Xinmeng Li

I. Baseline

- The baseline model predicts the tag of an individual word without any usage of context information. The step of baseline model without rarewords replacement to tag a word x is that 1. For all possible tags t in tags set S, calculate the emission parameter value e(x|t), where e(x|t) = count(t->x)/count(t). We define count(t->x) as the number of times that x was tagged as t in the training data, count(t) as the number of occurrences of t in the training data. So e(x|t) is the possibility that the tagged word is x when t is given. 2. Compare all e(x|t) across all tags t, and pick the t with the highest e(x|t), because the highest e(x|t) means that given this t, the possibility that x is tagged as t is high.

- Since there are unseen words in the test set, we need to map these unseen words into some word classes. The first strategy, also the default one, is to map unseen words into '_RARE_', which represents words occurred less than 5 times. The result for count<5 on dev dataset is the same as the writeup.
  On dev set : Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

	precision	recall	F1-Score
GENE:	0.158861	0.660436	0.256116

  On train set : Found 67658 GENEs. Expected 16637 GENEs; Correct: 11669.

GENE:	0.172470	0.701388	0.276861

  The performance on training dataset is better than that on the dev set, but their precision, recall and f1-score actually quite close.
  Since I am curious about the relationship of word occurrence threshold and the result, I also run this strategy for the case that count < 2,3,4 on dev set.
  <5: Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

	precision	recall	F1-Score
GENE:	0.158861	0.660436	0.256116

  < 4: Found 2608 GENEs. Expected 642 GENEs; Correct: 423.

GENE:	0.162193	0.658879	0.260308

  < 3: Found 2515 GENEs. Expected 642 GENEs; Correct: 422.

GENE:	0.167793	0.657321	0.267342

  <2: Found 2389 GENEs. Expected 642 GENEs; Correct: 428.

GENE:	0.179154	0.666667	0.282415

  It is obvious that f-1 score and precision increases as the threshold decreases. However, the recall score fluctuates, which indicates that as threshold decreases, the correct prediction rate of predicted 'I-GENE' words increases, but the correctly predicted rate of original 'I-GENE' words fluctuates. The increase of precision could be the reason that as threshold goes down, less infrequent words are mapped to '_RARE_',(the evidence is the decreased number of found genes) and then the emission value of these infrequent words (e.g. 2 < count < 5) are calculated more precisely.

- First design methods: After sorting the words that are tagged with 'I-GENE' , I observe that many words tagged by 'I-GENE' contain number. I am not a bio-majored student, so I am not familier with biological term feature. I guess that words with number might be codes of some DNA or proteins. To check if this guess is valid, I count words containing number and without number in 'I-GENE' and 'O' respectively. There are 2413 and 3023 words with number in 'O' and 'I-GENE' respectively, and there are 22682 and 5388 words without number in 'O' and 'I-GENE' respectively. The rate of word with number in all of words in 'O' is 2413/(2413+22682) = 0.09615461247260411, and the rate of that in words in 'I-GENE' is 0.3594102960408988, which

is quite large. You might feel unreasonable that there are so many words in 'I-GENE' contains number, because if you just manually look at the training file, you don't believe that words with number could have 35.9% in 'I-GENE'. Here you should note that I count every different word for 1 equally but not for their weights/occurrence, since we are designing strategy for unseen words, and we should pay attention to the feature of all words equally.
The result of this design method on dev set:
Found 2454 GENEs. Expected 642 GENEs; Correct: 426.

	precision	recall	F1-Score
GENE:	0.173594	0.663551	0.275194

On training set:
Found 56184 GENEs. Expected 16637 GENEs; Correct: 12184.

	precision	recall	F1-Score
GENE:	0.216859	0.732344	0.334629

Compared with the default rare words strategy ,the precision, recall and f1-score of both dev and training prediction are improved.

