Homework 4: (Entire code at the end)

Problem: Predicting useful questions on stack overflow

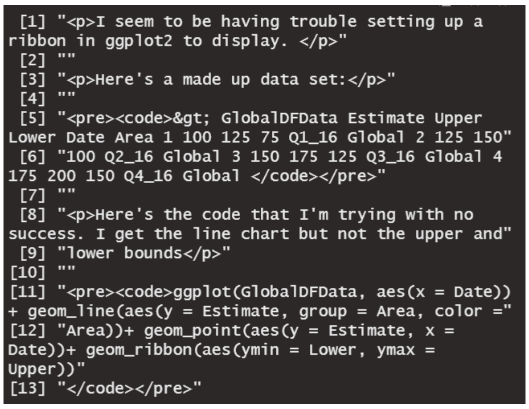
a) Data cleaning

After loading the dataset, we first create corpuses of the title and the body. We will clean the body and the title separately then merge the results. Here following is the corpus of the first question

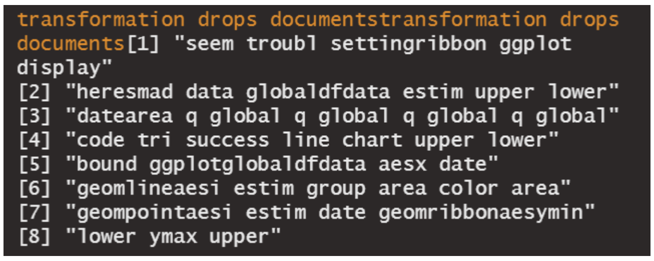
Step 1: First we will remove the lower cases and html tags. We will also remove the numbers; we assume that they are irrelevant since they are specific to the user's problem/answer.

Step 2: We remove punctuation and stop words ("R") since they are common to all our dataset, but we will keep the word "ggplot" since it can be relevant to predict the usefulness of a question

Step 3: Stemming and removing the sparse words; we choose to remove words that appears less than 1.5%. We target a ration features/observation 1:10

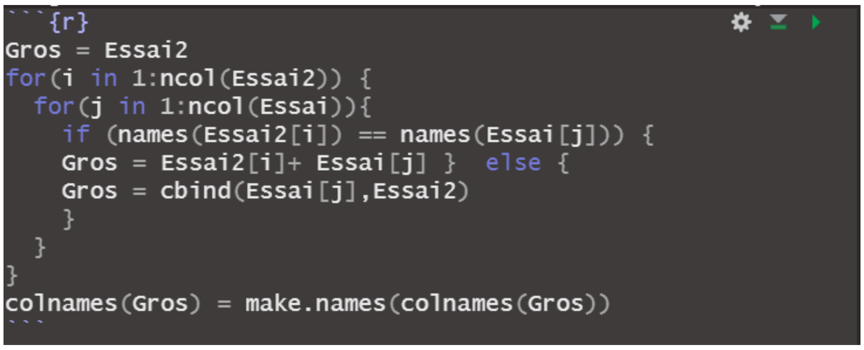


```
 [1] "<p>I seem to be having trouble setting up a
ribbon in ggplot2 to display. </p>"
 [2] ""
 [3] "<p>Here's a made up data set:</p>"
 [4] ""
 [5] "<pre><code>&gt; GlobalDFData Estimate Upper
Lower Date Area 1 100 125 75 Q1_16 Global 2 125 150"
 [6] "100 Q2_16 Global 3 150 175 125 Q3_16 Global 4
175 200 150 Q4_16 Global </code></pre>"
 [7] ""
 [8] "<p>Here's the code that I'm trying with no
success. I get the line chart but not the upper and"
 [9] "lower bounds</p>"
[10] ""
[11] "<pre><code>ggplot(GlobalDFData, aes(x = Date))
+ geom_line(aes(y = Estimate, group = Area, color ="
[12] "Area))+ geom_point(aes(y = Estimate, x =
Date))+ geom_ribbon(aes(ymin = Lower, ymax =
Upper))"
[13] "</code></pre>"
```



```
transformation drops documentstransformation drops
documents[1] "seem troubl settingribbon ggplot
display"
[2] "heresmad data globaldfdata estim upper lower"
[3] "datearea q global q global q global q global"
[4] "code tri success line chart upper lower"
[5] "bound ggplotglobaldfdata aesx date"
[6] "geomlineaesi estim group area color area"
[7] "geompointaesi estim date geomribbonaesymin"
[8] "lower ymax upper"
```

