Classification on Twitter posts

Data Science Lab

Davide Bartoletti
Politecnico di Torino
Student ID: s296091
s296091@studenti.polito.it

Domenico Bulfamante
Politecnico di Torino
Student ID: s301780
s301780@studenti.polito.it

Abstract - In this report we propose an approach for sentiment analysis classification based on Twitter posts. The chosen approach embeds data analysis, preprocessing and feature extraction to better characterize the data. Text analysis techniques are also adopted. The solution compares different classification models to get the best possible result.

I. Problem Overview

The project aims is to analyze a dataset that contains a collection of tweets in tabular format. The purpose is to predict the sentiment of a given content on Twitter. The sentiment provided in this task is either positive or negative. The dataset is split into two different CSV files that differ only for the size and the column related to the sentiment:

- a *development* set, containing 224,994 observations for which the sentiment of the user is provided
- an *evaluation* set, composed of 74,999 observations on which the sentiment has to be predicted

Considering the development set, we can find 278 **duplicated** data with opposite sentiment labels, so we decide to drop these rows because of ambiguity in the labeling; **no missing** data are present in the two different datasets reflecting the correct acquisition of data. Looking at the distribution of sentiment feature we have the 58% of data relative to negative sentiment, while the 42% for the positive sentiment; we conclude that the dataset has almost balanced classes.

Our dataset is based on five different features:

- **Date**: The publication date of the post collected from 2009/04/06 to 2009/06/25.
- **Flag**: The query used to collect the tweet. All posts have the same format so we drop this feature from the database.
- **Text**: This column contains the unstructured text of the different posts; characterized by different lengths, the shortest ones are posts of 6 words length, while the longest ones reache a value of 39 words.
- **Ids**: Include the numerical identifiers of the tweets. This is an incremental id that takes into account the trend of tweets over time.
- **Users**: List of users posting the tweet.

The label is represented by the *sentiment* column that can have two possible values, 0 for the negative sentiment and 1 for positive ones.

TABLE I
Feature Cardinality

Variable	Cardinality
ids	224716
date	189779
flag	1
user	10647
text	223106

In the Table I are reported the cardinalities of the dataset, while in Figure 1 and Figure 2 we report the study of user's name and day of the week according to each sentiment. In this case, we see a correlation between these features and the labels, so we decided to include them in our study.

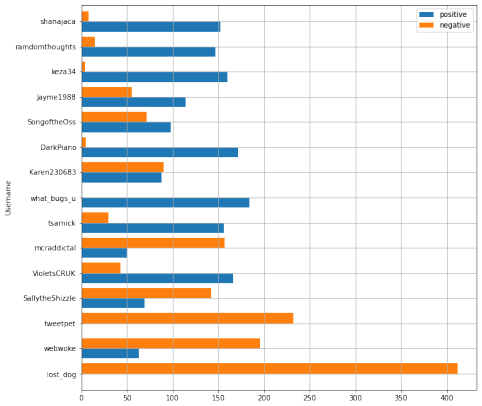

Figure 1. Comparison of users according to the sentiment of the tweets

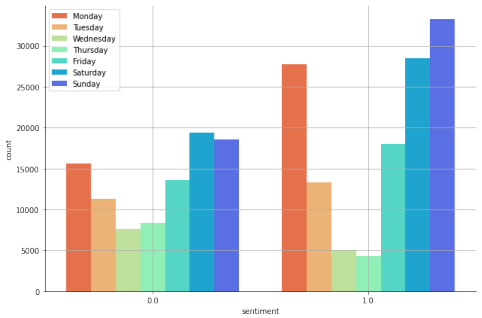

Figure 2. Comparison between days of week and tweet sentiment

II. Proposed Approach

A. Preprocessing

The two following preprocessing techniques have been implemented:

- *Text Cleaning*: we are dealing with **textual** data so we need to process the documents, in order to get the set of features that will be used in our classification model. For this reason, the phase of preprocessing and data cleaning is fundamental in this project.
  First of all, we transform our tweets in order to have a cleaned version of our data. We substitute the emoji according to their meanings (sad, neutral, or happy); we change all users' tags and hashtags by inserting the word 'tag' and 'hashtag' respectively; we cut off the URL, the numbers and the different punctuation. In the text are present special characters due probably to the collection of data from internet sources, like the string '"' or '&' that represent respectively ' " ' and 'and'. The following step consists in changing all slang words, text abbreviations and special characters with their sense. To reach this result we got a list of possible slang and chat abbreviations from a website [5], combining them with other abbreviations that we got looking to the records.
  Finally, we do the lemmatization of the text, which consists of grouping together the different inflected forms of a word so they can be analyzed as a single item. Lemmatization is similar to stemming but it brings context to the words. Figure 1 and Figure 2 are shown the words clouds of the positive and negative tweets. Word Cloud is a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance.

Figure 3. Word Cloud of positive tweets

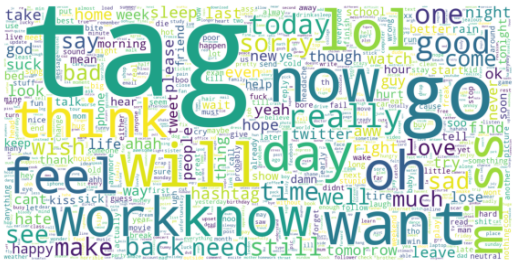

Figure 4. word cloud of negative tweets

- *Feature Extraction*: analyzing the name of users we find that are considered tweets of only 10,647 different people; in addition after the analysis shown in Figure 1. we add features representing: the day of the week, the number of posts for each user and the One Hot Encoding for each of them.
  Moreover, we decide to maintain date and ids features due to the fact both attributes are related to the temporal evolution of tweets, but as we can notice in Table I, some tweets are referred to the same date. So Ids allow us to distinguish these cases. Some of the starting features, the ids and the date (after ordinal encoding) have been normalized through a z-normalization technique, to make the convergence of the model faster.
- *Weighted document representation*: to extract relevant information from the textual description, we implement the tf-idf representation. Through the tf-idf vectorizer, we transform the documents into term vectors. Each term is a component of the vector and the value of each component is the number of times the corresponding term occurs in the document, multiplied for a specific weight. The weighting technique is the one relative to the term frequency-inverse document frequency (tf-idf), suitable for heterogeneous documents, where

$$tf\_idf(t) = freq(t,d) * log(m/freq(t,D)) \quad (1)$$

  With $freq(t,d)$ we represent the frequency of the term t in the specific document d, while the logarithm has the role of weight the frequency according to the number m of documents in which the term appears. This allows us to use the different words as separate features. Due to the big amount of tweets and different words we decided to let the td-idf matrix in a sparse format to concatenate the sparse matrix provided from the output of the One Hot Encoding of the users. Getting the complete matrix would be infeasible from a computational point of view and would saturate all the RAM available. Additionally, the meaning of adjectives and adverbs often depends on the context and it can have an impact on the sentiment of the tweet. To handle this variability not only single words were considered, but also ngrams (group of n words) of an adequate dimension.

B. Models Selection

After the preprocessing phase we test different classification models that are suitable for our binary sentiment problem:

- **Logistic Regression**: it belongs to the group of linear classifiers and is similar to polynomial and linear regression; is a binary method based on the usage of the sigmoid function and has the aim to find the logistic regression function $p(x)$ such that the predicted responses $p(x_i)$ are as close as possible to the actual response $y(i)$ for each observation $i = 1, ..., n$ that can only be 0 or 1 such that we are dealing with a binary classification

problem. The logistic regression function $p(x)$ is the sigmoid function of $f(x) = b_0 + b_1x_1 + ... + b_rx_r$: $sigm(f(x)) = 1/(1 + exp(-f(x)))$. As such, it's often close to either 0 or 1. The function $p(x)$ is often interpreted as the predicted probability that the output for a given x is equal to 1. During the training phase Logistic regression determines the best-predicted weights $b_0, b_1, ..., b_r$ such that the function $p(x)$ is as close as possible to all actual responses $y(i)$.

