

Data Science Lab: Process and methods

Politecnico di Torino

Project report

Student ID: s278727

Exam session: Winter 2020

1. Data exploration (max. 400 words)

The starting point of my effort to create a classification pipeline is an exploratory analysis of the two datasets, in order to better understand the available data (with respect to its size and distribution). This step has been performed directly in a Python IDE, by means of Python built-in methods.

There are two datasets of textual data to work with, both given in .csv format:

1. The “**development set**” (development.csv) is a dataset composed of 28754 records, each characterized by a *text* field containing a TripAdvisor review (written in Italian), and an associated *class* field, containing the sentiment (‘pos’ for positive, ‘neg’ for negative) associated to the review.

There are no missing values in any record.

The dataset is unbalanced: there are 19532 positive and 9222 negative reviews. This imbalance must be addressed before training a classifier, to allow it to make good predictions with respect to both positive and negative reviews (i.e. to avoid any bias towards the majority class ‘pos’).

2. The “**evaluation set**” (evaluation.csv) is a dataset composed of 12323 records, each characterized by a single *text* field containing a TripAdvisor review (in Italian). The purpose of the classifier pipeline to be built is to (correctly) predict the sentiment of these reviews.

There are no missing values in any record.

Nothing can be said about the distribution of positive and negative reviews in this dataset, as this dataset lacks the *class* field.

Both the datasets have reviews (in the *text* field) of varying length, ranging from 58 to 9221 characters.

The development dataset has a set of more than 50,000 unique words. In this context, we define the concept of “*word*” as an alphanumeric substring of any review, preceded and followed by white spaces (or placed at the beginning/end of the review). Words are also referred to as “*tokens*”.

Many usual and unusual non-alphanumeric characters are present in the reviews of both datasets; among these: punctuation, emojis, special characters like '\n', '\t', '€' and many others. I consider them to be irrelevant (not meaningful) for our goal of predicting the sentiment of the reviews, and I opted to clean them out, as described in the next section. However, a more complex approach could take emojis into account, as we can expect some of them to be meaningful expressions of the sentiment of one's review.

2. Preprocessing (max. 400 words)

The development set is split into two subsets, in a stratified fashion: 95% training set, 5% test set.

	Training set (95% of dev set)	Test set (5% of dev set)
# of 'pos'	18555	977
# of 'neg'	8761	461
TOTAL	27316	1438

Before training any classifier, the training set has been preprocessed through the following steps (presented in the order they are performed):

1. **Tokenization, case normalization and stop words removal:** every review is cleaned from all the non-alphabetic characters (replaced by white spaces) and is then tokenized (i.e. broken into single tokens); the resulting tokens are converted to lowercase; the most common stop words for the Italian language are then removed from the tokenized reviews.
A list of roughly 300 common Italian stop words has been retrieved from the *stop_words* Python package [1].
2. **Stemming:** the tokens of every document are stemmed, keeping only the Italian word stem of every processed token. Stemming acts as a useful feature reduction step.
The stemming algorithm for the Italian language is retrieved from the *nltk* package [2].
3. **Bigrams extraction:** bigrams (sequences of two adjacent tokens in a document) are extracted; this step hugely increases the number of features.
4. **Minimum and maximum document frequency thresholds:** tokens and bigrams (the features) which are present in less than a minimum number or in more than a maximum number of reviews are removed. Very frequent and infrequent words, just like stop words and words which appear only a few times in the corpus of reviews,

are considered irrelevant for the sentiment analysis. This serves as a feature reduction step.

5. **TF-IDF matrix computation:** the TF-IDF matrix for the training set is $N_{train} \times M$, where N_{train} is the number of reviews in the training set and M is the size of the “vocabulary” learnt from the training set (the number of unique tokens/bigrams kept until this step).
6. **Oversampling:** oversample the minority class (‘neg’) to make the training set balanced with respect to the ‘pos’ and ‘neg’ classes, thus increasing N_{train} . The technique used to perform oversampling is the Synthetic Minority Oversampling Technique (SMOTE) [3].
7. **Feature selection with ANOVA F-test:** keep only a percentage of features for each review: those ranking the highest according to the ANOVA F-test (a statistical test for telling which features are more discriminating for a certain class [4]). This is a major feature selection step, accountable for the largest feature reduction in the described preprocessing pipeline. This removes columns (features) from the TF-IDF matrix of the training set, thus decreasing M .

3. Algorithm choice (max. 400 words)

The classification algorithm I have chosen is the **Multinomial Naïve Bayes** (MNB) classifier, in a formulation which relies on the very simple TF-IDF textual data representation.

Let x_i be the count of the i -th token (single word or, eventually, an n-gram) appearing in a document x . In its most basic formulation, the MNB classifier is based on the strong (naïve) assumption that the feature probabilities $P(x_i|c_j)$ are independent given the class c_j , to which that document belongs:

$$P(x_1, \dots, x_M|c_j) = P(x_1|c_j) \cdot \dots \cdot P(x_M|c_j) \quad \forall j$$

Another assumption that is made regards the probability distribution $P(x|c_j)$, where $x = (x_1, \dots, x_M)$: it is assumed multinomial.

Despite the two severe assumptions of statistical independence of features and multinomial distribution of the set of features in a document, the MNB classifier is often used as a baseline in text classification because it is fast, easy to implement and relatively effective, especially on large datasets (whereas less erroneous algorithms tend to be slower and more complex), as argued in [5] and [6].

