UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Pavlos Ntais*
Classifier: Feed Forward Neural Network

Course: *Artificial Intelligence II*
Semester: *Fall Semester 2023*

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 . . . 4
2.4 Vectorization . . . 4

3 Algorithms and Experiments **4**
3.1 Experiments . . . 4
3.1.1 Table of trials . . . 4
3.2 Hyper-parameter tuning & Optimization techniques . . . 5
3.3 Evaluation . . . 5
3.3.1 Curves . . . 5

4 Results and Overall Analysis **7**
4.1 Results Analysis . . . 7
4.1.1 Best trial . . . 8
4.2 Comparison with Softmax Regression . . . 8

5 Bibliography **8**

1. Abstract

The task given is to develop a sentiment classifier using Feed Forward Neural Network for a twitter dataset about the Greek general elections. The classifier deals with 3 classes: POSITIVE, NEUTRAL and NEGATIVE. I plan to analyze the data, clean it by pre-processing and finally start the training process.

2. Data processing and analysis

2.1. Pre-processing

The pre-processing proved to be a rather difficult task because of the vastness of the Greek language. After experimenting I came up with the following methods:

- Remove hashtags
- Remove mentions
- Remove URLS
- Remove excess tabs and white spaces
- Remove numbers and special characters
- Remove stop words
- Remove accents
- Basic emoji replacement
- Insert the party the tweet is being referred to at the beginning of the text
- Lemmatization using the spaCy library

2.2. Analysis

Upon visualizing the dataset, it became apparent that the data had a substantial degree of noise. For example spelling and grammar errors, mentions etc. Also, it naturally contained a lot of stop words, hash tags and emojies which also add noise. So, in order to process the data and remove the noise I removed stop words, hash tags and emojies and used lemmatization to group together inflected or variant forms of a word.

