Text As Data HW 2

Jean An (cya220)

4/11/2022

```
library(tidyverse)
library(quanteda)
library(quanteda.textmodels)
library(readtext)
library(caret)
library(randomForest)
```

Q1

```
emails <- data.frame(sender = c("Perdue","Perdue","Ossoff","Ossoff",
                                "Ossoff","Perdue","Ossoff"),
                     content = c("immigration aliens criminals country loophole",
                                 "voter economy tax jobs security",
                                 "immigration country diversity help security",
                                 "healthcare affordable expansion unfair help",
                                 "voter relief debt jobs help",
                                 "healthcare cost socialism unfair help",
                                 "abortion choice right women help court"))
mystery <- "healthcare voter tax help jobs"
```

(a)

```
by_sender <- emails %>%
  group_by(sender) %>%
  summarize(words = paste(content, collapse = " "),
            email_ratio = n()/nrow(emails),
            total_words = str_count(words," ")+1,
            healthcare = str_count(words,"healthcare")/total_words,
            voter = str_count(words,"voter")/total_words,
            tax = str_count(words,"tax")/total_words,
            help = str_count(words,"help")/total_words,
            jobs = str_count(words,"jobs")/total_words,
            prediction = healthcare*voter*tax*help*jobs*email_ratio)

by_sender %>% select(sender, prediction)
```

```
## # A tibble: 2 x 2
##   sender  prediction
```

```
##   <chr>        <dbl>
## 1 Ossoff 0
## 2 Perdue 0.000000564
```

Based on the results, I would predict that this email was sent by David Perdue of the Republican Party. However, I don't fully trust this finding because Jon Ossoff of the Democrat Party has a predicted value of zero simply because he never said the word "tax." As a result, no matter how many times he says the other four words, his predicted value will always be zero.

(b)

```
all_words <- paste(emails$content, collapse = " ")
unique_words <- length(dfm(tokens(all_words)))
by_sender_sm <- emails %>%
  group_by(sender) %>%
  summarize(words = paste(content, collapse = " "),
            email_ratio = n()/nrow(emails),
            total_words = str_count(words," ")+1+unique_words,
            healthcare = (str_count(words,"healthcare")+1)/total_words,
            voter = (str_count(words,"voter")+1)/total_words,
            tax = (str_count(words,"tax")+1)/total_words,
            help = (str_count(words,"help")+1)/total_words,
            jobs = (str_count(words,"jobs")+1)/total_words,
            prediction = healthcare*voter*tax*help*jobs*email_ratio)

by_sender_sm %>% select(sender, prediction)
```

```
## # A tibble: 2 x 2
##   sender  prediction
##   <chr>        <dbl>
## 1 Ossoff 0.000000111
## 2 Perdue 0.000000134
```

Based on these new results, I would still predict that this email was sent by David Perdue of the Republican Party, because he has a slightly higher predicted value. However, we can see that the predicted values between the two of them are very similar. Applying the Laplace smoothing makes sense because it better adjusts for the slight differences in the candidate's shared language when the sample size is very small.

Q2

```
tripadvisor <- read_csv("tripadvisor.csv",show_col_types = FALSE)
```

(a)

```
tripadvisor <- tripadvisor %>%
  mutate(class = ifelse(stars >= median(tripadvisor$stars),"positive","negative"))
tripadvisor %>%
  group_by(class) %>%
  summarize(proportion = n()/nrow(tripadvisor))
```

```
## # A tibble: 2 x 2
##   class    proportion
##   <chr>         <dbl>
## 1 negative      0.263
## 2 positive      0.737
```

```
median(tripadvisor$stars)
```

```
## [1] 4
```

(b)

```
tripadvisor <- tripadvisor %>%
  mutate(anchor = ifelse(stars == 5,"positive",
                         ifelse(stars <= 2,"negative","neutral")))
tripadvisor %>%
  group_by(anchor) %>%
  summarize(proportion = n()/nrow(tripadvisor))
```

