UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Nikitas Rafail Karachalios*
sdi: -

Course: *Artificial Intelligence II (M138, M226, M262, M325)*
Semester: *Fall Semester 2025*

Contents

1. Abstract

The aim of this project is to develop a sentiment classifier using only TF-IDF and Logistic Regression on a given English language twitter dataset. More specifically, our model should be able to decide whether a twitter comment is positive or negative. To reach this goal, the steps were EDA, data preprocessing, vectorization and of course model evaluation through experiments. Since it was my first time working in such kind of project, after the analysis, I implemented the simplest TF-IDF + Logistic Regression model I could, and gradually I started improving my preprocessing and the parameters used in the model (manual fine tuning, grid search) so as to maximize the accuracy on validation set and, also, avoiding overfitting. Also, I explored several theoretical aspects of sentiment analysis by reading articles, as I found the concept and its practical applications particularly interesting.

2. Data processing and analysis

2.1. Pre-processing

Based on Homework slides, web search and the results of EDA: 2.2, so as to find ways of reducing noise and cleaning the data, I've concluded to some specific steps you can see below. My baseline so as to decide which steps should be included to the preprocessing function was the classification report for the validation dataset.

1. Lowercase, so as to reduce vocabulary size.
2. URL removal, URLs don't contribute to sentiment.
3. User Mentions removal
4. Remove hashtag symbols but keeping the word since it can contribute to sentiment.
5. Replace emoticons with sentiment words (positive, negative), the sentiment of the emojis is not lost.
6. Negations handling, pretty important for sentiment – Converting contractions to full form.
7. Corrections handling, for common (slang) abbreviations, and for some cases that were found on training dataset.
8. Replace multiple exclamation marks with sentiment booster: exclamation_emphasis. We do this replacement so as to help the vectorizer understand the emotional intensity as a distinct feature and not as random punctuation.
9. Replace repeated characters. Here the idea was to reduce noise and vocabulary size and treat words "flyyyy" and "fly" the same. But, with the command I used, I realised that it led to spelling incorrectness (eg "wheely" to "whely"). When I tried different approaches, for example keeping the first two consecutive characters and removing the rest of them, so as to prevent mispelling, this didn't help. In conclusion, I think this helped vectorizer generalize patterns more effectively due to vocabulary size reduction and I understand it as some kind of accidental regularization since the model focuses on more meaningful features (words like "love", "hate" remain untouched) and doesn't overfit in small differences ("wheeely", "whely", "wheelyy"). The takeaway? Models don't "understand" words, they just process patterns mathematically :)
10. Text strip for better tokenization
11. Tokenization

12. Removed punctuation tokens

Note: We avoided some common preprocessing methods like removing stopwords because they reduced accuracy during experiments. Lemmatization was tested and improved a bit the accuracy but eventually skipped since it was time-consuming and had issues with the library I used in Kaggle :/

2.2. Analysis

We start by printing the label distribution of the training and validation dataset. (Test dataset does not have labels). Also, we check for missing/duplicate values so as to be sure about the data quality.

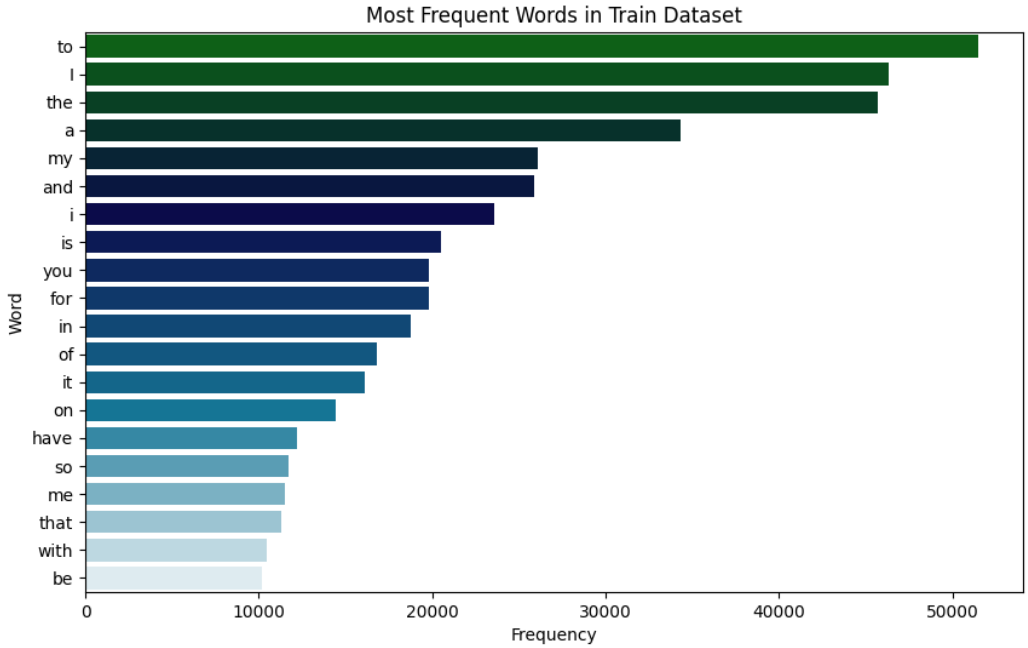
Basic Statistics for Train Dataset:
Number of samples: 148388
Positive labels: 74196 (50.00%)
Negative labels: 74192 (50.00%)
Missing values in each column:
ID 0
Text 0
Label 0
dtype: int64
Duplicate tweets: 0 (0.00%)

