**Natural Language Processing Assignment 3**

**Name: Shiang Hu**

**PID: A53267858**

*1. Baseline:*

(1) briefly describe the baseline tagger (what is it doing?) and rare word replacement used:

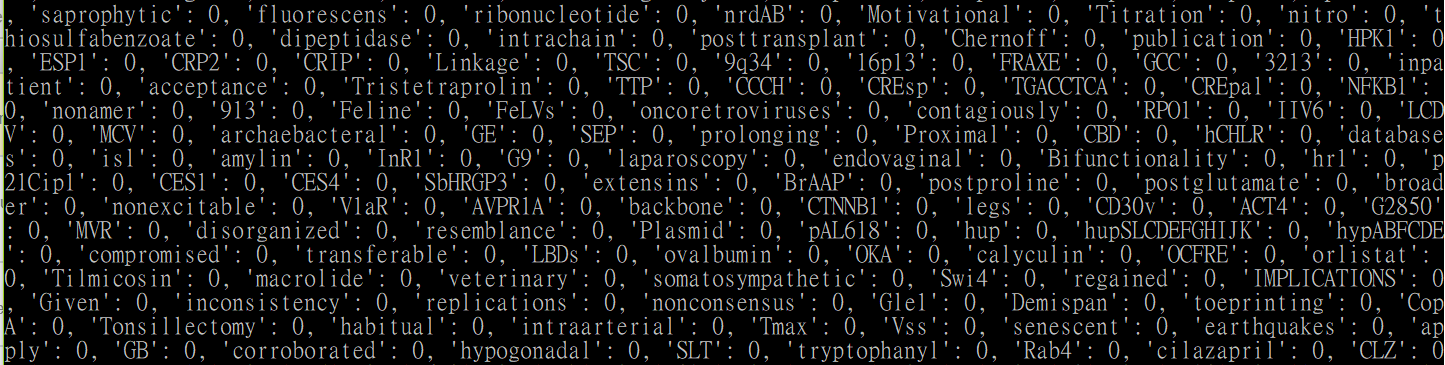
for the baseline tagger if calculates the emission parameter for each word, meaning it calculates the number of times a word is labeled as a tag in the training data then divide it by the number of times that tag appears in the training data.

For instance, the word “Reverse” was tagged “O” 3 times in the training data, and the tag “O” appeared

345128 times in the training data, so the emission parameter for the word “Reverse” with “O” tag is 3/345128

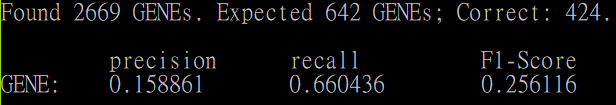
for rare words replacement, we replace words that appears less than 5 times in the training data as “\_\_RARE\_\_”, then when we read gene.dev we replace words that we haven’t seen (words count less than 5 in training, and words that didn’t appear in training) as “\_\_RARE\_\_”, the emission of these words is then consider as the same as \_\_RARE\_\_

ex: some \_\_RARE\_\_ words

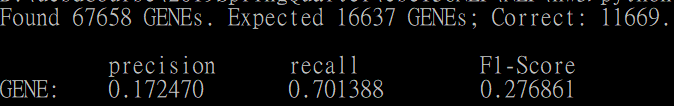


(2) evaluate the baseline on dev set:

evaluation on dev set:

