OBERÖSTERREICH

University of Applied Sciences

Web-Semantik-Technologien

Vorlesung WS 2019/20

thomas.kern@fh-hagenberg.at

Information Retrieval Basics

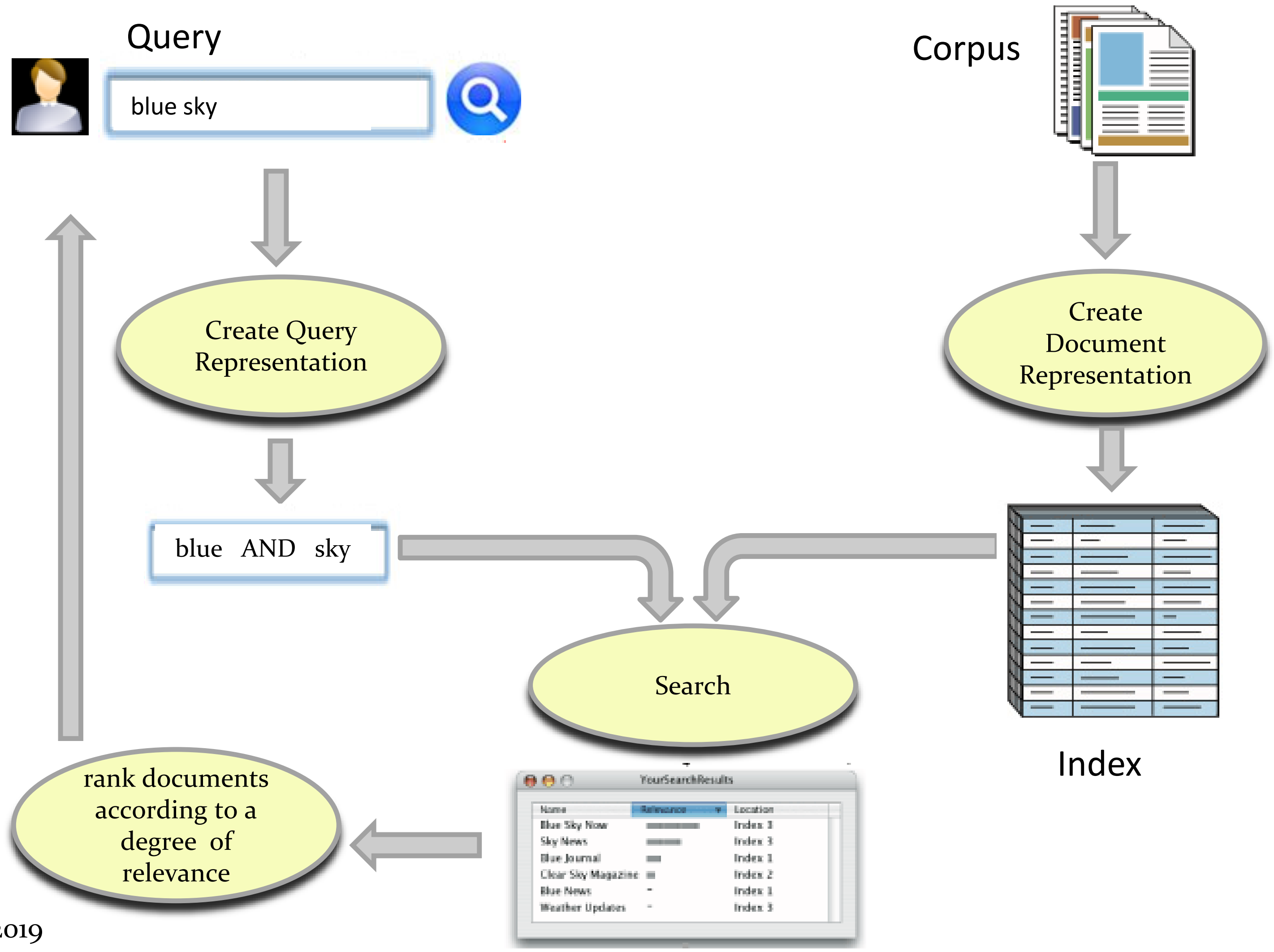
Outline – Part 1

- IR - Definitions
- IR – Process
- Preprocessing
- Indexing
- Boolean Retrieval Model
- Vector Space Model

IR - Definition

- Information Retrieval (IR) is finding material (usually documents) of an unstructured nature (usually text) that satisfies an information need of users from within large collections of documents (usually stored on computers).

IR-Process



Query
blue sky
Create Query Representation
blue AND sky
Corpus
Create Document Representation
Index
Search
rank documents according to a degree of relevance
YourSearchResults

Name	Relevance	Location
Blue Sky Now	▬▬▬▬	Index 3
Sky News	▬▬▬	Index 3
Blue Journal	▬	Index 1
Clear Sky Magazine	▬	Index 2
Blue News	-	Index 1
Weather Updates	-	Index 3

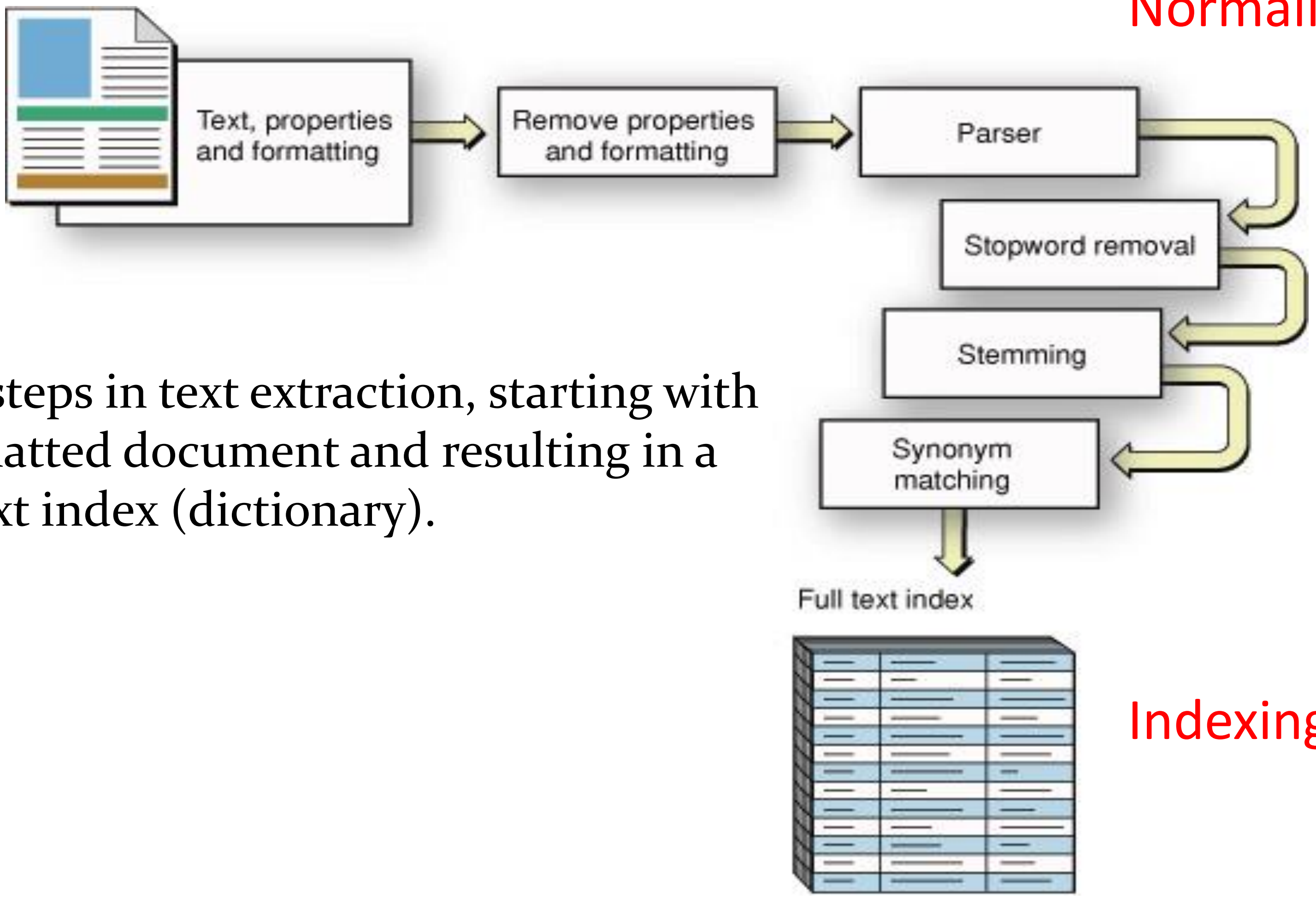
Retrieval Model

- **Determines**
  - the structure of the document representation
  - the structure of the query representation
  - the similarity matching function
- **Relevance**
  - determined by the similarity matching function
  - should reflect right topic, user needs, authority, recency
  - no objective measure
- **Quality of a retrieval model depends on how well it matches user needs !**
- **Comparison to database querying**
  - correct evaluation of a class of query language expressions (SQL)
  - can be used to implement a retrieval model

Definitions

- **QUERY**: a representation of what the user is looking for - can be a list of words or a phrase.
- **DOCUMENT**: an information entity that the user wants to retrieve
- **CORPUS** or **COLLECTION**: a set of documents
- **INDEX**: a representation of information that makes querying easier
- **TERM**: word or concept that appears in a document or a query
- **DICTIONARY**: a sorted list of all terms in the corpus, used by the index

Retrieval Model – Document Representation



Text, properties and formatting
Remove properties and formatting
Parser
Stopword removal
Stemming
Synonym matching
Full text index

Normalization

Basic steps in text extraction, starting with a formatted document and resulting in a full text index (dictionary).