- Second design methods: I looked through all of the words tagged by 'I-GENE' and picked 70 prefix. When I select the prefix, I first see in how many words does this prefix exist. The more different words containing the same prefix, the more likely this prefix is indicative. Similar to the first strategy, note that here I did not count the tagged occurence of each word, since this is the design methods for unseen words and words with more occurence should not be weighted too much. Then I check the uniqueness of the prefix, because if the prefix is common, it appears many times both in words of 'I-GENE' and 'O', then the prefix still does not provide indicative information. Usually, the longer the prefix, the more unusual it is. Also, capitalized prefix should also appears infrequently.
  The 70 prefix is just my guess, so I checked their validity and filtered by comparing the percentage of 'I-GENE' with that of 'O' among words containing the prefix. For example, assume that there are words ('study','O'),('studied','I-GENE'),('studying','I-GENE') starting with 'stud', then the percentage of 'O'/that of 'I-GENE' = 1/2. Prefix that has this division number greater than 1 should be removed, since although I noticed that it appeared a lot in the 'I-GENE' words, there are more words containing it in the 'O' set, and thus it does not provide indicative information anymore.
  The result of this design method on dev set:
  Found 2247 GENEs. Expected 642 GENEs; Correct: 423.

	precision	recall	F1-Score
GENE:	0.188251	0.658879	0.292835

  On training set:
  Found 49043 GENEs. Expected 16637 GENEs; Correct: 12053.

	precision	recall	F1-Score
GENE:	0.245764	0.724470	0.367022

  Although the recall value of this strategy is lower than that of the first strategy, the difference is quite small and almost can be neglected. The precision and f1-score are obviously better than These of the first strategy. So for the part I, the second strategy is better.

- Third design methods: I am curious about the result of combining the first design and the second method. When calculating the emission parameter, I first checked if the word contains the prefix, and if not, check if the word contains number.
  The result of this design method on dev set:

Found 2465 GENEs. Expected 642 GENEs; Correct: 425.

	precision	recall	F1-Score
GENE:	0.172414	0.661994	0.273576

On training set:

Found 56494 GENEs. Expected 16637 GENEs; Correct: 12175.

	precision	recall	F1-Score
GENE:	0.215510	0.731803	0.332964

The precision, recall and f1-score of the combination are lower than these two strategies, probably because it has more word classes and thus mapped more words to 'I-GENE'. For example, on the training set, the combination strategy found 56494 genes and get 12175 correct, but the prefix strategy found 49043 genes and get 12053 correct. Obviously, 56494-49043 is much more than 12175-12053.

II. Trigram HMM

- HMM is generative because it learns joint probability distribution related to words and tags from the training data. Suppose that the input sentence is x1,...,xn, then the tag sequence with the highest probability is the one which has the largest joint probability with x1,...,xn. To represent this by formula is that t1,...,tn = argmax(t1,...,tn) p(x1,...xn,t1,...,tn), where tag at position i, ti, belongs to the possible tag set at position i, Si. Computing p(x1,...xn,t1,...,tn) by the chain rule and second-order markov process, we get p(x1,...xn,t1,...,tn) = q(STOP|tn-1,tn) * q(t1|t-1,t0) * ...* q(tn|tn-2,tn-1) * e(x1|t1) *...* e(xn|tn). In implementation, I use q(ti|ti-1,ti-1) = count(ti-2,ti-1,ti)/ count(ti-2,ti-1), which is the trigram MLE. Also build a dynamic programming to store pi(k,u,v) = max(q(t1|t-1,t0) * … * q(tk|tk-2,tk-1) *e(x1|t1) *...* e(xk|tk)) where tk-1 = u, tk = v.
- Since brute force is very inefficient, to use dynamic programming, we use Viterbi algorithm
  Here are steps for Viterbi algorithm that I have implemented for this assignment: 1. Base case: pi(0,'*','*') = 1, pi(1,'*','O') = pi(0,'*','*')*q('O'|'*','*')*e(x1|'O'), pi(1,'*','I-GENE') = pi(0,'*','*') * q('I-GENE' |'*','*') * e(x1|'I-GENE'). Note that the tag at position -1 and 0 (i.e. k=-1,0) must be '*'.
  2. For k=2,...,n,

```
For u in ['O','I-GENE']
    For v in ['O','I-GENE']
        if k == 2:
            pi[(k,u,v)] = pi[(1,'*',u)]*q(v|'*',u)*e(x2|v)
            bp[(k,u,v)] = '*'
        else:
            pi1 = pi[(k-1,'O',u)]*q(v|'O',u)*e(xk|v)
            pi2 = pi[(k-1,'I-GENE',u)]*q(v|'I-GENE',u)*e(xk|v)
        # Save the tag at k-2 position such that  pi[(k,u,v)] is the maximum into
        #BP chart
            if(pi1>=pi2):
                pi[(k,u,v)] = pi1
                bp[(k,u,v)] = 'O'
            else:
                pi[(k,u,v)] = pi2
                bp[(k,u,v)] = 'I-GENE'
```

