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 2025*

Contents

1 Abstract **2**

2 Data processing and analysis **2**
2.1 Pre-processing . . . 2
2.2 Analysis . . . 2
2.3 Data partitioning for train, test and validation . . . 11
2.4 Vectorization . . . 11

3 Algorithms and Experiments **11**
3.1 Experiments . . . 11
3.1.1 Table of trials . . . 13
3.2 Hyper-parameter tuning . . . 14
3.3 Optimization techniques . . . 14
3.4 Evaluation . . . 14
3.4.1 ROC curve . . . 16
3.4.2 Learning Curve . . . 17
3.4.3 Confusion matrix . . . 18

4 Results and Overall Analysis **18**
4.1 Results Analysis . . . 18
4.1.1 Best trial . . . 19
4.2 Comparison with the first project . . . 20

1. Abstract

The goal is to develop a sentiment classifier using a Feed Forward Neural Network (FFNN) model with either Word2Vec or GloVe as the vectorization technique. We have chosen to use the pre-trained GloVe model "glove.twitter.27B.200d" for this task. The dataset, which is sourced from Twitter, consists of three columns: the first is an ID, the second contains the text (tweet), and the third is a label. In the label column, 0 represents a negative sentiment, and 1 represents a positive sentiment. The task is divided into three main steps: first, data pre-processing, second, loading the GloVe model and building a vocabulary from it, and finally, creating, training, and hyper-optimizing the Feed Forward Neural Network model.

2. Data processing and analysis

2.1. Pre-processing

The pre-processing process began by removing all non-ASCII characters from the text, including those embedded within strings containing ASCII characters. Any character that appeared three or more times consecutively was reduced to exactly two occurrences. Usernames were removed, as they were considered irrelevant to the sentiment analysis. Numbers were replaced with the tag <number>. Finally, the text was de-accented, converted to lowercase, and tokenized.

2.2. Analysis

Before Preprocessing:

The size of the vocabulary in the training dataset is 161,429 unique words out of a total of 2,453,768 words. The size of the vocabulary in the validation dataset is 64,533 unique words out of a total of 702,940 words. The size of the vocabulary in the test dataset is 39,033 unique words out of a total of 353,018 words.

Training Dataset	
Total Words	**Unique Words**
2,453,768	161,429

Table 1: Training Data Set Word Count Before Preprocessing

Validation Dataset	
Total Words	**Unique Words**
702,940	64,533

Table 2: Validation Data Set Word Count Before Preprocessing

Testing Dataset	
Total Words	**Unique Words**
353,018	39,033

Table 3: Testing Data Set Word Count Before Preprocessing

Sample (first 10 rows) from the Training dataset:

ID	User ID	Text	Label
0	189385	@whoisralphie dude I'm so bummed ur leaving!	0
1	58036	oh my god, a severed foot was foun in a wheely bin in cobham!!! where they found is literally minutes from my house! feel sick now!	0
2	190139	I end up "dog dialing" sumtimes. What's dog dialing, u ask? My dogs will walk across my phone & end up calling someone. aka "dog dialing"!	1
3	99313	@_rachelx meeeee toooooo!	0
4	157825	I was hoping I could stay home and work today, but looks like I have to make another trip into town	0
5	130560	says plurk karma finally reached the 50s. still no heartsy smileys. boo. http://plurk.com/p/z2x3y	0
6	121871	Good to hear it @Arth This is a bit more, but a la four tet Do you know Free Rotation festival? Am thinking ... ♪ http://blip.fm/7̃hcvo	1
7	86813	@davorg in that case im gonna start tweeting about nymphomanic pub owners who like to cook, well worth a shot, eh	1
8	197517	@belunyc its alright love, how are you?	1
9	6937	@brightondoll haha that has to be the best analogy ever. mogwai to gremlin. love it. i love gizmo and the gremlins movies	1

Table 4: Sample Data Before Pre-Processing

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

Training Dataset	
Word	**Frequency**
!	85,636
@	74,891
.	74,742
I	60,927
to	51,952
the	45,926
,	45,383
a	34,473
i	28,200
and	26,513

Table 5: Training Data Set Most Common Words Before Preprocessing

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

Validation Dataset	
Word	**Frequency**
!	24,852
.	21,913
@	21,508
I	17,517
to	14,640
the	13,084
,	12,933
a	9,886
i	8,117
my	7,624

Table 6: Validation Data Set Most Common Words Before Preprocessing

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

Testing Dataset	
Word	**Frequency**
!	12,425
.	10,707
@	10,703
I	8,564
to	7,495
the	6,670
,	6,646
a	5,009
i	3,941
my	3,866

Table 7: Testing Data Set Most Common Words Before Preprocessing

As expected, when stop words and punctuation are included, they dominate the list of the most common words in each dataset. However, since removing all stop words and punctuation leads to a decrease in accuracy, it has been decided that it is better to include them. Therefore, while stop words and punctuation may appear insignificant, they contribute to the overall semantic structure of the text. On the other hand, accent marks (diacritics) will be removed during data cleaning, as they do not add significant value to the sentiment analysis.

