

信息检索与数据挖掘

第5章 向量模型及检索系统

——第一讲 向量模型

课程内容

- 第1章 绪论
- 第2章 布尔检索及倒排索引
- 第3章 词典查找及扩展的倒排索引
- 第4章 索引构建和索引压缩
- 第5章 向量模型及检索系统
 - 向量模型
 - 检索系统
- 第6章 检索的评价
- 第7章 相关反馈和查询扩展
- 第8章 概率模型
- 第9章 基于语言建模的检索模型
- 第10章 文本分类
- 第11章 文本聚类
- 第12章 Web搜索
- 第13章 多媒体信息检索
- 第14章 其他应用简介

本讲提纲

- ① 回顾
- ② 排序式检索
- ③ 词项频率
- ④ tf-idf权重计算
- ⑤ 向量空间模型

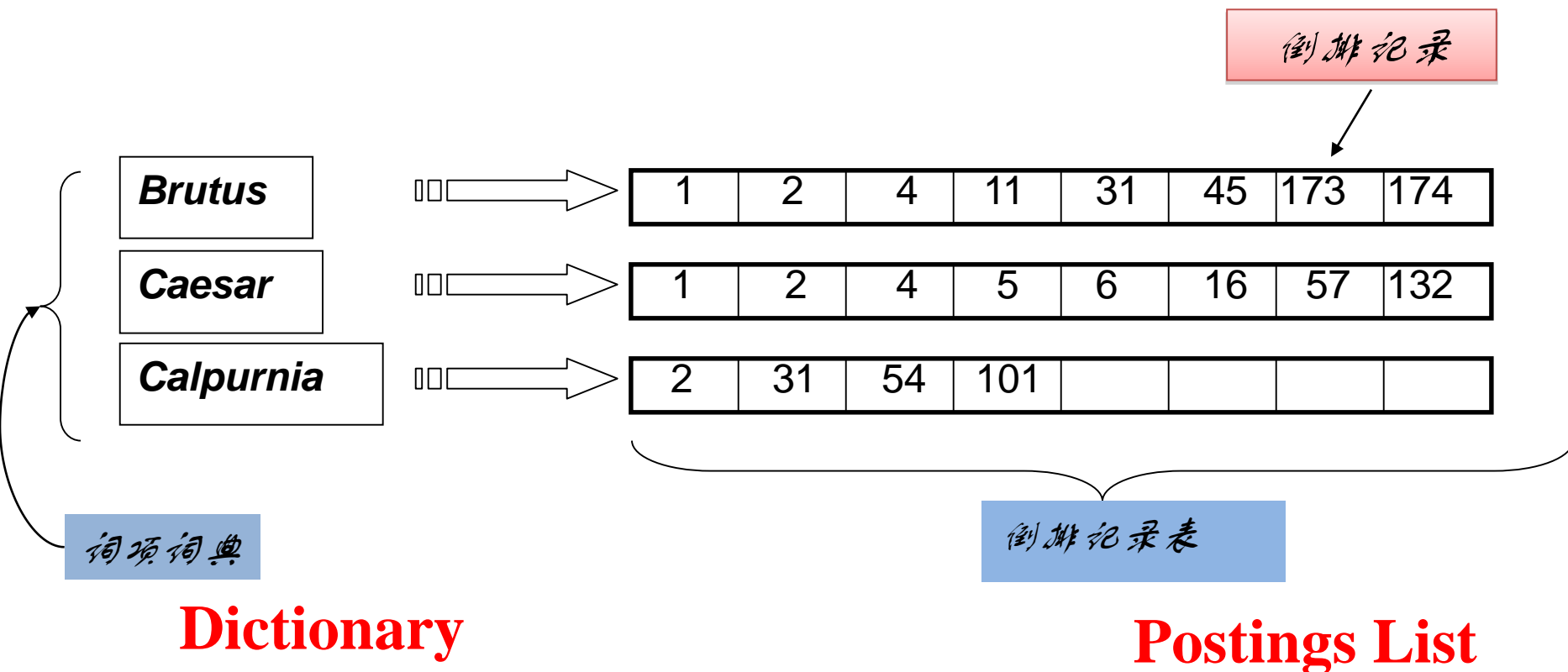
本讲提纲

- ① 回顾
- ② 排序式检索
- ③ 词项频率
- ④ tf-idf权重计算
- ⑤ 向量空间模型

回顾：布尔检索

- 文档表示
 - 一个文档被表示为**关键词的集合**
- 查询表示
 - 查询式(Queries)被表示为**关键词的布尔组合**，用“与、或、非”连接起来（主析取范式DNF）
- 相关度计算
 - 一个文档当且仅当它能够满足布尔查询式时，才将其检索出来
 - 检索策略是**二值匹配**

回顾：倒排索引



回顾：词典的建立及扩展的倒排索引

●如何建立词项词典？

- 文档解析：格式？语言？编码方式？
- 词条化：词条 (Tokens) / 词项 (Terms)
- 停用词：停用词表？查表法 or 基于文档频率
- 词项归一化：等价类 \leftrightarrow 同义词扩展表
- 词形归并：am, are, is \rightarrow be
- 词干还原：去除单词两端词缀、Porter算法

●如何实现倒排记录表？

- 跳表：跳表指针(位置、个数、更新问题)
- 短语查询
 - 二元词索引 \rightarrow 扩展的二元词索引：词性标注
 - 位置信息索引 \rightarrow 邻近查询
 - 混合索引机制

回顾：索引构建

- 基于排序的索引构建算法
 - 它是一种最原始的在内存中进行倒排的方法
 - 基于块的排序索引算法
 - 合并排序操作对于基于磁盘的排序来说很高效(避免寻道)
- 内存式单遍扫描索引构建算法
 - 没有全局的词典
 - 对每个块都生成单独的词典
 - 不对倒排记录进行排序
 - 有新的倒排记录出现时，直接在倒排记录表中增加一项
- 采用MapReduce的分布式索引构建算法
- 动态索引构建算法：多个索引，对数合并

