Title: Capstone Project - Podcast Recommendation

Course: CSDA1050, York University

Introduction:-

Podcast exists for near two decades. But it really takes over in recent two years. The podcast meta data may be useful for research in fields like machine learning, social science, or media in general. Listen Notes is the podcast search engine that actually works. It has the most comprehensive podcast database that you can find on the Internet.

This dataset includes the meta data of 13000 itunes podcasts april 2018.

If you want to play with podcast meta data in CSV files, you can check out Listen Datasets.

Scope of the article:-

Based on the variables, there are several supervised and unsupervised techniques that can could be performed on the above dataset to throw several insights on customer preferences. However, we would limit the scope of this blog only to using text mining extensively to analyze the customer reviews.

We will be using the following techniques to understand various aspects of text mining:

- Exploratory analysis of text data (Review Text) individually and based on how it impacts the customer decision to recommend the product (Recommended IND)
- Classification models that are built based on the review text as the independent variable to predict whether a customer recommends a product

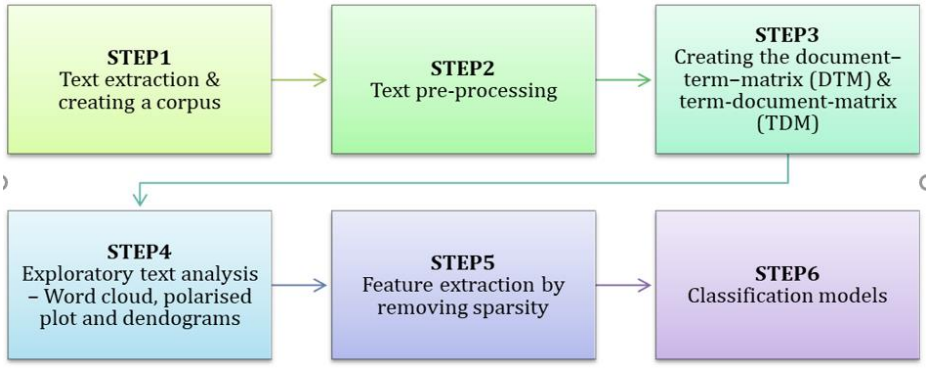
Since the objective of the blog is more to understand text mining, the focus will be to understand differences between customers who recommend a product and those who don't rather than predicting the customer action based on the review. In other words, we will be focusing more on variable importance and coefficient scores of the models than model performance measures.

In terms of text mining approaches, there are 2 broad categories

- Semantic parsing where the word sequence, word usage as noun or verb, hierarchial word structure etc matters
- Bag of words where all the words are analysed as a single token and order does not matter.

Our exercise will only be limited to the "bag of words" approach and will not look into semantic parsing.

High level approach



STEP1 — Text extraction & creating a corpus

Initial setup

The packages required for text mining are loaded in the R environment:

```
# install.packages("ggthemes")
# install.packages(qdap)
# install.packages(dplyr)
# install.packages(tm)
# install.packages(wordcloud)
# install.packages(plotrix)
# install.packages(dendextend)
# install.packages(ggplot2)
# install.packages(ggthemes)
# install.packages(RWeka)
# install.packages(reshape2)
# install.packages(quanteda)
```

```
library(qdap)
library(dplyr)
library(tm)
library(wordcloud)
library(plotrix)
library(dendextend)
library(ggplot2)
library(ggthemes)
library(RWeka)
library(reshape2)
library(quanteda)
```

Once the required packages are installed, the working directory is set and the csv files are read into R:

```
```{r}
setwd("C:/Users/Owner/Desktop/1050 Advanced Analytics and Capstone Course")
review=read.csv("poddf.csv", stringsAsFactors = FALSE)
names(review)
```
```

Text extraction

The column **Review.Text** contains the customer reviews received for various products. This is the focus for our analysis. We will now try to understand how to represent text as a data frame.

