CSE 256 Assignment 3

Ranak Roy Chowdhury, A53317421

November 13, 2019

1 Baseline

1.1 Description

The Baseline Tagger computes the probability of a word given each tag from the set of all possible tags. The tag assigned to the word is the tag with the highest probability.

1.2 Implementation Details

At first, I had to replace every word in the training set that appeared less than 5 times by the token '_RARE_'. From this new training set, I recompiled the count file which stores the number of time a particular tag has been associated with a word and the frequency of trigrams, bigrams and unigrams. Then for each word in the validation file, if the word does not exist in the count file, it had to be replaced by the token '_RARE_'. Then, the emission probabilities of each word, namely $P(word|`O')$ and $P(word|`I-GENE')$ was computed. The tag assigned to each word was the tag that gave the highest emission probability.

2 Informative Word Class 1

2.1 Motivation

The total number of words that appeared less than 5 times in the training corpus was 37513, which were initially replaced by the token '_RARE_'. Here are a couple of heuristics that I tried to estimate the distribution of the characteristics of these 37513 words:

1. Class 1: Number of words with all letters in uppercase - 6866
2. Class 2: Number of words that start with an uppercase letter - 14658
3. Class 3: Number of words that end with an uppercase letter - 5939

From the above distribution, it can be seen that among classes 1, 2, and 3, class 2 had the highest number of words. Therefore, if I consider this class, then approximately 39% (14658/37513) of the words will be in class and the rest will be in '_RARE_' class. This will be a much more balanced class distribution than if I consider any of the other two classes, namely Class 1 and 3.

2.2 Evaluation and comparison of the baseline models

Table 1: Validation set performance of the baseline tagger for different setup of the informative classes

Informative Classes	**Found**	**Expected**	**Correct**	**Precision**	**Recall**	**F1-Score**
Baseline (_RARE_)	2669	642	424	0.158861	0.660436	0.256116
Start with uppercase	2669	642	424	0.158861	0.660436	0.256116

As evident from Table 1, the values are all the same. The reason is that for this informative classes, $P(Uppercase|\text{“}O\text{”}) > P(Uppercase|\text{“}I-GENE\text{”})$. So the word that gets replaced with the informative class is always tagged with 'O' like it did when there was only one informative class '_RARE_'. Hence, there was no improvement on the precision, recall or F1 score.

3 Informative Word Class 2

3.1 Motivation

The total number of words that appeared less than 5 times in the training corpus was 37513, which were initially replaced by the token '_RARE_'. Here are a couple of heuristics that I tried to estimate the distribution of the characteristics of these 37513 words:

1. Class 1: Number of words with digits - 777
2. Class 2: Number of words that start with a digit - 1198
3. Class 3: Number of words that end with a digit - 3731

From the above distribution, it can be seen that among classes 1, 2, and 3, class 3 had the highest number of words. Therefore, if I consider this class, then approximately 9.9% (3731/37513) of the words will be in class and the rest will be in '_RARE_' class. This will be a much more balanced class distribution than if I consider any of the other two classes, namely Class 1 and 2.

3.2 Evaluation and comparison of the baseline models

Table 2: Validation set performance of the baseline tagger for different setup of the informative classes

Informative Classes	**Found**	**Expected**	**Correct**	**Precision**	**Recall**	**F1-Score**
Baseline (_RARE_)	2669	642	424	0.158861	0.660436	0.256116
Ends with digit	2669	642	424	0.158861	0.660436	0.256116

As evident from Table 2, the values are all the same. The reason is that for this informative classes, $P(Digit|\text{“}O\text{”}) > P(Digit|\text{“}I-GENE\text{”})$. So the word that gets

replaced with the informative class is always tagged with 'O' like it did when there was only one informative class '_RARE_'. Hence, there was no improvement on the precision, recall or F1 score.

4 Trigram HMM

4.1 Purpose of the Viterbi Algorithm

In sequence labelling problems, let S be the entire sequence and s be the length of the sequence. Let V be the set of possible labels that can be associated with each tag and let v be the size of v. Then the pros and cons of the greedy, brute force and dynamic programming approach are as follows:

- Brute Force: Considers all possible sequences. The number of such sequences would be v^s. Although this method will give the best possible sequence of labels but to consider all possible sequences is computationally expensive.
- Greedy: The greedy algorithm is fast. We consider the set of all possible tags for each word in the sequence. Hence, the computational complexity would be $v \times s$ which is less than v^s. However, this is suboptimal because it commits to a tag before seeing subsequent tags. It could be the case that ALL possible next tags have low transition probabilities. E.g., if a tag is unlikely to occur at the end of the sentence, that is disregarded when going left to right. So subsequent words have no effect on each decision, so the result is likely to be suboptimal.
- Dynamic Programming: Gives an optimal global solution, but requires more computation than greedy. It postpones decision about any tag until we can be sure it's optimal. It takes about $v^2 s$ time.