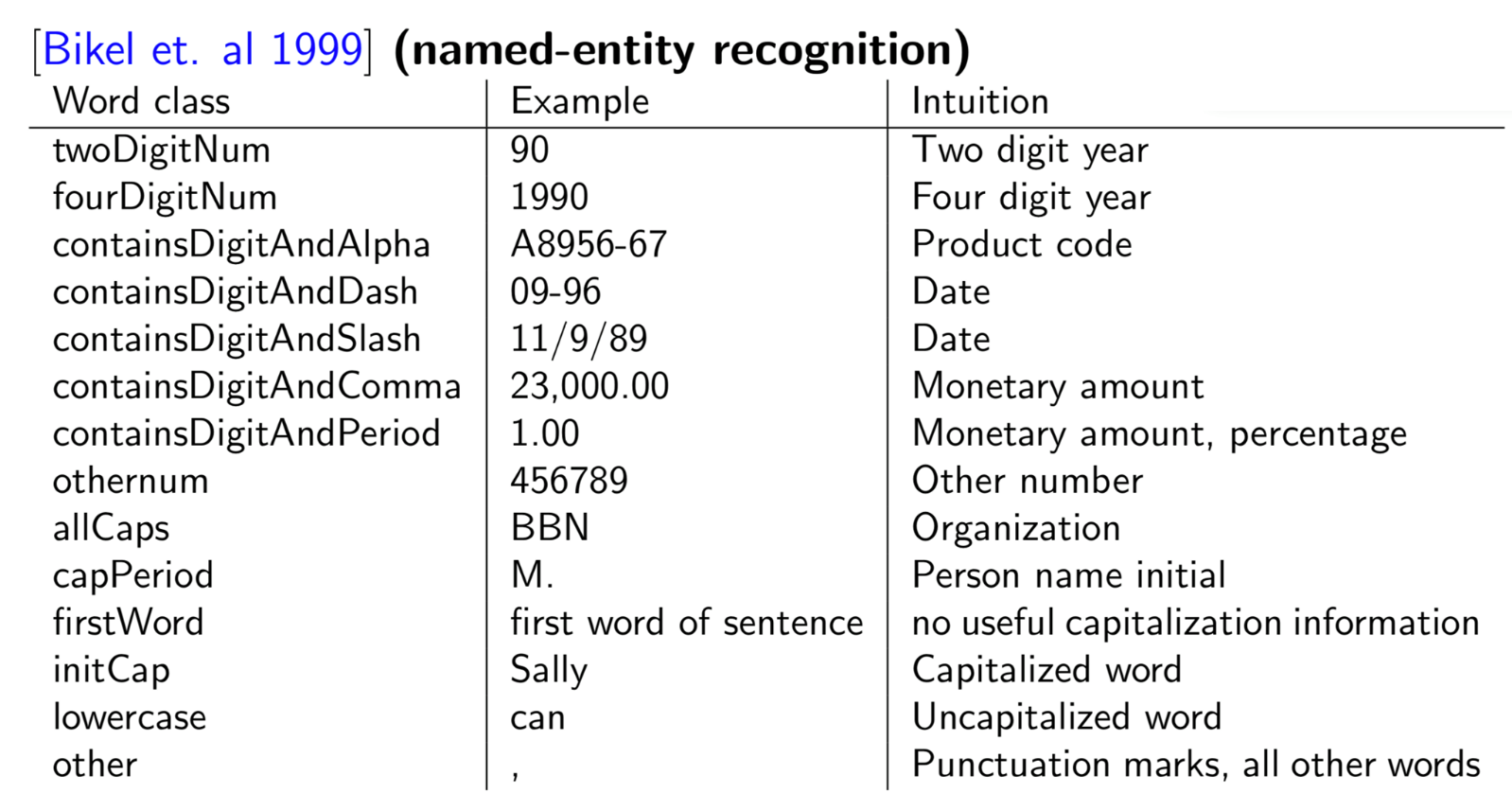


evaluation on train set:



(3) given a word, how would you categorize it into classes for each design method?

if the word provided appears above five times it uses the same emission defined, if the word appears less than 5 times, we consider different cases of rare words



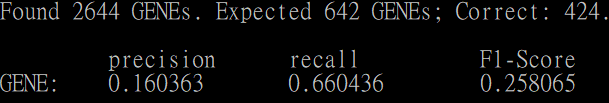
by taking a peak at the \_\_RARE\_\_ words, we consider more informative classes: \_\_DigitAndAlpha\_\_, \_\_othernum\_\_, \_\_allCaps\_\_, \_\_initCap\_\_, and \_\_RARE\_\_, since these classes dominants the rare words, if the word has digits and characters in it, it is considered as \_\_DigitAndAlpha\_\_ class, else if the word only has digits in it, it is considered as \_\_othernum\_\_ class, for words that only has digits, we check if all the letters are upper case then it is considered as \_\_allCaps\_\_ class, else if the first character is upper case, it is considered as \_\_initCap\_\_ class, for words that doesn’t belong to any of these classes, we

consider it as \_\_RARE\_\_

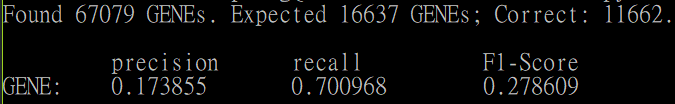
|  |  |  |
| --- | --- | --- |
|  | I-GENE | O |
| \_\_RARE\_\_ | 2276 | 17860 |
| \_\_DigitAndAlpha\_\_ | 3526 | 1631 |
| \_\_othernum\_\_ | 39 | 1141 |
| \_\_allcaps\_\_ | 1411 | 2698 |
| \_\_initCap\_\_ | 1480 | 5451 |

a. we use all five classes:

evaluation on dev set:

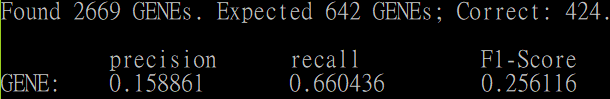


evaluation on train set:

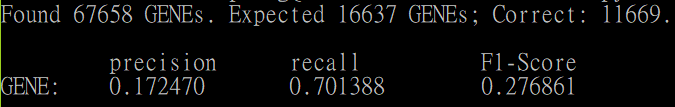


b. we exclude the \_\_othernum\_\_ class:

evaluation on dev set:

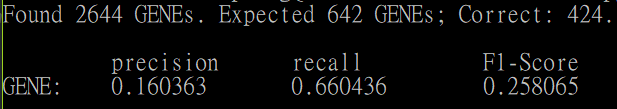


evaluation on train set:

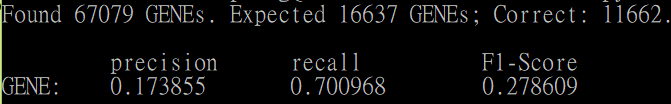


c. we exclude the \_\_initCap\_\_ and \_\_allCaps\_\_ classes:

evaluation on dev set:

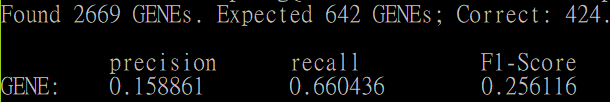


evaluation on train set:

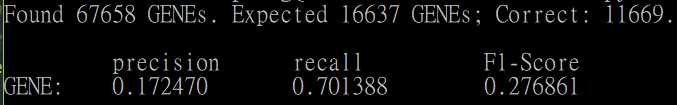


d. we exclude the \_\_othernum\_\_, \_\_initCap\_\_, \_\_allCaps\_\_ classes

evaluation on dev set:



evaluation on train set:



(4) provide insightful comparison based on these results:

we can see that by adding the five classes, the performance of the dev set and train set increases, for the dev set it only increases precision and f1 score, recall stays the same, meaning false positives decreases, it seems from the experiments, only a few class truly matters, the \_\_othernum\_\_ class and the DigitAndAlpha class with are classes with digits, since our data is gene data, some of these digit contained rare words may matter significantly, like “PDE4A”, “H2A2”, these words are probably chemical, or bio related words and may affect the tag prediction.

*2. Trigram HMM:*

(1) Describe the Viterbi algorithm:

*motivation behind using HMM for tagging:*

the motivation of using HMM is to find the hidden states that has the highest joint probability and assign tags for the given sentence

*purpose of the Viterbi algorithm – dynamic programming vs. brute force*

since brute force is very inefficient, we can use dynamic programming to solve an originally O(n^4) problem to O(n^3)

*specifics of implementation:*

k equals index of sentence, u is the tag of previous word, v is the tag of current word and w is the tag of previous second word

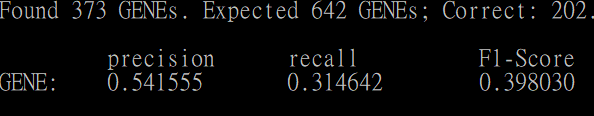
base case: pi(k=0,\*,\*) = 1 for u not \* or v not \*, pi(k=0, u,v) = 0

recursive happens since we use the pi(k-1,u,v) for our next iteration, so we have to store every pi(k,u,v) with max joint probability, and bp(k,u,v) stores the selected w tag of the max joint probability calculated, the joint probability is define as pi(k-1,u,v) \* q \* e where q is the trigram probability q(v|w,u) and e is the emission parameter, after iterating through the sentence k=1 to k=n, we use STOP to predict the last two word tags, by choosing the best u,v that max the joint probability pi(n,u,v)\*q(STOP|u,v) after predicting the last two tags, we reversely select the best tag by selecting tag equals bp(k+2, yk+1, yk+2) where yk+2 is the last previous second predicted tag yk+1 is the previous predicted tag

