Homework 2

options(scipen=999)

# Problem 1(a) - NB by hand

#Estimate the liklihood that the following e-mail was sent from Marine Le Pen or Emmanuel Macron.   
  
email <- c("immigration", "voter", "culture", "help", "neighborhood")  
  
lepen1 <- c("immigration", "women", "assimilate", "afraid", "win")  
lepen2 <- c("culture", "voter", "economy", "president", "help")  
macron1 <- c("voter", "women", "help", "reform", "education")  
macron2 <- c("union", "economy", "hope", "immigration", "neighborhood")  
macron3 <- c("win", "union", "europe", "elect", "president")  
lepen3 <- c("economy", "voter", "immigration", "president", "culture")  
macron4 <- c("success", "german", "culture", "help", "french")  
  
#calculate estimated probability of a given class from training data by dividing number of ocurrences of class by  
#total number of documents  
pr\_c\_lepen <- 3/7  
pr\_c\_macron <- 4/7  
  
#calculate estimated probability of a term given a class from training data by dividing number of term occurrences in   
#respective class (including multiple occurrences by total number of all terms in the training documents   
  
  
pr\_immigration\_lepen <- 2/15  
pr\_voter\_lepen <- 2/15  
pr\_culture\_lepen <- 2/15  
pr\_help\_lepen <- 1/15  
pr\_neighborhood\_lepen <- 0/15  
  
pr\_email\_lepen <- (2/15) \* (2/15) \* (2/15) \* (1/15) \* (0/15) \* (3/7)  
  
pr\_immigration\_macron <- 1/20  
pr\_voter\_macron <- 1/20  
pr\_culture\_macron <- 1/20  
pr\_help\_macron <- 2/20  
pr\_neighborhood\_macron <- 1/20  
  
pr\_email\_macron <- (1/20) \* (1/20) \* (1/20) \* (2/20) \* (1/20) \* (4/7)

# Problem 1(a) - answer - NB by hand

##### I don't completely trust these findings. Because LePen never uses the word "neighborhood" in the training documents, the value for the maximum likelihood estimation for this term is 0. This sparsity cancels out the other MLEs, making the class probability estimate 0, even if the MLEs for the other tems are high.

pr\_email\_lepen

## [1] 0

pr\_email\_macron

## [1] 0.0000003571429

# Problem 1(b) - NB by hand with LaPlace smoothing

#Perform LaPlace smoothing by adding 1 to each value   
  
#class probability does not change  
pr\_c\_lepen <- 3/7  
pr\_c\_macron <- 4/7  
  
#add one to each value, including total term count   
  
  
pr\_immigration\_lepen <- 3/30  
pr\_voter\_lepen <- 3/30  
pr\_culture\_lepen <- 3/30  
pr\_help\_lepen <- 2/30  
pr\_neighborhood\_lepen <- 1/30  
  
pr\_email\_lepen <- (3/30) \* (3/30) \* (3/30) \* (2/30) \* (1/30) \* (3/7)  
  
  
pr\_immigration\_macron <- 2/40  
pr\_voter\_macron <- 2/40  
pr\_culture\_macron <- 2/40  
pr\_help\_macron <- 3/40  
pr\_neighborhood\_macron <- 2/40  
  
pr\_email\_macron <- (2/40) \* (2/40) \* (2/40) \* (3/40) \* (2/40) \* (4/7)

# Problem 1(b) - answer - NB by hand with LaPlace smoothing

##### Because we added 1 to each term, thus blunting the effect of the sparse terms, LePen is now the more likely candidate to have written the e-mail.

pr\_email\_lepen

## [1] 0.000000952381

pr\_email\_macron

## [1] 0.0000002678571

# Problem 2(a) - creating positive and negative classifier

##### Preprocessing: Because we are simply creating a new variable, there is not much preprocessing of the text neccesary for this particular step.

amzn <- read.csv("amazon\_reviews.csv", stringsAsFactors = FALSE)  
names(amzn)[1] <- "ID"  
smed\_score <- median(amzn$Score)  
  
amzn$Class <- ifelse(amzn$Score <= 3, 0, 1)

# Problem 2(a) - answer - creating positive and negative classifier

##### Preprocessing: As we are actually dealing with the text, I will perform various preprocessing tasks. First, I will remove the encoding errors, as they will show up as words when tokenized. Then, I'll remove stopwords and punctuation, in addition to converting to lower case and stemming to account for for different tenses and spelling errors.

#Actual Score   
summary(amzn$Class)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.0000 0.0000 0.4993 1.0000 1.0000

# Problem 2(a) - creating anchor variables

amzn$Anchor <- ifelse(amzn$Score == 5 | amzn$Score == 1, 1, 0)  
amzn$Anchor\_Neg <- ifelse(amzn$Score ==1, -1, 0)  
  
for (i in 1:nrow(amzn)) {  
 if (amzn$Score[i] == 5) {  
 amzn$Anchor\_Combo[i] <- 1  
 }  
 else if (amzn$Score[i] == 1)  
 {  
 amzn$Anchor\_Combo[i] <- -1  
 }  
   
 else {  
 amzn$Anchor\_Combo[i] <- 0  
 }  
}

# Problem 2(a) - answer - creating anchor variables

#anchor  
  
summary(amzn$Anchor)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 0.0000 0.0000 0.4739 1.0000 1.0000

