Programming Assignment 3 : Sequence Tagging

Gitika Meher
Department of Electrical and Computer Engg.
University of California San Diego
9500 Gilman Dr, La Jolla, CA 92093
`gkarumur@eng.ucsd.edu`

1 Baseline

The Baseline tagger used is a unigram model.

1.1 Description

Our baseline assumes the Markov property which suggests that the distribution for a random variable in the future depends solely only on its distribution in the current state, and none of the previous states have any impact on the future states. In a sense, it is just a Unigram tagger and hence doesn't consider the transition probabilities to model the distribution. For determining the Part of Speech tag, it only uses a single word. It tags each word with its most frequent tag, irrespective of context. It is also sometimes referred to as the most frequent tagger.

1.2 Word-Replacement Policy

I replaced all the words that appear less than 5 times and that belong to the tag-'O' in the training set to '_RARE_' and trained the new file again to collect the new counts.

1.3 Modifications to improve grouping of Rare words

Improvement 1: Since the ratio of the number of the words that appear less than 5 times is high, I replaced the words belonging to the tag- 'O' in the training set appearing only once with the symbol, '_RARE_' so that we still have an implementation of word classes for unseen words and not many words are switched to '_RARE_'. In this way, all the genes earlier identified as genes will be recognized as genes. Also, this method improves the F1 score over the dev set from 25 to 26 because we correctly identify even the genes that appear less than 5 times in the training set.

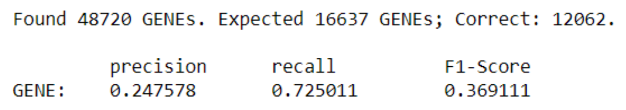
Improvement 2: From examining the training data that is provided, the structure of the formation of the word can help recognize if that word is a gene or not. Since, as a part of the above implementation of improvement 1, all new words are by default added to the 'O' tag, Following are some of the similarities exhibited by the words belonging to the 'I-GENE' tag to help identify new words that might fall into the 'I-GENE' category. There are obviously exceptions to the classes proposed but this shows better classification:
1.Acronyms and words formed with the combinations of capital letters and numbers(Examples from the train data: HSP70, HNF)
2.Words ending in '-ase' or '-ases'(Examples from the train data: phosphatase, reductases, acetyltransferase)
3.Words ending in '-or' or '-ors' or '-ory'(Examples from the train data: receptors, secretory)
4.Words ending in '-in' or '-gen'(Examples from the train data: cryoglobulin, fibrinogen, albumin)
5.Words ending in '-ica'(Examples from the train data: Brassica)

Given any word, if it's not in the train set, the first improvement classifies it as belonging to 'O'. The second improvement classifies it as belonging to 'I-GENE' if it falls in one of the categories or it belongs to 'O'.

1.4 Evaluation on Train and Dev set

1.4.1 Trainset

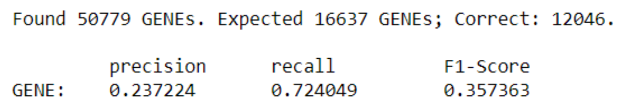
Baseline with only 1 word class for rare words (Improvement 1):



```
Found 48720 GENEs. Expected 16637 GENEs; Correct: 12062.

         precision      recall         F1-Score
GENE:    0.247578       0.725011       0.369111
```

Baseline with the listed word class for rare words (Improvement 2):

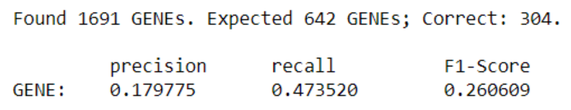


```
Found 50779 GENEs. Expected 16637 GENEs; Correct: 12046.

         precision      recall         F1-Score
GENE:    0.237224       0.724049       0.357363
```

1.4.2 Devset

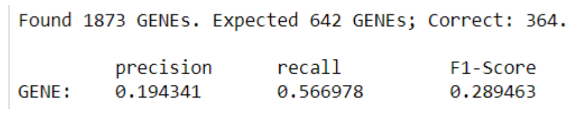
Baseline with only 1 word class for rare words (Improvement 1):



```
Found 1691 GENEs. Expected 642 GENEs; Correct: 304.

         precision      recall         F1-Score
GENE:    0.179775       0.473520       0.260609
```

