Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s276966

Exam session: Winter 2020

1 Data exploration (max. 400 words)

The proposed dataset contains italian reviews from `tripadvisor.it` Italian website.
The whole dataset is split into development set (with ground-truth labels) and evaluation set (unlabeled). During the whole pipeline (data exploration included) only the development set is taken into account. This stands for two main reasons. First, being the evaluation set unlabeled, it is hard to extract useful information regarding our sentiment analysis task. Second, to better generalize the KDD process we should not focus on this specific evaluation set since any other set could be presented in other instances.

It is then enough to analyze the development set.
The cardinality of the dataset is of the order of 10^4, in particular we have 28754 labeled samples.

In Figure 1 it is evident that the dataset is quite unbalanced towards positive reviews. Assuming that the distribution of the evaluation set is similar (*i.e.* the given dataset is a fair set of samples) we should not consider it as biased and therefore we should not re-balance the dataset during the preprocessing phase.

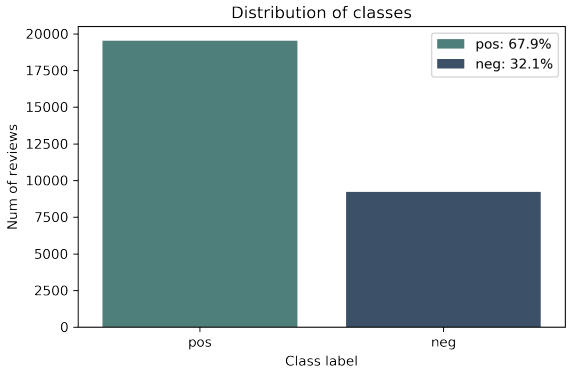

Figure 1: Bar-plot showing the distribution of the classes in the development set

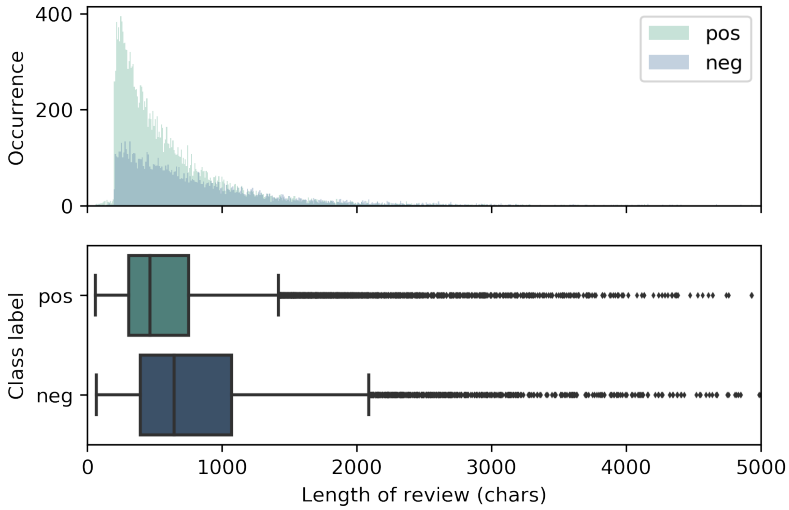

Figure 2: Top: compared histograms showing the distribution for the text length. Bottom: boxplots showing the percentiles also for the text length

That being said, unbalanced data can be a problem for many machine learning algorithms (*i.e.* decision trees) but the imbalance in our case is not really significant (around 2 positive reviews for each negative one) and therefore we should not care too much about it.

Among the two classes, it could also be interesting to analyze some immediate features of the text, for instance the review length.

As we can see from the visuals in Figure 2, text length is more or less equally distributed among the two review classes. This means that we can not exploit this text feature to better carachterize the two classes. The task is therefore accomplished mainly by a proper natural language processing (NLP) step.

2 Preprocessing (max. 400 words)

The first pre-processing step that must be done is *text cleaning*, *i.e.* selecting the most relevant words (or more in general *tokens*) to feed the machine learning algorithm with. The main issue raised by this step is the language of the text in the dataset: the Italian language is naturally not studied as deeply as the English one in text analysis literature.

Many ways to extract relevant tokens have been tested, below it is presented the most successful one.

