UNIVERSITÀ DI PISA

Human Language Teachnologies

News Articles Classification

Tommaso Amarante, Iacopo Bicchierini

Github repository: https://github.com/Bicchie/News-Classification

Contents

Abstract

The problem of text classification is well known and can be addressed in several ways, in this work we approach three commons methods like Transformers, in particular we make a comparison between some members of the BERT family, and like the CNN and Bayes. Furthermore we present a rudimental approach that doesn't need to be trained, the keyword embeddings method. The aim of this paper is to find the best architecture to categorize news articles.

Chapter 1

Introduction

Nowadays online newspapers are becoming more and more popular, both amateur and professional writers can make available tons of text daily. Moreover Internet has a huge amount of unlabelled text, for example the body of web page or archives with old texts. Since each article needs to be categorized in order to be indexed by the search browsers or extract useful metadata, it would be unfeasible for a human to manually assign each category. Our aim is to review the most prominent approaches to automatically give a tag to an article.

Our work focuses on the performance analysis of the three most used architectures for the task of multi-class classification: **Transformers**, **CNN** and **Bayesian classifiers**. In addition to these methods, we thought about a fourth one, that does not need to be trained and that uses keyword extraction and word embeddings, we called it **Keyword with Word Embeddings**.The Transformer architecture, in particular the BERT family, is analyzed in deep in most of its variants.

Chapter 2

Data Collection and Preparation

2.1 Dataset

The choice for the **Dataset** is driven by the need of several articles with various and possibly uncorrelated labels. After a while we found a Dataset on Kaggle containing 190k articles from **Medium**, an online publishing platform that has an hybrid collection of amateur and professional people and publications. Each row in the data is a different article published on Medium, the row has this structure:

- **Title** [string]: The Title of the article.
- **Text** [string]: The Text content of the article.
- **Authors** [list of strings]: The articles Authors.
- **Timestamp** [string]: The publication Timestamp of the article.
- **Tags** [list of strings]: List of Tags associated of the article.

title dataframe index	text The title of the article	authors The URL associated to the article	timestamp The article authors	tags The publication datetime of the article
Mental Note Vol. 24	Photo by Josh Riemer on Unsplash Merry Christmas and Happy Holidays, everyone! We just wanted ever...	['Ryan Fan']	2020-12-26 03:38:10.479000+00:00	['Mental Health', 'Health', 'Psychology', 'Science', 'Neuroscience']
Your Brain On Coronavirus	Your Brain On Coronavirus A guide to the curious and troubling impact of the pandemic and isolation...	['Simon Spichak']	2020-09-23 22:10:17.126000+00:00	['Mental Health', 'Coronavirus', 'Science', 'Psychology', 'Neuroscience']
Mind Your Nose	Mind Your Nose How smell training can change your brain in six weeks — and why it matters. By Ann-...	[]	2020-10-10 20:17:37.132000+00:00	['Biotechnology', 'Neuroscience', 'Brain', 'Wellness', 'Science']
The 4 Purposes of Dreams	Passionate about the synergy between science and technology to provide better care. Check out my new...	['Eshan Samaranayake']	2020-12-21 16:05:19.524000+00:00	['Health', 'Neuroscience', 'Mental Health', 'Psychology', 'Science']
Surviving a Rod Through the Head	You've heard of him, haven't you? Phineas Gage. The railroad worker who survived an explosion that	['Rishav Sinha']	2020-02-26 00:01:01.576000+00:00	['Brain', 'Health', 'Development', 'Psychology', 'Science']

Figure 2.1: Dataset

2.2 Exploratory Data Analysis

This phase consists on understanding the characteristics of the Dataset. We extract the **frequency** in which each tag appears in the articles. The majority of the high frequency tags are related to the computer science field, like ”blockchain”, ”data science”, ”programming”, ”cryptocurrency”, ”machine learning”. We decide 8 *semantically unrelated tags* among the ones that appear more often in the articles:

- **Technology**
- **Politics**
- **Health**
- **Education**
- **Music**
- **Travel**
- **Finance**
- **History**

After this choice we analyze how many articles belong to these categories:

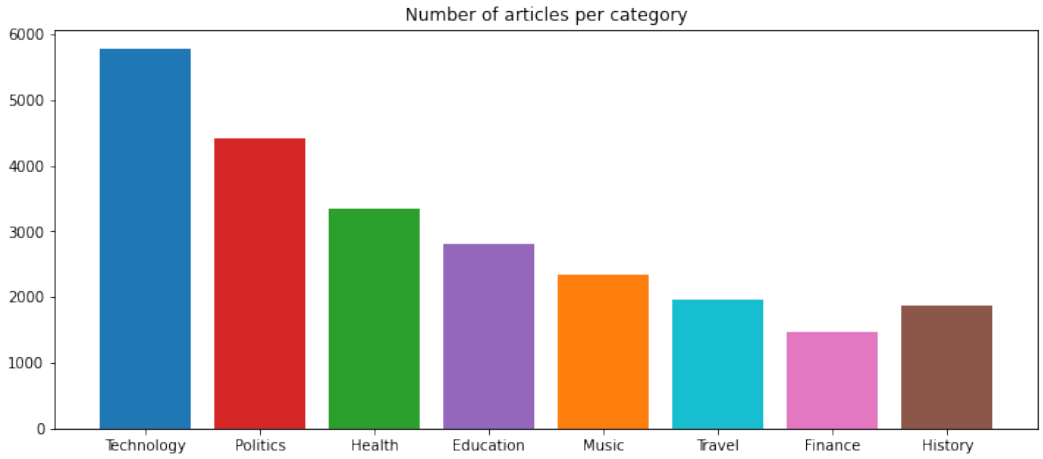

Figure 2.2: Number of Articles per Category

It's clear how the Dataset, after choosing these tags, is **unbalanced** over *Technology* that has almost 6000 articles compared to *Finance* that has only 1500. Indeed in Section 2.3 we deal with this problem.

We find also the average length of the articles per category, discovering how in this case, the texts have similar lengths with a spike in the *History* category.

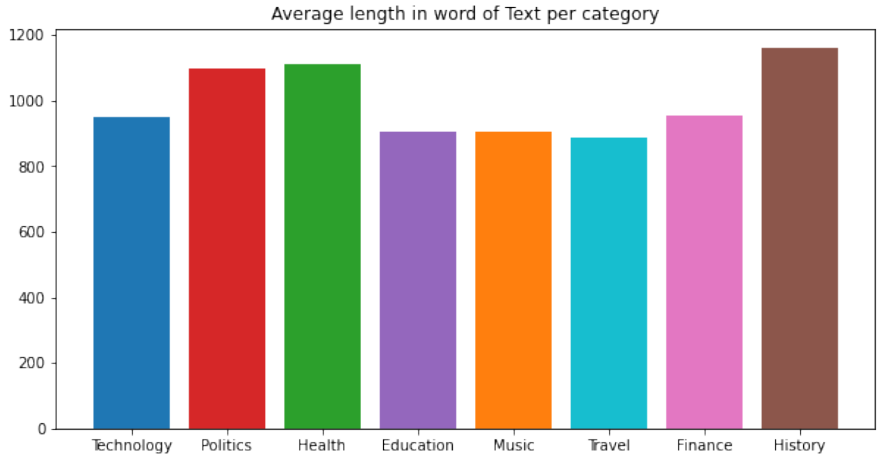

Figure 2.3: Average Length per Category

In Figure 2.4 it is shown the **Lexical Diversity** computed making the average value for each article of the category. The values are pretty similar with a spike in *Music* and *Travel* and lowest values in *Technology* and *Finance*.

