

**Dublin City University
School of Computing**

CA4009: Search Technologies

Section 3: Text Retrieval

Gareth Jones
October 2019

Introduction

The purpose of an *information retrieval* (IR) system is *to satisfy a user's information need*.

The IR system seeks to locate documents *relevant* to this information need.

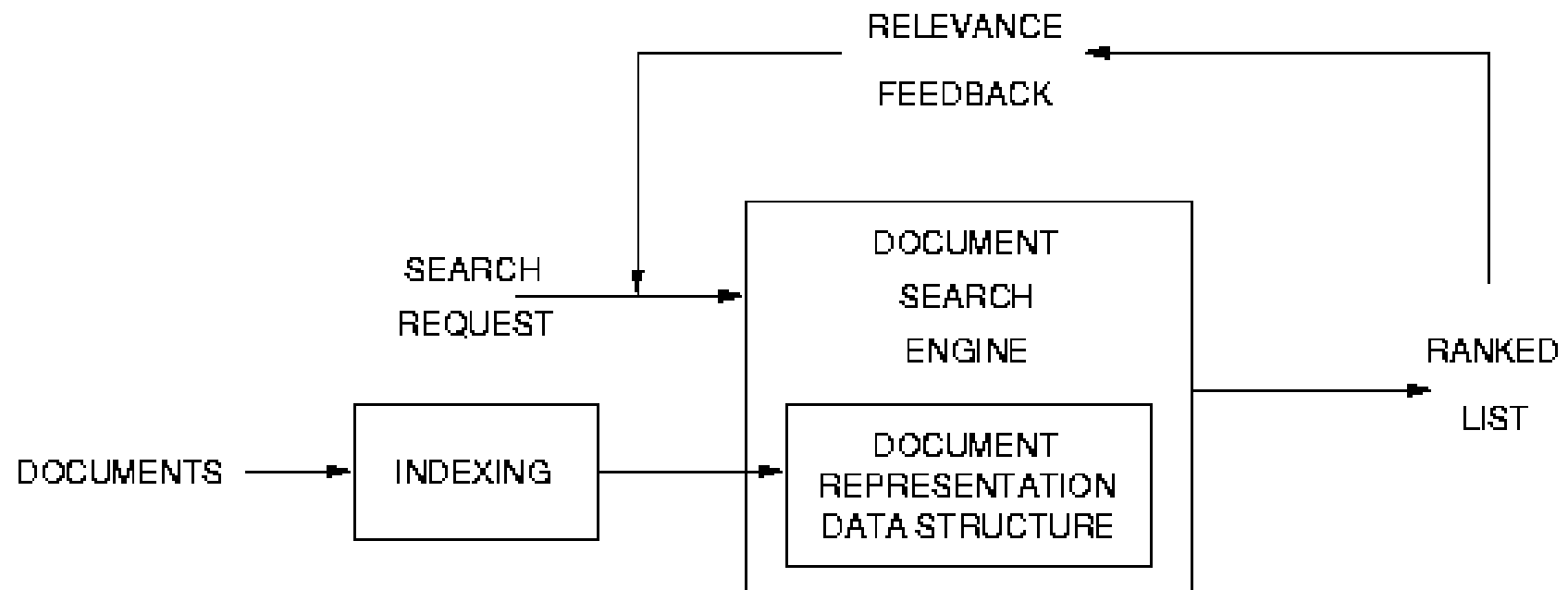
A standard text IR system attempts to do this based on the relationship between the contents of the user's search request and each potentially relevant document which is available in a document collection.

Many other factors can also be used as well to determine which documents the user might find most relevant or useful.

Use of some of these factors will be introduced later in the module, including hypertext structures, popularity with other searchers, quality of content, recency of updates, diversity of the contents delivered to the user, ...

Introduction

In this section, we focus on the fundamental challenge of document retrieval based on using a user search request to identify potentially relevant documents based on the contents of the documents.



Components of a standard text information retrieval system.

Components of an IR System

- Document Collection:

- published or professional documents, e.g. business reports, newspaper articles.
- web documents, collected using a web spider, crawler,
- social media contents, e.g. tweets.

The documents usually need to be preprocessed into a standard format, e.g. case conversion, removal of SGML/HTML or other markup, stop word removal, stemming.

- Document Indexing:

- convert documents into a file structure for rapid access, e.g. inverted file.

Components of an IR System

- Search Request:
 - user enters search request expressing their information need - apply similar text preprocessing as used for the documents .
- Document Searching:
 - calculate set of potentially relevant documents and return to user.
 - usually ranked by some score indicating likelihood of relevance, based on the match of the query with each document.
- Relevance Feedback:
 - modify search, e.g. expand query using extra search items, based on relevance data; and run the search again.

Introduction

Topics to be Covered in this Section

This section will introduce the following topics in information retrieval:

- indexing - text preprocessing
- file structures
- retrieval models: Boolean, best-match (ranked retrieval)
- relevance feedback
- distributed information retrieval
- multilingual information retrieval
- evaluation of information retrieval systems

The Nature of Text

Before considering how to perform IR, it is important to consider the general nature of text.

- Text is composed of **word tokens** taken from a surprisingly small vocabulary, e.g.
 - 10GB of web documents typically has around 160,000 unique words.Each token independently conveys some (often ambiguous!) meaning.
- Word **morphology** (e.g. tense, single/plural) is changed as words are combined into phrases and further combined into sentences when used in natural language.

For each language, there is a grammar of syntactically allowable word combinations to which phrases and sentences should conform.

The Nature of Text

- Properly formed sentences are constrained to the grammar to which they should conform.
- There is no way of enforcing the use of the accepted grammar of the language by the authors of documents.
- In formal publications an editor can seek to ensure writing standard, but may fail to do, so or may deliberately choose to break rules to create an effect with the use of language.
- There will also (almost inevitably!) be a number of typographical errors (spelling mistakes), even in published texts.

The Nature of Text

- Sentences are concatenated to make longer elements of **prose** which make up **documents**.
- Sufficiently large documents may be organised with structural elements, e.g. sections, chapters.
- These can be organised using hierarchies to ease navigation through the document.
- Documents may be linked via **hypertext** structures, as described in Section 2.
 - The linking of documents in this way has important implications for search of web and other content types.

The Nature of Text

For IR, we are concerned with the retrieval of elements of content from a collection of content items.

An important question is: what should constitute a search item?

Content may consist of:

- one single large structured document, e.g. an encyclopedia
- very many independent documents, e.g. news articles, legal or medical reports.
- very many connected documents, e.g. *wikipedia*, the web.

The Nature of Text

Many collections of content items may not fit easier into one of these categories,

- e.g. a collection of documents will often be partially connected.

People may disagree over the most appropriate connections that should be included.

In many situations it is not obvious what element should be retrieved,

- e.g. would you want to retrieve a complete encyclopedia or a whole website or a single webpage from the site?

Variations in Text Style

Texts are not the same:

- classical literature vs. modern prose
 - changes in vocabulary over time
 - standardised spellings are actually quite a modern feature of many languages
- spoken vs written language

There are many sub-languages, e.g.

- weather reports
- legal documents
- emails, blogs, microblogs (tweets), SMS

Variations in Text Style

Written text can take make forms, e.g.

- *technical documentation*: terse, tight prose, complex phrases and sentences because topic is complex.
- *journalism*: newspaper articles, often simple short easy to read sentences (will depend to some extent on your newspaper!)
- *storybook prose*: can be complex, but makes it difficult to read ... should be easy to assimilate, read for entertainment.
- *email messages*: ungrammatical, full of abbreviations, dialects and slang
... c u l8r!

Variations in Text Style

- *office memos*: grammatically correct, but not as complex as technical documentation.
- *formal language*: e.g. wills, deeds, legal documents, agreed domain specific use of vocabulary and grammar is vital.
- *Web pages*: wide variations in style and quality ... basically all of the above styles may appear.
- *blogs, microblogs*: informal varied style, very terse short statements.

Written vs Spoken Text

The structure and content of spoken language is very different from manually written content.

Consider the difference in language use between your own writing and speaking.

This has implications for how we search and interact with spoken content in a retrieval system.

What document unit should we search for?

What is the appropriate document unit to search for?

- Strictly only Individual files?
- What about an email file, consisting of many, sometimes chained. emails?
- What about email attachments?
- Converting a single Powerpoint file to HTML produces multiple webpages, what should we search for now?

What issues should we consider when deciding search units for a particular type of data?

What document unit should we search for?

The answers to these questions are often not straightforward, and may depend on the application in which the IR system is being used, or its context of operation.

The answer may be:

- a single file,
- multiple files dividing the content into smaller units,
- or perhaps both (dividing content in multiple ways at the same time in this way means that there can be more than one way of retrieving the same information).

Components of an IR System

Assume that the content collection to be searched has been divided into some agreed form of search units - the documents to be searched, e.g. news articles, web pages.

Assume also that we are building an IR system for English language documents.

Components of an IR System

A typical IR system will contain the following entities:

- Documents (or pointers to them!).
- List of *terms* – processed words used to index the Document contents
- Stored representations of the documents – generally representing full open vocabulary text.

Note in early IR systems storage limitations restricted representations to document titles/abstracts and/or a restricted keyword vocabulary.

Components of an IR System

Note:

1. This model does not take into account the user, i.e. the response of the IR system is the same for all users.
2. Assumes that the contents and characteristics of the document archive are static.

In many operational IR systems the data collection will be changing frequently, e.g. the list of pages and their contents indexed by a web search engine will be changing very frequently.

Indexing - text preprocessing

- The effectiveness and reliability of IR can be improved by careful content preprocessing.
- The large volume of data to be preprocessed generally means that preprocessing must be entirely automatic.
- Requirement for consistency of preprocessing (see the discussion of the Cranfield experiments in Section 1) also favours automatic indexing.
- Multiple stages of language specific automatic indexing are needed to ensure accurate and efficient retrieval, i.e. they will be different for English, French, etc.

Automatic Indexing

Documents can only be accessed via information stored about them.

The most simple approach would be to store the document contents and search these for matches with the words in the search request.

Semantically identical words may be instantiated differently in requests and documents, e.g.

- synonyms
- different word forms,

where this occurs a simple match between the words will fail.

Automatic Indexing

Semantic analysis of the documents and requests to interpret their meaning and then matching them somehow could address this problem.

In this scenario we would “understand” the contents of the documents and the request, i.e. we would know exactly what the documents are about and what the user is looking for.

We would then compare this the documents and the requests based on this information to select the relevant documents.

Such detailed comparison of documents and requests will be highly computationally expensive, and will often fail to sufficiently capture the meaning of the text to support useful comparison for information retrieval.

Automatic Indexing

The search requirements of an IR system are typically *underspecified* - the user's request does not give us a full precise description of what they are looking for.

So the IR system has to do as well as possible with the information available.

Thus, the IR systems needs to be designed to as far as possible reliably, identify potentially relevant document items within a broad scope of possible relevance.

Where it is understood that only the user is able to determine whether an item is actually relevant to their information need.

Fortunately, cheap and simple statistical methods, well suited to standard text IR, are available and functional effectively.

Automatic Indexing

An exception to this is generally when the request is in the form of a question looking for a specific piece of information:

“What is the capital of Ireland?”

If the user's request is identified as a question it can be passed to an augmented IR system referred to as a *question answering system*.

- see details in a later section of the module notes.

Automatic Indexing

- Extracting indexing units from text is referred to as *tokenization*.
- Tokenization is the process of chopping up and processing text strings into units that can form the basis of indexing units for retrieval.
- A *token* is a sequence of characters in a document that are grouped together into a meaningful semantic unit.
- For IR, an indexing unit may be a real “word”, but it may not, e.g. it may be a phrase or a part of a word.

Automatic Indexing

Typically we process extracted tokens in some way to form a more suitable indexing unit for IR.

A processed token to be used for search is referred to as a search *term*.

- The term units used for retrieval in an IR system forms the dictionary of the IR system.
- Note: the preprocessing of a token to form a stem”
 - may involve no change in the unit.
 - Tokens may be deleted and not be indexed (stop word removal) - see later.

Automatic Indexing

Processing of tokens to form search terms is often non-trivial.

- For specific domains there are unusual tokens that we may wish to recognize as terms.

