jieba分词详解

作者：尘嚣看客

链接：https://www.jianshu.com/p/2cccb07d9a4e

来源：简书

# 1 引言

“结巴”分词是一个Python中文分词组件，参见https://github.com/fxsjy/jieba

可以对中文文本进行分词、词性标注、关键词抽取等功能，并且支持自定义词典。

本文包括以下内容：

1、jieba分词包的安装

2、jieba分词的使用教程

3、jieba分词的工作原理与工作流程

4、jieba分词所涉及到的HMM、TextRank、TF-IDF等算法介绍

# 2 安装

可以直接使用pip来进行安装：

pip install jieba

# 3 简单使用

## 3.1 分词

对一句话进行分词

|  |
| --- |
| import jieba  text = "征战四海只为今日一胜，我不会再败了。"  # jieba.cut直接得到generator形式的分词结果  seg = jieba.cut(text)  print(' '.join(seg))  # 也可以使用jieba.lcut得到list的分词结果  seg = jieba.lcut(text)  print(seg) |

|  |
| --- |
| 征战 四海 只 为 今日 一胜 ， 我 不会 再败 了 。  ['征战', '四海', '只', '为', '今日', '一胜', '，', '我', '不会', '再败', '了', '。'] |

命令行进行分词

|  |
| --- |
| python -m jieba input.txt > output.txt |

## 3.2 词性分析

|  |
| --- |
| import jieba.posseg as posseg  text = "征战四海只为今日一胜，我不会再败了。"  # generator形式形如pair(‘word’, ‘pos’)的结果  seg = posseg.cut(text)  print([se for se in seg])  # list形式的结果  seg = posseg.lcut(text)  print(seg) |

|  |
| --- |
| [pair('征战', 'v'), pair('四海', 'ns'), pair('只', 'd'), pair('为', 'p'), pair('今日', 't'), pair('一', 'm'), pair('胜', 'v'), pair('，', 'x'), pair('我', 'r'), pair('不会', 'v'), pair('再败', 'v'), pair('了', 'ul'), pair('。', 'x')]  [pair('征战', 'v'), pair('四海', 'ns'), pair('只', 'd'), pair('为', 'p'), pair('今日', 't'), pair('一', 'm'), pair('胜', 'v'), pair('，', 'x'), pair('我', 'r'), pair('不会', 'v'), pair('再败', 'v'), pair('了', 'ul'), pair('。', 'x')] |

## 3.3 关键词抽取

关键词抽取有两种算法，基于TF-IDF和基于TextRank：

|  |
| --- |
| import jieba.analyse as analyse  text = "征战四海只为今日一胜，我不会再败了。"  # TF-IDF  tf\_result = analyse.extract\_tags(text, topK=5) # topK指定数量，默认20  print(tf\_result)  # TextRank  tr\_result = analyse.textrank(text, topK=5) # topK指定数量，默认20  print(tr\_result) |

|  |
| --- |
| ['一胜', '再败', '征战', '四海', '今日']  ['一胜', '再败', '征战', '四海', '今日'] |

# 4 完整用法

## 4.1 分词

jieba分词有三种不同的分词模式：精确模式、全模式和搜索引擎模式：

|  |
| --- |
| jieba.cut(sentence,cut\_all=False,HMM=True) # 精确模式  jieba.cut(sentence,cut\_all=True,HMM=True) # 全模式  jieba.cut\_for\_search (sentence, HMM=True) # 搜索引擎模式 |

对应的，函数前加l即是对应得到list结果的函数：

|  |
| --- |
| jieba.lcut(sentence,cut\_all=False,HMM=True) # 精确模式  jieba.lcut(sentence,cut\_all=True,HMM=True) # 全模式  jieba.lcut\_for\_search (sentence, HMM=True) # 搜索引擎模式 |

|  |
| --- |
| sentence = "征战四海只为今日一胜，我不会再败了。"  #---------------result----------------  '今天天气 真 好' # 精确模式  '今天 今天天气 天天 天气 真好' # 全模式  '今天 天天 天气 今天天气 真 好' # 搜索引擎模式 |

精确模式是最常用的分词方法，全模式会将句子中所有可能的词都列举出来，搜索引擎模式则适用于搜索引擎使用。具体的差别可在下一节工作流程的分析中详述。

在上述每个函数中，都有名为HMM的参数。这一项表示是否在分词过程中利用HMM进行新词发现。关于HMM，本文附录中将简述相关知识。

另外分词支持自定义字典，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。

具体使用方法为：

|  |
| --- |
| jieba.load\_userdict(file\_name) # 载入自定义词典  jieba.add\_word(word, freq=None, tag=None) # 在程序中动态修改词典  jieba.del\_word(word)  jieba.suggest\_freq(segment, tune=True) # 调节单个词语的词频，使其能/不能被分词开 |

