****

**Athens University of Economy and Business**

**School of Information Sciences & Technology: Department of Informatics**

**Master of Science in “Data Science”**

**Course:** Text Analytics

**“ASSIGNMENT 4 – NLP with RNN’s”**

**Students:** 1) Dionysios Voulgarakis (f3352307)

2) Rafail Mpalis (f3352308)

3) Dimitrios Stathopoulos (f3352318)

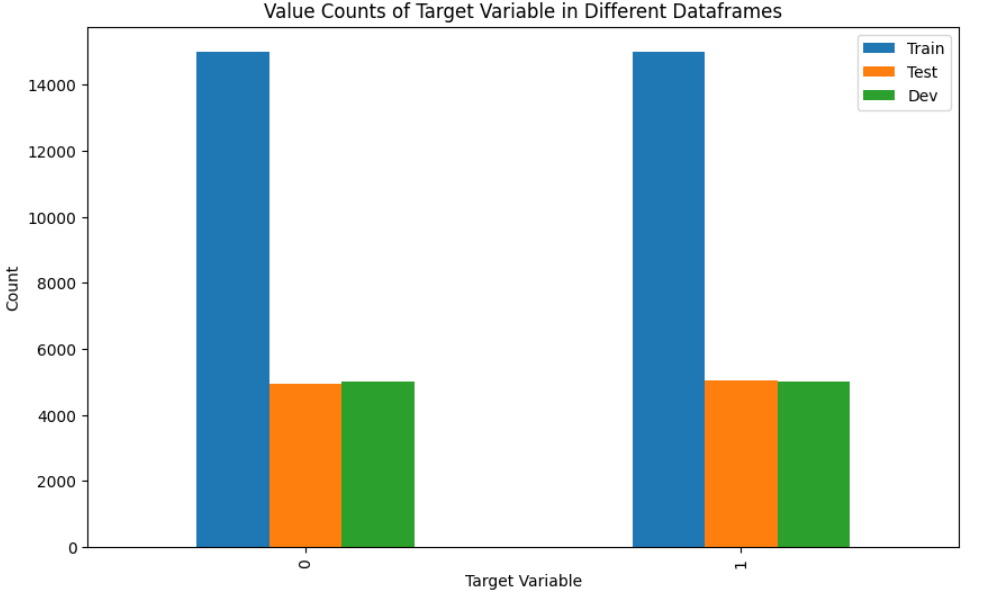
**Instructor:** Prof. Ion Androutsopoulos

**Grader:** Foivos Charalampakos

**EXERCISE 1**

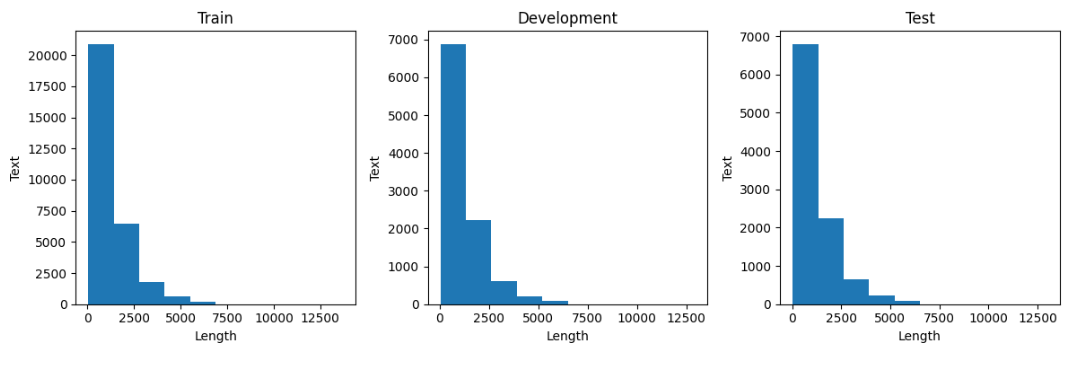
In this project our aim is to build a RNN classifier for sentiment Analysis. In order to train our model we use the [**Large Movie Review Dataset**](https://ai.stanford.edu/~amaas/data/sentiment/)**,** which contains 50,000 binary labeled movie reviews. The classes are two, 0 for negative and 1 for positive sentiment.

