Nicolas Kardous
IND 242 HW4
Paul Grigas

Homework Assignment #4

Problem 1: Predicting Useful Questions on Stack Overflow

1. The first step is to load the necessary library files that is shown below, and to read the excel file, in the case of our code, we are naming the excel as 'data'.

```
library(tm)
library(SnowballC)
library(wordcloud)
library(MASS)
library(caTools)
library(dplyr)
library(rpart)
library(rpart.plot)
library(randomForest)
library(caret)
library(tm.plugin.webmining)

# Problem 1

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

2. We then add a column that we call 'useful' for when the score is greater than or equal to 1.

```
# Add a column for Useful posts

data$Useful = as.factor(as.numeric(data$Score >= 1))
```

3. We then extract html strips from our data using the code below.

```
for (i in seq(1,length(data$Body),1)) {
  data$Body[i]=extractHTMLStrip(data$Body[i])
  }
```

4. We also notice in the body and title that there are a lot of numbers. So, we use regex and gsub to get rid of any numbers in our title and body.

```
data$Title = gsub("[0-9]*", "", data$Title)
data$Body = gsub("[0-9]*", "", data$Body)
```

5. After this, we now convert our title and body into a corpus. Here, we are separating our title and our body.

```
# A vector source interprets each element of the vector as a document.
# Corpus creates a collection of documents
corpusTitle = Corpus(VectorSource(data$Title))
corpusBody = Corpus(VectorSource(data$Body))
# The titles and body are now "documents"
```

6. We also need to convert the text in the title and body into lowercase.

```
# tm_map applies an operation to every document in our corpus
# Here, that operation is 'tolower', i.e., 'to lowercase'
corpusTitle = tm_map(corpusTitle, tolower)
corpusBody = tm_map(corpusBody, tolower)
# tolower is a function
```

7. The next step is that we remove all the punctuation found in the title and the body

```
corpusTitle = tm_map(corpusTitle, removePunctuation)
corpusBody = tm_map(corpusBody, removePunctuation)
```

8. The next step is to remove stopwords from the title and body. I also found that the words 'ggplot' and 'ggplot2' were used very often in the title and body, and so I chose to get rid of those words too.

```
# Remove stopword,these are words common in the title and body
corpusTitle = tm_map(corpusTitle, removeWords, c("apple", "ggplot","ggplot2",stopwords("english")))
corpusBody = tm_map(corpusBody, removeWords, c("apple", "ggplot","ggplot2",stopwords("english")))
```

9. We then stem the title and body words, to make slightly varying words into the same word.

```
# We chop off the ends of words that aren't maybe
# as necessary as the rest, like 'ing' and 'ed'
corpusTitle = tm_map(corpusTitle, stemDocument)
corpusBody = tm_map(corpusBody, stemDocument)
```

10. Here, we have cleaned our data sufficiently, and now we want to utilize sparsity to get rid of words that barely ever show up. In this instance, we chose a sparsity percentage of 98% for the Title, and 91% for the Body. Given this, we get 45 feature words for the title, and 88 feature words for the Body.

```
# We currently have way too many words, which will make it hard to train
# our models and may even lead to overfitting.
# Use findFreqTerms to get a feeling for which words appear the most

# Words that appear at least 50 times:
findFreqTerms(frequenciesTitle, lowfreq=50)
findFreqTerms(frequenciesBody, lowfreq=50)
# Words that appear at least 20 times:
findFreqTerms(frequenciesTitle, lowfreq=20)
findFreqTerms(frequenciesBody, lowfreq=20)

# Our solution to the possibility of overfitting is to only keep terms
# that appear in x% or more of the posts. For example:
# 5% of the posts or more for the title, and 9% or more for the body
# Here we will get 45 variables for the title, and 88 variables for the body
sparseTitle = removeSparseTerms(frequenciesTitle, 0.98)
sparseBody = removeSparseTerms(frequenciesBody, 0.91)
```

11. Our next step is to create a data frame for both the title and body. There are also some variable names that start with a number, so we also utilize the 'colnames' function to get rid of this issue.

