NEURAL NETWORKS CODING PROJECT REPORT

Professor: D. Katsaros

Neuro Fuzzy Computing: 2023-2024

Department Of Electrical & Computer Engineering

University of Thessaly, Volos

Pisxos Vasileios: 03175

Valsamis Georgios: 03259

{vpischos, gvalsamis} @e-ce.uth.gr

Contents

ABSTRACT

Text classification is the process of assigning predetermined categories to open-ended text documents using artificial intelligence and machine learning (AI/ML) systems. Many organizations have large document archives and business workflows that continually generate documents at scale like legal documents, contracts, research documents, user-generated data, and email. Text classification is the first step to organize, structure, and categorize this data for further analytics. It allows automatic document labelling and tagging. For our project we were given a dataset containing approximately 10.000 articles that belong in 2 levels of categories. Our task was to preprocess and manipulate the dataset so that it can be used to feed a model responsible for classifying the articles into the correct categories. In this report we will analyse the architecture of our model, the preprocessing of our data, the results of the classification as well as the steps we followed to get to those results.

INTRODUCTION

To better explain our thought process and the steps we took to complete the project we will split our report into chapters. First chapter will be about the actions we took to preprocess the dataset so that we can use the correct information to feed our model. Second chapter will be about the architecture of our model and how we decided on it. Third and final chapter will be about the results of our project. We will show testing metrics that show the model's generalization degree.

We used Python and run our code on Google Collab notebooks for the whole coding project.

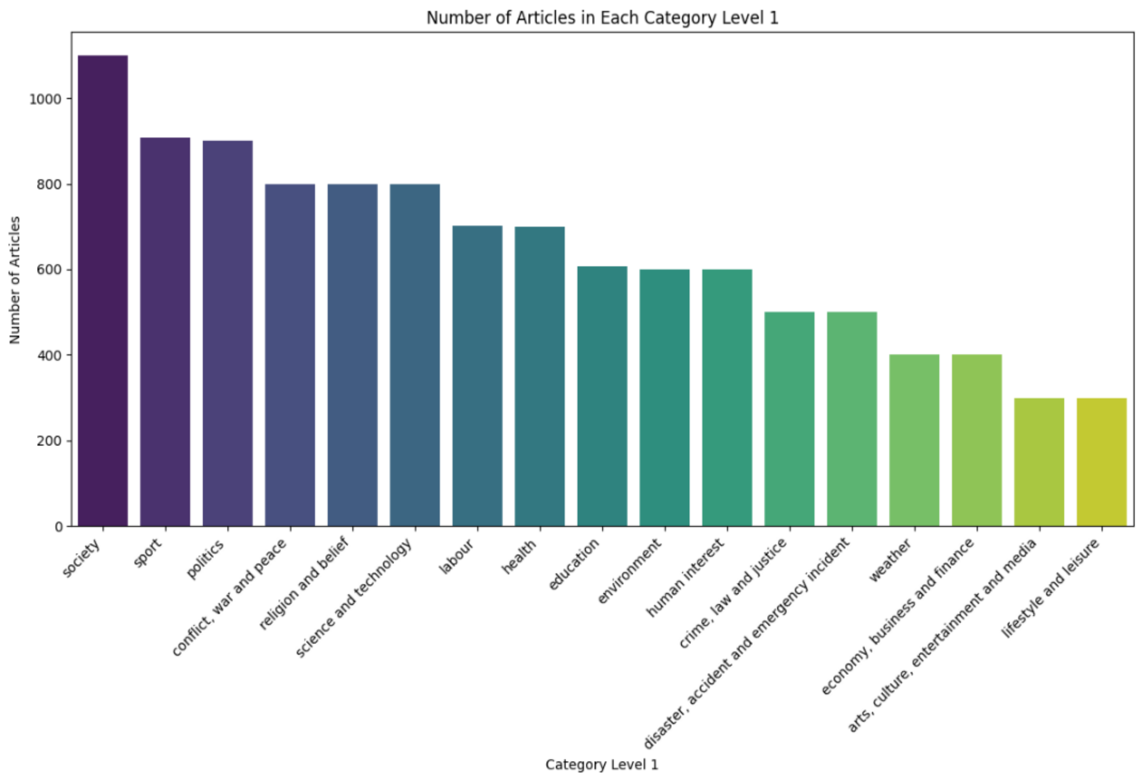
CHAPTER ONE – DATASET ANALYSIS AND PREPROCESSING

1) Choosing important information

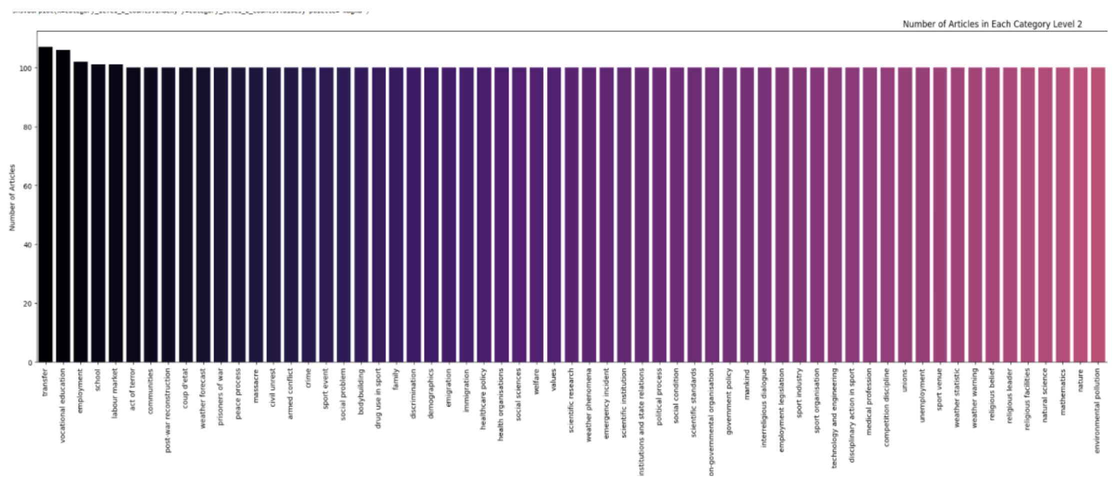
The dataset that we were given consists of 10.917 articles that belong in 17 primary and 109 secondary categories. Each article tuple consisted of these columns of information that we could use: { (data_id, id, date, source, title, content, author, url, published, published_utc, collection_utc, category_level_1, category_level_2 }. Out of these we decided that the useful information that we would use for our classification was: ***id, source, title and content,*** and the target outputs: ***category_level_1 and 2***. All the other columns were redundant in our opinion, so we got rid of them.