Below are the Word Clouds (excluding stop words):

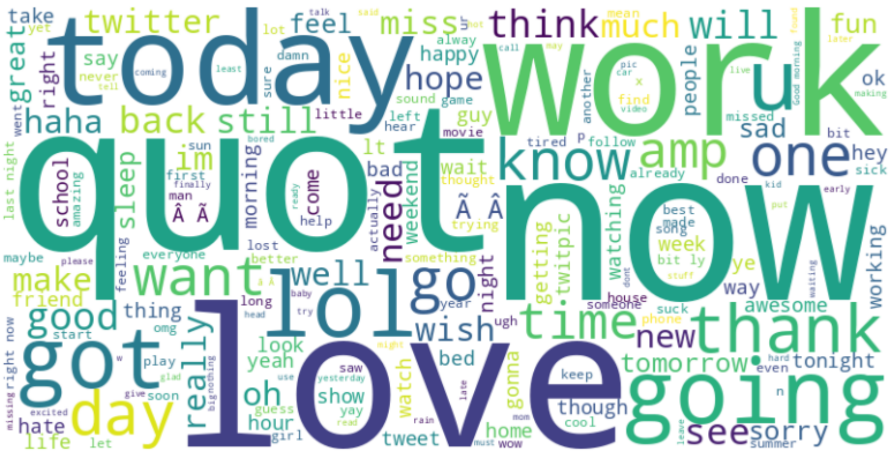


Figure 1: Training Dataset World Cloud Before Preprocessing

Figure 2: Validation Dataset World Cloud Before Preprocessing

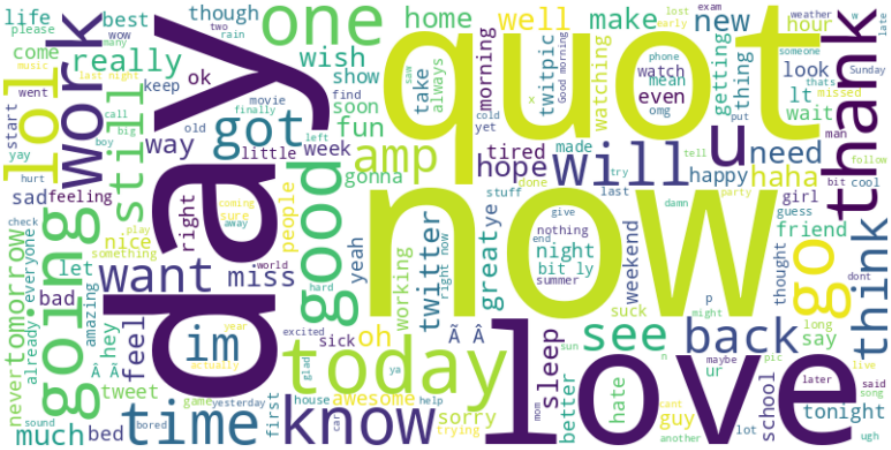


Figure 3: Testing Dataset World Cloud Before Preprocessing

After Preprocessing:

The size of the vocabulary in the training data set is 65,756 unique words out of a total of 1,797,300 words. The size of the vocabulary in the validation data set is 31,458 unique words out of a total of 513,863 words. The size of the vocabulary in the test data set is 20,824 unique words out of a total of 258,593 words.

Training Dataset	
Total Words	**Unique Words**
1,797,300	65,756

Table 8: Training Data Set Word Count After Preprocessing

Validation Dataset	
Total Words	**Unique Words**
513,863	31,458

Table 9: Validation Data Set Word Count After Preprocessing

Testing Dataset	
Total Words	**Unique Words**
258,593	20,824

Table 10: Testing Data Set Word Count After Preprocessing

Sample (first 10 rows) from the Training dataset:

ID	User ID	Text	Label
0	189385	[dude, so, bummed, ur, leaving]	0
1	58036	[oh, my, god, severed, foot, was, foun, in, wheely, bin, in, cobham, where, they, found, is, literally, minutes, from, my, house, feel, sick, now]	0
2	190139	[end, up, quot, dog, dialing, quot, sumtimes, what, dog, dialing, ask, my, dogs, will, walk, across, my, phone, amp, end, up, calling, someone, aka, quot, dog, dialing, quot]	1
3	99313	[mee, too]	0
4	157825	[was, hoping, could, stay, home, and, work, today, but, looks, like, have, to, make, another, trip, into, town]	0
5	130560	[says, plurk, karma, finally, reached, the, still, no, heartsy, smileys, boo, http, plurk, com]	0
6	121871	[good, to, hear, it, this, is, bit, more, but, la, four, tet, do, you, know, free, rotation, festival, am, thinking, http, blip, fm, hcvo]	1
7	86813	[in, that, case, im, gonna, start, tweeting, about, nymphomanic, pub, owners, who, like, to, cook, well, worth, shot, eh]	1
8	197517	[its, alright, love, how, are, you]	1
9	6937	[haha, that, has, to, be, the, best, analogy, ever, mogwai, to, gremlin, love, it, love, gizmo, and, the, gremlins, movies]	1