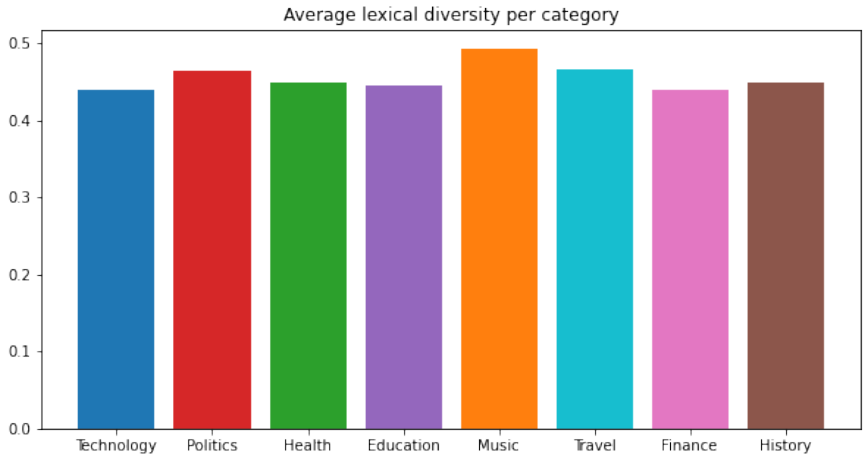

Figure 2.4: Average lexical diversity per Category

2.3 Data Preprocessing

Once we loaded the entire datset from kaggle we have to process it to make it suitable for our analysis. First of all we have to give a single **category tag** to each article, in order to do that we create a dictionary with eight categories, then we exploit the *Tags* column in the dataset. To be more precise, we assign to each article a category among the ones in our dictionary if and only if one of the chosen category appears in the article's list of tags. If we do not find any category in the list of tags then we assign **"NoTag"** label to the article.

Once we perform this step we can remove all the articles with **"NoTag"** as category, after that we obtain roughly 24000 articles from the 192000 of the entire dataset, we can also drop all the columns maintaining only the one that contains the article's body and the one that contains the category that we assigned in the past step.

Now we process the body of each article in the following way:

- *Expand contracted forms*: for this purpose we can use the **contractions** library, this can help us to transform *"they're"* in *"they are"*.
- *Make all the words to lower case*: this steps helps to standardize the article's text.
- *Filter out all the chars that are not part of the normal alphabet*: using a regular expression we can remove all the non standard chars like *question or exclamation marks*, *parenthesis* or *quotation marks*.

After this process we can add the column that contains the preprocessed text to the dataset and drop the raw text.

As we said in section 2.2, the dataset is heavily unbalanced towards the **Technology** category as shown in figure 2.2. To solve this problem we perform an **undersampling** approach to match the number of the minority category, **Finance** in our case. In this step we can also transform all the category into numerical label from 0 to 7. After this process all the categories have roughly 1500 articles.

	Tag	TextPreprocessed	Num_Tag
14676	Technology	when life closes a door you open many zoom win...	0
2924	Technology	for some 5g is a miracle network that will bri...	0
4962	Technology	the firstever published website came to the in...	0
15838	Technology	my reluctant iphone 12 purchase story time whe...	0
5253	Technology	baseline image on the left and incorrect backg...	0
...	...	...	...
797	History	steve jobs dark past the time steve jobs cheat...	7
11409	History	the san francisco oakland bay bridge had been ...	7
21947	History	the skydiver saved by fire ants after plunging...	7
15668	History	a collection of open letters to my family and ...	7
8527	History	feeling currents wash over me soothing current...	7

Figure 2.5: Dataset after Preprocessing phase

Chapter 3

Architectures

Here we present all the approaches used in our multi-class classification task. We range from a SOTA approach like **Transformers**, to a rudimental experiment like the **Keyword with Word Embeddings**. Together with the description of the method, we present also the results and in the case of BERT, the internal comparison between the architectures.
It's worth to mention the methodology used in order to train, validate and test all the architectures. The pre-processed dataset is split into **Train**, **Validation** and **Test** set with respectively **70%**, **15%** and **15%**. Some methods don't need the training. The comparison is valuable since the Test set is the same for every architecture because we use the same *seed* to split the data. The values that we consider useful in order to assess a method are:

- *Accuracy*
- *Training Time*
- *Inference Time*
- *F1 macro*

They are explained better in Chapter 4.

3.1 BERT family

Bidirectional **E**ncoder **R**epresentations from **T**ransformers "**BERT**" is a transformer-based machine learning technique for natural language processing, it is the SOTA for many tasks. This architecture is based on:

- *Self Attention*: is an attention mechanism relating different positions of a single sequence in order to compute a representation of the sequence, so it will consider also the context of the word taking into account the tokens before and after it.
- *Positional Encoding*: that injects input word-position information inside the encodings, it allows the model to be trained fast with respect to the RNN.

The Progenitor of the whole set of models used in this project is the **BERTbase** model. There are 3 parameters to configure the model:

- L: the number of encoders to be stacked (the left portion of the Transformer in Figure 3.2).
- H: the hidden dimension, so the dimension of the embeddings inside the network.
- A: the number of attention heads, the concept of multi-head Attention is a module for attention mechanisms which runs through an attention mechanism several times in parallel. The independent attention outputs are then concatenated and linearly transformed into the expected dimension. Intuitively, multiple attention heads allows for attending to parts of the sequence differently (e.g. longer-term dependencies versus shorter-term dependencies).

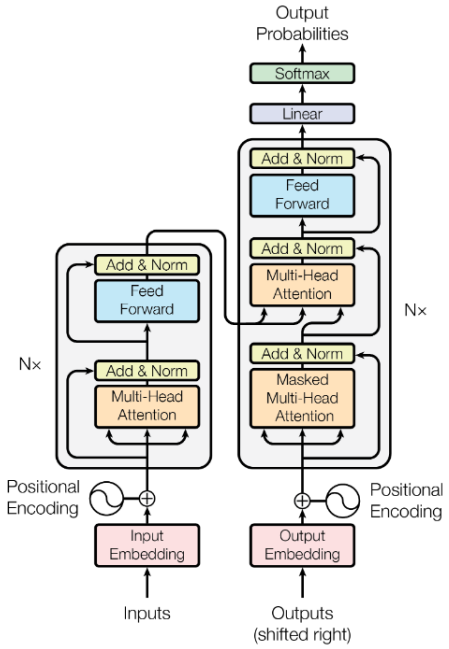

Figure 3.1: Architecture of Transformer from the paper "All you need is attention"

There are different configurations of the model, arranged using the parameters L, H and A. It is important to note that all these models have been trained in a self supervised way on *Wikipedia* and *BookCorpus* datasets using language modeling, specifically:

- *Masked Language Model*: as the name suggests, means we mask words from a sequence of input or sentences and the designed model needs to predict the masked words to complete the sentence, this is very useful for learning bidirectional representation of input.
- *Next Sentence Prediction*: consists of giving BERT two sentences, sentence A and sentence B. We then say, 'hey BERT, does sentence B come after sentence A?' — and BERT says either IsNextSentence or NotNextSentence. This is useful for capturing the relationship between pairs of input sentences.

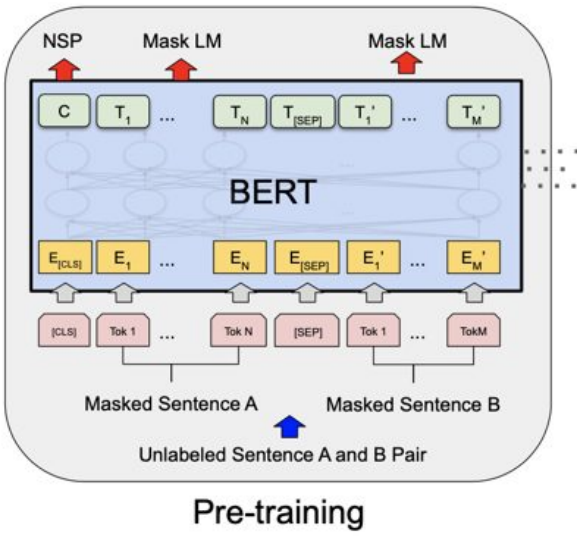

Figure 3.2: Illustrated pre-training of BERT

In order to fit our multi-class classification problem, we stack a **Dense Layer** with an output of 8 neurons, the number of categories in our problem.

