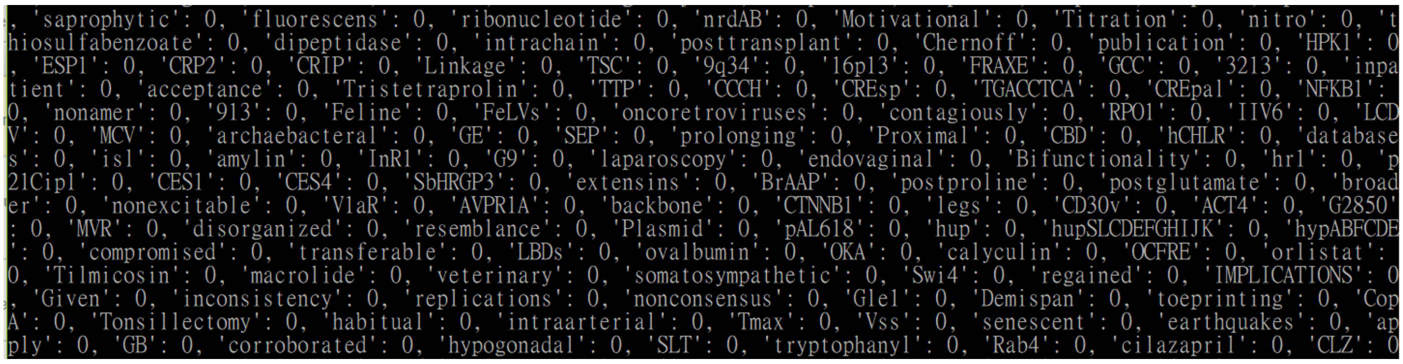
Natural Language Processing Assignment 3
Name: Shiang Hu
PID: A53267858

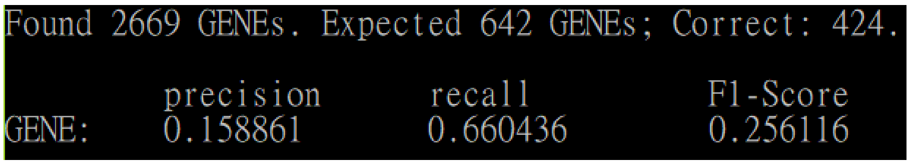
1. Baseline:

(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



```
, 'saprophytic': 0, 'fluorescens': 0, 'ribonucleotide': 0, 'nrdAB': 0, 'Motivational': 0, 'Titration': 0, 'nitro': 0, 't
hiosulfabenzoate': 0, 'dipeptidase': 0, 'intrachain': 0, 'posttransplant': 0, 'Chernoff': 0, 'publication': 0, 'HPK1': 0
, 'ESP1': 0, 'CRP2': 0, 'CRIP': 0, 'Linkage': 0, 'TSC': 0, '9q34': 0, '16p13': 0, 'FRAXE': 0, 'GCC': 0, '3213': 0, 'inpa
tient': 0, 'acceptance': 0, 'Tristetraprolin': 0, 'TTP': 0, 'CCCH': 0, 'CREsp': 0, 'TGACCTCA': 0, 'CREpal': 0, 'NFKB1':
0, 'nonamer': 0, '913': 0, 'Feline': 0, 'FeLVs': 0, 'oncoretroviruses': 0, 'contagiously': 0, 'RPO1': 0, 'IIV6': 0, 'LCD
V': 0, 'MCV': 0, 'archaebacteral': 0, 'GE': 0, 'SEP': 0, 'prolonging': 0, 'Proximal': 0, 'CBD': 0, 'hCHLR': 0, 'database
s': 0, 'isl': 0, 'amylin': 0, 'InR1': 0, 'G9': 0, 'laparoscopy': 0, 'endovaginal': 0, 'Bifunctionality': 0, 'hr1': 0, 'p
21Cip1': 0, 'CES1': 0, 'CES4': 0, 'SbHRGP3': 0, 'extensins': 0, 'BrAAP': 0, 'postproline': 0, 'postglutamate': 0, 'broad
er': 0, 'nonexcitable': 0, 'V1aR': 0, 'AVPR1A': 0, 'backbone': 0, 'CTNNB1': 0, 'legs': 0, 'CD30v': 0, 'ACT4': 0, 'G2850'
: 0, 'MVR': 0, 'disorganized': 0, 'resemblance': 0, 'Plasmid': 0, 'pAL618': 0, 'hup': 0, 'hupSLCDEFGHIJK': 0, 'hypABFCDE
': 0, 'compromised': 0, 'transferable': 0, 'LBDs': 0, 'ovalbumin': 0, 'OKA': 0, 'calyculin': 0, 'OCFRE': 0, 'orlistat':
0, 'Tilmicosin': 0, 'macrolide': 0, 'veterinary': 0, 'somatosympathetic': 0, 'Swi4': 0, 'regained': 0, 'IMPLICATIONS': 0
, 'Given': 0, 'inconsistency': 0, 'replications': 0, 'nonconsensus': 0, 'Gle1': 0, 'Demispan': 0, 'toeprinting': 0, 'Cop
A': 0, 'Tonsillectomy': 0, 'habitual': 0, 'intraarterial': 0, 'Tmax': 0, 'Vss': 0, 'senescent': 0, 'earthquakes': 0, 'ap
ply': 0, 'GB': 0, 'corroborated': 0, 'hypogonadal': 0, 'SLT': 0, 'tryptophanyl': 0, 'Rab4': 0, 'cilazapril': 0, 'CLZ': 0
```

