Nithilam Subbaian, NLP Proj 1

I finished the assignment using Python. In order to run the code, you must have these pip installed: nltk, os, sys, tqdm, log from math, numpy, PorterStemmer from nltk.stem, string, re, pandas. Note that when installing nltk.word_tokenize, I had to follow additional steps to download "punkt", as explained in resource [2].

To complete the assignment, the basic machine learning method I used is Naïve Bayes. I used nltk.word_tokenize to tokenize the training and test files. With Naïve Bayes, I decided to choose additive smoothing for the final version. The Jelinek-Mercer Smoothing did produce competitive accuracies with corpus1 and corpus3, but it reduced the accuracy of corpus2. To be safe I chose the additive smoothing.

I played with several optional parameters such as stop lists and punctuation removal. Removing stop words did not help the accuracy; however, removing punctuation marks did improve the accuracy by a small but measurable amount (less than 1 percent). Using PorterStemmer, significantly increased the accuracy by several percentage points; this change produced the most significant results. I also did play with word replacements, to combine similar words into one word (after peering into the data); however, this did not help with the accuracy. I also tried a different tokenizer, but they did not seem to have any differences, so I stuck with PorterStemmer.

I did a lot of testing to see which alpha values produced the best results, as well as which smoothing techniques produced the best results. I wrote a separate script to test the different combinations; I can provide this script if necessary. Using resource [7], I learned about the different possible smoothing techniques best used for Naïve Bayes and short text classification (I assumed this text was short in comparison to much larger pieces NLP can be used for). As the text I used was different than that in the research paper, I tested for the optimal variable values (alpha, lambda, ...), instead of using the ones they provided. In the key for the graph shown in the following page, laplace = Laplace Smoothing, JM = Jelinek-Mercer Smoothing, Dir = Dirchlet smoothing, AD = Absolute Discounting smoothing, TS = Two-stage smoothing. Note that these results are for corpus1. For corpus1, TS produced the best results for alpha = 0.01, with an accuracy a little greater than 90 percent. However, this performance was not consistent over the other datasets. Laplace Smoothing seemed to be the most consistent among various alpha values, as well as the different corpuses.

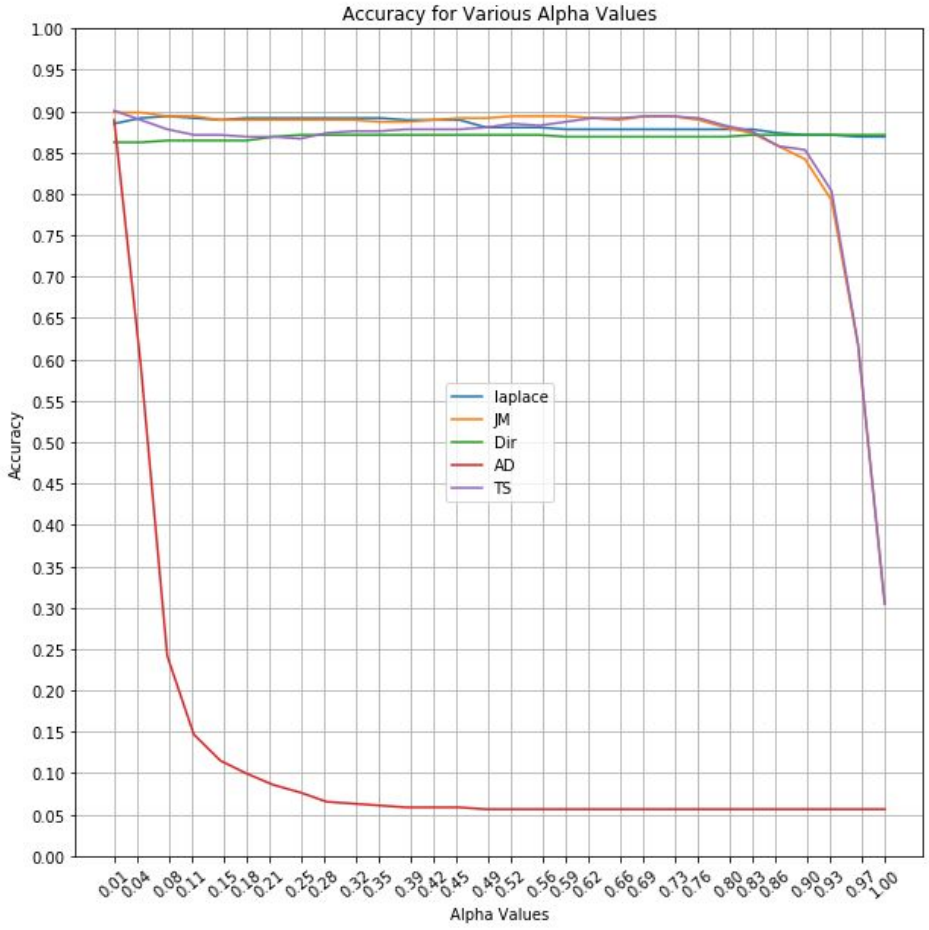
In order to test the code on the other corpuses, I separated the second and third datasets by various percentage splits. I generally used 75/25; however, to match with the accuracies from the presubmission, I was nearing 55/45 split. Unfortunately, the results I obtained when testing the data were aligning within several points from Prof. Sable's tests, so it looks like the contents of our tests/training data are similar, but not similar enough for me to make decisions based on 1-2% improvements or reductions. For this reason, I would like to use the code I previously submitted in which I know the results are reasonable for the other corpuses, if I can't test the new code with JM smoothing.

