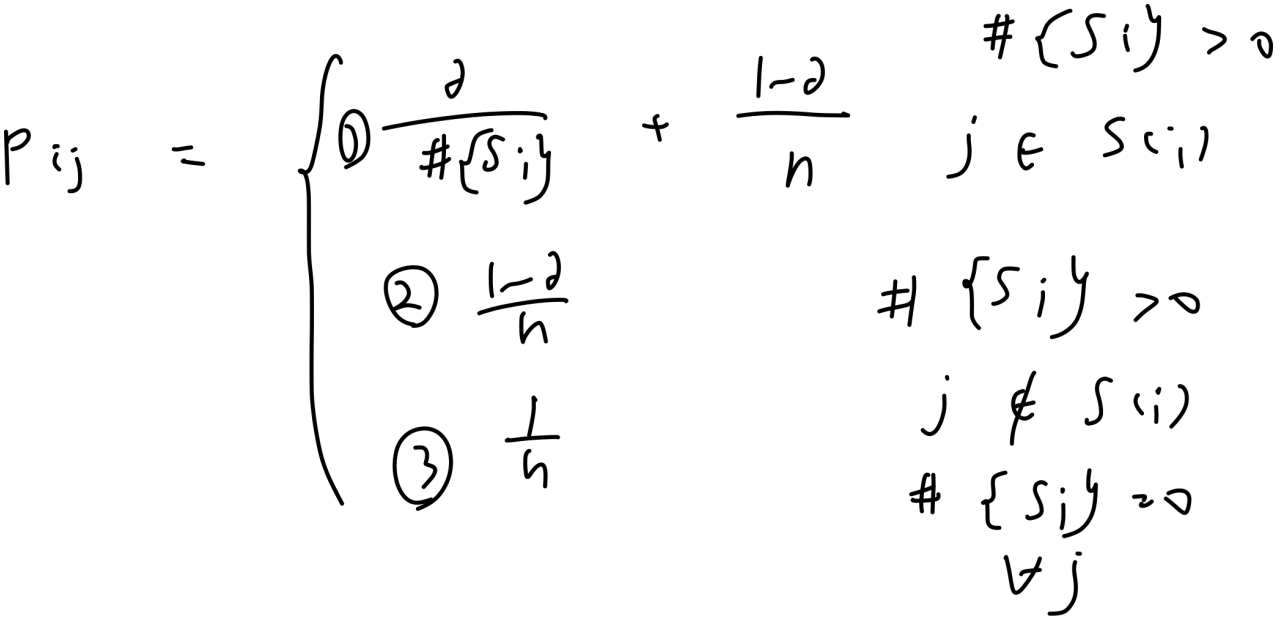
本文基于TextRank, 实现了关键词的提取和关键句算法.

原理概述

PageRank

TextRank其实是将PageRank算法应用到文本上. PageRank算法本质上是在求一个markov chain的平稳分布, PageRank的新颖之处, 在于它用到了随机过程的想法, 以及转移矩阵的构造. 转移矩阵的构造是: 一个网页wi有α的概率访问网页本身包含的链接, 也有(1-α)的概率访问其它网页. 设全部网页数为n. wi的可达状态集合为Si. 出现在网页wi中的链接的转移概率可以用全概率公式, `α/{#Si}+(1-α)/n`, 未出现在wi中的链接的转移概率是`(1-α)/n`.

一种特殊情况是一个网页不包含任何链接, 那么它到所有网页的转移概率都是1/n. 用列向量表示转移概率, 那么转移矩阵是:



/Users/quebec/Notes/文摘/assets/20210106164722-v0oou61.png

下一步就是平稳分布的求解, 有两种方法, 解方程, 因为平稳分布满足特征方程Px=x, 这个线性方程的解就是平稳分布, 也就是各状态到达的概率. 用一些近似方法, 在P非常稀疏的情况下是可行的. 但我们并不这样求, 原因是, 在TextRank中, 随求词以及句相似度方法的不同, 矩阵常常并不稀疏. 在矩阵稠密的情况下, 幂法是更好的方法.

幂法其实就是不断地求 $x_{k+1} = P * x_k$, 直到收敛, 收敛的判定可以用绝对条件数或者相对条件数, 代码中用到的是简化的求`x_{k+1}-x_{k}`的二范数.

能够使用幂法的原因在于:

- 转移矩阵P的主特征值为1, 而且是唯一主特征值, 重数为1
- 平稳分布是转移矩阵P的主特征值的特征向量, 概率分布满足和为1, 因此单位化后(除以和), 结果唯一.
- 幂法中 $P^n x$ 收敛到P的主特征值对应的特征向量.

另外使用幂法注意到:

- 计算过程中保持`Px`的列和为1, 因为`Px`本身也是概率分布, 满足和为1.

TextRank

TextRank关于关键词的提取, 在PageRank基础上, 只有一个不同, 那就是, 如何定义一个词到另一个词的转移关系. 答案就是加窗共现. 如果词j在词i的窗内共现, 相当于网页j在网页i中出现.

对于关键句的提取, TextRank相比与PageRank的不同是, 转移矩阵用句的相似度构造而不是再根据转移关系构造, Pij表示句i和句j的相似度. 至于句的相似度的计算, 是两句相同的词数/(log(句1长度)+log(句2长度)).

代码实现

本项目借鉴了TextRank4ZH的预处理方法, 计算速度, 内存, 效果, 可扩展性, 代码重用角度都要好得多.

