INFORMATION RETRIEVAL

Luca Manzoni
lmanzoni@units.it

Lecture 6

LECTURE OUTLINE

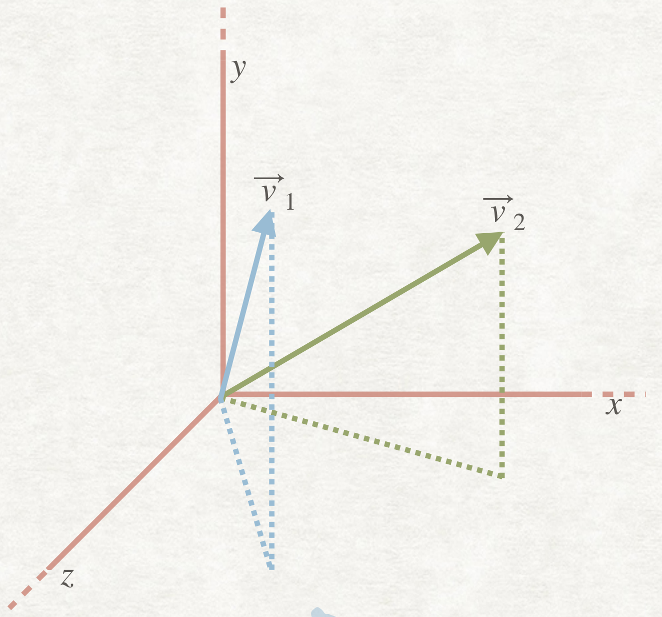
*MADE WITH ALIEN TECHNOLOGY

1) DOCUMENT 1

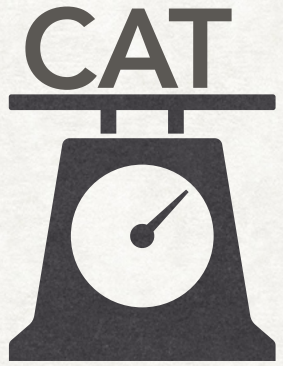
2) DOCUMENT 2

3) DOCUMENT 3

Ranked Retrieval



TF-IDF



Vector Space Model

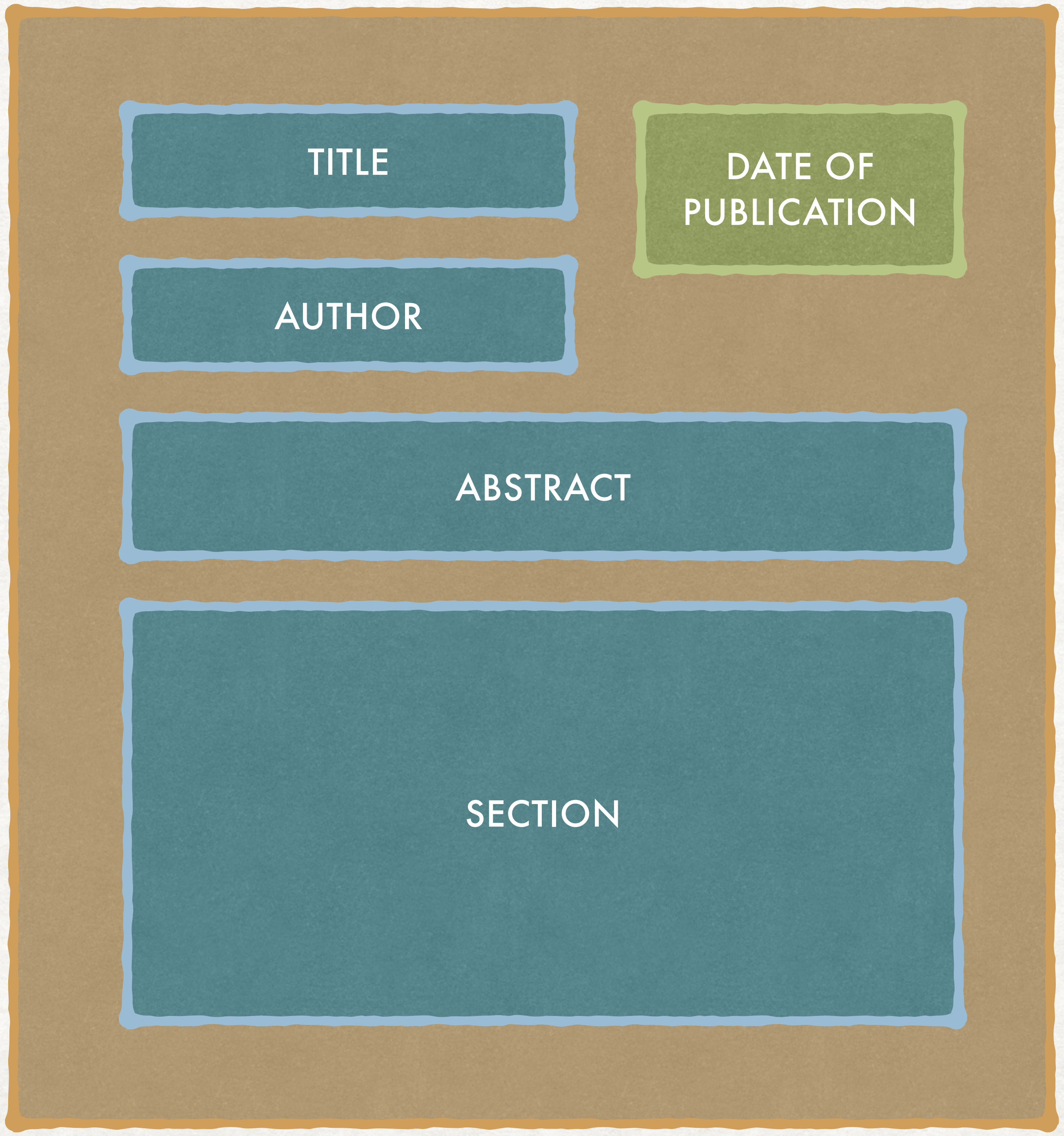
RANKED RETRIEVAL

MOTIVATIONS

- Until now we have returned all documents matching a Boolean query as a set.
- If many documents are returned then it might be important to rank them according to how relevant they are.
- A first way of ranking them is to “split” a document according to some structure and then weight different zones in different ways.
- We will then see how we can extend the idea of adding weights also to the terms of a document.

DOCUMENT STRUCTURE

METADATA, FIELDS, AND ZONES



- A text may have associated **metadata**.
- Some of them can be **fields**, with a set of values that can be finite, like publication dates.
- Others might be zones, arbitrary areas of free-form text (e.g., abstract, section, etc.).

PARAMETRIC INDEXES

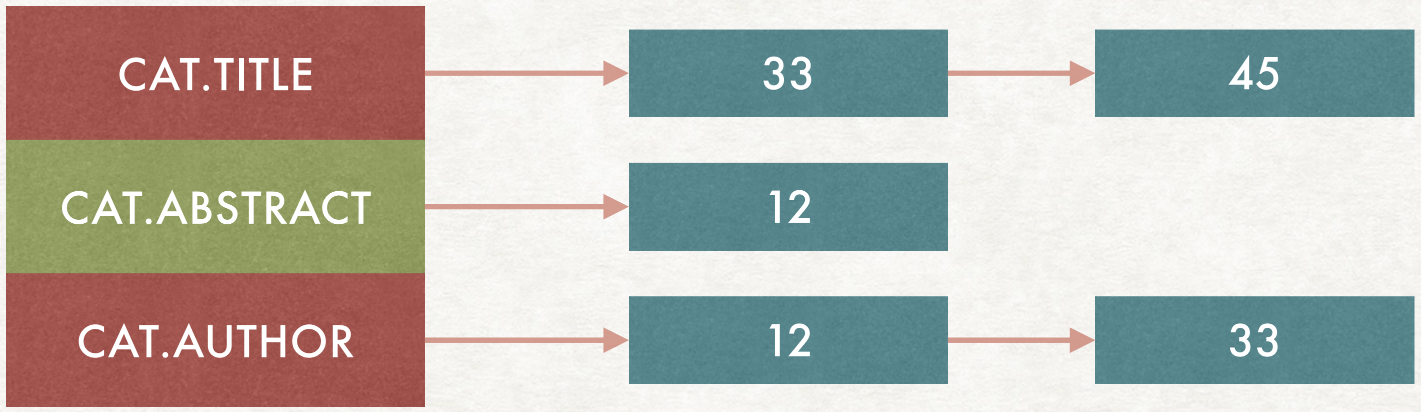
SEARCHING INSIDE FIELDS

- To allow for searching inside the fields we might want to build additional indexes, called **parametric indexes**.
- A parametric index can be thought as a standard index that only has information about a field (e.g., all the dates).
- If a query asks for “cat” in the title and “dog” inside the document we will retrieve the posting lists for dog from the “standard” index e “cat” from the parametric index for the title.
- The operations of union and intersections works as usual.

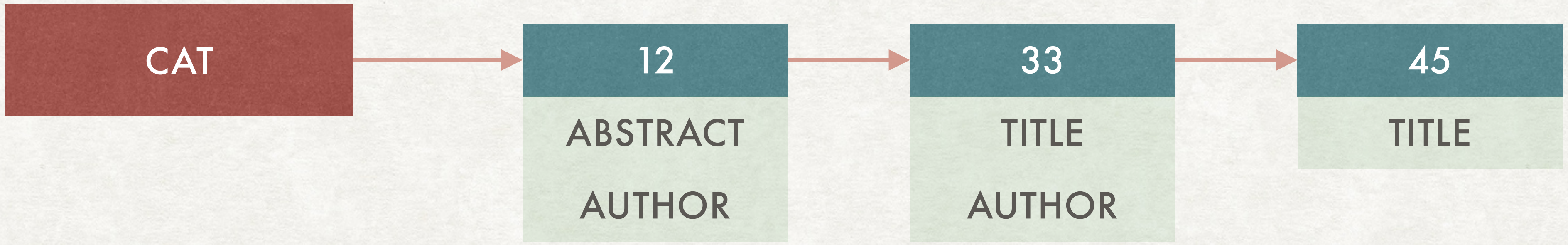
ZONE INDEXES

POSSIBLE APPROACHES

Separate inverted index for each zone



Single inverted index in which the zones are part of the postings



WEIGHTED ZONE SCORING

AN ADDITIONAL USE FOR ZONES

- We now have a way of searching inside different parts of a document…
- …but different parts might carry different importance: e.g., a title vs inside the main text.
- We can rank retrieved documents according to where the term is found inside the document.
- We can do this via **weighted zone scoring** (also called **ranked Boolean retrieval**).

SCORING FUNCTION
DEFINITION

- Consider a pair (q, d) of a query q and a document d.
- A **scoring function** associates a value in $[0,1]$ to each pair (q, d).
- Higher scores are better.
- Suppose that a document has ℓ zones.
- Each zone has a weight $g_i \in [0,1]$ for $1 \leq i \leq \ell$.
- The weights sums to one:

$$\sum_{i=1}^{\ell} g_i = 1$$

SCORING FUNCTION
PART II

- Given a query q let s_i be defined as
$$s_i = \begin{cases} 1 & \text{if } q \text{ matches in zone } i \\ 0 & \text{otherwise} \end{cases}$$

- Actually, s_i can also be defined to be any function that maps “how much” a query matches in the i-th zone.