Table 11: Sample Data After Pre-Processing

Out Of Vocabulary (OOV) words in the datasets after pre-processing:

Dataset	Total OOV Words	Total Words	Percentage of OOV Words
Training	16,653	1,797,300	0.93%
Validation	4,885	513,863	0.95%
Testing	2,583	258,593	1.00%

Table 12: OOV Words After Pre-Processing

The ten most common words in the training dataset are:

Training Dataset	
Word	**Frequency**
to	52,675
the	48,997
my	29,542
and	28,554
it	28,540
you	27,869
is	22,403
<number>	20,786
for	20,384
in	20,166

Table 13: Training Data Set Most Common Words After Preprocessing

The ten most common words in the validation dataset are:

Validation Dataset	
Word	**Frequency**
to	14,833
the	14,037
my	8,579
it	8,226
and	8,037
you	7,933
is	6,338
<number>	5,884
for	5,797
in	5,764

Table 14: Validation Data Set Most Common Words After Preprocessing

The ten most common words in the testing dataset are:

Testing Dataset	
Word	**Frequency**
to	7,593
the	7,096
my	4,332
and	4,148
it	4,120
you	3,990
is	3,210
in	2,998
for	2,903
<number>	2,888

Table 15: Testing Data Set Most Common Words After Preprocessing

After pre-processing the datasets, the number of both total and unique words decreased significantly. Words and characters that did not contribute much to sentiment analysis were removed. Additionally, all words were converted to lowercase.

An important thing to note is that the tag <number> occupies a significant portion of the vocabulary. This tag was used to replace all numeric values in the text. Since numbers appear frequently in the dataset retaining this tag helps preserve important context without losing generalization.

<number> TAG	
Dataset	**Frequency**
Training	20,786
Validation	5,884
Testing	2,888

Table 16: Frequency of <number> in Datasets

Class Distribution:

Training Class Distribution	
Sentiment	**Count Of Tweets**
Negative (0)	74,192
Positive (1)	74,196

Table 17: Distribution Of Classes In Validation Dataset

Training Class Distribution	
Sentiment	**Count Of Tweets**
Negative (0)	21,197
Positive (1)	21,199

Table 18: Distribution Of Classes In Validation Dataset

It is evident that both the training and validation datasets are well-balanced, which suggests that the model is likely to generalize effectively. While we can only assume that the testing dataset is also balanced, this balance in the training and validation sets provides a strong foundation for reliable model performance.

Below are the Word Clouds (excluding stop words):

Figure 4: Training Dataset World Cloud After Preprocessing

Figure 5: Validation Dataset World Cloud After Preprocessing

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

The vectorization was implemented using a GloVe embedding, specifically **glove.twitter .27B.200d**. This is a 2.06GB text file containing a pre-trained embedding model trained on 27 billion tokens (words), with each word represented as a vector of 200 dimensions. The same model is available in 25, 50 and 100 dimensions, but as expected, these versions performed slightly worse. The **glove.6B** model, which offers word representations as vectors of 300 dimensions, was also tested; however, it resulted in a two-point decrease in accuracy. Similar results were observed with the **glove.840B.300d** model. An attempt was made to create a custom Word2Vec model using the training data, but the performance did not meet expectations, so it was ultimately discarded. In the end, **glove.twitter.27B.200d** proved to be the best choice.

3. Algorithms and Experiments

3.1. Experiments

The initial model selected was a simple neural network consisting of a single layer with dimensions $(200, 1)$. The loss function used was `BCELoss()`, which required the model to return a sigmoid output. The parameter `embedding.weight.requires_grad` was set to `True`, allowing the GloVe embeddings to be fine-tuned during training.

In the first run, no advanced preprocessing was applied—only basic tokenization. This resulted in poor performance: a training accuracy of **50.0007** and a validation accuracy of **49.9953**. At this stage, the model performed no better than random guessing.

Next, numerical values were replaced with a `<number>` token, which was recognized by the vocabulary. Additional preprocessing steps were introduced: lowercasing, de-accenting, and filtering out words with fewer than two characters. These transformations were bundled into a single custom function, producing a result similar to Gensim's `simple_preprocess`, with the added benefit of handling numeric tokens. These changes slightly improved performance, increasing the training and validation accuracies to **50.0202** and **50.0259**, respectively.

