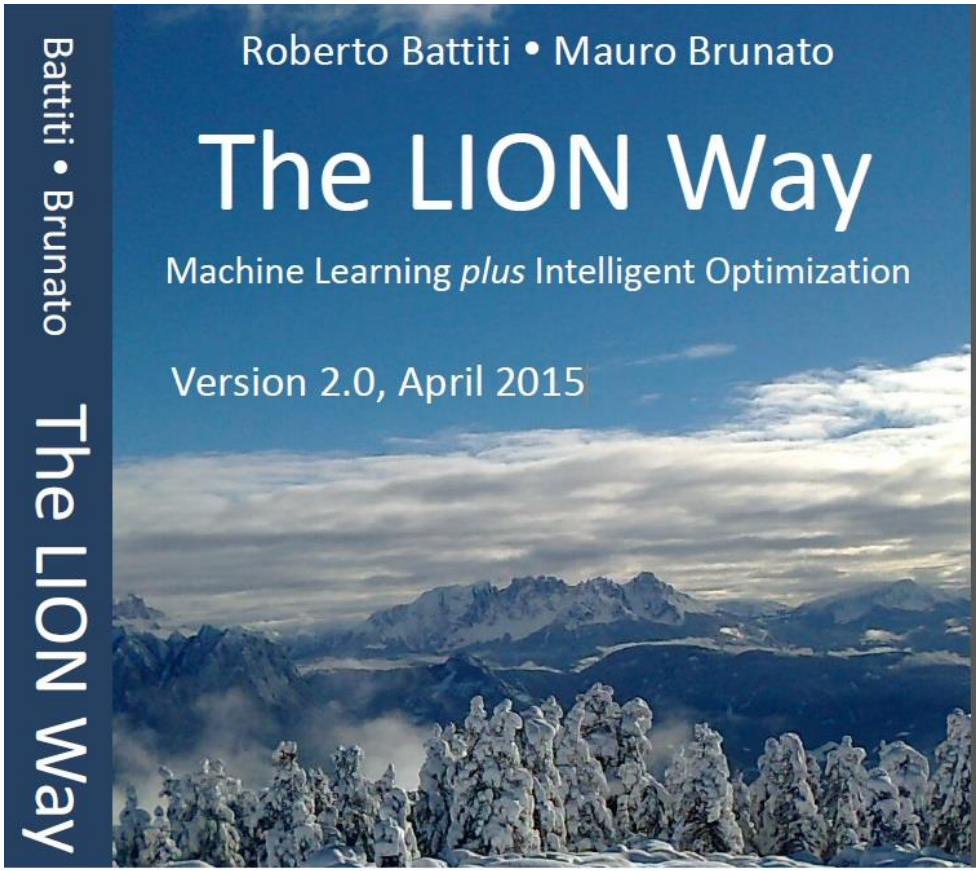


ROBERTO BATTITI, MAURO BRUNATO.
The LION Way: Machine Learning plus *Intelligent Optimization*.
LIONlab, University of Trento, Italy,
Apr 2015

http://intelligent-optimization.org/LIONbook

Text and web mining – part I

Wholly new forms of encyclopedias will appear, ready made with a mesh of associative trails running through them, ready to be dropped into the memex and there amplified.
(Vannevar Bush, 1945)

HTML – the *language* of the web

```
<html>
 <head>
  <title>Learning and Intelligent Optimization</title>
  <meta name="author" content="Roberto Battiti”/>
  <meta name="keywords”
    content="LION, ML, optimization, big data”/>
 </head>
 <body>
  <h1>The LION way is the future</h1>
  The reasons are explained in the
  <a href=”http://intelligent-optimization.org/”>LIONlab
     homepage </a>.
 </body>
</html>
```

HTTP – the *protocol* of the web

```
GET /thispath/thispage.html HTTP/1.1
Accept: */*
Accept-Language: it-it
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (Macintosh; U; PPC Mac OS X; it-it)
AppleWebKit/418.9.1 (KHTML, like Gecko) Safari/419.3 Connection:
keep-alive
Host: www.pippo.it
```

- Connect to well-known TCP port 80

What is web mining?

- The Web is an unstructured (or, at most, semi-structured) collection of data.
- Data come in form of human-readable texts and images. Data are hyperlinked.
- The Web is not a database
  - A complete description of data items (a schema) is missing.
  - Every word on a page can be an attribute
- The Web is a collection of human-readable data and human-exploitable hyperlinks.

Crawling the web

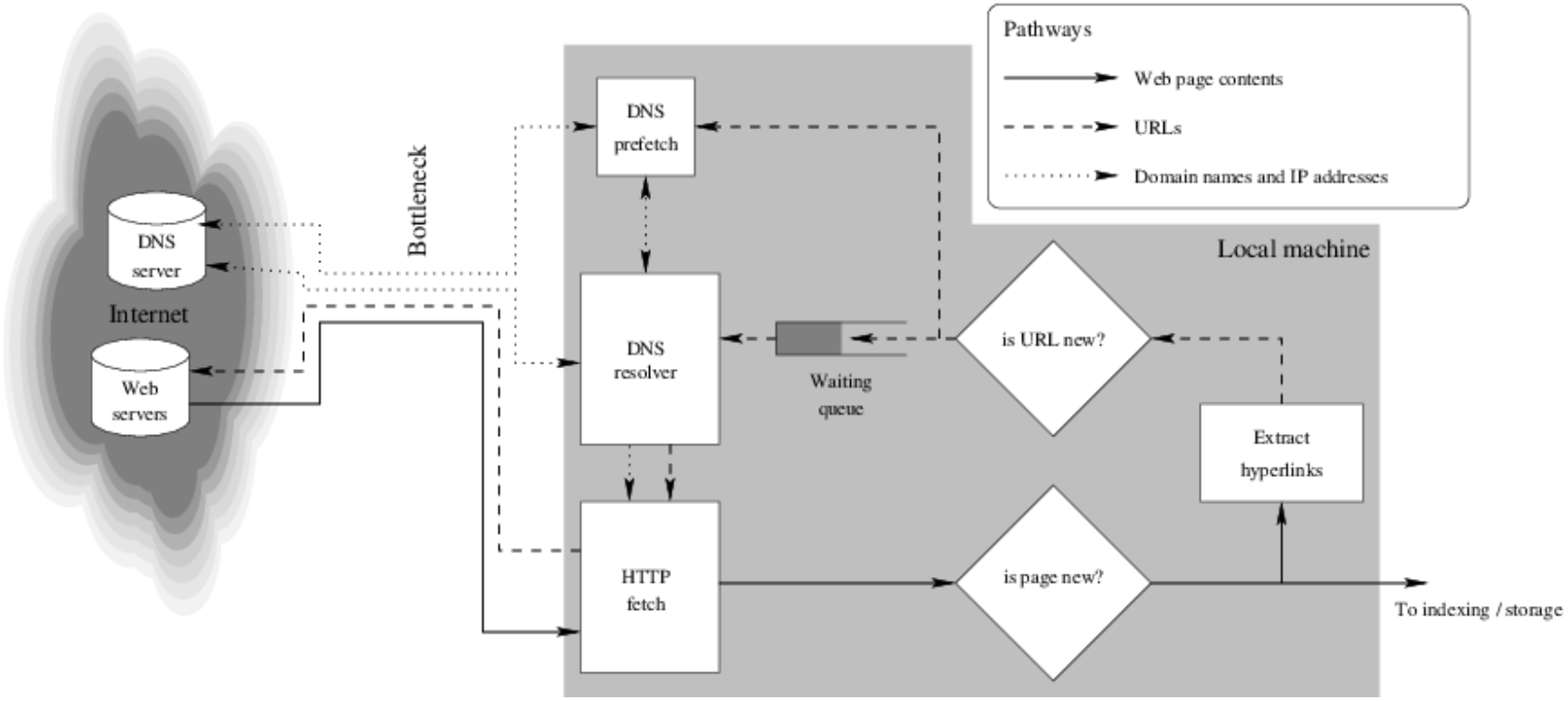
- One common need of hypertext processing: the ability of fetching and storing a large number of documents.
- Crawlers, Spiders, Web Robots, Bots
- Common examples:
  - wget
  - curl

