Hotel reviews (truthful vs deceptive)

Predictions using Text

December 03, 2020

```
## Load the required libraries
library(tm) #For text mining functionality
library(SnowballC) #For collapsing words to common roots
library(wordcloud) #For creating word cloud visualization
library(RColorBrewer) #For creating colorful graphs using pre-defined palettes
library(caret) # used for various predictive models
library(rpart) #For building Decision Tree Model
library(caTools) #For splitting the data
library(dplyr) #used for data transformation
library(rpart.plot) # used to plot decision tree
library(ranger) # used to random forest
library(randomForest) # used to random forest
library(quanteda) #to get ngrams without using RWeka
#library(RWeka) #used to create bi-grams and tri-grams
library("PRROC") # top plot ROC curve
library("ROCR") # top plot lift curve
```

1. Data Preparation

1.1 Loading data and basic checks

The dataset contains 400 Truthful positive , 400 Truthful negative , 400 Deceptive positive and 400 Deceptive negative reviews of the customers from 20 most popular hotels in Chicago.

Source of dataset is this paper - M. Ott, Y. Choi, C. Cardie, and J.T. Hancock. 2011. Finding Deceptive Opinion Spam by Any Stretch of the Imagination. In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, available at https://myleott.com/op_spamACL2011.pdf (https://myleott.com/op_spamACL2011.pdf)

"To solicit **gold-standard** deceptive opinion spam using AMT, we create a pool of 400 Human- Intelligence Tasks (HITs) and allocate them evenly across our 20 chosen hotels. To ensure that opinions are written by unique authors, we allow only a single submission per Turker. We also restrict our task to Turkers who are located in the United States, and who maintain an approval rating of at least 90%. Turkers are allowed a maximum of 30 minutes to work on the HIT, and are paid one US dollar for an accepted submission.

Each HIT presents the Turker with the name and website of a hotel. The HIT instructions ask the Turker to assume that they work for the hotel's marketing department, and to pretend that their boss wants them to write a fake review (as if they were a customer) to be posted on a travel review website; additionally, the review needs to sound realistic and portray the hotel in a positive light."

Target Variable - 'deceptive' column is the dependent variable and our task is to Predict if a review is truthful or deceptive.

```
#read the input file
data <- read.csv("deceptive-opinion.csv",head=TRUE)

#number of rows
nrow(data)
```

```
## [1] 1600
```

```
# column names
colnames(data)
```

```
## [1] "deceptive" "hotel"     "polarity"  "source"    "text"
```

There are 5 columns in the dataset.

‘deceptive’ is target column.

‘hotel’ column contains the name of the hotel.

‘polarity’ column tells us if the review was positive or negative.

‘source’ column has the information about the source of the review.

‘text’ column contains the actual reviews

Let’s look at the summary of the length of reviews

```
summary(nchar(as.character(data$text)))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##   151.0   487.0   700.0   806.4   987.5  4159.0
```

The minimum character count in a review is 151 and the maximum is 4,159. The average character count of all the reviews is about 806. Since, the minimum character count is 151, we also know from this that there is no empty review.

1.2 Data trasformation for Text Analysis

‘tm’ package is widely used for text mining in R. This package uses simple but very effective methodology and used in NLP. It counts the frequency of each word in the text and uses these counts as independent variables.

Next, we need to create a Corpus which is just a collection of text documents.

1.2.1 Corpus creation

```
# Create corpus
corpus = Corpus(VectorSource(data$text))
corpus
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 1600
```

```
#inspect a particular document
writeLines(as.character(corpus[[30]]))
```

```
## I booked a room at the Hyatt through Priceline and was able to get a rate of $45/night! Can't
go wrong with that! The rooms are nice, spacious, clean and contain all the amenties a traveler
would need. I have to say that I have fallen in love with the Hyatt chain due to their customer
service. I have stayed at a few of their hotels in different cities and the staff has always bee
n extremely friendly, which you don't often see at other hotels. I used the self-check in machin
e, and the customer service person checked in on me to make sure everything was going smoothly.
Then, when she saw that my room was not on a high level, she offered to have it changed for me s
o that I could have a better view. I would highly recommend this hotel!
```

```
#The tm package offers a number of transformations that ease the tedium of cleaning data.
#To see the available transformations  type getTransformations() at the R prompt:

# Convert to lower-case
corpus = tm_map(corpus, tolower)

# Remove punctuation
corpus = tm_map(corpus, removePunctuation)

# Look at stop words to understand which words we are removing for the analysis
stopwords("english")[1:10]
```

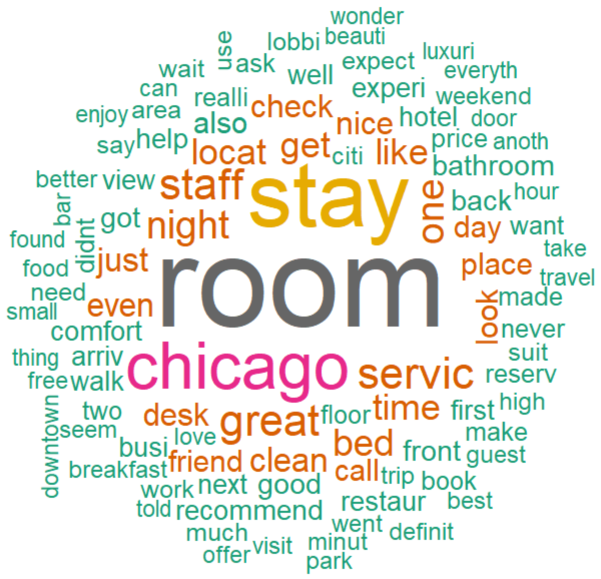
```
##  [1] "i"         "me"        "my"        "myself"    "we"        "our"
##  [7] "ours"      "ourselves" "you"       "your"
```

```
# Remove stopwords and the word 'hotel'
corpus = tm_map(corpus, removeWords, c("hotel", stopwords("english")))

# Getting roots of the words in the document
corpus = tm_map(corpus, stemDocument)

#see https://eight2late.wordpress.com/2015/05/27/a-gentle-introduction-to-text-mining-using-r/ f
or more on cleaning text data to use in prediction
```

