

目录

C O N T E N T

一 . 概述

二 . 短语结构句法分析

三 . 依存结构句法分析

四 . 总结

五 . 实验



目录

一、概述

1.1 定义

1.2 句法分析常见任务

1.3 常见方法





1.1 定义

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。





1.2 句法分析常见任务

- (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)。





1.3 常见方法

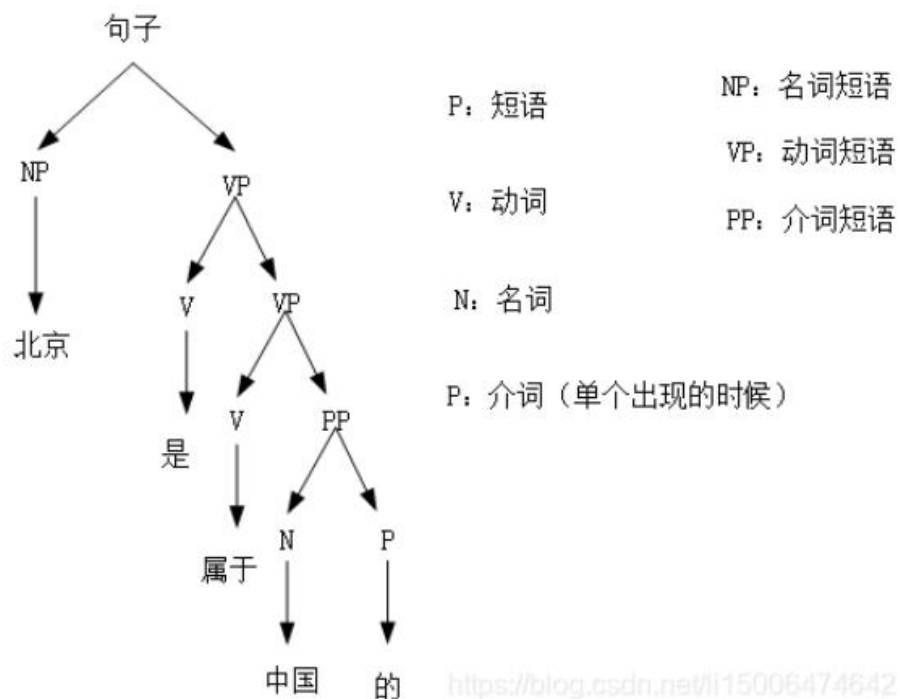
- 句法分析分为短语结构分析和依存关系分析。
- 以获取整个句子的句法结构或者完全短语结构为目的的句法分析，被称为成分结构分析（constituent structure parsing）或者短语结构分析（phrase structure parsing）；另外一种是以获取局部成分为目的的句法分析，被称为依存分析（dependency parsing）。





1.3 常见方法

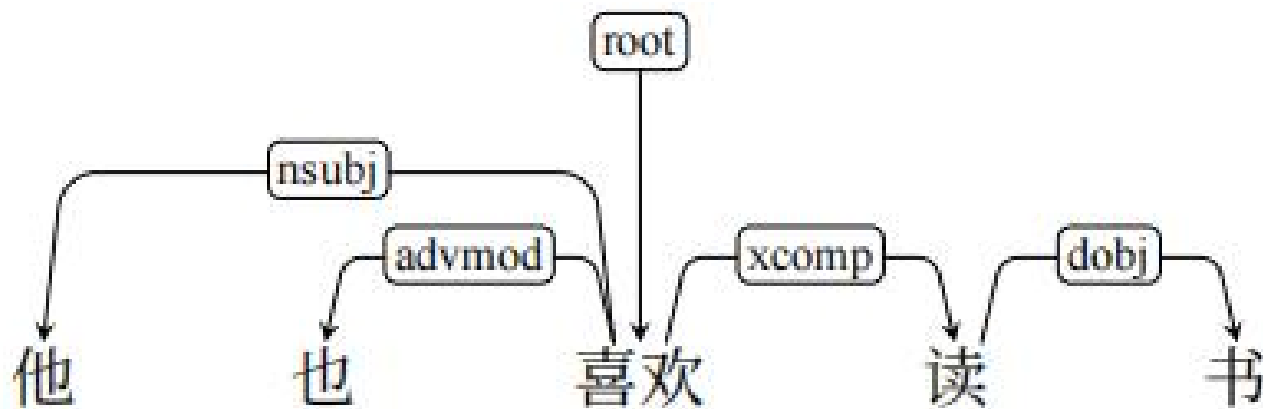
在短语结构句法表示中，S表示起始符号，NP、VP和PP分别表示名词短语、动词短语和介词短语。





