Data Science Lab: Process and methods

Politecnico di Torino

Project report
Student ID: s276798

Exam session: Winter 2020

Legend : the references to the code are made using (line xxx), when the script is not specified it refers to tripadvisor.py

The purpose of the project is to analyze a dataset of 28754 rows and two columns, one containing reviews about hotels and the other containing a label ('pos' or 'neg') that classifies the review as positive or negative. After the training of the model, the objective is to classify the reviews contained in the evaluation set.

1.Data exploration

The files used for this work can be retrieved at *[1]*

The unzipped file contains three different documents :

- Development.csv : containing the labeled reviews
- Evaluation.csv: containing the unlabeled reviews
- Sample_submission.csv: file that have to be written with the predicted labels

A brief look at the dataset shows some problem with the classification task:

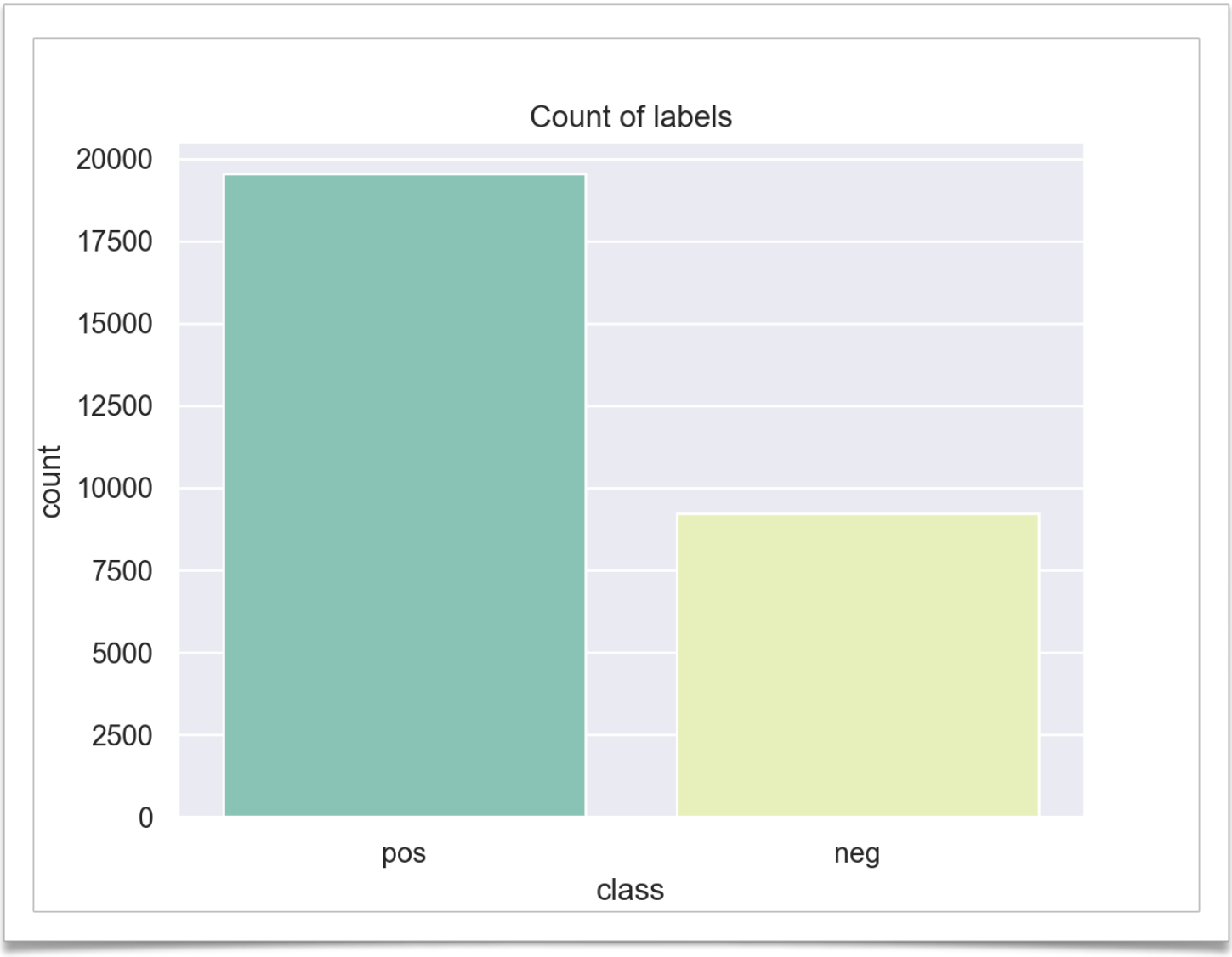
- Presence of grammatical errors, such as wrong use of punctuation, sentences without sense, wrong use of the tenses, lexical errors and so on
- Presence of emoticon
- Presence of reviews in different languages from Italian, such as French and English especially
- Presence of 'neutral' reviews labeled as negative or positive

