UNIVERSITY OF ATHENS
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS

Deep Learning for NLP

Student name: *Panagiota Gyftou*
sdi: sdi1900318

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

Contents

1. Abstract

The Analysis Twitter Sentiment is a technique for analyzing the sentiment expressed in texts posted on Twitter. Specifically, it leverages data generated by Twitter users and analyzes the emotions that arise from them. In other words, it identifies the sentiment present in a tweet as positive, negative, or neutral.

2. Data processing and analysis

2.1. Pre-processing

- The applied cleaning techniques are as follows:

  1. *Remove Urls* : URLs are not helpful for understanding the emotions of a tweet, without their removal, the model is negatively affected and may generate incorrect predictions.

  2. *Remove Hastags* : The presence of hashtags for sentiment analysis creates noise in the data, so we remove them.

  3. *Remove Tags* : Tags, just like hashtags, also create noise in the data.

  4. *Remove Symbols* : Symbols don't provide information about the emotions of the tweet. Additionally, with their removal, the "Text" becomes more readable.

  5. *Remove Emoji* : Emojis negatively affect the sentiment analysis of a tweet and they are prone to create confusion in the meaning of the text. In other words, the meaning of emojis may contain emotions that do not correspond to the user's actual sentiment. For example, in the text "Smile now ☺ , but there will come a time when you will have to take responsibility for your actions!" we observe that the sentiment is negative and simply inserting a smiling face expresses irony, not something positive.

  6. *Remove Numbers* : Numbers create noise in data. The information they provide is not useful for predicting emotions, their presence only creates confusion.

- Simplification:

By converting all the letters to lowercase, we achieve the accuracy of the results, as if a word appears in the text in different ways, the model can perceive it as the same word. For example, let's say "έλα", "Ελα", "ΕΛΑ" are the same word, just sometimes written in lowercase, sometimes in uppercase, and sometimes with a combination of lowercase and uppercase letters.

- StopWords:

Stopwords are removed, as they do not provide any information. More specifically, they are words that appear very frequently and are only helpful for text structure, not for sentiment prediction.

- Stemming:

With stemming, we manage to convert words derived from the same root into a common form, so that the model perceives them as the same word and not as different words.

(!BERT model) BERT does not use Stemming during its training because it relies on machine learning models that understand the semantic relevance of words in the context of their full environment. While Stemming can help in some cases, BERT's approach focuses on the relationships between words by using the full text without the loss of information that may arise from Stemming.

- Remove words that contain foreign characters:

Such words may be foreign or they may be a Greek word written in greeklish. The stemmer cannot process these words and leaves them as they are. In order to avoid extreme values and incorrect results produced by our model, we remove them.

2.2. Analysis

Figure 1: Word Clouds

```
t[14]:
{'τσιπρ': 8631,
 'μητσοτακ': 7886,
 'ειναι': 6669,
 'νδ': 5541,
 'συριζα': 4890,
 'αυτ': 4138,
 'αλλ': 3113,
 'εχ': 3100,
 'κκε': 2689,
 'ολ': 2423,
 'κυριακ': 2283,
 'πασοκ': 2055,
```

Figure 2: The occurrences of each word

```
{'NEGATIVE': 13954, 'NEUTRAL': 13954, 'POSITIVE': 13954}
```

Figure 3: The occurrences of each sentiment

2.3. Data partitioning for train, test and validation

I use the data as it was given to us in the description.

2.4. Vectorization

2.4.1. Vectorization - Logistic Regression. In order to be able to use machine learning algorithms and, by extension, be able to train machine learning models, the data must be in the form of numerical vectors and not texts. To convert texts into numerical vectors, I use the methods CountVectorizer and TfidfTransformer. Specifically, with CountVectorizer, we create numerical vectors that contain the frequency of occurrence of each word. Then, with TfidfTransformer, we calculate the importance of each word in the text, which represents the size of their frequency of occurrence. The importance of each word is represented by the product of TF and IDF.

- TF : The frequent frequency of a word appearing in the text.
- IDF : The rare frequency of a word appearing in the text.

2.4.2. Vectorization - Feedforward Neural Network. The data vectorization in this feedforward neural network implementation is done using the word2vec model, where each text is transformed into a set of numerical vectors known as word embeddings. In other words, it generates a numerical vector for each word and forms a set of vectors for the entire text.

2.4.3. Vectorization - Recurrent Neural Networks. Like Feedforward Neural Networks, Recurrent Neural Networks also require the conversion of texts into mathematical representations for the training of the neural network. The technique of word embeddings (word2vec) is used for vectorization, which is discussed below.

2.4.4. Vectorization - Bidirectional Encoder Representations from Transformers. Vectorization in BERT is done using a special pre-trained machine learning model (AutoTokenizer). During training, BERT learns to represent each word or token in a high-dimensional dense vector. These vectors represent the text data on which the model is trained. The AutoTokenizer in BERT models is used to create tokenizers that correspond to specific BERT models. The tokenizer is responsible for tokenizing the input text and converting them into representations that the BERT model can understand and process. This is achieved using the token vocabulary created for the specific BERT model.

3. Algorithms and Experiments

3.1. Word2Vec Model

For the creation of the Word2Vec model, the following process is followed:

First, the text is divided into words using the word_tokenize function from the nltk library and stored in a word_set list.

Initialization and training of the Word2Vec model: A new Word2Vec model is created using the data from the word_set. The following characteristics are defined:

vector_size: The number of features or independent variables in each word. In this case, 100 features are emphasized.

window: The size of the window used to predict the next word in a sentence. In this case, a window of size 5 is used.

min_count: The minimum number of times a word must appear in the data to be included in the vocabulary. In this case, each word must appear at least once.

workers: The number of cores used for training the model. In this case, 4 cores are specified.

Then, if there is an empty word in the model's vocabulary, it is removed from the vocabulary.

3.2. Architecture: Recurrent Neural Networks (RNNs)

The architecture of the implemented RNN is a flexible framework that allows for the use of different types of recurrent neural networks (RNNs), such as LSTM and GRU, with the addition of various features such as skip connections and attention.

The RNN class is initialized with various parameters such as the cell type, input size, hidden size, number of layers, number of heads, dropout ratio, output size, whether attention will be used, and whether skip connections will be used.

The most important method is the forward method, which defines how the model's output will be computed. At this point, the input tensor (x) passes through the RNN layer(s), with possible additions of dropout and skip connections, and then goes through an attention layer (if enabled) before the final linear layer that produces the output.

The attention mechanism added to the model is an additional layer that focuses on the important elements of the input during prediction. Typically, in recurrent neural networks (RNNs), the flow of information during processing is concentrated and there is no mechanism to focus attention on specific elements. In this implementation, the

attention mechanism is applied using the nn.MultiheadAttention class of the PyTorch library. This mechanism allows the model to assign different weights to each input, depending on its importance for the current prediction. Then, the important elements are reinforced and taken into account when generating the final output of the model. In the attention mechanism, heads are used, where in a multi-head attention mechanism, the input is subjected to parallel attention operations, and the outputs from these heads are then merged to create the final output. The number of heads, num_heads, is a hyperparameter that determines how many parallel attention operations will be used. This allows the model to focus on different aspects of the input simultaneously and extract richer representations. On the other hand, skip connections provide a bypass path that bypasses one or more layers of the neural network. Typically, bypassing this path allows the inputs of the layers to be accessed in the output of the network without necessarily passing through a series of dense layers. The addition of skip connections can provide the following benefits:

Vanishing gradients prevention: It allows the neural network to propagate information along all paths, regardless of how many layers need to be passed, thus helping prevent the problem of vanishing gradients.

Easier training: Using skip connections can make training more effective, as it can reduce the time required for network training and the number of epochs needed for convergence.

Performance improvement: In some cases, adding skip connections can improve the performance of the model, as it allows for more efficient information transmission through the network.

Overall, skip connections are a powerful mechanism for improving the performance and facilitating the training of neural networks. Finally, the use of bidirectional in recurrent neural networks, such as LSTM and GRU, refers to the ability to evaluate the network input from both directions of the sequence, both from the beginning to the end and vice versa.

Each bidirectional RNN layer consists of two independent neural networks: one that reads the sequence forward (forward RNN) and one that reads the sequence backward (backward RNN). Some advantages of using bidirectional RNNs include:

Better understanding of content: The ability to examine the sequence both forward and backward can lead to a better understanding of content and their connections.

Better performance in classification and prediction: In many cases, using bidirectionality can improve the accuracy of predictions, especially when the sequence contains complex relationships between elements.

More efficient representation: The combined use of information from both directions can lead to richer data representations.

