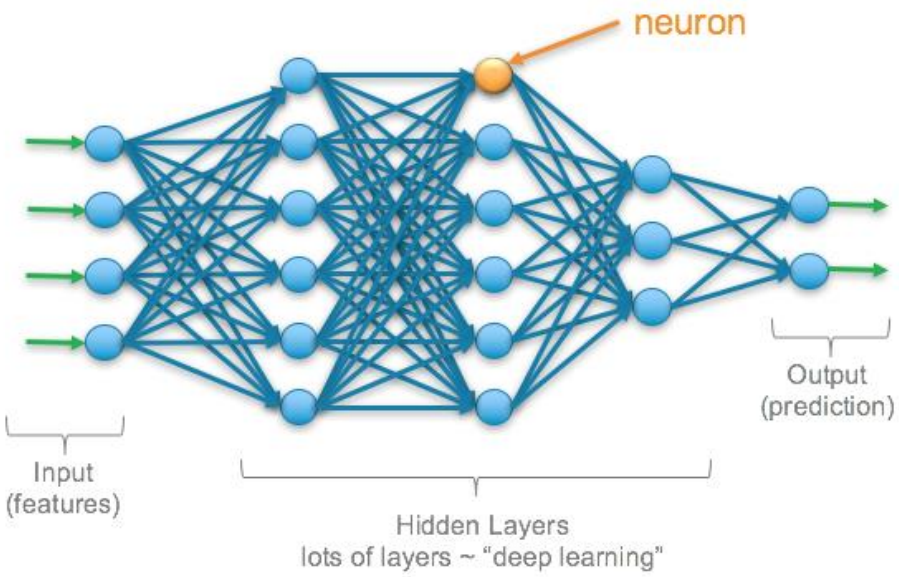
Text Analytics in Sustainable Development Goals (SDGs)



AUEB M.Sc. in Business Analytics (part-time)

Machine Learning and Content Analytics

Instructor: Haris Papageorgiou

Assistants: George Perakis, Aris Fergadis

Team Members:

Lianos Alexandros – p2821909

Marinos Efstratios – p2821913

Papanikou Lydia – p2821916

Table of Contents

Project Description

The challenge of this project is to create a support tool for the academic researcher/scientist who can contribute to making better use of existing scientific literature/research to develop his/her work. The main idea is to train a Neural Network with pre-classified data and create an algorithmic model so that, by entering new data, we can achieve the greatest predictability in new and future publications.

Such tools are already present in the business world and are widely used in various industries. A characteristic example is the biomedical science, where you can get notified of knowledge you need to augment your research.

However, there is a growing interest among governmental researchers and academic institutions in tracking the performance of Sustainable Development Goals (SDGs). To help this scientific research, we can use machine learning algorithms to analyze and connect millions of papers, to measure how SDGs are connected to different research themes.

This tool could be part of the United Nations website, where the user can navigate in a user-friendly platform and focus on specific target indicators.

Mission

The United Nations is a source of big data in the form of text. Between the many resolutions, speeches, meetings, conferences, studies, reports, and internal regulations that exist and that are produced each year, the UN is awash in text.

However, very few people are able to see much more than a small sliver of specialized text. Even fewer can parse the various streams into a coherent and useful picture. What is needed is a quick and objective way to analyse large quantities of United Nations publications according to a desired criterion.

So, our scope is to provide a solution by introducing a proof-of-concept classification system that measures the alignment of publications with each of the pre-defined research themes.

Using machine learning algorithms to analyse digital texts has many advantages. Algorithms can be used at scale with objectivity and can help identify patterns across publications and over time. This approach can also serve as a tool to explore and discover new texts, and to inform the direction of future research. More importantly, this method hopefully inspires other efforts to use modern data analytics to better understand the body of work of the United Nations.

In order to fully test our Natural Language Processing (NLP) solution we performed binary and multiclass classifications, by developing two distinct neural networks (NN) architectures:

- Convolutional NN (CNN)
- Multi-Layer Perceptron NN (MLP)

This project, therefore, is looking into the development, training, optimization/tuning, and evaluation of 15 neural network models, 11 for binary and 4 for multiclass classification. These models are compared based on specific metrics and their ability to predict the research theme of new papers.

To build such an algorithm however, the main issue that must be tackled is the proper annotation of the papers, in order to determine the research theme of them.

In our current iteration, our goal is to train the aforementioned models using Python, implemented in Keras & Tensorflow, and tune them so as to achieve good prediction scores as well as good score metrics (accuracy, f1, recall etc.)

Data

After the annotation process was completed, the raw data consisted of papers referring to several SDGs and had the following format:

- a folder containing all the papers in separate csv files
- a csv file with the correspondence of the class (research theme) to every paper

After running a parser to these files, and making the necessary transformations described in the **"parser_eda_extract.ipynb"** file, we extract a dataframe consisting of 828 papers (rows) and 3 columns as shown below.