回顾：索引压缩

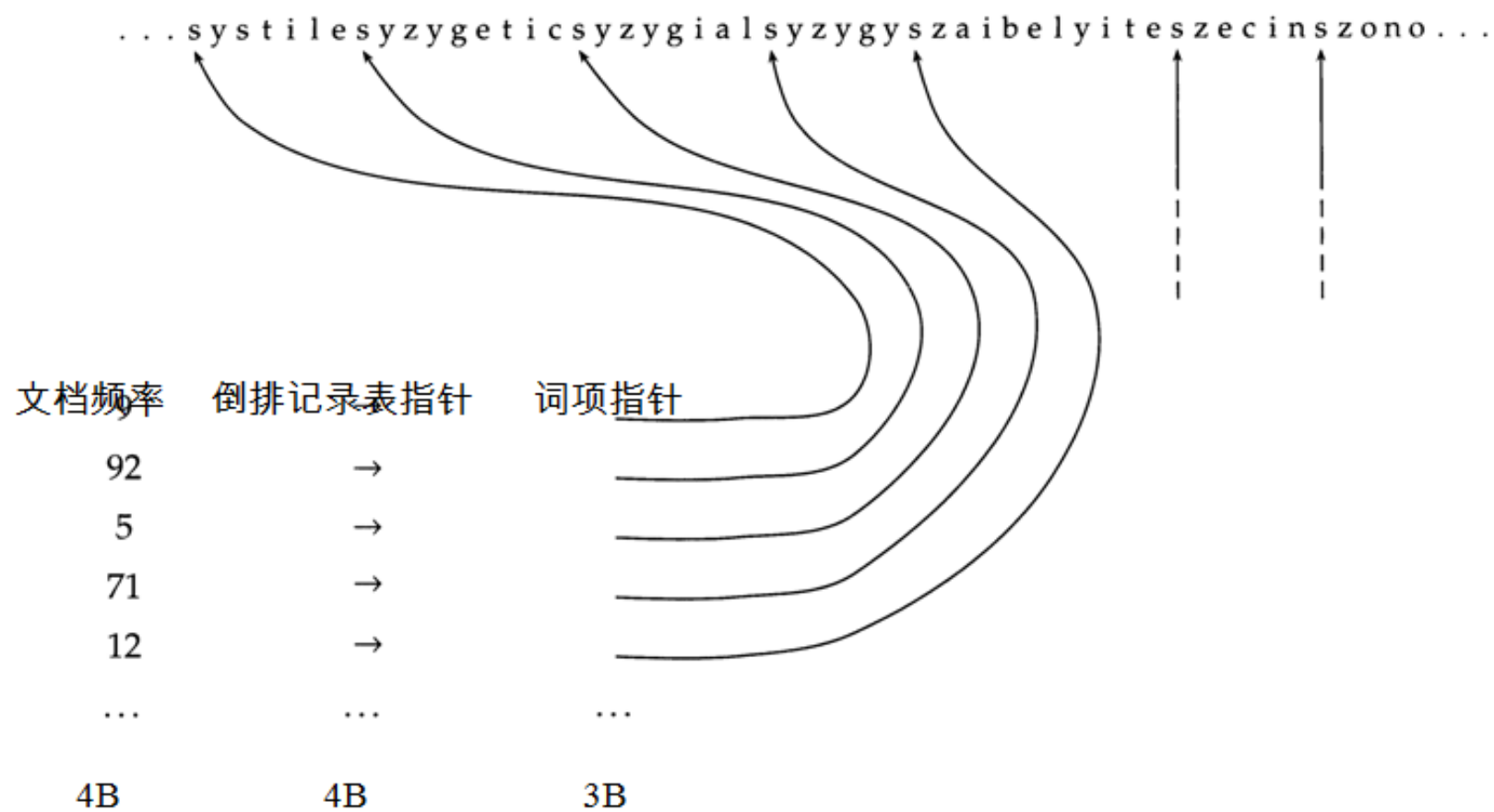
- 词项的统计特性：Heaps定律和Zipf定律
- 词典压缩
 - 将词典看成单一字符串、按块存储、前端编码
- 倒排记录表压缩
 - 可变字节编码和 γ 编码

文档集 (文本、XML标签等)	3,600.0
文档集 (文本)	960.0
词项关联矩阵	40,000.0
倒排记录表，未压缩 (32位字)	400.0
倒排记录表，未压缩 (20位)	250.0
倒排记录表，可变字节码	116.0
倒排记录表， γ 编码	101.0

