Data Science Lab: Process and methods

Politecnico di Torino

Project report

Exam session: Winter 2020

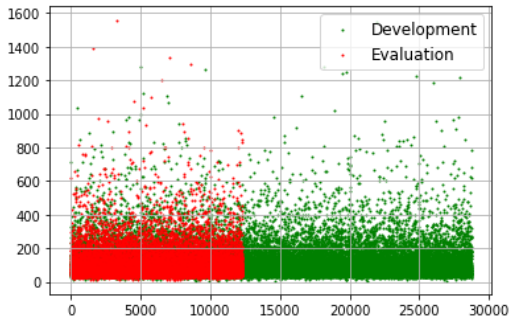
1. Data exploration (max. 400 words)

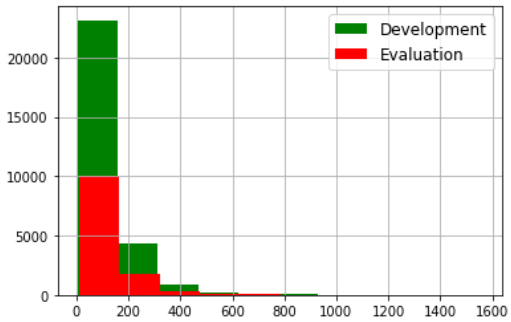
The first part of this data science process was exploring the data to obtain some knowledge from it. The main steps I computed were:

- Count how many reviews were inside each file: **28754** reviews for development and **12323** for evaluation.
- Look for data missing or outliers but there wasn't any.
- See how many different words were inside the development and evaluation files. What I got was: **153276** different words for development and **89558** for evaluation. So, I concluded that the size of the two sets was quite different.
- Check means and standard deviations of the review's length in both files and plot them.

	mean	**std**
development	113.55	100.67
evaluation	113.11	100.60

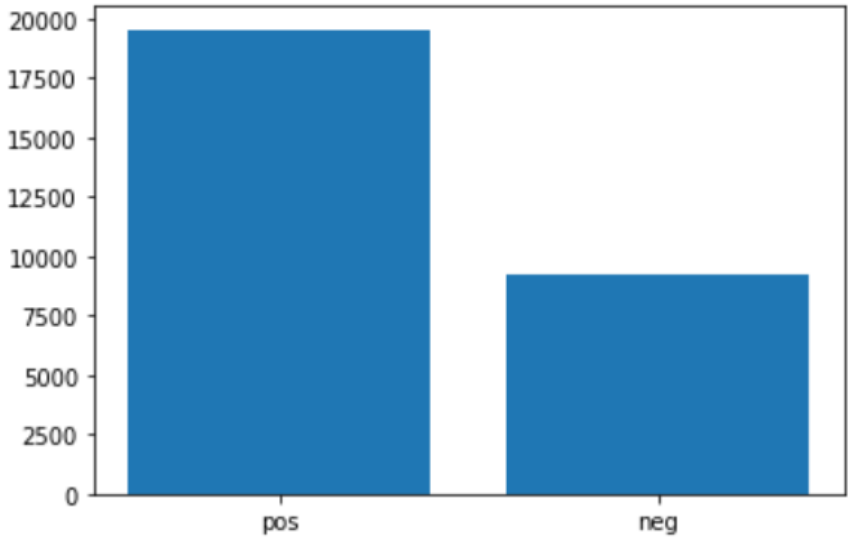
So, even if the two dataset sizes are different the review mean length is quite similar, this should be helpful.
Plotting them makes it clear that most of the sentences have less than 300 words, with an high concentration around 100;





- Find most frequent and meaningless words:
  Meaningless words like punctuation, symbols o numbers.
  The most frequent ones instead were most common in Italian vocabulary as: prepositions and articles.
  This helped a lot in the preprocessing part.

- Check if the 2 classes were balanced, I count the number of sentences with label '*pos*' and the ones with label '*neg*'
  I got: Positive: **19532,** Negative: **9222.**



This shows that the classes were quite unbalanced.

