UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Dimitrios Tsiompikas*
sdi: 7115112300036

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

Contents

1. Abstract

Goal of the first assignment is to train and evaluate a Logistic Regression model to identify sentiment (POSITIVE, NEUTRAL or NEGATIVE) of tweets based on a Tweet dataset from the 2023 Greek Elections. I will fulfill this task by following a classic ML process which is: Firstly clean the dataset from data that might harm our train , test and valid sets. Then create some visualizations to get a grasp of what I'm dealing with , using tokens , the sentiment column , lemmas etc. Train the model with the techniques mentioned below (vectorizers, hyperparameter tuning, feature extraction etc) and finally evaluate the model with metrics and plots.

2. Data processing and analysis

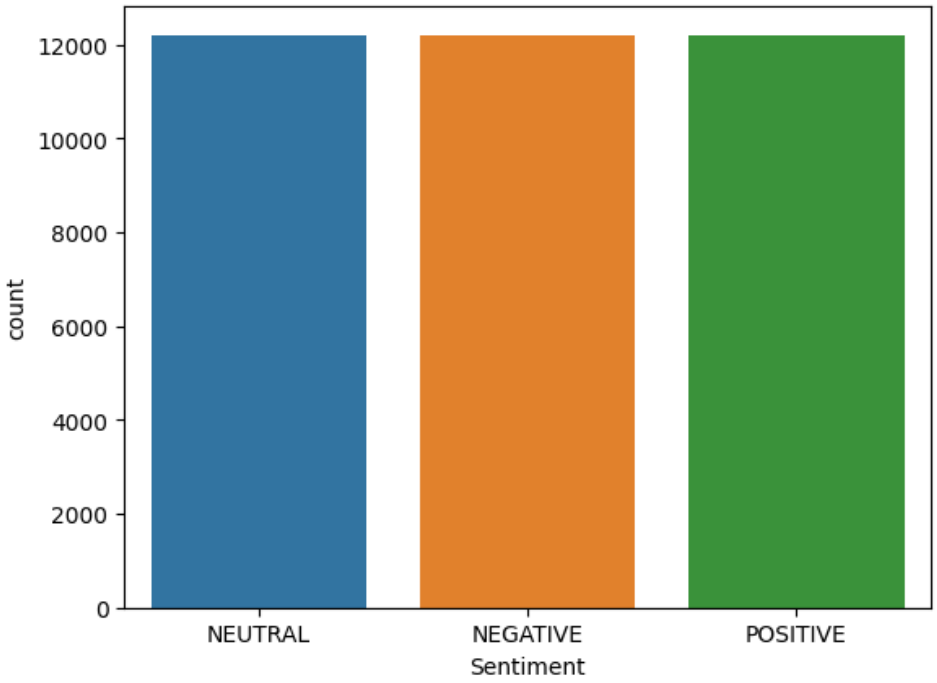
2.1. Pre-processing

For data pre-processing, I firstly checked for null values to clean them , there were none so I followed the cleanup with a hand-made tweet cleanup function that removes all mention signs , hashtags , links (URLs), punctuations and all words after mentions or hashtags (example: @akis18 , I removed akis18) as they were in English and would cause trouble with model training. I also created a stopwords list in Greek , removed all useless Greek words ,then I added English words in the list as well (taken from the tweets of all datasets) and removed them as well from the tweets in order to keep the model from having to deal with words in two languages. Lastly, I removed special symbols to further improve my model training and get the best possible results. In this part, I noticed that the dataset has some issues that cannot be fixed in such a large amount of data like wrong sentiment in specific tweets or sometimes even the wrong party. This will cause problems later on with overfitting and keeping the scores on low percentages.

