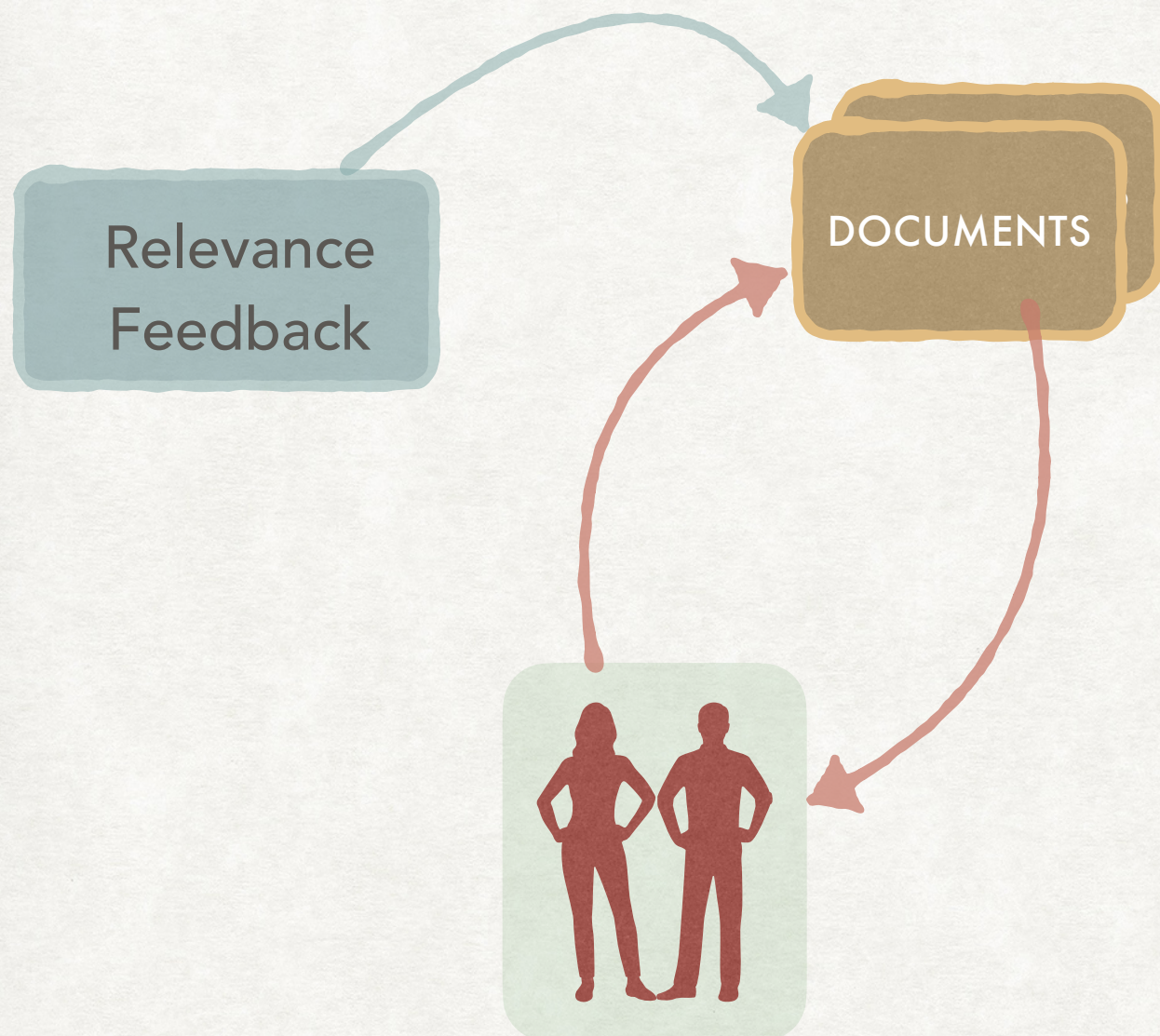


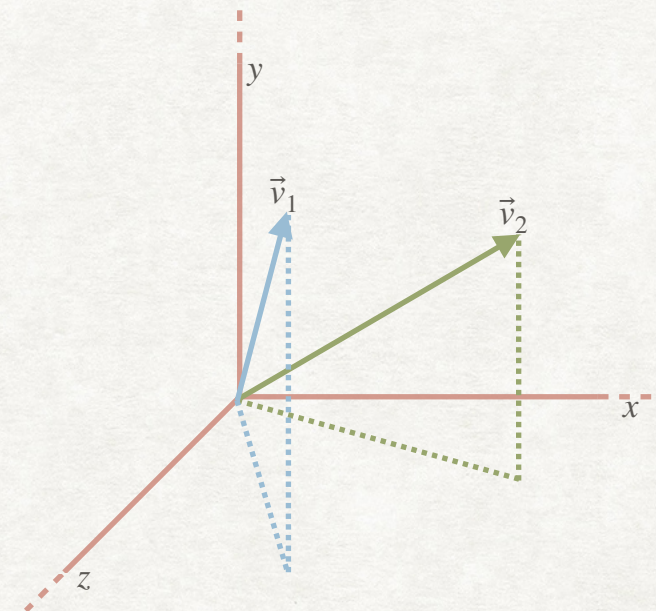
INFORMATION RETRIEVAL

Laura Nenzi
lnenzi@units.it

LECTURE OUTLINE



PRACTICAL PART
A PYTHON IMPLEMENTATION
OF A VECTOR SPACE MODEL



Vector Space
Model

THE VECTOR SPACE MODEL

VERY BRIEF RECAP

JUST TO REFRESH SOME BASIC NOTION AND FIX NOTATION

- In \mathbb{R}^n the Euclidean length of a vector $\vec{v} = (v_1, v_2, \dots, v_n)$ is

$$|\vec{v}| = \sqrt{\sum_{i=1}^n v_i^2}$$

- A vector is a unit vector if its length is one.
- The inner (or dot or scalar) product of two vectors

$\vec{v} = (v_1, v_2, \dots, v_n)$ and $\vec{u} = (u_1, u_2, \dots, u_n)$ is defined as $\sum_{i=1}^n v_i u_i$

DOCUMENTS AS VECTORS

THE START OF THE VECTOR SPACE REPRESENTATION

$$e_{\text{bart}} = (1, 0, 0, 0, 0)$$

$$e_{\text{box}} = (0, 1, 0, 0, 0)$$

$$e_{\text{cat}} = (0, 0, 1, 0, 0)$$

$$e_{\text{dog}} = (0, 0, 0, 1, 0)$$

$$e_{\text{drone}} = (0, 0, 0, 0, 1)$$

Each term is an element of the canonical base of \mathbb{R}^n with n the number of terms in the dictionary.

A document is a point in this n -dimensional space:

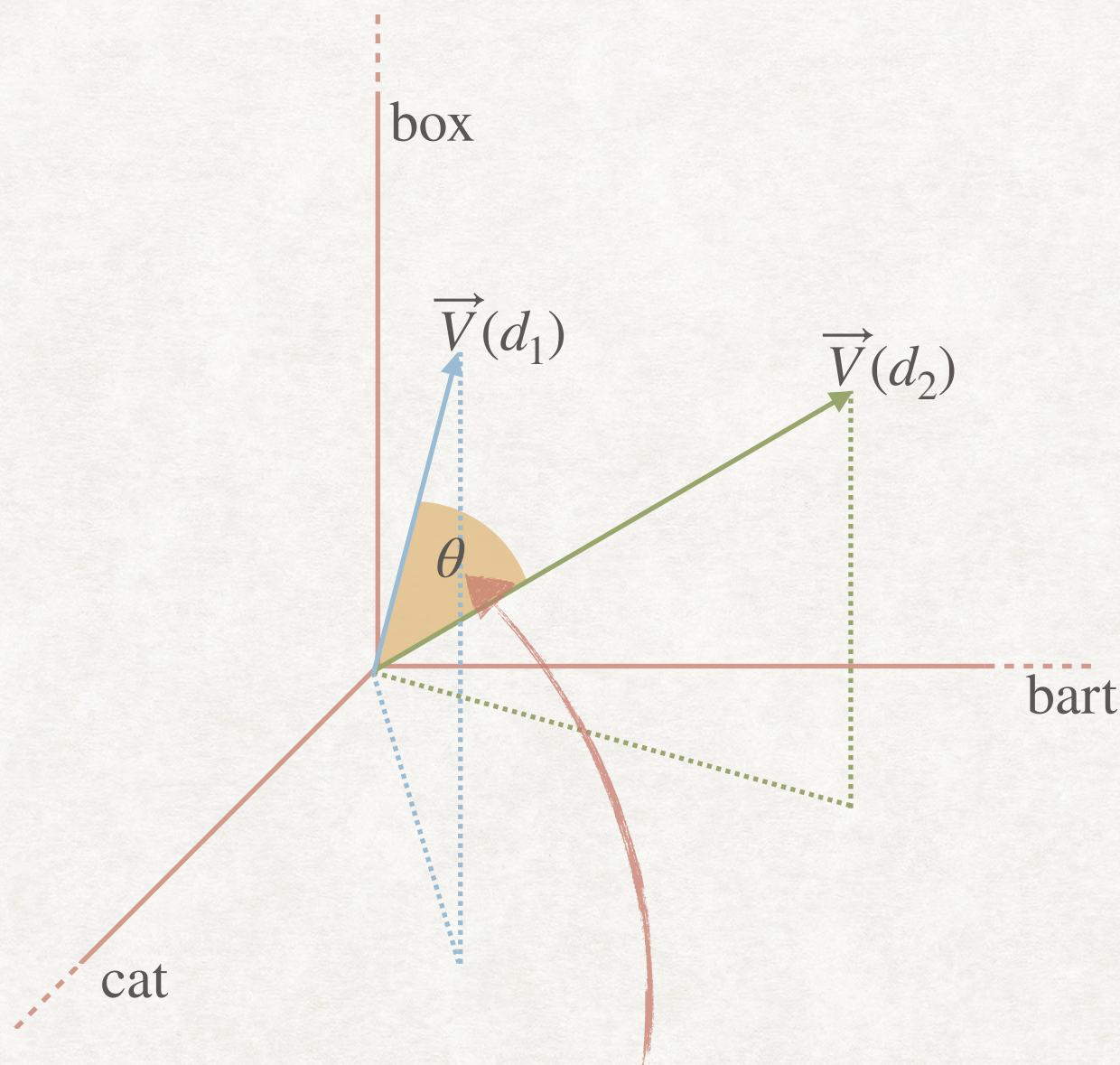
$$\vec{V}(d) = (0.6, 0.5, 0.1, 0, 0.9)$$

tf-idf_{cat,d}

We will limit ourselves to 3D visualisation due to the limits of the physical world

