Data Science Lab: Process and methods

Politecnico di Torino

Project report

Student ID: s275090

Exam session: Winter 2020

In this project, I tried to perform a sentiment analysis, analysing user's textual reviews, to understand if a comment includes a positive or negative mood. I built a classification model that is able to predict the sentiment contained in a text.

1. Data exploration

The whole dataset contains 41.077 user's Italian textual reviews that is splitted in two csv: a labelled set with 12.323 records for the development of my model and a collection of 12.323 unlabelled reviews. In a first analysis, the dataset not contains empty values.

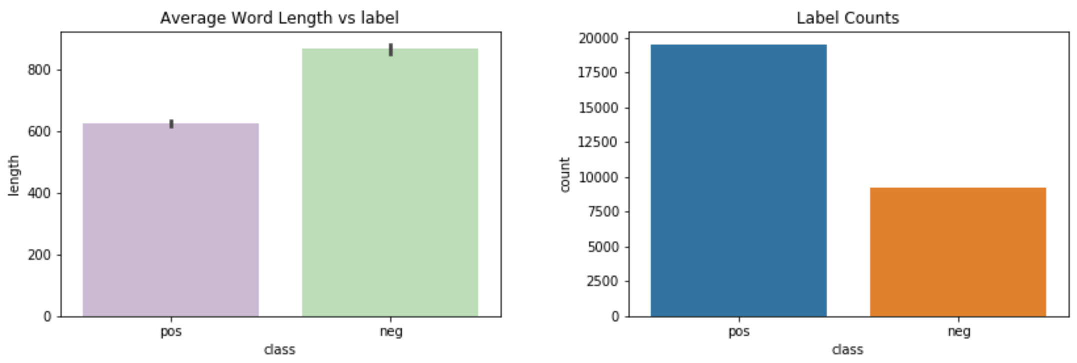


Figure 1 - Plots data exploration

The above two graphs tell us that the given data is an imbalanced one with less than half amount of "neg" labels, but nevertheless the length of the text plays a major role in classification. So, there are more positives (67%) than negatives (33%) and it's important to be careful with interpreting results from imbalanced data.

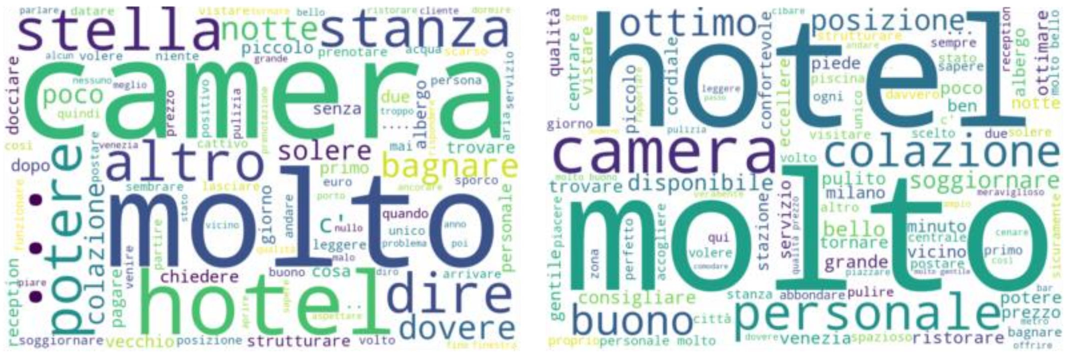


Figure 2 – WordCloud Class neg and pos (development set)

In a first analysis of the text, with a basic tokenizer, I generated a first vocabulary of more than 700k features that I tried to reduce. We can see (fig2) that there are many common and frequently words in the two classes, like as generic words, that not all of them are useful for the sentiment analysis. In addition, in some reviews there are different types of punctuation that can be useful for the purpose of classification (e.g. emoticons, ellipsis etc). There are also many grammar errors. So, I tried to extract the most relevant features with different filter/package.

2. Pre-processing

For pre-processing step, I used a merged version of the two dataset, which I called df. In this way, I can run the code only once on the entire dataset. Clearly, when studying df, I not consider the class column (since that column is only available on df_dev).

After that, in order to train the classifier, I need to filter, normalize and transform the dataset into numbers.

For this step, I have created a class called LemmaStemTokenizer, that tokenize, filter, stem and lemmatize the raw text. For the lemming process, I used the Italian vocabulary (it_core_news_sm) of spaCy library[1].

SpaCy tokenize and lowercase the text and extracts from each word some attributes, such as the base form of the word, the simple part-of-speech tag (verbs, adjective etc...), the word shape etc…

In this class, I filtered the tokens containing numbers, the words that have more of 20 and less 2 characters and the token that contains some punctuations. However, there are some groups of punctuation marks which could be useful for my classification, such as emoticons (e.g. :-(, :-), -.- etc) or the ellipsis[2] (three dots) that, on the internet and in the text messaging, indicating disagreement, disapproval or confusion. So, I tried to include them into my model.

