句法分析

概述

句法分析是指给定一个句子，分析句子的句法成分信息，如句子的主谓宾等成分。其目的是将词序列表示的句子转换成树状结构，从而有助于准确地理解句子的含义，并且辅助其它自然语言处理任务。

句法是现代语言学研究中的重要课题，有大量的句法理论相关研究。句法理论在很多语法理论研究中也都占据了主要部分。每种自然语言都可以看作由正确句子构成的集合。通常一种自然语言中包含的词和语素可以达到几万甚至几十万，而句子的长度可以超过 100 个单词，甚至更长。因此，不可能采用穷举的方式将一个语言中所有正确的句子保存起来。语法（Grammar）就是指自然语言中句子、短语以及词等语法单位的语法结构与语法意义的规律。根据语法就可以判断不同成分组成句子的方式以及决定句子是否成立。语法学的外延相关广泛，甚至有些语言学家认为音系学和语义学都包含在语法学中。我们采用比较狭义的语法学定义，即认为语言学不包含音系学和语义学，因此狭义的语法学研究基本等同于句法学。语言学家自 19 世纪 50 年代以来，构建了大量表达明确并且形式化的语法理论，对自然语言句法分析提供了理论支撑。

语法理论在构建时，一个重要的问题是该理论是基于成分关系还是基于依存关系。如果一种语法理论基于成分关系，那么该理论就属于成分语法（Constituent Grammar），也称短语结构语法（Phrase Structure Grammar）。成分语法主要包含：范畴语法（Categorical Grammar）、词汇功能语法（Lexical Functional Grammar）、最简方案（the Minimalist Program）等。如果一个语法理论基于依存关系，那么该理论就属于依存语法（Dependency Grammar）。依存语法主要包含：文本-意义理论（Meaning Text Theory）、词格理论（Lexicase）、功能生成描述理论（Functional Generative Description）等。

句法分析常见任务包括：（1）判断输入的字符串是否属于某种语言；（2）消除输入句子中词法和结构等方面的歧义；（3）分析输入句子的内部结构，如成分构成、上下文关系等。通常默认知道当前处理的文本的语言种类，着重的是任务(2)、(3)。下面将分别针对成分语法和依存语法理论进行简单介绍。

1. 成分句法分析

成分句法分析是一种常见的句法分析方法，用于分析自然语言中句子的结构。它将句子分解为各种成分（短语或词类），并确定它们之间的层次结构。这种方法通常基于短语结构语法（Phrase Structure Grammar）。

下面是成分句法分析的一般步骤：

（1）**分词**：首先，对输入的句子进行分词，将其划分为单词或词组，这是成分句法分析的基础。

（2）**词性标注**：对每个词语进行词性标注，确定每个词语的词类（名词、动词、形容词等）。词性标注是成分句法分析的重要预处理步骤，它为后续的句法分析提供了基本的词汇信息。

（3）**构建成分结构**：基于语法规则，将句子分解为各种成分，如名词短语（NP）、动词短语（VP）、形容词短语（AP）等。这些成分可以根据语法规则的描述进行递归地构建，直到句子被完全分解为最小的成分单元。

（4）**构建树状结构**：将各种成分组合成树状结构，其中树的根节点代表整个句子，内部节点代表不同的成分，叶节点代表词语或词组。这种树状结构被称为成分树（Parse Tree）或短语结构树（Phrase Structure Tree）。

（5）**分析树的关系**：确定树中各个节点之间的关系，通常使用父子关系（如一个成分是另一个成分的子节点）或兄弟关系（如同级成分之间的关系）来表示。这些关系描述了句子中各个成分之间的语法关系。

成分可以独立存在，或者可以用代词替代，又或者可以在句子中的不同位置移动。例如：他正在写一本小说，其中，“一本小说”是一个成分。在此基础上，根据不同成分之间是否可以进行相互替代而不会影响句子语法正确性，可以进一步地将成分进行分类，某一类短语就属于一个句法范畴（Syntactic Category）。比如“一本小说”、“一所大学”等都属于一个句法范畴：名词短语（None Phrase，NP）。句法范畴不仅仅包含名词短语（NP）、动词短语（VP）、介词短语（PP）等短语范畴，也包含名词（N）、动词（V）、形容词（Adj）等词汇范畴。除此之外还包含功能范畴（包括冠词、助动词等）。

句法范畴之间不是完全对等的，而是具有层级关系。例如：一个句子可以由一个名词短语和一个动词短语组成，一个名词短语可以由一个限定词和一个名词

组成，一个动词短语又可以由一个动词和一个名词短语组成。我们可以定义短语结构规则（Phrase Structure Rules）又称改写规则或重写规则，对句法范畴间的关系进行形式化描述。通常可以用 $X \rightarrow Y\ ZW \cdots$ 表示，其中 X 表示短语名称，“→”表示“改写为”，“Y ZW · · ·”定义了短语 X 的结构，如果 YZW 是短语，则还需要构造出它们的规则。例如：

$$S \text{ –> } NP\ VP \ ； \ NP \text{ –> } Det\ N \ ； \ VP \text{ –> } V\ NP$$

成分语法就是由句法范畴以及短语结构规则定义的语法。由于短语结构规则具有递归性，可以 使短语和句子无限循环组合。这也说明了语言的创造性和无限性。如图 1 和图 2 所示，一个句子根据成分语法分析得到的层级结构，可以使用成分句法树进行表示。

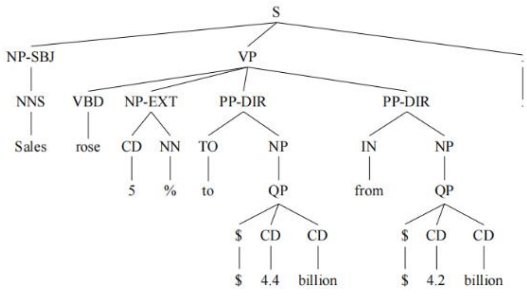


图 1. Penn Treebank3.0 成分句法树样例

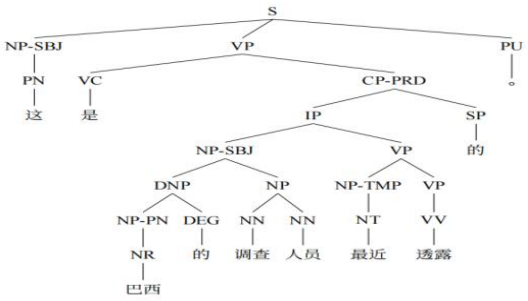


图 2 .Chnese Treebank 7.0 成分句法树样例

由于成分语法局限于表层结构分析，不能彻底解决句法和语义问题，因此存在非连续成分、结构歧义等问题。如图 3 和图 4 所示，对于句子“The boy saw the man with telescope”有两种可能的合法的句法结构树，不同的树结构对应不同的语义。第一种句法树表示“男孩使用望远镜看到了这个男人”，第二种句法树表示“男孩看到了一个拿着望远镜的男人”。

