Atitit nlp 文本相似性计算

目录

[1. 原理编 1](#_Toc27526)

[1.1. 4.1基于拼音相似度的汉语模糊搜索算法 2](#_Toc13015)

[1.2. （1）最长公共子串（基于词条空间） 2](#_Toc25916)

[1.3. 2](#_Toc25457)

[1.4. （2）最长公共子序列（基于权值空间、词条空间） 2](#_Toc23889)

[1.5. 2](#_Toc13346)

[1.6. （3）最少编辑距离法（基于词条空间） 2](#_Toc15501)

[1.7. 2](#_Toc11812)

[1.8. （4）汉明距离（基于权值空间） 2](#_Toc25737)

[1.9. 2](#_Toc23757)

[1.10. （5）余弦值（基于权值空间） 2](#_Toc10975)

[2. Apache Commons Text库计算文本相似性 3](#_Toc16509)

[3. 其他 3](#_Toc21204)

[3.1. 相同字符数 3](#_Toc27180)

[3.2. 编辑距离（edit distance） 4](#_Toc8509)

[3.2.1. 莱文斯坦距离，又称Levenshtein距离 4](#_Toc3115)

[3.2.2. Jaro距离 4](#_Toc17589)

[4. SimHash 文章相似性计算 4](#_Toc29194)

[4.1.1. 定义 4](#_Toc23646)

[4.1.2. 应用 5](#_Toc8407)

[5. 语义相似性 5](#_Toc31165)

[6. 参考: 5](#_Toc10756)

# 原理编

TFIDF

TFIDF的方法只能是算一个权重，比较文本相似度最常用的是计算余弦值，利用权重来建模型

可能我没说太明白，是用您说的方法求余弦值来作为相似度，这也只能是两两比较，做不到【批量】

如果只要是有一连串相同字节的就认为是相似, 可以这样: 把2个文件分成N份, 求出每一份的CRC32值(MD5等都行),然后判断这些校验值, 如果有任意2个相等,相似度就加一点.

如果是图像相似度的判断,比如一个人的2张照片,那就麻烦了,涉及到图像特征提取比较.

## 4.1基于拼音相似度的汉语模糊搜索算法

最长公共子序列

最小编辑距离算法

5.总结

衡量文本相似度的几种手段：

## （1）最长公共子串（基于词条空间）

## 

## （2）最长公共子序列（基于权值空间、词条空间）

## 

## （3）最少编辑距离法（基于词条空间）

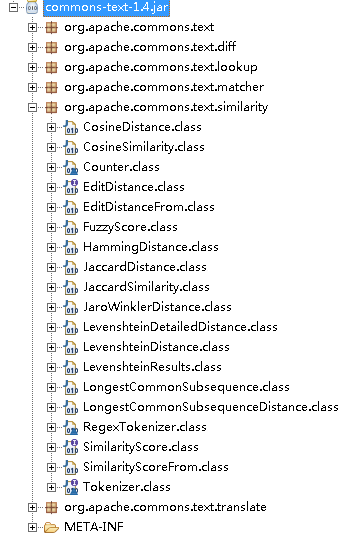
## 

## （4）汉明距离（基于权值空间）

## 

## （5）余弦值（基于权值空间）

# Apache Commons Text库计算文本相似性



# 其他

## **相同字符数**

在传统的字符串比较过程中，我们考虑字符串中每个字符是否相等，并且考虑了字符出现的顺序，如果不考虑字符出现的顺序，我们可以利用两个文本之间相同的字符数量，很简单不再赘述，可以利用common lang中的getFuzzyDistance：

int dis = StringUtils.getFuzzyDistance(term, query, Locale.CHINA);

## 编辑距离（edit distance）

### 莱文斯坦距离，又称Levenshtein距离

俄罗斯科学家弗拉基米尔·莱文斯坦在1965年提出这个概念。

实现方式

我们可以利用common lang中StringUtils的函数来计算：

int dis = StringUtils.getLevenshteinDistance(s1, s2);