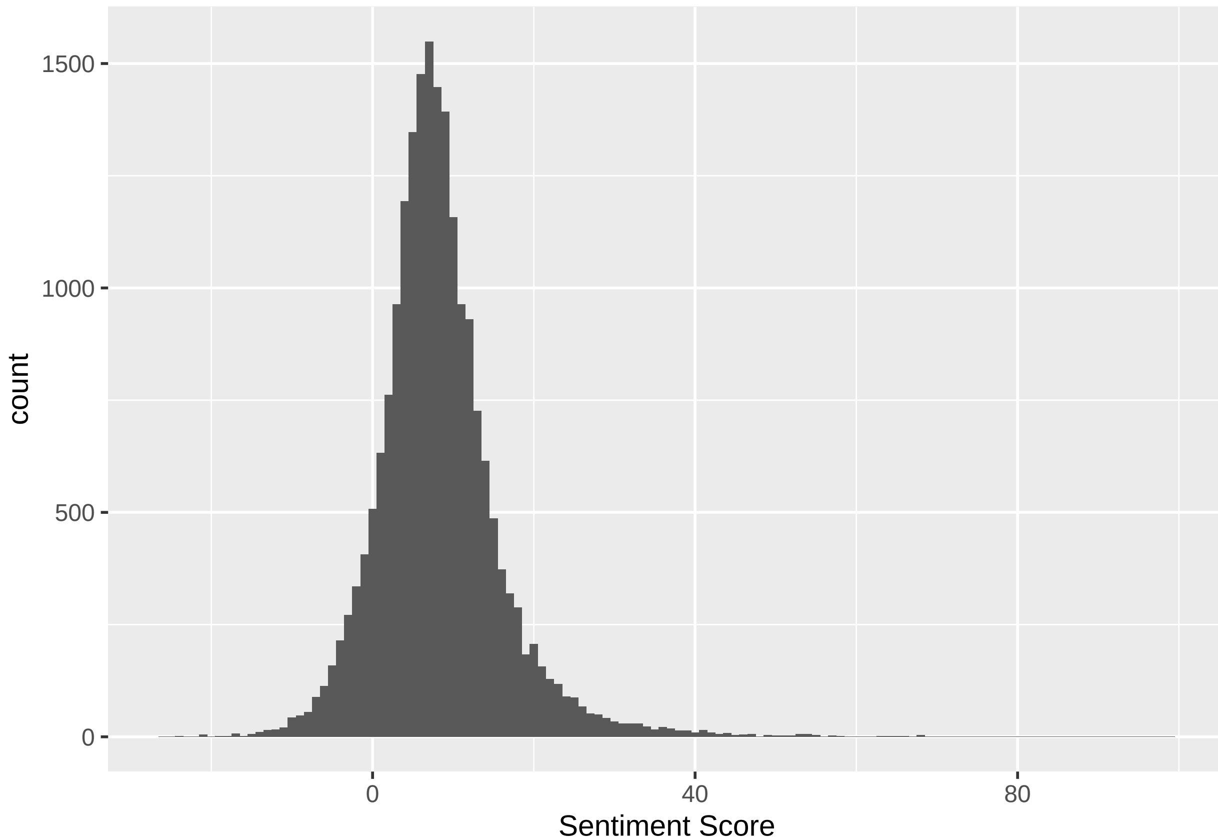
```
## # A tibble: 3 x 2
##   anchor   proportion
##   <chr>         <dbl>
## 1 negative      0.157
## 2 neutral       0.401
## 3 positive      0.442
```

Q3

```
pos_words <- c(read.table("positive-words.txt", header=FALSE)$V1)
neg_words <- c(read.table("negative-words.txt", header=FALSE)$V1)
```

