NLP hw2 – Programming Section   Name: Shenxiu Wu   UNI: sw3196

Question 5.

After running "python eval_parser.py parse_dev.key q5_prediction_file > q5_eval.txt", I get the evaluation on the performance of my model, as table 1 shown below.

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	0.333	0.133	0.190
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	0.842	0.795
NP	884	0.626	0.525	0.571
NP+ADJ	2	0.286	1.000	0.444
NP+DET	21	0.783	0.857	0.818
NP+NOUN	131	0.641	0.573	0.605
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	0.593	0.630	0.611
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	0.714	0.714	0.714

table 1

As we can see above, the question 5's model actually performed quite not good in both precision and recall on the non-terminals ADJP, ADVP, NP+NUM, SBAR and VP+VERB; On contrary, for the non-terminals NP+ADJ, it has a high score in Recall but lower one in Precision. Similarly, NP+QP has a relatively high score in Precision but lower one in Recall. This may be explained by the low total appearances of these types in the development data, ranging from 2 to 30, which is relatively small. But this theory is not absolute. For example, the types such as NP+DET and PRON still have few appearances in total but they are predicted accurately with high F1 Scores. The precision and F1 Scores for the VP and NP are not high, just at mediocre level. I hold the view that the reason of this phenomenon is just as the problem mentioned in question 6. I strongly believe we improve their precision in question 6 after modified. Below figure 1 shows a graph comparing the performance of each nonterminal, which makes us directly evaluate the model easier.

