Name: Adel Hamad Alkhamisy

Miner2 username: AA

Accuracy 0.77 on miner

The approach that I used is as follow:

Step 1: preprocessing part

Firstly, I analyzed test_file.txt and train_file to figure out unnecessary words and symbols that need to be deleted in the preprocessing step.

Secondly, I used read_csv from pandas library to read test_file.txt and train_file and load them to JupyterLab. I specified the separator to be the tab, so I separated the sentiment and the comment part of the train_file.txt.

Thirdly, I defined a function called preProcessing that utilizes Natural Language ToolKit library NLTK and Regular Expression library to delete symbols and stop words which are unnecessary words such as "myself, I, we, our, etc." This function receives a dataset, then it creates a loop that iterates over each review, and with the help of a regular expression (or RE) anything that does not match a character a-z or A-Z, I substituted it with a space. Then I converted all the characters to a lower case. After that, I splited each comment into words. then, I utilized PorterStemmer() to apply stemming to each word in the comment which converts each word to its root to avoid redundant words that mean the same thing. Next, I joined each word that belongs to this comment or review together but with a space between them to avoid concatenating the words without sapce. After that, I appended the comment to a dataframe corpus that represents the dataset. Next, the function returns the dataframe to its caller. Finally, I created a variable called sentiment and extract the review part which is "+1" or "-1" of the training data with the help of iloc function that allow us to specify a column and

return the data contained in that column and I called the preProcessing function and passed the train and text data and stored them in two variables called preprocessedReviewTrain and preprocessedReviewTest respectively.

Step 2: Create a sparse matrix

I used TfidfVectorizer from sklearn.feature_extraction.text to create the sparse matrix. TFIDF means term frequency – inverse term frequency which is a numerical metric that measures how important a word is to a document in a corpus or collection. With the help of TfidfVectorizer , I created the sparse matrics of preprocessedReviewTrain and preprocessedReviewTest which I named tfidfTrain and tfidfTest respectively. I used ngram_range=(2,5) and max_features=30000 which gave me the highest accuracy rate. I applied fit_transform() on the training dataset to get and transform the feature list and I applied transform() method on the test dataset which is very important technique in data mining because I do not want to get a new feature list from the test dataset.

Step 3: KNN

I passed the number of neighbor k, tfidfTest, tfidfTrain, , preprocessedReviewTest, and sentiment to knn. Then, I created a loop over the preprocessedReviewTest. After that, I calculated the cosine similarity of each instance of preprocessedReviewTest and all train sparse matrix “tfidfTrain” to find the most similar sentiment. Then I get the list of k nearest neighbor from the sentiment and stored them in nearestNeighborList. After that I converted nearestNeighborList from string type to int type to sum the score. Then, I sum the list of k neighbors. If the sum of the nearestNeighborList is >0, then this means that the +1 class is the majority. If the sum is zero, then the positive and negative are equal, but the program will classify it as positive. If the sum is negative, then the majority of the neighbors are "-1"

Step 4: I used KFold cross-validated to select the best k number which is 291 which gave me 0.79 accuracy rate.

The following table summarize the parameters tuning:

TfidfVectorizer	Distance Euclidean distance (U) Or Cosine similarity(CS)	No. Of Neighbor	Stem	N gram	SVD	Accuracy
40100	U	5	Yes		Yes	.59
Full	U	20	Yes		Yes	.59
Full	U	15	Yes		Yes	.52
100	U	100	Yes		No	.66
1000	U	15	Yes		No	.62
50	U	500	Yes		No	.65
1000	U	200	Yes		No	.74
30000	U	200	Yes	1-2	No	.75
30000	U	200	Yes	1-3	No	.76
30000	U	80	Yes	2-3	No	.77
30000	CS	55	Yes	2-3	No	.77
30000	CS	80	Yes	2-4	No	.77
30000	CS	236	Yes	2-5	No	.79