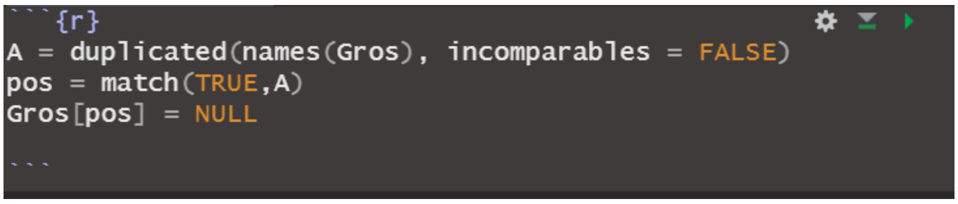
Step 4: Merging the words and their appearances for the columns and body
In this step, we consider that some words can appear both in the title document and in the body document. Thus, through two for loops, for each question, if a word appears in the two documents, then we sum the number of appearances, else we create a new column. Here is the code below.



~~~
```{r}
Gros = Essai2
for(i in 1:ncol(Essai2)) {
  for(j in 1:ncol(Essai)){
    if (names(Essai2[i]) == names(Essai[j])) {
    Gros = Essai2[i]+ Essai[j] }  else {
    Gros = cbind(Essai[j],Essai2)
    }
  }
}
colnames(Gros) = make.names(colnames(Gros))
```
~~~

Step 5: Sanity check

The goal of this step is just to check if there any duplicated label in the final dataframe. We notice that the algorithm doesn't differenciate "chart." and "chart", we will drop one.



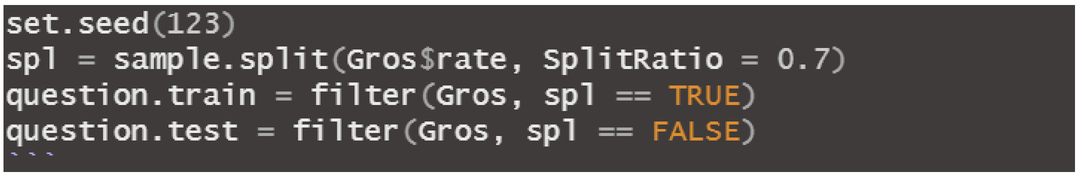
~~~
```{r}
A = duplicated(names(Gros), incomparables = FALSE)
pos = match(TRUE,A)
Gros[pos] = NULL

~~~
```

We finally have a dataset of 7468 observations with 527 independent variables. Here the dependant variable **"rate"** is binary: take the value 1 if the question is useful (Score > 1) and 0 else. We will predict the rate with the independent variable consisting of the words of each question.

