Title: Argument Detection

GOAL 13: Take urgent action to combat climate change and its impacts

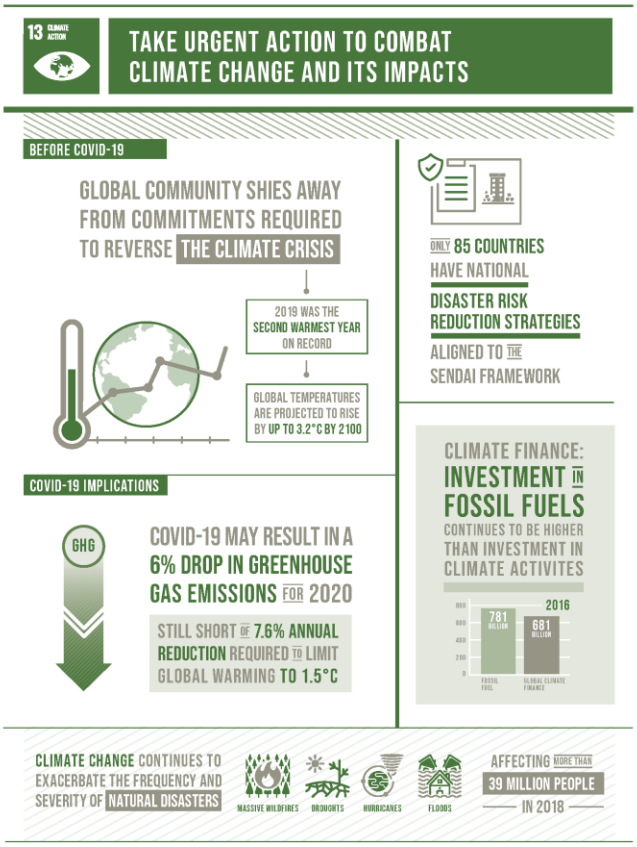
TABLE OF CONTENTS

1 Description

The Sustainable Development Goals are a call for action by all countries – poor, rich and middle-income – to promote prosperity while protecting the planet. They recognize that ending poverty must go hand-in-hand with strategies that build economic growth and address a range of social needs including education, health, social protection, and job opportunities, while tackling climate change and environmental protection.

In September 2015, the General Assembly adopted the 2030 Agenda for Sustainable Development that includes 17 Sustainable Development Goals (SDGs). Building on the principle of "leaving no one behind", the new Agenda emphasizes a holistic approach to achieving sustainable development for all. For the current project some of the main goals were assigned to specific teams for analysis.

In the current report the Goal that was chosen for analysis is Goal 13: Take urgent action to combat climate change and its impacts. In the below screenshot we can see an overview of goal 13 along with the targets and indicators that are connected with it.



Sustainable development and climate change are goals that should not be pursued independently by any country. This SDG emphasizes a major milestone since the challenge of climate change in the earlier Millennium Development Goals was not fully exhausted. A climate that continues warming up will affect food security, freshwater availability and energy among other necessities of life. Climate change is already having a profound and alarming impact worldwide. Concerted action is urgently needed to stem climate change and strengthen resilience to pervasive and ever-increasing climate-related hazards.

The Sustainable Development Goal on climate change has clearly set out several critical targets, listed below, which every country should acknowledge and outline the specific targets and responsibilities befalling them. Here are the main targets for SDG 13:

- Reinforce adaptive capacity and resilience to natural disasters and hazards related to climate change in every country
- Incorporate measures to fight climate change into national planning, strategies and policies
- Improve awareness-raising, education and institutional and human capacity on mitigation, impact reduction, early warning and adaptation to climate change
- Implement the assurance undertaken by the countries under the UN Framework Convention on Climate Change to mobilise jointly $100 billion every year by 2020 to deal with the unique needs of the developing nations in regards to effective mitigation actions, transparent implementation and full operationalising the Green Climate Fund soonest possible
- Promote actions for enhancing capacity for better climate change-linked management and planning in the least developed nation, including small island states, with a focus on youth, women, and marginalised local communities

There is an important collective responsibility on every country to fight climate change while acknowledging the unique circumstances each nation faces in their capacity to deal with the issue. The private sector must take on this responsibility of fighting climate change, where they will take the necessary actions in agreement with potential and circumstances.

Given SDG13 and its socioeconomic impacts, we gathered scientific papers from which we could extract whether this goal is achieved and if yes in what extend. In other words we approach the achievement of SDG 13 from the insights of scientific society. Claim and respective evidences in the papers give us the insights we need to analysis the evolution of goal 13.

Business Aspect

The private sector, including businesses and companies, have increased their part in fighting climate change, but there is still more to be done. There are numerous risks facing companies today including reputational risks, market risks and risks related to climate change policies. It is vital for businesses to act quickly and deal with this risk by seizing opportunities such as new streams of revenue or new investment.

An implementation of such a project, as described above, is included under the spectrum of Text Analytics Projects, which is supported by Natural Language Processing algorithms. Given the great volume, velocity and variety of existing publication, there is a lot of data that could be exploited to measure the progress of the United Nations towards target indicators that have been set.

Core tasks inside the business workflow for achieving this project, include data management, analytics, and visualizations. Such tasks will transform any organization that has to do with sustainable development into a data-driven organization. However, the business workflow is not a straight line but looks more like a circle. Therefore, each steps may not be done once but repeated until we finalize the outcome. Given for instance that “garbage in is also garbage out” the data preparation process is the most important one and took the 70% of our efforts and activities overall. The data collected to support the analysis passed through various steps before getting the needed format to feed out machine learning algorithms. The detailed process will be analyzed in following parts of current report.

Main departments inside an organization that would be impacted by such a project are for sure IT, Digital Transformation department, Innovation and New Businesses as well as Marketing departments that promote both technology and social responsibility combined actions.

2 Mission

The existing Problem

The problem that we have to face is that every year a great number of papers is published without any accurate content quality evaluation. Thus, there are many papers that do not include the necessary information in order to elaborate their findings. Bringing this situation in our project frame, there are plenty of papers in web which refer to climate change and its impacts worldwide, However, on the one hand a portion of those do not propose or support any new proposition and on the other hand they do not provide evidences that support their claims.

Therefore, researchers and analysts should be very cautious about the papers that they study to capture the progress of the SDG goals regarding the environmental extensions of climate change. As a result, the scientific society is under the danger to create a total false interpretation of the reality. Given the fact that environments is one of the most important factors for the Nations but also for the regular operation of the companies, any wrong decision and perspective may cause serious environmental, academic, business and financial damage.

Current Situation

Right now the United Nations publishes every year reports regarding the progress of the Indicators for every Goal based on various statistics and policy documents. Those reports show in what extend each goal is achieved or not. In that way United Nation can understand the status of each crucial factors that have been set and how they can proceed with actions and research to get results the soonest possible.

Our approach

We believe that a good solution for all this public knowledge available for analysis is to apply sophisticated filtering on it. This filtering has to do with specific criteria that we set in order to evaluate a document (paper, publication, statistical article, ebook etc) as good for analysis or not. Except from public online results, as students and analysts we granted access to various portals where we can browse and search for papers that will become our dataset to feed the machine learning algorithms.

After collecting the abstracts of the papers that seem to be suitable from a first glance, we proceed with a more detailed evaluation where we searched for claims supported by evidences under a consistent theme (related to climate change in our case). As it can be easily understood, there were many papers that were dropped out of our pool of data because the proved poor for our analysis. After finalizing the valid batch of data we proceed with working on a platform named Inception in order load the documents and assign to each document the below:

- Topic: The topic is what the discourse is about. The paper's title usually is a good indicator about the topic. Usually the topic is found in sentences where the authors state the aim, objective or purpose of their study
- Research Theme: The Research Theme expresses the overarching goals of the authors and here we investigate how they achieve it.
- Claim: To find out the material in a text, we need to identify and evaluate its arguments. An argument consists of one or more claims that something is, or should be, the case
- Evidence: the evidence is the justification for why claim or claims should be accepted

A combination of people from different background with technology knowledge are needed to evaluate the sources. We also come to the realization that that in order to confirm that paper is valid, we need to find claims supported by real evidences connected with them. In addition, we take the assumption that an accurate paper is accurate independently of the research topic that is being analyzed.

In that way United Nation could take into consideration only the papers that have at least one claim and evidence in order to be scientific elaborated. So, using Neural Networks and NLP algorithms our team managed to build a model that given a sentence, responds if this one is a claim, an evidence or is it insignificant. More specifically, two different types of neural networks, CNN and RNN, were implemented and the one that performed better was chosen as a winner.

3 Data

Data Acquisition

The first step regarding collecting the data is the data acquisition process.

