Lucene 打分算法

Email: zhangyi.zjuer@gmail.com QQ:259503698

Lucene 打分公式

$$score(q,d) = coord(q,d) \times queryNorm(q) \times \sum_{t\ in\ q} (tf(t\ in\ d) \times idf(t)^2 \times t.getBoost() \times norm(t,d))$$

公式中各个部分的含义：

- *t*：Term，这里的 Term 是指包含域信息的 Term，也即 title:hello 和 content:hello 是不同的 Term
- *coord(q,d)*：一次搜索可能包含多个搜索词，而一篇文档中也可能包含多个搜索词，此项表示，当一篇文档中包含的搜索词越多，则此文档则打分越高。只会在 BooleanQuery 的 OR 查询中该值可能小于 1。其余查询该值都为 1
- *queryNorm(q)*：计算每个查询条目的方差和，此值并不影响排序，而仅仅使得不同的 query 之间的分数可以比较。其公式如下：

$$queryNorm(q) = \frac{1}{\sqrt{\sum_{t\ in\ q} (idf(t) \times t.getBoost())^2}}$$

- *tf(t in d)*：关于 Term t 在文档 d 中出现的词频(*freq*)的一个值
  - 通过 Similarity 类中的 `public float tf(float freq)`方法获取，该方法为抽象方法
  - Lucene 中该方法的默认的实现使用 DefaultSimilarity 的方法：$tf = \sqrt{freq}$

- *idf(t)*：关于 Term t 在几篇文档（*docFreq*）中出现过以及总文档数（*numNocs*）的一个值
    - 通过 Similarity 类中的 `float idf(int docFreq, int numDocs)`方法获取，该方法为抽象方法
    - Lucene 中该方法的默认的实现使用 DefaultSimilarity 的方法：$1+\ln\left(\frac{numDocs}{1+docFreq}\right)$
- *norm(t,d)*：标准化因子

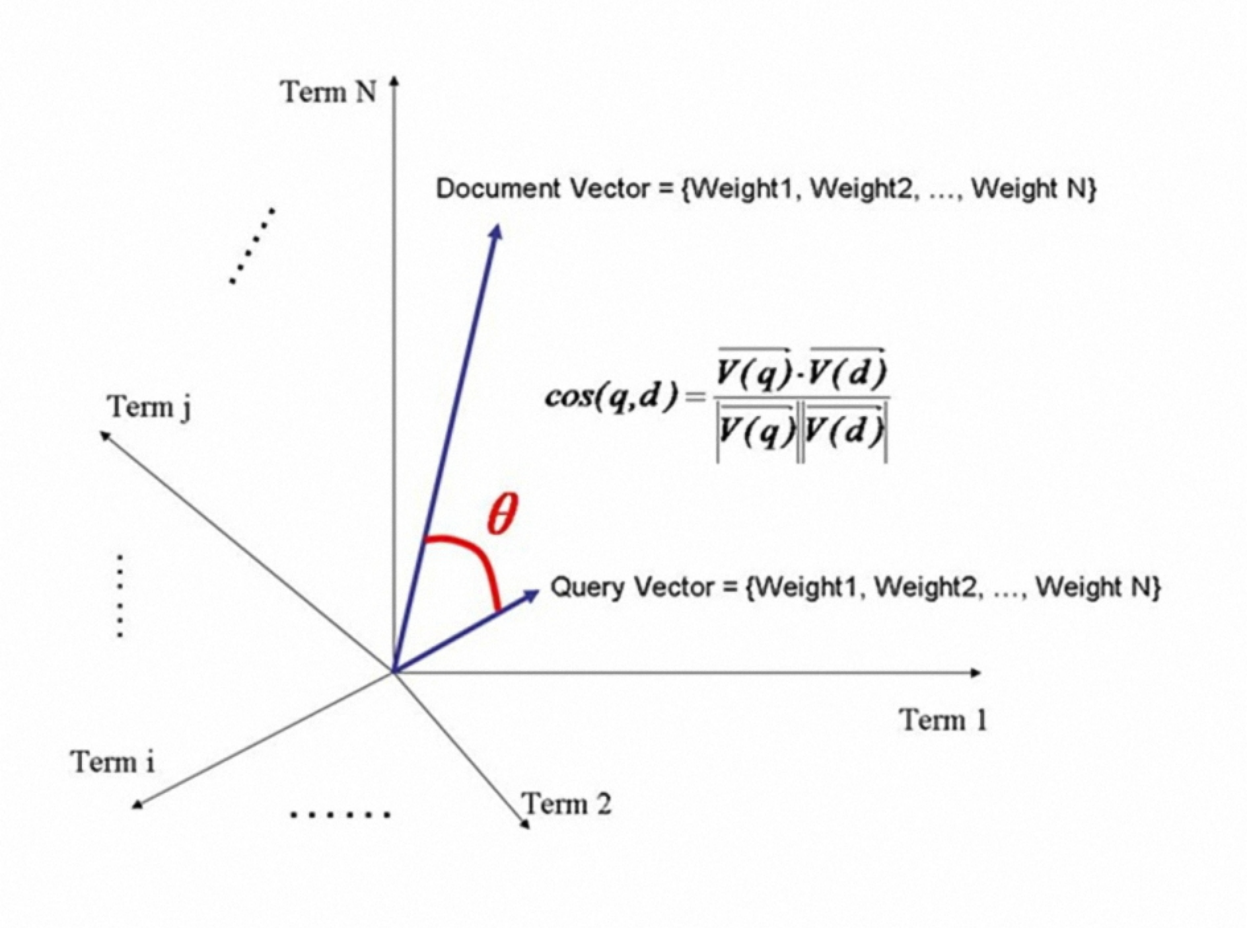
  $norm(t,d) = d.getBoost() \times lengthNorm(field) \times f.getBoost()$
    - *d.getBoost()*：Document boost，此值越大，说明此文档越重要。
    - *f.getBoost()*：Field boost，此域越大，说明此域越重要。
    - *lengthNorm(field)*：一般来说，一个域中包含的 Term 总数越多，也即文档越长，此值越小，文档越短，此值越大
        - 通过 Similarity 类中的 `float lengthNorm(String fieldName, int numTerms)`获取，该方法为抽象方法
        - Lucene 中该方法的默认实现使用 DefaultSimilarity 的方法：$\frac{1}{\sqrt{numTerms}}$
    - 通过 Similarity 类中的 `float computeNorm(String field, FieldInvertState state)`获取，该方法为抽象方法。参数 state 中包含各 boost 乘积的信息
- 各类 Boost 值
    - *t.getBoost()*：查询语句中每个词的权重，可以在查询中设定某个词更加重要，common^4 hello
    - *d.getBoost()*：文档权重，表明某些文档比其他文档更重要。
    - *f.getBoost()*：域的权重，表明某些域比其他的域更重要。

