Supervised Learning

Lingyi Zhang

3/22/2018

```
# Clear Global Environment and Set working directory
rm(list = ls())
#getwd()
setwd("/Users/Lingyi/TAD/HW2")
set.seed(8888)

#import needed libraries
library(quanteda)
library(tidyverse)
library(stringr)
library(tm)
library(caret)
library(RTextTools)
```

Part 1

1. We would like you to perform some Naive Bayes classification by hand (that is, you may use math functions in base R, but not any built-in naive Bayes functions)—make sure to show your work!

(a) Imagine a situation in which you are a French citizen and you receive eight emails—in English for some reason—from the two leading Presidential Candidates: Marine La Pen and Emmanuel Macron. The contents of those emails after all relevant preprocessing are displayed in Table 1. Using the standard Naive Bayes classifier without smoothing, estimate the probability (or rather, the prior multiplied by the likelihood) that the following email was sent from Le Pen or Macron: "immigration voter culture help reform". Report this estimate. Explain whether you trust your findings and why.

```
#store the data we have
candidates_email <- tribble(
                       ~email, ~content,
                       #------/---------
                       "lepen", "immigration women assimilate help win",
                       "lepen", "culture voter economy president afraid",
                       "macron", "voter women help reform education",
                       "macron", "union economy hope immigration neighborhood",
                       "macron", "win union success elect president",
                       "lepen", "economy voter immigration president culture",
                       "macron", "europe german culture help french",
                       "unknown", "immigration voter culture help reform"
                     )

#convert to dfm
email_dfm <- dfm(candidates_email$content)
```

```r
docnames(email_dfm) <- candidates_email$email
email_dfm
```

```
## Document-feature matrix of: 8 documents, 20 features (75% sparse).
## 8 x 20 sparse Matrix of class "dfm"
##          features
## docs      immigration women assimilate help win culture voter economy
##   lepen             1     1          1    1   1       0     0       0
##   lepen             0     0          0    0   0       1     1       1
##   macron            0     1          0    1   0       0     1       0
##   macron            1     0          0    0   0       0     0       1
##   macron            0     0          0    0   1       0     0       0
##   lepen             1     0          0    0   0       1     1       1
##   macron            0     0          0    1   0       1     0       0
##   unknown           1     0          0    1   0       1     1       0
##          features
## docs      president afraid reform education union hope neighborhood
##   lepen           0      0      0         0     0    0            0
##   lepen           1      1      0         0     0    0            0
##   macron          0      0      1         1     0    0            0
##   macron          0      0      0         0     1    1            1
##   macron          1      0      0         0     1    0            0
##   lepen           1      0      0         0     0    0            0
##   macron          0      0      0         0     0    0            0
##   unknown         0      0      1         0     0    0            0
##          features
## docs      success elect europe german french
##   lepen         0     0      0      0      0
##   lepen         0     0      0      0      0
##   macron        0     0      0      0      0
##   macron        0     0      0      0      0
##   macron        1     1      0      0      0
##   lepen         0     0      0      0      0
##   macron        0     0      1      1      1
##   unknown       0     0      0      0      0
```

Calculation:

```
Pr(lepen) = 3/7
Pr(macron) = 4/7

Pr(immigration|lepen) = 2 / 5*3 = 2/15
Pr(help|lepen) = 1 / 5*3 = 1/15
Pr(culture|lepen) = 2 / 5*3 = 2/15
Pr(voter|lepen) = 2 / 5*3 = 2/15
Pr(reform|lepen) = 0 / 5*3 = 0

Pr(immigration|macron) = 1 / 5*4 = 1/20
Pr(help|macron) = 2 / 5*4 = 1/10
Pr(culture|macron) = 1 / 5*4 = 1/20
Pr(voter|macron) = 1 / 5*4 = 1/20
Pr(reform|macron) = 1 / 5*4 = 1/20
```

```r
Pr_lepen_d <- (3/7) * (2/15) * (1/15) * (2/15) * (2/15) * 0
Pr_macron_d <- (4/7) * (1/20) * (1/10) * (1/20) * (1/20) * (1/20)
```

```r
Pr_lepen_d < Pr_macron_d
```

```
## [1] TRUE
```

Pr(lepen|d) < Pr(macron|d). This result shows that the email is from Macron. But the finding is not trustworthy, as there is one "0" for lepen's calculation, which totally changed the result. Applying some smoothing can mitigate this "0" effect on the result.

(b) Now impose Laplace smoothing on the problem. That is, add one to the numerator and the size of the vocabulary to the denominator, then recalculate your estimates from above. Report your findings. Comment on which candidate is now the more plausible originator of the email.

Calculation:

```
Pr(lepen) = (3+1) / (7+2) = 4/9
Pr(macron) = (4+1) / (7+2) = 5/9

Pr(immigration|lepen) = (2+1) / (5*3+20)
Pr(help|lepen) = (1+1) / (5*3+20)
Pr(culture|lepen) = (2+1) / (5*3+20)
Pr(voter|lepen) = (2+1) / (5*3+20)
Pr(reform|lepen) = (0+1) / (5*3+20)

Pr(immigration|macron) = (1+1) / (5*4+20)
Pr(help|macron) = (2+1) / (5*4+20)
Pr(culture|macron) = (1+1) / (5*4+20)
Pr(voter|macron) = (1+1) / (5*4+20)
Pr(reform|macron) = (1+1) / (5*4+20)
```

```r
Pr_lepen_d <- (4/9) * (2+1) / (5*3+20) * (1+1) / (5*3+20) * (
               2+1) / (5*3+20) * (2+1) / (5*3+20) * (0+1) / (5*3+20)
Pr_macron_d <- (5/9) * (1+1) / (5*4+20) * (2+1) / (5*4+20) * (
               1+1) / (5*4+20) * (1+1) / (5*4+20) * (1+1) / (5*4+20)
Pr_lepen_d < Pr_macron_d
```

```
## [1] FALSE
```

Pr(lepen|d) > Pr(macron|d). This result shows that the email is from Lepen. I think this result is more reliable, as it's not largely affected by the extreme value like "0".

