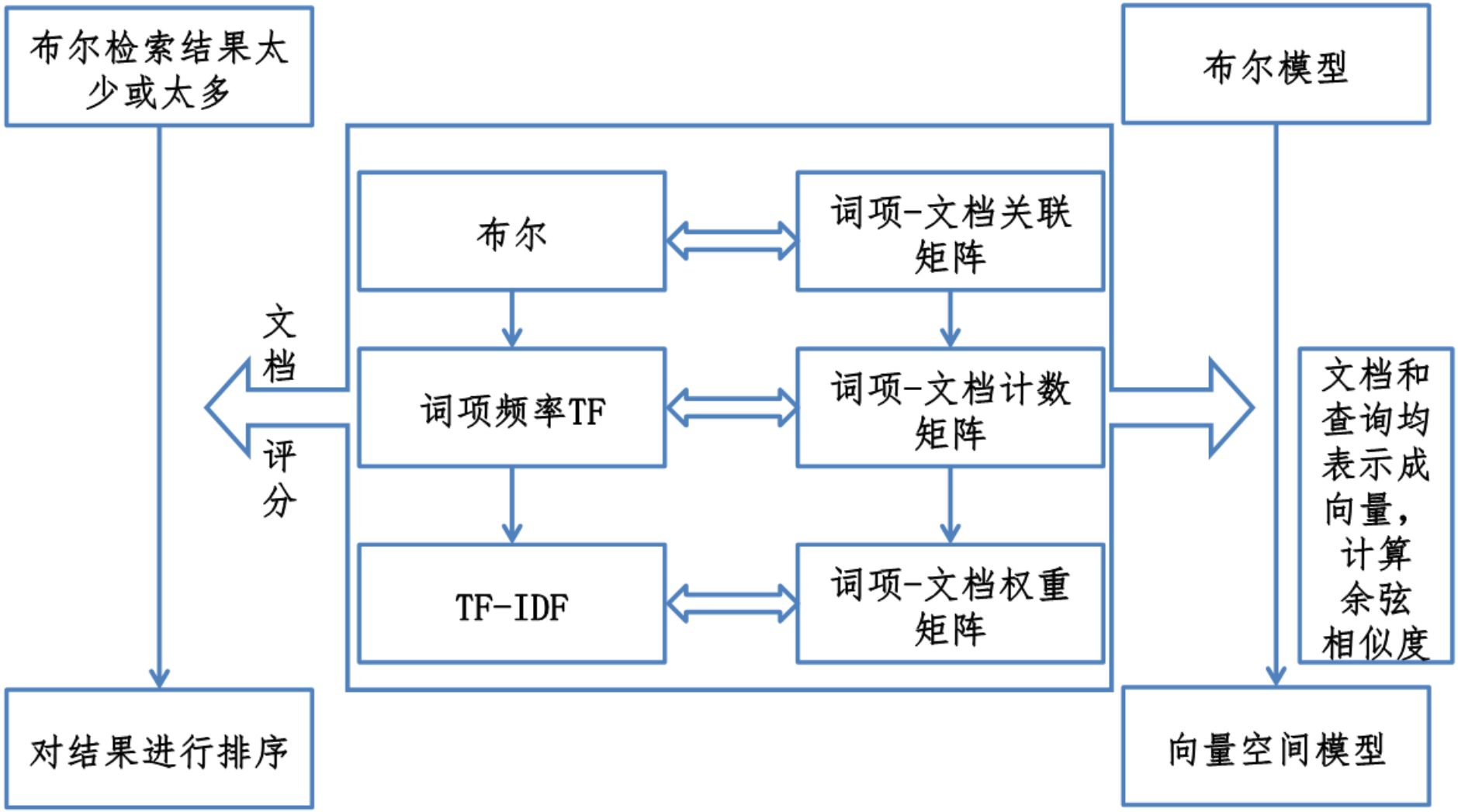
向量模型及检索系统

向量模型

本讲结构图


布尔检索结果太少或太多
布尔模型
布尔
词项-文档关联矩阵
文档
词项频率TF
词项-文档计数矩阵
评分
文档和查询均表示成向量，计算余弦相似度
TF-IDF
词项-文档权重矩阵
对结果进行排序
向量空间模型

目录

- 排序式检索
- 词项频率
- tf-idf权重计算
- 向量空间模型

排序式检索

- 迄今为止，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
  - 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

- 希望根据文档对查询者的有用性大小顺序排序
- **如何根据一个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**
- 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A,B重叠度的方法
  - Jaccard(A,B) = |A ∩ B| / |A ∪ B|
  - Jaccard(A,A) = 1
  - Jaccard(A,B) = 0 if A ∩ B = 0
- 集合A和B不需要具有同样的规模
- Jaccard(A,B)的取值在[0,1]

– Jaccard(A,B) = |A ∩ B| / |A ∪ B|

举例

– Query: ides of march

– Document 1: caesar died in march

– Document 2: the long march

– Jaccard(q, doc_1) = 1/6

– Jaccard(q, doc_2) = 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

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

目录

- 排序式检索
- 词项频率
- tf-idf权重计算
- 向量空间模型

词项-文档二值关联矩阵

	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
. . .						

- 每个文档用一个二值向量表示$\in \{0,1\}^{|V|}$

词项-文档词频关联矩阵

- 考虑词项在文档中出现的频率
  - 将每个文档看成是一个词频向量：矩阵中的一列

	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
. . .						

词袋模型(Bag of words)

- 词袋模型：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
- 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
- 在后面的课程中将可以看到如何“恢复”位置信息
- 现在只考虑：词袋模型

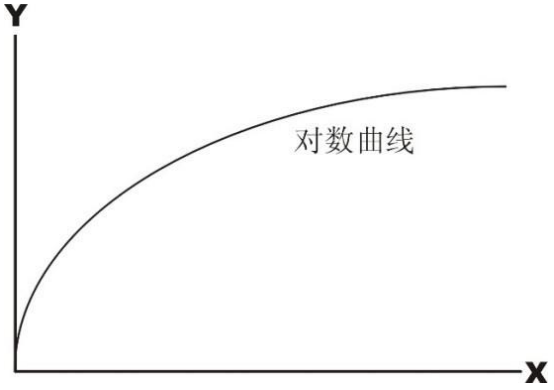
词项频率tf (Term frequency)

- 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*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
- 相关性不会正比于词项频率

一种替代原始tf的方法: 对数词频

- 词项t在文档d中的对数频率权重

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



- $tf_{t,d} \rightarrow w_{t,d}$: 0→0, 1→1, 2→1.3, 10→2, 1000 → 4
- 文档-词项的匹配得分是所有同时出现在q和文档d中的词项的词频的对数之和

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

- 评分为0，表示文档和query中没有公共词项

目录

- 排序式检索
- 词项频率
- tf-idf权重计算
- 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 vs 常见词项

- 罕见词项比常见词所蕴含的信息更多
- 罕见词项
  - 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，希望像ARACHNOCENTRIC一样的罕见词项将有较高权重
- 常见词
  - 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)

- 对罕见词项赋予高权重
- 对常见词项赋予低权重
- 使用**文档频率df**这个因子来实现上述目标
- **文档频率**：出现词项的文档数目

idf (inverse document frequency)逆文档频率

- 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 <= 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

idf的计算举例 N=1,000,000

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

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

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*在整个文档集合中出现的词的次数;
- 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 文档-逆文档频率(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 ×idf
- 词项*t*的tf-idf 由它的tf和idf组合而成

$$w_{t,d}=(1+\log tf_{t,d}) \times \log_{10}(N/df_t)$$

- tf-idf值随着词项在单个文档中出现次数(tf)增加而增大
- tf-idf值随着词项在文档集中数目(df)增加而减小

Query的最终文档排序(Query词)

$$Score(q,d) = \sum_{t \in q \bigcap d} tf * idf_{t,d}$$

目录

- 排序式检索
- 词项频率
- tf-idf权重计算
- 向量空间模型

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
. . .						