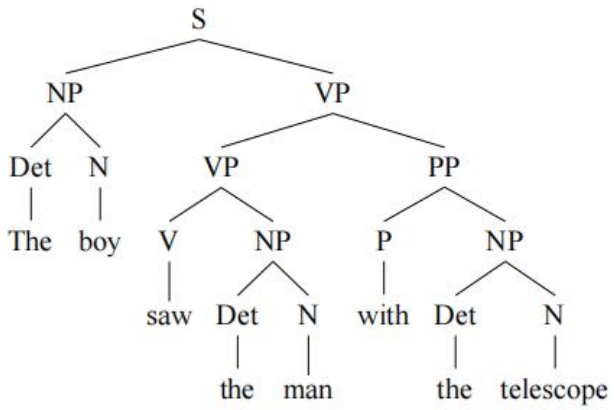


图 3. 句子“The boy saw the man with the telescope”的第一种成分句法树

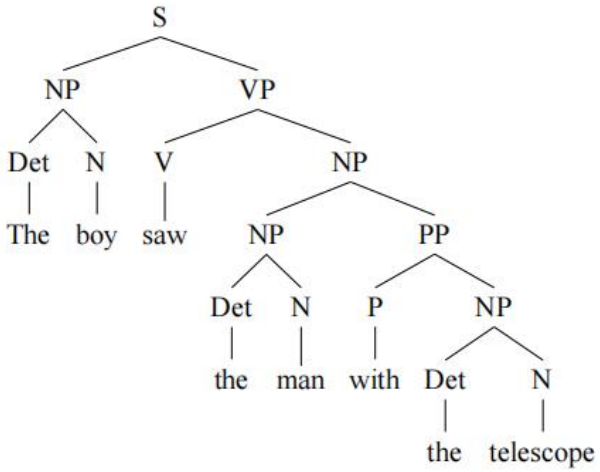


图 4. 句子“The boy saw the man with the telescope”的第二种成分句法树

成分句法分析的方法可以分为三类：基于规则的句法分析方法、基于统计的句法分析方法和基于概率上下文无关文法的句法分析方法。基于统计的句法分析方法是目前被研究者关注较多的方法，该方法的基本思想由生成语法定义被分析的语言及其分析出的类别，在训练数据中观察到的各种语言现象的分布以统计数据的方式与语法规则一起编码。成分句法分析最常用的算法为基于概率的上下文无关文法（PCFG）。

1.1 基于上下文无关文法的成分句法分析

上下文无关文法 G 包含以下四个部分：

（1）**终结符集合 P**（Terminal Symbols）：与语言中词汇对应的符号，用 u, v, w 等小写罗马字母表示。

（2）**非终结符集合 N**（Non-terminals）：对应词组等聚集类或概括性符号，用 A,B,C 等大写字母表示。例如：NP（名词短语），VP（动词短语），N（名词），介词（P）。

（3）**初始符 S**（Start symbol）：语法中指定的初始符，通常用 S 表示。

（4）**规则集 R**（Rules）：对应语法中的短语结构规则，从初始符号开始可以构造出合法句子的规则集合，在上下文无关语法中，一个规则的左边是一个单独的非终结符，右边是一个或者多个终结符或非终结符组成的有序序列$(P\cup N)^*$。用 α，β，γ 等小写希腊字母表示。例如：S→ NP VP，NP→ Det Adj N。

由于句法结构具有歧义，因此句法分析中最重要的工作之一也是如何消除歧义。成分语法中的结构歧义主要有两种：附着歧义（Attachment ambiguity）以

及并列连接歧义（Coordination ambiguity）。图 3.3 和图 3.4 所给出的关于句子“The boy saw the man with the telescope”的两种分析结果就是附着 歧义的一个例子，其核心就是“with the telescope”附着于动词 saw 还是名词短语 the man。并列连接歧义也是常见的结构歧义之一，例如，短语“重要政策和措施”中“重要”可以修饰“政策和措施”整体，也可以修饰“政策”，就可以表示为两种层级结构，“[重要 [政策和措施]]”和“[重要政策] 和 [措施]”。但是由于结构歧义通常伴随语义上的变化，因此句法结构歧义的消除通常需要依赖统计知识、语义知识甚至是语用知识才能更好地完成。

在给定上下文无关文法 G 的情况下，对于给定的句子 W = {w1, w2,..., wn}，输出其对应的句法结构，通常有两大类搜索方法：自顶向下和自底向上。自顶向下搜索试图从根节点 S 出发，搜索语法中的所有规则直到叶子节点，并行构造所有可能的树。自底向上的方法是从输入的单词开始，每次都是用语法规则，直到成功构造了以初始符 S 为根的树。

1.2 基于概率上下文无关文法的成分句法分析

基于概率的上下文无关文法（Probabilistic Context-Free Grammar，PCFG）是对上下文无关文法的扩展，它为文法的规则引入了概率权重，表示每条规则应用的可能性。上下文无关文法虽然比较简单且容易理解，但是对于句子结构歧义无法很好地处理。比如句子“He eat soup with spoon”具有两种可能的句法结构树，并且两种树结构都符合语法，如图 5 所示。使用上下文无关文法很难从这两种句法树中进行选择。基于概率上下文无关文法（Probabilistic Context-Free Grammar，PCFG）的句法分析则可以结合规则方法和统计方法，在一定程度上缓解了上述歧义问题。

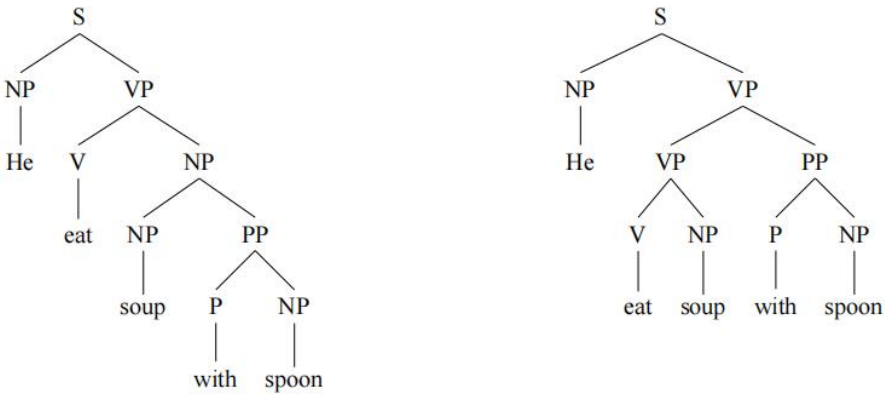


图 5. 句子“He eat soup with spoon”的成分句法树

PCFG 是 CFG 的扩展，因此 PCFG 的文法也是由终结符集合 P 、非终结符集合 N、初始符 S 以及规则集合 R 组成。只是在 CFG 的基础上对每条规则增加了概率，其规则用如下形式表示：

$$A \to \alpha, p$$

其中 A 为非终结符，$\alpha \in (P \cup N)^*$ 为终结符和非终结符组成的有序序列集合，p 为 A 推导出α的概率，即 $p = P(A \to \alpha)$，该概率分布要满足如下条件：

$$\sum_{\alpha} P(A \to \alpha) = 1$$

也就是说，每一个非终结符展开得到的概率之和为 1。因此可以根据一个句子及其句法分析树计算特定句法分析树的概率。利用特定分析树的概率可以用于消除分析树的歧义。

句法分析树概率计算是指在给定 PCFG 文法 G、句子 W = w1w2...wn 以及句法树 T 的情况下，计算句法分析树概率 P(T)。为了简化句法树概率计算，句法树概率计算还要应用以下三个独立假设：

