

Automatic PCFG Generator

Todd Layton, Anastasia Uryasheva

December 13, 2012

1 INTRODUCTION

Here will be an introduction

2 BACKGROUND

Here will be an article description

3 IMPLEMENTATION

The `pcfggen` module's primary access point is the `generate_pcfg()` function, which uses a POS-tagged training corpus to generate a PCFG through an iterative process of production rule creation and modification by transforming an initial trivial grammar into a more generalized one.

Algorithm 1 Pseudocode program flow of `pcfggen` module's `generate_pcfg()` function

```
function GENERATE-PCFG( $S$ )           ▷  $S$ : training corpus of POS-tagged sentences
   $G$  = Initial-Grammar( $S$ )
  loop
    while  $G$  has valid rule joining  $J$  do
       $G$  = Join-Rules( $G$ ,  $J$ )
    if  $G$  has valid most-frequent 2-gram expansion  $E$  then
       $G$  = Expand-2-Gram( $G$ ,  $E$ )
    else
      return  $G$ 
```

3.1 SIMPLIFYING ASSUMPTIONS

This PCFG generator program make a couple basic assumptions to reduce the complexity of the problem, allowing it to focus more effectively on the core of the automated generation concept.

3.1.1 PRE-TAGGED TRAINING CORPUS

Like many statistical parsers, the PCFGs generated by this program only describe production rules down to the level of parts of speech; the grammars' terminal symbols are not specific words, but rather POS tags. This means that the important training input to the generator is not a set of sentences in themselves, but a set of POS-tag sequences corresponding to a set of sentences. While this could be achieved by incorporating a separate tagging preparation step into the program itself, we chose to instead reduce the scope of the generator to exclude tagging; instead, the program's training corpus input is a set of POS-tagged sentences, which can have been tagged in any way the user sees fit.

3.1.2 FLAT-PROBABILITY TRAINING CORPUS

The program calculates the relative frequency of various features in the grammar, as it is generated, on the assumption that each of the tagged sentences in the training corpus has the same flat probability in the source language (namely, a probability of 1 over the size-in-sentences of the corpus). While this reduces the amount of input data necessary, it also removed the possibility of the generation process taking into account the true relative probabilities of the training corpus's content, which could have a non-negligible impact on the resulting PCFG when training on larger corpora.

3.2 INITIAL GRAMMAR

The program uses the input training corpus to create a simple grammar as a starting point for the rule modification. For each POS-tagged sentence in the training corpus, a corresponding production rule is added to the grammar. The left-hand side of this rule is `ROOT`, the right-hand side is the sequence of POS tags constituting that sentence, and the probability is 1 over the total size, in sentences, of the corpus.

3.3 ITERATION

Once the initial grammar is created, the program repeatedly looks for modifications to make in order to produce a grammar which is not so closely fitted to the training corpus. Generation of the PCFG is complete once there are no remaining modifications to be made. These modifications consist of two types of grammar transformations: 2-gram expansion and rule joining.

3.3.1 2-GRAM EXPANSION

The program finds the most frequently-occurring symbol 2-gram in the training corpus which is not already the right-hand side of a production rule in the grammar. A new nonterminal symbol is created, specified as an expansion of that 2-gram's symbols and with a unique numeric identifier, and each instance of that 2-gram is replaced with that nonterminal. An additional production rule is added to the grammar, of which the left-hand side is the new nonterminal, the right-hand side is the 2-gram, and the probability is 1. This step does not change the functional structure of the grammar, but rather decomposed it into a larger number of simpler rules in order to enable rule joining.

3.3.2 RULE JOINING

Before each 2-gram expansion step, the program checks whether any multiple production rules in the corpus, with more than one symbol in the right-hand side, consists of the exact same right-hand side except for the symbols at one particular index (for example, the third symbol in each rule's right-hand side). If such rules are found, then a new nonterminal symbol is created, specified as a rule joining of the symbols being replaced and with a unique numeric identifier, and the symbol at that index in each rule's right-hand side is replaced with that nonterminal. Then, for each of those rules, a new rule is added to the grammar, of which the left-hand side is the new nonterminal, the right-hand side is the symbol which that nonterminal replaced in that rule, and the probability is the relative frequency of the right-hand side symbol. Unlike 2-gram expansion, rule joining actually changes the functional structure of the grammar, allowing for symbol sequence which did not previously fit.

