Data Science Lab: Process and methods

Politecnico di Torino

Project report
Student ID: s276241

Exam session: Winter 2020

1. Data exploration

The given reviews from the website tripadvisor.it have been carefully explored to gain important knowledge from the dataset before building the model. Note that **all** graphs and observations are related to the development set only.
Throughout this report, reviews will also be referred as *documents.*

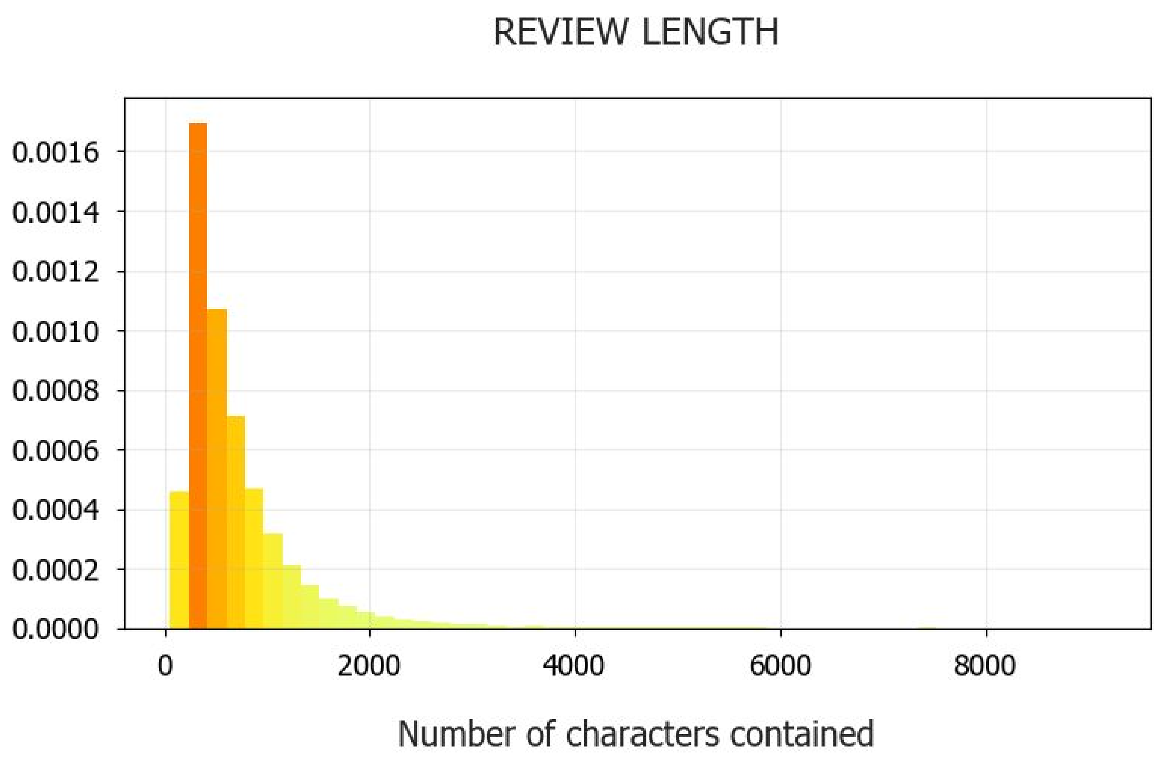
DATASET AND CLASS SIZES

Full dataset size: **41077** (*28754 reviews in development set*)
# of positive reviews in dev set: **19532** (*68%*)
# of negative reviews in dev set: **9222** (*32%*)

The two classes in the development set show a significant unbalance, hence some further attention on this has to be made when evaluating the results. In this case, an overall measure weighted by class support will be mainly considered (*f1_weighted*) when evaluating the model, assuming that both classes have the same importance in our sentiment analysis (getting one 'pos' review right is as important as getting one 'neg' review right).

