Yelp Challenge Project - Review Rating Prediction

David Álvarez Pons

Introduction

The Yelp company gathers reviews from users attending to restaurants, clinics and other services. Each review contains a star rating that is given by a user to the rated business. Combining all the ratings of all users gives a total rate for each business. The rating is given by the users and, in theory, it is based on the text of the review. Asking for a rating to a user that has already written a review is asking for duplicate information and storing duplicate information as well. Therefore, can we infer the rating of a review looking at the text written by the user and other meta data such as location, date, etc.? This way the criteria of ratings is unified and also the amount of information stored decreases which results in benefits for both the users and the Yelp Company.

This project presents a first approach of the implementation of **a prediction function for star-rating in reviews based on learning from words in the existing rated reviews**. In the first section, we will present how to obtain the data we used for this project as well as a description of all the phases the data has passed through to reach the final model, including data preprocessing, exploratory analysis, cleaning data and prediction modeling. After that, we will present, summarize and interpret the results obtained as well as the errors of the prediction model. Finally, we will present a brief discussion of the project results and methodology.

This report intends to be as brief as possible, as well as a reference for reproducible research. That is why, we provide the main explanations in this report, but code chunks and detailed information in the Appendix available here. Reading the Appendix is not essential to understand the main process followed in this project. For even deeper information, please, refer to this repository: https://github.com/davizuku/datascience-capstone.

Methods and Data

In this section, we describe the whole cycle that the data has followed from the downloaded zip file to the final prediction model. First, the preprocessing for adapting the downloaded data to manegable R data structures. After that, we present the exploratory analysis taken to get some knowledge of the downloaded data related to our problem. Finally, we go through the process of building the prediction model including cleaning data.

Data preprocessing

Coursera has provided us a URL with the data available for this project: https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/yelp_dataset_challenge_academic_dataset.zip. Once downloaded the data and extracted we can observe the following data files:

```
## [1] "yelp_academic_dataset_business.json"
## [2] "yelp_academic_dataset_checkin.json"
## [3] "yelp_academic_dataset_review.json"
## [4] "yelp_academic_dataset_tip.json"
## [5] "yelp_academic_dataset_user.json"
```

The data is stored as JSON files. Using the package `jsonlite` we have read them and transformed into R `lists`. Due to the large computation time required to read and transform the files, we have saved these lists into RDS files. We will use these files in the next section.

Exploratory analysis

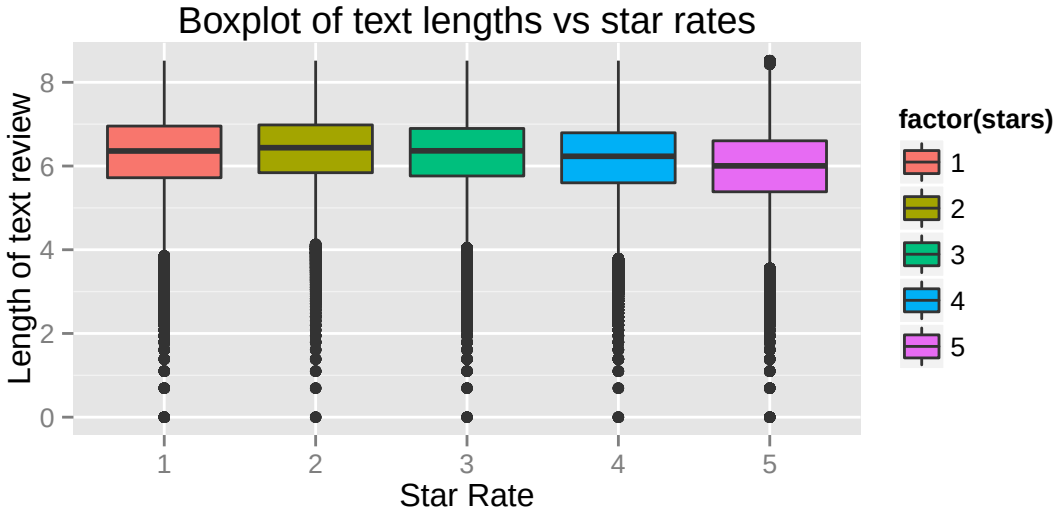
The main goal of this section is to know more about the data, how it is organized, which relations exist and to see the correlation between reviews text and star-rating.

Our data is split in several files, but the information is linked using ids. For example, the `business` dataset has a property called `business_id` that identifies uniquely the business. At the same time, the `review`, `checkin` and `tip` datasets reference the business by the same `business_id` property.

