Text as Data Homework 2

Vanessa (Ziwei) Xu and zx657

2022-04-11

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: ggplot2

## Loading required package: lattice

## Package version: 3.2.1
## Unicode version: 13.0
## ICU version: 69.1

## Parallel computing: 8 of 8 threads used.

## See https://quanteda.io for tutorials and examples.

## randomForest 4.7-1

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin

## The following object is masked from 'package:dplyr':
##
##     combine

## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

Q1a

P(healthcare|Perdue) = 1/15, P(healthcare|Ossoff) = 1/21 P(voter|Perdue) = 1/15, P(voter|Ossoff) = 1/21 P(tax|Perdue) = 1/15, P(tax|Ossoff) = 0/21 P(help|Perdue) = 1/15, P(help|Ossoff) = 4/21 P(jobs|Perdue) = 1/15, P(jobs|Ossoff) = 1/21 Therefore, P(Perdue|d) = (1/15)^5 = 1/759375, P(Ossoff|d) = 0

Even though the posterior probability of the test email being sent by Perdue is small, compared to a probability of 0 for Ossoff, I would make a prediction that Perdue sent the mystery email (without smoothing).

Q1b

P(healthcare|Perdue) = 2/16, P(healthcare|Ossoff) = 2/22 P(voter|Perdue) = 2/16, P(voter|Ossoff) = 2/22 P(tax|Perdue) = 2/16, P(tax|Ossoff) = 1/22 P(help|Perdue) = 2/16, P(help|Ossoff) = 5/22 P(jobs|Perdue) = 2/16, P(jobs|Ossoff) = 2/22 Therefore, P(Perdue|d) = 1/32768 = 0.00003 , P(Ossoff|d) = 5/644204 = 7.76152... E-6

With smoothing, the posterior probability of the mystery email sent by Ossoff is not 0 anymore, but still small compared to that of Perdue. Therefore, I would make the conclusion that the prediction would be Perdue who sent the mystery email.

Beyond avoiding computational difficulties of zero, the reason why smoothing is implemented is that there could be a lot of other words that match this catogory but only one mismatch would throw this whole catogory to 0. By adding one to the numerator and the size of the vocabulary to the denominator, we can change 0 to a rather small possibility which would not alter the result but producing much more meaningful result.

Q2a - TripAdvisor

```
# load the csv file first
path <- "/Users/vanessaxu/Desktop/school/1015/homework/hw2/tripadvisor.csv"
trip <- read.csv(path)
# View(trip)
median_stars <- median(trip$stars) # 4

labels <- ifelse(trip$stars > median_stars, 'positive', 'negative')
trip['labels'] <- labels
prop.table(table(trip$labels))
```

```
##
##  negative  positive
## 0.5581475 0.4418525
```

```
# print results
cat("The median star rating is", median_stars)
```

```
## The median star rating is 4
```

The proportion of positive reviews is 0.4418525, and the proportion of negative reviews is 0.5581475.

Q2b

```
trip$anchor = cut(trip$stars,c(0,2,4,5), labels = c("negative","neutral","positive"))
# levels(data$group) = c("0-5","6-10",">10")
prop.table(table(trip$anchor))
```

```
##
##  negative   neutral  positive
## 0.1568493 0.4012981 0.4418525
```

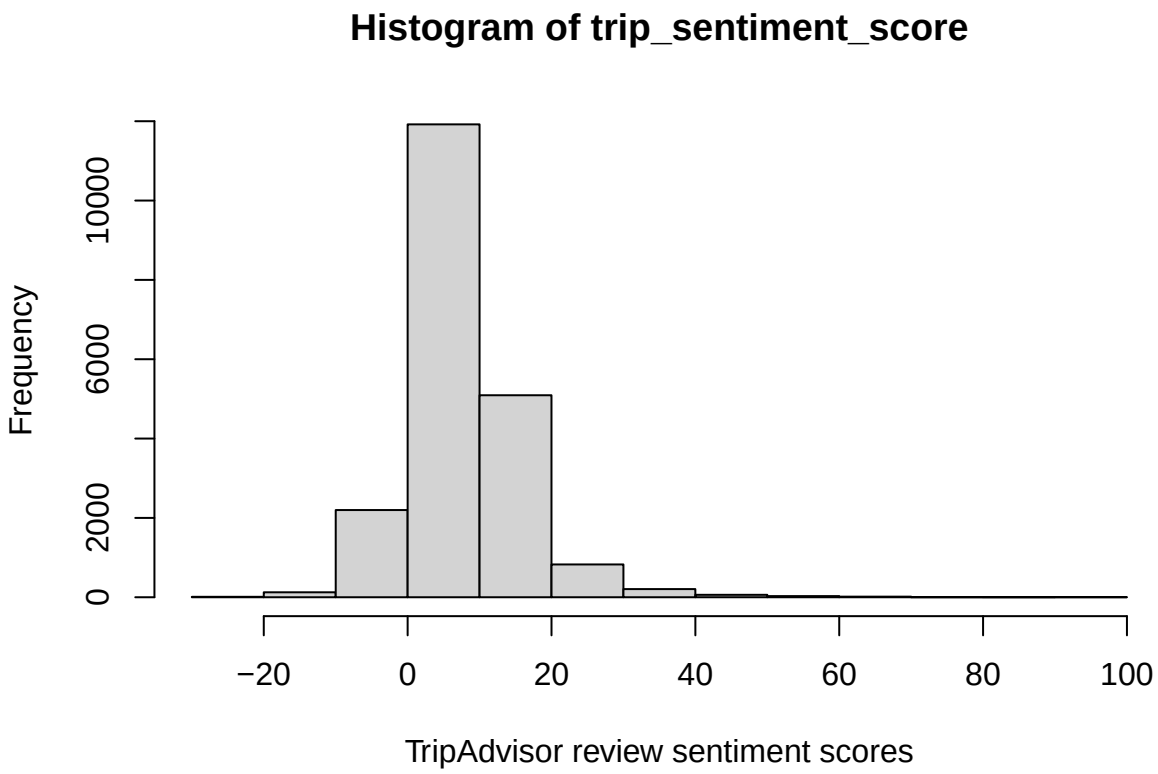
The proportion of reviews that are anchor positive is 0.4418525, anchor neutral is 0.4012981, and anchor negative is 0.1568493.

