信息检索导论

An Introduction to Information Retrieval

第7讲　完整搜索系统中的评分计算

Scores in a complete search system

授课人：李波

中国科学院信息工程研究所/国科大网络空间安全学院

*改编自”An introduction to Information retrieval”网上公开的课件，地址 http://nlp.stanford.edu/IR-book/

提纲

① 上一讲回顾

② 结果排序的动机

③ 再论余弦相似度

④ 结果排序的实现

⑤ 完整的搜索系统

提纲

① 上一讲回顾

② 结果排序的动机

③ 再论余弦相似度

④ 结果排序的实现

⑤ 完整的搜索系统

词项频率tf

- t 在 d 中的对数词频权重定义如下：

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

- 文档-词项的匹配得分 $\sum_{t \in q \cap d} (1 + \log \mathrm{tf}_{t,d})$

idf权重

- df_t 是出现词项t的文档数目
- df_t 是和词项t的信息量成反比的一个值
- 于是可以定义词项t的idf权重:

$$idf_t = \log_{10} \frac{N}{df_t}$$

(其中N 是文档集中文档的数目)

- idf_t 是反映词项t的信息量的一个指标

tf-idf权重计算

- 词项的tf-idf权重是tf权重和idf权重的乘积

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

- 信息检索中最出名的权重计算方法之一

tf-idf + 归一化方法

词项频率tf		文档频率df		归一化方法	
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)	$\frac{1}{\sqrt{w_1^2+w_2^2+\cdots+w_M^2}}$
a(augamented)	$0.5+\frac{0.5\times tf_{t,d}}{\max_t(tf_{t,d})}$	p(prob idf)	$\max\left\{0,\log\frac{N-df_t}{df_t}\right\}$	u(pivoted unique)	1/*u*(17.4.4节)
b(boolean)	$\begin{cases}1 \text{ if } tf_{t,d}>0\\0 \text{ otherwise}\end{cases}$			b(byte size)	$1/CharLength^a, a<1$
L(log ave)	$\frac{1+\log(tf_{t,d})}{1+\log(ave_{t\in d}(tf_{t,d}))}$				

向量空间模型

	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tfidf权重的实值向量 $\in R^{|V|}$
查询也表示成同一空间下的基于tfidf权重的实值向量
查询和文档之间就可以计算相似度

tf-idf 计算样例: lnc.ltn

word	query					document				product
	tf-raw	tf-wght	df	idf	weight	tf-raw	tf-wght	weight	n'lized	
auto	0	0	5000	2.3	0	1	1	1	0.52	0
best	1	1	50000	1.3	1.3	0	0	0	0	0
car	1	1	10000	2.0	2.0	1	1	1	0.52	1.04
insurance	1	1	1000	3.0	3.0	2	1.3	1.3	0.68	2.04

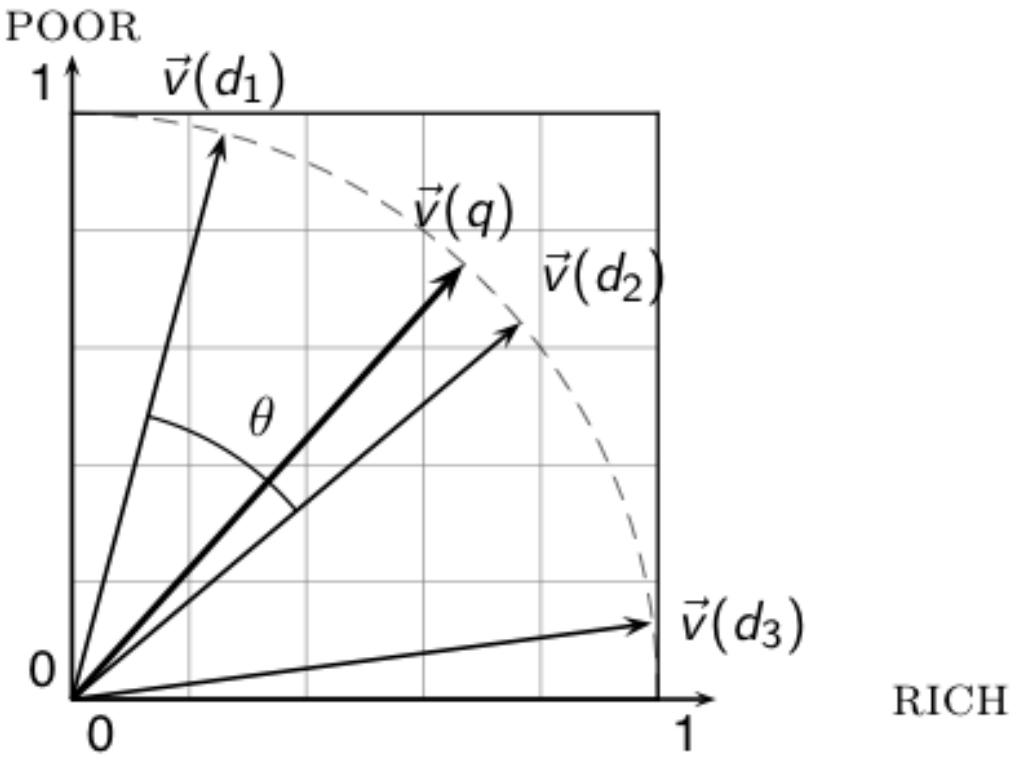
- 最终结果 $0 + 0 + 1.04 + 2.04 = 3.08$

查询和文档之间的余弦相似度计算

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

- q_i 是第i 个词项在查询q中的tf-idf权重
- d_i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 $\vec{d}$ 夹角的余弦

余弦相似度计算的图示



传统布尔模型和向量空间模型的优缺点

优缺点 / 模型	优　点	缺　点
传统布尔模型	检索式的结构化—用布尔算法明确的揭示了索引项之间的关系。	（1）不能对结果按相似度进行排序； （2）不能控制返回文档的数量； （3）不能进行相关性反馈。
向量空间模型	(1)检索结果的相关性排序； (2)可以控制输出结果的数量； (3)能够进行相关性反馈。	（1）理论上不够，基于直觉的经验性公式； （2）认为索引项相互独立，未能揭示词语之间的关系。 （例如 王励勤 乒乓球 的出现是不独立的）

本讲内容

- 排序的重要性：从用户的角度来看(Google的用户研究结果)
- 另一种长度归一化: 回转(Pivoted)长度归一化
- 排序实现
- 完整的搜索系统

提纲

① 上一讲回顾

② 结果排序的动机

③ 再论余弦相似度

④ 结果排序的实现

⑤ 完整的搜索系统

排序的重要性

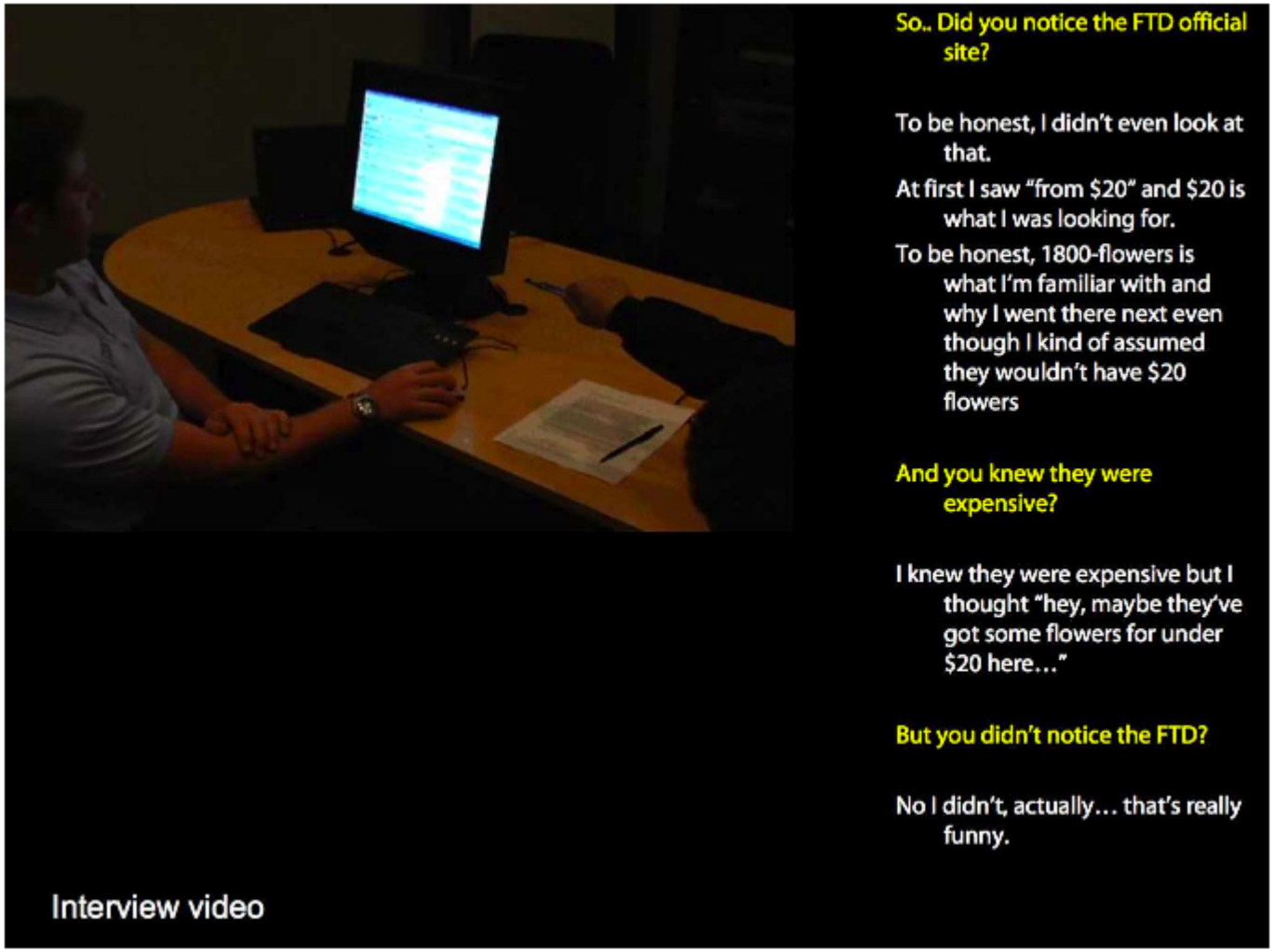
- 上一讲: 不排序后果很严重！
  - 用户只希望看到一些而不是成千上万的结果
  - 很难构造只产生一些结果的查询，即使是专家也很难
  - 排序能够将成千上万条结果缩减至几条结果，因此非常重要
- 接下来: 将介绍用户的相关行为数据
- 实际上，大部分用户只看1到3条结果

检索效果的经验性观察方法

- 如何度量排序的重要性？
- 可以在某种受控配置观察下搜索用户的行为
  - 对用户行为进行录像
  - 让他们边想边说出来
  - 访谈
  - 眼球跟踪
  - 计时
  - 记录点击和点击次数

- 下面的讲义来自Dan Russell在JCDL2007会议上的大会报告
- Dan Russell是Google的 Über Tech Lead for Search Quality & User Happiness

用户访谈



FTD：一个网上花店，1800-flower：另一个网上花店

用户对结果的浏览模式

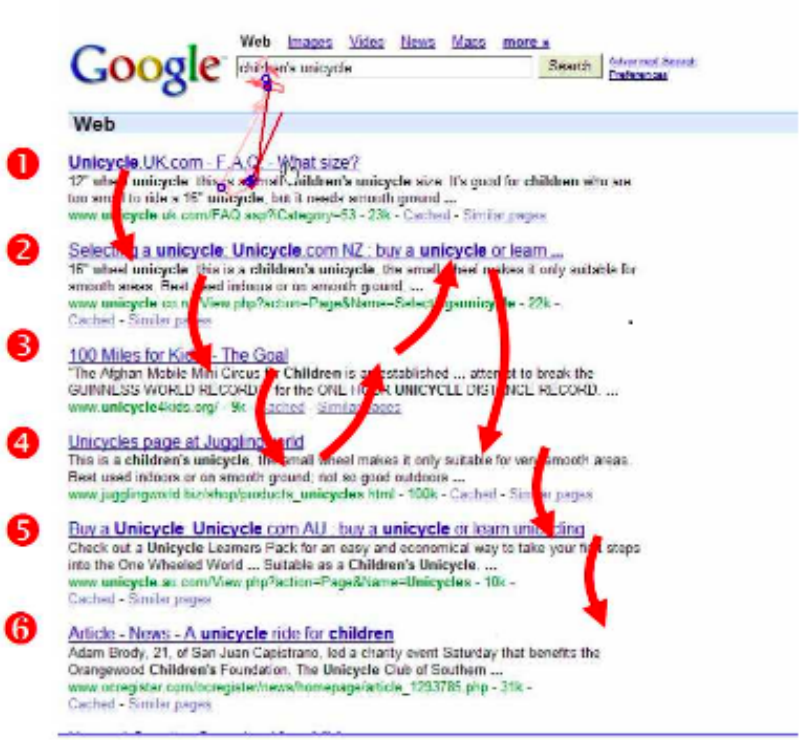
Rapidly scanning the results

Note scan pattern:

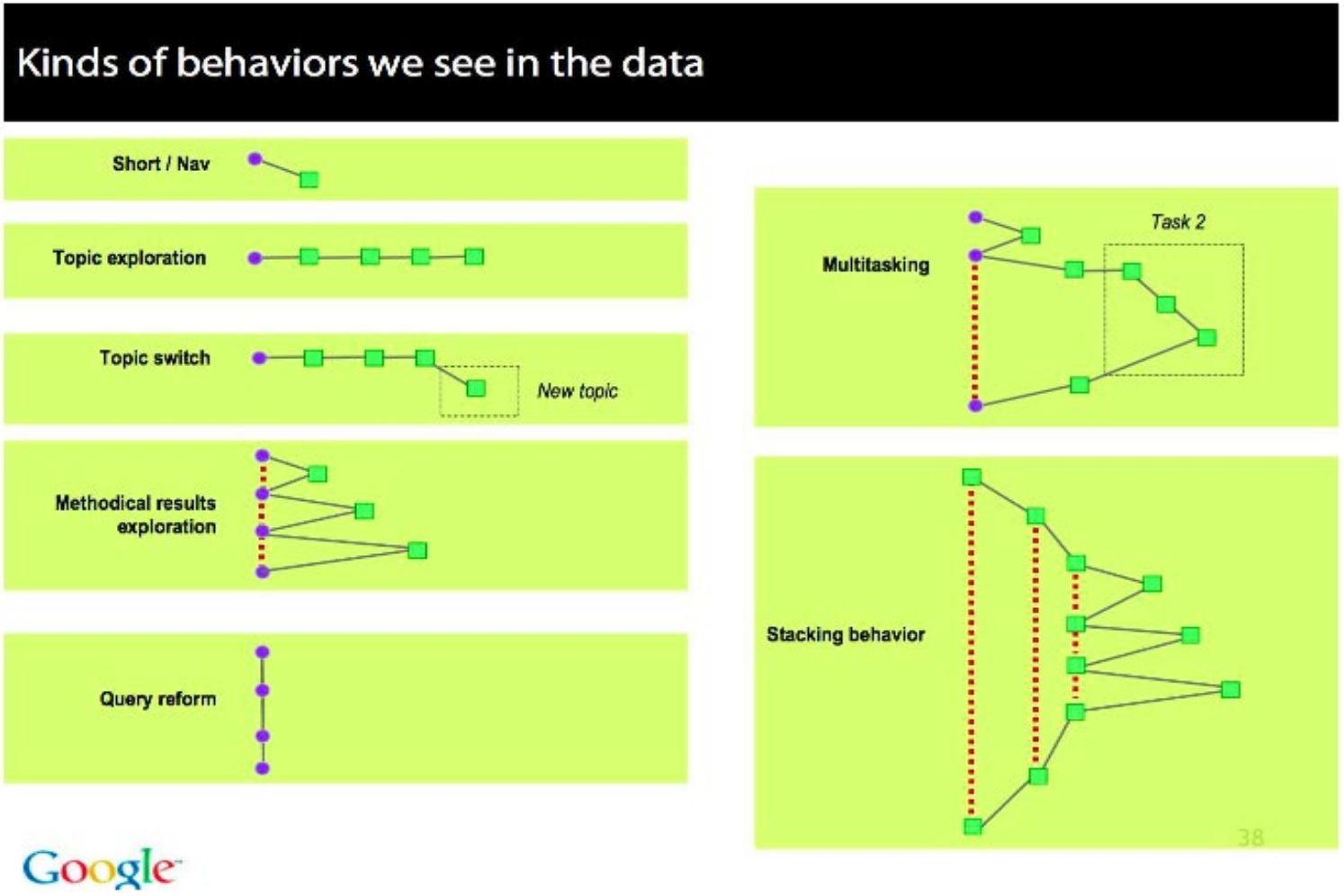
Page 3:	Result 1
	Result 2
	Result 3
	Result 4
	Result 3
	Result 2
	Result 4
	Result 5
	Result 6 <click>

Q: Why do this?

A: What's learned later influences judgment of earlier content.

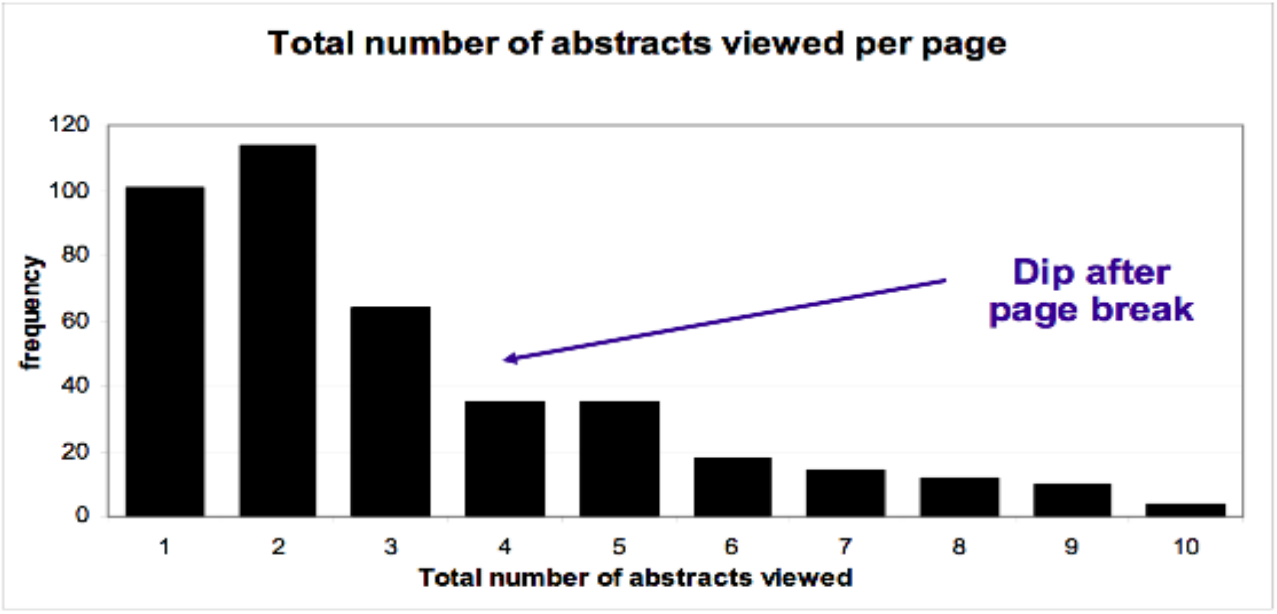


检索中的用户行为模式


Kinds of behaviors we see in the data
Short / Nav
Topic exploration
Topic switch
New topic
Methodical results exploration
Query reform
Multitasking
Task 2
Stacking behavior
38
Google

用户浏览的链接数

How many links do users view?

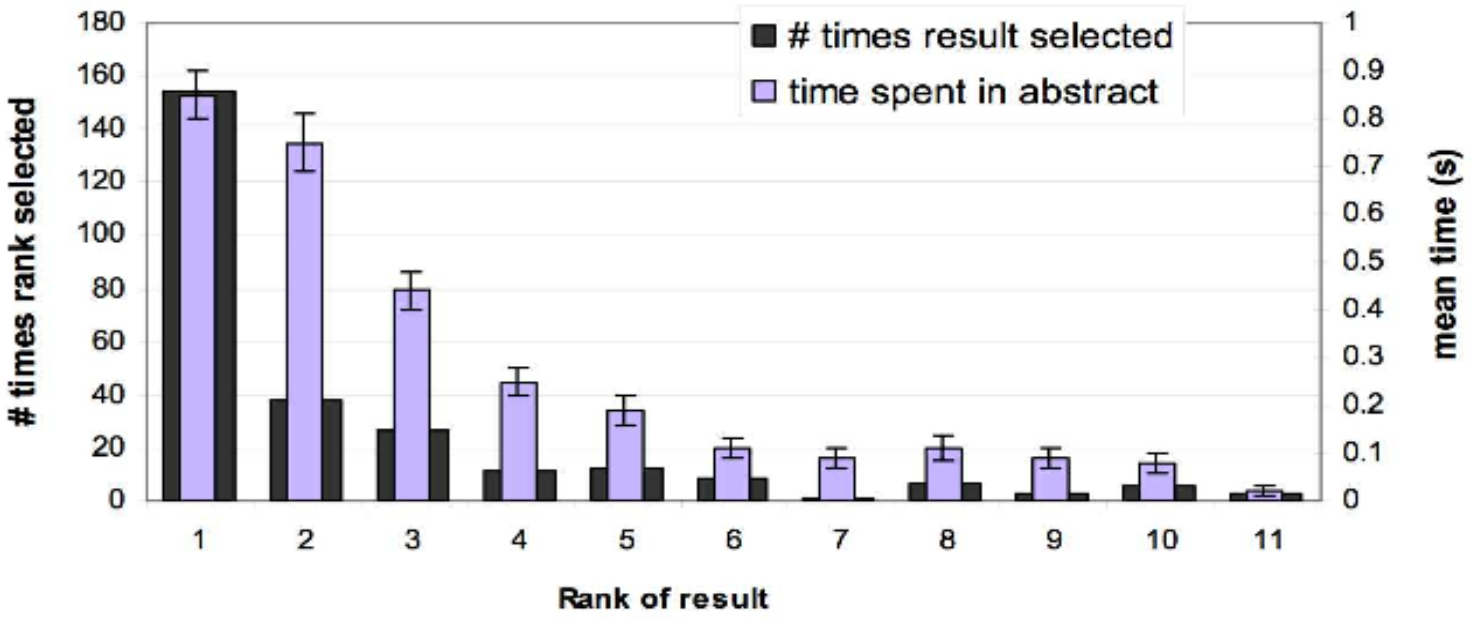


Mean: 3.07 Median/Mode: 2.00

Google

浏览 vs. 点击

Looking vs. Clicking



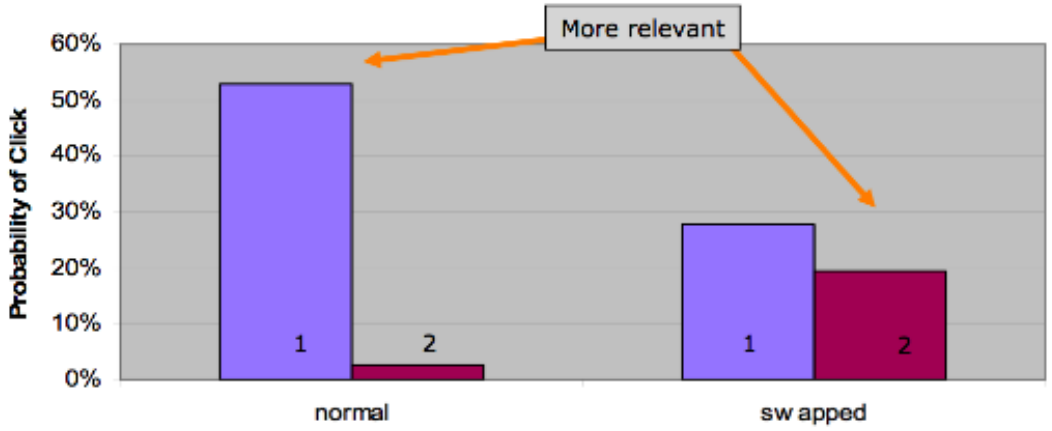
- Users view results one and two more often / thoroughly
- Users click most frequently on result one

Google

结果显示顺序对行为的影响

Presentation bias – reversed results

- Order of presentation influences where users look **AND** where they click



Google

排序的重要性: 小结

- 摘要阅读(Viewing abstracts): 用户更可能阅读前几篇(1, 2, 3, 4)结果的摘要
- 点击(Clicking): 点击的分布甚至更有偏向性
  - 一半情况下，用户点击排名最高的页面
  - 即使排名最高的页面不相关，仍然有30%的用户会点击它。
- → 正确排序相当重要
- → 排对最高的页面非常重要

