UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: Μιχαήλ Χρήστος Ναβάρο Αμαργιανός
sdi:2000151

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

Contents

1. Abstract

In this homework, I develop a sentiment classifier using a bidirectional stacked RNNs with LSTM/GRU cells for the Twitter dataset about the Greek general elections. For the implementation I use the machine learning framework PyTorch and the method of Word2Vec word embeddings.

2. Data processing and analysis

2.1. Pre-processing

Data cleaning and regularisation are very important steps since they simplify and help the model distinguish the basic characteristics of each class. Firstly, I process the column with the "Text" according to some rules.

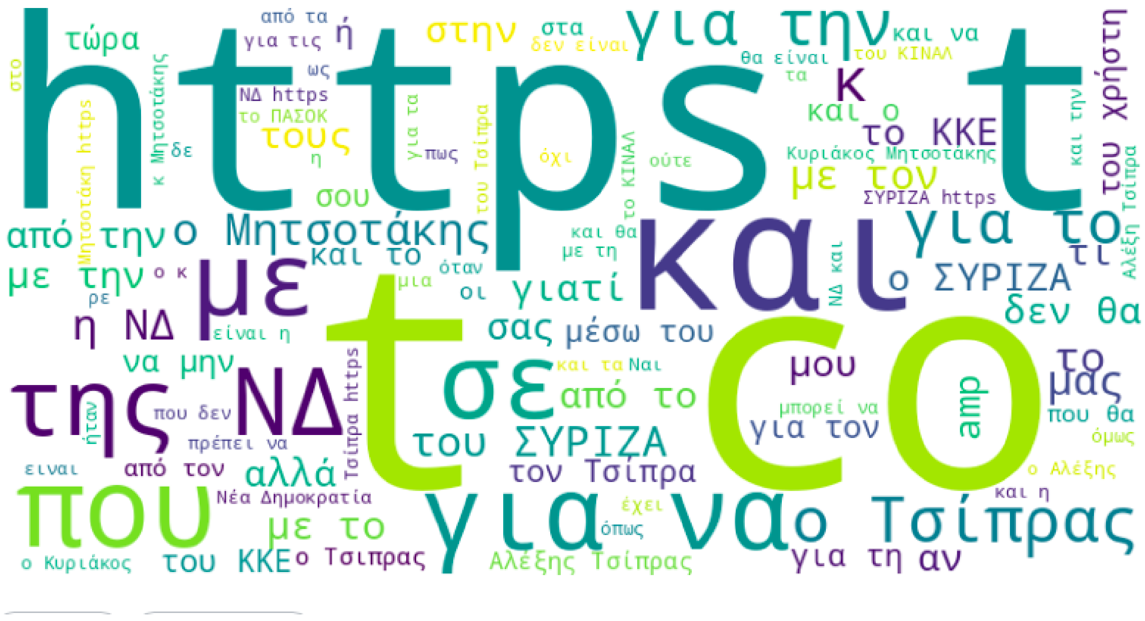
- Removing mentions, hashtags and links. These part of the text are not so important for the final conclusion.
- Converting capital to lowercase letters. Avoiding duplicates and limiting on the ascii codes of only lowercase characters
- Removing accends.
- Removing stop words. (https://www.translatum.gr/forum/index.php?topic=3550.0)
- Removing punctuation.