1.2.2 Corpus Visualization

Here, we will use wordcloud package to display the words based on their frequency in form of a cloud. Note that this will only include unigrams as it’s based on the original corpus.

```
wordcloud(corpus, max.words=100,
                  random.order=FALSE,
                  rot.per=0.10,
                  colors=brewer.pal(8,"Dark2"))
```



From the wordcloud we can see that the words ‘room’, ‘stay’, ‘chicago’, ‘service’ have a very high frequency.

1.2.3 Term Document Matrix Creation and Bigrams

Here we are using the quanteda package to extract bigrams.

```
#In the TDM, the documents are represented by rows and the terms (or words) by columns.
#If a word occurs in a particular document, then the matrix entry for corresponding to that row
 and column is 1, else it is 0
#Multiple occurrences within a document are recorded – that is,
#If a word occurs twice in a document, it is recorded as “2” in the relevant matrix entry
#TF-IDF is a more sophisticated appraoch that normalizes the frequency of a word in a given docu
ment by its frequency across all documents

# Let's convert TM unigram corpus into a TDM with TF-IDF weighting
frequency_df_unigrams <- TermDocumentMatrix(corpus,control=list(weighting = weightTfIdf,normaliz
e=TRUE))
#we get a big matrix with 7372 terms as rows and 1600 review documents as columns
#lets see a small slice of this
inspect(frequency_df_unigrams[11:12,1000:1005])
```

```
## <<TermDocumentMatrix (terms: 2, documents: 6)>>
## Non-/sparse entries: 1/11
## Sparsity           : 92%
## Maximal term length: 8
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##           Docs
## Terms      1000 1001       1002 1003 1004 1005
##   complet     0    0 0.06329421    0    0    0
##   concierg    0    0 0.00000000    0    0    0
```

```
# quanteda corpus now, to get bigrams
corpus2 = corpus(corpus)
bigrams <- tokens(corpus2) %>%
    tokens_ngrams(n = 2) %>%
    dfm()

frequency_df_bigrams <- as.TermDocumentMatrix(convert(bigrams, to = "tm"))
frequency_df_bigrams <- weightTfIdf(frequency_df_bigrams,normalize=TRUE)

# Create a review - word count matrix - use TF-IDF instead of frequency
# we are calculating TF-IDF for each word in a review

# Inspect first 10 words of 5 Reviews
inspect(frequency_df_unigrams[1:10,1:5])
```

```
## <<TermDocumentMatrix (terms: 10, documents: 5)>>
## Non-/sparse entries: 13/37
## Sparsity           : 74%
## Maximal term length: 10
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##             Docs
## Terms                 1         2          3          4 5
##   173        0.19710845 0.0000000 0.00000000 0.00000000 0
##   44in       0.19710845 0.0000000 0.00000000 0.00000000 0
##   7th        0.16007141 0.0000000 0.00000000 0.00000000 0
##   aaa        0.14512039 0.0000000 0.00000000 0.00000000 0
##   adult      0.12660188 0.0000000 0.00000000 0.00000000 0
##   bathroomno 0.19710845 0.0000000 0.00000000 0.00000000 0
##   beat       0.10808336 0.0000000 0.00000000 0.00000000 0
##   bose       0.15410978 0.0000000 0.00000000 0.00000000 0
##   breakfast  0.05515778 0.0960813 0.00000000 0.00000000 0
##   chicago    0.01350872 0.0000000 0.02124673 0.03419396 0
```

```
# Inspect first 10 bigrams of 5 Reviews
inspect(frequency_df_bigrams[1:10,1:5])
```

```
## <<TermDocumentMatrix (terms: 10, documents: 5)>>
## Non-/sparse entries: 11/39
## Sparsity           : 78%
## Maximal term length: 15
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
## Sample             :
##                  Docs
## Terms                  text1 text2      text3 text4 text5
##   173_steal       0.19352466     0 0.00000000     0     0
##   aaa_rate        0.19352466     0 0.00000000     0     0
##   famili_thursday 0.19352466     0 0.00000000     0     0
##   getaway_famili  0.19352466     0 0.00000000     0     0
##   night_getaway   0.17534284     0 0.00000000     0     0
##   one_night       0.08402706     0 0.04443739     0     0
##   rate_173        0.19352466     0 0.00000000     0     0
##   stay_one        0.09742644     0 0.00000000     0     0
##   thursday_tripl  0.19352466     0 0.00000000     0     0
##   tripl_aaa       0.19352466     0 0.00000000     0     0
```

```
# Remove sparse terms. We are allowing a maximum sparsity of 0.95
#this means we wil reove a column that has 95% or more of its rows at 0
# Note: the only bigrams we now retain with a cutoff of 0.9 are room-service and front-desk.
# This cutoff needs to be higher than when working purely with unigrams, because bigrams are inh
erently sparser
frequency_df_unigrams = removeSparseTerms(frequency_df_unigrams, 0.95)
frequency_df_bigrams = removeSparseTerms(frequency_df_bigrams, 0.95)

# Convert to a data frame
frequency_df_unigrams = as.data.frame(as.matrix(frequency_df_unigrams))
frequency_df_bigrams = as.data.frame(as.matrix(frequency_df_bigrams))

# give the two tdm's the same doc names so we can append them together.
colnames(frequency_df_bigrams) <- colnames(frequency_df_unigrams)

# combine the two matrices.
frequency_df <- rbind(frequency_df_unigrams,frequency_df_bigrams)

# transpose data frame to get words as columns and reviews as rows
frequency_mat <- t(as.matrix(frequency_df))
mode(frequency_mat)="numeric"
frequency_df <- data.frame(frequency_mat)

# Make all variable names R-friendly
colnames(frequency_df) = make.names(colnames(frequency_df))

# Add dependent variable
frequency_df$outcome = data$deceptive
```

