Capstone Project: Sentiment Analysis as a Tool to Predict the Outcome of a Review

Bruno Ribeiro

Abstract: Sentiment Analysis is considered the most common text classification tool that analyses an incoming message and tells what the underlying sentiment is. Often referred to as to as opinion mining or emotion artificial intelligence, the goal of sentiment analysis is the same: to know a user or audience opinion on a target object by analyzing a vast amount of text from various sources. This project's goal was to use a machine learning algorithm to extract the sentiment contained into a review and try to predict the outcome of such review, as Good or Bad, using only the text that was written, regardless the user or product. It was used a Support Vector Machine Algorithm to predict the outcome of the review and the performance was measured by the accuracy with which it predicted the outcome.The final result was an accuracy of 87.7%.

1. Introduction

Sentiment analysis, often referred to as opinion mining or emotion artificial intelligence, consist of a type of text research which makes use of a mix of statistics, natural language processing (NLP) and machine learning techniques to identify and extract subjective information from text files. Regardless of the name, the goal of sentiment analysis is the same: to know a user or audience opinion on a target object by analyzing a vast amount of text from various sources.

At its core, sentiment analysis aims to classify the polarity of a given text at the document (sentence or feature) as positive, negative or neutral. However, there are some advanced techniques that go "beyond polarity" sentiment classification and try to identify the emotional states, such as "angry", "sad", and "happy". Considering a world where we generate 2.5 quintillion bytes of data every day, sentiment analysis has become a key tool for making sense of that data.

In other words, we can say that sentiment analysis allows us to shed some light into all those unstructured information by transforming it into some sort of structured data of public opinions about products, services, brands, politics, or any topic that people can express opinions about. Once it's done, this information can be very useful for commercial applications like marketing analysis, public relations, product reviews, net promoter scoring, product feedback and customer service.

This project's goal was to use a machine learning algorithm to extract the sentiment contained into a review and try to predict the outcome of such review, as ***Good*** or ***Bad***, using only the text that was written, regardless the user or product. This work is a requirement to achieve a passing grade in the Data Science Professional Certificate Program, provided by Harvard University through edX.

In this project, primarily, the following key steps were performed:

1. Analyse the gathered data;
2. Define the predictors to be used to predict the outcome for the review;
3. Train and evaluate a machine learning algorithm;

1.1. The Data

The data used in this project was the Amazon Fine Food Reviews, obtained from Kaggle and consists of reviews of fine foods from amazon from October 1999 to October 2012. There are 568,454 reviews, from

256,059 users and 74,258 products. Reviews include product and user information, ratings, and a plain text review. It also includes reviews from all other Amazon categories.

The variables existing in the dataset are:

1. ***Row Id***.
2. ***ProductId:*** Unique identifier for the product.
3. ***UserId:*** Unqiue identifier for the user.
4. ***ProfileName:*** Profile name of the user
5. ***HelpfulnessNumerator:*** Number of users who found the review helpful.
6. ***HelpfulnessDenominator:*** Number of users who indicated whether they found the review helpful or not
7. ***Score:*** Rating between 1 and 5
8. ***Time:*** Timestamp for the review
9. ***Summary:*** Brief summary of the review
10. ***Text:*** Text of the review

The dataset used in this project can be downloaded in ***https://www.kaggle.com/snap/amazon-fine-food-reviews***.

2. Methods

After downloading the dataset form Kaggle website, it was performed some feature engineering to turn it into a format that would allow it to be properly analyzed. ***(1)*** since the goal of the project was to predict the outcome of a review, as a 'Good' or a 'Bad' review, using only the review's text, every other columns but text and score were removed; ***(2)*** there are some people that write a 'whole story' in the review. So, in order to keep a 'standard', reviews with more than 10 sentences were removed; ***(3)*** for simplicity purposes, scores equals 3 were removed; ***(4)*** the scores were 'binarized': Bad (1 and 2) and Good (4 and 5).

Once the steps above were taken, the reviews were broken into sentences and the sentiment of each sentence were extracted, replacing the texts with its values. After extracting all sentiments, the dataset was transformed to a sparse format and then split into a train set, containing 80% of the data, and a test set, with 20% of the data.

It was used a Support Vector Machine algorithm in the train set and then it was used to predict the outcomes of the test set. The model was tested using the accuracy metric.

Note that it wasn't performed any cleaning in the texts.

Table 1: Datasets dimensions.

	Raw.Dataset	Sparse.Dataset
Rows	568454	489194
Columns	10	12

2.1. Hardware Specifications

This project was done on a computer with an Intel Core i7-6700 CPU @ 3.40GHz and 32Gb DDR4 @ 2133MHz of RAM, running Ubuntu 18.04.1 LTS. All the scripts and algorithms were built on R Language, version 3.5.2. The whole session information, including the packages used, can be found in Appendix 1.