提纲

① 上一讲回顾

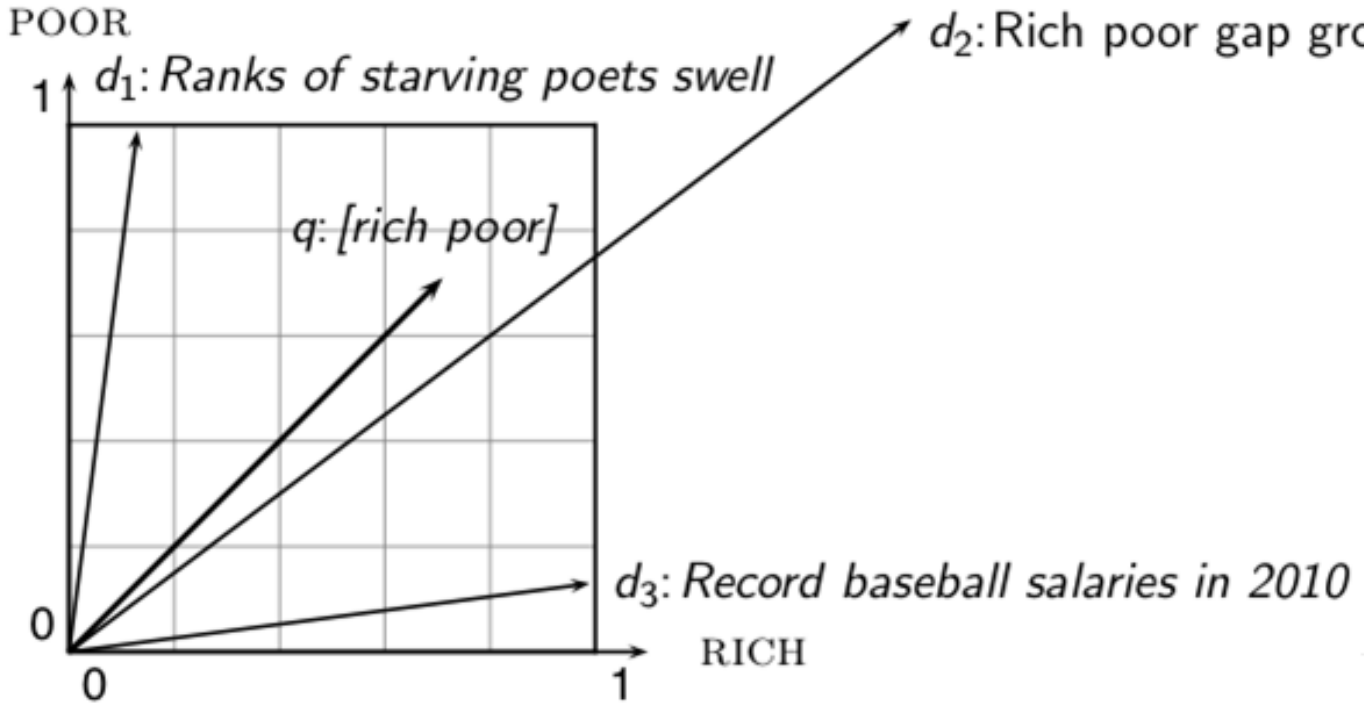
② 结果排序的动机

③ 再论余弦相似度

④ 结果排序的实现

⑤ 完整的搜索系统

距离函数不适合度量相似度



- 尽管查询q和文档d_2的内容很相似，但是向量 $\vec{q}$ 和 $\vec{d}_2$ 的欧氏距离却很大。这也是为什么要进行长度归一化的原因，或者说，我们前面采用余弦相似度的原因。

查询和文档之间的余弦相似度计算

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

- q_i 是第i 个词项在查询q中的tf-idf权重
- d_i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 $\vec{d}$ 夹角的余弦

余弦相似度计算的一个例子

- 查询和文档进行向量的相似度计算：
  - 采用内积：
    - 文档d_1与q的内积：1*1+3*2=7
    - 文档d_2与q的内积：2*2=4
  - 夹角余弦：
    - 文档d_1与q的夹角余弦：$\frac{7}{\sqrt{12\times5}}\approx0.90$
    - 文档d_2与q的夹角余弦：$\frac{4}{\sqrt{5\times8}}\approx0.63$

	d_1	d_2	q
2002	0	1	0
2006	1	0	1
世界杯	3	2	2
德国	1	0	0
韩国	0	1	0
日本	0	1	0
举行	1	1	0

Q: 2006世界杯
d1：2006年世界杯在德国举行
d2：2002年世界杯在韩国和日本举行

问题：如果d1的长度不断增大，结果会怎么样？

课堂练习：余弦相似度的一个问题

- 查询 q: “anti-doping rules Beijing 2008 Olympics” 反兴奋剂
- 计算并比较如下的三篇文档
  - d1: 一篇有关“ anti-doping rules at 2008 Olympics”的短文档
  - d2: 一篇包含d1 以及其他5篇新闻报道的长文档，其中这5篇新闻报道的主题都与Olympics/anti-doping无关
  - d3: 一篇有关“ anti-doping rules at the 2004 Athens Olympics ”的短文档
- 我们期望的结果是什么？
- 如何实现上述结果？

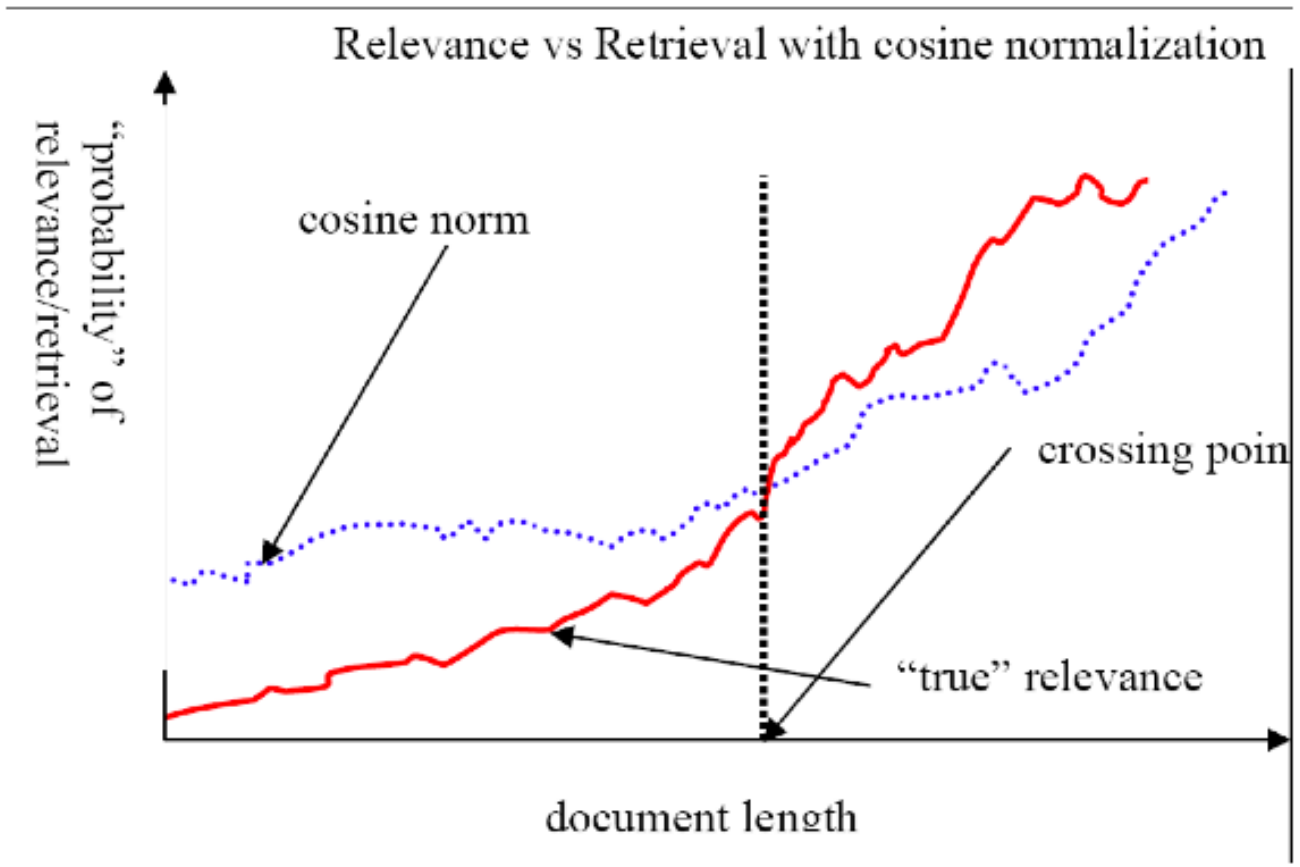
回转归一化

- 余弦归一化倾向于短文档，即对短文档产生的相关度预测高于真实值(也即此时归一化因子太大)，而平均而言对长文档产生的归一化因子太小

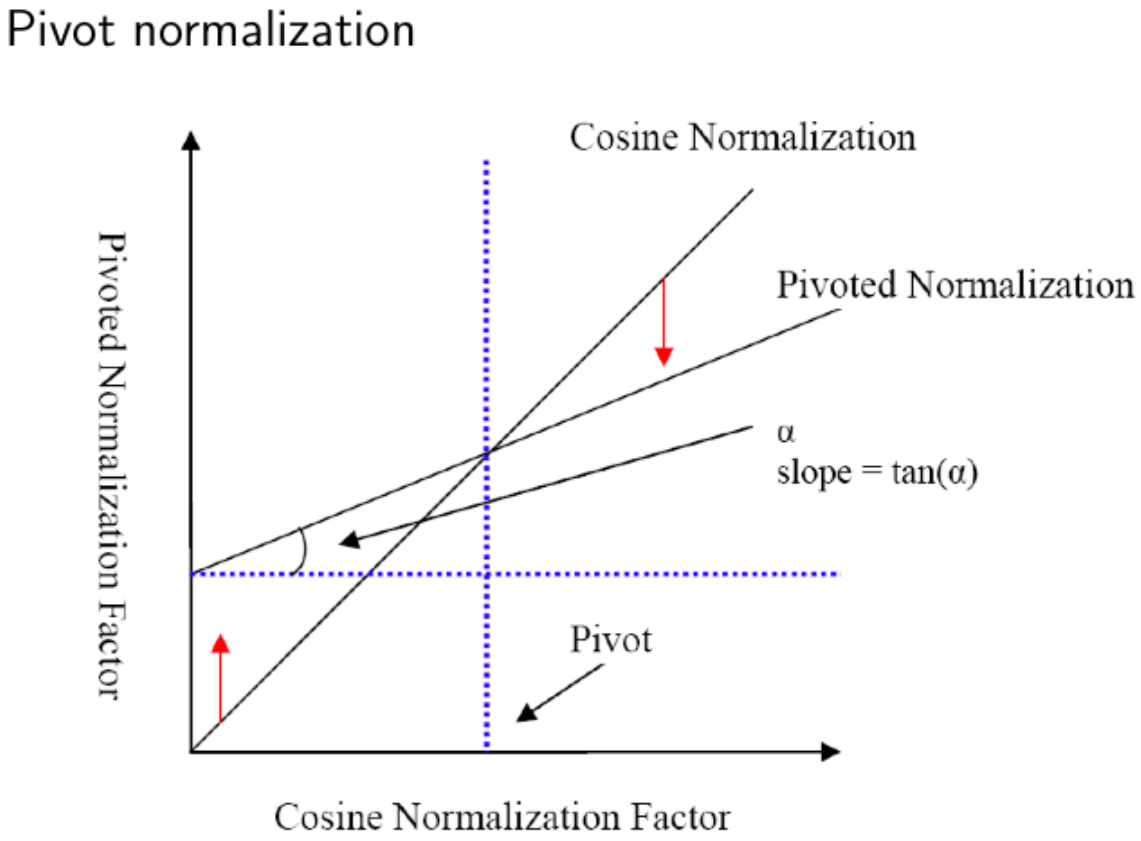
$$c(\text{cosine}) \qquad \frac{1}{\sqrt{w_1^2 + w_2^2 + \cdots + w_M^2}}$$

- 于是可以先找到一个支点(pivot，平衡点)，然后通过这个支点对余弦归一化操作进行线性调整。
- 效果：短文档的相似度降低，而长文档的相似度增大
- 这可以去除原来余弦归一化偏向短文档的问题

预测相关性概率 vs. 真实相关性概率



回转归一化(Pivot normalization)



注意:本文中的归一化因子用的是 $\sqrt{w_1^2 + w_2^2 + \ldots + w_t^2}$

回转归一化: Amit Singhal的实验结果

Cosine	Pivoted Cosine Normalization				
	Slope				
	0.60	0.65	0.70	**0.75**	0.80
6,526	6,342	6,458	6,574	**6,629**	6,671
0.2840	0.3024	0.3097	0.3144	**0.3171**	0.3162
Improvement	+ 6.5%	+ 9.0%	+10.7%	**+11.7%**	+11.3%

- 结果第一行：返回的相关文档数目
- 结果第二行： 平均正确率
- 结果第三行： 平均正确率的提高百分比

Amit Singhal

- 1989年本科毕业于印度IIT (Indian Institute of Technology) Roorkee分校
- 1996年博士毕业于Cornell University，导师是Gerard Salton
- 其论文获得1996年SIGIR Best Student Paper Award

Pivoted Document Length Normalization - [翻译此页]
citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9950 - 网页快照
作者： A Singhal - 1996 - 被引用次数： 716 - 相关文章
Document length normalization is used to fairly retrieve documents ...
• 2551 – Introduction to Modern Information retrieval – Salton, Mcgill - 1983
• 1078 – Term weighting approaches in automatic text retrieval – Salton, Buckley ...
• 203 – Inference networks for document retrieval – Turtle, Croft - 1990

- 2000年加入Google，2001年被授予Google Fellow称号
- Google 排序团队负责人,被财富杂志(Fortune, 2010)誉为世界科技界最聪明的50个人之一

