Twitter Sentiment Analysis

Davide Di Mauro
Politecnico di Torino
Student ID: s258592
s258592@studenti.polito.it

Giovanni Sciortino
Politecnico di Torino
Student ID: s302959
s302959@studenti.polito.it

***Abstract*—In this report we propose a possible approach to sentiment analysis on Twitter data. In particular, the effectiveness of different preprocessing steps are analyzed and using tf-idf we train different classification models. Moreover, summary statistics are computed and visualized.**

I. Problem overview

The aim of the competition is to predict the sentiment of tweet through a classification pipeline provided two different dataset consisting in:

- a **development** set of 224,994 tweets
- an **evaluation** set of 74,999 tweets

The datasets consist in a collection of tweets in tabular format. Each record is characterized by several attributes. The following is a short description for each of them.

- *ids*: a numerical identifier of the tweet;
- *date*: the publication date;
- *flag*: the query used to collect the tweet;
- *user*: the username of the original poster;
- *text*: the text of the tweet.

Furthermore in the development set is present an additional attribute, *sentiment*, i.e. the class to predict that can be either 0 (negative) or 1 (positive).

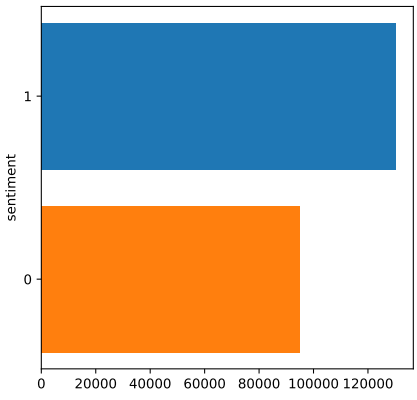

Fig. 1: Positive and Negative tweets distribution

Making an overview on given data, we notice that missing value in both datasets are not detected. Secondly, analysing the target value, the training set is slightly-unbalance: there are more positive tweets than negatives one as in Fig. 1. Considering tweets' length, as shown in Fig. 2, both positive and negative tweets have in average almost the same number of characters.

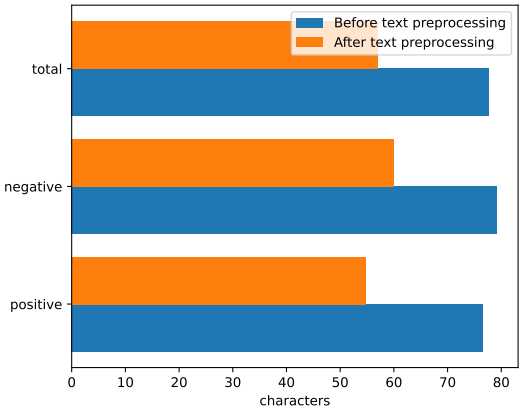

Fig. 2: Average tweets' length

Moreover, analysing the tweet's distribution during a day, we can notice that the positive ones are always greater. Overall, the bar chart of Fig. 3 shows that the tweet's amount decrease slowly during the day but having a clear upward in number from the end of working hours to midnight.

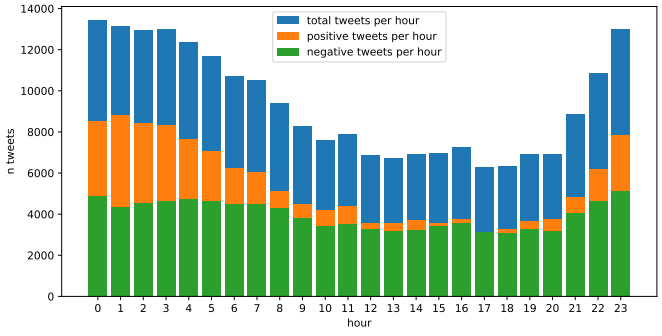

Fig. 3: Daily tweets distribution

Since the proposed classification problem is a sentiment analysis one, the text attribute plays a quite relevant part for

reaching a good solution. So one of the key part of the solution is the use of tf-idf, which is a very powerful text mining technique. In particular, it's a statistical measure that evaluates how relevant a word is to a document in a collection of documents using two metric: how many times a specific word appears in the document and the inverse document frequency of the word across the set of documents.

