

UNI: sw3196

Name: Shenxiu Wu

Question 4:

Q4_2

```
[dyn-160-39-144-44:hw1 shenxiu-wu$ python eval_ne_tagger.py ner_dev.key 4_2.txt ]  
Found 14043 NEs. Expected 5931 NEs; Correct: 3117.
```

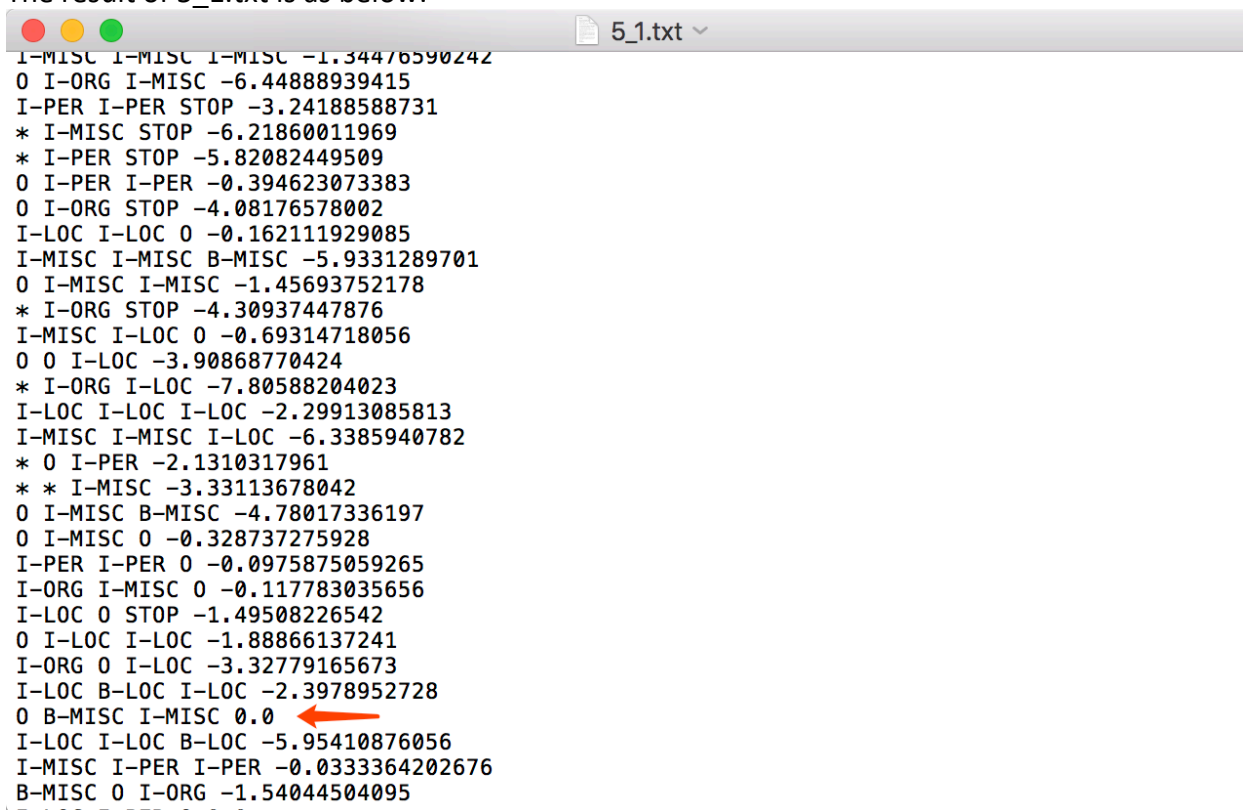
	precision	recall	F1-Score
Total:	0.221961	0.525544	0.312106
PER:	0.435451	0.231230	0.302061
ORG:	0.475936	0.399103	0.434146
LOC:	0.147750	0.870229	0.252612
MISC:	0.491689	0.610206	0.544574

```
dyn-160-39-144-44:hw1 shenxiu-wu$
```

From above screenshot we could see the precision is relatively low. This is because it takes no context into account. But I think the result meets my expectation.