Text has been tokenized and POS-tagged with the tool *TreeTagger* [1] which is available in many languages. For our purpose, it also comes handy *treetaggerwrapper* library [1] which handles TreeTagger data in a Python environment.

```
Tag(word='Il', pos='DET:def', lemma='il'),
Tag(word='mattino', pos='NOM', lemma='mattino'),
Tag(word='ha', pos='VER:pres', lemma='avere'),
Tag(word="l'", pos='DET:def', lemma='il'),
Tag(word='oro', pos='NOM', lemma='oro'),
Tag(word='in', pos='PRE', lemma='in'),
Tag(word='bocca', pos='NOM', lemma='bocca'),
Tag(word='.', pos='SENT', lemma='.')
```

Figure 3: Example of TreeTagger italian tokens tagging in the sentence *"Il mattino ha l'oro in bocca."*

As shown in Figure 3 this tagger recognises very well part-of-speech tags, as well as the base forms of the words (*lemmas*), contrarily to other Italian lemmatization tools like Spacy's Italian model[2].
Other approaches like stemmatization with NLTK SnowballStemmer [3] have been evaluated but did not provide better quality results for this task.

Since text data is notoriously one of the most unstructured type of data, along with the lemmatization step, also a filtering step is needed so as to clean the reviews from stopwords, punctuation and unwanted text content (like typos, symbols, urls, numbers etc.). This step has being carried out with the creation of a small pipeline which:

- keeps only alpha characters
- discards punctuation, symbols, conjunctions, articles, prepositions, foreign words and all other POS that are not useful for sentiment analysis
- discards common verbs like *'andare', 'essere', 'avere', 'fare', etc.*

The result is then something like the one in Figure 4

[1] `https://perso.limsi.fr/pointal/doku.php?id=dev:treetaggerwrapper`
[2] `https://spacy.io/models/it`
[3] `https://kite.com/python/docs/nltk.SnowballStemmer`

```
"Il mattino ha l'oro in bocca." : ['mattino', 'oro', 'bocca']
```

Figure 4: Example of tokenization

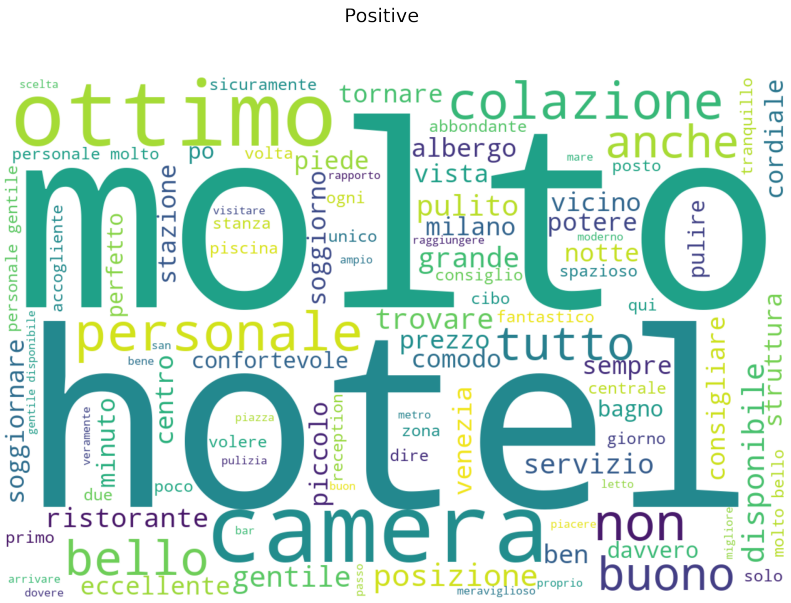

Figure 5: Positive word-cloud with TF-IDF statistic

Moreover the tokens of the whole dataset is processed to actually extract text features. The most common text feature extraction approaches make use of the token counts or the *TF-IDF* statistics. In our case TF-IDF statistic is chosen because, thanks to the Inverse Document Frequency property, it gives lower importance to useless words like *'hotel', 'albergo', etc.*.

