ΟΙΚΟΝΟΜΙΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

ATHENS UNIVERSITY
OF ECONOMICS
AND BUSINESS

M.Sc. in Data Science

Text Analytics

Assignment 5 - Transformers

Grammatikopoulou Maria - f3352310
Phevos A. Margonis - f3352317
Moniaki Melina - f3352321

Instructor: Ion Androutsopoulos
Grader: Foivos Charalampakos

March 13, 2024

Contents

1 Exercise 1 **2**

2 Exercise 2 **4**

1 Exercise 1

1.1 Introduction

Text classification is a fundamental task in natural language processing with applications ranging from sentiment analysis to topic categorization. In this report, we perform a sentiment analysis focusing on the implementation of a Transformer model. We also compare its performance with baselines models.

1.2 Data set

We conducted our analysis on the IMDB Dataset, consisting of 50,000 movie reviews, each annotated with sentiment labels (positive or negative). Our initial exploration revealed a balanced data set, evenly distributed with 25,000 positive and 25,000 negative reviews.

1.3 Data Preprocessing

Initially, we removed HTML tags from our dataset that were in the form of `<br>`. The text was converted to lowercase to establish uniformity and simplify subsequent processing. Following this, we proceeded to remove punctuation marks to enhance text consistency, as well as numerical digits and any single characters that remained after the removal of apostrophes. Multiple spaces and common stop-words were also removed from the dataset to eliminate non-essential words. Moving forward, we transformed sentiment labels into binary values. Positive sentiment was encoded as 1, while negative sentiment was encoded as 0. Then, we split the data set into training (35000 samples), validation (7500 samples) and test set (7500 samples). The Spacy library was used for sentence splitting and tokenization. For each tokenized word we generated TF-IDF features and then we employed SVD for dimensionality reduction, which reduced the feature space from 5000 dimensions to 500 dimensions.

1.4 Baseline Classifiers

Majority Classifier: We implemented a Majority Baseline Classifier using the Dummy Classifier from scikit-learn. This classifier predicts the most frequent class within the training set. The subsequent classification reports offer a comprehensive summary of precision, recall, and F1 score for each class across the training, validation, and test sets correspondingly. Moreover, the PrecisionRecall AUC (PR-AUC) scores provide valuable insights into the classifier's capability to distinguish between classes in terms of precision and recall.

Table 1: Majority Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.50	1.00	0.67	0.75	0.50	1.00	0.66	0.75	0.49	1.00	0.66	0.75
Positive	0.00	0.00	0.00	0.75	0.00	0.00	0.00	0.75	0.00	0.00	0.00	0.75
Macro Avg	0.25	0.50	0.33	0.75	0.25	0.50	0.33	0.75	0.25	0.50	0.33	0.75

Logistic Regression Classifier: The Logistic Regression model serves as an additional baseline for our text classification task. After training the logistic regression model, we observe the following result for the training, validation and test set subsequently.

Table 2: Logistic Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.89	0.87	0.88	0.96	0.89	0.86	0.87	0.95	0.89	0.87	0.88	0.95
Positive	0.87	0.90	0.88	0.95	0.87	0.875	0.88	0.95	0.87	0.89	0.88	0.95
Macro Avg	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95

Multi-Layer Perceptron (MLP) Classifier: An MLP classifier was implemented in Exercise 9 of Part 3 to enhance the performance of the above baseline models. This MLP classifier utilizes a Sequential model with multiple layers. The architecture includes a densely connected layer with 384 units and a Rectified Linear Unit (ReLU) activation. A dropout layer with a rate of 0.2 is incorporated to mitigate overfitting. The final layer consists of a single unit with a sigmoid activation function, suitable for binary classification. The results of this classifier are included in this report to compare the experimental results of an MLP with the results of a RNN model.

