UNIVERSITY OF
ILLINOIS CHICAGO

Assignment 3

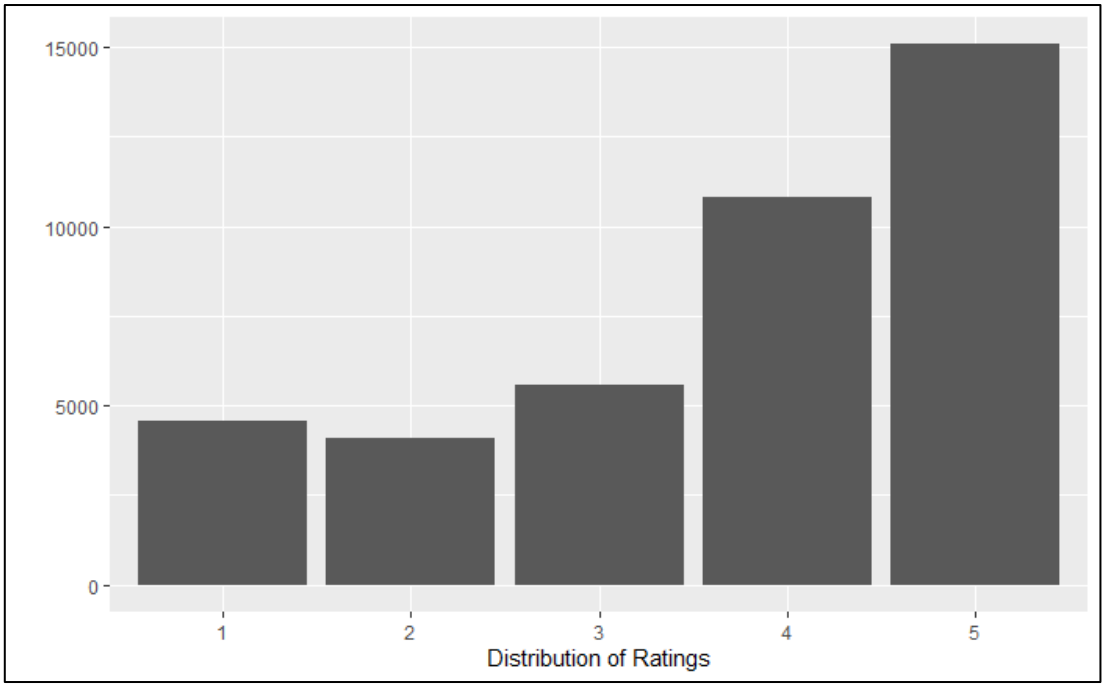
IDS 572 – DATA MINING

ZOHAIB SHEIKH, SHUBHAM KHODE & ANIRUDHA BALKRISHNA

1. **Explore the data.**

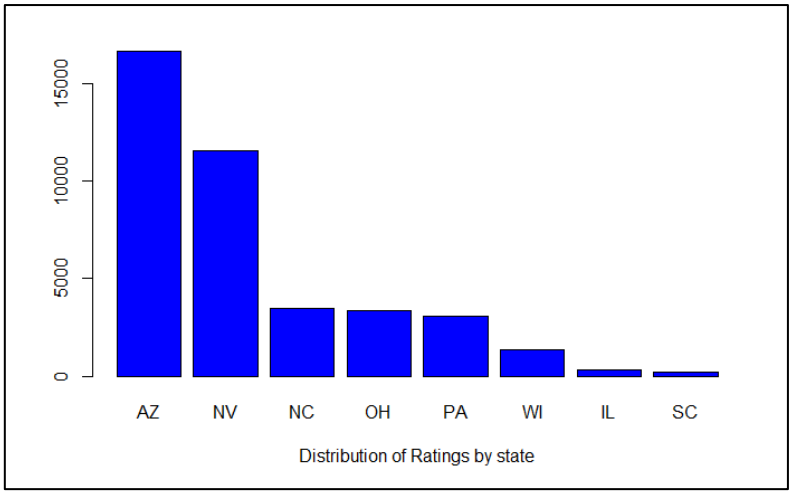
   (i) How are star ratings distributed? How will you use the star ratings to obtain a label indicating 'positive' or 'negative' – explain using the data, graphs, etc.?
   Do star ratings have any relation to 'funny', 'cool', 'useful'? Is this what you expected?
   (ii) How does star ratings for reviews relate to the star-rating given in the dataset for businesses (attribute 'businessStars')? (Can one be calculated from the other?)

The Star rating in the data has five different levels from 1 to 5 with 1 being the lowest and 5 being the highest rating. Overall, the number of reviews across different ratings has uneven distribution with most numbers of reviews receiving a 5-star rating.

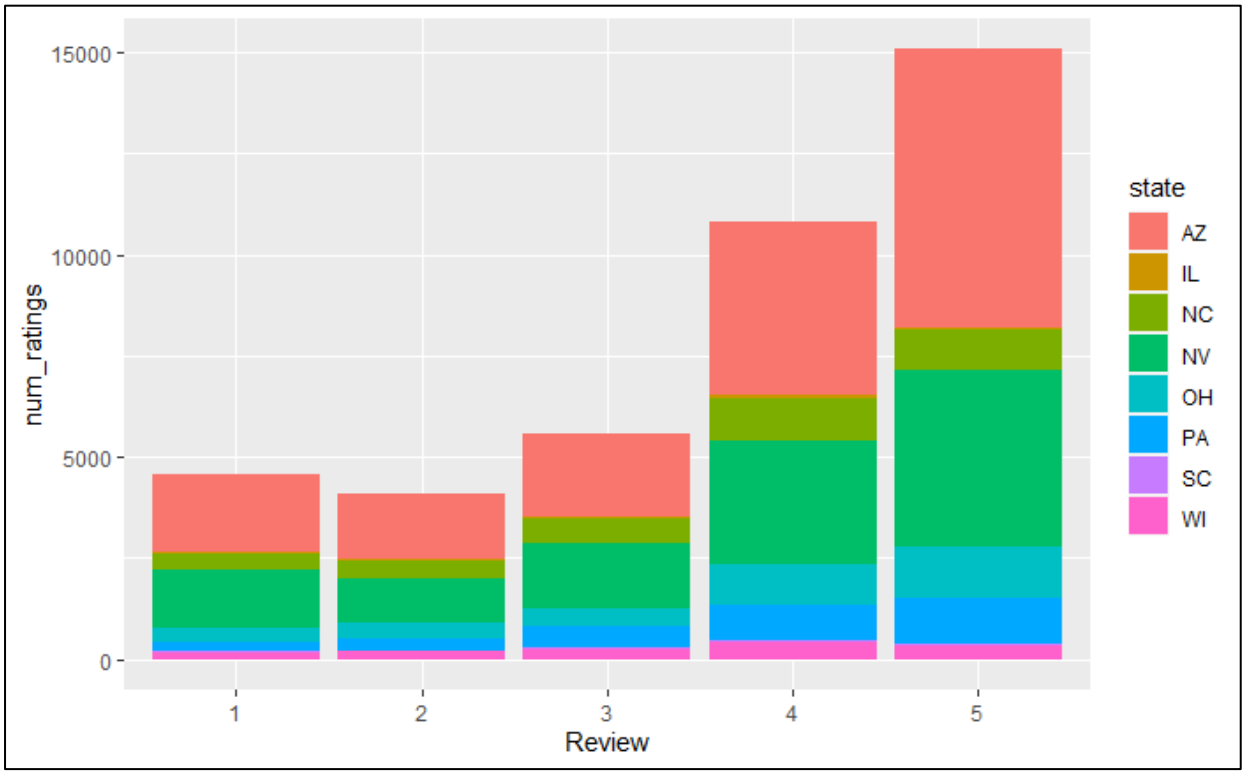


We looked at the start ratings for different dimensions of the data. Let's look at the distribution of star ratings by different states.

Overall, Arizona seems to have the maximum number of reviews received. This is evident due to the fact that Arizona has the highest number of businesses in our data.

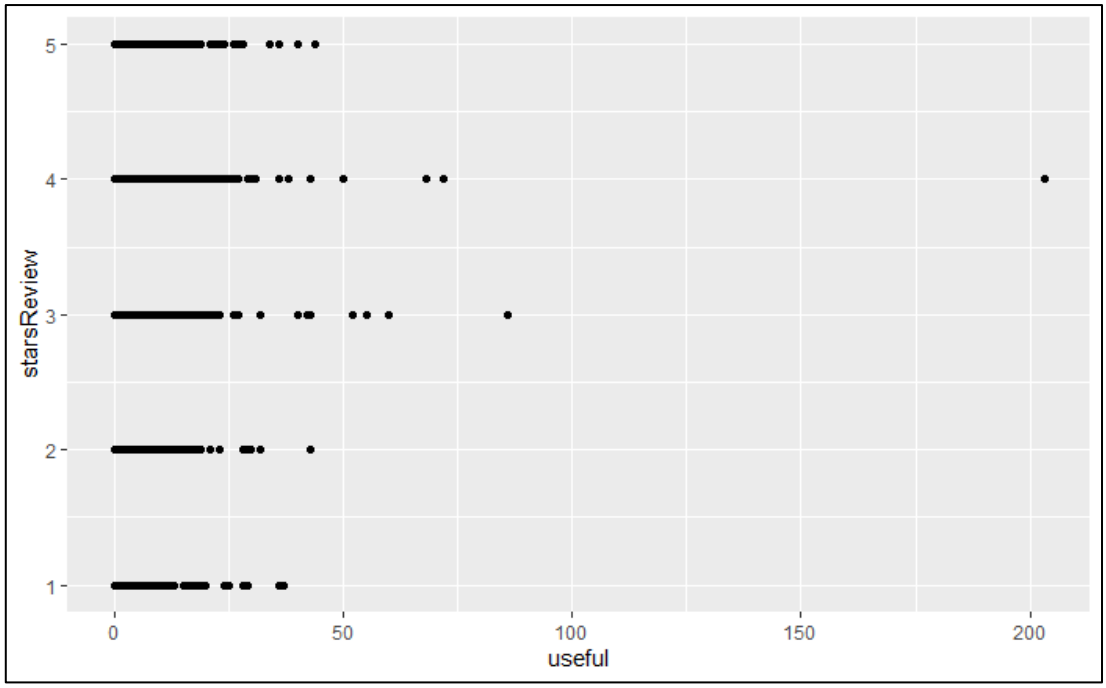


Let's slice the reviews for each state by their star ratings.

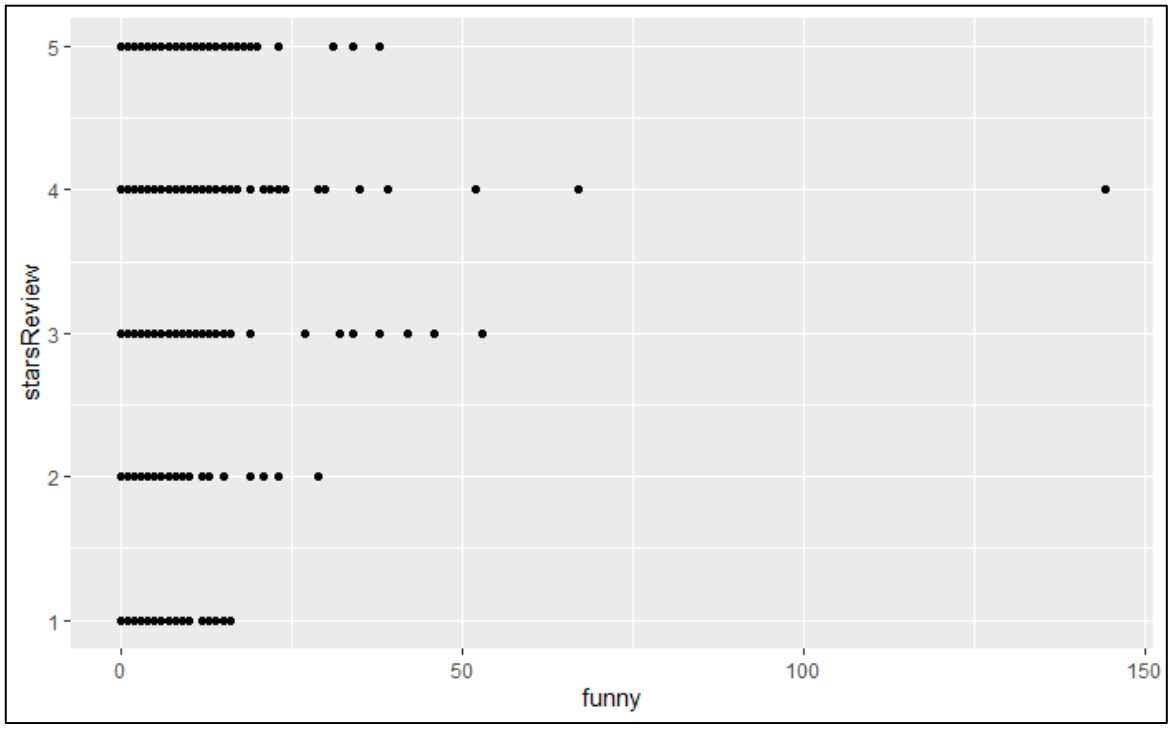


From the above graph, we can also observe that 'Arizona' has the maximum number high quality restaurants.

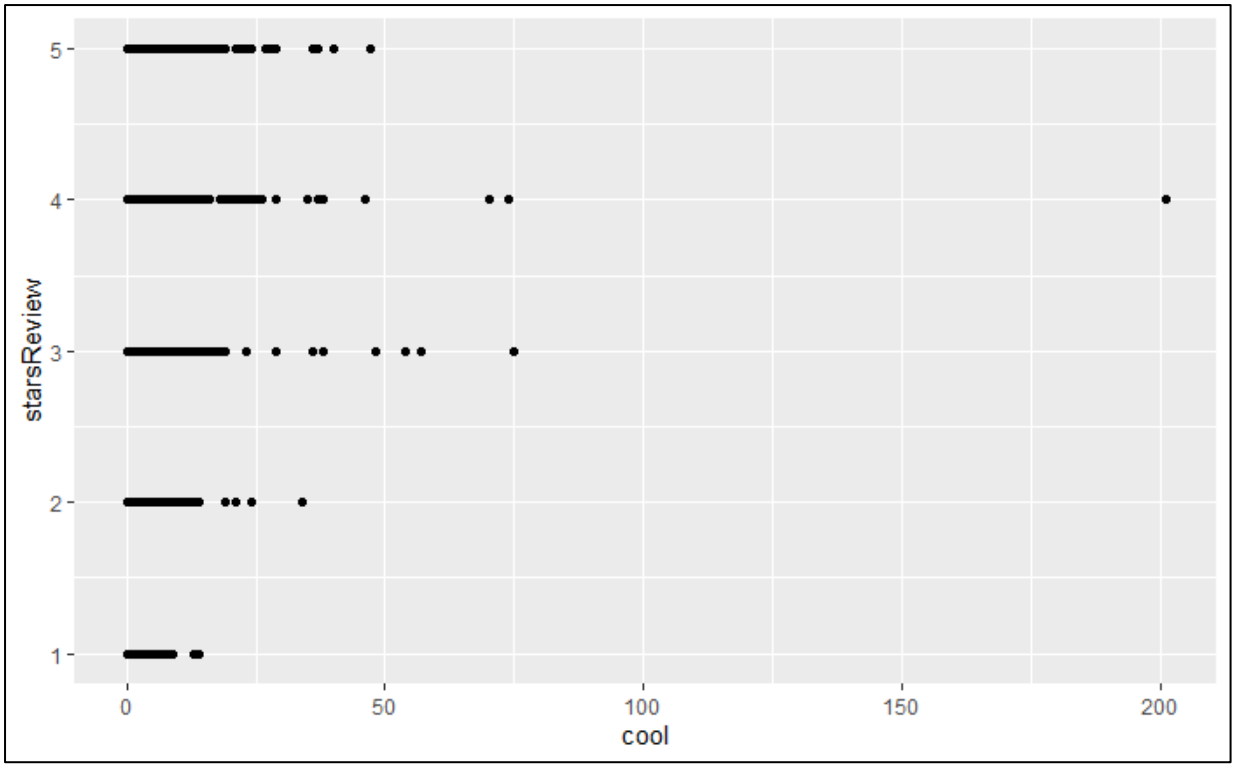
Next, we will explore the relationship between the star ratings and 'useful', 'funny', 'cool' dimension in our data. These columns indicate that for a particular review how many users of the 'Yelp' have identifies it as either 'useful', 'funny' or 'cool'.



