信息检索与Web搜索

第7讲 文档评分、词项权重计算及向量空间模型

Scoring, Term Weighting & Vector Space Model

授课人：高曙明

*改编自“现代信息检索”网上公开课件（http://ir.ict.ac.cn/~wangbin）

布尔搜索的不足

- 布尔查询常常会是过少(=0)或者过多(>1000)的结果
- 查询 1 (布尔与操作): [standard user dlink 650]
  - → 200,000 个结果 – 太多
- 查询2 (布尔与操作): [standard user dlink 650 no card found]
  - → 0 个结果 – 太少
- 在布尔检索中，需要大量技巧来生成一个可以获得合适规模结果的查询

布尔搜索问题的原因

- 将文档与查询的相关度仅仅分为相关与不相关两种情况

  过于粗略

  - 将在文档中出现一次和出现多次的词项都作为文档的一个同等重要的表征
  - 将查询中的所有词项等同看待

排序式检索(Ranked retrieval)

- 排序式检索：能够对返回的文档根据其与查询之间的相关度进行排序的检索
- 排序式检索可以避免产生过多或者过少的结果
  - 大规模的返回结果可以通过排序技术来避免
  - 可以只显示前10条结果，不会让用户感觉到信息太多
- 关键：排序算法真的有效，即相关度大的文档结果会排在相关度小的文档结果之前

排序式检索中的评分技术

- 我们希望，在同一查询下，文档集中相关度高的文档排名高于相关度低的文档
- 如何实现?
  - 通常做法是对每个查询-文档对赋一个[0, 1]之间的分值
  - 该分值度量了文档和查询的匹配程度

二值关联矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	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
. . .						

每篇文档可以看成是一个二值的词项向量 $\in \{0, 1\}^{|V|}$

词项频率 tf

- 词项频率 $\mathrm{tf}_{t,d}$：是指词项 t 在文档 d 中出现的次数
- 刻画了词项与文档的相关度，$\mathrm{tf}_{t,d}$ 越大，其相关度越高
- 直接采用原始 tf 值刻画相关度不太合适
  - 某个词项在A文档中出现十次，即tf = 10，在B文档中 tf = 1，那么A比B更相关
  - 但是相关度不会相差10倍
- 对数词频：

$$\mathrm{wf}_{t,d} = \begin{cases} 1 + \log \mathrm{tf}_{t,d} & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- $\mathrm{tf}_{t,d} \rightarrow \mathrm{wf}_{t,d}$: 0 → 0, 1 → 1, 2 → 1.3, 10 → 2, 1000 → 4

词袋(Bag of words)模型

- 词袋模型: 将文档表示成一个词频向量
- 能够对返回的文档进行排序
- *John is quicker than Mary* 及 *Mary is quicker than John* 的表示结果一样

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth . . .
ANTHONY	157	73	0	0	0	1
BRUTUS	4	157	0	2	0	0
CAESAR	232	227	0	2	1	0
CALPURNIA	0	10	0	0	0	0
CLEOPATRA	57	0	0	0	0	0
MERCY	2	0	3	8	5	8
WORSER	2	0	1	1	1	5
. . .						

文档集频率 vs.文档频率

- 词项t的文档集频率cf(Collection frequency)：整个文档集中出现的t词条的个数
- 词项t的文档频率df: 整个文档集中包含t的文档的篇数
- 在一定程度上刻画了词项对于文档的区分能力

单词	文档集频率	文档频率
INSURANCE	10440	3997
TRY	10422	8760

- 上例表明 df 比cf 更适合权重计算

逆文档频率

- 逆文档频率的定义

$$\mathrm{idf}_t = \log_{10} \frac{N}{\mathrm{df}_t}$$ (其中N 是文档集中文档的数目)