We decided to keep 30,000 reviews for training and we splitted the rest 20,000 in half for development and test sets. In each of the sets we kept the balance of positive and negative even.



Picture 1: Counts of reviews for each set

The length of the of documents in train and development set is seen below.



Picture 2: Word Counts of reviews for each set

**Preprocessing**

We used NLTK library to preprocess our data. For each of the above sets we performed:

* Removing of non-word characters,
* Removing of single characters,
* Removing of Numerical values,
* Converting into lowercase, to achieve homogeneity,
* Remove stop words,
* We didn't use Lemmatization, as it didn’t seem to improve our model.

**TF-IDF**

We based our model in tf-idf representation of the reviews. We used:

* Unigrams and bigrams
* max features 5000

**SVD**

In order to reduce the features we used truncated svd. We kept 500 components.

**Baseline Model 1 – Logistic Regression**

We started using Logistic regression as our baseline model, which achieved accuracy of 88%.

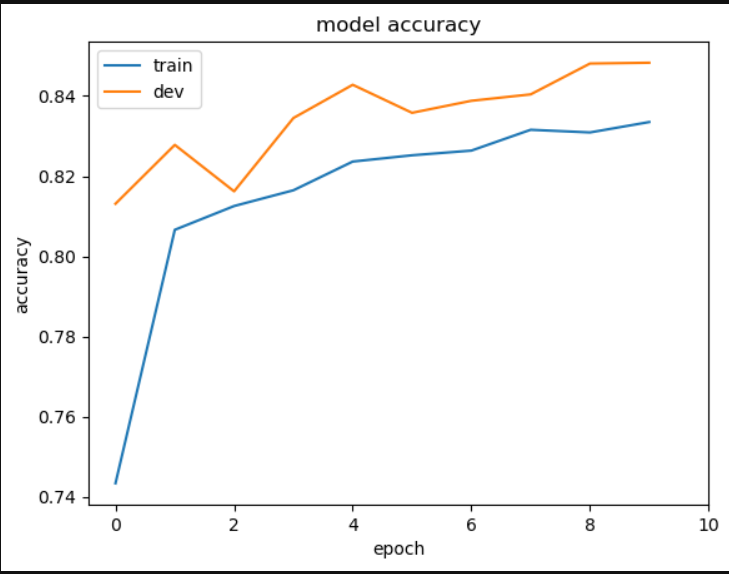
**Baseline Model 2 – Multi-Layer Perceptron**

The MLP is identical to the one we built in Exercice 9 of Part 2 which was shown to achieve an accuracy of 89.260% (our best model so far)

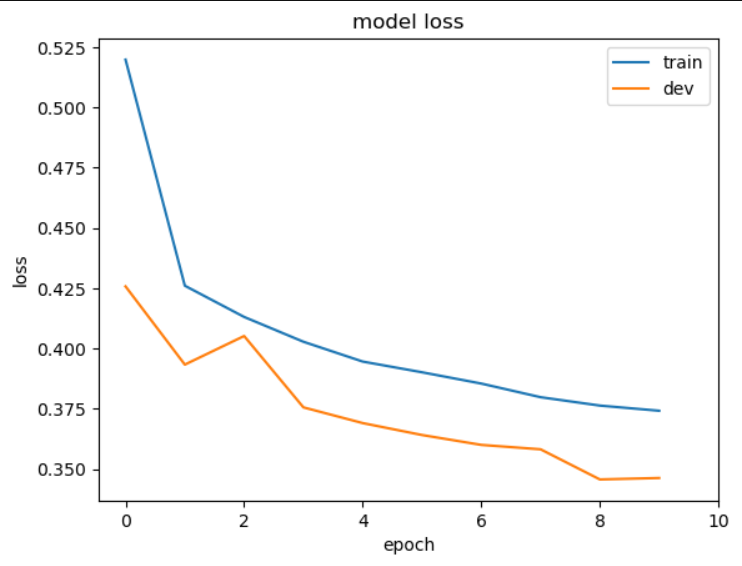
**Model-1 (BiDirectional LSTM+Linear Self Attention Scores)**

We built a BiLSTM as our base model using keras, as explained below:

* Input Layer: Accepts text input.
* Vectorization Layer: Converts text to numerical vectors (96 max sequence length)
* Embedding Layer: Maps numerical vectors to word embeddings of 300-dimensions using pretrained model via FastText.
* Dropout Layers: Randomly drops input units to prevent overfitting.
* Bidirectional LSTM Layers: Processes input in both directions.
* Variational Dropout Layers: Regularizes LSTM cells.
* Self-Attention Layer: Captures contextual information.
* Dense Layers: Perform nonlinear transformations.(activation function is ReLU with 1000 neurons)
* Output Layer: Produces binary predictions (activation func is sigmoid with 1 neuron).
* Model Compilation: Defines training configuration.
* Callbacks: Monitors training progress.