From above graph we can see that the maximum number of ‘useful’ tags has been received for the revies having 3 and 4 rating.



From the relationship between star rating and ‘funny’ tag, we can observe that the maximum number of ‘funny’ tags has been received for 3- & 4-star reviews.



We can see a similar distribution of number of 'cool' tags for different star ratings. From the relationship between star rating and 'cool' tag, we can observe that the maximum number of 'cool' tags has been received for 3- & 4-star reviews.

The above relationship between 'useful', 'funny', 'cool' with star ratings is highly expected as the reviews which have 3- and 4- star ratings are written by customers with careful consideration and their analysis of the business. This is also evident from the fact that 3- and 4-star rating have the highest average number of word count. Hence, the reviews with 3- and 4- star ratings have the highest number of different tags received.

According to the data we can see the star ratings is highly correlated with business star ratings. However, the interval scale of star ratings is 1,2,3,4,5. And the interval scale of business star ratings is 1.5,2,2.5,3,3.5,4,4.5,5.

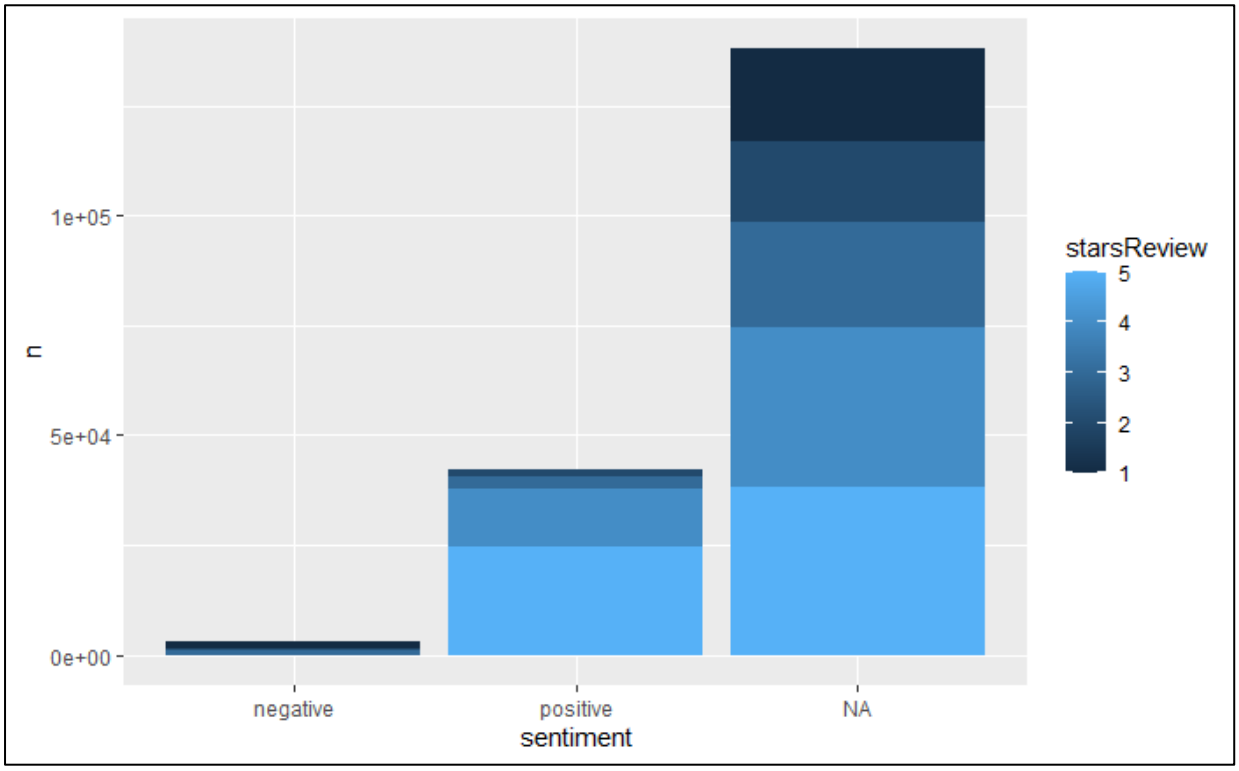
businessStars	**num_of_ratings**	**num_5_ratings**	**num_4_ratings**	**num_3_ratings**	**num_2_ratings**	**num_1_ratings**
5.0	267	93	6	1	0	1
4.5	6362	67	21	6	3	3
4	13799	44	30	12	7	6
3.5	11138	28	31	17	12	12
3	5820	18	25	20	17	20
2.5	2131	11	20	19	20	30

businessStars	num_of_ratings	num_5_ratings	num_4_ratings	num_3_ratings	num_2_ratings	num_1_ratings
2	405	6	11	15	20	49
1.5	165	3	2	10	12	73

Though it seems highly difficult as to predict either of the ratings from other, it can be calculated using expected value across different weights for each ratings. For ex. – the number of star reviews for business stars is distributed across all the different star ratings. Hence using different weights for each star ratings based on the distribution on number of reviews across each category, we can come up with a probabilistic estimate for business star and star ratings vice-versa.

We can use star ratings to obtain a label indicating 'positive' and 'negative' for each review. There can be two ways to do this.

1. Using sentiments from dictionary – We can use the sentiments from different available dictionaries like bing and nrc to get the label for each word. Then we can use a label of 1 for 'positive' sentiments and 0 for 'negative' sentiments for each word. Then we can take the average of these scores for each review. And based on the average score, we can label each review as either 'positive' or 'negative'. Positive means average score >= 0.5 and negative means average score < 0.5.
2. Another way to achieve the 'positive' and 'negative' label would be to use hardcoded scores. For ratings >= 4 we can label the review as 'positive' and for ratings <=2, we can label the review as 'negative'.

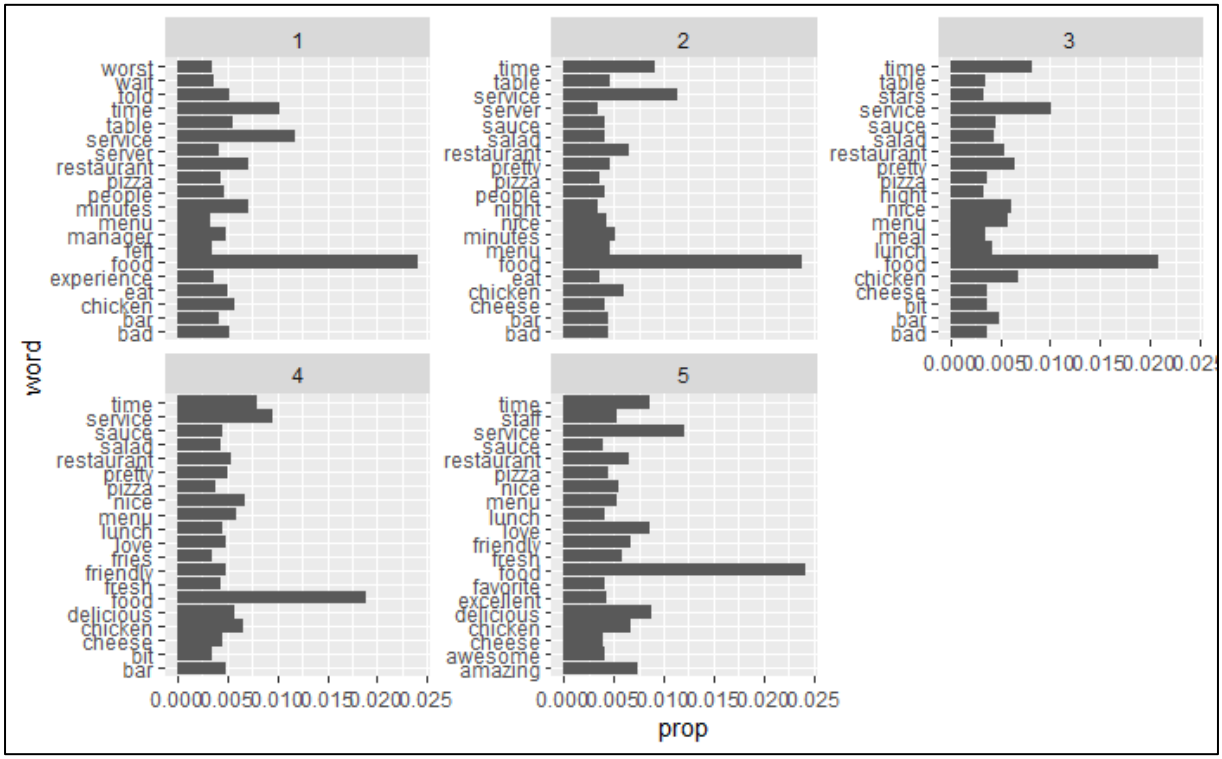


The above graph shows the distribution of 'negative' and 'positive' label based on 'BING' dictionary across different star ratings. Hence, the star ratings can be used to obtain a label.

2. **What are some words indicative of positive and negative sentiment? (One approach is to determine the average star rating for a word based on star ratings of documents where the word occurs). Do these 'positive' and 'negative' words make sense in the context of user reviews being considered? (For this, since we'd like to get a general sense of positive/negative terms, you may like to consider a pruned set of terms -- say, those which occur in a certain minimum and maximum number of documents).**

For further analysis, we will do some data cleaning before. We have executed different techniques as below for cleaning our data:

- Performed Tokenization
- Transformed case (to all lower/upper)
- Filtered stopwords
- Filtered tokens by length - min 3 and max 15.
- Filtered tokens by content – if they matched a 'dictionary' of terms
- Used Lemmatization

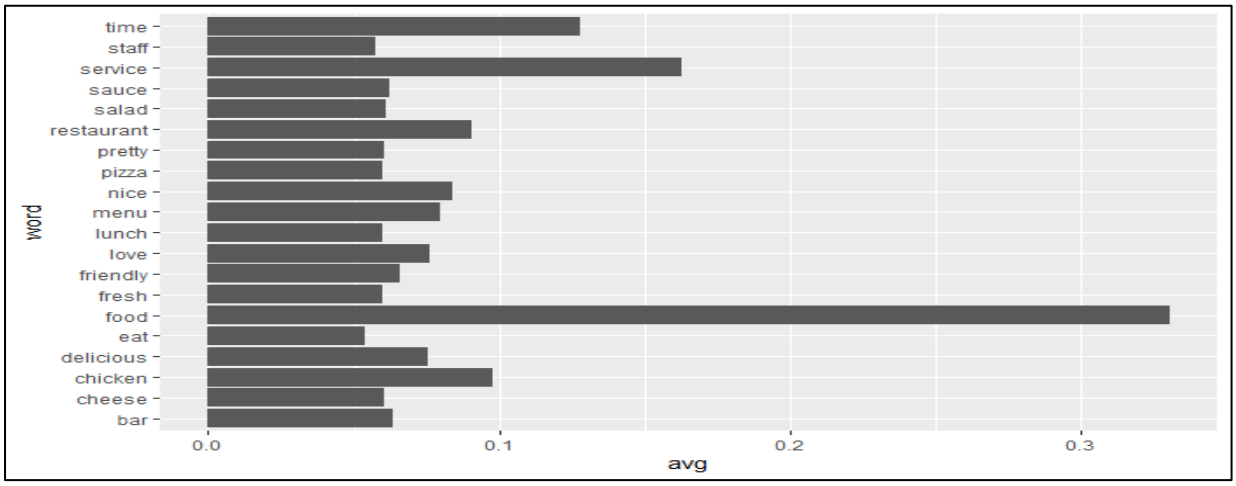


The above graph shows words related with different star ratings. We can see which words are more indicative for each rating based on the proportion of their occurrence within each

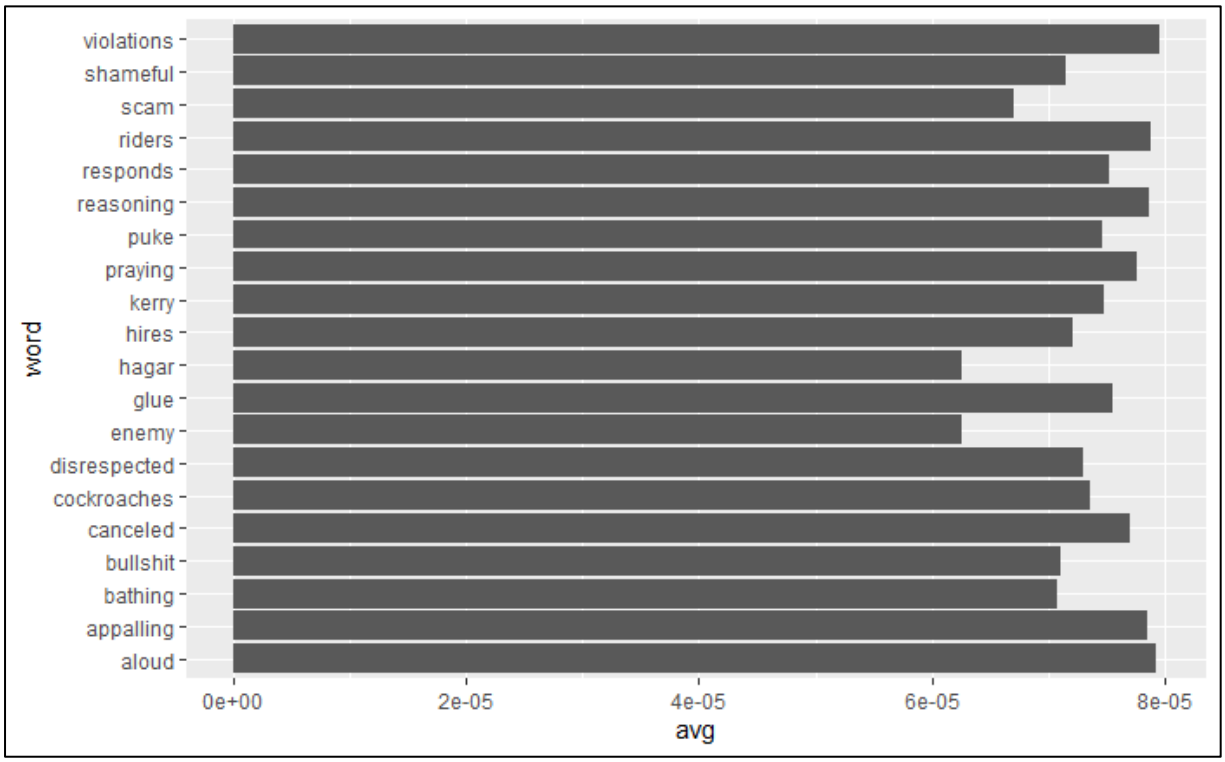
rating. Next, we will look at the average star ratings for each word. For calculating the average star rating associated with each word we take two steps:

- Sum the star ratings associated with reviews where each word occurs in
- And consider the proportion of each word among reviews with a star rating

Top 20 words based on their average STAR ratings



Bottom 20 words based on their average STAR ratings



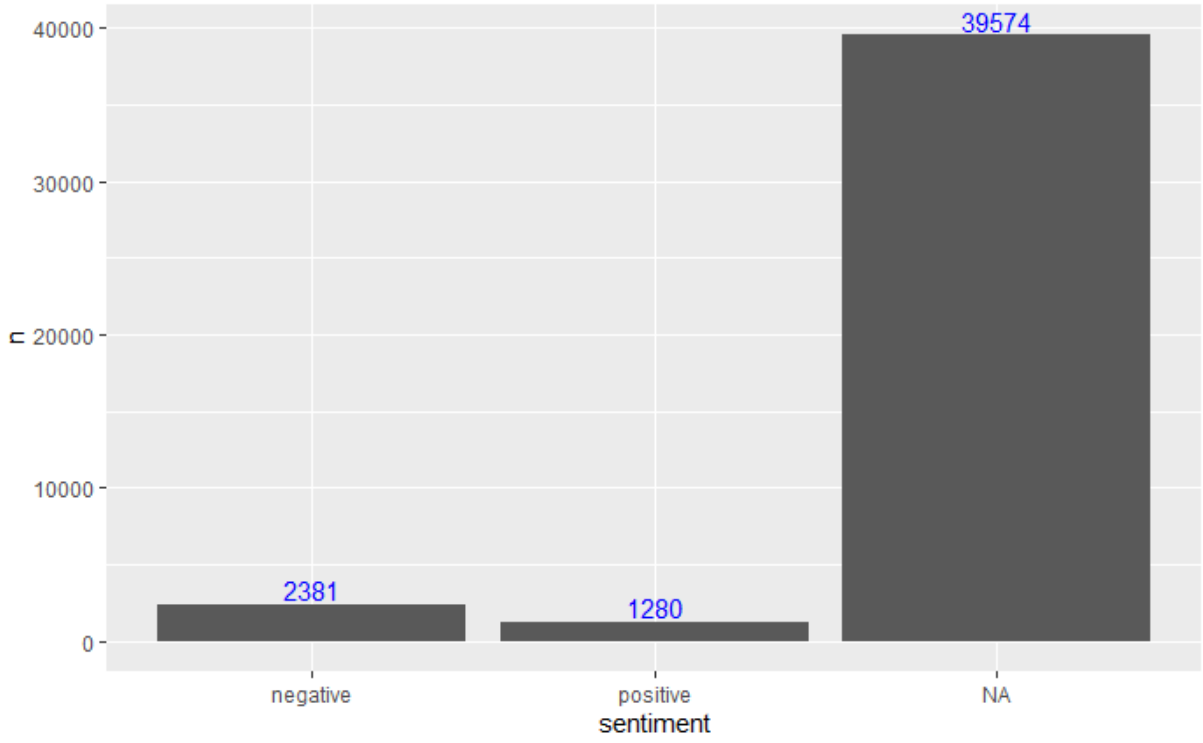
So, the presence of words from top 20 bucket are highly indicative of a positive segment. And the bottom 20 words bucket can be taken as the word's indicative of negative sentiment. These 'positive' and 'negative' words do make sense in the context of user reviews being

considered. The positive words are 'nice', 'fresh', 'delicious' etc. which are used positively in English language. Similarly, the negative words, which are 'violations', 'shameful', 'enemy' are used in a negative sense in English language.

3. **How many matching terms are there for each of the dictionaries? Describe how you obtain predictions based on aggregated scores. Are you able to predict review sentiment based on these aggregated scores, and how do they perform? Does any dictionary perform better?**

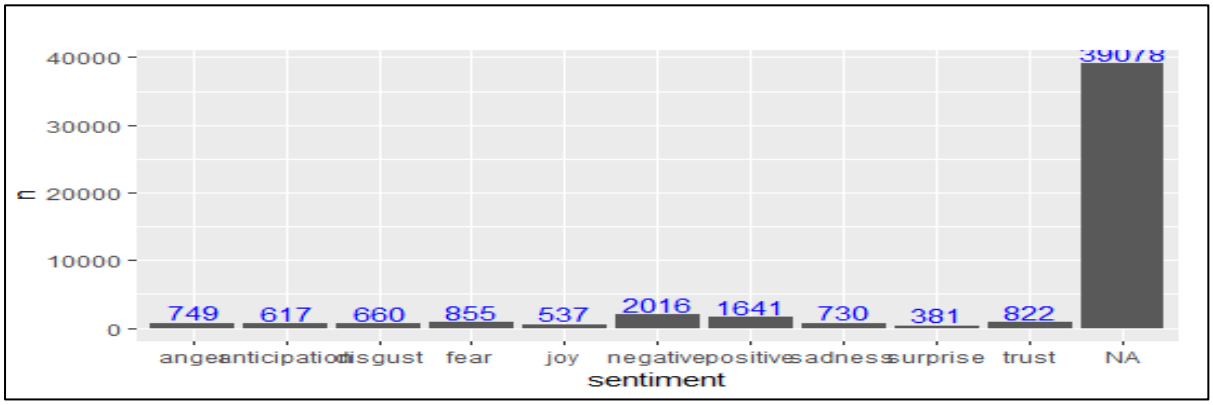
We will consider three dictionaries, available through the tidytext package – the NRC dictionary of terms denoting different sentiments, the extended sentiment lexicon developed by Prof Bing Liu, and the AFINN dictionary which includes words commonly used in user-generated content in the web. The first provides lists of words denoting different sentiment (for eg., positive, negative, joy, fear, anticipation, …), the second specifies lists of positive and negative words, while the third gives a list of words with each word being associated with a positivity score from -5 to +5.

Matching Rate with Bing:



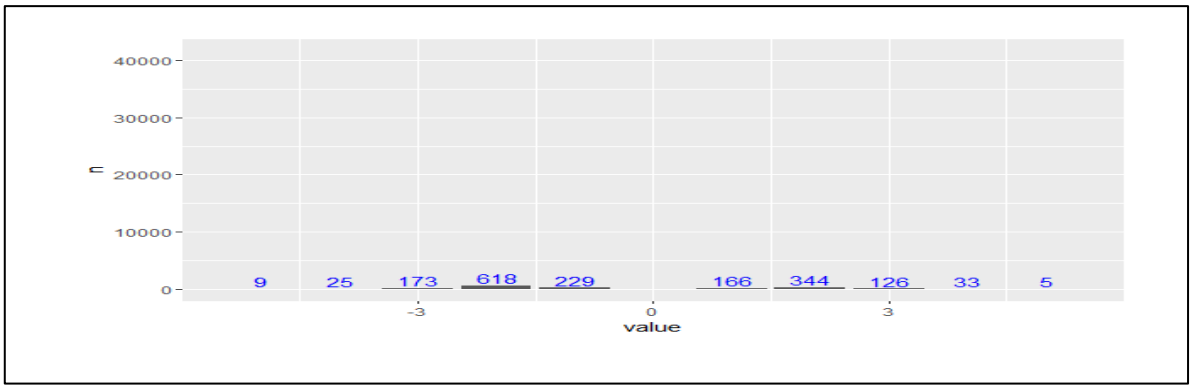
Out of all the distinct words that is in our data after performing data cleaning, there are 3661 words that match with the bing dictionary.

Matching Rate with NRC:



Out of all the distinct words that is in our data after performing data cleaning, there are 4157 words that match with the NRC dictionary.

Matching Rate with Afinn:



The matching rate is lowest for the Afinn dictionary with only 1728 words being matched to different score in Afinn.

Using proportion of occurrence for each words and across documents we calculate the TF-IDF values for each words. Next, we used the scores from different dictionaries to perform prediction on reviews and label them as either 'positive' or 'negative'. Using each dictionary, we obtained an aggregated positiveScore and a negativeScore for each review and for the AFINN dictionary we derived an aggregate positivity score for each review.

For calculating the positive and negative scores for each review we performed few calculations for each words using the three different dictionaries.

For BING: For each words we had calculated its 'occurrence rate'. Next we get our sentiments from bing and then if the sentiment for a particular word is positive we used positive value of the 'occurance rate' for that word. And if the sentiment is negative for a word we took negative value of the 'occurance rate' for that word. Next, for each review, we calculate the number of words within it, the sum of 'occurance rate' for positive sentiments words and the sum of 'occurance rate' for negative sentiments words. We then calculated the positive and negative proportion for each review dividing the sum of 'occurance rate' by number of words

within each review. We then calculated the 'Senti Score' which is the difference between the positive and negative scores. And at the last, for each star rating we take the average positive, negative and senti score.

starsReview	avgPos	avgNeg	avgSentiSc
1	0.3093150	0.6906850	-0.3813701
2	0.4475654	0.5524346	-0.1048691
3	0.6100337	0.3899663	0.2200675
4	0.7552362	0.2447638	0.5104725
5	0.8325914	0.1674086	0.6651827

For NRC: For NRC, we considered {anger, disgust, fear sadness, negative} to denote 'bad' reviews, and {positive, joy, anticipation, trust} to denote 'good' reviews. For each words, we had calculated its 'occurrence rate'. Next we get our sentiments from based on above positive and negative bag created, and then if the sentiment for a particular word is positive we used positive value of the 'occurance rate' for that word. And if the sentiment is negative for a word we took negative value of the 'occurance rate' for that word. Next, for each review, we calculate the number of words within it, the sum of 'occurance rate' for positive sentiments words and the sum of 'occurance rate' for negative sentiments words. We then calculated the positive and negative proportion for each review dividing the sum of 'occurance rate' by number of words within each review. We then calculated the 'Senti Score' which is the difference between the positive and negative scores. And at the last, for each star rating we take the average positive, negative and senti score.

For Afinn: AFINN itself assigns negative to positive sentiment value for words matching the dictionary took the sum of sentiment value for words in a review and then calculated the average senti score for each star ratings.

stars	avgLen	avgSenti
1	5.131027	-2.543253
2	5.148071	0.729226
3	5.060825	3.812583
4	4.940350	6.608025
5	4.399350	7.372450

Based on the above scores that we have created using each dictionaries, we can use it to perform predictions.

Using each of the three dictionaries, for each reviews we have two things: 1. Star Ratings and 2. Senti Score. Using the above two we can make our predictions.

Actual Class: For each review, if its rating is 4 and above, we will classify them as 1 and if the rating is 2 and below we will classify them as -1. We will remove the reviews with 3 star ratings to handle ambiguity. This will be our actual class for each reviews.

Predicted Class: Next, using the senti scores that we calculated, if the senti score > 0, we can predict that review as 1 else 0.

Now, we have our actual class and predicted class for each different reviews using the three different dictionaries. We can look at the confusion matrix for each of the dictionaries.

Using BING:

	Predicted	
Actual	**-1**	**1**
-1	**6371**	**1992**
1	**3440**	**21794**

Using Afinn:

	Predicted	
Actual	**-1**	**1**
-1	**5122**	**3076**
1	**2130**	**22574**

Using NRC:

	Predicted	
Actual	**-1**	**1**
-1	**3185**	**5412**
1	**1711**	**23955**

From above table we can see that the NRC dictionary is performing well in predicting the negative class and the Bing dictionary is doing better than others for positive class predictions.

4. **Develop models to predict review sentiment. For this, split the data randomly into training and test sets. To make run times manageable, you may take a smaller sample of reviews (minimum should be 10,000).**

   One may seek a model built using only the terms matching any or all of the sentiment dictionaries, or by using a broader list of terms (the idea here being, maybe words other than only the dictionary terms can be useful). You should develop at least three different types of models (Naïve Bayes, and at least two others of your choice ....Lasso logistic regression (why Lasso?), xgb, svm, random forest (ranger).

For developing various model to predict review sentiment, we decide to split our cleaned dataset randomly into two parts: Training set and Testing set. We trained our models on the training data using different combinations of model parameters, evaluated the models using AUC and ROC measures and verified it on test set. The split ratio used here as follows,

Training set – 70% | Test Set – 30%

All the models were trained on entire available sample values; no smaller subsets of reviews were used in training or testing.

(i) Develop models using only the sentiment dictionary terms – try the three different dictionaries. How do the dictionaries compare in terms of predictive performance? Then with a combination of the three dictionaries, i.e., combine all dictionary terms. Do you use term frequency, tfidf, or other measures, and why? What is the size of the document-term matrix? Should you use stemming or lemmatization when using the dictionaries?

For developing all our models, we decide to use the 'tf-idf' measure since it provides a 'balanced' statistical measure that considers the importance of a word in document with respect to the collection of documents. Term frequency (tf) measures how frequently a word occurs in a document but fails to capture importance of words. On the other hand, inverse document frequency (idf) is a measure that decreases the weight for commonly used words and increases the weight for words that are not used very much in a collection of documents but fails to give information about how important the word is to a given document. As 'tf-idf' measure is product of 'tf' an 'idf', it gives more weightage to the word that is rare withing the documents while providing more importance to the word that is more frequent in the document.

We use Lemmatization technique since we are using dictionaries to filter out tokens. Stemming algorithms work by cutting off the end or the beginning of the word, considering a list of common prefixes and suffixes. It works well in most cases, but at times can result into root/base forms that aren't real words. For example, stemming the word 'easily' results into 'easili' and this would create problems specially when dictionaries are used for filtering tokens. Lemmatization fixes that issue, as it returns the base or dictionary form of a word (known as the *lemma*) by using of a vocabulary and morphological analysis of words.