- The weighted zone score in then defined as:
$$\sum_{i=1}^{\ell} g_i s_i$$

WEIGHTED ZONE SCORING

A SIMPLE EXAMPLE

Query: **CAT**

	Title: 0.5	Author: 0.2	Body: 0.3	
TITLE: LIFE OF A CAT AUTHOR: JAMES CAT ONCE THERE WAS A CAT...	0.5	0.2	0.3	**1**
TITLE: DOGS AND OTHER PETS AUTHOR: ANONYMOUS DOGS AND CATS ARE THE...	0	0	0.3	**0.3**
TITLE: ORCHARDS MANAGEMENT AUTHOR: JAMES CAT THE MANAGEMENT OF ORCHARDS...	0	0.2	0	**0.2**

LEARNING WEIGHTS

OR SETTING THEM MANUALLY

- The new problem is now to find how to set the weights for the different scores.
- One possibility is to ask a domain expert.
- Another possibility is to have users label documents relevant or not with respect to a query…
- …and trying to learn the weights using the training data.
- In addition to the binary classification (relevant or not) more nuanced classifications might be used.

THE TRAINING SET

Example	DocID	Query	In the title	In the body	Judgment
e1	43	LISP	1	1	Relevant
e2	43	BASIC	1	0	Relevant
e3	76	LISP	0	1	Non-relevant
e4	76	BASIC	0	1	Relevant
e5	87	SMALLTALK	1	1	Relevant
e6	87	APL	1	0	Non-relevant

COMPUTING THE ERROR

HOW TO DECIDE IF OUR WEIGHTS WORKS

With only two zones, site score is computed as:

$$\text{score}(d, q) = g \cdot s_{\text{title}} + (1 - g) \cdot s_{\text{body}}$$

Since we know the queries and the real relevance of the documents in the training set we can compute the output that a weight g would give:

$$\text{score}(43,\text{LISP}) = g \cdot 1 + (1 - g) \cdot 1$$

$$\text{score}(43,\text{BASIC}) = g \cdot 1 + (1 - g) \cdot 0$$

$$\text{score}(76,\text{LISP}) = g \cdot 0 + (1 - g) \cdot 1$$

$$\vdots$$

COMPUTING THE ERROR

HOW TO DECIDE IF OUR WEIGHTS WORKS

If we decide that relevant is 1 and non-relevant is 0
we can compare the real score with the computed one
and compute an error:

$$\text{Err}(g, e1) = (1 - \text{score}(43,\text{LISP}))^2$$

$$\text{Err}(g, e2) = (1 - \text{score}(43,\text{BASIC}))^2$$

$$\text{Err}(g, e3) = (0 - \text{score}(76,\text{LISP})^2$$

$$\vdots$$

MINIMISING THE ERROR
(AND MAYBE IT CANNOT BE ZERO)

We now want to minimise the sum of the errors:

$$\sum_{i=1}^{n} \text{Err}(g, ei)$$

Notice that it might not be possible to reach an error of zero:

$$\text{score}(43,\text{BASIC}) = g \cdot 1 + (1 - g) \cdot 0 = g$$
$$\text{score}(87,\text{APL}) = g \cdot 1 + (1 - g) \cdot 0 = g$$

But:

$$\text{Err}(g, e2) = (1 - g)^2$$
$$\text{Err}(g, e6) = g^2$$

TF-IDF WEIGHTING

CHANGING SCORING

REFINING THE SCORING

- For now we have used a weight that is either 0 or 1 depending on wether a query term was present or not.
- We might want to assign different weight depending on the term and the number of times a term is present in the document.
- This works well with *free-form text* queries:
  - For each term in the query we compute a “match score”
  - The score of a document is the sum of the scores for each term

TERM FREQUENCY

A SIMPLE SCORE

Term frequency: $\mathrm{tf}_{t,d}$

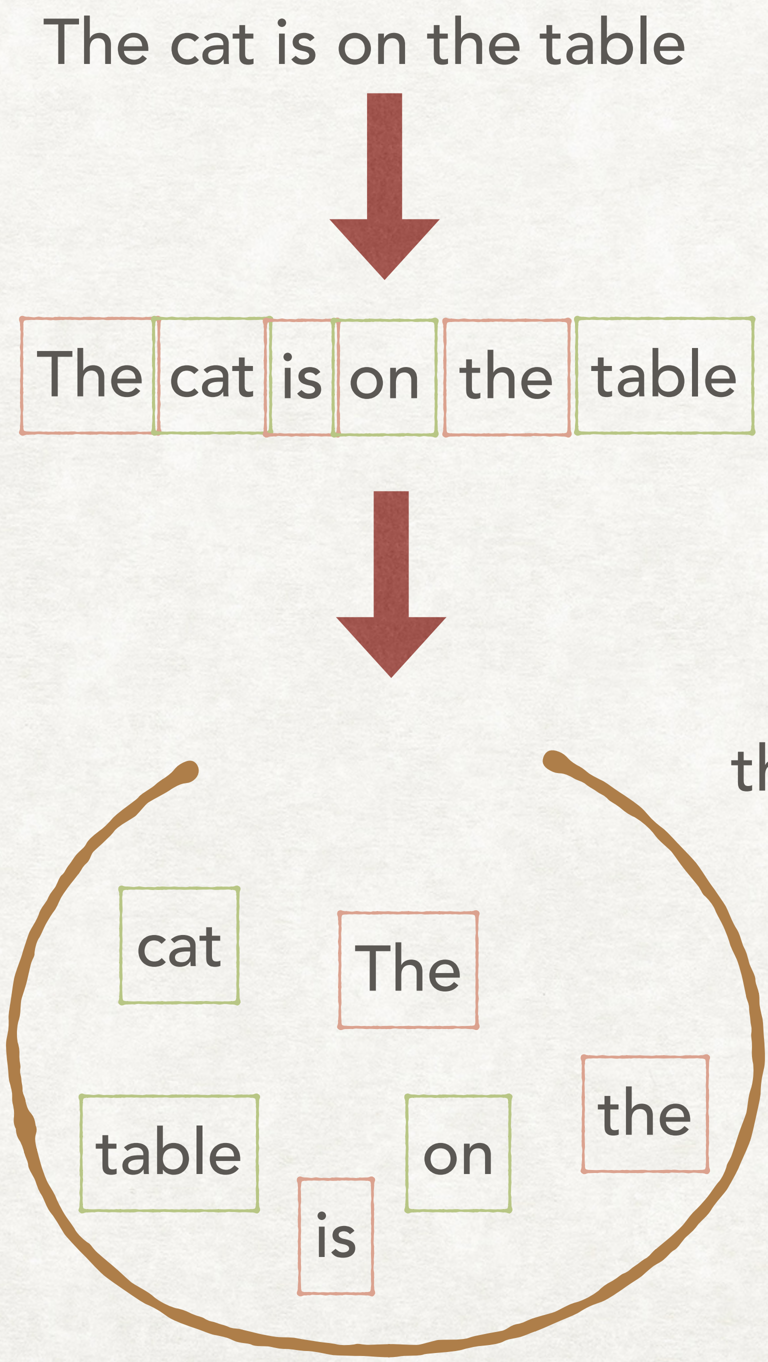
Number of occurrences of the term t inside the document d.

The main motivation is that the more a document is present inside a document the more we consider the document relevant with respect to that term.

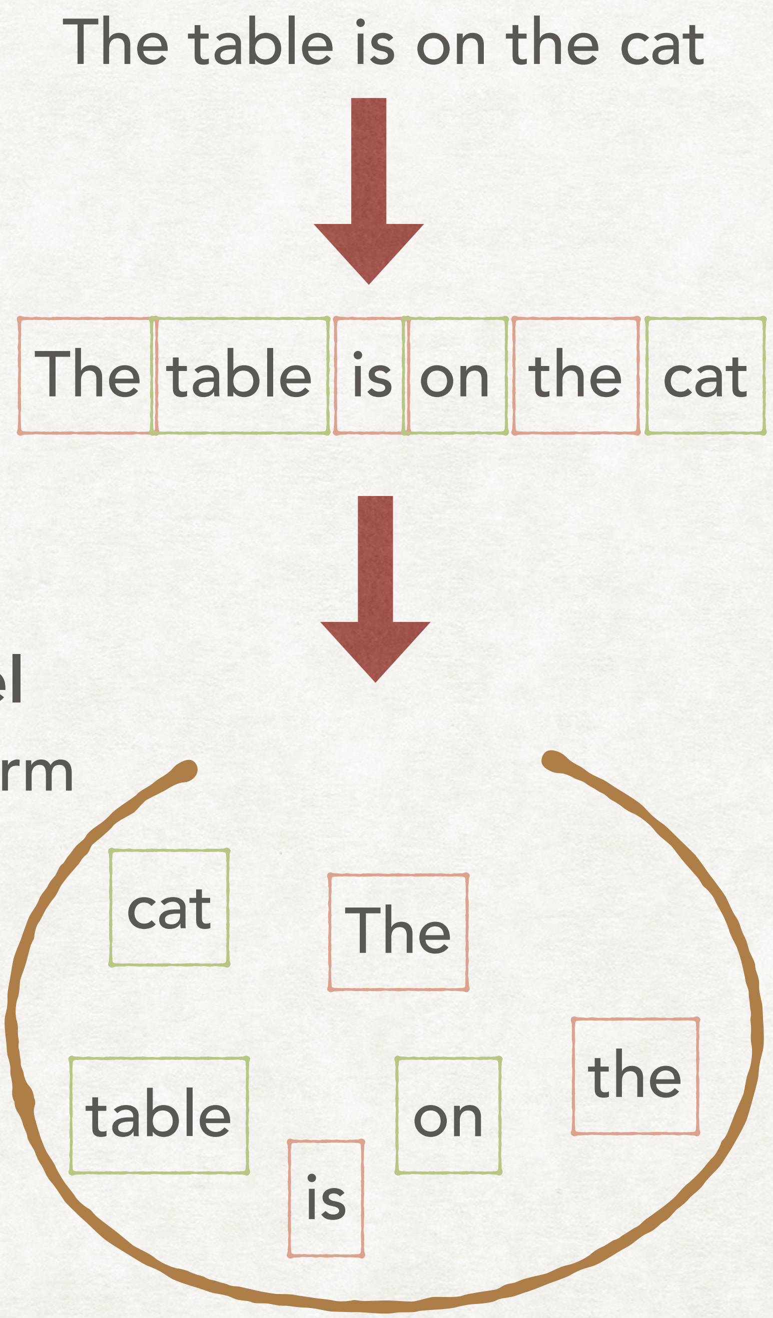
But what about the order of the words?

BAG OF WORDS

IGNORE THE ORDER!



