Programming Assignment 3 : Sequence Tagging CSE 256: Statistical NLP: Spring 2019

University of California, San Diego

Wei-Cheng Huang

1. Baseline

(1-a) Baseline Description: Our Baseline model first train on training data by computing emission parameters of each word: $y^* = argmax_y\ e(x|y)$, where x is the word, and y is the tag. However, we also have to predict probabilities for words in the dev set that do not occur in the training data. Our approach is to treat these unseen words as the "_RARE_" class, whose total word counts are less than 5. Then retrain the model by computing new emission parameters. We'll use the y^* to determine the tag each word, including words in RARE class.

(1-b) Evaluation of Baseline on dev set

Found 2669 GENEs. Expected 642 GENEs; Correct: 424.			
	precision	recall	F1-Score
GENE	0.158861	0.660436	0.256116

(1-c) Informative word classes :

Below are the word classes I chose. Based on the class information I observed, I regrouped them into:

(A) Classes that contain most rare words .I think they might make great influence in rare word classification, since they accounts for a great portion in rare class.

(B) Classes whose counts in two tags are more than 10 times different, since this class could make great difference if the class tag is coincidentally different from Rare class.

(C) Classes whose tag is different from _RARE_ class. Extracting the words whose tags are likely different from RARE class could potentially make a big help, since they were easily classified into wrong tag if they were just in the RARE word class.

Class	Examples	Counts: "O"	Counts : "I-GENE"	Tag	P(x| "O")	P(x|"I-GENE")	Class A	Class B	Class C
Number	1977, 20881	1141	39	O	0.0033	0.00094		✓	✓
OneNum	FT4I, MH2	1631	3526	I-GENE	0.004	0.08	✓		
Capital	SGPT, FLA	2698	1411	I-GENE	0.0078	0.034	✓		
FirstCap	Takayasu, Markovich	5451	1480	O	0.015	0.036	✓		✓
Contains Punctuation	++- , "? , -/	105	11	O	0.0003	0.00026			✓
LongWords	methazolamide	5871	532	O	0.017	0.012	✓	✓	✓
ContainDot	(. , ./	74	1	O	2E-04	2.43E-05			✓
OneStr	$, #, u ,t	11	4	I-GENE	3.18E-05	9.73E-05			
Rare	Other rare words	11799	1728	I-GENE	0.034	0.042	✓		

(1-d) Evaluation of new baseline models on train and dev set

Method	Dataset	Precision	Recall	F1-Score
(A)	Dev	0.1642	0.6339	0.2609
	Train	0.181057	0.688706	0.286733
(B)	Dev	0.1661	0.6339	0.263
	Train	0.180833	0.687864	0.286379
(C)	Dev	0.1648	0.6386	0.2620
	Train	0.181057	0.688706	0.286733
All word classes	Dev	0.1648	0.6386	0.2619
	Train	0.181057	0.688706	0.286733
No word classes	Dev	0.1589	0.6604	0.2561
	Train	0.172470	0.701388	0.276861

(1-e) Comparison based on these result

From the result above, we can see that including any word class could obtain better result than only using _RARE_ class. Moreover, based on the F1-Score of dev dataset, we can see that B class got the best result, which is possibly because the ”NUMBER” and “Longword” classes were originally classified into the same tag as _RARE_, which is the wrong tag of them. However, this result is against my intuition, since I though C could reach the highest accuracy because all classes’ tag are different from _RARE_, which means that they were potentially be considered as I-GENE by mistake by our model. One possible explanation against my expectation is that those additional minor cases are actually classified correctly more than I expected.

On the other hand, Comparing the precision and recall value of Class (A), and (B), though the F1-score of (A) is slightly lower than (B), the recall value are almost the same, while B’s precision is higher than A’s precision, which means B has better capability to predict the tag right.

By comparing dev and train set, apparently training set has better performance than dev set, and adding word class could increase the accuracy of training class. However, the best word classes for dev set doesn’t match the best of dev set for sure.

2. Trigram HMM

(2-a) Description of Viterbi Algorithm

Purpose: Viterbi algorithm is an approach to find the most likely sequence of hidden states and generate most likely tag sequence of a given sentence.

Goal : find $argmax\ p(x_1 \dots x_n, y_1 \dots y_{n+1})$, where the $argmax$ is taken all over the tag sequence $y_1 \dots y_n$ and $y_{n+1} = STOP$, and $x_1 \dots x_n$ is the word sequence. In the tag sequence, we assume that $y_0 = y_{-1} = *$, and that $p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i \mid y_{i-2}, y_{y-1}) \prod_{i=1}^{n} e(x_i \mid y_i)$. The reason why we only use y_i, y_{i-1}, y_{i-2} per iteration is because trigram parameters depend only on tag subsequences of length three. Here, we want to use recursive to compute the joint probability of the observation sequence together with the best state sequence, with dynamic programming to store maximum probability in the dictionary for next iteration usage. For each iteration, we need to store max possibility and tag of max possibility given adjacent two tags. Lastly, we use the last

“STOP” tag to compute the most possible combination of the last two tags of the sequence, then retrieve the most possible tag reversely from the stored tag.