Q3a

```
# load the positive dictionary file
f <- file("/Users/vanessaxu/Desktop/school/1015/homework/hw2/positive-words.txt", open="r")
positive_words <- readLines(f)
close(f)
positive_words <- positive_words[-1:-2] # delete first two empty lines
# load the negative dictionary file
f <- file("/Users/vanessaxu/Desktop/school/1015/homework/hw2/negative-words.txt", open="r")
negative_words <- readLines(f)
close(f)
negative_words <- negative_words[-1:-2] # delete first two empty lines

# create a dictionary
trip_dict <- dictionary(list(positive = positive_words, negative = negative_words))
trip_dfm <- as.data.frame(dfm(trip$text, toLower=TRUE, dictionary = trip_dict))
```

```
## Warning: 'dfm.character()' is deprecated. Use 'tokens()' first.

## Warning: toLower argument is not used.

## Warning: toLower argument is not used.

## Warning: 'dictionary' and 'thesaurus' are deprecated; use dfm_lookup() instead

## Warning: 'as.data.frame.dfm' is deprecated.
## Use 'convert(x, to = "data.frame")' instead.
## See help("Deprecated")
```

```
# Compute predicted scores
trip_sentiment_score <- trip_dfm$positive - trip_dfm$negative
hist(trip_sentiment_score, xlab="TripAdvisor review sentiment scores")
```



Pre-processing choices: I only chose to lowercase all words, because I noticed that the dictionaries are all lowercased. I did not remove numbers, punctuations, and stopwords as I see some numbers and "-" in the dictionaries and I'm not sure if the dictionaries contain any stopwords and would like to leave them in just in case there are.

Q3b

```
trip_dfm['sentiment_score'] <- trip_sentiment_score
labels <- ifelse(trip_dfm$sentiment_score > 0, 'positive', 'negative')
trip_dfm['labels'] <- labels
prop.table(table(trip_dfm$labels))
```

```
##
##  negative  positive
## 0.1139037 0.8860963
```

```
# View(trip_dfm)
```

About 11.39% of reviews have negative sentiments and about 88.61% of reviews have positive sentiments.

Q3c

```r
# "confusion matrix": Naive Bayes
trip_cmat <- table(trip$labels, trip_dfm$labels)
trip_cmat
```

```
##
##            negative positive
##   negative     2234     9203
##   positive      100     8954
```

```r
# baseline accuracy
trip_baseline <- max(prop.table(table(trip_dfm$labels)))
trip_baseline
```

```
## [1] 0.8860963
```

```r
# get scores
trip_acc <- sum(diag(trip_cmat))/sum(trip_cmat) # accuracy = (TP + TN) / (TP + FP + TN + FN)
trip_recall <- trip_cmat[2,2]/sum(trip_cmat[2,]) # recall = TP / (TP + FN)
trip_precision <- trip_cmat[2,2]/sum(trip_cmat[,2]) # precision = TP / (TP + FP)
trip_f1 <- 2*(trip_recall*trip_precision)/(trip_recall + trip_precision)

# print
cat(
  "Baseline Accuracy: ", trip_baseline, "\n",
  "Accuracy:",  trip_acc, "\n",
  "Recall:",  trip_recall, "\n",
  "Precision:",  trip_precision, "\n",
  "F1-score:", trip_f1
)
```

```
## Baseline Accuracy:  0.8860963
##  Accuracy: 0.5459958
##  Recall: 0.9889552
##  Precision: 0.4931431
##  F1-score: 0.6581162
```

In terms of accuracy, the classfier isn't performing very well compared to its baseline. This is probably caused by a lot of False Positives – Only 11.39% of reviews have negative sentiments, where in reality about half of reviews are catogorized as negative.

Q4a

Pre-processing choices: I chose to lowercase all words, removed numbers, punctuation, and stopwords since these information are not as relevant when training a Naive Bayes classifier. I also stemmed the words– deleting duplicates and leaving only lower-cased, relevant words.

