Athens University of Economics and Business

MSc in Business Analytics

Course*: Machine Learning and Content Analytics - PT*

Professor*: H. Papageorgiou*

Assistants professors*: G.Perakis & A. Fergadis*

Assignment: *SDG Classification*

Teammates

Moschou Christina (p2821914) & Tziraki Maria (p2821930)

Table of Contents

Why SDG 5 – Gender Equality?

As women of the 21st century, we do care about gender equality as it is the "unfinished business" of our time. Gender equality is not only a fundamental human right, but a necessary foundation for a peaceful, prosperous and sustainable world. Unfortunately, there is still a long way to go to achieve full equality of rights and opportunities between men and women, warns United Nations Women. Therefore, it is of paramount importance to end the multiple forms of gender violence and secure equal access to quality education and health, economic resources and participation in political life for both women and girls and men and boys. It is also essential to achieve equal opportunities in access to employment and to positions of leadership and decision-making at all levels.

It is a feasible topic to search and conclude to findings as there are the appropriate methods and tools which can predict with accuracy. The business workflows where our project belongs to are scientists, researchers, consultants of different backgrounds and everyday people who just love and are interested in Machine Learning.

Target

The purpose of this research is to provide a model which, given the paper will predict to which Sustainable Development Goal (SDG) this paper belongs to. This is a Multiclass Classification problem. We will apply MLPs, CNNs and RNNs and mixed models:

- Predict the SDG given the title
- Predict the SDG given the abstract
- Predict the SDG given the text
- Predict the SDG given the title and the abstract

Of course, we will comment on our findings, apply plots and conclude to the best one, according to its **accuracy**.

Data/Data Collection

In this part of the report, we are going to elaborate on the data collection of our research. More specifically, there is much data available on this topic. We focused on finding papers and scientific articles by searching in many websites, like Mendeley[1] and PubMed[2] which provide a plenty of citations.

Then, our next step was to search and extract papers which efficiently refer to the below targets[3]:

- 5.1. Gender-based violence, including domestic violence, is mitigated and reduced
- 5.2. Social protection and economic stimulus packages serve women and girls
- 5.3. People support and practice equal sharing of care work
- 5.4. Women and girls lead and participate in COVID-19 response planning and decision-making
- 5.5. Data and coordination mechanisms include gender perspectives

Of course, our data collection was not an "once-off" process; it consisted of **three batches** and each one of these batches included fifty texts regarding gender equality. Finally, we concluded to one hundred fifty texts. To extract the appropriate papers/texts for these batches we used queries like "discrimination against women", "violence against women", "value domestic work women", "gender differences". In addition, "gender equality and violence", "female genital mutilation Africa", "women participation", "gender equality in *education, *health; *jobs; *Africa; *Asia, *Europe", "women leadership", "COVID-19 women", "eliminate female genital mutilation", "women discrimination and exploitation" and "violence against women".

For each batch, we annotated every text that it included. More specifically, for each one of the texts our goal was to identify and annotate three components / layers. Namely:

- Discourse Topic
- Argument and
- Research Theme

The annotation process focuses on the annotation of the components. It is divided into three steps and the order in which they are listed below is not enforced.

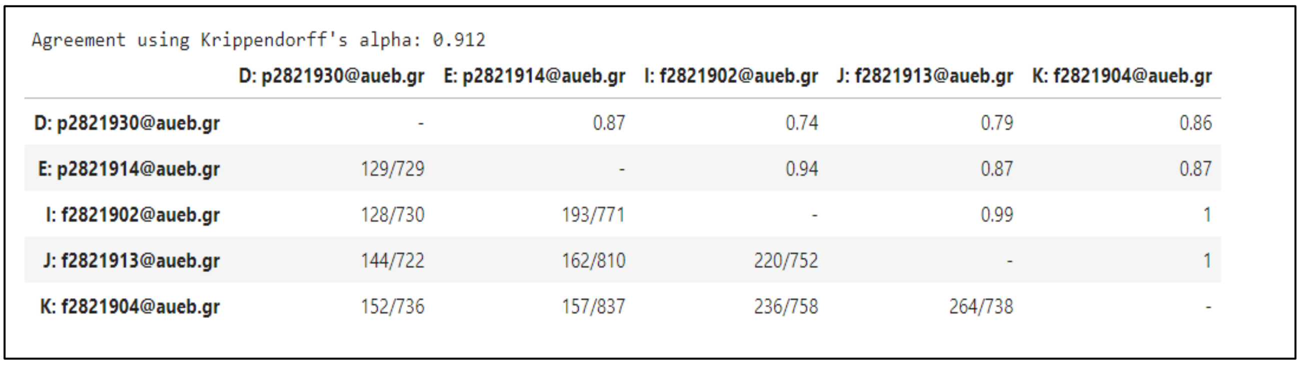
- Try to work out what the authors are writing about, what do they report (TOPIC).
- Try to work out the structure of the ARGUMENT:
  - Try to identify the claim(s) made
  - Adopt a skeptical stance towards the author's claims checking how these claims are supported by appropriate evidence(s).
- What is the goal of the research? It is named as RESEARCH THEME.

The results of the annotation process and some available KPIs are listed below:

Total & Pair-wise Agreement

In the following table the upper half is the pair-wise annotator agreement. The lower half has a pair of number for each pair of annotators. The first number is the number of the common annotated sentences and the second one the total sentences annotated by both annotators (intersection / union).

An agreement may seems good but you should also check the intersection number. For example, annotators X and Y might have agreement 1 but the common annotated sentences could be only 1.



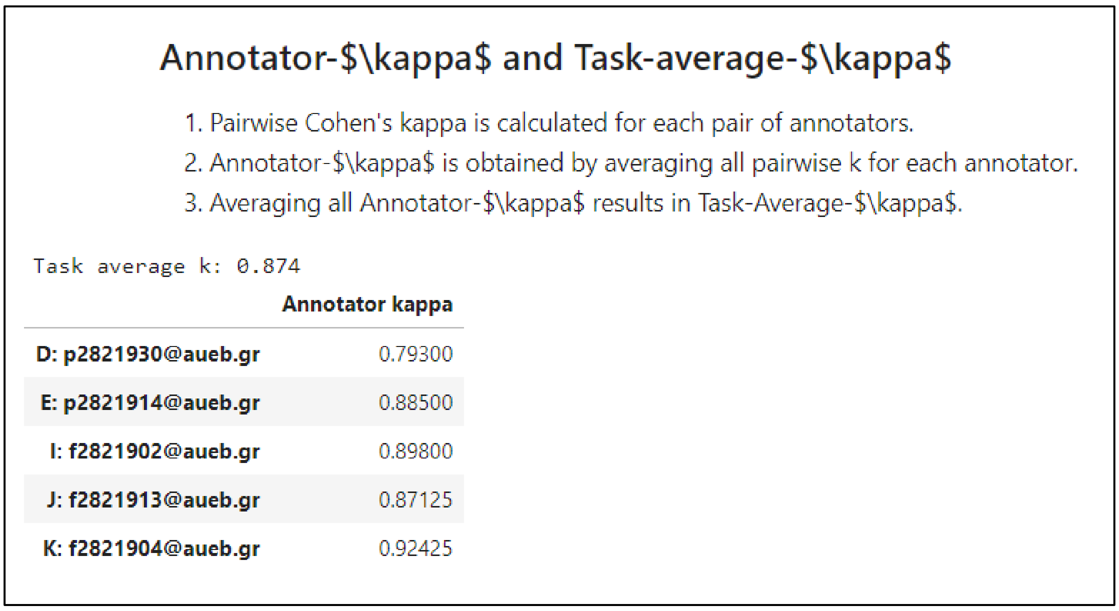
```
Agreement using Krippendorff's alpha: 0.912
```

	D: p2821930@aueb.gr	E: p2821914@aueb.gr	I: f2821902@aueb.gr	J: f2821913@aueb.gr	K: f2821904@aueb.gr
D: p2821930@aueb.gr	-	0.87	0.74	0.79	0.86
E: p2821914@aueb.gr	129/729	-	0.94	0.87	0.87
I: f2821902@aueb.gr	128/730	193/771	-	0.99	1
J: f2821913@aueb.gr	144/722	162/810	220/752	-	1
K: f2821904@aueb.gr	152/736	157/837	236/758	264/738	-

Table 1.Agreement table using Krippendorff's alpha: 0.912.

As we can observe, the agreement between the annotators is quite high.

Below, we may see the results for the task average k:



Annotator-κ and Task-average-κ

1. Pairwise Cohen's kappa is calculated for each pair of annotators.
2. Annotator-κ is obtained by averaging all pairwise k for each annotator.
3. Averaging all Annotator-κ results in Task-Average-κ.

```
Task average k: 0.874
```

	Annotator kappa
D: p2821930@aueb.gr	0.79300
E: p2821914@aueb.gr	0.88500
I: f2821902@aueb.gr	0.89800
J: f2821913@aueb.gr	0.87125
K: f2821904@aueb.gr	0.92425

Table 2.Task average

Of course, we did face many challenges during the annotation process. First of all, our team consists of two people and not three, so automatically the number of tasks per person increased. For instance, each one of us had to search for one hundred fifty abstracts and obviously annotate more of them. However, it was an interesting process with challenges as we had to agree as individuals, but simultaneously work individually as a team and annotate properly to have high agreement.

The process that followed annotation was the "Curation Process", in which every curator had to decide about "problematic" texts that he/she had not seen. Particularly, he/she had to "fix" by deciding about their topic, argument and research theme. This was, also, an interesting process during which we had to decide about the Topic, the Argument or/and the Research Theme of some abstracts that there were great discrepancies between the annotator's proposals.

After these processes, we are now ready to continue with the import of our dataset and apply our methods. Of course, our report and the Jupyter notebook containing our code, involve material included in the given material, but we have also done internet research.

To continue with, we can now focus on the import of our dataset. More specifically, we will use the script given in our last call in order to create a "csv" file including our data. By executing the appropriate commands, we will extract a csv file which contains the columns:

- Sdg
- extracted_title
- extracted_abstract
- initial_text and
- extracted

and each one respectively indicates:

- the sdg the paper belongs to
- its title
- its abstract
- the initial text and
- whether it was easily extracted or not

Regarding the last column of the file named "extracted", it is binary and takes the values "TRUE" or "FALSE" based on whether the title and the abstract of the paper were easily extracted from the script to the csv file. By checking the file, we conclude that there are 104 papers with value "FALSE". It is a number of texts easily manipulated manually in Excel. As a result, we choose to check manually whether these papers are truly correctly extracted or not. We conclude that most of them are correctly extracted, except for one or two, so we chose not to delete ant of the records and leave them as they were.

After this process in Excel, we are now ready to import our dataset in Python; it is named "sdg_df" and is a data frame. For its manipulation, we will need the "Pandas" library which is available in Python. "Pandas" is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Our next step is to take a look at our data by using commands like "shape()" and "head()". Our data frame consists of 889 rows and 5 columns and does not have any missing values. Of course, we are also going to use the "NumPy" library. "NumPy" is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. The core functionality of NumPy is its "ndarray", for n-dimensional array, data structure. These arrays are strided views on memory. In contrast to Python's built-in list data structure, these arrays are homogeneously typed: all elements of a single array must be of the same type[4]. In our assignment, we will use it to "feed" data.

Pre-Processing

The process of converting data to something a computer can understand is referred to as "pre-processing". In this stage, we will:

- remove all punctuations
- perform "word tokenization" and
- remove all English stopwords

We are now ready to continue with the pre-processing of the columns "extracted_title", "extracted_abstract" and "initial_text". First of all, we will remove all punctuations from our data. For this purpose, we will create three new columns named respectively "extracted_title_new", "extracted_abstract_new" and "initial_text_new" on which we will apply all our methods regarding the data cleaning. In addition, we will compute "word tokenization". Word tokenization is the process of splitting a large sample of text into words. This is a requirement in natural language processing tasks where each word needs to be captured and subjected to further analysis like classifying and counting them for a sentiment. The Natural Language Tool kit (NLTK) is the library that we will use to achieve this. We will, also, remove all English stopwords from our data: one of the major forms of pre-processing is to filter out useless data. In natural language processing, useless words (data), are referred to as stop words. A stop word is a commonly used word (such as "the", "a", "an", "in") that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query. We would not want these words to take up space in our database or taking up valuable processing time. For this reason, we can remove them easily, by storing a list of words that we consider as stop words. NLTK has a list of stopwords stored in 16 different languages.

Text Normalization

Text Normalization is also fundamental for NLP. This process includes all the techniques which transform a word into its root or basic form, in order to standardize text representation. It is the process of transforming text into a single canonical form that it might not have had before. Normalizing text before storing or processing it allows for separation of concerns, since input is guaranteed to be consistent before operations are performed on it.

Part-of-speech tagging (PoS tagger)

A part-of-speech (PoS) tagger is a software tool that labels words as one of several categories to identify the word's function in a given language. In the English language, words fall into one of eight or nine parts of speech. Part-of-speech categories include noun, verb, article, adjective, preposition, pronoun, adverb, conjunction and interjection.

PoS taggers use algorithms to label terms in text bodies. These taggers make more complex categories than those defined as basic PoS, with tags such as "noun-plural" or even more complex labels. Part-of-speech categorization is taught to school-age children in English grammar, where children perform basic PoS tagging as part of their education.

PoS taggers categorize terms in PoS types by their relational position in a phrase, relationship with nearby terms and by the word's definition. PoS taggers fall into those that use stochastic methods, those based on probability and those which are rule-based.

One of the first PoS taggers developed was the E. Brill tagger, a rule-based tagging tool. E. Brill is still commonly used today. Other tools that perform PoS tagging include Stanford Log-linear Part-Of-Speech Tagger, Tree Tagger, and Microsoft's POS Tagger. Part-of-speech tagging is also referred to as word category disambiguation or grammatical tagging.

PoS tagging is used in natural language processing (NLP) and natural language understanding (NLU).

In our project, we choose the Standford POSTagger which is a Java implementation.

Lemmatization

Our next step is to lemmatize**.** It is the process of grouping together the different inflected forms of a word so they can be analyzed as a single item.

For this implementation, we firstly download the appropriate files and place it in our working directory. Then, we will create a mapper for the arguments to wordnet according to the treebank POS tag codes. This mapper works as described below:

- Look for NN in the POS tag because all nouns begin with NN and maps 'NN': NOUN,
- Look for VB in the POS tag because all nouns begin with VB and maps 'VB': VERB,
- Look for JJ in the POS tag because all nouns begin with JJ and maps 'JJ': ADJ,
- Look for RB in the POS tag because all nouns begin with RB and maps 'RB': ADV

We will apply lemmas to the columns "extracted_title_new" and "extracted_abstract_new", as the column "initial_text_new" is the "union" of the two former. For this execution, we need to place in our code our path to java.exe.

Stemming

In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually enough that related words map to the same stem, even if this stem is not in itself a valid root.

Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation. A computer program or subroutine that stems word may be called a stemming program, stemming algorithm, or stemmer.

In our project, we will use the **"**PorterStemmer**"** and we will apply it to all of our columns ("extracted_title_new", "extracted_abstract_new" and "initial_text_new"). More specifically, "PorterStemmer" was written and maintained Martin Porter and is a process for removing the commoner morphological and inflexional endings from words in English. After the implementation of the **"**PorterStemmer**",** we conclude to our normalized columns named 'normalized_extracted_title', 'normalized_extracted_abstract' and 'normalized_initial_text' and our data is in the appropriate condition for further analysis.

Difference of Lemmatization and Stemming.

Lemmatization is the process of converting a word to its base form. The difference between stemming and lemmatization is, lemmatization considers the context and converts the word to its meaningful base form, whereas stemming just removes the last few characters, often leading to incorrect meanings and spelling errors.

Final Data Set

After executing the above processes, our data set now includes 6 more columns:

- To three of them has been applied the "basic cleaning" (remove punctuations, remove English stopwords and word tokenization) and these columns are:
  - "extracted_title_new"
  - "extracted_abstract_new" and
  - "initial_text_new"
- To the other three columns, we have also applied normalization techniques; apart from the basic data cleaning, we have applied POS Tagger, Lemmatization and Stemming and the columns that we conclude to are:
  - "normalized_extracted_title",
  - "normalized_extracted_abstract" and
  - "normalized_initial_text"

As a result, we are going to apply our models in each one of these columns.

Treating target variable

In this part of our project, we will apply our methods. Firstly, we will apply MLP, CNN and RNN with one variation. No matter which is the independent variable, the process will be:

- Split our data in train and test set
- Apply our model

Split of Data

During fitting, we will we split the data in 2 smaller datasets:

- Training dataset
- Test dataset (unseen dataset)

More specifically, as we fit each model in training sets, we made sure to add a validation split in the process, instead of making a separate validation dataset from the beginning.

The model is initially fit on a training dataset, which is a set of examples used to fit the parameters (e.g. weights of connections between neurons in artificial neural networks) of the model. The model (e.g. a neural net or a naive Bayes classifier) is trained on the training dataset using a supervised learning method, for example using optimization methods such as gradient descent or stochastic gradient descent. In practice, the training dataset often consists of pairs of an input vector (or scalar) and the corresponding output vector (or scalar), where the answer key is commonly denoted as the target (or label). The current model is run with the training dataset and produces a result, which is then compared with the target, for each input vector in the training dataset. Based on the result of the comparison and the specific learning algorithm being used, the parameters of the model are adjusted. The model fitting can include both variable selection and parameter estimation.

Successively, the fitted model is used to predict the responses for the observations in a second dataset called the validation dataset. The validation dataset provides an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters (e.g. the number of hidden units (layers and layer widths) in a neural network). Validation datasets can be used for regularization by early stopping (stopping training when the error on the validation dataset increases, as this is a sign of overfitting to the training dataset). This simple procedure is complicated in practice by the fact that the validation dataset's error may fluctuate during training, producing multiple local minima. This complication has led to the creation of many ad-hoc rules for deciding when overfitting has truly begun. In this direction, we will split and create our validation data in “Keras” when executing our code.

Keras

“Keras**”** is a deep learning API written in Python, running on top of the machine learning platform “TensorFlow”. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result as fast as possible is key to doing good research. “Keras” (κέρας) means horn in Greek. It is a reference to a literary image from ancient Greek and Latin literature, first found in the Odyssey, where dream spirits (Oneiroi, singular Oneiros) are divided between those who deceive dreamers with false visions, who arrive to Earth through a gate of ivory, and those who announce a future that will come to pass, who arrive through a gate of horn. It is a play on the words κέρας (horn) / κραίνω (fulfill), and ἐλέφας (ivory) / ἐλεφαίρομαι (deceive).

“Keras” was initially developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System)[5].

Tensorflow

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks.

TensorFlow offers multiple levels of abstraction so you can choose the right one for your needs. Build and train models by using the high-level Keras API, which makes getting started with TensorFlow and machine learning easy[6].

Finally, the test dataset is a dataset used to provide an unbiased evaluation of a final model fit on the training dataset.

The test dataset will work as the “unseen data” on which our model will be applied a give a sufficient prediction. To split our data in train-validation-test sets in we will use the “Stratified ShuffleSplit” cross-validator, which is available in the “sklearn.model_selection”. According to the official documentation of the “scikit-learn”, “Stratified ShuffleSplit” provides train/test indices to split data in train/test sets. This cross-validation object is a merge of StratifiedKFold and ShuffleSplit, which returns stratified randomized folds. The folds are made by preserving the percentage of samples for each class.

The size of the three sets is not standard and there is not a right or a wrong split; it depends on the data and their “nature”.

However, we are often required to convert the categorical i.e text features to its numeric representation. The two most common ways to do this is to use Label Encoder or OneHot Encoder. In our project, we choose “OneHot Encoder”. What one hot encoding does is, it takes a column which has categorical data, which has been label encoded and then splits the column into multiple columns. The numbers are replaced by 1s and 0s, depending on which column has what value. In our project, we’ll get six new columns, one for each sdg. For rows which have the first column value as sdg1, the ‘sdg1’ column will have a ‘1’ and all the other columns will have ‘0’s. Similarly, for rows which have the first column value as sdg2, the ‘sgd2’ column will have a ‘1’ and all the other columns will have ‘0’s and so on.

Vocabulary & Padding

Before executing the respective code, we will import our dataset and install the appropriate libraries (please check our Jupyter Notebook). In addition, we will do some research regarding our texts and their vocabulary. More specifically, we want to know the max words per entry, but also the average words per entry. In this way, we have a clear picture of the dimensions of our data. We proceed with extracting the size of our vocabulary. Due to different sizes of our texts, we are also going to do Padding, because we want the size of our input data to stay the same. To achieve this, we are going to import and use the “pad_sequences” library, which is available in Keras[7]. Our padding type will be “post”, which pads after each sequence; all the sequences are padded with zeroes in the end according to the longest sequence’s length or a chosen length longer than the longest length. In our data, this threshold is the maximum of total words count[8].

Models (MLP, CNN, RNN)

In the below graph, you may see neural networks that we will use:

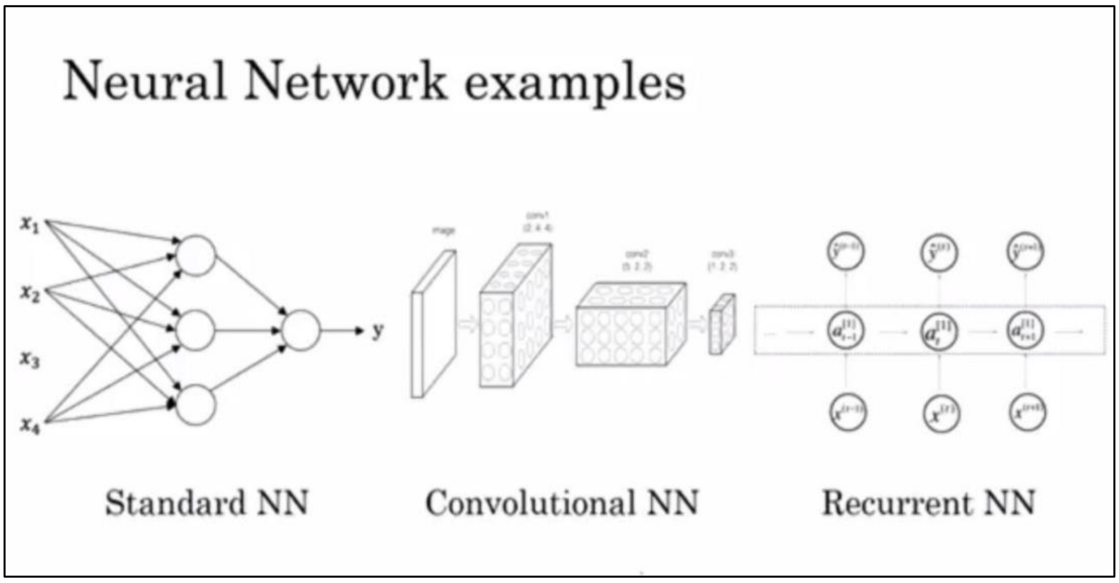


Figure 1.Neural Network example

MLP Model Build and Fit

In this part of the assignment, we are going to elaborate on **Multilayer Perceptron (MLP).** More specifically, a multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer[9].

