

# 文本相似度的探究

## 小组成员

廖铭骞	12012919
陈纪元	11811810
郑博方	12012918
明嘉浩	12012714
叶臻铭	12011404
肖煜玮	12012902
陈一戈	12011625
王铭琬	12012540

代码附录见 [https://github.com/Juan-Chen45/Chinese\\_NLP](https://github.com/Juan-Chen45/Chinese_NLP)

## 引言

在中文信息处理的领域当中，我们经常要判断两篇文章是否相似，以及相似度是多少，并使用该类信息进行相关应用。比如，在搜索引擎的应用实现当中，常常需要根据用户键入的内容迅速地进行相似文本信息的搜索，并快速地展示在预选栏当中；在专家系统的制作以及AI客服的制作当中，会事先准备好很多经典的问题，在这之后会使用用户键入的问题与文本库当中的内容进行匹配，并且找到对应的问题进行返回，这样会大大减轻人工客服的压力，大大提升相关平台的运作效率，同时用户也能得到高效率的反馈以及良好的体验；同时在语料库的预处理当中，我们可以基于文本之间的相似程度，将重复的语料挑出来并且删除，减少不必要的重复语料；以上种种应用，都在不同的方面说明了文本相似度匹配算法的重要性，我们小组的针对文本相似度算法进行了探究、实现以及应用。

## 任务说明

一般来说，探究文本相似度，输入的通常是两个文本，文本相似度计算的任务就在于通过一定的指标去计算确认两个输入文本在语义上面的相近程度。如果文本的语义相似数值越小，则两段文本的语义之间就差异越大，反之，就说明两个文本的语义所表示的意义相近。对于中文的文本相似度任务来说，由于文本当中常常存在比较多的近义词以及缩写等等，加上句法比较多变，这些在一定程度上面都加大了文本相似度计算的难度。同时科学家们在学术界以及工业界也针对这一问题指出了很多计算模型以及算法，以下就是我们小组探究的一些模型以及算法。该模型基于这样一种假设，第N个词的出现只与前面N-1个词相关，而与其它任何词都不相关，整句的概率就是各个词出现概率的乘积。

## N-Gram(N元文法)模型

N-Gram模型的基本思想是设置大小为N的滑动窗口, 将文本内容按照字符流或者单词流的形式进行窗口滑动操作, 形成多个长度为N的文本片段, 每个片段被称为一个N元组, 然后计算给定的两个文本中公共N元组的数量与总N元组数量的比值, 以此来表征两个文本的相似度。N-Gram元语法模型是基于(n-1)阶马尔科夫链的一种概率语言模型，可以通过n个语词出现的概率来推断语句的结构，即可以通过前  $n - 1$  个词对第  $n$  个词进行预测。

通过使用 `Python` 程序，我们小组演示了如何生成一个指定文本的 `n-grams` 信息：

```

from nltk.util import ngrams

sentence = input("Enter the sentence: ")
n = int(input("Enter the value of n: "))
n_grams = ngrams(sentence.split(), n)

for grams in n_grams:
    print(grams)

```

通过输入文本 `This is the n-grams example implemented by Chinese NLP group! Yey!`，产生的输出结果如下所示：

```

(torch) PS C:\Users\86181> python -u "c:\大二课程\中文信息处理\n-gram.py"
Enter the sentence: This is the n-grams example implemented by Chinese NLP group! Yey!
Enter the value of n: 3
('This', 'is', 'the')
('is', 'the', 'n-grams')
('the', 'n-grams', 'example')
('n-grams', 'example', 'implemented')
('example', 'implemented', 'by')
('implemented', 'by', 'Chinese')
('by', 'Chinese', 'NLP')
('Chinese', 'NLP', 'group!')
('NLP', 'group!', 'Yey!')