```
dataTitle = as.data.frame(as.matrix(sparseTitle))
dataBody = as.data.frame(as.matrix(sparseBody))

# We have some variable names that start with a number,
# which can cause R some problems. Let's fix this before going
# any further
colnames(dataTitle) = make.names(colnames(dataTitle))
colnames(dataBody) = make.names(colnames(dataBody))
```

12. Finally, we want to combine our title and body data frames. To do this, we utilize the "%in%" parameter to find instances where the features are found in both the title and

body data frame. We then sum the frequencies for where both features are found in the title and body dataframe. Then, we create two new dataframes, one for body and one for column, where we get rid of the features that are found in both the original body and title dataframes. Finally, we use 'cbind' to combine the dataframe for the sum of frequencies that is found in both title and body, the dataframe for body where we got rid of the overlapping features, and the dataframe for title where we got rid of the overlapping features. In this case, our final combined dataset is called 'datajoined'. All of this is shown in the code below.

```
duplicatedBody = dataBody[, names(dataBody) %in% names(dataTitle)]
duplicatedTitle = dataTitle[, names(dataTitle) %in% names(dataBody)]
duplicatedTotal = duplicatedBody + duplicatedTitle
Bodynew = select(dataBody, -c(names(duplicatedTitle)))
Titlenew = select(dataTitle, -c(names(duplicatedTitle)))
datajoined = as.data.frame(cbind(duplicatedTotal,Bodynew,Titlenew))
```

13. Our last step is to include the "Useful" term as a new column into our combined dataset.

```
datajoined$Useful = data$Useful
```

Problem 1: Part b)

1. We start off with splitting our data into a training set and a testing set. We also created a function to calculate the accuracy

```
spl = sample.split(datajoined$Useful, SplitRatio = 0.7)

Train <- datajoined %>% filter(spl == TRUE)
Test <- datajoined %>% filter(spl == FALSE)

table(Train$Useful)
table(Train$Useful)

# Function to compute accuracy of a classification model
tableAccuracy <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/sum(t)
  return(a)
}
```

```
> table(Train$Useful)

   0    1
2654 2574
> table(Test$Useful)

   0    1
1137 1103
```

Here, we find that the accuracy on our testing set, based on a baseline model is (1103/(1103+1137) = 49.24%

2. The first model we are going to implement is a basic random forest model. We train our random forest model on the training set, and we are predicting how well it performs on the testing set.

```
> table(Test$Useful, PredictRF)
   PredictRF
      0   1
  0 672 465
  1 540 563
> tableAccuracy(Test$Useful, PredictRF)
[1] 0.5513393
```

```
# Basic Random Forests:
DataRF = randomForest(Useful ~ ., data=Train)
PredictRF = predict(DataRF, newdata = Test)
table(Test$Useful, PredictRF)
tableAccuracy(Test$Useful, PredictRF)
```

Looking at our random forest model, we see that we get an accuracy of 55.13%. Additionally, we have a True Positive Rate = 563/(563+540) = 51.04% and we have a False Positive Rate = 465/(465+672) = 40.90%

3. We want to improve upon our basic random forest model by finding a suitable value for 'mtry', where 'mtry' is the number of features we randomly choose from for each step of the decision tree. Because of constraints on time and computer processing power, we chose mtry values ranging from 1 to 10, and a fold value of 5. Given these, we found that the optimal value for mtry is 3. This is shown in the code below.

```
# Random Forest with changing mtry:

DataRF2 <- train(Useful ~.,
                 data = Train,
                 method = "rf", # random forest
                 tuneGrid = data.frame(mtry=1:10),
                 trControl = trainControl(method="cv", number=5, verboseIter = TRUE),
                 metric = "Accuracy")

DataRF2.best <- DataRF2$finalModel
PredictRF2 = predict(DataRF2.best, newdata = Test)
table(Test$Useful, PredictRF2)
tableAccuracy(Test$Useful, PredictRF2)
```

```
Aggregating results
Selecting tuning parameters
Fitting mtry = 3 on full training set
```

```
> table(Test$Useful, PredictRF2)
   PredictRF2
      0   1
  0 722 415
  1 586 517
> tableAccuracy(Test$Useful, PredictRF2)
[1] 0.553125
```

The accuracy from this model is slightly higher than the previous basic random forest model, and it has an accuracy of 55.31%, which is actually lower than the accuracy from the basic random forest model above. For this updated Random Forest model, we have a True Positive Rate = 517/(517+586) = 46.87% and we have a False Positive Rate = 415/(415+722) = 36.5%

4. The next model we are going to implement is utilizing a CART model with cross validation.

