Sentiment Analysis Classification Problem

Niccolò Revel Garrone , Lorenzo Melchionna
Politecnico di Torino
Student id1: s302302 Student id2: s304651
s302302@studenti.polito.it , s304651@studenti.polito.it

***Abstract*—In this report we analyze a possible approach to the Sentiment Analysis classification problem. In particular, in the first phase we analyzed the distribution and conformation of the dataset, then we did the preprocessing in which we removed all the parts that disturbed the classification and modified the emoticons and the relevant punctuation with the respective recognition words. Subsequently, through the tokenization and the Tf-idf we extracted the features matrix and we concatenated them two different sparse matrix: one for users column and one for the days of week. We used the final sparse matrix for the construction and for the choice of the model and hyperparameters with the best performance.**

I. Problem overview

The proposed project is a classification problem on twitter sentiment analysis, a collection of comments posted on twitter by different users and by different dates. The goal of this project is to predict the sentiment of a specific comment on twitter and classify positive comments with a value of 1 and negative comments with a value of 0.
The difficulty of this problem is that people express similar feelings using different words. Especially on social network this phenomenon is quite evident; in fact, for example, words, emoticons or emojis can be used to express happiness or sadness.
The dataset is divide into two parts:

- a development set, containing records for which a label
- a evaluation test, containing records without label

Both dataset contain 5 columns and we did not use from the datasets the Ids column, which was a metadata that identify each tweet, and the flag column because all tweets have same value. We will need to use the development set to build a classification model to correctly label the records in the evaluation set.
We can now make some considerations based on the development set. First, the dataset is too large and it is not possible to analyze it all together because memory and processing speed problems are created. For doing the analyzes and the considerations regarding the problem, we decided to use, after the first part of preprocessing, a sparse matrix instead of a dense matrix. The difference is that sparse matrix contains only non-zero values and it occupies less memory that the dense matrix. Second, we found that in the dataset there are duplicate comments, so we decided to remove them to give the same weight to all comments. Third, Figure 1 [1] shows that the problem is not well-balanced,, however we have decided not to manually balance it before the preprocessing but to solve this problem during the construction of the model. We have also seen that the length of positive comments is usually shorter than that of negative comments. The boxplot in the Figure 2 shows the distribution of the respective lengths.
Last, we have observed that in the dataset there are duplicate comments that have different labels, so we decide to remove both comments in order to not compromise the training operations.