While the first three problems can be solved with some preprocessing steps, the third problem is more difficult to handle. After the data exploration, only the inspection of the labels distribution will explain how the 'neutral' reviews are labeled. Nevertheless the classification will be affected by this ambiguity.

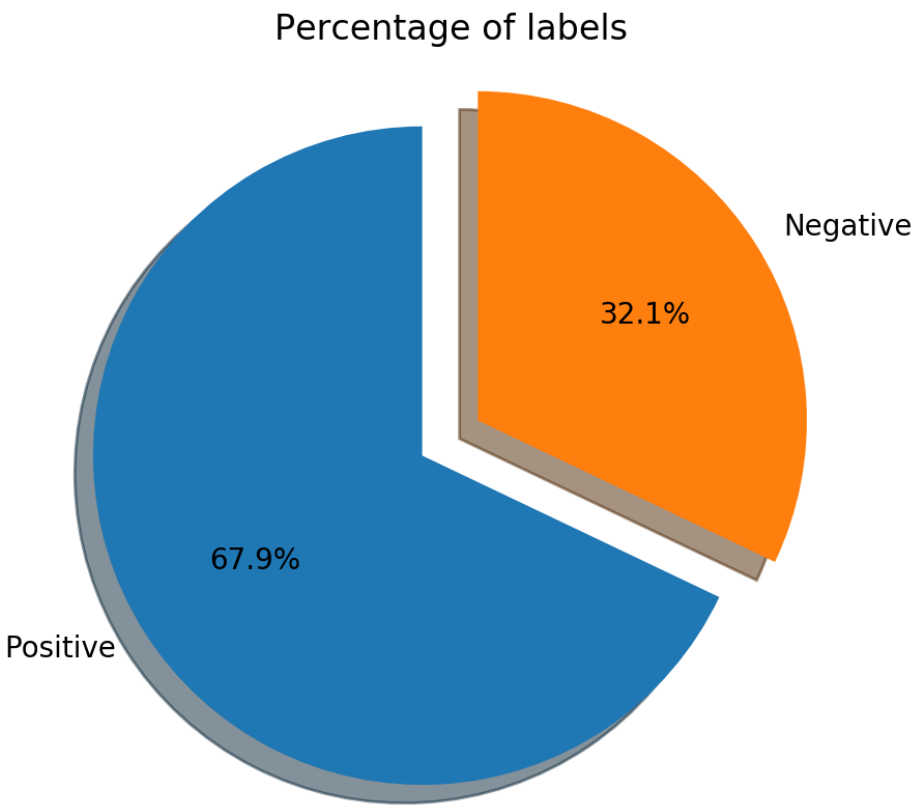
Data were loaded into two different pandas DataFrame, one for the development set and the other for the evaluation, then preprocessed and saved into two different files, 'd_clean.csv' and 'ev_clean.csv' (line 29 - 104).

Since the preprocessing of data requires at least 5 minutes, this passage seemed to be necessary for a faster processing of the entire algorithm. Using the *csv* library provided by python the cleaned data were loaded into three lists: files containing the labeled reviews, labels containing the labels of the reviews and evaluation_set containing the unlabeled reviews.

After that, the distributions of development set can be finally analyzed. At first an histogram is used to inspect the count of positive and negative reviews.



The histogram (line 137) shows that the bigger class is the positive one, with more or less 20000 reviews. The negative reviews seem to be about 9000. Observing this graph is clear that the ‘neutral’ reviews are probably labeled as positive.