COSINE SIMILARITY

HOW TO COMPARE DOCUMENTS



The similarity is the cosine of this angle θ

We can compute the similarity of two documents by computing the *cosine similarity* between the two corresponding vectors:

$$\text{sim}(d_1, d_2) = \frac{\vec{V}(d_1) \cdot \vec{V}(d_2)}{|\vec{V}(d_1)| |\vec{V}(d_2)|}$$

Which represents the cosine of the angle formed by the two vectors

NORMALISING VECTORS

LOOKING AGAIN AT COSINE SIMILARITY

If we look again at cosine similarity we can see that we can replace a vector $\vec{V}(d)$ with the *unit vector* $\vec{v}(d)$:

$$\vec{v}(d) = \frac{\vec{V}(d)}{|\vec{V}(d)|}$$

In fact, since the angle formed by the vectors does not depend on the magnitude of the vectors, we can assume, without loss of generality, each document vector to be a unit vector.

QUERIES AS VECTORS

THE MISSING HALF OF THE REPRESENTATION

In addition to documents, also queries can be represented as vectors

Query: CAT

Vector: $(0,0,1,0,0)$

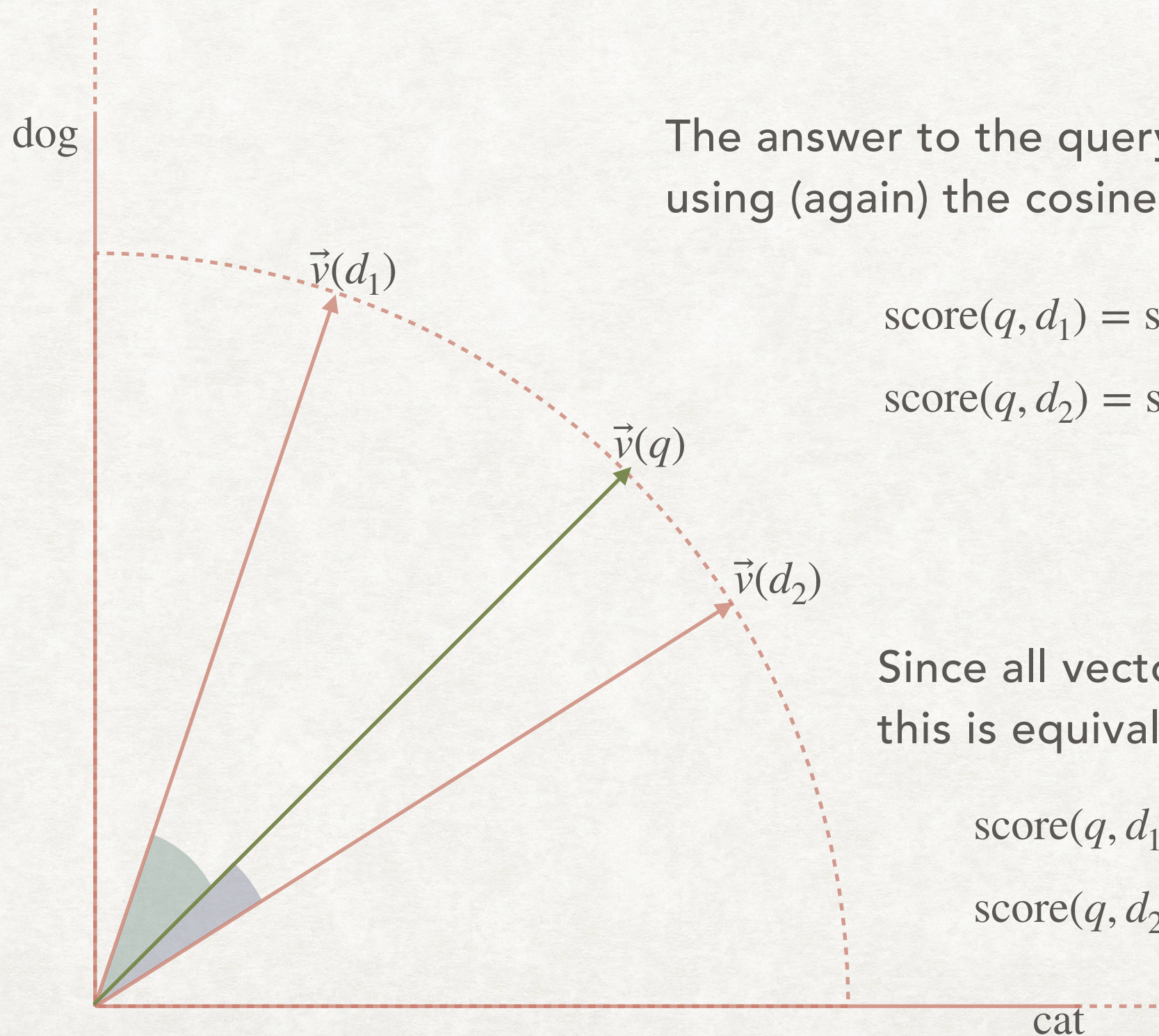
Query: CAT DOG

Vector: $(0,0,1/\sqrt{2},1/\sqrt{2},0)$

Each query is a unit vector
with the non-zero components
corresponding to the query terms

ANSWERING QUERIES

COSINE SIMILARITY (AGAIN)



The answer to the query can be computed using (again) the cosine similarity:

$$\text{score}(q, d_1) = \text{sim}(\vec{v}(q), \vec{v}(d_1))$$

$$\text{score}(q, d_2) = \text{sim}(\vec{v}(q), \vec{v}(d_2))$$

Since all vectors are unit vectors this is equivalent to:

$$\text{score}(q, d_1) = \vec{v}(q) \cdot \vec{v}(d_1)$$

$$\text{score}(q, d_2) = \vec{v}(q) \cdot \vec{v}(d_2)$$

VECTOR SPACE MODEL

CONSIDERATIONS

- The fact that we compute a similarity score means that we have a ranking of documents; we can retrieve the K most relevant documents.
- A document might have a non-zero similarity score even if not all terms are present: the matching is not exact like in the Boolean model.
- Even if we have used tf-idf to define the document vectors, any other measure might be used.
- Notice that we cannot exclude (for now) the computation of the cosine similarity for each document in the collection!

COMPUTING SIMILARITY EFFICIENTLY

A FEW INITIAL CONSIDERATIONS

THE LOW-HANGING FRUITS

- We can have an inverted index in which each term has an associated idf_t value (since it depends only on the term).
- Each posting will have the term frequency $\text{tf}_{t,d}$ associated to it (since it depends on both the term and the document).
- We can then compute the score of each document while traversing the posting lists.
- If a DocID does not appear in the posting list of any query term its score is zero.

COMPUTING TF-IDF



TOP K DOCUMENT RETRIEVAL

- To retrieve the K highest scoring documents we can use a *heap* data structure, which is more efficient than sorting all documents.
- The heap maintains the invariant that the root is always the minimum of the K highest scoring documents.
- Heaps can insert and delete elements in $O(\log K)$ time, making it much more efficient than keeping all documents in a sorted list, especially when the total number of documents is much larger than K.