Basic crawling principles

- Start from a given set of URLs.
- Collect pages.
- Scan collected pages for hyperlinks to pages that have not been collected yet.

- New URLS are potential new work and their set increases very fast.

A large-scale crawler



A large-scale crawler

- A single page fetch involves seconds of latency (⇒ More fetches at the same time).
- Highly concurrent DNS (possibly multiple servers).
- No multithreading, better asynchronous sockets.
- Avoid duplicate URLs.

A large-scale crawler: DNS usage

- Crawler access is often spread through different domains to avoid overloading web servers (but being more demanding to DNS servers).
- Cache can be slack with expiration times: better expired than late.
- Problem:
  - Standard DNS service in the OS does not handle concurrent requests, so a custom DNS client is necessary (asynchronous sending and receiving).
- Better solution: prefetching — do not wait for page request, but extract potential DNS queries from current pages.

A large-scale crawler: concurrent page fetches

- Web-scale crawlers fetch > 105 pages per second. Page retrievals must proceed in parallel.
- Two approaches:
  - Exploit OS-level multithreading (one thread per page).
  - Use non-blocking sockets and event handler.
- What about multiprocessors?
  - Bottlenecks are network and disks, not CPU.

A large-scale crawler: multithreading

- Multiple threads, *statically created* to avoid overhead. Call to `connect()`, `send()` or `recv()` may block one thread while others run.
- Pros
  - Easy coding, complexity delegated to OS
- Cons
  - Synchronization problems and consequent IPC overhead
  - Hardly optimized (OS assumes general purpose)
  - One disaster spoils all threads (better with processes)

A large-scale crawler: non-blocking sockets

- Single thread, arrays of non-blocking sockets, using `select()` to poll for received data.
- While doing other business (indexing, saving to disk) incoming data are buffered until the next polling cycle.
- Pros
  - Fast, little overhead from OS
  - Better control on overall status
  - No need of protection or synchronization.
- Cons
  - Harder to code: need multiple data structures.

A large-scale crawler: link extraction

- Web pages are parsed for hyperlinks. URLs must be *canonicalized*:

`www.pippo.com/here/not/../there#this`

⇓

`http://www.pippo.com:80/here/there/`

- Problems
  - Domain name - IP address relationship is many-to-many, due to load balancing needs and logical website mapping.

A large-scale crawler: avoiding repeated visits

- Visited URLs must be stored to avoid unneeded duplicate visits: need of a fast memory-based isUrlVisited? function.
- To save space, URLs are hashed, commonly by 2-level functions to exploit locality: (hostname,path).

A large-scale crawler: manage robot exclusion

- robots.txt usually helps crawlers avoid useless portions of a website

```
User-agent: LIONcrawler
Crawl-delay: 1000
Disallow: /this/path
Disallow: /that/directory

User-agent: *
Disallow: /secrets
Disallow: /dynamic/page
Disallow: /ever/changing/path
```

A large-scale crawler: avoid spider traps

- Some web sites can be maliciously designed in order to crash spiders:
  - Recursive links via soft aliases.
  - Long URLs to overflow lexers and parsers.

A large-scale crawler: per-server queues

- Web servers need to safeguard against DoS attacks.
- Crawlers must limit frequency of requests to the same server
- Span many different servers at once, but no more than *n* pages per second each (problem: DNS overload).
- Use queues.

Document indexing: queries

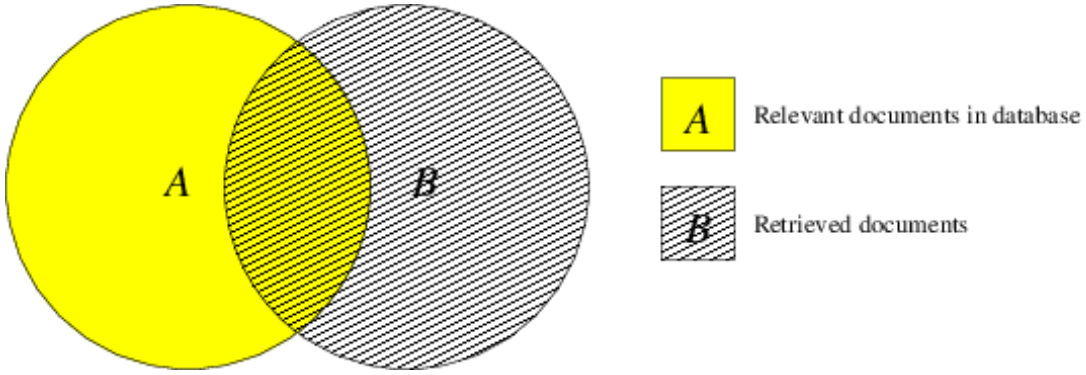
- The simplest kind of query involves relationships between terms and documents:
  - Documents containing the word “java”
  - Documents containing the word “java” but not “coffee”
- Proximity queries require the use of inverted indices.
  - Documents containing the phrase “java beans” or the word “API”
  - Documents where “java” and “island” occur in the same sentence.

Document indexing: operations on text

- filter out HTML tags
- tokenization
  - simplest case: tokens are all nonempty sequences of characters not including spaces or punctuation marks.
- stopword removal
- downcasing
- stemming
  - PLAYS PLAYING PLAYED REPLAY -> PLAY
- collapse variant forms (“am”, “is”, “are” all become “be”)

...But beware the loss of information!

