Yelp - Optimizing Ads for Lunch with Machine Learning

Nahel Rifai

4/10/2017

The data set used had 20,000+ data points. Each data point represents a review of a restaurant done by a user in the Yelp platform. For each row/data point, there is a dimension (column) that indicates if the restaurant they're reviewing is "Good for Lunch" (1 is Good for Lunch, 0 is not). The "Good for Lunch" variable is reported by restaurant owners; the problem is that many restaurants have not reported if they're good for lunch. The task of this project is to use user reviews of restaurants to predict if a restaurant is "Good for Lunch" (for those restaurants that are missing this variable). We'll use text mining and run a logistic regression to predict whether a restaurant is good for lunch based on its user reviews (thus removing the issue of having to have restaurnt owners report this variable). Once Yelp is able to accurately identify which restaurants are Good For Lunch, they'll be able to optimize the advertisement of these restaurants on their platform around lunch time.

Setting up the libraries and the working directory

```
setwd("~/Desktop/Prasanna/Yelp")
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

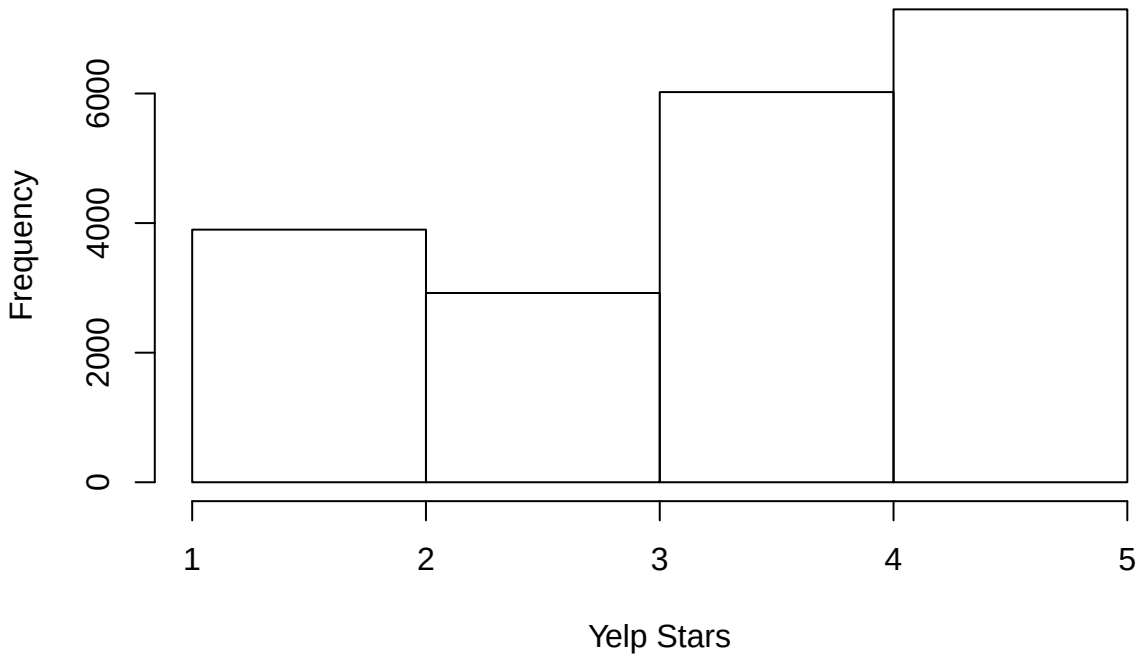
Loading up the yelp csv file

```
yelp_data = NULL
yelp_data = read.csv("YelpReviews_20k.csv", header = TRUE)
```

1. Summary Statistics

"Generate a histogram of all restaurant ratings (i.e. "stars") given by users in the dataset. You can use hist or geom_hist if you prefer to experiment with ggplot"

```
hist(yelp_data$stars, breaks = 5, main = "Restaurant Ratings Distribution", xlab = "Yelp Stars")
```



"How many reviews has the average restaurant in this sample received?"

```
reviews_per_restaurant = yelp_data %>% group_by(business_id) %>% tally()
mean(reviews_per_restaurant$n)
```

```
## [1] 2.438968
```

"How many reviews has the average user in this sample contributed?"

```
reviews_per_user = yelp_data %>% group_by(user_id) %>% tally()
mean(reviews_per_user$n)
```

```
## [1] 1.1614
```

"On average, do restaurants that have been marked as "GoodforLunch" receive a greater number of reviews?"

```
good_for_lunch = NULL
good_for_lunch = yelp_data[!(yelp_data$GoodforLunch== "False"), ]
reviews_per_good_lunch_rest = good_for_lunch %>% group_by(business_id) %>% tally()
mean(reviews_per_good_lunch_rest$n)
```

```
## [1] 2.206227
```

"Do they receive a higher number of stars?"

```
mean(good_for_lunch$stars)
```

```
## [1] 3.721129
```

```
mean(yelp_data$stars)
```

```
## [1] 3.733429
```

2. Exploratory Text Analysis

"Convert the reviews into a text corpus using VCorpus (located in the tm package)."

