玖、如何實做中文機率式無語境語法剖析器

自然語言處理的研究方法由 80 年代的規則式(rule-based)轉為資料導向(data-driven)的統計式機器學習演算法。標示各種語言訊息的語料因此成為非常重要的研究資源與工具。中研院的中文語法樹庫(Sinica Chinese Treebank)和美國賓州大學中文語法樹庫（Penn Chinese Treebank）就是其中的代表。另一方面自然語言處理學界也體會到開發一些共同的套件(toolkit)對於教學與研究的重要性，Bird, Klein, and Loper (2009)所發展出來的 NLTK (http://nltk.sourceforge.net/index.php/Main_Page)就是在這樣的背景下誕生。

從計算複雜度來看，無語境語法(context-free parser)比有限狀態機(finite-state machine)複雜，而有語境語法(context-sensitive grammar)又比無語境語法(context-free parser)複雜。然而即使有語境語法也無法完全解釋自然語言的現象（學者早已證明自然語言的複雜度等同於圖林機 Turing Machine）從效率和解釋力的觀點來看，無語境語法無疑是折衷的選擇。另一方面，無語境語法(context-free parser)等同於語言學的詞組結構律(phrase structure rules)，不但從語法樹庫擷取較容易，語言學家也比較容易解讀資料提出改善的意見。

Bird, Klein, and Loper (forthcoming)所發展的 NLTK 是一套以 python 語言寫成的程式模組、資料、與教學縱合套件。主要用來支援語言運算，自然語言處理方面的教學與研究之用。NLTK 適合想要學習 NLP（Natural language processing），對 NLP 或相近領域進行研究的學者。NLTK 已被成功的使用在教學、研究、與建立整個研究系統。我們使用 NLTK 中的 PCFG 工具來進行機率式無語境語法(probabilistic context-free parser)的建立。

Stanford parser (Klein and Manning (2002))是一個由史丹佛自然語言處理小組於 2002/12/05 首次釋出的多語 parser，最新的版本為 2006/6/11 釋出之 1.5.1 版。

Stanford parser 是 PCFG 的 JAVA 實作版本，並已做到詞彙處理的高度最佳化。在原始的版本中，這個 parser 主要是由 Dan Klein 與 Christopher Manning 在語言學文法上的基礎下撰寫而成；而之後新增的功能（如：國際化、語言專用模型、彈性 I/O、文法壓縮、使用者支援……等等。）則由 Roger Levy、Christopher Manning、Teg Grenager、Galen Andrew 一齊完成。Stanford parser 本身是一個多國語言剖析器，只要輸入符合英文 Penn Treebank 格式的語法樹庫語料，即可以得到這一個語言的語法剖析器。Penn Chinese Treebank 使用的中文樹庫格式與英文 Penn Treebank 大致相同，但有額外標記和訊息，此種格式由 Xue and Xia (2000)所提出。

Stanford parser 語法分析模型使用 Klein and Manning (2002)提出之 factored parsing 模型。這個模型包括了兩種獨立分析方式：其一為純 maximum likelihood-estimated PCFG 模型；另一個則是 constituent-free dependency parse。而在將原始模型套用在中文分析之上時，每個詞被分開而不標記，語法分析由主要觀察到歧異的幾個分類逐步做過改良。

與 Stanford Parser 同樣架構只要輸入 Penn ChineseTreebank 格式的中文語法樹庫語料即可以處理中文的多國語言的剖析器還包括 Chiang (2003)及 Bikel (2004)。

剖析(parsing)，即在一個句子上，將整個句子的語法樹建立出來的過程。在一般的 context free grammar 之上，語言會產生許多的歧異（ambiguous），舉例：

S → VP

S → NP

VP → V NN

NP → V NN

V → 進口

NN → 汽車

在這個文法之中，〔進口〕〔汽車〕可被建構成兩種不同的結構樹：

[S [VP [V 進口] [NN 汽車]]]

[S [NP [V 進口] [NN 汽車]]]

