Twitter Sentiment Classification Problem

Pietro Montresori, Giuseppe Suriano
Politecnico di Torino
Student id: s296440, s296605
s296440@studenti.polito.it, s296605@studenti.polito.it

***Abstract*—In this report we introduce a possible approach to a classification problem based on Twitter sentiment analysis. The proposed approach consists on applying to a classification model customized preprocessing functions to deal with the tweet lexicon, a vectorizing and a feature selection phase. The pipeline proposed have been tested with the *f1_score*, obtaining satisfactory results.**

I. Problem overview

The competition consists in a classification problem based on a dataset made by Twitter content. The goal is to assign to a given tweet the corresponding sentiment, 0 if it is negative and 1 if it is positive. The dataset is divided in two parts:

- a *development set*, containing 224994 tweets with the related sentiment.
- an *evaluation set*, containing 75000 tweets without the related sentiment.

We will use the development set to build a classification model to accurately assign the sentiment to the tweets of the evaluation set. The dataset attributes can be seen in Table I.

TABLE I
DATASET ATTRIBUTES

Name	Type	Description	Cardinality
Ids	Nominal	Numerical identifier of the tweet	224716
Date	Ordinal	Publication date	189779
Flag	Nominal	Query used to collect the tweet	1
User	Nominal	Username of the original poster	10647
Text	Text	Text of the tweet	223106
Sentiment	Nominal	Sentiment of the tweet	2

Due to the nature of Twitter, the dataset text is written in a colloquial way, and for this reason, as we can see in Figure 1, the collected data is filled with many informal expressions. Secondly, the length of the text should be 140 characters, but because of the noise that we will address later, 2700 lines of text are longer than that.

Finally, we can state that there are no missing values for each one of the features.

II. Proposed approach

A. Preprocessing

1) *Text Preprocessing* In this part we have focused on how to threat duplicates and informal expressions. From Table I we can observe that the cardinality of the feature *text* is lower than the one of the dataset, that's because of the presence of duplicates. After dropping duplicates we reduce the set by 2757 rows.
Concerning the informal words instead, whose frequency we can see in Figure 1, we have decided to operate with the aim of reducing the number of columns that the dataset will have after the vectorization phase. We have tested all the preprocessing functions [1] that can be seen in Table II, and we have decided not to apply *replaceContraction*, *addNotTag* and *removeNumbers*, because they was neither improving the performance nor reducing the number of columns.
2) *Additional Informations:* With a quick exploration we can say that not all the features contained in the dataset provides meaningful information: column *Ids*, for example, is just a numerical identifier, and *Flag* contains always the same information. Instead, we can extract useful information, other than from the *Text*, from other features:
   - the *Date* column, that contains dates from a trimester of 2009. We can see that there is no seasonality to measure, so the month doesn't provides useful information. Nevertheless, the distribution of positive and negative sentiment changes for different days of the week and for different hours, so we decided to keep those two information.
   - the *User* column, that we have used to collect the mean and the standard deviation of all the tweets for each user. We have used these statistics also in the evaluation set, setting up 1/2 as default value for both the mean and the standard deviation in case of unmatching users.

   All this methods have allowed us to halve the cardinality of the vectorizator's dictionary.
3) *Vectorizing process:* Glancing at the literature, we have found empirical evidence in support of the use of bigrams features [3]. In fact, because they can capture modified verbs and nouns, using them should improve the performance. Our feature vector will use a unigram+bigram approach, and a text like "I have a lovely daughter" will become: "i", "have", "a", "lovely", "daughter", "i have", "have a", "a lovely", "lovely daughter", each part with the relative column.
Then, we have decided which vectorization to use, by implementing in parallel *TfidfVectorizer* and *bag of*