Furthermore, I preferred not to use a precompile list of the most common words of the Italian language, because there are some stopwords they could be useful, like negation adverbs, that could play an important role in the bigram tokens. I used SpaCy, that is able to make a prediction of which part of speech most likely applies in a sentence, to filter the tokens that had a function a conjuction, pronoun, determiner or unknow in the text.

After checking the tokens, spaCy transforms the word into its basic form (lemma) and with SnowballStemmer from nltk, reduce inflection in word to his root form.

Now the textual data must be transformed into a more manageable representation. To do this, I used the method TfidfVectorizer, from sklearn library, that convert the raw dataset, processed by LemmaStemTokenizer class, into a matrix of TF-IDF features.

In this method, I added a little Italian stopwords list where, exploring the vocabulary's vectorizer, I manually entered some words that spaCy was unable to filter but which are very common in any text and worthless. Then I considered unigram and bigrams tokens, I removed accents and performed other character normalization during the pre-processing step.

Lastly, after several tests, I ignored terms that have a document frequency higher than 25% and less of three frequency in my dataset.

[1] https://spacy.io/
[2] https://en.wikipedia.org/wiki/Ellipsis

3. Algorithm Choice

To find the best classifier for my dataset, I selected some algorithms that could be ideal for boundary decision. So, with kfold from sklearn library, I fitted 5 folds of the labelled dataset for each of 8 classifiers. To compare the performance, I used the accuracy and f1 weighted metrics.

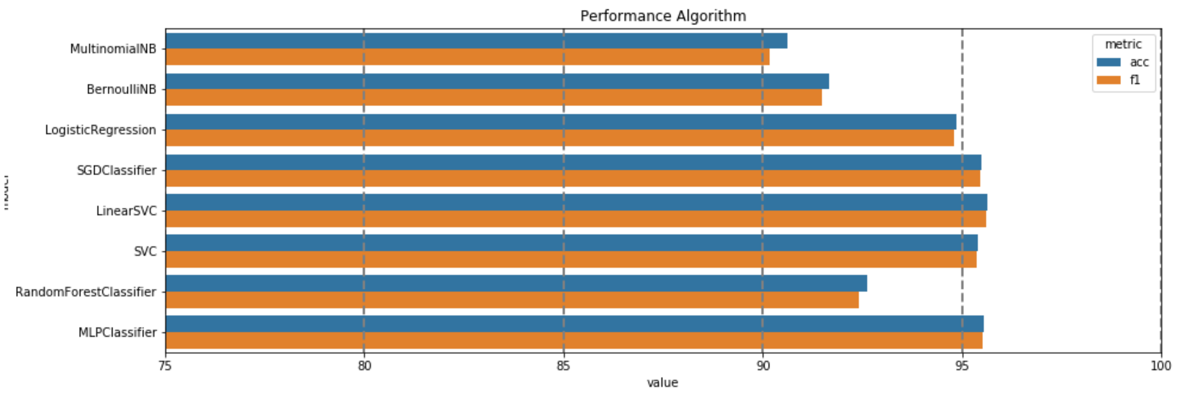


Figure 3 - Performance Algorithms

From the above graph (fig3) we can see that the most powerful classifier is the Linear Support Vector Classification from sklearn library. That's very powerful and especially fast for boundary decision.

Also MLPClassifier, accurate and robust to noise and outliers, has obtained good results, even if slightly lower, but for its long training times I preferred LinearSVC.

To avoid over-fitting, I divided the labelled dataset into training and test split, which gives us a better idea as to how algorithm performed during the testing phase. This way the algorithm is tested on un-seen data, as it would be in a real application.

So, with train_test_split method from sklearn.model_selection, the algorithm splitted the dataset into 80% training and 20% test data, in a stratified fashion, using the class labels (*stratify = y_train*).

Now, with the right format to give the LinearCSV, the algorithm is ready to fit the model with the training set and to do a predict with the test set.