(a) In the preprocessing, I chose to remove all apostrophes and all punctuation in the text. I chose not to remove stopwords or stem the words because I believe this better helps us identify the exact match of positive and negative words.

```
tripadvisor$text <- gsub(pattern = "'", "", tripadvisor$text)
pos_dfm <- convert(dfm(tripadvisor$text, select = pos_words, remove_punct = TRUE),to="data.frame")
tripadvisor$pos_count <- rowSums(pos_dfm[,c(-1)])
neg_dfm <- convert(dfm(tripadvisor$text, select = neg_words, remove_punct = TRUE),to="data.frame")
tripadvisor$neg_count <- rowSums(neg_dfm[,c(-1)])
tripadvisor <- tripadvisor %>%
  mutate(sentiment = pos_count - neg_count)
ggplot(tripadvisor, aes(x=sentiment)) +
  geom_histogram(binwidth = 1) +
  xlab("Sentiment Score")
```



(b)

```
tripadvisor <- tripadvisor %>%
  mutate(dichotomous = ifelse(sentiment > 0,"positive","negative"))
tripadvisor %>%
  group_by(dichotomous) %>%
  summarize(n()/nrow(tripadvisor))
```

```
## # A tibble: 2 x 2
##   dichotomous `n()/nrow(tripadvisor)`
##   <chr>                         <dbl>
## 1 negative                      0.114
## 2 positive                      0.886
```

Based on the Sentiment Score results, 88.6% of reviews are classified as positive. This is much higher than the "true class" classification we reported in Question 2, part (a).

(c)

```
cmat <- table(tripadvisor$dichotomous,tripadvisor$class)
cmat
```

```
##
##            negative positive
##   negative     2071      263
##   positive     3327    14830
```

```r
accuracy <- sum(diag(cmat))/sum(cmat)
accuracy
```

```
## [1] 0.8248011
```

```r
precision <- cmat[2,2]/sum(cmat[,2])
precision
```

```
## [1] 0.9825747
```

```r
recall <- cmat[2,2]/sum(cmat[2,])
recall
```

```
## [1] 0.8167649
```

```r
specificity <- cmat[1,1]/sum(cmat[1,])
specificity
```

```
## [1] 0.8873179
```

```r
f1_score <- 2*precision*recall/(precision+recall)
f1_score
```

```
## [1] 0.8920301
```

I think the accuracy (82.5%) of the classifier is somewhat disappointing. With 88.6% of the reviews classified as positive, I can simply predict all entries to be positive and beat that accuracy score.

Q4

(a) In addition to removing apostrophes and punctuation (which were already performed previously), I also stemmed the words and removed stopwords for this Naive Bayes classifier.

(b)

```r
set.seed(42)
prop_train <- 0.2
ids <- 1:nrow(tripadvisor)
ids_train <- sample(ids, ceiling(prop_train*length(ids)), replace = FALSE)
train_set <- tripadvisor[ids_train,]
test_set <- tripadvisor[-ids_train,]

train_dfm <- dfm(train_set$text, stem = TRUE, remove_punct = TRUE, remove = stopwords("english"))
test_dfm <- dfm(test_set$text, stem = TRUE, remove_punct = TRUE, remove = stopwords("english"))
test_dfm <- dfm_match(test_dfm, features = featnames(train_dfm))

nb_model_sm <- textmodel_nb(train_dfm, train_set$class, smooth = 1, prior = "uniform")
predicted_class_sm <- predict(nb_model_sm, newdata = test_dfm, force=TRUE)

cmat_sm <- table(test_set$class, predicted_class_sm)
cmat_sm
```

```
##           predicted_class_sm
##            negative positive
##   negative     2660     1608
##   positive      423    11701
```

```r
nb_acc_sm <- sum(diag(cmat_sm))/sum(cmat_sm)
nb_acc_sm
```

```
## [1] 0.8760981
```

```r
nb_precision_sm <- cmat_sm[2,2]/sum(cmat_sm[,2])
nb_precision_sm
```

```
## [1] 0.8791795
```

```r
nb_recall_sm <- cmat_sm[2,2]/sum(cmat_sm[2,])
nb_recall_sm
```

```
## [1] 0.9651105
```

```r
nb_f1_sm <- 2*(nb_recall_sm*nb_precision_sm)/(nb_recall_sm + nb_precision_sm)
nb_f1_sm
```

```
## [1] 0.9201431
```

