Sentiment Analysis

LEONE, Andrea
Politecnico di Torino
Institut EURECOM
leone@eurecom.fr

BONINO, Giulia
Politecnico di Torino
Institut EURECOM
bonino@eurecom.fr

NEPOTE, Luca
Politecnico di Torino
Institut EURECOM
nepote@eurecom.fr

August 29, 2024

Abstract

This project addresses the task of sentiment analysis on a tweets dataset, with the goal of classifying them as *negative*, *neutral* or *positive*. This task finds a huge variety of applications, like inspecting the overall sentiment of a user-base about some topic, in order to take targeted marketing actions. We are going to compare different machine learning models for this purpose: a logistic regressor, a custom transformer and a fine tuned BERT-model.

1 Introduction

Sentiment analysis is a subfield of Natural Language Processing (NLP) involving the classification of the polarity of some text. A popular application of sentiment analysis in the last twenty years has been in the context of social media. In these platforms (like Instagram, Facebook and Twitter), we can find a large variety of textual data, which can be used to analyze and interpret how the general public feels about certain topics. For example, sentiment analysis could be used by an enterprise to understand people's opinions on their products and their marketing strategies.

In this work, we apply sentiment analysis to text taken from Twitter, more precisely from the "Figure Eight's Data for Everyone" platform [1]. The goal is to classify each tweet as one of the following categories: positive, neutral, negative.
We perform an analysis of the dataset and implement some pre-processing techniques on the samples. Afterwards we present different models to perform the task: a Logistic Regressor as baseline model, a Custom Transformer and a fine-tuned BERT model.
We present and discuss the results of all three models.

2 Data analysis

2.1 Dataset description

We are provided with a labeled training dataset consisting of tweets a tweet ID and the associated text. We then have the associated label (*negative*, *neutral* or *positive*) and some *selected text*, which we would like to output together with the label to indicate which part of the text better represents the tweet sentiment.

We are also provided with an unlabeled test dataset, for which we would like to classify the samples and possibly extract a significant portion of text that emphasizes the sentiment of the tweet. We will not use this dataset for our experiments.

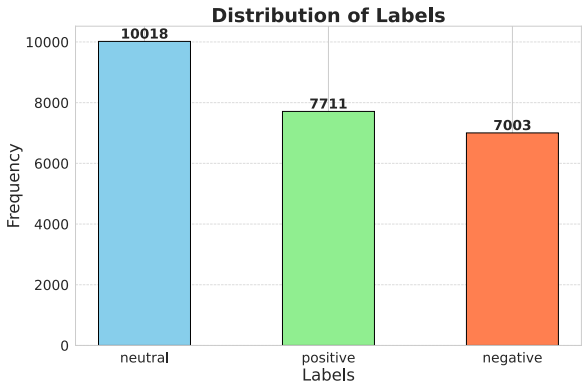
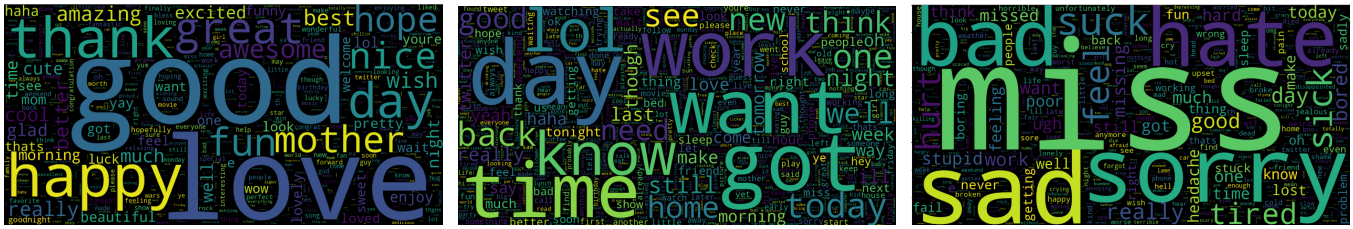


Figure 1: Distribution of labels

2.2 Dataset Exploration

In Fig 1, we can observe the frequency of the labels in the dataset. We note that the most frequent class is *neutral*, while the other two classes (*positive*, *negative*) have similar frequencies.