Lets see how a sample of how the overall text-to-numbers data engineering effort worked out. Here, for illustrative purposes, we show the first 5 of 263 ‘X’ columns and the outcome ‘Y’ column .

```
frequency_df[1:20, c(1:5, 264)]
```

	breakfast <dbl>	chicago <dbl>	complet <dbl>	concierg <dbl>	even <dbl>	outcome <chr>
1	0.05515778	0.013508723	0.07970382	0.06789956	0.03573353	truthful
2	0.09608130	0.000000000	0.00000000	0.00000000	0.00000000	truthful
3	0.00000000	0.021246730	0.00000000	0.00000000	0.01873408	truthful
4	0.00000000	0.034193956	0.00000000	0.00000000	0.03015017	truthful
5	0.00000000	0.000000000	0.00000000	0.00000000	0.00000000	truthful
6	0.00000000	0.008105234	0.04782229	0.04073974	0.00000000	truthful
7	0.05515778	0.000000000	0.00000000	0.00000000	0.00000000	truthful
8	0.00000000	0.010572044	0.00000000	0.00000000	0.00000000	truthful
9	0.00000000	0.010421015	0.00000000	0.00000000	0.00000000	truthful

	breakfast	chicago	complet	concierg	even	outcome
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<chr>
10	0.00000000	0.031716133	0.00000000	0.00000000	0.00000000	truthful

1-10 of 20 rows Previous **1** 2 Next

2. Predictive Models

Now, let's use the corpus that we created to design different predictive models. We will be creating 2 models which are KNN classification and Decision Tree.

But before that, we need to split the data into train and test group so that we can measure and compare the performance of the models.

Splitting the data

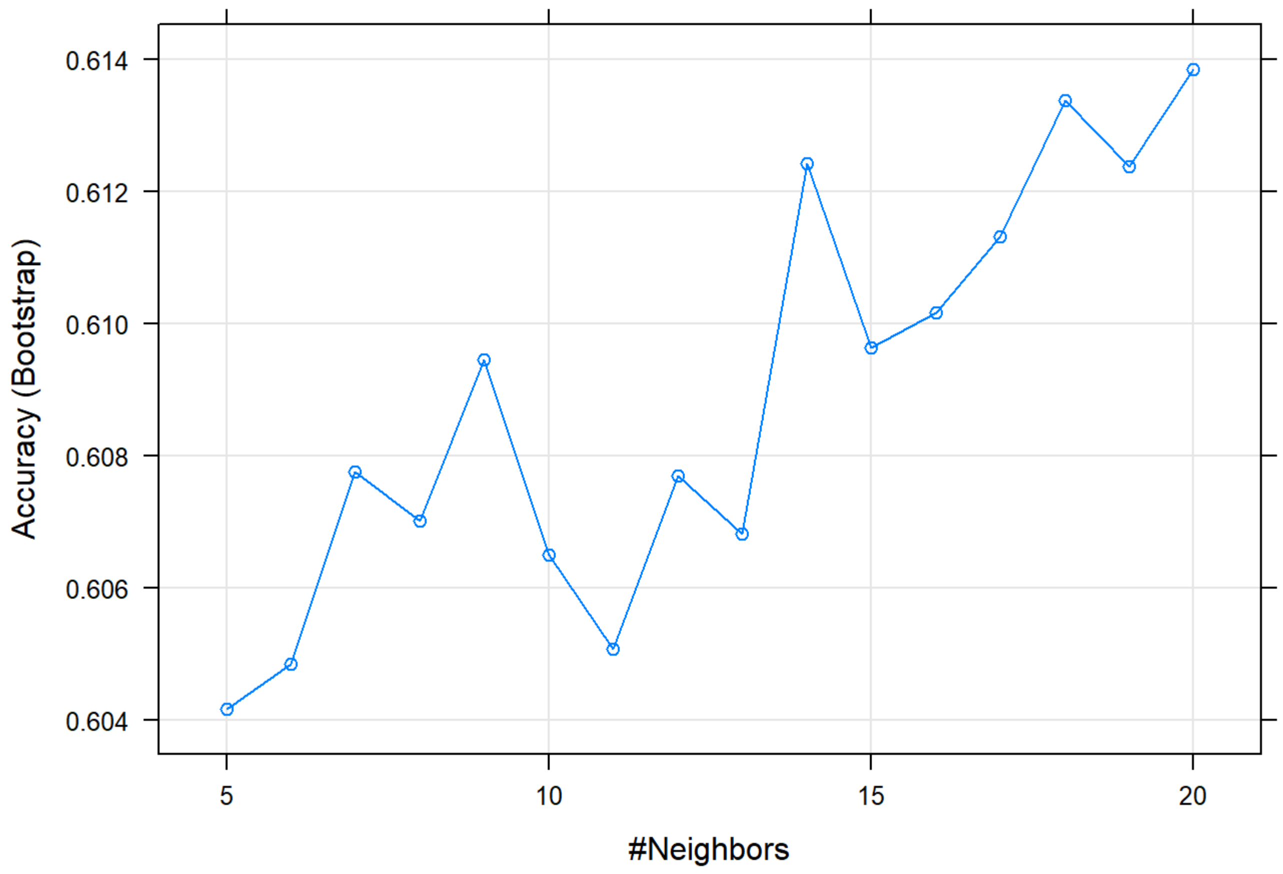
```
set.seed(1000)
# 75% of the data is used for training and rest for testing
smp_size <- floor(0.75 * nrow(frequency_df))

# randomly select row numbers for training data set
train_ind <- sample(seq_len(nrow(frequency_df)), size = smp_size)

# create Y (dependent) and X (independent) data frames
frequency_y = frequency_df %>% select("outcome")

frequency_x = frequency_df %>% select(-c("outcome"))

# creating test and training sets for x
frequency_x_train <- frequency_x[train_ind, ]
frequency_x_test <- frequency_x[-train_ind, ]

# creating test and training sets for y
frequency_y_train <- frequency_y[train_ind, ]
frequency_y_test <- frequency_y[-train_ind, ]
```