#anchor negative   
summary(amzn$Anchor\_Neg)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -1.0000 0.0000 0.0000 -0.1718 0.0000 0.0000

# Problem 2(b) - training NB to predict positive/negative reviews

#Read in sentiment dictionary by Hu & Liu   
library(quanteda)  
#read files and convert to vectors  
neg\_sent <- read.csv("negative-words.txt", sep=" ", header=FALSE, stringsAsFactors = FALSE)  
pos\_sent <- read.csv("positive-words.txt", sep=" ", header=FALSE, stringsAsFactors = FALSE)  
pos\_sent <- as.vector(pos\_sent$V1)  
neg\_sent <- as.vector(neg\_sent$V1)  
#create sentiment dictionary with positive and negative values   
sent\_dict <- dictionary(positive=pos\_sent, negative=neg\_sent)  
  
#remove escape characters and unicode  
amzn$Text <- gsub("[&#][1-9]\*", "", amzn$Text, fixed=TRUE)  
#create corpus with appropriate variables  
amzn.corp <- corpus(amzn$Text, docnames=amzn$ID, docvars=data.frame(class=amzn$Class, anchor\_combo = amzn$Anchor\_Combo, ID=amzn$ID))  
  
#create dfm with dictionary   
amzn.dfm <- dfm(tokenize(amzn.corp, removePunct=TRUE), dictionary=sent\_dict, stem=TRUE, tolower=TRUE)  
  
#convert to dfm and add postive and negative counts to df  
amzn.df <- as.data.frame(amzn.dfm)  
amzn <- cbind(amzn, amzn.df)  
  
#calculate sentiment score  
amzn$Sent\_Score <- amzn$posit - amzn$negat  
  
#create class for new sentiment score   
amzn$Sent\_Class <- ifelse(amzn$Sent\_Score >= 0, 1, 0)

# Problem 2(b) - answer - sentiment score

##### Pre-processing: The preprocessing for this step was similar to the previous question. I removed all non-alphanumeric characters and punctuation, while stemming words and converting them to lowercase. Again, I performed this step so to account for variations in tense and spelling.

#sentiment score   
summary(amzn$Sent\_Score)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## -26.000 1.000 2.000 3.602 5.000 115.000

# Problem 2(b) - answer - sentiment class

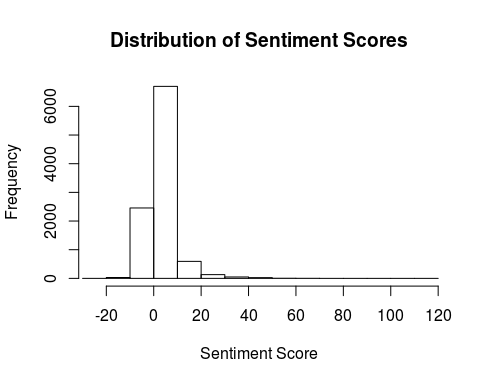
#sentiment class   
summary(amzn$Sent\_Class)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 0.0000 1.0000 1.0000 0.8452 1.0000 1.0000

#create histogram vizualizing distribution of positive versus negative words  
score\_min <- min(amzn$Sent\_Score)  
score\_max <- max(amzn$Sent\_Score)

# Problem 2(b) - answer - histogram

hist(amzn$Sent\_Score, breaks="Sturges", freq=TRUE, xlab="Sentiment Score", main="Distribution of Sentiment Scores", plot=TRUE, xlim=c(score\_min, score\_max))



# Problem 2(b) - creating confusion matrix and accuracy scores

#calculate percent positive   
pct\_pos <- (nrow(amzn[amzn$Sent\_Class == 1 ,]) / nrow(amzn)) \*100  
paste(pct\_pos, "% positive", sep="")

## [1] "84.52% positive"

#create confusion matrix  
confu\_mat <- table(amzn$Sent\_Class, amzn$Class)  
colnames(confu\_mat) <- c("New\_Score: Negative", "New\_Score: Positive")  
rownames(confu\_mat) <- c("Original\_Score: Negative", "Original\_Score: Positive")  
  
#calculate accuracy   
acc <- sum(confu\_mat[c(1, 4)])/sum(confu\_mat) \* 100  
  
#calculate precision   
prec <- round(sum(confu\_mat[4])/sum(confu\_mat[c(4, 3)]) \* 100, 2)  
  
#calculate recall   
recall <- round(sum(confu\_mat[4])/sum(confu\_mat[c(4, 2)]) \* 100, 2)

# Problem 2(b) - answer - creating confusion matrix and accuracy scores

confu\_mat

##   
## New\_Score: Negative New\_Score: Positive  
## Original\_Score: Negative 1291 257  
## Original\_Score: Positive 3716 4736

paste("Accuracy:", acc, "%", sep="")

## [1] "Accuracy:60.27%"

paste("Precision:", prec, "%", sep="")

## [1] "Precision:94.85%"

paste("Recall:", recall, "%", sep="")

## [1] "Recall:56.03%"

# Problem 2(b) - generate rank for scores and compute absolute difference

#rank by new score so that lower values come first  
amzn <- amzn[with(amzn, order(amzn$Sent\_Score, decreasing=TRUE)) ,]  
#add rank from 1 to 10,000  
amzn$Sent\_Rank <- seq(1:10000)  
  
