# AML - Challenge 3: Sentiment Analysis

Francesco Dente, Michele Ferrero, Niccolò Zanieri

May 2024

## 1 Introduction

*Sentiment analysis* is typically defined as the process of associating a specific emotional tone to a given sentence and labeling it as either positive, negative, or neutral. A brief online search will reveal the growing interest in this subject in recent years. Indeed, a search of Google Scholar[3] for articles containing the words "sentiment analysis" in the title reveals that, over the past four years, more articles on the subject have been published than in the 20 years from 2000 to 2019.
In our work, we will examine the performance of two different models: *DistilBERT*[5] to which we append a classification layer and *Logistic regression*

## 2 Data Analysis

### 2.1 Dataset

The dataset we worked on consists of about 24000 tweets taken from **Figure Eight's Data for Everyone platform**, each associated with a label indicating whether the sentence is *positive*, *neutral*, or *negative*. We also have access to the ground truth indicating the most important words in each tweet that contribute to the sentiment of the sentence.

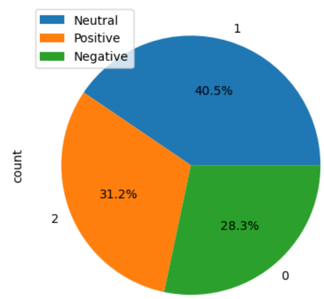


Figure 1: Label distribution of tweets


Figure 2: Figure representing most frequent words in the positive and negative classes

### 2.2 Data preprocessing

Since we are dealing with tweets, we expect preprocessing techniques to be helpful for our analysis [1]. We now present the main preprocessing techniques we used in our work, more detailed information is described inside Section 3:

- **Tweet related preprocessing**: we remove usernames, URLs [2], and convert emojis and emoticons into words.
- **Stopword and punctuation removal**: Stopwords and punctuation mostly do not contribute meaningful information for our task, by removing them, the model can focus on the more relevant words, potentially improving performance.

[1]Particularly when using simple models with Bag of Words features rather than transformers-based solutions
[2]Most likely, these are words that will never be seen again by our model after training

- **Stemming and lowercasing**: we reduce words to their base or root form by removing suffixes and sometimes prefixes and then we lowercase them. The main goal is to reduce the vocabulary size.

# 3 Model selection

Now we will analyze the performance of different machine learning models for our task by using different scores with different meanings. First, we will analyze the sentiment analysis task using **Macro F1-score** as a metric, then we will explore how to use our models to select the most important words for the sentiment of the sentence and measure the performance using **Jaccard scores**[3].

We will explore two possible approaches for solving our task:

- **Non Deep approach** using a *Logistic Regression* as a baseline model

- **Deep approach** finetuning *DistilBERT* pretrained transformer model.

## 3.1 Training Procedure

We split our dataset into train-validation-test (70%, 20%, 10%) maintaining the same balance between classes as in the starting data. We use the validation dataset to check for overfitting and to tune the hyperparameters of our models.

Then, we retrain our chosen model on both train and validation and we evaluate on the test set to see how well our model generalized.

## 3.2 Logistic Regression

We start by training a simple baseline logistic regression model.

### 3.2.1 Feature Extraction

We start with a simple bag of words model. A bag of words simply assigns an index to each word in our vocabulary, and embeds each sentence as a list of 0s, with a 1 at each index corresponding to a word present in the sentence.

In order to make the vocabulary size manageable, preprocessing the tweets is crucial, so we use all the techniques described in section 2.2.

**Example of features extraction:**

- **Original tweet**: Hmm..You can't judge a book by looking at its cover

- **Processed tweet**: ['hmm', '..', 'judg', 'book', 'look', 'cover']

### 3.2.2 Model validation

As we can see in Figure 1, our data set is unbalanced, so we expect that taking this into account within the *logistic regression* by adjusting the weights inversely proportional to the class frequencies in the input data will help our model.

We then perform cross-validation to choose the value for the L2 regularization term, and we obtain that the best model is with C=0.1[4], which obtains a **F1-score** of **0.687** on the validation set.

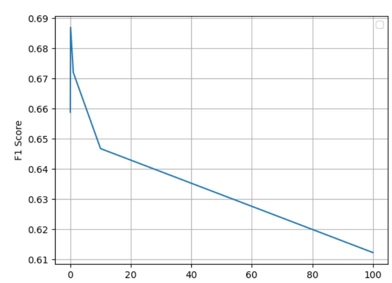


Figure 3: F1-score obtained using different values of C

### 3.2.3 Model explainability

A big advantage of using non deep models is that it is easy to explain why a model makes certain decisions. In this case, just by looking at the weights of the decision function of our model, we can see which are the most important words for each class (*Positive*, *Neutral*, *Negative*). It's interesting to notice that, as we can see in figure 4, when it comes to the neutral class, many of the least important words are among the most important words for the positive and negative classes.

[3] Intersection over union of two sets of words
[4] Inverse of regularization strength, smaller values specify stronger regularization


Figure 4: Most important words for negative (top-left) and positive (top-right) class. Least important words for neutral class (bottom).

#### 3.2.4 Keywords extraction

We can also try to use our model to predict the most important words to describe the sentiment of our sentence, and measure its performance using Jaccard scores [5].

