UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Apostolos Koukouvinis*
sdi: 2000098

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

Contents

1. Abstract

The task is to build a Neural Network for multiclass Sentiment Classifier (NEUTRAL, NEGATIVE, POSITIVE) for greek tweets using pytorch and skelarn. The input of the Neural Network have to be embeddings so we need to find a way to represent each tweet as an embedding. To tackle this we have to pre-process the data, create a Data Loader and a Neural Network, encode the Sentiments using Label Encoder, create a Word2vec model based on our dataset, represent the tweet based on word embeddings from Word2Vec, train the model and predict the test set Sentiments. To do that, we have dataset and validation set, based on these we will try to tune the hyperparameters to improve accuracy. We have chosen to improve accuracy since our dataset is balanced. After we find the best model and the best hyperparameters, we will concatenate the validation and dataset to create the word2vec model and predict the sentiments of the test set.

2. Data processing and analysis

2.1. Pre-processing

Basically the pre-process is simillar to the pre-process of the first assignment without removing special characters, the results we got with them included were better. First of all we lowercase all letters and remove stopwords (we have created a stopword list based on greek frequent words). Then we remove urls. Afterwards we remove acute tones and replace ς with σ and imply stemming using greek stemmer. We also have to add a space between words and special characters like @ and # to treat them as seperate words-tokens. Finally we remove extra spaces and save our clean dataset. We have to mention here that all the steps above were done for both data and validaiton set.

2.2. Analysis

Since the dataset is the same of the first assignment, the analysis is also the same. After analysis on the words frequency, we could not find any relationship between the 20 most frequent words and any sentiment, so we can drop these words or just dont mind them. However, after analysis in the relationship between politcal parties and sentiments, we deducted that some parties have the the tendancy towards a specific sentiment, we will use this fact later. It is also interesting, that after the data cleaning, the top 3 most frequent words per sentiment are common.

2.3. Data partitioning for train, test and validation

Since, as you might have seen in the Jupyter Notebook, the optuna studies are many, we decided not to partition te data using K-Fold or cross validtion. We kept the original partition in data and validation set

2.4. Vectorization

In this project the vectorization may be useful to represent each tweet. I tried Tfidf and count vectorizers, and tried to tuned hyperparameters for them using optuna. However those representation did not yield results as good as the naive calculation of mean embedding. So the method I used for representing each tweet is by calculating and storing its mean embedding after tokenizing the words of its text.

3. Algorithms and Experiments

3.1. Experiments

We have to decide the structure of our model (number of hidden layers, activation functions, use or not dropout), how to represent each tweet, number of epochs, criterion (loss function) optimizer, the size of each embedding, window size at Word2Vec, batch size in data loader, learning rate and the size of each hidden layer. So there are many parameters to tune that will result in many optuna studies. The main idea is to keep some of the parameters fixed and tune the rest of them so we can reject some approaches (for example keep fixed SGD optimizer and tune the hyperparameters and compare it to Admas optimzier to chose the best). Since the model is balanced we are trying to optimize accuracy score. There are mamy optuna studies I created with different set of hyperparameters, so there cannot be a table with fixed columns, the table I am going to present consists of some of the columns and potential extras of each trial (In hidden layer the number of the layers is in the parenthesis, out of parentheseis is the size of the layer, if there are more than 1 layer this size refers to the last layer with the first before last have twice the size, the second before last three times the size and so on. If there is no parenthesis we have only one layer). We will only use Leakyrelu since it is way better than simple sigmoid and relu, wherever not specified it means relu. Furthermore, keep in mind that the results in this table may not represent exactly the current version of the project, we have randomized algorithms so the results differ every time. I am representing the average scores and some representative values of the hyperparameters.

InDim	HidLayer	Window	Batch	Learning	LossFunc	optimzer	Extras	Score
99	24	7	103	0.19	CrossEntropy	SGD	Sigmoid	0.398
181	34	4	33	0.029	CrossEntropy	SGD	relu	0.4013
195	27(3)	4	74	0.103	CrossEntropy	SGD	relu	0.399
73	20(5)	6	41	0.079	CrossEntropy	SGD	relu	0.396
120	23	7	124	0.011	CrossEntropy	Adams	relu	0.40
132	13	7	74	0.035	NLLLoss	SGD	relu	0.396
136	37	6	104	0.046	MultiMargin	SGD	relu	0.406
104	20	9	32	0.022	MultiMargin	SGD	WDecay	0.402
190	35	4	80	0.086	MultiMargin	SGD	Leakyrelu	0.4062
150	22	7	85	0.023	MultiMargin	SGD	tanh	0.4023
60	48(2)	10	86	0.95	MultiMargin	SGD	dropout	0.40
104	50	5	101	0.031	MultiMargin	SGD	dropout	0.407