Question 5:

The result of 5_1.txt is as below:



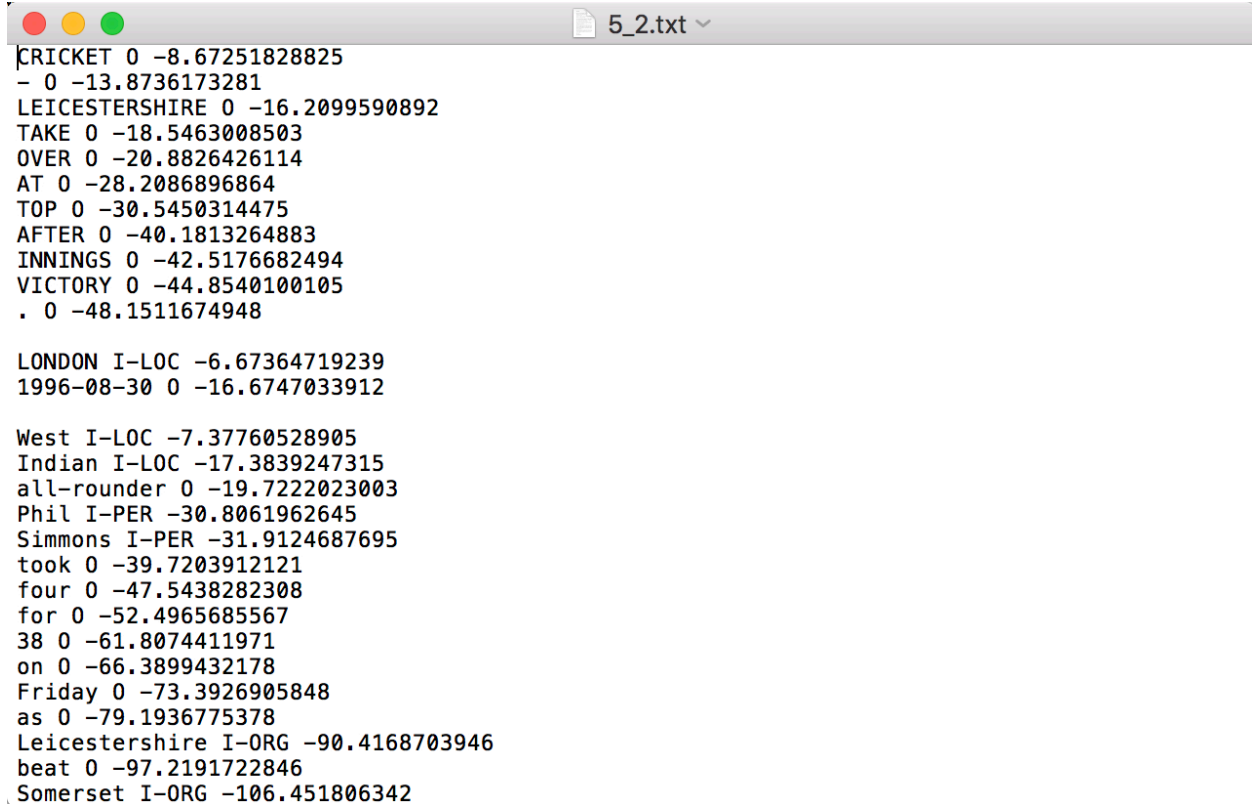
```
I-MISC I-MISC I-MISC -1.34476590242  
0 I-ORG I-MISC -6.44888939415  
I-PER I-PER STOP -3.24188588731  
* I-MISC STOP -6.21860011969  
* I-PER STOP -5.82082449509  
0 I-PER I-PER -0.394623073383  
0 I-ORG STOP -4.08176578002  
I-LOC I-LOC 0 -0.162111929085  
I-MISC I-MISC B-MISC -5.9331289701  
0 I-MISC I-MISC -1.45693752178  
* I-ORG STOP -4.30937447876  
I-MISC I-LOC 0 -0.69314718056  
0 0 I-LOC -3.90868770424  
* I-ORG I-LOC -7.80588204023  
I-LOC I-LOC I-LOC -2.29913085813  
I-MISC I-MISC I-LOC -6.3385940782  
* 0 I-PER -2.1310317961  
* * I-MISC -3.33113678042  
0 I-MISC B-MISC -4.78017336197  
0 I-MISC 0 -0.328737275928  
I-PER I-PER 0 -0.0975875059265  
I-ORG I-MISC 0 -0.117783035656  
I-LOC 0 STOP -1.49508226542  
0 I-LOC I-LOC -1.88866137241  
I-ORG 0 I-LOC -3.32779165673  
I-LOC B-LOC I-LOC -2.3978952728  
0 B-MISC I-MISC 0.0  
I-LOC I-LOC B-LOC -5.95410876056  
I-MISC I-PER I-PER -0.0333364202676  
B-MISC 0 I-ORG -1.54044504095
```

The result meets my expectation. Because the forth column represents the log-probability, as shown in above picture, 0.0 denotes the highest probability, which is 100%. After I verify this from original training data, this is correct.