b) Training the model

We will use 70% of the data for the training set and make sure that each group contains the same ratio of "useful" questions. We will use the function sample.split as following



```
set.seed(123)
spl = sample.split(Gros$rate, SplitRatio = 0.7)
question.train = filter(Gros, spl == TRUE)
question.test = filter(Gros, spl == FALSE)
```
```

### Baseline model:

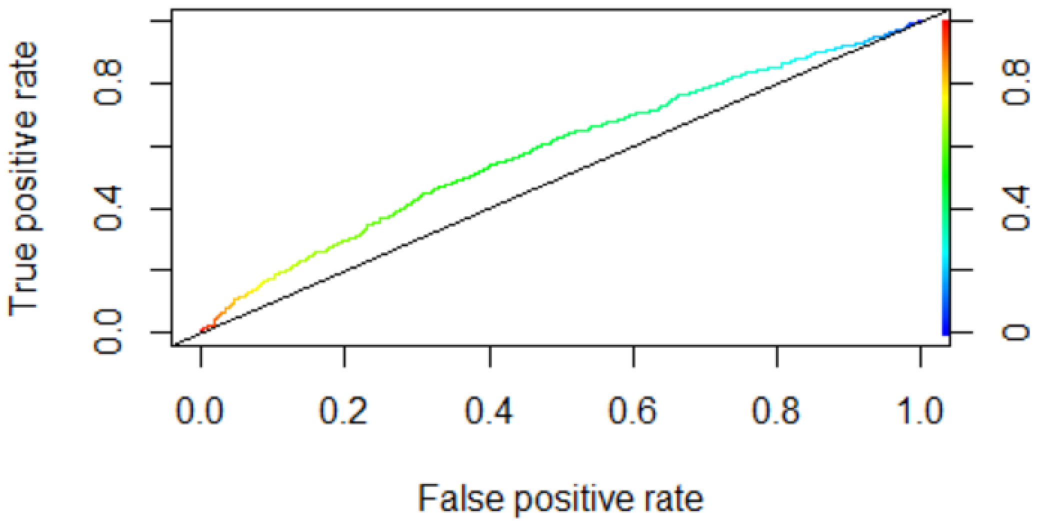
Our baseline model would be the most likely outcome, not useful (rate = 0)

| 0 | 1 |
|---|---|
| 1137 | 1103 |

$$\boldsymbol{Accuracy = \frac{1137}{1137 + 1103} = 0.507}$$

### Logistic Regression:

We will first start by a regaular logistic regression instead of a stepwise due to limited computation power. Since we don't have business data to implement a Loss function, we will start by a threshold of 0.5 then examine the ROC curve.



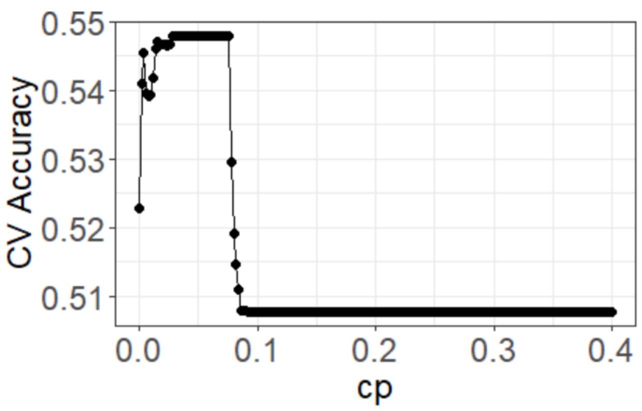
AUC = 0.58

| | p = 0.5 |
|---|---|
| **Accuracy** | **0.517** |
| **TPR** | **0.394** |
| **FPR** | **0.584** |

The ROC curve shows that we aren't really far from the baseline model, in this case it is not relevant to change the threshold.

**Cart:**

We will use crossvalidation on the cp value here to find the best model for our model, we will aim to ameliorate our accuracy. Here we have a sequence of $cp = \{0, 0.4, 0.002\}$. The final value used for the model was cp = 0.076.



| | |
|---|---|
| **Accuracy** | **0.563** |
| **TPR** | **0.447** |
| **FPR** | **0.352** |

**Random Forest:**

Due to computation power limitation we will not crossvalidate parameters of this algorithm. Instead, as a rule of thumb, we will select the parameter mtry = sqrt(ncolums) = 23. We will choose 500 trees and the minimal number of observations in node as 5

| | |
|---|---|
| Mtry | 23 |
| ntree | 500 |
| nodesize | 5 |

We have the results

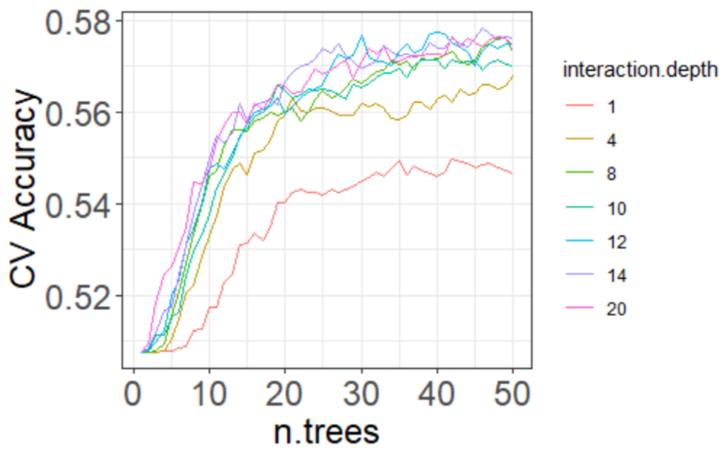
