TF-IDF原理与实践

# 1、TF-IDF原理

TF-IDF通常应用于文本关键词提取。要提取一个文章的关键词，一个容易想到的思路就是找到出现次数最多的几个词。这是因为如果某个词很重要，它应该在这篇文章中多次出现。于是，我们进行"词频"（Term Frequency，缩写为TF）统计。

然而，出现次数最多的词是----"的"、"是"、"在"----这一类最常用的词，无法代表文章的关键词。这种类型的词叫做"停用词"（stop words），表示对找到结果毫无帮助、必须过滤掉的词。

在把“停用词”全部过滤掉之后。按照“词频”统计得到的频次最多的几个词就可以代表一篇文章的关键词吗？此时，还需要考虑到这些高频词是否是在其他文章中很少出现，只有满足这两个条件，得到的词才是代表这篇文章的关键词。

所以，需要一个重要性调整系数，衡量一个词是不是常见词。如果某个词比较少见，但是它在这篇文章中多次出现，那么它很可能就反映了这篇文章的特性，正是我们所需要的关键词。

用统计学语言表达，就是在词频的基础上，要对每个词分配一个"重要性"权重。最常见的词（"的"、"是"、"在"）给予最小的权重，较常见的词给予较小的权重，较少见的词给予较大的权重。这个权重叫做"逆文档频率"（Inverse Document Frequency，缩写为IDF），它的大小与一个词的常见程度成反比。

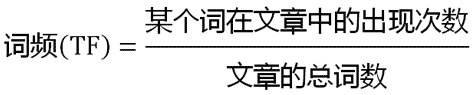
知道了"词频"（TF）和"逆文档频率"（IDF）以后，将这两个值相乘，就得到了一个词的TF-IDF值。某个词对文章的重要性越高，它的TF-IDF值就越大。所以，排在最前面的几个词，就是这篇文章的关键词。

下面就是这个算法的细节。

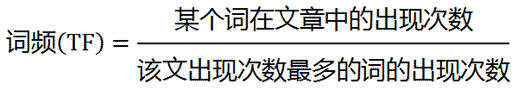
**第一步，计算词频。**

词频(TF) = 某个词在文章中出现的次数

考虑到文章有长短之分，为了便于不同文章的比较，需要进行"词频"标准化。

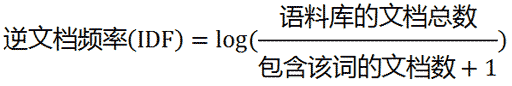


或者



**第二步，计算逆文档频率。**

这时，需要一个语料库（corpus），用来模拟语言的使用环境。



如果一个词越常见，那么分母就越大，逆文档频率就越小越接近0。分母之所以要加1，是为了避免分母为0（即所有文档都不包含该词）。log表示对得到的值取对数。

**第三步，计算TF-IDF。**



可以看到，TF-IDF与一个词在文档中的出现次数成正比，与该词在整个语言中的出现次数成反比。

# TF-IDF应用

## 关键词提取

TF-IDF在关键词提取方面的应用非常广泛，其实就是先对文本分词，然后计算每个词的TF-IDF，一般而言TF-IDF值比较大的几个词就是本篇文章的关键词。

## 信息检索

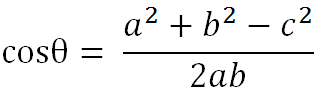
上文已经说到，采用TF-IDF可以得到一篇文档的关键词，当两篇文档的词非常相似的时候，我们认为两篇文档也是相似的，这是采用TF-IDF进行信息检索的基本思路。

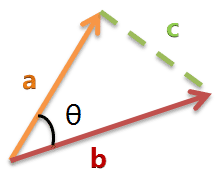
首先看看一种只使用词频的方式计算两篇文档相似的过程：

1. 分词
2. 计算词频
3. 得到词频向量
4. 根据得到的词频向量，采用余弦相似度计算两篇文档的相似度

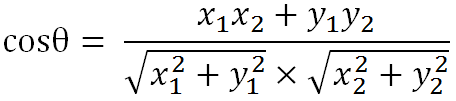
这里列出余弦公式的几种表示方法：

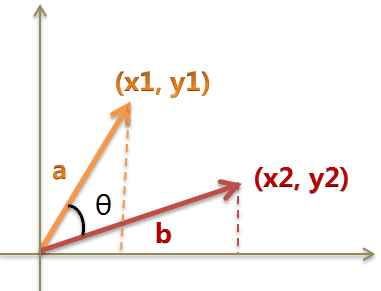
* 已知向量求向量相似：



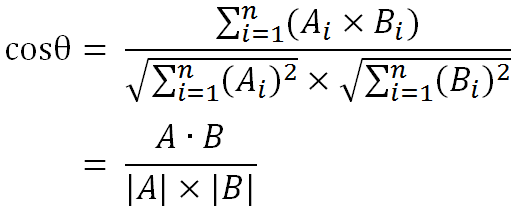


* 已知向量坐标，a向量是[x1, y1]，b向量是[x2, y2]，则余弦公式可以表示为：





* 两个n维向量，A是 [A1, A2, ..., An] ，B是 [B1, B2, ..., Bn]，则余弦公式为：



由此，我们就得到了"找出相似文章"的一种算法：

（1）使用TF-IDF算法，找出两篇文章的关键词；

（2）每篇文章各取出若干个关键词，合并成一个集合，计算每篇文章对于这个集合中的词的词频（为了避免文章长度的差异，可以使用相对词频）；

（3）生成两篇文章各自的词频向量；

（4）计算两个向量的余弦相似度，值越大就表示越相似。

# TF-IDF优缺点

**优点**：简单快速，结果比较符合实际情况。

**缺点**：单纯以"词频"衡量一个词的重要性，不够全面，有时重要的词可能出现次数并不多。而且，这种算法无法体现词的位置信息，出现位置靠前的词与出现位置靠后的词，都被视为重要性相同，这是不正确的。（一种解决方法是，对全文的第一段和每一段的第一句话，给予较大的权重。）

# TF-IDF与TextRank算法的比较

1. 从算法原理上来看，基础都是词频统计，只是TD-IDF通过IDF来调整词频的权值，而TextRank通过上下文的连接数来调整词频的权值。TextRank通过滑动窗口的方式，来实现词的位置对词的权值的影响。
2. TD-IDF计算简单，运行性能更好。

# 实践

包含纯python实现、gensim实现和sklearn实现三种。见github：

<https://github.com/jpegbert/NLP_Coding/tree/master/tf_idf/tf_idf1>

参考

<http://www.ruanyifeng.com/blog/2013/03/tf-idf.html>

<http://www.ruanyifeng.com/blog/2013/03/cosine_similarity.html>