TextMining

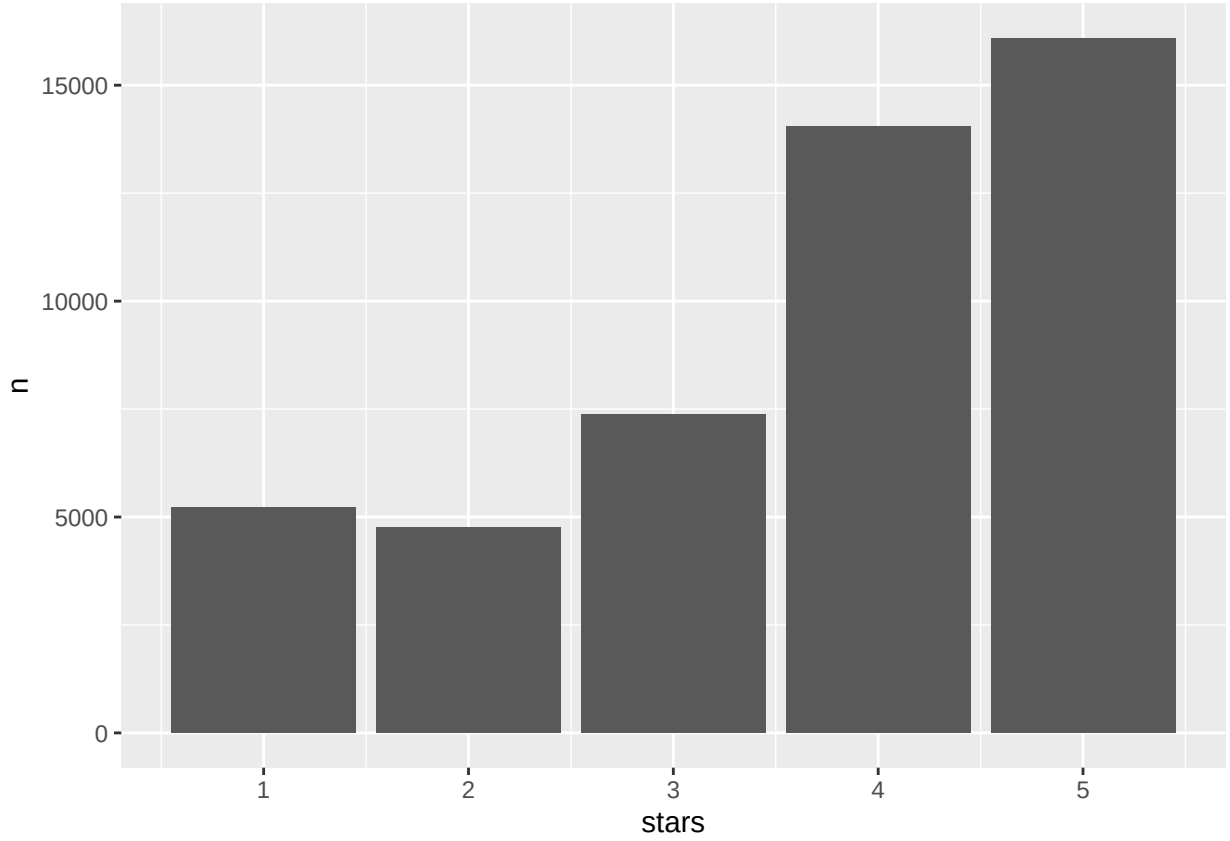
Omar Elhayboubi, Eduardo Herrera, Antonio Cruz

(a) Explore the data.

In this section, we will explore the data in coming from Yelp reviews.

Lets look at how star ratings are distributed.

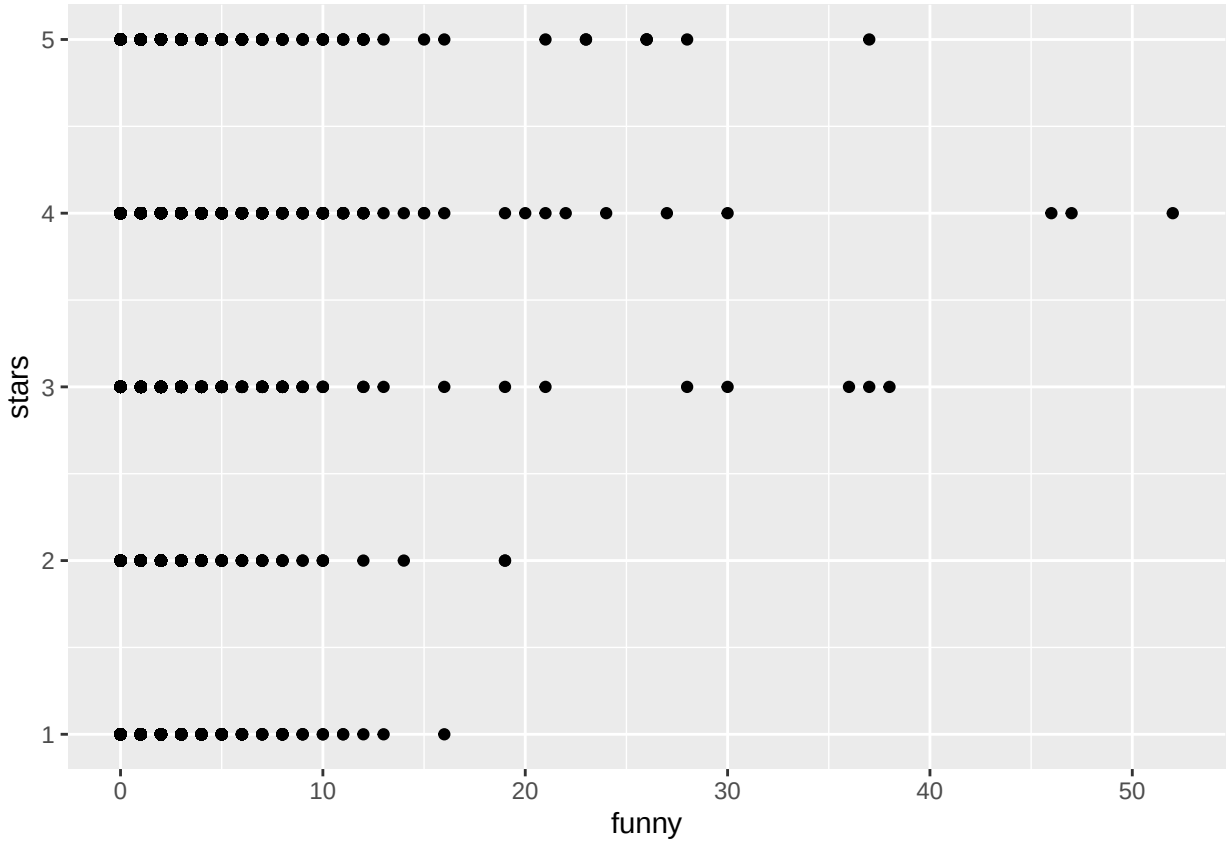
```
#read the Yelp ratings CSV file
resReviewsData <- read.csv2("yelpRestaurantReviews_sample.csv")
#number of reviews by star rating
star_dist <- resReviewsData %>% group_by(stars) %>% count()
#plot the distribution
ggplot(star_dist, aes(x= stars, y=n)) +geom_bar(stat="identity")
```



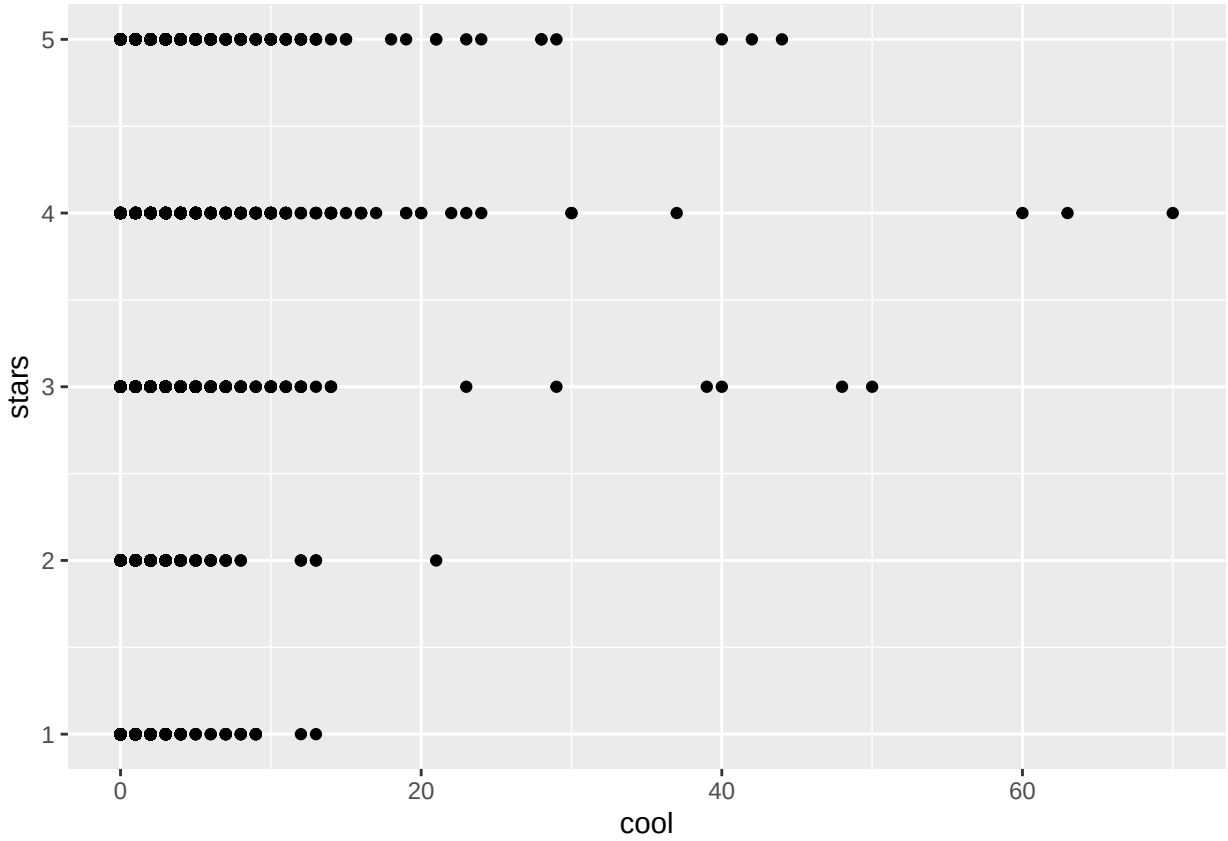
We could see the distribution of star ratings in the plot above, we could see that as star ratings increase, number of stars increases as well. In order to obtain a label indicating 'positive' or 'negative' using star ratings, we will label reviews that have star ratings < 3 as negative and the reviews that have star ratings > 3 as positive.

Now lets analyze the reviews that were labeled as 'funny', 'cool', 'useful' and see how they relate to the star ratings.