MB

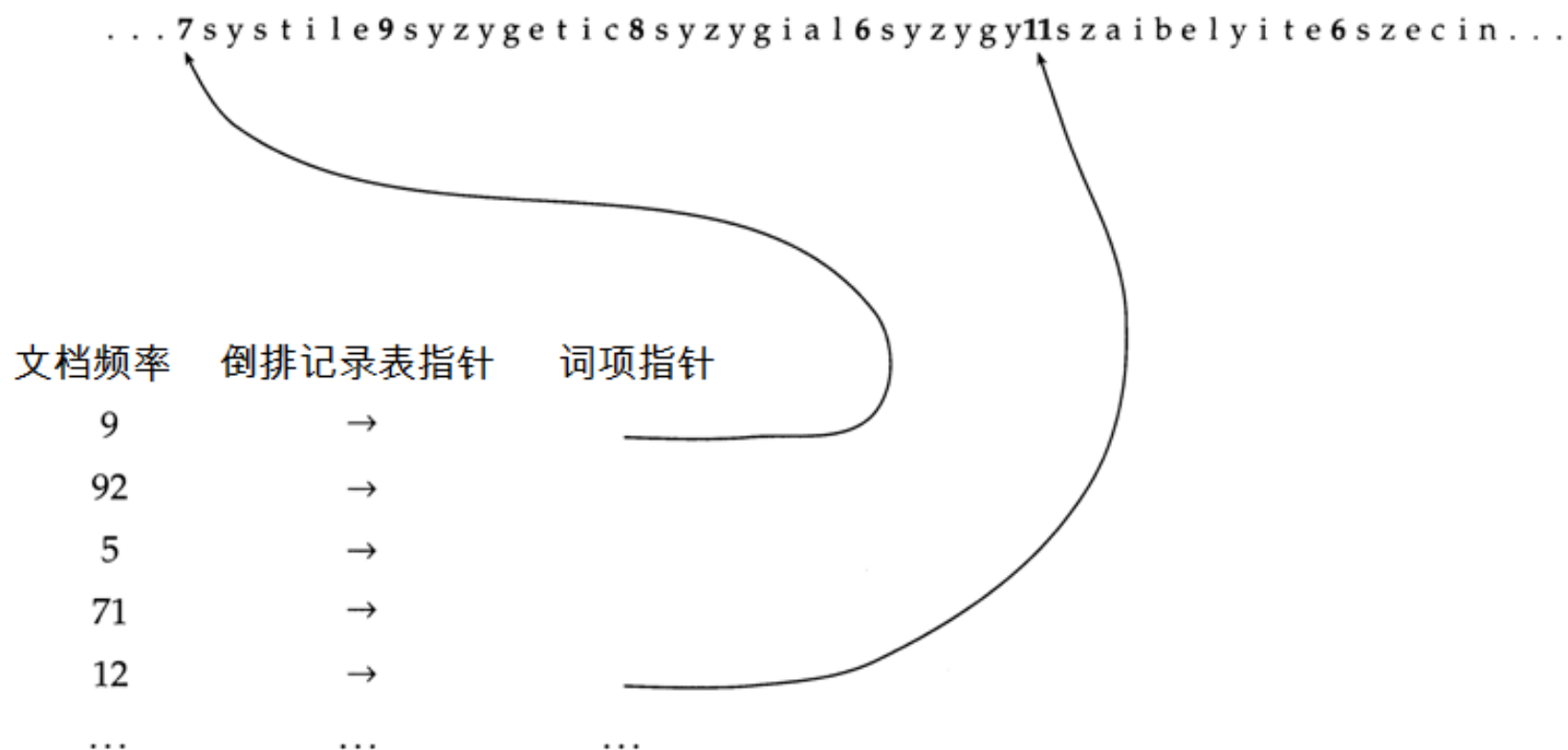
回顾：索引压缩

将整部词典看成单一字符串



回顾：索引压缩

单一字符串方式下按块存储



回顾：索引压缩

对间隔编码

	编码对象	倒排记录表				
the	文档ID	...	283 042	283 043	283 044	283 045 ...
	文档ID间距			1	1	2 ...
computer	文档ID	...	283 047	283 154	283 159	283 202 ...
	文档ID间距			107	5	43 ...
arachnocentric	文档ID	252 000	500 100			
	文档ID间距	252 000	248 100			

回顾：索引压缩

可变字节(VB)码

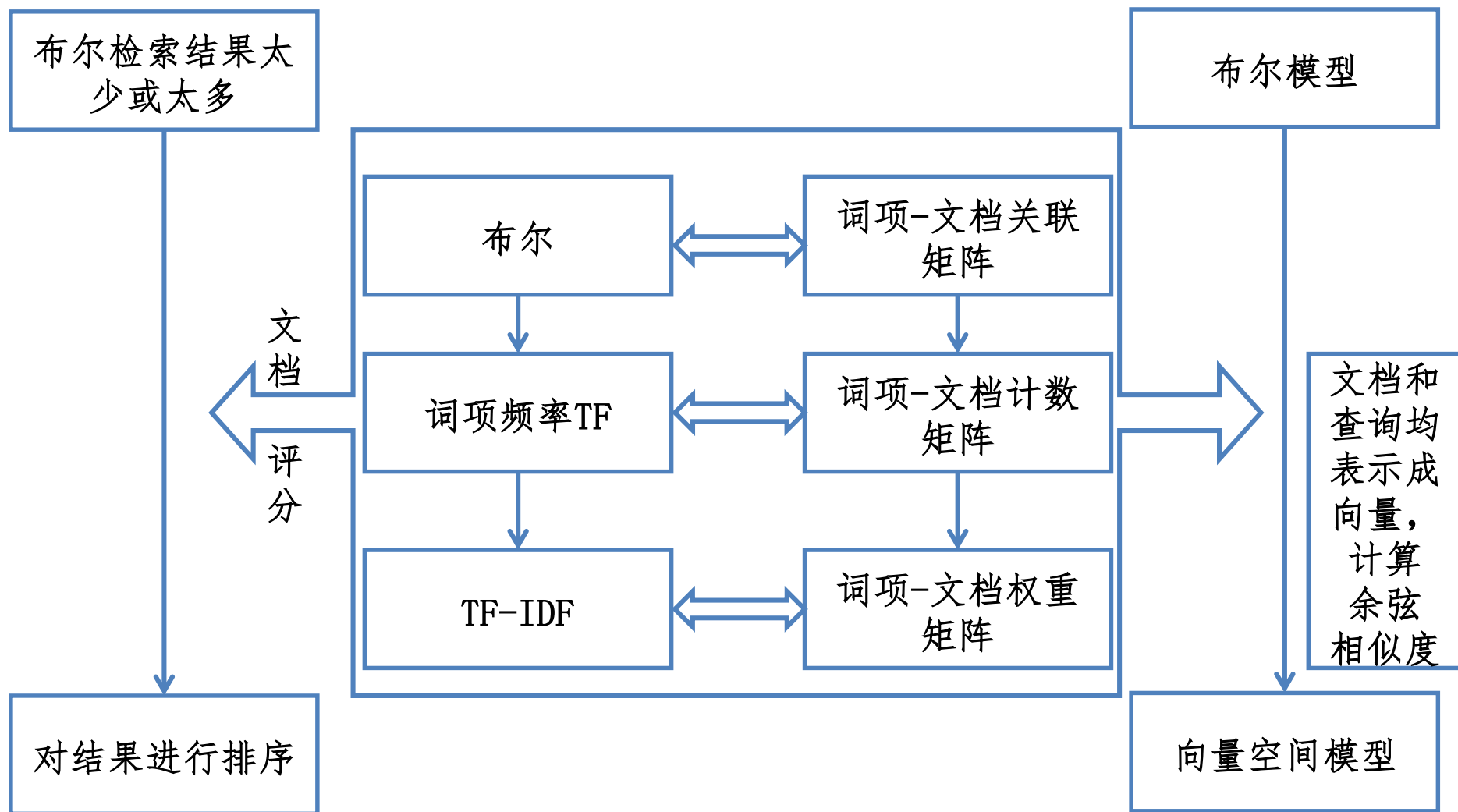
- 被很多商用/研究系统所采用
- 设定一个专用位（高位） c 作为延续位(continuation bit)
- 如果间隔表示少于7比特，那么 c 置 1，将间隔编入一个字节的后7位中
- 否则：将低7位放入当前字节中，并将 c 置 0，剩下的位数采用同样的方法进行处理，最后一个字节的 c 置1（表示结束）