Finally, after lemmatization, we filter out tokens based on three dictionaries - Bing, NRC and AFINN and their combination. Further, we filter out all reviews that are rated '3 stars' and create our dependable variable with negative class (-1) being all reviews rated '1 star' or '2 stars' and positive class (1) being all reviews rated '4 stars' or '5 stars'. To keep things clear, we created four different document term matrices, one for each of three dictionaries and their combination. The sizes of these document term matrices are as below,

	Observations	Variables	Positive Class	Negative Class
Bing	33597	1130	25234	8363
NRC	34264	1558	25667	8597
AFINN	32902	621	24704	8198
Combined	34420	2103	25799	8621

Using split ratio = 0.7, these four document term matrices are further divided into their corresponding training and testing sets. Since the ratio of negative to positive class is about 30-40%, we use a threshold of 0.4 to evaluate our models. Now, we build three different types of models as follows,

I) Random Forest

To find the best model, we did a full-grid search across range of values for following hyper-parameters and their values:

mtry -> values (SQRT(M) - 3, SQRT(M), SQRT(M) + 3)

The default value of 'mtry' for classification is sqrt (number of features = M) and we investigated in range +/– 3 from SQRT(M).

num.tress -> values (100, 200, 300, 400, 500)

The default value for 'num.trees' is set to 500, and we investigated in range 100 to 500. Anything with N > 500 for would not sufficiently decline OOB error and may potentially lead to overfitting.

The total number of hyper-parameter combinations are **15**. We trained a random forest model using ranger by iterating through each value combination in grid and stored the aggregate OOB-error for that model.

```
gridSearchRF <- function(mtry.values, trn.data.set){
  search_grid_rf <- expand.grid(
    mtry      = mtry.values,
    num.trees  = seq(100, 500, by = 100),
    OOB.error  = 0
  )

for(i in 1:nrow(search_grid_rf)) {
    rfModel <- ranger(
      dependent.variable.name = "hiLo",
      data       = trn.data.set,
      mtry       = search_grid_rf$mtry[i],
      num.trees   = search_grid_rf$num.trees[i],
      probability = TRUE
    )
    search_grid_rf$OOB.error[i] <- rfModel$prediction.error
  }

  return(search_grid_rf %>% dplyr::arrange(OOB.error))
}
```

We call above function to get the best model parameters and then choose the top row of parameter values to build our final model.

```
rfModelGridSearch <- gridSearchRF(seq(mtry.values, by = 3), trn.data.set)

rfModel <- ranger(dependent.variable.name = "hiLo", data= trn.data.set, mtry =
rfModelGridSearch[1,"mtry"], num.trees = rfModelGridSearch[1,"num.trees"], probability =
TRUE)
```

(a) <u>Model 1: Bing</u>

The grid search resulted into best model with parameters as mtry = 30, num.trees = 500 and evaluation results are as follows,

Train Data:

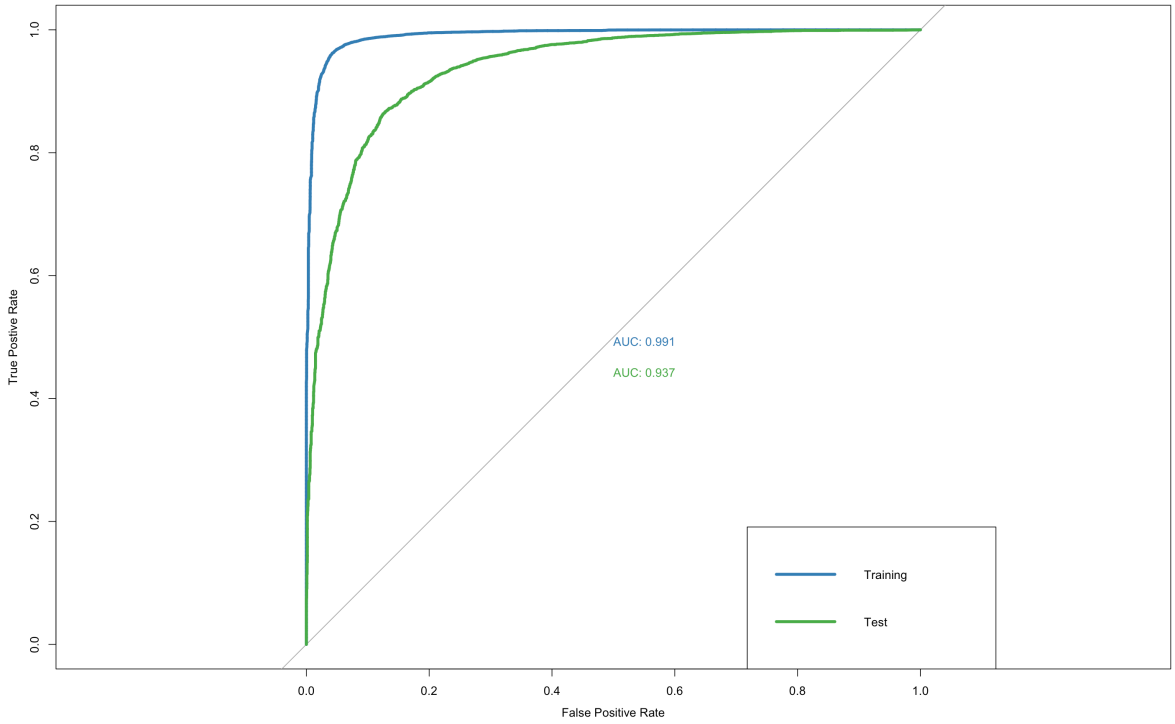
	Predicted	
Actual	**Negative**	**Positive**
Negative	4671	1135
Positive	96	17615

Accuracy = 0.9477 Precision = 0.9395 Recall = 0.9946

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1545	1012
Positive	185	7338

Accuracy = 0.8813 Precision = 0.8788 Recall = 0.9754



(b) <u>Model 2: NRC</u>

The grid search resulted into best model with parameters as mtry = 39, num.trees = 400 and evaluation results are as follows,

Train Data:

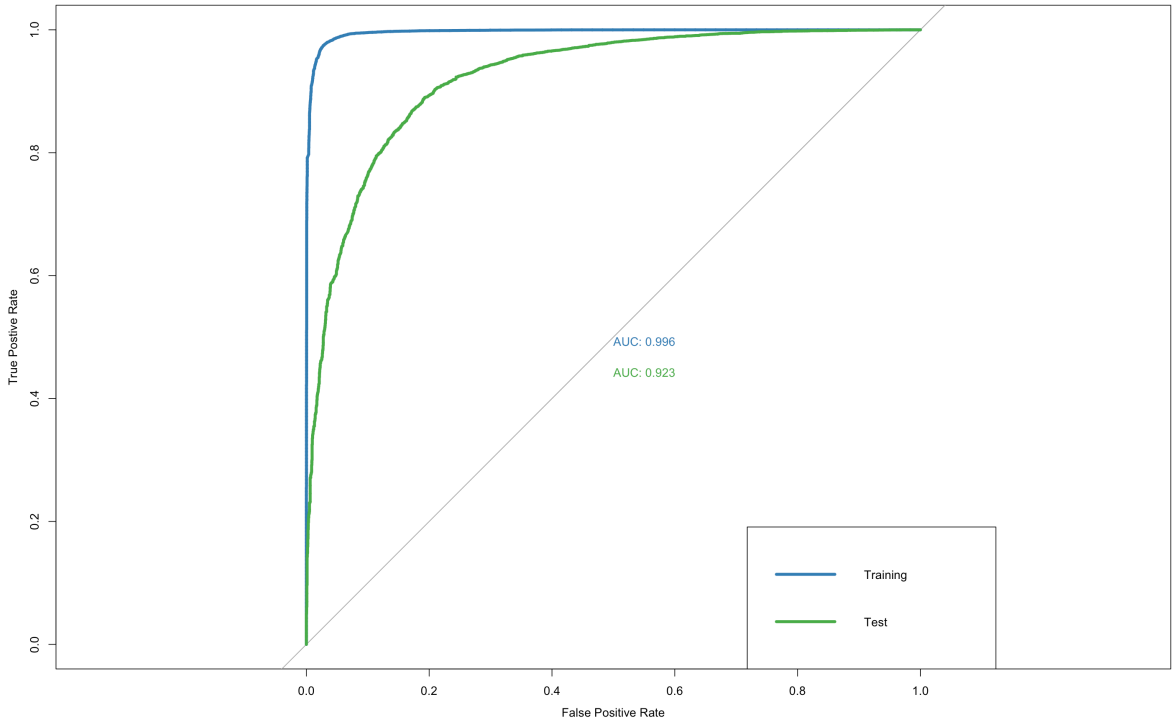
	Predicted	
Actual	**Negative**	**Positive**
Negative	4946	1075
Positive	32	17931

Accuracy = 0.9538 Precision = 0.9434 Recall = 0.9982

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1375	1201
Positive	192	7512

Accuracy = 0.8645 Precision = 0.8622 Recall = 0.9751



(c) Model 3: AFINN

The grid search resulted into best model with parameters as mtry = 23, num.trees = 500 and evaluation results are as follows,

Train Data:

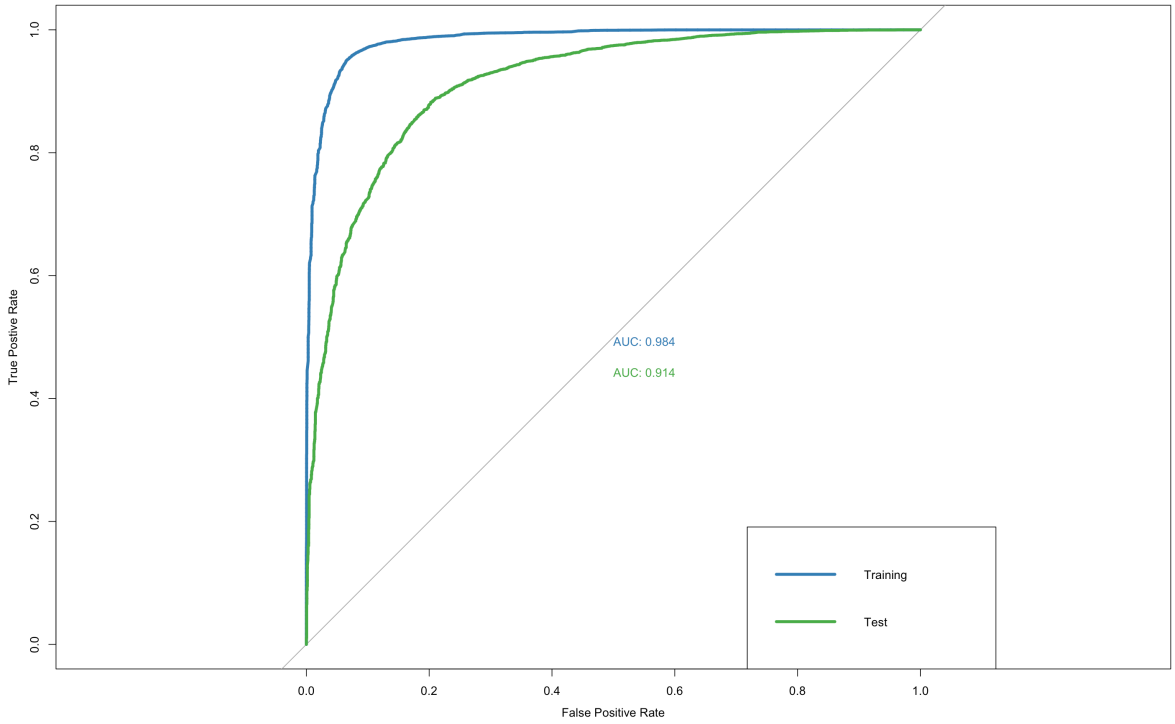
	Predicted	
Actual	**Negative**	**Positive**
Negative	4388	1362
Positive	167	17114

Accuracy = 0.9336 Precision = 0.9263 Recall = 0.9903

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1378	1070
Positive	277	7146

Accuracy = 0.8635 Precision = 0.8698 Recall = 0.9627



(d) <u>Model 4: Combined</u>

The grid search resulted into best model with parameters as mtry = 49, num.trees = 500 and evaluation results are as follows,

Train Data:

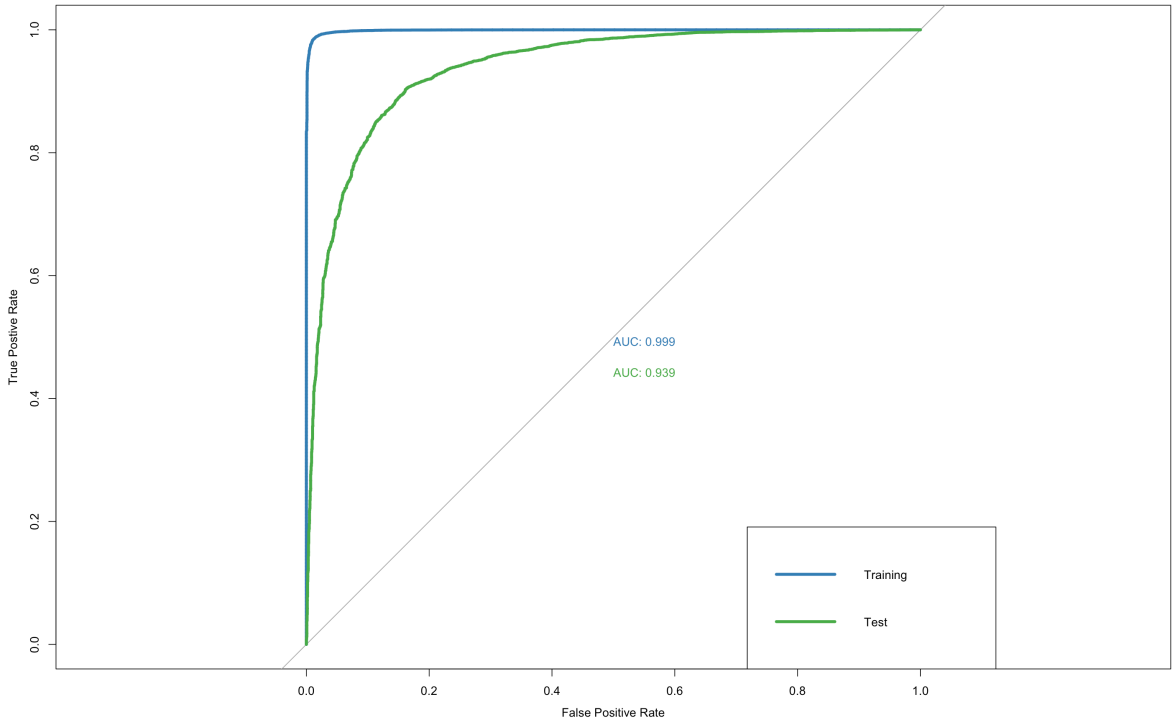
	Predicted	
Actual	**Negative**	**Positive**
Negative	5312	724
Positive	14	18044

Accuracy = 0.9694 Precision = 0.9614 Recall = 0.9992

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1485	1100
Positive	165	7576

Accuracy = 0.8775 Precision = 0.8732 Recall = 0.9787



Takeaway –

We observe that random forest performs quite well in classification of sentiment, irrespective of the dictionary used. It exhibits excellent performance metric values for accuracy, precision, and recall. ROC curve also confirms that the classifier maintains a high true positive rate while also having a low false positive rate, with AUC > 90% with all models. With text classification, we are not interested how good the model classifies any class of interest. Thus, with an imbalanced class distribution and more focus on precision and recall, F-score is a better metric to compare performance.