```
# Cross-validated CART model
train.cart = train(Useful ~ .,
                   data = Train,
                   method = "rpart",
                   tuneGrid = data.frame(cp=seq(0, 0.4, 0.002)),
                   trControl = trainControl(method="cv", number=10))
train.cart
train.cart$results
train.cart$bestTune
```

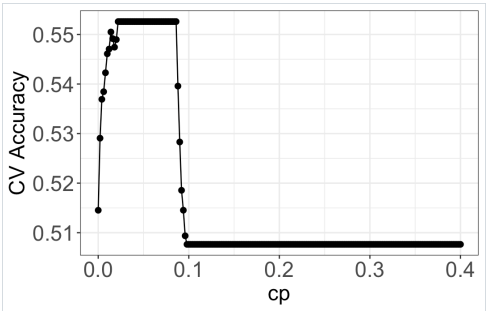
```
> train.cart$bestTune
      cp
44 0.086
```

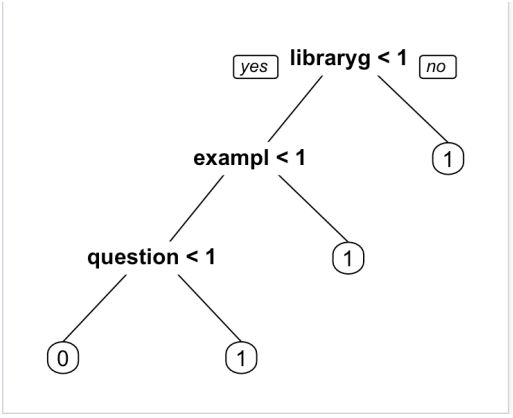
From this, we find that our CART model performs the best with a complexity parameter value of 0.086. From the code below, then we show a graph demonstrating the relationship between the complexity parameter and the accuracy. The next diagram is a sample CART model for what our decision tree will look like.

```
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 = Test, type = "class")
table(Test$Useful, predict.cart)
tableAccuracy(Test$Useful, predict.cart)
```






```
> table(Test$Useful, predict.cart)
   predict.cart
      0   1
  0 870 267
  1 740 363
> tableAccuracy(Test$Useful, predict.cart)
[1] 0.5504464
```

From looking at our CART tree, we see that the most important feature that would lead to the lowest cost is the feature “libraryg”. For our CART model we get an accuracy of 55.04%, a True Positive Rate = 363/(740+363) = 32.91% and a False Positive Rate = 23.48%.

```
# What about CART on training set?
PredictCARTTrain = predict(mod.cart, type = "class")
table(Train$Useful, PredictCARTTrain)
tableAccuracy(Train$Useful, PredictCARTTrain)
```

```
> table(Train$Useful, PredictCARTTrain)
   PredictCARTTrain
      0    1
  0 1898  756
  1 1855  719
> tableAccuracy(Train$Useful, PredictCARTTrain)
[1] 0.5005738
```

What is interesting is that when we run our CART model on our training data, we get an accuracy of 50.06%. This is interesting because our model performed better on the testing data where we got an accuracy of 55.04%.

5. The next model we run is a logistic regression model. The code for this is shown below.

```
# Logistic Regression

DataLog = glm(Useful ~ ., data = Train, family = "binomial")
summary(DataLog)

# Predictions on test set
PredictLog = predict(DataLog, newdata = Test, type = "response")
table(Test$Useful, PredictLog > 0.5)
tableAccuracy(Test$Useful, PredictLog > 0.5)

# But what about training set?
PredictLogTrain = predict(DataLog, type = "response")
table(Train$Useful, PredictLogTrain > 0.5)
tableAccuracy(Train$Useful, PredictLogTrain > 0.5)
```

```
> table(Test$Useful, PredictLog > 0.5)

   FALSE TRUE
  0   757  380
  1   547  556
> tableAccuracy(Test$Useful, PredictLog > 0.5)
[1] 0.5861607
>
> # But what about training set?
> PredictLogTrain = predict(DataLog, type = "response")
> table(Train$Useful, PredictLogTrain > 0.5)

   FALSE TRUE
  0  1787  867
  1  1181 1393
> tableAccuracy(Train$Useful, PredictLogTrain > 0.5)
[1] 0.6082632
```

From our logistic regression, we see that the accuracy on our test set is 58.62%, the True Positive Rate = 556/(556+547) = 50.4% and the False Positive Rate = 380/(380+757) = 33.42%. The accuracy we get when we run our logistic regression model on our training data is 60.83%, and this should make sense, because we should expect a higher accuracy for our model on the training data.

6. The next model we are running is a Linear Discriminant Analysis model. This is shown in the code below.