提纲

① 上一讲回顾

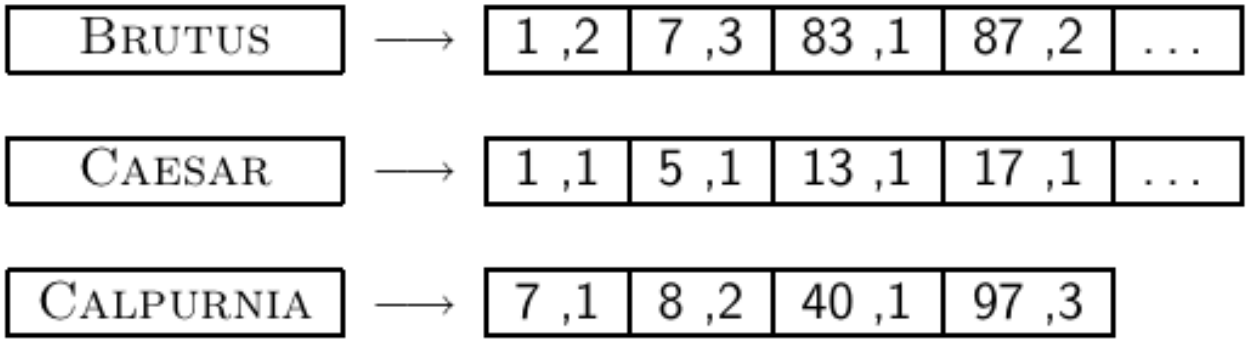
② 结果排序的动机

③ 再论余弦相似度

④ **结果排序的实现**

⑤ 完整的搜索系统

词项频率tf也存入倒排索引中



- 当然也需要位置信息，上面做了省略

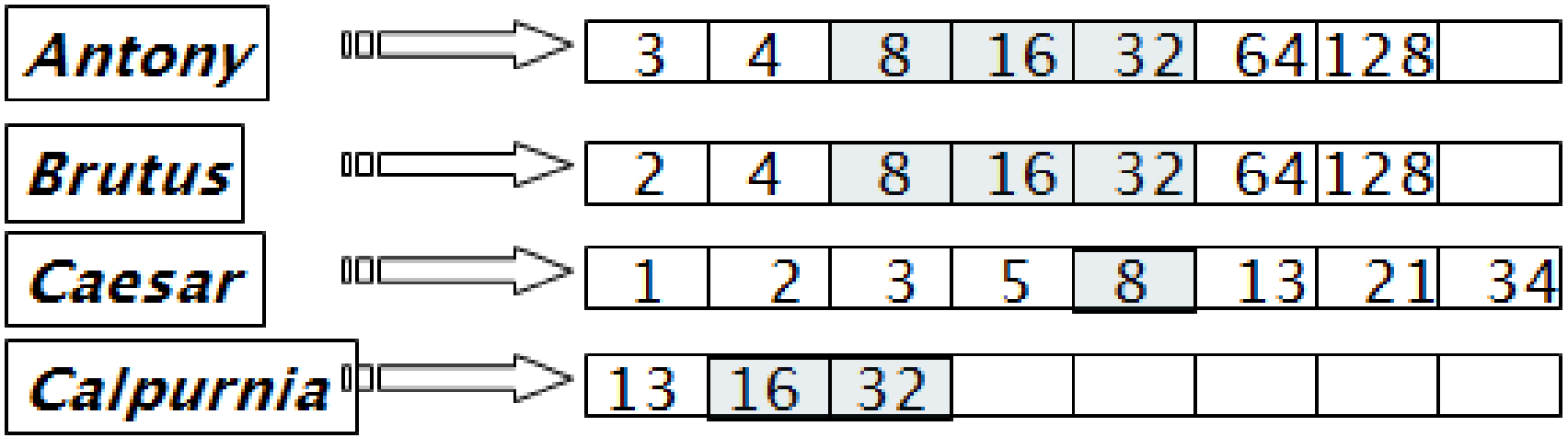
倒排索引中的tf存储

- 每条倒排记录中，除了docID 还要存储$tf_{t,d}$
- 通常存储是原始的整数词频，而不是对数词频对应的实数值，这是因为实数值不易压缩
- 对tf采用一元码编码效率很高
- 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
- 或者在可变字节码方式下每条倒排记录额外需要一个字节即可

余弦相似度计算算法

```
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} × 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[]
```

例子


Antony
3 4 8 16 32 64 128
Brutus
2 4 8 16 32 64 128
Caesar
1 2 3 5 8 13 21 34
Calpurnia
13 16 32

精确top k检索及其加速办法

- 目标：从文档集的所有文档中找出k个离查询最近的文档
- (一般)步骤：
  - 对每个文档评分(余弦相似度)
  - 按照评分高低排序
  - 选出前k个结果
- 如何加速：
  - 思路一：加快每个余弦相似度的计算
  - 思路二：不对所有文档的评分结果排序而直接选出Top k篇文档
  - 思路三：能否不需要计算所有N篇文档的得分？

精确top k检索加速方法一：快速计算余弦

- 检索排序就是找查询的k近邻
- 一般而言，在高维空间下，计算余弦相似度没有很高效的方法
- 但是如果查询很短，是有一定办法加速计算的，而且普通的索引能够支持这种快速计算

特例– 不考虑查询词项的权重

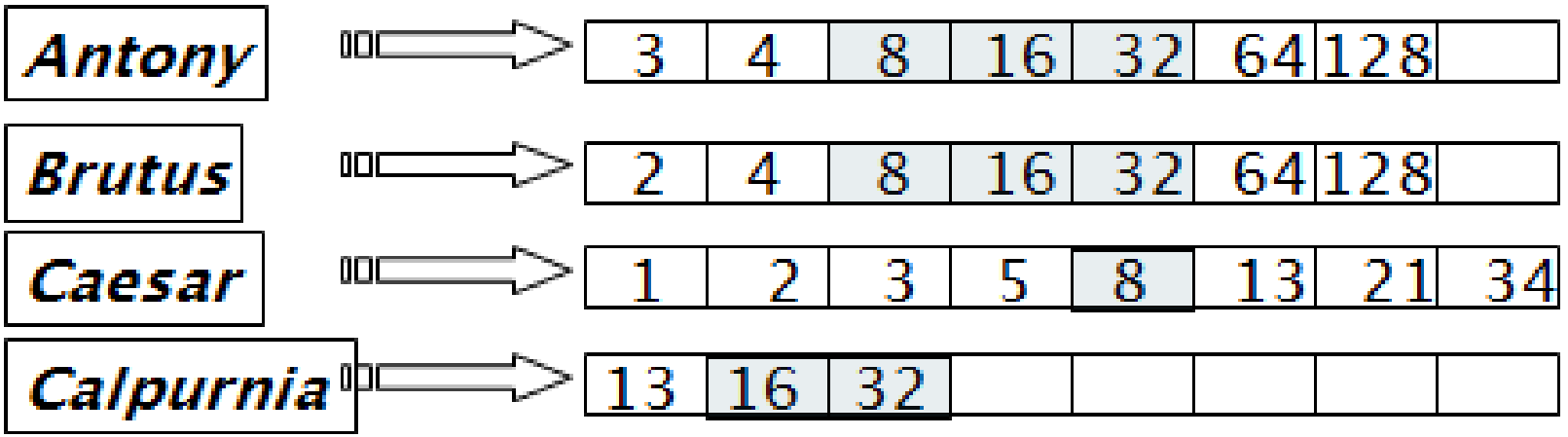
- 查询词项无权重
  - 相当于假设每个查询词项都出现1次
- 于是，不需要对查询向量进行归一化
  - 可以对上一讲给出的余弦相似度计算算法进行轻微的简化

快速余弦相似度计算: 无权重查询

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

Figure 7.1 A faster algorithm for vector space scores.

前面的例子


Antony
3 4 8 16 32 64 128
Brutus
2 4 8 16 32 64 128
Caesar
1 2 3 5 8 13 21 34
Calpurnia
13 16 32

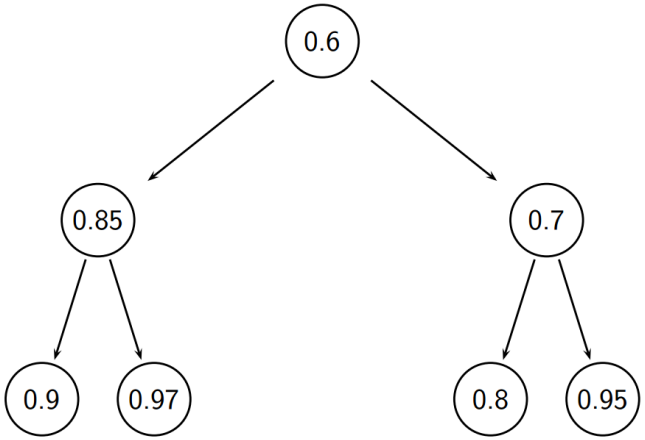
精确top k检索加速方法二：堆法N中选k

- 检索时，通常只需要返回前k条结果
  - 可以对所有的文档评分后排序，选出前k个结果，但是这个排序过程可以避免

- 令 J = 具有非零余弦相似度值的文档数目
  - 从J中选k个最大的

堆方法

- 堆：二叉树的一种，如果每个节点上的值> 子节点上的值，称为最大堆(Max Heap)，否则称为最小堆(Min Heap)
- 堆构建：构建包含k个节点的堆需O(J*logk)步
- 利用堆输出k个结果：需要O(klogk)步
- 如果 J=1M, k =100, 那么代价大概是全部排序代价的1/3



在O(Jlogk)步内找出top k的文档

- 目标：在每一步过程中，保留top k个文档
- 使用最小堆法
- 针对当前分值为s'的文档d'：
  - 从堆中找出当前最小分值hm (O(1))
  - 如果 $s' \leq hm$，忽略该文档
  - 如果$s' > hm$
    - Heap-delete-root (O(log k))
    - Heap-add d′/s′ (O(log k))
- 输出k个文档(O(klog k))
  - k次Heap-delete-root

精确top k检索加速方法三：提前终止计算

- 到目前为止的倒排记录表都按照docID排序
- 对于采用与查询无关的反映结果好坏程度的指标(静态质量)排序的情况
  - 例如: 页面d的PageRank g(d), 就是度量有多少好页面指向d的一种指标 (参考课本第 21章)
  - 于是可以将文档按照PageRank排序 $g(d1) > g(d2) > g(d3) > \ldots$
  - 将PageRank和余弦相似度线性组合得到文档的最后得分

$$\text{net-score}(q, d) = g(d) + \cos(q, d)$$

提前终止计算

- 假设:
  - (i) g → [0, 1];
  - (ii) 检索算法按照$d_1,d_2,\ldots$，依次计算(为文档为单位的计算，document-at-a-time)，当前处理的文档的 g(d) < 0.1；
  - (iii) 而目前找到的top k 的得分中最小的都 > 1.2
- 由于后续文档的得分不可能超过1.1 (cos(q,d) <1)
- 所以，我们已经得到了top k结果，不需要再进行后续计算

精确top k检索的问题

- 仍然无法避免大量文档参与计算
- 一个自然而言的问题就是能否尽量减少参与计算文档数目，即使不能完全保证正确性也在所不惜。
  - 即采用这种方法得到的top k虽然接近但是并非真正的top k ----非精确top k检索

非精确top k检索的可行性

- 检索是为了得到与查询匹配的结果，该结果要让用户满意
- 余弦相似度是刻画用户满意度的一种方法
- 非精确top k的结果如果和精确top k的结果相似度相差不大，应该也能让用户满意

一般思路

- 找一个文档集合A， k <|A|<<N，利用A中的top k 结果代替整个文档集的top k结果
  - 即给定查询后，A是整个文档集上近似剪枝得到的结果
  - 上述思路不仅适用于余弦相似度得分，也适用于其他相似度计算方法

方法一：索引去除(Index elimination)

- 一般检索方法中，通常只考虑至少包含一个查询词项的文档
- 可以进一步拓展这种思路
  - 只考虑那些包含高idf查询词项的文档
  - 只考虑那些包含多个查询词项的文档(比如达到一定比例，3个词项至少出现2个，4个中至少出现3个等等)