Part 2

Leslie has gathered 10,000 user reviews from Yelp. Each user left a star rating of 1-5 along with a written review. You'll be asked to use some of the supervised learning techniques we've discussed in class to analyze these texts.

Download the most recent version from the course GitHub. The data are available in the file "yelp reviews.csv".

Before we get started, be sure to actually read a few of the reviews, to get a feel for the language used, and any potential imperfections in the text created during the scraping process.

For each task (3) through (6), begin with the raw version of the text, and briefly explain which pre-processing steps are appropriate for that particular task.

2. Before we apply any classification algorithms to the Yelp reviews, we will need a general classifier that tells us whether the review was positive or negative—also referred to as the "actual score."

(a) Divide the reviews at the empirical median score and assign each review a label as being "positive"—if the user score was greater than the empirical median score—or "negative"—if the review is less than or equal to the empirical median. Alternatively, you may code "actual score" as 1 if a user score is greater than the median and 0 otherwise. Report the percent of reviews in each category.

```
df <- read.csv("yelp_reviews.csv", stringsAsFactors = FALSE)

#empirical median score
median_point <- median(df$stars)
cat("empirical median score is ", median_point)
```

```
## empirical median score is  4
```

```
#add a column of actual score either positivie or negtive
df <- mutate(df, actual_score = ifelse(stars > median_point, "positive", "negative"))

#calculate the ratio
df %>% group_by(actual_score) %>% summarise(ratio = length(actual_score)/10000)
```

```
## # A tibble: 2 x 2
##   actual_score ratio
##   <chr>        <dbl>
## 1 negative     0.645
## 2 positive     0.355
```

64.46% are negative, and 35.54% are positive.

(b) For some tasks, we're interested in having some "anchor" texts at the extreme of the distribution. Create a binary variable ("anchor positive") that is equal to one if the user score given to a review is equal to 5 and is 0 otherwise. Next, create a second binary variable ("anchor negative") that is equal to 1 when the user scores of a given review are equal to 1; otherwise, "anchor negative" is 0. Report the percent of reviews in each category or neither.

```
#add two seperate columns of whether the review
#is anchor_positive or is anchor_negative
df <- mutate(df, anchor_positive = ifelse(stars == 5, 1, 0))
df <- mutate(df, anchor_negative = ifelse(stars == 1, 1, 0))

#calculate the ratio
df %>% group_by(anchor_positive) %>% summarise(
    anch_posi_ratio = length(anchor_positive)/10000)
```

```
## # A tibble: 2 x 2
##   anchor_positive anch_posi_ratio
##             <dbl>           <dbl>
## 1            0              0.645
## 2            1.00           0.355
```

```
#calculate the ratio
df %>% group_by(anchor_negative) %>% summarise(
```

```
    anch_nega_ratio = length(anchor_negative)/10000)
```

```
## # A tibble: 2 x 2
##   anchor_negative anch_nega_ratio
##             <dbl>           <dbl>
## 1               0          0.913
## 2            1.00          0.0873
```

```
neither <- 1 - 0.3554 - 0.0873

#put these 3 categories into a matrix
category_matrix <- matrix(c(0.3554,0.0873,neither), nrow = 1, ncol = 3, byrow = TRUE,
                       dimnames = list("percent", c(
                           "anchor_positive", "anchor_negative", "neither")))
category_matrix
```

```
##         anchor_positive anchor_negative neither
## percent          0.3554          0.0873  0.5573
```

3. The first method we'll use to classify reviews as being positive or negative will be sentiment classification. To do so, you will use the dictionaries of positive and negative words discussed in Hu & Liu (2004)—available on GitHub.

(a) First, generate a sentiment score for each review based on the number of positive words minus the number of negative words. Then, create a vector of dichotomous variables, of equal length to the number of reviews, in which texts that have a positive sentiment score are labelled "positive," while those with a negative score are labelled "negative"; if any of them have a sentiment score of 0, score them as positive. If you used 1 and 0 to code reviews as being positive and negative for "actual score" in 2(a), do the same here. Report the percent of reviews in each category and neither, and discuss the results.

```
dict_pos <- read.table("positive-words.txt", stringsAsFactors = FALSE)
dict_neg <- read.table("negative-words.txt", stringsAsFactors = FALSE)

#construct the dictionary
dict_HuLiu <- dictionary(list(positive=dict_pos$V1, negative=dict_neg$V1))

dfm_yelp <- dfm(df$text, stem=FALSE, tolower=TRUE, dictionary=dict_HuLiu)
df_senti <- as.data.frame(dfm_yelp)
head(df_senti, n=3)
```

```
##       positive negative
## text1        4        0
## text2        3        0
## text3        2        0
```

```
#calculate and store the sentiment lable as negative or positive
df$senti_score <- df_senti$positive - df_senti$negative
df <- mutate(df, senti_label = ifelse(senti_score < 0, "negative", "positive"))
head(df, n=3)
```

```
##   stars
## 1     5
## 2     5
## 3     5
```

```
##
## 1
## 2 Small unassuming place that changes their menu every so often. Cool decor and vibe inside their 30
## 3
##   actual_score anchor_positive anchor_negative senti_score senti_label
## 1     positive               1               0           4    positive
## 2     positive               1               0           3    positive
## 3     positive               1               0           2    positive
```