For example, C++, C#, aircraft names such as B-52 or a television show such as M*A*S*H.

Also, computer terminology such as email addresses

jblack@gmail.com, URLS `http://www.dcu.ie`, numeric IP addresses (142.32.48.231), etc.

- What should we do about these?

Delete them? Delete individual characters from them? Implications?

Keep them? Implications?

Automatic Indexing

The key to token processing to create terms is to standardise the form of the token in both the documents and queries to encourage matching of related terms with the objective of maximising retrieval effectiveness.

The original form of token in the document and query may be identical, but often it is not.

Terms are intended to enable matching where the tokens are closely related, but they are different, which can occur for many reasons.

For clarity, we can distinguish between a search “request” entered by the user and the processed search “query” composed of search terms created by the preprocessing of the search request.

Term Utility in Retrieval

The objective of an IR system is to satisfy an user's information need, and it makes sense to seek to do this as efficiently as possible.

Efficient IR is most likely to be achieved if relevant documents are placed at the highest possible positions in a list of retrieved items, i.e. the ones that the user is most likely to look at first.

The ranking of documents depends on calculating the likelihood of relevance of each document based on comparison of the query and each document.

The question then arises: how should terms be used in this calculation and how they should be used?

The question really is, how can we use terms to distinguish likely relevant documents from non-relevant ones.

Term Utility in Retrieval

We can refer back to early work on text analysis.

i) Considering how often a term occurs in each document.

“It is hereby proposed that the frequency of word occurrence in an article furnishes a useful measurement of word significance.” (Luhn, 1957)

i.e. if a word occurs more often than expected in a document, the document is more likely to be related to the topic of this word and thus more likely to be relevant to a query containing this word

Term Utility in Retrieval

ii) Considering how frequent a term is within a collection of documents.

A good indexing term helps us discriminate one subset of a collection as relevant to a request, e.g. consider the usefulness of the terms “artificial intelligence” in a collection of documents about artificial intelligence, vs a general collection on computer science.

Good indexing terms reflect a relationship between an individual document and the collection from which it might be selected.

Such good indexing terms enable the selection of relevant documents for a query from among the non-relevant ones.

In practice, good keywords for retrieval are not the most frequent or the rarest, but rather those occurring a moderate number of times.

Term Utility in Retrieval

Let f be the frequency of occurrence of words in a document and r be the their rank order (the order of their frequency of occurrence).

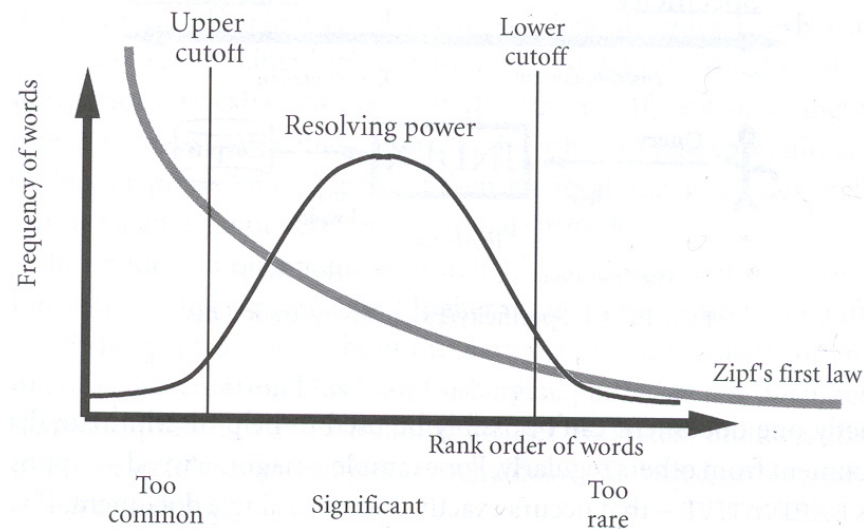


FIGURE 3.3 Resolving Power

Zipf's Law states that: $frequency(f) \times rank(r) = constant$.

Searching should use terms with maximum resolving power.

Stop Word Removal

Frequent words are shown to have low resolving power and intuitively are not of much benefit to the IR process,

They can thus be removed during indexing without impacting on retrieval effectiveness.

This removal of common terms is referred to as *stop word* removal.

Stop words are typically prepositions and conjunctions, e.g. *the, of, in, a, and, to, an*.

A list of stop words is usually chosen for a particular IR system by experiment.

Some standard lists are available, e.g. van Rijsbergen 1979.

Stop Word Removal

Removing stop words has two advantages:

- The computer memory requirement for document representations can often be reduced by more than 50% by removing only a small set of stop words.
- It can improve search efficiency since there is no matching of stop words which a user has included in their request.

Note: Removal of stop words will not interfere with the ranking of retrieved documents. Why not? Consider this point again taking account of use of *term weighting*.

Conflation

A conflation procedure is designed to bring together words that are related to each other in some way, e.g. stop, stopping, stops, stopped → stop.

We apply conflation methods to form standardised search terms.

The two main classes of conflation algorithms are:

- *stemming algorithms*
- *string-similarity measures.*

In both cases, the comparisons and processing resemblances are based on measures of character and character-substring similarities.

After conflation search tokens are referred to as search *terms*.

Conflation

Some examples of the need for term conflation are as follows:

- Incorrect spellings: LINEAR and LYNEAR
- Alternative spellings: SULPHUR and SULFUR
- Multi-word concepts: DATABASE and DATA BASE
- Transliteration: TCHEBYSHEFF and CHEBYSHEFF
- Affixes: SINGLE and SINGULAR
- Abbreviations and contractions: APPROX and APPROXN
- Historical changes: ARITHMETIC and AERITHMATICKE

Most of these problems can be addressed by string-similarity measures.

If the strings are similar they are probably related. What rules might be used for the above examples?

Affixes

Affixes are often removed to enable the matching different forms of semantically related words.

Some examples of common affix types are as follows:

Suffixes:

- CARRY, CARRIES
- ANALYSIS, ANALYTICAL
- INVERT, INVERSION
- FORGET, FORGETTING
- ABSORB, ABSORPTION

Prefixes:

- COGNITION, RECOGNITION
- POSSIBLE, IMPOSSIBLE
- COMPOSE, DECOMPOSE

While all these words are clearly related, we would generally regard only those differing in their suffix as semantically similar.

Stemming Algorithms

Stemming algorithms are designed to remove suffixes that inhibit matching.

A stemming algorithm operates by:

- matching the ending of a word against a suffix dictionary,
- removing any suffix that is identified,
- checking whether any context-sensitive rules apply.

The algorithm requires manual or automatic creation of a dictionary of suffixes, e.g. -ATION, -ING, -SES, etc.

In operation the end of each word in an input text or search request is compared against the suffix dictionary.

Stemming Algorithms

Two approaches to comparison:

- Longest-match: In a longest-match search, a word such as ALARMINGLY is stemmed to ALARM if both -INGLY and -LY are in the stem dictionary.
- Iterative search: In an iterative search, a word such as ALARMINGLY is first stemmed to ALARMING by removal of -LY and then to ALARM by removal of -ING.

Porter stemming dating from 1980 is an effective iterative stemming algorithm. It remains the most stemming popular algorithm currently in use.

Implementations of Porter stemming in several programming languages are available online for various natural languages, including English and French.

Stemming Algorithms

Context-Sensitive Rules

It is not always appropriate to apply stem removal.

When to apply stemming can be determined by the use of rules which examine the context in which the word is used.

Potential inappropriate stemming actions can be detected by means of a minimum stem length, e.g. -ATE and -ING should not be removed from CREATE and SING, respectively.

Alternatively context-sensitive rules may list exceptions to the general stem removal rules, e.g. rather than saying "do not stem to less than four letters". we could say to not stem the word SING.

Stemming Algorithms

Recoding can be used to allow for changes in spelling that occur in addition to removal of a suffix in order to enable matching of different word forms, e.g. INVERT and INVERSION or FORGET and FORGETTING, could appear as stems as INVERT or FORGET.

Manual evaluation of stemming algorithms suggests that around 95% of words can be processed to give reasonable stems.

Changing stemming rules to correct stemming errors tends to introduce different errors.

Retrieval experiments suggest that retrieval where word stemming is employed is generally more effective, but not greatly.

Stemming Problems

The following typical types of errors can be found in stemming algorithms:

Overstemming errors can lead to unrelated words being stemmed to the same term.

Word	Stem
MEDICAL	MED
MEDIA	MED
MEDIAN	MED

Over- and under- stemming of words that are semantically related can lead to failures to match during search.

Word	Stem
GASES	GAS (correct)
GAS	GA (over)
GASEOUS	GASE (under)

Automatic Indexing

Since document archives are often large the process of document *indexing* must be completely automatic.

- The original documents are preserved for retrieval.
- Stop word removal and conflation are designed to form representation of content which enable effective and efficient retrieval.

Information Retrieval for Other Languages

Most IR techniques were originally developed for English, since the majority of electronic texts were in English.

The amount of non-English electronic available has increased greatly in recent years, as a result of which interest emerged in developing effective retrieval for many other languages.

Many languages are in wide use across the world which have many different characteristics, the major languages needing to be processed include French, Spanish, German, Chinese, Japanese and Arabic, but there are many lesser used languages for which search needs to be supported including Basque in Spain, Flemish in Belgium, or Welsh or Irish.

Information Retrieval for Other Languages

In order to achieve IR in other languages, IR techniques developed for English need to be adapted to new languages.

Most IR algorithms actually make no underlying assumptions about the document language, and so need little or no adaption for use with different languages.

The earlier discussion about word distribution in individual documents and within document collections makes no assumptions of the language used in the documents.

However, the word level tokens and grammars of languages are highly varied, meaning that the preprocessing tokenisation and term creation need significant modification for IR in other languages.

Language Dependent Issues

Consider some examples of other languages.

French and Italian have similar characteristics to English, and can be processed in a similar manner to English.

We need to develop language specific stop word lists and conflation algorithms.

French:

- PERSON, PERSONNES
- DEUX, DEUXIEME
- AL, AUX

Italian:

- VOTARE, VOTAZIONI
- DELLA, DELLE
- RISULTARE, RISULTATO

Language Dependent Issues

German, Dutch:

- These are productive languages with respect to compounding of words from the language.
 - rare (or unique!) compounds in documents and requests are unlikely to match.
 - useful search terms must be extracted by decompounding.
- Decompounding of their component words ideally by looking up in a dictionary of compounds.
- If there is no dictionary entry for this compound word, search the letter string of the compound for known words of the language.
e.g. *Abendnachrichtensendungen*: Abend Nachricht Sendung
- Similar approach can be used for Dutch.

Language Dependent Issues

Chinese, Japanese:

- *agglutinating* languages, there are no spaces between words.
- Various indexing options:
 - extract single characters
 - fixed length n-gram character strings (typically 2-grams, 3-grams)
 - word-level segmentation using morphological analysis

Example of Japanese script

開始時間が午前 10 時の日経ビジネススクール

日経 = nikkei (Chinese characters (kanji))

ビジネススクール = biznesu sukuuru (phonetic transliteration)

Japanese has no notion of person (*I am = you are = they are = desu*) or singular or plural.

Language Dependent Issues

Arabic:

- Semitic language writing is right-to-left.
- alphabet of 28 letters; extended to about 90 using feature marks.
- complex inflection and morphology, suffix stripping is difficult.
- no space between some words and pronouns.
- prefixes and suffixes can be conjunction of two, three or four grammatical tokens.
- inconsistent transliteration of proper nouns.

Example of Arabic script

1. لكن الانفجار احدث فجوة كبيرة في الحافلة وتشاهد برك من الدم في المكان.

لوس انجليس = Los Angeles

Information Retrieval for Other Languages

In addition, two other issues can be considered in the area of multilingual search:

- Users may wish to access information contained in documents in languages with which they are not familiar.

We can use machine translation to cross the language barrier by translating queries or documents (which one is best? why?), in what is referred to as *cross-language information retrieval*.

- Document collections may contain information in more than one language. Retrieving from collections in more than one language is sometimes referred to as *multilingual information retrieval*.

Search File Structure

An important feature for users of an IR system is often the time taken for it to respond to their search request.

An important factor in determining the speed of an IR system response is the structure of the document representations.