回顾：索引压缩

γ 编码

- 将G 表示成长度(length)和偏移(offset)两部分
- 偏移对应G的二进制编码，只不过将首部的1去掉
- 例如 $13 \rightarrow 1101 \rightarrow 101 = \text{偏移}$
- 长度部分给出的是偏移的位数
- 比如G=13（偏移为 101），长度部分为 3
- 长度部分采用一元编码：1110.
- 于是G的 γ 编码就是将长度部分和偏移部分两者联接起来得到的结果。

本讲结构图



本讲提纲

- 1 回顾
- 2 排序式检索
- 3 词项频率
- 4 tf-idf权重计算
- 5 向量空间模型

排序式检索

- 迄今为止，我们只介绍了布尔查询
 - 文档要么匹配要么不匹配
- 对自身需求和文档集性质非常了解的专家而言，布尔查询是不错的选择
- 然而对大多数用户来说不方便
 - 大部分用户不能撰写布尔查询或者他们认为需要大量训练才能撰写合适的布尔查询
 - 大部分用户不愿意逐条浏览1000多条结果，特别是对Web搜索更是如此

布尔查询：“盛宴” or “饥荒”

- 布尔查询的结果经常不是太多就是太少
- Query1 “standard user dlink 650” -> 200,000 个匹配结果
- Query2 “standard user dlink 650 no card found” -> 0 个匹配结果
- 需要花费很多精力去构造一个合适的query才可以获得一个在数量上可以接受的查询结果

排序检索模型

- 在排序检索模型中，系统根据文档与query的相关性排序返回文档集合中的文档，而不是简单地返回所有满足query描述的文档集合
- 自由文本查询：用户query是自然语言的一个或多个词语而不是由查询语言构造的表达式
- 总体上，排序检索模型中有布尔查询和自由文本查询两种方式，但是实际中排序检索模型总是与自由文本查询联系在一起，反之亦然

“盛宴” or “饥荒”：不再是问题

- 当系统给出的是**有序的查询**结果，**查询结果数目多不再是问题**
 - 事实上，结果的数目不再是问题
 - 我们只需要给出**top K（10左右）**个结果
 - 为用户减轻负担

前提：合适的排序算法

排序检索的基本——评分

- 我们希望根据文档**对查询者的有用性**大小顺序将文档返回给查询者
- **怎样根据一个query对文档进行排序？**
- 给每个“查询-文档对”进行评分，在 $[0, 1]$ 之间
- **这个评分值衡量文档与query的匹配程度**

Query-document 评分

- 需要一种方法给一个“query-document对”评分
- 先以单个词组成的query为例
- 如果该词项不出现在文档中，该文档评分为0
- 该词项在文档中出现的频率越高，则评分越高
- 下面是几种备选方案

Jaccard 系数

- 一种常用的衡量两个集合重叠度的方法
- $\text{Jaccard}(A, B) = |A \cap B| / |A \cup B|$
- $\text{Jaccard}(A, A) = 1$
- $\text{Jaccard}(A, B) = 0$ if $A \cap B = 0$
- 集合A和B不需要具有同样的规模
- $\text{Jaccard}(A, B)$ 的取值在 $[0, 1]$

Jaccard 系数：举例

- 用Jaccard 系数的方法计算下面的query与文档之间的query-document评分
- Query: *ides of march*
- Document 1: *caesar died in march*
- Document 2: *the long march*

$$\text{Jaccard}(q, \text{doc1}) = 1/6$$

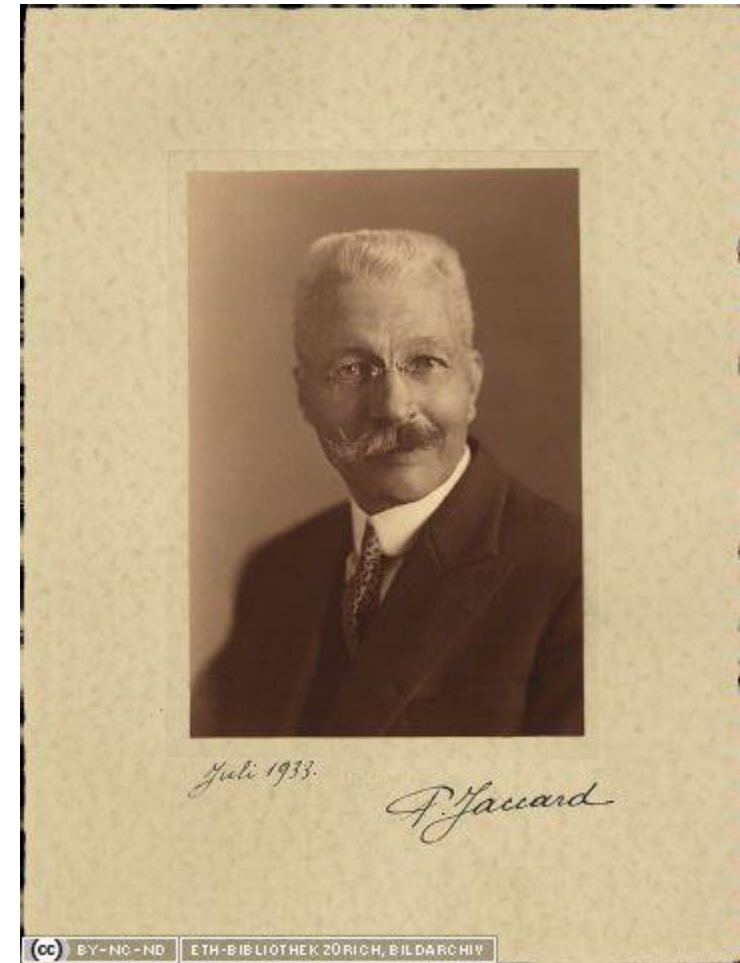
$$\text{Jaccard}(q, \text{doc2}) = 1/5$$

用Jaccard 系数评分的问题

- 没有考虑词项频率（词项在文档中出现的**次数**）
- **罕见词比高频词的信息量更大，更加具有区分度, Jaccard系数没有考虑这个信息**
- 需要一种复杂的方式来归一化长度
- 后面我们将用 $|A \cap B| / \sqrt{|A \cup B|}$ 而不是 $|A \cap B| / |A \cup B|$ 来归一化长度