Indexing

Steps in Normalization

- Strip unwanted characters/markup (e.g. HTML tags, punctuation, numbers, etc.).
- Tokenization:
  - Break into tokens (keywords) on whitespace.
- Lemmatization/Stemming:
  - Stem tokens to “root” words: computational → comput
  - Or do morphological analysis
- Stopword Removal:
  - Remove common stopwords (e.g. a, the, it, etc.).
- Build index

Tokenization

- Tokenization is the process of **breaking a stream of text up** into words, phrases, symbols, or other meaningful elements called tokens.
- **Tokens are small units of meaningful text**, such that a comparison between a token in the query and a token in a document can take place
- Typically, tokenization occurs at the word level. However, it is sometimes **difficult to define what is meant by a "word".**
- Often a tokenizer relies on simple heuristics, for example:
  - All **contiguous strings of alphabetic characters** are part of one token; likewise with numbers.
  - Tokens are **separated by whitespace characters**, such as a space or line break, or by punctuation characters.
  - Punctuation and whitespace may or may not be included in the resulting list of tokens.

In languages such as English where words are delimited by whitespace, approach is straightforward.

From: wikipedia

Tokenization in Biomedical Text

- Tokenization is **not trivial for languages in special domains** due to the domain-specific terminologies (Biomedicine)
- Biomedical text contains not only English words, but also many **special terms** such as the names of genes, proteins, chemicals and diseases
- Names contain characters such as numerals, hyphens, slashes and brackets, and the same entity often has different lexical variants.
- **A simple tokenizer for general English text cannot work well in biomedical text:**
  - If all non-alphanumerical characters inside a named entity are used as delimiters to separate the name into several tokens, the proximity of these tokens is lost
- Result is a **loss of the semantic meaning** of the tokens and mismatches

Tokenisation problems in Bio-text

- What should be done with punctuation that has linguistic meaning?
- Hyphens:
  - Negative charge (Cl-)
  - Absence of symptom (-fever)
  - Gene name (IL-2 –mediated)
  - Plus, “syntactic” uses (insulin-dependent)
- Commas
  - 2,6-diaminohexanoic acid
  - tricyclo(3.3.1.13,7)decanone

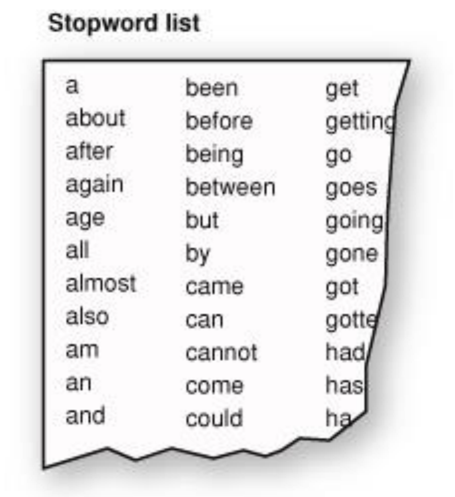
Heuristic rules to deal with non-functional characters in Bio-text

1. replace the following characters with spaces (not common in Bio-text)
   ! " # $ % & * < = > ? @ \ | ~
2. remove the following characters if they are followed by a space
   . : ; ,
3. remove the following pairs of brackets if the open bracket is preceded by a space and the close bracket is followed by a space:
   ( ) [ ]
4. remove the single quotation mark ′ if it is preceded by a space or if it is followed by a space
5. remove ′s and ′t if they are followed by a space
6. remove slash / if it is followed by a space

Following non-alphanumerical characters may still occur in the modified text:
( ) [ ] - _ / . : ; , ‘ +

Last step of tokenization is to change all upper case letters into lower cases.

Stop Word Removal


Stopword list

a	been	get
about	before	getting
after	being	go
again	between	goes
age	but	going
all	by	gone
almost	came	got
also	can	gotte
am	cannot	had
an	come	has
and	could	ha

- Stopwords: **words which are too frequent among the documents**
- They have **little semantic content**
  - Articles, prepositions, conjunctions, …
  - Some verbs, adverbs, and adjectives
  - Stop Word Removal according to a Stop Word List
- To **reduce the size** of the indexing structure
- Stopword removal **can be harmful**
  - Example: “to be or not to be”
- Stopwordlist used in Pubmed:
  - http://www.ncbi.nlm.nih.gov/books/NBK3827/table/pubmedhelp.T43/

Stemming

- Different word forms may bear similar meaning (e.g. search, searching): create a "standard" representation for them

- **Stemming:** Removing endings of word

$$\left.\begin{array}{l}\text{computer}\\ \text{compute}\\ \text{computes}\\ \text{computing}\\ \text{computed}\\ \text{computation}\end{array}\right\}\ \textbf{comput}$$

- **Definition of stemming:** Crude heuristic process that chops off the ends of words in the hope of achieving what lemmatization attempts to do with a lot of linguistic knowledge.

Stemming: Porter algorithm

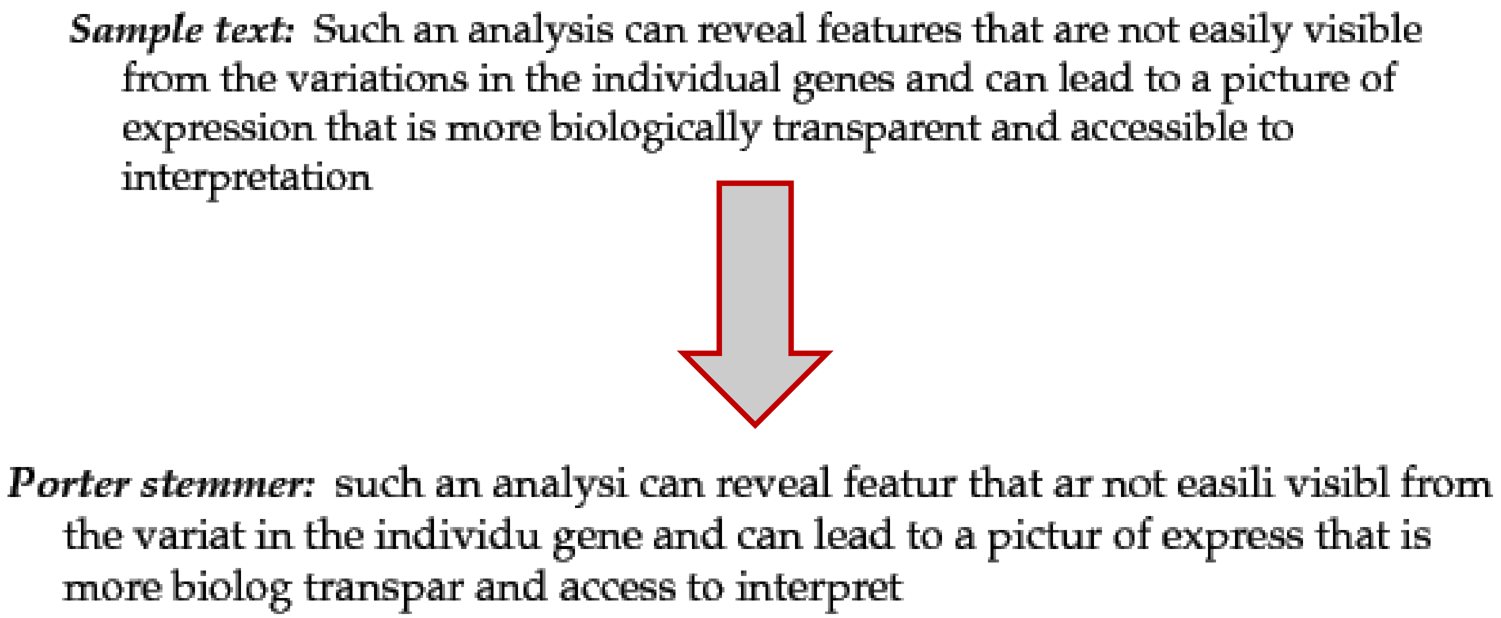
- Special algorithm for the English language **based on suffix removal**
  - 5 phases of word reductions, applied sequentially.
  - Within each phase there are various conventions to select rules, such as selecting the rule from each rule group that applies to the longest suffix of the word
  - Examples:

sses → ss	bosses → boss
ies → i	bodies → bodi
ss → ss	caress → caress
s → NULL	cats → cat
tional → tion	positional → position

- Weight of word sensitive rules
  - (m>1) EMENT: replacement → replac
    cement → cement

