LING 406 MP2

Question 1 (Letter Bigram)

Step 1: Data Processing

In order to process the data for this project, I created a separate readTrainingFile function which gets the text from a file, parsing it into a string. This string is modified using a series of regular expressions to clean any discrepancies. In the example of this particular training set, there are several problems with punctuation marks being separated by spaces, etc. This seems to especially be a problem with the Italian and French words where these punctuation marks are crucial parts of a word's meaning. For example, *E* in Italian means *and* whereas *E'* means *is*. If our input data had *E '* it could be less accurate since (E, ') would no longer be a letter bigram. This particular bigram indicates a very high probability of the language being Italian since it is seldom seen in the other two languages. In order to correct these issues, I use regular expressions to remove these incorrect spaces. Please note that while I am fluent in English and Italian, I have very little experience with French, and therefore many of the in-word reforms I made are based on my assumptions on what the words are *supposed* to be. You can see all the regex reforms in the readTrainingFile function of fileReader.py. I also get rid of end of word punctuation marks because they can interfere with the trainer. This is because words of certain languages have a tendency to end a certain way and having better insight into this can improve our classifier. By removing end of word punctuation, we can congregate all those bigrams into the ending letter followed by a space, thereby giving us more end of word data. I have also shifted all the letters to lowercase because there is not enough training data to have accurate probabilities for both lowercase and uppercase letters.

Step 2: Training and Classification

In order to train the model, I am using a basic Naïve Bayes classifier. The classifier assumes equal prior probability for all languages. However, keep in mind that in the real world these prior probabilities are likely to be drastically different. For example, if we were scraping the web randomly for languages, the prior probability of a word being English rather than French or Italian is much higher. If we were scraping through captions on an English Instagram account, the prior probability of their feed's captions being in English is significantly higher than Italian or French.

For OOV words I use a simple Laplace smoothing method. This method is very basic but also fairly effective for language classification. I chose it because it is simple to implement, easy to understand, and works efficiently for our use case. The letter bigram model can't really be implemented without smoothing because the number of possible letter bigrams is insanely high, and therefore it is almost impossible to create a full map of probabilities for every use case. Therefore, we must create a solution for the OOV words, which will use smoothing.

This program has an accuracy rate of 99% using the given training and test sets.

Question 2 (Word Bigram)

Step 1: Data Processing

This program uses the exact same data processing techniques as in Question 1. Removing all the punctuation marks tends to work particularly well for the word bigram, given that there is far less data to work with (less words than letters obviously). This means that smoothing out our capitalization, punctuation marks, etc. can create a much more accurate classifier given the lack of a giant training corpus.

Step 2: Training and Classification

The training and classification phases are also more or less the same as the above, except we are now using words instead of letters. We still use Laplace smoothing, and the reasoning for why we use it is even more pronounced for word bigrams, where there are trillions of possible word combinations (considering misspellings and random typos/garbage data).

This program has an accuracy rate of 98.33% using the given training and test sets.

Question 3 (Word Bigram w GT Smoothing)

Step 1: Data Processing

This program uses the exact same data processing techniques as in Question 1 and 2. Removing all the punctuation marks tends to work particularly well for the word bigram, given that there is far less data to work with (less words than letters obviously). This means that smoothing out our capitalization, punctuation marks, etc. can create a much more accurate classifier given the lack of a giant training corpus.

Step 1: Training and Classification

This program is mostly the same as above, but uses GT smoothing, as opposed to Laplace smoothing. In order to deal with potential unreliability, we use a linear regression to create better estimates for areas without continuity. You can see this use of regression in the goodTuring() function.