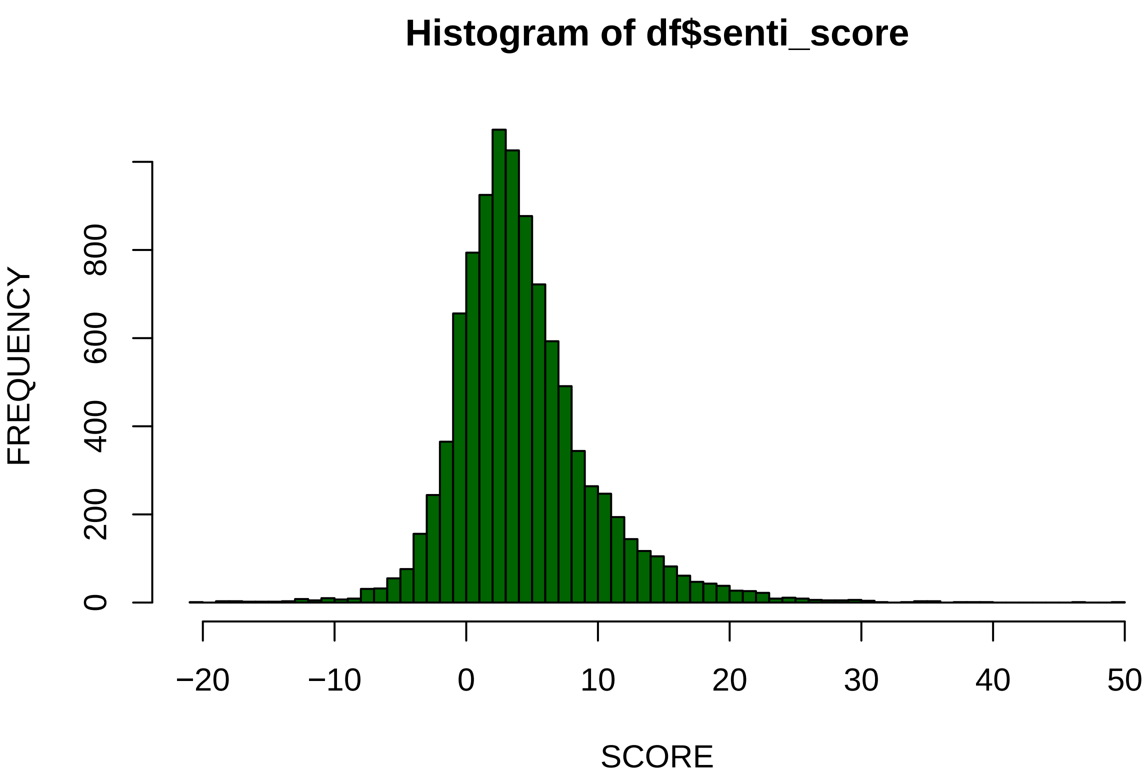
```r
#calculate the ratio
df %>% group_by(senti_label) %>% summarise(ratio = length(senti_label)/10000)
```

```
## # A tibble: 2 x 2
##   senti_label ratio
##   <chr>       <dbl>
## 1 negative    0.101
## 2 positive    0.899
```

All reviews have been labeled as either positive or negative. Based on the sentiment, less reviews are identified as negative compared to the actual score. Maybe because some reviewers were not satisfied but they expressed in a more positive way. And sometimes the stars are not very accurate as a measurement.

(b) Create a histogram to visualize the distribution of the continuous sentiment measure. Your answer should be a graph.

```r
hist(df$senti_score, col="darkgreen", ylab="FREQUENCY", xlab="SCORE", breaks=60)
```



(c) Determine the usefulness of your model at identifying positive or negative reviews by creating a confusion matrix with the positive and negative values assigned by the sentiment score (created in 3(a)) on one axis and the original binary classifications (created in 2(b)) on

the other axis. Use this confusion matrix to compute the accuracy, precision, and recall of the sentiment classifier. Report these findings. Include the confusion matrix in your answer.

```
#confusion matrix
pred <- df$senti_label
true <- df$actual_score
xtab <- table(pred, true)

confusionMatrix(xtab)
```

```
## Confusion Matrix and Statistics
##
##           true
## pred       negative positive
##   negative      937       77
##   positive     5509     3477
##
##                Accuracy : 0.4414
##                  95% CI : (0.4316, 0.4512)
##     No Information Rate : 0.6446
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0921
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.1454
##             Specificity : 0.9783
##          Pos Pred Value : 0.9241
##          Neg Pred Value : 0.3869
##              Prevalence : 0.6446
##          Detection Rate : 0.0937
##    Detection Prevalence : 0.1014
##       Balanced Accuracy : 0.5618
##
##        'Positive' Class : negative
##
```

```
#report the confusion matrix
results <- confusionMatrix(xtab)
as.matrix(results)
```

```
##          negative positive
## negative      937       77
## positive     5509     3477
```

```
accuracy <- (3477+937)/(3477+5509+77+937)
precision <- 937/(937+77)
recall <- 937/(937+5509)

cat(" ", paste0("accuracy: ", accuracy), "\n",
    paste0("precision: ", precision), "\n",
    paste0("   recall: ", recall))
```

```
##   accuracy: 0.4414
##  precision: 0.924063116370809
##     recall: 0.145361464474092
```

(d) For both the sentiment score (SentRank) and the actual score (ActRank), generate a rank for that review, where 1 is the most positive review and 10,000 is the most negative. Compute the sum of all of the absolute differences between SentRank and ActRank for each review (see RankSum represented in Equation 1). Report your findings.

```
#store the SentRank and ActRank
df <- mutate(df, SentRank = rank(df$senti_score, ties.method = "average"))
df <- mutate(df, ActRank = rank(df$actual_score, ties.method = "average"))

RankSum <- sum(abs(df$SentRank - df$ActRank))
cat("Ranksum: ", RankSum)
```

```
## Ranksum:  27162359
```

4. Next, we'll train a Naive Bayes classifier to predict if a review is positive or negative (according to the original labels).

