Data Science Lab: Process and methods

Politecnico di Torino

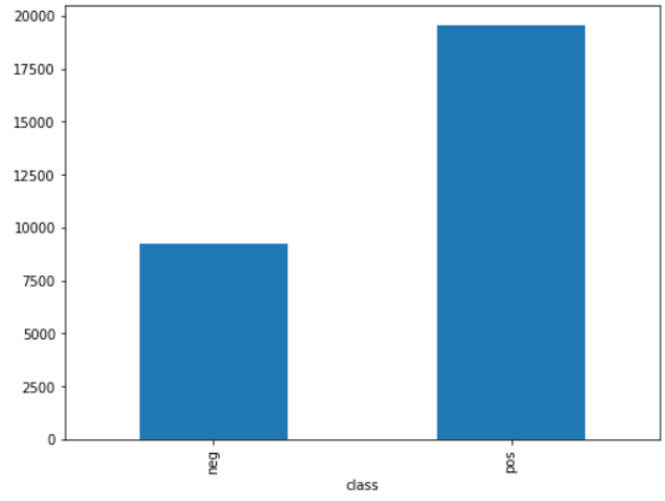
Project report
Student ID: s273693

Exam session: Winter 2020

1. Data exploration (max. 400 words)

Concerning the Data Exploration phase, the main goal was to get to know the dataset and try to start recognizing which pre-processing and classification methods would fit better to solve this specific task. Since the model must be trained using the *development.csv* file, all the steps mentioned on this section were performed on this dataset, and not on *evaluation.csv*.

After loading the development dataset as Pandas' DataFrame, the first basic step was to check if all the classifications were actually either 'pos' or 'neg', or if there was a misclassification, which would add a pre-processing step. On top of that, it was also checked the number of positive and negative classifications. In order to do this, Pandas' *groupby('class')* was used and the following histogram was generated:




```
class
neg     9222
pos    19532
Name: text, dtype: int64
```

This result shows that there are no evaluations other than 'pos' or 'neg', meaning there's no need to do a pre-processing step to solve this. Also, even though most of the evaluations are positive, there's a large number of negative evaluations as well, so there's no risk for the classification method to treat negative classifications as outliers

(which was already expected, since this is a common problem in classification tasks with many possible classes, which is not the case of this task).

The next step of the Data Exploration was to see the most common words for the positive and negative evaluations and check if there were any equalities between the two lists. In order to do this, CountVectorizer, from sklearn, was used. The following table shows the top 10 most common words for each of the classes:

	Pos	Neg
0	di	di
1	la	la
2	il	che
3	un	il
4	per	non
5	in	un
6	che	per
7	hotel	in
8	molto	una
9	una	era

It's possible to see that most of the words are meaningless words in terms of classification. Therefore, the Italian stop words available in nltk.corpus were removed, except for the word "non", given its negative meaning.

The table below shows the top 10 most common words for each of the classes after this small pre-processing:

	Pos	Neg
0	hotel	non
1	molto	hotel
2	non	camera
3	personale	molto
4	colazione	colazione
5	camera	personale
6	camere	stato
7	posizione	solo
8	disponibile	bagno
9	ottima	stelle

The words existing in both lists are great candidates for stop words for the pre-processing phase, which will be better detailed in the next section.

The last step of this phase was checking the vocabulary of the CountVectorizer to see if there was any kind of non-alphanumerical symbols. The only items found were all words separated by "_" instead of a blank space.

2. Preprocessing (max. 400 words)

The first step concerning the preprocessing phase was deciding how to represent the words of the documents. The method chosen to give a numerical representation was the TfidfVectorizer, a tool available in sklearn that not only converts textual data into count vectors, but also penalizes the words present on many documents. To include bigrams like "non + adjective", the parameter ngram_range was set to (1, 2).

The next step was creating a class based on the one provided by the course on laboratory 5. This class is responsible for tokenizing and stemming the documents. The decision to stemming over lemmatizing was due to the fact that the reviews are in the Italian language, and there's a stemmer available in nltk called SnowballStemmer (information available in https://www.nltk.org/howto/stem.html) that has Italian as one of its languages. The process of stemming is responsible for cutting the affixes of the words, even if it generates a non-word element. It's faster than lemmatizing, but it's less accurate, since it doesn't analyze the context of the word to return it to its root form, however it works well for reducing the number of features by grouping words with the same stem, given that many words that share the same stem, also share the same meaning. The class is also responsible for dropping out any "_" or "'" and replace it with a blank space, and for removing any punctuation, or word with a length smaller than 2, as long as it's not a number. An object of this class is then passed as the tokenizer parameter of the TfidfVectorizer. The following figure shows the class implementation:

```
class StemmeTokenizer(object):
    def __init__(self):
        self.stemmer = SnowballStemmer('italian', ignore_stopwords=True)
    def __call__(self, document):
        lemmas = []
        re_digit = re.compile("[0-9]")
        document = document.replace("_", " ")
        document = document.replace("'", " ")

        for t in word_tokenize(document):
            t = t.strip()
            lemma = self.stemmer.stem(t)
            if lemma not in string.punctuation and (re_digit.match(lemma) or len(lemma) > 1):
                lemmas.append(lemma)
        return lemmas
```