2.2. Analysis

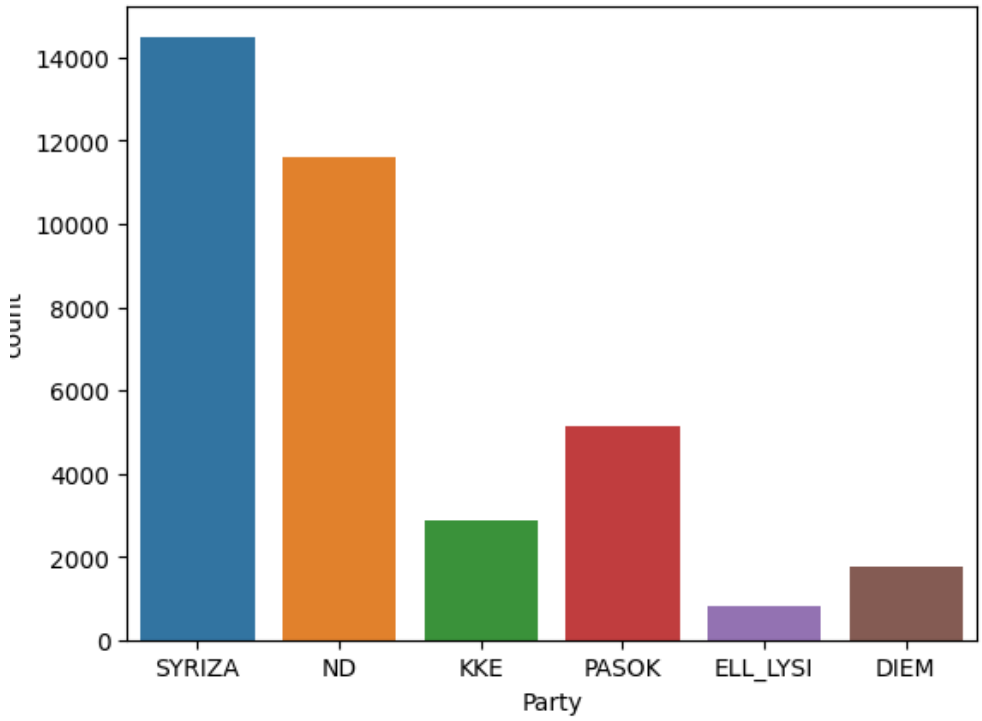
Firstly, I began with analyzing the dataset columns:

Figure 1: Sentiment countplot



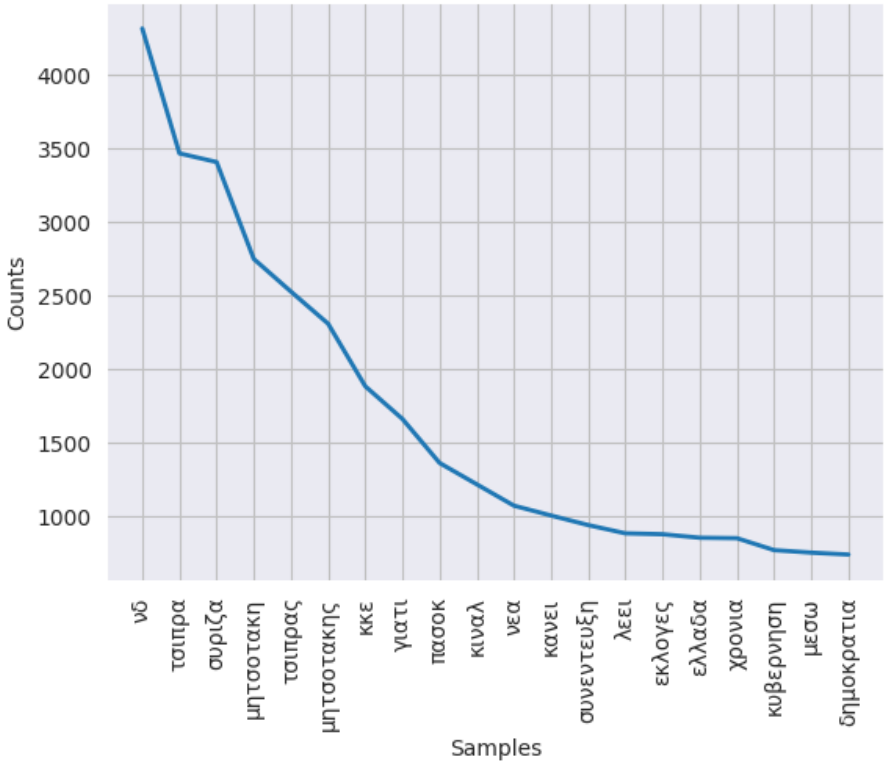
We have the same count for all sentiments.