| | |
|---|---|
| **Accuracy** | **0.567** |
| **TPR** | **0.584** |
| **FPR** | **0.447** |

**Boosting:**

We expect this algorithm to have the better accuracy and a good out-of-sample predicting quality, thus we will use cross-validation.

| | |
|---|---|
| shrinkage | 0.01 |
| n.minobsinnode | 10 |
| nodesize | 5 |
| ntrees | 1:50 |
| Interation.depth | 1,4,8,10,12,14,20 |

The final values used for the model were n.trees = 45, interaction.depth = 14, shrinkage = 0.01 and n.minobsinnode = 10.



We have the results

| | |
|---|---|
| **Accuracy** | **0.58** |
| **TPR** | **0.44** |
| **FPR** | **0.29** |

**Summary**

In this case accuracy the variables involved do not need interpretation since they somewhat do not make sense in a typical human's perspective); the goal of our model here is to have the best accuracy.

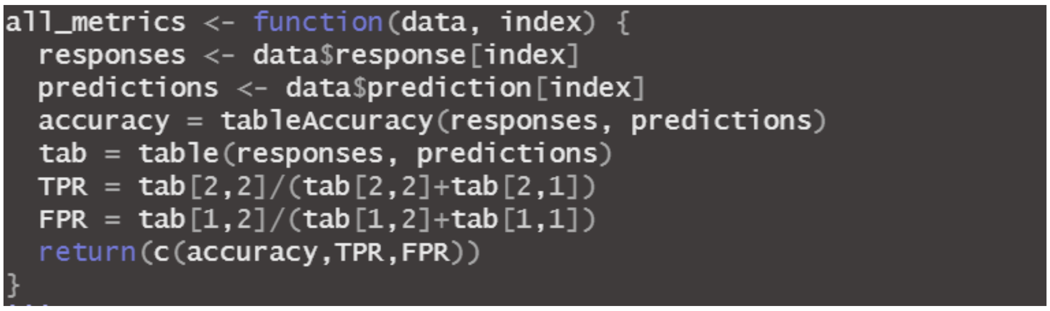
| | **Baseline model** | **Logistic Regression** | **CART** | **Random Forest** | **Boosting** |
|---|---|---|---|---|---|
| **Accuracy** | **0.507** | **0.517** | **0.563** | **0.567** | **0.58** |
| **TPR** | **0** | **0.394** | **0.477** | **0.584** | **0.44** |
| **FPR** | **0** | **0.584** | **0.352** | **0.447** | **0.29** |

Both Random Forest and Boosting have the best accuracies on our test set. The first maximizes the TPR while the second maximizes the FPR. We will select **Boosting** as our final model since it has a slightly better accuracy and the lowest FPR rate (a false negative is less expensive than a false positive; it is better wrongly predict that a question won't be useful than to predict that it would be useful when it won't actually).

In addition to this analysis, we will assess the variability of our performance metrics with bootstrap.

**Bootstrap:**

We create a function all_metrics that summarizes our three metrics accuracy, TPR, FPR