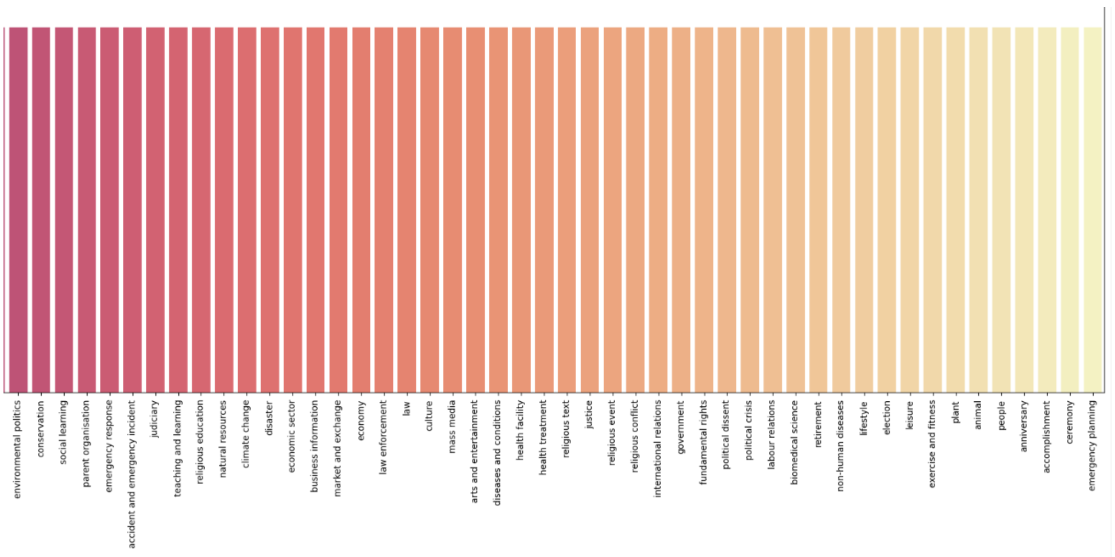
2) Dealing with data imbalances

Data imbalances are a common issue when dealing with classification problems. A simple explanation of this problem is that one or more classes have way more training samples that others. If not taken into consideration this problem can cause the model's testing accuracy to be poor because it will train and fit well only on the classes that have the most training samples. Let's see what we are dealing with in our dataset.



As we can see from the plot above, we have major imbalances for the level 1 categories. As we see there are over 1.000 articles that belong to the category society and only 300 articles for the categories: “arts, culture, entertainment and media” and “lifestyle and leisure”. Now let’s see the situation for level 2 categories.

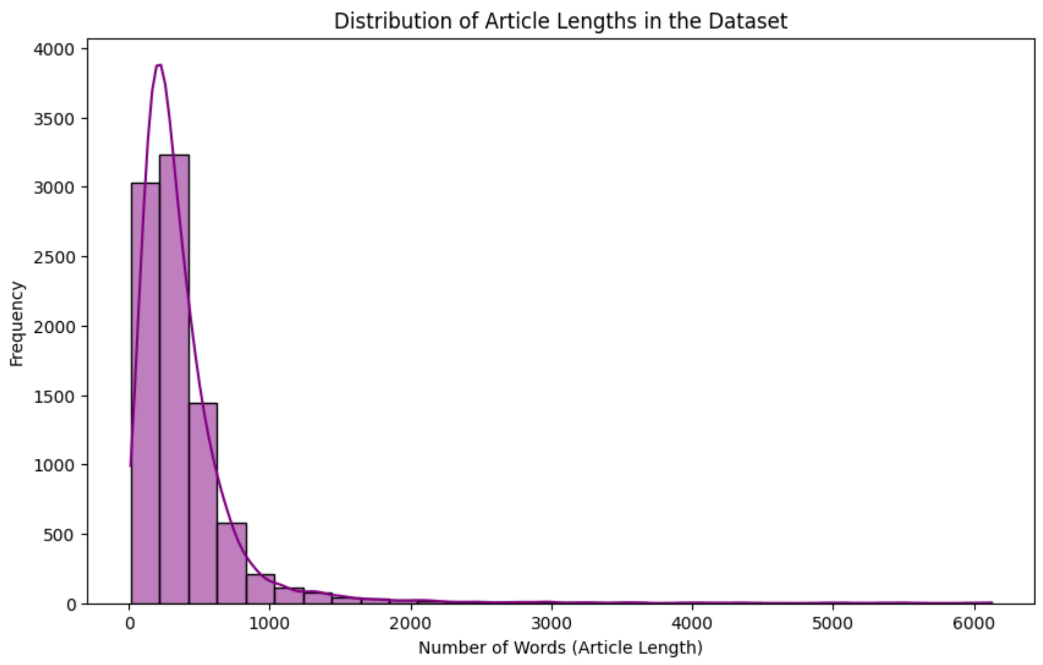




For the level 2 category we had to cut the diagram in two pieces but still we can see clearly see that we have no problem with data imbalances. Almost every level 2 category is equally represented in our dataset.