(a) Use the "textmodel" function in quanteda to train a smoothed Naive Bayes classifier with uniform priors, using 20% of the reviews in the training set and 80% in the test set. Report the accuracy, precision and recall of your predictions.

```
#randomly train test split
#get random 20% of samples' index
smp_size <- floor(0.2 * nrow(df))
train_ind <- sample(seq_len(nrow(df)), size = smp_size)

#store in train set and test set
train_df <- df[train_ind, ]
test_df <- df[-train_ind, ]

data_dfm <- dfm(df$text, stem=TRUE, tolower=TRUE, remove=stopwords("english"))

#train data
train_dfm <- data_dfm[train_ind, ]
label_true <- factor(train_df$actual_score) #original labels

#test data
test_dfm <- data_dfm[-train_ind, ]
test_label_true <- factor(test_df$actual_score) #original labels

#train the model
model_nb <- textmodel_nb(x=train_dfm, y=label_true, smooth=1, prior="uniform")

#predict
nb_pred <- predict(model_nb, newdata=test_dfm)
label_pred <- nb_pred$nb.predicted
conf_matix <- table(label_pred, test_label_true)
conf_matix
```

```
##           test_label_true
## label_pred negative positive
##   negative     4848     1751
##   positive      327     1074
```

```r
accuracy <- (conf_matix[1,1]+conf_matix[2,2])/(conf_matix[1,1]+conf_matix[2,2]+
                                                conf_matix[1,2]+conf_matix[2,1])
precision <- conf_matix[1,1]/(conf_matix[1,1]+conf_matix[1,2])
recall <- conf_matix[1,1]/(conf_matix[1,1]+conf_matix[2,1])

cat(" ", paste0("accuracy: ", accuracy), "\n",
    paste0("precision: ", precision), "\n",
    paste0("   recall: ", recall))
```

```
##   accuracy: 0.74025
##  precision: 0.734656766176693
##     recall: 0.936811594202899
```

(b) Were you to change the priors from "uniform" to "docfreq," would you expect this to change the accuracy of Naive Bayes predictions? Why? Re-estimate Naive Bayes with the "docfreq" prior and report the accuracy, precision, and recall of these new results.

When training classes are balanced (in this case), the accuracy will change. "When training classes are balanced in their number of documents (usually advisable), however, then the empirically computed"docfreq" would be equivalent to "uniform" priors." – R Document.

```r
#train the model
model_nb <- textmodel_nb(x=train_dfm, y=label_true, smooth=1, prior="docfreq")

#predict
nb_pred <- predict(model_nb, newdata=test_dfm)
label_pred <- nb_pred$nb.predicted
conf_matix <- table(label_pred, test_label_true)
conf_matix
```

```
##           test_label_true
## label_pred negative positive
##   negative     4907     1864
##   positive      268      961
```

```r
accuracy <- (conf_matix[1,1]+conf_matix[2,2])/(conf_matix[1,1]+conf_matix[2,2]+
                                                conf_matix[1,2]+conf_matix[2,1])
precision <- conf_matix[1,1]/(conf_matix[1,1]+conf_matix[1,2])
recall <- conf_matix[1,1]/(conf_matix[1,1]+conf_matix[2,1])

cat(" ", paste0("accuracy: ", accuracy), "\n",
    paste0("precision: ", precision), "\n",
    paste0("   recall: ", recall))
```

```
##   accuracy: 0.7335
##  precision: 0.724708314872249
##     recall: 0.948212560386473
```

(c) If you try to fit the model without smoothing, it's unable to make predictions about some of the reviews out-of-sample. Why might this be? HINT: think about how these texts differ from the Conservative manifestos we used to train the NB model on in recitation.

Some words in the test datasets might not exist in the training datasets, which will introduce 0 to the calculation.

5. Although there isn't really an "ideology" in this example, there is an underlying positive- negative latent space that we can use to train a "wordscores model." And the beautiful thing about this model is that it's simple enough to implement by hand!

(a) Create a vector of "wordscores" for the words that appear in the "anchor negative" and "anchor positive" reviews using the technique described in Laver, Benoit & Garry (2003). What are the most extreme words? Report your findings.

```
#only anchor_positive or anchor_negative
train_df <- filter(df, anchor_positive == TRUE | anchor_negative == TRUE)
#not anchor_positive or anchor_negative
test_df <- filter(df, anchor_positive == FALSE & anchor_negative == FALSE)

#train data
train_dfm <- dfm(train_df$text, stem=TRUE, tolower=TRUE, remove=stopwords("english"))

#test data
test_dfm <- dfm(test_df$text, stem=TRUE, tolower=TRUE, remove=stopwords("english"))

#train word score model
#positive=1, negative=-1
model_ws <- textmodel_wordscores(
    x=train_dfm, y=ifelse(train_df$anchor_positive==1, 1, -1), smooth=1)

#most positive words
posi_feature <- sort(model_ws$wordscores, decreasing=TRUE)
posi_feature[1:10]
```

```
##     great         !      love    delici      amaz     alway    friend
## 0.6949150 0.6881391 0.6810271 0.6556702 0.6531320 0.6471368 0.6457880
##      best   perfect     enjoy
## 0.6454163 0.6432279 0.6342887
```

```
#most negative words
nega_feature <- sort(model_ws$wordscores, decreasing=FALSE)
nega_feature[1:10]
```

```
##         "         .       ask      said      call      told         ?
## 0.4479380 0.4695042 0.5110110 0.5144733 0.5163655 0.5174967 0.5176870
##     order     minut         ,
## 0.5282789 0.5340348 0.5387128
```

(b) Apply these wordscores to the reviews, and calculate the RankSum statistic (described in Equation 1) of the reviews as scored by wordscores in the same way as you did for the dictionaries. Again, compute the absolute value of the sum of all of the differences in rank for each review. Report your findings. By this metric, which did better: dictionaries or wordscores?

```
df$ws_pred <- predict(model_ws, newdata=data_dfm)

#df$words_score <- df_senti$positive - df_senti$negative
df <- mutate(df, SentRank_ws = rank(df$ws_pred, ties.method="average"))
df <- mutate(df, ActRank_ws = rank(df$actual_score, ties.method="average"))
```

```r
RankSum <- sum(abs(df$SentRank_ws - df$ActRank_ws))
cat("Ranksum: ", RankSum)
```

```
## Ranksum:  24465707
```

Wordscores did better. Wordscores has lower RankSum, meaning closer to the actual score.