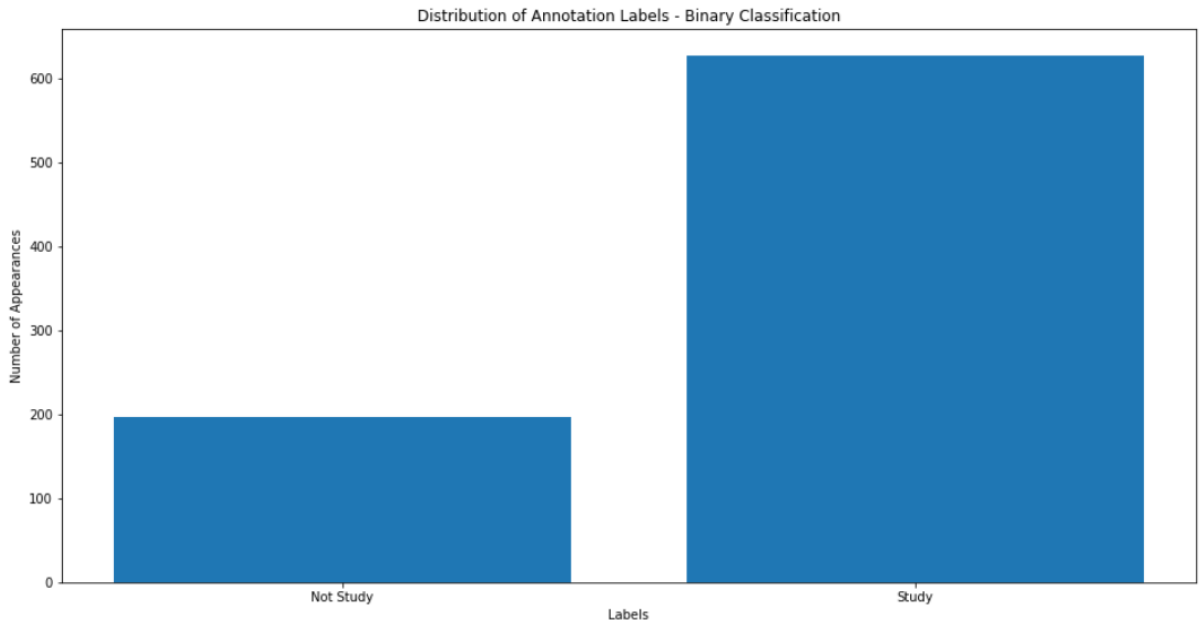
	label	binary_label	text
0	Study	Study	Multivariate Granger causality between CO2 emi...
1	Study	Study	Electricity consumption-GDP nexus in Pakistan:...
2	Study	Study	Ecological total-factor energy efficiency of r...
3	Method	Not Study	Exploration and corrective measures of greenho...
4	Study	Study	Solutions for improving the energy efficiency ...
...	...	...	...
823	Study	Study	How to quantify the resolution of surface clim...
824	Study	Study	A visual method to identify significant latitu...
825	Study	Study	Effect of production system and farming strate...
826	Study	Study	The effect of future climate scenarios on the ...
827	Study	Study	Comparing environmental impacts of beef produc...