- **Random Forest**: this classification model uses ensemble learning techniques based on the combination of multiple base models. In order to improve the accuracy and to avoid overfitting, a set of decision trees is built on a random subsets and random set of features coming from our training data. The class is then assigned by majority voting.
- **SVM**: the Support Vector Machine has the aim to find a linear hyperplane that will separate the data maximizing the margin of data split. It can be a good solution in our project because we are dealing with a binary problem.

For all the classifiers, except for the SVM model, the best configuration of hyperparameters has been determined through a grid search, as shown in the following section.

C. Hyperparameters tuning

In the Table II we show the grid search specifically for the tf-idf vectorizer. The best parameters are the following: *max_features= 50,000*, *ngram_range= (1,3)*, *stop_words= none*. The stop_word are set to none otherwise the vectorizer would cut off some useful information between groups of words. For this reason, we got the best results taking groups of a maximum of three words long. We divide the development set into training and validation set with proportionality of 80/20. We use the first one to train model configurations, while the second one to compare the models' performances to pick the best between them.

TABLE II
HYPERPARAMETERS TABLE OF TF-IDF VECTORIZER

Parameters	Values
max_features	[5000,50000]
ngram_range	[(1,2),(1,3)]
stop_words	[none, english]

The grid search is performed with cross-validation using 10 splits on the two best models and is reported in the Table III; SVM is discarded because of its low performance.

TABLE III
HYPERPARAMETERS TABLE

Model	*Parameter*	*Values*
Logistic Regression	solver	newton-cg
	C	[1, 1.5, 1.8, 2]
	max_iter	[10, 100, 500]
	n_jobs	-1
Random Forest	n_estimators	[10, 50, 150]
	max_features	[sqrt, log2]

III. RESULTS

In this section are reported the main results of our analysis on the two considered models. In the Table IV are shown the scores of accuracy and the f1 score of our predictions according to the sentiment and the support. The best configurations of the models selected are:

- *Random_Forest*: max_features=log2, n_estimators=150, n_jobs=-1
- *Logistic_Regression*:, max_iter=100, solver=newton-cg, n_jobs=-1 , C=1.8

TABLE IV
SCORE TABLE OF OUR MODELS

Model	*Accuracy*	*Sentiment*	*f1*	*support*
Logistic Regression	0.851	0	0.821	23649
		1	0.873	32530
Random Forest	0.826	0	0.774	23649
		1	0.858	32530

As it is shown in the Table IV we get the best results with the Logistic Regression model, with an f1_macro = 0.847, given by the average of the two values. Due to this consideration, we decide to use this model for our classification task. Figure 5 is shown the confusion matrix where we can see that a good portion of data is labeled as True Positive and True Negative allowing us to reach a value of accuracy equal to 0.851.

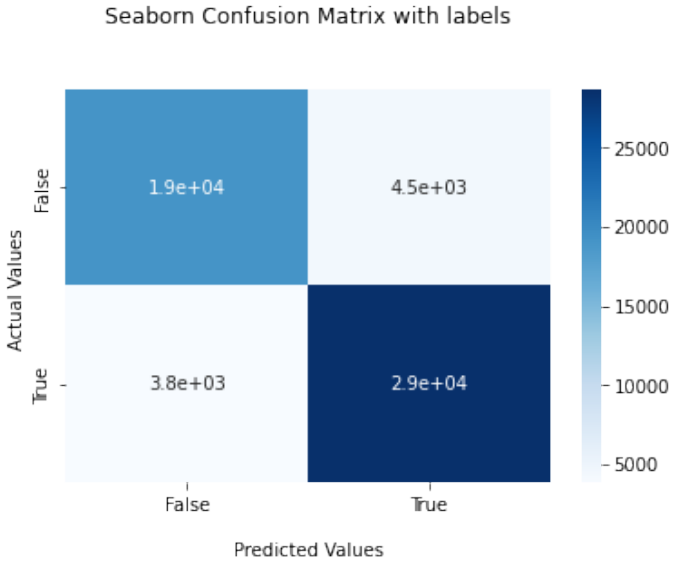

Figure 5. - Confusion Matrix of our model

To better understand the goodness of the prediction of the model we get the ROC curve and the specific AUC. A Receiver Operator Characteristic (ROC) curve is a graphical plot used to show the diagnostic ability of binary classifiers. It is constructed by plotting the true positive rate (TPR) against the false positive rate (FPR). The true positive rate