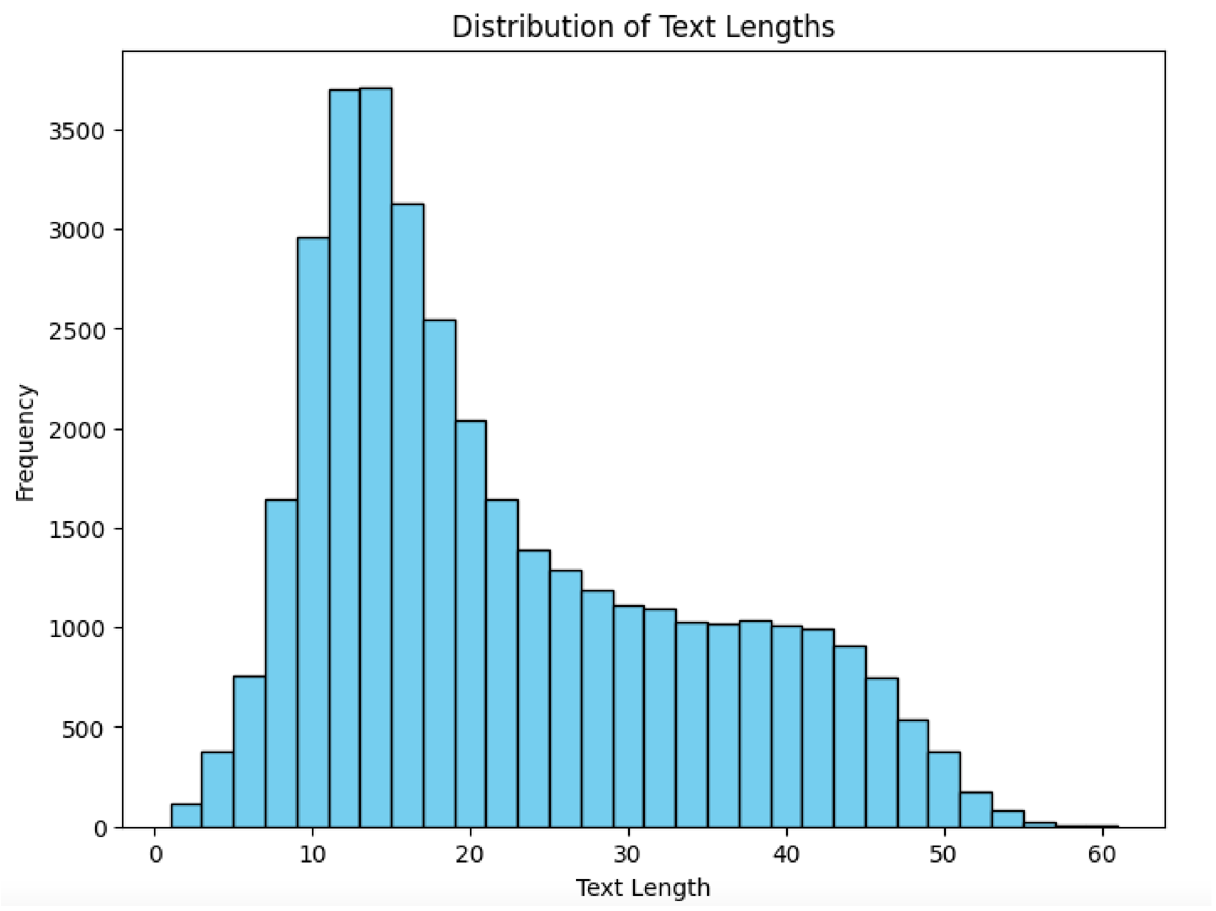
I had as a guide this site to get ideas https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/
For the pre-processed text we will use too the technique of Stemming as described in https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html, although in our problem the text is in greek language so we use "greek_stemmer" library