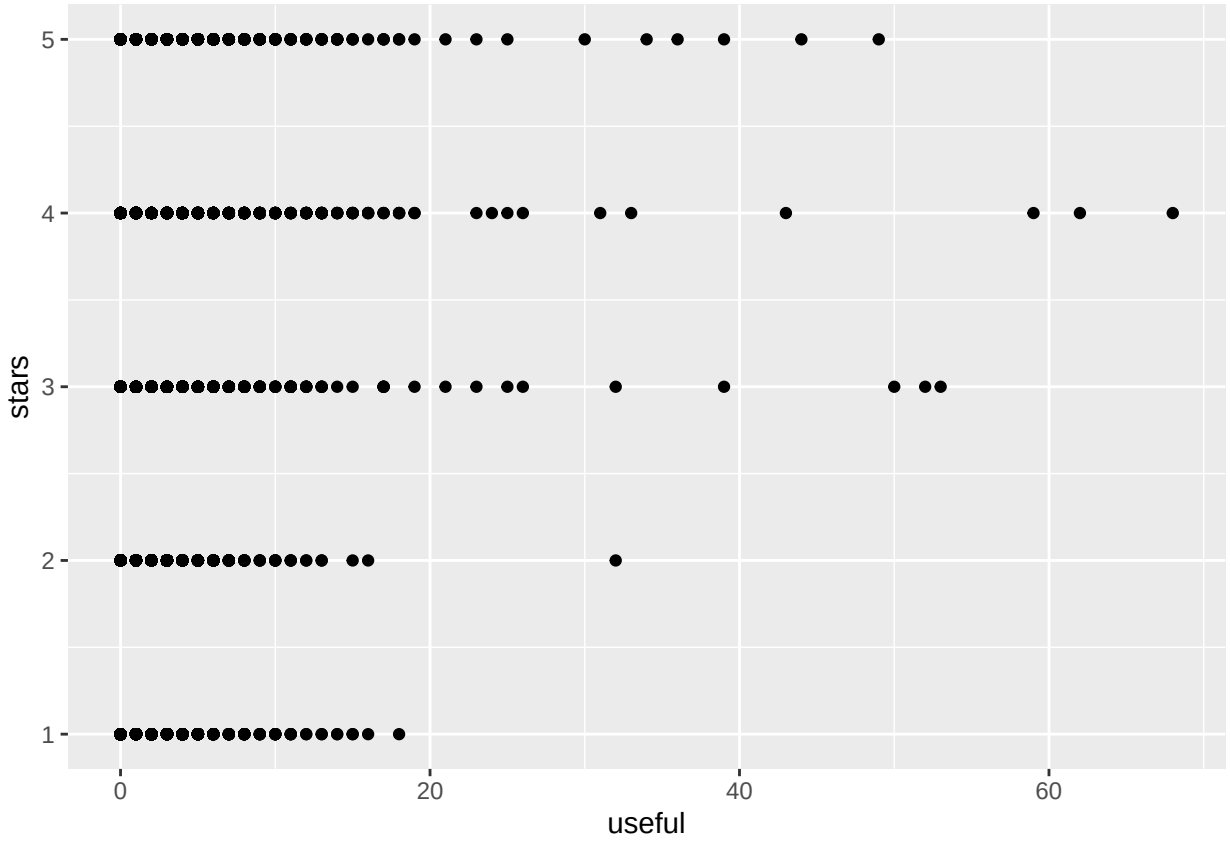
```
#funny
ggplot(resReviewsData, aes(x= funny, y=stars)) +geom_point()
```




```
#cool
ggplot(resReviewsData, aes(x= cool, y=stars)) +geom_point()
```




```
#Useful
ggplot(resReviewsData, aes(x= useful, y=stars)) +geom_point()
```



In the graphs above, we see that the reviews that were labeled as 'funny', 'cool', 'useful' are more present in the higher rated reviews but not as much as we expected. from these graphs I could conclude that if a review was labled as cool or useful for example, this does not necessarily mean that this review was positive.

We could also explore how the reviews are distributed by location.

```
resReviewsData %>% group_by(state) %>% tally() %>% view()
```

Most of the reviews are from the US, but we do see some reviews are coming from outside the US.

We could use the code below to only keep the reviews from the US

```
rrData <- resReviewsData %>% filter(str_detect(postal_code, "^[0-9]{1,5}"))
rrData %>% group_by(state) %>% tally() %>% view()
```

(b) What are some words indicative of positive and negative sentiment?

In this section, we will explore if the usage of some words indicates positive and negative sentiments.

Some Data Preperation

In order to do this, we will first tokenize the text of the reviews in the column named text. We will only keep the reviewID and star attributes. We will also remove the stop_words.

```
#tokenize the text of the reviews in the column named 'text'
rrTokens <- rrData %>% unnest_tokens(word, text)

#only keep reviewID stars and text
rrTokens <- rrData %>% select(review_id, stars, text ) %>% unnest_tokens(word, text)

#remove stopwords
rrTokens <- rrTokens %>% anti_join(stop_words)

dim(rrTokens)
```

```
## [1] 1318836       3
```

```
#display the results
head(rrTokens)
```

```
##                review_id stars       word
## 1 -K5z7DzXHJgEC1tsTLfFeA     3     dinner
## 2 -K5z7DzXHJgEC1tsTLfFeA     3  celebrate
## 3 -K5z7DzXHJgEC1tsTLfFeA     3    friends
## 4 -K5z7DzXHJgEC1tsTLfFeA     3   birthday
## 5 -K5z7DzXHJgEC1tsTLfFeA     3 restaurant
## 6 -K5z7DzXHJgEC1tsTLfFeA     3  beautiful
```

We could see from the results above that after we tokenized the text of the reviews, now we are able to look at each word with its star rating which will help us analyze if some words indicate postive or negative sentiments.

Now, we will plot the most frequent words by star ratings and analyze the results

```
#Most frequent words
rrTokens %>% count(word, sort=TRUE) %>% top_n(10)
```

```
##          word     n
## 1        food 25239
## 2     service 12379
## 3        time  9418
## 4     chicken  7700
## 5  restaurant  7244
## 6        nice  5969
## 7       pizza  5800
## 8   delicious  5571
## 9        love  5526
## 10       menu  5352
```

In the table above, we could see the top10 most frequently used words in the reviews.

Now lets look at the most rare words that are used in a very small number of reviews and remove them.

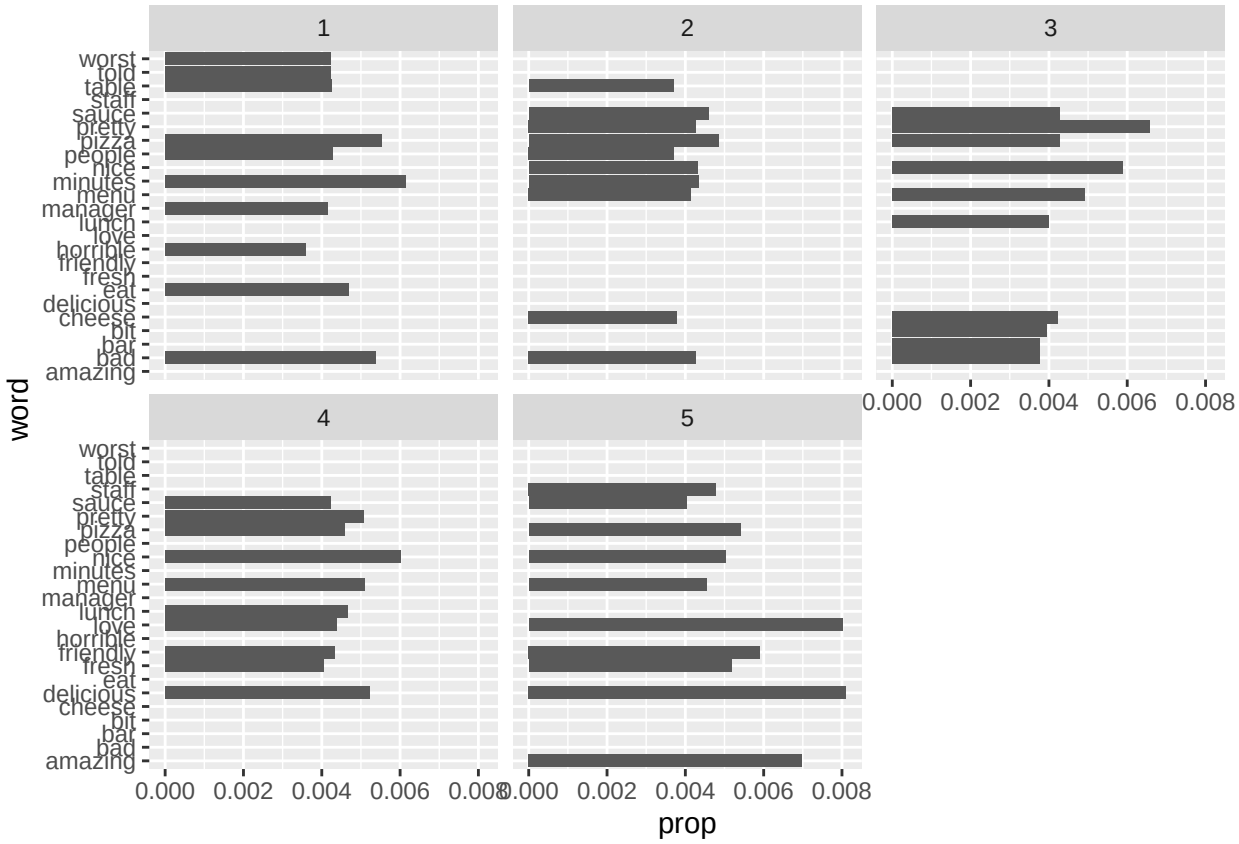
```
#Look for words used less than 10 times
rareWords <-rrTokens %>% count(word, sort=TRUE) %>% filter(n<10)
#remove these rare words
```

```
xx<-anti_join(rrTokens, rareWords)

#filter by rare words
xx2<- xx %>% filter(str_detect(word,"[0-9]")==FALSE)
#put filtered data back into rrTokens
rrTokens<- xx2
```

Analyze words/Sentiments Based on Star Ratings

Now after some data preparation, we could analyze words and how they indicate wether a review is positive or negative

```
#proportion of word occurrence by star ratings
ws <- rrTokens %>% group_by(stars) %>% count(word, sort=TRUE)
ws<-  ws %>% group_by(stars) %>% mutate(prop=n/sum(n))

#To plot this
ws %>% filter(! word %in% c('food', 'time', 'restaurant', 'service', 'chicken'))%>%
  group_by(stars) %>% arrange(stars, desc(prop)) %>%
  filter(row_number() <=10) %>%
  ggplot(aes(word, prop))+geom_col()+coord_flip()+facet_wrap((~stars))
```



In the plot above, I remove some words like food, time, restaurant, service as they are very frequent don't really help in our use case.

The plot above makes sense. We see that words like worst and bad are only found in the lower star ratings, while words like amazing, delicious, pretty are found in the higher star ratings. We do have some words that are found in all star ratings like pizza, minutes as these words could be used in both positive and negative review and doesn't make sense to use these as way to help us determine positive/negative sentiments

Now in order to get sense of which words are related to higher/lower ratings ('positive', 'negative' sentiments), we will calculate the average star rating associated with each word sum(stars*prop). Then we could look at the 20 words with highest ratings and the top 20 words with lowest ratings.

```
#calculate
xx<- ws %>% group_by(word) %>% summarise(totWS=sum(stars*prop))
#top 20 words with highest rating and top 20 words with lowest ratings
xx %>% top_n(20)
```

```
## # A tibble: 20 x 2
##    word        totWS
##    <chr>       <dbl>
##  1 amazing    0.0512
##  2 cheese     0.0583
##  3 chicken    0.0985
##  4 delicious  0.0716
##  5 eat        0.0555
##  6 food       0.321
##  7 fresh      0.0551
##  8 friendly   0.0612
##  9 love       0.0714
## 10 lunch      0.0571
## 11 menu       0.0690
## 12 nice       0.0776
## 13 pizza      0.0734
## 14 pretty     0.0609
## 15 restaurant 0.0924
## 16 salad      0.0476
## 17 sauce      0.0622
## 18 service    0.157
## 19 staff      0.0523
## 20 time       0.119
```

```
xx %>% top_n(-20)
```

```
## # A tibble: 21 x 2
##    word              totWS
##    <chr>             <dbl>
##  1 bullshit      0.000103
##  2 coffe         0.000106
##  3 disgust       0.0000948
##  4 disrespectful 0.0000948
##  5 dominoes      0.000107
##  6 evidently     0.000110
##  7 fax           0.000105
##  8 neven         0.000104
##  9 nnwas         0.000108
## 10 patronizing   0.000103
## # ... with 11 more rows
```

A lot of the words in the results above do make sense, like amazing, delicious, friendly, love for the higher ratings. However, some of the words here I did not expect like food, lunch, chicken (these words could be used in both high and lower star ratings). We see a mix of words that we expected to see and others that do not make sense. Same thing on the lower ratings, some words make sense like (bullshit, disgust, disrespectful, patronizing) bit many words don't like fax, coffee, vehicle, triangle.