II. Proposed approach

A. Data preprocessing

In order to extract relevant information from the data provided, now we focus on the data preprocessing step in which we apply different data mining techniques.

So we consider each class and make some consideration about each of them:

- ***Id*** and ***Flag***: Due to the nature of these attributes, we decide to don't elaborate them.
- ***Date***: All the tweets collected, in both datasets, are temporary located in 3 months of 2009. For this reason we convert this class into 4 different categories: month, day of the week, hour and minutes without considering the year.
- ***User***: Usernames are textual data that are not strictly useful but a user can post multiple tweets. For this reason we create a new numerical feature that represents the number of tweets each user made.
- ***Text***: As previously said one of the most important step for sentiment analysis is to process the textual data using sklearn's TfidfVectorizer; but before that we decide to make some preprocessing using other techniques. First, multiple suited regular expressions have been used to detect and erase the following elements: url, tags, links, extra space, punctuation and repeating character; the last one is done to make the text more uniform and correct as much as possible taping errors. Then the negations has been expanded, e.g. "isn't" become "is not".
  Once this done, the text is tokenized. Then, using NLTK's library, we perform stemming, i.e. reduce the words to their derived stems.
  We also created manually a positive and a negative lists of emoticons; which allow us to detect them and add two numerical attributes that counts the number of positive and negative emoticon in each tweet.
  In order to check the effect of the preprocessing and see how the text is changed after the previews techniques have been applied we created a numerical feature that keeps information about the length of the text. We also noticed that at the end of this step a few entries resulted in a blank text.
  Moreover, Fig. 2 shows how, after text preprocessing, average length decreases constantly for both positive and negative tweets.
  A necessary consideration about English stopwords need to be made. Stopwords are words which do not add much meaning to a sentence, so usually it's better to remove them. However, running some test keeping them in this step, we noticed a better result; so in this solution we decide to keep the stopwords.
  Finally two wordclouds gives us a graphical idea of the words' distribution after this step (Fig. 4-5)

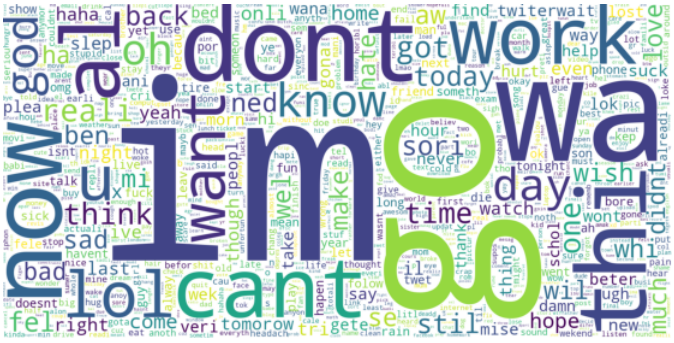

Fig. 4: Negative wordcloud

Fig. 5: Positive wordcloud

B. Model selection

Due to the different classification models' constrains we approached the problem in two ways.

TfidfVectorizer's output, a sparse matrix, need to be converted to a dense matrix in order to be insert in pandas' DataFrame. Since a dense matrix needs more memory to be stored, we had to use a limited number of max-features in the TfidfVectorizer (we tried with 300 features).

So using all the preprocessed classes and the new features given by the TfidfVectorizer we tested the following classification models:

- *Random forest*: this algorithm belongs to the category of ensemble techniques because it combines multiple decision trees (trained on different portions of the training set and subsets of features) to make predictions. This is generally done to enhance the performance and to avoid overfitting.
- *SVC*: this algorithm applies a transformation to the data which is in general non linear using a kernel function. It tries to find the hyperplane which maximizes the margin of errors between predicted and actual values that are tolerated.