The outcome of the TF-IDF preprocessing step is then briefly analyzed generating two word-clouds showing the most frequent words (in terms of TF-IDF frequency) among the two review classes (see Figure 5 and Figure 6)

As we can see in the figures, many words have already been recognized as carachterizing. The 'positive' one in particular shows many clearly positive words such as 'fantastico', 'bello', 'gentile'. However, since these wordclouds have been obtained by summing up for each word all the statistic values among the samples, very frequent words such as 'hotel' appear even if each single value is small (due to IDF factor as mentioned before). Therefore these figures may seem misleading and must be considered just as a coarse separation between the two classes.

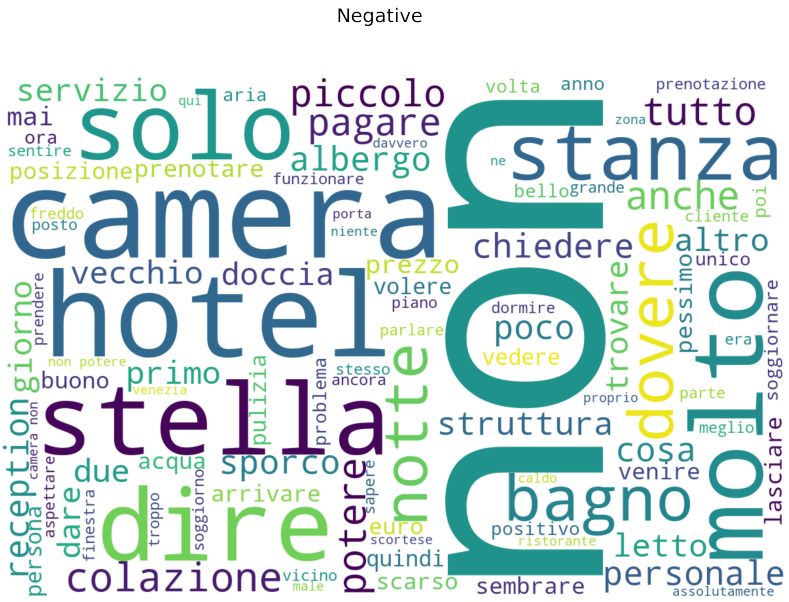

Figure 6: Negative word-cloud with TF-IDF statistic

3 Algorithm choice (max. 400 words)

In this section, two of the most common machine learning algorithm for text sentiment analysis *Multinomial Naive Bayes* and *Support Vector Machine* (SVM), are compared, since they both present advantages and disadvantages with respect to each other.

Naive Bayes is a popular scalable classification algorithm which makes use of Bayes Theorem for computing posterior probabilities from priors. Considering the kind of text data we are working on, Naive Bayes' efficiency becomes convenient: hotel reviews are generally big streams of data (perhaps not as much as e-mails or tweets, but anyway very common) and Naive Bayes algorithm plays straightforwardly for updating the posterior probabilities, not suffering from high samples cardinality.

However, SVM classifiers are commonly known to leave Naive Bayes classifiers behind in terms of accuracy of prediction. On the other hand SVM is not as efficient as Naive Bayes.

To summarize, Naive Bayes algorithm manages to reach more than acceptable F1-score on our dataset, also being fast and updatable with new data: the already trained model could be fed with additional labeled reviews to acquire more precise prior probabilities. This last property is not explicitly required for the assigned task. Nevertheless it might be significant since we are dealing with hotel reviews which consist in high-velocity stream of data. Moreover it boasts high interpretability since the label is assigned according to simple probability measures directly related to the words in each review.

Instead, SVM is slightly more accurate in predicting labels (approx. 0.004 more in F1-score) but its efficiency and scalability are in no way close to NB's. In particular, SVM with non-linear kernel suffers a lot from big sample size data. Its training complexity is, in fact, $O(dn^2)$ where d is the fea-

tures dimension and n the sample size, while the prediction complexity is $O(dn)$. It's easy to see that in our case ($n \approx 10^4$), given time-restricted conditions, non-linear SVM is not a desirable approach.

This being said, the competition only focuses on F1-score of the predicted labels for a given evaluation set, without time constraints, therefore the next section will address the steps that has been carried out for the model hyper-parameters tuning and validation.

