

Report for HW2

Quick Start

To run everything, simply place

- All default files from `hw2.tar.gz` . They are

```
count_cfg_freq.py
eval_parser.py
parse_dev.dat
parse_dev.key
parse_train_vert.dat
parse_train.dat
pretty_print_tree.py
```

- `run.sh`
- `pcfg.jar`

into the same folder.

Then run

```
sh run.sh
```

To run all of question 4, 5, and 6, and see outputs and run times for the CKY operations of question 5 and 6.

For specifics of how to use `pcfg.jar` , look at the `run.sh` . `pcfg.jar` only does two things -- replace rare words and count CKY.

Replacing

To replace words, `pcfg.jar` needs a count file, the file to be replaced, and the output file name.

```
java -jar pcfg.jar <countfile> <filetoreplace> <outputfile>
```

CKY

To run the cky algorithm, `pcfg.jar` needs a count file, the dev data, and the output file name

```
java -jar pcfg.jar <countfile> <devdata> <outputfile>
```

The default commands for both are in the `run.sh` script.

Runtime

Runtime is fast for CKY due to data structure optimization. For example, I wrapped the pi-table in a `PiTable` class that builds the table all using hashes and maps the 2d table to an 1d hashtable. Hence, all lookups are optimized.

Runtime for Question 5 is **25 seconds** on my machine.

Runtime for Question 6 is **50 seconds** on my machine.

Results

Question 5

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	0.632	0.529	0.576
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	0.588	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	0.714	0.714	0.714

Question 6

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

We see significant improvement from Question 5 to Question 6 due to vertical markovization.

I did not change my code from Question 5 to 6 simply because data structures were already optimized, hence the run time only increased from 25 to 50 seconds. I did not see the need to modify anything.