Final NLU project: Sentiment Analysis for Movie Reviews

Matteo Brugnera 232670

University of Trento

matteo.brugnera@studenti.unitn.it

1. Introduction

Sentiment analysis research has become very popular in the past years especially thanks to the large amount of information available online.
With the advent of the social web, opinions have become really common and thanks to that, product review mining has become a really important area of research in *Natural Language Processing*.

Extracting opinions from a text is challenging and poses many problems that cannot be merely solved with lexicon-based approaches. These difficulties are caused by the *subjectivity* of a document being not so much conveyed by the sentiment-carrying words that people use, but rather by the way in which these words are used.

Sentiment analysis can be seen as a "suitcase problem" because it requires tackling many *NLP* sub-tasks, including:

- **text analysis**;
- **pre-processing**;
- **subjectivity detection and extraction**.

The objective of this work is to create a procedure that performs **subjectivity/objectivity detection** and also to implement three different machine learning algorithms that perform **polarity analysis** both at sentence-level and document-level.

2. Task Formalisation

This project can be divided into two main tasks:

- *Subjectivity/Objectivity detection*, checks whether a sentence has some subjective components within it or not.
- *Polarity classification*, checks whether a text has a negative or positive connotation.

The idea behind this project is to create a robust and reliable model that is able to classify in the best possible way the polarity of movie reviews.

For the first step, a *Multinomial Naive Bayes model* is used in order to detect whether a snippet is subjective or objective.

In the second task we have to check whether a review (in this particular case related to movies) is either a positive or negative review.
The first Multinomial NB model is used to eliminate all the objective phrases from the movie-reviews dataset. In this way we are able to work with a dataset containing only the subjective information that will be then used to estimate the polarity of the review.
Three different models will be used to obtain such result:

- a *fine-tuned Multinomial Naive Bayes*;
- a *RoBERTa model*;
- a *fine-tuned BERT model*.

3. Data Description & Analysis

As described before, movie reviews are being used both for subjectivity/objectivity detection and sentiment analysis.

In the first case, the dataset used is *Rotten_IMDB* and it is composed by:

- *quote.tok.gt9.5000* which contains 5000 subjective sentences obtained from some Rotten Tomatoes reviews and snippets;
- *plot.tok.gt9.5000* which contains 5000 objective sentences obtained from plot summaries and in depth descriptions of stories.

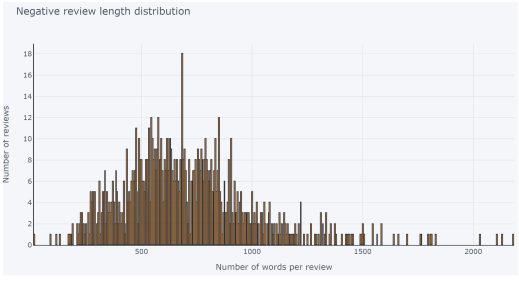
Thanks to this we are able to perform the extraction of the subjective sentences from the second dataset[2].Within this folder we can find 2000 processed down-cased text files.
The dataset is divided into two sub-directories (*"pos"* and *"neg"*) that indicate the true classification (sentiment) of each review belonging to it.

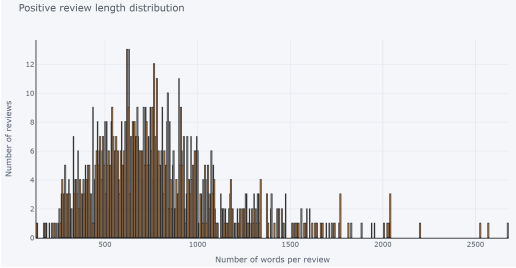
Some data analysis was executed for each review and useful information like *number of characters*, *number of words*, *number of unique words*, *number of sentences* and *number of stopwords* were computed.

num_of_chars	num_of_words	num_of_unique_words	num_of_sentence	num_of_stopwords
4043	825	353	35	364
1370	278	156	13	109
2848	547	274	23	217
2929	552	313	19	193
4418	841	380	37	332

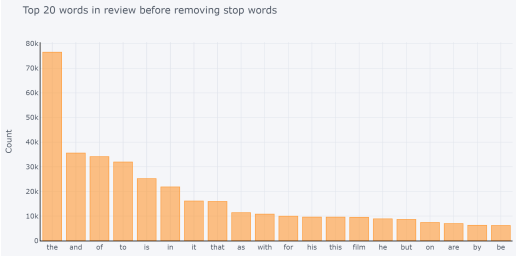
Also some useful statistics and graphs were obtained like:

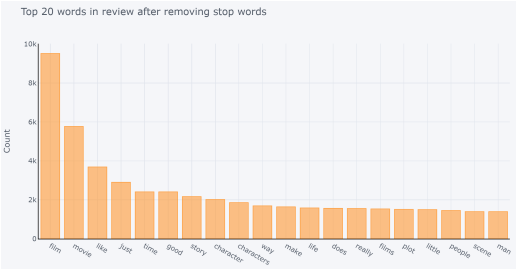
- the *length distribution* of the negative and positive reviews



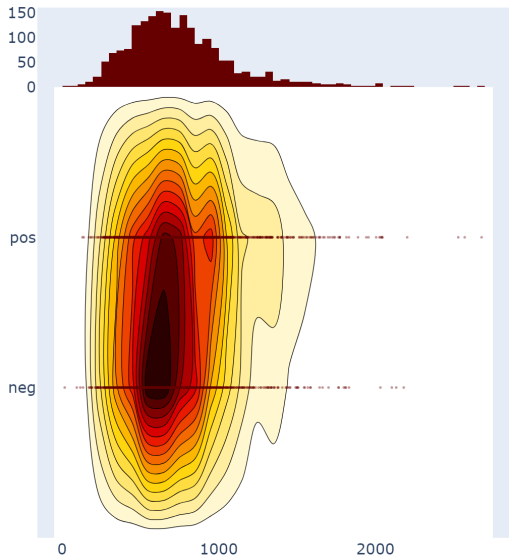


- the *top 20 words* in the movie-reviews dataset before and after removing the stopwords





It was also really important to check whether the *length of a review* affected its *polarity*. In fact, one can make the assumption that a negative review might be longer because, in general, criticizing a product makes you more invested towards it. So it is easier to produce more detailed description of what's wrong with respect to when you have to write something positive about a certain product.
In order to understand better this relation, a *2D density joint-plot* of sentiment polarity vs length was computed and the result obtained is the following:



4. Model

There are several ways to implement Sentiment Analysis and each data scientist has his/her own preferred method. I'll guide you through a quite intuitive (yet efficient) method so you can understand my ideas and reasoning behind the whole process (the results will be discussed in the next section).

First thing first, I computed a *baseline* of the overall problem without getting rid of the objective sentences from the movie reviews. Here we have to take into consideration two really important aspects related to how we deal with this particular type of data (this will also be really helpful especially for the future steps):

- *machine learning algorithms* are only able to understand and process *numeric data* (particularly floating point data). Thus we cannot feed them with plain text and wait for them to solve the problem. Instead, we have to make several transformations to our data until it reaches a representative numeric shape. This is done thanks to a particular procedure called **vectorization**.

  Since in most of the cases we will use a classic ML model (except for the *BERT* implementations), we will use the **CountVectorizer** method to convert a collection of text documents into a matrix of *token counts*.

- a really important aspect is related to how we *split* the dataset into *training and testing* before performing the vectorization.

  If we vectorize before splitting the dataset, our document-term matrix will contain *every single feature* (word) belonging to both the training and test sets. But this behavior is not realistic since, in a real world situation, it is common to have words that were not seen before.

  The right way of performing this procedure to properly simulate the real world is to do the train/test split before the vectorization. In this way the test data will most likely contain words that were not seen during the training phases.

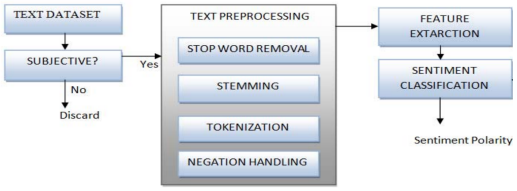
The model that was chosen for this task is a *Multinomial Naive Bayes*, a pretty common algorithm used in NLP.
This classifier uses *bayes theorem* to predict the probability that a given set of features is part of a particular label.

This model is a collection of many algorithms and they all share one common principle: *each feature being classified is not related to any other feature*.
As we can clearly understand this is a huge *drawback* because the meaning of a word/sentence may depend from words that were encountered previously.

$$P(label/features) = P(label) * P(features/label) / P(features)$$

The same reasoning was used to perform the *subjectivity/objectivity detection*. The only minor change applied was the use of **TF-IDF** for the vectorization step instead of CountVectorizer.
TF-IDF stands for *"term frequency - inverse document frequency"* and it's a numerical statistic that reflects how important a word is in a corpus. It combines the frequency of a *unigram* in a particular review document with its occurrence in the whole corpus.

In order to increase the accuracy of the classifier, all the objective sentences will be removed from the movie reviews dataset. Thanks to this, the dataset will contain only the subjective information that are needed to estimate the polarity of the review.



