Implementing CKY Parsing with Probabilistic Context Free Grammar Algorithms for NLP 11-711: Assignment 2

Kinjal Jain[1]

Abstract— **This paper addresses various strategies employed and experiments conducted to efficiently implement a generative, array-based CKY parser for English using Probabilistic Context Free Grammar. The parser is trained, validated and tested using Penn Treebank dataset and results are measured in terms of F1, Precision and Recall metrics. The generative parser's implementation is constrained by memory usage and parsing speed of the model. Additionally, some heuristics are presented for improving the evaluation metrics and parsing time.**

I. INTRODUCTION

Part of Speech (POS) Tagging is a common technique used to build parse trees from the sentences to extract the syntactic relations between the words in the sentence. It has multiple applications in Machine Translation, Information Retrieval, etc. A POS Tag is a label assigned to all words in the sentence to indicate its Part of speech. The process of assigning the POS Tags to a sentence is often called annotation. This is a difficult task owing to the underlying ambiguities present in the English language. Multiple parse trees can be generated for the same sentence which may convey different meanings. For instance, the word 'saw' can be tagged as both a Verb (VB) and a noun (NN) in different contexts. This problem can be resolved by creating a probability distribution over the different parses and choosing the one with highest probabilities. Probabilistic Context Free Grammars (PCFGs) defines rules which helps in achieving the solution mentioned above. PCFGs extend the regular grammars by assigning a probability to each rule. The probability of the parse tree is then calculated as a product of the probabilities of the rules used to produce it. Formally, a PCFG can be defined by a quintuple

$$G = (M, T, R, S, P) \text{ where:}$$

M is the set of non-terminal symbols
T is the set of terminal symbols
R is the set of production rules (r_1, r_2,...)
S is the start symbol (S)
P is the set of probabilities on production rules

Based on PCFGs, the probability of the entire parse tree for a sentence is given as:

$$P(parsetree) = \prod_{i=1}^{n} P(r_i) \tag{1}$$

where n is the number of rules used in the creation of the parse tree. To create the grammar, the training data from Penn Treebank files 200 to 2199 is used. Once the grammar is constructed, a dynamic programming based CKY algorithm is applied to store the best parse tree details for each sentence in validation data which comes from Penn Treebank files 2200 to 2299. However, the standard CKY (Cocke–Kasami-Younger) algorithm cannot be applied directly on the grammar generated from the training trees, since it is devised to work only with CNF (Chomsky Normal Form) grammars. Thus, binarization of the training trees is performed to generate only binary and unary grammar rules, and the CKY algorithm is altered to accept unary rules along with binary rules. The implementation details of binarization and CKY algorithm are discussed in the next section.

II. IMPLEMENTATION DETAILS

Binarization: The right-handed binarization of a non-binary tree involves the following steps:

1) Start from the ROOT level, store the leftmost child node as is.
2) Replace the right child with a new node which will act as the parent of the rest of the children (apart from the leftmost child).
3) From this new right node, repeat the process till all the left and right children are binarized or are leaf nodes.

The standard lossless binarization however, keeps track of all the siblings it has while creating the right(next parent) node at every step. But, it doesn't carry any information about the parent node from which the left and/ or right children came from. This may end up giving high probability to parses which aren't good and do not convey the right meaning of the sentence. In order to avoid that, the following two techniques are applied to the trees coming from the training data:

1) **Parent Annotation** is performed to make the grammar more context aware. This is done by appending $parentlabel$ to every node's label using ^ as a separator between its own label and parent's label.
2) **Horizontal Markovization** is performed by relaxing the sibling independence assumption and storing some history of siblings in each node while performing splitting. This is done by appending $->siblinglabels$ to the right child's label using _ as a separator between all sibling labels. Based on experiments, it was identified that Right-branching gives better grammar rules to generate the tree compared to Left-branching for English grammar.

Several different degrees of Horizontal Markovization and Parent Annotation were experimented with as part of

[1] Email:kinjalj@andrew.cmu.edu

the *Binarize* class, the results for which are shared in Experiments & Analysis section later.