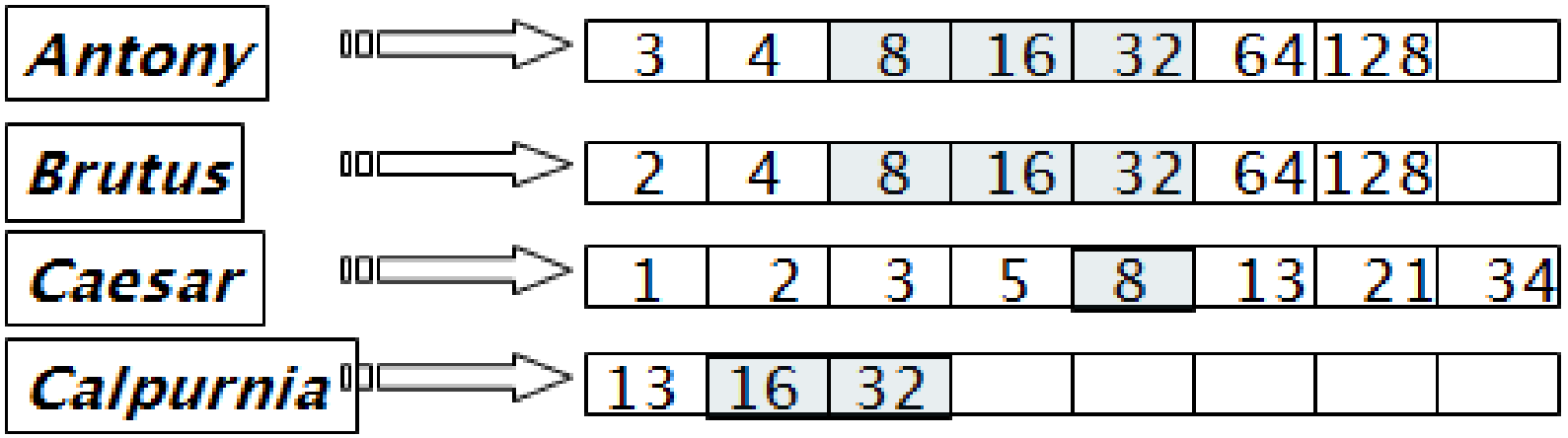
仅考虑包含高idf词项的文档

- 对于查询 catcher in the rye
- 仅考虑包含catcher和rye的文档的得分
- 直觉： 文档当中的in 和 the不会显著改变得分因此也不会改变得分顺序
- 优点：
  - 低idf词项会对应很多文档，这些文档会排除在集合A之外

仅考虑包含多个词项的文档

- Top k的文档至少包含一个查询词项
- 对于多词项查询而言，只需要计算包含其中大部分词项的文档
  - 比如，至少4中含3
  - 这相当于赋予了一种所谓软合取(soft conjunction)的语义 (早期Google使用了这种语义)
- 这种方法很容易在倒排记录表合并算法中实现

包含4个查询词项中的3个



仅对文档8、16和 32进行计算

方法二：胜者表(Champion list)

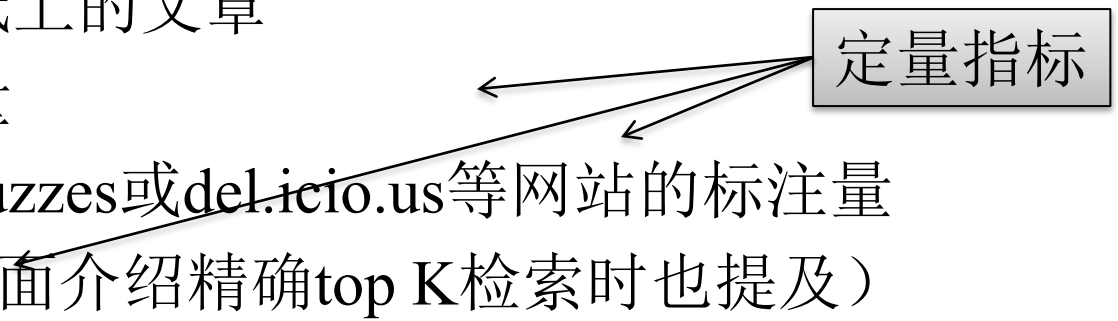
- 对每个词项t，预先计算出其倒排记录表中权重最高的r篇文档，如果采用tfidf机制，即tf最高的r篇
  - 这r篇文档称为t的胜者表
  - 也称为优胜表(fancy list)或高分文档(top docs)
- 注意：r 比如在索引建立时就已经设定
  - 因此，有可能 $r < k$
- 检索时，仅计算某些词项的胜者表中包含的文档集合的并集
  - 从这个集合中选出top k作为最终的top k

课堂思考

- 胜者表方式和前面的索引去除方式有什么关联？如何融合它们？
- 如何在一个倒排索引当中实现胜者表？
  - 提醒：胜者表与docID大小无关

方法三：静态质量得分排序方式

- 我们希望排名靠前的文档不仅相关度高(relevant)，而且权威度也大(authoritative)
- 相关度常常采用余弦相似度得分来衡量
- 而权威度往往是一个与查询无关的量，是文档本身的属性
- 权威度示例
  - Wikipedia在所有网站上的重要性
  - 某些权威报纸上的文章
  - 论文的引用量
  - 被 diggs, Y!buzzes或del.icio.us等网站的标注量
  - Pagerank（前面介绍精确top K检索时也提及）



权威度计算

- 为每篇文档赋予一个与查询无关的(query-independent) [0,1]之间的值，记为g(d)

- 同前面一样，最终文档排名基于g(d)和相关度的线性组合
  - net-score(q,d) = g(d) + cosine(q,d)
  - 可以采用等权重，也可以采用不同权重
  - 可以采用任何形式的函数，而不只是线性函数
  - 采用机器学习方法确定（6.1.2节）
- 接下来我们的目标是找net-score最高的top k文档（非精确检索）

利用g(d)排序的优点

- 这种排序下，高分文档更可能在倒排记录表遍历的前期出现
- 在时间受限的应用当中 (比如，任意搜索需要在50ms内返回结果), 上述方式可以提前结束倒排记录表的遍历

将g(d)排序和胜者表相结合

- 对每个词项维护一张胜者表，该表中放置了r篇 $g(d) + \text{tf-idf}_{t,d}$ 值最高的文档

- 检索时只对胜者表进行处理

高端表(High list)和低端表(Low list)

- 对每个词项，维护两个倒排记录表，分别称为高端表和低端表
  - 比如可以将高端表看成胜者表
- 遍历倒排记录表时，仅仅先遍历高端表
  - 如果返回结果数目超过k，那么直接选择前k篇文档返回
  - 否则，继续遍历低端表，从中补足剩下的文档数目
- 上述思路可以直接基于词项权重，不需要全局量g(d)
- 实际上，相当于将整个索引分层

方法四：影响度(Impact)排序

- 如果只想对 $tf_{t,d}$ 足够高的文档进行计算
- 那么就可以将文档按照 $tf_{t,d}$ 排序
- 需要注意的是：这种做法下，倒排记录表的排序并不是一致的
  - 排序指标和查询词项相关

- 那么如何实现非精确的top k检索？
  - 以下介绍两种做法

1. 提前结束法

- 遍历倒排记录表时，可以在如下情况之一发生时停止：
  - 遍历了固定的文档数目r
  - $wf_{t,d}$低于某个预定的阈值
- 将每个词项的结果集合合并

- 仅计算合并集合中文档的得分

2. 将词项按照idf排序

- 对于多词项组成的查询，按照idf从大到小扫描词项
- 在此过程中，会不断更新文档的得分(即本词项的贡献)，如果文档得分基本不变的话，停止

- 可以应用于余弦相似度或者其他组合得分

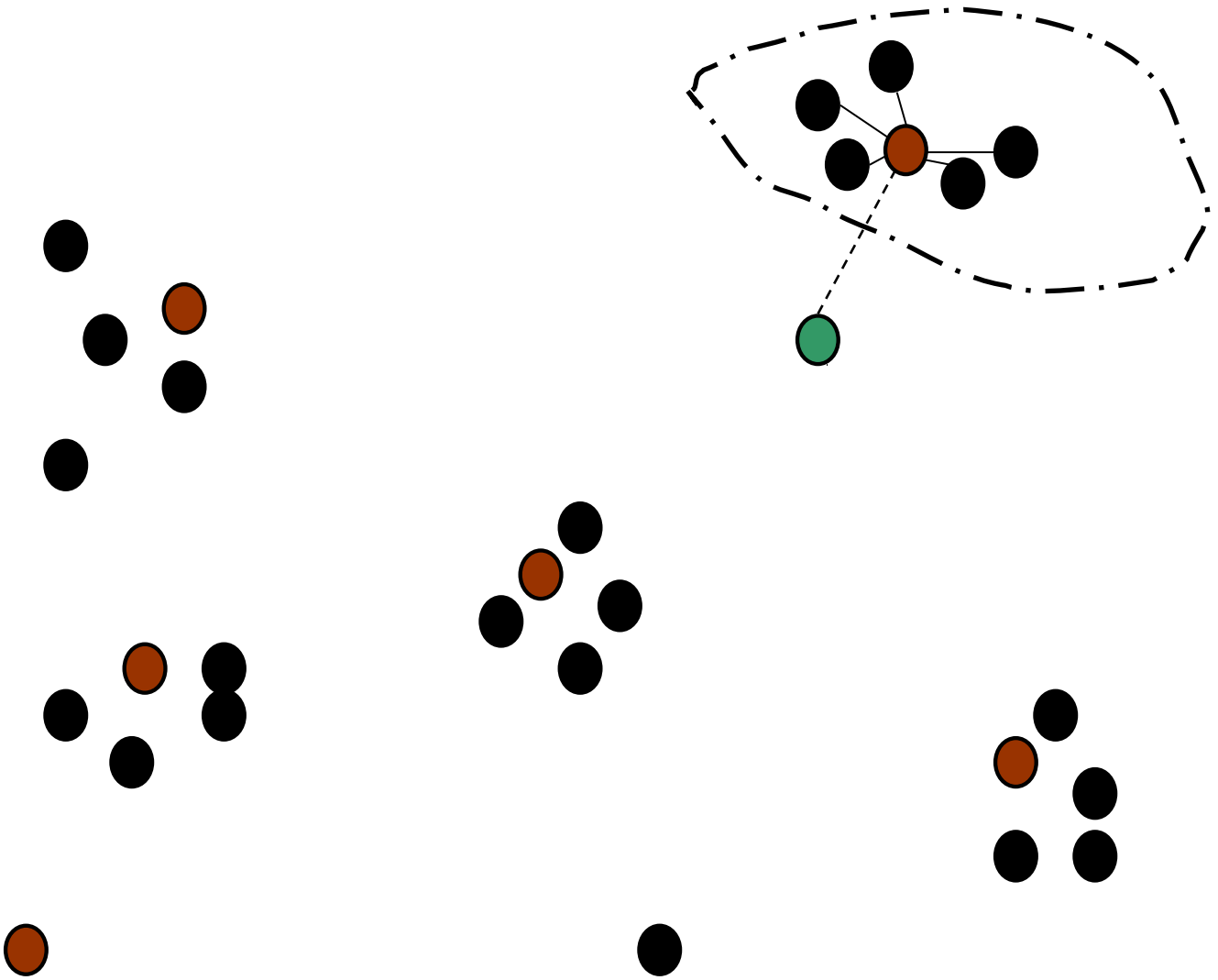
方法五： 簇剪枝(Cluster pruning)

- 随机选 √N 篇文档作为先导者
- 对于其他文档，计算和它最近的先导者
  - 这些文档依附在先导者上面，称为追随者(follower)
  - 这样一个先导者平均大约有 ~ √N 个追随者

查询处理过程

- 给定查询 Q, 找离它最近的先导者L
- 从L及其追随者集合中找到前k个与Q最接近的文档返回

可视化示意图



为什么采用随机抽样？

- 速度快
- 先导者能够反映数据的分布情况

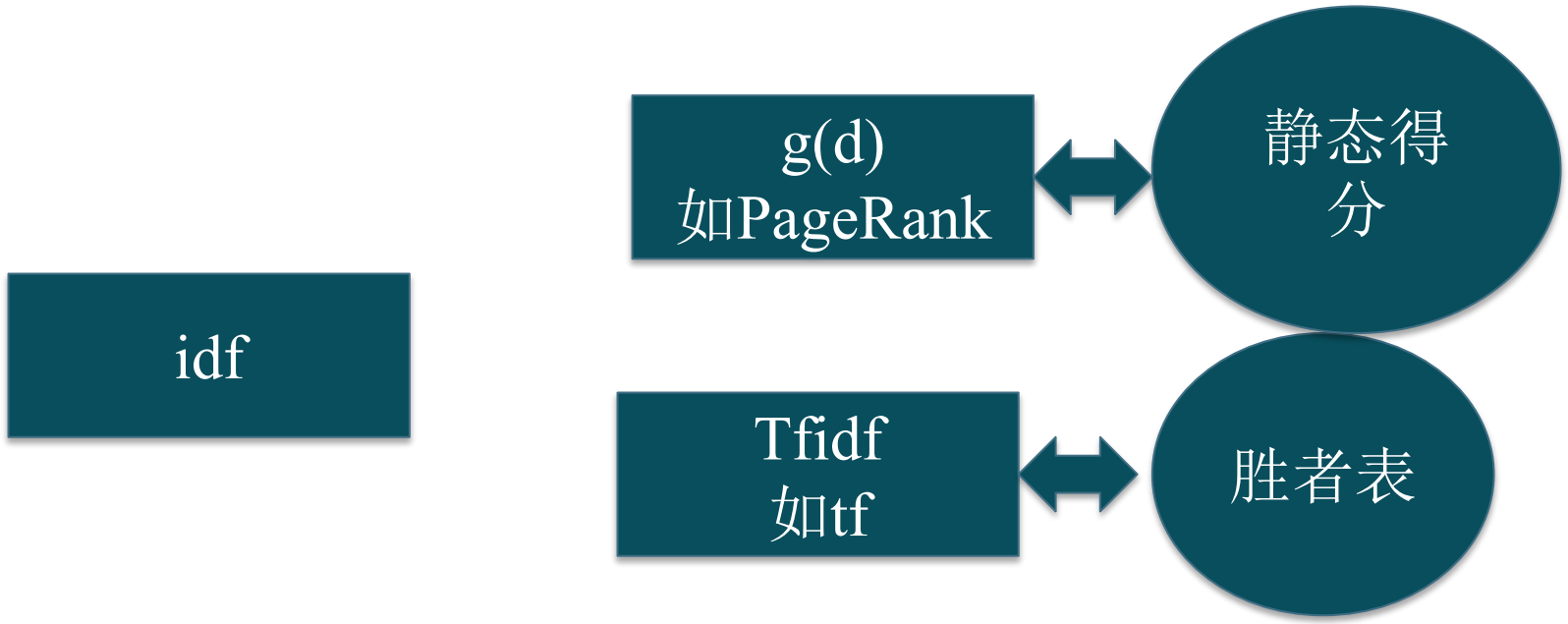
一般化变形

- 每个追随者可以附着在b1 (比如3)个最近的先导者上
- 对于查询，可以寻找最近的b2 (比如4)个先导者及其追随者

课堂练习

- 为了找到最近的先导者，需要计算多少次余弦相似度？
  - 为什么第一步中采用 √N 个先导者？
- 上一张讲义中的常数 b1, b2 会对结果有什么影响？
- 设计一个例子，上述方法可能会失败，比如返回的k篇文档中少了一篇真正的top k文档。
  - 这在随机抽样下是有可能的。