具体来说, 有这样几个类:

- `PageRank`抽象类, 要求继承它的子类实现`build_matrix`, 提供矩阵构造方法. 此类实现了幂法进行了平稳分布的计算(`analyze`), 概率的排序, 以及返回得分最高的`num`个items(`get_top_items`).
- `TextRank4Keyword`类, 继承了`PageRank`, `build_matrix`用窗内共现次数构造(默认为5, 也就是考虑一个词的前后各2个词).
- `TextRank4Sentence`类, 继承了`PageRank`, `build_matrix`用句的相似度构造.
- `TextProcessor`类, 进行文本处理以及保存文本处理的结果. 避免让`TextRank4Keyword`和`TextRank4Sentence`存储文本. 后两者不存储文本, 只存储矩阵和状态到词/句的映射关系. 预处理包括jieba分词, 去除含有中文以外字符的内容, 去除stopword, 去除不相关词性的词.

对比TextRank4ZH, 优点在于:

- 实现了`PageRank`抽象类, 计算速度优于`TextRank4ZH`调用的`networkx`包(估计是

因为没有多余的处理).

- 代码重用, 解除耦合, 可扩展性好得多. 解除耦合体现在`TextRank4Keyword`和`TextRank4Sentence`完全不处理文本处理的逻辑. 重用体现在, 如果有用`PageRank`算法的其它场景, 只需要继承`PageRank`类添加一个矩阵构造方法. 如果用其它句相似度方法, 只需要继承`TextRank4Sentence`实现新的相似度计算方法`sentence_similarity`.
- 能用`generator`的地方尽量用`generator`, 节省内存与时间.

效果

人工测评. 测例为`data`文件夹下的`期末报告.md`. 约8000字, 238个句子.

关键词

```
[Item(item='施剑翘', score=0.014601944405066906),
 Item(item='孙传芳', score=0.011189932623783835),
 Item(item='自首', score=0.009739049179722827),
 Item(item='复仇', score=0.008940864228086282),
 Item(item='郑', score=0.006979697276119705),
 Item(item='刑法', score=0.006392415114955813),
 Item(item='施案', score=0.005436760862669225),
 Item(item='道德', score=0.005103609966671799),
 Item(item='判决', score=0.004988133380010817),
 Item(item='法律', score=0.00496282764343094)]
```

原文是探讨施剑翘复仇案, 可以看到给出的关键词非常好. 包括了本文的全部关键词, 人名`施剑翘`和`孙传芳`, `复仇`, `法律`, `道德`, `判决`等. 其中唯一不好的词是`郑`, 这是分词的原因(正确的词应是`郑继成`).

关键句

```
[Item(item=(233, '本文首先叙述了施剑翘杀孙传芳案的始末，原告和被告的辩护策略，地方法院、高级法院、最高法院的意见和判决，然后叙述了与施案相似的郑继成案和刘景桂案'), score=0.016226721100442373),
 Item(item=(206, '高级法院认为施剑翘复仇出于纯孝，动机正义，其情可悯，体现出伦常观念的影响'), score=0.011172087877497076),
 Item(item=(0, '## 从施剑翘杀孙传芳案看伦常，礼法，舆论对司法的影响'), score=0.010341450345136503),
 Item(item=(225, '施案中注意到了施剑翘自首情节的不成立，考察出施从滨之死非法，刘案中考察出了刘景桂在与逯明解除婚约后与他幽会是自愿的，"悲愤杀人"不能成立'), score=0.01026323035807473),
 Item(item=(235, '对于礼法冲突的问题，分析对比了地方法院、高级法院、最高法院的不同策略，最后比较了舆论和公众同情对地方法院、高级法院、最高法院的不同影响'), score=0.009680607496982529),
 Item(item=(99, '原告认为量刑过轻，被告认为施剑翘有自首情节，双方不服再次上诉，案子遂被移交到了南京最高法院'), score=0.009669616828726994),
 Item(item=(39, '这一方面的代表是林郁沁所著《施剑翘复仇案—民国时期公众同情的兴起与影响》研究了公众围绕着一场轰动性的审判而进行的情感化的政治参与，追溯了“公众同情”这一新型情感于20世纪前期在中国兴起的脉络'), score=0.009600148002619201),
 Item(item=(230, '公众同情对地方法院具有明显影响，但对于高级法院和最高法院，影响不明显，同时各法院的判决中都避免提及舆论'), score=0.009503071969413626),
 Item(item=(207, '最高法院更为高明，巧妙地把礼法的难题转化为程序正义的问题，即施剑翘被判处最低刑罚，主要原因是施父的死即是程序不正义的，因此施的行为应得宽恕，显示出最高法院的权威'), score=0.00924784272478158),
 Item(item=(223, '但笔者不认为高级法院和最高法院的判决受到了舆论和公众同情的太多影响'), score=0.009238410012802832)]
```

这篇期末报告最后一段是对全文的总结, 可以看到10个结果中有6句都来自这个摘要, 1句来自标题. 另外3句也有很强的相关性. 可见效果是非常理想的.

时间上来说, 主要实现都是jieba分词的用时. 比如执行`python TextRank4Sentence.py`. 可以看到计算是非常快的:

```
处理分词用时:1.20953s,计算词用时0.02682s, 计算句用时:0.05115s
```

改进

由于很好的可扩展性, 对于句的相似度, 可以考虑用sentence embedding, 继承`TextRank4Sentence`, 重写`sentence_similarity`即可.