```
# Linear Discriminant Analysis
library(MASS)
lda.mod = lda(Useful ~ ., data = Train)

predict.lda = predict(lda.mod, newdata = Test)$class
table(Test$Useful, predict.lda)
tableAccuracy(Test$Useful, predict.lda)
```

```
> table(Test$Useful, predict.lda)
   predict.lda
      0   1
  0 769 368
  1 553 550
> tableAccuracy(Test$Useful, predict.lda)
[1] 0.5888393
```

From the Linear Discriminant Analysis, we get an accuracy of 58.88%. We have a True Positive Rate = 550/(550+553) = 49.86% and a False Positive Rate = 368/(368+769) = 32.37%.

Model	Accuracy	TPR	FPR
Basic Random Forest	55.13%	51.04%	40.90%
Random Forest with changing mtry	55.31%	46.87%	36.5%
CART	55.04%	32.91%	23.48%
Logistic Regression	58.62%	50.4%	33.42%
Linear Discriminant Analysis	58.88%	49.86%	32.37%

Problem 1: Part b) Bootstrapping

When it comes to implementing bootstrapping on our data, we are going to look at how well our Random Forest, CART, Logistic Regression, and Linear Discriminant Analysis models perform. We first create functions for Accuracy, FPR and TPR, this is shown in the code below.

```
library(boot)

TPR <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  contingency <- table(responses, predictions)
  TPR <- contingency[2, 2]/(contingency[2,2]+contingency[2,1])
  return(TPR)
}

FPR <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  contingency <- table(responses, predictions)
  FPR <- contingency[1, 2]/(contingency[1, 2] + contingency[1, 1])
  return(FPR)
}

Accuracy <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  t = table(responses, predictions)
  a = sum(diag(t))/length(responses)
  return(a)
}

all_metrics <- function(data, index) {
  accuracy <- Accuracy(data, index)
  tpr <- TPR(data, index)
  fpr <- FPR(data, index)
  return(c(accuracy, tpr, fpr))
}
```

1. We start with our basic random forest model. Our calculations is show in the code below.

```
###### Basic Random Forest ######
RF_test_set = data.frame(response = Test$Useful, prediction = PredictRF)

# do bootstrap
RF_boot <- boot(RF_test_set, all_metrics, R = 10000)
RF_boot

# get confidence intervals (not manually)
boot.ci(RF_boot, index = 1, type = "basic")
boot.ci(RF_boot, index = 2, type = "basic")
boot.ci(RF_boot, index = 3, type = "basic")
```

```
CALL :
boot.ci(boot.out = RF_boot, type = "basic", index = 1)

Intervals :
Level      Basic
95%   ( 0.5304,  0.5719 )
Calculations and Intervals on Original Scale
> boot.ci(RF_boot, index = 2, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

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

Intervals :
Level      Basic
95%   ( 0.4808,  0.5395 )
Calculations and Intervals on Original Scale
> boot.ci(RF_boot, index = 3, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

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

Intervals :
Level      Basic
95%   ( 0.3808,  0.4370 )
Calculations and Intervals on Original Scale
```

2. We implement the bootstrap on our next model which is another Random Forest where we set mtry = 3. The results are shown in the code below.

```
###### Random Forest with changing mtry ######

RF_test_set2 = data.frame(response = Test$Useful, prediction = PredictRF2)

# do bootstrap
RF_boot2 <- boot(RF_test_set, all_metrics, R = 10000)
RF_boot2

# get confidence intervals (not manually)
boot.ci(RF_boot2, index = 1, type = "basic")
boot.ci(RF_boot2, index = 2, type = "basic")
boot.ci(RF_boot2, index = 3, type = "basic")
```

```
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = RF_boot2, type = "basic", index = 1)

Intervals :
Level      Basic
95%   ( 0.5304,  0.5723 )
Calculations and Intervals on Original Scale
> boot.ci(RF_boot2, index = 2, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = RF_boot2, type = "basic", index = 2)

Intervals :
Level      Basic
95%   ( 0.4811,  0.5393 )
Calculations and Intervals on Original Scale
> boot.ci(RF_boot2, index = 3, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = RF_boot2, type = "basic", index = 3)

Intervals :
Level      Basic
95%   ( 0.3803,  0.4381 )
Calculations and Intervals on Original Scale
```

3. We implement bootstrapping on our next model which is the CART model. Our code is shown below.