- 每个文档用一个二值向量表示$\in \{0,1\}^{|V|}$

词频矩阵

	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
. . .						

- 每篇文档表示成一个词频向量 $\in \mathbb{N}^{|V|}$

二值 → 词频 → tf-idf矩阵

	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 $\in \mathbf{R}^{|V|}$

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 $\vec{d} \in R^{|V|}$ (V是词项集合，$|V|$表示词项个数)
- 于是，有一个$|V|$维实向量空间
  - 空间的每一维都对应一个词项
  - 文档是空间中的点或者向量
  - 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：对每个文档向量来说大部分都是0

Queries表示成向量

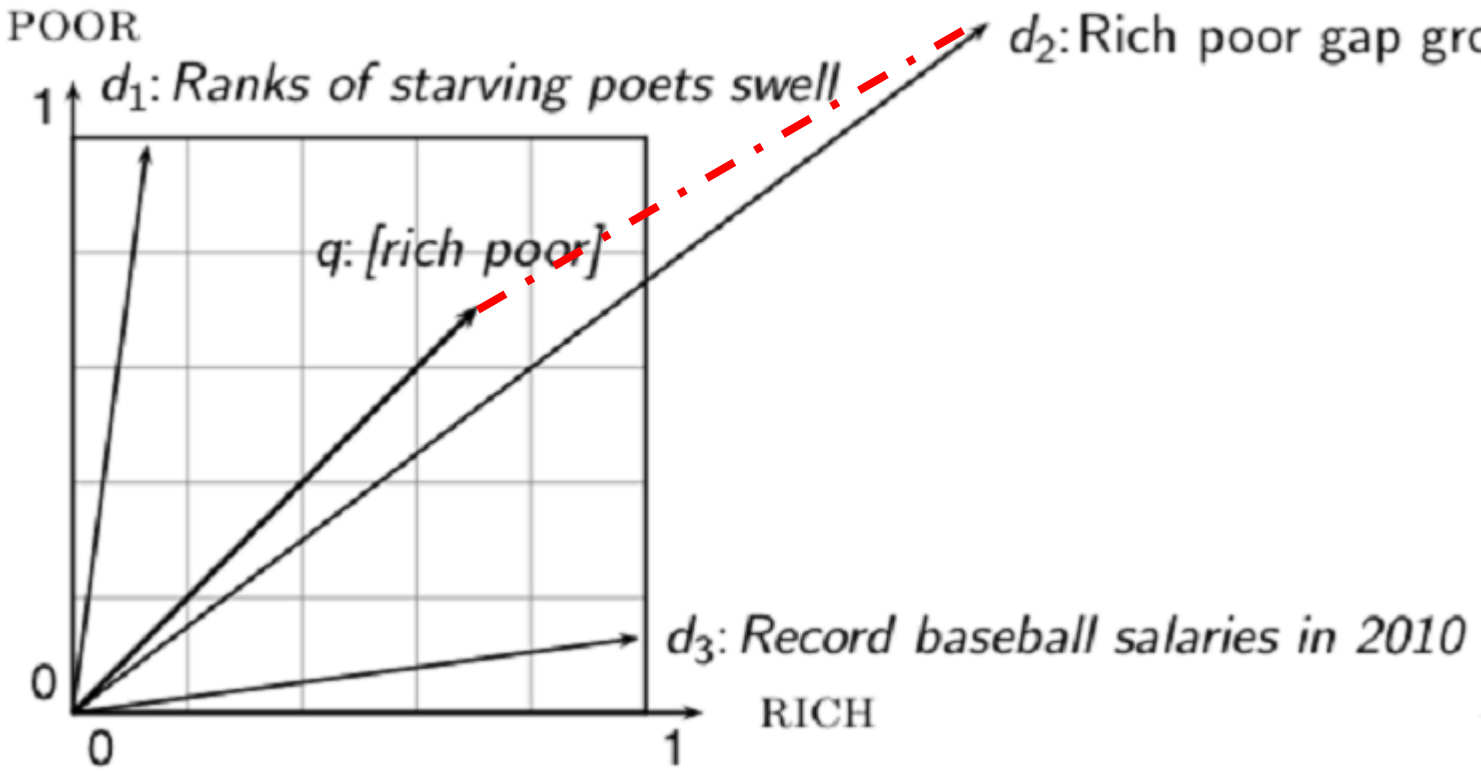
- 关键思路1
  - 对于查询做同样的处理，即将查询表示成同一高维空间的向量$\vec{q}$
- 关键思路2
  - 在向量空间内根据query与文档的**向量**间的距离来排序

向量空间下相似度的形式化定义

- 第一步：计算两点之间距离
  - 两个向量终点间距离
- 欧氏距离(Euclidean Distance)？
  - 不是一种好的选择
  - 因为，欧氏距离是欧氏空间中两点间的距离公式，对向量长度很敏感
  - |V|维空间

$$D_{Euc}(\vec{q},\vec{d}) = \sqrt{\sum_{i=1}^{|V|}(q_i - d_i)^2}$$

为什么欧氏距离不好？



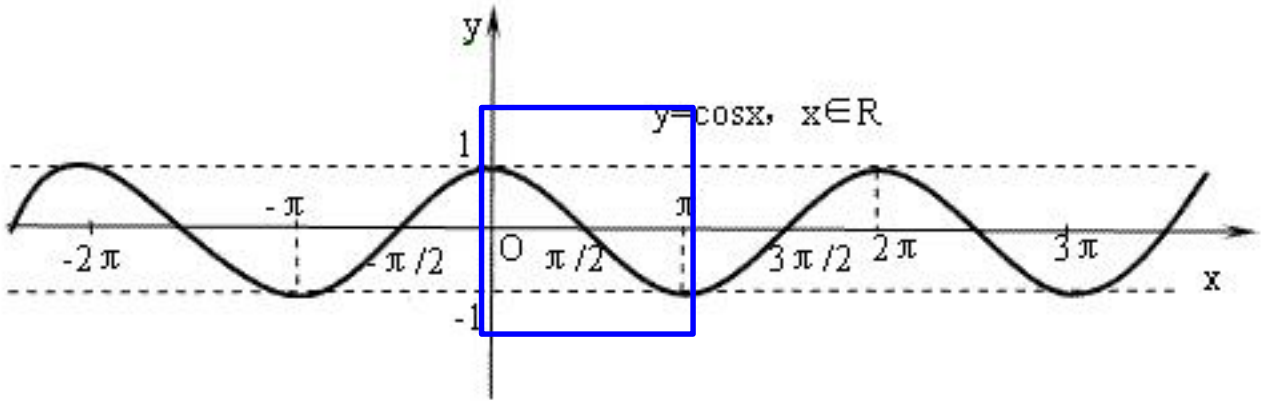
- 在欧氏空间，q与文档d_2的欧氏距离很大，但q与d_2的分布很相近

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1
  - 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Cosine相似度