REVIEW LENGTHS

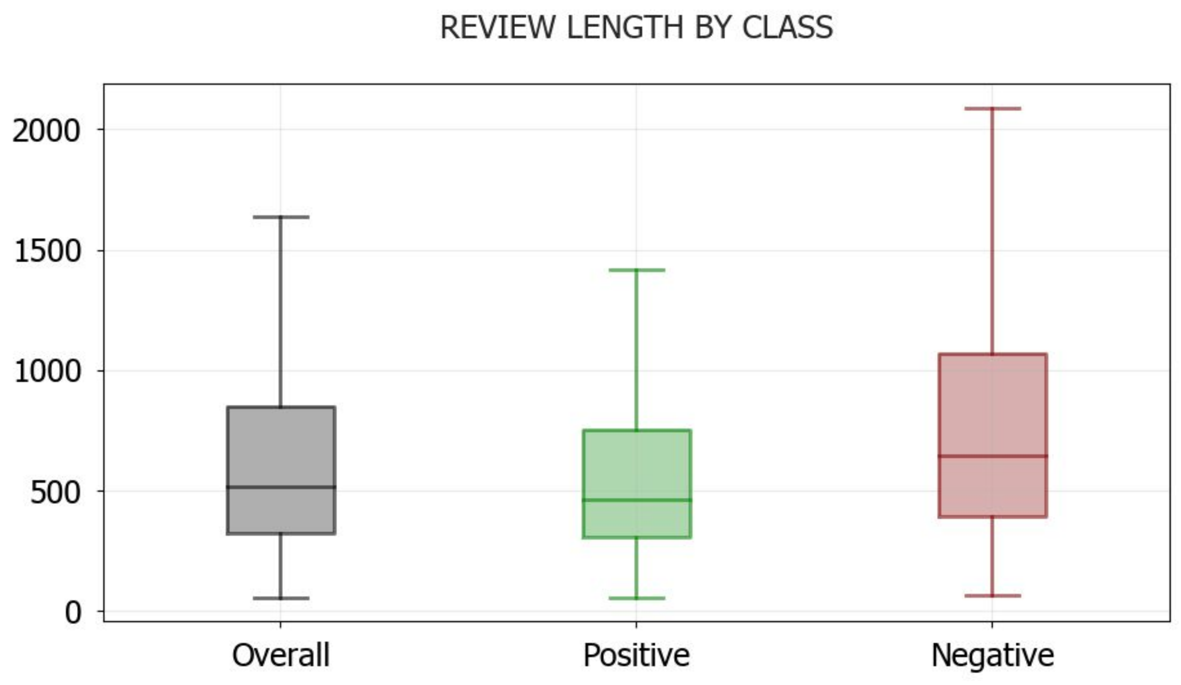
A deep analysis on the length of each document (of the dev set) by means of number of characters has been performed to better understand the distribution of review lengths:



The table below shows more in-depth statistics on review lengths (in number of characters), separately for each class:

	Overall	**Positive class**	**Negative Class**
Mean	*701*	*624*	*864*
St. Deviation	*606*	*513*	*740*
Median	*515*	*465*	*644*
Min; Max	*58; 9153*	*58; 7800*	*67; 9153*

Finally, review lengths distributions across the two classes can be summarized with boxplots:

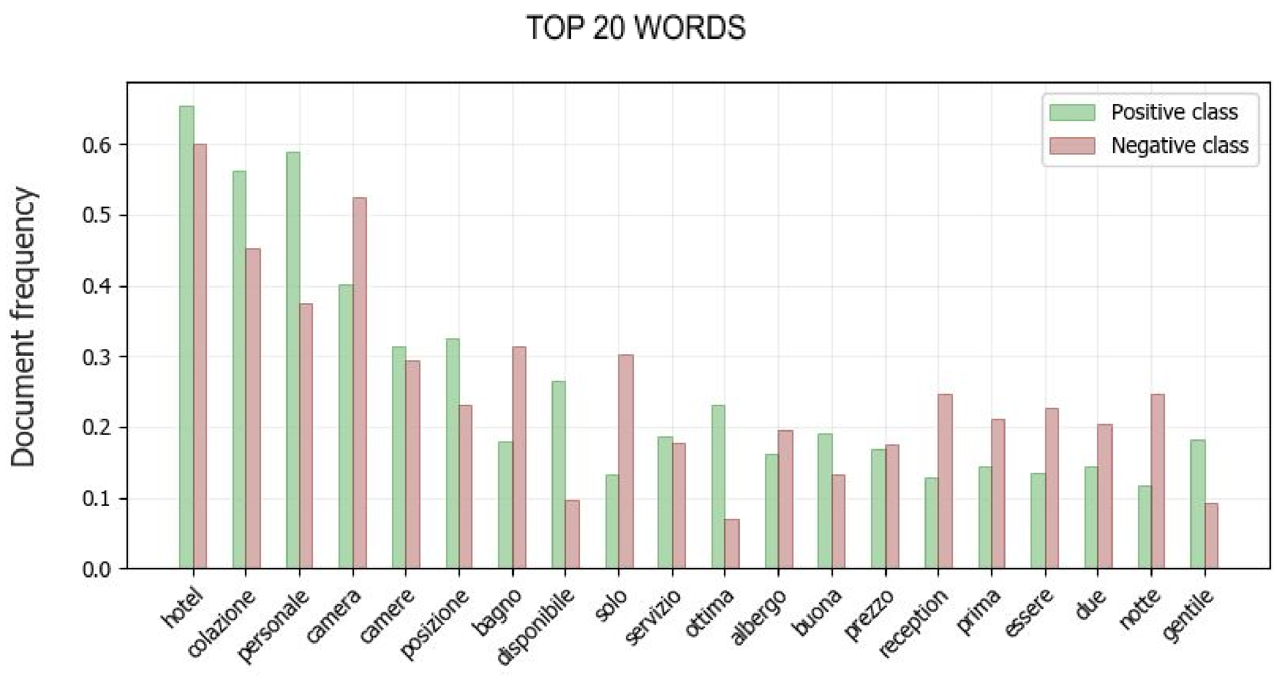


None of the edge cases have been excluded from the set, as shorter reviews can be considered just as important for the analysis as longer ones, and viceversa ([1]).

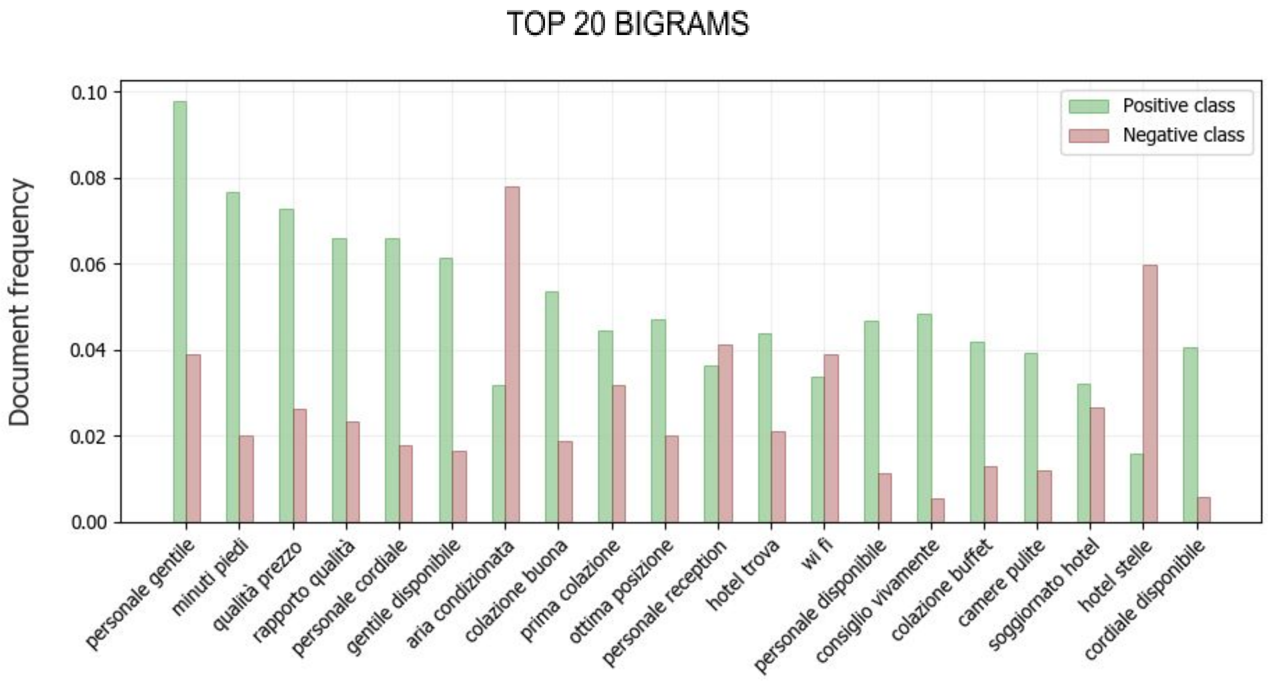
FURTHER EXPLORATION OF REVIEWS

Frequent sets of words have been analized through the *fpgrowth* algorithm.
Very frequent words as well as very rare ones will be carefully taken into account during the preprocessing and validating steps, to figure out how they are impacting the quality of the model.

