UNIVERSITÀ DI PISA

Study Program:

Master’s in Data Science and Business Informatics

Academic Year:

2020/2021

Course and Project:

Text Analytics | Final Project

Team Members:

José Mateo González Richardson 606566

Francesco Santucciu 599665

Alessandro Bonini 604482

Professor:

Andrea Esuli

Introduction

Text or sequences classification is a very important task in data science and machine learning studies widely used in several real-world applications. Over the past recent decades, many strategies and algorithms trying to solve this kind of problems have emerged from different approaches.

In this project, we decided to tackle this problem and test several of the most popular text classification methods to compare the different results, presenting how one improves against the other.

The dataset we used was obtained from the "Contradictory, Dear Mr. Watson" competition posted on the Kaggle.com site and is described later in this document. The way in which the different techniques that we use in this project will be presented are divided into 3 strategies, the first consisting of a series of classic machine learning estimators such as RandomForest, NaiveBayes, SVMs, among others. Second, we tested more advanced methods such as Deep Neural Network, for which we applied a network architecture of LSTM layers. Finally, as a last strategy, we apply and configure some of the variants of the BERT transformer models.

1. Exploratory Data Analysis

1.1. Data Definition

Our project involves two datasets, the first will be used to train the models while the second will be used to test the accuracy of the classifiers we use. The training dataset has 12120 records, and 6 attributes. The attributes are:

- Id: unique identification of the record
- Premise: a first sentence to compare with the hypothesis
- Hypotesis: a second sentence to compare with the premise
- Language: the language used in the sentences
- Lang_abv: the abbreviation of the language used
- Label: the classification of the record (0 = implication, 1 = neutral, 2 = contradiction)

1.2. Data Distribution and Statistics

The dataset dedicated to the test, on the other hand, has the same attributes minus the label column.

Our goal is to create NLI (Natural Language Inferencing) models able to assign the values 0,1 and 2 (corresponding to implication, neutrality, and contradiction) to pairs of premises and hypotheses.

First, we did an exploratory analysis of the dataset by going to see how many values we have for each label:

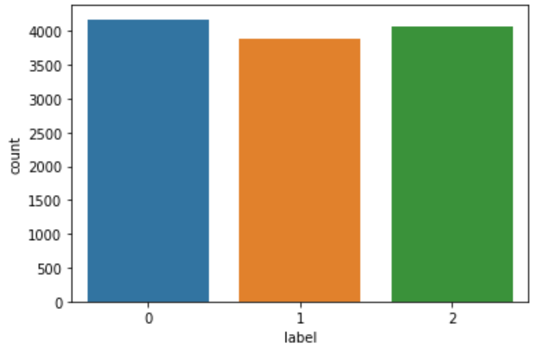

Figure 1: Labels distributions

We can observe in Fig. 1 that the dataset is well balanced, our training records are labeled in this distribution:

- 4176 as 0
- 3880 as 1
- 4064 as 2

Then we analyzed the distribution of the attribute "language", which contains 15 languages, namely: English, French, Thai, Turkish, Urdu, Russian, Bulgarian, German, Arabic, Chinese, Hindi, Swahili, Vietnamese, Spanish, Greek.

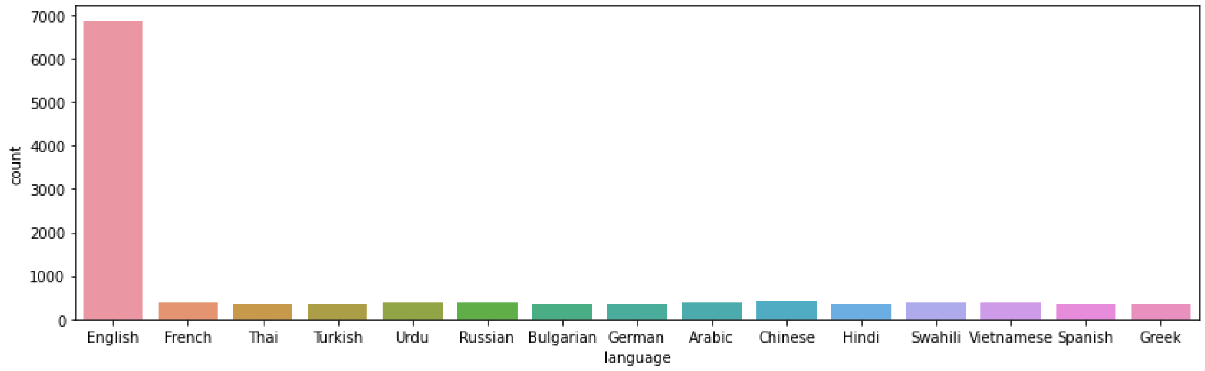

Figure 2: Languages count distribution

As a last thing in this phase, we checked if there were missing values or duplicated records, in both cases we did not find any for any attribute.