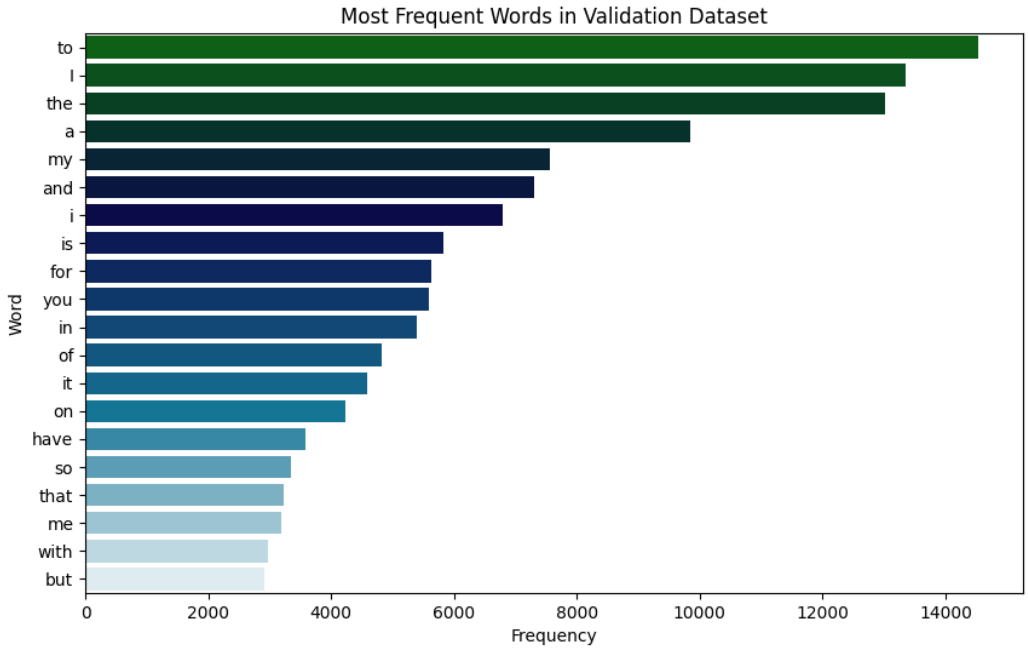
Basic Statistics for Validation Dataset:
Number of samples: 42396
Positive labels: 21199 (50.00%)
Negative labels: 21197 (50.00%)
Missing values in each column:
ID 0
Text 0
Label 0
dtype: int64
Duplicate tweets: 0 (0.00%)