The simplest way to compute the matching score between the query and each document would be to compare each query term against each terms in each document - but this will be hugely inefficient.

So, we want a data structure that will minimise the computational cost, and hence maximise the efficiency and speed of the query-document matching operation when a new request is entered.

Search File Structure

The speed of computing a query-document matching score will be maximised by considering only the terms in documents that have been specified in the request.

In order to be able to record the presence of each term in each document will take more time in preprocessing and indexing than simply tokenising and applying conflation to the content.

A data structure to store this information will also require more space than just storing the documents.

This is a classic data processing situation in computing where there is a trade off between space requirements, offline processing (indexing) and online processing (searching).

Search File Structure

In this case using more space and more complex offline processing enables faster and computationally cheaper online processing.

This is particularly useful in the case of search systems where the indexing only needs to be performed once^a to enable rapid processing of the (sometimes very large number of queries) when they are received.

For many years the principal file structure for IR systems has been the *inverted file*.

^abut of course needs to be incrementally updated if new documents are added or existing documents are revised or deleted.

Inverted File Example

Consider a very simple example. Suppose that a document archive consists of just 3 documents represented only by their titles.

D1. Information retrieval systems

D2. Database management systems

D3. Retrieval of information from computer systems

After stop word removal the following unique words remain:

Word	Word No
computer	T1
database	T2
information	T3
management	T4
retrieval	T5
systems	T6

Inverted File Example

Representing the documents in their original form produces:

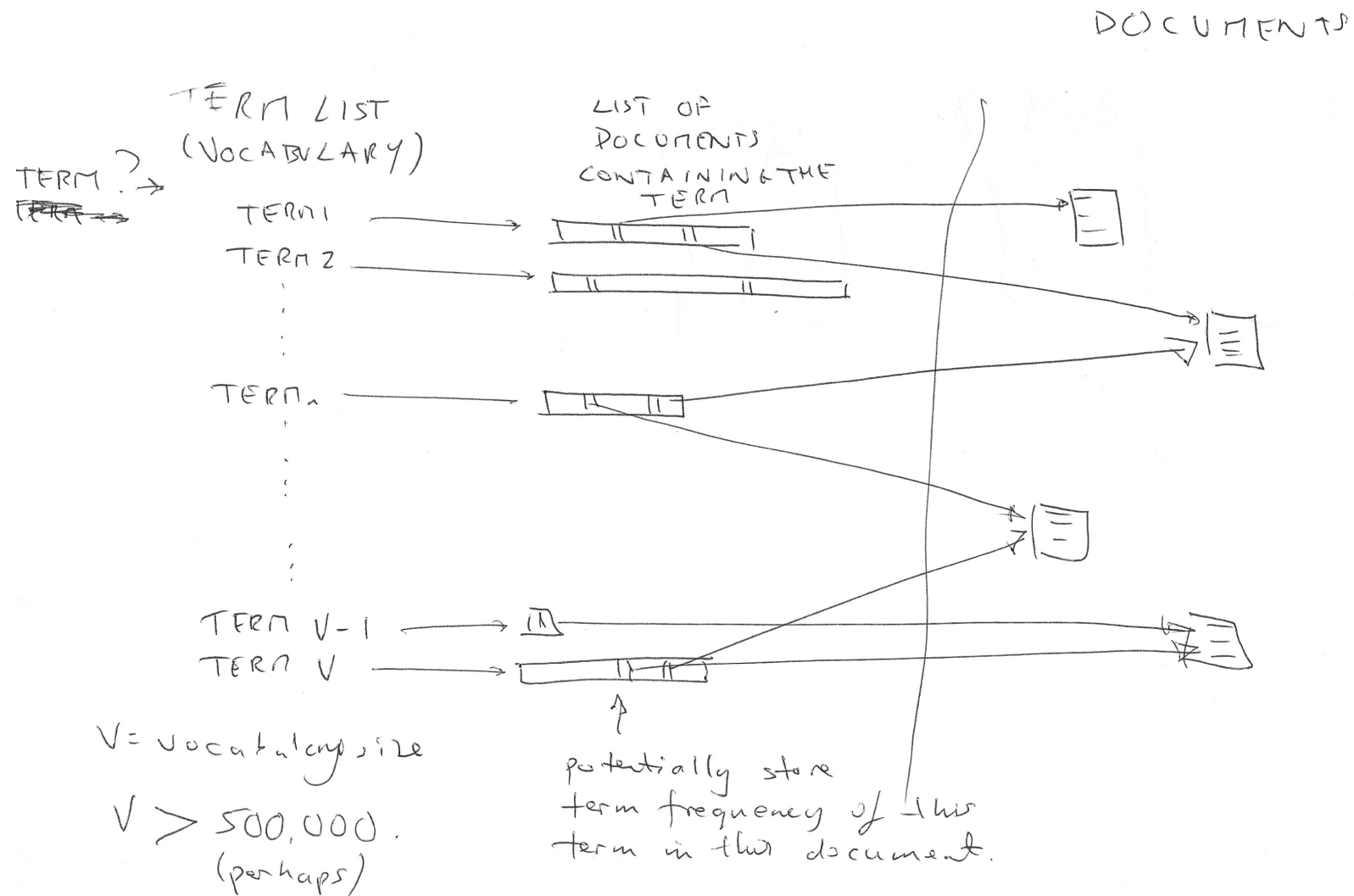
Document	Contents
D1	T3,T5,T6
D2	T2,T4,T6
D3	T1,T3,T5,T6

Representing the documents in an *inverted file* organisation produces:

Term	Document
T1	D3
T2	D2
T3	D1,D3
T4	D2
T5	D1,D3
T6	D1,D2,D3

Potentially relevant matching documents are identified using the list of terms.

Inverted File Example



Storing Term Locations

Merely knowing that a term is present in a document ignores potentially important information for the identifying relevant documents.

Many complex or technical concepts and many names are multiword compounds or phrases.

It is often useful to be able to search for these as a phrase rather than just look for documents containing the individual words.

Additionally, terms appearing in close proximity are likely to be semantically related (within 5 words of each other from Luhn's analysis of text documents again in the 1950s).

Storing Term Locations

Thus if query terms appear close together in one document, they are more likely to be related (in the sense of referring to the same thing) than in another document where they are far apart.

A document in which two adjacent terms in a query appear close together is more likely to be relevant to the user than a document in which they appear far apart.

Storing Term Locations

The term data in the inverted file can be extended to include the location of each term within each document, although adding this information will make the inverted file larger.

This can be used to incorporate term proximity into retrieval matching functions.

Storing proximity information can also allow for phrasal searching.

Storing this proximity information means that we can give a higher score to documents where the query terms are closer together than those where they are further apart.

Storing Term Locations

There are various ways of storing location information. Two options are:

Biword indexes

- Consider every word pair in a document as a phrase.
- Treat each word pair as a vocabulary term.
- Build an inverted file which stores details of which documents contain which biwords.
- Process longer phrases by breaking them down into pairs.

Storing Term Locations

- Extended biwords can be used for longer phrases which include stop words,
e.g. “cost overruns on a power plant” can be parsed to form the biwords “cost overruns” AND “overruns power” AND “power plant”
This works because words in a phrase will typically always contain the same stop words, so removing them will give the same matching pair.
- Biwords have space implications for the size of the inverted file!

Storing Term Locations

Positional Indexes:

Information retrieval systems

Database management systems

Retrieval of information from computer systems.

T1 D3 location of stop words?

computer → D3/5

database → D2/1

information → D1/1, D3/3

management → D2/2

retrieval → D1/2, D3/1

systems → D4/3, D2/3, D3/6

Storing Term Locations

- Rather than store phrases in an inverted file, in this case phrases can be identified when the query is entered.

Why is this a good idea?

Why might it be a bad idea?

Hybrid Schemes

- Some phrases can be efficiently looked up at search time. This will be the case if the individual terms are relatively rare, e.g. “Britney Spears”.
- Other phrases consisting of common words, e.g. the name “John Smith”, are better stored in a biword index.

Efficiency of Term Lookup

While in principle using the inverted file structure is very attractive, an operational question is:

How do we know where to find the terms appearing in queries within the inverted file?

The efficiency of term lookup up is vitally important for the effective use of inverted index files in IR.

Sequential string comparison of all words in the order in which they first occurred when processing the documents, i.e. the order in which they were added to the inverted file, will be very inefficient.

Efficiency of Term Lookup

Some improvement can be gained by sorting the vocabulary of the inverted file alphabetically, and using an array to indicate the first location in the inverted file list of a word with each leading letter.

However, this would still involve a lot of string comparisons, and hence be very inefficient.

Efficiency of Term Lookup

A much more efficient method is to compute term locations using a hashing algorithm or/and hash table.

The hashing algorithm should fulfil the following conditions:

- it must be repeatable;
- ideally have an even distribution;
- and minimise synonyms. Where synonyms occur string matching may be required in the inverted file to identify the correct search term.

Efficiency of Term Lookup

$T_2 \rightarrow \text{hashing function} \rightarrow \text{H value}$

Hash Value.

~~T_2~~

$T_1 \rightarrow H_2$

$T_2 \rightarrow H_3$

$T_3 \rightarrow H_2$

$T_4 \rightarrow H_6$

$T_5 \rightarrow H_1$

$T_6 \rightarrow H_4$

$H_1 \quad D_1, D_3$

$\rightarrow H_2 \rightarrow T_1, T_3$

$H_3 \quad D_3$

$H_4 \quad D_1, D_2, D_3$

$H_5 \quad \text{NULL}$

$H_6 \quad D_2$

different
hash

Preprocessing Summary

In summary,

- the search archive consists of document representations,
- (stop words removed and terms conflated)
- in an inverted file structure.

The inversion process is time consuming and computationally expensive, as well as expensive to maintain when the document collection changes.

- Consider what would be required to add or delete documents from an inverted file.

Retrieval Models for IR

Information retrieval researchers have developed a number of distinct models which seek to enable searchers to find relevant documents.

Operationally the main distinction is between Boolean and best-match searching models.

Early IR systems used a Boolean model. These systems were exclusively used by expert searchers such as librarians.

More recent work has focused on best match search models which, as we shall see, are better suited to non-experts such as web searchers.

Retrieval Models for IR

A number of distinct approaches to best-match search have been developed.

The best known are: the Vector-Space Model, the Probabilistic Model, Language Modelling and the Divergence from Randomness model.

We will look at the Vector-Space Model and the Probabilistic Model.

Many current IR systems are actually hybrids of Boolean and best-match searching.

Boolean Text-Retrieval Systems

Boolean retrieval models use AND, OR and NOT logical operators.

For example, retrieve documents which satisfy the following conditions: They contain: “information” AND “retrieval”, or “information” OR “retrieval”

Traditionally Boolean systems were almost always used by trained intermediaries (librarians) who chose the database to be searched, identified terms for inclusion in the query, formulated and submitted queries, refined them based on initial results and actually printed retrieved items for users or gave them details of a reference.

Boolean systems generally require the user to interactively build complex queries by incremental refinement over successive retrieval operations.

Boolean Text-Retrieval Systems

An essential piece of information for effective operation of a Boolean IR system is the *postings* information.

The postings value of a term is the number of entries that it has in the inverted file, i.e. how many documents within a collection it occurs in.

Recall (see earlier slides in this section) that for IR we are generally interested in terms which occur in a medium number of documents - not too many, and not too few - which have maximum resolving power.

A trained search intermediary can use the postings information to decide which terms might be useful to add to a query to identify documents potentially relevant to the information need.

Boolean Text-Retrieval Systems

(As we will see later, information of the number of documents that a term occurs in is also vitally important for effective retrieval with the vector-space and probabilistic IR models.)

By choosing an appropriate set of terms based on their understanding of the user's information need and the postings information, the search intermediary attempts to construct an effective query.

i.e. one that will retrieve the maximum number of relevant documents without retrieving large numbers of non-relevant documents.

Boolean Text-Retrieval Systems

Boolean systems have a number of limitations:

- High complexity of query formulation for multi-concept information needs.
- Lack of control over the size of the output set that is produced.
- No mechanism for ranking the output in order of decreasing likelihood of relevance.
- Once selected for inclusion in the query, all terms are taken to be equally useful in determining relevant documents.

Boolean Text-Retrieval Systems

This means that the searcher has not ordered way of deciding in which order to examine the retrieved items.

