Nicholas Gao

ECE467 Natural Language Processing

Professor Sable

Spring 2018

Project #1: Text Categorization

Submission Description

**Compiling the Program**

The program should be ready to run using the “python” command.

**Program Description**

The program uses a Naive Bayes machine learning classification method. The training and test documents from the corpuses are converted to lowercase, and then tokenized using the word\_tokenize() method from NLTK.tokenize toolkit/module. I chose to use this tokenization method as oppose to other methods such as the regular expression tokenizer (RegExpTokenizer). This was because it was able to recognize important abbreviations like “U.S.” as a token, instead of separating it into “u” and “s” tokens. This choice made sense to me, since an abbreviation like “U.S.” would seem indicative of categories like Politics from corpus 1 and US News from corpus 3.

While Naïve Bayes uses a bag-of-words approach, it doesn’t exactly have TF-IDF weights. Rather each category keeps track of the frequency of each token of training documents labeled with that category. This is then used to calculate the conditional probability or likelihood of having a token given a category.

In order to treat unknown words, Laplace smoothing was used with a smoothing parameter of .1. This value seemed to provide relatively better results.

**Tuning Parameters and Evaluation**

In order to evaluate and tune my program, I followed a similar process as described in the assignment handout’s description. In order to avoid touching the test set, I divided the training set of each of the three corpuses into a smaller training subset and tuning/test subset. The trailing one third (1/3) or approximately a third (when the total number of documents is not divisible by 3) of the training set was placed into a tuning/test set. The remaining preceding two thirds was placed in the new training set. While this process was used to find the optimal values for the hyperparameters, the performance is likely overestimated. However the point is that it shows that the chosen parameter settings/values perform better relative to other parameter settings/values.

The three parameters I tried tuning were the smoothing parameter in the Laplace smoothing for Naïve Bayes, whether or not to include case sensitivity in the tokens, and whether or not to conduct stop word filtering.

In regard to the smoothing technique, it began as Add-One smoothing (smoothing parameter equal to 1), but reducing the smoothing parameter to .1 improved performance. Table 1 summarizes the different values that I tested with the tuning set from corpus 1. The improvement in performance begins to taper off as I dropped the smoothing parameter below .1. Thus I decided .1 is suitable and would avoid over-tuning.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Smoothing Parameter | .01 | .05 | .1 | .3 | .5 | 1 | 2 |
| Accuracy | 0.88136 | 0.88136 | 0.87797 | 0.86102 | 0.85085 | 0.83051 | 0.81017 |

Table 1: Summarization of the accuracies achieved using different values for the Laplace smoothing parameter. This experiment was conducted using the tuning set created for corpus 1.

Next I experimented with case sensitivity, whether or not to let tokens have capital letters. Table 2 summarizes the comparison on the tuning set. Converting the tokens to lowercase led to small improvements. I eventually decided to keep this feature as it helped to reduce the number of tokens overall.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Corpus 1 | Corpus 2 | Corpus 3 |
| Case sensitive | 0.877966101694915 | 0.828859060402685 | 0.301075268817204 |
| Case INsensitive | 0.877966101694915 | 0.838926174496644 | 0.320224719101124 |

Table 2: Summarization comparing the accuracies (ratio of correct classifications) on the tuning set of the three corpuses, when using or not using case sensitivity. In the case of case insensitivity, all tokens were converted to lowercase. To address the noticeably low accuracy on corpus 3, it appears that my crude way of creating the tuning set led to a bad class imbalance in the tuning set.

The final feature I tried was stop word filtering. Unfortunately, it didn’t appear to offer any improvement in the results. In most cases for all three corpuses, the difference in percentage correct between having or not having stop word filtering was <1%. In one case, stop word filtering actually decreased accuracy. In this case, I used my judgment and decided the results on the tuning set did not indicate any performance benefit to keep stop word filtering in the program.

**Files**

*main.py* – The text categorization program. It asks for the training set, test set, and the name for an output file.

*makeSets.py* – The python program written to divide a training set into subsets – a new training set and new tuning set. It outputs two values, which are respectively the number of documents in the new training and tuning sets.