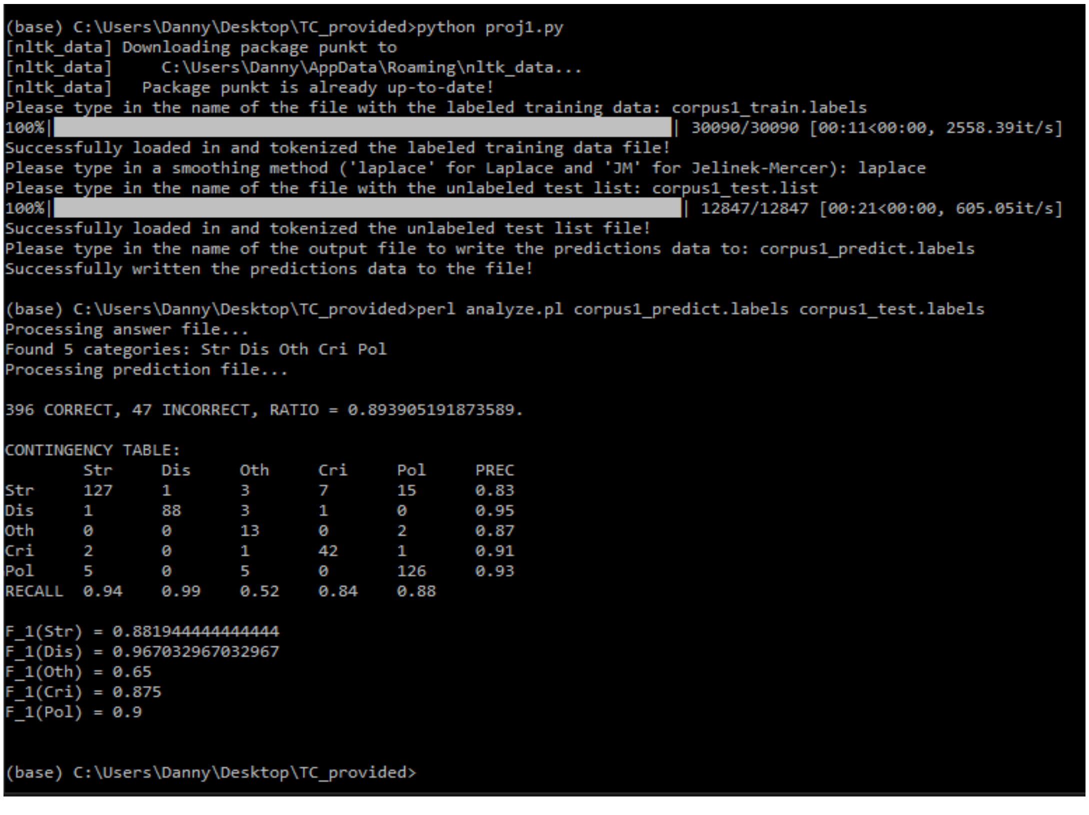
Danny Hong
ECE 467 Project 1: Text Categorization
Spring 2021

Compiling

This text categorization program was written in Python Version 3.8.5 (some older versions would work as well) and compiled in an Anaconda Version 4.9 environment (Can be compiled in an Ubuntu/Cygwin/MacOS Terminal environment as well). To compile the program, first make sure that the necessary packages have been pip installed: os, sys, string, nlty. Tqdm, numpy, math, log from math, PorterStemmer, and nltk.stem from PorterStemmer. Additionally, as seen from the image of the sample run as shown below, when the program is compiled, the 'punkt' package from nltk is installed. In line 11 of the program, I wrote "nltk.download('punkt')," since while I was debugging my code, the compiler reported a Lookup Error when the punkt package was being loaded in from nltk, and so to prevent this from happening I had to code the downloading of 'punkt' that into my program. If 'punkt' had already been installed in your system, I suppose this shouldn't be an issue and so this line can just be commented out or removed. In any case, this line can just be commented out or removed after this program has been compiled once already.

