Coronavirus Tweets Sentiment Analysis

Simona Mazzarino
s.mazzarino@studenti.unipi.it
Student ID: 620022

ABSTRACT

In this project, I focused on the sentiment analysis of English tweets, contained in the *covid-19-nlp-text-classification* dataset, using several classification methods, from the simplest ones to the most complex.[1]

1 INTRODUCTION

Nowadays, we live in a world where billions millions of terabytes of text data are produced everyday. Thanks to recent studies we are able to analyze them in several ways and to extract from them knowledge about the world around us. In this project, I used different machine learning techniques to classify some Twitter posts about the recent Covid-19 pandemic, considering the sentiment that they imply. In the first part of the project, the classification considers five different sentiment labels (*Positive, Negative, Extremely Positive, Extremely Negative, Neutral*), while, in the second part, considers just three different labels. The methods used are: Support Vector Machine (SVM), Decision Tree, Random Forest, Naïve Bayes Classifiers (NBC), Convolutional Neural Network (CNN), Long Short Term Memory (LSTM), BERT. The results obtained with these techniques are shown in this report.

2 DATA UNDERSTANDING

Kaggle provides two datasets for the Coronavirus tweets sentiment analysis: one for the training phase and the other for the testing phase. Both datasets are composed by 6 columns:
- *UserName* and *ScreenName*, which refer to the Twitter user who created the tweet;
- *Location* and *TweetAt*, which describe respectively the state or city from which the tweet was posted and the date of posting;
- *OriginalTweet*, which contains the text of the tweet;
- *Sentiment*, which are the sentiment labels linked to every tweet.

The first step was removing missing values to obtain 32567 valid rows for the train set and 2964 rows for the test set. Then, to better understand the dataset, I observed the distribution of tweets for each location (plotting the top 20, Fig.1) and for each day (Fig.2) in the train set and the distribution of sentiment labels in both train and test set (Fig.3). As it can be seen, the labels in both datasets are distributed almost in the same way: the biggest part of tweets are classified as *Neutral*, but the amount of *Positive* tweets is almost the same. *Extremely Negative* is the least frequent label in the considered datasets.