(1) 位置不变性（Place in-variance）：子树的概率不依赖该子树所在的位置；

(2) 上下文无关性（Context-free）：子树的概率不依赖于子树以外的单词；

(3) 祖先无关性（Ancestor-free）：子树的概率不依赖于子树的祖先节点。

基于上述独立性假设，一个特定句法树 T 的概率定义为该句法树 T 中用来得到句子 W 所使用的 m 个规则的概率乘积：

$$P(T) = \prod_{i=1}^{m} P(A_i \to \alpha)$$

假设给定如下 PCFG 文法：

G(S):	S → NP VP	1.0	NP → He	0.3
	VP → VP PP	0.8	NP → soup	0.3
	VP → V NP	0.2	NP → spoon	0.2
	NP → NP PP	0.2	V → eat	1.0
	PP → P NP	1.0	P → with	1.0

句子“He eat soup with spoon”的两种句法结构包含概率的形式如图 6 所示：

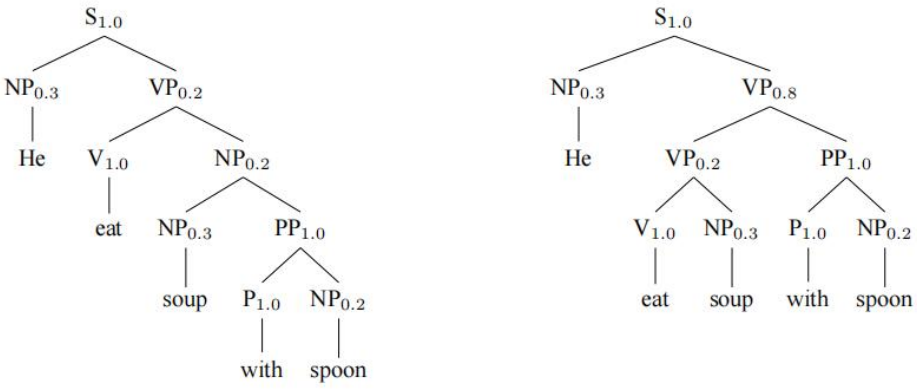


图 6. 句子“He eat soup with spoon”两种可能的句法结构包含概率的形式

根据句法树概率计算公式，图 6 所示的两个句法树的概率分别为：

P(TLeft) =P(S→NP VP)×P(NP→He)×P(VP→V NP)×

P(V→eat)×P(NP→NP PP)×P(NP→soup)×

P(PP→P NP)×P(P→with)×P(NP→spoon)

=1.0 × 0.3 × 0.2 × 1.0 × 0.2 × 0.3 × 1.0 × 1.0 × 0.2 = 0.00072

P(TRight)=P(S→NP VP)×P(NP→He)×P(VP→VP PP)×

P(VP → V NP)×P(V→eat)×P(NP→soup)×

P(PP→P NP)×P(P →with)×P(NP→spoon)

=1.0 × 0.3 × 0.8 × 0.2 × 1.0 × 1.0 × 0.3 × 1.0 × 0.2 = 0.00288

由此可以选择图 6 右侧树结构，从而解决针对句子“He eat soup with spoon”的句法树歧义问题。

2. 依存结构句法分析

依存结构句法分析是一种句法分析方法，其核心思想是通过分析词与词之间的依存关系来揭示句子的结构。在这种分析中，句子被表示为一组词汇和它们之间的有向边，这些边表示了词与词之间的依存关系，其中每个词都有一个中心词，并与其他词形成依存关系。以下是依存结构句法分析的基本步骤：

（1）**分词**：首先，将输入的句子分割成单词或词组的序列。

（2）**词性标注**：对每个词语进行词性标注，确定每个词语的词类（名词、动词、形容词等）。

（3）**建立依存关系**：通过分析词与词之间的语法关系，建立句子中词与词之间的依存关系。每个词都有一个中心词（头部），而其他词依存于中心词，形成不同类型的依存关系，如主谓关系、动宾关系等。

（4）**构建依存树**：根据建立的依存关系，构建一棵依存树，其中树的根节点通常是句子的核心词（如动词或名词），而其他词则通过有向边与核心词相连，表示它们与核心词之间的依存关系。

（5）**分析依存树**：对生成的依存树进行分析，包括识别不同类型的依存关系、理解句子的语法结构等。这有助于理解句子的含义和结构。

在基于依存关系的语法中，句子中的每个成分对应句法结构中的唯一一个节点。两个成分之间的依存关系是二元的非对称关系，具有方向性，一个成分是中心语，另一个成分依附于中心语存在，关系从中心语成分指向依存成分。中心成分称为中心词或支配者（Governor，Regent，Head），依存成分也称为修饰词或从属者（Modifier，Subordinate，Dependency）。例如："读书"中 "读"是中心语,"书"依存于"读",有多种方式可以用于表示依存关系，如图 7 所示。

图 7. "读书"的依存关系表示实例

两个单词之间是否存在依存关系？单词之间谁处于支配地位？谁处于从属地位？建立这些词与词之间关系的依据是什么？很多依存句法理论从不同方面对上述问题进行了回答。配价（Valency）理论是其中最为经典的论著之一。这里"价"的概念是由 Lucien Tesnière 从化学中的"化合价"的概念引入语言学研究。价是词语的一个属性，表示某个词语与其他词语结合的能力。配价模式则是描述了某一个具有特定意义的词的出现语境，以及当一个词出现在特定的模式下时，还有哪些词语会出现在这个模式下及其语义角色。通过对不同词类的支配和被支配能力（配价）以及词类间依存关系类型的定义，就可得出某种具体的依存句法。利用配价理论，可以将汉语里的动词 V 根据其价数,分为以下四类：

(1) 零价动词不强制与某个行动元关联的动词，例如：地震、刮风及下雨等；

(2) 一价动词强制与一个行动元关联的动词，例如：病、醉、休息及咳嗽等；

(3) 二价动词强制与两个行动元关联的动词，例如：爱、采、参观及讨论等；

(4) 三价动词强制与三个行动元关联的动词，例如：给、送、告诉及退还等。

有一种特殊的关系是并列关系，构成这种关系的元素之间不存在支配与被支配关系。但是为了能够使得与其他的关系统一，也通过并列关系标记将其纳入句法树中，通常将第一个词作为中心语，其余的词作为该词的从属成分。这种情况下，在树结构上表现出来的层级关系是一种伪层级。

词语间的依存关系还可以根据语法关系定义为不同的类型，Carroll 等人将依存关系细分为了 20 种，并给出了关系之间的层级结构。Marneffe 等人在上述工作的基础上对依存关系进行了进一步的细化，定义了 48 种依存关系，主要分为论元依存关系和修饰语依存关系两大类。图 8 给出了一个包含依存关系类型的句法树的样例。