A significant improvement was observed after introducing the **Adam** optimizer with a learning rate of 1×10^{-3}. The training accuracy rose to **81.5531**, and the validation accuracy improved to **78.8518**.

Subsequent efforts focused on reducing Out-of-Vocabulary (OOV) words. Removing usernames led to a slight decrease in training accuracy (**80.7653**) but improved the validation accuracy to **78.9862**. Lemmatization increased training accuracy to **83.5728** but reduced validation performance to **78.2998**, so it was retained for its training benefits. Stemming, however, decreased both metrics to **79.4478** and **78.3612** and was discarded.

Reducing repeated characters (more than two consecutive occurrences) yielded a training accuracy of **80.9270** and validation accuracy of **79.1277**. Removing non-ASCII characters had a minor negative effect, slightly lowering training and validation scores to **80.9095** and **79.1136**, respectively. After all preprocessing steps, the number of OOV words in the training dataset dropped to under 5,000, fewer than 0.5% of the vocabulary.

Adding a hidden layer of 100 units initially led to overfitting, spiking training accuracy to **93.1012** while decreasing validation accuracy to **78.0074**. Instead, the middle layer was resized to 738 units, and **batch normalization**, **ReLU**, and **dropout** (with $p = 0.18$) were applied. This improved both metrics: training accuracy reached **81.9763**, and validation accuracy improved to **79.5570**.

Using **Optuna**, the final model architecture and hyperparameters were selected:

```
FFNN(
 (embedding): Embedding(1193516, 200)
 (fc1): Linear(in_features=200, out_features=738, bias=True)
 (bn1): BatchNorm1d(738, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (drop1): Dropout(p=0.18300243039826286, inplace=False)
 (fc2): Linear(in_features=738, out_features=446, bias=True)
 (bn2): BatchNorm1d(446, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (drop2): Dropout(p=0.4217747676930847, inplace=False)
 (fc3): Linear(in_features=446, out_features=178, bias=True)
 (bn3): BatchNorm1d(178, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (drop3): Dropout(p=0.08787363292289922, inplace=False)
 (fc4): Linear(in_features=178, out_features=89, bias=True)
 (bn4): BatchNorm1d(89, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
 (drop4): Dropout(p=0.1, inplace=False)
 (fc5): Linear(in_features=89, out_features=1, bias=True)
)
```

This configuration achieved a training accuracy of **83.0519** and a validation accuracy of **79.1348**.

Several learning rate schedulers were tested: **StepLR**, **CosineAnnealingLR**, **CosineAnnealingWarmRestarts**, and **ReduceLROnPlateau**. Among these, **ReduceLROnPlateau** performed best during the Optuna tuning.

Finally, the optimizer was changed to **AdamW**, with hyperparameters optimized via Optuna:

- Learning rate: 9.3339×10^{-5}
- Betas: $(0.95, 0.98)$
- Weight decay: 2.5533×10^{-5}

While the training accuracy slightly decreased to **82.3759**, the validation accuracy surpassed the 80% threshold for the first time, peaking at **80.091**.

3.1.1. Table of trials.

Following is a table of the main experiments performed:

Trial	Description (Technique + Pre-Processing)	Training Score	Validation Score
Trial 1	Default model with no preprocessing	50.0007	49.9953
Trial 2	Default model with lowercasing, de-accenting, length filtering, and `<number>` tag	50.0202	50.0259
Trial 3	Adam optimizer (lr = 1e-3), no preprocessing changes	81.5531	78.8518
Trial 4	Removed usernames	80.7653	78.9862
Trial 5	Lemmatization applied(Reverted)	83.5728	78.2998
Trial 6	Stemming applied(Reverted)	79.4478	78.3612
Trial 7	Reduced repeated characters	80.9270	79.1277
Trial 8	Removed non-ASCII characters	80.9095	79.1136
Trial 9	Model changed to 200 $\rightarrow$ 100 $\rightarrow$ 1(Reverted), ReLU activations	93.1012	78.0074
Trial 10	Model: 200 $\rightarrow$ 738 $\rightarrow$ 1 with BatchNorm, Dropout (0.18), ReLU	81.9763	79.5570
Trial 11	Model: 200 $\rightarrow$ 738 $\rightarrow$ 446 $\rightarrow$ 178 $\rightarrow$ 89 $\rightarrow$ 1 with ReLU, BatchNorm, Dropouts	83.0519	79.1348
Trial 12	Scheduler added: ReduceLROnPlateau (mode='max', factor=0.2, patience=5)	83.2985	79.1325
Trial 13	Optimizer changed to AdamW with tuned lr, betas, weight decay via Optuna	82.3759	80.0901

Table 19: Summary of Experimental Trials

3.2. Hyper-parameter tuning

Feed Forward Neural Network Parameter Values:

Layer	Type	Input Size	Output Size	BatchNorm	Dropout
Embedding	Embedding	1,193,516	200	–	–
Layer 1	Linear	200	738	BatchNorm1d(738)	Dropout(0.1830)
Layer 2	Linear	738	446	BatchNorm1d(446)	Dropout(0.4218)
Layer 3	Linear	446	178	BatchNorm1d(178)	Dropout(0.0879)
Layer 4	Linear	178	89	BatchNorm1d(89)	Dropout(0.1)
Output	Linear	89	1	–	–

Table 20: FFNN Architecture Overview

The training accuracy reached **84.17864%**, while the validation accuracy peaked at **80.13728%**. The absolute difference between these accuracies is relatively small, at **4.04136%**, which is well below 5%. This difference is acceptable, suggesting that the model is generalizing well. In combination with the learning curve (showing accuracy and loss), the ROC curve, and the confusion matrix, we can confidently conclude that the model is neither overfitting nor underfitting.

Metric	Value
Training Accuracy	84.17864
Validation Accuracy	80.13728
Absolute Difference (Percentage)	4.04136%

Table 21: Training and Validation Accuracy Comparison

3.3. Optimization techniques

For optimization, the **Optuna** framework was employed to fine-tune all hyperparameters of the model, including model depth, layer sizes, dropout rates, and activation functions. Each hyperparameter was adjusted based on the highest validation accuracy achieved during testing. After conducting experiments with various optimizers, **AdamW** was selected over **Adam** and **SGD**, as it showed better performance in terms of convergence and stability. The respective hyperparameters of the **AdamW** optimizer, such as learning rate and weight decay, were also tuned through Optuna's trials. In addition to the optimizer, the learning rate scheduler was chosen after evaluating four different scheduler options, allowing for more efficient learning rate adjustments throughout training. Due to Kaggle's limited session time, rather than running a large single Optuna trial, multiple smaller trials were performed, each focusing on optimizing a single hyperparameter. This approach proved effective in fine-tuning the model and yielded satisfactory results.

3.4. Evaluation

The scores used to evaluate the predictions include Loss(Cost) function, 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 Loss function measures how far off the model's predictions are from the actual values, with the goal of minimizing this difference:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right]$$

where:

- N is the number of samples,
- y_i is the true label (0 or 1),
- p_i is the predicted probability for the positive class (1).

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:

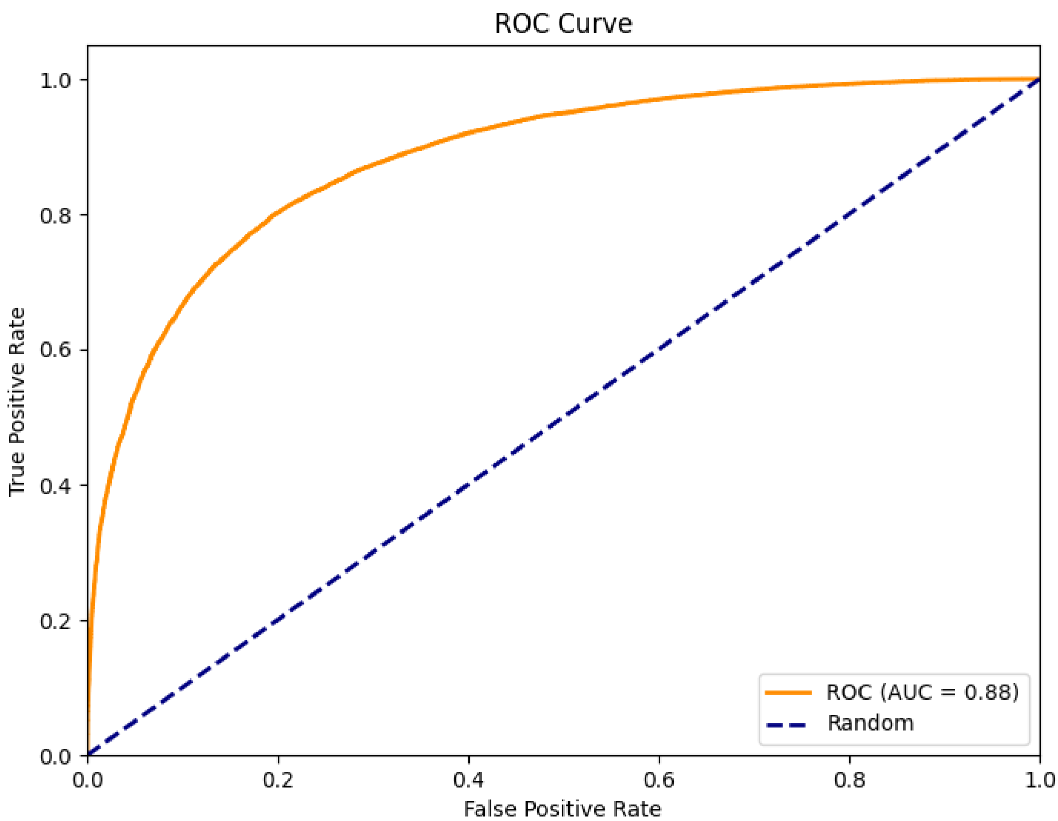
$$\text{Absolute Difference (Percentage)} = \left|\frac{\text{Training Score} - \text{Validation Score}}{\text{Training Score}}\right| \times 100$$

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.

