Data Science Lab: Process and methods

Politecnico di Torino

Project report
Student ID: s277882

Exam session: Winter 2020

1. Data exploration

First of all I analyzed the development set counting the number of reviews labelled with *pos* or *neg* and checked that none of them had missing class or text. Then I explored both the development and the evaluation set plotting the distribution of their review lengths realizing they have the same distribution (Figure 1.1, Figure 1.2).

Then, considering only the development set, I created a histogram (Figure 1.3) trying to find a correlation between the class of a review and its length but the result was that both positive and negative reviews follow the same distribution as the length increases, so I decided not to consider the length of a review as a meaningful feature for the sentiment analysis.

Besides I built a set of emojis and emoticons searching for the ones that mostly appears in positive reviews and I built an analogous set for negative reviews: to make these two sets I tried not to consider the ambiguous ironic emojis and emoticons. I developed a similar research, using regular expressions, for two punctuation characters: *'?'* and *'!'*, even written more times in a point of the text (i.e. *?? !!*). The result of this research, shown in Figures 1.4, 1.5, is that the more are the *'?'* written in the same point, the more probable is the review to be negative; although this doesn't happen with *!*, the presence in a review of at least one *'!'*, without another *'!'*, makes it more probable to be a positive one. These last two analyses led me to a choice in the feature extraction explained later.

I also made sure of the presence of proper names of people [1] [2], days and moths in both sets and I decided to add them in the stopwords taken from [3]. I also discovered a significant presence of cities and places names, as expected from this kind of dataset, but I decided not to include them in the stopwords after having tried it with a worsening in accuracy in the validation phase: that may even have sense if we suppose that in certain places or cities it's more probable to find good or bad hotels.
A further analysis I made was plotting two wordclouds of the datasets after having tokenized, lemmatized and stemmed all the reviews. The result shows they are basically the same, so the two sets are coherent (Figures 1.6, 1.7).

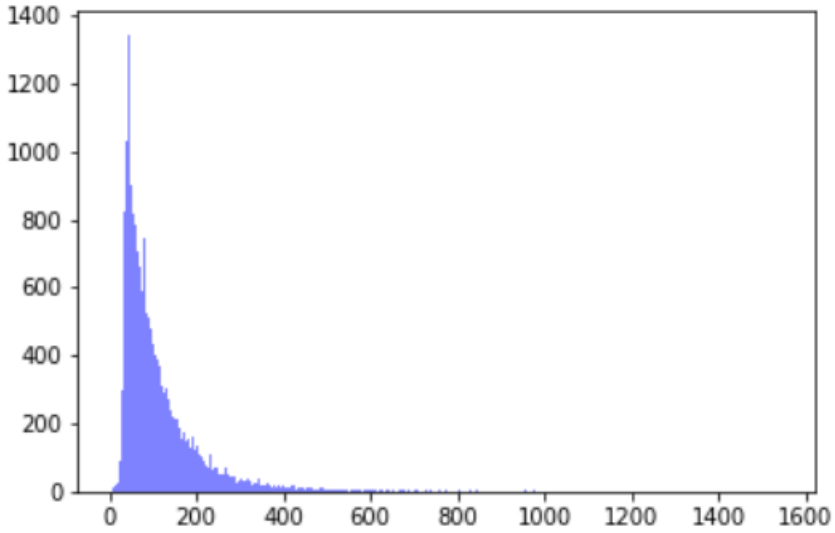

Figure 1.1 *Length distribution, measured in number of words, of the development reviews*

