Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s278221

Exam session: Winter 2020

The aim of the project is to conduct a sentiment analysis task, analyzing user's textual reviews from the *tripadvisor.it* Italian web site, to understand if a comment includes a positive or negative mood.
In practice, the goal is to build a robust classification model that is able to predict the sentiment contained in a text.
The project can be divided into different segments that will be described in the following document.

1. Data exploration

1.1. Load the Dataset

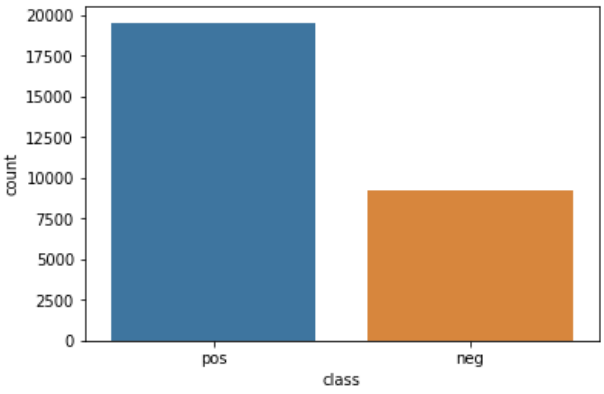
The first step is loading the dataset. The entire dataset is made of 41077 textual reviews written in the Italian language and it is divided into 2 files:
- *development.csv* (which has 2 columns: text, class)
- *evaluation.csv* (which has 1 column: the one corresponding to the text)

As a *.csv file, we load it as a dataframe using the pandas library.

1.2. Data Exploration

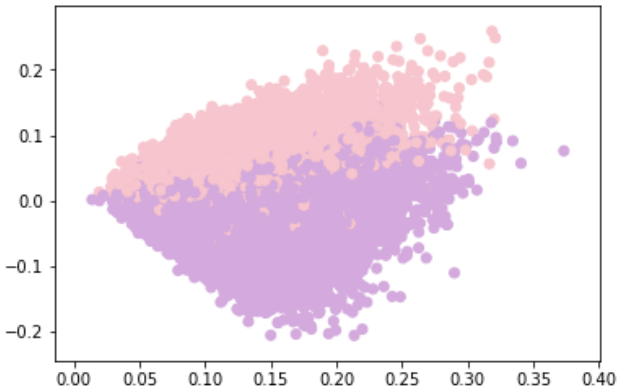
The first thing to do is visualizing the first few records of the dataframe (.head() method) , then checking if there are missing values (.info() method) . However, none were detected at this stage. The development set was then loaded as a dataframe made of 2 columns (text, class) and 28754 rows (each row is a different review).
Then the dataframe was split into 2 parts: one for the "text" column and the other one for the "class" column.

The percentages of positive / negative reviews were : Positive: 67.93 %
Negative: 32.07 %
Printing and plotting the number of positive / negative reviews can offer further evaluation of the data collected.



Blue color corresponds to number of positive reviews (19532) , orange color corresponds to negative reviews (9222). As we can see, the majority class is “positive”.

Creating a scatterplot of the points (in a way that each point represents a review) proved itself to be another useful tool to examine the data.
To do this, each review was represented as a vector: a way to perform it was using a *TfidfVectorizer* in order to transform the single review into a *feature vector*. Since a plotted 2d graph was needed , a *TruncatedSVD(n_components= 2)* was also used to reduce the dimensionality of the sparse matrix obtained by the vectorizer.



Points were plotted in a way that plum color corresponded to positive reviews , pink color corresponded to negative reviews.

Decision boundaries generated by decision trees classifiers are parallel to axes because test condition involves a single attribute at-a-time. This still looks like a linear problem, but they don't work well in this case. So the option of using a *tree classifier* as our classification model was excluded.

2. Preprocessing

TF-IDF

Since data mining algorithms are unable to directly process textual data in their original form, each review has to be represented as a feature vector. One possible way to do this is the *tf-idf technique.*
The tf-idf algorithm is implemented by *TfidfVectorizer()* class in the *sklearn* library. Tokenization, case normalization, stemming and stopword elimination are all performed in there.

