PSTAT 134 Final Project

Rahul Narula, Kendall Brown, Lauren Gripenstraw, Jory Oberlies

June 14, 2018

```
### Install and load required packages

#install.packages("tm")
#install.packages("SnowballC")
#install.packages("wordcloud")
#install.packages("RColorBrewer")
#install.packages("topicmodels")
#install.packages("ldatuning")
#install.packages("SentimentAnalysis")

library(tm)
library(SnowballC)
library(wordcloud)
library(RColorBrewer)
library(topicmodels)
library(ldatuning)
library(SentimentAnalysis)
```

We begin by installing the necessary modules for our analysis.

```
### Read Data

reviews <- read.csv("C:/Users/kebro/Downloads/yelp.csv", stringsAsFactors = F)       #Read dataset
reviews <- as.matrix(reviews[5])                                #Convert data to matrix
reviews.vec <- VectorSource(reviews)                            #Create data vector
reviews.corpus <- Corpus(reviews.vec)                           #Create corpus
```

The next step is to read in our data and convert it into a corpus so that we can apply text mining methods to it.

```
### Prepare and Clean Data

#Preprocessing
reviews.corpus <- tm_map(reviews.corpus, tolower)               #Remove capital letters
reviews.corpus <- tm_map(reviews.corpus, removePunctuation)     #Remove punctuation
reviews.corpus <- tm_map(reviews.corpus, removeNumbers)         #Remove numbers
reviews.corpus <- tm_map(reviews.corpus, removeWords,           #Remove stopwords
                         stopwords("english"))
reviews.corpus <- tm_map(reviews.corpus, stemDocument)          #Stem words
reviews.corpus <- tm_map(reviews.corpus, stripWhitespace)       #Remove whitespace
```

Once the data has been read, it is necessary to clean it. Preprocessing makes the text more uniform and ultimately improves results. Here, we convert all text to lower case, remove punctuation, numbers, and stopwords, stem words, and remove extra whitespace. The benefits to each process are:

Lower case: Converting all text to lower case ensures that words with different capitalizations are not counted differently. For example, we do not want "Text" and "text" to be counted as different words as this can significantly alter results. Perhaps this process is not desired if one wishes to understand what people start their sentences with, but that is not the aim of our project.

Remove Punctuation: Again, this further renders our data uniform. It is also possible that failing to remove punctuation could count "text-mining," "text mining", and "text mining." differently.

Remove Numbers: Although extracting numbers in a review may uncover certain information, they are difficult to work with and would most likely have very little impact on our project.

Remove Stop Words: Stop words are words with very little predictive value. These often clutter the dataset without providing any insight. An added benefit is that working with the dataset after removing these words is more computationally efficient.

Stem Words: Words of various conjugations and forms have the same meaning for our purposes so we do not want them to be counted separately. For example, "visit," "visited," and "visitor" all convey the same general meaning and thus we would like them grouped together. This method may backfire for certain words, but it generally improves results overall.

Stripping White Space: After performing these pre-processing operations, we are left with many empty spaces which make the data difficult to read. Stripping unnecessary whitespace remedies this issue and makes our data more coherent.

```
### Word Frequencies


#Create Term Document Matrix
tdm <- TermDocumentMatrix(reviews.corpus,
       control=list(weighting=weightTfIdf))            #Weight by TF-IDF
```

```
## Warning in weighting(x): empty document(s): 6451 6709
```

```
removeSparseTerms(tdm, 0.99)                           #Remove infrequent terms
```

```
## <<TermDocumentMatrix (terms: 958, documents: 10000)>>
## Non-/sparse entries: 400431/9179569
## Sparsity           : 96%
## Maximal term length: 12
## Weighting          : term frequency - inverse document frequency (normalized) (tf-idf)
```

```
tdm <- as.matrix(tdm)

#Inspect data
v <- sort(rowSums(tdm),decreasing=TRUE)                #Sort in order
d <- data.frame(word = names(v),freq=v)                #Convert to dataframe
head(d, 20)                                            #Preview data
```

```
##          word     freq
## great   great 198.3620
## food     food 183.9265
## good     good 171.4096
## place   place 164.3998
## love     love 155.1456
## servic servic 145.8586
## best     best 121.8910
## time     time 120.8788
## like     like 120.8679
## get       get 119.4903
## friend friend 119.1571
## just     just 113.6094
## price   price 112.9885
## pizza   pizza 112.4825
```

```
## one       one 110.9634
## alway   alway 110.3223
## realli realli 107.4621
## nice     nice 106.8261
## order   order 106.4126
## tri       tri 100.9112
```

Now that our dataset is ready for analysis, we create a term-document matrix which shows the TF-IDF weight of each word in each review. Here, we remove infrequent terms to keep our matrix small while preserving its statistical abilities. Before moving on, we also sort our data, convert it into a dataframe, and inspect our term-document matrix.

```
### Create Word Cloud

set.seed(9162)
wordcloud(words=d$word, freq=d$freq, min.freq=1,max.words=35,
          random.order=FALSE, rot.per=0.3, colors=brewer.pal(8, "Dark2"))
```