```
all_metrics <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  accuracy = tableAccuracy(responses, predictions)
  tab = table(responses, predictions)
  TPR = tab[2,2]/(tab[2,2]+tab[2,1])
  FPR = tab[1,2]/(tab[1,2]+tab[1,1])
  return(c(accuracy,TPR,FPR))
}
```

Then we will find **95% level confidence intervals** of our metrics on **10000 samples** bootstrapped from our test set. Here are the results

	Logistic Regression	**CART**	**Random Forest**	**Boosting**
Accuracy	$[\mathbf{0.54; 0.58}]$	$[\mathbf{0.54; 0.58}]$	$[\mathbf{0.54; 0.59}]$	$[\mathbf{0.56; 0.60}]$
TPR	$[\mathbf{0.48; 0.54}]$	$[\mathbf{0.44; 0.51}]$	$[\mathbf{0.55; 0.61}]$	$[\mathbf{0.41; 0.47}]$
FPR	$[\mathbf{0.36; 0.42}]$	$[\mathbf{0.32; 0.38}]$	$[\mathbf{0.41; 0.47}]$	$[\mathbf{0.25; 0.31}]$

On overall, Boosting has the best performance in terms of Accuracy and minimizing the FPR. We can even think about increasing its performance through cross validation if we had more computation power.

c) Maximize the probability that a question is useful

i) Select a model

Let us consider the following confusion matrix

	0	1
0	TN	FP
1	FN	TP

$$P_{useful} = \frac{TP}{FP + TP}$$

The probability that a question predicted useful is actually useful.

To accomplish this goal, we would select a model that has a good accuracy (probability of a good prediction) and a maximizes P_{useful}.

	Logistic Regression	**CART**	**Random Forest**	**Boosting**
Accuracy	$\mathbf{0.563}$	$\mathbf{0.565}$	$\mathbf{0.579}$	$\mathbf{0.58}$
Probability	$\mathbf{0.56}$	$\mathbf{0.57}$	$\mathbf{0.56}$	$\mathbf{0.6}$

Boosting is the algorithm with the best accuracy and that maximizes the probability of a question being useful.

ii)

- Baseline model

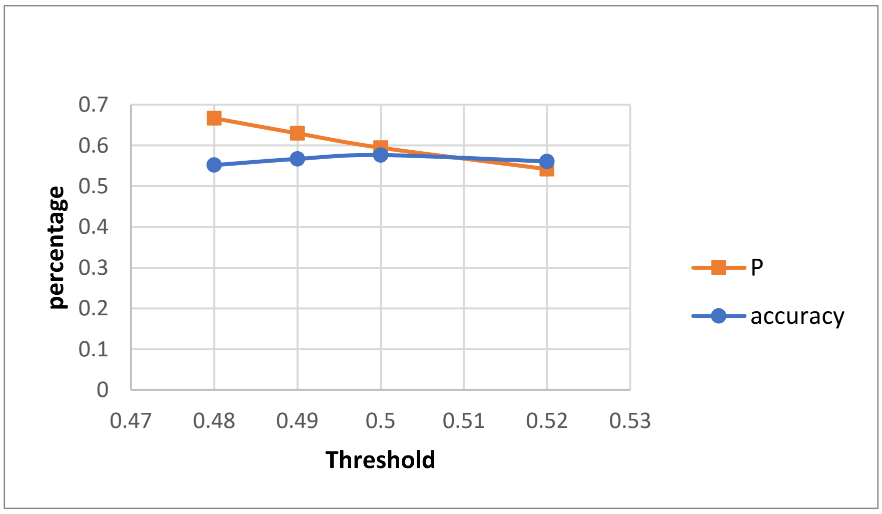
For the current Stack Overflow's algorithm, the probability that among 15 chosen questions the top one is useful is:

$$P_{top\ useful} = \frac{1103}{1103 + 1137} = 0.492$$

- Boosting performances

When building the boosting model, the output of our prediction is a probability. In our first model we put a threshold at 0.5 (We must note here that the probability is flipped in gbm). Now we will change the threshold and see the consequences on the metrics

Threshold	P	accuracy	Increase from baseline
0.52	0.541872	0.560714	10%
0.5	0.594363	0.576339	21%
0.49	0.62963	0.566964	28%
0.48	0.666667	0.551786	36%



Interpretation:

If we reduce the threshold for boosting, it means that we increase the likeliness of a positive prediction to be true. However, we reduce the overall accuracy of the algorithm. There would be less positive predictions, but they would be more likely to be right. Concerning Stack Overflow, there is a risk that in a subset of 15 questions, the algorithm can predict that there is no useful question.

For this purpose, I would recommend taking a threshold of 0.48 that would guaranty an accuracy 55.2% with a probability of having the first question useful increased by 36%.

Load Packages