Lucene 打分公式推导

首先让忽略所有的 boost，因为这些属于人为的调整，也省略 coord，这和公式所要表达的原理无关。得到下面的公式：

$$scord(q,d) = \frac{1}{\sqrt{\sum_{t\ in\ q} idf(t)^2}} \times \sum_{t\ in\ q} (tf(t\ in\ d) \times idf(t)^2 \times \frac{1}{\sqrt{num\ of\ terms\ in\ field\ f}})$$

把文档看作一系列词(Term)，每一个词(Term)都有一个权重(Term weight)，不同的词(Term)根据自己在文档中的权重来影响文档相关性的打分计算。



把所有此文档中词(term)的权重(term weight) 看作一个向量

Document = {term1, term2, …… ,termN}

Document Vector = {weight1, weight2, …… ,weightN}

同样我们把查询语句看作一个简单的文档，也用向量来表示。

Query = {term1, term2, …… , termN}

Query Vector = {weight1, weight2, …… , weightN}

把所有搜索出的文档向量及查询向量放到一个 N 维空间中，每个词(term)是一维

两个向量之间的夹角越小，则相关性越大。

所以计算夹角的余弦值作为相关性的打分，夹角越小，余弦值越大，打分越高，相关性越大。

余弦公式如下： $score(q,d) = \cos(\theta) = \frac{V(q).V(d)}{|V(q)||V(d)|}$

查询向量：*V(q)= <w(t1, q), w(t2, q), ……, w(tn, q)>*

文档向量：*V(d)= <w(t1, d), w(t2, d), ……, w(tn, d)>*

向量空间维数为 n，是查询语句和文档的并集的长度，当某个 Term 不在查询语句中出现的时候，

*w(t, q)*为零，当某个 Term 不在文档中出现的时候，*w(t, d)*为零。

w 代表 weight，计算公式为： $w = tf \times idf$

计算余弦公式的分子部分，即两个向量的点积：

*V(q).V(d) = w(t1, q)*w(t1, d) + w(t2, q)*w(t2, d) + …… + w(tn ,q)*w(tn, d)*
*= tf(t1, q)*idf(t1, q)*tf(t1, d)*idf(t1, d) + tf(t2, q)*idf(t2, q)*tf(t2, d)*idf(t2, d) + …… +*
*tf(tn ,q)*idf(tn, q)*tf(tn, d)*idf(tn, d)*

在这里有三点需要指出：

- 由于是点积，则此处的 *t1, t2, ……, tn* 只有查询语句和文档的并集有非零值，只在查询语句出现的或只在文档中出现的 Term 的项的值为零。
- 查询语句中的每个词都是独立，所以 *tf(t, q)*都为 1