Model	Parameters	Values
TfidfVectorizer	*max_features*	{100, 500, 1000, 5000, 10000, 50000, 100000, 500000}
	ngram_range	{(1,1), (1,2), (1,3)}
LogisticRegressor	*C*	1, 2
	solver	lbfgs, liblinear
	max_iter	100, 500, 1000, 5000
LinearSVC	*C*	1, 2
	max_iter	100, 500, 1000, 5000
KNeighborsClassifier	*n_neighbors*	5, 10
	weights	uniform, distance

TABLE I: Hyperparameters considered

- *MLPC*: it's a classification model based on a class of feed forward artificial neural network

The only model that provided good result was the Random Forest (*f1 score* $\approx$ 0.760). For this reason we decided use a different approach based on using only the TfidfVectorizer's output. So we tested only classification models that can receive a sparse matrix as input, which are:

- *Logistic Regression*: an algorithm that measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative distribution function of logistic distribution.
- *LinearSVC*: a method that applies a linear kernel function; that, according to the sklearn documentation, is flexible in the choice of penalties and loss and should scale better to large numbers of samples
- *K-Nearest Neighbors*: in approach to classification problems, is a method that estimates how likely a data point is to be a member of one group or the other depending on what group the data points nearest to it are in.

These last models gave us better result than the previous ones; so from this point onward the argument of this report will be developed using the last approach.

C. Hyperparameters tuning

There are two main groups of hyperparameters took in consideration, each corresponding to a different phase:

- *preprocessing*: max_features and ngram_range.
- *final predictions*: classifier models hyperparameters.

To find the best parameters we applied the cross validation method with 5 folds using GridSearchCV. Doing so we considered the combinations of parameters summarized in Table I.

III. Results

As expected, the results are improving as the number of tdidf's max_features increases. Fig. 6 and Fig. 7 show how *f1 score* grows and it stabilizes once 10000 features are reached.

Another TfidfVectorizer's parameter that affects the results is the ngram_range(a n-gram is a string of n words in a row). In our case we select the range (1,3) that gives us the best result as it's possible to see in Fig. 8. Both classifiers, Logistic Regressor and LinearSVC, achieve satisfactory performance while KNN gives us a result even worse than Random forest.