3.3. Architecture:Bidirectional Encoder Representations from Transformers (BERT)

The Bert models are based on a transfer learning architecture that utilizes transformers for training. This architecture includes multiple layers of transformers that process texts at different levels of information abstraction. The key ideas that distinguish the BERT architecture are bidirectional design and the pretraining process on a large amount of text, meaning the BERT model is trained in a pretraining process where it predicts words in a sentence based on the content of nearby words. The 'from_pretrained' parameter is used to load a pretrained Bert model.

3.3.1. GreekBERT. GreekBERT is a model that uses the same structure as BERT but has been specifically trained for the Greek language. The architecture of GreekBERT uses a Transformer for text processing, following the basic principles of BERT. In this specific architecture, the model loads the **"nlpaueb/bert-base-greek-uncased-v1"** model that has been trained on Greek texts. This model is designed for sequence classification and has three output layers (num_labels = 3).

3.3.2. DistilGREEK-BERT. DistilGREEK-BERT is a simplified version of BERT designed specifically for processing Greek text. This is achieved with fewer layers and parameters, making it lighter and faster during training and execution. The idea is to reduce the size of the original BERT without losing much performance, using various techniques. DistilGREEK-BERT maintains the capabilities of BERT, but with fewer resource requirements, making it suitable for applications that require high performance with limited resources. Just like in the architecture of GreekBERT, here the model loads the **"EftychiaKarav/DistilGREEK-BERT"** model that has been trained on Greek texts. This model is designed for sequence classification and has three output layers (num_labels = 3).

3.4. Experiments

3.4.1. Experiments - Logistic Regression. Before training a machine learning model, it is necessary to preprocess the data. Data preprocessing plays a major role in model training as it improves the quality of the data, resulting in accurate and consistent models. Data that has not been processed may contain spelling errors, extreme values, empty values, and other issues that negatively affect the model. Once we have appropriately processed our data, another step before training our model is to modify the labels, in our case, the sentiments. Since we are dealing with regression, the output should be continuous. Therefore, we modify the 'Sentiments' column so that the values are converted into their corresponding numerical representation, as follows:

0: NEGATIVE
1: NEUTRAL

2: POSITIVE

This conversion is done using the LabelEncoder. Finally, we need to modify the texts into numerical vectors, as described in section 2.4. Initially, we run the LogisticRegression algorithm with default values to obtain an initial estimation of our model.

	precision	recall	f1-score	support
0	0.38	0.40	0.39	1744
1	0.39	0.38	0.39	1744
2	0.40	0.38	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

From the Confusion Matrix, we extract the following:

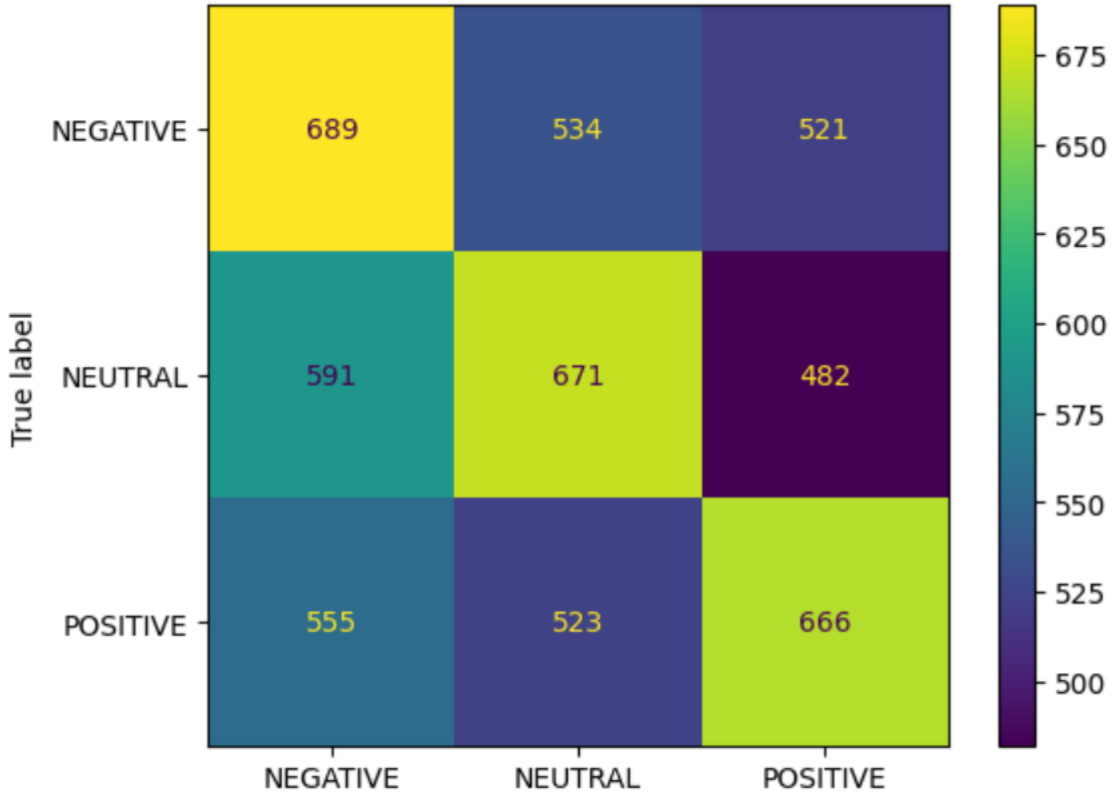


Figure 4: Confusion Matrix: default parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 689

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 555

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 521

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 666

We want TN and TP to be greater than FN and FP, which is the case.

Then we check which is the best segmentation of the texts into n-grams, that is, how many consecutive symbol sequences we take into account when splitting the text. After experimental checks, we observe that setting n-gram = 2 gives us a higher Average Accuracy.

```
Accuracy: [0.38110838 0.38301938 0.37865138 0.38288288 0.38315588]

Average Accuracy: 38.18 %

Average std: 0.17 %


----------------------------------

Accuracy: [0.38138138 0.3961234  0.38301938 0.39312039 0.38479388]

Average Accuracy: 38.77 %

Average std: 0.58 %
```

Figure 5: n_gram = 1 and n_gram = 2

We observe that for n_gram = 2, accuracy increases. This may be because reading specific combined words requires understanding their connected words, rather than individual words.

Finally, we are examining which parameters of LogisticRegression give the highest accuracy. We have arrived at one of the many combinations.

```
LogReg = LogisticRegression(penalty = 'l2',
                            C = 0.1,
                            solver = 'liblinear',
                            multi_class = 'ovr',
                            tol = 0.001,
                            n_jobs = -1,
                            max_iter = 100).fit(X_train_tfidf, Y_train)
```

Figure 6: LogisticRegression: best parameters

	precision	recall	f1-score	support
0	0.39	0.52	0.44	1744
1	0.41	0.32	0.36	1744
2	0.42	0.38	0.40	1744
accuracy			0.41	5232
macro avg	0.41	0.41	0.40	5232
weighted avg	0.41	0.41	0.40	5232

Figure 7: Accuracy LogisticRegression: best parameters

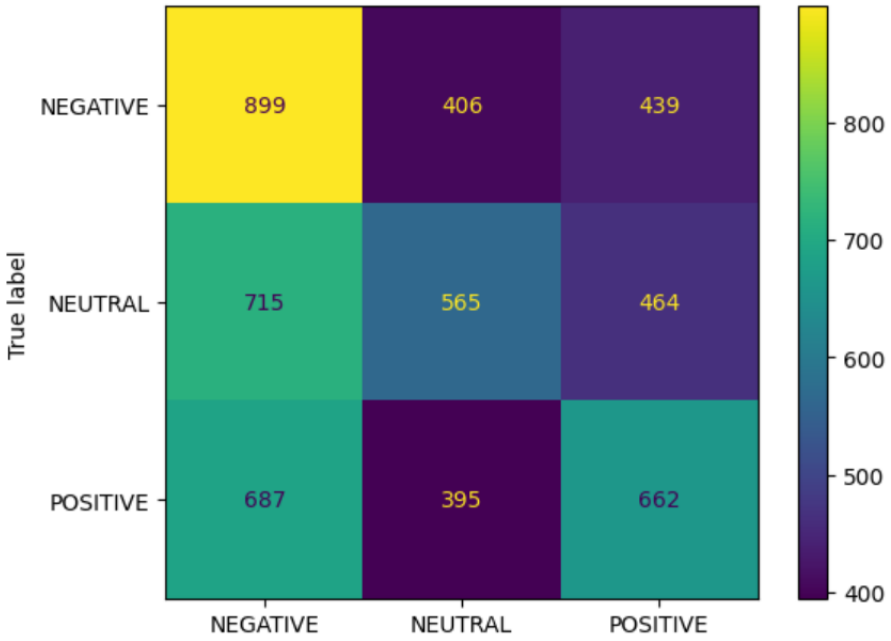


Figure 8: Confusion Matrix LogisticRegression: best parameters

3.4.1.1 Table of trials

Table 1: Trials	
Trial	Score
default	0.39
penalty = 'l2', C = 0.1, solver = 'liblinear', multi_class = 'ovr', tol = 0.001, n_jobs = -1, max_iter = 100	0.41