#rank by original score so that lower values come first  
amzn <- amzn[with(amzn, order(amzn$Score, decreasing=TRUE)) ,]  
#add rank from 1 to 10,000  
amzn$Act\_Rank <- seq(1:10000)  
#compute abs difference between two ranks  
rank\_sum\_dict <- sum(abs(amzn$Sent\_Rank - amzn$Act\_Rank))

# Problem 2(b) - answer - generate rank for scores and compute absolute difference

rank\_sum\_dict

## [1] 23824160

# Problem 2(c) - creating NB classifier

##### Pre-processing: Preprocessing in this question was similar to the previous ones.

amzn.corp <- corpus(amzn$Text, docvars=data.frame(ID= amzn$ID, class=amzn$Class, score = amzn$Score, anchor=amzn$Anchor, anchor\_neg = amzn$Anchor\_Neg))  
#create training and test sets  
samp <- floor(0.20 \* nrow(amzn))  
set.seed(100)  
train\_set <- sample(seq\_len(nrow(amzn)), size=samp)  
train.df <- amzn[train\_set,]  
test.df <- amzn[-train\_set,]  
amzn$Split <- ifelse(amzn$ID %in% train.df$ID, "train", "test")  
  
#create test and training corpus   
corp.train <- corpus\_subset(amzn.corp, subset= amzn.corp$documents$ID %in% train.df$ID, select=c(class, ID))  
corp.test <- corpus\_subset(amzn.corp, subset= amzn.corp$documents$ID %in% test.df$ID, select=c(class, ID))  
  
#create dfm with training set and save class labels  
train.dfm <- dfm(tokenize(corp.train, removePunct=TRUE),dictionary=sent\_dict, groups=corp.train$documents$ID, stem=TRUE, tolower=TRUE)  
labelids <- as.data.frame(train.dfm@Dimnames$docs)  
names(labelids) <- "ID"  
library(plyr)  
labeldf <- join(labelids, train.df)  
labels <- labeldf$Class  
  
  
# train NB model with "uniforms" as prior   
nb.dict <- textmodel\_NB(x=train.dfm, y=labels, data=NULL,  
 smooth=1, prior="uniform")   
  
#create dfm for test to calculate accuracy, precision and recall   
test.dfm <- dfm(tokenize(corp.test, removePunct=TRUE),dictionary=sent\_dict, groups=corp.test$documents$ID, stem=TRUE, tolower=TRUE)  
test.pr1 <- predict(nb.dict, newdata=test.dfm)  
  
#convert to df and add original class labels   
pred.df <- as.data.frame(test.pr1[1:4])  
pred.df$ID <-as.integer(rownames(pred.df))  
pred.df <- merge(x=pred.df, y=test.df, by="ID")  
  
#create confusion matrix and add labels  
library(caret)  
confu\_mat2 <- confusionMatrix(pred.df$nb.predicted, pred.df$Class)  
confu\_mat2 <- confu\_mat2$table  
  
  
#calculate accuracy   
acc <- sum(confu\_mat2[c(1, 4)])/sum(confu\_mat2) \* 100  
  
#calculate precision   
prec <- round(sum(confu\_mat2[4])/sum(confu\_mat2[c(4, 3)]) \* 100, 2)  
  
#calculate recall   
recall <- round(sum(confu\_mat2[4])/sum(confu\_mat2[c(4, 2)]) \* 100, 2)

# Problem 2(c) - answer - NB uniform priors

confu\_mat2

## Reference  
## Prediction 0 1  
## 0 2593 1217  
## 1 1397 2793

paste("Accuracy:", acc, "%", sep="")

## [1] "Accuracy:67.325%"

paste("Precision:", prec, "%", sep="")

## [1] "Precision:69.65%"

paste("Recall:", recall, "%", sep="")

## [1] "Recall:66.66%"

# Problem 2(c) - NB with "docfreq""

amzn.corp <- corpus(amzn$Text, docvars=data.frame(ID= amzn$ID, class=amzn$Class, score = amzn$Score, anchor=amzn$Anchor, anchor\_neg = amzn$Anchor\_Neg))  
#create training and test sets  
samp <- floor(0.20 \* nrow(amzn))  
set.seed(100)  
train\_set <- sample(seq\_len(nrow(amzn)), size=samp)  
train.df <- amzn[train\_set,]  
test.df <- amzn[-train\_set,]  
amzn$Split <- ifelse(amzn$ID %in% train.df$ID, "train", "test")  
  
#create test and training corpus   
corp.train <- corpus\_subset(amzn.corp, subset= amzn.corp$documents$ID %in% train.df$ID, select=c(class, ID))  
corp.test <- corpus\_subset(amzn.corp, subset= amzn.corp$documents$ID %in% test.df$ID, select=c(class, ID))  
  
#create dfm with training set and save class labels  
train.dfm <- dfm(tokenize(corp.train, removePunct=TRUE),dictionary=sent\_dict, groups=corp.train$documents$ID, stem=TRUE, tolower=TRUE)  
labelids <- as.data.frame(train.dfm@Dimnames$docs)  
names(labelids) <- "ID"  
library(plyr)  
labeldf <- join(labelids, train.df)  
labels <- labeldf$Class  
  
  
  
  
# train NB model with "docfreq" as prior   
nb.dict <- textmodel\_NB(x=train.dfm, y=labels, data=NULL,  
 smooth=1, prior="docfreq")   
  