is the proportion of observations that are correctly predicted to be positive out of all positive observations. Similarly is done for the false positive rate. Classifiers that give curves closer to the top-left corner indicate better performance; to compare different classifiers, it can be useful to summarize the performance of each classifier into a single measure, the area under the ROC curve, which is abbreviated to AUC. When the area is closer to the value of 1 we have a good classification model, while the value of 0.5 is referred to a random guess model of the label. In this project, the AUC has a value of 0.846.

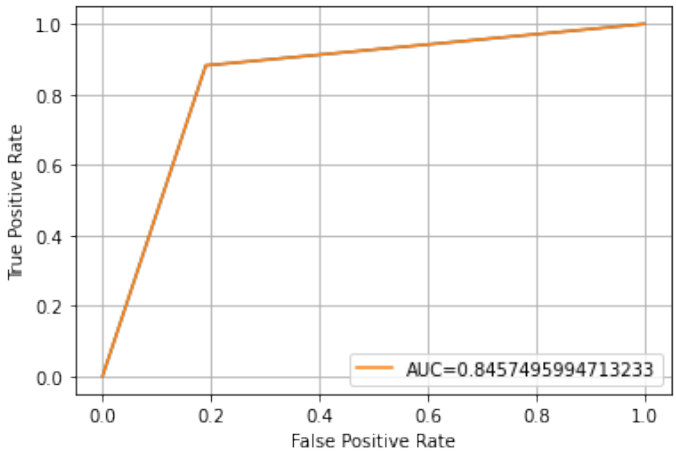

Figure 6. - ROC curve of our model

IV. Discussion

As we already mentioned, the proposed approach achives the best performances with Logistic Regression. We tried to balance our dataset according to the negative and positive sentiment, but this didn't improve our result so we discarded this option. Considering only the output of the tf-idf vectorizer and the One Hot Encoder of the users we got a score of 0.832 probably thanks to a good preprocessing on the text feature. Adding the remaining features we got a private score of 0.847, but with a public score of 0.849. That means that our solution isn't affected by the overfitting of the data. Also, the difference between this solution and the head of the leader board is around 0.03 (excluding the top performer with 0.998). Various aspects might be taken into account to improve the obtained results:

- A Better definition of the threshold used before One Hot Encoding or the adoption of different techniques of encoding for categorical features (i.e. *frequency* encoding)
- A Better text preprocessing with improved handling of emoticon detection, that might give a more precise prediction
- Use a more advanced text mining technique (i.e. word embedding) that allows us to extract correlation between meaningful words.
- Run a deeper grid search both on preprocessing and classification hyperparameters. In this project, very few combinations of parameters have been tested for each model (due to a too long computational training time) Furthermore additional classification models might be considered.
- Balance the data with a better solution than oversampling or undersampling, because a not perfectly balanced data distribution affects the model training. As we can see in the Table IV there is an important difference between scores from label to label due to the different support.

It could be meaningful to reduce the number of features, which is important for model interpretability. The achieved results are already promising and they will only benefit from the proposed improvements.

References

[1] https://www.kdnuggets.com/2017/03/beginners-guide-tweet-analytics-pandas.html
[2] https://www.analyticsvidhya.com/blog/2018/02/the-different-methods-deal-text-data-predictive-python/
[3] https://www.pluralsight.com/guides/building-a-twitter-sentiment-analysis-in-python
[4] https://github.com/dbdmg/data-science-lab#readme
[5] https://www.webopedia.com/reference/text-abbreviations/
[6] https://www.kaggle.com/stoicstatic/twitter-sentiment-analysis-using-word2vec-bilstm