6. Now we'll attempt to do our best on the classification task using a Support Vector Machine (SVM). Since SVM functions are computationally intensive, restrict your analysis to the first 1000 reviews using the original ordering of the review data.

(a) Describe an advantage offered by SVM or Naive Bayes relative to the dictionary approach or wordscores in classifying positive and negative reviews.

SVM and Naive Bayes can provide a more detailed information about the performance of the classification, using accuracy, precision, and recall.

(b) Using the "cross validate" command, train an SVM. Your goal is to maximize out of sample accuracy with 10-fold cross-validation. Optimize over the relative size of the training and test sets—try each value from 10 to 90 (by 10s) for the test set and report which gives you the highest average accuracy.

```r
#cross validation with linear model
df_svm <- df[1:1000,]
dtm_yelp <- create_matrix(df_svm$text, stemWords = TRUE,
                          removePunctuation = FALSE, removeStopwords=TRUE)

for(i in 1:9){
  size_train <- floor(0.1*i*nrow(df_svm))
  container <- create_container(dtm_yelp, df_svm$actual_score, trainSize=1:size_train,
                                testSize=(size_train+1):nrow(df_svm), virgin=FALSE)
  cv.svm <- cross_validate(container, nfold=10, algorithm = 'SVM', kernel = 'linear')
  cat(paste0(" ","Relative test sets size: ", i*10, "%"), "\n",
      paste0("Average accuracy: ", cv.svm$meanAccuracy, "\n"),
      paste0("----------", "\n"))
}
```

```
## Fold 1 Out of Sample Accuracy = 0.755102
## Fold 2 Out of Sample Accuracy = 0.7391304
## Fold 3 Out of Sample Accuracy = 0.8282828
## Fold 4 Out of Sample Accuracy = 0.7719298
## Fold 5 Out of Sample Accuracy = 0.7938144
## Fold 6 Out of Sample Accuracy = 0.75
## Fold 7 Out of Sample Accuracy = 0.81
## Fold 8 Out of Sample Accuracy = 0.6796117
## Fold 9 Out of Sample Accuracy = 0.7857143
## Fold 10 Out of Sample Accuracy = 0.7373737
##  Relative test sets size: 10%
##  Average accuracy: 0.765095923500632
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7757009
## Fold 2 Out of Sample Accuracy = 0.7684211
## Fold 3 Out of Sample Accuracy = 0.754717
```

```
## Fold 4 Out of Sample Accuracy = 0.7272727
## Fold 5 Out of Sample Accuracy = 0.7352941
## Fold 6 Out of Sample Accuracy = 0.7956989
## Fold 7 Out of Sample Accuracy = 0.7640449
## Fold 8 Out of Sample Accuracy = 0.7938144
## Fold 9 Out of Sample Accuracy = 0.8095238
## Fold 10 Out of Sample Accuracy = 0.7708333
##  Relative test sets size: 20%
##  Average accuracy: 0.769532125766112
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7227723
## Fold 2 Out of Sample Accuracy = 0.8061224
## Fold 3 Out of Sample Accuracy = 0.6666667
## Fold 4 Out of Sample Accuracy = 0.7142857
## Fold 5 Out of Sample Accuracy = 0.7909091
## Fold 6 Out of Sample Accuracy = 0.7373737
## Fold 7 Out of Sample Accuracy = 0.7857143
## Fold 8 Out of Sample Accuracy = 0.7943925
## Fold 9 Out of Sample Accuracy = 0.7904762
## Fold 10 Out of Sample Accuracy = 0.7619048
##  Relative test sets size: 30%
##  Average accuracy: 0.757061769690225
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7674419
## Fold 2 Out of Sample Accuracy = 0.7473684
## Fold 3 Out of Sample Accuracy = 0.78
## Fold 4 Out of Sample Accuracy = 0.7333333
## Fold 5 Out of Sample Accuracy = 0.7211538
## Fold 6 Out of Sample Accuracy = 0.7232143
## Fold 7 Out of Sample Accuracy = 0.7931034
## Fold 8 Out of Sample Accuracy = 0.7314815
## Fold 9 Out of Sample Accuracy = 0.8
## Fold 10 Out of Sample Accuracy = 0.7857143
##  Relative test sets size: 40%
##  Average accuracy: 0.758281096219084
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7765957
## Fold 2 Out of Sample Accuracy = 0.7659574
## Fold 3 Out of Sample Accuracy = 0.7352941
## Fold 4 Out of Sample Accuracy = 0.755102
## Fold 5 Out of Sample Accuracy = 0.7788462
## Fold 6 Out of Sample Accuracy = 0.7692308
## Fold 7 Out of Sample Accuracy = 0.8214286
## Fold 8 Out of Sample Accuracy = 0.8055556
## Fold 9 Out of Sample Accuracy = 0.7446809
## Fold 10 Out of Sample Accuracy = 0.7087379
##  Relative test sets size: 50%
##  Average accuracy: 0.76614291151553
##  ----------
## Fold 1 Out of Sample Accuracy = 0.8
## Fold 2 Out of Sample Accuracy = 0.752381
## Fold 3 Out of Sample Accuracy = 0.7065217
## Fold 4 Out of Sample Accuracy = 0.7692308
## Fold 5 Out of Sample Accuracy = 0.7545455
```

```
## Fold 6 Out of Sample Accuracy = 0.7596154
## Fold 7 Out of Sample Accuracy = 0.79
## Fold 8 Out of Sample Accuracy = 0.7931034
## Fold 9 Out of Sample Accuracy = 0.7684211
## Fold 10 Out of Sample Accuracy = 0.7603306
##  Relative test sets size: 60%
##  Average accuracy: 0.765414937932283
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7362637
## Fold 2 Out of Sample Accuracy = 0.733945
## Fold 3 Out of Sample Accuracy = 0.7757009
## Fold 4 Out of Sample Accuracy = 0.754717
## Fold 5 Out of Sample Accuracy = 0.7403846
## Fold 6 Out of Sample Accuracy = 0.8372093
## Fold 7 Out of Sample Accuracy = 0.7608696
## Fold 8 Out of Sample Accuracy = 0.7956989
## Fold 9 Out of Sample Accuracy = 0.7606838
## Fold 10 Out of Sample Accuracy = 0.7684211
##  Relative test sets size: 70%
##  Average accuracy: 0.766389382707778
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7920792
## Fold 2 Out of Sample Accuracy = 0.7857143
## Fold 3 Out of Sample Accuracy = 0.6831683
## Fold 4 Out of Sample Accuracy = 0.7731959
## Fold 5 Out of Sample Accuracy = 0.8446602
## Fold 6 Out of Sample Accuracy = 0.6960784
## Fold 7 Out of Sample Accuracy = 0.7169811
## Fold 8 Out of Sample Accuracy = 0.773913
## Fold 9 Out of Sample Accuracy = 0.79
## Fold 10 Out of Sample Accuracy = 0.7252747
##  Relative test sets size: 80%
##  Average accuracy: 0.758106521313119
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7326733
## Fold 2 Out of Sample Accuracy = 0.7809524
## Fold 3 Out of Sample Accuracy = 0.6605505
## Fold 4 Out of Sample Accuracy = 0.7706422
## Fold 5 Out of Sample Accuracy = 0.75
## Fold 6 Out of Sample Accuracy = 0.7263158
## Fold 7 Out of Sample Accuracy = 0.7685185
## Fold 8 Out of Sample Accuracy = 0.7888889
## Fold 9 Out of Sample Accuracy = 0.8
## Fold 10 Out of Sample Accuracy = 0.7777778
##  Relative test sets size: 90%
##  Average accuracy: 0.755631928348844
##  ----------
```