Figure 2: Party countplot



Here, we can see that the most prevalent party in the tweets is SYRIZA which makes sense since SYRIZA was the political opposition for ND.

Figure 3: Word frequency diagram from Text column



From this diagram we can see the most frequent tokens in the Text column. We can see that the first ones are related to Alexis Tsipras, Kyriakos Mitsotakis , ND and SYRIZA as they were the two main competitors for the 2023 elections. Afterwards, parties with lower percentages follow such as KINAL and KKE and some other words related to elections or politics.

Figure 4: Word cloud of Text column

As we can see in the above word cloud , the most prevalent words are related to the top Greek parties (SYRIZA, ND , PASOK/KINAL and KKE) and of course the main political competitors of the 2023 Greek elections, Kyriakos Mitsotakis and Alexis Tsipras. There are some mentions of Giannis Varoufakis , the leader of DiEM 25 and Fofi Genimmata , the previous leader of KINAL. Some other words include synenteyxi (interview) , ekloges (elections) and kivernisi (government) which all relate to the elections. The rest are mostly verbs and some other words that aren't really useful.

2.3. Data partitioning for train, test and validation

I've used the train set for training , test set to predict the labels and valid set to create the metrics.

2.4. Vectorization

For vectorization I tried CountVectorizer and the TF-IDF vectorizer. It appears that TF-IDF yields the best results without of course changing the final metrics too much due to dataset issues (1 percent difference). Below, in the Evaluation part, you will find metrics showcasing their performance. I have also noticed that Count Vectorizer takes much more time to complete the fit-transform process. TF-IDF took 1 minute to complete the whole Kaggle cell while Count Vectorizer took 5 whole minutes to finish with the same steps in each cell!

3. Algorithms and Experiments

3.1. Experiments

Firstly, the algorithm that was used in the experiments in order to train the model was Logistic Regression with the LBFGS solver since it was the best one from different runs. I began with a Brute-force experiment (without pre-processing) used the train set to create the X-train and y-train sets. Then I used the test set to create X-test. I also

created X-valid and y-valid arrays using the valid set for evaluation (text for X-valid and sentiment for y-valid). For the X sets I used the Text column as the feature, and the Sentiment column as the label for the y sets.

Then, I used both count vectorizer and TF-IDF in separate experiments to see the difference in performance. TF-IDF was superior both in time and in metrics so I continued with that. After the brute force experiments , I added the pre-processing techniques mentioned above , cleaning all tweets and re-training the TF-IDF vectorized model again. This yielded much better results and this paved the way for the next experiments. I tokenized and lemmatized the text from the train set using spaCy (for lemmatization) and NLTK (for tokenization). I used both as features instead of text but they produced worse or similar results and since Lemmatization took a considerable amount of time to be executed I put it in comments in the code. One final try I did was using the Party column along with the Text column as features (using hstack function from scipy) without producing any different results. Therefore, I continued my experiments using Text as the feature. I also used cross-validation in some runs to also create the learning curves shown below.

Lastly, as I will mention below , I used Optuna to calculate the best hyperparameters and find the optimal metric score. Optuna helped a lot actually and yielded the best possible results without of course raising the score a lot due to dataset issues. In the table below, the score described is the F1-Score with micro average (similar to Accuracy metric). Optuna value are the best hyperparameters I found using optuna for tol and C, you can see more about these in subsection 3.2.