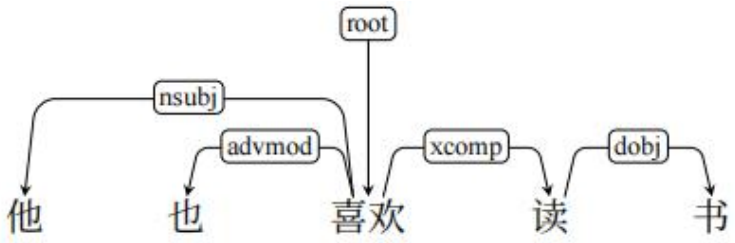


图 8. 依存关系类型的句法树样例

通常依存句法树引入 root 做为句子或者单棵树的主要支配者，是树的根节点。一般情况下动词是 root 节点的直接从属成分，其余节点应该直接或者间接依存于动词节点。没有动词的句子除外，但是应该存在一个句子成分做为 root 的从属成分。依存句法树中的每个节点只能有一个支配者，但可以有多个从属者。一个符合语法的句子中，所有节点应该是相连的，不允许存在游离在外的节点。

依存语法中根据依存成分与中心语或姐妹成分在语序上的关系，可以分为符合投射性原则和违反投射性原则两类。在依存层级中如果每个依存成分与其中心或姐妹成分相邻，那么该依存层级就是符合投射性原则（Projectivity），如图 9 符合投射原则依存句法树样例所示，依存树中没有任何两线交叉，因此是投射性的。如果依存成分的相邻成分使其与中心语分离，那么就是违反投射性原则的。如图 10 违反投射原则依存句法树样例所示，成分“about this book”与其中心语不相邻，因此存在交叉线，违反了投射原则。这种现象在依存句法中通常称为远距离依赖（Long-distance Dependencies）。

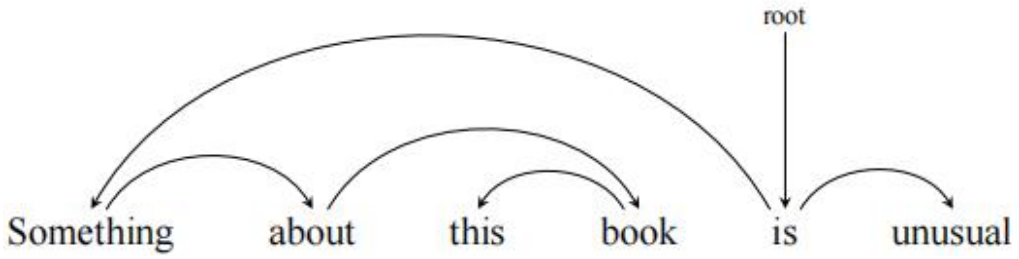


图 9. 符合投射原则依存句法树样例

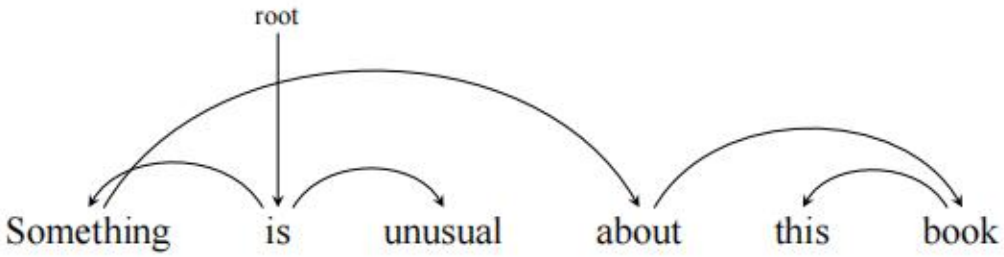


图 10. 违反投射原则依存句法树样例

2.1 基于转移的依存句法分析

基于转移的依存句法分析是一种通过一系列转移动作逐步构建句子中词语之间依存关系的句法分析方法。在这种方法中，分析器利用一个栈和一个缓冲区来存储待处理的词语，并通过移动和操作这些词语来逐步生成依存结构。

下面是基于转移的依存句法分析的详细步骤，并通过一个示例说明：

（1）**初始化**：首先，将输入的句子分词，并将词语依次放入缓冲区。同时，初始化一个空栈用于存储已处理的词语和生成的依存关系。

（2）**转移动作**：通过一系列转移动作逐步构建依存结构。转移动作根据当前栈和缓冲区中的状态以及预定义的转移规则进行选择。常见的转移动作包括：

a. **移进（Shift）**：将缓冲区中的词语移入栈中。

b. **规约（Reduce）**：根据某些条件，将栈顶的一个或多个词语与栈中其他词语之间建立依存关系，并将它们合并为一个词语。

c. **左弧（Left-Arc）**：将栈顶词语与下一个词语之间建立依存关系，并将下一个词语移入栈中。

d. **右弧（Right-Arc）**：将下一个词语与栈顶词语之间建立依存关系，并将栈顶词语移出栈。

（3）**结束条件**：当所有词语都被处理完毕且栈中只剩下一个根节点时，分析过程结束。

（4）**生成依存结构**：根据一系列转移动作生成的依存关系，构建句子依存结构。通常以依存树或依存图的形式展示，其中节点表示词语，边表示依存关系。

示例：“I eat apples”（我吃苹果）。

（1）**初始化**：将词语 “I”、“eat”、“apples” 放入缓冲区，初始化一个空栈。

（2）**转移动作**：

•移进（Shift）：将 “I” 移入栈中。

•移进（Shift）：将 “eat” 移入栈中。

•移进（Shift）：将 “apples” 移入栈中。

•右弧（Right-Arc）：将 “eat” 和 “apples” 之间建立依存关系，表示 “apples” 依存于 “eat”。

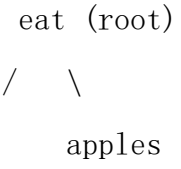
•右弧（Right-Arc）：将“I”和“eat”之间建立依存关系，表示“eat”依存于“I”。

•左弧（Left-Arc）：将“I”和“ROOT”（根节点）之间建立依存关系，表示“I”依存于“ROOT”。

（3）**结束条件**：所有词语都被处理完毕，栈中只剩下一个根节点。

（4）**生成依存结构**：构建依存树，其中“eat”依存于“I”，“apples”依存于“eat”，同时“I”作为根节点。

最终的依存树如下所示：



这个依存树清晰地展示了句子中词语之间的依存关系，帮助我们理解句子的语法结构和含义。

2.2 基于图的依存关系句法分析

基于图的依存句法分析是一种用于分析句子结构的语言学方法，其核心思想是将句子中的单词视为图中的节点，并通过图中的边表示单词之间的依存关系。这种方法可以提供对句子结构的深层理解，包括单词之间的语法关系和句子的语义信息。下面是基于图的依存句法分析的详细步骤，并通过一个示例说明：

（1）构建依存图

在依存句法分析中，首先要根据句子中单词之间的依存关系构建一个有向图。每个单词对应图中的一个节点，而依存关系则通过有向边表示。每条边上通常会带有标签，表示依存关系的类型，例如主谓关系、动宾关系、修饰关系等。构建依存图的过程涉及以下步骤：