When relative test sets size equals to 20%, the average accuracy is the highest (0.7695).

(c) Take a guess as to which kernel would be best to use in this context, and discuss what assumptions about the data cause you to make that choice. Try both the radial and linear kernels; were you correct?

I guess the radial kernel would be better. Because the text data are in high dimension, which might not be linear separable.

```r
#corss validation with radial kernel
for(i in 1:9){
  size_train <- floor(0.1*i*nrow(df_svm))
  container <- create_container(dtm_yelp, df_svm$actual_score, trainSize=1:size_train,
                                 testSize=(size_train+1):nrow(df_svm), virgin=FALSE)
  cv.svm <- cross_validate(container, nfold=10, algorithm = 'SVM', kernel = 'radial')
  cat(paste0(" ","Relative test sets size: ", i*10, "%"), "\n",
      paste0("Average accuracy: ", cv.svm$meanAccuracy, "\n"),
      paste0("----------", "\n"))
}
```

```
## Fold 1 Out of Sample Accuracy = 0.7586207
## Fold 2 Out of Sample Accuracy = 0.7596154
## Fold 3 Out of Sample Accuracy = 0.7706422
## Fold 4 Out of Sample Accuracy = 0.7959184
## Fold 5 Out of Sample Accuracy = 0.8068182
## Fold 6 Out of Sample Accuracy = 0.7474747
## Fold 7 Out of Sample Accuracy = 0.7864078
## Fold 8 Out of Sample Accuracy = 0.7717391
## Fold 9 Out of Sample Accuracy = 0.6947368
## Fold 10 Out of Sample Accuracy = 0.808
##  Relative test sets size: 10%
##  Average accuracy: 0.769997331227562
##  ----------
## Fold 1 Out of Sample Accuracy = 0.699115
## Fold 2 Out of Sample Accuracy = 0.7843137
## Fold 3 Out of Sample Accuracy = 0.7962963
## Fold 4 Out of Sample Accuracy = 0.7857143
## Fold 5 Out of Sample Accuracy = 0.7857143
## Fold 6 Out of Sample Accuracy = 0.8021978
## Fold 7 Out of Sample Accuracy = 0.7410714
## Fold 8 Out of Sample Accuracy = 0.8607595
## Fold 9 Out of Sample Accuracy = 0.8019802
## Fold 10 Out of Sample Accuracy = 0.7755102
##  Relative test sets size: 20%
##  Average accuracy: 0.78326727640044
##  ----------
## Fold 1 Out of Sample Accuracy = 0.816092
## Fold 2 Out of Sample Accuracy = 0.787234
## Fold 3 Out of Sample Accuracy = 0.74
## Fold 4 Out of Sample Accuracy = 0.7982456
## Fold 5 Out of Sample Accuracy = 0.7478261
## Fold 6 Out of Sample Accuracy = 0.7261905
## Fold 7 Out of Sample Accuracy = 0.8061224
## Fold 8 Out of Sample Accuracy = 0.7727273
## Fold 9 Out of Sample Accuracy = 0.7835052
## Fold 10 Out of Sample Accuracy = 0.7722772
##  Relative test sets size: 30%
##  Average accuracy: 0.775022027782708
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7380952
## Fold 2 Out of Sample Accuracy = 0.7142857
```

```
## Fold 3 Out of Sample Accuracy = 0.7524752
## Fold 4 Out of Sample Accuracy = 0.8125
## Fold 5 Out of Sample Accuracy = 0.7962963
## Fold 6 Out of Sample Accuracy = 0.7941176
## Fold 7 Out of Sample Accuracy = 0.7425743
## Fold 8 Out of Sample Accuracy = 0.7978723
## Fold 9 Out of Sample Accuracy = 0.8431373
## Fold 10 Out of Sample Accuracy = 0.7096774
##  Relative test sets size: 40%
##  Average accuracy: 0.77010314153689
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7294118
## Fold 2 Out of Sample Accuracy = 0.8072289
## Fold 3 Out of Sample Accuracy = 0.8018018
## Fold 4 Out of Sample Accuracy = 0.7634409
## Fold 5 Out of Sample Accuracy = 0.7613636
## Fold 6 Out of Sample Accuracy = 0.8190476
## Fold 7 Out of Sample Accuracy = 0.7777778
## Fold 8 Out of Sample Accuracy = 0.7669903
## Fold 9 Out of Sample Accuracy = 0.7768595
## Fold 10 Out of Sample Accuracy = 0.8035714
##  Relative test sets size: 50%
##  Average accuracy: 0.780749359954022
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7238095
## Fold 2 Out of Sample Accuracy = 0.74
## Fold 3 Out of Sample Accuracy = 0.8198198
## Fold 4 Out of Sample Accuracy = 0.8415842
## Fold 5 Out of Sample Accuracy = 0.7407407
## Fold 6 Out of Sample Accuracy = 0.7826087
## Fold 7 Out of Sample Accuracy = 0.7850467
## Fold 8 Out of Sample Accuracy = 0.7532468
## Fold 9 Out of Sample Accuracy = 0.7920792
## Fold 10 Out of Sample Accuracy = 0.7857143
##  Relative test sets size: 60%
##  Average accuracy: 0.776464991429189
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7572816
## Fold 2 Out of Sample Accuracy = 0.75
## Fold 3 Out of Sample Accuracy = 0.7352941
## Fold 4 Out of Sample Accuracy = 0.8431373
## Fold 5 Out of Sample Accuracy = 0.7663551
## Fold 6 Out of Sample Accuracy = 0.8390805
## Fold 7 Out of Sample Accuracy = 0.7522936
## Fold 8 Out of Sample Accuracy = 0.7252747
## Fold 9 Out of Sample Accuracy = 0.7669903
## Fold 10 Out of Sample Accuracy = 0.7962963
##  Relative test sets size: 70%
##  Average accuracy: 0.773200341671892
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7070707
## Fold 2 Out of Sample Accuracy = 0.7578947
## Fold 3 Out of Sample Accuracy = 0.8043478
## Fold 4 Out of Sample Accuracy = 0.8303571
```

```
## Fold 5 Out of Sample Accuracy = 0.8080808
## Fold 6 Out of Sample Accuracy = 0.83
## Fold 7 Out of Sample Accuracy = 0.75
## Fold 8 Out of Sample Accuracy = 0.7741935
## Fold 9 Out of Sample Accuracy = 0.7474747
## Fold 10 Out of Sample Accuracy = 0.7207207
##  Relative test sets size: 80%
##  Average accuracy: 0.773014023752028
##  ----------
## Fold 1 Out of Sample Accuracy = 0.7745098
## Fold 2 Out of Sample Accuracy = 0.8031496
## Fold 3 Out of Sample Accuracy = 0.8019802
## Fold 4 Out of Sample Accuracy = 0.755102
## Fold 5 Out of Sample Accuracy = 0.8170732
## Fold 6 Out of Sample Accuracy = 0.745283
## Fold 7 Out of Sample Accuracy = 0.7802198
## Fold 8 Out of Sample Accuracy = 0.7821782
## Fold 9 Out of Sample Accuracy = 0.7647059
## Fold 10 Out of Sample Accuracy = 0.7555556
##  Relative test sets size: 90%
##  Average accuracy: 0.77797572746066
##  ----------
```