Model	F-Score
RF Bing	0.9246
RF NRC	0.9152
RF AFINN	0.9139
RF Combined	0.9229

II) Naïve Bayes

For Naïve Bayes, we follow the same approach and perform a grid search across hyper-parameters to decide our best model for each case.

```
search_grid_nb <- expand.grid(
  usekernel = c(TRUE, FALSE),
  laplace = 0:3,
  adjust = seq(0, 3, by = 0.5)
)
```

We then use caret library's train function to build our models across the grid parameters and additionally perform 10-fold cross validation to find the best tune.

```
nbModel <- train(
  hiLo ~ .,
  data = trn.data.set,
  method = "naive_bayes",
  trControl = trainControl(method='cv',number=10),
  tuneGrid = search_grid_nb,
)
```

Our final model can be taken from nbModel$finalModel and the corresponding hyper-parameters can be taken from nbModel$bestTune.

(a) <u>Model 1: Bing</u>

The grid search resulted into best model with parameters as laplace = 0, usekernel = TRUE, adjust = 0.5 and evaluation results are as follows,

Train Data:

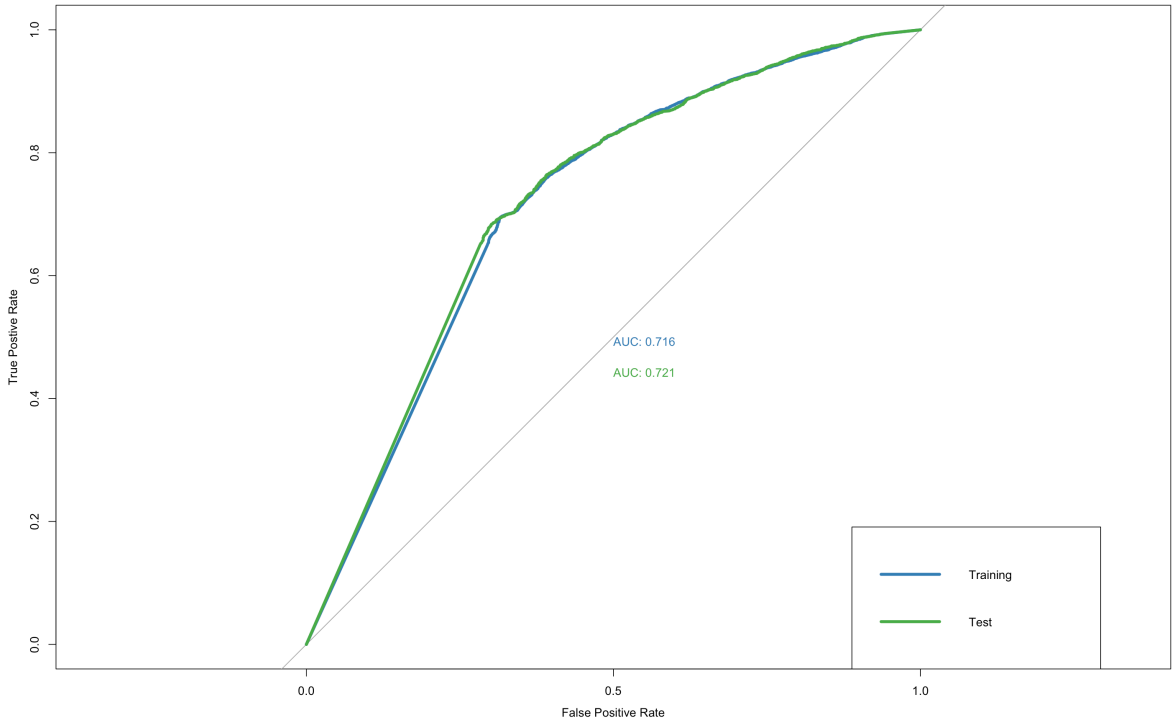
	Predicted	
Actual	**Negative**	**Positive**
Negative	3698	2216
Positive	4570	13033

Accuracy = 0.7114 Precision = 0.8547 Recall = 0.7404

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1534	915
Positive	1966	5665

Accuracy = 0.7142 Precision = 0.8609 Recall = 0.7424



(b) <u>Model 2: NRC</u>

The grid search resulted into best model with parameters as laplace = 0, usekernel = TRUE, adjust = 0.5 and evaluation results are as follows,

Train Data:

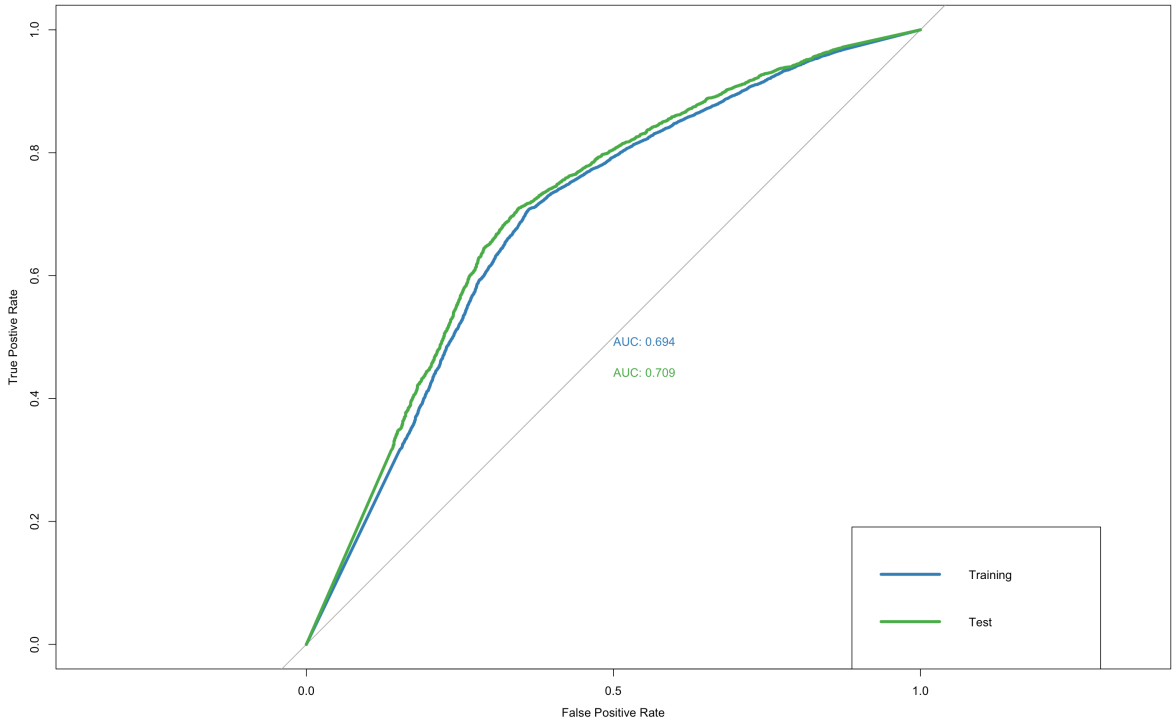
	Predicted	
Actual	**Negative**	**Positive**
Negative	4944	1060
Positive	11442	6538

Accuracy = 0.4787 Precision = 0.8605 Recall = 0.3636

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	2180	413
Positive	4865	2822

Accuracy = 0.4866 Precision = 0.8723 Recall = 0.3671



(c) Model 3: AFINN

The grid search resulted into best model with parameters as laplace = 0, usekernel = FALSE, adjust = 0 and evaluation results are as follows,

Train Data:

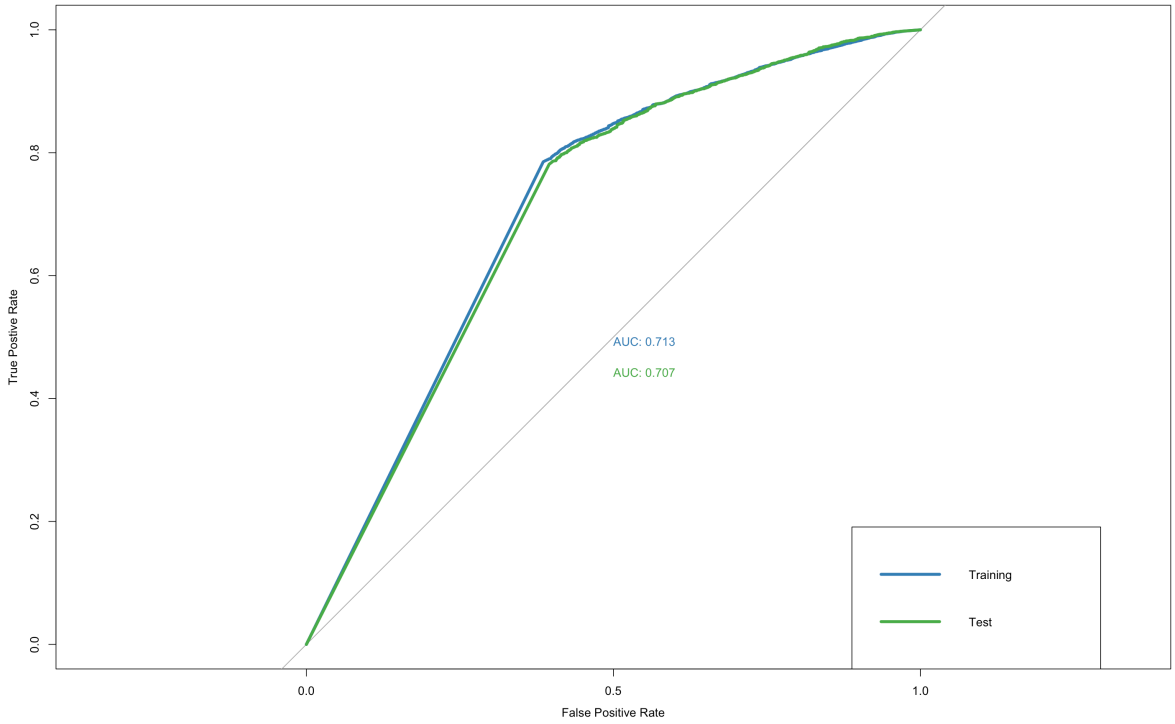
	Predicted	
Actual	**Negative**	**Positive**
Negative	3014	2704
Positive	2892	14421

Accuracy = 0.757 Precision = 0.8421 Recall = 0.8333

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1266	1214
Positive	1235	6156

Accuracy = 0.7519 Precision = 0.8353 Recall = 0.8329



(d) <u>Model 4: Combined</u>

The grid search resulted into best model with parameters as laplace = 0, usekernel = TRUE, adjust = 0.5 and evaluation results are as follows,

Train Data:

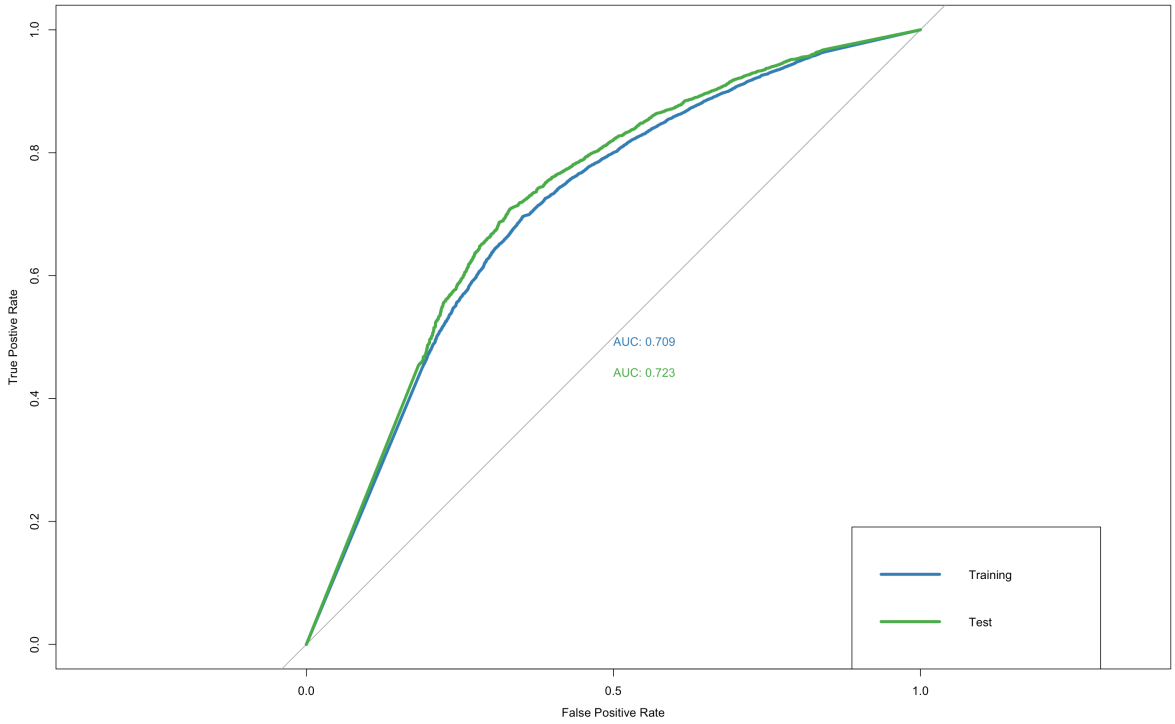
	Predicted	
Actual	**Negative**	**Positive**
Negative	4735	1289
Positive	8975	9095

Accuracy = 0.5740 Precision = 0.8759 Recall = 0.5033

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	2058	539
Positive	3779	3950

Accuracy = 0.5818 Precision = 0.8799 Recall = 0.5111



Takeaway -

We observe that naïve bayes classifier doesn't perform well for the given data. Furthermore, the performance highly varies with the dictionary used. The prominent reasons for relatively poor performance and high variation of results can be,

- Model works with an assumption of independence among predictors. In simple terms, a naïve bayes classifier assumes that the presence of a particular feature/variable in a class is unrelated to the presence of any other feature, it gives equal importance to each feature in the data set.

- Zero values, i.e., if the variable (here word tokens) *tf-idf* value is not seen in training data set then model assigns a zero probability to that category and then it is difficult to perform prediction. This can also be verified from the fact that the dictionary-based datasets having a smaller number of variables (like Bing, Afinn) outperforms the ones with larger number of variables (like NRC, Combined) due to the presence of less zero *tf-idf* values and less sparsity.

Model	F-Score
NB Bing	0.7973
NB NRC	0.5167
NB AFINN	0.8341
NB Combined	0.6466

III) Lasso Logistic Regression

We use LASSO regression because of its simple implementation and interpretation. It solves the problem of dimensionality that complicates textual analysis because inputs (tokens) are too numerous compared to a standard size of sample. It penalizes those variables with large coefficients by adding a penalty term in the process of maximizing the log-likelihood. With a proper weight (λ) chosen, LASSO assigns zeros to some coefficients and helps reducing the dimensions significantly.