A very large number of documents may be retrieved, and it is not possible to select a preferred subset of the retrieved documents to examine.

In practice documents can be ranked by date of creation or filtered using metadata such as the source of the documents, author, publisher, project ID, etc.

Boolean Text-Retrieval Systems

Let U represent the set of names of all documents stored.

Let D_1 be the set of documents containing a pattern P_1 .

Let D_2 be the set of documents containing a pattern P_2 .

1. $U - D_1$ is the set of documents not containing P_1 . (NOT)
2. $D_1 \cap D_2$ is the set of documents containing both P_1 and P_2 . (AND)
3. $D_1 \cup D_2$ is the set of documents containing either P_1 or P_2 . (OR)
4. $D_1 \cup D_2 - D_1 \cap D_2$ is the set of documents containing either P_1 or P_2 , but not both. (XOR)

Boolean Text-Retrieval Systems

D1. algorithm, information, retrieval

D2. retrieval, science

D3. algorithm, information, science

D4. pattern, retrieval, science

D5. science, algorithm

The Boolean expression “information” is the names of all documents containing the term “information”: $\{D_1, D_3\}$

The expression “information AND retrieval” is
 $\{D_1, D_3\} \cap \{D_1, D_2, D_4\} =$

The expression “information OR retrieval” is
 $\{D_1, D_3\} \cup \{D_1, D_2, D_4\} =$

Best-Match Searching

Best-match IR corresponds to the situation of a ranked output list of potentially relevant documents with which we are familiar.

Ideally this list is ranked in decreasing likelihood of the document being relevant to the user's information need (as described by their query).

As stated earlier, there are a number of best-match IR ranking models. We look here at the *Vector-Space Model* and the *Probabilistic Model*.

The vector-space model and the probabilistic model have quite different derivations, but essentially do the same thing.

The similarity between a document and a request is a function of the number of search terms that they have in common.

Best-Matching Searching

Search *requests* in best-match search can be expressed in natural language which is then preprocessed to form a search *query* which is used for the actual matching.

Example of a search request and corresponding query.

“I am interested in novel matching algorithms for document-retrieval systems. Anything on best-match or nearest-neighbour retrieval, and partial-match searching might also be of interest.”

After preprocessing this produces the following search query,

algorithm, best, docum, match (3), near, neighb, novel, part, retriev (2), search, system

Since in this simple form of best-match search we focus only on matching of single terms, the request can just be a list of search terms rather than a natural language sentence - as many people enter to web search engines.

Best-Match Searching

Let us consider a simple example based on the query “information science” in the previous example.

If we score the documents based only on the number of terms which match between the query and each document, we have what is referred to as “coordination” level matching, and the following ranked list is produced.

Document	Score
D3	2
D1	1
D2	1
D4	1
D5	1

What is an obvious weakness of this approach?

Best-Match Searching

An effective best-match IR system requires:

- A weighting scheme to reflect the importance of different indexing terms in measuring the similarity.

Essentially this automates the role of the professional intermediary in determining effective terms for search, and incorporates this information into a ranking of the list of returned documents.

- A *matching score* between the query and each document.

This provides a similarity coefficient to provide a numerical quantification of the degree of resemblance between a query and a document.

This attempts to provide a meaningful ranking of documents in terms of their potential relevance.

Best-Match Searching

Advantages

- No need for complex Boolean queries - easier for novice users to use!
- Complete control over output size, since documents are ranked.
- Ability to differentiate between terms using term weighting schemes (see later).

Limitations

- The absence of phrase and multi-word terms which would be enabled by inclusion of an AND operator.
- The absence of the OR operator precludes the use of synonyms.
- A query needs to contain several terms to generate meaningful rankings.
 - Consider the problem of ranking documents in a web IR system.

Boolean vs Best-Match Searching

Comparative evaluation between Boolean and best-match searching is not straightforward.

Assuming that an information need is expressed separately as a Boolean query and a free text query and entered into Boolean and best-match IR systems respectively.

- How can ranked and unranked lists for a query be compared?
- How should differences in the size of the returned sets be handled?

Untrained searchers (i.e. typical users of web search engines) invariably achieve better results using best-match IR systems.

Professional searcher, e.g. patent examiners, legal organisations and librarians, continue to use Boolean search systems.

Boolean vs Best-Match Searching

Careful experimental investigation has shown that best-match searching retrieves more relevant documents and presents them to users at ranks which are likely to be browsed rather than any form of facet (e.g. date) ranking used for Boolean search.

The required computational resources are similar.

Boolean and best-matching can be used in various hybrid combinations, e.g. in advanced search options in best-match search engines, enabling OR (synonyms) and AND constructions to be used in queries.

Term Weighting

Simply summing the terms in common between the query and the document for best-matching, as used in the earlier example, is a very basic approach.

Better retrieval performance can be achieved by the introduction of *term weighting*.

Terms weighting enables terms with high utility in differentiating relevant documents to be emphasised.

The idea behind term weighting is *selectivity*: what makes a term a good one is whether it can pick relevant documents from among the many non-relevant ones.

Term Weighting

There are three commonly used components in terms weighting schemes:

- Collection Frequency
- Term Frequency
- Document Length

Collection Frequency

Key concept: Terms that occur in fewer documents are often more valuable than ones which occur in many documents.

Collection frequency weights (also known as *inverse document frequency weights*) are defined as follows for a search term $t(i)$.

Given

$n(i)$ = the number of documents term $t(i)$ occurs in,

N = the total number of documents in the collection archive.

The *cfw* for term $t(i)$ is

$$cfw(i) = \log \frac{N}{n(i)}$$

The logarithm can be taken to any convenient base.

Term Frequency

Key concept: The more often a term occurs in a document, the more likely it is to be important for that document.

This refers to the term's within-document frequency.

Thus, while a term $t(i)$'s collection frequency is the same for any document, its document frequency varies.

The term frequency for term $t(i)$ in document $d(j)$ is:

$tf(i, j) =$ the number of occurrences of term $t(i)$ in document $d(j)$.

Document Length

Key concept: Document relevance is *independent* of document length.

A term that occurs the same number of times in a short document and in a long document, is likely to be more valuable for the former.

Without compensating for document length, longer documents will tend to have higher matching scores merely because they are long, e.g. the $tf(i, j)$ will tend to be higher for long documents.

Document length of document $d(j)$ is

$dl(j)$ = the total number of term occurrences in document $d(j)$

One length compensation method requires a normalised average document length defined as follows:

$$ndl(j) = \frac{dl(j)}{\text{average } dl \text{ for all documents}}$$

“tf × idf”

One commonly used empirically derived weighting scheme is often referred to as “tf×idf”. (Note: $idf(i) = cfw(i)$.)

$$w(i, j) = f(tf(i, j)) * idf(i)$$

where $w(i, j)$ is the weight of term i in document j . Some $tf(i, j)$ functions,

$$f(tf(i, j)) = tf(i, j)$$

$$f(tf(i, j)) = 0.5 + 0.5 \frac{tf}{maxtf}$$

$$f(tf(i, j)) = \log(tf(i, j) + 1)$$

where $maxtf$ is the maximum term frequency in document j .

Vector-Space Model

In the vector-space model, both the documents and queries are represented as vectors in a t -dimensional space. where t is the number of unique index terms in the collection.

The degree of similarity between document $d(j)$ and a query q is calculated as the cosine of the angle between the two vectors.