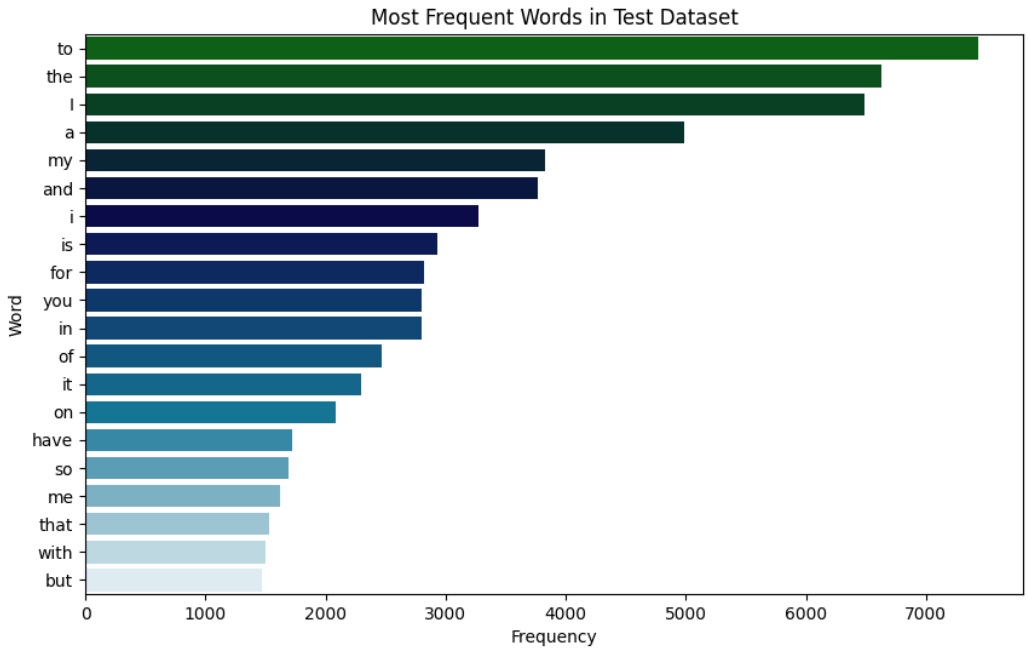
Basic Statistics for Test Dataset:
Number of samples: 21199
Missing values in each column:
ID 0
Text 0
dtype: int64
Duplicate tweets: 0 (0.00%)

It is clear that we don't have any issues with data quality. Also, both datasets (training and validataion) are perfectly balanced, so the model can learn equally from positive and negative examples. If we had an imbalanced set, we might have increased the importance of the minority class so as to help minimizing the loss function and of course, improve performance.