- 下面两个观点是等价的
  - 按query与文档夹角递增给文档排序
  - 按余弦cosine(query,document)递减给文档排序
- 这是因为在[0°，180°]区间上，cosine是单调递减函数
- 只考虑相对顺序



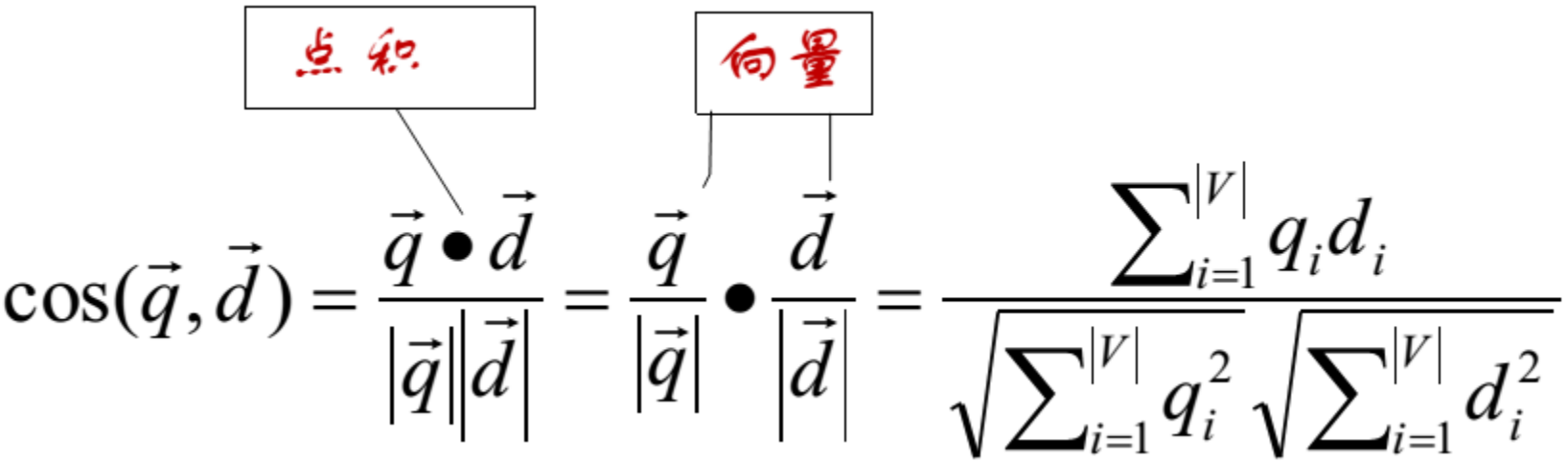
文档长度归一化

- 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
- 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)



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

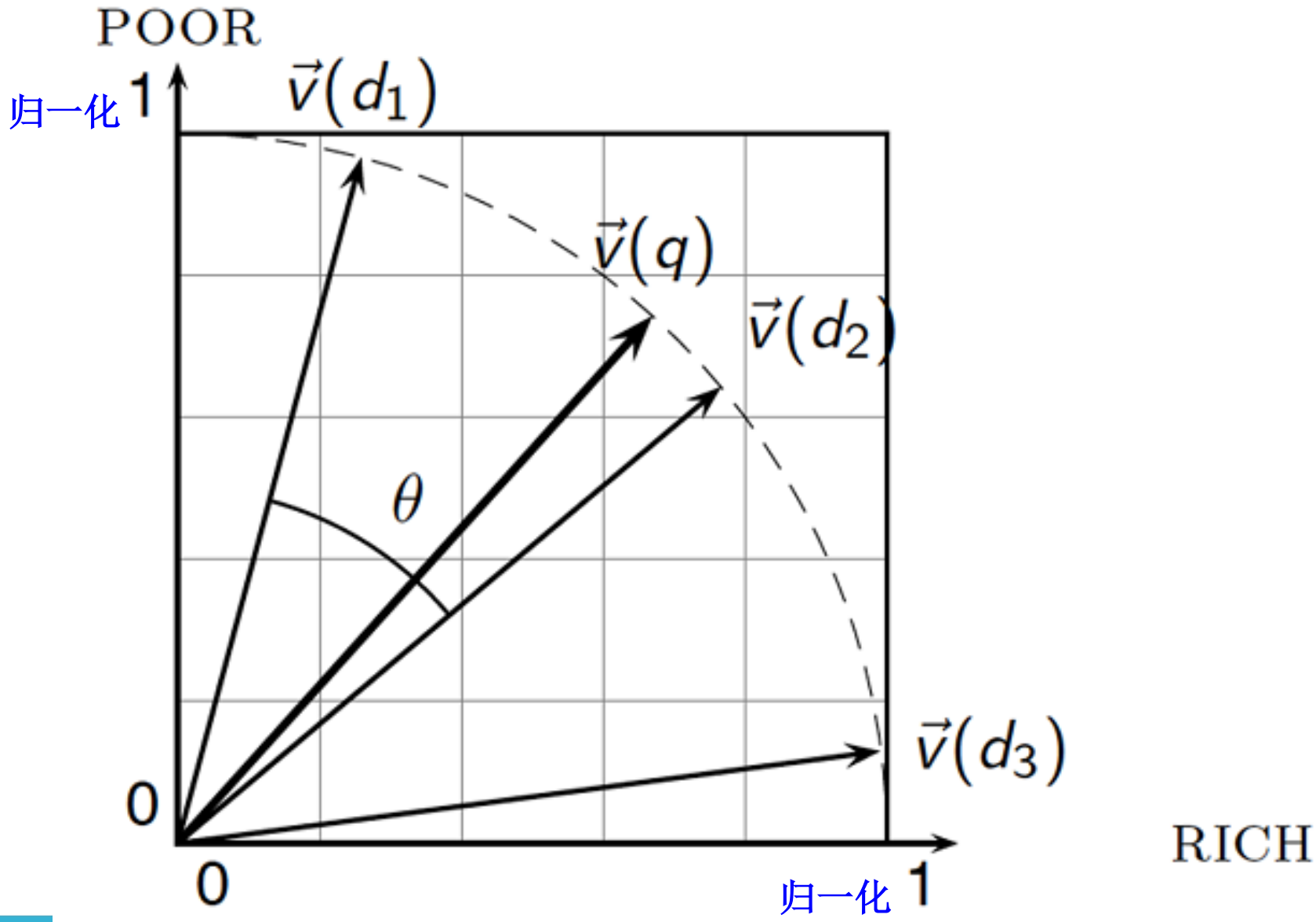
向量长度归一化后的余弦计算

- 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)

词项	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

余弦相似度计算

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

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

为了简化计算，上述计算过程中没有引入idf

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

$$0.789 = \frac{3.06}{\sqrt{3.06^2 + 2^2 + 1.3^2 + 0^2}}$$

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

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

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

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

计算cosine

- N文档总数
- Length[N] 数组中存放的是每个文档向量的长度(即归一化因子)
- Scores[N] 数组放的是每篇文档的得分。

```
1   float Scores[N] = 0
2   Initialize Length[N]
3   for each query term t                遍历查询的所有词项
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}   分数累加
7   Read the array Length[d]
8   for each d
9   do Scores[d] = Scores[d]/Length[d]   归一化
10  return Top K components of Scores[]
```

tf-idf 权重机制的变形

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(consine)**	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_m^2}}$
a(auamented)	$0.5+\frac{0.5\times tf_{t,d}}{\max_t(tf_{t,d})}$	**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)**	$1/CharLength^a$, $a<1$
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。

