University of Athens
Department of Informatics and Telecommunications

Deep Learning for NLP

Student name: ΟΡΕΣΤ ΜΟΥΤΣΑΙ
sdi: sdi1900120

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2023*

Contents

1. Abstract

The task is to develop a Sentiment Classifier using a Logistic Regression model and the Term Frequency-Inverse Document Frequency(TF-IDF) vectorization technique. The data set has been extracted from Twitter and contains three columns. The first column is an ID, the second column is a Text and the third column is a Label. In the Label 0 represents a negative sentiment, whereas 1 represents a positive sentiment. The task has been partitioned in three main sub-tasks. Firstly, the data will be preprocessed, then the TF-IDF technique will be optimized and at last, the Logistic Regression model will be tuned.

2. Data processing and analysis

2.1. Pre-processing

The preprocessing started by lower casing all text. Common abbreviations were expanded manually, for example "lol" was expanded to "laugh out loud". Contractions were expanded to their full form using the function fix() from the contractions library. URLs and mentions were removed using regular expressions, maintaining only alphanumeric characters and underscores. Emojis were removed using the function demojize() from the emoji library. Punctuations were removed using a couple functions from the string library. The NLTK library was used to perform Stemming on the data. The Porter Stemming Algorithm was selected over the Snowball algorithm due to it's higher accuracy.

Negation handling did not succeed in increasing accuracy and was opted out of the text preprocessing. Lemmatization techniques using Spacy or NLTK(with POS Tagging) reduced accuracy so they were not selected either. Removing Stop words also reduced model effectiveness.

Removing short text(sentences with less than two words) increased accuracy but was not integrated due to Kaggle's testing limitations.

2.2. Analysis

Before Preprocessing:

The size of the vocabulary in the training dataset is 167274 unique words out of total 2442621 words. The size of the vocabulary in the validation dataset is 66687 unique words out of total 699886 words. The size of the vocabulary in the test dataset is 40191 unique words out of total 351355 words.

Training Dataset	
Total Words	**Unique Words**
2442621	167274

Table 1: Training Data Set Word Count Before Preprocessing

Validation Dataset	
Total Words	**Unique Words**
699886	66687

Table 2: Validation Data Set Word Count Before Preprocessing

Testing Dataset	
Total Words	**Unique Words**
351355	40191

Table 3: Testing Data Set Word Count Before Preprocessing

The ten most common words in the training data set are:

Training Dataset	
Word	**Frequency**
!	85636
.	76885
@	74891
I	60523
to	51924
the	45883
,	45383
a	34452
i	27911
...	27135

Table 4: Training Data Set Most Common Words Before Preprocessing

The ten most common words in the validation data set are:

Validation Dataset	
Word	**Frequency**
!	24852
.	22536
@	21508
I	17400
to	14630
the	13073
,	12933
a	9877
i	8031
...	7758

Table 5: Validation Data Set Most Common Words Before Preprocessing

The ten most common words in the test data set are:

Testing Dataset	
Word	**Frequency**
!	12425
.	10983
@	10703
I	8501
to	7493
the	6665
,	6646
a	5002
...	3928
i	3898

Table 6: Testing Data Set Most Common Words Before Preprocessing

As expected, when stop words and punctuations are included, they occupy almost exclusively the list with the most common words in each data set. However, since removing all the stop words from the data sets decreases accuracy, it has been determined that it is better to include them. It is therefore concluded that while stop words may seem insignificant, they contribute to the overall semantic structure of the text. Punctuations on the other hand will be removed while cleaning the data.

Below are the Word Clouds (excluding stop words):