In terms of percentage, the distribution of labels (line 125) :
The dataset is imbalanced, 70 % of reviews are positive, 30 % are negative. For this reason the positive words will be more numerous and heavier than the negative ones. The classification will be affected by this problem, in particular many negative reviews will be wrongly labeled as positive. This problem can be in part handled with the usage of class weight in the classifier.

2. Preprocessing

Cleaning and lemmatization

At first some techniques are used to remove emoticons and special characters, since this kind of characters could compromise the analysis. Then every word is converted to lower case (line 29 - 66). The cleaned data are obtained using the code from an open source project in GitHub *[2].*
After that, the words were lemmatized using the TreeTagger provided by *treetaggerwrapper* package (line 73 - 104). The TreeTagger *[3]* is a tool for annotating text with part-of-speech and lemma information. This tool was used because it is one of the few that provides lemmatization of the words and inspection of the POS tagging for Italian language. The lemmatization is fundamental for the creation of a good model, indeed it provides the transformation of every word to his primitive form.
An example of TreeTagger' s output is :

word	pos	lemma
Siamo	VB	essere

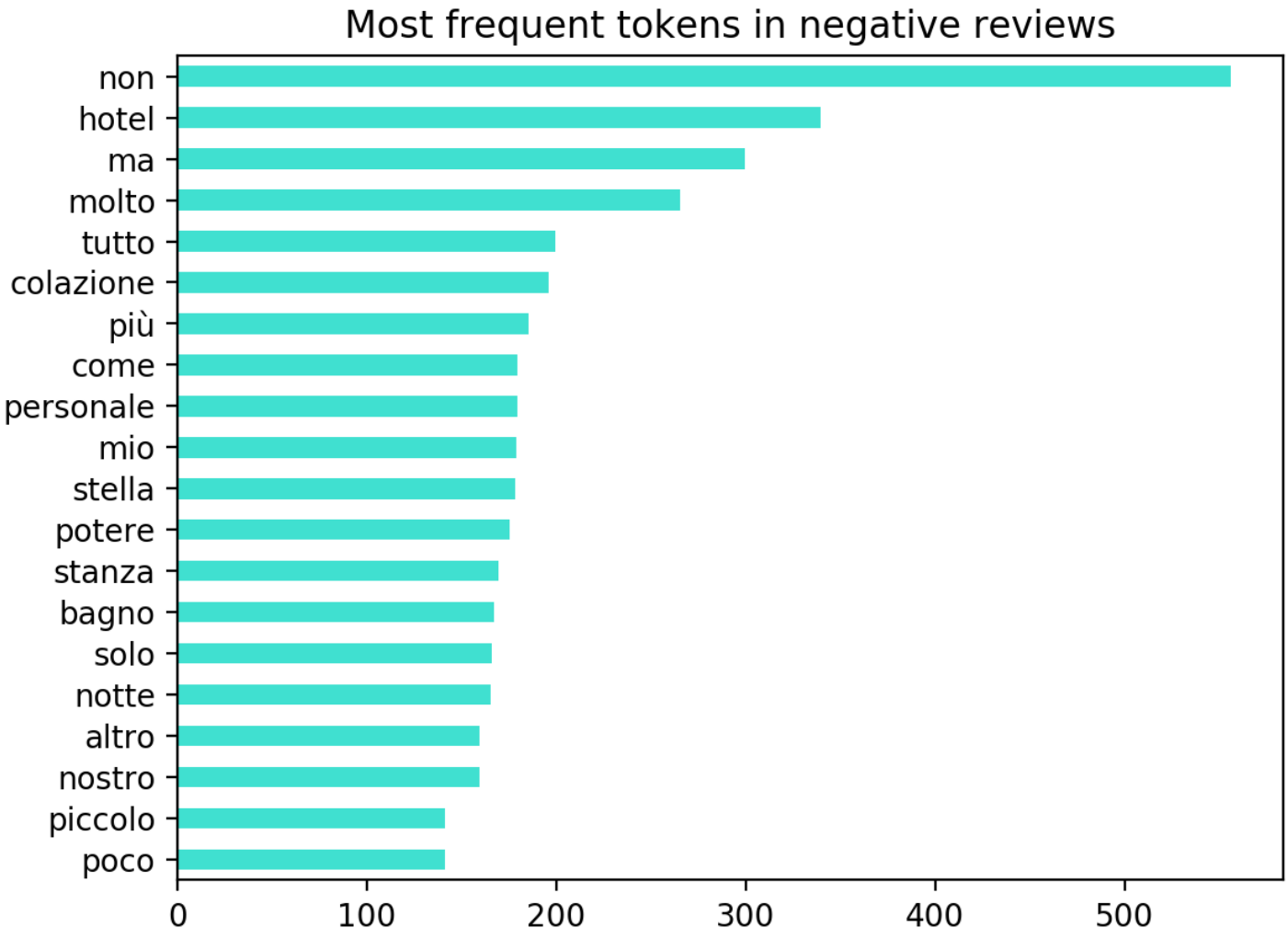
Vectorization

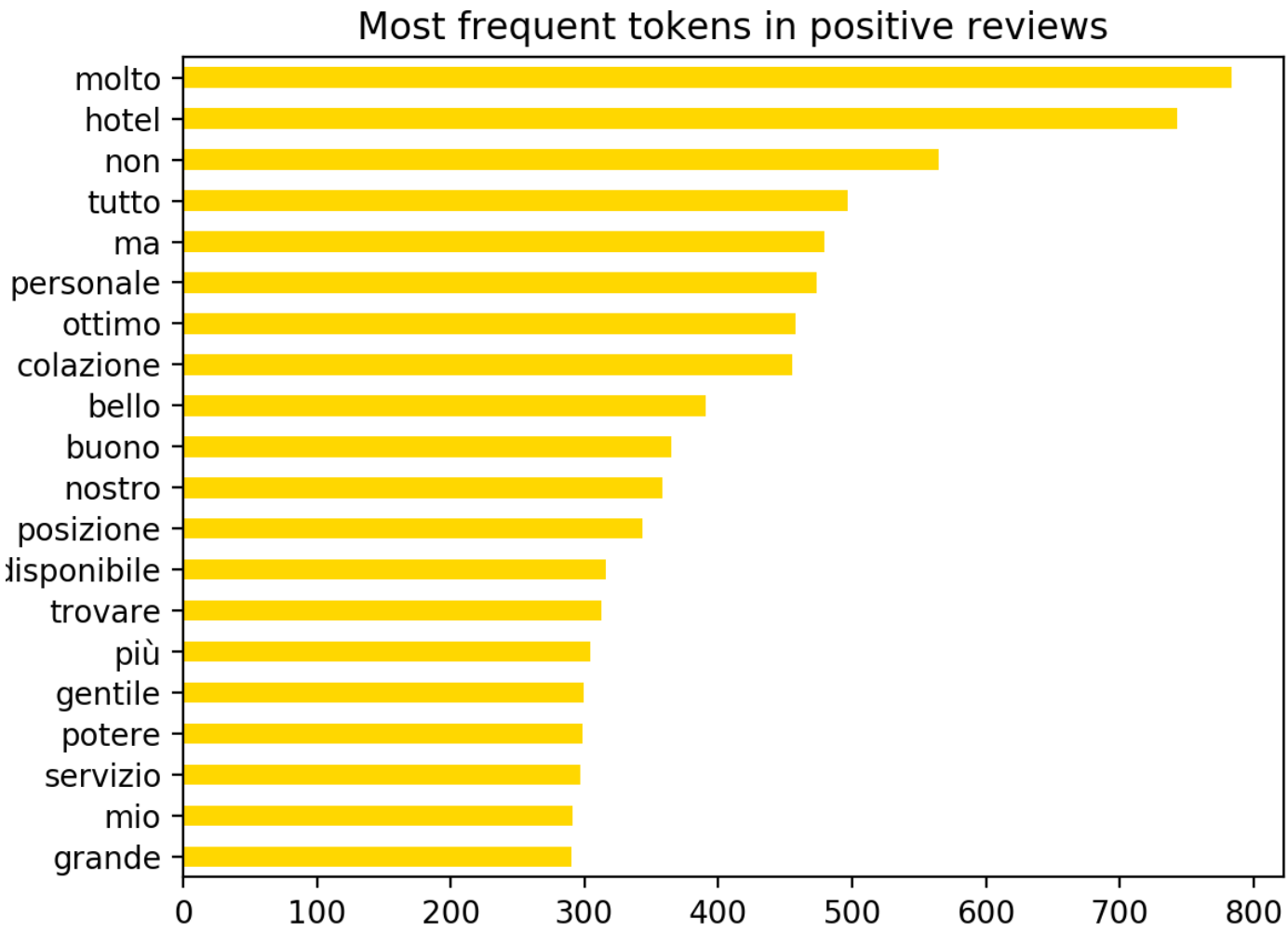
For classification is necessary to extract features from documents with the transformation of text (categorical data) into floating points values. This can be done using the TfidfVectorizer provided by *sklearn* library (line 144). It generates a sparse matrix, in which rows correspond to the documents and columns to the words. Each cell is the Term Frequency Inverse Document Frequency.
Vectorizer has some important parameters (the choice of the first three parameters will be analyzed in the fourth part of the report):