Q4b

```r
# View(trip)
# split sample into training & test sets
set.seed(1984L)
prop_train <- 0.2 #we will use 20% of the data as our training set
# Save the indexes
ids <- 1:nrow(trip)

ids_train <- sample(ids, ceiling(prop_train*length(ids)), replace = FALSE)
ids_test <- ids[-ids_train]
train_set <- trip[ids_train,]
test_set <- trip[ids_test,]

# tokenize, filter, and get dfm for both train and test set
tokenized_train <- tokens(train_set$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_train <- tokens_tolower(tokenized_train, keep_acronyms = FALSE)
tokenized_train <- tokens_remove(tokenized_train, pattern = stopwords("english"))
train_dfm <- dfm(tokenized_train)
train_dfm <- dfm_wordstem(train_dfm)
tokenized_test <- tokens(test_set$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_test <- tokens_tolower(tokenized_test, keep_acronyms = FALSE)
tokenized_test <- tokens_remove(tokenized_test, pattern = stopwords("english"))
test_dfm <- dfm(tokenized_test)
test_dfm <- dfm_wordstem(test_dfm)
# test_dfm - 56,144
# train_dfm - 22,689

# match test set dfm to train set dfm features
test_dfm <- dfm_match(test_dfm, features = featnames(train_dfm)) # test_dfm - 22,689

# train model on the training set using Laplace smoothing and uniform prior
nb_model_sm <- textmodel_nb(train_dfm, train_set$labels, smooth = 1, prior = "uniform")

# evaluate on test set
predicted_class_sm <- predict(nb_model_sm, newdata = test_dfm, force=TRUE)

# get confusion matrix
baseline_acc <- max(prop.table(table(test_set$labels)))
cmat_sm <- table(test_set$labels, predicted_class_sm)
nb_acc_sm <- sum(diag(cmat_sm))/sum(cmat_sm) # accuracy = (TP + TN) / (TP + FP + TN + FN)
nb_recall_sm <- cmat_sm[2,2]/sum(cmat_sm[2,]) # recall = TP / (TP + FN)
nb_precision_sm <- cmat_sm[2,2]/sum(cmat_sm[,2]) # precision = TP / (TP + FP)
nb_f1_sm <- 2*(nb_recall_sm*nb_precision_sm)/(nb_recall_sm + nb_precision_sm)

# print
cmat_sm
```

```
##           predicted_class_sm
##            negative positive
##   negative     6675     2444
##   positive     1601     5672
```

```
cat(
  "Baseline Accuracy: ", baseline_acc, "\n",
  "Accuracy:",  nb_acc_sm, "\n",
  "Recall:",  nb_recall_sm, "\n",
  "Precision:",  nb_precision_sm, "\n",
  "F1-score:", nb_f1_sm
)
```

```
## Baseline Accuracy:  0.556308
##  Accuracy: 0.7532333
##  Recall: 0.7798708
##  Precision: 0.6988664
##  F1-score: 0.7371499
```

Q4c

```
# train model on the training set using Laplace smoothing and docfreq prior
nb_model_sm2 <- textmodel_nb(train_dfm, train_set$labels, smooth = 1, prior = "docfreq")
# evaluate on test set
predicted_class_sm2 <- predict(nb_model_sm2, newdata = test_dfm, force=TRUE)

# get confusion matrix
baseline_acc2 <- max(prop.table(table(test_set$labels)))
cmat_sm2 <- table(test_set$labels, predicted_class_sm2)
nb_acc_sm2 <- sum(diag(cmat_sm2))/sum(cmat_sm2) # accuracy = (TP + TN) / (TP + FP + TN + FN)
nb_recall_sm2 <- cmat_sm2[2,2]/sum(cmat_sm2[2,]) # recall = TP / (TP + FN)
nb_precision_sm2 <- cmat_sm2[2,2]/sum(cmat_sm2[,2]) # precision = TP / (TP + FP)
nb_f1_sm2 <- 2*(nb_recall_sm2*nb_precision_sm2)/(nb_recall_sm2 + nb_precision_sm2)

# print
cmat_sm2
```

```
##           predicted_class_sm2
##            negative positive
##   negative     6765     2354
##   positive     1677     5596
```

```
cat(
  "Baseline Accuracy: ", baseline_acc2, "\n",
  "Accuracy:",  nb_acc_sm2, "\n",
  "Recall:",  nb_recall_sm2, "\n",
  "Precision:",  nb_precision_sm2, "\n",
  "F1-score:", nb_f1_sm2
)
```

```
## Baseline Accuracy:  0.556308
##  Accuracy: 0.7540874
##  Recall: 0.7694211
##  Precision: 0.7038994
##  F1-score: 0.7352033
```

I expect the change of priors from "uniform" to "docfreq" to slightly increase the performance of Naive Bayes predictions but not very much. This is because when training classes are balanced in their number of documents, the empirically computed "docfreq" would be equivalent to "uniform" priors. And in our case, the labels split the data in half (approximately). In terms of accuracy, the performance of this classifier has improved from 0.7532333 to 0.7540874, which is not a lot.

Q4d

```
# train model on the training set without smoothing and using uniform prior
nb_model3 <- textmodel_nb(train_dfm, train_set$labels, smooth = 0, prior = "uniform")
# evaluate on test set
predicted_class3 <- predict(nb_model3, newdata = test_dfm, force=TRUE)

# get confusion matrix
baseline_acc3 <- max(prop.table(table(test_set$labels)))
cmat_sm3 <- table(test_set$labels, predicted_class3)
nb_acc_sm3 <- sum(diag(cmat_sm3))/sum(cmat_sm3) # accuracy = (TP + TN) / (TP + FP + TN + FN)
nb_recall_sm3 <- cmat_sm3[2,2]/sum(cmat_sm3[2,]) # recall = TP / (TP + FN)
nb_precision_sm3 <- cmat_sm3[2,2]/sum(cmat_sm3[,2]) # precision = TP / (TP + FP)
nb_f1_sm3 <- 2*(nb_recall_sm3*nb_precision_sm3)/(nb_recall_sm3 + nb_precision_sm3)

# print
cmat_sm3
```

```
##           predicted_class3
##            negative positive
##   negative     7222     1897
##   positive     4031     3242
```

```
cat(
  "Baseline Accuracy: ", baseline_acc3, "\n",
  "Accuracy:",  nb_acc_sm3, "\n",
  "Recall:",  nb_recall_sm3, "\n",
  "Precision:",  nb_precision_sm3, "\n",
  "F1-score:", nb_f1_sm3
)
```

```
## Baseline Accuracy:  0.556308
##  Accuracy: 0.6383602
##  Recall: 0.4457583
##  Precision: 0.630862
##  F1-score: 0.5223977
```

The accuracy decreased from about 0.75 in previous models to about 0.64. This is because without smoothing, some features are counted as zero and would make predictions impossible.