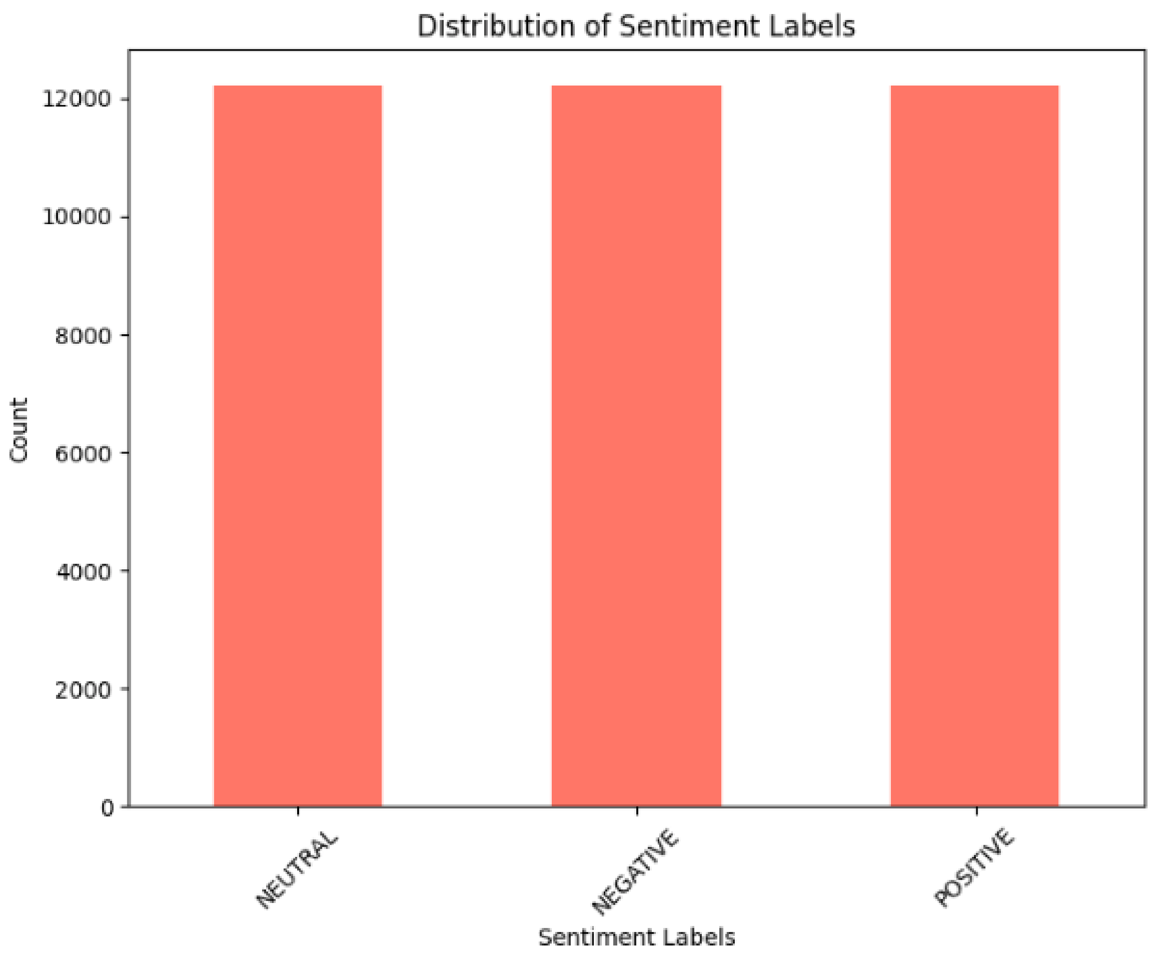
2.2. Analysis

In this step I create word clouds to visualize the most frequent words or terms in the text data and also I use visualizations like bar charts and histograms to display the distribution data and sentiment label.

Word Cloud


https
t
co
και
με
της
ΝΔ
σε
που
για
να
για την
για το
ο Τσίπρας
του ΣΥΡΙΖΑ
ο ΣΥΡΙΖΑ
ο Μητσοτάκης
το ΚΚΕ
με τον
στην
τους
amp
του χρήστη
δεν θα
μου
μας
από το
για τον
τον Τσίπρα
αλλά
με το
από την
με την
η ΝΔ
να μην
σας
μέσω του
γιατί
του ΚΚΕ
Αλέξης Τσίπρας
για τη
αν
τώρα
σου
και το
Κυριάκος Μητσοτάκης
ΣΥΡΙΖΑ https
Νέα Δημοκρατία
ο Κυριάκος
ο Αλέξης


Distribution of Text Lengths
Frequency
Text Length
0
500
1000
1500
2000
2500
3000
3500
0
10
20
30
40
50
60


Distribution of Sentiment Labels
Count
12000
10000
8000
6000
4000
2000
0
NEUTRAL
NEGATIVE
POSITIVE
Sentiment Labels

2.3. Data partitioning for train, test and validation

To manipulate the data for train, test and validation, I create a Y label for the column "Sentiments". Although, this column has valuesof type "string" so I encoded them into type "int" (the encoding was done by using the library "sklearn.preprocessing" and the function "LabelEncoder")

2.4. Vectorization

For passing the "Text" data to the model we have to convert it firstly to a vector. So, for vectorization I choose the technique of Tfidf_Vectorizer of the libraty "sklearn.feature_extraction.text' This is a good method because it's a combiantion of count vectorization with the TFIDF transformer. (https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weightir)

3. Algorithms and Experiments

3.1. Experiments

Experiments have been done in the stage of "Labeling" and "Vectorization".

- I tried to merge the two columns of "Text" and "Party" to give as input to the vectorizer, but finally this experiment didn't return good results.
- Also I experimented by using two differents vectorizers the Count-Vectorizer and the Tfidf-Vectorizer. The conclusion is that in our problem the Tfidf-Vectorizer gives better results.

Also, another experiment is to change the activation function. Instead of use only the ReLU function, I experiment too with the LeakyReLU, ELU, Sigmoid functions.
Finally, the ReLU gives better results

All the experiment are commented in the main code.

3.2. Hyper-parameter tuning

For the Hyper-parameter tuning I use the Optuna framework to search for the optimal hyper-parameters for the model. The objective was to maximize the accuracy of the model on the test dataset.
The hyper-parameters that were tuned are:

- **Hidden Size**: We explored two options for the hidden size of the RNN, 64 and 128. This parameter controls the dimensionality of the hidden state of the RNN, which influences the capacity and expressiveness of the model.
- **Cell Type**: We experimented with two types of RNN cells, namely LSTM and GRU. These cells offer different mechanisms for capturing temporal dependencies in the data, and we sought to determine which one was more effective for our task.

