Exploratory Data Analysis - Final Project

Amazon Reviews for Beauty and Fashion Products - Sentiment Classification

Cristea Stefania, Neagu Anastasia, Sora Andreea

1. Introduction

In our project, we will analyse product reviews on Amazon from the beauty and fashion category, in order to identify and classify the sentiments of the users towards the products. Sentiment analysis is a branch of natural language processing that deals with the identification and evaluation of sentiments expressed in a text. The domain has extensive applications in the analysis of public opinion or in the understanding of user reactions in the online environment. In this project, we will classify the reviews into 3 classes: positive, negative and neutral.

2. Description of the dataset

This dataset has been sampled from a larger dataset created by Jianmo Ni, which contains 233.1 million product reviews and their metadata from Amazon, spanning May 1996 - Oct 2018. As the dataset is very large and already divided by the product categories Amazon provides, we have selected only four of those, namely: Amazon Fashion, All Beauty, Clothing Shoes and Jewellery and Luxury Beauty, in order to maintain a theme in the data we use.

Initially, the data was used for 'generating reviews (or 'tips') as a form of explanation as to why a recommendation might match a user's interests' (Ni & Li, 2019). The data the owners provide on their website is the 5-core, or the dense subset, meaning each user and each product has at least five reviews. Since the data contained duplicates, we also had to make sure to remove them after reading the files, as they offered no additional information.

JSON Review Example:

```
{
  "reviewerID": "A2SUAM1J3GNN3B",
  "asin": "0000013714",
  "reviewerName": "J. McDonald",
  "vote": 5,
  "style": {
    "Format:": "Hardcover"
  },
  "reviewText": "I bought this for my husband who plays the piano.  He
is having a wonderful time playing these old hymns.  The music  is at
times hard to read because we think the book was published for singing
from more than playing from.  Great purchase though!",
  "overall": 5.0,
  "summary": "Heavenly Highway Hymns",
  "unixReviewTime": 1252800000,
  "reviewTime": "09 13, 2009"
}


{
"image": ["https://images-na.ssl-images-
amazon.com/images/I/71eG75FTJJL._SY88.jpg"],
"overall": 5.0,
"vote": "2",
"verified": True,
"reviewTime": "01 1, 2018",
"reviewerID": "AUI6WTTT0QZYS",
"asin": "5120053084",
"style": {
      "Size:": "Large",
      "Color:": "Charcoal"
      },
"reviewerName": "Abbey",
"reviewText": "I now have 4 of the 5 available colors of this shirt...
",
"summary": "Comfy, flattering, discreet--highly recommended!",
"unixReviewTime": 1514764800
}
```

- reviewerID: the ID of the user who made the review;
- asin: the ID of the product reviewed;
- reviewerName: the name of the reviewer;
- vote: the amount of people who found the review helpful;
- style: metadata describing the product;
- reviewText: the review itself;
- overall: the rating ;
- summary: the summary of the review;
- unixReviewTime: the unix time the review was made;

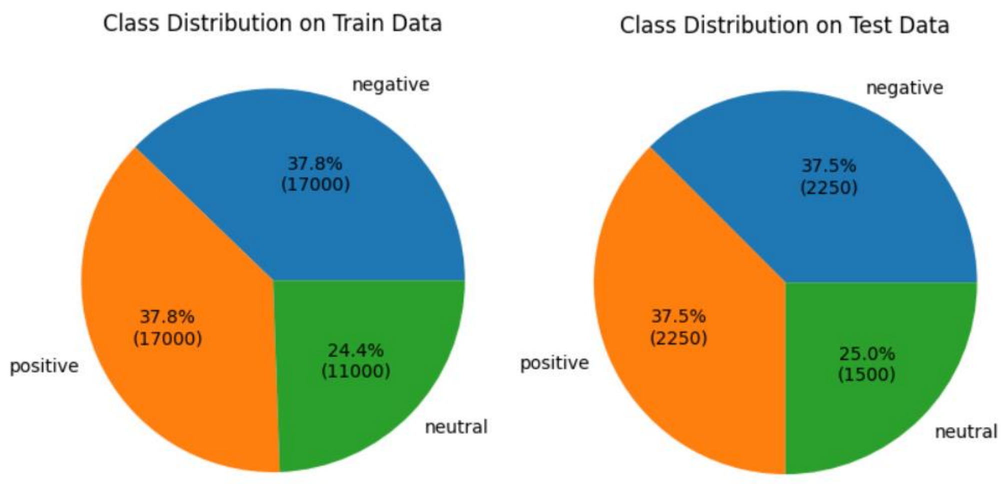
- reviewTime: the raw time the review was made;
- image: the user can provide an image of the item they received.

Of these attributes, the ones useful for our project were reviewText and overall. The ratings were originally on a scale from 1 to 5, however we only required three classes - positive, neutral and negative, so we converted all 4-5 star reviews into positive, 3 star reviews into neutral and the rest into negative reviews. We saved this new classification, along with the review text, into a separate CSV file.

3. Exploratory data analysis

The exploratory data analysis part is particularly important in any machine learning task. With the help of histograms or various similar charts, we can observe the distribution of the data and decide which techniques to use. This analysis allows us to understand our data, which helps us make informed decisions about data preprocessing and model choice.

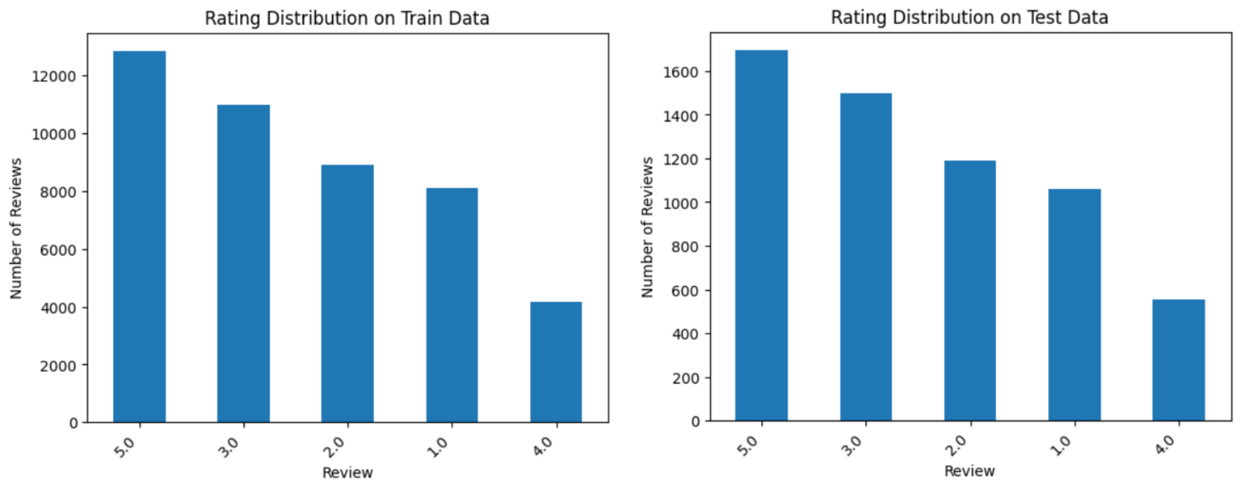
Class Distribution



Class distribution in the train and test data

The above pie charts illustrate the distribution of our dataset across the three classes that we will be using to train and test the model. The amount of neutral reviews is slightly lower than that of the positive and negative ones, as the latter two classes are more important from the standpoint of the practical use of this model - it is more important to know whether a review is positive or negative, whereas neutral reviews offer less information.