Let's use this connection to make a simple count of how many reviews has each of n businesses chosen at random from the whole `business` dataset. We will compare the computed count with the existing field `review_count`. The comparison table is available in the appendix.

The problem that we would like to solve with the data set is the prediction of the review star rating using the available data related to the review, mainly the free text field.

Since we want to predict the star rating of the reviews, let's have a look at the relationship between the star rating and the length of the review text.



We observe a very slight trend towards a better rating as the text length decreases. However, it seems negligible, but let's make a hypothesis test under $H_0 : cor = 0$.

```
cor.test(log(textLengths), stars)
```

```
##
##  Pearson's product-moment correlation
##
## data:  log(textLengths) and stars
## t = -184.2916, df = 1569262, p-value < 2.2e-16
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1470798 -0.1440169
## sample estimates:
##        cor
## -0.1455487
```

However, it is not representative enough, since the small p-value leads to reject the alternative hypothesis: $H_\alpha : cor \neq 0$.

Cleaning data

Despite having transformed the original text-plain JSON files into `lists`, this type of data in R cannot be easily used to process or perform queries on datasets. Therefore, we have decided to transform it to `data.frame` so we can easily perform further analysis on this data. Together with this transformation, in this section we describe the feature selection performed for further analysis.

Using the functions available in the appendix, we have selected a set of features at the same time that we transformed the R `lists` of data into `data.frames`. The resulting dataset has the following column names:

```
##  [1] "review_id"          "business_id"        "business_name"
##  [4] "business_latitude"  "business_longitude" "business_city"
##  [7] "business_state"     "business_open"      "user_id"
## [10] "date"               "votes_funny"        "votes_useful"
## [13] "votes_cool"         "text"               "stars"
```

Once we have selected the features, it is time to process the text of the reviews. For that we use the `tm`, `wordnet`, `openNLP` and `SnowballC` packages. The second package requires an additional database to be downloaded and installed. Instructions on how to download, install and confiture the `wordnet` package are available in the appendix.

Using these packages we have created two functions (`splitReviewsIntoSentences` and `normalizeSentences`) to be used as a preprocessing function for a given data set. Having these functions we can apply them to the `training` and `testing` data set when generating the prediction algorithm.

The text mining process is the following. First, we split the text of each review into its sentences with the help of the `openNLP` package. Then, we transform each sentence by using the tools of the `tm`, `wordnet` and `SnowballC` first by deleting stopwords, numbers, punctuation, word sufixes. After that, we sort the words of each sentence alphabetically deleting the duplicated words. As a result, we have *normalized* the texts of the reviews.

Due to the increasing amount of complexity and the exponentially increase in size, in order to keep execution times affordable, we have decided to choose a subset of the reviews selecting randomly a 2% of the businesses, resulting in a total of ~30k reviews.

Prediction modelling

After all the preprocessing functions, we have a data set with *normalized* sentences. In this section, we will describe the process for building the prediction model using these sentences.

This data set consists of a table with rows containing normalized or *hashed* sentences, the *star rating* and other *meta data* relative to the business and original review. In the preprocessing stage, we accomplished to map similar sentences to the same value, first splitting reviews into sentences and then *normalizing* them.

Although the ratings of reviews can be considered as 5 categories, we will face this part of the problem as a regression model. Lately we will treat numerically the ratings of the sentences and having more flexibility will likely make the final prediction algorithm perform better.

Before building any kind of prediction model, first we have to split the data into `training` and `testing` sets in a ratio of 75-25%.

```
set.seed(4587);
inTrain = createDataPartition(nsent$stars, p = 3/4)[[1]]
training = nsent[ inTrain,]
testing = nsent[-inTrain,]
```

The prediction model learns form the words in the sentences of the `training` set. We count the occurrences of each word as the number of sentences that it appears in. At the same time, we compute the average star-rate of all the sentences where it appears in (exact algorithm in the appendix section). After this process, we extract a kind of *semantinc* information of each of the words, including the number of appearances and the total sum of stars that the sentences have received.

