Elements of Language Processing and Learning

Probabilistic Context-Free Parsing

Authors:

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

December 19, 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, punctuation)
3. N : Set of non-terminal symbols $N_1, \dots, N_n$ (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 terminal or two 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 algorithm requires a (P)CFG to be in Chomsky Normal Form (CNF) for it to work properly. However, since our treebank still contains unary rules at the lowest branches and at one unary rule at the top ($\texttt{TOP} \rightarrow N_i$), these are handled by the algorithm as a special case [3]. The pseudo code of the CYK algorithm with the Viterbi addition is shown in Algorithms 2 and 3 in Appendix A, the algorithm is based on the pseudo code provided in by [5].

The CYK method can be summarized as follows. Let $G = (N, \Sigma, P, S)$, $\epsilon \notin G$ (no empty string) be a Chomsky normal form CFG . 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. It can be used to find out if $w \in G$ by consulting the last filled cell of the table. It can also be used to generate all parses of w [1].

We use a Probalistic CFG (PCFG) instead of a CFG, which means that we can also use the CYK table to generate the *most likely* parse. The Viterbi algorithm is suitable to do so.

4 Viterbi Algorithm

t The Viterbi algorithm is a general dynamic programming algorithm for finding the most likely sequence of successive states for a Hidden Markov model (HMM).

The CYK algorithm applied on a PCFG can be viewed as a HMM where the states are the (partial) derivations of w, up to a final state (the sentence w); and the transition probabilities are defined by the rule probabilities $P(N_i \to C_j)$.

To derive the most likely state sequence (i.e. the most likely derivation tree), we only need to edit the bottom-up parsing process of the CYK algorithm. The crux is storing back pointers along with each state that denote which states it could cause. In our case that means attaching a pointer to the position of the right hand side C_j in the CYK chart for every left hand side N_i that is deemed by the PCFG to be able to produce C_j.

The resulting most probable derivation from the CYK algorithm is produced as follows. We start with the non-terminal `TOP`, its right hand side(s) are appended to `TOP` as children. The next left hand side(s) are looked up in the referenced 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 referenced position(s) and used as the next left hand side(s). This process repeats 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 Algorithm 1.

Algorithm 1 buildTree

```
Require: tree $t$, table containing objects
Ensure: each object contains:
    left hand side non-terminal: $lhs1$
    right hand sides or back pointers: $r1$, $r2$
    locations of right hand sides in the CYK chart: $l1, l2$
 1: current object $c \leftarrow$ TOP
 2: next location $nl \leftarrow l1$
 3: insert $lhs1$ of $c$ into $t$ as root
 4: position in the tree $p \leftarrow$ root
 5: RECURSION$(t, p, nl, r1)$
 6: procedure RECURSION$(t, p, l, lhs)$
 7:     $c \leftarrow$ object where $lhs = lhs1$ in $l$
 8:     add child node with $r1$ to $p$ in tree $t$
 9:     next position $np \leftarrow$ child node
10:     RECURSION$(t, np, l1, r1)$
11:     if $r2$ is not empty then
12:         add child node with $r2$ to $p$ in tree $t$
13:         next position $np \leftarrow$ child node
14:         RECURSION$(t, np, l2, r2)$
15:     end if
16: end procedure
```

5 Implementation choices

5.1 PCFG

Our PCFG is extracted from the provided training data, which contains 39832 sentences in Penn WSJ format. The probabilities $P(N_i \to C_j)$ in our PCFG are estimated using Maximum Likelihood Estimation.

5.2 Smoothing

In our algorithm we apply a method of smoothing that uses the conditional probabilities of word features being present. This smoothing method is inspired by [4], but mathematically simplified.

Feature		
Capitalization	Suffix	Hyphen
starts with capital	*-ing*	*hyphen present*
all capitals	*-ly*	*no hyphen present*
no capitals	*-es* or *-s*	
	-ed	
	no suffix	

Table 1: Features that can be combined in our conditional-based smoothing method

In a very simple smoothing case, features and left-hand sides could all be assumed to be independent of each other, and the probability of an unknown word a_u being caused by non-terminal