3) Article Length Distribution

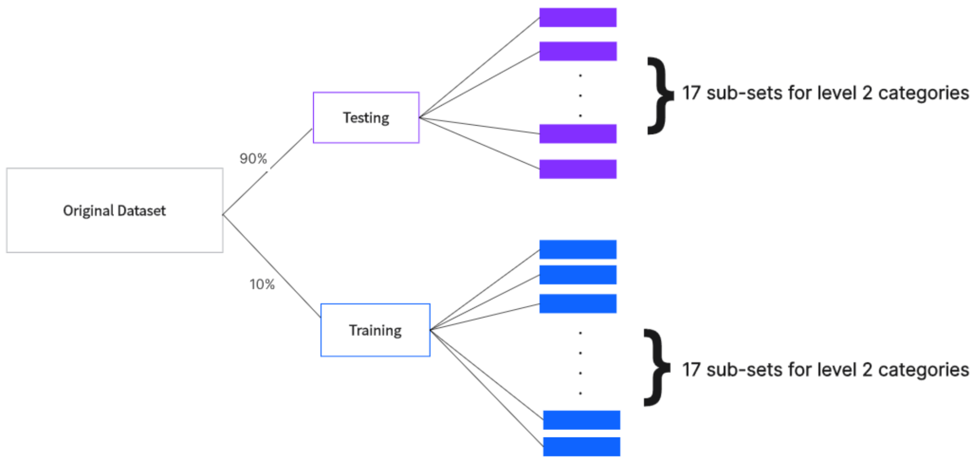
The length of each article is different and especially when combined with the other columns we thought were useful, that’s when things get a little out of hand. The truth is that even though the lengths are way different from each other we must feed our model with data that is strictly the same length. To do that we should decide what will be the maximum length of each training sample and then either truncate a sample if it’s longer or fill it with zeros till it reaches that maximum length.



We decided to choose a length that fully covers 90% of all sentences so that we include all the important information and try to save some memory.

4) Dataset Splits

We split our dataset into training and testing data. To have enough information to train the model we kept 90% of the original dataset for training and 10% for testing. After splitting our dataset into training and testing we split it again and ended up with 17 sub-datasets that were used as training and testing data for the level 2 categories. The diagram below shows the whole procedure.


Original Dataset
90%
10%
Testing
Training
17 sub-sets for level 2 categories
17 sub-sets for level 2 categories

All the sub-sets were saved in our personal cloud storage with names according to their belonging level 1 category. We performed a stratified split meaning that we made sure that the proportions of samples between the classes remained the same.

5) Data Preprocessing

The format that our dataset was given in (.csv) is not suitable for an input to a classifier. The data needs to go through many processing steps before it's usable. Also, the target output needs to be encoded to be used in our model. We will present all the techniques we followed to achieve those results for our dataset in the order that we followed them.

- Stop word Removal

  This process is used to remove all the stop words and punctuation marks which are not useful for our classification task. We created a set of these from *ntlk* library and proceeded to exclude them from our data.

- Lemmatization

  Lemmatization goes beyond truncating words and analyses the context of the sentence, considering the word's use in the larger text and its inflected form. After determining the word's context, the lemmatization algorithm returns the word's base form (lemma) from a dictionary reference.

This technique effectively handles different grammatical categories and tenses, providing a more accurate language representation. For example:
“Saw” would return as “see” or “saw” depending on the context of the word (i.e., whether it is a noun or verb in the sentence).
“Ponies” would return “pony.”
“Requirement” and “required” would return separate words.

- Column merging

  We already talked about which columns we considered important for our classification problem. In this step we merged all the important columns so that we can form a continuous stream of information that we can work with.

- Tokenization

  Tokenization, in the realm of Natural Language Processing (NLP) and machine learning, refers to the process of converting a sequence of text into smaller parts, known as tokens. These tokens can be as small as characters or as long as words. One example could be if our sentence is: “I just finished my coding project” then the tokenized version of it would be: [’I’, ’just’, ’’finished‘, ’my’, ’coding’, ‘project’]. For our tokenization we used the function simple_preprocess from the library genism This function is used to preprocess a list of text documents by tokenizing them, lowercasing, and removing short or non-alphanumeric tokens. The result is a list of tokenized and cleaned text documents, which can then be used for further NLP tasks such as training machine learning models.

- Vectorization

  Vectorization in Natural Language Processing (NLP) is a method used to convert text data into a numerical representation that Machine Learning algorithms can understand and process. We used the ’Tokenizer’ object from ’tensorflow.keras’ library. After the fitting, the ’Tokenizer’ has a vocabulary to match each word to a unique integer index. We performed this matching with the ’texts_to_sequences function that the ’Tokenizer’ provides.

- Padding

  We already discussed this solution above on “article length distribution”. We used ’pad_sequences’ function from ’tensorflow.keras’

- Output label mapping

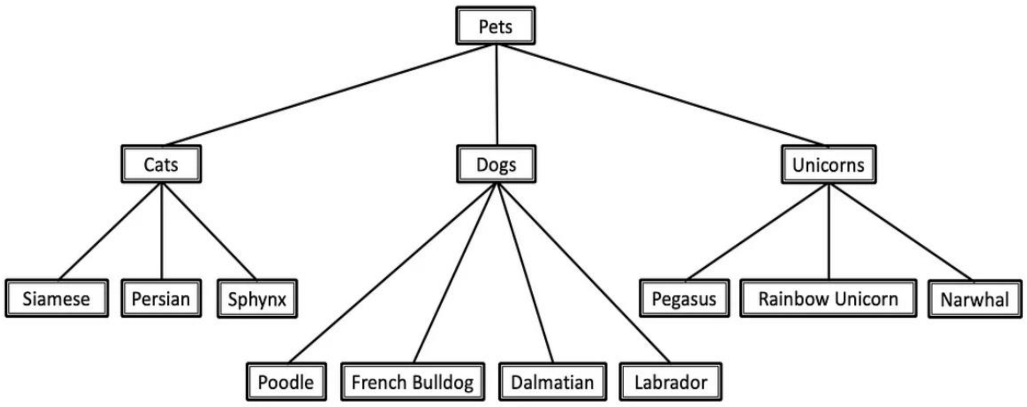
  We replaced our 17 categories with numerical values from 0 to 16 so that we could get the results from our model depending on these numbers. We also did the same thing for the level 2 categories by assigning values starting from 0 to every sub-category of a main category. Now every category is assigned to an integer. The loss function that we use later on is called: “sparse_categorical_crossentropy” and the "sparse" in Sparse Categorical Cross entropy refers to the fact that the true labels are provided as integers (class indices) rather than one-hot encoded vectors.