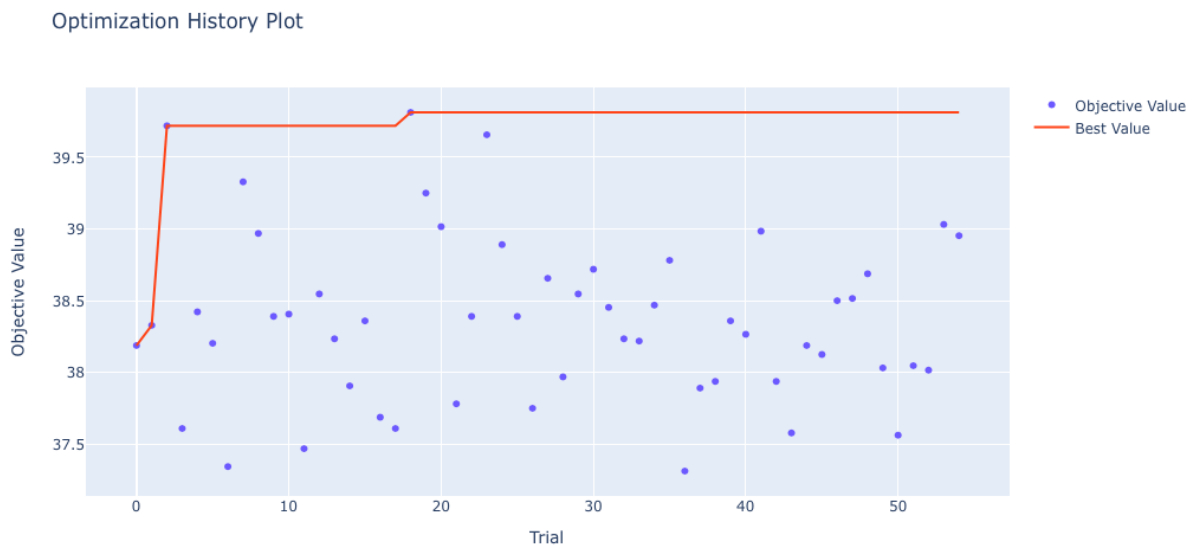
- **Number of Layers**: We varied the number of stacked RNN layers from 1 to 3. Increasing the number of layers allows the model to capture more complex patterns in the data but also increases the risk of overfitting.

- **Dropout** Probability: Dropout regularization was applied with a dropout probability ranging from 0.0 to 0.5. Dropout helps prevent overfitting by randomly dropping connections between neurons during training.

The optimization process involved running a total of 55 trials, where each trial corresponds to a different combination of hyperparameters. For each trial, the model was trained on the training dataset using the Adam optimizer with a learning rate of 0.01 and the cross-entropy loss function. During training, the model's performance was evaluated on the test dataset, and the accuracy metric was used as the objective function for optimization.

The Optuna framework efficiently explored the hyperparameter search space using an adaptive algorithm, aiming to find the configuration that maximizes the accuracy of the model on the test dataset.

Visualizations and analysis of the optimization process:



3.3. Optimization techniques

I apply several strategies to enhance the performance and efficiency of the recurrent neural network (RNN) model for our task. These optimization techniques are crucial for improving training stability, accelerating convergence, and enhancing the model's ability to generalize to unseen data. Below are the optimization techniques used into the model:

- **Adam Optimizer**: We utilized the Adam optimizer, a variant of SGD (Stochastic Gradient Descent), to update the parameters of our RNN model. Adam adapts the learning rates for each parameter dynamically, providing faster convergence and better generalization compared to traditional SGD.

- **Cross-Entropy Loss Function**: For calculating the loss during training, we employed the cross-entropy loss function. This loss function is well-suited for classification tasks, penalizing the model based on the difference between predicted and actual class probabilities.
- **Gradient Clipping**: To deal with exploding gradients during training, we applied gradient clipping. This technique constrains the magnitude of gradients during backpropagation, preventing them from becoming too large and destabilizing the training process.
- **Dropout Regularization**: Dropout regularization was used to prevent overfitting by randomly dropping out neurons during training. This technique introduces noise into the network, forcing it to learn more robust features and reducing reliance on specific neurons.
- **Bidirectional Stacked RNN Model**: I include to the model structure skip connections to capture temporal dependencies in both forward and backward directions. This architecture enables the model to learn from past and future context simultaneously, enhancing its ability to capture long-range dependencies in sequential data.
- **Hyperparameter Tuning with Optuna**: I use the Optuna framework for hyperparameter optimization to find the optimal configuration for our model. Optuna employs efficient sampling algorithms to search for hyperparameters that maximize a given objective function, in our case, test accuracy.