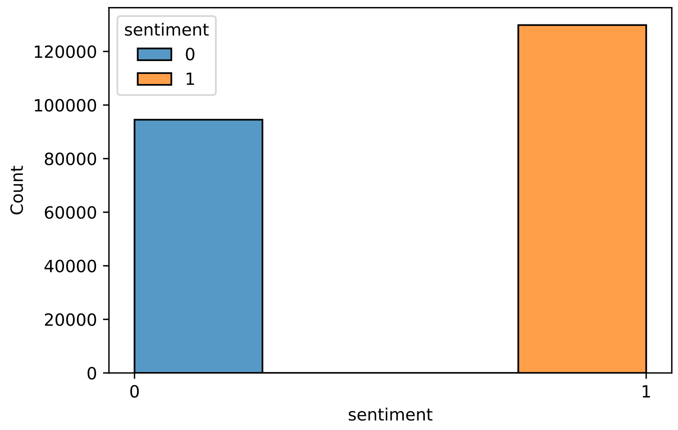

Fig. 1: unbalanced problem

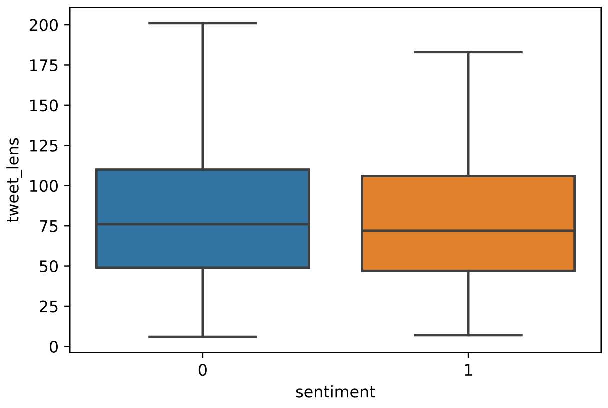

Fig. 2: comment leghts boxplot

II. Proposed approach

A. Preprocessing

The preprocessing part is fundamental in order to analyze and optimize the classification problem, in this project we have decided to proceed by stages.
We separated the development dataset in two parts: features and target, in order to extract the relevant and characterizing data from features.
We noticed that some symbols were not displaying correctly; we decided to import the html functions to be able to encode and display the symbols correctly. Through the unescape function we were able to obtain the desired result. The next step was to make text uniform by transforming it all into a lower case.
In the various comments there were elements that penalize and complicate the classification; therefore we decided to remove that did not contribute to improving the performance of our model. To remove these parts we used regex, regular expressions that allows us to identify the chosen parts within the text and, through the lambda functions, remove them by replacing them with the value " ". We used this method to remove all URLs, i.e. all external links, mentions, i.e. all words starting with @, numbers, which were not needed to improve the model, and all punctuation repeated several times replace with the label "multiple_ + punctuation type".
Observing the large presence of emoticons within the different comments, we decided to import the emot library and replace emoticons with a label that represent each emoticon differently.
For tokenization we decided to use the TweetTokenizer, function imported from NTKL [2]. This module allows us to remove letters that are repeated too many times compare to the original word. After tokenization, punctuation must be removed. By importing the string.punctuation function from string, we obtain a list of all the most common punctuation marks which are then remove from our tokenized dataset.
To conclude the preprocessing part we used the TfidfVectorizer, function imported from the sklearn library [3]. We have applied the Tf-idf function, which gives the importance of a term with respect to a document or a collection of documents. The idea of this function is to give importance to terms that appear in the document but are rare in the collection.
After the Tf-idf process we create a vector that contains all the extracted features that we use to fit the model and we have concatenated the user column after transforming it into a sparse matrix as well. We also used the date column but we extracted only the days of the week and, after making it a sparse matrix, we concatenated it to the result obtained earlier. We observed how the numbers of comments varied as the days changed; the distribution of the number of comments during the week changes considerably; the number of positive comments is consistently greater than the number of negative comments. This comes from the fact that we did not re-sample before preprocessing. The Figure 3 shows the distribution graphically.

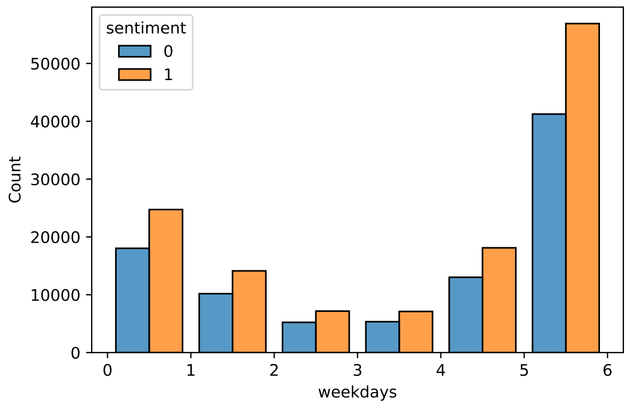

Fig. 3: number of positive and negative comments in the different days

B. Model selection

The choice of the best model to classify is important but also difficult because, depending on the preprocessing done, the different algorithms work and have different performances. The following algorithms have been tested:

- LinearSVC: the Linear Support Vector Classifier (SVC) [4] method applies a linear kernel function to perform classification and it performs well with a large number of samples. It is implemented in terms of liblinear rather than libsvm, so it has more flexibility in choosing penalties, loss function and should scale better across a large number of samples.
  This class support both dense and sparse input and the multiclass support is handled according to a one-vs-others scheme.
- RidgeClassifier: the ridge classifier [5], based on the Ridge regression method, coverts the label data into [-1 , 1] and solves the problem with regression method. The highest value in prediction is accepted as a target class and for multiclass data multi-output regression is applied. Ridge works by penalizing the magnitude of the characteristic and minimizing the error between predicted and actual observation; it works by performing the L2 regularization that add a penalty equivalent to the square of the magnitude of the coefficients. This is called "regularization" technique.
- LogisticRegression: logistic regression [6] is a processing of modelling the probability of a discrete outcome given an input variable. It is a useful analysis method for classification problem, where you are trying to determinate if a new sample fits best into a category. Logistic regression is a powerful supervised machine learning algorithm used for binary classification problem. It use essentially a logistic function to model a binary output variable. The logistic regression range is between 0 and 1 and does not require a linear relationship between input and output

variables; this is due to the application of a non-linear logarithmic transformation.

To observe which model actually worked best, we imported cross validation score which, after setting the right hyperparameters, allows to divide the set into several parts and for each iteration train the model on a part and test it on the other.

C. Hyperparameters tuning

Choosing the right hyperparameters is fundamental as it significantly increases or decreases the performance of the chosen model. In this project we have decided to focus on the tuning of the parameters of cross validation, Tf-idf and logistic regression.
In cross validation, the main parameter to be analyzed is the ”cv”, which is used to determinate the strategy of subdivision of the cross-validation. We decide to adopt a cv= 5 which divides the set into five parts and trains the model on four and tests it on one. At each iteration, the parts that train the model and the one it is tested change,
To improve the results of the Tf-idf we decided to analyze the following parameters:

- Max_df: setting it to 0.5 we only take into account words that appear in less then half of the documents. This operation is to avoid considering unnecessary words for classification but in common use.
- Ngram_range: this parameter is used to decide if you want to extract only one word at time or several word together. Setting it to (1,2) we decided to take both single word and word pairs.
- Max_features: setting this parameter as a result we obtain a matrix with n features. In this project we have decided to set n = 60000.
- Sublinear_tf: this parameter, if set, allows the standardization of the comments thus making them all with the same weight.