For each label, we plotted a WordCloud to show the most frequent words in each sentiment class. Note that,



(a) Positive Class (b) Neutral Class (c) Negative Class

Figure 2: WordClouds graphs for the three different sentiment classes

before producing the graphs, we performed some simple preprocessing techniques to have a more meaningful representation: we converted the text to lowercase, removed the stopwords and removed the punctuation. In Fig 2 we show the three WordClouds.

3 Data processing

As the pre-processing for the dataset is very different depending on the models we implemented, we will discuss it further in the dedicated sub-paragraphs in Section 4.

4 Model selection

In this section, we analyzed different models that we tested for our purpose and we selected the best one. The proposed models are:

- Logistic Regressor as baseline;
- Custom Transformer;
- Fine-tuned pretrained BERT model.

For evaluating the models we used the *validation loss* and the *f1-score*, looking for an optimal balance between precision and recall.

4.1 Logistic Regression

We implemented a Logistic Regressor as baseline model.

Dataset preparation: We performed the following pre-processing steps on each tweet:

- we converted all the characters to lowercase;
- we extended the contractions (e.g. *can't* $\rightarrow$ *can not*);
- we removed the stopwords. Note that we didn't remove the word *not* from the tweets, because it was meaningful for our sentiment analysis;
- we removed the punctuation;
- we lemmatize the words (e.g. *going* $\rightarrow$ *go*).

From this process we obtain a list of tokens for each sample, then we vectorized the tokens using TF-IDF.

Description: We tuned the C hyperparameter for the Logistic Regressor. Note that we performed a train-val-test split with percentages (70%, 20%, 10%) using the same seed for reproducibility, and the same split has been done for all the other models too.

Results: In the following table, we present the results of the hyperparameter tuning, computed on the validation set.

	Value of C			
	0.5	**1**	**2**	**3**
F1 Score	0.6842	0.6941	0.6967	**0.6976**

Table 1: Results of grid-search on the value of C

Given this results, we chose the value C=3, which gives an F1-score of **0.6967**.

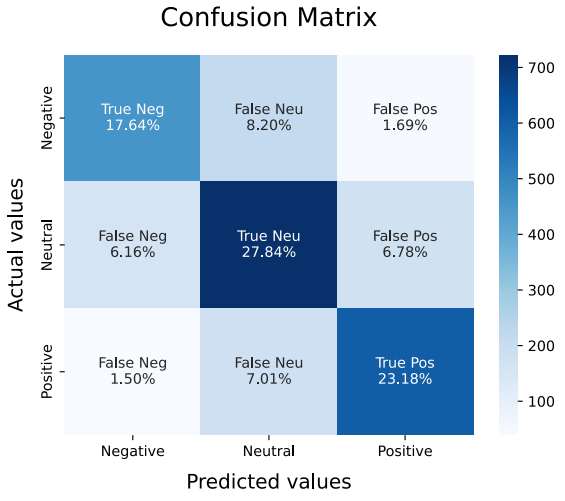


Figure 3: Confusion matrix for Logistic Regressor

In Fig 3, we show the confusion matrix for the prediction on the test set using this model. We can observe that there is a preference for predictions of the class *neutral*,

in fact, more than 40 % of the samples are classified with this sentiment class. This could be due to the unbalanced dataset, which injects a bias towards the neutral class into the model.

4.2 Custom Transformer

Let's now analyze the performance of a transformer-based sentiment analysis classifier.