Trial	max-iter	tol	C	Score
Brute-force count vectorizer	2000	default	default	38.5%
Brute-force TF-IDF vectorizer	2000	default	default	39.4%
Lemmas as feature	2000	default	default	38.9%
Party-Text combined as features	2000	default	default	38.3%
Tokens as feature	2000	default	default	38.8%
Pre-processed data with TF-IDF vectorizer	2000	default	default	39.8%
cross-validated run	2000	default	default	37.9%
cross-validated run (best params)	2000	optuna value	optuna value	39.7%
Pre-processed data TF-IDF (best params)	2000	optuna value	optuna value	40.4%

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning

I used the Optuna library to tune the model's hyperparameters. I created an objective function that trains a Logistic Regression model with my data and through a number of trials finds the best hyperparameters for tolerance and C (inverse of regularization strength). Due to problems in the dataset there is still some overfitting but the hyperparameters fix some of it. After many trials, The best values Optuna found for these two params are the following

$$tol = 0.0007294770991206959$$

$$C = 0.06169882722761676$$

Providing the best score which was 40.4%.

3.3. Optimization techniques

I used Optuna mostly to find the best hyperparameters and cross-validation using train-test-split as optimization techniques. They didn't produce tremendous difference though.

3.4. Evaluation

The predictions were evaluated using the valid set sentiment column and I used the following metrics:

Classification report: this produces several metrics all-in-one , namely Recall,Precision,F1-score and Accuracy.

Recall: this shows how many true results are returned from the predictions.

Precision: this shows how relevant are the results from the predictions.

F1-score: this uses recall and precision to find out how accurate the model is with its predictions. It ranges from 0 to 1 and the closer it is to 1 the better.

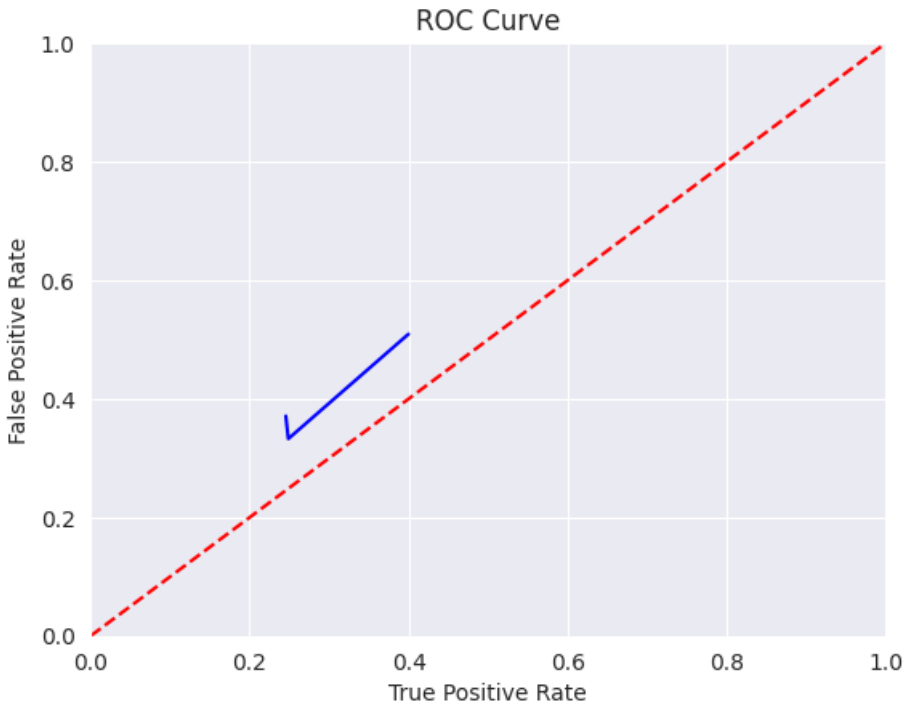
Accuracy: this calculates the accuracy of the model by calculating the fraction of number of true results to the number of total results.

Confusion matrix: This is a table that shows the True positive , true negative, false positive and false negative counts of values that the model has found. It's used to calculate the metrics above.