Tuning in Logistic regression is quite limited. To tune the hyperparameters of the logistic regression we used the HalvingGridSearch function. The function, after setting the parameters to be modified, returns the best value of each parameter in relation to the other parameters. We decided to use halvinGridSearch instead of GridSearch because, even if in an experimental state, it allows a faster hyperparameters tuning by initially testing a small number of candidates and subsequently implementing the number of them using more and more resources. The choice of some parameters was made without tuning as, taking into account the characteristics of the data available to us, they were more adequate. The results obtained by the hyperparameters tuning of the logistic regression are shown in Table 1

parameters	best chiose	motivation
Max_iter	10000	the model converges
Dual	False	number of samples greater than number of features
Solver	Saga	hyperparameters tuning
Class_weight	Balance	to give equal weight to both classes

TABLE I: Hyperparameters tuning

III. Results

The results obtained from this project are quite satisfactory. The 3 models had very similar performances to each other: we got a score of 0.829 with Logistic regression, 0.81 with LinearSVC and 0.815 with RIDGE. Table 2 summarize the results obtained by the different models through different measurements.
The decision to use Logistic Regression for this model is that it requires little hyperparameters tuning and the execution speed is high. As Figure 4 also shows, the results obtained from the logistic regression are better than the other models. Using the right combination of hyperparameters for preprocessing and logistic regression we were able to obtain a result with good performance both in terms of execution speed and performance. Through Figure 5 we can observe the words that have influenced the ranking the most. The word cloud represents, through the coefficient of each word, the impact that the word has on the model. The largest represented words are those that most influence the model.
We can observe how the result obtained on the Leaderboard is the same as that obtained locally; this is a positive sign because theoretically our model does not overfit.

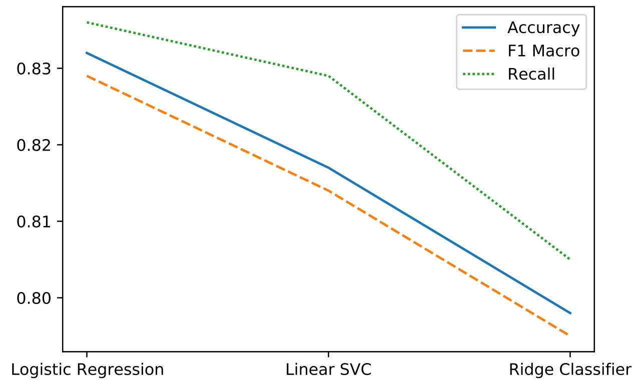

Fig. 4: model performance

Model	F1 score	Accuracy	Recall
Logistic Regression	0.829	0.832	0.836
Linear SVC	0.814	0.817	0.829
RIDGE Classifier	0.795	0.798	0.805

TABLE II: Model results

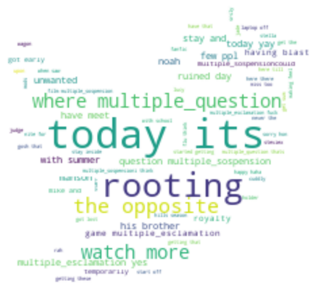

Fig. 5: wordcloud

IV. Discussion

A first consideration should be made on the use of users for classification. The presence of users significantly increases the score obtained, passing from 0.78 to 0.829 using Logistic regression. A possible reason for this increase in the score is that the users present in the development set are the same as those present in the evaluation set. However, it must be emphasized that the aggregation of users for classification only works in a specific problem like ours, where the users remain almost unchanged.

Making a general discourse, the users would not affect the classification but would make it slower because unnecessary data would be added to the features that should be processed. A possible different approach to this problem could be to use two classifier: one for the final part of the preprocessing and one for the creation of the model. The first classifier could be used after the Tf-idf for the extraction of the most relevant features for the model, thus skimming the excessive number of features obtained after vectorization. The second classifier would have classified the result obtained by the first one. We tried this approach to the project but the indexing of the sparse matrix to be concatenated to the users was complex and the results obtained were lower than the normal classification.

We probably have not optimized this idea as well as possible and therefore the results were not entirely satisfactory.

References

[1] P. Lemenkova, "Python libraries matplotlib, seaborn and pandas for visualization geospatial datasets generated by qgis," *Analele stiintifice ale Universitatii" Alexandru Ioan Cuza" din Iasi-seria Geografie*, vol. 64, no. 1, pp. 13–32, 2020.

[2] N. Hardeniya, J. Perkins, D. Chopra, N. Joshi, and I. Mathur, *Natural language processing: python and NLTK*. Packt Publishing Ltd, 2016.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[4] S. Sulaiman, R. A. Wahid, A. H. Ariffin, and C. Z. Zulkifli, "Question classification based on cognitive levels using linear svc," *Test Eng Manag*, vol. 83, pp. 6463–6470, 2020.

[5] A. Singh, B. S. Prakash, and K. Chandrasekaran, "A comparison of linear discriminant analysis and ridge classifier on twitter data," in *2016 International Conference on Computing, Communication and Automation (ICCCA)*, pp. 133–138, IEEE, 2016.

[6] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of biomedical informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.