COMP 150 NLP

Fall 2016

Problem Set 4

Statistical Parsing

Mingzhe Li

1. Preprocessing

   This part is very similar to the previous ones. Just initialize the vocabulary from the training set and do the work. First sentences of the training set and test set are printed out.

   Program output:

```
(S (NP (NP (NNP <UNK>) (NNP Vinken)) (NP|<,-ADJP-,> (, ,)
(NP|<ADJP-,> (ADJP (NP (CD 61) (NNS years)) (JJ old)) (, ,))))
(S|<VP-.> (VP (MD will) (VP (VB join) (VP|<NP-PP-NP> (NP (DT the)
(NN board)) (VP|<PP-NP> (PP (IN as) (NP (DT a) (NP|<JJ-NN> (JJ
nonexecutive) (NN director)))) (NP (NNP Nov.) (CD 29))))))
(. .)))

(S (NP (NP (JJ <UNK>) (NN trading)) (PP (IN during) (NP (DT the)
(NN session)))) (S|<VP-.> (VP (VBD was) (VP (VBN <UNK>) (PP
(ADVP+RB largely) (PP|<TO-NP> (TO to) (NP (NP (DT a) (NN round))
(NP|<PP-PP-,-SBAR> (PP (IN of) (NP (NN buy) (NNS programs)))
(NP|<PP-,-SBAR> (PP (IN near) (NP (DT the) (NN close))) (NP|<,-
SBAR> (, ,) (SBAR (WHNP+WDT which) (S+VP (VBD helped) (S+VP (VB
offset) (NP (NP (DT the) (NN impact)) (NP|<PP-PP> (PP (IN of)
(NP+NN profit-taking)) (PP (IN among) (NP (JJ blue) (NNS
chips))))))))))))))) (. .)))
```

2. PCFG

   Nothing too fancy for this part. After learning the PCFG by calling the built-in induce function, I just created a dictionary to store/map all the NP nonterminal.

   Program output:

```
Total number for NP nonterminal:  1539

The most probable 10 productions for the NP nonterminal:

      [NP -> DT NN [0.122667],

      NP -> NP PP [0.11057],
```

```
NP -> NNP NNP [0.0495922],
NP -> DT NP|<JJ-NN> [0.0377142],
NP -> JJ NNS [0.0343752],
NP -> DT NNS [0.0255624],
NP -> JJ NN [0.0201981],
NP -> NN NNS [0.0183918],
NP -> DT NP|<NN-NN> [0.0168592]]
```

3. Training

3.1 – 3.3

Building the inverted index right to left is not too complicated. I pretty much followed the algorithm to do the dynamic programming in Parse(), populating the dynamic programming table with log probabilities of every constituent spanning a sub-span of a given test sentence (i, j) and storing the appropriate back-pointers.

The log probability and the parse tree were reported in the output. The spirit is the same as in any DP, but this table is particularly complex. I consulted some senior students to get it working. I found myself learning many python coding paradigms during the process.

Program output:

```
The log probability of the 5-token sentence:  -33.9906326282
The parse tree for the 5-token sentence:
(S
  (NP+NNS Terms)
  (S|<VP-.>
    (VP (VBD were) (VP|<RB-VP+VBN> (RB n't) (VP+VBN disclosed)))
    (. .)))
```

3.4 – 3.5

I defined a function to count the buckets from the test set. After that, it's just calling the built-in normal form function to analyze each bucket. For detailed results, please refer to the attached test_* and gold_* files.

I noticed the performance results obtained by EVALB are poorer as the buckets grow larger.

Program outputs:

```
      Number of sentences in each bucket:  23 134 187 86 28
Bucket1:
      Bracketing Recall         =  79.49
      Bracketing Precision      =  81.82
      Bracketing FMeasure       =  80.64
      Complete match            =  25.32
      Average crossing          =   0.94
Bucket2:
      Bracketing Recall         =  79.59
      Bracketing Precision      =  81.80
      Bracketing FMeasure       =  80.68
      Average crossing          =   1.04
      Tagging accuracy          =  89.17
Bucket3:
      Bracketing Recall         =  70.67
      Bracketing Precision      =  72.22
      Bracketing FMeasure       =  71.44
      Average crossing          =   3.03
      Tagging accuracy          =  87.59
Bucket4:
      Bracketing Recall         =  61.98
      Bracketing Precision      =  62.41
      Bracketing FMeasure       =  62.19
      Average crossing          =   6.72
      Tagging accuracy          =  88.18
Bucket5:
      Bracketing Recall         =  57.04
      Bracketing Precision      =  59.70
```

Bracketing FMeasure	=	58.34
Average crossing	=	9.14
Tagging accuracy	=	87.07