#create dfm for test to calculate accuracy, precision and recall   
test.dfm <- dfm(tokenize(corp.test, removePunct=TRUE),dictionary=sent\_dict, groups=corp.test$documents$ID, stem=TRUE, tolower=TRUE)  
test.pr1 <- predict(nb.dict, newdata=test.dfm)  
  
#convert to df and add original class labels   
pred.df <- as.data.frame(test.pr1[1:4])  
pred.df$ID <-as.integer(rownames(pred.df))  
pred.df <- merge(x=pred.df, y=test.df, by="ID")  
  
#create confusion matrix and add labels  
library(caret)  
confu\_mat2 <- confusionMatrix(pred.df$nb.predicted, pred.df$Class)  
confu\_mat2 <- confu\_mat2$table  
  
confu\_mat2

## Reference  
## Prediction 0 1  
## 0 2651 1245  
## 1 1339 2765

#calculate accuracy   
acc <- sum(confu\_mat2[c(1, 4)])/sum(confu\_mat2) \* 100  
  
#calculate precision   
prec <- round(sum(confu\_mat2[4])/sum(confu\_mat2[c(4, 3)]) \* 100, 2)  
  
#calculate recall   
recall <- round(sum(confu\_mat2[4])/sum(confu\_mat2[c(4, 2)]) \* 100, 2)

# Problem 2(c) - answer - NB witn "docfreq"

##### I would expect changing the priors from uniform to "docfreq" to decrease the accuracy of the model. Because using document frequency would imply the Bernoulli model, which ignores multiple ocurrences of a term in a document, I would expect that this would weaken the predicative power of the model, as it is one piece less of information. As long as we are confident in our class labels, looking at term ocurrence as a fraction of all words in a particular class (i.e. uniform) should provide enough information to take into account the varying frequency of terms across documents of the same class.

confu\_mat2

## Reference  
## Prediction 0 1  
## 0 2651 1245  
## 1 1339 2765

paste("Accuracy:", acc, "%", sep="")

## [1] "Accuracy:67.7%"

paste("Precision:", prec, "%", sep="")

## [1] "Precision:68.95%"

paste("Recall:", recall, "%", sep="")

## [1] "Recall:67.37%"

# Problem 2(c) - answer - removing smoothing from NB

##### Because the test set may contain terms that were not used in the training set, those documents will be given a zero probability unless you add some sort of smoothing to account for this sparsity.

# Problem 2(d) - wordscores model

##### Pre-processing: Preprocessing in this question was similar to the previous ones, however I did add smoothing to the DFM in order to score it against the wordscores model with smoothing.

#subset original df to just include extreme anchors  
amzn\_anch <- amzn[amzn$Anchor\_Combo %in% c(1, -1) ,]  
  
#create corpus   
anch.corp <- corpus(amzn\_anch$Text, docnames=amzn\_anch$ID, docvars=data.frame(class=amzn\_anch$Class, anchor\_combo = amzn\_anch$Anchor\_Combo, ID=amzn\_anch$ID))  
  
#create dfm   
pos\_neg\_dfm <- dfm(tokenize(anch.corp, removePunct=TRUE), stem=TRUE, tolower=TRUE, remove=stopwords("english"))  
  
#create vector with labels   
labels <- amzn\_anch$Anchor\_Combo  
  
#run wordscore model with smoothing factor of 1  
ws\_smooth<-textmodel(pos\_neg\_dfm,   
 y = labels,   
 model="wordscores", smooth=1)  
  
  
#create vector of terms and their respective weights  
feats<- sort(ws\_smooth@Sw, decreasing=TRUE)

# Problem 2(d) - answer - wordscores model

##### On the positive end of the spectrum, some of the most powerful words are "season", "love", "great" and "enjoy", while on the negative end, some of the more common words are "terrible", "get", "seem", "stupid".

#positive features  
head(feats, 10)

## season love great seri enjoy well show   
## 0.3955344 0.3850454 0.3761505 0.3524319 0.3224858 0.3102947 0.3078401   
## best excel episod   
## 0.3041227 0.3009014 0.2993644

#negative features  
tail(feats, 10)

## minut wast even bore film peopl like   
## 0.2207506 0.2174409 0.2166325 0.2163339 0.2130821 0.2118506 0.2103590   
## bad just movi   
## 0.1884533 0.1723957 0.1268244

# Problem 2(d) - create rank sum variable for wordscores

amzn.dfm\_smooth <- dfm\_smooth(amzn.dfm, smoothing=1)  
#create wordscore prediction model   
ws\_prediction <- predict(ws\_smooth, newdata = amzn.dfm\_smooth,  
 rescaling = "none", level = 0.95, verbose = TRUE)   
  
#extract desired features   
test <- as.data.frame(cbind(ws\_prediction@newdata@Dimnames$docs, ws\_prediction@textscores$textscore\_raw))  
names(test) <- c("ID", "Wordscore")  
  
#merge with original data   
amzn <- merge(x=amzn, y=test, by.x="ID", by.y="ID")  
amzn$Wordscore <- as.numeric(as.character(amzn$Wordscore))  
  
#create rank by wordscore   
amzn <- amzn[with(amzn, order(amzn$Wordscore, decreasing=TRUE)) ,]  
amzn$Word\_Rank <- seq(1:10000)  
  
#compute abs difference between two ranks  
rank\_sum\_ws <- sum(abs(amzn$Word\_Rank - amzn$Act\_Rank))

# Problem 2(d) - answer - rank sum

##### Based on the difference between the absolute values of the differences between actual and predicted scores, the dictionary method performed better than the wordscore method.