Table 1: Trials

3.4.2. Experiments - Feedforward Neural Network. After experimental tests of hyper-parameters, batch_size, learning_rate, and optimizer, we conclude that the best snapshot of our neural network with an accuracy of 40.1 was generated in the 29th epoch, with batch_size = 64, learning_rate = 0.1, and optimizer: Adagrad.

```
Accuracy: 40.1%, Avg loss: 0.016994

              precision    recall  f1-score   support

           0       0.39      0.41      0.40      1744
           1       0.39      0.42      0.40      1744
           2       0.42      0.38      0.40      1744

    accuracy                           0.40      5232
   macro avg       0.40      0.40      0.40      5232
weighted avg       0.40      0.40      0.40      5232

Done!
```

Figure 9: Accuracy Feedforward Neural Network: best parameters

From the Confusion Matrix, we extract the following:

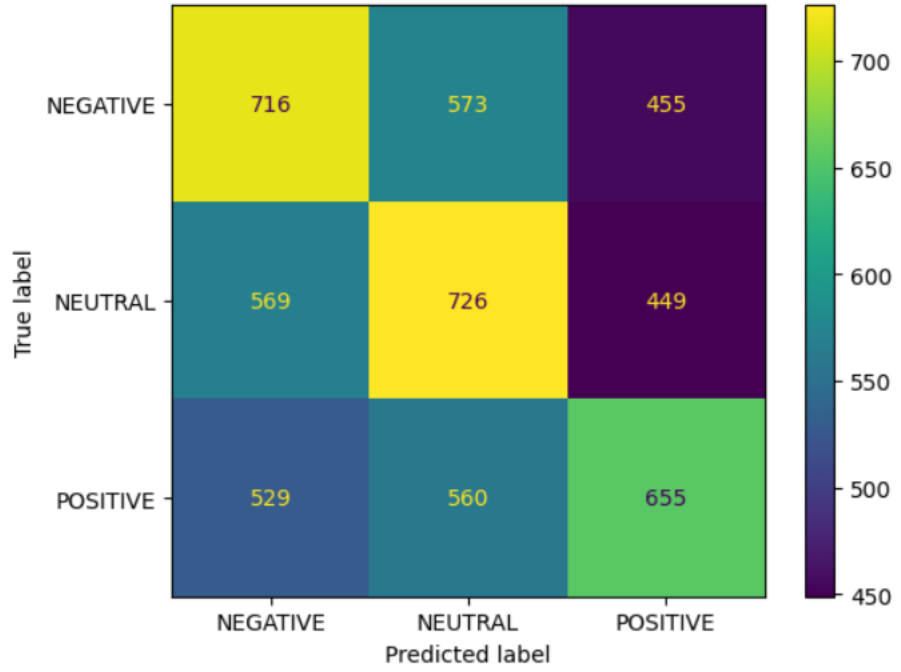


Figure 10: Confusion Matrix Feedforward Neural Network: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 716

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 529

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 455

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 655

We want TN and TP to be greater than FN and FP, which is the case.

3.4.3. Experiments - Recurrent Neural Networks. After the creation of the model, various types of tests are conducted to evaluate its performance. Specifically, experimental tests are conducted in the following areas:

- Model Architecture
  - The performance of two different RNN architectures, **LSTM** and **GRU**, is evaluated.

- Experimentation with the number of layers **num_layers** is performed, where different values ranging from 2 to 8 are tested. Additionally, the number of hidden states **hidden_size** is experimented with, using values from the range *(16 to 128) multiplied by num_heads*. The multiplication is done as a precaution since I have implemented the attention mechanism to avoid the error "embed_dim must be divisible by num_heads" that is generated by the multAttention method.

- The choice of the appropriate number of num_heads affects the model's ability to extract and represent the complex features of the input data. A very small num_heads can decrease the model's ability to extract significant characteristics, while a very large num_heads can result in excessive complexity and overfitting.Typically, choosing a moderate number of num_heads, e.g., between 1 and 5, is usually suitable for the majority of problems. However, the optimal value for this hyperparameter depends heavily on the nature of the problem and the size of the training data. Here, the range of 1 to 5 is used.

- Education and Hyperparameter Tuning

  - Using the Adam optimization algorithm, we perform experimental tests on it by varying the **learning_ rate**, with a range of examination from 0.001 to 0.1.

  - The **dropout** rejection rate is examined with values ranging from 0.1 to 0.9.

  - The **clip** for gradient clipping is examined with cuts ranging from 1 to 5.

- Vanishing Gradient

  - We use skip connections to address the problem of the vanishing scale. We apply different RNN architectures that support these connections, such as LSTM and GRU, and evaluate their performance.

After the above experimental tests that we analyzed, the model we obtained we conclude that the best snapshot of our neural network with an accuracy of 38.9 was generated in the 47th epoch, with:

- *batch_size* = 64
- learning_rate = 0.0045000000000000005
- cell_type = 'LSTM'
- num_layers = 3

- hidden_size = 42
- dropout = 0.65
- num_heads = 1
- skipConnections= True
- clip = 5

```
Accuracy: 38.9%, Avg loss: 0.016832

              precision    recall  f1-score   support

           0       0.37      0.66      0.47      1744
           1       0.40      0.33      0.36      1744
           2       0.48      0.20      0.28      1744

    accuracy                           0.39      5232
   macro avg       0.42      0.39      0.37      5232
weighted avg       0.42      0.39      0.37      5232

Done!
```

Figure 11: Accuracy Recurrent Neural Networks: best parameters

From the Confusion Matrix, we extract the following:

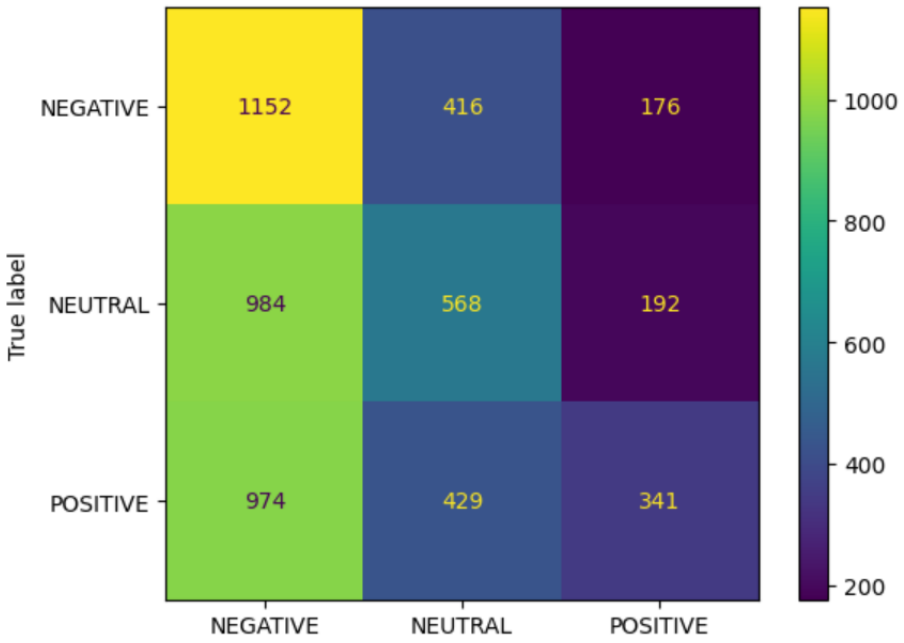


Figure 12: Confusion Matrix Recurrent Neural Networks: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 1152