1.3 常见方法

在依存结构句法表示中，sub和obj分别表示主语和宾语，root表示虚拟根节点，其指向整个句子的核心谓词。





目录

二、短语结构句法分析

2.1 概述

2.2 任务

2.3 面临的问题

2.4 基本方法





2.1 概述

短语结构句法分析，识别句子的主语、谓语、宾语、定语等成分，判断其结构是否符合给定的语法，并分析各成分之间的关系。句法结构可由句法分析树表示。句法分析任务的过程由句法结构分析器完成。





2.2 任务

- (1) 对输入句子中词法和结构等方面的信息歧义消除;
- (2) 对输入句子进行内部结构的分析, 如上下文关系、成分构成等。





2.3 面临的问题

短语结构句法分析面临的主要困难是句法结构消歧问题，即现有的方法无法很好地识别和消除句法结构的歧义。通过短语结构句法分析，我们能够分析出语句的主干，以及各成分间关系。对于复杂语句，仅仅通过词性分析，不能得到正确的语句成分关系。





2.4 基本方法

短语结构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。





2.4.1 基于统计的句法分析

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。

其中最常用的算法为基于概率上下文无关文法（PCFG）的短语结构句法分析方法。





2.4.2 上下文无关文法

上下文无关文法 G 包含以下四个部分:

- 1) **终结符集合 P** : 与语言中词汇对应的符号, 用 u, v, w 等小写罗马字母表示;
例如: 上海, walk
- 2) **非终结符集合 N** : 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** : 语法中指定的初始符, 通常用 S 表示;
- 4) **规则集合 R** : 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: $S \rightarrow NP VP$, $NP \rightarrow Det AdjN$, $Det \rightarrow the$





2.4.2 上下文无关文法

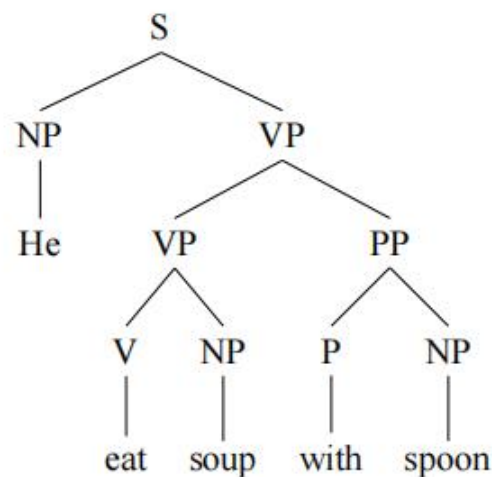
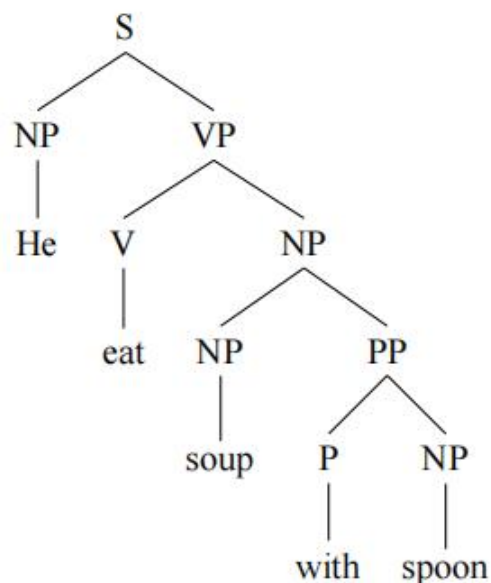
当给定一个句子时，我们可以按照从左到右的顺序来解析语法。例如，句子“I am happy”就可以表示为(S (NP (DT I) (NN am)) (VP happy))。