- Publications data [time period: July-Sept]: SDG search queries based on indicators were used/run (primarily) on Scopus and Mendeley, in order to find publications for each selected SDG. Specifically, we kept information regarding the title and abstract, along with the DOI identifier from each paper.

More specifically, we used search queries to define our advance search in Scopus . Search Queries are elegant constructions with keyword combinations and boolean operators, in the syntax specific to the Scopus Query Language. Indicative queries used for this process can be found in the Appendix section.

- Annotation Process [time period Aug-September]: We focused on the annotation of three components (Topic, Argument, Research Theme). We continued this procedure until we had at least 150 complete documents (i.e. they have at least one Evidence and one Claim sentences).

As the final stage of data acquired for feeding the models in our project was a batch of 889 documents that were collected from different teams working on different SDG target. Therefore, our initial dataset wasn't dedicated to SDG 13 that was assigned to our team, but was generalized to all the SDG targets that were studied in the current assignment.

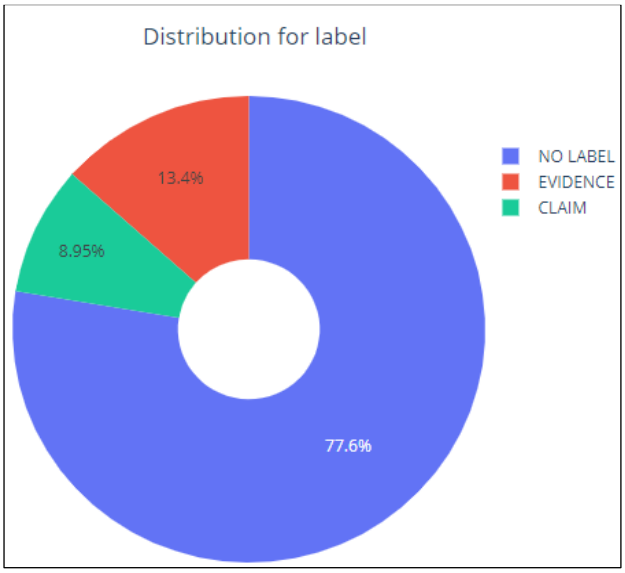
Each document was a .csv format file with the following structure:

OPQ_G6B1_PMID_27346321.csv

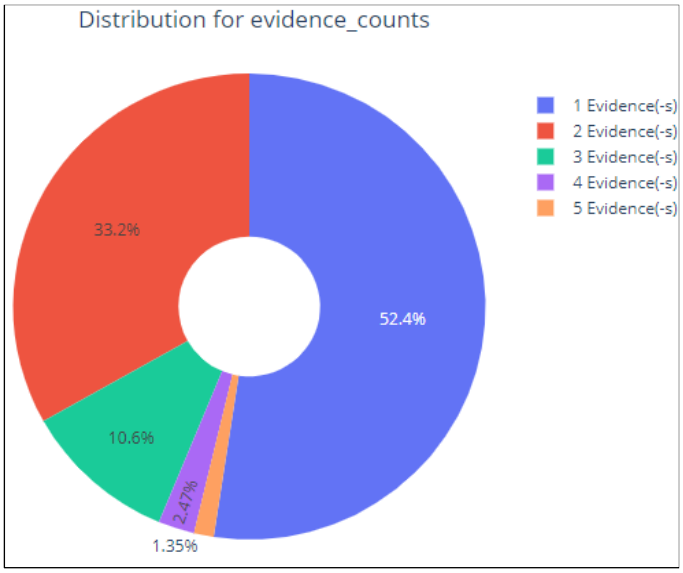
```
label,sentence
_,"Epigenetic basis of sensitization to stress, affective episodes, and stimulants: implications for illness progression and prevention"
_,"Objectives: The process of sensitization (increased responsivity) to the recurrence of stressors, affective episodes, and bouts of substance abuse that can drive illness progression in the recurrent affective disorders requires a memory of and increased reactivity to the prior exposures."
_,A wealth of studies now supports the postulate that epigenetic mechanisms underlie both normal and pathological memory processes.
_,Methods: We selectively reviewed the literature pertinent to the role of epigenetics in behavioral sensitization phenomena and discuss its clinical implications.
_,"Results: Epigenetics means above genetics and refers to environmental effects on the chemistry of DNA, histones (around which DNA is wound), and microRNA that change how easily genes are turned on and off."
Evidence,"The evidence supports that sensitization to repeated stressor, affective episodes, and substance is likely based on epigenetic mechanisms and that these environmentally based processes can then become targets for prevention, early intervention, and ongoing treatment."
_,"Sensitization processes are remediable or preventable risk factors for a poor illness outcome and deserve increased clinical, public health, and research attention in the hopes of making the recurrent unipolar and bipolar affective disorders less impairing, disabling, and lethal by suicide and increased medical mortality."
Claim,"Conclusions: The findings that epigenetic chemical marks, which change in the most fundamental way how genes are regulated, mediate the long-term increased responsivity to recurrent stressors, mood episodes, and bouts of substance abuse should help change how the affective disorders are conceptualized and move treatment toward earlier, more comprehensive, and sustained pharmacoprophylaxis."
```

Data Pre-processing

The initial step was loading the different files in one unified data frame for further processing. We imported multiple csv files into pandas and concatenate into one DataFrame. Based on this data frame, we get the below statistical distribution of whether a sentence inside a document is labeled as claim, evidence or it is no labeled at all. We also present the number of evidences in each document.



Pie Chart 1: distribution of each label in the collection of our documents



Pie Chart 2: number of evidences in each document

From the above pie chart, it is obvious that the majority of sentences are no labeled, which is understandable and can give us an insight of the possible outcome of the predictive models that we are building. The data was already tokenized at a sentence level. Sentence tokenization is the process of splitting text into individual sentences. Our sentences are already tokenized in this data format, in different rows. So, before proceeding to data cleansing, the data set should be as similar as possible. Therefore, a prior step to data cleansing is data transformation. In this phase we transform all the tokens in the sentences to lower cased.

Data Cleansing

During this phase of data cleansing we start by using the regex-transformation. The need was to replace any non-letter, space, or digit character in the headline and remove all special characters, punctuation and spaces from the tokens. In addition, we replaced any comma existed with a space character, because we observed that by leaving commas between words the vocabulary was corrupted. Following this process, we tried to remove the noise from our data and produce sentences that include only letters and numbers that add value to our analysis. In order to achieve the above mention, we used mainly two libraries: sklearn and the NLTK. Those were imported and used to provide already built methods and functions.

After removing symbols and non-value characters we proceed with stopwords exclusion. In order to take into account all the possible stopwords that may exist in our dataset, we create a superset that combined both stopwords lists coming from the two libraries.

Data Preparation

The third step of our data preparation phase includes the lemmatization process. **Lemmatization** usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma

For grammatical reasons, documents are going to use different forms of a word, such as organize, organizes, and organizing. Additionally, there are families of derivationally related words with similar meanings, such as democracy, democratic, and democratization. In many situations, it seems as if it would be useful for a search for one of these words to return documents that contain another word in the set. The goal of lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form. For instance:

am, are, is → be

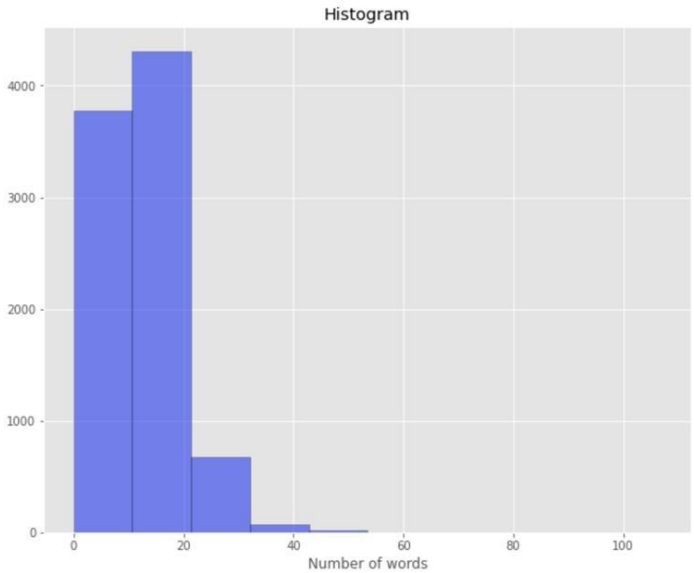
car, cars, car's, cars' → car

In order to implement lemmatization on our data, we use a lemmatizer, a tool from Natural Language Processing, which does full morphological analysis to accurately identify the lemma for each word. We apply a lemmatizer because lemmatizers rely on correct language data (dictionaries) to identify a word with its lemma. Also, the result will always be another dictionary item (infinitives, singular forms…), and not a "stem", what sometimes can be difficult to define (especially if working with typologically different languages).