检索系统

目录

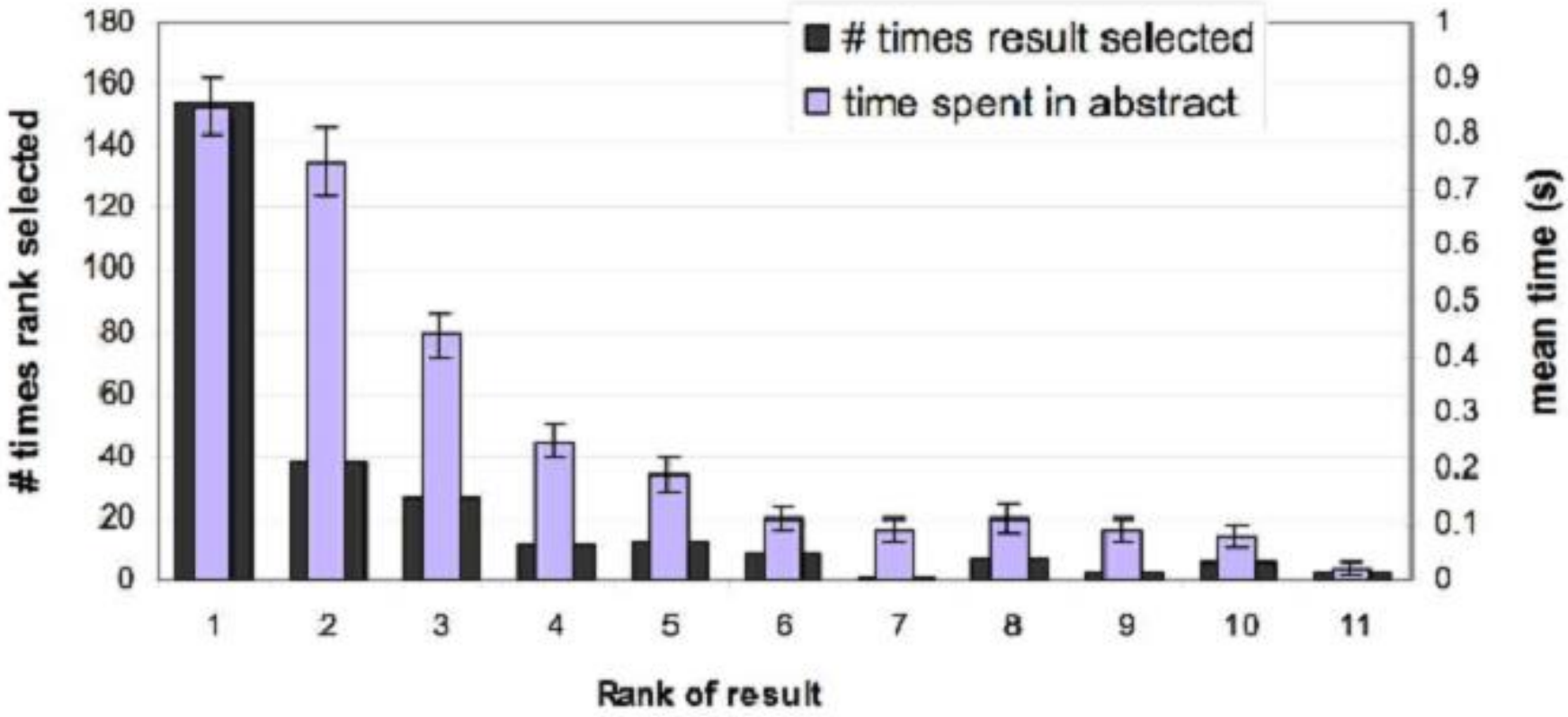
- 结果排序的重要性
- 结果排序的实现
- 完整的搜索系统

排序的重要性

- 不排序问题的严重性
  - 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
- 实际上，大部分用户只看1到3条结果

浏览 vs. 点击

Looking vs. Clicking

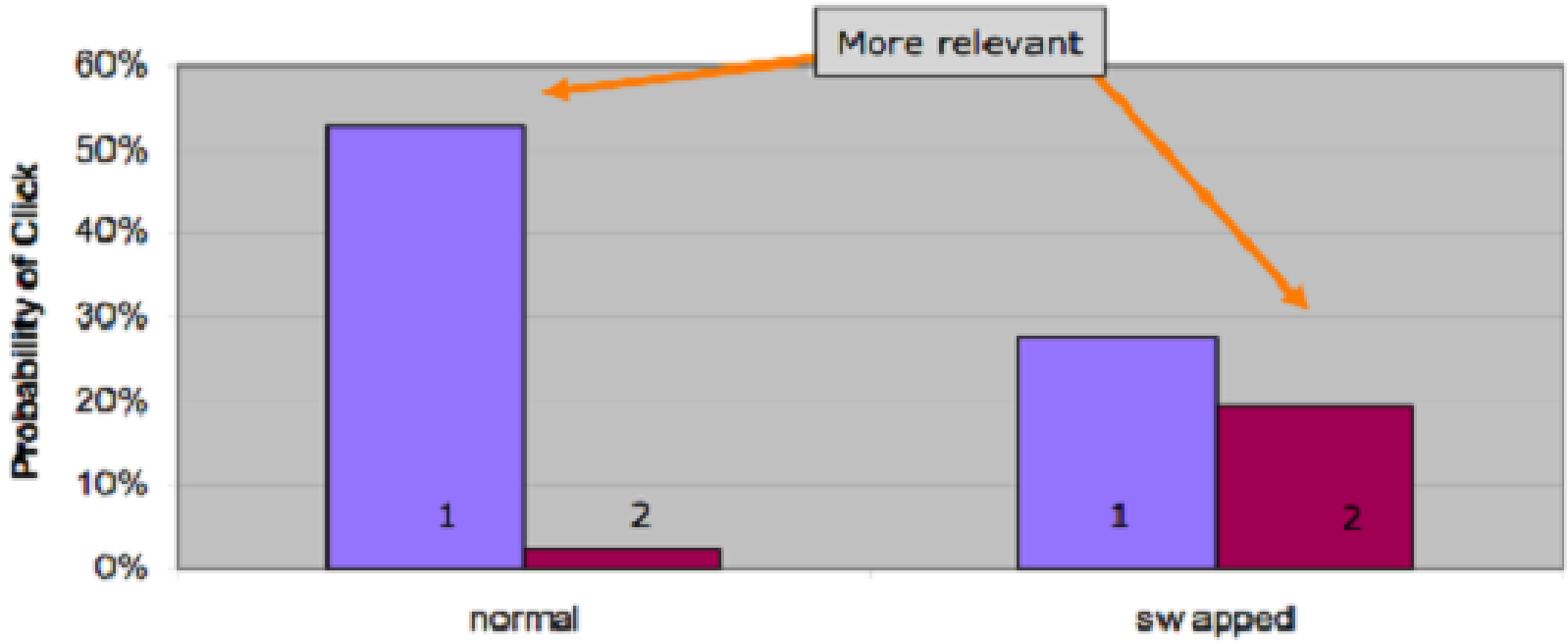


- Users view results one and two more often / thoroughly
- Users click most frequently on result one