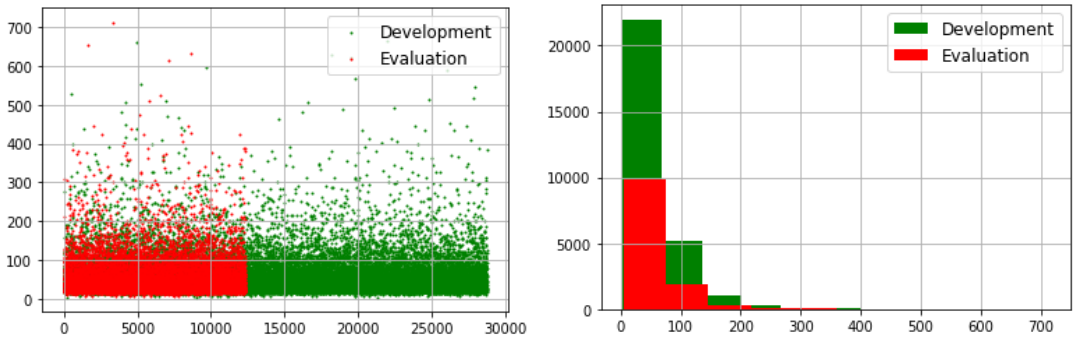
2. Preprocessing (max. 400 words)

The second part is to preprocess the data before feeding it to some machine learning algorithm. This step consists of:

- Apply case normalization: all to lower case.
- Remove from the sentences all the meaningless words, such as: numbers, punctuation, more frequent Italian stop words, words too long (more than 18 characters) or too small (less than 3/4 characters), some other words that are usually common in reviews such as '*hotel*', *'cibo'* and so on.
  I also used a lemmatizer from the **spacy** library and it worked well, so that I could remove similar words with the same root easily.
  This brings to: **32302** different words for development and **21522** for evaluation. It is almost 1/5 of the initial datasets.
- For solving the problem of unbalanced classes, I tried to find some negative Italian reviews repository or csv, but I found nothing. I tried to increment the number of negative reviews coping some of the existing ones, but this took to overfit; I tried also to balance the two classes by removing some of the positive ones, but this reduced the performances.
- Then I tried to introduce some positive words and negative words, also to balance classes, but it didn't improve the performance, so I kept the datasets as they were.
- After the cleaning process, the new values were:

	mean	**std**
development	55.13	47.46
evaluation	54.93	47.36

  The number of words in each sentence decreased to more less 50/60 words, so the cleaning process reduce the number of features to be stored and analyzed.



  From the graphs can be seen that now the number of words in each review is less than 150/200 with a high concentration around 50. It means there were lot of meaningless words and that this process helped.
- I used TfidfVectorizer to transform the reviews in a sparse matrix of tf-idf features. I also used min_df, max_df and ngrams.
- I tried to reduce the number of features with PCA or SVD but with high number of n_components they got stuck, with lower number of n_components the performances were bad.
- Finally, I split the dataset in training and test sets.

3. Algorithm choice (max. 400 words)

The third part is choosing a classification algorithm to classify the reviews to 'pos' or 'neg' reviews.

Since it is a supervised text classification problem and it as a lot of features (after the vectorizing the reviews) I thought to use some linear algorithm. The main algorithms, I thought of, were:

- **Support Vector Machine**: since works well with lot of features as in this case and since there are only 2 classes; divide them with a hyperplane should be simple and it should take to good results.
- **Naive Bayes:** since it classifies a review based on the probability of finding a word knowing that it has already found other word.
- **KNeighborsClassifier:** even if it works with distances from the different words and it may be difficult to calculate them.
- **RandomForestClassifier:** even if it needs some nodes to decide where to go in the next condition and it would be difficult to find them in a set which has so many features.
- **DecisionTreeClassifier:** I didn't try it since it would be difficult for it to classifier so many features and because there is RandomForestClassifier.
- **DeepNeuralNetwork:** the data set is not that big, anyway it is not linear.
- **SGDClassifier:** it works well with a high number of features.
- **LogisticRegression**: since it is a binary classification problem.
- I also tried the different algorithms, trained them, and calculate the score they gave.