In particular, a t*okenizer class* was implemented and passed as a parameter to the *TfidfVectorizer()*. It tokenizes the words in the review, returning a list.
In there, the following operations are performed: for each review, only alphabetical words are considered; every word, considered as a token, is then stemmed with the italian stemmer (provided by nltk library, in particular the snowball italian stemmer was used); a token is taken into account only if his length is more than 1 or less than 16.
To account for the fact that a word preceded by a negative word has opposite sentiments, both *unigram* and *bigram* features were used in this project; the feature vectors thus produced span a very high dimensional feature space and hence is expected to be very sparse.
Bigrams creation is implemented inside the tokenizer class.

Also a *min_df* = 5 was passed as a parameter in the *TfidfVectorizer()*: it means that the algorithm will ignore terms that have a document frequency strictly lower than the given threshold.

Finally, the list of *italian stopwords* (provided by nltk library) was extended with stemmed italian words , removing “non”, “pi”, “contr”, “anche”, “ma”.
An important thing to notice is that stopwords were applied only on unigrams. Stopwords on bigrams were not applied because the italian language has a very complex morphology and they help to better understand the meaning of a sentence (in this case: a sentence made of 2 words).

SelectKBest

As the vectorizer returns a sparse matrix with *k * 10^5* features, dimensionality reduction of the matrix was needed to apply the classification model.
During the first attempts, *TruncatedSVD()* was used to perform feature selection, however, a different and more rapid algorithm was chosen to perform this task.
So *SelectKBest()* was preferred, which is a feature selection algorithm that scores the features using a function (in this case chi2 function was selected because it performed better than f_classif) and then removes all but the k highest scoring features.
Parameter ‘k’ was determined in the GridSearch.

3. Algorithm choice

As mentioned in Data Exploration section, decision tree classifiers do not work well in this case, so the choice of a *random forest classifier* as our classification model was excluded.

During the very first attempts, *KNearestNeighbours(n_neighbours=5) , DecisionTreeClassifier(), SVC(kernel = 'linear'), LinearSVC()* were tried out.
To test each classifier, the entire development dataset was divided into train and test set, with a *test_size = 0.2* .

This split only happened the first days of developing the project: in fact, the code related to this part was commented once the algorithm was chosen.

The resulting f1-scores were:

clf : KNearestNeighbours(n_neighbours=5)	f1-score : 0.553919274715548
clf : DecisionTreeClassifier()	f1-score : 0.858744892903621
clf : LinearSVC(C=1, dual=False)	f1-score : 0.970894298731255
clf : SVC(kernel='linear', C=1)	f1-score : 0.970337359829707

A Support Vector Machine (SVM) algorithm was used for this classification task. SVM algorithms construct a hyperplane that divides the space into regions (in this case: one is for positive reviews and one for negative ones).
In particular, some attempts were done with the SVC(kernel='linear') but it was not time efficient, so the final choice was *LinearSVC()*.
Parameter 'C' was determined during the GridSearch. Parameter "dual = False" was also set.

4. Tuning and validation

The development set is split into train and test dataset. The *training set* is exploited to perform the grid search of the parameters with cross validation in order to build a robust and reliable model. The *test set* is then used to test the best configuration found during the training process.

A *test_size* = 20% of the dataset was chosen and a *random_state* was set to produce replicable results.

Then a Pipeline was created : this means that a new different *TfidfVectorizer(), SelectKBest(), LinearSVC()* will be used in each step of the grid-search cross validation.
Then a dictionary was created, having the names of the hyperparameters for *keys* and the lists corresponding to the values to be assigned for *values.*
C = [0.75, 1, 10]
k = [15000, 30000, 50000]

The grid search was then performed. In particular, *GridSearchCV* performs the exploration of all the possible parameters combinations trying to optimize the scoring function (in this case scoring = '*f1_weighted*', according to the assignment). The parameter cv=5 was also set.
After computing the *fit()* method to the *training set*, the best parameters were shown by printing the best configuration found by the grid search:
[*C = 0.75, k = 30000*].

The detailed classification report was also printed after the *predict()* method had been applied on the *test set,* and the resulting f1-score was 0.97036 (local), that is a reasonable result for our model.

Having found the best parameters configuration, the vectorizer, selectKBest and the classifier were trained again with all the available data in the development set. In this way all the information available were exploited by the model when performing a prediction of new unknown data.