3. Results

3.1. Data Exploration

The Amazon Reviews dataset was composed of 568,454 entries of 10 variables, ranging from October 1999 to October 2012. The datased comprises the reviews of 256,059 users and 74,258 products, providing a rating between 1 and 5 with no intermediate values (i.e. 0.5). There was no missing values in the dataset.

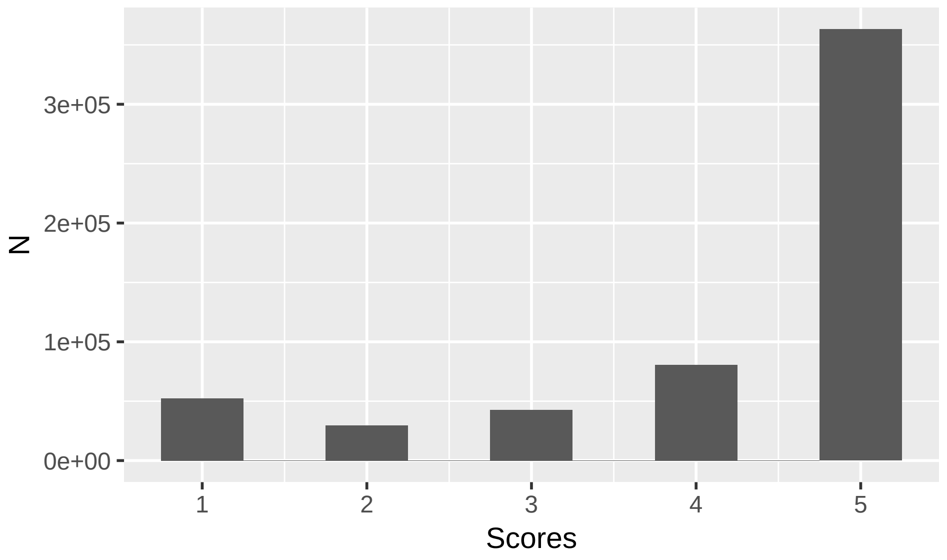

Figure 1: Scores distribution prior to 'binarization'.

A quick look at the plot above shows that the dataset is quite unbalanced: There's a lot of 5 when compared to the other existing scores. However, problems concerning unbalanced datasets will no be adressed here.

It's important to note that some people write more than others. While some people are very straight in their reviews, other write a whole story about the products they bought. This can be seen in Figure 2, below: There are some pretty big reviews. However, it can be observed that the average number of senteces per review is around 4, and it starts droping right after that.

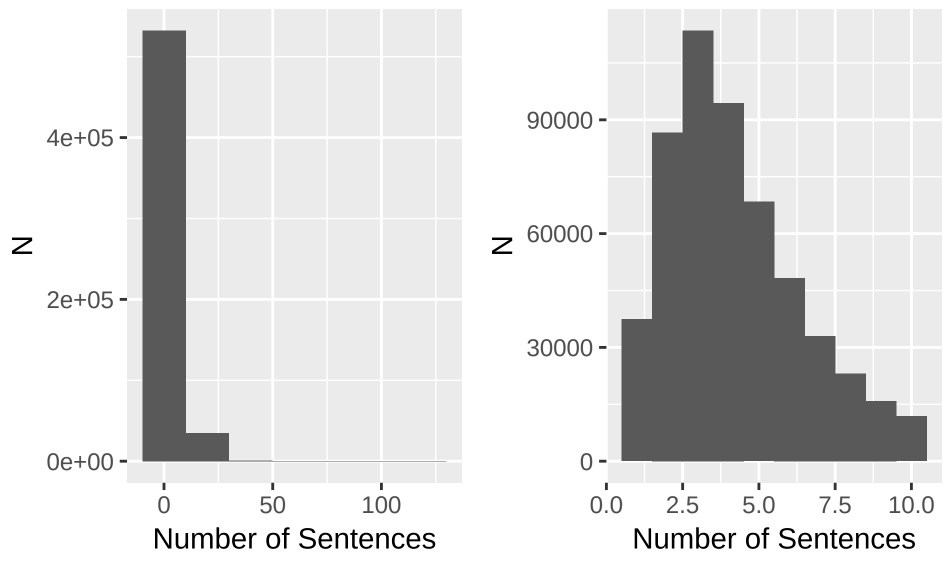

Figure 2: Number of Sentences per Review. On the left, the raw number of sentences per review. On the left, a filtered distribution with less or equal 10 sentences per review.

3.2. Modelling

After filtering the number of sentences, removing the scores equals 3 and binarizing the scores (Good or Bad), it was necessary to separate the sentences of each text review. The ***sentimentr*** package requires this to be done in order to improve the RAM usage and the speed of the 'sentiment extraction', to be performed. Below is a sample of how the ***get_sentences()*** function returns the reviews (the text was removed to fit the page).