Baseline with the listed word class for rare words (Improvement 2):



```
Found 1873 GENEs. Expected 642 GENEs; Correct: 364.

         precision      recall         F1-Score
GENE:    0.194341       0.566978       0.289463
```

1.5 Intuition for new word classes and Comparison

The dataset given for this tagging problem is highly specific for name-entity tagging. Our model has to learn to tag the words that are genes. Since, biological gene naming follows a protocol, word classes can be identified among the words that are genes. Basing on the same, above are the word classes identified by looking at the training dataset.

Both the methods are usually not great because it is quite possible for a single word to have a different part of speech tag in different sentences based on different contexts. And out model ignores the context completely. However, from the results of the dev set above, we see an improvement in the precision and recall and hence the F1 score using multiple classes for tackling the rare words. Both these improved rare word class handling methods show better F1 score when compared to the one with classifying rare words as the words appearing less than 5 times. The table below is a comparison of the F1 scores on the Dev set using different replacement policies.

Method	Baseline Replacement	Improvement 1	Improvement 2
F1 score	25.6	26.1	28.94

The train set F1 score using improvement 2 is lower probably indicates that the bias is lower and the data is not over-fitted to the training set.

2 Trigram HMM with Viterbi algorithm

2.1 Description

2.1.1 Motivation for HMMs

Hidden Markov models are generative models used for labelling sequence data. These models define the joint probability of a sequence of symbols and their labels (state transitions) as the product of the starting state probability, the probability of each state transition, and the probability of each observation being generated from each state. HMMs share the Markov chain's assumption, being that the probability of transition from one state to another only depends on the current state - i.e. the series of states that led to the current state are not used. They are also time invariant.

An HMM is desirable for this task of name entity tagging as the highest probability tag sequence can be calculated for a given sequence of word forms. This differs from other tagging techniques which often tag each word individually, seeking to optimize each individual tagging greedily without regard to the optimal combination of tags for a larger unit, such as a sentence. The HMM does this with the Viterbi algorithm, which efficiently computes the optimal path given the sequence of words forms.

2.1.2 Viterbi Algorithm

The Viterbi algorithm is a dynamic programming algorithm that efficiently computes the the most likely states of the latent variables of a Hidden Markov Model (HMM), given an observed sequence of emissions. For HMMs, the observed and unobserved variables are related through emission probabilities.

Naively computing the most likely path requires considering all possible paths through the state lattice and evaluating probabilities for each path. The problem with this approach is the exponential growth in the number of possible paths with respect to the number of time-steps. Instead, the Viterbi algorithm computes incremental path probabilities, working from left to right. Mathematically, we want to maximize the probability of a label sequence given a set of observations can be defined in terms of the transition probability and the emission probability given by:

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i)$$

Viterbi algorithm lets us compute a sub-problem which is:

$$r(y_1 \dots y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i)$$

The main idea behind the Viterbi Algorithm is that we can calculate the values of the term p(k, u, v) efficiently in a recursive, memoized fashion.

2.2 Implementation

2.2.1 HMM

A Hidden Markov model is implemented to estimate the transition and emission probabilities from the training data. Emission probabilities govern the distribution of the observed variable at a particular time given the state of the hidden variable at that time. The size of this set depends on the nature of the observed variable. In our case, they represent the probability of the given word corresponding to the given the tag. The transition probabilities control the way the hidden state at time t is chosen given the hidden state at previous timesteps. In out case, since our HMM model is a trigram model, transition probabilities represent the probability of seeing a given tag corresponding to the given two previous tags of earlier time steps.

2.2.2 Viterbi Algorithm

The Viterbi algorithm is used for decoding, i.e. finding the most likely sequence of hidden states for previously unseen observations (sentences). Consider a timestep t and a state q, in order to find the most probable path through q at t we only require the path of highest probability that reaches q. All other paths to q at time t can be discarded. This is because HMMs satisfy the Markov property, that is, the state at timestep t+1 depends only on timestep t and timestep t-1, and is independent of timesteps {1,2,. . . ,t - 2}. To incrementally calculate these probabilities, the Viterbi algorithm creates two in memory matrices P and Back. The elements of P are the probabilities of a state at a given point in time, and Back represents backpointers to the most likely previous state.