Table 1: Trials

3.1.1. Table of trials. Lets not get lost here, we keep leakyrelu, MultiMargin, SGD and dropout layer, these are fixed lets try some more extras like different vectorization technique:

InDim	HidLayer	Window	Batch	optimzer	Extras	Score
122	50	8	86	0.036	tfidf	0.400
137	41	5	70	0.047	CountVectorizer	0.401
136	43	10	86	0.026	SkipGram instead of CBOW	0.406

Table 2: Trials

So the optuna trials basically tune the input dimension, the hidden layer, window size and batch size. When we also add the negative slope hyperaparameter we tune this as well. For each optuna study we try different optimziers, activation steps, loss functions, add more features like dropout layer, different vectorization etc. So after all these

3.2. Hyper-parameter tuning

The main framework used for hyperparameters tuning is optuna. Based on the learning curves and the ROC curves of my model, there is no overfitting or underfitting. The results remain bad, close to 0.40 score, but I dont think we should call this underfitting, it is more like due to the mislabeled dataset. Since the thechniques we used like optuna and data loader are randomized I also made some more trials on some of the "good" models I came up with to take somehing like the weighted score, actually I kept the model that performed well most of the times.

3.3. Optimization techniques

As said before, mainly optuna with different hyperparameters and scores. The idea is to keep lets say the optimizer that performs well most of the times and dismiss others, in our case this was the SGD optimizer.

3.4. Evaluation

As said before, the goal was to increase accuracy score. The final evaluation is based on some curves that are seplayed below. The final loss is about 0.65 both in training and validation set. Based on learning curves and roc curves, we can see that there is no overfitting or underfitting since the learning curves are pretty much the same in training and validation. The ROC curves may indicate underfitting since the result are really poor, close to 0.40, however this is perhaps due to the mislabeled dataset and not on the model itself.

3.4.1. ROC curve. The ROC curve is :

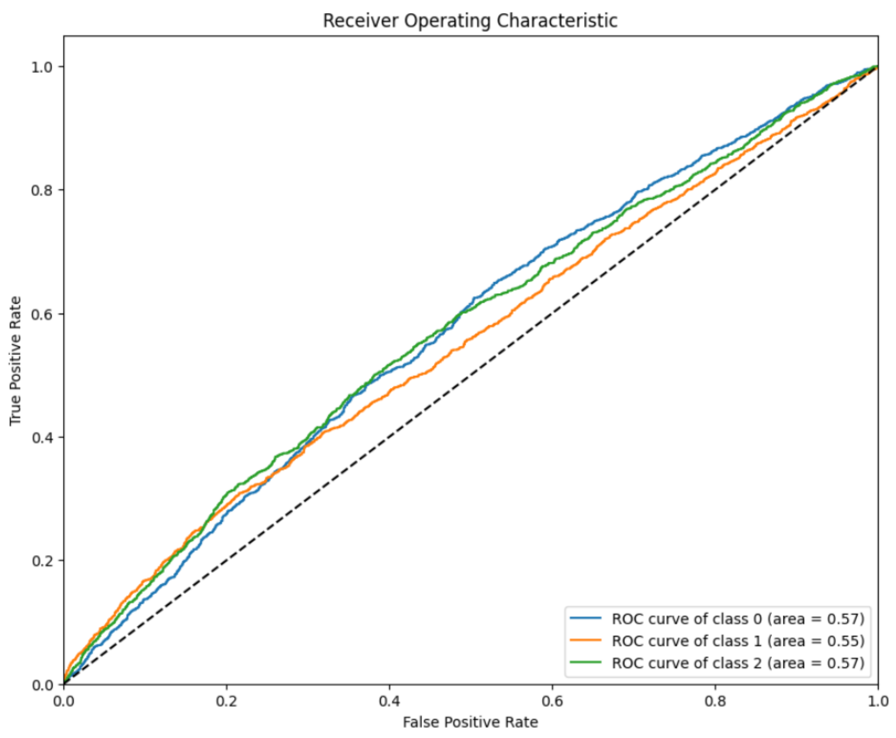


Figure 1: ROC curve

All three classes have an AUC of 0.57, which is slightly better than random guessing (which would have an AUC of 0.5, represented by the dashed diagonal line). An AUC of 1 represents a perfect model, while an AUC of 0.5 represents a model that is no better than random. Therefore, the model does have some predictive capacity, but it is not very strong. The curves are quite close to the diagonal line, which indicates that the model's ability to discriminate between the positive class and the negative class is limited. A more ideal ROC curve would bow significantly towards the top left corner of the plot. Since all three curves have a similar AUC, it suggests that the model has similar performance across all three classes. However, this performance is not particularly high.