- Full description: http://snowball.tartarus.org/algorithms/porter/stemmer.html

Stemming: Porter algorithm Example



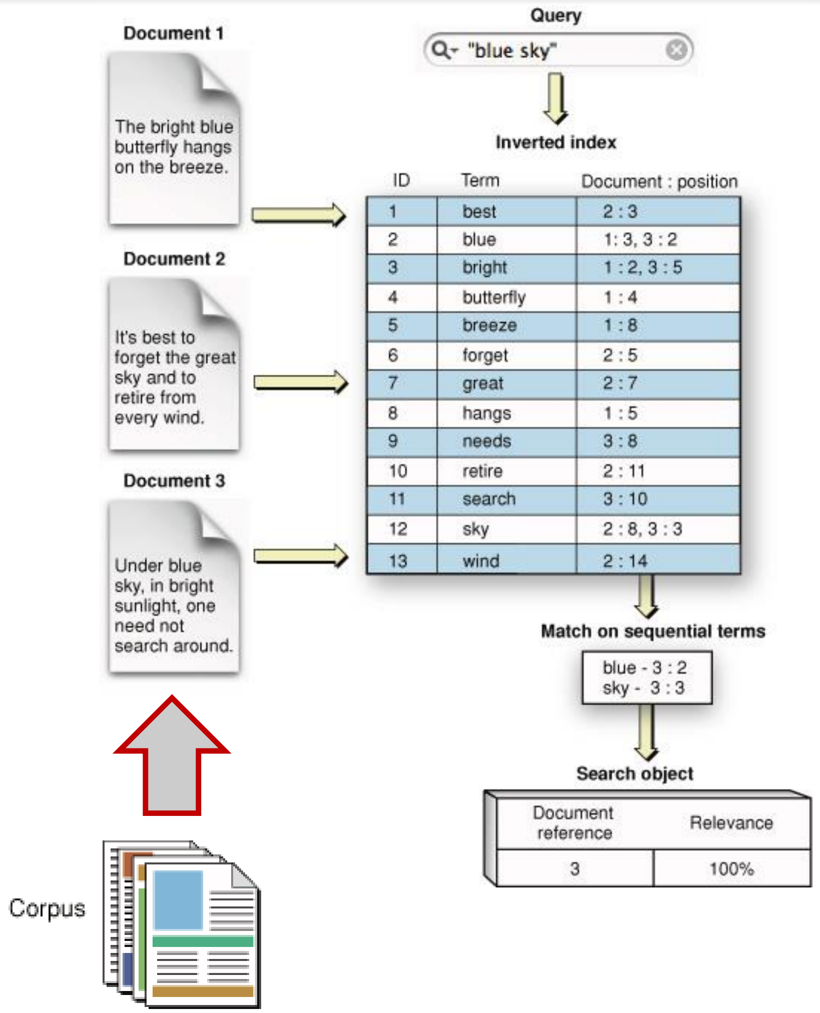
Sample text: Such an analysis can reveal features that are not easily visible from the variations in the individual genes and can lead to a picture of expression that is more biologically transparent and accessible to interpretation

Porter stemmer: such an analysi can reveal featur that ar not easili visibl from the variat in the individu gene and can lead to a pictur of express that is more biolog transpar and access to interpret

Lemmatization

- Lemmatization is **more complex**, it uses **morphological analysis** and needs **vocabularies** to find the base form
- Reduce inflectional/variant forms to base form
  - Example: am, are, is → be
  - Example: car, cars, car's, cars' → car
  - Example: the boy's cars are different colors →
    the boy car be different color
- Lemmatization implies doing "**proper**" **reduction** to dictionary headword form (the lemma).
- Inflectional morphology (cutting → cut) vs. Derivational morphology (destruction → destroy)

Index


Document 1
The bright blue butterfly hangs on the breeze.
Document 2
It's best to forget the great sky and to retire from every wind.
Document 3
Under blue sky, in bright sunlight, one need not search around.
Corpus
Query
"blue sky"
Inverted index
Match on sequential terms
blue - 3 : 2
sky - 3 : 3
Search object

ID	Term	Document : position
1	best	2 : 3
2	blue	1 : 3, 3 : 2
3	bright	1 : 2, 3 : 5
4	butterfly	1 : 4
5	breeze	1 : 8
6	forget	2 : 5
7	great	2 : 7
8	hangs	1 : 5
9	needs	3 : 8
10	retire	2 : 11
11	search	3 : 10
12	sky	2 : 8, 3 : 3
13	wind	2 : 14

Document reference	Relevance
3	100%

- An index is a data structure that is used to **quickly retrieve documents** matching the user query.

- The index type depends on the sort of user-queries it should support.

Index Construction

- Manual Indexing
- Automatic Indexing
  - Inverted Index
  - Vector Space Models

Manual Indexing

- Every document is catalogued based on some individual's or group's assessment of what that document is about, and an appropriate list of descriptive entries is generated.
- **Advantage**
  - Human indexers can establish **relationships and concepts between seemingly different topics that can be very useful** to future readers
  - Broader, narrower and related subjects
- **Disadvantage**
  - Slow and expensive

Automatic indexing: Index Types

- **Inverted indexes map terms to documents.** They allow users to discover which documents match their queries.
- **Vector Space Models (VSM) map documents to terms.** They allow users to discover not only which documents match their query, but also get a ranking which documents match best.

Inverted Indexes

Document 1 (ID=1)

The bright blue butterfly hangs on the breeze.

Document 2 (ID=2)

It's best to forget the great sky and to retire from every wind.

Document 3 (ID=3)

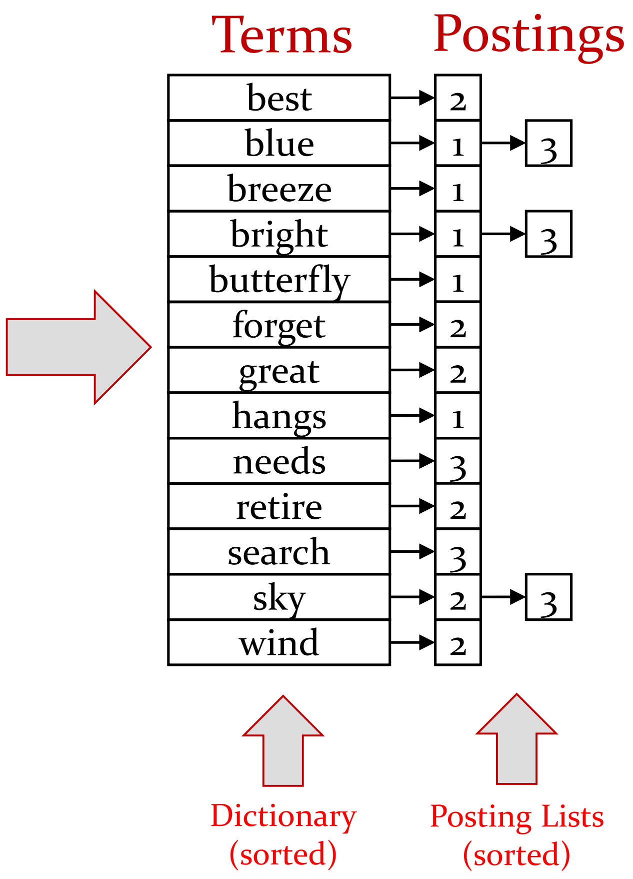
Under blue sky one needs not search around.

Terms	Document 1	Document 2	Document 3
best	0	1	0
blue	1	0	1
breeze	1	0	0
bright	1	0	1
butterfly	1	0	0
forget	0	1	0
great	0	1	0
hangs	1	0	0
needs	0	0	1
retire	0	1	0
search	0	0	1
sky	0	1	1
wind	0	1	0

Stopword List
and
around
every
it
from
not
one
the
to
under

Inverted indexes

Terms	Document 1	Document 2	Document 3
best	0	1	0
blue	1	0	1
breeze	1	0	0
bright	1	0	1
butterfly	1	0	0
forget	0	1	0
great	0	1	0
hangs	1	0	0
needs	0	0	1
retire	0	1	0
search	0	0	1
sky	0	1	1
wind	0	1	0



Terms	Postings
best	2
blue	1 → 3
breeze	1
bright	1 → 3
butterfly	1
forget	2
great	2
hangs	1
needs	3
retire	2
search	3
sky	2 → 3
wind	2

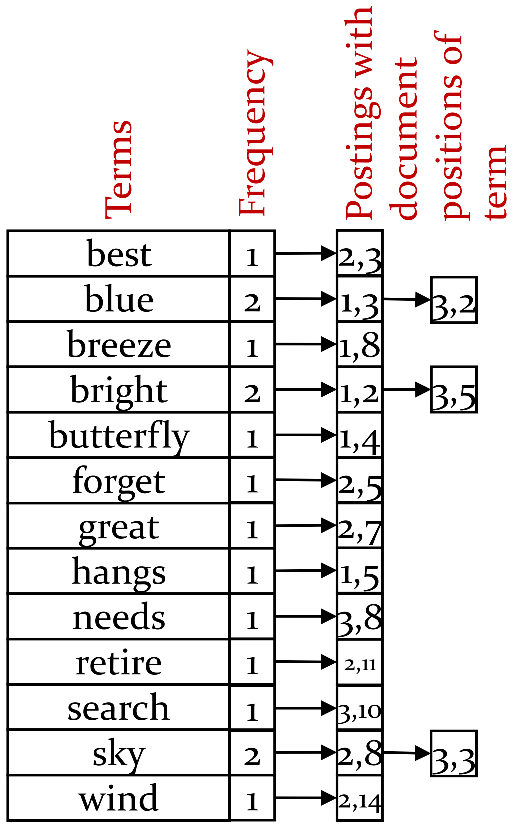
Building an Inverted Index

```
For each document d in the corpus
  For each term t in document d
     Find term t in the term dictionary
     If term t exists
       add a node to its posting list
     Otherwise,
       add term t to the dictionary
       add a node to the posting list
```

After all documents processed, save inverted index.