2.2.3 Pseudocode

I am considering a trigram HMM in which I would be considering all of the trigrams as a part of the execution of the Viterbi Algorithm. I start with the first trigram window from the first three words of the sentence but then the model would miss out on those trigrams where the first word or the first two words occurred independently. For that reason, I consider two special start symbols as * and so the first trigram I consider then would be (*, *, x1) and the second one would be (*, x1, x2). The end goal of the algorithm would be to estimate labels such that

$$\max_{y_1 \cdots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \max_{u \in \mathcal{K}, v \in \mathcal{K}} (\pi(n, u, v) \times q(STOP|u, v))$$

which establishes the recursive relation. Every sequence would end with a special STOP symbol. For the trigram model, we would also have two special start symbols "*" in the beginning. The algorithm for this task is presented below:

```
Input: a sentence $x_1 \dots x_n$, parameters $q(s|u, v)$ and $e(x|s)$.
Initialization: Set $\pi(0, *, *) = 1$, and $\pi(0, u, v) = 0$ for all $(u, v)$ such that $u \neq *$
or $v \neq *$.
Algorithm:

  • For $k = 1 \dots n$,

      – For $u \in \mathcal{K}$, $v \in \mathcal{K}$,

            $\pi(k, u, v) = \max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$
            $bp(k, u, v) = \arg\max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v))$

  • Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n, u, v) \times q(\text{STOP}|u, v))$

  • For $k = (n-2) \dots 1$,

            $y_k = bp(k+2, y_{k+1}, y_{k+2})$

  • Return the tag sequence $y_1 \dots y_n$
```

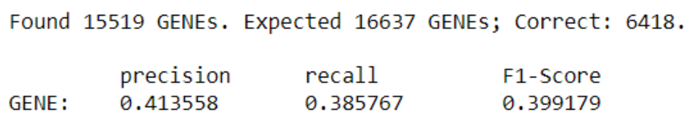
The Viterbi Algorithm not only helps us find each of cost values for all the sequences using the concept of dynamic programming, but it also helps us to find the most likely tag sequence given a start state and a sequence of observations. The algorithm has a step in which the back-pointers are updated after calculating the probabilities. This is to keep track of the label with the highest joint probability. After the end of sentence is reached, we back-track and assign labels to each of the words in our sentence following the updates in the bp table which is shown by the for loop that is present after reaching the end of the sentence in the algorithm above. The initialization step in the algorithm represented above serves as the base case because the probability of the start sentence given the bigram (*, *) is always 1. As we go over the first for loop for k, we calculate the probabilities and corresponding labels and save them in two dp tables. We refer to these tables, whenever we need the probability values corresponding to the k-1 elements. When we reach the end of the sentence, we calculate the probability for modeling STOP at the end of the sentence and obtain the last two labels. We then use the bp table and go back from n-2 to 1 to find the labels of the other words.

2.2.4 Modelling of rare words

Improvement 1: Same as above. Modelled words only from the 'O' class appearing once in the training set.
Improvement 2: Same as above. Modelled the following word classes along with improvement 1.
1.Acronyms and words formed with the combinations of capital letters and numbers
2.Words ending in '-ase' or '-ases'
3.Words ending in '-or' or '-ors' or '-ory'
4.Words ending in '-in' or '-gen'
5.Words ending in '-ica'
The above mentioned classes are to identify words belonging to the I-GENE class.Whenever a rare word belongs to one of these categories, The emission probability for the word from the I-GENE class is set to be high. Otherwise,the emission probability for the word from the 'O' class is set to be high.