Information Retrieval: Performance measures



- Retrieved relevant items (true positives): $A \cap B$
- Retrieved irrelevant items (false positives): $B \setminus A$
- Unretrieved relevant items (false negatives): $A \setminus B$

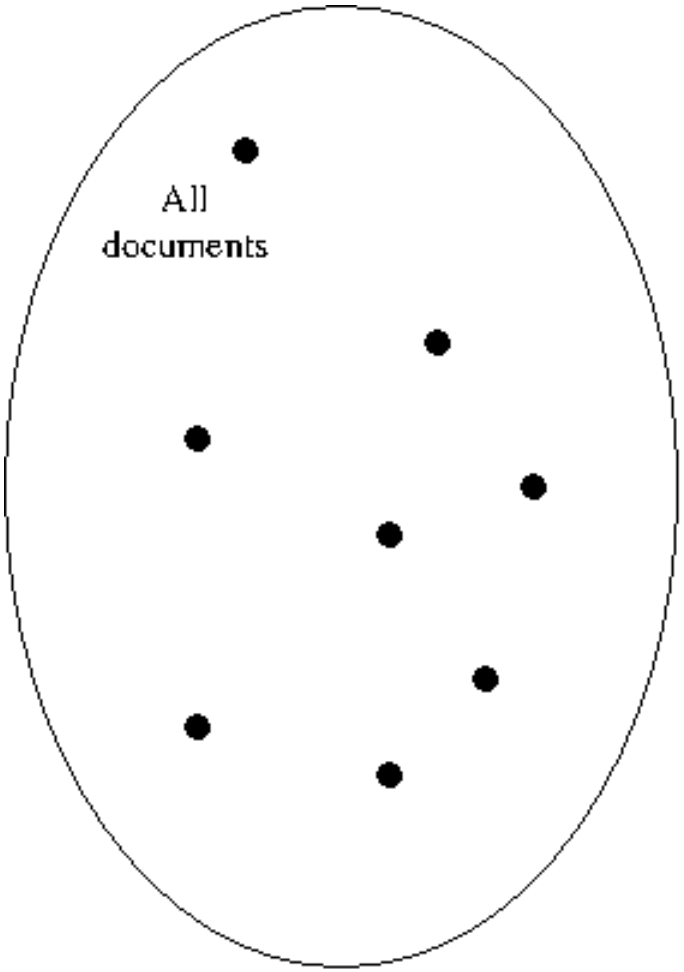
Which fraction of retrieved documents is relevant?

$$\text{Precision} = \frac{|A \cap B|}{|B|}$$

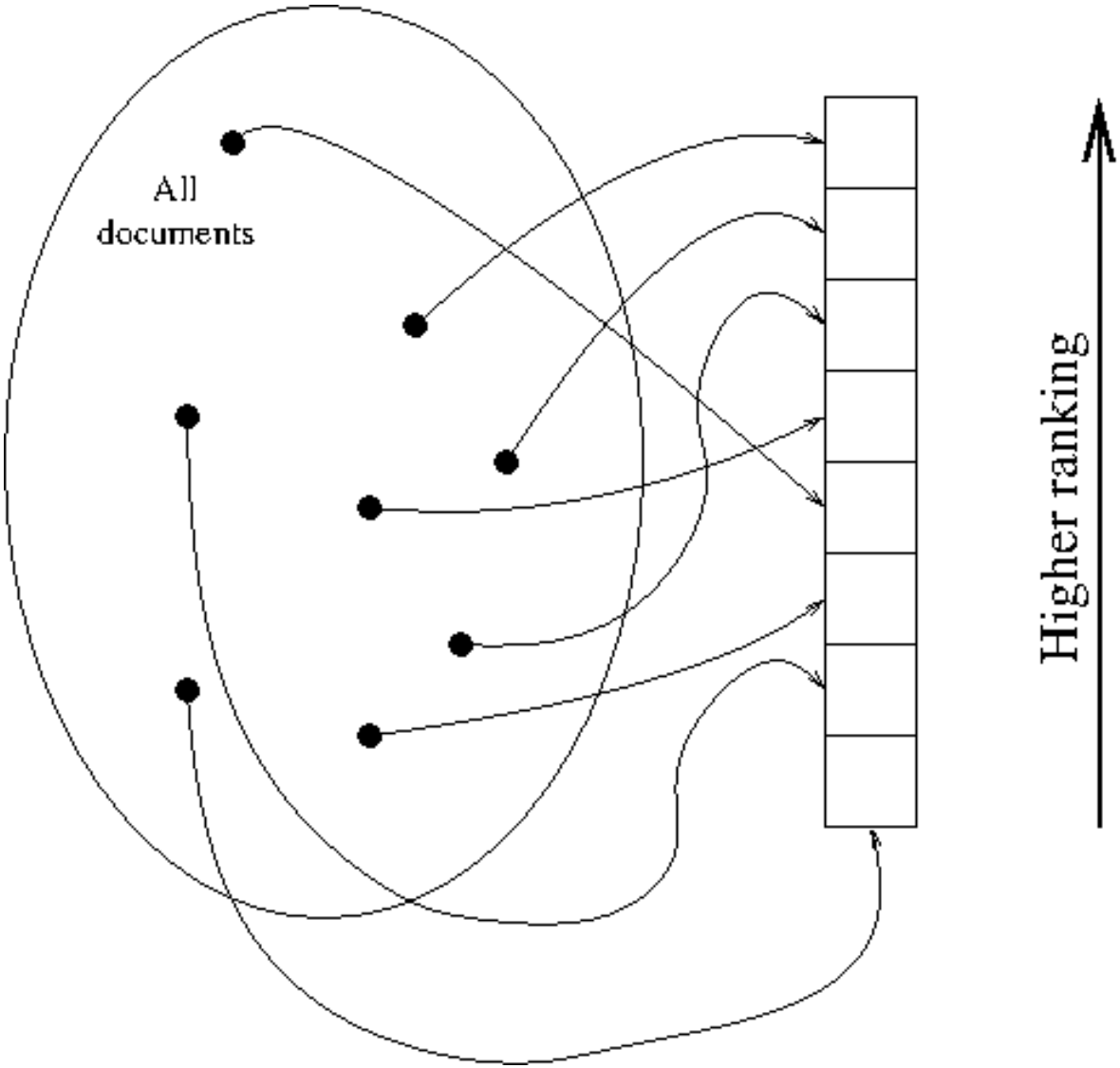
Which fraction of relevant documents has been retrieved?