```
###### CART ######
CART_test_set = data.frame(response = Test$Useful, prediction = predict.cart)

# do bootstrap
CART_boot <- boot(CART_test_set, all_metrics, R = 10000)
CART_boot

# get confidence intervals
boot.ci(CART_boot, index = 1, type = "basic")
boot.ci(CART_boot, index = 2, type = "basic")
boot.ci(CART_boot, index = 3, type = "basic")
```

```
CALL :
boot.ci(boot.out = CART_boot, type = "basic", index = 1)

Intervals :
Level      Basic
95%   ( 0.5308,  0.5710 )
Calculations and Intervals on Original Scale
> boot.ci(CART_boot, index = 2, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = CART_boot, type = "basic", index = 2)

Intervals :
Level      Basic
95%   ( 0.5287,  0.5860 )
Calculations and Intervals on Original Scale
> boot.ci(CART_boot, index = 3, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = CART_boot, type = "basic", index = 3)

Intervals :
Level      Basic
95%   ( 0.4260,  0.4844 )
Calculations and Intervals on Original Scale
```

4. We then implement bootstrapping on our next model which is the Logistic Regression. Our code is shown below.

```
###### Logistic Regression ######
Log_test_set = data.frame(response = Test$Useful, prediction = PredictLog)

# do bootstrap
Log_boot <- boot(Log_test_set, all_metrics, R = 10000)
Log_boot

# get confidence intervals
boot.ci(Log_boot, index = 1, type = "basic")
boot.ci(Log_boot, index = 2, type = "basic")
boot.ci(Log_boot, index = 3, type = "basic")
```

```
CALL :
boot.ci(boot.out = Log_boot, type = "basic", index = 1)

Intervals :
Level      Basic
95%   (-0.0013,  0.0009 )
Calculations and Intervals on Original Scale
```

5. Finally, we implement bootstrapping on our last model which is the Linear Discriminant Analysis.

```
###### Linear Discriminant Analysis ######
LDA_test_set = data.frame(response = Test$Useful, prediction = predict.lda)

# do bootstrap
LDA_boot <- boot(LDA_test_set, all_metrics, R = 10000)
LDA_boot

# get confidence intervals
boot.ci(LDA_boot, index = 1, type = "basic")
boot.ci(LDA_boot, index = 2, type = "basic")
boot.ci(LDA_boot, index = 3, type = "basic")
```

```
> boot.ci(LDA_boot, index = 1, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = LDA_boot, type = "basic", index = 1)

Intervals :
Level      Basic
95%   ( 0.5219,  0.5621 )
Calculations and Intervals on Original Scale
> boot.ci(LDA_boot, index = 2, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = LDA_boot, type = "basic", index = 2)

Intervals :
Level      Basic
95%   ( 0.4275,  0.4864 )
Calculations and Intervals on Original Scale
> boot.ci(LDA_boot, index = 3, type = "basic")
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 10000 bootstrap replicates

CALL :
boot.ci(boot.out = LDA_boot, type = "basic", index = 3)