## 4.2 关键词抽取

关键词抽取的两个函数的完整参数为：

|  |
| --- |
| jieba.analyse.extract\_tags(sentence, topK=20, withWeight=False, allowPOS=(), withFlag=False)  # topK 表示返回最大权重关键词的个数，None表示全部  # withWeight表示是否返回权重，是的话返回(word,weight)的list  # allowPOS仅包括指定词性的词，默认为空即不筛选。  jieba.analyse.textrank(self, sentence, topK=20, withWeight=False, allowPOS=('ns', 'n', 'vn', 'v'), withFlag=False)  # 与TF-IDF方法相似，但是注意allowPOS有默认值，即会默认过滤某些词性。 |

## 4.3 并行分词

可以通过

|  |
| --- |
| jieba.enable\_parallel(4) # 开启并行分词模式，参数为并行进程数，默认全部  jieba.disable\_parallel() # 关闭并行分词模式 |

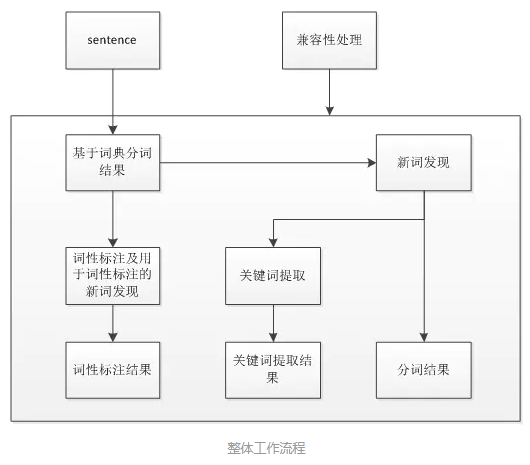
来打开或关闭并行分词功能。个人感觉一般用不到，大文件分词需要手动实现多进程并行，句子分词也不至于用这个。

