
```

(k) Build a table to compare the test set MSE of your best model for:

- Boosting
- Bagging
- Random Forests
- Support vector machine
- Support vector machine with nonlinear kernel
- GAM

```
MSE_test

## [1] 30.43285

MSE_bag_test

## [1] 57.97372

MSE_rand_test
```

```
## [1] 46.48643

sup.mse

## [1] 30.02192

sup.non_mse

## [1] 26.30088

gam.mse

## [1] 27.53584

df_com= data.frame(c(MSE_test,MSE_bag_test, MSE_rand_test,
sup.mse,sup.non_mse, gam.mse))


row.names(df_com)= c(" Boosting", " Bagging ", " Random Forrest", " Support
Vector Linear", "Support Vector Machine Non Linear","GAM")
colnames(df_com)= "Mean Squared Error"
df_com

##                                    Mean Squared Error
##  Boosting                                    30.43285
##  Bagging                                     57.97372
##  Random Forrest                              46.48643
##  Support Vector Linear                       30.02192
## Support Vector Machine Non Linear            26.30088
## GAM                                          27.53584
```

(I) Discuss and explain your results of the previous table. Why do you think that some algorithms performed better than others? What explains the result of the best algorithm?

- According to the table above, it's obvious that SVM with Nonlinear kernel, GAM and boosting gives lower results among all methods.

- Among all the methods applied on trees, boosting outperforms bagging and random forrest. Boosting, unlike in bagging, the construction of each tree depends strongly on the trees that have already been grown and algorithm updates the construction of the tree depending on the residuals.

- Comparing bagging and random forrest, they both use bootstrap t reduce the variance of the models. Random forest reduces the weights of unimportant variables in the modek, hence MSE is less than the bagging, but it does not perform as good as boosting model.

- SVM method tries to maximize the width of the gap between the two categories, and choose the hyperplane accordingly, it's an effective method for high dimensional data.

- Support vector classifier could misclassify a few training observations in order to do a better job in classifying the remaining observations. This gives more flexibility and better overall results.
- SVM with radial kernel, further reduces the MSE and more efficient
- GAM model, used subselection method to get the most important predictors, and used those two predicters with 5 degrees of freedom.