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \cos \theta$$

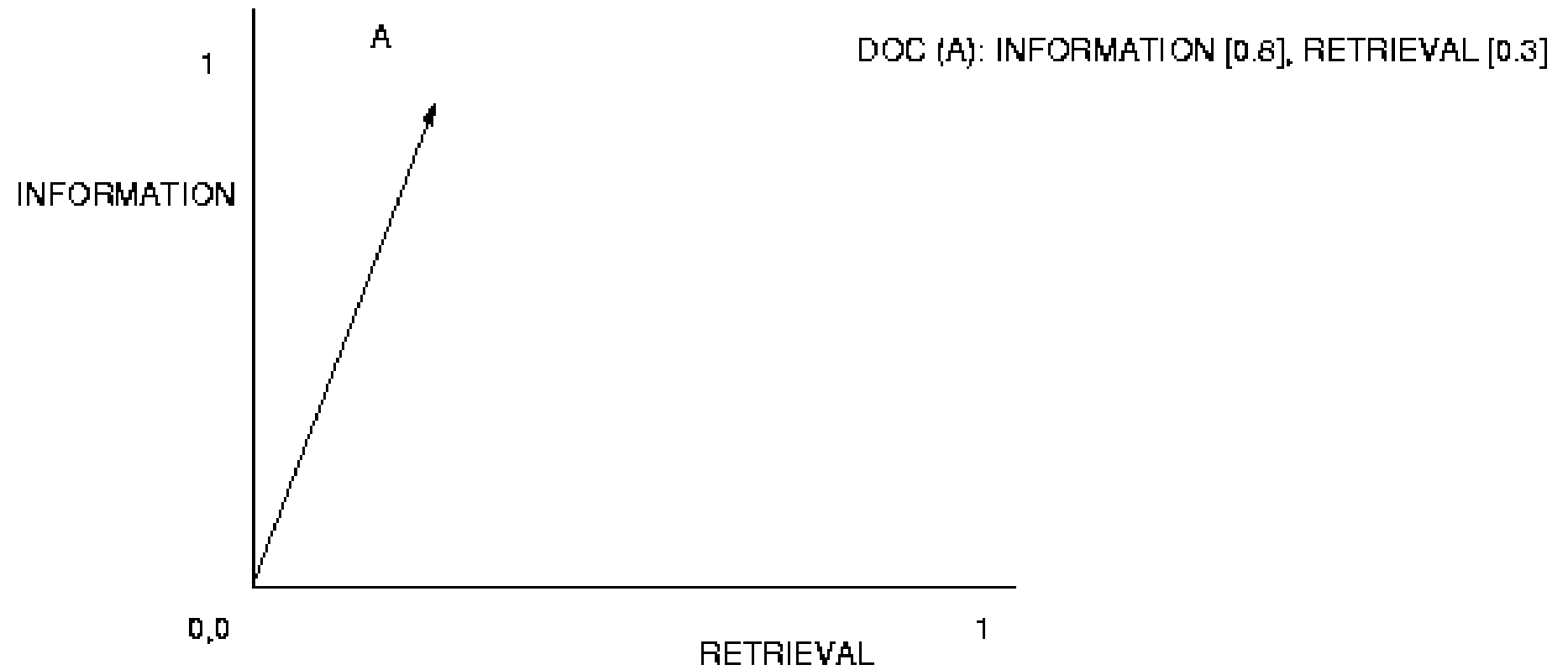
Vector-Space Model

The similarity between the query and each document $d(j)$ can be computed as,

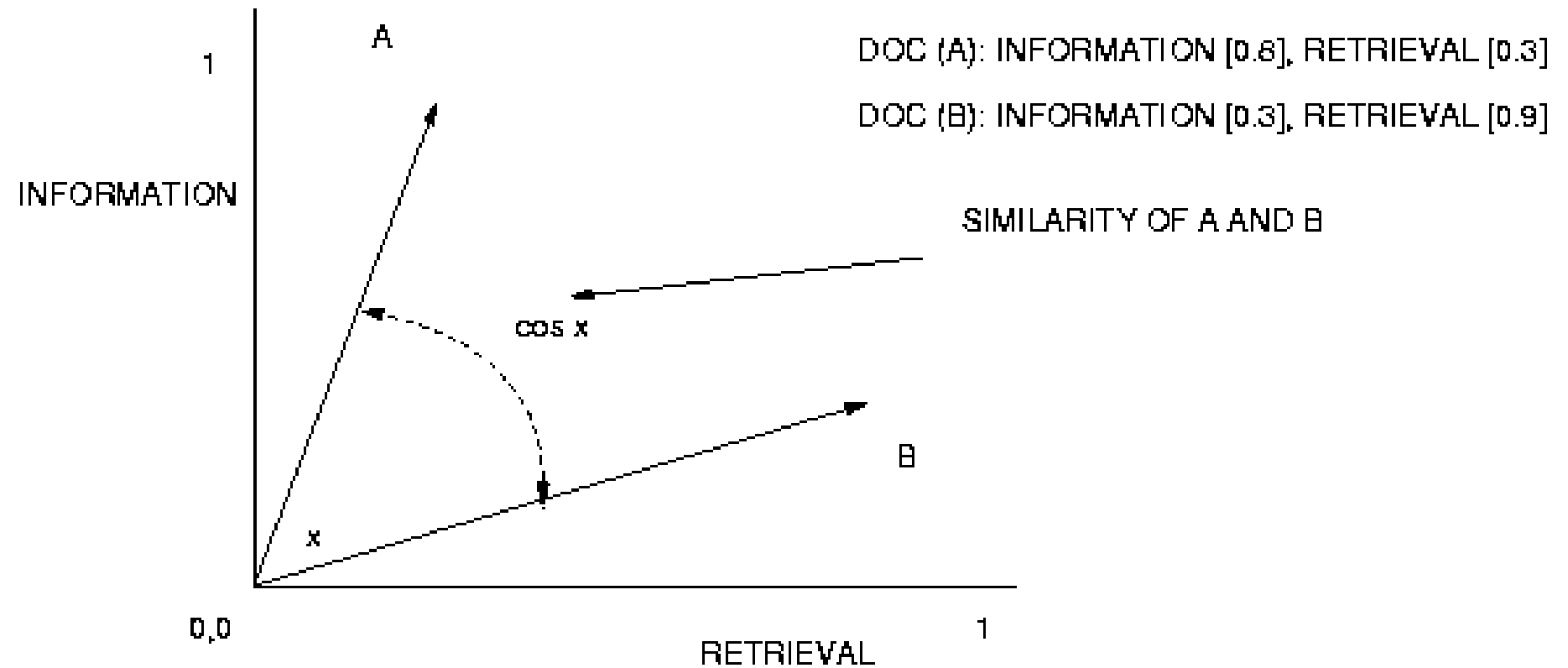
$$Sim(q, d(j)) = \frac{\sum_{i=0}^{I-1} w_q(i) \cdot w_d(i, j)}{\sqrt{\sum_{i=0}^{I-1} w_q(i)^2} \sqrt{\sum_{i=0}^{I-1} w_d(i, j)^2}}$$

The documents are then ranked in decreasing order of similarity.

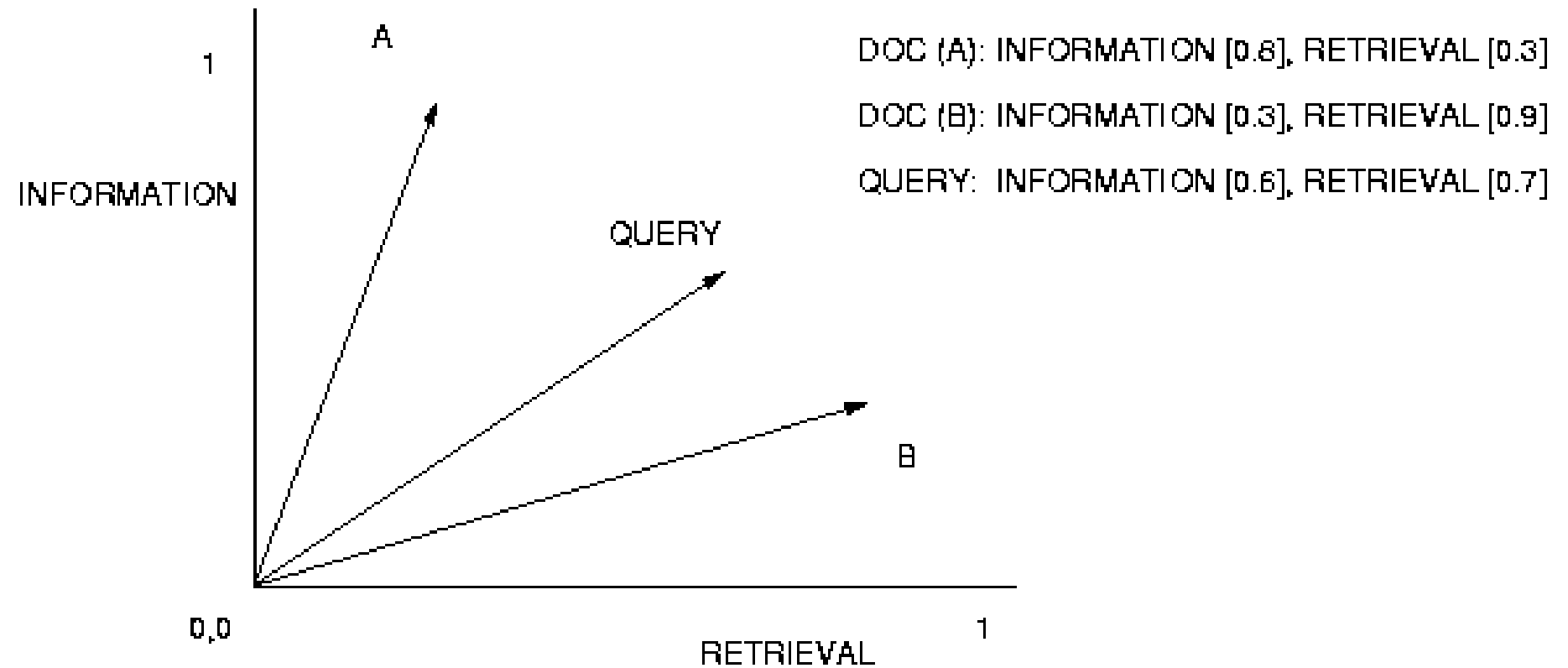
Vector-Space Model



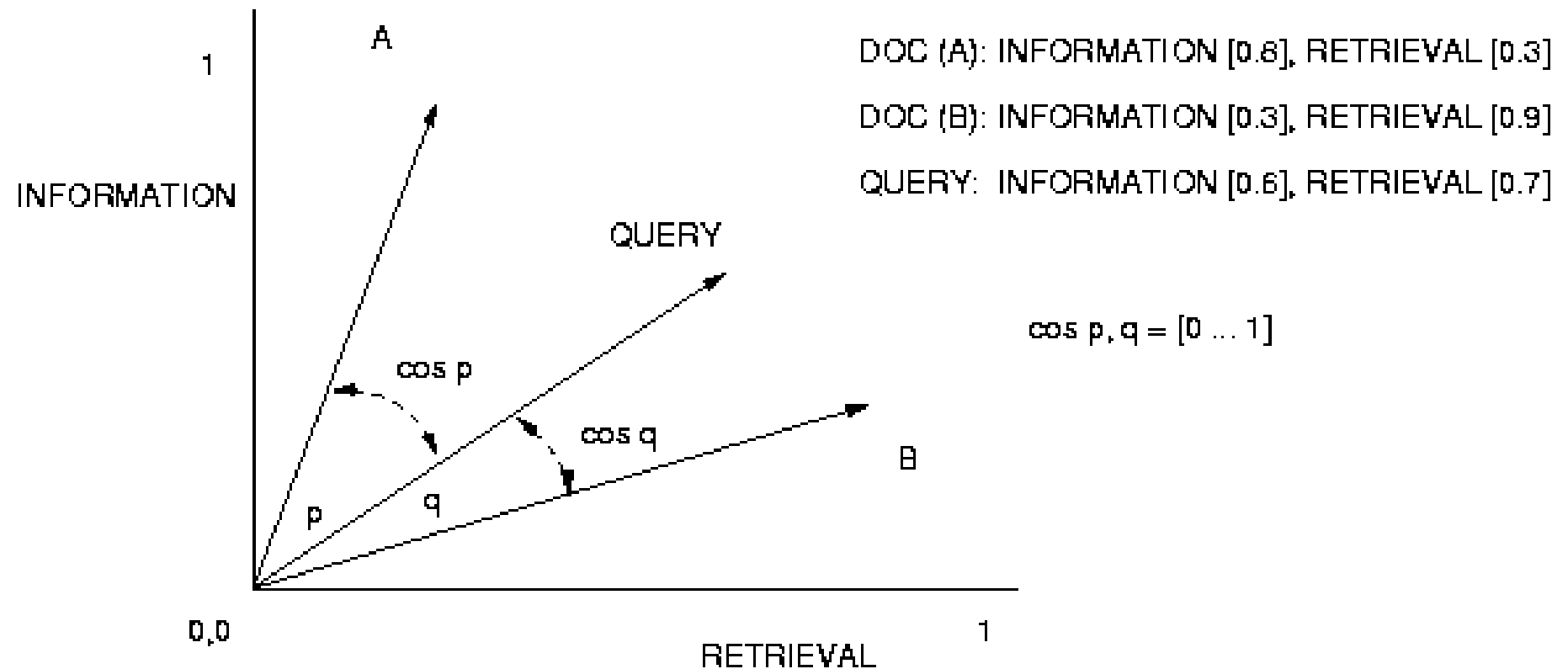
Vector-Space Model



Vector-Space Model



Vector-Space Model



$\cos(q) > \cos(p)$ so B is more similar to the QUERY than A, hence should be ranked higher in a ranked retrieval list.

Probabilistic Model

Although the vector-space model employs frequency counts, its underlying mathematics as a retrieval model are fairly ad hoc.

The matching score assigned to a document in a result set is not a probability of relevance, but rather a measure that attempts to estimate how much evidence there is in favour of this document being relevant.

The *probabilistic model* of retrieval is an attempt to formalize the idea behind ranked retrieval in terms of probability theory.

It attempts *to compute the probability that a document is relevant to a query*, given that it possesses certain attributes or features.

- In the basic case that we will consider, these attributes related to whether they each contain certain search terms.

Probabilistic Model

In the derivation of the probabilistic model it is generally assumed that:

- each document is either relevant or irrelevant to the query.
- judging one document to be relevant or irrelevant tells us nothing about the relevance of another document.

The *Probability Ranking Principle* states that ranking documents by decreasing order of probability of relevance to a query will yield “optimal performance”, i.e. the best ordering possible based on the available data.

Probabilistic Model

The probabilistic model query-document matching score $ms(j)$ for a document j can be calculated in a similar manner to the other ranked retrieval schemes that we have already seen.

$$ms(j) = \sum_{i=0}^{I-1} w(i, j)$$

where $w(i, j)$ indicates a probabilistic term weighting scheme, and I is the set of all search terms.

This can be shown as part of the derivation of the probabilistic model to be a valid means of calculating the probability of the relevance of a document to a particular query.

Probabilistic Model

The evidence available as a description of the information need is the user search request.

This is an imperfect or incomplete description of the information need, and so any ranked list produced cannot be completely reliable, it is based on probabilities calculated using the available evidence.

In addition, derivation of probabilistic models makes various simplifying assumptions in the mathematics, so that the calculated probabilities are only approximation to their true values.

Probabilistic Model

Derivation of a probabilistic model is beyond the scope of this module, but we can still examine and use a derived probabilistic model.

One effective probabilistic model is the *Okapi BM25 combined weighting* $cw(i, j)$ scheme developed at City University, London:

$$cw(i, j) = cfw(i) \times \frac{tf(i, j) \times (k_1 + 1)}{k_1 \times ((1 - b) + (b \times ndl(j))) + tf(i, j)}$$

where $cw(i, j)$ indicates the combined weighting scheme, $ndl(j)$ is the normalised length of document j , and k_1 and b are experimentally determined constants that control the effect of $tf(i, j)$ and the degree of length normalisation respectively.

Okapi BM25

Let us look at the components of the Okapi BM25 term weighting function in a little detail.

First notice that it is composed of a $tf \times idf$ type function, since $cfw(i)$ is the same as $idf(i)$ and the other part of the function is a function $tf(i, j)$.

The k_1 factor determines the impact of term frequency in a document. A typical value for k_1 would be 1.5.

b determines the degree of document length normalisation. The value of b can vary in the range 0 to 1.

Okapi BM25

Let us consider the two extreme cases of $b = 0$ and $b = 1$.

$b = 0$

$$cw(i, j) = cfw(i) \times \frac{tf(i, j) \times (k_1 + 1)}{k_1 + tf(i, j)}$$

Here there is no adjustment of the term weight to take account of document length.

Longer documents will tend to have higher term weights, since on average they will have higher $tf(i, j)$ values.

That is, $cw(i, j)$ for a term t will on average have a higher value if the document j is a long document.

