BUSINESS ANALYTICS
Master of Science

COURSE: Machine Learning and Content Analytics

PROFESSOR: H. Papageorgiou

Assistants responsible for this assignment: G. Perakis, A. Fergadis

DATE: 05/09/2021

NAME	**STUDENT-ID**
Nikolaos Gizelis	F2822003
Aristotelis Michailidis	F2822019

Contents

Introduction - The project and goals

The purpose of this machine learning project is to predict the annotated labels of the sentences of published scientific papers' abstracts. In terms of argument mining analysis, we focus on finding argumentative statements. The argumentative statements we annotate in this project in terms of argument mining analysis are:

1. Claim
2. Evidence,
3. No Label (by default)

Claim statements are usually sentences that report the study findings and derives from the author's original work as well as Evidence statement are usually the data and facts offered to support the Claim statement.

However, not all of the abstracts contain claims and evidences in their content. This is the difficulty that lies in this project. Claim should derive from the authors' research/study findings, even if there no explicit evidence in the abstract that supports those claims. In other words, the main goal of this project is to predict the argumentative statements.

The goal of this paper is to implement an algorithm that could predict accurately the sentences in terms of argument mining. In other words, the algorithm will be used as a tool to validate the strength of a scientific paper quickly via only it's abstract.

Methodology

In order to reach our goal, we had to follow some steps that are described briefly below.

Firstly, we conducted some explanatory steps of our dataset in order to have an overview of our data. We create a simple baseline based on explanatory data analysis. More specifically, we examine the data e.g., labels distribution in abstracts, we create lexicons. We utilize the insights to create a classifier that assigns labels to sentences.

Moving on, we implement various models with different architectures and search their hyper-parameters in order to find the one with the best behavior with our data. More specifically we perform 4 neural network models:

1. Feed Forward MLP with One-Hot Vectors
2. Feed Forward MLP with Self- Trained embeddings
3. Feed Forward MLP with Pre-Trained Glove embeddings
4. Convolutional 1D Neural Network (CNN)

Finally, we implement k-means approach for abstract clustering. We create clusters using only document embeddings from the abstract and its sentence.

Dataset

The abstracts of the published scientific papers were collected from the Horizon 2020. It is the biggest EU Research and Innovation programme ever with nearly €80 billion of funding available over 7 years (2014 to 2020) – in addition to the private investment that this money will attract. It promises more breakthroughs, discoveries and world-firsts by taking great ideas from the lab to the market.

The abstracts that were included in the final dataset are from different scientific fields. More information one may find here.

To begin with, data had been collected from the sources described earlier. Following the successful data gathering, the following step was to pre-process these papers in order to create the dataset. The dataset contains all sentences, as well as their labels, based on the papers collected.

After that, the dataset was processed once more to ensure that it could be used as a model input for predicting a paper's label. Finally, neural network models were developed, and more information about them, as well as the findings, will be presented in the next chapter.

The selection of the labels that each sentence of each abstract was assigned to, was performed by each team member/annotator. For the different labelling of the sentences, a classifier was used, called MACE, by learning competence estimates for each annotator and computing the most likely answer based on those competences.

Next, each of the abstract alongside with their labelled sentences were imported and then all of them were concatenated in order to create the final dataset. The final dataset has 32.004 rows and 3 columns: the sentence, the corresponding id document of the abstract that the sentence belongs to and the corresponding label of the sentence. A sample of the dataset is displayed below:

	doc_id	sentence	label
0	0	Concordance Between Different Amyloid Immunoas...	NEITHER
1	0	Importance Visual assessment of amyloid positr...	NEITHER
2	0	Several immunoassays have been developed to me...	NEITHER
3	0	The agreement between CSF Aβ42 measures from d...	NEITHER
4	0	Objective To determine the concordance between...	NEITHER
...	...	...	...
31999	1668	No statistically significant difference in con...	EVIDENCE
32000	1668	Latanoprost 0.005% once daily reduced IOP more...	NEITHER
32001	1668	Latanoprost had no statistically or clinically...	CLAIM
32002	1668	There was no difference in hyperemia between t...	CLAIM
32003	1668	Both concentrations of latanoprost reduced IOP...	CLAIM

32004 rows × 3 columns

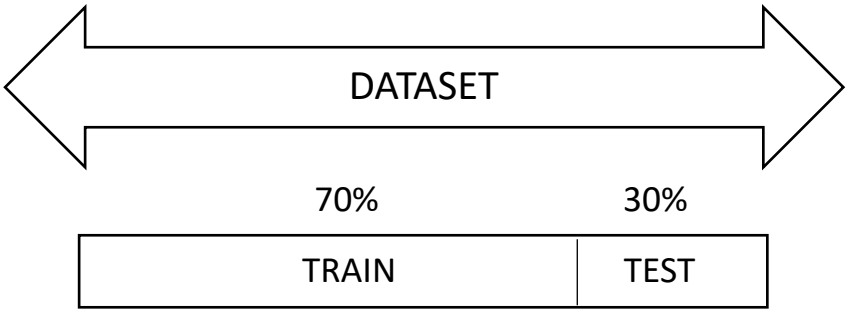
Figure 1. The final dataset

Pre-processing

Before moving to the implementation of the models, it is important to apply some pre-processing steps to our dataset. Firstly, we remove from the sentences all the stop words that exist in the corresponding English dictionary as well as characters such as numbers and punctuations. Secondly, we apply lemmatization to the sentences of the dataset. Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item.

Splitting the dataset

Moving on, the dataset is split to training dataset and to test dataset with stratified sampling with ratio 70:30, using the '*train_test_split'* library from *sklearn*. The 70% of the data will be used for training the models while 30% will be used for testing the models that are built out of it. Since we have heterogenous classes of target variable (label: Neither, Evidence, Claim), it is important to divide the dataset into relatively homogenous groups of samples. Finally, the training dataset consists of 22.402 sentences while the test dataset consists of 9.602 sentences.