# 5 代码研读与工作流程分析

## 5.1 整体工作流程

jieba分词主要通过词典来进行分词及词性标注，两者使用了一个相同的词典。正因如此，分词的结果优劣将很大程度上取决于词典，虽然使用了HMM来进行新词发现。

jieba分词包整体的工作流程如下图所示：

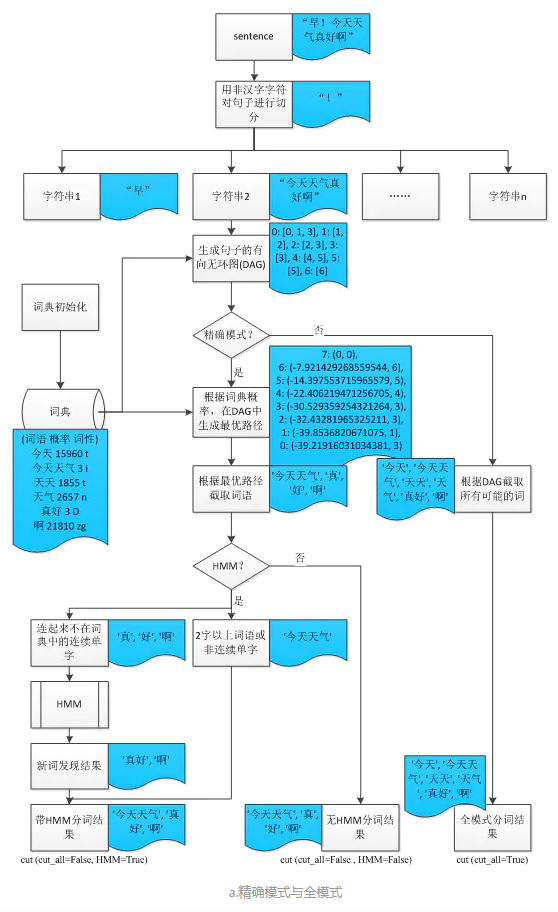


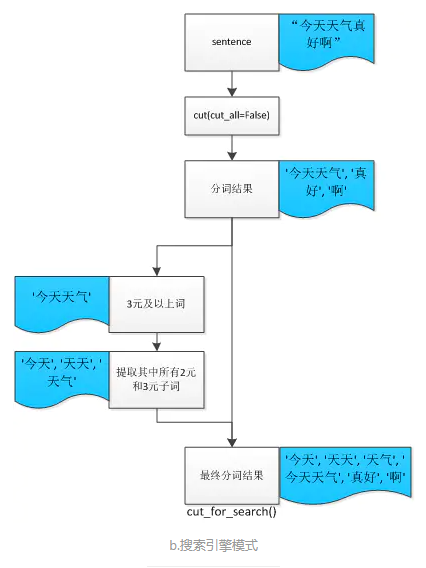
下面将根据源码详细地分析各个模块的工作流程。

在之后几节中，我们在蓝色的方框中示范了关键步骤的输出样例或词典文件的格式样例。在本节中都采用类似的表示方式。

## 5.2 分词

jieba分词中，首先通过对照典生成句子的有向无环图，再根据选择的模式不同，根据词典寻找最短路径后对句子进行截取或直接对句子进行截取。对于未登陆词（不在词典中的词）使用HMM进行新词发现。





a图中演示了分词的主要过程，但是其中只演示了被切分出来的一个子字符串的分词操作过程，在实际操作流程中，将对每一个子字符串都分别进行图中的处理，最后将切分的分词结果与非汉字部分依次连接起来，作为最终的分词结果。

如果开启了HMM，那么将会连起来不在词典中出现的连续单字进行新词发现。比如例子中的“真好啊”，词典中没有这个词，所以会拿去HMM模型中进行新词发现；但是如果原句是“今天天气真好”，基于词典切分为“今天天”“真”“好”，词典中有“真好”一词，但是因为频率小所以未被选择为最佳路径，所以“真”“好”两个字不会被拿去做新词发现（即便其通过HMM的结果将会是“真好”），最终分词结果将是“今天天气”“真”“好”。

词典的格式应为

word1 freq1 word\_type1

word2 freq2 word\_type2

…

其中自定义用户词典中词性word\_type可以省略。

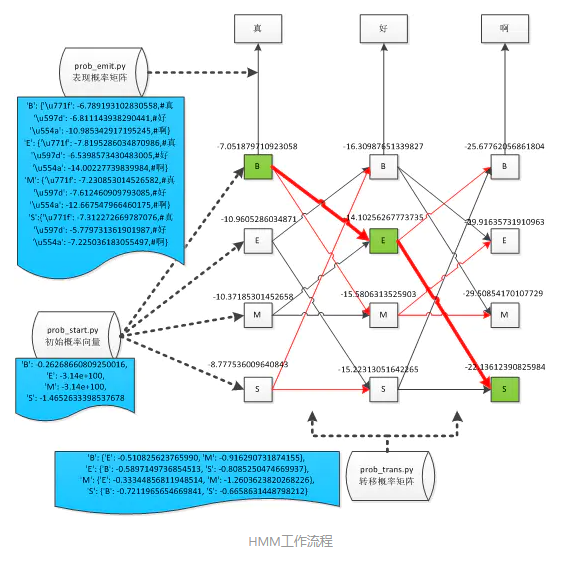
词典在其他模块的流程中可能也会用到，为方便叙述，后续的流程图中将会省略词典的初始化部分。

图b演示了搜索引擎模式的工作流程，它会在精确模式分词的基础上，将长词再次进行切分。

## 5.3 HMM与新词发现

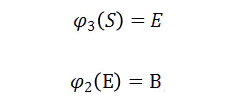
在这里我们假定读者已经了解HMM相关知识，如果没有可先行阅读下一章内容中的HMM相关部分或者跳过本节。

在jieba分词中，将字在词中的位置B、M、E、S作为隐藏状态，字是观测状态，使用了词典文件分别存储字之间的表现概率矩阵（finalseg/prob\_emit.py）、初始概率向量(finalseg/prob\_start.py)和转移概率矩阵(finalseg/prob\_trans.py)。这就是一个标准的解码问题，根据概率再利用viterbi算法对最大可能的隐藏状态进行求解。