```
corp.original <- VCorpus(VectorSource(yelp_data$text))
```

"Clean the reviews by eliminating "stopwords", removing whitespace, and converting words to lowercase."

```
corp <- tm_map(corp.original, removePunctuation)
corp <- tm_map(corp, removeNumbers)
corp <- tm_map(corp, content_transformer(removeWords), c("TIL")  ,lazy=TRUE)
corp <- tm_map(corp, content_transformer(removeWords), stopwords("SMART") ,lazy=TRUE)
corp <- tm_map(corp, content_transformer(tolower) ,lazy=TRUE)
corp <- tm_map(corp, content_transformer(stemDocument) ,lazy=TRUE)
corp <- tm_map(corp, stripWhitespace)
```

"Generate a document-term matrix from the Corpus."

```
dtm = DocumentTermMatrix(corp)
m = as.matrix(dtm)
```

"Provide a list of the fifteen most frequently appearing words among all reviews"

```
word.freq = colSums(m)
word.freq = sort(word.freq, decreasing=T)
top15 = word.freq[1:15]
top15
```

```
##    food    good   place   order   great    veri    time  servic     tri
##   15889   13998   13868   10127    9731    8848    8547    8090    6483
##    back  realli restaur  friend    love    onli
##    5919    5908    5671    5358    5236    4414
```

"Generate a word cloud using the document-term matrix, where the word is sized by its frequency in the review"

```
wordcloud(names(word.freq), word.freq, scale = c(3.5, .5),
          max.words = 100, colors = brewer.pal(6, "Dark2"), random.order = F)
```

3. Text analytics and prediction

"How many unique terms are in your document-term matrix?"

```
dtm
```

```
## <<DocumentTermMatrix (documents: 20141, terms: 34255)>>
## Non-/sparse entries: 828056/689101899
## Sparsity           : 100%
## Maximal term length: 109
## Weighting          : term frequency (tf)
```

"Remove sparse terms using RemoveSparseTerms"

```
dtms = removeSparseTerms(dtm, .970)
dtms
```

```
## <<DocumentTermMatrix (documents: 20141, terms: 280)>>
## Non-/sparse entries: 423850/5215630
## Sparsity           : 92%
## Maximal term length: 10
## Weighting          : term frequency (tf)
```

"Calculate the correlation matrix between the words in the document-term matrix and goodforlunch"

```
dtm_matrix = as.matrix(dtms)
yelp_data$GoodforLunch_logic = yelp_data$GoodforLunch == "True"
corr_posneg = cor(yelp_data$GoodforLunch_logic, dtm_matrix)
corr_posneg[1,1:10] ##A sample of the first 10 words in alphabetical order
```

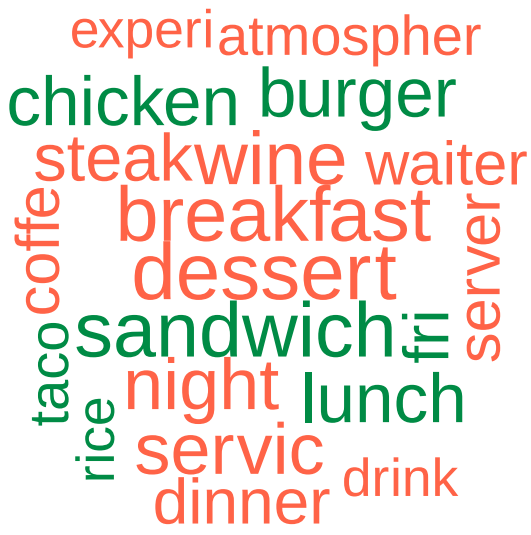
```
##       absolut        actual         alway          amaz           ani
## -0.0118752295 -0.0101189530  0.0191299021 -0.0399779843  0.0043106831
##         anoth         anyth         appet          area         arriv
## -0.0007456817 -0.0133061629 -0.0456038225  0.0050904600 -0.0305574526
```

"Keep only the 200 terms with the highest correlation magnitudes (i.e. positive or negative) Create a new document-term matrix containing only these words."

```
corr = abs(corr_posneg)
top200 = order(corr, decreasing=T)[1:200]
top200words_matrix = colnames(corr)[top200]
```

"Using the top 20 positive and negative words in terms of correlation strength, generate a wordcloud where the size corresponds to the correlation strength, and where positive and negative terms are shown in different colors."

```
top200words = as.data.frame(top200words_matrix) ##we convert new matrix to a df
top200words$ABScorrelation = corr_posneg[1,colnames(corr_posneg)[top200]] ##add corr col
top200words$Sign = corr_posneg[1,colnames(corr_posneg)[top200]] > 0 ##creating color col
colnames(top200words)[3] = "Color" ##renaming color col
top200words$Color[top200words$Color == TRUE] <- "springgreen4" ##color green - words to use
top200words$Color[top200words$Color == FALSE] <- "tomato1" ##color red - words to avoid
top200words$ABScorrelation = abs(top200words$ABScorrelation) ##making correlation ABSOLUTE
wordcloud(top200words$top200words, top200words$ABScorrelation, scale=c(2.5,0.1),
          max.words=20, random.order=FALSE, colors=top200words$Color, ordered.colors=TRUE)
```



"Partition the matrix into training and test rows so you can use the test data to evaluate your model performance. Set the last 20% of your rows aside for testing, and use the first 80% to build your model as specified below."

