# Amazon Reviews for Sentiment Analysis

Ioana Ciripan (407), Dana Dăscălescu (407), Bogdan Noană (407),
Simona Pop (410), Mihnea Smarandache (407)

January 15, 2022

**Abstract**

Automatic identification of sentiment about an event, topic, or product in a text is an important area of research in natural language processing, especially given the ever-increasing amount of data provided by users, which makes this task very difficult or even impossible for human annotators. This project explores simple and efficient base models, implementing and benchmarking several sentiment analysis models, fine-tuning large Transformers networks that use the mechanism of self-attention for text classification, different pre-processing stages, and feature extraction methods to improve experimental results.

## Contents

# 1 Introduction, Motivation and Objectives

## 1.1 Introduction

In this project, we are exploring ways to analyze the dataset, pre-process the data and extract features to scale baselines to a large corpus in the context of text classification. Our focus is on extracting good features for ensembles methods and mechanisms of attention-based models.

## 1.2 Task and composition of the dataset

The Sentiment Analysis task, also known as Opinion Mining, aims to identify, extract and study the emotional states and the opinion of the general public on a particular topic. The range of practical applications is vast, from monitoring popular events (election campaigns, Olympic Games, etc.) to the correct segmentation of the target audience and understanding the client's voices, which is also the principal motivation for choosing the theme for this project.

For this task, we choose the **Amazon Reviews Full Score** dataset constructed by Xiang Zhang from the **Amazon Review** dataset and used for a series of text classification benchmarks like in Xiang Zhang's paper[6]. Originally, it was used for a five-point scale classification task, but later it became popular for the three-point scale classification (positive, neutral or negative sentiment towards the review) and for two-point scale classification (positive and negative sentiment). We tried out both tasks and for this purpose, we have converted the rating column into the corresponding sentiment labels, i.e. ratings less than or equal to 2 were given a negative label, ratings equal to 3 were given a neutral label, and ratings greater than or equal to 4 were given a positive label.

The Amazon Reviews dataset consists of approximately 35 million product reviews from Amazon. The data includes product and user information, ratings, and a plaintext review, which spans a period of 18 years, up to March 2013.

For each review score from 1 to 5, 600,000 training samples and 130,000 testing samples are randomly selected to create the Amazon reviews full score dataset. A total of 3,000,000 training samples and 650,000 testing samples are available.

All of the training samples are stored in the files ***train.csv*** and ***test.csv*** as comma-separated values. They include three columns, one for each class label (a score between 1 and 5), one for the review title, and one for the review text. Double quotes (") are used to escape the review title and text, and any internal double quotes are separated by two double quotes (""). A backslash followed by a "n" character is used to escape newlines.

Due to computational and memory available resources, we extracted a subset from this dataset: 80000 samples for training and 20000 samples for inference.

## 1.3 Overview and organization

The rest of the documentation is organized as follows: Section 2 provides an overview of some theoretical concepts of machine learning used in the proposed models, Section 3 describes in detail the investigations performed on the data, and it offers a summarization of the main characteristics, discovered patterns, and spotted anomalies, Section 4 presents in detail the proposed approaches, implementation and experiments performed, describes the data set and the evaluation protocol of the models, and the last section is a summary of the information presented and the conclusions drawn.

# 2 Theoretical concepts

## 2.1 Word Embeddings

Hidden layers in a deep neural network encode a distributed representation of the input layer. Distributed word representations are called ”word embeddings”. They are usually the hidden units at the top of deep learning architecture, driven by an immense amount of data in which words with the same meaning have a related representation.

A single word is represented by a vector of real values in a predefined space by encoding its properties. The key to this approach is the use of a densely distributed representation for each word. This approach reduces the size to tens or hundreds of units, as opposed to ”one-hot” encoding, where the dimensions go into the thousands or even millions when we are dealing with a large corpus.

Over time, various approaches have become known. These include encoding syntactic/semantic similarity, encoding correlation between words (Glove), encoding contextual information (Word2Vec), or dynamic contextual information(BERT). The latter approache are used in the proposed approaches as they aim to capture the semantics of words in different contexts to address the problem of polysemantic, and in the case of dynamic representation, the problem of context-dependency of words is addressed.

## 2.2 Transformers

In short, the task of the encoder in the left half of the Transformer architecture is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder (figure 1).

The decoder in the right half of the architecture receives the output of the encoder along with the output of the decoder in the previous time step to produce an output sequence.

At each step, the model consumes the symbols previously generated by the decoder as additional input when generating the next sequence.

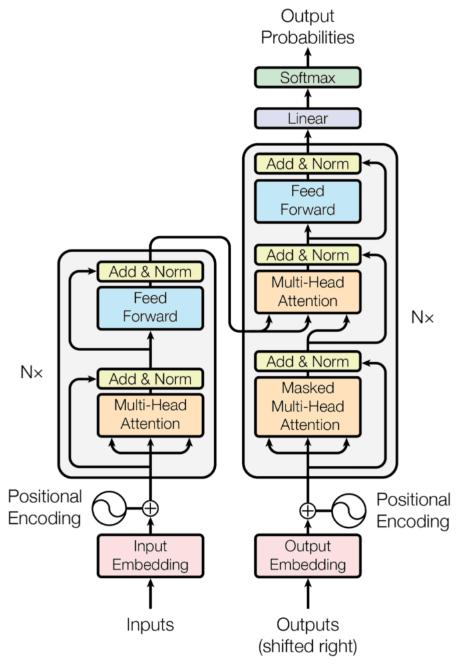


Figure 1: The Encoder-Decoder Structure of the Transformer Architecture [5]

The encoder part consists of a stack of N = 6 identical layers, where each layer is composed of two sublayers:

1. The first sublayer implements a multi-head self-attention mechanism consisting of the implementation of h heads, each of which receives a different linearly projected version of the queries, keys, and values to produce h outputs in parallel, which are then used to produce a final result.

2. The second sublayer is a fully connected feed-forward network, consisting of two linear transformations with Rectified Linear Unit(ReLU) activations in between:

$$FFN(x) = ReLU(W_1x + b_1)W_2 + b_2$$

Each sublayer is also followed by a normalisation layer, Norm(.), which normalises the sum between the input of the sublayer, x, and the output produced by the sublayer itself, sublayer(x):

$$Norm(x + sublayer(x))$$

In addition, each of these two sub-layers has a residual connection. All six layers of the transformer encoder apply the same linear transformations to all tokens in the input sequence, but each layer uses different weights and bias parameters.

An important consideration is that the transformer architecture cannot inherently capture information about the relative positions of words in the sequence because it does not use recursion. This information must be introduced by introducing positional encodings into the input embeddings.

The position encoding vectors have the same dimension as the input embeddings and are generated using sine and cosine functions with different frequencies. They are then simply added to the input embeddings to insert the position information

Analogous, the decoder consists of a stack of N = 6 identical layers similar to the encoder ones:

1. The first sub-layer receives the previous output of the decoder stack, augments it with position information and implements multi-head self-attention over it. While the encoder is designed to pay attention to all words in the input sequence, regardless of their position in the sequence, the decoder is modified to pay attention only to the preceding words. Therefore, the prediction for a word at that position can only depend on the known outputs for the words that come before that word in the sequence. In the multi-head attention mechanism (which implements multiple individual attention functions in parallel), this is achieved by introducing a mask for the values produced by the scaled multiplication of the matrices.

2. The second layer implements a multi-head self-attention mechanism similar to that implemented in the first sub-layer of the encoder. On the decoder side, this multi-head mechanism receives the queries from the previous decoder sub-layer and the keys and values from the output of the encoder. This allows the decoder to process all the words in the input sequence.

3. The third layer implements a fully connected feed-forward network similar to the network implemented in the second sub-layer of the encoder.

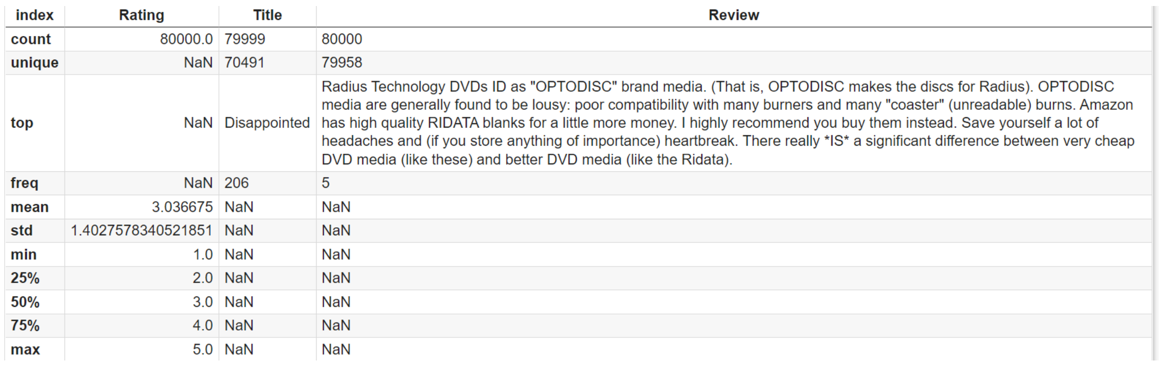
# 3 Exploratory Data Analysis

## 3.1 Descriptive statistics

Before building the models, it was necessary to perform an analysis of the dataset, steps for text cleaning and pre-processing the data.

Firstly, we generate descriptive statistics about our dataset without performing any pre-processing step. We wanted to summarize the central tendency, dispersion, shape of the dataset's distribution, and check for missing values (figures 2 and 3).