3.4. Evaluation

To evaluate the performance of our model, I utilized a diverse set of evaluation metrics and visualization techniques. The primary objective was to ensure the accuracy and reliability of the model's predictions across different classes. I use precision, recall, F1 score, ROC curve analysis, and confusion matrices as our main evaluation tools, using the scikit-learn toolkit for efficient computation and analysis.

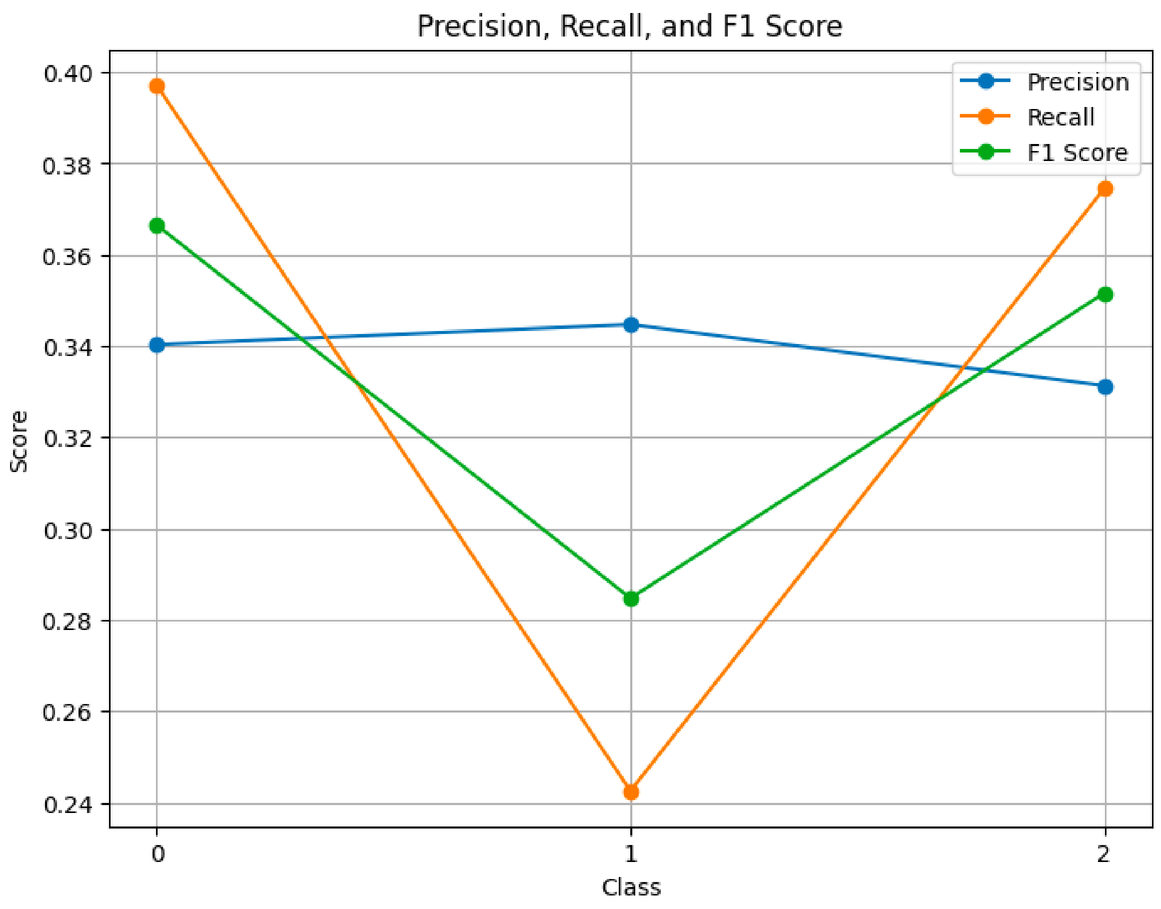
Precision measures the proportion of true positive predictions among all positive predictions made by the model, indicating its ability to avoid false positives. Recall quantifies the model's ability to capture all positive instances within a class, providing insights into its sensitivity to detecting relevant examples. F1 score, the harmonic mean of precision and recall, offers a balanced assessment of the model's performance, particularly useful in scenarios with class imbalance.

I visualized precision, recall, and F1 score for each class to assess the model's performance across different categories. These plots demonstrated the model's ability to achieve high precision, recall, and F1 score values across various classes, indicating robust performance in distinguishing between different categories.