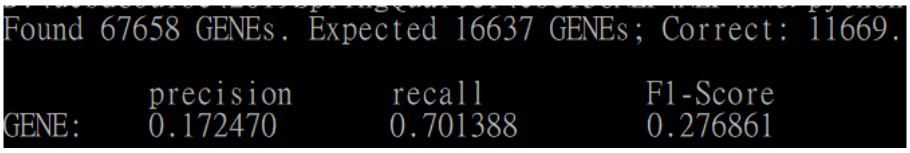
(2) evaluate the baseline on dev set:
evaluation on dev set:



```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

         precision       recall          F1-Score
GENE:    0.158861        0.660436        0.256116
```

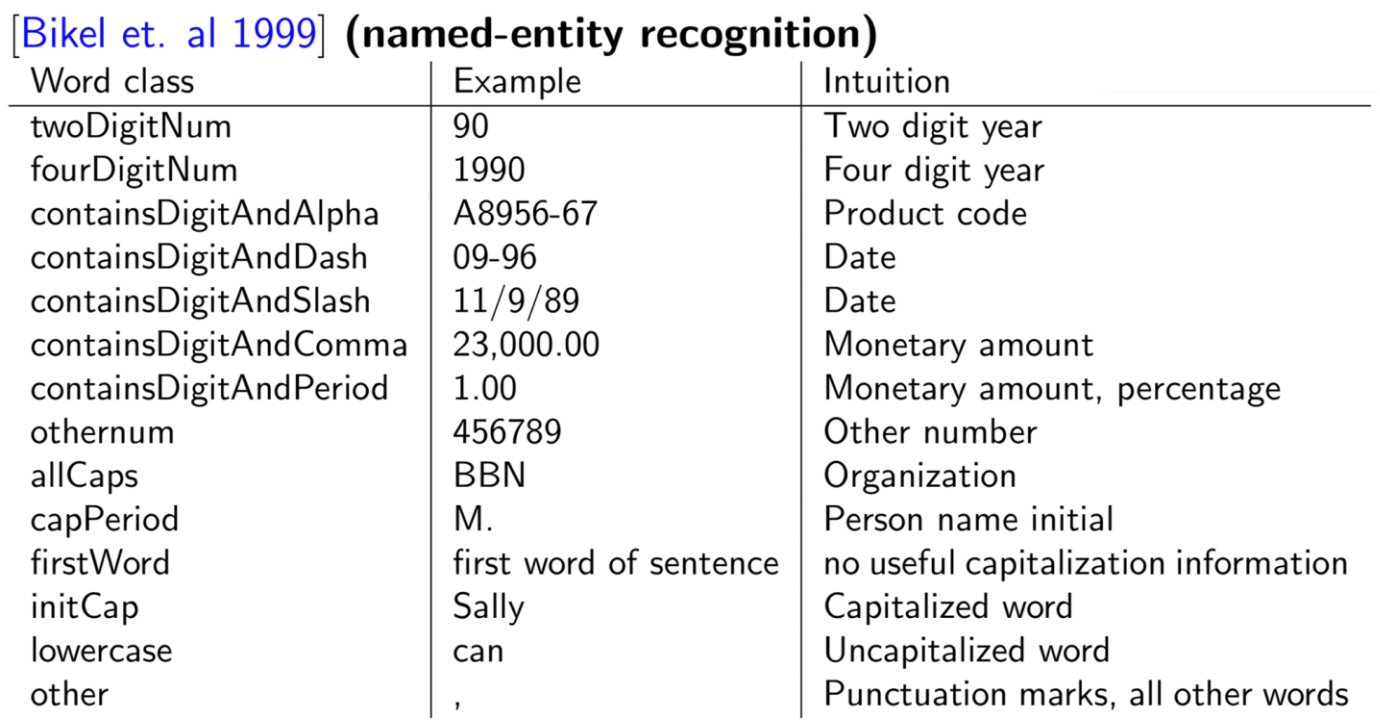
evaluation on train set:



```
Found 67658 GENEs. Expected 16637 GENEs; Correct: 11669.

         precision       recall          F1-Score
GENE:    0.172470        0.701388        0.276861
```

(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



[Bikel et. al 1999] **(named-entity recognition)**

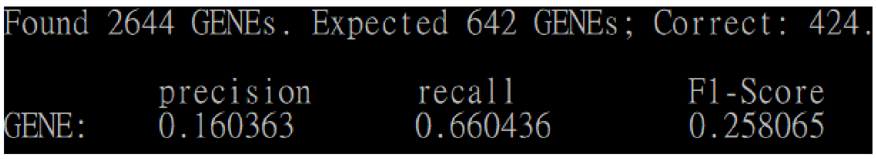
Word class	Example	Intuition
twoDigitNum	90	Two digit year
fourDigitNum	1990	Four digit year
containsDigitAndAlpha	A8956-67	Product code
containsDigitAndDash	09-96	Date
containsDigitAndSlash	11/9/89	Date
containsDigitAndComma	23,000.00	Monetary amount
containsDigitAndPeriod	1.00	Monetary amount, percentage
othernum	456789	Other number
allCaps	BBN	Organization
capPeriod	M.	Person name initial
firstWord	first word of sentence	no useful capitalization information
initCap	Sally	Capitalized word
lowercase	can	Uncapitalized word
other	,	Punctuation marks, all other 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:

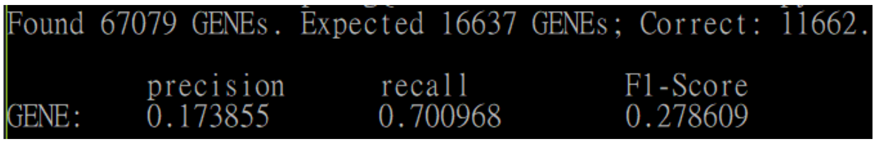
evaluation on dev set:



```
Found 2644 GENEs. Expected 642 GENEs; Correct: 424.

         precision       recall          F1-Score
GENE:    0.160363        0.660436        0.258065
```

evaluation on train set:

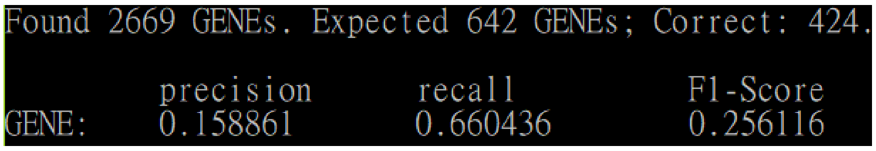


```
Found 67079 GENEs. Expected 16637 GENEs; Correct: 11662.

         precision       recall          F1-Score
GENE:    0.173855        0.700968        0.278609
```

b. we exclude the __othernum__ class:

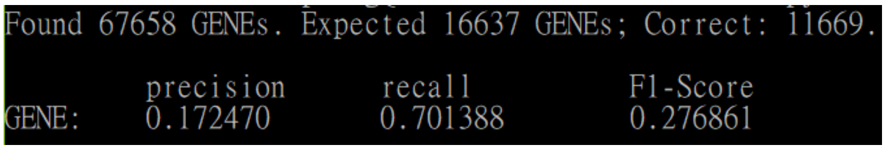
evaluation on dev set:



```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

         precision       recall          F1-Score
GENE:    0.158861        0.660436        0.256116
```

evaluation on train set:

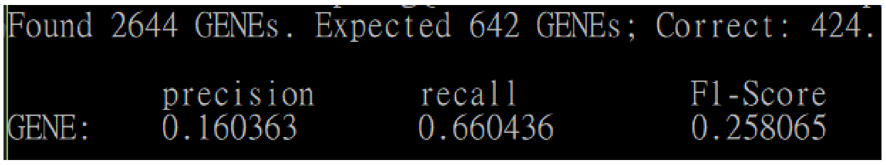


```
Found 67658 GENEs. Expected 16637 GENEs; Correct: 11669.

         precision       recall          F1-Score
GENE:    0.172470        0.701388        0.276861
```

c. we exclude the __initCap__ and __allCaps__ classes:

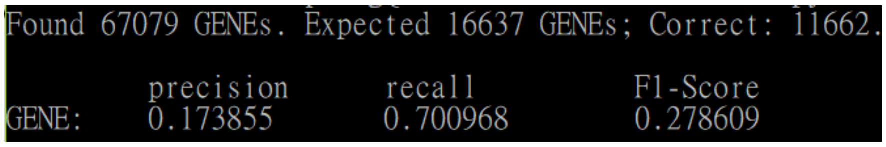
evaluation on dev set:



```
Found 2644 GENEs. Expected 642 GENEs; Correct: 424.

         precision       recall          F1-Score
GENE:    0.160363        0.660436        0.258065
```

evaluation on train set:

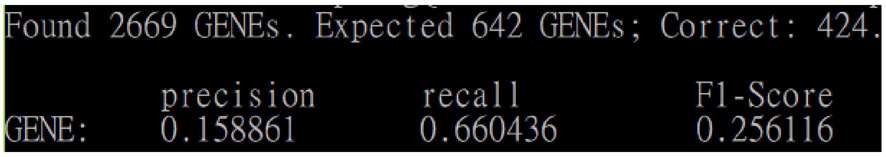


```
Found 67079 GENEs. Expected 16637 GENEs; Correct: 11662.

         precision       recall          F1-Score
GENE:    0.173855        0.700968        0.278609
```

d. we exclude the __othernum__, __initCap__, __allCaps__ classes

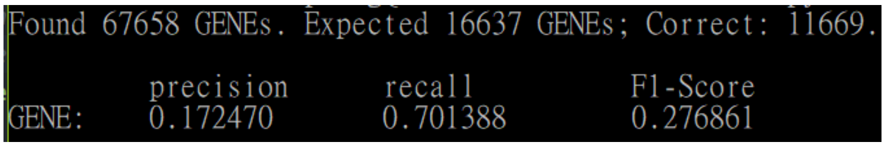
evaluation on dev set:



```
Found 2669 GENEs. Expected 642 GENEs; Correct: 424.

         precision       recall          F1-Score
GENE:    0.158861        0.660436        0.256116
```

evaluation on train set:



```
Found 67658 GENEs. Expected 16637 GENEs; Correct: 11669.

         precision       recall          F1-Score
GENE:    0.172470        0.701388        0.276861
```