3.4.2. Learning Curve. The learning curve is :

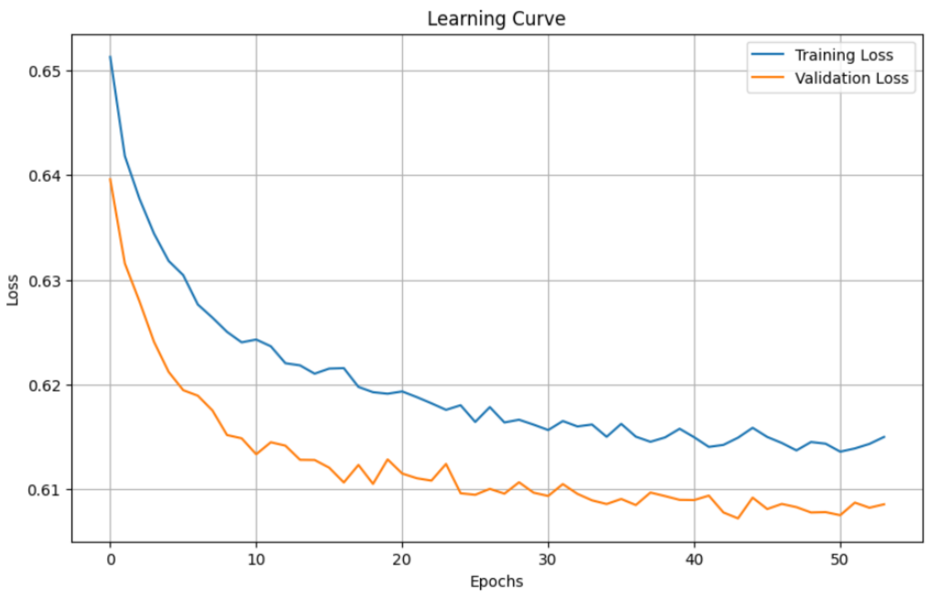


Figure 2: Learning Curve

We can see that both the training loss and validation loss is decreasing, which suggests that the model is learning and improving its predictions over time. After an initial rapid decrease, the rate of decrease in both losses slows down. By around epoch 20, the losses seem to stabilize, indicating that the model may be approaching its performance capacity given the current architecture and data. There is no clear sign of overfitting. Overfitting would be indicated by a decrease in training loss coupled with an increase in validation loss. Since the validation loss continues to decrease (although more slowly than the training loss), it suggests that the model is generalizing well to unseen data.

