Data Science Lab: Process and methods

Politecnico di Torino

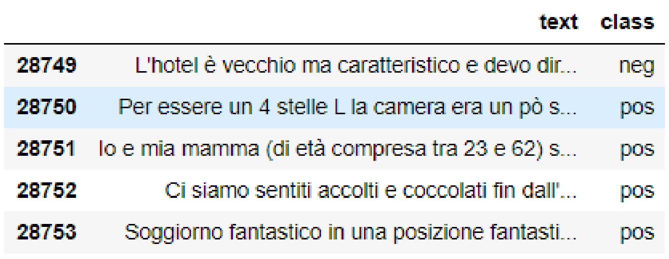
Project report

Student ID: s265529

Exam session: Winter 2020

1. Data exploration

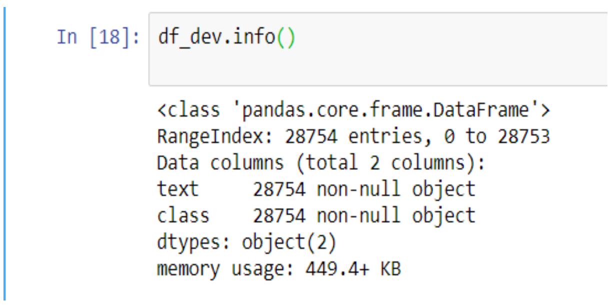
The goal of this project is to analyze user's textual reviews written in the Italian language, to predict the sentiment contained in the text. We have to build a classification model to understand if a reviews is a positive or negative based on the available data of tripadvisor.it italian web site.
The development dataset contains 28754 textual reviews.
We can use pandas' read_csv() method to obtain DataFrames and analyze the content of the file.



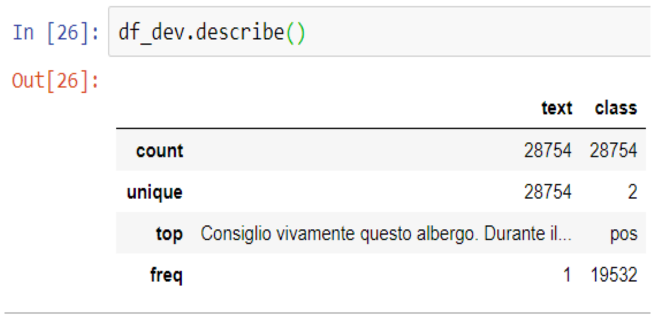
	text	class
28749	L'hotel è vecchio ma caratteristico e devo dir...	neg
28750	Per essere un 4 stelle L la camera era un pò s...	pos
28751	Io e mia mamma (di età compresa tra 23 e 62) s...	pos
28752	Ci siamo sentiti accolti e coccolati fin dall'...	pos
28753	Soggiorno fantastico in una posizione fantasti...	pos

The dataset is provided as textual files with multiple lines.
Each line is composed of two fields: text and class. The text field contains the review written by the user, while the class field contains a label
- *pos: if the review shows a positive sentiment.*
- *neg: if the review shows a negative sentiment.*

Using pandas we can inspect our dataset.



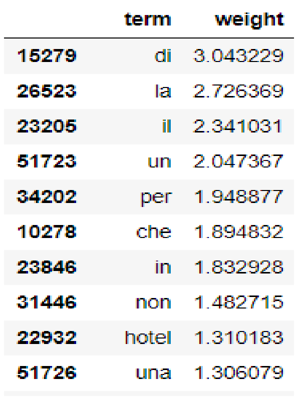
```
In [18]: df_dev.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 28754 entries, 0 to 28753
Data columns (total 2 columns):
text     28754 non-null object
class    28754 non-null object
dtypes: object(2)
memory usage: 449.4+ KB
```



```
In [26]: df_dev.describe()
Out[26]:
```

	text	class
count	28754	28754
unique	28754	2
top	Consiglio vivamente questo albergo. Durante il...	pos
freq	1	19532

As we can see there are no, null/missing value in the dataset. We can also see that class are not balanced because we have 19532 positive and only 9222 negative samples.
We apply now CountVectorizer algorithm to the set of documents implemented by the sklearn library. This allow as to show how much different word are present in the file and which are the most common.
In the files there are 55535 word, without preprocessing data, the most common word in the document are not useful to classification. We need to preprocessing our data in order to remove useless word and reduce the number of "features"



	term	weight
15279	di	3.043229
26523	la	2.726369
23205	il	2.341031
51723	un	2.047367
34202	per	1.948877
10278	che	1.894832
23846	in	1.832928
31446	non	1.482715
22932	hotel	1.310183
51726	una	1.306079