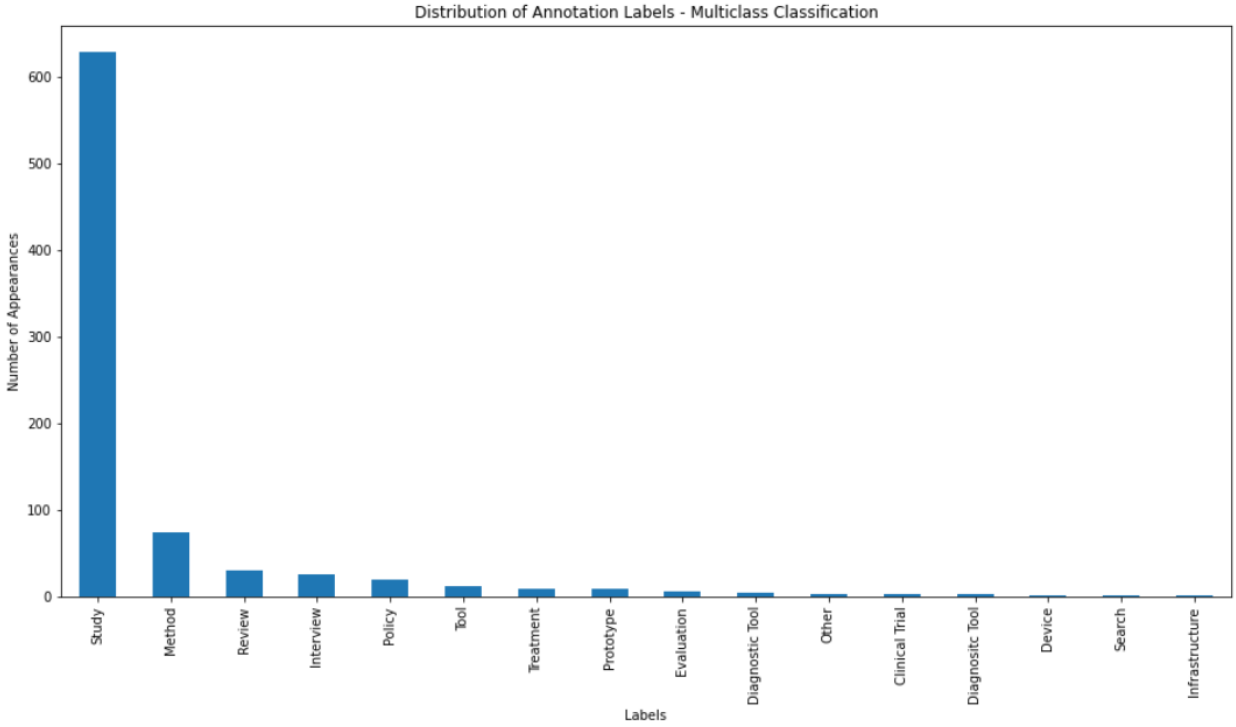
828 rows × 3 columns

The next step is to perform Exploratory Analysis in our data. We make sure that there are no missing values and remove some duplicate papers during the process. Our final dataset now consists of 825 unique papers. Finally, we create two separate dataframes for binary and multi-class classification purposes, before extracting them to different csv files.

The tables and figures below help us get an overview of our data and reach useful conclusions.

	count	unique	top	freq
label	825	16	Study	628
binary_label	825	2	Study	628
text	825	825	Gender norms and social norms: differences, si...	1





In the case of Binary classification, data are divided into 2 categories (Study, No Study). On the other hand, in the case of Multiclass classification of 15 categories, the 9 of them were selected as statistically significant, while the others were rejected. In both approaches, we encountered the problem of imbalanced classes. Imbalanced data typically refers to a problem with classification problems where the classes are not represented equally.

Most classification datasets do not have exactly equal number of instances in each class, but a small difference often does not matter. It is the case where your accuracy measures tell the story that you have excellent accuracy (such as 90%), but the accuracy is only reflecting the underlying class distribution. It is very common, because classification accuracy is often the first measure we use when evaluating models on our classification problems. That is why we added more metrics such as precision, recall, f1 etc.

See the Methodology and Comments/Problems chapters for more on this matter

Label Encoder

We have to convert the categorical text output data into model-understandable numerical data. Label Encoder is part of the SciKit Learn library and it is used to convert categorical data, or text data, into numbers, which our predictive models can better understand.

Basic Preprocessing of text data

The papers that comprise our dataset, originate from an annotation tool. This means that most of heavy preprocessing, such as special characters, tags, html\code\xml etc., was conducted during the annotation process. Even so, we continue with some basic preprocess methods to better modify our dataset for the text classification that takes place later.

Tokenizing text data using Word Embeddings

We cannot feed raw text directly into deep learning models. Text data must be encoded as numbers to be used as input or output for machine learning and deep learning models. Words are called tokens and the process of splitting text into tokens is called tokenization.

A word embedding is a class of approaches for representing words and documents using a dense vector representation. It is an improvement over the more traditional bag-of-word model encoding schemes where large sparse vectors were used to represent each word or to score each word within a vector to represent an entire vocabulary. These representations were sparse because the vocabularies were vast and a given word or document would be represented by a large vector comprised mostly of zero values.

Instead, in an embedding, words are represented by dense vectors where a vector represents the projection of the word into a continuous vector space. The position of a word within the vector space is learned from text and is based on the words that surround the word when it is used. The position of a word in the learned vector space is referred to as its embedding.

For this deep learning project, we will use 2 methods of learning word embeddings:

- GloVe (Using Pre-Trained GloVe Embedding)
- Self-learned as part of a deep learning model (Learning an Embedding)

Keras Tokenizer

We look at tokenizing and further preparing text data for feeding into a neural network using TensorFlow and Keras preprocessing tools. We will make use of two Keras preprocessing tools: the **Tokenizer** class, and the **pad_sequences** module.

Some hyperparameters are necessary for performing tokenization and preparing the standardized data representation, with explanations below.

- **max_words:** maximum number of words from our resulting tokenized data vocabulary which are to be used
- **max_len:** maximum length of sequences
- **oov_token:** token which will be used for tokens encountered during the tokenizing and encoding of test data sequences, created using the word index built during tokenization of our training data
- **pad_type:** When we are encoding our numeric sequence representations of the text data, our sentences lengths will not be uniform, and so we will need to select a maximum length for sentences and pad unused sentence positions in shorter sentences with a padding character. By default, this is done by padding 0 in the beginning of each sequence until each sequence has the same length as the longest sequence.
- **trunc_type:** On the contrary, longer sequences are truncated so that they fit the desired length. By default, truncating is also set to "pre", which truncates the beginning part of the sequence.

Splitting/Shuffling Dataset Procedure

As stated before, our dataset is highly imbalanced as the classes are not represented equally. In this case, random sampling is a very bad option for splitting (we tried it and the results were entertaining…).

So, we reached the conclusion that we should split the dataset in a **stratified** manner to prevent the distribution of the training data to create bias for our model. In this way, the class is proportionally splitted between training and test set.

