EURECOM
Sophia Antipolis

Challenge 3 Report
Tweets Sentiment Analysis

Federico Germinario, Andrea Ghiglione, Gabriele Gioetto, Nour Jamoussi

08-06-2022

Abstract

In this report we present our approach to deal with a Sentiment analysis problem. In particular, we work on a realistic scenario using tweets from the dataset *Figure Eight's Data for Everyone platform*. Our aim is to classify Twitter posts with positive, neutral, or negative labels to reflect the feeling of the writer of the tweet. In this purpose, we processed the text and we opted for a Transformer as a model.

1 Problem Overview

1.1 Introduction

The challenge is about extracting the sentiment: negative-neutral-positive of tweets of *Figure Eight's Data for Everyone* platform. The original training data consists of *textID*, *text*, *selected_text* as features and the *sentiment* as a label. We didn't use the selected text for 2 reasons. First, it's provided to handle the bonus part. Second, it implicitly contains information about the sentiment label. Thus, we decided to process only the text and to use it as an input to our model. In general, this kind of problems can be solved by using directly a Transformer without preprocessing. However we adopted a different strategy that consists in preprocessing the text and use it to fine-tune a pretrained language model to classify sentiments. Our aim is to obtain a model which performs well, evaluated in terms of f1 score.

1.2 Data Exploration

The raw data consist in 24732 non-null samples and 3 features *textID*, *text*, *selected_text*. The label is the *sentiment*. It can be either positive, neutral or negative. Figure 1 shows the distribution of the data over the three classes. We can deduce from it that the dataset is almost balanced which spares us of balancing it. Figure 2 presents the most common word in positive, neutral, negative tweets. We can notice that *love* is a keyword of a positive tweet at the opposite of negative tweets where the words *miss* and *sad* are present too.

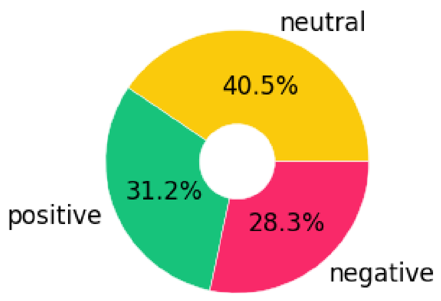

Figure 1: Distribution of the classes

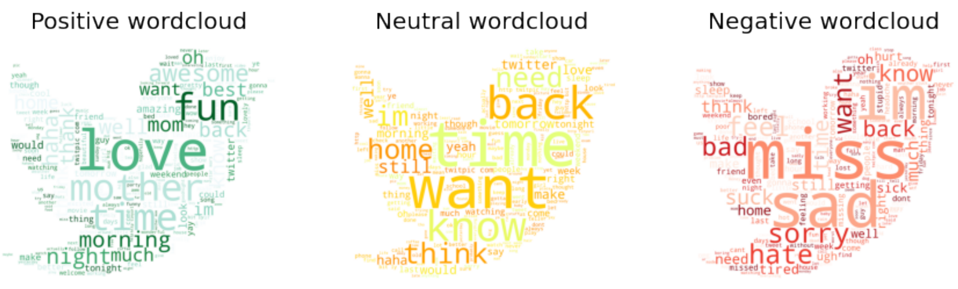

Figure 2: The most common words in each sentiment class

Figure 3 shows the most important mentions present in the tweets. We can notice that the mentions don't add a lot of sentiment to the tweet since they are just other twitter user's accounts. The case of hashtags is different. In fact, some of them can provide additional information about the feeling of the user.
We also tried to explore the existence of any correlation between the number of the words of the tweets or the number of the characters with its sentiment, but we deduced that no correlation is present between these features and the target. Figure 4 makes that fact more explicit. However, we can notice that there is a linear correlation between the number of the words and the number of the characters which is an expected thing. In addition to that, we know that the average tweet's length is around 68 characters.