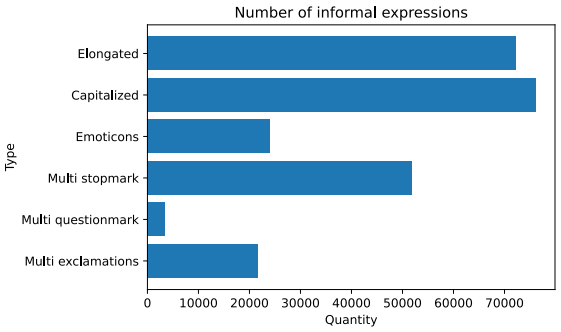

Fig. 1. Number of informal expressions for each type

TABLE II
PREPROCESSING FUNCTIONS

Name:	Usage:
replaceURL	Replaces URL address with the tag ”url”
removeUnicode	Removes all of unicodes strings
replaceAtUser	Replaces all the mentions ”@user” with the tag ”atUser”
removeHashtagsInFrontOfWord	Removes hashtags in front of a word
addNotTag	Finds ”not”,”never”,”no” and adds the tag NEG_ to all words that follow them
replaceContraction	Replaces contractions with their equivalent word
replaceMultiExclamationMark	Replaces repetitions of exclamation marks with ”multiExclamation”
replaceMultiQuestionMark	Replaces repetitions of question marks with ”multiQuestion”
replaceMultiStopMark	Replaces repetitions of stop marks with ”multistop”
replaceNegation	Replaces negation with their antonym
removeNumbers	Removes all integers in the text
replaceElongated	Replaces an elongated word with its basic form
removeCodes	Removes the HTML entity names ”"” and ”&”
emot_obj.emoticons	Extract the emoticons in the text and replace them with their meaning [2]

words representation using *CountVectorizer*. We have not included in the vectorization the *stopwords* removal, mainly because the *stopwords* list contains also positive and negative words that could provide useful information for the classification process.

B. Model selection

Here we have opted for a pipeline that includes a feature selection phase and a classification phase [4]. Concerning to the feature selection, we have decided to use *SelectFromModel*, a Scikit-Learn based transformer that, given a classifier that weight the importance of the columns [1], reduce the number of columns by removing the less useful ones. Regarding instead the model selection, we have tested the following algorithms, focusing on performance and timing of model building:

- *ridgeClassifier:* : it is useful because, associating coefficient to the features, provides useful informations about its behaviour (as we will see in the the wordcloud in the DISCUSSION phase).
- *Stochastic Gradient Descent (SGDClassifier):* It is particularly suited for large datasets, because it has a fast convergence due to the frequent update of the parameters. Also, like *ridgeClassifier*, it provides coefficients that are useful for an interpretability analysis [5].

C. Hyperparameters tuning

In this phase we have considered the two previous models, whose hyperparameters are represented in Table III. As will be seen in the RESULTS part, our hyperparameters will be tuned with a Grid Search to obtain the optimal configuration for the model. This Grid Search, as all the ones that are done in this project, are based on the sklearn *GridSearchCV*, trained on the 75% of the *developement.csv* with a 5-fold cross validation and tested on the remaining 25%.

TABLE III
CONSIDERED HYPERPARAMETERS

Model:	Hyperparameters:	Values:
SDG	loss	[”hinge”]
	penalty	[”l2”, ”elasticnet”]
	tol	[1e-3, 1e-4]
	learning_rate	[”adaptive”, ”constant”]
	eta0	[0.1, 0.5, 1]
Ridge	solver	[”auto”, ”saga”, ”sparse_cg”]
	fit_intercept	[True, False]
	tol	[1e-3, 1e-4]
	alpha	[0.1, 0.5, 1]

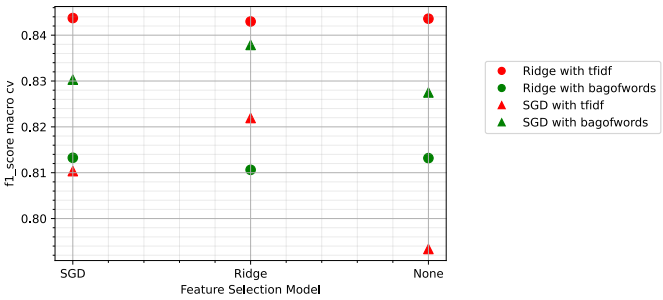

Fig. 2. Representation of the Grid Search’s results

III. RESULTS

At this point, in order to achieve the best possible result, we have tested with a Grid Search all the combinations of the three main parts of the pipeline:

1) The vectorization methods: *TfIdf* and *Bag Of Words*
2) The feature selection method: *SelectFromModel* with Ridge Classifier and *SelectFromModel* with SGDClassifier. We have also tested a scenario where there is no feature selection.

[1]We have tested as classifier for *selectFromModel* both our models in the RESULTS phase

3) The classification model: *RidgeClassifier* and *SGDClassifier*, with their default configuration.

The different scores obtained by the 12 possible combination can be observed in Figure 2.

