A3 CSE 156 Report

David Dai
A13665770
May. 13/2019

Section 1. Baseline

1. Baseline algorithm

The baseline algorithm takes several steps. In the first step, it will takes in a training file and then produces the trigram, bigram and emission counts. The emission counts here is calculated by the formula $e(x|y) = \frac{Count(x \rightarrow y)}{Count(y)}$. $Count(x \rightarrow y)$ is the number of times a word has appeared in the data as I-GENE; the denominator, $Count(y)$, stands for the unigram count, or in other words, the total number of appearances. Then the second step of the baseline algorithm is to generate a new train file names "gene-rare.train". The importance of this file is to replace all the words with a count less than 5 with the symbol '_RARE_'. Then, with this new train file, the baseline algorithm can move on to re-read the "gene-rare.train" file with some of the words being replaced. This is especially useful when we encounter words that we have never seen before. This way when we encounter words that we have not seen, we can assume them as '_RARE_' words. Lastly, the baseline algorithm will generate a file named "gene-rare.counts" and we will use the counts to predict whether a word's type in the "gene.dev". Then, we will run the eval_gene_tagger and generate the baseline result below.

2. Baseline Result

For the baseline result on the development set, the baseline algorithm found 2669 Genes. The development set is expecting 642 Genes and 424 of them are correct. The result is displaying in the table below.

	precision	recall	F1-score
GENE	0.158861	0.660436	0.256116

3. Word Classification

There are a couple scenarios when we see a word from the development dataset. One of it is that the word has been seen from the count file. On the other hand, if the word has been seen less than 5 times in the original count file, it will be marked as '_RARE_'. And all the rare words or the words that have not been seen from the training dataset will all be treated as a word unseen.

a. Words in training

If we see a word from training, then we can directly use the count file to do the calculation. Here is the implementation of emission-count. There are two cases. One is that the word is a 'O' symbol and the other is that the word is an 'I-GENE' symbol. In both cases, we will calculate two different kind of numbers for the word: one is the 'O' count number over the total number of 'O' type words appearances, and the other is the 'I-GENE' count number over the total number of 'I-GENE' type words appearances. If the first number is bigger than we will attach 'O' to the word and write to the file; else, we attach 'I-GENE' to the word and write to the file.

b. Words unseen

For the words that we have not checked or viewed from the training file, we have the '_RARE_' words for it. The baseline algorithm will simply replace the word with '_RARE_'. Then, the algorithm will create two potential scenarios in this case; one is '_RARE_' word plus the 'O' symbol, and the other is the '_RARE_' word plus the 'I-GENE' symbol. Both two scenarios will be tested with the count and the calculation will be the same as above.

4. New Baseline Model

a. Improvements

There are several ways to improve based on the baseline model. My implementation is working with the '_RARE_' words. Due to the limit size of the training dataset, there might be a lot of words not being checked; and simply categorizing all the words to be '_RARE_' is rather a thoughtless idea. Thus I propose we create three more cases other than '_RARE_'. One case is checking if the word is a number, then we will replace the word with '_NUMBER_'. The second case is checking if the word contains an uppercase letter, then we will replace the word with '_UPPER_'. Lastly, my implementation should check fi the word contains a punctuation. If the word does contain, then we will replace it will '_PUNC_'.

b. Result Comparison

With my implementation, it founds 2012 GENEs and it expected 642 Genes which 418 of them are correct.

	precision	recall	F1-score
Baseline	0.158861	0.660436	0.256116
Improved	0.207753	0.651090	0.314996

Precision here is defined by $\frac{|A \cap B|}{|A|}$ and the recall is defined as $\frac{|A \cap B|}{|B|}$. The F1-score here is the geometric mean of these two values, which is $\sqrt{precision^2 + recall^2}$. As the graph above has showed, the precision and the F1-score both increased because we categorized the '_RARE_' words with more details.

Section 2. Trigram HMM

1. **HMM Tagging**

HMM stands for Hidden Markov Model. The model takes in an input sentence and a tag sequence and the HMM is used to define the number of possible sequences exponential in n, where n is the length of the input sentence. In the general case, there will be W to be a set of n numbers of words and T to be a set of x number of tags. We then can define a new set of sequence S to be a set of all sequence pairs. And generate $p(x_1, x_2, ...x_n, y_1, y_2, ..., y_n)$ which is the probability distribution over the set S. The HMM will thus generate the *argmax* of the probability distribution, or in other words, the HMM will generate the highest probability tag sequence.