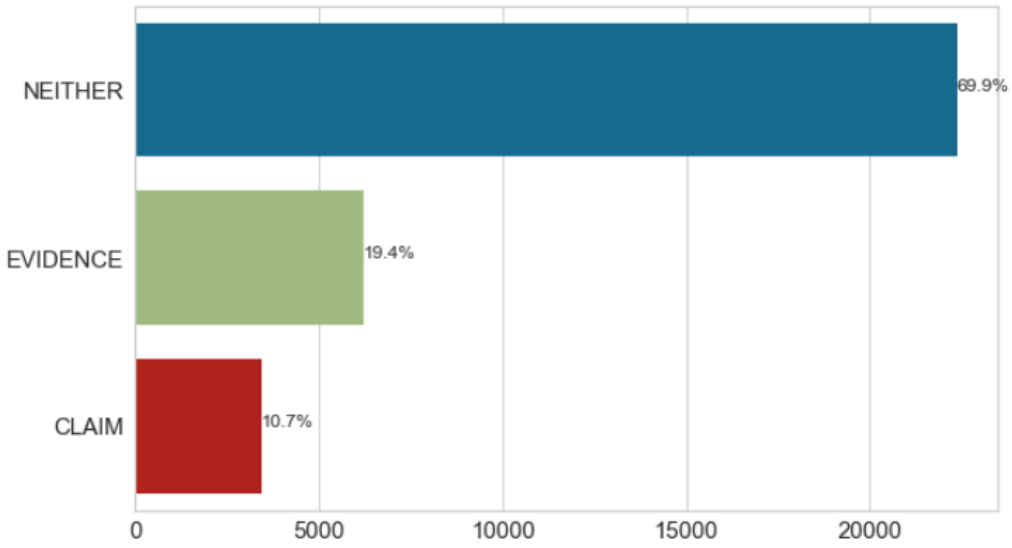
Intuitive Baseline

Before implementing any neural network model, we have to explore some data insights. First of all, by just assigning as claim the last sentence of each of the abstract, we have accomplished a 45% accuracy. In fact, 1550 out of 2686 of the abstracts have at least the last sentence assigned as Claim (57.7%). More specifically, with this method we predict 1550 out of 3419 sentences that are actually claim. Thus, our goal is to create a classifier that predicts the label of the sentences with more than 45% in terms of accuracy.

Label distribution

After the annotation processing has finished, we ended up with 2.686 abstracts with 32.004 sentences. Out of these:

- 3.419 were assigned as Claim
- 6.210 were assigned as Evidence
- 22.375 were assigned as Neither (by default)



In other words, sentences that do not have label (Neither), are almost 70% of the sentences from the final dataset. This means that, our target is to find a model that predicts correctly the labels of the sentences with accuracy more that 70%, which will be considered as the minimum threshold in terms of accuracy.

Lexicons

Next, we extract two subsets of the train dataset: one only with the sentences assigned as Claim and one only with the sentences assigned as Evidence. Based on these two datasets, we will investigate the most common words and regular expressions that occur in sentences labelled as Claim and as Evidence. We create the corresponding lexicons with the 25 most common words for each label.

To begin with, in order to check which words, occur in sentences that are Claims, we have to exclude common words, called stop words, such as “the”, “of”, “and”, etc. We conclude that the most common words are: ‘results’, ‘patient’, ‘conclusion’ and ‘treatment’ (see figure below). The same for evidence sentences, the most common words are: ‘p’, ‘patient’, ‘group’ and ‘difference’.

Worthy to mention at this point is that it is noticed that evidence sentences and Claim sentences share a lot of words that are same such as: ‘results’, ‘patient’, ‘level’ and ‘associated’.

The most common words for evidence and claim are depicted in the below word cloud plots.

Figure 6. Most common words in Claim sentences

Figure 6. Most common words in Evidence sentences

Finally, we are interested if we could predict the label of the sentences just by the occurrence of any word of the lexicon is in the sentence. The accuracy for claim sentences is 43% and for evidence is 55% as well as the recall metric is 79% and 74% respectively.

Based on the confusion matrix below, it is noticed that this method predicts correctly the labels for Claims of 4.194 out of total 9.602 sentences.

	Other	**Claim**
Other	815	211
Claim	5197	3379

Confusion matrix of claim sentences

Based on the confusion matrix below, it is noticed that this method predicts correctly the labels for Evidence of 5.303 out of total 9.602 sentences.

	Other	**Evidence**
Other	1392	471
Evidence	3828	3911

Confusion matrix of evidence sentences

Overall:

	Claim	**Evidence**
Accuracy	43%	55%
Misclassification error	56%	44%
Precision	13%	26%
Recall	79%	74%
F1- Score	23%	39%

Evaluation Metrics

Models

As discussed earlier, we are going to implement 4 different neural network models.

Feed Forward MLP with One-Hot Vectors

Next, we convert the train and test dataset of the target variable (labels) to categorical encoding as a one-hot numeric array. The input to this model is an array-like of integers, and more specific with 0 and 1. For example, for sentences labelled as 2 (=Claim), the corresponding transformation is [0,0,1] for our Y target variable. The 1 is putted in the corresponding column while the rest remain 0.

After, we apply text tokenization utility class only for the training dataset, with max words to be 15,000. This class allows to vectorize each sentence, by turning each text into either a sequence of binary integers (0 or 1) assigning 1 or 0 to the corresponding column-word (15,000 words).

The first model, that we are going to implement, is a deep feedforward network also known as MLP. The batch size, the number of training examples utilized in one iteration, is equal to 32 while the number of epochs, the number of complete passes through the training dataset, is equal to 20.

In order to avoid overfitting, we use a form of regularization called early stopping with the argument of patience to be 3. In other words, we constrain the number of epochs with no improvement after which training will be stopped to be equal to 3.

We create a sequential model, which is a linear stack of layers, by passing a list of layer instances to the constructor. We define an empty sequential structure and add a dense layer (MLP) with 64 neurons. We use a *relu* activation on MLP's outputs and we use a dropout layer with 40% of inputs dropped, in order to avoid overfitting. Finally, since we have 3 class (labels) we use a *softmax* activation on the last MLP's output. Once the model is built, we use the summary method to display its content (see below). It is noticed that the number of trainable parameters is equal to 960.259. This number represents the number of trainable elements in our network neurons that are affected by backpropagation.

```
Building model...
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                960064
_________________________________________________________________
activation (Activation)      (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 195
_________________________________________________________________
activation_1 (Activation)    (None, 3)                 0
=================================================================
Total params: 960,259
Trainable params: 960,259
Non-trainable params: 0
_________________________________________________________________
None
```

Before training the model, we need to configure the learning process, which is done via the *compile* method with argument loss (Loss function) to be an objective function called *categorical_crossentropy* and as metrics a function called *categorical_accuracy*.