4. Tuning and Validation

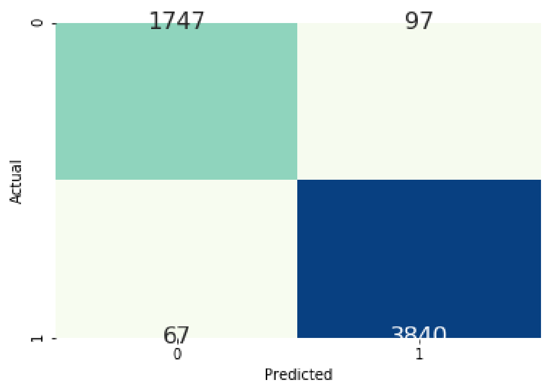


Figure 4 - Test confusion matrix

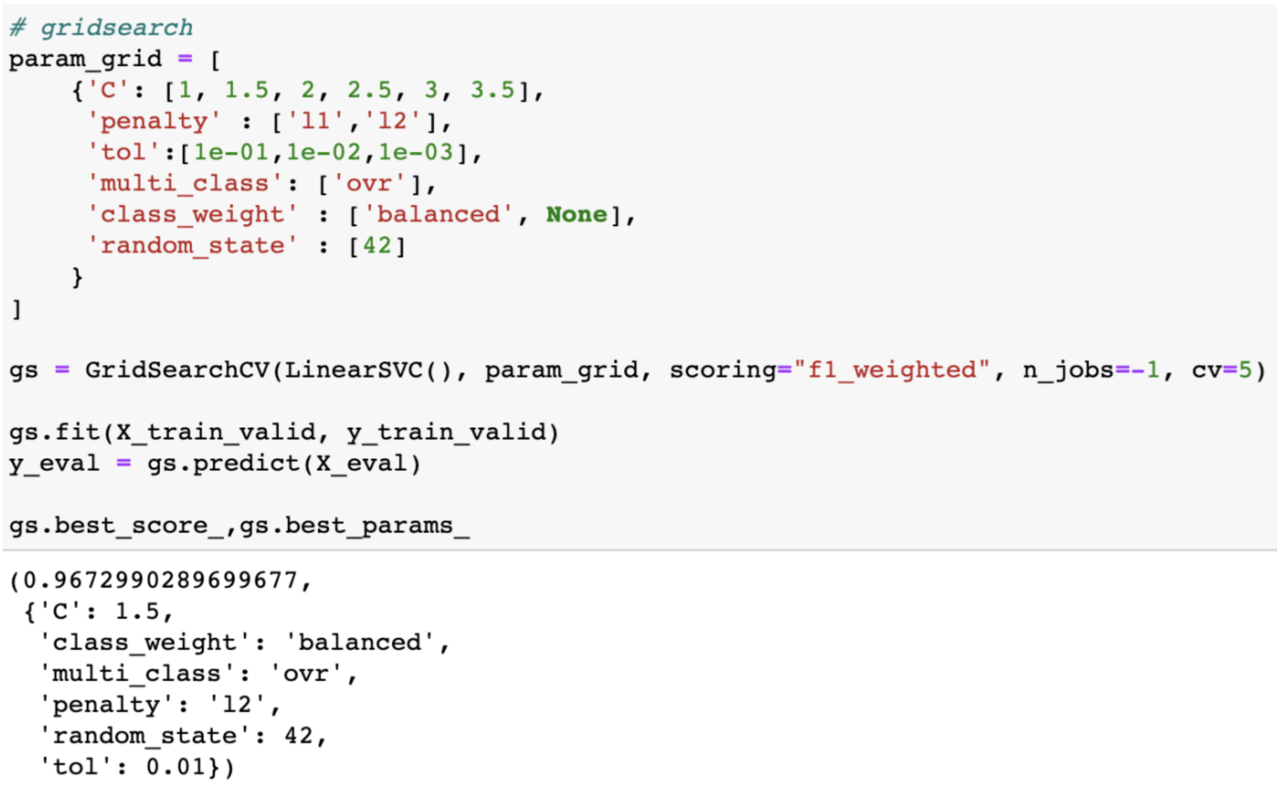
From the label predict on the test, I built the confusion matrix and there is no particular concern about the validity of my classifier.

```
f1_weighted_score = 0.971
              precision    recall  f1-score   support

         neg       0.95      0.96      0.96      1814
         pos       0.98      0.98      0.98      3937

    accuracy                           0.97      5751
   macro avg       0.97      0.97      0.97      5751
weighted avg       0.97      0.97      0.97      5751
```

Table 1 -Classification report

The f1 score tells us that the classifier has no particular problems predicting the two classes. The f1 score for the pos class is slightly higher than the others, same thing for accuracy, but probably is for the imbalance data between the two classes.

To improve my model and define the best hyper-parameters, I utilized cross validation, using the GridSearch object from sklearn library. For the grid search, I used 5-fold cross validation to make sure that I do not overfit a single subset of data.



```python
# gridsearch
param_grid = [
    {'C': [1, 1.5, 2, 2.5, 3, 3.5],
     'penalty' : ['l1','l2'],
     'tol':[1e-01,1e-02,1e-03],
     'multi_class': ['ovr'],
     'class_weight' : ['balanced', None],
     'random_state' : [42]
    }
]

gs = GridSearchCV(LinearSVC(), param_grid, scoring="f1_weighted", n_jobs=-1, cv=5)

gs.fit(X_train_valid, y_train_valid)
y_eval = gs.predict(X_eval)

gs.best_score_,gs.best_params_
```

```
(0.9672990289699677,
 {'C': 1.5,
  'class_weight': 'balanced',
  'multi_class': 'ovr',
  'penalty': 'l2',
  'random_state': 42,
  'tol': 0.01})
```

Figure 5 - GridSearch Parameters

Since GridSearchCV already refits the best model with the entire dataset to predict the class for evaluation set. Then, I can generate a csv file with the predicted values.

By submitting output.csv to the submission platform, I get the result: 0.974