The **configuration of training** is the same for the whole set of BERT models:

- *BATCH SIZE*: 32.
- *EPOCHS*: 20.
- *Early Stopping* with *patience* equals to 2 on validation loss.
- *Optimizer*: RMSprop.

3.1.1 BERTbase

As we can see from this table, there are several configuration of BERT. The **BERTbase** is the configuration with **L=12**, **H=768** and **A=12**, highlighted in red. It has **110M** of parameters, where L = number of layers, H = hidden size and A = number of self-attention operations.

	H=128	**H=256**	**H=512**	**H=768**
L=2	4.4	9.7	22.8	39.2
L=4	4.8	11.3	29.1	53.4
L=6	5.2	12.8	35.4	67.5
L=8	5.6	14.4	41.7	81.7
L=10	6.0	16.0	48.0	95.9
L=12	6.4	17.6	54.3	110.1

Figure 3.3: Million of Parameters

Results

In this part are shown the results obtained with this architecture. The Confusion Matrix shows a balanced behaviour on different classes, the most incorrectly predicted label is *Education*, that is confused with *Technology*. The model struggles also with the *Health* category that is confused with both *Education* and *Travel*.

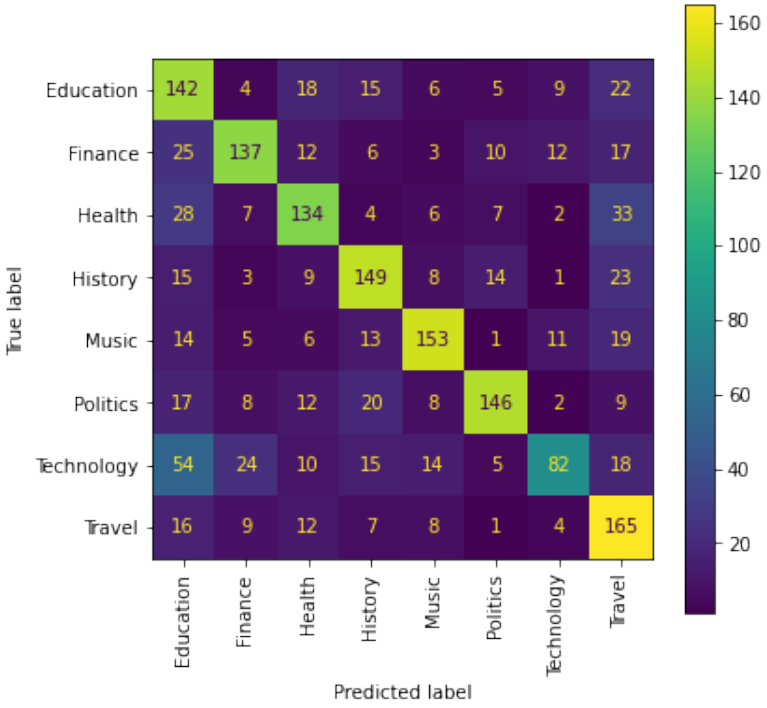

Figure 3.4: Confusion Matrix

The *training time* is 1200s in 10/20 Epochs thanks to Early Stopping, with an average of 120s per epoch.
The *inference time* on the test set is 17.94s.
The overall *accuracy* is $62,46\%$, the *F1 metric* for all the classes is reported in the array.

$$\begin{bmatrix} 0.53383459 & 0.65393795 & 0.61751152 & 0.66075388 \\ 0.71495327 & 0.71046229 & 0.47536232 & 0.625 \end{bmatrix} \tag{3.1}$$

The *F1 - macro* is 0.62397.

3.1.2 smallBERT

smallBERT has the same BERT architecture but fewer and smaller Transformer blocks and number of attention heads. The configuration is **L=4**, **H=512** and **A=8**, it is highlighted in green in the table 3.3. There is a novelty in this model, the usage of *knowledge distillation*. The idea is that the smaller student model is trained to recover the predictions of a highly accurate teacher using as ground truth, not only the hard labels of the dataset, but also soft labels related to the teacher's predictions, that allows better generalization. The soft labels coincide with the class probabilities produced by the teacher, obtained with a Softmax on the logits divided by a temperature parameter.

Results

In this part are shown the results obtained with this architecture. It achieves an high level of accuracy in almost all the classes, with exception of *Education* and *Technology*.

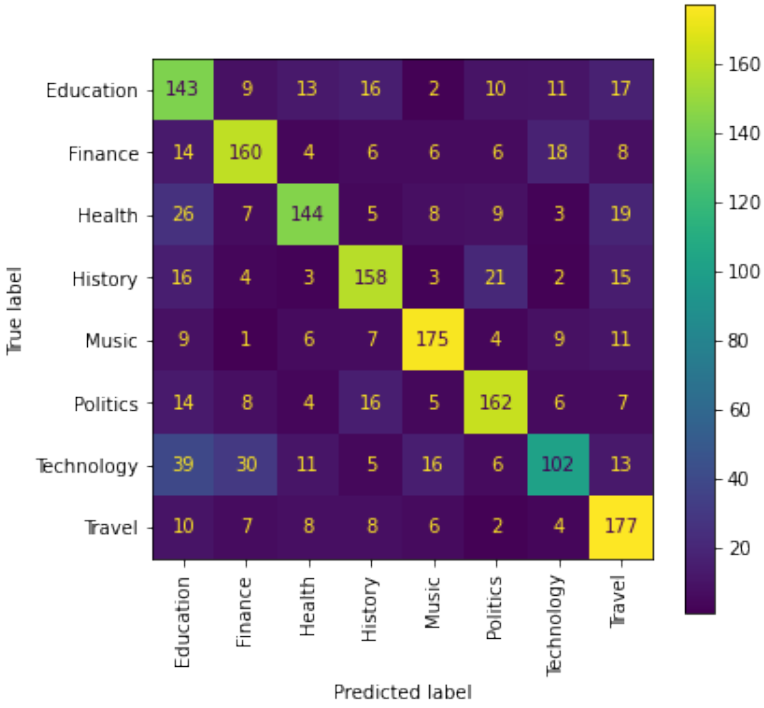

Figure 3.5: Confusion Matrix

The *training time* is 650s in 13/20 Epochs thanks to Early Stopping, with an average of 50s per epoch.
The *inference time* on the test set is 7.35s.
The overall *accuracy* is $68,83\%$, the *F1 metric* for all the classes is reported in the array.

$$\begin{bmatrix} 0.58130081 & 0.71428571 & 0.69565217 & 0.71331828 \\ 0.79006772 & 0.73303167 & 0.54111406 & 0.72392638 \end{bmatrix} \tag{3.2}$$

The *F1 - macro* is 0.68658.

3.1.3 BERTexpert

This model uses a BERTbase architecture pretrained from scratch on Wikipedia and BooksCorpus. This model is intended to be used for a variety of English NLP tasks. It is important to consider that the pre-training data contains more formal text and the model may not generalize to more colloquial text such as social media or messages, so it is suitable for our purpose since journal articles are suppose to use formal language.

Results

In this part are shown the results obtained with this architecture. Here we observe the usual *Education* class that causes problems, in addition with *Technology* that struggle to identify correctly the article at the expense of *Finance* and viceversa.