Q4e

Other than words, we could also use positions and collocations as features to help classify the sentiment of a document. In addition, we could also consider who's the reviewer, who the reviewer is writing a review about, and the context of the review. Semantics that are expressed in a subtle manner and implicitly conveying relevant information can also be a feature in classification.

Q5a

```
# load files into corpus and dataframe
filenamesfull <- list.files(path = "/Users/vanessaxu/Desktop/school/1015/homework/hw2/manifestos", full
filenames <- list.files(path = "/Users/vanessaxu/Desktop/school/1015/homework/hw2/manifestos")
party <- unlist(regmatches(unlist(filenames), gregexpr("^[[:alpha:]]{3}", unlist(filenames))))
year <- unlist(regmatches(unlist(filenames), gregexpr("[[:digit:]]+", unlist(filenames))))
manifestos <- corpus(readtext(filenamesfull))
docvars(manifestos, field = c("party", "year") ) <- data.frame(cbind(party, year))
manifestos_df <- tibble(text = texts(manifestos), class = party, year = as.integer(year))
```

```
## Warning: 'texts.corpus' is deprecated.
## Use 'as.character' instead.
## See help("Deprecated")
```

```
# split the data into training and test sets
train_set <- manifestos_df[2:3,]
test_set <- manifestos_df[c(1,4),]
# tokenize, filter, and get dfm for both train and test set
tokenized_train <- tokens(train_set$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_train <- tokens_tolower(tokenized_train, keep_acronyms = FALSE)
tokenized_train <- tokens_remove(tokenized_train, pattern = stopwords("english"))
train_dfm <- dfm(tokenized_train)
train_dfm <- dfm_wordstem(train_dfm)
tokenized_test <- tokens(test_set$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_test <- tokens_tolower(tokenized_test, keep_acronyms = FALSE)
tokenized_test <- tokens_remove(tokenized_test, pattern = stopwords("english"))
test_dfm <- dfm(tokenized_test)
test_dfm <- dfm_wordstem(test_dfm)

# Word Score model w/o smoothing
ws_base <- textmodel_wordscores(train_dfm,
                                y = (2 * as.numeric(train_set$class == "Lab")) - 1, smooth = 1)
# Y variable must be coded on a binary x in {-1,1} scale,
# so -1 = Conservative and 1 = Labour

# Look at strongest features
lab_features <- sort(ws_base$wordscores, decreasing = TRUE)  # for labor
cat("Ten most extreme words for the Labour party are: ")
```

```
## Ten most extreme words for the Labour party are:
```

```
lab_features[1:10]
```

```
##    materi       won  ministri       war       let    declar   victori   section
## 0.8864467 0.8563334 0.8563334 0.8482868 0.8343719 0.8343719 0.8343719 0.8343719
##   suitabl   struggl
## 0.8343719 0.8044846
```

```
con_features <- sort(ws_base$wordscores, decreasing = FALSE)  # for conservative
cat("Ten most extreme words for the Conservative party are: ")
```

```
## Ten most extreme words for the Conservative party are:
```

```
con_features[1:10]
```

```
##   encourag     scheme      train    pension        now    continu      shall
## -0.8876039 -0.8674221 -0.8384063 -0.8310093 -0.8229027 -0.8108012 -0.8061654
##    council       four      spend
## -0.8041084 -0.7931315 -0.7931315
```

Pre-processing choices: I chose to lowercase all words, removed numbers, punctuation, and stopwords since these information are not as relevant when creating word scores here. I also stemmed the words–deleting duplicates and leaving only lower-cased, relevant words.

I decided to use smoothing as this yields a more hollistic result. I also tried both with and without smoothing and found that the words / features with smoothing generates make more sense.

Q5b

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

```
## Warning: 499 features in newdata not used in prediction.
```

```
## Con1979.txt Lab1951.txt
## -0.12962611 -0.02540098
```

This actually did NOT generate the correct answer for classfication of party for Lab1951.txt. Although the score for Lab1951.txt is more towards the expected, which is 1, compared to Con1979.txt, the score is still negative, which would classify this as "Con" instead of "Lab".

The warning message suggests that 499 features in test data were not used in prediction. This means that 499 new words in the test documents were not in the reference / training texts are ignore: the sum is only over the words we've seen in the reference / training texts. The shrinkage of the scores can be problematic as wrong classification can be made; therefore, further normalization and rescaling are needed to obtain more accurate results.

Q5c

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

```
## Warning: 499 features in newdata not used in prediction.
```

```
## Con1979.txt Lab1951.txt
##  -1.0775135   0.9224865
```

After fitting the model with the standard LBG rescaling, results differ greatly from previous ones. This time, score for Lab1951.txt is positive and very close to 1. Score for Con1979.txt also approaches -1 a lot more. Rescaling solved the problem of shrinkage of the scores.

Q6a

Pre-processing choices: I chose to lowercase all words, removed numbers, punctuation, and stopwords since these information are not as relevant when using a Support Vector Machine model. I also stemmed the words–deleting duplicates and leaving only lower-cased, relevant words. I also deleted apostrophes as I discovered they would skew the results by adding features like "n't".