Paul Jaccard (1868–1944)

- 瑞士植物学家，ETH教授
- 1894年毕业于苏黎世联邦理工学院ETH(出过包括爱因斯坦在内的21位诺贝尔奖得主)
- 1901年提出Jaccard Index即Jaccard Coefficient概念



本讲提纲

- 1 回顾
- 2 排序式检索
- 3 词项频率**
- 4 tf-idf权重计算
- 5 向量空间模型

词袋模型 (*Bag of words*)

- 不考虑词在文档中出现的顺序
- “John is quicker than Mary ” 和 “Mary is quicker than John ” 的表示结果一样
- 这就是**词袋模型**
- 从一定程度上讲，这是一种倒退，位置索引可以很容易区分这两个文档
- 在后面的课程中我们将可以看到如何“恢复”位置信息
- 现在只考虑：**词袋模型**

回顾（p3）：词项-文档关联矩阵

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	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

- 每个文档用一个二值向量表示 $\in \{0, 1\}^{|V|}$

词项-文档计数矩阵

- 考虑词项在文档中出现的次数
 - 将每个文档看成是一个计数向量：矩阵中的一列

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

词项频率 tf (Term frequency)

- 词项频率：词项 t 在文档 d 中出现的次数，记为 $tf_{t,d}$
- 如何利用 tf 计算 query-document 评分？
- 第一种方法是采用原始的 tf 值 (raw tf)
- 但是，原始 tf 值不太合适：
 - 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
- 相关性并不随词项频率成比例的增加

NB: frequency = count in IR

tf的对数表示

- 词项 t 在文档 d 中频率的对数表示

$$w_{t,d} = \begin{cases} 1 + \log_{10} \text{tf}_{t,d}, & \text{if } \text{tf}_{t,d} > 0 \\ 0, & \text{otherwise} \end{cases}$$

- $\text{tf}_{t,d} \rightarrow w_{t,d}$:

$0 \rightarrow 0, 1 \rightarrow 1, 2 \rightarrow 1.3, 10 \rightarrow 2, 1000 \rightarrow 4$

- **文档-词项的匹配得分**是所有同时出现在 q 和文档 d 中的词项的对数词频之和

$$\text{Score}(q, d) = \sum_{t \in q \cap d} (1 + \log \text{tf}_{t,d})$$

- **评分为0**，表示文档和query中没有公共词项

本讲提纲

- ① 回顾
- ② 排序式检索
- ③ 词项频率
- ④ **tf-idf权重计算**
- ⑤ 向量空间模型

文档中的词频 vs. 文档集中的词频

- 除词项频率 tf 之外，我们还想利用词项在整个文档集中的频率进行权重和评分计算

罕见词项所期望的权重

- 罕见词项比常见词所蕴含的信息更多
- 考虑查询中某个词项，它在整个文档集中非常罕见 (例如 ARACHNOCENTRIC).
- 某篇包含该词项的文档很可能相关
- 于是，我们希望像ARACHNOCENTRIC一样的罕见词项将有较高权重

常见词项所期望的权重

- 常见词项的信息量不如罕见词
- 考虑一个查询词项，它频繁出现在文档集中 (如 GOOD, INCREASE, LINE 等等)
- 一篇包含该词项的文档当然比不包含该词项的文档的相关度要高
- 但是，这些词对于相关度而言 **并不是非常强** 的指示词
- 于是，对于诸如 GOOD、INCREASE 和 LINE 的频繁词，**会给一个正的权重，但是这个权重小于罕见词权重**

文档频率 (Document frequency, df)

- 对于罕见词项我们希望赋予高权重
- 对于常见词项我们希望赋予正的低权重
- 接下来我们使用文档频率df这个因子来计算查询-文档的匹配得分
- 文档频率是指: 出现词项的文档数目

idf 权重

- df_t 是词项 t 的文档频率：文档集合中包含 t 的文档数目
 - df_t 与词项 t 包含的信息量成反比
 - $df_t \leq N$ (N 是文档的总数)
- 定义 t 的逆文档频率为 idf

$$idf_t = \log_{10} (N/df_t)$$

- idf_t 是反映词项 t 的信息量的一个指标
- 用 $\log (N/df_t)$ 代替 N/df_t 来抑制 idf 的作用

对数的底不会对文档的相对排序产生实际影响（习题6-12）

idf的计算举例 $N=1,000,000$

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

$$idf_t = \log_{10} (N/df_t)$$

文档集合中每个词项t都有一个逆文档频率 idf_t

idf对排序的影响

- 对于含有两个以上查询词的query，idf才会影响排序结果
- 例如：
 - Query为“arachnocentric line”，idf会提高“arachnocentric”的相对权重，同时降低“line”的相对权重。
- 对于只有一个查询词的query，idf对排序结果没有影响

文档集频率 vs. 文档频率

- 文档集频率(collection frequency, cf)是指 t 在整个文档集中出现的次数;
- 文档频率(document frequency, df) 包含词项 t 的文档数目
- 例如

词项	文档集频率(cf)	文档频率(df)
insurance	10440	3997
try	10422	8760

- 哪个词项更适合作为query? 即应该赋予更高的权重
- 上例表明, df (和idf) 比cf (和“icf”) 更适合权重计算

tf-idf权重（单个词）

- tf-idf 是信息检索中最著名的权重计算方法
 - 注意： tf-idf中“-”是连接号而不是减号
 - 还可以写成： tf.idf, tf x idf
- 词项t的tf-idf 是由它的tf和idf组合而成

$$w_{t,d} = (1 + \log \text{tf}_{t,d}) \times \log_{10}(N / \text{df}_t)$$

- tf-idf值随着词项在单个文档中出现次数增加而增大
- tf-idf值随着词项在文档集中数目减少而增加

Query的最终文档排序 (Query词)

$$\text{Score}(q, d) = \sum_{t \in q \cap d} \text{tf.idf}_{t,d}$$

本讲提纲

- ① 回顾
- ② 排序式检索
- ③ 词项频率
- ④ tf-idf权重计算
- ⑤ 向量空间模型

二值关联矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbet h . . .
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
. . .						

