Sentiments Analysis Using Logical Regression

Name: Rahul Sharma

Username: rsharm8

These steps mentioned below is the approach I used to develop the classifier for predicting the product review sentiments.

Step 1: Read the given files into dataframe using basically two functions “read_table” and “read_csv”. As their name suggests one is used to read from tabular structure and other is used to read from csv format file. The purpose of using two different functions is that the training file is in the tabular structure whereas the test data file is fully text file containing strings which result in missing few rows after reading through read_table. Then there is a need for read_csv functions that read the file successfully with the same number of rows.

Step 2: Converted the data values which is in string format to lowercase first and then used regex to remove any numbers and special characters from both the data acquired from training and test file.

Step 3: Performed “stemming” and removing “stopwords” from each individual data-frame of both files using nltk (Natural Language Tool Kit) library and all those modifications took in-place.

Stemming is the process of reducing inflection in words to their root forms (e.g.- going converts to go)

A **stopword** is a commonly used word (such as “the”, “a”, “an”, “in”)

Step 4: Applied TF-IDF vectorizer on training dataset. Let’s understand what TF-IDF is.

Term Frequency: This measures the frequency of a word in a document. But we perform normalization on the frequency value, we divide the frequency with the total number of words in the document. Final value of the normalised TF value will be in the range of [0 to 1]. 0, 1 inclusive.

TF is individual to each document and word; hence we can formulate TF as follows:

Tf (t, d) = count of t in d / number of words in d

Document Frequency: This measures the importance of documents in a whole set of the corpus. This is very similar to TF, but the only difference is that TF is the frequency counter for a term t in document d, whereas DF is the count of occurrences of term t in the document set N. In other words, DF is the number of documents in which the word is present.

df(t) = occurrence of t in N documents

To keep this also in a range, we normalize by dividing by the total number of documents. Our main goal is to know the informativeness of a term, and DF is the exact inverse of it. that is why we inverse the DF

Inverse Document Frequency: IDF is the inverse of the document frequency which measures the informativeness of term t.

idf(t) = N/df

Now there are few other problems with the IDF, when we have a large corpus size say N=10000, the IDF value explodes. So, to dampen the effect we take the log of IDF. In few cases, we use a fixed vocabulary, and few words of the vocab might be absent in the document, in such cases, the df will be 0. As we cannot divide by 0, we smoothen the value by adding 1 to the denominator.

idf(t) = log(N/ (df + 1))

Finally, by taking a multiplicative value of TF and IDF, we get the TF-IDF score.

tf-idf(t, d) = tf(t, d) * log(N/(df + 1))

The function which I used would do the vectorization along with TF-IDF in one statement. After which I do the transform on the test data on the basis of given vocabulary formulated using TF-IDF.

Step 5: In this I used logistic regression model from `sklearn` library in which I passed training data X and its label Y to train the model using fit method.

```
log_reg = LogisticRegression(max_iter=20000)

log_reg.fit(X, Y)
```

Here X is the transformed vocabulary from training data in vectorized form in array format and Y is the label defining {+1/-1} against each row.

Step 6: Finally, after training the model on given data. Now the time is to predict on our test data which we already transformed and passed as an argument in predict function which later print the labels in a file called output.

```
prediction = log_reg.predict(x_test)
```

Accuracy: 0.87

Rank: 87

Disclaimer: I tried Bag of words approach as well in which I got 0.86 accuracy.