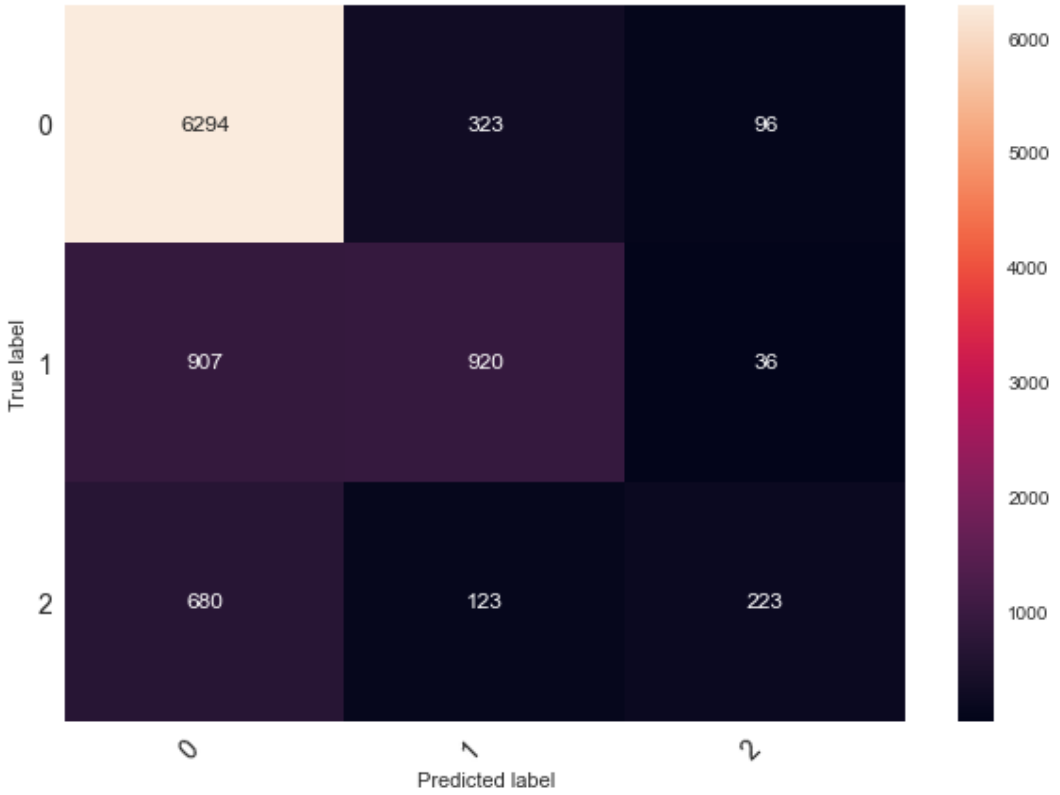
We fit the model in our training dataset with the appropriate features (train-test), labels, the number of epochs, we define the batch size and we use the early stopping class, it was described earlier. Moreover, we separate a portion of the training data into a validation dataset with ratio 20%.

Finally, we evaluate the model on test dataset and store on score variable. It returns the loss function value which is equal to 0.57 and metrics values, and more specifically the *categorical_accuracy* for the model in the test mode which is equal to 77.45% (see below).

```
301/301 [==============================] - 1s 2ms/step - loss: 0.5773 - categorical_accuracy: 0.7745
[0.5773221254348755, 0.7745261192321777]

Test accuracy: 77.453 %
```

Moreover, in order to summarize the performance of our first model we create a confusion matrix as well as a classification report. It is observed from the confusion matrix that the model predicts correctly the labels of 7.437 sentences out of 9.602 sentences.

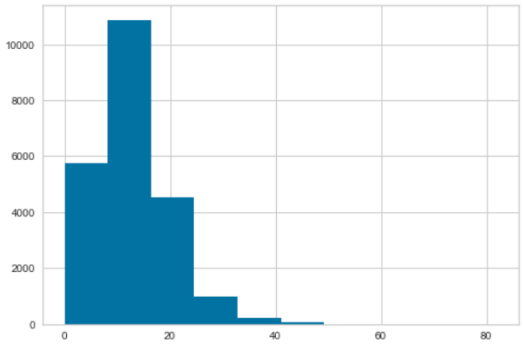


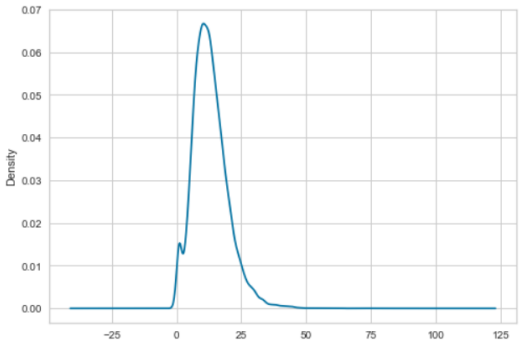
Based on the classification report, it is observed that, as previous the accuracy is 77.4%. Moreover, since our classes are imbalanced it is noticed that precision, recall and f1-score for the first class (Neither) are quite high, and for the classes 1 and 2 (Evidence & Claim, respectively) are relatively low.

	precision	recall	f1-score	support
0	0.7986	0.9376	0.8625	6713
1	0.6735	0.4938	0.5698	1863
2	0.6282	0.2173	0.3230	1026
accuracy			0.7745	9602
macro avg	0.7001	0.5496	0.5851	9602
weighted avg	0.7561	0.7745	0.7481	9602

Model with Self-Trained Embeddings

Firstly, we are going to explore the length of each sequence (sentence) by plotting a histogram and a density plot (see below). It seems that on average, the length of each sequence is around 15 words.