Most frequent words in each dataset

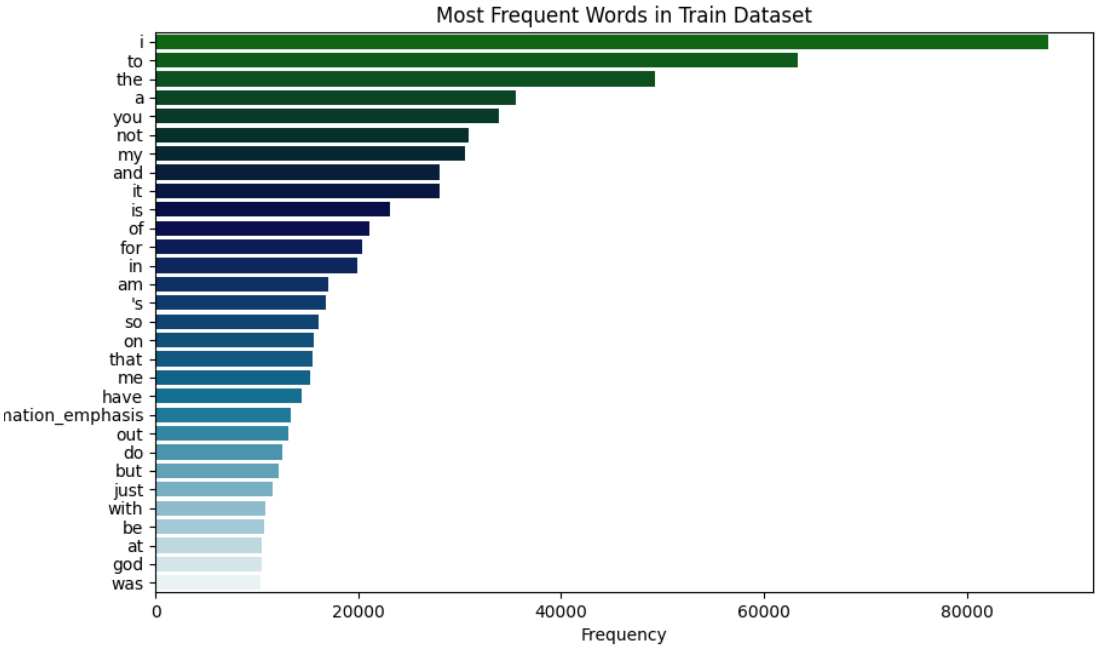






The fact that the most frequent words were stopwords led us to test stopword removal in the **pre-processing**. However, removing them reduced accuracy. If we try to explain this outcome, this may happen because in the case of bigrams, stopwords may create meaningful bigrams that contribute to sentiment analysis (eg "not good" or some phrasal verb). But this is just a thought and it may be something much simpler, such as the characteristics of these specific datasets. Long story short, we keep stopwords since we have better accuracy with them.

Some Examples after preprocessing



In top 30 word frequency now, after preprocessing, we can see some key differnces. For example, the reduction of vocabulary size is clear since words "I" and "i" are treated as one. Also, "exclamation_emphasis" is in top 30, which means that there were a lot of comments with multiple exclamation marks. It is obvious that people often use repeated exclamation marks to intensify emotion, which is crucial for sentiment analysis.

The changes in our data can be seen also through the head of the training dataset **before** and **after** the preprocessing.

```
Train Dataset Sample:
0       @whoisralphie dude  I'm so bummed ur leaving!
1    oh my god, a severed foot was foun in a wheely...
2    I end up &quot;dog dialing&quot; sumtimes. Wha...
3                          @_rachelx meeeee toooooo!
4    I was hoping I could stay home and work today,...
Name: Text, dtype: object
```

Figure 1: Data before

```
Train Dataset Sample:
0                      dude i am so bumed your leaving
1    oh my god a severed fot was found in a whely b...
2    i end up quot dog dialing quot sometimes what ...
3                                                 me to
4    i was hoping i could stay home and work today ...
Name: clean_text, dtype: object
```

Figure 2: Data after

To remember some steps from preprocess 2.1, mentions are removed, the spelling of some words with 2 consecutive characters has changed ("fot" instead of "foot") - we explained why, punctuation that doesn't contribute to sentiment is also removed, "meeeee tooooooo" is now "me to" and the more we search, the more differences we find!

2.3. Data partitioning for train, test and validation

In this project, as we have said, the datasets were provided pre-partitioned by the instructor into training, validation, and test sets. According to the instructions, these sets were used respectively for training, validation, and testing the model. As a result, no manual partitioning was needed. The main execution block in the code shows this by directly loading the datasets from separate .csv files for each set.

After loading the data, text preprocessing was applied to create a new column called "clean_text" in each dataset. This column contains the processed text data, which was then used for feature extraction, model training, and evaluation.

2.4. Vectorization

For vectorization, we used the TF-IDF Vectorizer from the scikit-learn library, following the approach presented in our lecture slides. The parameters we selected are:

- **ngram_range=(1,2)**: We considered both unigrams and bigrams to capture more context in the text data.
- **max_features=30000**: The ideal scenario is to have a complex model with the "best" accuracy, but also avoid the risk of overfitting, but this is not possible! Now, we use 30k features having as a baseline the validation dataset accuracy! More research for this decision in Units 3 and 4.
- **min_df=3**: We set the minimum document frequency to 3 to exclude very rare terms that might add noise.
- **max_df=0.7**: We set the maximum document frequency to 0.7 to remove very common terms that might not be useful for classification.

3. Algorithms and Experiments - Best Trial

3.1. Experiments

Below you can see some specific trials I kept during the multiple(!) experiments I tried. As we said in 1, I started with the simplest model, and through lecture slides, EDA, preprocess and fine-tuning, I improved model's accuracy and reliability.

Trial	Score	Comment
No Preprocess-Default TF-IDF and Log Reg	0.79090008	Impressive for nothing done
Final Preprocess-Default TF-IDF and Log Reg	0.79115954	Preprocess only not enough
No Stopwords- Final Model (features 10k)	0.77132277	Avoid Stop Word Removal
Final Preproc- Final Model (features 10k)	0.80196245	No overfit, decent accuracy
Final Preproc-Grid Search(features 10k)	0.79494969	Grid Search didn't help
Final Preproc- Final Model (features 30k)	0.80460421	More risk for overfit, best acc
Final Preproc- Final Model (features 20k)	0.80311822	Balance acc and overfit

Table 1: Trials

3.1.1. Table of trials. Note: Experiments in preprocessing have been analysed in the preprocessing section. As far as feature extraction is concerned, apart from grid search and features modification that are in the above table, we tried different combinations of n-gram range and min/max document frequency, as we said in 2.4 as well.

