CKY Parsing with Probabilistic Context Free Grammar

Samridhi Shree Choudhary
sschoudh@cs.cmu.edu

Abstract

This document presents an implementation of a generative, array based, CKY parser for English. The parser is trained on the files 200 to 2199 of the Penn Treebank. The training trees are binarized with horizontal and vertical markovization. The parser described here is trained with h=2,v=2 markovized trees. Files 2200 to 2299 are read for validation and files 2300 to 2399 are read for testing. The results for the parser in terms of precision(P), recall(R), harmonic mean(F1) and the decoding time for different markovization parameters over varying test and train sentence lengths is presented along with an analysis.

1 Introduction

Parsing or part-of-speech tagging in Natural Language Processing helps uncover the inherent structure of a language. Modelling a sentence to its possible tag sequence is the primary focus of parsing. However, words in English and other languages can take several possible parts of speech. This inherent ambiguity is a major challenge for the tagging models. They tend to produce multiple parses of the same sentence with potentially different meanings. One way to overcome this ambiguity is to build a probabilistic model to give a distribution over the possible derivations of a sentence and select the one with the highest probability. This is the key idea in probabilistic context-free grammars (PCFGs). These grammars extend the definition of a traditional CFG to define the aforementioned distribution over the possible parse trees of a sentence.

2 PCFG - Basic Definition

A probabilistic context free grammar is defined as follows:

1. *A context-free grammar G = (N,$\sum$,S,R), where:*
   - N - finite set of non terminals
   - $\sum$ - finite set of terminals
   - R - finite set of all the rules in the grammar
   - S - start symbol
2. *A parameter :* $q(\alpha \rightarrow \beta)$
   for each rule $\alpha \rightarrow \beta \epsilon R$. This parameter gives the conditional probability of choosing the rule $\alpha \rightarrow \beta$ in a left most derivation, given the non-terminal being expanded is α.

   Given a parse tree $t \epsilon \tau_G$ (τ_G is the set of all possible left-most derivations under the grammar G) containing the rules $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots\ldots\ldots \alpha_n \rightarrow \beta_n$, the probability of *t* is given by:

$$p(t) = \prod_{i=1}^{n} q(\alpha_i \rightarrow \beta_i)$$

In order to parse sentences we first construct a grammar, as described above, from our training corpus. The grammar rules that are generated are either of the form $A \rightarrow B$ (unary) or of the form $A \rightarrow BC$ (binary). This is not exactly similar to CNF since we also allow for unary rules between two non-terminal symbols and not just from a pre-terminal to a terminal.

3 Binarization and Markovization

Probabilistic parsing models deal with ambiguous parses by selecting the tree with highest probability. However ,more often than not, the most probable tree might not be the correct parse of a sentence. The parsing accuracy of an unlexicalized

PCFG can be improved by adding annotations to the grammar. The annotations attach some context to the grammars in order to improve their statistical fit. The parser described in this document makes use of Binarization and Markovization.

3.1 Binarization

Binarization is the process to create binary trees from the raw n-ary trees read from the tree bank, appropriate for learning a PCFG form. At each level of the tree, the process is as follows:

- The left most node is retained and kept the same.
- A new right node is created which now becomes the parent of the rest of the siblings (nodes at the same level).
- The same process is repeated for the entire tree by propagating down this newly created parent.

These intermediate nodes created become new rules and improve the grammar dependency knowledge of the parser in general.

3.2 Horizontal and Vertical Markovization

There are two main challenges that unlexicalized PCFG parsers face while dealing with ambiguity. First, a particular non-terminal might have a different expansion depending on the context in which it appears (for eg *NP* occurring as a subject *NP* or object *NP*). Secondly, the rules that have been seen only once in training have their probabilities overestimated while predicting a parse. Similarly the rules that occur in test data but not in training data, have their probabilities underestimated. A successful way to combat these issues is to *markovize* the rules or attaching some context to it.

Vertical Markovization attaches the vertical history/context to a node in the tree. Every node has a vertical history, including the node itself, parent, grandparent, and so on. We assume only v vertical ancestors matter to the current expansion and use the value $v = 2$ in the parser described here. Hence, we retain the current node and annotate it with its immediate parent.