#Wordscores ranking  
summary(amzn$Word\_Rank)

## Min. 1st Qu. Median Mean 3rd Qu. Max.   
## 1 2501 5000 5000 7500 10000

#Absolute difference between wordscores and actual score   
rank\_sum\_ws

## [1] 20353900

# Problem 2(e) - creating SVM model

##### Pre-processing: Preprocessing involved removing stemwords and punctuation, as the other characters were already removed in a previous step.

library(NLP)  
library(tm)  
library(RTextTools)  
library(wordcloud)  
  
  
#create function to use for testing different training sizes   
splt\_data <- function(x, kerntype) {  
   
 #subset data to just first 1,000 entries for purposes of time   
 amzn\_svm <- amzn[1:1000, ]  
 #create weighted TfIdf matrix with training text   
 amzn\_svm.dtm <- create\_matrix(amzn\_svm$Text, language="english", stemWords = FALSE,  
 weighting = weightTfIdf, removePunctuation = FALSE)  
 #create container with document term matrix and appropriate classes for training data   
 container <- create\_container(amzn\_svm.dtm, t(amzn\_svm$Class), trainSize=1:x,  
 testSize=(x+1):nrow(amzn\_svm), virgin=FALSE)  
 #perform cross validation with 5 folds and the appropriate kernel type   
 cv.svm <- cross\_validate(container, nfold=5, algorithm = 'SVM', kernel = kerntype)  
   
 #return average accuracy   
 return (cv.svm$meanAccuracy)  
   
  
}  
  
  
#try with different ratios of test/training  
  
#10/90  
train\_size <- seq(1:9)  
train\_size <- as.data.frame(train\_size)  
kerntype <- 'linear'  
lin\_time <- system.time(for (i in 1:nrow(train\_size)) {  
 split <- train\_size$train\_size[i]\*100  
 try <- splt\_data(split, kerntype)  
 train\_size$acc\_linear[i] <- try })

## Fold 1 Out of Sample Accuracy = 0.8282828  
## Fold 2 Out of Sample Accuracy = 0.7864078  
## Fold 3 Out of Sample Accuracy = 0.8052632  
## Fold 4 Out of Sample Accuracy = 0.8037383  
## Fold 5 Out of Sample Accuracy = 0.8489583  
## Fold 1 Out of Sample Accuracy = 0.8190955  
## Fold 2 Out of Sample Accuracy = 0.8  
## Fold 3 Out of Sample Accuracy = 0.8136364  
## Fold 4 Out of Sample Accuracy = 0.8324324  
## Fold 5 Out of Sample Accuracy = 0.7959184  
## Fold 1 Out of Sample Accuracy = 0.8325581  
## Fold 2 Out of Sample Accuracy = 0.8349515  
## Fold 3 Out of Sample Accuracy = 0.8306011  
## Fold 4 Out of Sample Accuracy = 0.8029557  
## Fold 5 Out of Sample Accuracy = 0.8031088  
## Fold 1 Out of Sample Accuracy = 0.8031088  
## Fold 2 Out of Sample Accuracy = 0.8148148  
## Fold 3 Out of Sample Accuracy = 0.7956989  
## Fold 4 Out of Sample Accuracy = 0.8275862  
## Fold 5 Out of Sample Accuracy = 0.8217822  
## Fold 1 Out of Sample Accuracy = 0.8146341  
## Fold 2 Out of Sample Accuracy = 0.8457447  
## Fold 3 Out of Sample Accuracy = 0.8301887  
## Fold 4 Out of Sample Accuracy = 0.7875648  
## Fold 5 Out of Sample Accuracy = 0.7772277  
## Fold 1 Out of Sample Accuracy = 0.7868852  
## Fold 2 Out of Sample Accuracy = 0.784141  
## Fold 3 Out of Sample Accuracy = 0.815  
## Fold 4 Out of Sample Accuracy = 0.8217822  
## Fold 5 Out of Sample Accuracy = 0.8457447  
## Fold 1 Out of Sample Accuracy = 0.8316327  
## Fold 2 Out of Sample Accuracy = 0.8645833  
## Fold 3 Out of Sample Accuracy = 0.7840376  
## Fold 4 Out of Sample Accuracy = 0.8502415  
## Fold 5 Out of Sample Accuracy = 0.8229167  
## Fold 1 Out of Sample Accuracy = 0.8383838  
## Fold 2 Out of Sample Accuracy = 0.8  
## Fold 3 Out of Sample Accuracy = 0.8232323  
## Fold 4 Out of Sample Accuracy = 0.7819149  
## Fold 5 Out of Sample Accuracy = 0.8203883  
## Fold 1 Out of Sample Accuracy = 0.840796  
## Fold 2 Out of Sample Accuracy = 0.7927461  
## Fold 3 Out of Sample Accuracy = 0.8099548  
## Fold 4 Out of Sample Accuracy = 0.8078818  
## Fold 5 Out of Sample Accuracy = 0.8021978

kerntype <- 'radial'  
  
rad\_time <- system.time(for (i in 1:nrow(train\_size)) {  
 split <- train\_size$train\_size[i]\*100  
 try2 <- splt\_data(split, kerntype)  
 train\_size$acc\_radial[i] <- try2 })

