Data Science Lab: Process and methods

Politecnico di Torino

Project report

Student ID: s277757

Exam session: Winter 2020

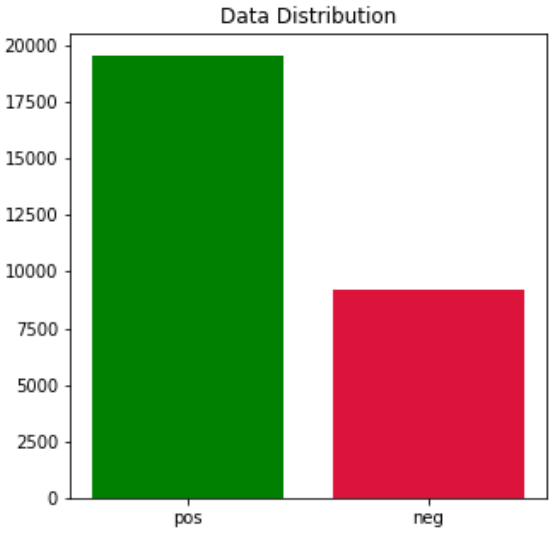
1. Data exploration (max. 400 words)

The dataset provided is a collection of reviews divided in two blocks: the first one called 'development.csv' which has for each review a label that corresponds to the class where the review belonging to and a second one called 'evaluation.csv' that is unlabelled; to give it a class will be our goal.

The reviews are from Tripadvisor.com, in particular these are written in Italian language.

Avoiding for the moment the evaluation dataset, we focus now on the development one to explore its data.
At a first look, exploiting some *Pandas* methods like '*head()*' and '*describe()*' (since data have been imported as Pandas data frame) we can see that the dataset is complete, in sense that neither a label is missed.

Given this first good news, I can now look at the labels' distribution and for doing that I built a function which plots a bar char representing our data. The result is in the following picture.



As we can see there is not balancing between the two classes, the positive reviews are more or less two times the negatives.
Dealing with an unbalanced dataset could be a problem because it may cause an overfitting.
Since in this case even a single word review like 'ok' or 'no' could be enough to identify clearly the class, I decided to take for the beginning the whole dataset and postpone the problem after having seen the classifier's behaviour.

2. Preprocessing (max. 400 words)

Having to deal with text files, requires following a few steps to transform the content in vector of numerical features because the classifier is not able to work directly with words.

First of all, I built a class called *ItalianStemmerTokenizer* aimed to clean the reviews from digit chars and punctuation; to apply the stemmer in order to normalize my dataset and as last step, to split each review in a list of tokens.

Here I found that actually *nltk* is little optimized for the Italian language, no lemmatizer are provided and the stemmer coming from *nlkt.stem.snowball* [1] is definitely slower than the English one, but by the way, it is the only choice, therefore I used it.
It follows an example of a processed review, before the vectorization:

```
'stat qui nott prim sinistr crocier venez personal molt gentil dispon
cam pul confortevol colazion eccellent aiut trasport priv findng
dirett terminal croc prezz molt buon'
```

To represent my reviews as numerical features I exploited the *TfidfVectorizer* from *sklearn* that gave as output a matrix of TF-IDF values. I passed here my stemmer - tokenizer and I set ngram_range = (1, 2) for counting also how many times a couple of terms (called 'bigram') occurs, for example looking the previous review it is not hard to understand that '`molt buon`' kept together can determine the relative class. I also did an attempt with (1, 3) but it didn't improve the result. My decision has been to don't modify *max_df* in order to consider all the most frequent terms because they can help to give a meaning and to set *min_df* = 8 in order to discard some unusual terms that wouldn't provide any useful information. Regarding the stop words, I used the stop - words library [2], that is better than the *nltk* one.

I decided to try to perform a dimensionality reduction with *TruncatedSVD*, but it didn't improve the score achieved without it, so I left this idea.

After my first trial, the program gave me a warning about many stop words belonging to reviews and I added them in the stop words list.

Since with a dedicated class for cleaning text (which is *ItalianStemmerTokenizer*) made the code's execution pretty slow, I tried to perform the pre-processing steps (cleaning, removing stop words and stemming) externally and before the tokenization in order to pass at the *TfidfVectorizer* the reviews already cleaned. Well, it was definitely faster (we talk about 2 min faster) but it reduced a little the classifier's score, so I decided to approve the 'class' version.