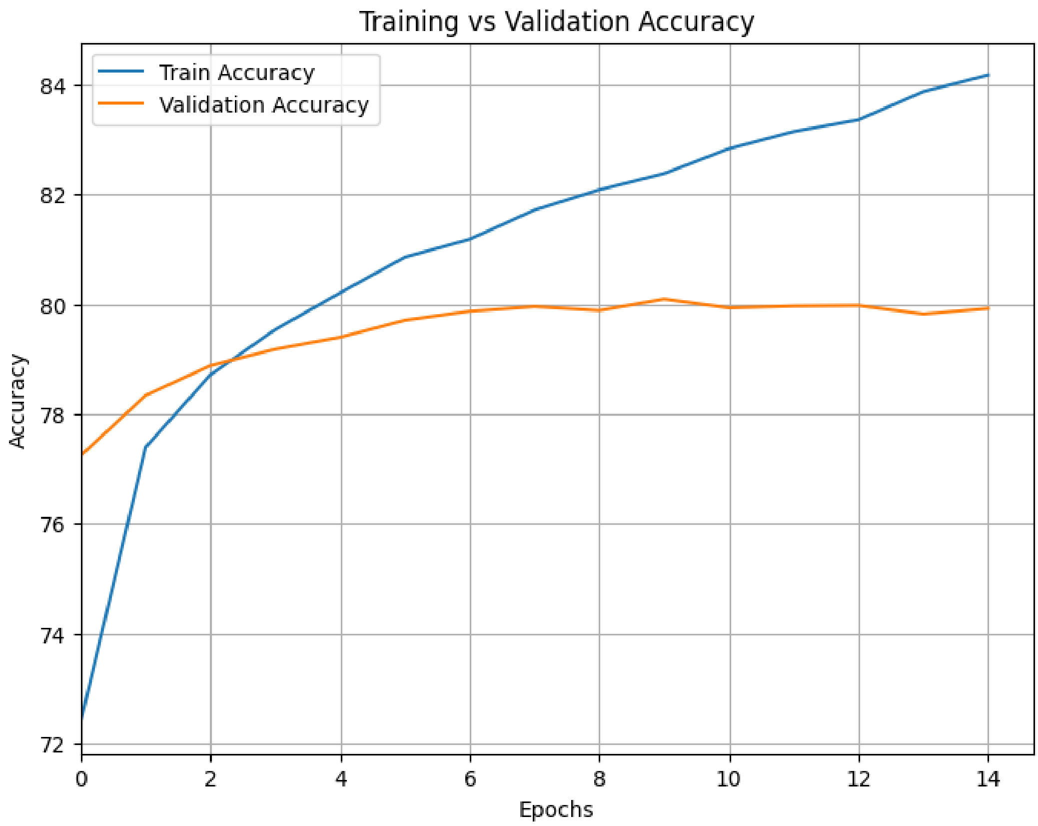
The results for the Classification Report (all metrics mentioned above) will be analysed in the Results section of the report.

