Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s269991

Exam session: Winter 2020

1 Data exploration

The dataset consists of reviews written in the Italian language about hotels on tripadvisor.it. The aim of the project is sentiment analysis, which means establish if a review is positive or negative, so this is a classification task. We are given two datasets in csv format, one labelled for training the model, *development*, and one without labels, *evaluation*. Let's start exploring the data.

The development dataset has 28754 unique and non-empty entries and contains two columns: text and label, both are nominal attributes. The text column contains the review itself and label represent if the reviews is positive or not, the values reported are *pos* and *neg*. The evaluation dataset has 12323 entries, which have to be classified. Among all the labelled reviews, we have 9222 negative and 19532 positive, with an approximately ratio of 1 to 2.

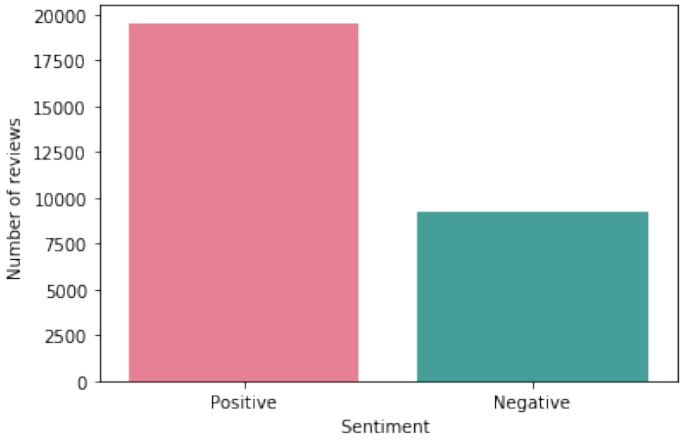

Figure 1: Data distribution

In order to understand how to classify the reviews, it's suggested to try to extract some information without the help of any algorithm. In this case I'm analysing text data, so I can use a word

cloud, which is a visualization technique in which the size of each word indicates its frequency or importance.

Figure 2: Word cloud

Note that there are words related to hotel like: *personale, stanza, colazione, servizio, posizione* but I don't actually see words related to good or bad opinions. There are words like articles and adverbs which don't add a positive or negative "weight" to the sentence. These words are called stopwords and their removal is part of the preprocessing part.

2 Preprocessing

The preprocessing is used to clean the raw data, in order to obtain a suitable format for the classification algorithm. I've tried so many techiques to clean the sentences in order to extract just the essential part from each review. One of the preprocessing pipelines I implemented was the following: tokenization, stop words removal, stemming, TfidfVectorizer.

I performed tokenization splitting each sentence into words, removing punctution, special characters and numbers, then i converted all characters to lower case. To remove stop words I just iterate over all the words and at the end I did stemming, which is reducing words to their stem form (i.e. *bellissimo* becomes *bell*).

Now that I have cleared all the reviews, I create a new column in the dataframe and I apply the TfidfVectorizer. The Tfidf, term frequency inverse document frequency, is a tool to extract feautures from raw textual data and it is computed as it follows

$$TFIDF = freq(t, d)\ log\frac{|D|}{freq(t, D)}$$

where t is the term, D is the collection of documents d and |D| is its cardinality. In other words, the frequent terms present in our collection of document have an higher tfidf.

The best pipeline I used is the following: CountVectorizer and TfidfTransformer. CountVectorizer is a feature extraction tool that counts the occurences of each term in the collection of documents and it builds a matrix used from the TfidfTransformer to calcucate the inverse document frequency. Note that the tokenization is already applied by the CountVectorizer and I didn't remove stopwords, because I used the hyperparameters max_df and min_df, which remove terms with a document frequency higher than max_df and lower than min_df.

At this point this pipeline has generated thousands of feautures per review, so I used a dimension reduction tool, SelectKbest, in order to pick just the ones more significant.

```
cv = CountVectorizer()
corpus = [ 'Questo hotel è bellissimo', 'Albergo bellissimo',  'Camere pulite']
test = cv.fit_transform(corpus)
print(cv.get_feature_names())
print(test.toarray())
```

```
['albergo', 'bellissimo', 'camere', 'hotel', 'pulite', 'questo']
[[0 1 0 1 0 1]
 [1 1 0 0 0 0]
 [0 0 1 0 1 0]]
```

Figure 3: CountVectorizer example

3 Algorithm choice

Given a general 2D classification task, we have to find a way to divide the points in two or more groups. The immediate approach would be to draw a straight line, but that line is not unique. The solution to this problem is to use Support Vector Machines. I choose one of the Support Vector Machine algorithms, LinearSVC, similar to SVC but it is suggested for large scale problems. Support Vector Machine chooses a line which maximizes the margin between the groups and the points located in the boundaries of it are called support vectors. The wideness of the margin can be chosen changing the hyperparameter C in the model, if C is small we have soft margin, otherwise hard margin. Note that choosing a right C is crucial also for outliers, which have to be avoided.