上图简单示范了jieba分词中新词发现模块的工作流程，其具体计算过程可参考附录内容，为求页面整齐，这里不再累述。

在最后时刻，即“啊”对应的时刻里，最大概率的为S，而



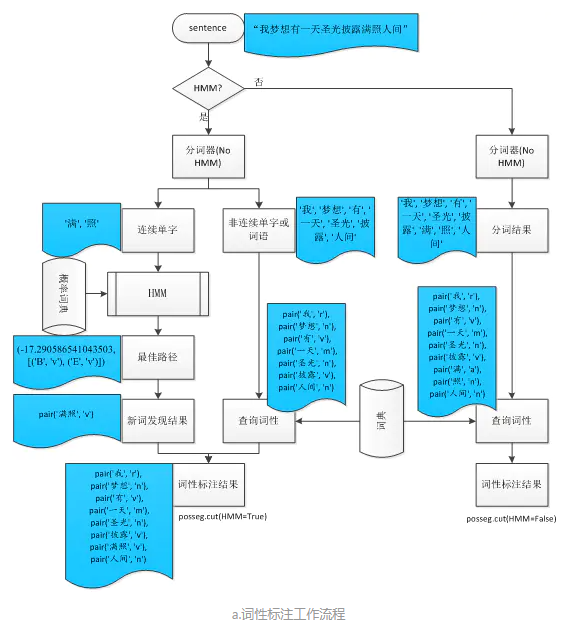
，那么隐藏状态序列即为(BES)，对应于汉字“真好啊”——即“真好”是一个词，“啊”字单字成词。

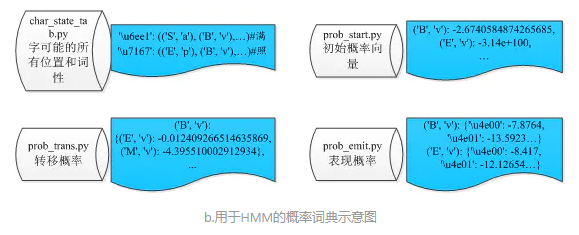
## 5.4 词性分析

词性分析部分与分词模块用了同一个基础的分词器，对于词典词的词性，将直接从词典中提取，但是对于新词，词性分析部分有一个专属的新词及其词性的发现模块。

用于词性标注的HMM模型与用于分词的HMM模型相似，同样将文字序列视为可见状态，但是隐藏状态不再是单单的词的位置（B/E/M/S），而变成了词的位置与词性的组合，如(B,v)(B,n)(S,n)等等。因此其初始概率向量、转移概率矩阵和表现概率矩阵和上一节中所用的相比都要庞大的多，但是其本质以及运算步骤都没有变化。

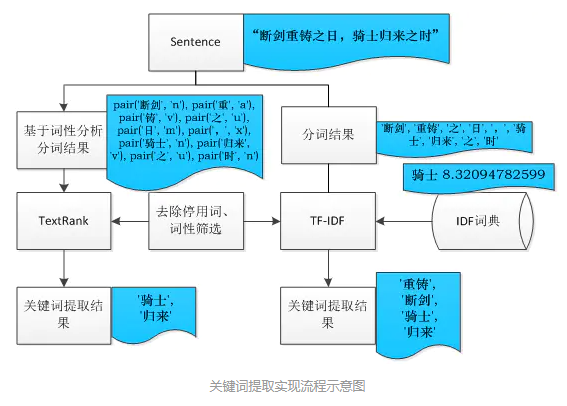
具体的工作流程如下图所示。





## 5.5 关键词提取

jieba分词中有两种不同的用于关键词抽取的算法，分别为TextRank和TF-IDF。实现流程比较简单，其核心在于算法本身。下面简单地画出实现流程，具体的算法可以参阅下一章内容。



TextRank方法默认筛选词性，而TF-IDF方法模型不进行词性筛选。

# 6 附录

在本章中，将会简单介绍相关的算法知识，主要包括用于新词发现的隐马尔科夫模型和维特比算法、用于关键词提取的TextRank和TF-IDF算法。

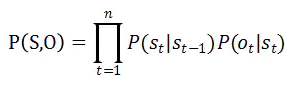
## 6.1 HMM

HMM即隐马尔科夫模型，是一种基于马尔科夫假设的统计模型。之所以为“隐”，是因为相较于马尔科夫过程HMM有着未知的参数。在世界上，能看到的往往都是表象，而事物的真正状态往往都隐含在表象之下，并且与表象有一定的关联关系。

此处我们假设读者已经对机器学习或统计模型等相关内容有了一个大致的了解，我们利用各种模型的目的在于对于给定的输入X，能够预测出类别Y。生成模型通过学习联合概率分布P(X,Y)，然后通过贝叶斯定理求解条件概率