SVM or Naive Bayes offers an advantage of making no assumptions of text data compared to dictionary and wordscore methods. The dictionary and wordscores methods are generally not as effective as machine-learning approaches like Naive Bayes or SVM because they've decided which words are important for discriminating the two classes instead of learning straight from the data. The dictionary approach does this by choosing all reference words completely indepentently of texts that are actually used in the dataset. The wordscores approach does this by deciding that the anchor documents, which are the most extreme ones, are the most useful for discrimination. Naive Bayes and SVM make no such assumptions.

Q6b

```
# sample the reviews dataset to the first 1000 rows
subtrip = trip[1:1000,]
prop.table(table(subtrip$labels))
```

```
##
## negative positive
##    0.666    0.334
```

```
# randomize order
subtrip <- subtrip %>% sample_n(nrow(subtrip))
rownames(subtrip) <- NULL

# tokenize, filter, and get dfm for subtrip
subtrip$text <- gsub(pattern = "'", "", subtrip$text)  # replace apostrophes
tokenized <- tokens(subtrip$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenizedn <- tokens_tolower(tokenized, keep_acronyms = FALSE)
tokenized <- tokens_remove(tokenized, pattern = stopwords("english"))
subtrip_dfm <- dfm(tokenized)
subtrip_dfm <- dfm_wordstem(subtrip_dfm)

# create a for loop to use different levels of training and test sets
for (i in c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9))
{
ids_train <- createDataPartition(1:nrow(subtrip_dfm), p = i, list = FALSE, times = 1)
train_x <- subtrip_dfm[c(ids_train), ] # train set data
train_y <- subtrip$labels[ids_train] %>% as.factor()  # train set labels
test_x <- subtrip_dfm[c(-ids_train), ] # test set data
test_y <- subtrip$labels[-ids_train] %>% as.factor() # test set labels

# baseline
baseline_sub <- max(prop.table(table(test_y)))

# now we will have train / test / validation
trctrl <- trainControl(method = "cv", number = 5)
```

```r
# svm - linear
svm_mod_linear <- train(x = train_x, y = train_y, method = "svmLinear", trControl = trctrl, scale = FALS

# predict on heldout validation data
svm_linear_pred <- predict(svm_mod_linear, newdata = test_x)
svm_linear_cmat <- confusionMatrix(svm_linear_pred, test_y)

# confusion matrix on predictions
svm_linear_cmat

cat(
  "Iteration of training size:", i, "\n",
  "Baseline Accuracy: ", baseline_sub, "\n",
  "SVM-Linear Accuracy:",  svm_linear_cmat$overall[["Accuracy"]], "\n"
)
}
```

```
## Iteration of training size: 0.1
##  Baseline Accuracy:  0.6633333
##  SVM-Linear Accuracy: 0.6688889
## Iteration of training size: 0.2
##  Baseline Accuracy:  0.6525
##  SVM-Linear Accuracy: 0.7425
## Iteration of training size: 0.3
##  Baseline Accuracy:  0.6614286
##  SVM-Linear Accuracy: 0.7357143
## Iteration of training size: 0.4
##  Baseline Accuracy:  0.68
##  SVM-Linear Accuracy: 0.725
## Iteration of training size: 0.5
##  Baseline Accuracy:  0.64
##  SVM-Linear Accuracy: 0.734
## Iteration of training size: 0.6
##  Baseline Accuracy:  0.6475
##  SVM-Linear Accuracy: 0.74
## Iteration of training size: 0.7
##  Baseline Accuracy:  0.6533333
##  SVM-Linear Accuracy: 0.7733333
## Iteration of training size: 0.8
##  Baseline Accuracy:  0.695
##  SVM-Linear Accuracy: 0.73
## Iteration of training size: 0.9
##  Baseline Accuracy:  0.63
##  SVM-Linear Accuracy: 0.84
```

The model with training size of 90% has the highest accuracy of 0.84 for out-of-sample predictions made on the validation set. In terms of accuracy, the performance of this classifier has improved from previous methods, which generated a best accuracy of 0.75.

Q6c

```
ids_train2 <- createDataPartition(1:nrow(subtrip_dfm), p = 0.8, list = FALSE, times = 1)
train_x2 <- subtrip_dfm[c(ids_train2), ] # train set data
train_y2 <- subtrip$labels[ids_train2] %>% as.factor()  # train set labels
test_x2 <- subtrip_dfm[c(-ids_train2), ] # test set data
test_y2 <- subtrip$labels[-ids_train2] %>% as.factor() # test set labels
train_x2 <- train_x2 %>% as.data.frame()
```

```
## Warning: 'as.data.frame.dfm' is deprecated.
## Use 'convert(x, to = "data.frame")' instead.
## See help("Deprecated")
```

```
train_x2 <- train_x2[,-1] # delete first column of doc_id
test_x2 <- test_x2 %>% as.data.frame()
```

```
## Warning: 'as.data.frame.dfm' is deprecated.
## Use 'convert(x, to = "data.frame")' instead.
## See help("Deprecated")
```

```
test_x2 <- test_x2[,-1] # delete first column of doc_id

# now we will have train / test / validation
trctrl2 <- trainControl(method = "cv", number = 5)
svm_mod_logistic <- train(x = train_x2, y = train_y2,
                          method = "glm", family = "binomial", trControl = trctrl2)
```

```
## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading

## Warning: glm.fit: algorithm did not converge
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
## Warning: glm.fit: algorithm did not converge
```

```
# predict on heldout validation data
svm_logistic_pred <- predict(svm_mod_logistic, newdata = test_x2)
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type = if (type == :
## prediction from a rank-deficient fit may be misleading
```

```
svm_logistic_cmat <- confusionMatrix(svm_logistic_pred, test_y2)

# confusion matrix on predictions
svm_logistic_cmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       60       26
##   positive       68       46
##
##                Accuracy : 0.53
##                  95% CI : (0.4583, 0.6008)
##     No Information Rate : 0.64
##     P-Value [Acc > NIR] : 0.9994
##
##                   Kappa : 0.0955
##
##  Mcnemar's Test P-Value : 2.349e-05
##
##             Sensitivity : 0.4688
##             Specificity : 0.6389
##          Pos Pred Value : 0.6977
##          Neg Pred Value : 0.4035
##              Prevalence : 0.6400
##          Detection Rate : 0.3000
##    Detection Prevalence : 0.4300
##       Balanced Accuracy : 0.5538
##
##        'Positive' Class : negative
##
```

```
cat(
  "Training size:", 0.8, "\n",
  "SVM-Linear Accuracy:",  svm_logistic_cmat$overall[["Accuracy"]], "\n"
)
```

```
## Training size: 0.8
##  SVM-Linear Accuracy: 0.53
```

