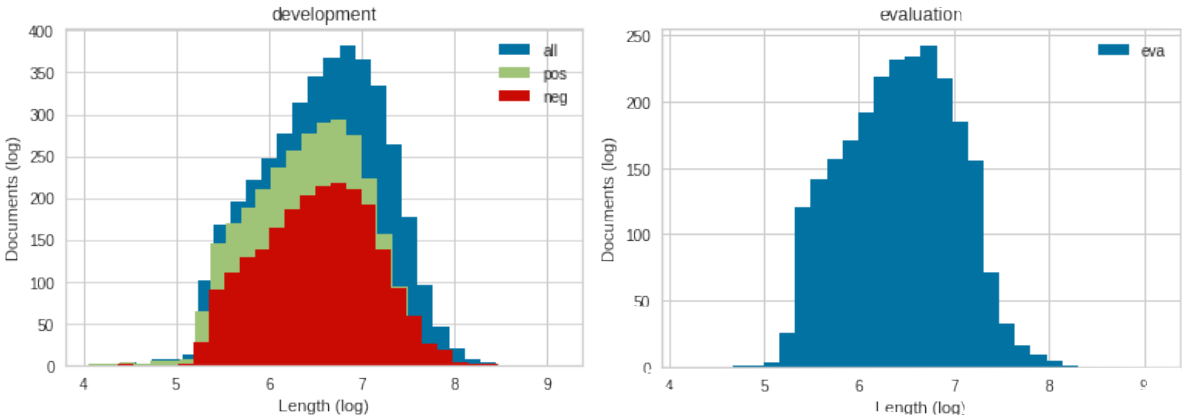
Data Science Lab: Process and methods
Politecnico di Torino

Project report
Student ID: s267475

Exam session: Winter 2020

1. Data exploration (max. 400 words)

The task's goal is to perform a sentiment analysis by mean of classification techniques on a dataset of tripadvisor's reviews.
Particularly, the dataset source is located at http://dbdmg.polito.it/wordpress/wp-content/uploads/2020/01/dataset_winter_2020.zip
and contains selected reviews of tripadvisor's italian website.
The given data is stored in two csv files containing 28'754 labeled documents and 12'323 unlabeled respectively.
Labeled data is not uniformly distributed (19'532 positive and 9'222 negative - ratio > 2:1).
As illustrated in the left image the documents' length distribution show the same trend for both classes and is very similar to the evaluation set (right image).



(The histograms consider the logarithm function for both x,y axes to show a more intuitive and visible shape)

```
development statistics: (pos)
max length: 7800
min length: 58
length avg: 625

development statistics: (neg)
max length: 9153
min length: 67
length avg: 864
```

```
general statistics: (development)
max length: 9153
min length: 58
length avg: 701

general statistics: (evaluation)
max length: 9221
min length: 65
length avg: 699
```

Given this informations we may assume that there is not any empty document in both sets, the evaluation set emulates the statistic of the development set, and that negative comments could be featured by an higher length average than positive ones.

By checking the encoding of documents it is possible to notice that some of them fail to be encoded in utf-8 because of emoticons (utf-16), and few of them present also idioms characters.

2. Preprocessing (max. 400 words)

Due to the nominal property of data, we must extract the interesting features from text to be able to process them.
In order to accomplish it I used sklearn's tfidf vectorizer with a personalized tokenizer where I managed also the stop words.
In the 2 solutions presented I managed to follow 2 different approaches, get most data as possible or get best data as possible.
The first approach aims to an extensive cover, getting best results, but it is liable of time-computation expensive cost.
The second aims to remove noise, poor variety of general best terms.
I used the same Vectorizer's parameters for both solution:

```
TfIdfVectorizer (input='content',encoding="utf-8",tokenizer=tokenizer,ngram_range = (1,3),max_df=0.90,min_df=0.0003)
```