Picture 3: Model Accuracy curve

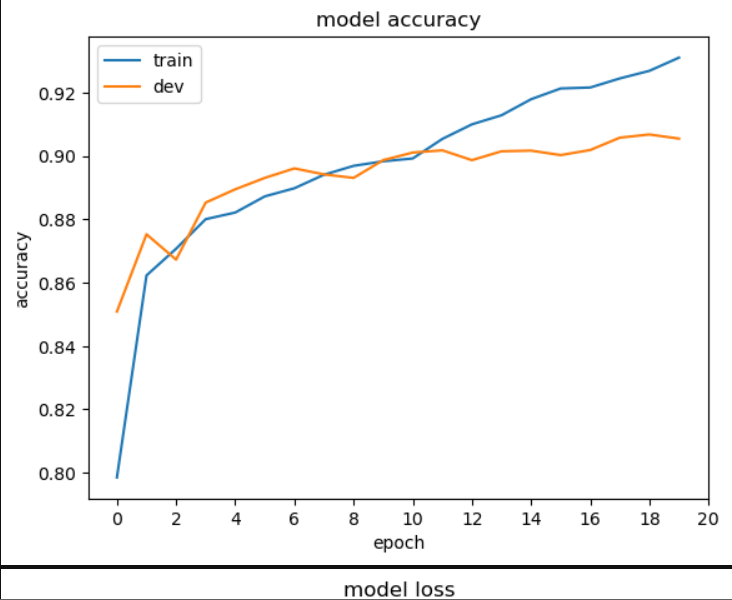


Picture 4: Model Loss curve

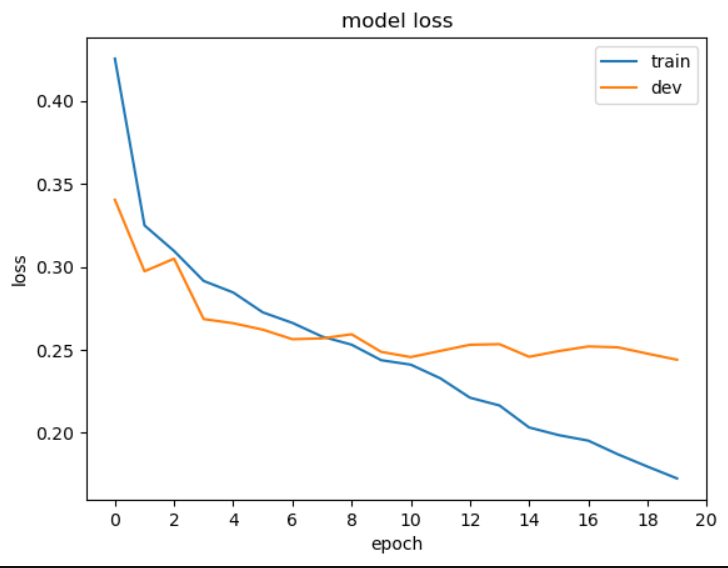
In the evaluation mode we achieved **accuracy of 85%.**

**Model-2 (BiDirectional GRU + Deep Self Attention)**

Secondly, we built a BiGRU using keras. The structure was the same as the model-1 (BiLSTM), however instead of BiLSTM layers we used BiGRU layers and Deep Self Attention mechanism (2 hidden layers in the attention mechanism.)