After that we proceed with **tokenization and padding**. We use keras tokenizer in order to represent each sentence as the list of numbers and not words Tokenizer create a dictionary of words that maps every word of the vocabulary in a number. As the best practice showcases to apply tokenization in training dataset, we split our dataset in test and training prior to tokenization application.

As a first step we create a tokenizer for all the words in the training set. In that way we identify the maximum number of words of the training dataset. However, there is no need to keep all words from train dataset, thus we chose to apply tokenizer in a portion of maximum words included in train dataset. Therefore, in a following step we create a tokenizer that includes only the 90% most frequent words.

	count	mean	std	min	25%	50%	75%	max
0	8851.0	12.42933	6.795068	0.0	8.0	12.0	16.0	107.0



As it is observed from the above charts the most sentences have around 33 words. In other words, after running some statistics on the data, 33 represents the 98th percentile of the sentences' length. To proceed, we have to make sure that all text sequences we feed into the model, later on, have the same length. So, we consider 33 as the most suitable number of the length of the sentences that we choose for the padding afterwards.

There are couple of reasons padding is important: It's easier to design networks if we preserve the height and width and don't have to worry about tensor dimensions when going from one layer to another. It also allows us to design deeper networks, because without padding, reduction in volume size would reduce too quickly. Padding actually improves performance by keeping information at the borders.

In this step we ensure that all sequences in a list have the same length (33). With the intention to achieve that we pre-padded and pre-truncated the sentences. In other words, in order to achieve the limit of 33 words per sentences we either add zeros at the end of the sentence or remove the words from the end in order to achieve the uniformity.

As a final step before building our models, we deal with the categorical variables we applied **one hot encoding,** a process by which categorical variables are converted into a form that could be provided to ML algorithms to do a better job in prediction. Therefore, we replace the label column that referrers to the category of the sentence (NO LABEL, EVIDENCE, CLAIM) in three discrete columns with binary inputs: 1 if the sentence is what the column represents or 0 otherwise.

```
Convert class vector to binary class matrix (for use with categorical_crossentropy)
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       ...,
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]], dtype=float32)
```

Below you can find the flow of data transformation with an indicative example from the data set:

```
what is apparent is that the tools and techniques afforded by this array of novel and gamechanging sensing platfor
ms present our community with a unique opportunity to develop new insights that advance fundamental aspects of the
hydrological sciences
```

```
['apparent', 'tool', 'technique', 'afforded', 'array', 'novel', 'gamechanging', 'sensing', 'platform', 'present',
'community', 'unique', 'opportunity', 'develop', 'new', 'insight', 'advance', 'fundamental', 'aspect', 'hydrologic
al', 'science']
```

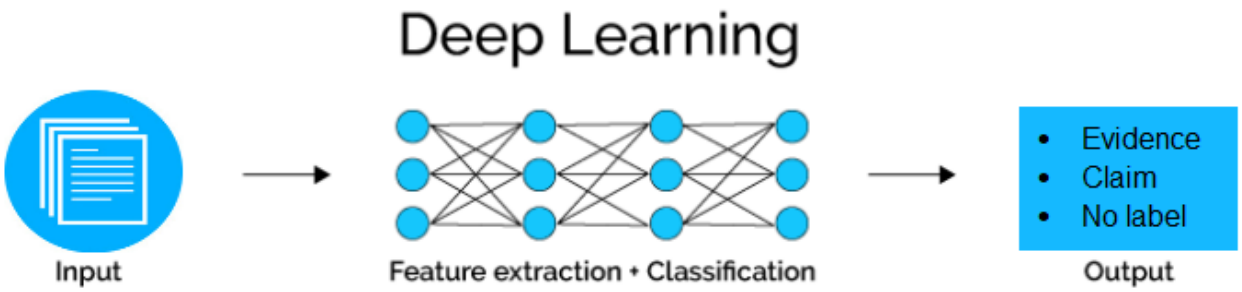
```
['apparent tool technique afforded array novel gamechanging sensing platform present community unique opportunity
develop new insight advance fundamental aspect hydrological science']
```

The final form of the training dataset is the following:

```
New data shape: (8851, 33)
[[1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 ...
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]
 [1 1 1 ... 0 0 0]]
```

4 Methodology

Deep Learning Argument Classification for SDG Goals



Deep learning is the application of artificial neural networks using modern hardware. It allows the development, training, and use of neural networks that are much larger (more layers) than was previously thought possible.
The classes of artificial neural networks that we are going to use are:

- Convolutional Neural Networks (CNNs)
- Recurrent Neural Networks (RNNs)

Models' Layers

Input layer

It is the first layer in the neural network. It takes input values and passes them on to the next layer without applying any operations to them. We begin by defining an input layer that accepts a list of words with a dimension of 33 (= max words in one sentence).

Embedding layer

The Embedding layer is defined as the first hidden layer of a network. It can be used for neural networks on text data. It requires that the input data be integer encoded, so that each word is represented by a unique integer. The Embedding layer is initialized with weights from the embedding matrix and will learn an embedding for all of the words in the training dataset. The output of the embedding layer is a list of the coordinates of the words in this vector space. We need to define the size of this vector space and the number of unique words we are using.

Convolutional Layers (for CNN model only)

These layers create a convolution kernel that is convolved with the layer input over a single spatial dimension to produce a tensor of outputs. A convolutional layer consists of a set of "filters". These filters only take in a subset of the input data at a given time, but are applied across the full input by sweeping over it. The operations performed here are still linear, but they are generally followed by a non-linear activation function.

Bidirectional LST Layer (for RNN model only)

This layer allows the networks to have both backward and forward information about the sequence at every time step. Using bidirectional run our inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backward you preserve information from the future and using the two hidden states combined you are able in any point in time to preserve information from both past and future.

Bidirectional LSTMs are supported in Keras via the Bidirectional layer wrapper. This wrapper takes a recurrent layer (e.g. the first LSTM layer) as an argument. Here we chode to use 2 Bi-LSTM layers back to back. The two output of the two LSTMS layers are concatenated together (the default), providing double the number of outputs to the next layer.

By wrapping the LSTM hidden layer with a Bidirectional layer, we create two copies of the hidden layer, one fit in the input sequences as-is and one on a reversed copy of the input sequence. The two output of the two LSTMS layers are concatenated together (the default), providing double the number of outputs to the next layer.

Pooling Layer (for CNN model)

It downsamples the output of the prior layer, reducing the number of operations required for all following layers, but still passing on the valid information from the previous layer. Here we will use global max pooling, which is a pooling operation that downsamples the input representation by taking the maximum value over the time dimension. Global pooling layers can be used in a variety of cases. Primarily, it can be used to reduce the dimensionality of the feature maps output by some convolutional layer, to replace flattening and sometimes even dense layers in your classifier.

Fully Connected (FC) Layer (for CNN model)

It is a linear operation in which every input is connected to every output by a weight, generally followed by a non-linear activation function. This layer will compute the class scores.

RegularizationLayer (Dropout)

They are used to overcome the over-fitting problem (increase the validation accuracy) thus increasing the generalizing power. Here, we applied a dropout layer where at each training stage, individual nodes are either "dropped out" of the network with probability p or kept with probability 1−p, so that a reduced network is left. Incoming and outgoing edges to a dropped-out node are also removed and only the reduced network is trained.

For CNN model:

The dropout layer will randomly assign 0 weights to the neurons in the network. Since we chose a rate of 0.2, 20% of the neurons will receive a zero weight. After training for the epoch the removed nodes are then reinserted into the network with their original weights. At test time, no dropout is performed. Instead, the layer's output values are scaled out by p to compensate for the fact that there are more neurons active than in training.

For RNN model:

The dropout layer will randomly assign 0 weights to the neurons in the network. Since we chose a rate of 0.1, 10% or 0.5, 50% of the neurons will receive a zero weight. After training for the epoch the removed nodes are then reinserted into the network with their original weights. At test time, no dropout is performed. Instead, the layer's output values are scaled out by p to compensate for the fact that there are more neurons active than in training.

With this operation, the network becomes less sensitive to react to smaller variations in the data. As expected, the models' performance improved after using dropout which proves the effectiveness of dropout in reducing models' overfitting.

Output Layer (for CNN model)

It is the last (fully-connected) layer and receives its input from the last hidden layer. In classification settings it represents the class scores. In our network we have a multiclass classification scheme with three classes, that's why we decided to have 3 neurons in the output layer.