3.2. Hyper-parameter tuning

In this phase, we tried various hyper-parameter configurations to optimize the performance of the logistic regression model and try to confirm the theory behind each hyper-parameter. The key hyper-parameters tuned included based on lecture slides:

- **Learning rate - Solver**: In logistic regression, the solver implicitly manages the learning rate as part of its optimization algorithm, we used 'liblinear' because it had better score than 'saga' and 'lbfgs'.
- **C (Inverse of regularization strength)**: Explored values were 0.1, 1, 2, 5. Best value was 1 via manual hyper-tuning and grid search. When we used significantly small "C", the underfit phenomenon was clear in learning curves. For big "C", overfitting popped up, as we expected.
- **Max iterations**: The idea here was (according to Lecture 2- Regression Slides to have a large number of iterations and interrupt the algorithm when the gradient vector becomes less than the tolerance. Best values were max_iter = 1000 and the default value of tolerance = 0.0001.

Note: Random state was used so as to ensure the reproductability of our results.

3.3. Optimization techniques

To summarize the main optimization techniques we used, we have:

- **Feature Selection**: A parameter with huge impact. We observed how different values of max_features affected performance. Specifically, increasing the number of features improved accuracy up to a point, but a very high number of features caused overfitting. We finally selected 30,000 features as the best balance between performance and generalization.
- **Use of Balanced Class Weight**: Even though our dataset was perfectly balanced, we tested the use of the class_weight='balanced'parameter to see if it improved stability. The results were similar to not using this parameter, indicating no class imbalance issues.
- **Grid Search for Parameter Tuning**: Grid Search was another technique we attempted for tuning logistic regression hyperparameters. Since Grid Search uses cross-validation on the training set to evaluate parameter combinations, it sometimes overfits to these internal folds. This was clear in our case, where the best parameters from Grid Search didn't lead to the best validation set performance. In contrast, manual hyperparameter tuning, with the validation score as our baseline, resulted in better performance. This highlights that relying only on Grid Search might not always lead to the most generalizable results.
- **Learning Curve Analysis**: By plotting the learning curve, we monitored the training and validation accuracy as the number of training samples increased. This helped us detect signs of overfitting or underfitting and adjust parameters like C and max_features accordingly.

Overall, these optimization techniques were crucial in refining our model. By focusing on feature selection and carefully tuning hyperparameters, we achieved a strong performance with minimal risk of overfitting.

3.4. Evaluation - Best Trial

From the metrics we know from lecture-homework slides (Accuracy, Precision, Recall, F1-Score), we have as a baseline accuracy ("measures the proportion of correctly classified instances out of the total instances"), since the dataset is perfectly balanced. We could focus in F1-Score (the balance of precision and recall), if the dataset was imbalanced. The classification report on validation dataset (which was our baseline, as instructed in Piazza) can be seen below:

```
Accuracy on Validation Set: 0.80460421
Classification Report on Validation Set:
              precision    recall  f1-score   support

           0       0.81      0.80      0.80     21197
           1       0.80      0.81      0.81     21199

    accuracy                           0.80     42396
   macro avg       0.80      0.80      0.80     42396
weighted avg       0.80      0.80      0.80     42396
```

3.4.1. Learning Curve. This model was chosen **as Best Trial** since we maximized our accuracy score. Below, in **??**, you will find the comparisons with the other two approaches depending on feature selection.