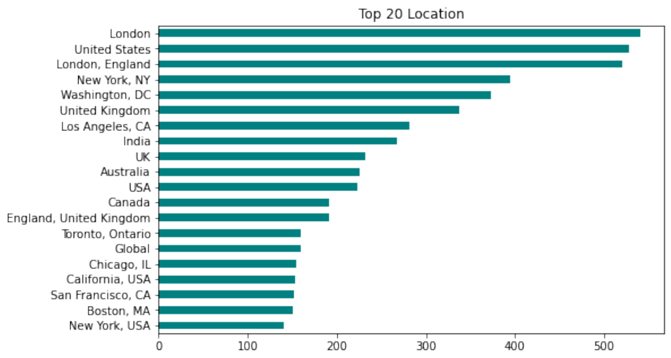

Figure 1: Top 20 Tweets by Location

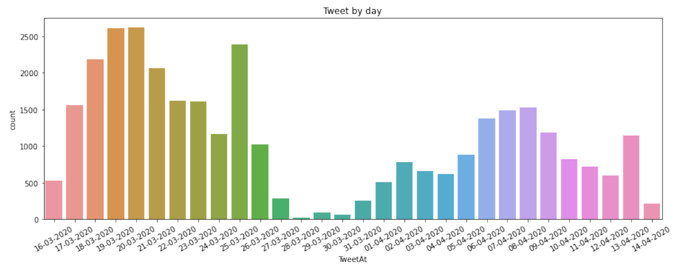

Figure 2: Number of Tweets by Day

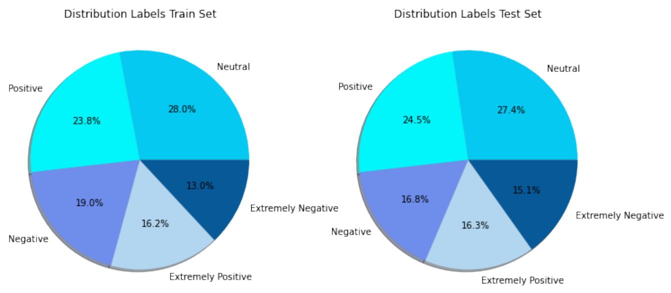

Figure 3: Distribution of Sentiment Labels in both datasets

3 DATA CLASSIFICATION

Before proceeding with the application of several text classification methods, I decided to create 2 different tasks: in the first one, classifiers have to predict the original 5 sentiment labels of the dataset i.e. *Positive, Negative, Extremely Positive, Extremely Negative, Neutral*, while, in the second one, they have to predict just 3 labels, obtained combining the labels *Extremely Positive* and *Positive* in the label *Positive*, the labels *Extremely Negative* and *Negative* in the label *Negative* and maintaining the *Neutral* one, in order to observe if the performance of the classifiers improves decreasing the number of labels.

[1]**Dataset**: https://www.kaggle.com/datatattle/covid-19-nlp-text-classification

	SVM (K=7000, C=1)			**SVM (K=3000, C=10)**		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Extremely Negative	0.59	0.60	0.59	0.60	0.61	0.60
Extremely Positive	0.64	0.63	0.63	0.66	0.63	0.65
Negative	0.53	0.46	0.49	0.57	0.48	0.52
Neutral	0.60	0.73	0.66	0.61	0.79	0.69
Positive	0.51	0.50	0.51	0.52	0.53	0.53

Table 1: SVM (Five Labels) Classification Results

input =	Input(shape=(maxlen,))
x =	Embedding(max_features, embed_size)(input)
x =	Dropout(0.2)(x)
x =	Conv1D(10, 3, padding='valid',activation='relu', strides=1)(x)
x =	GlobalMaxPooling1D()(x)
x =	Dense(64, activation="relu")(x)
x =	Dropout(0.2)(x)
x =	Dense(32, activation="relu")(x)
x =	Dropout(0.2)(x)
x =	Dense(5, activation="softmax")(x)

Table 2: Convolutional Neural Network

3.1 Five-Labels Classification: SVM, Decision Tree, Random Forest, Naïve Bayes Classifier

Firstly, I defined a function to tokenize the tweets using tools from SpaCy library. The function is designed to:
- substitute all space characters with a single space;
- remove URLs and mentions;
- lemmatize and lower case;
- remove stopwords and punctuations;
- create bigrams and trigrams.

I decided to not remove emoji or emoticons from the tweets because they can be involved in and useful for the sentiment classification. Subsequently, in order to proceed with the classification, I used the CountVectorizer, with, as tokenizer, the SpaCy function previously described, to generate the feature space and the vectors for every single token in that space. The result of this application was used as starting point in the pipeline of every classification method reported in this paragraph. Then, it was performed the feature selection using the *Chi2* method and the vectors were weighted with the *TfidfTransformer* (except for the Naïve Bayes Classifier which is based on term frequencies). The first classification method that I tried was the ***Support Vector Classifier***: I used it in two different ways, changing the values of some parameters. In the first case, the classification was done setting the number of selected features at 7000 and using as C value of the *LinearSVC* the default one (C= 1.0). After that, I applied the *GridSearchCV* method on the train set to find the best number of selected features (K) among 3000, 5000, 7000, and the best value for the parameter *C* among 0.01, 0.1, 1, 10, 100. A 7-fold cross-validation was performed on the train set to discover the best values. The best parameters chosen by the optimization method were 3000 for the number of selected features and 10 for C value. The results of these two models applied on the test set are reported in table 1. As it can be seen, using the model with the optimized parameters, there was just a slight improvement of the performance: indeed, the accuracy of the two classifiers differs by just 3 points (0.56 for the first model, 0.59 for the second one). Furthermore, I observed which words affected the choice of the sentiment labels: some words that guided the choice towards a negative label were *deny, undermine, furious, struggle, shame, tragic, hate, collapse, etc.*; instead, the ones that oriented the choice towards a positive label were *thank, care, win, like, help, great, excellent, laugh, etc.* As second classification method, I used a ***Decision Tree*** algorithm whose performances were bad (accuracy: 0.41); therefore, in order to improve them, I decided to run a ***Random Forest Classifier***, choosing with the *GridSearchCV* the right number of features selected (K = 3000) and of trees in the forest (n_estimators = 300). In this way, the performance improved (accuracy: 0.51), but not enough to consider this method a good one. Lastly, I ran a ***Multinomial Naïve Bayes Classifier*** whose accuracy was about 0.49. In conclusion, we can say that the best most performing classifier used so far was the SVM with optimized parameters.