2.1 KNN Classification

```
# Hyperparameter tuning
# k = number of nearest neighbors
Param_Grid <-  expand.grid( k = c(5:20))

# fit the model to training data
knn_clf_fit <- train(frequency_x_train,
                     frequency_y_train,
                     method = "knn",
                     tuneGrid = Param_Grid)

# check the accuracy for different models
knn_clf_fit
```

```
## k-Nearest Neighbors
##
## 1200 samples
##  263 predictor
##    2 classes: 'deceptive', 'truthful'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    5  0.6041646  0.2045296
##    6  0.6048566  0.2055525
##    7  0.6077633  0.2111561
##    8  0.6070151  0.2099635
##    9  0.6094450  0.2145408
##   10  0.6065009  0.2087410
##   11  0.6050844  0.2061794
##   12  0.6077024  0.2113058
##   13  0.6068223  0.2095872
##   14  0.6124240  0.2207591
##   15  0.6096328  0.2148628
##   16  0.6101634  0.2160387
##   17  0.6113167  0.2183032
##   18  0.6133735  0.2227525
##   19  0.6123718  0.2205651
##   20  0.6138469  0.2234504
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 20.
```

```
# Plot accuracies for different k values
plot(knn_clf_fit)
```



```
# print the best model
print(knn_clf_fit$finalModel)
```

```
## 20-nearest neighbor model
## Training set outcome distribution:
##
## deceptive  truthful
##       602       598
```

```
# Predict on test data
knnPredict <- predict(knn_clf_fit, newdata = frequency_x_test)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(knnPredict, as.factor(frequency_y_test))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  deceptive truthful
##   deceptive        44        7
##   truthful        154      195
##
##                Accuracy : 0.5975
##                  95% CI : (0.5476, 0.6459)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : 0.0001252
##
##                   Kappa : 0.189
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.2222
##             Specificity : 0.9653
##          Pos Pred Value : 0.8627
##          Neg Pred Value : 0.5587
##              Prevalence : 0.4950
##          Detection Rate : 0.1100
##    Detection Prevalence : 0.1275
##       Balanced Accuracy : 0.5938
##
##        'Positive' Class : deceptive
##
```

```
x1 <- confusionMatrix(knnPredict, as.factor(frequency_y_test))[["overall"]]
y1 <- confusionMatrix(knnPredict, as.factor(frequency_y_test))[["byClass"]]

# Print Accuracy and F1 score

cat("Accuarcy is ", round(x1[["Accuracy"]],3), "and F1 is ", round (y1[["F1"]],3)  )
```

```
## Accuarcy is  0.598 and F1 is  0.353
```