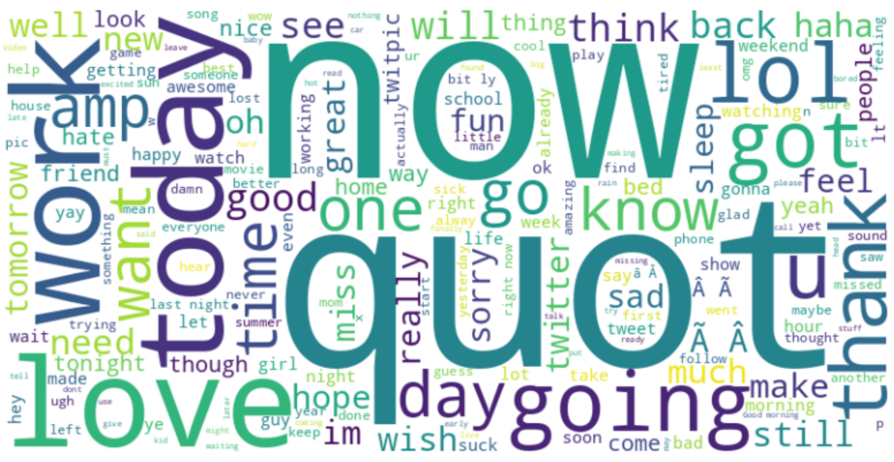


Figure 1: Training Dataset World Cloud Before Preprocessing

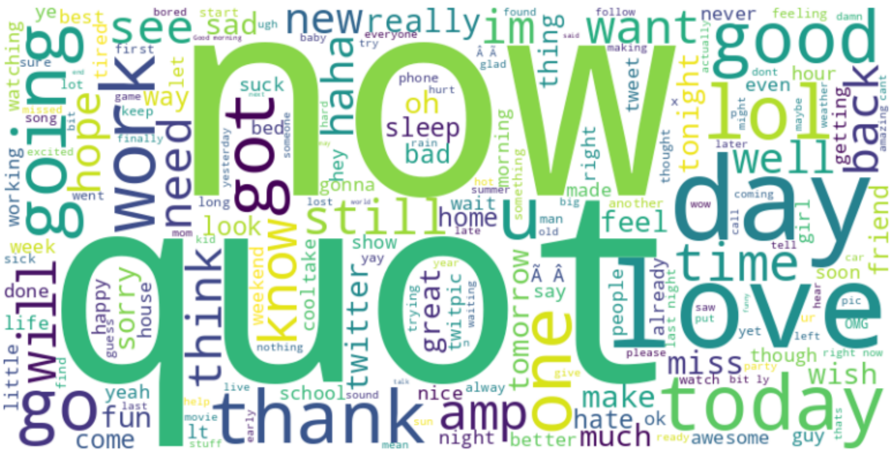


Figure 2: Validation Dataset World Cloud Before Preprocessing

now
quot
day
love
today
going
work
thank
good
know
time
lol
one
amp
want
still
got
back
go
will

Figure 3: Testing Dataset World Cloud Before Preprocessing

After Preprocessing:

The size of the vocabulary in the training data set is 5441 unique words out of total 201275 words. The size of the vocabulary in the validation data set is 25772 unique words out of total 576031 words. The size of the vocabulary in the test data set is 17120 unique words out of total 289224 words.

Training Dataset	
Total Words	**Unique Words**
2012754	54491

Table 7: Training Data Set Word Count After Preprocessing

Validation Dataset	
Total Words	**Unique Words**
576031	25772

Table 8: Validation Data Set Word Count After Preprocessing

Testing Dataset	
Total Words	**Unique Words**
289224	17120

Table 9: Testing Data Set Word Count After Preprocessing

The ten most common words in the training dataset are:

Training Dataset	
Word	**Frequency**
i	95736
to	60885
the	49815
is	36147
a	35850
you	33636
it	32843
not	32619
my	30431
and	28565

Table 10: Training Data Set Most Common Words After Preprocessing

The ten most common words in the validation dataset are:

Validation Dataset	
Word	**Frequency**
i	27599
to	17064
the	14248
is	10258
a	10249
you	9567
it	9476
not	9331
my	8853

Table 11: Validation Data Set Most Common Words After Preprocessing

The ten most common words in the testing dataset are:

Testing Dataset	
Word	**Frequency**
i	13464
to	8718
the	7205
a	5216
is	5176
you	4832
it	4761
not	4640
my	4457
and	4149

Table 12: Testing Data Set Most Common Words After Preprocessing

After preprocessing the data sets, the number of both total and unique words decreased significantly. Words and characters that did not add much to sentiment analysis were removed. Some words were stemmed, while all words were converted to their lowercase form.