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



排序的重要性: 小结

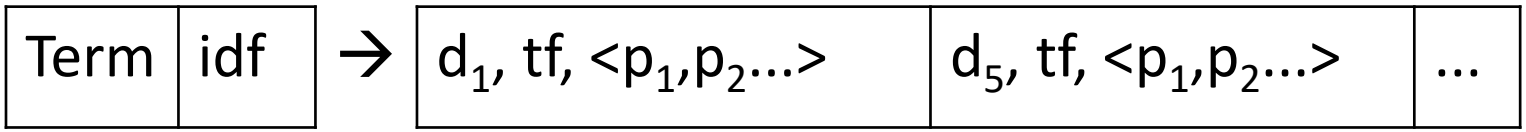
- 摘要阅读(Viewing abstracts): 用户更可能阅读前几页(1, 2, 3, 4)的结果的摘要
- 点击(Clicking): 点击的分布甚至更有偏向性
  - 一半情况下，用户点击排名最高的页面
  - 即使排名最高的页面不相关，仍然有30%的用户会点击它
- → 正确排序相当重要
- → 把最相关的页面放在首页非常重要

目录

- 结果排序的重要性
- 结果排序的实现
- 完整的搜索系统

词项频率tf和逆文档频率idf的存储

- 词典中保存每个词的idf值
- tf存入倒排索引中
- position



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

- 查询的多个词项无权重
  - 相当于假设查询的每个词项都出现1次
  - 如q= jealous gossip，那么其归一化的两个非零向量都是

$$\vec{v}(q) = \frac{1}{\sqrt{1+1}} = 0.707$$

- 排序只需要相对得分
  - 于是，不需要对查询向量进行归一化
  - 只需要计算非归一化的$\vec{V}(q)$和归一化的 $\vec{v}(d)$，然后计算 $\vec{V}(q) \bullet \vec{v}(d)$
  - 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

- Length[] 数组中存放的是每个文档向量的长度(即归一化因子)
- Scores[] 数组放的是每篇文档的得分。

```
1   float Scores[N] = 0
2   Initialize Length[N]
3   for each query term t                          遍历所有词项
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}         分数累加
7   Read the array Length[d]
8   for each d                                     归一化
9   do Scores[d] = Scores[d]/Length[d]
10  return Top K components of Scores[]
```

精确top k检索加速方法二：堆排序法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
  - 利用堆结构从J中选K个最大的

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
- 接下来将采用与查询无关的另外一种反映结果好坏程度的指标(静态质量)
  - 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
  - 于是可以将文档按照PageRank排序 $g(d_1) > g(d_2) > g(d_3) > \ldots$
  - 将PageRank和余弦相似度线性组合得到文档的最后得分

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

提前终止计算

- 假设:
  - PageRank $g \rightarrow [0, 1]$;
  - 检索算法按照$d_1,d_2,...$，依次计算(以文档为单位的计算，document-at-a-time)，当前处理的文档的$g(d) < 0.1$;
  - 而目前找到的top K 的文档得分中最小的都 > 1.2
- 由于后续文档的得分*net-score*(q,d)不可能超过1.1 (根据*net-score*$(q,d) = g(d) + cos(q,d)$，这里$cos(q,d) <1$，而$g(d) <0.1$)
- 所以，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
- 一个自然而然的问题就是能否尽量减少参与计算文档的数目，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，也适用于其它相似度计算方法

策略一：索引去除(Index elimination)

- 对于一个包含多个词项的查询来说，很显然可以仅仅考虑那些至少包含一个查询词项的文档
- 可以进一步拓展这种思路
  - 只考虑那些词项的idf 值超过一定阈值的文档
  - 只考虑包含多个查询词项(一个特例是包含全部查询词项)的文档

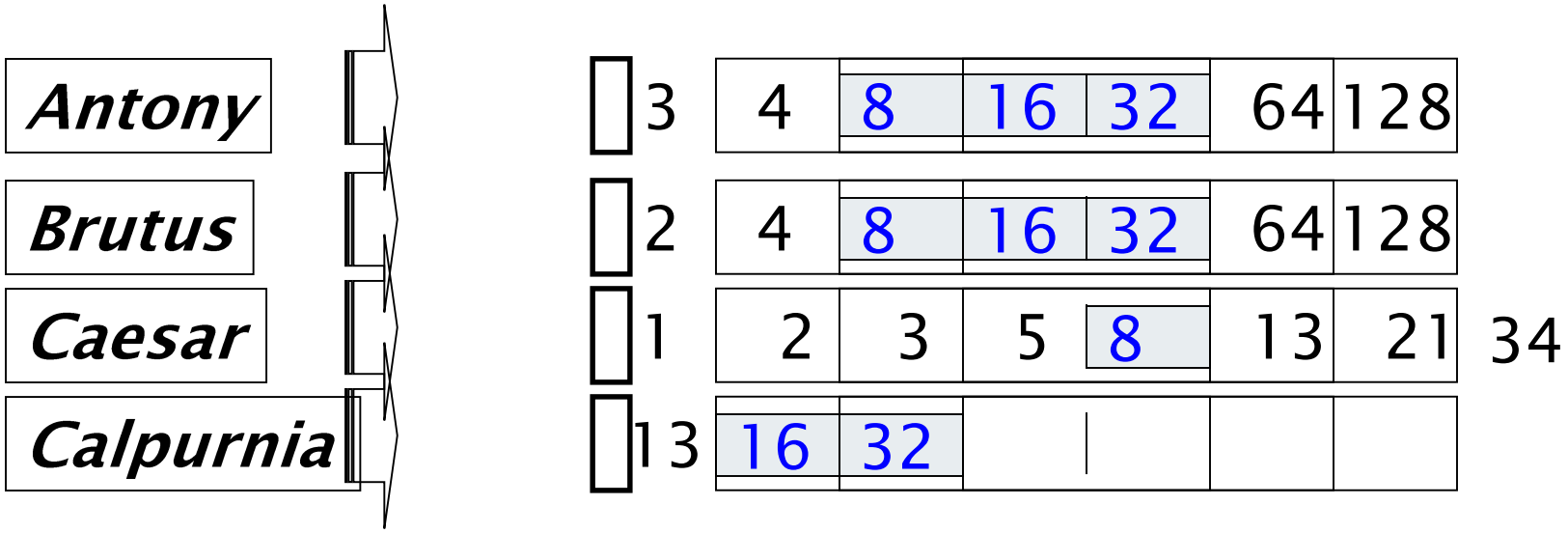
词项的idf 值超过一定阈值的文档

- 在查询catcher in the rye 时
- 只有catcher 和rye 的倒排记录表才会被遍历
- 显而易见：in 和the 的作用很小(idf很小)
- 优点：
  - 含有低idf值的词项的文档非常多，采用这种方法可以将大量无关的文档从候选集合A中去除

仅考虑包含多个查询词项的文档

- 那些至少包含一个查询词项的文档才有可能成为候选文档
- 对于多词项查询，只考虑那些包含较多查询词项的文档
  - 比如，至少含有超过3/4的查询词项
- 可以在倒排记录表遍历过程中实现