(c)

```r
nb_model_sm_2 <- textmodel_nb(train_dfm, train_set$class, smooth = 1, prior = "docfreq")
predicted_class_2 <- predict(nb_model_sm_2, newdata = test_dfm, force=TRUE)

cmat_sm_2 <- table(test_set$class, predicted_class_2)
cmat_sm_2
```

```
##           predicted_class_2
##            negative positive
##   negative     2533     1735
##   positive      342    11782
```

```r
nb_acc_sm_2 <- sum(diag(cmat_sm_2))/sum(cmat_sm_2)
nb_acc_sm_2
```

```
## [1] 0.8732918
```

```r
nb_precision_sm_2 <- cmat_sm_2[2,2]/sum(cmat_sm_2[,2])
nb_precision_sm_2
```

```
## [1] 0.8716431
```

```r
nb_recall_sm_2 <- cmat_sm_2[2,2]/sum(cmat_sm_2[2,])
nb_recall_sm_2
```

```
## [1] 0.9717915
```

```r
nb_f1_sm_2 <- 2*(nb_recall_sm_2*nb_precision_sm_2)/(nb_recall_sm_2 + nb_precision_sm_2)
nb_f1_sm_2
```

```
## [1] 0.9189969
```

I expected the change in prior to lead to a change in the performance of the Naive Bayes predictions, because using document frequency as the prior is essentially using the training set to provide information. However, it turns out that the accuracy is basically the same as part(b), where a uniform prior was used.

(d)

```r
nb_model <- textmodel_nb(train_dfm, train_set$class, smooth = 0, prior = "uniform")
predicted_class <- predict(nb_model, newdata = test_dfm, force=TRUE)

cmat <- table(test_set$class, predicted_class)
cmat
```

```
##           predicted_class
##            negative positive
##   negative     2706     1562
##   positive     3741     8383
```

```r
nb_acc <- sum(diag(cmat))/sum(cmat)
nb_acc
```

```
## [1] 0.6764885
```

```r
nb_precision <- cmat[2,2]/sum(cmat[,2])
nb_precision
```

```
## [1] 0.8429361
```

```r
nb_recall <- cmat[2,2]/sum(cmat[2,])
nb_recall
```

```
## [1] 0.6914385
```

```r
nb_f1 <- 2*(nb_recall*nb_precision)/(nb_recall + nb_precision)
nb_f1
```

```
## [1] 0.7597082
```

Without smoothing, the model performed very poorly, with an accuracy of less than 68%. I believe this is because of something similar to the the missing value issue that we had in question 1, where the lack of smoothing returned zeroes.

(e) I'm not sure if this will work computationally, but I suppose emojis might be a possible feature to help classify the sentiment of a document.

Q5

```
filenames <- list.files(path = "manifestos")
party <- unlist(regmatches(unlist(filenames), gregexpr("^[[:alpha:]]{3}", unlist(filenames))))
year <- unlist(regmatches(unlist(filenames), gregexpr("[[:digit:]]+", unlist(filenames))))
cons_labour_manifestos <- corpus(readtext("manifestos/*.txt"))
docvars(cons_labour_manifestos, field = c("party", "year")) <- data.frame(cbind(party, year))
cons_labour_df <- tibble(text = as.character(cons_labour_manifestos),
                         class = party,year = as.integer(year))
```

(a)

```
anchor_labor <- cons_labour_df %>% filter(class == 'Lab', year == 1945)
anchor_cons <- cons_labour_df %>% filter(class == 'Con', year == 1983)
train_set <- rbind(anchor_labor, anchor_cons)
train_dfm <- dfm(train_set$text, remove_punct = TRUE, remove = stopwords("english"))

ws_base <- textmodel_wordscores(train_dfm, y = (2*as.numeric(train_set$class == "Lab"))-1)
head(sort(ws_base$wordscores, decreasing = TRUE),10)
```

```
##          1945           let   declaration consideration       victory
##             1             1             1             1             1
##          east          goes      japanese     barbarism      defeated
##             1             1             1             1             1
```

```
head(sort(ws_base$wordscores, decreasing = FALSE),10)
```

```
##         1983      foreward     challenge          four    recovered    confidence
##           -1            -1            -1            -1           -1            -1
## self-respect      regained    admiration          seen
##           -1            -1            -1            -1
```