2.2 Decision Tree

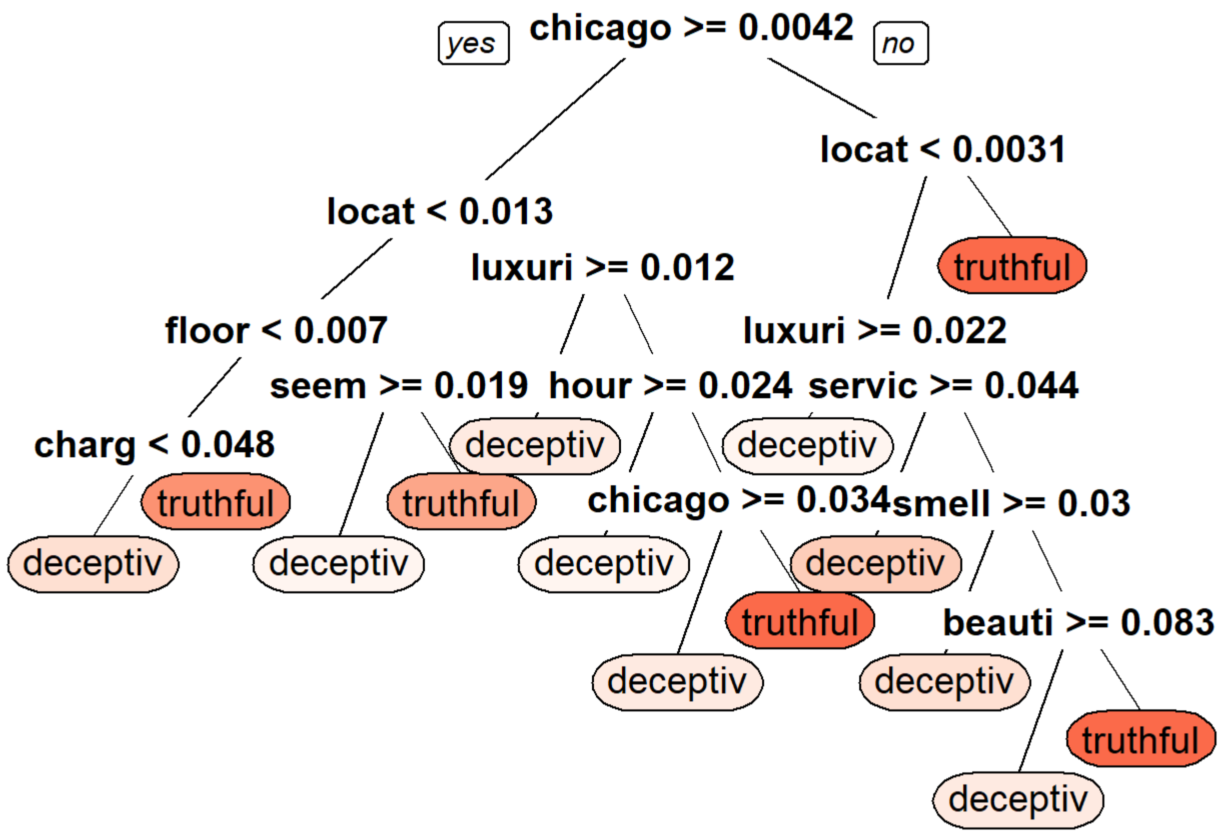
```
# Hyperparamter tuning
# maxdepth =  the maximum depth of the tree that will be created or
# the length of the longest path from the tree root to a leaf.

Param_Grid <-  expand.grid(maxdepth = 2:15)

dtree_fit <- train(frequency_x_train,
                   frequency_y_train,
                   method = "rpart2",
                   parms = list(split = "gini"),
                  tuneGrid = Param_Grid)

# check the accuracy for different models
dtree_fit
```

```
## CART
##
## 1200 samples
##  263 predictor
##    2 classes: 'deceptive', 'truthful'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1200, 1200, 1200, 1200, 1200, 1200, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    2        0.6700790  0.3411655
##    3        0.6719945  0.3467709
##    4        0.6778094  0.3576362
##    5        0.6817179  0.3647468
##    6        0.6830503  0.3675781
##    7        0.6811039  0.3635410
##    8        0.6847819  0.3700162
##    9        0.6850825  0.3703543
##   10        0.6851507  0.3706301
##   11        0.6845128  0.3691224
##   12        0.6840787  0.3685328
##   13        0.6834460  0.3673339
##   14        0.6833197  0.3669291
##   15        0.6801489  0.3606409
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 10.
```

```
# print the final model
dtree_fit$finalModel
```

```
## n= 1200
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##   1) root 1200 598 deceptive (0.50166667 0.49833333)
##     2) chicago>=0.004244698 713 253 deceptive (0.64516129 0.35483871)
##       4) locat< 0.01292922 498 139 deceptive (0.72088353 0.27911647)
##         8) floor< 0.006983922 438 106 deceptive (0.75799087 0.24200913)
##          16) charg< 0.0481671 417  92 deceptive (0.77937650 0.22062350) *
##          17) charg>=0.0481671 21   7 truthful (0.33333333 0.66666667) *
##         9) floor>=0.006983922 60  27 truthful (0.45000000 0.55000000)
##          18) seem>=0.01901707 7   0 deceptive (1.00000000 0.00000000) *
##          19) seem< 0.01901707 53  20 truthful (0.37735849 0.62264151) *
##       5) locat>=0.01292922 215 101 truthful (0.46976744 0.53023256)
##        10) luxuri>=0.01181214 31   4 deceptive (0.87096774 0.12903226) *
##        11) luxuri< 0.01181214 184  74 truthful (0.40217391 0.59782609)
##          22) hour>=0.02391759 20   2 deceptive (0.90000000 0.10000000) *
##          23) hour< 0.02391759 164  56 truthful (0.34146341 0.65853659)
##            46) chicago>=0.03354333 23   4 deceptive (0.82608696 0.17391304) *
##            47) chicago< 0.03354333 141  37 truthful (0.26241135 0.73758865) *
##     3) chicago< 0.004244698 487 142 truthful (0.29158111 0.70841889)
##       6) locat< 0.003082495 330 123 truthful (0.37272727 0.62727273)
##        12) luxuri>=0.02190286 18   1 deceptive (0.94444444 0.05555556) *
##        13) luxuri< 0.02190286 312 106 truthful (0.33974359 0.66025641)
##          26) servic>=0.04357843 32   9 deceptive (0.71875000 0.28125000) *
##          27) servic< 0.04357843 280  83 truthful (0.29642857 0.70357143)
##            54) smell>=0.03040444 16   3 deceptive (0.81250000 0.18750000) *
##            55) smell< 0.03040444 264  70 truthful (0.26515152 0.73484848)
##             110) beauti>=0.08341136 12   2 deceptive (0.83333333 0.16666667) *
##             111) beauti< 0.08341136 252  60 truthful (0.23809524 0.76190476) *
##       7) locat>=0.003082495 157  19 truthful (0.12101911 0.87898089) *
```

```
# Plot decision tree
prp(dtree_fit$finalModel, box.palette = "Reds", tweak = 1.2)
```



```
# Predict on test data
dtree_predict <- predict(dtree_fit, newdata = frequency_x_test)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(dtree_predict,  as.factor(frequency_y_test))
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction  deceptive truthful
##   deceptive       138       55
##   truthful         60      147
##
##                Accuracy : 0.7125
##                  95% CI : (0.6654, 0.7564)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.4248
##
##  Mcnemar's Test P-Value : 0.7091
##
##             Sensitivity : 0.6970
##             Specificity : 0.7277
##          Pos Pred Value : 0.7150
##          Neg Pred Value : 0.7101
##              Prevalence : 0.4950
##          Detection Rate : 0.3450
##    Detection Prevalence : 0.4825
##       Balanced Accuracy : 0.7123
##
##        'Positive' Class : deceptive
##
```

```
# Add results into clf_results dataframe
x2 <- confusionMatrix(dtree_predict,  as.factor(frequency_y_test))[["overall"]]
y2 <- confusionMatrix(dtree_predict,  as.factor(frequency_y_test))[["byClass"]]

# Print Accuracy and F1 score

cat("Accuarcy is ", round(x2[["Accuracy"]],3), "and F1 is ", round (y2[["F1"]],3)  )
```

```
## Accuarcy is  0.713 and F1 is  0.706
```