3.4 EXAMPLE

The effects of these two types of grammar modification are best illustrated with a simple example. Consider a toy corpus of three sentences, corresponding to three POS tag sequences:

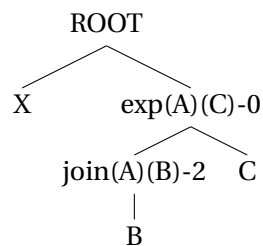
- (a) A C B C
- (b) X A C
- (c) Y B C

Given this corpus as the training input, the program would perform the following steps:

1. Create the initial grammar:
 - (a) $\text{ROOT} \rightarrow \text{A C B C} \ [1/3]$
 - (b) $\text{ROOT} \rightarrow \text{X A C} \ [1/3]$
 - (c) $\text{ROOT} \rightarrow \text{B C Y} \ [1/3]$
2. Check for rule joining, but no production rules are sufficiently similar.
3. Expand the 2-gram A C:

- (a) $\text{ROOT} \rightarrow \text{exp}(\text{A})(\text{C})-0 \text{ B C } [1/3]$
 - (b) $\text{ROOT} \rightarrow \text{X exp}(\text{A})(\text{C})-0 [1/3]$
 - (c) $\text{ROOT} \rightarrow \text{B C Y } [1/3]$
 - (d) $\text{exp}(\text{A})(\text{C})-0 \rightarrow \text{A C } [1]$
4. Check for rule joining, but no production rules are sufficiently similar.
5. Expand the 2-gram B C:
- (a) $\text{ROOT} \rightarrow \text{exp}(\text{A})(\text{C})-0 \text{ exp}(\text{B})(\text{C})-1 [1/3]$
 - (b) $\text{ROOT} \rightarrow \text{X exp}(\text{A})(\text{C})-0 [1/3]$
 - (c) $\text{ROOT} \rightarrow \text{exp}(\text{B})(\text{C})-0 \text{ Y } [1/3]$
 - (d) $\text{exp}(\text{A})(\text{C}) \rightarrow \text{A C } [1]$
 - (e) $\text{exp}(\text{B})(\text{C}) \rightarrow \text{B C } [1]$
6. Join A C and B C:
- (a) $\text{ROOT} \rightarrow \text{exp}(\text{A})(\text{C})-0 \text{ exp}(\text{B})(\text{C})-1 [1/3]$
 - (b) $\text{ROOT} \rightarrow \text{X exp}(\text{A})(\text{C})-0 [1/3]$
 - (c) $\text{ROOT} \rightarrow \text{exp}(\text{B})(\text{C})-1 \text{ Y } [1/3]$
 - (d) $\text{exp}(\text{A})(\text{C})-0 \rightarrow \text{join}(\text{A})(\text{B})-2 \text{ C } [1]$
 - (e) $\text{exp}(\text{B})(\text{C})-1 \rightarrow \text{join}(\text{A})(\text{B})-2 \text{ C } [1]$
 - (f) $\text{join}(\text{A})(\text{B})-2 \rightarrow \text{A } [1/2]$
 - (g) $\text{join}(\text{A})(\text{B})-2 \rightarrow \text{B } [1/2]$
7. Check for rule joining, but no production rules are sufficiently similar.
8. Check for 2-gram expansion, but no unexpanded 2-grams remain.
9. Return the grammar.

The resulting generated PCFG is able to parse not only the training sentences, but also other sentences of a similar structure. For example, the sentence X B C would be parsed as follows:



3.5 POTENTIAL IMPROVEMENTS

This implementation of a PCFG generation program could be improved significantly in several ways. At the moment, all valid 2-gram expansions and rule joinings which exist in the training corpus are executed, and the generation process only terminates once all such modifications have been completed. Allowing for the possibility of valid but statistically-negligible modification would improve the system's behavior on large training sets. Similarly, it is very possible for a generated grammar to include groups of production rules which could be simplified down to an equivalent form with fewer rules (as can be seen in the above example). This would limit the growth of the overall size of the generated PCFG's rule set, and might even create additional valid rule joinings of which the generation program could make use.

Another major improvement would be the incorporation of word and sentence probabilities into the training data. If the generation program could be run on raw sentences rather than on POS-tag sequences, and could account for the relative probabilities of each POS mapping to each word, then the generated grammar would be able to parse raw sentences directly, which would improve the accuracy of its results and massively increase its ability to generate natural-sounding random sentences. Similarly, incorporating information about the relative frequency of the training corpus's sentences would prevent less-grammatical input data from distorting the overall structure of the generated PCFG.

4 RESULTS

Here will be a table with results and results' description

5 CONCLUSION

Here will be a conclusion