CHAPTER TWO – PRESENTATION OF OUR ARCHITECTURE

1) Our research

To end up using the model we decided to use first we had to identify the type of problem we were dealing with and do some extensive research on the topic. The problem we had to face is called a hierarchical classification problem. While researching the term we found two articles that really helped us land on an accurate model and architecture. We found a set of 4 articles from this writer: https://medium.com/@noa.weiss that were helpful on the topic of multy level classification. Also, we got a lot of help from this report on a hierarchical multy level classification problem that was written by a team of researchers from “Data Science Lab at Ryerson University, Toronto, Canada”.
We attach the link here: https://arxiv.org/pdf/2109.01084.
Specifically, from all the research we did we ended up using Local Classifiers. We used Local Classifiers Per Parent Node (LCPN), a system that consists of two levels of classifiers. Essentially, we use one

classifier for every class that is divided into sub-classes. The first level decides the level 1 category of the input and then based on that category it sends it to the corresponding level 2 classifier that decides on the level 2 category of the input. The schema bellow is an example of that concept taken from the set of articles we referenced above.



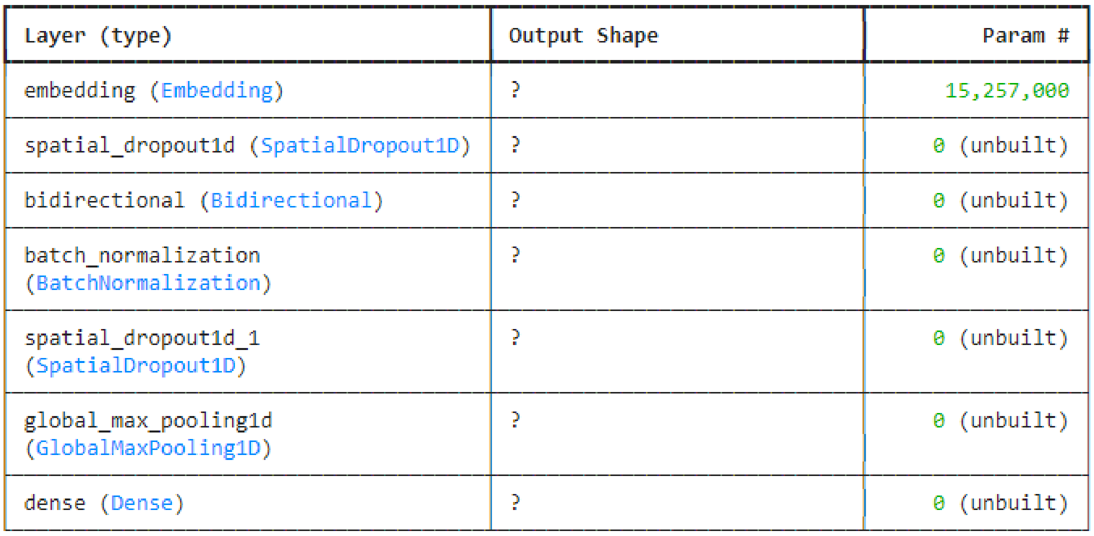
2) Drawbacks of our architecture

The main drawback of the architecture that we decided to use is that because there are two levels of classifiers the final accuracy of the project is dependant of the level 1 accuracy. That means that every wrong prediction that we get from the level 1 classifier is automatically 100% wrong for the whole model and will not give us a result on the level 2 classifiers because the input will be sent to the wrong classifier.

3) A more detailed analysis of our models

1) Level 1 Model

The first level classifier gets an input sample and produces the probability of the input belonging to each one of the 17 categories. First, we decided to try using CNNs for the first level classifier combining them with a dense layer with 100 units. As we were not satisfied with the accuracy percentage of this model, we decided to use a bi-directional LSTM layer instead of a CNN an idea that we got from the report we linked in our research above. We noticed significant improvement on our results, but the model was still clearly overfitting. So, we added two dropout layers and removed the dense layer of 100 units as well. This was the result:



Layer (type)	Output Shape	Param #
embedding (Embedding)	?	15,257,000
spatial_dropout1d (SpatialDropout1D)	?	0 (unbuilt)
bidirectional (Bidirectional)	?	0 (unbuilt)
batch_normalization (BatchNormalization)	?	0 (unbuilt)
spatial_dropout1d_1 (SpatialDropout1D)	?	0 (unbuilt)
global_max_pooling1d (GlobalMaxPooling1D)	?	0 (unbuilt)
dense (Dense)	?	0 (unbuilt)

Let's analyse these layers one by one:

- Embedding layer:
  We trained our own Word2Vec embedding layer using the library: *gensim*, and the functions: *build_vocab* and *train* from the same library. This process mapped every word to a vector , and we also set the *trainable* parameter of our embedding to "True" so that the layers fit our data even more.
- Spatial Dropout:
  We used spatial dropout instead of regular dropout because dropout operation resets some elements as zeros randomly and independently, whereas SpatialDropout would randomly replace all values in a specific dimension with zeros, which can benefit the independence among feature maps. From experimenting with different values, we landed on using 0.3 as it was the dropout value that gave us the best results.
- Batch Normalization:
  Batch normalization is a technique that normalizes the activations of a layer within a mini batch during the training of deep neural networks. It operates by calculating the mean and variance of the activations for each feature in the mini-batch and then normalizing the activations using these statistics. It makes the training faster and more stable through

normalization of the layers' inputs by re-centering and re-scaling. It also helped minimize overfitting issues.