For LASSO regression models, we first use 'cv.glmnet' that does k-fold cross-validation for 'glmnet' and returns values of lambda used in the fits. To control the coefficients by adding "L1" penalty, we set alpha = 1.

```
cvglmModel <- cv.glmnet(data.matrix(x_trn), y_trn, family = "binomial", alpha = 1)
```

We filter out the variables (here tokens) that had non-zero coefficients after full LASSO regression with lambda.min as λ and remove these tokens from our train and test data to reduce dimensionality.

```
nzCoef <- tidy(coef(cvglmModel, s=cvglmModel$lambda.min))
nzCoefVars <- nzCoef[-1,1]
x_trn_filtered <- x_trn %>% select(nzCoefVars)
x_tst_filtered <- x_tst %>% select(nzCoefVars)
```

Our final model is then built using 'glm' on filtered data,

```
glmModel <- glm(formula = hiLo ~ ., data = x_trn_filtered, family = "binomial")
```

(a) Model 1: Bing

The evaluation results for our best model using 'glm' are as follows,

Train Data:

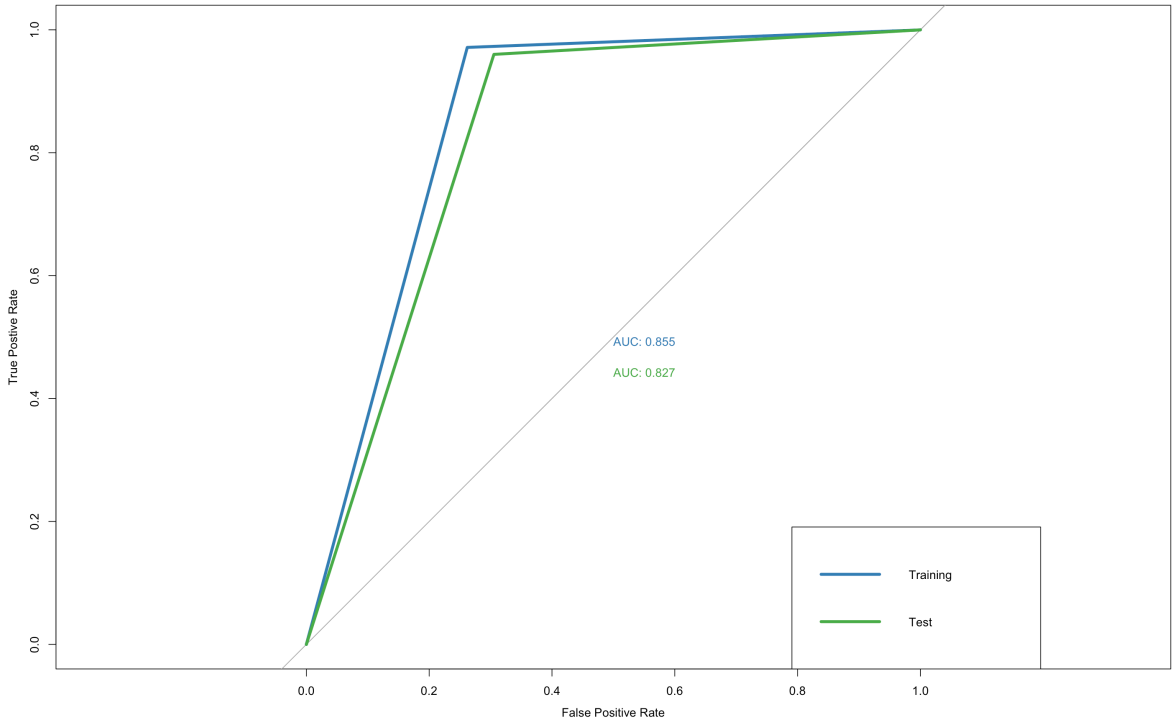
	Predicted	
Actual	**Negative**	**Positive**
Negative	4364	1550
Positive	504	17099

Accuracy = 0.9127 Precision = 0.9169 Recall = 0.9714

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1701	748
Positive	305	7326

Accuracy = 0.8955 Precision = 0.9074 Recall = 0.96



(b) <u>Model 2: NRC</u>

The evaluation results for our best model using 'glm' are as follows,

Train Data:

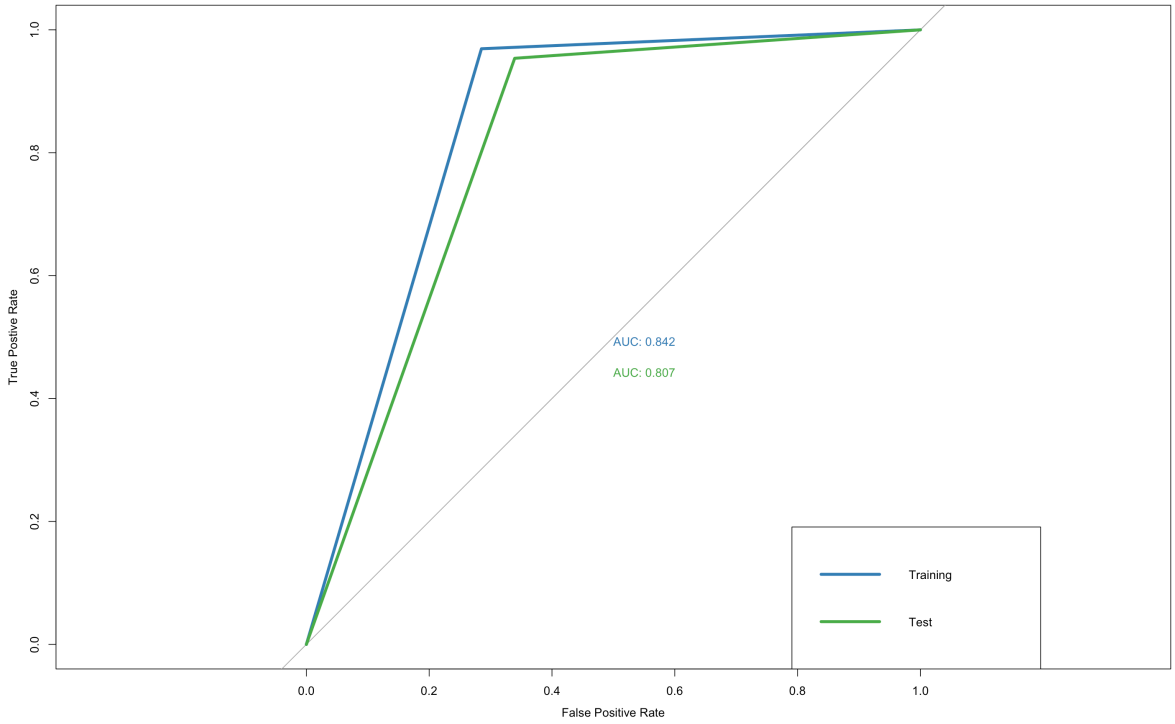
	Predicted	
Actual	**Negative**	**Positive**
Negative	4292	1712
Positive	554	17426

Accuracy = 0.9055 Precision = 0.9105 Recall = 0.9692

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1713	880
Positive	356	7331

Accuracy = 0.8798 Precision = 0.8928 Recall = 0.9537



(c) Model 3: AFINN

The evaluation results for our best model using 'glm' are as follows,

Train Data:

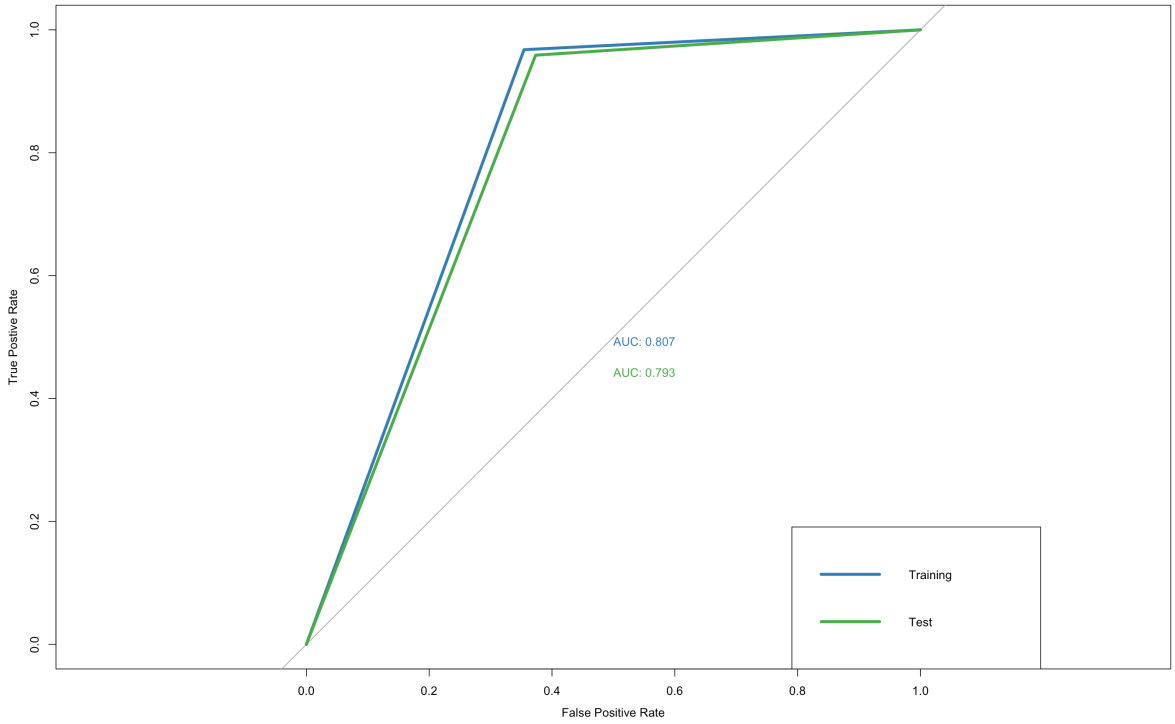
	Predicted	
Actual	**Negative**	**Positive**
Negative	3691	2027
Positive	560	16753

Accuracy = 0.8877 Precision = 0.8921 Recall = 0.9677

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1554	926
Positive	305	7086

Accuracy = 0.8753 Precision = 0.8844 Recall = 0.9587



(d) <u>Model 4: Combined</u>

The evaluation results for our best model using 'glm' are as follows,

Train Data:

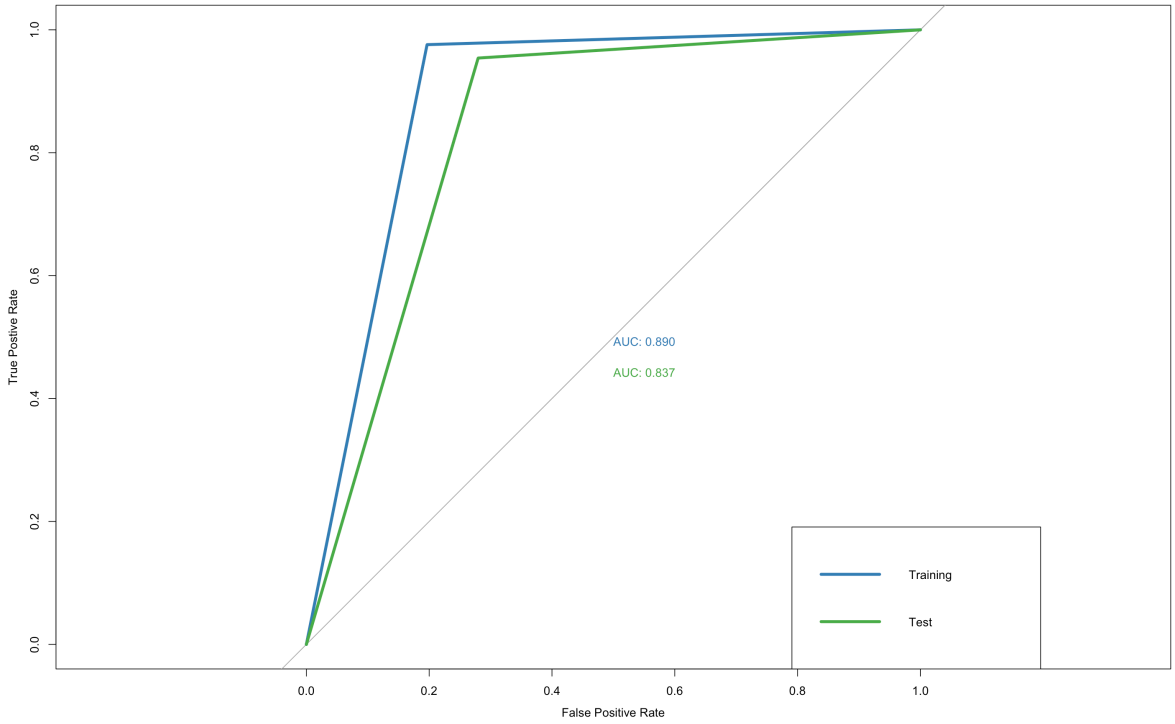
	Predicted	
Actual	**Negative**	**Positive**
Negative	4839	1185
Positive	436	17634

Accuracy = 0.9327 Precision = 0.937 Recall = 0.9759

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1870	727
Positive	355	7374

Accuracy = 0.8952 Precision = 0.9103 Recall = 0.9541



Takeaway -

We observe that LASSO logistic regression performs quite well in classification of sentiment as well, irrespective of the dictionary used. It exhibits excellent performance metric values for accuracy, precision, and recall. ROC curve also confirms that the classifier maintains a high true positive rate while also having a low false positive rate, but a slightly lower AUC (> 80% with all models) as compared to random forest models.

Model	F-Score
LR Bing	0.933
LR NRC	0.9222
LR AFINN	0.9201
LR Combined	0.9317

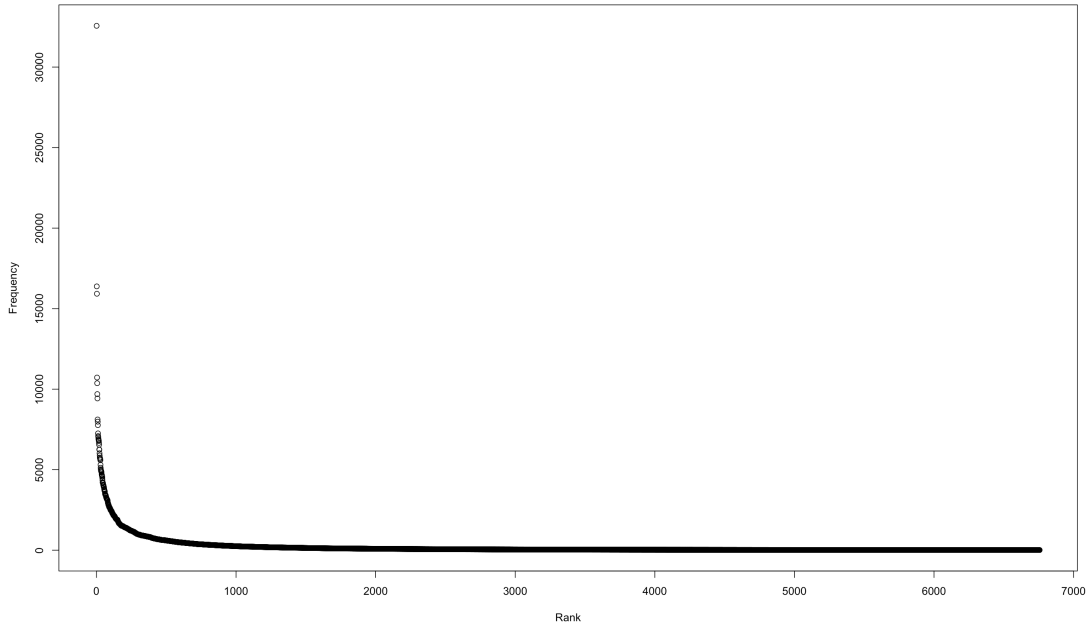
(ii) Develop models using a broader list of terms (i.e., not restricted to the dictionary terms only) – how do you obtain these terms? Will you use stemming here? Report on performance of the models. Compare performance with that in part (c) above. How do you evaluate performance? Which performance measures do you use, why?