•词法分析（Tokenization）：将句子分割成单词或词汇单元，每个单词作为图中的一个节点。

•依存关系抽取：通过语法分析或依存关系抽取算法，确定每个单词与其他单词之间的依存关系，并为边赋予相应的标签。

（2）图分析算法

构建依存图后，需要利用图论中的算法来分析这个图，以解析句子的结构和依存关系。常用的图分析算法包括：

•深度优先搜索（DFS）：从图中的一个节点出发，沿着一条路径尽可能深地探索，直到到达最远的节点，然后回溯到前一个节点继续探索。

•广度优先搜索（BFS）：从图中的一个节点开始，依次访问其所有邻居节点，然后逐层向外扩展。

•最小生成树算法（如 Kruskal 算法、Prim 算法）：用于构建最小生成树，可以在依存图中找到一些重要的依存关系路径。

（3）句法结构解析

通过图分析算法，可以找出句子中单词之间的各种依存关系，如主谓关系、动宾关系、修饰关系等。这些关系可以用于构建句子的句法结构。例如，可以从中心词（如动词或名词）开始，通过遍历依存图中的边，找到与之相关联的修饰词，从而构建出句子的结构。

示例说明：

考虑句子：“The cat chased the mouse”。我们将这个句子转换成依存图：

节点（Nodes）：The, cat, chased, mouse

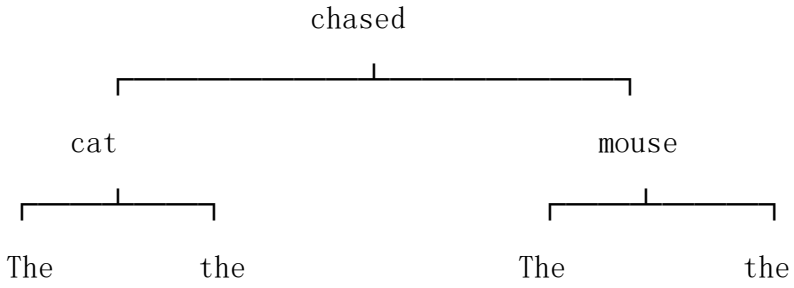
边（Edges）：

chased --> cat（依存关系：主谓关系）

chased --> mouse（依存关系：动宾关系）

cat --> The（依存关系：定中关系）

mouse --> the（依存关系：定中关系）



通过上述依存图，我们可以清楚地看到每个单词之间的依存关系。例如，“chased”依存于“cat”（主谓关系），同时“cat”依存于“The”（定中关系），而“chased”还依存于“mouse”（动宾关系），同时“mouse”依存于“the”（定中关系）。

基于图的依存句法分析可以应用于自然语言处理的许多任务中，包括机器翻译、问答系统、信息抽取等。通过分析句子结构和依存关系，可以更准确地理解句子的含义，从而提高自然语言处理任务的性能和效果。

实践一：基于 HanLP 的成分句法分析

1. 目标

（1）成功安装并调试 HanLP；

（2）能够调用 HanLP_API 完成成分句法分析任务。

2. 项目描述

直接调用 HanLP_API 处理中文句子示例，进行成分句法分析任务，并可视化成分句法分析树。本次实验在基于 Windows 的 Jupyter Notebook 环境中完成，具体依赖为：python=3.9.17，hanlp_restful=0.0.23。

项目实现流程，如图 11 所示。

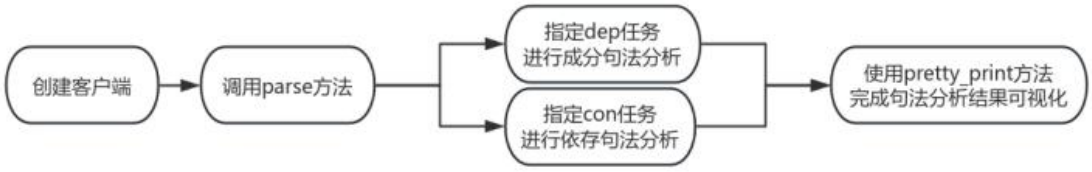


图 11. 基于 HanLP 的成分句法分析流程图

3. 实验过程

3.1 环境配置

在项目路径下打开 Powershell 终端，首先键入“conda create -n env_name python=3.9.17”创建虚拟环境，然后键入“conda activate env_name”激活新创建的虚拟环境，再后键入“pip install hanlp_restful=0.0.23”完成针对于句法分析任务的 HanLP 依赖包安装，最后键入“jupyter notebook”进入到 Jupyter 编辑环境中。

3.2 创建客户端

在 HanLP_API_SyntacticAnalysis.ipynb 文件中，使用 HanLPClient 函数实例化客户端，并指定接口地址及文本类型为中文。具体代码，如下所示。

```
from hanlp_restful import HanLPClient

# 实例化客户端：指定 API 和语言类型
HanLP = HanLPClient('https://www.hanlp.com/api', auth=None, language='zh')
```

3.3 成分句法分析

调用实例化对象 HanLP 的 parse 方法，完成句子“小吴在净月潭徒步的过程中偶遇了小王。”的成分句法分析。具体代码，如下所示。

```
# 成分句法分析：指定了成分句法分析任务 dep
doc = HanLP.parse('小吴在净月潭徒步的过程中偶遇了小王。', tasks='dep')
```

3.4 可视化成分句法分析树

可通过 pretty_print 方法，完成句法分析树的可视化。具体代码，如下所示。

```
# 显示成分句法分析结果：描述了句子结构和成分关系
print(doc.to_conll())
```

可视化结果，如图 12 所示。

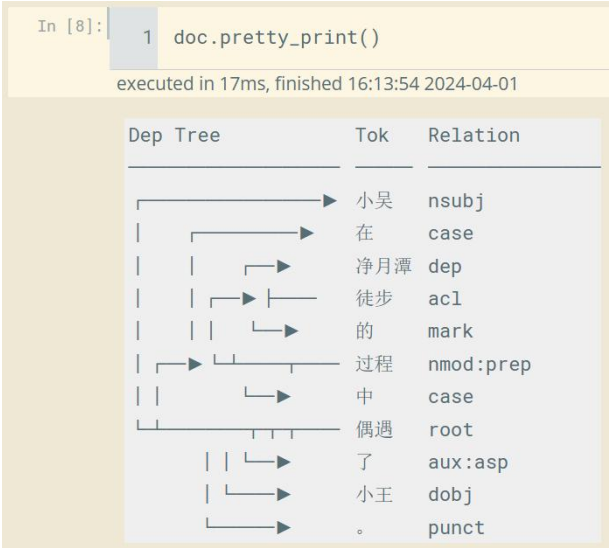


图 12. 成分句法分析结果可视化

其中，句子中不同成分的关系标签解释，如下所示。

```
'''
    “小吴” nsubj：主语
    “在” case：介词，表示位置关系
    “净月潭” dep：依赖成分，介词“在”的宾语
    "徒步" acl：名词短语，表示行为的方式
    “的” mark：标记，用于连接
    “过程” nmod:prep：名词短语“徒步”的介词短语修饰
    “中” case：介词“过程”的位置修饰
    “偶遇” root：句子的核心动词
    “了” aux:asp：助动词，表示动作的完成
    “小王” dobj：动词“偶遇”的直接宾语，表示了动作的对象
    “。” punct：句子的结尾标点
'''
```