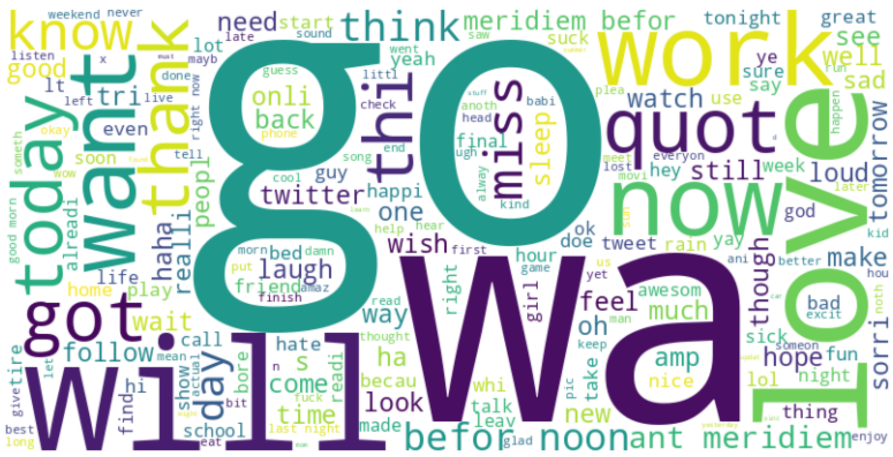


Figure 4: Training Dataset World Cloud After Preprocessing

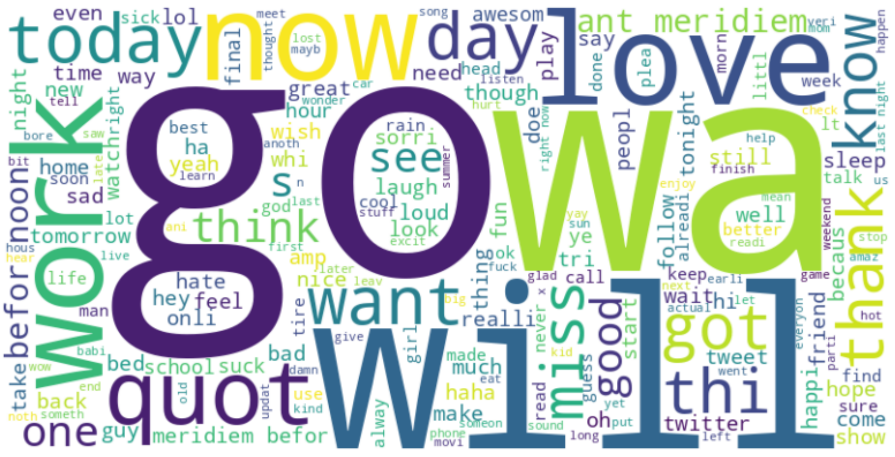


Figure 5: Validation Dataset World Cloud After Preprocessing

go
wa
wil
love
work
now
day
know
quot
thank
today
time
good
want
think
miss

Figure 6: Testing Dataset World Cloud After Preprocessing

2.3. Data partitioning for train, test and validation

The data for this assignment has already been partitioned by the instructors. The train dataset is used to train the model, the validation dataset is used to test the model and the test dataset is used for Kaggle's competition scoring.

2.4. Vectorization

Term Frequency-Inverse Document Frequency(TF-IDF) has been used for vectorization. The Term Frequency measures how often a words appears in a document. Meanwhile, the Inverse Document Frequency reduces the weight (significance) of the common words and at the same time increases the weight of rare words. To dampen the effect of frequently appearing words, a logarithmic scale is used. Below are the formulas:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \text{IDF}(t)$$