Finally, having noticed that 56.7% of the records are in English, we decided to eliminate the other languages to reduce the dimensionality given the exponential growth for each language inserted. We have also eliminated the attribute "lang_abv" because it is redundant.

In this way our training dataset remains with 4 attributes and 6870 records, distributed as follows: 2427 = 0, 2166 = 1, 2277 = 2.

The second phase of our work focused on a statistical analysis of the text within the dataset, considering both the attribute "premise" and the attribute "hypothesis", in fact our analysis was done on the concatenation of these two attributes to create a single new attribute called "joined".

We then created a string containing all the values of the attribute "joined", removing all capital letters and punctuation to obtain some information about the text that we were going to use for classification.

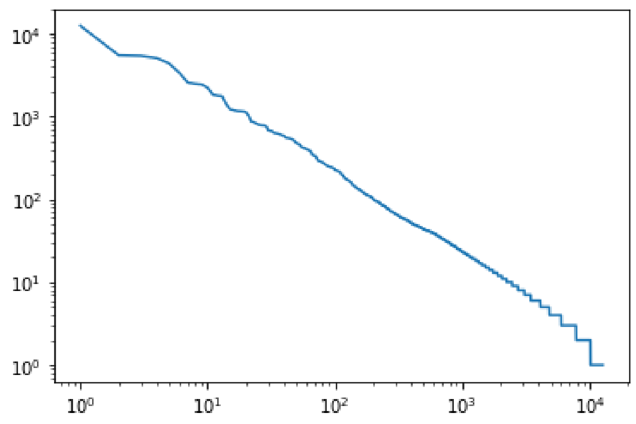

Figure 3: Word count frequency plot

First, we divided this string into tokens to figure out the total size of the text and the vocabulary used, resulting in 199063 and 12699 words, respectively. We also counted the frequency of these words, but obviously the most frequent were all stop-words (i.e., words that are common but give little information.)

As we can see from the graph, we have few words repeated many times and many words that appear few times.

For this reason, we have decided to eliminate the stopwords and redo the same previous analysis. This time we obtain words more specific to the context we analyzed. We report below a word cloud image that summarizes the most frequent words:

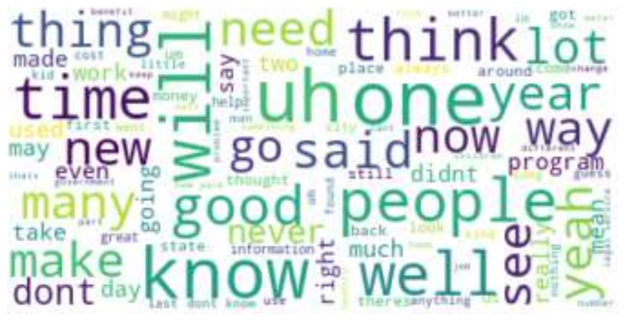

Figure 5: Most frequents without stopwords cloud

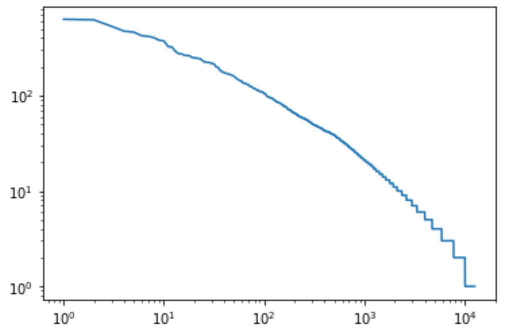

Figure 4: Without stopwords count frequencies

In this case we can notice a trend of the graph like the one obtained before the elimination of the stopwords but with less words used many times

We then did another step to see the most frequent words in the attribute 'premise' and in the attribute 'hypothesis' to understand the words that characterize both attributes, we report the two-word clouds thus obtained:

- Premises:

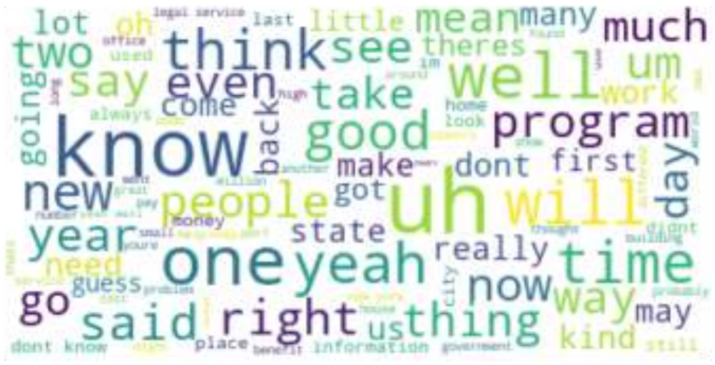

Figure 6: Premises most frequents words cloud

- Hypothesis

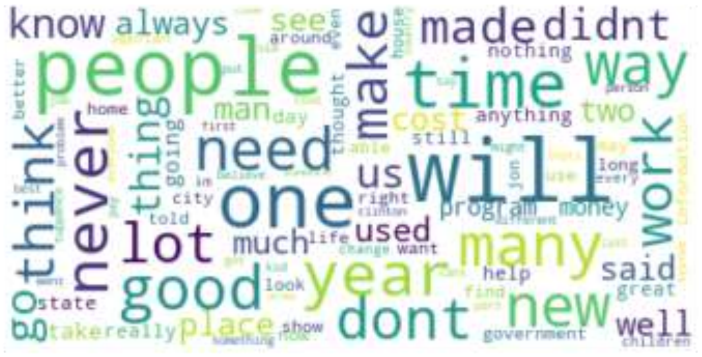

Figure 7: Hypothesis most frequents words cloud

Finally, we have also analyzed the frequency of bigrams and the 3 most frequent with a certain detachment from the others are:

- Don’t know: 89 times
- Legal services: 82 times
- New York: 72 times

Our analysis proceeded with the topic modeling algorithm set to obtain 5 topics and the 10 most important words for each topic, the results we obtained are:

- Topic 0: year, Jon, data, Gao, last, also, two, set, service, national
- Topic 1: one, city, many, around, old, three, also, built, century, small
- Topic 2: legal, would, services, one, state, information, money, two, system, programs
- Topic 3: uh, know, like, yeah, think, well, get, really, would, one
- Topic 4: new, good, life, Clinton, times, made, said, may, long, water

1.3. Similarities with word embedding

During this phase we used the sent_tokenize methods to extract the context and word_tokenize to experiment on the most similar words based on those we provided through the skipgram methodology (we give a word as input and find the context). The algorithms used were Word2Vec and FastText. In the first case we provided as input the words:

LIKE

hate	0.511
Iraq	0.505
feel	0.504
Philadelphia	0.491
pounds	0.491
eagles	0.483
nights	0.464

TIME

planned	0.489
Lloyd	0.476
Danes	0.465
minute	0.461
cranes	0.454
month	0.448
limit	0.446
sibling	0.445

With the FastText algorithm, we tested words instead:

YEAH

yeah	0.693
Uh-huh	0.678
Um-hum	0.666
um	0.659
Um-hum	0.652
Uh-huh	0.651
yep	0.641
tigon	0.608

NEW

York	0.724
newly	0.499
proponent	0.475
news	0.469
full-time	0.469
magazine	0.467
McGrath	0.461
times	0.444

The choice of words given as input was made based on the most frequent words found using the previous methodology. We note that the FastText algorithm has slightly better performance.

2. Data Common Preprocessing

For the preprocessing procedures, we applied multiple tools that varied depending on the strategy or set of algorithms that we decided to apply and that will be detailed with greater precision in the following section, however in general terms among the common data cleaning techniques that we use. They include removing special characters, converting to lowercase, removing numbers, removing unnecessary blank spaces, among others.

In the case of generating the parameter vectors we use different techniques that included tools such as Word2Vec, Doc2Vec, CountVectorizers, BertTokenizers, among several others, including custom tools created for each specific strategy.

3. Data Modeling

3.1. Classic Machine Learning Algorithms.

In this first strategy, we decided to try our classification task with some traditional algorithms such as Random Forest, Naive Bayes Classifier, Kth Nearest Neighbors, SVM Classifiers, among others.