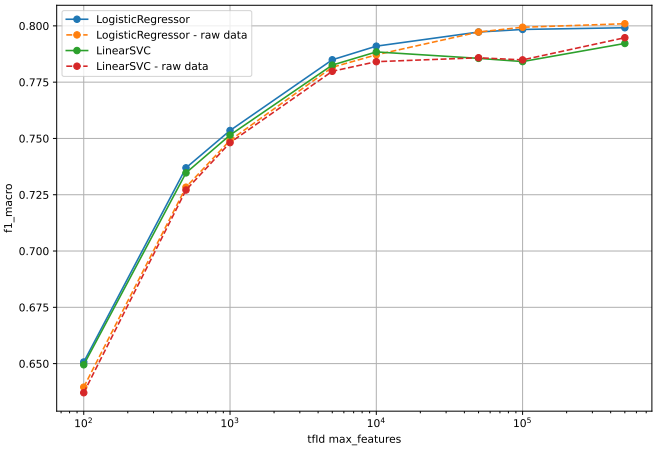

Fig. 6: f1 score with or without data preprocessing

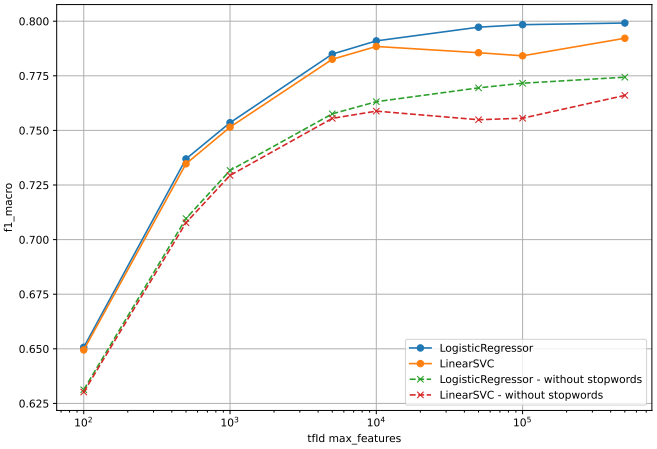

Fig. 7: *f1 score* with or without stopwords

{n_neighbors=10, weights=uniform}(*f1 score* $\approx$ 0.537) The best configuration for the Logistic Regressor is found for {C=2, solver='liblinear', max_iter=500}(*f1 score* $\approx$ 0.801), whereas the best configuration for LinearSVC is found for {C=1, max_iter=100}(*f1 score* $\approx$ 0.796). We train the best performing Logistic regressor and LinearSVC on all available development data. Then the models have been used to label the evaluation set. The public score obtained is of 0.804 for the logistic regressor and of 0.801 for the LinearSVC.

IV. Discussion

In the proposed approach we can notice that training the models using preprocessed or not preprocessed data gives us basically the same results. But since TfidfVectorizer applies some preprocessing on text, we can assume that it's roughly

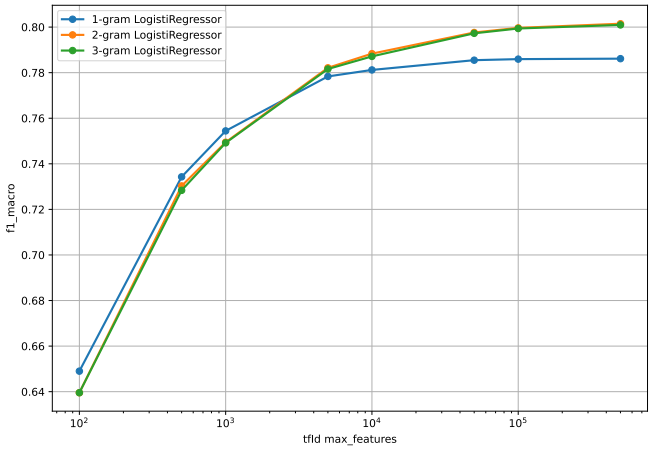

Fig. 8: *f1 score* with different n-gram

the same as the one proposed in Part II. In fact, it's possible to check the performance of both classifier in Fig. 6.

An other interesting insights that the results shows is that removing English stopwords gives us a significant decrease of the f1 score.

As previously said, the best results reached was obtained only by using tweets' text due to storage limitation. One aspect to consider that might improve the actual result is to find a way to better perform Random Forest; in particular, adding the sparse matrix to the DataFrame without converting it in a dense form. In fact, using this algorithm, we would be able to select other features provided instead of using only the text.

REFERENCES

[1] H. Garg "A Complete Guide to Twitter Sentiment Analysis-Part I'. [Online]. Link: https://medium.com/analytics-vidhya/a-complete-guide-to-twitter-sentiment-analysis-part-one-8dd6d82118b3.

[2] H. Garg "A Complete Guide to Twitter Sentiment Analysis-Part II". [Online]. Link: https://medium.com/analytics-vidhya/a-complete-guide-to-twitter-sentiment-analysis-part-two-7349550bdea9.

[3] D. Subramanian "Text Preprocessing for Data Scientists". [Online]. Link: https://towardsdatascience.com/text-preprocessing-for-data-scientist-3d2419c8199d.

[4] H. Jabeen "Stemming and Lemmatization in Python". [Online]. Link:https://www.datacamp.com/community/tutorials/stemming-lemmatization-python.