W_i is then computed similar to $P(W_i|a_u) = \frac{1}{Z}P(\text{unknown word}|W_i) \cdot P(\text{capitalization}(a_u)) \cdot P(\text{hyphen}(a_u))$. That assumes independence between the features themselves, and between the features and the left-hand side W_i. This independence assumption is unlikely. For example, we would intuitively say that a capitalized word is more likely to be an `NNP` than a `VP`. But certain suffixes, for example "-ly", seem also more likely to contain hyphens. Therefore, we look at the conditional probability

$$P(a_u|W_i) = \frac{1}{Z} \cdot P(W_i|\text{capitalization}(a_u), \\ \text{suffix}(a_u), \\ \text{hyphen}(a_u))$$

This data is obtained by counting the frequencies of all these combinations of the features and the non-terminals in the training data. We use add-one smoothing for these frequencies, so that each non-terminal has at least a minimal chance to cause the unknown term. We only take into account non-terminals that have caused a terminal in the training data, because we assume only such non-terminals can cause unknown terminals. Other non-terminals, such as `S`, are thus not in the smoothing table. Using the features in Table 1, we get a table of size [$3 \times 5 \times 2 \times$ *# of non-terminals having caused terminals*].

As such, we do not take the prior probability of $P(\text{unknown word}|W_i)$ into account. We use a Maximum Likelihood Estimation method with add-one smoothing. To see this, consider that if none of the features is present, the unknown word has most chance to be generated by the non-terminal that occurred most times in the training data when producing a terminal with none of the features.

We also always use another, more specific, smoothing-like method regardless of whether or not the above mentioned smoothing method is used. We always check if a terminal is a number. Here, a number is (very simply) defined as a string containing just digits and/or the symbols "." and ",". If so, we simply represent it in the PCFG as the symbol "<[(number)]>" and also look it up as such.

5.3 Log-probabilities

By default the CYK algorithm is not defined as using log-probabilities. We have converted the probabilities in the PCFG and the probabilistic computations in the CYK algorithm to work with log-probabilities to avoid so-called arithmetic underflows in the CYK algorithm. We have done this because the probabilities of the derivations can easily become extremely small. Using log-probabilities indeed turned out to make a difference, but we have not tested this difference.

6 Results and Analysis

For our results we compare the effects of smoothing to not smoothing on the algorithm.

When smoothing is not used, some sentences can not be derived. In this case, the derivation tree will be made flat where all terminals are assigned a left-hand side of `POS`, and those non-terminals are all the children of `TOP`. We will call this a "fake" derivation, and say the sentence is not being derived "properly". This is done to provide EvalC with a sentence to compare even if there is no possible derivation with our algorithm.

The comparison is done with the program EvalC [6]; the EvalC tool is used for evaluating constituency parsing.

6.1 Provided train data

The test and train sets we uses were provided to us by the course. The train set contained which contained 39832 sentences, and the test set a total of 2416 sentences. The sentences that were longer than 16 terms were excluded from parsing because the algorithm has trouble deriving longer sentences within a reasonable time frame. Which left us with 684 sentences. The results are in Table 2.
All measurements have better scores when using smoothing compared to when using no smoothing, as one would expect.

The F-Measure is increased with 8.29 %. But smoothing had an especially large effect on the tagging accuracy (20.8 %) and bracketing recall (14.64 %). It is likely that the correct tagging

EvalC results		
	not smoothed	smoothed
Percentage of sentences derived properly	79.9%	100%
Bracketing Recall	59.79%	74.43%
Bracketing Precision	75.37%	75.51%
Bracketing F-Measure	66.68%	74.97%
Complete match	16.52%	21.20%
Tagging accuracy	71.68%	92.48%

Table 2: The results of EvalC on the test sentences using our algorithm with and without smoothing. A total of 684 sentences were parsed.

would result in better bracketing in the CYK algorithm, explaining the increase in recall.

6.2 Reduced train data

To further examine the effect of smoothing and our use of "fake" derivations, we have also trained the PCFG on an extremely reduced data set. This reduced data set contains just seven randomly chosen sentences from the original treebank data set. The reduced data set can be found in Appendix B. The we tested our implementation using the provided test set, the same test set that we used in Section 6.1. The results are in Table 3.

EvalC results		
	not smoothed	smoothed
Percentage of sentences derived properly	0%	100%
Bracketing Recall	0%	24.77%
Bracketing Precision	NaN	36.92%
Bracketing FMeasure	NaN	29.65%
Complete match	0%	1.9%
Tagging accuracy	0.56%	27.78%

Table 3: The results of EvalC on the test sentences using our algorithm with and without smoothing while using an extremely reduced train data set of 7 sentences. A total of 684 sentences were parsed.

