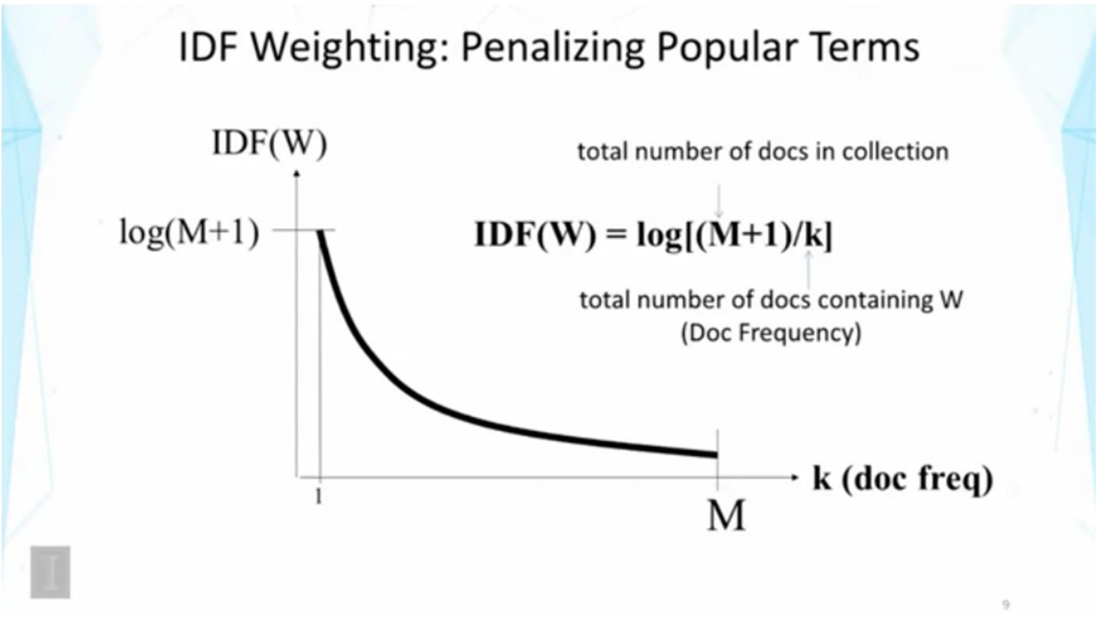
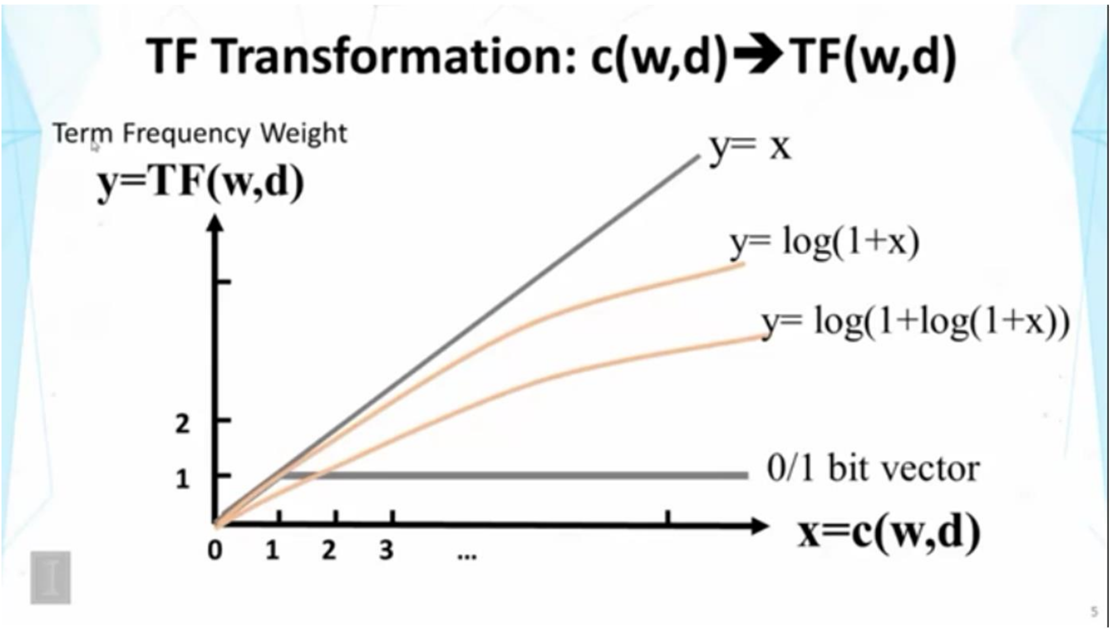
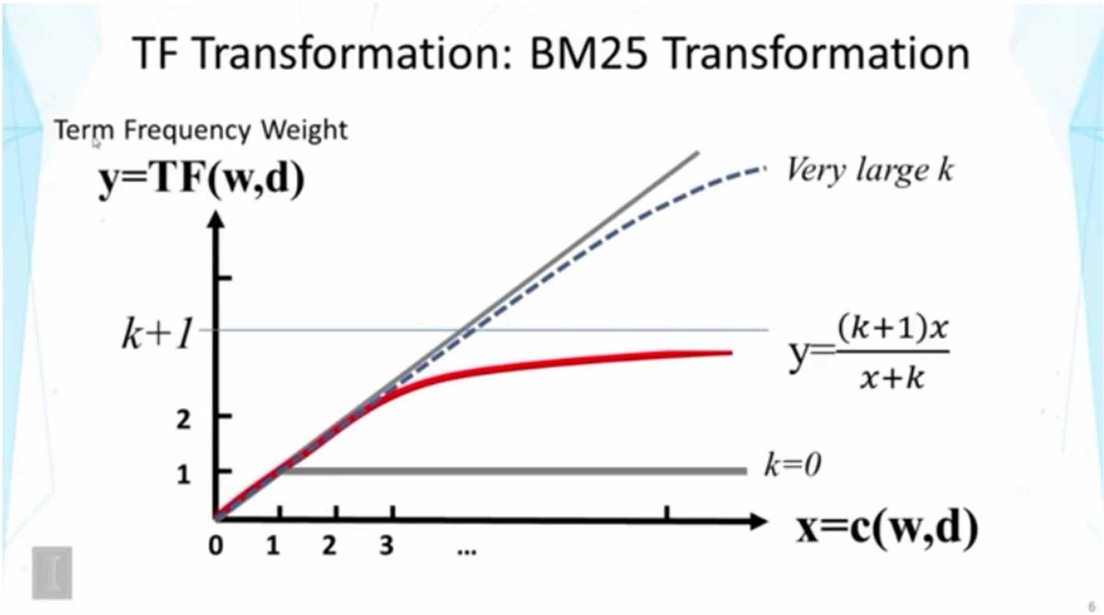
山东大学_____计算机科学与技术_____学院