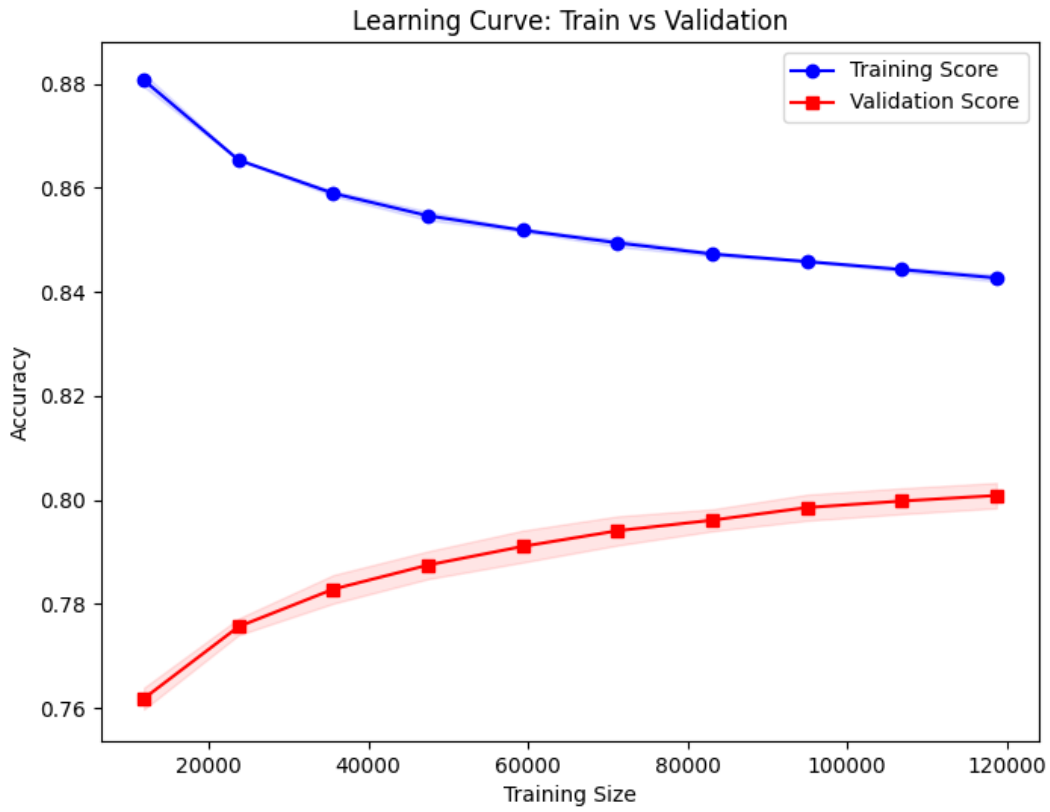
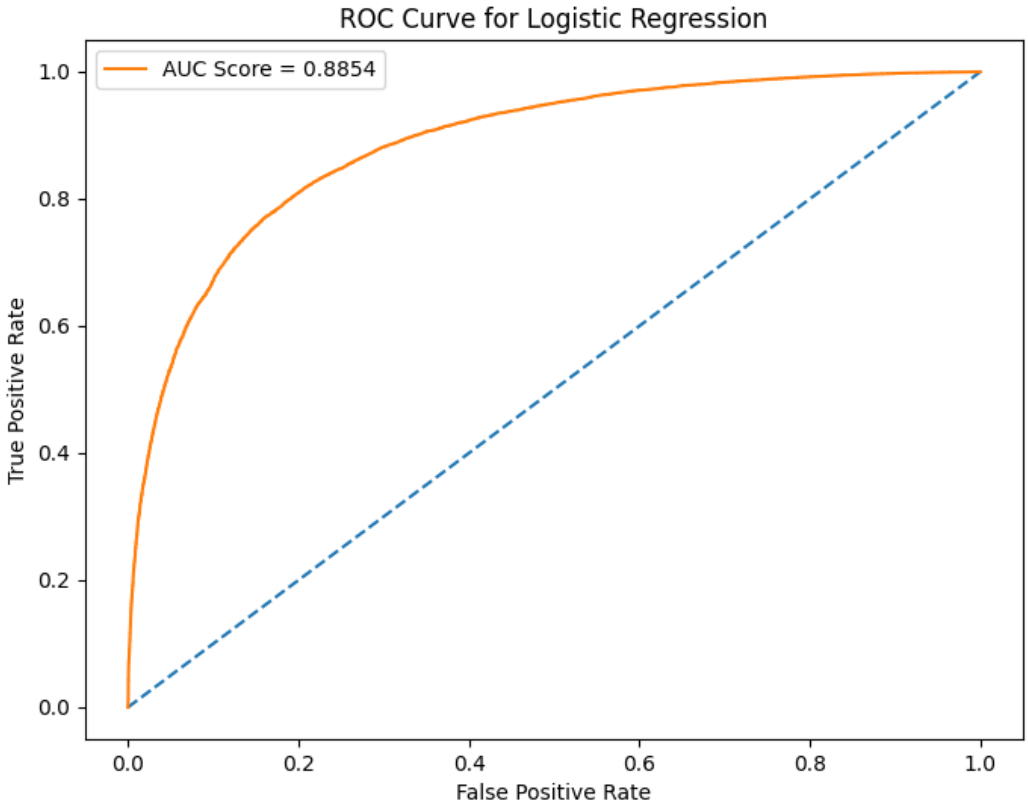


