Sentiment Analysis 2-step classification

Luca Dal Zotto (223714)
luca.dalzotto-1@studenti.unitn.it

University of Trento Via Sommarive, 9, 38123 Povo,Trento TN

Abstract

The task of Sentiment Analysis, usually applied to context such as reviews of a given product, consists in giving a polarity score for a given text or sentence, where such score will summarize the whole input into a positive or negative prediction for it. But for this project this task was extended to also initially do a pre-processing of the training data to remove the objective sentences of a movie review, and this was done by using the TextBlob library to find all those sentences to be removed.

1 Introduction

Now that we are in the age where information is abundant and going through most document could be either impossible or even not useful, tasks like sentiment analysis where from a given text we have get a simple score for how much positive or negative such text is could prove to be very helpful in many contexts. But to build a better model removing all the factual information has been proved to give better results in this field. In fact, subjectivity detection is considered one of the most important sub-task of sentiment analysis because when judging a given review to be interpreted as positive or negative, things like facts that do not contribute to the actual opinion of the user is not useful in the prediction, so if we were to remove such type of sentences we should improve the accuracy of our model, so that it can focus on subjective opinions.

My implementation of this problem uses TextBlob, a library that processes textual data to do many of the tasks related to NLU and sentiment analysis, to find if a given sentence is subjective or objective and, using a threshold value, decide if this sentence can be left as data or simply be discarded since it probably won't help in the prediction. At the end of this step the remaining data is used along a classifier to evaluate the resulting model.

This report is structured like this: the next section will cover the problem more formally, then an analysis regarding the actual dataset will be made along side the descripton of the models used for the task, and lastly a discussion about the results will be made.

2 Problem Statement

The problem itself is pretty straightforward, after downloading the dataset of the movie reviews and divide it by positive and negative reviews, I used the TextBlob library to find all those sentences inside the reviews that are considered objective. This was done by feeding the to TextBlob a given sentence to return a "blob" object that will not only have the text itself but also scores for polarity and subjectivity/objectivity. To be more precise about this last one, which is the only one that will be used, it will be a value that goes from 0, meaning that the sentence is totally objective, and 1, which in turn will mean totally subjective. So if a given sentence has such score above a certain threshold value, default value for the project of 0.4 for reason that will be discussed in Section 6, than it will be appended to the "new review" that will be then added to the updated dataset if such review is formed by at least one sentence. All of the sentences that do not fit such

criteria won't be considered in the classification.

Now that the first part of this 2-step classification is done, using the scikit-learn library the dataset is vectorized since it offers easy-to-use tools to perform both tokenization and feature extraction of text data. Then a numpy array is created to link all of the reviews from the dataset that are negative and assign a score of zero, and the ones that are considered positive as one. After deciding which classifier to use, the F1 score is computed by employing cross-validation on the training dataset. The F1 score is used to measure a model accuracy from the precision and recall of the test, where the precision is the number of true positive results divided by the number of all positive results and the recall is the number of true positive results divided by the number of all samples that should have been identified as positive. Cross-validation on the other end is a way to actually compute the accuracy of a model, and this is done by dividing the dataset by a number of fold, in this case 10, and iteratively use one of such folds as validation while all of the others will be used to train the model, than at the next iteration the batch of validation and test data will change. This operation is repeated until all of the folds are used as validation set.

3 Data Analysis

For this project we were tasked to use the IMBD movie review dataset by Bo Pang and Lillian Lee, which collects 1000 positive and 1000 negative reviews all coming from the imbd.com website which were released in the 2002. These reviews are from a total of 312 authors but only a maximum of 20 reviews were used per author. The reviews themselves are all in English, and some modifications were done to improve the processing. To be more precise:

1. all the text has been converted in lowercase,
2. addition of white space around punctuation's like commas, periods and brackets,
3. text was split into one sentence per line.

After downloading the dataset from the NLTK library we have two data structures, one for positive and the other for negative reviews, that are lists containing the reviews, then each review will contain another list dividing these sentences, and then inside each sentence another list for the tokens.

4 Models

In this section I will describe the different model settings that I tried to see where I was able to find the best possible performances. To be more precise I will start from describing the two types of vectorization of the dataset used (CountVectorizer and TfidfVectorizer) then an in-depth description of how I used TextBlob to remove the objective sentences and lastly the three classifiers used to evaluate the Sentiment Analysis task.

