CS 584 – SENTIMENT ANALYSIS OF PRODUCT REVIEWS REPORT

Username on miner: **Spydyyy** Rank on leaderboard: **69**

Accuracy score: **0.90**

Problem Statement:

To design a classifier for predicting sentiment values (+1, -1) for Amazon baby product reviews.

Methodology:

Bag of words and n-grams representations without TF-IDF Weighting.

Steps followed for developing the classifier:

Step 1: Data preprocessing or data cleaning: At first, the data must be noise-free for being able to process it uniformly throughout the document. Hence the first step was extracting all the textual reviews, converting all alphabets to lowercase, search for patterns with regular expression and remove all special characters and non ascii values to normalize all records.

Step 2: Transformation of text representation to fixed-length vectors: The raw form of text can't be used for processing just as it is since the processors can't understand the complex structure of textual data. So, they must be converted into numeric forms or binaries, which will easily allow the machines to process the given inputs. Hence This step is necessary for any model to train and learn from a given set.

I have used CountVectorizer () class here from scikit-learn which is a machine learning library available in python to implement bag of words and n-grams. CountVectorizer () is used to transform a given text into a vector based on the frequency of each word in a record. Fitting and transforming a text document using CountVectorizer () would return a (Compressed Sparse Row) CSR matrix format which will contain review records as rows and vocabulary of words in the entire document as columns.

Fit () method is used to calculate the mean and standard deviation of the set and transform () method is used to transform all the data points depending on the calculations of fit (). While these can be used separately, I have used it together as fit_transform () that would perform both these functions at one time and in-turn increase efficiency.

While trying to transform data efficiently, I just played around with the set using some attributes of count vectorizer. I was able to find that, generally while using the count vectorizer to count the frequency of words, it takes more time to frame the vocabulary from the entire document, count number of occurrences and transform into matrix finally. So, I tried to use binary values instead of counting the occurrences, i.e., if a word is present in a review, its vocab value is 1, else it's 0, regardless of the frequency of occurrences of that word in an entire text record.

I knew that this would eliminate the importance of a word in the whole text. So, I tried both the ways and checked for the accuracy, and it was approximately the same. It might be different elsewhere, but I think this is appropriate for this model. Hence, I executed it with the binary=True attribute that would not count the number of occurrences, to increase the overall processing speed.

n-grams methodology:

I have used n-grams along with CountVectorizer () to take unigrams as minimum and bigrams as maximum. This would add one word and two-word combinations of the training set to the vocabulary which is a highly effective way for the model to learn the features of a given training set.

Also tried experimenting with only unigrams, only bigrams and unigrams to trigrams combinations. In the case of only unigrams, the model would develop only some partial features from the training set and would leave out combinations of words used, which would result in less accuracy. While using only bigrams, it may leave out many single words that may be of high importance in the whole text. So that too wouldn't be appropriate. In the case of unigrams to trigrams, it will have high accuracy since the model would learn the meaning behind the combinations of three words together. But the reason why I didn't go for it is because it would develop a very huge vocabulary set that takes a lot of time for processing and a large amount of memory. Hence unigrams and bigrams would be an appropriate fit for this model.

Step 3: Splitting the training set into two subsets and perform Logistic Regression:

I have used LogisticRegression, train_test_split and accuracy_score classes from sklearn library to split up the training set, perform logistic regression and check for accuracy. Train_test_split splits the training set itself into two subsets – one part for training and one part for validation. It takes these two as arguments including train_size that is used to mention the size of training subset. It returns 4 values – x_train & x_val i.e., 75% of training set and the values (+1, -1) for it and – y_train & y_val i.e., 25% of the training set (validation set) and the values for it. The LogisticRegression () method takes in a c value as argument that mentions the amount of regularization. As we increase the value of regularization, we could see that the accuracy of the model increases along with it.

Step 4: Predicting the test set using the trained model:

We can now predict the sentiment values of the test set using the predict () method of logistic regression.

Conclusion:

Therefore, we were able to successfully design a classifier to train a model using bag of words and n-grams and achieve a good accuracy score of 0.90.