Extension to simple inverted indexes

- ***Dictionary***
  - May also include the frequency of each term in the dictionary (global frequency)
- ***Posting Lists***
  - Each Posting can also contain the position(s) of a specific term in the document
  - May help in supporting queries of Phrases (consecutive keywords: blue sky)
  - Helps to find words within a specified proximity



Dictionary (sorted)

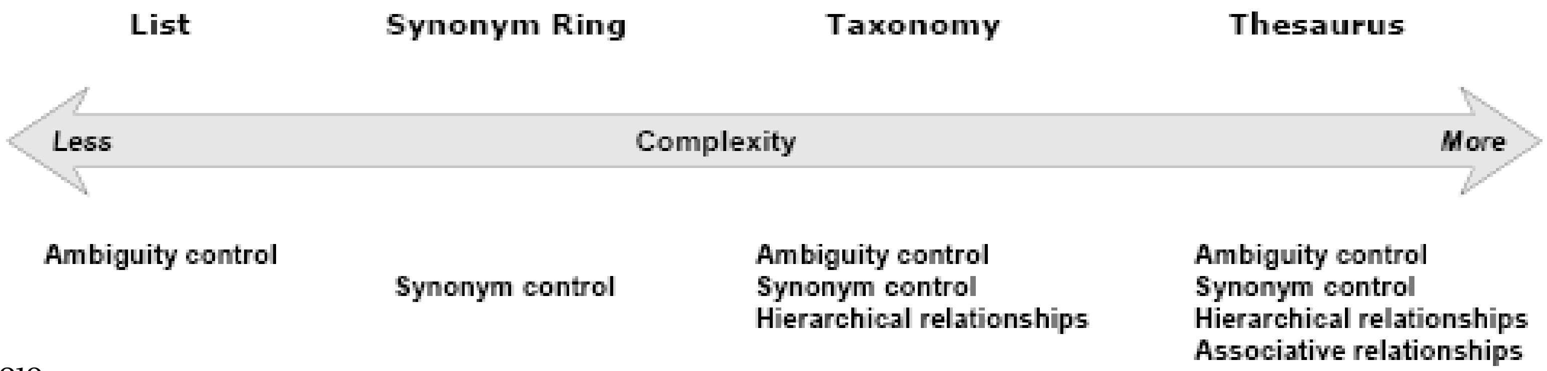
Indexing with Controlled Vocabulary

- Only approved terms can be used for indexing to ensure consistent indexing
- A **controlled vocabulary** is a **predefined list of terms** used to describe concepts covered in documents
- Controlled vocabularies **remove ambiguity** inherent in natural language
  - There are many ways to say the same thing. For instance *car, auto, vehicle, taxi, and Mercedes* all express the concept automobile.
- A controlled vocabulary can increase performance of an information retrieval system
- Lots of controlled vocabularies relevant to specific domains exists
  - **Example:** Medical Subject Headings (MeSH), controlled vocabulary used for indexing in PubMed.

Structures of Controlled Vocabularies

- Controlled vocabularies incorporate relationsships between terms
  - **Example**: synonym relationship **car** and **automobile**

- According to complexity of relationsship between four types of controlled vocabularies are identified
  - **Lists**
  - **Synonym Rings**
  - **Taxonomy**
  - **Thesaurus**

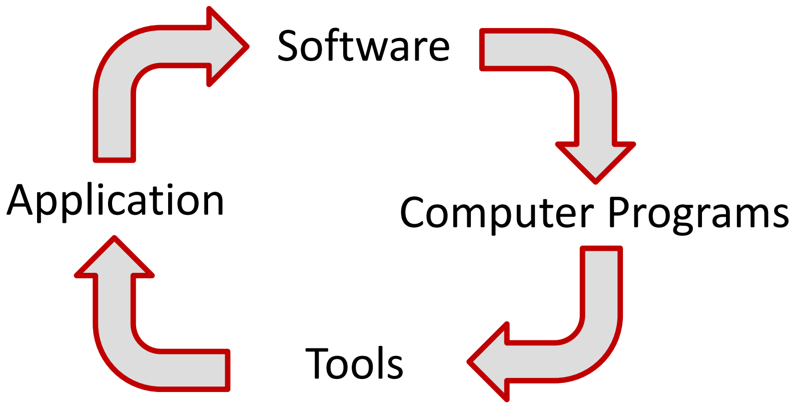


Conrolled Vocabulary – Lists

- **Lists are limited sets of terms** arranged as a simple alphabetical list or in some other logically way.
- Lists are frequently used to display small sets of terms that are to be used for quite narrowly defined purposes such as a web pull-down list or list of menu choices
- Example: (US-States)
  - *Alabama*
  - *Alaska*
  - *Arizona*
  - *Arkansas*
  - ………

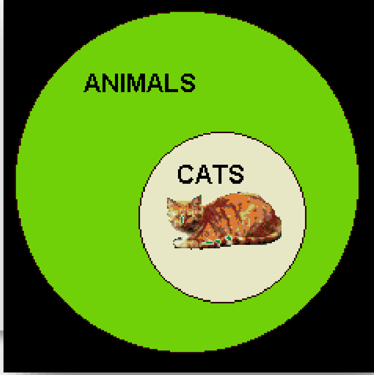
Conrolled Vocabulary – Synonym Ring

- **A Synomym Ring is a set of synonyms for each term**
  - none of synonyms is designated as the preferred term
  - known as a synonym ring or a synset because all synonyms are equal and can be expressed in a circular ring of interrelationships
  - Synonym Ring are common in search engines, but invisible to the user
  - frequently used behind-the-scenes to enhance retrieval, especially in an environment in which the indexing uses an uncontrolled vocabulary



Example for a synonym ring

Controlled Vocabulary – Taxonomy



- **A Taxnomy is a hierarchical classification system**
  - each term is connected to a designated broader term (unless it is the top-level term)
  - one or more narrower terms (unless it is a bottom level term)
  - supertype-subtype relationships (generalization-specialization relationships)
  - all the terms are organized into a single large hierarchical structure
  - may or may not make use of preferred terms
  - some hierarchical taxonomies permit terms to have multiple broader terms (polyhierarchical structure)

Controlled Vocabulary - Thesaurus

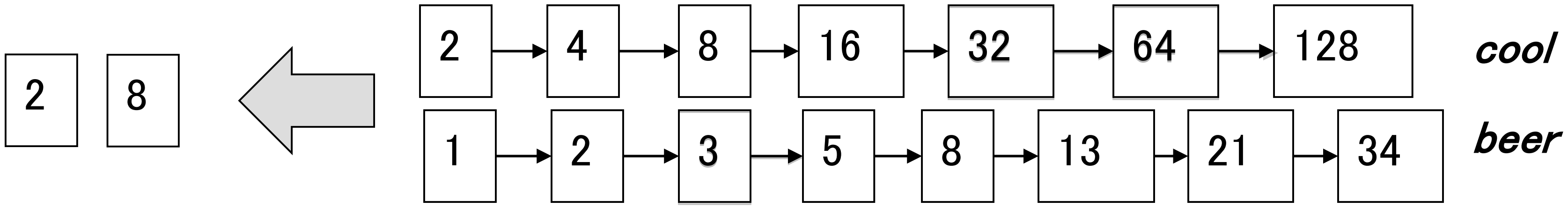
- **A Thesaurus typically includes the features of a taxonomy plus additional relationships**

- **Typical Relationships in a Thesaurus:**
  - hierarchical (broader term/narrower term),
  - associative (related term)
  - nonpreferred terms that redirect to the accepted term
  - equivalence (use/used for).
  - scope notes may be included to clarify usage

- **See Demo:** http://www.openthesaurus.de/

Boolean Queries

- The Boolean retrieval model is able to ask a query that is a Boolean expression.
- Keywords (e_i) combined with Boolean operators:
  - OR: (e_1 OR e_2)
  - AND: (e_1 AND e_2)
  - BUT: (e_1 BUT e_2) Satisfy e_1 but **not** e_2
- Negation only allowed using BUT to allow efficient use of inverted index by filtering another efficiently retrievable set.
- Naive users have trouble with Boolean logic



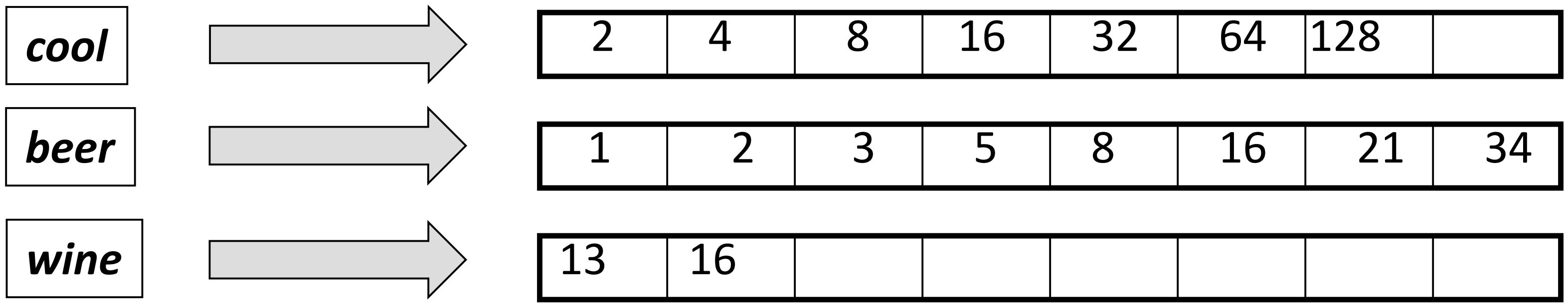
If the list lengths are *x* and *y*, the merge takes O(*x*+*y*) operations.