2.4.3 概率分布的上下文无关语法

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，如下图所示。



句子“He eat soup with spoon”的成分句法树





2.4.3 概率分布的上下文无关语法

基于概率上下文无关文法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。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：

$$A \rightarrow \alpha, p$$

其中 A 为非终结符， α 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。





2.4.3 概率分布的上下文无关语法

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:

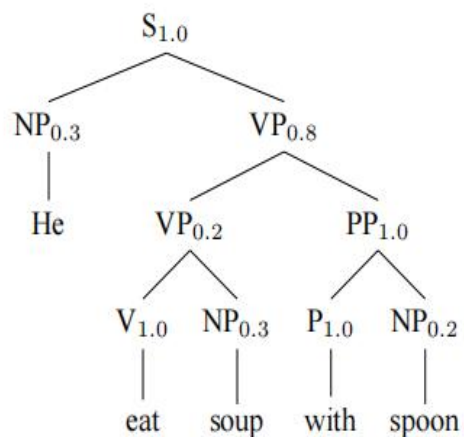
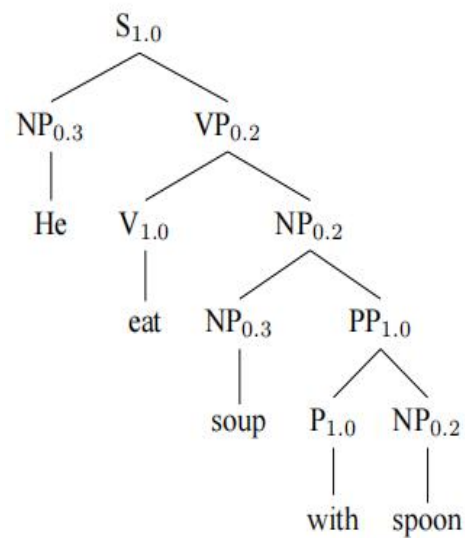
$$P(T) = \prod_{i=1}^m P(A_i \rightarrow \alpha)$$





2.4.3 概率分布的上下文无关语法

对于句子 “He eat soup with spoon” 的两种可能的句法结构包含概率的形式如图所示：



根据句法树概率计算公式，如图所示的两个句法树的概率分别为：

$$P(\text{TLeft}) = 1.0 \times 0.3 \times 0.2 \times 1.0 \times 0.2 \times 0.3 \times 1.0 \times 1.0 \times 0.2 = 0.00072$$

$$P(\text{TRight}) = 1.0 \times 0.3 \times 0.8 \times 0.2 \times 1.0 \times 1.0 \times 0.3 \times 1.0 \times 0.2 = 0.00288$$

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





目录

三、依存结构句法分析

3.1 概述

3.2 任务

3.3 面临的问题

3.4 基本方法





3.1 概述

依存结构句法分析 (Dependency Parsing) 是一种句法分析方法，旨在分析句子中词语之间的依存关系，从而揭示句子的结构。在依存结构句法分析中，句子被视为一个词语之间相互依存的网络，其中每个词语（或标记）都有一个中心词，其余的词语依赖于中心词。依存关系通常以有向弧线表示，从依存词指向中心词。





3.2 任务

- 1) 确定每个单词在句子中的语法功能，例如主语、宾语、定语等。
- 2) 确定每个单词的父节点，即它在句子中的直接修饰词或者被修饰的词。
- 3) 确定每个单词与其他单词之间的语法关系，例如主谓关系、动宾关系、定中关系等。
- 4) 建立依存树或依存图，表示单词之间的依存关系。