Table 3: MLP Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.99	0.99	0.99	1.00	0.89	0.87	0.88	0.95	0.89	0.86	0.88	0.95
Positive	0.99	0.99	0.99	1.00	0.87	0.90	0.88	0.95	0.87	0.90	0.88	0.95
Macro Avg	0.99	0.99	0.99	1.00	0.88	0.88	0.88	0.95	0.88	0.88	0.88	0.95

1.5 Bi-directional stacked RNN with self-attention

A bi-directional stacked Recurrent Neural Network (RNN) with self-attention mechanisms was implemented in Exercise 1 of Part 4 for the task of sentiment analysis. The process begins with text vectorization using a Keras TextVectorization layer, configured to handle a maximum vocabulary size (`MAX_WORDS`) and a maximum sequence length (`MAX_SEQUENCE_LENGTH`). Pre-trained word embeddings, obtained from a FastText model, are utilized to initialize the embedding layer. The RNN architecture consists of two bidirectional GRU layers with recurrent dropout, followed by layer normalization. The outputs of these layers are combined using an element-wise addition, creating a residual connection. A self-attention mechanism, implemented as a Multi-Layer Perceptron (MLP) with dropout, processes the residual states, assigning attention scores to each word in the sequence. The final attention-weighted representation is computed using these scores. The model also includes an MLP layer for further feature extraction, followed by a sigmoid-activated dense layer for binary classification. This MLP architecture closely resembles the MLP classifier architecture implemented in Exercise 9, which was fine-tuned for optimal performance.

During training, the model is compiled with binary cross-entropy loss and Adam optimizer. Checkpoints and early stopping callbacks are employed to monitor and save the model based on validation accuracy. The effectiveness of the model is evaluated on the test set, and training time is reported for reference.

Table 4: RNN Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.95	0.94	0.95	0.99	0.89	0.89	0.89	0.96	0.89	0.89	0.89	0.96
Positive	0.94	0.95	0.95	0.99	0.89	0.89	0.89	0.95	0.89	0.89	0.89	0.96
Macro Avg	0.95	0.95	0.95	0.99	0.89	0.89	0.89	0.96	0.89	0.89	0.89	0.96

1.6 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) was designed for the task of sentiment analysis. The model architecture begins with text vectorization utilizing a Keras TextVectorization layer, configured to handle a maximum vocabulary size (`MAX_WORDS`) and a maximum sequence length (`MAX_SEQUENCE_LENGTH`). To initialize the embedding layer, pre-trained word embeddings from a FastText model were employed. The embedding layer is followed by a dropout layer to introduce regularization and avoid overfitting.

The core of the CNN architecture lies in the multi-filter convolutional layers with residuals and stacking. Different filter sizes (2, 3, and 4) were applied, each undergoing a series of convolutional operations with padding to preserve sequence length. To enhance learning, each convolutional layer incorporates residual connections. The outputs from these layers undergo global max-pooling over time. The resulting representations are then concatenated, and a dropout layer is applied for further regularization.

The final layers of the CNN comprise densely connected networks. Two dense layers with rectified linear unit (ReLU) activation are integrated with dropout layers to capture complex features. The output layer consists of a single neuron with sigmoid activation for binary classification. The model is compiled with binary cross-entropy loss and the Adam optimizer. To monitor and save the model based on validation loss, checkpoints and early stopping callbacks were implemented during training. The effectiveness of the CNN is evaluated on a test set, and the training time is reported for reference.

Table 5: RNN Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.88	0.92	0.90	0.96	0.80	0.84	0.82	0.90	0.79	0.84	0.81	0.90
Positive	0.91	0.87	0.89	0.96	0.84	0.79	0.81	0.90	0.83	0.78	0.81	0.89
Macro Avg	0.90	0.89	0.89	0.96	0.82	0.82	0.82	0.90	0.81	0.81	0.81	0.90

1.7 Transformers

The provided code segment configures and trains a BERT-based classifier for a binary classification task. In the compilation phase, the model is compiled using the Adam optimizer with a specific learning rate of $2e-6$. The loss function utilized here is Binary Crossentropy, which is apt for binary classification problems. Furthermore, the model's performance during training is monitored using Binary Accuracy as the evaluation metric, which assesses the accuracy of the model's predictions against the ground truth labels.

During the training phase, the model is trained for a specified number of epochs (10 in this case) on the training data, which comprises tokenized input sequences and their corresponding attention masks. The training is conducted in batches of size 8. Additionally, to prevent overfitting, early stopping is implemented as a callback mechanism, which halts training if the validation loss does not improve for a specified number of epochs (3 in this case). This strategy helps in restoring the model's best weights based on validation performance. After training, the model is saved as `'checkpoints/distilbert_ex1'`, ensuring that the trained model can be utilized for inference or further analysis without the need to retrain from scratch.