Horizontal Markovization, similarly attaches the horizontal ancestors to a node. We assume only the previous h ancestors matter and use the value $h = 2$. The results for different h *and* v values are detailed in the later section.

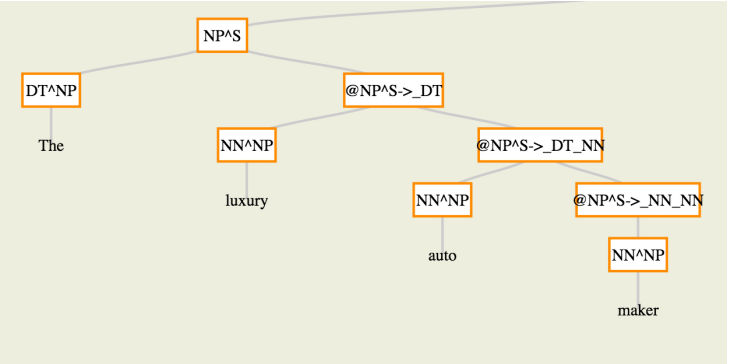

Figure 1: The binarized tree with $h = 2, v = 2$ markovization

Figure 1 shows a segment of a tree read from the treebank and binarized with annotations for h=2, v=2. Each node is annotated with its immediate parent (second order vertical annotation). Similarly each node is annotated with its two most recent horizontal sibling (second order horizontal annotation). The intermediate nodes created as a result start with '@'.

4 CKY Algorithm

In order to build a parse tree for a given input sentence we must find a tree *t* with the highest probability $p(t)$ as described in the sections above. This is done by implementing the CKY algorithm using dynamic programming. In order to handle the unary rules, which now include rules of the form $A \rightarrow B$, where A and B are non-terminals, we maintain two charts - a *unary* score chart and *binary* score chart. The CKY recurrence is used to apply binary rules to elements in the unary chart to produce entries in the binary chart, and then unaries are used to produce entries in the unary chart. Unary rules of the form $A \rightarrow A$ with probability 1 are added for every symbol to allow for unaries to be discarded. This produces trees with alternating unary and binary rules. These trees are later pruned to remove the reflexive unary rules. The closed unary rules are also expanded to produce full chain of the non terminal symbols. The dynamic recurrence is defined below:

- Define 2 charts - $scoreUnary[n][n][N]$ and $scoreBinary[n][n][N]$
- For all $X \epsilon N$, $i \epsilon [1 : n]$

  $scoreUnary[i][i{+}1][X] \leftarrow tagScore(X, word_i)$

- For all $span \epsilon [2 : n]$ and $i \epsilon [1 : n - span]$
  1. $j \leftarrow i + span$
  2. For all binary rules $X \rightarrow YZ$ and $k \epsilon [i + 1 : j - 1]$
     $scoreBinary[i][j][X] \leftarrow$
     $max(scoreBinary[i][j][X],$
     $(ruleScore(X \rightarrow YZ) *$
     $scoreUnary[i][k][Y] *$
     $scoreUnary[k][j][Z])$
  3. For all unary rules $X \rightarrow Y$
     $scoreUnary[i][j][X] \leftarrow$
     $max(scoreUnary[i][j][X],$
     $(ruleScore(X \rightarrow Y) *$
     $scoreBinary[i][j][Y])$

N - All the non terminals.
n - Number of words in the input sentence.
Backpointers are stored whenever the scores are updated in order to trace the path of the optimal rules to generate the final parse tree. Two backpointer structures, for binary rules and unary rules respectively, are maintained and updated when their corresponding scores get modified.

5 Data Structures

The charts used in the implementation of the CKY recurrence, to store the scores, are *3 dimensional arrays*. Lists are used to iterate over the binary and unary rules. Backpointers have been defined as 2 separate classes: one for binary rules and another for unary rules. The non terminals are indexed using a two way map. An indexer is used to return the string representation of a non-terminal.