$$\text{TF}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

$$\text{IDF}(t) = \log\left(\frac{N}{n_t}\right)$$

3. Algorithms and Experiments

3.1. Experiments

The first brute-force run was executed without any preprocessing on the data sets, nor any hyperparameter tuning was performed. The results were surprisingly good achieving a score of 0.79090 on the validation data set. On the training data set, the score was 0.83619. The absolute difference (percentage) of the scores was 4.529%, neither great nor bad. After preprocessing the datasets, the validation accuracy stayed fairly similar. The training score however was reduced to around 81%, resulting in an absolute difference (percentage) score of approximately 2.5%. Nonetheless, preprocessing the data is a crucial step which was necessary for the increase of the accuracy when using the final model. The next step was the hypertuning of Term Frequency-Inverse Document Frequency (TF-IDF) algorithm. Both Grid Search and Random Search algorithms were used in order to configure what were deemed as the most important parameters in TF-IDF. This process took many hours, especially executing the Grid Search algorithm. The exact parameters optimized will be discussed below. Regarding the Logistic Regression, each parameter was configured using a Parameter Grid and a pipeline, along with manual adjustments. More info on the optimizations to be found below likewise. The LogisticRegression() function from the sklearn library is not gradient descent-based by default, so gradient descent was not performed. However, SGDClassifier() with the parameter loss='log_loss' was used as a test, providing very similar, albeit lower, results.

3.1.1. Table of trials.

Following is a table of the main experiments performed:

Trial	Technique/Experiment	Prerocessing	Validation Score	Testing Score
Trial 1	Default Parameters	No preprocessing	0.79090	0.78928
Trial 2	Added max_iter=1000	Applied Lowercasing	0.79085	0.78937
Trial 3	No Update	Expanded Abbreviations	0.79116	0.78895
Trial 4	No Update	Expanded Contractions	0.79118	0.79017
Trial 5	No Update	Handled Negations	0.79342	0.79003
Trial 6	No Update	Removed URLs & Mentions	0.79241	0.79098
Trial 7	No Update	Removed Emojis	0.79241	0.79098
Trial 8	No Update	Removed Punctuations	0.79243	0.79088
Trial 9	No Update	Applied Stemming using Porter	0.79057	0.78767
Trial 10	No Update	Replaced Porter with Snowball	0.79088	0.78899
Trial 11	Tuned the Hyper-Parameters of TF-IDF	Replaced Snowball with Porter	0.80460	0.80296
Trial 12	No Update	Tested without Stemming	0.80342	0.80296
Trial 13	No Update	Removed Negation Handling	0.80628	0.80371
Trial 14	Tuned the Hyper-Parameters of Logistic Regression	No Update	0.80640	0.80390

Table 13: Trials

3.2. Hyper-parameter tuning

In regards of hyperparameter tuning the Term Frequency-Inverse Document Frequency technique, lower casing the data was excluded since it was already done in the preprocessing step. However, tokenization was implemented using TF-IDF's in built tokenizer. Stop Words were excluded since they greatly reduced the accuracy. A max_features limit of 100000 was set after manual testing. The min_df and max_df parameters were found by a Grid Search algorithm. A Random Search algorithm was tested as well giving similar, albeit lower, results. The sublinear_tf, the use_idf and the smooth_idf parameters were set as true after executing the Grid Search algorithm. The ngram_range was set as (1,2) by the algorithm and was later confirmed after testing many parameters to be indeed the best setting. Trigrams reduced the accuracy by a small margin. No other parameter was used for the TF-IDF.

In Regards of hyperparameter tuning the Logistic Regression model, due to the time complexity of running one of the above algorithms for our data set, the regression model was tuned after tuning the TF-IDF model and not together. The max_iter parameter was set to 1000 in order to allow the model to converge in certain cases where it by default didn't. A pipeline was created along with a parameter grid which had the following parameter options. The model_C parameter had the options of 0.001, 0.01, 0.1, 1, 10 and 100. Setting C = 1 was found to be the better regularization strength which was later reduced to 0.95. The model_penalty parameter had the availability of options 'l1' and 'l2'. The models which were selected to be tested were 'liblinear' and 'saga' which are compatible with 'l1' and 'l2' regularization types. The 'saga' was detected to be extremely slow in this case and was scrapped. The rest solvers (along with their equivalent allowed penalty types) were tested one by one. In the end, the default 'lbfgs' solver was chosen. No penalty was selected so the default L2 regularization was used. The class_weight had the options 'None' or 'Balanced' from which 'None' seemed to perform better and was as a result selected.

