Name : Divya Khiani

SJSU ID : 015273245

Report On Movie Review Classification

Rank : 37

Accuracy : 75.98%

Approach:

Step 1

Reading the train.dat file line by line and splitting the response (+1 and -1) from rest of the review. The response is treated as the label for data and stored in X and Y accordingly.

Step 2

Applying TFIDF Vectorizer on the reviews part of data to convert the sentences into a matrix of TF-IDF features

Step 3

Applying Dimensionality Reduction using truncated SVD on the TF-IDF matrix with n_components=100

Step 4

Training the KNN model on the reviews of train.dat file with algorithm=kd_tree and k=300

Step 5

Repeating Step 2 and 3 on test.dat file

Step 6

Using the matrix generated from step 5 to predict the results

Methodology :

After splitting the data into review and label part. I implemented TF-IDF vectorizer on the text part to convert it into a inverse term frequency matrix.

TF-IDF vectorizer was used to diminish the impact of frequently occurring words. Also, the stop_words parameter was set to be 'english' such that the TF-IDF matrix is generated after removing stop words from the text.

After performing TF-IDF vectorization, the matrix we get is approximately of size 25000*76k which is very huge to deal with during computation. Hence, to reduce the dimensionality of the matrix I applied truncatedSVD.

truncatedSVD reduces the dimension of the matrix to 25000*100 which will be easy to handle also it works best with tfidf matrices. Since truncatedSVD is efficient in handling sparse matrices its been used here.

KNN Classifier model is build with appropriate parameters viz. Distance=weights, algorithm=kd_tree and neighbors=300 after trying multiple combinations of parameters I found these to be most accurate.

Using weights as distance metric for KNN model result in giving more weightage to the closer neighbors than the far ones.

The kd_tree algorithm requires a dense matrix to perform KNN, this was also the reason why I performed Dimensionality Reduction.

Notings:

The TFIDF vectorizer is equivalent to CountVectorizer followed by TfidfTransformer.The **TFIDF vectorizer** outputs a sparse matrix with the TFIDF score of each word in every document. The output in the form of (A,B) = C represent the tfidf score(C) for word B in document A, where A is document index and B is word-vector index. It is calculated using the formula tf-idf(t, d) = tf(t, d) * idf(t) and idf(t) = log [1+n / df(t)] where n is total number of documents. 1 is added to the numerator and denominator as if one extra document was seen carrying every word of the built vocabulary. This is done to prevent zero divisions.

TFIDF vectorizer states how important that word is to that document whereas Count Vectorizer gives equal weightage to every word in a document. It will result in biased results if a word is repeated many times in the document but it is of no use, mostly the most important words are used fewer times in a document. Hence using TFIDF gives better result than Count Vectorizer.

TruncatedSVD is similar to PCA but it does not center the data before computing SVD. Also, in normal SVD for m*n matrix the result will have m columns whereas truncatedSVD will have specified number of columns which will result in faster computation and less space complexity. It drop off all least important features and will keep only specified number of most important features.

Using **KD Tree algorithm** (K dimensional tree) in KNN result in building a binary tree at each step. It is a space partitioning algorithm which organize points in K dimensional space. Each node in tree has K-d datapoint, the datapoints lying to the left of this point will be represented by left subtree. The tree is divided into two halves at each step using the median of data. Brute force is efficient when the dataset is small as the data increases KD tree algorithm becomes more efficient as well as fast in computation.

By using **weighted distances** we are giving more importance to the nearer datapoints as compared to the far ones, hence the output will be more dependent on the near datapoints. The weight assigned will be the inverse of the distance hence farther the datapoints less is the weight and less is its influence on the output.