Table 6: Transformer Classifier: Performance Metrics by Class across Data Subsets

Class	Training				Development				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
Negative	0.97	0.96	0.97	0.99	0.92	0.91	0.91	0.97	0.92	0.91	0.91	0.97
Positive	0.96	0.97	0.97	0.99	0.91	0.92	0.92	0.97	0.91	0.92	0.92	0.97
Macro Avg	0.97	0.97	0.97	0.99	0.91	0.91	0.91	0.97	0.91	0.91	0.91	0.97

1.8 Conclusions

As a result of the Bert implementation, we observe that the model achieved validation and test accuracies, reaching 91%. Therefore, the model generalizes well to unseen data. Additionally, the Precision-Recall Area Under the Curve (AUC) approached 97%, indicating a strong ability to balance precision and recall. In a comparative analysis, Bert significantly outperformed all previous classification methods (MLP, CNN, RNN).

2 Exercise 2

2.1 Introduction

This study aims to address the problem of Part-of-Speech tagging using Transformers that are capable of understanding the context of each word. For the purpose of comparison with the simple

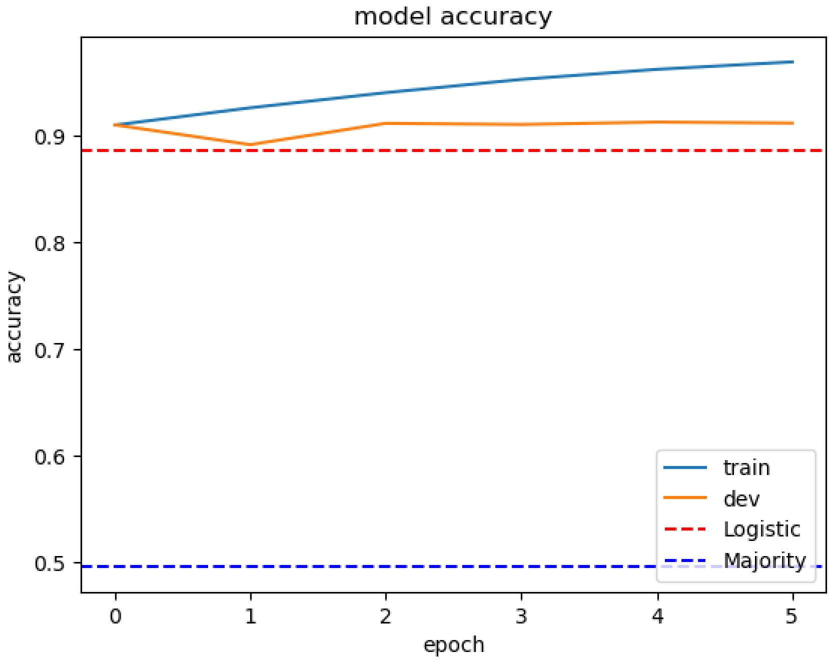

Figure 1: Accuracy Curves for Training and Validation Data as a Function of Training Epoch