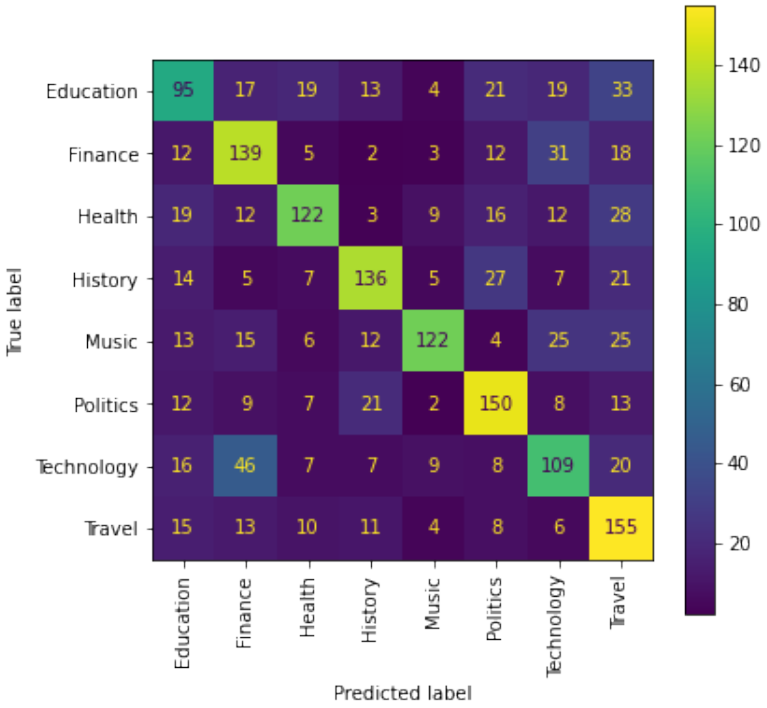

Figure 3.6: Confusion Matrix

The *training time* is 2124s in 18/20 Epochs thanks to Early Stopping, with an average of 120s per epoch.
The *inference time* on the test set is 17.81s.
The overall *accuracy* is $57,95\%$, the *F1 metric* for all the classes is reported in the array.

$$\begin{bmatrix} 0.45563549 & 0.58158996 & 0.6039604 & 0.63700234 \\ 0.64210526 & 0.64102564 & 0.49658314 & 0.57943925 \end{bmatrix} \tag{3.3}$$

The *F1 - macro* is 0.57966.

3.1.4 Electra

The **ELECTRA** model was proposed in the paper *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. ELECTRA is the pretraining approach, therefore there is nearly no changes done to the underlying model BERT, the configuration is **L=12**, **H=256** and **A=4**. This pretraining approach is the *replaced token detection* that randomly selects words in the text, but unlike the latter it does not mask them with a token, but replaces them with plausible alternatives returned by a small generator network, which samples from a proposal distribution. However this is not an adversarial model, in fact the generator is trained with maximum likelihood and not to fool the discriminator, due to the difficulty of applying GANs with text.

Results

This model behaves very poorly with almost all the classes.

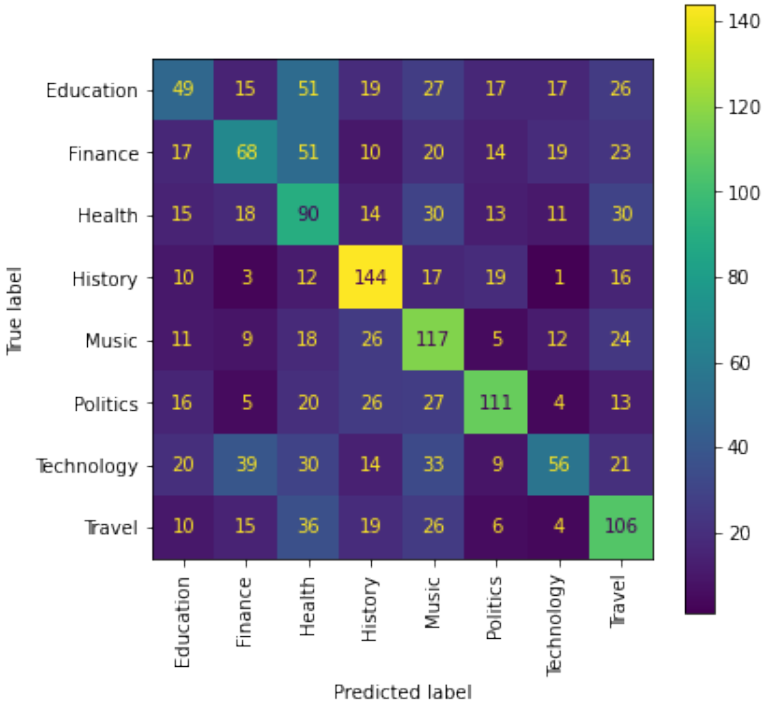

Figure 3.7: Confusion Matrix

The *training time* is 1060s with an average of 51s per epoch.
The *inference time* on the test set is 7.70s.
The overall *accuracy* is $41,77\%$, the *F1 metric* for all the classes is reported in the array.

$$\begin{bmatrix} 0.26558266 & 0.34517766 & 0.34026465 & 0.58299595 \\ 0.45086705 & 0.53365385 & 0.32369942 & 0.44074844 \end{bmatrix} \tag{3.4}$$

The *F1 - macro* is 0.41037.

3.1.5 Albert

Albert is light version of BERT with reduced number of parameters, that was presented in *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. This model introduces 3 main ideas to the original BERT's design choices:

- *Factorized embedding parameterization*: concerns to reduce the complexity of the representation of each word, that is not projected directly into the hidden space, but is first projected into a lower embedding space of dimension E.
- *Cross-layer parameter sharing*: states that all the parameters between transformer layers are shared, in particular both feed-forward network and sharing attention parameters.
- *Inter-sentence coherence loss*: changes the NSP task on which BERT is pre-trained, since with this approach the network learns more about topic prediction than coherence prediction. Here the same technique is used to generate positive samples (two consecutive segments of the same document), while for the negative examples are used the same two consecutive segments but with their order swapped.

Results

In this part are shown the results obtained with this architecture. They are poor as we can see from the confusion matrix.

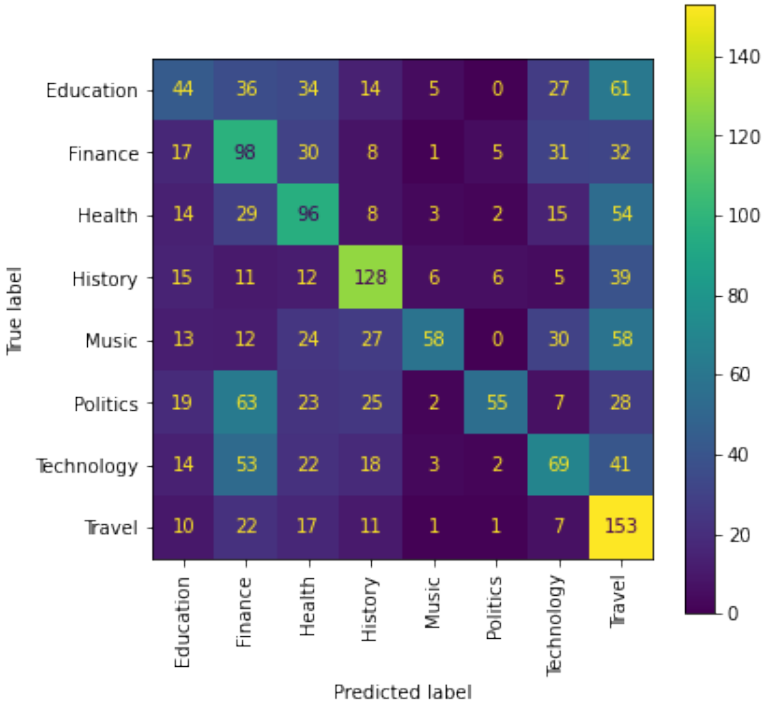

Figure 3.8: Confusion Matrix

The *training time* is 875s in 7/20 Epochs thanks to Early Stopping, with an average of 123s per epoch.
The *inference time* on the test set is 18.58s.
The overall *accuracy* is $39,51\%$, the *F1 metric* for all the classes is reported in the array.

$$\begin{bmatrix} 0.23978202 & 0.35897436 & 0.40083507 & 0.55531453 \\ 0.38538206 & 0.37542662 & 0.33414044 & 0.44476744 \end{bmatrix} \tag{3.5}$$

The *F1 - macro* is 0.386827.