3.2 Five-Labels Classification: CNN, LSTM with and without pre-trained embedding

After performing the classification with the methods mentioned above, I used, as classifiers, a feed-forward neural network, the ***Convolutional Neural Network*** (CNN), and a recurrent one, the ***Long Short Term Memory*** (LSTM). First of all, to do the classification with the neural networks, I did some pre-processing: I re-mapped the original labels in train and test sets with numbers instead of words, then, these new labels were converted in one hot encoding vectors and the values for parameters, for both networks, have been set up in the following way:
- max_features (i.e. the maximum number of words to keep, based on word frequency): 20000;
- embedding_size (i.e. the number of dimension in the embedding vectors): 128;
- max_len (i.e. maximum length of all sequences): 200.

Finally, data were tokenized and transformed to sequences. The CNN used embeddings created from scratch and was designed, after several attempts to avoid overfitting, as it can be seen in Table 2. I chose the *categorical crossentropy* loss function and the *Adam* optimizer.

The same pre-processing was done for the LSTM, defined in Table 3. As previously, I used the *categorical crossentropy* loss function and the *Adam* optimizer. The results of these two networks are reported in table 4. Comparing the results of the CNN with the best classifier of the previous paragraph (i.e. the SVM), we can see an overall improvement, even if the global accuracy of the CNN was 0.57 (two point less than the SVM). Also the LSTM has overcome the performance of the SVM, and, in this case, the accuracy was 0.70.

Next, instead of using embeddings created from scratch, I decided to use two different kinds of pre-trained embeddings: ***FastText***

input =	Input(shape=(maxlen,))
x =	Embedding(max_features, embed_size)(input)
x =	LSTM(60, return_sequences=True,name='lstm_layer')(x)
x =	GlobalMaxPool1D()(x)
x =	Dense(64, activation="relu")(x)
x =	Dropout(0.2)(x)
x =	Dense(5, activation="softmax")(x)

Table 3: Long Short Term Memory

	CNN Embeddings from scratch			**LSTM Embeddings from scratch**		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Extremely Negative	0.72	0.53	0.61	0.77	0.69	0.73
Extremely Positive	0.66	0.73	0.69	0.85	0.70	0.77
Negative	0.69	0.48	0.57	0.72	0.67	0.69
Neutral	0.83	0.72	0.77	0.90	0.76	0.82
Positive	0.63	0.50	0.56	0.69	0.73	0.71

Table 4: Neural Networks (Five Labels) Classification Results

	CNN FastText			**LSTM FastText**		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Extremely Negative	0.78	0.50	0.61	0.79	0.69	0.73
Extremely Positive	0.78	0.50	0.61	0.77	0.75	0.76
Negative	0.68	0.42	0.52	0.72	0.64	0.68
Neutral	0.71	0.77	0.74	0.81	0.78	0.79
Positive	0.70	0.32	0.44	0.68	0.61	0.64

Table 5: Neural Networks with FastText Embeddings (Five Labels) Classification Results

English embedding (i.e. 2 millions word vectors trained on Common Crawl) and ***GloVe***, which are embeddings trained on 2 billions tweets. Both methods presented almost the same number of words without embedding (FastText: 4173, GloVe: 4681) and most of them were (quite obviously) Covid19-related words. The values of the parameters max_features and max_len remained the same, while the embedding size was changed according to the pre-trained embeddings size (300 for the networks trained with FastText, 200 for the ones trained with GloVe). Both networks maintained the same structure as above. With FastText embedding, CNN and LSTM were trained over seven epochs; instead, with GloVe, the CNN was trained over 30 epochs, while the LSTM just over seven again. The results obtained with the FastText and GloVe embedding are reported, respectively, in Table 5 and Table 6.
For both network models, the best classifier was the version with embeddings created from scratch: indeed, even if some precision values of the FastText and GloVe models were higher than the