小结



倒排记录表文档排序方法

- 按照docID排序
- 按与查询无关的静态得分（如PageRank）排序
  - net-score(q, d) = g(d) + cos(q, d)
- 按影响度排序
  - 如对词频高的文档更感兴趣：按词频 $tf_{t,d}$ 排序

- 问题：
  - 对与上述不同排序方法，哪些能保证文档在倒排记录表中的排序是一致的？

课堂练习

- 1、画出如下数据集按照docID排序的倒排记录表
- 2、画出按照词项频率排序的倒排记录表

	Doc1	Doc2	Doc3
car	27	4	24
auto	3	33	0
insurance	0	33	29
best	14	0	17

如何进行评分（相似度）计算？

- 倒排记录表的排序是一致的
  - 按照docID排序
  - 按与查询无关的静态得分（如PageRank）排序
- 倒排记录表的排序是不一致的
  - 按影响度排序
- 能否参考倒排记录表的合并算法实现？

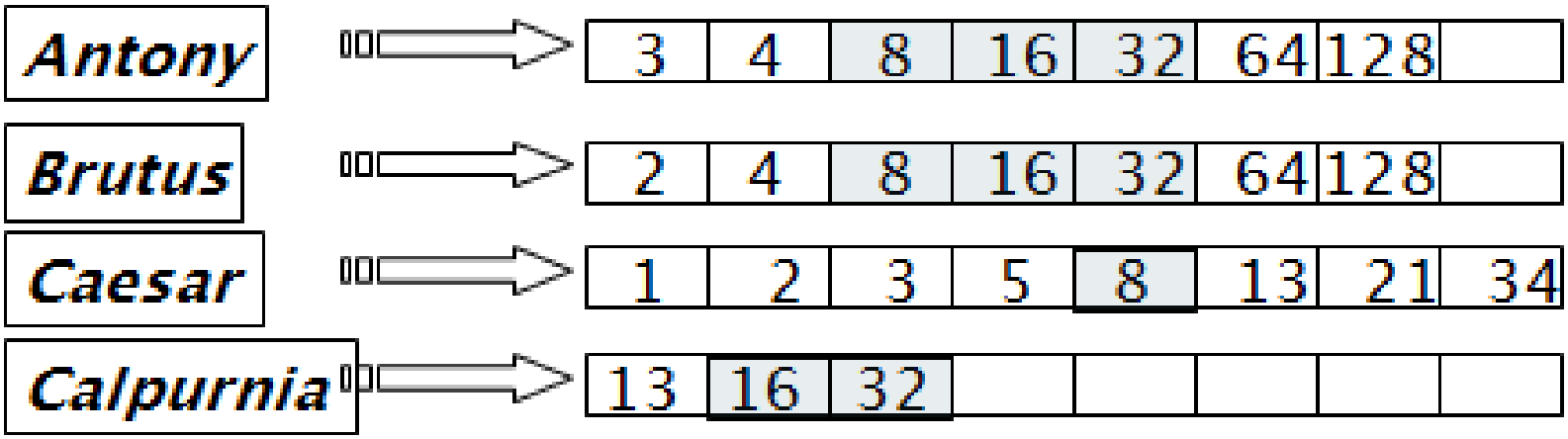
```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

基于net-score的Top k文档检索

- 倒排记录表按照g(d)从高到低排序
- 该排序对所有倒排记录表都是一致的(与docID类似，只与文档本身有关)
- 因此，可以同时遍历不同查询词项的倒排记录表，对每个查询词项的倒排记录表仅扫描1次，
  - 进行倒排记录表的遍历
  - 及评分/余弦相似度的计算
- 课堂练习：写一段伪代码来实现上述方式下的余弦相似度计算

基于net-score的Top k文档检索

- 假设有按g(d)排序的倒排表如下所示，其中g(8)=0.3, g(16)=0.2, g(32)=0.1。
- 请给出如下查询的评分计算过程及评分结果。
  - Antony Caesar
  - Antony Caesar Calpurnia



以文档为单位(Document-at-a-time)的处理

- 按照docID排序和按照PageRank排序都与词项本身无关(即两者都是文档的固有属性)，因此在全局上，这种排序都是一致的
- 上述计算余弦相似度的方法可以采用以文档为单位的处理方式
- 即在开始计算文档d_{i+1}的得分之前，先得到文档d_i的得分
- 这是“精确top k检索加速方法三：提前终止计算”的基础

- 另一种方式: 以词项为单位(term-at-a-time)的处理

以词项为单位(Term-at-a-time)的处理方式

- 最简单的情况：对第一个查询词项，对它的倒排记录表进行完整处理
- 对每个碰到的docID设立一个累加器
- 然后，对第二个查询词项的倒排记录表进行完整处理
- . . . 如此循环往复

以词项为单位(Term-at-a-time)的处理算法

CosineScore(q)

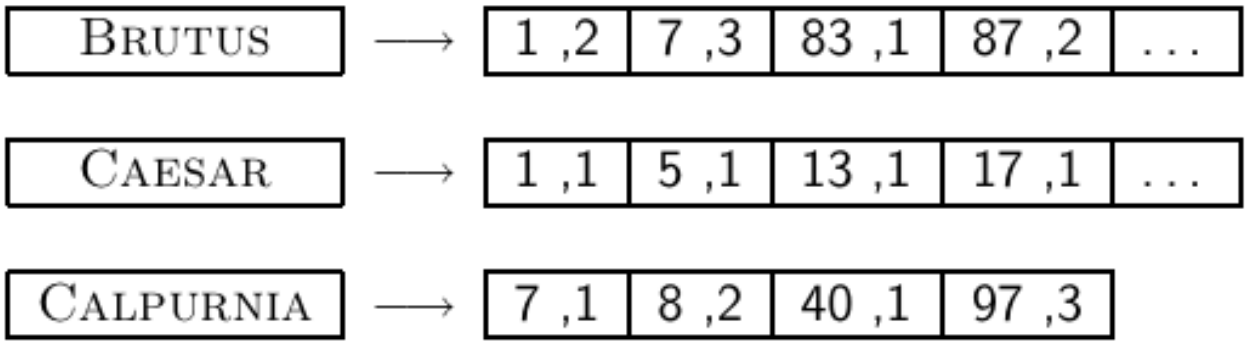
1 *float Scores*$[N] = 0$
2 *float Length*$[N]$
3 **for each** query term t
4 **do** calculate $w_{t,q}$ and fetch postings list for t
5 **for each** pair$(d, tf_{t,d})$ in postings list
6 **do** $Scores[d] += w_{t,d} \times w_{t,q}$
7 Read the array *Length*
8 **for each** d
9 **do** $Scores[d] = Scores[d]/Length[d]$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

The elements of the array "Scores" are called accumulators.

累加器

- 对于Web来说(200亿页面), 在内存中放置包含所有页面的累加器数组是不可能的
- 因此，仅对那些出现在查询词项倒排记录表中的文档建立累加器
- 这相当于，对那些得分为0的文档不设定累加器(即那些不包含任何查询词项的文档)

累加器举例



- 查询: [Brutus Caesar]:
- 仅为文档 1, 5, 7, 13, 17, 83, 87设立累加器
- 不为文档 8, 40, 97 设立累加器

强制执行合取查询

- 可以强制执行一个合取查询 (类似Google)
  - 仅返回全部关键词均出现的文档
  - 上例中，对于[Brutus Caesar]查询，仅需一个累加器
  - 因为仅有 *文档1* 同时包含这两个词

结果排序实现小结

- 实际应用中，VSM评分计算代价非常大，需要对文档集中的所有文档计算相似度