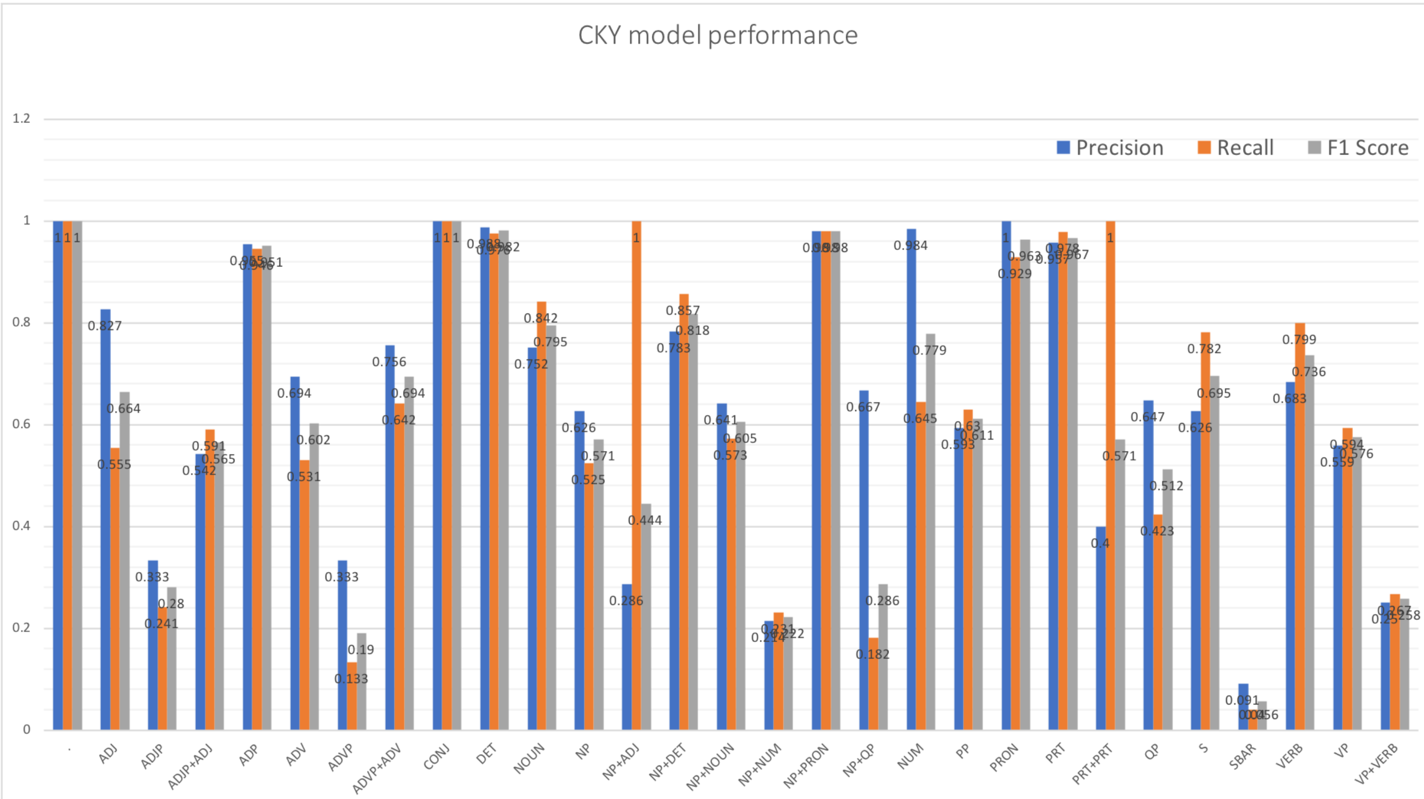

Figure 1

What's more, as the below screenshot states, the running time of question 5 is `25.1441979408` (s). I think the running time is reasonable and acceptable.

```
dyn-160-39-149-78:hw2 shenxiu-wu$ python parser.py q5 parse_train.RARE.dat parse
_dev.dat q5_prediction_file
running time: 25.1441979408 s
dyn-160-39-149-78:hw2 shenxiu-wu$
```

Figure 2

Question 6.

I believe I can still use the code and algorithm in question 5 to implement in question 6. There is no necessary to change the algorithm because the Q6 running time (which is `40.0279300213 (s))` is still reasonable, as we can see in below screenshot(Figure 3).

```
dyn-160-39-149-78:hw2 shenxiu-wu$ python parser.py q6 parse_train_vert.RARE.dat
parse_dev.dat q6_prediction_file
running time: 40.0279300213 s
```

Figure 3

Also, although with vertical markovization having a much larger set of non- terminals N, the dynamic programming and back pointer used greatly help the program run in an efficient time. What's more, I mainly use dictionary and list to store those processed data. Hence, the algorithm in question 5 is scalable to deal with tremendously more non-terminals. After running "python eval_parser.py parse_dev.key q6_prediction_file > q6_eval.txt", I get the evaluation on the performance of my model, as table 2 shown below.

Type	Total	Precision	Recall	F1 Score
.	370	1.000	1.000	1.000
ADJ	164	0.689	0.622	0.654
ADJP	29	0.324	0.414	0.364
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	0.417	0.167	0.238
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	0.795	0.845	0.819
NP	884	0.617	0.548	0.580
NP+ADJ	2	0.333	0.500	0.400
NP+DET	21	0.944	0.810	0.872
NP+NOUN	131	0.610	0.656	0.632
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	0.704	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	0.742	0.742	0.742

Table 2

As the total scores show, in almost all cases, the second training file with vertical markovization make out PCFG model's performance improve slightly. As mentioned above, the performance of VP and NP types are more or less improved slightly. It turned out the using of vertical markovization works. This is self-explanatory that this method gives more importance to context and avoid making the independence assumption become too strong. I noticed that ADJ, PRT+PRT and PRT's precision decreased slightly with using the vertical markovization. However, the overall performance and average scores have definitely increased. Similarly, below figure 4 shows a graph comparing the performance of each nonterminal, which makes us directly evaluate the model in question 6.