What would help with convergence: increasing the number of iterations and examining the number of features used (potentially lowering the number of features used in the model). With the training size of 80%, the accuracy measure using regression of 0.53 is lower than the highest accuracy produced by the SVM mode.

Q7a

```
# use the first 500 Trip Advisor reviews in the dataset
rftrip <- trip[1:500,]
prop.table(table(rftrip$labels))
```

```
##
## negative positive
##    0.724    0.276
```

```
# Split the dataset into a training (80%) and a test set (20%)
prop_train3 <- 0.8 # we will use 80% of the data as our training set
# Save the indexes
ids3 <- 1:nrow(rftrip)

ids_train3 <- sample(ids3, ceiling(prop_train3*length(ids3)), replace = FALSE)
ids_test3 <- ids3[-ids_train3]
train_set3 <- rftrip[ids_train3,]
test_set3 <- rftrip[ids_test3,]

# tokenize, filter, and get dfm for both train and test set
tokenized_train3 <- tokens(train_set3$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_train3 <- tokens_tolower(tokenized_train3, keep_acronyms = FALSE)
tokenized_train3 <- tokens_remove(tokenized_train3, pattern = stopwords("english"))
train_dfm3 <- dfm(tokenized_train3)
train_dfm3 <- dfm_wordstem(train_dfm3)
tokenized_test3 <- tokens(test_set3$text, remove_numbers=TRUE, remove_punct = TRUE)
tokenized_test3 <- tokens_tolower(tokenized_test3, keep_acronyms = FALSE)
tokenized_test3 <- tokens_remove(tokenized_test3, pattern = stopwords("english"))
test_dfm3 <- dfm(tokenized_test3)
test_dfm3 <- dfm_wordstem(test_dfm3)

# match features in the test set to the set of features in the training set
test_dfm3 <- dfm_match(test_dfm3, features = featnames(train_dfm3))
```

Pre-processing choices: I chose to lowercase all words, removed numbers, punctuation, and stopwords since these information are not as relevant when using a Support Vector Machine model. I also stemmed the words–deleting duplicates and leaving only lower-cased, relevant words.

Q7b

```
rftrain_x <- train_dfm3 %>% convert("matrix")
# rftrain_x <- as.data.frame(rftrain_x)
rftest_x <- test_dfm3 %>% convert("matrix")
# rftest_x <- as.data.frame(rftest_x)
```

```
rftrain_y <- train_set3$labels %>% as.factor()
rftest_y <- test_set3$labels %>% as.factor()

system.time(rf.base <- randomForest(x = rftrain_x, y = rftrain_y, importance = TRUE))
```

```
##    user  system elapsed
##  36.195   0.111  36.335
```

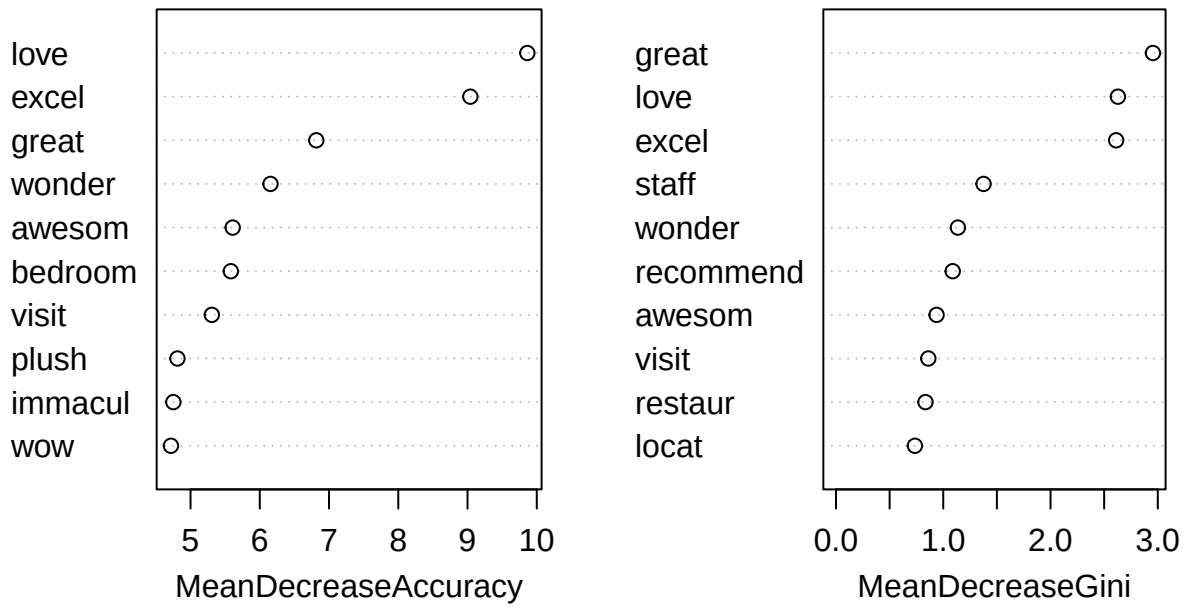
```
token_importance <- round(importance(rf.base, 2), 2)
rownames(token_importance)[order(-token_importance)][1:10]
```

```
##  [1] "great"     "love"      "excel"     "staff"     "wonder"    "recommend"
##  [7] "awesom"    "visit"     "restaur"   "locat"
```

```
# print results
print(rf.base)
```

```
##
## Call:
##  randomForest(x = rftrain_x, y = rftrain_y, importance = TRUE)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 67
##
##         OOB estimate of  error rate: 27%
## Confusion matrix:
##          negative positive class.error
## negative      284        3  0.01045296
## positive      105        8  0.92920354
```

```
varImpPlot(rf.base, n.var = 10, main = "Variable Importance")
```