In this extreme case, the F-measure is almost 30 % higher when using smoothing compared to no smoothing at all.

From these poor scores for the no smoothing case in which all derivation results were "fake" derivations; we can assume that our use of "fake" derivations when no derivation could be found could not have influenced the results in Section 6.1 to the benefit of the no smoothing case.

7 Conclusion and future research

We have shown that the CYK algorithm in combination with the Viterbi algorithm can be a powerful tool for producing the most likely derivations of sentences when trained with a PCFG.

While the time complexity of CYK is just $O(n^3)$, an obvious problem on CYK is its struggle with longer sentences due to a growing table size. This is what we found out in the previous assignment when we tried to produce *all* derivations. Of course, when applying the Viterbi algorithm on it this problem of heap size is significantly reduced. However, the program still has hard times processing sentences longer than approximately 16 terms, which will increase when the PCFG used is larger.

Our smoothing method provided to be useful. Tagging accuracy and bracketing recall were increased with 20.8 % and 14.64 % with respect to the test data, respectively. Smoothing provided to be especially useful when training data was sparse. Of course, in reality, training data is abundant. However, the training data available may poorly reflect the new data to be parsed, because of (social, professional) domain differences in language use.

In this conditional smoothing method we used we still used quite few features. We could easily extend the model with more suffixes and more features such as prefixes; word length; inflection use; etc. This is likely to improve performance, as [4] found. Furthermore, a comparison with other smoothing methods could be done.

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] Alexander Franz. Independence assumptions considered harmful. In *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics*, EACL '97, pages 182–189, Stroudsburg, PA, USA, 1997. Association for Computational Linguistics.

[5] Christopher Manning, 2007. Slides explaining the CYK algorithm.

[6] Federico Sangati. `http://homepages.inf.ed.ac.uk/fsangati/`, 2010. EvalC - Graphical tool for constituency parsing evaluation.

Appendices

A Pseudo code

Algorithm 2 CYK part 1 : base case

```
Require: numWords, numNonterminals, prob[numWords, numWords, numNonterminals],
    back[numWords, numWords, numNonterminals], grammar, nonterms
 1: procedure baseCYK
 2:     for $i \leftarrow 1$, numWords do
 3:         for $A \leftarrow 1$, numNonterminals do
 4:             if $A \rightarrow w_i$ is in grammar then
 5:                 prob [i, i, A] = P($A \rightarrow w_i$)
 6:             end if
 7:             added $\leftarrow$ true
 8:             while added do
 9:                 added $\leftarrow$ false
10:                 for $A, B$ in nonterms do
11:                     if prob $[i][i+1][B] > 0$ and $A \rightarrow B$ in grammar then
12:                         tempProb = P($A \rightarrow B$)*prob$[i][i+1][B]$
13:                         if tempProb > prob$[i][i+1][A]$ then
14:                             prob$[i][i+1][A]$ = tempProb
15:                             back$[i][i+1][A]$ = $B$ and chart location of $B$
16:                             added $\leftarrow$ true
17:                         end if
18:                     end if
19:                 end for
20:             end while
21:         end for
22:     end for
23: end procedure
```

Algorithm 3 CYK part 2 : recursive case]

```
procedure RECURSIVECYK
    for span ← 2, numWords do
        for begin ← 1, numWords - span + 1 do
            end ← begin + span - 1
            for $m$ = begin to end - 1 do
                for $A = 1$ to numNonterminals do
                    for $B = 1$, numNonterminals do
                        for $C = 1$ to numNonterminals do
                            tempProb = prob [begin,$m, B$] * prob [$m + 1$, end,$C$] * P($A \to BC$)
                            if tempProb > prob[begin, end, A] then
                                prob [begin, end, $A$] = tempProb
                                back[begin, end, $A$] = $B, C$ and chart locations of $B$ and $C$
                            end if
                            added ← true
                            while added do
                                added ← false
                                for $A, B$ in nonterms do
                                    if prob $[i][i + 1][B] > 0$ and $A \to B$ in grammar then
                                        tempProb = P($A \to B$) * prob$[i][i + 1][B]$
                                        if tempProb > prob$[i][i + 1][A]$ then
                                            prob$[i][i + 1][A]$ = tempProb
                                            back$[i][i + 1][A] = B$ and chart location of $B$
                                            added ← true
                                        end if
                                    end if
                                end for
                            end while
                        end for
                    end for
                end for
            end for
        end for
    end for
end procedure
```