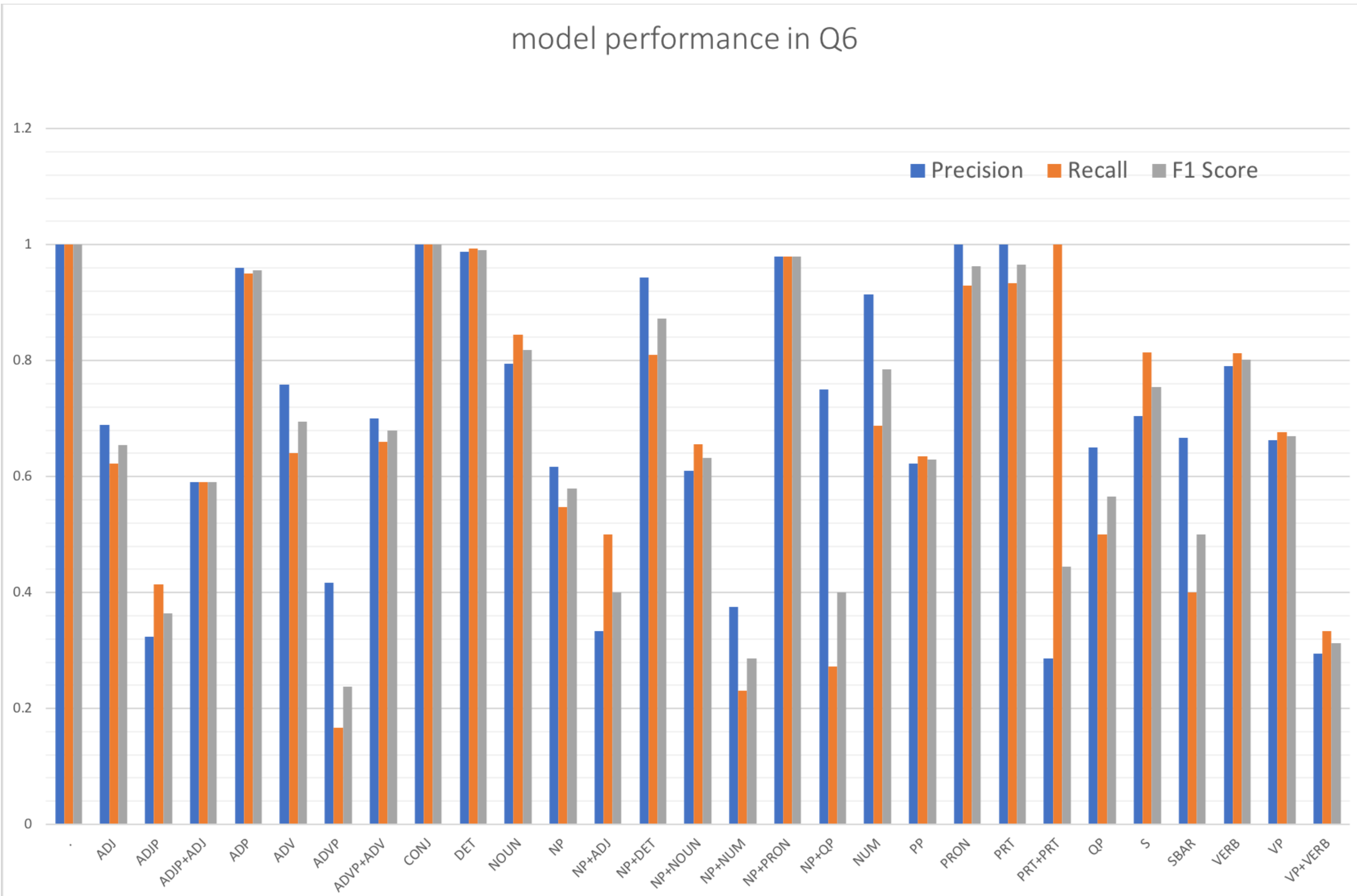

Figure 4

Comparison:

Below figure 5 and figure 6 contains comparisons of the precision and recall indices of the performance results got from Q5 and Q6, indicated by Precision_Q5, Precision_Q6, Recall_Q5 and Recall_Q6.