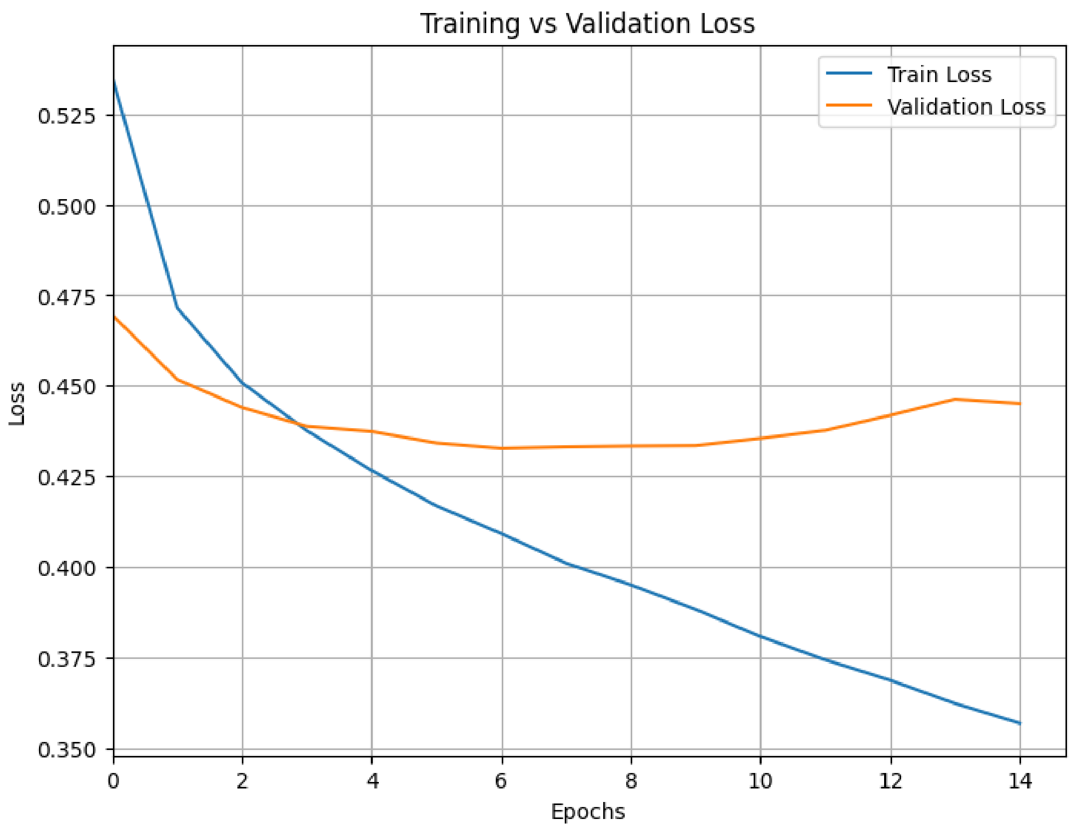
3.4.1. ROC curve.



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 0.88. The higher the value the better, with a value of 1 signifying a perfect classifier.

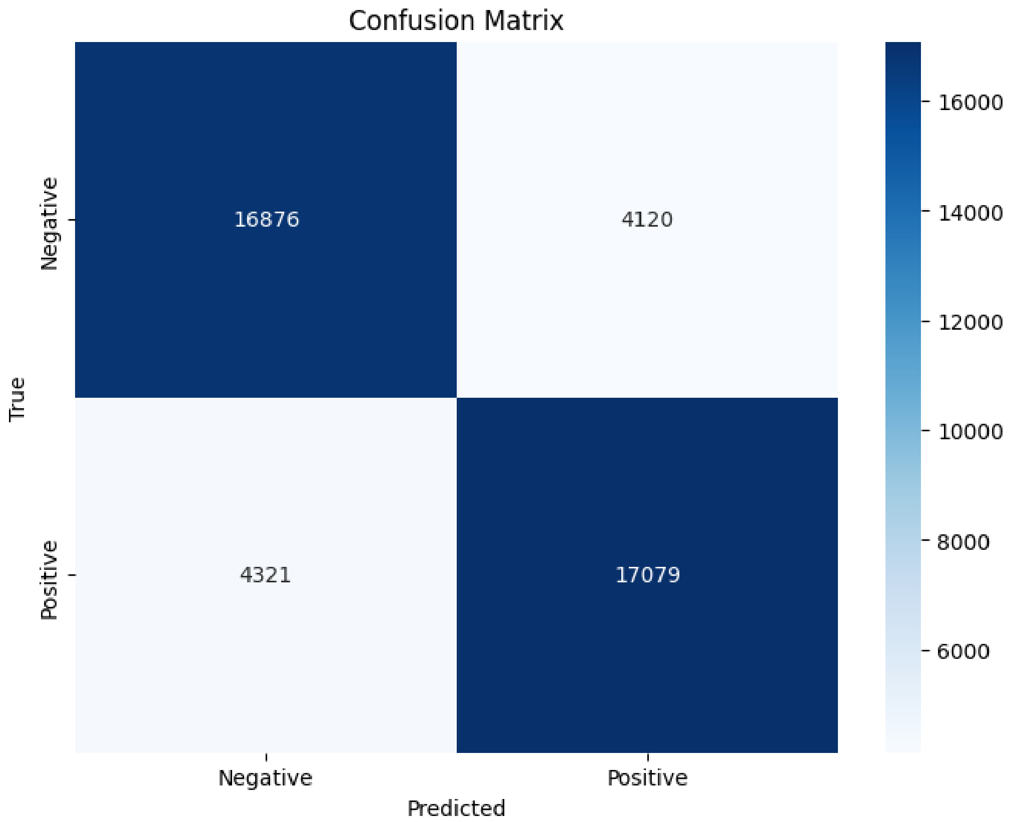
3.4.2. Learning Curve.





As the size of the dataset increases, the training and validation scores gradually converge, though at a diminishing rate. Eventually, the model begins to overfit the training data, while the validation performance stagnates or even starts to decrease. At this point, early stopping is triggered, halting the training process, and the model weights corresponding to the best validation accuracy are saved. As mentioned earlier, the absolute difference in percentage between the training and validation scores is **4.04136%**, which is an acceptable result. Thus the conclusion is made that learning curves show a healthy model which generalizes well.