2.3 Random Forest

```
set.seed(100)

#By default, the train function without any arguments re-runs the model over 25 bootstrap sample
s and across 3 options of the tuning parameter (the tuning #parameter for ranger is mtry; the nu
mber of randomly selected predictors at each cut in the tree).

fitControl <- trainControl(
  method = "repeatedcv",
  number = 10,
  repeats = 1,
  classProbs = TRUE,
  )



rf_ranger_fit <- train(frequency_x_train,
                   frequency_y_train,
                   method = "ranger",
                    trControl=fitControl,
                   importance = "permutation", #this is to get feature importance later
                  )

#Permutation Importance is assessed for each feature by removing the association between that fe
ature and the target. #This is achieved by randomly #permuting the values of the feature and mea
suring the resulting increase in error. The influence of the correlated features #is also remove
d.

# print the  model
rf_ranger_fit
```

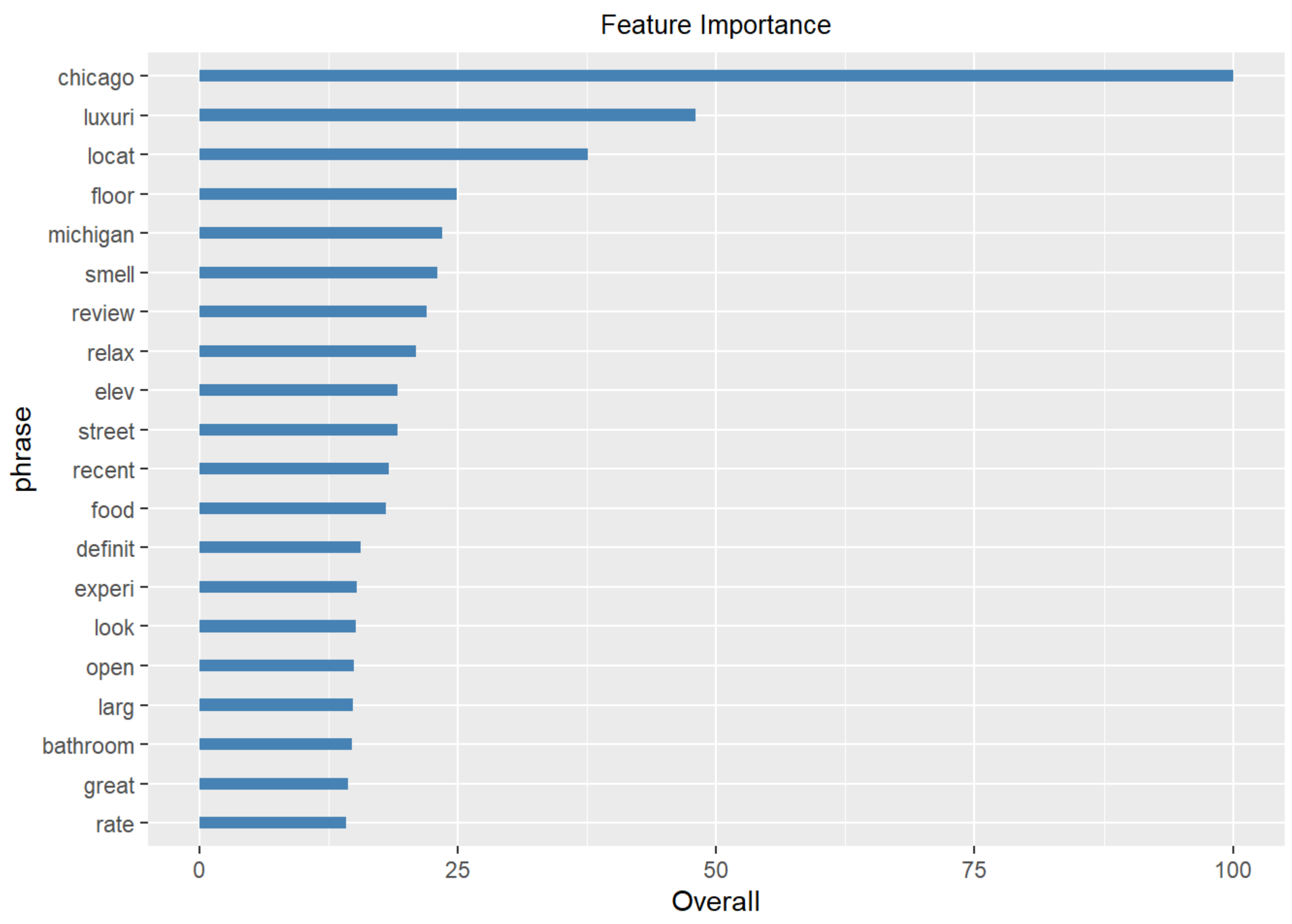
```
## Random Forest
##
## 1200 samples
##  263 predictor
##    2 classes: 'deceptive', 'truthful'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 1 times)
## Summary of sample sizes: 1080, 1080, 1079, 1080, 1081, 1080, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   Accuracy   Kappa
##     2   gini        0.8233145  0.6465555
##     2   extratrees  0.8166196  0.6332192
##   132   gini        0.7809250  0.5619035
##   132   extratrees  0.8075642  0.6151123
##   263   gini        0.7742655  0.5485820
##   263   extratrees  0.7959111  0.5918338
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 2, splitrule = gini
##  and min.node.size = 1.
```