We also used activation functions in order to squash the values into a smaller range. We have a multiclass, single-label classification problem. So, the last-layer activation function we will use is softmax and the loss function is the categorical cross entropy. By training the CNN network with last softmax dense layer we are able to identify the correct weights for the network by multiple forward and backward iterations, which try to minimize categorical cross entropy. As a result, we can now extract features from the CNN model.

Output Layer (for RNN model)

Last, we have the output Layer. The output layer decides what the next hidden state should be, given that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

In our network we have a multiclass classification scheme with three classes, that's why we decided to have 3 neurons in the output layer. We also used activation functions in order to squash the values into a smaller range. We have a multiclass, single-label classification problem. So, the last-layer activation function we will use is softmax and the loss function is the categorical cross entropy. By training the RNN network with last softmax dense layer we are able to identify the correct weights for the network by multiple forward and backward iterations, which try to minimize categorical cross entropy. As a result, we can now extract features from the RNN model.

Convolutional neural networks (CNNs)

CNN is a deep learning architecture that consists of an input layer, multiple neural hidden layers and an output layer. In NLP tasks token sequences are usually used as input. Then, CNN filters preform as n-grams over continuous representations. After that, these n-grams filters are combined by subsequent network layers, dense layers.

CNN can learn the features and distinguish between them automatically. All the weights in the convolutional layers are shared which means that the same filter is used for all the fields within a layer to improve the performance and decrease the memory space.
They are very effective at document classification, namely because they are able to pick out salient features (e.g. sequences of tokens) in a way that is invariant to their position within the input sequences. Simply put, a convolution is a sliding window function applied to a matrix.
Here we will use 1D CNN, where the kernel can only move in one dimension. The input and output data of 1D CNN is 2-dimensional (tensor).

Each word is a vector that represents a word. The filter covers at least one word. In contrast with 2D networks, 1D networks allow us to use larger filter sizes, because a filter of size 7 or 9 contains only 7 or 9 feature vectors. Also we can afford to use larger convolution windows with 1D CNNs, because a window of size 3 contains only 3 feature vectors, while with a 2D convolution layer, a 3 × 3 convolution window contains 3 × 3 = 9 feature vectors.

Pre-trained GloVe Embedding

In order to represent words to our deep learning algorithms we will use word embedding models, which map words to vectors (where the size of the vector is much smaller). A word embedding is a way of representing text where each word in the vocabulary is represented by a real valued vector in a high-dimensional space. The words that have similar meanings will have similar representation in the vector space.
In our analysis we will add an embedding layer to our neural network by using a pre-trained embedding layer**,** which is a good way to improve the performance of text classification. We will use the GloVe1 method.

Model Structure

We used a pre-trained word embedding (GloVe) to capture similarities between words and semantics of word sentences. After that, we applied a CNN architecture to encode and leverage the information from the input text sequence. Specifically, we used two convolutional layers (Conv1D + activation + dropout + pooling). Finally, a fully connected layer with a softmax layer were used to compute the class distribution for each sentence.

[1] GloVe embedding is a publicly available 100-dimentional embedding trained on 6 billion words from web text and Wikipedia.

CNN Architecture Overview

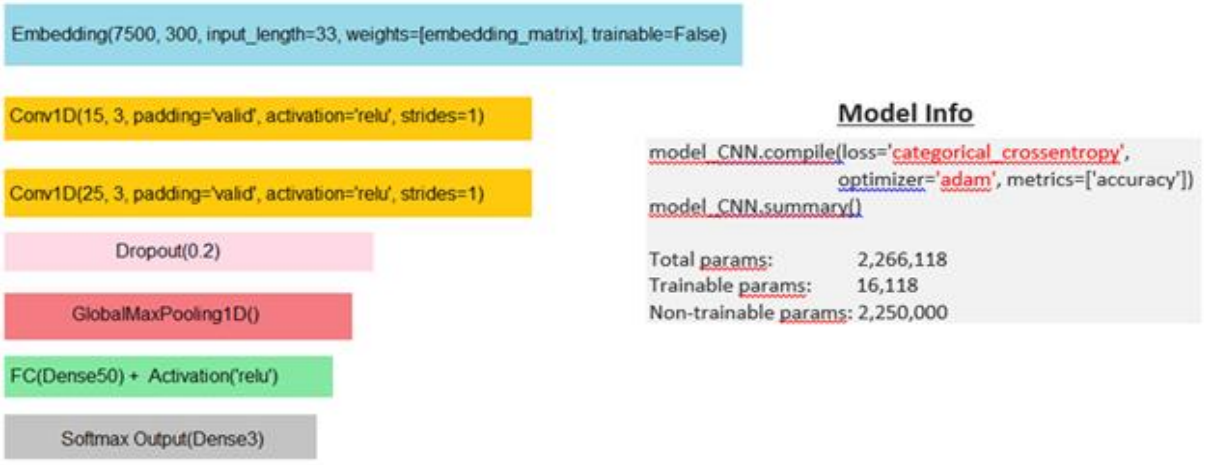


Figure 1: CNN Architecture

A simple CNN is a sequence of layers, and every layer of a CNN transforms one volume of activations to another through a differentiable function. We used four types of layers to build CNNs architectures: Embedding Layer, Convolutional Layer, Pooling Layer and Fully-Connected Layer. We will stack these layers to form a full CNN architecture. In the following table we present the shape and the number of parameters for each of the above layers:

Layer	Shape	Number of parameters
Embedding Layer	(batch_size, input_length, output_dim) = (None, 33, 300)	(input_dim * output_dim) = 7500 * 300 = 2,250,000
1st Conv1d Layer	(batch_size, length_input_sequences – kernel_size +1, number_of_filters) = (None, 33-3+1,15)=(None,31,15)	(number_of_filters) * (kernel_size * number_of_features) + one_bias_per_filter =n * (k * emb_dim) + n = 15 * (3*300) +15 = 13515
Note: The shape and the number of parameters for the 2nd convolutional layer will be computed as before, but this time we will take as features the ones produced from the previous convolutional layer.		
2nd Conv1d Layer	(batch_ size, 31-3+1, 25) = (None, 29, 25)	25*3*15 + 25 = 1150
Dropout Layer	Same as before (None, 29, 25)	0
Pooling Layer	(batch_ size, features) = (None, 25)	0
Comment: The global max pooling layer, has got no learnable parameters because all it does is calculate a specific number, no backprop learning involved. Thus, the number of parameters = 0.		
FC Layer	(batch_size, units) = (None, 50), where units is the dimensionality of the output space	((current layer neurons c *previous layer neurons p)+1*c) = (50 * 25) + 1*50 = 1300
Output Layer	(batch_size, number_of_classes) = (None,3)	(3 * 50) + 1*3 = 153
Note: For the output layer, the number of parameters is the product of the number of neurons in the current layer c and the number of neurons on the previous layer p, plus the bias term.		

Table: Shape and number of parameters of each layer

Hyper-Parameters

In order to improve the performance of our model we did the following:

1. Fine tune Hyper-parameters: Hyper-parameters are the variables which are set before training and determine the network structure & how the network is trained. (e.g. batch size, number of epochs).
2. Use Dropout Layer

After our analysis, which can be found in the file *'CNN_parameter_testing.ipynb'*, we observed that the best results were achieved using the values shown in table 2.

Layer	Hyper-parameters	Values
CNN	Number of filters (1st layer)	15
	Number of filters (2nd layer)	25
	Kernel size	3
	Hidden size	50
	Epochs	100
	Batch size	32
	Dropout rate	0.2

Table: Hyper-parameters for CNN

Compile the network

The loss function computes the error for a single training example. The cost function is the average of the loss functions of the entire training set. Our choice here is categorical cross entropy.

The optimizer is a search technique, which is used to update weights in the model. Our choice here is Adaptive Moment Estimation (Adam) which uses adaptive learning rates.
Performance metrics are used to measure the performance of the neural network. Accuracy, loss, validation accuracy, validation loss, mean absolute error, precision, recall and f1 score are some performance metrics. Our choice here is accuracy.

Performance Measurement

After performing the hyper-parameter tuning process and fitting the model to the training data, we need to evaluate its performance on totally unseen data (the validation set). When dealing with classification problems, there are several ways that can be used to gain insights on how the model is performing. We have created the confusion matrix and studied the accuracy, along with $F1-score$, when comparing models and when choosing the best hyper-parameters.

Confusion Matrix

Confusion matrix is a useful metric to evaluate performance. Each cell $(i,)$ in the confusion matrix contains the number of instances of class i that were predicted to be in class. A good classifier will accumulate values on the diagonal. We will use the confusion matrix to take a gist of the distribution of our correct predictions and compute some other metrics in order to be more accurate in our evaluation.