(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

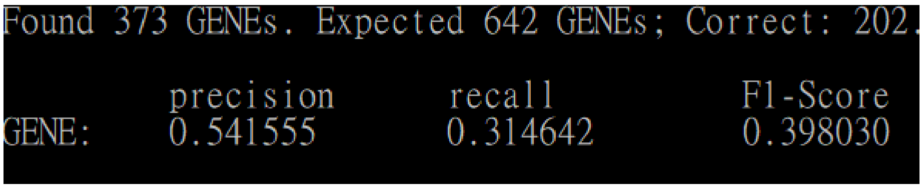
purpose of the Viterbi algorithm – dynamic programming vs. brute force
since brute force is very inefficient, we can use dynamic programming to solve an originally O(n^4) problem to O(n^3)

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:

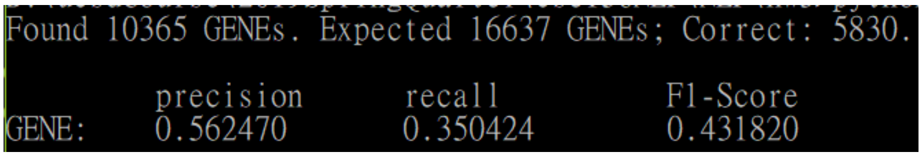
evaluation on dev set:



```
Found 373 GENEs. Expected 642 GENEs; Correct: 202

        precision       recall          F1-Score
GENE:   0.541555        0.314642        0.398030
```

evaluation on train set:



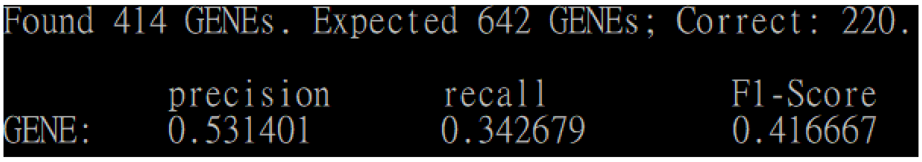
```
Found 10365 GENEs. Expected 16637 GENEs; Correct: 5830.

        precision       recall          F1-Score
GENE:   0.562470        0.350424        0.431820
```

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

a. we use all five classes:

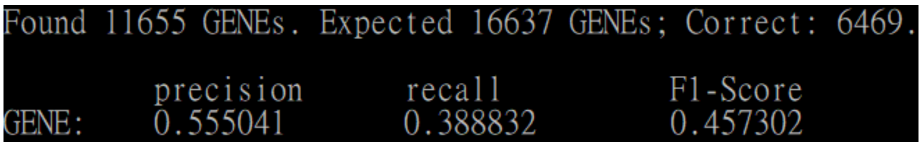
evaluation on dev set:



```
Found 414 GENEs. Expected 642 GENEs; Correct: 220.

        precision       recall          F1-Score
GENE:   0.531401        0.342679        0.416667
```

evaluation on train set:

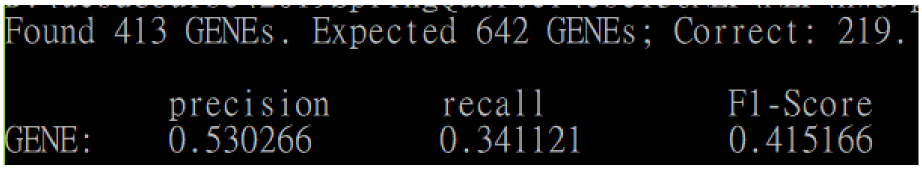


```
Found 11655 GENEs. Expected 16637 GENEs; Correct: 6469.

        precision       recall          F1-Score
GENE:   0.555041        0.388832        0.457302
```

b. we exclude the __othernum__ class:

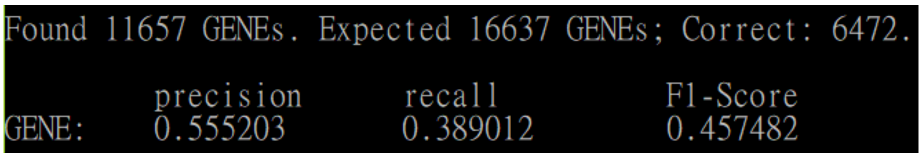
evaluation on dev set:



```
Found 413 GENEs. Expected 642 GENEs; Correct: 219.

        precision       recall          F1-Score
GENE:   0.530266        0.341121        0.415166
```

evaluation on train set:

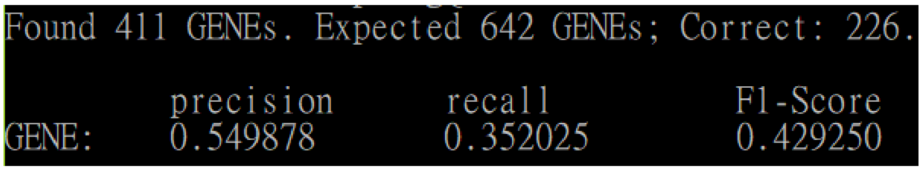


```
Found 11657 GENEs. Expected 16637 GENEs; Correct: 6472.

        precision       recall          F1-Score
GENE:   0.555203        0.389012        0.457482
```

c. we exclude the __initCap__ and __allCaps__ classes:

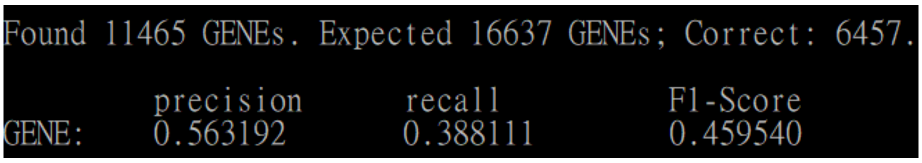
evaluation on dev set:



```
Found 411 GENEs. Expected 642 GENEs; Correct: 226.

        precision       recall          F1-Score
GENE:   0.549878        0.352025        0.429250
```

evaluation on train set:

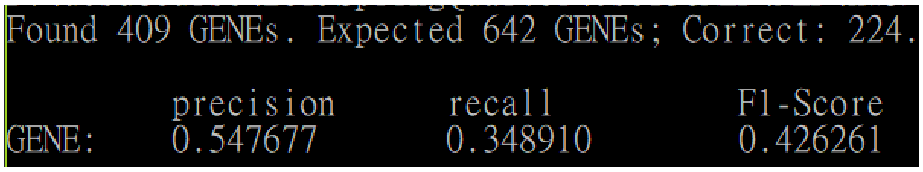


```
Found 11465 GENEs. Expected 16637 GENEs; Correct: 6457.

         precision       recall          F1-Score
GENE:    0.563192        0.388111        0.459540
```

d. we exclude the __othernum__, __initCap__, __allCaps__ classes

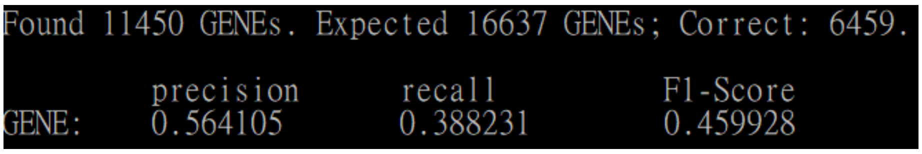
evaluation on dev set:



```
Found 409 GENEs. Expected 642 GENEs; Correct: 224.

         precision       recall          F1-Score
GENE:    0.547677        0.348910        0.426261
```

evaluation on train set:



```
Found 11450 GENEs. Expected 16637 GENEs; Correct: 6459.

         precision       recall          F1-Score
GENE:    0.564105        0.388231        0.459928
```

(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