LING 406 Machine Problem 2 Report

Spring 2020 Feb 21, 2020

Question 1

a) I considered every single letter (except space, newline, comma and period) a token for the letter-bigram model. I purposefully excluded those letters since they are not language-specific. Including those might skew the probability of sentences, hence skewing the accuracy of the model. However, I did not exclude **all** punctuation marks since some of them *are* language-specific. Before feeding the data into my language model, I segmented all three training sets into three separate lists of sentences for the sake of simplicity. However, not segmenting the text into sentences also works just fine.

b) There might be a considerable amount of OOV words in the test file. It is possible that the OOV words we identified do not exist in a language we are speculating. However, it is also possible that some of the OOV words exist in a language we are speculating but do not exist in the training set. So, for all OOV words, I used Laplacian Smoothing to increase the probability of a token from zero to a very small floating number.

c) In fact, the diagram model *can* be implemented without any smoothing since, by chance, every single token appeared at least once. Without any smoothing, my implementation of the model yields an accuracy of 91.33%. However, after I applied the Add-One smoothing, I saw a 5 percent increase in accuracy. Thus, add-one smoothing is a better algorithm for the letter-bigram model.

Question 2

a) Similar to the answer I answered in the previous question, I considered every word (except space, newline, comma, period and any combination of those four). For preprocessing, I only segmented all three training sets into three separate lists of sentences. However, there is no need for me to do that.

b) For all OOV words, I first *tried* grouping all low-frequency words into a big unknown group. However, my implementation yields a abysmally low accuracy rate when I implemented the model like this. If we group the low-frequency words into one big group, then any of the words within this group is significantly over represented. Thus skewing the accuracy of the model. So, similarly, I used Laplacian Smoothing to increase the probability of a token from zero to a very small floating number.

c) In this case, the word bigram model cannot really be implemented without smoothing. Since the size of the training sets is quite limited, we cannot guarantee that if a bigram does not exist, the first word of the bigram exists. Without using any kind of smoothing, the probability of a non-existent diagram with a non-existent first unigram is 1/0, which is infinity. So, Add-one smoothing should be used to prevent this from happening. However, add-one smoothing is not really optimal since, for the case of a non-existent diagram with an existent unigram, it might identify the sentence that the diagram is a part of as the wrong language.

Question 3

For all unreliable words, we adjust the probabilities of unreliable words by using the formula (c+1) * Nc+1/Nc. After applying this formula, in most cases, the probability of unreliable words increases while the probabilities of frequently seen words decreases. For all unseen words, we treat the words as the words that are seen once and compute the probability by dividing the number of words seen once with the number of all words.

Each language model has its drawbacks. However, *empirically* speaking, the best model is the word-bigram model with Good-Turing Smoothing which yields a shocking accuracy rate of 99%.
Below are the advantages and disadvantages of each language model I implemented.

Letter-Bigram Model

Pros: High accuracy rate with or without using the add-one smoothing.

Cons: Theoretically speaking only a small number of bigrams exist. Suppose there are 200 characters in a language, then there only exists 200^2 = 40,000 combinations of bigrams. As the the size of the text to be identified increases, I could see the accuracy of the model decreases.

Word-Bigram Model

Pros: Consistent Accuracy with respect to the size of the text to be identified.

Cons: Impossible to implement without any kind of smoothing. Relatively low accuracy and a large false-positive rate if the number of words in each language differs dramatically.

Word-Bigram Model with Good-Turing smoothing

Pros: Best accuracy across the board! The size of tokens in each language does not skew the result of identification to a wrong language.

Cons: By applying GT smoothing, the probability of each word homogenizes. If the size of words in each language is large, I could see that there might be words with different frequencies, but after adjusting, they all have the same frequency. This could cause some inaccuracies in language identification.