- 在多数应用中，大部分文档与给定查询的相似度为0 → 具有很多加速的空间
- 然而，即使在这种场景下，也没有快速的最近邻算法能保证返回结果完全正确
- 实践中，使用启发式方法来缩小搜索空间，通常能取得很好的效果

提纲

① 上一讲回顾

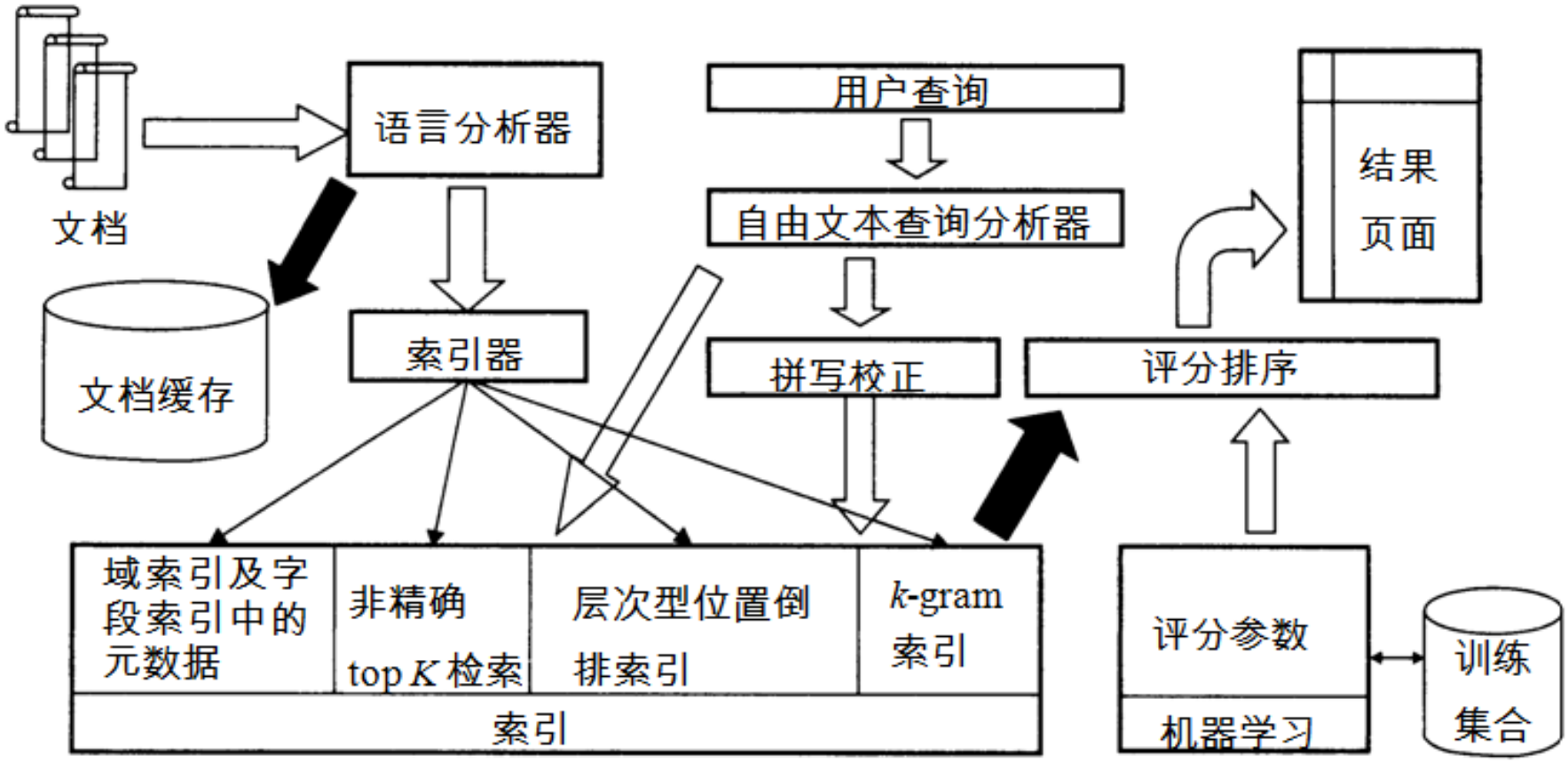
② 结果排序的动机

③ 再论余弦相似度

④ 结果排序的实现

⑤ **完整的搜索系统**

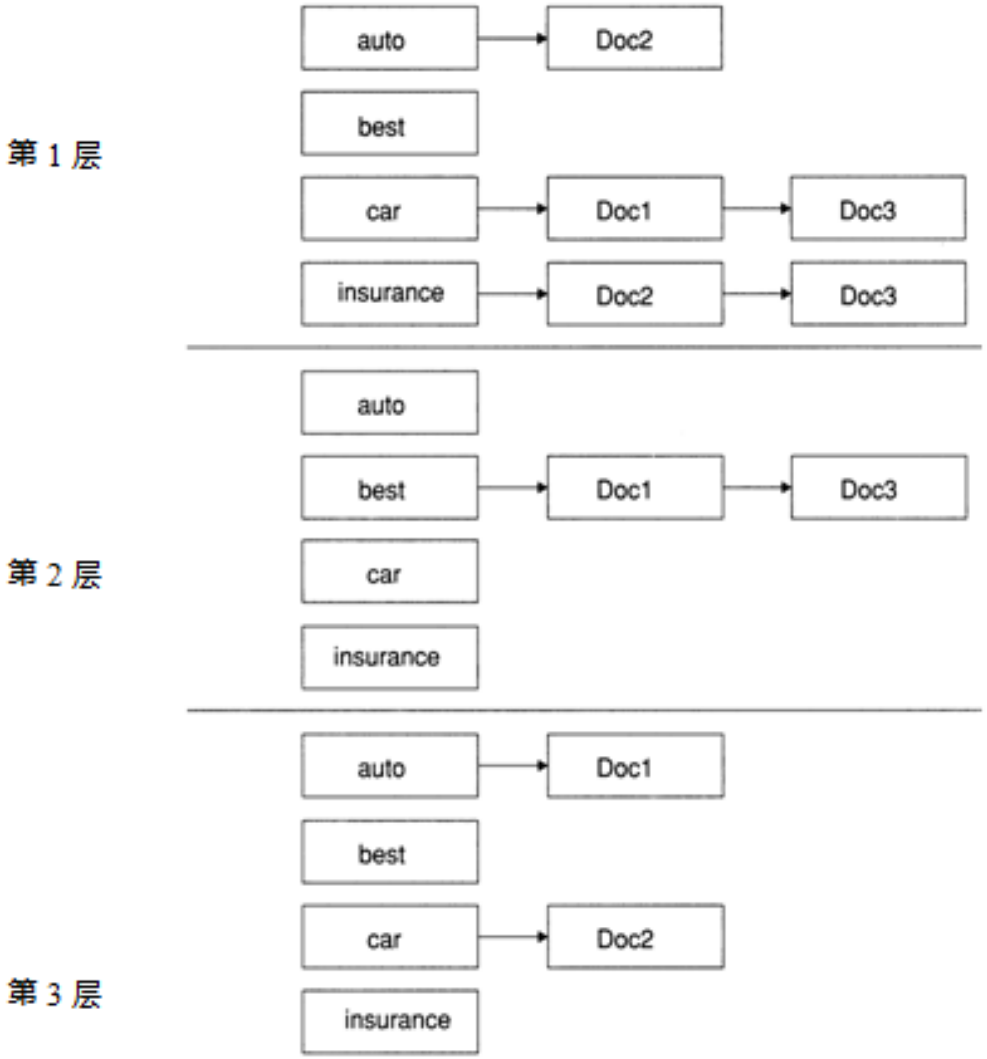
完整的搜索系统示意图


文档
语言分析器
用户查询
自由文本查询分析器
结果
页面
文档缓存
索引器
拼写校正
评分排序
域索引及字
段索引中的
元数据
非精确
top K 检索
层次型位置倒
排索引
k-gram
索引
索引
评分参数
机器学习
训练
集合

多层次索引

- 基本思路:
  - 建立多层索引，每层对应索引词项的重要性
  - 查询处理过程中，从最高层索引开始
  - 如果最高层索引已经返回至少k (比如, $k = 100$)个结果，那么停止处理并将结果返回给用户
  - 如果结果 $< k$ 篇文档，那么从下一层继续处理，直至索引用完或者返回至少k 个结果为止
- 例子：
  - 第1层：所有标题的索引
  - 第2层：文档剩余部分的索引
  - 标题中包含查询词的页面相对于正文包含查询词的页面而言，排名更应该靠前
  - 第1层：所有tf>20的索引
  - 第2层：所有tf>10的索引
  - 第3层：所有tf<10的索引

多层次索引的例子


第 1 层
auto
Doc2
best
car
Doc1
Doc3
insurance
Doc2
Doc3
第 2 层
auto
best
Doc1
Doc3
car
insurance
第 3 层
auto
Doc1
best
car
Doc2
insurance

多层次索引

- 大家相信，Google (2000/01)搜索质量显著高于其他竞争者的一个主要原因是使用了多层次索引
- (当然还有PageRank、锚文本以及邻近限制条件的使用)

搜索系统组成部分(已介绍)

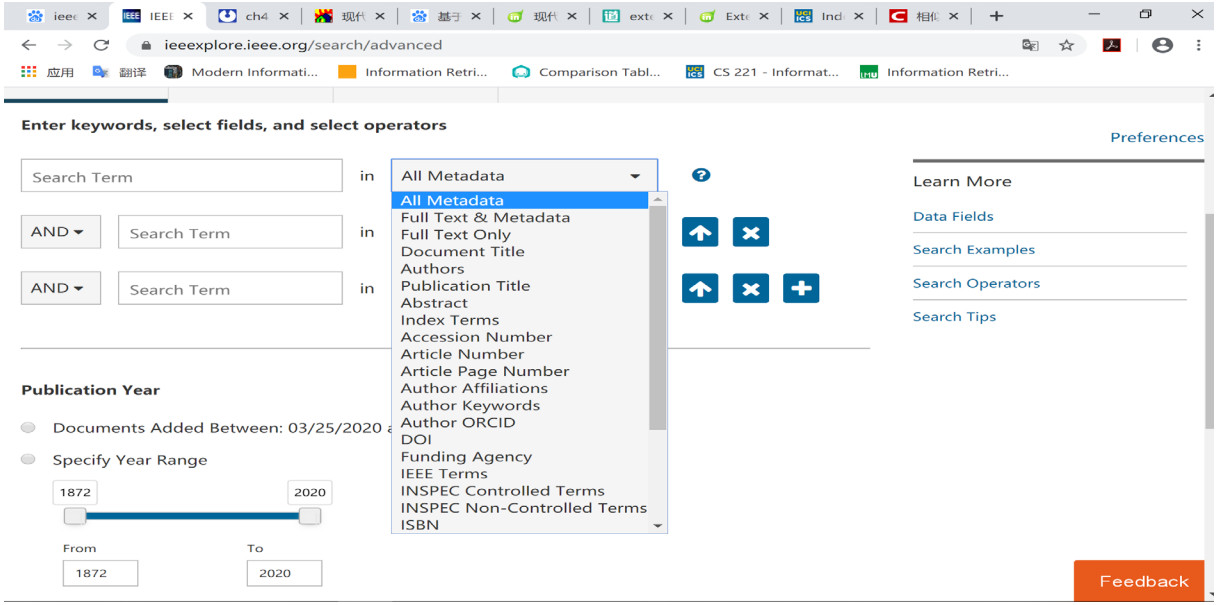
- 文档预处理 (语言及其他处理)
- 位置信息索引
- 多层次索引
- 拼写校正
- *k*-gram索引(针对通配查询和拼写校正)
- 查询处理
- 文档评分
- 以词项为单位的处理方式

搜索系统组成部分(未介绍)

- **域索引**：按照不同的域进行索引，如文档正文，文档中所有高亮的文本，锚文本、元数据字段中的文本等等
- **邻近式排序** (如, 查询词项彼此靠近的文档的得分应该高于查询词项距离较远的文档)
- 查询分析器
- 文档缓存(cache): 用它来生成文档摘要(snippet)
  - 第8.7节
- 基于机器学习的排序函数
  - Learn to rank：第15.4节

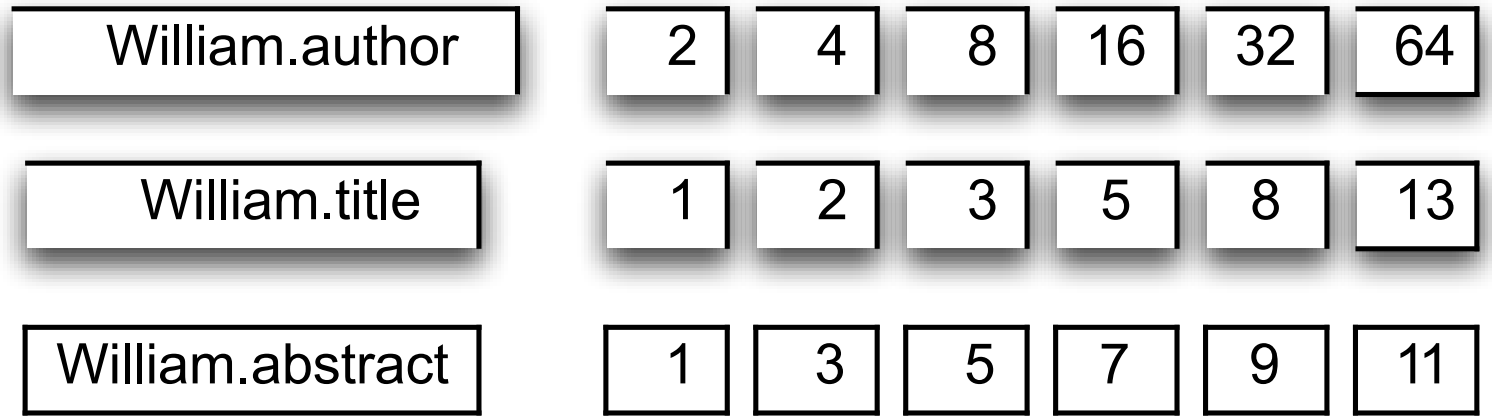
域索引

- 与数据库的字段类似，域是文档中可标识的区域
  - e.g., title, abstract, bibliography
  - <title>Romeo and Juliet</title>
- 域的内容可以是结构化数据，也可以是自由文本



域索引

- 对域构建倒排索引
  - (term, docID) → (term, field, docID)
- 一种倒排索引的形式为:



- 课程所介绍的压缩和排序方法适用于域索引

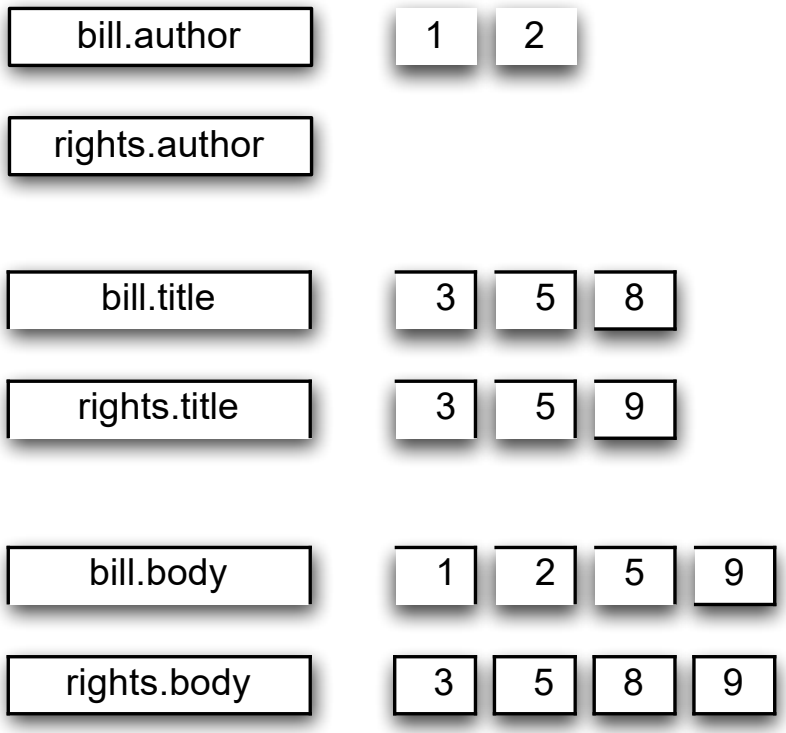
域索引的应用

- 布尔检索
  - (instant in TITLE) AND (oatmeal in BODY)
- 作为相关性检索的过滤条件
  - 检索包含“UCI”的pdf文档
- 排序式布尔检索
  - 对不同域赋予不同的权重

$$Score = 0.6(T \in TITLE) + 0.3(T \in BODY) + 0.1(T \in ABSTRACT)$$

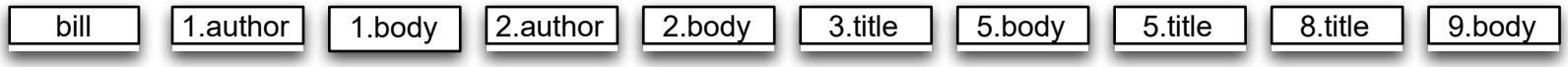
排序式布尔检索举例

- 基于权重比例计算文档的评分
  - (0.1 author), (0.3 body), (0.6 title)
- 查询为：
  - “bill” AND “rights”
  - “bill” OR “rights”
- 方法：
  - 遍历各个域的倒排表
  - 对不同域的评分进行累加



排序式布尔检索举例

- (0.1 author), (0.3 body), (0.6 title)
- Q1: “bill”

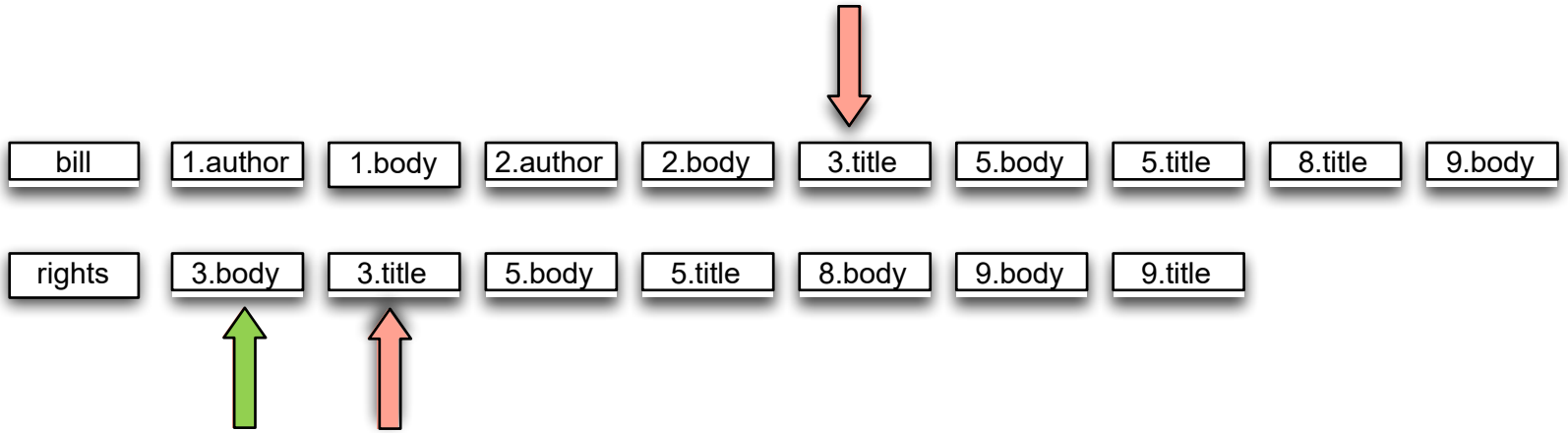


1: 0.1+0.3=0.4
2:0.4
3:0.6

5:0.9
8:0.6
9:0.3