每篇文档表示成一个二值向量 $\in \{0, 1\}^M$

词频矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbet h . . .
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
. . .						

每篇文档表示成一个词频向量 $\in \mathbb{N}^M$

二值 \rightarrow 词频 \rightarrow 权重矩阵

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbet h . . .
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 $\in \mathbb{R}^M$

文档表示成向量

- 每篇文档表示成一个基于tf-idf权重的实值向量 $\in \mathbb{R}^{|V|}$.
- 于是，我们有一个 $|V|$ 维实向量空间
- 空间的每一维都对应词项
- 文档是空间的点或者向量
- **维度非常高**：特别是互联网搜索引擎，空间可能达到千万维或更高
- **向量空间非常稀疏**：对每个向量来说大部分都是0

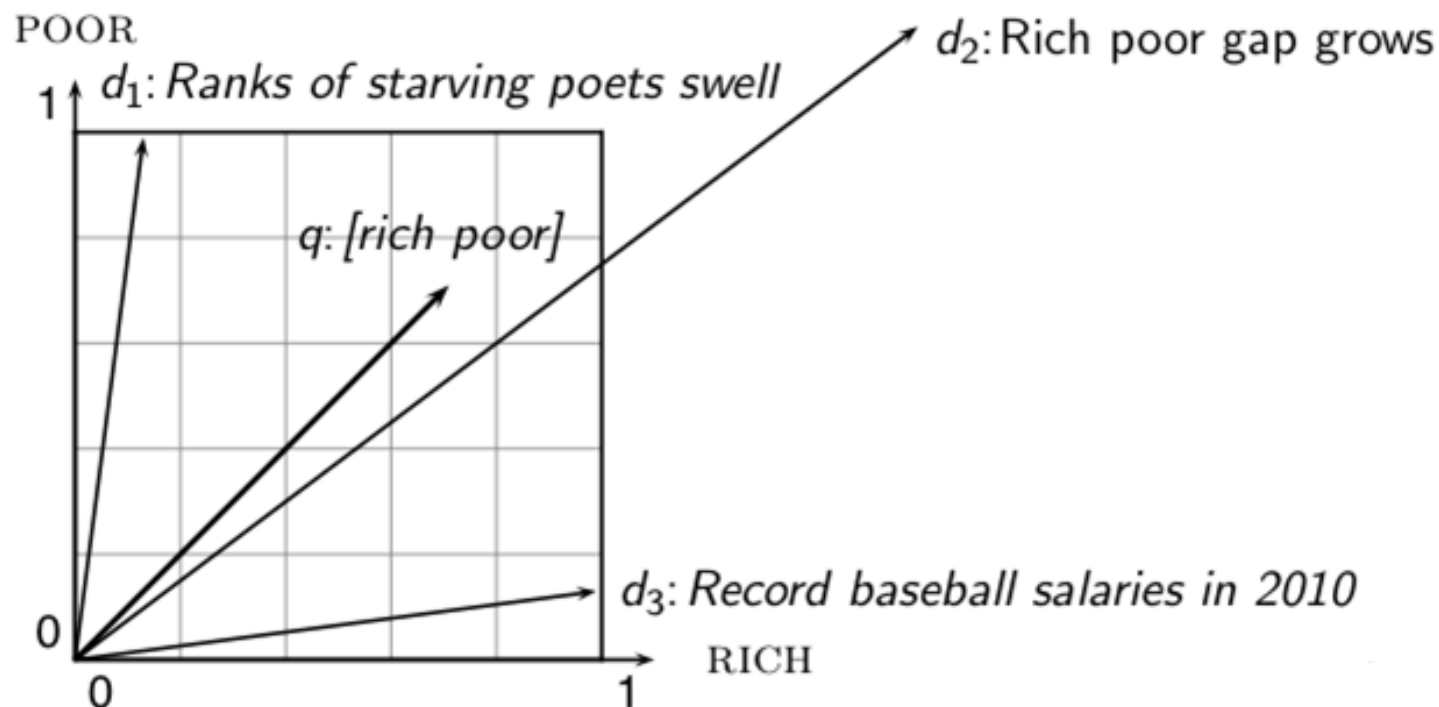
Queries表示成向量

- 关键思路1: 对于查询做同样的处理, 即将查询表示成同一高维空间的向量
- 关键思路2: 在向量空间内根据queries与文档向量间的距离来排序

向量相似度计算

- 第一步：计算两点之间距离
 - 两个向量终点间距离
- 欧氏距离？ ——bad idea
- 因为，欧氏距离对向量长度很敏感

为什么欧氏距离不好？



- 在欧氏空间， q 与文档 d_2 的欧氏距离很大，虽然 q 与 d_2 的分布很相近

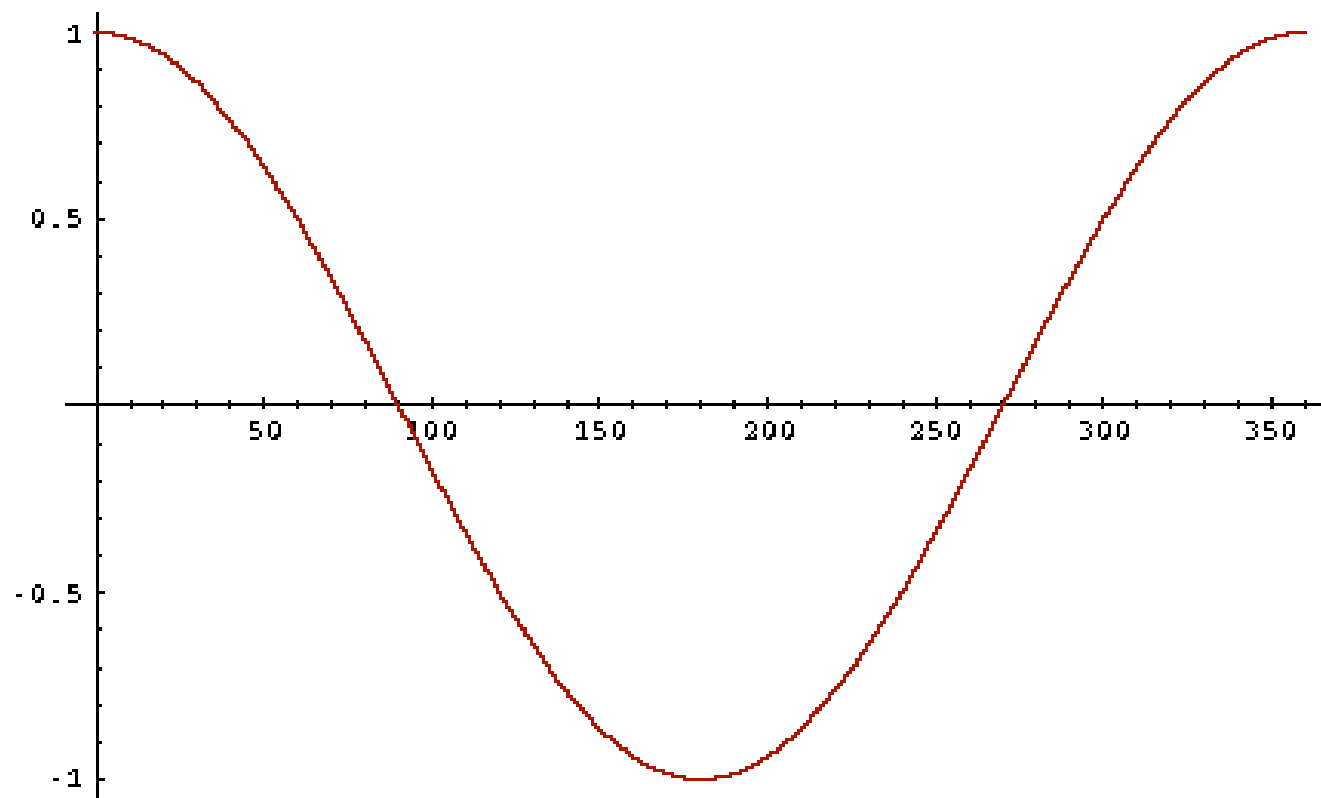
利用夹角代替距离

- 举个例子：将一篇文档 d 的内容复制一份加在自身末尾，构造一个新文档 d'
- 从语义上看，文档 d 和 d' 的内容是相同的
- 这两个文档的向量间夹角为0，表示最大相似度
- 但是这两个文档的欧氏距离却是非常大