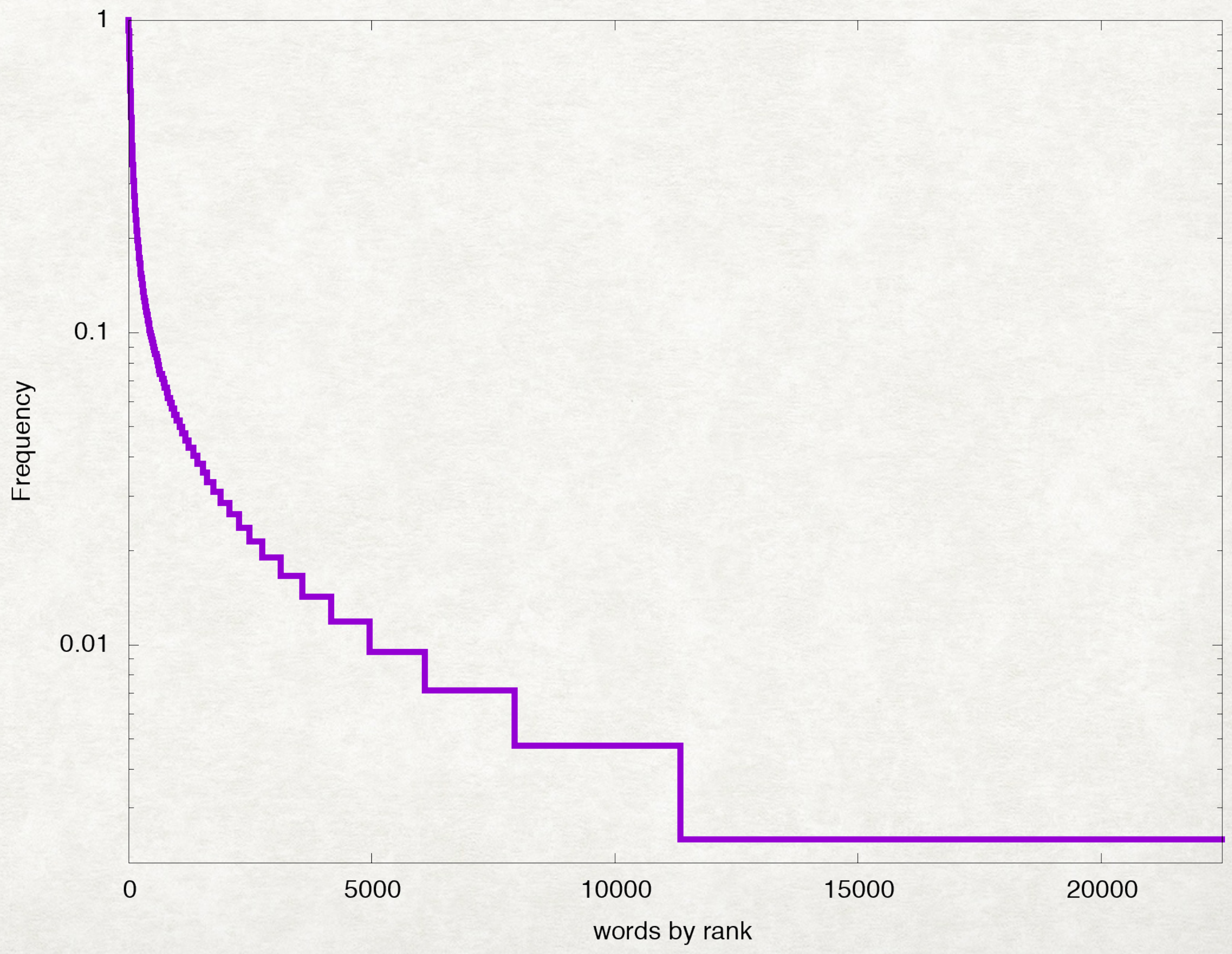
In the
bag of words model
the ordering of the term
is *immaterial* but
the amount
of occurrences
is *material*



TERM FREQUENCY

SOME LIMITATIONS

- Does the number of occurrences really represents the importance of a term?
- Which terms are more frequent?
- A small hint:
- Stop words!
- Not all terms carry the same weight in determining the relevancy of a document



COLLECTION AND DOCUMENT FREQUENCIES

RARE WORDS COUNT MORE

- The main characteristic of stop words is that they are present in most documents.
- Therefore, we might want to scale the importance of a word based on some measure of the frequency of the term:
- cf_t is the **collection frequency** of the term t:
  total number of occurrences of the term t in the collection.
- df_t is the **document frequency** of the term t:
  total number of document in which t appears in the collection.

COLLECTION AND DOCUMENT FREQUENCIES

RARE WORDS COUNT MORE

- The document frequency df_t of a term is usually preferred.
- We prefer to use a document-based measure to weight documents.
- cf_t and df_t can behave quite differently. For example:
    - A single document with 1000 instances of a term t_1 in a collection of 1000 documents.
    - Each one of 1000 documents contains a term t_2 exactly once.

INVERSE DOCUMENT FREQUENCY

MODIFYING DOCUMENT FREQUENCY

df_t is larger when we want the penalties to be larger

We use a modification of it:

Number of documents in the collection

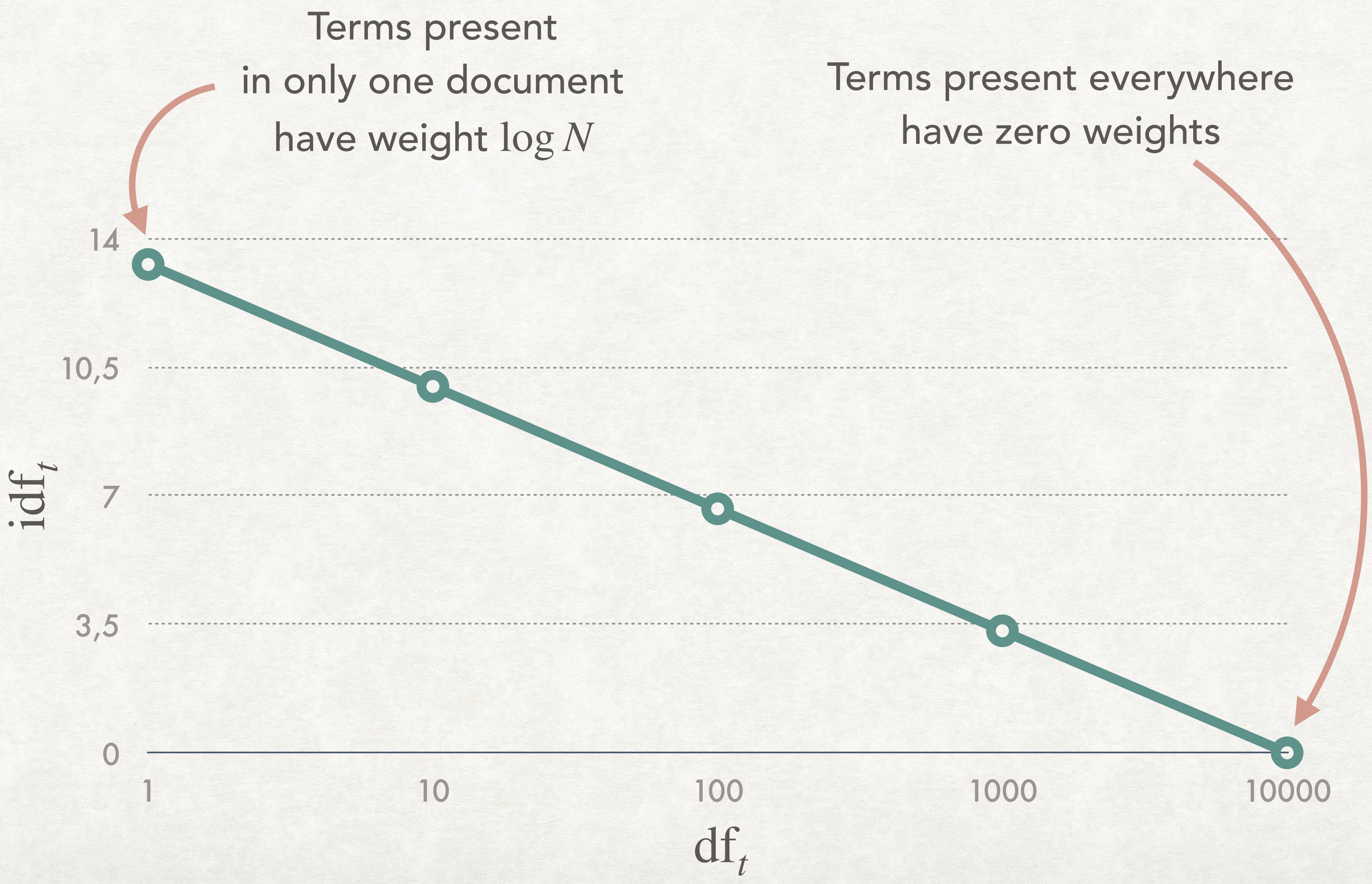
$$\mathrm{idf}_t = \log \frac{N}{\mathrm{df}_t}$$

Inverse document frequency

Document frequency

INVERSE DOCUMENT FREQUENCY

EFFECTS ON THE WEIGHTS



TF-IDF WEIGHTING

HOW TO COMBINE $\mathrm{tf}_{t,d}$ AND idf_t

We now need to combine the two ideas:

$$\text{tf-idf}_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

- When a rare term is present a many times in a document then the value is high
- When a frequent term is present many times or a rare term is present only a few time the value is low
- When a very frequent term is present only a few times then the value is the lowest

SCORING A DOCUMENT

TOWARDS THE VECTOR SPACE MODEL

The cat is on the table

We can see a document as a vector with a components for each term in the dictionary and having as elements the $\text{tf-idf}_{t,d}$ of the term t in the document

cat	is	on	table	the
$\text{tf-idf}_{\text{cat},d}$	$\text{tf-idf}_{\text{is},d}$	$\text{tf-idf}_{\text{on},d}$	$\text{tf-idf}_{\text{table},d}$	$\text{tf-idf}_{\text{the},d}$

$\text{tf-idf}_{t,d} = 0$ for all terms not in the document

SCORING A DOCUMENT

TOWARDS THE VECTOR SPACE MODEL

To score a document for a query q
we can simply sum the $\text{tf-idf}_{t,d}$ values
for all terms appearing in q:

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

Notice that in this way a document where a term
does *not* appear might still have a positive score.
The "penalty" will depend on which term is not present

VARIANTS OF TF-IDF

AND WHEN TO USE THEM

- There are some possible alternative in using directly tf-idf.
- One first consideration is that not all instances of a term inside a document carry the same weight.
- There is the idea of “diminishing returns”: is a document with 20 occurrences really twice as important as one with 10 occurrences?
- Another observation is that we might be interested in the frequency of a term relative to the other terms in the document.