- min_df = 2
- max_df = 1.0
- n_gram_range = (1,3)
- stopwords = lemmaTokenizer.stopwords
- tokenizer = lemmaTokenizer

Let's focus now on lemmaTokenizer, it provides the last preprocessing step before the vectorization of the words. It is useful to remove irrelevant or ambiguous words for the analysis. The best configuration obtained is the one that removes all the words with length lower than one (line 28, tokenizer.py), due to the fact that conjunction are irrelevant for the analysis, but the elimination of words longer than one character decreases the accuracy. Indeed words like 'ma', 'si', 'no' can change the polarity of a sentence. Stopwords are the words removed from the analysis because of their ambiguousness or their irrelevance.
The presence of these words can badly affect the analysis. The set of stopwords was updated during the improvement of the algorithm, using an

empirical method : most frequent words in positive and negative set were printed with their weight, then words with big weight contained in both the group were added in ‘Stopwords’.
The best configuration is in line 10 - 21, tokenizer.py.
Good results are also obtained adding the words ‘camera’, ‘dovere’, ‘dire’, ‘chiedere’ in the set of stop words (Solution 2).
Now take a brief look to the more frequent words (in terms of tfidf) in the development set for both the labels.





The barchart shows many words contained both in positive and negative reviews. This is a suggestion for using not only unigrams, that can be ambiguous, but also bigrams and trigrams, for a better classification of the reviews.

3. Algorithm choice

The algorithm used is the PassiveAggressiveClassifier *[4]*, a linear classification model provided by *sklearn* library. The algorithm is useful to classify massive streams of data. It works better with a large amount of values, this can be proven with a feature reduction performed by the TfidfVectorizer. Every time that a dimensionality reduction is performed, the computational time increases and the accuracy score decreases.
The idea behind Passive Aggressive is intuitive; passive: if the classification is correct it keeps the model; aggressive: if classification is incorrect, it updates to adjust to this misclassified example. In passive, the information hidden in the example is not enough for updating; in aggressive, the information shows that at least this time you are wrong, a better model should modify this mistake.
Good results in term of accuracy are also obtained using SGDClassifier *[5]* or LogisticRegression *[6]*, both provided by *sklearn* library. The first one uses the Stochastic Gradient Descent method to solve a problem. The class SGDClassifier implements a plain stochastic gradient descent learning routine which supports different loss functions and penalties for classification.
Last but not least the LogisticRegression. In terms of computational time and accuracy is similar to the SGDClassifier. Good results are obtained because it is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.
After some trials, the results are always the same : in terms of computational time the algorithms behaviors are equals, while in terms of accuracy the PassiveAggressive seems to be the best one. In addition, both SGD and LinearRegression need a features reduction to better perform, while the PassiveAggressive works better with a large amount of data.