2. Preprocessing

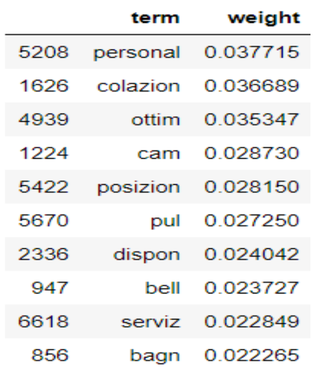
Here we can make use of the TfidfVectorizer class from scikit-learn. LemmaTokenizer class can be used in place of the TfidfVectorizer's standard tokenizer. In our tokenizer, We create tokens through nltk WordPunctTokenizer that Tokenize a text into a sequence of alphabetic and non-alphabetic characters, using the regexp \w+|[^\w\s]+,
this allows us to separate strings that contain symbols inside like the apostrophe so that we can separate articles and words.
Then we need to Stemming the words. This is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem. To perform stemming we had use nltk snowball which supports different languages including the Italian one. We also remove words less than 3 characters long and stopword that indicating the common terms to be ignored and words that contain numbers and special characters. The list of stopwords is taken from nltk library and is extended with *[1]* and other words that we consider not informative in this context
Another thing to do is ignore words that are too common or too rare that usually does not bring much information. We can use **min_fd** and **max_fd** parameters.
max_df is used for removing terms that appear too frequently.

max_df = 0.95 ignore terms that appear in more than 95% of the documents

min_df = 25 ignore terms that appear in less then 25 documents.

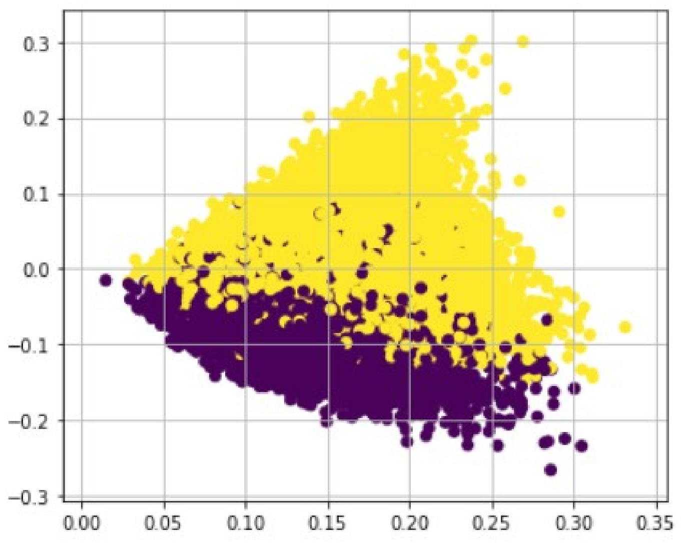
We also include bigrams to try to improve the result

With this step the matrix obtained consists in **6971 columns**, a column for each distinct word present in the dataset that has not been eliminated.
The **top 10 words** based on tfidf score are



	term	weight
5208	personal	0.037715
1626	colazion	0.036689
4939	ottim	0.035347
1224	cam	0.028730
5422	posizion	0.028150
5670	pul	0.027250
2336	dispon	0.024042
947	bell	0.023727
6618	serviz	0.022849
856	bagn	0.022265

We can transform and reduce our input feature space through the Principal Component Analysis (PCA) by means of the Singular Value Decomposition (SVD). Each component extracted by the decomposition will express a given amount of the variance of our data.
In the best case, all the variance is expressed by a low number of new features. As suggested, the TruncatedSVD class let us decide how many components we want to retain in the new space.
We plot the first 2 principal component.



3. Algorithm choice

There are several algorithms that can be used for supervised text classification, we chose to use MNB classifier and SVM and compare the result

MultinomialNB implements the naive Bayes algorithm for multinomially distributed data naive Bayes variants used in text classification when data are represented as word vector counts, or tf-idf vectors
We use sklearn.naive_bayes. ComplementNB that is an adaptation of the standard MNB particularly suited for imbalanced data sets as our dataset. This is a “simple” algorithm so we do not need to perform dimensionality reduction with PCA

SVM draw a hyperplane that divides a space into two subspaces: one subspace that contains vectors that belong to a class and another subspace that contains vectors that do not belong to that class. Those vectors represent the training texts. This means that we are using a subset of training points in the decision function ,called support vectors, so it is memory efficient.
Effective in high dimensional spaces and we can easily use Kernel functions in case data are not linearly separable

Both algorith support sparse sample vectors as input

4. Tuning and validation

To perform training and validation we split the dataset using train_test_split function from the sklearn.model_selection library. We use 80% of data for training set and the other 20% for validation set

For **NB Classifier** we train on the training set and we get the following results:

- **Validation set: 0.948**
- **submission platform: 0.956**

For the **SVM classifier** we perform the grid search with cross validation. The score that we want to optimize is the f1_weighted. The parameters grid is

```
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-2,1e-1,1,10],
                     'C': [0.1,1, 10]},
                    {'kernel': ['linear'], 'C': [0.1,1, 10]}]
```

The get the best result with {'C': 10, 'gamma': 1, 'kernel': 'rbf'}.
If we use 3088 components that retain 80% of the explained variance with this hyperparameters we get **0.961** on our validation set.
Then we train again the model on the whole development dataset and By submitting to the submission platform we get **0.969** that is higher than the result of the naive bayes model

5. References

[1] https://github.com/stopwords-iso/stopwords-it