## Fold 1 Out of Sample Accuracy = 0.84375  
## Fold 2 Out of Sample Accuracy = 0.7980769  
## Fold 3 Out of Sample Accuracy = 0.796875  
## Fold 4 Out of Sample Accuracy = 0.8341969  
## Fold 5 Out of Sample Accuracy = 0.8418605  
## Fold 1 Out of Sample Accuracy = 0.8088235  
## Fold 2 Out of Sample Accuracy = 0.8436019  
## Fold 3 Out of Sample Accuracy = 0.7857143  
## Fold 4 Out of Sample Accuracy = 0.8342246  
## Fold 5 Out of Sample Accuracy = 0.8457447  
## Fold 1 Out of Sample Accuracy = 0.8190955  
## Fold 2 Out of Sample Accuracy = 0.7926829  
## Fold 3 Out of Sample Accuracy = 0.8465608  
## Fold 4 Out of Sample Accuracy = 0.8595506  
## Fold 5 Out of Sample Accuracy = 0.8085106  
## Fold 1 Out of Sample Accuracy = 0.8591549  
## Fold 2 Out of Sample Accuracy = 0.800995  
## Fold 3 Out of Sample Accuracy = 0.8244681  
## Fold 4 Out of Sample Accuracy = 0.8483412  
## Fold 5 Out of Sample Accuracy = 0.7754011  
## Fold 1 Out of Sample Accuracy = 0.8186275  
## Fold 2 Out of Sample Accuracy = 0.8067633  
## Fold 3 Out of Sample Accuracy = 0.8484848  
## Fold 4 Out of Sample Accuracy = 0.8041237  
## Fold 5 Out of Sample Accuracy = 0.8375635  
## Fold 1 Out of Sample Accuracy = 0.8469388  
## Fold 2 Out of Sample Accuracy = 0.8309179  
## Fold 3 Out of Sample Accuracy = 0.8216216  
## Fold 4 Out of Sample Accuracy = 0.7971698  
## Fold 5 Out of Sample Accuracy = 0.82  
## Fold 1 Out of Sample Accuracy = 0.8241206  
## Fold 2 Out of Sample Accuracy = 0.785  
## Fold 3 Out of Sample Accuracy = 0.8686869  
## Fold 4 Out of Sample Accuracy = 0.8440367  
## Fold 5 Out of Sample Accuracy = 0.7891892  
## Fold 1 Out of Sample Accuracy = 0.8917526  
## Fold 2 Out of Sample Accuracy = 0.8351064  
## Fold 3 Out of Sample Accuracy = 0.7828054  
## Fold 4 Out of Sample Accuracy = 0.8041237  
## Fold 5 Out of Sample Accuracy = 0.8078818  
## Fold 1 Out of Sample Accuracy = 0.7832512  
## Fold 2 Out of Sample Accuracy = 0.8342857  
## Fold 3 Out of Sample Accuracy = 0.8457711  
## Fold 4 Out of Sample Accuracy = 0.8731707  
## Fold 5 Out of Sample Accuracy = 0.7824074

train\_size$train\_size <- train\_size$train\_size \* 100

# Problem 2(e) - answer - creating SVM model

##### An advantage of SVMs or Naive Bayes relative to the dictionary or wordscores approach is that you can be more specific with your classifications. Whereas the dictionary or wordscores approaches require you to dichotomize everything into "positive" and "negative", you miss picking up on language that might fall out of that spectrum, but is equally important to studty. With SVMs, you can train your model to also identify "middle of the road" examples and then study their use of language.

##### With respect to kernel selection, here I predicted that the best kernel would be of the linear sort. I made this decision based off the assumption that we have data that is linearly separable, with a high ratio of features to documents. Because of this, it is unnecessary to project the data into a three dimensions and could actually lead to overfitting. Using a radial kernel is also more expensive computationally, especially as the number of documents increases. My prediction was correct, as the linear kernel performed better and faster than the radial.

##### Here it appears that the optimium training size is around 50-70% of the original data set. When only given 10-20% of the data the model appears to underfit, and thus has poor predicative power because of high variane when applied to test folds. Between70-90%,we see problems at the other end of the spectrum - we have little bias (i.e. a complex model), but high variance, as the model does not generalize well too virgin data.

#Table with averages for linear and radial kernels   
train\_size

## train\_size acc\_linear acc\_radial  
## 1 100 0.8145301 0.8229519  
## 2 200 0.8122165 0.8236218  
## 3 300 0.8208350 0.8252801  
## 4 400 0.8125982 0.8216721  
## 5 500 0.8110720 0.8231125  
## 6 600 0.8107106 0.8233296  
## 7 700 0.8306824 0.8222067  
## 8 800 0.8127839 0.8243340  
## 9 900 0.8107153 0.8237772

#training times for reference   
lin\_time

## user system elapsed   
## 24.552 0.184 24.740

rad\_time

## user system elapsed   
## 23.380 0.212 23.593

# Question 3 - differences by nationality

#read in data  
  
hit <- read.csv("CF\_rate\_trustworthiness.csv")  
  
hit$demo\_group <- as.factor(gsub("[0-9]", "", hit$image\_name))  
  
hit$X\_country <- as.factor(hit$X\_country)  
#create ANOVA model for checking differences between nationalities   
model <- lm(rating ~ X\_country, data=hit)