An artificial neural network takes some input data and transforms this input data by calculating a weighted sum over the inputs and applies a non-linear function to this transformation to calculate an intermediate state. The three steps above constitute what is known as a layer, and the transformative function is often referred to as a unit. The intermediate states—often termed features—are used as the input into another layer[10].

Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Embeddings

Word embeddings are a type of word representation that allows words with similar meaning to have a similar representation and provide a dense representation of words and their relative meanings. They are an improvement over sparse representations used in simpler bag of word model representations. Word embeddings can be learned from text data and reused among projects. They can also be learned as part of fitting a neural network on text data. We can use word embeddings with Keras[11]. In practice, according to lecture's material, they work like described below:

1. We must specify how large we want the word vectors to be.
2. A 64-dimensional vector is able to capture good embeddings even for quite large vocabularies. It is an arbitrary hyperparameter and is set during fitting.
3. We also must specify:
   - for how many words we want embeddings
   - How long our sequences are.

However, our problem is a multiclass classification problem and not a binary classification problem. As a result, the One Hot Encoding method that we have already applied is appropriate.

We are now ready to build up our MLP model:

- A sdg belongs to one of the six possible categories, so the number of classes for the Y labels is equal to six.
- The number of Epochs indicates how many times we will train our Feed Forward Network and it is not standard because of early stopping. More specifically, we set this number quiet high and the training stops according to the validation loss.
- The batch size illustrates how many samples each time the algorithm takes to train the network.
- The Dropout Rate indicates how many neurons to shutdown each time and is equal to 0.4.
- The model that we will create is a sequential model meaning that each layer that we add per line will use as input the output of the former layer added to the model.
- We are going to use "relu" as activation function. "Relu" stands for rectified linear unit, mathematically is defined as y = max(0, x) and is the most commonly used activation function in the deep layers of a neural networks.
- We will also apply "early stopping". "Early stopping" is a method that allows you to specify an arbitrarily large number of training epochs and stop training once the model performance stops improving on the validation dataset. In the "early stopping", we will monitor the "validation loss" and stop the execution when the defined "patience" is transcended.
- To decrease and avoid overfitting, we also changed the "learning rate" from the default 0.001 to 0.01. We chose to do so in order to avoid many iterations with many changes of validation loss. In fact, we give a "boost" to the optimization algorithm so

as the neural network converges faster. Although this is a risky approach- as we jeopardize the finding of the global minimum, we did not observe any features indicating such scenario.

Our next step is to compile our model:

- A **loss function** is an error function and can estimate the loss of the model so that the weights can be updated to reduce the loss on the text evaluation. Due to the fact that our problem is a multiclass classification problem, we are going to use "categorical crossentropy" as a loss function.
- As on **optimizer**, we will use "Adam", which is more robust than the Gradient Descent.
- Our metric will be "accuracy".

We will continue with Fitting (training) our Feed Forward Network Model.

For this purpose, we will use:

- features (as dense inputs)
- labels
- the batch size
- the number of epochs
- the verbosity level and
- the validation split that we did before

Our next step is the Model Evaluation. We will evaluate and store on score variable on the test dataset. For this purpose, we will use again:

- features (as dense inputs)
- labels
- the batch size and
- the most extended verbose

We will, also, check the **prediction** ability of the model. We will conduct the **confusion matrix** and the **classification report** to evaluate the performance of the classifier.

- The general idea of the confusion matrix is to count the number of times instances of class A are classified as class B. Each row in a confusion matrix represents an actual class, while each column represents a predicted class.
- The classification report displays the precision, recall, F1, and support scores for the model.

To define the values: precision, recall, F1, and support scores and check whether the predictions are right or wrong, we will need the following metrics:

1. TN / True Negative: the case was negative and predicted negative
2. TP / True Positive: the case was positive and predicted positive
3. FN / False Negative: the case was positive but predicted negative
4. FP / False Positive: the case was negative but predicted positive

After the appropriate internet search and the knowledge of previous lectures, we can define:

- **Precision:** Precision is the ability of a classifier not to label an instance positive that is actually negative. For each class, it is defined as the ratio of true positives to the sum of a true positive and false positive.

  Precision: Accuracy of positive predictions and Precision = TP/(TP + FP)

- **Recall:** Recall is the ability of a classifier to find all positive instances. For each class it is defined as the ratio of true positives to the sum of true positives and false negatives.

  Recall: Fraction of positives that were correctly identified and Recall = TP/(TP+FN)

- **F1 score**: The F1 score is a weighted harmonic mean of precision and recall such that the best score is 1.0 and the worst is 0.0. F1 scores are lower than accuracy measures as they embed precision and recall into their computation. As a rule of thumb, the weighted average of F1 should be used to compare classifier models, not global accuracy.

  F1 Score = 2*(Recall * Precision) / (Recall + Precision)

- **Support:** Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing. Support does not change between models but instead diagnoses the evaluation process.

We, also, apply a **custom prediction function** in order to check that the One Hot Encoder predictions have worked properly and that we have only one "1" in each line.

CNN Model Build and Fit

In this part of the project, we are going to build, fit and comment on CNN models. Of course, we will follow the same steps as described above in the MLP model and "monitor" the same values.

A **convolutional neural network (CNN, or ConvNet)** is a class of deep neural networks. CNN is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.

CNN image classifications take an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). CNNs are regularized versions of multilayer perceptrons. Multilayer perceptrons usually mean fully connected networks, that is, each neuron in one layer is connected to all neurons in the next layer. The "fully-connectedness" of these networks makes them prone to overfitting data. Typical ways of regularization include adding some form of magnitude measurement of weights to the loss function. CNNs take a different approach towards regularization: they take advantage of the hierarchical pattern in data and assemble more complex patterns using smaller and simpler patterns. Therefore, on the scale of connectedness and complexity, CNNs are on the lower extreme[12].

RNN Model Build and Fit

A **recurrent neural network (RNN)** is a type of artificial neural network commonly used in speech recognition and natural language processing (NLP). RNNs are designed to recognize a data's sequential characteristics and use patterns to predict the next likely scenario. In an RNN, connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs[13]. All recurrent neural networks have the form of a chain of repeating modules of neural network.

LSTM is a special kind of RNN which- for many tasks has better performance than the RNN. LSTMs are explicitly designed to avoid the long-term dependency problem and remembering information for long periods of time is in fact their default behavior[14]. The way these networks are enhanced is by implementing an input, an output and a forget-gate in a single set. This allows the computation of the "hidden state", which is basically encoded information from previous time points.

Bidirectional LSTMs are an extension of traditional LSTMs that can improve model performance on sequence classification problems.

For all these reasons, for computing the RNN code, we will follow the steps described above and use the LSTM to maximize the performance (we used the code of the Lab's material).

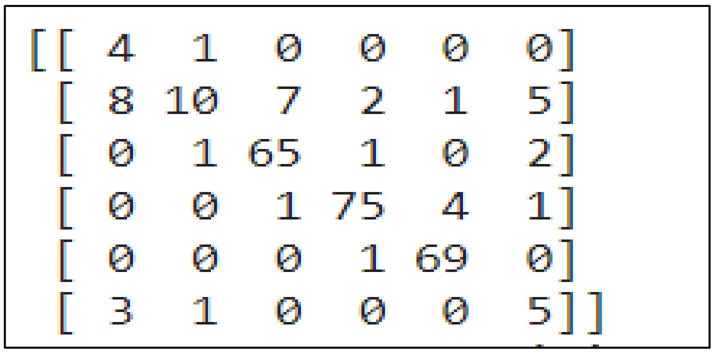
PART I - Models with ONE Variation

"initial_text_new" variable

In this part of the assignment, we are going to apply the above models and comment the respective features as described by applying the MLP, CNN and RNN models.

MLP Results

The **confusion matrix** is:

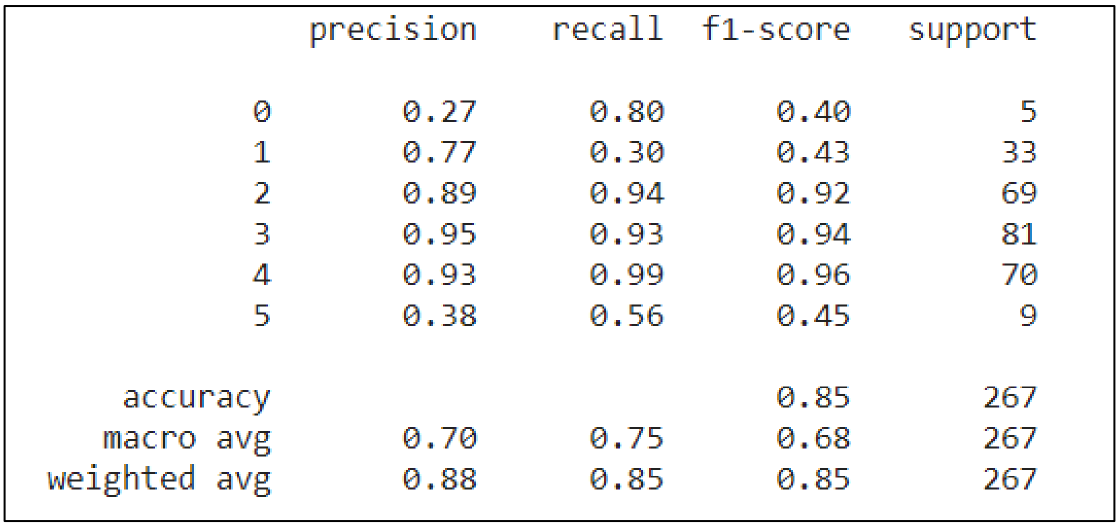


```
[[ 4  1  0  0  0  0]
 [ 8 10  7  2  1  5]
 [ 0  1 65  1  0  2]
 [ 0  0  1 75  4  1]
 [ 0  0  0  1 69  0]
 [ 3  1  0  0  0  5]]
```

Figure 2. Confusion matrix-initial_text_new-MPL

We observe that texts which belong to SDG 3, 5 or 13 are successfully classified to the SDG that they actually belong to. For instance, only one paper of SDG 5 is misclassified. However, the same pattern does not exist for the texts which belong to SDG 7,10 or 12. We expected that, as the amount of papers of these types is not as high as for the papers of SGD 3, 5 or 13. For instance, four papers that actually belong to SDG 7 are misclassified.

The respective **classification report** is displayed below:



```
              precision    recall  f1-score   support

           0       0.27      0.80      0.40         5
           1       0.77      0.30      0.43        33
           2       0.89      0.94      0.92        69
           3       0.95      0.93      0.94        81
           4       0.93      0.99      0.96        70
           5       0.38      0.56      0.45         9

    accuracy                           0.85       267
   macro avg       0.70      0.75      0.68       267
weighted avg       0.88      0.85      0.85       267
```

Figure 3.Clasiffication report-initial_text_new-MPL

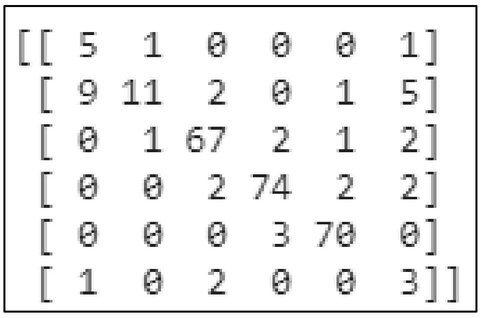
Taking under consideration the definitions mentioned above and classification report for the independent variable "initial_text_new", we conclude that:

- SDGs 3, 5 and 13 have the highest **precision scores**: 0.89, 0.95 and 0.93, respectively. In other words, they have the highest accuracy of positive predictions. On the other hand, SDGs 7,10 and 12 have the lowest scores. In fact, the respective precision values are 0.38, 0.27 and 0.77, but we must not omit the fact that there are not many available papers referring to these SDGs.
- In this direction, regarding **recall,** we observe again that SDGs 3, 5 and 13 have the highest performance.
- The same pattern occurs for **f1-score**; SDGs 3, 5 and 13 not only have the highest values, but they are also close to 1.0, which is the max value it can be equal to. However, this does not occur for SDGs 7, 10 and 12; SDG's 7 f1-score is 0.45, SDG's 10 is 0.40 and SDG's 12 is 0.43.
- **Support** indicates how many times a class actually occurs in the dataset that we work on. Of course, it "confirms" that there are not many abstracts related to SDGs 7, 10 and 12.
- Finally, the **accuracy is 85%**. In other words, 85% of the observations were correctly classified.

CNN Results

The respective results after applying the CNN model to the "initial_text_new" are displayed below:

The **confusion matrix** is:

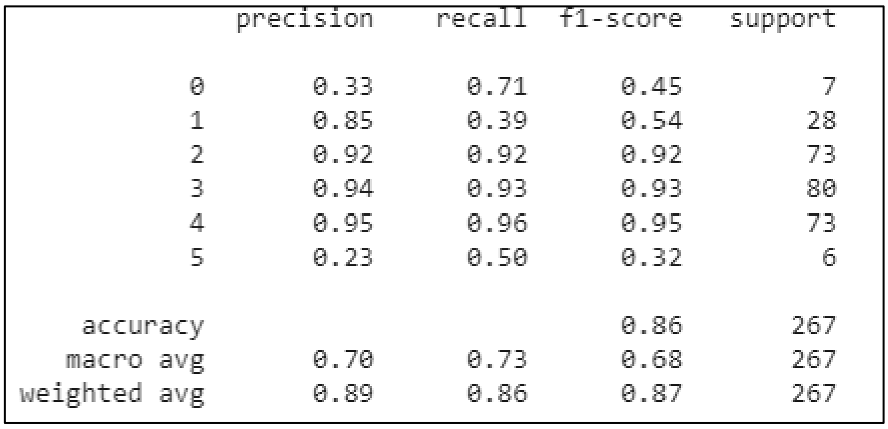


```
[[ 5  1  0  0  0  1]
 [ 9 11  2  0  1  5]
 [ 0  1 67  2  1  2]
 [ 0  0  2 74  2  2]
 [ 0  0  0  3 70  0]
 [ 1  0  2  0  0  3]]
```

Figure 4.Confusion matrix-initial_text_new-CNN

We can observe that texts which belong to SDGs 3, 5 and 13 are properly classified. On the other hand, texts from SDG 10 and 7 are misclassified.

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.33      0.71      0.45         7
           1       0.85      0.39      0.54        28
           2       0.92      0.92      0.92        73
           3       0.94      0.93      0.93        80
           4       0.95      0.96      0.95        73
           5       0.23      0.50      0.32         6

    accuracy                           0.86       267
   macro avg       0.70      0.73      0.68       267
weighted avg       0.89      0.86      0.87       267
```

Figure 5.Clasiffication report-initial_text_new-CNN

We conclude that:

- SDGs 13,3 and 5 have the highest **precision** (0.95, 0.96 and 0.93 respectively) and **f1-score** (0.91, 0.94 and 0.95 respectively) values. Their f1-score values are close to 1, which is the best value this feature can take. They, also, have the highest **recall** values (088, 0.92 and 0.97 respectively).
- However, SDGs 7, 10 and 12 have the lowest **precision** (0.54 0.13 and 0.00 respectively), **f1-score** (0.32, 0.21 and 0.00 respectively) and **recall** (0.23, 0.50 and 0.00 respectively) values.
- The model's **accuracy** is **84%**.

RNN Results

We apply the Bidirectional LSTM model and we get the below results:

The **confusion matrix** is:

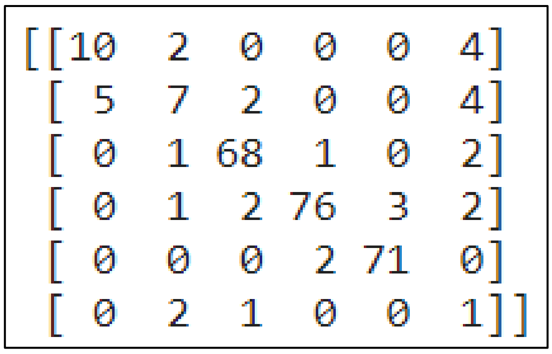

```
[[10  2  0  0  0  4]
 [ 5  7  2  0  0  4]
 [ 0  1 68  1  0  2]
 [ 0  1  2 76  3  2]
 [ 0  0  0  2 71  0]
 [ 0  2  1  0  0  1]]
```

Figure 6.Confusion matrix-initial_text_new-RNN

And we conclude that:

- 10 out of the 16 available texts regarding SDG 10 are properly classified when given only the "initial_text_new"
- In the same direction, we observe that 60 out of 68 texts belonging to SDG 13 are properly classified.
- Again, we expect better results regarding SDGs 3, 5 and 13 because of the available texts in our data.

The **classification report** is:

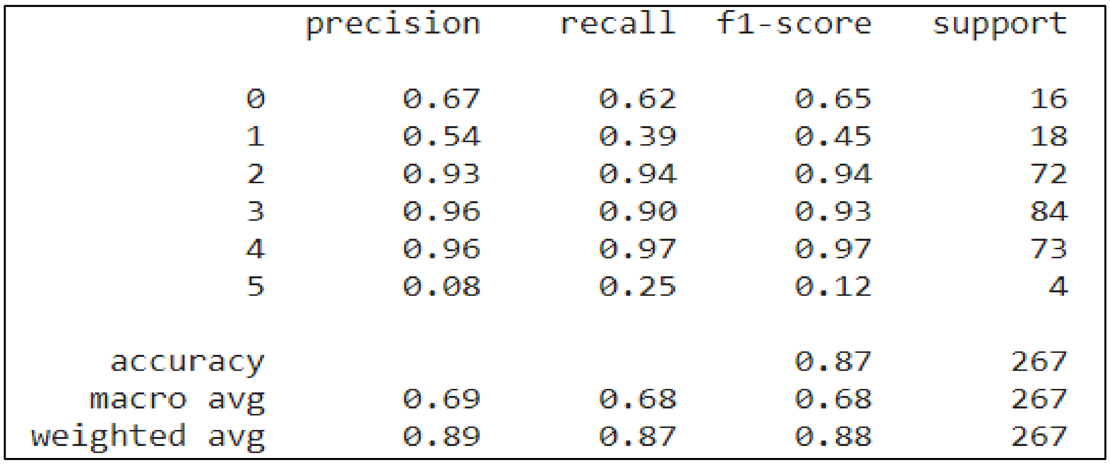

```
              precision    recall  f1-score   support

           0       0.67      0.62      0.65        16
           1       0.54      0.39      0.45        18
           2       0.93      0.94      0.94        72
           3       0.96      0.90      0.93        84
           4       0.96      0.97      0.97        73
           5       0.08      0.25      0.12         4

    accuracy                           0.87       267
   macro avg       0.69      0.68      0.68       267
weighted avg       0.89      0.87      0.88       267
```

Figure 7.Clasiffication report-initial_text_new-RNN

We conclude that:

- The classes of SDGs 13, 3 and 5 have higher performance in **precision** than the classes of SDGs 7, 10 and 12. In other words, they have higher accuracy of positive predictions.
- A similar trend exists when considering **recall and f1-score**. More specifically, we observe that SDGs 13, 3 and 5 have higher performance than the other SDGs and f1-scores are almost 1.00.
- The **support** values per class ensure that there are more available data regarding texts from SDGs 3, 5 and 13.
- The **accuracy** of the RNN model is **87%**.

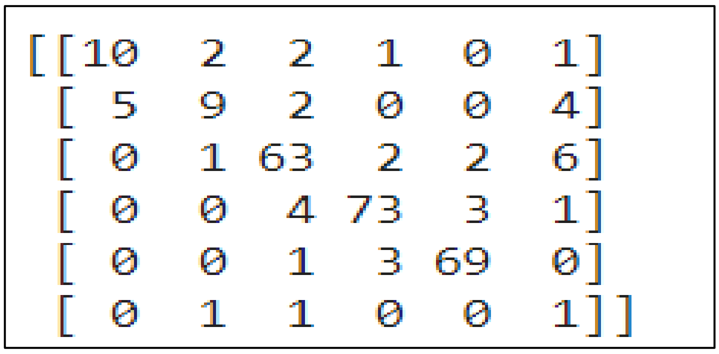
To continue with, to execute the MLP, CNN and RNN models for the other variables, each time, we replace in our code the variable's name with the name o the variable that we want to examine and execute again our code.

"extracted_title_new" variable

MLP Results

For applying the MLP model to the variable "extracted_title_new", we will use the same parameters that we have already mentioned.

The respective **confusion matrix** is:

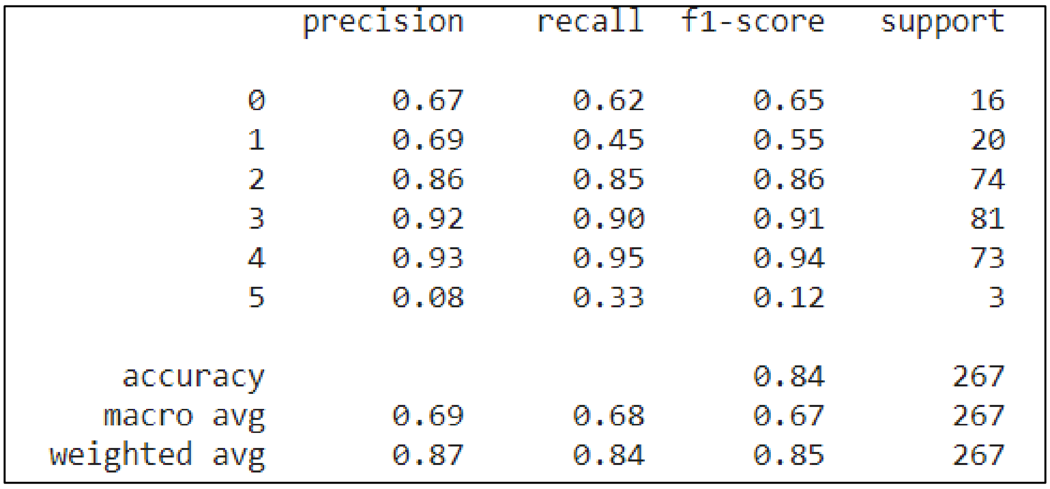