4.2 Specifics of the implementation

The basecase of the Trigram HMM is given by $\Pi(0, *, *) = 1$. For the first word, I compute $transition(*, *, O) * emission(firstWord|O)$ and $transition(*, *, I-GENE) * emission(firstWord|I-GENE)$. For the second word, I compute $transition(*, O, O) * emission(firstWord|O)$, $transition(*, I-GENE, O) * emission(firstWord|O)$, $transition(*, O, I-GENE) * emission(firstWord|I-GENE)$, and $transition(*, I-GENE, I-GENE) * emission(firstWord|I-GENE)$. So for each tag in the second word, there are two possible paths, one that goes through tag O in the first word and the other that goes through tag I-GENE in the first word. I computed both of them and retained the maximum of the two. Then for each subsequent word in the sentence until the STOP symbol, I compute the following for the $t-th$ word in the sequence, $emission(word_t|O) * transition(O, O, O) * probabilityO(t-2)$, $emission(word_t|O) * transition(O, I-GENE, O) * probabilityO(t-2)$, $emission(word_t|O) * transition(I-GENE, O, O) * probabilityI-GENE(t-2)$, and $emission(word_t|O) * transition(I-GENE, I-GENE, O) * probabilityI-GENE(t-2)$. I take the maximum and store it in $probabilityO(t)$. If the maximum comes from the first or second values, I store the tag "O" in $tagO(t)$, otherwise I store the tag "I-GENE" in $tagO(t)$. Similarly, I compute $emission(word_t|I-GENE) * transition(O, O, I-GENE) * probabilityO(t-2)$, $emission(word_t|I-GENE) * transition(O, I-GENE, I-GENE) *$

$probabilityO(t-2)$, $emission(word_t|I-GENE)*transition(I-GENE, O, I-GENE)*$ $probabilityI-GENE(t-2)$, and $emission(word_t|I-GENE)*transition(I-GENE, I-GENE, I-GENE)*probabilityI-GENE(t-2)$. I take the maximum and store it in $probabilityI-GENE(t)$. If the maximum comes from the first or second values, I store the tag "O" in $tagI-GENE(t)$, otherwise I store the tag "I-GENE" in $tagI-GENE(t)$. $probabilityO(t)$ and $probabilityI-GENE(t)$ stores the aggregate probability through the sequence till the $t-th$ word. And $tagO(t)$ and $tagI-GENE(t)$ stores the tag at the $(t-2)-th$ word that give rise to the maximum probability value at the $t-th$ word. This helps when I will be backtracking to find the tag sequence that gave the maximum probability value at the STOP symbol. When I encounter the STOP symbol, I similarly compute all the probabilities as above. But additionally I find the maximum of the two probabilities, one for tag "O" and the other for tag "I-GENE". I store the trigram that gave the maximum probability value for the STOP symbol. This trigram gives the tag sequence for the last two words and the STOP symbol in the sentence. So if there are n words in the sequence, excluding the STOP symbol, then $tagO(t)$ and $tagI-GENE(t)$ gives the predicted tag at the $(t-2)th$ word. I traverse through the entire sequence in the opposite order based on the tag given by either $tagO(t)$ or $tagI-GENE(t)$. This backtracking generates the desired predicted tag sequence in the reverse order.

4.3 Evaluation and Comparison of HMM Tagger with Baseline Tagger

Table 3: Validation set performance of the Baseline and HMM tagger for the '_RARE_' informative class

Tagger Used	Found	Expected	Correct	Precision	Recall	F1-Score
Baseline	2669	642	424	0.158861	0.660436	0.256116
Trigram	2317	642	390	0.168321	0.607477	0.263603

As expected, the Trigram HMM outperforms the Baseline Tagger in both Precision and F1 score as can be seen in Table 3. This is because whereas the Baseline Tagger just considers the emission probabilities, the Trigram HMM considers the transition probabilities, emission probabilities and the previously computed probabilities which propagate through the whole sequence. As a result, the Trigram HMM takes the context of a word with respect to the sequence into consideration before computing the probabilities of each possible tag that can be associated with a word.

5 Evaluation of HMM Tagger on Informative Word Classes

5.1 Evaluation of HMM Tagger on Class 1: Uppercase

As evident from Table 4 and the second row of Table 3, the HMM Tagger with the Informative Class 1: Uppercase, outperforms the HMM Tagger with no informative class in both precision and recall.

Table 4: Validation set performance of the HMM tagger on Class 1: Uppercase

Informative Classes	Found	Expected	Correct	Precision	Recall	F1-Score
Start with uppercase	1742	642	344	0.197474	0.535826	0.288591

5.2 Evaluation of HMM Tagger on Class 2: Digit

Table 5: Validation set performance of the HMM tagger on Class 2: Digit

Informative Classes	Found	Expected	Correct	Precision	Recall	F1-Score
End with digit	1894	642	349	0.184266	0.543614	0.275237

As evident from Table 5 and the second row of Table 3, the HMM Tagger with the Informative Class 2: Digit, outperforms the HMM Tagger with no informative class in both precision and recall.

6 Comparative Analysis

Table 6: Validation set performance of the HMM tagger on Class 1: Uppercase and Class 2: Digit

Informative Classes	Found	Expected	Correct	Precision	Recall	F1-Score
Start with uppercase + End with digit	1790	642	360	0.201117	0.560748	0.296053

As evident from the Table 6, when we add the two informative classes (Class 1: Uppercase and Class 2: Digit), the precision and F1 score improves on when we used a single informative class in isolation. Moreover the precision and F1 score in Table 4 are better than those in Table 5. Therefore, the informative class which replaces words starting with uppercase letters outperforms the one that replaces words ending with digits. As shown in the previous section, the number of words starting with uppercase letters exceeds the number of words that end with digits. As a result, more words can be grouped into the uppercase class, giving a more balanced class distribution than when words are grouped according to whether they end with a digit.