(b)

```
test_labor <- cons_labour_df %>% filter(class == 'Lab', year != 1945)
test_cons <- cons_labour_df %>% filter(class == 'Con', year != 1983)
test_set <- rbind(test_labor, test_cons)
test_dfm <- dfm(test_set$text, remove_punct = TRUE, remove = stopwords("english"))
```

```
predict(ws_base, newdata = test_dfm, rescaling = "none", level = 0.95)
```

```
## Warning: 849 features in newdata not used in prediction.

## Lab1951.txt Con1979.txt
## -0.04107512 -0.27281496
```

If I am understanding the predictions correctly, the model was able to successfully predict the Conservative text (negative predicted value) but was incorrect in predicting the Labor text (also negative predicted value, should be positive). The warning message suggests that 849 of the words used in the testing set were not present in the training set, hence were not included in the model. This could be problematic as these excluded words could be highly suggestive for the nature of the text being either from the Conservative Party or the Labor Party.

(c)

```
predict(ws_base, newdata = test_dfm, rescaling = "lbg", level = 0.95)
```

```
## Lab1951.txt Con1979.txt
##    0.843055   -1.156945
```

With the standard LBG rescaling, the resulting predictions are correct. The predicted value for the Labor text has a positive sign, and the predicted value for the Conservative text has a negative sign.

Q6

```
trip_samp <- tripadvisor[1:1000,]
```

(a) In addition to removing apostrophes and punctuation (which were already performed previously), I also stemmed the words and removed stopwords for this SVM classifier. An advantage offered by SVM or Naive Bayes relative to the dictionary approach or wordscores is that it does not require previously built dictionaries or lists of pre-classified words. It simply uses the data to train itself.

(b)

```
tokenized <- tokens(trip_samp$text, remove_punct = TRUE)
no_stop <- tokens_remove(tokenized, pattern = stopwords("en"))
stemmed <- tokens_wordstem(no_stop)
trip_dfm <- dfm(stemmed) %>% convert("matrix")

results <- data.frame(train_size = NA, accuracy = NA)
for (i in 1:9) {

  ids_train <- createDataPartition(1:nrow(trip_dfm), p = 0.1*i, list = FALSE, times = 1)
  train_x <- trip_dfm[ids_train, ] %>% as.data.frame()
  train_y <- trip_samp$class[ids_train] %>% as.factor()
  val_x <- trip_dfm[-ids_train, ] %>% as.data.frame()
  val_y <- trip_samp$class[-ids_train] %>% as.factor()

  trctrl <- trainControl(method = "cv", number = 5)
  svm_mod_linear <- train(x = train_x, y = train_y,
                          method = "svmLinear", trControl = trctrl)
  svm_linear_pred <- predict(svm_mod_linear, newdata = val_x)
  svm_linear_cmat <- confusionMatrix(svm_linear_pred, val_y)
  results[i,] <- c(0.1*i, svm_linear_cmat$overall[1])

}

results %>% filter(accuracy == max(accuracy))
```

```
##   train_size accuracy
## 1        0.8    0.835
```

In terms of accuracy, I think the model did pretty well. As a reminder, the "true class" classification on the entire data set had about 73.7% of the reviews classified as positive. Even though this is only a subset that takes the first 1,000 reviews, the accuracy was still almost 10% higher than the baseline (had one simply predicted all to be positive).