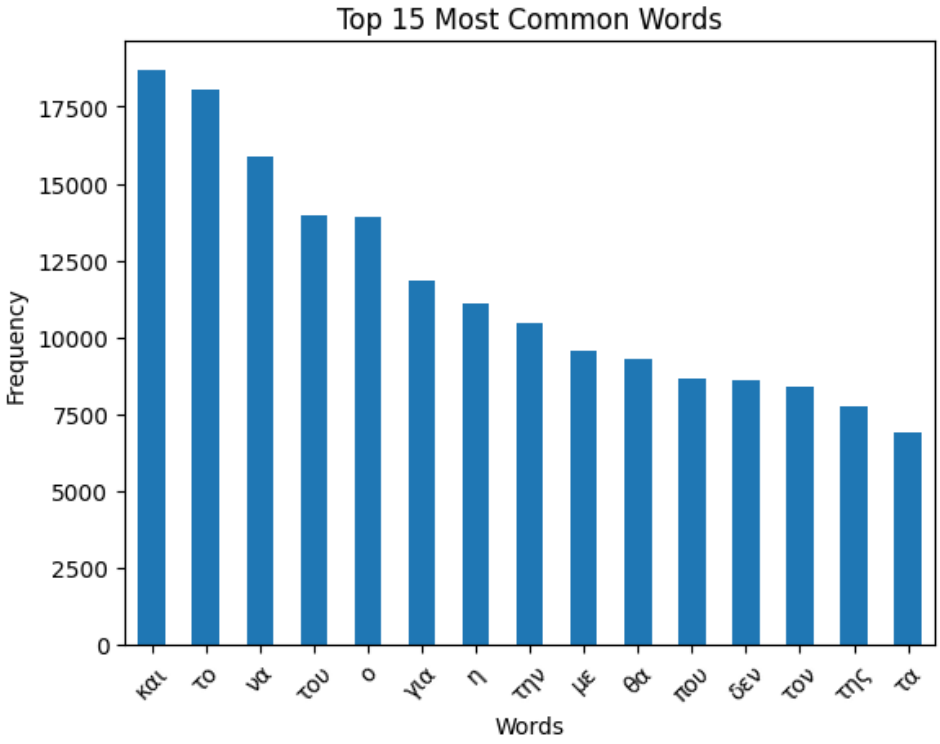

Figure 1: Before pre-processing

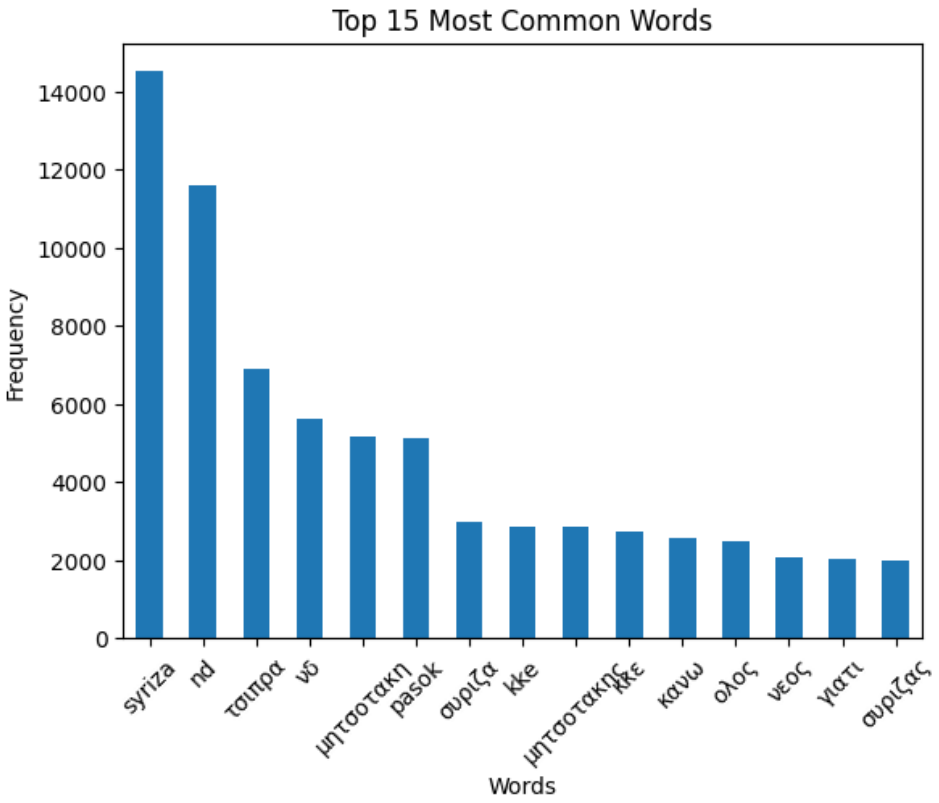

Figure 2: After pre-processing

2.3. Data partitioning for train, test and validation

For the training and testing I used the given datasets because I evaluated that their respective size is sufficient, so no partitioning occurred.

2.4. Vectorization

The vectorization technique used is based on pre-trained Word2Vec embeddings. The embeddings used were found on the fasttext website.

3. Algorithms and Experiments

3.1. Experiments

I experimented using a different number of hidden layers and neurons per layer using pytorch neural networks. To be specific:

- nn.Linear is used to define the network's layers.
- nn.ReLU() is used as an activation function.
- nn.Dropout is used as a dropout layer.

The optimizers tested were:

- Stohastic Gradient Descent
- Adam

Notes:
Overall, due to the faster learning process and higher f1 score Adam proved to be the better choice.

Early stopping stopping is also used in order to stop the training process if no improvements have been made during a certain number of epochs, when it comes to the loss.

Trial	Precision	Recall	F1 score
1 HL (Linear Activations) & SGD	0.3688649644502	0.3660168195718	0.342679295283818
2 HL (ReLu Activations) & Adam	0.3696615006065	0.3685015290519	0.358405247151357
3 HL (ReLu Activations) & Adam	0.4175924447030	0.4023318042813	0.376216626008775
4 HL (ReLu Activations) & Adam	0.7779479326186	0.3348623853211	0.169902157634727
Optuna 1	0.4420720885747	0.3935397553516	0.350504850282233
Optuna 2	0.3991988742038	0.3971712538226	0.390645372781579
Optuna 3	0.4240774425540	0.3998470948012	0.367559379319100
Final Model	0.3999113385520	0.3988914373088	0.391157635045650

HL = Hidden Layer(s)

Table 1: Trials

3.1.1. Table of trials.

3.2. Hyper-parameter tuning & Optimization techniques

For optimization I used the Optuna framework, an open-source tool designed for automating hyperparameter optimization. I primarily fine-tune my model running it, ultimately enhancing its performance. Specifically, ran optuna 3 times, each time reducing the search range in order to find the hyperparameters that give the best results. In the optimization function we tune:

- The number of hidden layers.
  The optimal number of layers seemed to be either either three (3) or four (4). Three (3), though, seemed to yield better results.
- The number of neurons per layer.
  We begin with a wide range of neurons then narrow down to a specific range where the best results came from.
- The learning rate.
- The epochs.

3.3. Evaluation

I am evaluating the predictions using 3 curves:

- ROC curve
  The ROC curve is a graphical representation of a binary classification model's ability to distinguish between positive and negative classes.
- Loss learning
  Loss is the difference between the predicted values of a model and the actual ground truth labels, guiding the optimization process to minimize this discrepancy and improve the model's performance.
- Learning rate
  A learning curve visualizes how a model's performance improves with more training data using the fscore. Fscore is a performance metric that combines precision and recall into a single value.

