NLP Assignment 3
Sentiment Analysis

Alberto Parravicini

1 Introduction

Assignment weight: 3

Full code and data available at
https://github.com/AlbertoParravicini/nlp-ulb/tree/master/assignment-3

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.

Note: instead of keeping all the code in a single file, I preferred to use multiple files: one takes care of the preprocessing, while the others contain the models used for the predictions.

- `preprocess.py` is used to pre-process the original dataset and create different embeddings/representations of the dataset, to be used in the predictions. These representations are stored as `hdf`, to save space. The size of the dataset makes them unsuitable to be added alongside this report. They are available at https://github.com/AlbertoParravicini/nlp-ulb/tree/master/assignment-3
- `naive_bayes_and_svm.py` contains the models to be used with the occurrences matrices. **Naive Bayes** is suitable for the binary matrix, while **SVM** should be used with dense vectors. Refer to the specific section of the report for more informations.
- `sent_scores.py` performs classification using the sentiment lexicon.
- `binary_class.py` does binary classification instead of multi-class classification.

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.

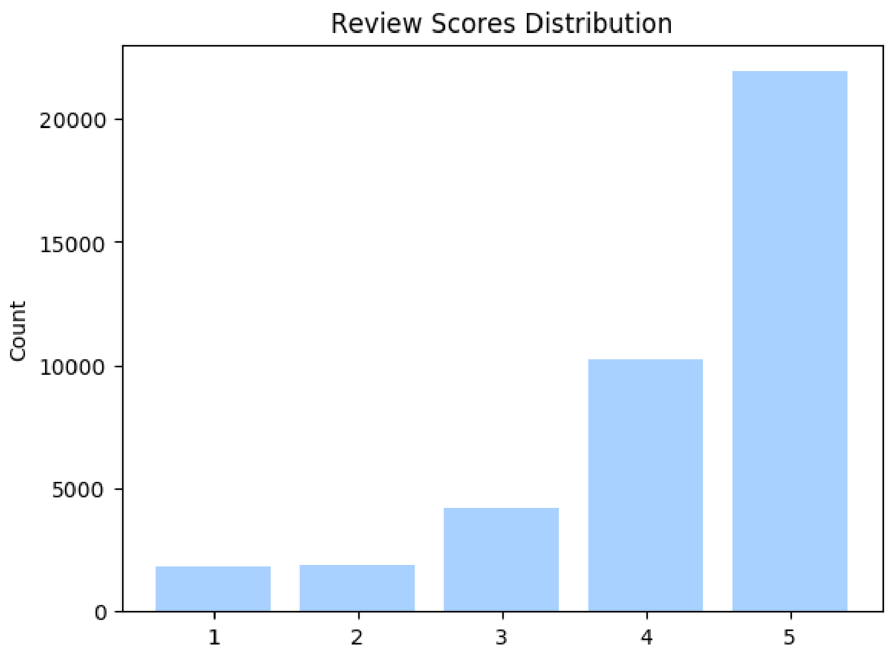


Figure 1: *Distribution of the review scores in the dataset.*

If we look at the distribution of the review scores in the dataset, it is clear how reviews with high scores are overrepresented. People usually buy songs that they like, and it's possible to see how most negative reviews contains complaints that aren't about the song itself, but about the sound quality or mastering (*"I love this song, but the sound quality is awful!"* is a very common pattern). 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.