We choose to split the data according to Pareto Principle, where 80/20 is the common occurring ratio.

Methodology

After the above transformations, and having converted the data to the desired format, we proceeded to classification using two different architectures:

- Multi-Layer Perceptron Neural Network (MLP)
- Convolutional Neural Network (CNN)

To find the most efficient model, we proceeded with different hyperparameters and combinations for each neural network's parameters. All the necessary information, regarding the hyper-parameters of our models and the types of networks, is provided below.

Note: *We highly advise to read this chapter first, and then study the building of our deep learning models.*

Multi-Layer Perceptron Neural Network

Multi-Layer Perceptrons, or MLPs for short, are the classical type of neural network. They are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.

Convolutional Neural Network Architecture

Convolutional Neural Networks, or CNNs, were designed to map image data to an output variable. They have revolutionized image classification and computer vision by being able to extract features from images and using them in neural networks.

More generally, CNNs work well with data that has a spatial relationship. The CNN input is traditionally two-dimensional, a field or matrix, but can also be changed to be one-dimensional, allowing it to develop an internal representation of a one-dimensional sequence. This allows the CNN to be used more generally on other types of data that has a spatial relationship. For example, there is an order relationship between words in a document of text.

Regarding CNNs' Layers

A CNN has hidden layers which are called convolutional layers. When you think of images, a computer has to deal with a two-dimensional matrix of numbers and therefore you need some way to detect features in this matrix. These convolutional layers are able to detect edges, corners and other kinds of textures which makes them such a special tool. The convolutional layer consists of multiple filters which are slid across the image and are able to detect specific features.

When you are working with sequential data, like text, you work with one dimensional convolution, but the idea and the application stays the same, as described above.

Variations - Hyperparameter Tuning

Keras Embedding Layer

Keras offers an Embedding layer that can be used for neural networks on text data with either approaches:

- Learning an Embedding
- Using Pre-Trained GloVe Embedding

It requires that the input data be integer encoded, so that each word is represented by a unique integer. This data preparation step was performed before, using the Tokenizer API also provided with Keras.

The Embedding layer is initialized with random weights and will learn an embedding for all of the words in the training dataset. It is defined as the first hidden layer of a network and it must specify 3 arguments:

- **input_dim:** This is the size of the vocabulary in the text data. For example, if your data is integer encoded to values between 0-10, then the size of the vocabulary would be 11 words.
- **output_dim:** This is the size of the vector space in which words will be embedded. It defines the size of the output vectors from this layer for each word. For example, it could be 32 or 100 or even larger.
- **input_length:** This is the length of input sequences, as you would define for any input layer of a Keras model. For example, if all of your input documents are comprised of 1000 words, this would be 1000.

Regarding the **trainable parameter,**

- We set it to *False* during the pre-trained GloVe approach because the weights of the embeddings are already trained.
- We set it to *True* during the self-trained approach because the weights of the embeddings should be randomly trained.

Dropout Regularization

Dropout Regularization is a computationally cheap way to regularize a deep neural network.

Dropout works by probabilistically removing, or "dropping out," inputs to a layer, which may be input variables in the data sample or activations from a previous layer. It has the effect of simulating a large number of networks with very different network structure and, in turn, making nodes in the network generally more robust to the inputs.

Dropout value needs finetuning because a probability too low has minimal effect and a value too high results in under-learning by the network. Dropout can be used on visible as well as hidden units. Application of Dropout at each layer of the network has shown good results. Regarding the value, we experimented in the range, 0.2 to 0.5 .

Batch Normalization

Batch Normalization is a technique designed to automatically standardize the inputs to a layer in a deep learning neural network.

The general use case is to use Batch Normalization between the linear and non-linear layers in your network, because it normalizes the input to your activation function, so that you are centered in the linear section of the activation function (such as sigmoid) . It applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Once implemented, Batch Normalization has the effect of dramatically accelerating the training process of a neural network, and in some cases improves the performance of the model via a modest regularization effect. The original batch-normalization paper (along with various GitHub threads, such as this thread from 2016) pointed us to the specific implementations in our models.

A "momentum" argument allows us to control how much of the statistics from the previous mini batch to include when the update is calculated. By default, this is kept high with a value of 0.99. This can be set to 0.0 to only use statistics from the current mini batch. After 1-2 tries of setting the *momentum < 0.5*, we did not get the results we expected, mainly due to small batch size. Lastly at one point, we set it to 0.9 - highest low value - to see the impact it might have on the model's performance.

Activation Functions

Activation functions, in layman's terms, are a choice that you must make for each layer:

- **relu:** Very popular neural network activation function. Used for hidden layers, cannot output negative values. No trainable parameters
- **softmax:** Used for multi-class classification. Ensures all output neurons behave as probabilities and sum to 1 .
- **sigmoid:** Classic neural network activation. Often used on output layer of a binary classifier.

Loss functions

BinaryCrossentropy

Computes the cross-entropy loss between true labels and predicted labels. We use this cross-entropy loss for binary classification (labels: assumed to be 0 and 1).