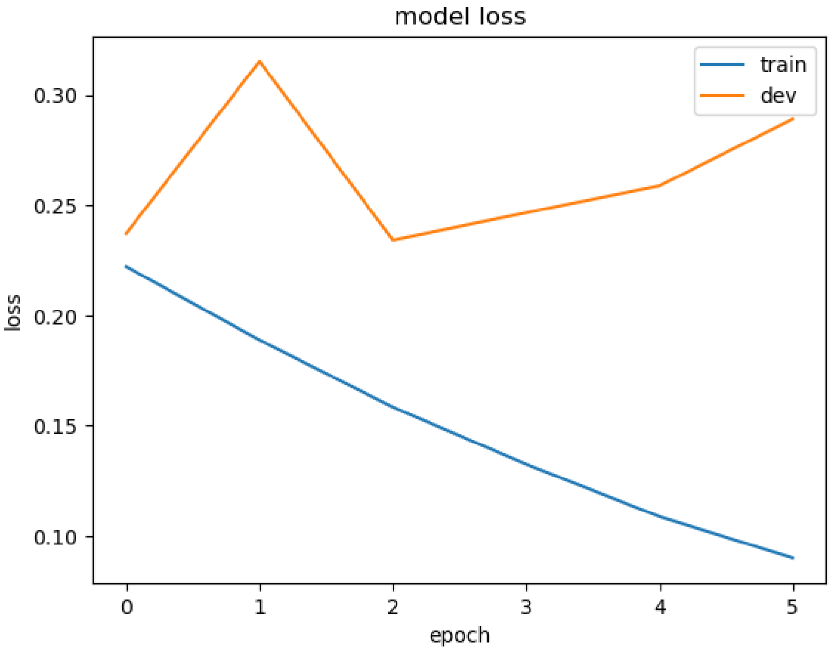

Figure 2: Loss Curves for Training and Validation Data as a Function of Training Epoch

Multilayer Perceptron (MLP) network presented in an earlier study, the final classification layers in the model's architecture have been retained unchanged.

2.2 Dataset

The language of choice for this exercise is English. A collection of annotated text corpora across multiple languages, standardized to represent grammatical information and syntactic relations, is provided by the Universal Dependencies treebanks. From this collection, the English corpus that will serve as the foundation for this study, provided by Georgetown University Multilayer, can be found here. This corpus contains 10,761 sentences, 184,373 tokens[1], and 187,417 syntactic words. More details about the dataset's intricacies can be further explored here.

[1] https://www.datacamp.com/blog/what-is-tokenization

Universal Dependencies have standardized CONLLU as their file format of choice. The main difference from a plain TEXT file is the inclusion of metadata for each token like POS Tags, Features, Relations, etc., and for that reason, the dedicated Python library conllu is required to parse them.

While parsing the data, the features that are extracted are (a) the token to be classified, (b) its preceding and subsequent words, forming a context window of $n = 3$, and (c) the token's UPOS label. The UPOS labels denote 17 distinct classes (i.e., ADJ, ADP, ADV, AUX). To account for the first and last word in a sentence, and by extension their context window, a pseudo-token *PAD* is prepended and appended accordingly. Hence, a DataFrame is created where each row holds the above information for each available token. Lastly, the DataFrame is stored as CSV to streamline the following steps. This process is repeated for all three available Train-Validation-Test datasets.

2.3 Pre-processing

Initially, the data are parsed from the CONLLU files and stored in a DataFrame. This frame comprises two columns. The first column, *text*, contains one sentence per row in the form of a string. Similarly, the second column, *label*, contains the respective UPOS labels as a string. This process is repeated for all Train-Validation-Test data, resulting in approximately 8000, 1000, and 1000 sentences, respectively. For speed and efficiency, the DataFrames are stored as CSV files, which will be used as the primary data source.

Upon loading the CSV files, several exploratory queries are performed. Firstly, in the training set, duplicate entries are found, usually corresponding to the valediction part of a document. Since they offer no extra information, they are dropped. Secondly, statistics regarding the length of each sentence are calculated, revealing that 90% of them comprise approximately 35 words. This information will later influence the chosen value for the maximum context window that the RNNs can handle.

For faster training and inference times, a context window of 32 words (max sequence length) is chosen. This means that if a sentence exceeds 32 words, only the first 32 will be considered. Conversely, sentences with fewer words will be padded to meet the max sequence length requirement.

Following this, both texts and labels must be converted to a format that the model can handle. For speed and simplicity, the TextVectorization layer from Keras is used to tokenize the labels into sequences of integers. An important detail is that this layer reserves the first two positions for UNKNOWN and PAD tokens, meaning the target labels will take values in the range of 2-19 instead of 0-17.

Having vectorized the label values, we proceed with the texts. Here, a TextVectorization layer is initialized to accept sentences, tokenize them, convert the tokens to lowercase, and match them to their integer values found in the vocabulary adapted from the training set. This preconfigured layer has been stored as an object that will later be included in the model as an on-the-fly Text-to-Vector converter.

Finally, utilizing the Vocabulary created in the last step, along with the *Fasttext* model, every word in the Vocabulary is converted to its respective embedding, thus creating the embedding matrix. This matrix reserves the first two positions for the PAD and UNK tokens, which take a zero-vector embedding.