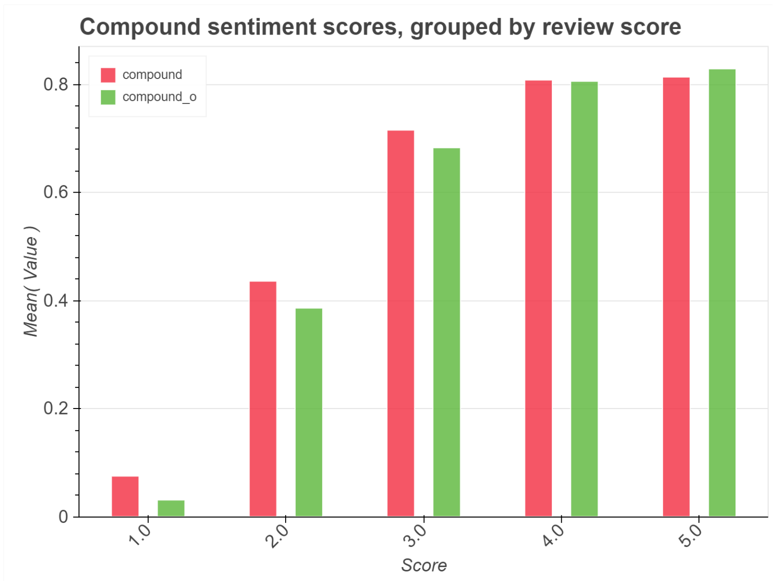
Sentiment values could be learnt from our dataset, in order to have values that are fine-tuned to the data at our disposal (for instance, a word such as *"melody"* could have a more positive meaning than *"song"*).
However, this approach is computationally intensive, and it's not guaranteed to provide 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**).

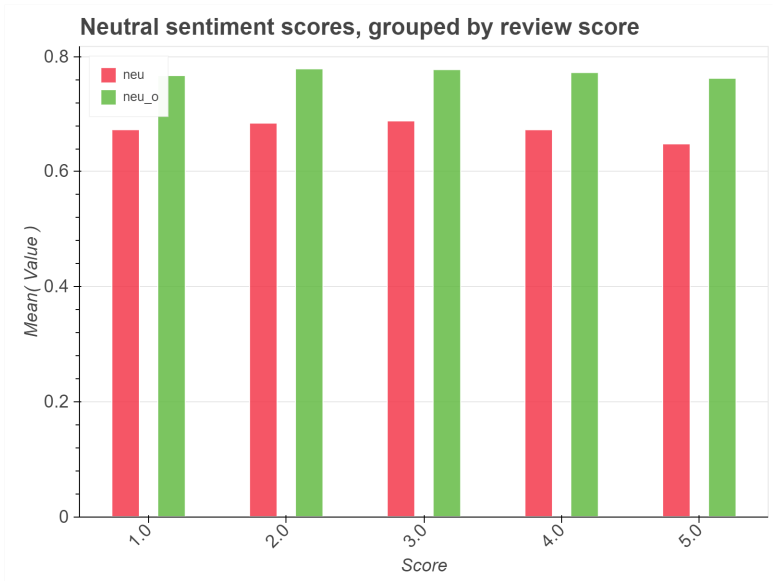
VADER doesn't just score a sentence word-by-word, but looks at the relation of words, and makes use of negated contexts, punctuation, and adjectives.

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.

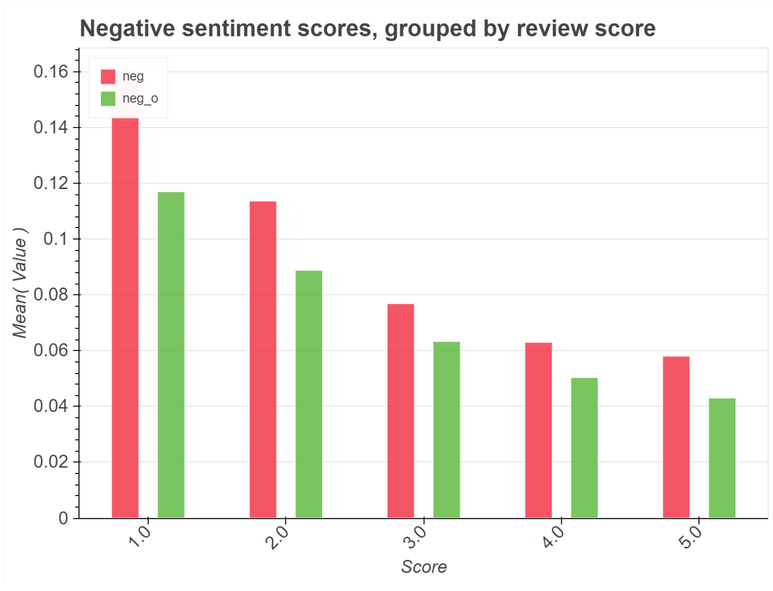
Below, the distribution of scores for each type and each review score, divided by pre-processed and original text.