- idf_t 是反映词项t的文档区分能力的一个指标
- 作用：可以用作查询词项的权重，提升罕见词的影响，如在查询“arachnocentric line”中前者具有更高的权重
- 采用[log N/df_t]而不是 [N/df_t]，这可以对idf的影响有所抑制

idf的计算样例

- 利用右式计算idf_t: $\text{idf}_t = \log_{10} \frac{1{,}000{,}000}{\text{df}_t}$

词项	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

tf-idf权重

- 词项的tf-idf权重：是tf权重和idf权重的乘积

$$\mathrm{wf}_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \cdot \log \frac{N}{\mathrm{df}_t}$$

- 信息检索中最出名的权重计算方法
- tf-idf权重的性质
  - 是与文档和查询皆相关的权重
  - 随着词项频率的增大而增大
  - 随着词项罕见度的增加而增大

二值-->权重关联矩阵

- 词项-文档的权重关联矩阵

$$\begin{array}{c|cccc} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{array}$$

课堂练习：词项、文档集及文档频率

统计量	符号	定义
词项频率	$tf_{t,d}$	*t*在文档*d*中出现的次数
文档频率	Df_t	出现 t的文档数目
文档集频率	cf_t	t在文档集中出现的总次数

- df 和 cf 有什么关系?
- tf 和 cf 有什么关系?
- tf 和 df 有什么关系?

向量空间模型

- 将文档和查询都表示为一个 t 维向量：

$$V_j = (w_{1j}, w_{2j}, \cdots , w_{tj})$$

这里 t= | vocabulary | , j 代表一个文档或查询

$w_{ij} = tf_{ij} * idf_i$ 代表词项i在文档或查询中的权重

- 基于查询向量和文档向量之间的相似度确定它们之间的相关度（非二值）
- 输出：基于相关度排序的文档集

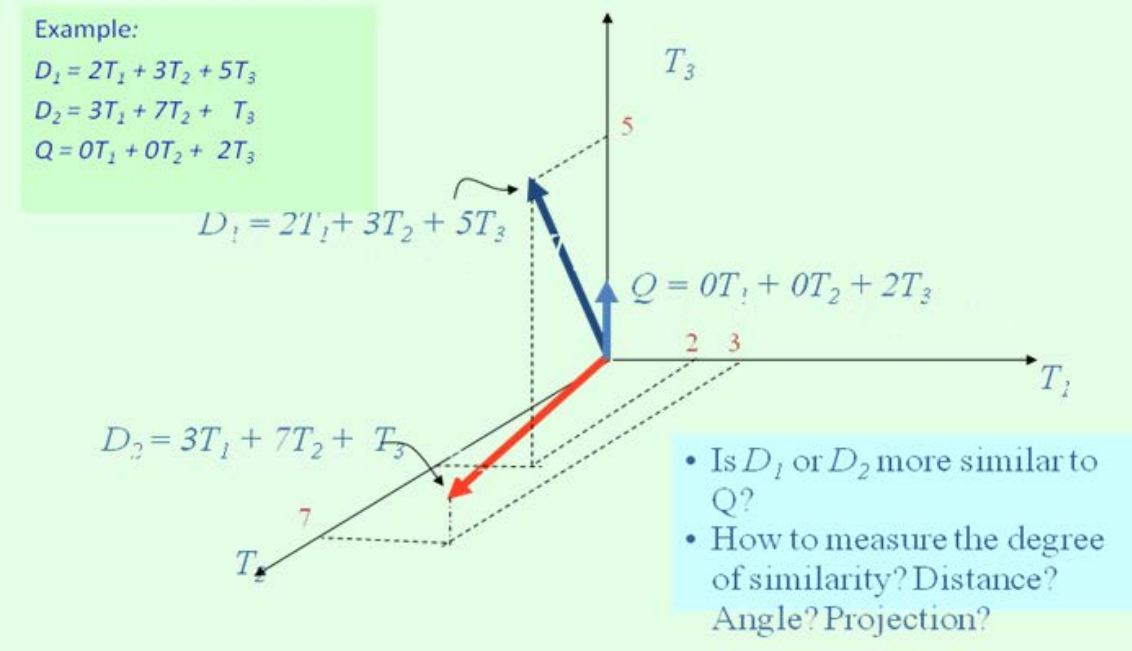
向量空间举例

	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
. . .						