For 5_2.txt, my 5_2.py can successfully run and produce 5_2.txt. But when I run "python eval_ne_tagger.py ner_dev.key 5_2.txt" script, I meet the following problem. And I have tried every way to solve it, but unfortunately, I cannot figure out what the problem is and how to fix it before I hand in my homework. Thus, I don't have the chance to evaluate the performance of my tagger model which implement the Viterbi algorithm. But I believe it definitely has a better performance and precision because it at least takes the context (the trigram) into account.

```
Traceback (most recent call last):
  File "eval_ne_tagger.py", line 285, in <module>
    evaluator.compare(gs_iterator, pred_iterator)
  File "eval_ne_tagger.py", line 130, in compare
    pred_word, pred_tag = prediction.next() # Get the corresponding item from the prediction stream
StopIteration
```

From my perspective, I think my program is correct. And at least, the format of 5_2.txt is quite the same as 4_2.txt. Below is 5_2.txt 's screenshot. Thus, I really don't know why the eval_ne_tagger.py cannot run with my 5_2.txt.



```
CRICKET 0 -8.67251828825
- 0 -13.8736173281
LEICESTERSHIRE 0 -16.2099590892
TAKE 0 -18.5463008503
OVER 0 -20.8826426114
AT 0 -28.2086896864
TOP 0 -30.5450314475
AFTER 0 -40.1813264883
INNINGS 0 -42.5176682494
VICTORY 0 -44.8540100105
. 0 -48.1511674948

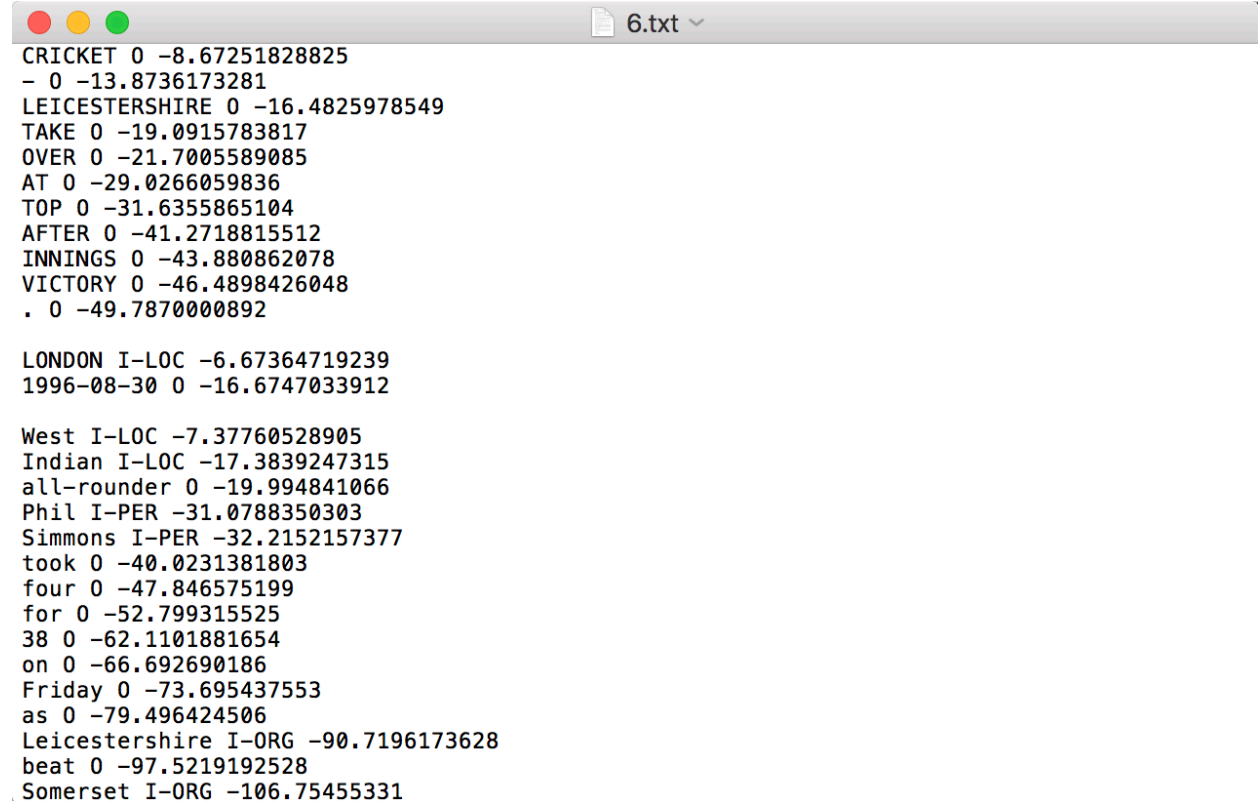
LONDON I-LOC -6.67364719239
1996-08-30 0 -16.6747033912

West I-LOC -7.37760528905
Indian I-LOC -17.3839247315
all-rounder 0 -19.7222023003
Phil I-PER -30.8061962645
Simmons I-PER -31.9124687695
took 0 -39.7203912121
four 0 -47.5438282308
for 0 -52.4965685567
38 0 -61.8074411971
on 0 -66.3899432178
Friday 0 -73.3926905848
as 0 -79.1936775378
Leicestershire I-ORG -90.4168703946
beat 0 -97.2191722846
Somerset I-ORG -106.451806342
```