Feature importance

```
# Order the variables based on their importance
temp1 <- varImp(rf_ranger_fit)$importance
top20 <- temp1 %>% arrange((Overall)) %>% top_n(20)
```

```
## Selecting by Overall
```

```
#plot
ggplot(top20 %>%
       mutate(phrase=factor(rownames(top20), levels=rownames(top20)) ),
       aes(x = phrase, y = Overall)) +
       geom_bar(stat = "identity" , width=0.3, fill="steelblue") + coord_flip() +
       ggtitle("Feature Importance") +
       theme(plot.title = element_text(color="black", size=10, hjust = 0.5))
```



Prediction on test data

```
# Predict outcome on test data
rf_pred <- predict(rf_ranger_fit,frequency_x_test )

# Print Confusion matrix, Accuracy, Sensitivity etc
confusionMatrix(rf_pred,  as.factor(frequency_y_test))
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  deceptive truthful
##   deceptive       172       40
##   truthful         26      162
##
##                Accuracy : 0.835
##                  95% CI : (0.7949, 0.87)
##     No Information Rate : 0.505
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.6702
##
##  Mcnemar's Test P-Value : 0.1096
##
##             Sensitivity : 0.8687
##             Specificity : 0.8020
##          Pos Pred Value : 0.8113
##          Neg Pred Value : 0.8617
##              Prevalence : 0.4950
##          Detection Rate : 0.4300
##    Detection Prevalence : 0.5300
##       Balanced Accuracy : 0.8353
##
##        'Positive' Class : deceptive
##
```

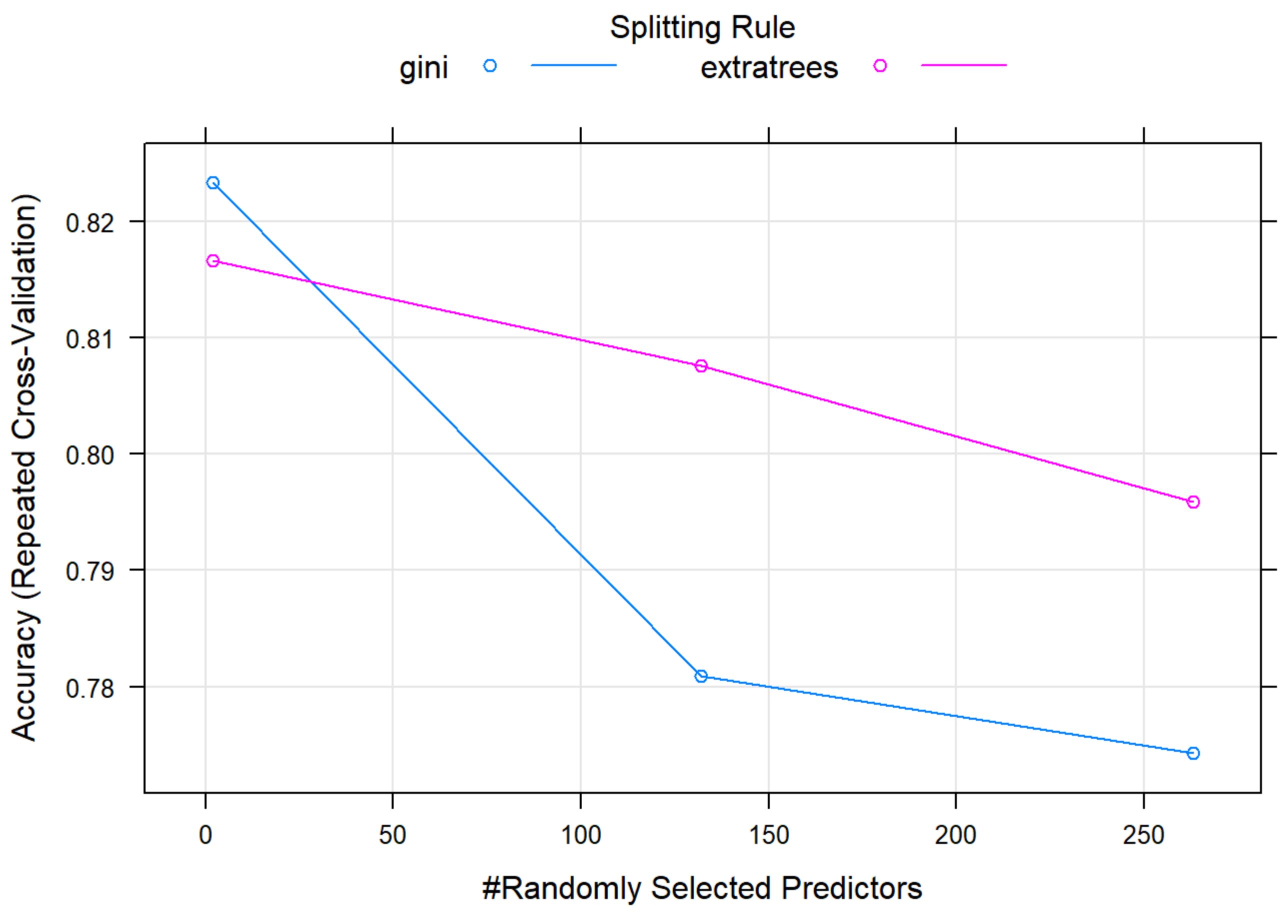
```
# Add results into clf_results dataframe
x3 <- confusionMatrix(rf_pred,  as.factor(frequency_y_test))[["overall"]]
y3 <- confusionMatrix(rf_pred,  as.factor(frequency_y_test))[["byClass"]]

# Print Accuracy and F1 score

cat("Accuarcy is ", round(x3[["Accuracy"]],3), "and F1 is ", round (y3[["F1"]],3)  )
```

```
## Accuarcy is  0.835 and F1 is  0.839
```

```
#model take a long time hence I am recording the result
# Confusion Matrix and Statistics
#
#            Reference
# Prediction  deceptive truthful
#   deceptive       176       43
#   truthful         22      159
#
#                Accuracy : 0.8375
#                  95% CI : (0.7976, 0.8723)
#     No Information Rate : 0.505
#     P-Value [Acc > NIR] : < 2e-16
#
#                   Kappa : 0.6753
#
#  Mcnemar's Test P-Value : 0.01311
#
#             Sensitivity : 0.8889
#             Specificity : 0.7871
#          Pos Pred Value : 0.8037
#          Neg Pred Value : 0.8785
#              Prevalence : 0.4950
#          Detection Rate : 0.4400
#    Detection Prevalence : 0.5475
#       Balanced Accuracy : 0.8380
#
#        'Positive' Class : deceptive
```