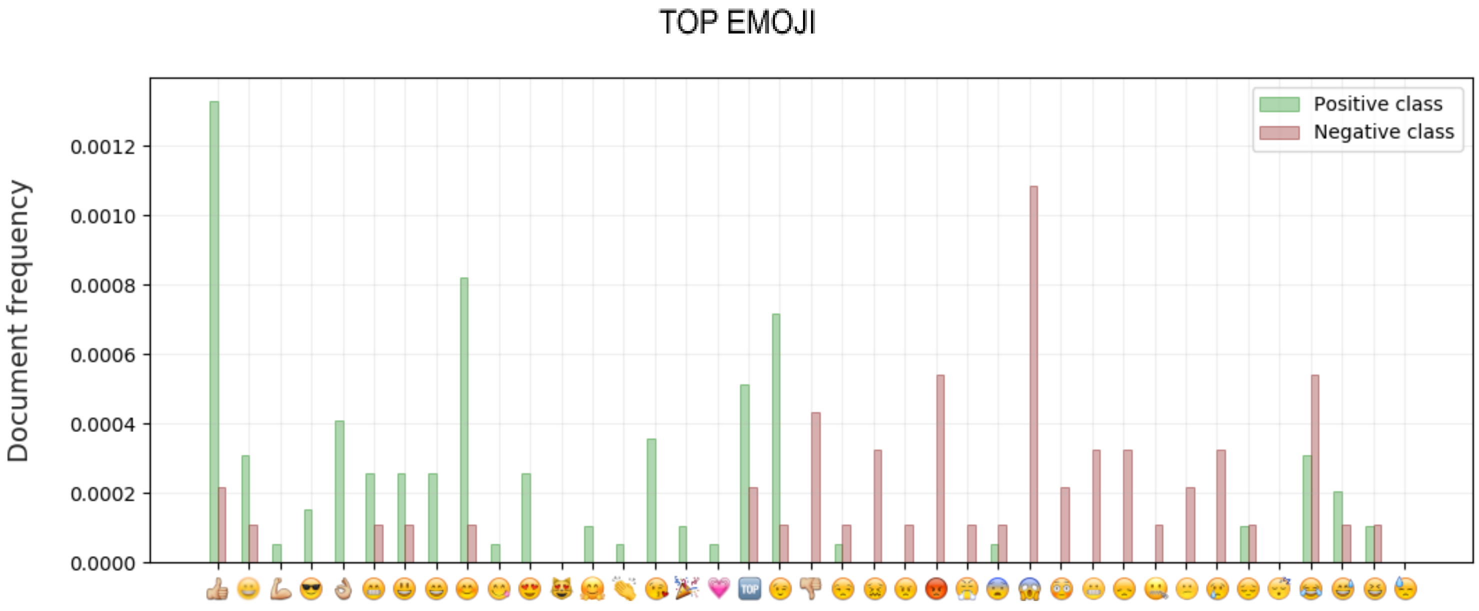
The top 20 overall most frequent words (after stop-words removal) have been compared to how frequent they are in each class, respectively:



The same analysis has been done for bigrams:



In addition, emojis have also been analized with the same algorithm to check how they have been used based on the review sentiment:



Finally, reviews have been investigated with the help of a language detection library ([2]):

- *Italian reviews*: **28749**
- *English reviews*: **2**
- *Turkish reviews*: **1**
- *Japanese reviews*: **1**
- *Spanish reviews*: **1**

2. Preprocessing

The tf-idf approach has been chosen for feature engineering the dataset. This strategy is widely used in literature for text classification tasks and sentiment analysis, as we can scale absolute word counts with respect to how frequent each word is in the training dataset.
Also, as previously mentioned, each review is given the same importance regardless of length, so the preprocessing phase makes sure that all documents are getting normalized to have l2-norm=1, after being transformed into the Tf-Idf space (this is done by default by the sklearn class *TfidfVectorizer*).