As we may notice even if every document is about the same topic (so it would be better to set 'use_tfidf=False', I left 'use_idf = True' in order to try to balance the 'pos' in spite of the 'neg' documents frequencies. Otherwise if any 'pos' review contains a sentence like "Bell' Hotel" would probably get a doubled score compared to 'neg's containing "Brutto Hotel".
Ngram_range is set as (1,3) for extracting words relationships like ("Hotel Molto Bello") or ("Non Piacere Hotel").
Then I set max_df and min_df just to remove non filtered stopwords or very very rare terms. In the first case I needed common terms like "hotel" while in the second case I set 0.003 to remove "only 1 presence" words but not others rare terms in sight of the italian extensive range of ways to express an opinion.
The main differences between solutions are about tokenizer and Tfidf reduction:

Solution 1
The tokenizer is callable, and it is callen from Tfidf 1 time per document. Each document is treated by compacting repeated letters ('Bellissimoooo' → 'Belisimo') replacing emoticons with their meaning, by marking interrogative/esclamative sentences, by replacing everything it is not alphabetic or a space and then is tokenized using nlp of spaCy. Each token is parsed by spacy lemmatization, and then stemmed by SnawballStemmer, then gets filtered by length and by its presence in the stopwords' list (provided by nltk.stop_words("italian").

Solution 2
Same as solution 1 but stop_words list is augmented with a list of italian names.
The result is then processed by sklearn's TruncatedSVD and reduced to the 70 more representative features by 'algorithm=arpack'.

3. Algorithm choice (max. 400 words)

To train and improve the preprocessing part I used the Naive Bayes approach that is based on the statistical indipendence of features.
In particular I adopted the Multinomial Naive Bayes algorithm, because of it's better scores compared to Gaussian one (both provided by sklearn library, that specifies in its documentation that Multinomial approach suits well in case of text classification) and it has cheaper computational cost compared to Support Vector Machine (SVC fit time grows quadratically to the number of features).
When preprocessing tuning was stable I adopted the Support Vector Classifier of sklearn to get better results.
I implemented this strategy because MultinomialNB other than being fast, shows also almost a similar score trend of SVC while working on preprocessing. MNB works well even with just few and short documents (or snippets) while SVC needs lots of training data and suits better long documents. In this case documents lengths were generally medium-sized (compared to single sentences comments or by the other hand newspapers' articles) and train set was big enough so SVC appeared to be the most suitable one.
This supposition has then been empirically established by f1 weighted cross validation as you may notice in the figure above (Solution 1), but it is also true that their reliability and scores are comparables (0.01 f1 wwighted score difference on the training set)

```
model = MultinomialNB()
score_type = "f1_weighted"
cvs = cross_val_score(model,X_tfidf,datadev['class'],cv=5,scoring = score_type, n_jobs = 7)
print(f"{score_type} for each iteration:{cvs}")
print(f"{score_type} (statistics): {cvs.mean():.3f} (+/- {cvs.std() * 2:.3f})")
```

```
f1_weighted for each iteration:[0.96074725 0.95941106 0.9538043  0.95639121 0.95980215]
f1_weighted (statistics): 0.958 (+/- 0.005)
```

```
#model = MultinomialNB() #used just for fast testing lemmatizer and vectorizer -> svc too slow
model = SVC(kernel="rbf", C=2.2 , gamma=0.725 , class_weight='balanced')
score_type = "f1_weighted"
cvs = cross_val_score(model,X_tfidf,datadev['class'],cv=5,scoring = score_type, n_jobs = 7)
print(f"{score_type} for each iteration:{cvs}")
print(f"{score_type} (statistics): {cvs.mean():.3f} (+/- {cvs.std() * 2:.3f})")
```

```
f1_weighted for each iteration:[0.96913932 0.97098886 0.9642998  0.96747226 0.96614143]
f1_weighted (statistics): 0.968 (+/- 0.005)
```

For the truncated solution (Solution 2) I directly put on SVC because NaiveBayes is based on the statistical indipendence of features and so any approssimation that represents relationships would be nonsense, and we must also consider that due to the drastical reduction of data, SVC does not involve high time costs anymore.