4中含3

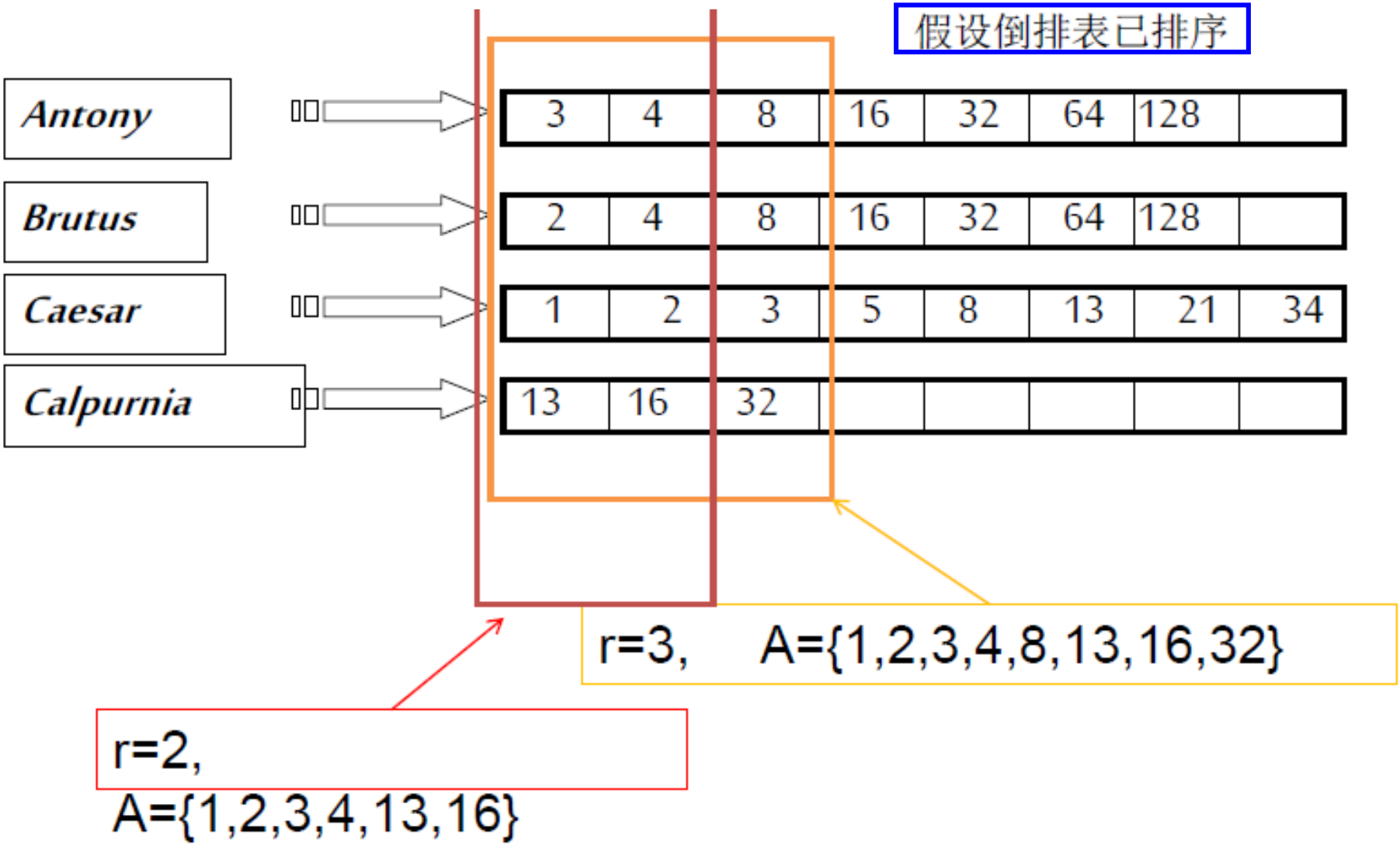


仅对文档8、16和 32进行计算

策略二：胜者表(Champion list)

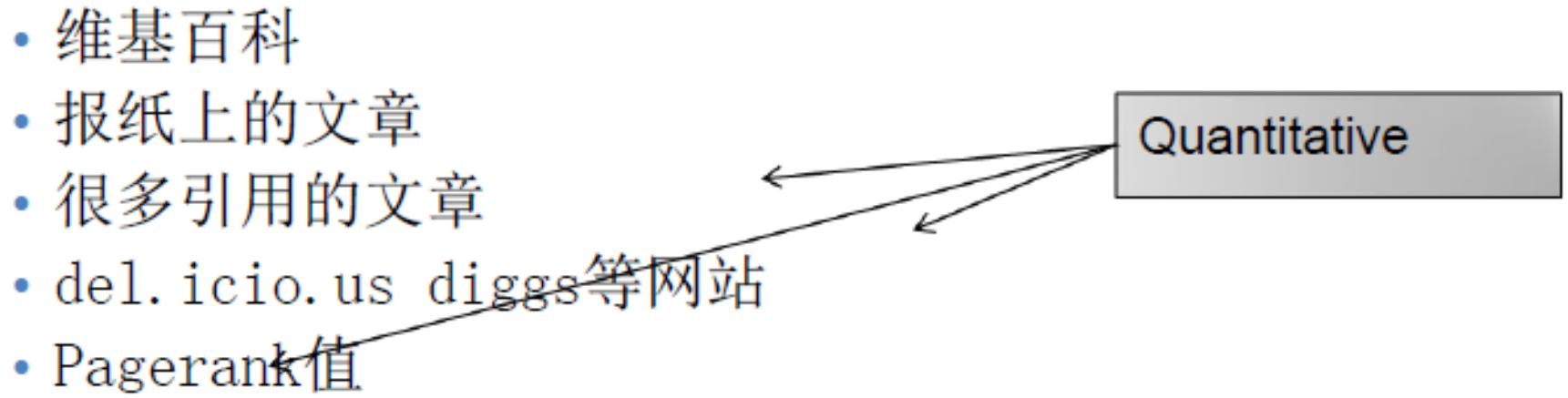
- 对于词典中的每个词项t，预先计算出r个最高权重的文档
  - 词项t 所对应的tf值最高的r篇文档构成t的胜者表
  - 也称为优胜表(fancy list)或高分文档(top doc)
  - 其中r 的值需要在索引建立时给定
    - 因此，有可能出现$r < K$ 的情况
- 给定查询q，对查询q中所有词项的胜者表求并集生成集合A
  - 根据余弦相似度大小从A中选取前top K 个文档

胜者表例子


假设倒排表已排序
Antony
Brutus
Caesar
Calpurnia
3 4 8 16 32 64 128
2 4 8 16 32 64 128
1 2 3 5 8 13 21 34
13 16 32
r=3, A={1,2,3,4,8,13,16,32}
r=2,
A={1,2,3,4,13,16}

策略三：静态得分

- 希望排序靠前的文档既是相关的又是权威的
  - 相关性通过余弦相似度得分来判断
  - 权威性是与query无关的文档本身的属性决定的
    - 权威性标志举例



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。
  - $net\text{-}score(q,d) = g(d) + cosine(q,d)$
  - 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
- 接下来的目标是找net-score最高的top K文档(非精确检索)

基于net-score的Top *K*文档检索

- 首先按照$g(d)$从高到低将倒排记录表进行排序
  - 该排序对所有倒排记录表都是一致的(只与文档本身有关)
- 因此，可以并行遍历不同查询词项的倒排记录表来
  - 进行倒排记录表的合并
  - 余弦相似度的计算

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

全局胜者表

- 将$g(d)$排序和胜者表相结合
- 对于选择好的r 值，对每个词项t 构建一个全局胜者表
  - 其中包含了$g(d) + tf\text{-}idf_{td}$得分最高的$r$篇文档
- 当查询提交以后，对所有全局胜者表的并集中的文档计算其最后得分
- 根据最终得分选择top K