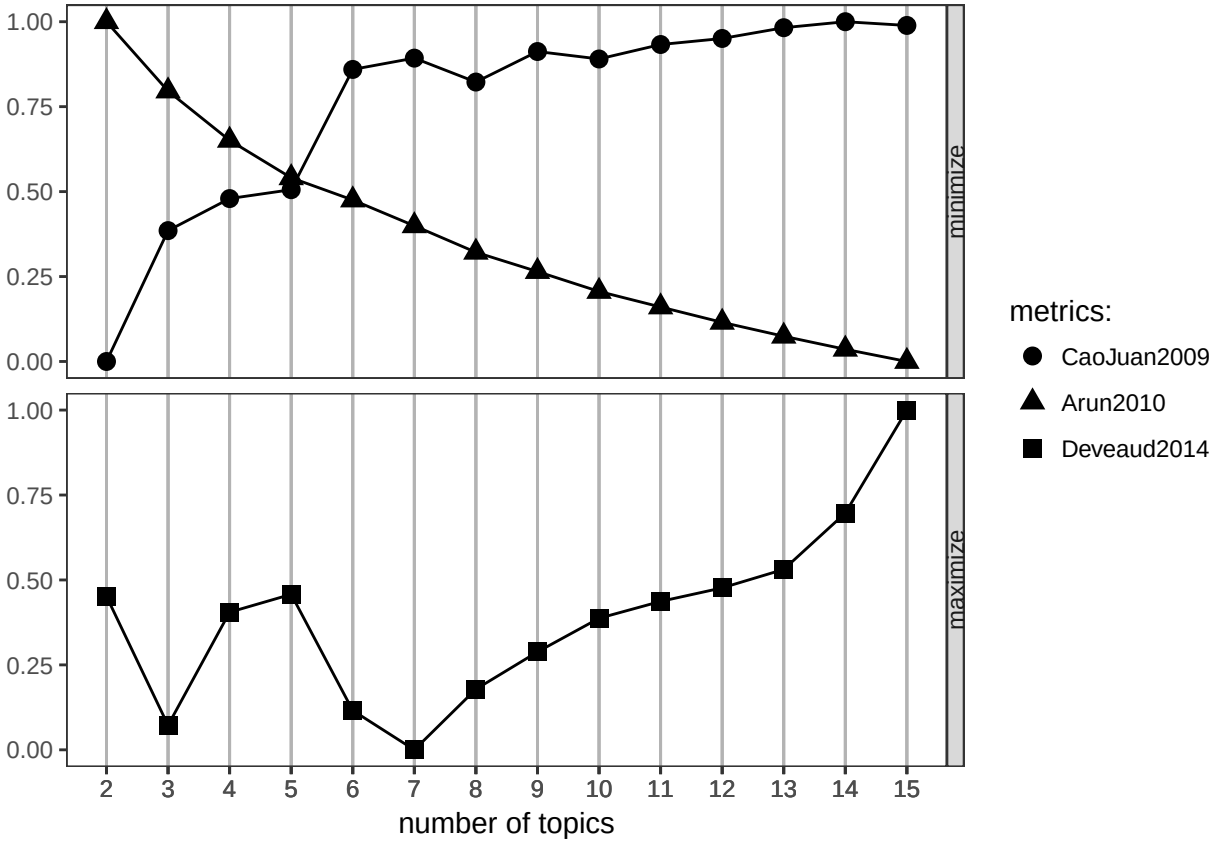
Word clouds are an effective exploratory methods in text-mining as they visually show important words in the corpus. As a result, we receive a quick, general idea of the kinds of words our dataset contains. More important (e.g. prevalent, high TF-IDF weight) words are larger and closer to the center of the cloud.

```
### Topic Modeling Preparation


dtm <- DocumentTermMatrix(reviews.corpus,
       control=list(weighting = weightTf))                    #(Different than preivous tdm)
rowTotals <- apply(dtm, 1, sum)
```

```
dtm.new   <- dtm[rowTotals > 0, ]                           #New tdm for LDA


#Finding ideal number of topics (running 5-15, takes a long time)
idealk <- FindTopicsNumber(dtm.new,
          topics = seq(from=2, to=15, by=1),
          metrics = c("CaoJuan2009", "Arun2010", "Deveaud2014"),
          method = "VEM", control = list(seed=77),
          mc.cores = 2L, verbose = TRUE)
```

```
## fit models... done.
## calculate metrics:
##   CaoJuan2009... done.
##   Arun2010... done.
##   Deveaud2014... done.
```

```
#Plot ideal k
FindTopicsNumber_plot(idealk)
```