3.3 面临的问题

- 1) 歧义问题: 有些句子中存在歧义, 例如多义词、语序不同但语法结构相同的句子等, 这些都会给依存关系句法分析带来挑战。
- 2) 标注问题: 依存关系句法分析需要用到标注数据来训练模型, 但标注数据的准确性和质量会影响模型的表现, 同时标注数据的收集和标注也需要耗费大量时间和人力资源。
- 3) 多语言问题: 不同语言之间的语法结构差异较大, 因此依存关系句法分析需要为不同语言开发不同的模型, 并且需要解决跨语言的问题。
- 4) 对于大规模的数据集和复杂的句子, 依存关系句法分析需要具备高效和可扩展的能力, 否则会影响模型的性能和实用性。

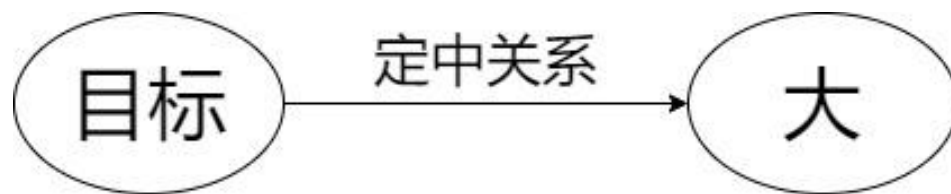




3.4 基本方法

依存关系句法理论认为词与词之间存在主从关系，这是一种二元不等价的关系。在句子中，如果一个词修饰另一个词，则称修饰词为从属词（dependent），被修饰的词语成为支配词（head），两者之间的语法关系称为依存关系（dependency relation）。

比如句子“大目标”中形容词与名词“目标”之间的依存关系，如图所示：



其中，图中的箭头由支配词指向从属词。

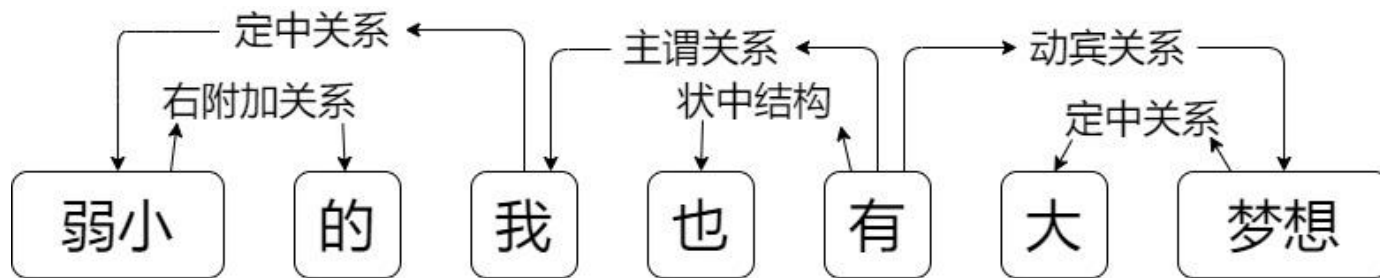




3.4 基本方法

依存句法树

将一个句子中所有词语的依存关系以有向边的形式表示出来就会得到一棵树，称为依存句法树（dependency parse tree）。例如，句子“弱小的我也有大梦想”的依存语法树可表示为





目录

四、总结





4 总结

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规则。然而，由于此类方法没有显式地明确成分间的具体关系，使其不能很好地处理歧义问题。

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。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规则，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和信息损失等问题。





目录

五、实验

5.1 实验一：基于HanLP的成分句法分析

5.2 实验二：基于pyHanLP的依存句法分析

5.3 实验三：基于LTP的依存句法分析





目录

5.1、实验一：基于HanLP的成分句法分析

5.1.1. 目标

5.1.2. 项目描述

5.1.3. 实验过程





5.1.1 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。





5.1.2 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 Windows11 的 Jupyter Notebook 环境中完成，主要依赖为：
`python=3.9.17, hanlp_restful=0.0.23。`