### Jaro距离

定义

Jaro Distance也是字符串相似性的一种度量方式，也是一种编辑距离，Jaro 距离越高本文相似性越高;而Jaro–Winkler distance是Jaro Distance的一个变种。据说是用来判定健康记录上两个名字是否相同，也有说是是用于人口普查。从最初其应用我们便可看出其用法和用途，其定义如下

同样我们可以利用common lang中的getJaroWinklerDistance函数来实现，注意这里实现的是Jaro–Winkler distance

double dis = StringUtils.getJaroWinklerDistance(reviewName.toLowerCase(), newsName.toLowerCase())

# SimHash 文章相似性计算

### **定义**

SimHash是一种局部敏感hash，它也是Google公司进行海量网页去重使用的主要算法。  
传统的Hash算法只负责将原始内容尽量均匀随机地映射为一个签名值，原理上仅相当于伪随机数产生算法。传统的hash算法产生的两个签名，如果原始内容在一定概率下是相等的；如果不相等，除了说明原始内容不相等外，不再提供任何信息，因为即使原始内容只相差一个字节，所产生的签名也很可能差别很大。所以传统的Hash是无法在签名的维度上来衡量原内容的相似度，而SimHash本身属于一种局部敏感哈希算法，它产生的hash签名在一定程度上可以表征原内容的相似度。  
我们主要解决的是文本相似度计算，要比较的是两个文章是否相似，当然我们降维生成了hash签名也是用于这个目的。看到这里估计大家就明白了，我们使用的simhash就算把文章中的字符串变成 01 串也还是可以用于计算相似度的，而传统的hash却不行。

我们可以来做个测试，两个相差只有一个字符的文本串，“你妈妈喊你回家吃饭哦，回家罗回家罗” 和 “你妈妈叫你回家吃饭啦，回家罗回家罗”。  
通过simhash计算结果为：  
1000010010101101**1**11111100000101011010001001111100001**0**0101**1**001011  
1000010010101101**0**11111100000101011010001001111100001**1**0101**0**001011  
通过传统hash计算为：  
0001000001100110100111011011110  
1010010001111111110010110011101

通过上面的例子我们可以很清晰的发现simhash的局部敏感性，相似文本只有部分01变化，而hash值很明显，即使变化很小一部分，也会相差很大

### **应用**

simhash从最一开始用的最多的场景便是大规模文本的去重，对于爬虫从网上爬取的大规模语料数据，我们需要进行预处理，删除重复的文档才能进行后续的文本处理和挖掘，那么利用simhash是一种不错的选择，其计算复杂度和效果都有一个很好的折中。  
但是在实际应用过程中，也发现一些badcase，完全无关的文本正好对应成了相同的simhash，精确度并不是很高，而且simhash更适用于较长的文本，但是在大规模语料进行去重时，simhash的计算速度优势还是很不错的。

# **语义相似性**

在NLP中有时候我们度量两个短文本或者说更直接的两个词语的相似性时，直接通过字面距离是无法实现的，如：中国-北京，意大利-罗马，这两个短语之间的相似距离应该是类似的，因为都是首都与国家的关系；再比如（男人、男孩），（女人、女孩）应该是相同的关系，但是我们看其字面距离都是0。  
想要做到语义层面的度量，我们需要用到机器学习建模，而自然语言的问题转化为机器学习的首要问题便是找到一种方法把自然语言的符号数学化。

NLP点滴——文本相似度 - 小琪琪来啦 - 博客园.html

# 参考:

文本相似度算法 - chenglinhust的专栏 - 博客频道 - CSDN.NET.htm

paip.判断文本相似度的算法.txt

(9+条消息)Java回炉重造(三)使用Apache Commons Text库计算文本相似性\_jaccard相似系数、余弦相似度 - The quiter you become, the more you can hear. - CSDN博客.html

使用simhash以及海明距离判断内容相似程度 - 纪玉奇 - 博客园.html