  3. for u in ['O','I-GENE']:

```
    for v in ['O','I-GENE']:
        pi1 = pi[(n,u,v)]*prob('STOP'|u,v)
```

  Compare pi1 here and get the maximum pi1, then tn-1=u and tn = v where u,v has the maximum pi1.

4. Since tn and tn-1 have been decided, we can get tn-2 by getting access to bp(n,tn-1,tn). Repeat this process until we get t1 and thus get the whole tag sequence. Every tag tk, k<=n-2, in this resulted tag sequence gives the maximum probability that tags at position k+1, k+2 are tk+1, tk+2.

- Run HMM with the rare words (count < 5) strategy, the results are:
  On dev set: Found 374 GENEs. Expected 642 GENEs; Correct: 202.

	precision	recall	F1-Score
GENE:	0.540107	0.314642	0.397638

  On training set: Found 10438 GENEs. Expected 16637 GENEs; Correct: 5835.

	precision	recall	F1-Score
GENE:	0.559015	0.350724	0.431025

  Obviously, the performance is much better than the result in part I.
  Similar as part I, I also compare the performance across difference threshold for rare words on dev set.
  Count <4
  Found 389 GENEs. Expected 642 GENEs; Correct: 212.

	precision	recall	F1-Score
GENE:	0.544987	0.330218	0.411251

  Count < 3
  Found 410 GENEs. Expected 642 GENEs; Correct: 223.

	precision	recall	F1-Score
GENE:	0.543902	0.347352	0.423954

  count<2
  Found 436 GENEs. Expected 642 GENEs; Correct: 234.

	precision	recall	F1-Score
GENE:	0.536697	0.364486	0.434137

  The f1-score still increases as the threshold value decreases. I noticed an interesting thing that here the recall increases and precision fluctuates as threshold goes down, which is opposite to that in part I. This change is because HMM is more complicated and the emission value is not the only determinant for tag anymore.

- I also perform the same strategies that I have mentioned in part I. Here are the results:
  Word with Number
  On dev set: Found 265 GENEs. Expected 642 GENEs; Correct: 148.

	precision	recall	F1-Score
GENE:	0.558491	0.230530	0.326351

  On training set: Found 12646 GENEs. Expected 16637 GENEs; Correct: 7566.
  GENE: 0.598292 0.454769 0.516750
  Prefix
  On dev set: Found 248 GENEs. Expected 642 GENEs; Correct: 145.

	precision	recall	F1-Score
GENE:	0.584677	0.225857	0.325843

  On training set: Found 15615 GENEs. Expected 16637 GENEs; Correct: 9924.
  GENE: 0.635543 0.596502 0.615404
  Combination
  On dev set: Found 269 GENEs. Expected 642 GENEs; Correct: 150.

	precision	recall	F1-Score
GENE:	0.557621	0.233645	0.329308

On training set: Found 12629 GENEs. Expected 16637 GENEs; Correct: 7507.
GENE: 0.594426 0.451223 0.513019

- Poor performance analysis: I observe that although the precision of these strategies is higher than rare words replacement, the f1-score of these design methods is lower than that of rare words replacement on dev set, because it has a very low recall (~0.23). However, the recall value on training dataset improved a lot. It is interesting that the recall my design methods swings between much lower and much higher on dev set and training set.