CategoricalCrossentropy

Computes the cross-entropy loss between the labels and predictions. We expect labels to be provided in a one hot representation. We use this loss function for the multiclass classification.

Optimizers

Optimizers are algorithms used to change the attributes of the neural network such as weights and learning rate to reduce the losses. We used the following 3 in our case and heavily customized the SGD:

- Nadam
- Adam
- SGD

Note: We wanted to experiment more with the various optimizers, so we tried to adapt the learning rate, in hopes of increasing performance and reducing the training time (training time was already short but we could always aim for a "shorter" value).

Learning Rate determines the step size at each iteration while moving toward a minimum of a loss function. It influences to what extent newly acquired information overrides old information.

We read various academic papers on this subject but this link proved quite helpful. We observed that fiddling with the learning rate and attributes of SGD produced our 2 best models (based on accuracy and predictability). So that is a win!

Metrics

- Accuracy

Note: In addition to binary/multiclass classification accuracy, there are 3 more metrics that are commonly required for a neural network model on a binary classification problem. However they have been removed from Keras and we needed to calculate them manually -see this link for more. Also read this StackOverflow thread to better understand the context – datascience so thread

- Recall
- Precision
- F1

Keras Callbacks

Model Checkpoint

Save the current weights of the model at different points during training.

Early Stopping

Interrupt training once a target metric being monitored has stopped improving for a fixed number of epochs. For instance, this callback allows you to interrupt training as soon as you start overfitting, thus avoiding having to retrain your model for a smaller number of epochs.

ReduceLROnPlateau

Reduce learning rate when a metric has stopped improving.

Configuration of the Model

The model that we will create is a sequential model meaning that each layer that we add, will use as input the output of the former layer added to the model. We have to specify:

- **embedding_dim:** how large we want the word vectors to be. We set 2 different embedding dims, for self-embedding and for the pre-trained GloVe embeddings
- **max_words:** for how many words we want embeddings
- **maxlen:** how long our sequences are

Training Parameter

Regarding **batch size** and **epochs**, both are integer values:

- The batch size is a hyperparameter that controls the number of training samples to work through before the model's internal parameters are updated.
- The number of epochs is a hyperparameter that controls the number of complete passes through the training dataset. eventually reducing the learning rate step size. Each pass is known as an epoch.

Neurons and Layers

Usually it is not a good idea to use less neurons on intermediate layers. Small layers may act as information bottlenecks, permanently dropping relevant information. When considering neurons and layers as hyper parameters, we mainly examined these 2 factors:

- **Width:** The number of nodes in a specific layer
- **Depth**: The number of layers in a neural network

Note: *Regarding the above observations we set as many as 1024 neurons per layer, for a 12th layered CNN model. It took some time (1-2 hours) but we concluded that these 2 parameters did not have the impact we wanted on our models' performance.*

Results

Binary Classification Results

The 11 binary NN models are compared based on accuracy and their ability to predict the research theme of new papers.

- Predictability: The probability that a text has a research theme classified as "Study"
- Accuracy: Number of correct predictions / Total Number of predictions

Based on the criteria, we observed all manual executions of the models and selected the 2 models with the highest values on accuracy and predictability. The metrics for our binary classification models are summarized below:

Binary Classification - Evaluation Table

Metrics / Scores	Multi-Layer Perceptron Neural Network									Convolutional Neural Network	
	No Embedding Layer			Self-trained Embeddings			Pre-trained GloVe Embeddings			Self-trained Embeddings	Pre-trained GloVe Embeddings
	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6	Model 7	Model 8	Model 9	Model 10	Model 11
Loss	89.13%	54.15%	54.38%	52.36%	55.03%	54.57%	54.77%	55.02%	55.20%	53.83%	63.03%
Accuracy	76.36%	76.36%	76.36%	76.36%	76.36%	76.36%	76.36%	76.36%	76.36%	76.97%	78.79%
Recall	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%	100%	99.38%	91.03%
Precision	76.88%	76.88%	76.88%	76.88%	76.88%	76.88%	76.88%	76.88%	76.88%	77.62%	83.23%
F1	86.74%	86.74%	86.74%	86.74%	86.74%	86.74%	86.74%	86.74%	86.74%	86.96%	86.67%
Predictability	75.42%	59.71%	78.03%	80.85%	76.90%	76.53%	67.61%	66.56%	75.36%	87.01%	64.66%

- **Model with the best Accuracy:** Model 11
- **Model with the best Predictability:** Model 10

Both models were developed with the architecture of Convolutional Neural Networks in mind. The differences in building these 2 models that defined these results are the following:

- **Convolutional layer:** Both models use 1D convnet via the Conv1D layer with different number of filters as the dimensionality of the output space, and a 3-kernel size as the length of the 1D convolution window with a ReLu activation function.
- **Embedding Layer:** Model 10 trains its own embeddings, while Model 11 uses pre-trained GloVe embeddings
- **Batch Normalization:** For Model 10 we set the momentum parameter to 0.9. Model 11 uses the default, 0.99
- **Width and Depth of Neurons and Layers:** Model 10 has only 1 hidden layer, while Model 11 has 2. Moreover Model 10 has less neurons per layer than Model 11.