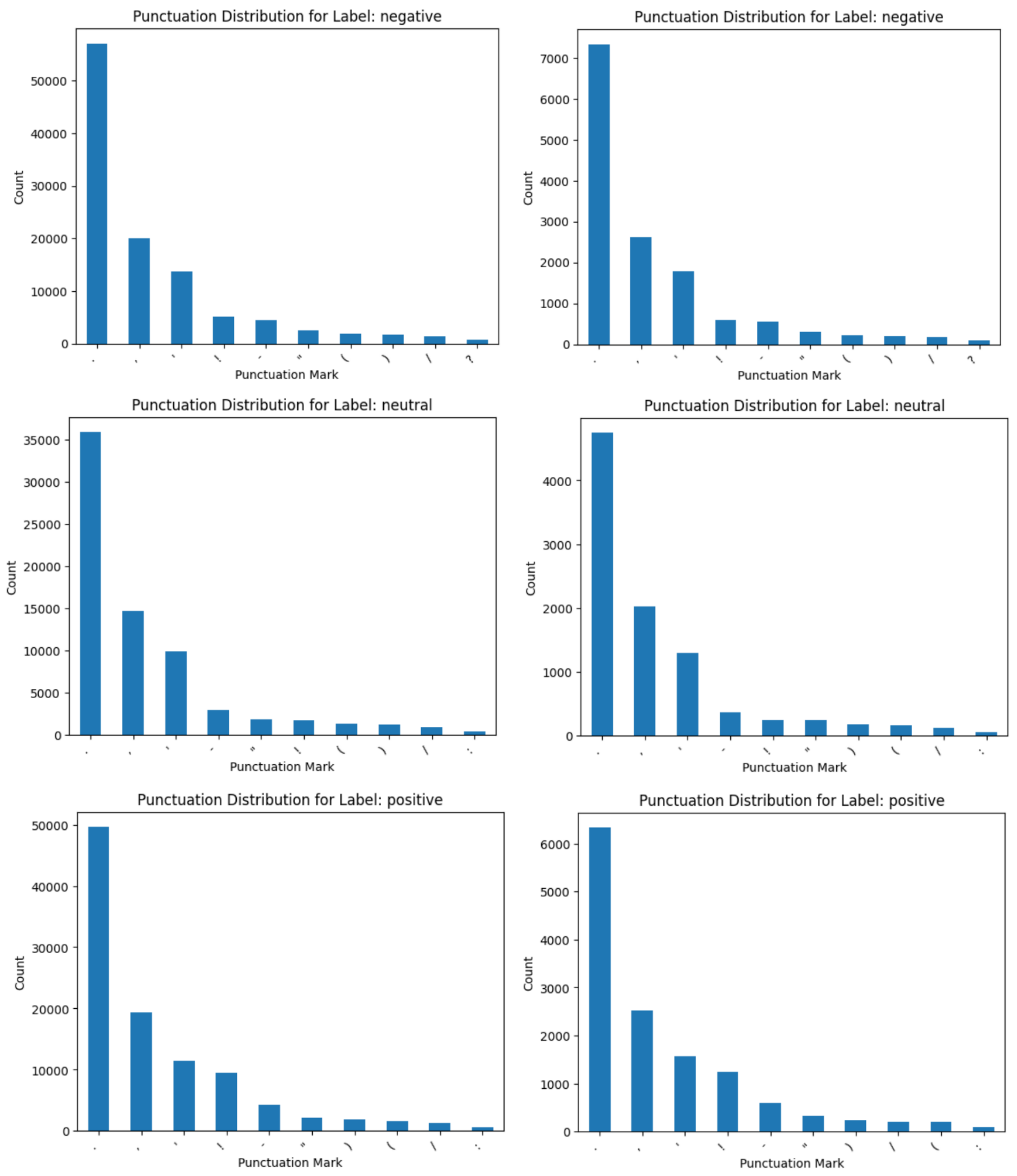
Rating Distribution



Rating distribution in the train and test data

The bar pie charts illustrate the rating distribution of the dataset in the classic five-point rating system used by Amazon. Both the train and test sets have the same hierarchy, having the highest number of 5-star reviews and the lowest number of 4-star reviews. The second class are the neutral reviews, namely the ones having 3 stars, followed by the two ratings which make up the negative review class.

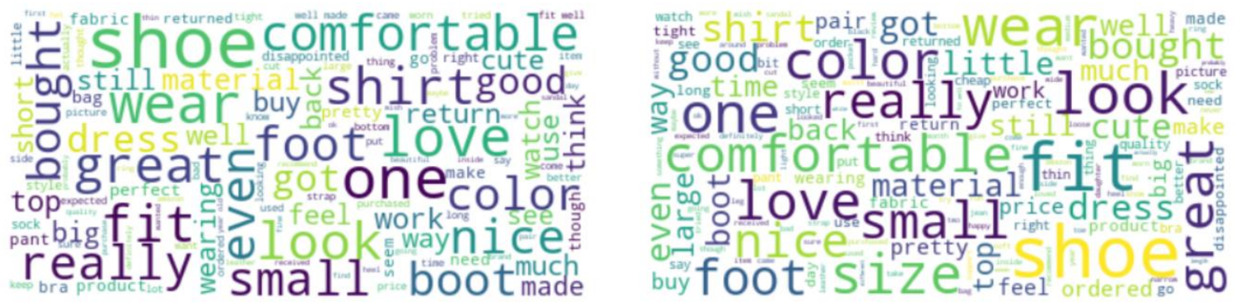
Punctuation



Punctuation distribution divided by labels on the train(left) and test(right) data

As punctuation marks can denote what the writer feels, we have also done an analysis on the number of the 10 most commonly occurring punctuation marks in each class, divided by the train and test sets. We can see that the period is exponentially more common than any other punctuation mark, followed by the comma and the apostrophe, which do not offer any additional information as to the reviewer's sentiment. As they will not help in the classification and will otherwise clutter the dataset, we have elected to remove all punctuation marks.

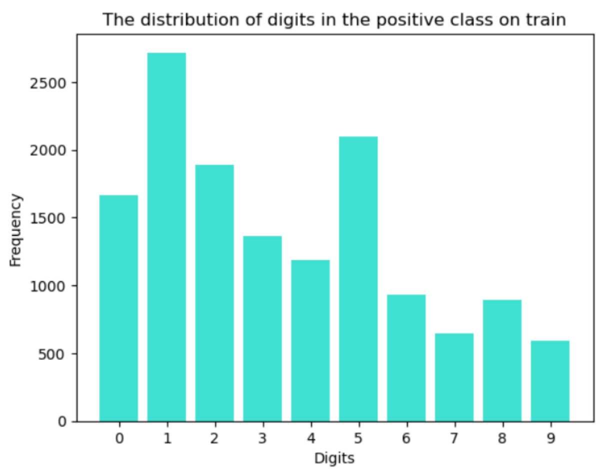
WordCloud

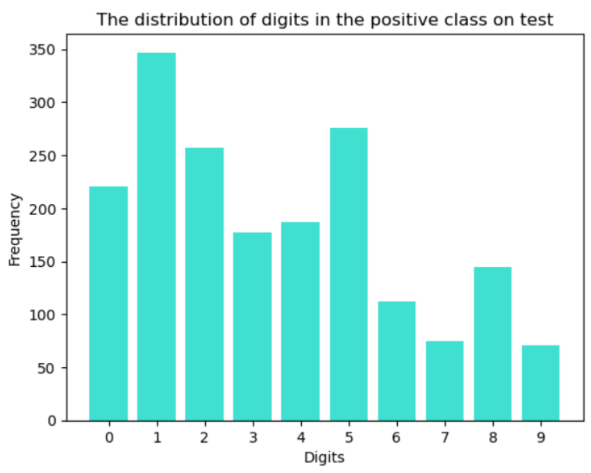


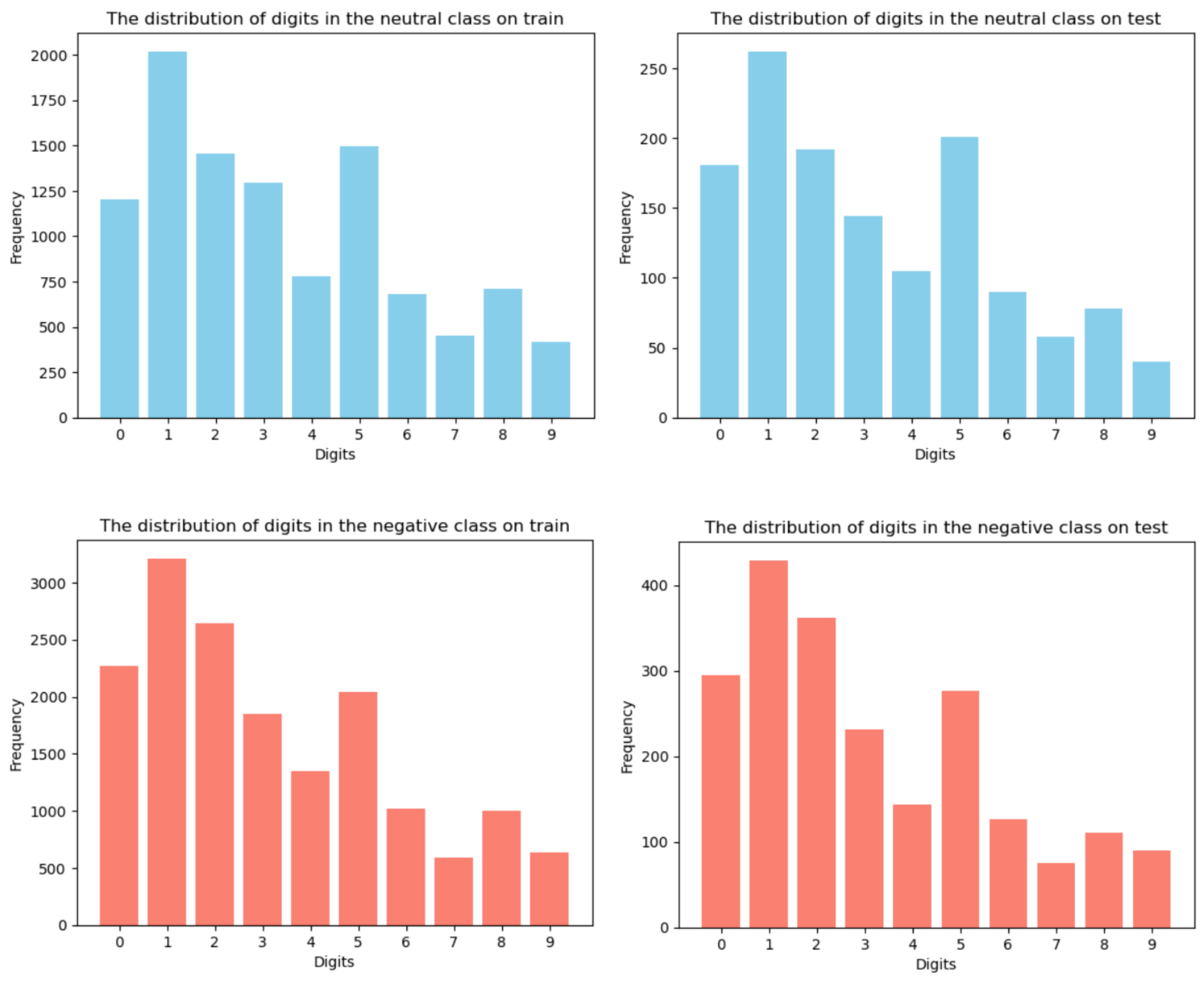
Wordclouds for the train(left) and test(right) data

The wordcloud illustrates the most common words found in the reviews. As the dataset contains reviews of fashion and beauty items, most of the words found above have to do with the fit of the item and other characteristics, such as its colour, alongside adjectives describing the person's opinion of the product they purchased.