$$\text{Recall} = \frac{|A \cap B|}{|A|}$$

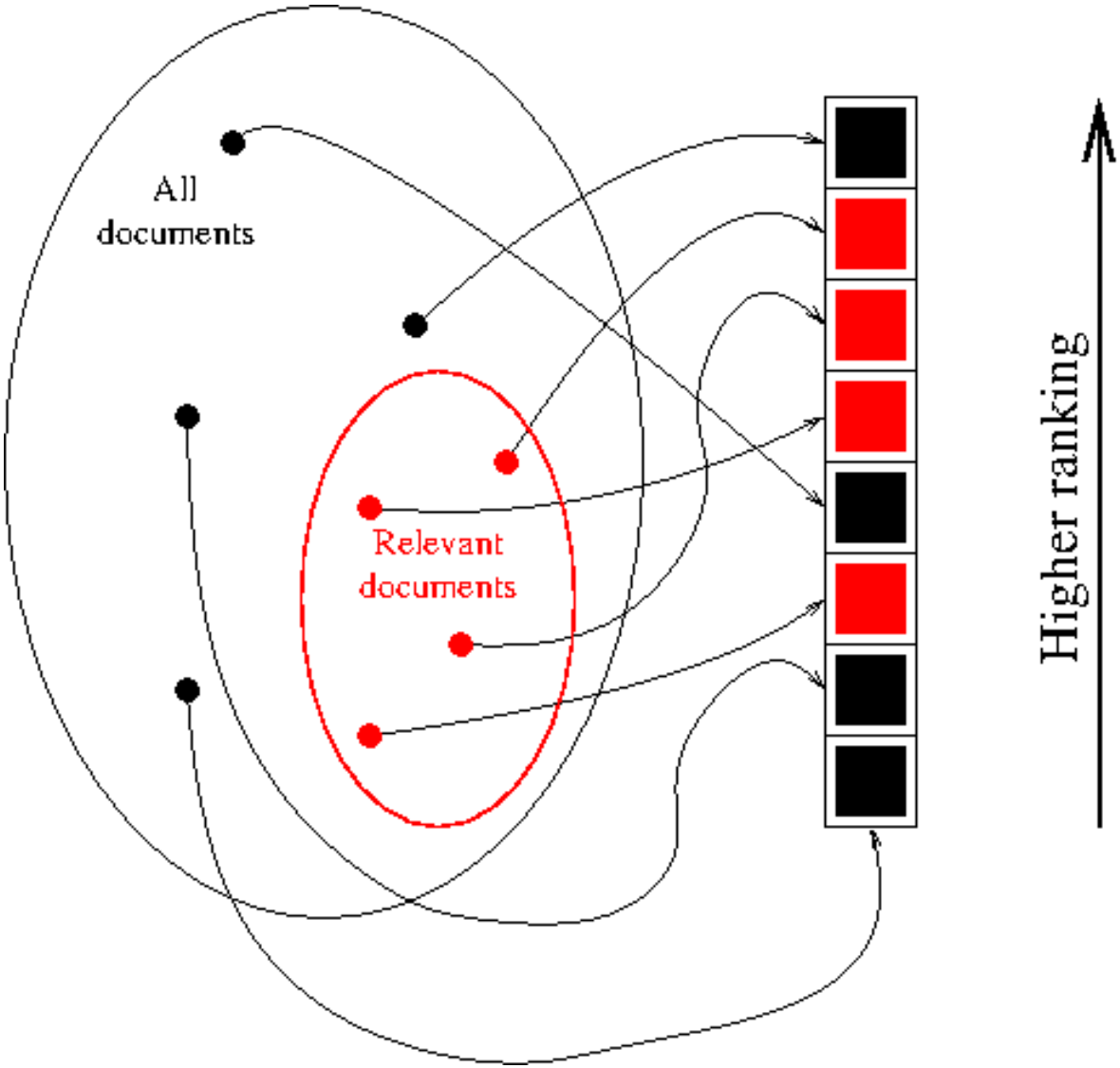
Document ranking: intuition



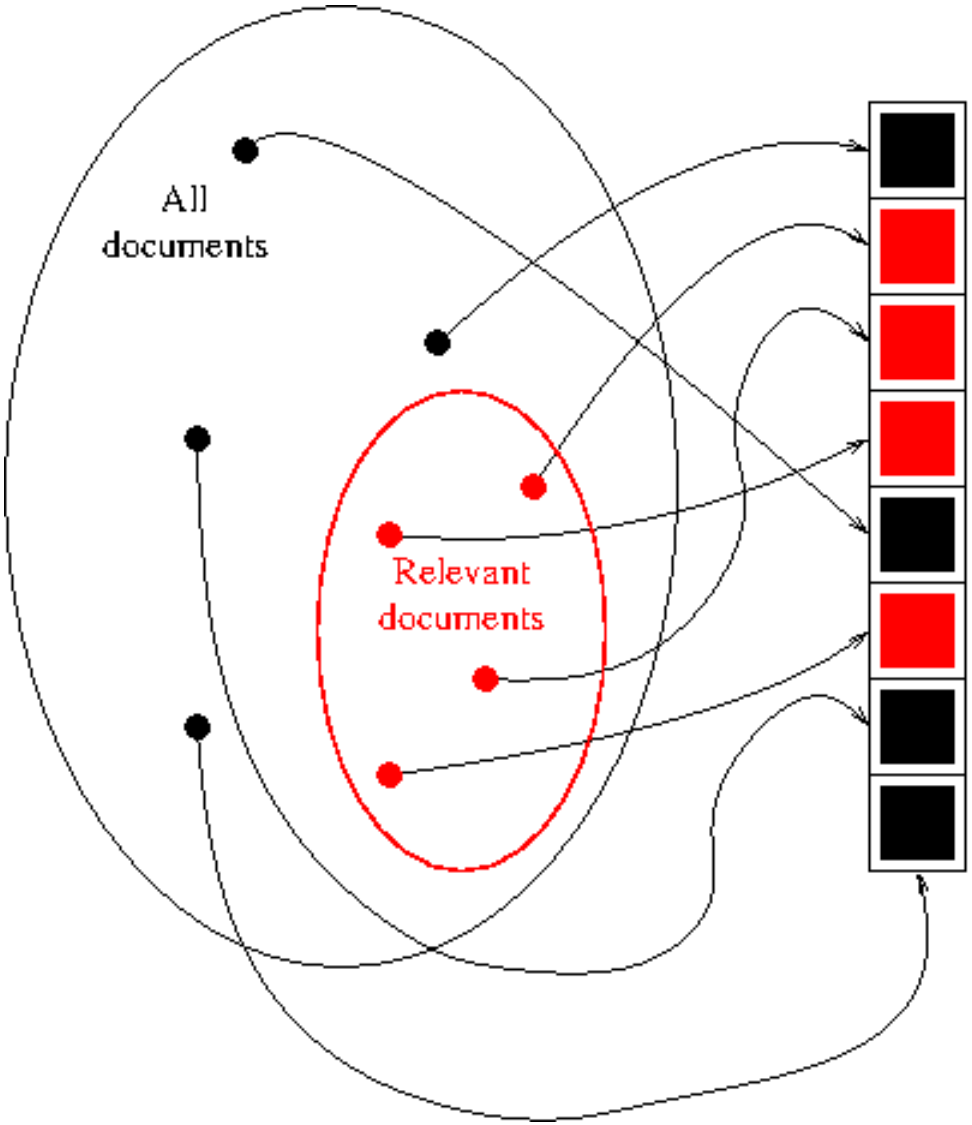
Document ranking: intuition

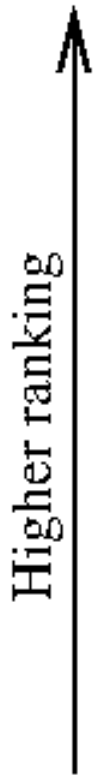


Document ranking: intuition



Document ranking: intuition





Red blocks
should appear
at the top!

Precision and recall w/ ranking

- D: corpus of $n = |D|$ documents; Q: set of queries.
- For query $q \in Q$, define $D_q \subset D$ as the set of all relevant documents (exhaustive, manually defined).
- Let $(d_1^q, d_2^q, \ldots, d_n^q)$ be an ordering ("ranking") of D returned by system in response to query q.
- Let $(r_1^q, r_2^q, \ldots, r_n^q)$ be defined as

$$r_i^q = \begin{cases} 1 \text{ if } d_i^q \in D_q \\ 0 \text{ otherwise} \end{cases}$$

Precision and recall w/ ranking

- Recall(*k*): fraction of relevant documents found in the top *k* positions

$$\text{recall}_q(k) = \frac{1}{|D_q|} \sum_{i=1}^{k} r_i^q$$

- Precision(*k*): fraction of top *k* documents that are relevant

$$\text{precision}_q(k) = \frac{1}{k} \sum_{i=1}^{k} r_i^q$$

Precision and recall w/ ranking

- Average precision

$$\text{avg.precision}_q = \frac{1}{|D_q|} \sum_{k=1}^{|D|} r_k^q \, \text{precision}_q(k)$$

Precision / recall tradeoff

- Average precision is 1 iff all relevant documents are ranked before irrelevant ones
- Interpolated precision at recall = ρ: maximum precision for recall greater or equal to ρ.
- By convention, $\text{precision}_q(0) = 1$ and $\text{recall}_q(0) = 0$.
- Recall can be increased by increasing k, but then more and more irrelevant documents occur, driving down precision.
- Therefore, a recall-precision plot has a downward slope.