Classification Report

The classification report function builds a text report showing the main classification metrics.

- Accuracy: is the percentage of sentences correctly classified
- Precision: is the percentage of predicted positives that were correctly classified
- Recall: is the percentage of actual positives that were correctly classified
- F1-Score: is the harmonic mean between recall and precision values

The final value of these metrics relies between 0 and 1, where 1 indicates a perfect model.
The formula to compute the above metrics is shown below:

$$\text{Accuracy} = \frac{\text{true samples}}{\text{total samples}} \qquad \text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}} \qquad F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precistion+recall}}$$

Learning Rate

The learning rate is a hyper-parameter that controls how much to change the model in response to the estimated error each time the model weights are updated. It has a small positive value, often in the range between 0 and 1.
Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process.
In order to get a better understanding of the differences in learning rate, we decided to create line plots of train and test accuracy for eight different values of learning rate to see their evolution.

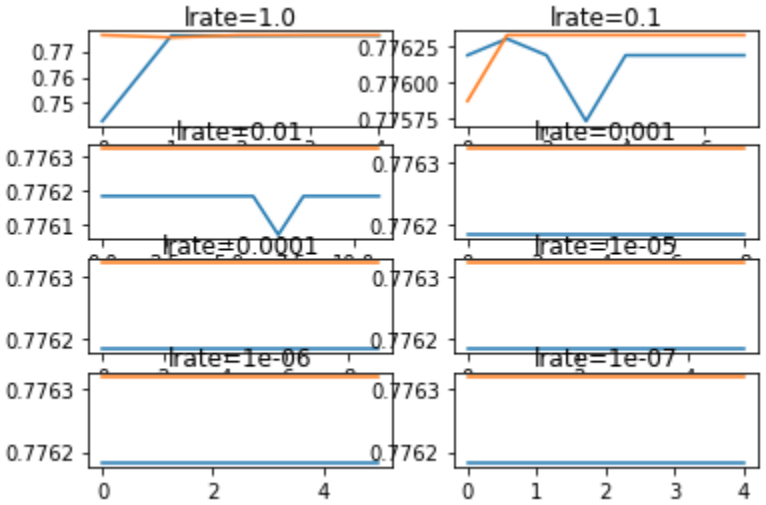


Figure 1: Line plots of train and test accuracy for different learning rates (without class weights)

We observed that the accuracy is not changing for learning rate lower than 1E-03, which means that the optimizer has found a local minimum for the loss. This may be an undesirable minimum. One common local minimum is to always predict the class with the most number of data points. So, we may have to use weights on the classes.

Baseline classifier

A way to gauge how the accuracy we obtained is good is to compare it against a baseline figure.
We see that is we set all sentences to 'NO LABEL' then we have correctly identified the 77.6% of them. So, we would expect that any analytic approach for a classification problem should be better than a random guessing approach.

Based on ***Pie Chart 1*** we had indications that our dataset was imbalanced because the precision was relatively high, but the recall and wasn't as high as we might like (e.g. for class 0, precision = .726 and recall=.308). Classifiers often face challenges when trying to maximize both precision and recall, which is especially true when working with imbalanced datasets. So, in order to avoid the local minimum and due to the fact that we have an imbalanced dataset, we decided to use weighting on the classes.

Class Weights

Our goal is to identify arguments in sentences. The problem is that we have an imbalanced dataset to work with, approximately 22% are arguments, so we would want to have our classifier heavily weight these few sentences that are available. We provided a weight (or bias) for each output class in order to cause our model to "pay more attention" to sentences from an under-represented class. Then, we re-trained and evaluated our model with class weights to see how that affected our predictions. We observed that using class weights the accuracy and precision were lower because there are more false positives regarding the first two argument classes, but conversely the recall was higher because the model also found more true positives. Despite having lower accuracy, this model has higher recall and F1 score, especially in the evidence class.

We also saw from the confusion matrix* that the number of correct predictions shown in the confusion matrix for the first two argument classes were more than 10% higher when using class weights.

	Claims	Evidences
Without class weights	61/198 = 30%	41/297 = 13%
With class weights	93/198 = 46%	126/297 = 42%

The fact that we only want to correctly predict the arguments made us to use class weights going forward.

Although the accuracy in the test set and the precision drop significantly, the main target of our analysis is to correctly predict as many arguments as we can. We observe that with the addition of class weights, the recall and the F1-scores for the first two argument classes are in general a lot better. That's why in the confusion matrix we found more evidences and claims. So, from now on we will use class weights when fitting our model.

Optimizers and Learning Rate

Using the best-chosen CNN model we applied the following:

- Weights on classes
- Optimizers (Adams and SGD)
- Checking for different learning rates (1e-2, 1e-3, 1e-4, 1e-5).

We will first analyze the impact of optimizers and learning rate using the model that takes into account the class weights we computed. We first tested what happens with different optimizers. We observed that the Adam optimizer performed better than the Stochastic Gradient Descent (SDG), due to smaller loss and bigger accuracy in the test set, as well as better F1-scores. It also helped us identify more arguments as we saw in the confusion matrix, so this is the one we will use from now on.
We can now investigate the dynamics of different learning rates for the Adam optimizer on the test accuracy of the model.

f1-score	lr = 1e-2	lr = 1e-3	lr = 1e-4	lr = 1e-5
0	.3723	.3933	.3860	.3876
1	.3279	.3836	.3811	.3815
2	.8339	.7917	.7888	.7900
accuracy	.7135	.6679	.6647	.6661
weighted avg	.7247	.7013	.6980	.6992

While the learning rate was decreased, the accuracy dropped also. Nevertheless, in our experiment we care more about the F1-scores in the first two classes, where the model was able to learn our problem well with learning rate equal to 1E-3. After that value, the scores remain almost stable. So, this is the one we will use from now on. This can also be seen from the Figure 1 from Figures in Appendix.

Recurrent neural networks (RNNs)

RNN is a class of neural networks that is naturally suited to processing time-series data and other sequential data. More specifically, RNN is a neural network designed for analyzing streams of data by means of hidden units. Because of their internal memory, RNN's can remember important things about the input they received, which allows them to be very precise in predicting what's coming next. This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather and much more. Recurrent neural networks can form a much deeper understanding of a sequence and its context compared to other algorithms.

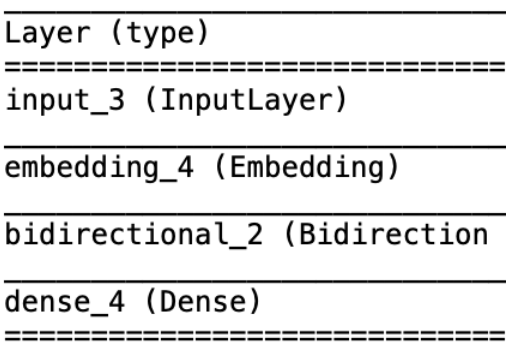
In general, RNNs are provided with the input samples which contain more interdependencies. Some of the most popular recurrent architectures in use include long short-term memory (LSTM) and gated recurrent units (GRUs).

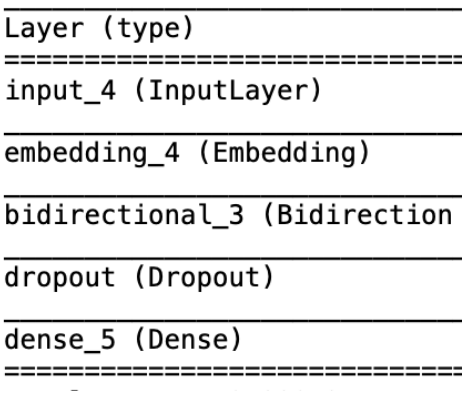
Long Short-Term Memory (LSTM) networks are a type of recurrent neural network capable of learning order dependence in sequence prediction problems. LSTMS are recurrent networks whose inputs are not fixed but rather constitute an input sequence can be used to transform an input sequence into an output sequence while taking into account contextual information in a flexible way.

For the current project, in order to meet the target, we decided to implement Bidirectional LSTM models. Bidirectional LSTM models are an extension of traditional LSTMs that can improve model performance on sequence classification problems. In problems where all timesteps of the input sequence are available, Bidirectional LSTMs train two instead of one LSTMs on the input sequence. The first on the input sequence as-is and the second on a reversed copy of the input sequence. This can provide additional context to the network and result in faster and even fuller learning on the problem.

