jieba分词用法以及原理

Jieba库是一个开源库 使用的话就**pip install jieba**

分词:

有4种模式，最常用的还是前3种:

1. **精确模式**：**就是把一段文本精确地切分成若干个中文单词，若干个中文单词之间经过组合，就精确地还原为之前的文本。其中**不存在冗余单词 。
2. **全模式**：**将一段文本中所有可能的词语都扫描出来，可能有一段文本它可以切分成不同的模式，或者有不同的角度来切分变成不同的词语，在全模式下，Jieba库会将各种不同的组合都挖掘出来。分词后的信息再组合起来**会有冗余，不再是原来的文本 。
3. **搜索引擎模式**：在精确模式基础上，对发现的那些长的词语，我们会对它再次切分，进而适合搜索引擎对短词语的索引和搜索。**也有冗余。**
4. **paddle模式**：利用PaddlePaddle深度学习框架，训练序列标注（双向GRU）网络模型实现分词。同时支持词性标注。paddle模式使用需安装paddlepaddle-tiny，`pip install paddlepaddle-tiny==1.6.1`。目前paddle模式支持jieba v0.40及以上版本。jieba v0.40以下版本，请升级jieba，`pip install jieba --upgrade` 。

算法:

- 基于前缀词典实现高效的词图扫描，生成句子中汉字所有可能成词情况所构成的有向无环图 (DAG)
- 采用了**动态规划查找最大概率路径, 找出基于词频的最大切分组合**
- 对于**未登录词，采用了基于汉字成词能力的 HMM 模型，使用了 Viterbi 算法**

参数:

`jieba.cut` 方法接受四个输入参数:

- 需要分词的字符串；
- `cut_all` 参数用来控制是否采用全模式；
- `HMM` 参数用来控制是否使用 HMM 模型；
- `use_paddle` 参数用来控制是否使用paddle模式下的分词模式，paddle模式采用延迟加载方式，通过enable_paddle接口安装paddlepaddle-tiny，并且import相关代码；

`jieba.cut_for_search` 方法接受两个参数：

- 需要分词的字符串；
- 是否使用 HMM 模型。该方法适合用于搜索引擎构建倒排索引的分词，粒度比较细

待分词的字符串可以是 unicode 或 UTF-8 字符串、GBK 字符串。注意：不建议直接输入 GBK 字符串，可能无法预料地错误解码成 UTF-8

`jieba.cut` 以及 `jieba.cut_for_search` 返回的结构都是一个可迭代的 generator，可以使用 for 循环来获得分词后得到的每一个词语(unicode)，或者用

`jieba.lcut` 以及 `jieba.lcut_for_search` 直接返回 list

`jieba.Tokenizer(dictionary=DEFAULT_DICT)` 新建自定义分词器，可用于同时使用不同词典。`jieba.dt` 为默认分词器，所有全局分词相关函数都是该分词器的映射。

添加自定义词典：

- 开发者可以指定自己自定义的词典，以便包含 jieba 词库里没有的词。虽然 jieba 有新词识别能力，但是自行添加新词可以保证更高的正确率
- 用法： `jieba.load_userdict(file_name)` ， file_name 为文件类对象或自定义词典的路径
- 词典格式和 `dict.txt` 一样，一个词占一行；**每一行分三部分：词语、词频（可省略）、词性（可省略）**，用空格隔开，顺序不可颠倒。`file_name` 若为路径或二进制方式打开的文件，则文件必须为 UTF-8 编码。
  例子：

  ```
  创新办 3 i
  云计算 5
  凯特琳 nz
  台中
  ```