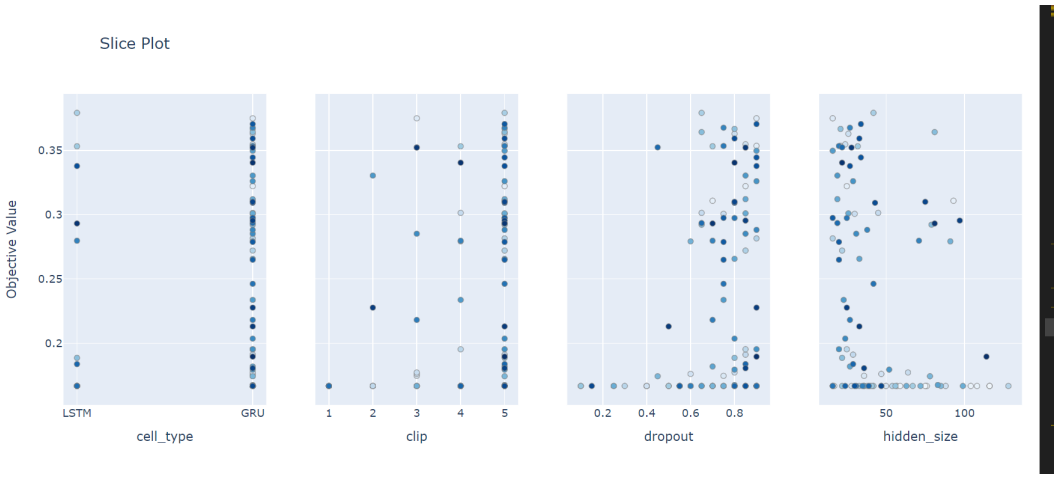
FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 974

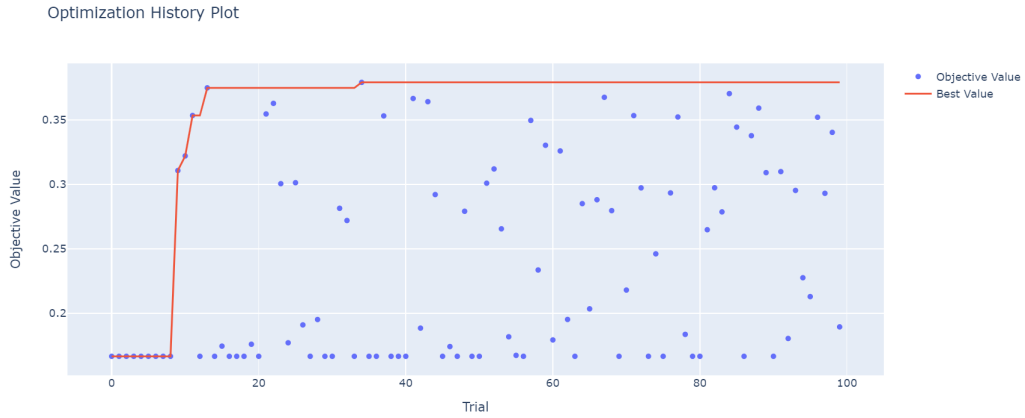
FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 176

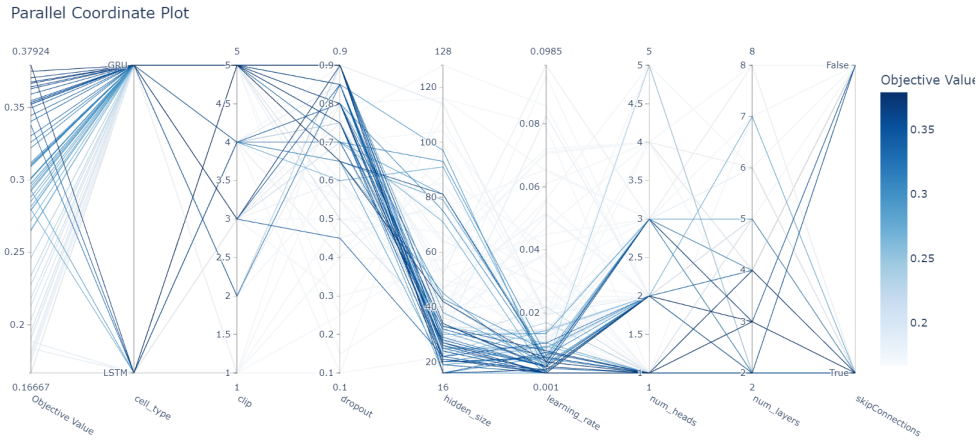
TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 341

We want TN and TP to be greater than FN and FP, which is the case.

PLOTS


Slice Plot
Objective Value
0.35
0.3
0.25
0.2
LSTM
GRU
cell_type
1
2
3
4
5
clip
0.2
0.4
0.6
0.8
dropout
50
100
hidden_size


Optimization History Plot
Objective Value
Best Value
Objective Value
0.35
0.3
0.25
0.2
0
20
40
60
80
100
Trial


Parallel Coordinate Plot
Objective Value
0.37924
0.16667
GRU
LSTM
cell_type
clip
dropout
hidden_size
learning_rate
num_heads
num_layers
False
True
skipConnections

.

3.4.4. Experiments - Bidirectional Encoder Representations from Transformers. After the creation of the model, various types of tests are conducted to evaluate its performance. Specifically, experimental tests are conducted in the following areas:

- Model Architecture
  - The performance of two different BERT architectures, **GreekBERT** and **DistilGREEK-BERT**, is evaluated.

- Education and Hyperparameter Tuning
  - The parameters being experimented with are the value of the learning rate and epsilon for the **AdamW** optimizer.
  - Specifically, for each value of the **learning rate** from a list [1e-5, 1e-4, 1e-3], each value of epsilon from a list [1e-8, 1e-7, 1e-6] is experimented with. Therefore, a total of 9 different experiments are conducted for each training epoch.
  - The experiments include training the model for a number of **epochs** (specified by the variable epochs), as well as evaluating it on a validation set after each epoch. Each experiment records the training loss, validation loss, validation accuracy, as well as the training and validation times.

After completing the experiments, the optimal parameters (learning rate, epsilon) that led to the best results on the validation set are selected.

GreekBERT

After the above experimental tests that we analyzed, the model we obtained we conclude that the best snapshot with an accuracy of 0.359375 was generated in the 1st epoch, with:

- *batch_size* = 32
- learning_rate = 1e-05
- epsilon = 1e-08
- clip = 1

	precision	recall	f1-score	support
0	0.35	0.38	0.36	1744
1	0.35	0.35	0.35	1744
2	0.35	0.31	0.33	1744
accuracy			0.35	5232
macro avg	0.35	0.35	0.35	5232
weighted avg	0.35	0.35	0.35	5232

Figure 13: Accuracy GreekBERT: best parameters

From the Confusion Matrix, we extract the following:

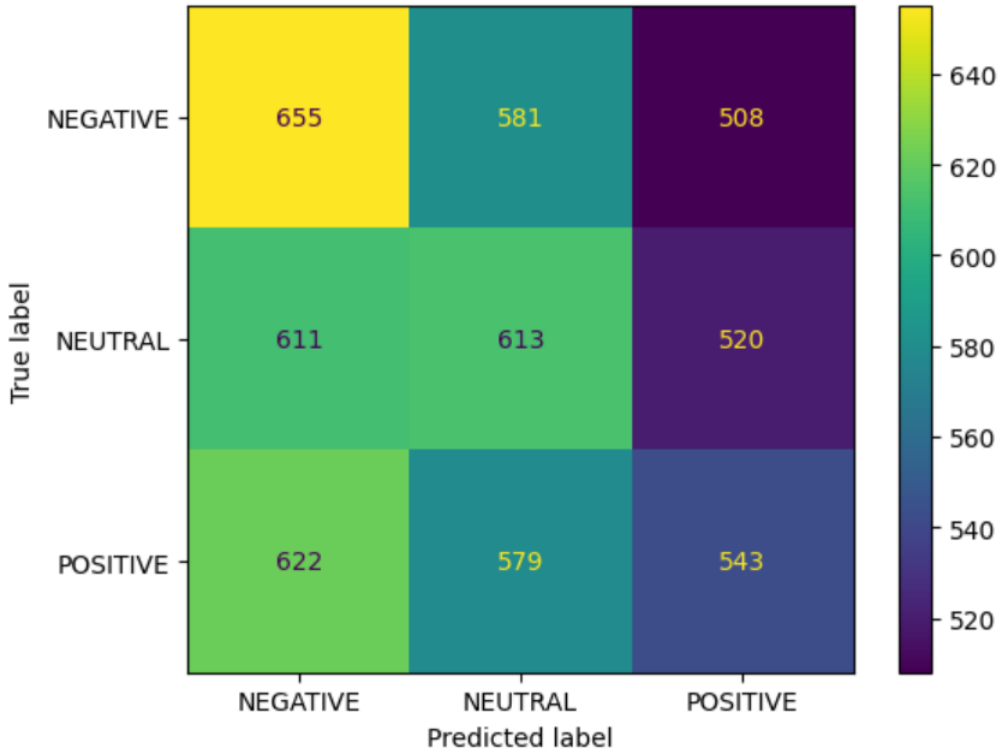


Figure 14: Confusion Matrix GreekBERT: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 655

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 622

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 508

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 543

We want TN and TP to be greater than FN and FP, which is the case.