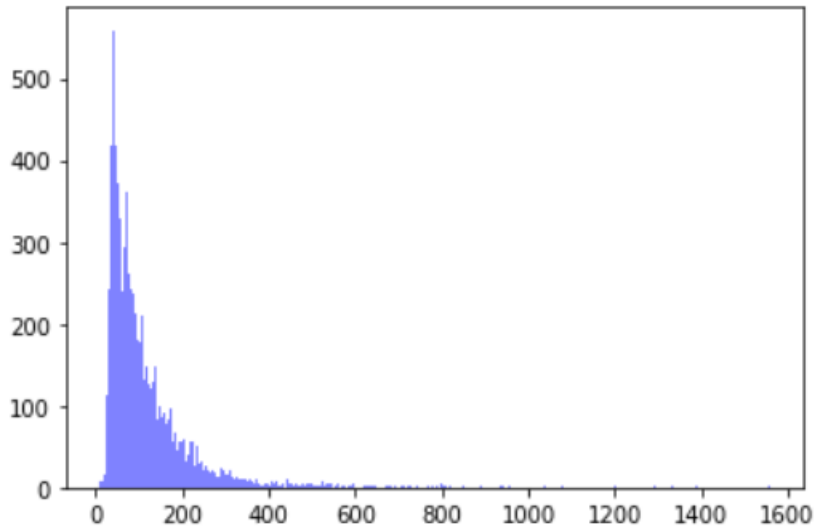

Figure 1.2 *Length distribution, measured in number of words, of the evaluation reviews*

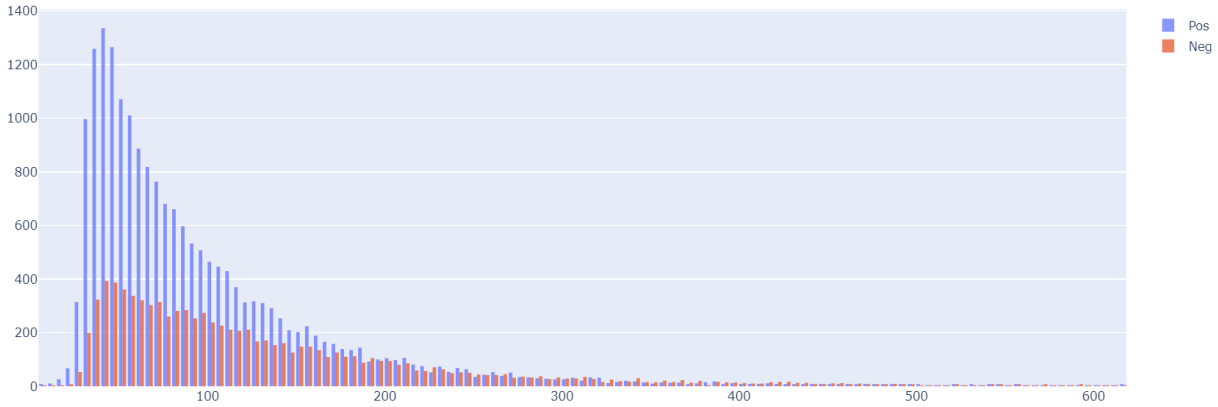

Figure 1.3 *Distribution of review lengths, measured in number of words, based on labels*