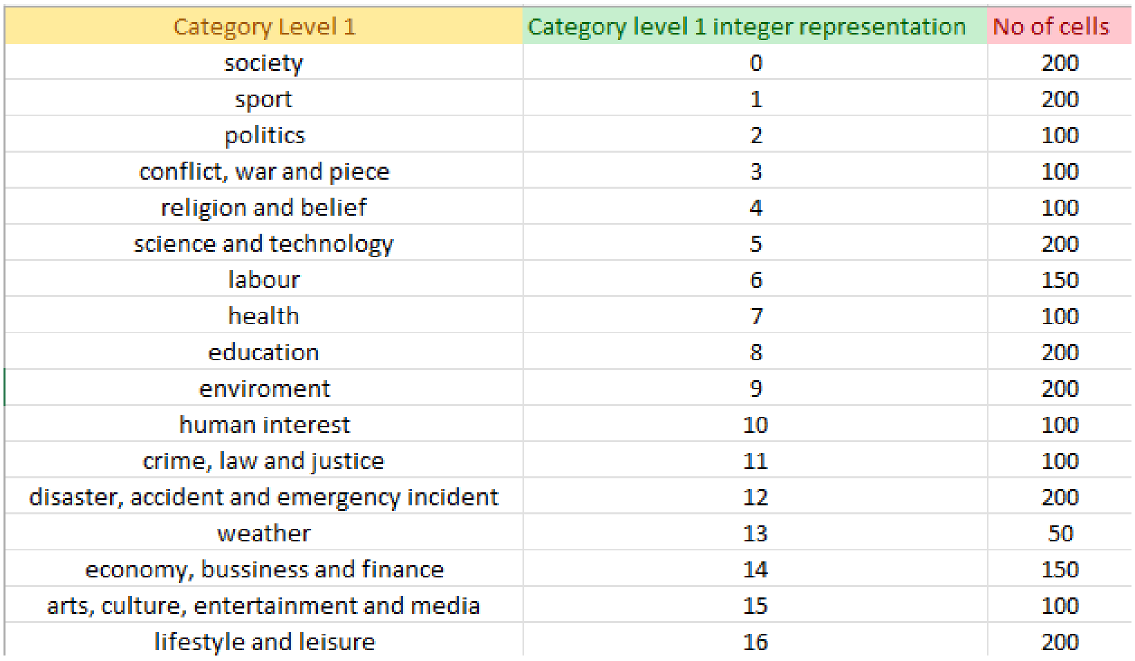
- Global Max Pooling 1D:
  A 1-D global max pooling layer performs downsampling by outputting the maximum of the time or spatial dimensions of the input.
- Dense Output Layer:
  This is the output layer that consists of 17 neurons and classifies the input into one of the 17 level 1 categories using the probabilities produced by the SoftMax activation function.

Now we can talk about the training parameters we used for our level 1 classifier.

For our optimizer we used Adam optimizer with learning rate 0.001. Through vigorous research we found that Adam is the most widely used optimizer in these kind of problems as it combines all the advantages from RMSProp and Adagrad optimizers. We used the function 'compute_class_weight' from 'sklearn' library to resolve the problem of data imbalances that we already talked about. By having custom weights, the model gets penalized more if it makes a mistake on an input that belongs on a class that is underrepresented on our dataset. The maximum number of epochs was set to 50 and we used 'EarlyStopping' by 'keras.callbacks' library to reduce overfitting. This means that if the validation loss was not decreasing after ten consecutive epochs, then the weights that had the least validation loss would be restored. After trial and error, we set the batch size to 128 and our bi-directional LSTM consisted of 50 neurons.

2) Level 2 models

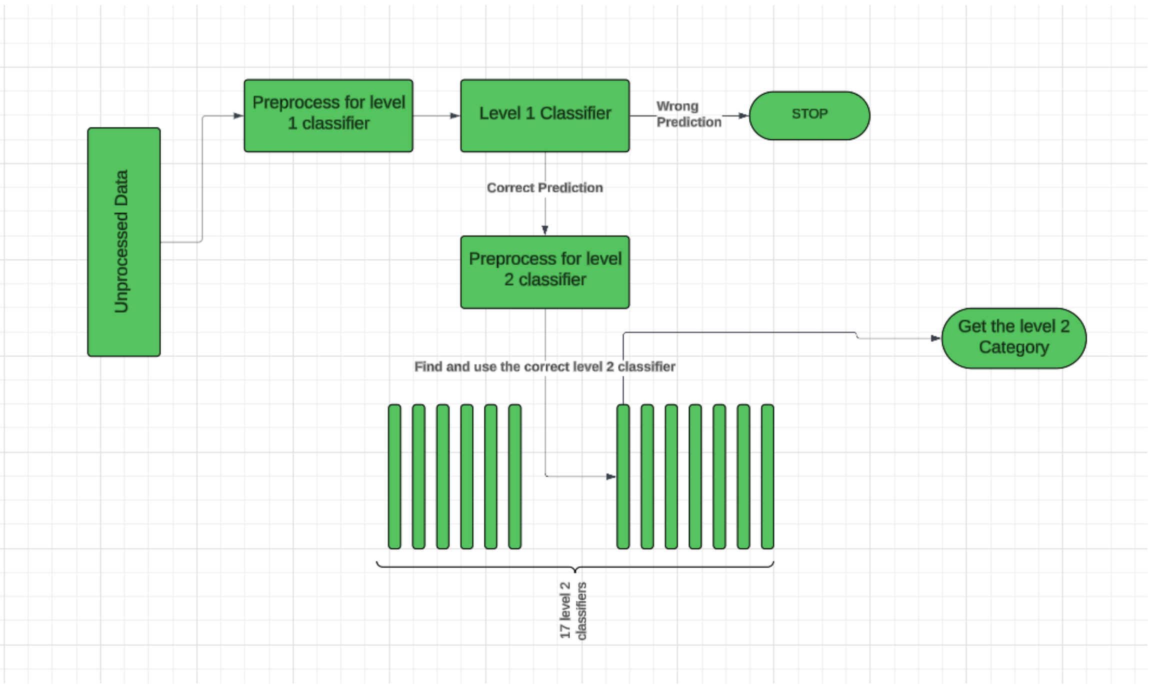
Our level 2 models had the same architecture as our level one models. The only difference in that now we used the value 0.2 for our spatial dropout instead of 0.3 and the batch size was 64 instead of 128. We only changed the number of cells for each one of the 17 categories through trial and error to get the best possible results. Here's a table that shows how many cells each Bi-directional LSTM had depending on the category.