Below I will add figures showcasing each of the results. Also I will add the results of the brute-force experiments:

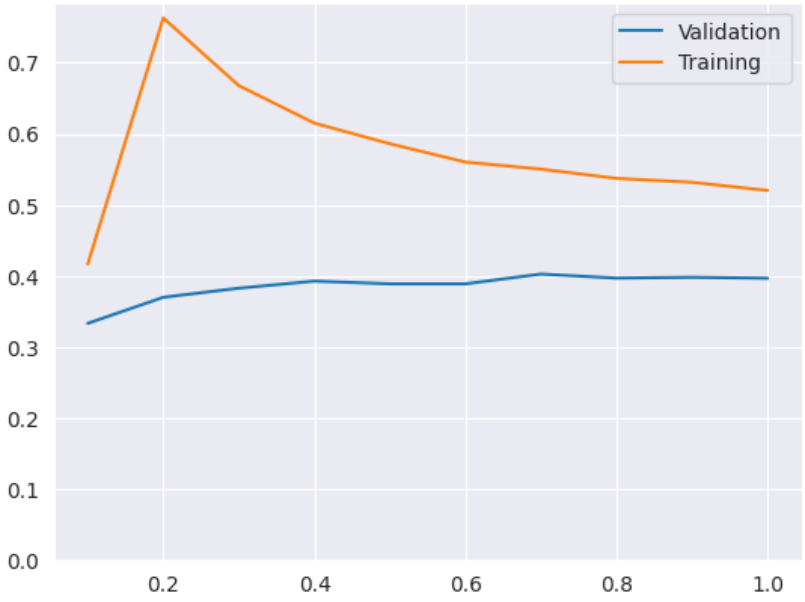
3.4.1. ROC Curve.

Figure 5: ROC Curve with best hyperparams



3.4.2. Learning Curve. Here we can see the overfitting of the model even with best hyperparameters. However, it is most likely that with more iterations of training, the model could converge to the Training data at some point.

Figure 6: Learning curve with best hyperparameters



3.4.3. Confusion matrix.

Figure 7: Confusion Matrix with best hyperparameters (best trial)

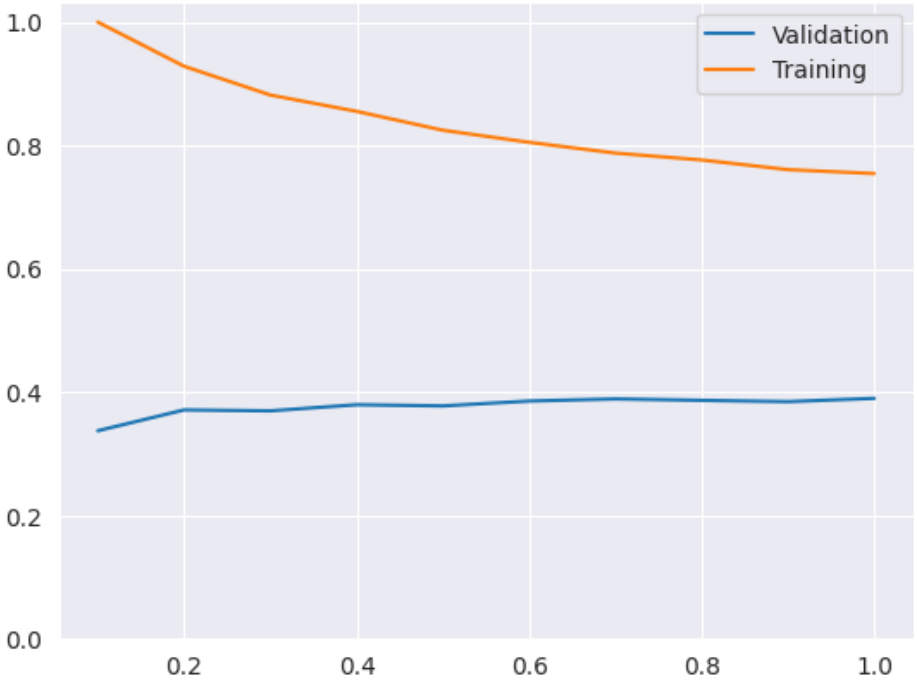
```
[[888 448 408]
 [716 580 448]
 [678 419 647]]
```

4. Results and Overall Analysis

4.1. Results Analysis

In conclusion, the results of the experiment were pretty underwhelming. Since the dataset problems were beyond repair the model only produced results up to 40.4%. On average, classification models usually produce around 85-88% with proper tuning and data cleanup. So the expected results would be in that amount of percentages. Some more techniques I could have used include: Hashing vectorizer instead of TF-IDF to see what it would produce, Stemming (although I did lemmatization) in order to see if it would produce any better results than Lemmatization, Dimensionality reduction, batch partition and Gradient descent. I believe I did a handful of experiments taking note of each percentage so the only thing I'd do additionally is using the techniques mentioned previously.