More Observations

- Model 10 has the 2nd lowest loss while Model 11 has the 2nd highest loss.
- Model 10 has the 2nd highest accuracy. Considering that Model 11 has the highest accuracy, this shows that our CNN models performed overall better on this regard.
- Both CNN models have lower recall values than the MLP models.
- Both CNN models have higher precision values than the MLP models.
- Model 1 has the highest loss amongst the 11 models. Model 1 is our baseline model. It does not have an Embedding layer, or a Batch Normalization layer and it has the lowest number of neurons per layer.

Note: *On the CNN models we also heavily customized the parameters of SGD optimizer (learning rate, decay rate, momentum).*

Statistic Plot Observations

On the next 2 pages we present the summary of the 2 models, along with some statistic plots. We can clearly see that:

For model 11, the training loss decreases with every epoch, and the training accuracy increases with every epoch. On the other hand, the validation loss and accuracy "spike" early on and then proceed to show reverse behavior - validation loss steadily increases, and validation accuracy steadily decreases.

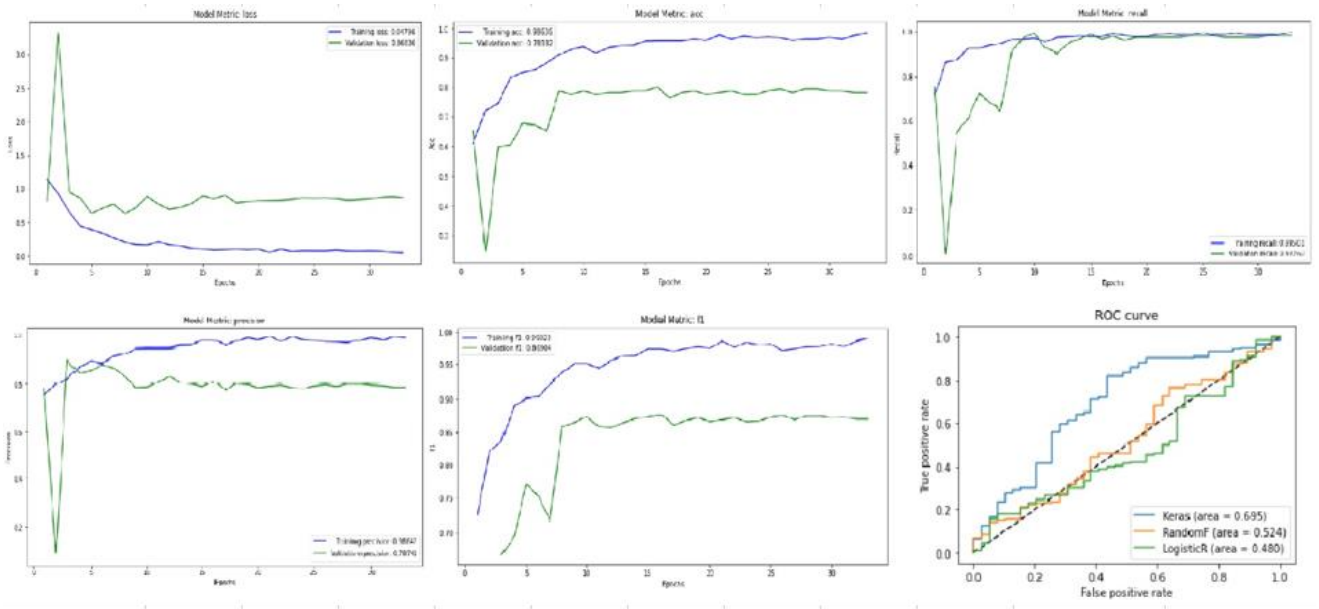
Expectations do not change for model 10 since the pairs of validation and training show similar behavior minus the "big spike" we observed in Model 11. But unlike Model 11, in Model 10 before epoch 5 validation values for accuracy and loss change dramatically.

This is an example of a model that performs better on the training data but does not necessarily perform better on data it has never seen before. In precise terms, what we're seeing is overfitting: especially for Model 10, after the 5th epoch, we're overoptimizing on the training data, and we end up learning representations that are specific to the training data and don't generalize to data outside of the training set.

