Data Science Lab: Process and methods

Politecnico di Torino

Project report

Student ID: s278106

Exam session: Winter 2020

1. Data exploration

Data exploration is a very important step in our pipeline since allows us to understand the flaws of the data in the group of provided texts. To analyze the content of the corpus of documents I used a CountVectorizer to extract all the words present in it. Taking advantage of the efficient tokenization implemented in the vectorizer I printed on a text file the whole list of tokens.
This file contains a huge number of entries but after a brief analysis of the content of the file we can notice the presence of many words that contain numbers, a high number of misspelled words and of course the presence of stop words.
We are going to tackle all these problems in the data preprocessing part by filtering these tokens.

2. Preprocessing

The data preprocessing that my solution implements different techniques to vectorize the corpus of documents in the dataset and to clean-up the data.
In my implementation I built a class that incorporate the TF-IDF vectorizer and a custom analyzer. In this way we can stem all the words (using the Snowball Stemmer from the NLTK library) and remove all the words that contain numbers. Then we proceed with the tokenization of the text and the removal of stop words. The set of stop-words used in the algorithm is the one provided by NLTK with one exception: the word “non” – Italian for “not”. The reason of this inclusion is that, in my opinion, the negation of a positive feeling can express a very different opinion from what we could understand if we ignored the presence of this noun.
Once we have obtained a set of more meaningful tokens, which represent the set of “good” word that appear in the corpus, we proceed to compute the TF-IDF for our corpus of documents. In the vectorization phase we ignore all the words that occurred less than three times and that appeared in more than the 30% of the documents to remove misspells and non-meaningful words.
Please note that the algorithm considers one-grams and two-grams in the text - i.e. couples of tokens that appear together beside the single word occurrences. This highly increases the number of tokens but also gives us the chance to take into consideration

a various number of terms juxtapositions (e.g. “not satisfied”) that can be much more informative than the two separated tokens.
Since our tokenization gives us with a very high number of features (around 100k), we limit the maximum number of features considered by the vectorizer at 30000, based on their TF-IDF value.
Then as a last step we perform a truncated SVD to reduce the number of features that we want to use to represent our data.
Please note that the feature reduction step (i.e. limiting the maximum number of features and applying the SVD) is not part of the *Solution2.py*, since in that case the whole set of tokens is passed to the MLP classifier that in this case takes care of the feature selection process by setting the weight of each parameter. This is the only difference between the two solutions.

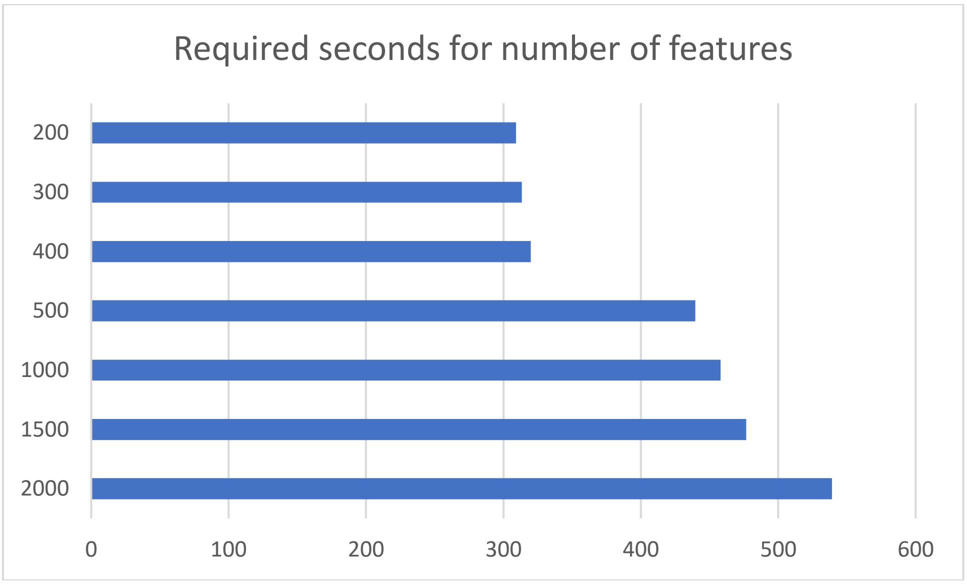
3. Algorithm choice

The classifier that I selected for this task is a Multi-Layer Perceptron neural network. The implementation of this neural network was not done by me since I used the MLP classifier provided in the SKLEARN package.
The main reason that led me to select this classifier is the fact that natural language related problems are not well linearly separable. For this reason, an MLP classifier that can perform non-linear separation of the input parameters represents a solid choice, in addition, the capability of the classifier of addressing what parameters are the most important presents a solution for the feature selection problem.
Two common downsides of using a neural network: the need of a large datasets for the training and the uninterpretable output, were mitigated by the nature of the problem. First, our dataset was quite large so we can properly train the classifier, second, since we are not interested in understanding the reasons behind the classifier’s choices, the uninterpretable output doesn’t represent a problem for our use case.
The choice of an MLP classifier over a Convolutional Neural Network is subsequent to the choice of the TF-IDF technique. Since the TF-IDF vectorization can compute the occurrence of one-grams and two-grams by himself we don’t need to realize convolutional filters to extract information from our data representation. If instead we chose a word2vec embedding, a CNN would have been a great choice to study the collinearity of groups of tokens, hence deriving the notion of a similar meaning.
Finally, the choice of an MLP classifier is performance oriented since the classifier gave more consistent results over the other classifiers I tried, performing well even when the feature space is reduced by means of selection and principal component analysis.