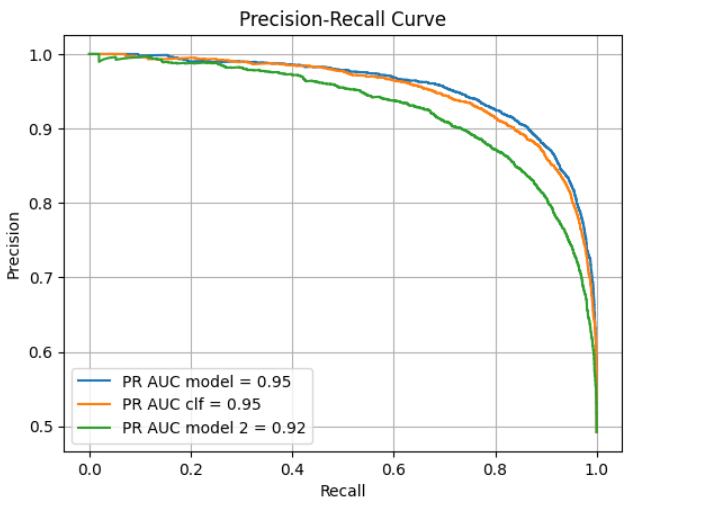
Picture 5: Model Accuracy curve



Picture 6: Model Accuracy curve

With this similar structure as before, the result was **accuracy of 91.5%** in the test set.

**PRECISION-RECALL CURVE**

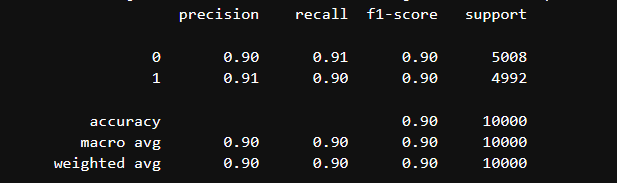
****

**Tune Best Perfomance Model via KERAS Tuner**

So, our best model is **BI-GRU with deep self attention mechanism**, thus we tried to tune it via KERAS TUNER. We ran 5 trials in order to fine tune our model and discover a better classifier.

However, due to lack of computing resources we could run 20 trials in order to find a better model.

The best results by the tuner are not better than our best classifier, nevertheless are the second best:



Picture 7: Classification Report by TUNER

**EXERCISE 10**

In order to develop a part-of-speech (POS) tagger for one of the languages of the Universal Dependencies treebanks we downloaded via link:

<https://github.com/UniversalDependencies/UD_English-EWT.git>

We built two functions in order to read the files and create a Dataframe with only the necessary data (id, token, UPOS tag).

We started by analyzing our data and make a preprocessing. We observed that we had 18 POS tags. However, one POS tag was the “\_” which is incorrect and we do not need it, thus we dropped the rows that include this POS tag.

Moreover, we observed that we had id’s of the form **(8, . , 1)** and these id’s had similar words, thus we had to drop these columns too.

**Remake Sentences from Corpus**

In order to re-make our sentences, we built a function called make\_sentences which is responsible for re-building the sentences of each word. For that purpose we used the id of each word and built our sentences again.



**Vectorize Our Dataset**

Moreover, we create a vectorizer with the use of our training dataset in order to create fixed size of length 96 for each sentence. By using the vectorizer we created word embeddings for the words of our training dataset with the support of FastText pretrained model (cc.en.300.bin). So, we had a vectorizer of fixed length and also word embeddings of 300-dimensions for each word of the training dataset.

**Baseline Model 1 – Logistic Regression**

We started using Logistic regression as our baseline model, which achieved accuracy of 87%.

**Baseline Model 2 – Multi-Layer Perceptron**

The MLP is identical to the one we built in Exercice 10 of Part 2 which was shown to achieve an accuracy of 89% (our best model so far).

**INTEGER REPRESENTATION OF Y LABELS**

In this part, the goal was to transform the original 17 classes into a representation that an MLP model can utilize and understand. For such a task the LabelBinarizer() class from sklearn was utilized to transform each of the target variables of each data set (y\_train, y\_dev, y\_test) and by using fit\_transform on the target variables of the trainset we transformed the target variables of the remaining data sets (i.e. .transform(y\_test) ).

Initially, the y\_train,y\_test,y\_dev contained numbers that mapped to the corresponding value (string) of the list holding all the target names. In our case, it was the unique values of the UPOS column. However, we had to add padding values in our y\_labels in order to include the same size as the X\_train value, thus we added padding values (0 value) of fixed length (96 max sequence length). Moreover, we did not need to make them one-hot vectors for the reason the we could use the sparse\_categorical crossentropy and in combination with the mask\_zero=0 we would not include these in the crossentropy loss.

