COMS 4705
HW 1
Kaili Zhang
kz2203

Cover Letter

Codes are in Python.

Modules used: sys re(regular expression) match

The main idea of these codes are based on matching function of regular expression. I used regular expression to find the pattern for each data file, and then extract information and do calculation.

!!!!!! There's a serious problem should remind all the class!!!!!!

If you are using window 'powershell', all '.counts' file could be open correctly in 'notepad++' and in correct format, but the file format is wrong actually. It becomes more complicated with lots operational characters.

Instead of using 'powershell', 'cmd' works well. Windows is a joke!

1. For baseline tagger, it owns a quite weak performance without considering value of parameter q, however, it works fast!

2. For the Viterbi tagger, I just followed the pseudocode in lectures. Obviously, comparing to the baseline, viterbi tagger performs much better. The precision, recall and F1-Score show good result. And what's more, for different tags, it also shows a stable performance.

   However, viterbi costs much on time and space.

3. For problem 6, I designed 5 return value:

'_CAPALL_' for words with more than one characters and all capital characters;

'_CAP_' for single captital character;

'_CAPFST_' for words with capital headers;

'_NUM_' for numbers;

'_ORG_' for words containing only capitals and dots.

We can see that, after using 6 categories to represent these low forewent words instead of 1, we gain a much better result again!

Correct tag number increase by near 700, Even the three main options are not as well as previous one.

I think that is because the so call 'long-tail 'phenomenon. Since in testing set there may appear lots of unseen words and lots none words appear rarely.

COMS 4705
HW 1
Kaili Zhang
kz2203

Problem 4

To fun problem 4, just follow the procedure in codes ‘Problem4’.

4.1 Compute Emission Parameter

Codes: emission.py --- compute emission parameter

Output file: emission.file

Format: [value] [word] [tag]

Run: python emission.py ner.counts emission.file

Screen Cut of Output file:

```
7.07638962601e-05 O history
0.00197541703248 I-MISC Taleban
5.89699135501e-06 O 10.19
5.89699135501e-06 O 146.2
1.7690974065e-05 O releases
0.000179726815241 I-PER Forget
5.89699135501e-06 O blessed
0.000100248853035 O thought
0.000599940005999 I-ORG Brann
0.00153643546971 I-MISC Tamil
9.99900009999e-05 I-ORG party
0.00116822429907 I-PER Salim
1.7690974065e-05 O specify
0.000362056480811 I-LOC Whittier
1.7690974065e-05 O lawsuits
8.98634076204e-05 I-PER Trigger
```

4.2 Replace Low Frequent Words by ‘_RARE_’

Codes: replace.py --- replace low frequent words by ‘_RARE_’

Checker.py --- here, only return ‘_RARE_’ when input a low frequent word

Output File: ner_train_replaced.dat --- containing the new training data set

ner_replaced.counts ---

Run: python replace.py ner_train.dat ner_train_replaced.dat no_opt

python count_freqs.py ner_train_replaced.dat > ner_replaced.counts

Screen Cut of Output File:

COMS 4705
HW 1
Kaili Zhang
kz2203

```
EU I-ORG
_RARE_ O
German I-MISC
call O
to O
boycott O
British I-MISC
_RARE_ O
. O
```

```
7 WORDTAG O camp
10 WORDTAG O research
4 WORDTAG I-PER Games
7 WORDTAG O grenades
13 WORDTAG O 11th
19243 WORDTAG O _RARE_
5 WORDTAG O iron
7 WORDTAG I-PER Baker
7 WORDTAG O true
7 WORDTAG I-LOC Auckland
9 WORDTAG O shown
```

4.3 Baseline Tagger & Evaluation

Codes: base_tagger.py --- produce tags based on y = argmax e(x|y)

Output File: base_pre.file

Format: [word] [tag] [value]

Run: python base_tagger.py prior.file ner_derv.key base_pre.file

Screen Cut of Output File:

```
CRICKET O B-LOC -0.200670695462
- O B-LOC -0.200670695462
LEICESTERSHIRE I-ORG B-LOC -0.200670695462
TAKE O B-LOC -0.200670695462
OVER O B-LOC -0.200670695462
AT O B-LOC -0.200670695462
TOP O B-LOC -0.200670695462
AFTER O B-LOC -0.200670695462
INNINGS O B-LOC -0.200670695462
VICTORY O B-LOC -0.200670695462
. O B-LOC -0.200670695462
```