One of the parameters of the TfidfVectorizer are the stop words. As mentioned before, among the top 10 most common words in each of the classes (already not considering the Italian stop words contained in nltk.corpus), the stems of the ones present on both classes (6 in total) were added to the stop words (except for the word "non"). Besides that, the Italian stop words include many conjugated verbs, but don't include them on the infinitive form, so the stems of the verbs "avere", "fare", "essere" and "stare" were also passed to the stop_words parameter.

Even though the parameter min_df is part of the preprocessing phase, its value was set during the tuning process, described in the last section.

3. Algorithm choice (max. 400 words)

The algorithms chosen to test their initial results were Random Forest Classifier, Linear Support Vector Classification (Linear SVC) and MLPClassifier, all of them available in the sklearn library. The explanations of why these algorithms were chosen are the following:

- Random Forest Classifier: this algorithm was chosen due to its high accuracy, especially when compared to Decision Trees, to its fast training phase and because it estimates which features are important for the classification task.
- LinearSVC: this method belongs to the set of learning methods known as Supported Vector Machines (SVM). According the sklearn website itself, these methods are very effective in high dimensional spaces (which is the case, given the large number of features generated even after preprocessing), they are memory efficient and, even though there's a risk of overfitting, they work well even if the number of features is greater than the number of samples. The choice for LinearSVC instead of SVC happened because, also according to sklearn website, the linear one should scale better for a large number of samples.
- MLPClassifier: a Feed Forward Neural Network model, it was chosen due to its very high accuracy, even though it takes a long time to fit the train dataset and is very sensitive to feature scaling. It's also robust to noise and outliers, which is very important when analyzing texts. It's different from Logistic Regression because there may be one or more non-linear layers between the input and output layers (the so called hidden layers).

In order to choose which of these 3 algorithms to use, a cross-validation test was performed using the tool K-fold from sklearn library with n_split = 5 to avoid overfitting. Each time a new split happened, the TfidfVectorizer.fit_transform method was called on the training set and then only the transform would be applied to the test set. Therefore, for each of the 3 models, 5 f1_scores would be calculated. The average between these 5 measures would tell which algorithm would be chosen. All the methods were applied with its default parameters, only the 'tol' for the MLPClassifier was set to 0.01 in order to make the process faster. These were the results:

- Random Forest Classifier: 0.903
- LinearSVC: 0.934
- MLPClassifier: 0.945

Given these results, MLPClassifier was chosen, even if this was the slowest method among all of them, it was the most accurate.

4. Tuning and validation (max. 400 words)

To tune and validate the model, the tool ParameterGrid from sklearn was used. The parameters alpha and batch_size were tested, with the default value, a higher value and a lower value. The tolerance was set to 0.01 and the number of iterations with no change was set to 1, so the search for the best parameters would be faster. For each combination of the parameters, the f1_score was computed.

Also in order to make the process faster, the parameter min_df from TfidfVectorizer was set to 7, which reduced the number of features. After all the combinations were tested, it was possible to see that even changing the parameters, the results were all very similar. So the default values for alpha and batch size were adopted, where the f1_score for a training set corresponding to 80% of the development dataset was 0.968.

The parameters of the TfidfVectorizer have a meaningful impact over the results. With that in mind, tests were made to see if removing the stop words included on the preprocessing phase improved the results, but it did not and also made the algorithm a lot slower. The value for min_df was also another thing to consider, but the tests showed that for values lower than 4 (which increased the number of features in the classification) there was a memory problem, given that a negative aspect of the neural networks is that they consume a lot of memory; and for higher values, the accuracy of the model decreased. So it was set to 4.

The tolerance adopted for the MLPClassifier was set to 0.01, because setting the parameter Verbose to True, it showed that the value for the loss starts to decrease very slowly after it reaches 0.01. When set to 0.001, the algorithm took a much longer time to converge and the impacts in the result were meaningless.

With all the parameters stablished, a new test was made and the confusion matrix generated was as follows:

	Neg	Pos
Neg	1765	96
Pos	78	3812

It's possible to see that there are more negative classifications judged as positive than the other way around. But the difference is not big and was the best result achieved, adding the word "non" back to stop words list only made the result worse.

In order to improve the model, the whole development dataset was used as training set to predict the evaluation dataset.