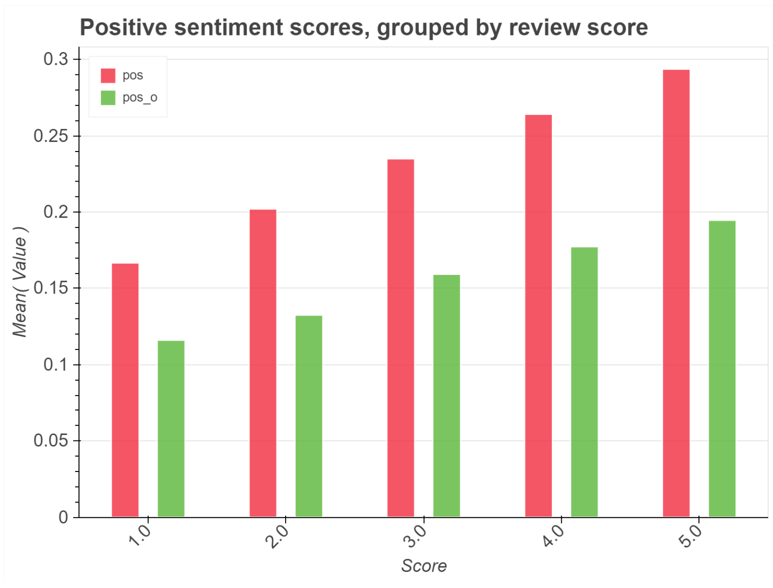
(a) Compund Score



(b) Neutral Score



(c) Negative Score



(d) Positive Score

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

It can be noted how the sentiment scores reflect what we would expect: **negative** scores are associated to **negative** reviews, while **positive** scores are associated to **positive** reviews. **Neutral** scores show a very slight bump on the **average** scores, which is more noticeable in the pre-processed text.
In general, the distributions of pre-processed and unprocessed reviews are similar, but values are stronger in the pre-processed text, as most of the superfluous information was taken out.

3 Model analysis

After preprocessing the dataset, it is possible to train different models and see how they perform. As we have different representations of the same dataset (sparse matrix, dense matrix, and sentiment scores), we will also use different models that are best suited to work with each type of data.

20% of the dataset is used for validation; this set is built with *stratified sampling*, to preserve the score distribution.
The remaining 80% is used for training. Training is done through 10-*fold cross-validation*, to obtain a precise estimate of the model accuracy and find the best values of the hyper-parameters.
Then, the model is trained with the entire training set, and this model is tested on the validation set. This approach is used for every model that has been considered.

3.1 *Multinomial Naive Bayes*

The first model to be considered is a **Multinomial Naive Bayes**. This model was applied to the sparse **term-document** matrix. In this model features are supposed to have a multinomial distribution, which can be assumed as entries of the input matrix are word counts.
Interestingly, using the binary occurrence matrix with a **Multinomial Naive Bayes** gives results that are better than both using the same model on the **term-document** matrix and of using a **Binomial Naive Bayes** on the binary occurrence matrix.

In the Naive Bayes model it is possible to set the value used for *Laplace smoothing*: different values were tried and tested through cross-validation, and the best result seems to be around 3.
Overall, the accuracy of the model is about 59%, just 5% above the baseline. Still, this is the best model that was obtained. For a more in-depth discussion of the result, refer to the specific section.