	CNN GloVe			**LSTM GloVe**		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Extremely Negative	0.68	0.53	0.59	0.66	0.80	0.72
Extremely Positive	0.82	0.40	0.54	0.82	0.61	0.70
Negative	0.65	0.30	0.41	0.68	0.55	0.60
Neutral	0.71	0.71	0.71	0.82	0.73	0.77
Positive	0.58	0.47	0.52	0.63	0.65	0.64

Table 6: Neural Networks with GloVe Embeddings (Five Labels) Classification Results

ones of the 'Embedding from scratch' model, the global accuracy of the models with both pre-trained embeddings was way worse than the accuracy of the models with embedding from scratch. A possible explanation for that could be the lack of embeddings for those Covid-related words, which, probably, affect the sentiment classification.

3.3 Five-Labels Classification: BERT

Last but not least, I implemented the classification with the transformer based algorithm, ***BERT***. First of all, I prepared the text to be processed by the algorithm adding the [CLS] token at the beginning of each sentence and use it as reference token which collect relevant information for the classification. Next, after choosing *bert-base-uncased* as BERT model, sentences were tokenized and the max_length parameter for train and test set was set respectively at 128 and 512. Each token in each sentence was converted into an integer index in BERT vocabulary and any shorter sentences were padded to max_length indices with trailing zeros. Then, an attention mask was defined to distinguish between real tokens and padding. After loading the pre-trained BERT model, the remaining hyperparameters were chosen as follows:
- batch_size: 32;
- epochs: 4;
- weigth_decay: 0.01;
- learning rate: 2e-5.

So, the model was trained and tested, after every epoch, on a validation set. The results obtained on the test set are reported in Table 7. The accuracy was 0.83, the highest value achieved in the five-labels classification task.

3.4 Three-Labels Classification: SVM, Naïve Bayes Classifier

In order to improve the performance of the classification methods implemented so far, I decided to decrease the number of the sentiment labels, merging together the *Positive* and *Extremely Positive* labels and the *Negative* and *Extremely Negative* ones. The distribution of these new labels is shown in Fig.4. Among the classifiers described in 3.1, I chose to use just the method with best performance, the SVM with optimized parameters, and the Multinomial Naïve Bayes Classifier. The pre-processing of data was the same defined above for both algorithms. As it can be seen in Table 8, in

	BERT		
	Precision	Recall	F-1 Score
Extremely Negative	0.77	0.90	0.83
Extremely Positive	0.86	0.88	0.87
Negative	0.81	0.77	0.79
Neutral	0.94	0.84	0.88
Positive	0.80	0.80	0.80

Table 7: BERT model (Five Labels) Classification Results

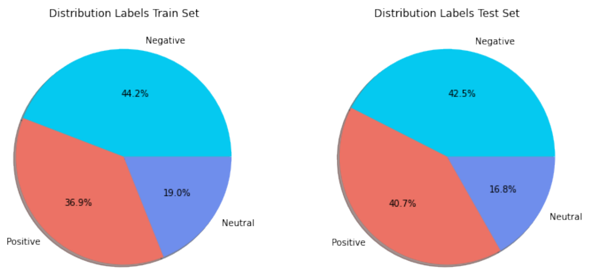

Figure 4: Distribution of New Sentiment Labels in both datasets

	SVM (K=3000, C=10)			NBC		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Negative	0.83	0.79	0.81	0.71	0.71	0.71
Neutral	0.70	0.74	0.72	0.52	0.51	0.52
Positive	0.82	0.84	0.83	0.72	0.72	0.72

Table 8: SVM and NBC (Three Labels) Classification Results

both cases the performance was way better than the five-labels classification (SVM accuracy: 0.80, MultinomialNBC accuracy: 0.68).