Data preparation: The data has been preprocessed by removing URLs and emojis from the tweets, converting the text to lowercase and expanding contractions. Then, punctuation has been remoeved and the resulting string has been split into word tokens. Each token has then been stemmed with the PorterStemmer [5] and stop-words have been removed.
We build a vocabulary based on the terms appearing in the training set, so some terms of the test set will never be seen by the model, and we will substitute them with an *UNK* token. This token is used also for padding each sample up to a block size of 32. Given that the test data will have some *UNK* token in the middle of the sentence, we insert the padding at random positions in the text in order to get a more robust model. This also leaves doors open to explore with data augmentation later.

Model description: The model takes as input a batch of token IDs sequences with a length of 32, and computes the token embeddings as a sum of actual token embeddings and position embeddings. Then feeds this into a *TransformerEncoder* consisting of different Layers [8] of self-attention. Finally, the result is average-pooled and fed into a fully connected layer giving out three scores, one for each class.

We have experimented with different learning rates, embedding size, number of heads and number of attention layers. We also noted that the model tends to overfit after the very first epochs, so we set the maximum number of epochs to 15.

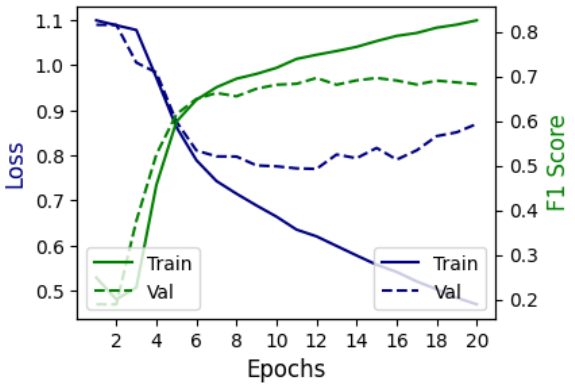


Figure 4: Custom Transformer training history

Results: The optimal learning rate among 10^{-4}, $5 \cdot 10^{-4}$ and 10^{-3} is $5 \cdot 10^{-4}$. Also, we noticed that using too many attention layers gets the model significantly worse at some point. The grid search has been done with the following hyperparameters:

- Embedding size: 24, 36, 48
- Number of heads: 4, 6
- Number of attention layers: 3, 4, 5, 6

The best results have been obtained with embedding size 36, 4 attention head and 4 attention layers, with an F1-score of **0.7022** (validation loss = 0.7476). Figure 4 shows the training history of such model for 20 epochs. The model saturates after the first 6 epochs and slightly improves until epoch 10-12.

Figure 5 shows the confusion matrix obtained on the test set with such model.

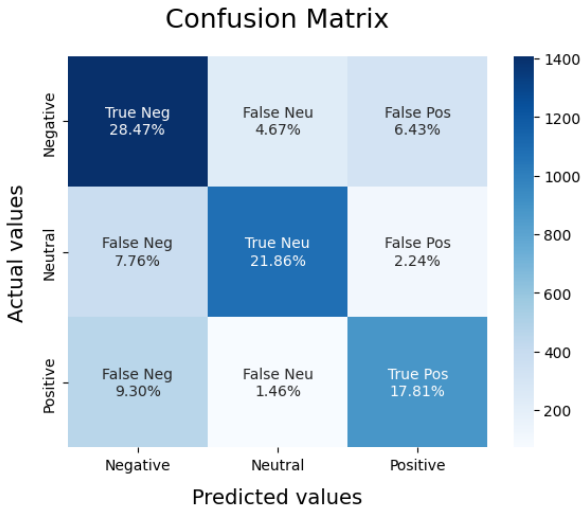


Figure 5: Custom Transformer confusion matrix

Data Augmentation: Since the padding is added at random, we tried to duplicate and even triplicate the samples of the training dataset relying on the fact that after the preprocessing the copies will result in different input tensors for the model. However, this process didn't give any improvement in terms of validation loss or F1-score.