The dense matrix obtained through **LSA** was used as input of an **SVM** classifier. Training the **SVM** on the sparse matrices proved computationally infeasible, while the training is rather fast on the dense matrix.
Still, the accuracy of this model is mediocre, at around 56%. However, using the dense matrix as input of the **Naive Bayes** gives even worse results (worse than the baseline): this could indicate that using a larger amount of dimensions, or even the sparse matrix, as input of the SVM could result in improvement on the best result obtained.

The dataset obtained from the sentiment lexicon was tested with different models, both for regression and classification (in the case of regression, prediction are rounded to the nearest integer): **Random Forest** regressors and classifiers, **Generalized Linear Models**, **Gaussian Naive Bayes**.
Overall, the best model seems to be a simple **Multiclass Logistic Regression**: even in this case, the results are mediocre, giving an accuracy of 56%.

This last result is rather surprising, considering that there is a strong correlation between the sentiment scores and the review scores, as shown in the previous plots.

4 Result Analysis

This section will focus on the analysis of the results obtained by the models. Specific attention will be given to the **Multinomial Bayes Model**, as it is the one that performed the best.

First, it is possible to see how many values of each type were predicted, versus the real ones.

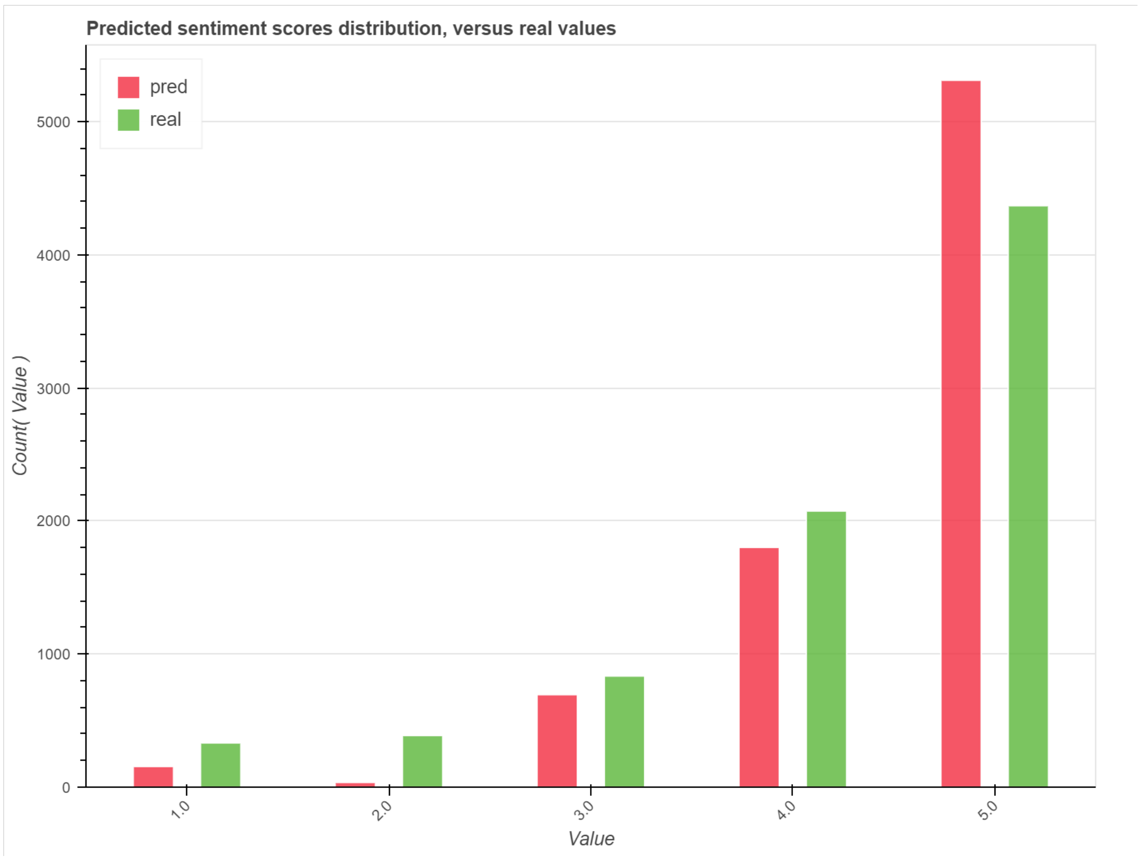


Figure 3: *Distribution of the predicted review scores, versus the real ones.*

It seems that the classifier overestimates the number of reviews with score 5, at the expenses of negatively rated reviews. This could be caused by the disproportion of classes in the dataset, a further evidence that having a more balanced training set could give better results.