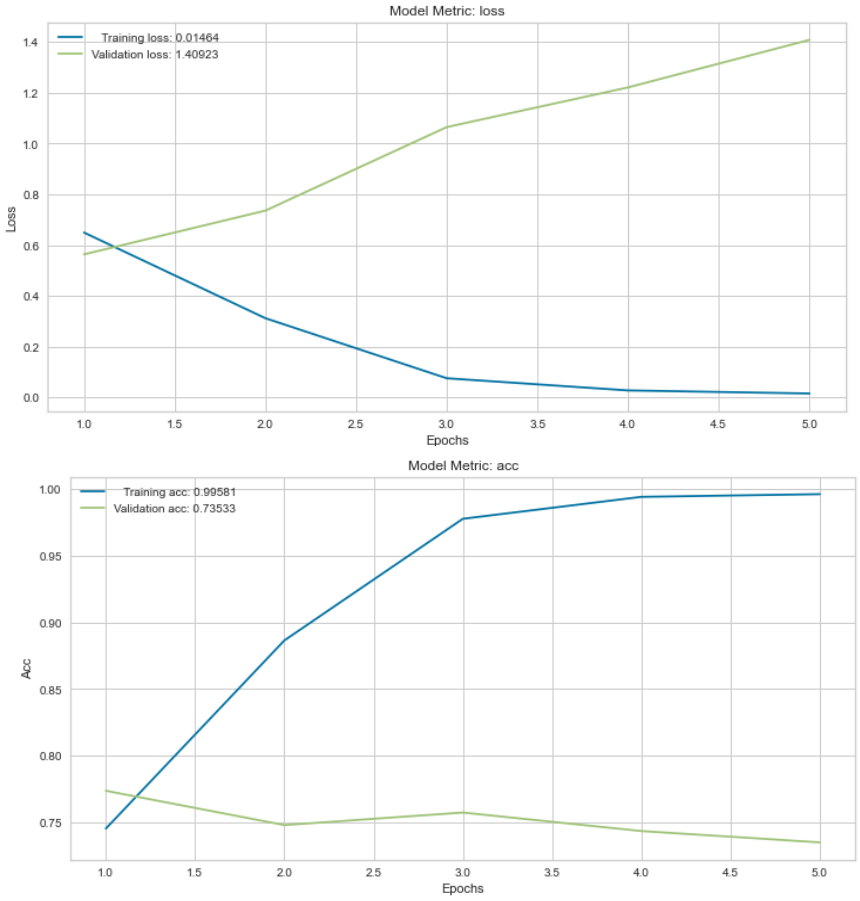


Based on that, we will use embedding, a function called *pad_sequences*, on the training as well as on the test dataset, with the max length of words to be equal to 50. This function transforms a list of sequences into a 2D Numpy array of shapes. Furthermore, we use an argument called *truncate* that cuts the words of sequences longer than 50 so that they fit the desired length. Since the valuable information is usually in the beginning of the sentences, we decided to keep the first 50 words of each sentence (if there are sentences that exceed the number of 50 words, otherwise it takes as input the whole sentence).