We observed that we have missing values, and since reviews and ratings are the main information in this dataset, we eliminated the rows with missing reviews or ratings. Missing titles were replaced with the text "Missing title".



| index | Rating | Title | Review |
|---|---|---|---|
| count | 80000.0 | 79999 | 80000 |
| unique | NaN | 70491 | 79958 |
| top | NaN | Disappointed | Radius Technology DVDs ID as "OPTODISC" brand media. (That is, OPTODISC makes the discs for Radius). OPTODISC media are generally found to be lousy: poor compatibility with many burners and many "coaster" (unreadable) burns. Amazon has high quality RIDATA blanks for a little more money. I highly recommend you buy them instead. Save yourself a lot of headaches and (if you store anything of importance) heartbreak. There really *IS* a significant difference between very cheap DVD media (like these) and better DVD media (like the Ridata). |
| freq | NaN | 206 | 5 |
| mean | 3.036675 | NaN | NaN |
| std | 1.4027578340521851 | NaN | NaN |
| min | 1.0 | NaN | NaN |
| 25% | 2.0 | NaN | NaN |
| 50% | 3.0 | NaN | NaN |
| 75% | 4.0 | NaN | NaN |
| max | 5.0 | NaN | NaN |

Figure 2: Descriptive statistics of training samples

As it can be observed in the tables, there are duplicates in the Review column in both sets that can negatively influence the outcome of the prediction, so we choose to remove them.

After performing this step, the datasets contains:

- For binary classification:
  Number of train samples: 63448
  Number of test samples: 15904

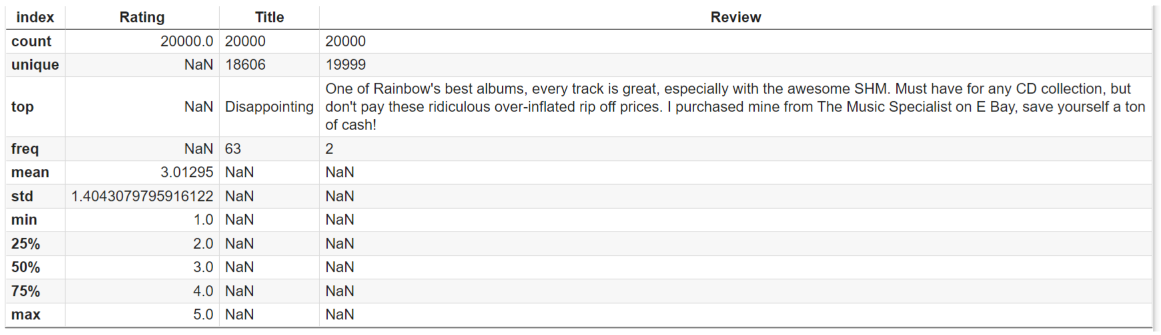
- For multiclass classifictaion:
  Number of train samples: 79957
  Number of test samples: 1998

For a better analysis of the data, it is necessary to perform the following pre-processing steps:

1. The text is decoded and then normalized, i.e., the data is transformed from complex symbols into simple characters. Characters can be subjected to different forms of coding, such as "Latin", "ISO / IEC 8859-1" etc. Therefore, for better analysis, it is necessary to keep the complete data in a standard encoding format. We choose "UTF-8" encoding for this task because it is widely accepted and often recommended.

2. We replace emoticons and emojis with the corresponding descriptive words.

3. Any letter repeated more than three times in a row is replaced by two repetitions of the same letter as the usual rules of English spelling forbid triple letters (for example "cooool" is replaced by "cool").



| index | Rating | Title | Review |
|---|---|---|---|
| count | 20000.0 | 20000 | 20000 |
| unique | NaN | 18606 | 19999 |
| top | NaN | Disappointing | One of Rainbow's best albums, every track is great, especially with the awesome SHM. Must have for any CD collection, but don't pay these ridiculous over-inflated rip off prices. I purchased mine from The Music Specialist on E Bay, save yourself a ton of cash! |
| freq | NaN | 63 | 2 |
| mean | 3.01295 | NaN | NaN |
| std | 1.4043079795916122 | NaN | NaN |
| min | 1.0 | NaN | NaN |
| 25% | 2.0 | NaN | NaN |
| 50% | 3.0 | NaN | NaN |
| 75% | 4.0 | NaN | NaN |
| max | 5.0 | NaN | NaN |

Figure 3: Descriptive statistics of test samples

4. The URL addresses are replaced by the URL token. Hyperlinks are removed too.

5. We eliminate all email addresses.

### 3.2 Exploring relevant features in the data

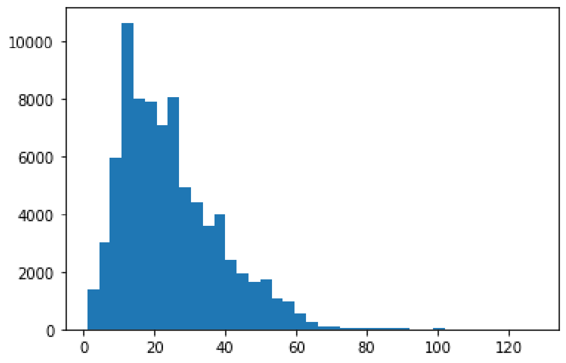
We will create some additional features that provide relevant information about the composition of our data. In particular, we will analyze the distribution of the textual characteristics regarding the user's review to identify the main insights on the data distribution and find possible outliers.

This step is necessary due to its impact on the final results of the model since it will tell us what type of transformation we need to apply to our data, which features need to be extracted, or which hyperparameters to choose.

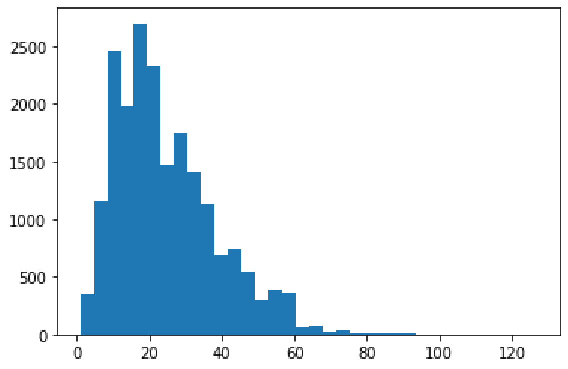
A histogram gives an immediate impression of the values, relative frequency, and spread associated with the data. It is obtained by dividing the ordinate of the histogram by the total number of observations, in this case, 79.957 for the training data set and 19.998 for the test data set, and the width of the $\Delta$ range. It is essential to choose the number of intervals according to the information that wants to be extracted from the mathematical model. The number of intervals is defined as follows:

$$\Delta = 1 + 3,3 * ln(n)$$ [1]

where $n$ is the number of observations.



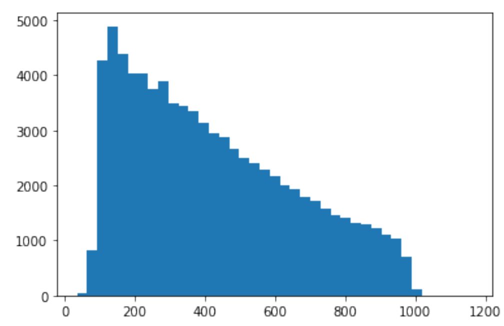
(a) Number of characters of each title in train data



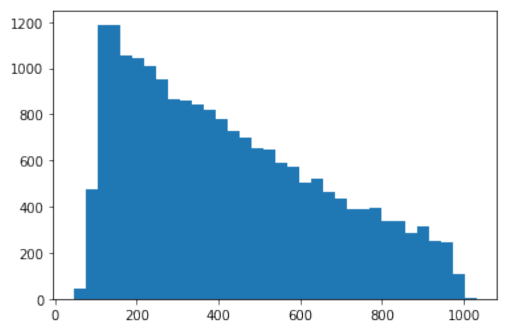
(b) Number of characters of each title in test data

Figure 4: Histograms of characters present in Reviews' titles

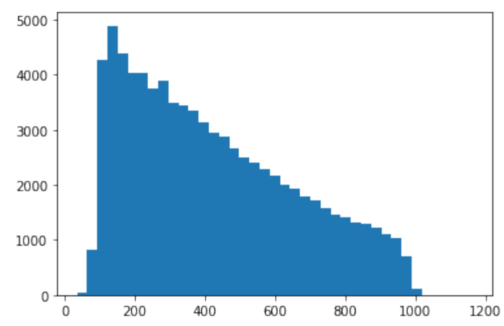
[1]Surges suggested it as a practical model in 1926.



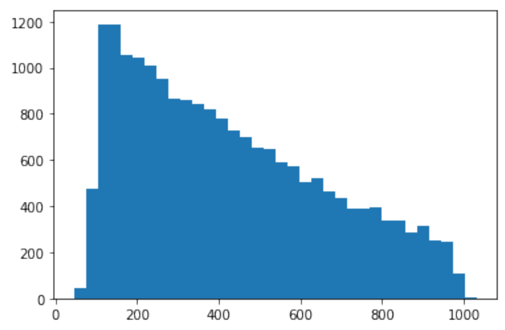
(a) Number of characters of each review in train data



(b) Number of characters of each review in test data



(c) Number of non-special characters (except for '.!?) of each review in train data

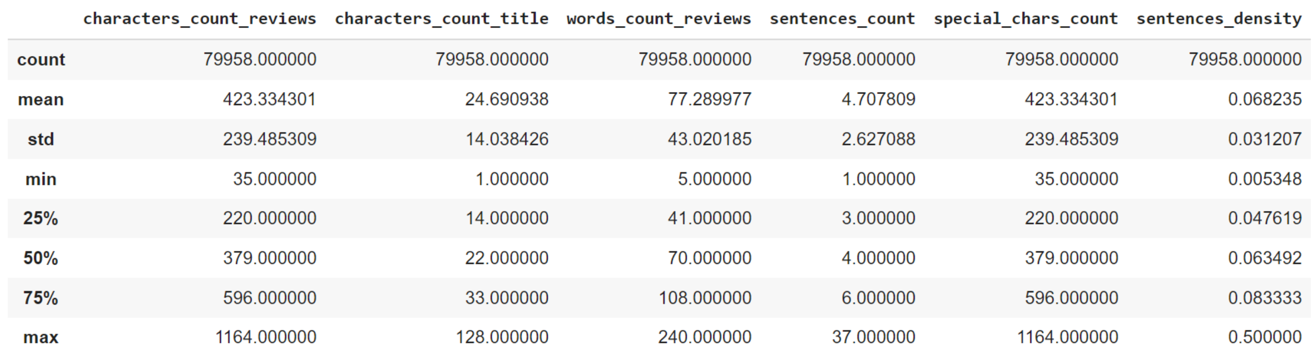


(d) Number of non-special characters (except for '.!?) of each review in test data