Table 2: Sentences extraction output

score	element_id	sentence_id
Good	1	1
Good	1	2
Good	1	3
Bad	2	1
Bad	2	2
Bad	2	3

Once the sentences were 'split', the sentiment of each sentence was extracted. The sentimentr output, is a 'data frame' containing the text of the review, the score, the id of the review, the id of the sentence, a word count and a sentiment value.

Table 3: Sentiment output

score	element_id	sentence_id	word_count	sentiment
Good	1	1	21	0.7092081
Good	1	2	15	-0.0903696
Good	1	3	12	0.5167285
Bad	2	1	7	0.0000000

score	element_id	sentence_id	word_count	sentiment
Bad	2	2	7	0.0000000
Bad	2	3	18	-0.4831896

Once the sentiments were all extracted, the unnecessary columns were removed and the dataset made into a sparse matrix, containing the sentiment of each sentence of the reviews and the score given.

Table 4: Sparse Matrix of Sentiments (head)

score	element_id	1	2	3	4	5	6	7	8	9	10
Bad	2	0.0000000	0.0000000	-0.4831896	0.0000000	0.000000	0	0	0	0	0
Bad	4	0.1500000	0.1677051	0.0000000	0.0000000	0.000000	0	0	0	0	0
Bad	13	0.1386750	0.3207135	0.1539601	0.1206045	0.213809	0	0	0	0	0
Bad	17	0.3872983	-0.3354102	0.2132007	0.0000000	0.000000	0	0	0	0	0
Bad	27	0.0000000	0.0000000	0.0000000	0.0000000	0.000000	0	0	0	0	0
Bad	47	-0.3354102	-0.0566947	0.0000000	0.0000000	0.000000	0	0	0	0	0

3.3. Machine Learning Algorithm

The training set, created from the sparse matrix of sentiments, was submited to a Support Vector Machine algorithm to create the model for predicting the outcomes of the reviews. Using only the text of the reviews, the model was able to predict the outcomes with **87.70%** of accuracy.

4. Conclusion

Sentiment Analysis is, perhaps, the most common text classification tool that analyses an incoming message and tells what the underlying sentiment is. The use of this kind of analysis can help companies to understand the social sentiment of their brand, product or service while monitoring online conversations. Modern packages, like ***sentimentr***, attempts to take into account valence shifters (i.e., negators, amplifiers (intensifiers), de-amplifiers (downtoners), and adversative conjunctions), while maintaining speed, and providing a better analysis of the sentiment of the text.

Just using the text of the reviews, without any cleaning, the SVM algorithm, after some 'sentiment processing', was able to predict the true outcome of a review with ***87.7%*** of accuracy. With a proper cleaning (mispelling corrections, for example), and also creating some sort of baseline, it's possible to greatly improve the accuracy.

APPENDIX 1: Session Info

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 18.04.2 LTS
##
## Matrix products: default
## BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.7.1
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.7.1
##
## locale:
##  [1] LC_CTYPE=pt_BR.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=pt_PT.UTF-8        LC_COLLATE=pt_BR.UTF-8
##  [5] LC_MONETARY=pt_PT.UTF-8    LC_MESSAGES=pt_BR.UTF-8
##  [7] LC_PAPER=pt_PT.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=pt_PT.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] gridExtra_2.3   forcats_0.3.0   stringr_1.4.0   dplyr_0.8.0.1
##  [5] purrr_0.3.0     readr_1.2.1     tidyr_0.8.3     tibble_2.0.1
##  [9] ggplot2_3.1.0   tidyverse_1.2.1
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5  haven_2.0.0       lattice_0.20-38
##  [4] colorspace_1.4-0  htmltools_0.3.6   yaml_2.2.0
##  [7] rlang_0.3.1       syuzhet_1.0.4     pillar_1.3.1
## [10] textclean_0.9.3   glue_1.3.0        withr_2.1.2
## [13] lexicon_1.1.3     modelr_0.1.2      readxl_1.1.0
## [16] plyr_1.8.4        sentimentr_2.6.1  munsell_0.5.0
## [19] gtable_0.2.0      cellranger_1.1.0  rvest_0.3.2
## [22] evaluate_0.12     labeling_0.3      qdapRegex_0.7.2
## [25] knitr_1.20        highr_0.7         broom_0.5.0
## [28] Rcpp_1.0.0        scales_1.0.0      backports_1.1.2
## [31] jsonlite_1.6      hms_0.4.2         digest_0.6.18
## [34] stringi_1.3.1     grid_3.5.2        rprojroot_1.3-2
## [37] cli_1.0.1         tools_3.5.2       magrittr_1.5
## [40] lazyeval_0.2.1    crayon_1.3.4      pkgconfig_2.0.2
## [43] data.table_1.12.0 xml2_1.2.0        lubridate_1.7.4
## [46] assertthat_0.2.0  rmarkdown_1.10    httr_1.3.1
## [49] rstudioapi_0.8    R6_2.4.0          nlme_3.1-137
## [52] compiler_3.5.2
```

APPENDIX 2: R Script