Boolean Retrieval with Inverted Indices

- One primitive keyword: Retrieve documents containing keyword (e_1) using the inverted index.
- OR: Retrieve e_1 and e_2 and take union of results.
- AND: Retrieve e_1 and e_2 and take intersection of results.
- BUT: Retrieve e_1 and e_2 and take set difference of results

Boolean queries: Query Optimization

- What is the best order for query processing?
- Consider a query that is an AND of n terms.
  - For each of the n terms, get its postings, then AND them together.
  - Process in order of increasing **frequency** (stored in dictionary)
    - start with smallest set, then keep "anding" further



Query: ***cool*** *AND* ***beer*** *AND* ***wine***

Execute Query: (***wine*** *AND* ***cool)*** *AND* ***beer***

Boolean Queries: Problems

- **Very rigid**: AND means all; OR means any.
- Difficult to express **complex user requests**
  - User must know Boolean Logic
- Difficult to **control the number of documents** retrieved.
  - *All* matched documents will be returned.
- Difficult to **rank output**.
  - *All* matched documents logically satisfy the query.
- Difficult to perform **relevance feedback**.
  - If a document is identified by the user as relevant or irrelevant, how should the query be modified

Scoring as the basis of ranked retrieval

- **Rank documents that are more relevant higher than documents that are less relevant.**
- Assign a score to each query-document pair, perhaps in [0, 1].
- This score measures how well document and query "match".

Simple approach: Jaccard coefficient

A commonly used measure of overlap of two sets:

- Let A and B be two sets, then the Jaccard coefficient is defined

$$\textit{Jaccard}\,(A,B) = |A \cap B| \,/\, |A \cup B|$$

- (with $A \neq \emptyset$ or $B \neq \emptyset$); Jaccard (A,A) = 1; Jaccard (A,B) = 0 if $A \cap B = 0$;
- A and B don't have to be the same size, always assigns number between [0,1]

- **Example:** Jaccard coefficient for Query (q): "The sky is blue" and Document (d): "sky hack download"
  $\rightarrow$ **Jaccard(q, d) = 1/6**

- **Problems with Jaccard coefficient:**
  - It doesn't consider term frequency (how many occurrences a term has).
  - Rare terms are more informative than frequent terms.

Statistical Models

- Use of statistical information in form of term frequencies to determine relevance of documents with respect to a query
- **As output produce a list of documents ranked by their estimated relevance**
  - vector space model
  - probabilistic retrieval model

Statistical Models

- Retrieval based on similarity between query and documents.
- Output documents are ranked according to similarity to query (score).
- Similarity based on occurrence frequencies of keywords in query and document.
- Relevance feedback can be supported

Statistical Models

- A document is typically represented by a **bag of words** (unordered words with frequencies). Words in the query appear frequently in the document, in any order.
- Bag = set that allows multiple occurrences of the same element.
- User can specify a set of desired terms:
  - Unweighted query terms:
    Q = < database; text; information >
  - No Boolean conditions specified in the query.

The Vector-Space Model

- Assume t distinct terms remain after preprocessing; call them index terms or the vocabulary.
- These "orthogonal" terms form a vector space.

  $$\text{Dimension} = V = |\text{vocabulary}|$$

- Each term, i, in a document or query, j, is given a real-valued weight, $w_{ij.}$
- Both documents and queries are expressed as t-dimensional vectors:

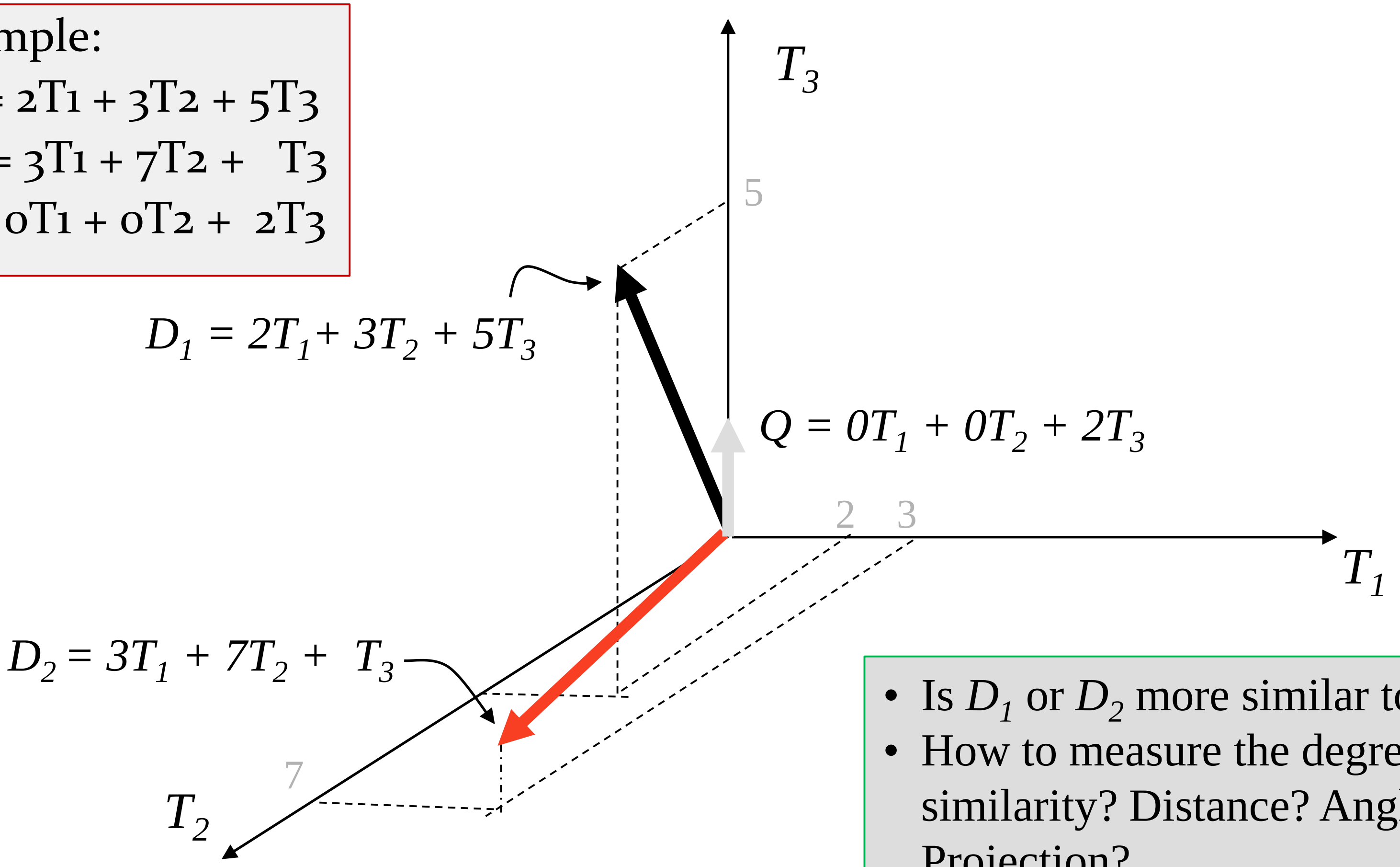
  $$d_j = (w_{1j}, w_{2j}, \ldots, w_{tj}) \text{ being } d_j \, document(j)$$

Documents as vectors

- Each document is now represented as a real-valued vector of weights ∈ $R^{|V|}$.
- So we have a |V|-dimensional real-valued vector space.
- **Terms are axes of the space.**
- **Documents are points or vectors in this space.**
- Very high-dimensional: tens of millions of dimensions when to apply this to web search engines
- Each vector is very sparse - most entries are zero.

Graphic Representation (3 Terms)

Example:
D1 = 2T1 + 3T2 + 5T3
D2 = 3T1 + 7T2 + T3
Q = 0T1 + 0T2 + 2T3



- Is D_1 or D_2 more similar to Q?
- How to measure the degree of similarity? Distance? Angle? Projection?