While hyperparameter tuning the whole model, the overfitting was within the margin of 3-5%. In the end the training dataset had 0.85294 accuracy and the validation dataset had an accuracy score of 0.80640. The absolute difference (percentage) is 4.6543% so along with evidence from the graphs, the model does not seem to produce any significant overfit.

3.3. Optimization techniques

In the Logistic Regression model, the fast Limited-memory BFGS ('lbfgs') solver was used in conjunction with L2 (default) regularization, which uses the squared euclidean norm. Training the model is a matter of a couple minutes, so no further optimization was needed.

3.4. Evaluation

The scores used to evaluate the predictions include Accuracy, Precision, Recall, F1-Score, as well as macro and weighted averages. Additionally, the Log Loss function score and the Absolute Difference (Percentage) between the training and validation scores will be evaluated. Finally, the Area Under the Curve (AUC) will also be considered.

The Accuracy score measures how many of the predictions made by the Logistic Regression model are actually correct:

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

The Precision score measures how many of the positive predictions made by the model are actually correct:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

The Recall score measures how many of the positive instances were correctly identified by the model:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

It is worth noting that Precision and Recall are inversely related. Increasing one likely reduces the other.

The Absolute Difference (Percentage) between training score and validation score measures the gap between model performance on training data and validation data, in order to detect overfitting or underfitting:

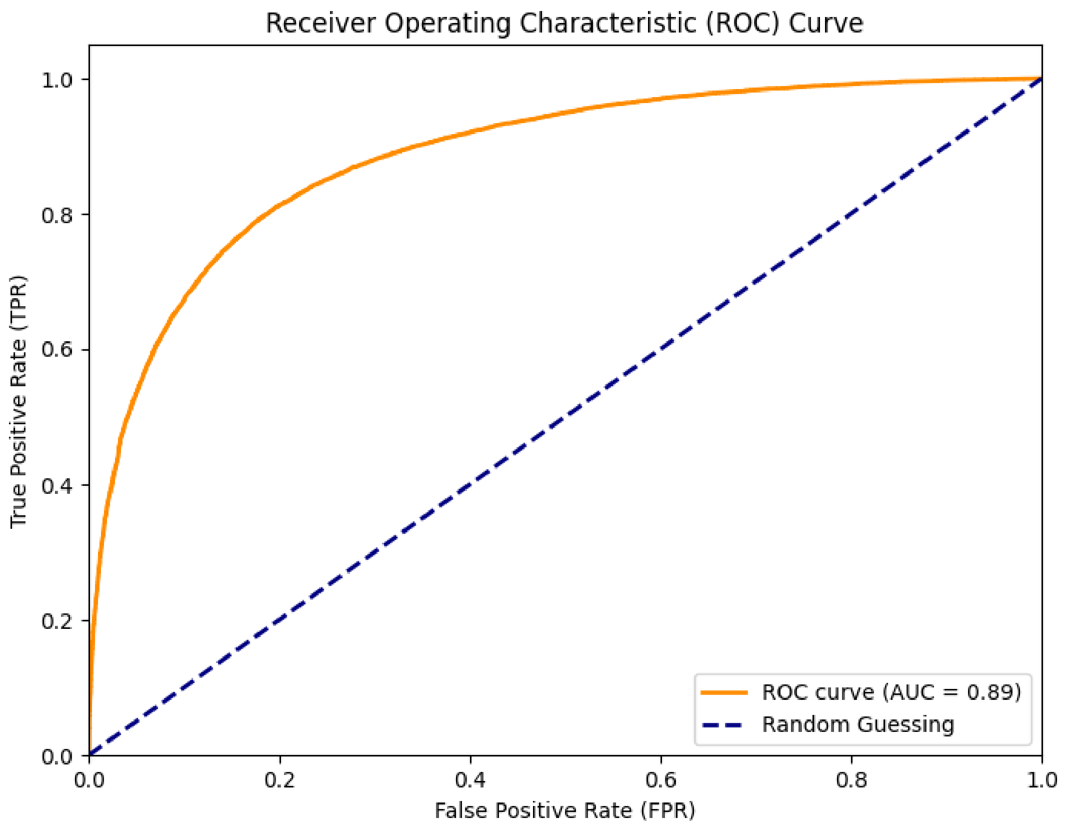
$$\text{Absolute Difference (Percentage)} = \left| \frac{\text{Training Score} - \text{Validation Score}}{\text{Training Score}} \right| \times 100$$

The macro average computes the metrics for each class independently and then takes the average.