```
library(ggplot2) #for visu
library(GGally)

library(boot)
library(tm)

library(SnowballC)
library(wordcloud)

## Loading required package: RColorBrewer

library(MASS)
library(caTools)
library(dplyr)

library(rpart)
library(rpart.plot)
library(randomForest)

library(caret)

library(tm.plugin.webmining)

library(ROCR)
```

Question a): Cleaning up the dataset

Function to compute accuracy of a classification model

```
tableAccuracy <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/length(test)
  return(a)
}

metric <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/length(test)
  TPR = t[2,2]/(t[2,2]+t[2,1])
  FPR = t[1,2]/(t[1,2]+t[1,1])
  return(c(TPR,FPR))
}
```

Load the data set

```
Questions = read.csv("ggplot2questions2016_17.csv", stringsAsFactors=FALSE
)
```

Create corpuses of the Title and the body

```
Qtitle = Corpus(VectorSource(Questions$Title))
Qbody = Corpus(VectorSource(Questions$Body))

strwrap(Qbody[[1]])
```

Remove the lower cases and html tags (around 10 min) Also remove all the numbers. They are irrelevant because there is a randomness of their values specific to the user’s problem/answer

```
Qtitle = tm_map(Qtitle, tolower)

## Warning in tm_map.SimpleCorpus(Qtitle, tolower): transformation drops
## documents

Qbody = tm_map(Qbody, tolower )

## Warning in tm_map.SimpleCorpus(Qbody, tolower): transformation drops
## documents

for(i in 1:length(Qbody)) {
  Qbody[[i]] <- gsub("\\d+","\\s",Qbody[[i]])
  Qtitle[[i]] <- gsub("\\d+","\\s",Qtitle[[i]])
  Qbody[[i]] <- extractHTMLStrip(Qbody[[i]])

}

strwrap(Qbody[[1]])
```

Remove punctuation

```
Qtitle = tm_map(Qtitle, removePunctuation)

## Warning in tm_map.SimpleCorpus(Qtitle, removePunctuation): transformati
on
## drops documents

Qbody = tm_map(Qbody, removePunctuation)

## Warning in tm_map.SimpleCorpus(Qbody, removePunctuation): transformatio
n
## drops documents

strwrap(Qbody[[1]])
```

Remove stop words, “ggplot” and “R” since it’s common to all our dataset

```
Qtitle = tm_map(Qtitle, removeWords, c("R", stopwords("english")))

## Warning in tm_map.SimpleCorpus(Qtitle, removeWords, c("R",
## stopwords("english"))): transformation drops documents

Qbody = tm_map(Qbody, removeWords, c("R", stopwords("english")))

## Warning in tm_map.SimpleCorpus(Qbody, removeWords, c("R",
## stopwords("english"))): transformation drops documents

strwrap(Qbody[[1]])
```

Stem the douments

```
Qtitle = tm_map(Qtitle, stemDocument)

## Warning in tm_map.SimpleCorpus(Qtitle, stemDocument): transformation dr
ops
## documents

Qbody = tm_map(Qbody, stemDocument)

## Warning in tm_map.SimpleCorpus(Qbody, stemDocument): transformation dro
ps
## documents

strwrap(Qbody[[1]])
```

Create a word count matrix

```
FreqTitle = DocumentTermMatrix(Qtitle)
FreqBody = DocumentTermMatrix(Qbody)
```

Remove sparse words (words that appears less than 1.5%)

```
sparseTitle = removeSparseTerms(FreqTitle, 0.985)
sparseBody = removeSparseTerms(FreqBody, 0.985)

Essai = as.data.frame(as.matrix(sparseTitle))
Essai2 = as.data.frame(as.matrix(sparseBody))
```

Merge the columns of the title that aren't in the body

```
Gros = Essai2
for(i in 1:ncol(Essai2)) {
  for(j in 1:ncol(Essai)){
    if (names(Essai2[i]) == names(Essai[j])) {
    Gros = Essai2[i]+ Essai[j] }  else {
    Gros = cbind(Essai[j],Essai2)
    }
  }
}
colnames(Gros) = make.names(colnames(Gros))
```

Add the last column with the classification useful or not

```
Gros$rate = as.factor(as.numeric(Questions$Score >= 1))
```

Check if there is any duplicated label in the final dataframe We notice that the algorithm doesn't differenciate "chart." and "chart" we will drop one

```
A = duplicated(names(Gros), incomparables = FALSE)
pos = match(TRUE,A)
Gros[pos] = NULL
```

Questions 2: Building the model

First we separate the training and test set

```
set.seed(123)
spl = sample.split(Gros$rate, SplitRatio = 0.7)
question.train = filter(Gros, spl == TRUE)
question.test = filter(Gros, spl == FALSE)
```

Baseline accuracy on the test set