The top 10 most important ordered by the mean decrease in Gini index would be "great", "love", "excel", "wonder", "staff", "awesom", "restaur", "recommend", "perfect", and "fantast". (These are results after stemming and other pre-processing steps I've done which was explained earlier in Q7a.)

Q7c

```
# prediction
predict_test <- predict(rf.base, newdata = rftest_x)
# get confusion matrix
rfcmat <- confusionMatrix(data = predict_test, reference = rftest_y)
rfcmat
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction negative positive
##   negative       75       24
##   positive        0        1
##
##                Accuracy : 0.76
##                  95% CI : (0.6643, 0.8398)
##     No Information Rate : 0.75
##     P-Value [Acc > NIR] : 0.4617
##
##                   Kappa : 0.0588
```

```
##
##  Mcnemar's Test P-Value : 2.668e-06
##
##             Sensitivity : 1.0000
##             Specificity : 0.0400
##          Pos Pred Value : 0.7576
##          Neg Pred Value : 1.0000
##              Prevalence : 0.7500
##          Detection Rate : 0.7500
##    Detection Prevalence : 0.9900
##       Balanced Accuracy : 0.5200
##
##        'Positive' Class : negative
##
```

```
# print
cat(
  "Accuracy:", rfcmat$overall[1], "\n",
  "Recall:", rfcmat$byClass[6], "\n",
  "Precision:", rfcmat$byClass[5], "\n",
  "F1-score:", rfcmat$byClass[7]
)
```

```
## Accuracy: 0.76
##  Recall: 1
##  Precision: 0.7575758
##  F1-score: 0.862069
```

Q7d

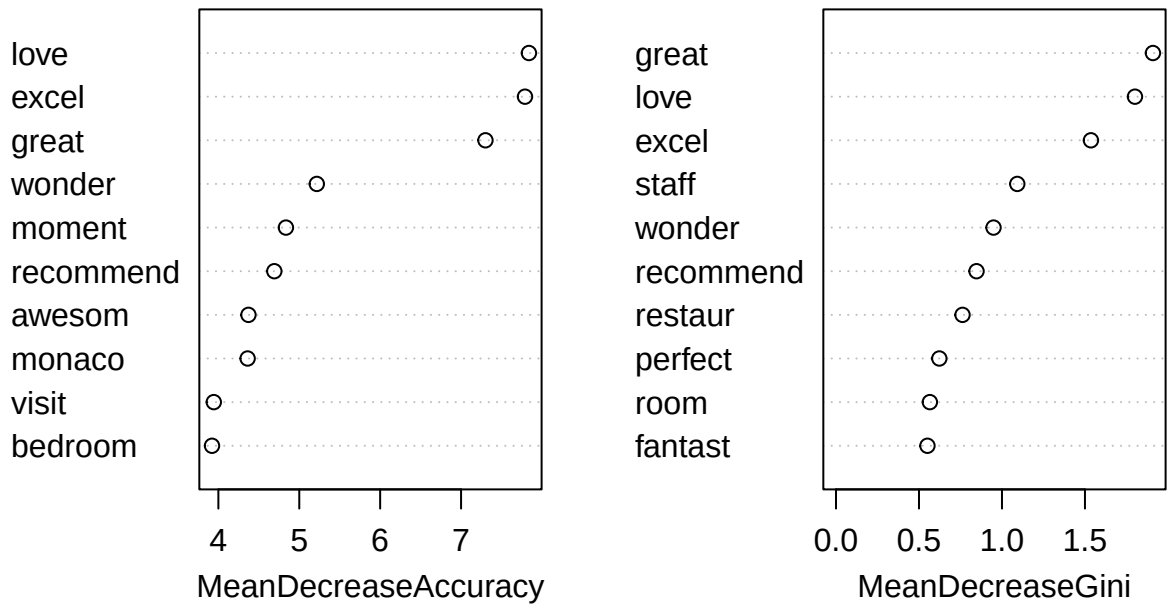
```
mtry1 <- 0.5*sqrt(ncol(rftrain_x))
mtry2 <-1.5*sqrt(ncol(rftrain_x))
# use each of the fitted models to predict the sentiment values for the test set.
system.time(rf.base1 <- randomForest(x = rftrain_x, y = rftrain_y, mtry = mtry1, importance = TRUE))
```

```
##    user  system elapsed
##  41.237   0.168  41.424
```

```
token_importance1 <- round(importance(rf.base1, 2), 2)
rownames(token_importance1)[order(-token_importance1)][1:10]
```

```
##  [1] "great"     "love"      "excel"     "staff"     "wonder"    "recommend"
##  [7] "restaur"   "perfect"   "room"      "fantast"
```

```
# print results
varImpPlot(rf.base1, n.var = 10, main = "Variable Importance")
```




```
# prediction
predict_test1 <- predict(rf.base1, newdata = rftest_x)
# get confusion matrix
rfcmat1 <- confusionMatrix(data = predict_test1, reference = rftest_y)
# print
cat(
  "Accuracy:", rfcmat1$overall[1], "\n",
  "Recall:", rfcmat1$byClass[6], "\n",
  "Precision:", rfcmat1$byClass[5], "\n",
  "F1-score:", rfcmat1$byClass[7]
)
```

```
## Accuracy: 0.75
##  Recall: 1
##  Precision: 0.75
##  F1-score: 0.8571429
```