DistilGREEK-BERT

After the above experimental tests that we analyzed, the model we obtained we conclude that the best snapshot with an accuracy of 0.36 was generated in the 1st epoch, with:

- *batch_size* = 32
- learning_rate = 1e-05
- epsilon = 1e-08
- clip = 1

	precision	recall	f1-score	support
0	0.37	0.33	0.35	1744
1	0.36	0.52	0.42	1744
2	0.33	0.23	0.27	1744
accuracy			0.36	5232
macro avg	0.36	0.36	0.35	5232
weighted avg	0.36	0.36	0.35	5232

Figure 15: Accuracy DistilGREEK-BERT: best parameters

From the Confusion Matrix, we extract the following:

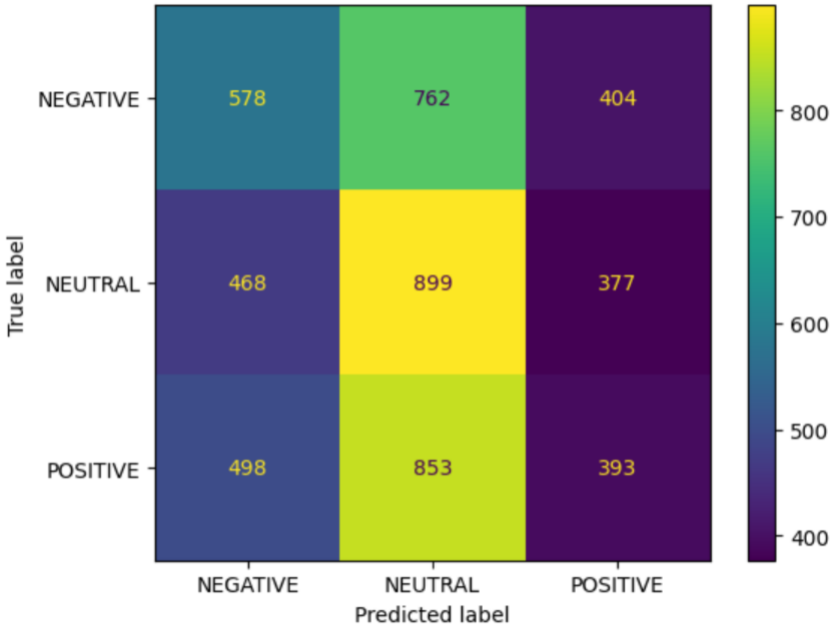


Figure 16: Confusion Matrix GreekBERT: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 578

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 498

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 404

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 393

We want TN and TP to be greater than FN and FP, which is the case.

3.5. Hyper-parameter tuning

3.5.1. Hyper-parameter tuning - Logistic Regression. .

Penalty = l2: l2 was chosen as it protects against underfitting and reduces the risk of overfitting.

C=0.1: We know that C affects both underfitting and overfitting.

Small values of C: try to avoid overfitting by high regularization but can cause underfitting.

Large values of C: can cause overfitting.

However, C = 0.1 seems to have good performance on the validation set, so it strikes a good balance between underfitting and overfitting.

Solver = liblinear does not affect underfitting and overfitting. This definition is the simplest and most effective for economic predictions.

The tolerance affects underfitting for large valu-
es and overfitting for very small values, we observe that tol=0.001 is a good value.

Multi_class = ovr: Almost all optimal results were produced by ovr.

3.5.2. Hyper-parameter tuning - Feedforward Neural Network. .

Number of epochs: The number of epochs has been set to be 100. The optimal snapshot of the neural network was captured at the 29th epoch. This means that the model does not exhibit overfitting issues as the number of epochs is not large, and it also does not suffer from underfitting problems as the number of epochs passed to produce the optimal snapshot is not too small for the model to have learned adequately.

Batch Size: The optimal snapshot of the neural network was captured with a batch_size of 64. The batch size is good as it is not too small to cause problems of overfitting and instability.

Learning Rate: The optimal snapshot of the neural network was captured with a learning_rate = 0.1. The size of the learning _rate is ideal, as it is not too large to lead to overfitting of the data nor too small to prevent the model from converging to a good solution.

3.5.3. Hyper-parameter tuning - Recurrent Neural Networks. .

num_layers = 3: The number of layers can affect overfitting and underfitting depending on the complexity of the model. A small number of layers can lead to underfitting, while a large number of layers can lead to overfitting due to excessive complexity. In the case of the optimal model that was created, a number of 3 layers is good and does not negatively affect the model.

hidden_size = 42: The size of the hidden states affects the complexity of the model. A very large hidden size can lead to overfitting if there is not enough training data to support it, while a very small hidden size can lead to underfitting due to lack of capacity. Here we have hidden_size = 42 which is good for our problem data.

dropout = 0.65: Adding dropout can help reduce overfitting by preventing over-reliance on training data. However, excessive use of dropout can lead to underfitting as it may remove valuable information during training.

learning_rate = 0.0045000000000000005: A high learning rate can lead to overfitting, while a very low learning rate can lead to underfitting due to slow convergence.

The value learning_rate=0.0045000000000000005 is good based on the above.

skipConnections = True: Adding skip connections can help avoid overfitting as it allows for more effective information propagation during training. However, their use can lead to a complex architecture that can hinder learning and lead to underfitting. In our model design, this does not appear to be the case, and as a result, its application helps with the good functioning of the model and avoiding overfitting and underfitting.

clip = 5: Clipping gradients can help avoid overfitting, but excessive clipping can lead to underfitting if training does not progress sufficiently.

num_heads = 1: A high number of num_heads can lead to overfitting. This occurs because the multiple attention heads give the model more flexibility in representing the input data. If the model has excessive flexibility, it may learn to adapt well to the training data but struggle to generalize to new data. On the other hand, a very low number of num_heads can lead to underfitting. This happens when the model lacks enough flexibility to handle the complexity of the input data. In this case, the model may not effectively extract the important features from the training data and may perform poorly on new data. In this specific case, the value of num_heads is 1, which is within the normal range (1 to 5).

3.5.4. Hyper-parameter tuning - Bidirectional Encoder Representations from Transformers. .

3.5.4.1 GreekBERT

.

learning_rate = 1e-05: A high learning rate can lead to overfitting, while a very low learning rate can lead to underfitting due to slow convergence. The value learning_rate=1e-05 is good based on the above.

clip = 1: Clipping gradients can help avoid overfitting, but excessive clipping can lead to underfitting if training does not progress sufficiently.

epsilon = 1e-08: The epsilon in the AdamW optimizer is used to prevent division by zero when updating the model weights. By using very small values for epsilon, we ensure the stability of the training process and prevent overfitting. However, if we use very small values for epsilon in combination with a very small learning rate, we may delay the convergence of the model and cause underfitting, as the model may not adequately learn the data features. The range of values for epsilon in the AdamW optimizer is usually very small and a value very close to zero is typically chosen, such as for example 1e-8 or 1e-7.

3.5.4.2 DistilGREEK-BERT

.

learning_rate = 1e-05

clip = 1

epsilon = 1e-08

3.6. Optimization techniques

To find the optimal values, I conducted manual experimental tests. The experimental test to find the most efficient model of RNNs used optuna.

3.7. Evaluation

3.7.1. ROC Curve. The ROC (Receiver Operating Characteristic) curve is used to evaluate the performance of a classification model. It depicts the trade-off between sensitivity (identifying positive cases) and specificity (avoiding false positive cases) of the model for various decision thresholds. The closer the curve is to the point (0,1), the higher the performance of the model. We observe that the AUC is close to 1, indicating that the model has excellent performance for all models (Logistic Regression & Feed-forward Neural Network & Recurrent Neural Networks).