- 词频省略时使用自动计算的能保证分出该词的词频。

调整词典：

- 使用 `add_word(word, freq=None, tag=None)` 和 `del_word(word)` 可在程序中动态修改词典。
- 使用 `suggest_freq(segment, tune=True)` 可调节单个词语的词频，使其能（或不能）被分出来。使用suggest_freq()调整某个词语的词频，使得其在设置的词频高是能分出，词频低时不能分出。
- 注意：自动计算的词频在使用 HMM 新词发现功能时可能无效。

```
# 1 使用del_word()使得某个词语不会出现
print('/'.join(jieba.cut('如果放到post中将出错。', HMM=False)))
如果/放到/post/中将/出错/。

jieba.del_word("中将")
print('/'.join(jieba.cut('如果放到post中将出错。', HMM=False)))
如果/放到/post/中/将/出错/。

# 2 使用add_word()添加新词到字典中
print('/'.join(jieba.cut('「台中」正确应该不会被切开', HMM=False)))
「/台/中/」/正确/应该/不会/被/切开

jieba.add_word("台中")
print('/'.join(jieba.cut('「台中」正确应该不会被切开', HMM=False)))
「/台中/」/正确/应该/不会/被/切开

# 3 使用suggest_freq()调整某个词语的词频，使得其在设置的词频高是能分出，词频低时不能分出
jieba.suggest_freq('台中', True)
69
print('/'.join(jieba.cut('「台中」正确应该不会被切开', HMM=False)))
「/台中/」/正确/应该/不会/被/切开
```

关键词提取：

将文本中最能表达文本含义的词语抽取出来

(1)基于TF-IDF的关键词抽取算法

目标是获取文本中词频高，也就是TF大的，且语料库其他文本中词频低的，也就是IDF大的。这样的词可以作为文本的标志，用来区分其他文本。

API函数：

- `jieba.analyse.extract_tags(sentence, topK=20, withWeight=False, allowPOS=())`
  - `sentence` 为待提取的文本
  - `topK` 为返回几个 TF/IDF 权重最大的关键词，默认值为 20
  - `withWeight` 为是否一并返回关键词权重值，默认值为 False
  - `allowPOS` 仅包括指定词性的词，默认值为空，即不筛选
- `jieba.analyse.TFIDF(idf_path=None)` ，新建 TFIDF 实例，`idf_path` 为 IDF 频率文件

(2)基于TextRank的关键词抽取算法

先将文本进行分词和词性标注，将特定词性的词（比如名词）作为节点添加到图中。

1. 出现在一个窗口中的词语之间形成一条边，窗口大小可设置为2~10之间，它表示一个窗口中有多少个词语。
2. 对节点根据入度节点个数以及入度节点权重进行打分，入度节点越多，且入度节点权重大，则打分高。
3. 然后根据打分进行降序排列，输出指定个数的关键词。

API函数

- `jieba.analyse.textrank(sentence, topK=20, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v'))` 直接使用，接口相同，注意默认过滤词性。
- `jieba.analyse.TextRank()` 新建自定义 TextRank 实例

词性标注:

利用 `jieba.posseg` 模块来进行词性标注，会给出分词后每个词的词性。词性标示兼容ICTCLAS 汉语词性标注集，可查阅网站

API函数

- `jieba.posseg.POSTokenizer(tokenizer=None)` 新建自定义分词器，`tokenizer` 参数可指定内部使用的 `jieba.Tokenizer` 分词器。`jieba.posseg.dt` 为默认词性标注分词器。
- 标注句子分词后每个词的词性，采用和 ictclas 兼容的标记法。
- 除了jieba默认分词模式，提供paddle模式下的词性标注功能。paddle模式采用延迟加载方式，通过 `enable_paddle()` 安装 `paddlepaddle-tiny`，并且import相关代码；

Paddle模式词性标注对应表如下：

paddle模式词性和专名类别标签集合如下表，其中词性标签 24 个（小写字母），专名类别标签 4 个（大写字母）。

标签	含义	标签	含义	标签	含义	标签	含义
n	普通名词	f	方位名词	s	处所名词	t	时间
nr	人名	ns	地名	nt	机构名	nw	作品名
nz	其他专名	v	普通动词	vd	动副词	vn	名动词
a	形容词	ad	副形词	an	名形词	d	副词
m	数量词	q	量词	r	代词	p	介词
c	连词	u	助词	xc	其他虚词	w	标点符号
PER	人名	LOC	地名	ORG	机构名	TIME	时间

并行分词：

将文本按行分隔后，每行由一个jieba分词进程处理，之后进行归并处理，输出最终结果。这样可以大大提高分词速度。

原理：将目标文本按行分隔后，把各行文本分配到多个 Python 进程并行分词，然后归并结果，从而获得分词速度的可观提升

基于 python 自带的 multiprocessing 模块，目前暂不支持 Windows

用法：

- `jieba.enable_parallel(4)` # 开启并行分词模式，参数为并行进程数
- `jieba.disable_parallel()` # 关闭并行分词模式