Model Architecture and Plots for best Accuracy

Below is a summary of model 11:

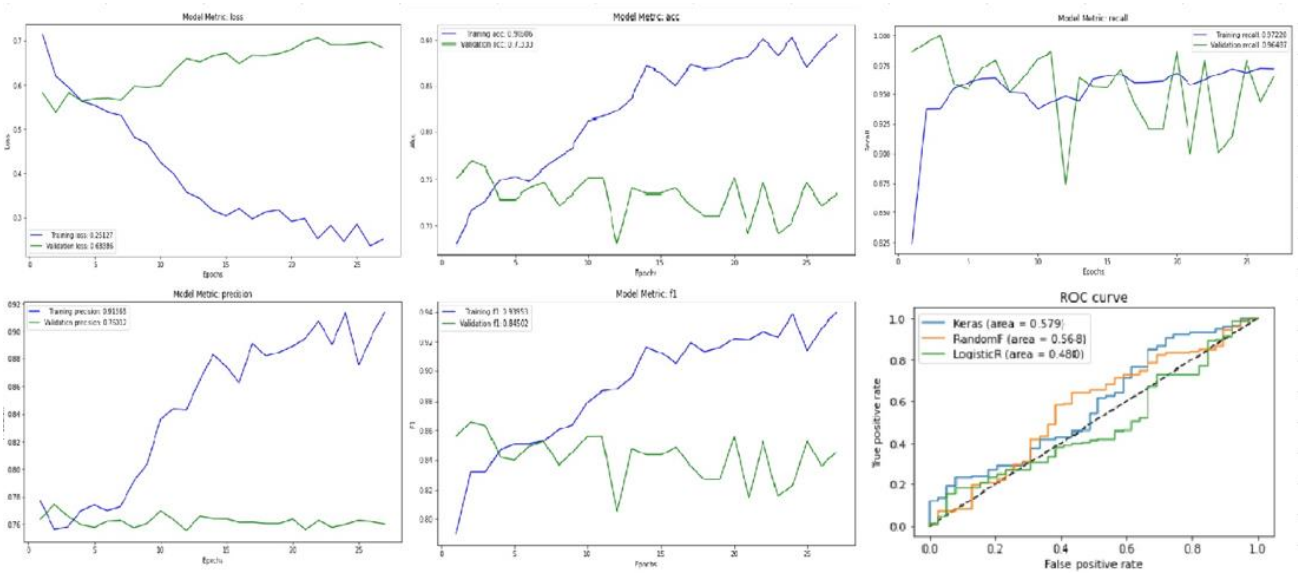
```
--------------------------------------------------

Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_7 (Embedding)      (None, 200, 100)          500000
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 198, 128)          38528
_________________________________________________________________
batch_normalization_36 (Batc (None, 198, 128)          512
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0
_________________________________________________________________
flatten_7 (Flatten)          (None, 128)               0
_________________________________________________________________
batch_normalization_39 (Batc (None, 128)               512
_________________________________________________________________
dropout_43 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_49 (Dense)             (None, 1)                 129
=================================================================
Total params: 539,681
Trainable params: 39,169
Non-trainable params: 500,512
_________________________________________________________________
```



Model Architecture and Plots for best Predictability

Below is a summary of model 10:

```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 200, 50)           250000
_________________________________________________________________
conv1d (Conv1D)              (None, 198, 32)           4832
_________________________________________________________________
batch_normalization_33 (Batc (None, 198, 32)           128
_________________________________________________________________
global_max_pooling1d (Global (None, 32)                0
_________________________________________________________________
flatten_6 (Flatten)          (None, 32)                0
_________________________________________________________________
batch_normalization_35 (Batc (None, 32)                128
_________________________________________________________________
dropout_40 (Dropout)         (None, 32)                0
_________________________________________________________________
dense_46 (Dense)             (None, 1)                 33
=================================================================
Total params: 255,121
Trainable params: 254,993
Non-trainable params: 128
_________________________________________________________________
```



Multiclass Classification Results

The 4 multiclass NN models are compared based on accuracy. The metrics for our multiclass classification models are summarized below:

Multiclass Classification - Evaluation Table				
Metrics / Scores	**Multi-Layer Perceptron Neural Network**		**Convolutional Neural Network**	
	Self-trained Embeddings	Pre-trained GloVe Embeddings	Self-trained Embeddings	Pre-trained GloVe Embeddings
	Model 1	Model 2	Model 3	Model 4
Loss	98.61%	95.34%	120.61%	107.92%
Accuracy	77.16%	77.16%	76.54%	77.16%

Tools - Specifications

We used Google Colab to develop our deep learning models. We did not have any problems regarding performance because the dataset was small, for the task at hand. The only times we had to wait 1-2 hours was when we experimented with neural network depth (we developed a 12^{th} layer CNN pre-trained GloVe at one time)

Members/Roles

Our team consists of three (3) members with programming and business background, so the roles are well-defined. Efstratios Marinos and Alexandros Lianos operate this project as Machine Learning Engineer and Data Scientist accordingly, and Lydia Papanikou as Business Analyst.

The members of the team dealt with all aspects of the project, but due to limited time restrictions and in the context of time management, each member focused on the responsibilities below:

- Alexandros Lianos: pre-processing and transformation of the data
- Efstratios Marinos: DL model building variations and hyperparameter tuning
- Lydia Papanikou: business implementation

A brief background of the team:

Efstratios Marinos

Graduate of Digital Systems | Communication Systems and Networks at University of Piraeus
DevOps Engineer in Piraeus Bank | Winbank

- Responsible of the release management lifecycle of web applications
- Investigating systems sustainably through mechanisms like automation, so as to avoid performing repetitive tasks.
- Measuring and monitoring progress to ensure application releases are delivered on time and within budget, and that they meet or exceed expectations.

