W4705 HW2 (wj2227)

For fast run of all scripts, please use *run.sh*

Question 4

Part1: how to run your code step by step

To run the code, please follow the following commands:

python count_cfg_freq.py parse_train.dat > cfg.counts
python question4.py > parse_train_rev.dat
python count_cfg_freq.py parse_train_rev.dat > cfg_rare.counts

Part2: performance for your algorithm

For replacing rare words in the training corpus, a recursive function was called to find terminal words, in which each recursive call will check the length of list. If the length is 3, then recursively check its second and third list elements; if the length is 2, then check if the second elements has a count >5. If not, replace it with "_RARE". The script revises train data with rare words (count<5) replaced by "_RARE_", and the output result is saved in *parse_train_rev.dat*, after which the counting script *count_cfg_freq.py* was ran again to create new counts and stored in *cfg_rare.counts*.

Part3: observations and comments about your experimental results.

The sample sentences of output is shown below, we can see that rare words are replaced by "_RARE_".

```
["S", ["NP", ["NOUN", "Ms."], ["NOUN", "_RARE_"]], ["S", ["VP", ["VERB", "plays"], ["NP+NOUN", "_RARE_"]], [".", "."]]]
["S", ["NP", ["DET", "The"], ["NP", ["NOUN", "_RARE_"], ["NP", ["NOUN", "auto"], ["NOUN", "maker"]]]], ["S", ["NP", ["ADJ", "last"], ["NOUN", "year"]], ["VP",
["VERB", "sold"], ["VP", ["NP", ["NUM", "_RARE_"], ["NOUN", "cars"]], ["PP", ["ADP", "in"], ["NP", ["DET", "the"], ["NOUN", "U.S."]]]]]]]
["S", ["NP", ["NOUN", "_RARE_"], ["NP", ["NOUN", "_RARE_"], ["NOUN", "Inc."]]], ["S", ["VP", ["VERB", "increased"], ["VP", ["NP", ["PRON", "its"], ["NOUN",
"quarterly"]], ["VP", ["PP", ["PRT", "to"], ["NP", ["NUM", "10"], ["NOUN", "cents"]]], ["PP", ["ADP", "from"], ["NP", ["NP", ["NUM", "seven"], ["NOUN", "cents"]],
["NP", ["DET", "a"], ["NOUN", "share"]]]]]]]], [".", "."]]]
["S", ["NP", ["DET", "The"], ["NP", ["ADJ", "new"], ["NOUN", "rate"]]], ["S", ["VP", ["VERB", "will"], ["VP", ["VERB", "be"], ["ADJP", ["ADJ", "payable"], ["NP",
["NOUN", "_RARE_"], ["NUM", "15"]]]]]], [".", "."]]]
["S", [".", "``"], ["S", ["ADVP+ADV", "_RARE_"], ["S", ["NP", ["DET", "the"], ["NOUN", "_RARE_"]], ["S", ["VP", ["VERB", "did"], ["VP", ["ADV", "not"], ["VP",
["ADVP+ADV", "really"], ["VP", ["VERB", "_RARE_"], ["PP", ["ADP", "in"], ["NP", ["DET", "this"], ["NOUN", "_RARE_"]]]]]]]], ["S", [".", "."], [".", "''"]]]]]
```

Figure 1.

Question 5

Part1: how to run your code step by step

To run the code, please follow the following commands:

python question5.py > parse_dev_result
python eval_parser.py parse_dev.key parse_dev_result

Part2: performance for your algorithm

For this question, CKY algorithm was implemented to return the predicted parse tree for the test corpus. For the data given, it takes about 1 to 2 minute to run program *question5.py*. The result is output to file *parse_dev_result* The performance of the algorithms is show below:

```
I:\W4705-NLP\hw2\hw2>python eval_parser.py parse_dev.key parse_dev_result
```

Type	Total	Precision	Recall	F1 Score
.	370	1.000	1.000	1.000
ADJ	164	0.827	0.555	0.664
ADJP	29	0.333	0.241	0.280
ADJP+ADJ	22	0.542	0.591	0.565
ADP	204	0.955	0.946	0.951
ADV	64	0.694	0.531	0.602
ADVP	30	0.250	0.133	0.174
ADVP+ADV	53	0.756	0.642	0.694
CONJ	53	1.000	1.000	1.000
DET	167	0.988	0.976	0.982
NOUN	671	0.752	0.841	0.794
NP	884	0.630	0.526	0.573
NP+ADJ	2	0.286	1.000	0.444
NP+DET	21	0.783	0.857	0.818
NP+NOUN	131	0.636	0.573	0.602
NP+NUM	13	0.214	0.231	0.222
NP+PRON	50	0.980	0.980	0.980
NP+QP	11	0.667	0.182	0.286
NUM	93	0.984	0.645	0.779
PP	208	0.588	0.625	0.606
PRON	14	1.000	0.929	0.963
PRT	45	0.957	0.978	0.967
PRT+PRT	2	0.400	1.000	0.571
QP	26	0.647	0.423	0.512
S	587	0.626	0.782	0.695
SBAR	25	0.091	0.040	0.056
VERB	283	0.683	0.799	0.736
VP	399	0.559	0.594	0.576
VP+VERB	15	0.250	0.267	0.258
total	4664	0.713	0.713	0.713