Model Structure

We used a pre-trained word embedding (GloVe) to capture similarities between words and semantics of word sentences. After that, we applied a RNN architecture to encode and leverage the information from the input text sequence. Specifically, we used the two below version of layers sequencing:





It is obvious, that both RNNs share the same architecture with the different of adding a dropout layers in the second one, in order to avoid overfitting. Furthermore, the addition of the dropout layer, gave us the flexibility to apply hyper-parameters tuning by changing the dropout rate.

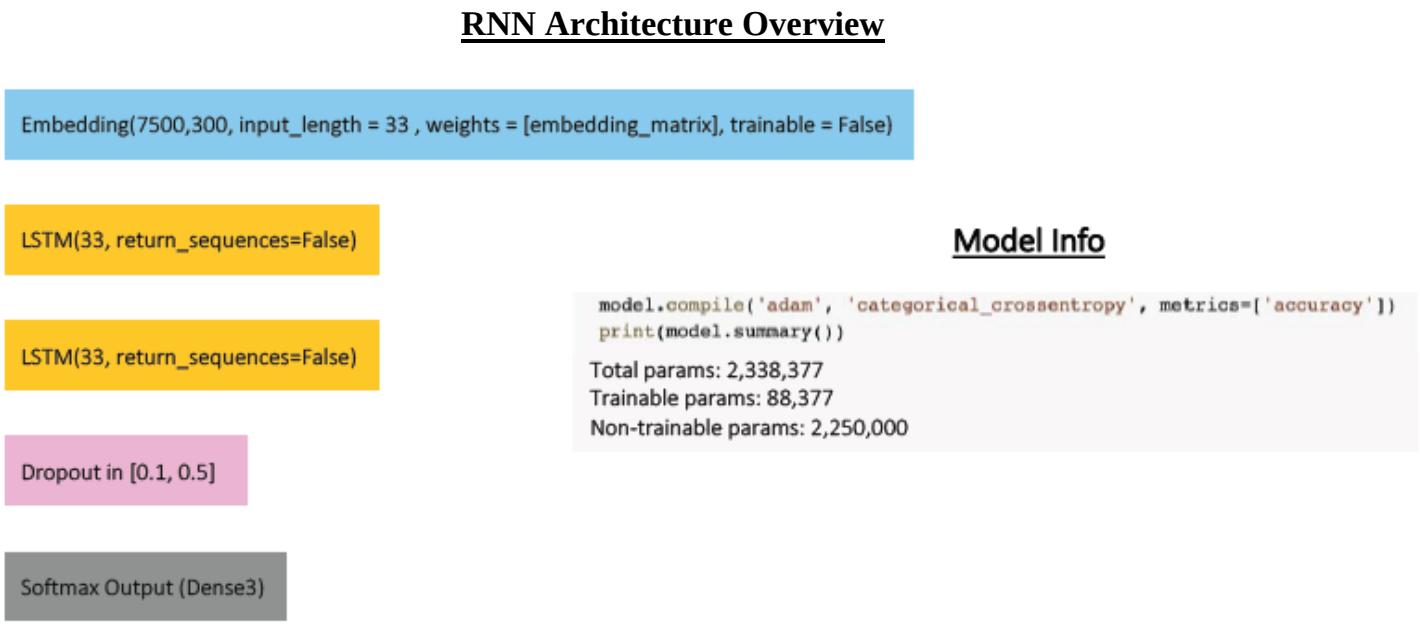


Figure 1: RNN Architecture

A simple RNN is a sequence of layers, and every layer of a RNN transforms one volume of activations to another through a differentiable function. We used four types of layers to build RNNs architectures: Embedding Layer, a double LSTM Layer, Dropout Layer and Output Layer. We will stack these layers to form a full RNN architecture.

Hyper-Parameters

In order to improve the performance of our model we did the following:

1. Fine tune Hyper-parameters: Hyper-parameters are the variables which are set before training and determine the network structure & how the network is trained. (e.g. batch size, number of epochs).
2. Use Dropout Layer

The application of the above hyper parameters lead to produce different versions of RNN models and therefore we need to check their accuracy in order to pick the one that is performing the best. In the following table we present the 4 different versions of RNN models along with their f1-score:

RNN Model	Dropout Rate	Epochs	Batch Size	Early Stopping	f1-score
rnn_model1	✗	30	128	✓	0.7311
rnn_model1	✗	30	128	✗	0.7551
rnn_model1	✗	60	128	✗	0.7492
rnn_model_withdr1	0.5	60	256	✓	0.7845
rnn_model_withdr2	0.1	60	256	✓	0.7917

Table: RNN Models comparison

From the above Table we conclude that the RNN Model (*rnn_model_withdr2*) with drop out layer rate 0.1 is the best performing model comparing with the other 4 models. We trained our model for epochs = 60 and batch size = 256, utilizing early stopping to prevent overfitting. For the current report in order to meet the size requirements and

be less verbose we will present results and comments only for the best chosen model. **However, in the code file we included all processes for the 5 RNN models.**

To confirm the performance of the chosen RNN model, we proceed with its detailed evaluation on the test dataset, in order to check the training categorical cross entropy (loss).

- loss: 0.5463 - accuracy: 0.7917

From now on we will use *rnn_model_withdr2* model to apply the following:

- Weights on classes
- Optimizers (Adams and SGD)
- Checking for different learning rates (1e-2,1e-3,1e-4,1e-5).

Model Parameters

The model parameters of the chosen RNN model can be found on the below figure generated from the output of summary function on the chosen RNN model.

```
Model: "functional_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_6 (InputLayer)         [(None, 33)]              0
_________________________________________________________________
embedding_4 (Embedding)      (None, 33, 300)           2250000
_________________________________________________________________
bidirectional_5 (Bidirection (None, 66)                88176
_________________________________________________________________
dropout_2 (Dropout)          (None, 66)                0
_________________________________________________________________
dense_7 (Dense)              (None, 3)                 201
=================================================================
Total params: 2,338,377
Trainable params: 88,377
Non-trainable params: 2,250,000
_________________________________________________________________
```

Figure: Summary of chosen RNN Model

Class Weights

Following the same line of thought as we did for building the CNN model, in order to avoid the local minimum and due to the fact that we have an imbalanced dataset, we use weighting on the classes also for the RNN model.

We observed that using class weights the accuracy and precision were lower because there are more false positives regarding the first two argument classes, but conversely the recall was higher because the model also found more true positives. Despite having lower accuracy, this model has higher recall and F1 score, especially in the evidence class.

We also saw from the confusion matrix* that the number of correct predictions shown in the confusion matrix for the first two argument classes were almost 10% higher when using class weights.

	Claims	Evidences
Without class weights	67/198 = 33%	88/297 = 29%
With class weights	82/198 = 41%	104/297 = 35%

The fact that we only want to correctly predict the arguments made us to use class weights going forward.

Although the accuracy in the test set and the precision drop significantly, the main target of our analysis is to correctly predict as many arguments as we can. We observe that with the addition of class weights, the recall and the F1-scores for the first two argument classes are in general a lot better. That's why in the confusion matrix we found more evidences and claims. So, from now on we will use class weights when fitting our model.

Optimizers and Learning Rate

We follow the same process as applied in the CNN model. We first tested what happens with different optimizers both Adam and Stochastic Gradient Descent (SDG). The respective final results from the model using Adam and SGD optimizers are presented below:

	F1-scores (Adam)	F1-scores (SDG)
0	.3942	.3810
1	.3478	.3500
2	.8499	.8538
accuracy	.7393	.7451
weighted avg	.7418	.7439

We observed that the SGD optimizer performed better than the Adam, due to smaller loss and bigger accuracy in the test set, as well as better F1-scores. It also helped us identify more arguments as we saw in the confusion matrix, so this is the one we will use from now on.

* see Table 2 from Tables in Appendix

We can now investigate the dynamics of different learning rates for the SGD optimizer on the test accuracy of the model. We applied 4 different learning rates and compare them as following:

f1-score	lr = 1e-2	lr = 1e-3	lr = 1e-4	lr = 1e-5
0	.4029	.4029	.4029	.4029
1	.3703	.3708	.3708	.3708
2	.8444	.8447	.8447	.8447
accuracy	.7361	.7366	.7366	.7366
weighted avg	.7413	.7416	.7416	.7416

We observe that the accuracy in the test set increase as the learning rate drops. Nevertheless, we care more about the F1-scores in the first two classes, where the **best results were achieved with learning rate = 1e-3.** After that value, the scores remain almost stable. So, this is the one we will use from now on.

5 Results

Comparison of best RNN and CNN models

In this section we compare the results originated from the best CNN and RNN models. In order to do so we present their classification reports as well as the confusion matrices.