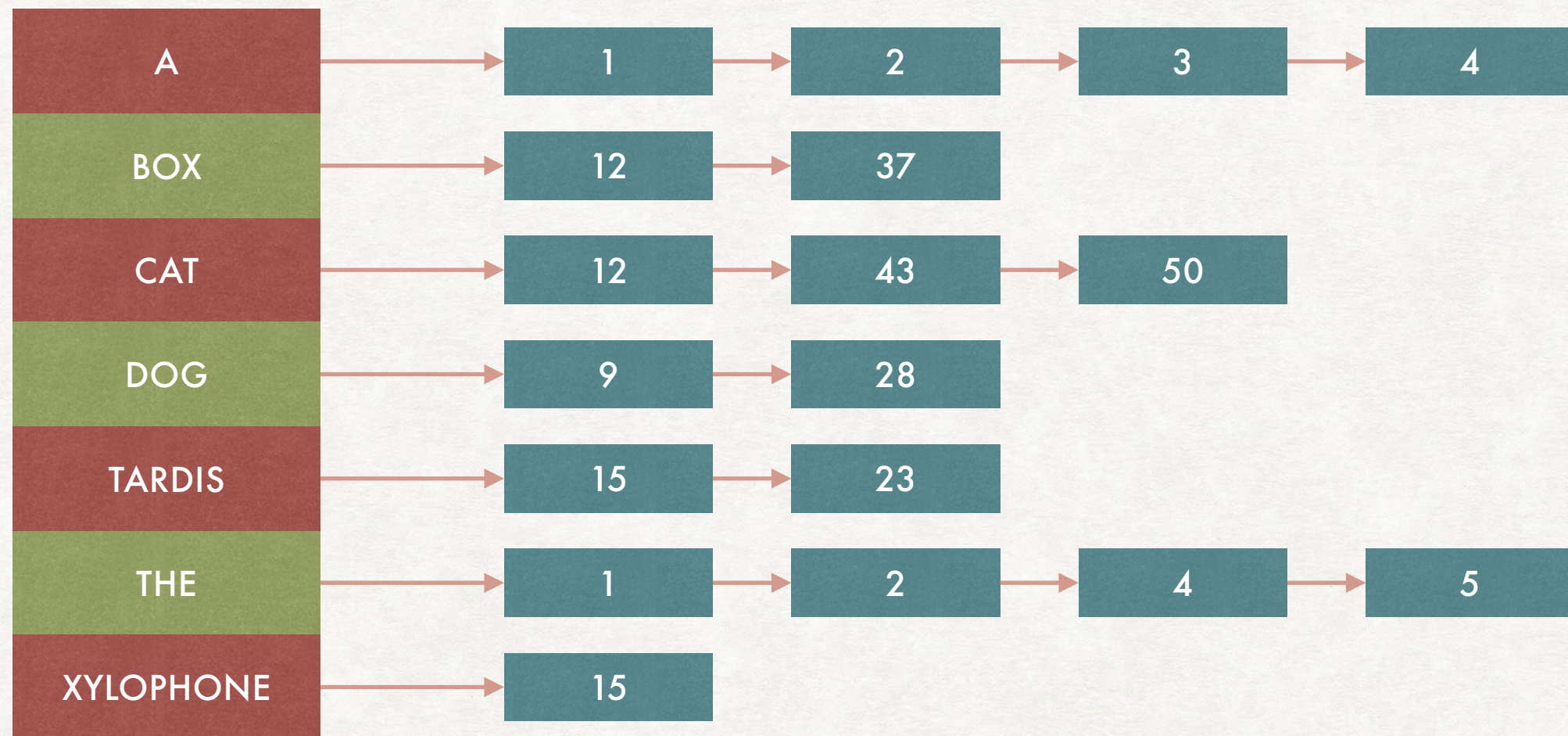
INEXACT TOP K DOCUMENT RETRIEVAL

BEING FAST AND “WRONG”

- Sometimes it is more important to be efficient than to retrieve exactly the K highest scoring documents.
- We want to retrieve K documents that are *likely* to be among the K highest scored.
- Notice that the similarity score is a proxy of the relevance of a document to a query, so we already have some “approximation”.
- The main idea to perform an inexact retrieval is:
 - Find a subset A of the documents that is both small and likely to contain documents with scores near to the K highest ranking.
 - Return the K highest ranked documents in A .

INDEX ELIMINATION

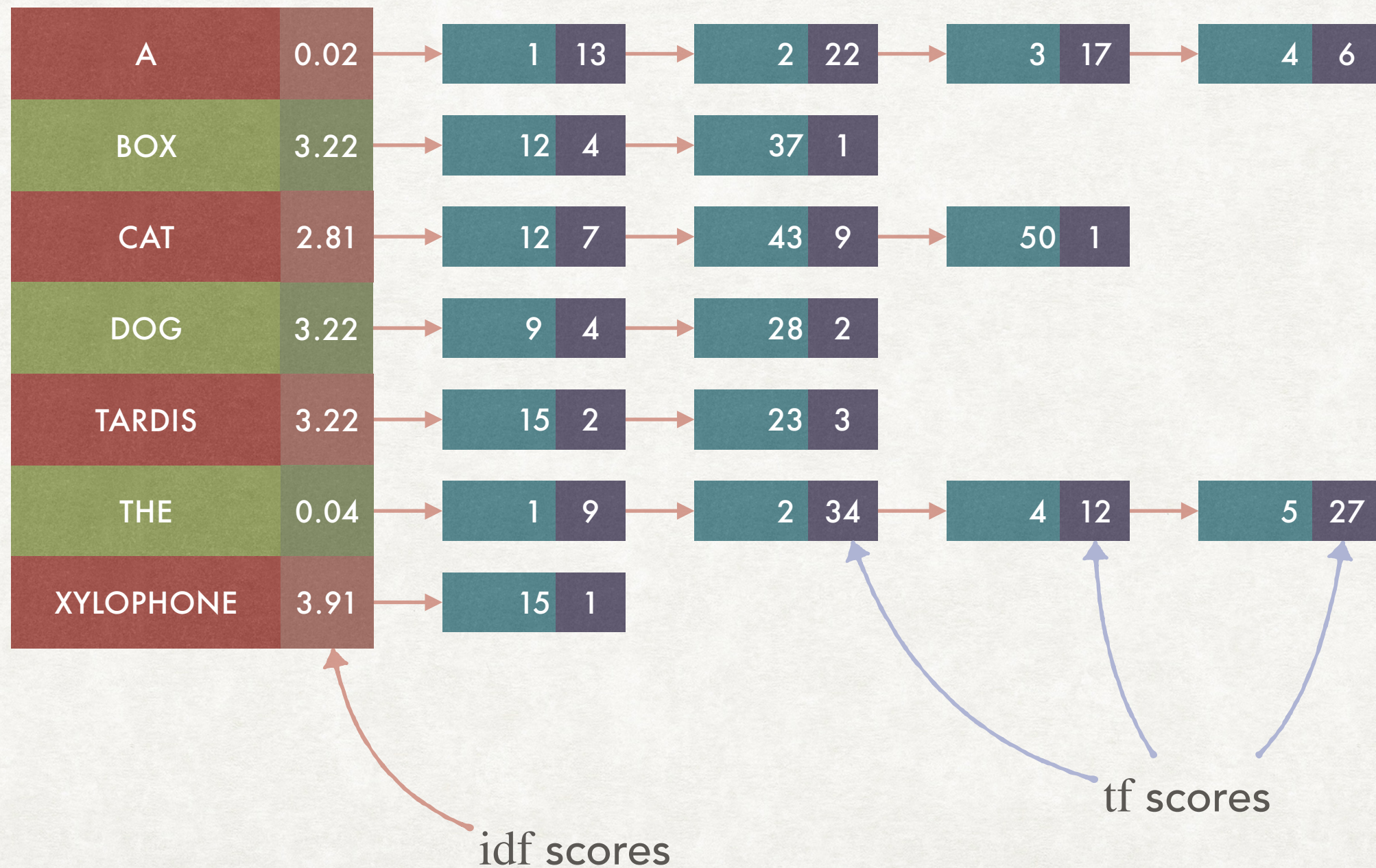
HOW TO IGNORE SOME TERMS



standard inverted index

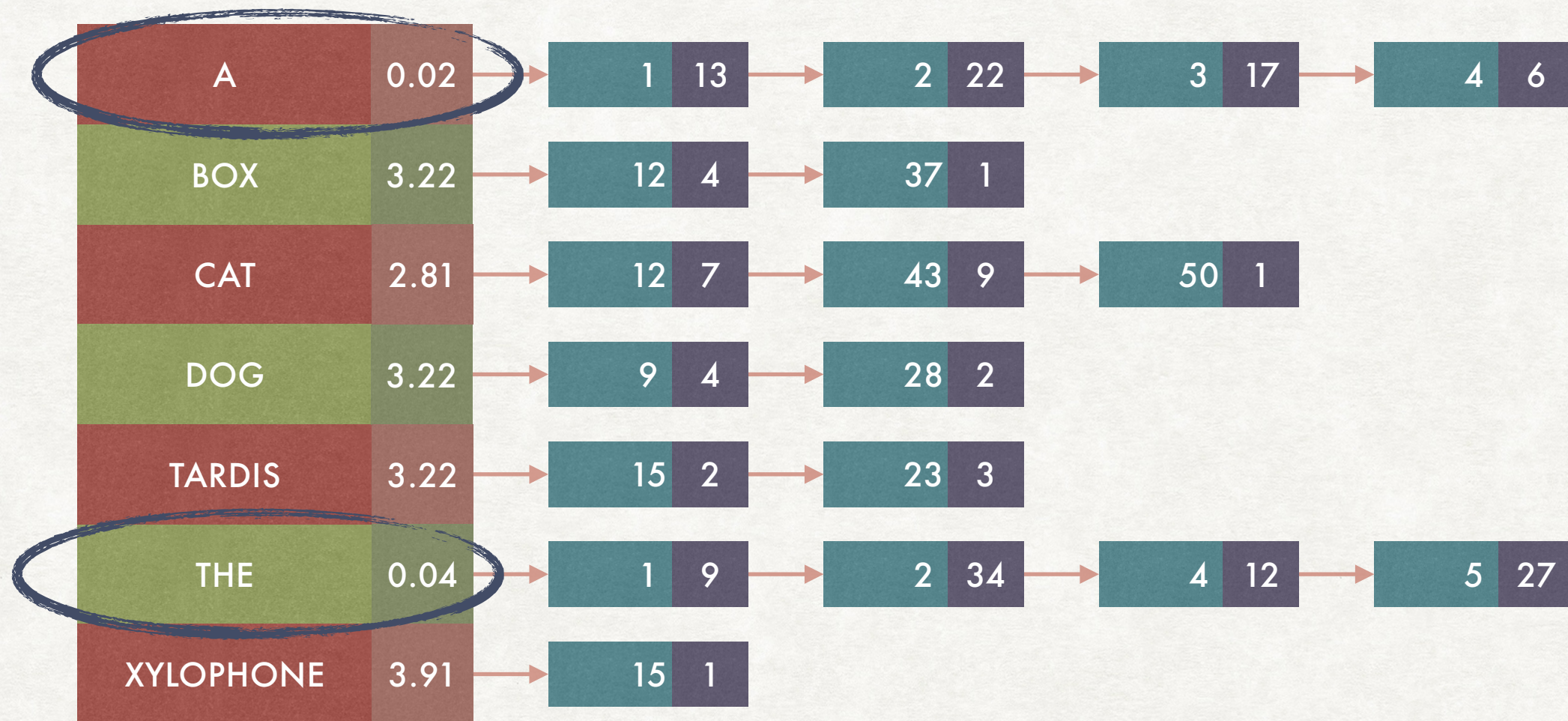
INDEX ELIMINATION

HOW TO IGNORE SOME TERMS



INDEX ELIMINATION

HOW TO IGNORE SOME TERMS



We can remove terms with very low idf score from the search:
they are like "stop words" with very long postings list

INDEX ELIMINATION

HOW TO IGNORE SOME TERMS

- By removing terms with low idf value we can only work with relatively shorter lists.
- The cutoff value can be adapted according to the other terms present in the query.
- We can also only consider documents in which most or all the query terms appears...
- ...but a problem might be that we do not have at least K documents matching all query terms.