SUBLINEAR TF SCALING

We can scale the $\mathrm{tf}_{t,d}$ value to have
the influence of additional terms reduced:

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

The new value can be replaced where $\mathrm{tf}_{t,d}$ is used:

$$\text{wf-idf}_{t,d} = \mathrm{wf}_{t,d} \times \mathrm{idf}_t$$

TF NORMALIZATION

We can scale the $\mathrm{tf}_{t,d}$ value to be dependant on the maximum term frequency in the document $\mathrm{tf}_{\max}(d)$:

$$\frac{\mathrm{tf}_{t,d}}{\mathrm{tf}_{\max}(d)}$$

Another possibility is to normalise according to the number of terms in the entire document:

$$\frac{\mathrm{tf}_{t,d}}{\sum_{t'\in d}\mathrm{tf}_{t',d}}$$

In both cases there are drawbacks and some smoothing might be applied to limit large swings in the normalised value

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 $\overrightarrow{v} = (v_1, v_2, \ldots, v_n)$ is $|\overrightarrow{v}| = \sqrt{\sum_{i=1}^{n} v_i^2}$
- A vector is a unit vector if its length is one.
- The inner products of two vectors $\overrightarrow{v} = (v_1, v_2, \ldots, v_n)$ and $\overrightarrow{u} = (u_1, u_2, \ldots, u_n)$ is defined as $\sum_{i=1}^{n} v_i u_i$

DOCUMENTS AS VECTORS

THE START OF THE VECTOR SPACE REPRESENTATION

$$
\left.\begin{aligned}
e_{\text{bart}} &= (1,0,0,0,0)\\
e_{\text{box}} &= (0,1,0,0,0)\\
e_{\text{cat}} &= (0,0,1,0,0)\\
e_{\text{dog}} &= (0,0,0,1,0)\\
e_{\text{drone}} &= (0,0,0,0,1)
\end{aligned}\right| \longrightarrow
$$

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:

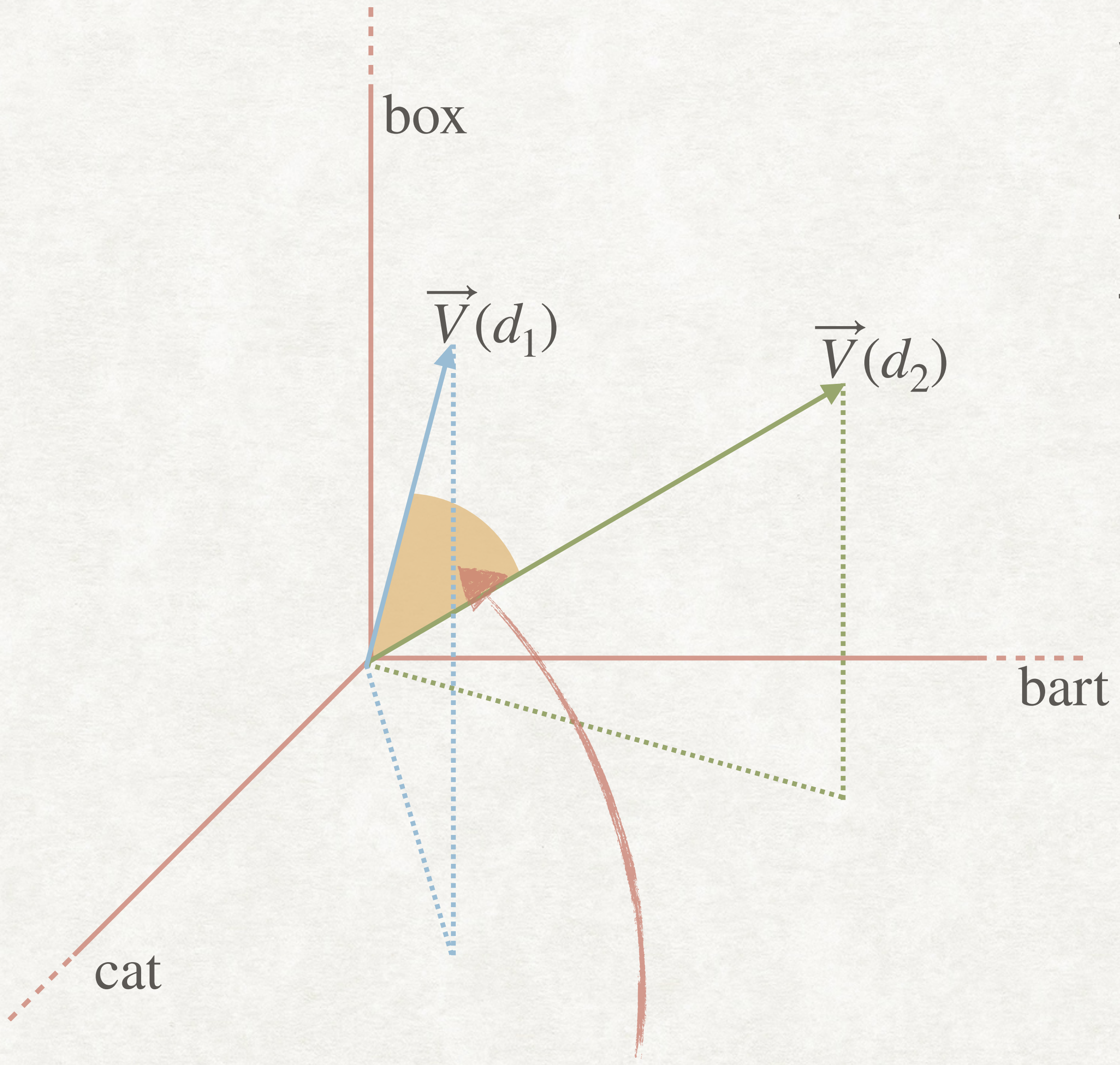
$$\overrightarrow{V}(d) = (0.6,\ 0.5,\ 0.1,\ 0,\ 0.9)$$

$\text{tf-idf}_{\text{cat},d}$

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

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

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

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 $\overrightarrow{V}(d)$ with the *unit vector* $\overrightarrow{v}(d)$:

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

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

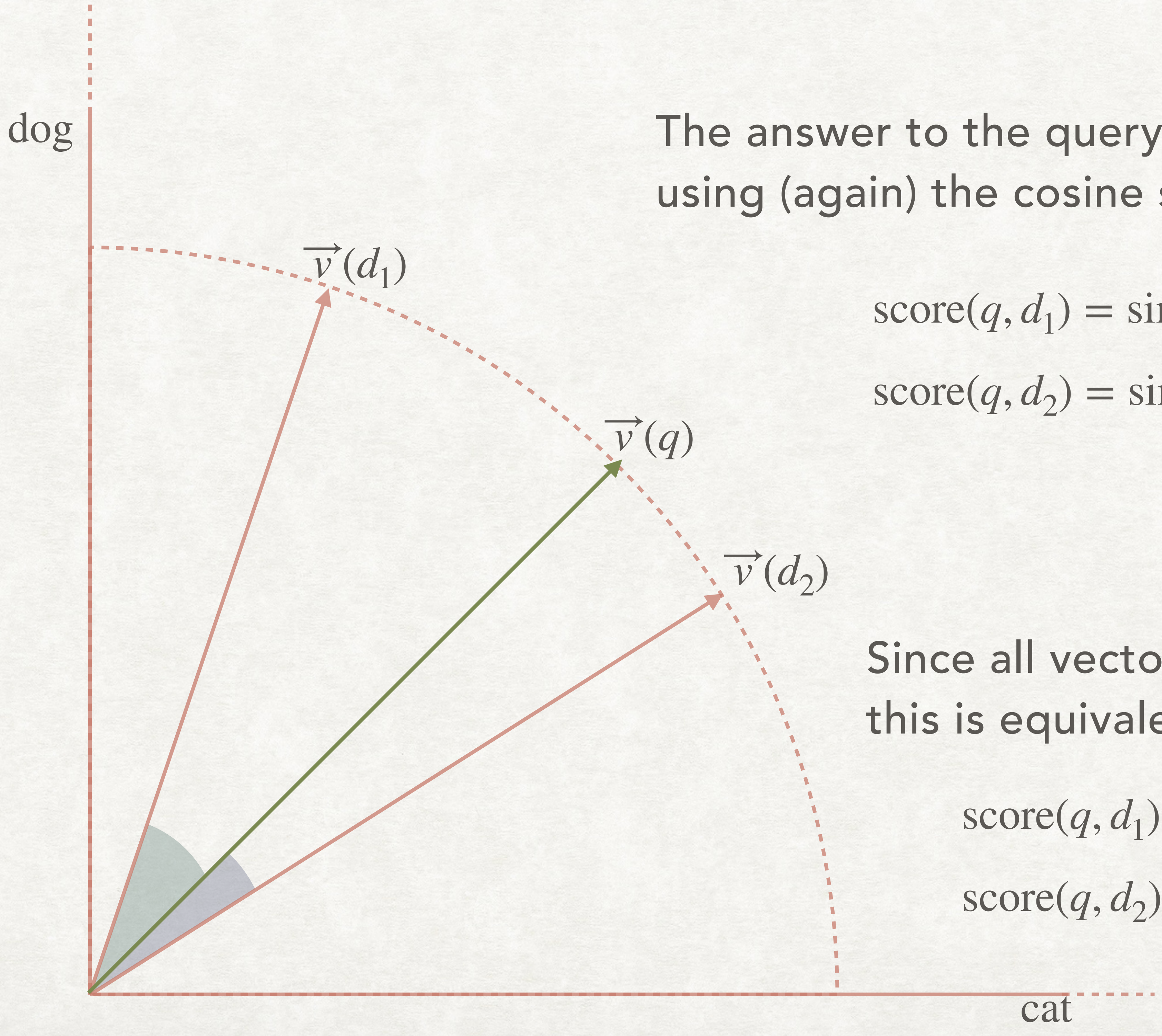
In addition to documents, also queries can be represented as vectors

Query: CAT	Vector: $(0,0,1,0,0)$
Query: CAT DOG	Vector: $\left(0,0,1/\sqrt{2},1/\sqrt{2},0\right)$

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 $\mathrm{tf}_{t,d}$ 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.
- To retrieve the K highest scoring documents we can use a *heap* data structure, which is more efficient than sorting all documents.