Same as before, we create a sequential model with the same layers like these one of the previous model, but now we use the aforementioned embedding layer and the model is named as *sequential_1*. Once the model is built, we its summary method to display its content (see below). It is noticed that the number of trainable parameters is equal to 5.460.259.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 50, 300)           4500000
_________________________________________________________________
flatten (Flatten)            (None, 15000)             0
_________________________________________________________________
dense_2 (Dense)              (None, 64)                960064
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 3)                 195
=================================================================
Total params: 5,460,259
Trainable params: 5,460,259
Non-trainable params: 0
_________________________________________________________________
None
```

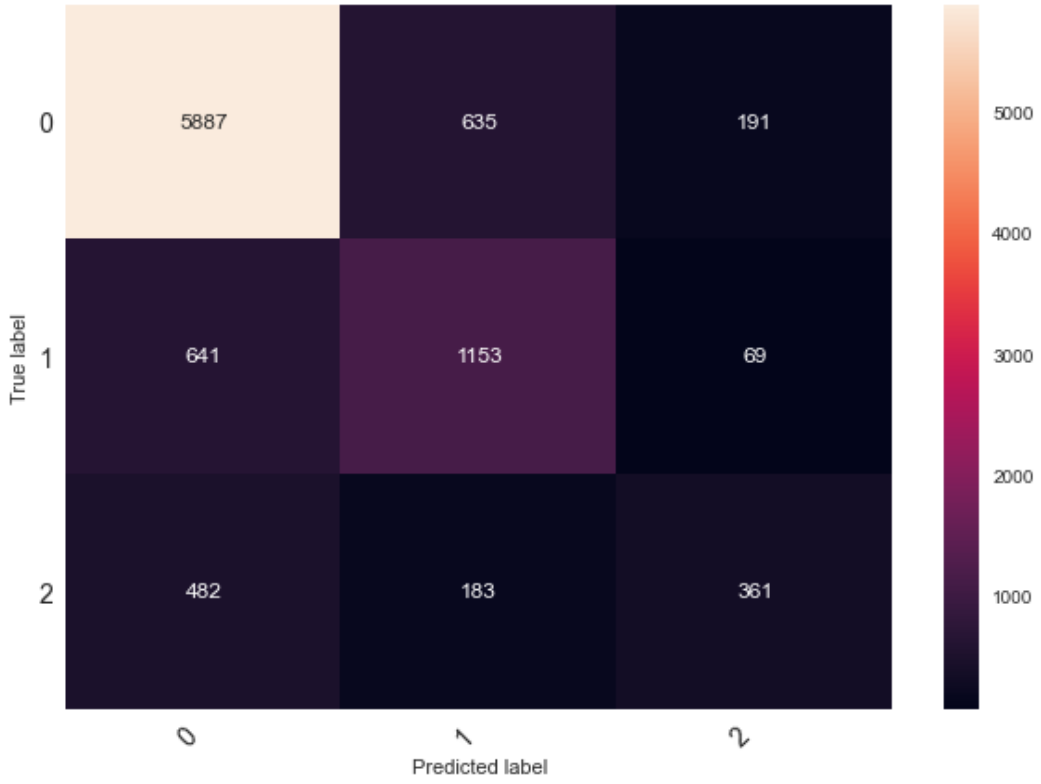
Next, we fit on training dataset and display two plots in order to present the accuracy and loss in both train and validation dataset (see below). It is noticed that the model performs different on both datasets and there is a sign of overfitting.



Second's model Accuracy and Loss per epoch

Next, we evaluate on test dataset. The model stopped on 5th epoch and the test accuracy that was displayed is equal to 77.08%.

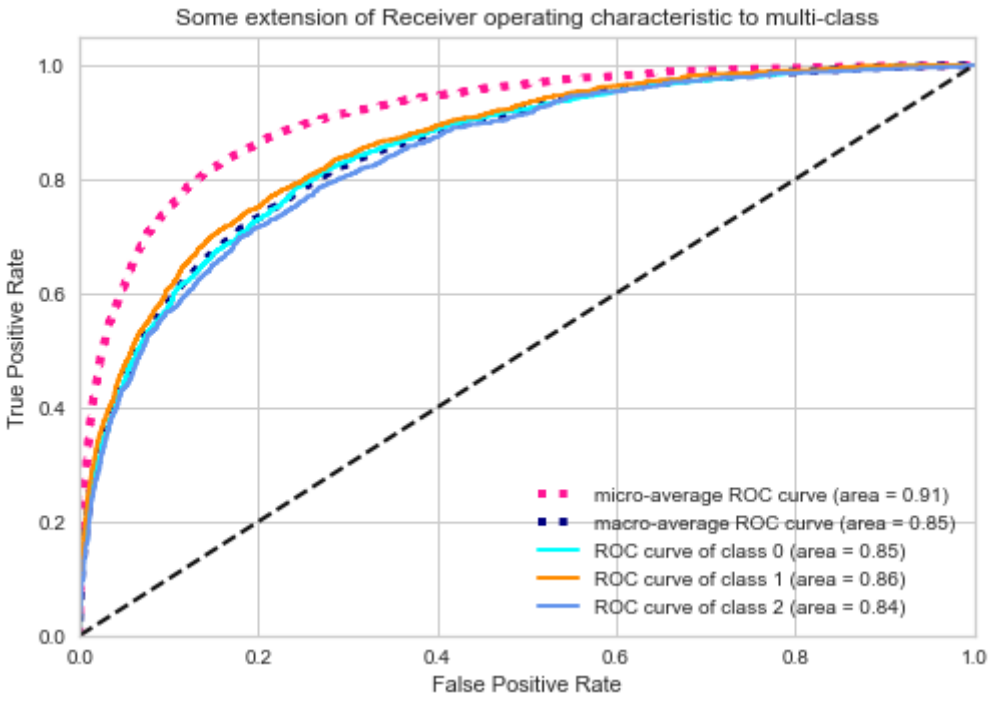
Moreover, in order to summarize the performance of our second model we create a confusion matrix as well as a classification report. It is observed from the confusion matrix that the model predicts correctly the labels of 7.401 sentences out of 9.602 sentences.



Based on the classification report, it is observed that, as previous the accuracy is 77.08% which is slightly better than the previous model. Moreover, since our classes are imbalanced, it is noticed that precision, recall and f1-score for the first class (Neither) are quite high, and for the classes 1 and 2 (Evidence & Claim, respectively) are relatively low. However, it is noticed that the metrics precision recall and f1-score are better than the ones of the previous model. Thus, this model is considered to be the best one till now.

	precision	recall	f1-score	support
0	0.8398	0.8770	0.8580	6713
1	0.5850	0.6189	0.6015	1863
2	0.5813	0.3519	0.4384	1026
accuracy			0.7708	9602
macro avg	0.6687	0.6159	0.6326	9602
weighted avg	0.7627	0.7708	0.7634	9602

Finally, we procced to the illustration of a plot for evaluation of classifier output quality. We plot the ROC curve (see below). A macro-average ROC curve will compute the metric independently for each class and then take the average, whereas a micro-average ROC curve will aggregate the contributions of all classes to compute the average metric. In our case, the macro average is equal to 85% and the micro average is equal to 91%. In addition, the ROC and AUC values of each class are only slightly different, that gives us a good indication of how good our model is at classifying individual class.



Model with Glove (Pre-Trained) Embeddings

Moving on, we are going to create a model with glove embeddings, which is an alternate method to create word embeddings. Although it is very similar to Word2Vec technique, Glove method was preferred because does not rely just on local context information of words but incorporates word co-occurrence to obtain word vectors.

We will use 300-dimensional GloVe embeddings of 400K words. Next, we create the vocabulary of our dataset that consists of all the unique words, which are in total 24.537 words. Worthy to mention at this point is that word embeddings are created using a neural network with one input layer, one hidden layer and one output layer.

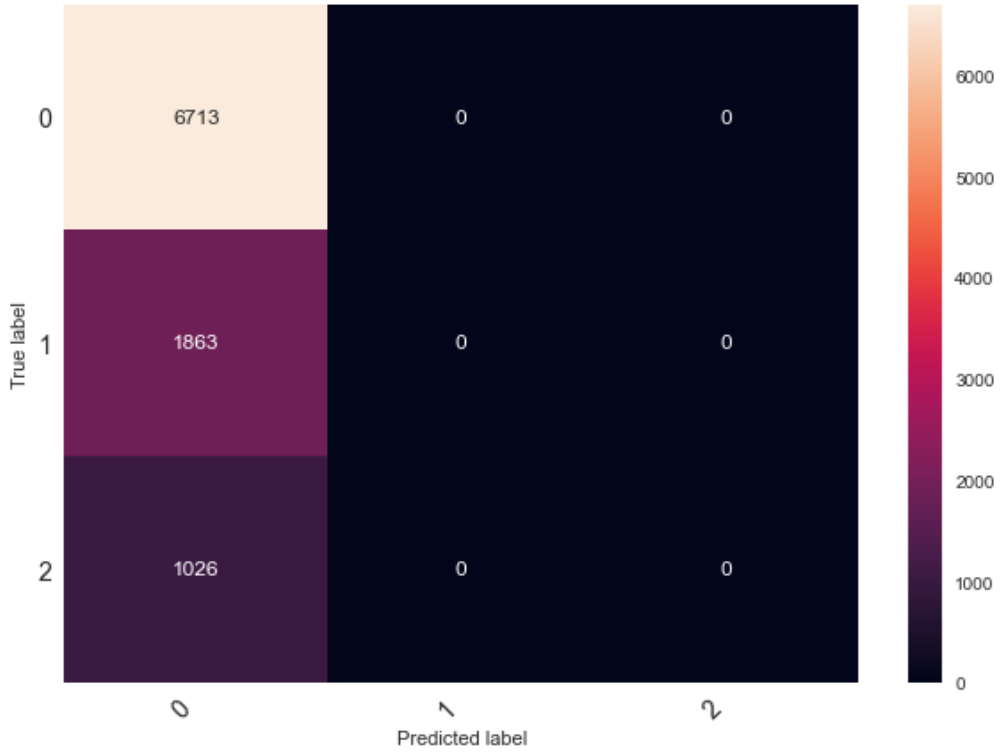
Next, we create the embedding matrix that in each of the 15.001 words has the corresponding unique ID for each word of our vocabulary and its vector of 300 values. Moreover, we create the sequential model with the aforementioned embedding. It is noticed that now we have far fewer trainable parameters. In fact, we have 1.937.027 trainable parameters out of possible total 6.437.327 parameters. The model is named *sequential_3.*

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 50, 300)           4500000
_________________________________________________________________
flatten_1 (Flatten)          (None, 15000)             0
_________________________________________________________________
dense_6 (Dense)              (None, 128)               1920128
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               16512
_________________________________________________________________
dropout_4 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_8 (Dense)              (None, 3)                 387
=================================================================
Total params: 6,437,027
Trainable params: 1,937,027
Non-trainable params: 4,500,000
_________________________________________________________________
None
```