2.2.5 Evaluation

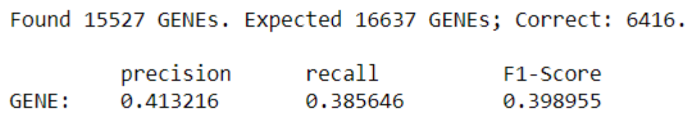
1.Train Set:
Viterbi algorithm with only 1 word class for rare words (Improvement 1):



```
Found 15519 GENEs. Expected 16637 GENEs; Correct: 6418.

         precision      recall          F1-Score
GENE:    0.413558       0.385767        0.399179
```

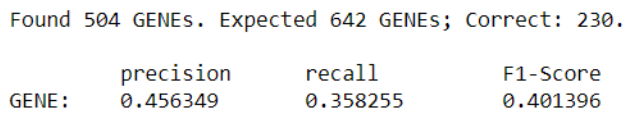
Viterbi algorithm with multiple word class for rare words (Improvement 2):



```
Found 15527 GENEs. Expected 16637 GENEs; Correct: 6416.

         precision      recall          F1-Score
GENE:    0.413216       0.385646        0.398955
```

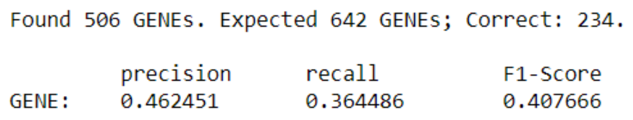
2.Dev Set:
Viterbi algorithm with only 1 word class for rare words (Improvement 1):



```
Found 504 GENEs. Expected 642 GENEs; Correct: 230.

         precision      recall          F1-Score
GENE:    0.456349       0.358255        0.401396
```

Viterbi algorithm with multiple word class for rare words (Improvement 2):



```
Found 506 GENEs. Expected 642 GENEs; Correct: 234.

         precision      recall          F1-Score
GENE:    0.462451       0.364486        0.407666
```

2.2.6 Comparisons

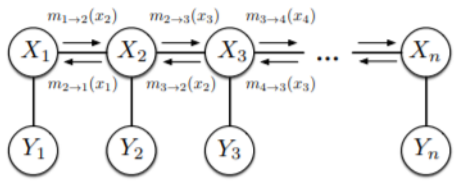
When compared to the baseline model, the F1 scores recorded on the train set and the dev set are higher for the Viterbi algorithm. We don't observe a lot of difference in the F1 score when we introduce multiple classes as before because, most of the genes are being identified based on the context and these additional classes do not identify any more genes that are not already recognized by the Viterbi algorithm. And so, the improvement seen is not significant. The table below is a comparison of the F1 scores on the Dev set using different replacement policies using the Viterbi algorithm.

Method	Baseline Replacement	Improvement 1	Improvement 2
F1 score	39.7	40.14	40.767

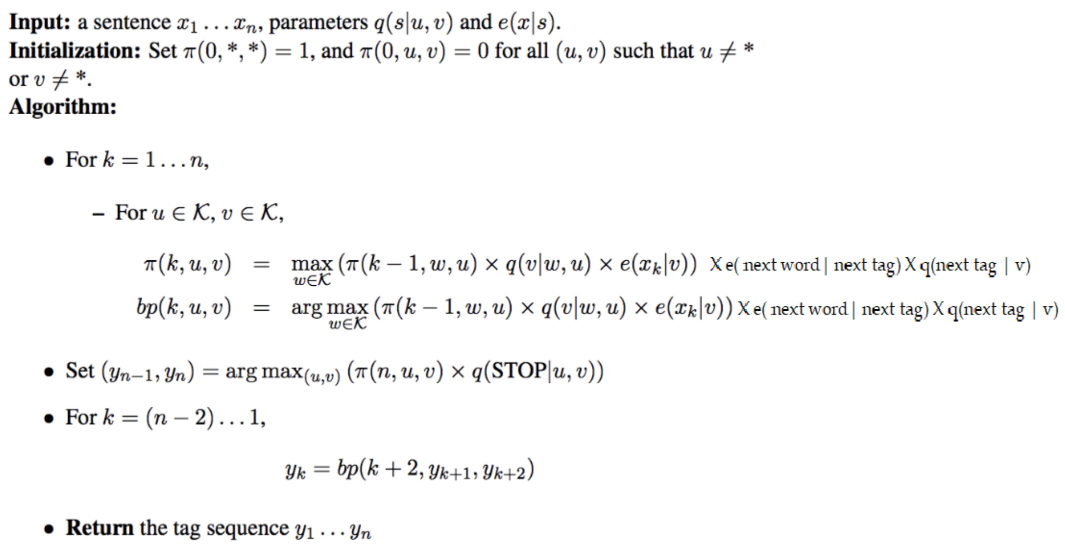
3 Extensions: Forward-Backward Algorithm