1. First, the review.text is converted into a collection of text documents or "*Corpus*".
2. To convert the text into a corpus, we use the "tm" package in R.
3. In order to create a corpus using tm, we need to pass a "Source" object as a parameter to the VCorpus method.

4. The source object is similar to an abstract input location. The source we use here is a “Vectorsource” which inputs only character vectors.
5. The Review.text column is now converted to a corpus that we call “corpus_review”.

```
## Make a vector source and a corpus
corpus_review=Corpus(VectorSource(review$Review.Text))
```

STEP2 — Text Pre-processing

The ultimate objective of any text mining process using the “bag-of-words” approach is to convert the text to be analysed to a data frame which consists of the words used in the text and their frequencies. These are defined by the **document term matrix (DTM)** and the **term document matrix (TDM)**which we will look into, in the subsequent sections.

To ensure that the DTM and TDM are cleaned up and represent the core set of relevant words, a set of pre-processing activities need to be performed on the corpus. This is similar to the data clean-up done for structured data before data mining. The following are some of the common pre-processing steps:

1. Convert to lower case:-

```
```{r}
corpus_review=tm_map(corpus_review, tolower)
```
```

2. Remove punctuation:-

```
```{r}
corpus_review=tm_map(corpus_review, removePunctuation)
```
```

3. Remove stopwords: “stopwords” is a very important concept to understand while doing text mining. When we write, the text generally consists of a large number of prepositions, pronouns, conjunctions etc. These words need to be removed before we analyse the text. Otherwise, stopwords will appear in all the frequently used words list and will not give the correct picture of the core words used in the text.There is a list of common stopwords used in English which we can view with this command:

```
```{r}
corpus_review=tm_map(corpus_review,
removeWords,c("i","my","time","know","and","the","have","other","a",
"in","is","with","for","as","our","of","you","de","are","at","it","s
how","we","by","to","be","or","an","on"))
```
```

4. We might also want to remove custom stopwords based on the context of the text mining. These are words specific to the dataset that may not add value to the text.

Stemming a document

In linguistics, stemming is the process of reducing inflected (or derived) words to their word stem, base or root form-generally a written word form.

The SnowballC package is used for document stemming. For example “complicated”, “complication” and “complicate” will be reduced to “complicate” after stemming. This is again to ensure that the same word is not repeated as multiple versions in the DTM and TDM and we only have the root of the word represented in the DTM and TDM.

```
```{r}
corpus_review=tm_map(corpus_review, stemDocument)
```
```

Frequently used words

We now have a text corpus which is cleaned and only contains the core words required for text mining. The next step is exploratory analysis. The first step in exploratory data analysis is to identify the most frequently used words in the overall review text.

```
# Find the 20 most frequent terms: term_count
term_count <- freq_terms(corpus_review, 20)

# Plot 20 most frequent terms
plot(term_count)
```

STEP3 — Create the DTM & TDM from the corpus

The pre-processed and cleaned up corpus is converted into a matrix called the document term matrix.
A document-term matrix is a mathematical matrix that describes the frequency of terms that occur in a collection of documents. In a document-term matrix, rows correspond to documents in the collection and columns correspond to terms.

The term-document matrix is a transpose of the document-term matrix. It is generally used for language analysis. An easy way to start analyzing the information is to change the DTM/TDM into a simple matrix using as.matrix().

```
review_dtm <- DocumentTermMatrix(corpus_review)
review_tdm <- TermDocumentMatrix(corpus_review)
```