Precision / recall tradeoff

- "Interpolated precision"
  - Answering the question "What is the best precision I can get for a recall score no smaller than *r*?"

$$\text{interpolated\_precision}_q(r) = \max_{k:\text{recall}_q(k) \geq r} \text{precision}_q(k)$$

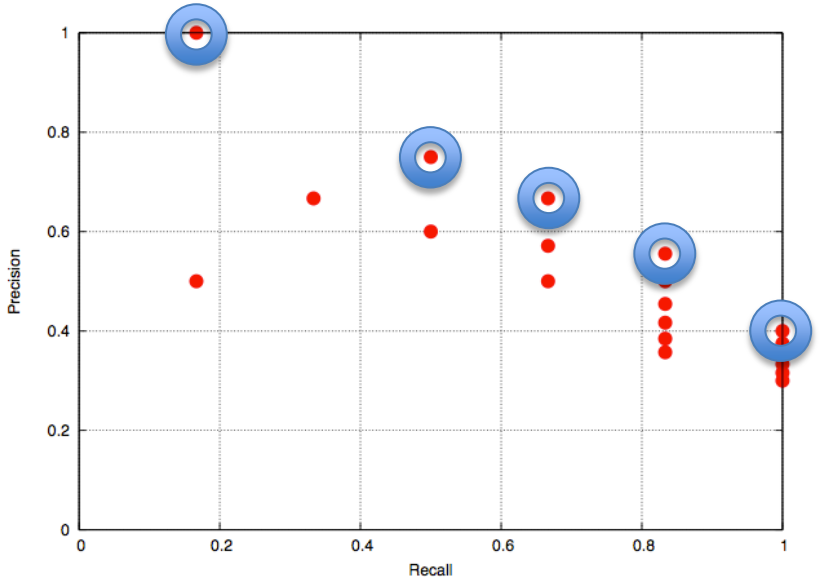
- Stepwise constant, non-increasing function of recall rate.

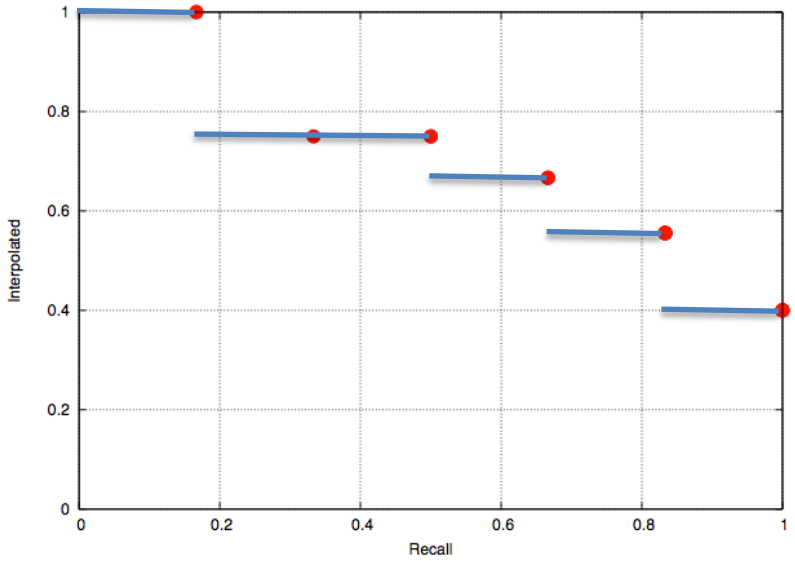
Precision / recall tradeoff

= non-dominated points
(in the Pareto sense)

= interpolated precision

k	r_k^q
1	1
2	0
3	1
4	1
5	0
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	0
15	1
16	0
17	0
18	0
19	0
20	0





The vector-space model

- Representing document as points in a multi-dimensional space, each axis representing a term (token).
- Coordinate of document *d* in direction of term *t* determined by:

$$\mathrm{TF}(d,t) = \frac{n(d,t)}{|d|}$$

The vector-space model

- Inverse document frequency:
  - rewards rare terms, small for frequent terms
  - smooth, slow growth

$$\mathrm{IDF}(t) = \log \frac{1 + |D|}{|D_t|}$$

The vector-space model

- Document *d* is represented by vector

$$\mathbf{d} = (d_t)_{t \in T} \in \mathbf{R}^{|T|}$$

- where component d_t is

$$d_t = \mathrm{TF}(d,t) \times \mathrm{IDF}(t)$$

- A query is a sequence of terms, therefore it has the same representation.

Proximity between documents

- Euclidean distance: to avoid artifacts, vectors should be normalized: an *n*-fold replica of document *d* should have the same similarity to *q* as *d* itself.

$$\text{dist}(\mathbf{d}, \mathbf{q}) = \left\| \frac{\mathbf{d}}{\|\mathbf{d}\|} - \frac{\mathbf{q}}{\|\mathbf{q}\|} \right\|$$

Proximity between documents

- Cosine similarity: cosine of the angle between vectors **d** and **q**.

$$\mathrm{sim}(\mathbf{d}, \mathbf{q}) = \left\| \frac{\mathbf{d} \times \mathbf{q}}{\|\mathbf{d}\| \|\mathbf{q}\|} \right\|$$

TFIDF-based IR system

- Information Retrieval system based on TFIDF coordinates:
  - Build inverse index with TF(t,d) and IDF(t) information
  - Given a query, map it onto TFIDF space
  - Sort documents according to similarity metric
  - return most similar documents
- Now we are ready to refine the search!

Relevance feedback

- The average web query is as few as two terms long!
- After the first response, a sophisticated user learns how to improve his query.
  For everybody else. . .
  - Results page may include a rating form for documents (“Please mark documents that you have found useful”)
  - User’s form submission is a form of relevance feedback.