- 结论：可以通过计算文档与query的夹角给文档排序

余弦相似度

- 下面两个观点是等价的：
 - 按query与文档夹角递减给文档排序
 - 按 $\cos(\text{query}, \text{document})$ 递增给文档排序
- 这是因为在 $[0^\circ, 180^\circ]$ 区间上，Cosine是单调递减函数

余弦相似度



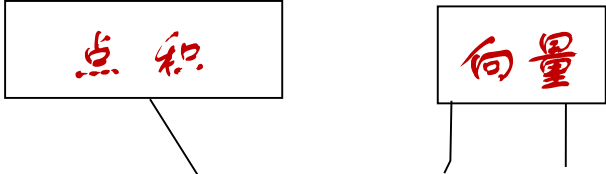
文档长度归一化

- 可以用 L_2 范数对文档长度进行归一化，文档 x 的 L_2 范数为：

$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

- 一个文档向量除以它的 L_2 范数就是给这个文档进行了长度归一化
- 进行了长度归一化后，前一页中的文档 d 和 d' 就可以用同一个向量表示了
 - 这样长文档和短文档之间的长度差异就不会影响相关性了

cosine(query, document)



The diagram shows two boxes at the top. The left box contains the Chinese characters '点积' (dot product) and has a line pointing to the dot product symbol \bullet in the formula. The right box contains the Chinese characters '向量' (vector) and has lines pointing to the vector symbols \vec{q} and \vec{d} in the formula.

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

- q_i 是在query中词项i的tf-idf 权值
- d_i 是在文档中词项i的tf-idf 权值
- $\cos(\vec{q}, \vec{d})$ \vec{q} 与 \vec{d} 的余弦相关性
- 等价于向量 \vec{q} 与 \vec{d} 夹角的余弦值

