Unlexicalized PCFG Parser: Implementation choices and insights

Harsh Jhamtani
Language Technology Institue
Carnegie Mellon University
jharsh@cs.cmu.edu

Abstract

This report discusses implementation of an unlexicalicalized PCFG parser using CKY algorithm. Various implementation choices are discussed and exeriments are conducted to see the effect of these choices. Penn Treebank data set is used to train and validate various model configurations. Final model is selected based on accuracy on validation data (82.49 F1 score on validation data). The selected model achieves F1 score of 82.28 on test.

1 Introduction

Parsers are often broadly classified into Lexicalized parsers and Unlexicalized parsers. Lexicalized parsers use head words to annotate phrasal nodes, while unlexicalized parsers do not use words for such annotations. Even for a simple unlexicalied parser, there can be many implementation choices with significant effect on accuracy and parsing time. In this report, I discuss a Java implementation of an unlexicalized parser alongwith impact of various optimizations. Lot of techniques in this report are inspired by work of Klein and Manning (Klein and Manning, 2003). The best configurations will be summarized in conclusion (Section 3).

1.1 CKY Algorithm

CKY (CockeYoungerKasami algorithm) (Kasami, 1965) algorithm is a dynamic programming algorithm to efficiently compute the best parse tree given grammar rules. Back pointers are used to reconstruct the best parse so found. Penn Treebank data set is split into training, validation, and test sets.

2 Implementation choices and their impact

Various implementation choices are described in this section. Main techniques tried are markovization, variable degree of markovization, contextualization of tags, different choices in binarization of n-ary rules, etc.

2.1 Simple Speedup Choices

- **Array access patterns:** Note that Java stores the arrays in row-major fashion. So that $Array[symbol]leftIndex][rightIndex]$ may have different access time of elements as compared to $Array[leftIndex][rightIndex][symbol]$ based on the nature of successive queries. Latter was found to be slower in the current problem setting. The reason is that outer loops are over $leftIndex$ and $rightIndex$, so inner loop(s) will all access elements differing in $symbol$. Former array representation is more favorable in this case.

- **Skipping rules with *dead ends*:** All rules $Y(i,j)- > X(i,k)Z(k+1,j)$ can be skipped if either of $X(i,k)$ or $Z(k+1,j)$ has no solution. Similar case can be made for unary rules i.e. $X(i,j)- > Y(i,j)$ can be skipped if $Y(i,j)$ has no subsequent path. This can be achieved by accessing rules using child (for unary rules) or left child(for binary rules) instead of earlier method of getting rules for a given parent, and then ignoring all the rules if earlier mentioned condition applies. For $maxTrainLength$ and $maxTestLength$ to be both 15, this reduced decoding time from 92 to 67 seconds, which is about a 25% reduction.

2.2 Degree of vertical and horizontal markovization

Parameters v and h, representing degree of vertical and horizontal markovization respectively, influence the symbol space and grammar size and rules, thereby having an impact on the speed and accuracy.

Experiments were carried for following paramter space:

- $v = \{0, 1, 2, 3\}$
  Note that notation differs from Klein and Manning (Klein and Manning, 2003) in the sense that no parent annotation is referred by me as $v = 0$ and not $v = 1$
- $h = \{0, 1, 2, \infty\}$
- $(maxTrainLength, maxTestLength) = \{(15, 15), (25, 25), (1000, 40)\}$

v	h	# Symbol	F1	Parsing time (s)
0	1	489	77.54	3.6
0	2	1521	79.57	13.2
0	inf	3152	79.24	35.6
1	1	1380	81.51	8.4
1	2	3063	**82.27**	37.7
1	inf	4900	**82.04**	47.1
2	1	2926	**82.88**	19.6
2	2	5259	**82.50**	16.6
2	inf	7181	**82.01**	20.8
3	1	4849	81.78	29.6
3	2	7691	81.39	19.4
3	inf	9615	81.16	24.2

Table 1: Varying degrees of vertical and horizontal markovization for $maxTrainLength = 15$ and $maxTestLength = 15$ (Note that notation differs from Klein and Manning (Klein and Manning, 2003) in the sense that no parent annotation is referred by me as $v = 0$ and not $v = 1$)

Tables 1, 2 and 3 show symbol space size, F1 score, and parsing time for various values of v and h. Beyond $v = 2$, scores in general decrease. Changing h doesn't cause too much change in F1 score.