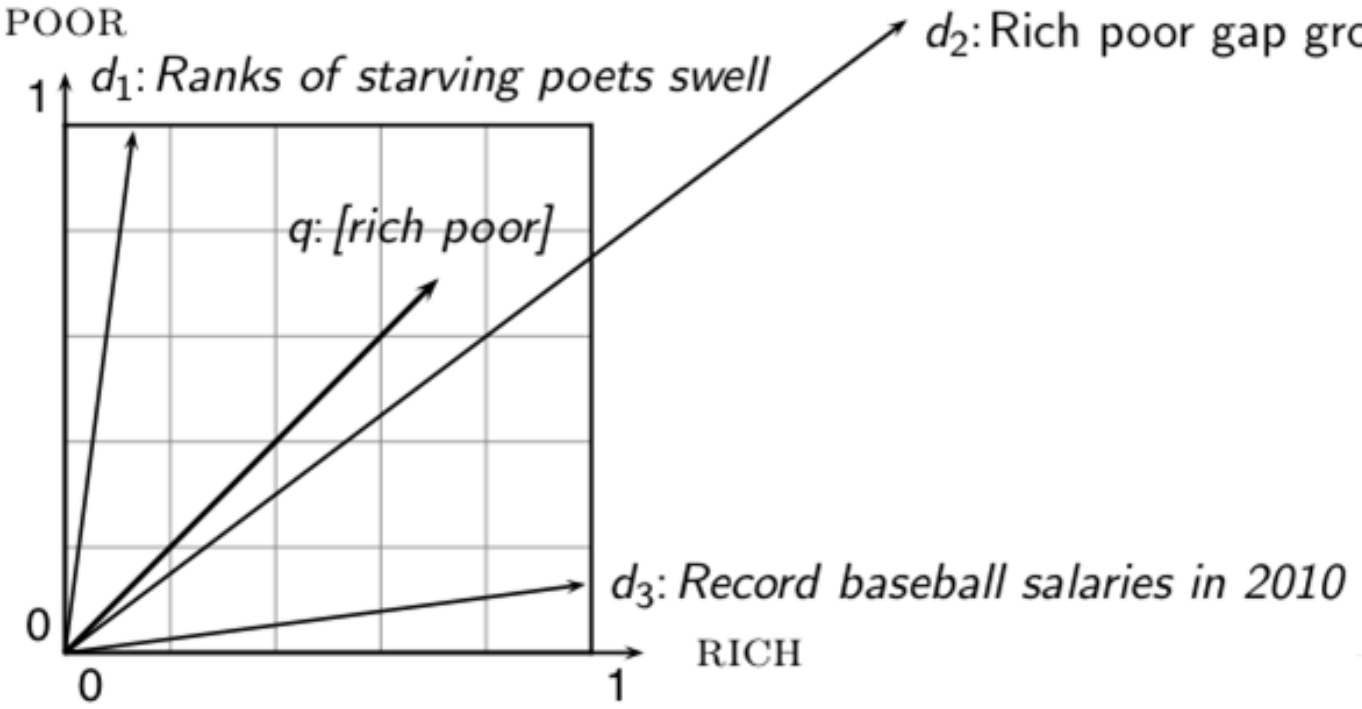
每篇文档表示成一个基于 tf-idf 权重的实值向量

相似度度量

- **相似度度量：**是指计算两个向量之间相似程度的函数
- **作用**
  - 支持对相关文档进行排序
  - 通过给定阈值控制返回相关文档的数量



欧氏距离不好的例子



尽管查询q和文档d2的词项分布$\vec{d}_2$非常相似，但是采用欧氏距离计算它们对应向量之间的距离非常大。

余弦相似度计算方法

$$\cos(\vec{q},\vec{d}) = \text{SIM}(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