3.1.6 Comparison

In this section we compare the models belonging to the **BERT family** in order to find the best one to be compared with the other methods.
The Table 3.1 shows a global overview about the models, the **number of parameters** ranges from the 12M of Albert to the 108M of *BERTbase* and *BERTexpert*, and not surprisingly this reflect to the **Accuracy**. Indeed the *Electra* and *Albert* models are not suitable for our task achieving roughly $20-30\%$ less accuracy comparing to the others.
It is surprisingly instead the behaviour of the models with respect to the **training time** and **inference time**, since *Albert* has the largest Inference Time despite being the tiniest model.

Another strange behaviour comes from *smallBERT* model that achieves the highest accuracy despite being 4 times smaller in terms of number of parameters compared to the *BERTbase* and *BERTexpert* models.

Model	Parameters	Accuracy (%)	F1 - Macro	Inference Time (s)	Training Time (s)
BERTbase	108M	62.46	0.62397	17.94	1200
smallBERT	*29M*	*68.83*	*0.68658*	*7.35*	*650*
BERTexpert	108M	57.95	0.57966	17.81	2124
Electra	14M	41.77	0.41037	7.7	1060
Albert	12M	39.51	0.38682	18.58	875

Table 3.1: Comparison of evaluation metrics among BERT family

In the Figure 3.9 it is highlighted the trade-off between **Training Time** and **Accuracy**, as we can see the *smallBERT* behaves outstandingly in both metrics, so it is easy to make a choice among the models. As we expected *Albert* and *Electra* train fastly but with a bad Accuracy. It is interesting the behaviour of BERTexpert that despite the same amount of Parameters of BERTbase, has a training time that is almost doubled.

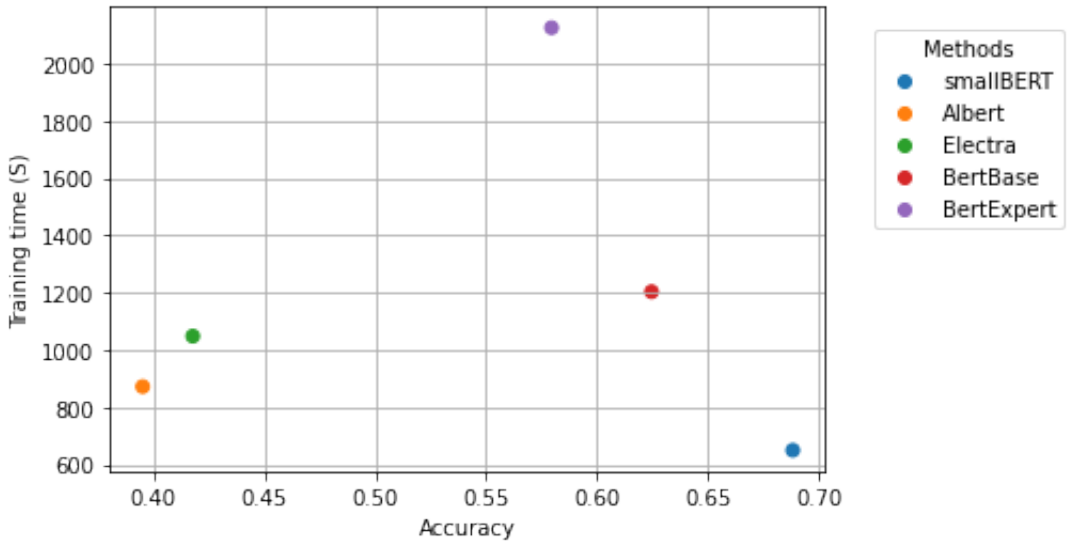

Figure 3.9: Comparison between different Bert architectures

In the Figure 3.10 instead can be seen a strange behaviour of *Albert* that has the major inference time despite being the tiniest model, also here the *smallBERT* has the best numbers.

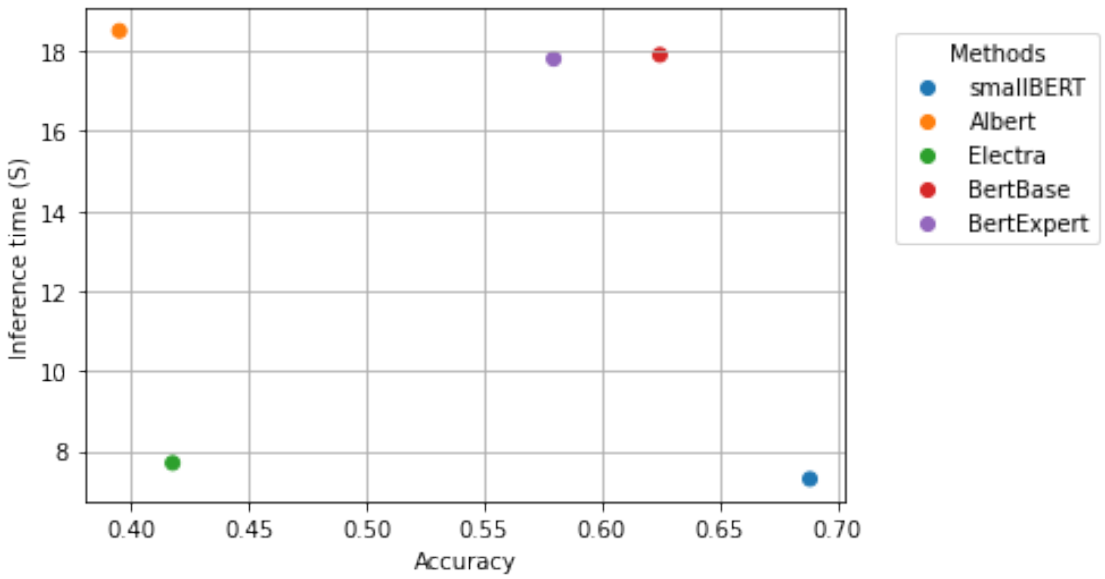

Figure 3.10: Comparison between different Bert architectures

The Table 3.2 wants to understand how the categories are approached by all models. The worst category is *Education* with an average **F1 Score** of 0.41, similar to the *Technology* category with 0.44, we expected this because giving a look at some articles it is common that the Education articles deals with modern education approaches that in some way can involve terms that belong to the Technology vocabulary. The best category is *History* along with *Music* and *Politics*, they are categories with a peculiar and strongly uncorrelated vocabulary, considering the other categories.

Model	**Education**	**Finance**	**Health**	**History**	**Music**	**Politics**	**Technology**	**Travel**
BERTbase	0.533	0.653	0.617	0.660	0.714	0.710	0.475	0.625
smallBERT	*0.581*	*0.714*	*0.695*	*0.713*	*0.790*	*0.733*	*0.541*	*0.723*
BERTexpert	0.455	0.581	0.603	0.637	0.642	0.641	0.496	0.579
Electra	0.265	0.345	0.340	0.582	0.450	0.533	0.323	0.440
Albert	0.239	0.358	0.400	0.555	0.385	0.375	0.334	0.444
Average	0.41	0.53	0.53	0.63	0.6	0.6	0.44	0.56

Table 3.2: Comparison of F1 Score among BERT family for categories

We cannot assert that this homogeneous behaviour with respect to the classification of the categories is only due to the characteristics of the articles, it can also be that the whole set of models, being very similar, act also in similar way.
Regarding this comparison we conclude that the best BERT model for our problem is the **smallBERT**.

3.2 CNN

The structure of the CNN that we implemented is shown in Figure 3.11.