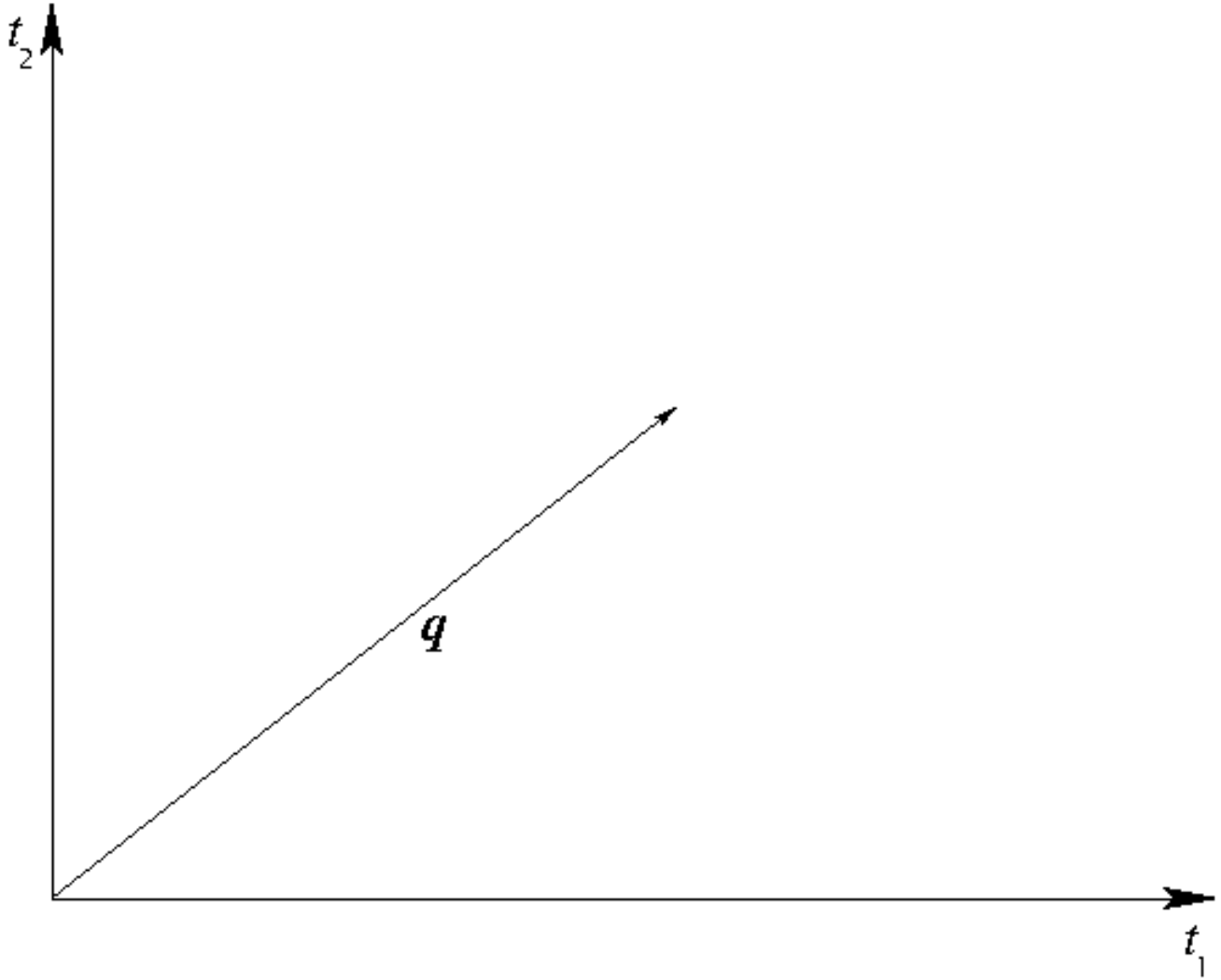
Relevance feedback: Rocchio's method

Correct query **q** by pushing it closer to a set of useful documents D_+ and pulling it apart from a set D_- of useless docs:

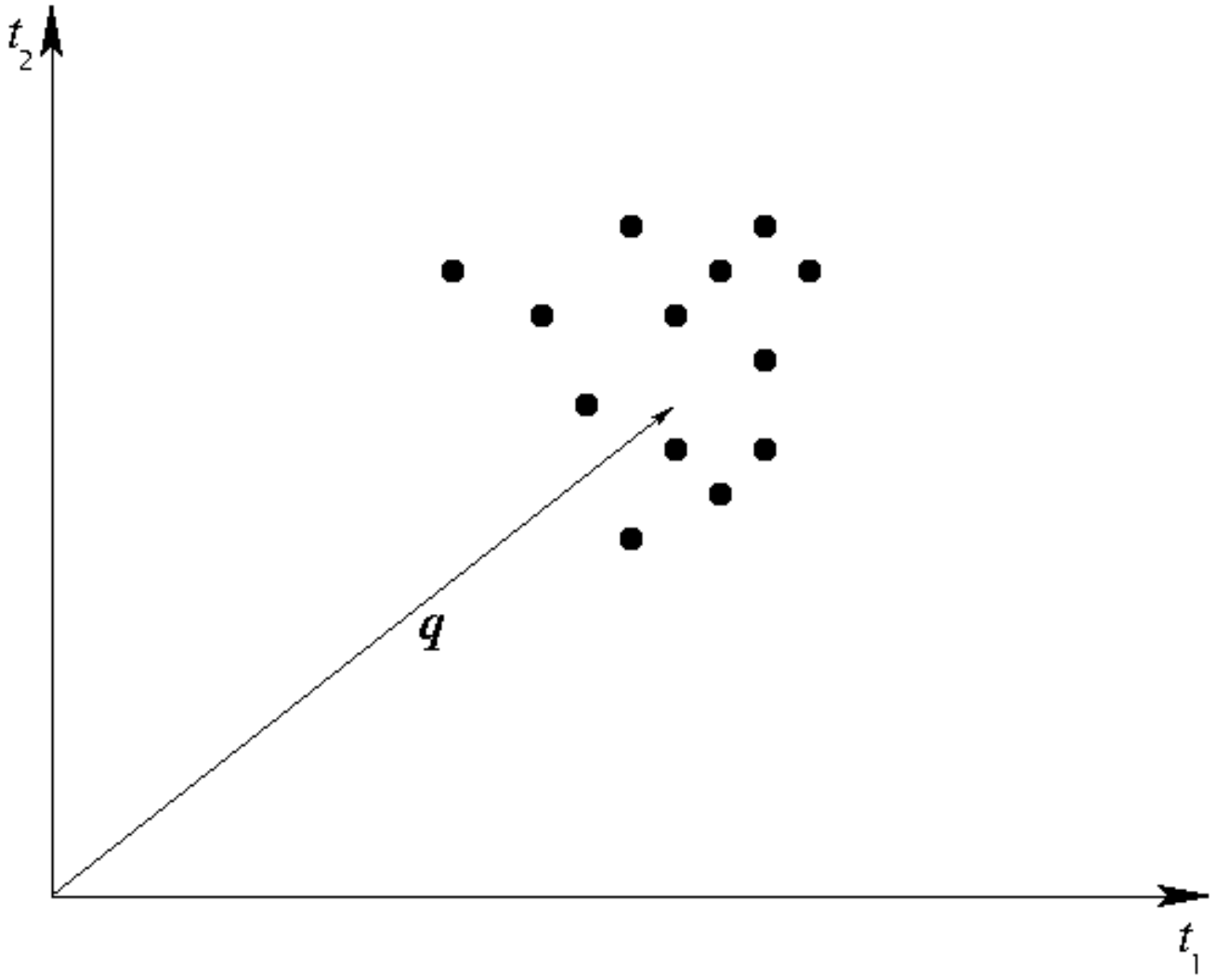
$$\mathbf{q}' = a\mathbf{q} + b \sum_{\mathbf{d} \in D_+} \mathbf{d} - g \sum_{\mathbf{d} \in D_-} \mathbf{d}$$

Parameters α, β and γ control the amount of modification.