Figure 2.

Part3: observations and comments about your experimental results.

We can see that the total precision, recall and F1-Score are 0.713 based on the evaluation script. In the table, there're several nonterminals with very low precision, recall and F1-score, such as "SBAR", "ADJP", "ADVP", "NP+NUM", "NP+QP" and "VP+VERB". It may partially due to the way of conversion to Chomsky Normal Form. It is also due to the nature of these types in English grammar that leads to the difficulties to predict correctly. Because of the high demand of calculation for Viterbi algorithm, this script takes about 1 to 2 minutes to run on my computer.

Note that there're small deviation of performance of classmates (0.711 – 0.715), which may be explained by the fact that there may exist multiple X for $\max_{x \in N} \pi[1, n, X]$, and consequently multiple parse tree with same probabilities.

Question 6

Part1: how to run your code step by step

To run the code, please follow the following commands:

python count_cfg_freq.py parse_train_vert.dat > cfg_vert.counts
python question6_1.py > parse_train_vert_rev.dat
python count_cfg_freq.py parse_train_vert_rev.dat > cfg_vert_rare.counts
python question6_2.py > parse_dev_vert_result
python eval_parser.py parse_dev.key parse_dev_vert_result

Part2: performance for your algorithm

In this part, vertical markovization is used in the training data, and the performance of testing corpus is as below:

```
I:\W4705-NLP\hw2\hw2>python eval_parser.py parse_dev.key parse_dev_vert_result
      Type       Total   Precision      Recall     F1 Score
===============================================================
         .         370     1.000         1.000       1.000
       ADJ         164     0.689         0.622       0.654
      ADJP          29     0.300         0.414       0.348
  ADJP+ADJ          22     0.591         0.591       0.591
       ADP         204     0.960         0.951       0.956
       ADV          64     0.759         0.641       0.695
      ADVP          30     0.333         0.167       0.222
  ADVP+ADV          53     0.700         0.660       0.680
      CONJ          53     1.000         1.000       1.000
       DET         167     0.988         0.994       0.991
      NOUN         671     0.796         0.845       0.820
        NP         884     0.613         0.543       0.576
    NP+ADJ           2     0.333         0.500       0.400
    NP+DET          21     0.944         0.810       0.872
   NP+NOUN         131     0.613         0.664       0.637
    NP+NUM          13     0.375         0.231       0.286
   NP+PRON          50     0.980         0.980       0.980
     NP+QP          11     0.750         0.273       0.400
       NUM          93     0.914         0.688       0.785
        PP         208     0.623         0.635       0.629
      PRON          14     1.000         0.929       0.963
       PRT          45     1.000         0.933       0.966
   PRT+PRT           2     0.286         1.000       0.444
        QP          26     0.650         0.500       0.565
         S         587     0.703         0.814       0.755
      SBAR          25     0.667         0.400       0.500
      VERB         283     0.790         0.813       0.801
        VP         399     0.663         0.677       0.670
   VP+VERB          15     0.294         0.333       0.312

     total        4664     0.741         0.741       0.741
```

Figure 3

It is also noticed that when decreasing the threshold of the counts of rare words (from <5 to <2), the vertical makovization has a slightly improved performance shown below:

I:\W4705-NLP\hw2\hw2>python eval_parser.py parse_dev.key parse_dev_vert_result

Type	Total	Precision	Recall	F1 Score
.	370	1.000	1.000	1.000
ADJ	164	0.704	0.726	0.715
ADJP	29	0.250	0.448	0.321
ADJP+ADJ	22	0.476	0.455	0.465
ADP	204	0.956	0.951	0.953
ADV	64	0.789	0.703	0.744
ADVP	30	0.435	0.333	0.377
ADVP+ADV	53	0.732	0.774	0.752
CONJ	53	1.000	1.000	1.000
DET	167	0.988	0.988	0.988
NOUN	671	0.862	0.878	0.870
NP	884	0.672	0.592	0.629
NP+ADJ	2	1.000	0.500	0.667
NP+DET	21	0.895	0.810	0.850
NP+NOUN	131	0.650	0.710	0.679
NP+NUM	13	0.444	0.308	0.364
NP+PRON	50	0.980	0.980	0.980
NP+QP	11	1.000	0.364	0.533
NUM	93	0.958	0.742	0.836
PP	208	0.678	0.659	0.668
PRON	14	1.000	0.929	0.963
PRT	45	0.955	0.933	0.944
PRT+PRT	2	0.286	1.000	0.444
QP	26	0.737	0.538	0.622
S	587	0.722	0.835	0.774
SBAR	25	0.632	0.480	0.545
VERB	283	0.851	0.887	0.869
VP	399	0.663	0.677	0.670
VP+VERB	15	0.429	0.400	0.414
total	4664	0.773	0.773	0.773

Figure 4

Part3: observations and comments about your experimental results.

1. For the parsing implementing vertical markovation (Figure 3), the total precision, recall and F1-Score is 0.741, which is slightly higher (0.028) than the performance in question 5. We could see that the prediction of some of the types improved a lot (eg. “SBAR”, precision and recall improve from < 0.1 to >0.4). “VP+VERB”, “ADVP”, “ADJP+ADJ” have some improvement, while some other types experience slightly improvement in one or two fields of precision, recall and F1 Score, but have slightly decrease for the rest.
2. It is also found that by defining the rare word to be words with less than 2 counts in training data can lead to slightly improvements of performance, as shown in Figure 4. The precision, recall and F1-Score is 0.773, which is 0.032 higher than the performance when rare counts is defined to be < 5. It may be because that when we define the rare words to be with wider count range, some information has been lost during the process of rare word replacement. The result is output to the file *parse_dev_vert_result*

Part4: additional information that is requested in the problem.

Comparing the result in problem 5 and 6 we could see that under the same conditions, the parsing result of vertical Markovation PCFG is a little better than that of normal CKY PCFG. The vertical Markovation helped by encoding the information of parent non-terminals, so that it has greater capability distinguishing the non-terminals and increase the flexibility of rules. However, at the same time since the number of rules increases dramatically, a larger train corpus may be needed to get a good estimation of the model parameters since the count for each rule will be lower. In the case of parsing problem in this homework, the train corpus is not quite big, which may be a barrier to the real performance of vertical Markovation PCFG method.

For the example below, the SBAR is labeled right in vertical Markovation PCFG, but not in normal PCFG:

Normal PCFG:

```
["S", ["VP", ["VERB", "Conversation"], ["VP", ["VERB", "was"], ["VP", ["VERB", "subdued"], ["VP", ["ADV",
"as"], ["VP", ["ADV", "most"], ["VP", ["VERB", "patrons"], ["VP", ["VERB", "watched"], ["NP", ["DET", "the"],
["NP", ["ADJ", "latest"], ["NP", ["NP", ["NOUN", "market"], ["NOUN", "statistics"]], ["PP", ["ADP", "on"],
["NP+NOUN", "television"]]]]]]]]]]]], [".", "."]]
```

Vertical Markovation PCFG:

```
["S", ["NP^<S>+NOUN", "Conversation"], ["S", ["VP^<S>", ["VERB", "was"], ["VP", ["ADJP^<VP>+ADJ", "subdued"],
["SBAR^<VP>", ["ADP", "as"], ["S^<SBAR>", ["NP^<S>", ["ADJ", "most"], ["NOUN", "patrons"]], ["VP^<S>",
["VERB", "watched"], ["NP^<VP>", ["NP^<NP>", ["DET", "the"], ["NP", ["ADJ", "latest"], ["NP", ["NOUN",
"market"], ["NOUN", "statistics"]]]], ["PP^<NP>", ["ADP", "on"], ["NP^<PP>+NOUN", "television"]]]]]]]], [".",
"."]]]
```

Key:

```
["S", ["NP+NOUN", "Conversation"], ["S", ["VP", ["VERB", "was"], ["VP", ["ADJP+ADJ", "subdued"], ["SBAR",
["ADP", "as"], ["S", ["NP", ["ADJ", "most"], ["NOUN", "patrons"]], ["VP", ["VERB", "watched"], ["VP", ["NP",
["DET", "the"], ["NP", ["ADJ", "latest"], ["NP", ["NOUN", "market"], ["NOUN", "statistics"]]]], ["PP",
["ADP", "on"], ["NP+NOUN", "television"]]]]]]]], [".", "."]]]
```