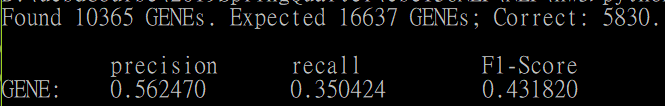
* Describe the Viterbi algorithm, using equations when necessary: Asking yourself these questions would help!
  + What is the motivation behind using HMMs for tagging -- generative modeling with Markov assumptions and exploitation of conditional independences in the sequence model?
  + What is the purpose of the Viterbi algorithm -- dynamic programming vs. brute force?
  + What are the specifics of your implementation: what is the base case; how did you implement the recursive formulation; how did you obtain the joint probability of word sequence and tag sequence; how do you go from this joint probability to final tag sequence, i.e., what path in your DP table gives you the final tag sequence?
  + If you got stuck and your implementation is not working, describe the key challenges and the issues you faced

(2) evaluate the HMM tagger on the dev set, verify the F-1 score:

evaluation on dev set:



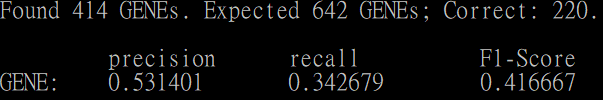
evaluation on train set:



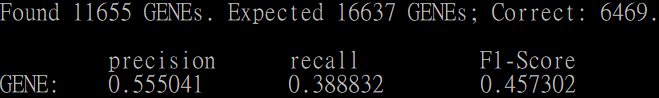
(3) evaluate the new HMM model with informative word classes on train and dev sets:

a. we use all five classes:

evaluation on dev set:

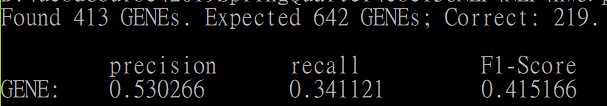


evaluation on train set:

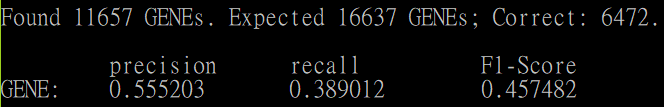


b. we exclude the \_\_othernum\_\_ class:

evaluation on dev set:

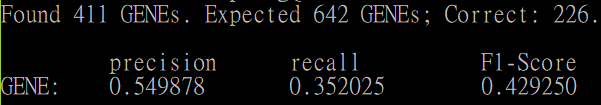


evaluation on train set:

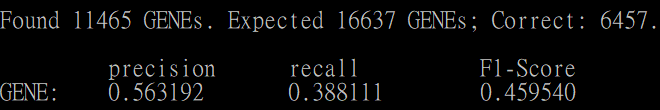


c. we exclude the \_\_initCap\_\_ and \_\_allCaps\_\_ classes:

evaluation on dev set:

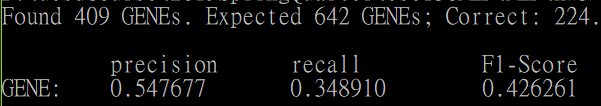


evaluation on train set:

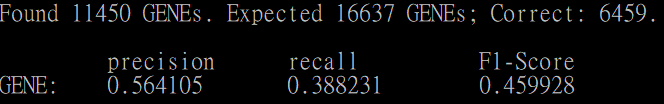


d. we exclude the \_\_othernum\_\_, \_\_initCap\_\_, \_\_allCaps\_\_ classes

evaluation on dev set:



evaluation on train set:



(4) provide insightful comparison based on these results:

we can see that by adding informative classes, every experiment increases the F1-Score as opposed to using emission to predict tags, the HMM model captures more previous context, so add informative class increases its results, but it seems that not adding the more informative classes the better, we can see that the best result is when we only add \_\_DigitAndAlpha\_\_, and \_\_RARE\_\_ class, the second best result is when add \_\_DigitAndAlpha\_\_, \_\_othernum\_\_ and \_\_RARE\_\_, it’s possible that the model over fits when it’s too informative, since our model is HMM trigram model when we consider digit and alpha words it has the best performance, then when we consider digit words in context it decreases accuracy, it means that words with digits often appear together in context, but perhaps the digit words have different tag meaning so the model over fits