Here a f1 score('weighted') comparison of the different algorithms with the same train and test sets with default parameters:

Classifier	f1 score('weighted')
LinearSVC	0.9670
SGDClassifier	0.9618
LogisticRegression	0.9530
MultinomialNB	0.9526
SVC (max_iter=1000)	0.9401
BernoulliNB	0.9362
GaussianNB	0.9266
RandomForestClassifier (n_est =100)	0.9076
KNeighborsClassifier (k=80)	0.8917

So I picked the ones that gave the best results: LinearSVC (SVC got stuck with too many iterations, even with kernel='linear') and SGDClassifier but also because they are linear, so the results are more human-readable they would be representable on a graph, they both use hyperplanes to divide data but the difference is on how they are trained (SGDClassifier with a Stochastic Gradient Descent(SGD) approach while LinearSVC use Soft Margin approach) and because they work well with sparse data, like in this case. Since they gave the similar results, I thought Support Vector Machine classifier was the right way to handle text data.

Other approaches like KNeighborsClassifier and RandomForestClassifier and other algorithms gave not good results, even with different hypermeters.

4. Tuning and validation (max. 400 words)

The fourth and last part is about tuning and validation, so the main object is to find the best hyperparameter and check if the results are coherent.

- About the tuning part I tried different settings both for preprocessing part and for the classifiers:
    - For the preprocessing part I saw the there was a lot of repetitions of the word 'volere' and other verbs, so I added them as stop words. I also added some English words that were common but meaningless for the classification.
    - For the Vectorizer I add min_df=20 and max_df=0.80, since they seem good values for the number of reviews and because after I tried, I get some improvements in the results. I used ngrams of 1 and 2 words since a sentence with the word '*bene*' could be classified as a positive review but with the word '*non*' before it has a different meaning. I tried also using ngrams with 3 words, but it didn't increase the score obtained and it slowed the training process, so I keep 2 as max ngrams length.
    - I divided the training and test set in 0.90, 0.10 respectively, 0.10 seams low but this gave a better performance since some submissions with higher values of the train ratio gave a higher score than the score on the test set, so the classifiers were under trained. I also tried to perform training using kfold but it brought to overfit.
    - For the classifiers I tried different combinations of values, in particular, for LinearCSV and SGDClassifier I tried:
        - For LinearCSV I tried different values for: loss, tollerance, max number of iterations, penalty and regularization parameter(C) and I saw that the best combination was: **loss='hinge', tol=1e-8, max_iter=100000, C=0.7, penalty='l2'**.
          Max f1 score got in the test set after tuning: **0.9696**.
        - For SGDClassifier I tried different values for: loss, max number of iterations, penalty, tolerance and validation fraction and I saw that the best combination was: **loss='hinge', penalty='l2', validation_fraction=0.1, max_iter=100000**, the tolerance value didn't change the score so I kept the default value.
          Max f1 score: **0.9668**.
- About the validation part I printed precision, recall and confusion matrix for both SGDClassifier and LinearCSV: there are some words bad classified, but they are not that much.
- Here a word cloud of the most common words found:


direttamente
complessivo
lussuoso
iniziale
esperienza
classe
molto
lindo
primo
mai
servizio
cordiale
efficiente
eccezionale
piccolo
approccio
sapientemente
impegnativo
limitato
scogliera
squadrare
facile
piuttosto
basso
curato
occhio
fiero
grande
sottostare
fuori
mezzo
caro
caldo
sotto
inferiore
minuto
piede
eccellere
spaziare
lettino
servire
asciugamano
continentale
niente
cibare
credere
andato
sapere
opzione
sottofondo
pensione
guardare
sacco
variare
pranzare
tangere
finale
bello
bevanda
tuttavia
decisamente
migliorare
problema
vistare
cambiamento
sommo
firmare
notte
quando
aspettare
potere
tanto
veduta
leggermente
totale
alcun
realtà
motivare
sperimentare
splendido
sicuro
dovere
contare
trattare
sostituzione
strutturare
edificio
buono
nuotare
mobilità
elegantemente
prevedere
settimana
superiora
adiacente
isolare
stesso
entrare
venire
cliente
preferire
difficile
dimensione
caratteristico
stare
fisico
occupare
famiglia
utilizzare
architettare
crociera
ottenere
aiutare
solere
bel
gustare
cristallino
proprio
senza
uscire
colorire
riposare
esigenza
connessione
stazione