In Table 1 the two different classifiers (one in two different versions) are characterized by the F1 score obtained with optimal hyper-parameters (found with grid search strategy) on a 5-fold cross validation.

Classifier	**F1 score**
MultinomialNB	0.962
LinearSVC	0.965
SVC w/ RBF ker	0.966

Table 1: F1 weighted scores with three different classifiers

4 Tuning and validation (max. 400 words)

As a first step, *grid-search* strategy combined with *5-fold cross validation* has been adopted for deciding which combination of hyper parameters best suits our data.

The following are the hyper-parameters found for the two classifiers:

- **MultinomialNB**: `alpha=0.2`
- **SVC**: `C=10, gamma=1, kernel='rbf'`

Regarding Multinomial Naive Bayes, here the only parameter is α. According to MultinomialNB Scikit-Learn documentation [4], a value of $\alpha < 1$ results in a setting called Lidstone smoothing.

For what concerns the `C` parameter of the SVC, it trades off correct classification of training examples against maximization of the decision function's margin. For larger values of `C`, a smaller margin will be accepted if the decision function is better at classifying all training points correctly. In our case `C = 10` is not very high so that the model will not overfit the data.

Also quoting Scikit-Learn documentation[5]: "...the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'"

In fact, the RBF kernel of two samples $\mathbf{x}$ and $\mathbf{x}'$ is defined as follows:

$$K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$$

and in particular it can be easily identified as a similarity measure since it takes values between $(0, 1)$ [6].

To validate the models, the ROC curve can be plotted (see Figure 7).

As we can see, the classifier overall accuracy is so high that the ROC curve is hardly visible. Therefore it may be worth considering just the confusion matrix instead.

		Predicted	
		'neg'	'pos'
Actual	'neg'	2601	203
	'pos'	132	5691

Table 2: Confusion matrix for the MultinomialNB classifier

		Predicted	
		'neg'	'pos'
Actual	'neg'	2631	173
	'pos'	114	5709

Table 3: Confusion matrix for the SVC classifier

[4] `https://scikit-learn.org/stable/modules/naive_bayes.html`
[5] `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`
[6] `https://en.wikipedia.org/wiki/Radial_basis_function_kernel`

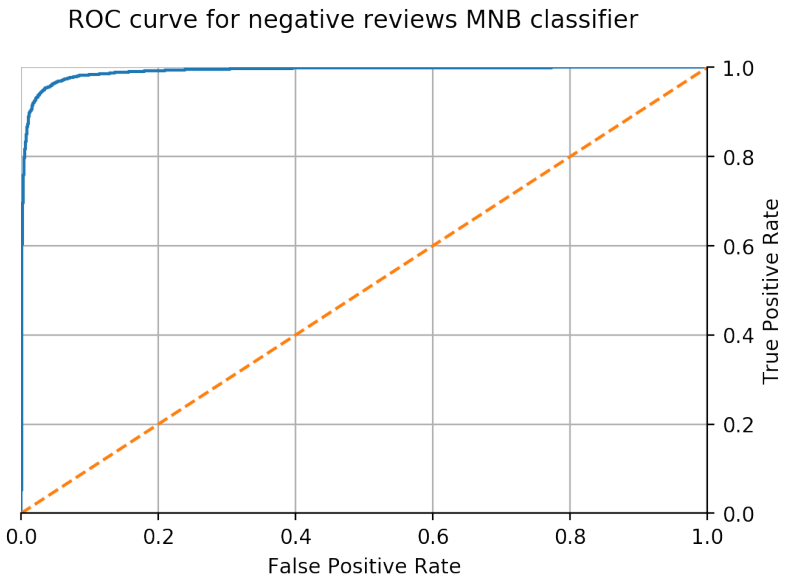

Figure 7: ROC curve of the 'neg' class with MultinomialNB classifier

From Table 2 and Table 3 it's evident that with both classifiers the ratio between false negatives and false positives is more or less the same, *i.e.* $\approx \frac{1}{2}$. As we could expect, this imbalance directly reflects the original imbalance of the dataset.

References

[1] Helmut Schmid. Treetagger. https://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/.