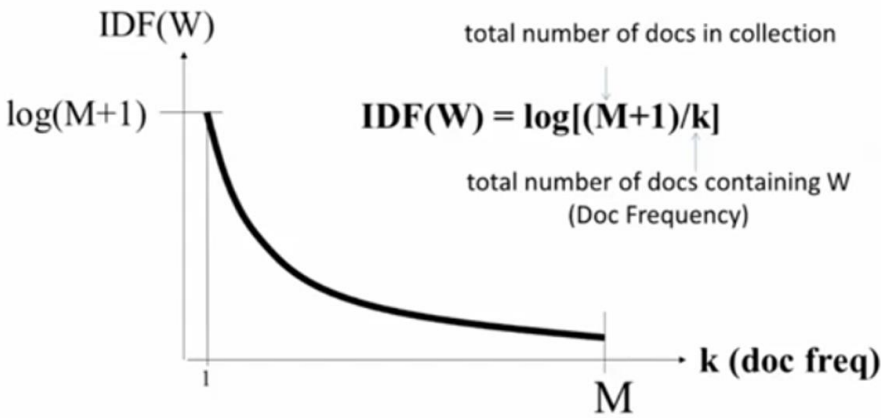


山东大学 计算机科学与技术 学院

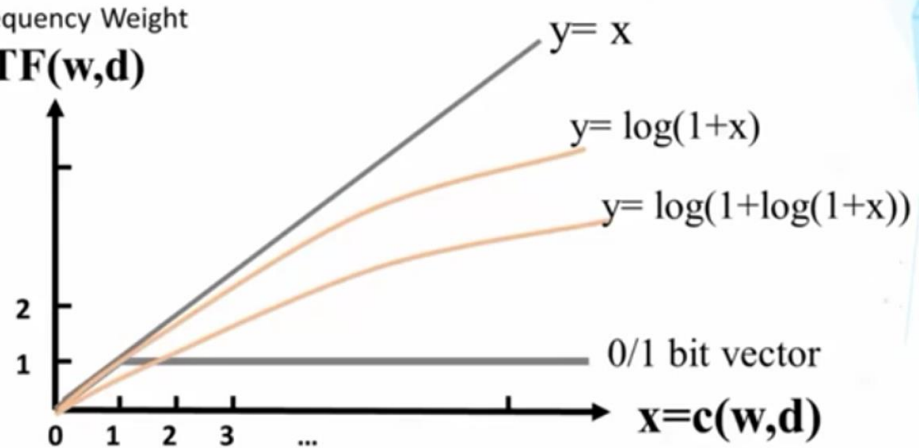
信息检索 课程实验报告

学号：201600150109	姓名：沈棋韬	班级：16 人工智能班
实验题目：Pivoted Length Normalization VSM and BM25		
<p>实验内容：</p> <ul style="list-style-type: none">实现 Pivoted Length Normalization VSM;实现 BM25;改进 Postings: (docID, Freq), 不仅记录单词所在的文档 ID, 也记录其在文档中的 Frequency;构建 inverted index 时, 记录文档的长度, 以及计算 average document length (avdl)		
<p>实验过程中遇到和解决的问题：</p> <p>计算 IDF 公式：</p> <div data-bbox="293 929 1402 1547"><p>IDF Weighting: Penalizing Popular Terms</p></div>		
<p>在 VSM 中使用取 log 的方式来对词频加权：</p>		

