UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP using Neural Networks

Student name: *Panagiotis Zazos*
sdi: 7115112300009

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

Contents

1. Abstract

The task at hand is to create a sentiment analysis model for Twitter data about Greek Elections in the Greek language. Sentiment analysis involves classifying text into one of three categories: Negative, Neutral, Positive. The objective here is to develop a model which accurately predicts the sentiment of given tweets. To achieve this, a combination of ML and NLP techniques were used.

2. Data processing, Analysis and Preparation

2.1. Pre-processing

The data underwent a comprehensive cleaning and pre-processing to ensure the model's training on high-quality and meaningful text. Through the implementation of the following data cleaning techniques, the text has reduced the irrelevant information, noise and inconsistencies. The following data cleaning techniques were applied:

- **Text Normalization:** Tweets were preprocessed to remove URLs, hashtags, mentions, punctuations and special characters. More importantly, hashtags were split into words according to the underscore character, i.e. "ΝΔ_πολιτικό_σχόλιο" into "ΝΔ πολιτικο σχολιο".
- **Lowercasing**: All text was converted to lowercase to guarantee case insensitivity, ensuring that words were recognizable regardless of their capitalization. This step eliminates the duplication of features as well.
- **Accent Stripping:** Accents from words were stripped to maintain consistency and reduce vocabulary size.
- **Greek Stopwords**: Since nltk's stopwords didn't match the needs of the data, a custom Greek stopwords list was constructed to eliminate common words that do not provide valuable information for sentiment analysis. Since the removal of accents, words in the stopwords list are exclusively without accents, reducing its size by half.
  Note: Additionally, given the prevalence of mixed Greek and English text in Greek tweets, it may be beneficial to explore the inclusion of English stopwords in the pre-processing pipeline to handle such linguistic diversity.

2.2. Analysis

In addition to the initial pre-processing steps, specific adjustments were made to align the corpus with the characteristics of the pre-trained Word2Vec model. **Figure 1** shows the actual coverage of the pre-trained model over the Twitter data about Greek Elections in the Greek language.

Consistency with Pre-Trained Model Vocabulary: The pre-trained model's vocabulary does not employ stemming, therefore in order to maintain consistency, stemming was not applied to the corpus. This ensures that the word embeddings generated by the pre-trained model accurately represent the words in the tweets. The same applies

to lowercasing the corpus in order to match the requirements of the pre-trained model, as capitalized words were out of the vocabulary.

Accent Removal: Despite the pre-trained model's vocabulary incorporating accented words, the decision to strip accents from the tweets were driven by the need to reduce the overall vocabulary size, which can enhance model's performance. Stripping accents can aid the model in generalizing better to new data as well. Lastly, its computational efficient to simplify the text overall.