4. 实验结论

成分句法分析将句子划分为若干成分，并使用成分句法分析树表示句子的语法结构。然而，由于此类方法没有显式地明确成分间的具体关系，只是简单地将句子分解为若干成分，因此成分句法分析往往不能很好地处理歧义问题。此外，成分句法分析将句子分解为独立的成分，可能会导致部分语法信息的丢失。

实践二：依存句法分析

1. 目标

（1）基于 HanLP 训练依存句法分析模型 Con，用于完成依存句法分析任务；

（2）直接使用 LTP 的预训练模型，完成中文句子示例的分词、词性标注及依存句法分析任务，并可视化依存句法分析图。

2. 项目描述

针对于 HanLP。不仅可以仿照实践一，直接调用 API 完成依存句法分析及结果可视化。还可以使用 HanLP 官方提供的封装实现类，投喂语料数据集 CTB8.0，在本地完成依存句法分析模型 Con 的训练与使用。然而，这种高度封装的实现类，仅提供调整数据及嵌入表示文件的自由度，无法灵活调整模型超参。由此，还可使用来自 LTP 的预训练模型 cws.model、pos.model、parser.model，完成中文句子的分词、词性标注及依存句法分析任务，并自行实现依存句法分析图表示。

调用 HanLP_API 训练依存句法分析模型 Con，主要环境依赖如下：

python=3.7.12；pyhanlp=0.1.85

直接使用 LTP 的预训练模型来完成依存句法分析，主要环境依赖如下：

python=3.6.15；pyltp=0.2.1；matplotlib=3.3.4；networkx=2.5.1

3. 实验过程

3.1 基于 pyHanLP 的依存句法分析

项目实现流程，如图 13 所示：

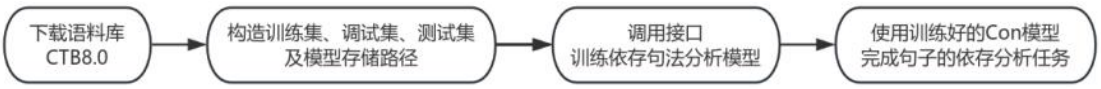


图 13. 基于 pyHanLP 的依存句法分析流程图

3.1.1 下载语料库 CTB8.0

依赖文件操作库 os，编写数据集下载解压及路径载入函数 ensure_data()。具体而言，会验证是否存在指定语料数据，若无则根据链接下载并解压数据，有则返回数据集的存储路径。代码实现及详细解释，如下所示。

```
# 获取存储数据的根目录：.\envs\env_name\Lib\site-packages\pyhanlp\static\data\test
def test_data_path():
    data_path = os.path.join(HANLP_DATA_PATH, 'test')
    if not os.path.isdir(data_path):
        os.mkdir(data_path)
```

```
    return data_path
# 验证是否存在语料库 CTB8.0，如果没有自动下载
def ensure_data(data_name, data_url):
    # 获取根路径
    root_path = test_data_path()
    # 根据根路径和语料库名称，构造目标路径
    dest_path = os.path.join(root_path, data_name)
    # 若目标路径存在，则直接返回该路径，表明语料库已经存在
    if os.path.exists(dest_path):
        return dest_path
    if data_url.endswith('.zip'):
        dest_path += '.zip'
    # 调用 download 函数，下载指定语料库文件到目标路径
    download(data_url, dest_path)
    # 解压并删除下载的压缩文件
    if data_url.endswith('.zip'):
        with zipfile.ZipFile(dest_path, "r") as archive:
            archive.extractall(root_path)
        remove_file(dest_path)
        dest_path = dest_path[:-len('.zip')]
    return dest_path  # 返回目标路径
```

3.1.2 训练依存句法分析模型

使用来自 pyhanlp 的 JClass 函数，通过传入特定训练接口的方式，完成依存句法分析模型的实例化加载。然后，根据语料库 CTB8.0 的元存储路径，构造出训练集、验证集及 Brown 词聚类文件的存储路径，继而传入到实例化对象中完成模型训练。代码实现及详细解释，如下所示。

```
# 从 HanLP 库中导入“KBeamArcEagerDependencyParser”类（封装有依存句法分析实现）
KBeamArcEagerDependencyParser=JClass('com.hankcs.hanlp.dependency.perceptron.parser.KBeamArcEagerDependencyParser')
# 定义用于训练、验证、测试和 CTB8.0 数据集的文件路径
CTB_ROOT = ensure_data("ctb8.0-dep",
"http://file.hankcs.com/corpus/ctb8.0-dep.zip")
CTB_TRAIN = CTB_ROOT + "/train.conll"
CTB_DEV = CTB_ROOT + "/dev.conll"
CTB_TEST = CTB_ROOT + "/test.conll"
# 存储训练好的依存句法分析模型，以便于按需加载使用
CTB_MODEL = CTB_ROOT + "/ctb.bin"
BROWN_CLUSTER = ensure_data("wiki-cn-cluster.txt",
"http://file.hankcs.com/corpus/wiki-cn-cluster.zip")
# 训练依存句法解析器
parser = KBeamArcEagerDependencyParser.train(CTB_TRAIN, CTB_DEV, BROWN_CLUSTER,
CTB_MODEL)
```

实例化类 KBeamArcEagerDependencyParser，封装有基于 ArcEager 转移系统以平均感知机作为分类器的柱搜索依存句法分析器，具体参数说明详见：

KBeamArcEagerDependencyParser (HanLP portable-1.7.7 API) (javadoc.io)

3.1.3 依存句法分析

使用训练好的句法分析模型对象 parser，针对中文句子“小吴在净月潭徒步的过程中偶遇了小王。”，完成依存句法分析。代码实现及详细解释，如下所示。

```
# 使用训练好的解析器，对指定句子完成依存句法分析
doc = parser.parse("小吴在净月潭徒步的过程中偶遇了小王。")
# 依存分析结果
print(doc)
```