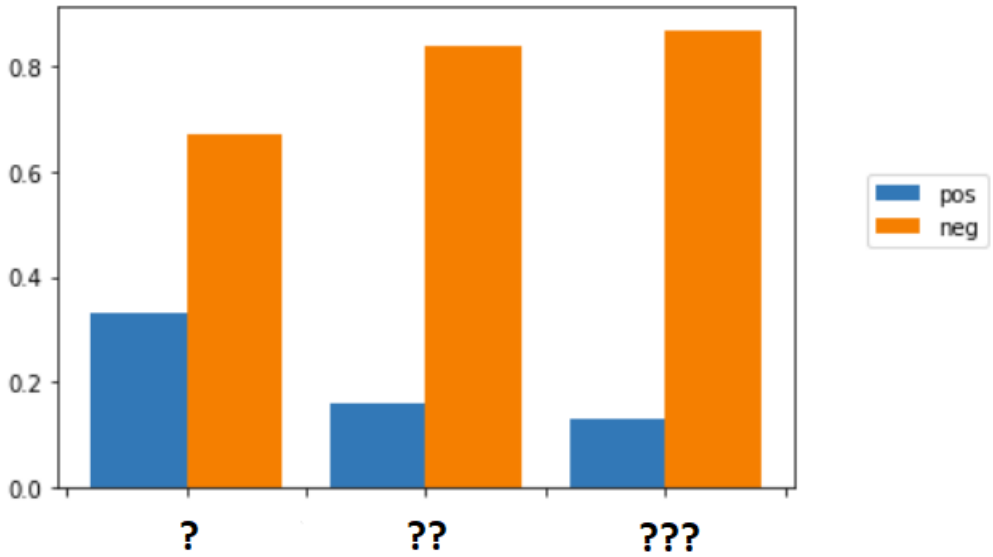

Figure 1.4 *Positivity and negativity in % of reviews containing ?, ?? and ???*

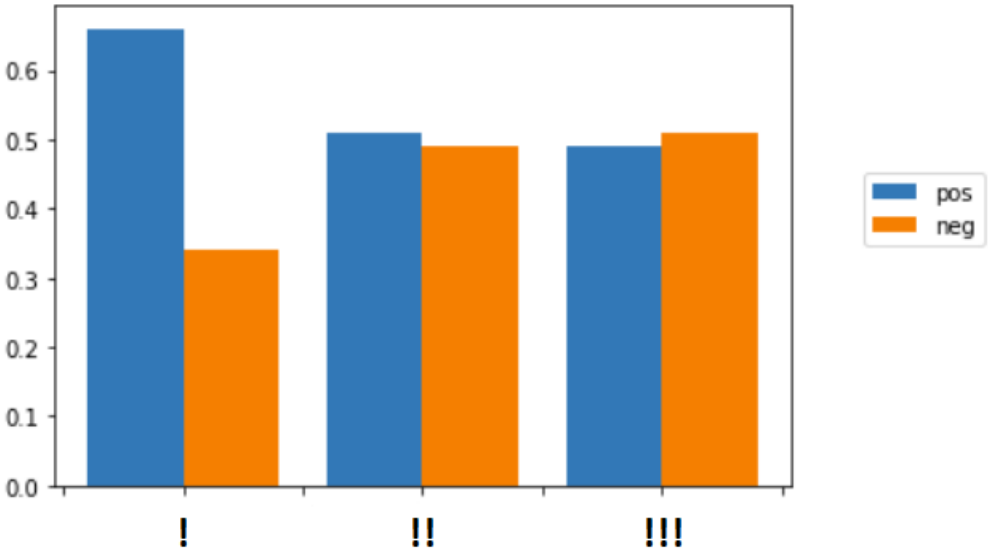

Figure 1.5 *Positivity and negativity in % of reviews containing !, !! and !!!*

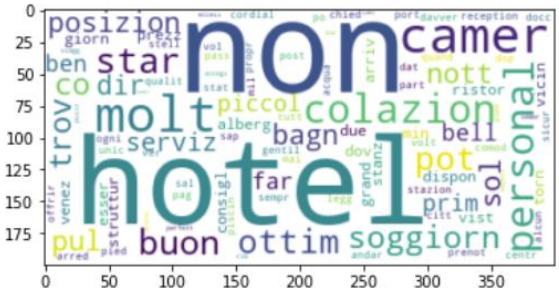

Figure 1.6 *Wordcloud of labelled reviews*

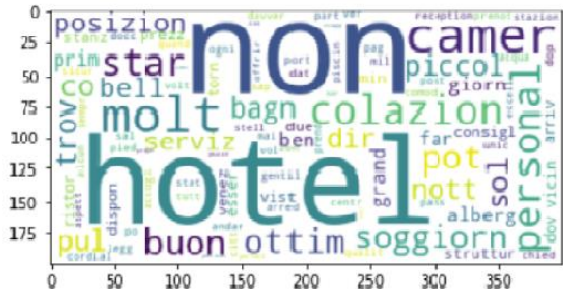

Figure 1.7 *Wordcloud of unlabelled reviews*

2. Preprocessing

In this phase I firstly removed the word *'non'* from the stopwords. Then, taking into account both datasets, I tokenized all the reviews into words, I removed all the numbers, punctuation characters and all the symbols with the exception of alphabetic letters. I converted all the remaining letters into lower-case and I performed a lemmatization and a subsequent stemming of all the words. In particular I used the *ItalianStemmer* from *nltk.stem.snowball* and the *spaCy* library[4].