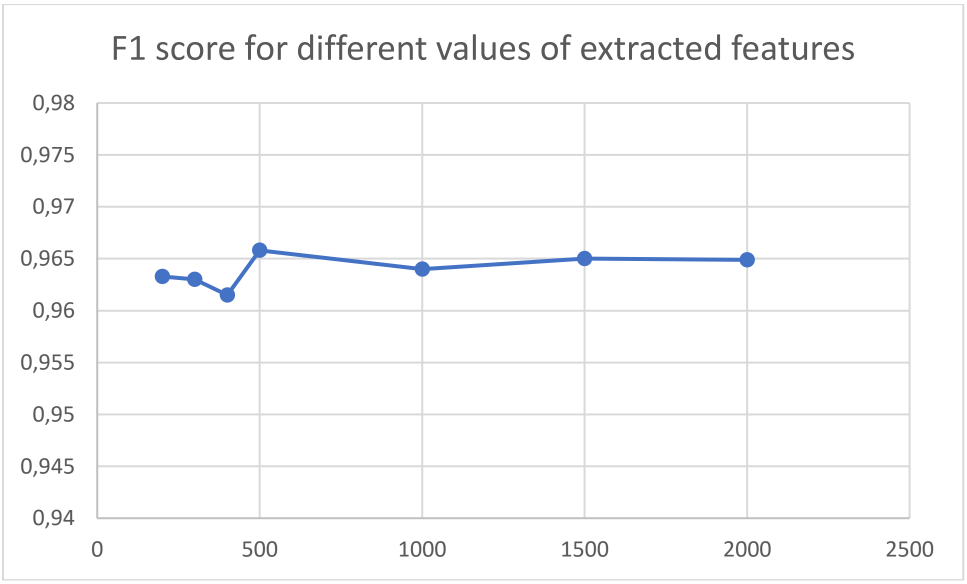
4. Tuning and validation

The parameter tuning procedure was assessed mainly through experiments and the SKLEARN tool GridSearchCV.

The two main parameters that I tried to improve were the number of features to use for the classification and the hyperparameters of the MLP classifier.
To select number of features to extract with the SVD I decided to start with the full array of tokens. This configuration is extremely taxing on the hardware and is by far the one that requires the longest time to run. On the other hand, this approach is the one that gave the highest score on the public evaluation set (0.973) and is the one associated to the *Solution2.py* file.
Starting from this feature set, I started to test how reducing the number of features would impact my results. The results showed that the stemming and SVD were the portions of the algorithm that required the longest time to run when considering relatively small sets of features (around 1k), while the classification procedure was not heavily influenced by the number of features we used for the task.
Since upscaling the number of features didn’t yield a performance increase, I set the SVD to extract 500 features. This number of features is a good compromise between the required time to run and the performance obtained (0.966 on the public solution).
For what concerns the classification execution time I found that enabling the *early stopping* parameter for the classifier quickened the classification task by a consistent margin. This feature stops the execution of the classification if after a certain number of iterations, it doesn’t register a decrease in the loss function higher than a set threshold.
The tuning procedure for the classifier hyperparameters was much simpler since SKLEARN provides a function (GridSearchCV) that test all the possible combinations of parameters that we want to try. This procedure has been carried on a smaller feature set (5000 max features with SVD reducing them to 200) since it is extremely time consuming, and the results are the parameters that my classifier currently uses.
Finally, for validation purposes I reserved a portion of the training set to be used as a test set to evaluate the performance of the classifier using the F1 score. Note that in my solutions this portion of code is commented and is kept only to demonstrate the approach used.



The graph shows how the runtime increases with the number of features we want to extract putting in evidence the increase in time required to extract over 400 features.



As we can see the graph shows no performance gain once we use more than 500 features

5. References

[1] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html
[2] Windeatt T. (2008) Ensemble MLP Classifier Design. In: Jain L.C., Sato-Ilic M., Virvou M., Tsihrintzis G.A., Balas V.E., Abeynayake C. (eds) Computational Intelligence Paradigms. Studies in Computational Intelligence, vol 137. Springer, Berlin, Heidelberg
[3] https://www.nltk.org/_modules/nltk/stem/snowball.html
[4] https://en.wikipedia.org/wiki/Multilayer_perceptron#Applications