CHAMPION LISTS

OR "TOP DOCS"

- Keep an additional pre-computed list for each term containing only the r highest-scoring documents (usually $r > K$).
- These additional lists are known as *champion lists*, *fancy lists*, or *top docs*.
- We compute the union of the champion lists of all terms in the query, obtaining a set A of documents.
- We find the K highest ranked documents in A (computing cosine similarity).
- Problem: we might have too few documents if K is not known until the query is performed. Possibility setting an r higher for rarer terms.

STATIC QUALITY SCORES

ADDING A PRE-COMPUTABLE SCORE TO DOCUMENTS

- In some cases we might want to add a score to a document that is independent from the query: a **static quality score**, denoted by $g(d) \in [0,1]$.
- Example: good reviews by users might “push” a document higher in the scoring.
- We need to combine $g(d)$ with the scoring given by the query, a simple possibility is a linear combination:
$$\text{score}(q, d) = g(d) + \vec{v}(d) \cdot \vec{v}(q).$$
- We can also sort posting list by $g(d) + \text{idf}_{t,d}$ to process documents more likely to have high scores first.

IMPACT ORDERING

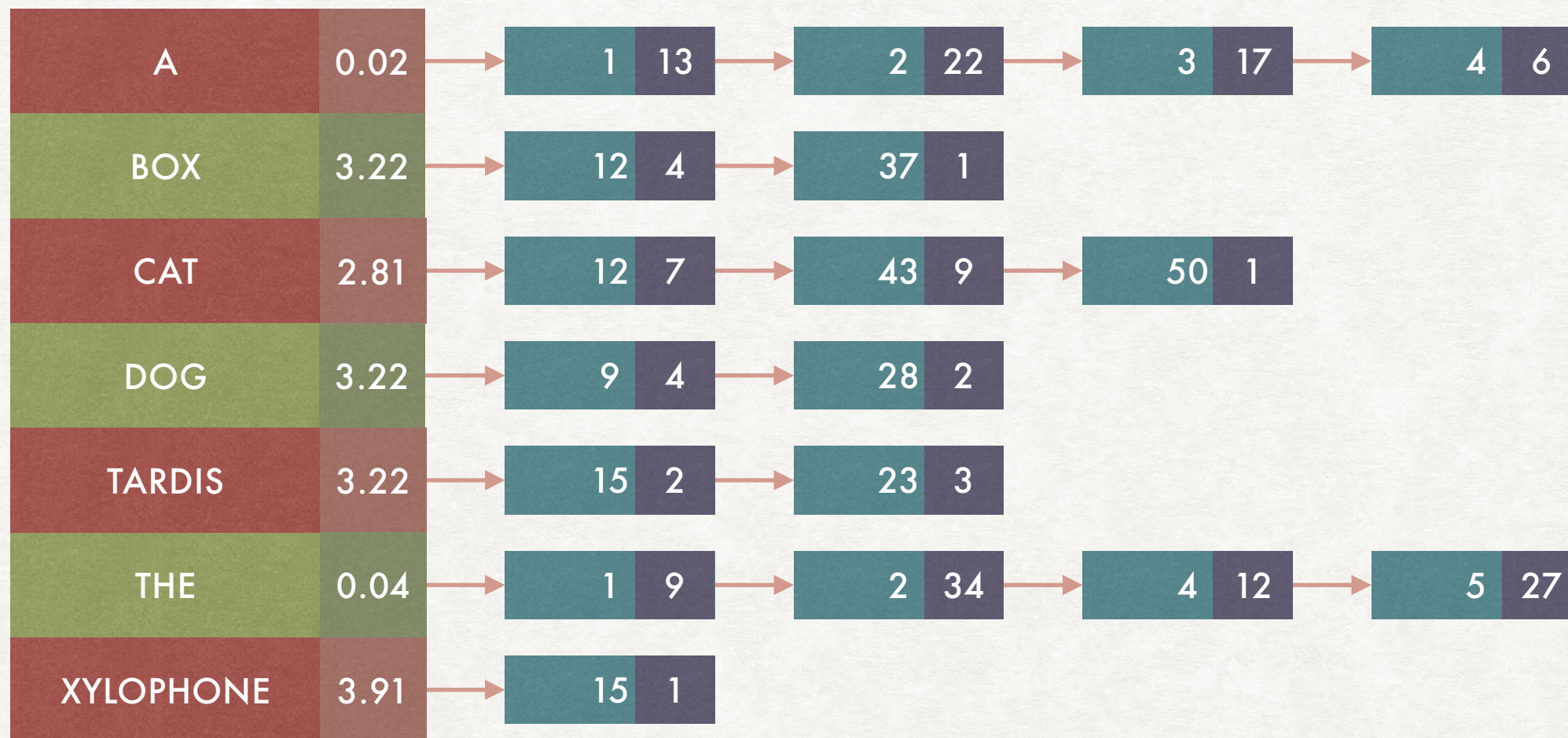
SORTING POSTING LISTS NOT BY DOCID

- We only want to compute scores for docs for which $tf_{t,d}$ is high enough
- Idea: Order the documents by decreasing $tf_{t,d}$. In this way the documents which will obtain the highest scoring will be processed first.
- If the $tf_{t,d}$ value drops below a threshold, then we can stop.

IMPACT ORDERING

SORTING POSTING LISTS NOT BY DOCID

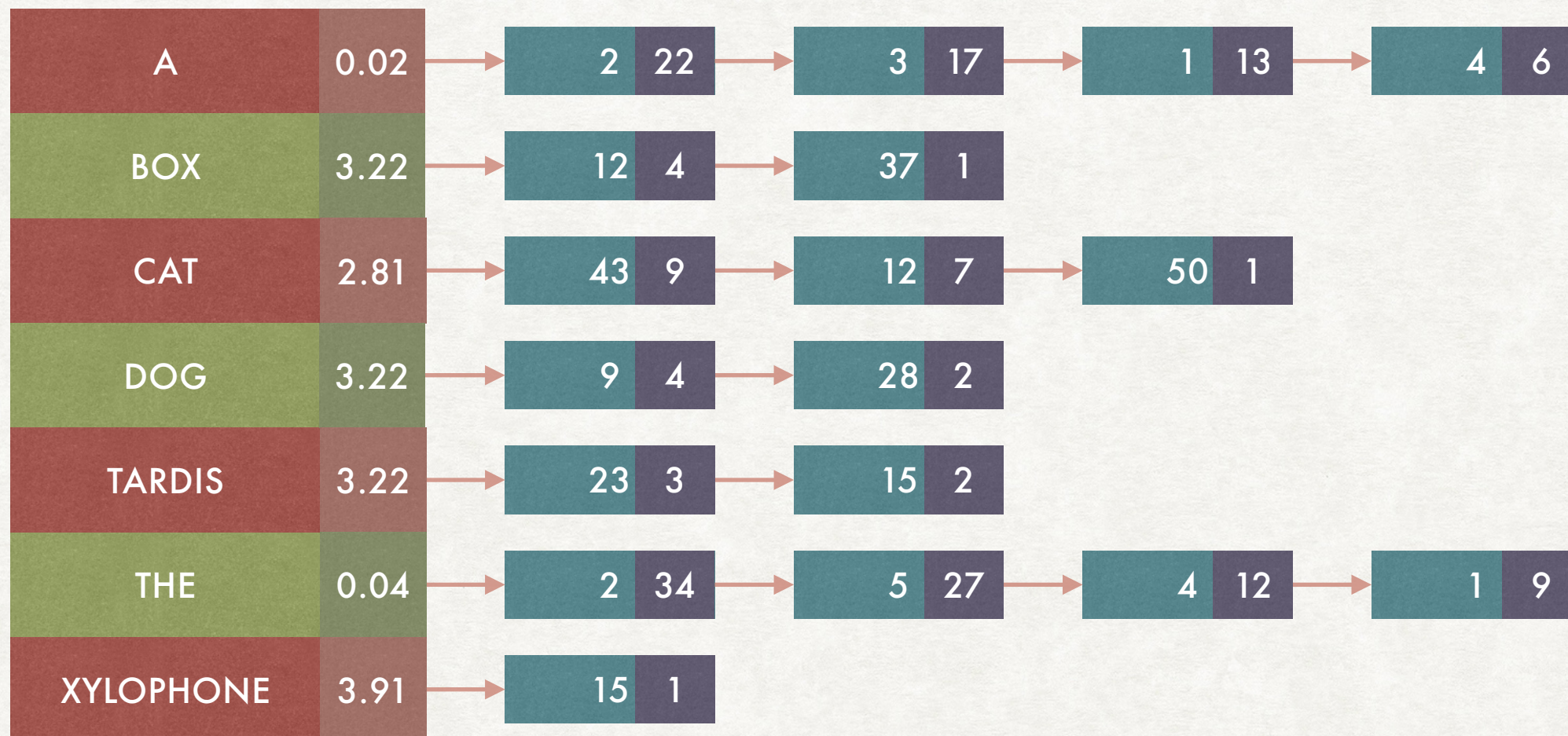
From this...