The performance of radial kernel is better. The best score of radial kernel is 0.7833, while the best score of linear kernel is 0.7695.

Part 3

7. Finally, in the file "CF rate trustworthiness.csv", you have been provided the results of a Human Intelligence Task (HIT) conducted on CrowdFlower. The task was for each worker to rate pictures of politicians based on how trustworthy they appeared, from 1 to 10. In the results file, there are a number of variables of interest (credit to the creation of this data goes to Kevin Munger):

- rating: the rating assigned to the image
- image name: the name of the image, which contains the race and gender of the person pictured
- country: the nationality of the worker

The goal of the HIT was to ensure that there was a sample of images of white men, white women, and black men that were balanced in terms of their trustworthiness.

(a) Is there any nationality that is likely to give statistically significant—at a 5% level— higher than average ratings?

```r
#read in data
df_cf <- read.csv("CF_rate_trustworthiness.csv")
head(df_cf, n=3)
```

```
##    X_unit_id  X_created_at       X_id  X_started_at X_tainted X_channel
## 1 726049306 5/26/15 15:32 1643434598 5/26/15 15:32     FALSE clixsense
## 2 726049306 5/26/15 15:33 1643434627 5/26/15 15:32     FALSE clixsense
## 3 726049306 5/26/15 15:33 1643434630 5/26/15 15:32     FALSE    neodev
##    X_trust X_worker_id X_country X_region           X_ip rating image_name
```

```
## 1       1    32145508       BIH        1 185.13.242.248      4  blackman1
## 2       1    29593510       BIH        2 79.143.166.165      6  blackman1
## 3       1    30666873       ESP       58   62.32.151.38      7  blackman1
##                                                                   url
## 1 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 2 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 3 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
```

```r
gr <- levels(df_cf$X_country) #all countries
mu <- mean(df_cf[,"rating"])  #mean of all countries

resultlist <- vector()
gr_new <- vector()
signifi <- vector()

for(i in 2:length(gr)){  #the first one is empty str so pass

    gr_new <- c(gr_new, gr[i])  #store country name
    #how many samples this country has
    grouplen <- length(df_cf$rating[df_cf$X_country==gr[i]])

    if (grouplen < 2) {  #t-test only valid for sample size>1
        #in this case, do direct compare
        greater <- df_cf$rating[df_cf$X_country==gr[i]] < mu
        resultlist <- c(resultlist, greater)
        signifi <- c(signifi, greater<0.05)

    } else {
        #t-test
        temp <- t.test(df_cf$rating[df_cf$X_country==gr[i]],
                       df_cf[,"rating"], alternative="greater")
        resultlist <- c(resultlist, temp$p.value)
        signifi <- c(signifi, temp$p.value<0.05)
    }
}

cbind(gr_new, resultlist, signifi)
```

```
##       gr_new resultlist             signifi
##  [1,] "ARG"  "0.580907264540556"    "FALSE"
##  [2,] "AUT"  "0.0902662672111224"   "FALSE"
##  [3,] "BGR"  "0.00607014265886928"  "TRUE"
##  [4,] "BIH"  "0.995343909006068"    "FALSE"
##  [5,] "CAN"  "0.261917759439176"    "FALSE"
##  [6,] "CHL"  "2.07273715568737e-72" "TRUE"
##  [7,] "CHN"  "0.993252883438418"    "FALSE"
##  [8,] "DEU"  "0.0147379202059379"   "TRUE"
##  [9,] "EGY"  "0.985297189361608"    "FALSE"
## [10,] "ESP"  "0.669135846788521"    "FALSE"
## [11,] "FIN"  "1"                    "FALSE"
## [12,] "GBR"  "0.00803544546042112"  "TRUE"
## [13,] "GRC"  "0.837465220769936"    "FALSE"
## [14,] "HKG"  "0.0269211414095008"   "TRUE"
## [15,] "HRV"  "1"                    "FALSE"
## [16,] "IDN"  "0.687538031527664"    "FALSE"
```

```