```
x = table(question.test$rate)
x

##
##    0    1
## 1137 1103

base_acc = x[[1]]/sum(x)
base_acc

## [1] 0.5075893

#Accuracy of Baseline model 0.507
```

Random forest we choose mtry = sqrt(ncolumns)

```
set.seed(311)

mod.rf <- randomForest(rate ~ ., data = question.train, mtry = 23, nodesiz
e = 5, ntree = 500)
predict.rf = predict(mod.rf, newdata = question.test)


table(question.test$rate, predict.rf)

##    predict.rf
##       0   1
##   0 665 472
##   1 469 634

rf_acc = tableAccuracy(question.test$rate, predict.rf)
rf_acc

## [1] 0.5799107

rf_param = metric(question.test$rate, predict.rf)
rf_param

## [1] 0.5747960 0.4151275

#rf_acc = 0.58
```

Cart (cp cross validated)

```
set.seed(311)
train.cart = train(rate ~ .,
                   data = question.train,
                   method = "rpart",
                   tuneGrid = data.frame(cp=seq(0, 0.4, 0.002)),
                   trControl = trainControl(method="cv", number=10))
train.cart

train.cart$results

ggplot(train.cart$results, aes(x = cp, y = Accuracy)) +
  geom_point(size = 2) +
  geom_line() +
  ylab("CV Accuracy") +
  theme_bw() +
  theme(axis.title=element_text(size=18), axis.text=element_text(size=18))
```

```
mod.cart = train.cart$finalModel
prp(mod.cart)
```

```
predict.cart = predict(mod.cart, newdata = question.test, type = "class")
# why no model.matrix?
table(question.test$rate, predict.cart)

##    predict.cart
##       0   1
##   0 704 433
##   1 546 557

cart_acc = tableAccuracy(question.test$rate, predict.cart)
cart_acc

## [1] 0.5629464

cart_param = metric(question.test$rate, predict.cart)
cart_param

## [1] 0.5049864 0.3808267

# cart_acc = 0.565
#Accuracy was used to select the optimal model using the largest value.
#The final value used for the model was cp = 0.076.
```

Logistic Regression

```
train.log = glm(rate ~ ., data = question.train, family = "binomial")

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

PredictLog = predict(train.log, newdata = question.test, type ="response")
tabLog = table(question.test$rate, PredictLog > 0.5)
tabLog

##
##     FALSE TRUE
##   0   696  441
##   1   532  571
```

```
logReg_acc = tableAccuracy(question.test$rate, PredictLog > 0.5)
TPR = tabLog[2,2]/(tabLog[2,2]+tabLog[2,1])
FPR = tabLog[1,2]/(tabLog[1,2]+tabLog[1,1])
TPR

## [1] 0.5176791

FPR

## [1] 0.3878628

rocr.log.pred <- prediction(PredictLog , question.test$rate)
logPerformance <- performance(rocr.log.pred, "tpr", "fpr")
plot(logPerformance, colorize = TRUE)
abline(0, 1)
```

```
as.numeric(performance(rocr.log.pred, "auc")@y.values)

## [1] 0.5798227

#LogReg_acc = 0.5638393
```

Boosting

```
tGrid = expand.grid(n.trees = (1:50), interaction.depth = c(1,4,8,10,12,14
,20),
                    shrinkage = 0.01, n.minobsinnode = 10)
set.seed(232)

train.boost <- train(rate ~ .,
                     data = question.train,
                     method = "gbm",
                     tuneGrid = tGrid,
                     trControl = trainControl(method="cv", number=5, verbo
seIter = TRUE),
                     metric = "Accuracy",
                     distribution = "bernoulli")




train.boost$results

ggplot(train.boost$results, aes(x = n.trees, y = Accuracy, colour = as.fac
tor(interaction.depth))) + geom_line() +
  ylab("CV Accuracy") + theme_bw() + theme(axis.title=element_text(size=18
), axis.text=element_text(size=18)) +
  scale_color_discrete(name = "interaction.depth")
```

```
mod.boost = train.boost$finalModel

question.test.mm = as.data.frame(model.matrix(rate ~ . +0, data = question
.test))
predict.boost = predict(mod.boost, newdata = question.test.mm, n.trees = 3
000, type = "response")