3.4.3. Confusion matrix. The confusion matrix is :

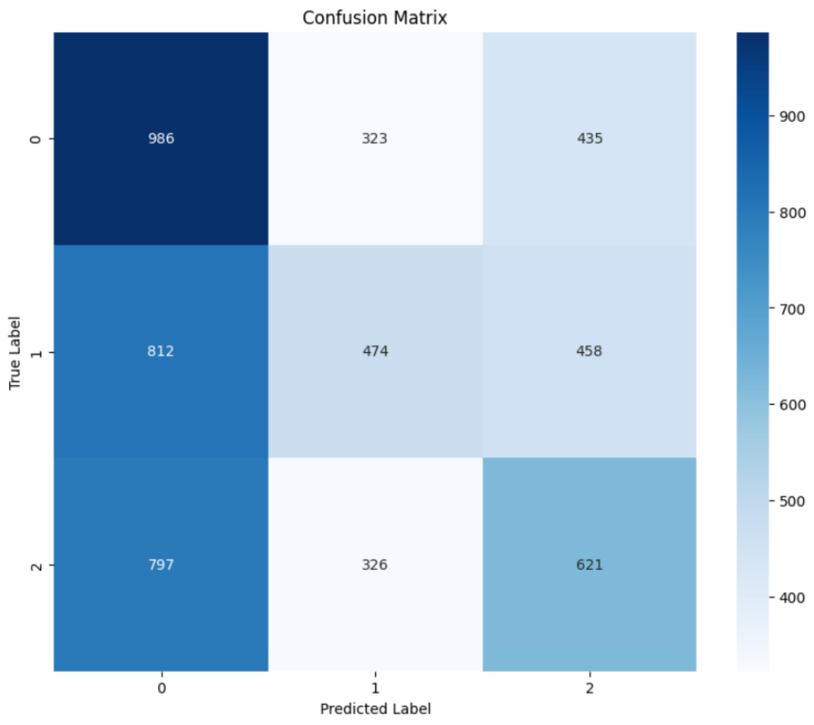


Figure 3: Confusion matrix

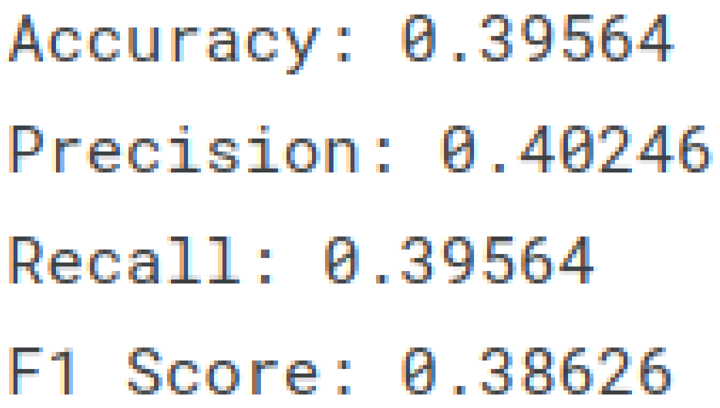
Keep in mind that 0 corresponds to ’NEGATIVE’, 1 to ’NEUTRAL’, and 2 to ’POSITIVE’. The model correctly identifies 986 instances as NEGATIVE. However, there are a significant number of NEGATIVE instances that are misclassified: 323 are incorrectly labeled as NEUTRAL, and 435 as POSITIVE. The model struggles the most with the NEUTRAL class, with only 474 instances correctly identified. There are 812 instances that are actually NEUTRAL but are mislabeled as NEGATIVE, and 458 NEUTRAL instances are mislabeled as POSITIVE. The model performs moderately well in identifying POSITIVE instances with 621 correct predictions. However, there are 797 instances labeled as NEGATIVE that should have been POSITIVE, and 326 instances labeled as NEUTRAL that are actually POSITIVE. Overall Performance: The model is best at identifying NEGATIVE sentiment correctly, followed by POSITIVE, and performs the worst on NEUTRAL sentiment. Perhaps there is a bias towards negative, but we cant really say, since the dataset is not that good.

4. Results and Overall Analysis

4.1. Results Analysis

As said in the begining the data cleaning in this project is different that the data cleaning of the first project. In fact, I have 2 versions of the second project, one with the same data cleaning of the first project and one with the new. The main differences are that in the data cleaning of the first project, I removed all non alphabet characters and words that consist of one letter, however not removing them yield better results, so I kept all the non alphabet characters and all words. Of course, by using "less" data cleaning we get more tokens per sentence, thus we increase time complexity, however, since we do not have time constraints and we want to improve our accuracy, there is no problem doing this. Chosing the best model is kinda hard, since there are many models with pretty much the same accuracy scores. For sure there are some models that we can for sure dismiss, like models using sigmoid (relu is always better) or using mean embedding over all other methods. But for example, using or not weight decay is not clear : sometimes the results are better with it while others not. Same with LeakyRelu vs tanh, MultiMargin vs NLLLoss, SkipGram vs CBOW. However, all the results are always between 0.402 and 0.408, which is not huge difference. Anyways, the final model contains a neural netwrok with one hidden layer of size 10 that uses dropout with dorpout rate equal to 0.20289531425438015 . The acitvation function is leaky relu that uses negative slope equal to 0.2996496033775225. The size of each embedding is equal to 704 with a window of size 3 and CBOW method. For the representative embedding we use the mean embedding of each text. The criterion / loss function is MultiMarginLoss in which Softmax is applied. The learning rate is equal to 0.021794863937645848. A data loader is being used with batch size equal to 85. The optimizer being used is SGD. Also StepLR is being used to adjust the learning rate as loss is changing. Keep in mind, that the results, are from previous optuna studies and not the current, results in the sumbission code will differ. In the end of my project there are some trials with fixed hyperparameters, thse are some of previous optuna studies results. To find the optimal number of epochs to use at testing, I am using early stopping in validation set. Most of the time the early sopping happens at epoch 30 to 50. So the fixed number of epochs in predicting the sentiments of the testing set is 45. Furthermore, while in the validation set predictions, for the word2vec model only the training set is used, for predicting the sentiments of test set I am using the concatenation of training and validation set.

4.1.1. Best trial. The hyperparameters that were used for the best trial are above, the actual scores are :


```
Accuracy: 0.39564

Precision: 0.40246

Recall: 0.39564

F1 Score: 0.38626
```

Figure 4: Best Trail

4.2. Comparison with the first project

The final result were worse than those of the first project. Not with huge differences, but worse. I have no clear explanation why this happens. Perhaps the Neural Network needs more data to train or this only due to the bad dataset. In any case, this was an interesting project and we can see the potential in the NNs.