Dependency Parsing by Qiao

Monday, August 24, 2020 12:57 PM

主要的讨论：
1.依存分析树
2.Transition-based Parsing
3.Graph-based parsing

By Qiao for NLP7 2020-08-30
Core reference: Speech and Language Processing 3rd

Dependency Parsing vs. Constituency Parsing

- 直接表示词汇间的依赖关系，成分分析则较为复杂抽象
- 有向、带标签的边，从被依赖的词(**head**)指向依赖词(**dependent**)
- 对于词序灵活(free word order)的语言较成分分析语法的优势更突出

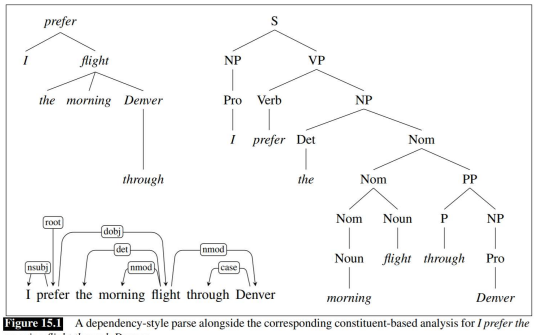
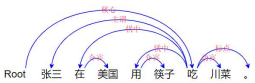


Figure 15.1 A dependency-style parse alongside the corresponding constituent-based analysis for *I prefer the morning flight through Denver.*

LTP依存句法分析模型使用的关系标签



依存句法关系类型

关系类型	Tag	Description	Example
主谓	SBV	subject-verb	我送她一束花 (我 <- 送)
动宾	VOB	直接宾语，verb-object	我送她一束花 (送 -> 花)
间宾	IOB	间接宾语，indirect-object	我送她一束花 (送 -> 她)
前置宾语	FOB	前置宾语，fronting-object	他什么书都读 (书 <- 读)
兼语	DBL	double	他请我吃饭 (请 -> 我)
定中	ATT	attribute	红苹果 (红 <- 苹果)
状中	ADV	adverbial	非常美丽 (非常 <- 美丽)
动补	CMP	complement	做完了作业 (做 -> 完)
并列	COO	coordinate	大山和大海 (大山 -> 大海)
介宾	POB	preposition-object	在贸易区内 (在 -> 内)
左附加	LAD	left adjunct	大山和大海 (和 <- 大海)
右附加	RAD	right adjunct	孩子们 (孩子 -> 们)
独立	IS	independent structure	两个单句在结构上彼此独立
核心	HED	head	指整个句子的核心
标点	WP	punctuation	标点

Parsing结果的常用评价方法

Evaluation
- labeled attachment score (LAS)
  - 每一个单词，其关系标签和有向边都正确的比例
- unlabeled attachment score (UAS)
  - 每一个单词，有向边正确的比例
- label accuracy score （LS）
  - 跟踪某个依赖关系(例如：NSUBJ)在测试集中预测的precision和recall

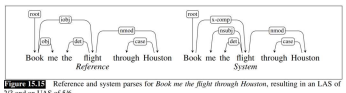


Figure 15.2 Reference and system parses for *Book me the flight through Houston*, resulting in an LAS of 2/3 and an UAS of 5/6.

依存句法分析算法1： Transition-Based依存分析

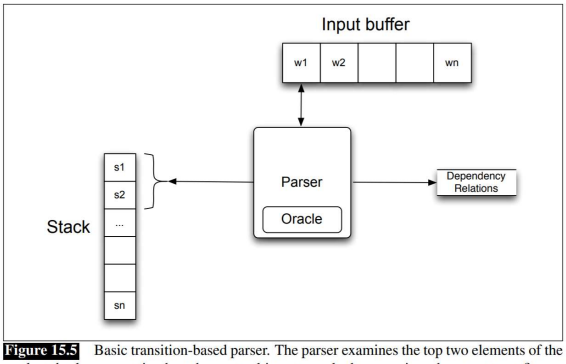


Figure 15.5 Basic transition-based parser. The parser examines the top two elements of the stack and selects an action based on consulting an oracle that examines the current configuration.