IMPACT ORDERING

SORTING POSTING LISTS NOT BY DOCID

...to this



CLUSTER PRUNING

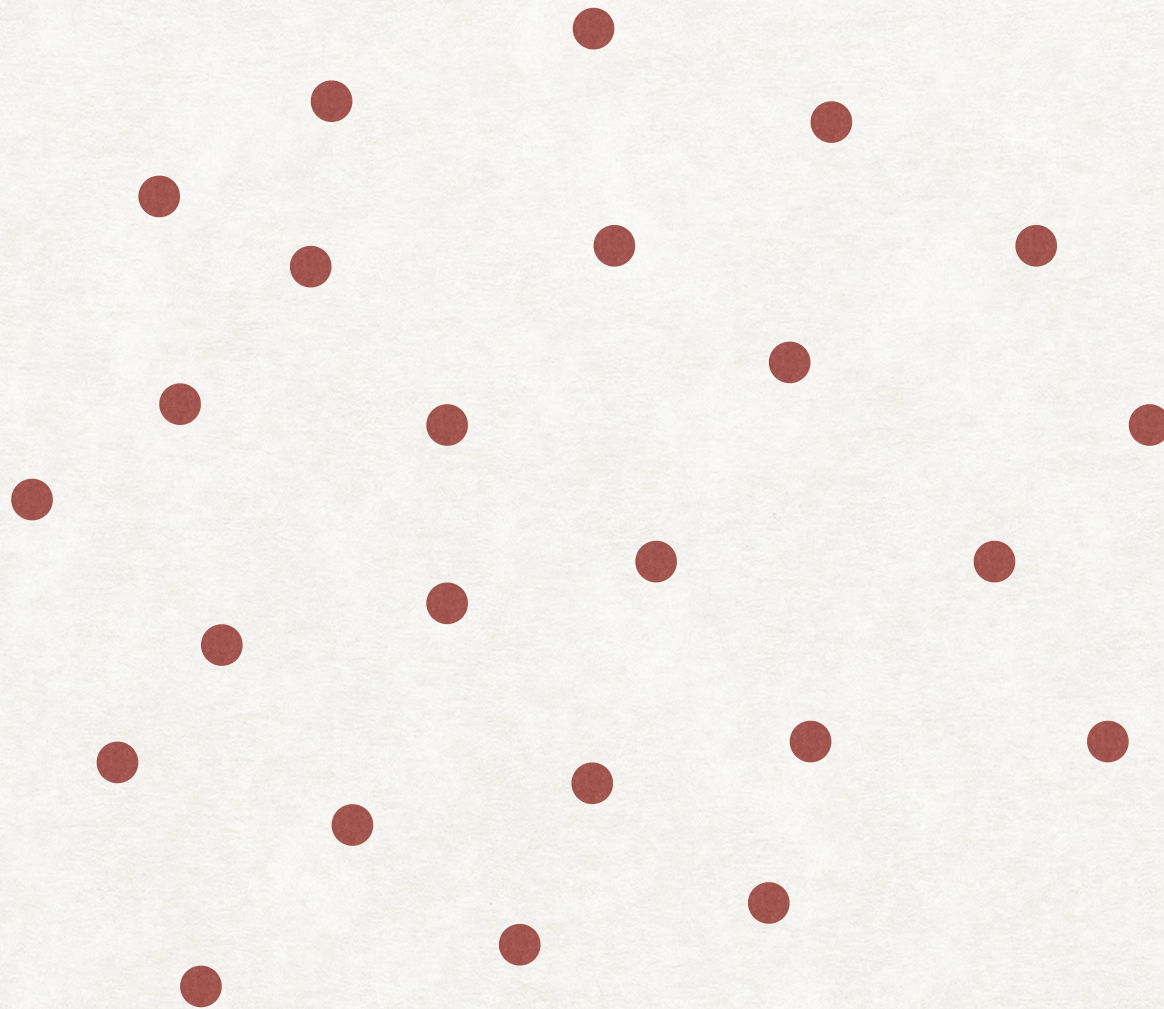
SEARCHING ONLY INSIDE A CLUSTER

- With N document, $M = \sqrt{N}$ are randomly selected as *leaders*. Each leader identifies a cluster of documents.
- For each of the remaining documents (*followers*), we find the most similar among the M documents selected and we add it to the corresponding cluster.
- For a query q we find the document among the M leaders that is most similar to it.
- The K highest ranked documents are selected among the ones in the cluster of the selected leader.

