Programming 1

Medical Text Classification

By,
Poojitha Amin
SJSU ID: 011811306
Rank – 9
F1-Score - 0.7927

8 March, 2018

Text Processing

In order to achieve better results from the model, the raw data has to be converted to a clean data set. Data pre-processing is the preliminary step followed before doing the actual machine learning task.
Following steps were used to pre-process the data:

- **Read the data set**
  Read the training data from the file train.dat.txt.

- **Split into Training and Evaluation Data**
  Split the labeled data into train and test subsets, 80% for training and 20% for evaluation, in order to evaluate the predictive quality of the model, before making predictions on the unseen data set.

- **Eliminate stop words**
  Removed the stop words as it appeared to be of no use in helping classify the text data.

- **Stemming**
  Reduced the words to its base form. It helps reduce the size of the dictionary and avoids the classifier from putting two similar words under two different frequencies.

- **Change to lower case**
  Changed the entire corpus to lower case.

- **Removed punctuations and numbers**

- **Extract features from text data**
  Used scikit-learn's library CountVectorizer, to convert text documents to a matrix of token counts. It involved breaking the text document into words and counting the frequency of its occurrence in documents.

- **Term frequency-inverse document frequency**
  Used TF-IDF to measure the frequency of the term in the document and to measure how important the term is. The frequent words are given less weightage and the rare ones are scaled up.

Classifier Model Development

- **Selection of classifier model**
  Trained multiple classification models like the Naive Bayes, LinearSVC, kNN, RandomForestClassifier, Perceptron and SGDClassifier. Analyzed the performance outcome of each of them. SGDClassifier seemed to be giving the best results.

- **Random state**
  Set the random state value for the test, train split set and classifier model so that we get the same result set and output each time the program is run.

- **Pipeline**
  Used Pipeline to combine the transformer, tokenizer and the classifier model sequentially.

- **Handle imbalanced data set**
  To handle the imbalanced data set, multiple options were used like the 'class_weight' parameter of the SGDClassifier, Over_Sampling, Under_Sampling, Random_Sampling and SMOTE options of the imbalanced-learn extension. However, none of them contributed to improving the F1-score significantly.

- **Performance metrics**
  Used F1-score to measure the accuracy of the model as we had an imbalanced data set. It considers both the precision and the recall to evaluate the model. Trained the model with the split train data and then measured the f1 score using the test subset split from the train data.

- **F1 score improvement**
  Tuned multiple parameters of the classifier to improve the performance of the model.
  N_iter – Varied the n_iter value, that decides the number of passes over the training data. Got the best score at value 6.
  Loss – Used multiple loss functions such as, log, hinge, modified_huber, squared_hinge, of which hinge gave the best results which gives a Linear SVM.
  Penalty – Tried multiple standard regularizer for linear SVM models like l1, l2 and elasticnet.
  Alpha – Tried a range of alpha values to get the best results. Alpha is the constant that multiplies the regularization term. It worked best at a value of 0.004.

Instructions for running the code

- Add the file paths for the test and train data files under,
  #Read train data
  #Read test data

- Download nltk

- Modify the output file path in the last step.