After that I created a set of stemmas, from the *5000* most common stemmas in the development reviews, which were the most meaningful ones (e.g. *'hotel'* is widespread among the reviews but isn't meaningful for our purpose). In particular with meaningful I mean a stemma such that at least the *75%* of reviews in which it appears are labelled positively (or negatively). Then I've added only the stemma *'non'*, which didn't result as

a meaningful stemma, to this set and I filtered all the reviews in order to have only stemmas in them belonging to this set. That is because in my opinion the word *'non'* shouldn't be considered as a stopword and should be included in the set of final filtered stemmas: its presence in a sentence can overturns its meaning (e.g. *'consiglio l'hotel'*, *'non consiglio l'hotel'*). The wordcloud resulting from the development set processed until now is represented in Figure 2.1.

Afterwards I implemented a method in order to transform every review (currently represented as a text of stemmas) in a sequence of unigrams and bigrams in a customized way: if a word is preceded by a *'non'* it won't appear as a unigram but only in a bigram with *'non'* because in that sentence the word by itself would be misleading. Besides every *'non'* won't constitute a unigram by itself.

Moreover, as a consequence of the analyses previously conducted, I decided to map all the occurrences of emojis/emoticons in the set of positive ones I created with the same unigram and I did analogously for the negative ones. The method also maps the character *'?'* with a unigram (so that *'??'* is mapped into two of this unigrams) while the character *'!'* is mapped into another unigram only if it appears without another *'!'* attached. These additional unigrams are added analyzing the original text of reviews since in the final filtered stemmas the punctuation has already been removed.

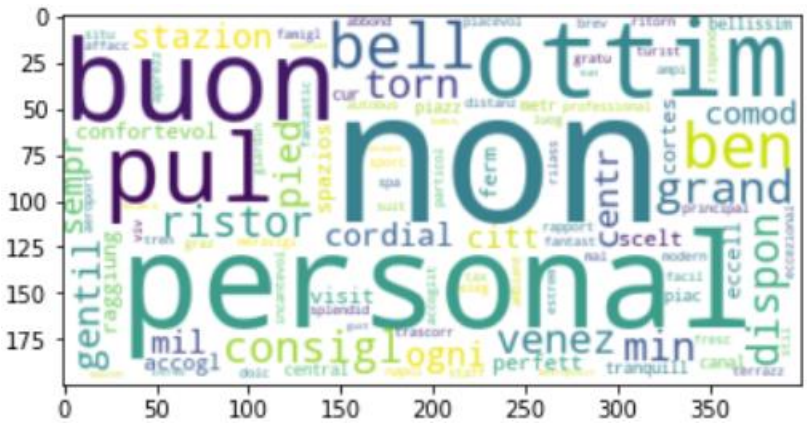

Figure 2.1 *Resulting wordcloud*

3. Algorithm choice

I used the *TfidVectorizer* from *sklearn.feature_extraction.text* to exract the *tf-idf* values of the previously generated n-grams obtaining the feature matrix for my classifier (the vectorizer was fit on the development reviews). In particular I didn't use neither the bag of words representation because there's no guarantee that the most common stemmas are also the most meaningful ones nor the binary weighting schema because doing this way I would have given the same importance to all the occurrences of the stemmas in

the reviews. I didn't even use the *tf-df* measure since it is typically used to summarize homogeneous documents.