Note:
despite articles stating different opinions, in this solution ONLY the development set is used for fitting the transformers (e.g. TfidfVectorizer) in the pipeline[1] .

STOP WORDS AND TOKENIZING

Given the really high number of dimensions (due to the nature of the problem), a feature selection task has to be taken into consideration. In fact, dimensions that are not useful for the task could negatively affect the analysis (over-fitting, curse of dimensionality).
Features (words) should then be analized and removed if considered to give little to zero contribution to a sentiment analysis task. The next steps will take care of analyzing how the absence of stop words impacts on the final score, and also whether an automatic feature dimensionality step (e.g. PCA) is needed to improve the quality of the classifier.

Example of stop words based on high frequency counts: *'di'*, *'la', 'il', 'un', 'non', 'per', 'che', 'in', 'hotel', 'una', 'con', 'era', 'molto'*

The initial configuration of the TfidfVectorizer set in the preprocessing step to account for very frequent words as well as possibly misspelled words is as shown:
- max_df = 0.5
- min_df = 2

In addition, all punctuation, numbers, urls and special characters have been replaced by blank spaces.

Finally, "positive" and "negative" emojis have been replaced by a placeholder string respectively, while neutral ones have been filtered out.

STEMMING

An existing italian stemmer from nltk (*nltk.stem.snowball.ItalianStemmer* [3]) has been tried out during preprocessing. This stemmer seems to successfully cut off last syllables. Some further investigation will be done in the tuning step to figure out the efficiency of the stemmer in terms of time and accuracy.

3. Algorithm choice

Different classifiers have been taken into account for building the model, among the ones covered during the course:

- **Random Forest Classifier**: provides interpretability of the results with a ranking on feature importances. Medium accuracy.
- **Naive Bayes Classifiers**: based on probability distributions they are intuitive and fairly fast, but usually less accurate. Since features are coming from a Tf-idf approach, MultinomialNB (and its variant ComplementNB [4]) have been considered, instead of a GaussianNB model (which assumes features are more or less gaussian distributed in the first place).
- **Linear SVM Classifier (LinearSVC)**: medium-to-high accuracy, but lacks of interpretability. Suited for high dimensional problems and fairly fast when the problem is linear.
- **KNearestNeighbors Classifier:** based on distance, suffers from curse of dimensionality but it's very fast at training.

A general analysis has been initially performed on all of them on the development set, to check how they would perform out of the box.