Intervals :
Level      Basic
95%   ( 0.3477,  0.4039 )
Calculations and Intervals on Original Scale
```

Model	Accuracy		TPR		FPR	
Bound	**Lower**	**Upper**	**Lower**	**Upper**	**Lower**	**Upper**
Basic Random Forest	53.04%	57.19%	48.08%	53.95%	38.08%	43.70%
Random Forest with changing mtry	53.04%	57.23%	48.11%	53.93%	38.03%	43.81%
CART	53.08%	57.10%	52.87%	58.60%	42.60%	48.44%
Logistic Regression	-0.13%	0.009%	50.4%	50.4%	33.42%	33.42%
Linear Discriminant Analysis	52.19%	56.21%	42.75%	48.64%	34.77%	40.39%

What is interesting from implementing bootstrapping in all our models, is we get to see the upper bounds and lower bounds for all the accuracies for all our models. Because I did not set a seed in my models, I actually ended up getting a lower accuracy for the upper bound for LDA, compared to the accuracy of LDA I computed before. Additionally, bootstrapping was not effective on the Logistic Regression model, where we get really small percentage values for accuracy, and error values for the TPR and FPR. Additionally, from looking at these values, the CART model gave us the highest upper bound for the TPR, and it was a value of 58.6%. A high TPR is very important because we want a high percentage of saying a model is useful, when it actually is.

If we choose our optimal model based on the highest accuracy value, our model will either be the logistic regression model or the LDA model, with accuracy values of 58.62% and 58.88% respectively. We should also note that our CART model got an accuracy of 55.04% when running on the testing set. This shows that our CART model could work really well on future testing data. However, I believe our optimal model in this case will be using the Random Forest model. This is because if we have sufficient time and computer processing, we could run a Random Forest model that could run through all the possible values of mtry and all the folds and find a value of mtry and fold that will lead to the highest possible accuracy. Additionally, the accuracy we got for the Random Forest model was 55.31%, which is not that far from 58.88%, and with an improved Random Forest model, I believe that the accuracy could definitely surpass the accuracy for the Logistic Regression and LDA.

Problem 1: Part c)

i)

We want to maximize the probability that the top question is useful. The criteria I would like to use is a model with the highest accuracy. Given our models on Basic Random Forest, Random Forest with changing the mtry value, CART, Logistic Regression, and Linear Discriminant Analysis, we received the highest accuracy values for the Logistic Regression Model and the LDA. If we were a data scientist working for StackOverflow, and we had to update the model on a daily basis given the new posts every day, then time will be a constraint. If time is a constraint, then it will not be realistic to run a Random Forest model where we are tuning through mtry and fold values to find the optimal values. This is because given 100+ features, tuning through all these parameters would take a lot of time. Thus, even though a Random Forest model could be more accurate, we will not be selecting it, based on the time constraint criteria. For me, I am deciding my biggest criteria to be the model with the highest accuracy value. This is because I believe the larger the dataset, or the more questions there are, the more beneficial a model with a higher accuracy would give us. Now, our Logistic Regression model has an accuracy of 58.62% and our LDA has an accuracy of 58.88%. Thus, our LDA model has a slightly higher accuracy. However, it is important to note that our Logistic Regression model has a True Positive Rate, but also a higher False Positive Rate. Even though the Logistic Regression model has a higher True Positive Rate, it also has a higher False Positive Rate, which

negates the benefit of a higher TPR. Because our ultimate objective is to maximize the probability that the top question is useful, we are going to choose the model with the highest accuracy, which in our case, is the Linear Discriminant Analysis model.

ii)

So, from part b), we thought the best model to be the Random Forest Model, because we could train the model across a wide variety mtry and fold values and get a very high accuracy. Additionally, the upper bound from our random forest model was 57.23% which is very high, but not as high as the accuracy value we first got for Logistic Regression and LDA which was 58.62% and 58.88% respectively. In part c)i), we then said instead of the Random Forest model, an LDA model would be more realistic to utilize because there is less of a time constraint with this model. I said that if we were to train a model every day, it would be difficult to do that with a Random Forest, and so LDA works well because it takes less time and has a high accuracy.

However, if we want to maximize the probability that the top question is useful, then a good metric to follow is to have a lower False Positive Rate. This is because we want to reduce the chance that we are saying a question is useful, when it actually is not. The models with the lowest FPR are the CART model and the LDA. If we choose our selected of Linear Discriminant Analysis, it improves Stack Overflow's current approach of showing the most recent posts fire, because our model will make a prediction of which of those 15 most recent posts are useful, and thus Stack Overflow could list those 15 most recent posts based on what our LDA model predicted for which questions were useful. Additionally, if we looked implementing this model for regression instead of classification, we could rank the questions based on which ones had the highest "useful score". From the baseline model, we see that we get an accuracy of 49.24%. If we implement our LDA model, which has an accuracy of 58.88%, we will have an increase in accuracy of 58.88-49.24 = 9.64%. From this, we will see that our estimate will increase the probability that the top question is useful by 9.64%.

CODE

```
# IND242HW4
# Nicolas Kardous

# First, install the required packages
install.packages("tm")
install.packages("SnowballC")

library(tm)
library(SnowballC)
library(wordcloud)
library(MASS)
library(caTools)
library(dplyr)
library(rpart)
library(rpart.plot)
library(randomForest)
library(caret)
library(tm.plugin.webmining)

# Problem 1

data = read.csv("ggplot2questions2016_17.csv", stringsAsFactors=FALSE)

# Add a column for Useful posts

data$Useful = as.factor(as.numeric(data$Score >= 1))

# Step 1: Extract the html strips from the data

for (i in seq(1,length(data$Body),1)) {
  data$Body[i]=extractHTMLStrip(data$Body[i])
  }

# Step 2: Get rid of all the numbers in the title and body

data$Title = gsub("[0-9]*", "", data$Title)
data$Body = gsub("[0-9]*", "", data$Body)

# Step 3: Convert title and body to a "corpus"

# A vector source interprets each element of the vector as a document.
# Corpus creates a collection of documents
corpusTitle = Corpus(VectorSource(data$Title))
corpusBody = Corpus(VectorSource(data$Body))
# The titles and body are now "documents"