Now, we will try using TF-IDF (term-frequency, inverse document frequency) which is a measure that will help us see how important a words is to a document in a collection of documents. Calculation: n times a word appears in a doc, inverse document frequency of the word across a set of documents. we will also lemmitize the rrtoken the words in order to get the original form of each word using the lemmatization algorithm.

```
#lemmatizing words in rrtokens
rrTokens<-rrTokens %>%  mutate(word = textstem::lemmatize_words(word))
#filtering by words with character <=3 or <=15
rrTokens<-rrTokens %>% filter(str_length(word)<=3 | str_length(word)<=15)
#Count of words in each review in column n
rrTokens<- rrTokens %>% group_by(review_id, stars) %>% count(word)
#function to calculate TF-IDF
rrTokens<-rrTokens %>% bind_tf_idf(word, review_id, n)
head(rrTokens)
```

```
## # A tibble: 6 x 7
## # Groups:   review_id, stars [1]
##   review_id              stars word        n     tf   idf tf_idf
##   <chr>                  <int> <chr>   <int>  <dbl> <dbl>  <dbl>
## 1 --9qM_dRW4rrKTWO_SX_qQ     1 buffet      1 0.0909 4.08  0.371
## 2 --9qM_dRW4rrKTWO_SX_qQ     1 copper      1 0.0909 6.89  0.626
## 3 --9qM_dRW4rrKTWO_SX_qQ     1 food        1 0.0909 0.726 0.0660
## 4 --9qM_dRW4rrKTWO_SX_qQ     1 kettle      1 0.0909 7.38  0.671
## 5 --9qM_dRW4rrKTWO_SX_qQ     1 price       1 0.0909 1.92  0.174
## 6 --9qM_dRW4rrKTWO_SX_qQ     1 service     1 0.0909 1.19  0.108
```

(c) Working with Dictionaries (AFINN, NRC, BING)

In this section, we will use 3 sentiment dictionaries BING, NRC and AFINN to get sentiment analysis of the words used in each reviews. All 3 dictionaries following a different approach on how to label positive/negative. We will first start with the BING dictionary

Bing Dictionary

We will get the sentiments for all words in the dictionary, then perform an inner_join with rr_tokens to avoid the words that will get an NA value. The bing dictionary results in either 'Positive' or 'Negative' sentiment for each word. Some words will give us NA results, so we will use an inner join to avoid the na values.

```
#Get sentiments inner join rrTokens
rrSenti_bing<- rrTokens %>% inner_join(get_sentiments("bing"), by="word")

#Make positive count for 'positive words' and negative count for negative words
xx<-rrSenti_bing %>% group_by(word, sentiment) %>% summarise(totOcc=sum(n)) %>%
  arrange(sentiment, desc(totOcc))
```

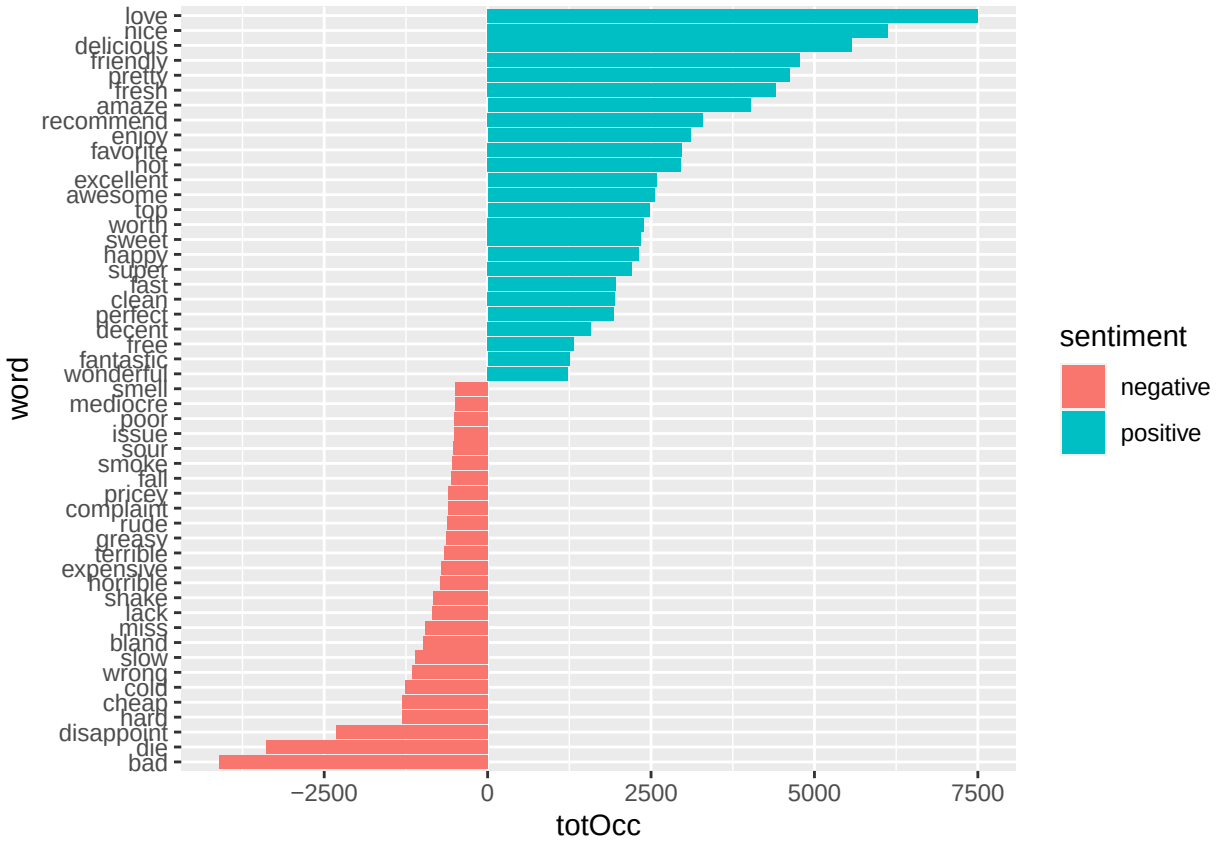
```r
xx<- xx %>% mutate (totOcc=ifelse(sentiment=="positive", totOcc, -totOcc))

#the most positive and most negative words
xx<-ungroup(xx)
xx %>% top_n(25)
```

```
## # A tibble: 25 x 3
##    word      sentiment totOcc
##    <chr>     <chr>      <int>
##  1 love      positive    7505
##  2 nice      positive    6126
##  3 delicious positive    5571
##  4 friendly  positive    4775
##  5 pretty    positive    4623
##  6 fresh     positive    4408
##  7 amaze     positive    4020
##  8 recommend positive    3293
##  9 enjoy     positive    3103
## 10 favorite  positive    2975
## # ... with 15 more rows
```

```r
xx %>% top_n(-25)
```

```
## # A tibble: 26 x 3
##    word       sentiment totOcc
##    <chr>      <chr>      <int>
##  1 bad        negative   -4106
##  2 die        negative   -3393
##  3 disappoint negative   -2311
##  4 hard       negative   -1307
##  5 cheap      negative   -1306
##  6 cold       negative   -1265
##  7 wrong      negative   -1149
##  8 slow       negative   -1114
##  9 bland      negative    -989
## 10 miss       negative    -955
## # ... with 16 more rows
```

```r
#plot the results
rbind(top_n(xx, 25), top_n(xx, -25)) %>% mutate(word=reorder(word,totOcc)) %>%
  ggplot(aes(word, totOcc, fill=sentiment)) +geom_col()+coord_flip()
```



In the graph above, we could see the most frequent 'Positive words' and most frequent 'negative words' according to the Bing dictionary. These results make total sense to us. For example the most used negative word is bad and the most used positive word is love.

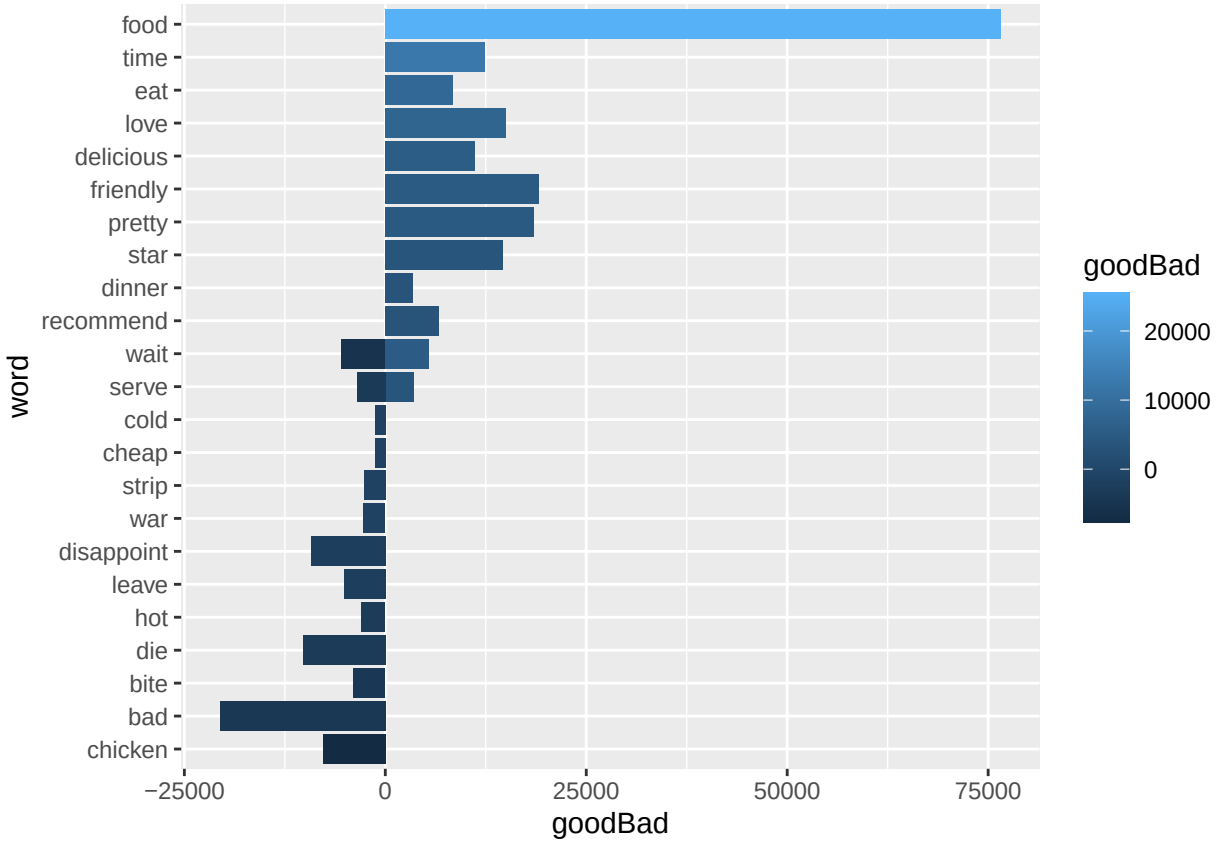
NRC Dictionary

Now lets see how NRC will label sentiments to the words in the reviews. NRC works differently than BING; It assigns different emotions to words: anger, anticipation, disgust, fear, joy, sadness, surprise, trust and sentiments 'positive', 'negative'. We will again get all the sentiments from this dictionary, perform an inner join with rrtokens and finally, we will try to convert the emotions to either negative or postive sentiments as follows:

- Negative: 'anger', 'disgust', 'fear', 'sadness', 'negative'
- Positive: 'positive', 'joy', 'anticipation', 'trust'

```
#with "nrc" dictionary
rrSenti_nrc<-rrTokens %>% inner_join(get_sentiments("nrc"), by="word") %>%
  group_by (word, sentiment) %>% summarise(totOcc=sum(n)) %>%
  arrange(sentiment, desc(totOcc))
#split into only Positive and Negative sentiments
xx<-rrSenti_nrc %>%
  mutate(goodBad=ifelse
         (sentiment %in% c('anger', 'disgust', 'fear', 'sadness', 'negative'),
           -totOcc, ifelse(sentiment %in% c('positive', 'joy', 'anticipation',
                                            'trust'), totOcc, 0)))
```

```
#ungroup
xx<-ungroup(xx)
#Plot the results
rbind(top_n(xx, 25), top_n(xx, -25)) %>%
  mutate(word=reorder(word,goodBad)) %>%
  ggplot(aes(word, goodBad, fill=goodBad)) +geom_col()+coord_flip()
```



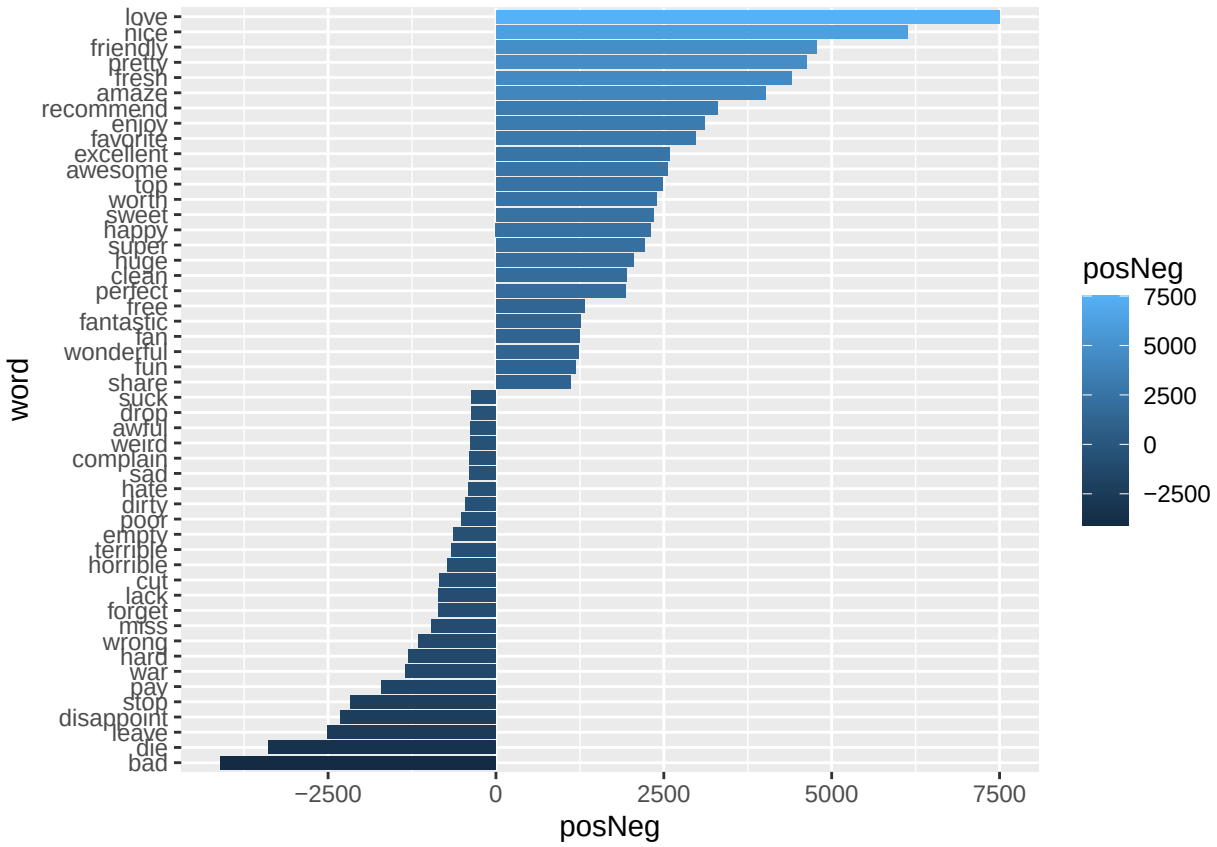
A lot of words here make sense like bad being the most frequent bad word; However, we do see some words that don't make total sense like Chicken, food, time; These words cold be used in both good and bad. Generally these results make sense but still are different than bing results. The Bing results make more sense to us.

AFINN Dictionary

Now, we will use the AFINN dictionary. Afinn assigns an integer between -5 for negative and +5 for positive. What we could is change all the negative words to 'Negative' and all positive words to 'positive' in order to analyze the results in the same manner we did with the two other dictionaries.

```
#with "afinn" dictionary
rrSenti_afinn<-rrTokens %>% inner_join(get_sentiments("afinn"), by="word") %>%
  group_by (word, value) %>% summarise(totOcc=sum(n)) %>%
  arrange(value, desc(totOcc))
#split into only Positive and Negative sentiments
xx<-rrSenti_afinn %>%
  mutate(posNeg=ifelse
```

```
        (value < 0,
          -totOcc, ifelse(value > 0, totOcc, 0)))
#ungroup
xx<-ungroup(xx)
#Plot the results
rbind(top_n(xx, 25), top_n(xx, -25)) %>%
  mutate(word=reorder(word,posNeg)) %>%
  ggplot(aes(word, posNeg, fill=posNeg)) +geom_col()+coord_flip()
```



As you can see in the plot above, these results from Afinn make total sense and are very similar to the bing results.

To conclude, we see that each sentiment dictionary uses different methods to assign a sentiment to each word. We do see similarities between bing and afinn but not as much with NRC. Which makes sense because NRC use emotions and not just sentiments.

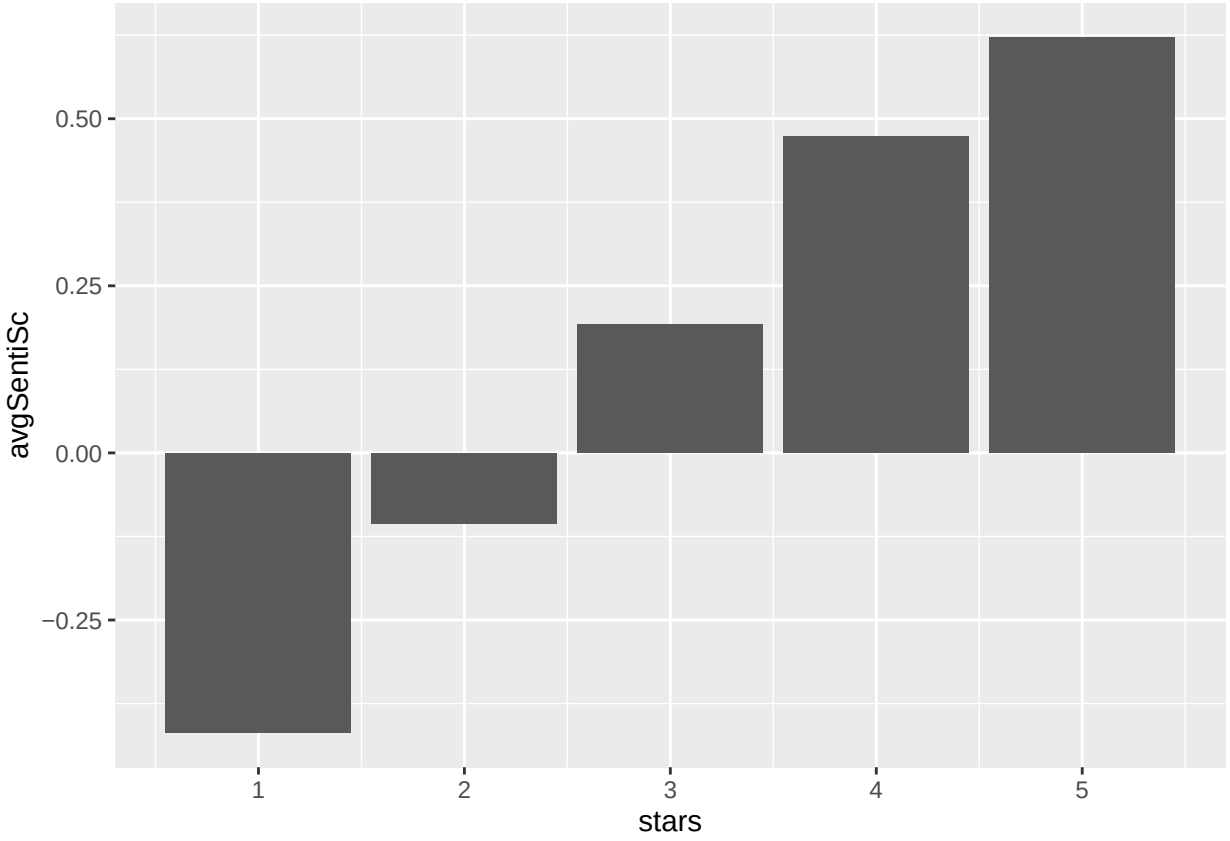
Based on these results, We will use the afinn and bing dictionary going forward.

Perform analysis by review Ratings/Word Sentiment

In the previews section, we analyzed the sentiments for words based on the dictionaries. Now, we want to see how these sentiments relate to reviews (each review id) and it's star rating. This will give a good idea on how on how these dictionaries perform in determining whether a review is positive or negative. For Bing, We could do this by calculating the number of positive words and negative words in each reviews, they divide them by the number of words in the review. Then, we could get the average sentiment score based on these proportions of pos and neg in each review.

Lets start with the bing dictionary

```
revSenti_bing <- rrSenti_bing %>%
  group_by(review_id, stars) %>%
  summarise(nwords=n(),posSum=sum(sentiment=='positive'),
            negSum=sum(sentiment=='negative'))

revSenti_bing<- revSenti_bing %>%
  mutate(posProp=posSum/nwords, negProp=negSum/nwords)
revSenti_bing<- revSenti_bing %>%
  mutate(sentiScore=posProp-negProp)

#Do review start ratings correspond to the the positive/negative sentiment words
revSenti_bing %>% group_by(stars) %>%
  summarise(avgPos=mean(posProp), avgNeg=mean(negProp), avgSentiSc=mean(sentiScore)) %>%
  ggplot (aes(x= stars, y=avgSentiSc)) +geom_bar(stat="identity")
```

