NLP Assignment 3
Sentiment Analysis

Alberto Parravicini

1 Introduction

The goal of the assigment is to experiment with different techniques used for sentiment analysis, and compare the results obtained.
As a starting point, it was used the dataset of **Digital Music** reviews on **Amazon**, compiled by **Julian McAuley**. The goal was to predict the score given to a product by analysing its review, by taking advantage of *Natural Language Processing* and *Machine Learning*.
This report will present various preprocessing and modelling techniques that have been tried, such as **Latent Semantic Analysis** and **Support Vector Machines**, and discuss their efficiency.

The first section of the report will detail the data that have been used, and the preprocessing techniques applied to them.
The second section is focused on the models that were used for sentiment analysis, and on the selection and validation techniques that have been adopted. The third and last section will present the results of the models, and discuss problems and potential improvements that can be adopted.

2 Data analysis and pre-processing

The dataset used in the assignment is a collection of **Digital Music** reviews of songs and albums sold on **Amazon**.
Each review is stored as a **JSON**, with different fields such as the *reviewer name*, the *review date*, how many people found it *useful*, and more.
Our goal is to process the **review text** (and the **review title**), in order to predict the review score. Scores range on a $1 - 5$ scale, and can be interpreted as the **sentiment** of the reviewer towards the product he has bought.
It should be noted that if our goal was to predict the scores as accurately as possible, then all the information in the review should be taken into account

(such as the *reviewer name*); however, our focus is on the text analysis, and we can discard those fields.

The dataset contains 64.706 reviews, but we will use a smaller subset in order to reduce the computational costs of the algorithms, and to check whether using subsets of higher size can be beneficial.
Indeed, moving from a subset of 10000 entries to a subset of 40000 entries seems to positively impact the quality of the predictions. It can be assumed that using even more data could give further improvements, at the cost of greatly increased execution time of the training.

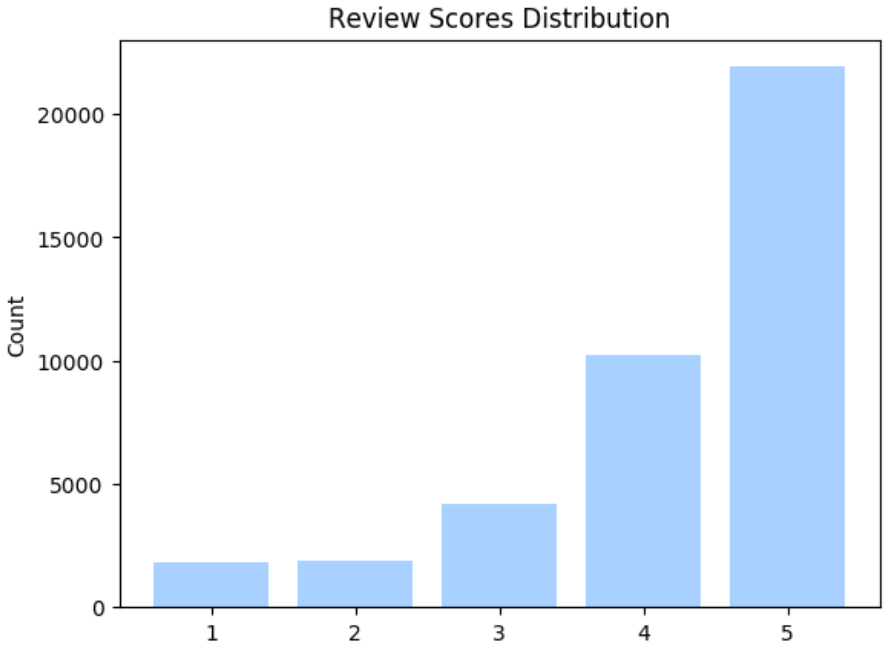


Figure 1: *Distribution of the review scores in the dataset.*

If we look at the distribution of the review scores in the dataset, it is clear how reviews with high scores are overrepresented (it's not common to buy a song one doesn't like). This has a few important implications:

- Any model we use will have to consider the *a-priori* probability of each class, as they are different.
- If our goal is to infer the sentiment of a sentence and not to predict the review score, it could be beneficial to build a new dataset in which every score has the same probability of appearing. However, this would heavily reduce the data at our disposal, and overall lead to worse results.
- Models based on regression are likely to perform badly, compared to multi-class classification models. Most regression models (such as any model based on linear regression) assume the output to be *normally* (or at least symmetrically) distributed, while we have a highly skewed distribution. Models won't be able to accurately predict the values at the extremes of the distribution. On the other hand, multi-class classification will ignore the ordinal relation in the scores, hence using less information that they could.
- A trivial predictor that always give the majority class will have an accuracy of 54% This value is our baseline, and will prove surprisingly hard to beat.

2.1 *Introduction to preprocessing*

A number of preprocessing techniques can be applied to the dataset, depending on what kind of approach we want to use to predict the scores. Different techniques have been adopted, and they are detailed below.

First, it was decided to work with a **unigram** model. For each review (from now on referred to as *document*), we can count the number of occurrences of each word in the vocabulary. This will give a **term-document** matrix, which could be used as-is, or processed further.

To build a **term-document** matrix, there are a few things to keep into account.
First, we need to treat differently words that are in a **negated context**: it is obviously different to say that something is *"good"* compared to being *"not good"*, and we have to consider that when analysing the sentences. A common approach to deal with negated contexts is to append a prefix (e.g. **NOT_**) to words that are between a negation and the following punctuation mark.
Instead, a more refined approach was taken: first, every document was analysed using the **spaCy** library for *Python*. This will give the dependency tree of each sentence. Then, one can look for arcs labelled as **neg**, which denote a negation

relationship: it is possible to append the **NOT_** prefix to any word dependent on the head of the **neg** arc; this will mark the entire subordinate clause, which is more precise than the first basic approach that was presented.

Then, it should be considered that the dataset contains a large amount of different words, and it is likely that not all of them will be useful in the prediction. Punctuation signs, stop-words, and rare words will probably not be useful, and can be removed. To remove uncommon words, it is necessary to set an arbitrary threshold.
Using a value of 20 (meaning that words with less than 20 occurrences were removed) seems to yield a good compromise between accuracy and dataset size.

Words were turned to lower-case: keeping words in upper-case could maybe be beneficial, as words written in upper-case have usually more *"strength"* associated to them (think *"bad"* vs. *"BAD"*). The same applies to certain punctuation marks such as *"!"*. Still, the increase in complexity caused by handling all these subtleties was deemed excessive for the **term-document** matrix.
It was also extracted the **lemma** of the words, instead of keeping them in their original form. Once again, we can assume that there is little difference between singular and plural nouns, or between different tenses. Overall, the final vocabulary will contain about 10000 terms.

From this processed dataset, it was built the **term-document** matrix (of size $|Documents| \times |Vocabulary|$), that lists for each document the number of occurrences of each word.
Note that instead of counting the number of occurrences, it was built a binary matrix, where a value of 1 indicates that a certain word appears in the document. The idea is that repetitions of words won't be meaningful in the classification, compared to having or not a word.
To be more precise, both approaches were tried, with the binary matrix seeming sightly more powerful.

2.2 *Latent Semantic Analysis*

Note that the **term-document** matrix is very sparse, and rather difficult to handle. Moreover, binary features don't allow to represent the semantic of a word, or its importance in the document. Hence, it was decided to process further the binary matrix, by using **Latent Semantic Analysis**.
First, the occurrences matrix (not the binary one) was used to compute the **Tf-idf** values of each word, relative to each document. This matrix, also of size $|Documents| \times |Vocabulary|$, expresses the importance of each word, relatively to each document in which it appears.

Then, **Truncated SVD** was applied to the matrix, to reduce its dimensionality and obtain a dense representation of each document. Dimensions of 100 and 500 were tried, with the second value being sightly better.

Note that both the binary matrix and the dense matrix were kept, and tried with different models. The idea is that some models might prefer to work with dense numerical data, while others will prefer the binary values.
Moreover, some models such as **SVD** proved to be impossible to train on the binary matrix, due to its size. On the other hand, the dense matrix is more manageable, and could be used to train a larger amount of different models.