排序式布尔检索举例

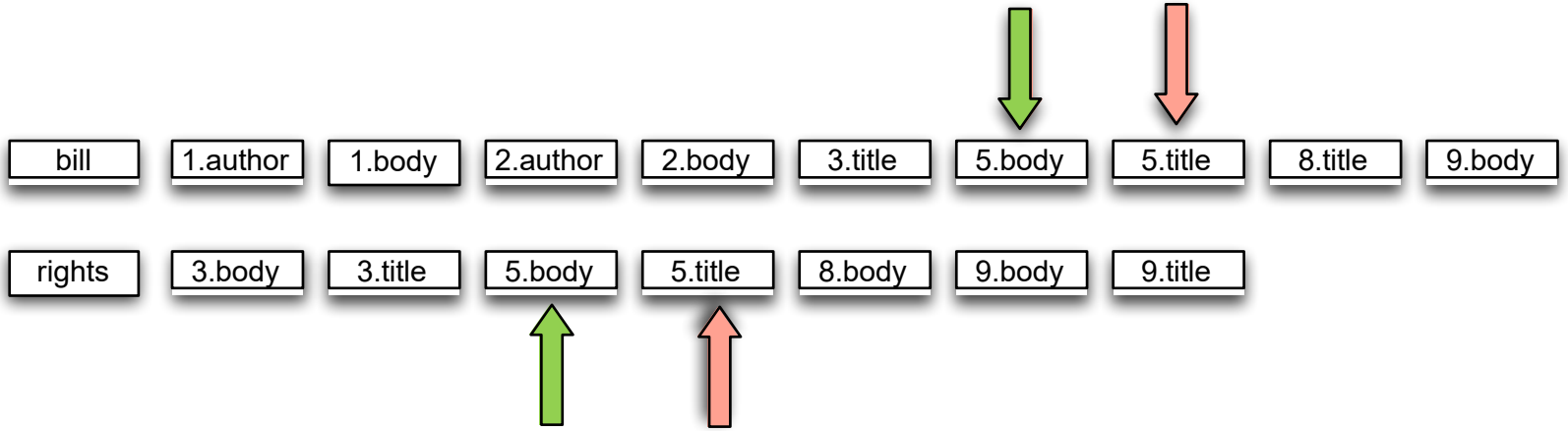
- (0.1 author), (0.3 body), (0.6 title)
- Q2: “bill” AND “rights”【出现在同一个域中】



3: 0.6

排序式布尔检索举例

- (0.1 author), (0.3 body), (0.6 title)
- Q2: “bill” AND “rights”

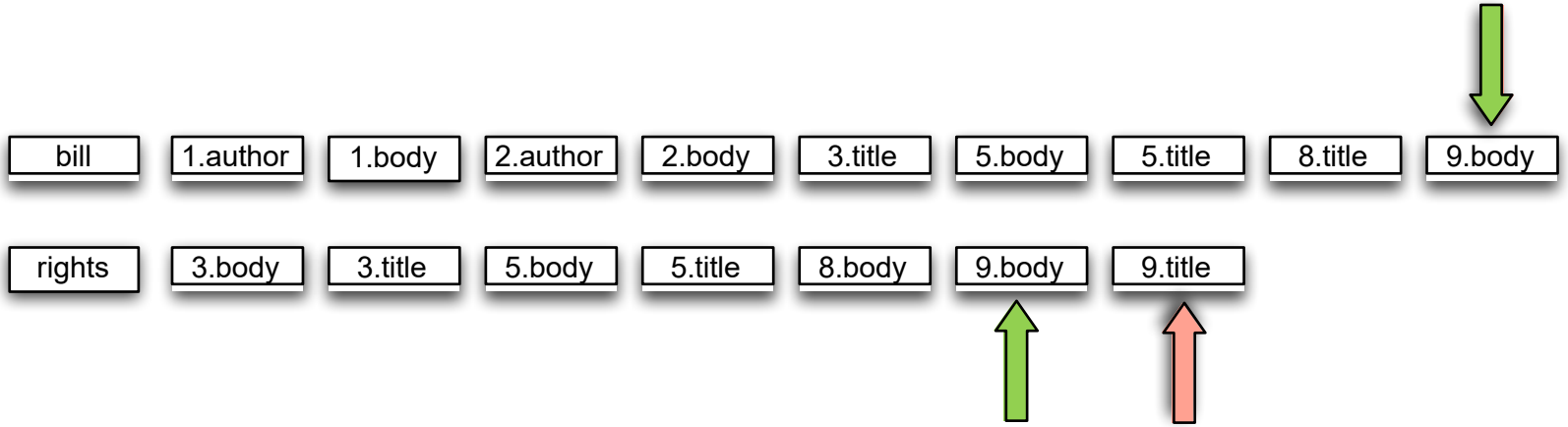


3: 0.6

5: 0.3+0.6=0.9

排序式布尔检索举例

- (0.1 author), (0.3 body), (0.6 title)
- Q2: “bill” AND “rights”



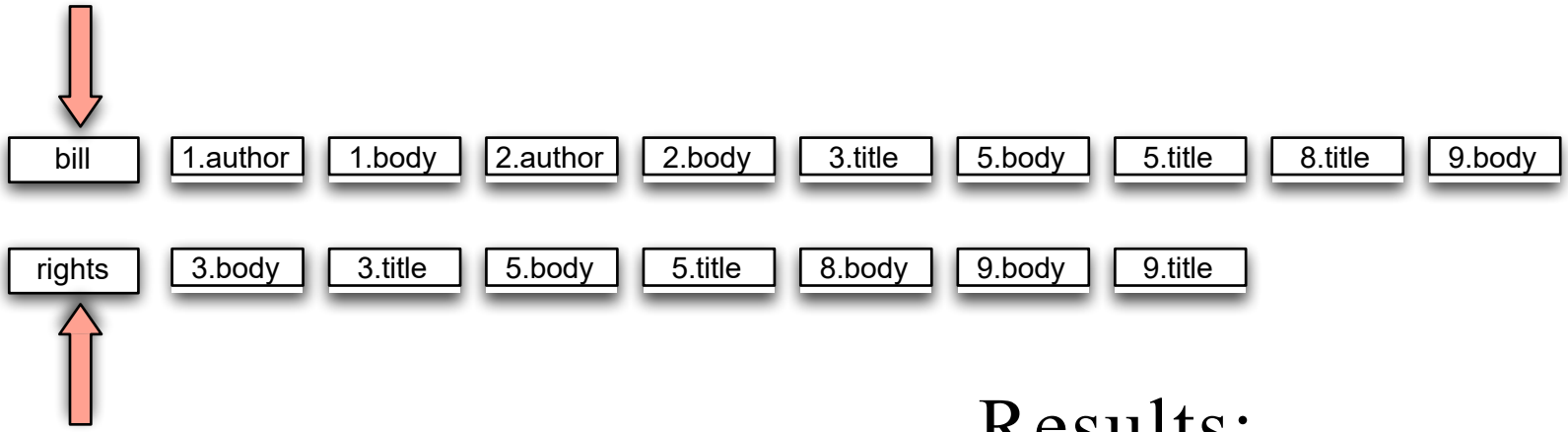
3: 0.6

5: 0.3+0.6=0.9

9: 0.3

排序式检索举例

- (0.1 author), (0.3 body), (0.6 title)
- Q3: “bill” OR “rights”



1: 0.4 5: 0.9
2: 0.4 8: 0.9
3: 0.9 9: 0.9

Results:
9,8,5,3,2,1

排序式检索

- 权重如何确定？ (上例中的0.6,0.3,0.1)
- 很少由用户指定，而是内置于搜索引擎中
- 搜索引擎如何确定这些权重？
  - 手工编码：专业经验
  - 机器学习（6.1.2节,15.4节）

临近式排序

- 查询：t1, t2, …, tk
- 定义文档d中包含所有查询词项的最小窗口大小为ω
  - D: The Quality of mercy is not strained
  - Q: strained mercy
- 基于位置索引计算ω
  - ω越小，匹配程度越高
  - 文档中不包含所有查询词项，则设置较高的ω
- 设计一个基于ω的加权评分函数
  - 手工编码
  - 机器学习
- 典型应用：Google“软合取”

向量空间检索与其他方法的融合

- 如何将短语检索和向量空间检索融合在一起？
  - 我们不想对每个短语都计算其idf值，为什么？
- 如何将布尔检索和向量空间检索融合在一起？
  - 例如: “+”-限制条件和“-”-限制条件
- 后过滤很简单，但是效率低下---没有好的解决办法
- 如何将通配符查询融入到向量空间检索中？同样，没有好的解决方法

短语检索和向量空间检索

- Biword索引
  - 将短语视为一个独立词项
- 给定短语查询，并不能预先确定该查询是否建立二元索引
- Q: “rising interest rates”
- 查询组合
  - 首先将3个词看成一个词项，执行查询
  - 如果返回文档数目少，将查询分解为两个短语查询
    - “rising interest” “interest rates”
  - 如果返回文档数目仍较少，则将三个词作为三个独立的查询词项

- 基于位置索引（参考Lucene）
  - content: “rising interest rates”
  - content: “rising interest rates”~3【编辑距离】

布尔检索与向量空间检索

- 扩展布尔模型
  - p-norm模型
- 排序式布尔检索

- 布尔查询作为向量空间检索的筛选条件
  - +site:wikipedia.org
  - -site:wikipedia.org
  - +filetype:pdf
  - -filetype:pdf

通配符检索与向量空间检索

- 考虑查询: “rom* restaurant”？
- 是否可以将通配符扩充为字典中匹配的所有词项？
- 问题：
  - 需要考虑词项的tf、idf
  - 文档rome roma rommer的得分将高于rome restaurant
- 因此，没有好的办法。

本讲内容

- 排序的重要性：从用户的角度来看(Google的用户研究结果)
- 另一种长度归一化: 回转(Pivoted)长度归一化
- 排序实现
- 完整的搜索系统

参考资料

- 《信息检索导论》第6、7 章
- How Google tweaks its ranking function
- Interview with Google search guru Udi Manber
- Yahoo Search BOSS: Opens up the search engine to developers. For example, you can rerank search results.
- Compare Google and Yahoo ranking for a query
- How Google uses eye tracking for improving search

课后练习

- 有待补充