3.1.4 实验结果

句法分析结果，如图 14 所示。

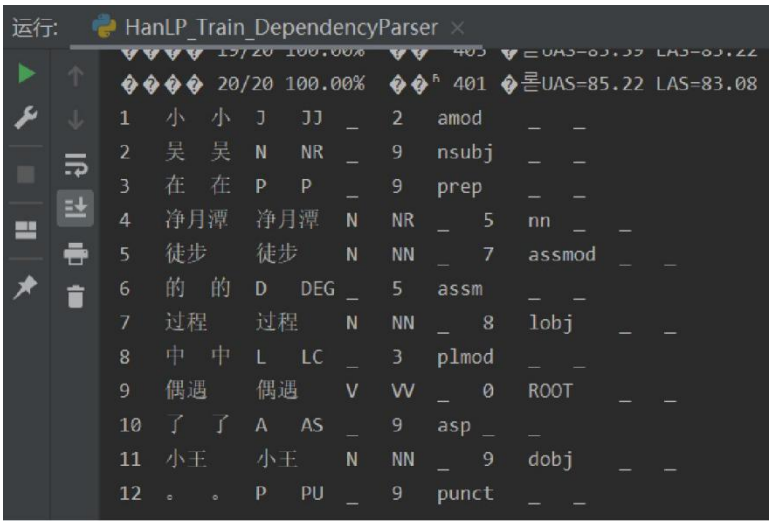


图 14. 基于 HanLP 训练模型的句法分析结果

其中，句子示例不同分词的词性，解释如下所示：

```
'''
    “小吴” NR：人名
    “在” P：介词
    “净月潭” NR：地名
    "徒步" VV：动词
    “的” DEC：连接词
    “过程” NN：名词
    “中” LC：方位词
    “偶遇” VV：动词
    “了” AS：助动词
    “小王” NR：人名
    “。” PU：句号
'''
```

此外，不同分词间的依存关系，解释如下所示：

```
'''
    TOP：表示整个句子的顶层标签
    IP：表示简单句的标签
    NP：表示名词短语的标签
    VP：表示动词短语的标签
    PP：表示介词短语的标签
    CP：表示从句的标签
'''
```

3.2 基于 LTP 的依存句法分析

项目实现流程，如图 15 所示。

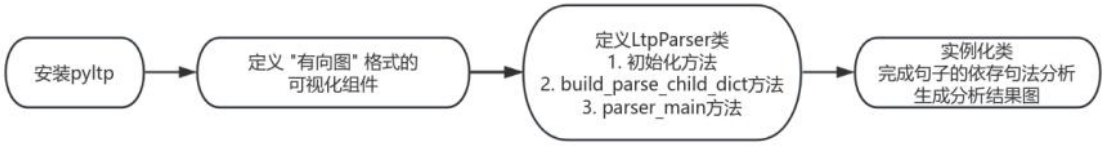


图 15. 基于 LTP 的依存句法分析流程图

3.2.1 安装 pyltp

【参考博文：pyltp 安装教程——保姆级_pyltp wheel 文件-CSDN 博客】

直接使用命令“pip install pyltp”安装 pyltp，会提示无 wheels，导致安装失败。故此，需要手动下载 wheels，放置到“.\envs\Python36\Lib\site-packages”路径下。然后，执行命令“pip install pyltp-0.2.1-cp36-cp36m-win_amd64.whl”即可完成按照。具体操作，如图 16 所示。此外，还需要进入 LTP 官网下载 3.4.0 版本的 SRL 模型，具体详见上述参考博文。

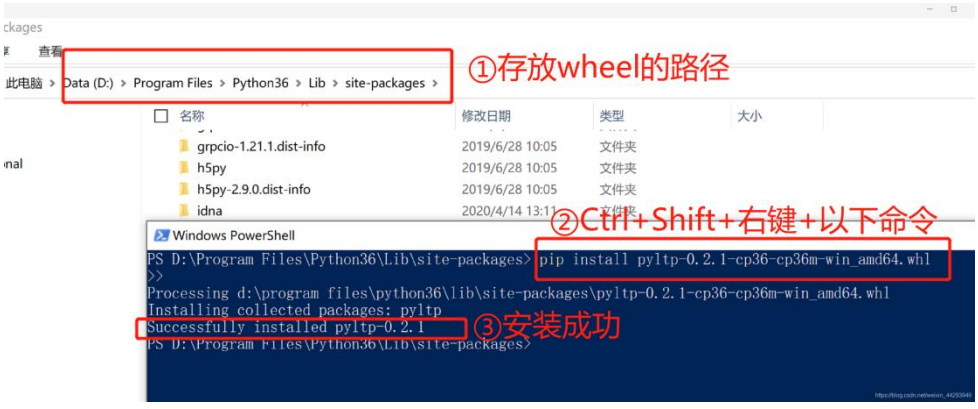


图 16. 安装 pyltp

3.2.2 定义“有向图”形式的可视化组件

依赖 matplotlib 和 networkx，处理依存句法分析结果 child_dict_list 及分词列表 words，从而完成分析结果的有向图表示。代码实现及详细解释，如下所示。

```
def visualize_dependency_tree(words, child_dict_list):
'''
输入1: words = ['小','吴','在','净月潭','徒步',',','偶遇','了','小王','。']
输入2: child_dict_list =
[{},{'ATT':[0]},{'POB':[3]},{},{'SBV':[1],'ADV':[2],'WP':[5]},{},{'ADV': [4],
'RAD':[7],'VOB':[8],'WP':[9]},{},{},{}]
'''
    G = nx.DiGraph()
    for i, word in enumerate(words):  # 添加节点
        G.add_node(word)
    for i, child_dict in enumerate(child_dict_list):  # 添加边
        for relation, children in child_dict.items():
            for child in children:
                G.add_edge(words[i], words[child], label=relation)
    pos = nx.spring_layout(G, k=0.5)  # 自定义布局
    nx.draw(G, pos, with_labels=True, arrows=True)  # 绘制节点和边
    labels = nx.get_edge_attributes(G, 'label')
    nx.draw_edge_labels(G,pos,edge_labels=labels,label_pos=0.5,font_size=8)
    plt.show()
```

3.2.3 定义类 LtpParser

在初始化方法中，需要加载分词模型 cws.model、词性标注模型 pos.model 及依存句法分析模型 parser.model。代码实现及详细解释，如下所示。

```
# 基于 pyltp 预训练模型的依存句法分析实现
class LtpParser:
    def __init__(self):
        LTP_DIR = "./myLTP/ltp_data_v3.4.0"
        self.segmentor = Segmentor()  # 分词
        self.segmentor.load(os.path.join(LTP_DIR, "cws.model"))
        self.postagger = Postagger()  # 词性标注
        self.postagger.load(os.path.join(LTP_DIR, "pos.model"))
        self.parser = Parser()  # 句法依存分析
        self.parser.load(os.path.join(LTP_DIR, "parser.model"))
```

在 build_parse_child_dict 方法中，输入句子的分词及词性标注列表，输出存储每个分词依存父节点及关系类型的字典 child_dict_list。具体而言：首先，使用分词及词性标注器，将句子分词并进行词性标注；然后，根据分词及词性列表，使用依存句法分析模型完成依存句法分析树的建立；再后，遍历依存句法分析树 arcs，提取并使用列表封装每个分词的依存父节点及其关系类型；最后，按输入序列中的分词顺序，使用字典对象 child_dict_list，有序组织分词的依存关系列表。代码实现，如下所示。

```
def build_parse_child_dict(self, words, postags):
'''
输入1：words = ['小','吴','在','净月潭','徒步',',','偶遇','了','小王','。']
输入2：postags = ['a', 'nh', 'p', 'ns', 'd', 'wp', 'v', 'u', 'nh', 'wp']
输出：child_dict_list = [{},{'ATT':[0]},{'POB':[3]},{},{'SBV':[1],'ADV':[2],
WP':[5]},{},{'ADV':[4],'RAD':[7],'VOB':[8],'WP':[9]},{},{},{}]
'''
    child_dict_list = []
    # 获取句子的依存句法分析结果
    arcs = self.parser.parse(words, postags)
    # 提取每个分词的依存父节点 id【-1 表示 ROOT，id 从 0 开始】
    rely_ids = [arc.head - 1 for arc in arcs]
    # 根据 rely_ids 记录每个分词的依存父节点分词
    heads = ['Root' if rely_id == -1 else words[rely_id] for rely_id in rely_ids]
    relations = [arc.relation for arc in arcs]  # 提取依存关系
    # 循环遍历每个分词并构建字典：键为依存关系，值为存储对应依存关系词语索引的列表
    for word_index in range(len(words)):
        child_dict = dict()
        for arc_index in range(len(arcs)):
            if word_index == rely_ids[arc_index]:
                if relations[arc_index] in child_dict:
                    child_dict[relations[arc_index]].append(arc_index)
                else:
                    child_dict[relations[arc_index]] = []
                    child_dict[relations[arc_index]].append(arc_index)
        child_dict_list.append(child_dict)
    return child_dict_list
```