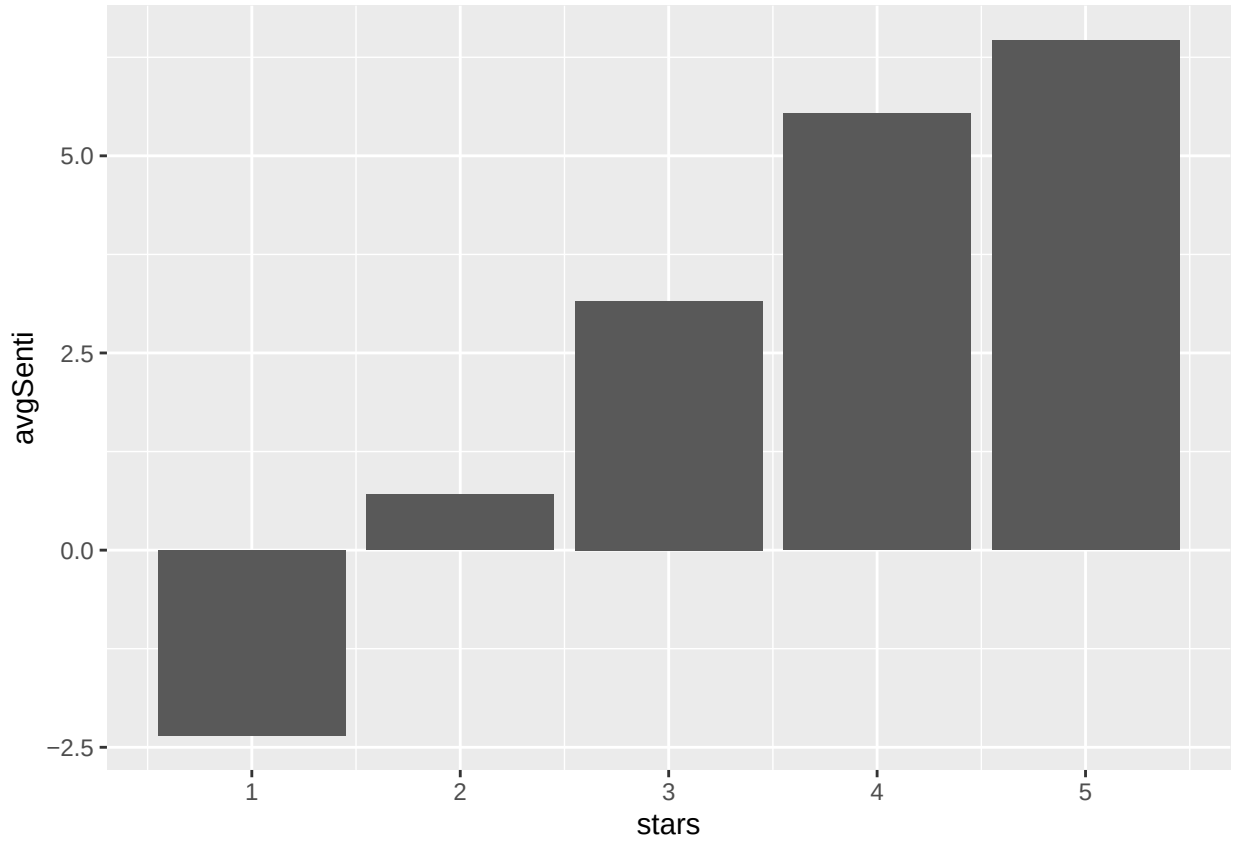


This makes total sense, as we could see in the table above, as the star ratings go up, the average sentiment score goes up as well, which means the sentiments determined by the dictionary do match the start ratings for the reviews.

Now lets do the same for the AFINN dictionary

```
rrSenti_afinn<- rrTokens %>% inner_join(get_sentiments("afinn"), by="word")

revSenti_afinn <- rrSenti_afinn %>%
```

```
  group_by(review_id, stars) %>%
  summarise(nwords=n(), sentiSum =sum(value))

revSenti_afinn %>%
  group_by(stars) %>%
  summarise(avgLen=mean(nwords), avgSenti=mean(sentiSum)) %>%
  ggplot (aes(x= stars, y=avgSenti)) +geom_bar(stat="identity")
```



Same thing with the afinn dictionary, the results make total sense as the star ratings go up, the average sentiment score goes up which means the sentiments determined by the dictionary do match the start ratings for the reviews in terms of neg/pos reviews. (taking into consideration that higher star ratings are positive and lower star ratings are negative as discussed in the first section of the assignment)

Predicting high/low reviews based on on aggregated sentiment of words in the reviews

Now, we cold predict if the reviews will be high or low based on the dictionaries without actually building a model. In order to do this, we will consider reviews with start ratings <2 as negative and reviews with star ratings > 4 as positive. Then, if the sentiment is >0 then we will label the pred_hilo as 1 else, it it is <0 will label it as -1. Finally, we will be able to print out our confusion matrix with the predicted and actual and see how each dictionary performs.

Lets start with afinn dictionary

```
#we can consider reviews with 1 to 2 stars as positive, and this with 4 to 5 stars as negative
revSenti_afinn <- revSenti_afinn %>% mutate(hiLo=ifelse(stars<=2,-1, ifelse(stars>=4, 1, 0 )))
```

```
revSenti_afinn <- revSenti_afinn %>% mutate(pred_hiLo=ifelse(sentiSum >0, 1, -1))
#filter out the reviews with 3 stars, and get the confusion matrix for hiLo vs pred_hiLo
xx<-revSenti_afinn %>% filter(hiLo!=0)
table(actual=xx$hiLo, predicted=xx$pred_hiLo )
```

```
##       predicted
## actual    -1     1
##     -1  4422  2475
##     1   2686 18832
```

Accuracy from afinn dictionary is: 81.78% which good.

Now lets see how bing dictionary will perform

```
revSenti_bing <- revSenti_bing %>% mutate(hiLo=ifelse(stars<=2,-1, ifelse(stars>=4, 1, 0 )))
revSenti_bing <- revSenti_bing %>% mutate(pred_hiLo=ifelse(sentiScore >0, 1, -1))
#filter out the reviews with 3 stars, and get the confusion matrix for hiLo vs pred_hiLo
xx<-revSenti_bing %>% filter(hiLo!=0)
table(actual=xx$hiLo, predicted=xx$pred_hiLo )
```

```
##       predicted
## actual    -1     1
##     -1  5383  1654
##     1   3758 18210
```

Accuracy from the bing dictionary is: 81.30% which is good but slightly lower than afinn.

Afinn dictionary gave us the best performance.

(d) Develop models to predict review sentiment.

In this section, we will develop models to predict review sentiment. We will use three different models on each dictionary. (random forest, SVM and Naive bayes)

Random Forest model Using Bing

In the code below, we will develop a random forest model to predict HiLo using the bing dictionary. We had to split the data into 50% train and 50% test just because the data is very large

```
#Or, since we want to keep the stars column
revDTM_sentiBing <- rrSenti_bing %>%  pivot_wider(id_cols = c(review_id,stars), names_from = word, valu

#filter out the reviews with stars=3, and calculate hiLo sentiment 'class'
revDTM_sentiBing <- revDTM_sentiBing %>% filter(stars!=3) %>% mutate(hiLo=ifelse(stars<=2, -1, 1)) %>% s

#how many review with 1, -1  'class'
revDTM_sentiBing %>% group_by(hiLo) %>% tally()
```

```
## # A tibble: 2 x 2
##    hiLo     n
## * <dbl> <int>
## 1    -1  7037
## 2     1 21968
```

```r
#develop a random forest model to predict hiLo from the words in the reviews

library(ranger)

#replace all the NAs with 0
revDTM_sentiBing<-revDTM_sentiBing %>% replace(., is.na(.), 0)

revDTM_sentiBing$hiLo<- as.factor(revDTM_sentiBing$hiLo)


library(rsample)
revDTM_sentiBing_split<- initial_split(revDTM_sentiBing, 0.5)
revDTM_sentiBing_trn<- training(revDTM_sentiBing_split)
revDTM_sentiBing_tst<- testing(revDTM_sentiBing_split)

rfModel1<-ranger(dependent.variable.name = "hiLo", data=revDTM_sentiBing_trn %>% select(-review_id), nu
```

```
## Growing trees.. Progress: 46%. Estimated remaining time: 36 seconds.
## Growing trees.. Progress: 93%. Estimated remaining time: 4 seconds.
## Computing permutation importance.. Progress: 3%. Estimated remaining time: 18 minutes, 28 seconds.
## Computing permutation importance.. Progress: 8%. Estimated remaining time: 14 minutes, 44 seconds.
## Computing permutation importance.. Progress: 12%. Estimated remaining time: 12 minutes, 57 seconds.
## Computing permutation importance.. Progress: 16%. Estimated remaining time: 12 minutes, 30 seconds.
## Computing permutation importance.. Progress: 20%. Estimated remaining time: 11 minutes, 51 seconds.
## Computing permutation importance.. Progress: 24%. Estimated remaining time: 11 minutes, 4 seconds.
## Computing permutation importance.. Progress: 28%. Estimated remaining time: 10 minutes, 33 seconds.
## Computing permutation importance.. Progress: 31%. Estimated remaining time: 10 minutes, 13 seconds.
## Computing permutation importance.. Progress: 36%. Estimated remaining time: 9 minutes, 21 seconds.
## Computing permutation importance.. Progress: 39%. Estimated remaining time: 8 minutes, 56 seconds.
## Computing permutation importance.. Progress: 43%. Estimated remaining time: 8 minutes, 20 seconds.
## Computing permutation importance.. Progress: 47%. Estimated remaining time: 7 minutes, 40 seconds.
## Computing permutation importance.. Progress: 51%. Estimated remaining time: 7 minutes, 9 seconds.
## Computing permutation importance.. Progress: 55%. Estimated remaining time: 6 minutes, 35 seconds.
## Computing permutation importance.. Progress: 59%. Estimated remaining time: 5 minutes, 58 seconds.
## Computing permutation importance.. Progress: 63%. Estimated remaining time: 5 minutes, 26 seconds.
## Computing permutation importance.. Progress: 66%. Estimated remaining time: 4 minutes, 52 seconds.
## Computing permutation importance.. Progress: 71%. Estimated remaining time: 4 minutes, 15 seconds.
## Computing permutation importance.. Progress: 74%. Estimated remaining time: 3 minutes, 43 seconds.
## Computing permutation importance.. Progress: 78%. Estimated remaining time: 3 minutes, 9 seconds.
## Computing permutation importance.. Progress: 82%. Estimated remaining time: 2 minutes, 33 seconds.
## Computing permutation importance.. Progress: 86%. Estimated remaining time: 2 minutes, 1 seconds.
## Computing permutation importance.. Progress: 90%. Estimated remaining time: 1 minute, 26 seconds.
## Computing permutation importance.. Progress: 94%. Estimated remaining time: 55 seconds.
## Computing permutation importance.. Progress: 97%. Estimated remaining time: 22 seconds.
## Computing permutation importance.. Progress: 100%. Estimated remaining time: 0 seconds.
```

```r
rfModel1
```

```
## Ranger result
##
## Call:
##  ranger(dependent.variable.name = "hiLo", data = revDTM_sentiBing_trn %>%      select(-review_id), nu
##
```

```
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      14503
## Number of independent variables:  960
## Mtry:                             30
## Target node size:                 10
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.08919068
```

Now that we trained the model, lets look at the performance. First, we will look at the confusion matrix and then compute the accuracy for both train set and test set.

```
revSentiBing_predTrn<- predict(rfModel1, revDTM_sentiBing_trn %>% select(-review_id))$predictions
revSentiBing_predTst<- predict(rfModel1, revDTM_sentiBing_tst %>% select(-review_id))$predictions

ConfM_train <- table(actual=revDTM_sentiBing_trn$hiLo, preds=revSentiBing_predTrn[,2]>0.5)
#display confusion Matrix
ConfM_train
```

```
##       preds
## actual FALSE  TRUE
##     -1  3097   437
##     1    152 10817
```

```
#calculate accuracy
RF1_train_accuracy <- sum(diag(ConfM_train))/sum(ConfM_train)
RF1_train_accuracy
```

```
## [1] 0.9593877
```

Now lets look at the confusion matrix and the accuracy on the test set.

```
ConfM_test <- table(actual=revDTM_sentiBing_tst$hiLo, preds=revSentiBing_predTst[,2]>0.5)
#print the confusion matirx
ConfM_test
```

```
##       preds
## actual FALSE  TRUE
##     -1  2333  1170
##     1    561 10438
```