CKY Parsing The CKY Algorithm implements parsing in a bottom up manner and has the worst case running time of $O\left(n^3 \cdot |G|\right)$ where n is the length of the sentence to be parsed and $|G|$ is the number of grammar rules used to build it. The *GenerativeParser* class implements two functions namely *fillChart* and *buildTree* which are called by the *getBestParse* function to first create the score chart based on the grammar and their probability scores. Then, *buildTree* is invoked to recursively build the best parse tree based on the score chart and backpointers. To store the scores at every step a single three-dimensional array is used, which is filled for both unary and binary rules. For the backpointers, two different approaches were tried, one storing the array of *BackPointer* class which contains *unaryChild*, *binaryLeftChild*, *binaryRightChild* and *splitIndex*, and another one which stores two different arrays for unary and binary back pointers. The array of unary back pointer being of type *short* and the array of binary back pointer being of type *String* which holds _ separated values of *binaryLeftChild*, *binaryRightChild* and *splitIndex*. The latter resulted in significant gain in terms of parsing time as reported in the Experiments & Analysis section later.

III. Penn Treebank Statistics

The following table mentions the number of sentences belonging to various length ranges in test and validation data:

Type	Sentence Length <=15	Sentence Length <=40
Valid	421	1578
Test	603	2245

IV. Experiments & Analysis

Some optimizations were done in order to increase the accuracy of the parse (F1 score):

1) The unary rules are more exhaustive when taken from the *UnaryClosure* class instead of *Grammar* class. For instance, a rule of type A $->$ B then B $->$ C might be more probable than simply A $->$ C, but the *Grammar* class doesn't give the former ones. So, rules were fetched from *UnaryClosure* while filling the score chart, keeping in mind reflexives are not added.
2) Based on the paper *Klein and Manning 2003 "Accurate Unlexicalized Parsing"*, tagging % label as % and, adding a *-U* suffix to a *DT* or *RB* label if it is observed for a node with just one child while binarizing the tree resulted in a very slight increase in the overall F1 score, but it substantially increased the parsing time.

Some optimizations were done in order to reduce the parsing time of the generative parser while maintaining an above 79.8 F1 score:

1) Instead of binarizing for every non-terminal node, stopping the binarization when the number of children for that node are already 2 reduces the number of grammar rules thereby decreasing the time to fill the score chart.
2) The subset of non-terminals to loop over while filling the score chart rules were reduced by using *getClosedUnaryRulesByChild* and *getBinaryRulesByLeftChild* functions implemented by *UnaryClosure* and *Grammar* class respectively.

Submit.jar : Putting threshold to grammar : Validation Data (files 2200 to 2299)

Max Train Length	Max Test Length	v	h	P	R	F1	EX	Parse Time
15	15	1	Inf	82.38	76.68	79.43	32.77	8.182 secs
15	15	2	Inf	84.74	83.56	84.15	45.13	11.356 secs
15	15	1	1	80.84	75.08	77.86	28.02	2.409 secs
15	15	2	1	84.8	82.65	83.71	41.56	5.070 secs
15	15	1	2	83.1	76.74	79.79	34.67	5.690 secs
15	15	2	2	85.8	83.98	**84.88**	45.6	8.819 secs
40	40	1	0	74.44	68.41	71.3	8.55	1.662 mins
40	40	1	1	74.44	68.41	71.3	8.8	1.826 mins
40	40	1	2	75.82	70.23	72.92	10.64	6.013 mins
40	40	2	0	78.96	77.27	78.11	17.04	7.158 mins
40	40	2	1	78.96	77.27	78.11	17.04	6.893 mins
40	40	2	2	80.85	79.3	**80.07**	20.72	21.381 mins

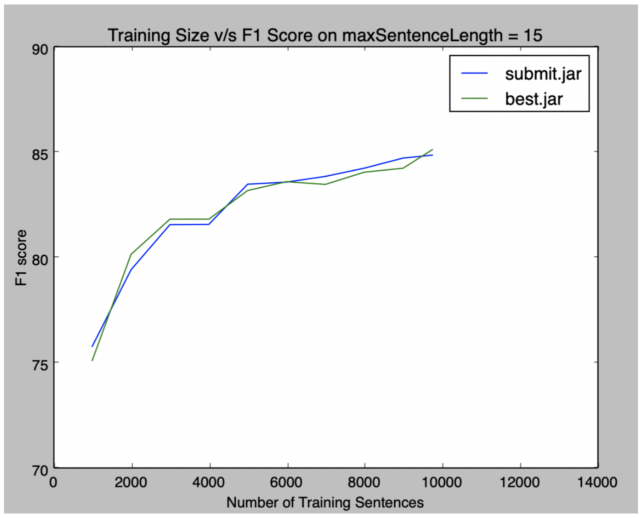
Best.jar : Putting threshold to grammar : Validation Data (files 2200 to 2299)