Furthermore, the following *pre-processing steps* were implemented to increase the accuracy of the overall model and to make the dataset more robust and consistent:

- **tokenization**. It is a way of separating a piece of text into smaller units called *tokens*. This tokens can be either words, characters or subwords.
  In this setting, we splitted the documents into single words. This will be helpful for all the next preprocessing steps.
- **Part-of-speech (POS) tagging**. Tagging is a very popular NLP process which refers to categorizing words in a corpus in correspondence with a particular *part of speech*.
  This type of information is one of the most promising among all features. In fact, it is commonly used in subjectivity detection. Moreover, a large number of researches reveal that *adjectives* are a quite good indicator of opinion/polarity of a text;
- **changes** related to the sintax of some verbs and to the pos-tag of some words in order to make the pos-tagging feature more efficient. Also words like *"looooooottt"* are replaced with its normal spelling *"lot"*;
- **removal of useless information**. In order to make the reviews more robust, plenty of useless data was removed, such as *special characters* like *", [,], (,)* and *punctuation*;

Also the following pre-processing procedures were tested but weren't implemented in the final pipeline because they didn't increase the overall accuracy:

- *stopword removal*;
- *lemmization*;
- *stemming*;
- *negation handling*;
- *spelling correction*.

At the end, two additional *BERT* (*Bidirectional Encoder Representations from Transformers*) models were implemented.

BERT is a transformer-based machine learning technique for NLP developed by Google.
It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model *can be fine-tuned with just one additional output layer* to create state-of-the-art models for a wide range of different tasks.

The first BERT model proposed is a **RoBERTa**: *A Robustly Optimized BERT Pretraining Approach*. It is build on BERT and modifies its key hyperparameters, removes the next-sentence pre-training objective and trains with much larger mini-batches and learning rates.
This model rates the sentiment of every single sentence with a score that goes from 1 to 5.
The second BERT model is the classic **bert-base-uncased**, a transformers model pre-trained on a large corpus of English data in a self-supervised fashion. This means it was pre-trained on the raw texts only, with no humans labelling them in any way.
In both cases, instead of using *CountVectorizer* to vectorize the data, a pre-trained tokenizer provided by the very same model is used. The data needs also to be formatted in a particular way by adding special special tokens like *[CLS], [SEP] and [PAD]*.

5. Evaluation

In this section, experimental setups and evaluation of the results of the proposed systems are presented. We will analyze every single metric that we used in the different models by also making a comparison between the baseline and the other results.

5.1. Baseline of sentiment analysis without getting rid of the objective sentences

The first results obtained take into consideration the whole dataset without applying any sort of pre-processing step or additional feature.
In this particular case I want to highlight the different results that we get if we split the dataset before and after performing the vectorization (as explained previously).

```
#Wrong approach
vectorizer = CountVectorizer()
classifier = MultinomialNB()

vectors = vectorizer.fit_transform(negative_positive_reviews)

# 10-fold cross-validation
scores = cross_validate(classifier, vectors, target_vector, cv=StratifiedKFold(n_splits=10) , scoring=['f1_micro'])
average = sum(scores['test_f1_micro'])/len(scores['test_f1_micro'])
print(round(average, 3))
```

```
0.814
```

```
#Correct approach
X_train, X_test, y_train, y_test = train_test_split(negative_positive_reviews, target_vector, random_state=1)

vectorizer = CountVectorizer()
classifier = MultinomialNB()

train_elements = vectorizer.fit_transform(X_train).toarray() #both fit and transform
test_elements = vectorizer.transform(X_test).toarray() #only transform

classifier.fit(train_elements, y_train) #train the classifier
predictions = classifier.predict(test_elements) #make predictions

report = classification_report(y_test, predictions, target_names=["neg", "pos"])
print(report)
```

```
              precision    recall  f1-score   support

         neg       0.78      0.80      0.79       255
         pos       0.79      0.77      0.78       245

    accuracy                           0.78       500
   macro avg       0.78      0.78      0.78       500
weighted avg       0.78      0.78      0.78       500
```

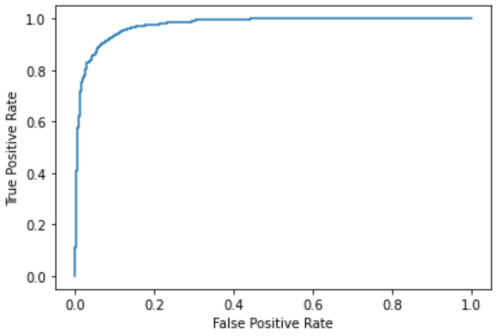
As we can see from this two images, if we apply the wrong approach we gain more than 3% points in accuracy.

We obtain also the AUC value equal to 0.8675 which represents how much the model is capable of distinguishing between classes. The higher the AUC, the better the model is at predicting subjective classes as subjective and objective classes as objective. In this case the value is between 0.8 and 0.9 so it can be considered a good result.