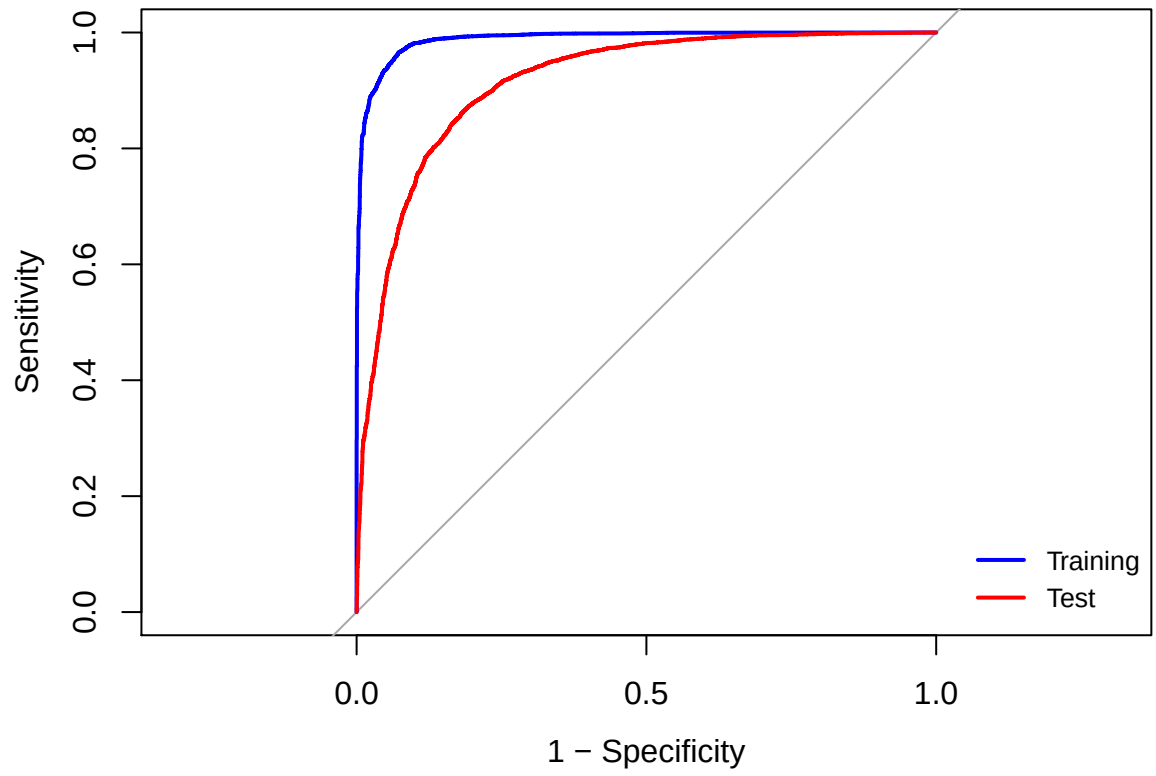
```
#Calcl Accuracy
RF1_test_accuracy <- sum(diag(ConfM_test))/sum(ConfM_test)
RF1_test_accuracy
```

```
## [1] 0.8806372
```

We could see from the results above that accuracy is at 96% on the train data and at 88% on the test data. There is a little bit of overfit but overall these results are very good.

Now lets look at the ROC curve for both test and train.

```
rocTrn <- roc(revDTM_sentiBing_trn$hiLo, revSentiBing_predTrn[,2], levels=c(-1, 1))
rocTst_bing <- roc(revDTM_sentiBing_tst$hiLo, revSentiBing_predTst[,2], levels=c(-1, 1))

plot.roc(rocTrn, col='blue', legacy.axes = TRUE)
plot.roc(rocTst_bing, col='red', add=TRUE)
legend("bottomright", legend=c("Training", "Test"),
        col=c("blue", "red"), lwd=2, cex=0.8, bty='n')
```



We see again that performance on the train data is better then the test data but this ROC indicates good results.

Random Forest model Using AFINN

Now lets look at how predicting HiLo with random forest will perform with AFINN.

```
#Or, since we want to keep the stars column
revDTM_sentiAfinn <- rrSenti_afinn %>%  pivot_wider(id_cols = c(review_id,stars), names_from = word, va

#filter out the reviews with stars=3, and calculate hiLo sentiment 'class'
revDTM_sentiAfinn<- revDTM_sentiAfinn %>% filter(stars!=3) %>% mutate(hiLo=ifelse(stars<=2, -1, 1)) %>%

#how many review with 1, -1  'class'
revDTM_sentiAfinn %>% group_by(hiLo) %>% tally()
```

```
## # A tibble: 2 x 2
```

```
##    hiLo     n
## * <dbl> <int>
## 1    -1  6897
## 2     1 21518
```

```
#develop a random forest model to predict hiLo from the words in the reviews

library(ranger)

#replace all the NAs with 0
revDTM_sentiAfinn<-revDTM_sentiAfinn %>% replace(., is.na(.), 0)

revDTM_sentiAfinn$hiLo<- as.factor(revDTM_sentiAfinn$hiLo)


library(rsample)
revDTM_sentiAfinn_split<- initial_split(revDTM_sentiAfinn, 0.5)
revDTM_sentiAfinn_trn<- training(revDTM_sentiAfinn_split)
revDTM_sentiAfinn_tst<- testing(revDTM_sentiAfinn_split)

rfModel2 <-ranger(dependent.variable.name = "hiLo", data=revDTM_sentiAfinn_trn %>% select(-review_id), n
```

```
## Growing trees.. Progress: 74%. Estimated remaining time: 10 seconds.
## Computing permutation importance.. Progress: 9%. Estimated remaining time: 4 minutes, 58 seconds.
## Computing permutation importance.. Progress: 19%. Estimated remaining time: 4 minutes, 21 seconds.
## Computing permutation importance.. Progress: 30%. Estimated remaining time: 3 minutes, 43 seconds.
## Computing permutation importance.. Progress: 39%. Estimated remaining time: 3 minutes, 15 seconds.
## Computing permutation importance.. Progress: 49%. Estimated remaining time: 2 minutes, 42 seconds.
## Computing permutation importance.. Progress: 59%. Estimated remaining time: 2 minutes, 10 seconds.
## Computing permutation importance.. Progress: 69%. Estimated remaining time: 1 minute, 39 seconds.
## Computing permutation importance.. Progress: 79%. Estimated remaining time: 1 minute, 8 seconds.
## Computing permutation importance.. Progress: 88%. Estimated remaining time: 38 seconds.
## Computing permutation importance.. Progress: 98%. Estimated remaining time: 7 seconds.
```

```
rfModel2
```

```
## Ranger result
##
## Call:
##  ranger(dependent.variable.name = "hiLo", data = revDTM_sentiAfinn_trn %>%      select(-review_id), n
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      14208
## Number of independent variables:  527
## Mtry:                             22
## Target node size:                 10
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.09996373
```

Now that we trained the model, lets look at the performance. First, we will look at the confusion matrix and then compute the accuracy for both train set and test set.

```
revSentiAfinn_predTrn<- predict(rfModel2, revDTM_sentiAfinn_trn %>% select(-review_id))$predictions
revSentiAfinn_predTst<- predict(rfModel2, revDTM_sentiAfinn_tst %>% select(-review_id))$predictions

ConfM_train <- table(actual=revDTM_sentiAfinn_trn$hiLo, preds=revSentiAfinn_predTrn[,2]>0.5)
#display confusion Matrix
ConfM_train
```

```
##       preds
## actual FALSE  TRUE
##     -1  2892   571
##     1    185 10560
```

```
#calculate accuracy
RF2_train_accuracy <- sum(diag(ConfM_train))/sum(ConfM_train)
RF2_train_accuracy
```

```
## [1] 0.9467905
```

Now lets look at the confusion matrix and the accuracy on the test set.

```
ConfM_test <- table(actual=revDTM_sentiAfinn_tst$hiLo, preds=revSentiAfinn_predTst[,2]>0.5)
#print the confusion matirx
ConfM_test
```

```
##       preds
## actual FALSE  TRUE
##     -1  2092  1342
##     1    595 10178
```

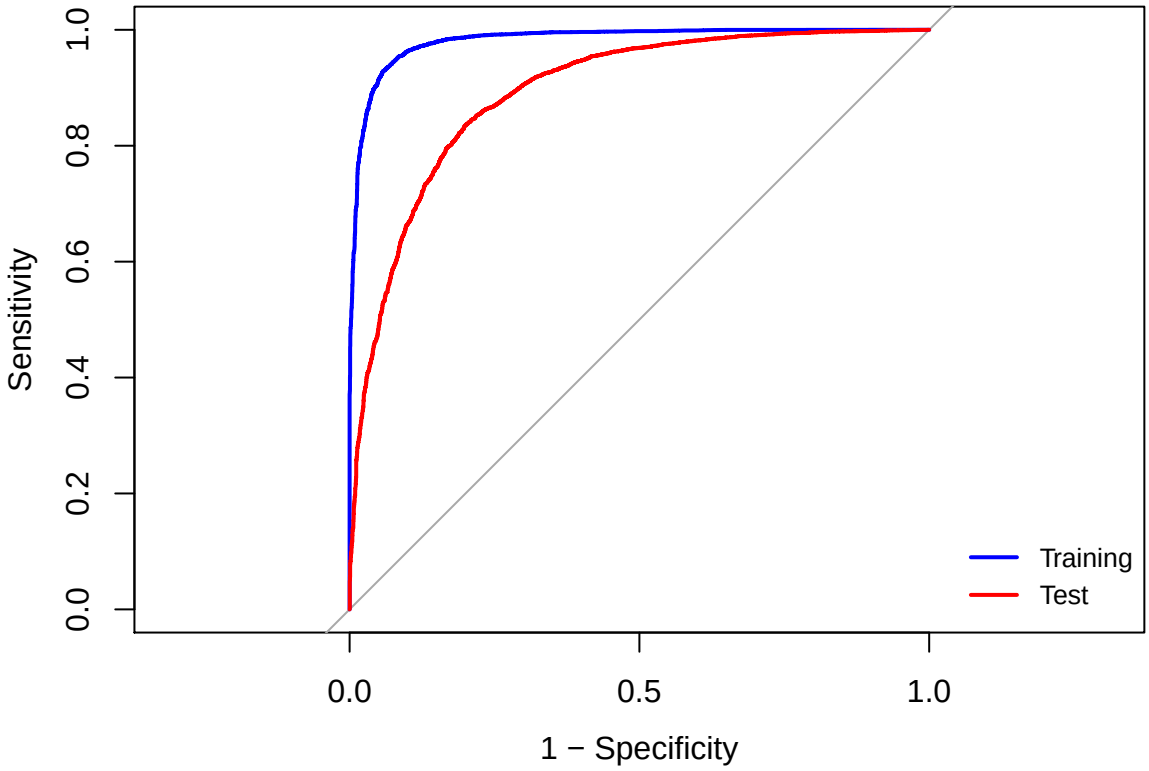
```
#Calcl Accuracy
RF2_test_accuracy <- sum(diag(ConfM_test))/sum(ConfM_test)
RF2_test_accuracy
```

```
## [1] 0.8636588
```

We could see from the results above that accuracy is at 94% on the train data and at 86% on the test data. There is a little bit of overfit but overall these results are very good.