2.3 Variable degree of markovization

Not all grammatical constituents occur with same variety of contexts. For example, NP and VP, noun phrases and verb phrases respectively, occur in a large variety of different contexts as compared to many other constituents.

v	h	Symbol	F1	Parsing time
0	1	621	74.01	**27.5**
0	2	2519	75.40	97.3
0	inf	7763	74.98	412.5
1	1	2137	78.32	815.4
1	2	5591	79.42	202.3
1	inf	11986	79.20	379.7
2	1	5180	79.97	152.0
2	2	10903	**80.80**	312.2
2	inf	18226	**80.33**	514.8
3	1	10235	79.57	239.3
3	2	18451	**80.25**	678.4
* 3	inf	26353	-	-

Table 2: Varying degrees of vertical and horizontal markovization for $maxTrainLength = 25$ and $maxTestLength = 25$ (The ones marked with $*$ exceeded memory limit)

The strategy is as follows: After performing vertival markovization of degree v, have repeated iterations over current set of symbols. A symbol S is stripped off it's last existing mark (to generate S') if it's count is below a threshold $thresh$. All occurrences of S are replaced with S', and its count is added to the counts of S'.

I experimented with top configurations from Table 3. The results are shown in Table 4. As can be seen, variable vertical markovization provides reasonable reduction in grammar symbol space with negligible reduction, and sometimes even increment, in accuracy as measured by F1 scores. This reduced grammar symbol space can be useful in compensating increased symbol space in further steps when we try to split tags for specific types of tags.

2.4 Contextualizing tags (Part 1)

Most of the ideas being analyzed in this section will be based on work by (Klein and Manning, 2003)

2.4.1 Annotating preterminals with parents

Earlier we saw vertical markovization of non-terminals (excluding pre-terminals). Same idea can be extended to mark pre-terminals (various Parts-of-speech tags) with their parent symbols.

v	h	# Symbol	F1	Parsing time (s)
0	0	98	65.25	25.7
0	1	2974	75.50	450.3
0	2	3557	73.02	521.3
0	inf	14984	72.73	1578.6
1	0	563	69.49	82.8
1	1	2974	75.50	423.6
1	2	8697	**78.0**	2657.0
*1	inf	-	-	-
2	0	2036	72.64	216.3
2	1	7929	**78.06**	850.8
2	2	18282	**79.89**	4670.1
*2	inf	-	-	-
3	0	5057	72.59	423.9
3	1	17326	78.44	2086.0
*3	2	-	-	-
*3	inf	-	-	-

Table 3: Varying degrees of vertical and horizontal markovization for $maxTrainLength = 1000$ and $maxTestLength = 40$ (The ones marked with $*$ exceeded memory limit)

The intuition being that POS tags occur in varying contexts, and tagging with parent may provide some context. Results at Table 5 show an increase of 2.59 percentage points increase on addition of this feature as compared to $v <= 1, h = 1$ baseline, with addition of relatively few symbols only to the symbol space.

2.4.2 Unary Internal Mark

Consider two grammatical rules of the form $X- > YZ$ and $X- > Y$. Now, for some subtree, let Y be the root. Subsequent rules can either be binary like $Y- > AB$ or unary like $Y- > C$. In english grammar, both the type of rules occur in different contexts. However, since in both cases parent of Y can be X, our formulation will not allow the two contexts to be distinguised. (Klein and Manning, 2003) demonstrate this problem using an example. To deal with this, we can have two symbols for Y: $Y - U$ and Y. The former one corresponds to the case when Y is derived from a unary rule.

Results at Table 5 show increase of 1.02 and 0.58 to the two previous configurations respectively (there are two baselines, so therefore two previous configurations)

v	h	*thresh*	# Symbols	F1
1	2	-	8697	78.0
<=1	2	10	8469	77.95
		20	8331	77.92
		30	**8244**	**77.93**
2	2	-	18282	79.89
<=2	2	10	16954	79.80
		20	16492	**79.94**
		30	**16131**	79.83
<=3	2	50	26889	79.59

Table 4: Varying degrees of vertical and horizontal markovization for $maxTrainLength = 1000$ and $maxTestLength = 40$

2.4.3 Splitting 'DT' POS tag

DT tag covers words like 'a','the','that','this', etc. As per (Klein and Manning, 2003), regular determiners can be distinguished from demonstratives by marking if 'DT' is the only child of the parent in the tree. Regular determiners will usually occur as part of a constituents like noun phrase, and will have siblings.

However, in contrast to results by (Klein and Manning, 2003), this does not provide any significant increase in accuracy. This resulted in *decrease* in F1 score by 0.02 precentage points as compared to previous configuration when baseline is $v <= 1, h = 2$, and an increase of meagre 0.03 points when baseline was $v <= 2, h = 2$

2.4.4 Contextualizing 'RB' POS tag

Similar to above case, adverb tags like 'RB' are overloaded in Penn Treebank. Same as above case, we mark if adverb tag is the only child of its parent. However, this did not provide any increase in accuracy (results in Table 5).