Distribution of digits



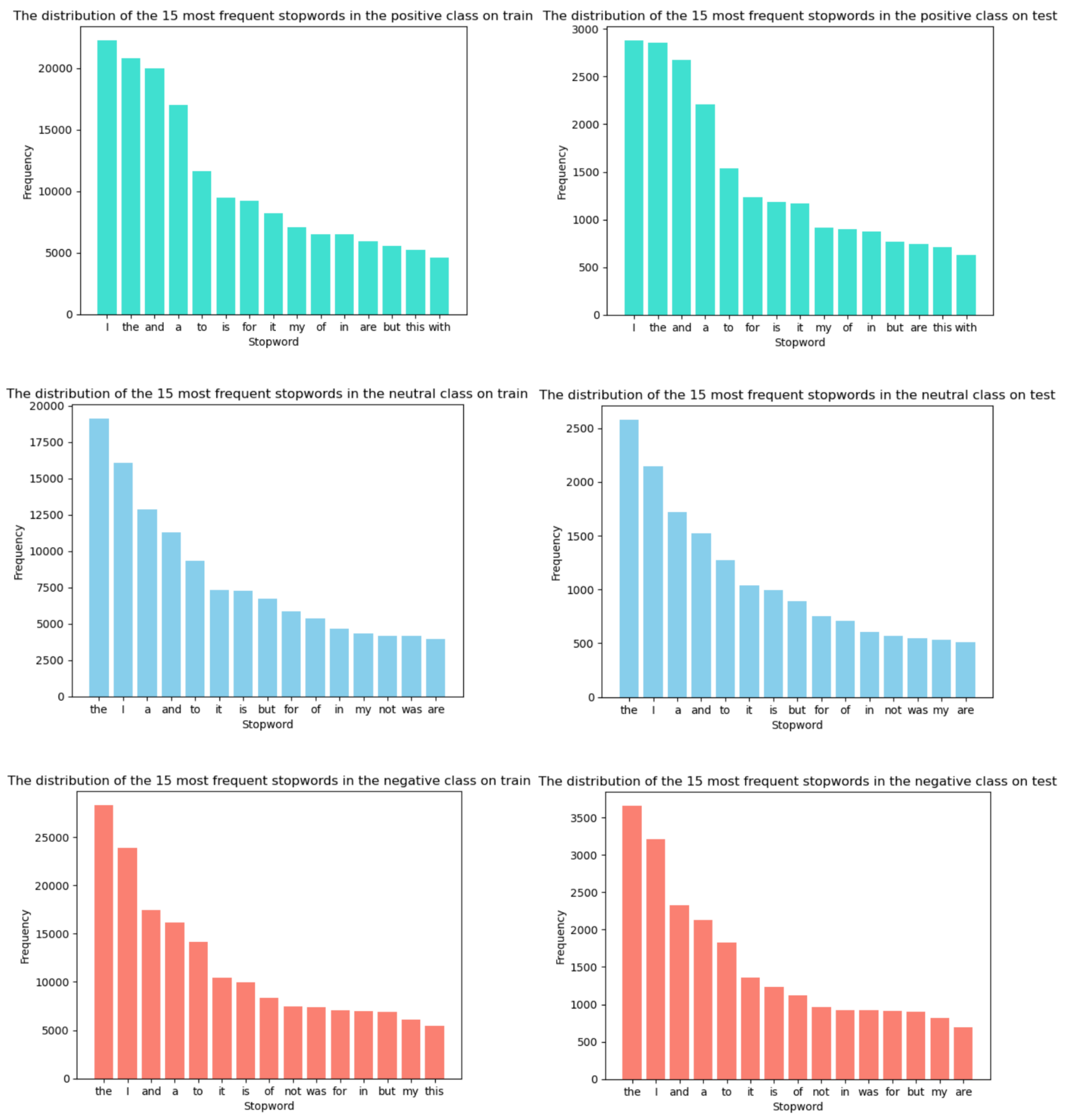




`Digit distribution divided by labels on the train(left) and test(right) data`

As can be seen from the histograms above, the distribution of digits by text class tells us nothing. 1 and 5 appear several times as opposed to 7 and 9 in all categories. So, it would be best to remove the digits in the preprocessing stage of the text as they do not give us any extra relevant information, help to reduce the size of the data and remove noise, creating a cleaner dataset.

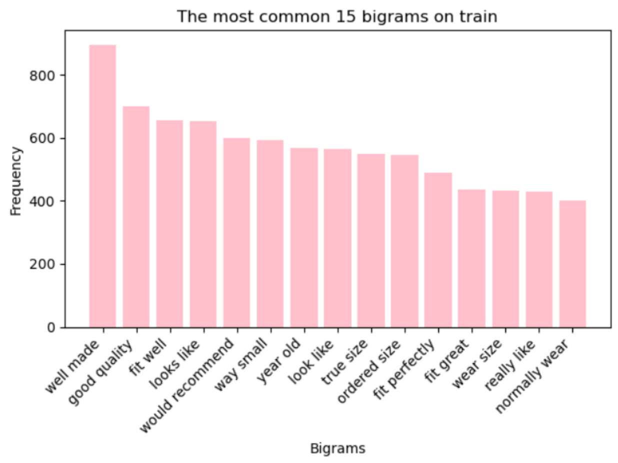
Stopwords

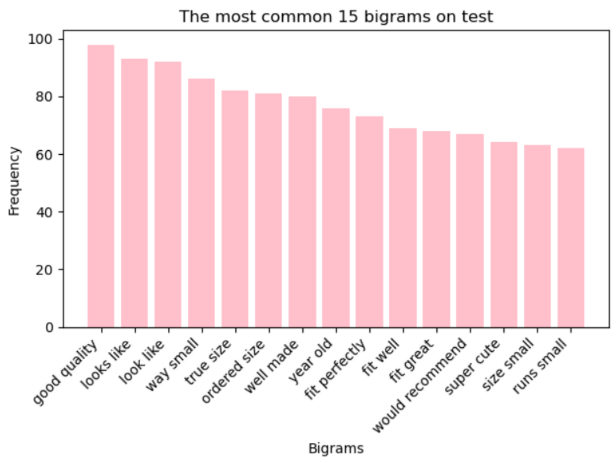


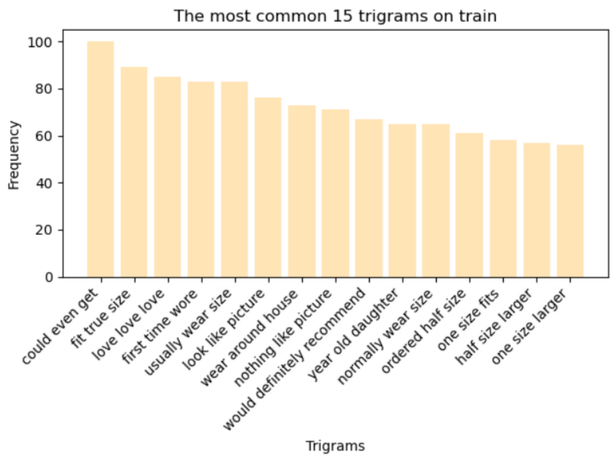
`Stopword distribution divided by labels on the train(left) and test(right) data`

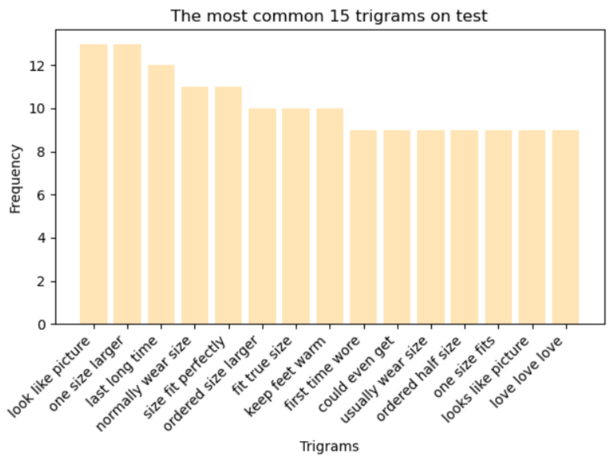
Exploratory analysis of the data indicates that the 15 most common stopwords do not show significant variation between our classes. Removing them in the text preprocessing stage may be beneficial for simplifying the dataset, allowing the models to focus on the really important words.

Bigrams/trigrams



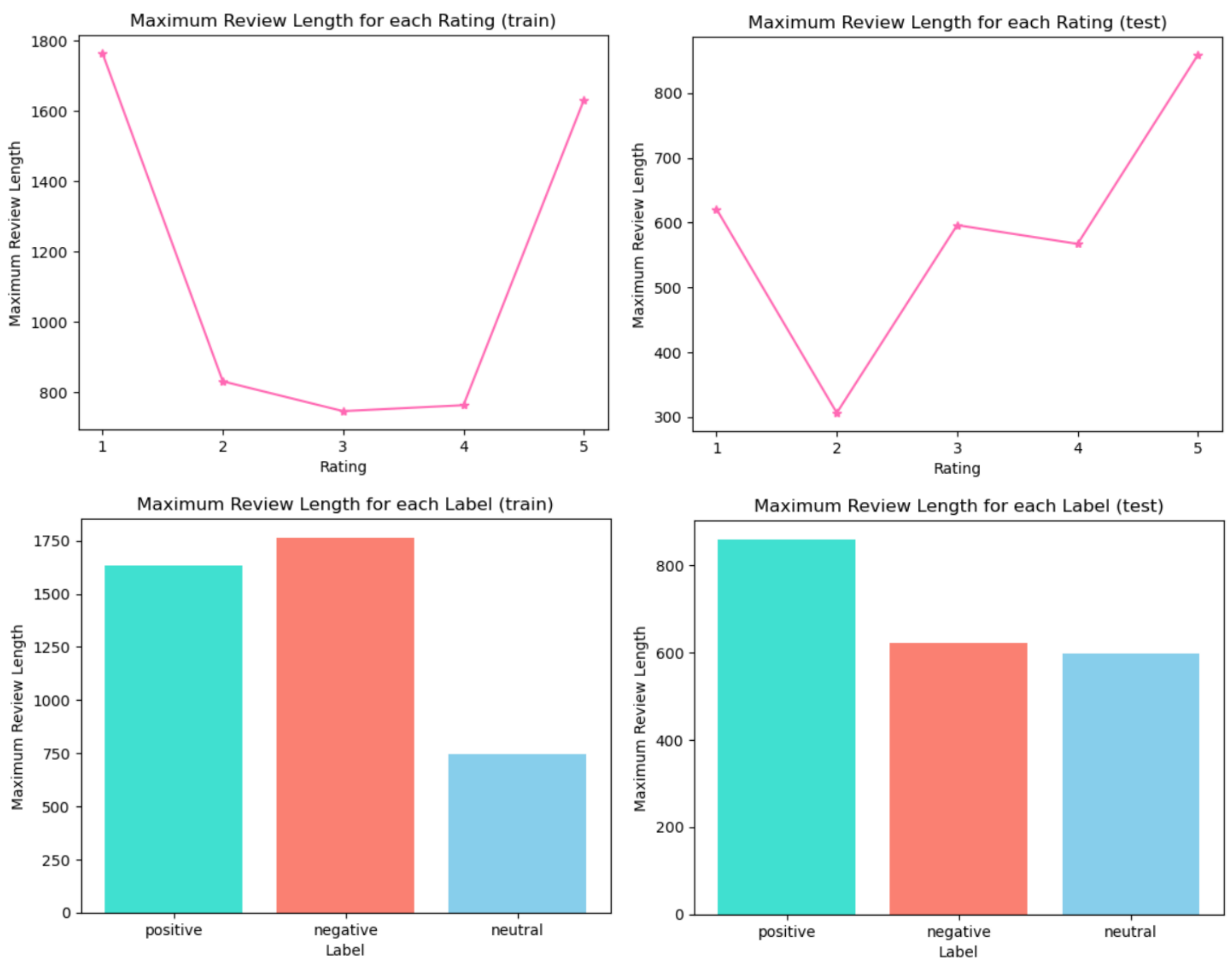






N-grams are consecutive groups of n words in a text. For our task, we analysed bigrams (n = 2) and trigrams (n = 3). These histograms help to understand the context in which certain words appear and reveal information that can lead to new insights. For example, the fact that "look like" and "looks like" appear so many times in the bigram diagram would suggest that it is good to test lemmatization in the preprocessing stage. This is bringing the words into dictionary form.