The Bing model performs better than the Afinn model according to the accuracy. Now lets look at the ROC curves

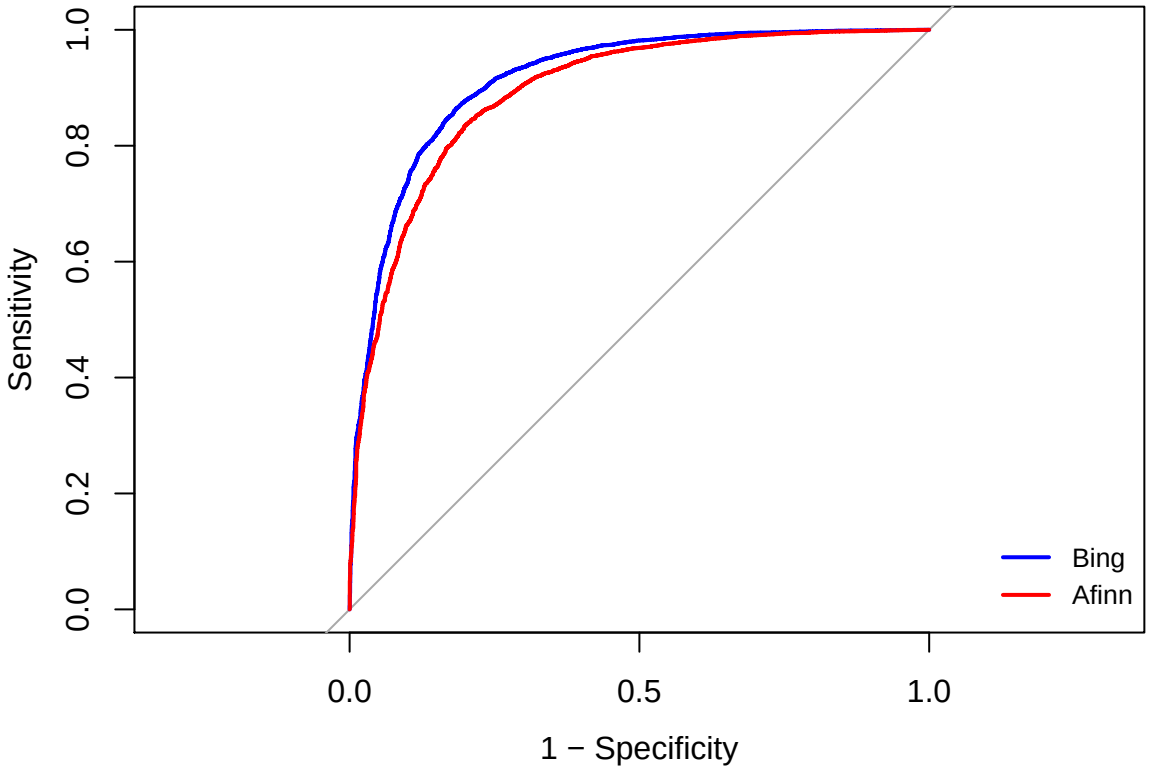
```
rocTrn <- roc(revDTM_sentiAfinn_trn$hiLo, revSentiAfinn_predTrn[,2], levels=c(-1, 1))
rocTst_Afinn <- roc(revDTM_sentiAfinn_tst$hiLo, revSentiAfinn_predTst[,2], levels=c(-1, 1))

plot.roc(rocTrn, col='blue', legacy.axes = TRUE)
plot.roc(rocTst_Afinn, col='red', add=TRUE)
legend("bottomright", legend=c("Training", "Test"),
        col=c("blue", "red"), lwd=2, cex=0.8, bty='n')
```



We see again that performance on the train data is better then the test data but this ROC indicates good results.

Lets compare the ROC curves for the AFINN test set vs the Bing test set

```
plot.roc(rocTst_bing, col='blue', legacy.axes = TRUE)
plot.roc(rocTst_Afinn, col='red', add=TRUE)
legend("bottomright", legend=c("Bing", "Afinn"),
        col=c("blue", "red"), lwd=2, cex=0.8, bty='n')
```



We could also see from the ROC curve that the Bing dictionary performs better when trying to predict HiLo of reviews through a Random Forest model.

Naive-Bayes model using Bing dictionary

Now lets use a different model to predict HiLo. We will use the Naive-bayes model for both Bing and Afinn and then compare the results at the end.

```
nbModel1<-naiveBayes(hiLo ~ ., data=revDTM_sentiBing_trn %>% select(-review_id))

revSentiBing_NBpredTrn<-predict(nbModel1, revDTM_sentiBing_trn, type = "raw")
revSentiBing_NBpredTst<-predict(nbModel1, revDTM_sentiBing_tst, type = "raw")
```

Now lets look at the AUC value for the train set:

```
auc(as.numeric(revDTM_sentiBing_trn$hiLo), revSentiBing_NBpredTrn[,2])
```

```
## Area under the curve: 0.7072
```

And the AUC value on the test set

```
auc(as.numeric(revDTM_sentiBing_tst$hiLo), revSentiBing_NBpredTst[,2])
```

```
## Area under the curve: 0.7228
```

We see the AUC value for the Bing Naive base model on the test set is 72.43% which is good.

Naive-Bayes model using Afinn dictionary

Now lets compare to how the AFINN dictionary will perform using the Naive base model

```
nbModel2<-naiveBayes(hiLo ~ ., data=revDTM_sentiAfinn_trn %>% select(-review_id))

revSentiAfinn_NBpredTrn<-predict(nbModel2, revDTM_sentiAfinn_trn, type = "raw")
revSentiAfinn_NBpredTst<-predict(nbModel2, revDTM_sentiAfinn_tst, type = "raw")
```

Now lets look at the AUC value for the train set:

```
auc(as.numeric(revDTM_sentiAfinn_trn$hiLo), revSentiAfinn_NBpredTrn[,2])
```

```
## Area under the curve: 0.733
```

And the AUC value on the test set

```
auc(as.numeric(revDTM_sentiAfinn_tst$hiLo), revSentiAfinn_NBpredTst[,2])
```

```
## Area under the curve: 0.7318
```

We see the AUC value for the Afinn dictionary on the test set is 74.12%. So using the Naive bayes model, the Afinn dictionary gives us better predictions.

SVM Model using Bing Dictionary

Now lets try our 3rd model: SVM classification – for restaurant reviews.

First, we will build the SVM model using the data from the bing dictionary and then we will build the same using the AFINN dictionary and compare the results

```
system.time( svmM1 <- svm(as.factor(hiLo) ~., data = revDTM_sentiBing_trn
%>% select(-review_id), kernel="radial", cost=5, gamma=5, scale=FALSE) )
```

```
##    user  system elapsed
##   48.92    0.39   49.39
```

```
revDTM_predTrn_svm1<-predict(svmM1, revDTM_sentiBing_trn)
revDTM_predTst_svm1<-predict(svmM1, revDTM_sentiBing_tst)
```

Now lets look at the performance of this model. We will print out the confusion matrix for the train data and the test data then calculate the accuracy for both

```
#Condusion Matrix on train data
cm_svm_train <- table(actual= revDTM_sentiBing_trn$hiLo, predicted= revDTM_predTrn_svm1)
cm_svm_train
```

```
##       predicted
## actual    -1     1
##     -1  3072   462
##     1    124 10845
```

```
#Calculate accuracy
svm1_train_accuracy <- sum(diag(cm_svm_train))/sum(cm_svm_train)
svm1_train_accuracy
```

```
## [1] 0.9595946
```

We get an accuracy of 96,57%

Now lets see the performance on the test data

```
#Condusion Matrix on train data
cm_svm_test <- table(actual= revDTM_sentiBing_tst$hiLo, predicted= revDTM_predTst_svm1)
cm_svm_test
```

```
##       predicted
## actual    -1     1
##     -1  2373  1130
##     1    499 10500
```

```
#Calculate accuracy
svm1_test_accuracy <- sum(diag(cm_svm_test))/sum(cm_svm_test)
svm1_test_accuracy
```

```
## [1] 0.8876707
```

We get accuracy of 87.84% on the test set which is good. We do have a little bit of overfit but the results are still good overall.

We could still tune the SVM parameters using the tune function in SVM. lets experiment with it and see if we get even better results. The code below with give us the best SVM model; However, since it takes hours to run and causes trouble on the knitted RMD, we will just stick with the model above.

```
system.time( svm_tune <- tune(svm, as.factor(hiLo) ~., data = revDTM_sentiBing_trn %>% select(-review_i
kernel="radial", ranges = list( cost=c(0.1,1,10,50), gamma = c(0.5,1,2,5, 10))) )

#Check performance for different tuned parameters
svm_tune$performances
#Best model
svm_tune$best.parameters
svm_tune$best.model
```

SVM Model using Afinn Dictionary

Now lets build the SVM model using the Afinn dictionary and compare the results to the bing SVM model

```
system.time( svmM2 <- svm(as.factor(hiLo) ~., data = revDTM_sentiAfinn_trn
%>% select(-review_id), kernel="radial", cost=5, gamma=5, scale=FALSE) )
```

```
##    user  system elapsed
##   31.67    0.14   31.89
```

```
revDTM_predTrn_svm2<-predict(svmM2, revDTM_sentiAfinn_trn)
revDTM_predTst_svm2<-predict(svmM2, revDTM_sentiAfinn_tst)
```

Now lets look at the performance of this model. We will print out the confusion matrix for the train data and the test data then calculate the accuracy for both

```
#Condusion Matrix on train data
cm_svm2_train <- table(actual= revDTM_sentiAfinn_trn$hiLo, predicted= revDTM_predTrn_svm2)
cm_svm2_train
```

```
##       predicted
## actual    -1     1
##     -1  2737   726
##     1    236 10509
```

```
#Calculate accuracy
svm2_train_accuracy <- sum(diag(cm_svm2_train))/sum(cm_svm2_train)
svm2_train_accuracy
```

```
## [1] 0.9322917
```

We get an accuracy of 93.08%

Now lets see the performance on the test data

```
#Condusion Matrix on train data
cm_svm2_test <- table(actual= revDTM_sentiAfinn_tst$hiLo, predicted= revDTM_predTst_svm2)
cm_svm2_test
```

```
##       predicted
## actual    -1     1
##     -1  2150  1284
##     1    652 10121
```

```
#Calculate accuracy
svm2_test_accuracy <- sum(diag(cm_svm2_test))/sum(cm_svm2_test)
svm2_test_accuracy
```

```
## [1] 0.8637291
```

We get accuracy of 86.47% on the test set which is good. We do have a little bit of overfit but the results are still good overall.

So we could see that using Naive base model to predict review sentiment gives better results with the Bing dictionary based on the accuracy performance.

Develop a model on broader set of terms

In this section, we will develop a model using a broader set of terms and not just the terms we have in each dictionary. In order to do this, we will use the rrtokens directly. We will then do some clean-up activities like removing words that are found in $> 90\%$ of the reviews or $<30\%$ of the reviews.

We will use a random forest ranger model for this section

Data Prep

```
#in how many reviews each word occurs
rWords<-rrTokens %>% group_by(word) %>% summarise(nr=n()) %>% arrange(desc(nr))

#delete the words that are in more than 90% of less than 30% of reviews
reduced_rWords<-rWords %>% filter(nr< 6000 & nr > 30)
#Store the result bach in rrtokens
reduced_rrTokens <- left_join(reduced_rWords, rrTokens)

#Now convert it to a DTM, where each row is for a review (document), and columns are the terms (words)
revDTM  <- reduced_rrTokens %>%  pivot_wider(id_cols = c(review_id,stars), names_from = word, values_fr

#create the dependent variable hiLo of good/bad reviews absed on stars, and remove the review with star
revDTM <- revDTM %>% filter(stars!=3) %>% mutate(hiLo=ifelse(stars<=2, -1, 1)) %>% select(-stars)

#replace NAs with 0s
revDTM<-revDTM %>% replace(., is.na(.), 0)

revDTM$hiLo<-as.factor(revDTM$hiLo)

revDTM_split<- initial_split(revDTM, 0.4)
revDTM_trn<- training(revDTM_split)
revDTM_tst<- testing(revDTM_split)
```