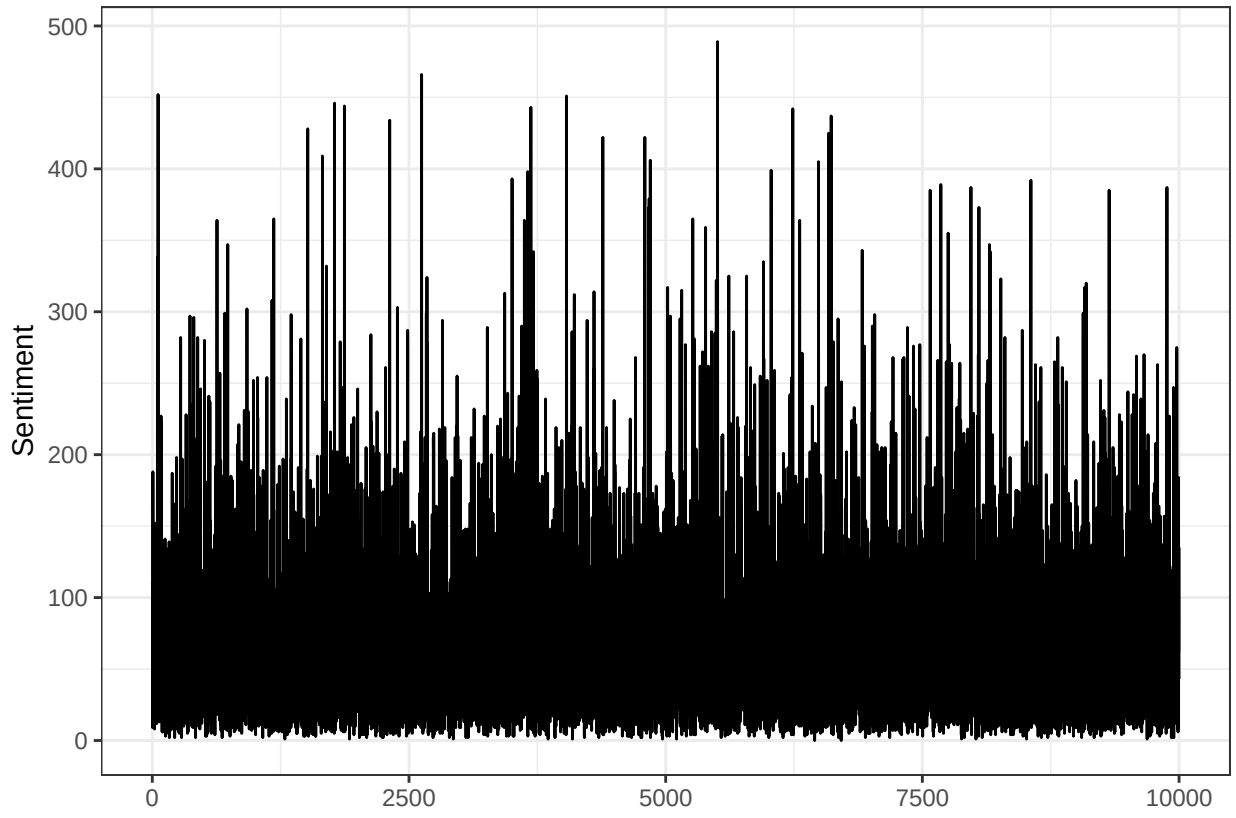
Although we performed our topic modeling in Python, we wanted to see if we could find an optimal number of topics, k. This function uses various metrics to find the answer; however, we did not use these results to proceed with our analysis for various reasons.

1: This method indicates our number of topics should be much higher than 15 (the maximum we tested), but we opted to use 10 because using many topics was computationally expensive

2: This function is designed for finding the best number of LDA topics. We used both LDA and NMF in our analysis.

3: This method takes a long time to run. We are currently running it between 5 and 15 topics and the marginal increase in running time increases with each additional topic

```
### Sentiment Analysis


sent <- analyzeSentiment(reviews.corpus)    #Sentiment Analysis
plotSentiment(sent)                         #Plot sentiments
```




```
binSent <- convertToBinaryResponse(sent)    #Binary response
dirSent <- convertToDirection(sent)         #Directional response

#Create sentiment datasets
sentByMethod <- cbind(binSent[,1], binSent[,2], binSent[,5], binSent[8], binSent[,12])
sentMethods2 <- cbind(dirSent[,1], dirSent[,2], dirSent[,5], dirSent[8], dirSent[,12])

#Write sentiment datasets to CSV files
write.csv(sentByMethod, file="Review Sentiments.csv")
write.csv(sentMethods2, file="Directional Review Sentiments.csv")
```

Finally, we used sentiment analysis to understand how Yelp reviewers felt in their experiences. A quick look at the dataset shows that most reviews were positive, but we wanted a more in-depth understanding as well.

The first thing we did was plot the sentiment of each review. This was not extremely useful for our purposes but could have been for identifying trends over time if our reviews came with a time-stamp.

We then studied the binary and directional sentiments of each review through four different sentiment analysis dictionary. Each dictionary defines different words with a certain positivity or negativity scores, which the

model aggregates. Using a binary response gives a score of "positive" or "negative" for each review and a directional response gives a score of "positive," "neutral," or "negative."

We then output the results of our analysis in CSV files. Each file contains the review number, the number of words in the review which were referenced by all dictionaries, and the label each dictionary provided the review. As a result we are left with four scores for each review. In Excel, we assigned values as such: positive = 1, neutral = 0, negative = -1 and averaged these scores to arrive at an ultimate "average sentiment."

Again, we saw that reviews are mostly positive, but this look provides a more concrete conclusion that simply using star ratings.