注意：并行分词仅支持默认分词器 `jieba.dt` 和 `jieba.posseg.dt`。

Tokenize:返回词语在原文的起止位置

注意，输入参数只接受 unicode

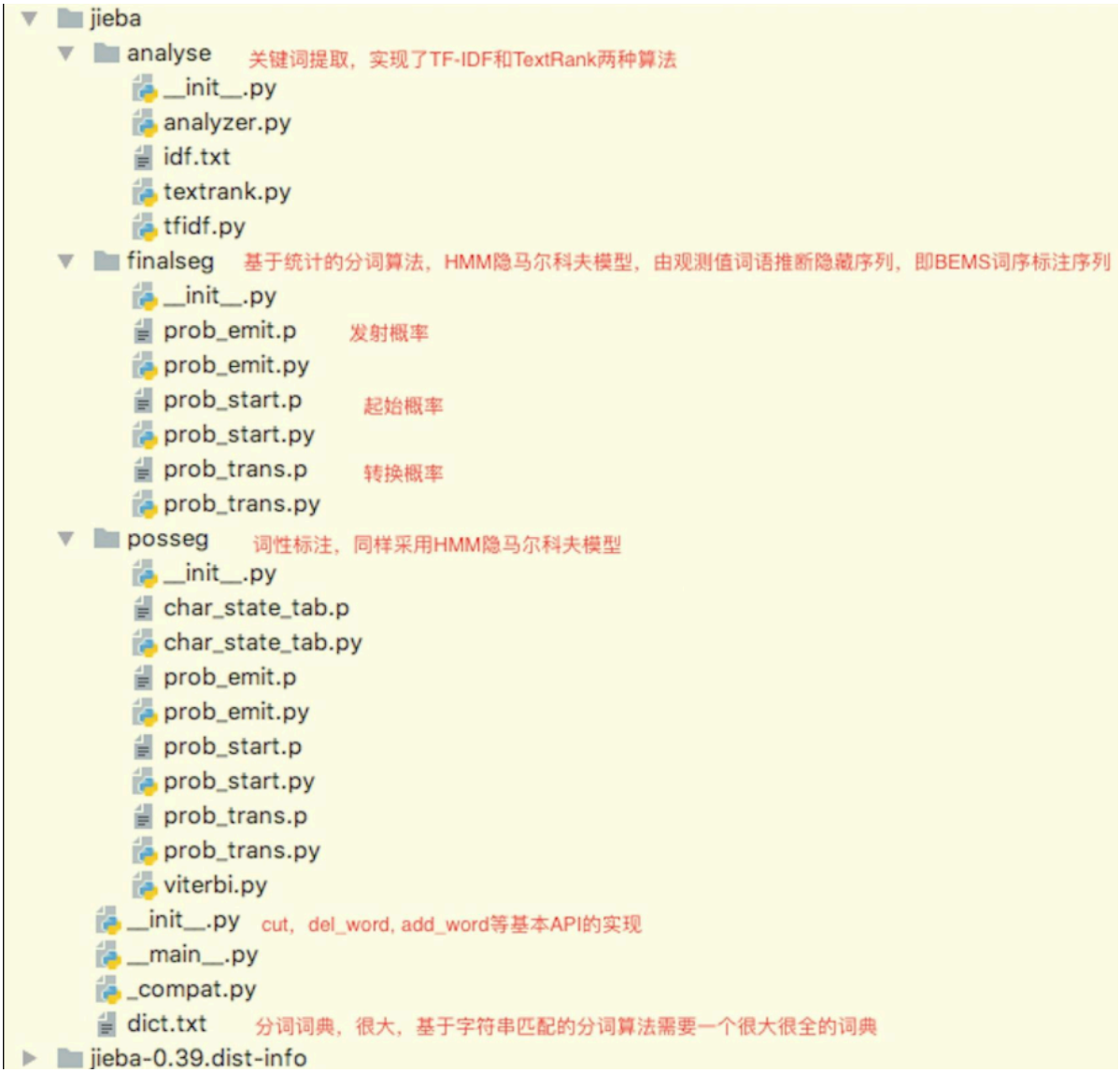
默认模式：

```
result = jieba.tokenize(u'永和服装饰品有限公司')
for tk in result:
    print("word %s\t\t start: %d \t\t end:%d" % (tk[0],tk[1],tk[2]))

# 输出为
word 永和                start: 0                end:2
word 服装                start: 2                end:4
word 饰品                start: 4                end:6
word 有限公司            start: 6                end:10
```

