Predicting Yelp Ratings from Reviews

Introduction

With this report I've investigated using text mining and predictive models in order to guess a rating provided by a user given the corresponding review. This investigation will be useful to several parties. First, it will help Yelp in doing consistency checks with reviews and ratings provided by users and improving their current reviews system. It could also be useful to business owners who are interested in what thoughts in customers lead to high ratings. With a strong prediction model, one could get a good idea of what numerical rating people feel about a place with just access to textual reviews.

Methods and Data:

First I load the necessary packages for my data manipulation, analysis, and modelling. I also set a seed to keep the work reproducible.

```r
library(dplyr)
library(e1071)
library(tm)
library(MASS)
library(SnowballC)
set.seed(123)
```

Then I read in the Review Data. Then I subsetted it so that the only columns were the rating stars and the reviewtext, as all other information is irrelevant to my question. I also added a new feature "ReviewLength" which is the number of chracters in the corresponding revew. I also converted the stars variable into a factor as I'll be performing classification with my models rather than regression.

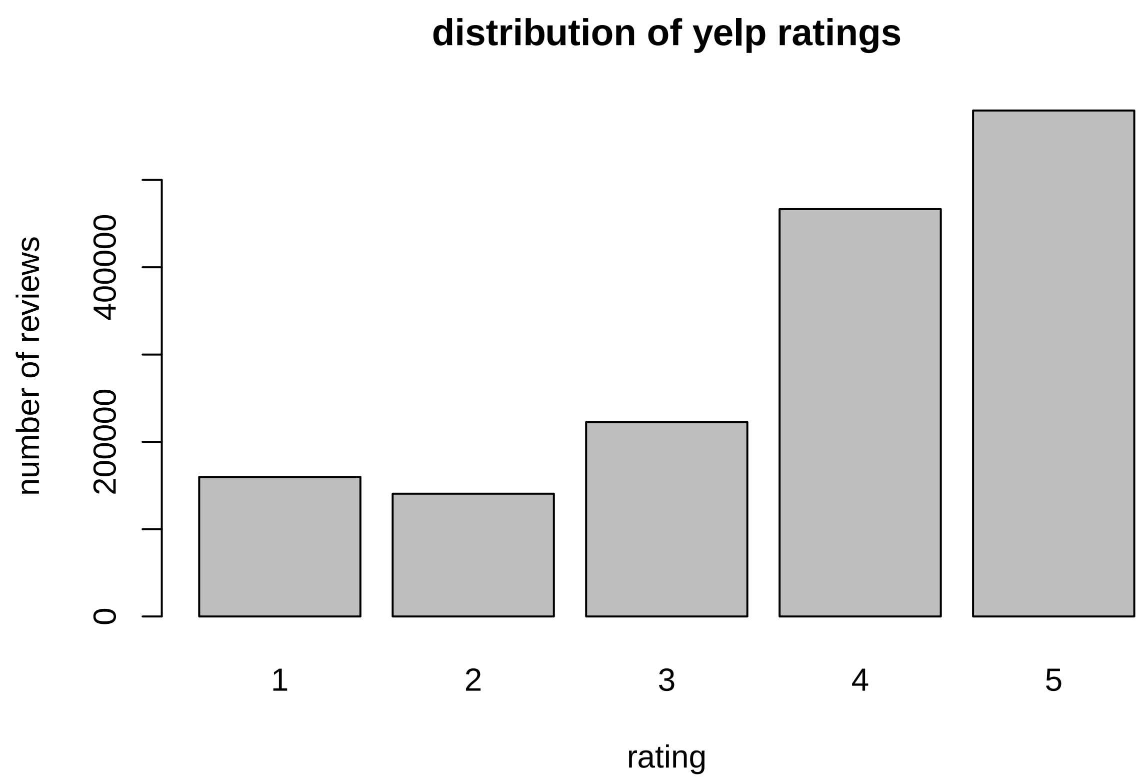
```r
reviews <- readRDS("ReviewData.Rds")
reviews <- reviews[,c("stars", "text")]
reviews <- mutate(reviews, ReviewLength = nchar(text))
reviews$stars <- as.factor(reviews$stars)
```

Here I perform some basic exploratory analysis by creating a table and a barplot of the distribution of ratings provided by users over all of the revews:

```r
table(reviews$stars)
```

```
##
##      1      2      3      4      5
## 159811 140608 222719 466599 579527
```

```r
options(scipen = 5)
plot(reviews$stars, main = "distribution of yelp ratings", xlab = "rating", ylab = "number of reviews")
```