Still, increasing the size of the input dataset shows improvements: the **Multiclass Naive Bayes** goes from an accuracy of 56% on a dataset with 10000 reviews to 59% if the dataset contains 40000 reviews. With enough data, and computational power, it is likely that better results are achievable.
In any case, the results aren't very encouraging: if the bad results obtained by regression can be explained by the skewed distribution of the scores, the classifiers should perform much better.

Using bigram counts would probably be very beneficial, but it is computationally infeasible given the available resources. It might be possible if the vocabulary was filtered further; however in this case the bigram would lose significance, as they would count words that aren't really close to each other.
Also, consider that with a vocabulary of just 1000 words (down from the current 1000), we would end up with 1000000 possible bigrams. The real count would obviously be much lower, but still very hard to manage.

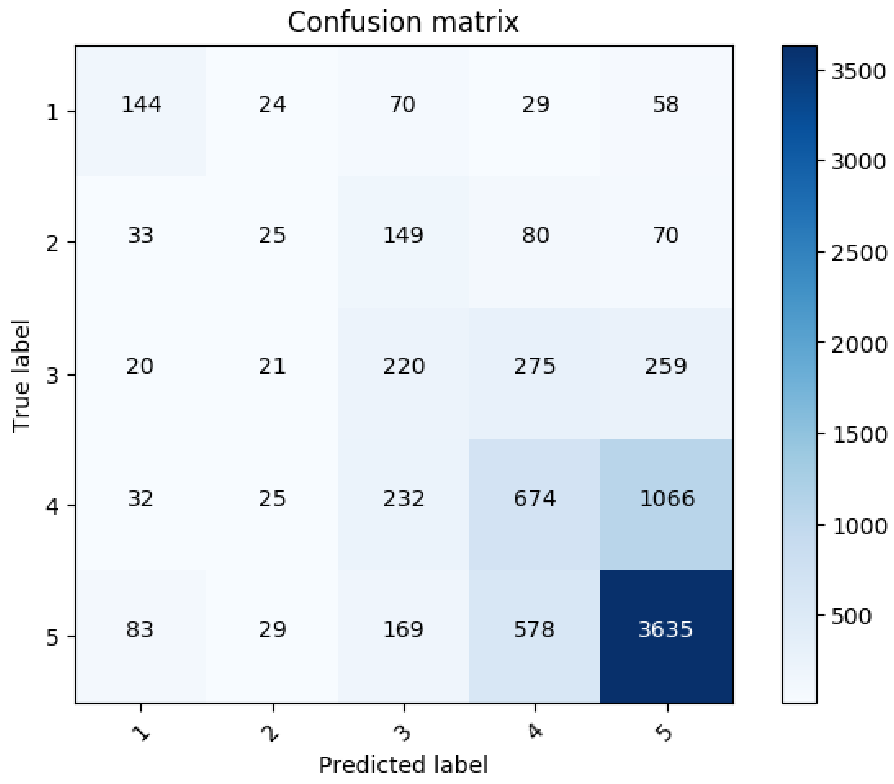


Figure 4: *Confusion matrix of the* ***Multiclass Naive Bayes*** *classifier.*

Most of the mistakes are done by predicting scores of value 4 as having value 5. Interestingly, there are also many mistakes in the opposite sense, which makes

hard to say that our classifier is being too *"generous"* with the scores it's giving.
There is also a certain tendency to assing 3 instead of 2: indeed, most reviews with score 2 are actually classified as 3.
Amusingly, there is a large set of reviews with score 5 that are classified as 1, and viceversa. We will look below at some of these mistakes.