Figure 5: Histograms of characters present in Reviews

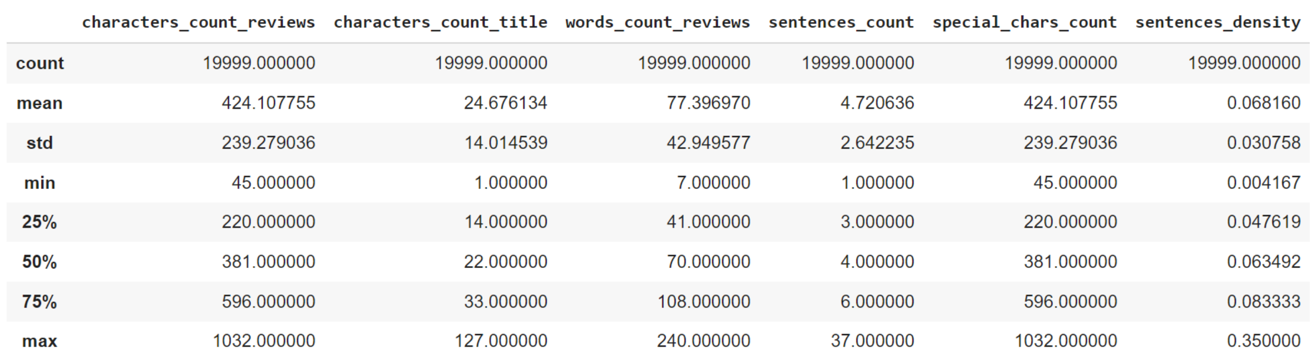
The distribution is approximate in the form of a bell (figures 4a and 4b), which is the shape of the normal distribution. Unlike a Gaussian distribution, this distribution is asymmetric; it has a tail that extends more to the right than to the left, but the differences are not large and can be ignored.

The histogram indicates a right-skewed data set in terms of its characters. This is due to a lower bound, so we will consider normalizing our data before feeding it to our models.



| | characters_count_reviews | characters_count_title | words_count_reviews | sentences_count | special_chars_count | sentences_density |
|---|---|---|---|---|---|---|
| **count** | 79958.000000 | 79958.000000 | 79958.000000 | 79958.000000 | 79958.000000 | 79958.000000 |
| **mean** | 423.334301 | 24.690938 | 77.289977 | 4.707809 | 423.334301 | 0.068235 |
| **std** | 239.485309 | 14.038426 | 43.020185 | 2.627088 | 239.485309 | 0.031207 |
| **min** | 35.000000 | 1.000000 | 5.000000 | 1.000000 | 35.000000 | 0.005348 |
| **25%** | 220.000000 | 14.000000 | 41.000000 | 3.000000 | 220.000000 | 0.047619 |
| **50%** | 379.000000 | 22.000000 | 70.000000 | 4.000000 | 379.000000 | 0.063492 |
| **75%** | 596.000000 | 33.000000 | 108.000000 | 6.000000 | 596.000000 | 0.083333 |
| **max** | 1164.000000 | 128.000000 | 240.000000 | 37.000000 | 1164.000000 | 0.500000 |

Figure 6: Statistical count features for train samples



| | characters_count_reviews | characters_count_title | words_count_reviews | sentences_count | special_chars_count | sentences_density |
|---|---|---|---|---|---|---|
| count | 19999.000000 | 19999.000000 | 19999.000000 | 19999.000000 | 19999.000000 | 19999.000000 |
| mean | 424.107755 | 24.676134 | 77.396970 | 4.720636 | 424.107755 | 0.068160 |
| std | 239.279036 | 14.014539 | 42.949577 | 2.642235 | 239.279036 | 0.030758 |
| min | 45.000000 | 1.000000 | 7.000000 | 1.000000 | 45.000000 | 0.004167 |
| 25% | 220.000000 | 14.000000 | 41.000000 | 3.000000 | 220.000000 | 0.047619 |
| 50% | 381.000000 | 22.000000 | 70.000000 | 4.000000 | 381.000000 | 0.063492 |
| 75% | 596.000000 | 33.000000 | 108.000000 | 6.000000 | 596.000000 | 0.083333 |
| max | 1032.000000 | 127.000000 | 240.000000 | 37.000000 | 1032.000000 | 0.350000 |

Figure 7: Statistical count features for test samples

### 3.2.1 Statistical count features

Statistical count features that we are going to explore:

- Total number of characters in text excluding spaces and numbers.
- Total number of words.
- Total number of sentences in the reviews.
- Total number of special (punctuation and numerical) characters.
- Total number of stopwords in the user's reviews.
- Number of sentences relative to the total number of words.

The following conclusions can be drawn after viewing the statistical data (figures 6 and 7):

1. Standard deviations for all the features except for the *number of sentences* are large.
2. Statistically, both data sets, train and test, have similar characteristics.
3. Reviews contain on average 5 sentences, 78 words, and 423 characters, which is quite close to the median. But the maximum number of all three characteristics are far from the median (379 characters, 70 words and 4 sentences) that indicates we may deal with some outliers. For example, the maximum number of sentences found in a Review is 37. For now, we will not remove these samples and we will make a decision after a future more detailed analysis.
4. Even for the binary classification task (where we ignore samples containing ratings equal to 3) the mean and median remain approximately equal, which means that the statistical data remained valid for both tasks.

We also wanted to visualize the boxplots (figure 11) for some features. Boxplots are a standardised method of displaying the distribution of data based on a summary of five numbers (minimum value, first (lower) quartile (Q1), median, third (upper) quartile (Q3), and maximum value).

The main information is contained in the words, so we turned our attention to the total number of words in each review. We identified a small amount of outliers (one in the train data and two in the test data).

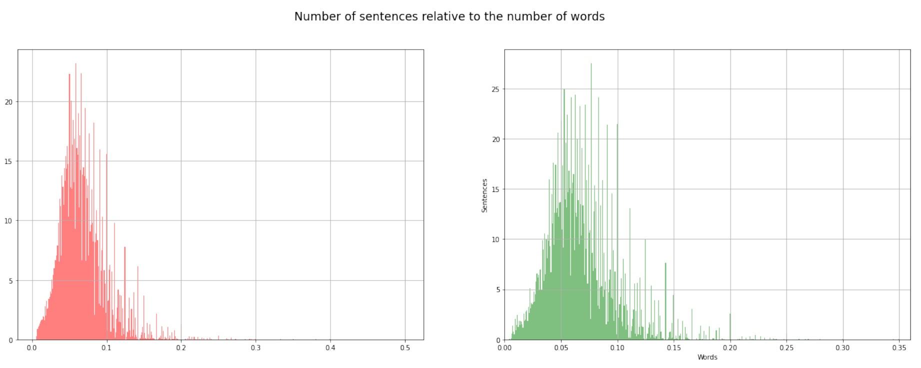
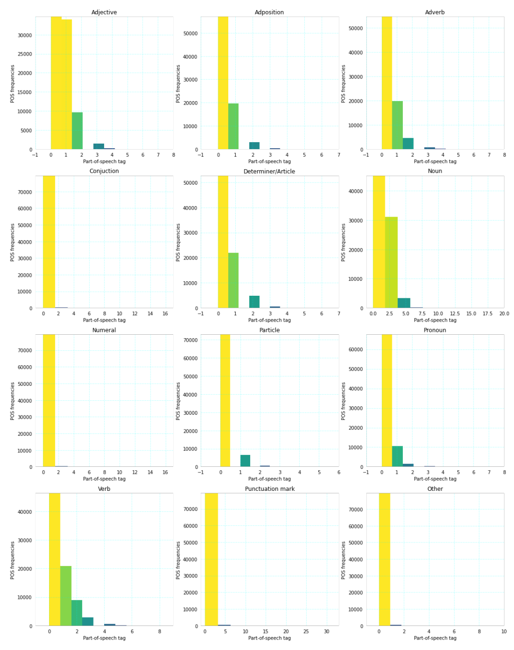


Figure 8: Number of sentences relative to the number of words in the reviews

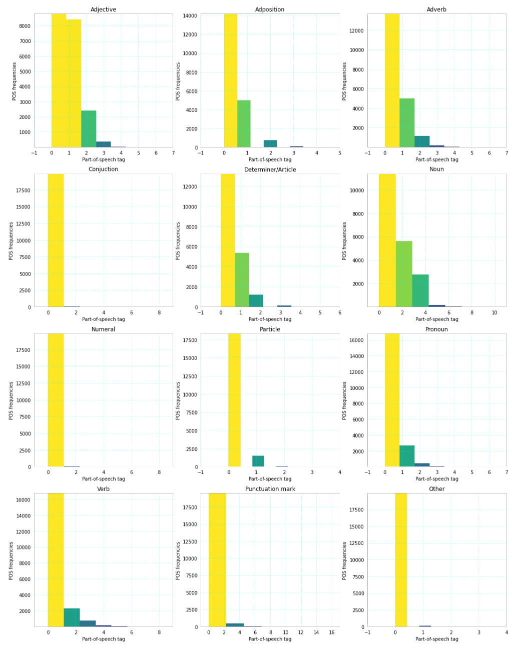
The *sentences density* (figure 8) is left-skewed with 75% smaller or equal with 0.08333333333333333 and the maximum value equal to 0.350000 for the training examples. The visualization of these data made it possible to identify some strange examples (samples that contain a small number of words in the review which will make it difficult to identify the overall sentiment or sentences which contain only one word).

### 3.2.2 Analyze features related to part-of-speech (POS-tagging)

Another group of features we can inspect in our text data is part-of-speech tagging, also known as word classes or lexical categories. Part-of-speech tagging[2] consists in assigning to each token (word) in a text corpus the part of speech and often other grammatical categories such as tense, number (plural/singular), case, and so on. They are especially used in corpus searches and in-text analysis tools and algorithms.