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

策略四：影响度(Impact)排序

- 以前的方法都是对文档采用单一排序方式排序，如按照文档ID或静态得分$g(d)$(文档内容无关)，能支持并发扫描
- 本部分：多个term对应的文档次序不是统一的，即多种顺序(文档内容相关的排序方式)
  - 不能通过并发扫描多个倒排记录表的方式来计算文档得分
  - 在遇到每个词项时得分进行累加，即以词项为单位的得分计算(term-at-a-term)
- 思路
  - 将词项t对应的所有文档d按照$tf_{t,d}$值降序排列(不同的文档对不同的t具有不同的顺序)
- 有两种思路可以显著降低用于累加得分的文档数目

思路1：提前结束

- 对某个查询词项t对应的倒排记录表进行从前往后扫描时，可以在某个阶段停止
  - 停止条件一：扫描了r 篇固定数目的文档

  或者采用
  - 停止条件二：是当前记录的tf_{td} 已经低于某个阈值

思路2：词项按照idf降序排列

- 词项按照idf降序排列(指query里面的词项)
  - 对最终得分贡献最大的查询词项首先被考虑
- 在查询处理过程中进行自适应处理
  - 当遇到具有较低idf值的查询词项时，可以根据和前一个查询词项的文档得分的改变值(即累加得分的增量)来决定是否需要处理，改变值达到最小限度的时候终止。
- 例如：查询catcher in the rye 时，按idf排序应该是catcher，rye，in，the，当我们依次处理，处理到in时发现分数改变值很小，对排序影响很小，所以终止。

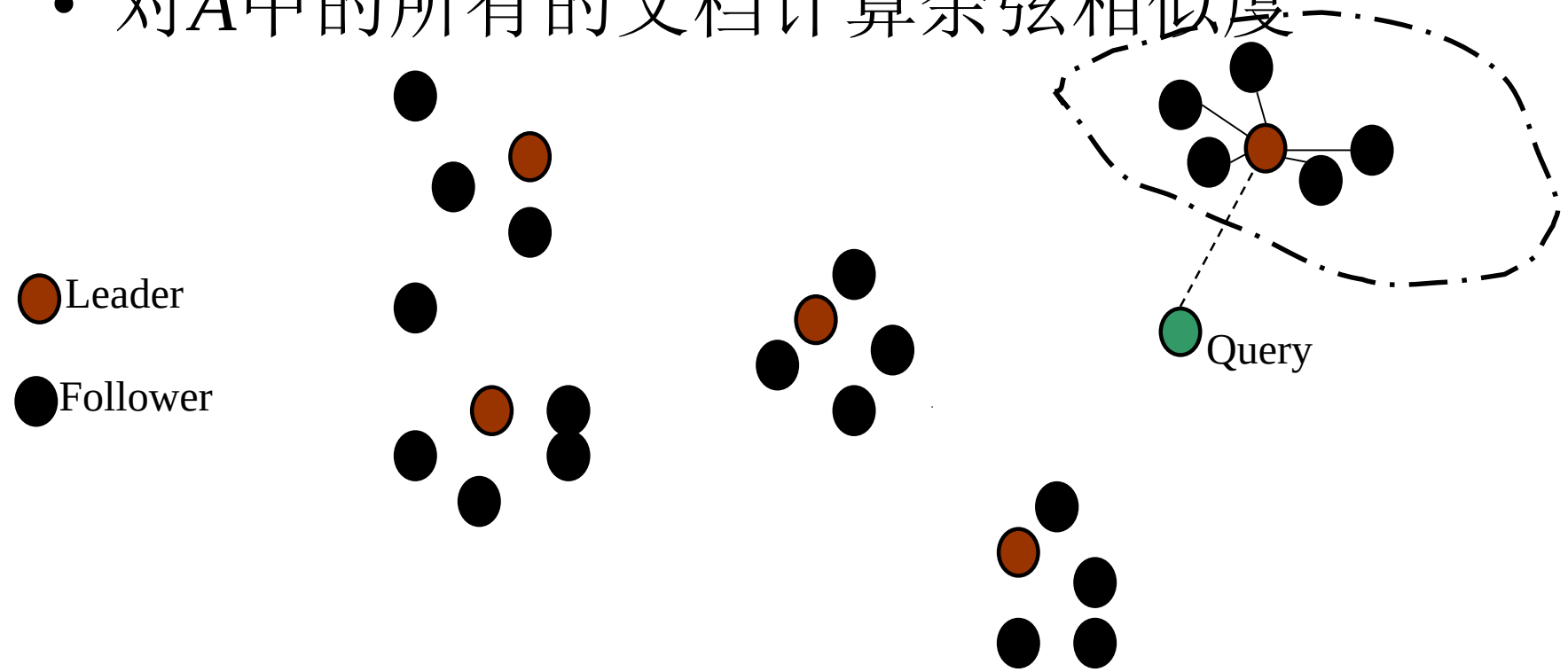
策略五：簇剪枝方法—预处理

- 随机选$\sqrt{N}$ 篇文档作为先导者

- 对于其它文档，计算和它最近的先导者
  - 这些文档依附在一个先导者上面，称为追随者(follower)
  - 这样一个先导者平均大约有 ~$\sqrt{N}$个追随者

簇剪枝方法—查询处理

- 给定查询q，通过与先导者计算余弦相似度，找出和它最近的一个先导者L
- 候选集合A：包括L及其追随者
- 对A中的所有的文档计算余弦相似度



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

常见变形

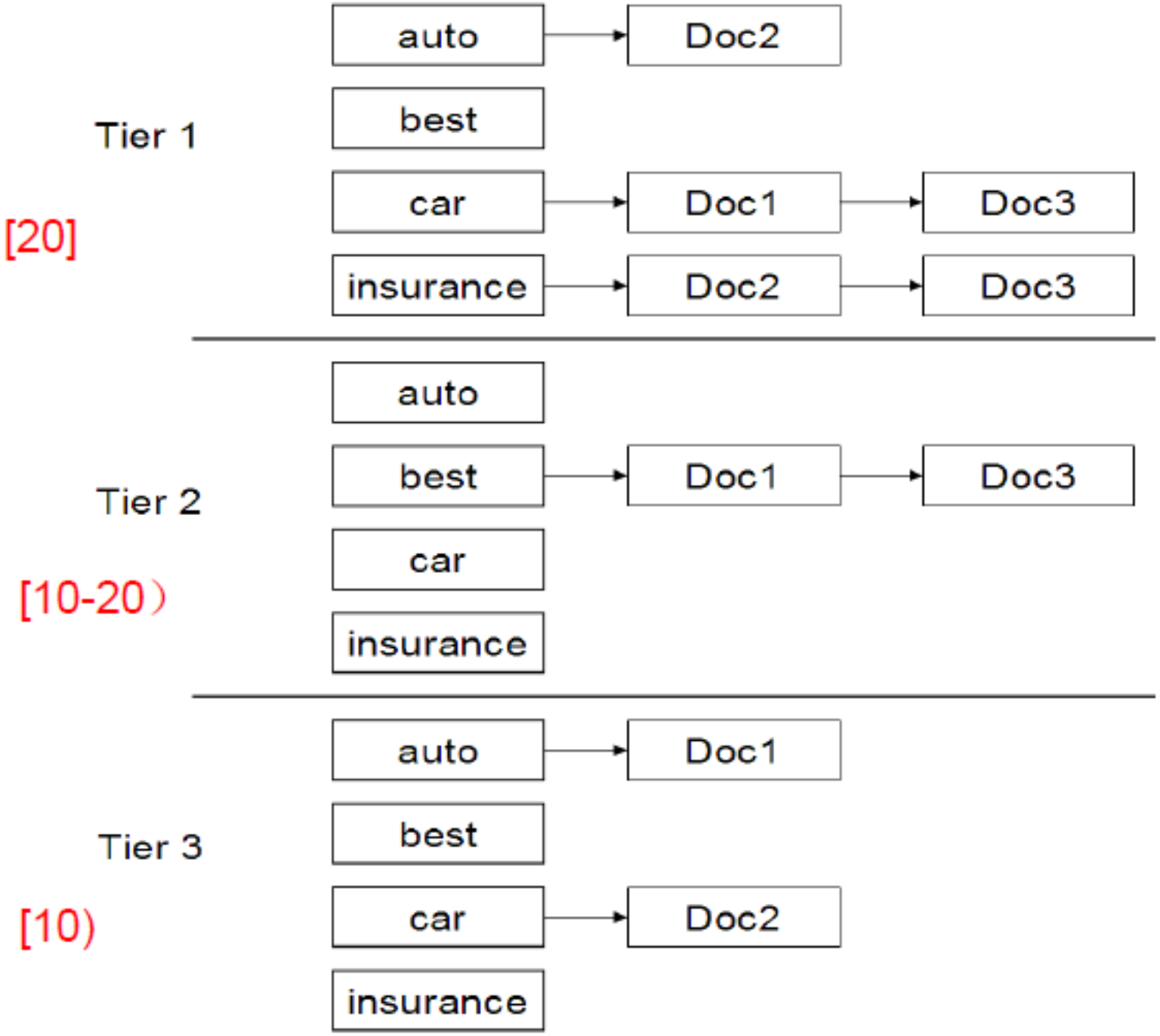
- 预处理时，将每个追随者分配给离它最近的b_1个先导者
- 查询处理时，将考虑和查询q最近的b_2个先导者
- 很显然，前面讲到的方法只是该方法在$b_1 = b_2 = 1$情况下的一个特例