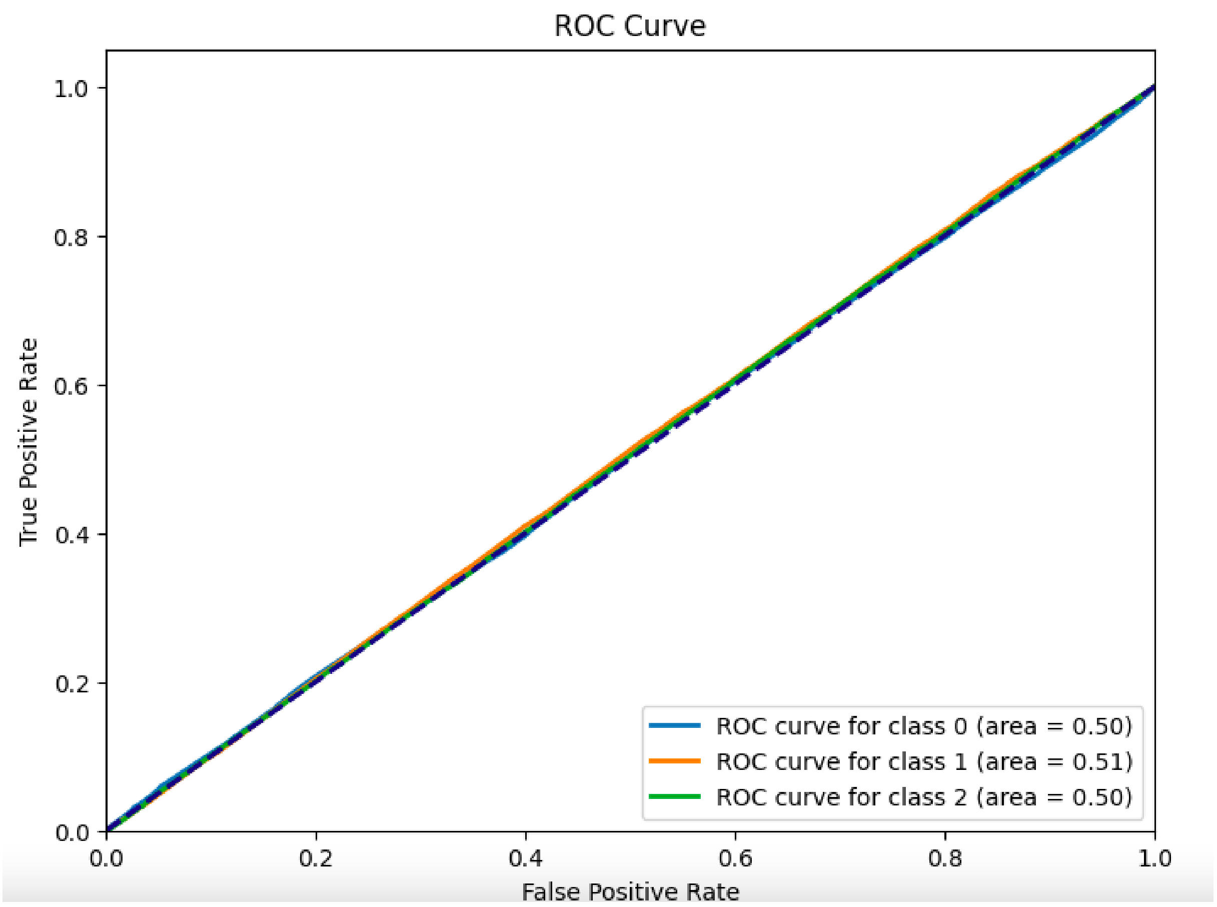
Furthermore, I utilized ROC curve analysis to evaluate the model's discrimination ability across different thresholds. The ROC curves illustrated the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity), providing insights into the model's performance at various classification thresholds.

I used too confusion matrices to visualize the distribution of predictions across different classes. These matrices offered a comprehensive overview of the model's classification performance, highlighting its ability to correctly classify instances into their respective classes while also identifying any misclassifications or areas for improvement.