Same as before, we fit on training dataset and evaluate on test dataset. The model stopped on 8th epoch and the test accuracy that was displayed is equal to 69.92%, which much lower than the two previous model.

Moreover, in order to summarize the performance of our third model we create a confusion matrix as well as a classification report. It is observed from the confusion matrix that the model predicts correctly the labels of 6.713 sentences out of 9.602

sentences. In fact, it actually assigns all the sentences with label 0 (Neither). As it was mentioned in the intuitive baseline, this is the minimum threshold in our study.



Based on the classification report, it is observed that, as previous the accuracy is 69.91%. However, since it actually assigns all the sentences with label 0 (Neither) except from two sentences, it is observed that metrics such as precision recall and f1-score of the labels 1 and 2 are zero (see below).

	precision	recall	f1-score	support
0	0.6991	1.0000	0.8229	6713
1	0.0000	0.0000	0.0000	1863
2	0.0000	0.0000	0.0000	1026
accuracy			0.6991	9602
macro avg	0.2330	0.3333	0.2743	9602
weighted avg	0.4888	0.6991	0.5753	9602

Convolutional Neural Network (CNN)

Finally, our last model is a convolutional 1D neural network (CNN). Although it is a type of artificial neural network used in image recognition, it has been explored for natural language processing problems such in the case of our study.

The first layer is the embedding layer which will learn an embedding for all of the words of the training dataset. Next, we define in the second layer, which is the convolutional layer, with filters to be equal to 25 and the kernel size to be equal to 3 (searching tri-grams). This allows us to train 25 different features on the first layer of the network.

Then through the Global Max Pooling layer we achieve the reduction of the input dimensionality.

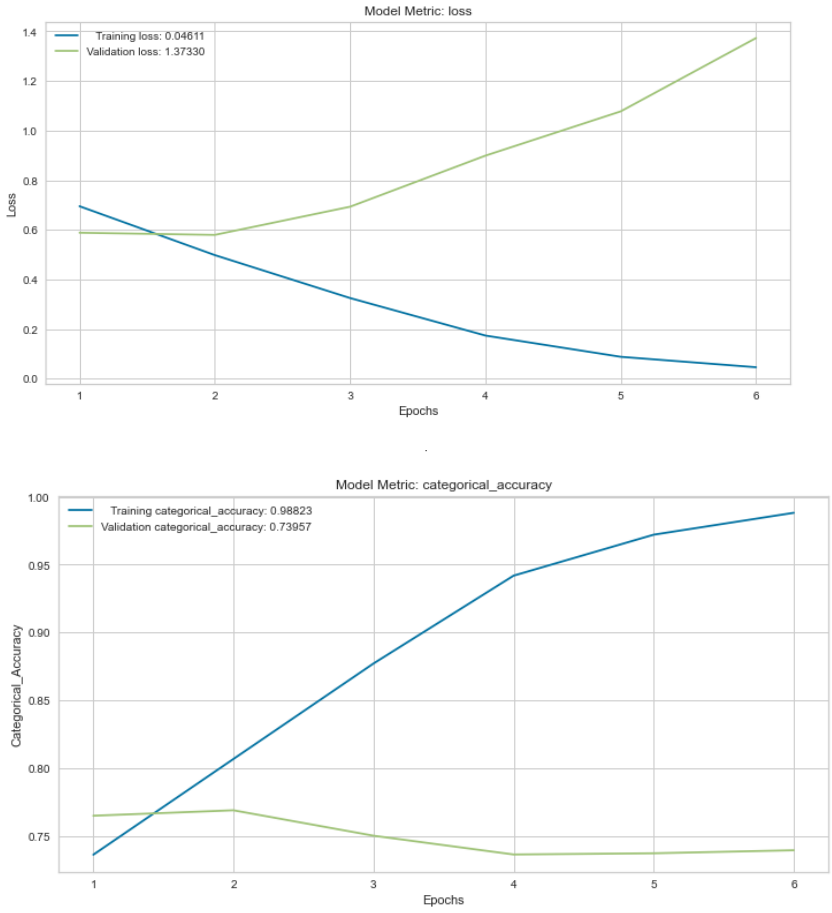
Once the model is built, we its summary method to display its content (see below). It is noticed that the number of trainable parameters is equal to 4.524.384 (equal to total parameters).