2.3 *Sentiment Lexicon*

Instead of simply looking at word occurrences, one can assign scores to words based on their meaning: *"good"* and *"excellent"* have both a positive connotation, but *"excellent"* is definitely stronger than *"good"*. On the other hand, a word such as *"spaghetti"* will hardly denote a positive or negative meaning.
By computing the sentiment values of each word, it is possible to build more refined models to predict the review scores.

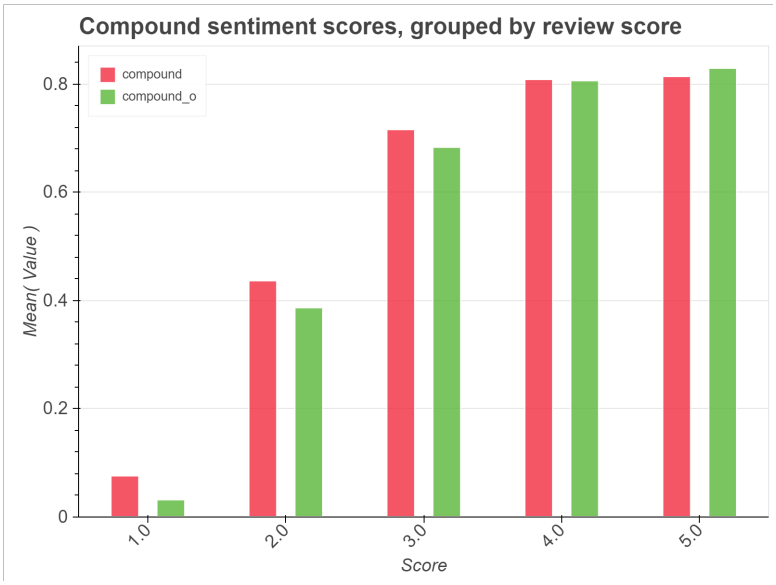
Sentiment values could be learnt from our dataset, in order to have values that are fine-tuned to the data at our disposal (for instance, a word such as *"melody"* could have a more positive meaning than *"song"*).
However, this approach is computationally intensive, and it's not guaranteed to provide good results, likely due to the small dataset at our disposal (compared to something like **WordNet**), which could lead to overfitting.
Instead, it is possible to use a pre-trained **Sentiment Lexicon**, which will significantly speed-up our work and is guaranteed to provide good results.

NLTK provides a pre-trained lexicon called **VADER** [1], which was used to score each review. For each review, it is possible to compute the **composite**, the **negative**, the **neutral** and **positive** scores. If one has to simply distinguish whether a review is positive or negative, it would be possible to simply look at the difference between positive and negative scores, and classify the document according to that.
However, if it is required to give a numerical classification, it can be beneficial to keep all 4 values, and use them as features for some model (such as a **Multiclass Logistic Regression**).

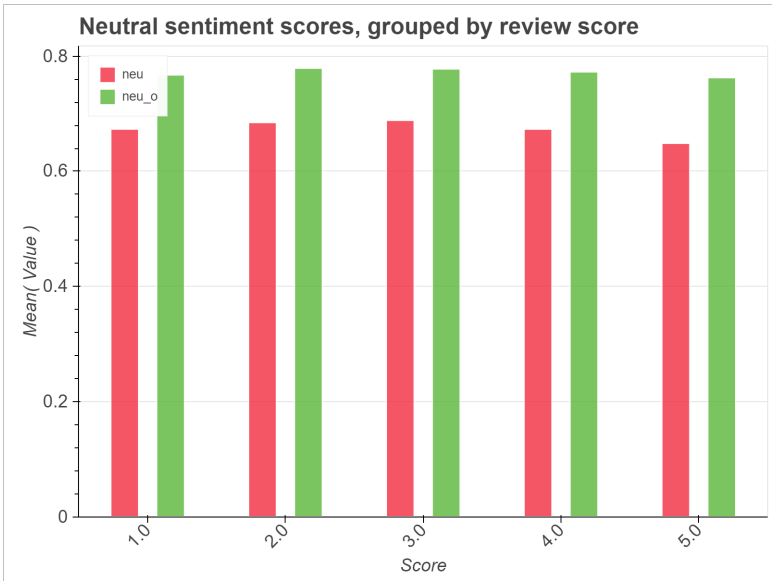
VADER doesn't just score a sentence word-by-word, but looks at the relation of words, and makes use of negated contexts, punctuation, and adjectives.
As such, it can be useful to evaluate not only the sentences that were already processed in the first section (so, at the tokenized sentences with only the lemma,