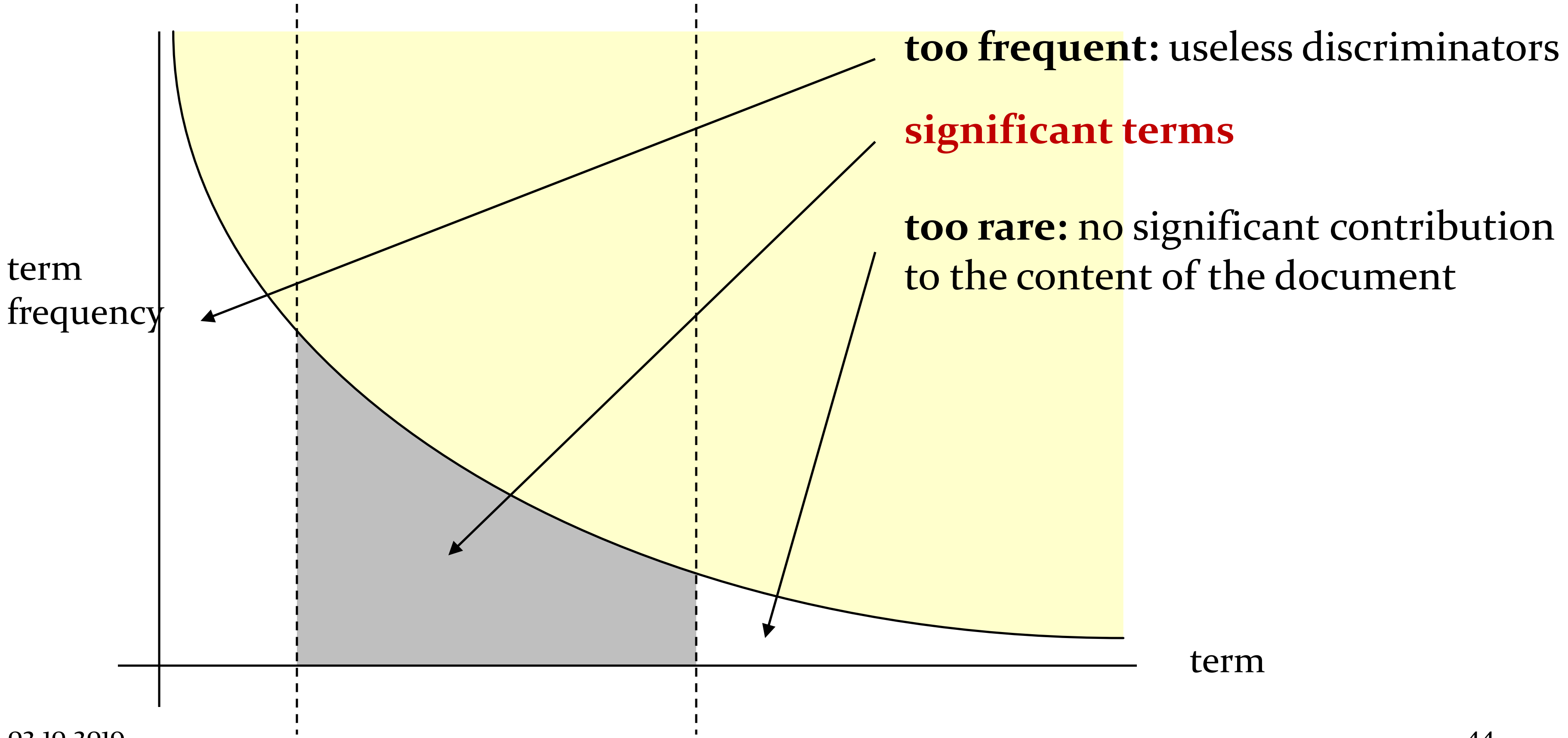
Document Collection

- A collection of n documents can be represented in the vector space model by a term-document matrix.
- An entry in the matrix corresponds to the "weight" of a term in the document; zero means the term has no significance in the document or it simply doesn't exist in the document.

$$\left(\begin{array}{ccccc} & T_1 & T_2 & \cdots & T_t \\ D_1 & w_{11} & w_{21} & \cdots & w_{t1} \\ D_2 & w_{12} & w_{22} & \cdots & w_{t2} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ D_n & w_{1n} & w_{2n} & \cdots & w_{tn} \end{array}\right)$$

Term Weighting

Term occurrence frequency is a measure for the significance of terms and their discriminatory power



too frequent: useless discriminators
significant terms
too rare: no significant contribution to the content of the document
term frequency
term

Term Weighting

- Term weights consist of two components
  - **Local:** how important is the term in this doc?
  - **Global:** how important is the term in the corpus?
- **Intuition:**
  - Terms that appear often in a document should get high weights
  - Terms that appear in many documents should get low weights
- **How do we capture this mathematically?**
  - Term frequency (local)
  - Inverse document frequency (global)

Term Weights: Term frequency *tf*

- The term frequency $tf_{t,d}$ of term t in document d is defined as the number of times that t occurs in d.
- We want to use *tf* when computing query-document match scores. But how?
- Raw term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with one occurrence of the term.
  - But not 10 times more relevant.
- Relevance does not increase proportionally with term frequency.

Term Weights:
Term Frequency *tf*

- More frequent terms in a document are more important, i.e. more indicative of the topic.

$$\boldsymbol{tf}_{ij} = f_{ij} \text{ ....frequency of term } i \text{ in document } j$$

- May want to **normalize** *term frequency* (*tf*) by dividing by the frequency of the most common term in the document:

$$tf_{ij} = f_{ij} \,/\, max_i\{f_{ij}\}$$

Term Weights: Document frequency *df*

- Rare terms are more informative than frequent terms. (Remember stop word removal)
- Consider a term in the query that is rare in the corpus (e.g., *arachnophobia*)
- A document containing this term is very likely to be relevant to the query *arachnophobia.*
- We want a high weight for rare terms like *arachnophobia.*

Term Weights: Document frequency *df*

- Consider a query term that is frequent in the collection (e.g., *high, increase, line*)
- A document containing such a term is more likely to be relevant than a document that doesn't, but it's not a sure indicator of relevance.
- → For frequent terms, we want positive weights for words like *high, increase, and line*, but lower weights than for rare terms.
- We will use ***document frequency (df)*** to capture this in the score.
- *df* ($\leq N$) is the number of documents that contain the term

Term Weights: Inverse Document Frequency *idf*

- Terms that appear in many *different* documents are *less* indicative of overall topic.

  df_i = document frequency of term i

  = number of documents containing term i

  idf_i = inverse document frequency of term i,

  $= \log_2 (N/df_i)$

  (N: total number of documents)

- An indication of a term's *discrimination* power.
- Log used to dampen the effect relative to *tf*.

TF-IDF Weighting

- A typical combined term importance indicator is *tf-idf weighting*:

$$w_{ij} = tf_{ij} * idf_i = tf_{ij} * \log_2 (N/ df_i)$$

- **A term occurring frequently in the document but rarely in the rest of the collection is given high weight.**
- Experimentally, *tf-idf* has been found to work well
- *tf-idf* weighting is the **most common term weighting approach** for information retrieval in the Vector Space Model.

Computing TF-IDF Example

Given a document containing terms with given frequencies:

T1(3), T2(2), T3(1)

Assume collection contains 10,000 documents and document frequencies of these terms are:

T1(50), T2(1300), T3(250)

Then (using normalized term frequencies):

T1: *tf* = 3/3; *idf* = $\log_2(10000/50) = 7.6$; *tf-idf* ≈ 7.6

T2: *tf* = 2/3; *idf* = $\log_2(10000/1300) = 2.9$; *tf-idf* ≈ 2.0

T3: *tf* = 1/3; *idf* = $\log_2(10000/250) = 5.3$; *tf-idf* ≈ 1.8

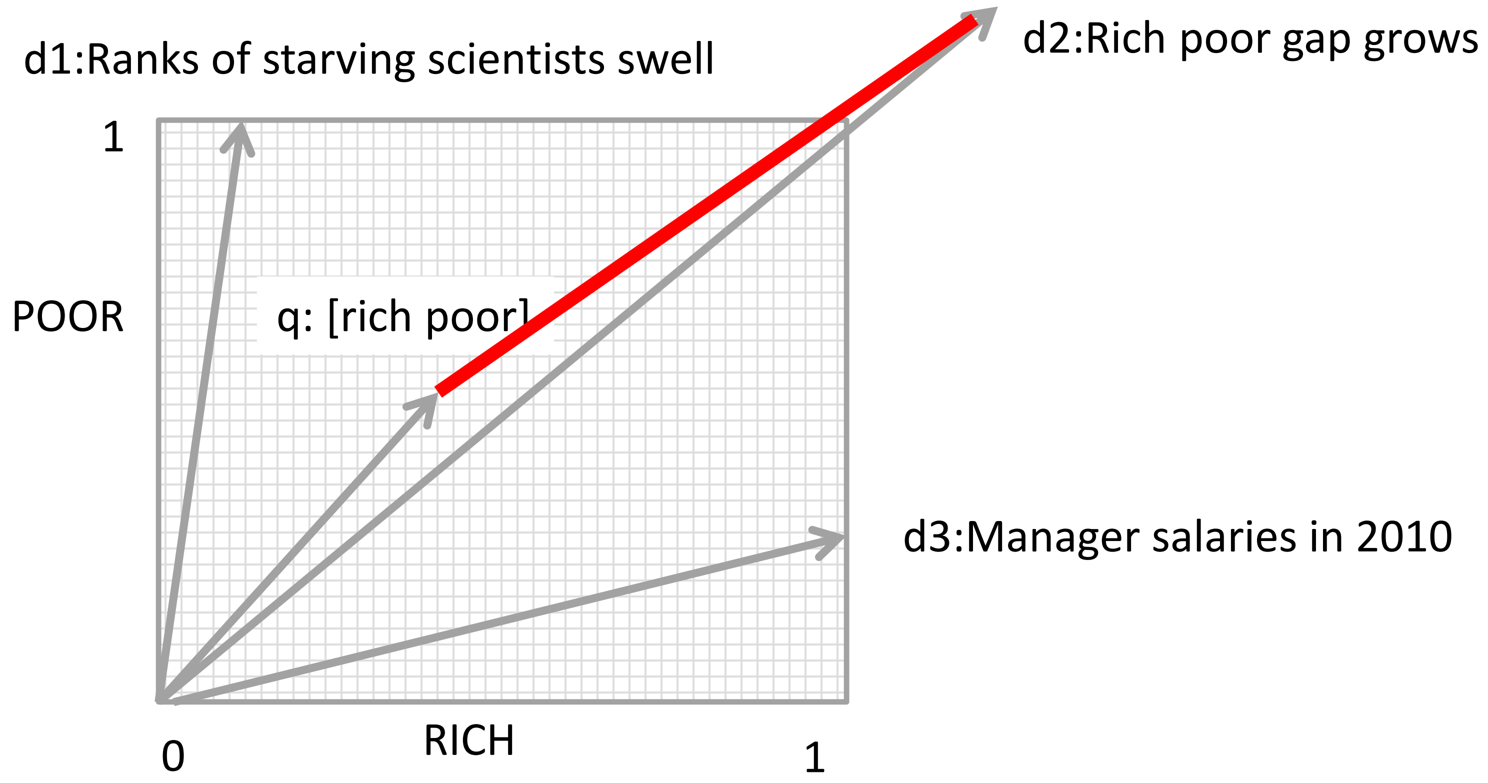
Query Vector

- A query vector is typically treated as a document and is also *tf-idf* weighted.
- Compute weights of query vector:
  - $w_{iq} = tf_{iq} * log_2 (N / df_i)$
  - tf_{iq} likely to be 1 as terms rarely used more than once in query
- Alternative is for the user to supply weights for the given query terms (not commonly used)

Similarity Measure

- A similarity measure is a function that computes the *degree of similarity* between two vectors.
- Using a similarity measure between the query and each document:
  - It is possible to rank the retrieved documents in the order of presumed relevance.
  - It is possible to enforce a certain threshold so that the size of the retrieved set can be controlled.