Evaluation:

Run: python base_tagger.py prior.file ner_dev.key base_pre.file

Result:

COMS 4705
HW 1
Kaili Zhang
kz2203

```
Found 13766 NEs. Expected 5931 NEs; Correct: 3144.

         precision       recall          F1-Score
Total:   0.228389        0.530096        0.319236
PER:     0.429461        0.225245        0.295503
ORG:     0.522908        0.392377        0.448335
LOC:     0.147927        0.877317        0.253167
MISC:    0.647123        0.647123        0.647123
```

Problem 5

Problem 5.1 Calculate Log Probability of Trigrams

Codes: trigram.py --- compute trigram probability

Output file: trigram.file

Format: [tag1] [tag2] [tag3] [value]

Run: python trigram.py ner_replaced.counts trigram.dat

Screen Cut of Output file:

```
O O B-MISC -11.1482615512
I-ORG O I-MISC -5.16151532506
I-MISC I-MISC I-MISC -1.34476590242
O I-ORG I-MISC -6.44888939415
I-PER I-PER STOP -3.24188588731
* I-MISC STOP -6.21860011969
* I-PER STOP -5.82082449509
O I-PER I-PER -0.394623073383
O I-ORG STOP -4.08176578002
I-LOC I-LOC O -0.162111929085
```

Problem 5.2 Calculate Log Probability of Trigrams

Codes: viterbi_tagger.py --- Using Viterbi algorithm to tag

Output file: trigram.file

Format: [tag1] [tag2] [tag3] [value]

Run: python Viterbi_tagger.py prior.file trigram.dat ner_dev.dat viterbi_pre.file no_opt

Screen Cut of Output file:

```
CRICKET O -8.67251828825
- O -13.8736173281
LEICESTERSHIRE O -16.2099590892
TAKE O -18.5463008503
OVER O -20.8826426114
AT O -28.2086896865
TOP O -30.5450314475
AFTER O -40.1813264884
INNINGS O -42.5176682494
VICTORY O -44.8540100105
. O -50.7701096748

LONDON I-LOC -6.6736471924
1996-08-30 O -18.1697856567

```

Evaluation:

Run: python eval_ne_tagger.py ner_dev.key viterbi_pre.file

Result:

```
Found 4704 NEs. Expected 5931 NEs; Correct: 3649.

         precision      recall          F1-Score
Total:   0.775723       0.615242        0.686225
PER:     0.763928       0.596844        0.670128
ORG:     0.611855       0.478326        0.536913
LOC:     0.876458       0.696292        0.776056
MISC:    0.830065       0.689468        0.753262
```

Obviously, comparing to the baseline, viterbi tagger performs much better. The precision, recall and F1-Score show good result. And what's more, for different tags, it also shows a stable performance.

However, viterbi cost much on time.

Problem 6

1. For problem 6, I designed 5 return value instead of '_RARE_':

'_CAPALL_' for words with more than one characters and all capital characters;

'_CAP_' for single captital character;

'_CAPFST_' for words with capital headers;

'_NUM_' for numbers;

'_ORG_' for words containing only capitals and dots.

COMS 4705
HW 1
Kaili Zhang
kz2203

2. Then, we need to run the whole procedure to tag.

1) Replace low frequent words in training set;

2) Count frequency of WORDTAG, 1-GRAM, 2-GRAM, 3-GRAM

3) Calculate emission based on replaced training set

4) Calculate trigram based on replaced training set

5) Use viterbi tagger to tag the testing set

6) Evaluate.

To run this program, please follow the procedure of ‘Problem6.py’

3. Here is the final evaluation result:

```
Found 5790 NEs. Expected 5931 NEs; Correct: 4330.

         precision      recall        F1-Score
Total:   0.747841       0.730062      0.738845
PER:     0.810364       0.774211      0.791875
ORG:     0.544127       0.668161      0.599799
LOC:     0.843521       0.752454      0.795389
MISC:    0.838411       0.687296      0.755370
```

We can see that, after using 6 categories to represent these low forewent words instead of 1, we gain a much better result again!

Correct tag number increase by near 700, Even the three main options are not as well as previous one.

I think that is because the so call ‘long-tail ‘phenomenon. Since in testing set there may appear lots of unseen words and lots none words appear rarely.