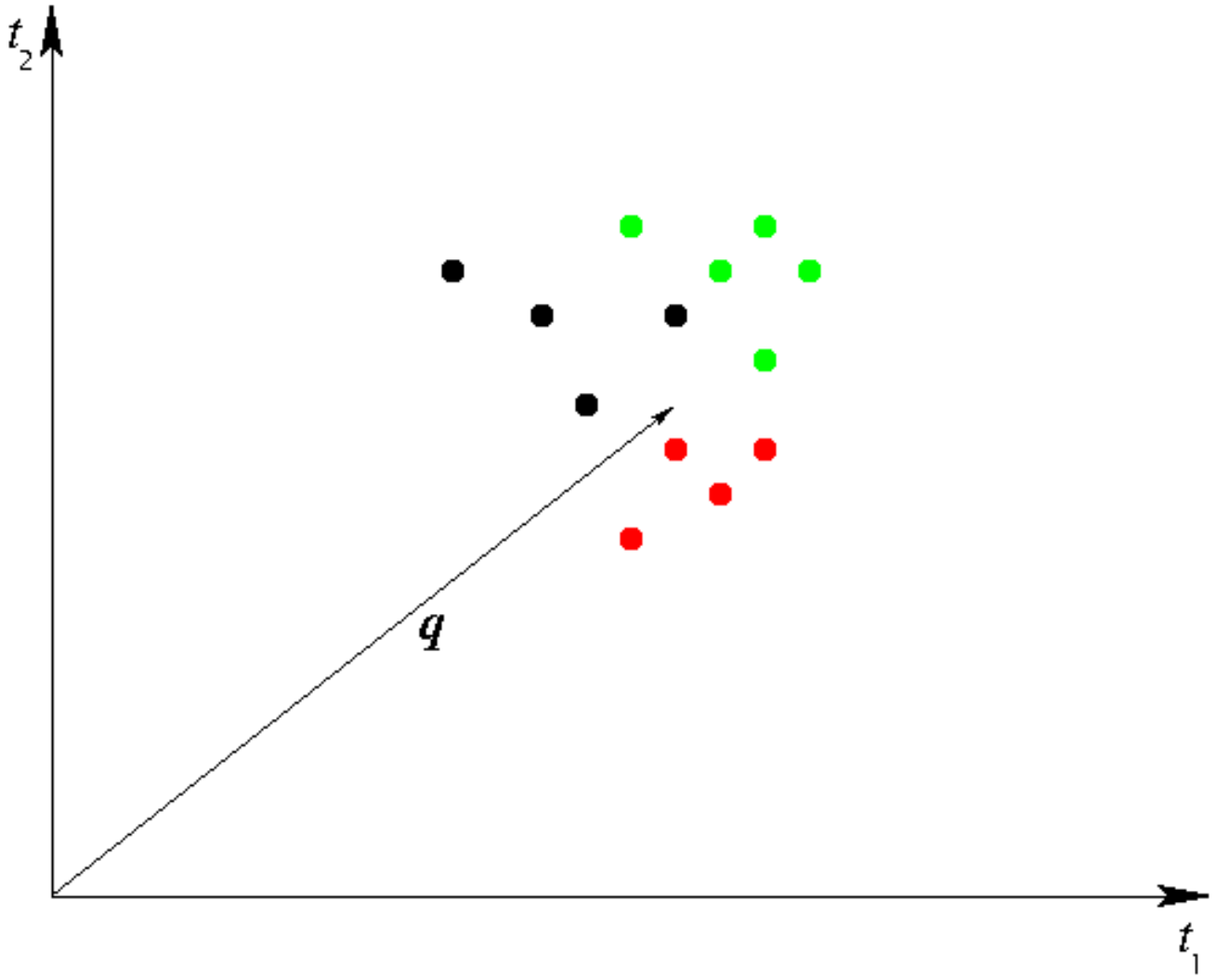
Relevance feedback: Rocchio's method



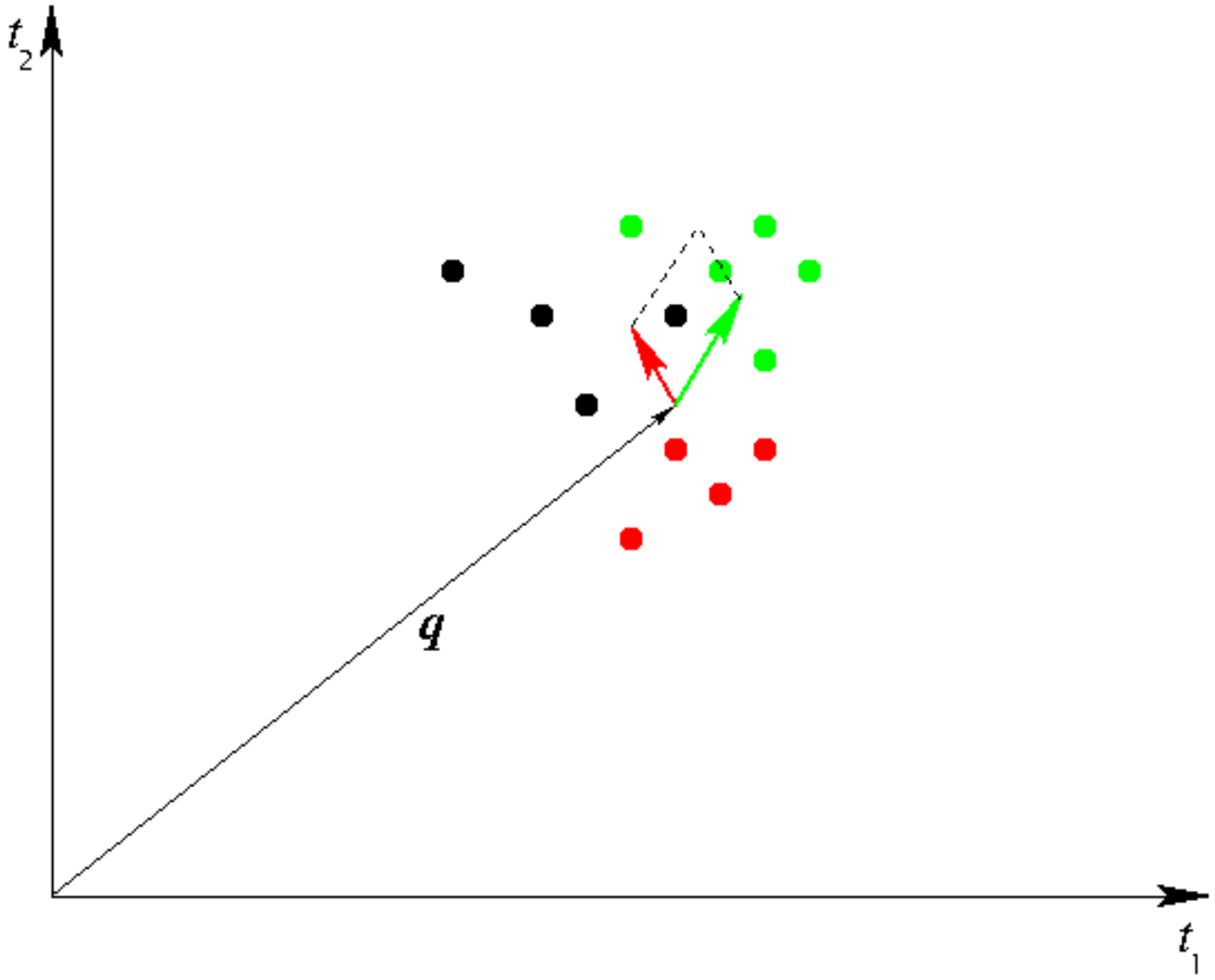
Relevance feedback: Rocchio's method



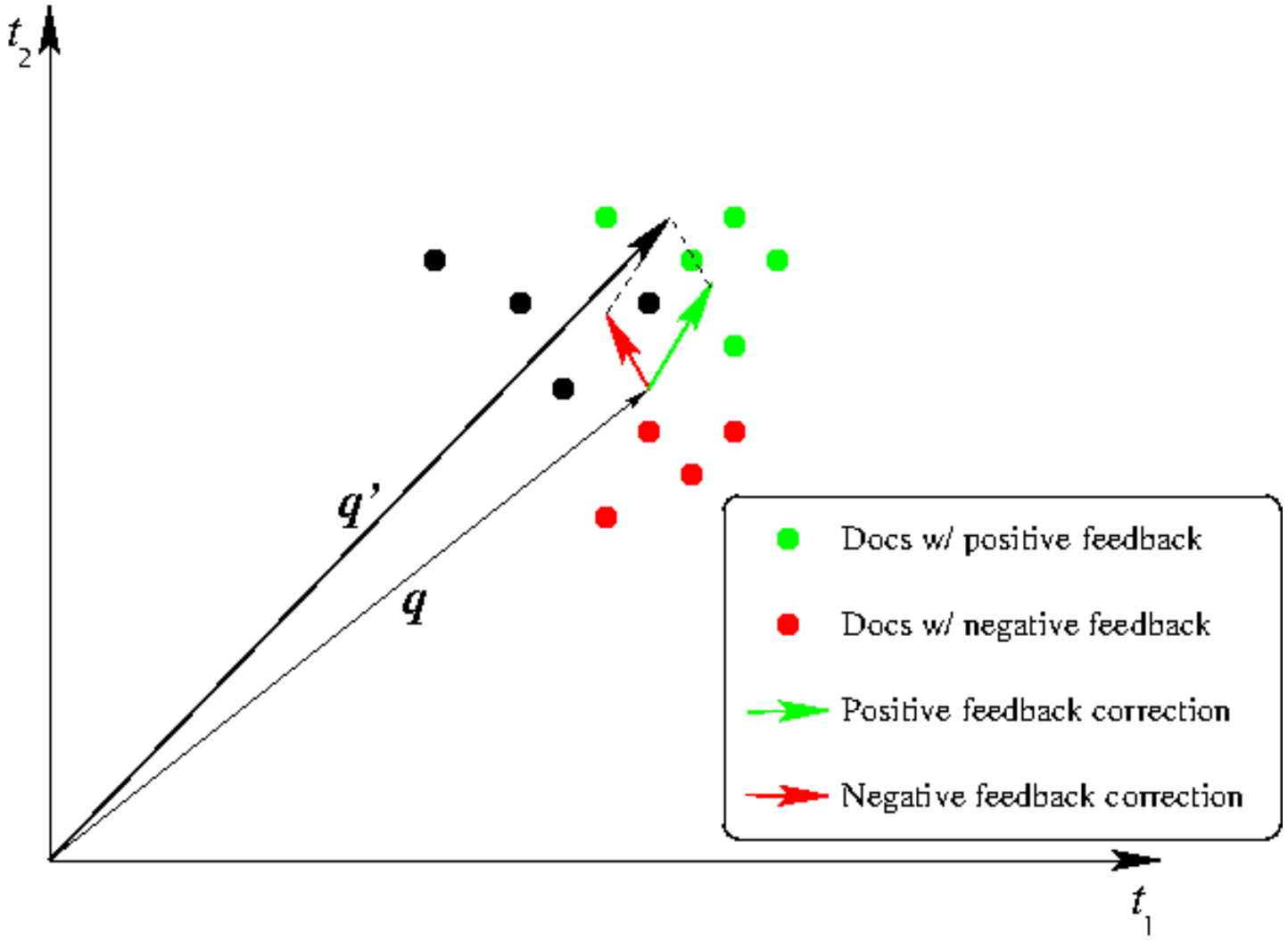
Relevance feedback: Rocchio's method



Relevance feedback: Rocchio's method



Relevance feedback: Rocchio's method



Relevance feedback: Rocchio's method

- If user input is absent:
  - Automatically build D_+ by assuming that a certain number (e.g., 10) of highest-ranked documents are more relevant than others.
- One bad word may spoil it all
  - Not all terms in documents in D_+ and D_- should be used in the formula.
  - E.g., for every document in D_+ and D_- only take the 10 terms with the highest IDF index.

Documents as sets

- Another, simpler, representation of documents: sets of terms
  - even less information retained: no term order, no term proximity, no term count.
  - “Bag of words” can refer to this representation (but is often associated to *multiset* representation, where elements retain count information)

Similarity of sets

- Jaccard coefficient: number of common elements (intersection), normalized by overall size (union):

$$r'(A,B) = \frac{|A \text{Ç} B|}{|A \text{È} B|}$$

- Similarity measure, ranging from 0 to 1.
- 0 if sets are disjoint, 1 if sets are equal.
- 1-*r'*(*A*,*B*) is a metric.

Approximating the Jaccard coefficient