(c)

```r
trip_samp <- tripadvisor[1:100,]
tokenized <- tokens(trip_samp$text, remove_punct = TRUE)
no_stop <- tokens_remove(tokenized, pattern = stopwords("en"))
stemmed <- tokens_wordstem(no_stop)
trip_dfm <- dfm(stemmed) %>% convert("matrix")

ids_train <- createDataPartition(1:nrow(trip_dfm), p = 0.8, list = FALSE, times = 1)
train_x <- trip_dfm[ids_train, ] %>% as.data.frame()
train_y <- trip_samp$class[ids_train] %>% as.factor()
val_x <- trip_dfm[-ids_train, ] %>% as.data.frame()
val_y <- trip_samp$class[-ids_train] %>% as.factor()

trctrl <- trainControl(method = "cv", number = 5)
svm_mod_logit <- train(x = train_x, y = train_y, method = "glm",
                       trControl = trctrl, family = 'binomial')

svm_logit_pred <- predict(svm_mod_logit, newdata = val_x)
svm_logit_cmat <- confusionMatrix(svm_logit_pred, val_y)
svm_logit_cmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative        4        9
##   positive        2        5
##
##                Accuracy : 0.45
##                  95% CI : (0.2306, 0.6847)
##     No Information Rate : 0.7
##     P-Value [Acc > NIR] : 0.99486
##
##                   Kappa : 0.0179
##
##  Mcnemar's Test P-Value : 0.07044
##
##             Sensitivity : 0.6667
##             Specificity : 0.3571
##          Pos Pred Value : 0.3077
##          Neg Pred Value : 0.7143
##              Prevalence : 0.3000
##          Detection Rate : 0.2000
##    Detection Prevalence : 0.6500
##       Balanced Accuracy : 0.5119
##
##        'Positive' Class : negative
##
```

Q7

```
trip_samp <- tripadvisor[1:500,]
```

(a)

```
tokenized <- tokens(trip_samp$text, remove_punct = TRUE)
no_stop <- tokens_remove(tokenized, pattern = stopwords("en"))
stemmed <- tokens_wordstem(no_stop)
trip_dfm <- dfm(stemmed) %>% convert("matrix")

ids_train <- createDataPartition(1:nrow(trip_dfm), p = 0.8, list = FALSE, times = 1)
train_x <- trip_dfm[ids_train, ] %>% as.data.frame()
train_y <- trip_samp$class[ids_train] %>% as.factor()
test_x <- trip_dfm[-ids_train, ] %>% as.data.frame()
test_y <- trip_samp$class[-ids_train] %>% as.factor()
```

(b)

```
rf.base <- randomForest(x = train_x, y = train_y, importance = TRUE)
token_importance <- round(importance(rf.base, 2), 2)
head(rownames(token_importance)[order(-token_importance)],10)
```

```
##  [1] "great"   "room"    "night"   "love"    "manag"   "comfort" "terribl"
##  [8] "staff"   "old"     "dirti"
```

(c)

```
predict_test <- predict(rf.base, newdata = test_x)
cmat <- confusionMatrix(data = predict_test, reference = test_y)
cmat$table
```

```
##           Reference
## Prediction negative positive
##   negative       21        3
##   positive       19       57
```

```
cmat$overall[1]
```

```
## Accuracy
##     0.78
```

```
precision <- cmat$byClass[5]
precision
```

```
## Precision
##     0.875
```

```r
recall <- cmat$byClass[6]
recall
```

```
## Recall
##  0.525
```

```r
f1_score <- 2*precision[[1]]*recall[[1]]/(precision[[1]]+recall[[1]])
f1_score
```

```
## [1] 0.65625
```

(d)

```r
mtry <- sqrt(ncol(train_x))

rf.base_1 <- randomForest(x = train_x, y = train_y, mtry = 0.5*mtry, importance = TRUE)
predict_test_1 <- predict(rf.base_1, newdata = test_x)
cmat_1 <- confusionMatrix(data = predict_test_1, reference = test_y)
cmat_1$overall[1]
```

```
## Accuracy
##     0.77
```

```r
rf.base_2 <- randomForest(x = train_x, y = train_y, mtry = 1.5*mtry, importance = TRUE)
predict_test_2 <- predict(rf.base_2, newdata = test_x)
cmat_2 <- confusionMatrix(data = predict_test_2, reference = test_y)
cmat_2$overall[1]
```

```
## Accuracy
##     0.83
```

mtry = 1.5*sqrt(# of features) yielded the best accuracy.