在 parser_main 方法中，需要封装自接收句子输入开始，具体如何调用初始化好的预训练模型及自定义方法，依次完成分词、词性标注及依次句法分析任务。代码实现，如下所示。

```
def parser_main(self, sentence):
    words = list(self.segmentor.segment(sentence))  # 分词
    postags = list(self.postagger.postag(words))  # 词性标注
    child_dict_list = self.build_parse_child_dict(words, postags)  # 依存句法分析
    return words, postags, child_dict_list
```

3.2.4 依存句法分析

在主函数中，调用实例化 parse 对象的 parser_main 方法，传入中文句子示例“小吴在净月潭徒步，偶遇了小王。”即可直接获得分词列表 words、词性标注列表 postags 及分词依存关系字典 child_dict_list。代码实现，如下所示。

```python
# 主函数
if __name__ == '__main__':
    # 实例化句法分析实现类
    parse = LtpParser()
    # 定义待处理的中文句子
    sentence = '小吴在净月潭徒步，偶遇了小王。'
    # 调用 parse 对象中的 parser_main 函数，完成句子的分词、词性标注及依存句法分析任务
    words, postags, child_dict_list = parse.parser_main(sentence)
    # 在控制台打印结果
    print("\n 分词-->len(words) = {0}----words = {1}".format(len(words), words))
    print("\n 词性标注-->postags={0}".format(postags))
    print("\n 依存句法分析-->child_dict_list={0}".format(child_dict_list))
    # 生成依存句法分析结果的有向图表示
    visualize_dependency_tree(words, child_dict_list)
```

3.2.5 实验结果

输出结果，如图 17、18 所示。

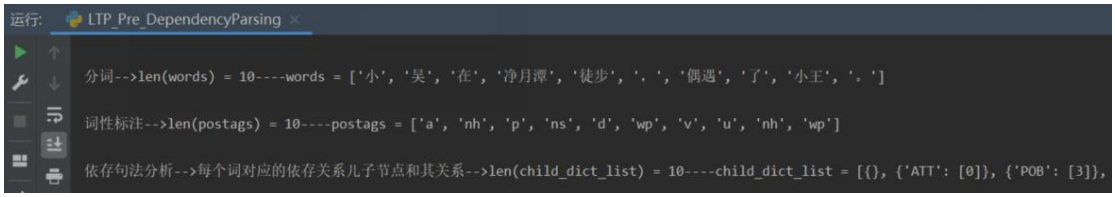


图 17. 分词、词性标注及依存句法分析结果

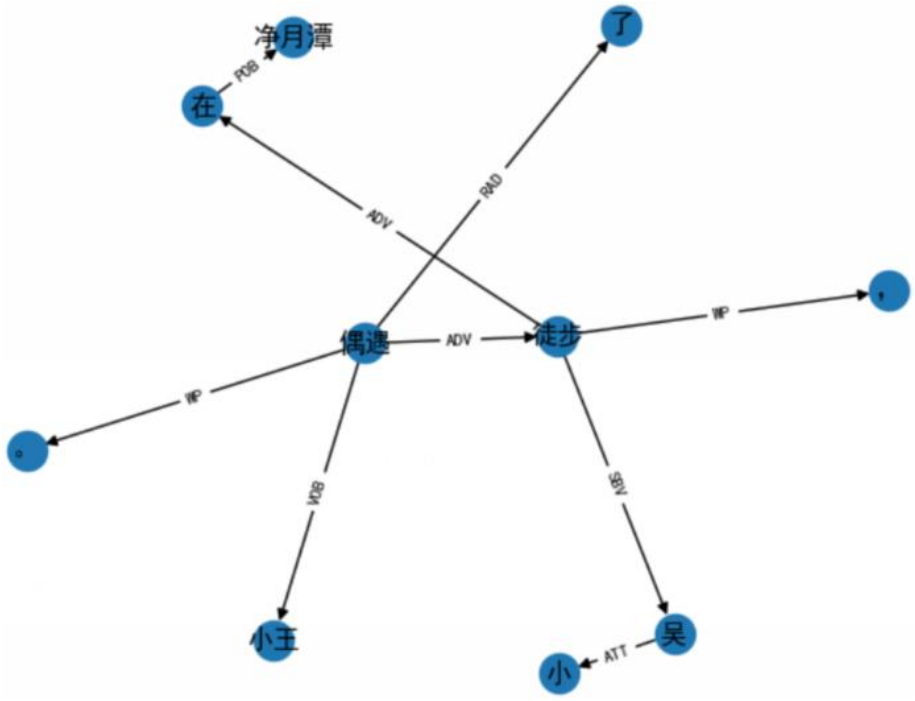


图 18. 依存句法分析图

其中，不同分词间的依次关系标签解释，具体如图 19 所示。

关系类型	Tag	Description	Example
主谓关系	SBV	subject-verb	我送她一束花 (我 <- 送)
动宾关系	VOB	直接宾语，verb-object	我送她一束花 (送 -> 花)
间宾关系	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)
定中关系	ATT	attribute	红苹果 (红 <- 苹果)
状中结构	ADV	adverbial	非常美丽 (非常 <- 美丽)
动补结构	CMP	complement	做完了作业 (做 -> 完)
并列关系	COO	coordinate	大山和大海 (大山 -> 大海)
介宾关系	POB	preposition-object	在贸易区内 (在 -> 内)
左附加关系	LAD	left adjunct	大山和大海 (和 <- 大海)
右附加关系	RAD	right adjunct	孩子们 (孩子 -> 们)
独立结构	IS	independent structure	两个单句在结构上彼此独立
核心关系	HED	head	指整个句子的核心

图 19. 依存关系标签解释

4. 实验结论

依存句法分析，则通过考虑句子中分词间依存关系的方式，揭示了句子结构及其语法规则。分析结果可以使用有向图来表示，其中，图中的节点指代句子中的分词，图中的有向边指代分词间的依存关系类型。相较于成分句法分析，此类方法能够更直观地表示句子的语法结构，具备更强的语义解释性。然而，存在难以处理长距离依存关系及信息丢失等问题。