Figure 8: Learning curve for TF-IDF preprocessed model



Here we can notice the massive overfitting that happens without adjusting the hyperparameters. This is due to the dataset problems mentioned above.

Figure 9: Brute-force no preprocessing count vectorizer run metrics

```
X_train shape: (36630,)
X_test shape: (10470,)
(10470,)
(5232,)
Model training complete.


---------------- BRUTE FORCE METRICS (COUNT VECTORIZER) ----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.38      0.37      0.37      1744
     NEUTRAL       0.39      0.40      0.40      1744
    POSITIVE       0.39      0.39      0.39      1744

    accuracy                           0.39      5232
   macro avg       0.39      0.39      0.39      5232
weighted avg       0.39      0.39      0.39      5232

accuracy:  0.38551223241590216
f1:  0.38551223241590216
total f1:  [0.37039187 0.39561675 0.39017341]
```

Figure 10: Brute-force no preprocessing TF-IDF vectorizer run metrics

```
X_train shape: (36630,)
X_test shape: (10470,)
(10470,)
(5232,)
Model training complete.


---------------- BRUTE FORCE METRICS (TF-IDF) ----------------------
              precision    recall  f1-score   support

    NEGATIVE       0.38      0.40      0.39      1744
     NEUTRAL       0.40      0.39      0.39      1744
    POSITIVE       0.40      0.39      0.40      1744

    accuracy                           0.39      5232
   macro avg       0.39      0.39      0.39      5232
weighted avg       0.39      0.39      0.39      5232

accuracy:  0.3937308868501529
f1:  0.39373088685015295
total f1:  [0.39013453 0.39365994 0.39754816]
```

Here we see the difference of performance between count and TF-IDF vectorizers. TF-IDF was better in this experiment so I used it to run my best trial.

4.1.1. Best trial. Below, you will find my metrics for my best trial. Essentially, after testing the problem thoroughly , I concluded in using the following techniques: TF-IDF vectorizer, pre-processing as mentioned in 2.1 and Optuna hyperparameter tuning. Best f1-score resulted in 40.4%.

Figure 11: Best trial classification report

	precision	recall	f1-score	support
NEGATIVE	0.39	0.51	0.44	1744
NEUTRAL	0.40	0.33	0.36	1744
POSITIVE	0.43	0.37	0.40	1744
accuracy			0.40	5232
macro avg	0.41	0.40	0.40	5232
weighted avg	0.41	0.40	0.40	5232

Figure 12: Best trial accuracy and f1-score with micro average

```
accuracy:  0.40424311926605505
f1:  0.40424311926605505
total f1:  [0.44113264 0.36352241 0.39852171]
```

Figure 13: Confusion Matrix with best hyperparameters (best trial)

```
[[888 448 408]
 [716 580 448]
 [678 419 647]]
```

5. Bibliography

References

[1] lucidv01d. Calculate metrics through confusion matrix. https://stackoverflow.com/questions/31324218/scikit-learn-how-to-obtain-true-positive-true-negative-false-positive-and-fal, 2017.

[2] Milind Soorya. Introduction to Word Frequency in NLP using python. https://www.milindsoorya.com/blog/introduction-to-word-frequencies-in-nlp#data-processing, 2021.

[3] Yue Sun. Optuna hyperparameter tuning with Logistic Regression. https://www.kaggle.com/code/yus002/logistic-regression-optuna-tuning, 2020.

[4] Tutorialspoint. Python - Tokenization. https://www.tutorialspoint.com/python_text_processing/python_tokenization.htm#:~:text=In%20Python%20tokenization%20basically%20refers,for%20a%20non%2DEnglish%20language, unknown.

[3] [2] [4] [1] [Optuna hyperparameter tuning with Logistic Regression] [Word frequency with NLTK] [Python - Tokenization] [metric calculation through confusion matrix]