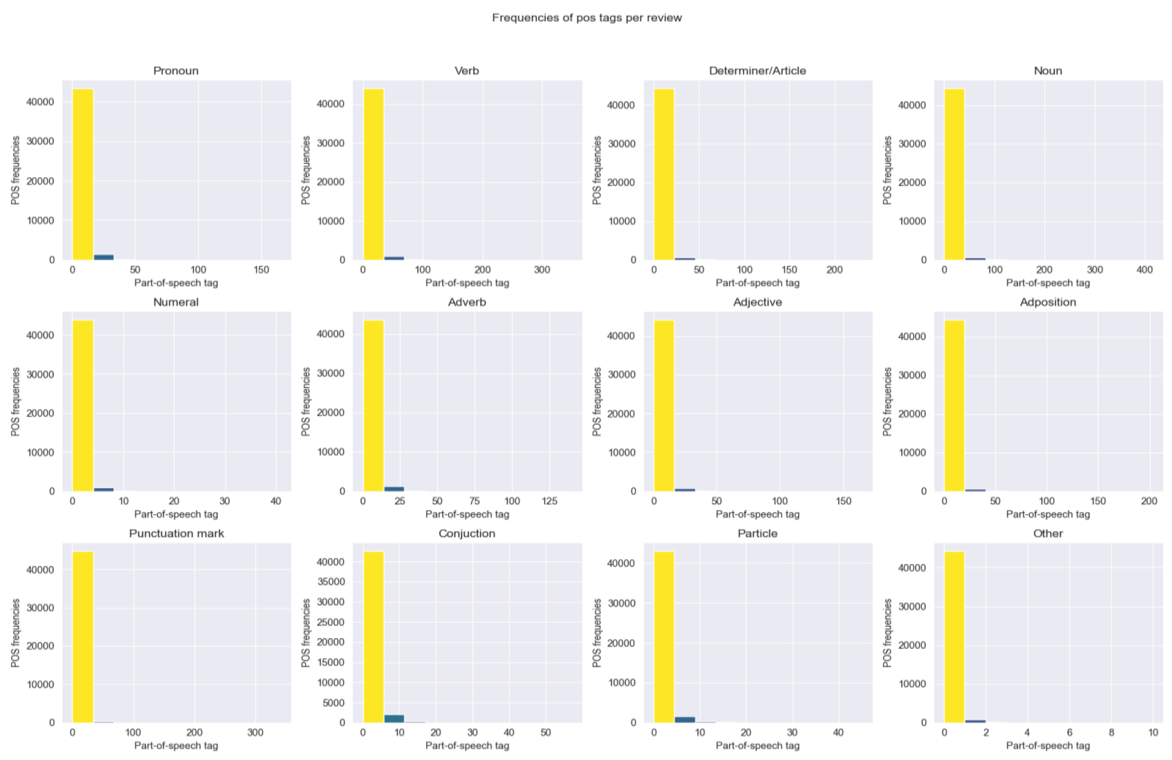
Maximum review length

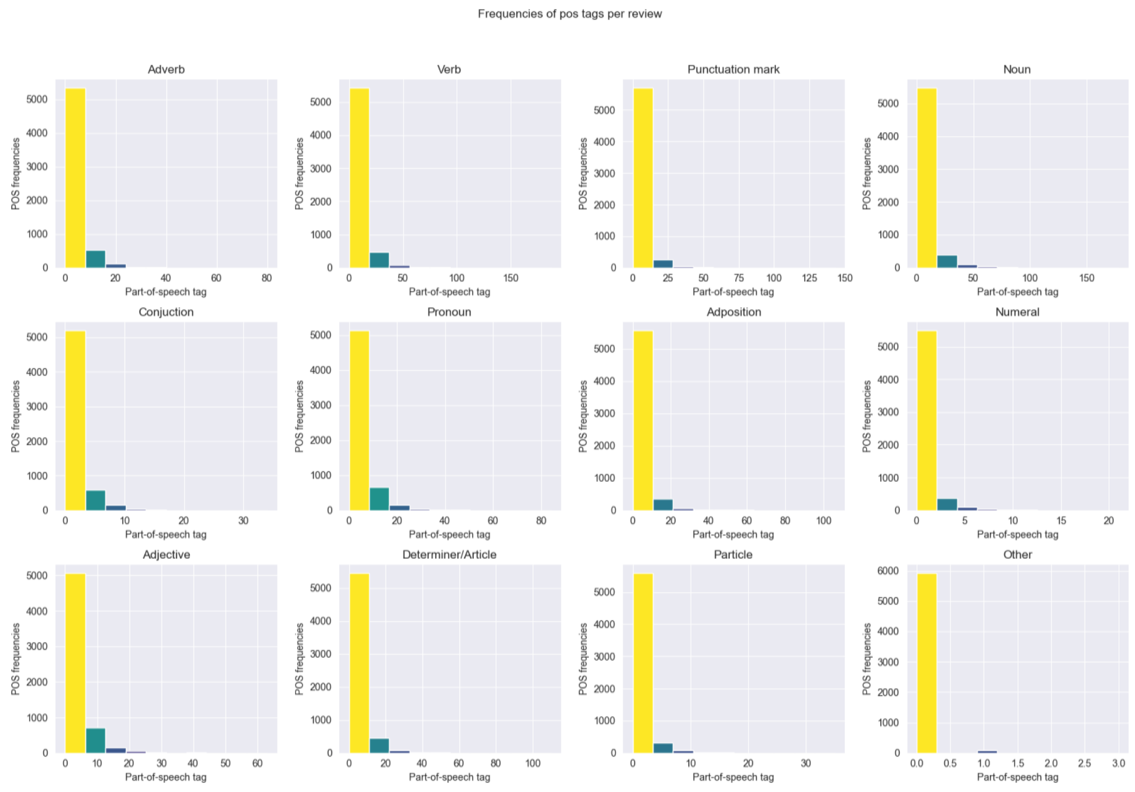


Review length divided by labels on the train(left) and test(right) data

To observe how the data is distributed and whether there is any correlation between the length of reviews and their respective class or rating, we analysed the training set and found that there is an inverse relationship: as the score decreases, the length of the review tends to increase. However, within the test set, no correlation is evident. Therefore, the length of the review is irrelevant and we don't consider this in training the models.

Part of speech





`POS frequency divided by labels on the train(top) and test(bottom) data`

Another group of features we can inspect in data is part-of-speech tagging, aka word-classes or lexical categories.

Text tagging using NLTK involves assigning labels or tags to different parts of a given text. In simpler terms, it's like labelling or categorising words in a sentence based on their grammatical or semantic roles. For example, NLTK can tag words as nouns, verbs, adjectives, etc., providing information about the word's function in the context of the sentence. This process helps in understanding the structure and meaning of the text, making it easier for computers to analyse and process natural language.

In the histograms shown above, we can observe that the parts of speech that are most common are: nouns, adjectives and adverbs.

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.

4. Features representation

For feature representation we used Term Frequency-Inverse Document Frequency (TF-IDF). This way of representation is based on the frequency of words in a text and their distribution in the entire text corpus.

$$TF-IDF = TF \times IDF$$

$$TF(w,t) = \frac{number\ of\ occurrences\ of\ word\ w\ in\ text\ t}{number\ of\ words\ in\ text\ t}$$

$$IDF(w,T) = log(\frac{number\ of\ texts\ in\ corpus\ T}{number\ of\ texts\ containing\ the\ word\ w} + 1)$$

When adjusting hyperparameters we also tested max_features and ngram_range (TF-IDF parameters):

- max_features: the number of terms with which the vocabulary will be built, taken according to the frequency of terms in the corpus ∈ {3000, 5000};
- ngram_range: n-gram type considered during TF-IDF transformation ∈ {(1, 1), (1, 2), (1, 3)}.

5. Models

We tested the following algorithms for classification the reviews: K-nearest neighbours, Logistic Regression, Random Forest Classifier and Neural Network. The best results of these algorithms can be seen in the table below:

Model	**Parameters**	**Best accuracy**
K-nearest neighbours	max_features = 5000; ngram_range = (1, 2); n_neighbors = 9; weights = “uniform”; metric = “cosine”	0.5925
Logistic Regression	max_features = 3000; ngram_range = (1, 3); solver = “lbfgs”; tol = 3; C = 0.3	0.7045
Random Forest Classifier	max_features = 3000; ngram_range = (1, 2); criterion = “gini”; n_estimators = 100	0.6765
Neural Network	max_features = 5000; ngram_range = (1, 3); no_of_layers = 1; no_of_units = 64; learning_rate = 0.0001	0.7124

K-nearest neighbours

K-nearest neighbours (KNN) is a machine learning algorithm based on the fact that similar objects are in the neighbourhood of each other in feature space.

To determine the class of a new object, the most frequent class among the k nearest neighbours of the object is calculated.