(a)

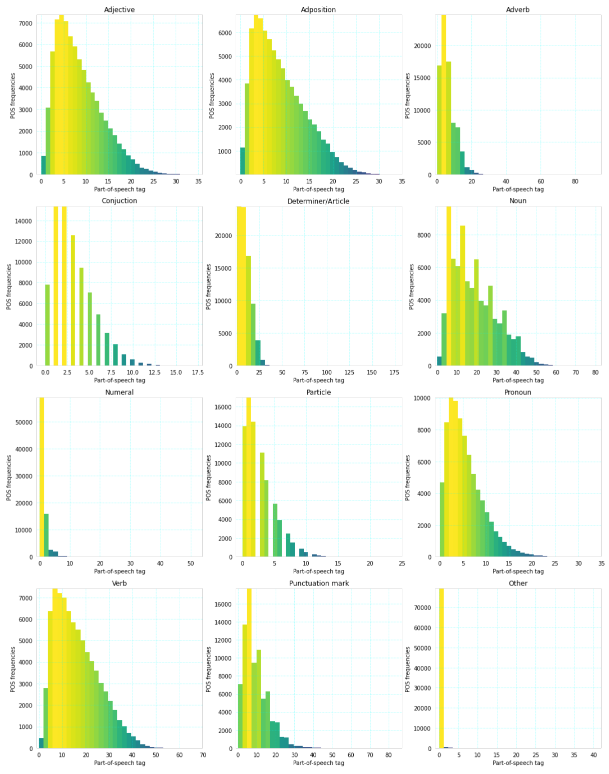


(b)

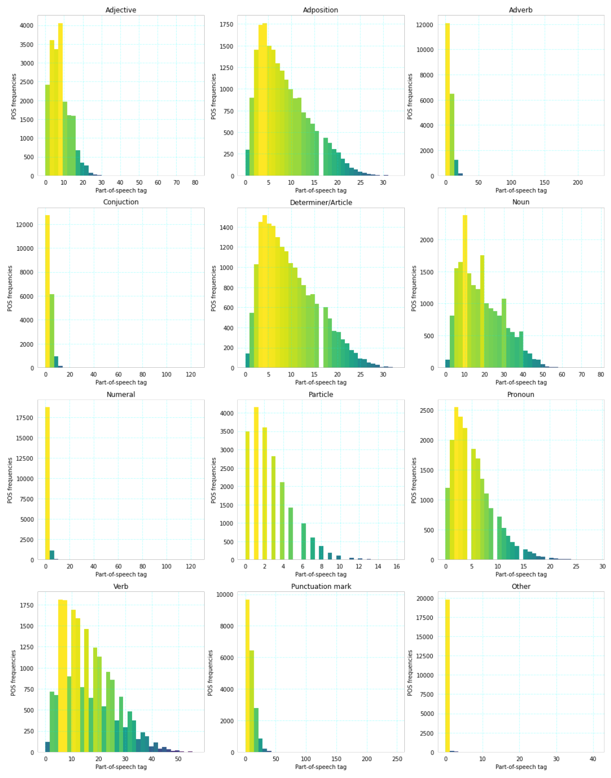
Figure 9: Inspect part-of-speech tags for reviews' titles in train (left) and test (right) samples.

The most expressive parts of speech regarding the feelings expressed are adjectives, adverbs, and verbs. For this reason, it is essential to identify the distribution of these parts of speech in the reviews and their titles to see how qualitative the samples are.

Most titles contain nouns that indicate that people tend to put the product name (proper noun) or just the type of product in the title of a review. In addition, almost 10% of the samples have "Missing value" so we can conclude that the reviews' titles contain relatively little information about the user's opinion of that product.



(a) Inspect part-of-speech tags for reviews in training samples.



(b) Inspect part-of-speech tags for reviews in test samples.

Figure 10: POS tags in Reviews

The textual content of the reviews is richer in terms of information (rich in adjective and adjective-adverbs combinations) on the polarity of the user's feelings towards the correspondent product, as can be deduced from the figure 10. Several specialized works explore this area, namely identifying sentiments restricting only adjectives, adverbs, and verbs as features [1], [4].

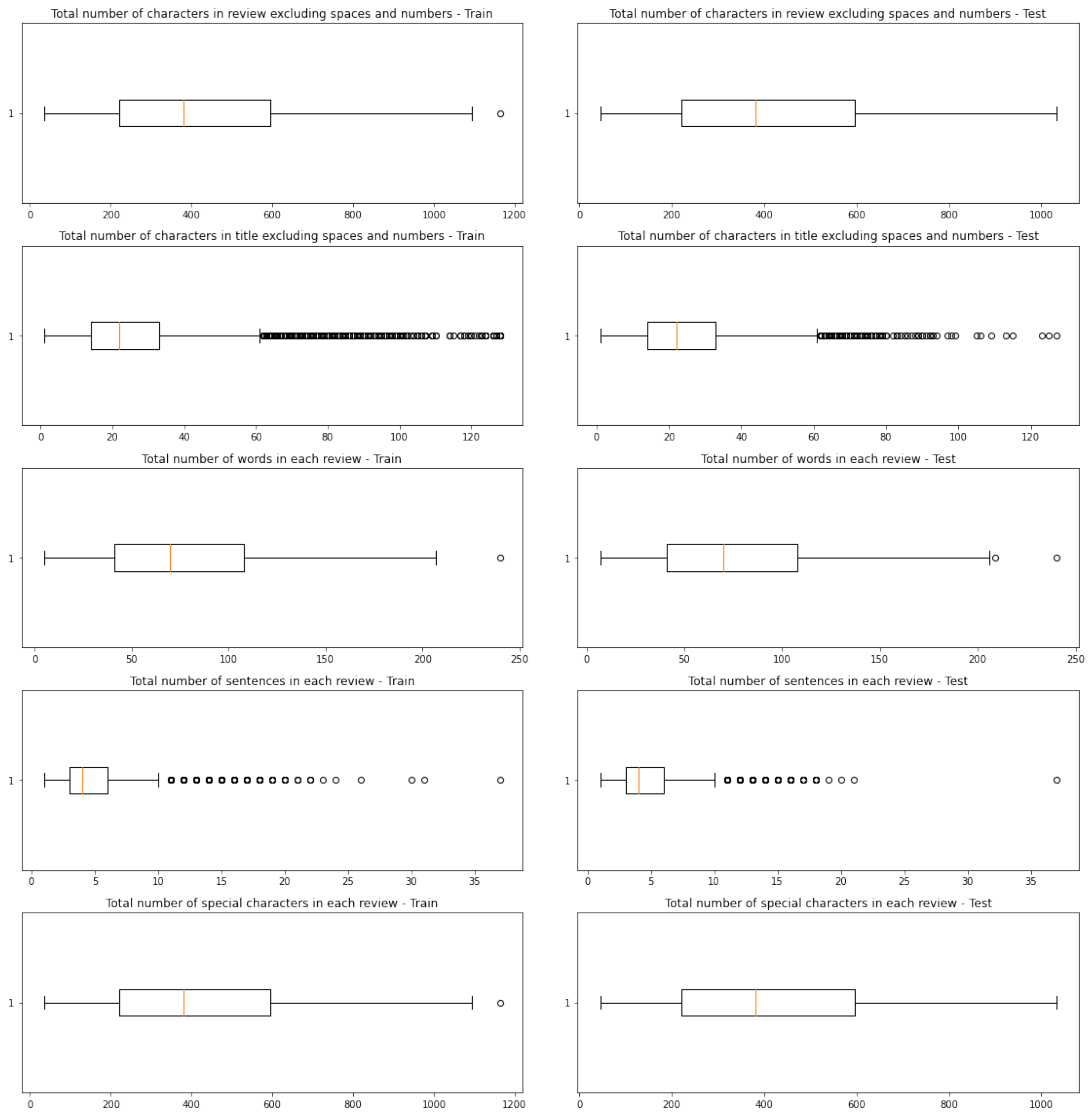
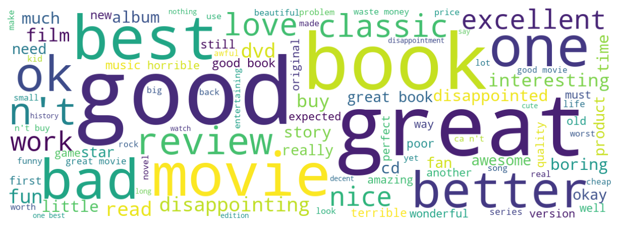


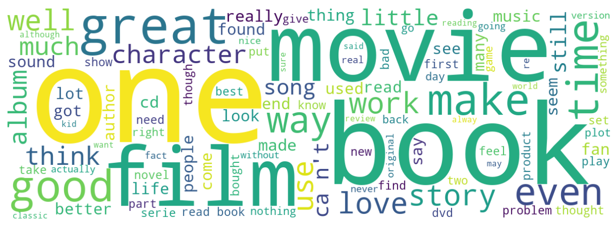
Figure 11: Boxplots for statistical count features for train samples

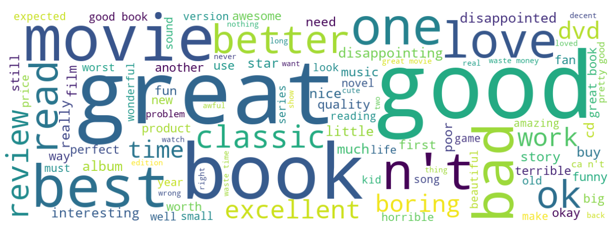
We also visualize the most common words excluding stopwords, bigrams, and trigrams in the reviews and their titles.

### 3.2.3 Most frequent words excluding stopwords

As can be seen, the common words in the title of the review are also found in the content of the review (figure 12). Therefore, the title of the review contains redundant information and we decided, taking into account the previous analyzes, to remove the title column.