**MODEL (BI-LSTM+LINEAR SELF ATTENTION SCORES)**

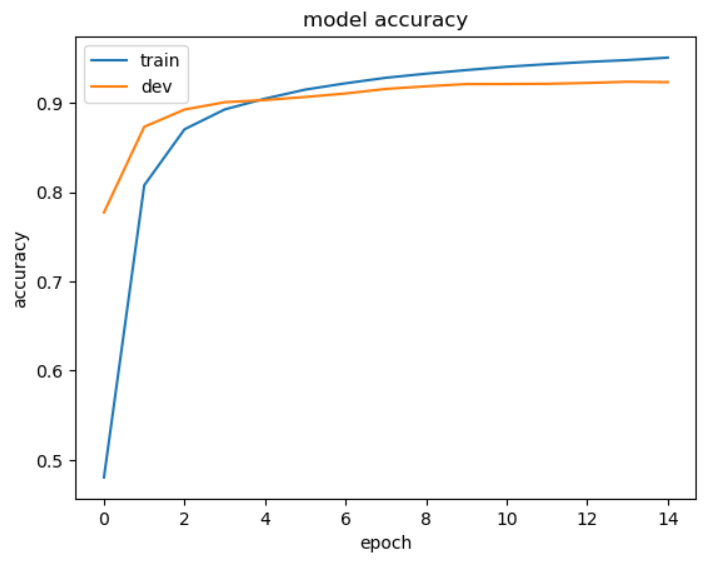
For this task we utilized a keras() Functional() model with an improved implementation. The key points here include the softmax() output layer activation function that is a multiclassification problem such as the one we encountered. The reason is that we aim to compute probabilities for all the classes in each output and softmax provides these valid probabilities.

We built a BiLSTM as our base model using keras, as explained below:

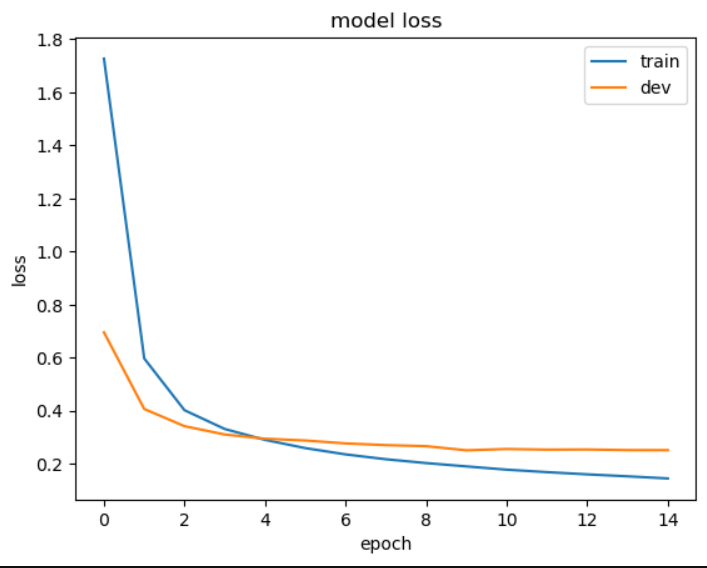
* Input Layer: Accepts text input.
* Vectorization Layer: Converts text to numerical vectors (96 max sequence length)
* Embedding Layer: Maps numerical vectors to word embeddings of 300-dimensions using pretrained model via FastText.
* Dropout Layers: Randomly drops input units to prevent overfitting.
* Two (2) Bidirectional LSTM Layers: Processes input in both directions.
* Variational Dropout Layers: Regularizes LSTM cells.
* Layer Normalization in between each LSTM layer.
* Dense Layers: Perform nonlinear transformations. (activation function is ReLU with 1000 neurons)
* Output Layer: Produces multiclass predictions (activation function is softmax with 18 neurons).
* Model Compilation: Defines training configuration.
* Callbacks: Monitors training progress.

For the backpropagation procedure we used Adam Gradient Descent Algorithm while for the loss metric we used Sparse Crossentropy Loss method which is suitable when we must manipulate integers encoded represantions of y\_train and not one-hot vector.