# Question 3 - answer - differences by nationality

##### Is there any nationality that is likely to give statistically significant higher than average ratings? No. Because an effort was made to collect responses from a wide variety of nationalities, and the dataset is very small (250) compared to the number of countries that exist, it is very unlikely that the difference in average rating by nationality will be statistically significant. As you can see, considering an alpha of .01, this holds true.

model

##   
## Call:  
## lm(formula = rating ~ X\_country, data = hit)  
##   
## Coefficients:  
## (Intercept) X\_countryARG X\_countryAUT   
## 7.00000000000003908 -1.33333333333337989 0.19999999999995297   
## X\_countryBGD X\_countryBGR X\_countryBIH   
## -4.33333333333337833 1.59999999999996523 -2.58333333333337123   
## X\_countryCAN X\_countryCHL X\_countryDEU   
## -0.25000000000003192 2.99999999999996136 -0.00000000000003415   
## X\_countryEGY X\_countryESP X\_countryFIN   
## -2.71428571428575482 -1.22222222222226318 -2.00000000000003375   
## X\_countryGBR X\_countryGRC X\_countryHKG   
## 0.71428571428567433 -2.40000000000004166 -0.00000000000004074   
## X\_countryHRV X\_countryIDN X\_countryIND   
## -4.00000000000003730 -3.00000000000004219 -0.77777777777781809   
## X\_countryITA X\_countryJOR X\_countryJPN   
## -0.00000000000003945 -0.42857142857146957 1.16666666666662322   
## X\_countryMEX X\_countryMYS X\_countryPAK   
## -1.66666666666670715 -2.00000000000004041 -2.00000000000004041   
## X\_countryPHL X\_countryPOL X\_countryPRT   
## 1.99999999999996181 -3.00000000000004086 -0.66666666666671370   
## X\_countryROU X\_countryRUS X\_countrySRB   
## -0.00000000000003460 0.49999999999995920 -1.56250000000004152   
## X\_countrySVK X\_countryTUR X\_countryUKR   
## -3.00000000000003997 -0.11111111111114989 -1.00000000000004086   
## X\_countryUSA X\_countryVEN   
## -0.40000000000003616 -1.08333333333337478

summary(model)

##   
## Call:  
## lm(formula = rating ~ X\_country, data = hit)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -4.438 -1.175 0.000 1.090 4.562   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)  
## (Intercept) 7.00000000000003908 1.90075539861684506 3.683 0.000303  
## X\_countryARG -1.33333333333337989 2.19480328211014397 -0.607 0.544267  
## X\_countryAUT 0.19999999999995297 2.08217321624434470 0.096 0.923582  
## X\_countryBGD -4.33333333333337833 2.19480328211014086 -1.974 0.049829  
## X\_countryBGR 1.59999999999996523 2.08217321624433227 0.768 0.443213  
## X\_countryBIH -2.58333333333337123 1.97836894330118551 -1.306 0.193246  
## X\_countryCAN -0.25000000000003192 2.12510913995349027 -0.118 0.906480  
## X\_countryCHL 2.99999999999996136 2.32794042622585673 1.289 0.199114  
## X\_countryDEU -0.00000000000003415 2.08217321624433316 0.000 1.000000  
## X\_countryEGY -2.71428571428575482 2.03199299362433194 -1.336 0.183264  
## X\_countryESP -1.22222222222226318 1.95284053257543055 -0.626 0.532172  
## X\_countryFIN -2.00000000000003375 2.68807406347782241 -0.744 0.457804  
## X\_countryGBR 0.71428571428567433 2.03199299362433372 0.352 0.725598  
## X\_countryGRC -2.40000000000004166 2.08217321624434248 -1.153 0.250545  
## X\_countryHKG -0.00000000000004074 2.01605554760836547 0.000 1.000000  
## X\_countryHRV -4.00000000000003730 2.68807406347781530 -1.488 0.138439  
## X\_countryIDN -3.00000000000004219 2.32794042622585895 -1.289 0.199114  
## X\_countryIND -0.77777777777781809 1.92697395299476670 -0.404 0.686954  
## X\_countryITA -0.00000000000003945 2.68807406347782107 0.000 1.000000  
## X\_countryJOR -0.42857142857146963 2.03199299362433417 -0.211 0.833188  
## X\_countryJPN 1.16666666666662322 2.05305047825577436 0.568 0.570547  
## X\_countryMEX -1.66666666666670715 2.05305047825577391 -0.812 0.417950  
## X\_countryMYS -2.00000000000004041 2.19480328211013997 -0.911 0.363353  
## X\_countryPAK -2.00000000000004041 2.19480328211014175 -0.911 0.363353  
## X\_countryPHL 1.99999999999996181 2.19480328211013997 0.911 0.363353  
## X\_countryPOL -3.00000000000004086 1.99352908027653064 -1.505 0.134063  
## X\_countryPRT -0.66666666666671370 2.19480328211013820 -0.304 0.761661  
## X\_countryROU -0.00000000000003460 2.68807406347782063 0.000 1.000000  
## X\_countryRUS 0.49999999999995920 2.32794042622585762 0.215 0.830174  
## X\_countrySRB -1.56250000000004152 1.95925381924006281 -0.797 0.426185  
## X\_countrySVK -3.00000000000003997 2.32794042622585540 -1.289 0.199114  
## X\_countryTUR -0.11111111111114989 2.00357211149683279 -0.055 0.955835  
## X\_countryUKR -1.00000000000004086 2.05305047825577436 -0.487 0.626778  
## X\_countryUSA -0.40000000000003616 2.08217321624433360 -0.192 0.847869  
## X\_countryVEN -1.08333333333337478 1.97836894330119217 -0.548 0.584634  
##   
## (Intercept) \*\*\*  
## X\_countryARG   
## X\_countryAUT   
## X\_countryBGD \*   
## X\_countryBGR   
## X\_countryBIH   
## X\_countryCAN   
## X\_countryCHL   
## X\_countryDEU   
## X\_countryEGY   
## X\_countryESP   
## X\_countryFIN   
## X\_countryGBR   
## X\_countryGRC   
## X\_countryHKG   
## X\_countryHRV   
## X\_countryIDN   
## X\_countryIND   
## X\_countryITA   
## X\_countryJOR   
## X\_countryJPN   
## X\_countryMEX   
## X\_countryMYS   
## X\_countryPAK   
## X\_countryPHL   
## X\_countryPOL   
## X\_countryPRT   
## X\_countryROU   
## X\_countryRUS   
## X\_countrySRB   
## X\_countrySVK   
## X\_countryTUR   
## X\_countryUKR   
## X\_countryUSA   
## X\_countryVEN   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 1.901 on 185 degrees of freedom  
## Multiple R-squared: 0.3498, Adjusted R-squared: 0.2303   
## F-statistic: 2.927 on 34 and 185 DF, p-value: 0.000002036