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 50, 300)           4500000
_________________________________________________________________
conv1d (Conv1D)              (None, 48, 25)            22525
_________________________________________________________________
global_max_pooling1d (Global (None, 25)                0
_________________________________________________________________
dense_9 (Dense)              (None, 64)                1664
_________________________________________________________________
dropout_5 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_10 (Dense)             (None, 3)                 195
=================================================================
Total params: 4,524,384
Trainable params: 4,524,384
Non-trainable params: 0
_________________________________________________________________
None
```

Before training the model, we need to configure the learning process, which is done via the *compile* method with argument loss to be an objective function called *categorical_crossentropy* and as metrics a function called *categorical_accuracy*.

We fit the model in our training dataset with the appropriate features, labels, the number of epochs (=50), we define the batch size (=32) and we use the early stopping class, as it was described earlier. Moreover, we separate a portion of the training data into a validation dataset with ratio 0.2.

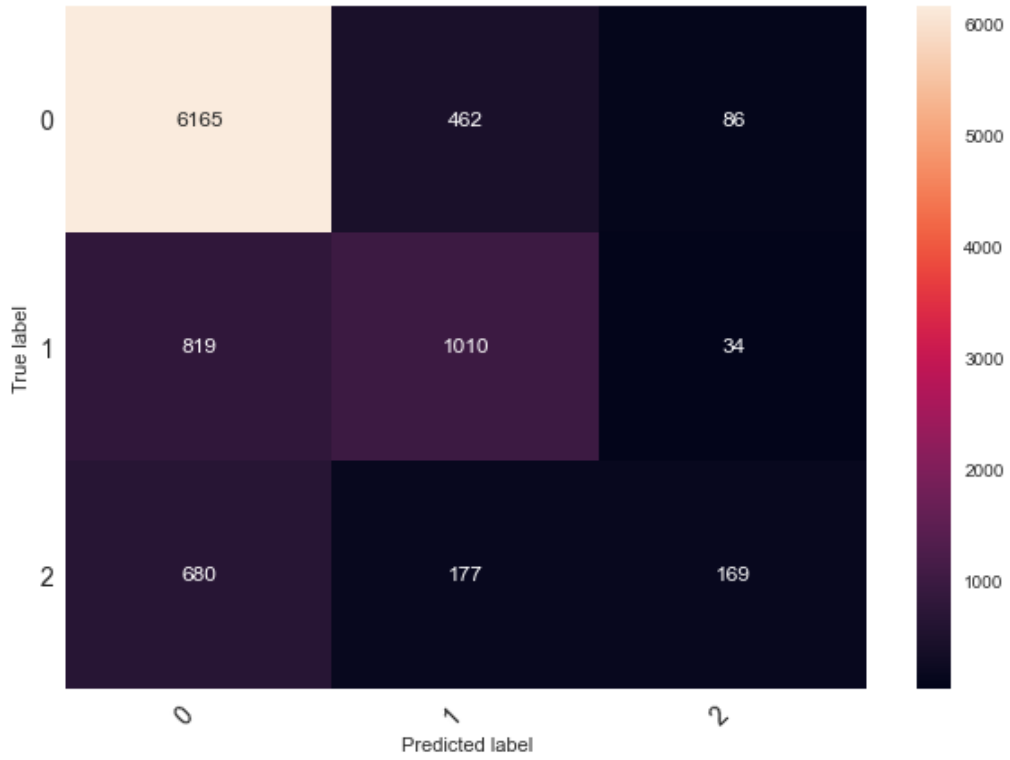
Next, we display the two plots in order to present the accuracy and loss in both train and validation dataset (see below). It is noticed again that the model performs different on both datasets.



CNN model Accuracy and Loss per epoch

Finally, we evaluate the model on test dataset and store on score variable. It returns the loss value which is equal to 0.58 and metrics values, and more specifically the *categorical_accuracy* for the model in the test mode which is equal to 76.48% (see below).

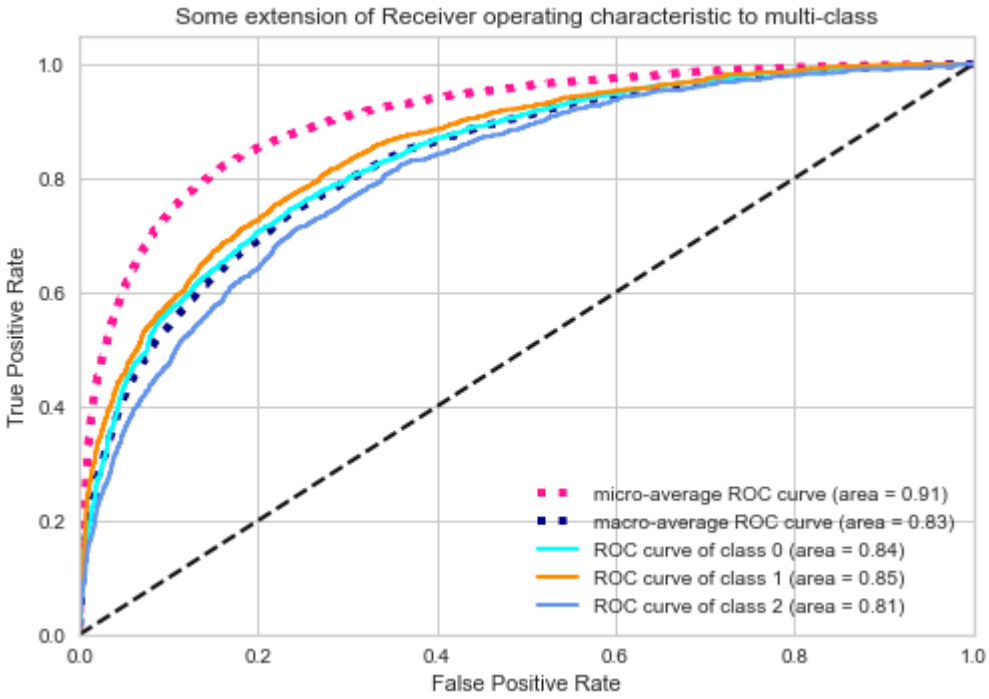
Moreover, in order to summarize the performance of our second model we create a confusion matrix as well as a classification report. It is observed from the confusion matrix that the model predicts correctly the labels of 7.344 sentences out of 9.602 sentences.



Based on the classification report, it is observed that, as previous the accuracy is 76.96% which is slightly worse than the best's so far model accuracy. The recall score of the label 1,2 and 3 is 91%, 54% and 16%, respectively.

	precision	recall	f1-score	support
0	0.8044	0.9184	0.8576	6713
1	0.6125	0.5421	0.5752	1863
2	0.5848	0.1647	0.2570	1026
accuracy			0.7648	9602
macro avg	0.6672	0.5417	0.5633	9602
weighted avg	0.7437	0.7648	0.7386	9602

Finally, we procced to the illustration of a plot for evaluation of classifier output quality. We plot the ROC curve (see below). In our case, the macro average is equal to 83% and the micro average is equal to 91%. In addition, the ROC and AUC values of each class are only slightly different, that gives us a good indication of how good our model is at classifying individual class.



CNN model ROC curve

Abstract Clustering

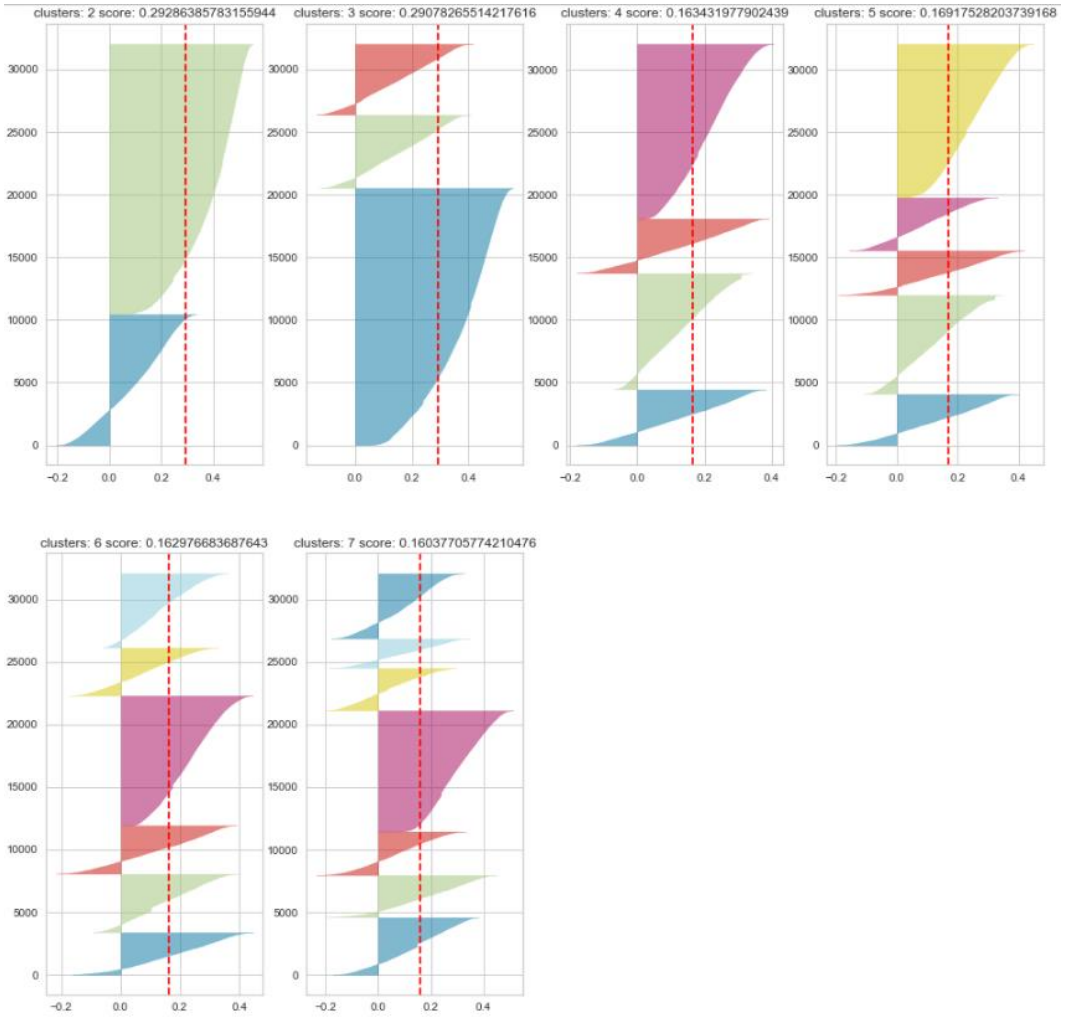
Moving on, we are going to implement the k-means approach with abstract embeddings. We are going to use the function *word_tokenize* from the NTLK library, in order to transform each sentence to a list with elements the words of each sentence.

Next, we apply the model called *Word2Vec* with each word to be 100 shape vector. This method takes as input a text corpus and produces the word vectors as output. The algorithm first creates a vocabulary from the training text data and then learns vector representations of the words.