I use a sentence from dev to show the difference between rare words method and prefix method:
['Therefore',
 ',',
 'we',
 'suggested',
 'that',
 'both',
 'proteins',
 'might',
 'belong',
 'to',
 'the',
 'PLTP',
 'family',
 '.']
From the following, I observe that the pi value becomes 0 easily for the prefix method, and since I am using recursive method, so if the pi value of previous position is 0, then it will make its product, which is pi value in later position becomes 0. This is not a long sentence, so the prediction result for these two methods are the same: ['O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O', 'O']. However, for longer sentences that are common in the dev dataset, the 0 pi value will provide little indicative information for the later position, and thus the accuracy decreases. The prefix method works on training set probably because I extracted prefix based on the training data, so it still can provide indicative information. In dev set, there are so many unseen words, and the prefix method did not work well for providing indicative information.
Pi table for prefix method:
{(0, '*', '*'): 1, (1, '*', 'O'): 0.00010720660160867851, (1, '*', 'I-GENE'): 0.0, (2, 'O', 'O'): 1.4848596405018943e-07, (2, 'O', 'I-GENE'): 7.696410643338835e-09, (2, 'I-GENE', 'O'): 0.0, (2, 'I-GENE', 'I-GENE'): 0.0, (3, 'O', 'O'): 1.897575232826306e-10, (3, 'O', 'I-GENE'): 0.0, (3, 'I-GENE', 'O'): 4.29095968973283e-12, (3, 'I-GENE', 'I-GENE'): 0.0, (4, 'O', 'O'): 4.016255338835847e-14, (4, 'O', 'I-GENE'): 0.0, (4, 'I-GENE', 'O'): 0.0, (4, 'I-GENE', 'I-GENE'): 0.0, (5, 'O', 'O'): 3.1053665020140405e-16, (5, 'O', 'I-GENE'): 0.0, (5, 'I-GENE', 'O'): 0.0, (5, 'I-GENE', 'I-GENE'): 0.0, (6, 'O', 'O'): 4.683995617178435e-19, (6, 'O', 'I-GENE'): 0.0, (6, 'I-GENE', 'O'): 0.0, (6, 'I-GENE', 'I-GENE'): 0.0, (7, 'O', 'O'): 4.0408020236605833e-22, (7, 'O', 'I-GENE'): 1.974178961563237e-21, (7, 'I-GENE', 'O'): 0.0, (7, 'I-GENE', 'I-GENE'): 0.0, (8, 'O', 'O'): 3.89731307990885e-26, (8, 'O', 'I-GENE'): 0.0, (8, 'I-GENE', 'O'): 8.306859104625818e-26, (8, 'I-GENE', 'I-GENE'): 0.0, (9, 'O', 'O'): 8.353154290707845e-31, (9, 'O', 'I-GENE'): 0.0, (9, 'I-GENE', 'O'): 0.0, (9, 'I-GENE', 'I-GENE'): 0.0, (10, 'O', 'O'): 1.1567839565458657e-32, (10, 'O', 'I-GENE'): 1.2088004378859481e-34, (10, 'I-GENE', 'O'): 0.0, (10, 'I-GENE', 'I-GENE'): 0.0, (11, 'O', 'O'): 4.802806736037231e-34, (11, 'O', 'I-GENE'): 2.3017548147116585e-36, (11, 'I-GENE', 'O'): 2.1895257940617565e-36, (11, 'I-GENE', 'I-GENE'): 1.2164074924471453e-38, (12, 'O', 'O'): 0.0, (12, 'O', 'I-GENE'): 0.0, (12, 'I-GENE', 'O'): 0.0, (12, 'I-GENE', 'I-GENE'): 0.0, (13, 'O', 'O'): 0.0, (13, 'O',