From the observation by my eyes, I think 5_2.txt definitely has been improved. For example, in 4_2.txt, "TAKE" & "OVER" are tagged with "B-LOC". But in 5_2.txt, they are tagged with "O". I think the result in 5_2.txt is much reasonable.

Question 6:

When I run 6.txt, I met the same problem as above in question 5 because my Viterbi algorithm is the same one. The screenshot below is my 6.txt content. I also think it's quite normal and should have been implemented successfully with eval_ne_tagger.py.



```
CRICKET 0 -8.67251828825
- 0 -13.8736173281
LEICESTERSHIRE 0 -16.4825978549
TAKE 0 -19.0915783817
OVER 0 -21.7005589085
AT 0 -29.0266059836
TOP 0 -31.6355865104
AFTER 0 -41.2718815512
INNINGS 0 -43.880862078
VICTORY 0 -46.4898426048
. 0 -49.7870000892

LONDON I-L0C -6.67364719239
1996-08-30 0 -16.6747033912

West I-L0C -7.37760528905
Indian I-L0C -17.3839247315
all-rounder 0 -19.994841066
Phil I-PER -31.0788350303
Simmons I-PER -32.2152157377
took 0 -40.0231381803
four 0 -47.846575199
for 0 -52.799315525
38 0 -62.1101881654
on 0 -66.692690186
Friday 0 -73.695437553
as 0 -79.496424506
Leicestershire I-ORG -90.7196173628
beat 0 -97.5219192528
Somerset I-ORG -106.75455331
```

I again replace words in the original training data with more classes not only “_RARE_”. I have added “_Numeric_”, “_All-Capitals_”, “_Acronyms_”. This creation of different word classes helps to improve the Viterbi algorithm.

```

raw = readfile(f_raw)
regex = r"\b[A-Z][a-zA-Z\.\-]*[A-Z]\b\."
pat = r"\b[0-9\.\-\"'\,\?]*\b"
with open(f_write, 'w+') as f:
    for row in raw:
        segments = row.split(" ")
        if segments[0] in record:
            if segments[0] in re.findall(pat, segments[0]):
                segments[0] = "_Numeric_"
            elif segments[0].isupper():
                segments[0] = "_All-Capitals_"
            elif segments[0] in re.findall(regex, segments[0]):
                segments[0] = "_Acronyms_"
            else:
                segments[0] = "_RARE_"
        f.write(" ".join(segments))

def run():

```

Numeric means the word is rare and contains at least one numeric characters.

Acronyms means the word is rare and the possibly abbreviations for first names or companies.

All-capitals means the word is rare and consists entirely of capitalized letters.