3.7.1.1 ROC Curve - Logistic Regression

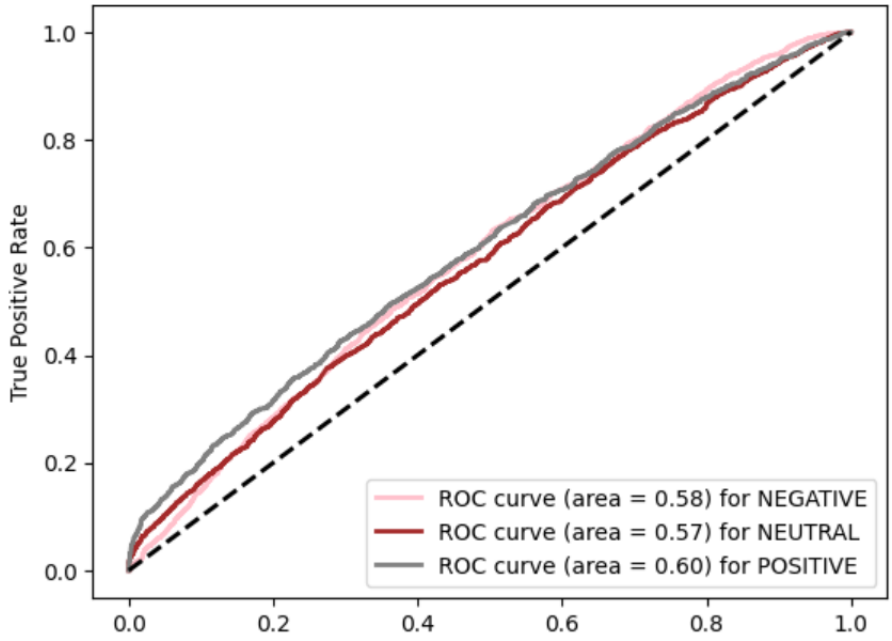


Figure 17: Roc Curve - Logistic Regression

3.7.1.2 ROC Curve - Feedforward Neural Network

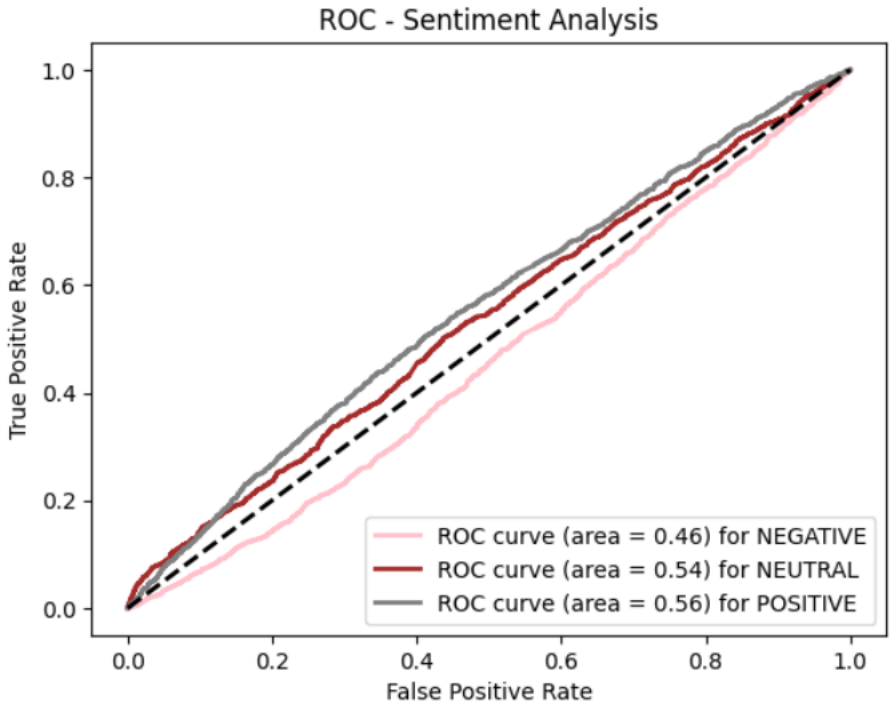


Figure 18: Roc Curve - Feedforward Neural Network

3.7.1.3 ROC Curve - Recurrent Neural Networks

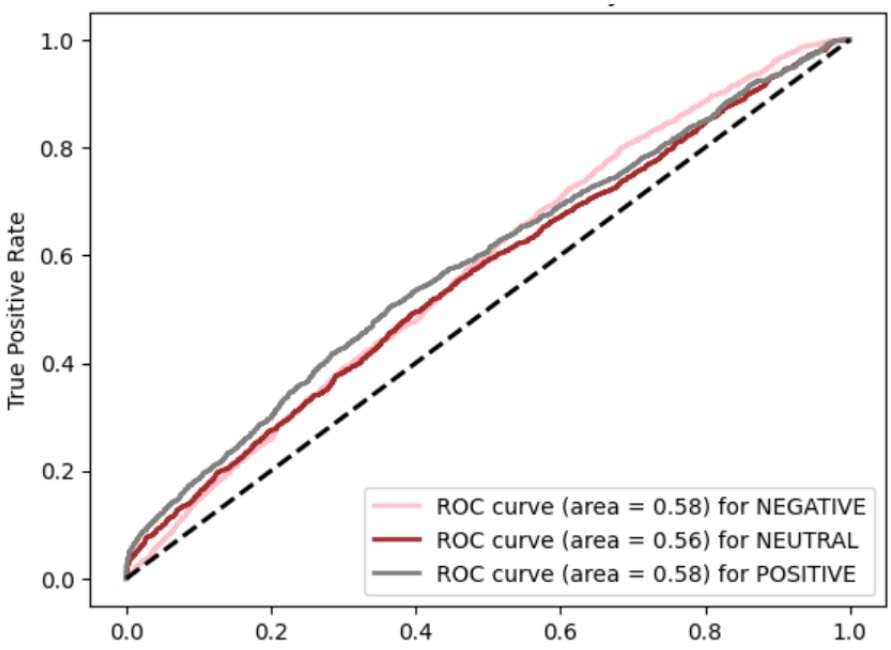


Figure 19: Roc Curve - Recurrent Neural Networks

3.7.1.4 Hyper-parameter tuning - Bidirectional Encoder Representations from Transformers

.

GreekBERT

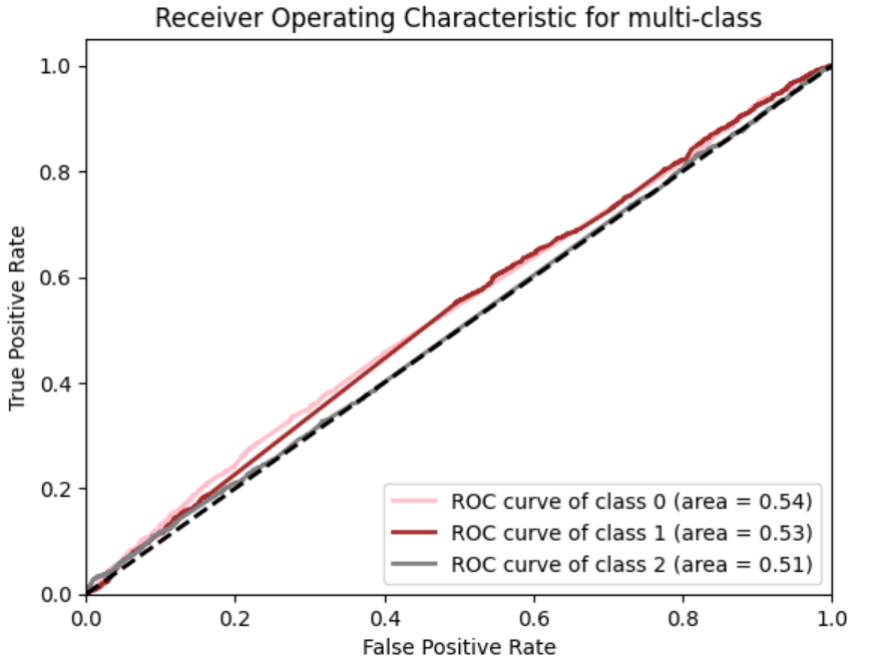


Figure 20: Roc Curve - GreekBERT

DistilGREEK-BERT

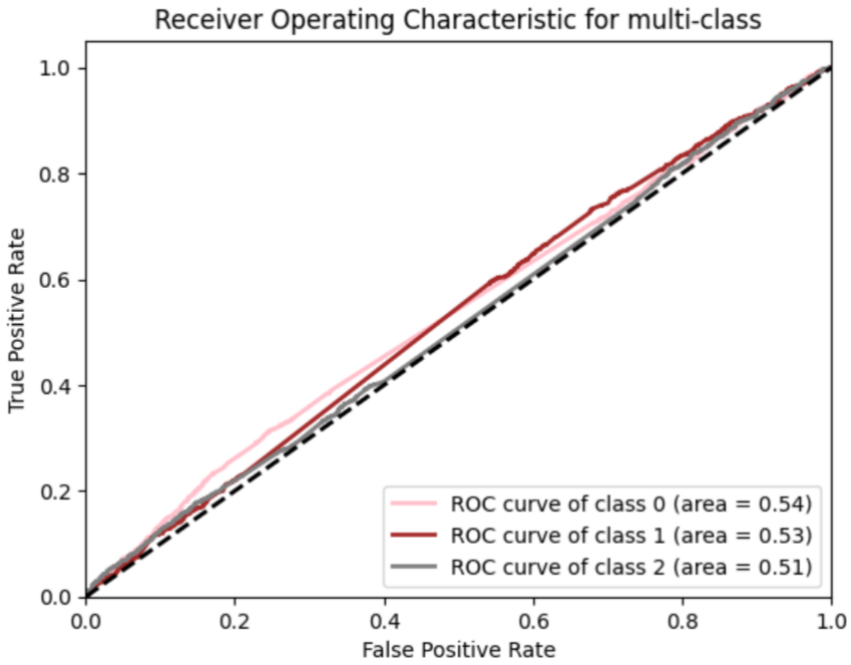


Figure 21: Roc Curve - DistilGREEK-BERT

3.7.2. Learning Curve. The learning curve of Twitter sentiment analysis is the learning curve required to develop an accurate and effective sentiment analysis on Twitter. Sentiment analysis aims to evaluate public opinion and the emotions expressed in tweets posted on the platform.