3.5 Three-Labels Classification: CNN, LSTM with and without pre-trained embedding

For the three-labels classification with CNN and LSTM (with all different kind of embeddings), I didn't change the structure of the two networks, except for the number of neurons in the last layer (from 5 to 3). The accuracy of the CNN was 0.80, while the accuracy of the LSTM was 0.83. Then, I ran the two networks with FastText and GloVe embeddings. With FastText the CNN was trained over 10 epochs and the LSTM over 4. Also with this configuration, the accuracy values were very good (CNN: 0.77, LSTM: 0.82). Instead, with GloVe embeddings, the CNN, trained over 20 epochs, obtained an accuracy on the test set of 0.75, while the LSTM, trained over 4 epochs, reached an accuracy of 0.81. The results got with FastText and GloVe embeddings are shown in Table 9 and Table 10. So,

	CNN FastText			LSTM FastText		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Negative	0.80	0.83	0.82	0.87	0.82	0.84
Neutral	0.72	0.81	0.76	0.85	0.74	0.79
Positive	0.88	0.71	0.78	0.84	0.86	0.85

Table 9: Neural Networks with FastText Embeddings (Three Labels) Classification Results

	CNN GloVe			LSTM GloVe		
	Precision	Recall	F-1 Score	Precision	Recall	F-1 Score
Negative	0.83	0.74	0.78	0.83	0.85	0.84
Neutral	0.69	0.70	0.70	0.83	0.72	0.77
Positive	0.82	0.77	0.79	0.86	0.82	0.84

Table 10: Neural Networks with GloVe Embeddings (Three Labels) Classification Results

	BERT		
	Precision	Recall	F-1 Score
Negative	0.91	0.90	0.91
Neutral	0.76	0.87	0.81
Positive	0.93	0.89	0.91

Table 11: BERT model (Three Labels) Classification Results

as it can be seen by confronting performances reported in the previous paragraph, by decreasing the number of labels to predict the accuracy of each classifiers increased a lot.

3.6 Three-Labels Classification: BERT

As previously, also in this case, no substantial changes have been made to the configuration of the BERT classifier. The pre-processing steps were the same as above and the chosen parameters as well. Nevertheless, reducing the number of labels the model has an accuracy of 0.89, outperforming the version with five labels and all the three-labels classifiers seen so far. The results are reported in Table 11.

4 CONCLUSION

As has been shown in the previous paragraphs, the five labels classification task presented some problems for the less powerful classifiers. Indeed, just two classifiers had an accuracy higher than 0.70 (the LSTM with embeddings from scratch with 0.71 and BERT with 0.83). Instead, using just three sentiment labels for the classification, the accuracy of each classifier improved a lot: all classifiers, except for the MultinomialNBC, returned an accuracy equal or higher than 0.75. A possible reason for this general improvement can be the number of samples for each label: using three labels, instead of five, the number of train samples for the *Positive* and *Negative* labels increased, facilitating the training. Moreover, in this way, the classifiers didn't have to distinguish between two very similar kinds of tweets such as the *Positive* and the *Extremely Positive* ones (same for the *Negative* and *Extremely Negative* ones), which could have included similar (if not identical) words, making the classification more complicated. In both classification tasks, as expected for the nature of the network itself, the LSTM performed better than CNN because it stores in memory information about the lexical structure of tweets. Indeed, CNN and other non-neural classification methods consider every word as a separate input, so words don't create meaning as a sentence and the class is assigned according to statistics and not according to an actual meaning of the word structure. LSTM, instead, can store in memory information about the structure of the sentence building a global sentence meaning which is used for the classification. After using FastText embeddings (which could be useful to work with tweets because they use ngrams, so they can find an embedding also for that misspelled words, which are typically present in the social networks' language), I decided to use also the GloVe ones to see if the performance of the networks increased, considering that GloVe embeddings are created from a Twitter corpus. However, the performance of the neural networks with GloVe embeddings were worse than the one obtained with FastText embeddings. A reason can be the number of words without embedding: indeed, GloVe had more words without an embedding than FastText. In conclusion, BERT was the best method in both classification tasks with an accuracy of 0.83 for the five labels classification, and an accuracy of 0.89 for the three labels classification.