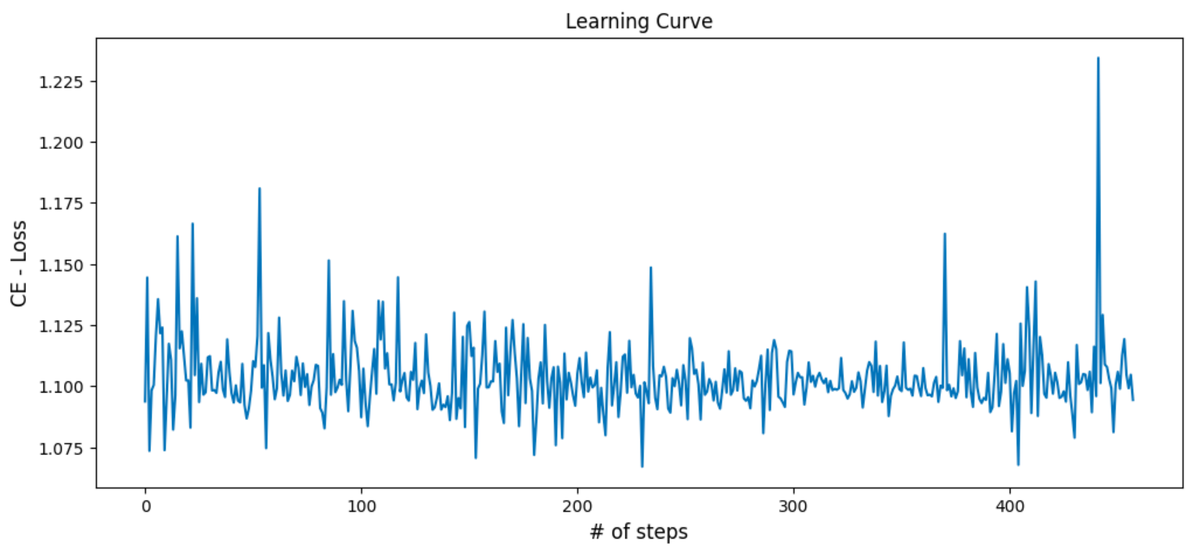
The combination of precision, recall, F1 score, ROC curve analysis, and confusion matrices provided a full evaluation of the model's performance. These evaluation metrics and visualizations enabled to validate the model's efficacy in accurately categorizing instances and making reliable predictions across diverse datasets.



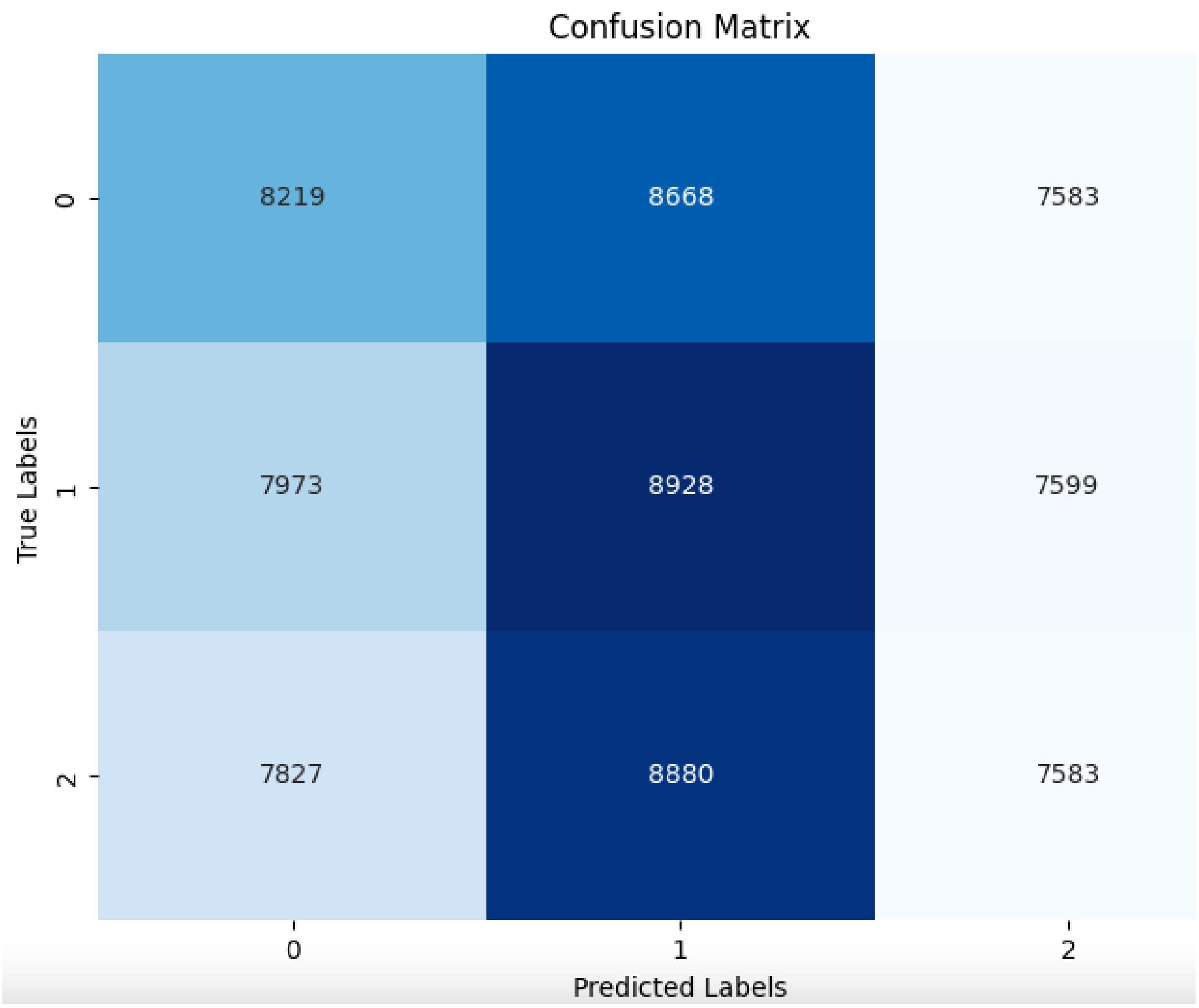
3.4.1. Precision, Recall, and F1 Score.