HMM属于生成模型的有向图PGM，通过联合概率建模：



其中，S、O分别表示状态序列与观测序列。

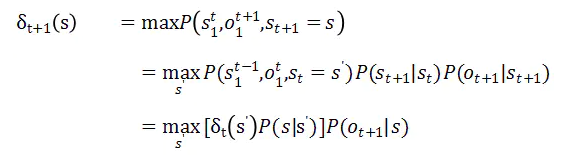
HMM的解码问题为



定义在时刻t状态为s的所有单个路径st1中的概率最大值为



则有：



此式即为用于HMM解码问题的Viterbi算法的递推式。

如果读者还对这部分内容心存疑问，不妨先往下阅读，下面我们将以一个比较简单的例子对HMM及解码算法进行实际说明与演示，在读完下一小节之后再回来看这些式子，或许能够恍然大悟。

下面以一个简单的例子来进行阐述：

假设小明有一个网友小红，小红每天都会在朋友圈说明自己今天做了什么，并且假设其仅受当天天气的影响，而当天的天气也只受前一天天气的影响。

于小明而言，小红每天做了什么是可见状态，而小红那里的天气如何就是隐藏状态，这就构成了一个HMM模型。一个HMM模型需要有五个要素：隐藏状态集、观测集、转移概率、观测概率和初始状态概率。

我们定义隐藏状态集为N，N中包括了所有有可能出现的隐藏状态，在本例中我们认为：



定义观测集为M，M中包括了所有可能出现在观测中的表现状态，在本例中我们假设：



接下来定义观测概率矩阵：



其中，



即在第j个隐藏状态时，表现为i表现状态的概率。式中的n和m表示隐藏状态集和观测集中的数量。

本例中在不同的天气下，小红要做不同事情的概率也不同，观测概率以表格的形式呈现如下：



HMM中还定义了转移概率矩阵：

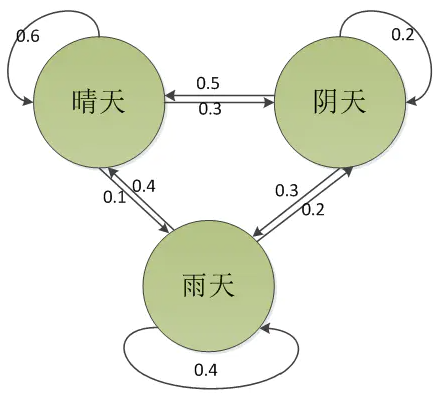


其中



表示第i个隐藏状态转移为第j个隐藏状态的概率。

本例中我们认定，其转移概率如下图所示：



除此之外，还需要一个初始状态概率向量π，它表示了观测开始时，即t=0时，隐藏状态的概率值。本例中我们指定π={0,0,1}。

至此，一个完整的隐马尔科夫模型已经定义完毕了。

HMM一般由三类问题：

* 概率计算问题，即给定A,B,π和隐藏状态序列，计算观测序列的概率；
* 预测问题，也成解码问题，已知A,B,π和观测序列，求最优可能对应的状态序列；
* 学习问题，已知观测序列，估计模型的A,B,π参数，使得在该模型下观测序列的概率最大，即用极大似然估计的方法估计参数。

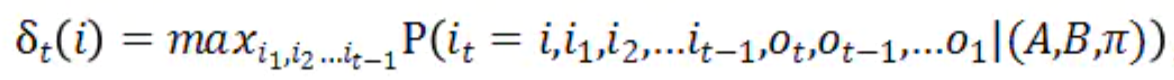
在jieba分词中所用的是解码问题，所以此处对预测问题和学习问题不做深入探讨，在下一小节中我们将继续以本节中的例子为例，对解码问题进行求解。

## 6.2 Viterbi算法

在jieba分词中，采用了HMM进行新词发现，它将每一个字表示为B/M/E/S分别代表出现在词头、词中、词尾以及单字成词。将B/M/E/S作为HMM的隐藏状态，而连续的各个单字作为观测状态，其任务即为利用观测状态预测隐藏状态，并且其模型的A,B,π概率已经给出在文件中，所以这是一个标准的解码问题。在jieba分词中采用了Viterbi算法来进行求解。

Viterbi算法的基本思想是：如果最佳路径经过一个点，那么起始点到这个点的路径一定是最短路径，否则用起始点到这点更短的一条路径代替这段，就会得到更短的路径，这显然是矛盾的；从起始点到结束点的路径，必然要经过第n个时刻，假如第n个时刻有k个状态，那么最终路径一定经过起始点到时刻n中k个状态里最短路径的点。

