Assignment 1 Report
PR1 - Medical Test Classification

By Anshul Patni
SJSU ID: 012415156

Rank: 13
F1-Score: 0.7887

Approach:

We have been given a dataset with medical abstracts that have been categorized under 5 classes, namely, digestive system diseases, cardiovascular diseases, neoplasms, nervous system diseases, and general pathological conditions. We need to build a predictive model, that can identify the class of the problems given to us with high precision. In order to build such a model, the approach taken was to, first train the model based on the train.dat file. For this, the steps taken were to first load the dataset in the jupyter notebook and then extract featured from this dataset using bag of words model. After this was done, the model had to be applied with some machine learning algorithm to classify the test dataset given to us. As the model started giving predictions with decent accuracy, removal of stop words was added to improve the performance of our model to give more accurate predictions.

Methodology:

I. Dataset was first uploaded to the jupyter notebook.

For loading the dataset, we have used read_csv method from pandas library. The read_csv method is used to read comma-separated file into a DataFrame.

In our case, we have a .dat file of the training dataset which we will load as-

```
import pandas as pd                                  #importing the library pandas
df = pd.read_csv("train.dat", sep="\t", header=None) #read_csv() method used to read the train.dat
                                                     file which has been splitted into two columns
                                                     using tab-seperator
```

II. Extracting all the features (word count) from our file by creating vectors.

The text file that we have here is like a series of words. So, if want to run algorithms on this text file, we will have to create a vector with numerical features in it. For this, we have used bag of words model, where we have segmented each text file into words. We have then counted the number of times each word occurred in each line of the line. What this does is that it assigns unique value to each word which corresponds to a unique feature in our file.

```
from sklearn.feature_extraction.text import CountVectorizer   #Countvectorizer is the method which will
                                                              extract the features from the text file as
                                                              frequency of all the unique words

vector = CountVectorizer()
counts = vector.fit_transform(df[1])                          #this will return a document matrix of
                                                              n_samples and n_features

counts.shape
```

III. Finding the TF-IDF in order to reduce the weightage of common words

Counting the number of words in previous step will result in giving proportional weightage to the documents with respect to their length. This is not desired. To avoid this, we will use normalized version of the count of words, i.e., number of words / total number of words. For this, we will be using TF-IDF transform and then we will find term frequency times inverse document frequency.

```
from sklearn.feature_extraction.text import TfidfTransformer
tfidf_transformer = TfidfTransformer()
tfidf_freq = tfidf_transformer.fit_transform(counts)
tfidf_freq.shape
```

IV. Train the classifier which will be used to apply machine learning algorithm

The next step is to use an appropriate algorithm to train our classifier using the training data, which will then be used to find the classification of the data in test file. We have used Naïve-Bayes algorithm to do this.

```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB().fit(tfidf_freq, df[0])                    #training the classifier using multinomial
                                                                Naïve-Bayes algorithm
```

V. Build a pipeline to perform multiple transformation steps in a single step

We will build a pipeline in-order-to reduce the efforts in the above steps and we will apply vectorizing method, TF-IDF transform as well as linear svc in a single step.

```
from sklearn.pipeline import Pipeline
from sklearn.svm import LinearSVC
text_clf = Pipeline([('vect', CountVectorizer()), ('tfidf', TfidfTransformer()), ('svc', LinearSVC()),
])
```

VI. Create a .dat file of the predictions made using test file

We will use our model to find the classification in our test file and build a prediction file which can be tested to find the accuracy of our algorithm.

```
test_data = open("test.dat")
predict_class = text_clf.predict(test_data)                     #applying the classifier to perform
                                                                classification on our test file

result = pd.DataFrame(predict_class)
result.to_csv('myPrediction1.dat', index=False, header=None)
```

Conclusion:

Our model gives prediction with a decent accuracy of 78.87%. I tried to use a random forest algorithm as well as removing the stop words, but these methods somehow degraded the accuracy of our model. I look forward to work on the model with more algorithms on this data to improve the accuracy of our model.