5.2. Subjectivity/Objectivity baseline

In this particular case we computed the baseline for this task by using a simple Multinomial Naive Bayes without adding extra features other than the usual vectorized train elements.
The accuracy achieved by this simple model is really good:

	precision	recall	f1-score	support
Subjective	0.92	0.92	0.92	1229
Objective	0.92	0.92	0.92	1271
accuracy			0.92	2500
macro avg	0.92	0.92	0.92	2500
weighted avg	0.92	0.92	0.92	2500

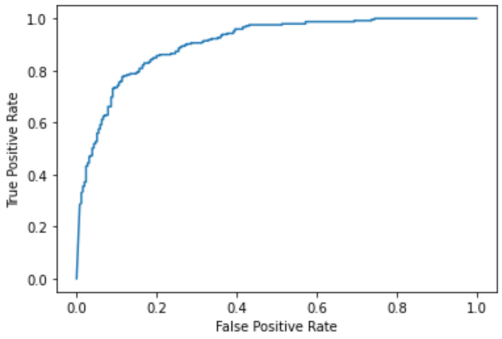


The AUC value that we obtain in this case can be considered almost perfect because it is equal to 0.9759. In fact, the ROC curve shown above describes almost the ideal situation: the two classes have a really small overlap which leads them to be well separated and easily distinguishable.

5.3. Second baseline: elimination of objective sentences

In this particular case a second baseline is computed as we did in the first case but considering only the subjective sentences. As we can see from the image below, the overall accuracy of the model increases by 5% because the model is handling less information with respect to the previous case but the quality of the data that we have increased, leading so to a better result.

	precision	recall	f1-score	support
neg	0.83	0.84	0.83	255
pos	0.83	0.82	0.82	245
accuracy			0.83	500
macro avg	0.83	0.83	0.83	500
weighted avg	0.83	0.83	0.83	500



AUC value: 0.9057062825130052

5.4. Final results of the system after pre-processing and fine-tuning

Now we take a look at how the pre-processing steps plus the introduction of extra features, change the accuracy of the final Multinomial NB model.
The introduction of the ngrams as an extra feature is really important since it overcomes the problem introduced by the Multinomial NB model that we discussed previously. By correlating multiple words together (in this particular case we have unigrams, bigrams and trigrams), we are able to correlate the meaning of a word with others.
We also ignore terms that have a document frequency strictly higher then 450 and strictly lower than 2. In this way we are able to eliminate a lot of useless information.

As far as the pre-processing steps are concerned, the model will only apply the ones that were listed in the previous section because the others just lower the overall accuracy. Moreover, a feature optimization procedure was implemented thanks to the use of chi2. In this way we were able to lower the amount of feature to ”just” 20000.

	precision	recall	f1-score	support
neg	0.91	0.90	0.90	248
pos	0.90	0.91	0.91	252
accuracy			0.91	500
macro avg	0.91	0.91	0.91	500
weighted avg	0.91	0.91	0.91	500

5.5. Results with RoBERTa and fine-tuned BERT model

BERT models can only handle text with length below 512 characters. In order to overcome this particular problem, I divided every single review in the dataset into sentences.
In this way the RoBERTa model is able to compute the sentiment score related to each sentence (goes from 1 to 5, as seen previously). We sum all the scores that we obtain from a single review and then we average it over the total amount of sentences that we considered. If the final score obtained is above 2, the review is considered positive, otherwise negative.

	precision	recall	f1-score	support
neg	0.82	0.92	0.86	1000
pos	0.91	0.80	0.85	1000
accuracy			0.86	2000
macro avg	0.86	0.86	0.86	2000
weighted avg	0.86	0.86	0.86	2000

On the other hand, the results that we get with the BERT model by using Adam as optimizer and SparseCategorical-Crossentropy as loss-function ranges between 89% and 92% (depends on how the BERT model is initialized).
It is also really important to notice that after two epochs the model overfits, since the accuracy obtained in the test set is lower with respect to the training set. The main drawback of both this models is the amount of time needed to train, even if the amount of data that we are handling is not large.

6. Conclusion & Feature Scope

This paper presents a survey of sentiment analysis and classification algorithms.
As we can see from the results that we obtained from the various models, the one that seems the most suitable for this task is the fine-tuned Multinomial Naive Bayes.
The pre-processing steps were really useful in order to increase the accuracy of 8%, especially adding as extra features the *n-grams* and *pos-tags*.
As far as the two *BERT models* are concerned, even if the results that we got are acceptable, the fact that it takes too long to train (around 1 hour per epoch) even with a small set of data, makes this models quite difficult to use for real life applications.

However, it is worthy to investigate for further possible pre-processing options in order to find the optimal settings.
Regarding the Machine Learning model selection strategies, a deeper study can focus on the choice of the best algorithm to assess this particular task. In fact, it could be really interesting to try different approaches, especially the ones that exploits *deep neural networks*, and making comparison between the different structures and pre-processing strategies.

7. References