```
[[10  2  2  1  0  1]
 [ 5  9  2  0  0  4]
 [ 0  1 63  2  2  6]
 [ 0  0  4 73  3  1]
 [ 0  0  1  3 69  0]
 [ 0  1  1  0  0  1]]
```

Figure 8.Confusion matrix-extracted_title_new-MLP

We observe that texts which belong to SDGs 13, 3 and 5 are properly classified to the right class. However, this is not true for texts referring to SDGs 7, 10 and 12. Fir instance, only one text of the SDG 7 is correctly classified to the class of SDG 7. Of course, we must not omit the fact the number of the available texts of this class.

Considering the **classification report**, we have the below results:



```
              precision    recall  f1-score   support

           0       0.67      0.62      0.65        16
           1       0.69      0.45      0.55        20
           2       0.86      0.85      0.86        74
           3       0.92      0.90      0.91        81
           4       0.93      0.95      0.94        73
           5       0.08      0.33      0.12         3

    accuracy                           0.84       267
   macro avg       0.69      0.68      0.67       267
weighted avg       0.87      0.84      0.85       267
```

Figure 9.Clasiffication report-extracted_title_new-MLP

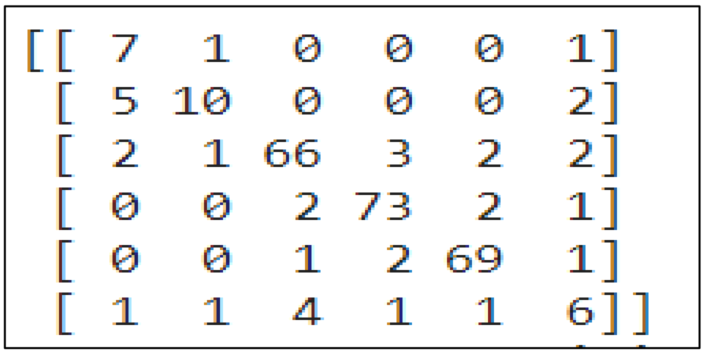
Once again, we conclude that **precision, recall and f1-score** are the highest for the classes of SDGs 3, 5 and 13. However, we do not have the same results for the rest classes. For example, for SDG 10, we have 16 actual occurrences in our data and the accuracy of the positive predictions (precision) is 0.67. It's f1-score is 0.65, which is a medium performance.

The model’s **accuracy** is **84%**.

CNN Results

In the same direction, we conclude to the below outputs:

The **confusion matrix**, which illustrates the number of times instances of class A are classified as class B, is:

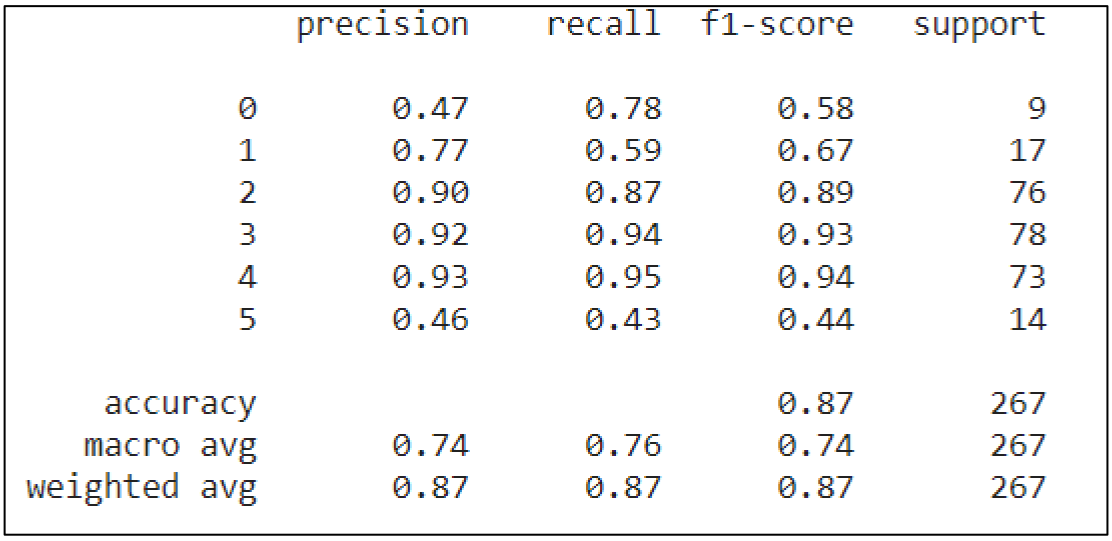


```
[[ 7  1  0  0  0  1]
 [ 5 10  0  0  0  2]
 [ 2  1 66  3  2  2]
 [ 0  0  2 73  2  1]
 [ 0  0  1  2 69  1]
 [ 1  1  4  1  1  6]]
```

Figure 10.Confusion matrix-extracted_title_new-CNN

Once again, texts of SDGs 13, 3 and 5 are properly classified in a sufficient extend, whereas texts belonging to SDGs 7, 10 and 12 do not have similar results.

When considering the **classification report**, we conclude to the following matrix:



```
              precision    recall  f1-score   support

           0       0.47      0.78      0.58         9
           1       0.77      0.59      0.67        17
           2       0.90      0.87      0.89        76
           3       0.92      0.94      0.93        78
           4       0.93      0.95      0.94        73
           5       0.46      0.43      0.44        14

    accuracy                           0.87       267
   macro avg       0.74      0.76      0.74       267
weighted avg       0.87      0.87      0.87       267
```

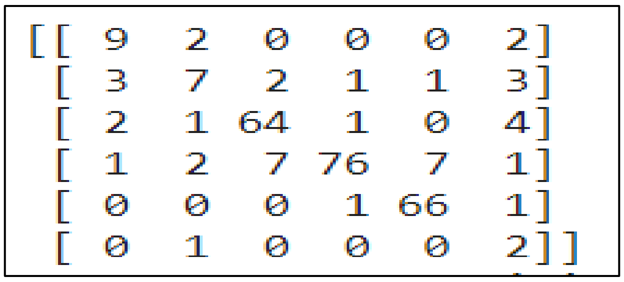
Figure 11.Clasiffication report-extracted_title_new-CNN

We can notice the high values of precision, recall and f1-score that the class of SDG 5 has. Particularly, precision is equal to 0.93, recall is 0.95 and f1-score is 0.94. However, its support is not the highest, as there are 73 actual occurrences of this class in the data that we consider.

The **accuracy** of the model is **87%**.

RNN Results

By following the same steps and taking onto account the same values, we apply the RNN model to the "extracted_title_new" variable and we get the below **confusion matrix**:

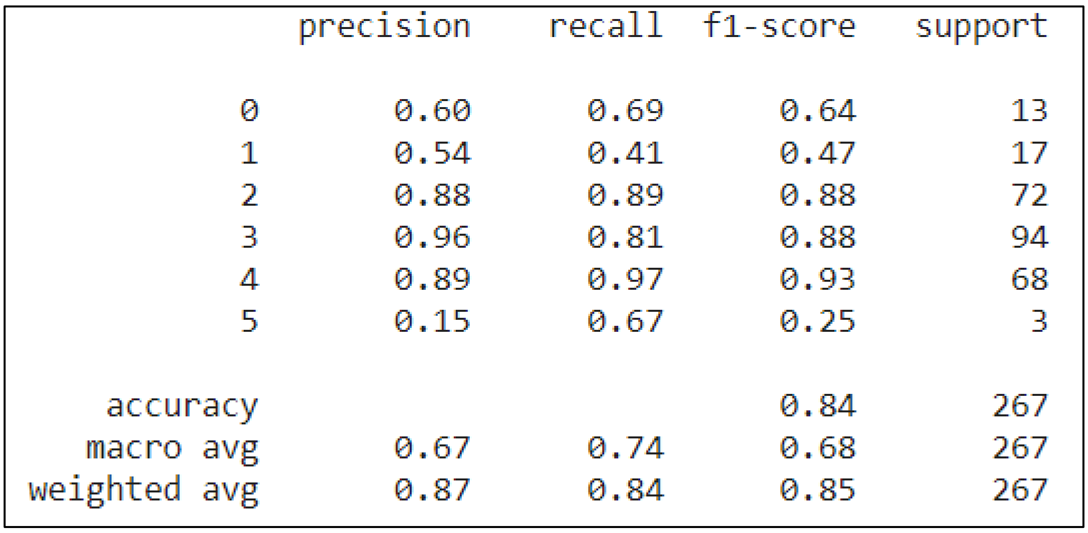


```
[[ 9  2  0  0  0  2]
 [ 3  7  2  1  1  3]
 [ 2  1 64  1  0  4]
 [ 1  2  7 76  7  1]
 [ 0  0  0  1 66  1]
 [ 0  1  0  0  0  2]]
```

Figure 12.Confusion matrix-extracted_title_new-RNN

It is obvious, that due to the luck of texts regarding SDG 12, the available texts are misclassified. Actually, ten of them are classified to wrong classes. However, this is not right for the class of SDG 5, where only two texts are wrongly classified.

The **classification report** is displayed below:



```
              precision    recall  f1-score   support

           0       0.60      0.69      0.64        13
           1       0.54      0.41      0.47        17
           2       0.88      0.89      0.88        72
           3       0.96      0.81      0.88        94
           4       0.89      0.97      0.93        68
           5       0.15      0.67      0.25         3

    accuracy                           0.84       267
   macro avg       0.67      0.74      0.68       267
weighted avg       0.87      0.84      0.85       267
```

Figure 13.Clasiffication report-extracted_title_new-RNN

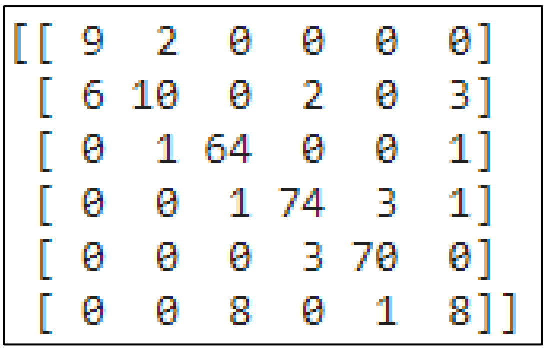
Once again, precision, recall and f1-score take their highest values when calculated for the classes of SDGs 13, 3 and 5. More specifically, SDG 3 has the highest precision, SDG 5 has the highest recall and f1-score. Support's maximum value 94 and illustrates the number of available texts of SDG 3.

The **accuracy** of the model is **84%**.

"extracted_abstract_new" variable

MLP results

After executing the respective MLP code for the "extracted_abstract_new" variable, the results of the **confusion matrix** are:

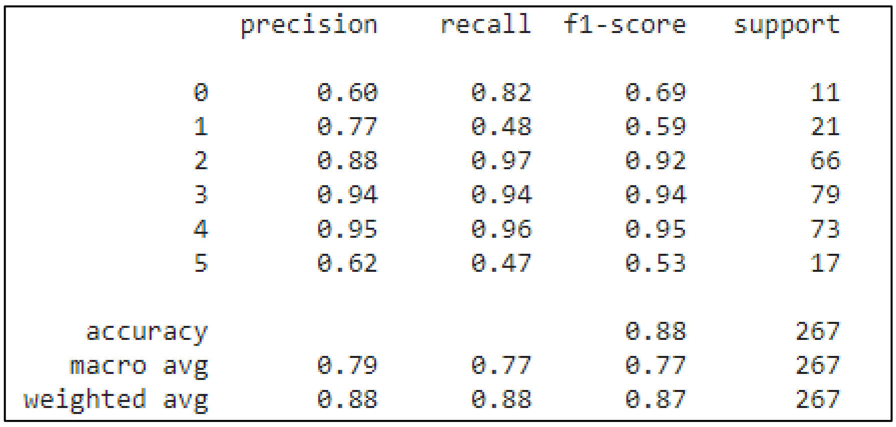


```
[[ 9  2  0  0  0  0]
 [ 6 10  0  2  0  3]
 [ 0  1 64  0  0  1]
 [ 0  0  1 74  3  1]
 [ 0  0  0  3 70  0]
 [ 0  0  8  0  1  8]]
```

Figure 14.Confusion matrix-extracted_abstract_new-MLP

We notice that the elements of SDG 12 are not properly classified; 19 elements of this class are classified to wrong classes.

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.60      0.82      0.69        11
           1       0.77      0.48      0.59        21
           2       0.88      0.97      0.92        66
           3       0.94      0.94      0.94        79
           4       0.95      0.96      0.95        73
           5       0.62      0.47      0.53        17

    accuracy                           0.88       267
   macro avg       0.79      0.77      0.77       267
weighted avg       0.88      0.88      0.87       267
```

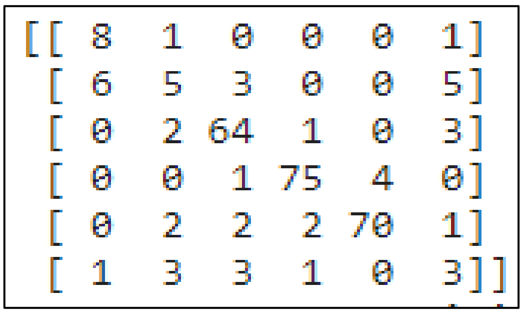
Figure 15.Clasiffication report-extracted_abstract_new-MLP

We observe that f1-score almost reaches its maximum values (1.00) for the classes 3 and 4, which are the classes of SDGs 3 and 5, respectively. At the same time, these two classes have the highest support. Classes 2 and 3 (SDGs 13 and 3 respectively) have the highest recall.

The **accuracy** of the model is **88%**.

CNN results

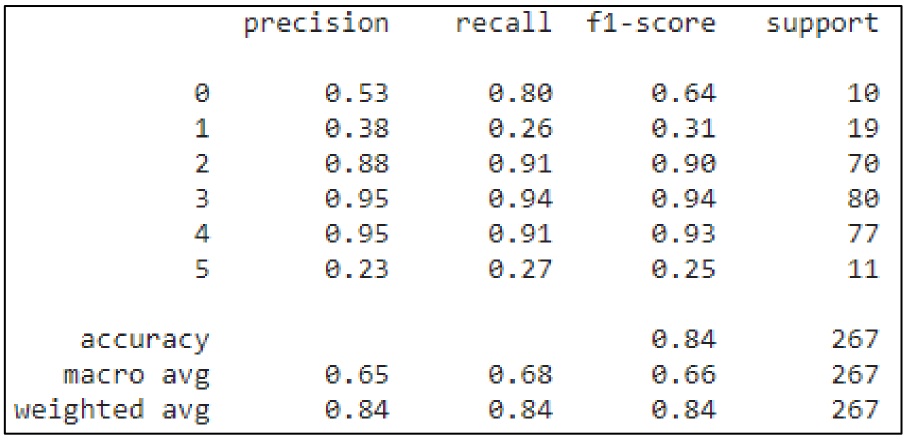
The respective **confusion matrix** is:



```
[[ 8  1  0  0  0  1]
 [ 6  5  3  0  0  5]
 [ 0  2 64  1  0  3]
 [ 0  0  1 75  4  0]
 [ 0  2  2  2 70  1]
 [ 1  3  3  1  0  3]]
```

Figure 16.Confusion matrix-extracted_abstract_new-CNN

And the **classification report** is:



```
              precision    recall  f1-score   support

           0       0.53      0.80      0.64        10
           1       0.38      0.26      0.31        19
           2       0.88      0.91      0.90        70
           3       0.95      0.94      0.94        80
           4       0.95      0.91      0.93        77
           5       0.23      0.27      0.25        11

    accuracy                           0.84       267
   macro avg       0.65      0.68      0.66       267
weighted avg       0.84      0.84      0.84       267
```

Figure 17.Clasiffication report-extracted_abstract_new-CNN

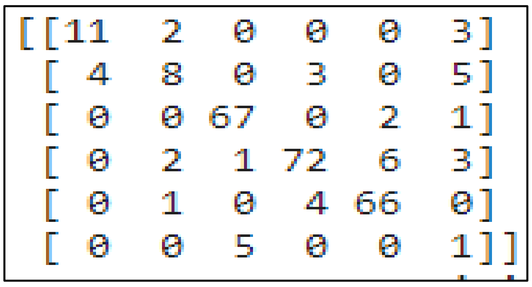
We observe similar trends and patterns in the confusion matrix and in the classification report regarding SDGs 3, 5 and 13 and SDGs 7, 10 and 12, as we have already commented.

The **accuracy** of the model is **84%**.

RNN results

After applying the RNN model, the results are displayed in the below graphs.

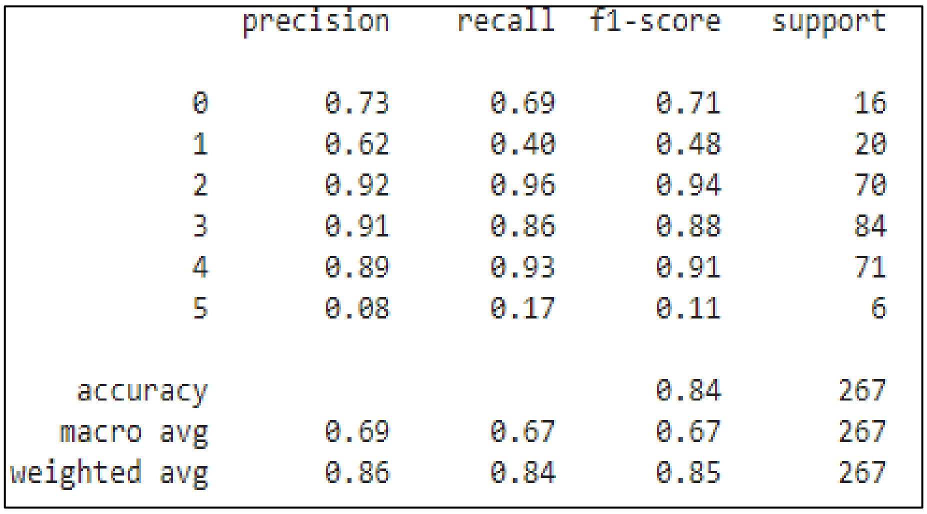
The **confusion matrix** is:



```
[[11  2  0  0  0  3]
 [ 4  8  0  3  0  5]
 [ 0  0 67  0  2  1]
 [ 0  2  1 72  6  3]
 [ 0  1  0  4 66  0]
 [ 0  0  5  0  0  1]]
```

Figure 18.Confusion matrix-extracted_abstract_new-RNN

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.73      0.69      0.71        16
           1       0.62      0.40      0.48        20
           2       0.92      0.96      0.94        70
           3       0.91      0.86      0.88        84
           4       0.89      0.93      0.91        71
           5       0.08      0.17      0.11         6

    accuracy                           0.84       267
   macro avg       0.69      0.67      0.67       267
weighted avg       0.86      0.84      0.85       267
```

Figure 19.Clasiffication report-extracted_abstract_new-RNN

We observe that class 5 (SDG 7) has f1-score equal to 0.11, while the minimum value f1-score can take is 0.00.

The **accuracy** of the model is **84%**.

To continue with, we will apply the same models, but now for the normalized columns; these columns are the ones that apart from the "basic" cleaning, we have also applied normalization techniques, like stemming.

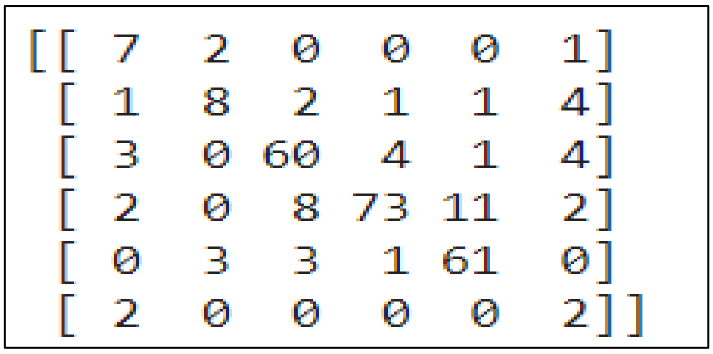
"normalized_extracted_title" variable

MLP results

When applying our MLP model to the "normalized_extracted_title" variable we get an error. Most probably, this is because our model cannot be executed for such simple data, on which have also been applied many normalization methods.

CNN results

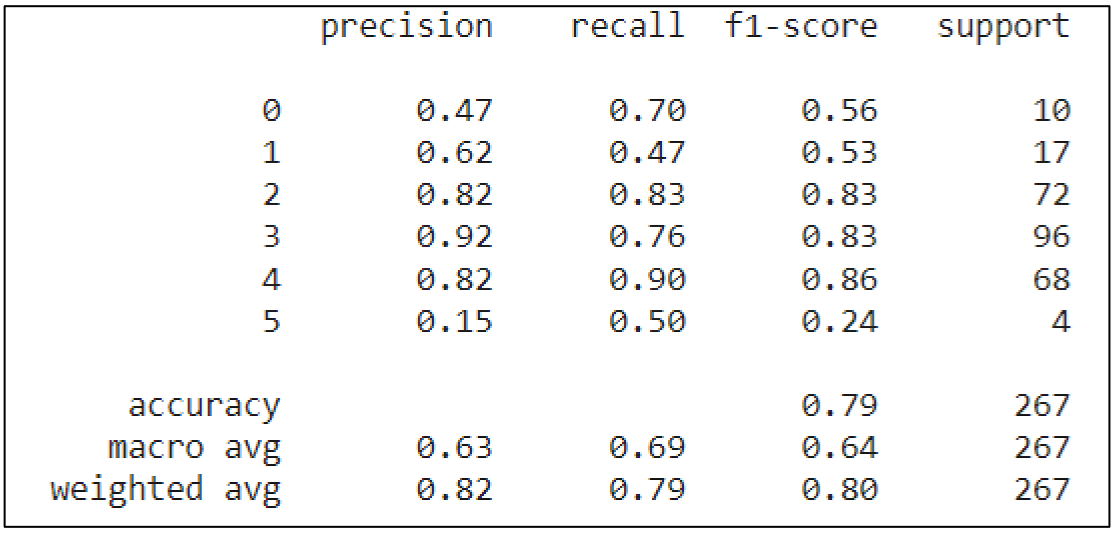
However, the CNN model runs properly and the **confusion matrix** is:



```
[[ 7  2  0  0  0  1]
 [ 1  8  2  1  1  4]
 [ 3  0 60  4  1  4]
 [ 2  0  8 73 11  2]
 [ 0  3  3  1 61  0]
 [ 2  0  0  0  0  2]]
```

Figure 20.Confusion matrix-normalized_extracted_title-CNN

The **classification report** is:



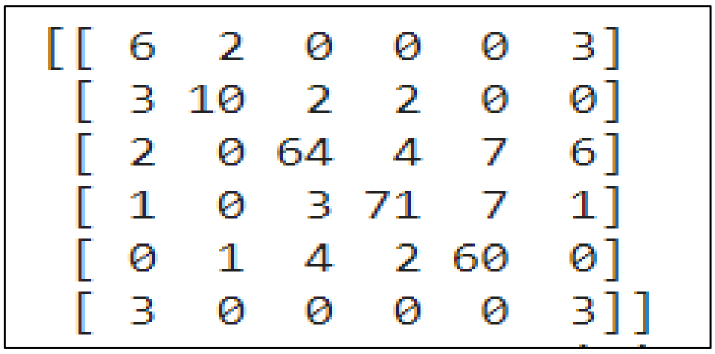
```
              precision    recall  f1-score   support

           0       0.47      0.70      0.56        10
           1       0.62      0.47      0.53        17
           2       0.82      0.83      0.83        72
           3       0.92      0.76      0.83        96
           4       0.82      0.90      0.86        68
           5       0.15      0.50      0.24         4

    accuracy                           0.79       267
   macro avg       0.63      0.69      0.64       267
weighted avg       0.82      0.79      0.80       267
```

Figure 21.Clasiffication Report--normalized_extracted_title-CNN

The **accuracy** of the model is **79%**. This means that 79% of the observations were correctly classified.

RNN results

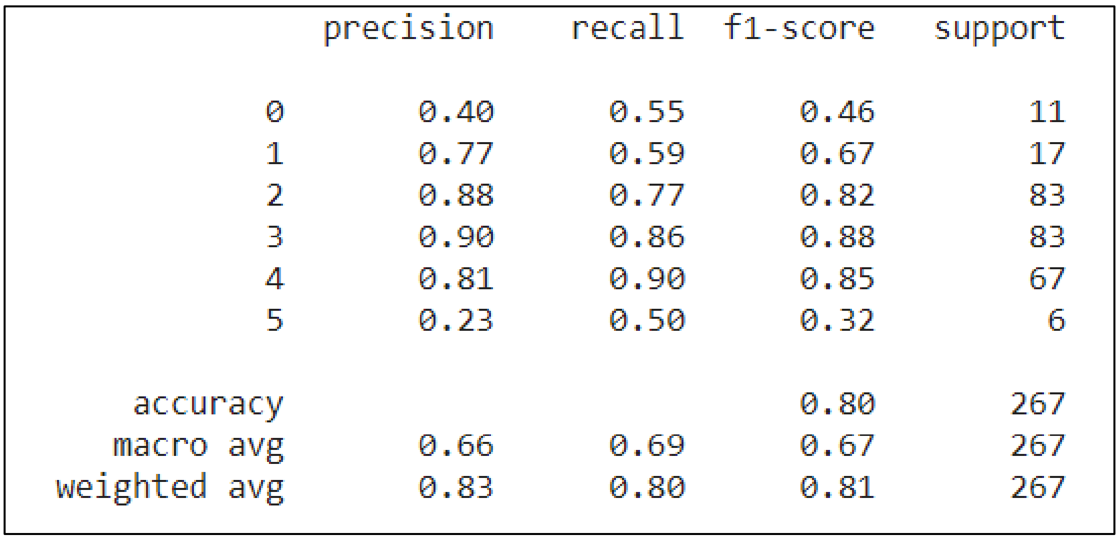
By applying the RNN model to this variable, we get the following **confusion matrix**:



```
[[ 6  2  0  0  0  3]
 [ 3 10  2  2  0  0]
 [ 2  0 64  4  7  6]
 [ 1  0  3 71  7  1]
 [ 0  1  4  2 60  0]
 [ 3  0  0  0  0  3]]
```

Figure 22.Confusion matrix-normalized_extracted_title-RNN

The respective **classification report** is:



```
              precision    recall  f1-score   support

           0       0.40      0.55      0.46        11
           1       0.77      0.59      0.67        17
           2       0.88      0.77      0.82        83
           3       0.90      0.86      0.88        83
           4       0.81      0.90      0.85        67
           5       0.23      0.50      0.32         6

    accuracy                           0.80       267
   macro avg       0.66      0.69      0.67       267
weighted avg       0.83      0.80      0.81       267
```

Figure 23.Clasiffication Report--normalized_extracted_title-RNN

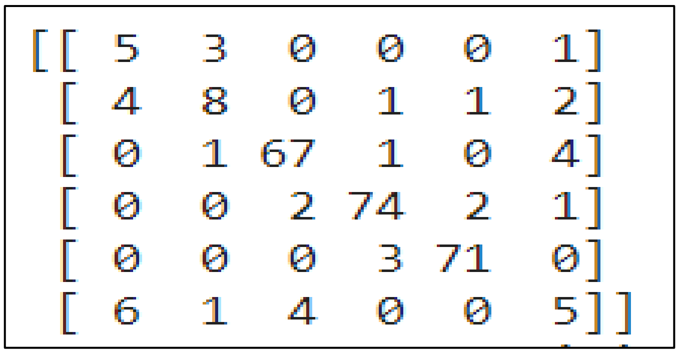
The f1-scores for SDGs 3, 5 and 13 are the highest and are lower than 0.90.

The **accuracy** of the model is **80%**.

"normalized_extracted_abstract" variable

MLP results

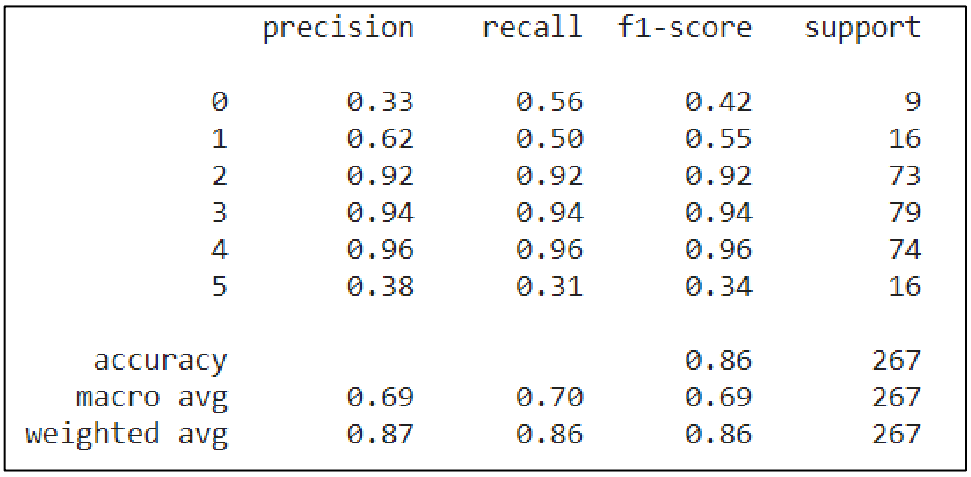
The **confusion matrix** of the MLP model with independent variable the "normalized_extracted_abstract" is displayed below:



```
[[ 5  3  0  0  0  1]
 [ 4  8  0  1  1  2]
 [ 0  1 67  1  0  4]
 [ 0  0  2 74  2  1]
 [ 0  0  0  3 71  0]
 [ 6  1  4  0  0  5]]
```

Figure 24.Confusion matrix-normalized_extracted_abstract-MLP

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.33      0.56      0.42         9
           1       0.62      0.50      0.55        16
           2       0.92      0.92      0.92        73
           3       0.94      0.94      0.94        79
           4       0.96      0.96      0.96        74
           5       0.38      0.31      0.34        16

    accuracy                           0.86       267
   macro avg       0.69      0.70      0.69       267
weighted avg       0.87      0.86      0.86       267
```

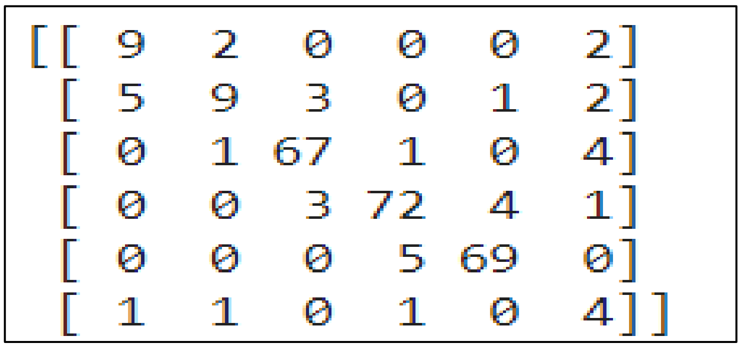
Figure 25.Clasiffication Report-normalized_extracted_abstract-MLP

F1-score of class 4 (SDG 5) is 0.96, while the possible maximum value of f1-score is 1.00.

The **accuracy** of the model is **86%**.

CNN results

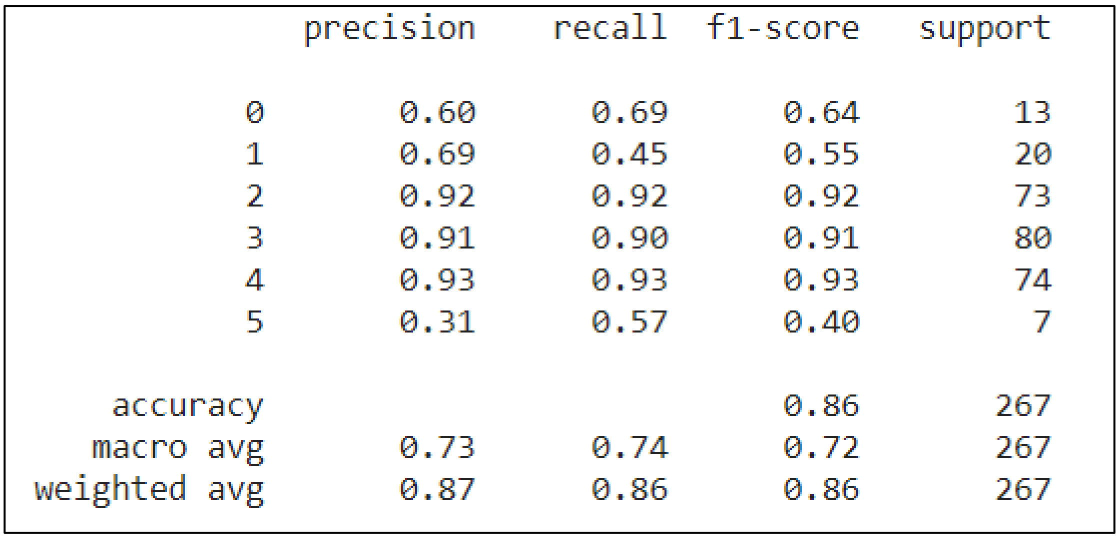
The **confusion matrix** is:



```
[[ 9  2  0  0  0  2]
 [ 5  9  3  0  1  2]
 [ 0  1 67  1  0  4]
 [ 0  0  3 72  4  1]
 [ 0  0  0  5 69  0]
 [ 1  1  0  1  0  4]]
```

Figure 26.Confusion matrix-normalized_extracted_abstract-CNN

The **classification report** is:



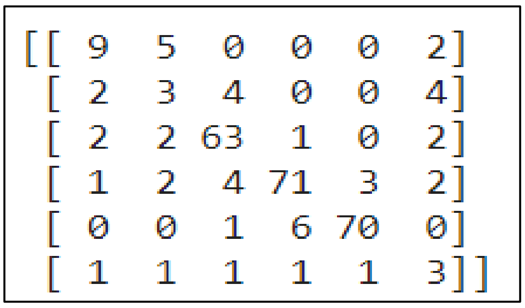
```
              precision    recall  f1-score   support

           0       0.60      0.69      0.64        13
           1       0.69      0.45      0.55        20
           2       0.92      0.92      0.92        73
           3       0.91      0.90      0.91        80
           4       0.93      0.93      0.93        74
           5       0.31      0.57      0.40         7

    accuracy                           0.86       267
   macro avg       0.73      0.74      0.72       267
weighted avg       0.87      0.86      0.86       267
```

Figure 27.Clasiffication Report-normalized_extracted_abstract-CNN

The **accuracy** of the model is **86%**.

RNN results

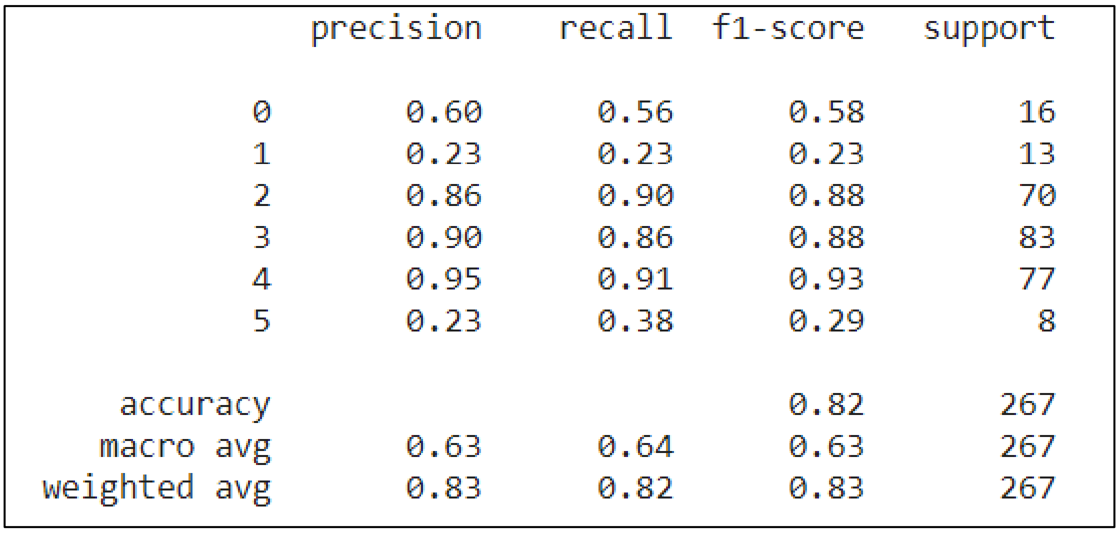
The **confusion matrix** is:



```
[[ 9  5  0  0  0  2]
 [ 2  3  4  0  0  4]
 [ 2  2 63  1  0  2]
 [ 1  2  4 71  3  2]
 [ 0  0  1  6 70  0]
 [ 1  1  1  1  1  3]]
```

Figure 28.Confusion matrix-normalized_extracted_abstract-RNN

And the **classification report** is:



```
              precision    recall  f1-score   support

           0       0.60      0.56      0.58        16
           1       0.23      0.23      0.23        13
           2       0.86      0.90      0.88        70
           3       0.90      0.86      0.88        83
           4       0.95      0.91      0.93        77
           5       0.23      0.38      0.29         8

    accuracy                           0.82       267
   macro avg       0.63      0.64      0.63       267
weighted avg       0.83      0.82      0.83       267
```

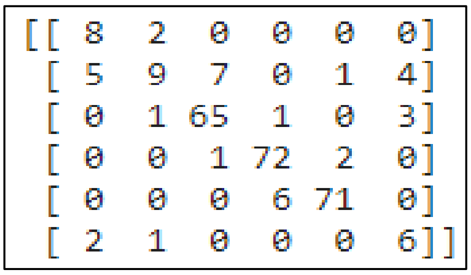
Figure 29.Clasiffication Report-normalized_extracted_abstract-RNN

The **accuracy** of the model is **82%**.

“normalized_initial_text” variable

MLP results

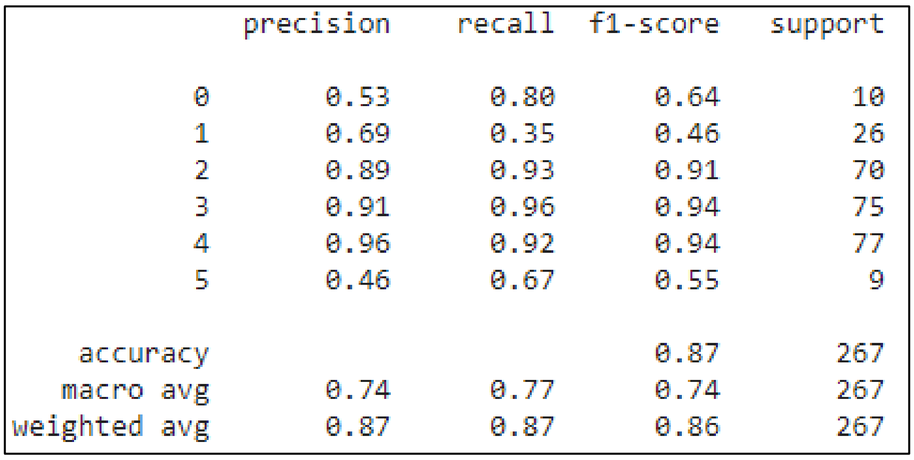
The **confusion matrix** is:



```
[[ 8  2  0  0  0  0]
 [ 5  9  7  0  1  4]
 [ 0  1 65  1  0  3]
 [ 0  0  1 72  2  0]
 [ 0  0  0  6 71  0]
 [ 2  1  0  0  0  6]]
```

Figure 30.Confusion matrix-normalized_initial_text_MLP

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.53      0.80      0.64        10
           1       0.69      0.35      0.46        26
           2       0.89      0.93      0.91        70
           3       0.91      0.96      0.94        75
           4       0.96      0.92      0.94        77
           5       0.46      0.67      0.55         9

    accuracy                           0.87       267
   macro avg       0.74      0.77      0.74       267
weighted avg       0.87      0.87      0.86       267
```

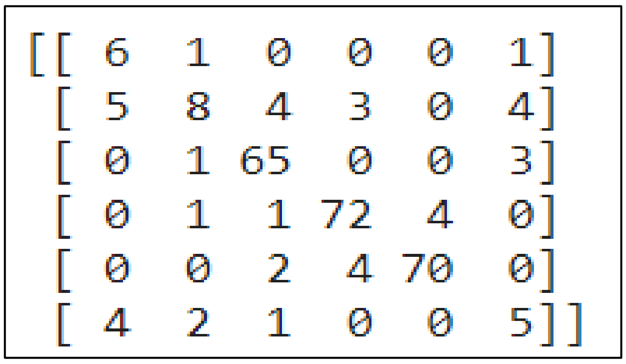
Figure 31.Classification report-normalized_initial_text_MLP

Once again, for some classes the f1-score is almost 1.00.

The **accuracy** of the model is **87%**.

CNN results

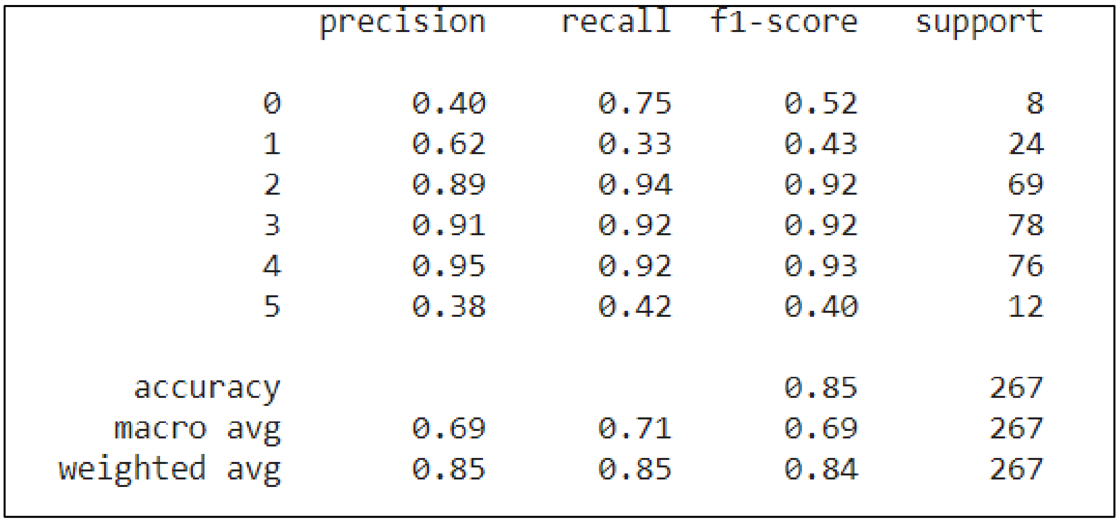
The **confusion matrix** is:



```
[[ 6  1  0  0  0  1]
 [ 5  8  4  3  0  4]
 [ 0  1 65  0  0  3]
 [ 0  1  1 72  4  0]
 [ 0  0  2  4 70  0]
 [ 4  2  1  0  0  5]]
```

Figure 32.Confusion matrix-normalized_initial_text_CNN

The **classification report** is:



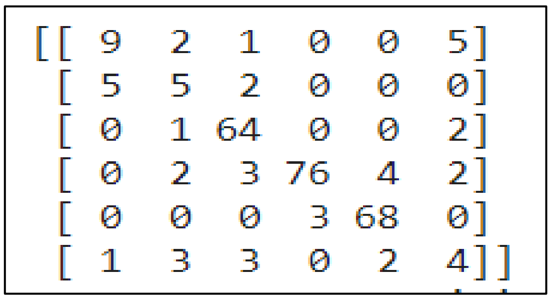
```
              precision    recall  f1-score   support

           0       0.40      0.75      0.52         8
           1       0.62      0.33      0.43        24
           2       0.89      0.94      0.92        69
           3       0.91      0.92      0.92        78
           4       0.95      0.92      0.93        76
           5       0.38      0.42      0.40        12

    accuracy                           0.85       267
   macro avg       0.69      0.71      0.69       267
weighted avg       0.85      0.85      0.84       267
```

Figure 33.Classification report-normalized_initial_text_CNN

We conclude that the **accuracy** of the model is **85%**.

RNN results

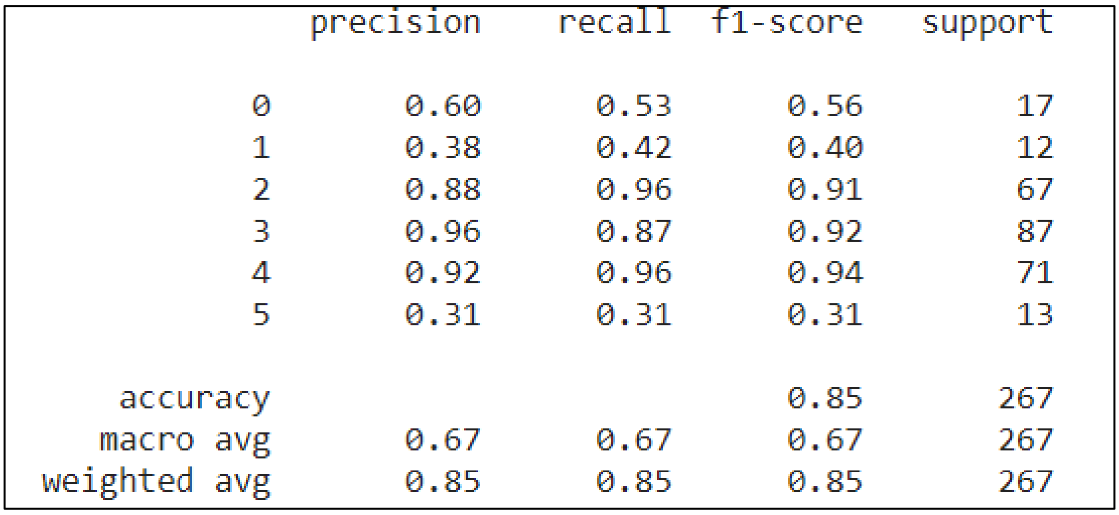
The **confusion matrix** is:



```
[[ 9  2  1  0  0  5]
 [ 5  5  2  0  0  0]
 [ 0  1 64  0  0  2]
 [ 0  2  3 76  4  2]
 [ 0  0  0  3 68  0]
 [ 1  3  3  0  2  4]]
```

Figure 34.Confusion matrix-normalized_initial_text_RNN

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.60      0.53      0.56        17
           1       0.38      0.42      0.40        12
           2       0.88      0.96      0.91        67
           3       0.96      0.87      0.92        87
           4       0.92      0.96      0.94        71
           5       0.31      0.31      0.31        13

    accuracy                           0.85       267
   macro avg       0.67      0.67      0.67       267
weighted avg       0.85      0.85      0.85       267
```

Figure 35.Classification report-normalized_initial_text_RNN

The **accuracy** of the model is **85%**.