Okapi BM25

$$b = 1$$

$$cw(i, j) = cfw(i) \times \frac{tf(i, j) \times (k_1 + 1)}{k_1 \times ndl(j) + tf(i, j)}$$

Here there is maximum application of adjustment of term weights to account for document length.

Shorter documents will have $ndl(j)$ values less than 1.0 - leading to an increase in the $cw(i, j)$ value compared to a document of average length.

Longer documents will have $ndl(j)$ values greater than 1.0 - leading to a decrease in the $cw(i, j)$ value compared to a document of average length.

Okapi BM25

The overall effect should be to provide a degree of compensation to document matching scores $ms(j)$ based on document length, i.e there should be less tendency to score longer documents higher, and so less tendency to place them nearer the top of a ranked retrieval list.

Note the b value is typically set between 0.0 and 1.0 to give optimal retrieval effectiveness for a document collection using a set of training queries and corresponding relevance assessments.

Retrieval effectiveness can be measured using precision, recall or another standard IR evaluation metric.

Okapi BM25

Returning to the case of $b = 0.0$.

$$cw(i, j) = cfw(i) \times \frac{tf(i, j) \times (k_1 + 1)}{k_1 + tf(i, j)}$$

We can see that the function increases more slowly with respect to $tf(i, j)$ for large values of k_1 and more quickly for smaller values of k_1 .

When $tf(i, j) = 1$, $cw(i, j)$ is simply equal to $cfw(i)$, i.e. there is no impact of the $tf(i, j)$ function here when a term occurs in a document only once.

Example

Simple example of document scoring for a search query.

Consider a query consisting of 3 terms: $t(1)$, $t(2)$ and $t(3)$.

Suppose the document content representations are stored in an inverted file accessed via a hash table.

Each of the search terms must first be transformed to the corresponding hash table entry using the selected hash algorithm.

Example

Thus,

$$t(1) \rightarrow x(1)$$

$$t(2) \rightarrow x(2)$$

$$t(3) \rightarrow x(3)$$

In the example,

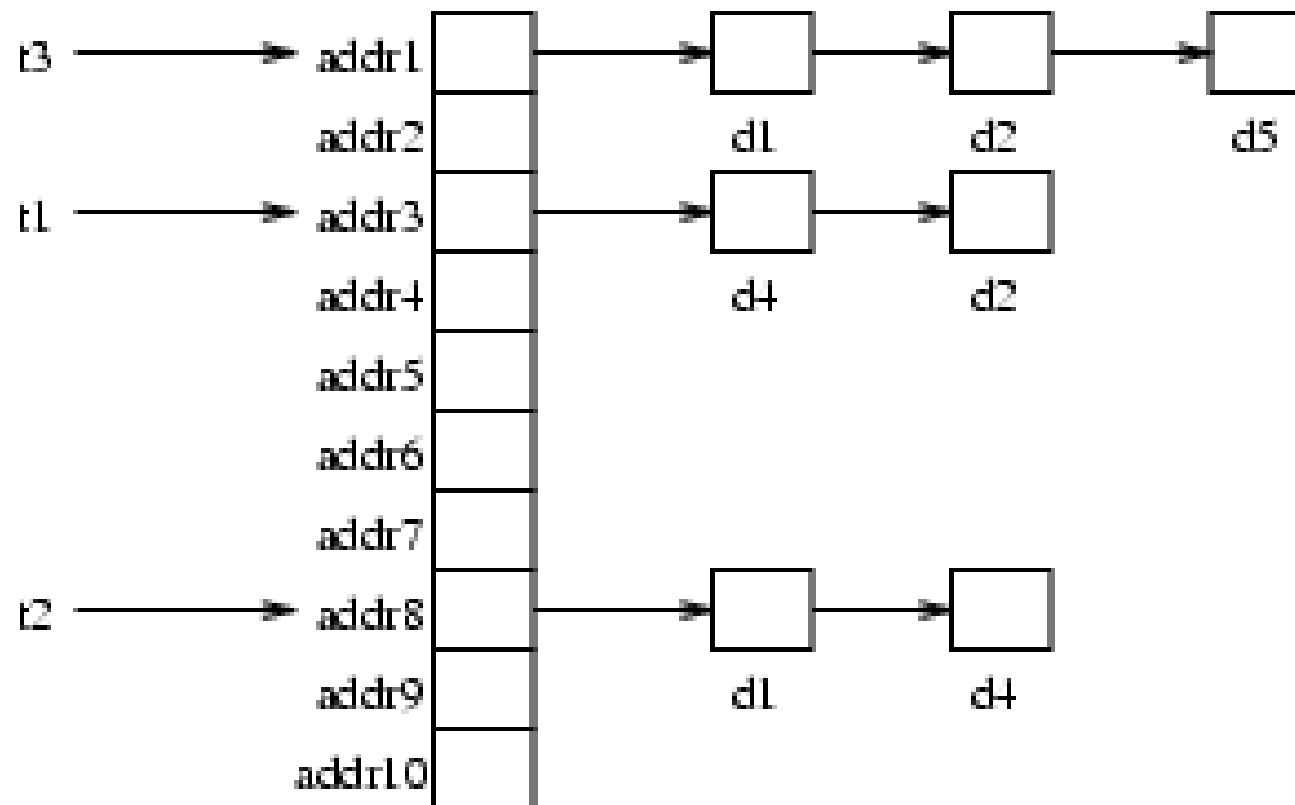
$$t(1) \rightarrow addr(3)$$

$$t(2) \rightarrow addr(8)$$

$$t(3) \rightarrow addr(1)$$

Example

addr[1-10] - hash addresses



Example

Let the matching score between the query and document $d(j)$ be $ms(j)$.

Initially $ms(j)$ is set to 0.0 for all documents j .

For each term $t(i)$ the hash table entry $x(i)$ is evaluated.

The hash table connects to the term entries in the inverted file structure.

Entries in the inverted file can simply be connected using a linked list where the entry for one document points to the address of the entry for the next document in the list.

Example

When each matching document $d(j)$ in the linked list for term $t(i)$ is reached the matching score $ms(j)$ is increased by $cw(i, j)$, the term weight for $t(i)$ in $d(j)$. Thus working sequentially through $t(1)$, $t(2)$, $t(3)$, the scores for this example are incrementally calculated as follows.

	init.	$t(1)$	$t(2)$	$t(3)$
$ms(1)$	0.0		$+cw(2, 1)$	$+cw(3, 1)$
$ms(2)$	0.0	$+cw(1, 2)$		$+cw(3, 2)$
$ms(3)$	0.0			
$ms(4)$	0.0	$+cw(1, 4)$	$+cw(2, 4)$	
$ms(5)$	0.0			$+cw(3, 5)$

Example

The final values of $ms(j)$ will obviously depend on values of $cw(i, j)$ for this document set. But let us suppose that ranked in order of decreasing matching score the system output was as follows:

$$\begin{array}{lll} 1 & d(4) & \rightarrow \\ 2 & d(2) & \rightarrow \\ 3 & d(5) & \rightarrow \\ 4 & d(1) & \rightarrow \end{array} \left. \vphantom{\begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array}} \right\} \text{links to documents}$$

The user is now free to select any of the retrieved documents. A link can access the full original contents of the document.

Example

Note 1: document $d(4)$ does not appear in the list since no terms matched.

Note 2: score $ms(5) > ms(1)$ despite having fewer matching terms, rank order is determined by the term weights $cw(i, j)$.

Importantly, it does not matter here that all terms in the query do not match since the output matching score is merely the sum of the weights which do match.

Example

The likely effect of adding further terms to the query is either: to refine the ordering, add some additional documents in the lower part of the ranked list, or to have no effect if the term is not present in the documents.

This example illustrates retrieval in a simple static document set. In practice this is rarely the case and individual documents may be added or removed at any time.

This means that the inverted file structure must be regularly updated and term weights similarly updated. This process needs to be planned very carefully if management of the inverted file is to be efficient.

Example

The example also assumes a one-to-one mapping between $t(i)$ and $x(i)$.

This is rarely (if ever) the case, usually it is many-to-one.

Thus once $x(i)$ has been located, the system must ensure that it matches only with the correct $t(i)$ entries.

This is most easily accomplished by including the string $t(i)$ in the linked list entry and comparing $t(i)$ with the term string in each entry.

Clearly, this is inefficient and more complex strategies, involving using additional alternative hashing functions, are often used in practice.

Techniques for Improving Information Retrieval

In IR there is often mismatch between the terms that appear in a document and in a query. Essentially the ASK problem from earlier.

For example, the query

“reports on new Smartphones”

will fail to find a document containing only the following relevant fragment

“press releases for the latest mobile computing devices”

In these cases a query will either fail to retrieve all the relevant items, or they will be retrieved at low rank if not all the terms in the query match the relevant documents (probably often too low for the user to spend time looking for them!).

Techniques for Improving Information Retrieval

IR researchers have explored a number of methods to modify or reformulate the search query to attempt to overcome this problem.

These can broadly be categorised as *global* and *local* methods.

Global Query Reformulation Methods

Vocabulary Tools for Query Reformulation

The ASK problem means that users are often not able to describe their information need effectively.

After an initial attempt to address the information need, the IR system can provide the user with information which may help them form a more effective reformulated query. The user might be shown:

- Words from their query treated as stop words.
- Stems of the words as used by the IR system.
- No of hits of each word. (The “postings” information.)
- Details of the use of words in a phrase.

Global Query Reformulation Methods

Query Expansion and Thesauri

A possible solution to the matching problem is to try to expand the query by adding terms that are related to the original query terms.

A thesaurus describes relationships between words and phrases.

A simple use of a thesaurus method is to add (or suggest synonyms) of the original query word, e.g. “internet” → “WWW” “world wide web” “net” “web”

Thesauri may add unhelpful terms, e.g. the inclusion of “web” and “net” may retrieve documents on “spiders” and “fishing”, or may fail to include useful terms, e.g. if we are generally interested in computing networking “WAN” or “LAN” might be useful or perhaps protocols such as “TCP/IP”

Using a thesaurus for query expansion means that no user input is required.

Global Query Reformulation Methods

It is this failure to take into account the context of the term, i.e the way in which it is used in the query, that leads to this being referred to as a *global* reformulation method.

Adding additional words in this way generally leads to improved recall, i.e. more of the relevant documents in the collection may be returned.

However, the failure to take context into account can also lead to the return of many additional non-relevant documents.

The unreliable nature of the expansion terms means that they can be downweighted relative to the original query terms.

Generally thesaurus based query expansion has not been found to be a reliable method for improving precision in IR.

Global Query Reformulation Methods

Query Expansion via Log Mining

The current query can be compared to archives of logged queries from other users.

The IR system may suggest similar queries taken from the log archives to the user. The user may recognise one or more of these as better descriptions of their information need.

Retrieval results may favour documents in which searchers entering these earlier queries have shown interested by clicking on them.

The user can accept one of the previous queries in its entirety to replace their query, or they may revise their own query by adding or removing words based on the suggestions from the log.

Local Query Reformulation Methods

Local query reformulation methods adjust the query based on the documents retrieved in the current or previous runs for this query.

These are *local* in the sense that the process makes use of terms which have been brought together to form a local grouping based on their similarity to the current query.