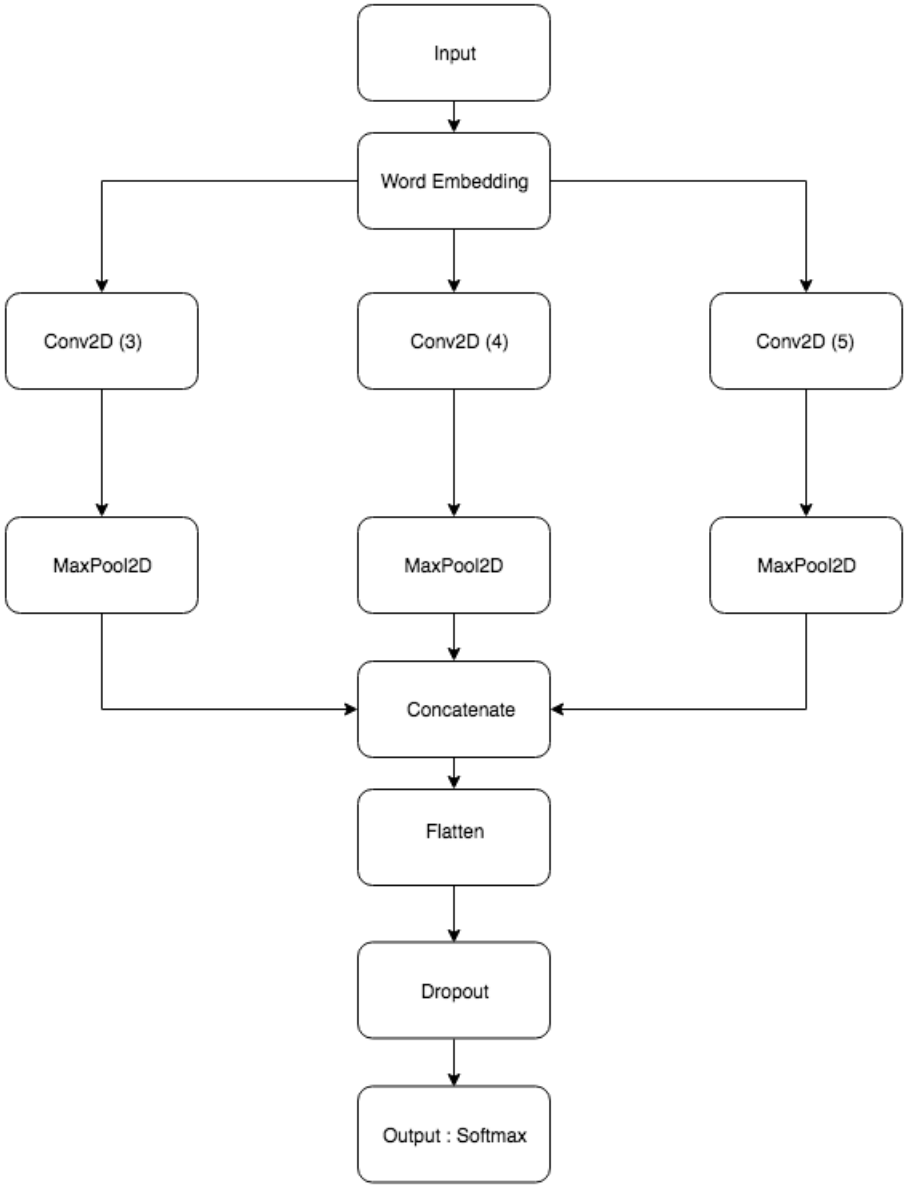

Figure 3.11: Architecture of the CNN

The input data needs to be preprocessed, we create an instance of a keras Tokenizer that takes as input the number of the most frequent word that have to be considered in the resulting dictionary, in our case 2000. This **dictionary** maps each word into an integer that correspond to a rank based on the frequency in which the word appears in the corpus (the whole set of articles), the higher is the rank the more frequent the word is.

After this operation we use this vocabulary to transform each text in a **sequence of inte-**

gers, basically the tokenizer takes each word in the article and replaces it with with its corresponding rank value from the dictionary. Now each sequence is padded so that the length is the same among all the articles, in particular we choose a length of 1000; if an article is longer it will be truncated otherwise it will be padded with zeroes.

The first layer of our CNN is dedicated to the **embeddings**, to build this layer we can use the **Glove** pretrained embeddings and give them as weights to the layer. Then we use three different **Convolutional** layers with different filter sizes; 3, 4 and 5, the second dimension of the filter matches the embedding length that is 100, this results in taking a window of words wide 3, 4 or 5 at a time. We can finally concatenate the results and use a **Dropout** layer to prevent the overfitting phenomenon. The last layer is a **Dense** layer that works as classifier.

Results

In this part are shown the results obtained with this architecture. The results are really great, as also shown by the *F1Metric* in 3.6.

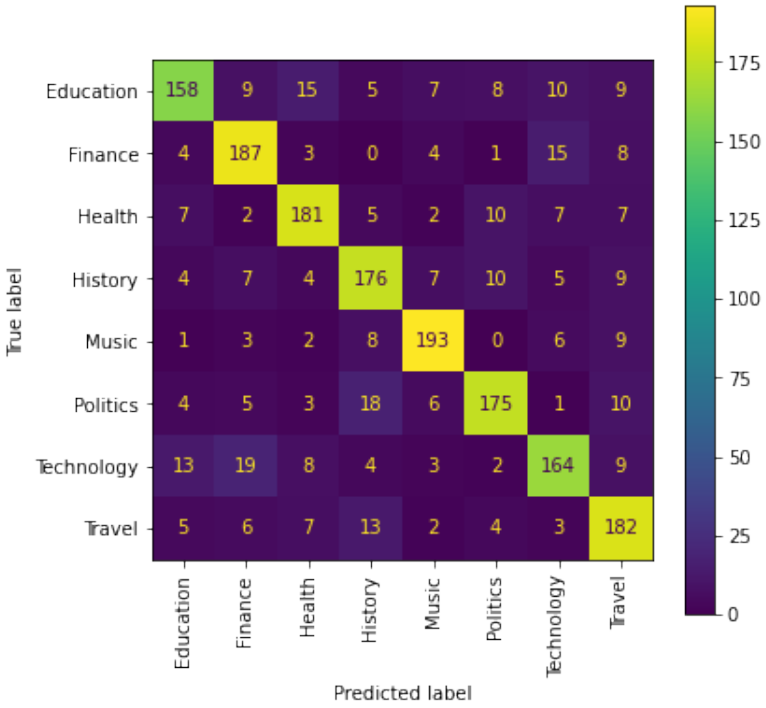

Figure 3.12: Confusion Matrix

The *training time* is 415s in 20/20 epochs.
The *inference time* on the test set is 1.54s.
The overall *accuracy* is 79.81%, the *F1 metric* for all the classes is reported in the array 3.6.

$$\begin{bmatrix} 0.75779376 & 0.81304348 & 0.81531532 & 0.7804878 \\ 0.86547085 & 0.81018519 & 0.75750577 & 0.7827957 \end{bmatrix} \tag{3.6}$$

The *F1 - macro* is and 0.7978.

3.3 Bayes

In this section we will use a **bayesian classifier** to choose the category given an article. In order to implement this approach we can build a pipeline. The first element of the pipeline is a **TF-IDF Vectorizer**, this component of the pipeline converts a collection of articles to a matrix of TF-IDF features. The second component is a **Multinomial Naive Bayesian Classifier** since this type of classifier is suitable for classification with discrete features like the TF-IDF features that were extracted in the previous component of the pipeline.

A Multinomial Naive Bayes algorithm is a probabilistic learning method that is used in NLP. The algorithm is based on the Bayes theorem and predicts the category of a text such as a piece of email or newspaper article. It calculates the probability of each category for a given sample and then gives the category with the highest probability as output.