To continue with, we use the function *vectorize*. This code will get all the word vectors of each sentence and average them to generate a vector per each sentence. More specifically, we define the vectorize function that takes a list of sequences and a gensim model as input, and generates a feature vector per document as output. Next, we apply the function to the sentences' tokens in tokenized_sent, using the Word2Vec model we trained earlier. Moving on, the function prints the length of the list of sentences and the size of the generated vectors.

Finally, we have created a matrix with shape (32004,100) that consists of the word vectors of each sentence.

Through the KMeans module we obtained the below diagram which represents the silhouette scores and diagrams for different number of clusters.



We notice that the optimal value regarding the silhouette value is for k=2 which means that the abstracts can be divided into two clusters.

Conclusion

To summarize, all the models have slight differences in terms of evaluation metrics except from the MLP with Glove embeddings (300 shape vectors) model which has lower accuracy than the others. Moreover, it seems that MLP with Self-Trained embeddings is the best one in terms of accuracy metric. However, the MLP model with Self-Trained embeddings has slightly lower accuracy but significantly higher precision, recall and f1-score metrics in the corresponding classes(labels). Thus, we conclude that MLP model with Self-Trained embeddings is the best model in our study in terms of predicting the labels of each sentence.

Worth to mention is that there are some components in the implementation of this study that carry a risk of randomness such splitting the dataset or the dropout argument in each model.

We also mention that in every model training phase we added a Model Checkpoint class which saves the best model's weights.

For the abstract clustering, we used the document embeddings for each sentence and applied the K-means algorithm for finding the optimal value of clusters in order for the abstracts to be grouped together appropriately.

Members and Roles

The team contributing in this study consists of two members:

- Nikolaos Gizelis and
- Aristotelis Michailidis,

Both of them are postgraduate students of Athens University of Economics and Business. However, Nikolaos is Electrical Engineer and Aristotelis has studied Mathematics.

Regarding the team roles, both of them contributed on the annotation of the specific abstracts (alongside with other members of this project) in the last 2 weeks of July 2021. In general, the load of work was split among the two members. As soon as the annotation processing finished, they started the data preparation in order to convert them to valid input for the models discussed earlier. Moreover, they continued on the creation and implementation of the models, described earlier in this study. The study finished on the first week of September 2021, when it was finally delivered for evaluation.

BIBLIOGRAPHY

- https://www.tensorflow.org/
- https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html
- https://e-mscba.dmst.aueb.gr/course/view.php?id=115
- https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939#:~:text=A%20Convolutional%20Neural%20Network%2C%20also,topology%2C%20such%20as%20an%20image.&text=Each%20neuron%20works%20in%20its,cover%20the%20entire%20visual%20field.
- https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network
- https://dylancastillo.co/nlp-snippets-cluster-documents-using-word2vec/