Category Level 1	Category level 1 integer representation	No of cells
society	0	200
sport	1	200
politics	2	100
conflict, war and piece	3	100
religion and belief	4	100
science and technology	5	200
labour	6	150
health	7	100
education	8	200
enviroment	9	200
human interest	10	100
crime, law and justice	11	100
disaster, accident and emergency incident	12	200
weather	13	50
economy, bussiness and finance	14	150
arts, culture, entertainment and media	15	100
lifestyle and leisure	16	200

3) Level 1 and Level 2 connection

To get our results we had to figure out a way to connect level 1 and level 2 classifiers so that they create a cohesive structure that we can use to feed our testing set and get our results. The diagram below shows the whole procedure that takes place from the moment we get our data to the results.



Of course, to achieve all that we have saved to our cloud 17 different level 2 models, one level 1 model, 17 different level 2 Tokenizers and Max lengths and 1 level 1 Tokenizer and Max length. And finally, 17 datasets for training, validation and testing. We will give access to those files later in the report.

CHAPTER THREE – THE RESULTS

1) Level 1 Results

Here we will present the results that our level 1 model produced. Here's a detailed analysis:

Accuracy on training set: 96.59579280705722

Accuracy on test set: 82.32600732600733

Correct predictions on training data: 8541

Correct predictions on test data: 899

Incorrect predictions on training data: 301

Incorrect predictions on test data: 193

Confusion matrix on training data:

```
[[806   1  11   1  12   8   8  14  10   4   6   3   3   0   1   3   0]
[  0 721   0   0   1   0   0   0   1   0   2   0   1   0   0   2   6]
[  6   0 679   7   8   3   8   0   2   2   1  10   2   0   1   0   0]
[  0   0   4 641   1   0   0   0   0   0   0   0   2   0   0   0   0]
[  1   0   4   0 626   0   0   2  11   1   0   1   1   0   0   1   0]
[  2   1   2   0   3 596   1  24   8   3   5   0   0   0   2   0   1]
[  1   0   0   0   0   0 564   1   0   0   0   1   0   0   3   0   0]
[  1   0   0   0   0   2   2 560   1   0   0   1   0   0   0   0   0]
[  1   0   0   0   4   3   3   0 480   0   0   0   0   0   0   0   0]
[  0   0   0   0   0   2   0   0   0 479   3   0   2   0   0   0   0]
[  5   1   0   0   0   0   0   1   0   5 471   0   1   0   0   0   2]
```

```
 [  1   0   1   0   1   1   2   0   0   0   0 399   0   0   0   0   0]
 [  0   0   1   0   0   0   1   0   0   2   0   0 394   5   2   0   0]
 [  0   0   0   0   0   1   0   0   0   0   0   0   0 323   0   0   0]
 [  1   0   0   0   0   0   3   0   0   1   0   0   0   0 319   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0 242   0]
 [  0   2   0   0   0   0   0   0   0   0   0   0   0   0   0   0 241]]
Confusion matrix on test data:
 [[83  1  5  1  5  1  1  6  1  0  5  0  0  0  0  1  0]
 [ 0 81  0  0  2  0  2  0  0  0  1  1  0  0  0  0  4]
 [ 4  0 65  6  6  0  2  0  0  1  0  2  2  0  2  0  0]
 [ 0  0  4 70  2  0  0  1  0  0  0  1  2  0  0  0  0]
 [ 0  0  1  1 69  0  0  1  5  1  0  2  0  0  0  0  0]
 [ 5  0  1  0  0 60  0  2  3  4  2  0  3  0  0  0  0]
 [ 2  1  1  0  0  1 60  0  2  1  1  0  0  0  1  0  0]
 [ 2  0  0  0  1  4  0 61  1  1  0  0  0  0  0  0  0]
 [ 2  2  1  0  4  1  1  0 49  0  0  1  0  0  0  0  0]
 [ 0  0  1  0  0  2  0  0  0 55  0  0  1  0  1  0  0]
 [ 4  1  1  0  1  0  0  1  0  2 48  0  0  0  1  1  0]
 [ 1  0  1  0  0  1  1  0  0  0  1 44  1  0  0  0  0]
 [ 0  0  3  1  0  1  0  0  1  5  0  2 32  5  0  0  0]
 [ 0  0  0  0  0  0  0  0  0  1  0  0  1 38  0  0  0]
 [ 1  0  2  0  0  0  1  0  0  0  0  1  1  0 33  0  1]
 [ 0  1  1  0  1  0  0  0  0  0  2  1  0  0  0 24  0]
 [ 0  2  0  0  0  0  0  0  0  0  1  0  0  0  0  0 27]]
Classification report of training data:
              precision    recall  f1-score   support
```

```
          0     0.9770    0.9046    0.9394       891
          1     0.9931    0.9823    0.9877       734
          2     0.9672    0.9314    0.9490       729
          3     0.9877    0.9892    0.9884       648
          4     0.9543    0.9660    0.9601       648
          5     0.9675    0.9198    0.9430       648
          6     0.9527    0.9895    0.9707       570
          7     0.9302    0.9877    0.9581       567
          8     0.9357    0.9776    0.9562       491
          9     0.9638    0.9856    0.9746       486
         10     0.9652    0.9691    0.9671       486
         11     0.9614    0.9852    0.9732       405
         12     0.9681    0.9728    0.9704       405
         13     0.9848    0.9969    0.9908       324
         14     0.9726    0.9846    0.9785       324
         15     0.9758    0.9959    0.9857       243
         16     0.9640    0.9918    0.9777       243

   accuracy                         0.9660      8842
  macro avg     0.9659    0.9723    0.9689      8842
weighted avg     0.9663    0.9660    0.9658      8842
```