# Step 4: Change all the text to lower case.
# tm_map applies an operation to every document in our corpus
# Here, that operation is 'tolower', i.e., 'to lowercase'
```

```
corpusTitle = tm_map(corpusTitle, tolower)
corpusBody = tm_map(corpusBody, tolower)
# tolower is a function

# Step 5: Remove all punctuation
corpusTitle = tm_map(corpusTitle, removePunctuation)
corpusBody = tm_map(corpusBody, removePunctuation)

# Step 6: Remove stop words
# Remove stopword,these are words common in the title and body
corpusTitle = tm_map(corpusTitle, removeWords, c("apple", "ggplot","ggplot2",stopwords("english")))
corpusBody = tm_map(corpusBody, removeWords, c("apple", "ggplot","ggplot2",stopwords("english")))

# Step 7: Stem our document
# We chop off the ends of words that aren't maybe
# as necessary as the rest, like 'ing' and 'ed'
corpusTitle = tm_map(corpusTitle, stemDocument)
corpusBody = tm_map(corpusBody, stemDocument)

# Step 8: Create a word count matrix (rows are each post, columns are words)
# We've finished our basic cleaning, so now we want to calculate frequencies
# of words across each post
frequenciesTitle = DocumentTermMatrix(corpusTitle)
frequenciesBody = DocumentTermMatrix(corpusBody)

# Step 9: Account for sparsity
# We currently have way too many words, which will make it hard to train
# our models and may even lead to overfitting.
# Use findFreqTerms to get a feeling for which words appear the most

# Words that appear at least 50 times:
findFreqTerms(frequenciesTitle, lowfreq=50)
findFreqTerms(frequenciesBody, lowfreq=50)
# Words that appear at least 20 times:
findFreqTerms(frequenciesTitle, lowfreq=20)
findFreqTerms(frequenciesBody, lowfreq=20)

# Our solution to the possibility of overfitting is to only keep terms
# that appear in x% or more of the posts. For example:
# 2% of the posts or more for the title, and 9% or more for the body
# Here we will get 45 variables for the title, and 88 variables for the body
sparseTitle = removeSparseTerms(frequenciesTitle, 0.98)
sparseBody = removeSparseTerms(frequenciesBody, 0.91)

# Step 10: Create data frame from the document-term matrix, we have 45 variables for the title,
# and 88 variables for the body
dataTitle = as.data.frame(as.matrix(sparseTitle))
dataBody = as.data.frame(as.matrix(sparseBody))

# We have some variable names that start with a number,
# which can cause R some problems. Let's fix this before going
# any further
colnames(dataTitle) = make.names(colnames(dataTitle))
```

```
colnames(dataBody) = make.names(colnames(dataBody))

# Step 11: Join the independent variables together

duplicatedBody = dataBody[, names(dataBody) %in% names(dataTitle)]
duplicatedTitle = dataTitle[, names(dataTitle) %in% names(dataBody)]
duplicatedTotal = duplicatedBody + duplicatedTitle
Bodynew = select(dataBody, -c(names(duplicatedTitle)))
Titlenew = select(dataTitle, -c(names(duplicatedTitle)))
datajoined = as.data.frame(cbind(duplicatedTotal,Bodynew,Titlenew))

# Step 12: Create new column to say a score is positive if it is greater than or equal to 1
datajoined$Useful = data$Useful

# Part b)

spl = sample.split(datajoined$Useful, SplitRatio = 0.7)

Train <- datajoined %>% filter(spl == TRUE)
Test <- datajoined %>% filter(spl == FALSE)

table(Train$Useful)
table(Test$Useful)

# Function to compute accuracy of a classification model
tableAccuracy <- function(test, pred) {
  t = table(test, pred)
  a = sum(diag(t))/sum(t)
  return(a)
}

# Basic Random Forests:
DataRF = randomForest(Useful ~ ., data=Train)
PredictRF = predict(DataRF, newdata = Test)
table(Test$Useful, PredictRF)
tableAccuracy(Test$Useful, PredictRF)

# Random Forest with changing mtry:

DataRF2 <- train(Useful ~.,
                 data = Train,
                 method = "rf", # random forest
                 tuneGrid = data.frame(mtry=1:10),
                 trControl = trainControl(method="cv", number=5, verboseIter = TRUE),
                 metric = "Accuracy")

