Rohan Kamat
013759252
Prof. David Anastasiu
CMPE-255

Medical Text Classification PR-1

Approach 1:There were 2 datasets, train.dat(14438) and test.dat(14442).The first approach was a naive one using trivial Data structures like list and dictionaries.The data was imported into a dataframe , the classes from the train.data were separated and stored in a list.**Preprocessing** of the data was done using a simple regex expression r'\w{4,}' filtering out all words with lengths lower than 4.The cosine similarity was calculated, and a list of tuples was created (dictionary of word count, class, similarity).Based on this the class was determined using a voting mechanism, and if all equal classes were present, the most similar was chosen. This method generated a very poor F1 score of **40%** and the time complexity deteriorated due to multiple functions having complexity of O(n^2).The value of **K** was chosen as **9**, and **epsilon** as **0.02** which provided highest F1 score.The code takes more than 2 hours to produce results.

Approach 2: To improve the F1-score further-on and to reduce the time complexity, as mentioned by the professor a more advanced data structure was used called CSR matrix.This considerable reduced time required for traversal.The preprocessing part was much more mature where first the first approach, nltk stopwords were used to get rid of all unnecessary words and then converted to lowercase.Lancaster Stemming was used to reduce words to their root.This was the preprocessing part for both the test data as well as the train data.The data was then split into lists of list having words , this was to enable the CSR matrix to use the dictionary of word counts.Then TF-IDF Term Frequency Inverse Document Frequency was used to only consider the relevant words,two vectors were created from the iteration through rows of both the CSR matrices(Test,Train) and cosine similarity was calculated, numerator was the dot product of these two vectors and denominator was the product of magnitude.A list called *final* was created which was a list of lists for each test vector compared to each

train vector using cosine similarity.A list called *Listoftuples* was created which contained a tuple of (classes,cosine similarity).A method was created called *getneighbours* which had parameters (list from the listoftuples and k),the function sorted the tuples within the list using key operator and itemgetter in a descending order and returned k neighbours.A function was created to set the value of epsilon called *setepsilon* it simply filtered the neighbours with cosine similarity lesser than the given epsilon, after applying the filter now I had neighbours in a list of tuples with cosine similarity values higher than epsilon.The next step was a simple voting mechanism which finally revealed the class of the the test profile.The method *classvote* took input parameters as neighbours, a variable called *response was* created which simple stored the classes, a dictionary called *classvote* was created having a structure{class:votes}, then this dictionary was sorted to determine the class having the highest vote.This gave rise to two cases:1. All classes in the *classvote* had same number of votes, under such circumstances the class with highest similarity was considered, else the class with highest number of votes.A for loop was created which iterated through the above function to find neighbours for each test sample.A file called *results.txt* was created which collected all the classes in a synchronize order. On multiple attempts of trial and error, the value of **K=100** and **epsilon=0.09** was chosen which reflected a F1 score of approx. **75.74%.**The code can be further improved for accuracy and reduction of time complexity by implementation of K-d tree which is being worked on. The code takes approx 25 mins to run on HPC, besides this no other instructions are necessary for running it.