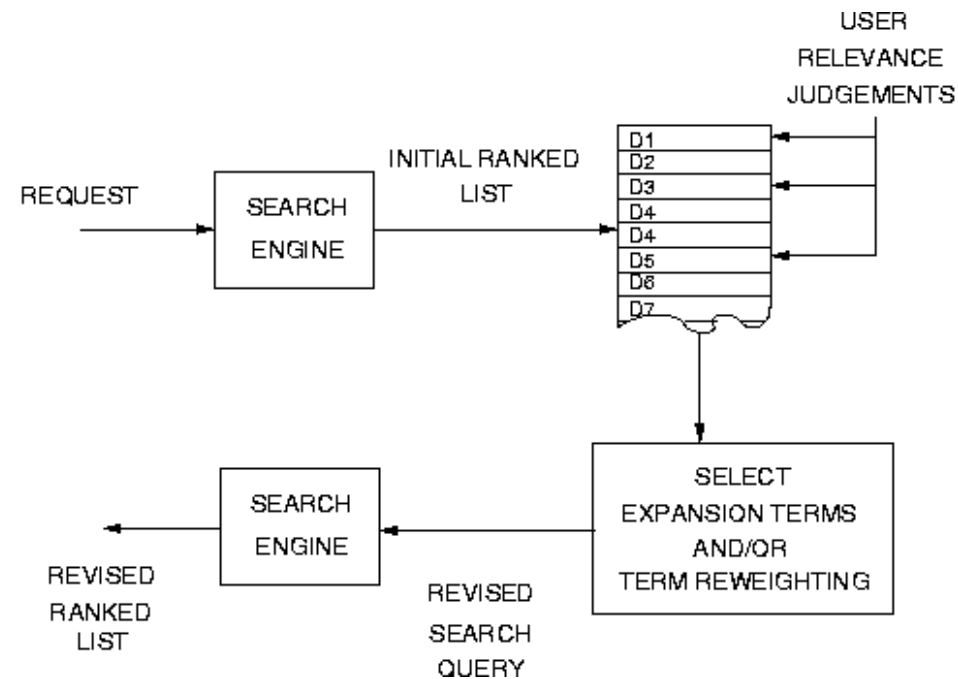
The most popular form of local query reformulation is *relevance feedback (RF)*.

Relevance Feedback

Relevance feedback uses relevance information from an initial retrieval run for the current query to perform one or both of two processes:

- add significant terms from relevant documents to the current query.
- modify the weights of terms to improve the ranks of documents which are likely to be relevant.

Relevance Feedback



Relevance feedback can be applied in the case of both the vector-space model and the probabilistic model.

Relevance Feedback

Relevance information can come from three possible sources:

- Explicit feedback: The user can mark documents from the current run as relevant (or non-relevant).
- Implicit feedback: The system can assume that returned documents clicked on by the user are relevant.

This assumes that the user clicks on documents because they are trying to locate relevant items, and that the snippet summaries (or other item surrogates) are reasonable indicators of relevance to the query.

- Blind or pseudo feedback: The system can simply assume the top ranked documents are all relevant, e.g. the top 5 or 10 returned documents.

Relevance Feedback

Relevance feedback makes several important assumptions:

- The user is assumed to know enough about the topic to make a reasonable initial attempt at a query to retrieve relevant items.
- Relevant documents are assumed to be similar to each other, and that additional unseen relevant documents will be similar to those that are already known.

Relevance feedback methods assume that this similarity will reveal itself in the form of similar distributions of terms in all relevant documents.

By implication this means that relevant documents will be dissimilar to non-relevant ones.

Sufficient documents must be judged, otherwise RF can be unstable.

Relevance Feedback in the Vector-Space Model

Query expansion can be seen as adjusting term weights in the query vector.

Adding a new term to the query corresponds to giving this term a non-zero weight in the vector.

Emphasizing or reducing the importance of a query term corresponds to increasing or decreasing its weight.

Given an initial query vector represented by non-zero weighted terms,

$$Q = (w_1, w_2, \dots, w_t)$$

the relevance feedback process produces the new vector,

$$Q' = (w'_1, w'_2, \dots, w'_t, w'_{t+1}, \dots, w'_{t+k})$$

the weights have been updated and k new terms added.

Relevance Feedback in the Vector-Space Model

The standard relevance feedback for Vector-Space IR was developed by Rocchio. The standard Rocchio query modification is:

$$Q' = \alpha Q + \beta \frac{1}{R} \sum_{x=0}^{R-1} RD(x) - \gamma \frac{1}{NonR} \sum_{x=0}^{NonR-1} NonRD(x)$$

Q' = modified query vector

Q = original query vector

R = no of **known** relevant documents

$NonR$ = no of **known** non-relevant documents

$RD(x)$ = relevant document vector x

$NonRD(x)$ = non-relevant document vector x

α, β, γ = are empirically determined constants

Relevance Feedback in the Vector-Space Model

Q' is the vector sum of the original query plus vectors of the relevant and non-relevant documents.

Many variants on Rocchio have been developed, e.g., investigations have explored:

- reweighting the original terms and not adding new terms.
- only adding the highest weighted new terms (but how many should be added?)
- only using relevant documents to complete Q' , i.e. set $\gamma = 0$

What would be the effect of each of these?

Like the vector-space model itself, Rocchio is an empirical method without an underlying formal theory.

Relevance Feedback in the Probabilistic Model

The relevance feedback methods for the probabilistic model are again theoretically motivated.

Term reweighting and query expansion are treated separately.

Computing relevance weights seeks to answer the question:

“How much evidence does the presence of this term provide for the relevance of this document?”

Selecting new terms to add to a query should answer a different question:

“How much will adding this term to the request benefit the overall performance of the search formulation?”

Relevance Feedback in the Probabilistic Model

Terms are reweighted by replacing $cfw(i)$ by $rw(i)$ (known as the Robertson/Sparck Jones weight).

$$rw(i) = \log \frac{(r(i) + 0.5)(N - n(i) - R + r(i) + 0.5)}{(n(i) - r(i) + 0.5)(R - r(i) + 0.5)}$$

$n(i)$ and N are defined as previously, R and $r(i)$ are new variables.

$n(i)$ = the number of documents term $t(i)$ occurs in,

N = the total number of documents in the collection archive.

$r(i)$ = the number of **known relevant** documents term $t(i)$ occurs in,

R = the total number of **known relevant** documents in the collection archive.

Relevance Feedback in the Probabilistic Model

$rw(i)$ reduces to something similar to $cfw(i)$ if $r(i) = 0$ and $R = 0$, i.e. $cfw(i)$ is essentially $rw(i)$ where is no relevance information available.

The 0.5 constants are added for smoothing, otherwise $rw(i)$ would be set to 0 for terms in the query that have not been observed in any relevant documents so far.

All terms appearing in relevant documents are potential query expansion terms.

These terms are first ranked using the Robertson offer weight $ow(i)$.

$$ow(i) = r(i) \times rw(i)$$

Relevance Feedback in the Probabilistic Model

A number of top ranked terms are then added to the original query.

How many should we add?

Note: Terms with high $ow(i)$ values, and which are thus potentially good expansion terms, generally need both high $r(i)$ and high $rw(i)$.

What does this tell us in about terms which are typically added to a query?

Relevance Feedback in the Probabilistic Model

Query expansion for the probabilistic model also often incorporates other empirical elements.

Original query terms are often upweighted using a scalar constant.

This is done since the original query terms were entered by the user, they can be regarded as more reliable than automatically selected expansion terms.

This is reflected in giving them more significance in computing the query-document matching score, i.e. the relevance feedback $cw(i, j)$ of the original query terms is calculated as $\alpha \times cw(i, j)$. This gives these terms a greater contribution to the overall matching score.

α is typically set between 1.5 and 3.5.

Data Fusion

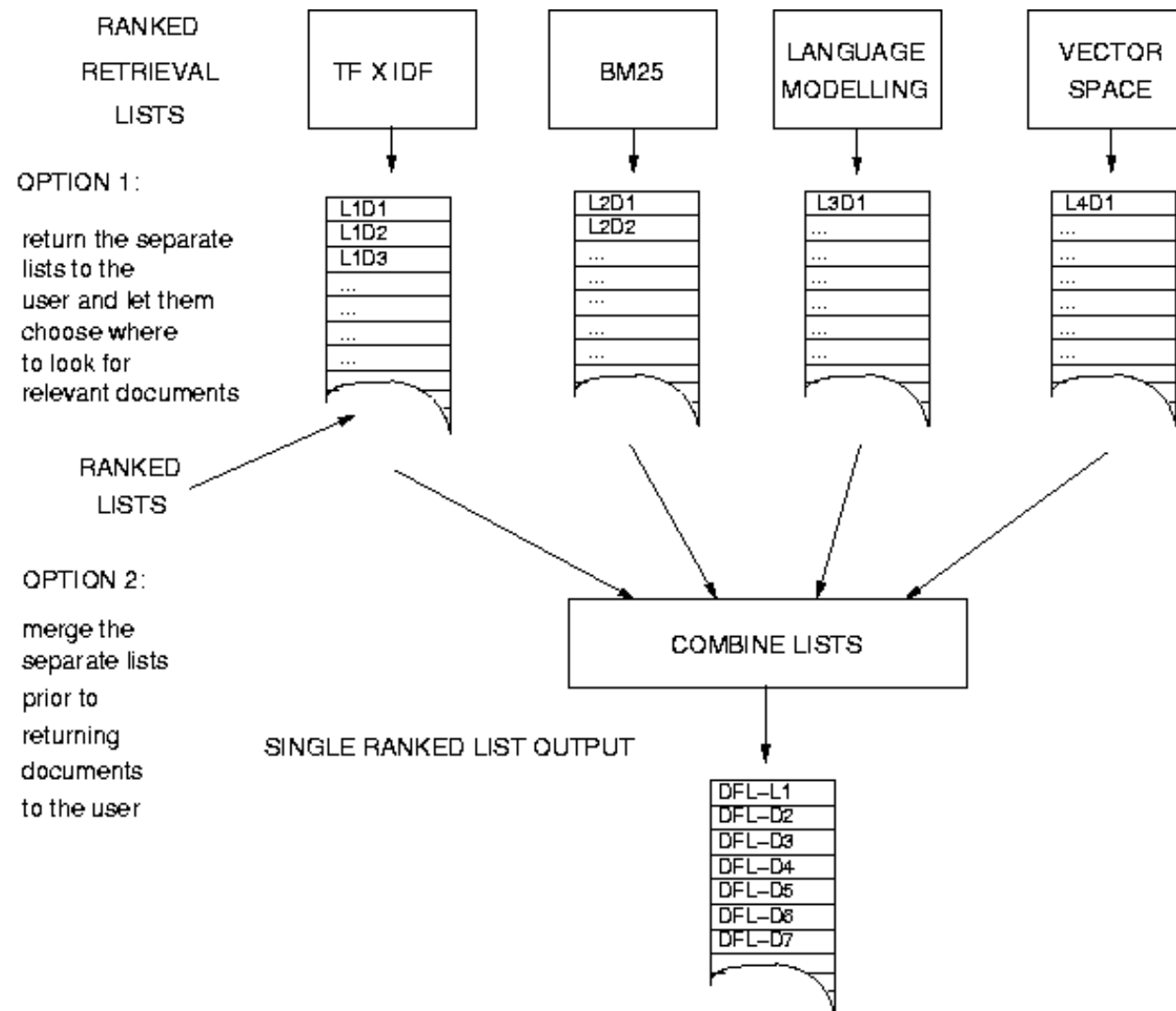
Different retrieval methods and/or term weighting schemes usually produce different rankings of documents.

An improved overall retrieval list can often be achieved by merging ranked lists generated independently by different IR systems

This merging process is referred to as *data fusion* involves adding the query-document matching scores of the different lists.

For example, when using data fusion, a document ranked 3rd by all the IR systems used in the data fusion may end up ranked 2nd in the merged list, if the alternative IR systems each rank a different document at rank 2.

Data Fusion



Data Fusion

The different weighting schemes are likely to produce matching scores which are incompatible in absolute value or possibly in range over the ranked list.

Merging methods:

Raw Score: ignore score incompatibility issues and merge documents based on query-document matching scores.

Normalised Score: for each list divide the query-document matching score of each document by the maximum score for the list (maximum score for each list is then 1.0). Then merge as for Raw Score method.

Many other more complex methods have been explored.

Evaluation of IR Systems

In order to know the how effective an IR system is, i.e. how good it is at retrieving relevant documents and not retrieving non-relevant ones, how much users like it, and how much it enables them to address their information needs, an IR system must be evaluated.

Evaluation is a key component in the development of IR systems and research into new IR methods and tools.

This requires some form of assessment of the effectiveness of the IR system in addressing the searcher's information need.

Evaluation of IR Systems

An important consideration here is the measurement of the effectiveness of one IR system relative to another.

- Which system is better at achieving its objective?
- But better at what exactly? This depends on the task for which the IR system is being developed.

Determining how to measure this and to implement the evaluation can pose a significant challenge.

There is no simple way to determine whether the output of one IR system is better than another one for a particular search application.

The only way to determine which system is most suitable in a particular situation is to adopt a formal process of *evaluation*.

Evaluation of IR Systems

An important question about any software system: Does it work correctly as designed?