```

可以看出，对于 `n=3` 的情况，n-grams算法不断地将长度为 `n` 的滑动窗口右移，不断地“切割”出相应的 `n` 元词组。

在《数学之美》当中，有对n-grams算法与马尔科夫链的讲述，如果我们有一个由 `m` 个词组成的序列（或者说一个句子），我们希望算得概率  $p(w_1, w_2, \dots, w_m)$ ，根据链式规则，可得

$$p(w_1, w_2, \dots, w_m) = p(w_1) * p(w_2|w_1) * p(w_3|w_1, w_2) * \dots * p(w_m|w_1, \dots, w_{m-1})$$

由于这个公式牵涉到太多的变量，所以这个概率并不好算，此时利用马尔科夫链的假设，即当前这个词仅仅跟前面几个有限的词相关，因此也就不必追溯到最开始的那个词，这样便可以大幅缩减上述算式的长度。即对于一个n元文法模型来说，有下列公式成立：

$$P(w_1, w_2, \dots, w_m) = \prod_{i=1}^m P(w_i | w_{i-n+1}, \dots, w_{i-1})$$

可以发现，这个式子的计算可以利用概率论与数理统计中学到的贝叶斯定理，对 `n = 2` 的情况进行分析可以分别求出右侧累乘的每一项对应的值

$$p(w_i | w_{i-1}) = \frac{p(w_i, w_{i-1})}{p(w_{i-1})} = \frac{\text{count}(w_i, w_{i-1})}{\text{count}(Allwords)} * \frac{\text{count}(AllWords)}{\text{count}(w_{i-1})} = \frac{\text{count}(w_i, w_{i-1})}{\text{count}(w_{i-1})}$$

其中， $\text{count}(w_i, w_{i-1})$  表示由  $w_i, w_{i-1}$  组成的二元组的数量

通过这样的计算方法，就可以通过计算出对于n元组来说，出现前 `n-1` 个字符之后，出现第 `n` 个字符的概率。

以下结合谷歌搜索的例子，详细讲述一下n-gram是如何计算工作的。



假设当前使用的是二元文法模型，则通过搜索“南科大”，谷歌浏览器会自动根据下一个短语的出现频率高低给显示的结果排序，通过如上的结果，可以判断出

$$p(\text{双一流}|\text{南科大}) > p(\text{邮箱}|\text{南科大}) > p(\text{图书馆}|\text{南科大}) > \dots > p(\text{常用系统}|\text{南科大})$$

很自然地，我们会想到  $n$  的大小对文本相似度分析结果会有什么影响呢？当  $n$  越大的时候，对可能出现的下一个词语的约束就会越强，因为需要综合考虑位于该词语之前的文本因素更多了，同时这种考虑也会使得对下一个词语的识别度更高，但是在两个不同的文本当中，很少会出现两段很长相同的文字，所以得到的匹配结果也会更加稀疏。与此相对，当  $n$  越小的时候，对下一个词语的约束力就会变弱，从而在两个不同的文本当中，会出现更多的相同的匹配结果，往往得到的结果也更为准确。

利用N-gram这种方法，我们能进行许多工作：

- **判断词性：**

当我们通过语料库计算出N-gram之后，我们拿到一句话，就可以通过计算好的模型来判断某个词的前置词的词性。

譬如：“我爱中国” 我们想判断“爱”的词性，那可以通过计算的2-gram模型得到：前面是名词的情况下“爱”出现的次数，前面是名词的情况下“爱”作为动词出现的次数，前面是名词的情况下“爱”作为名词出现的次数。

那么经过计算，我们就可以得到这句话中“爱”作为什么词性出现的概率更大，然后即可将其认为是这句话中“爱”的词性。

- **分词方法：**

通过N-gram，我们也可以改进分词方法。当我们通过语料库建立好N-gram模型之后，我们就可以对一句话中不同的分词方法的结果进行计算比对，最后选取N-gram模型认为概率最大的分词方法。

譬如：“我喜欢中文信息处理” -> {我 / 喜欢 / 中文 / 信息 / 处理} || {我 / 喜欢 / 中文信息 / 处理} || {我 / 喜欢 / 中文 / 信息处理}

那么我们可以通过N-gram模型计算这三种分词方法中不同组合的概率，然后得到最可能的分词方法。

- **机器翻译：**

当我们进行英译汉或者汉译英的时候，由于语法不同，经常会遇到词语顺序的问题。那么通过N-gram，我们可以计算出哪种顺序的概率最大，以此作为最佳的翻译结果。

譬如：“I was there for you.” -> “我在那为你” || “我为你在那” || “我为在你那” ...

