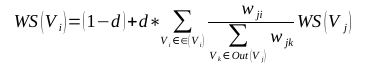
**TextRank**

TextRank算法是一种用于文本的基于图的排序算法。其基本思想来源于谷歌的PageRank算法, 通过把文本分割成若干组成单元(单词、句子)并建立图模型, 利用投票机制对文本中的重要成分进行排序, 仅利用单篇文档本身的信息即可实现关键词提取、文摘。和 LDA、HMM 等模型不同, TextRank不需要事先对多篇文档进行学习训练, 因其简洁有效而得到广泛应用。

TextRank 一般模型可以表示为一个有向有权图 G =(V, E), 由点集合 V和边集合 E 组成, E 是V ×V的子集。图中任两点 Vi , Vj 之间边的权重为 wji , 对于一个给定的点 Vi, In(Vi) 为 指 向 该 点 的 点 集 合 , Out(Vi) 为点 Vi 指向的点集合。点 Vi 的得分定义如下:



其中, d 为阻尼系数, 取值范围为 0 到 1, 代表从图中某一特定点指向其他任意点的概率, 一般取值为 0.85。使用TextRank 算法计算图中各点的得分时, 需要给图中的点指定任意的初值, 并递归计算直到收敛, 即图中任意一点的误差率小于给定的极限值时就可以达到收敛, 一般该极限值取 0.0001。

**基于TextRank的关键词提取**

关键词抽取的任务就是从一段给定的文本中自动抽取出若干有意义的词语或词组。TextRank算法是利用局部词汇之间关系（共现窗口）对后续关键词进行排序，直接从文本本身抽取。其主要步骤如下：

1. 把给定的文本T按照完整句子进行分割，
2. 对于每个句子，进行分词和词性标注处理，并过滤掉停用词，只保留指定词性的单词，如名词、动词、形容词，其中是保留后的候选关键词。
3. 构建候选关键词图G = (V,E)，其中V为节点集，由2生成的候选关键词组成，然后采用共现关系（co-occurrence）构造任两点之间的边，两个节点之间存在边仅当它们对应的词汇在长度为K的窗口中共现，K表示窗口大小，即最多共现K个单词。
4. 根据上面公式，迭代传播各节点的权重，直至收敛。
5. 对节点权重进行倒序排序，从而得到最重要的T个单词，作为候选关键词。
6. 由5得到最重要的T个单词，在原始文本中进行标记，若形成相邻词组，则组合成多词关键词。例如，文本中有句子“Matlab code for plotting ambiguity function”，如果“Matlab”和“code”均属于候选关键词，则组合成“Matlab code”加入关键词序列。

**基于TextRank的自动文摘**

基于TextRank的自动文摘属于自动摘录，通过选取文本中重要度较高的句子形成文摘，其主要步骤如下：

1. 预处理：将输入的文本或文本集的内容分割成句子得，构建图G =（V,E），其中V为句子集，对句子进行分词、去除停止词，得，其中是保留后的候选关键词。
2. 句子相似度计算：构建图G中的边集E，基于句子间的内容覆盖率，给定两个句子，采用如下公式进行计算：simikarity-1若两个句子之间的相似度大于给定的阈值，就认为这两个句子语义相关并将它们连接起来，即边的权值：similarity-2
3. 句子权重计算：根据公式，迭代传播权重计算各句子的得分；
4. 抽取文摘句：将3得到的句子得分进行倒序排序，抽取重要度最高的T个句子作为候选文摘句。
5. 形成文摘：根据字数或句子数要求，从候选文摘句中抽取句子组成文摘。

参考资料：

* <https://github.com/letiantian/TextRank4ZH>
* <https://github.com/chenbjin/ASExtractor>

https://github.com/fxsjy/jieba/blob/master/jieba/analyse/textrank.py

关键词抽取中的TextRank

之前我们也提到了，就是将PageRank模型中的page替换成word。因为是用于文本中，所以与用于网页中是不一样的。改动的地方主要有：

PageRank中因为有链接的存在，所以是将存在链接的两个网页相连。而在文本处理中，一个词的出现，与其前后词之间是有联系的。在TextRank中存在一个窗口的概念，将一个词与窗口内的所有词进行连接。窗口值的设置也是比较讲究的。实验表明，如果窗口值设置的过大，不仅带来计算时间上的提高（因为图中的边数增加），而且会造成模型性能的下降。这里模型的性能是使用具体的任务去衡量的，关键词抽取是作为具体任务的前驱。

因为链接是具有指向性的，所以PageRank模型构造的是一个有向图。而在文本中，词与词之间的联系是很难确定方向的，所以既可以构造一个有向图也可以构造一个无向图。实验表明，有向图和无向图在收敛性上是没有差别的，就是它们都会收敛到一个固定值上。至于初始值的设置，在PageRank中就已经证明了，最终的结果与初值的设定无关，只与迭代的次数有关。只要迭代的次数足够，就能收敛到一个固定值。

在PageRank中，会将爬虫所爬到的所有网页都加入到模型中去，至于死链的去除，这都是后话。但在TextRank中，不会将文本中出现的所有词都加入到图模型中。因为不是所有的词都是有其实际意义的，例如说介词。所以在构建图模型时，会利用一定的语法规则过滤掉一些词，以减小图的规模。这里的过滤规则一般来说是基于词性的，所以在之前进行分词是还需要将词的词性给标记出来。实验表明，最佳的选择是保留名词和形容词。