CLUSTER PRUNING

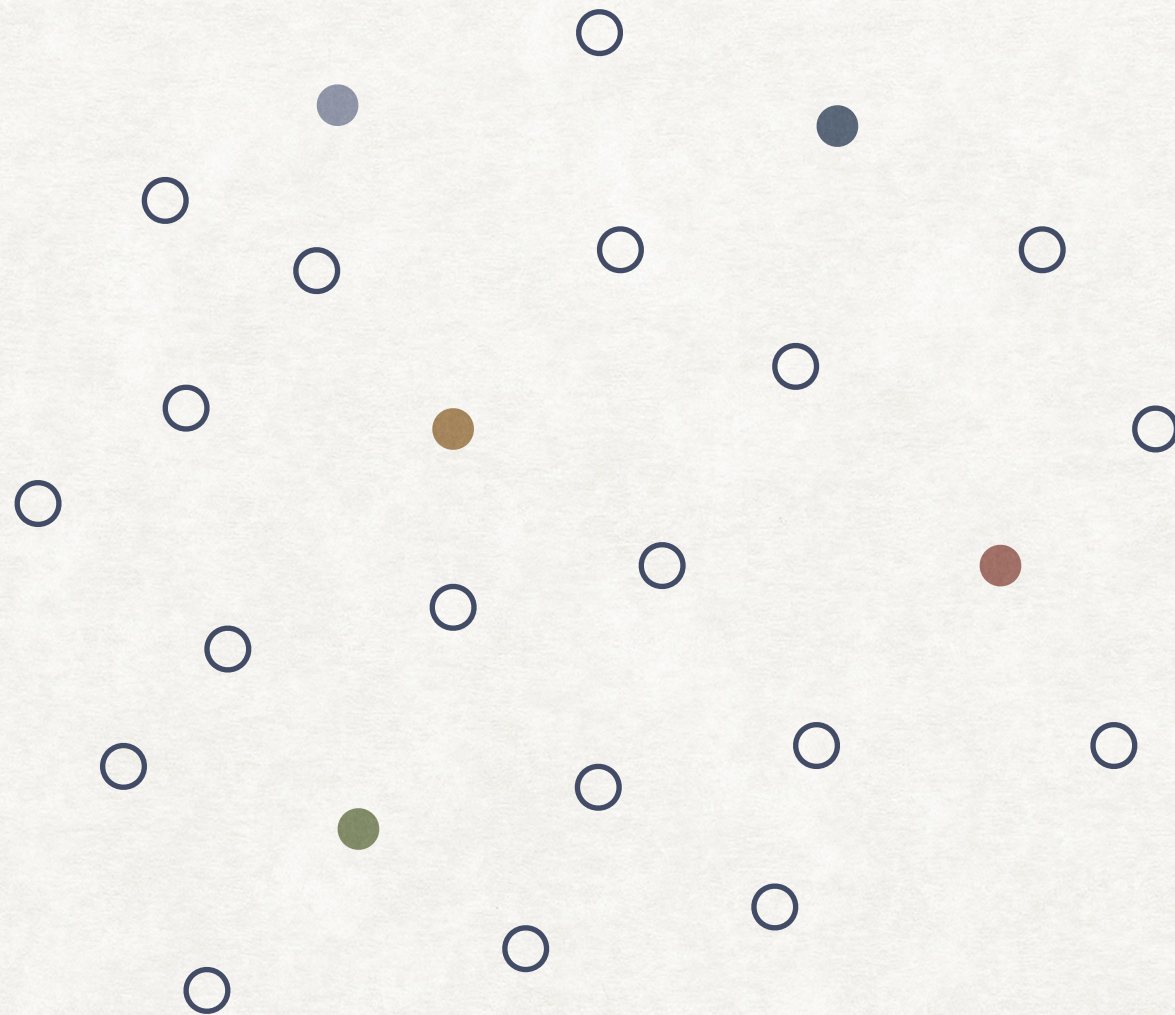
AN EXAMPLE

Documents represented
as points in space



CLUSTER PRUNING

AN EXAMPLE

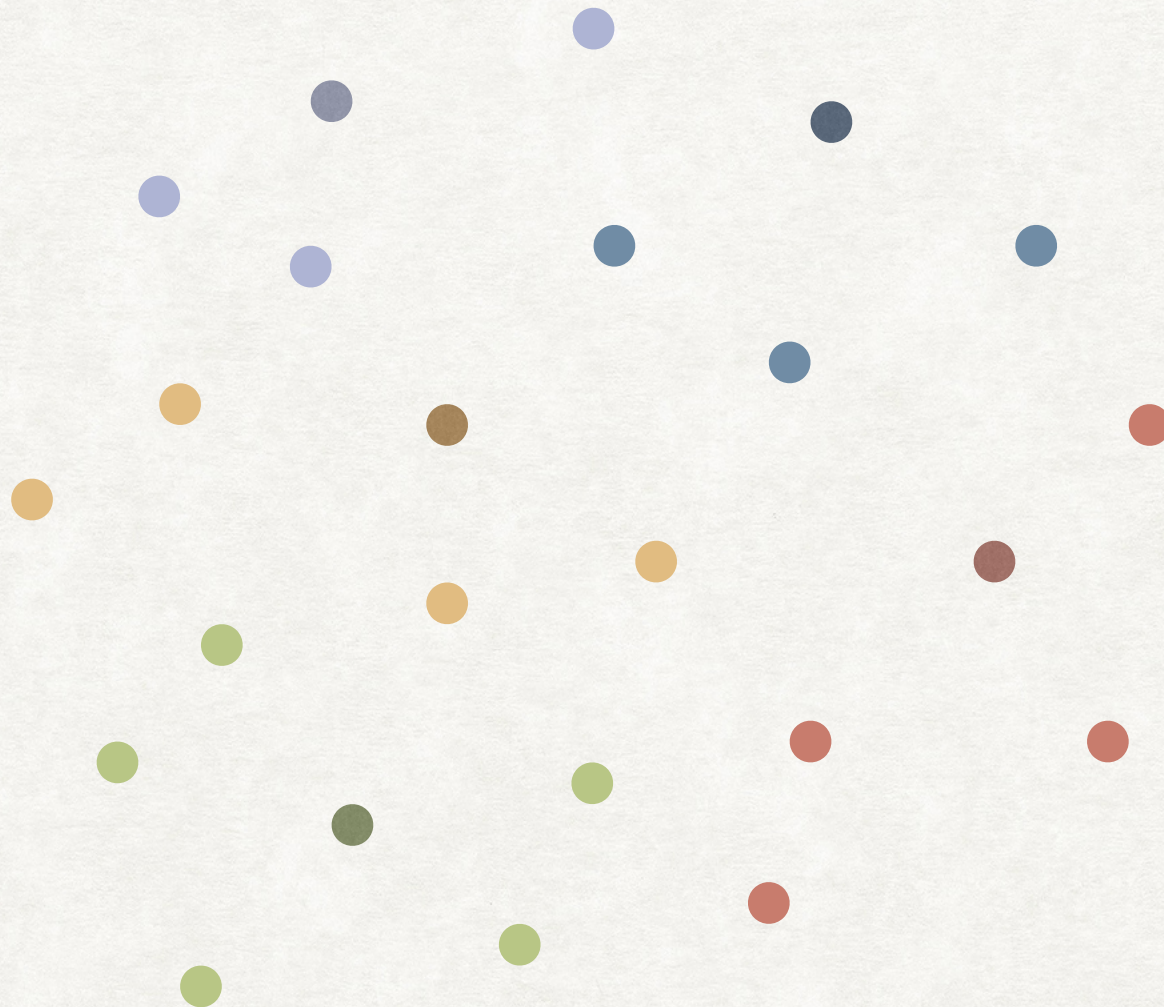


Documents represented
as points in space

Selection of the leaders

CLUSTER PRUNING

AN EXAMPLE



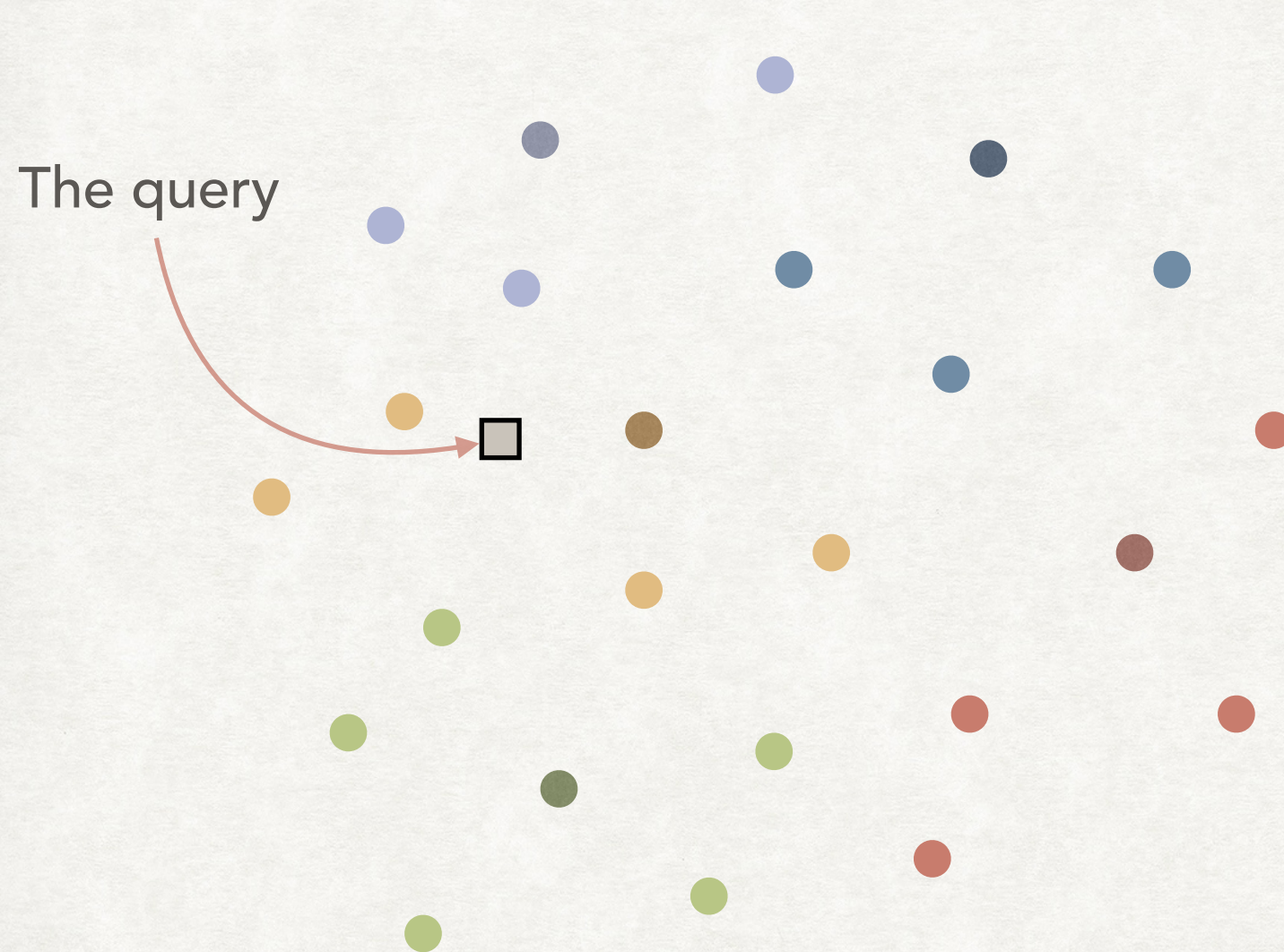
Documents represented
as points in space