I decided I didn't need to train a model with the entirety of the reviews data provided, so I took a random sample of 10,000 reviews from the over 1 million provided. Of these 10,000, I split them up into a training set of 8,000 and a test set of 2,000.

```
sampleReviews <- reviews[sample(nrow(reviews), 10000),]
TrainDataRows <- sample(nrow(sampleReviews), 8000)
TrainData <- sampleReviews[TrainDataRows,]
TestData <- sampleReviews[-TrainDataRows,]
```

I used the functionality of the "tm" package in order to see how often particular words appeared in each review. The Reviews were transformed into a corpus object and a series of transformations were performed onto the corpus that made it suitable for creating a Document-Term Matrix with words appearing in the reviews as columns and the reviews as rows. One such transformation was to remove words that don't provide any useful information like "the", "their", or "of". Their value would be the TF-IDF weighting rather than just term frequency.

```
corpus <- VCorpus(VectorSource(sampleReviews$text))
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeWords, stopwords("english"))
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, stemDocument, language="english")
DocTermMatrix <-DocumentTermMatrix(corpus,control = list(removePunctuation = TRUE, weighting = function
weightTfIdf(x, normalize = FALSE)))
```

I only decided to look at words that appeared above a certain threshold percent of all the documents. I also convert it so its a R object of class matrix.

```r
DocTermMatrix <- removeSparseTerms(DocTermMatrix, .8)
DocTermMatrix <- weightTfIdf(DocTermMatrix, normalize = TRUE)
DocTermMatrix <- as.matrix(DocTermMatrix)
```

I add a set of new columns to the training and test data, corresponding to the features tf-idf values of the frequently occuring words of interest.

```r
for (i in 1:ncol(DocTermMatrix)) {
  TrainData[,dimnames(DocTermMatrix)$Terms[i]] = DocTermMatrix[TrainDataRows,i]
  TestData[,dimnames(DocTermMatrix)$Terms[i]] = DocTermMatrix[-TrainDataRows,i]
}
```

With that information, I used both an ordinal logistic regression model and a support vector machine model to predict the ratings. An ordinal logistic regression model is a generalization of a normal logistic regression model which applied to ordinal dependant variable. For example dependant variable takes values "very bad", "bad", "good", or "very good". This particular question with the ratings as stars is an applicable case. I used the key words' tf-idf value features for the model, as well as a final feature of the length in characters of the reviews.

```r
OlrModel <- polr(stars ~ back + food + friend + get + good + great + just + like + one + order + place +
SvmModel <- svm(stars ~ back + food + friend + get + good + great + just + like + one + order + place +
```

Results

I initially trained the models with the training set and evaluated how effective they were at predicitng ratings by using them on the test set.

Here is a table of predicted ratings as rows versus actual ratings as columns for the ordinal logistic regression model:

```r
table(predict(OlrModel, TestData), TestData$stars)
```

```
##
##       1   2   3   4   5
##   1   5   3   1   6   0
##   2   0   0   0   0   0
##   3   0   0   0   0   0
##   4 118 115 129 227 191
##   5  85  75 127 370 548
```

And here is a table of predicted ratings as rows versus actual ratings as columns for the support vector machine model:

```r
table(predict(SvmModel, TestData), TestData$stars)
```

```
##
##       1   2   3   4   5
##   1   9  10   3   3  10
##   2   0   0   0   0   0
##   3   0   0   0   0   0
##   4  60  88 143 285 201
##   5 139  95 111 315 528
```

Both models used showed moderately effective results. The ordinal logistic regression model correctly predicted the rating 39% of the time, whereas the support vector machine model correctly predicted the rating 42% of the time. Neither model predicted Reviews to have 2 or 3 stars. This is because the distribution of the ratings is heavily skewed towards 4 and 5 stars, so often the models still would predict these as even certain negative trigger words are not enough to overcome the default high chance the rating is 4 or 5.

Conclusion

In conclusion, the models I created accurately predicted the exact rating for almost half the cases. Although that doesn't immediately appear to be very high, the fact there are 5 choices suggests its a quite good predictor. However, I don't see any practical implications of these results being used by Yelp or business owners seeing as it appears words from reviews don't have an extremely powerful effect on ratings, and such variation may just come from how an individual relays words to numbers. "This restaurant was good" may mean giving a 3 star rating for some and a 5 star rating for others. Nonetheless, it is clear that there its lot of potential in using similar text mining techniques to discover potentially useful predictive patterns and relationships.