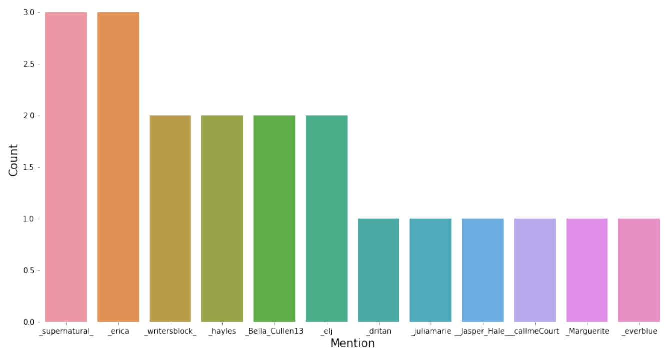

Figure 3: Most Popular Mentions

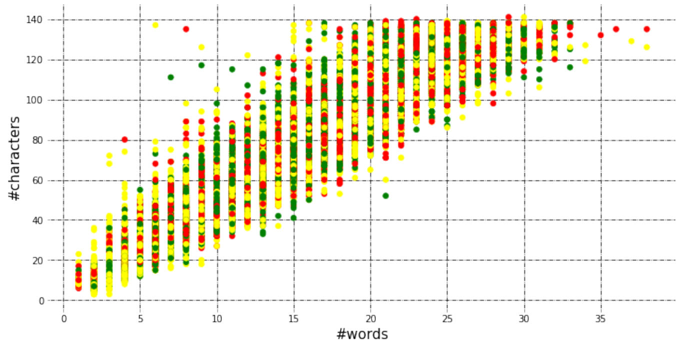

Figure 4: Correlation between number of words and characters and the label of the tweet

2 Proposed Approach

2.1 Data processing

Since tweets are messages which are very heterogeneous, we decided to apply preprocessing to them in order to obtain cleaner tweets, which also means an easier sentiment extraction in most of the cases.
In particular, we remove the links and mentions from the tweets, since they carry noise and do not provide sentiment information. Then, we processed the hashtags in a more complex way: first, if the hashtag consists of an english word we keep it, otherwise we check if the hashtag contains some well-known abbreviation and if it is the case, we replace it with its extendend version. After that we used the *wordninja* library to split the hashtag into English words; this library split concatenated words based on English Wikipedia unigram frequencies. Finally, we check that the split provides real English words only, and in case we substitute the hashtag with the correspondent set of words, otherwise we delete the whole hashtag because it would mean that it lacks of meaning in terms of English vocabulary.
After processing the hashtags we handled the emojis and emoticons, replacing them with actual words. This was very useful since emojis and emoticons are used very often in order to express feelings in text messages. Then, we removed redundant spaces, spaces at beginning and end of a tweet and numbers, since they don't carry sentiment information. Finally, we used the *TextBlob* library to check and correct words which are spelled wrongly, and we lower case all the tweets. Results of preprocessing can be visualized in Table 1

Tweet	**Preprocessed tweet**
Download movie 'Jackass 3' http://tinyurl.com/caotku cool #movie	download movie 'jackass ' cool movie
LOL watching Big Bang Theory latest episode	laughing out loud watching big bang theory latest episode
http://twitpic.com/4jhp8 - Waitin on them 2 tell me what‘s the plan	waiting on them tell me what‘s the plan
Can‘t #tweetb4Ueat	can't tweet before you eat

Table 1: Examples of Preprocessed tweets

2.2 Model Selection

We chose to use a pretrained transformer, in particular we chose the *BERTweet* model. BERTweet is a language model based on RoBERTa, pre-trained on a corpus of 850M tweets. The tokenizer we used was *TweetTokenizer* (provided in the NLTK toolkit), which is the same used to train the BERTweet architecture. The main differences between TweetTokenizer and the standard WordTokenizer is that hastags and emojis are treated differently. Some differences are shown in table 2 with the example sentence: "Snow White and the Seven Degrees #MakeAMovieCold@midnight:)".