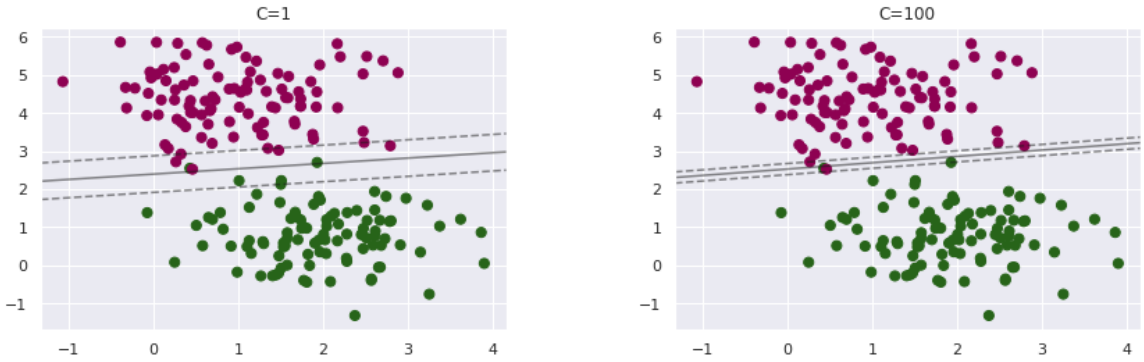

Figure 4: Soft margin and hard margin

I've tried other classification algorithms such as DecisionTreeClassifier, RandomForestClassifier, SGDClassifier, GradientBoostingClassifier but the accuracy is smaller compared to the LinearSVC one, about 85

4 Tuning and validation

Once the project workflow is completed, I can try to tune the model in order to maximize his performance. Before starting, I split the development dataset in train and test, so I can verify my f1 score before making a submission to the platform. The dataset is a bit unbalanced, the number of positive reviews is two times the negative ones, so I tried other approaches in order to split the dataset, like StratifiedShuffleSplit, which creates splits with equally distributed samples and RandomOverSampler, which over sample the minority class. Both of them didn't work, in particular these techniques are needed when the dataset is highly imbalanced, i.e. ratio of 1 to 100. So I proceeded with a train_test_split, with a test size of 0.25.

Now that I have the data splitted, I can start tuning my model, to make it more quickly I used ParameterGrid.

As I said before, I chose CountVectorizer and TfidfTransformer for the preprocessing part because it provides a better accuracy result. For the CountVectorizer I set:

- max_df= 0.5, it selects only words that are present in less than 50% of the reviews;
- min_df=3, it selects only words that are present in more than 3 reviews;
- ngram_range= (1, 3), it generates tokens with 1, 2 and 3 words;
- token_pattern=r'\b[^ \d \W]+\b', it excludes numbers and punctuation from tokens.

I didn't add any custom hyperparameters to TfidfTransformer, I just left the default ones.

Every time I finished my preprocessing pipeline I checked the number of features generated, in the best case about 280000, I adjusted the hyperparameter k in SelectKbest. After some test, the best performance is obtained when k is about an half of the total number of features, 130000. The score function I've used is Chi-squared.

For LinearSVC I've used C equals to 1.3, which means I set a soft margin.

For comparing my different configurations I've used these metrics:

- confusion matrix, to see the distribution of false positive and false negative;
- classification report, which displays precision, recall, f1 score and support divided per category;
- f1 score weighted, which is the metric computed by the submission platform.

The two solutions chosen has different hyperparameters, the model used is the same.

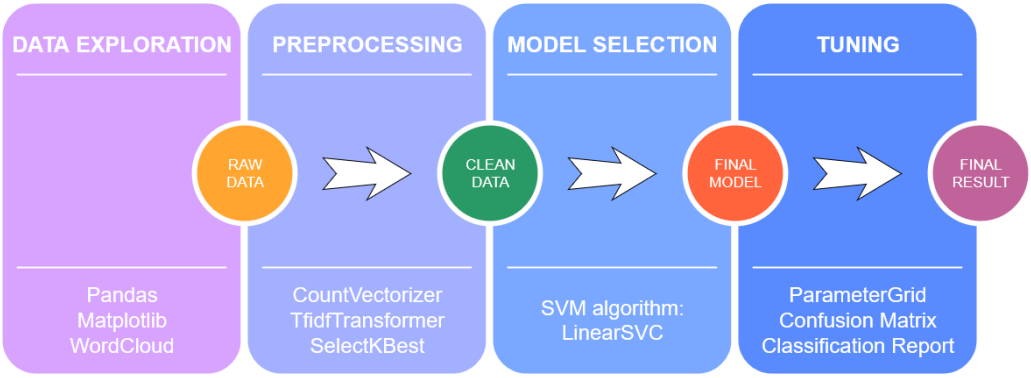

Figure 5: Project workflow