Since our HMM Viterbi model only considers the previous bi-grams while estimating the transition probabilities, it completely ignores the dependence of the word tags that follow the current word in the process of estimating the joint probability. As for the training set that is provided, count is non-zero for all possible tri-gram combinations. Therefore, I do not believe that smoothing will improve the current model's performance.

3.1 Description



As far as the implementation goes, I used a HMM with the Viterbi algorithm as before with the multiple classes implementation for dealing with the rare words. But, this HMM model is different from the one in the previous section. This model estimates the probabilities of the label of the given word depending on the labels of the previous two words and the label of the next word. I incorporate this improvement in the estimation step of the joint probability of the Viterbi algorithm. In a sense, This model is 4-gram model except that it is characterized not only by the words on the left of the word in a sentence but also the right. This implementation uses the ideas of the bi-directional RNN model in a HMM model.



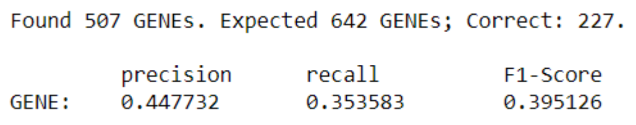
Input: a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
Initialization: Set $\pi(0,*,*) = 1$, and $\pi(0,u,v) = 0$ for all (u,v) such that $u \neq *$ or $v \neq *$.
Algorithm:

```
• For $k = 1 \dots n$,
    – For $u \in \mathcal{K}, v \in \mathcal{K}$,
        $\pi(k,u,v) = \max_{w \in \mathcal{K}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$ X e( next word | next tag) X q(next tag | v)
        $bp(k,u,v) = \arg\max_{w \in \mathcal{K}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$ X e( next word | next tag) X q(next tag | v)
• Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
• For $k = (n-2) \dots 1$,
        $y_k = bp(k+2, y_{k+1}, y_{k+2})$
• **Return** the tag sequence $y_1 \dots y_n$
```

The algorithm for the same is detailed above. The idea is from the Baum-Welch algorithm but Baum-Welch algorithm uses EM algorithm for the parameter estimation. Since, this is time consuming and computation intensive, and we already know the next word of the sentence, I used the emission probabilities of the next word to predict the tag of the next tag and used it as a part of the joint probability calculation. The pseudo code above explains the procedure and the differences between the trigram Viterbi model.

3.2 Evaluation

The following results are obtained after using this algorithm with multiple classes for dealing with unseen words on the Dev set. The table below shows the comparison of different approaches with the



```
Found 507 GENEs. Expected 642 GENEs; Correct: 227.

         precision      recall         F1-Score
GENE:    0.447732       0.353583       0.395126
```

best F1 score obtained using that method. (In all the 3 approaches, it was by using improvement 2)

Method	Baseline	HMM-Viterbi	Forward-Backward
F1 score	28.94	40.767	39.51

3.3 Analysis

The F1 score is lower when compared to the Viterbi algorithm. The following could be the possible reasons.
1. The labels might not depend on the labels of the words to the right of it in a sentence.
2. Estimation of the next tag associated with the next word could be incorrect. The method used above is different from the one used conventionally. It probably needs to be looked at like proposed with the EM algorithm and soft assignments which considers more possibilities. The approach followed calculates the next tag based on the emission probabilities of the next word. Since, there is no soft assignment that is followed, it makes this a rigid approach.
3. As we increase the n-gram dependency of the HMM model, the probabilities of the tags to be assigned get lower. Since this model is a 4-gram model, the contradictions in the assignment could be higher in number when compared to the 3-gram Viterbi model.