信息检索 课程实验报告

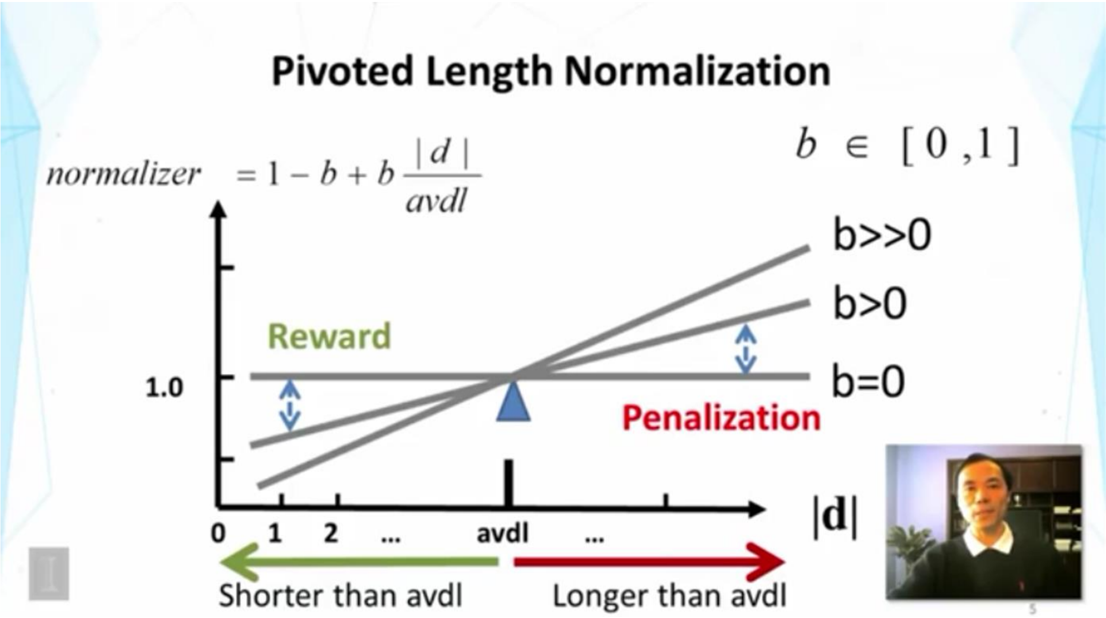
学号：201600150109	姓名：沈棋韬	班级： 16人工智能班
实验题目：Pivoted Length Normalization VSM and BM25		
实验内容： 实现Pivoted Length Normalization VSM; 实现BM25； 改进Postings：(docID, Freq)，不仅记录单词所在的文档ID，也记录其在文档中的Frequency； 构建inverted index时,记录文档的长度,以及计算average document length (avdl)		
实验过程中遇到和解决的问题： 计算 IDF 公式：  在VSM中使用取log的方式来对词频加权：		