DataRF2.best <- DataRF2$finalModel
PredictRF2 = predict(DataRF2.best, newdata = Test)
table(Test$Useful, PredictRF2)
tableAccuracy(Test$Useful, PredictRF2)

# Cross-validated CART model
train.cart = train(Useful ~ .,
```

```
              data = Train,
              method = "rpart",
              tuneGrid = data.frame(cp=seq(0, 0.4, 0.002)),
              trControl = trainControl(method="cv", number=10))
train.cart
train.cart$results
train.cart$bestTune

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 = Test, type = "class")
table(Test$Useful, predict.cart)
tableAccuracy(Test$Useful, predict.cart)

# What about CART on training set?
PredictCARTTrain = predict(mod.cart, type = "class")
table(Train$Useful, PredictCARTTrain)
tableAccuracy(Train$Useful, PredictCARTTrain)

# Logistic Regression

DataLog = glm(Useful ~ ., data = Train, family = "binomial")
summary(DataLog)

# Predictions on test set
PredictLog = predict(DataLog, newdata = Test, type = "response")
table(Test$Useful, PredictLog > 0.5)
tableAccuracy(Test$Useful, PredictLog > 0.5)

# But what about training set?
PredictLogTrain = predict(DataLog, type = "response")
table(Train$Useful, PredictLogTrain > 0.5)
tableAccuracy(Train$Useful, PredictLogTrain > 0.5)

# Linear Discriminant Analysis
library(MASS)
lda.mod = lda(Useful ~ ., data = Train)

predict.lda = predict(lda.mod, newdata = Test)$class
table(Test$Useful, predict.lda)
tableAccuracy(Test$Useful, predict.lda)

### Bootstrap ###

library(boot)
```

```
TPR <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  contingency <- table(responses, predictions)
  TPR <- contingency[2, 2]/(contingency[2,2]+contingency[2,1])
  return(TPR)
}

FPR <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  contingency <- table(responses, predictions)
  FPR <- contingency[1, 2]/(contingency[1, 2] + contingency[1, 1])
  return(FPR)
}

Accuracy <- function(data, index) {
  responses <- data$response[index]
  predictions <- data$prediction[index]
  t = table(responses, predictions)
  a = sum(diag(t))/length(responses)
  return(a)
}

all_metrics <- function(data, index) {
  accuracy <- Accuracy(data, index)
  tpr <- TPR(data, index)
  fpr <- FPR(data, index)
  return(c(accuracy, tpr, fpr))
}

###### Basic Random Forest ######
RF_test_set = data.frame(response = Test$Useful, prediction = PredictRF)

# do bootstrap
RF_boot <- boot(RF_test_set, all_metrics, R = 10000)
RF_boot

# get confidence intervals (not manually)
boot.ci(RF_boot, index = 1, type = "basic")
boot.ci(RF_boot, index = 2, type = "basic")
boot.ci(RF_boot, index = 3, type = "basic")

###### Random Forest with changing mtry ######

RF_test_set2 = data.frame(response = Test$Useful, prediction = PredictRF2)

# do bootstrap
RF_boot2 <- boot(RF_test_set, all_metrics, R = 10000)
RF_boot2

# get confidence intervals (not manually)
```

```
boot.ci(RF_boot2, index = 1, type = "basic")
boot.ci(RF_boot2, index = 2, type = "basic")
boot.ci(RF_boot2, index = 3, type = "basic")

###### CART ######
CART_test_set = data.frame(response = Test$Useful, prediction = predict.cart)

# do bootstrap
CART_boot <- boot(CART_test_set, all_metrics, R = 10000)
CART_boot

# get confidence intervals
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 ######
Log_test_set = data.frame(response = Test$Useful, prediction = PredictLog)

# do bootstrap
Log_boot <- boot(Log_test_set, all_metrics, R = 10000)
Log_boot

# get confidence intervals
boot.ci(Log_boot, index = 1, type = "basic")
boot.ci(Log_boot, index = 2, type = "basic")
boot.ci(Log_boot, index = 3, type = "basic")

###### Linear Discriminant Analysis ######
LDA_test_set = data.frame(response = Test$Useful, prediction = predict.lda)

# do bootstrap
LDA_boot <- boot(LDA_test_set, all_metrics, R = 10000)
LDA_boot

# get confidence intervals
boot.ci(LDA_boot, index = 1, type = "basic")
boot.ci(LDA_boot, index = 2, type = "basic")
boot.ci(LDA_boot, index = 3, type = "basic")
```