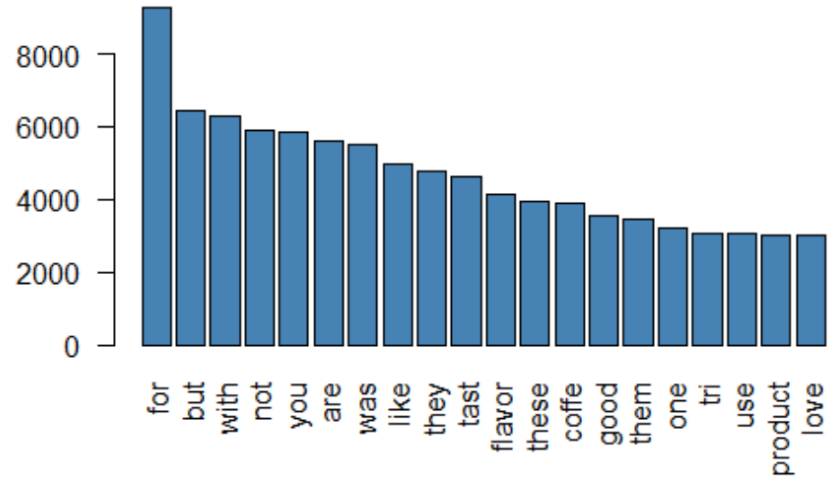
Using the TDM to identify frequent terms

The TDM can also used to identify frequent terms and in subsequent visualization related to the review text.

```
# Convert TDM to matrix
review_m <- as.matrix(review_tdm)
# Sum rows and frequency data frame
review_term_freq <- rowSums(review_m)
# Sort term_frequency in descending order
review_term_freq <- sort(review_term_freq, decreasing = T)
# View the top 10 most common words
review_term_freq[1:10]
```

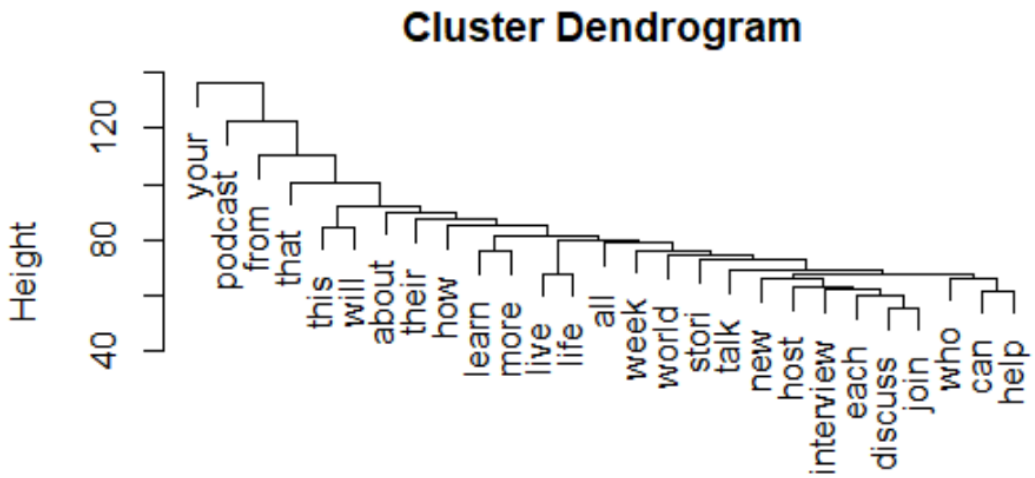
STEP4 — Exploratory text analysis

```
# Plot a barchart of the 20 most common words
barplot(review_term_freq[1:20], col = "steel blue", las = 2)
```