Selection of the leaders

Assigning documents
to clusters

CLUSTER PRUNING

AN EXAMPLE



Documents represented
as points in space

Selection of the leaders

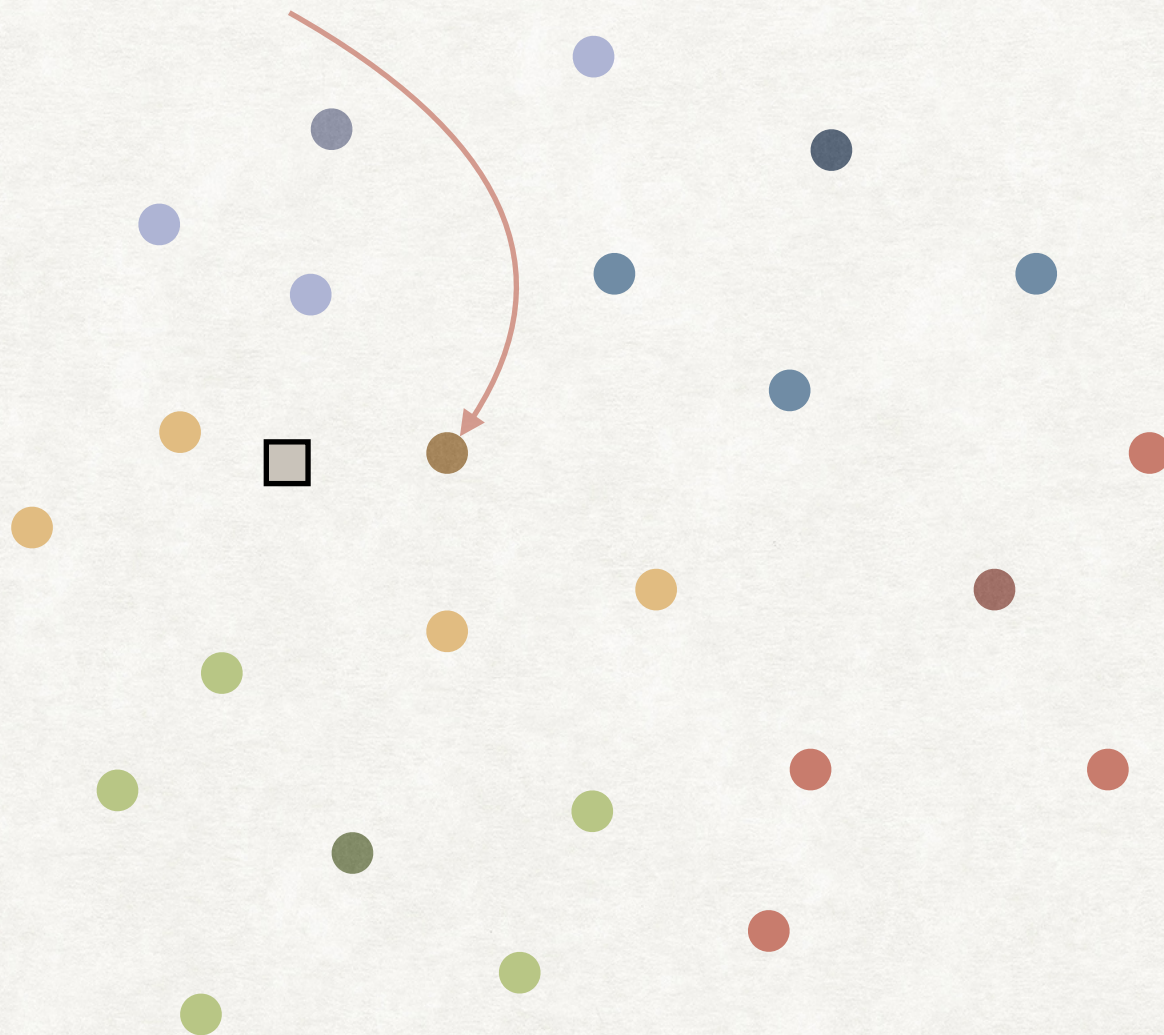
Assigning documents
to clusters

A query arrives

CLUSTER PRUNING

AN EXAMPLE

Nearest leader



Documents represented
as points in space

Selection of the leaders

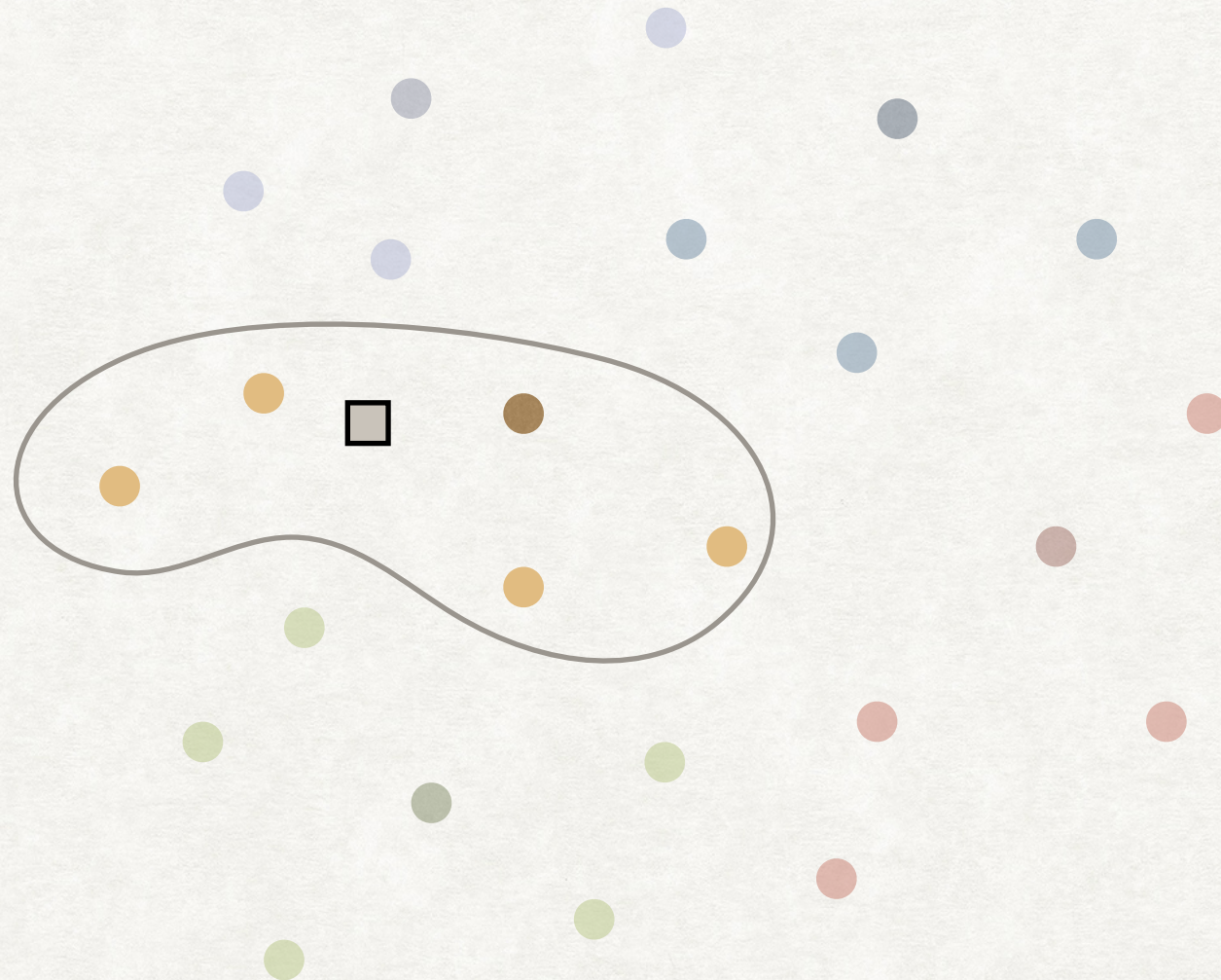
Assigning documents
to clusters

A query arrives

The nearest leader
is found

CLUSTER PRUNING

AN EXAMPLE



Documents represented
as points in space

Selection of the leaders

Assigning documents
to clusters

A query arrives

The nearest leader
is found

The similarity is computed
only in one cluster

CLUSTER PRUNING

ADDITIONAL CONSIDERATIONS

- The selection of \sqrt{N} leaders randomly likely reflects the distribution of documents in the vector space: the most crowded regions will have more leaders.
- A variant more likely to return the “real” K highest ranked document is the following:
 - When creating clusters, each document is associated to b_1 leaders (i.e., it is part of more than one cluster).
 - When a query is received the clusters of the b_2 nearest leaders are considered.

RELEVANCE FEEDBACK

WHAT IS RELEVANCE FEEDBACK

RECEIVING FEEDBACK FROM THE USER

- The main idea is to involve the user in giving feedback on the initial set of results:
- The user issues a query.
- The system returns an initial set of results.
- The user decides which results are relevant and which are not.
- The system computes a new set of results based on the feedback received by the user.
- If necessary, repeat.

WHAT RELEVANCE FEEDBACK CAN SOLVE

AND WHAT IT CANNOT SOLVE

- Relevance feedback can help the user in refining the query without having him/her reformulate it manually.
- It is a *local method*, where the initial query is modified, in contrast to *global methods* that change the wording of the query (like spelling correction).
- Relevance feedback can be ineffective when in the case of
 - Misspelling (but we have seen spelling correction techniques).
 - Searching documents in another language.
 - Vocabulary mismatch between the user and the collection.

THE ROCCHIO ALGORITHM

FEEDBACK FOR THE VECTOR SPACE MODEL

- It is possible to introduce relevance feedback in the vector space model
- We will see the Rocchio Algorithm (1971)
- It was introduced in the SMART (*System for the Mechanical Analysis and Retrieval of Text*) information retrieval system...
- ...which is also where the vector space model was firstly developed