Figure 3: 30k features Learning Curve

3.4.2. ROC Curve. Through web research and a really useful video I found for ROC Curves and AUC Score tutorial, (Mentioned in Bibliography) I realised the vital role of them. The ROC curve and AUC score are really useful tools for understanding how well our model is doing. The ROC curve shows us how good our model is at telling the difference between positive and negative sentiments. It does this by plotting the true positive rate against the false positive rate. The closer this curve gets to the top-left corner of the plot, the better our model is performing. Now, the AUC score is basically just a single number that summarizes the ROC curve. It stands for "Area Under the Curve" and ranges from 0 to 1. A score of 0.5 means our model is no better than random guessing, while a score of 1 would be a perfect model (which I guess is pretty much impossible in real life). In our case, we got an AUC score of 0.88, which is pretty good! It tells us that our model is doing a solid job of distinguishing between positive and negative sentiments. It's way better than random guessing, which is always nice to see. I saw that for an excellent project, the AUC score is minimum 0.9, so we may have room for improvement but for first time, it is a pretty decent score!



4. Final Comparisons

4.1. Eliminated Approaches - Decision Making

Now, we will see the approaches that were eliminated for the one we discussed above (3.4) and their curves. The differentiation is based on feature selection. With 10k features, the gap between the scores is minimized as you can see below, so less risk for overfit, but the accuracy score is lower. The choice of 20k features we could say that it is a balance between overfit risk and accuracy. To summarize, we went **all in** for accuracy and we selected 30k features. With more features, we were far off our overfit risk preferations.