5.1.3 实验过程

(1) 环境配置

在项目路径下打开Powershell终端，主要命令如下：

```
conda create -n env_name python=3.9.17
```

```
conda activate env_name
```

```
pip install hanlp_restful=0.0.23
```

```
jupyter notebook
```





5.1.3 实验过程

(2) 创建客户端

```
In [5]: 1 from hanlp_restful import HanLPClient  
        2 HanLP = HanLPClient('https://www.hanlp.com/api', auth=None, language='zh')
```

executed in 9ms, finished 16:12:01 2024-04-01





5.1.3 实验过程

(3) 成分句法分析

```
In [6]: 1 doc = HanLP.parse('小吴在净月潭徒步的过程中偶遇了小王。', tasks='dep')
        2 print(doc)
```

executed in 1.39s, finished 16:13:22 2024-04-01

```
{
  "tok/fine": [
    ["小吴", "在", "净月潭", "徒步", "的", "过程", "中", "偶遇", "了", "小王", "。"]
  ],
  "dep": [
    [[8, "nsubj"], [6, "case"], [4, "dep"], [6, "acl"], [4, "mark"], [8, "nmod:prep"], [6, "case"], [0, "root"], [8, "aux:asp"],
    [8, "dobj"], [8, "punct"]]
  ]
}
```





5.1.3 实验过程

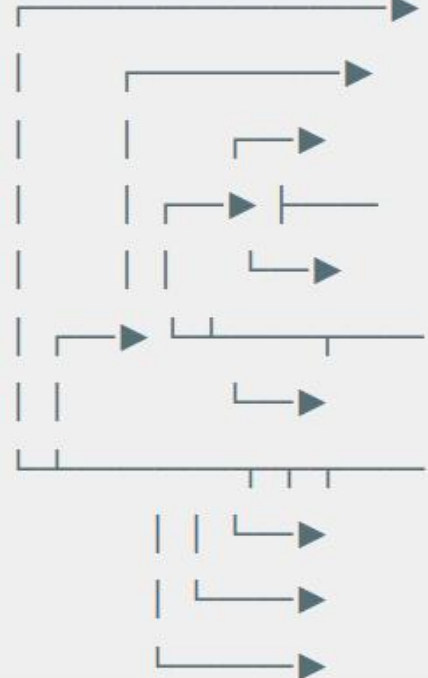
(4) 可视化成分句法分析树

- “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**: “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**: “偶遇”的直接宾语, 表示动作的对象
“。” **punct**: 句子的结尾标点

In [8]:

```
1 doc.pretty_print()
```

executed in 17ms, finished 16:13:54 2024-04-01

Dep Tree	Tok	Relation
	小吴	nsubj
	在	case
	净月潭	dep
	徒步	acl
	的	mark
	过程	nmod:prep
	中	case
	偶遇	root
	了	aux:asp
	小王	dobj
	。	punct



目录

5.2、实验二：基于pyHanLP的依存句法分析

5.2.1. 项目描述

5.2.2. 实验过程



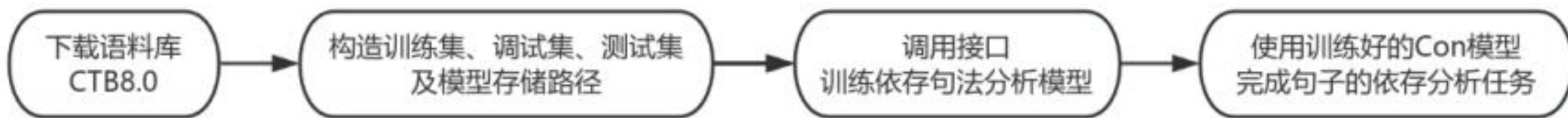


5.2.1 项目描述

仿照实验一，直接调用API接口完成依存句法分析及结果可视化。

使用HanLP官方提供的训练接口，投喂语料数据集CTB8.0，在本地完成依存句法分析模型的训练。调用HanLP_API训练依存句法分析模型，所需的环境依赖如下：

`python=3.7.12; pyhanlp=0.1.85`





5.2.2 实验过程

(1) 下载语料库CTB8.0

```
# 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)
```