(a) Most common 100 words in reviews' titles - Train


(b) Most common words in reviews - Train


(c) Most common 100 words in reviews' titles - Test

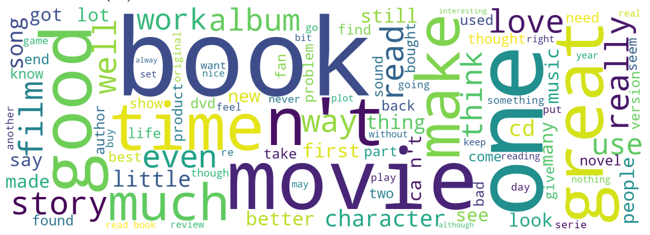

(d) Most common 100 words in reviews - Test

Figure 12: Top 100 frequent words

When analysing textual data, it is common to make charts showing the frequency of words. This gives a good impression of what is talked about most in the reviews.

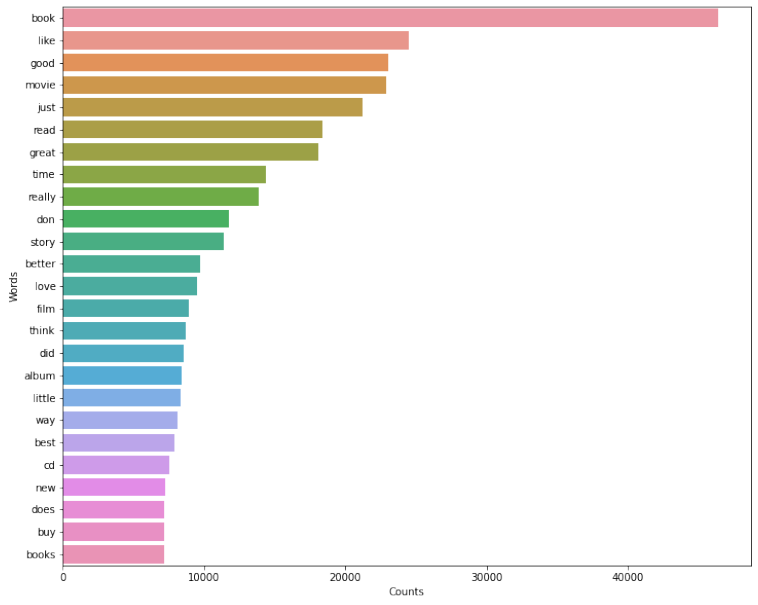
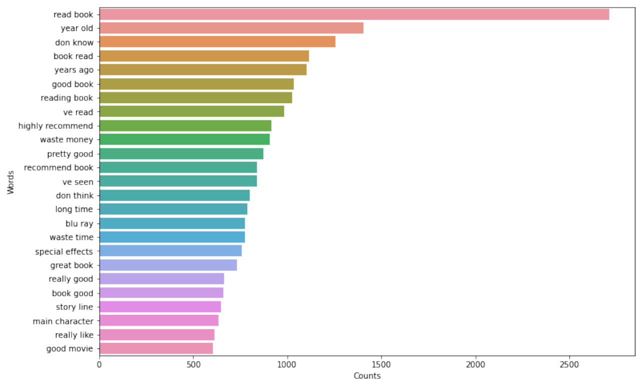


Figure 13: Frequent words

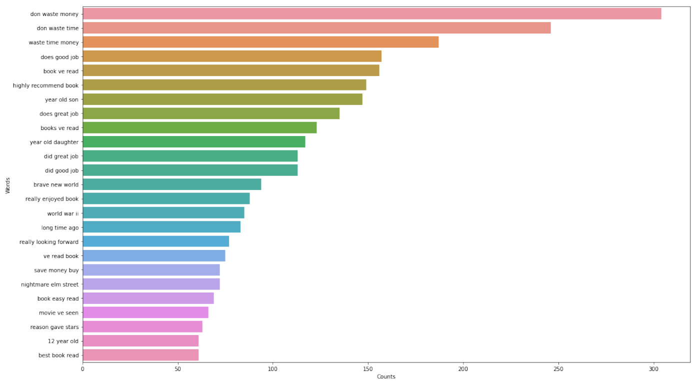
It is interesting that among the most common words (figure 12) in the reviews, we find adjectives and adverbs that express the polarity of the feeling such as "good", "great" etc.

### 3.2.4 Most frequent bigrams and trigrams

The advantage of examining the most frequent n-grams instead of the most frequent words is that it allows us to better understand the context in which the word was used.



(a) Most common bigrams



(b) Most common trigrams

Figure 14: N-grams frequency

### 3.2.5 Topic modeling exploration

Topic modeling is an unsupervised learning technique used to extract the main topics from documents. We will use Latent Dirichlet Allocation algorithm to identify the main topics in the Reviews.

The main idea of Latent Dirichlet Allocation (LDA) is to model texts as arising from multiple topics, where a topic is defined by a distributionn on over a fixed vocabulary of terms from a given corpus. More concretely, we asume that K topics are associated.

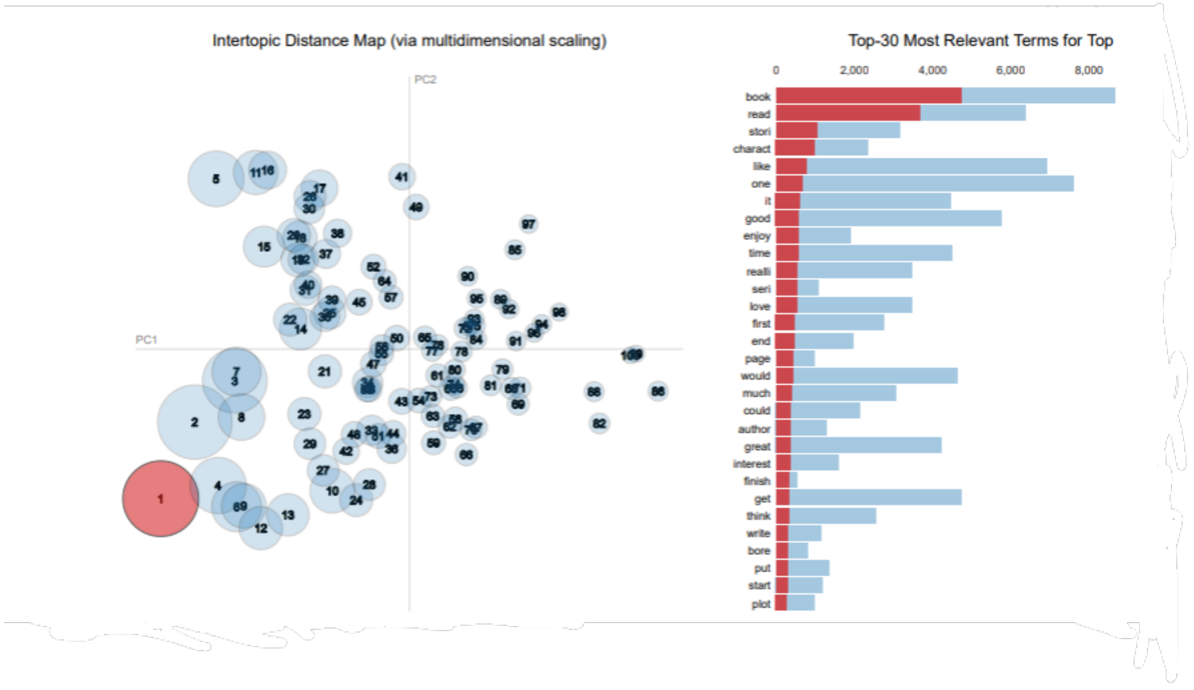


Figure 15: Topic modeling using LDA algorithm

The motivation for using this algorithm to identify the main topic of each review came from looking at the most common words, bi-grams, and tri-grams. Only from these, we could identify two important categories: movies and books, but we want an overview of the topics/product categories present in the review in order to identify the complexity of the dataset. We can also see the most common words divided by topics.

Before applying the LDA algorithm we have to perform another stage of preprocessing on our text data to be suitable for this algorithm:

1. We filter out the stop words because they are common in any text, and they do not contain significant information for releaving topics.
2. Another common pre-processing step is converting words to their "root" lexical forms (e.g., "politics" and "political" would be converted to "politic").
3. Compute a vocabulary by giving an unique id to each word in the documents and then represent the textual data with a vector which contains the unique id instead of the respective words.
4. Eliminate the tokens with a frequency equal to 1 from each document, as some tokens are not even real words or are rarely used in the corpus.

We find around 100 different topics (some of them are correlated). Each topic is related to a product subcategory and has among the most common words per topic adjectives, adverbs, verbs that emphasize the polarity and intensity of the sentiment about that product.

### 3.2.6 Length of Reviews vs. Ratings

It will be interesting to see if the review length in terms of number of words changes with the rating and if it is any linear correlation between the review length and the rating.