Alexandros Lianos

Graduate of Business Administration at University of Piraeus
Media Planner in AB Vassilopoulos | Ahold Delhaize Group

- Interacting with internal departments to understand their needs and wants before formulating a media strategy
- Recommending media plans to ensure marketing campaigns reach their target audience in the suitable media mix combination
- Analyzing and interpreting post-campaign results to evaluate the advertising performance and recommend future refinements

Lydia Papanikou

Graduate of Computer Engineering & Informatics at University of Patras
Business/System Analyst in Intrasoft International

- Working closely with clients such as the European Customs Development project (CUSDEV)
- Gathering customer requirements concerning critical messages exchange between EU customs to map system functionalities to the business system operations.
- Evaluate, analyze, and communicate systems requirements, including the delivery of monthly status reports

Bibliography

MCLA Class Notes

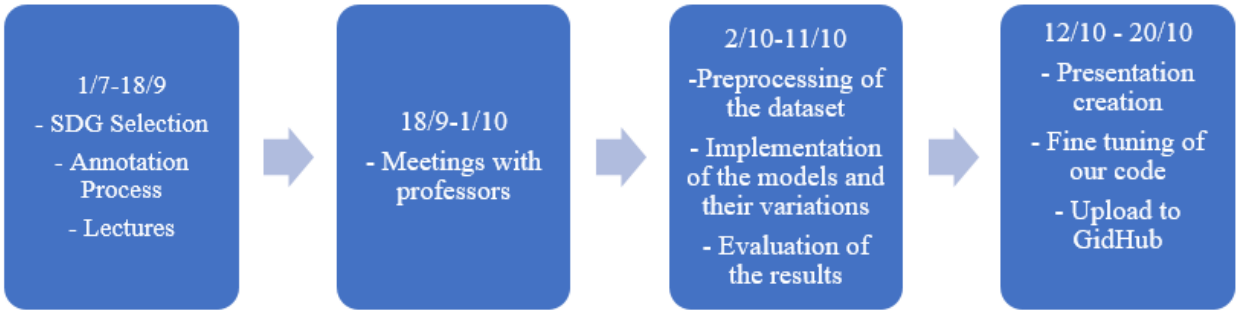
Textbooks:

- Deep Learning with Python, François Chollet
- Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow-2nd Edition, Aurélien Géron

Links:

- https://keras.io
- https://machinelearningmastery.com
- https://www.kdnuggets.com/2020/03/tensorflow-keras-tokenization-text-data-prep.html
- https://www.un.org/esa/desa/papers/2019/wp159_2019.pdf

Time Plan



Contact Persons

Efstratios Marinos

Mobile: 6971883409

stratos.marin@gmail.com

Lianos Alexandros

Mobile: 6945262871

alexlianos88@hotmail.com

Papanikou Lydia

Mobile: 6973680757

lidiapapanikou94@gmail.com

Comments/Problems

Our diverse academic backgrounds combined with our distinct work experience made up for interesting ideas on how to approach the various aspects of the project. Through conversations and different perspectives, we managed to achieve the synergy required, to produce some amusing results (from unusual validation losses to precise accuracies). At the end we developed a functional solution and output good quality metrics.

The difficult part - considering the time limitations - was to decide which type of Neural Network Architecture we should implement to solve the problem in the most effective way. Because there is no clear rule of thumb we decided on a manual approach and tested numerous parameters to see which provided the best model performance. This was not the most efficient way, but it was challenging, and we expanded our understanding on Deep Learning. All in all, it was made clear that finding the best configuration for these models, mainly through hyperparameter optimization, is not a trivial challenge and should be automated (food for thought for the next dl project).

The elephant in the room

During the initial development, we observed the imbalanced data classes. Imbalanced data typically refers to a problem with classification algorithms (either binary or multiclass), where the classes are not represented equally. Imagine the following scenario:

"We create a binary classification model and get a 90% accuracy. It is very common, because classification accuracy is often the first measure we use when evaluating models on our classification problems. Before the celebrations, we decide to dig a little deeper and discover most of 90% of the data belongs to one class. Meaning we cannot really trust the accuracy metric. By the way, the scientific term is "accuracy paradox" which is the paradoxical finding that accuracy is not a good metric for predictive models when classifying in predictive analytics – from Wikipedia"

After mild frustrations on this subject we came up with some options to implement the next time:

- Collect more data – *on this project our data were already classified*
- Different Performance Metrics – *we did that on this project for binary classification*
    - **Precision:** A measure of a classifiers exactness.
    - **Recall:** A measure of a classifiers completeness
    - **F1 Score:** A weighted average of precision and recall.
    - **ROC Curves:** Like precision and recall, accuracy is divided into sensitivity and specificity and models can be chosen based on the balance thresholds of these values.
- Resampling the dataset, either with over-sampling or under-sampling

Note: *In a real case scenario it is not always possible to collect more data. And the multiple performance metrics produce "subjective" results. The best approach might be to play around with resampling (or automate it, if possible). It is also the most data engineering implementation out of the 3 already mentioned.*