Arc Standard transition 操作

- **LEFTARC**:
  为栈顶两个元素建立一条依存关系边，边的方向为从栈顶元素指向栈顶第二个元素，同时去除第二个元素
- **RIGHTARC**:
  为栈顶两个元素建立一条依存关系边，边的方向为从栈顶第二个元素指向栈顶元素，同时去除栈顶元素
- **SHIFT**:
  将Input buffer的头部元素压入栈中

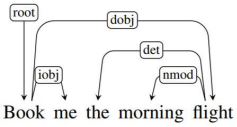
Arc Standard特点：transition操作仅对栈顶部的元素进行，一旦元素已经和父节点建立关系，就将其从栈中移除，不再参与后续计算。

Arc-eager transition 操作

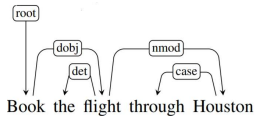
- **LEFTARC**:
  为栈顶元素和input buffer前部元素建立依存关系边，边的方向为从buffer顶部元素指向stack顶部元素，同时弹出栈顶元素。
- **RIGHTARC**:
  为栈顶元素和input buffer的前部元素建立依存关系边，边的方向为从stack顶部元素指向buffer顶部元素，同时将buffer头部元素压入栈中。
- **SHIFT**:
  将buffer头部元素压入栈中
- **REDUCE**:
  弹出栈顶元素

Arc Standard的特点是RIGHTARC的执行通常会比较晚。例如book和flight, 在句子Book the flight through Houston中，由于flight必须等到houston->through处理完成之后才能进行RIGHTARC操作, 于是BOOk->flight弧的建立会很晚，尽管它们入栈很早。通常等待越久，出错可能性越高。Arc-eager 系统则能优先建立依存关系，因为它的RIGHTARC不要求移除元素。

Step	Stack	Word List	Action	Relation Added
0	[root]	[book, me, the, morning, flight]	SHIFT	
1	[root, book]	[me, the, morning, flight]	SHIFT	
2	[root, book, me]	[the, morning, flight]	RIGHTARC	(book → me)
3	[root, book]	[the, morning, flight]	SHIFT	
4	[root, book, the]	[morning, flight]	SHIFT	
5	[root, book, the, morning]	[flight]	SHIFT	
6	[root, book, the, morning, flight]	[]	LEFTARC	(morning ← flight)
7	[root, book, the, flight]	[]	LEFTARC	(the ← flight)
8	[root, book, flight]	[]	RIGHTARC	(book → flight)
9	[root, book]	[]	RIGHTARC	(root → book)
10	[root]	[]	Done	



Step	Stack	Word List	Action	Relation Added
0	[root]	[book, the, flight, through, houston]	RIGHTARC	(root → book)
1	[root, book]	[the, flight, through, houston]	SHIFT	
2	[root, book, the]	[flight, through, houston]	LEFTARC	(the ← flight)
3	[root, book]	[flight, through, houston]	RIGHTARC	(book → flight)
4	[root, book, flight]	[through, houston]	SHIFT	
5	[root, book, flight, through]	[houston]	LEFTARC	(through ← houston)
6	[root, book, flight]	[houston]	RIGHTARC	(flight → houston)
7	[root, book, flight, houston]	[]	REDUCE	
8	[root, book, flight]	[]	REDUCE	
9	[root, book]	[]	REDUCE	
10	[root]	[]	Done	



Parser的机器学习:

任务:
给定Stack和Input buffer的当前状态，预测Action:
SHIFT, RIGHTARC[relation label], LEFTARC[Relation Label]

训练集：
原始数据: 人工标注的"句子-依存分析树"对（各种树库）
利用原始数据构建Parsing中Transition过程的Action序列。因为每一个训练集的句子和分析树已知，所以我们可以按照如下规则在parsing过程中选择Action序列。
（初始化stack和buffer）
- 在当前的stack和buffer状态下，如果选择LEFTARC可以生成一个正确的依存关系边，则在Action序列中加入一个LEFTARC，同时按照transition rule （Arc Standard）更新stack和buffer
- 否则, 如果选择RIGHTARC可以生成一个正确的依存关系边，并且依存于栈顶元素子节点都已经建立完边，则在Action序列中加入一个RIGHTARC，同时按照transition rule 更新stack和buffer
- 否则，选择SHIFT并且更新stack和buffer

特征工程

Source	Feature templates		
One word	$s_1.w$	$s_1.t$	$s_1.wt$
	$s_2.w$	$s_2.t$	$s_2.wt$
	$b_1.w$	$b_1.w$	$b_0.wt$
Two word	$s_1.w \circ s_2.w$	$s_1.t \circ s_2.t$	$s_1.t \circ b_1.w$
	$s_1.t \circ s_2.wt$	$s_1.w \circ s_2.w \circ s_2.t$	$s_1.w \circ s_1.t \circ s_2.t$
	$s_1.w \circ s_1.t \circ s_2.t$	$s_1.w \circ s_1.t$	

Figure 15.9 Standard feature templates for training transition-based dependency parsers. In the template specifications s_n refers to a location on the stack, b_n refers to a location in the word buffer, w refers to the wordform of the input, and t refers to the part of speech of the input.