TweetTokenizer	['snow', 'white', 'and', 'the', 'seven', 'degrees', '#makeamoviecold', '@midnight', ':)']
WordTokenizer	['Snow', 'White', 'and', 'the', 'Seven', 'Degrees', '#', 'MakeAMovieCold', '@', 'midnight', ':', ')']

Table 2: Differences between TweetTokenizer and WordTokenizer with an example sentence

We finetuned the model on our training set, which is composed by 80% of the original dataset. We chose to use the ADAM optimizer and the Cross Entropy Loss.

The hyperparameters chosen are:

- max length : 128
- batch size : 32
- learning rate : 2e-5
- epsilon : 1e-8
- epochs : 1

The max length, which is the maximum amount of token in a sentence, was chosen as little as possible not to truncate sentences. While the batch size was chosen as high as possible in order to speed up the training phase. The reason of having only one epoch is that the model started to overfit after the first epoch.
To understand which words affect more our predictions we implemented a gradient-based analysis. We can rely on gradients to infer how changes in the output are influenced by changes in the input. Concretely, we compute the gradient of the predicted output class with respect to the input tokens.
In the original implementation of the method each token was treated separately, but we thought that the words that have been splitted by the tokenizer would have given better results if treated together (Ex. tokens: ["Let", " ' ", "s"] becomes ["Let's"]). To summarize the explainability score of the original tokens we first tried to average the scores, but a lower score in a word out of all the words in the final token was compromising the results. Therefore we instead decided to get the maximum out of all the scores (Ex. ["Mother", " ' ", "s"] with scores [1, 0.05, 0.12] becomes "Mother's" with score 1). We maintained the separator tokens of BERT, however in most of the cases they have low scores.
Then we defined a set of thresholds of explainability scores, and we filtered the words below the threshold. Lastly we computer the BLEU score between the obtained sentence and the *selected text* column. The threshold used are [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]

