Data Science Lab: Process and methods

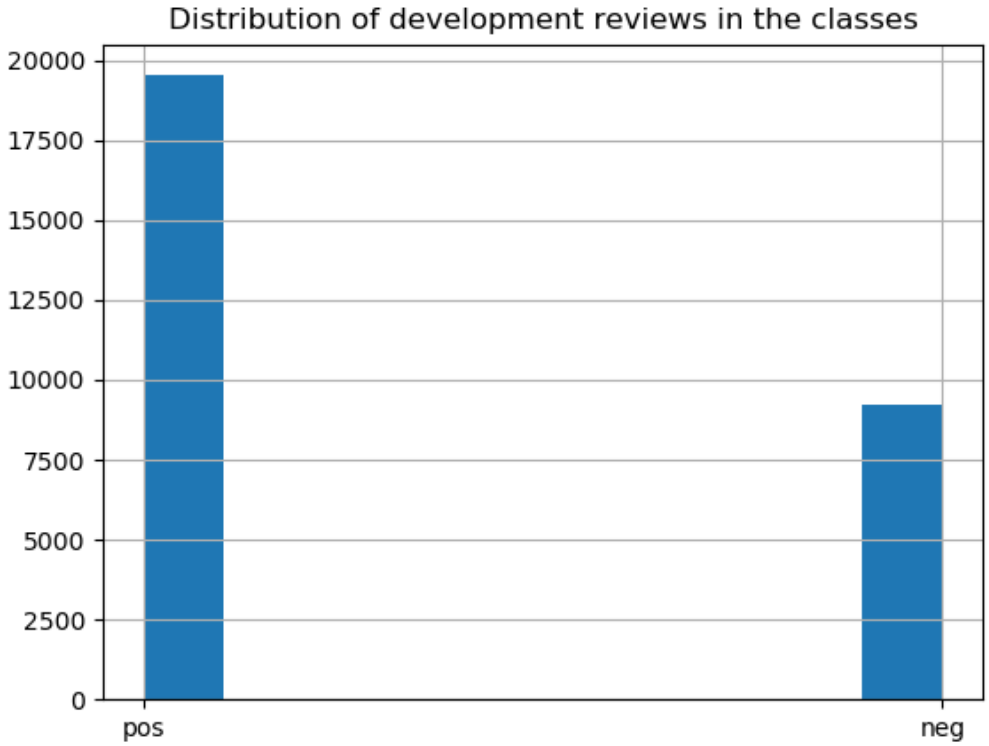
Politecnico di Torino

Project report
Student ID: s273672

Exam session: Winter 2020

1. Data exploration (max. 400 words)

The first step was loading the development dataset in a pandas *dataframe,* then we had to explore it to see if we could get some insights on the data.
Since the aim of the project is to classify a set of review from the *Tripadvisor* Italian website, I decided to plot the distribution of the two classes among all the reviews.



As it can be seen from the plot the dataset is really unbalanced in term of positive reviews, they are almost the 66% of the database, so I'll have to take this knowledge into account in the analysis.
Then i started to read manually some of them to get a first flavor of the reviews and how they are generally composed.
The reviews as we could expect have a lot of typos, misspelled word and punctuation,

but we can see that often personal names and cities names are present, so it's better to explore this characteristic more precisely and evaluate the percentages of presences of this kind of words.
The frequency of personal name in the dataset is approximately 5%, a quite high number, and counting the number of presences of some of the most visited cities in Italy we can see that for some cities like 'Venice' or 'Milan' we have high values, around 20% for 'Venice' and 13.5 % for 'Milan'.
All this information is going to be useful in the preprocessing phase.