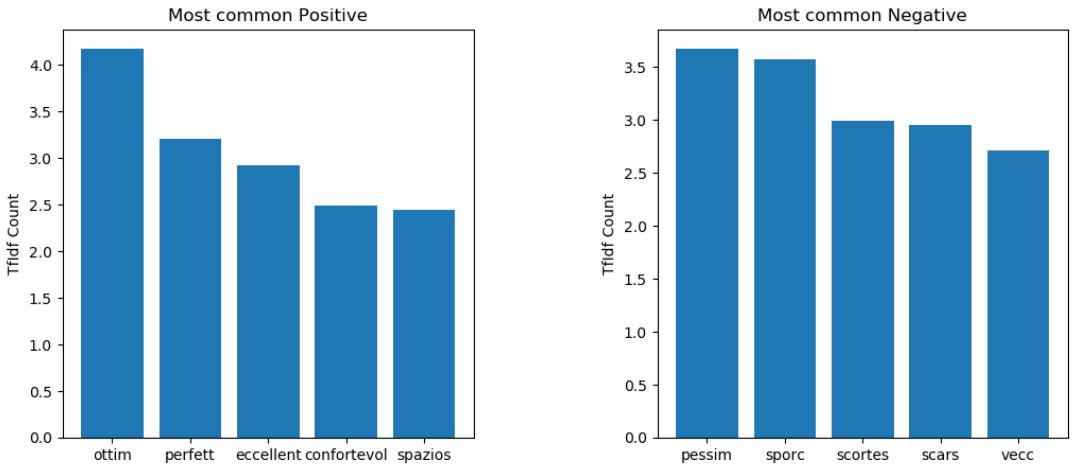
3. Algorithm choice (max. 400 words)

Among many available algorithms provided by *sklearn* [3], I found a group of these in order to perform my classification, test them and select the one which achieves the best result according to the metric in use for the evaluation (*f1 score weighted*). The tested algorithms were:

- *RandomForestClassifier*: already met in our laboratories, it scored close to 0.85, but it was really slow;
- *GradientBoostingClassifier*: it didn't provide any result because after more than 5 minutes it didn't converge, so I decided to break the process and discard it;
- *MultinomialNB*: another classifier from *sklearn.naive_bayes* which scored more than 0.8;
- *LogisticRegressor*: this classifier gave me a huge result, more than 0.9;
- *LinearSVC*: one of the best classifiers tested for my purpose that gave me an f1 score = 0.96.
- *SDGClassifier:* it is a variant of *LinearSVC*, it provides the best score overall;

In this step each algorithm has been started with default hyperparameters just in order to find the best one which is the most suitable for my dataset. Since *LinearSVC* and *SDGClassifier* have similar performance, I took both of them for the tuning part, that allowed me to improve again the result.

I also made a function that plots in a bar chart the most common terms that are present either in negative or positive class attributed by the classifier in order to verify if it gets the meaning (this function allowed me also to remove many useless stop words just watching if in the top *x* terms there had been useless words) :



As we can see the classifier gives a really good interpretation about the meaning of our reviews (obviously terms are stemmed by the pre-processing process).

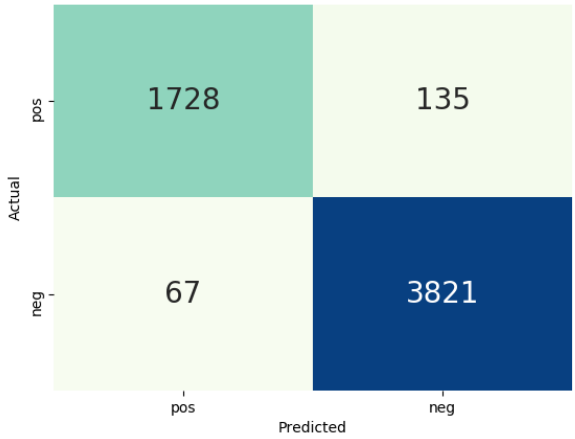
4. Tuning and validation (max. 400 words)

After have chosen as algorithms the *SDGClassifier* and *LinearSVC*, in order to improve the result obtained with the default values I performed a grid search cross validation over the hyperparameters that make sense to modify in these classifiers which are '*loss*' that is the loss function to be used , '*warm_start*' and '*penalty* that is the regularization term regarding the first one, and on the hyperparameter *C* (that is the regularization parameter) for the second one. Then I defined a list of values that those hyperparameters can assume in order to test which configuration is the best one. Obviously as score in the grid-search I have adopted the weighted f1 score as reference metric.

The best output was obtained using the *SDGClassifier* with hyperparameters: {'loss': 'modified_huber', 'penalty': 'l2', 'warm_start': True} with an f1-weighted score higher than 0.96 that is really good.

Then I fitted the final classifier (that will predict the evaluation data) with the previous setting and using the *classification_report* [4] method from *sklearn* I saw that also *accurancy*, *precision*, *recall* and the '*macro-metrics*' are higher than 95%. This means that even though our first decision was to avoid balancing the dataset, the classifier is definitely not overfitted.

As last validation I built the confusion matrix for watching how many positive reviews have been assigned in the negative class and vice versa:



As we can see only a few reviews are in the wrong class and it is a very good result.
The mistakes are legit because it can happen that a review made by a customer has been written for example with 'sarcasm' and so it can contain negative terms that mean 'good', or positive terms that mean 'bad', or again that a review does not express clearly its meaning.

5. References

- [1] NLTK. Stemmers. URL: https://www.nltk.org/howto/stem.html;
- [2] stop-words. stop – words. URL: https://pypi.org/project/stop-words;
- [3] Scikit-learn. Classification. URL: https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- [4] Scikit-learn. Metrics, Classification report. URL: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html