Constituency Parsing with Probabilistic Context-Free Grammars

Raghu Chittersu
Department of Computation and Data Sciences
Indian Institute of Sciences, Bengaluru
chittersuv@iisc.ac.in

1 Introduction

Constituency parsing extracts constituency-based parse tree from a given sentence. It represents syntactic structure of the sentence.

Constituency parsing are widely used in solving problems like Parts of Speech tagging and text mining. Context Free grammar is essential for generation of parse trees. Since there are ambiguities one selects the tree with highest generation probability. CYK algorithm is a Dynamic Programming type algorithm which can be used to generate parse tree given Probabilistic Context Free Grammar

In this assignment constituency parsing is implemented using PCFG and CYKParser. Smoothing and prior probabilities are used to handle the sparsity of training data. Then the model's performance is various methods along with comparing with some of the best parsers.

The code is available at https://github.com/rv-chittersu/CYK-Parser.

2 Data

In this assignment, nltk's treebank corpus is used as data-set. It contains 3914 parsed and raw sentences in 200 files.

The production rules from parsed sentences of train set are used to build the PCFG. The raw sentences from test set are used to predict the parse tree and evaluation.

3 Methodology

The complete implementation of Constituency Parsing involves following components
(1) Probabilistic Context Free Grammar
(2) CYK Parser

3.1 Probabilistic Context Free Grammar

The PCFG is made of
(1) set of non-terminal symbols M,
(2) set of terminal symbols T,
(3) set of production rules R,
(4) start symbol S and
(5) probabilities for with production rules P

The CYK Parser requires the PCFG to be in Chomsky Normal Form i.e each production should satisfy following rules

$$X- > Y\ Z$$

$$X- > t$$

where X, Y, Z are Non Terminals and t is Terminal.

3.2 CYK Parser

The CYK Algorithm uses PCFG which is in Chomsky Normal Form to build a parse tree from given sentence The CYK Algorithm follows Dynamic Program Paradigm

The recurrence relation for probability of assigning a segment from i to j a non terminal X is given by

$$P_X(i,j) = \max_{i<k<j}(P_Y(i,k) * P_Z(k,j) * P(X- > YZ))$$

In bottom up approach or memoization method of dynamic program we start from sub-string of length 1 and keep adding rules satisfying above recurrence relation to the parse table till we reach string length. One should also keep track of the rules that generated maximum probability at each position through back pointers.

Once the parsing is done. The tree is generated recursively from Start Symbol S in final cell in the table through back pointers.

4 Handling Sparsity

Since the number words in vocabulary are limited. If new token show up while parsing a sentence the above algorithm will fail to parse as there are no production rule that lead to the token.

Method1
for a given OOV token t

$$P(X->t) = \frac{\#(X->Terminal)}{\#(Terminals)}$$

The another way of looking at above probability is probability of X generating being the lhs given the rhs of the production a non terminal.

We augment PCFG with above rule if OOV token t appears in the sentence.

Method2

$$P(X->t) = \frac{1}{\#X \ +1}$$

Along with above method add one smoothing is used to get the probability for unknown at each Terminal generating non terminal.

5 Experiment Setup

5.1 Pre Processing

The important preprocessing step is to convert the parse tree from treebank dataset to Chomsky Normal Form. It is handles using nltk module.

The other alterations done are converting strings to lower case and numbers to special NUM_TOKEN

5.2 Training

During training phase the parse tress from treebank training data set are processed to produce occurances of Production Rules. The produces output is saved to checkpoint file and will be used in further steps.

5.3 Testing

The testing is done by parsing the sentences in test set dervied from treebank datatset. The task is accomplished by multiple processes where each of them writes gold and generated parse tree to individual files. Then all the files by multiple processes are stitched to generate final gold and generated results

5.4 Evaluation

The performance of the model is computed based on comparing predicted parses and gold parses from nltk dataset. Metrics such as Bracketing Recall, Bracketing Precision and Bracketing FMeasure, Tagging accuracy is reported along with other metrics. The results are obtained using NYU's program EVALB (`https://nlp.cs.nyu.edu/evalb/`)

6 Results

The results of various metrics for above experiment with two different sparsity methods can be found below.

Table 1: Results on Test Data for **Method 1**

Metric	Result
Bracketing Recall	44.32
Bracketing Precision	42.04
Bracketing FMeasure	43.15
Complete match	1.91
Average crossing	11.66
No crossing	4.31
2 or less crossing	11.72
Tagging accuracy	84.30

Table 2: Results on Test Data for **Method 2**

Metric	Result
Bracketing Recall	44.69
Bracketing Precision	41.87
Bracketing FMeasure	43.32
Complete match	1.91
Average crossing	10.93
No crossing	3.18
2 or less crossing	11.46
Tagging accuracy	82.70

It can be seen that the results of both methods are almost similar. The Bracketing FMeasure is higher for the second model while the Tagging Accuracy is higher for the first model.

The higher tagging accuracy is expected for **Method1** as we have used likelihood of tags given terminals to initialize production rules.