搜索模式

```
result = jieba.tokenize(u'永和服装饰品有限公司', mode='search')
for tk in result:
    print("word %s\t\t start: %d \t\t end:%d" % (tk[0],tk[1],tk[2]))

# 输出为
word 永和                start: 0                end:2
word 服装                start: 2                end:4
word 饰品                start: 4                end:6
word 有限                start: 6                end:8
word 公司                start: 8                end:10
word 有限公司            start: 6                end:10
```

jieba分词源码结构：



主要的模块如下

1. 基本API的封装，在Tokenizer类中，相当于一个外观类。如`cut` `del_word` `add_word` `enable_parallel initialize` 等
2. 基于字符串匹配的分词算法，包含一个很大很全的词典，即`dict.txt`文件
3. 基于统计的分词算法，实现了HMM隐马尔科夫模型。jieba分词使用了字符串分词和统计分词，结合了二者的优缺点。

4. 关键词提取，实现了TFIDF和TextRank两种无监督学习算法
5. 词性标注，实现了HMM隐马尔科夫模型和viterbi算法

jieba原理分析：

jieba分词综合了**基于字符串匹配**的算法和**基于统计**的算法，其分词步骤为

1. 初始化。加载词典文件，获取每个词语和它出现的词数

   初始化可以简单理解为，**读取词典文件，构建词语-词数键值对，方便后面步骤中查词典，也就是字符串匹配。**

2. 切分短语。利用正则，将文本切分为一个个语句，之后对语句进行分词

   首先进行将语句转换为UTF-8或者GBK。
   然后根据用户指定的模式，设置cut的真正实现。
   然后根据正则，将输入文本分为一个个语句。
   最后遍历语句，对每个语句单独进行分词

3. 构建DAG。通过字符串匹配，构建所有可能的分词情况的有向无环图，也就是DAG

   主体步骤如下：

   1. 得到语句的有向无环图DAG
   2. 动态规划构建Route，计算从语句末尾到语句起始，DAG中每个节点到语句结束位置的最大路径概率，以及概率最大时节点对应词语的结束位置
   3. 遍历每个节点的Route，组装词语组合。
   4. 如果词语不在字典中，也就是新词，使用HMM隐马尔科夫模型进行分割
   5. 通过yield将词语逐个返回。

   下面我们来看构建DAG的过程。先遍历一个个切分好的短语，对这些短语来进行分词。首先要构建短语的有向无环图DAG。查词典进行字符串匹配的过程中，可能会出现好几种可能的切分方式，将这些组合构成有向无环图，如下图所示：