将时刻t隐藏状态为i所有可能的状态转移路径i1到i2的状态最大值记为：



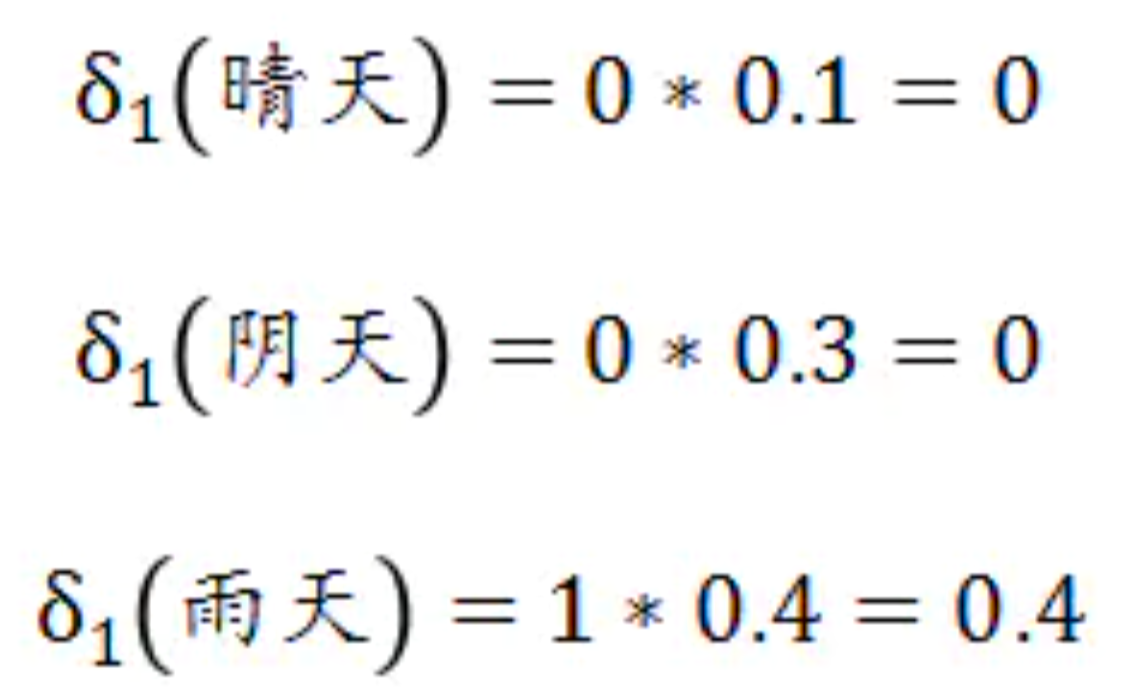
我们可以据此由初始时刻依次向后推出每一个时刻的最大概率隐藏状态。

下面我们继续以上一节中的例子来对viterbi算法进行阐述：

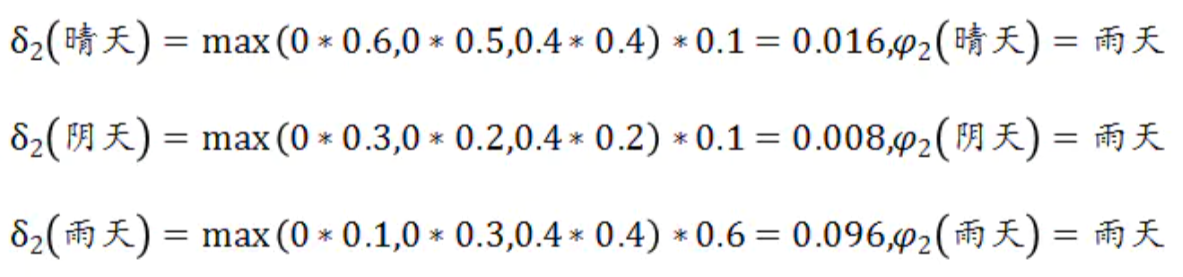
小明不知道小红是哪里人，他只能通过小红每天的活动来推断那里的天气。

假设连续三天，小红的活动依次为：“睡觉-打游戏-逛街”，我们将据此计算最有可能的天气情况。

我们需要得到三种天气在第一天对应的可能出现“睡觉”的可能性，考虑到初始概率向量：



现在开始递推三个隐藏状态（天气）在第二天时对应的各自可见状态（打游戏）：



在上式中，



表示能使得

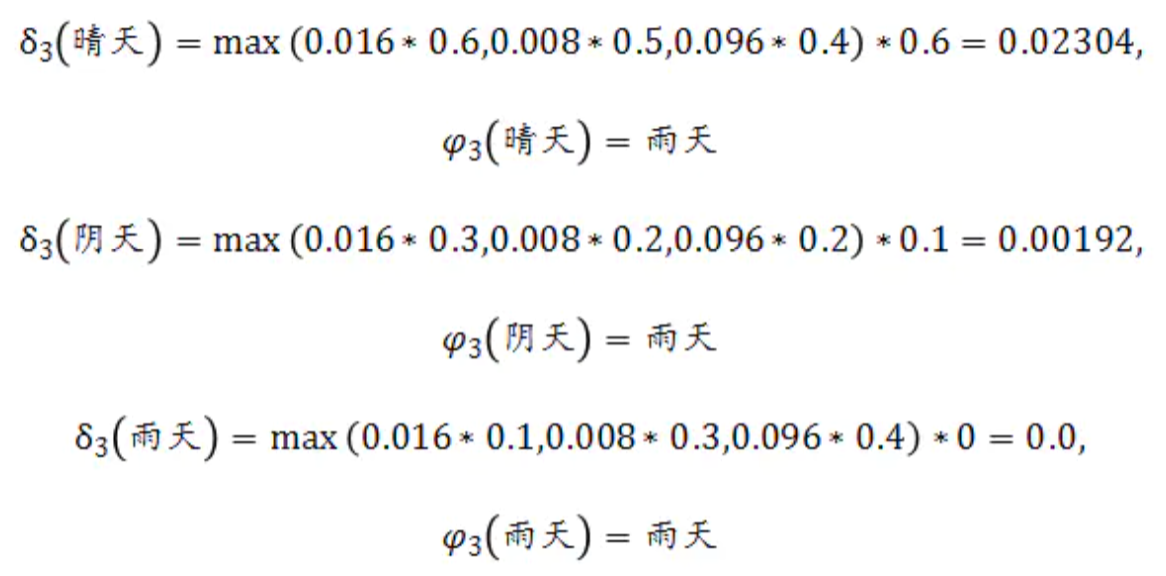


最大的前一时刻的隐藏状态，比如



表示第一天为雨天能够使得第二天为晴天的概率最大（也就是说如果第二天是晴天在最短路径上的话，第一天是雨天也一定在最短路径上，参见上文中Viterbi算法的基本思想）

