Data Science Lab: Process and methods

Politecnico di Torino

Project report
Student ID: s277971

Exam session: Winter 2020

1. Data exploration

The data exploration step of my project started with the print of the head of the pandas dataframe I used to store the development dataset, in order to get used to the format, topic and the language vocabulary in use.
I then divided the dataset in positive and negative reviews and proceed calculating the number of reviews per partition to check the balance of the development dataset, the length of the reviews, the mean length and the standard deviation (figure1).

```
Data Exploration Phase --------------------------------
Number of positive reviews: 19532
Number of negative reviews: 9222
Mean length positive reviews: 624.5674278107721
Mean length negative reviews: 864.228475384949
Standard deviation length positive reviews: 513.6230925334596
Standard deviation length negative reviews: 740.8239343333894
```

This result seemed promising to be used as additional features during the training of the classifier but unfortunately it led to worse accuracy scores.

I then checked for Null and NaN values in the dataframe (which were not present) and proceed in building two basic worclouds from a dataframe where all reviews were trasformed to lower case while punctuation and stopwords removed.

In order to remove the stopwords I decided to compare the words of the reviews to a list of italian stopwords downloaded from Github which I found more complete than the one provided by NLTK package (download link in references).
From this list I then decided to remove all the words which I felt having potential value for the task of the analysis, that are common italian stopwords but with a clear positive or negative meaning such as (bene, buono, poco, purtroppo..) leaving inside the list only the neutral-meaning words.

POSITIVE

NEGATIVE

From the two wordclouds I was able to add to the list of stopwords the ones frequent in both the classes (hotel, camera, camere, struttura...).

On a final note for what concerns data exploration I examined all the reviews of the test set my classifier was wrongly labelling so I was able to confirm many of them were written in multiple languages (most common languages: english, french, spanish, german), a few of them entirely written in a foreign language while a few misclassified.

```
"Venezia è sempre meravigliosa,in tutte le stagioni, ma il Carnevale le da un'aria particolare.
Ogni calle è invasa da maschere ,si nota l'allegria settecentesca che trasuda da ogni vicolo e da ogno veneziano.
E' un esperoienza da fare almeno una volta nemma vita",neg

"Salve a tutti.
Il palazzo delle stelline è in un'ottima posizione, facile da raggiungere in qualsiasi modo.
Colazione davvero molto abbondante e ottima pulizia.
Unica pecca sono le stanze, che sono un pò scarne.
Molto suggestivo il palazzo.",neg
```

2. Preprocessing

I decided to apply a tf-idf function (even if initially I opted for a tf-df since all the documents were about the same topic but did not obtain better accuracy score) by mean of a Tfidfvectorizer which accepted as a parameter a custom tokenizer where I focused the main text tranformation actions.

One of the recurring aspects of this analysis is that the big part of the dataset is in italian language, language that is far from having the support that english has. This and the fact of having foreign language reviews or many reviews with more than one language led me to the will of translating the reviews in a single language (ideally english to be able to use efficient lemmatizers and stemmers).

In order to achieve that I used the stopwords from the other four main languages to detect what language was present in the review, and Google Translate API to translate. Unfortunately Google blocks API requests from a certain IP over a certain amount so the only possible way was to translate every language but english, thing that did not lead to better results.. Unlucky try but it has been fun learning to use Google API from PyCharm.

```
from googletrans import Translator
translator = Translator()
review_list = review.split(' ')
if custom_at_least_n(stopwords_de, review_list, 3):
    translated = translator.translate(review, src='de', dest='it')
    print('Traduzione dal tedesco effettuata')
    review = translated.text
if custom_at_least_n(stopwords_es, review_list, 3):
    translated = translator.translate(review, src='ca', dest='it')
    print('Traduzione dallo spagnolo effettuata')
    review = translated.text
if custom_at_least_n(stopwords_fr, review_list, 3):
    translated = translator.translate(review, src='fr', dest='it')
    print('Traduzione dal francese effettuata')
    review = translated.text
```

So I first cleaned the reviews replacing punctuation with blanck spaces, then removed numbers and special chars by mean of regular expressions.
At this point I wanted to lemmatize the text and found Spacy Lemmatizer worked best with Italian ("it_core_news_sm").
I finally tokenized the reviews and used a stemmer (SnowballStemmer) on the tokens not in the stopword list (removing stopwords) and with length greater or equal to two (removing small words).