Similarity: Euclidean Distance


d1:Ranks of starving scientists swell
d2:Rich poor gap grows
1
POOR
q: [rich poor]
d3:Manager salaries in 2010
0
RICH
1

Problem: The Euclidean distance of q and d2 is large although the distribution of terms in the query q and the distribution of terms in the document d2 are very similar.

Similarity: Inner Product

- Similarity between vectors for the document $\boldsymbol{d}_i$ and query $\boldsymbol{q}$ can be computed as the vector inner product (dot product):

$$\mathrm{sim}(\boldsymbol{d}_j, \boldsymbol{q}) = \boldsymbol{d}_j \bullet \boldsymbol{q} = \sum_{i=1}^{t} w_{ij} w_{iq}$$

  where w_{ij} is the weight of term i in document j and w_{iq} is the weight of term i in the query

- For binary vectors, the inner product is the number of matched query terms in the document (size of intersection).
- For weighted term vectors, it is the sum of the products of the weights of the matched terms.

Similarity: Properties of Inner Product

- The inner product is **unbounded**.
- **Favors long documents** with a large number of unique terms.
- **Measures how many terms matched** but not how many terms are not matched.

Example (weighted)

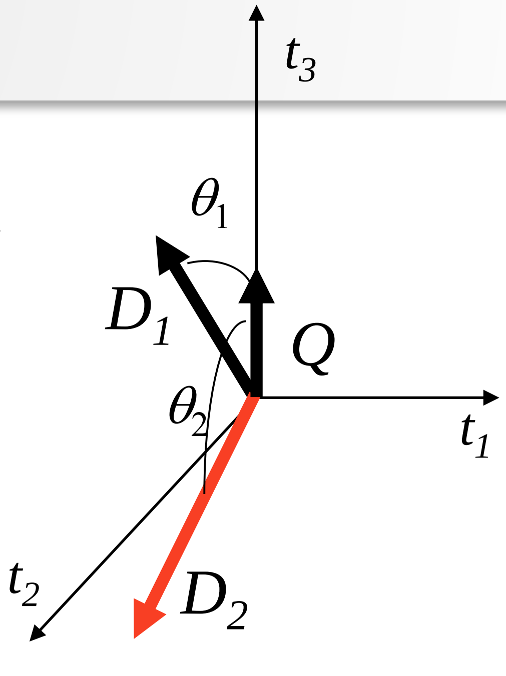
$$D_1 = 2T_1 + 3T_2 + 5T_3$$
$$D_2 = 3T_1 + 7T_2 + T_3$$
$$Q = 0T_1 + 0T_2 + 2T_3$$

$\text{sim}(D_1, Q) = 2*0 + 3*0 + 5*2 = 10$
$\text{sim}(D_2, Q) = 3*0 + 7*0 + 1*2 = 2$

Similarity: Cosine Similarity Measure

- Cosine similarity measures the cosine of the angle between two vectors.
- Inner product normalized by the vector lengths.

$$\text{CosSim}(\mathbf{d}_j, \mathbf{q}) = \frac{\vec{d_j} \cdot \vec{q}}{\left|\vec{d_j}\right| \cdot \left|\vec{q}\right|} = \frac{\sum_{i=1}^{t}(w_{ij} \cdot w_{iq})}{\sqrt{\sum_{i=1}^{t} w_{ij}^2 \cdot \sum_{i=1}^{t} w_{iq}^2}}$$



$D_1 = 2T_1 + 3T_2 + 5T_3$ CosSim(D_1 , Q) = 10 / $\sqrt{(4+9+25)(0+0+4)}$ = 0.81
$D_2 = 3T_1 + 7T_2 + 1T_3$ CosSim(D_2 , Q) = 2 / $\sqrt{(9+49+1)(0+0+4)}$ = 0.13
$Q = 0T_1 + 0T_2 + 2T_3$

D_1 is about 6 times better than D_2 using cosine similarity
…but only 5 times better using inner product.

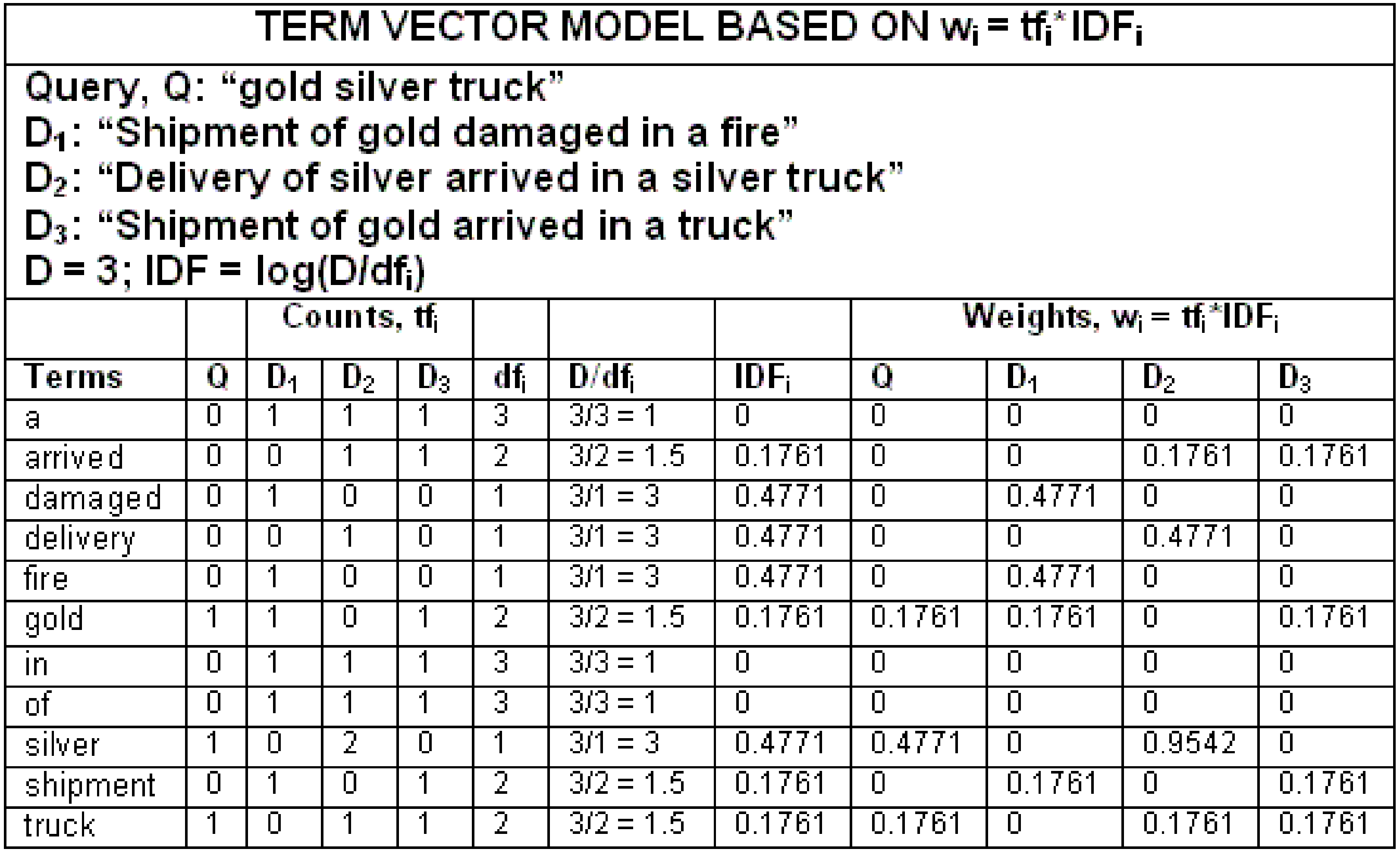
Similarity: Cosine Similarity Measure Why normalize by vector length?

- Longer Documents have:
  - **Higher term frequencies**: the same term appears more often
  - **More terms**: increases the number of matches between a document and a query
- → Long documents are more likely to be retrieved
- The "cosine normalization" lessens the impact of long documents, but favors small documents

Similarity: Example for Cosine Similarity

- From Professors David Grossman and Ophir Frieder, from the Illinois Institute of Technology in their book *Information Retrieval: Algorithms and Heuristics*.
- For simplification
  1. Do not take into account WHERE the terms occur in documents
  2. Use all terms, including very common terms and stopwords
  3. Do not use stemming
  4. Use raw frequencies for terms and queries (unnormalized data).