PART I - Models with ONE Variation

Conclusion - Summary

After applying MLP, CNN and RNN models to the nine available variables (“extracted_title_new”, “extracted_abstract_new”, “initial_text_new”, “normalized_extracted_title”, “normalized_extracted_abstract” and “normalized_initial_text”), we have 18 different models. The metric that we monitor is “accuracy”.

Finally, we enclose all the results in the below matrix:

RESULTS SUMMARY	**Accuracy per Model**		
	Accuracy		
Variable's Name	**MLP**	**CNN**	**RNN**
extracted_title_new	84%	87%	84%
extracted_abstract_new	**88%**	84%	84%
initial_text_new	85%	84%	87%
normalized_extracted_title	Error	79%	80%
normalized_extracted_abstract	86%	86%	82%
normalized_initial_text	87%	85%	85%

Table 3.Accuracy per Model with 1 variation

From Table 3 we can conclude that the model with the best accuracy is MLP with “extracted_abstract_new” as the independent variable.

PART II - Models with TWO Variations

We continue by applying classification using two independent variables.

Firstly, we will import the “statistics” library.

Now, we will not use a sequential model, as we have two inputs, but instead the functional API of Keras. Basically, it is a different way to write the model. More specifically, the sequential API allows us to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs. The functional API in Keras is an alternate way of creating models that offers a lot more flexibility, including creating more complex models. The main idea is that a deep learning model is usually a directed acyclic graph (DAG) of layers. So, the functional API is a way to build graphs of layers[15].

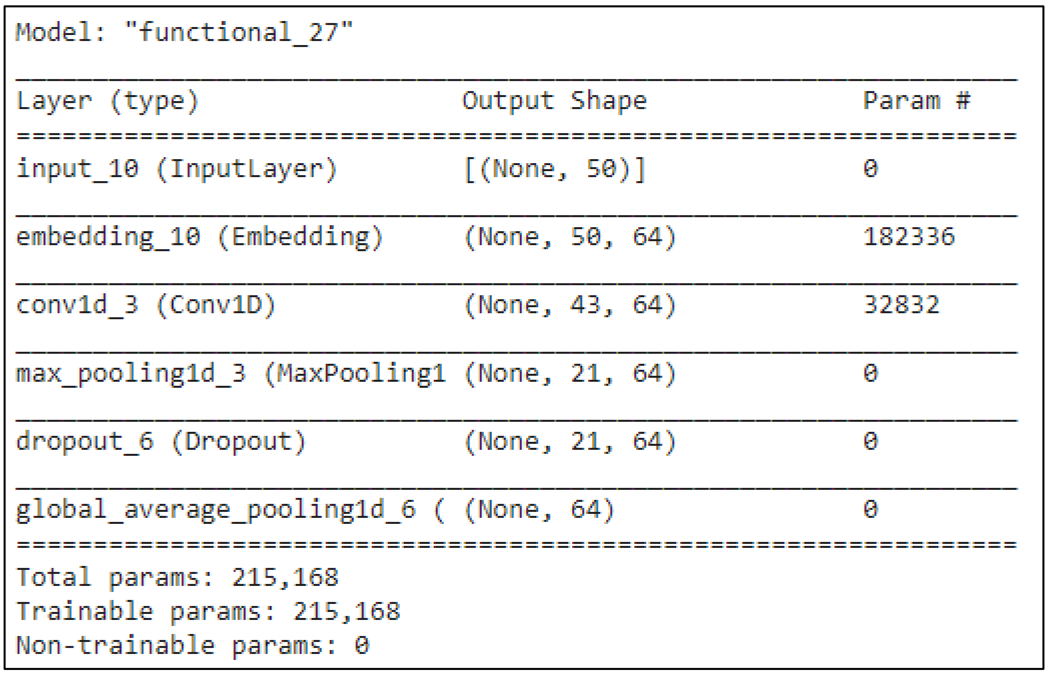
Therefore, we start by creating a function and placing an input layer for each variable that we are going to use. In this case, we want the title as the first input and the abstract as the second. Our next step is to create two tokenizers, because we have two dictionaries. As in Part I, we will explore the vocabulary of our variables. We move on by making one embedding layer, which will take as input the abstract, and one that will take the title. Each embedding is added to a Dense, add dropout, apply flattening, and a line is drawn that separates the inputs from the outputs. We will, also, do padding, for each one of our variables, as described in Part I. As a result, the respective architecture is created. We will concatenate the final outputs of each architecture and finally use the classifier of our interest. Then, we will be able to define our model and extract its summary. We will continue by applying, checking and examining different architectures. At the end, we will do “fine tuning” the model that we will conclude to.

We will apply different combinations of models to our variables; ex CNN and LSTM and then use concatenate to extract the mixed model.

“extracted_abstract_new” and “extracted_title_new” variables

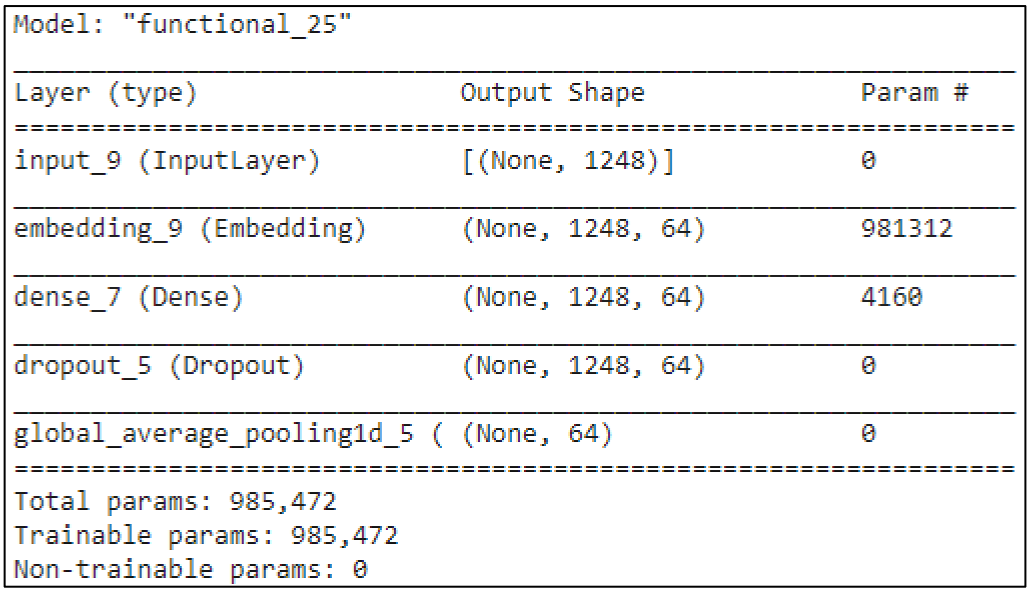
1st Architecture: CNN (“extracted_title_new”) – MLP (“extracted_abstract_new”)

We will now apply the techniques/methods that we described above. The first combination is the CNN model for the title and the MLP for the abstract.



```
Model: "functional_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 50)]              0
_________________________________________________________________
embedding_10 (Embedding)     (None, 50, 64)            182336
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 43, 64)            32832
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, 21, 64)            0
_________________________________________________________________
dropout_6 (Dropout)          (None, 21, 64)            0
_________________________________________________________________
global_average_pooling1d_6 ( (None, 64)                0
=================================================================
Total params: 215,168
Trainable params: 215,168
Non-trainable params: 0
```

Table 4. title_cnn.summary()



```
Model: "functional_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 1248)]            0
_________________________________________________________________
embedding_9 (Embedding)      (None, 1248, 64)          981312
_________________________________________________________________
dense_7 (Dense)              (None, 1248, 64)          4160
_________________________________________________________________
dropout_5 (Dropout)          (None, 1248, 64)          0
_________________________________________________________________
global_average_pooling1d_5 ( (None, 64)                0
=================================================================
Total params: 985,472
Trainable params: 985,472
Non-trainable params: 0
```

Table 5. text_mlp.summary()

Let’s see in a plot below what we described above and how our model with two dimensions is structured:

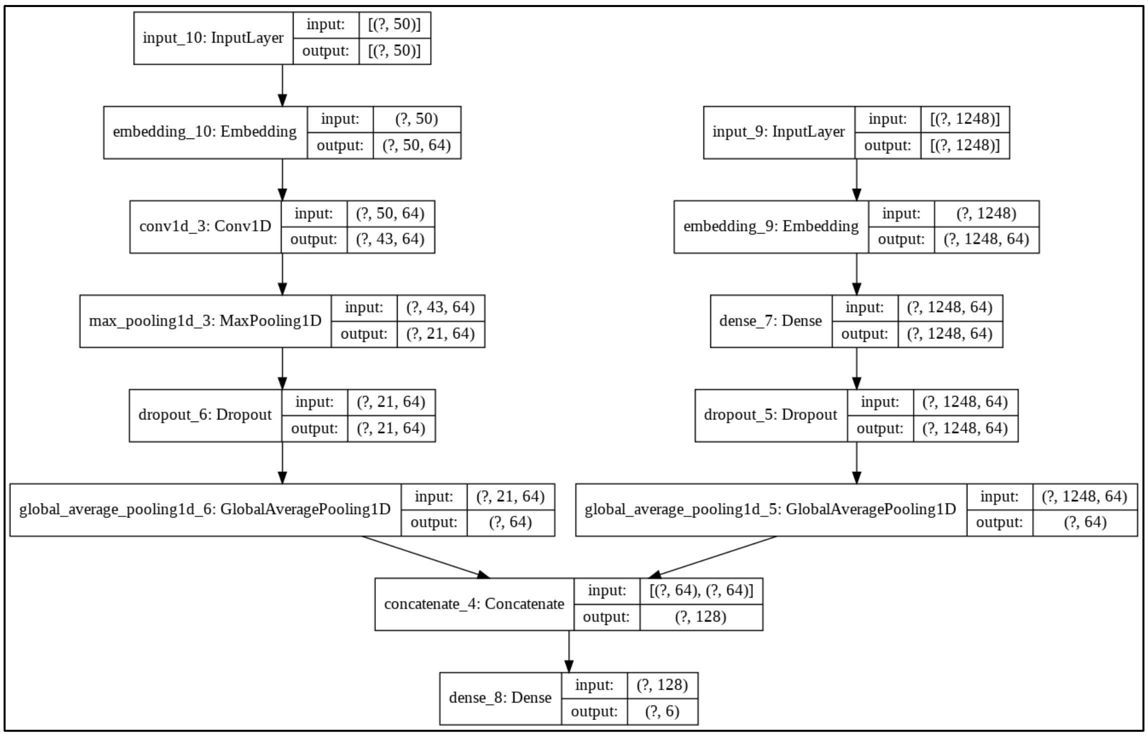
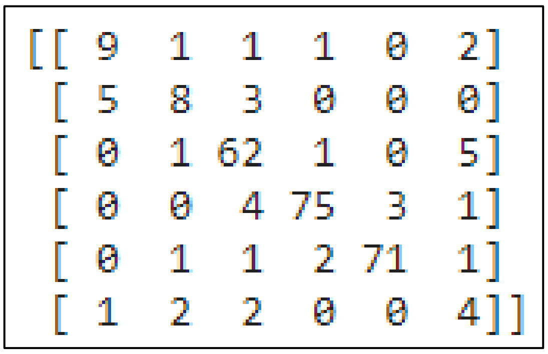

input_10: InputLayer | input: [(?, 50)] | output: [(?, 50)]
embedding_10: Embedding | input: (?, 50) | output: (?, 50, 64)
conv1d_3: Conv1D | input: (?, 50, 64) | output: (?, 43, 64)
max_pooling1d_3: MaxPooling1D | input: (?, 43, 64) | output: (?, 21, 64)
dropout_6: Dropout | input: (?, 21, 64) | output: (?, 21, 64)
global_average_pooling1d_6: GlobalAveragePooling1D | input: (?, 21, 64) | output: (?, 64)
input_9: InputLayer | input: [(?, 1248)] | output: [(?, 1248)]
embedding_9: Embedding | input: (?, 1248) | output: (?, 1248, 64)
dense_7: Dense | input: (?, 1248, 64) | output: (?, 1248, 64)
dropout_5: Dropout | input: (?, 1248, 64) | output: (?, 1248, 64)
global_average_pooling1d_5: GlobalAveragePooling1D | input: (?, 1248, 64) | output: (?, 64)
concatenate_4: Concatenate | input: [(?, 64), (?, 64)] | output: (?, 128)
dense_8: Dense | input: (?, 128) | output: (?, 6)

Figure 36. extracted_abstract_new & extracted_title_new-CNN & MPL plot_model()

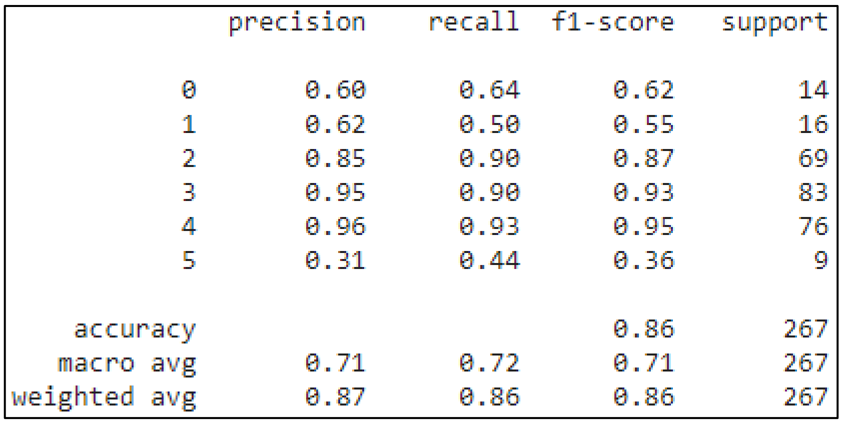
The **confusion matrix** is:



```
[[ 9  1  1  1  0  2]
 [ 5  8  3  0  0  0]
 [ 0  1 62  1  0  5]
 [ 0  0  4 75  3  1]
 [ 0  1  1  2 71  1]
 [ 1  2  2  0  0  4]]
```

Figure 37.Confusion matrix-extracted_abstract_new & extracted_title_new-CNN & MLP

The **classification report** is:



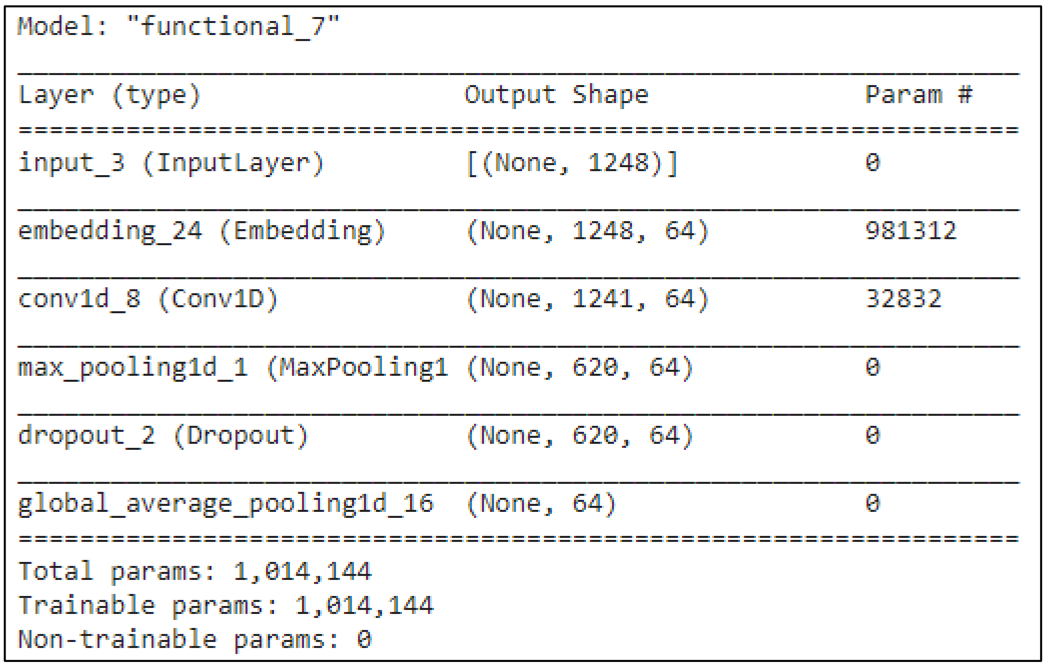
```
              precision    recall  f1-score   support

           0       0.60      0.64      0.62        14
           1       0.62      0.50      0.55        16
           2       0.85      0.90      0.87        69
           3       0.95      0.90      0.93        83
           4       0.96      0.93      0.95        76
           5       0.31      0.44      0.36         9

    accuracy                           0.86       267
   macro avg       0.71      0.72      0.71       267
weighted avg       0.87      0.86      0.86       267
```

Figure 38.Classification report-extracted_abstract_new & extracted_title_new-CNN & MLP

The **accuracy** of the model is **86%**.

