NLP-HW2-Report
Kuo Lin kl2734

Files:
Report: This report for HW2
RARE.py: For replacing the rare words
CKY.py: Implement for CKY algorithm, for Q5
CKY_MARKOV.py: Implement for CKY algorithm, for Q6
lab2.sh: shell for applying all the commands
And all other files in the hw2.tar.gz given by assignment.

SHELL generated files:
parse_train_withrare.dat: The training file with replacement of '_RARE_'.
parse_train_vert_withrare.dat: The training file with replacement of '_RARE_'.
count_train: count from parse_train_withrare.dat
count_train_vert: count from parse_train_vert_withrare.dat
parse_result: parse result for Q5, with CKY.py
parse_result_VM: parse result for Q6, with CKY_MARKOV.py
CKYperformance: evaluate of parse_result, with eval_parser.py
CKYVMperformance: evaluate of parse_result_VM, with eval_parser.py

Report:
Q4:
Running code:
Python _RARE_.py inputfile outputfile
Performance:
Running time: Almost immediately
Basic design idea and observations:
The code run over the whole tree set first, applied a recursive function to find every end point of the tree, and using a dictionary to record the number of time of each word appeared. Then it run through the tree set second time, replace the word with count<5 with _RARE_.
I found that out of about 10000 word, about 8500 of them are rare words, only about 1500 are frequent words, it's a quite low rate.

Q5:
Running code:
Python _RARE_.py parse_train.dat outputfile
Python count_cfg_freq.py inputfile>outputfile
Python CKY.py countfile evaluatefile outputfile
Performance:
Running time: About 4 minutes.
Performance:

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	0.579	0.615	0.597
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	0.629	0.785	0.698
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

Basic design idea and observations:

It's a simple implement of CKY algorithms, we can see that the function does good at predicting '.' , 'COUNJ' , 'DET' , 'PRON' , 'ADP', these sets of part-of-speech have only very limited vocabulary size in English, this is the reason why they performs good. It's encouraging that the algorithm also performances relatively good at 'VERB' and NOUN, these word varies much in English, and this indicate the algorithm has a good output. We can also find that the algorithms doesn't preform good when facing the part-of-speech with low total number, this indicate the algorithm cannot capture low frequent part-of-speech.

Q6:

Running code:

Python _RARE_.py parse_train_vert.dat outputfile

Python count_cfg_freq.py inputfile>outputfile

Python CKY.py countfile evaluatefile outputfile

Performance:

Running time: About 7 minutes

Performance:

Type	Total	Precision	Recall	F1 Score
.	370	0.995	1.000	0.997
ADJ	164	0.698	0.634	0.665
ADJP	29	0.325	0.448	0.377
ADJP+ADJ	22	0.583	0.636	0.609
ADP	204	0.960	0.951	0.956
ADV	64	0.750	0.656	0.700
ADVP	30	0.500	0.233	0.318
ADVP+ADV	53	0.706	0.679	0.692
CONJ	53	1.000	1.000	1.000
DET	167	0.988	0.988	0.988
NOUN	671	0.790	0.842	0.815
NP	884	0.607	0.551	0.578
NP+ADJ	2	0.333	0.500	0.400
NP+DET	21	0.895	0.810	0.850
NP+NOUN	131	0.576	0.634	0.604
NP+NUM	13	0.429	0.231	0.300
NP+PRON	50	0.980	0.980	0.980
NP+QP	11	0.600	0.273	0.375
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	0.500	1.000	0.667
QP	26	0.611	0.423	0.500
S	587	0.695	0.809	0.748
SBAR	25	0.562	0.360	0.439
VERB	283	0.776	0.806	0.790
VP	399	0.640	0.654	0.647
VP+VERB	15	0.294	0.333	0.312
total	4664	0.739	0.739	0.739

Basic design idea and observations:

The CKY algorithm is exactly the same as one above, the difference is this one generated two set of q(X → Y1 Y2), one contains the whole set of rules as the one in Q5, the other contains only the rules with log-probability larger than -9. When applying CKY, the algorithm will first predict the tree using the rules in the second group, if can successfully predict a tree, output it, if can't, the algorithm will then using the rules in the first group to predict a tree.

We can see that the performance is better than the one in Q5, the general performance are better for every part-of-speech. The most interesting part is it does much better at 'SBAR' and all other part-of-speech with low frequency.