Of course, human-generated text is not a set of variables (tokens) which follow a fixed multinomial distribution, therefore one could expect the MNB classifier to perform poorly on text classification tasks.

However, it is argued that these performance problems can be addressed by performing specific feature transformation on textual data to make it look “more multinomial”, as described in detail in [6].

The specific feature transformation to be performed is the **TF-IDF text representation**; in this version of the MNB classifier, the input documents (i.e. the reviews, in our use case) are in the form $x = (x_1, \dots, x_M)$, where every x_i is the TF-IDF value of the i -th token in the document, as opposed to the basic model previously described (in which x_i was the absolute frequency of the i -th token in the document).

4. Tuning and validation (max. 400 words)

There are several hyperparameters to be set in order to train the classifier. The ones which I decided to manually set are resumed in the following table

HYPERPARAMETER	DESCRIPTION	PHASE
max_df	Maximum document frequency a token or bigram must have in order to be stored in the vocabulary.	Preprocessing (min_df and max_df thresholds)
min_df	Minimum document frequency a token or bigram must have in order to be stored in the vocabulary.	Preprocessing (min_df and max_df thresholds)
percentile	Percentage of features to keep when performing feature selection.	Preprocessing (feature selection with ANOVA F-test)

All the other hyperparameters values have been set automatically with default values, provided by the python packages used.

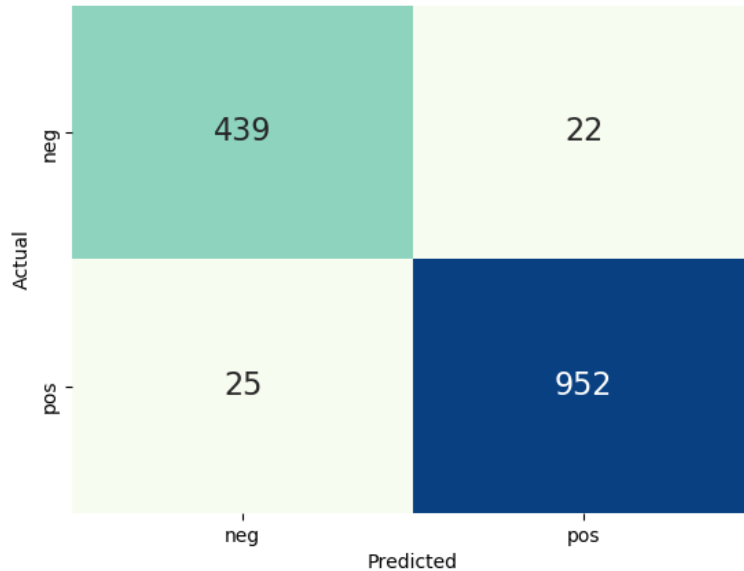
Different configurations of possible hyperparameters values have been tried. Each trial has been performed using k-fold cross-validation on the training set, with k=5 folds: for each configuration, 5 MNB classifiers are trained on 4 of the 5 folds and evaluated on the remaining one (validation set) computing the weighted F1 score. The number of trials to perform is equal to the number of possible configurations of the hyperparameters values, and each trial consists of the five rounds of the 5-fold cross-validation.

The best configuration is that which maximizes the mean of the five weighted F1 scores on the five validation sets, computed for every different configuration. For my pipeline, it happens to be:

max_df = 0.06	Only tokens/bigrams appearing in at most the 6% of the documents in the training set are kept
min_df = 2	Only tokens/bigrams appearing in at least 2 documents in the training set are kept
percentile = 18	Keep only the 18% of the remaining features with the highest ranking score on the ANOVA F-test.

With these values, the classification pipeline achieves a weighted F1 score of 0.967 on the test set, which seems very good (despite the “strong” assumptions of the MNB classifier).

The confusion matrix for the test set predictions is the following:



Actual	neg	pos
	439	22
pos	25	952
Predicted		

In particular, the F1 score for the 'neg' and 'neg' class are 0.949 and 0.976 respectively. The lower F1 score obtained on the class 'neg' might be due to the oversampling performed on that class: most of the 'neg' samples the classifier has been trained on were artificially made up with the SMOTE technique, so they are not totally reliable and representative of the 'neg' class.

The same weighted F1 score is achieved on the public part of the evaluation set, proving that the classification pipeline thus created generalizes well to new samples.

5. References

- [1] List of Italian most common stop words, URL: <https://github.com/Alir3z4/stop-words/blob/bd8cc1434faeb3449735ed570a4a392ab5d35291/italian.txt>
- [2] Italian stemming algorithm, URL: <http://snowball.tartarus.org/algorithms/italian/stemmer.html>
- [3] Chawla et al., *SMOTE: Synthetic Minority Over-sampling Technique*, Journal of Artificial Intelligence Research 16 pp. 321–357 (2002)
- [4] ANOVA F-test entry on Wikipedia, URL: https://en.wikipedia.org/wiki/Analysis_of_variance#The_F-test
- [5] McCallum, Nigan, *A Comparison of Event Models for Naive Bayes Text Classification*, AAAI-98 workshop on learning for text categorization pp. 41–48 (1998).
- [6] Rennie et al., *Tackling the Poor Assumptions of Naive Bayes Text Classifiers*, Proceedings of the 20th international conference on machine learning (ICML-03) pp. 616–623 (2003)