Classification report of test data:

```
              precision    recall  f1-score   support


          0     0.7981    0.7545    0.7757       110
          1     0.9101    0.8901    0.9000        91
```

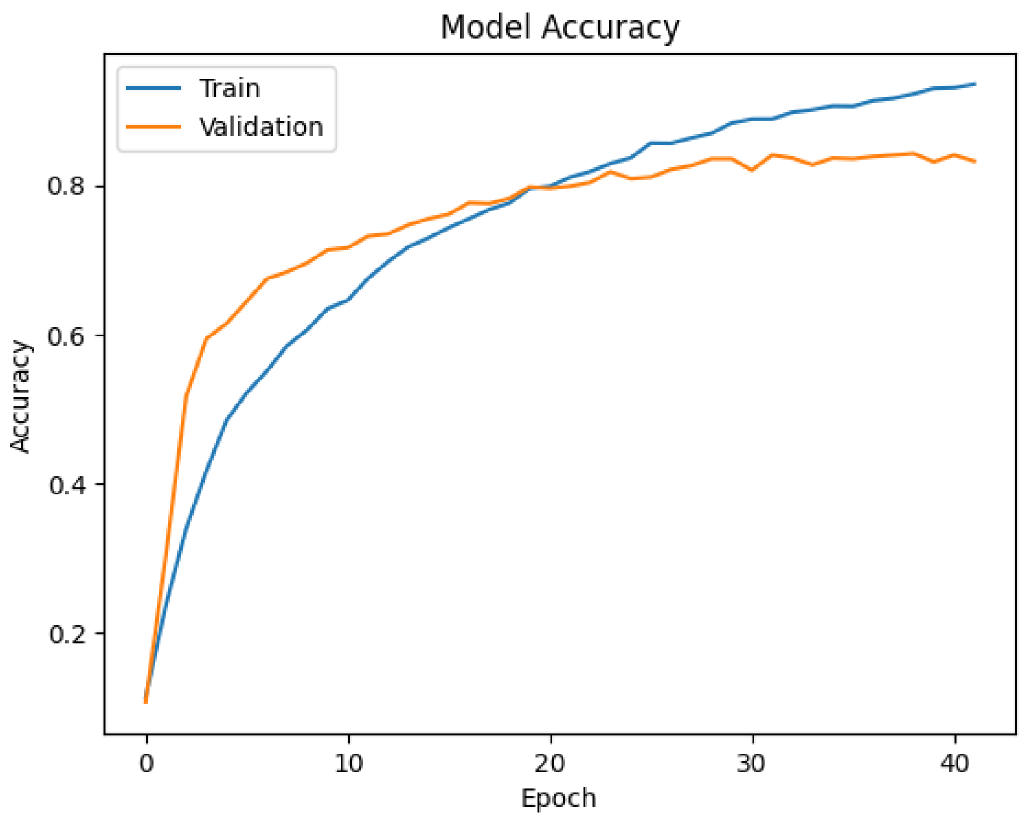
```
           2     0.7471    0.7222    0.7345        90
           3     0.8861    0.8750    0.8805        80
           4     0.7582    0.8625    0.8070        80
           5     0.8451    0.7500    0.7947        80
           6     0.8824    0.8571    0.8696        70
           7     0.8472    0.8714    0.8592        70
           8     0.7903    0.8033    0.7967        61
           9     0.7746    0.9167    0.8397        60
          10     0.7869    0.8000    0.7934        60
          11     0.8000    0.8800    0.8381        50
          12     0.7442    0.6400    0.6882        50
          13     0.8837    0.9500    0.9157        40
          14     0.8684    0.8250    0.8462        40
          15     0.9231    0.8000    0.8571        30
          16     0.8438    0.9000    0.8710        30

    accuracy                         0.8233      1092
   macro avg     0.8288    0.8293    0.8275      1092
weighted avg     0.8244    0.8233    0.8225      1092
```

