Tweets Sentiment Analysis

Beatrice Macchia, Riccardo Bosio
Politecnico di Torino
Student ids: s292178, s291299
s292178@studenti.polito.it, s291299@studenti.polito.it

***Abstract*—In this report we introduce a possible solution to the Tweets Sentiment classification problem. The proposed approach consists in the extraction of keywords from the text, through vectorization and stemming, and the encoding of the user name of the tweet publisher and of the publication hour. The solution obtains overall satisfactory results.**

I. Problem overview

The objective of this competition is to perform Sentiment Analysis of some tweets, which means to determine whether they represent a positive or negative attitude.

More in detail, the training dataset contains 224994 tweets. Each of them is described by:

- *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;
- *sentiment* : The target variable, it is 0 when the tweet's sentiment is Negative, 1 when it is Positive.

We compute our final results on an evaluation set composed of 74999 tweets.

First we analyse the training set and we compute the number of unique values of each variable, as reported in Table I.

TABLE I
Unique values for each feature.

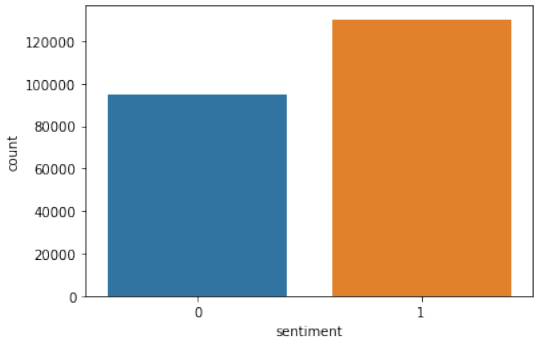
Feature	Cardinality
ids	224716
date	189779
flag	1
user	10647
text	223106

We observe that the number of unique *ids* is lower than the total number of records. It means that there are some duplicate values: we do not know if they have a specific aim, such as increasing the importance of some entries or they are the result of a mistake concerning the aggregation of similar datasets. We decide to drop the duplicates. The *date* is given in a very specific form, with the publication time in terms of hours, minutes, seconds and time zone. The variable *flag* is set to *'NO_FLAG'* for all the tweets, so we will not take it into consideration. The number of unique *user*'s values is quite small with respect to the number of total tweets. On average we have more than 20 tweets for each user, it means that it could be a useful parameter since some users could be more inclined to use certain words to express a specific sentiment. The *text* seems to present some duplicate values: some of them are those that have the same *ids*, while the others are for example same tags, very common expressions (e.g. 'i have a headache') or copied tweets.

Moreover we observe that there are no missing values in any feature.

For what concerns the distribution of the target variable $sentiment$ in the training set, it presents a little imbalance as can be seen in Figure 1: 130157 positive sentiments against 94837 negative sentiments.

Fig. 1. Sentiment distribution.



Here you can find the dataset and the code.

II. Proposed approach

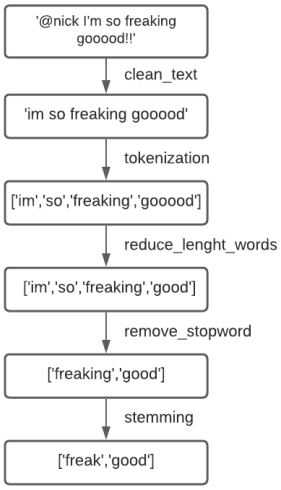
A. Preprocessing

We drop the *flag* feature and focus our attention on *text*, *date* and *user*:

- **text**: the text of tweets is the core of our analysis and it needs to be cleaned. First of all we delete the tags and re-tweets (e.g. '@nick' or 'RT@nick'), then we make the text lowercase and remove punctuation, links and words containing numbers. We tokenize the text in order to split it into the single words, then we reduce the length of words according to the following idea: if a word has more than two identical letters next to each other, they are replaced by only two of them. For example: "im so haaaappy" becomes "im so haappy". Then we remove the stopwords. We tried the prebuilt stopwords library of *nltk* [1] module, but in our opinion also some relevant words were removed (e.g. "not"). This is why we create our own list of stopwords and remove them from text. Finally we apply Stemming to the words, reducing derived words to

their stem. In the following plot the preprocessing of a tweet example is reported in Figure 2.

Fig. 2. Preprocessing on 'text' example.



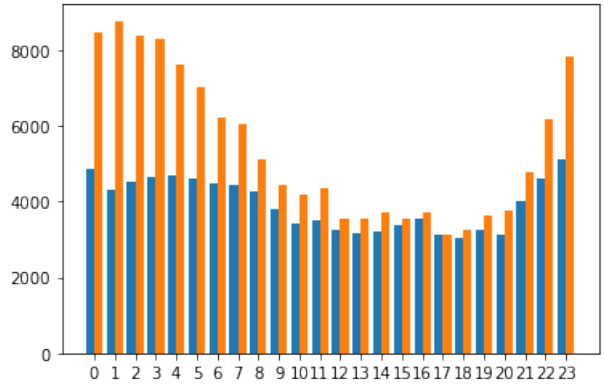
We also observe that some texts are reduced to an empty line and we remove these entries from the training dataset. Now our strategy is to use only a certain number k of most frequent words in tweets. Once the list of k frequent words is selected, we delete each word in text that does not belong to that list.