Now that we prepared our data, we could run the random forest model

```
rfModel2<-ranger(dependent.variable.name = "hiLo", data=revDTM_trn %>% select(-review_id), num.trees = 5
```

```
## Growing trees.. Progress: 39%. Estimated remaining time: 48 seconds.
## Growing trees.. Progress: 80%. Estimated remaining time: 15 seconds.
## Computing permutation importance.. Progress: 0%. Estimated remaining time: 7 hours, 4 minutes, 9 sec
## Computing permutation importance.. Progress: 2%. Estimated remaining time: 1 hour, 41 minutes, 50 se
## Computing permutation importance.. Progress: 3%. Estimated remaining time: 1 hour, 18 minutes, 7 sec
## Computing permutation importance.. Progress: 5%. Estimated remaining time: 1 hour, 9 minutes, 40 sec
## Computing permutation importance.. Progress: 7%. Estimated remaining time: 1 hour, 5 minutes, 19 sec
## Computing permutation importance.. Progress: 8%. Estimated remaining time: 1 hour, 1 minute, 0 second
## Computing permutation importance.. Progress: 9%. Estimated remaining time: 57 minutes, 40 seconds.
## Computing permutation importance.. Progress: 10%. Estimated remaining time: 58 minutes, 48 seconds.
## Computing permutation importance.. Progress: 11%. Estimated remaining time: 56 minutes, 25 seconds.
## Computing permutation importance.. Progress: 12%. Estimated remaining time: 57 minutes, 10 seconds.
## Computing permutation importance.. Progress: 13%. Estimated remaining time: 56 minutes, 39 seconds.
## Computing permutation importance.. Progress: 14%. Estimated remaining time: 54 minutes, 28 seconds.
## Computing permutation importance.. Progress: 15%. Estimated remaining time: 54 minutes, 24 seconds.
## Computing permutation importance.. Progress: 16%. Estimated remaining time: 53 minutes, 27 seconds.
## Computing permutation importance.. Progress: 17%. Estimated remaining time: 53 minutes, 3 seconds.
## Computing permutation importance.. Progress: 18%. Estimated remaining time: 51 minutes, 27 seconds.
## Computing permutation importance.. Progress: 19%. Estimated remaining time: 51 minutes, 10 seconds.
## Computing permutation importance.. Progress: 20%. Estimated remaining time: 50 minutes, 4 seconds.
## Computing permutation importance.. Progress: 21%. Estimated remaining time: 49 minutes, 52 seconds.
## Computing permutation importance.. Progress: 22%. Estimated remaining time: 48 minutes, 49 seconds.
## Computing permutation importance.. Progress: 23%. Estimated remaining time: 48 minutes, 32 seconds.
```

```
## Computing permutation importance.. Progress: 24%. Estimated remaining time: 47 minutes, 42 seconds.
## Computing permutation importance.. Progress: 25%. Estimated remaining time: 47 minutes, 55 seconds.
## Computing permutation importance.. Progress: 26%. Estimated remaining time: 46 minutes, 32 seconds.
## Computing permutation importance.. Progress: 27%. Estimated remaining time: 45 minutes, 41 seconds.
## Computing permutation importance.. Progress: 28%. Estimated remaining time: 45 minutes, 24 seconds.
## Computing permutation importance.. Progress: 29%. Estimated remaining time: 44 minutes, 37 seconds.
## Computing permutation importance.. Progress: 30%. Estimated remaining time: 44 minutes, 0 seconds.
## Computing permutation importance.. Progress: 31%. Estimated remaining time: 43 minutes, 19 seconds.
## Computing permutation importance.. Progress: 32%. Estimated remaining time: 43 minutes, 0 seconds.
## Computing permutation importance.. Progress: 33%. Estimated remaining time: 42 minutes, 18 seconds.
## Computing permutation importance.. Progress: 34%. Estimated remaining time: 41 minutes, 11 seconds.
## Computing permutation importance.. Progress: 35%. Estimated remaining time: 40 minutes, 55 seconds.
## Computing permutation importance.. Progress: 36%. Estimated remaining time: 40 minutes, 6 seconds.
## Computing permutation importance.. Progress: 37%. Estimated remaining time: 39 minutes, 36 seconds.
## Computing permutation importance.. Progress: 38%. Estimated remaining time: 38 minutes, 50 seconds.
## Computing permutation importance.. Progress: 39%. Estimated remaining time: 37 minutes, 42 seconds.
## Computing permutation importance.. Progress: 40%. Estimated remaining time: 37 minutes, 36 seconds.
## Computing permutation importance.. Progress: 41%. Estimated remaining time: 36 minutes, 41 seconds.
## Computing permutation importance.. Progress: 42%. Estimated remaining time: 36 minutes, 9 seconds.
## Computing permutation importance.. Progress: 43%. Estimated remaining time: 35 minutes, 24 seconds.
## Computing permutation importance.. Progress: 44%. Estimated remaining time: 34 minutes, 53 seconds.
## Computing permutation importance.. Progress: 45%. Estimated remaining time: 34 minutes, 13 seconds.
## Computing permutation importance.. Progress: 46%. Estimated remaining time: 33 minutes, 48 seconds.
## Computing permutation importance.. Progress: 47%. Estimated remaining time: 32 minutes, 49 seconds.
## Computing permutation importance.. Progress: 48%. Estimated remaining time: 31 minutes, 55 seconds.
## Computing permutation importance.. Progress: 49%. Estimated remaining time: 31 minutes, 37 seconds.
## Computing permutation importance.. Progress: 50%. Estimated remaining time: 30 minutes, 48 seconds.
## Computing permutation importance.. Progress: 51%. Estimated remaining time: 30 minutes, 9 seconds.
## Computing permutation importance.. Progress: 53%. Estimated remaining time: 29 minutes, 21 seconds.
## Computing permutation importance.. Progress: 54%. Estimated remaining time: 28 minutes, 49 seconds.
## Computing permutation importance.. Progress: 55%. Estimated remaining time: 28 minutes, 9 seconds.
## Computing permutation importance.. Progress: 56%. Estimated remaining time: 27 minutes, 31 seconds.
## Computing permutation importance.. Progress: 57%. Estimated remaining time: 26 minutes, 54 seconds.
## Computing permutation importance.. Progress: 58%. Estimated remaining time: 26 minutes, 18 seconds.
## Computing permutation importance.. Progress: 59%. Estimated remaining time: 25 minutes, 41 seconds.
## Computing permutation importance.. Progress: 60%. Estimated remaining time: 25 minutes, 2 seconds.
## Computing permutation importance.. Progress: 60%. Estimated remaining time: 24 minutes, 35 seconds.
## Computing permutation importance.. Progress: 62%. Estimated remaining time: 23 minutes, 47 seconds.
## Computing permutation importance.. Progress: 63%. Estimated remaining time: 23 minutes, 0 seconds.
## Computing permutation importance.. Progress: 64%. Estimated remaining time: 22 minutes, 35 seconds.
## Computing permutation importance.. Progress: 65%. Estimated remaining time: 21 minutes, 45 seconds.
## Computing permutation importance.. Progress: 65%. Estimated remaining time: 21 minutes, 31 seconds.
## Computing permutation importance.. Progress: 67%. Estimated remaining time: 20 minutes, 34 seconds.
## Computing permutation importance.. Progress: 68%. Estimated remaining time: 19 minutes, 48 seconds.
## Computing permutation importance.. Progress: 69%. Estimated remaining time: 19 minutes, 1 seconds.
## Computing permutation importance.. Progress: 70%. Estimated remaining time: 18 minutes, 36 seconds.
## Computing permutation importance.. Progress: 71%. Estimated remaining time: 17 minutes, 57 seconds.
## Computing permutation importance.. Progress: 72%. Estimated remaining time: 17 minutes, 28 seconds.
## Computing permutation importance.. Progress: 73%. Estimated remaining time: 16 minutes, 41 seconds.
## Computing permutation importance.. Progress: 74%. Estimated remaining time: 16 minutes, 13 seconds.
## Computing permutation importance.. Progress: 75%. Estimated remaining time: 15 minutes, 34 seconds.
## Computing permutation importance.. Progress: 76%. Estimated remaining time: 14 minutes, 47 seconds.
## Computing permutation importance.. Progress: 77%. Estimated remaining time: 14 minutes, 20 seconds.
## Computing permutation importance.. Progress: 78%. Estimated remaining time: 13 minutes, 35 seconds.
```

```
## Computing permutation importance.. Progress: 79%. Estimated remaining time: 12 minutes, 48 seconds.
## Computing permutation importance.. Progress: 80%. Estimated remaining time: 12 minutes, 30 seconds.
## Computing permutation importance.. Progress: 81%. Estimated remaining time: 11 minutes, 42 seconds.
## Computing permutation importance.. Progress: 82%. Estimated remaining time: 11 minutes, 13 seconds.
## Computing permutation importance.. Progress: 83%. Estimated remaining time: 10 minutes, 30 seconds.
## Computing permutation importance.. Progress: 84%. Estimated remaining time: 9 minutes, 35 seconds.
## Computing permutation importance.. Progress: 85%. Estimated remaining time: 9 minutes, 15 seconds.
## Computing permutation importance.. Progress: 86%. Estimated remaining time: 8 minutes, 30 seconds.
## Computing permutation importance.. Progress: 87%. Estimated remaining time: 7 minutes, 52 seconds.
## Computing permutation importance.. Progress: 88%. Estimated remaining time: 7 minutes, 16 seconds.
## Computing permutation importance.. Progress: 89%. Estimated remaining time: 6 minutes, 39 seconds.
## Computing permutation importance.. Progress: 90%. Estimated remaining time: 5 minutes, 54 seconds.
## Computing permutation importance.. Progress: 91%. Estimated remaining time: 5 minutes, 25 seconds.
## Computing permutation importance.. Progress: 92%. Estimated remaining time: 4 minutes, 41 seconds.
## Computing permutation importance.. Progress: 94%. Estimated remaining time: 3 minutes, 55 seconds.
## Computing permutation importance.. Progress: 95%. Estimated remaining time: 3 minutes, 19 seconds.
## Computing permutation importance.. Progress: 96%. Estimated remaining time: 2 minutes, 42 seconds.
## Computing permutation importance.. Progress: 96%. Estimated remaining time: 2 minutes, 13 seconds.
## Computing permutation importance.. Progress: 98%. Estimated remaining time: 1 minute, 28 seconds.
## Computing permutation importance.. Progress: 99%. Estimated remaining time: 44 seconds.
## Computing permutation importance.. Progress: 100%. Estimated remaining time: 7 seconds.
```

```
rfModel2
```

```
## Ranger result
##
## Call:
##  ranger(dependent.variable.name = "hiLo", data = revDTM_trn %>%      select(-review_id), num.trees =
##
## Type:                             Probability estimation
## Number of trees:                  500
## Sample size:                      12133
## Number of independent variables:  3410
## Mtry:                             58
## Target node size:                 10
## Variable importance mode:         permutation
## Splitrule:                        gini
## OOB prediction error (Brier s.):  0.0846823
```

We could see from above that oob error is very low. which means this model is also performing well.

Discussion

in this section, we will summurize what we did for questions (c) amd (d).

First, in question in (c), We saw which words contributed to to positive/negative senteiment according to the three lexicons: Bing, Afinn, and NCR. We saw results that made sense for Afinn and Bing; However, for NCR, we saw some words that don't necessarily belong to negative or positive sentiments like (food, chickenm restaurant,etc). This was expected as the NCF dictionary uses emotions and not just negative and positive sentiments.

Then, we performed analysis by review sentiment for Bing and Afinn to look into sentiment by review and see how it relates to review's star ratings. At this point, we are not developing a model, but we are predicting the sentiment (positive or negative) based on the results we get from the dictionaries. We classified reviews

on high/low stats based on aggregated sentiment of words in the reviews. This enabled us to get the actual (based on start ratings <2 negative >2 positive) and the predicted (based on the dictionaries). We printed the confusion matrices for both Afinn and Bing dictionaries and found the following performances:

- Bing accuracy in predicting review sentiment: 81.30%
- Afinn accuracy in predicting review sentiment: 81.78%

As you can see, Afinn performed slightly better.

In question (d) we developed 3 different models using the data from both Afinn and bing dictionaries to predict review sentiment. We developed the following models:

- Random forest (Bing dictionary)
- Random forest (Afinn dictionary)
- Naive Base (bing Dictionary)
- Naive Base (Afinn Dictionary)
- SVM (bing Dictionary)
- SVM (Afinn dictionary)

Before we started building our models, we performed lemmatization on both the Afinn words and the bing words. Lemmitzation is the process of bring words back to their 'original form' for example, words like building, builds, built will be transformed to build.

All the models performed well in terms of accuracy and ROC. The accuracy was general from 84% to 88% on the test data and the AUC was between (0.70 and 0.73) on the test data. We also noticed the models (Random forest, Naive base, and SVM) perform better than the predictions in question (c) (only based on the dictionary prediction: no models developed).

Finally, we developed a model using a broader list of terms that are not restricted just to the dictionary terms only. We developed a random forest model for this purpose and used the data from rrtokens as it has all the words from the reviews. We had to do some data preparations before running the model like removing words that are used in more than 90% of the reviews or less than 30%, created the hiLo field (<2 stars negative >2 stars positive), and of course replaced all na values. Finally, we ran the random forest model and got high accuracy rate.