In hyperparameter tuning, we modified the following:

- n_neighbors: This parameter specifies the number of nearest neighbours to consider when making a prediction for a new object. n_neighbors $\in$ `{5, 7, 9};`
- weights: This parameter controls how neighbour contributions are taken into account: uniform - All neighbours have the same influence; distance - Closer neighbours have more influence. weights $\in$ `{“uniform”, “distance”};`

- metric: Metric for distance computation. metric ∈ {“`cosine`”, “`euclidean`”, “`minkowski`”}.

max_features	**ngram_range**	**n_neighbors**	**weights**	**metric**	**accuracy**
3000	(1, 1)	5	uniform	cosine	0.5746
3000	(1, 1)	5	uniform	euclidean	0.4435
3000	(1, 1)	5	uniform	manhattan	0.3883
3000	(1, 1)	5	distance	cosine	0.5711
3000	(1, 1)	5	distance	euclidean	0.4438
3000	(1, 1)	5	distance	manhattan	0.3873
3000	(1, 1)	7	uniform	cosine	0.5875
3000	(1, 1)	7	uniform	euclidean	0.438
3000	(1, 1)	7	uniform	manhattan	0.3866
3000	(1, 1)	7	distance	cosine	0.5815
3000	(1, 1)	7	distance	euclidean	0.4365
3000	(1, 1)	7	distance	manhattan	0.3861
3000	(1, 1)	9	uniform	cosine	0.5906
3000	(1, 1)	9	uniform	euclidean	0.4808
3000	(1, 1)	9	uniform	manhattan	0.4015
3000	(1, 1)	9	distance	cosine	0.5883
3000	(1, 1)	9	distance	euclidean	0.4818
3000	(1, 1)	9	distance	manhattan	0.3998
3000	(1, 2)	5	uniform	cosine	0.5725
3000	(1, 2)	5	uniform	euclidean	0.4488
3000	(1, 2)	5	uniform	manhattan	0.3898

3000	(1, 2)	5	distance	cosine	0.5701
3000	(1, 2)	5	distance	euclidean	0.4498
3000	(1, 2)	5	distance	manhattan	0.39
3000	(1, 2)	7	uniform	cosine	0.5783
3000	(1, 2)	7	uniform	euclidean	0.448
3000	(1, 2)	7	uniform	manhattan	0.388
3000	(1, 2)	7	distance	cosine	0.5771
3000	(1, 2)	7	distance	euclidean	0.4475
3000	(1, 2)	7	distance	manhattan	0.3885
3000	(1, 2)	9	uniform	cosine	0.5863
3000	(1, 2)	9	uniform	euclidean	0.4883
3000	(1, 2)	9	uniform	manhattan	0.3978
3000	(1, 2)	9	distance	cosine	0.5875
3000	(1, 2)	9	distance	euclidean	0.4903
3000	(1, 2)	9	distance	manhattan	0.398
3000	(1, 3)	5	uniform	cosine	0.575
3000	(1, 3)	5	uniform	euclidean	0.4495
3000	(1, 3)	5	uniform	manhattan	0.3905
3000	(1, 3)	5	distance	cosine	0.5693
3000	(1, 3)	5	distance	euclidean	0.4503
3000	(1, 3)	5	distance	manhattan	0.3908
3000	(1, 3)	7	uniform	cosine	0.5776
3000	(1, 3)	7	uniform	euclidean	0.448
3000	(1, 3)	7	uniform	manhattan	0.3893
3000	(1, 3)	7	distance	cosine	0.5758
3000	(1, 3)	7	distance	euclidean	0.4475
3000	(1, 3)	7	distance	manhattan	0.3898

3000	(1, 3)	9	uniform	cosine	0.583
3000	(1, 3)	9	uniform	euclidean	0.4916
3000	(1, 3)	9	uniform	manhattan	0.3983
3000	(1, 3)	9	distance	cosine	0.5848
3000	(1, 3)	9	distance	euclidean	0.4925
3000	(1, 3)	9	distance	manhattan	0.3985
5000	(1, 1)	5	uniform	cosine	0.5693
5000	(1, 1)	5	uniform	euclidean	0.4375
5000	(1, 1)	5	uniform	manhattan	0.3895
5000	(1, 1)	5	distance	cosine	0.5658
5000	(1, 1)	5	distance	euclidean	0.4385
5000	(1, 1)	5	distance	manhattan	0.3885
5000	(1, 1)	7	uniform	cosine	0.587
5000	(1, 1)	7	uniform	euclidean	0.4351
5000	(1, 1)	7	uniform	manhattan	0.3871
5000	(1, 1)	7	distance	cosine	0.5815
5000	(1, 1)	7	distance	euclidean	0.4338
5000	(1, 1)	7	distance	manhattan	0.3866
5000	(1, 1)	9	uniform	cosine	0.5911
5000	(1, 1)	9	uniform	euclidean	0.4735
5000	(1, 1)	9	uniform	manhattan	0.3998
5000	(1, 1)	9	distance	cosine	0.5873
5000	(1, 1)	9	distance	euclidean	0.4738
5000	(1, 1)	9	distance	manhattan	0.3985
5000	(1, 2)	5	uniform	cosine	0.5791
5000	(1, 2)	5	uniform	euclidean	0.4418
5000	(1, 2)	5	uniform	manhattan	0.3876

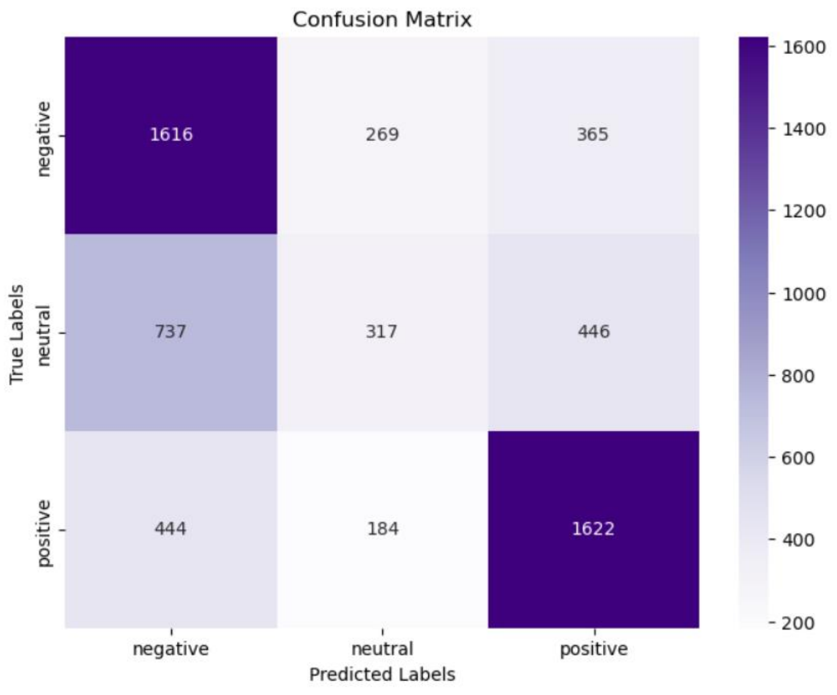
5000	(1, 2)	5	distance	cosine	0.575
5000	(1, 2)	5	distance	euclidean	0.4385
5000	(1, 2)	5	distance	manhattan	0.3873
5000	(1, 2)	7	uniform	cosine	0.591
5000	(1, 2)	7	uniform	euclidean	0.4351
5000	(1, 2)	7	uniform	manhattan	0.3868
5000	(1, 2)	7	distance	cosine	0.5893
5000	(1, 2)	7	distance	euclidean	0.436
5000	(1, 2)	7	distance	manhattan	0.387
5000	(1, 2)	9	uniform	cosine	0.5925
5000	(1, 2)	9	uniform	euclidean	0.4798
5000	(1, 2)	9	uniform	manhattan	0.39
5000	(1, 2)	9	distance	cosine	0.592
5000	(1, 2)	9	distance	euclidean	0.4786
5000	(1, 2)	9	distance	manhattan	0.3893
5000	(1, 3)	5	uniform	cosine	0.5768
5000	(1, 3)	5	uniform	euclidean	0.441
5000	(1, 3)	5	uniform	manhattan	0.3878
5000	(1, 3)	5	distance	cosine	0.5713
5000	(1, 3)	5	distance	euclidean	0.4378
5000	(1, 3)	5	distance	manhattan	0.3875
5000	(1, 3)	7	uniform	cosine	0.5875
5000	(1, 3)	7	uniform	euclidean	0.4348
5000	(1, 3)	7	uniform	manhattan	0.3868
5000	(1, 3)	7	distance	cosine	0.5856
5000	(1, 3)	7	distance	euclidean	0.4346
5000	(1, 3)	7	distance	manhattan	0.387

5000	(1, 3)	9	uniform	cosine	0.5903
5000	(1, 3)	9	uniform	euclidean	0.4861
5000	(1, 3)	9	uniform	manhattan	0.3906
5000	(1, 3)	9	distance	cosine	0.5866
5000	(1, 3)	9	distance	euclidean	0.484
5000	(1, 3)	9	distance	manhattan	0.39

The above table shows that the best accuracy was obtained for max_features = 5000, ngram_range = (1, 2), n_neighbors = 9, weights = "distance" and metric = "cosine".

Classification report and confusion matrix for this model:

```
              precision    recall  f1-score   support

    negative       0.58      0.72      0.64      2250
     neutral       0.41      0.21      0.28      1500
    positive       0.67      0.71      0.69      2250

    accuracy                           0.59      6000
   macro avg       0.55      0.55      0.54      6000
weighted avg       0.57      0.59      0.57      6000
```



From these reports, it can be seen that the worst results were obtained on the neutral class.

Logistic Regression

Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict the probability of an instance belonging to a particular class. Despite its name, logistic regression is employed for classification, not regression.

In logistic regression, the algorithm models the relationship between a dependent variable (which is binary, taking on two possible outcomes) and one or more independent variables. The logistic function, also known as the sigmoid function, is applied to the linear combination of these independent variables, transforming the output into a range between 0 and 1. The result can be interpreted as the probability of the instance belonging to the positive class.

Logistic regression often performs well when the relationship between features and the target variable is approximately linear.