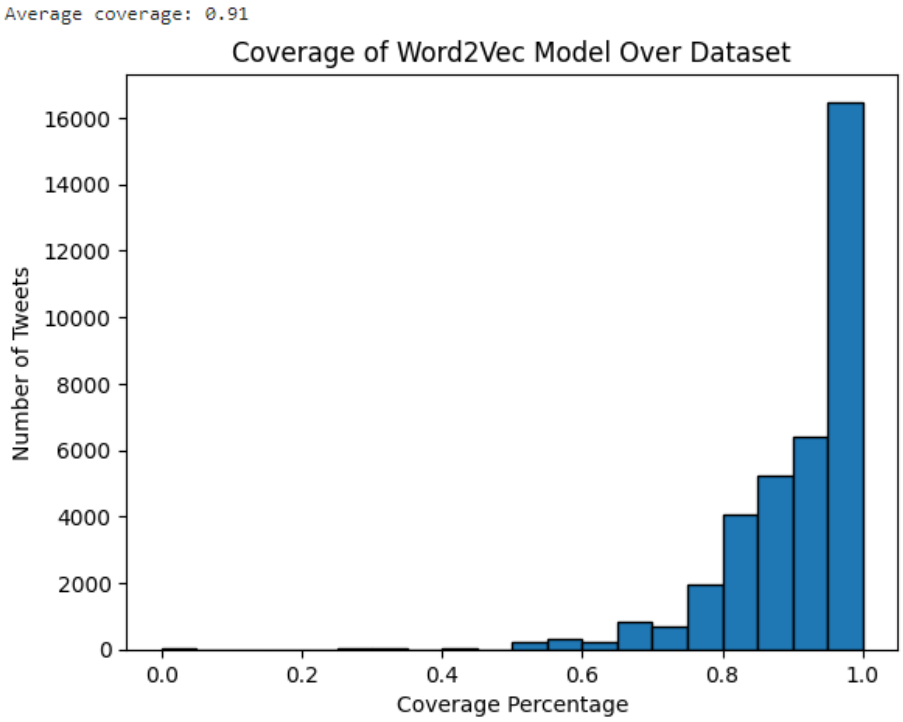

Figure 1: Coverage of word2vec_greek_model Over Dataset

2.3. Data Preparation

Conversion to embeddings: The processed tweets were tokenized and converted to embeddings using the pre-traine Greek Word2Vec model to capture the semantic information and context of the dataset.

Label Encoding: Scikit-learn's LabelEncoder is used to convert categorical sentiment labels (Positive-Neutral-Negative) into numerical form. PyTorch tensors of type torch.long, regarding Sentiment, are created by converting the encoded labels of each dataset. This datatype is selected as it is suitable for classification labels that are used in conjunction with CrossEntropyLoss loss function.

Conversion of Embeddings to PyTorch Tensors: Lastly, the embeddings are converted to PyTorch tensors, which are required for the model's training. The conversion is being achieved by stacking embeddings lists into tensors of type torch.float, which is typical for floating-point numerical computations in the neural network.

3. Model Architecture and Training

3.1. Neural Network Design

Layer Configuration: The model consists of three linear layers. The first two are following by Batch Normalization and Dropout and the third is the output layer.
Batch Normalization is applied after the first and second linear layers to normalize the output of the previous layer, providing improvement to the stability in model's training, as it makes the network less sensitive to the initial weights.
Initially, dropout has been set to 0.5, but after some experimentation, the dropout rate of 0.3 served a better accuracy throughout the epochs. Dropout randomly deactivates a fraction of neurons during training, after the first and second linear layers, which helps in preventing overfitting and basically ensures that the network does not rely on any specific set of neurons.

Activation Functions: ReLU (Rectified Linear Unit) is used after the first and the second linear layers. This activation function offers non-linearity, allowing the model to learn more complex patterns.

Output Layer: The final layer maps the features to the number of sentiment classes. The reason behind the absence of an activation function is that the raw outputs are used in the CrossEntropyLoss function during training.

3.2. Model Parameter Selection and Evaluation

The insights gained from a grid search are instrumental in fine-tuning the model's parameters.

Grid Search for Optimal Parameters: In the pursuit of the optimal hyperparameters for the sentiment analysis model, a grid search was conducted. This approach involved experimenting with commonly used combinations of learning rates, batch sizes and number of epochs:

- **Learning Rates:** 0.01, 0.001 and 0.0001
- **Batch Sizes:** 32, 68 and 128
- **Learning Rates:** 50 and 100

Observations and Decisions: The combination yielding the highest average validation accuracy (approx 0.39) was found with a learning rate of 0.0001, batch size 64 and 100 epochs. However, it was observed that the model's accuracy plateaued fairly quickly in the training phase, in the range of 0.39 to 0.40.
In contrast, a different parameter set comprising a learning rate of 0.001, batch size 128 and 100 epochs, while achieving a similar average validation accuracy (approx 0.387), demonstrated higher ceiling and a way better average accuracy during training, reaching up to 0.47 and an average of 0.45.
The decision to select the latter parameter set (learning rate = 0.001, batch size = 128, epochs = 100) was driven by its ability to achieve higher training accuracy. This indicates a better learning capacity of the model, even though the validation accuracies were similar to both parameter sets. However, one can also address the danger of overfitting on the training corpus. That's a risk that this model's implementation has taken