一個純粹的 CFG parser 便無法判斷此句子的結構，無法良好的應用在語言處理之上。因此，(probabilistic context free grammar)出現了，其中主要是加權文法的概念（weighted grammar）。在每一個 derivation(也就是詞組結構律)之中，我們增加了另一個參數：機率值。PCFG 建構原始 CFG parser 會建構出來的分析結果，但在每個結果之中，PCFG 為其連結了一個機率值，其值即簡單的由所有使用到的 derivation 機率值相乘而得到。

PCFG 的實作方面，我們先對中研院語法樹庫及賓州大學中文語法樹庫做前處理，由於中研院句法樹庫的格式與 Penn Chinese Treebank 不同，因此我們先將中研院語法樹庫改成跟賓州大學中文語法樹庫一致的結構，接下來寫一個剖析這些樹狀結構的剖析器，計算每條詞組結構律的機率，再利用 NLTK 裡面的函式庫。我們採用 Viterbi-style 的分析模型。Viterbi PCFG parser 是一由下而上的剖析器(bottom-up parser)，使用動態規畫(dynamic programming)來找出最有可能的分析結構樹。它藉由反覆的填寫一個最有可能的詞組表格“most likely constituent table”來做句子的分析，這個表格為所有分支與節點記錄了最有可能之樹結構。更詳細的說，它擁有四個欄位，分別記錄：

1. 分支開始索引
2. 分支結束索引
3. 節點標記
4. 結構樹

舉例來說，在分析「我 在 樓梯 上 看到 教授」之後，表格可能如表（五）所示：

表（五）

Most Likely Constituents Table			
分支	節點	結構樹	機率
[0:1]	NP	(NP 我)	0.3
[2:3]	NP	(NP 樓梯)	0.3
[5:6]	NP	(NP 教授)	0.3
[1:4]	PP	(PP 在 (NP 樓梯) 上)	0.05
[5:6]	VP	(VP 看到 (NP 教授))	0.03
[0:4]	NP	(NP (NP 我) (PP 在 (NP 樓梯) 上))	0.01
[0:6]	S	(S (NP (NP 我) (PP 在 (NP 樓梯) 上)) (VP 看到 (NP 教授)))	0.0001

當成功的填寫完這個表格時，parser 簡單的傳回起始節點（S）中，機率值最大的結構樹。

由於 Viterbi parser 使用的文法是 PCFG，任何一個元素的機率值都可以由它的子孫求得，而任何一個分支並不可能含括到比自己還要大的分支，因此任何一個分支便僅與較小的分支有關。

Viterbi parser 利用上述的條件，由較小的單元開始填表，從大小為 1 的元素開始，再來大小為 2、3、4，以此下去直到表內所有的空位都被填滿為止。

以下為上面例子的分析標記過程：

Inserting tokens into the most likely constituents table...

Insert: |=.....| 我

Insert: |.=....| 在

Insert: |..=...| 樓梯

Insert: |...=..| 上

Insert: |....=.| 看到

Insert: |.....=| 教授

Finding the most likely constituents spanning 1 text elements...

Insert: |=.....| NP -> '我' (p=0.15) 0.1500000000

Insert: |.=....| P -> '在' (p=0.61) 0.6100000000

Insert: |.=....| NP -> '樓梯' (p=0.5) 0.500000000

Insert: |..=...| LC -> '上' (p=0.1) 0.5000000000

Insert: |...=..| V -> '看到' (p=0.61) 0.6500000000

Insert: |....=.| VP -> V (p=0.2) 0.1300000000

Insert: |.....=| NP -> '教授' (p=0.5) 0.5000000000

Finding the most likely constituents spanning 2 text elements...

Insert: |....==| VP -> V NP (p=0.7) 0.0455000000

Finding the most likely constituents spanning 3 text elements...

Insert: |.===..| PP -> P NP LC (p=1.0) 0.1525000000

Finding the most likely constituents spanning 4 text elements...

Insert: |====..| NP -> NP PP (p=0.25) 0.0057187500

Finding the most likely constituents spanning 5 text elements...

Insert: |.=====| VP -> PP VP (p=0.1) 0.0069387500

Discard: |.=====| VP -> PP VP (p=0.1) 0.0069387500

Finding the most likely constituents spanning 6 text elements...

Insert: |======| S -> NP VP (p=1.0) 0.0002602031