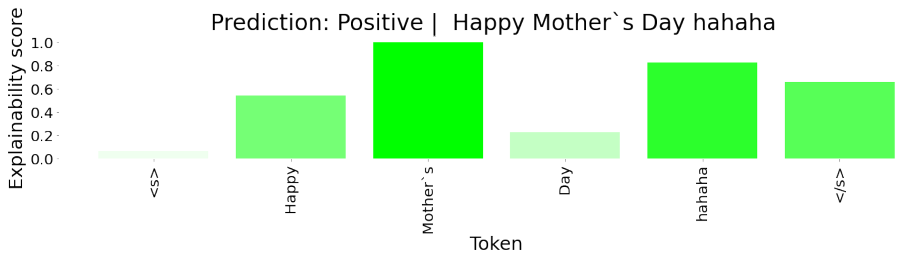

Figure 5: Interpretability results for a positive tweet

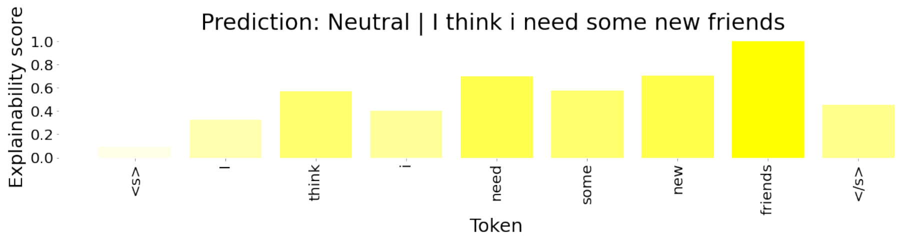

Figure 6: Interpretability results for a neutral tweet

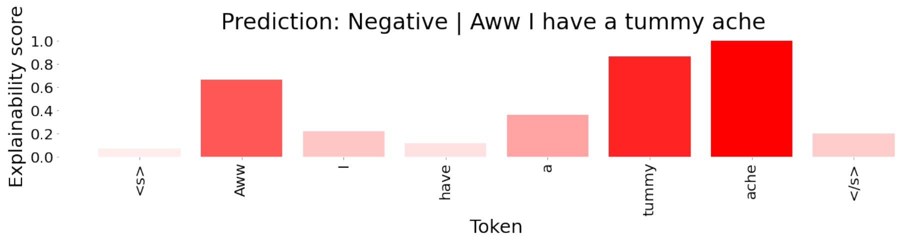

Figure 7: Interpretability results for a negative tweet

3 Results

3.1 Numerical results

Using our model we got an accuracy on the training set equal to 0.79 and an accuracy on the validation set of 0.80. We noticed from the confusion matrix that most of the errors we obtained come from neutral sentences that are predicted as negative. One possible solution could be to change the threshold to decide if the output of the BERT sequence classifier is negative or neutral (right now we use a simple argmax)
For the interpretability part, we obtained the best BLEU scores using the threshold 0.5, for which we get the following results. A lot of processed sentences have a really low score, probably because picking token based on a threshold is not an optimal method.

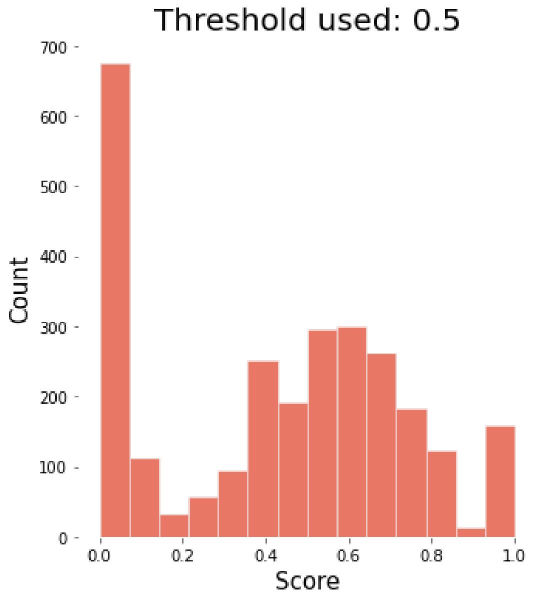

Figure 8: BLEU scores

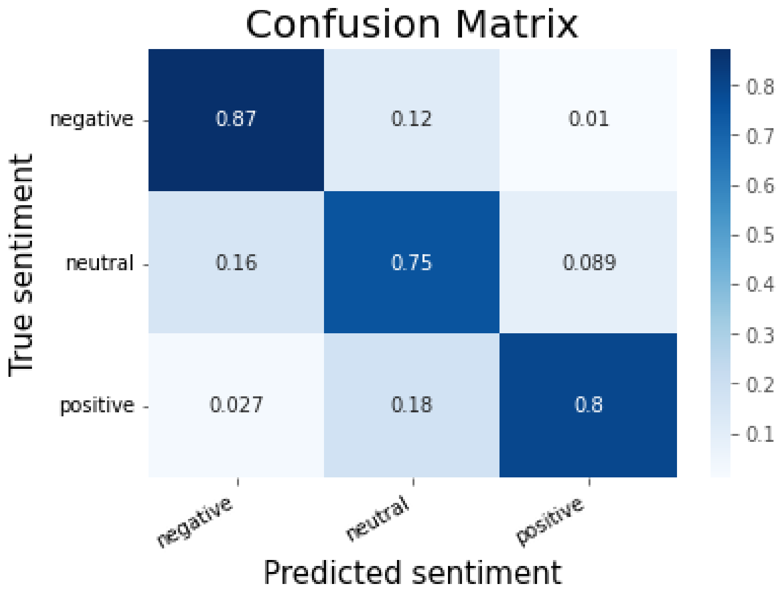

Figure 9: Confusion Matrix

3.2 Limits and merits of the model

The biggest advantages of transformers with respect to more classical model such as RNNs and LSTMs are that they are parallelizable and they don't suffer from the vanishing gradients problem thanks to the Attention paradigm.
However, the transformers' training can be very time consuming. This is the reason why we opted for a pre-trained language model, which needed to be fine-tuned on our classification task.