I finally took in consideration many parameters of the TfidfVectorizer many of which resulted in worse results or made no differences except for a very important one which is ngram_range. This parameter allows TfidfVectorizer to keep track not only of the single words but to conceive the n-gram features of the text, this way the tfidf was able to encode the difference between “bene” and “non” + “bene”.

(During this step I also tried to add features to the training dataset such as length of the review, number of words among lists of positive words or negative words, difference between the number of positive and negative words in the review.
No result improvement was noted.)

3. Algorithm choice

Given the task of the analysis and the fact that in this case accuracy and performance were much more taken into account with respect to interpretability, therefore less interpretable but more efficient algorithms have been prefered.

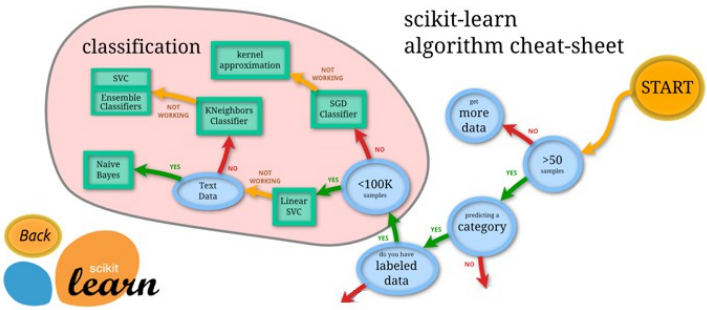
I started my very first classification attempt with the easiest classifier possible, a multinomial Naive Bayes classifier, because of its speed (it just consist of probability counts and if the Naive Bayes conditional independence assumption actually holds then the classifier converges very quickly. Even if the assumption doesn't hold this classifier is known to achieve good results) and its simplicity (not many

hyperparameters to work with for a quick “plug & play” and very low cpu and memory resource usage), properties that made it a perfect starting point.

I then implemented a much more computationally intensive Random Forest classifier to see how it would have performed since this algorithm is very good in identifying the most significant variables from many input variables and it's also highly scalable to any number of dimensions and has generally quite acceptable performance results.
Unfortunately it turned out to be too slow in the training phase due to the magnitude of the dataset and was not performing particularly better than the previously used algorithm.

I briefly tried the classification task by mean of a multilayer perceptron (MLP) which is a feeding forward neural network, class that is known often reaching high accuracy scores for extremely complex problems but also in this case memory and time resources usage did not justify its performance.

Finally, also thanks to scikit-learn's algorithm cheat-sheet, I implemented a support vector machine SVC (not linear from start) which demostrated to be the best in performace (both for speed, in the variant of LinearSVC, and for accuracy).



Support vector machines in fact, when the data is exactly divided in two classes, provide high accuracy, nice theoretical guarantees regarding overfitting, and with an appropriate kernel they can work well even if the data isn’t linearly separable in the base feature space (which is not our case).
This properties make SVC very useful for our text classification problems, where very high-dimensional spaces are the norm, as well as the best performing algorithm.

4. Tuning and validation

Given the magnitude of the dataset and the relatively low variance of data I opted for a less computationally expensive way of validating the model choosing the hold out validation method with a test size of 25% of the dataset, rather than a more expensive K-Fold cross validation.
As for the hyperparameters tuning I implemented a ParameterGrid in order to test all the possible hyperparameters configurations of my SVC classifier.

```
hyp_parameters = {
    "random_state": [0],
    "C": [1, 10, 100, 1000],
    "kernel": ['linear', 'poly', 'rbf'],
    "gamma": [0.1, 1, 10, 100],
    "degree": [2],
    "class_weight": [None, 'balanced', {0: 1, 1: 2}],
    "max_iter": [5000]
}
```

Finally the best configuration turned out to be the one with linear kernel, due to the linearly separability of data in the feature space (therefore degree and gamma were ignored since they work respectively on 'poly' kernel and 'poly' or 'rbf'), and 'C' regularization parameter equal to 10.
What 'C' does in fact is telling SVC how much to avoid misclassification working on smaller or larger margins of the hyperplane.
The parameter 'class_weight' did not provide improvement in either accuracy nor performance.

Since the only relevant hyperparameters were (kernel=linear, C=10) I finally opted for the much faster LinearSVC (with C=10), training it on the complete development set before using it on the evaluation one in order to submit the solution.

5. References

https://github.com/stopwords-iso/stopwords-it/blob/master/stopwords-it.txt
https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html