这种情况下，我们可以通过N-gram来组合不同的分词顺序，并且计算概率，从而得到最优的结果。

通过利用GitHub上zheng5yu9开发的工具进行试验。

**思路：**利用jieba对文档分词，3个相邻词为一组，计算两个词的左信息熵，右信息熵，内部的凝聚度，并据此进行计算分数，根据分数大小获取新词。

效果如下所示：

```
('_重大_疾病', 0.017789747314352424)

('_保障_范围', 0.015639743403053734)

('_本_公司', 0.014212133249451173)

('_完全_丧失', 0.013672071599779227)

('_意外_伤害', 0.010722245979224557)

('_明确_诊断', 0.009062853195861094)

('_日常生活_活动', 0.008990786509666062)

('_六项_基本_日常生活', 0.008813957372202039)

('_基本_日常生活', 0.008694797110512052)

('_基本_日常生活_活动', 0.008671016020472998)

('_保险_事故', 0.008504469334120192)

('_六项_基本_日常生活_活动', 0.008471400808888209)

('_能力_完全_丧失', 0.008404916576493579)

('_全部_条件', 0.008136980840438046)

('_无法_独立', 0.008091270307811042)

('_满足_下列_全部_条件', 0.008055553080109046)

('_现金_价值', 0.007895715475057304)
```

## TF-IDF+向量空间模型

**TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率)**是一种用于资讯检索与资讯探勘的常用加权技术。TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。上述引用总结就是，**一个词语在一篇文章中出现次数越多，同时所有文档中出现次数越少，越能够代表该文章。**这也就是TF-IDF的含义。

在该算法当中，**TF (term frequency)** 指的是词频，也就是在一段给定的文本当中一个词语出现的频率，计算公式为  $TF = 1 + \log \left( \frac{\text{count}(\text{Specified word})}{\text{count}(\text{Total words})} \right)$ 。但是仅仅使用TF的缺陷也是很明显的，在一篇文章中一个词语的TF越高，代表这个词语在这篇文章中出现的频率很高，但它一定能成为这篇文章的代表词吗？答案是否定的，比如在一个有关教育的文章合集（包含N篇文章）中，"学校"这个词可能会在里面每一篇文章中都出现并且频率很高，但是它的高频率带来的副作用就是信息量的降低，每篇文章里

面都有大量的"学校"这个词，那你自然不能选择它作为任意一篇文章的代表词。由此，我们引入**IDF (Inverse Document Frequency)**。IDF指的是逆文档频率，计算公式为 $IDF = \log \left( \frac{\text{语料库文档总数量}}{\text{含有指定词文档的数量} + 1} \right)$ ，注意此处分母加一是为了避免分母为0。通过这个公式可以看出，如果一个词语越常见，也包含该指定词的文档数量也会越多，分母随着变大，IDF也越接近0，越小。因此IDF代表着一个词在所有文档中出现的频率，频率越小，代表它越是一个罕见词，那么它所包含的信息量就越大。


因此**TF-IDF**算法其实是一个权衡，对于一篇文章中的代表词，我们既希望它在这篇文章中出现的频率高，又希望这个词只在这篇文章中出现。最终**TF-IDF**权重的计算公式为 $TF - IDF = TF * IDF$ ，这个值随着指定词在单个文档中的频率增多而变大，而随着该指定词在整个语料库中的出现次数成反比。当我们需要选出一篇文章中的关键词时，这个算法就是通过计算出文档中每一个词语的 $TF - IDF$ 值，之后按照降序排列，最后取出排在最前面的几个词。现成的Python包中如jieba, nltk中都包含了TF-IDF的实现，这里我们就不展示了。

接着我们这里引申一点，**TF-IDF**可以结合空间向量模型进行小型搜索引擎的开发。这里简单借鉴中科大俞能海老师，南科大宋轩老师的课件内容简单阐述一下。

## Term frequency ( tf )

- The term frequency  $tf_{t,d}$  of term  $t$  in document  $d$  is defined as the number of times that  $t$  occurs in  $d$ .