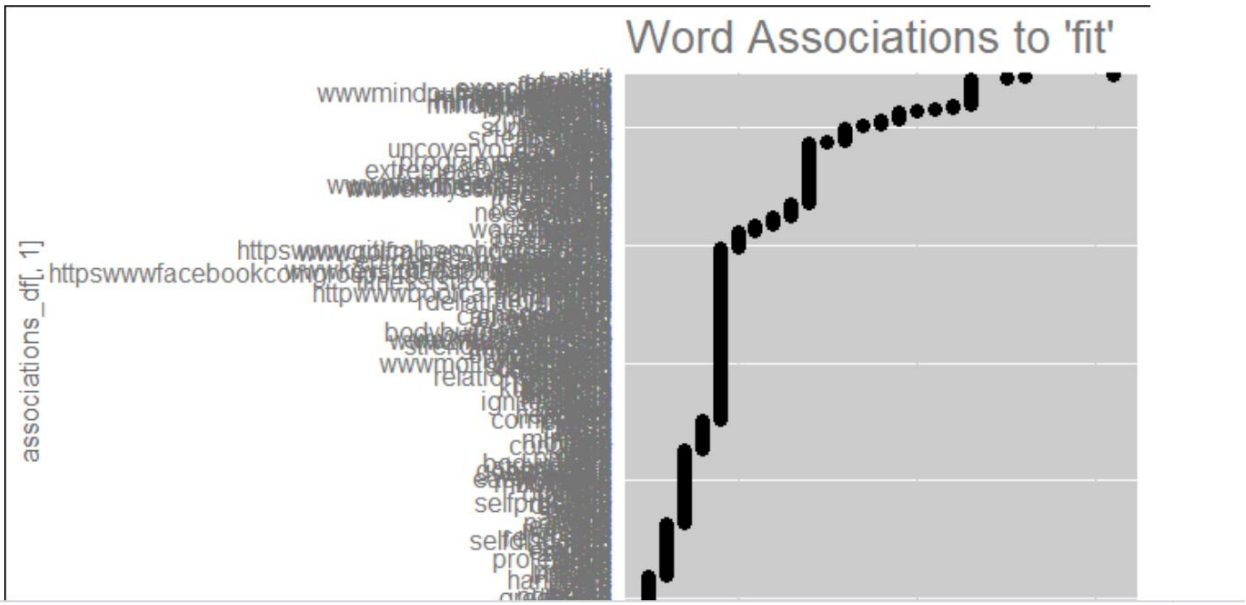
Word clouds:- Word cloud is a common way of visualizing a text corpus to understand the frequently used words. The word cloud varies the size of the words based on the frequency. The word cloud can receive a set of colors or a color palette as input to distinguish between the more and the lesser frequent words in the cloud.

Simple word clustering: - Word clustering is used to identify word groups used together, based on frequency distance. This is a dimension reduction technique. It helps in grouping words into related clusters. Word clusters are visualized with dendrograms.



Word associations:-

Word association is a way of calculating the correlation between 2 words in a DTM or TDM. It is yet another way of identifying words used together frequently. For our corpus, the word association plot indicates correlation between various words and the word “fit”.



STEP5 — Feature extraction by removing sparsity

##Concept of sparsity
Sparsity is related to the document frequency of a term. In DTM, since the terms form the columns, every document will have several columns each representing one term — a unigram, bi-gram, tri-gram, etc.

##Feature extraction
The exploratory text analysis has given several insights based on the customer reviews. We will now use the same review text as predictor variable to predict whether the product will be recommended by the customer. In terms of classification algorithms used, there is not much of a difference between data and text input. We will try 3 of the most popular classification algorithms — CART, Random forest

```{r}
## Load the required libraries
suppressMessages(library(irlba))
suppressMessages(library(e1071))
suppressMessages(library(caret))
suppressMessages(library(randomForest))
suppressMessages(library(rpart))
suppressMessages(library(rpart.plot))
suppressMessages(library(ggplot2))
suppressMessages(library(SnowballC))
suppressMessages(library(RColorBrewer))
suppressMessages(library(wordcloud))
suppressMessages(library(biclust))
suppressMessages(library(igraph))
suppressMessages(library(fpc))
```

Tokenisation

Tokenisation is the process of decomposing text into distinct pieces or tokens.