- qi 是第i 个词项在查询q中的tf-idf权重
- di是第i 个词项在文档d中的tf-idf权重
- $|\vec{q}|$ 和 $|\vec{d}|$ 分别是 $\vec{q}$ 和 $\vec{d}$ 的长度
- 上述公式就是 $\vec{q}$ 和 $\vec{d}$ 的余弦相似度，或者说向量 $\vec{q}$ 和 $\vec{d}$ 的夹角的余弦

一个实例

3本小说之间的相似度

(1) SaS(理智与情感):

Sense and Sensibility

(2) PaP(傲慢与偏见):

Pride and Prejudice

(3) WH(呼啸山庄):

Wuthering Heights

词项频率tf

词项	SaS	PaP	WH
AFFECTION	115	58	20
JEALOUS	10	7	11
GOSSIP	2	0	6
WUTHERING	0	0	38

一个实例（续）

对数词频$(1+\log_{10}tf)$

词项	SaS	PaP	WH
AFFECTION	3.06	2.76	2.30
JEALOUS	2.0	1.85	2.04
GOSSIP	1.30	0	1.78
WUTHERING	0	0	2.58

对数词频的余弦归一化结果

词项	SaS	PaP	WH
AFFECTION	0.789	0.832	0.524
JEALOUS	0.515	0.555	0.465
GOSSIP	0.335	0.0	0.405
WUTHERING	0.0	0.0	0.588

$\cos(SaS,PaP) \approx 0.789 * 0.832 + 0.515 * 0.555 + 0.335 * 0.0 + 0.0 * 0.0 \approx 0.94.$

$\cos(SaS,WH) \approx 0.79$

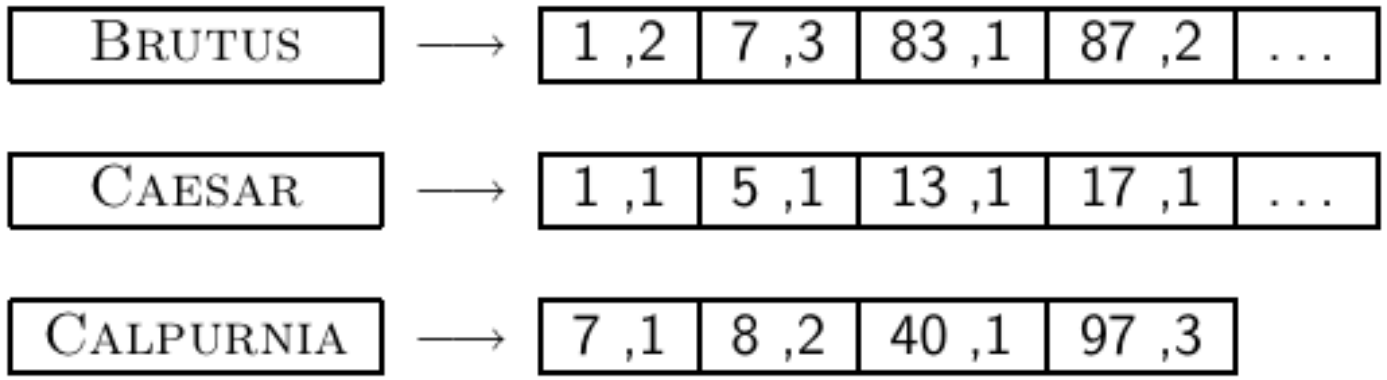
$\cos(PaP,WH) \approx 0.69$

$\cos(SaS,PaP) > \cos(SaS,WH) > \cos(PaP,WH)$

余弦相似度计算算法

```
CosineScore(q)
 1   float Scores[N] = 0
 2   Initialize Length[N]
 3   for each query term t
 4   do calculate w_{t,q} and fetch postings list for t
 5       for each pair(d, tf_{t,d}) in postings list
 6       do Scores[d] += wf_{t,d} × w_{t,q}
 7   Read the array Length[d]
 8   for each d
 9   do Scores[d] = Scores[d] / Length[d]
10   return Top K components of Scores[]
```