ROCCHIO ALGORITHM: MAIN IDEA

MOVING THE QUERY VECTOR



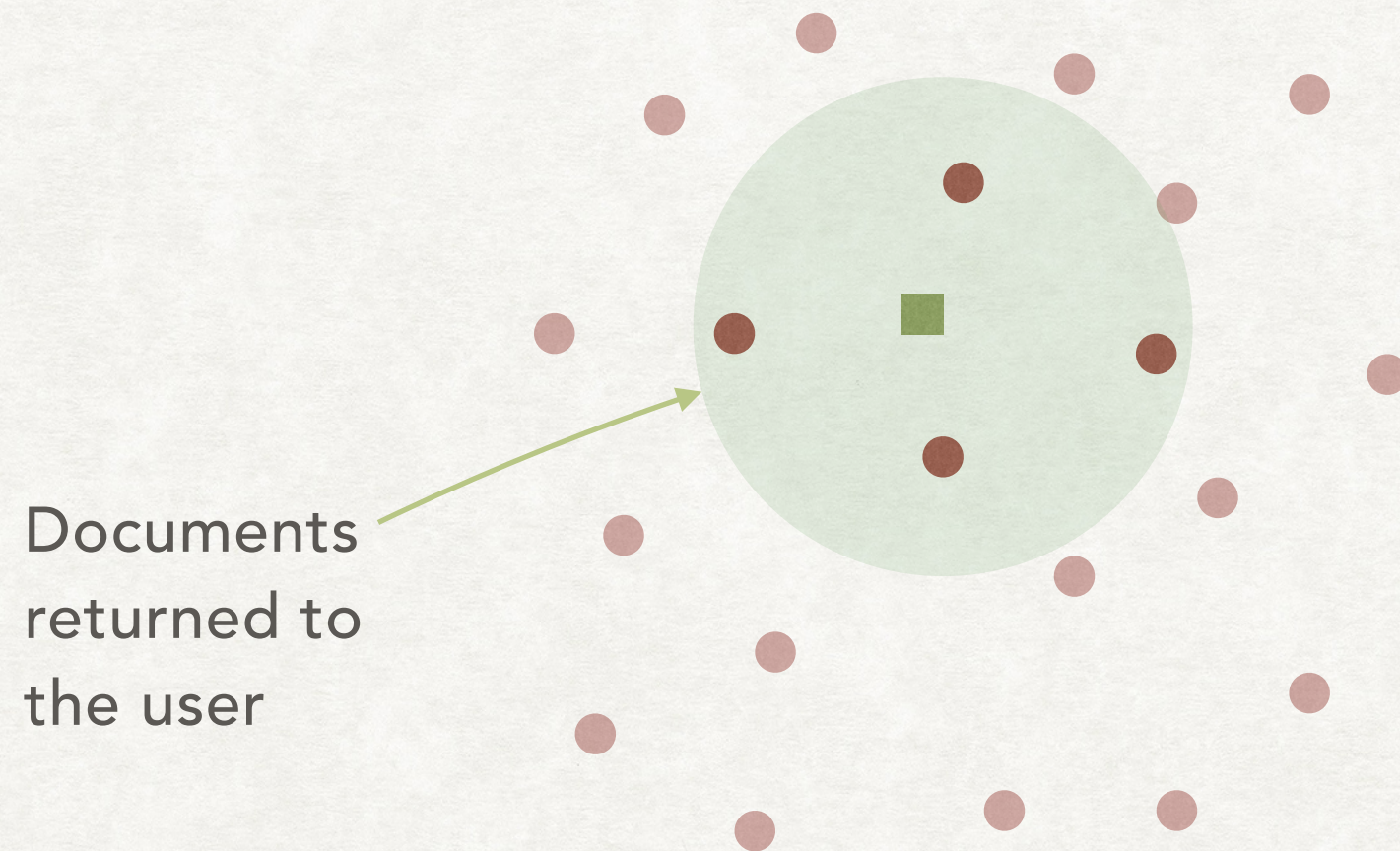
ROCCHIO ALGORITHM: MAIN IDEA

MOVING THE QUERY VECTOR



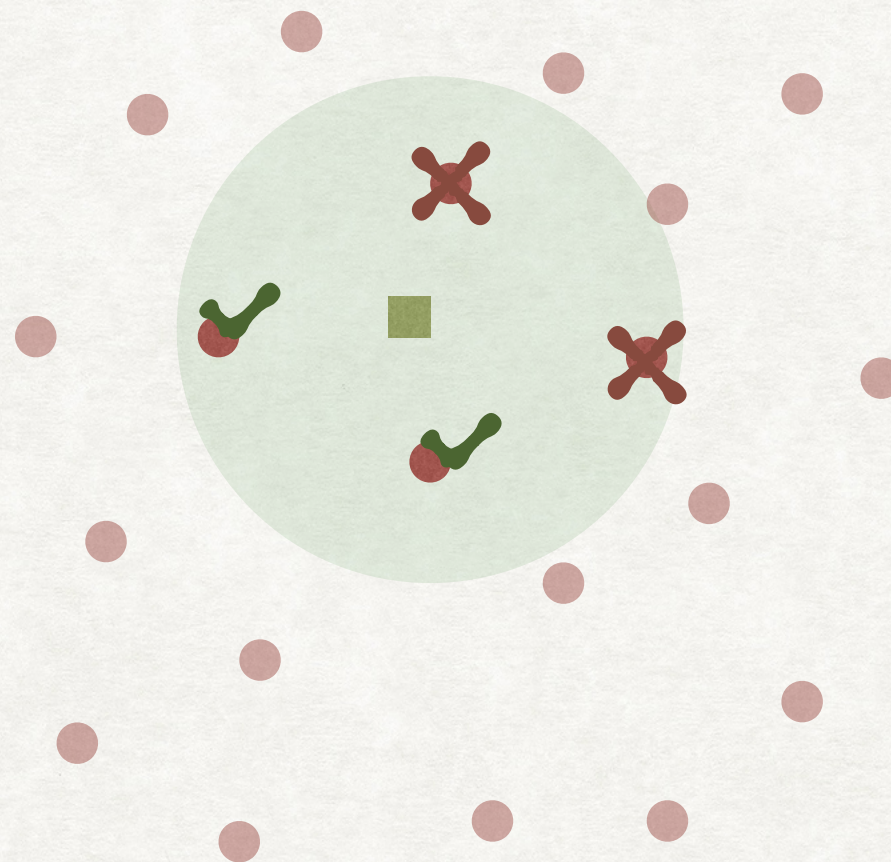
ROCCHIO ALGORITHM: MAIN IDEA

MOVING THE QUERY VECTOR



ROCCHIO ALGORITHM: MAIN IDEA

MOVING THE QUERY VECTOR



Feedback from the user

ROCCHIO ALGORITHM: MAIN IDEA

MOVING THE QUERY VECTOR



ROCCHIO ALGORITHM: THEORY

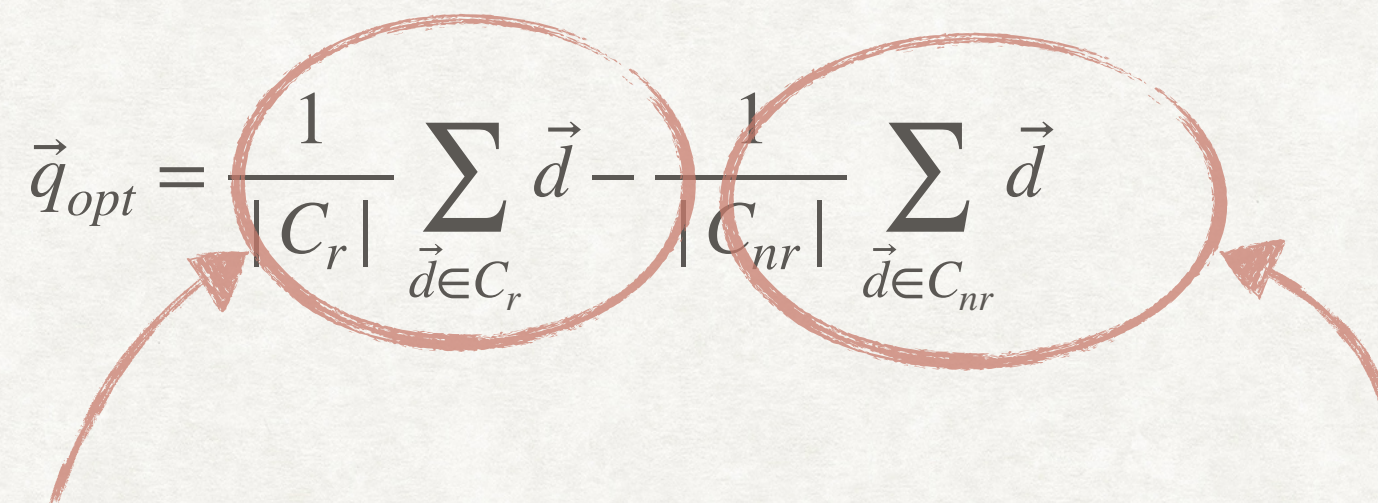
- The user gives us two sets of documents:
 - The relevant documents C_r
 - The non-relevant documents C_{nr}
- We want to maximise the similarity of the query with the set of relevant documents...
- ...while minimising it with respect to the set of non-relevant documents.

ROCCHIO ALGORITHM: THEORY

This can be formalised as defining the *optimal* query \vec{q}_{opt} as:

$$\vec{q}_{opt} = \arg \max_{\vec{q}} [\text{sim}(\vec{q}, C_r) - \text{sim}(\vec{q}, C_{nr})]$$

If we use cosine similarity, we can reformulate the definition as:

$$\vec{q}_{opt} = \frac{1}{|C_r|} \sum_{\vec{d} \in C_r} \vec{d} - \frac{1}{|C_{nr}|} \sum_{\vec{d} \in C_{nr}} \vec{d}$$


Centroid of
relevant documents

Centroid of
non-relevant documents

ROCCHIO ALGORITHM

However, we usually do not have knowledge of the relevance of *all* documents in the system. Instead we have:

- a set D_r of *known relevant* documents
- a set D_{nr} of *known non-relevant* documents

We also have the original query \vec{q}_0 performed by the user.

We can perform a linear combination of:

- The centroid of D_r
- The centroid of D_{nr}
- The original query \vec{q}_0

ROCCHIO ALGORITHM

In the Rocchio algorithm the query is updated as follows:

$$\vec{q}_m = \alpha \vec{q}_0 + \beta \frac{1}{|D_r|} \sum_{\vec{d} \in D_r} \vec{d} - \gamma \frac{1}{|D_{nr}|} \sum_{\vec{d} \in D_{nr}} \vec{d}$$

The diagram shows the Rocchio algorithm formula with three red hand-drawn circles highlighting specific parts. The first circle is around the term $\alpha \vec{q}_0$, with an arrow pointing from the text 'Original query' below it. The second circle is around the term $\beta \frac{1}{|D_r|} \sum_{\vec{d} \in D_r} \vec{d}$, with an arrow pointing from the text 'Centroid of the known relevant documents' below it. The third circle is around the term $-\gamma \frac{1}{|D_{nr}|} \sum_{\vec{d} \in D_{nr}} \vec{d}$, with an arrow pointing from the text 'Centroid of the known non-relevant documents' below it.

Original query

Centroid of the known relevant documents

Centroid of the known non-relevant documents

If one of the components of \vec{q}_m is less than 0, we set it to 0
(all documents have non-negative coordinates)

ROCCHIO ALGORITHM

SELECTING THE WEIGHTS

- We need to select reasonable weights α , β , and γ :
- Positive feedback is more valuable than negative feedback, so usually $\gamma < \beta$.
- Reasonable values might be $\alpha = 1$, $\beta = 0.75$, and $\gamma = 0.15$.
- It is also possible to have only positive feedback with $\gamma = 0$.

PSEUDO-RELEVANCE FEEDBACK

NOW WITHOUT THE USER

- It is possible to perform a relevance feedback without the user...
- ...even before he/she receives the results of the first query.
- Perform the query \vec{q} as usual.
- Consider the first k retrieved documents in the ranking as relevant.
- Perform relevance feedback using this assumption.
- Might provide better results, but the retrieved documents might drift the query in an unwanted direction.