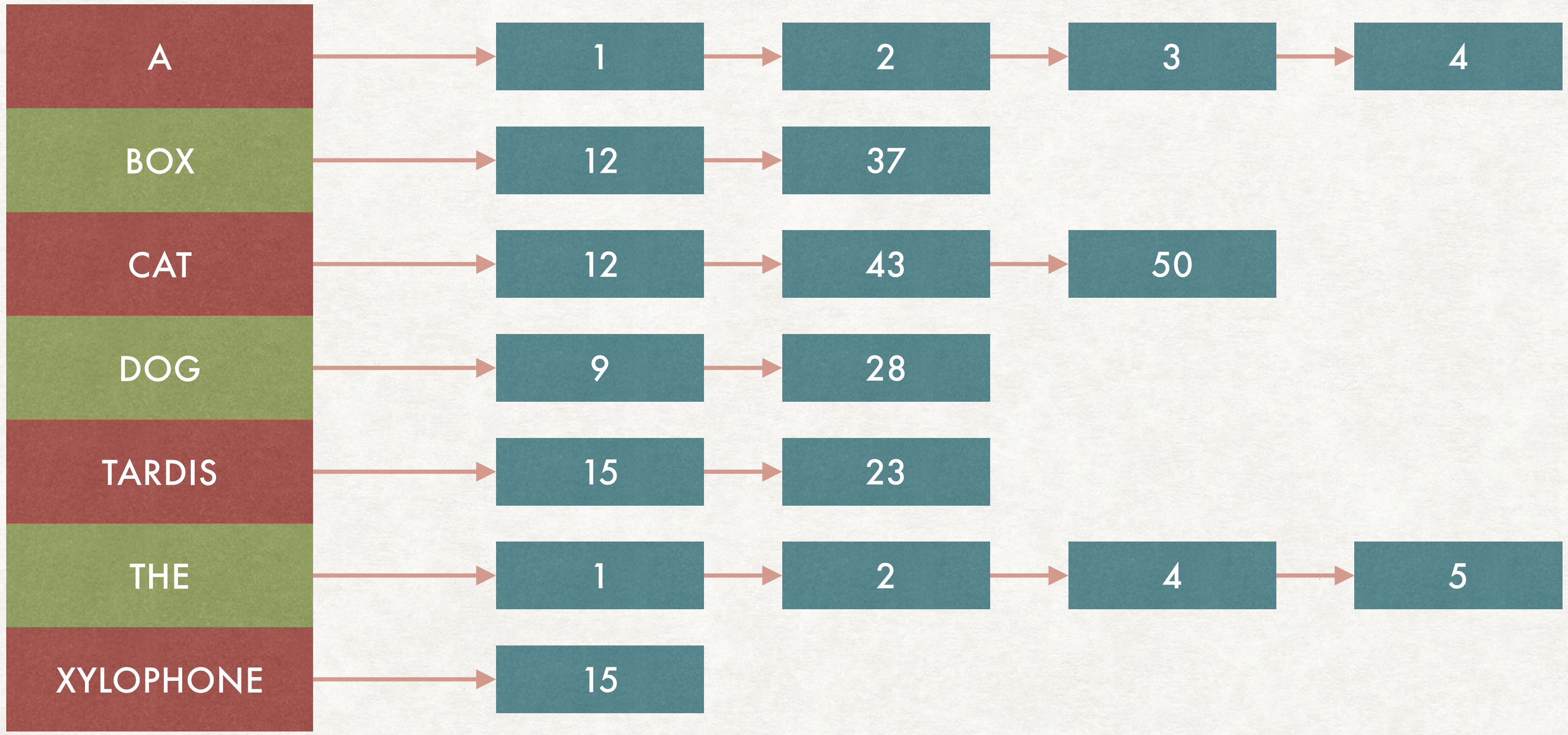
INEXACT TOP K DOCUMENT RETRIEVAL

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

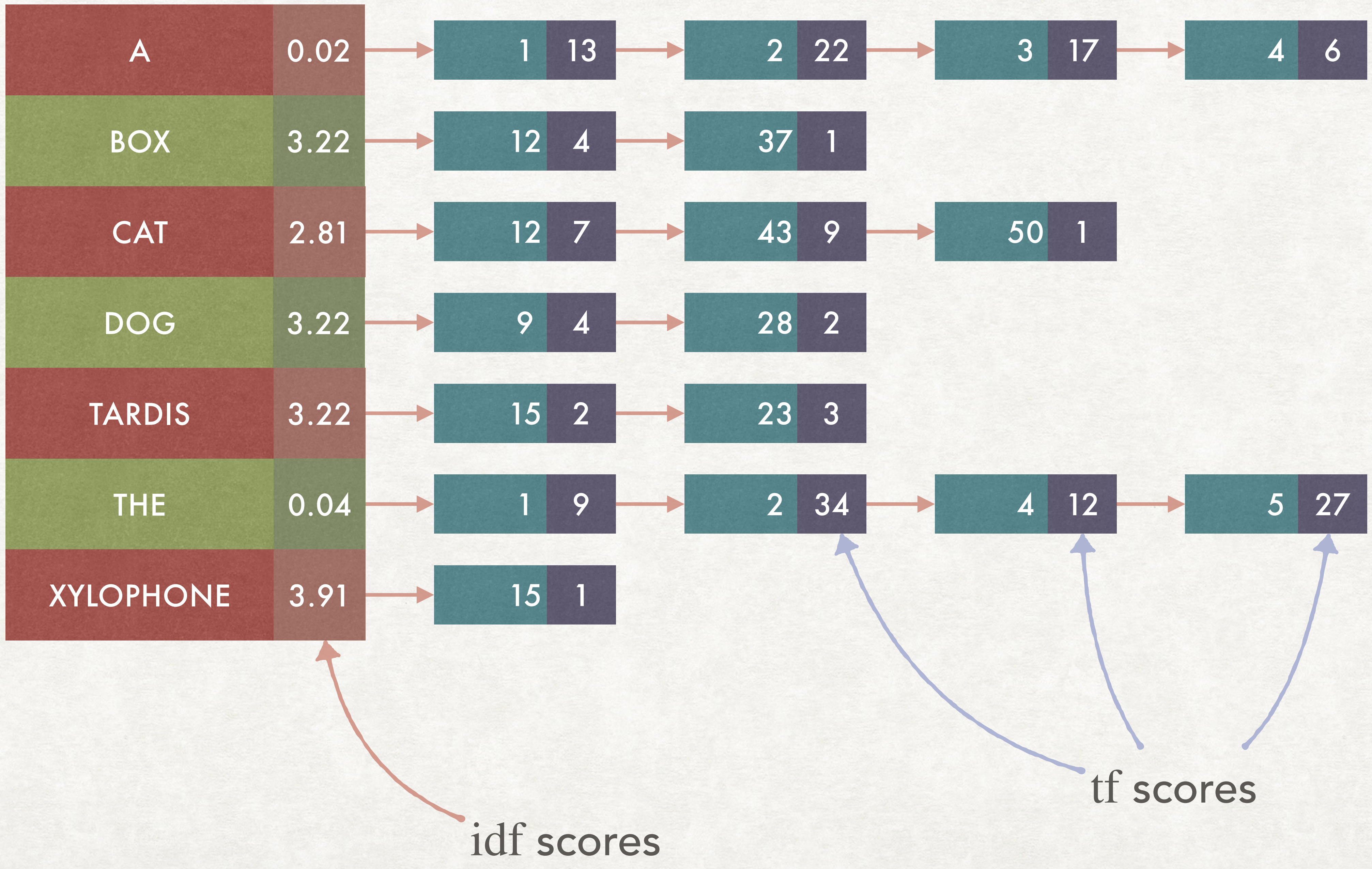
HOW TO IGNORE SOME TERMS



standard inverted index

INDEX ELIMINATION

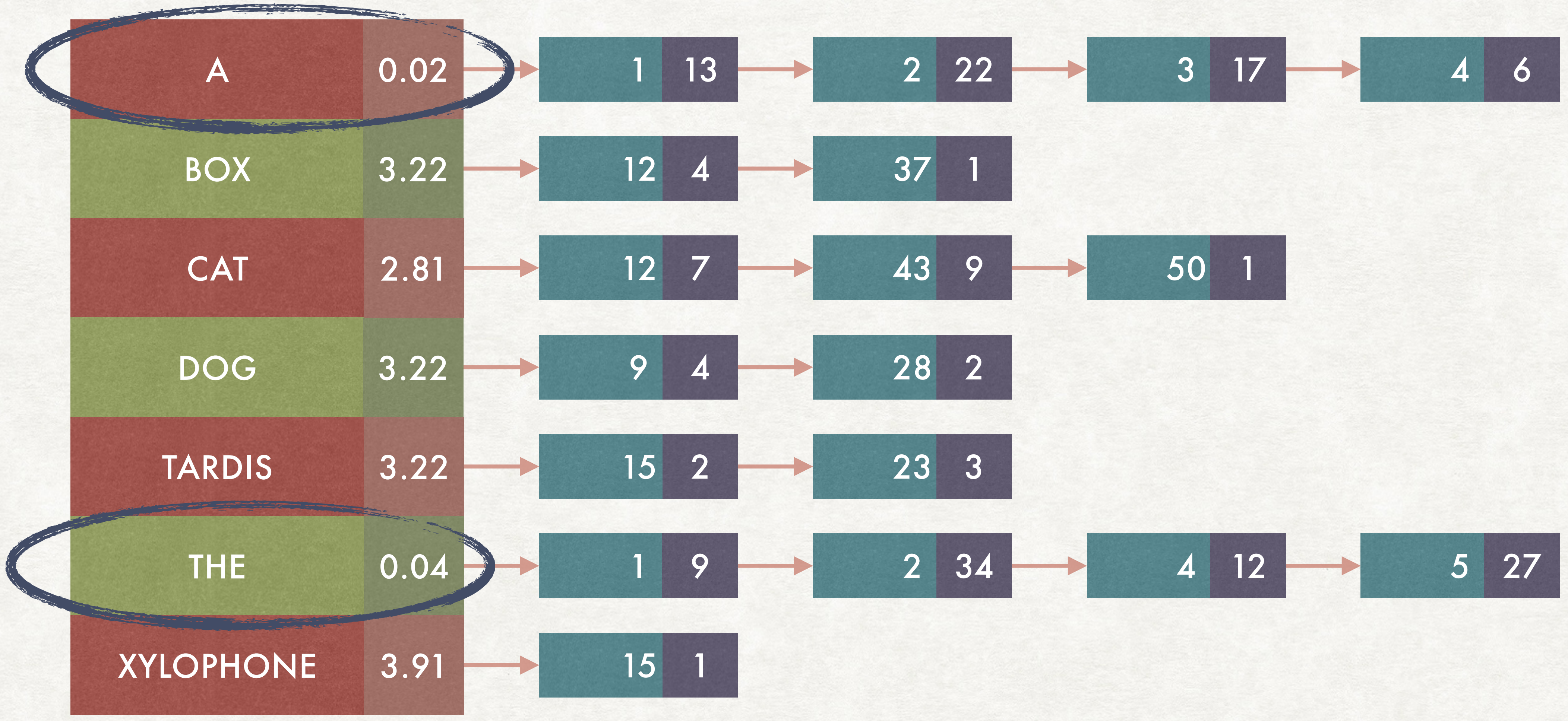
HOW TO IGNORE SOME TERMS



A 0.02 → 1 13 → 2 22 → 3 17 → 4 6
BOX 3.22 → 12 4 → 37 1
CAT 2.81 → 12 7 → 43 9 → 50 1
DOG 3.22 → 9 4 → 28 2
TARDIS 3.22 → 15 2 → 23 3
THE 0.04 → 1 9 → 2 34 → 4 12 → 5 27
XYLOPHONE 3.91 → 15 1
idf scores
tf scores

INDEX ELIMINATION

HOW TO IGNORE SOME TERMS


A 0.02 → 1 13 → 2 22 → 3 17 → 4 6
BOX 3.22 → 12 4 → 37 1
CAT 2.81 → 12 7 → 43 9 → 50 1
DOG 3.22 → 9 4 → 28 2
TARDIS 3.22 → 15 2 → 23 3
THE 0.04 → 1 9 → 2 34 → 4 12 → 5 27
XYLOPHONE 3.91 → 15 1

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.
- Problem: we might have too few documents if K is not known until the query is performed.