If we pick the majority class, 5, as reference point, it is also possible to compute other useful metrics.

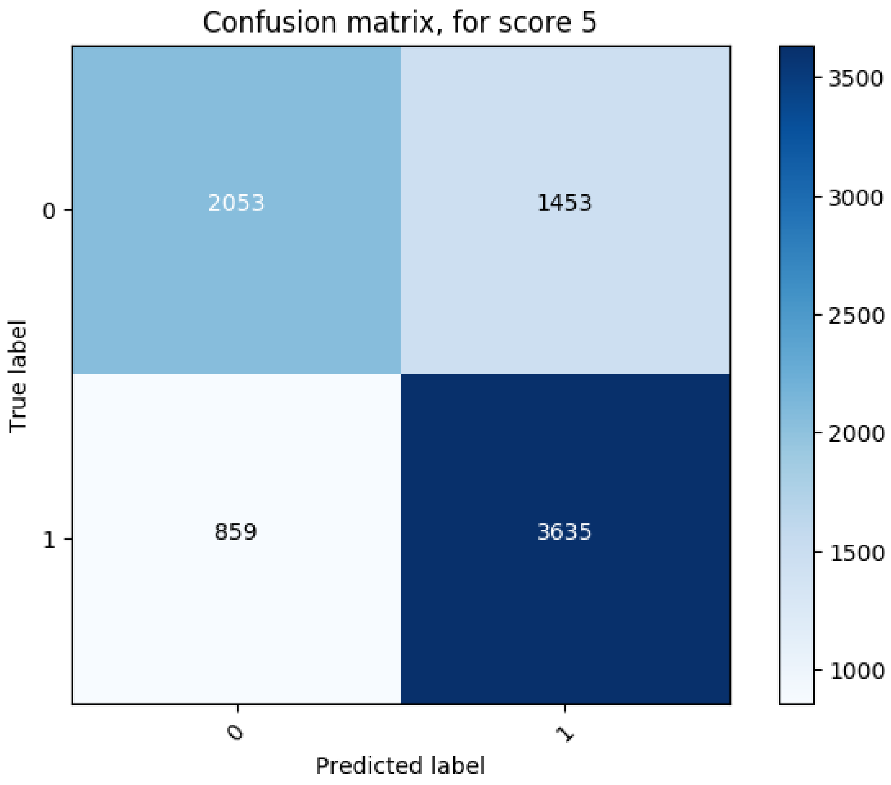


Figure 5: *Confusion matrix of the* ***Multiclass Naive Bayes*** *classifier, for score* 5.

We see once again how the number of predicted 5 is much higher than the real amount. We can also measure:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} = 0.68$$
$$balanced\ accuracy = \frac{1}{2}(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}) = 0.65$$
$$sensitivity = \frac{TP}{TP + FN} = 0.8$$
$$specificity = \frac{TN}{TN + FP} = 0.5$$

$$precision = \frac{TP}{TP + FP} = 0.71$$

1. **Balanced Accuracy** measures the accuracy taking into account the different class distributions, and has value 0.5 in case the majority class is predicted. Here we see that the balanced accuracy is only sightly lower than the accuracy, which means that we are actually able to predict the class 5, instead of just predicting the majority class.
2. **Sensitivity** measures how many of the reviews with value 5 were actually found. We can be satisfied with a value of 0.8.
3. **Specificity** measures the percentage of reviews with value lower than 5 that were actually found: here the value is not very satisfactory, as it was seen how the classifier is overestimating the amount of 5.
4. **Precision** shows how many of the reviews that we classified as 5 are actually 5. Indeed, the value is not very high, as we predicted too many reviews to have value 5.