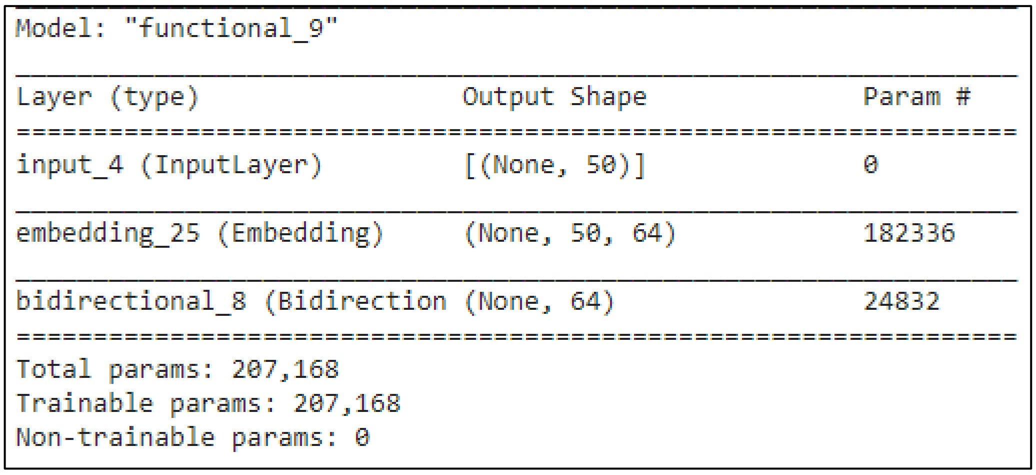
2nd **Architecture:** CNN (“extracted_abstract_new”) – LSTM (“extracted_title_new”)

Similarly, we apply the techniques/methods that we described above. The second combination is the CNN model for the text and the RNN for the title.



```
Model: "functional_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 1248)]            0
_________________________________________________________________
embedding_24 (Embedding)     (None, 1248, 64)          981312
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 1241, 64)          32832
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 620, 64)           0
_________________________________________________________________
dropout_2 (Dropout)          (None, 620, 64)           0
_________________________________________________________________
global_average_pooling1d_16  (None, 64)                0
=================================================================
Total params: 1,014,144
Trainable params: 1,014,144
Non-trainable params: 0
```

Table 6.text_cnn.summary()



```
Model: "functional_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, 50)]              0
_________________________________________________________________
embedding_25 (Embedding)     (None, 50, 64)            182336
_________________________________________________________________
bidirectional_8 (Bidirection (None, 64)                24832
=================================================================
Total params: 207,168
Trainable params: 207,168
Non-trainable params: 0
```

Table 7.title_rnn.summary()

Let’s see in a plot below what we described above and how our model with two dimensions is structured:

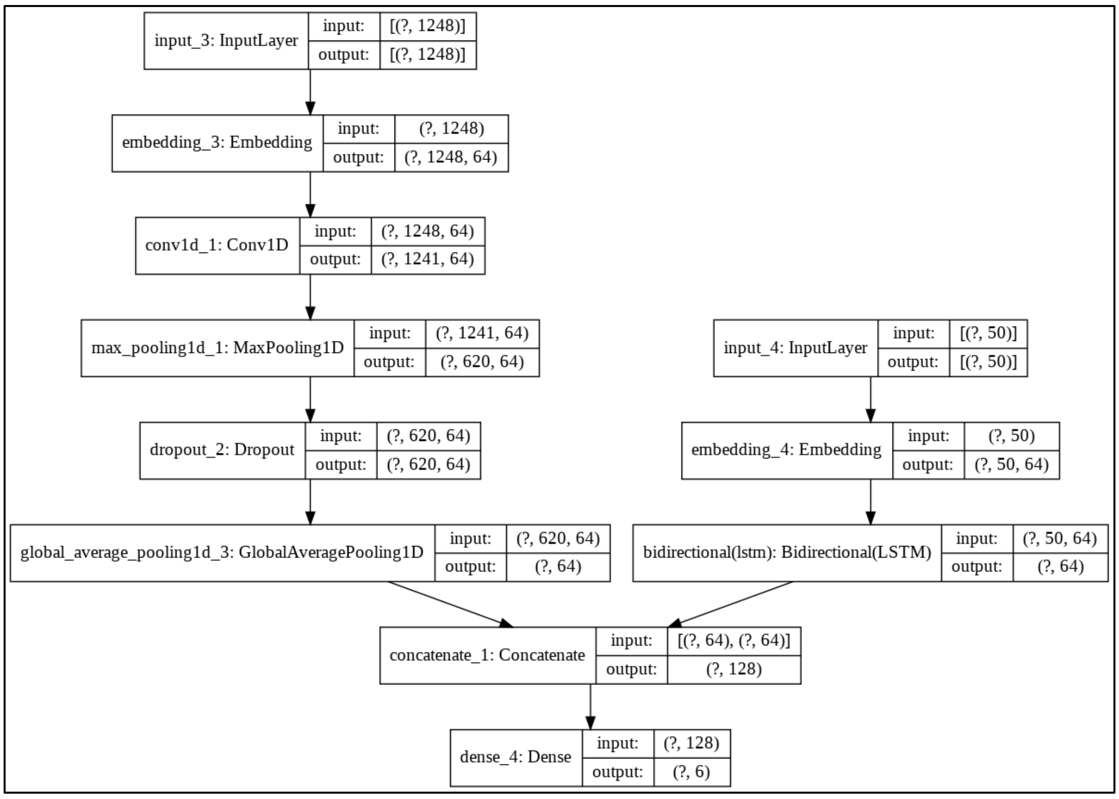
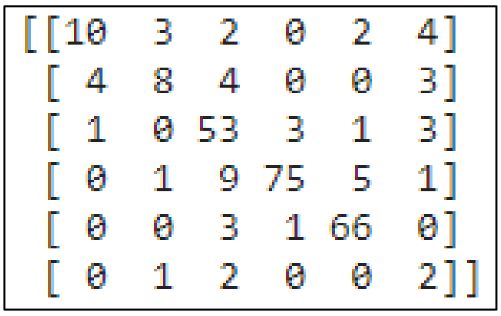


Figure 39.extracted_abstract_new & extracted_title_new-CNN & LSTM plot_model()

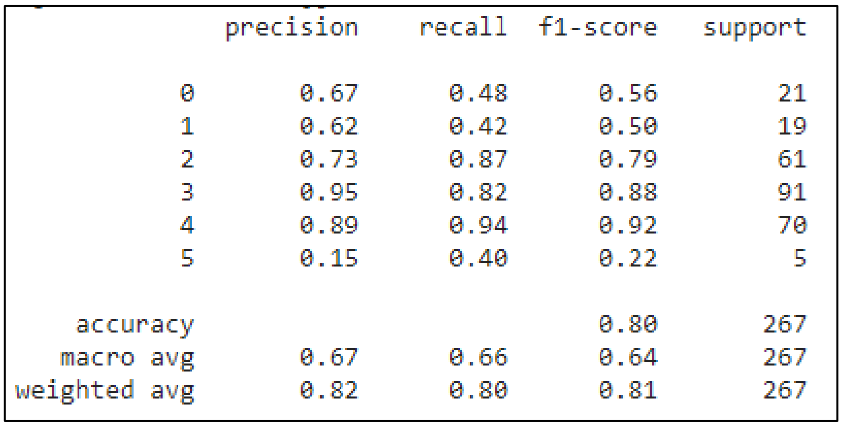
The **confusion matrix** is:



```
[[10  3  2  0  2  4]
 [ 4  8  4  0  0  3]
 [ 1  0 53  3  1  3]
 [ 0  1  9 75  5  1]
 [ 0  0  3  1 66  0]
 [ 0  1  2  0  0  2]]
```

Figure 40.Confusion matrix-extracted_abstract_new & extracted_title_new-CNN & LSTM

The **classification report** is:



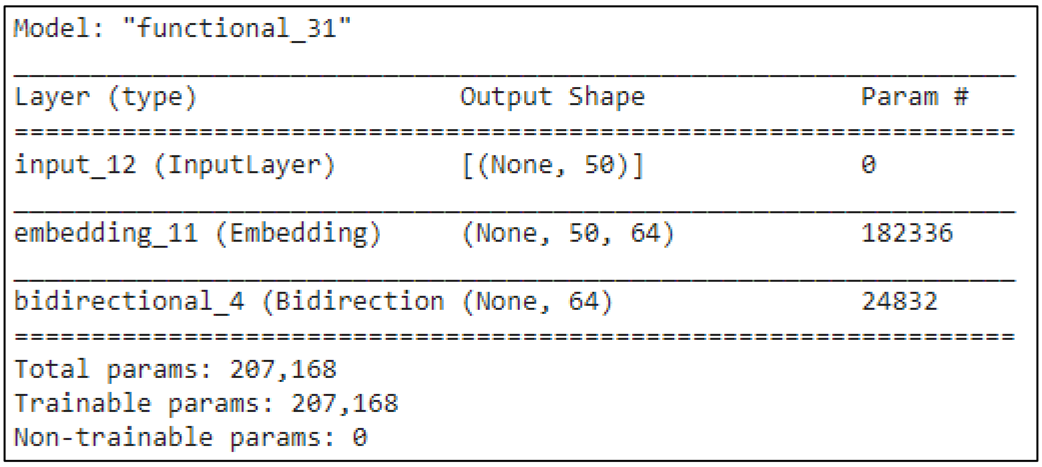
```
              precision    recall  f1-score   support

           0       0.67      0.48      0.56        21
           1       0.62      0.42      0.50        19
           2       0.73      0.87      0.79        61
           3       0.95      0.82      0.88        91
           4       0.89      0.94      0.92        70
           5       0.15      0.40      0.22         5

    accuracy                           0.80       267
   macro avg       0.67      0.66      0.64       267
weighted avg       0.82      0.80      0.81       267
```

Figure 41 Classification report-extracted_abstract_new & extracted_title_new-CNN & LSTM

The **accuracy** of the model is **80%**.

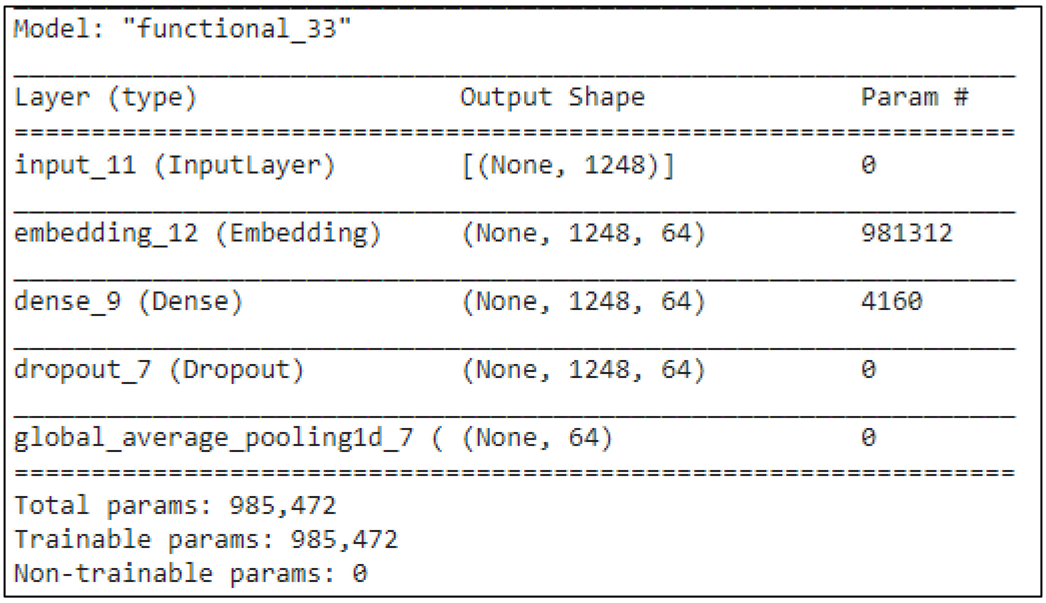
3rd Architecture: RNN (“extracted_title_new”) – MLP (“extracted_abstract_new”)

Similarly, we apply the techniques/methods that we described above. The third combination is the RNN model for the title and the MLP for the text.



```
Model: "functional_31"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_12 (InputLayer)        [(None, 50)]              0         
_________________________________________________________________
embedding_11 (Embedding)     (None, 50, 64)            182336    
_________________________________________________________________
bidirectional_4 (Bidirection (None, 64)                24832     
=================================================================
Total params: 207,168
Trainable params: 207,168
Non-trainable params: 0
```

Table 8. title_rnn.summary()



```
Model: "functional_33"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_11 (InputLayer)        [(None, 1248)]            0         
_________________________________________________________________
embedding_12 (Embedding)     (None, 1248, 64)          981312    
_________________________________________________________________
dense_9 (Dense)              (None, 1248, 64)          4160      
_________________________________________________________________
dropout_7 (Dropout)          (None, 1248, 64)          0         
_________________________________________________________________
global_average_pooling1d_7 ( (None, 64)                0         
=================================================================
Total params: 985,472
Trainable params: 985,472
Non-trainable params: 0
```

Table 9. text_mlp.summary()

Let’s see in a plot below what we described above and how our model with two dimensions is structured:

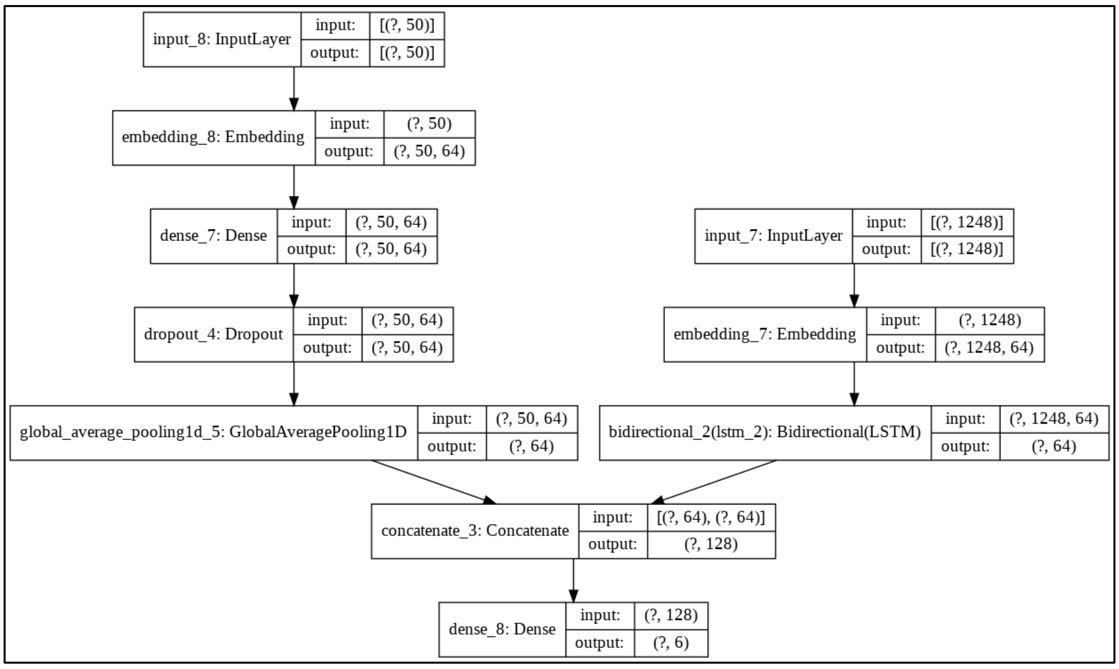


Figure42.extracted_abstract_new & extracted_title_new-RNN & MLP plot_model()

The **confusion matrix** is:

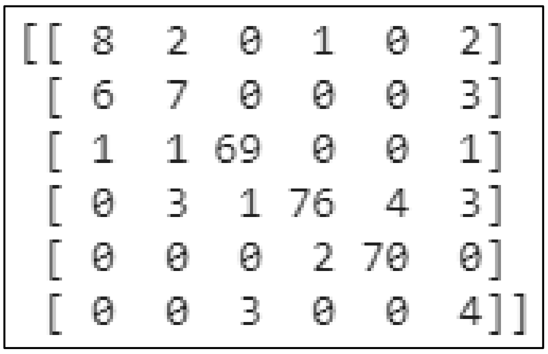

```
[[ 8  2  0  1  0  2]
 [ 6  7  0  0  0  3]
 [ 1  1 69  0  0  1]
 [ 0  3  1 76  4  3]
 [ 0  0  0  2 70  0]
 [ 0  0  3  0  0  4]]
```

Figure 43.Confusion matrix-extracted_abstract_new & extracted_title_new-RNN & MLP

The **classification report** is:

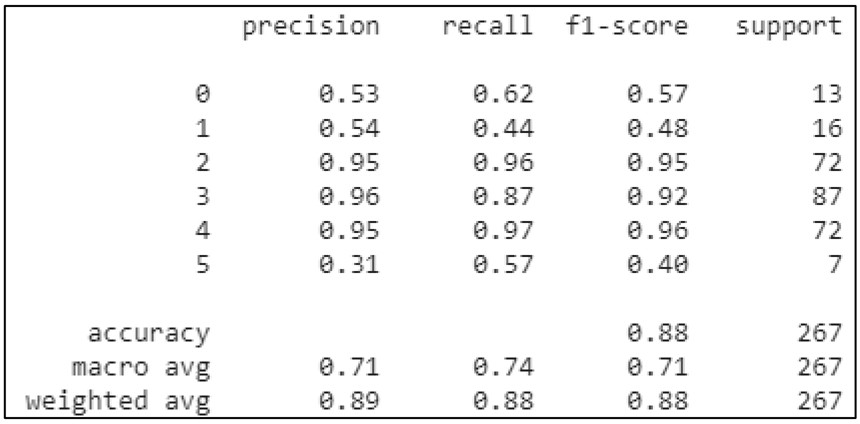
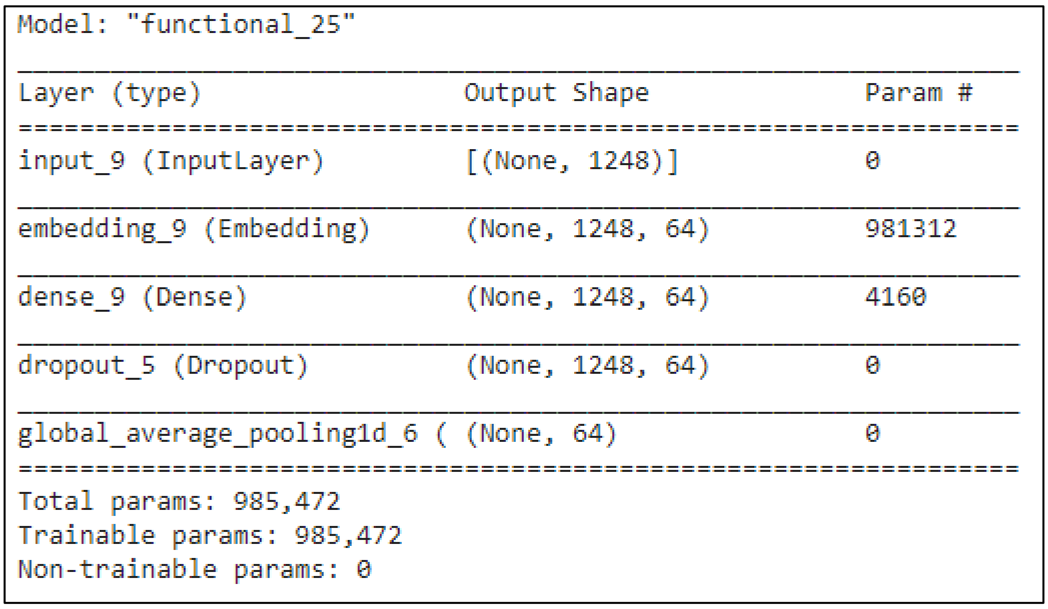

```
              precision    recall  f1-score   support

           0       0.53      0.62      0.57        13
           1       0.54      0.44      0.48        16
           2       0.95      0.96      0.95        72
           3       0.96      0.87      0.92        87
           4       0.95      0.97      0.96        72
           5       0.31      0.57      0.40         7

    accuracy                           0.88       267
   macro avg       0.71      0.74      0.71       267
weighted avg       0.89      0.88      0.88       267
```

Figure 424.Classification report-extracted_abstract_new & extracted_title_new-RNN & MLP

The **accuracy** of the model is **87%**.

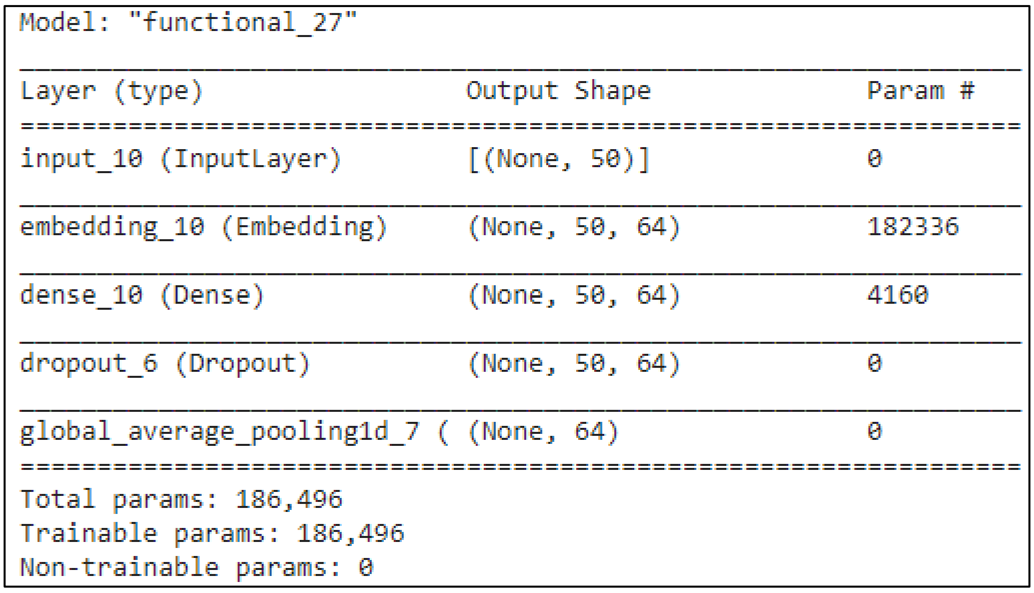
4th Architecture: MLP (“extracted_abstract_new”) – MLP (“extracted_title_new”)

Lastly, we apply the techniques/methods that we described above. The fourth combination is the MLP model for both the text and the title.



```
Model: "functional_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_9 (InputLayer)         [(None, 1248)]            0
_________________________________________________________________
embedding_9 (Embedding)      (None, 1248, 64)          981312
_________________________________________________________________
dense_9 (Dense)              (None, 1248, 64)          4160
_________________________________________________________________
dropout_5 (Dropout)          (None, 1248, 64)          0
_________________________________________________________________
global_average_pooling1d_6 ( (None, 64)                0
=================================================================
Total params: 985,472
Trainable params: 985,472
Non-trainable params: 0
```