Document 1	
foo	
bar	
foobar	
foo	
foo	
footbar	
bar	



t	$tf_{t,d}$
bar	2
foo	3
foobar	2

## Term frequency ( tf )

- We want to use **tf** when computing query-document match scores.  
But:
  - Raw term frequency is not what we want:

t	tf <sub>t,docA</sub>	tf <sub>t,docB</sub>
bar	10	1

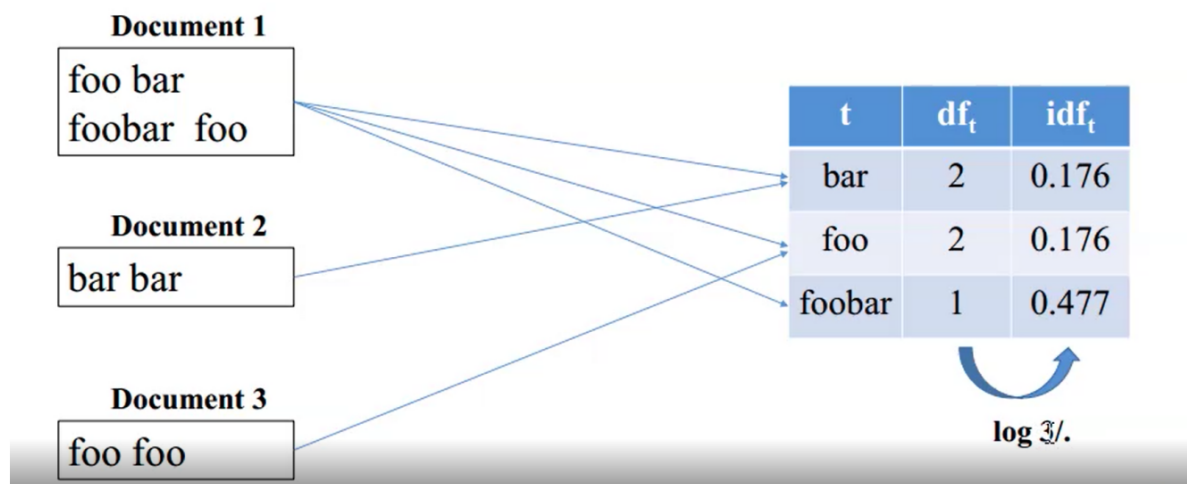


score	A	B
	10	1

But document A is not 10 times more relevant than document B.



## Document frequency (df)



## tf-idf weighting

- The tf-idf weight of a term is the product of its **tf** weight and its **idf** weight.

$$w_{t,d} = (1 + \log_{10} \text{tf}_{t,d}) \times \log_{10} (N / \text{df}_t)$$

- **Increases** with the number of occurrences within a document.
- **Increases** with the rarity of the term in the collection.



# 回顾（p3）：词项-文档关联矩阵

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth ...
ANTHONY	5.25	3.18	0.0	0.0	0.0	0.35
BRUTUS	1.21	6.10	0.0	1.0	0.0	0.0
CAESAR	8.59	2.54	0.0	1.51	0.25	0.0
CALPURNIA	0.0	1.54	0.0	0.0	0.0	0.0
CLEOPATRA	2.85	0.0	0.0	0.0	0.0	0.0
MERCY	1.51	0.0	1.90	0.12	5.25	0.88
WORSER	1.37	0.0	0.11	4.15	0.25	1.95
...						

每篇文档表示成一个基于tf-idf权重的实值向量  $\in \mathbb{R}^M$

当每一篇文档都被表示为一个基TF-IDF的向量的时候，我们可以将输入的query同样计算出一个TF-IDF向量，两个向量之间可以采用最基础的相似度计算（余弦距离，欧几里得距离等）来衡量query和document的相似度，并根据相似度来进行搜索。在这次作业里我们自己实现了一个使用TF-IDF+空间向量模型的小搜索引擎。代码见 [https://github.com/Juan-Chen45/Chinese\\_NLP](https://github.com/Juan-Chen45/Chinese_NLP)。

```
In [71]: query = input("请输入你想要查询的内容")
print()
query = list(jb.cut_for_search(query))
query = [w for w in query if w not in stop_words]
# 返回前三(或唯一)个搜索结果,但是我们只展示最相关的那篇文章
doc_idx = search(query)
if len(doc_idx) == 0:
    print("很抱歉, 搜索不到相关内容(例子: 可以搜索 "我认为中东立场不好" 试试! )")
else:
    with open(all_files[doc_idx[0]], mode="r", encoding="utf-8") as f:
        print(f.read())
```

请输入你想要查询的内容运动生理学

09年成考北京250余名运动员参加单独招生考试

2009年国家体育总局成人高等教育运动训练专业单独招生考试于11月7日、8日举行，北京考区共250多名考生参加。

记者在北京市教育考试院考点看到，设有考场的楼层入口处有保安人员现场执勤。所有考生、监考人员凭证件才可入内。楼道内，有巡视人员检查；考场内，数十名运动员考生在专心答题，两名监考人员认真履行着监督职责。