目录

- 结果排序的重要性
- 结果排序的实现
- 搜索系统的构成

层次索引

- 可以看成是优胜表的一般化形式
  - 最重要
  - …
  - 最不重要
- 可以用静态得分(查询无关)或者其它得分衡量
- 倒排记录表按照重要性降序转化成层次索引
- 查询是只用上层索引，除非上层索引返回结果小于K
  - 上层返回结果小于K则再从下一层中检索


auto
Doc2
best
Tier 1
car
Doc1
Doc3
[20]
insurance
Doc2
Doc3
auto
best
Doc1
Doc3
Tier 2
car
[10-20）
insurance
auto
Doc1
best
Tier 3
car
Doc2
[10)
insurance

查询词项的邻近性

- 自由文本查询：用户输入几个词项到搜索框——一般的互联网检索
- 用户往往希望返回的文档中大部分或者全部查询词项之间的距离比较近
- 令文档d中包含所有查询词项的最小窗口大小为ω，其取值为窗口内词的个数
- 假设某篇文档仅仅包含一个句子The quality of mercy is not strained，那么查询strained mercy在此文档中的最小窗口大小是4
- 用窗口大小来度量位置关系

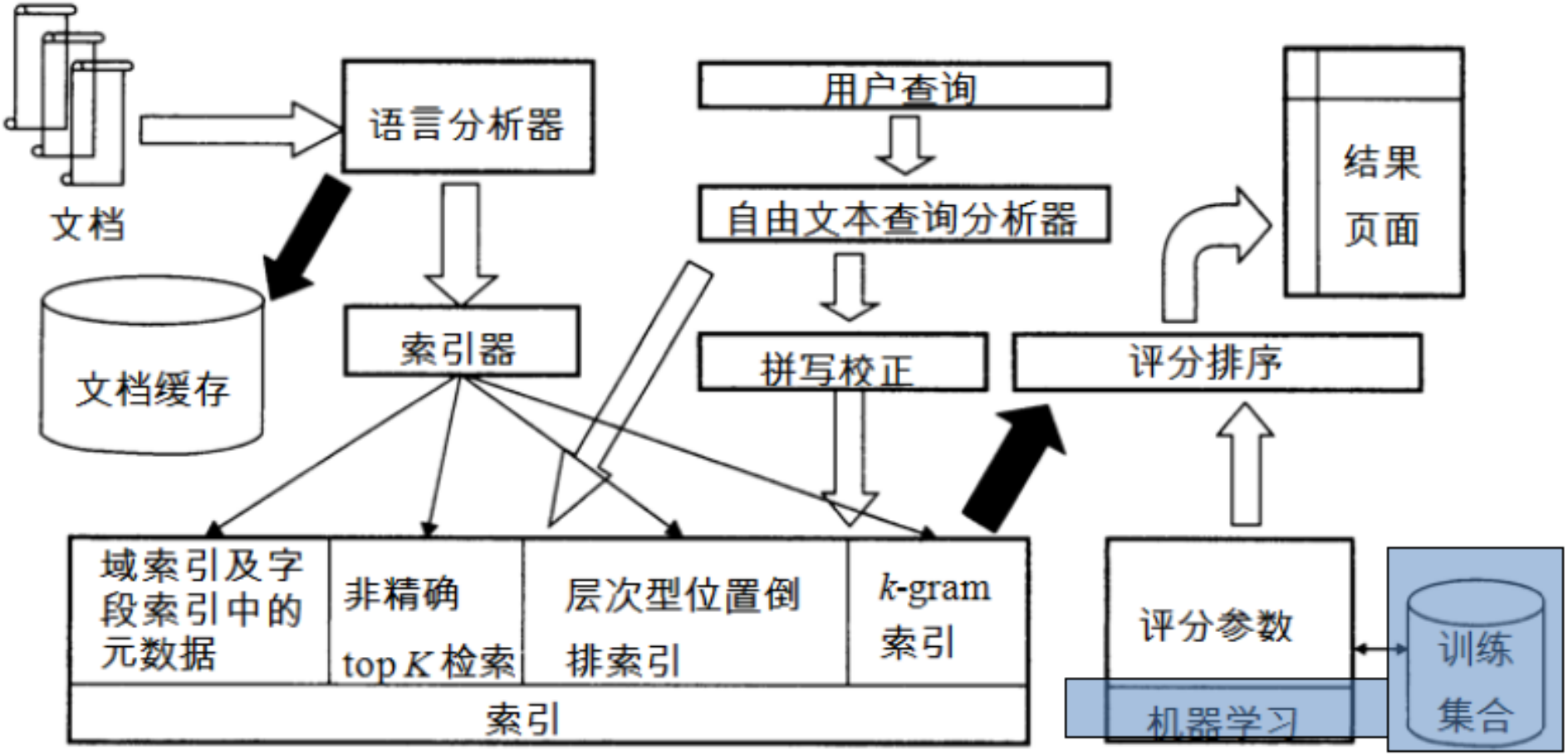
查询分析器

- 自由文本查询对用户输入的关键词可能需要基于底层索引结果对多个查询进行处理，如查询rising interest rates之类query时，查询分析器可能做如下操作：
  - 1. 将用户输入的查询字符串看成一个短语查询
  - 2. 如果包含短语“rising interest rates”的文档数目少于10 篇，那么会将原始查询看成rising interest和interest rates两个查询短语，同样通过向量空间方法来计算
  - 3. 如果结果仍然少于10个，重新利用向量空间模型求解，认为3个查询词项之间是互相独立的

综合评分

- 已经介绍的评分函数有余弦相似度、静态得分、近邻性等。
- 如何将这些评分组合才是最优的？
- 通用方法——机器学习

搜索系统组成


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