- Even with the most efficient set representation, computing the Jaccard coefficient is linear in the set size.
- Computing Jaccard coefficients between all pairs of documents in a corpus has therefore a high time complexity:

$$O(m^2 |d|)$$

where m is the number of documents (millions?) and $|d|$ is the average document size (thousands?).

Approximating the Jaccard coefficient

Observation:

$$\frac{|A \cap B|}{|A \cup B|} = \Pr(x \in A \cap B \mid x \in A \cup B).$$

So we can approximate the Jaccard coefficient by picking random elements in the union and counting how many belong to both sets.

Approximating the Jaccard coefficient

- To do it efficiently: let π be a random permutation on *T* (the set of terms). Then:

$$t = \arg\min \pi(A \cup B)$$

  is a uniformly chosen term in the union.
- The term *t* also belongs to the intersection if and only if

$$\min \pi(A) = \min \pi(B).$$

Approximating the Jaccard coefficient

- Precompute N permutations $\pi_1,\ldots,\pi_N$ of term set;
- for all document ids $i=1,\ldots,m$ and for all permutations, compute

$$\mathrm{m}_{ik} = \min \rho_k(d_i);$$

- for all pair of documents (d_i,d_j), just let

$$r'(d_i,d_j) \approx \frac{\left|\{k=1,\ldots,N : m_{ik}=m_{jk}\}\right|}{N},$$

  i.e., the frequency of permutations that end up to the same minimum.

- Complexity is significantly reduced:

$$O\left(N\left(n+|d|+m^2\right)\right).$$