Results

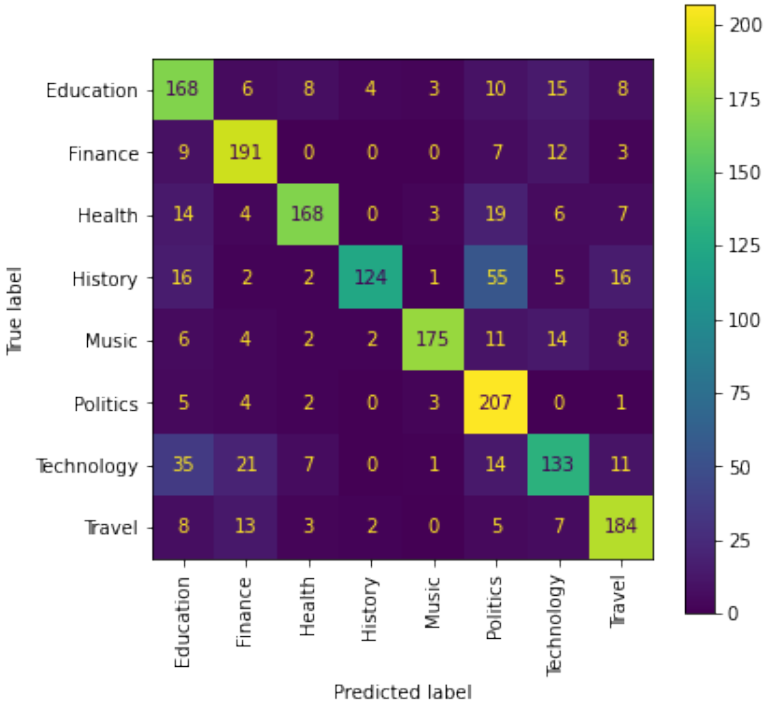

Figure 3.13: Confusion Matrix

The *training time* is 6.82s.
The *inference time* on the test set is 2.42s.
The overall *accuracy* is 76.09%, the *F1 metric* for all the classes is reported in the array 3.7.

$$\begin{bmatrix} 0.69565217 & 0.81798715 & 0.81355932 & 0.70254958 \\ 0.85784314 & 0.75272727 & 0.64251208 & 0.8 \end{bmatrix} \tag{3.7}$$

The *F1 - macro* is 0.7603.

3.4 Keyword with Word Embeddings

In this approach we want to use the keywords extracted from an article's body and compare their embeddings with the embeddings of the eight categories.

To extract the keywords from the articles we use **YAKE** that is a light-weight unsupervised automatic keyword extraction method which rests on text statistical features extracted from single documents to select the most important keywords of a text. We extract 9 keywords from each article and we can add the list of keywords to the correspondent row.

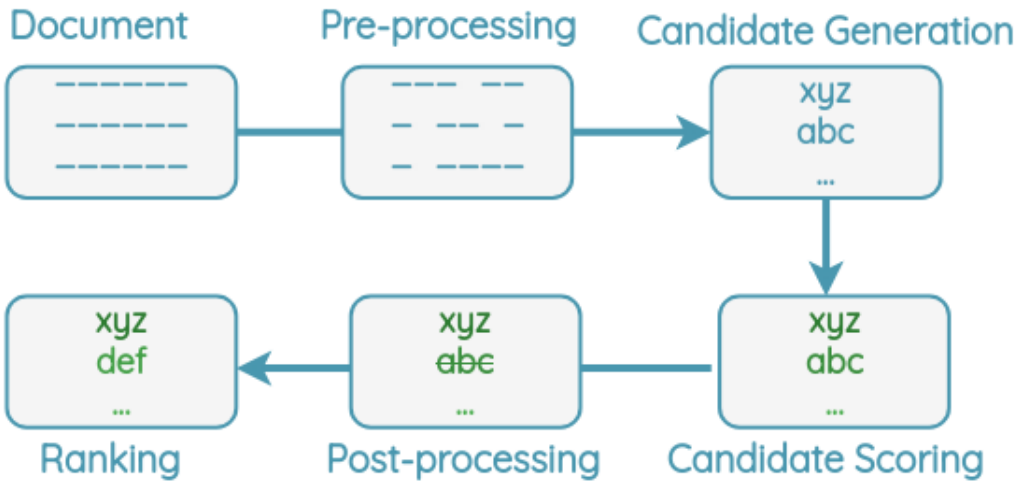

Figure 3.14: General pipeline for a keyword extraction

YAKE was introduced in 2018 with stunning results. It uses statistical features to identify and rank the most important keywords from a given text, it does not require any logical information like POS tagging, because of that it can be used with any language, it only requires a list of stop words for the chosen language.

First of all sentences are split into terms using spaces and special character as delimiters, then we chan chose the maximum length of the keywords to be generated. After that we can remove phrases that contain punctuation marks and phrases that begin and end with a stop word.

The next step is computing how good a word is. YAKE uses five different features:

- **Casing**: this feature gives more importance to capitalized words and acronyms.
- **Word Positional**: this feature gives more importance to words that are at the beginning of the document, it is based on the assumption that the main concepts are usually concentrated ad the beginning.
- **Word Frequency**: this feature takes into account how frequent a word is.
- **Word Relatedness to Context**: this feature quantifies how related a word is to its context. For that, it counts how many different terms occur to the left or right of a candidate word. If the word occurs frequently with different words on the left or right side, it is more likely to be a stop word.

- **Word Different Sentence**: this features takes into account how much a word appears into different sentences.

All the five features are then combined together to compute a score for each word. Now, for each candidate keyword, a score is computed and less frequent keywords are penalized. Since it is common to obtain similar words when we extract keyphrases, we simply take a list of current keywords sorted by their score, then a new keyword is **skipped** if it has a low *Levenshtein* distance with all the current keywords, otherwise it is added to the list. As a final step we can sort all the keywords and return the desired number. Once we have extracted the keywords we can compute the embeddings, to do that we use **Glove**. Glove is an unsupervised learning algorithm for obtaining vector representations for words, given a corpus as input we can produce for each word a position in an high dimensional space such that similar words are close in the space.

We use a pretrained model since it gives us good results. We can download "glove.6B.100d.txt" that contains the embeddings for roughly 400000 words, each embidding is composed of 100 numbers.

Now we have to create a dictionary that contains the embeddings for the eight categories that we chose at the beginning of our analysis. Once we have performed this step we can iterate through the list of keywords that we extracted before and, as a result, we create and add a list of embeddings to the relative row.

In order to give the **predicted category** we adopt a voting approach. To be more precise, given an article with its nine keywords, for each of them we compute the cosine similarity and assign the closest category. At this point, to predict the category for the article we can choose the most frequent category.

Results

In this part are shown the results obtained with this architecture.

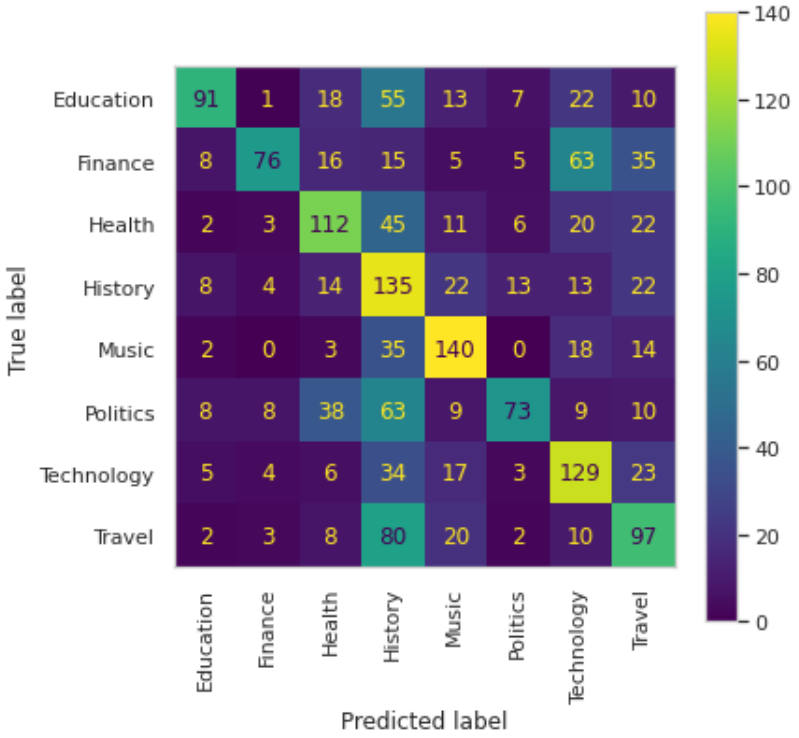

Figure 3.15: Confusion Matrix

In figure 3.15 is shown the confusion matrix, as we can see the results are quite good. It is important to notice that some categories like *Education*, *Finance* etc. have some problems to be correctly classified. This may be due to the fact that the keywords extracted does not represent the general category in the best way.