For RNN model:

Classification Report

	precision	recall	f1-score	support
0	0.3942	0.4141	0.4039	198
1	0.3503	0.3939	0.3708	297
2	0.8564	0.8329	0.8445	1718
accuracy			0.7366	2213
macro avg	0.5336	0.5470	0.5398	2213
weighted avg	0.7471	0.7366	0.7415	2213

Confusion Matrix

	CLAIM	EVIDENCE	NO LABEL
CLAIM	82	28	88
EVIDENCE	28	117	152
NO LABEL	98	189	1431

For CNN model:

Classification report

	precision	recall	f1-score	support
0	0.3149	0.5455	0.3993	198
1	0.3370	0.5152	0.4075	297
2	0.9004	0.7421	0.8137	1718
accuracy			0.6941	2213
macro avg	0.5174	0.6009	0.5401	2213
weighted avg	0.7724	0.6941	0.7221	2213

Confusion Matrix

	CLAIM	EVIDENCE	NO LABEL
CLAIM	108	40	50
EVIDENCE	53	153	91
NO LABEL	182	261	1275

Interpretation of confusion matrices

In this particular application, we want the arguments to be correctly predicted. The costs of false positives or false negatives are the same to us. For this reason, it does not matter to us whether our classifier is more specific or more sensitive, as long as it classifies correctly as much arguments as possible.

- False negatives (FN) and false positives (FP) are samples that were incorrectly classified
- True negatives (TN) and true positives (TP) are samples that were correctly classified

Here we have a multi-class classification problem with 3 classes. Thus, we will find TP, TN, FP and FN for each individual class.
For example, if we take class CLAIM, from the CNN confusion matrix, we have the following values:
TP = 108, TN = (153+91+261+1275) =1780, FP = (40+50) = 90, FN = (53+182) = 235

Conclusion

Our final decision for the best model was based on the examination of the f1-score, the accuracy and the confusion matrix as depicted below:

Best Chosen Model	f1-score (Class 0)	f1-score (Class 1)	Accuracy
RNN	.403	.370	.736
CNN	.399	.407	.694

Best Chosen Model	Claims	Evidences
RNN	82/198 = 41%	117/297 = 39%
CNN	108/198 = 54%	153/297 = 51%

We observe that the accuracy in the test set drops significantly for the CNN model. However, given that the main target of our analysis is to correctly predict as many arguments as we can, our conclusions will be based on the f1 – scores of the 2 argument classes (class 0 and class 1) respectively. **From there we identify a higher f1-score regarding class 1 generated by CNN model**. This is also supported from the confusion matrix by the more than 10% evidences identified from CNN model. As an extra criterion, we also used the AUC (Area Under Curve) metric, which is computer via the ROC (Receiver Operating Characteristic) curves. This metric is equal to the probability that a classifier will rank a random positive sample higher than a random negative sample.

ROC curve is useful because it shows, at a glance, the range of performance the model can reach just by tuning the output threshold. It extends to problems with three or more classes with what is known as the one-vs-all approach. Here we have three classes, so we will create three ROC curves.

For each class, we take it as the positive class and group the rest classes jointly as the negative class. So, we will use the Roc curve which will depict:

- Class 0 vs classes 1&2
- Class 1 vs classes 0&2
- Class 2 vs classes 0&1

For better understanding, the arithmetic value of AUC along with visualization of these curves are shown below:

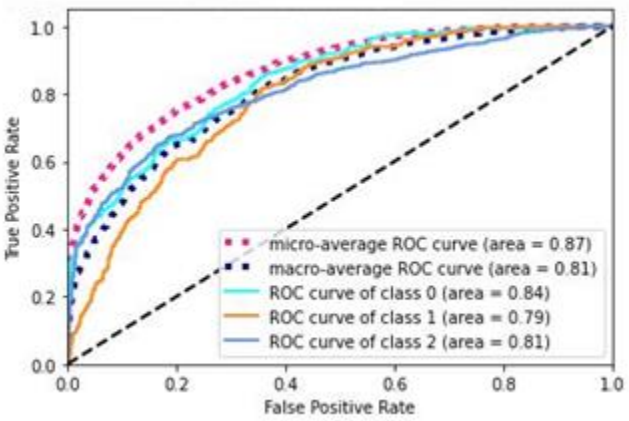
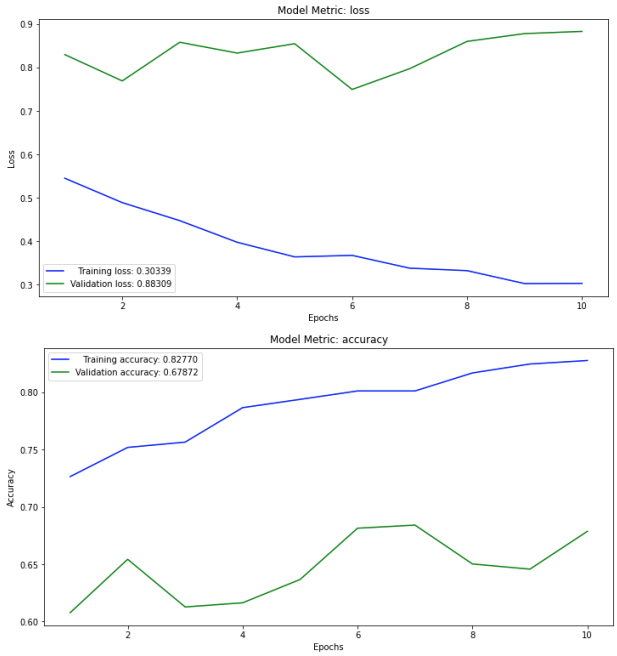


Figure 2: Roc curve for the 3 classes

Comment: We deemed that the zoomed in at the top left version of the Roc curve was not necessary for our evaluation.



We also created loss and accuracy plots of this model as shown above.

Comments for runtime and machines:

We did not face any important issues regarding the performance of the models. Moreover, when added the callbacks the performance time was reduced much more and the models run in some seconds' time. Specific times are included inside the code file.

6 Bibliography

A) WEB SOURCES

- Jason Brownlee, Best Practices for Text Classification with Deep Learning. Available at https://machinelearningmastery.com/best-practices-document-classification-deep-learning/
- 2008 Cambridge University Press, Stemming and lemmatization. Available at https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
- Anon, Keras Conv1D: Working with 1D Convolutional Neural Networks in Keras. Available at https://missinglink.ai/guides/keras/keras-conv1d-working-1d-convolutional-neural-networks-keras/
- Jason Brownlee, 1D Convolutional Neural Network Models for Human Activity Recognition. Available at https://machinelearningmastery.com/cnn-models-for-human-activity-recognition-time-series-classification/
- Christian Versloot, What are Max Pooling, Average Pooling, Global Max Pooling and Global Average Pooling?. Available at https://www.machinecurve.com/index.php/2020/01/30/what-are-max-pooling-average-pooling-global-max-pooling-and-global-average-pooling/
- Britta Bettendorf, Visualizations + Deep Learning: CNN, RNN, GloVe. Available at https://www.kaggle.com/brittabettendorf/visualizations-deep-learning-cnn-rnn-glove
- Jason Brownlee, Deep Convolutional Neural Network for Sentiment Analysis (Text Classification). Available at https://machinelearningmastery.com/develop-word-embedding-model-predicting-movie-review-sentiment/
- Rodrigo Bressan, Building a multi-output Convolutional Neural Network with Keras. Available at https://towardsdatascience.com/building-a-multi-output-convolutional-neural-network-with-keras-ed24c7bc1178
- Simon O'Keefe, Deep Learning and Word Embeddings for Tweet Classification for Crisis Response. Available at https://arxiv.org/ftp/arxiv/papers/1903/1903.11024.pdf
- Sourish Dey, CNN application on structured data-Automated Feature Extraction. Available at https://towardsdatascience.com/cnn-application-on-structured-data-automated-feature-extraction-8f2cd28d9a7e
- Jason Brownlee, When to Use MLP, CNN, and RNN Neural Networks. Available at https://machinelearningmastery.com/when-to-use-mlp-cnn-and-rnn-neural-networks/
- Akshat Maheshwari, Report on Text Classification using CNN, RNN & HAN. Available at https://medium.com/jatana/report-on-text-classification-using-cnn-rnn-han-f0e887214d5f#:~:text=Text%20Classification%20Using%20Convolutional%20Neural,designed%20to%20require%20minimal%20preprocessing.
- Stanford University, CS231n: Convolutional Neural Networks for Visual Recognition. Available at https://cs231n.github.io/convolutional-networks/?fbclid=IwAR1EcFiaqltk2uXZ7GQfmQPhgWwnp3mG4p9pqROzfMJhGw9O3givepvF0Uc
- Rakshith Vasudev, Understanding and Calculating the number of Parameters in Convolution Neural Networks (CNNs). Available at https://towardsdatascience.com/understanding-and-calculating-the-number-of-parameters-in-convolution-neural-networks-cnns-fc88790d530d

- Jason Brownlee, Understand the Impact of Learning Rate on Neural Network Performance. Available at https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

B) OTHER SOURCES

- Perakis G., Deep learning labs (.ipynb files)
- Papageorgiou H., Machine Learning and Content Analytics Slides.