5.2.2 实验过程

```
    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
```





5.2.2 实验过程

(2) 训练依存句法分析模型

```
# 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)
```





5.2.2 实验过程

(3) 依存句法分析

```
# 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)
```





5.2.2 实验过程

(4) 实验结果

“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

运行:  HanLP_Train_DependencyParser



197/20

100.00%

20/20

100.00%

401

UAS=85.22

LAS=83.08

1

小

小

J

JJ

—

2

吴

吴

N

NR

—

3

在

在

P

P

—

4

净月潭

净月潭

N

NR

—

5

徒步

徒步

N

NN

—

6

的

的

D

DEG

—

7

过程

过程

N

NN

—

8

中

中

L

LC

—

9

偶遇

偶遇

V

VV

—

10

了

了

A

AS

—

11

小王

小王

N

NN

—

12

。

。

P

PU

—

2

amod

—

—

9

nsubj

—

—

9

prep

—

—

5

nn

—

—

7

assmod

—

—

5

assm

—

—

8

lobj

—

—

3

plmod

—

—

0

ROOT

—

—

9

asp

—

—

9

dobj

—

—

9

punct

—

—





5.2.3 实验过程

(5) 备注

“**amod**”用于表示形容词修饰名词的关系。例如，在句子“big house”中，“big”是形容词，修饰名词“house”，这种修饰关系可以用“amod”来表示。

“**nsubj**”用于表示主语和谓语之间的依存关系。例如，在句子“John eats apples”中，“John”是动词“eats”的主语，因此“John”与“eats”之间的关系可以用“nsubj”标签表示。

“**prep**”用于表示介词短语（PP）与其修饰的名词、动词或形容词之间的依存关系。例如，在句子“The book is on the table”中，“on the table”是介词短语，修饰名词“table”，表示书在桌子上。这里的关系可以用“prep”标签来表示。





目录

5.3、实验三：基于LTP的依存句法分析

5.3.1. 目标

5.3.2. 项目描述

5.3.3. 实验过程





5.3.1 目标

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。

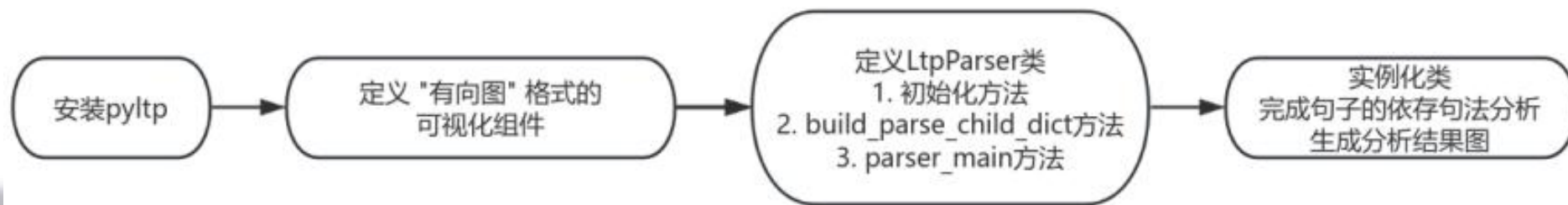




5.3.2 项目描述

使用来自LTP的预训练模型`cws.model`、`pos.model`、`parser.model`，完成中文句子示例的分词、词性标注及依存句法分析任务，并自行实现依存句法分析图表示。直接使用LTP的预训练模型来完成依存句法分析，所需的环境依赖如下：