To decide on broader list of terms, we first explore our tokens by calculating their frequency and arranging them from highest frequency (rank 1) to lowest frequency (rank 6757).

```
rWords <- revTokens_lemm %>% group_by(word) %>% summarise(freq=n()) %>% arrange(desc(freq))
rWords <- rWords %>% mutate(rank=seq(1, nrow(rWords)))
```

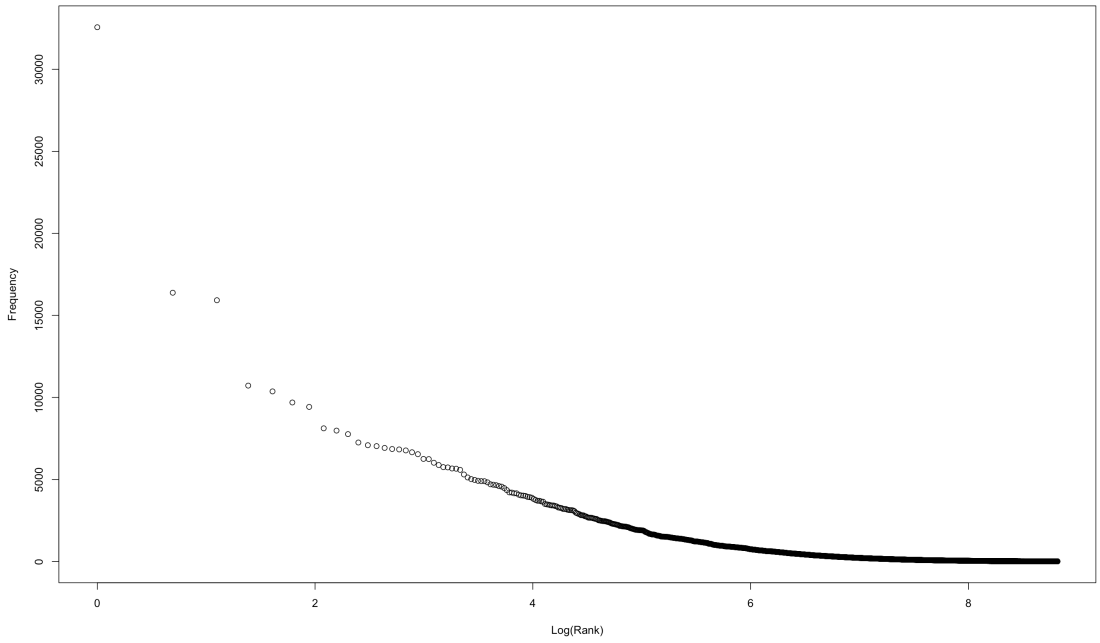
The frequency vs rank plot is as seen as follows,

plot(rWords$rank, rWords$freq, xlab = "Rank" , ylab = "Frequency")



As expected, the distribution follows **Zipf's law** (frequencies of words are inversely proportional to their rank). To better understand the relation, we plot frequency vs log(Rank) as follows,

rWords <- rWords %>% mutate(logRank=log(seq(1, nrow(rWords))))
plot(rWords$logRank, rWords$freq, xlab = "Log(Rank)" , ylab = "Frequency")



In text analysis, words with very high and very low frequencies do not provide significant information to the model, hence we should remove them. From above plot, we can see frequency to be fairly stable in region log(Rank) = 3 to log(Rank) = 8 hence we use this criteria to filter out words.

reduced_rWords <- rWords %>% filter(between(logRank, 3, 8))

The final broader list of words can be obtained by joining these reduced set of words with original tokens and reduce from **6757** words to **2960** words.

reduced_revTokens <- left_join(reduced_rWords, revTokens_lemm)

With a broader list of words too we use lemmatization since is a lot more powerful. It looks beyond word reduction and considers a language's full vocabulary to apply a morphological analysis to words, aiming to remove inflectional endings only and to return the base. Also, after lemmatization the result will always be another dictionary item (infinitives, singular forms…), and not a "stem", which sometimes can be difficult to define and interpret if needed.

Now, as before, we filter out all reviews that are rated '3 stars' and create our dependable variable with negative class (-1) being all reviews rated '1 star' or '2 stars' and positive class (1) being all reviews rated '4 stars' or '5 stars'. The size of the document term matrix for broader list of words is as below,

	Observations	Variables	Positive Class	Negative Class
Broader List of Words	34513	2960	25869	8644

Using split ratio = 0.7, the document term matrix is further divided into corresponding training and testing sets. Now, we build three different types of models as follows,

I) Random Forest

To find the best model, we did a full-grid search across range of values for following hyper-parameters and their values, by using the same function and steps as done for models based on different dictionaries previously. The grid search resulted into best model with parameters as mtry = 57, num.trees = 500 and evaluation results are as follows,

Train Data:

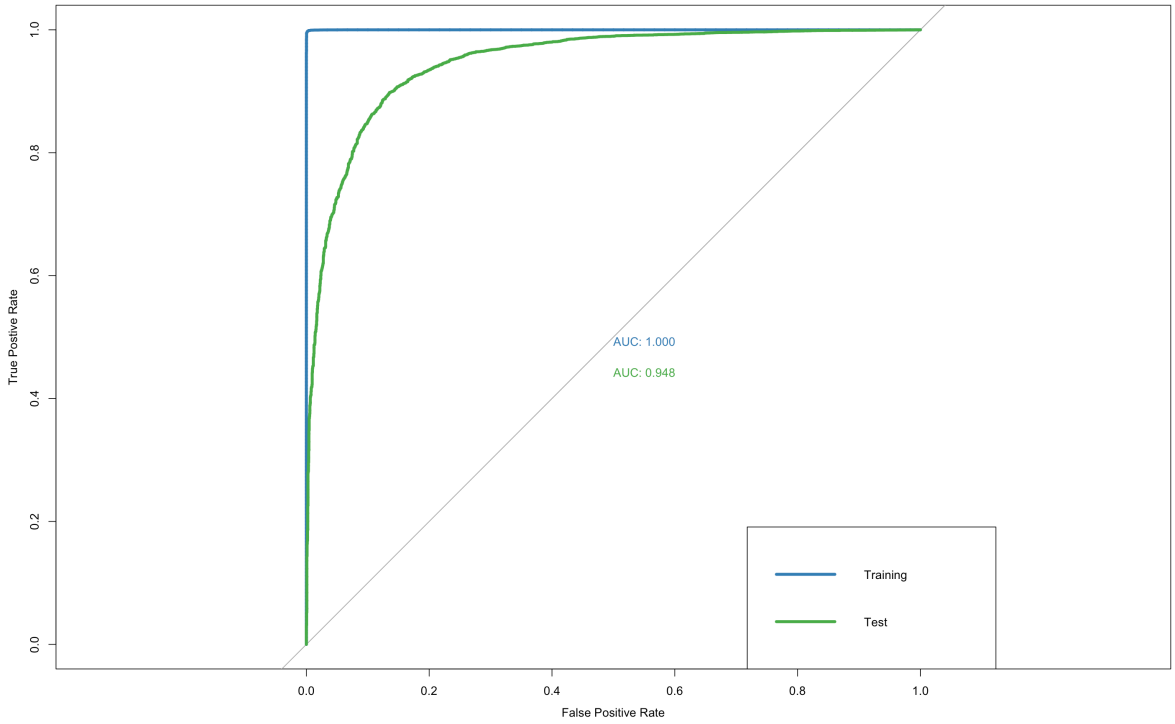
	Predicted	
Actual	**Negative**	**Positive**
Negative	5596	405
Positive	0	18158

Accuracy = 0.9832 Precision = 0.9782 Recall = 1

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1503	1140
Positive	116	7595

Accuracy = 0.8787 Precision = 0.8695 Recall = 0.985



From above ROC plot, we can see that the model is overfitting on training data. To reduce overfitting, we need to further tune additional parameters to introduce more bias and reduce variance.

mtry: Reduce the number of variables to sample at a time introduces bias, hence we make the mtry value to half of our current value as **27**.

sample.fraction: Reduce the number of samples to train on. The default value is 63.25% of the training set since this is the expected value of unique observations in the bootstrap sample. Lower sample sizes may introduce more bias, and hence we set it to **0.5**.

min.node.size: Increase the minimum number of samples within the terminal nodes resulting into shallower trees, hence we set it to **5**.

num.trees: Reduce the number of trees grown to handle complexity of model and in turn reduce variance, hence make the num.trees value to half of our current value as **250**.

rfModel5Tuned <- ranger(dependent.variable.name = "hiLo", data = broader_words_trn, mtry = 27, num.trees = 250, probability = TRUE, min.node.size = 5, sample.fraction = 0.5)

Furthermore, for evaluation of results as well, we change our prediction threshold from **0.4** to **0.5**, and the results are as follows,

Train Data:

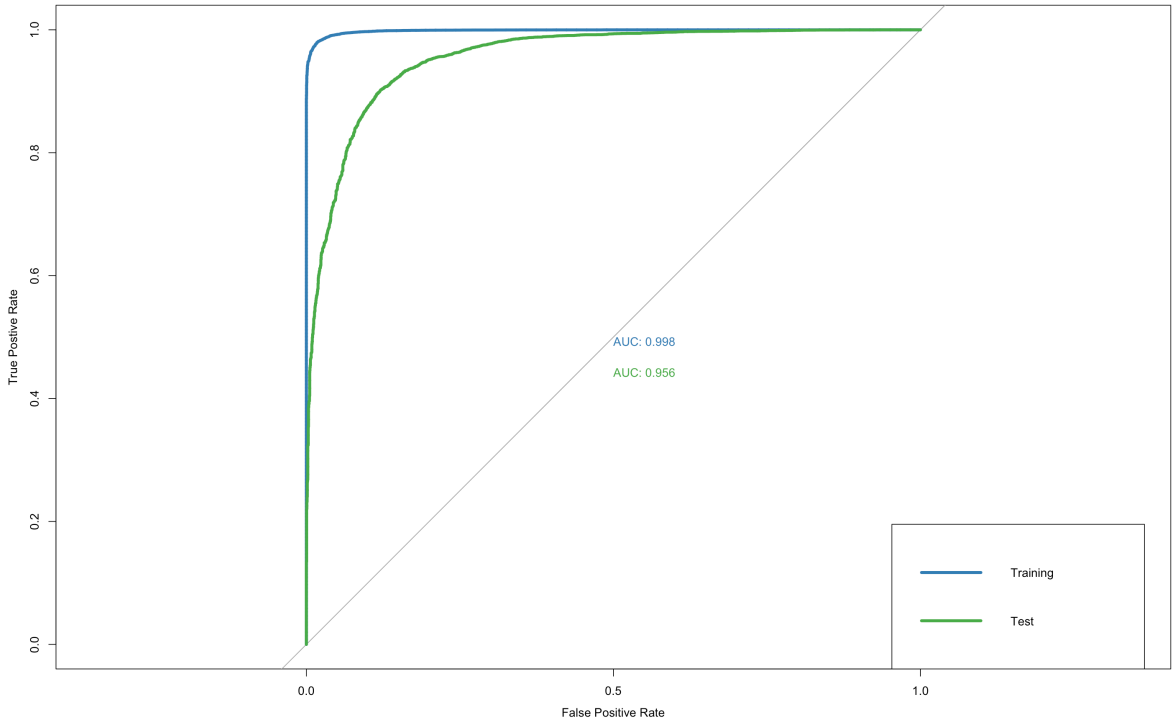
	Predicted	
Actual	**Negative**	**Positive**
Negative	5361	706
Positive	35	18057

Accuracy = 0.9693 Precision = 0.9624 Recall = 0.9981

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	1653	924
Positive	103	7674

Accuracy = 0.9008 Precision = 0.8925 Recall = 0.9868



Takeaway –

As expected, we observe that random forest again performs quite well in classification of sentiment with broader set of terms, slightly better than dictionary words and their combination. It exhibits excellent performance metric values for accuracy, precision, and recall. ROC curve also confirms that the classifier maintains a high true positive rate while having a low false positive rate with AUC > 95%.

Model	F-Score
RF Broader	0.9373

II) Naïve Bayes

For Naïve Bayes, we follow the same approach and perform a grid search across hyper-parameters to decide our best model for each case. The grid search resulted into best model with parameters as laplace = 0, usekernel = TRUE, adjust = 1.5 and evaluation results are as follows,

Train Data:

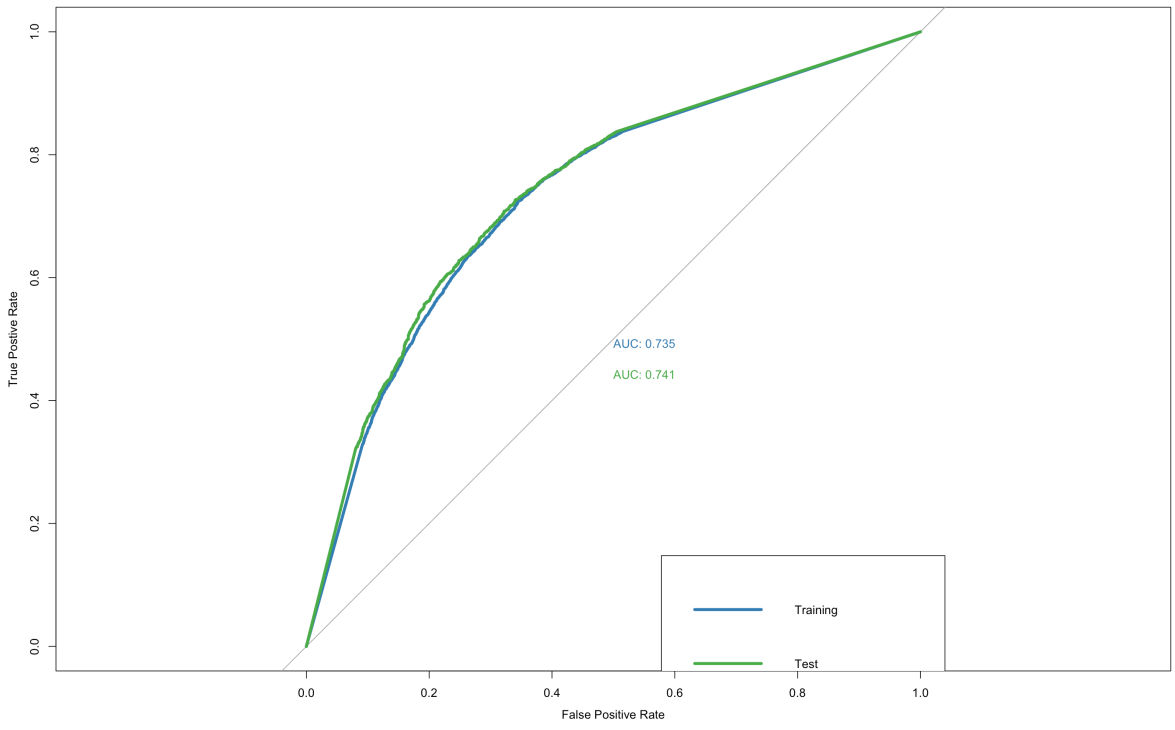
	Predicted	
Actual	**Negative**	**Positive**
Negative	5399	602
Positive	11750	6408

Accuracy = 0.4887 Precision = 0.9141 Recall = 0.3529

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	2403	240
Positive	5031	2680

Accuracy = 0.4909 Precision = 0.9178 Recall = 0.3476



Takeaway -

We observe that naïve bayes classifier doesn't perform well for the given data. An assumption of independence among predictors and high sparsity in data resulting with more zero values for *tf-idf* are the prominent reasons for relatively poor performance.