The overall accuracy is $48,32\%$, the *F1 metric* for all the classes is reported in the array 3.8.

$$\begin{bmatrix} 0.53061224 & 0.47204969 & 0.51376147 & 0.38961039 \\ 0.62360802 & 0.44648318 & 0.51089109 & 0.42637363 \end{bmatrix} \tag{3.8}$$

The *training time* is 0s, since we use pre-trained embeddings and a simple keyword extractor. The *inference time* on the test set is 136s, this value is composed by 3 parts:

- 109s related to the keyword extraction
- 0s to convert the keyword in its embedding
- 27s to compute the predicted tag

The *F1 - macro* is and 0.4891.

Chapter 4

Evaluation Metrics

The metrics that we use in order to assess each model are:

- ***Accuracy***: this value is not the proper one alone when dealing with multi-class classification problem but we keep it to give an overview. It is worth to remember how in a multi-class problem, a good value of accuracy depends on the number of classes, so in our case we have to compare the accuracy with the random choice that is $\frac{1}{8} = 12.5\%$. For example a classifier with an accuracy of 62.5% is 5 times more accurate than the random choice, so it can be considered good.
- ***Confusion Matrix***: this matrix can help us understanding how each sample is classified, we can use it together with the accuracy to have a better overview of the performances of a given approach. For example, in an imbalanced situation an high accuracy could derive from the correct classification of the majority class, the confusion matrix helps us to give an objective evaluation.
- ***Training Time***: this value shows how much time the model need to understand the characteristics of the dataset in order to carry out the task properly.
- ***Inference Time***: this value shows how much time the model needs to give a prediction on the category. Together with the *training time* they can be use to evaluate the trade-off of a given approach.
- ***F1-macro***: $\frac{\sum F1-scores}{numberofclasses}$, it gives each class equal importance.

Chapter 5

Comparison

In this Chapter we propose a comparison between methods, in the case of multiple instances of the same approach we pick the best one to be compared with the others, like in the case of BERT.

In Table 5.1 we can see that generally speaking the CNN approach is the best since it performs well in almost all the evaluation metrics, in particular it has outstanding results in term of accuracy and F1-Macro.

Model	Accuracy (%)	F1 - Macro	Inference Time (s)	Training Time (s)
smallBERT	68.83	0.68658	7.35	650
CNN	***79.81***	***0.7978***	1.54	415
Bayes	76.02	0.7590	***1.25***	7.4
Keyword Embeddings	48.32	0.4891	136	***0***

Table 5.1: Comparison of evaluation metrics among all the approaches

In Figure 5.1 the inference time against the accuracy is shown, Bayes and CNN have the best behaviour. Keyword Embeddings approach performs the inference in a very long time but we have to keep into account that this method has no training time, as we can see in Figure 5.2. So the idea could be to use this method only in the case when no training set is available to build a model and so this solution can give a baseline with a reasonable accuracy.

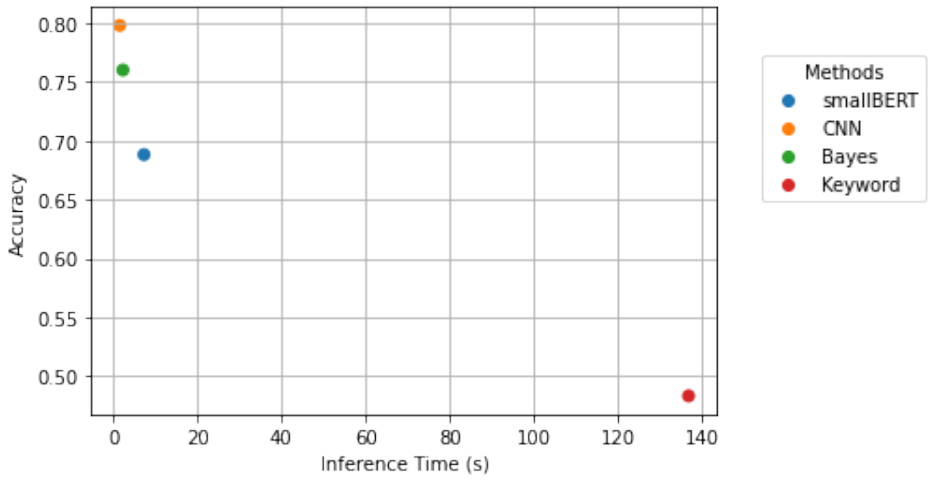

Figure 5.1: Comparison Among all the architectures

In Figure 5.2 the training time against the accuracy is shown, we can clearly see that the Bayesian approach has a negligible training with the respect to the CNN even if the Bayes accuracy is slightly lower. In the situation in which the training time is critical, the Bayes approach can be a reasonable solution.

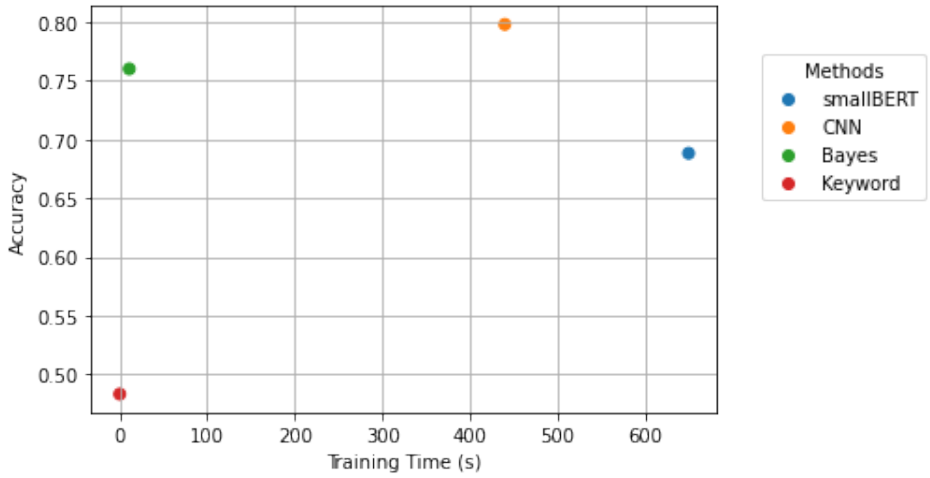

Figure 5.2: Comparison Among all the architectures

In table 5.2 is shown the F1 - score for all the categories. We can notice that **Music** has the highest F1-score across all the approaches, this may be due to the fact that music articles have peculiar vocabulary and structure that can help the classification, as we can see in Figure 2.4. On the other hand, **Technology** has the worst performances, this may be due to the fact that nowadays technology is a quite pervasive topic and the possibility to get semantically distant fields is very high.

It is interesting to notice that **History** has quite good performances in all the approaches but on the keyword approach. This is due to the fact that many keywords extracted from

the articles are Proper names, this result may be due to the fact that the embeddings of hystorical people's names could be distant from the embedding of History as a word. The same phenomenon occurs also with the **Travel** category.

Model	Education	Finance	Health	History	Music	Politics	Technology	Travel
smallBERT	0.581	0.714	0.695	0.713	0.790	0.733	0.541	0.723
CNN	0.757	0.813	0.815	0.780	0.865	0.810	0.757	0.782
Bayes	0.700	0.816	0.801	0.716	0.850	0.766	0.625	0.794
Keyword Embeddings	0.530	0.472	0.513	0.389	0.623	0.446	0.510	0.426

Table 5.2: Comparison of F1 Score among all architecture for categories

To sum up, the best approach to assign a category to a piece of text, according to our work, is the **CNN** since it can achieves almost an 80% general accuracy with high values of F1-score for each class as reported in array 3.6.