4.1 Text Encoding

As previously mentioned to do the encoding I first started using the CountVectorizer, which provides a simple way to tokenize a collection of text and build a vocabulary of known word. Not only that but we can also use it to encode a new set of documents using that same vocabulary. Moreover I want to point out that traditional stop word removal was not used because certain stop words (e.g. negating words) are indicative of sentiment [1].

It works as follows:

1. create an instance of the CountVectorizer class,
2. call the *fit()* function to learn a vocabulary one or more documents,
3. call the *transform()* function to encode one or more documents as a vector,
4. or simply call the *fit_transform()* function which is used to first do the *fit()* and the *transform()* on the same document.

An encoded vector is returned with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. Since this vector will contain a lot of zeros it is usually called sparse.

Basically CountVectorizer does simple word count, which is a good starting point but pretty basic. In fact we can occur in problems with words like "the" that appear many times, making their counts not very meaningful in the encoded vectors. An alternative technique is to calculate word frequencies using the method called TF-IDF, which stands for "Term Frequency - Inverse Document". It not only is a statistic based on the frequency of a word in the corpus, but it also provides a numerical representation of how important a word is for the analysis. TF-IDF is considered better than CountVectorizer because it not only focuses on the frequency of words present in the corpus but also provides the importance of the words. We can then remove the words that are less important for analysis, hence making the model building less complex by reducing the input dimensions. The way in which TF-IDF is computed for term i in document j is as follows:

$$w_{i,j} = tf_{i,j} \times \log(N/df_i)$$

Where $tf_{i,j}$ is the number of occurences of i in j, df_i is he number of documents containing i and N the total number of documents. Basically the TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents.

The *fit()* and *transform()* works the same as with the CountVectorizer.

4.2 Objectivity removal

To do the removal of sentences which are objective I used the TextBlob library that was build on top of NLTK and, as of now, is one of the most used libraries used for any NLP problem since with it you can accomplish tasks like tokenization, text classification, spell checking and more with just a few lines of code. But for this projects I only used it to get the polarity score of objectivity/subjectivity of a given sentence and if such value was above a threshold then the sentence would be kept. This is done because the way in which TextBlob computes this score is that from a TextBlob object it is possible to have two type of *sentiment* values, the first one is the "usual" sentiment polarity score, while the other is the one related to the objectivity/subjectivity of the sentence. The value is an integer from range 0 to 1, where 0 means that the sentence is totally objective and 1 that it is totally subjective. This is the reason why in my implementation only the sentences that had a score above a threshold could be passed to the next step, since they are the one deemed to be too objective.

Everything else is pretty standard, the input of the function (together with the threshold value) is one of the two positive or negative dataset directly coming from the NLTK library. This means that the some type of pre-processing has to be made, since each sentence is divided word by word, but with a *for* loop this is not a problem at all.

So after the review is reconstructed as a whole chunk of text to be vectorized, it will be stored in a list that will contain all of the updated reviews, but this only if the review has at least one or more sentences, since it is possible that some reviews have all sentences that are deemed too objective (especially if the threshold value is above 0.5, see Table 2 and 3 for more details), meaning that no sentence is saved from the process. Also in the code are present two variables to count the number of sentences and the number of sentences removed, but of course this are just used for the statistical analysis.

This procedure is done for both the positive and negative reviews and not the dataset as a whole, since then the length of these new data structures will be used to compute the labels used for the training and evaluation of the model because of the problem of vanishing reviews.

4.3 Classifiers

I chose four type of classifier to do the evaluation on document level of the Sentiment Analysis, which all came from the sklearn library. Namely I used:

1. a Naive Bayas classifier using the *MultinomialNB()*,
2. a Linear Support Vector Machine (LSVM) using *SVC(kernel="linear", C=0.025)*,
3. a vanilla SVM using *SVC()*,

4. and lastly a Multi-layer Perceptron classifier using *MLPClassifier(alpha=1, max-iter=1000)*.

5 Experiments

Like I stated and discussed on the previous sections, many experiments were done trying to find which model was able to maximise the F1 score on the new dataset of reviews, done using a cross-validation with 10 folds every time. Lastly two tests were also done to find the best possible threshold value for the *objectivityRemoval* method, which was done on the best settings found from the previous test, to be discussed in the next section. The best value was then set as default for the whole project.

The code used for this project can be found in this repository publicly avaliable on Github. The scripts were written in the Python language, with the use of different libraries like: nltk, numpy, sklearn and TextBlob. All of the tests were done on the GoogleColab platform for faster processing and (as of the last check) was runned over an Nvidia Tesla T4 of 16GB.