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: $\text{score}(q, d) = g(d) + \overrightarrow{v}(d) \cdot \overrightarrow{v}(q)$.

- 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

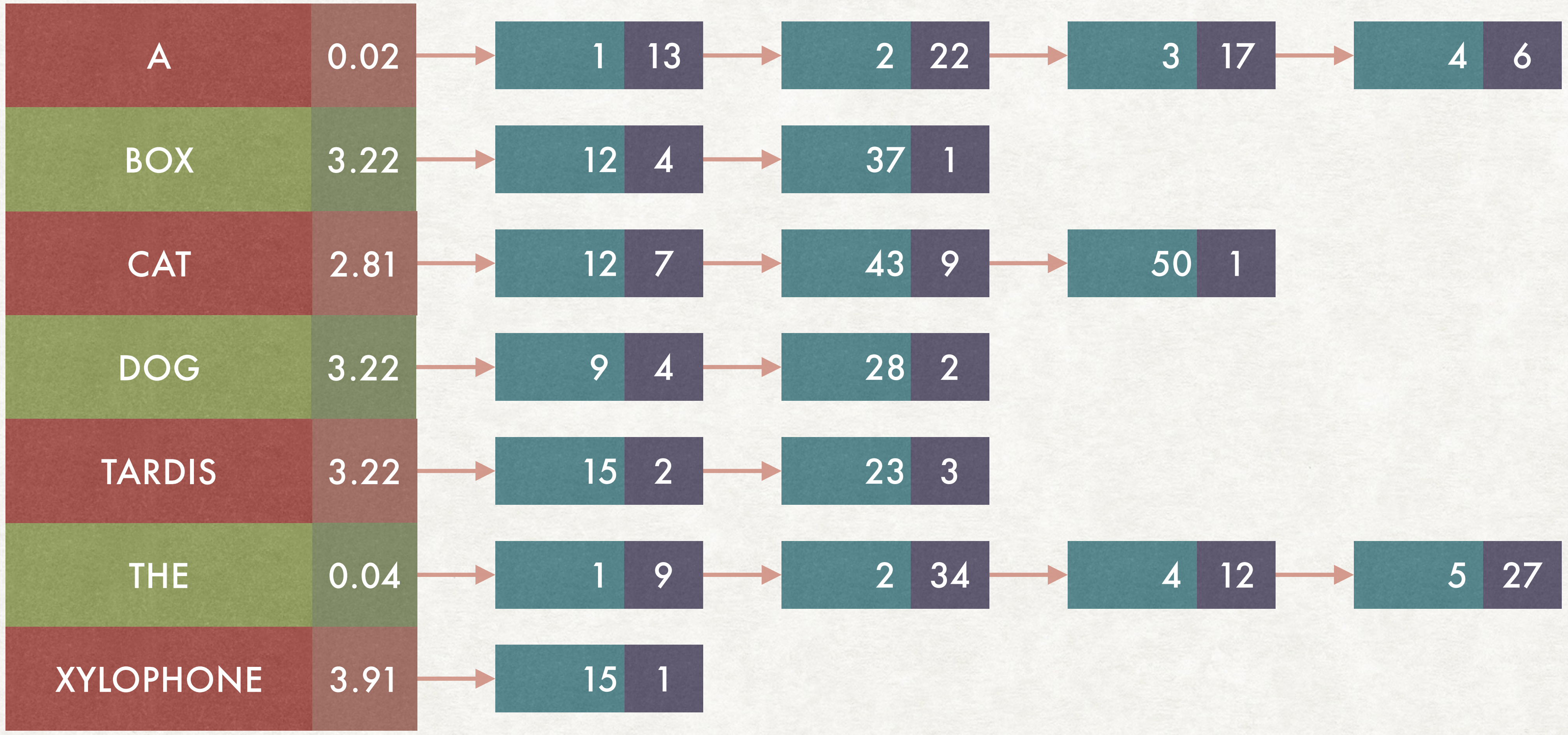
SORTING POSTING LISTS NOT BY DOCID

- Union and intersection for posting lists works efficiently because of the ordering…
- …but everything work as long as they are ordered with some criterium, not necessarily by DocID.
- Idea: Order the documents by decreasing $\mathrm{tf}_{t,d}$. In this way the documents which will obtain the highest scoring will be processed first.
- If the $\mathrm{tf}_{t,d}$ value drops below a threshold, then we can stop.

IMPACT ORDERING

SORTING POSTING LISTS NOT BY DOCID

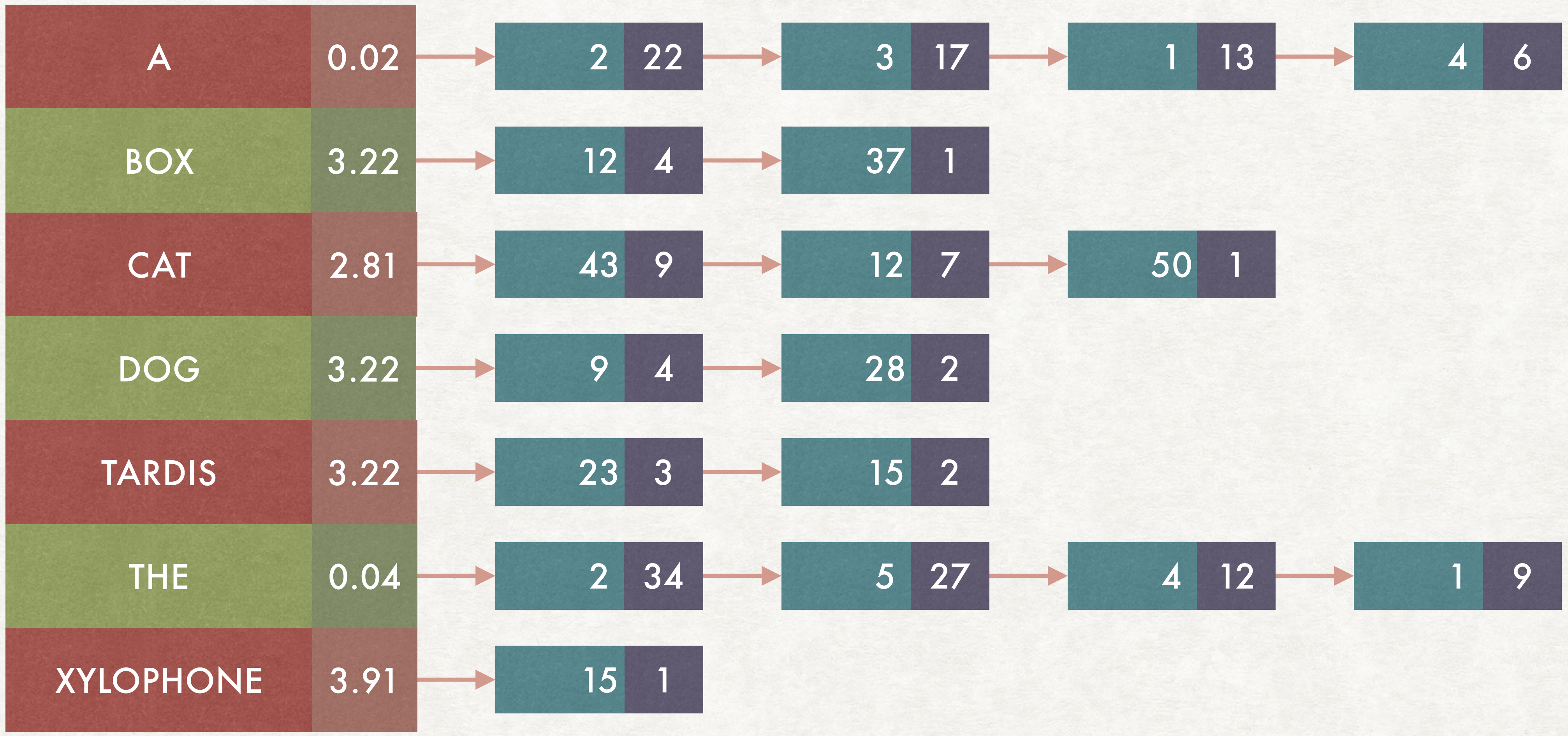
From this…


A 0.02 → 1 13 → 2 22 → 3 17 → 4 6
BOX 3.22 → 12 4 → 37 1
CAT 2.81 → 12 7 → 43 9 → 50 1
DOG 3.22 → 9 4 → 28 2
TARDIS 3.22 → 15 2 → 23 3
THE 0.04 → 1 9 → 2 34 → 4 12 → 5 27
XYLOPHONE 3.91 → 15 1

IMPACT ORDERING

SORTING POSTING LISTS NOT BY DOCID

…to this


A 0.02 → 2 22 → 3 17 → 1 13 → 4 6
BOX 3.22 → 12 4 → 37 1
CAT 2.81 → 43 9 → 12 7 → 50 1
DOG 3.22 → 9 4 → 28 2
TARDIS 3.22 → 23 3 → 15 2
THE 0.04 → 2 34 → 5 27 → 4 12 → 1 9
XYLOPHONE 3.91 → 15 1