no punctuation marks, and more), but also at the original text. This will give 4 more scores for each sentence, and they can also be used as features.

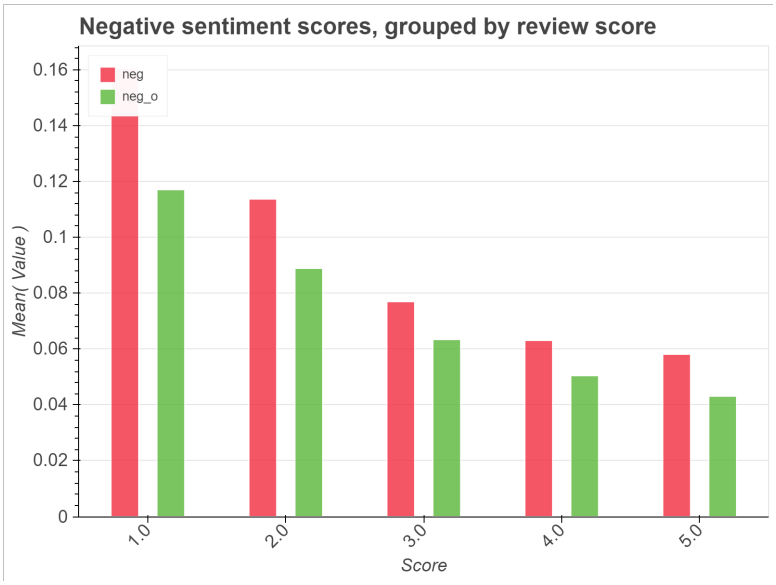
Below, the distribution of scores for each type and each review score, divided by pre-processed and original text.



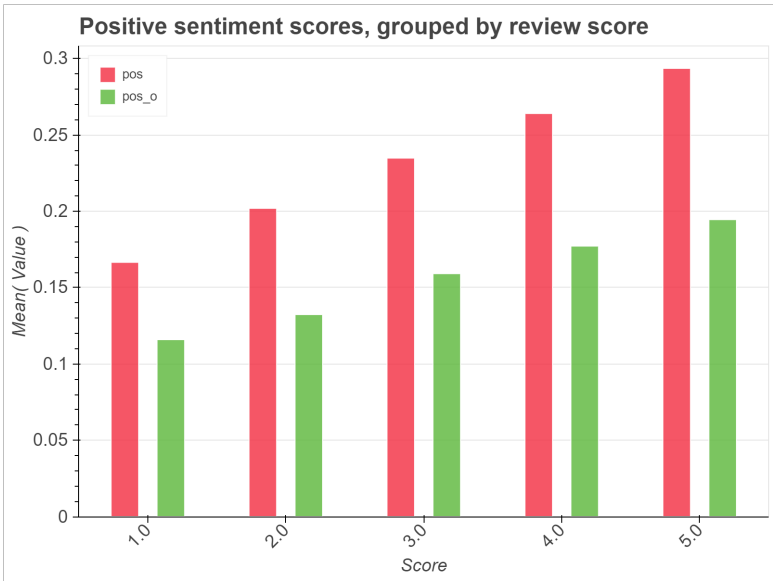
(a) Compund Score



(b) Neutral Score



(c) Negative Score



(d) Positive Score

Figure 2: Distribution of sentiment scores based on review score.

References

[1] Clayton J Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth international AAAI conference on weblogs and social media*, 2014.