- idf 是指 Term 在多少篇文档中出现过，其中也包括查询语句这篇小文档，因而 idf(t, q)和 *idf(t, d)*是一样的。总的文档数是索引中的文档总数加一，当索引中的文档总数足够大的时候，查询语句这篇小文档可以忽略，因而可以假设 *idf(t, q) = idf(t, d) = idf(t)*

基于上述三点，点积公式为：

*V(q).V(d) = tf(t1, d) * idf(t1) * idf(t1) + tf(t2, d) * idf(t2) * idf(t2) + …… + tf(tn, d) * idf(tn) * idf(tn)*

所以余弦公式变为：

$$score(q,d) = \cos(\theta) = \frac{V(q).V(d)}{|V(q)||V(d)|} = \frac{1}{|V(q)|} \times \sum_{t\ in\ q} tf(t,d) \times idf(t)^2 \times \frac{1}{|V(d)|}$$

查询语句中 tf 都为 1，idf 的计算都忽略查询语句这篇文档，得到如下公式

$$|V(q)| = \sqrt{\sum_{t\ in\ q} w(t,q)} = \sqrt{\sum_{t\ in\ q} (tf(t,q) \times idf(t,q))^2} = \sqrt{\sum_{t\ in\ q} idf(t,q)^2}$$

本来文档的长度计算公式为：

$$V(d) = \sqrt{\sum_{t\ in\ q} w(t,q)^2}$$

在默认状况下，Lucene 采用 DefaultSimilarity，消除文档域长度带来的影响，认为在计算文档的向量长度的时候，每个 Term 的权重就不再考虑在内了，而是全部为一，并且一个 term 只会存在于一个 field，所以文档长度的计算公式为：

$$V(d) = \sqrt{\sum_{t\ in\ q} w(t,q)^2} = \sqrt{num\ of\ terms\ in\ field\ f}$$

将 V(d)代入公式得到

$$score(q,d) = \frac{1}{\sqrt{\sum_{t\ in\ q} idf(t)^2}} \times \sum_{t\ in\ q} (tf(t\ in\ d) \times idf(t)^2 \times \frac{1}{\sqrt{num\ of\ terms\ in\ field\ f}})$$

再将人为加入的各个 boost 及 coord(q,d)代入到公式中就得到了 lucene 的打分公式

Lucene 中 lengthNorm 的存储方式

Lucene 在建索引的时候就将 lengthNorm 的值存储在索引中。在建索引时可以通过 Field.Index.ANALYZED 将生成 lengthNorm 或者 Field.Index.ANALYZED_NO_NORMS 取消 lengthNorm，取消后 lengthNorm 的值固定为 1。

Lucene 采用一个 byte 存储 lengthNorm，而 lengthNorm 的值为 float 类型，这样就会造成精度的丢失。

Lucence 使用 Smallfloat 类中的 `byte floatToByte315(float f)`对 float 类型的数值进行编码，使用 `float byte315ToFloat(byte b)`解码。该方法使用对 float 类型的二进制码保留三位有效数字的形式。

例如：

一个 field 的 term 数量为 7，lengthNorm 的精确值为：$\frac{1}{\sqrt{7}} = 0.37796447$

将该数值的二进制值为：0.01100000110000100100011110001101

保留三位有效数字后为：0.0110

这样实际存储的值为：$0.0110_2 = 0.375_{10}$

Lucene 采用这样的方式存储的优点是节省了存储空间，缺点就是精度丢失

例如一个 term 数量为 6，lengthNorm 的精确值为：$\frac{1}{\sqrt{6}} = 0.40824829$

其二进制的表示为：0.01101000100000101111010110111111

取三位有效数字为：0.0110

最后存储的值也是 0.375，与 term 数量为 7 时相同

Lucene 的查询过程

Lucene 的查询过程分为三个部分：

- Query：查询词分析，获取 query 中的各 term 以及这些 term 的 boost 信息
    - `Weight createWeight(Searcher searcher)`：生成 weight
- weight：对 query 中各个 term 进行量化，在与索引中文档进行比对时可以重复使用
    - `float sumOfSquaredWeights()`： $queryNorm = \frac{1}{\sqrt{sumOfSquaredWeights}}$
    - `normalize(float norm)`：对整体权重的格式化，计算公式中每个 term 的静态变量 idf 和 boost
- Score：获取 tf，并根据公式计算得分

TermQuery

Lucene 中最基本的查询方式是 TermQuery，其它与匹配度相关的查询都是基于 TermQuery。

公式中各部分的值：

$coord(q,d)=1$

$$queryNorm = \frac{1}{\sqrt{idf^2 \times t.getBoost()^2}} = \frac{1}{idf \times t.getBoost()}$$