We can see that there are two main successful results:

- *SGDClassifier* performed over a *BagOfWords* vectorization with *SelectFromModel*, that use *RidgeClassifier* as a classifier.
- *RidgeClassifier* performed over a *TfIdf* vectorization with *SelectFromModel*, that use *SGDClassifier* as a classifier.

Subsequently, in order to find the definitive combination, we have done another Grid Search, comparing the previous two solutions with all their possible combinations of hyperparameters (described in Table III).

TABLE IV
BEST CONFIGURATION WITH THE RELATED SCORE

Vectorization:	Tf-Idf
Feature Selection:	SelectFromModel with SDGClassifier
Model:	RidgeClassifier
Hyperparameters:	alpha= 0.1 fit_intercept= False solver= "saga" tol= 0.001
f_1 score (macro):	0.8455

As can be seen, the f1_score macro of the winning configuration in Table IV is slightly more than the public score, which is 0.837. This is probably caused by the different distribution of data between the *developement.csv* and the public dataset. From the public scoreboard we can see also that our solution outperforms the baseline, which is set at 0.753.

It is quite interesting that even a simple *Multinomial Naive Bayes* classifier is able to achieve an f1_score of 0.785, if evaluated with the preprocessed and vectorized dataset[2]. This probably means that alle the preprocessing step that we have applied are very effective.

IV. DISCUSSION

With the proposed pipeline we have achieved considerable results, even if there is still potential for further improvement. Some additional experimentation can still be done on some aspects:

- Perform undersampling on the dataset to balance it (right now the positive sentiments are 60% of the total).
- Detection of the outliers in the dataset.
- Usage of SVM and other computational intensive algorithm to improve the performance.
- Perform an all-inclusive Grid Search that includes the combinations of vectorization, feature selection, classification models and also their hyperparameters. With this process probably we could have achieved higher performances, but it would have necessary required a higher computational time.
- Try to add to the unigram+bigrams features also the POS tagging ones.

In the end, another aspect that we decided to analyze is model interpretability, in order to be able to understand more which are the factors that contribute, both positively and negatively, to the classification of a tweet. To do that, we have done an interpretability analysis on the selected pipeline (available on Table IV). Because *RidgeClassifier* assigns coefficients to each one of the columns to perform classification, we can extract the words with higher positive or negative weight and represent them in a world cloud, available in Figure 3. As we can see in the picture, many of the words that have double letters are misspelled. This could have been caused by the preprocessing stage, where informal expressions are solved algorithmically instead of with the support of external dictionaries. However, our solution is not affected by spelling errors but considers the meaning of the words, therefore we can notice that the majority of the plotted words have a clear positive or negative meaning.

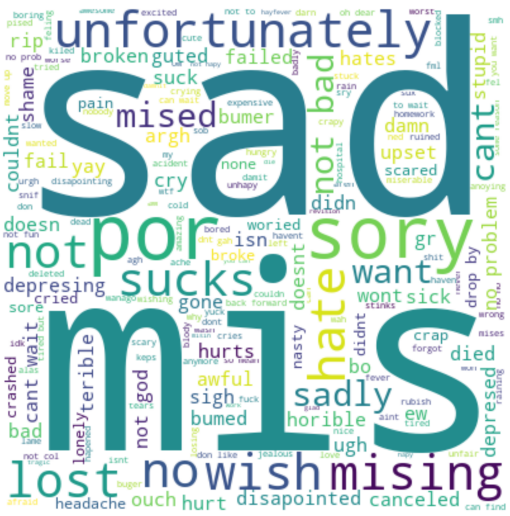

Fig. 3. Wordcloud representation of the most important words according to RidgeClassifier

[2]We have performed *bag of words* for the vectorization phase

REFERENCES

[1] D. Effrosynidis, S. Symeonidis, and A. Arampatzis, "A comparison of pre-processing techniques for twitter sentiment analysis," in *International Conference on Theory and Practice of Digital Libraries*, pp. 394–406, Springer, 2017.

[2] N. Shah, "emot." https://github.com/charlespwd/project-title.

[3] S. I. Wang and C. D. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 90–94, 2012.

[4] G. Prettenhofer and B. Blondel, "Classification of text documents using sparse features," scikit-learn.org.

[5] L. Bottou, "Stochastic gradient descent tricks," in *Neural networks: Tricks of the trade*, pp. 421–436, Springer, 2012.