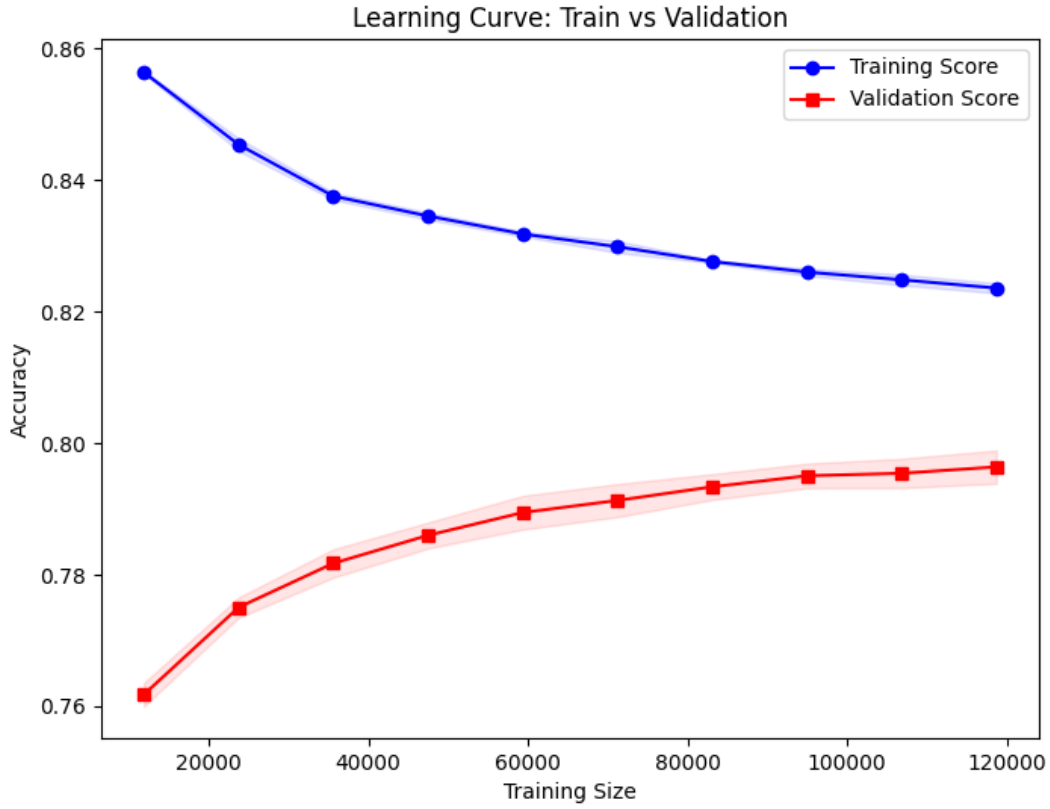


Figure 4: max_features = 10k

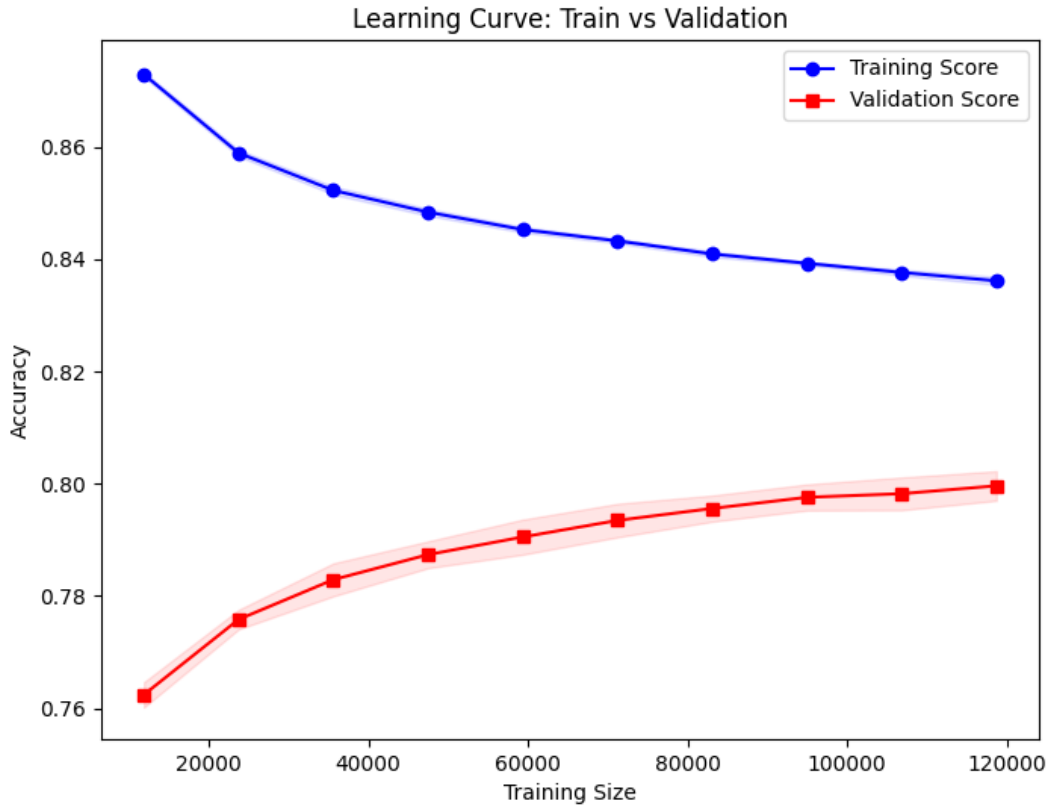


Figure 5: max_features = 10k

5. Bibliography

References

[1] Manolis Koumparakis. Artificial intelligence ii, deep learning for natural language processing, advanced artificial intelligence. https://cgi.di.uoa.gr/~ys19/#homework, 2025.

[2] Manolis Koumparakis. Introductory concepts of machine learning. 2025.

[3] Manolis Koumparakis. Regression. 2025.

[4] Yorgos Pantis. Homework 1. 2025.

[5] Shaul ES. Exploratory data analysis for natural language processing: A complete guide to python tools. https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools, 2023.

[6] GeoDev. How to generate a roc curve with sklearn: A tutorial for machine learning beginners | geodev. https://www.youtube.com/watch?v=rQq5UgZ_H2c, 2023.

[7] GeeksForGeeks. What is sentiment analysis? https://www.geeksforgeeks.org/what-is-sentiment-analysis/, 2024.

[1] [2] [3] [4] [5] [6] [7]