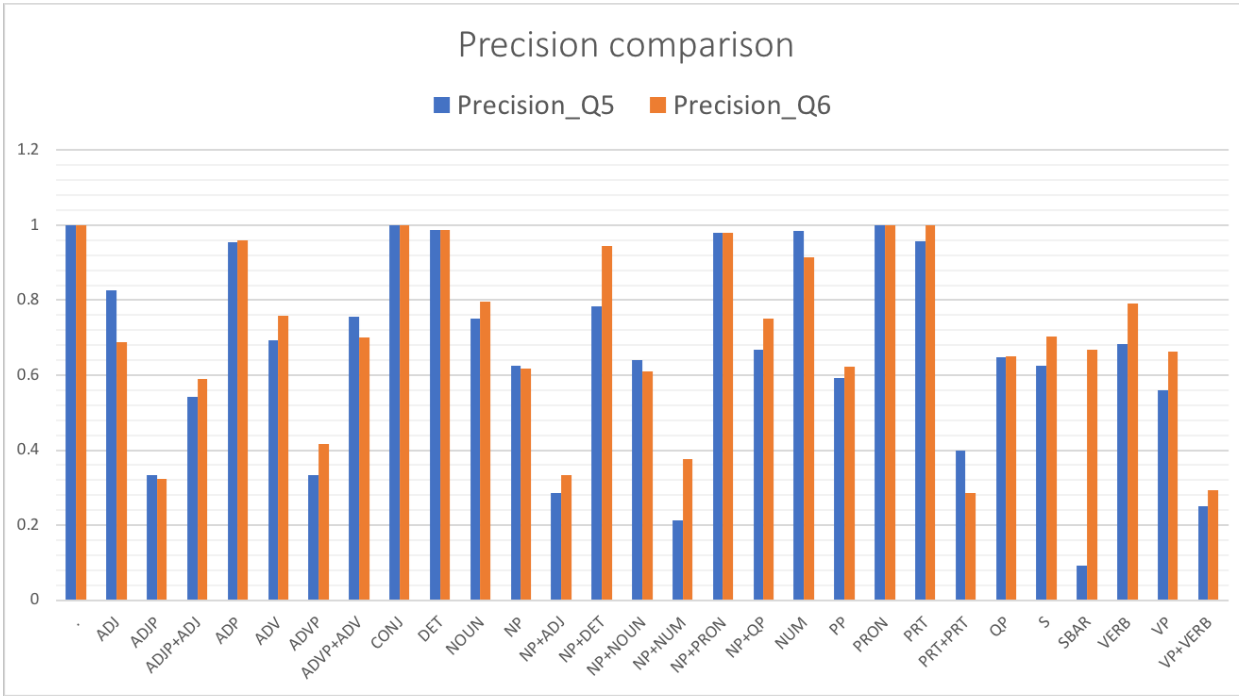

Figure 5

From figure 5, we could clearly figure out the precision of type SBAR is improved tremendously.