into account and built upon.
StepLR Scheduler was implemented to dynamically adjust the learning rate by 10% every 20 epochs, offering a more static and naive way to interpret the scheduler. This gradual reduction allows the model to make larger updates to the weights initially and finer adjustments in later stages. Note that there are better and more dynamic ways to compute the step size (frequency) and the gamma (the percentage of decrease) of the scheduler.
Lastly, **Adam** optimizer was chosen for its adaptability and efficiency, as its suitable for common problems in NLP with sparse gradients and noisy data. In contrast to SGD optimizer, the Adam optimizer is less sensitive to the initial learning rate and other hyperparameters. For this, the grid search that was cited earlier did not include many different parameters to tune, only a few of learning rates and batch sizes.

4. Results and Overall Analysis

4.1. Results

Figure 2: Training-Validation Loss Over Time

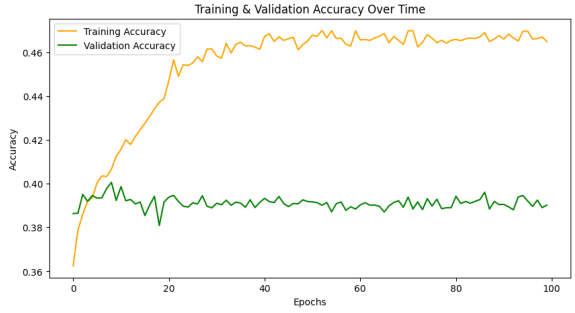

Figure 3: Training-Validation Accuracy Over Time

Training-Validation Loss: While the training loss curve starts quite high, it sees a significant drop in the initial epochs, indicating that the model is learning from the data quite effectively. As the epochs progress, the training loss continues to decrease, until it reaches a point of tiny deviations.
Validation loss on the other hand, does not decrease. It fluctuates around a certain value without a clear downward trend. Therefore, the model is not improving its performance on the validation set.
Training-Validation Accuracy: Training accuracy shows a substantial increase initially and then continues to rise at a slower pace. Thus the model's predictions are becoming more accurate as training proceeds.
On the other hand, validation accuracy quickly plateaus and fluctuates within a narrow boundary. This behavior indicates that while the model is becoming more proficient at classifying the training data, its ability to generalize to unseen tweets is not significantly improving.

4.2. Analysis

To begin with, both graphs suggest the possibility of **overfitting**, where the model performs increasingly well on the training data but fails to show comparable improvement on the validation data. The most potent sign of overfitting is the plateau of the accuracy on the validation set while the training accuracy continues to rise.