This is what we called as bag-of-words in our previous section. Once tokenization is done, after all the pre-processing, it is possible to construct a dataframe where each row represents a document and each column represents a distinct token and each cell gives the count of the token for a document. This is the DTM that we learnt in the previous section.
The pre-processing steps are carried out on the tokens, similar to what was done on the text corpus.

```
# Tokenize descriptions
reviewtokens=tokens(review$Review.Text,what="word",
remove_numbers=TRUE,remove_punct=TRUE, remove_symbols=TRUE,
remove_hyphens=TRUE)

# Lowercase the tokens
reviewtokens=tokens_tolower(reviewtokens)

# remove stop words and unnecessary words
rmwords <- c("dress", "etc", "also", "xxs", "xs", "s")
reviewtokens=tokens_select(reviewtokens, stopwords(),selection =
"remove")
reviewtokens=tokens_remove(reviewtokens,rmwords)

# Stemming tokens
reviewtokens=tokens_wordstem(reviewtokens,language = "english")
reviewtokens=tokens_ngrams(reviewtokens,n=1:2)
```

The tokens are now converted to a document frequency matrix and treated for sparsity.

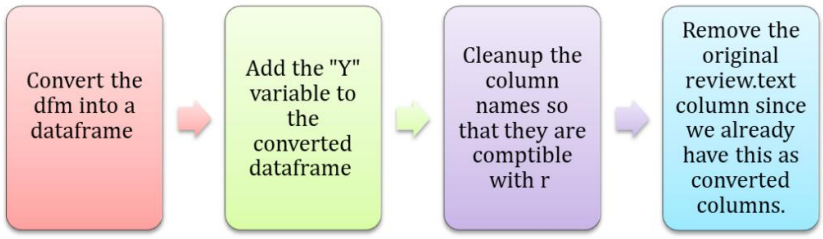
```
# Creating a bag of words
reviewtokensdfm=dfm(reviewtokens,tolower = FALSE)

# Remove sparsity
reviewSparse <- convert(reviewtokensdfm, "tm")
tm::removeSparseTerms(reviewSparse, 0.7)

# Create the dfm
dfm_trim(reviewtokensdfm, min_docfreq = 0.3)
x=dfm_trim(reviewtokensdfm, sparsity = 0.98)
```

STEP6 — Building the Classification Models

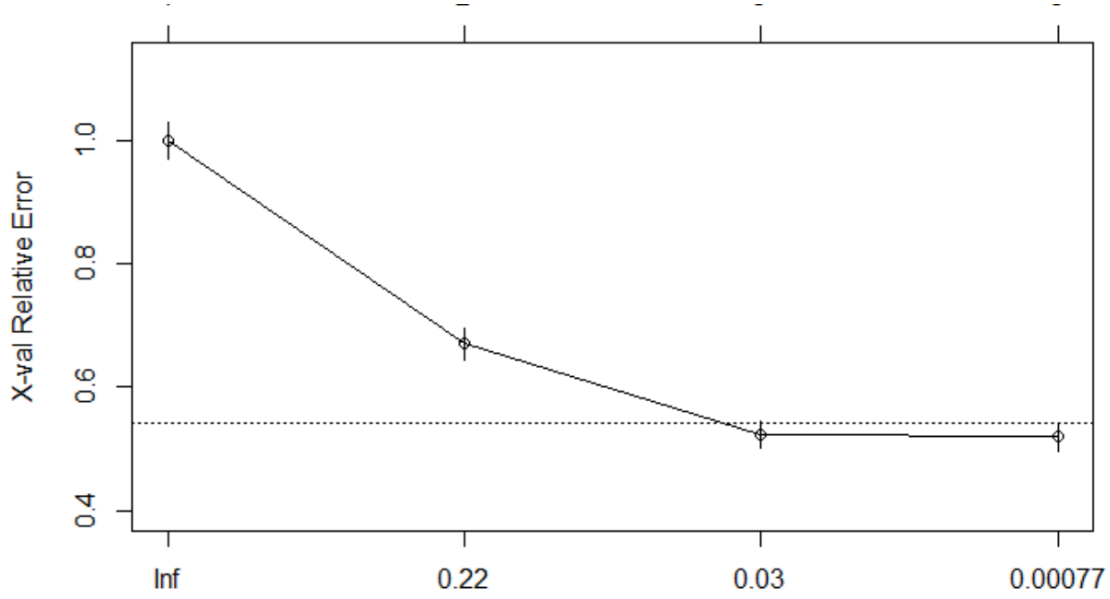
We now have the dfm which is pre-processed, treated and ready to be used for classification. To use this in a classification model, the following steps are carried out:



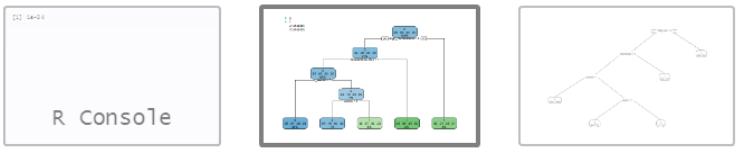
CART model

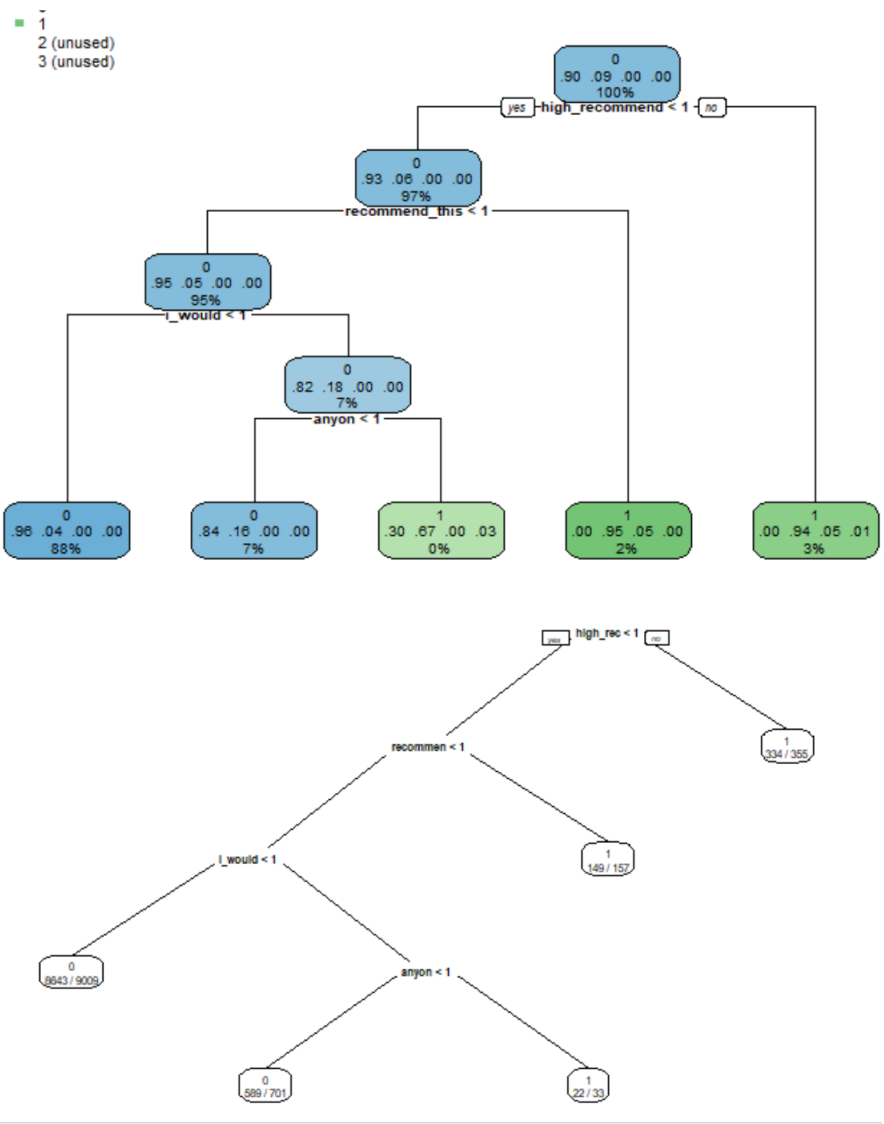
We will first use the CART algorithm for classification. First the complete tree is built and the optimum cp value is identified for which the error is minimum. Then this cp value is used to obtain the pruned tree. The pruned tree is plotted to understand the classification.

```
## Build the CART model
tree=rpart(formula = recommend ~ ., data = reviewtokensdf, method="class",control =
rpart.control(minsplit = 200,  minbucket = 30, cp = 0.0001))
printcp(tree)
plotcp(tree)
```




```
##Prune down the tree
bestcp=tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
bestcp
ptree=prune(tree,cp=bestcp)
rpart.plot(ptree,cex = 0.6)
prp(ptree, faclen = 0, cex = 0.5, extra = 2)
```