# Question 3 - differences by demographic

#create ANOVA model for checking differences between groups  
model <- lm(rating ~ demo\_group, data=hit)  
demo\_anova <- anova(model)

# Question 3 - answer - differences by demographic

##### Yes, an analysis of variance shows that there are statistically significant differences in the average ratings of each demographic group.

demo\_anova

## Analysis of Variance Table  
##   
## Response: rating  
## Df Sum Sq Mean Sq F value Pr(>F)   
## demo\_group 2 34.57 17.2842 3.7755 0.02444 \*  
## Residuals 217 993.41 4.5779   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

summary(demo\_anova)

## Df Sum Sq Mean Sq F value   
## Min. : 2.00 Min. : 34.57 Min. : 4.578 Min. :3.776   
## 1st Qu.: 55.75 1st Qu.:274.28 1st Qu.: 7.755 1st Qu.:3.776   
## Median :109.50 Median :513.99 Median :10.931 Median :3.776   
## Mean :109.50 Mean :513.99 Mean :10.931 Mean :3.776   
## 3rd Qu.:163.25 3rd Qu.:753.70 3rd Qu.:14.108 3rd Qu.:3.776   
## Max. :217.00 Max. :993.41 Max. :17.284 Max. :3.776   
## NA's :1   
## Pr(>F)   
## Min. :0.02444   
## 1st Qu.:0.02444   
## Median :0.02444   
## Mean :0.02444   
## 3rd Qu.:0.02444   
## Max. :0.02444   
## NA's :1

# Question 3 - differences by gender

#create ANOVA model for just women and men   
  
hit$gender <- gsub("black", "", hit$demo\_group)  
hit$gender <- gsub("white", "", hit$gender)  
  
  
model\_gen <- lm(rating ~ gender, data=hit)  
gen\_anova <- anova(model\_gen)

# Question 3 - answer - differences by gender

##### There is only a statistical difference if you choose an alpha of > .05, which I would not in this case, because of the sensitive nature of the research.

summary(gen\_anova)

## Df Sum Sq Mean Sq F value   
## Min. : 1.00 Min. : 15.1 Min. : 4.646 Min. :3.249   
## 1st Qu.: 55.25 1st Qu.: 264.5 1st Qu.: 7.259 1st Qu.:3.249   
## Median :109.50 Median : 514.0 Median : 9.872 Median :3.249   
## Mean :109.50 Mean : 514.0 Mean : 9.872 Mean :3.249   
## 3rd Qu.:163.75 3rd Qu.: 763.4 3rd Qu.:12.484 3rd Qu.:3.249   
## Max. :218.00 Max. :1012.9 Max. :15.097 Max. :3.249   
## NA's :1   
## Pr(>F)   
## Min. :0.07283   
## 1st Qu.:0.07283   
## Median :0.07283   
## Mean :0.07283   
## 3rd Qu.:0.07283   
## Max. :0.07283   
## NA's :1

gen\_anova

## Analysis of Variance Table  
##   
## Response: rating  
## Df Sum Sq Mean Sq F value Pr(>F)   
## gender 1 15.1 15.0972 3.2493 0.07283 .  
## Residuals 218 1012.9 4.6463   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1

# Question 4 - answer - questions to CrowdFlower raters

##### I would probably ask them to specify their race - as it would be helpful to know if there is some effect of "rater bias" as far as the race of the rater is concerned. With that information, we might be able to control for that issue. I would also ask them to specify their trust of politicians as a whole. In the case that the sample of politician race is not equally represented, it would be helpful to know what proportion of trustworthiness corresponds to "race" and what proportion is due to that rater's inherent distrust of politicians in general.