Resources:

1. https://www.nltk.org/book/ch03.html
2. https://stackoverflow.com/questions/4867197/failed-loading-english-pickle-with-nltk-data-load
3. https://web.stanford.edu/~jurafsky/slp3/4.pdf
4. https://pythonprogramming.net/stemming-nltk-tutorial/
5. https://www.ntu.edu.sg/home/gaocong/papers/wpp095-yuan.pdf

6. https://nlp.stanford.edu/~wcmac/papers/20050421-smoothing-tutorial.pdf
7. https://www.ntu.edu.sg/home/gaocong/papers/wpp095-yuan.pdf

Accuracy plotted for various Alpha Values and Smoothing Techniques for Corpus 1:



From Resource 7:

2. SMOOTHING FOR NAIVE BAYES

Given a question d to be classified, Naive Bayes (NB) assumes that the features are conditionally independent and finds the class c_i that maximizes $p(c_i)p(d|c_i)$.

$$p(c_i) = \frac{|c_i|}{|C|}, \qquad p(d|c_i) = \prod_{k=1}^{|d|} p(w_k|c_i)$$

where $|c_i|$ is the number of questions in c_i, and $|C|$ is the total number of questions in collection. For NB, likelihood $p(w_k|c_i)$ is calculated by Laplace smoothing as follows:

$$p(w|c_i) = \frac{1 + c(w, c_i)}{|V| + \sum_{w' \in V} c(w', c_i)}$$

where $c(w, c_i)$ is the frequency of word w in category c_i, and $|V|$ is the size of vocabulary.

With different smoothing methods, $p(w_k|c_i)$ will be computed differently. We consider the following four smoothing methods [4] used in langauge models for information retrieval. Let $c(w, c_i)$ denote the frequency of word w in category c_i, and $p(w|C)$ be the maximum likelihood estimation of word w in collection C

1) Jelinek-Mercer (JM) smoothing:

$$p_\lambda(w|c_i) = (1 - \lambda)\frac{c(w, c_i)}{\sum_{w' \in V} c(w', c_i)} + \lambda p(w|C)$$

2) Dirichlet (Dir) smoothing:

$$p_\mu(w|c_i) = \frac{c(w, c_i) + \mu p(w|C)}{\sum_{w' \in V} c(w', c_i) + \mu}$$

3) Absolute Discounting (AD) smoothing:

$$p_\delta(w|c_i) = \frac{max(c(w, c_i) - \delta, 0) + \delta|c_i|_u p(w|C)}{\sum_{w' \in V} c(w', c_i)}$$

where $\delta \in [0, 1]$ and $|c_i|_u$ is the number of unique words in c_i.

4) Two-stage (TS) smoothing:

$$p_{\lambda,\mu}(w|c_i) = (1 - \lambda)\frac{c(w, c_i) + \mu p(w|C)}{\sum_{w' \in V} c(w', c_i) + \mu} + \lambda p(w|C)$$