7 Appendix

Queries

Indicative Queries used for SDG13 by Group 5 can be found below:

- Indicator: 13.1
- Query: (("climat*" OR "natural disaster*") W/3 "resilien*")
- Indicator: 13.2.1
- Query: (("climat*") W/3 ("polic*" OR "strateg*" OR "plan" OR "plans" OR "planning"))
- Indicator: 13.3.2
- Query: ("climate") W/3 ("information" OR "awareness" OR "educat*" OR "teach*" OR "learn*")
- Indicator: 13.b.1
- Query: ("climate") AND (("assist*" OR "support*" OR "aid" OR "program*" OR "development*" OR "capacity*") W/3 ("develop* countr*" OR "least developed countr*" OR "small island*"))

```
TITLE-ABS-KEY ( ( "climat*" W/3 "anthropogenic*" ) OR ( "climat*" W/3 "action*" ) OR ( "climat*" W/3 "adapt*" ) OR ( "climat*" W/3 "biodiversity" ) OR ( "climat*" W/3 "carbon*" ) OR ( "climat*" W/3 "change*" ) OR ( "climat*" W/3 "crisis" ) OR ( "climat*" W/3 "deforestati*" ) OR ( "climat*" W/3 "desertificati*" ) OR ( "climat*" W/3 "ecolog*" ) OR ( "climat*" W/3 "environment*" ) OR ( "climat*" W/3 "GHG" ) OR ( "climat*" W/3 "global change" ) OR ( "climat*" W/3 "greenhouse gas*" ) OR ( "climat*" W/3 "hazard*" ) OR ( "climat*" W/3 "reforestati*" ) OR ( "climat*" W/3 "variabilit*" ) OR ( "climat*" W/3 "warming" ) OR ( "climat*" W/3 "water stress" ) OR ( ( "climate" ) AND ( ( "Paris" ) W/3 ( "agreement" OR "COP21" ) ) ) OR ( ( "climate" ) AND ( ( "Kyoto" ) W/3 ( "protocol" ) ) ) OR ( "climate action*" ) OR ( "Climate Effect*" ) OR ( "Climate Model*" ) OR ( "Climate Variability" ) OR ( "Climate Variation*" ) OR ( "climate-driven" ) OR ( "Climatology" ) OR ( "eco-innovation*" ) OR ( "environmental change*" ) OR ( "Environmental Impact" ) OR ( "Global Climate" ) OR ( "global warming" ) OR ( "Greenhouse Effect*" ) OR ( "Green-house Effect*" ) OR ( "Greenhouse Gas*" ) OR ( "Green-house Gas*" ) OR ( ( "sea level*" ) AND ( "chang*" OR "rising" ) ) ) ) AND ( PUBYEAR > 2015 AND PUBYEAR < 2020 )
```

Tables

Table 1 : Confusion Matrices for CNN

Without class weights

	CLAIM	EVIDENCE	NO LABEL
CLAIM	61	9	128
EVIDENCE	7	41	249
NO LABEL	16	28	1674

With class weights

	CLAIM	EVIDENCE	NO LABEL
CLAIM	93	39	66
EVIDENCE	30	126	141
NO LABEL	119	192	1407

Table 2: Confusion Matrices for RNN

Without Class Weights

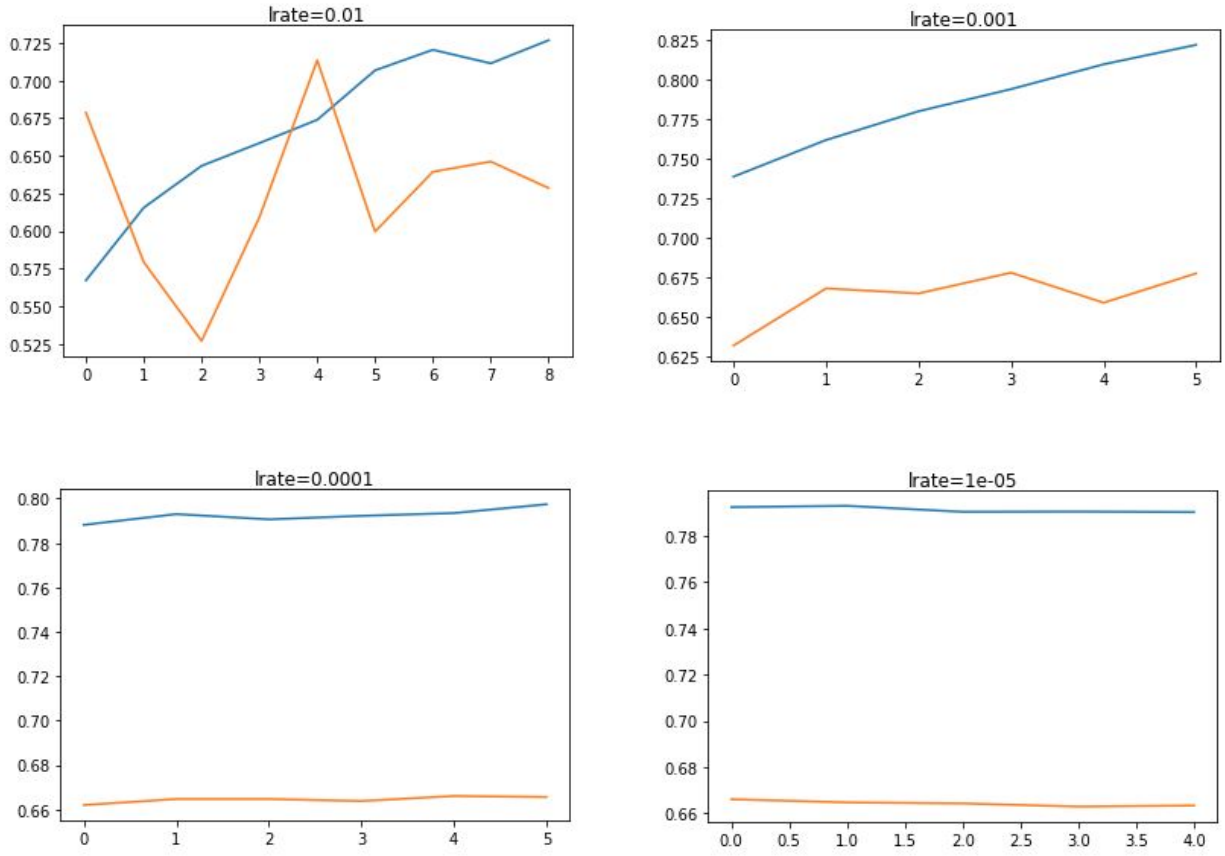
	CLAIM	EVIDENCE	NO LABEL
CLAIM	67	17	114
EVIDENCE	23	88	186
NO LABEL	47	83	1588

With Class Weights

	CLAIM	EVIDENCE	NO LABEL
CLAIM	82	31	85
EVIDENCE	34	104	159
NO LABEL	102	166	1450

Figures

Figure1: Line plots of train and test accuracy for different learning rates (with class weights)



We can see that the model was able to learn the problem well with the learning rates 1E-2 and 1E-3, although successively slower as the learning rate was decreased. With the chosen model configuration, the results suggest a moderate learning rate of 0.01 results in good model performance on the train and test sets.

Glossary

Accuracy	It determines the overall predicted accuracy of the model. It is the proportion of correct classifications. We care to predict better than pure chance.
Batch size	The number of training examples in one forward/backward pass. In general, larger batch sizes result in faster progress in training, but don't always converge as quickly. Smaller batch sizes train slower, but can converge faster. And the higher the batch size, the more memory space we need.
Early Stopping	We build network and train it until validation loss reduces
Epochs	The number of times that the model is exposed to the training dataset. One epoch equals one forward pass and one backward pass of all the training examples. In general, the models improve with more epochs of training, to a point. They'll start to plateau in accuracy as they converge.
Overfitting	Overfitting means that the model is too complex for the data set. It yields a model that matches the model extremely closely, but may be unable to accurately predict what occurs between the data points, because it adds trends to the data set that don’t actually exist. Overfitting affects the ability of the model to perform well for unseen data, which is known as a generalization.