6 Optimizations for Speed

Two ideas were used to speed up the chart computation in CKY:

1. *Selective Binary Rules* - Instead of iterating over all the binary rules of the grammar, while updating the binary scores, we get the binary rules by left child where the unary score values for those children are not $-\infty$ (all the chart scores have been initialized to $-\infty$).

2. *Selective Unary Rules* - Similarly, instead of iterating over all the unary rules of the grammar, we get the unary rules by child, where the binary scores for those children are not $-\infty$.

3. *Array instead of class for unary backpointers* - There was a minimal increase in decoding speed by using a 3 dimensional integer array for storing unary backpointers instead of instantiating an array of the separate class created.

7 Results

Table 1 shows precision(P), recall(R) and accuracy(F1, harmonic mean) values of the parser for different training and test sentence lengths and the different markovization parameters.
Table 2 shows the total time taken by the parser to parse the test sentences of varying length. The times are shown for parsers with different markovization parameters similar to the values in Table 1.

h	**v**	**Max-Training Length**	**Max-Test Length**	**P**	**R**	**F1**	**EX**
2	2	15	15	85.63	83.68	84.65	44.41
2	2	1000	40	81.31	79.52	80.41	20.97
∞	1	15	15	82.22	76.3	79.15	31.59
∞	1	1000	40	75.39	70.14	72.67	11.15
0	1	15	15	72.88	63.71	67.99	12.82
0	1	1000	40	64.69	52.64	58.05	3.35

Table 1: Accuracy values of the generative CKY parser for different h and v values.

h	**v**	**Max-Training Length**	**Max-Test Length**	**Time Taken**
2	2	15	15	10.43 seconds
2	2	1000	40	25.69 minutes
∞	1	15	15	11.69 seconds
∞	1	1000	40	30.47 minutes
0	1	15	15	2.9 seconds
0	1	1000	40	36.11 seconds

Table 2: Total decoding time of the CKY parser for different h and v values.

8 Analysis

The default notion a treebank PCFG grammar takes is $v = 1$ and $h = \infty$. With infinite horizontal markovization (keeping track of all the siblings that have come before the current node) a complete context is obtained. It is similar to an infinite order n-gram in HMM. Such a model is however prone to overfitting and runs the problem of performing poorly on the test data. Thus we can see that increasing v (getting context) and decreasing h (removing some horizontal context) helps improve accuracy. The results show that the second vertical and horizontal order is an optimum order ($v = 2, h = 2$). The second order *horizontal markovization* retains the two most recent horizontal context but does not over-fit the data or become too sparse like the inifite order. The second order *vertical markovization* distinguishes the identical symbols with different parents. This helps produce a better grammatical structure of the test sentence. This CKY parser produces a good F1 score of ***80.41*** as compared to the F1 score of ***72.67*** for ($v = 1, h = \infty$) parser (for max train length = 1000 and test length = 40). The lowest accuracy scores are seen for ($v = 1, h = 0$) grammar with FI scores of ***67.99*** (for max train and test length = 15) and ***58.05*** (for max train length = 1000 and test length = 40). Since we do not retain either vertical or the horizontal context for this grammar, the low accuracy is expected.

9 Conclusion

State of the art parsers can parse the sentences with length over 40 words in one-tenth of a second. The parser presented in this document parses a sentence in a few seconds and achieves a good accuracy of around **80.41%**. We understood the different aspects of the parser, its limitations and ways to optimize its speed and accuracy in this assignment.

References

Jurafsky, Dan, and James H. Martin. *Speech and language processing.* Pearson, 2014.

Manning, Christopher D., and Hinrich Schtze. *Foundations of statistical natural language processing. Vol. 999.* Cambridge: MIT press, 1999.

Klein, Dan, and Christopher D. Manning. "Accurate unlexicalized parsing." *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1.*
Association for Computational Linguistics, 2003.

Collins, Michael. "Three generative, lexicalised models for statistical parsing." *Proceedings of the eighth conference on European chapter of the Association for Computational Linguistics.*
Association for Computational Linguistics, 1997.