代入计算公式得到：

$$\begin{aligned} score(q,d) &= \frac{1}{idf \times t.getBoost()} \times tf \times idf^2 \times t.getBoost() \times norm(t,d) \\ &= tf \times idf \times norm(t,d) \end{aligned}$$

BooleanQuery

通过公式可以看出，BooleanQuery 中可以看成是多个 TermQuery 叠加的结果。与 TermQuery 不同的时 BooleanQuery 在计算 queryNorm 时需要计算 query 中的所有 term 的 idf 平方和。最后的是将各个 TermQuery 获得的 score 相加

在 booleanQuery 中如果使用 OR 查询会出现 coord(q, d)小于 1 的情况，该值通过 Similartiy 类中的 `float coord(int overlap, int maxOverlap)`计算，该方法为抽象方法，Lucene 默认使用 DefaultSimilarity 中的实现：$\frac{overlap}{\max Overlap}$

PhraseQuery

PhraseQuery 在查询的形式上类似于 BooleanQuery 的 AND 查询在计算得分的时候类似于 TermQuery。

PhraseQuery 要求查询的 field 上必须包含所有的查询词，并且这些查询词的间距小于给定的值。PhraseQuery 将查询短语看成一个查询词，它计算得分的公式与 TermQuery 相同

$score(q,d) = tf \times idf \times norm(t,d)$

这里的 idf 是每个 term 的 idf 加和。

tf 通过 phraseScore 的 phraseFreq()方法获得：$tf = \sqrt{phraseFreq}$

tf 与由各个 term 间的距离(slop)决定

Lucene 中 slop 的概念

比如想查“big car”这个短语，那么如果待匹配的 document 的指定项里包含了"big car"这个短语，这个 document 就算匹配成功。可如果待匹配的句子里包含的是“big black car”，那么就无法匹配成功了，如果也想让这个匹配，就需要设定 slop，先给出 slop 的概念：slop 是指两个项的位置之间允许的最大间隔距离，下面举例来解释：

待匹配的句子是：***the quick brown fox jumped over the lazy dog.***

例 1： 如果想用“***quick fox***”来匹配出上面的句子，发现原句里是 ***quick [brown] fox***，就是说和“***quick fox***”中间相差了一个单词的距离，所以，我这里把 slop 设为 1，表示 ***quick*** 和 ***fox*** 这两项之间最大可以允许有一个单词的间隔，这样所有“***quick [***] fox***”就都可以被匹配出来了。

例 2：如果想用“***fox quick***”来匹配出上面的句子，这也是可以的，不过比例 1 要麻烦，需要看把“***fox quick***”怎么移动能形成“***quick [***] fox***”，如下表所示，把 ***fox*** 向右移动 3 次即可：

	fox	quick		
1		fox|quick		
2		quick	fox	
3		quick		fox

例 3：如果想用“***lazy jumped quick***”该如何匹配上面的句子呢？这个比例 2 还要麻烦，要考虑 3 个单词，不管多少个单词，slop 表示的是间隔的最大距离，

lazy jumped:原句是 ***jumped [over] [the] lazy***，就是说它们两个之间间隔了 2 个词,如下所示：需要把 ***lazy*** 向右移动 4 位

	lazy	jumped			
		lazy|jumped			
		jumped	lazy		
		jumped		lazy	
		jumped			lazy

- ***lazy jumped quick：***主要看 ***lazy*** 和 ***quick***，但是由于 ***jumped*** 是在中间，所以移动的时候还是要把 ***jumped*** 考虑在内，原句里 ***lazy*** 和 ***quick*** 的关系是：***quick [brown] [fox] [jumped] [over] [the] lazy ，quick lazy*** 中间间隔了 5 个词，所以如下图所示，把 lazy 向右移动 8 次

	lazy	jumped	quick						
1		lazy|jumped	quick						
2		jumped	lazy|quick						
3		jumped	quick	lazy					
4		jumped	quick		lazy				
5		jumped	quick			lazy			
6		jumped	quick				lazy		
7		jumped	quick					lazy	
8		jumped	quick						lazy

- 最后是 ***jumped qucik*** 需要把 jumped 向右移动 4 次。

综合以上 3 种情况，所以需要把 slop 设为 8 才令“***lazy jumped quick***”可以匹配到原句。

词频的计算

Lucene 通过 phraseFreq()方法获得词频。该方法调用 DefaultSimilarity 中的 `float sloppyFreq(int distance)`获取单次词频，具体计算方法是 $\frac{1}{\text{distance}+1}$。总的词频是各个词频的加和

例如设定查询短语为”hello world”~2 表示 hello 和 world 间最大距离为 2

当该短语去匹配“hello test wold hello test test”时，

有一个短语距离为 1：hello test world

有一个短语距离为 2：world hello

其词频为 1/2+1/3= 0.833333