Here we provide some examples of *good* and *bad* words.

```
##           word count stars avg.stars
## 4975   horribl    24    28  1.166667
## 5260    orbitz     6     7  1.166667
## 7795 hesitated     6     7  1.166667
## 3     favorite  1496  6488  4.336898
## 94   delicious  1573  6648  4.226319
## 172  sweetness    57   242  4.245614
```

Having this data, the function *simply* computes the weighted sum of the rate of the sentence's words. We have considered that, having deleted stopwords, a word that appears a lot is more relevant than other that does not appear that often. That is why, we give it more importance to the final prediction.

```
predictSentence <- function(sentence){
    trainWordsData <- words[words$word %in% unlist(strsplit(sentence, " ")),]
    totalCount <- sum(trainWordsData$count);
    sum((data$count / totalCount) * data$avg.stars)
}
```

Results

In this section, we will analyze the results of applying the prediction function to our `testing` data set.

```
pred <- predictDataset(testing)
saveRDS(pred, "data/testing_predictions.rds")
```

First of all, we compute the Root Mean Squared Error on the predictions:

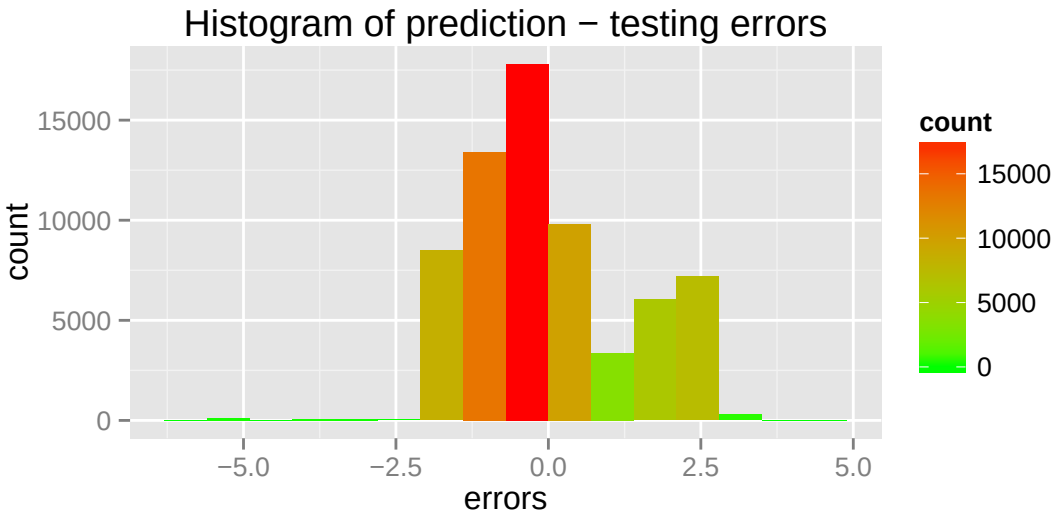
```
errors <- pred - as.numeric(testing$stars)
summary(errors)
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -5.000000 -1.198000 -0.343700 -0.006751  0.714300  4.000000
```

```
rmse <- sqrt(sum(errors^2) / length(pred)); rmse
```

```
## [1] 1.31582
```

From the error reports obtained, we observe that **the prediction function is a bit too positive** in predicting the ratings. That is, it is very difficult to obtain lower ratings than higher ones. The following figure represents this positivity.



Discussion

In this project we have tried to predict the ratings of the review by using a naive count-based approach. First, we have split the reviews into sentences in order to treat each sentence separately. However, the first intend of using synonyms for the words was not affordable due to the huge execution time that would have been required for this computation. Instead, we have chosen to use single word counting and average the stars assigned to the review that contained each word to get a sense of the influence that each word has in the rating of the review. Combining this influences in a weighted sum gives our prediction function, which does not have a bad Root Mean Square Error, but can be improved quite a bit.

Other approaches that we could not take were the following:

1. Consider applying cross validation in our `training` dataset for a more accurate data analysis.
2. Consider using synonyms in words so each word is *hashed* to its first synonym, getting a more uniform set of words and a better relationship of word - rating.
3. Consider gathering sets of two words, three words, etc. into the prediction function. This way combinations such as `don't like` can be associated to a negative value, instead of having `don't` and `like` as a separated words.
4. Deriving from the previous consideration appears the idea of hashing sentences using synonyms and ordering of words. Instead of gathering sets of fixed number of words, do it with entire sentences.
5. We have not exploited all the benefits of using the `tm` library. Creating a Corpus would have allowed us to reach further levels of analysis in Text Mining.
6. Consider using other meta data e.g. votes or date to refine weights in words/sentences.

Finally, we have to mention that the main problem we have face during this project has been the size of the data. We decided to do analysis over the `review` dataset. Although we have subset the database to only 2% of the businesses -resulting in a dataset of ~30k reviews instead of the 1.5M of the original dataset-; the execution time of functions for processing the whole dataset was *mesured in hours* which lead to a very simple pipeline, and thus, a simple prediction model.