TF Transformation: $c(w,d) \rightarrow TF(w,d)$

Term Frequency Weight

$$y = TF(w,d)$$



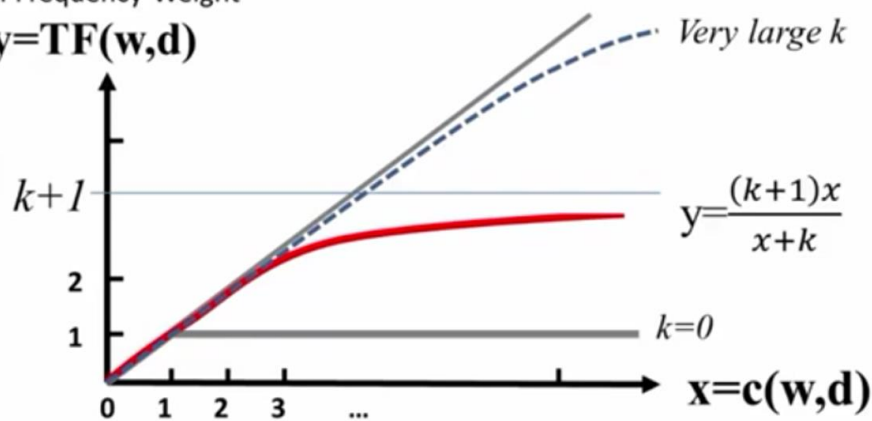
5

在 BM25 中，使用一个系数 k 来对词频加权：

TF Transformation: BM25 Transformation

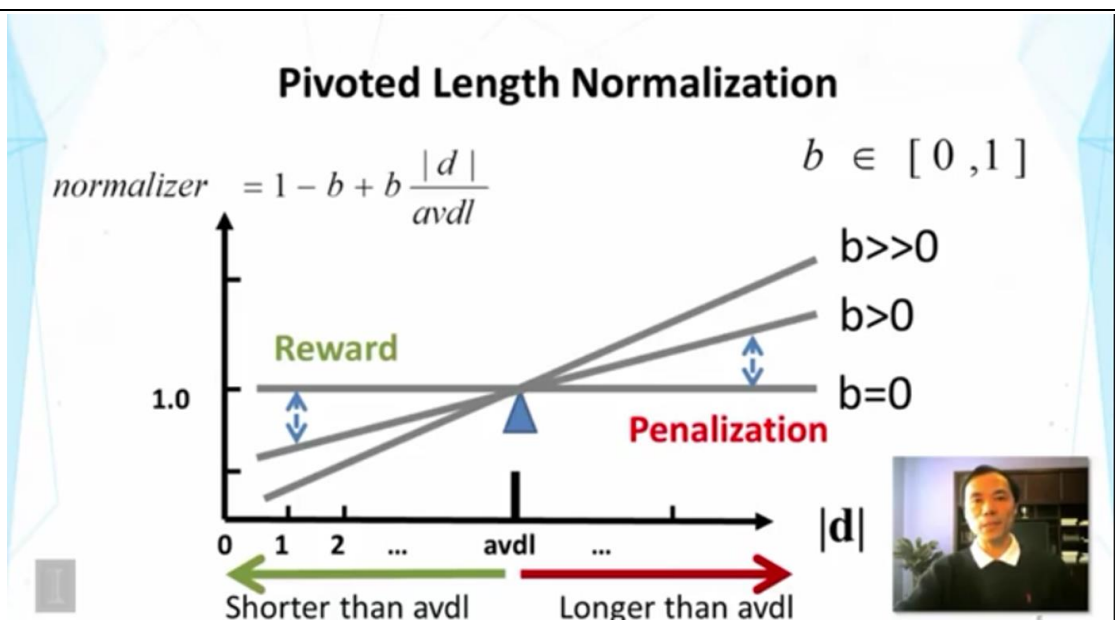
Term Frequency Weight

$$y = TF(w,d)$$



6

为了解决长文章更有可能取得更高分数的问题，需要引入一个惩罚量，对文章长度大于平均的文章进行惩罚，而对短于平均长度的文章进行奖励。



最后的公式为：

State of the Art VSM Ranking Functions

- Pivoted Length Normalization VSM [Singhal et al 96]

$$f(q, d) = \sum_{w \in q \cap d} c(w, q) \frac{\ln[1 + \ln[1 + c(w, d)]]}{1 - b + b \frac{|d|}{avdl}} \log \frac{M + 1}{df(w)}$$
- BM25/Okapi [Robertson & Walker 94]

$b \in [0, 1]$
 $k_1, k_3 \in [0, +\infty)$

$$f(q, d) = \sum_{w \in q \cap d} c(w, q) \frac{(k + 1)c(w, d)}{c(w, d) + k(1 - b + b \frac{|d|}{avdl})} \log \frac{M + 1}{df(w)}$$