Solving "unseen" words: The fact that at test time the model is receiving as input some terms that have never been seen during training (and so they have been replaced with *UNK*), is certainly having a bad impact on the model. In fact, a big downside of using a custom model with respect to BERT, is that the size of the training dataset is limited and the model will be trained only with the terms appearing in the dataset, which may not be exhaustive. Because of this, we tried to use GloVe

[6] pretrained word vectors (50-dimensional) as an embedding table, with zero-padding at random positions as explained before. To enhance the model performances we also changed the preprocessing: we left the stopwords in their place and avoided stemming, since GloVe provides embedding vectors for non-stemmed words. Furthermore, we designed some ad-hoc function to fix duplicate characters (*haaaaappyyy* becomes *happy*) and non-spaced words (*verygood* becomes *very good*).

Before, about 28% of the test dataset vocabulary was not contained in the training dataset vocabulary. Now instead, only 4% of the vocabulary is "unknown", so we already expect some improvements because of this.

The updated model takes as input a batch of tweets where each tweet is represented as a vector of 36 token embeddings (we needed to increase the block size since stopwords are not removed). In this case, the embedding size is fixed to 50, and the only valid and reasonable values for the number of heads are 5 and 10. We stick to 5 which is the value nearer to the optimal we found before and we also use 4 as number of attention layer. We run a single training with these hyperparameters and a learning rate of $5 \cdot 10^{-4}$, getting an F1-score of **0.7440** and a validation loss of 0.6505 after 11 training epochs. The relative confusion matrix is reported in Figure 6

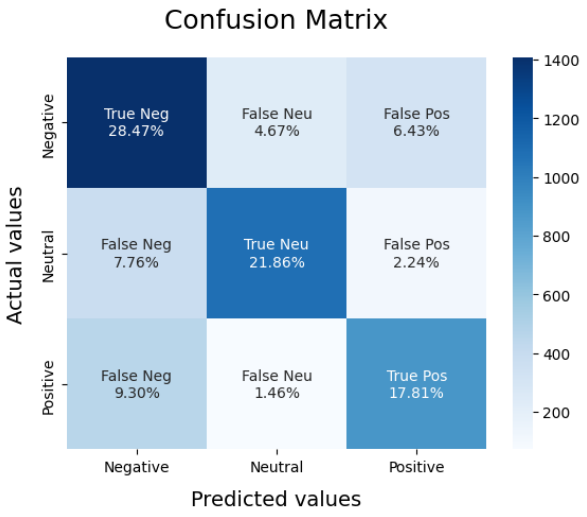


Figure 6: Custom Transformer confusion matrix (pretrained embeddings)

In this way, we managed to achieve some improvements with respect to the baseline model. Changing the hyperparameters of the model with pretrained embeddings doesn't help further.

4.3 BERT

Description: We tried also a fine tuned BERT ("Bidirectional Encoder Representations from Transformers") model [4]. This is able to capture the correlation between the words, considering the context, in both sentence directions.
Moreover, it is very easy to fine-tuned because we have just to add a final classification layer for the specific purpose.
Notice that we tried different pretrained BERT models, such as the "cased" and "uncased" versions, to which we add a dropout and a 3-class classification layers for sentiment classification.

Data Preparation: Considering the data preprocessing, we used the tokenizer in order to assign tokens to the words present in the sentence. Because we are dealing with tweets, that usually are small, we fixed an intermediate *max_length* parameter equal to 128, in order to represent the distribution of the assigned tokens. Moreover, we ensure that the tweet is not bigger than the maximum length setting the *Truncation* parameter as **True**.
Then, we noticed that the number of assigned tokens to each tweet was never bigger than 70 (Fig 7), and we set the *MAX_LEN* parameter to this value.