(2-b) Evaluation on dev set and Comparison with baseline.

Comparison of basline and Trigram HMM			
	precision	recall	F1-Score
Baseline	0.1588	0.6604	0.2561
Trigram HMM	0.5415	0.3146	0.3980

We can see that the Trigram HMM performs much better than the Baseline, according to the precision and recall. Trigram can get more than 50 % accuracy given the possible I-GENE tag.

(2-c) New informative class

Here, I introduced the same informative class combination to see comparison between HMM Trigram and baseline model. Below are the word classes I chose. Based on the class information I observed, I regrouped them into:

(A) Classes that contain most rare words , since I think they might make great influence in rare word classification.

(B) Classes whose counts in two tags are more than 10 times different, since this class could make great difference if the class is coincidentally different from Rare class.

(C) Classes whose tag is different from _RARE_ class. Extracting the words whose tags are likely different from RARE class could potentially make a big help, since they were easily classified into wrong tag if they were just in the RARE word class.

Class	Examples	Counts: “O”	Counts : “I-GENE”	Tag	P(x| “O”)	P(x|”I-GENE”)	Class A	Class B	Class C
Number	1977, 20881	1141	39	O	0.0033	0.00094		✓	✓
OneNum	FT4I, MH2	1631	3526	I-GENE	0.004	0.08	✓		
Capital	SGPT, FLA	2698	1411	I-GENE	0.0078	0.034	✓		

Class	Examples	Counts: "O"	Counts : "I-GENE"	Tag	P(x| "O")	P(x|"I-GENE")	Class A	Class B	Class C
FirstCap	Takayasu, Markovich	5451	1480	O	0.015	0.036	✓		✓
ContainsPunctuation	++- , "? , -/	105	11	O	0.0003	0.00026			✓
LongWords	methazolamide	5871	532	O	0.017	0.012	✓	✓	✓
ContainDot	(. , ./	74	1	O	2E-04	2.43E-05			✓
OneStr	$, #, u ,t	11	4	I-GENE	3.18E-05	9.73E-05			
Rare	Other rare words	11799	1728	I-GENE	0.034	0.042	✓		

(2-d) Evaluate HMM model with informative classes on train and dev sets

Method	Dataset	Precision	Recall	F1-Score
(A)	Dev	0.5302	0.3411	0.4151
	Train	0.55520	0.3890	0.45748
(B)	Dev	0.54987	0.3520	0.4292
	Train	0.5631	0.3881	0.4595
(C)	Dev	0.5314	0.3426	0.4166
	Train	0.5550	0.3888	0.4573
No word classes	Dev	0.5415	0.3146	0.3980
	Train	0.5624	0.3504	0.4318

(2-e) Comparison

We can see from the above result that adding informative word classes contribute a lot to the performance of our Trigram HMM model. The reason why it works much better than Baseline model is because it considers not only the previous content by sequence, which makes more sense in terms of the relation between adjacent tags. Moreover, surprisingly, (B) performs best among the word classes but using least word class. One possible reason is that Trigram

HMM already uses word sequence to tag word sequence, so it's easier to overfit the data. Therefore, using too many word classes would undermine the performance.

3. Extension : 4-Gram HMM

To construct a 4-Gram HMM, we need to compute 4-gram translation parameter which is $q(y_i \mid y_{i-3}, y_{i-2}, y_{i-1}) = \frac{count(y_{i-4}, y_{i-3}, y_{i-2}, y_{i-1})}{count(y_{i-3}, y_{i-2}, y_{i-1})}$, and redefine algorithm as the pseudo code below:

Set: $\pi(0, *, *, *) = 1$

Definition: $S_{-2} = S_{-1} = S_0$, $S_k = S \; for \; k \in \{1...n\}$

Algorithm:

For $k = 1...n$

For $\omega \in S_{k-2}$, $\mu \in S_{k-1}$, $\nu \in S_k$

$$\pi(k, \omega, \mu, \nu) = \max_{t \in S_{k-3}} (\pi(k-1), t, \omega, \mu) \times q(\nu \mid t, \omega, \mu) \times e(x_k \mid \nu)$$

$$bp(k, \omega, \mu, \nu) = arg \max_{t \in S_{k-3}} (\pi(k-1), t, \omega, \mu) \times q(\nu \mid t, \omega, \mu) \times e(x_k \mid \nu)$$

Set $(y_{n-2}, y_{n-1}, y_n) = argmax_{\omega,\mu,\nu} \pi(n, \omega, \mu, \nu) \times q(STOP \mid \omega, \mu, \nu)$

For $k = (n-3) \ldots 1$, $y_k = bp(k+3, y_{k+1}, y_{k+2}, y_{k+3})$

Return the tag sequence $y_1, \ldots, y_n$

I implemented 4-gram HMM algorithm above but I couldn't successfully implement it. I got the result below instead, which I know it is the wrong result but I couldn't find the bug yet. If my intuition is right, 4-gram model should have worked similarly more or less as the trigram model. Since there is no certainty that more grams would obtain better result.

```
Found 643 GENEs. Expected 642 GENEs; Correct: 95.

         precision      recall         F1-Score
GENE:    0.147745       0.147975       0.147860
```