Table 3: Comparison with Stanford parser **Case - 1**

Sentence	Rose is red .
Gold Parse	(ROOT (S (S (VP (VBD rose))) (VP (VBZ is) (ADJP (JJ red))) (. .)))
Predicted	(S<VP-.>(VP (VBD Rose) (VP (VBZ is) (ADJP (JJ red)))) (. .))
Fix	The mismatch is due to the absence of following production rules $$S+VP->VBD$$ $$S->S+VP\ S\vert<VP-.>$$ adding above rules to grammar with high probability fixed the error.

Table 4: Comparison with Stanford parser **Case - 2**

Sentence	He knows.
Gold Parse	(ROOT (ROOT (S (NP (PRP He)) (VP (VBZ knows)) (. .)))
Predicted	(S (NP (PRP He)) (S <VP-.>(VP (VBZ knows)) (. .)))
Fix	The mismatch is due to the relative probabilities of following production rules $$S->NP\ \ VP$$ $$S->NP\ \ S\vert<VP-.>$$ updating the rule with high probability fixed the error.

Table 5: Comparison with Stanford parser **Case - 3**

Sentence	It is a far, far better thing that I do, than I have ever done; it is a far, far better rest I go to than I have ever known.
Gold Parse	(S (S (NP (PRP It)) (VP (VBZ is) (NP (DT a) (ADJP (JJ far)) (, ,) (ADJP (RB far) (JJR better)) (NN thing)) (SBAR (IN that) (S (NP (PRP I)) (VP (VBP do)))) (, ,) (SBAR (IN than) (S (NP (PRP I)) (VP (VBP have) (ADVP (RB ever)) (VP (VBN done)))))) (: ;) (S (NP (PRP it)) (VP (VBZ is) (NP (NP (DT a) (JJ far) (, ,) (ADJP (RB far) (JJR better)) (NN rest)) (SBAR (S (NP (PRP I)) (VP (VBP go) (PP (TO to) (SBAR (IN than) (S (NP (PRP I)) (VP (VBP have) (ADVP (RB ever)) (VP (VBN known)))))))))))) (. .))
Predicted	(S (S (NP (PRP It)) (VP (VBZ is) (NP (DT a) (NP-NN-SBAR>(ADJP (ADJP (JJ far)) (ADJP<,-ADJP>(, ,) (ADJP (RB far) (JJR better)))) (NP-SBAR>(NN thing) (SBAR (SBAR (WHNP (WDT that)) (S (NP (PRP I)) (VP (VB do)))) (SBAR<,-SBAR>(, ,) (SBAR (IN than) (S (NP (PRP I)) (VP (VBP have) (ADJP (RB ever) (VBN done))))))))))) (S<:-S-.>(: ;) (S-.>(S (NP (PRP it)) (VP (VBZ is) (NP (NP (NP (DT a) (RB far)) (NP<,-ADJP>(, ,) (ADJP (RB far) (JJR better)))) (NP-S>(NN rest) (S (NP (PRP I)) (VP (VB go) (PP (TO to) (SBAR (IN than) (S (NP (PRP I)) (VP (VBP have) (ADJP (RB ever) (VBN known))))))))))) (. .))))
Fix	In above complex sentence all POS tags are correct except for word *that*. The model predicted IN while Stanford parser predicted it as **WDT**. Just increasing the count for production $WDT->that$ fixed the error

Table 6: Comparison with Stanford parser **Case - 4**

Sentence	Real courage is when you know youre licked before you begin, but you begin anyway and see it through no matter what.
Gold Parse	(S (S (NP (JJ Real) (NN courage)) (VP (VBZ is) (SBAR (WHADVP (WRB when)) (S (NP (PRP you)) (VP (VBP know) (SBAR (S (NP (PRP you)) (VP (VBP ’re) (VP (VBN licked) (SBAR (IN before) (S (NP (PRP you)) (VP (VBP begin)))))))))))) (, ,) (CC but) (S (NP (PRP you)) (VP (VP (VB begin) (ADVP (RB anyway))) (CC and) (VP (VB see) (NP (PRP it)) (PP (IN through) (NP (DT no) (NN matter)))) (NP (WP what)))) (. .))
Predicted	(S (S (NP (JJ Real)) (VP (MD courage) (VP (VBZ is) (SBAR (WHADVP (WRB when)) (S (NP (PRP you)) (VP (VB know) (S (NP (PRP you)) (VP (VBP ’re) (VP-SBAR>(PRT (RP licked)) (SBAR (IN before) (S (NP (PRP you)) (VP (VB begin))))))))))))) (S<,-CC-S-.>(, ,) (S-S-.>(CC but) (S-.>(S (NP (PRP you)) (VP (VP (VB begin) (ADVP (RB anyway))) (VP-VP>(CC and) (VP (VB see) (VP-SBAR>(NP (PRP it)) (SBAR (IN through) (S (NP (DT no)) (VP (VB matter) (NP (WDT what)))))))))) (. .)))))
Fix	In above sentence there are three POS Tagging mistakes among them two are OOV words

7 Conclusion

In this assignment a Probabilistic Context Free Grammar is built from training set. Then CYK algorithm is used along with two different type of smoothing to generate Constituency Parse of given sentence. Then the results produces is compared with the Stanford Parser.