We also enabled the special tokens and the padding,

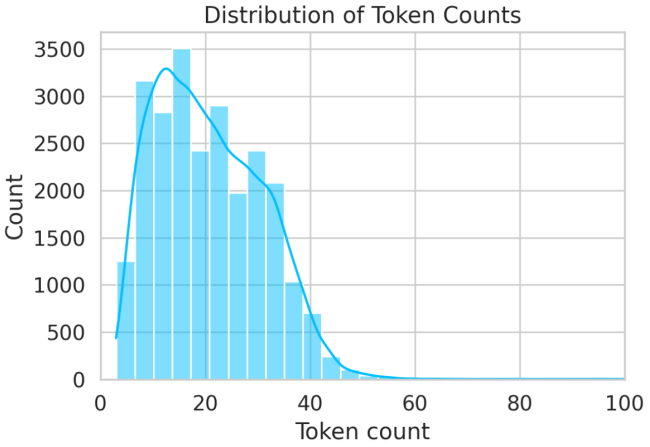


Figure 7: Token Distribution

and, starting from the original train dataset, we split it in 70-20-10, respectively for training, validation and test.

Data Augmentation experiments: Since we noticed that the dataset was unbalanced, with a majority of *neutral* samples, we tried to mitigate this imbalance using SMOTE[3].
Synthetic Minority Oversampling Technique (SMOTE) creates new synthetic examples for the minority classes. It works as follows: we select a random vector (v_A) from the minority class, we find the K closest vectors to it (us-

	UNCASED	CASED
f1-score	**0.7892**	0.7676

Table 2: F1-score on validation set

ing KNN) and we select one of them at random (v_B), we draw a line between v_A and v_B and we get a random point (v_C) on this line. Finally, we insert v_C in our dataset.
In order to use this technique, first we tokenized the samples using the BERT tokenizer.
Unfortunately, also this data augmentation process didn't yield meaningful results in terms of F1 score, so we didn't take it into account for the selected model.

Results: Setting the learning rate to $2 \cdot 10^{-5}$ and running for 10 epochs, the best model was the one using the uncased version of BERT, as shown in Table 2, for which we reported the plots.

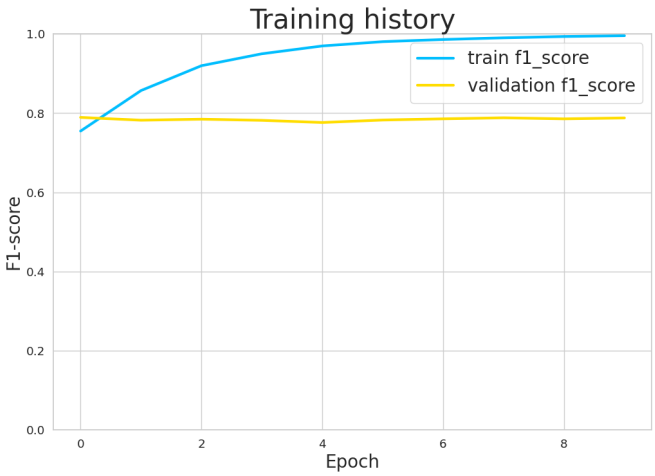


Figure 8: BERT Training History

From Fig 8 we can clearly see how the model is overfitting over the training data, but it has still good f1-score on the validation set, around 79%. This is due to the very high number of parameters inside the model, that it is able to learn in few epochs.

We report also the confusion matrix, representing the percentages of the predicted values with respect to the actual ones. As we can see from Fig 9, the model is able to correctly label most of the tweets in the dataset.
Finally, we evaluate the model on the test set, obtaining an F1-score of **0.7741**.

4.4 Model choice

Considering all the achieved results from all the models, the best one is the fine-tuned pretrained BERT model, starting from the uncased version.