2.4 Baselines

Before proceeding with the creation of the CNN, a Majority classifier is built to act as a baseline for comparison. This classifier tags each word with the most frequent tag it encountered in the training data. For words not encountered in the training data, the baseline returns the most frequent tag (over all words) from the training data. This simplistic approach achieves a validation score of about 88%. The results can be found in Table 7.

Furthermore, the MLP and RNN classifiers, presented in the previous studies as robust baselines, are included for comparison. Their classification abilities are detailed in Table 8 and Table 9.

2.5 Convolutional Neural Networks (CNNs)

Initially, the model begins with an input layer of size one and type string. This layer is responsible for receiving batches of raw strings and then forwarding them to the Vectorization layer. The Vectorization layer tokenizes, lowercases, and matches each word to its respective index in the Vocabulary. Subsequently, the now-serialized text sequences pass through the Embedding layer,

Table 7: Majority Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.91	0.93	0.92	-	0.90	0.81	0.85	-	0.88	0.80	0.84	-
ADP	0.90	0.89	0.90	-	0.91	0.91	0.91	-	0.91	0.91	0.91	-
ADV	0.89	0.85	0.87	-	0.87	0.80	0.84	-	0.89	0.77	0.82	-
AUX	0.87	0.95	0.91	-	0.88	0.96	0.92	-	0.87	0.94	0.91	-
CCONJ	0.99	1.00	0.99	-	0.98	1.00	0.99	-	0.98	1.00	0.99	-
DET	0.95	0.98	0.96	-	0.95	0.98	0.97	-	0.96	0.98	0.97	-
INTJ	0.73	0.76	0.74	-	0.75	0.74	0.74	-	0.78	0.68	0.72	-
NOUN	0.92	0.95	0.93	-	0.74	0.94	0.83	-	0.68	0.93	0.78	-
NUM	0.95	1.00	0.98	-	0.95	0.86	0.91	-	0.97	0.81	0.89	-
PART	0.69	0.90	0.78	-	0.75	0.92	0.83	-	0.69	0.90	0.78	-
PRON	0.92	0.96	0.94	-	0.93	0.96	0.94	-	0.92	0.97	0.94	-
PROPN	0.94	0.82	0.88	-	0.78	0.40	0.53	-	0.79	0.35	0.49	-
PUNCT	0.99	1.00	1.00	-	1.00	1.00	1.00	-	0.99	1.00	1.00	-
SCONJ	0.85	0.33	0.48	-	0.87	0.36	0.51	-	0.88	0.38	0.54	-
SYM	0.95	0.68	0.79	-	1.00	0.36	0.53	-	0.82	0.53	0.64	-
VERB	0.93	0.89	0.91	-	0.92	0.77	0.84	-	0.90	0.76	0.83	-
X	0.97	0.73	0.84	-	1.00	0.50	0.67	-	0.88	0.50	0.64	-
Macro Avg	0.90	0.86	0.87	-	0.89	0.78	0.81	-	0.87	0.78	0.81	-