词项频率tf也存入倒排索引中



当然也需要位置信息，上面没显示出来

面向VSM的倒排索引

- 最好存储$W_{t,d}$，但其为实数，空间消耗大
- 变通的方法
  - 在每条倒排记录中，除了docID 还要存储 $tf_{t,d}$
  - 对于词典中的每个词，保存其idf值
  - 对 tf 采用一元码编码，效率很高（易于分割）
- 总体而言，额外存储 tf 所需要的开销不是很大：采用位编码压缩方式，每条倒排记录增加不到一个字节的存储量

基于VSM的查询计算

查询: “best car insurance”；一篇文档: “car insurance auto insurance”； N = 1, 000,000

term	query				document			product
	tf	df	idf	$w_{t,q}$	tf	wf	$w_{t,d}$	
auto	0	5000	2.3	0	1	1	0.41	0
best	1	50000	1.3	1.3	0	0	0	0
car	1	10000	2.0	2.0	1	1	0.41	0.82
insurance	1	1000	3.0	3.0	2	2	0.82	2.46

L2：欧几里得正则化过了

最终结果 $\sum w_{qi} \cdot w_{di} = 0 + 0 + 0.82 + 2.46 = 3.28$

向量空间模型特点分析

- 是一个简单的，基于数学的方法
- 对文档的表示相对于布尔检索模型更加合理
- 同时考虑查询词项的局部（tf）和全局（idf）出现频率
- 提供排序检索结果和部分匹配检索结果
- 对大规模文档集也能够有效实施

向量空间模型的问题

- 忽略了语义信息（不作词义理解）
- 忽略了句法信息（不考虑短语、词序、词间距等）
- 假设词项之间都无关，不尽合理
- 缺少布尔模型的硬控制力（如一定需要某个词项出现在返回结果中）

参考资料

- 《信息检索导论》第6、7章
- http://ifnlp.org/ir
  - 向量空间入门
  - Exploring the similarity space (Moffat and Zobel, 2005)
  - Okapi BM25 (另外一种著名的权重计算方法，《信息检索导论》11.4.3节)

课后作业

- 见课程网页：

  http://10.76.3.31

130的VB码生成：

130的二进制码为10000010

先取后七位（mod 128就相当于取一个数对应二进制的后七位）有0000010，并将原数右移7位（div 128），剩下的是1

则现在的结果为00000010（开头补零到8位）

再取后七位为0000001，此时剩下的为0

将之加入到结果前面（伪代码第3行的prepend是将数字添加到数组最前面的函数）

则此时结果为00000001,00000010,

将最后一个字节的最高位置为1（+128），

最终结果为00000001,10000010。

130的VB码解码：

从第一字节开始，

若当前字节大于等于128，说明最高位为1，则为结尾字节

对于00000001,10000010，

第一字节的十进制是1，第二个字节为130

开始前置结果为0，

先从第一个开始，其小于128

则将之与结果乘以128（左移7位）相加，现在的结果为0*128+1 = 1，

第二个大于128，为结尾字节，

将其减去128（去掉最高位的1），并将原来的结果乘以128（左移7位）并与其相加，

为1*128+ (130 – 128) = 130

130的Y编码生成：

而对于γ编码，其偏移应为0000010（去掉最前边的1），
偏移长度为7，对应一元编码为11111110，
所以结果为11111110,0000010

解码先通过之前一元码得到其偏移的长度为7，读出其偏移为0000010，
　　最高位加上一个1，
得到结果10000010，转换为十进制为130