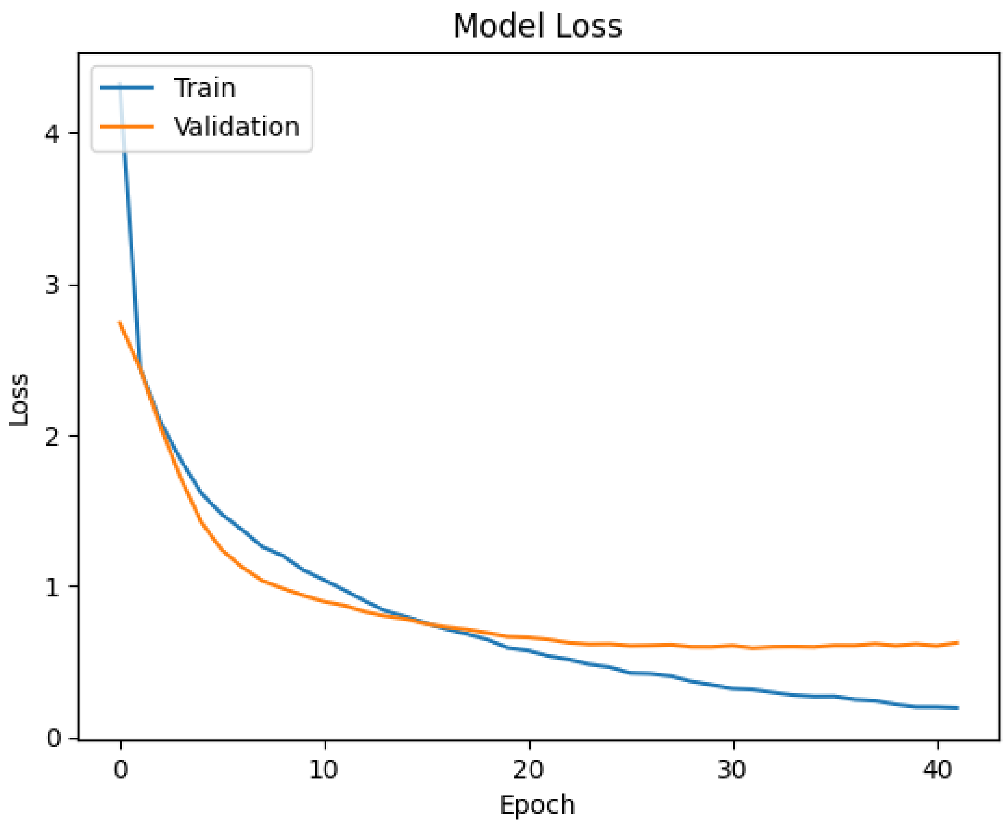
In summary we achieved:
Training accuracy: 96.6%

Testing accuracy: 82.33%

Val loss: 0.5881

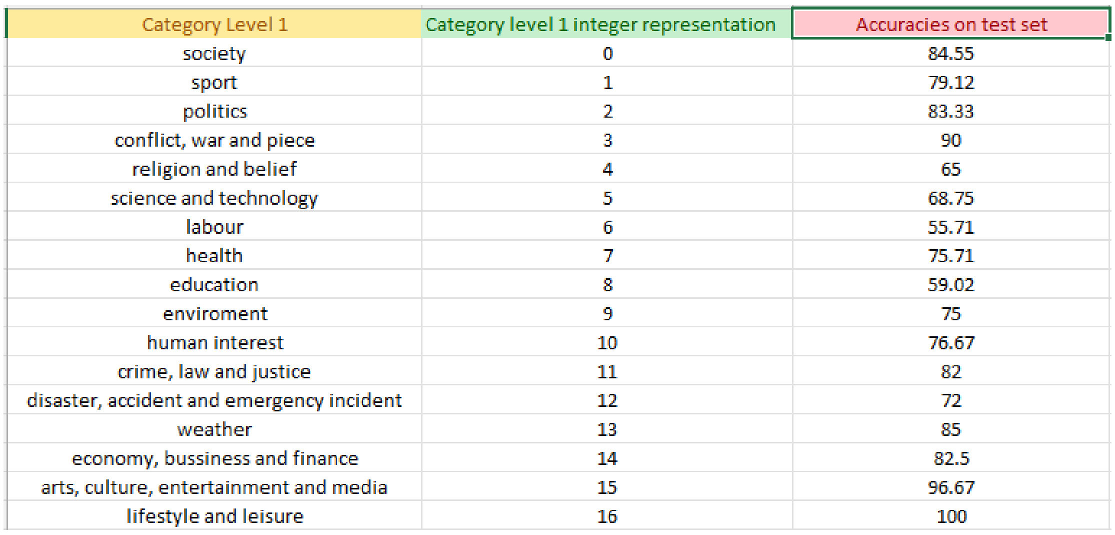
Here are plots about the model accuracy and model loss:


Model Accuracy
Train
Validation
Accuracy
0.2
0.4
0.6
0.8
0
10
20
30
40
Epoch


Model Loss
Train
Validation
Loss
0
1
2
3
4
0
10
20
30
40
Epoch

2) Level 2 Results

Here’s a table with the results of the level 2 classifiers on the testing set.



Category Level 1	Category level 1 integer representation	Accuracies on test set
society	0	84.55
sport	1	79.12
politics	2	83.33
conflict, war and piece	3	90
religion and belief	4	65
science and technology	5	68.75
labour	6	55.71
health	7	75.71
education	8	59.02
enviroment	9	75
human interest	10	76.67
crime, law and justice	11	82
disaster, accident and emergency incident	12	72
weather	13	85
economy, bussiness and finance	14	82.5
arts, culture, entertainment and media	15	96.67
lifestyle and leisure	16	100

The average testing accuracy of all the level 2 classifiers is: 78.30%

3) Final Results

By following the diagram that we showed before we reached a 64.20% accuracy on the combination of the two levels.

ALL THE FILES OF THE PROJECT

In this section we will link all the files that are needed to run our project. These files contain our models, tokenizers, max lengths, and datasets that need to exist for our levels to run. We will also link 3 collaboratories one for the level 1, one for the level 2 and the final.

https://drive.google.com/drive/folders/1nJe7WYJ7mQEpJzo0aw0qVeFnpGbuBT7N?usp=sharing

https://drive.google.com/drive/folders/1dkdc0AwNwrWu9hQH7blflK03kxjM-_Dk?usp=sharing

https://drive.google.com/drive/folders/1I2KooYG10jYJgD_6yqx_K4GECF8t0pSs?usp=sharing https://drive.google.com/drive/folders/1a3kQvYhU6Jx57T7Hhm23_kQaU3Ei0p1E?usp=sharing