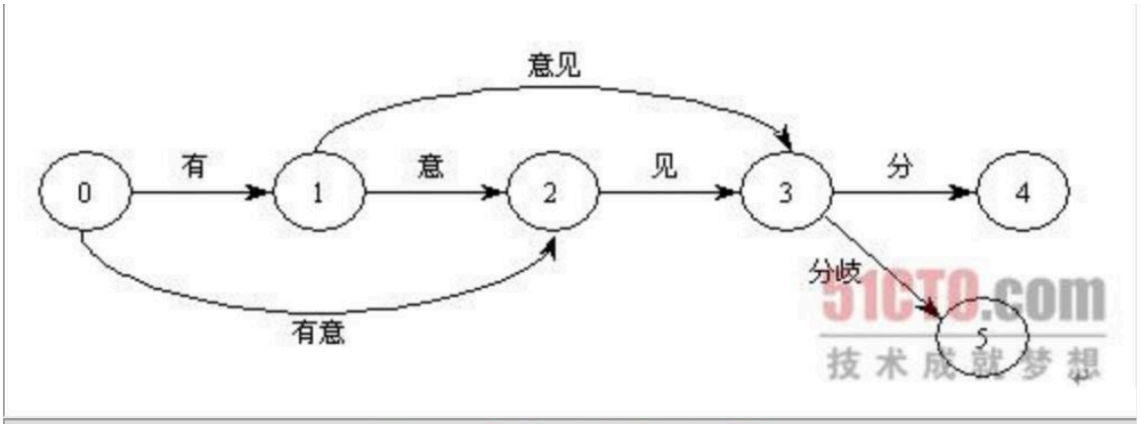


图4-9 中文分词切分路径

   可以看到，构成了两条路径：

   DAG中记录了某个词的开始位置和它可能的结束位置。开始位置作为key，结束位置是一个list。比如位置0的DAG表达为 `{0: [1, 2]}`，也就是说0位置为词的开始位置时，1，2位置都有可能是词的结束位置。上面语句的完整DAG为

```
{
    0: [1, 2],
    1: [2, 3],
    2: [3],
    3: [4, 5],
    4: [5]
}
```

4. 构建节点最大路径概率，以及结束位置。计算每个汉字节点到语句结尾的所有路径中的最大概率，并记下最大概率时在DAG中对应的该汉字成词的结束位置。
   中文一般形容词在前面，而相对来说更关键的名词和动词在后面。考虑到这一点，**jieba中对语句，从右向左反向计算路径的最大概率，这个类似于逆向最大匹配。每个词的概率 = 字典中该词的词数 / 字典总词数**。对于上图构建每个节点的最大路径概率的过程如下：

```
p(5)= 1,
p(4)= max(p(5) * p(4-&gt;5)),
p(3)= max(p(4) * p(4-&gt;5),   p(5) * p(3-&gt;5)),  # 对于节点3，他有3-&gt;4，
3-&gt;5两条路径，我们取概率最大的路径作为节点3的路径概率，并记下概率最大时节点3的结束位置
p(2) = max(p(3) * p(2-&gt;3))
p(1) = max(p(2) * p(1-&gt;2),   p(3) * p(1-&gt;3))
p(0) = max(p(1) * p(0-&gt;1),   p(2) * p(0-&gt;2))
```

5. 构建切分组合。根据节点路径，得到词语切分的结果，也就是分词结果。
   从节点0开始，按照步骤4中构建的最大路径概率以及结束位置，取出节点0的结束位置，构成词语。如果是单字词语，则直接通过yield返回。如果词语在字典中，也直接通过yield返回。如果词语不在字典中，也就是新词，则需要通过HMM隐马尔科夫模型来分割。节点0处理完毕，则跳到下一个词语的开始处进行处理，直至到达语句末尾。

6. HMM新词处理：对于新词，也就是dict.txt中没有的词语，通过统计方法来处理，jieba中采用了HMM隐马尔科夫模型来处理。

7. 对于新词，也就是`dict.txt`中没有的词语，通过统计方法来处理，jieba中采用了HMM隐马尔科夫模型。
   回顾下HMM的五要素：**观测序列，隐藏序列，发射概率，起始概率，转移概率。**由这五大要素可以对短语建模。**通过语料大规模训练，可以得到发射概率，起始概率和转移概率。通过viterbi算法，可以得到概率最大的隐藏序列**，也就是 BEMS标注序列，通过BEMS就可以对语句进行分词了。观察发现，新词被分成二字词语的概率很大。

   Viterbi算法的步骤的核心是动态规划：通过每次走一个边就乘以它的路径概率，到最后得到最大概率。

   （路径概率 = 初始概率 × 转移概率 × 发射概率）

步骤	作用
初始化	计算初始状态概率
递推	动态规划求解每一步的最优路径
终止	找到最终时刻的最大概率
回溯	从终点反向追溯最优路径

8. 返回分词结果：通过yield将上面步骤中切分好的词语逐个返回。yield相对于list，可以节约存储空间。

> **Note**
>
> `yield` 是 Python 中用于定义生成器（generator）的关键字。与普通函数不同，生成器函数通过 `yield` 逐步返回值，而不是一次性返回所有结果。