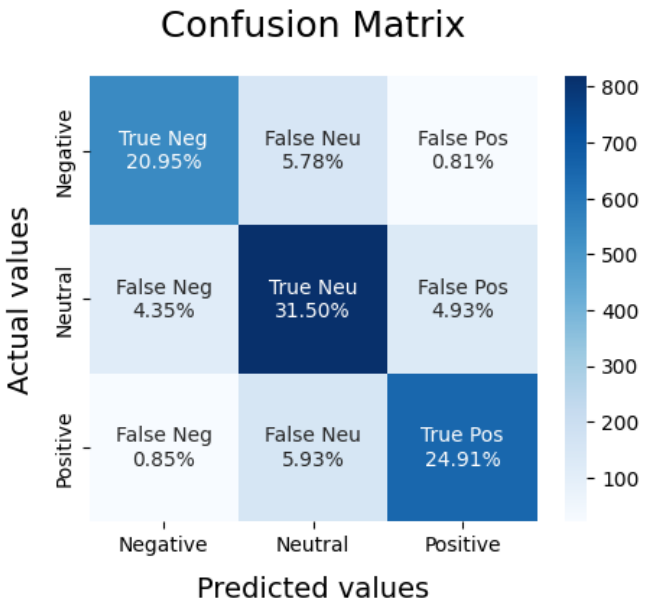


Figure 9: BERT Confusion Matrix

However, we want to highlight that the results are not surprisingly good (under the 80%): this is due to the high complexity of the problem, for which it is difficult to correctly classify the sentiment present in a tweet.

5 Experiments & Future Works

5.1 Extracting relevant text

For the bonus task, we experimented with a tool that extracts and assigns a relevance score to the tokens in the sample, based on the BERT model. The tool is called **transformers interpret** [7].

This procedure could be useful in a real-world scenario, where it is needed to interpret the output of a complicated and "black-box" model.
We selected 200 random samples, then we extracted the scores for each word, considering the scores assigned to our predicted sentiment label. We sorted the scores in decreasing order and selected the first n scores such that the sum was less than a threshold.
For the threshold, we did some manual tuning, taking care that we didn't select too many or too few words on average.
Since we were given the ground truth column *selected text*, which contains the actual more meaningful words for each sample, we compared it with our predicted most relevant words. We compared the two sets of words using the Jaccard Coefficient which is defined as the ratio between the intersection of two sets over their union. The resulting

score is **J = 0.2047**.

This experiment shows that the proposed method requires further testing and tuning. For example, the threshold used in the word extraction process might not be optimal, leading to either too many or too few words being classified as relevant.

5.2 Future Works

One possibility to increase the model performance could be using a "Decoder-Only Transformer" [2]: this is a traditional transformer but, instead of using a self attention layer for all the sequence, we use the "causal self attention" version, masking out the tokens following a given word inside the sequence.
Moreover, we used different attention heads, which all use the same masked self attention: in this way every head is able to learn and focus on different parts of the underlying sequence.

6 Conclusions

In this work we dealt with the sentiment analysis problem on a tweet dataset, and our goal was to classify tweets as positive, neutral or negative. We experimented with different machine learning models, starting from a logistic regressor baseline and studying a custom transformer and the pre-trained model BERT.

In conclusion, the best results were obtained with the fine tuned BERT-model, which obtains an F1-score of **0.7741**, and is therefore the model we recommend for dealing with this task.

We also proposed some further improvements to experiment on in order to increase the performances in terms of F1-score.

References

[1] Appen. Figure eight inc. URL: `https://en.wikipedia.org/wiki/Figure_Eight_Inc.`

[2] PhD Cameron R. Wolfe. Decoder-only transformer. URL: `https://cameronrwolfe.substack.com/p/decoder-only-transformers-the-workhorse`.

[3] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] NLTK. `https://www.nltk.org/api/nltk.stem.porter.html`.

[6] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL: `http://www.aclweb.org/anthology/D14-1162`.

[7] Charles Pierse. URL: `https://github.com/cdpierse/transformers-interpret`.

[8] PyTorch. `https://pytorch.org/docs/stable/generated/torch.nn.TransformerEncoderLayer.html`.