Using Yelp review text to predict user rating

Stéphane Nyombayire

November 17, 2015

Introduction

This project aims at analyzing Yelp review data set to understand whether one could infer the rating based on text analysis of the verbatim review text. The rationale behind this study is both an academic exercise on natural language techniques for deriving insights from text, but more importantly this could help Yelp determine the importance of the verbiage provided by the users' review text to assess whether there is an incremental benefit of having the text for rating a business.

Methods and Data

The dataset used is located here. Loading data:

```
dat <- fromJSON(sprintf("[%s]", paste(readLines(rFile), collapse=",")), 
flatten = TRUE)
```

We used the "yelp_academic_dataset_review.json" file. To read-in the file, we first flatten the data structure into a table for further processing.

```
str(dat)
```

```
## 'data.frame':    1569264 obs. of  10 variables:
##  $ user_id     : chr  "Xqd0DzHaiyRqVH3WRG7hzg"
##  $ review_id   : chr  "15SdjuK7DmYqUAj6rjGowg" "RF6UnRTtG7tWMcrO2GEoAg" "-
TsVN230RCkLYKBeLsuz7A" "dNocEAyUucjT371NNND41Q" ...
##  $ stars       : int  5 2 4 4 4 1 5 5 1 5 ...
##  $ date        : chr  "2007-05-17" "2010-03-22" "2012-02-14" "2012-03-02"
##  $ text        : chr  "dr. goldberg offers everything__truncated__ ...
##  $ type        : chr  "review" "review" "review" "review" ...
##  $ business_id : chr  "vcNAWiLM4dR7D2nwwJ7nCA"...
##  $ votes.funny : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ votes.useful: int  2 2 1 0 2 0 0 0 1 0 ...
##  $ votes.cool  : int  1 0 1 0 1 0 0 0 0 0 ...
```

Many of the techniques applied require to use a document-term matrix as input. To obtain such matrix we have processed each of the reviews to build a bag of words language model. To create this model we preprocessed each document in the corpus as follows:

1. Remove non-writable characters.
2. Strip extra white spaces.
3. Lower case.

4. Remove punctuation
5. Remove numbers
6. Stemming
7. Stop words removal.

After that, each text was tokenized into unigrams, and the unigram frequencies were counted and stored into a document-term matrix of counts.This matrix will serve as the base of our modeling and analysis.

```
matrix <- getDtm(data$text)
```

Exploratory Analysis and Feature Engineering

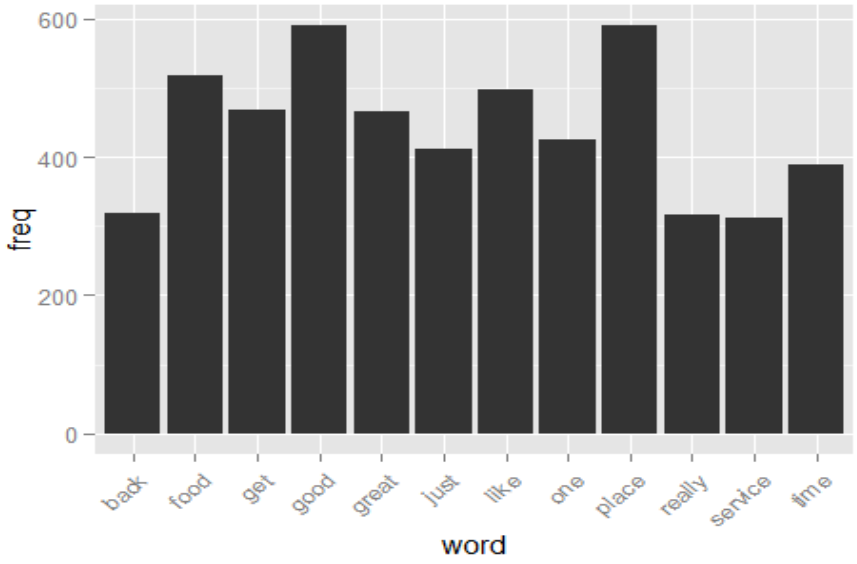
Frequent Terms Classifier

In order to further understand our data set, let's look at the most frequently used terms using the matrix produced above:

```
head(findFreqTerms(matrix, lowfreq=100), 10)
```

```
##  [1] "also"    "always"  "amazing" "another" "area"    "around"  "back"
##  [8] "bad"     "bar"     "best"
```

We noticed that there are words such as: best, nice, love, amazing, better, bad, pretty that are widely used and not too surprising given that these are individual reviews of business. This is already a hint at the need to evaluate sentiment for this feedback. We will get to it later. We can also plot word frequencies (we choose at least 300 mentions):



Alternatively, for better visual consumption we can use a word cloud:

These terms based on frequency of occurence serve as the basis our initial data set that we will be using for modeling.

