Elements of Language Processing and Learning
Probabilistic Context-Free Parsing

Authors:

Agnes van Belle *(10363130)*,
Norbert Heijne *(10357769)*

December 15, 2012

1 Introduction

This report is for the course Elements of Language Processing and Learning. The goal of the assignment is to use a Probabilistic Context Free Grammar (PCFG) together with the CYK algorithm outfitted for probabilities to train a system on Context Free Grammar (CFG) trees that can then generate new trees from sentences. First, the training trees need to be parsed and transformed into a PCFG. Secondly, The generated PCFG is used to generate trees for the given test sentences and lastly the most likely tree is compared to the test tree for all sentences.

The goal is to reconstruct the parses of training examples and to assign good parses to new unseen examples, building a robust grammar that captures our training examples and uses probability to deal with ambiguity in choosing from a set of possible parses [2].

2 PCFGs - Probabilistic Context Free Grammars

The amount of trees that can be generated from a sentence through bottom up parsing is very large and grows exponentially with the sentence length. Therefore it is required that the solution space is narrowed. The PCFG assigns probabilities to each side of the rules in the grammar which gives us a way to construct the trees under the PCFG and capture how likely each tree is [2].

The treebank that is used to train the system has been provided to us for this course. the treebank consists of binarized trees that are in Chomsky Normal Form. Though these trees still contain unary rules at the lowest branches of the trees, therefore they are handled as a special case in the CYK algorithm.

Formal definition of the PCFG [2]:

1. A probabilistic Context Free Grammar (PCFG) is a five tuple $< W, N, N_1, R, P >$
2. W : Set of terminal symbols (i.e. words)
3. N : Set of non-terminal symbol $N_1, \ldots, N_n$ (i.e. labels)
4. $N_1 \in N$: Distinguished starting symbol
5. R : Set of rules, each has form $N_i \rightarrow C_j$, with C_j a string of terminals and non-terminals. Each rule has probability $P(N_i \rightarrow C_j)$
6. P : a (probability) function assigning probabilities in range $[0, 1]$ to all rules such that $\forall X \in N \left[\sum_{\beta \in V^*} P(X \rightarrow \beta)\right]$

3 CYK - CockeYoungerKasami Algorithm

The CYK algorithm is a bottom up chart parsing algorithm, the simplest form PCFG is required to be in Chomsky Normal Form (CNF). Since our treebank still contains unary rules at the lowest branches and at one unary rule at the top ($\texttt{TOP} \rightarrow S$), these are handled by the algorithm as a special case.

"The method works as follows. Let $G = (N, \Sigma, P, S)$ be a Chomsky normal form CFG with no e-production. A simple generalization works for non-CNF grammars as well.

Let $w = a_1 a_2 \cdots a_n$ be the input string which is to be parsed according to G. The essence of the algorithm is the construction of a triangular parse table. If we want one (or all) parses of w, we can use the parse table to construct these parses. [1]"

4 Viterbi Algorithm

The Viterbi algorithm in the case of the CYK chart is storing back pointers to the right hand sides that are produced by the corresponding left hand side with the highest probability. In our case that means attaching a reference to the position(s) of the corresponding right hand side(s) in the CYK chart for every left hand side in each cell. Starting with the non-terminal `TOP`, the right hand side(s) are appended to `TOP` as children. The next left hand side(s) are looked up in the referred position(s), the corresponding right hand side(s) are then appended as children. The right hand side(s) are then looked up again in their referred position(s) and used as the next left hand side(s). This repeats itself until all right hand sides that are found through the back pointers are terminals.

A more detailed description of how the tree is built with the back pointers in the form of pseudocode can be found in the appendix under algorithm 1.

Algorithm 1 buildTree

```
Require: tree $t$, table containing objects
Ensure: each object contains:
left hand side: $lhs1$
right hand sides: $r1$, r2
locations of right hand sides: $l1, l2$

current object $c \leftarrow$ TOP
next location $nl \leftarrow l1$
insert $lhs1$ of $c$ into $t$ as root
position in the tree $p \leftarrow$ root
RECURSION$(t, p, nl, r1)$

procedure RECURSION$(t, p, l, lhs)$
    $c \leftarrow$ object where $lhs = lhs1$ in $l$
    add child node with $r1$ to $p$ in tree $t$
    next position $np \leftarrow$ child node
    RECURSION$(t, np, l1, r1)$
    if $r2$ is not empty then
        add child node with $r2$ to $p$ in tree $t$
        next position $np \leftarrow$ child node
        RECURSION$(t, np, l2, r2)$
    end if
end procedure
```

5 Smoothing

In our algorithm we apply a method of smoothing that uses the conditional probabilities of certain word characteristics being present, this data is obtained by counting the frequencies in the training data.

We also use a a couple of smoothing methods regardless of whether or not the above mentioned smoothing method is used. The first smoothing method is to check if the word begins with a number. If it does begin with a number it is immediately classified as such. The second smoothing method is used when the sentence cannot be derived properly, the derivation tree will be made flat where all terms go to the non-term `POS`, and those non-terms are the children of `TOP`. This is done to provide EvalC with a sentence to compare even if there is no possible derivation with our algorithm.

The smoothing algorithm classifies an unknown term to a non-term using the following characteristics: Capitalization, suffixes, and hyphen presence. Capitalization contains three categories. A word begins with a capital letter and is not the first term of a sentence. A term contains only capital letters. And the term contains no capital letters or is not a word. The suffixes used are "ing", "ly", "s", and "ed". Any word that does not end with any of these suffixes falls into a different category. And whether or not the word contains a hyphen or not. The conditional log probability $\texttt{log}(P(\text{non-term}|c, s, h))$ where c = capitalization category, s = suffix category, and h = hyphen category, is calculated using the frequency of the characteristics in the training data.

6 Results and Analysis

For our results we compare the effects of smoothing to not smoothing on the algorithm. The test set we use was provided to us by the course. The test set contains 2416 sentences. The sentences that were longer than 16 terms were cut out because the algorithm could not handle longer sentences well enough to compute them within a reasonable time frame. Which leaves us with 684 sentences.

EvalC results		
	not smoothed	smoothed
Number of sentences	684	684
Bracketing Recall	59.79%	74.43%
Bracketing Precision	75.37%	75.51%
Bracketing FMeasure	66.68%	74.97%
Complete match	16.52%	21.20%
Average crossing	0.66%	0.83%
No crossing	73.83%	66.81%
2 or less crossing	90.50%	88.01%
Tagging accuracy	71.68%	92.48%

Table 1: The results of EvalC on the test sentences using our algorithm with and without smoothing

Smoothing had a large effect on the tagging accuracy, which would make it likely that the correct tagging would result in better bracketing in the CYK algorithm explaining the increase in recall.

7 Conclusion

References

[1] Aho and Ullmann. *The theory of parsing, translation, and compiling.* Prentice-Hall, Inc., 1972.

[2] Gideon Maillette de Buy Wenniger, 2012. Slides explanation of PCFGs and CYK used in Lab.