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, **16,876** were labeled correctly, while **4,120** were mislabeled. Out of those which were predicted to have a positive sentiment, **17,079** were labeled correctly, while **4,321** 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 model achieved a validation accuracy of **80.0901**, which is the highest value observed throughout the pre-processing and hyperparameter tuning phases. At the corresponding epoch (**10**), the training accuracy was **82.3759**, reflecting the model's performance before overfitting could occur. As expected, training accuracy continues to increase with more epochs and can eventually approach **100%**. The absolute difference between training and validation accuracy at this point is **4.07978%**, which is within acceptable bounds and suggests good generalization. Finally, when evaluated on the test dataset, the model attained a test accuracy of **79.947**, further supporting the conclusion that the model generalizes well across unseen Twitter data.

We could have also experimented with K-Fold Cross-Validation, as well as different non-linearities such as GELU and Leaky ReLU. Additionally, training multiple models with different random seeds and averaging their predictions could have further improved performance.

Classification Report as produced by the scikit-learn library:

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Negative	0.80377	0.79615	0.79994	21197
Positive	0.79808	0.80565	0.80185	21199
Accuracy		0.80090		42396
Macro avg	0.80093	0.80090	0.80090	42396
Weighted avg	0.80093	0.80090	0.80090	42396

Table 22: Classification Report for Best Model

As noted in the updated Classification Report, the **precision** is slightly higher when detecting **negative sentiment**. The classifier labeled tweets as negative with a precision of **0.80377**, while it labeled tweets as positive with a slightly lower precision of **0.79808**. In this context, a minor difference in precision between classes is not particularly critical, especially given the balanced nature of the dataset.

The **recall** (also referred to as the True Positive Rate) is slightly higher when detecting **positive sentiment**. The recall score for negative tweets is **0.79615**, while for positive tweets it is **0.80565**. This small trade-off between precision and recall is expected and generally acceptable in sentiment classification.

Looking at the **F1-scores**, which balance both precision and recall, the values for the two classes are again quite close. The F1-score for negative sentiment is **0.79994**, while for positive sentiment it is **0.80185**, indicating that the model performs comparably across both classes.

The support values for both classes are nearly identical (**21197** for negative and **21199** for positive), confirming that the dataset is evenly balanced. This balance is further reflected in the macro and weighted averages.

Both the **macro average** and **weighted average** precision are **0.80093**, while the recall and F1-scores are both **0.80090** for each averaging method. The equality of macro and weighted averages confirms the dataset's class balance and suggests that the classifier does not show significant bias toward either class.

4.1.1. Best trial. **Project2:**

Metric	Accuracy
Training Accuracy	84.1699
Validation Accuracy	80.0901
Testing Accuracy	79.947

Table 23: Accuracy Results for the Best Trial of Project2

4.2. Comparison with the first project

Below are the results of the first project (a Sentiment analysis model using Logistic Regression and TF-IDF vectorization):

Project1	
Metric	**Accuracy**
Training Accuracy	0.85294
Validation Accuracy	0.80647
Testing Accuracy	0.80503

Table 24: Accuracy Results for the Best Trial of Project1

Classification Report For Project1				
	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	0.80646	0.80646	0.80646	42396

Table 25: Classification Report produced by the Sklearn Library for Project1

Logistic Regression with TF-IDF is a simpler model which works quite well when the dataset is not large. Our Feed Forward Neural Network model in combination with GloVe is slightly more complex requiring more precise hyperparameter tuning.

The TF-IDF Logistic Regression model slightly outperforms the Feed Forward Neural Netwrok model in terms of accuracy, by about 0.55

In both models, precision is quite close for both classes, but the Logistic Regression model with TF-IDF has a slightly higher precision for both negative and positive classes, about 0.007-0.008 higher.

The Logistic Regression model demonstrates slightly better recall for both classes. It has a higher recall for both the negative and positive classes, suggesting it is slightly better at detecting both classes correctly.

The F1-scores for the Logistic Regression model are slightly better across both classes. This suggests that Logistic Regression model balances precision and recall better than the FFNN model.

The macro averages for FFNN are all slightly lower than those for Logistic Regression, reinforcing the fact that the Logistic Regression model performs better overall.

The weighted averages for the Logistic Regression model are also slightly higher than those for FFNN, which is expected from our previous results and the fact that the dataset is (almost) perfectly balanced.

All the scores are very similar with an approximately 1% variance. The Logistic Regression model was way faster to train, even on the provided CPU, whereas the Feed Forward Neural Network required the use of a GPU. This made the first model easier to tune and thus provide better results for our limited dataset. On the other hand, although the model from Project1 seems to be near it's limit in regards of improvement, our Feed Forward Neural Network model has potential for way more improvement, even only by using the same dataset.