As far as the classifier is concerned in the end I decided to use a *linearSVM*. I also tried different algorithms but I always obtained worse results in the validation phase with respect to the ones got with the usage of *SVM*. Infact I think that decision trees aren't suitable in this context since they make splits on a feature at a time and in this context I have many features (all the unigrams and bigrams) and moreover they aren't independent from each other (e.g. if in a review I find a negative word, then it's more probable to find others). That makes also *Naïve Bayesian* and *K-Nearest-Neighbor* classifiers inadequate because one is based on the assumption that features are statistically independent and the other performs badly with high dimensional data (curse of dimensionality). Even random forest led me to worse results. Additionally I also implemented a rule based classifier computing the average polarity of the words in a review and classifying it basing on that measure but the performances were still worse. (these further attempts are reported at the end of the notebook file).

I add that *SVM* implemented with the *LinearSVC* model from *sklearn.svm* runs fast so I found it useless to perform a *PCA* through *SVD*: I tried it and the results were even less accurate. Furthermore I think that the speed of the classification is due to the double filtering, previously described in the preprocessing phase, applied first among most common stemmas (without stopwords) and then on the most meaningful ones.

Further improvements could be made by adding *POS* tagging (e.g. using *spaCy*) at the n-grams in order to better understand the meaning and the polarity of a word since it also depends on the different contexts in which it may be.

4. Tuning and validation

I made two kinds of tuning: one relating the preprocessing and feature extraction phase and one on the *SVM* parameters.
The first one involved the decision to perform a lemmatization or a stemming, the decision of considering just unigrams and bigrams or even trigrams and the choice of the following constants: number of most common stemmas from which I would have found the most meaningful ones, the thresholds in percentage basing on which I would have decided whether a stemma was going to be meaningful or not for my purpose and some parameters in the *TfidfVectorizer* object (*max_df, min_df, max_features, norm*).

In the end, after many trials, I decided to perform even stemming, to consider n-grams with *n=2* and to choose from the *5000* most common stemmas the most meaningful ones (with thresholds of percentage previously mentioned in the preprocessing phase). As far as the *Vectorizer* is concerned I finally set *max_df=0.9, min_df=5, max_features=None* and *norm='l2'*.

I'd like to point out that the reason because I set *max_features=None* is the same for which I decided not to perform *SVD*.

With regard to the classifier, the parameters I tuned were: *tol, C, fit_intercept* and *max_iter*; while in all the trials I fixed the *random_state=1*.
In particular I split the labelled reviews in *training_validation* set and *test* set in a stratified fashion according to the class labels and then for each combination of parameters (which I explored using the *ParameterGrid* from *sklearn.model_selection*) I computed the average accuracy of the classifiers trained on *5* different subsets of 4 folds with those parameters set and validated using the last fold. Specifically I performed a K-Fold Cross-Validation implemented with the *StratifiedKFold* from *sklearn.model_selection*.
The research of the best configuration of parameters for the model was conducted in this way to avoid overfitting on subsets of the labelled reviews at each iteration.

After this process the best configuration of parameters turned out to be the one with *C=0.5, fit_intercept=False, max_iter=1000, tol=0.01*.
I make notice that in the submission platform I loaded two predictions: one resulting from the choice of these parameters and another choosing their default values as indicated in the documentation [5].

As final mark I'd add that another important issue related to sentiment analysis is sarcasm and irony detection and I think one way to try to deal with it could be providing the model with other specific training sets.

5. References

[1] Female proper names. URL:
https://gist.githubusercontent.com/enryold/0676352e48788a24fe792e66b7e99e3c/raw/a182501d03bf1b52001706f932a72606a443c50a/Italian%2520woman%2520names

[2] Male proper names. URL:
https://gist.githubusercontent.com/marciuz/5874431/raw/65d9dfaadcb8d68aecf8ff48f0e2683b06600737/first_name_ITA_M.txt

[3] Italian Stopwords. URL: https://raw.githubusercontent.com/Alir3z4/stop-words/master/italian.txt

[4] SpaCy. A free open-source library for Natural Language Processing

[5] LinearSVC documentation. URL: https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html