The text is now given as input to the *Tf-Idf* [2] algorithm that is able to reflect how important a word is with respect to the set of all the tweets. It is the product of two statistics, the 'term frequency', that indicates how many times a given word occurs in a given tweet, and the 'inverse document frequency', that measures how common the word is in the whole collection of tweets. We use the *sklearn*'s *TfIdfVectorizer* passing as *vocabulary* the k common words we have selected before. We try with $k = \{1000, 5000, 10000, 20000\}$ and reach the best result with $k = 10000$.

- **user**: in order to use the $user$ feature, we need to apply the One-Hot Encoding, since it is not a numerical attribute. This technique generates a new binary feature column for every possible user.
- **date**: regarding the *date*, only 3 months are present in the dataset (May, April and June), so we do not take the month into consideration. Instead, we observe that the distribution of positive/negative sentiments over hours has the behaviour represented in Figure 3. What is interesting is that in some given moments of the day, mostly during night, we have an higher distance between the number of positive tweets (orange) and the number of negative tweets (blue).

  Therefore we create a new feature *hour* where we put the hour in which the tweet was published. Then we apply the One-Hot Encoding on this new feature because, with the *hour* values in the original form $h \in \{0, 1, 2...23\}$, the model would associate misleading distances to them: for example the hours 0 and 23 would be considered the farthest from each other, while they are only 1 hour far.

Fig. 3. Sentiment distribution over hours (blue:negative, orange: positive).



Now that we have worked on the single features, we need to merge the outputs of the different processes. The result of Tf-Idf (related to *text*) is a sparse matrix, while the One-Hot Encoding ones (*user* and *hour*) are *pandas* DataFrame. In order to reduce memory usage we decide to convert the last two into sparse matrices. Finally we join them and obtain the final dataset to train our models.

B. Model selection

We test the following algorithms:

1) **LinearSVC**: the Linear Support Vector Classifier method applies a linear kernel function to perform classification. It is similar to SVC with $kernel = linear$, but it has more flexibility in the choice of penalties and loss functions and it should scale better to large numbers of samples.
2) **Logistic Regression**: it is a supervised learning classification algorithm used to predict the probability of a target variable. Input values are combined linearly using weights or coefficient values and the result is passed to the Logistic function (sigmoid) that produces a probability in order to assign a class. The coefficients are estimated using training data and maximum-likelihood estimation. This model is often used for binary classification.

C. Hyperparameters tuning

The hyperparameters to be tuned are not only those associated to the models, but also the number of most frequent words k. Concerning the models, we do a grid search for both the models on the hyperparameters in table II, using a Cross Validation splitting strategy with 5 folds. The performances are evaluated using $F1\ score\ Macro$. Concerning the number of most common words taken into account, we try with $k = 1000, 5000, 10000, 20000$ and we reach the best results with $k = 10000$. In figure 4 and 5 we observe the most frequent words respectively in positive and negative tweets.

TABLE II
GRID SEARCH.

Model	Hyperparameter	Values
LinearSVC	*C*	{0.01, 0.1, 1, 10, 100, 1000}
	penalty	{"l1", "l2"}
	loss	{"hinge", "squared_hinge"}
	dual	{False, True}
	tol	{0.01, 0.001, 0.0001, 0.00001}
	max_iter	{100, 200,..., 1200}
	fit_intercept	{False, True}
Logistic Regression	*C*	{0.01, 0.1, 1, 10, 100, 1000}
	penalty	{"l1", "l2"}
	max_iter	{100, 200,..., 1200}
	solver	{"newton-cg", "lbfgs", "liblinear", "sag", "saga"}

Fig. 4. Most frequent words in positive tweets.

Fig. 5. Most frequent words in negative tweets.

III. RESULTS

The best configuration for LinearSVC is {*'C': 0.1, 'dual': False, 'fit_intercept': True, 'loss': 'squared_hinge', 'max_iter': 100, 'penalty': 'l2', 'tol': 0.01*}. We split the training data with the Hold Out method generating a traning set composed of 80% of the original one and a validation set with the remaining 20%. The *F1 score Macro* obtained on the validation set is equal 0.815. Consequently, we train this model on the entire training dataset and we predict sentiments of the evaluation dataset. LinearSVC achieves an *F1 score Macro* of 0.816.

Instead, the best configuration for the Logistic Regression is {*'C':10, 'max_iter':300, 'penalty':'l2', 'solver':'newton-cg'*}. It obtains an *F1 score Macro* of 0.815 on the validation set and 0.816 on the public leaderboard.

IV. DISCUSSION

The models achieve the same results in terms of *F1 score Macro*. We can say that there is no overfitting since the results achieved on the public leaderboard are equal to the ones obtained on the validation set. The solution found performs well above the baseline. The results obtained seem promising and some experiments can still be done on the following aspects:

- try Neural Networks, especially RNNs can perform well in processing the attribute *text*;
- run a wider grid search;
- test different configurations in *text* preprocessing and find other methods to extract the keywords.

REFERENCES

[1] E. S.Bird, "Nltk: The natural language toolkit," *21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, 2006.

[2] H. C.D.Manning, P.Raghavan, *Introduction to Information Retrieval*. Cambridge University Press, 2008.