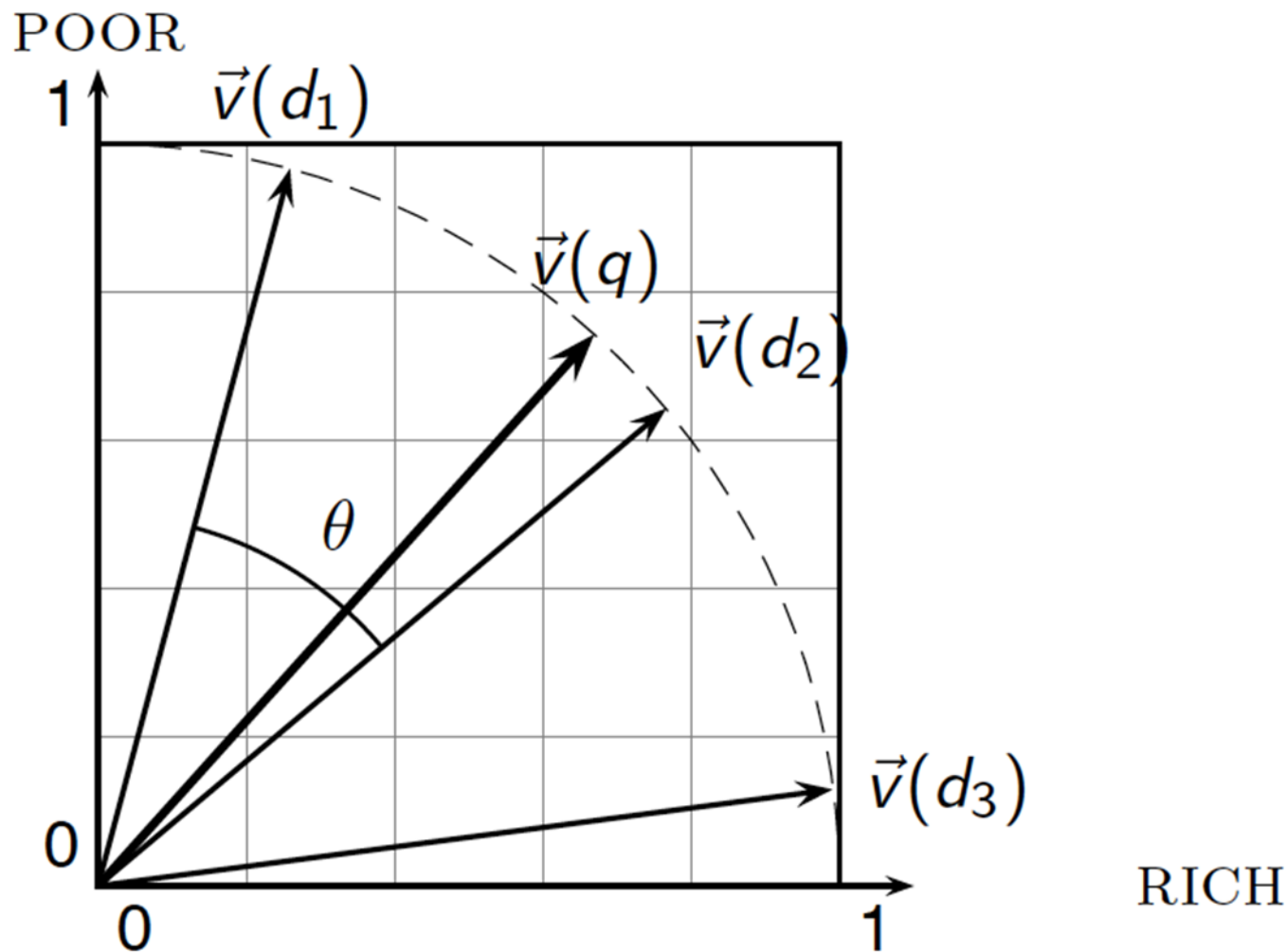
向量长度归一化后的余弦计算

- 长度归一化后，余弦的计算可直接通过点积的方式得到：

$$\cos(\vec{q}, \vec{d}) = \vec{q} \bullet \vec{d} = \sum_{i=1}^{|V|} q_i d_i$$

- 其中 \vec{q}, \vec{d} 是长度归一化后的向量（注意与前一页的公式的区别）

余弦相似度图示



余弦相似度计算示例

- 计算下面三部小说相似度
- **SaS**: *Sense and Sensibility* (**理智与情感**)
- **PaP**: *Pride and Prejudice* (**傲慢与偏见**)
- **WH**: *Wuthering Heights*? (**呼啸山庄**)

term	SaS	PaP	WH
affection	115	58	20
jealous	10	7	11
gossip	2	0	6
wuthering	0	0	38

词项频率 (tf)

暂不考虑idf权值

余弦相似度示例

词项频率取对数

term	SaS	PaP	WH
affection	3.06	2.76	2.30
jealous	2.00	1.85	2.04
gossip	1.30	0	1.78
wuthering	0	0	2.58

长度归一化

term	SaS	PaP	WH
affection	0.789	0.832	0.524
jealous	0.515	0.555	0.465
gossip	0.335	0	0.405
wuthering	0	0	0.588

$$\cos(\text{SaS}, \text{PaP}) \approx ? \quad 0.789 \times 0.832 + 0.515 \times 0.555 + 0.335 \times 0.0 + 0.0 \times 0.0 \approx 0.94$$

$$\cos(\text{SaS}, \text{WH}) \approx ? \quad 0.79$$

$$\cos(\text{PaP}, \text{WH}) \approx ? \quad 0.69$$

计算cosine

COSINESCORE(q)

```
1  float Scores[ $N$ ] = 0
2  float 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  $Scores[d] += w_{t,d} \times w_{t,q}$ 
7  Read the array  $Length$ 
8  for each  $d$ 
9  do  $Scores[d] = Scores[d] / Length[d]$ 
10 return Top  $K$  components of  $Scores[]$ 
```

tf-idf 权重机制变形

Term frequency		Document frequency		Normalization	
n (natural)	$tf_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(tf_{t,d})$	t (idf)	$\log \frac{N}{df_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times tf_{t,d}}{\max_t(tf_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N - df_t}{df_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^\alpha$, $\alpha < 1$
L (log ave)	$\frac{1 + \log(tf_{t,d})}{1 + \log(\text{ave}_{t \in d}(tf_{t,d}))}$				

文档权重和query权重（p89）

- 不同检索系统中的**权重机制**并不相同
- SMART标记：即对于每种不同的权重计算方法采用不同的标记。文档向量和query向量权重计算方法的组合字母表示为 *ddd. qq*
 - 例如： *lnc. ltn*
 - 文档：对数tf，无idf因子，余弦长度归一化
 - 查询：对数tf，idf，无归一化

Inc. Itc举例 (p86)

- 文档: *car insurance auto insurance*
- Query: *best car insurance*

词项	查询						文档				内积
	tf-raw	tf-wt	df	idf	wt	n'lize	tf-raw	tf-wt	wt	n'lize	
auto	0	0	5000	2.3	0	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0.34	0	0	0	0	0
car	1	1	10000	2.0	2.0	0.52	1	1	1	0.52	0.27
insurance	1	1	1000	3.0	3.0	0.78	2	1.3	1.3	0.68	0.53

$$\text{文档长度} = \sqrt{1^2 + 0^2 + 1^2 + 1.3^2} \approx 1.92$$

$$\text{Score} = 0 + 0 + 0.27 + 0.53 = 0.8$$

向量空间模型小结

- 将query看作带tf-idf权重的向量
- 将每个文档也看作带tf-idf权重的向量
- 计算query向量与每个文档向量间的余弦相似度
- 根据相似度大小将文档排序
- 将top K 个结果返回给用户

Gerard Salton (1927–1995)

- 信息检索领域的奠基人之一，向量空间模型的完善者和倡导者，SMART系统的主要研制者，ACM Fellow
- 1958年毕业于哈佛大学应用数学专业，是Howard Aiken的关门博士生。Howard Aiken是IBM第一台大型机ASCC的研制负责人。
- 是康奈尔大学计算机系的创建者之一。



本讲内容

- 对搜索结果排序(Ranking)：为什么排序相当重要？
- 词项频率(Term Frequency, TF)：排序中的重要因子
- tf-idf 权重计算方法：最出名的经典排序方法
- 向量空间模型(Vector space model)：信息检索中最重要的形式化模型之一（其他模型还包括布尔模型和概率模型）

思考题

- 习题6-10
- 习题6-15
- 习题6-17
- 习题6-19

谢谢大家!