One of the HMMs that we used in this case is the trigram HMM. The trigram HMM is a little bit different parameter-wise. W = W$\cup \{STOP\}$ and T = T $\cup \{*\}$. $\{STOP\}$ stands for the stop word and $\{*\}$ means the start symbol for the trigram language model. One of the parameter for the trigram HMM is $q(s|u, v)$ where u, v are from W and s is from T. $q(s|u, v)$ is the probability of seeing a bigram tag(u,v) and follows with the tag s. The other parameter is $e(x|s)$ where x is from V and s is from T and it could be interpreted as the probability of x paired with s. Then $p(x_1, x_2, ...x_n, y_1, y_2, ..., y_n) = \Pi^{n+1}_{i=1} q(yi\,|yi{-}2, yi{-}1)\, \Pi^{n}_{i=1} e(xi\,|yi)$ and HMM calculates the armax of this equation.

2. Viterbi Algorithm

- **General Description**

Viterbi algorithm is a dynamic programming of finding the argmax of $p(x_1, x_2, ...x_n, y_1, y_2, ..., y_n)$ as section above has described regarding trigram HMM. $argmax_{y_1 ... y_{n+1}}\, p(x_1 \ldots x_n, y_1 \ldots y_{n+1})$ in this case we have assigned y_{n+1} to be {STOP} and y_0 and y_{-1} both be {*}. Since we are using a trigram HMM, we have $p(x_1, x_2, ...x_n, y_1, y_2, ..., y_n) = \Pi^{n+1}_{i=1} q(yi\,|yi{-}2, yi{-}1)\, \Pi^{n}_{i=1} e(xi\,|yi)$. Then we are going to define a dynamic programming table with a recursive definition defined below: $\pi(k, u, v) = max_w(\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|\, v))$ where the basecase if $\pi(0, *, *) = 1$.

- **Implementation**

My implementation for the Viterbi algorithm uses partial functions from the baseline implementation. The first step is following the improved baseline functionality. I replaced all the words from the training dataset that are less than 5 counts into 4 categories. The categories are words containing uppercase, containing a number, containing a punctuation and other cases which is the rare word case. Then after replacing, the algorithm will load the file as sentences and add the two "*" s in the front of the tags. For the trigram model, the stop word will be appended.

The algorithm will go on to create two 2-D arrays to be the dynamic programming table and we will constantly update the values. The first array carries the value of π and we set the first index of π (0, "*, *") to be 1, which is the base case. Then, the second array is tracking the value w from S_{k-2} after u and v has been taken out. Then, for the dynamic programming part, we are

going to loop through through all the k from 1 to n, u from S_{k-1} and v from S_k. For each sentence we will treat them as a trigram language model and if the current index has been seen from the training dataset, then we can directly implement the $(\pi(k-1,\ w,\ u)\ \times q(v|w,u)\ \times\ e(x_k|\ v))$ where $\pi(k-1,\ w,\ u)$ is taken from the last iteration or the previous index of the array. In the first iteration, since we have declared the first index to be 1, $\pi(k,\ u,\ v)\ =\ (1\ \times q(v|w,u)\ \times\ e(x_k|\ v))$ where k here is the 1st iteration. $e(x_k|\ v)$ is the emission count is the same as the baseline model emission count. As for the $q(v|w,u)$ is the number of appearances of v divided by the number of appearances of bigram w and u. On the other hand, if the word of the current index has not been seen, then we will treat it into the four categories of containing uppercase, containing a number, containing a punctuation or rare word case. After looping through the first sentence, we are going to let the first array to store the maximum value of the $\pi(1,\ u,\ v)$ output and let the second array to store the maximum w_1 value. In the second iteration if the resulting $\pi(2,\ u,\ v)$ is bigger than the previous index, $\pi(k,\ u,\ v)$, the algorithm should store this second index with $\pi(2,\ u,\ v)$ and w_2 value. If $\pi(2,\ u,\ v)$ is not bigger than, the arrays should still store $\pi(1,\ u,\ v)$ *and* w_1 value. This rule applies throughout the for-loop from 1 to the length of the sentence + 1 (N+1).

- **Difficulties**

During the implementation I am always having a hard trouble understanding the separation of S_{k-2}, S_{k-1} , and S_k. And more importantly, my understanding of the tagging was not really well-established. I spent majority of the time trying to implement the baseline model from scratch and thus I did not have enough time to finish the implementation of the trigram HMM. Thus in short, I did not finish this part of the programing assignment. I would love to finish this later.

3. **New HMM Model**

Improvements can be made besides the splitting '_RARE_' words into 4 categories. One way I am proposing is to check the suffix and the prefix of the word. Prefixes like ["a", "pre", "post", "un"] all tend to be like adjectives. And if we take a looking at suffixes like ["ly"] all tend to be adverbs, ["ing"] tend to be adjectives or verbs and so forth. Another way I think can improve the HMM model is adding the conditional maximum likelihood (CML) aspect .[1]

4. **Provide insightful comparison based on these results;**

Based on my understanding the HMM trigram model will have a higher precision and higher recall comparing the the baseline model and improved baseline model. The HMM trigram model would also generate an overall higher F1-score around 0.39.

Section 3. References

Krogh, Anders. "Two methods for improving performance of an HMM application for gene finding." *Center for Biological Sequence Analysis. Phone* 45: 4525.