Sentiment Analysis

Our next strategy with feature engineering would be to infer "sentiment" from each word in our matrix. We will score eac word based on the frequency of usage and as reference for positive, neutral and negative we use the following dictionary:

```
lexicon <- read.csv("subjectivity.csv", head = FALSE, col.names = c('word', '
polarity', 'sentiment'))
str(lexicon)
```

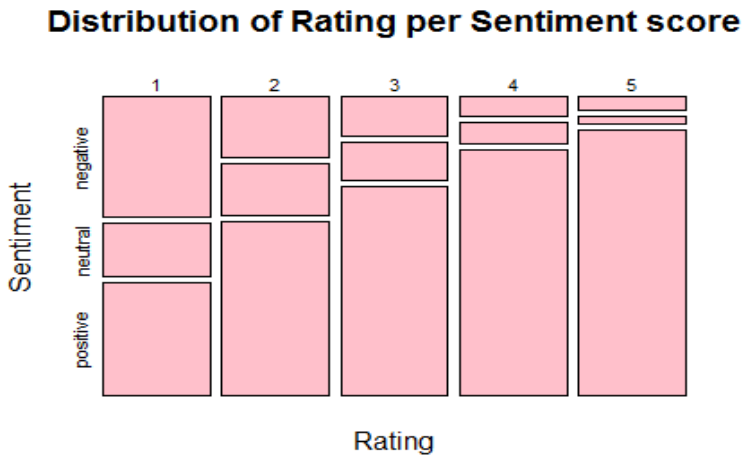
```
## 'data.frame':    6518 obs. of  3 variables:
##  $ word     : Factor w/ 6457 levels "abandon","abandoned",..: 2 3 1 4 ..
##  $ polarity : Factor w/ 2 levels "strongsubj","weaksubj": 2 2 2 1 1...
##  $ sentiment: Factor w/ 4 levels "both","negative",..: 2 2 2 2 2 2 2 2 2 2
```

We will run getSentiment to expand the list of predictors to include the sentiment score as follow:

```
sentiment = getSentiment(matrix)
data$pScore = as.numeric(sentiment[, 1])
data$nScore = as.numeric(sentiment[, 2])
data$rScore = as.numeric(sentiment[,3])
data$sentiment = sentiment[, 4]
```

Now let's see how the sentiment analysis performed, if we assume negative to postive rating from 1 through 5

```
plot(prop.table(table(data$stars, data$sentiment ), 1), main = "Distribution
of Rating per Sentiment score", xlab = "Rating", ylab = "Sentiment", color =
"pink")
```



Topic based modeling

To enhance our features for better modeling, in addition to term frequency and sentiment analysis we will also extract the key topics using Latent Dirichlet Allocation (i.e: LDA) using Gibbs method. Now for each word in our document matrix, we will compute the gamma distance to each topic as follow:

```
lda <- LDA(matrix, method = "Gibbs", control = list(alpha = 0.3), 15)
gammaDF <- as.data.frame(lda@gamma)
names(gammaDF) <- terms(lda)
names(gammaDF)

##  [1] "service", "store", "menu", "breakfast", "cest","cream", "show"
"room", "job", "pizza", "great", "kids", "bar", "good","get"

df <- cbind(data, gammaDF)
```

Let's add the avgRating predictor - we want to examine how average behavior from users perform in our final model, We will also tally up all the votes in another column TotalVotes:

```
dr = ddply(df,.(user_id),transform,avgRating = mean(stars))
dr$TotalVotes = dr$votes.funny + dr$votes.useful + dr$votes.cool
```

Results

We will try two models for this classification problem. SVM (support vector machines) and Decision Tree classifier using RPart. First, we split our data sets in training and validation data sets:

```
inTraining = createDataPartition(d$stars, p = .75, list = FALSE)
training = d[ inTraining,]
testing = d[-inTraining,]
```

SVM

Find below the result from SVM. The model performance is rather poor with accuracy of 38%.

```
svm.model <- svm(stars ~ ., data = training, cost = 100, gamma = 1, type = "C
-classification")
svm.pred <- predict(svm.model, testing[,-1])
confusionMatrix(svm.pred, testing[,1])["overall"]

##       Accuracy          Kappa
##     0.39516129     0.03908248
```

RPart

Decision tree model performs alot better acheiving 97% accuracy with very good recall and specificity as well. Kappa is also pretty high.

```
rpart.model = rpart(stars ~ ., data= training )
rpart.pred = predict(rpart.model, testing[, -1], type = "class")
confusionMatrix(rpart.pred, testing[,1])["overall"]
##       Accuracy          Kappa
##   9.677419e-01   9.563803e-01
```

Discussion

Going back to the original question: "Are we able to predict rating solely base on the review text?". The answer based on our analysis is yes. However, digging deeper our model perfoms rather well when only considering avgRating. Using the same RPart algorithm without average rating we notice a significant drop in accuracy to 39% accuracy. This shows that users dont really change their average behavior when it comes to rating!

Only AvgRating:

```
rpart.model = rpart(stars ~ ., data= subset(training, select = c(stars, avgRating)))
rpart.pred = predict(rpart.model, testing[, -1], type = "class")
confusionMatrix(rpart.pred, testing[,1])["overall"]
##       Accuracy          Kappa
##   9.677419e-01   9.563803e-01
```

No AvgRating:

```
rpart.model = rpart(stars ~ ., data= subset(training, select = -c(avgRating)))
rpart.pred = predict(rpart.model, testing[, -1], type = "class")
confusionMatrix(rpart.pred, testing[,1])["overall"]

##       Accuracy          Kappa
##      0.3870968      0.1509145
```