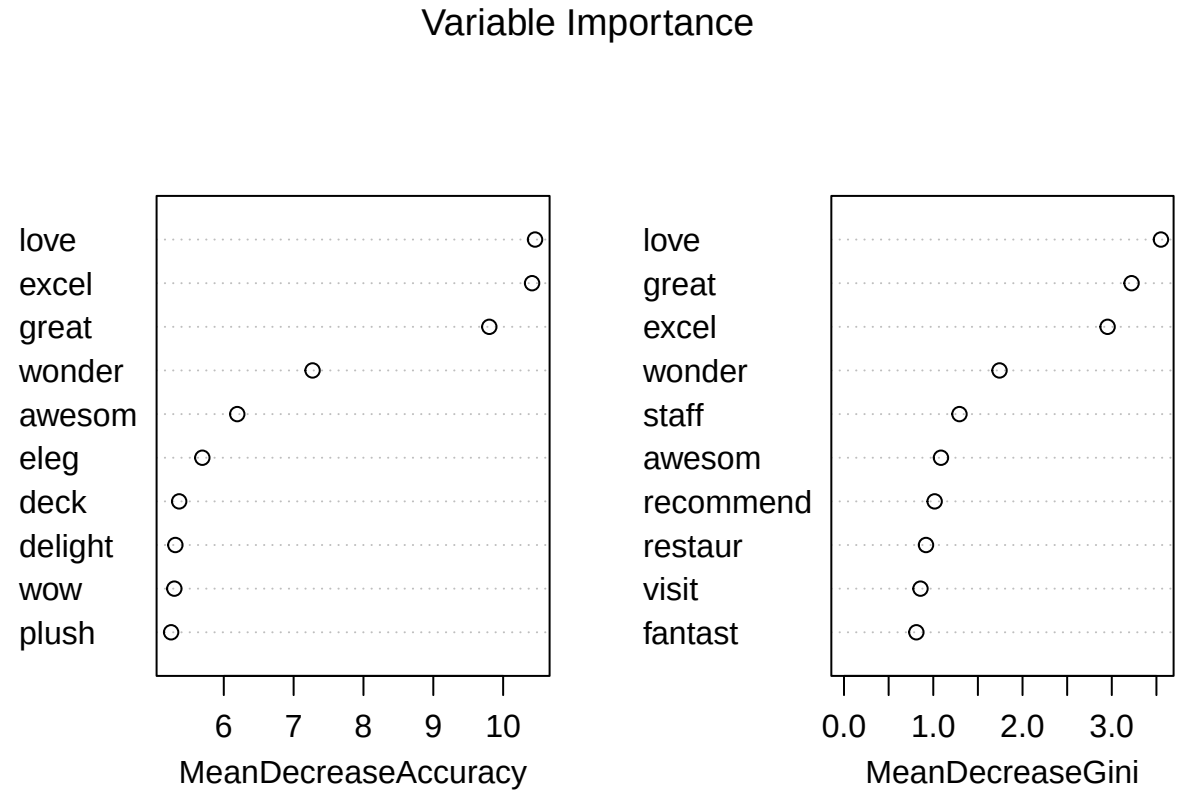
```
system.time(rf.base2 <- randomForest(x = rftrain_x, y = rftrain_y, mtry = mtry2, importance = TRUE))
```

```
##    user  system elapsed
##  33.685   0.104  33.845
```

```
token_importance2 <- round(importance(rf.base2, 2), 2)
rownames(token_importance2)[order(-token_importance2)][1:10]
```

```
##  [1] "love"      "great"     "excel"     "wonder"    "staff"     "awesom"
##  [7] "recommend" "restaur"   "visit"     "fantast"
```

```
# print results
varImpPlot(rf.base2, n.var = 10, main = "Variable Importance")
```




```
# prediction
predict_test2 <- predict(rf.base2, newdata = rftest_x)
# get confusion matrix
rfcmat2 <- confusionMatrix(data = predict_test2, reference = rftest_y)
# print
cat(
  "Accuracy:", rfcmat2$overall[1], "\n",
  "Recall:", rfcmat2$byClass[6], "\n",
  "Precision:", rfcmat2$byClass[5], "\n",
  "F1-score:", rfcmat2$byClass[7]
)
```

```
## Accuracy: 0.77
##  Recall: 1
##  Precision: 0.7653061
##  F1-score: 0.867052
```

Accuracy is 0.75 for 0.5*sqrt(# of features) and 0.78 for 1.5*sqrt(# of features). Therefore, the mtry value of 1.5*sqrt(# of features) yielded the best accuracy.