Model	F-Score
NB Broader	0.5042

III) Lasso Logistic Regression

Similar to previous models, we first use 'cv.glmnet' that does k-fold cross-validation for 'glmnet', then filter out the variables (tokens) that had non-zero coefficients after full LASSO regression with lambda.min as λ. These non-zero coefficients are removed from our train and test data and our final model is then built using 'glm' on filtered data,

The evaluation results for our best model using 'glm' are as follows,

Train Data:

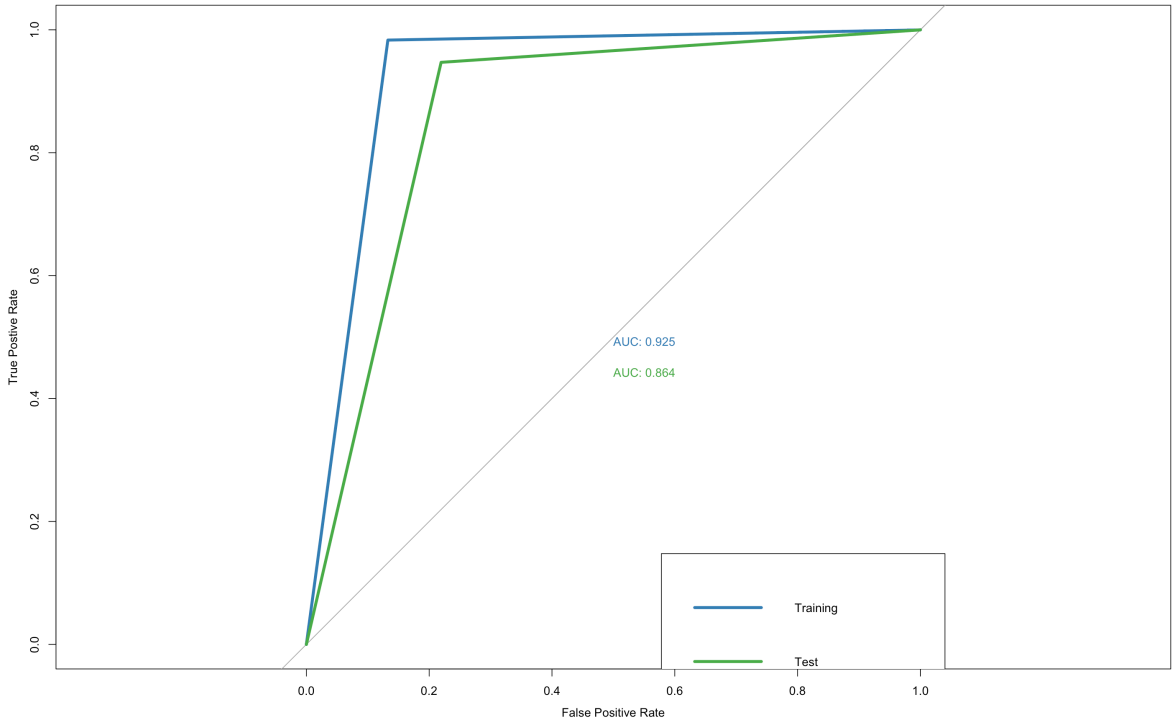
	Predicted	
Actual	**Negative**	**Positive**
Negative	5204	797
Positive	303	17855

Accuracy = 0.9545 Precision = 0.9573 Recall = 0.9833

Test Data:

	Predicted	
Actual	**Negative**	**Positive**
Negative	2063	580
Positive	408	7303

Accuracy = 0.9046 Precision = 0.9264 Recall = 0.9471



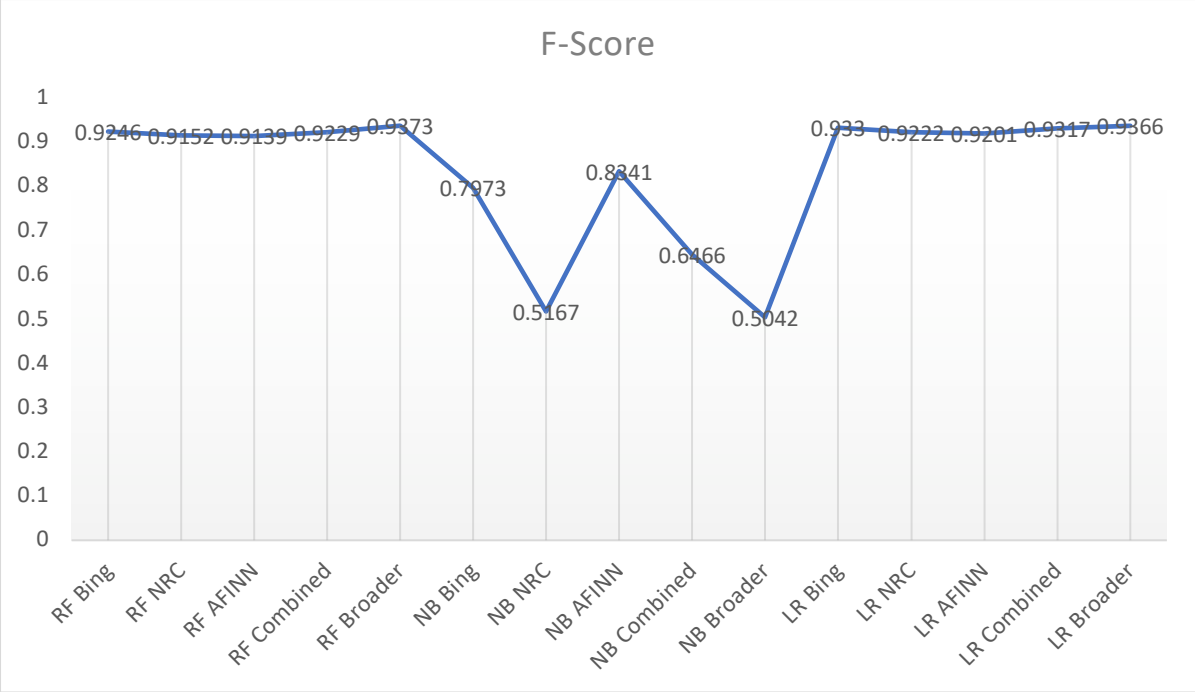
Takeaway –

We observe that LASSO logistic regression with broader set of terms performs quite well in classification of sentiment as well. It exhibits excellent performance metric values for accuracy, precision, and recall. ROC curve also confirms that the classifier maintains a high true positive rate while also having a low false positive rate and improved AUC > 85%. Here, even though number of variables are increased with broader set of terms, the LASSO penalty on the original dataset shrinks down less significant variables and thus final model performs well with filtered set of variables having non-zero coefficients only.

Model	F-Score
LR Broader	0.9366

Using F-score as a metric, we now compare all models, and our best model is **random forest with broader set of terms**.

Model	F-score	Model	F-score	Model	F-score
RF Bing	0.9246	NB Bing	0.7973	LR Bing	0.933
RF NRC	0.9152	NB NRC	0.5167	LR NRC	0.9222
RF AFINN	0.9139	NB AFINN	0.8341	LR AFINN	0.9201
RF Combined	0.9229	NB Combined	0.6466	LR Combined	0.9317
RF Broader	0.9373	NB Broader	0.5042	LR Broader	0.9366



5. **Consider some of the attributes for restaurants – this is specified as a list of values for various attributes in the 'attributes' column. Extract different attributes (see note below).**

   (i) Consider a few interesting attributes and summarize how many restaurants there are by values of these attributes; examine if star ratings vary by these attributes.

The attributes chosen are

(1) Alcohol

The Alcohol attribute has 4 categories – Beer & Wine, Ful Bar, None & Blanks (NA values). If we summarize the number of restaurants by these categories, we see that most restaurants have a full bar. The table below shows the count

Alcohol <chr>	n <int>
beer_and_wine	71
full_bar	205
none	171
NA	9

If we summarize the reviews to examine the star ratings, we see that most ratings are dispersed but the category "None" has 5-star ratings in 44% of its total instances. The table below shows star rating percentages for each star rating.

Alcohol <chr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
full_bar	18712	34	27	16	12	12
none	13598	43	26	12	8	11
beer_and_wine	7121	38	27	13	10	12
NA	533	44	24	14	7	11

As evident, we do not see a glaring difference in the distribution of star ratings. Therefore, we may conclude that alcohol availability at restaurants may not influence star ratings heavily.

(2) Good for Kids

This attribute has 3 values – True, False & Blanks (NA values).

GoodForKids <chr>	n <int>
False	95
True	352
NA	9

Most restaurants are rated as good for kids, there are a select few rated as False and blanks. When it comes to star ratings, The distributions of ratings are quite similar between True and False values. The table below shows the same

GoodForKids <chr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
True	30621	39	27	13	10	11
False	8804	32	29	17	11	11
NA	539	49	23	10	7	12

Since, the distributions are similar, we cannot derive much from this attribute.

(3) WiFi

The attribute "WiFi" has 4 factors – Free, Paid, No & Blanks (NA values). A select few restaurants provide paid Wifi. Most restaurants either provide free wifi or do not have any means of providing a wireless connection.

WiFi <chr>	n <int>
free	198
no	217
paid	5
NA	36

The star ratings vary with these values, restaurants with paid WiFi has lower 5-star ratings while restaurants with no or free wifi have a similar distribution.

WiFi <chr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
no	19895	37	27	13	10	12
free	18189	39	26	14	10	11
NA	1592	29	33	16	10	12
paid	288	18	24	21	18	19

(4) Noise Level

The noise attribute has 5 ranks – Very Loud, Loud, Average, Quiet & Blanks (NA values). Most restaurants are rated as average in terms of Noise level.

NoiseLevel <chr>	n <int>
average	316
loud	31
quiet	79
very_loud	9
NA	21

The star ratings for loud and very loud restaurants follow a similar distribution with lower % of reviews rated as 5-Star and majority share of reviews is dispersed. On the other hand, Average and quiet restaurants are reviewed as 5-star restaurants by more than 40% reviewers.

NoiseLevel <chr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
average	30433	39	27	14	10	11
quiet	5170	41	25	13	10	11
loud	2673	20	27	18	16	20
NA	1089	41	27	15	8	9
very_loud	599	25	23	18	16	18

(5) Price Range

The restaurant price attribute has 4 ranks – 1,2,3 & 4. Most restaurants have a low or mid score. Only a handful of restaurants have a price range of 3 or higher.

RestaurantsPriceRange2 <chr>	n <int>
1	173
2	259
3	19
4	5

The 5-star ratings for the cheapest and most expensive restaurants are higher compared to price ranges 2 & 3. The distribution of ratings however is very similar across all price ranges.

RestaurantsPriceRange2 <chr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
2	25183	37	27	14	11	11
1	12477	40	27	12	9	12
3	1931	35	27	17	10	10
4	373	46	18	19	11	6

(6) Touristy (From Ambience)

This ambience attribute is classified as either True or False. A few restaurants are rated as having a touristy ambience. Also, fewer restaurants with a touristy ambience have a 5-Star rating.

Touristy <fctr>	n <int>
FALSE	449
TRUE	7

Touristy <fctr>	num_ratings <int>	num_5_rat <dbl>	num_4_rat <dbl>	num_3_rat <dbl>	num_2_rat <dbl>	num_1_rat <dbl>
FALSE	39241	38	27	14	10	11
TRUE	723	23	20	16	14	27

(ii) For one of your models (choose your 'best' model from above), does prediction accuracy vary by certain restaurant attributes? You do not need to investigate all attributes; choose a few which you think may be interesting and examine these.

As observed above, the Random Forest model gave us the best results with a training accuracy of around 96% & Test accuracy of 87%. If we examine the attributes and group the accuracy as per the attribute values, we get the following –

(1) Good For Kids

The training data when grouped by this attribute, gives us the following accuracy -

GoodForKids <fctr>	Train_Count <int>	Accuracy <dbl>
False	5138	0.9634099
True	18560	0.9695043
NA	396	0.9595960

As evident, training accuracy is almost constant across all the values. In testing dataset, the accuracy for True instances is slightly higher.

GoodForKids <fctr>	Test_Count <int>	Accuracy <dbl>
False	2183	0.8685295
True	7957	0.8799799
NA	186	0.8655914

(2) Noise Level

If we examine the Training set grouped as per noise data, we see that accuracy for loud restaurants is comparatively lower than other categories.

NoiseLevel <fctr>	Train_Count <int>	Accuracy <dbl>
average	18325	0.9699318
loud	1549	0.9464170
quiet	3140	0.9671975
very_loud	346	0.9653179
NA	734	0.9713896

Additionally in the test data, we see a drastic reduction in accuracy for loud and very loud restaurants.

NoiseLevel <fctr>	Test_Count <int>	Accuracy <dbl>
average	7904	0.8841093
loud	642	0.8084112
quiet	1352	0.8816568
very_loud	143	0.7902098
NA	285	0.8666667

(3) WiFi

When grouped by WiFi, we observe that the training dataset shows similar accuracy but there are variations in the test set.

WiFi <fctr>	Train_Count <int>	Accuracy <dbl>
free	10924	0.9694251
no	12023	0.9672295
paid	151	0.9668874
NA	996	0.9628514

WiFi <fctr>	Test_Count <int>	Accuracy <dbl>
free	4681	0.8876308
no	5143	0.8722535
paid	75	0.7466667
NA	427	0.8477752

(4) Touristy (From Ambience)

When grouped by a touristy ambience, we observe that the training dataset shows dissimilar accuracy but there are similarities in the test set accuracies

Touristy <fctr>	Train_Count <int>	Accuracy <dbl>
FALSE	23601	0.9677556
TRUE	429	0.9277389

Touristy <fctr>	Test_Count <int>	Accuracy <dbl>
FALSE	10117	0.8837600
TRUE	178	0.8820225

From the above tables, we can conclude that the accuracy changes when grouped by certain attributes and examining these attributes may tell us about how well the model performs.