Of all the 3 Models, Random Forest gave a better accuracy.

```
plot(rf_ranger_fit)
```



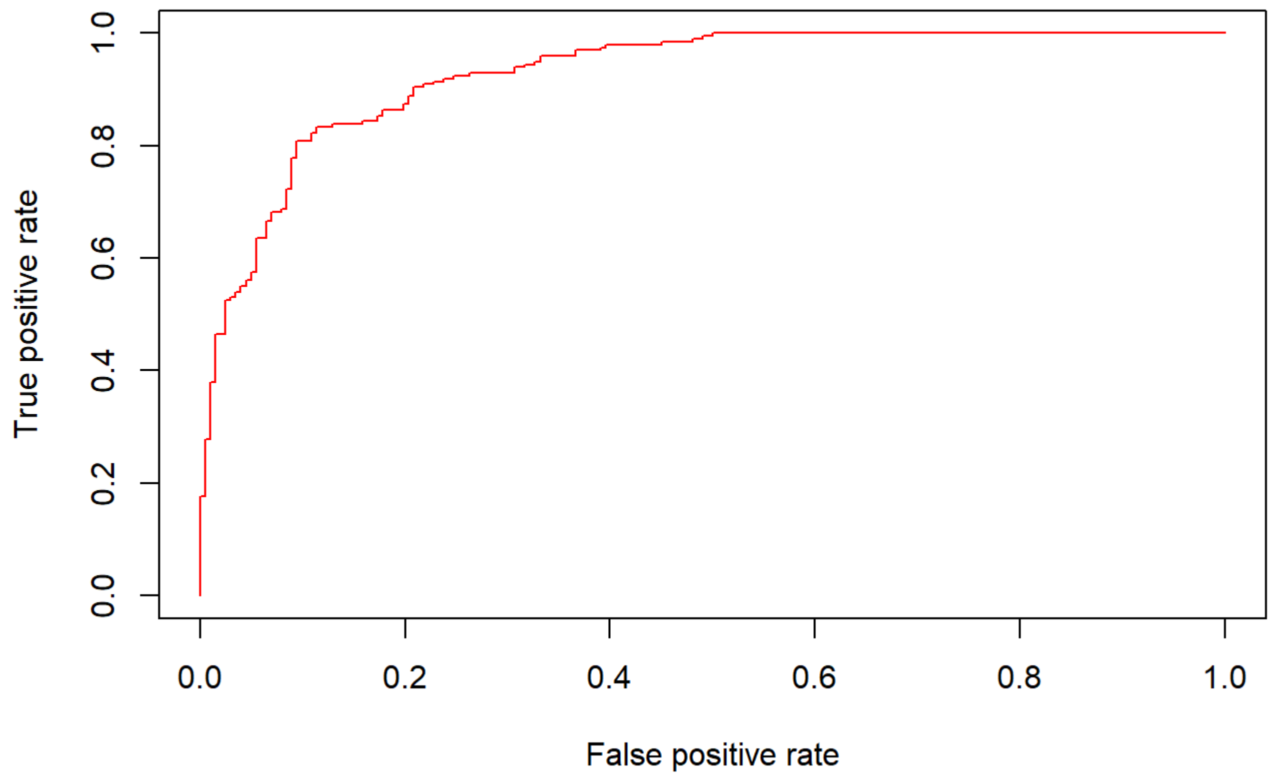
```
varImp(rf_ranger_fit)
```

```
## ranger variable importance
##
##   only 20 most important variables shown (out of 263)
##
##          Overall
## chicago   100.00
## luxuri     48.00
## locat      37.60
## floor      24.92
## michigan   23.49
## smell      23.00
## review     21.95
## relax      20.93
## elev       19.21
## street     19.14
## recent     18.34
## food       18.09
## definit    15.63
## experi     15.27
## look       15.18
## open       14.96
## larg       14.83
## bathroom   14.75
## great      14.38
## rate       14.21
```

```
confusionMatrix(rf_ranger_fit)
```

```
## Cross-Validated (10 fold, repeated 1 times) Confusion Matrix
##
## (entries are percentual average cell counts across resamples)
##
##            Reference
## Prediction  deceptive truthful
##   deceptive      42.3      9.8
##   truthful        7.8     40.0
##
##  Accuracy (average) : 0.8233
```

```
rf.pred <- predict(rf_ranger_fit, newdata = frequency_x_test, type = 'prob')
pred.rf <- prediction(rf.pred[,1]   , frequency_y_test,  label.ordering = c('truthful','deceptiv
e'))
perf.rf <- performance(pred.rf   , "tpr", "fpr")

plot(perf.rf    , col ='Red'  )
```



```
performance(pred.rf    , "auc")@y.values
```

```
## [[1]]
## [1] 0.9270177
```

3. Conclusions

In this tutorial, we learned how to use text data to create predictive models. The aim was to showcase how text data can be transformed and used in a predictive model.