2. Preprocessing (max. 400 words)

In order to give our classification algorithm a numerical matrix to compute the classification we have to extract from the development reviews a set of features.
This can be done through the *tf-idf vectorizer* of the *sklearn feature extraction* pack.
In order to process the reviews, I decided to implement my own tokenizer class, where I also lemmatized their content.
The tokenizer object at first takes away all punctuations of the document trough the *string translate* method passing as argument a table constructed with the *maketrans* function of the *string* module. Then using the *wordtokenize* tokenizer of the *NLTK* library, divides the document in tokens. The tokens are then stripped of eventual space, brought to lowercase characters through simple string methods and discarded if they are digits.
The tokenizer does a lemmatization process to reduce words to their dictionary form in order to have a less sparse set of features.
Since the reviews are in Italian and the *nltk lemmatizer* for Italian language wasn't performing well, I adopted the *spacy* library.
Spacy permits to do lemmatization of Italians words with quite good results, thanks also to his *part of speech* analysis.
After passing as the "tokenizer" parameter of the TFIDF this object, I passed the range (1,3) as the "ngrams" parameter, in this way I made the *tf-idf vectorizer* to take in to account also bigrams and trigrams of word, this could bring more information on the sentiment of the review.
I also needed to eliminate *stopwords* cause usually they don't bring useful information on the documents content, so I decided to use the Italian *stopwords* of the NLTK library.
To these *stopwords* I added a list of the most common Italian names and the names of some of the most visited cities in Italy, to address the problem found in the data exploration phase.
I also decided to take away from the *stopwords* some word that I thought could be meaningful in the analysis such as: "non, contro, perché, più'" and this actually increased the F1 score of my result.
In the end I lemmatized the *stopwords* as well and passed them to the *tf-idf vectorizer*.

Finally, I set the "min_df" parameter of the *tf-idf*, in this way I discard all that word that appear just in a few documents and that are probably meaningless for my analysis.

3. Algorithm choice (max. 400 words)

The *tf-idf vectorizer* give a matrix with a very high number of features, approximately hundreds of thousands, so I had to choose an algorithm that could deal with such a high dimensional matrix and with high accuracy.
I implemented a random forest classifier, but it didn't give really good results. I then implemented the *linearSVC* classifier of the *sklearn* library, this is a Support Vector Machine classifier with parameter 'kernel' set to 'linear' and implemented in a way that should scale better with large number of samples.
I also implemented a logistic classifier, that is usually used in spam filtering and a *Naïve Bayesian Gaussian classifier,* all of them are from the *sklearn* library.
By testing all of them with *k-fold cross validation* the one that emerged performing better was the *linearSVC*, so I adopted it in the final version of my algorithm.

4. Tuning and validation (max. 400 words)

To tune the total algorithm, I proceeded step by step, starting to evaluate the result if I changed the content of my *stopwords*. I then observed that if I kept out some city names from the *stopwords* my results on the *F1 score* improved.
For example, not removing 'Milano' improved my score. This could be due to the fact that some of these names could have been lemmatized to some other word by the lemmatizer.
So, I decided to create two different types of algorithm, one more general where I kept the cities names and one more fitted to the test dataset where I tuned specifically the *stopwords* to optimize my results.
This was also done through a *wordcloud* visualization that permitted me to see easily the words that had the highest *tf-idf* values in the two classes.
Here are the results after the tuning of the *stopwords*, printing the top 100 words in *tf-idf*

neg class

pos class

There are some common words like ‘molto’ and ‘non’ that could be discarded from the analysis, but this worsened my results, probably because the bigrams where the words “non” and “molto” are present have usually an important meaning. I then decided to not discard these words. Instead by discarding words as “hotel” and “camera” I improved my results.

I then tuned the “min_df” (minimum document frequency) parameter and the *linearSVM* parameters with a grid search to optimize my results on the test dataset. These are the result in terms of *F1score* and *confusion matrix* of my final algorithm on a test set of size 25% of the total evaluation test:

```
F1_SCORE:0.9666780174598626
Conf_matrix:
[[4851  101]
 [ 138 2099]]
```

5. References

[1] Spacy library URL: https://spacy.io/models