PageRank模型中，连接的关系一般只会发生一次，很少会有一个网页中存在两个指向同一个网页的链接。但在文本中却不同，有一些词是固定的搭配，它们会在一起出现多词。所以TextRank所构造的图应该是一个赋权图。这里的权值应该是两者共同出现的次数。相应的每个结点权值的更新公式应该更改为：

WS(Vi)=(1−d)+d×∑Vj∈In(Vi)wji∑Vk∈Out(Vj)wjkWS(Vj)

其中d表示衰减系数，这个在PageRank模型中就已经有了的，WS(Vi)表示结点Vi的权值。在无向图中，In(Vi)、Out(Vi)均表示与结点Vi相连的结点；在有向图中，则分别表示结点的入边和出边。

在构造完图之后，就与PageRank一样，使用迭代的算法计算每一个结点的权值。根据你想获得的关键词的数目，输出权值最高的那些结点所对应的词。

自动文摘中的TextRank

在关键词抽取任务中，图模型中的结点是词，在自动文摘中结点表示的就是句子。模型的构建方式自然也要发生一些改变，主要有以下两点：

首先，窗口模型在这个任务中并不适用，因为前后句子间的关系不像是词间的关系那么紧密。

其次，图中的权值使用的是句子间的相似度，具体是使用哪一种衡量标准，还是要看实现吧。

代码详解：

所谓的无向有权图就是一个词典，词典的key是起始点代表的词，词典的value则是一个由（起始点，终止点，边的权重）构成的三元组所组成的列表，表示以这个词作为起始点的所有的边。

def addEdge(self, start, end, weight):

self.graph[start].append((start, end, weight))

self.graph[end].append((end, start, weight))

执行textrank算法迭代是在rank函数中完成的。

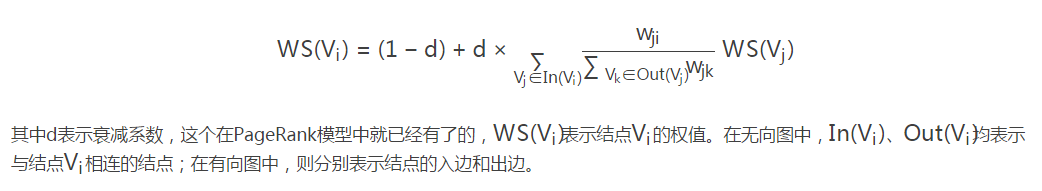
首先对每个结点赋予相同的权重，以及计算出该结点的所有出度的次数之和；

然后迭代若干次，以确保得到稳定的结果；

在每一次迭代中，依次遍历每个结点；对于结点n，首先根据无向有权图得到结点n的所有  
入度结点（对于无向有权图，入度结点与出度结点是相同的，都是与结点n相连的结点），在前面我们已经计算出这个入度结点的所有出度的次数，而它对于结点n的权值的贡献等于它本身的权值 乘以 它与结点n的共现次数 / 这个结点的所有出度的次数 ，将各个入度结点得到的权值相加，再乘以一定的阻尼系数，即可得到结点n的权值；

迭代完成后，对权值进行归一化，并返回各个结点及其对应的权值。

每个结点权值的更新公式应该更改为



def rank(self):

ws = defaultdict(float)

outSum = defaultdict(float)

# 计算初始平均权重， or是短路操作符

wsdef = 1.0 / (len(self.graph) or 1.0)

#n是当前节点 out就是以n为key的三元组的集合，e[2]是权重

for n, out in self.graph.items():

ws[n] = wsdef # 初始化各个结点的权值

# 统计 各个结点的出度的次数之和

outSum[n] = sum((e[2] for e in out), 0.0) # 这个, 0.0没用

sorted\_keys = sorted(self.graph.keys())

# 遍历若干次

for x in xrange(10): # 10 iters

# 遍历各个结点

for n in sorted\_keys:

s = 0

# 遍历结点的end结点

for e in self.graph[n]:

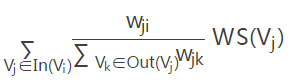
# 将这些end结点贡献后的权值相加

# 贡献率 = end结点与结点n的共现次数 / end结点的所有出度之和

# e[2]就是当前节点n与end结点之间的权重（这边就是贡献次数）

# outSum[e[1]]是end结点的总出度次数，

#end结点的当前权重ws[e[1]]

#

#e[2]就是 outSum[e[1]]就是 (这个是不是简化了)

#应该是End的所有出度的边的权重

#ws[e[1]]就是

s += (e[2] / outSum[e[1]]) \* ws[e[1]]

# 更新结点n的权值

ws[n] = (1 - self.d) + self.d \* s

(min\_rank, max\_rank) = (sys.float\_info[0], sys.float\_info[3])

# 获取权值的最大值和最小值

for w in itervalues(ws):

if w < min\_rank:

min\_rank = w

if w > max\_rank:

max\_rank = w

# 对权值进行归一化

for n, w in ws.items():

# to unify the weights, don't \*100.

ws[n] = (w - min\_rank / 10.0) / (max\_rank - min\_rank / 10.0)

return ws