We rescale the weights of the decision function of our model into the range [0,1], and then for each sentence select only those words whose weight for the predicted class is above a certain threshold, which we can fine-tune as a hyperparameter using our validation set. It turns out that we get the best results (**Jaccard score** of **0.585**) by always selecting the whole preprocessed input text as the predicted selected text. This suggests that the preprocessing techniques alone (without using the information learned by the weights of our model) can already distinguish important words in our text, and that our described technique may be too naive, since we only consider the weights related to the predicted class[6].

### 3.3 DistilBERT

We now use *DistilBERT*, which is a small, fast, cheap and light Transformer model trained by distilling *BERT*[2] base. It has 40% less parameters than google-bert/bert-base-uncased, runs 60% faster while preserving over 95% of *BERT*'s performances as measured on the GLUE language understanding benchmark.

We add a classification head and fine-tune it on our small dataset for our sentiment analysis task. Since the dataset is quite small, and *BERT* has millions of parameters and can therefore quickly adapt to all the nuances (and even noise) of the dataset, we choose the smaller *DistilBERT* version (less parameters) to limit the overfitting problem. Furthermore, this smaller version is able to fit our computational constraints on the Kaggle platform.

#### 3.3.1 Model validation

We then perform validation to choose the best preprocessing techniques. It turns out that using more aggressive preprocessing techniques like **stopword/punctuation removal** or **stemming/lemmatization** only leads to worse results. This is to be expected, since such a complex pre-trained model can exploit almost any part of a sentence to produce interesting and useful representations.

We get the best **F1-score** of **0.797** on the validation set by simply removing usernames and URLs from the original tweets.

#### 3.3.2 Model explainability

For this model, the explainability part is more complicated, but we could go deeper into it using a specialized library[1] that uses Captum[4] under the hood. The latter is a specialized library created for PyTorch by Facebook AI with various functionalities. Among them, the transforms-interpret library uses the Layer Attribution functionality[5] which contains various algorithms used for understanding how the input of a layer can influence its output; this can be used to compute the weights for each token as shown in figure 6 as colors.

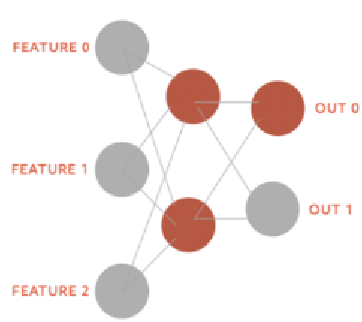


Figure 5: Layer Attribution Algorithm

[5] We perform stemming on the words within the predicted and true sets before computing the score

[6] In multiclass logistic regression we have a weight vector for each class

This allows us to look at this interesting example, which clearly shows the different behavior between transformer models and simple Bag of Words models. If we take the sentence *"The dinner yesterday was not so amazing"* we can see in Figure 6 how the model is able to select *"was not so amazing"* and correctly label the sentence as negative because of the presence of *not*. On the other hand, our LR-BoG model misclassifies as positive the sentence because the features extracted from the tweet are only the words *dinner* and *amazing*, which, without the *not*, is mainly a positive word.

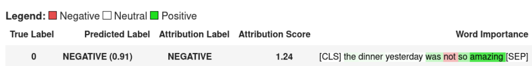


Figure 6: phrase explained by *Captum (transformers-interpret)*

#### 3.3.3 Keywords extraction

We now try to predict the most important words to describe the sentiment of the sentence according to our model and measure its performance using **Jaccard scores**. We use the token weights given in the output of the *transformers-interpet* explainer, and we use a threshold (fine-tuned over a validation set) to decide whether a token should be considered important or not.

We get a very low **Jaccard score** of **0.289**. The main problem is that, even though we use the same tokenizer, some words are divided in chunks by the explainer, leading to a different word parsing between predicted text and the ground truth, thus to low Jaccard scores.

As a future improvement, it's possible to try to use metrics that are less susceptible to tokenization problems to evaluate the selected text, like using a pre-trained Bert model to convert both the true selected and our predicted one to the embedding space and then compare them using a cosine similarity score (given the properties of the space, two semantically similar strings should be close). This process takes a lot more resources, but it's more accurate.

### 3.4 Model choice

| | **BoG-LR** | **DistilBERT** |
|---|---|---|
| **F1 on test** | 0.695 | 0.810 |

We now compare our two models and see that the deep model clearly outperforms our baseline logistic regression. This is to be expected, as pre-trained transformer models have proven to be very effective in NLP tasks such as sentiment analysis.

## 4 Conclusions

In this report, we explored sentiment analysis using both a non-deep approach with Logistic Regression (BoG-LR) and a deep learning approach with DistilBERT. While Logistic Regression offers advantages in model interpretability and ease of explaining decisions based on word importance, DistilBERT significantly outperforms it in terms of classification performance, achieving a higher F1-score. Future work could focus on improving keyword extraction techniques and exploring alternative evaluation metrics to address tokenization issues.

## References


[1] cdpierse. 'transformers-interpret repository'.

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.

[3] Google. Google Scholar. Last accessed 6 June 2024.

[4] Narine Kokhlikyan, Vivek Miglani, Miguel Martin, Edward Wang, Bilal Alsallakh, Jonathan Reynolds, Alexander Melnikov, Natalia Kliushkina, Carlos Araya, Siqi Yan, and Orion Reblitz-Richardson. Captum: A unified and generic model interpretability library for pytorch, 2020.

[5] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter, 2020.