The weighted average takes into account the weight, meaning classes with more samples in each dataset contribute more to the final score.

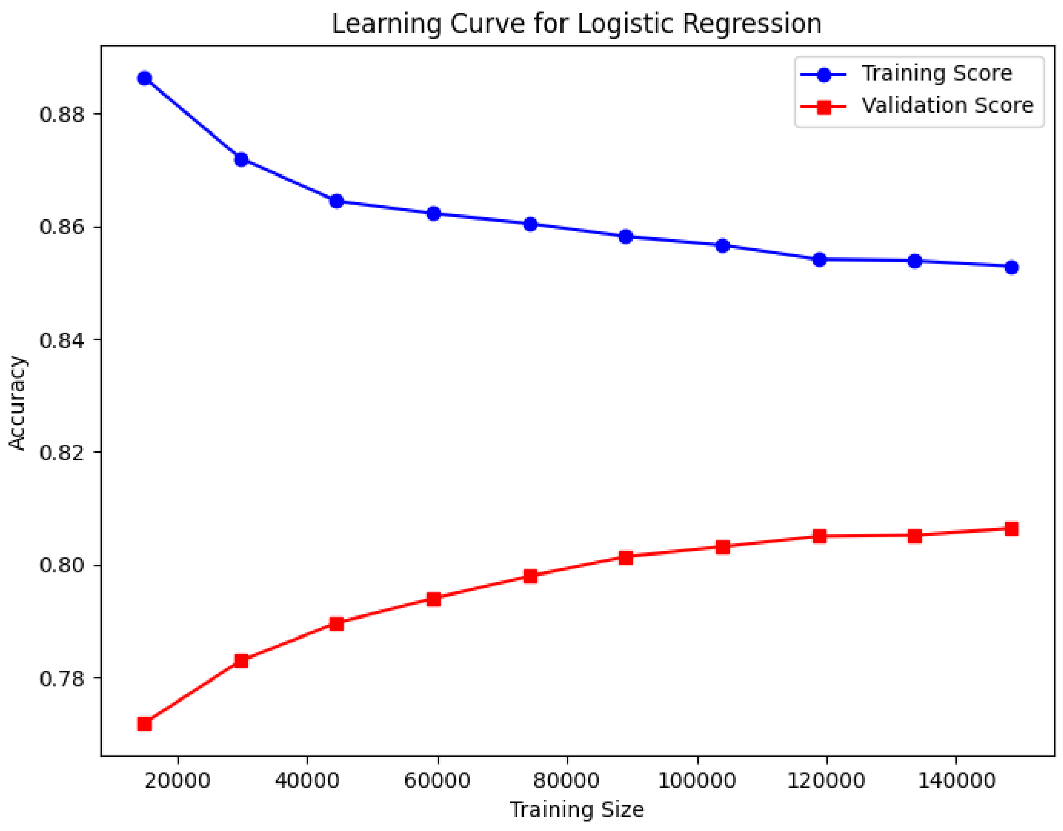
The results for the Classification Report (all metrics mentioned above) will be analysed in the Results section of the report.