Table 8: MLP Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.96	0.98	0.97	0.99	0.91	0.93	0.92	0.97	0.88	0.92	0.90	0.96
ADP	0.98	0.99	0.99	0.99	0.95	0.97	0.96	0.99	0.95	0.98	0.97	0.99
ADV	0.98	0.94	0.96	0.99	0.94	0.87	0.91	0.96	0.96	0.87	0.91	0.97
AUX	0.99	0.99	0.99	0.99	0.98	0.97	0.98	0.99	0.98	0.97	0.98	0.99
CCONJ	0.93	0.94	0.93	0.98	0.83	0.87	0.85	0.91	0.79	0.84	0.81	0.89
DET	0.99	0.99	0.99	0.99	0.95	0.96	0.96	0.99	0.95	0.96	0.96	0.99
INTJ	0.89	0.87	0.88	0.93	0.82	0.76	0.79	0.89	0.88	0.81	0.85	0.89
NOUN	0.96	0.98	0.97	0.99	0.93	0.95	0.94	0.97	0.89	0.94	0.92	0.96
NUM	0.99	0.97	0.98	0.99	0.99	0.96	0.97	0.98	0.98	0.96	0.97	0.99
PART	0.99	0.97	0.98	0.99	0.93	0.90	0.92	0.96	0.92	0.83	0.87	0.94
PRON	0.99	1.00	0.99	1.00	0.99	0.99	0.99	0.99	0.98	0.99	0.98	0.99
PROPN	0.89	0.88	0.88	0.95	0.79	0.73	0.76	0.83	0.80	0.70	0.75	0.83
PUNCT	0.99	0.99	0.99	0.99	0.98	0.98	0.98	0.99	0.98	0.98	0.98	0.99
SCONJ	0.96	0.91	0.94	0.98	0.86	0.77	0.81	0.89	0.84	0.81	0.83	0.90
SYM	0.96	0.83	0.89	0.92	1.00	0.64	0.78	0.97	0.96	0.68	0.79	0.79
VERB	0.98	0.98	0.98	0.99	0.95	0.95	0.95	0.98	0.95	0.94	0.94	0.98
X	0.97	0.34	0.50	0.67	0.80	0.33	0.47	0.44	1.00	0.39	0.56	0.52
Macro Avg	0.96	0.91	0.93	0.96	0.92	0.86	0.88	0.92	0.92	0.86	0.88	0.92

Table 9: RNN Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	1.00	1.00	1.00	1.00	0.88	0.88	0.88	0.95	0.86	0.87	0.86	0.94
ADP	1.00	1.00	1.00	1.00	0.96	0.98	0.97	0.99	0.97	0.98	0.97	1.00
ADV	1.00	0.98	0.99	1.00	0.88	0.87	0.88	0.95	0.89	0.89	0.89	0.95
AUX	1.00	1.00	1.00	1.00	0.96	0.99	0.98	1.00	0.94	0.98	0.96	1.00
CCONJ	1.00	1.00	1.00	1.00	0.99	1.00	0.99	1.00	0.99	0.99	0.99	1.00
DET	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00	0.99	0.99	0.99	1.00
INTJ	0.92	1.00	0.96	1.00	0.82	0.94	0.88	0.95	0.81	0.94	0.87	0.96
NOUN	1.00	1.00	1.00	1.00	0.89	0.93	0.91	0.97	0.85	0.93	0.89	0.96
NUM	1.00	1.00	1.00	1.00	0.98	0.85	0.91	0.93	0.97	0.83	0.90	0.91
PART	1.00	1.00	1.00	1.00	0.87	0.98	0.92	0.99	0.73	0.96	0.83	0.96
PRON	1.00	1.00	1.00	1.00	0.99	0.99	0.99	1.00	0.99	0.99	0.99	1.00
PROPN	1.00	0.99	0.99	1.00	0.77	0.63	0.70	0.79	0.82	0.59	0.68	0.80
PUNCT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SCONJ	0.99	1.00	0.99	1.00	0.87	0.81	0.84	0.90	0.85	0.87	0.86	0.91
SYM	1.00	1.00	1.00	1.00	1.00	0.71	0.83	1.00	0.87	0.81	0.84	0.82
VERB	1.00	1.00	1.00	1.00	0.94	0.91	0.93	0.98	0.93	0.90	0.92	0.97
X	0.99	0.92	0.95	0.99	0.80	0.36	0.50	0.41	0.67	0.25	0.36	0.38
Macro Avg	0.99	0.99	0.99	1.00	0.92	0.87	0.89	0.93	0.89	0.87	0.87	0.92

resulting in a 3D tensor of shape (batch, max-sequence-length, embedding-dimension). Finally, a Dropout layer with a probability of 0.33 is applied to the embeddings for regularization.

Following this, the pre-trained FastText embeddings are processed through several convolutional layers. Each layer increases the receptive field of each word, resulting in embeddings that are more *context-aware*. Furthermore, the receptive field is also influenced by the kernel size applied to the filters. In the context of NLP, the kernel sizes can be interpreted as the sizes of *n-grams*. Specifically, this architecture employs Bi-gram, Tri-gram, and Quad-gram kernels. For each n-gram, a distinct set of convolutional blocks is strung together to produce n-gram-specific embeddings. Consequently, the embeddings produced by the three kernels are concatenated, forming a larger embedding for each word.