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



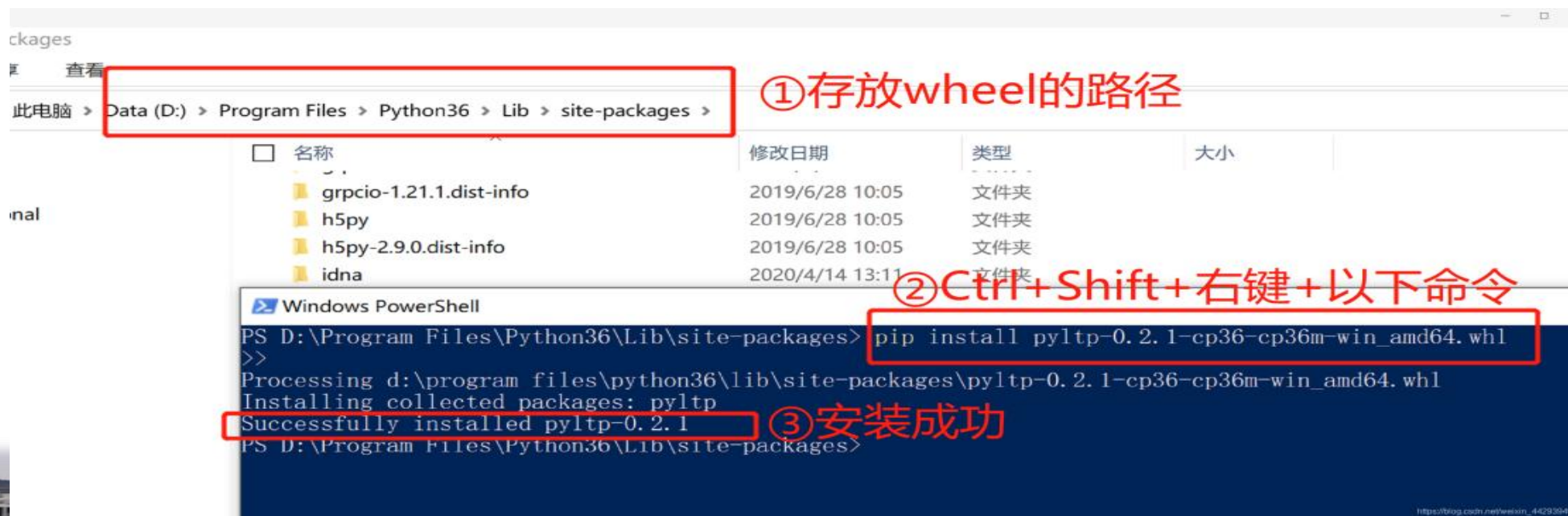


5.3.3 实验过程

(1) 安装pyltp

无法直接使用命令“pip install pyltp”安装pyltp。故此，需要手动下载特定版本的wheels。

https://blog.csdn.net/weixin_44293949/article/details/108631514?spm=1001.2014.3001.5506





5.3.3 实验过程

(2) 定义“有向图”形式的可视化组件

```
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()
```





5.3.3 实验过程

(3) 定义类LtpParser

```
# 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"))
```





5.3.3 实验过程

```
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
```





5.3.3 实验过程

(4) 依存句法分析

在主函数中，实例化自定义类LtpParser，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。





5.3.3 实验过程

(4) 依存句法分析

```
# 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)
```




5.3.3 实验过程

(5) 实验结果

分词、词性标注及依存句法分析结果

运行: LTP_Pre_DependencyParsing ×

分词-->len(words) = 10----words = ['小', '吴', '在', '净月潭', '徒步', ',', ' ', '偶遇', '了', '小王', '。']

词性标注-->len(postags) = 10----postags = ['a', 'nh', 'p', 'ns', 'd', 'wp', 'v', 'u', 'nh', 'wp']