I also provide the precision & recall scores. The most important thing I'll be taking in mind though, since neither precision nor recall seem to be our priority, is the fscore. It provides a balance between these 2 and makes the model more "generic".

3.3.1. Curves. Here are the curves of the trials showed above:
You have to zoom in order to see the results clearly.

At first, we will be testing models using a custom number of layers and neurons per layer (no optimization).

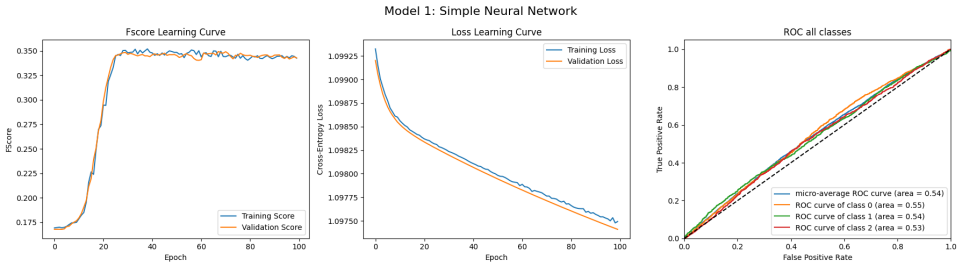

Figure 3: Model 1: 1 hidden layer

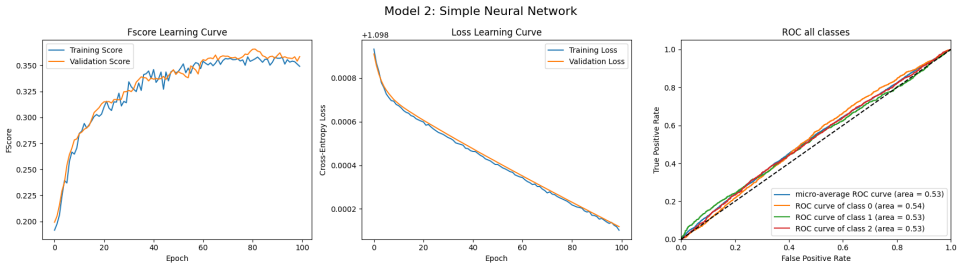

Figure 4: Model 2: 2 hidden layers

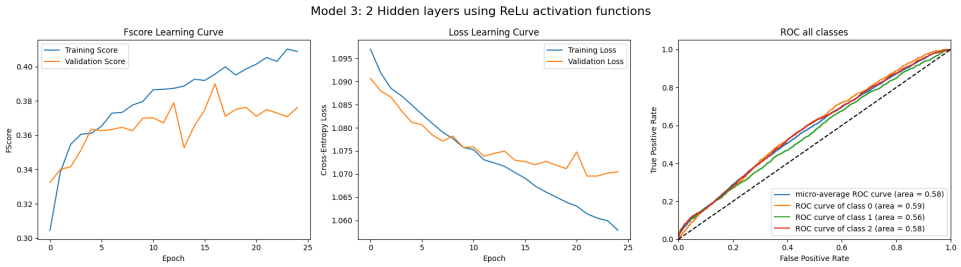

Figure 5: Model 3: 3 hidden layers

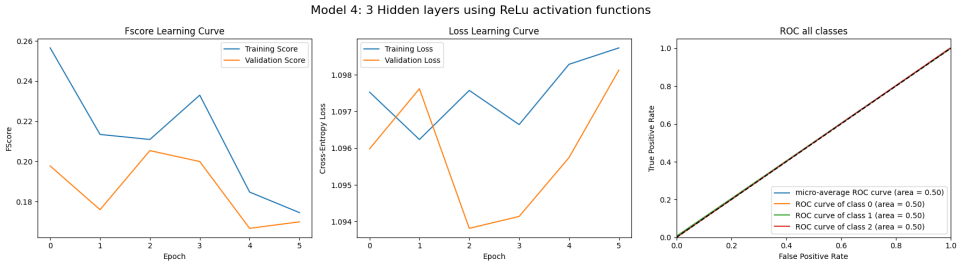

Figure 6: Model 4: 4 hidden layers

Now we will be running Optuna to hopefully optimize our model.