在代码实现中，分别建立两个 dictionary，一个来保存一个词的词频和包含它的文章数量，另一个用来保存包含这个词的文章序号和对应的词频。

如下：

Word_dic[beef]的结果如下

```
In [3]: word_dic['beef']
Out[3]: [127, 107]
```

说明 beef 这个词一共出现了 127 词，有 107 个文档包含这个词。

Posting[beef]结果如下

```
In [4]: posting['beef']
```

```
Out[4]:
```

```
{931: 2,  
 943: 3,  
 984: 1,  
 998: 1,  
1043: 1,  
1064: 2,  
1095: 1,  
1133: 1,  
1138: 1,  
1156: 1,  
1167: 1,  
1216: 1,  
1222: 1,  
1240: 1,  
1246: 1,  
1248: 1,  
1250: 1,  
1269: 2,  
1276: 1,  
1287: 1,  
1297: 1,  
1300: 1,  
1301: 1,  
1305: 2,  
1306: 2,  
1326: 2}
```

可以看到在第 943 篇推特中确实包含了 3 个 beef:

```
In [5]: doc_text[943]
```

```
Out[5]: 'Shouldn\'t beef.. be 100% beef? Gross. RT  
@dennya: Hmm... Taco Bell\'s "beef" is alleged to be  
only 36% meat. http://bit.ly/hsQXer'
```

在最终的查询结果中，两种方式最后的结果类似：

例如，查询的语句为：

“beef law firm”

结果为：

```
In [6]: VSM('beef law firm')
```

```
(998, 10.541680575652906)
(1240, 9.141135808875863)
(1305, 8.205819639049135)
(1043, 7.7655219815272805)
(1793, 7.48396385838425)
(1330, 7.22210852822369)
(1138, 6.97795777191502)
(984, 6.521121995326817)
(4788, 6.369200461378444)
(1572, 5.709939198273135)
(1248, 5.47764398617856)
(1374, 5.47764398617856)
(1536, 4.9836141746407225)
(1438, 4.809243218315029)
(1376, 4.716883170925269)
(4888, 4.498311922270867)
(943, 4.452585792368652)
(1379, 4.299099940940998)
(1306, 4.116784271080647)
(1326, 4.116784271080647)
(2885, 4.116784271080647)
(4635, 4.094711833690861)
(19126, 4.094711833690861)
(29965, 3.9494027164582866)
(1419, 3.9493027546947084)
(2155, 3.9493027546947084)
.....
```

```
In [7]: BM25('beef firm law')
```

```
(998, 19.143751421409178)
(1305, 17.520422166164483)
(1240, 17.248385640147497)
(1043, 15.236963799181083)
(1793, 14.805332545873728)
(1330, 14.397482051237798)
(1138, 14.01149969892709)
(984, 12.161107078975114)
(4788, 11.877793323994535)
(1572, 10.891757362304158)
(943, 10.747176260549606)
(1248, 10.554560456265609)
(1374, 10.554560456265609)
(1536, 9.858955363939671)
(1376, 9.694919055326764)
(1438, 9.58736539395301)
(4888, 9.433044506091724)
(1379, 9.184945130323296)
(1306, 8.949561875915698)
(1326, 8.949561875915698)
(2885, 8.949561875915698)
(29965, 8.859786246698233)
(4635, 8.748276360639752)
(19126, 8.748276360639752)
(1419, 8.72594153978286)
(2155, 8.72594153978286)
.....
```

可以看到在结果中，除了小幅度的差异，基本相同。而且查询语句中的顺序并没有影响结果。

在最高得分的文章 998 中：

```
In [8]: doc_text[998]
Out[8]: "Alabama law firm to Taco Bell: That's not beef
http://on.msnbc.com/g6kcaE"
```

可见文章与查询的相关度很高，结果较为满意。

在得分较低的文章 1306 中：

```
In [9]: doc_text[1306]
Out[9]: 'Guess @TacoBell may have to stop calling their
Beef Burritos ... Beef Burritos. *gross* http://
tinyurl.com/4rg3mv9'
```

可见相关度比文章 998 要低，证明分数的高低与相关性正相关。

结论分析与体会：

在处理数据时，先生成数据的 posting 和 inverted index 可以加快查询时的速度。如果数据的处理放在查询时再进行，则效率很低，并且每进行一次查询都要处理一次，很浪费时间。

Inverted index 和 posting 可以分开保存，方便维护。只需要用相同的一个 keyword 就可以把它们互相联系在一起。