B Data in the reduced training data file

```
(TOP (S (NP (NNP Ms.) (NNP Haag)) (S@
(VP (VBZ plays) (NP (NNP Elianti))) (.
.))) )

(TOP (S (S (NP (PRP He)) (VP (VBZ
believes) (PP (IN in) (SBAR (WHNP
(WP what)) (S (NP (PRP he)) (VP (VBZ
plays))))))) (S@ (, ,) (S@ (CC and) (S@
(S (NP (PRP he)) (VP (VBZ plays) (ADVP
(RB superbly)))) (. .))))) )

(TOP (S (S%%%%%VP (VBN Clad) (PP (IN
in) (NP (PRP$ his) (NP@ (NN trademark)
(NP@ (JJ black) (NP@ (NN velvet)
(NN suit))))))) (S@ (, ,) (S@ (NP
(DT the) (NP@ (JJ soft-spoken) (NN
clarinetist))) (S@ (VP (VBD announced)
(SBAR (SBAR (IN that) (S (NP (NP (PRP$
his) (NP@ (JJ new) (NN album))) (NP@
(, ,) (NP@ (‘‘ ‘‘) (NP@ (NP (JJ Inner)
(NNS Voices)) (NP@ (, ,) ('' ''))))))
(VP (VBD had) (VP@ (ADVP (RB just))
(VP (VBN been) (VP (VBN released))))))
(SBAR@ (, ,) (SBAR@ (SBAR (IN that) (S
(NP (PRP$ his) (NN family)) (VP (VBD
was) (PP (IN in) (NP (DT the) (NP@
(NN front) (NN row))))))) (SBAR@ (,
,) (SBAR@ (CC and) (SBAR (IN that) (S
(S (NP (PRP it)) (VP (VBD was) (NP (NP
(PRP$ his) (NP@ (NN mother) (POS 's)))
(NN birthday)))) (S@ (, ,) (S@ (RB so)
(S (NP (PRP he)) (VP (VBD was) (VP (VBG
going) (S%%%%%VP (TO to) (VP (VB play)
(NP (NP (PRP$ her) (NP@ (JJ favorite)
(NN tune))) (PP (IN from) (NP (DT the)
(NN record)))))))))))))))))) (. .)))))
)

(TOP (S (PP (IN In) (NP (DT the) (NN
end))) (S@ (NP (NNS politics)) (S@ (VP
(VBD got) (PP (IN in) (NP (DT the) (NN
way)))) (. .)))) )

(TOP (S (NP (PRP He)) (S@ (VP (VBD
said) (SBAR (IN that) (S (NP (NP (CD
one)) (PP (IN of) (NP (DT the) (NNS
computers)))) (VP (VBD took) (VP@ (NP
(DT a) (NP@ (JJ three-foot) (NN trip)))
(S%%%%%VP (VBG sliding) (PP (IN across)
(NP (DT the) (NN floor)))))))))) (. .)))
)

(TOP (S (NP (NN Exchange) (NNS
officials)) (S@ (VP (VBD emphasized)
(SBAR (IN that) (S (NP (DT the) (NP@
(NNP Big) (NNP Board))) (VP (VBZ is)
(VP (VBG considering) (NP (NP (DT a)
(NN variety)) (PP (IN of) (NP (NP (NNS
actions)) (SBAR%%%%%S%%%%%VP (TO to)
(VP (VB deal) (PP (IN with) (NP (NN
program) (NN trading))))))))))))) (.
.))) )

(TOP (S (NP (NNP Tom) (NNP Panelli))
(S@ (VP (VBD had) (NP (NP (DT a) (NP@
(ADJP (RB perfectly) (JJ good)) (NN
reason))) (PP (IN for) (S (RB not) (VP
(VBG using) (NP (NP (DT the) (NP@ (ADJP
($ $) (CD 300)) (NP@ (NN rowing) (NN
machine)))) (SBAR%%%%%S (NP (PRP he))
(VP (VBD bought) (ADVP (NP (CD three)
(NNS years)) (RB ago)))))))))) (. .)))
)
```