Similarity: Example for Cosine Similarity



TERM VECTOR MODEL BASED ON $w_i = tf_i{}^*IDF_i$

Query, Q: “gold silver truck”
D_1: “Shipment of gold damaged in a fire”
D_2: “Delivery of silver arrived in a silver truck”
D_3: “Shipment of gold arrived in a truck”
D = 3; IDF = log(D/df_i)

		Counts, tf_i						Weights, $w_i = tf_i{}^*IDF_i$			
Terms	**Q**	**D_1**	**D_2**	**D_3**	**df_i**	**D/df_i**	**IDF_i**	**Q**	**D_1**	**D_2**	**D_3**
a	0	1	1	1	3	3/3 = 1	0	0	0	0	0
arrived	0	0	1	1	2	3/2 = 1.5	0.1761	0	0	0.1761	0.1761
damaged	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
delivery	0	0	1	0	1	3/1 = 3	0.4771	0	0	0.4771	0
fire	0	1	0	0	1	3/1 = 3	0.4771	0	0.4771	0	0
gold	1	1	0	1	2	3/2 = 1.5	0.1761	0.1761	0.1761	0	0.1761
in	0	1	1	1	3	3/3 = 1	0	0	0	0	0
of	0	1	1	1	3	3/3 = 1	0	0	0	0	0
silver	1	0	2	0	1	3/1 = 3	0.4771	0.4771	0	0.9542	0
shipment	0	1	0	1	2	3/2 = 1.5	0.1761	0	0.1761	0	0.1761
truck	1	0	1	1	2	3/2 = 1.5	0.1761	0.1761	0	0.1761	0.1761

Similarity: Example for Cosine Similarity

1) For each document and query compute all vector lengths

$$|D_1| = \sqrt{0.4771^2 + 0.4771^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.5173} = 0.7192$$

$$|D_2| = \sqrt{0.1761^2 + 0.4771^2 + 0.9542^2 + 0.1761^2} = \sqrt{1.2001} = 1.0955$$

$$|D_3| = \sqrt{0.1761^2 + 0.1761^2 + 0.1761^2 + 0.1761^2} = \sqrt{0.1240} = 0.3522$$

$$\therefore\ |D_i| = \sqrt{\sum_i w_{i,j}^2}$$

$$|Q| = \sqrt{0.1761^2 + 0.4771^2 + 0.1761^2} = \sqrt{0.2896} = 0.5382$$

$$\therefore\ |Q| = \sqrt{\sum_i w_{Q,j}^2}$$

2) compute all inner products

$$Q \bullet D_1 = 0.1761 * 0.1761 = 0.0310$$

$$Q \bullet D_2 = 0.4771 * 0.9542 + 0.1761 * 0.1761 = 0.4862$$

$$Q \bullet D_3 = 0.1761 * 0.1761 + 0.1761 * 0.1761 = 0.0620$$

$$\therefore\ Q \bullet D_i = \sum_i w_{Q,j} w_{i,j}$$

3) calculate the similarity values

$$\text{Cosine } \theta_{D_1} = \frac{Q \bullet D_1}{|Q| * |D_1|} = \frac{0.0310}{0.5382 * 0.7192} = 0.0801$$

$$\text{Cosine } \theta_{D_2} = \frac{Q \bullet D_2}{|Q| * |D_2|} = \frac{0.4862}{0.5382 * 1.0955} = 0.8246$$

$$\text{Cosine } \theta_{D_3} = \frac{Q \bullet D_3}{|Q| * |D_3|} = \frac{0.0620}{0.5382 * 0.3522} = 0.3271$$

$$\therefore\ \text{Cosine } \theta_{D_i} = \text{Sim}(Q, D_i)$$

$$\therefore\ \text{Sim}(Q, D_i) = \frac{\sum_i w_{Q,j} w_{i,j}}{\sqrt{\sum_j w_{Q,j}^2}\sqrt{\sum_i w_{i,j}^2}}$$

4) rank the documents in descending order

Rank 1: Doc 2 = 0.8246
Rank 2: Doc 3 = 0.3271
Rank 3: Doc 1 = 0.0801

Similarity: Cosine Similarity - Observations

- Very frequent terms such as "a", "in", and "of" tend to receive a low weight (zero in this case).
    - **Model correctly predicts that very common terms, occurring in many documents in a corpus are not good discriminators of relevancy**
- Computationally expensive

Comments on Vector Space Models

- Simple, mathematically based approach.
- Considers both local (*tf*) and global (*idf*) word occurrence frequencies.
- Provides partial matching and ranked results.
- Tends to work quite well in practice despite obvious weaknesses.
- Allows efficient implementation for large document collections.

Problems with Vector Space Model

- **Missing semantic information** (e.g. word sense).
- Bag of Word Model: **Missing syntactic information** (e.g. phrase structure, word order, proximity information).
- Assumption of **term independence** (e.g. ignores synonomy).
- **Lacks the control of a Boolean model** (e.g., requiring a term to appear in a document).
  - Given a two-term query "A B", may prefer a document containing A frequently but not B, over a document that contains both A and B, but both less frequently.

Naive Implementation of Vector Space Model

1. Convert all documents in corpus D to *tf-idf* -weighted vectors, $\boldsymbol{d}_j$, for keyword vocabulary V.
2. Convert query to a *tf-idf*-weighted vector $\boldsymbol{q}$.
3. For each $\boldsymbol{d}_j$ in D do
   Compute score $s_j = \text{cosSim}(\boldsymbol{d}_j, \boldsymbol{q})$
4. Sort documents by decreasing score.
5. Present top ranked documents to the user.

Time complexity: $O(|V| \cdot |D|)$ Bad for large V & D !

$|V| = 10{,}000$; $|D| = 100{,}000$; $|V| \cdot |D| =$ **1,000,000,000**

Implementation based on Inverted Index

- In practice, document vectors are not stored directly; an inverted organization provides much better efficiency.
- Terms that are not in both, the query and the document, do not effect cosine similarity.
  - *Product of term weights is zero and does not contribute to the dot product.*
- Usually the query is fairly short, and therefore its vector is extremely sparse.
- Use inverted index to find the limited set of documents that contain at least one of the query words.

Inverted index Extension to VSM

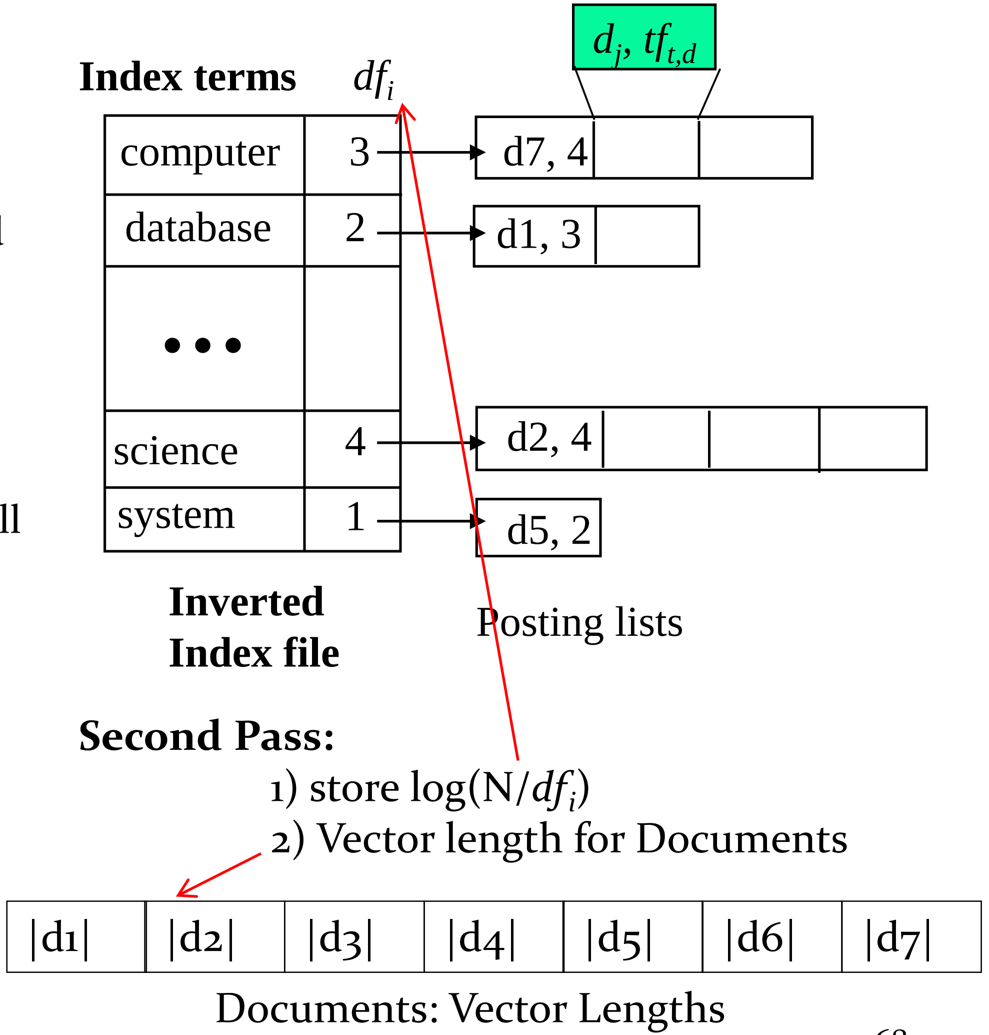
- Remember Inverted Index:
  - One entry for each term in the vocabulary
  - Documents containing Term are stored in the Posting List for Term:

    add corresponding term frequency for each document $tf_{t,d}$

    For each term, keep total number of occurrences in all documents: df_j
- A second pass through all the tokens after all documents have been indexed is required for:

    Computing IDF for each term: $\log(N/df_i)$ with N being the total number of documents in corpus

    Incrementally precompute vector lengths for all documents in Corpus for faster computation of similarity values later



$d_j, tf_{t,d}$
Index terms
df_i
computer 3
d7, 4
database 2
d1, 3
• • •
science 4
d2, 4
system 1
d5, 2
Inverted Index file
Posting lists
Second Pass:
1) store $\log(N/df_i)$
2) Vector length for Documents
|d1| |d2| |d3| |d4| |d5| |d6| |d7|
Documents: Vector Lengths