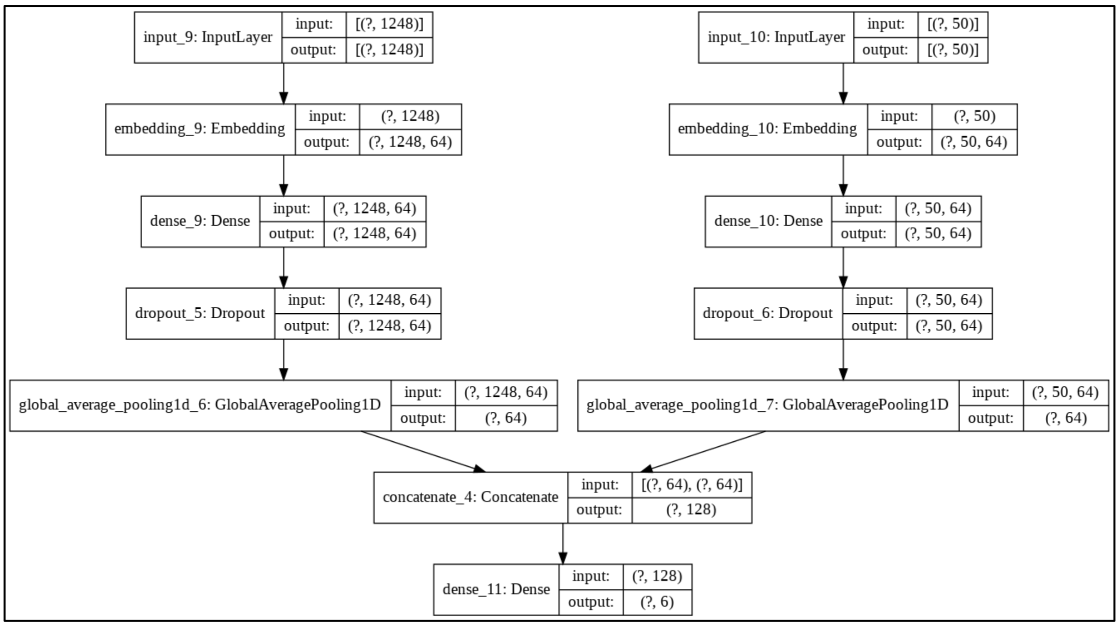
Table 10.text_mlp.summary()



```
Model: "functional_27"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_10 (InputLayer)        [(None, 50)]              0
_________________________________________________________________
embedding_10 (Embedding)     (None, 50, 64)            182336
_________________________________________________________________
dense_10 (Dense)             (None, 50, 64)            4160
_________________________________________________________________
dropout_6 (Dropout)          (None, 50, 64)            0
_________________________________________________________________
global_average_pooling1d_7 ( (None, 64)                0
=================================================================
Total params: 186,496
Trainable params: 186,496
Non-trainable params: 0
```

Table 11.title_mlp.summary()

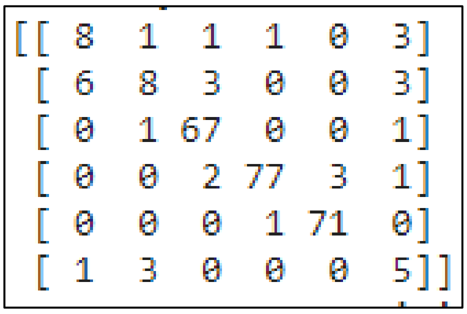
Let’s see in a plot below what we described above and how our model with two dimensions is structured:



input_9: InputLayer | input: [(?, 1248)] | output: [(?, 1248)]
embedding_9: Embedding | input: (?, 1248) | output: (?, 1248, 64)
dense_9: Dense | input: (?, 1248, 64) | output: (?, 1248, 64)
dropout_5: Dropout | input: (?, 1248, 64) | output: (?, 1248, 64)
global_average_pooling1d_6: GlobalAveragePooling1D | input: (?, 1248, 64) | output: (?, 64)
input_10: InputLayer | input: [(?, 50)] | output: [(?, 50)]
embedding_10: Embedding | input: (?, 50) | output: (?, 50, 64)
dense_10: Dense | input: (?, 50, 64) | output: (?, 50, 64)
dropout_6: Dropout | input: (?, 50, 64) | output: (?, 50, 64)
global_average_pooling1d_7: GlobalAveragePooling1D | input: (?, 50, 64) | output: (?, 64)
concatenate_4: Concatenate | input: [(?, 64), (?, 64)] | output: (?, 128)
dense_11: Dense | input: (?, 128) | output: (?, 6)

Figure45.extracted_abstract_new & extracted_title_new-MLP & MLP plot_model()

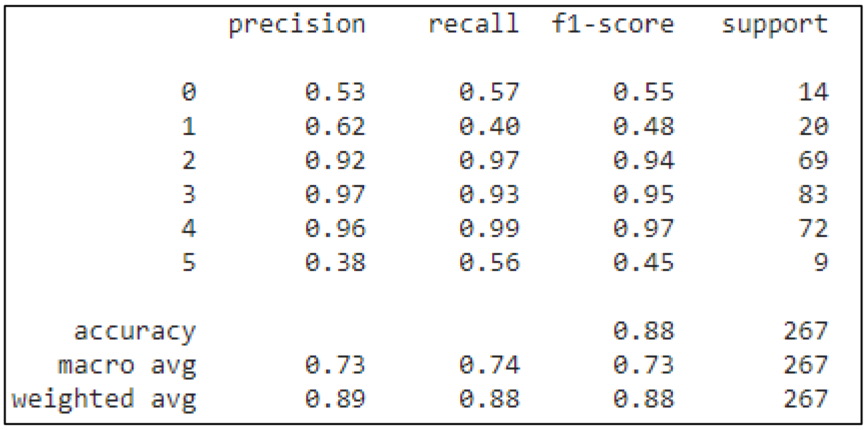
The **confusion matrix** is:



```
[[ 8  1  1  1  0  3]
 [ 6  8  3  0  0  3]
 [ 0  1 67  0  0  1]
 [ 0  0  2 77  3  1]
 [ 0  0  0  1 71  0]
 [ 1  3  0  0  0  5]]
```

Figure 46.Confusion matrix-extracted_abstract_new & extracted_title_new-MLP & MLP

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.53      0.57      0.55        14
           1       0.62      0.40      0.48        20
           2       0.92      0.97      0.94        69
           3       0.97      0.93      0.95        83
           4       0.96      0.99      0.97        72
           5       0.38      0.56      0.45         9

    accuracy                           0.88       267
   macro avg       0.73      0.74      0.73       267
weighted avg       0.89      0.88      0.88       267
```

Figure 437.Classification report-extracted_abstract_new & extracted_title_new-MLP & MLP

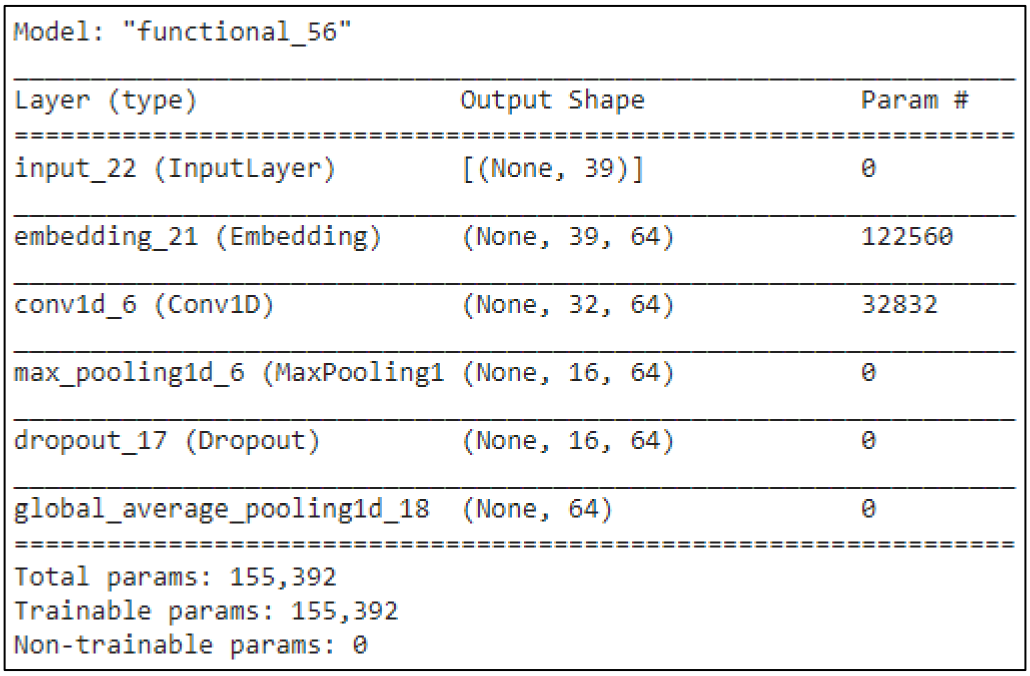
The **accuracy** of the model is **88%**.

“normalized_extracted_abstract” and “normalized_extracted_title” variables

1st Architecture: CNN (“normalized_extracted_title”) – MLP (“normalized_extracted_abstract”)

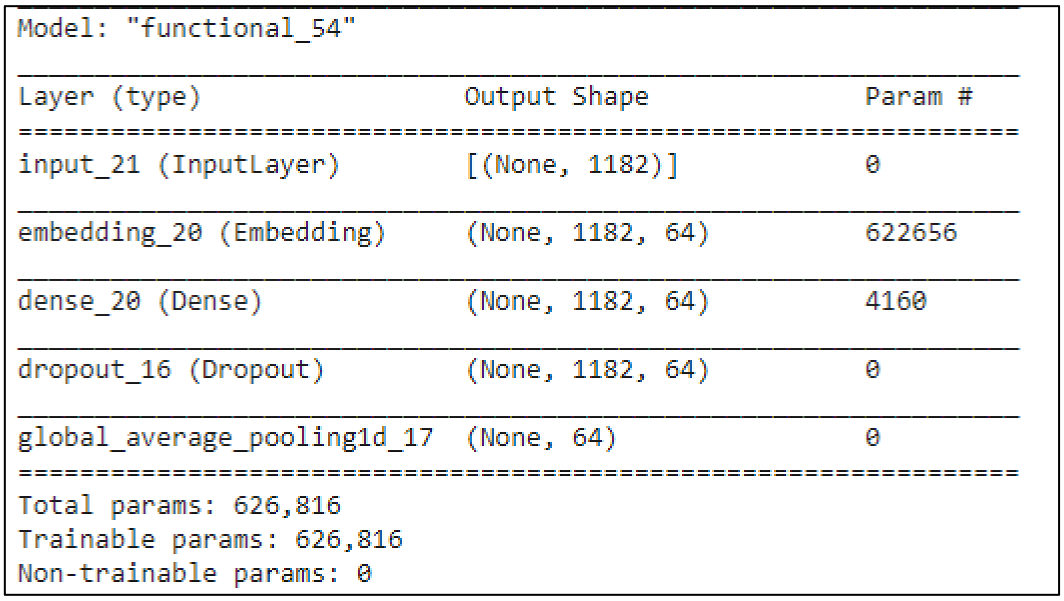
Same as before, when we tried to apply MPL to the “normalized_extracted_title” variable, we get an error. Most probably, this is because our model cannot be executed for such simple data, on which many normalization methods have been applied.

Therefore, we changed the combination of the model to CNN for the title and the MLP for the text.



```
Model: "functional_56"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_22 (InputLayer)        [(None, 39)]              0
_________________________________________________________________
embedding_21 (Embedding)     (None, 39, 64)            122560
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 32, 64)            32832
_________________________________________________________________
max_pooling1d_6 (MaxPooling1 (None, 16, 64)            0
_________________________________________________________________
dropout_17 (Dropout)         (None, 16, 64)            0
_________________________________________________________________
global_average_pooling1d_18  (None, 64)                0
=================================================================
Total params: 155,392
Trainable params: 155,392
Non-trainable params: 0
```

Table 12.title_cnn.summary()



```
Model: "functional_54"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        [(None, 1182)]            0
_________________________________________________________________
embedding_20 (Embedding)     (None, 1182, 64)          622656
_________________________________________________________________
dense_20 (Dense)             (None, 1182, 64)          4160
_________________________________________________________________
dropout_16 (Dropout)         (None, 1182, 64)          0
_________________________________________________________________
global_average_pooling1d_17  (None, 64)                0
=================================================================
Total params: 626,816
Trainable params: 626,816
Non-trainable params: 0
```

Table 13.text_mlp.summary()

Let's see in a plot below what we described above and how our model with two dimensions is structured:

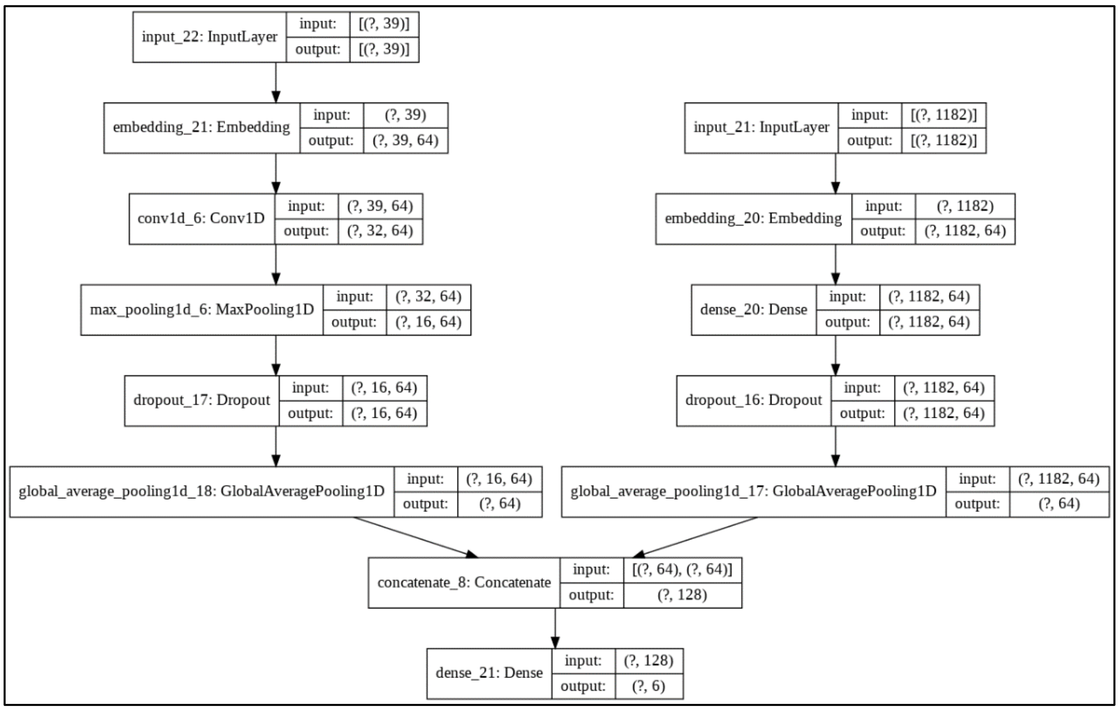
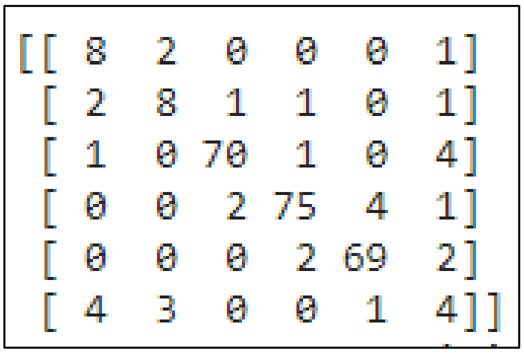

input_22: InputLayer | input: [(?, 39)] | output: [(?, 39)]
embedding_21: Embedding | input: (?, 39) | output: (?, 39, 64)
conv1d_6: Conv1D | input: (?, 39, 64) | output: (?, 32, 64)
max_pooling1d_6: MaxPooling1D | input: (?, 32, 64) | output: (?, 16, 64)
dropout_17: Dropout | input: (?, 16, 64) | output: (?, 16, 64)
global_average_pooling1d_18: GlobalAveragePooling1D | input: (?, 16, 64) | output: (?, 64)
input_21: InputLayer | input: [(?, 1182)] | output: [(?, 1182)]
embedding_20: Embedding | input: (?, 1182) | output: (?, 1182, 64)
dense_20: Dense | input: (?, 1182, 64) | output: (?, 1182, 64)
dropout_16: Dropout | input: (?, 1182, 64) | output: (?, 1182, 64)
global_average_pooling1d_17: GlobalAveragePooling1D | input: (?, 1182, 64) | output: (?, 64)
concatenate_8: Concatenate | input: [(?, 64), (?, 64)] | output: (?, 128)
dense_21: Dense | input: (?, 128) | output: (?, 6)

Figure 44.normalized_extracted_abstract & normalized_extracted_title-CNN & MLP plot_model()

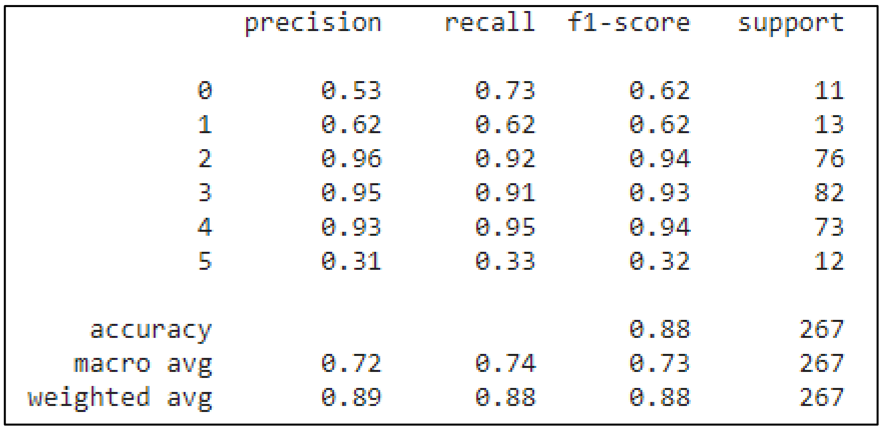
The **confusion matrix** is:



```
[[ 8  2  0  0  0  1]
 [ 2  8  1  1  0  1]
 [ 1  0 70  1  0  4]
 [ 0  0  2 75  4  1]
 [ 0  0  0  2 69  2]
 [ 4  3  0  0  1  4]]
```

Figure 45.Confusion matrix-extracted_abstract & normalized_extracted_title-CNN & MLP

The **classification report** is:



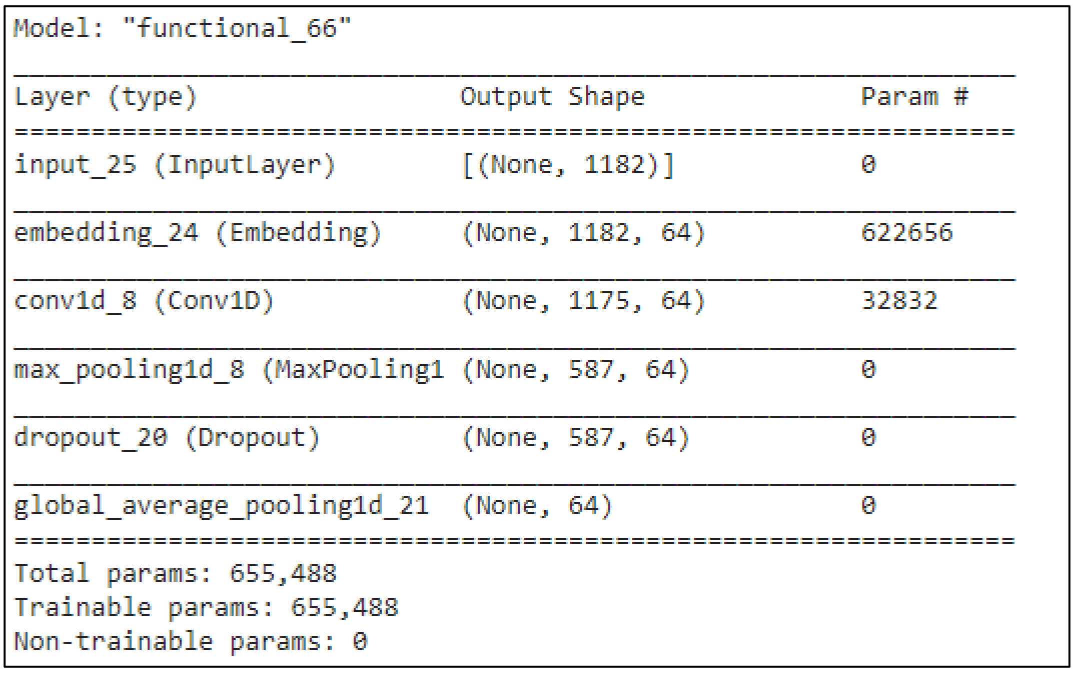
```
              precision    recall  f1-score   support

           0       0.53      0.73      0.62        11
           1       0.62      0.62      0.62        13
           2       0.96      0.92      0.94        76
           3       0.95      0.91      0.93        82
           4       0.93      0.95      0.94        73
           5       0.31      0.33      0.32        12

    accuracy                           0.88       267
   macro avg       0.72      0.74      0.73       267
weighted avg       0.89      0.88      0.88       267
```

Figure 46.Classification report-extracted_abstract & normalized_extracted_title-CNN & MLP

The **accuracy** of the model is **87%.**

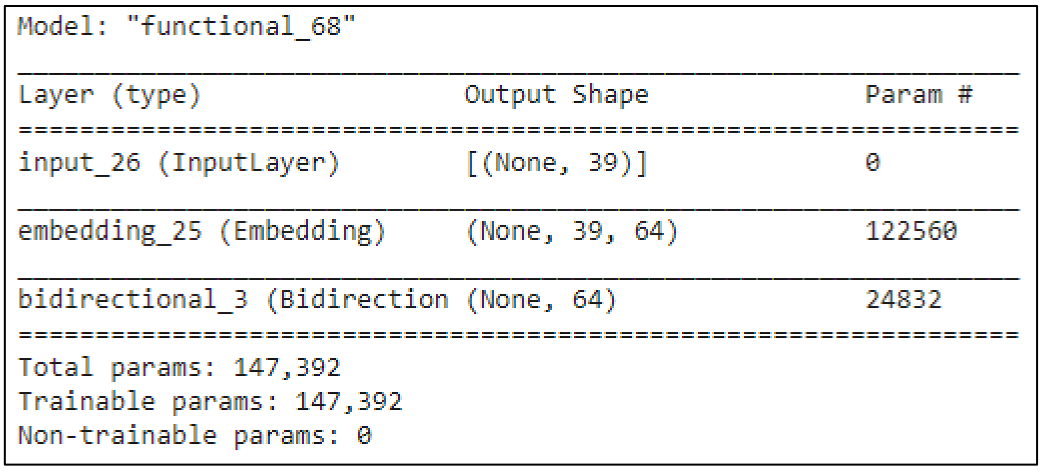
2nd **Architecture:** CNN ("normalized_extracted_abstract") – LSTM ("normalized_extracted_title")

Similarly, we apply the techniques/methods that we described above. The second combination is the CNN model for the text and the RNN for the title.