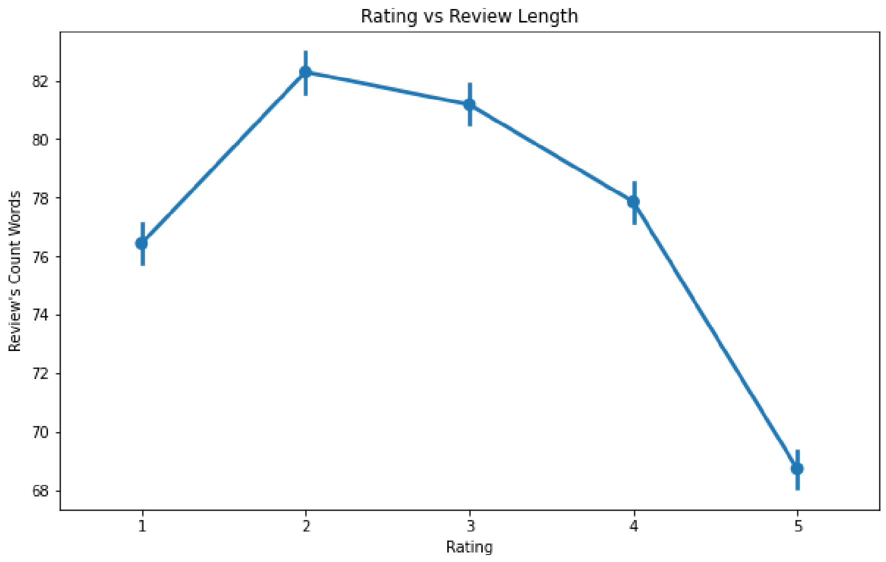


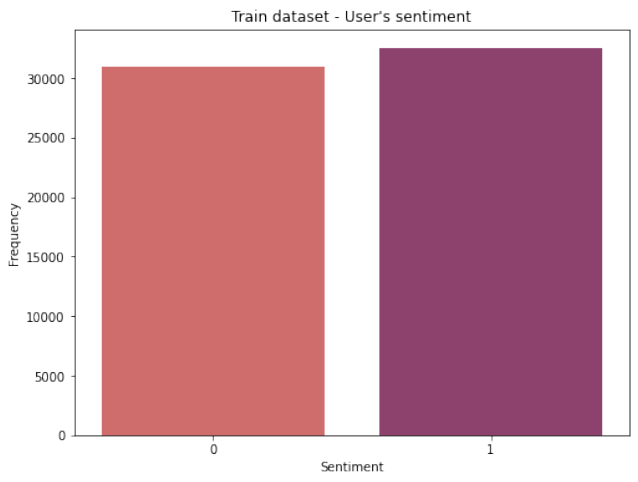
Figure 16: Length of Reviews vs. Rating

We can see (figure 16) that the tendency of users to write as little as possible when they are satisfied with the product. Also, there is no linear correlation between these two features.

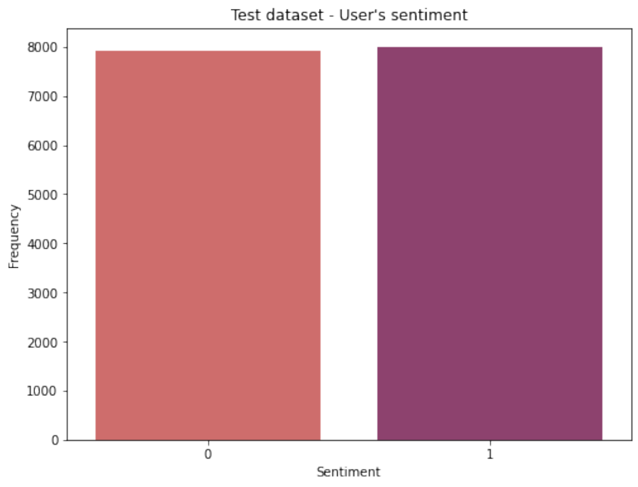
#### 3.2.7 Count of Reviews for each Rating and Sentiment Polarity

| Ratings | | |
|---|---|---|
| Rating | Trainnig | Test |
| 1 | 14957 | 3833 |
| 2 | 15997 | 4080 |
| 3 | 16510 | 4095 |
| 4 | 16121 | 3979 |
| 5 | 16373 | 4012 |

We can conclude that we deal with an balanced set regards the binary classification (figure 17), respectevely an unbalanced dataset regards the multiclass classification (figure 18).



(a) Train data



(b) Test data

Figure 17: Sentiment polarity on 2-scale

We are interested in the correlation between the reviews lengths as a possible input value with the sentiment polarity as output variable in order to provide insight into it may or may not be relevant as input for developing our models.

After calculating the Spearman's rank correlation coefficient, we concluded that there is no linear correlation between the length of the review regards of the number of words and the user's sentiment polarity regards 3 classes or 2 classes.

*Spearman's correlation for 2-scale polarity: -0.080*

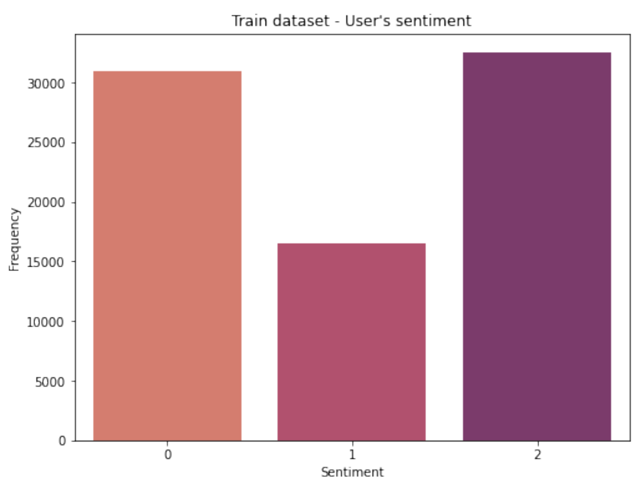
*Spearman's correlation for 3-scale polarity: -0.072.*

## 4 Experiments and their outcomes

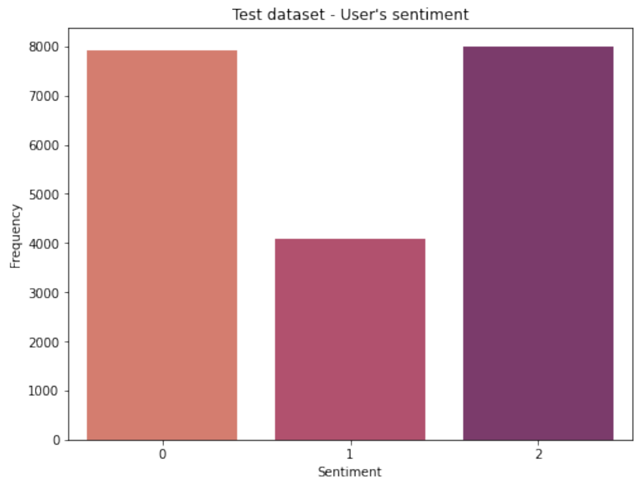
### 4.1 Evaluation measures

The main measures for evaluating the model are *accuracy* and $F_1^{PN}$. The last one is the average of the $F_1$ values calculated for the positive and negative classes, respectively. It is calculated as follows:

$$F_1^{PN} = \frac{1}{2}(F_1^P + F_1^N)$$



(a) Train data



(b) Test data

Figure 18: Sentiment polarity on 3-scale

Moving forward, we use the average recall value as a secondary measure to evaluate the model, because this measure has the desirable theoretical properties for this task.

For the text classification on the two-point scale it is calculated as follows:

$$AvgRec = \frac{1}{2}(R^N + R^P)$$

where $R^N$, and $R^P$ are the recall values for the NEGATIVE and POSITIVE classes.

For the text classification on the three-point scale it is calculated as follows:

$$AvgRec = \frac{1}{3}(R^N + R^U + R^P)$$

where $R^N$, $R^U$ and $R^P$ are the recall values for the NEGATIVE, NEUTRAL and POSITIVE classes.

The advantage of the *AvgRec* value over the standard accuracy is that it is more robust to the class imbalance. The accuracy of the majority class classifier is the relative frequency, known as the majority class prevalence, which can be greater than 0.5 if the test set is highly unbalanced. $F_1$ is also sensitive to class imbalance for the same reason presented above. Another advantage of *AvgRec* over $F_1$ is that *AvgRec* is invariant in switching the POSITIVE class with the NEGATIVE class, while is not.

## 4.2 Feature extraction

The analysis described in the previous section showed that most of the information about the user's sentiment is contained in the product review. Therefore, we chose only the product review as input and thus reduced the number of parameters considerably. It is necessary for each sequence to have the same dimensionality, same number of tokens in the input sequence. The number of input tokens was chosen based on the median number of words and sentences. As a consequence, sequences containing fewer tokens are filled with padding tokens, and those containing more tokens are reduced to the first set number of tokens.

After splitting the text into the token list, the encoding is then converted to a list of vocabulary indexes. We normalized the data and this had a positive impact on the predictions results.

## 4.3 Proposed approaches

### 4.3.1 Random Forest

Random forests, also known as random decision forests, are an ensemble learning method for classification, regression, and other problems that works by training a large number of decision trees. The random forest's output for classification problems is the class chosen by the most trees. The mean or average prediction of the individual trees is returned for regression tasks. Random choice forests correct the tendency of decision trees to overfit their training set. Random forests outperform decision trees in most cases, but they are less accurate than gradient-boosted trees. Data features, on the other hand, can have an impact on their performance.

#### 4.3.1.1 Hyperparameter Optimization

The grid search algorithm (from the scikit-learn library) was used to do the hyperparameter optimisation operation, utilising a 5 fold cross-validation splitting technique. Trying all of the possible combinations of the following parameters yielded the optimum hyperparameter configuration:

- *max_features*: {*'auto'*, *'sqrt'*, *'log2'*}
- *n_estimators*: $\{10, 50, 75, 100, 125, 150, 200\}$
- *max_depth*: $\{2, 5, 10, 25, 50, 75, 100\}$
- *max_leaf_nodes*: $\{2, 3, 5, 7, 10, 15, 25, 50, 75\}$

#### 4.3.1.2 Training and Validation Results

Depending on the number of classes, different hyperparameter configurations have been obtained:

| **Num. classes** | **max_features** | **n_estimators** | **max_depth** | **max_leaf_nodes** |
|---|---|---|---|---|
| *2* | *'auto'* | *100* | *75* | *75* |
| *3* | *'auto'* | *100* | *100* | *75* |

Table 1: Random forest best hyperparameter configurations

For these hyperparameter configurations the following training and validation results were obtained:

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.828* | *0.827* | *0.830* | *0.828* |
| *3* | *0.510* | *0.501* | *0.498* | *0.499* |

Table 2: Random forest Train data results

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.829* | *0.827* | *0.830* | *0.829* |
| *3* | *0.501* | *0.500* | *0.495* | *0.497* |

Table 3: Random forest Validation data results

### 4.3.2 SVM (Support Vector Machine)

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection. The advantages of support vector machines are: Effective in high dimensional spaces. Still effective in cases where number of dimensions is greater than the number of samples.