'I-GENE'): 0.0, (13, 'I-GENE', 'O'): 0.0, (13, 'I-GENE', 'I-GENE'): 0.0, (14, 'O', 'O'): 0.0, (14, 'O', 'I-GENE'): 0.0, (14, 'I-GENE', 'O'): 0.0, (14, 'I-GENE', 'I-GENE'): 0.0}
Pi table for rare words:
{(0, '*', '*'): 1, (1, '*', 'O'): 0.00010720660160867851, (1, '*', 'I-GENE'): 0.0, (2, 'O', 'O'): 1.4848596405018943e-07, (2, 'O', 'I-GENE'): 7.696410643338835e-09, (2, 'I-GENE', 'O'): 0.0, (2, 'I-GENE', 'I-GENE'): 0.0, (3, 'O', 'O'): 1.897575232826306e-10, (3, 'O', 'I-GENE'): 0.0, (3, 'I-GENE', 'O'): 4.29095968973283e-12, (3, 'I-GENE', 'I-GENE'): 0.0, (4, 'O', 'O'): 4.016255338835847e-14, (4, 'O', 'I-GENE'): 0.0, (4, 'I-GENE', 'O'): 0.0, (4, 'I-GENE', 'I-GENE'): 0.0, (5, 'O', 'O'): 3.1053665020140405e-16, (5, 'O', 'I-GENE'): 0.0, (5, 'I-GENE', 'O'): 0.0, (5, 'I-GENE', 'I-GENE'): 0.0, (6, 'O', 'O'): 4.683995617178435e-19, (6, 'O', 'I-GENE'): 0.0, (6, 'I-GENE', 'O'): 0.0, (6, 'I-GENE', 'I-GENE'): 0.0, (7, 'O', 'O'): 4.0408020236605833e-22, (7, 'O', 'I-GENE'): 1.974178961563237e-21, (7, 'I-GENE', 'O'): 0.0, (7, 'I-GENE', 'I-GENE'): 0.0, (8, 'O', 'O'): 3.89731307990885e-26, (8, 'O', 'I-GENE'): 0.0, (8, 'I-GENE', 'O'): 8.306859104625818e-26, (8, 'I-GENE', 'I-GENE'): 0.0, (9, 'O', 'O'): 8.353154290707845e-31, (9, 'O', 'I-GENE'): 0.0, (9, 'I-GENE', 'O'): 0.0, (9, 'I-GENE', 'I-GENE'): 0.0, (10, 'O', 'O'): 1.1567839565458657e-32, (10, 'O', 'I-GENE'): 1.2088004378859481e-34, (10, 'I-GENE', 'O'): 0.0, (10, 'I-GENE', 'I-GENE'): 0.0, (11, 'O', 'O'): 4.802806736037231e-34, (11, 'O', 'I-GENE'): 2.3017548147116585e-36, (11, 'I-GENE', 'O'): 2.1895257940617565e-36, (11, 'I-GENE', 'I-GENE'): 1.2164074924471453e-38, (12, 'O', 'O'): 3.7033623909630266e-35, (12, 'O', 'I-GENE'): 7.586176738870117e-35, (12, 'I-GENE', 'O'): 7.743062483998038e-38, (12, 'I-GENE', 'I-GENE'): 1.8386733441175026e-37, (13, 'O', 'O'): 1.0715567801711531e-38, (13, 'O', 'I-GENE'): 8.038806522752096e-38, (13, 'I-GENE', 'O'): 9.576229289164967e-39, (13, 'I-GENE', 'I-GENE'): 8.327909465199032e-38, (14, 'O', 'O'): 4.4050313106831725e-40, (14, 'O', 'I-GENE'): 3.1982562051093937e-41, (14, 'I-GENE', 'O'): 2.1430100741997278e-39, (14, 'I-GENE', 'I-GENE'): 2.026665826899016e-40}