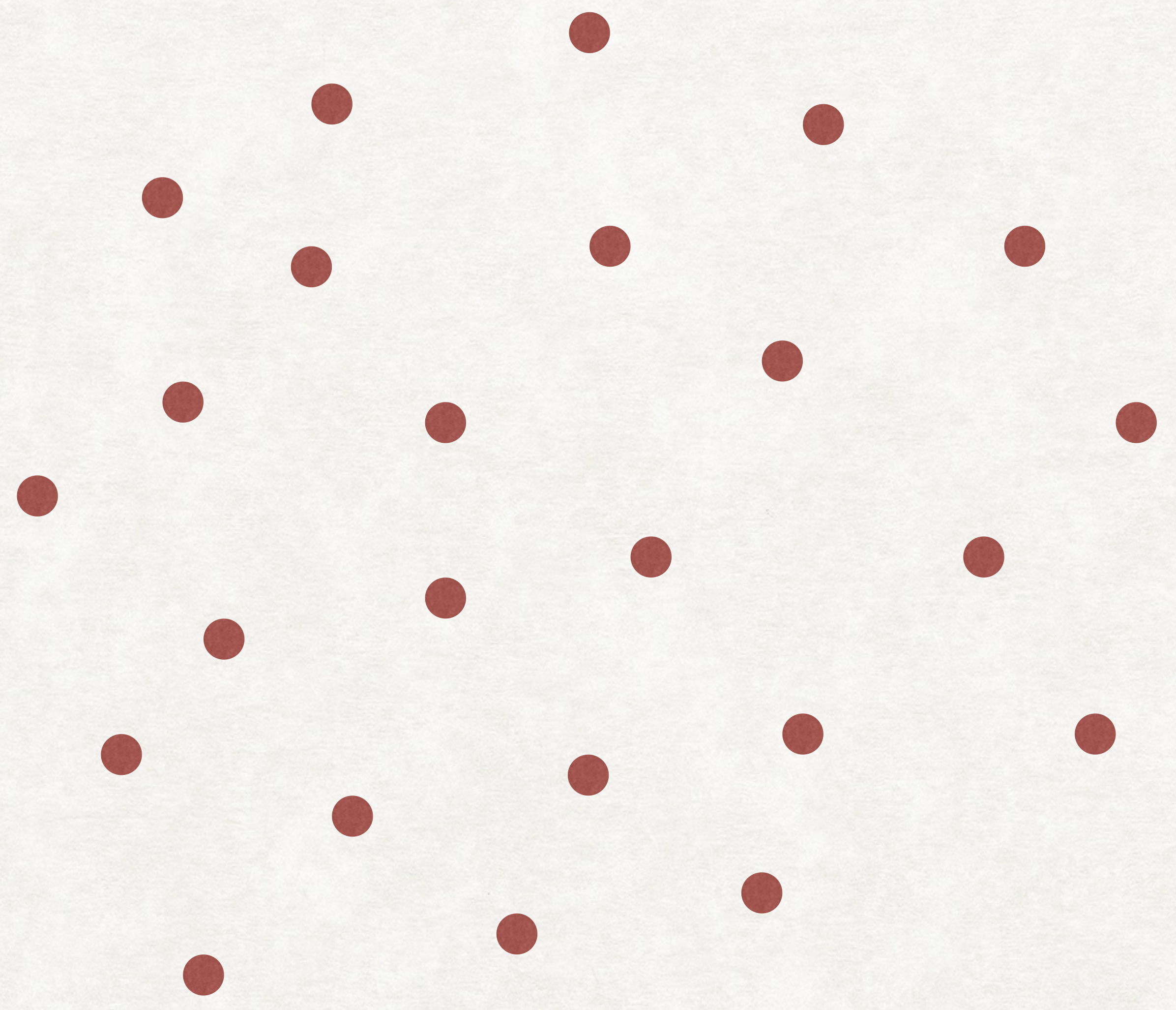
CLUSTER PRUNING

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, 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