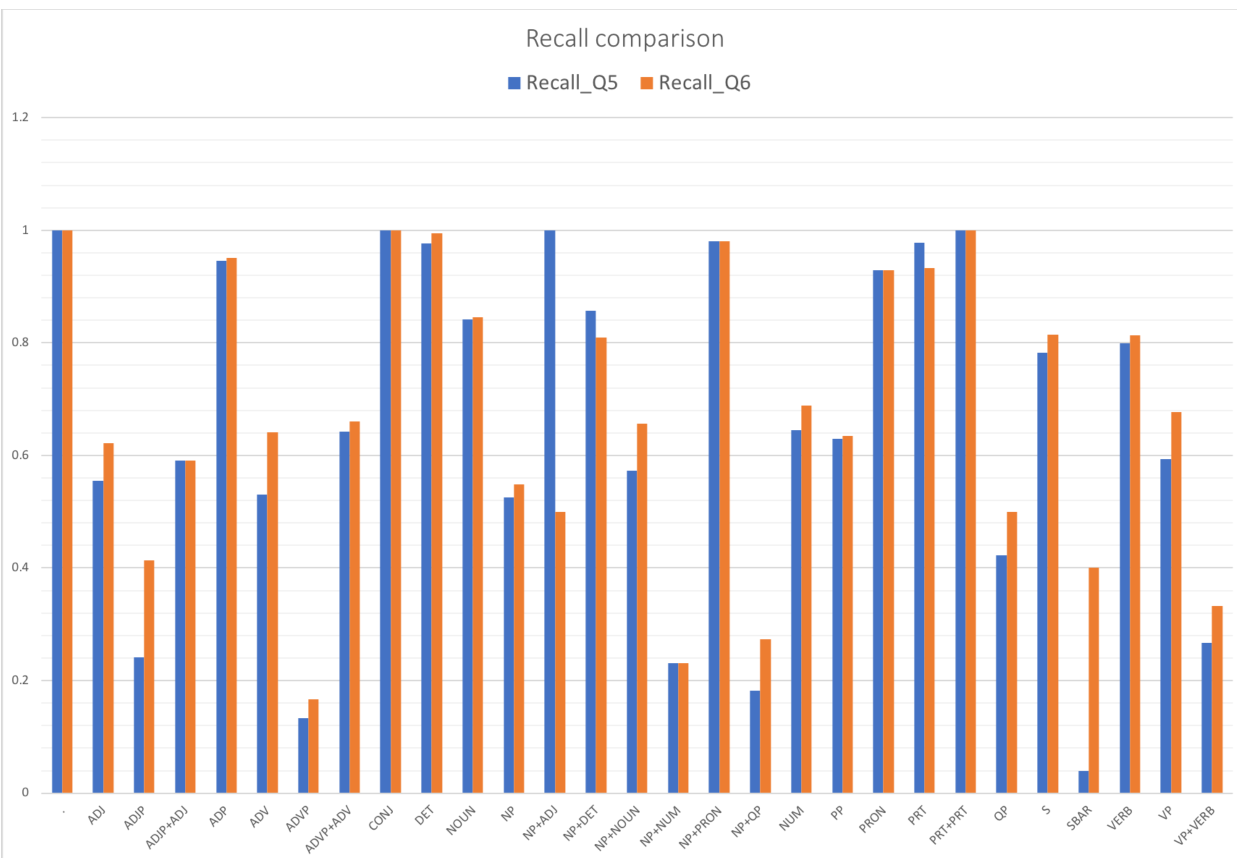

Figure 6

From figure 6, we could see almost all of these types' performance are improved, except the NP+ADJ.

I also make a comparison of F1 scores between question 5 and 6's performances. As shown in the figure 7 below.

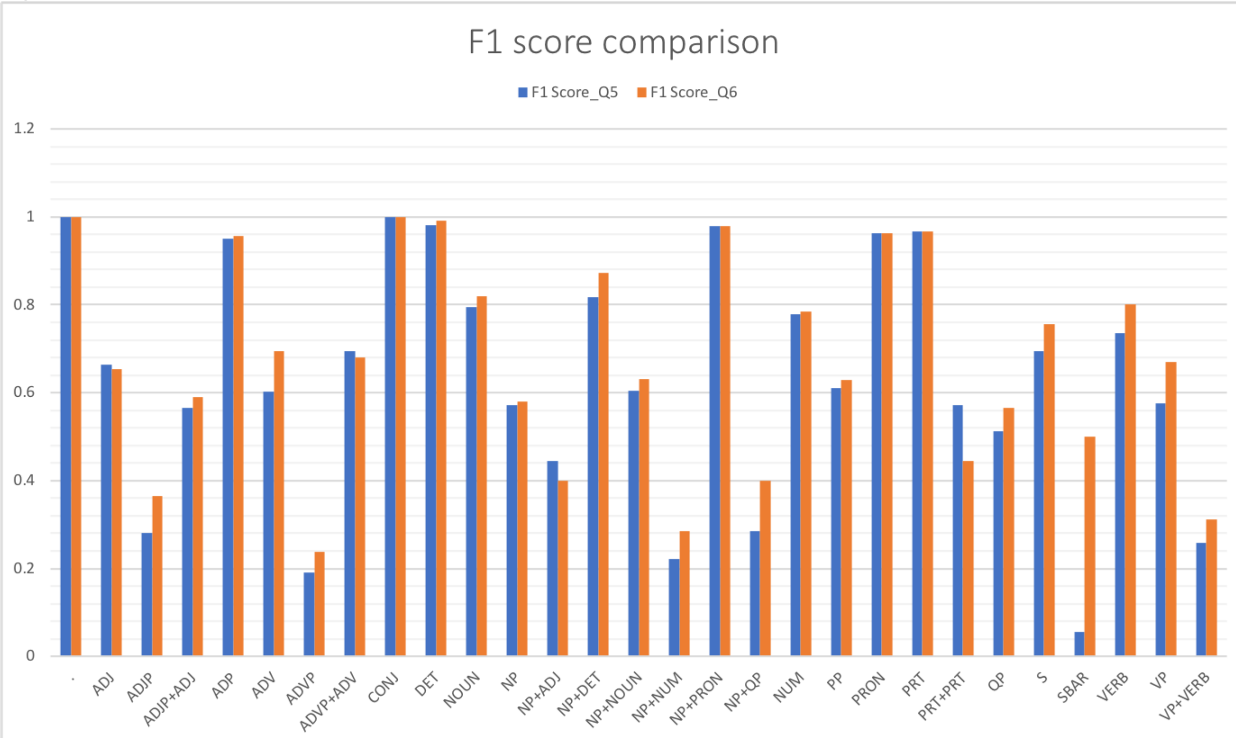

Figure 7

From figure 7 we could know almost all types' F1 scores have been improved except NP+ADJ decreased slightly, but it's still in reasonable range.