4.3. Confusion Matrices

4.3.1. Overview.

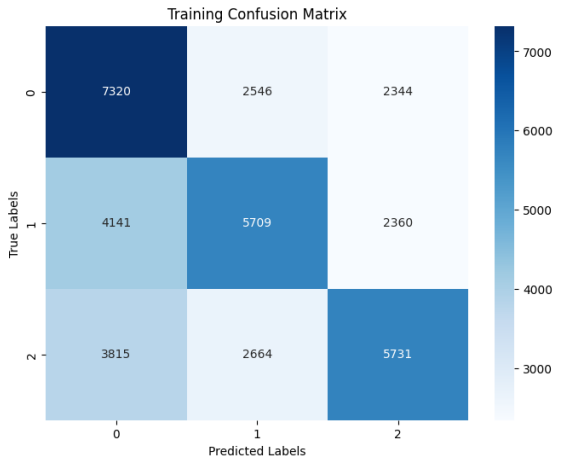

Figure 4: Training Confusion Matrix

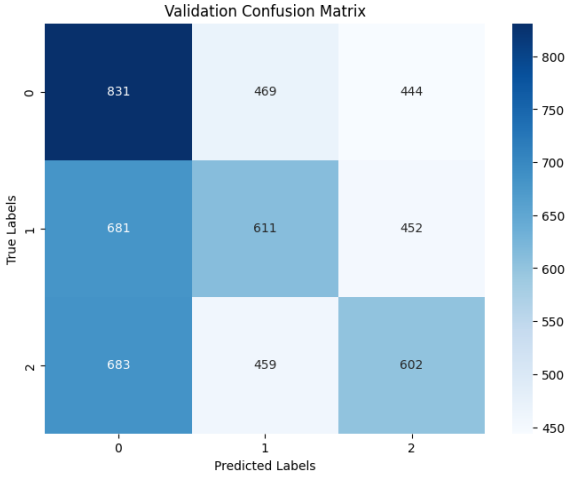

Figure 5: Validation Confusion Matrix

A confusion matrix is a table used to describe the performance of a classification model on a set of data for which the true values are known. Let's preface with the encoding of the categories:

- **0: Negative**
- **1: Neutral**
- **2: Positive**

Diagonal Values (True Positives) for training dataset Figure 4:

- For Negatives tweets, the model correctly predicted 7320 instances.
- For Neutral tweets, the model correctly predicted 5709 instances.
- For Positive tweets, the model correctly predicted 5731 instances.

Off-diagonal Values (Misclassifications):

- Negatives were incorrectly predicted as Neutrals 2546 times, and as Positives 2344 times.
- Neutrals were incorrectly predicted as Negatives 4141 times, and as Positives 2360 times.
- Positives were incorrectly predicted as Negatives 3815 times, and as Neutrals 2664 times.

Diagonal Values (True Positives) for validation dataset Figure 5:

- For Negatives tweets, the model correctly predicted 831 instances.
- For Neutral tweets, the model correctly predicted 611 instances.
- For Positive tweets, the model correctly predicted 602 instances.

Off-diagonal Values (Misclassifications):

- Negatives were incorrectly predicted as Neutrals 469 times, and as Positives 444 times.
- Neutrals were incorrectly predicted as Negatives 681 times, and as Positives 452 times.
- Positives were incorrectly predicted as Negatives 683 times, and as Neutrals 459 times.

4.3.2. Analysis. As mentioned before, the issue of overfitting is can be seen using the matrices as well. This is indicated by the higher numbers on the diagonals of the training confusion matrix compared to those of the validation confusion matrix. There is also the struggle to distinguish features that separate Neutral (1) tweets from Negative (0) and Positive (2). Lastly, the model appears to be biased towards Negative tweets on both training and validation, possibly because Negative tweets have more distinct features and there are more instances of these in the dataset that is provided.

4.4. Possible Improvements

Since the model's accuracy does not improve on par with training, it may be useful to experiment with different hyperparameters, model architectures, maybe a different strategy on layers management or incorporation of regularization techniques. Even a more advanced learning rate scheduler might yield better generalization. The goal here is to achieve better generalization
One more way to improve the model is to adapt a data augmentation technique. It is known that the dataset that has been provided is problematic, due to too many incorrect labels on both training and validation datasets. Data augmentation or the addition of more varied and more accurate tweets with their proper sentiment values could potentially improve the model's generalization capabilities.

4.4.1. Best Scores.

Loss: 1.026, Accuracy: 0.466, Classification Report:

	precision	recall	f1-score	support
0	0.45	0.50	0.48	12210
1	0.47	0.42	0.45	12210
2	0.48	0.47	0.47	12210
accuracy			0.47	36630
macro avg	0.47	0.47	0.46	36630
weighted avg	0.47	0.47	0.46	36630

Figure 6: Training Classification Report

Loss: 1.085, Accuracy: 0.391, Classification Report:

	precision	recall	f1-score	support
0	0.38	0.48	0.42	1744
1	0.40	0.35	0.37	1744
2	0.40	0.35	0.37	1744
accuracy			0.39	5232
macro avg	0.39	0.39	0.39	5232
weighted avg	0.39	0.39	0.39	5232

Figure 7: Validation Classification Report

5. Bibliography

References

[1] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, 1984.

<You should link and cite whatever you used or "got inspired" from the web. Use the / cite command and add the paper/website/tutorial in refs.bib>
<Example of citing a source is like this:> [1] <More about bibtex>