3.7.2.1 Learning Curve - Logistic Regression

We observe that the learning curve is steep at the beginning, as the system needs to learn to distinguish between positive, negative, and neutral sentiments. As the system gains more experience, its performance improves and the learning curve becomes smoother.

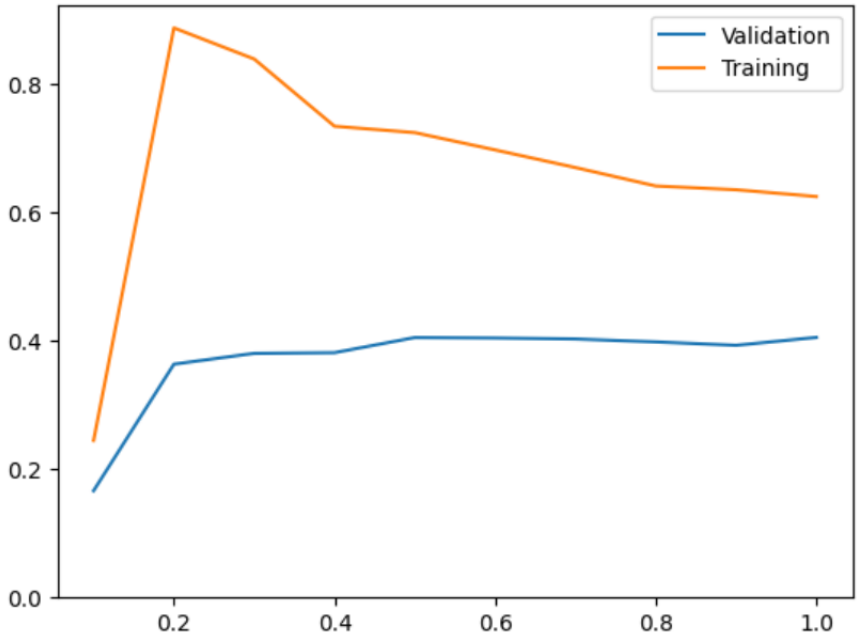


Figure 22: Learning Curve - Logistic Regression

3.7.2.2 Learning Curve - Feedforward Neural Network

We observe that the error for the training and testing data initially increases and then tends to stabilize. Consequently, the model does not present overfitting problems.

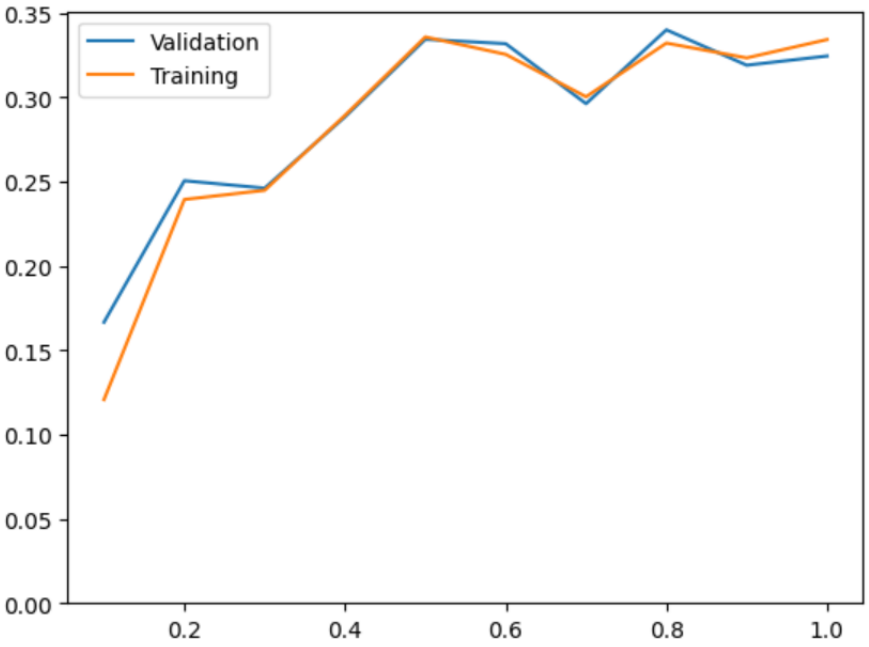


Figure 23: Learning Curve - Feedforward Neural Network

3.7.2.3 Learning Curve - Recurrent Neural Networks

We observe that the error for the training and testing data initially increases and then tends to stabilize. Consequently, the model does not present overfitting problems.

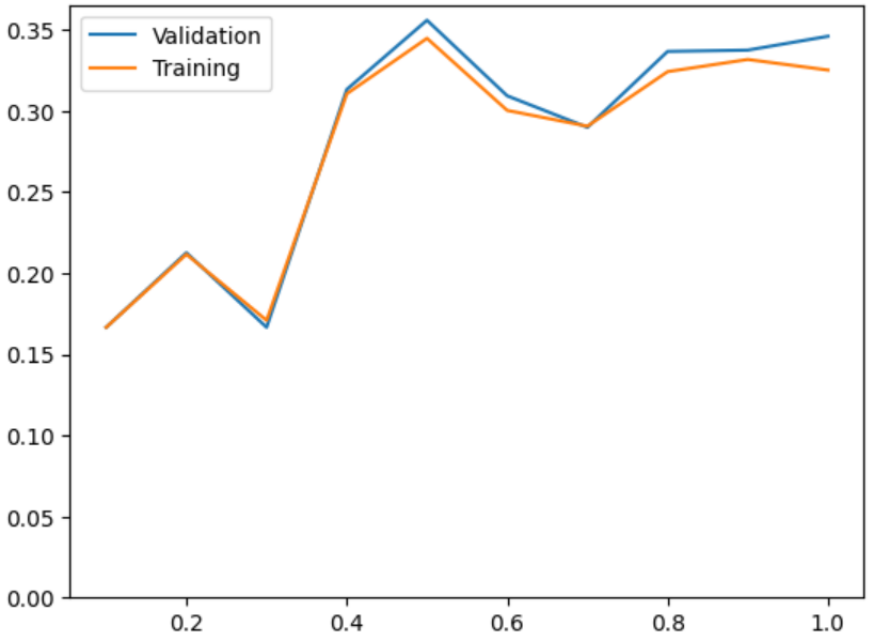


Figure 24: Learning Curve - Recurrent Neural Networks

3.7.2.4 Learning Curve - Bidirectional Encoder Representations from Transformers

.

GreekBERT

In the chart, it is evident that the loss for the training set is very low and stabilizes, while the loss for the validation set has remained almost constant and low as well. This shows that the model has learned the data well and generalizes well to the validation data.

The fact that the validation loss does not increase as the epochs progress is positive, as an increasing validation loss would indicate overfitting. If the validation loss started to increase as the training loss continued to decrease, then this would suggest overfitting.

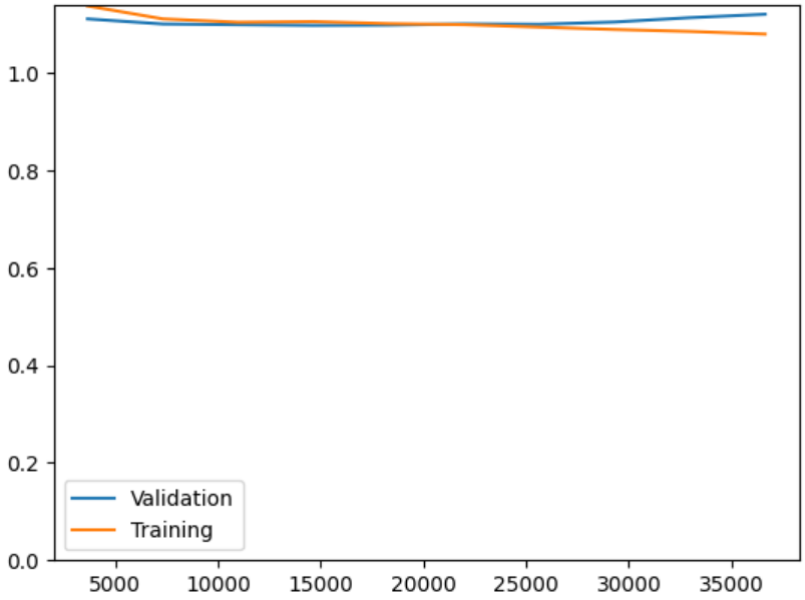


Figure 25: Learning Curve - GreekBERT

DistilGREEK-BERT

Looking at the loss chart, I can see that both the training loss and the validation loss are very low and remain stable throughout the training. This indicates that the model is being trained properly and generalizes well on the validation data, without showing clear signs of overfitting or underfitting.