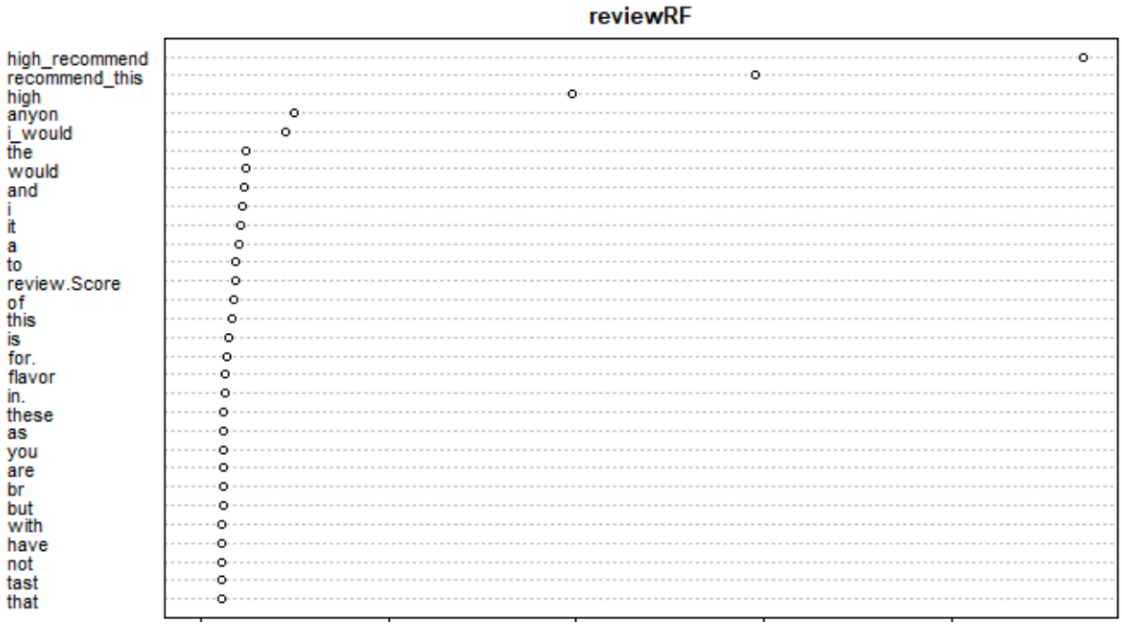


As can be seen from the tree plot, words like “High Rec”, “recommend”, “I would”, etc are used by happy customers — i.e., customers do recommend the product. The tree can be interpreted further to understand the word patterns used by customers who recommend the product vs those who don’t.

Random forest

The next classification algorithm we will use is the Random forest. We will examine the varimp plot of the random forest model to understand which words affect the classification the most.

```
library(randomForest)
reviewRF=randomForest(recommend~., data=reviewtokensdf)
varImpPlot(reviewRF, cex=.7)
```



In sync with the CART model, the varimp plot of the Random forest model also , words like “High Rec”, “recommend”, “I would”, etc are used by happy customers — i.e., customers do recommend the product. The tree can be interpreted further to understand the word patterns used by customers who recommend the product vs those who don’t.