**EVALUATION OF BI-LSTM**

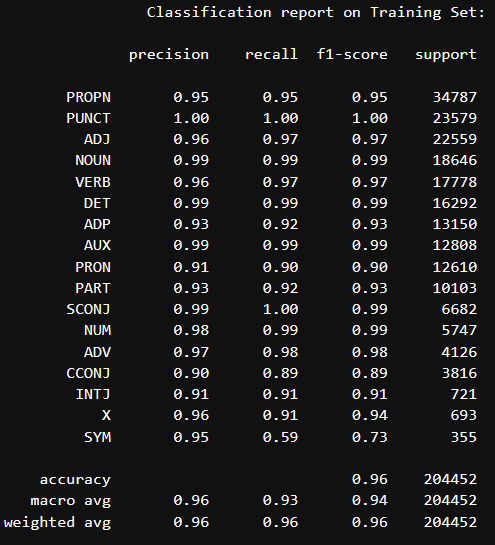
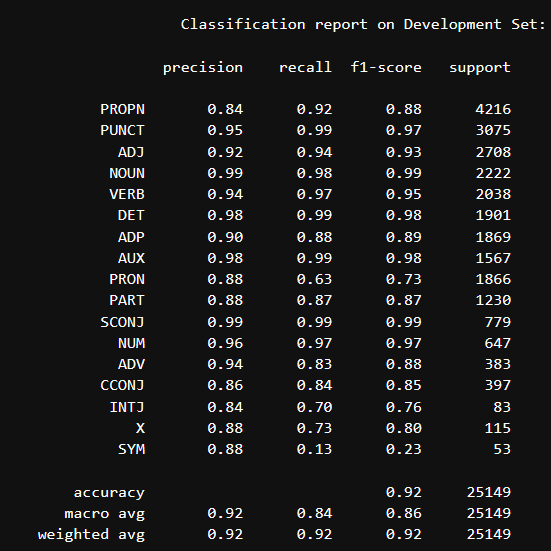


Picture 8: Model Accuracy for Bi-LSTM



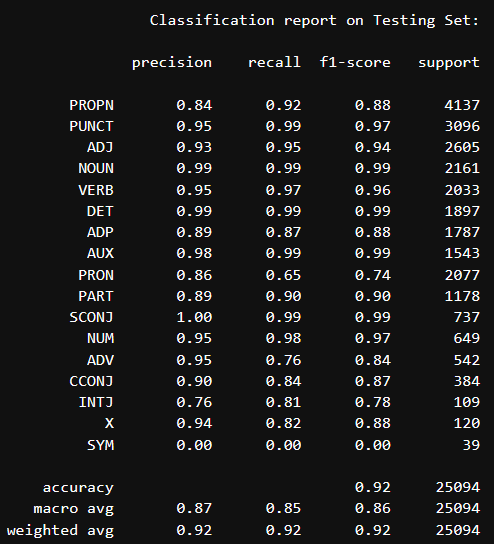
Picture 9: Model Loss for Bi-LSTM

The classification report of test set was:



Picture 10: Classification Report for Development set

Picture 11: Classification Report for Training set



Picture 12: Classification Report for Test set

This classification report provides a high-level overview of the RNN’s performance. Which in contrast to LogisticRegression and MLP, performs better. But we utilized more evaluation metrics and methods to develop a deeper understanding of our model and its weaknesses or identify potential improvement points.

Model accuracy and model loss both as a function of epochs, was calculated and visualized along with precision-recall AUC scores and their Macro Average. We highly recommend that you visit the corresponding notebook file to have a better view of the diagrams and calculations.

**Example in a Random Sentence**

We predicted the tags in a random sentence via our trained model, in order to check if it is efficient in random sentences which are not similar to the training set as the development and test set.

**Results:**



As we can observe from the above results our model is very efficient and tagging the words very accurately.

**Participation in the Assignment**

For this assignment each member of the team contributed equally.

Links for Google Collab notebook:

Exercise 1: [Exercise1\_Final (clf task).ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1w_l_Nk7grynLgaQW3XBh3uqwLAbFYQ_P)

Exercise 2: [Exercise2\_Final (pos\_tag).ipynb - Colaboratory (google.com)](https://colab.research.google.com/drive/1ZzMJ54aKLXzsu6XrEWKMOHUWbvGq3Qvz)