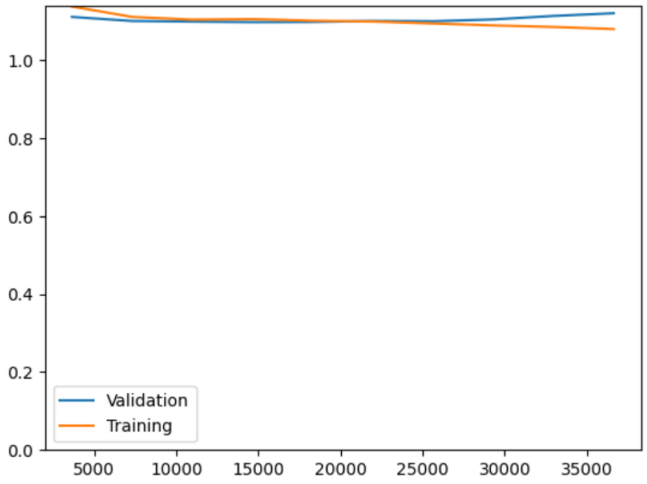


Figure 26: Learning Curve - DistilGREEK-BERT

3.7.3. Confusion matrix.

3.7.3.1 Confusion matrix - Logistic Regression

See the figure 8 Confusion Matrix LogisticRegression: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 899

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 687

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 439

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 662

We want TN and TP to be greater than FN and FP, which is the case.

3.7.3.2 Confusion matrix - Feedforward Neural Network

See the figure 10 Confusion Matrix LogisticRegression: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 716

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 529

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 455

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 655

We want TN and TP to be greater than FN and FP, which is the case.

3.7.3.3 Confusion matrix - Recurrent Neural Networks

See the figure 12 Confusion Matrix Recurrent Neural Networks: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 1152

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 974

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 176

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 341

We want TN and TP to be greater than FN and FP, which is the case.

3.7.3.4 Confusion matrix - Bidirectional Encoder Representations from Transformers

GreekBERT

See the figure 14 Confusion Matrix Recurrent Neural Networks: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 655

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 622

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 508

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 543

We want TN and TP to be greater than FN and FP, which is the case.

DistilGREEK-BERT

See the figure 16 Confusion Matrix Recurrent Neural Networks: best parameters

TN: the number of cases where the model correctly predicted a negative sentiment and the actual sentiment is negative = 578

FP : the number of cases where the model incorrectly predicted a negative sentiment but the actual sentiment is positive = 498

FN: the number of cases where the model incorrectly predicted a positive sentiment but the actual sentiment is negative = 404

TP: the number of cases where the model correctly predicted a positive sentiment and the actual sentiment is positive = 393

We want TN and TP to be greater than FN and FP, which is the case.

4. Results and Overall Analysis

4.1. Results Analysis

We obtained the expected results, comments have been made above.

4.1.1. Best trial.

4.1.1.1 Best trial - Logistic Regression

penalty = l2
C = 0.1
solver = liblinear
multi_class = ovr
tol = 0.001
n_jobs = -1
max_iter = 100

4.1.1.2 Best trial - Feedforward Neural Network

batch_size = 64
learning_rate = 0.1

optimizer = Adagrad

4.1.1.3 Best trial - Recurrent Neural Networks

batch _size= 64

learning_rate = 0.0045000000000000005

cell_type = 'LSTM'

num_layers = 3

hidden_size = 42

dropout = 0.65

num_heads = 1

skipConnections= True

clip = 5

4.1.1.4 Best trial - Bidirectional Encoder Representations from Transformers

GreekBERT

batch _size= 32

learning_rate = 1e-5

epsilon= 1e-8

clip = 1

DistilGREEK-BERT

batch _size= 32

learning_rate = 1e-5

epsilon= 1e-8

clip = 1

4.2. Comparison with the first project

<Use only for projects 2,3,4>

! From the Vectorization section onwards, I have added information about the implementation of the Feedforward Neural Network.

4.2.1. Compare project 1 and 2. .

We observe that the regression technique is satisfactory for the problem, albeit with a slight difference compared to the neural network. Specifically, the performance in logistic regression is 0.41, while the performance of the neural network is 0.40. The two techniques have significant differences between them. Neural networks are more complex and include multi-layered structures and multiple neurons, allowing them to learn more complex data representations. In contrast, logistic regression is a simple linear algorithm where it can only learn linear representations. Finally, a very important difference is that logistic regression is stable in terms of overfitting, while neural networks are more prone to overfitting problems. However, in the current implementation of the project, this has been avoided and overfitting is not observed in the data. More details are described in subchapters 3.2.2, 3.4.1 (& 3.4.1.2) and 3.4.2.2

4.2.2. Compare project 1 and 3. .

As in the comparison between Logistic Regression and Feedforward Neural Network, we observe a difference in the performance of the two models in the comparison between Logistic Regression and Recurrent Neural Networks. Specifically, for Logistic Regression, we have a performance of Logistic Regression 0.41, while for RNNs, we have 0.39. This difference is due to the fact that neural networks are more complex, with multi-layered structures and multiple neurons, allowing them to learn more complex data representations. In contrast, logistic regression is a simple linear algorithm, where it can only learn linear representations.

4.2.3. Compare project 1 and 4. .

BERT is a more complex architecture that usually yields better results in natural language processing problems, while logistic regression is a simple method that can be quickly and easily used in basic classification problems. In some cases, BERT may not be the optimal choice compared to logistic regression. This mainly occurs in small datasets, simple classification problems, and environments with limited computational resources. The choice between BERT and logistic regression depends on many factors and the nature of the problem we are facing.

4.3. Comparison with the second project

<Use only for projects 3,4>

4.3.1. Compare project 2 and 3. .

We observe that there is a small difference between the performances of the two models. Specifically, in the Feedforward Neural Network, we have 0.40, while in the RNNs we have 0.39. Both types of neural networks perform equally well for our problem, the performance is almost the same, and they do not exhibit any overfitting or underfitting issues.

4.3.2. Compare project 2 and 4. .

BERT takes into account the full information of the sentence and the connections between words, while feedforward neural networks examine each word independently. This makes BERT more suitable for multiple language processing applications, while feedforward neural networks are more suitable for simpler prediction tasks. In some cases, BERT may not be the optimal choice, especially in small datasets, simple classification problems, and environments with limited computational resources. The choice between BERT and logistic regression depends on many factors and the nature of the problem we are facing.

4.4. Comparison with the third project

<Use only for project 4>
The architecture of BERT is based on the transformer and allows for bidirectional representation of the text, while RNNs process the input sequence one step at a time. BERT is suitable for problems where a full understanding of the sentence is important, while RNNs are suitable for problems that require the sequence of data. BERT is trained with self-supervision, while RNNs are trained with the backpropagation through time algorithm.In some cases, BERT may not be the optimal choice, especially in small datasets, simple classification problems, and environments with limited computational resources. The choice between BERT and logistic regression depends on many factors and the nature of the problem we are facing. <Comment the results. Why the results are better/worse/the same?>

5. Bibliography

References

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

[2] https://eclass.uoa.gr/modules/document/index.php?course=D424&openDir=/6244007bLDia

[3] https://colab.research.google.com/drive/1VGkXKiGcJ7npt4NKt9K3LleJfNS270Ak?usp=sharing

[4] https://pytorch.org/tutorials/beginner/basics/optimization_tutorial.html

[5] https://keras.io/api/optimizers/

[6]https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/#AUC-ROC_Curve_for_Multi-Class_Classification

[7]https://www.turing.com/kb/mathematical-formulation-of-feed-forward-neural-network

[8]https://discuss.pytorch.org/t/predict-proba/109730

[9]https://stackoverflow.com/questions/71788074/how-to-get-the-roc-curve-of-a-neural-network

<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>