The picture below shows a sample run of the program, which is called proj1.py. While running, the program will ask the user to input the names of the necessary files, along with offering the user to choose a smoothing method (either Laplace or Jelinek_Mercer) to be used (since this text categorization employs a Naive-Bayes system created from scratch). After the UI interactions were completed, I ran Professor Sable's test script to compare the predicted labels data with the actual testing labels data provided, in which certain important statistics regarding the accuracy of my system were displayed.



```
(base) C:\Users\Danny\Desktop\TC_provided>python proj1.py
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Danny\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Please type in the name of the file with the labeled training data: corpus1_train.labels
100%|███████████████████████████████████████████████████████████████| 30090/30090 [00:11<00:00, 2558.39it/s]
Successfully loaded in and tokenized the labeled training data file!
Please type in a smoothing method ('laplace' for Laplace and 'JM' for Jelinek-Mercer): laplace
Please type in the name of the file with the unlabeled test list: corpus1_test.list
100%|████████████████████████████████████████████████████████████████| 12847/12847 [00:21<00:00, 605.05it/s]
Successfully loaded in and tokenized the unlabeled test list file!
Please type in the name of the output file to write the predictions data to: corpus1_predict.labels
Successfully written the predictions data to the file!

(base) C:\Users\Danny\Desktop\TC_provided>perl analyze.pl corpus1_predict.labels corpus1_test.labels
Processing answer file...
Found 5 categories: Str Dis Oth Cri Pol
Processing prediction file...

396 CORRECT, 47 INCORRECT, RATIO = 0.893905191873589.

CONTINGENCY TABLE:
        Str     Dis     Oth     Cri     Pol     PREC
Str     127     1       3       7       15      0.83
Dis     1       88      3       1       0       0.95
Oth     0       0       13      0       2       0.87
Cri     2       0       1       42      1       0.91
Pol     5       0       5       0       126     0.93
RECALL  0.94    0.99    0.52    0.84    0.88

F_1(Str) = 0.881944444444444
F_1(Dis) = 0.967032967032967
F_1(Oth) = 0.65
F_1(Cri) = 0.875
F_1(Pol) = 0.9


(base) C:\Users\Danny\Desktop\TC_provided>
```

Figure 1: Screenshot of a sample run.

Information about System

As stated before, the machine learning method used to create this text categorization program was Naive-Bayes. I decided to tokenize the training and test sets by using nltk.word_tokenize. In regards to smoothing, I considered two smoothing methods, Laplace and Jelinek-Mercer, and my program was written so that it would allow the user to choose between either one of those two smoothing methods that will be implemented when running the code. I also experimented with a few alpha values to see which one would give the best results, and I concluded that using alpha values of 0.01 for Jelinek Mercer and 0.055 for Laplace would generate the most accurate data. Upon comparing results in the end, I noticed that Jelinek-Mercer provided slightly better accuracies than Laplace for corpora 1 and 3, with the differences in accuracy for both corpora being less than 1%. However, for corpus 2, Laplace ended up providing more accurate results than Jelinek-Mercer, with a difference in accuracy by about 1.6%. I believe that in this case, using Laplace smoothing would be slightly better suited for all three corporas than using Jelinek-Mercer smoothing since it provided slightly better well-rounded results as using Jelinek-Mercer negatively impacts the accuracy for corpus 2.

In regards to optional parameters, I experimented with replacing words, removing punctuation, and PorterStemmer. I found PorterStemmer to be extremely effective in increasing the accuracies for all three corpora. I also noticed that taking away punctuation did help the accuracy a bit as well. Finally, I tried replacing words by combining separate words that were similar to one word, but that turned out to be ineffective.

Since the testing list and labels data were only provided for corpus 1, I had to break up the corpus 2 and corpus 3 training sets respectively into a smaller training set and a tuning set. Therefore, I would then be training my system for corpus 2 and corpus 3 with the smaller training sets and then use the tuning sets for testing. I experimented with the percentage regarding the train/test division, and I concluded that a 70% train/30% test split would be the most effective.