In hyperparameter tuning, we modified the following:

- Solver: This parameter specifies the optimization algorithm used to find the coefficients that minimise the logistic loss function. solver ∈ {'lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'}
- Tol: Tolerance for stopping criteria. tol ∈ {5, 0.3, 3};
- C: This parameter is the regularisation strength, also known as the inverse of regularisation strength. It controls the amount of regularisation applied to the model. Like in support vector machines, smaller values specify stronger regularisation.
  C∈ {5, 0.3, 3}.

Because the hyperparameter tuning table would place too much in the document, we extract some of the best results in the next table. All the data can be found here.

max_features	ngram_range	solver	tol	C	accuracy

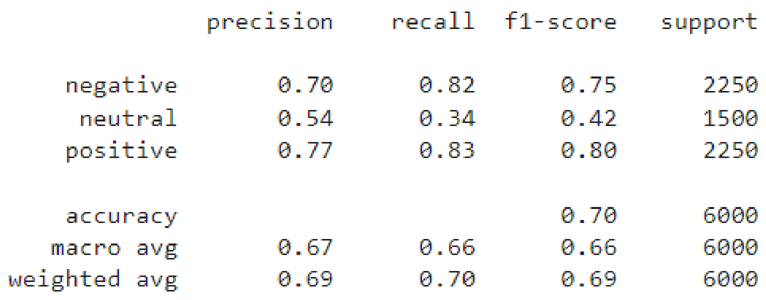
5000	(1, 2)	lbfgs	5	0.1	0.6942
3000	(1, 3)	lbfgs	5	0.3	0.7045
3000	(1, 2)	lbfgs	5	3	0.7022
3000	(1, 2)	lbfgs	5	10	0.701
5000	(1, 2)	lbfgs	0.3	0.1	0.6938
5000	(1, 2)	lbfgs	0.3	0.3	0.7038
5000	(1, 2)	lbfgs	0.3	3	0.702
3000	(1, 2)	lbfgs	0.3	10	0.701
3000	(1, 3)	lbfgs	3	0.1	0.6937
3000	(1, 3)	lbfgs	3	0.3	0.7045
3000	(1, 2)	lbfgs	3	3	0.7022
3000	(1, 2)	lbfgs	3	10	0.701
3000	(1, 1)	liblinear	5	0.1	0.375
3000	(1, 1)	liblinear	5	0.3	0.375
3000	(1, 1)	liblinear	5	3	0.375
3000	(1, 1)	liblinear	5	10	0.375
3000	(1, 2)	liblinear	0.3	0.1	0.6605
5000	(1, 3)	liblinear	0.3	0.3	0.6703
5000	(1, 3)	liblinear	0.3	3	0.669
3000	(1, 3)	liblinear	0.3	10	0.6678
3000	(1, 1)	liblinear	3	0.1	0.375
3000	(1, 1)	liblinear	3	0.3	0.375
3000	(1, 1)	liblinear	3	3	0.375
3000	(1, 1)	liblinear	3	10	0.375
5000	(1, 3)	newton-cg	5	0.1	0.6935
5000	(1, 3)	newton-cg	5	0.3	0.704

3000	(1, 3)	newton-cg	5	3	0.7018
3000	(1, 2)	newton-cg	5	10	0.699
5000	(1, 2)	newton-cg	0.3	0.1	0.6938
5000	(1, 3)	newton-cg	0.3	0.3	0.704
5000	(1, 2)	newton-cg	0.3	3	0.7032
3000	(1, 2)	newton-cg	0.3	10	0.6997
5000	(1, 3)	newton-cg	3	0.1	0.6935
5000	(1, 3)	newton-cg	3	0.3	0.704
5000	(1, 2)	newton-cg	3	3	0.7027
3000	(1, 3)	newton-cg	3	10	0.7
3000	(1, 1)	newton-cholesky	5	0.1	0.6913
5000	(1, 2)	newton-cholesky	5	0.3	0.7038
5000	(1, 2)	newton-cholesky	5	3	0.7032
3000	(1, 2)	newton-cholesky	5	10	0.7002
3000	(1, 1)	newton-cholesky	0.3	0.1	0.6913
5000	(1, 2)	newton-cholesky	0.3	0.3	0.7038
5000	(1, 2)	newton-cholesky	0.3	3	0.7032
3000	(1, 2)	newton-cholesky	0.3	10	0.7002
3000	(1, 1)	newton-cholesky	3	0.1	0.6913
5000	(1, 2)	newton-cholesky	3	0.3	0.7038
5000	(1, 2)	newton-cholesky	3	3	0.7032
3000	(1, 2)	newton-cholesky	3	10	0.7002
3000	(1, 3)	sag	5	0.1	0.5752
5000	(1, 2)	sag	5	0.3	0.661
5000	(1, 3)	sag	5	3	0.6487
3000	(1, 3)	sag	5	10	0.6477

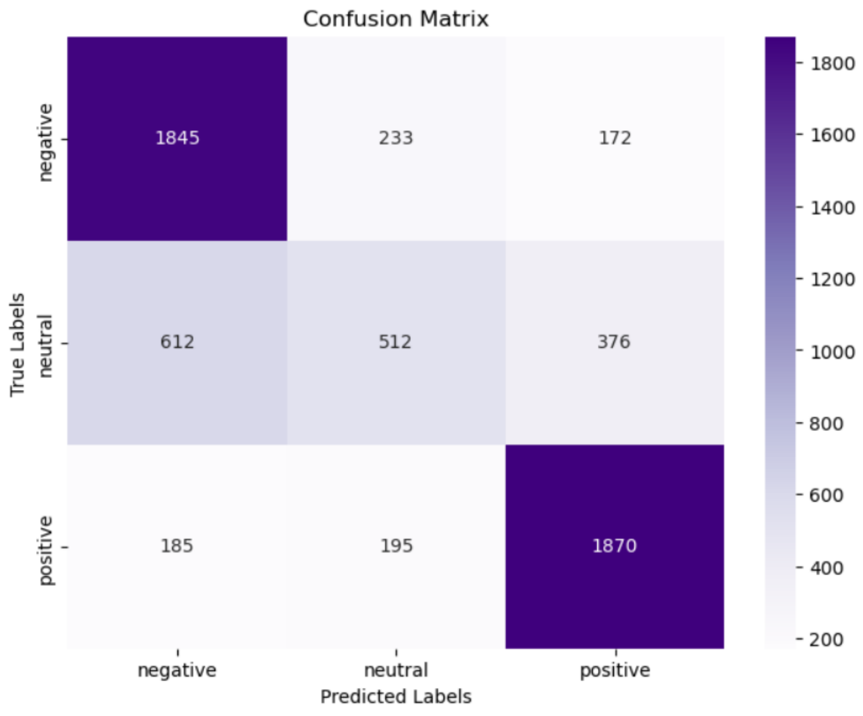
5000	(1, 3)	sag	0.3	0.1	0.6758
3000	(1, 3)	sag	0.3	0.3	0.693
5000	(1, 1)	sag	0.3	3	0.6925
5000	(1, 2)	sag	0.3	10	0.6818
5000	(1, 2)	sag	3	0.1	0.6833
5000	(1, 3)	sag	3	0.3	0.6807
5000	(1, 1)	sag	3	3	0.653
5000	(1, 3)	sag	3	10	0.6565
3000	(1, 3)	saga	5	0.1	0.684
3000	(1, 2)	saga	5	0.3	0.6922
3000	(1, 2)	saga	5	3	0.6877
5000	(1, 3)	saga	5	10	0.6775
3000	(1, 1)	saga	0.3	0.1	0.6958
5000	(1, 3)	saga	0.3	0.3	0.7033
5000	(1, 3)	saga	0.3	3	0.6945
3000	(1, 1)	saga	0.3	10	0.6873
3000	(1, 1)	saga	3	0.1	0.6873
3000	(1, 2)	saga	3	0.3	0.68
5000	(1, 3)	saga	3	3	0.6807
3000	(1, 2)	saga	3	10	0.68

The above table shows that the best accuracy was obtained for max_features = 3000, solver = 'lbfgs', tol = 3, C = 0.3, n_gram = (1, 3).

Classification report and confusion matrix for this model:



```
              precision    recall  f1-score   support

    negative       0.70      0.82      0.75      2250
     neutral       0.54      0.34      0.42      1500
    positive       0.77      0.83      0.80      2250

    accuracy                           0.70      6000
   macro avg       0.67      0.66      0.66      6000
weighted avg       0.69      0.70      0.69      6000
```



As we can see in the graphs shown above, the worst results were obtained when classifying the neutral class.

Random Forest

A Random Forest Classifier uses multiple decision trees in its prediction. Each tree is trained on a different, random subset of the data and casts a ‘vote’ with its prediction of the class of the object. The final result is then established by majority vote.

The hyperparameters used for tuning are:

- n_estimators: The number of trees in the forest $\in \{10, 20, 100\}$;

- criterion: The criteria for choosing the split in the decision tree∈ {“`gini`”, “`entropy`”, “`log_loss`”}.