## [17,] "IND"  "0.276667008570233"    "FALSE"
## [18,] "ITA"  "0"                    "TRUE"
## [19,] "JOR"  "0.0955575518704469"   "FALSE"
## [20,] "JPN"  "0.000941829156451328" "TRUE"
## [21,] "MEX"  "0.854544394789288"    "FALSE"
## [22,] "MYS"  "0.9999999999532"      "FALSE"
## [23,] "PAK"  "0.661312619498986"    "FALSE"
## [24,] "PHL"  "0.046310098486831"    "TRUE"
## [25,] "POL"  "0.983814756562153"    "FALSE"
## [26,] "PRT"  "0.434771355913353"    "FALSE"
## [27,] "ROU"  "0"                    "TRUE"
## [28,] "RUS"  "0.0907777478126093"   "FALSE"
## [29,] "SRB"  "0.825555034652184"    "FALSE"
## [30,] "SVK"  "0.855453172109754"    "FALSE"
## [31,] "TUR"  "0.0892222646393836"   "FALSE"
## [32,] "UKR"  "0.512158578127833"    "FALSE"
## [33,] "USA"  "0.0368038522799616"   "TRUE"
## [34,] "VEN"  "0.556862253837715"    "FALSE"
```

For country that has column 'signifi' being TRUE, the rating is significant higher than average under the significance of 0.05.

(b) Of the three demographic groups in the picture, is there a statistically significant difference between the average ratings given to them?

```r
#get the demographic information from image name
df_cf$demo <- as.factor(gsub("*[0-9]", "", df_cf$image_name))
head(df_cf, head=3)
```

```
##    X_unit_id  X_created_at       X_id  X_started_at X_tainted X_channel
## 1 726049306 5/26/15 15:32 1643434598 5/26/15 15:32     FALSE clixsense
## 2 726049306 5/26/15 15:33 1643434627 5/26/15 15:32     FALSE clixsense
## 3 726049306 5/26/15 15:33 1643434630 5/26/15 15:32     FALSE    neodev
## 4 726049306 5/26/15 15:33 1643434693 5/26/15 15:32     FALSE    neodev
## 5 726049306 5/26/15 15:32 1643434614 5/26/15 15:32     FALSE    neodev
## 6 726049307 5/26/15 15:32 1643434480 5/26/15 15:32     FALSE    neodev
##   X_trust X_worker_id X_country X_region           X_ip rating image_name
## 1       1    32145508       BIH        1 185.13.242.248      4  blackman1
## 2       1    29593510       BIH        2 79.143.166.165      6  blackman1
## 3       1    30666873       ESP       58   62.32.151.38      7  blackman1
## 4       1    28513920       JOR        2  212.34.12.190      7  blackman1
## 5       1    30448360       POL       73  31.61.136.172      1  blackman1
## 6       1    27667288       CAN       MB 216.36.187.168      8  blackman2
##                                                                    url
## 1 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 2 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 3 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 4 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 5 https://raw.githubusercontent.com/kmunger/images/master/blackman1.jpg
## 6 https://raw.githubusercontent.com/kmunger/images/master/blackman2.jpg
##       demo
## 1 blackman
## 2 blackman
## 3 blackman
## 4 blackman
```

```
## 5 blackman
## 6 blackman
```

```r
#here I will use the ANOVA instead of three t-test
#as there is an increasing risk of type 1 error for t-test here
#https://stats.stackexchange.com/questions/90760/
#example-where-comparison-of-three-mean-anova-and-t-test-have-different-results
summary(aov(rating ~ demo, data = df_cf))
```

```
##              Df Sum Sq Mean Sq F value Pr(>F)
## demo          2   26.8  13.404   2.817  0.062 .
## Residuals   219 1042.2   4.759
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As p-value=0.062, under the significance of 0.05, the differences are not signicant between the average ratings given to them.

(c) Likewise, do you observe a statistically significant difference in reported trustworthiness that is associated with gender?

```r
#get the gender information from demographic column
df_cf$gender <- grepl("*woman$", df_cf$demo)

t.test(df_cf$rating[df_cf$gender==FALSE],df_cf$rating[df_cf$gender==TRUE])
```

```
##
##  Welch Two Sample t-test
##
## data:  df_cf$rating[df_cf$gender == FALSE] and df_cf$rating[df_cf$gender == TRUE]
## t = -1.4066, df = 137.1, p-value = 0.1618
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.090587  0.183954
## sample estimates:
## mean of x mean of y
##  5.849315  6.302632
```

As p-value=0.1618, under the significance of 0.05, the differences are not signicant.