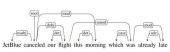
深度学习Neural Parser

- Feedforward + Features

A fast and accurate dependency parser using neural networks
- LSTM

Transition-Based Dependency Parsing with Stack Long Short-Term Memory

有趣的应用，在句子生成中加入语法信息 (利用的是Parsing过程中，transition的Action序列):
Sequence-to-Dependency Neural Machine Translation

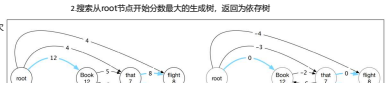


依存句法分析算法2：基于图的依存分析

- 基于Transition的Parser做的是局部决策，对短的依赖关系的分析效果好，对长程依赖效果欠佳，此外，它无法解决non-projective的依存关系。
- 基于图的算法可以处理，它在评估树可能性时为整棵树打分，做全局决策。

Edge-factored scoring:

基于最大生成树(MST)的Parsing算法:
1.构建全连接的有向图：主要工作为边权重的计算
2.搜索从root节点开始分数最大的生成树，返回为依存树



生成树(Spaning Tree)的概念:
给定一个全连接图G = (V,E), 设G的一个子图是T = (V,F)，当T没有环且每个节点（除了根节点）有且仅有一条入边时，称T是G的一棵生成树。我们求最大/最小生成树即找到生成树中得分最大/最小的那一棵。

依存句法分析算法2：基于图的依存分析

- 基于Transition的Parser做的是局部决策，对短的依赖关系的分析效果好，对长程依赖效果欠佳 。此外，它无法解决non-projective的依存关系。
- 基于图的算法可以处理，它在评估树可能性时为整棵树打分，做全局决策。

Edge-factored scoring:

$$\hat{T}(S) = \underset{t \in \mathscr{G}_S}{\operatorname{argmax}}\, score(t, S)$$

$$score(t, S) = \sum_{e \in t} score(e)$$

$$\text{score}(e) = \sum_{i=1}^{N} w_i f_i(e)$$

机器学习的特征工程

• Wordforms, lemmas, and parts of speech of the headword and its dependent.
• Corresponding features derived from the contexts before, after and between the words.
• Word embeddings.
• The dependency relation itself.
• The direction of the relation (to the right or left).
• The distance from the head to the dependent. As with transition-based approaches, pre-selected comb

1.构建全连接的有向图：主要工作为边权重的计算
2.搜索从root节点开始分数最大的生成树，返回为依存树

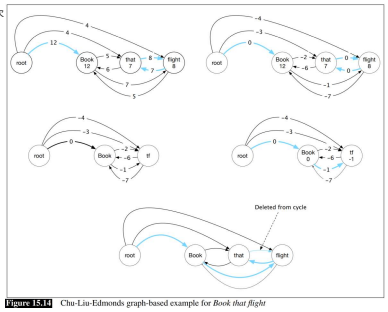


Figure 15.14 Chu-Liu-Edmonds graph-based example for *Book that flight*

生成树(Spaning Tree)的概念:

给定一个全连接图G = (V,E), 设G的一个子图是T = (V,F)，当T没有环且每个节点（除了根节点）有且仅有一条入边时，称T是G的一棵生成树。我们求最大/最小生成树即找到生成树中得分最大/最小的那一棵。

求MST算法的两个重要根据：

i. 生成树中的每个节点（除根节点）严格的只有一条入边。
ii. 在搜索MST时，需要评估每一个节点的所有入边的分值，但入边的绝对分值不是必须的，将绝对分值替换为相对分值（例：所有入边的分值减去同一数值）不影响搜索结果。
（帮助你理解这个事实：某个节点的所有入边分值减去同一数值后，所有的生成树因为都必须包含这个节点，所以这个节点的入边分值都被减去了这个数值，于是所有生成树的总分值也就都减去了这一数值，MST不会变化。）

算法步骤：

1. 构建全连接图：从根节点连出单向有向边至所有单词节点，每个单词节点与其他所有节点构建有向边。
2. 对所有节点（除根节点），标记分值最大的入边。
3. 如有环，递归进行消环操作：
   a. 调节每条边的权重：对每一个节点，将其所有入边的分数减去其中最大的分数。
   结果是所有第2步中选择的边的分数调整为0，包括环的所有边。
   b. 选择一个环，将它压缩为一个新的节点（原来环上节点的所有外部出入边都链接到这个新节点上）从而构建了一个新的图。
   c. 对所有节点(除根节点)，选择分值最大的入边。
   d. 递归调用b-c步直到新图不再有环。
   e. 将压缩部分展开，恢复原节点和边。这时，原来环中某个节点会有两条被标记的入边：1）原来属于环的一条标记边；2）压缩后新标记的一条由环外连入环的入边。此时，移去1）中环的标记，保留2）中的新标记。这样便破坏了原来环的结构。
   f. 反向递归，执行e步，直到递归栈顶，消除所有环。
4. 被标记的边构成合法的MST，计算其得分并返回结果。

环的坍缩操作是为了替换环中某个节点的入边为环外节点，起到破坏环的作用。