3.4.1. ROC curve.



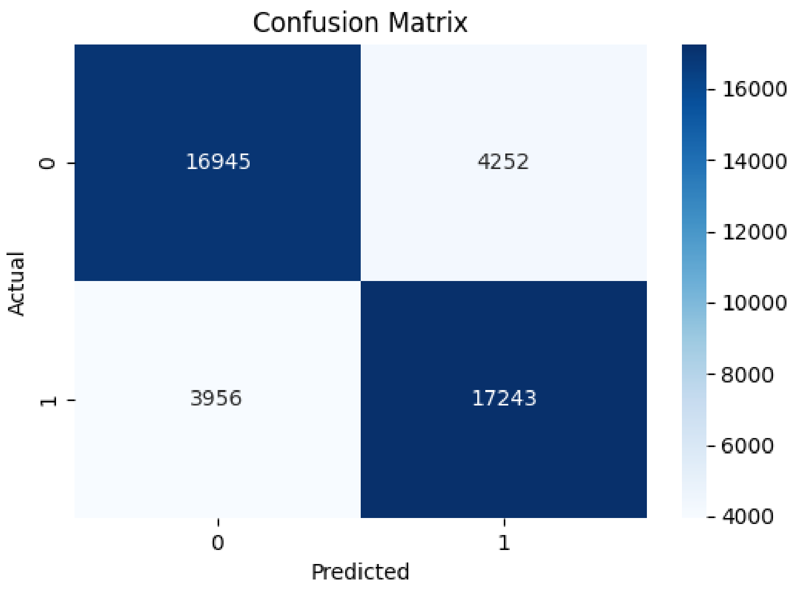
The Receiver Operating Characteristic (ROC) curve shown in the above graph illustrates the trade off between the True Positive Rate (Recall) and the False Positive Rate. In our the classifier is working great in detecting positive instances while maintaining a low false positive rate. The bigger the curve, the better the model. The Area Under the Curve (AUC) value is 0.89. The higher the value the better, with a value of 1 signifying a perfect classifier.

3.4.2. Learning Curve.



The learning curve graph reinforces the argument that the model is in fact not overfitting. The higher the size of the dataset, the closer the training and the validation scores converge, obviously at a lower rate. As mentioned previously, the absolute difference (percentage) between training score and the validation score is 4.65428%.

3.4.3. Confusion matrix.



Out of 42.396 different tweets in the validation dataset, 21.197 were predicted to have negative sentiment and 21.199 were predicted to have positive sentiment. Out of those which were predicted to have negative sentiment, 16945 were labeled correctly, while 4252 were mislabeled. Out of those which were predicted to have a positive sentiment, 17243 were labeled correctly, while 3956 were mislabeled. More on the accuracy of each sentiment will be discussed in the Results Analysis.

4. Results and Overall Analysis

4.1. Results Analysis

The validation accuracy of the model is 0.80640. The highest it ever reached between preprocessing and hyperparameter tuning was 0.80647. The accuracy when testing the model on the training data set is 0.85294. The Absolute Difference (Percentage) as discussed previously is determined to be 4.65428%. The accuracy score when testing the model on the testing dataset is 0.80390.

Classification Report as produced by the sklearn library:

Classification Report				
	Precision	Recall	F1-Score	Support(Samples)
Negative Sentiment (0)	0.81073	0.79941	0.80503	21197
Positive Sentiment (1)	0.80219	0.81339	0.80775	21199
Accuracy			0.80640	42396
Macro Average	0.80646	0.80646	0.80639	42396
Weighted Average	Precision	0.80646	0.80646	42396

Table 14: Classification Report produced by the Sklearn Library

As noted in the Classification Report, the precision is slighty higher when detecting negative sentiment. The classifier labeled tweets as negative with a precision of 0.81073, while it labeled tweets as positive with a precision of 0.80219. In our case, having a higher precision in one class over the other is not as important.

The Recall or else the True Positive Rate (TPR) is is lower when detecting negative a sentiment than when detecting a positive one. The trade off between precision and recall is expected and does not play a huge role in our classifier. The probability of detection is 0.79941 when labeling a tweet as havnig a negative sentiment and 0.81339 when labeling a tweet as having a positive sentiment.

In regards to the F1-score, the values between when detecting a negative and a positive sentiment are quite close. Since F1-score takes into consideration both precision and recall and since the classes are evenly balanced, the results are to be expected. The F1-score for a negative sentiment is 0.80503, whereas the F1-score for a positive sentiment is 0.80775.

The two classes have an even distribution in the dataset. Support (samples) confirms it.

The macro averages and the weighted averages for the precision are 0.80646. The macro averages and the weighted averages for the recall are again 0.80646. The macro averages and the weighted averages for the F1-score are 0.80639. The fact that the macro averages and the weighted averages are the same for each score, concludes that the dataset is indeed evenly balanced.

4.1.1. Best trial. Training Accuracy: 0.85294
Validation Accuracy: 0.80647
Testing Accuracy: 0.80503