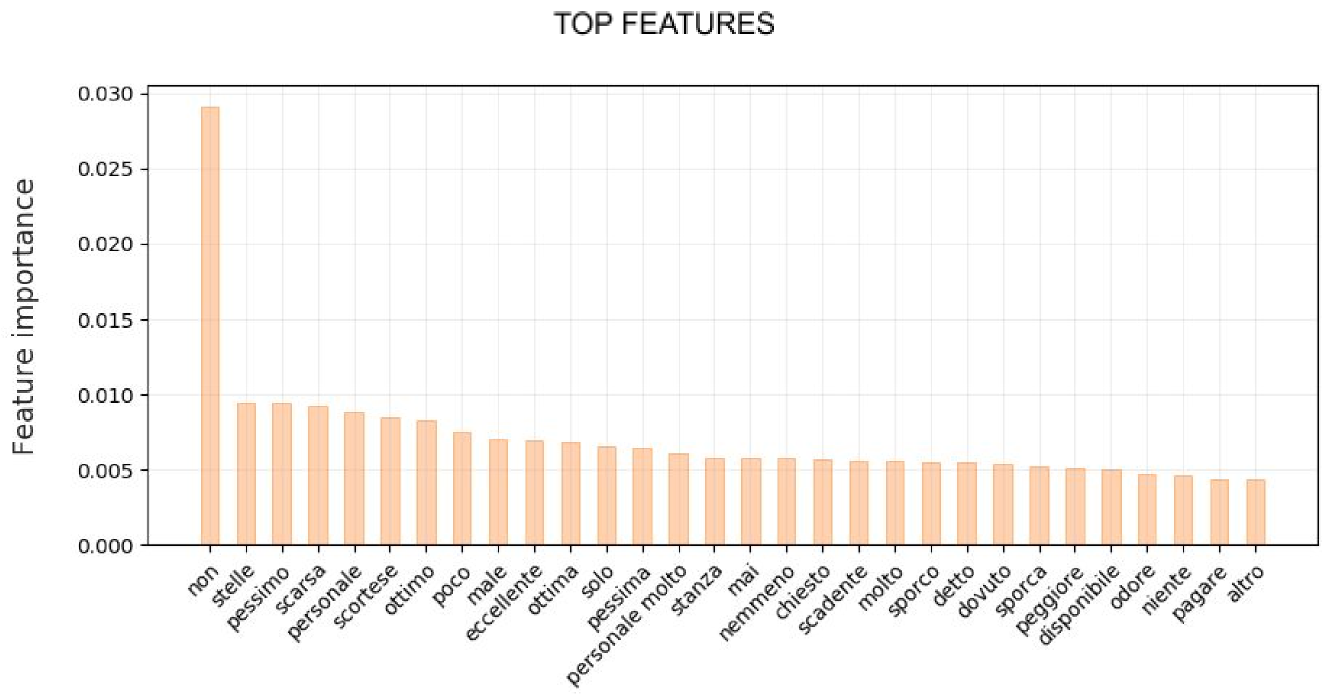
		KNN	**Random Forest**	**Multinomial NB**	**Complement NB**	**LinearSVC**
With stopwords	*No PCA*	0,87398	0,91453	0,92305	0,9485	**0,96069**
	PCA	0,88017	0,86083	0,92403	0,9492	0,95851
Without stopwords	*No PCA*	0,86651	**0,92061**	**0,92769**	0,950179	0,96037
	PCA	**0,89812**	0,87236	0,92677	**0,95036**	0,95951

F1_weighted score on a 10-fold cross validation.

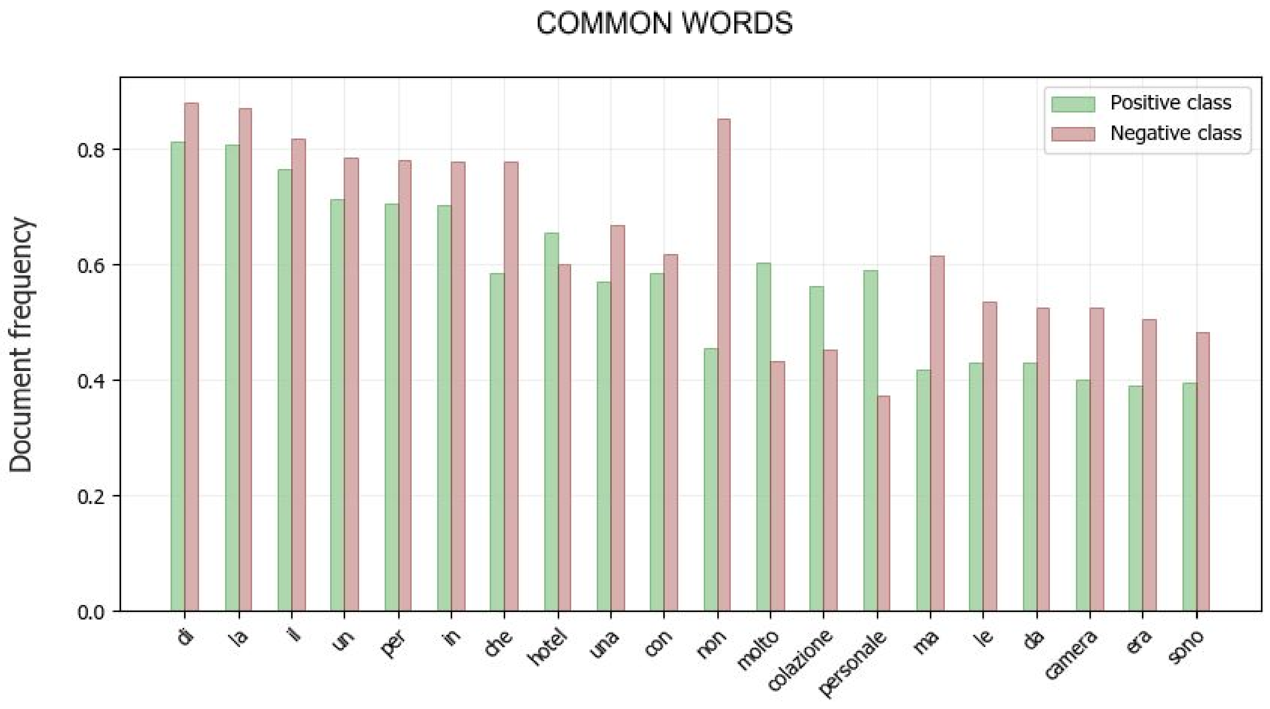
Scaling techniques have not been considered for the analysis as features are in a sparse representation and mostly 0's anyway, plus kind of already scaled by the Tf-idf technique. In addition, each row is normalized by default to l2-norm=1, as previously mentioned.