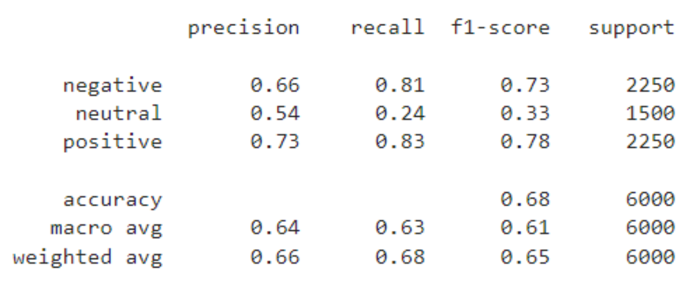
max_features	**ngram_range**	**criterion**	**n_estimators**	**accuracy**
1000	(1,1)	gini	10	0.6391
1000	(1,1)	gini	20	0.6546
1000	(1,1)	gini	100	0.6696
1000	(1,1)	entropy	10	0.6351
1000	(1,1)	entropy	20	0.6545
1000	(1,1)	entropy	100	0.6653
1000	(1,1)	log_loss	10	0.6421
1000	(1,1)	log_loss	20	0.655
1000	(1,1)	log_loss	100	0.6695
1000	(1,2)	gini	10	0.6406
1000	(1,2)	gini	20	0.6508
1000	(1,2)	gini	100	0.6703
1000	(1,2)	entropy	10	0.6336
1000	(1,2)	entropy	20	0.6525
1000	(1,2)	entropy	100	0.6673
1000	(1,2)	log_loss	10	0.6355
1000	(1,2)	log_loss	20	0.659
1000	(1,2)	log_loss	100	0.6681
1000	(1,3)	gini	10	0.6396
1000	(1,3)	gini	20	0.654

1000	(1,3)	gini	100	0.6698
1000	(1,3)	entropy	10	0.639
1000	(1,3)	entropy	20	0.6586
1000	(1,3)	entropy	100	0.6658
1000	(1,3)	log_loss	10	0.6491
1000	(1,3)	log_loss	20	0.6555
1000	(1,3)	log_loss	100	0.6655
3000	(1,1)	gini	10	0.6361
3000	(1,1)	gini	20	0.6558
3000	(1,1)	gini	100	0.6695
3000	(1,1)	entropy	10	0.6401
3000	(1,1)	entropy	20	0.6531
3000	(1,1)	entropy	100	0.669
3000	(1,1)	log_loss	10	0.641
3000	(1,1)	log_loss	20	0.6561
3000	(1,1)	log_loss	100	0.6683
3000	(1,2)	gini	10	0.644
3000	(1,2)	gini	20	0.666
3000	(1,2)	gini	100	0.6765
3000	(1,2)	entropy	10	0.6523
3000	(1,2)	entropy	20	0.663
3000	(1,2)	entropy	100	0.6668
3000	(1,2)	log_loss	10	0.6505
3000	(1,2)	log_loss	20	0.658

3000	(1,2)	log_loss	100	0.6721
3000	(1,3)	gini	10	0.6435
3000	(1,3)	gini	20	0.6568
3000	(1,3)	gini	100	0.6753
3000	(1,3)	entropy	10	0.6491
3000	(1,3)	entropy	20	0.6553
3000	(1,3)	entropy	100	0.6745
3000	(1,3)	log_loss	10	0.6443
3000	(1,3)	log_loss	20	0.6606
3000	(1,3)	log_loss	100	0.6715
5000	(1,1)	gini	10	0.6406
5000	(1,1)	gini	20	0.6486
5000	(1,1)	gini	100	0.6736
5000	(1,1)	entropy	10	0.6335
5000	(1,1)	entropy	20	0.6555
5000	(1,1)	entropy	100	0.6726
5000	(1,1)	log_loss	10	0.6378
5000	(1,1)	log_loss	20	0.6506
5000	(1,1)	log_loss	100	0.6675
5000	(1,2)	gini	10	0.6476
5000	(1,2)	gini	20	0.6593
5000	(1,2)	gini	100	0.668
5000	(1,2)	entropy	10	0.6413
5000	(1,2)	entropy	20	0.6663

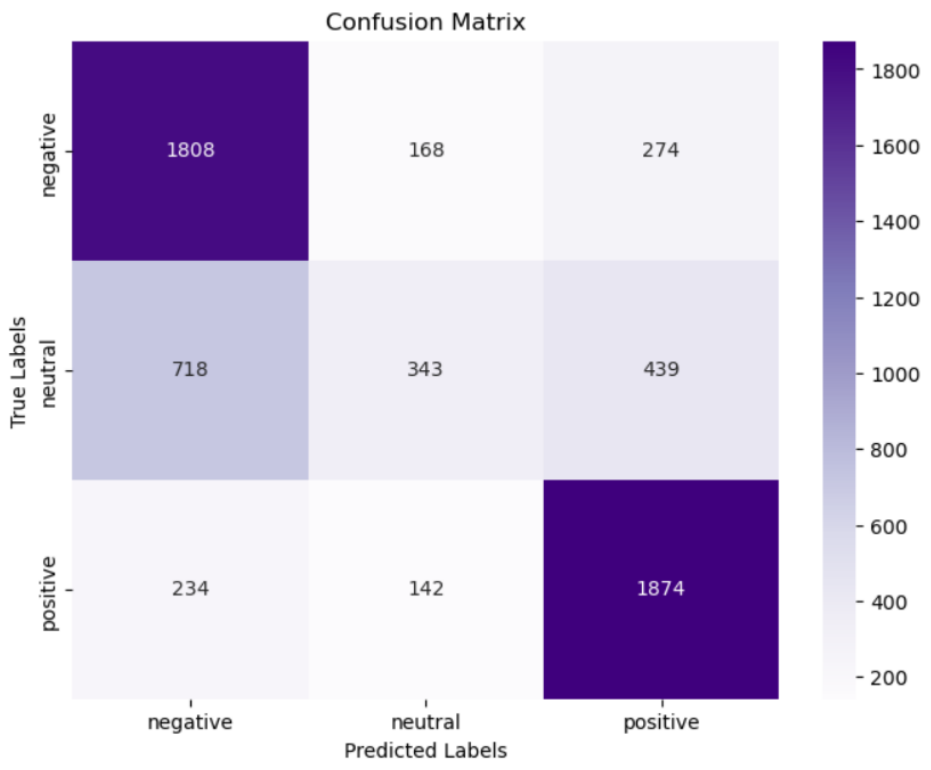
5000	(1,2)	entropy	100	0.6758
5000	(1,2)	log_loss	10	0.6403
5000	(1,2)	log_loss	20	0.6561
5000	(1,2)	log_loss	100	0.6726
5000	(1,3)	gini	10	0.6463
5000	(1,3)	gini	20	0.6608
5000	(1,3)	gini	100	0.6738
5000	(1,3)	entropy	10	0.6498
5000	(1,3)	entropy	20	0.6561
5000	(1,3)	entropy	100	0.6728
5000	(1,3)	log_loss	10	0.6451
5000	(1,3)	log_loss	20	0.6613
5000	(1,3)	log_loss	100	0.6698

The best performance of the Random Forest Classifier was with the gini criterion and 100 estimators alongside a Tf-Idf vectorizer with 3000 max_features and (1,2) n-gram range. For this configuration, the classification report is as follows:



```
              precision    recall  f1-score   support

    negative       0.66      0.81      0.73      2250
     neutral       0.54      0.24      0.33      1500
    positive       0.73      0.83      0.78      2250

    accuracy                           0.68      6000
   macro avg       0.64      0.63      0.61      6000
weighted avg       0.66      0.68      0.65      6000
```

And this was the resulting confusion matrix:



Neural Network

Neural networks attempt to mimic the way neurons are connected and communicate in the human brain. The basic building block of the network is therefore also called a neuron, which takes an input, processes it using a certain function, and produces an output. They are organised into an input layer, hidden layers, and an output layer. Information flows from the input layer through the hidden layers to the output layer.

For the neural network, we used Keras. Keras is the high-level API of the TensorFlow platform. It provides an approachable, highly-productive interface for solving machine learning (ML) problems, with a focus on modern deep learning.

Hyperparameter tuning for the neural network:

- no_of_layers: This parameter modifies the depth of the neural network. The number of

layers in a neural network is a crucial aspect of its architecture and can significantly impact its performance and ability to learn from data. No_of_layers $\in$ {1,2,3}

- no_of_units: Positive integer, dimensionality of the output space.

For the layers of the NN we used no_of_layers layers, all having the output space equal to no_of_units. The activation function that we used is ‘relu’.

ReLU formula is : f(x) = max(0,x)

ReLU is the most often used activation function in neural networks.

- learning_rate: is a hyperparameter that determines the size of the steps taken during the optimization process. It controls the amount by which the model's weights are updated during training. The learning rate is a critical parameter because it affects the convergence and performance of the neural network.

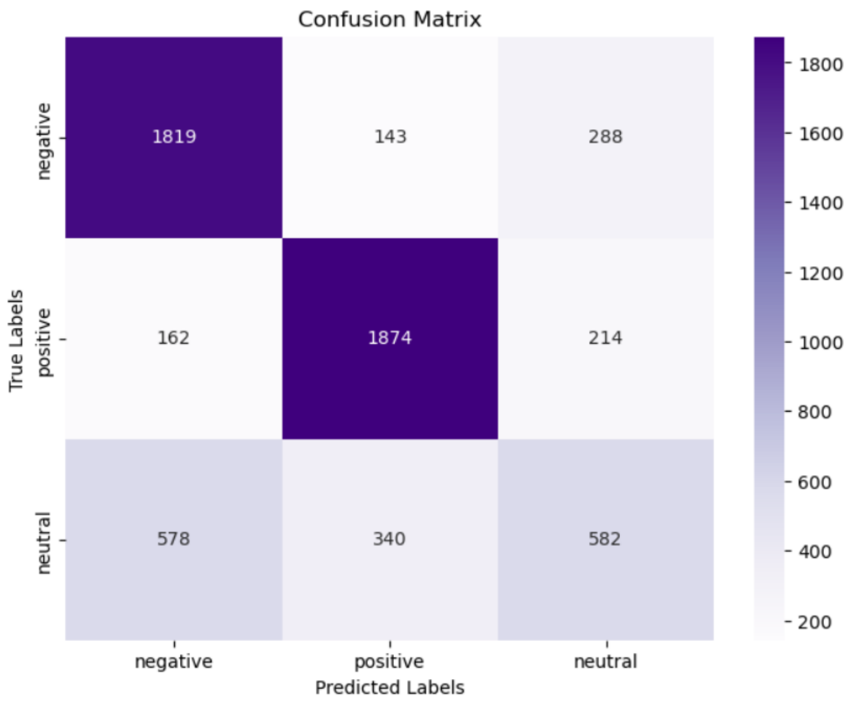
max_features	**ngram_range**	**no_of_layers**	**no_of_units**	**learning_rate**	**accuracy**
3000	(1, 1)	1	64	0.0001	0.7028
3000	(1, 1)	1	64	0.00001	0.6504
3000	(1, 1)	1	96	0.0001	0.7001
3000	(1, 1)	1	96	0.00001	0.6511
3000	(1, 1)	2	64	0.0001	0.6956
3000	(1, 1)	2	64	0.00001	0.6598
3000	(1, 1)	2	96	0.0001	0.6956
3000	(1, 1)	2	96	0.00001	0.6701
3000	(1, 1)	3	64	0.0001	0.6923
3000	(1, 1)	3	64	0.00001	0.6790
3000	(1, 1)	3	96	0.0001	0.6818
3000	(1, 1)	3	96	0.00001	0.6875
3000	(1, 2)	1	64	0.0001	0.7093
3000	(1, 2)	1	64	0.00001	0.6506