4. Tuning and validation

Tuning is performed using a function defined in the main script (line 281) . At first the training set is divided into 5 different folds, using StratifiedKfold provided by *sklearn*. Fold division is useful to repeat many times the search of best parameters. This repetition will guarantee a stronger result. Every fold is then divided into training and test set. The next step is the creation of a pipeline containing TfidfVectorizer and PassiveAggresiveClassifier.
Is really important to re-extract the features after Kfold, to avoid an impropriate usage of the tfidf computed on the entire training set. Before usage of *sklearn* method GridSearchCV, a set of candidate parameters is defined. The method mentioned before will evaluate this parameters set. When fitted on the training data, GridsearchCV finds the best set of parameters, that will be used automatically in line 309 to predict the test set labels. Finally the F1-score is computed using the labels of the test set and the prediction. If F1-score is bigger than the best score, the configuration is saved and returned by the function.
With the computation of F1-score the result are already validated, also for this reason the Kfold is fundamental. It provides 5 different F1 values, that make stronger the evaluation of the model.
The best set found by the method is the following one:

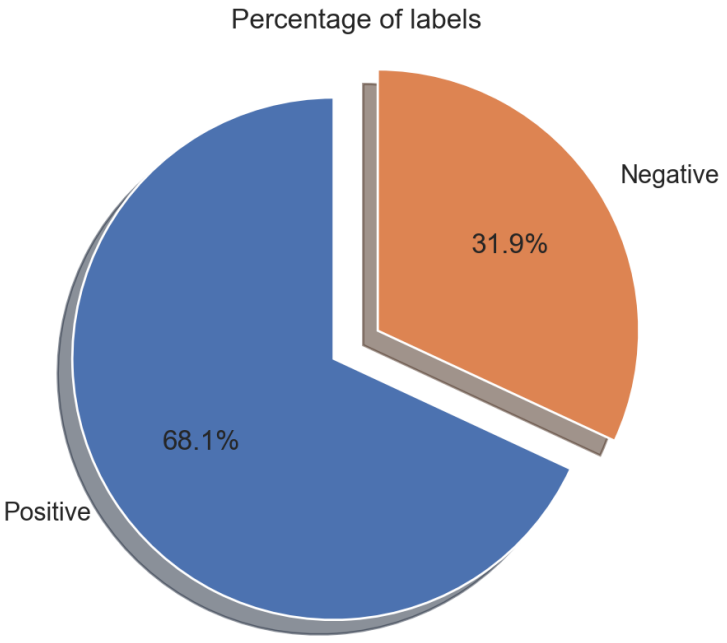
- TfidfVectorizer:
  - min_df = 2, words contained in less than 2 documents are ignored
  - max_df = 1.0, words contained in less more than 100% of the documents are ignore
  - ngram_range = (1,3), Unigrams (one word), Bigrams (two words) and Trigrams (three words) will be considered for the analysis
- PassiveAggressiveClassifier:
  - fit_intercept = True, intercept will be used in calculations
  - class_weight = {‘pos’ : 0.6792794046045768,
    ‘neg’: 0.3207205953954232}, those values reflects the distribution of the labels showed in the first section of the report.

The best value computed by F1-score is 0.9685817235644748, that seems to be a good result for the classification.

At the end of the algorithm the predicted labels are saved into a csv file, (line 226).
A good way to evaluate the model is to generate a Wordcloud, using *wordcloud* library, about the more frequent positive and negative words found in the evaluation set (line 241).
The results are the following:

The result seems to be good, a great presence of positive words can be noticed. The presence of neutral words at appearance such as ‘colazione’ is probably due to the usage of Bigrams and Trigrams in which those words can be used with positive adjective, pronoun or adverbs.

Looking at the negative WordCloud we can do the same considerations made for the positive one.
We can also look at the distribution of the predicted labels, we expect a distribution more or less equal to the one of development set (line 192).



The percentage are more or less the same as the development set. The effectiveness of the model is proven.

5. References

[1] *Dataset* used for the analysis : http://dbdmg.polito.it/wordpress/wp-content/uploads/2020/01/dataset_winter_2020.zip

[2] Data cleaning *:* https://github.com/giuseppegambino/Italian-Sentiment-Analysis-with-Spark/blob/master/tweetClean.py

[3] TreeTagger : https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

[4] PassiveAggressiveClassifier : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html

[5] SGDClassifier : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html

[6] LinearRegression : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html