As expected, the **KNN** model was not the most accurate for the task, due to the high number of dimensions of the problem. In all other cases, dimensionality reduction did not show any significant improvements, or even proved to be worse.

The **Random Forest** classifier showed only a medium score, but has been used to further investigate feature importances:



As shown, even **very common words** such as "*non*" and "*solo*" highly affect the splits in the decision trees. This may explain why different results were obtained when removing stopwords in the previous table. Therefore, a new analysis on the distribution of stopwords across the two classes has been done:



As expected, the word "*non*" shows a very high correlation with the negative class[2].

Finally, the **SVC classifier** has been chosen for building the model, since it showed the highest accuracy and the highest robustness to number of dimensions (articles [5] and [6] on SVC advantages for high-dimensional problems).

4. Tuning and validation

The hyperparameters taken into account for the tuning process are as follows:

- minimum word length
- stop words removal *(with significant stopwords whitelisted)*
- stemming
- emojis preprocessing *(splitting into 'positive', 'negative' and 'neutral' emojis)*
- ngram range *(possible use of bigrams and trigrams)*
- maximum document frequency of a term
- minimum document frequency of a term
- PCA
- LinearSVC's own hyperparameters: loss, tolerance, C (regularization)

Only the linear kernel has been used for the SVC, as the number of dimensions is already very high and it proved to be definitely more efficient during training.

The tuning process has been performed by using the entire development dataset, by grid-searching over all the hyperparameters mentioned and evaluating each configuration on a 10-fold cross validation using f1_weighted score.

Finally, the predictions with the best configuration found have been submitted to the platform to evaluate whether the trained model could generalise well to new **unseen** data points.

OBSERVATIONS

The removal of nltk stop words did not improve the model, unless a small number of common words were whitelisted (*'non', 'ma', 'solo', 'molto'*).
The use of a stemmer and bigrams together with unigrams showed to positively affect the results by a considerable amount, but on the other hand it highly increased the time taken during the preprocessing phase and the number of final dimensions (~*70 seconds for predicting the eval set*).
The cleaning and filtering of emojis also proved some significant positive changes in the quality of the prediction.

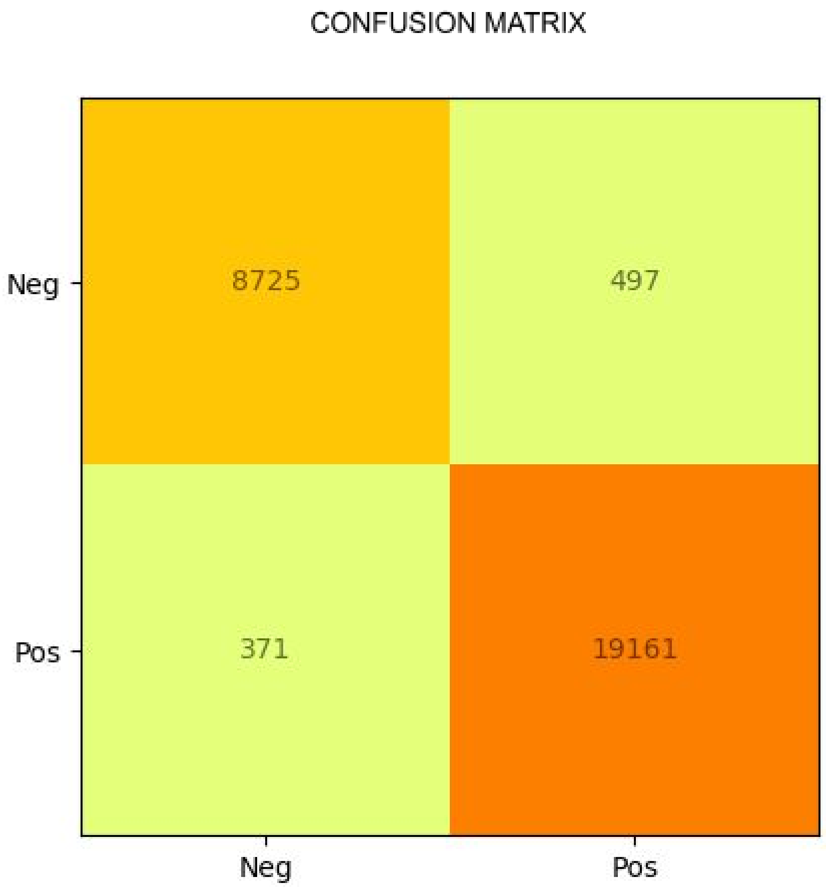
Best config found:

- min_length: *>1*
- stop words removal: *yes, whitelisted*
- stemming: *yes*
- emojis preprocessing: *yes*

- ngram_range: *unigrams & bigrams*
- max_df: *0.5*
- min_df: *>=1*
- PCA: *No automatic dimensionality reduction*
- loss=*'hinge'*, tol=*1e-8*, C=*3.0*

Overall, the model went from the "out-of-the-box" score of *0.96069* to the final higher score of *0.97034*, on the dev set. Note that, in terms of accuracy, a 1% change in the evaluation set would represent a net difference of *123 reviews*, which is definitely NOT negligible.

The confusion matrix can be inspected for more info on single review predictions (*cross_val_predict on dev set*):



Once submitted, the predictions obtained a score of *0.97545*. Since the model performed similarly with respect to the training set, it proved to generalize well on new unseen data.

References

[1] Outlier management: when to drop or not to drop
https://www.theanalysisfactor.com/outliers-to-drop-or-not-to-drop/

[2] Language detection library for python *langdetect*
https://pypi.org/project/langdetect/

[3] Italian stemming algorithm
http://snowball.tartarus.org/algorithms/italian/stemmer.html

[4] Naive Bayes classifiers and their variants for text classification
https://scikit-learn.org/stable/modules/naive_bayes.html

[5] Support vector machines (SVM's) for classification
https://scikit-learn.org/stable/modules/svm.html

[6] Top 4 advantages and disadvantages of SVC
https://medium.com/@dhiraj8899/top-4-advantages-and-disadvantages-of-support-vector-machine-or-svm-a3c06a2b107

[7] Text classification using ngrams
https://medium.com/machine-learning-intuition/document-classification-part-2-text-processing-eaa26d16c719

[8] A complete data exploration task on text data
https://towardsdatascience.com/a-complete-exploratory-data-analysis-and-visualization-for-text-data-29fb1b96fb6a

[9] "Data science lab: process and methods": course slides and lab experiences

Notes

[1]despite articles stating different opinions (TfidfVectorizer: should it be used on train only or train+test), in this solution ONLY the development set is used for fitting the transformers in the pipeline (e.g. TfidfVectorizer). Therefore, the IDF values and the vocabulary will be learned from the training set only, in order to check if the model generalizes well to **actual unseen** data (evaluation set). If the latter was instead taken into account for fitting the transformers, some information would be leaked into the model from the test set, and the risk of over-fitting would be higher.

[2]since the probability of finding "non" in a negative review is approx. 0.9, it seems fair to say that when "non" is encountered, then the 'neg' class should be predicted almost for certain. In reality, if class supports in the dev set were taken into account (68% 'pos' - 32% 'neg') the conditional probability P['neg' | "non"] would be just about 0.5, because 'pos' reviews are much more likely to happen in the first place. For example, Naive Bayes classifiers do consider class supports in the training set (https://towardsdatascience.com/all-about-naive-bayes-8e13cef044cf). Finally, it is still very true that when "non" is NOT found, it is very likely that the review is a positive one (P['pos' | *not*("non")] ≅ 0.9), thus the GINI index is affected. This new knowledge has been used to whitelist some words in the nltk stop words list.