To carry out this classification, we joined each premise and each hypothesis in a single sentence. Then, as embedding methods we used Word2Vec and CountVectorizer, in both cases we have tested the embedding methods both keeping and removing the stopwords. Between the two methods the one that gave us better results is CountVectorizer, so we will report only the results using this tool, with the attribute min_df equal to 100.

For all classifiers we used RandomizerSearchCV algorithm for hyperparameter tuning and cross validation. As scoring ranker for best estimator we used the F1 measure to guarantee a good balance between precision and recall. The only classifier for which this algorithm was not used is the Naïve Bayes because they do not require parameters to be set.

We obtained not so good results as can be expected, sequence classification is a very hard task that requires models to understand the underneath context of the sentences to improve. In general, our classification model got about 30-35% accuracy.

Model	**Average**			
	Accuracy	**Precision**	**Recall**	**F1-Score**
Naïve Bayes	0.399	0.395	0.399	0.387
Linear SVC	0.397	0.404	0.397	0.382
SVC RBF	0.362	0.363	0.362	0.355
Decision Tree	0.394	0.394	0.394	0.391
KNN	0.316	0.311	0.316	0.306
Log. Regression	0.381	0.378	0.381	0.364
Random Forest	0.419	0.428	0.419	0.376

Table 1: Classic Machine Learning Algorithms results

3.2. Long short-term memory (LSTM)

For this model we have followed the following procedure:

- Preprocessing of the data choosing various combinations of the parameters with the function made in the code.
- Model selection with various parameters and complexities, analyzing if it overfit or underfit
- Once we have obtained the best model, we have fitted it to all the data without distinction between validation and training data, and finally check the accuracy on the test set

With this procedure, we have noticed that regarding the choice of the parameters during the preprocessing, removing the punctuation leads the model to not distinguish very well the classes, this means that is an important feature. About the stopwords, removing them leads the model to a better generalization, this is given by the fact that the accuracy got an improvement. Finally, about the lemmatization with POS on verbs, the accuracy got better, so, if the model can understand the meaning of the verbs, it leads to a better distinction of the classes.

Regarding the choice of the parameters of the model, we have noticed how the more complex the model, the more the learning curves results tends to get an overfitting problem, despite various regularization techniques were applied. So for this reason we pointed to a simple model, hence in this way there is not any trace of overfitting.

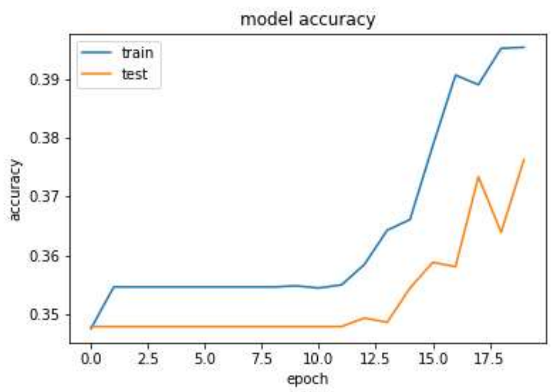

Figure 9: Model Accuracy Plot

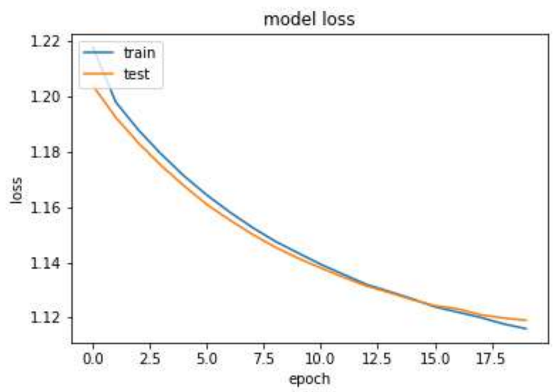

Figure 8: Model Loss Plot

From these plots, we can see, with the validation set, how it can perform with all the three labels. With the test set, the accuracy drops around to 10, but this demonstrates how this model cannot predict so well the data labeled as “neutral”. This is proven even with the Bidirectional-LSTM (which is gotten with the same procedure followed for the classic LSTM)

The explanation for these bad results may be given by the fact that even if the LSTM model takes a sort of memory of the different words in the phrase, it cannot get an attention on specific part of the phrase or words which could lead to a better classification result. In fact, the LSTM models predict almost every sentence as label 0, and this can be seen by their confusion matrices.