## Warning in predict.gbm(mod.boost, newdata = question.test.mm, n.trees =
## 3000, : Number of trees not specified or exceeded number fit so far. Us
ing
## 49.

taboost = table(question.test$rate, predict.boost < 0.52) # for some reaso
n the probabilities are flipped in gbm
taboost

##
##      FALSE TRUE
##    0   493  644
##    1   347  756

boost_ac = tableAccuracy(question.test$rate, predict.boost < 0.52)
boost_ac

## [1] 0.5575893

TPRboost = taboost[2,2]/(taboost[2,2]+taboost[2,1])
FPRboost = taboost[1,2]/(taboost[1,2]+taboost[1,1])
precision = taboost[2,2]/(taboost[1,2]+taboost[2,2])
precision

## [1] 0.54

TPRboost

## [1] 0.6854034

FPRboost

## [1] 0.5664028

#boosting accuracy 0.58
```

Tuning parameter ‘shrinkage’ was held constant at a value of 0.01 Tuning parameter ‘n.minobsinnode’ was held constant at a value of 10 Accuracy was used to select the optimal model using the largest value. The final values used for the model were n.trees = 45, interaction.depth = 14, shrinkage = 0.01 and n.minobsinnode = 10.

Now we are going to Boostrap to assess the variability of the performance metrics through boostrap

```
all_metrics <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  accuracy = tableAccuracy(responses, predictions)
  tab = table(responses, predictions)
  TPR = tab[2,2]/(tab[2,2]+tab[2,1])
  FPR = tab[1,2]/(tab[1,2]+tab[1,1])
```

```r
  return(c(accuracy,TPR,FPR))
}

          ###### Random Forests ######
RF_test_set = data.frame(response = question.test$rate, prediction = predi
ct.rf)

# do bootstrap
set.seed(892)
RF_boot <- boot(RF_test_set, all_metrics, R = 10000)
RF_boot

#Confidence interalls
boot.ci(RF_boot, index = 1, type = "basic")

boot.ci(RF_boot, index = 2, type = "basic")

boot.ci(RF_boot, index = 3, type = "basic")



        ###### CART ############
cart_test_set = data.frame(response = question.test$rate, prediction = pre
dict.cart)

# do bootstrap
set.seed(892)
cart_boot <- boot(cart_test_set, all_metrics, R = 10000)
cart_boot

#Confidence interalls
boot.ci(cart_boot, index = 1, type = "basic")

boot.ci(cart_boot, index = 2, type = "basic")

boot.ci(cart_boot, index = 3, type = "basic")

        ###### Logistic Regression ######
LR_test_set = data.frame(response = question.test$rate, prediction = as.nu
meric(PredictLog > 0.5))

# do bootstrap
set.seed(892)
LR_boot <- boot(LR_test_set, all_metrics, R = 10000)
LR_boot

#Confidence interalls
boot.ci(LR_boot, index = 1, type = "basic")

boot.ci(LR_boot, index = 2, type = "basic")

boot.ci(LR_boot, index = 3, type = "basic")

        ###### Boosting ######
boosting_test_set = data.frame(response = question.test$rate, prediction =
as.numeric(predict.boost<0.5))
```

```
# do bootstrap
set.seed(892)
boosting_boot <- boot(boosting_test_set, all_metrics, R=10000)

#Confidence interalls
boot.ci(boosting_boot, index = 1, type = "basic")

boot.ci(boosting_boot, index = 2, type = "basic")

boot.ci(boosting_boot, index = 3, type = "basic")
```

Part C:

```
performance<-function(model){
  moy = 0
  for (i in c(1:3000)){
  test=sample_n(question.test,size=15,replace = FALSE)
  test.mm = as.data.frame(model.matrix(rate ~ . +0, data = test))
  pred = predict(model, newdata = test.mm,n.trees=45, type = "response")
  n=sum(as.numeric(as.numeric(test$rate ==1) & as.numeric(pred < 0.50)))
  d = sum(as.numeric(test$rate ==1))
  p = n/d
  moy = moy + p
  }
  return (moy/3000)
}
eval2 = performance(mod.boost)
eval2

## [1] 0.4341699
```