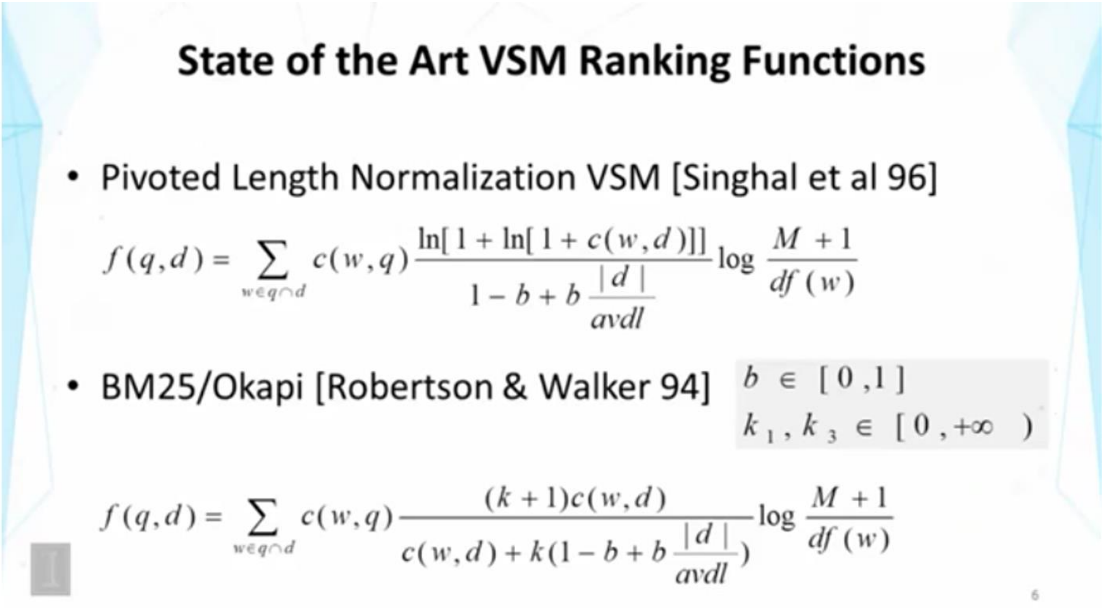
在 BM25 中，使用一个系数 k 来对词频加权：



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:



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,
```

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/hsQXeY'
```