下面继续递推第三天（逛街）的隐藏状态：

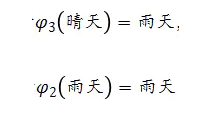


此时已经到了最后的时刻，我们开始回溯。

此时的最大概率为

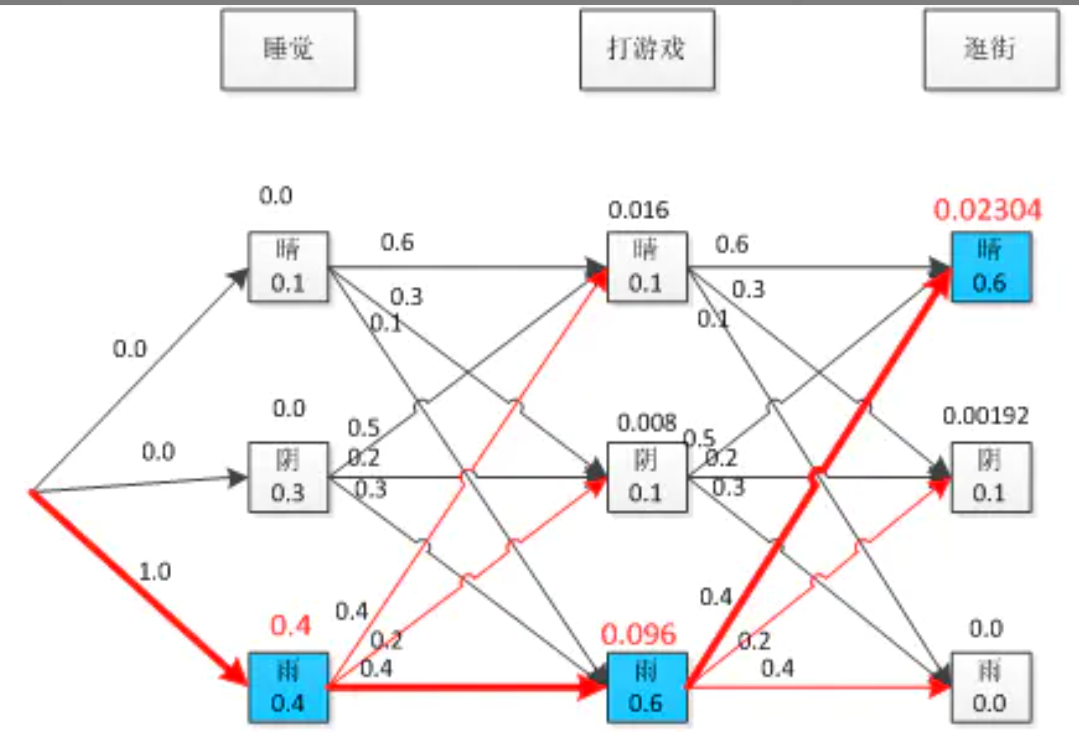


由于



从而得到最终最有可能的隐藏状态序列为：（雨天，雨天，晴天）。

其计算过程示意图如下图所示。



在图中，线条上方的数字表示转移概率（或初始概率），隐藏状态点框内的数字表示表现概率，隐藏状态点框上方的数字表示此隐藏状态点的最大联合概率，即



，指向隐藏节点i的红色的线条表示使得节点i处联合概率最大的路径，加粗的红色线条表示能使最终时刻联合概率最大（即

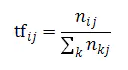


## 6.3 TF-IDF

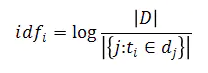
TF-IDF（词频-逆文本频率）是一种用以评估字词在文档中重要程度的统计方法。它的核心思想是，如果某个词在一篇文章中出现的频率即TF高，并且在其他文档中出现的很少，则认为这个词有很好的类别区分能力。



其中



式中，分子为i词在j文档中出现的次数，分母为j文档中所有字词出现的次数之和。



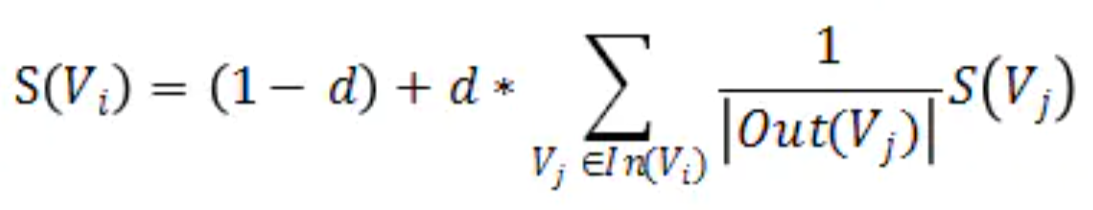
式中分子为语料库中的文件总数，分母为包含该词的文件数目。

jieba分词中逆文档频率直接由词典读入。

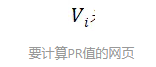