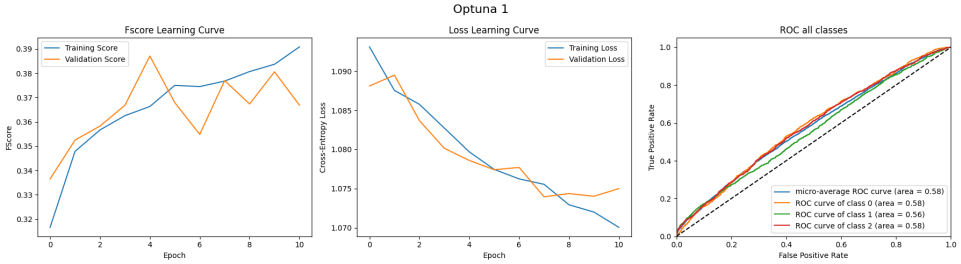

Figure 7: Optuna 1

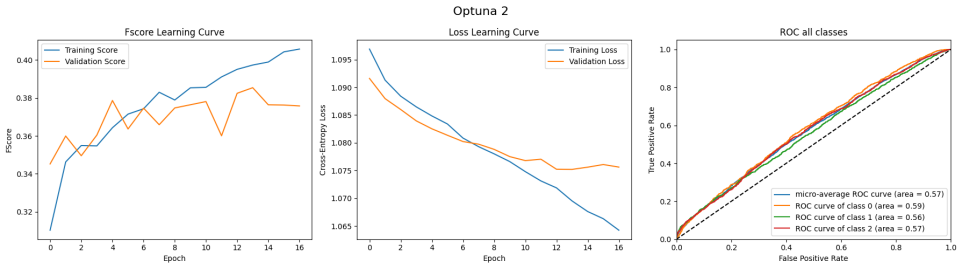

Figure 8: Optuna 2

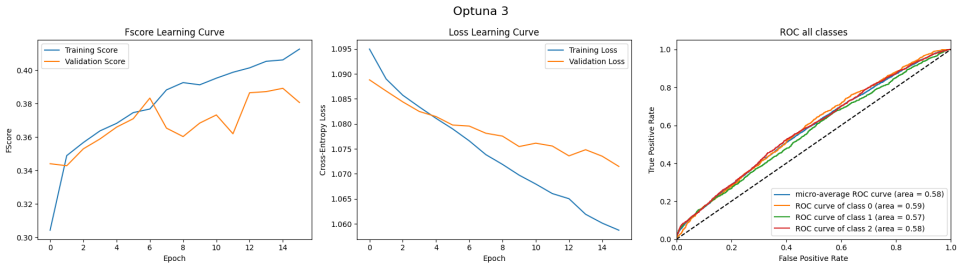

Figure 9: Optuna 3

4. Results and Overall Analysis

4.1. Results Analysis

Overall, although the performance of the model fell short of my expectations, there are several factors contributing to this outcome:

- The dataset presented inherent challenges due to its Greek language content. Greek, being linguistically complex and distinct from widely-used NLP languages, posed inherent difficulties for the model. Also, the limited availability of good performing Greek NLP libraries posed a significant challenge.
- The dataset itself may have contributed to the suboptimal performance. A good number of labels were not correct thus hindering the ability of the model to learn.

4.1.1. Best trial. Ultimately my best trial came after running hypertuning optimization. The model uses the Adam optimizer, using 3 layers with [427, 508, 91] neurons per layer respectively.

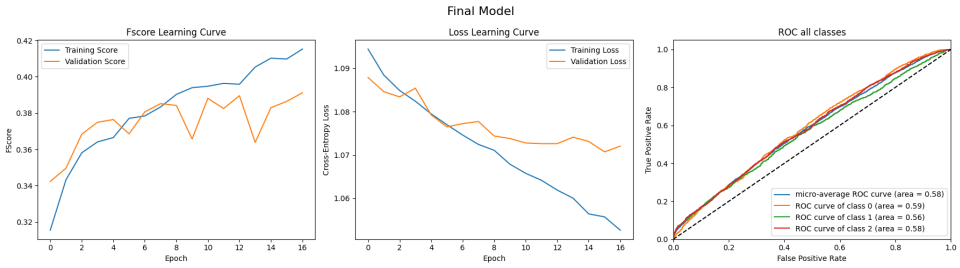

Figure 10: Final Model

4.2. Comparison with Softmax Regression

Overall, in comparison with the Softmax Regression, our model has about the same performance (roughly 0.9% worse).

5. Bibliography

References

[1] Optuna. Optuna hyperparameter tuning. https://optuna.org/#code_pytorch.

[2] Stack Overflow. Stack overflow q&a. https://stackoverflow.com/.

[3] Pytorch. Pytorch tutorial. https://pytorch.org/tutorials/.

[4] Stanford. Speech and language processing. https://web.stanford.edu/~jurafsky/slp3/.

[4] [2] [3] [1]