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。

使用助教发的代码进行效果评估：

```
In [1]: runfile('D:/pythonCode/eval_hw4/eval_hw4/eval_hw4.py', wdir='D:/
pythonCode/eval_hw4/eval_hw4')
query: 171 ,AP: 0.9955702442922506
query: 172 ,AP: 0.31954090042576716
query: 173 ,AP: 0.40063144988809235
query: 174 ,AP: 0.8707503221330161
query: 175 ,AP: 0.38910505836575876
query: 176 ,AP: 0.9255127524326174
query: 177 ,AP: 0.6477787355907725
query: 178 ,AP: 0.4454171486187664
query: 179 ,AP: 0.5541488750801512
query: 180 ,AP: 0.17271157167530224
query: 181 ,AP: 0.9346178286129265
query: 182 ,AP: 0.19305019305019305
query: 183 ,AP: 0.4018277891836399
query: 184 ,AP: 0.4993474097928063
query: 185 ,AP: 0.8336406784936197
query: 186 ,AP: 0.8426303124846924
query: 187 ,AP: 0.9983900226757368
query: 188 ,AP: 0.42620672692882616
query: 189 ,AP: 0.16448358954467976
query: 190 ,AP: 0.6158455150634802
query: 191 ,AP: 0.7073748031316675
query: 192 ,AP: 0.7052430711756998
query: 193 ,AP: 0.3456102310871636
query: 194 ,AP: 0.9849826388888889
query: 195 ,AP: 0.25683710592566433
query: 196 ,AP: 0.676712416806247
query: 197 ,AP: 0.8979591836734694
query: 198 ,AP: 0.43319813385278405
query: 199 ,AP: 0.2375296912114014
query: 200 ,AP: 0.39501666273660396
query: 201 ,AP: 0.37037037037037035
query: 202 ,AP: 0.6802721088435374
query: 203 ,AP: 0.04569681297073339
query: 204 ,AP: 0.9101219074886248
query: 205 ,AP: 0.6060606060606061
query: 206 ,AP: 0.790423384968748
query: 207 ,AP: 0.768356710752457
query: 208 ,AP: 0.303951367781155
query: 209 ,AP: 0.16447368421052633
query: 210 ,AP: 0.8845162733610249
query: 211 ,AP: 0.9028069051774217
query: 212 ,AP: 0.5505621131641893
query: 213 ,AP: 0.3834276972988598
query: 214 ,AP: 0.704225352112676
query: 215 ,AP: 0.30120481927710846
query: 216 ,AP: 0.579203451532937
query: 217 ,AP: 0.4625182142687342
query: 218 ,AP: 0.23016357815976218
query: 219 ,AP: 0.3704424995267279
query: 220 ,AP: 0.517348082783178
query: 221 ,AP: 0.1988071570576541
query: 222 ,AP: 0.3993814465334278
query: 223 ,AP: 0.7572867339551992
query: 224 ,AP: 0.70828173374613
query: 225 ,AP: 0.9981904607573056
MAP = 0.5610866279087597
```

```
query 171 , NDCG:  0.9760164953868642
query 172 , NDCG:  0.9066834025199301
query 173 , NDCG:  0.5605000546944076
query 174 , NDCG:  0.9001135647674106
query 175 , NDCG:  0.752355975448928
query 176 , NDCG:  0.8382154465331632
query 177 , NDCG:  0.7905017925816821
query 178 , NDCG:  0.8090177303858183
query 179 , NDCG:  0.6549652409163867
query 180 , NDCG:  0.5808344258416988
query 181 , NDCG:  0.8100954954156911
query 182 , NDCG:  0.5610352389255088
query 183 , NDCG:  0.9288037313400754
query 184 , NDCG:  0.6580931551119361
query 185 , NDCG:  0.8536744468767384
query 186 , NDCG:  0.8206009021715979
query 187 , NDCG:  0.8192671092090744
query 188 , NDCG:  0.5912797485031454
query 189 , NDCG:  0.33391953733565555
query 190 , NDCG:  0.7015023372754727
query 191 , NDCG:  0.7774440146191183
query 192 , NDCG:  0.798895146791585
query 193 , NDCG:  0.40349865828328674
query 194 , NDCG:  0.9580808899433048
query 195 , NDCG:  0.5801288345487705
query 196 , NDCG:  0.7578589087500729
query 197 , NDCG:  0.8998946547351205
query 198 , NDCG:  0.5728089917730196
query 199 , NDCG:  0.8316111002441205
query 200 , NDCG:  0.7649253779621528
query 201 , NDCG:  0.769363027730994
query 202 , NDCG:  0.8309851102351588
query 203 , NDCG:  0.1293880434085598
query 204 , NDCG:  0.8818950929235301
query 205 , NDCG:  0.9364666141437938
query 206 , NDCG:  0.7260584750375837
query 207 , NDCG:  0.8119416787924614
query 208 , NDCG:  0.6674775525542138
query 209 , NDCG:  0.7019133242235883
query 210 , NDCG:  0.9161107711073098
query 211 , NDCG:  0.895956857648288
query 212 , NDCG:  0.837195084455298
query 213 , NDCG:  0.9615031378891293
query 214 , NDCG:  0.8379562779932439
query 215 , NDCG:  0.5223553767740823
query 216 , NDCG:  0.8060437863307118
query 217 , NDCG:  0.6143145825736944
query 218 , NDCG:  0.45063470934456984
query 219 , NDCG:  0.46582944335814414
query 220 , NDCG:  0.6185907673841651
query 221 , NDCG:  0.6993262191586171
query 222 , NDCG:  0.5249464677991027
query 223 , NDCG:  0.7196085236176019
query 224 , NDCG:  0.754173586934544
query 225 , NDCG:  0.8573237537585741
NDCG = 0.7296360122557943
```

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。