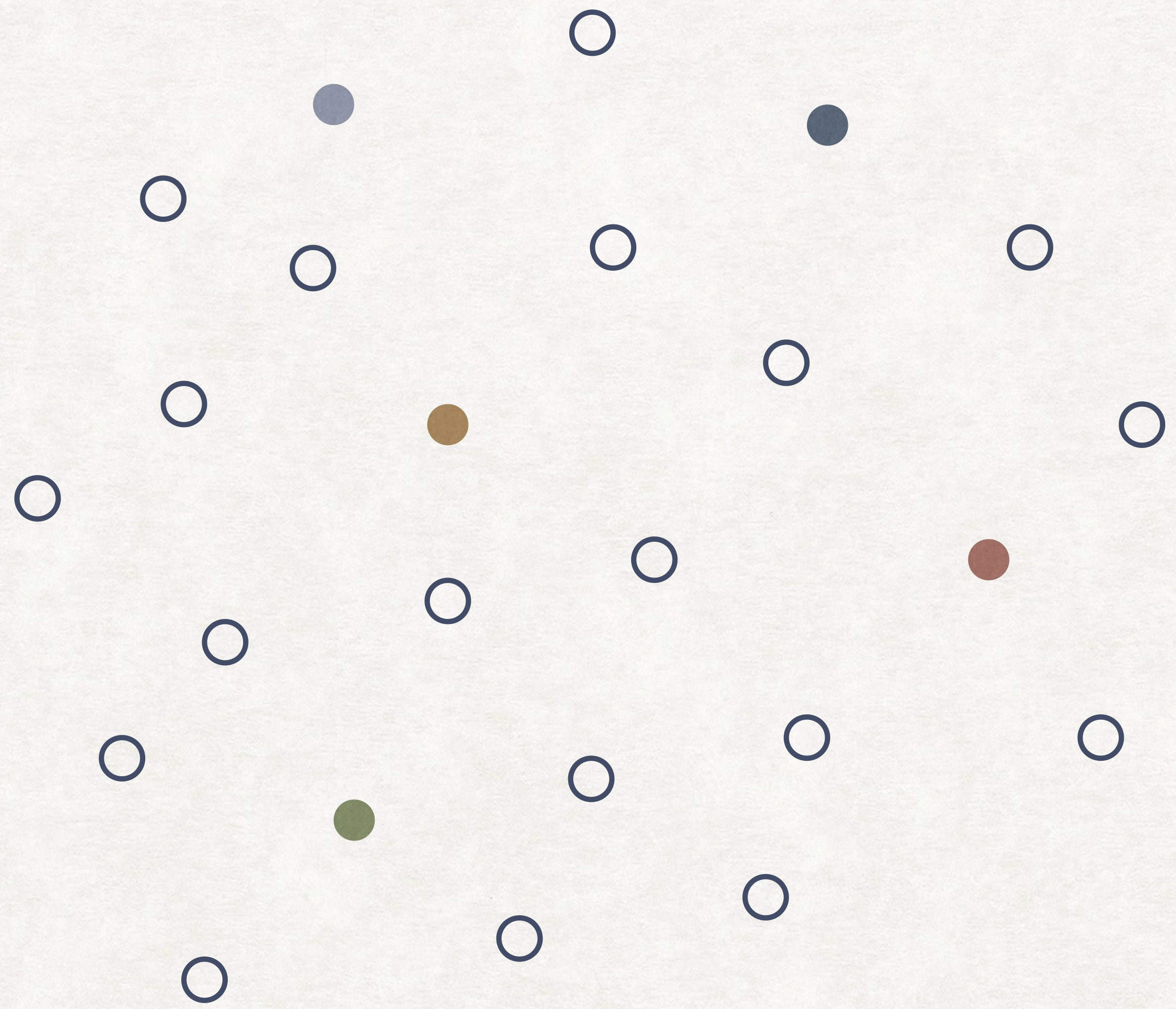
AN EXAMPLE



Documents represented as points in space

CLUSTER PRUNING

AN EXAMPLE

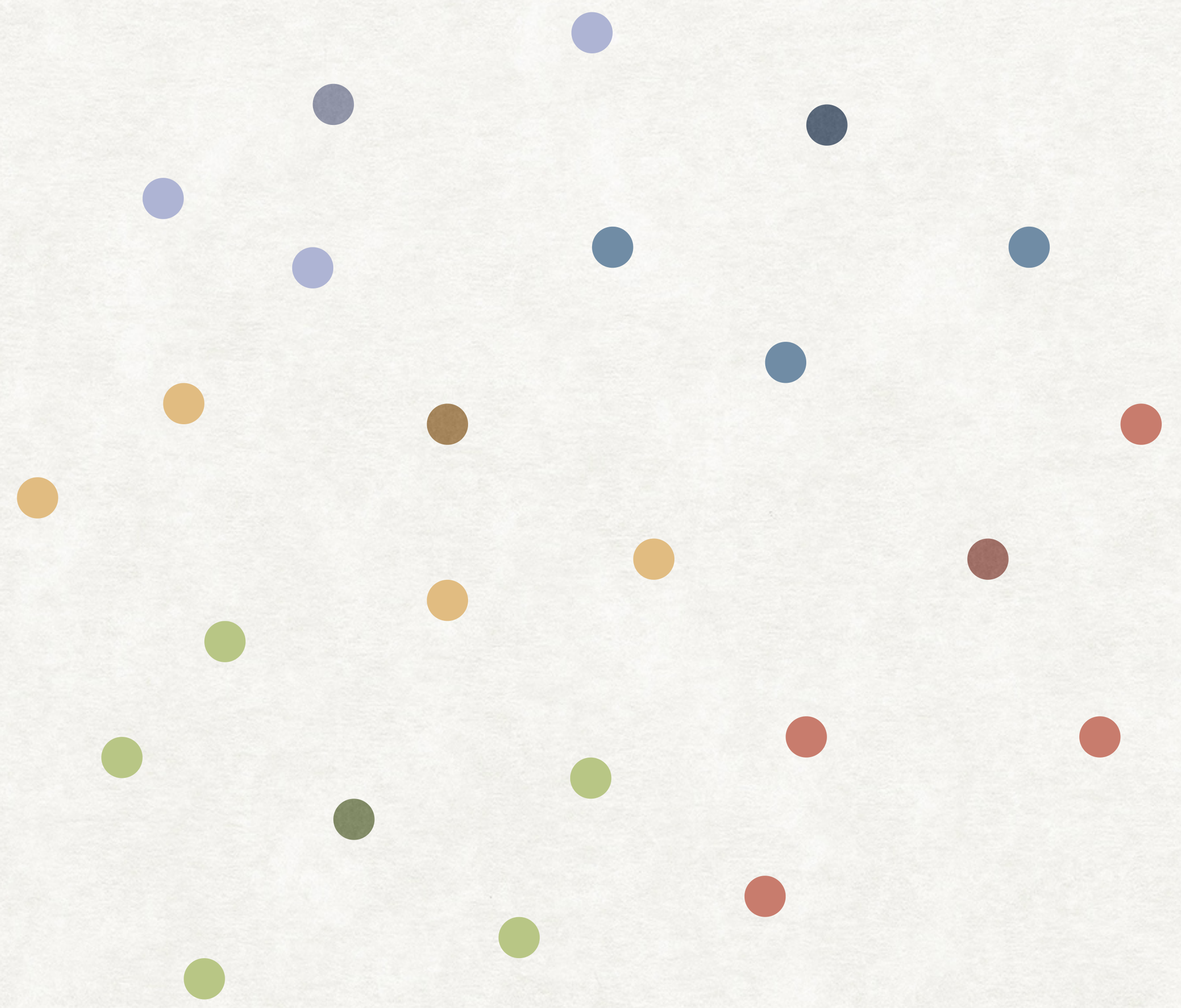


Documents represented as points in space

Selection of the leaders

CLUSTER PRUNING

AN EXAMPLE



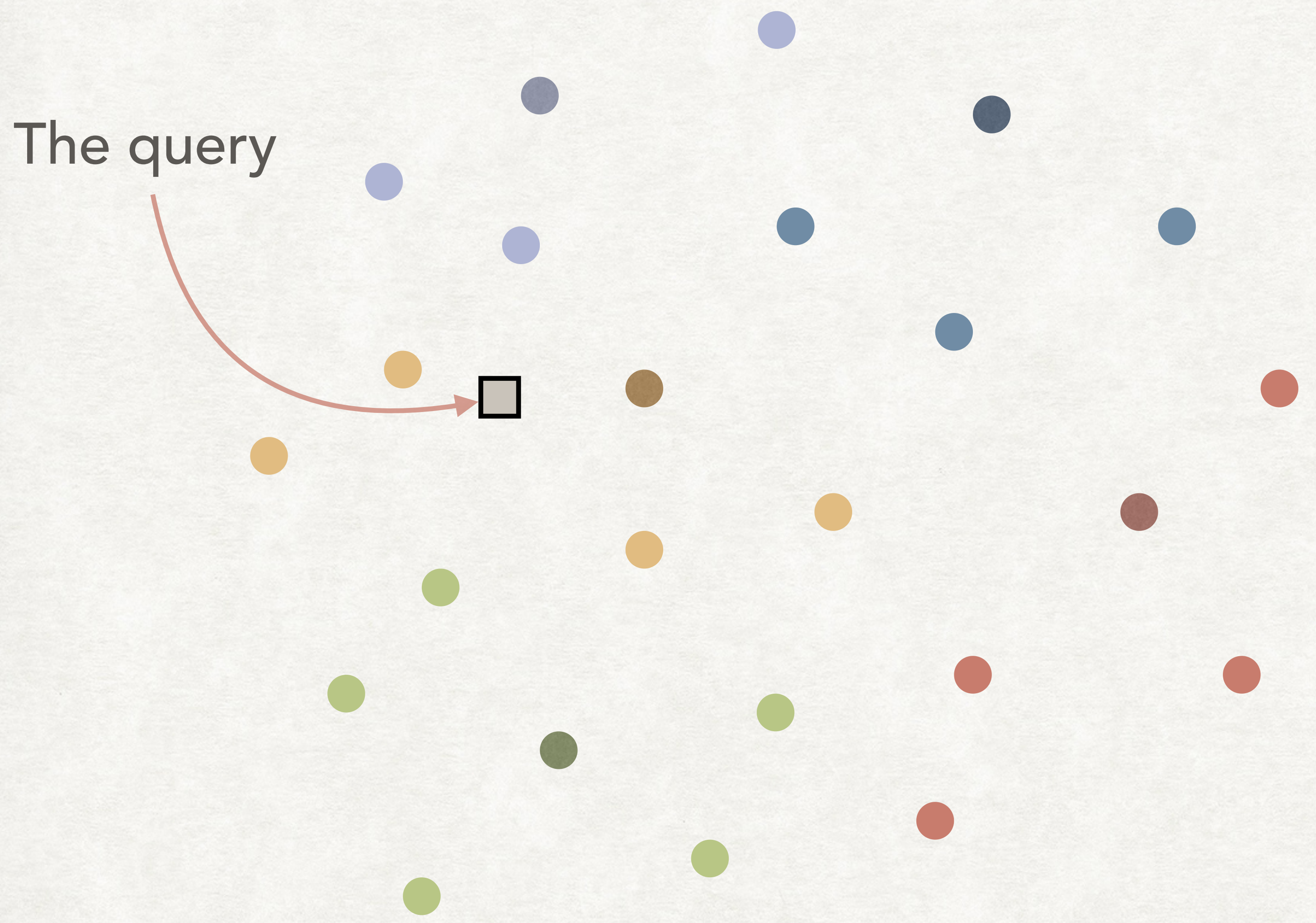
Documents represented as points in space

Selection of the leaders

Assigning documents to clusters

CLUSTER PRUNING

AN EXAMPLE



Documents represented as points in space

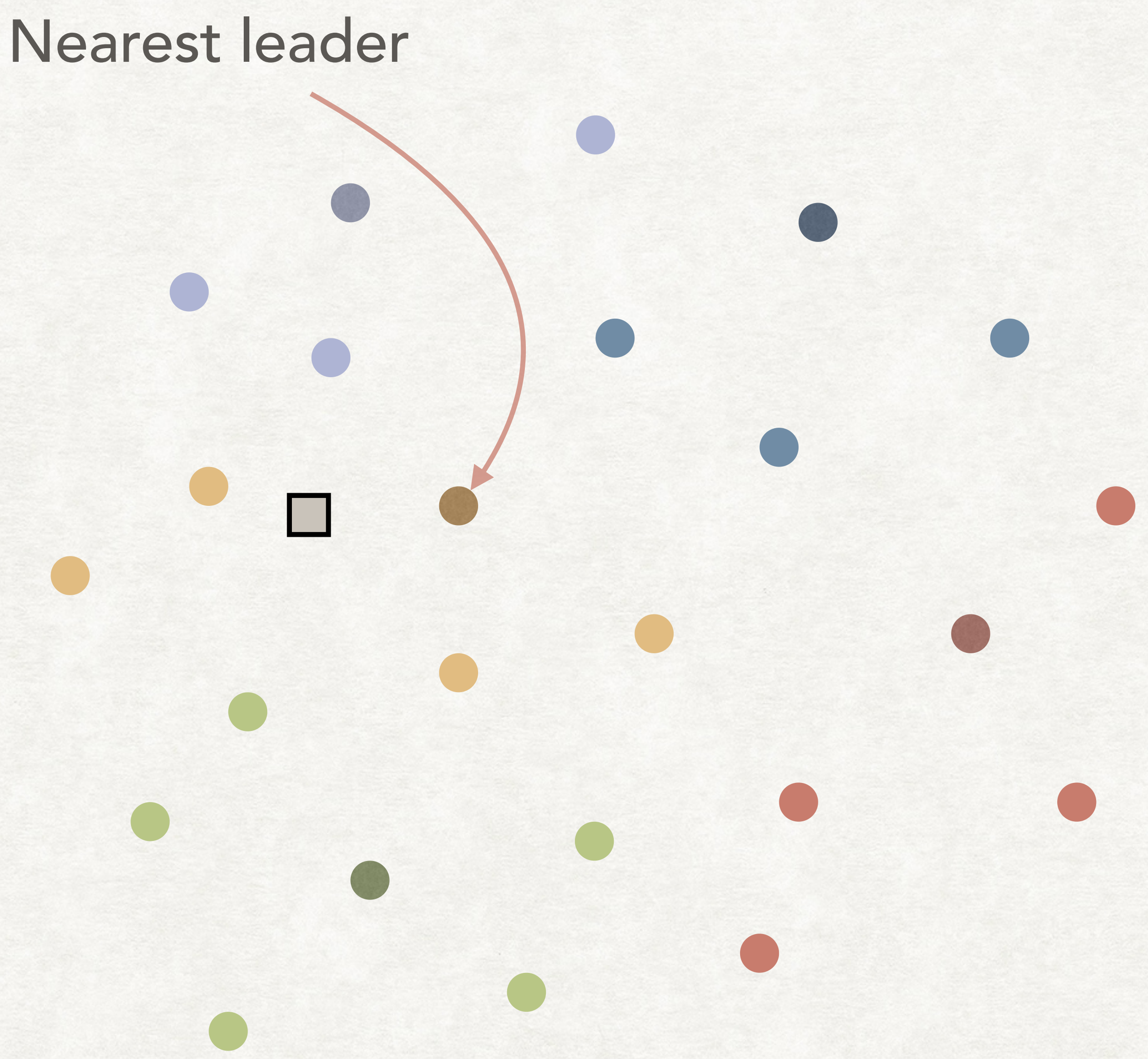
Selection of the leaders

Assigning documents to clusters

A query arrives

CLUSTER PRUNING

AN EXAMPLE



Documents represented as points in space

Selection of the leaders

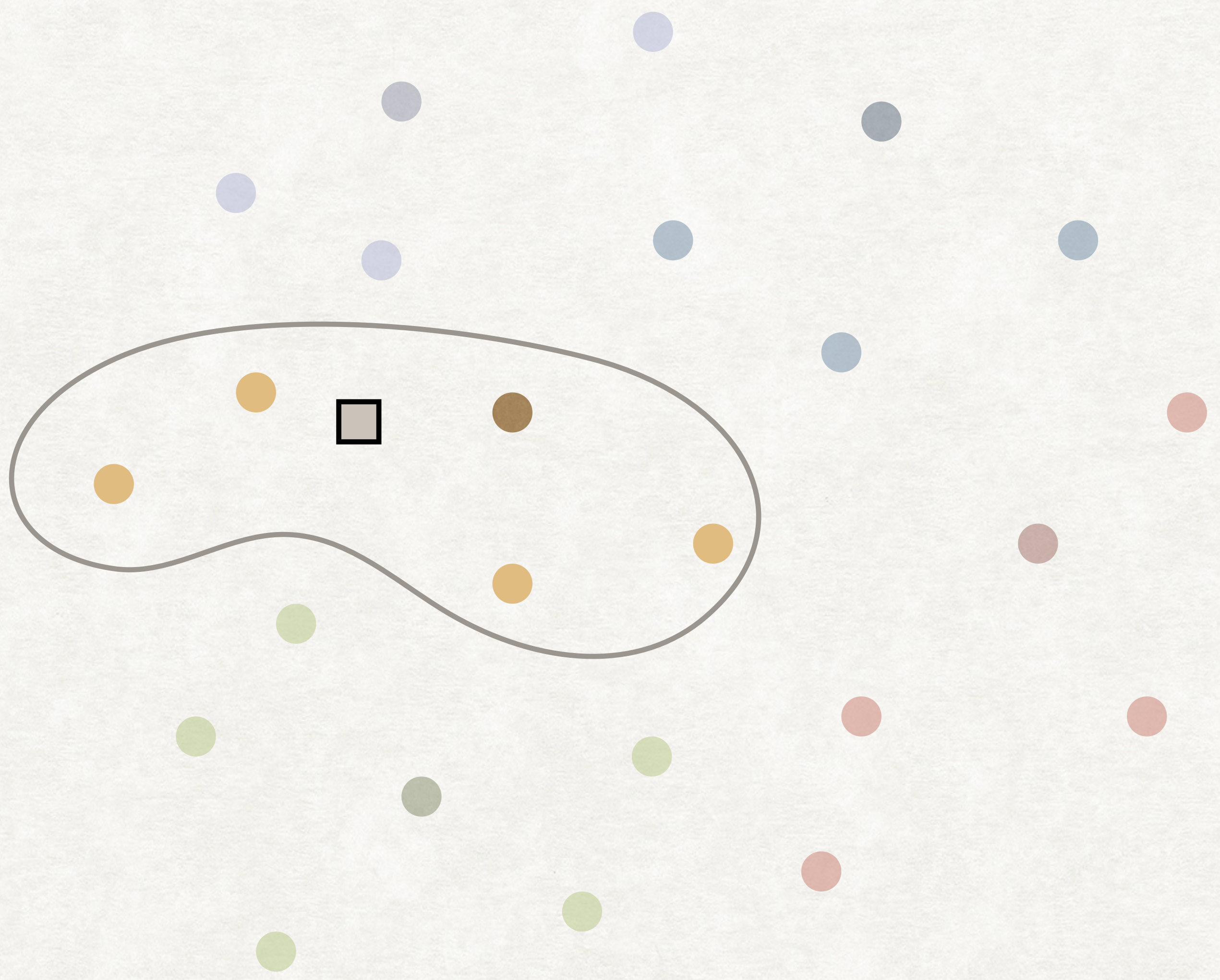
Assigning documents to clusters

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.