For optimizing the hyperparameters, the grid search algorithm (from the scikit-learn library) was used, utilising a 3 fold cross-validation splitting technique. Trying all of the possible combinations of the following parameters yielded the optimum hyperparameter configuration:

- $C$: $\{0.25, 0.5, 0.75, 2, 30\}$
- $gamma$: $\{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$
- $kernel$: $\{'rbf'\}$

#### 4.3.2.1 Training and Validation Results

Depending on the number of classes, different hyperparameter configurations have been obtained:

| **Num. classes** | **C** | **gamma** | **kernel** |
|---|---|---|---|
| *2* | *2* | *0.3* | *rbf* |
| *3* | *2* | *0.5* | *rbf* |

Table 4: SVM best hyperparameter configurations

For these hyperparameter configurations the following training results were obtained:

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.7038* | *0.9856* | *0.2744* | *0.4293* |
| *3* | *0.6104* | *0.6938* | *0.6104* | *0.5881* |

Table 5: SVM Training data results

For these hyperparameter configurations the following validation results were obtained:

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.5791* | *0.4049* | *0.0672* | *0.1153* |
| *3* | *0.4131* | *0.3745* | *0.4131* | *0.3611* |

Table 6: SVM Validation data results

### 4.3.3 XGBoost

XGBoost is the abbreviation of eXtreme Gradient Boosting, which is an open source implementation of the gradient boosted trees algorithm. It's used for supervised learning for both regression and classification. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. Boosting is an ensemble method, i.e is a procedure which assembles predicts from several models into one. It achieves this by running each individual predictor sequentially and modelling it based on its predecessor's error, giving more weight to predictors that perform better.

#### 4.3.3.1 Hyperparameter Optimization

For the optimization of the hyperparameters has been used Optuna which automatically searches and finds the optimal values by trial and error. It uses an internal record of all runs and it estimates the next possible area of the best values and then it search in that area. More specifically, it employs a Bayesian optimization algorithm called Tree-structured Parzen Estimator.

The ranges for all hyperparameters are:

- *booster*: *"gbtree", "gblinear", "dart"*
- *lambda*: *min: 1e-8, max: 1.0*
- *alpha*: *min: 1e-8, max: 1.0*
- *subsample*: *min: 0.2, max: 1.0*
- *max_depth*: *min: 3, max: 9*
- *min_child_weight*: *min: 2, max: 10*
- *grow_policy*: *"depthwise", "lossguide"*

#### 4.3.3.2 Training and Validation Results

Depending on the number of classes, different hyperparameter configurations have been obtained:

| **Num. classes** | **booster** | **lambda** | **alpha** | **subsample** |
|---|---|---|---|---|
| *2* | *dart* | *0.076* | *0.055* | *0.467* |
| *3* | *gbtree* | *0.076* | *0.055* | *0.467* |

| **Num. classes** | **max_depth** | **min_child_weight** | **grow_policy** |
|---|---|---|---|
| *2* | *3* | *3* | *lossguide* |
| *3* | *3* | *3* | *lossguide* |

Table 7: XGBoost best hyperparameter configurations

For these hyperparameter configurations the following training and validation results were obtained:

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.713* | *0.701* | *0.718* | *0.704* |
| *3* | *0.395* | *0.291* | *0.391* | *0.313* |

Table 8: XGBoost Train data results

| **Num. classes** | **Accuracy** | **Precision** | **Recall** | **F1 score** |
|---|---|---|---|---|
| *2* | *0.591* | *0.573* | *0.591* | *0.577* |
| *3* | *0.362* | *0.282* | *0.371* | *0.294* |

Table 9: XGBoost Test data results

### 4.3.4 Long short-term memory (LSTM)

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can process not only single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is applicable to tasks such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDSs (intrusion detection systems).

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell.

#### 4.3.4.1 Architecture

The recurrent neural network's architecture that is used here is as follows:

1. An embedding layer, which transforms the input data into dense vectors of fixed size.
2. An LSTM layer.
3. A dropout layer to prevent overfitting.
4. A dense layer with 2 or 3 units, based on the number of classes used, which represents the output layer. For activation, it uses the *softmax* function.

Also, an *Adam* optimizer is used, and the loss function that was used is the *binary crossentropy.*

#### 4.3.4.2 Hyperparameter Optimization

For optimizing the hyperparameters, the *optuna* library was used. Each hyperparameter was given a value range and then, for each trial, the library would choose a random value for each hyperparameter. The ranges for all hyperparameters are:

- *embedding_output_dim*: *min: 64, max: 256*
- *lstm_units*: *min: 8, max: 64*
- *dropout_value*: *min: 0.1, max: 0.4*
- *learning_rate*: *min: 1e-6, max: 1e-2*

#### 4.3.4.3 Training and Validation Results

Depending on the number of classes, different hyperparameter configurations have been obtained:

| **Num. classes** | **embedding_output_dim** | **lstm_units** | **dropout_value** | **learning_rate** |
|---|---|---|---|---|
| *2* | *112* | *40* | *0.3* | *0.006043* |
| *3* | *112* | *16* | *0.25* | *0.003554* |

Table 10: LSTM best hyperparameter configurations

For these hyperparameter configurations the following training and validation results were obtained:

| Num. classes | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|
| *2* | *0.9477* | *0.9477* | *0.9477* | *0.9458* |
| *3* | *0.8540* | *0.8754* | *0.8273* | *0.8244* |

Table 11: LSTM Training data results

| Num. classes | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|
| *2* | *0.7896* | *0.7896* | *0.7896* | *0.7827* |
| *3* | *0.6548* | *0.6814* | *0.6171* | *0.6091* |

Table 12: LSTM Validation data results; For validation, 20% of the training data was used.

#### 4.3.5 BERT (Bidirectional Encoder Representations from Transformers)

BERT[3] is a bidirectional pre-trained transformer, an attention-based model that uses a combination of masked modeling objective and next sentence prediction for language modeling. It uses an attentional mechanism of Transformer that learns contextual relationships between words or subwords in a text. As described in the previous section, a Transformer in its vanilla form consists of two separate mechanisms: an encoder that reads the text input and encodes a distributed representation of the language units, and a decoder that predicts the task. Since the goal of the BERT model is to create a language model, only the encoder mechanism is needed.

Its main feature is that the Transformer encoder reads the entire sequence of words at once. Therefore, it is considered bidirectional, although it would be more accurate to say that it is non-directional. This feature allows the model to learn the context of a word based on its entire surrounding.

##### 4.3.5.1 Experiments

We conducted two sets of experiments with two neural networks based on the masked language model (MLM) BERT. BERT was designed to create a bi-directional representation of the text, i.e. it allows the model to learn the context of a word based on all the circumstances (conditioning on both the right and left contexts).

Both models were defined by loading the existing neural architecture with defined layers, removing the last output layer from the specified network, and replacing it with a new output layer according to the requirements of the Opinion Mining task. We add a linear layer that performs a linear transformation and returns the probabilities of the two classes in the case of the first task, binary classification, or the logits of the three classes for multiclass classification.

In the case of the first model, the weights of the BERT model are stabilized (froze), and only the classification layer is trained. This method has the advantage of being less computationally expensive. For the second model, the training is performed on the entire architecture. Although the following approach is more expensive in terms of time and space used, it achieves better results because it allows the model to learn the textual representations specific to the requirement.

We obtained the best results with the second approach: first initialized the model with the pre-trained parameters, and then fine-tuned all the parameters using the labeled data from our task.

##### 4.3.5.2 Feature extraction.

After finishing the first preprocessing step, the textual component of the review is processed to match the special input format expected by the BERT model. Each text sequence is divided into individual language units, specific vocabulary tokens using the *BertTokenizer* method.

Each input sequence begins with a [CLS] token, the embeddings of which are used for sentence-level classification. Similarly, each sequence ends with the [SEP] token. In the case of multi-sentence posts, an [SEP] token is inserted to separate the sentences.

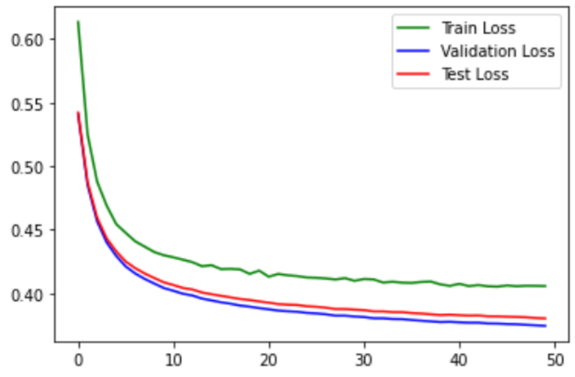
It is necessary for each sequence to have the same dimensionality $n'$, where $n'$ is equal to 100 for the binary classification task, respectively 128 for the multiclass classification, approximately the median number of language units in a review plus the average number of tokens needed to separate the sentences (average number of sentences in a review). We chose to use the median rather than the other means of central tendency because our data is skewed (i.e. form a skewed distribution) and we chose not to remove the outliers. Instead of removing the outliers from our data, we try to choose the hyperparameter (number of tokens fed into the model) as a function of the median words and sentences per review. The reason we did not remove the outliers is that they do not significantly distort the mean and they actually help the model to generalize, so they are not that influential.

As a consequence, sequences containing fewer tokens are filled with the [PAD] token, and those containing more tokens are reduced to the first $n' - 1$ tokens and the [SEP] token is appended at the end. We also do not want the end result to be influenced by the tokens used for padding. For this, we will use the attention mechanism of the BERT model, so that, when calculating the representations of each input token, no attention is given to the [PAD] tokens. We accomplish this by providing an attention mask along with distributed representations of words. The attention mask consists of a tensor that contains values of 0 for [PAD] tokens and 1 for the rest.