3.3. Bidirectional Encoder Representations from Transformers (BERT)

In our third strategy we decided to try it with a last generation model like the BERT family of transformers. For this, the preprocessing process was like the previous trainings exercises, removing capital letters, special characters, stopwords, etc. However, for the tokenization process and embedding, the BERT tokenizer classes provided by HugginFace were used. With these tokenizers we created our embeddings which in the case of BERT must follow a special composition with a classification token (**CLS**) at the beginning of the premise and a separation token at the end of each sentence (**SEP**) as we show in the example below:

- **[CLS]** and she came to you? **[SEP]** the woman asked if he came to her. **[SEP]**

Then, when encoded we get:

- [**101**, 1998, 2016, 2234, 2000, 2017, 1029, **102**, 1996, 2450, 2356, 2065, 2002, 2234, 2000, 2014, 1012, **102**]

In BERT slang this last vector representation is known as "input_ids", and it is the main parameter that BERT models receive. However, this family of models requires that the length of each vector be the same, so it is necessary to perform procedures such as truncation and/or padding to accommodate the maximum length established. Thus, BERT requires a second parameter known as "attention_mask", which represents in the form of 1s and 0s the padding values that the model should ignore. In our case, the length of these vectors varied for the different tests that we carried out, with a minimum value of 30 according to the mean of words in the sentences to avoid excess padding as can be refer in Fig.10, up to a maximum value of 80 that we did in some tests.

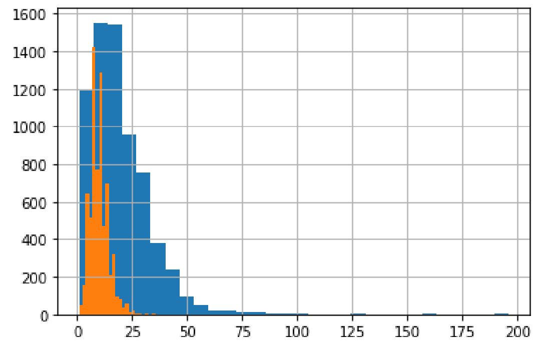

Figure 10: Sentences length distributions

With BERT we carried out many tests, we trained different models such as the BERT base uncased or BERT for sequence classification, but also as these models took a while for the training process, we carried out a lot of tests with lighter versions such as the DistilBert Model and others even more recent transformer models like Electra.

In general terms, we achieve an accuracy up to slightly above 65%, although we understand that it could be achieved around 70%, or even more if external data were included to increase the data set or additional techniques are applied.

In Table 2 we can find a summary of some of the tests we performed.

Model	**Target**	**Precision**	**Recall**	**F1**	**Accuracy**
DistilBertForSequenceClassification	0	0.38	0.06	0.10	35%
Not tuned	1	0.35	0.26	0.20	
About 67MM params	2	0.34	0.74	0.47	
DistilBertForSequenceClassification	1	0.67	0.65	0.66	64%
Tuned Dropouts/Dense layers	2	0.62	0.64	0.63	
About 67MM params	3	0.62	0.62	0.62	
Bert Base Model	1	0.62	0.71	0.68	66%
Tuned Dropouts/GlobalAVGPooling/Dense	2	0.62	0.63	0.63	
About 110MM params	3	0.70	0.63	0.66	
BertForSequenceClassification	1	0.66	0.64	0.65	64%
Tuned Dropouts/Dense layers	2	0.58	0.71	0.64	
About 110MM params	3	0.72	0.58	0.64	
ElectraForSequenceClassification	1	0.73	0.76	0.75	56%
Tuned Dropouts/Dense layers	2	0.53	0.06	0.13	
About 13.5MM params	3	0.45	0.82	0.58	

Table 2: BERT family models results

Conclusions

In this project, we have presented some of the best known strategies for performing text classification tasks. As we could see, as more recent and complex methods are used, better results can be obtained but also different challenges appear, as in the case of LSTM and the difficulty in identifying characteristic elements that allow the data to be classified correctly. Likewise, we were able to see how the models of the BERT family of transformers easily surpass the accuracy of the previous strategies, although at a cost in time and higher processing.

There are two directions that we have in mind regarding future work. We would like to increase the data set including external sources such as the SLNI corpus or some other, which allows our models to find more precise classification boundaries. Likewise, we understand that it is possible to apply better techniques in data preprocessing that allow adding value to vector contexts.