```
Model: "functional_66"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_25 (InputLayer)        [(None, 1182)]            0         
_________________________________________________________________
embedding_24 (Embedding)     (None, 1182, 64)          622656    
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 1175, 64)          32832     
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 587, 64)           0         
_________________________________________________________________
dropout_20 (Dropout)         (None, 587, 64)           0         
_________________________________________________________________
global_average_pooling1d_21  (None, 64)                0         
=================================================================
Total params: 655,488
Trainable params: 655,488
Non-trainable params: 0
```

Table 14.text_cnn.summary()



```
Model: "functional_68"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_26 (InputLayer)        [(None, 39)]              0
_________________________________________________________________
embedding_25 (Embedding)     (None, 39, 64)            122560
_________________________________________________________________
bidirectional_3 (Bidirection (None, 64)                24832
=================================================================
Total params: 147,392
Trainable params: 147,392
Non-trainable params: 0
```

Table 15.title_rnn.summary()

Let's see in a plot below what we described above and how our model with two dimensions is structured:

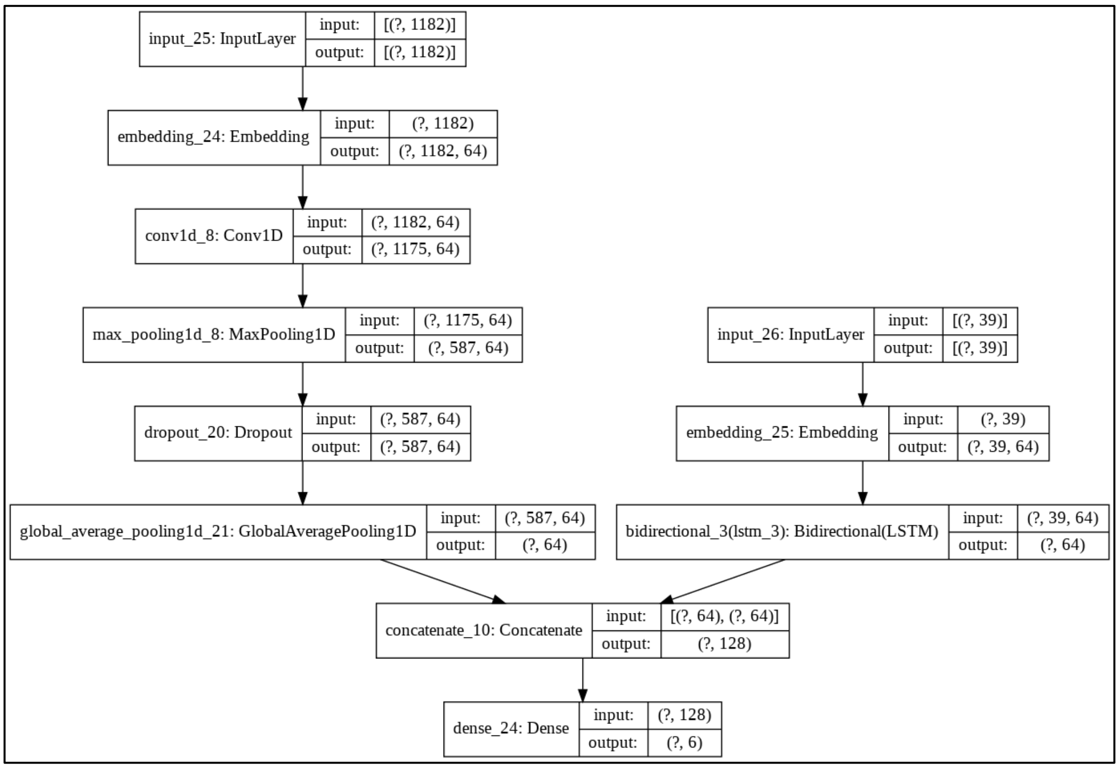
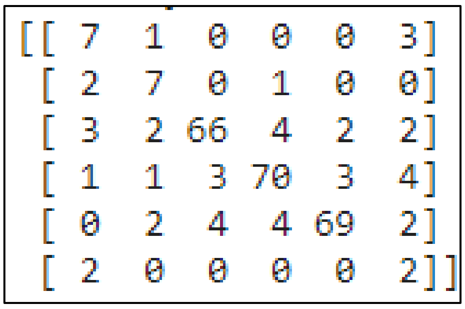

input_25: InputLayer | input: [(?, 1182)] | output: [(?, 1182)]
embedding_24: Embedding | input: (?, 1182) | output: (?, 1182, 64)
conv1d_8: Conv1D | input: (?, 1182, 64) | output: (?, 1175, 64)
max_pooling1d_8: MaxPooling1D | input: (?, 1175, 64) | output: (?, 587, 64)
dropout_20: Dropout | input: (?, 587, 64) | output: (?, 587, 64)
global_average_pooling1d_21: GlobalAveragePooling1D | input: (?, 587, 64) | output: (?, 64)
input_26: InputLayer | input: [(?, 39)] | output: [(?, 39)]
embedding_25: Embedding | input: (?, 39) | output: (?, 39, 64)
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (?, 39, 64) | output: (?, 64)
concatenate_10: Concatenate | input: [(?, 64), (?, 64)] | output: (?, 128)
dense_24: Dense | input: (?, 128) | output: (?, 6)

Figure 47.normalized_extracted_abstract & normalized_extracted_title-CNN & LSTM plot_model()

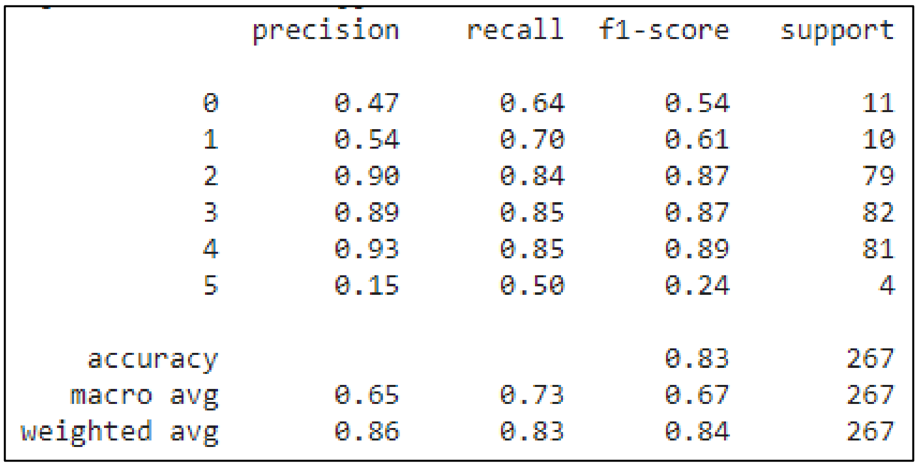
The **confusion matrix** is:



```
[[ 7  1  0  0  0  3]
 [ 2  7  0  1  0  0]
 [ 3  2 66  4  2  2]
 [ 1  1  3 70  3  4]
 [ 0  2  4  4 69  2]
 [ 2  0  0  0  0  2]]
```

Figure 48.Confusion matrix-extracted_abstract & normalized_extracted_title-CNN & LSTM

The **classification report** is:



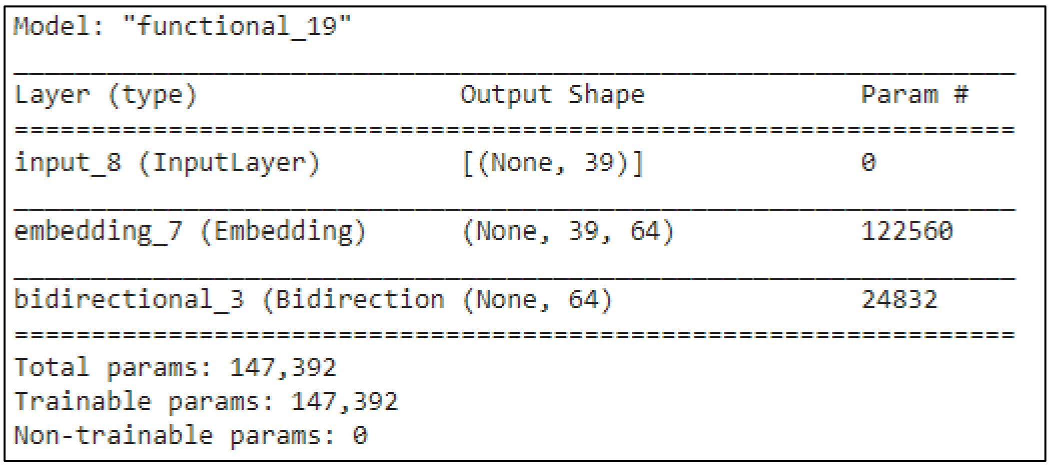
```
              precision    recall  f1-score   support

           0       0.47      0.64      0.54        11
           1       0.54      0.70      0.61        10
           2       0.90      0.84      0.87        79
           3       0.89      0.85      0.87        82
           4       0.93      0.85      0.89        81
           5       0.15      0.50      0.24         4

    accuracy                           0.83       267
   macro avg       0.65      0.73      0.67       267
weighted avg       0.86      0.83      0.84       267
```

Figure 49.Classification report-extracted_abstract & normalized_extracted_title-CNN & LSTM

The **accuracy** of the model is **83%.**

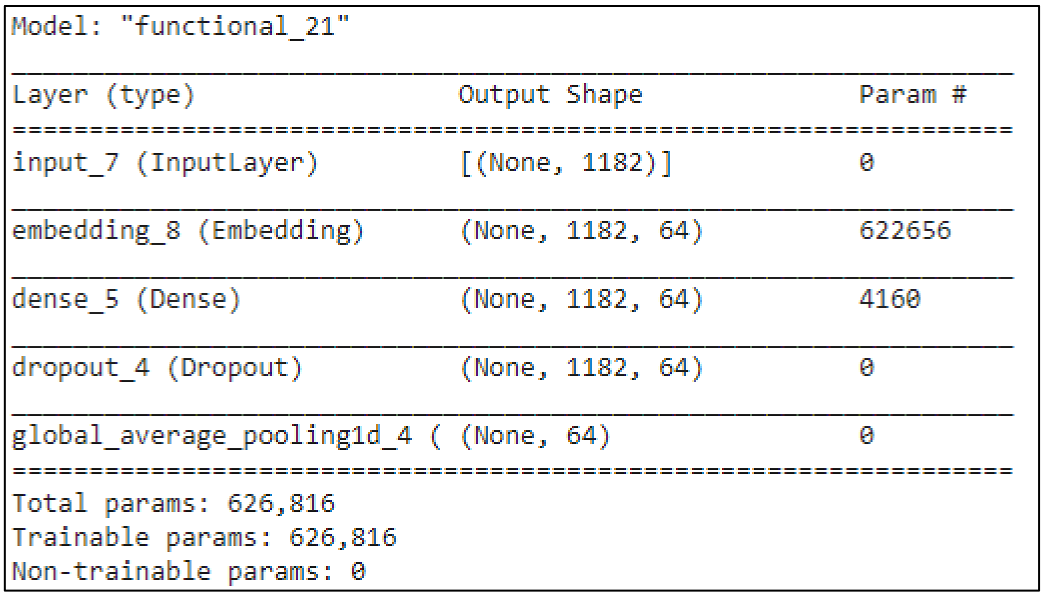
3rd Architecture: RNN (“normalized_extracted_title”) – MLP (“normalized_extracted_abstract”)

Similarly, we apply the techniques/methods that we described above. The third combination is the RNN model for the title and the MLP for the text.



```
Model: "functional_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_8 (InputLayer)         [(None, 39)]              0
_________________________________________________________________
embedding_7 (Embedding)      (None, 39, 64)            122560
_________________________________________________________________
bidirectional_3 (Bidirection (None, 64)                24832
=================================================================
Total params: 147,392
Trainable params: 147,392
Non-trainable params: 0
```

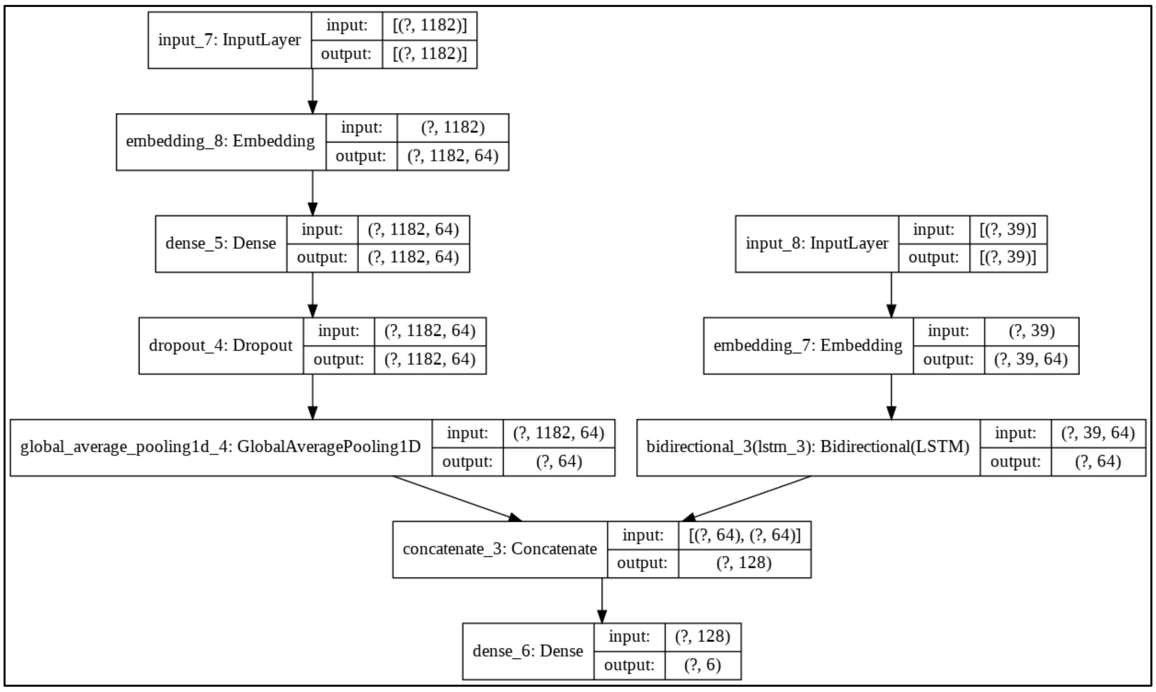
Table 166. title_rnn.summary()



```
Model: "functional_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 1182)]            0
_________________________________________________________________
embedding_8 (Embedding)      (None, 1182, 64)          622656
_________________________________________________________________
dense_5 (Dense)              (None, 1182, 64)          4160
_________________________________________________________________
dropout_4 (Dropout)          (None, 1182, 64)          0
_________________________________________________________________
global_average_pooling1d_4 ( (None, 64)                0
=================================================================
Total params: 626,816
Trainable params: 626,816
Non-trainable params: 0
```

Table 17. text_mlp.summary()

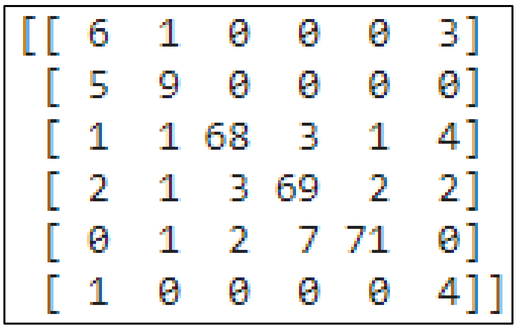
Let's see in a plot below what we described above and how our model with two dimensions is structured:



input_7: InputLayer | input: [(?, 1182)] | output: [(?, 1182)]
embedding_8: Embedding | input: (?, 1182) | output: (?, 1182, 64)
dense_5: Dense | input: (?, 1182, 64) | output: (?, 1182, 64)
dropout_4: Dropout | input: (?, 1182, 64) | output: (?, 1182, 64)
global_average_pooling1d_4: GlobalAveragePooling1D | input: (?, 1182, 64) | output: (?, 64)
input_8: InputLayer | input: [(?, 39)] | output: [(?, 39)]
embedding_7: Embedding | input: (?, 39) | output: (?, 39, 64)
bidirectional_3(lstm_3): Bidirectional(LSTM) | input: (?, 39, 64) | output: (?, 64)
concatenate_3: Concatenate | input: [(?, 64), (?, 64)] | output: (?, 128)
dense_6: Dense | input: (?, 128) | output: (?, 6)

Figure 50.normalized_extracted_abstract & normalized_extracted_title-RNN & MPL- plot_model()

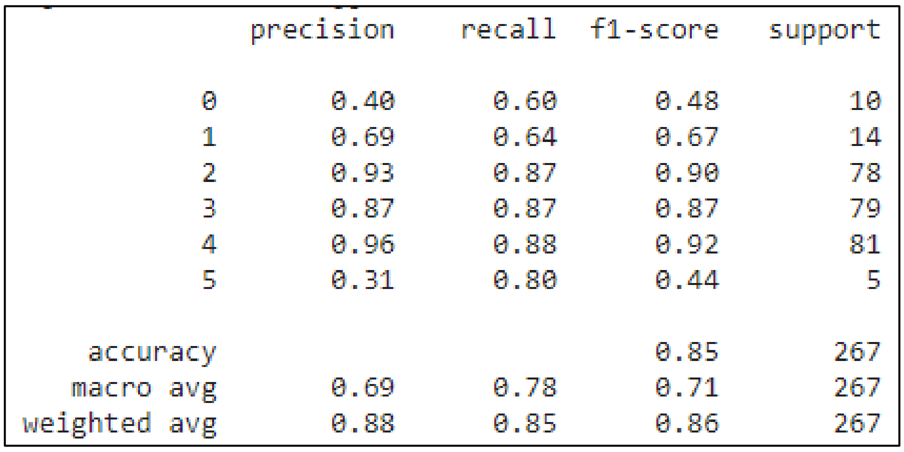
The **confusion matrix** is:



```
[[ 6  1  0  0  0  3]
 [ 5  9  0  0  0  0]
 [ 1  1 68  3  1  4]
 [ 2  1  3 69  2  2]
 [ 0  1  2  7 71  0]
 [ 1  0  0  0  0  4]]
```

Figure 51.Confusion matrix-normalized_extracted_abstract & normalized_extracted_title-RNN & MPL

The **classification report** is:



```
              precision    recall  f1-score   support

           0       0.40      0.60      0.48        10
           1       0.69      0.64      0.67        14
           2       0.93      0.87      0.90        78
           3       0.87      0.87      0.87        79
           4       0.96      0.88      0.92        81
           5       0.31      0.80      0.44         5

    accuracy                           0.85       267
   macro avg       0.69      0.78      0.71       267
weighted avg       0.88      0.85      0.86       267
```

Figure 52.Classification report-normalized_extracted_abstract & normalized_extracted_title- RNN & MPL

The **accuracy** of the model is **85%.**

4th Architecture: MLP ("normalized_extracted_abstract") – MLP ("normalized_extracted_title")

We will not run this model, as we know from the previous attempts that MLP is not an appropriate model to fit the data with X as normalized_extracted_title.

PART II - Models with TWO Variations

Conclusion - Summary

After applying CNN-MLP, CNN-LSTM, RNN-MLP and MLP-MLP mixed models to the two variable combinations ("extracted_title_new" & "extracted_abstract_new", "normalized_extracted_title" & "normalized_extracted_abstract), we have 8 different models. The metric that we monitor is "accuracy".

Finally, we enclose all the results in the below matrix:

RESULTS SUMMARY	**Accuracy per Model**			
	Accuracy			
Variables' names	**CNN-MLP**	**CNN-LSTM**	**RNN-MLP**	**MLP-MLP**
extracted_abstract_new & extracted_title_new	86%	80%	87%	**88%**
normalized_abstract_new & normalized_title_new	87%	83%	85	-

Table 18.Accuracy per Model with 2 variations

Fine tuning with the MLP-MLP mixed model architecture

From Table 18, we can conclude that the mixed model with the best accuracy is MLP-MLP with "extracted_title_new" & "extracted_abstract_new" as the independent variables.

Therefore, we proceed with fine tuning our model, which is structured like MLP-MLP model. The resulting accuracy is 90%.

PART III – Prediction on given test set

In this part of the assignment, we are given a blind test set which consists of 99 abstracts. Our task is to perform the final evaluation of our best model on this test set.

Our first step is to unzip and import the given data. Then, we observe some basic features regarding our blind test set.

Pre – Processing of Test Set

In this section, we will focus on the pre-processing of the test set. In fact, as we have already done in the original dataset, we will:

- Execute "word tokenization"
- Remove symbols/punctuations
- Remove English stopwords.

However, this is not enough, as observe that each text starts with a "b", which must be removed. In addition, like in the original dataset, we will apply padding as well.

Prediction

When performing classification with one variable models, we concluded that the best model is the MLP with "extracted_abstract_new" as the independent variable. Therefore, we will use its weight, that we named "Dense.hdf5" to predict the given test data set. Below is an example of the first five rows of the extracted csv. The whole csv file with the predictions is attached to the deliverables of this project.

Title	**Prediction**
ABC_G1B1_10.1016_j.energy.2018.11.091.txt	SDG12
ABC_G1B2_10.1016_j.apenergy.2018.01.084.txt	SDG10
ABC_G1B2_10.1016_j.enpol.2020.111284.txt	SDG10
ABC_G1B2_10.1016_j.jclepro.2020.121262.txt	SDG12
ABC_G1B2_10.1016_j.renene.2020.05.131.txt	SDG12

Table 189.Predictions Table

Sources

[1] https://www.mendeley.com/?interaction_required=true
[2] https://pubmed.ncbi.nlm.nih.gov/
[3] https://www.un.org/sustainabledevelopment/gender-equality/
[4] https://en.wikipedia.org/wiki/NumPy
[5] https://keras.io/about/)
[6] https://www.tensorflow.org/about
[7]https://keras.io/api/preprocessing/timeseries/
[7]https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/sequence/pad_sequences
[8] https://deepai.org/publication/effects-of-padding-on-lstms-and-cnns
[9] https://en.wikipedia.org/wiki/Multilayer_perceptron
[10]https://developer.nvidia.com/blog/deep-learning-nutshell-core-concepts/
[11]https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/
[12]https://en.wikipedia.org/wiki/Convolutional_neural_network, https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148)
[13] https://en.wikipedia.org/wiki/Recurrent_neural_network
[14] https://colah.github.io/posts/2015-08-Understanding-LSTMs/
[15] https://keras.io/guides/functional_api/