```
dtm_matrix_new = dtm_matrix
nrows_dtm = nrow(dtm_matrix_new)
eightypct = round(nrows_dtm * 0.8)
dtm.train = dtm_matrix_new[1:eightypct,] ##upper 80%
dtm.test = dtm_matrix_new[(eightypct+1):nrow(dtm_matrix_new),] ##lower 20%
```

"Fit a logistic regression model to the selected variables in the training data."

```
foo = as.data.frame(cbind(goodforlunch =
                            yelp_data$GoodforLunch_logic[1:nrow(dtm.train)], dtm.train))
model = glm(goodforlunch~., data=foo, family=binomial)
```

"Produce a word cloud that separates the top 15 positive words and top 15 negative words. Plot these two groups in different colors."

```
coef = coef(model)[-1]

#Top 15 Positive Terms on a DataFrame
positive.terms = coef[coef>0]
top.positive = sort(positive.terms,decreasing=T)[1:15]
top.positive.df = as.data.frame(top.positive)
top.positive.df$Word = rownames(top.positive.df)
rownames(top.positive.df) = NULL
top.positive.df = top.positive.df[,c(2,1)]
colnames(top.positive.df)[2] = "Correlation"
top.positive.df$Color = "seagreen4"

#Top 15 Negative Terms on a DataFrame
negative.terms = coef[coef<0]
top.negative = sort(negative.terms,decreasing=F)[1:15]
top.negative.df = as.data.frame(top.negative)
top.negative.df = as.data.frame(top.negative)
top.negative.df$Word = rownames(top.negative.df)
rownames(top.negative.df) = NULL
top.negative.df = top.negative.df[,c(2,1)]
```

```
colnames(top.negative.df)[2] = "Correlation"
top.negative.df$Color = "tomato3"

##R Binding Both DataFrames

merged = rbind(top.positive.df, top.negative.df)
merged$Correlation = abs(merged$Correlation) ##Absolute Correlation
colnames(merged)[2] = "ABScorrelation"

##Generating the word cloud

wordcloud(merged$Word, merged$ABScorrelation, scale=c(2.5,1), max.words=30,
          random.order=FALSE, colors=merged$Color, ordered.colors=TRUE)
```

"Using the model you have generated, choose a probability threshold to maximize accuracy and classify the restaurants in your training data as 1 or 0 according to whether they are "GoodForLunch". How well does this model perform on the training data in terms of classification accuracy (i.e. the percentage of GoodForLunch values that you get correct)?"

```
##Setting threshold

predict.df = as.data.frame((dtm.train))
prediction_model = predict(model, predict.df)
predict.df = cbind((prediction_model[] > 0),predict.df)
colnames(predict.df)[1] = "Prediction"

##calculating effectiveness of predictor when used on train.data

effectiveness.df =
  cbind(predict.df$Prediction, yelp_data$GoodforLunch_logic[1:nrow(predict.df)])
effectiveness.df = as.data.frame(effectiveness.df)
colnames(effectiveness.df)[1] = "Prediction"
colnames(effectiveness.df)[2] = "Actual"
effectiveness.df$Match = effectiveness.df$Prediction == effectiveness.df$Actual
effectiveness.df$Match = as.integer(as.logical(effectiveness.df$Match))
temp_counter = effectiveness.df %>% group_by(Match) %>% tally()
Rate_of_Effectiveness = (temp_counter[2,2])/(temp_counter[1,2]+temp_counter[2,2])
```

```
Rate_of_Effectiveness[1,]
```

```
## [1] 0.6715695
```

"Predict values for "GoodforLunch" " in your test data. How well does the model perform in terms of classification accuracy (i.e. the percentage of GoodForLunch values that you get correct)?"

```
##Predicting Good for Lunch in Test Data

predict.test.df = as.data.frame((dtm.test))
prediction_model_test = predict(model, predict.test.df)
predict.test.df = cbind((prediction_model_test[] > 0),predict.test.df)
colnames(predict.test.df)[1] = "Prediction"
effectiveness.test.df =
  cbind(predict.test.df$Prediction,
  yelp_data$GoodforLunch_logic[(nrow(dtm.train)+1):nrow(yelp_data)])
effectiveness.test.df = as.data.frame(effectiveness.test.df)
colnames(effectiveness.test.df)[1] = "Prediction"
colnames(effectiveness.test.df)[2] = "Actual"
effectiveness.test.df$Match = effectiveness.test.df$Prediction == effectiveness.test.df$Actual
effectiveness.test.df$Match = as.integer(as.logical(effectiveness.test.df$Match))
temp_counter = effectiveness.test.df %>% group_by(Match) %>% tally()
Rate_of_Effectiveness = (temp_counter[2,2])/(temp_counter[1,2]+temp_counter[2,2])
Rate_of_Effectiveness[1,]
```

```
## [1] 0.6660874
```