11-711 Assignment 2: Parsing

Xin Qian
xinq@cs.cmu.edu

Abstract

This report summarizes the implementation details and experimental findings on building a generative unlexicalized PCFG parser. The basic generative parser consists of an Markovization annotator and a Viterbi-driven CKY parser. The basic requirement best performance is achieved with additional tag splitting under maxTrainLength=1000 and maxTestLength=400 with a decoding speed of 446s and a F1=80.71(i5 1.5GHz MacBook Pro). The best extra credit performance is achieved with coarse-to-fine pruning with a decoding speed of 283s and an F1=81.09. Extra extensive experiments were done on a server such that despite several over optimistic reported absolute speed value, we could rely on the relative speed up to draw the conclusion that coarse-to-fine pruning helps.

1 Introduction

In this project we implemented a generative unlexicalized PCFG parser with several optimization techniques. The performance of the parser is evaluated by F1 score and decoding time. In this report we discuss how we boost the effectiveness through markovization and tag-splitting and the through coarse-to-fine pruning. Our best basic model, achieves an F1 score of 80.71 with a decoding time of 446s. Our best advanced coarse-to-fine pruning model, achieves an F1 score of 81.09 with a decoding time of 283 (1.57 times faster than earlier).

1.1 Minimal requirements

Table 1 summarizes the results for maxTrainLength=maxTestLength=15. Table 2 summarizes the results for maxTrainLength=1000 and maxTestLength=40. Table 3 summarizes the results when we exclude pre-terminal for parent annotation with maxTrainLength=1000 and maxTestLength=40. Table 4 summarizes the results when we further refine some of tags by subcategories, again results are reported with maxTrainLength=1000 and maxTestLength=40. For the latter two tables, we omit running time here since the added feature is primarily an effectiveness improvement trick. In brief, h=v=2 works the best as a tradeoff between efficiency and effectiveness. Although h=2 and v=3 achieves a higher F1 value, the speed down is 1.4 times than before, which doesn't worth the benefit.

Table 1: maxTrainLength=maxTestLength=15

v	h	# sym	# bin	# una	F1	time
1	1	489	1164	1449	77.39	1.831
1	2	1521	2247	2392	79.67	4.519
1	INF	3152	3355	3080	79.11	7.444
2	1	930	1412	1943	81.97	2.540
2	2	1962	2495	2886	**84.25**	4.133
2	INF	3593	3603	3574	83.93	7.544
3	1	1992	2873	3650	82.21	4.322
3	2	3024	4413	4817	83.11	6.971
3	INF	4655	5573	5487	82.69	8.316

2 Implementation details

2.1 Efficiency optimization

2.1.1 Coarse-to-fine pruning

Coarse-to-fine pruning trims out unlikely parses by thresholding the likelihood from another coarsely annotated tree. This saves an amount of operations during grammar rule iteration: unnecessary rules are not evaluated as a valid path in fine pass Viterbi calculation. Different

Table 2: maxTrain=1000, maxTest=40

v	h	# sym	# bin	# una	F1	time
1	1	735	2665	3484	71.13	65
1	2	3557	6517	7842	72.97	298
1	INF	14984	14972	14911	72.73	2295
2	1	1436	3165	4751	76.86	112
2	2	4258	7017	9109	79.26	369
2	INF	15685	15472	16178	**79.46**	2647
3	1	3993	7646	11006	78.86	270
3	2	6815	13824	17673	**80.64**	505
3	INF					

Table 3: No preterminal parent annotation

v	h	# symbols	# binary	# unary	F1
1	1	735	2665	3484	71.13
1	2	3557	6517	7842	72.97
1	INF	14984	14972	14911	72.73
2	1	1007	3165	4751	74.46
2	2	3829	7017	9109	76.5
2	INF	15256	15472	16178	76.29
3	1	2028	6131	8687	75.12
3	2	4850	11015	13507	77.07
3	INF	16277	19665	20669	76.82

threshold value decides how much operations to be saved from. The general pipeline of our coarse-to-fine pruning

Some minor implementation choices we have include:

- Log space probability summation is done using `Sloppy.logAdd` where the value to be added is passed in as an += operation. The function also handles `-inf` properly
- Inside pass simulates actual CYK parsing with a bottom-up order while outside pass follows a top-down fashion. This prevents outside score from calculating from unpopulated cells.

2.1.2 Other minor tricks

We optimized the loop order in a way that avoids dead-end rules by first checking either the parent or the child in a binary / unary rule has a non-zero probability before proceeding to evaluate the entire rule. This helps the parser get rid of evaluating exhaustively every rule (low frequency ones as well). A further speedup that hasn't been attempted is to put symbol as the first dimension

Table 4: Tag splitting

v	h	# symbols	# binary	# unary	F1
2	1	1648	3617	5459	78.71
2	2	4470	7805	10097	80.71
2	INF	15897	16343	17289	80.47
3	1	4721	8527	12422	79.62
3	2	7543	15071	19351	81.5
3	INF				

in the 3d array. This ensures successive cache read in computer architecture.

2.2 Effectiveness optimization

2.2.1 Pre-terminal parent annotation

Manning et.al (Klein and Manning, 2003) notes that exhaustively marking all preterminals with their parent category works the most effective. We replicate the idea here and compares the effect of excluding per-terminal and including preterminal.

2.2.2 Tag Splitting: SPLIT-IN, UNARY-EXTERNAL and UNARY-INTERNAL

Tag splitting allows POS tags to be more fine-grained. Here we implemented three types of splitting mentioned in the original paper. SPLIT-IN, UNARY-EXTERNAL and UNARY-INTERNAL. UNARY-EXTERNAL and UNARY-INTERNAL marks nonterminal node of whether they have only one child or has no siblings. As IN tags contain many subcategory, we listed four types with extra notation: prepositions, subordinating conjunctions, except and complementizer. This allows the parsing process to be more linguistically motivated.

3 Experiment results

Our minimal experiment summarizes the effect of varying degree of vertical and horizontal markovization. We conduct a parameter sweep of h={1,2,INF}, v={1(no parent annotation), 2(parent annotation), 3(parent and grandparent annotation)} and train / test combinations of 15 and 15 vs. 1000 and 40. In general, the number of symbols and rules increase along the markovization degree increase. Decoding speed

grows approximately quadratically.

Experiment results from table 2 and 3 validates the idea that pre-terminal parent annotation is more effective than excluding pre-terminal. Comparing the number of symbols, binary and unary rules added after including pre-terminal, the overhead is negligible to speed down the parser. Experiment results from table 2 and 4 validates the effectiveness of tag splitting, for an F1 improvement from 79.26 to 80.71. However, the number of tags increase by 30% to 80%.

References

Dan Klein and Christopher D Manning. 2003. Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*. Association for Computational Linguistics, pages 423–430.