Elements of Language Processing and Learning
Probabilistic Context-Free Parsing

Authors:

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

December 11, 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 [de Buy Wenniger(2012a)].

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 [de Buy Wenniger(2012a)].

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 [de Buy Wenniger(2012a)]:

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 ($TOP \rightarrow S$), 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. [Aho and Ullmann(1972)]"

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 Results and Analysis

6 Conclusion

References

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

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

[de Buy Wenniger(2012b)] Gideon Maillette de Buy Wenniger, 2012b. Slides explanation of Step 3: Computing most likely tree.

[Manning(2007)] Christopher Manning. `http://nlp.stanford.edu/courses/lsa354/SLoSP-2007-` 2007. Slides explaining the CYK algorithm.