2.5 Yet another speedup choice: Tradeoff between Symbol size vs Grammar size

Running time complexity of CKY depends on the grammar size. A closer look at empirical results reveals that with the size of symbol space and grammar we are dealing with, having a smaller symbol size is faster even if that means having bigger grammar due to implementation optimizations i.e. if there is a trade off available, the system, under certain conditions, can benefit more by reduction of symbol space at cost of increased grammar.

To achieve this, a modification is made to the

Method	# Symbols	F1
v <=1 h =2	8331	77.92
+ Preterminal parent tagging	8551	80.51
+ Internal Unary rule	10121	**81.53**
+ Unary DT	10133	81.51
+ Unary RB	10153	81.31
v <=2 h =2	16492	79.94
+ Preterminal parent tagging	17201	81.78
+ Internal Unary rule	19070	**82.36**
+ Unary DT	19077	**82.39**
+ Unary RB	19101	82.26

Table 5: Results on contextualizing tags for $maxTrainLength = 1000$ and $maxTestLength = 40$

binarization procedure. The last child of multi-ary rule shares the parent with second-last child, instead of a new intermediate node. This brings down the symbol size by 40%, but increase grammar size by about 40%. However, overall the running time is **reduced by 5 times**. Results are shown in Table 6. An added benefit is that F1 score improves by 0.20 percentage points.

Method	Symbol size	Gram. size	F1	Parsing time
$v <= 1$, $h = 2$ with contextual tags	6472	37283	81.62	556 s
$v <= 2$, $h = 2$ with contextual tags	11900	60044	82.19	759 s

Table 6: Results when binarizing is done such that last two children of n-ary rule are siblings

2.6 Refining Part-of-speech tags

Here we discuss and analyze more techniques for contextualizing POS tags.

Method	# Symbols	F1
v <=1 h =2 (Table 6)	6472	81.62
+ Split 'IN'	6506	82.05
+ Split 'AUX'	6652	82.22
+ Split 'CC'	6765	**82.27**
v <=2 h =2 (Table 6)	11900	82.19
+ Split 'IN'	12085	82.32
+ Split 'AUX'	12303	**82.48**
+ Split 'CC'	12447	**82.49**

Table 7: Results on refining POS tags IN, CC, AUX

2.6.1 Splitting IN, CC, AUX Part-of-speech tags

IN, AUX, CC tags in PennTreebank are overloaded. They are split as per following criteria respectively:

- Split is motivated by various articles about linguistic use of prepositions. e.g. `http://grammar.ccc.commnet.edu/grammar/prepositions.htm` and `https://www.englishclub.com/vocabulary/common-prepositions-25.htm`
  IN: IN $\wedge$ LOC $\wedge$ TIME. For 'in','at','on'
  IN: IN $\wedge$ TIME. For 'for', 'since', 'after', 'before'
  IN: IN$\wedge$OF. For 'of'
  IN: IN. For all other IN
- AUX: AUX $\wedge$ BE. For all forms of 'be'
  AUX: AUX $\wedge$ HAVE. For all forms of 'have'
  AUX: AUX. For all other forms
- CC: CC $\wedge$ BUT. For 'but'
  CC: CC $\wedge$ &. For '&'
  CC: CC. For all other forms.

Results are shown in Table 7. SPLIT-IN and SPLIT-AUX provide decent improvements in F1 score.

3 Conclusion

Markovization on original treebank helps significantly in improving the accuracy. Variable degree of markovization offers reduction in grammar and symbol space with negligible reduction in accuracy. Many of the Penn Treebank tags are overloaded - contextualizing such tags can provide non-trivial increase in accuracy. However, some of the techniques like ’Unary DT’ and ’Unary RB’ provide little or no incremental benefit. From Tables 1,2,3, it can be inferred that performance is usually worse on longer sentences.

The final model achieves F1 score of **82.49** on validation and has following features:
- Variable vertical markovization with max. degree 2 and with frequency threshold as 20.
- Horizontal markovization of degree 2.
- ‘Preterminal parent tagging’
- ‘Internal Unary Rule’
- ‘Unary DT’
- ‘SPLIT IN’
- ‘SPLIT AUX’
- ‘SPLIT CC’

Corresponding decoding time is **708.8** seconds. Above model achieves F1 score of **82.28** on test data.

References

Tadao Kasami. 1965. An efficient recognition and syntaxanalysis algorithm for context-free languages. Technical report, DTIC Document.

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