## 6.4 TextRank

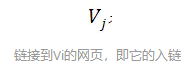
TextRank是一种用以关键词提取的算法，因为是基于PageRank的，所以先介绍PageRank。

PageRank通过互联网中的超链接关系确定一个网页的排名，其公式是通过一种投票的思想来设计的：如果我们计算网页A的PageRank值，那么我们需要知道哪些网页链接到A，即首先得到A的入链，然后通过入链给网页A进行投票来计算A的PR值。其公式为：

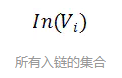


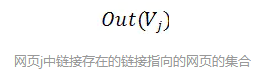
其中：

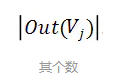










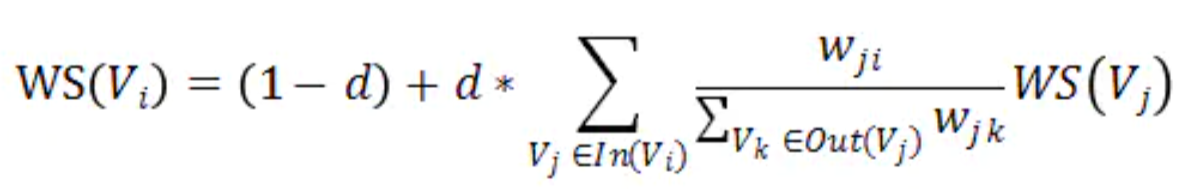


d为阻尼系数，取值范围为0-1，代表从一定点指向其他任意点的概率，一般取值0.85。

将上式多次迭代即可直到收敛即可得到结果。

TextRank算法基于PageRank的思想，利用投票机制对文本中重要成分进行排序。如果两个词在一个固定大小的窗口内共同出现过，则认为两个词之间存在连线。

TextRank算法的得分定义为：



公式与PageRank的基本相同。多次迭代直至收敛，即可得到结果。

在jieba分词中，TextRank设定的词窗口大小为5，将公式1迭代10次的结果作为最终权重的结果，而不一定迭代至收敛。