Each kernel group begins with a 1D Convolutional layer that excludes both a bias term and an activation function. This design choice is twofold. Firstly, the bias term, a learned variable that shifts the layer from purely linear to affine, becomes redundant as the output is centered on zero through BatchNormalization, thus negating the need for a bias term. Secondly, applying the activation function after normalization leverages the non-linearity introduced by activation functions around zero.

This raw output enters a Residual Block, which can be repeated as needed. In the first step of each block, the output from the previous layer is stored as a residual. Then, the BatchNormalization layer and the ReLU Activation layer are applied. This pattern is repeated with a 1D Convolutional layer (excluding bias and activation), followed by BatchNormalization and then Activation. A second 1D Convolutional layer is included within that block. To complete the residual connection, the output of the last layer is added to the residual stored at the beginning of the block. The final block in the architecture is not activated because empirical results have demonstrated that allowing gradients to flow more freely can result in an average increase in accuracy of 0.2%.

Additionally, the same MLP is added for comparative reasons and consists of two Dense layers interspersed with Dropout layers with probabilities of 0.4 and 0.2. The activation functions selected are ReLU and Tanh, respectively. The output layer comprises 19 neurons, designed to classify the 17 classes, along with the UNKNOWN and PAD tokens, using the SoftMax activation function. The final architecture can be seen in **??**.

Sparse categorical crossentropy is chosen as the loss function because the labels are presented in serialized format. The learning rate is empirically set to 0.001, and accuracy is the chosen metric. Moreover, an EarlyStopping callback is set up with a patience of 10 to monitor validation loss and restore the best weights. Subsequently, the model is trained on the training data using batch sizes of 256 for a maximum of 150 epochs.

Consequently, the CNN model leverages its ability to parallelize the computation of each embedding and converges by the 28th epoch in a record time of two and a half minutes. It is worth mentioning a slight discrepancy introduced by using the option `mask_zero=True` in the embedding layer. Previous studies on RNNs have established that Keras can correctly utilize the masking mechanism in the `.fit` and `.evaluate` functions, but it cannot properly apply it in `.predict`, leading to unsuccessful attempts to classify padding tokens. Here, the problem is reversed, arising in the computation of accuracy both in `.fit` and `.evaluate`, as it includes the support of class 0 (pad), thus falsely inflating the metric. By accounting for this discrepancy with manual masking and calculations, the validation accuracy reaches 93% and the f1 macro-average is 86%. Ultimately, the results can be scrutinized in Table 10.

Table 10: CNN Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.94	0.96	0.95	0.99	0.84	0.87	0.86	0.93	0.78	0.89	0.83	0.92
ADP	0.98	0.98	0.98	1.00	0.96	0.97	0.96	0.99	0.97	0.96	0.96	1.00
ADV	0.94	0.96	0.95	0.99	0.80	0.87	0.83	0.94	0.82	0.87	0.84	0.94
AUX	0.99	0.99	0.99	1.00	0.97	0.98	0.97	1.00	0.96	0.97	0.97	0.99
CCONJ	1.00	1.00	1.00	1.00	0.99	0.99	0.99	1.00	0.99	0.98	0.99	1.00
DET	0.99	1.00	1.00	1.00	0.98	0.99	0.99	1.00	0.99	0.99	0.99	1.00
INTJ	0.98	0.91	0.94	0.99	0.91	0.78	0.84	0.94	0.94	0.78	0.85	0.94
NOUN	0.97	0.97	0.97	0.99	0.89	0.91	0.90	0.96	0.85	0.90	0.87	0.94
NUM	0.99	0.99	0.99	1.00	0.98	0.85	0.91	0.92	0.99	0.82	0.89	0.90
PART	0.99	0.99	0.99	1.00	0.94	0.97	0.95	0.99	0.88	0.94	0.91	0.97
PRON	1.00	0.99	0.99	1.00	0.99	0.97	0.98	1.00	0.98	0.98	0.98	1.00
PROPN	0.92	0.93	0.92	0.98	0.80	0.59	0.68	0.76	0.80	0.55	0.65	0.77
PUNCT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SCONJ	0.90	0.92	0.91	0.97	0.79	0.80	0.80	0.87	0.75	0.84	0.79	0.87
SYM	0.96	0.96	0.96	0.99	1.00	0.57	0.73	0.75	0.86	0.75	0.80	0.74
VERB	0.98	0.97	0.98	1.00	0.91	0.91	0.91	0.97	0.92	0.90	0.91	0.97
X	1.00	0.43	0.61	0.81	1.00	0.18	0.31	0.46	0.88	0.29	0.44	0.57
Macro Avg	0.97	0.94	0.95	0.98	0.93	0.84	0.86	0.91	0.90	0.85	0.86	0.91