4. Tuning and validation (max. 400 words)

As said before for tuning parameters of tokenizer and vectorizer I used the Multinomial Naives Bayes, tested throught a cross validation with 5 kfold to check mean and variance of result (given the 2:1 factor could be result overfitted).
Insted, abount tuning the classification algorithm used (Support Vector Classifier) I uppermost compared results of kernels type's (linear vs rbf), and weighted vs non weighted classes. Obviously due to the data distribution weighted was the right choice for the Solution1 while non weighted for the second, while choosing between linear or rbf meant choosing better performances or better results. I opted for the second possibility by leaving for both solutions default kernel = 'rbf'.

Kernel functions comparison:
linear = <x,x'>
rbf = e^(-gamma||x-x'||^2)

For the second solution I did not performed any hyperparameter tuning because I wanted to leave as more non-overfitting as possible.
Instead in solution 1 by using GridSearchCV tool I searched the best possibles combinations by an incremental focus step approach, on hyperparameters like C (it denotes the regularization's strength by an inverse ratio) and gamma (parameter of kernel) that must be greater than 0, otherwise it will diverge.
In the picture above it is possible to check the steps performed for the Solution 1 hyperparameters tuning described before.

```
# C10   gamma 1     - Cs = [0.01,0.1,1,10]                          gammas = [0.001,0.01,0.1,1]
# C8    gamma 0.8   - Cs = [8,9,10,11,12]                           gammas = [0.6,0.8,1,1.2,1.4]
# C5    gamma 0.7   - Cs = [5,6,7,8]                                gammas = [0.7,0,75,0.8,0.85,0.9]
# C2    gamma 0.73  - Cs = [2,3,4,5,5.5]                            gammas = [0.65,0.67,0.69,0.71,0.73]
# C2.4  gamma 0.72  - Cs = [1.4,1.6,1.8,2,2.2,2.4]                  gammas = [0.72,0.73,0.74]
# C2.2  gamma 0.725 - Cs = [2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9]   gammas = [0.725,0.73,0.735]
```

5. References

[1] Sklearn. MultinomialNB. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

[2] Sklearn. SVC. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

[3] Sklearn. SVM. URL:
https://scikit-learn.org/stable/modules/svm.html#svm-kernels

[4] NLP Stanford, MultinomialNB vs SVC. URL:
https://nlp.stanford.edu/pubs/sidaw12_simple_sentiment.pdf

[5] Sklearn. TFIDFVectorizer. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

[6] Sklearn. TruncatedSVD. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

[7] Sklearn. CrossValidation. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html

[8] Sklearn. GridSearchCV. URL:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[9] NLTK, STOPWORDS and SnowballStemmer. URL:
https://www.nltk.org/

[9] WORDCLOUD, Wordclouds. URL:
https://amueller.github.io/word_cloud/

general libraries:

- numpy
- matplotlib
- pandas

External tools and motivations:

[1] SPACY, Italian nlp for Tokenization and Lemmatization. URL: https://spacy.io/models

I used this library instead of NLTK presented WordNeyLemmatizer because it language is not supported, it is for the stemmer used but alone was not enough. So I needed to tokenize and lemmatize first with spacy and then stemming with snowball stemmer
I disabled from spacy nlp's pipeline tagger,textcat,ner,parser because were expensive and for example the tagger is still not enough precise to do a reasonable sentiment analysis (in italian) sent by sent.

[2] GITHUB mrblasco, public list of 32k Italian names. URL: https://raw.githubusercontent.com/mrblasco/genderNamesITA/master/gender_firstnames_ITA.csv
I need this repository to fetch italian names not well analyzed by nltk neither spacy, it contains also multiple names so I filtered them to get single word names, for further information see Solution2 import section (reduction to c.a. 9000)