今年成考中，北京地区只有北京体育大学通过单独招考方式招收运动员考生。该考点负责人介绍，全国共有1000余名考生报考，其中北京地区有250余名。考生报考专业均为运动训练专业，有高职和专升本两个学习层次。考生根据学习层次选择考试科目，其中专升本层次为运动生理学、运动心理学、运动训练学和综合能力4科，高职层次为语文、数学、外语和政治4科。该考点参加考试的考生中，200余名为高职层次，50余名为专升本层次。

据了解，国家体育总局的成人高等教育运动训练专业单独招考，一般在全国成人高校统一招生考试后举行。优秀运动队(省、自治区、直辖市、计划单列市、大军区、各行业体协)的教练员、运动员以及运动学校和业余体校的教练员均可参加。该招生专业只设置了高职和专升本两个层次。

相关链接

单独招考考生可不用参加全国统考。单独招考的学校单列招生计划，考试单独命题、单独录取。目前北京地区开展的成人高考单独招考项目有两类，一类是北京联合大学特殊教育学院针对残疾考生进行的单独招考，另一类是国家体育总局对运动员进行的单独招考。参加单独招考的考生，可享受一定的学费、录取等优惠政策。单独招考一般有固定的专业，同时对报考人员身份有一定限制。

缺陷：

我们在这里简单使用**TF-IDF**+空间向量模型构建出来的搜索引擎利用属于词袋模型(Bag of Words), 就是搜索的时候我们无法不考虑词出现在文档中的顺序。比如说我们搜索关键词 "猫和狗" 和 我们搜索关键词 "狗和猫" 效果是一样的。

## Simhash 算法

之前提到的tf-idf是一种描述文章（字符串）特征的方法，可以用于比较短文本和文章相关性，以用于构建搜索引擎。

这里的simhash也是描述文章特征的方法，它将一篇文章映射到一个simhash签名，可以用于比较文章之间的相似性。

对于 Simhash 算法，大致的实现思路如下: 先将要对比的文章中的关键词提取出来，选取某些词作为特征词，并为其赋权重。为每个特征词随机生成64位（位数以需求而定，这里采取64位是因为笔者在c语言中可以用long long来存储）hash串，hash串的每一位按特征词的权重加权。最后将这些hash串相加，再映射到一个简单、便于比较的字符串，这个串就是simhash签名。

### 一种可行的实现方式

这里介绍一种具体的实现方式，但很多其他尝试仍是可以作为替代的。

#### 1. 分词（用以提取关键词）

先将文章分词，删除掉停用词（即平凡的、不能体现出特征的词，一般直接使用停用词表），按tf-idf的权重选出排名最前的一些词（数量以需求而定），并按排名赋权重（对不同的文章权重不同）。

#### 2.生成hash串

为每个特征词随机生成64位01串。对某一篇文章，按以下方式对01串加权：某一位为0则改为该特征词权重  $\times (-1)$ ，某一位1则改为 该特征词权重  $\times (+1)$ 。（这里的随机方法不作限制；生成其他类型的hash串的优劣有待考量）

#### 3.映射到simhash串

对某一篇文章，将每个特征词求出的hash串相加，某一位为正则映射到1，某一位为负则映射到0。这样就将hash串映射到了一个64位的01-simhash签名。（生成其他类型的simhash串的优劣有待考量）

### 使用simhash签名进行比较

对两篇文章，比较它们的simhash签名有多少位不同，认为3位以上不同则两篇文章不想似，否则相似。也可以根据simhash签名的相似关系建立序数性的映射，从而比较各种文章之间的相似度关系。

### simhash的特点

文本内容较长时，simhash比较的准确率高。处理短文本时准确率难以保证，可以辅以更复杂但更精确的比较方法来处理。Simhash还有一个特点是，对文本的“相同”与否特别敏感：当两篇文档相同时，相似度为1；当其中一篇略有不同，相似度会有明显降低。因此非常适合用来判断两篇文档内容是否相同。并且simhash的计算比较简单，速度上有一定优势。如果配合一定的检索策略来召回候选相似文档，simhash可以用来对海量文档进行去重——这就是simhash最常见的一个应用场景。