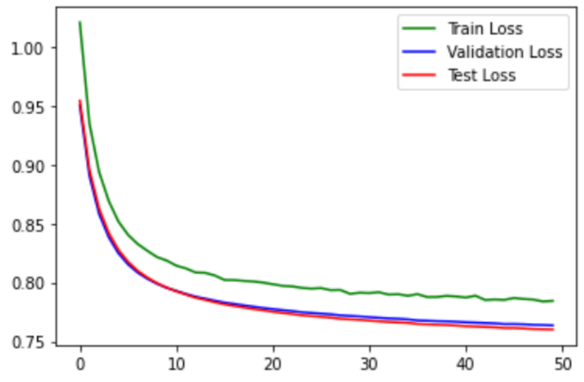
The last step of the second stage of textual data processing consists in replacing the tokens we obtained by following the above steps with their corresponding indices from the BERT vocabulary, obtaining a tensor i, with $i \in R^{n' \times d}$.

#### 4.3.5.3 Model training.

We use Cross-Entropy Loss Function and AdamW optimizer for the training process. We set the learning rate to 3e-5, the size of mini-batch to 32, the dimension of word-embeddings to 128. In order to avoid overfitting dynamically updating the learning rate tricks are also employed. Due to the imbalance of the dataset we also estimate the class weights when computing the loss.



(a) Binary classification task



(b) Multiclass classification

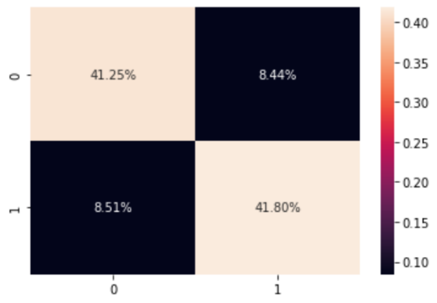
Figure 19: Loss for BERT fine-tuned model

## 4.4 Results

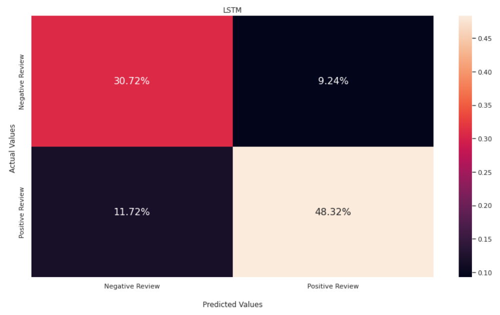
### 4.4.1 Text classification on two-point scale

*Binary classification:* Given a product review in plain text, clasify the review on a two-point scale(negative or positive sentiment of a user towards a product).

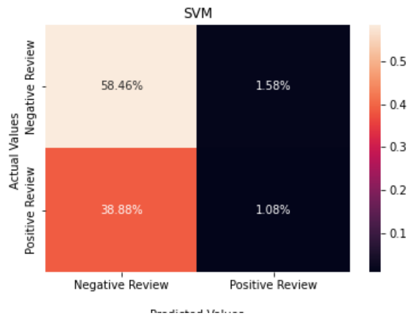
| Model | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|
| BERT fine-tuned | *0.831* | - | *0.828* | *0.821* |
| Random Forest | *0.829* | *0.827* | *0.830* | *0.829* |
| LSTM | *0.7963* | *0.7963* | *0.7963* | *0.7845* |
| XGBoost | *0.591* | *0.573* | *0.591* | *0.577* |
| SVM | *0.596* | *0.404* | *0.027* | *0.050* |



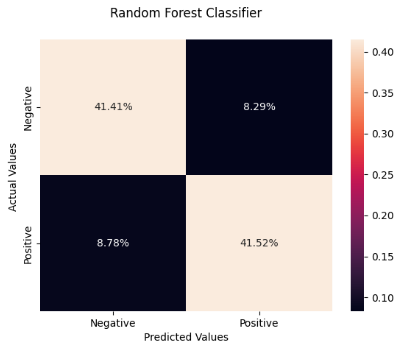
(a) BERT



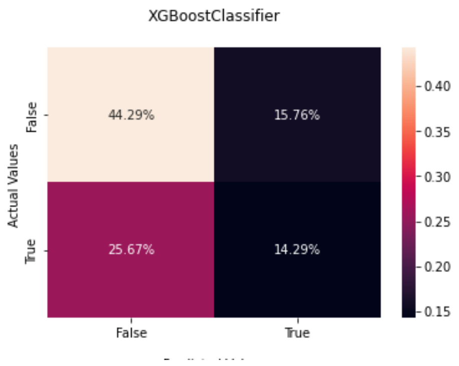
(b) LSTM



(c) Support Vector Machines



(d) Random Forest



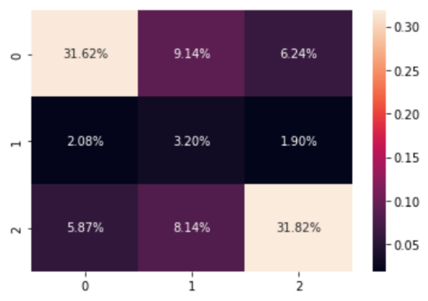
(e) XGBoost

Figure 20: Confusion matrix

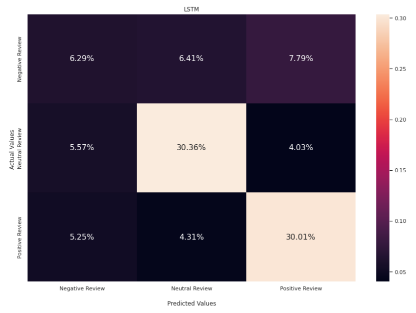
### 4.4.2 Text classification on three-point scale

*Multiclass classification:* Given a product review in plain text, clasify the review on a three-point scale(negative, neutral or positive sentiment of a user towards a product).

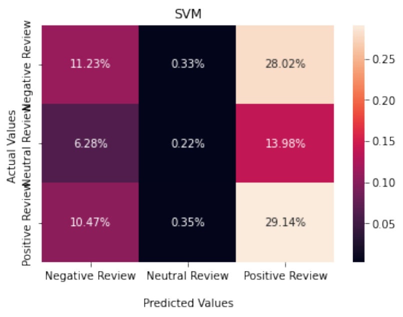
| Model | Accuracy | Precision | Recall | F1 score |
|---|---|---|---|---|
| BERT fine-tuned | *0.667* | - | *0.582* | *0.470* |
| Random Forest | *0.521* | *0.512* | *0.515* | *0.507* |
| LSTM | *0.6594* | *0.6226* | *0.6903* | *0.6116* |
| XGBoost | *0.331* | *0.214* | *0.334* | *0.263* |
| SVM | *0.406* | *0.372* | *0.406* | *0.345* |



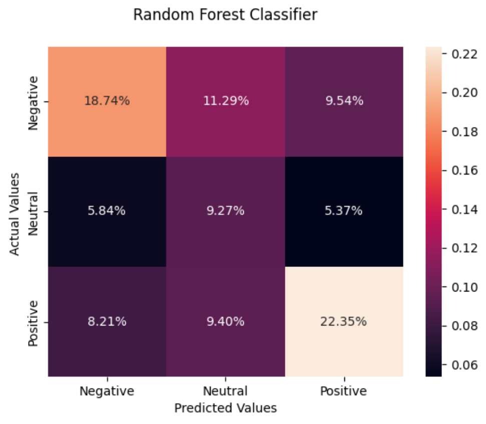
(a) BERT



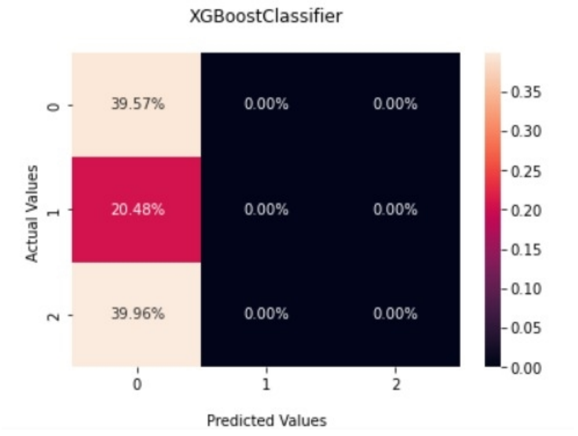
(b) LSTM



(c) Support Vector Machines



(d) Random Forest



(e) XGBoost

Figure 21: Confusion matrix

## 5 Conclusions

After a thorough analysis of the data, we were able to find redundant features that did not contribute significantly to predicting a user's sentiment towards a particular product. Therefore, we omitted these features and kept only the important information in order to reduce the number of parameters and the amount of memory used.

We choose our hyperparameters depending on the descriptive statistic analysis we made. We used the median number of words per review and the median number of sentences per review to choose the input length. We chose to use the median rather than the other means of central tendency because our data is skewed (i.e. a skewed distribution) and we chose not to remove the outliers because deduced from the experiments that these outliers help the model in generalization.

Regarding the results and the experiments we conducted, multi-class classification seemed more difficult than simple binary classification, which can be seen in the accuracy results. This is due to the fact that dataset is strongly unbalanced, because there are much fewer samples that have a neutral sentiment compared to the positive and negative ones. It can be seen in confusion matrix with the 3 classes that there were fewer accurate predictions with the neutral sentiment than the other two.

At the same time, we can conclude that when trying to solve complex problems such as opinion analysis, the choice of pre-trained models and the use of components from already defined network architectures can be very useful by achieving better results in less time.

## References

[1] Farah Benamara et al. "Sentiment analysis: Adjectives and adverbs are better than adjectives alone". In: *ICWSM* (Nov. 2005).

[2] *Categorizing and Tagging Words.* URL: `https://www.nltk.org/book/ch05.html`.

[3] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* 2019. arXiv: `1810.04805 [cs.CL]`.

[4] Sajjad Haider et al. "Impact analysis of adverbs for sentiment classification on Twitter product reviews". In: *Concurrency and Computation: Practice and Experience* 33.4 (2021). e4956 CPE-18-0194.R2, e4956. DOI: `https://doi.org/10.1002/cpe.4956`. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.4956`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.4956`.

[5] Ashish Vaswani et al. *Attention Is All You Need.* 2017. arXiv: `1706.03762 [cs.CL]`.

[6] Xiang Zhang, Junbo Zhao, and Yann LeCun. *Character-level Convolutional Networks for Text Classification.* 2016. arXiv: `1509.01626 [cs.LG]`.