3000	(1, 2)	1	96	0.0001	0.7098
3000	(1, 2)	1	96	0.00001	0.6541
3000	(1, 2)	2	64	0.0001	0.7026
3000	(1, 2)	2	64	0.00001	0.6583
3000	(1, 2)	2	96	0.0001	0.7001
3000	(1, 2)	2	96	0.00001	0.6729
3000	(1, 2)	3	64	0.0001	0.6961
3000	(1, 2)	3	64	0.00001	0.6818
3000	(1, 2)	3	96	0.0001	0.6888
3000	(1, 2)	3	96	0.00001	0.6891
3000	(1, 3)	1	64	0.0001	0.7103
3000	(1, 3)	1	64	0.00001	0.6506
3000	(1, 3)	1	96	0.0001	0.7120
3000	(1, 3)	1	96	0.00001	0.6510
3000	(1, 3)	2	64	0.0001	0.7033
3000	(1, 3)	2	64	0.00001	0.6593
3000	(1, 3)	2	96	0.0001	0.7043
3000	(1, 3)	2	96	0.00001	0.6751
3000	(1, 3)	3	64	0.0001	0.6945
3000	(1, 3)	3	64	0.00001	0.6809
3000	(1, 3)	3	96	0.0001	0.6915
3000	(1, 3)	3	96	0.00001	0.6873
5000	(1, 1)	1	64	0.0001	0.7038
5000	(1, 1)	1	64	0.00001	0.6501
5000	(1, 1)	1	96	0.0001	0.6998
5000	(1, 1)	1	96	0.00001	0.6578
5000	(1, 1)	2	64	0.0001	0.6924
5000	(1, 1)	2	64	0.00001	0.6568

5000	(1, 1)	2	96	0.0001	0.6844
5000	(1, 1)	2	96	0.00001	0.6748
5000	(1, 1)	3	64	0.0001	0.6784
5000	(1, 1)	3	64	0.00001	0.6733
5000	(1, 1)	3	96	0.0001	0.6668
5000	(1, 1)	3	96	0.00001	0.6821
5000	(1, 2)	1	64	0.0001	0.7116
5000	(1, 2)	1	64	0.00001	0.6561
5000	(1, 2)	1	96	0.0001	0.7073
5000	(1, 2)	1	96	0.00001	0.6561
5000	(1, 2)	2	64	0.0001	0.6973
5000	(1, 2)	2	64	0.00001	0.6668
5000	(1, 2)	2	96	0.0001	0.6926
5000	(1, 2)	2	96	0.00001	0.6855
5000	(1, 2)	3	64	0.0001	0.6863
5000	(1, 2)	3	64	0.00001	0.6901
5000	(1, 2)	3	96	0.0001	0.6685
5000	(1, 2)	3	96	0.00001	0.6911
5000	(1, 3)	1	64	0.0001	0.7124
5000	(1, 3)	1	64	0.00001	0.6499
5000	(1, 3)	1	96	0.0001	0.7099
5000	(1, 3)	1	96	0.00001	0.6591
5000	(1, 3)	2	64	0.0001	0.7001
5000	(1, 3)	2	64	0.00001	0.6648
5000	(1, 3)	2	96	0.0001	0.6931
5000	(1, 3)	2	96	0.00001	0.6821
5000	(1, 3)	3	64	0.0001	0.6875
5000	(1, 3)	3	64	0.00001	0.6809

5000	(1, 3)	3	96	0.0001	0.6743
5000	(1, 3)	3	96	0.00001	0.6906

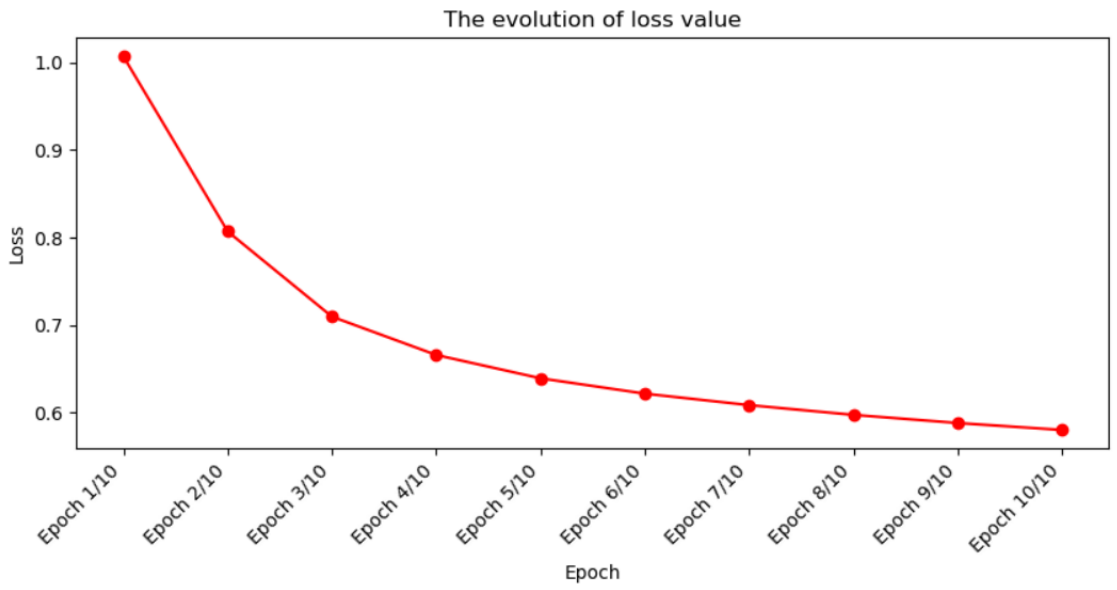
From the results it can be seen that learning_rate 0.0001 has better results than 0.00001. From the above table we observe that the highest accuracy was obtained for max_features = 5000, ngram_range = (1, 3), 1 layer, 64 units and learning_rate = 0.0001. Confusion matrix and classification report for this result:



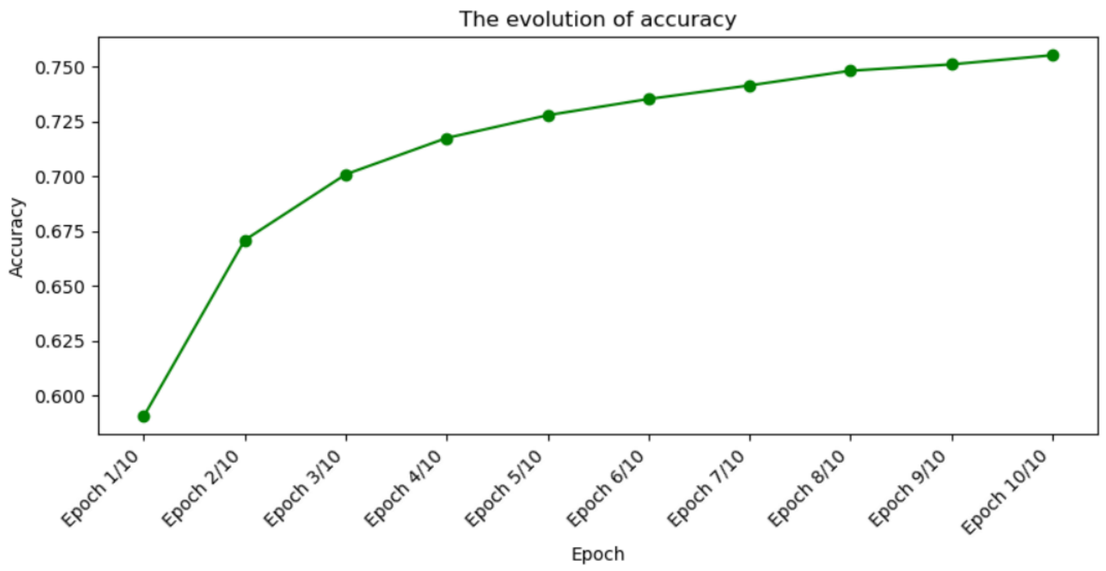
```
              precision    recall  f1-score   support

           0       0.71      0.81      0.76      2250
           1       0.80      0.83      0.81      2250
           2       0.54      0.39      0.45      1500

    accuracy                           0.71      6000
   macro avg       0.68      0.68      0.67      6000
weighted avg       0.70      0.71      0.70      6000
```

It can be seen that, similar to the other models, the neutral class is the most difficult to predict.

For this top accuracy we analyse the evolution of loss value and accuracy in the train stage:



Decreasing the loss value during training is something we want at this stage. Loss is a metric that measures the difference between model predictions and actual values. A decrease in loss indicates that the model is adjusting its parameters and making more accurate predictions. As can be seen from the plot above, the loss value decreases as the model goes through the epochs.



Increased accuracy during training is a sign of improved performance. This increase indicates that the model is becoming increasingly capable of making accurate predictions on the training data. This can be seen in the plot above.

6. Conclusion

Sentiment classification is a common and relevant problem in Natural Language Processing. This project has shown us the importance of studying the data we use in order to gain a better understanding of how to process and use it in any Machine Learning task. Due to a more in depth study of the dataset, we also learned why it is common practice to strip the text of things like digits, punctuation marks and and stopwords. Data preprocessing helps us eliminate factors that could negatively affect the performance of the models.

We have noticed that neutral reviews are more difficult to identify for machine learning models, because these texts contain a wide variety of information without conveying a particular strong feeling. Of all the models we have tried, Neural Networks had the best performance with an accuracy of 0.7124, which could possibly perform even better with further testing on different feature representations and layer configurations.

7. References

Ni, J., Li, J., & McAuley, J. (2019). Empirical Methods in Natural Language Processing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, (Vol. 3, pp. 1-12). Empirical Methods in Natural Language Processing (EMNLP). https://www.aclweb.org/anthology/D19-1001/