```r
#install.packages(tidyverse)
#install.packages(sentimentr)
#install.packages(caret)
#install.packages(sentimentr)
#install.packages(quanteda)
#install.packages(e1071)
library(tidyverse)
library(sentimentr)
library(caret)
library(sentimentr)
library(quanteda)
library(e1071)

reviews <- read_csv('data/Reviews.csv')

# Check the dimensions
reviews %>%
  dim

# Take a look at the first entries
reviews %>%
  head()

# Since the names are all in upper case, I'll turn it to lower. Also, I'm only interested
# in the score and text variables.

# Turn the dataset names to lowercase
names(reviews) <- reviews %>%
  names %>%
  tolower

# Filter
reviews <- reviews %>%
  select(text, score)

reviews %>%
  head()

# I won't bother in cleaning the text. i.e. mispellings, emojis and everything else.
# I'll deal with the text reviews 'as it is'.

# Plot our target variable to see its distribution
reviews %>%
  pull(score) %>%
  qplot(binwidth=1,
        main='Number of Scores', xlab='Scores', ylab='N')

# It's a very unbalanced dataset. There's a lot of 5 when compared to the other
# existing scores. However, I'll no be dealing with it here either.

# Check how the number of sentences are distributed.
```

```r
n_sentences <- reviews %>%
  pull(text) %>%
  nsentence()

# Check the range
n_sentences %>%
  range

#Plot the number of sentences distribution
n_sentences %>%
  qplot(binwidth=20,
        main='Number of Sentences per Text', xlab='Number of Sentences', ylab='N')

# The majority of the texts reviews have less than 20 sentences. So I'll narrow it down
# to the maximum of 10 sentences per review. # Since it'll, later, become a sparse dataset,
# the sentences reduction will be an important step.

index <- n_sentences<=10
reviews <- reviews[index, ]
reviews %>%
  dim()

# For the sake of simplicity, and to improve calculations time in this project, I'll remove
# the scores equals 3, i.e. neutral score.

reviews <- reviews %>%
  filter(score!=3)

reviews %>%
  dim()

# Turn scores into binary terms: Good (4 and 5) and Bad (1 and 2). Also, turn it into factors
reviews <- reviews %>%
  mutate(score=ifelse(score>2, 'Good', 'Bad')) %>%
  mutate(score=as.factor(score))

reviews %>%
  pull(score) %>%
  qplot(main='Binary Scores', ylab='N')

##########################################################

#                         Modelling

##########################################################

# Separate the sentences of each text review. This improves the RAM usage and the speed of the
# 'sentiment extraction' by the sentimentr package

reviews <- reviews %>%
  get_sentences(text)

# Now, extract the sentiment of each sentence
```

```r
sentiments <- sentiment(reviews)

# The sentimentr output, is a data.frame with the text, the score, the element id, which is the
# review, the sentence id, which is the sentence of the review, a word count and a sentiment value.
# I'll remove the columns that aren't necessary for the analysis.

# Also, since quanteda and sentimentr count the number of sentences differently, I'll filter the
# number of sentences again, to keep on 10 sentences per review.

# Remove reviews with more than 10 sentences

#Select the elements ids with 10 or less sentences.
elements_10 <- sentiments %>%
  mutate(count=1) %>%
  group_by(element_id) %>%
  summarize(S=sum(count)) %>%
  filter(S<=10) %>%
  pull(element_id)

sentiments <- sentiments %>%
  filter(element_id %in% elements_10) %>%
  select(-word_count, -text) # These two columns are not necessary any longer.

# Now, we turn this sentiments dataset into a sparse matrix to consider each sentence sentiment
# as a variable for that review.

sparse_sentiments <- sentiments %>%
  spread(sentence_id, sentiment, fill=0)

set.seed(221)
index <- createDataPartition(y=sparse_sentiments$score, times=1, p=0.20, list=FALSE)
train_set <- sparse_sentiments[-index, ]
test_set <- sparse_sentiments[index, ]

# I'll fit a SVM model with radial kernel
# I don't recommend to run this. It will take a few hours to run.
fit <- svm(score ~., train_set, kernel='radial')
y_hat <- predict(fit, test_set)
mean(y_hat==test_set$score)

# Naive bayes algorithm can also be used. It provides a small different accuracy than SVM.
# However, it runs much, much faster.

#fit_bayes <- naiveBayes(score ~., train_set)
#y_bayes <- predict(fit_bayes, test_set)
```