4.1 *Misclassification Analysis*

Below are reported snippets of 10 misclassified examples. 5 were predicted to have score 5, but had a real score of 1. The other 5 are the opposite.

- "[...] Mind you, Commin from Where I'm From is weak in comparison to some of my favorites on the CD like Lucille, Float, Charlene, and Mama Knew Love. This Brother testifies on this CD and it is worth every dime. Buy it and prep yourself for an experience."
- "[...] It's also not the first thing I'd buy in the store.While I can appreciate the effort, this record will largely appeal only to people for whom old Motown-type tracks from the 60s still hold sway. [...]".
- "[...] A lot of the edge is taken out, some of the saxophone solo isn't there, and it loses some of the darkness of the earlier version.""It Goes On,"" ""Dumb Waiters"", and ""Into You Like a Train"" have riffs boppy enough to dance to. [...]".
- "[...] I have a problem with reviewers who probably have all of their favorite artist(s)' albums and go on record complaining that this song or that song wasn't on the ""greatest hits"" or ""best-of"" collection. I was a fan of Eddie Money's from the beginning, and was a little surprised to find that one reviewer thought his debut album was lame. [...]".

- "[. . .] Track Listing. Side One1. Rock and Roll, Hoochie Koo2. Joy Ride [Instrumental]3. Teenage Queen4. Cheap Tequila5. Uncomplicated6. Hold-Side Two1. Airport Giveth (The Airport Taketh Away)2. Teenage Love Affair3. It's Raining4. Time Warp [Instrumental]5. Slide on Over Slinky6. Jump, Jump, Jump,".

We see how in many cases there are problems due to minor complains that might be considered more important than they are. The last example is harder to interpret, as it's just a sequence of titles. Indeed, it can be seen that there are many reviews which don't really contain any coherent text, and it's not surprising that they are misclassified.

- "A Great Album Deserves Better Sound. [. . .] But this album deserves to be heard with modern mastering. Anyone who cares about good sound, save your money, listen to your vinyl."
- "I Had No Idea. I kind of liked some of the songs these guys had on the radio. The one about Africa and Roseanna were cool songs, so I gave this a listen. [. . .]."
- "Buy the SACD or the Blu-Ray. You want a ""deluxe"" edition? Go for superior quality and buy the SACD edition. It's incredible. CDs are low-fi crap no matter what you add to the song list."
- "Original master tape. Audio Fidelity does not use the Original master tapes on this and about all of the CD releases. They claim the sources they use are going to give the consumer a great sounding CD.If you're not using the Original master tape then you're comparing HDTV to analog TV.This version sounds very slightly better then the Motown release but compare the cost.I wish Mobile Fidleity would release this because they use nothing but the Original Master tape and the sound comes through.A great Stevie Wonder CD, but all of his CD's are great!!!!!!!!!!!!!!!!!"
- "Not his best, but a very attractive. Almost any CD from this gifted singer, is a delight for your ears! ""L Is For Lover"", not being one of his best, is certainly an excellent addition to your Male Jazz/Pop singers collection. I don't know exactly why there is now a ""shortage"" of Al Jarreau's CDs in the market (perhaps it is due to that usually stupid fight between record companies for copyrights &/or royalties), but if you have the chance to get it, do it right now! (I had to buy mine as a used one -a German issue, the sound quality is outstanding!)."

In the case of negative reviews classified as positive, it can be seen how words with positive connotation are often used to describe other songs or artist, but are

used nonetheless to mark the review as positive.
The last 2 cases seem to be human error, as the reviews contain a positive description, followed by an extremely negative 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.