依存句法分析-->每个词对应的依存关系儿子节点和其关系-->len(child_dict_list) = 10----child_dict_list = [{}, {'ATT': [0]}, {'POB': [3]},

“小吴” NR: 人名

“的” DEC: 连接词

“了” AS: 助动词

“在” P: 介词

“过程” NN: 名词

“小王” NR: 人名

“净月潭” NR: 地名

“中” LC: 方位词

“。” PU: 句号

“徒步” VV: 动词

“偶遇” VV: 动词

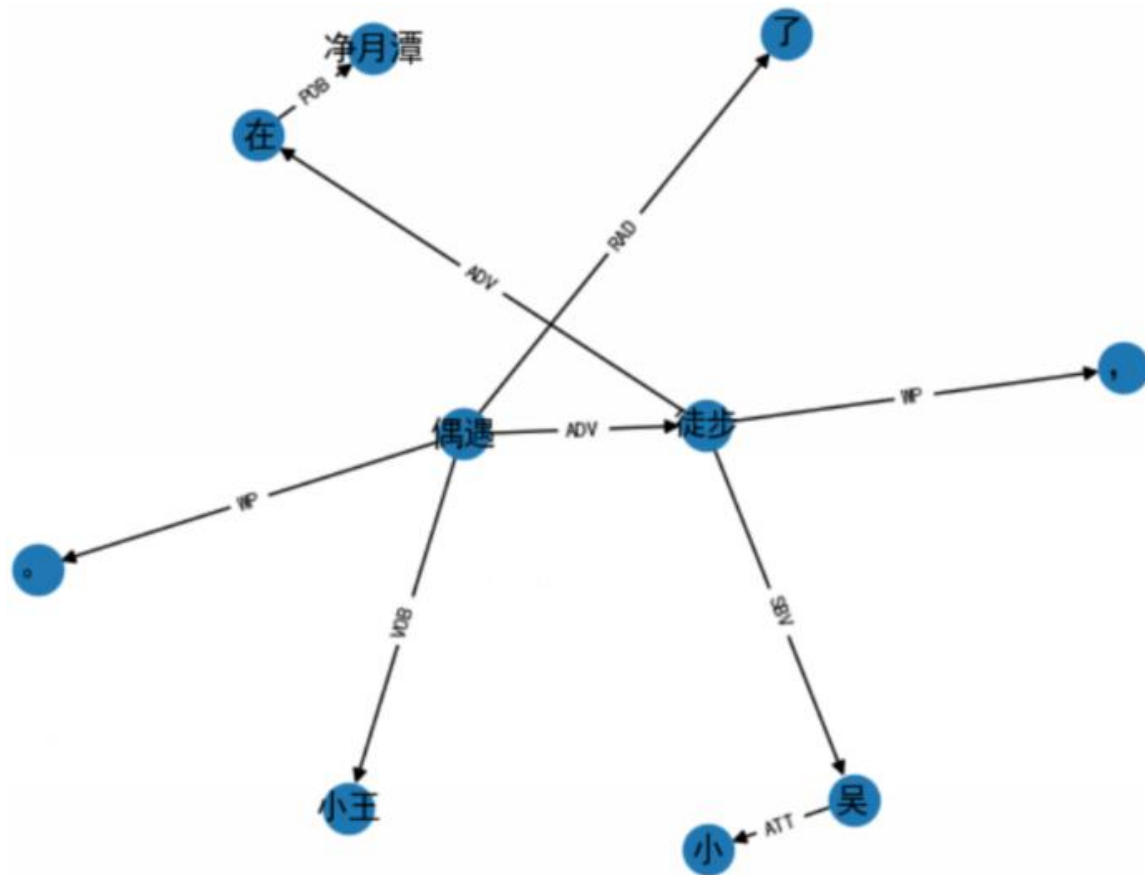




5.3.3 实验过程

(5) 实验结果

依存句法分析图：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。



5.3.3 实验过程

(5) 备注

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





Thank you !