Elements of Language Processing and Learning
Probabilistic Context-Free Parsing

Authors:

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

November 26, 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 ($TOP \rightarrow S$), 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 Implementation

We first had the impression that we had to produce all possible derivations for a sentence. That is, we kept track of all left hand sides of a rule $A \rightarrow B, C$ in a cell in the CYK derivation table, even if a left hand side with a lower probability (but with a different right hand side) $A \rightarrow D, E$ was already added in that cell. However, using this implementation the heap needed more than 3 GB of space for a sentence containing just 7 terms.

Regardless if this was the aim of the second part of the assignment or not, it was infeasible. Therefore, in our current implementation, we only keep track of the left hand side of a rule $A \rightarrow B, C$ in a cell if its probability is higher than a rule $A \rightarrow D, E$ already there. This means we can not get equal trees (but with different probability) per level of the tree. However, because we still had to output all productions $TOP \rightarrow XP$, we made an exception for the TOP node. We do keep adding TOP nodes even though the might already exist more than one.

To distinguish terminals from non-terminals, we prefixed each non-terminal with "nt_". The sets of terminals and non-terminals overlap, and this can cause problems when applying the recursive parts in the CYK algorithm.

We did not parse sentences that did have more than 15 terms. This was recommended, and the parsing procedure indeed consumed quite some time for sentences having more than 15 terms. In our file containing the productions $TOP \rightarrow X$, these sentences are given the productions $TOP \rightarrow$ (and empty right hand side), because we did not implement these sentences having POS tags yet.

Our algorithm is almost exactly implemented as in [4], except that we do not go through all possible rules $A \rightarrow B$ (base case) or $A \rightarrow B, C$ (recursive case), but look up the left hand side A given B or B, C, respectively.

5 Results and Analysis

So far what we can see from the output for the test sentences are two things.

First, it is noticeable that we didn't implement any smoothing method yet. Test sentences with terms that never occurred before, such as "over-the-counter" or a abbreviation as "CNN", have no rules $TOP \rightarrow XP$. These kind of terms simply have no left hand sides, and thus the derivation stops with them. We could also insert feature-based rules in the CFG.

Second, we intend to use log-probabilities from now on, since the final probabilities become very small easily. For example, a probability of $1.23372e-143$ is not uncommon. While so far, this seems to be handled fine, we can't be too sure about that. Furthermore, we simply waste space using these normal probabilities.

6 Conclusion

We still need to implement the Viterbi algorithm and evaluate the results given that implementation using *evalC* or *evalB*, so we can not draw definite conclusions yet. Or plans thus far are in Section 5.

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.

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

[4] Christopher Manning. `http://nlp.stanford.edu/courses/lsa354/SLoSP-2007-2.pdf`, 2007. Slides explaining the CYK algorithm.