3.4.2. ROC curve.



3.4.3. Learning Curve.



3.4.4. Confusion matrix.

4. Results and Overall Analysis

4.1. Results Analysis

According to the results the model's performance appears to be moderate.
In more detail:

- Test Accuracy: The test accuracies range from around 37.77% to 39.52% across different epochs. While these accuracies are not exceptionally high, they indicate that the model is learning to make correct predictions better than random guessing.
- Loss: The training loss values range from approximately 1.0873 to 1.1262 across different epochs. The fluctuations in training loss suggest that the model is adjusting its parameters to minimize the loss function during training.
- Precision, Recall, and F1 Score by Class:
  - Class 0: The precision, recall, and F1 score for class 0 are 0.342, 0.336, and 0.339. These metrics indicate that the model has some ability to correctly identify instances of class 0, but it's not optimal.
  - Class 1: For class 1, the precision, recall, and F1 score are 0.337, 0.364, and 0.350. These metrics suggest that the model performs slightly better in identifying instances of class 1 compared to class 0.
  - Class 2: The precision, recall, and F1 score for class 2 are 0.333, 0.312, and 0.322. These metrics indicate that the model's performance is relatively lower for class 2 compared to the other classes.

Overall, the model demonstrates a fair ability to classify instances into different classes, with varying performance across different classes. While the accuracies are not high, they suggest that the model is learning some patterns present in the data
So, according to the results of scores the conclusion is that our model doesn't work very well. The scores we get are very low, that could be caused by a lot of factors:

- **Quality of Text**. Text noise, misspellings, abbreviations, or slang in the Greek language tweets can cause difficulties to do an accurate sentiment analysis.
  Label Quality is also an important factor. The labeling of sentiments could be inaccurate or subjective.
- **Linguistic Complexity**. Greek language is a very complex language with it's nuances, word morphology, or unique sentence structures might require specialized pre-processing or feature extraction techniques.
- The selected machine learning algorithm might not be the most appropriate for sentiment analysis in Greek text, so the model suitability it's not the best.

4.2. Comparison with the first and second project

As I mentioned above using the method of sentiment classifier with a bidirectional stacked RNNs with LSTM/GRU cells gives a test accuracy from around 37.77 - 39.52%.

In the first project I had a test accuracy around 38%, and in the second project around 35-36%.
In conclusion, there isn't an important improvement to our model by trying different methods.

5. Bibliography

5.1. code and slides from the tutorials of the lesson:

https://eclass.uoa.gr/modules/document/file.php/DI517/Tutorials%202023-24/Homework%203/Mnist_Classification.ipynb
https://eclass.uoa.gr/modules/document/file.php/DI517/Tutorials%202023-24/Homework%203/RNNs_LSTMs_GRUs_fromScratch.ipynb

5.2. sites:

https://scikit-learn.org/stable/index.html https://www.translatum.gr/forum/index.php?topic=3550.0
https://www.analyticsvidhya.com/blog/2022/01/text-cleaning-methods-in-nlp/

https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
https://scikit-learn.org/stable/modules/feature_extraction.html#tfidf-term-weighting
https://scikit-learn.org/stable/modules/cross_validation.html
https://www.w3schools.com/python/python_ml_auc_roc.asp
https://www.analyticsvidhya.com/blog/2021/08/understanding-bar-plots-in-python-beginners-
https://www.analyticsvidhya.com/blog/2022/01/tutorial-on-rnn-lstm-gru-with-implementation
https://neptune.ai/blog/understanding-gradient-clipping-and-how-it-can-fix-exploding-gradien
https://engineering.purdue.edu/DeepLearn/pdf-kak/SkipConsAndBN.pdf
https://medium.com/optuna/using-optuna-to-optimize-tensorflow-hyperparameters-57b6d4d31