2.6 Transformers

The process commences with data preparation, where training, validation, and test subsets are encapsulated within a DatasetDict structure. Following data preparation, the workflow transitions to model and tokenizer setup. A pre-trained tokenizer, specifically the AutoTokenizer for the distilbert-base-uncased model, is employed to convert text into a format amenable to the model's processing capabilities. This step is pivotal, as it addresses the challenge of transforming raw text into a structured form that the model can interpret. Subsequent tokenization examples are provided to exemplify the process, highlighting the tokenizer's capability to handle inputs that are pre-split into words. The method `tokenize_and_align_labels` is introduced to reconcile the misalignment between tokens and labels, a common issue arising from tokenization, especially when words are fragmented into subwords. This alignment ensures that each token is appropriately labeled, facilitating the model's learning process. The comprehensive workflow, encompassing data preparation, tokenization, model preparation, and training.

Contributions

In this assignment, the work was divided between the members of our team. The first part of exercise 2, up to and including the baseline classifiers, was implemented by *Maria Grammatikopoulou*, while the rest was designed by *Melina Moniaki*. Lastly, exercise 3 was implemented by *Phevos A. Margonis*.

Table 11: Transformer Classifier: Performance Metrics by Class across Data Subsets

Class	Train				Validation				Test			
	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC	Pre	Rec	F1	AUC
ADJ	0.97	0.91	0.94	0	0.92	0.86	0.89	0	0.90	0.83	0.87	0
ADP	0.95	0.93	0.94	0	0.92	0.90	0.91	0	0.90	0.88	0.89	0
ADV	0.96	0.92	0.94	0.98	0.93	0.86	0.90	0.94	0.89	0.85	0.87	0.92
AUX	0.98	0.96	0.97	0.99	0.96	0.94	0.95	0.97	0.95	0.95	0.95	0.96
CCONJ	0.97	0.92	0.94	0.98	0.95	0.89	0.92	0.94	0.92	0.87	0.90	0.93
DET	0.98	0.96	0.97	0.99	0.95	0.93	0.94	0.98	0.94	0.93	0.93	0.98
INTJ	0.96	0.88	0.92	0.98	0.90	0.81	0.85	0.96	0.90	0.79	0.84	0.95
NOUN	0.90	0.97	0.93	0.99	0.86	0.93	0.89	0.98	0.84	0.90	0.87	0.97
NUM	0.90	0.88	0.89	0.97	0.85	0.80	0.82	0.93	0.83	0.84	0.83	0.93
PART	0.98	0.94	0.96	0.99	0.96	0.92	0.94	0.97	0.93	0.92	0.93	0.95
PRON	0.97	0.96	0.97	0.95	0.95	0.95	0.95	0.91	0.95	0.94	0.94	0.91
PROPN	0.91	0.92	0.91	0.98	0.81	0.78	0.80	0.97	0.79	0.75	0.77	0.96
PUNCT	0.94	0.94	0.94	0.99	0.92	0.92	0.92	0.99	0.89	0.88	0.89	0.98
SCONJ	0.94	0.90	0.92	0.98	0.87	0.81	0.84	0.88	0.86	0.85	0.86	0.85
SYM	0.90	0.38	0.53	0.99	0.00	0.00	0.00	0.98	0.89	0.25	0.39	0.97
VERB	0.95	0.96	0.95	0.96	0.91	0.92	0.91	0.89	0.93	0.90	0.91	0.92
X	0.89	0.51	0.65	0.66	1.00	0.09	0.17	0.03	0.73	0.46	0.56	0.46
Macro Avg	0.94	0.87	0.90	0.96	0.86	0.78	0.80	0.89	0.88	0.81	0.84	0.91