6 Results

As we can see from the results in Table 1, the best accuracy was achieved by the SVM using the TfidfVectorizer, but since it was very close to the one of MLP (which has to be said doesn't give constants results) I wanted to find of the two which of them could perform better using different thershold values. So the next two tests were done using first the SVM for different threshold values (results in Table 2), and then another one with the MLP (Table 3). Both of them used the TfidfVectorizer since it previously lead to the best results. The results from Table 2,3 do not report all of the used threshold values since it would be impossible to read and use too much space, so only the most meaningful are reported. Also the percentages of sentences that were removed along side the number of reviews removed if present can be found in those tables.

	Before Removal		After Removal	
	CV	TF-IDF	CV	TF-IDF
Bayes	0.814	0.808	0.824	0.834
Linear SVM	0.835	0.752	0.826	0.778
SVM	0.764	0.841	0.771	0.855
MLP	0.838	0.849	0.832	0.853

Table 1: F1 score for models before and after the removal of the objective sentences. CV stands for the CountVectorizer while TF-IDF for the TfidfVectorizer. All these results were found using 0.4 as threshold value for the removal

So after finding the value for the threshold which maximized the F1 score (0.3, even if for the SVM it actually decreases for it) I wanted to see if this number could improve the performances also for the other combinations of vectorizer and classifier. All the results can be found in Table 4. From it it is possible to see that for most of the models there was no improvement, so the default threshold value was kept at 0.4.

Taking a better look at Table 1 (also applies for Table 4), we can spot a recurring theme that is the fact that a Linerar SVM always performs better than the "vanilla" one with the CountVectorizer, while on the other end with the TfidfVectorizer it is the opposite for the two classifiers.

In conclusion it seems that using objectivity removal usually bring to an accuracy improvement, which most of the times occurs when the CountVectorizer is used. Another thing to note (from Table 2 and 3) is the drop in accuracy specifically when the number of reviews was decreased. This could be due to the fact that at the end of the day we are making a small but meaningful drop in the number of training samples, since this could also mean that the remaining reviews may not have many sentences themself.

Like I said previously, the best classifier was the MLP, which is to be expected since it is much more powerful than the other tpe of classifier. That said it is also the one that takes the most amount of time, taking around 10 to 20 minutes for each evaluation. Also its results were not consistent even if the same settings were kept, this is because solving a problem using a Multi Layer Percepron is non-convex, mean-

	0.2	0.24	0.28	0.30	0.32	0.36	0.38	0.4	0.42	0.44	0.48	0.50	0.56	0.62
F1	0.844	0.844	0.843	0.846	0.843	0.845	0.838	0.855	0.85	0.846	0.843	0.833	0.812	0.793
% sent. removed	32.13	33.28	35.91	38.71	39.52	43.27	45.34	48.98	50.36	52.31	56.66	63.37	69.39	76.10
# removed reviews	0	0	0	0	0	0	0	0	0	0	0	3	6	19

Table 2: F1 score, percentage of removed sentences and reviews for each threshold value analysed using TF-IDF and SVM as classifier

	0.22	0.26	0.30	0.32	0.34	0.36	0.38	0.4	0.42	0.44	0.46	0.50	0.56	0.62
F1	0.844	0.852	0.862	0.86	0.855	0.853	0.845	0.856	0.854	0.852	0.848	0.836	0.819	0.793
% sent. removed	32.63	34.96	38.71	39.52	41.53	43.27	45.34	48.98	50.36	52.31	54.73	63.37	69.39	76.10
# removed reviews	0	0	0	0	0	0	0	0	0	0	0	3	6	19

Table 3: F1 score, percentage of removed sentences and reviews for each threshold value analysed using TF-IDF and MLP as classifier

	Threshold value = 0.4		Threshold value = 0.3	
	CV	TF-IDF	CV	TF-IDF
Bayes	0.824	0.834	0.823	0.833
Linear SVM	0.826	0.778	0.82	0.782
SVM	0.771	0.855	0.768	0.846
MLP	0.832	0.853	0.823	0.862

Table 4: F1 score for models after the removal of the objective sentences. All these results were found using XYZ as threshold value for the removal since it was the best one found from Table 2

ing that the algorithm could end up in a local minima rather than in a preferable global minima. Nevertheless the results shown by the MLP are still the best overall.

References

[1] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts *Learning Word Vectors for Sentiment Analysis*. Addison-Wesley, Reading, Massachusetts, 1993.