Max Train Length	Max Test Length	v	h	P	R	F1	EX	Parse Time
15	15	1	Inf	82.05	76.35	79.1	32.77	8.182 secs
15	15	2	Inf	85.06	84.01	84.53	45.36	10.999 secs
15	15	1	1	80.5	74.82	77.56	26.84	2.409 secs
15	15	2	1	84.61	82.89	83.74	41.56	4.461 secs
15	15	1	2	82.72	76.38	79.42	34.44	5.690 secs
15	15	2	2	86.05	84.24	**85.14**	45.84	7.982 secs
40	40	1	0	74.0	67.54	70.62	8.55	1.662 mins
40	40	1	1	74.0	67.54	70.62	8.55	1.672 mins
40	40	1	2	75.51	69.45	72.36	10.45	6.186 mins
40	40	2	0	78.93	77.35	78.13	17.04	6.693 mins
40	40	2	1	78.93	77.35	78.13	17.04	6.178 mins
40	40	2	2	80.53	79.12	**79.82**	20.59	18.082 mins

The comparative results from various experiments for different values of h(degree of horizontal markovization) and v(degree of vertical parent annotation) on validation data are listed above. The results were noted for different values of maximum length of sentences to be trained and tested as well and it was observed that $h = 2$, $v = 2$ gives the best results. Various strategies were tried for storing the score chart. First one being a *BackPointer class* to store the indexes of *unaryChild* ,*binaryLeftChild* ,*binaryRightChild* and *splitIndex*. Second time, two arrays, an array of unary back pointer being of type *short* and an array of binary back pointer being of type *String* which holds _ separated values of *binaryLeftChild*, *binaryRightChild* and *splitIndex* was tried, but it was too inefficient in terms of parsing time. Then, a hybrid system of storing unary back pointer separately into an array, and binary back pointer details in an Object class was ultimately used. This seemed like a good implementation for reducing the decoding time.

The above results are corresponding to the **submit.jar** which satisfy the minimum requirements of F1 and parsing time. The second table represents the results achieved on the validation data corresponding to the **best.jar** which reduces the number of grammar rules by putting a threshold to qualify so as to reduce the parsing time. Through some experiments it was identified that a rule score of -4.8 for unary and -4.5 for binary rules were good thresholds on this dataset which substantially reduced the parse time. Surprisingly, it didn't affect the F1 score as expected. Additionally, the annotation scheme used in **best.jar** doesn't markovize the nodes of the tree if its already binary ie. it has no more than two children.

General trends of F1 score v/s Max Sentence Length show that as Max Sentence Length for training data increases, F1 score on test and validation sets increase. This is expected as the probability distribution for the grammar rules adapts better with increased number of training examples the model witnesses.



V. Conclusions

As part of the assignment, CKY algorithm based generative parser were successfully implemented. The F1 score obtained for sentence length 15 with **best.jar** is 85.14 and it took 7.982 seconds to parse the sentences on average. The F1 score obtained for sentence length 15 with **submit.jar** is 84.88 and it took 8.819 seconds to parse the sentences on average. For the sanity check, both the models observed an F1 score of 87.5 and took about 11 milliseconds. As the size of training data increase, the performance of the model (F1 score) improves at the cost of parse time as shown in the plot below.

References

[1] Dan Klein and Christopher D. Manning. 2003. Accurate unlexicalized parsing. In Proceedings of the 41st Annual Meeting on Association for Computational Linguistics - Volume 1 (ACL '03), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 423-430. DOI: https://doi.org/10.3115/1075096.1075150

[2] Jurasfky Dan, James H Martin, "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition", Prentice Hall, 2008

[3] https://www.youtube.com/watch?v=CFEGKVjEH1Q

[4] https://en.wikipedia.org/wiki/Branching_(linguistics)

[5] http://christos-c.com/treeviewer/