- Determined by program testing - really a software engineering issue.

Checking this may not straightforward for this type of system.

If it doesn't crash, and it produces sensible numbers based on the calculations being carried out within the system, how can we be sure that they are correct? Might there be an error somewhere in the implementation of the algorithms?

We won't look at this topic further in this module.

Evaluation of IR Systems

So, let's assume that the system has been coded correctly and is working as intended.

Some important questions in evaluation of an IR system:

1. Is it useful? Is it “good enough” when applied to the task for which it has been designed?
2. Is it better than other systems which attempt to do the same thing?

Evaluation must adopt a planned strategy which enables these factors to be measured in a reliable way.

It cannot rely on anecdotal evidence from individual users with individual requests, i.e. you can't try a few queries on a system and base assessment of its effectiveness on personal gut reaction of how good you think it is.

Evaluation of IR Systems

Key Assumption: if an IR system performs well in a (valid) experimental setting, then it will perform well in an operational setting.

The system should be evaluated using a group of users completing search tasks for a carefully chosen set of search requests.

Evaluation of IR Systems

There two categories of evaluation:

- Qualitative evaluation: Users provide feedback on their views of the use of a particular IR system while using it to complete a task or after completing this task.

Neither is ideal - feedback during the task disrupts the user's engagement with the task, afterwards means that they might forget to include an important detail of their search experience.

Qualitative evaluation: users' impressions, e.g. using interviews, questionnaires, observing use of the system, ...

– rather like evaluation of a user interface in any interactive software system

– often referred to as *user-centered* evaluation.

Evaluation of IR Systems

- Quantitative evaluation: Numerical evaluation of the effectiveness of an IR system for completing a search task.

This may involve interactive tests with a user or use offline laboratory-based experiments:

- Interactive evaluation of live system: e.g. user time to complete a search task, number of relevant or useful documents that the user is able to retrieve using the system.

In general use evaluation methods from standard user testing of interactive systems.

- Repeatable laboratory-based experiments enabling comparison of search methods based on IR *test collections*. (Our focus here)

Document Relevance

A document that satisfies a user's information need is said to be *relevant* to the information need.

The notion of relevance should not be confused with documents scored highly in the output of an IR system.

High scoring documents match the query well (compared to other documents), but this does not mean that they are relevant or in any way useful!

Relevance of documents with respect to an information need can *only* be judged by humans.

Human relevance judgements form the basis of quantitative IR evaluation.

Document Relevance

What does it mean to say that a document is relevant?

1. does the document completely satisfy the initial information need?
2. does the document answer part of the information need?
3. does the document merely help the user to develop their understanding of the topic of their information need without really answering the problem, in which case it might be argued to be at least partially relevant to the information need?
4. is the document completely non-relevant to the information need?

Document Relevance

Look back to the ASK model of IR as a cognitive process in *Section 1: Introduction* of this module.

If the searcher cannot describe their information need, perhaps they are only able to create search queries which retrieve documents of relevance class 3.

However, what the user learns about the topic from documents of type 3. may help them to revise their request for a subsequent search by informing them about the subject.

This revised request might then enable documents of types 1. and 2. to be retrieved in the subsequent retrieval operation.

Document Relevance

Relevance is often a complex concept, and deciding the relevance of a document to a request not straightforward.

Questions about relevance and assessment of relevance include:

- What *exactly* is it relevant to, the underlying information need or the request,
 - only if the relevance is judged by the user with the information need, can the judgment truly be with regard to the information need, rather than to a statement of the information need.
- Who should judge make the relevance of a document?
 - often not realistic to have the person with the information need make the assessment - see later.

Document Relevance

- When, how, and on what basis of what data should the relevance judgment be made?
- Subjectively/objectively of relevance judgements?
- What degrees of relevance should be available in the assessment?
- Individual items/sets of items/sequencing of items?
 - Sequencing effect can introduce bias in the assessments, if the assessor has seen the same information in a previous document, they may be less likely to mark the current one as relevant.

Document Relevance

In IR evaluation relevance is most easily treated as a binary decision:

- a document is deemed either relevant or non relevant to the information need or request.
- but, human relevance assessors actually generally prefer to give a graded judgment.
 - in graded judgments assessors consider which of a number of categories each document should be assigned to, e.g. the set of four relevance classes shown earlier.
- a set of graded relevance judgments can be reduced to a binary one by making a cutoff, e.g. only documents of class 1 might be labelled relevant, and documents of class 2, 3 and 4 are labelled non-relevant.

Document Relevance

- In terms of assessing the relative effectiveness of the IR system, it generally doesn't matter where the cutoff is taken, i.e. a good system is still a good system irrespective of the cut off position.

Since the relevance of a document is decided by a human, it is to some extent often subjective.

Different users may make different judgments about the relevance or non-relevance of a document to a given search request.

Studies of Relevance

Instructions on how to make relevance judgments which seek to address questions of what defines relevance for a relevance assessment exercise are typically given at the start of an assessment session.

However, even when this is done, studies of relevance have shown that:

- Even with very carefully defined statements of the information need expressed in a user request, different judges may disagree on the relevance of individual documents.
- On the whole, these disagreements are on the edges, i.e. documents for which it is difficult to decide relevance.

Studies of Relevance

- On average, IR systems show the same relative effectiveness with the slightly different sets of relevance judgements created by alternative assessors,

i.e., if a system is more effective based on the judgments of one assessor, it will still be more effective based on the judgments of another assessor
- Different assessment decisions will change the *absolute* values of the evaluation metrics,

but we are generally more interested in *relative* effectiveness of different systems or methods.

Evaluation of IR Systems

Summarising the analysis of laboratory-based evaluation to this point.

Evaluation of the effectiveness of an IR system for a search task requires:

- a *test collection* consisting of:
 - a set of documents from which items are to be retrieved.
 - a set of requests expressing user information needs.
 - relevance data telling us which documents are relevant to each request.
- A test collection can be used to evaluate the search effectiveness of alternative IR systems – these might be completely different or closely related since *any* change in an IR system (e.g. changing a conflation algorithm or stop word list) gives us a different system.

Evaluation of IR Systems

To run a laboratory-based IR experiment:

- enter the documents into each IR system(s) to be evaluated via indexing;
- put each request to each system;
- calculate the matching score for each document for each request in each system;
- collect the output.

Then,

- evaluate the output for each request, document by document using the relevance data for each request;
- analyse the results.

Measurement of IR Performance

In order to evaluate an IR system using a test collection, we need to define evaluation metrics or measures.

The metrics which follow assume simple binary relevance categorisation, i.e. in the case of a set of graded relevance judgments are reduced to binary ones using a cutoff as described earlier.

In essence, the goal of an IR system is:

1. to retrieve relevant documents;
2. not to retrieve non-relevant ones.

For any request a number of relevant documents may be present.

We need evaluation metrics which measure the effectiveness with which an IR system can retrieve them.

Measurement of IR Performance

A measure for 1. the ability of an IR system to retrieve relevant documents is:

$$\text{Recall} = \frac{\text{No of relevant docs retrieved}}{\text{Total relevant docs in collection}}$$

Within the retrieved documents, what proportion of the *known* relevant documents for this request have been retrieved?

For a Boolean retrieval system only a single Recall value is computed.

For a Best-Match retrieval system Recall is usually calculated at specific cut off points in a ranked retrieval list, e.g. what is the Recall after the top 5 ranked documents have been retrieved?

Measurement of IR Performance

A measure for 2. the ability of an IR system not to retrieve non-relevant documents is:

$$\text{Precision} = \frac{\text{No of Relevant docs retrieved}}{\text{Total docs retrieved}}$$

What proportion of the retrieved documents are relevant?

For Boolean retrieval systems a single precision value is computed.

For Best-Match retrieval precision is calculated at selected cut off points in the ranked list. For example, at a cut off after 5 documents, how many of the retrieved documents are relevant?

Recall and Precision are the two traditional measurements of IR system performance.

Measurement of IR Performance

A single measure of search effectiveness for best-match IR is the *Mean Average Precision (MAP)*, calculated as follows.

To compute MAP for each request first calculate the single figure for the *Average Precision (AP)* for this request,

$$\text{Average Precision for Request} = \frac{\sum_{\text{all rel. docs}} \text{Precision at rel doc cut off}}{\text{Total no of rel. docs}}$$

Then, calculate the Mean Average Precision over all the test requests.

Measurement of IR Performance

$$\begin{array}{l} \text{Mean} \\ \text{Average} \\ \text{Precision} \\ \text{(MAP)} \end{array} = \frac{\sum_{\text{request set}} \text{Average Precision for Request}}{\text{No of Requests}}$$

- In general, higher Recall and Precision indicate a better IR system.
- It is often found that adjusting an IR system to improve Recall will impair Precision and vice versa.
- Operational systems are usually developed to produce a task specific compromise between Recall and Precision.

For Boolean systems, a Mean Precision can be calculated by taking the average of the precision values over all the requests.

Example

Suppose that there is a very small collection of 1000 documents.

For a particular request a trial Boolean system retrieves 16 documents of which 8 are relevant.

Suppose also that there are a further 12 documents in the document collection which are relevant, but NOT retrieved. For this example,

Suppose that there are 4 further requests for which the precisions are respectively: 0.57, 0.41, 0.73, 0.48. the Average Precision is:

Example

For the same document collection with a best-match IR system, suppose the following retrieval behaviour is observed.

Rank	Rel of Doc.	Precision
1	Rel	—
2	Rel	
3	Not Rel	
4	Rel	—
5	Rel	
6	Not Rel	
7	Rel	—
8	Rel	
9	Not Rel	
10	Rel	

Example

Calculate the Recall and Precision at cut offs of 5 and 10 documents.

At cut off 5 documents in the ranked list.

At cut off 10 documents in the ranked list.

We could equally well calculate the precision any other cut off value.

Comparing retrieval performance at defined cut off levels for different systems gives a means of examining variations in behaviour between them , as well as the performance of individual systems.

Example

Mean Average precision for this request would be calculated as follows:

- Work down the ranked list adding the precision value each time a relevant document is found.
- After all retrieved documents have been examined divide the total by the number of relevant documents.

The overall Mean Average Precision for the 5 test requests considered earlier would then be calculated by summing the AP for each, and dividing by the total number of requests.

Designing and Building Test Collections

IR test collections have to be designed carefully with consideration of the search task which they represent:

- Scale – number of documents in the collection and the size of the request set.
- Documents - are they from just one source or a variety; would the contents be expected to change over time e.g. news material; is the collection typical of the operational environment of the system?
- Requests - how have they been collected? how well do they represent expected operational search requests for this task?

Designing and Building Test Collections

- Variety - how the documents and requests might be represented in the IR system, how are they used within the IR system?
e.g Boolean vs Best-Match.
- Relevance judgements
 - how have they been gathered, and to what extent can they be trusted?
who made the judgments, did they take the assessment task seriously? (as we will discuss shortly, it is often not practical to ask the person who created the requests to make all the relevance judgements)

Designing and Building Test Collections

A key question for the development of a test collection is :

Where do the relevance judgements come from?

As noted earlier, they must be gathered manually!

Considerations:

- – Ideally the relevance of each document should be judged for each request.
- Ideally the relevance of each document should be judged by the user with the information need who created the request. Why?

Designing and Building Test Collections

- This relevance judgement should be independent, that is the document should be judged either relevant or not relevant independent of the relevance of other documents.
 - Each document read will change the assessor's knowledge.
 - This may change their relevance judgments of later documents, particularly if they repeat information from earlier one, i.e. if takes the searcher no further towards addressing their information need.
- Documents are usually presented to the assessor in a randomised sequence.
 - Based on their experience with operational search engines, people generally expect documents to be less likely to be relevant as they work down a ranked retrieval list.

Designing and Building Test Collections

- – In a large collection (or even a modest one) there are too many documents for an individual assessor to judge the relevance of each document to each request.
- Since the relevance of all documents cannot be judged for each request, some approximation to the relevance set must be created. This is most commonly done using a method referred to as *pooling*. This is done by combining and assessing the output of a number of search engines.

To use the pooling method for relevance set creation, the retrieval output must be obtained first, and the relevance assessment carried out afterwards.

The retrieved documents down to an agreed rank, e.g. 100, are then formed into a union pooled set of documents.

Pooling

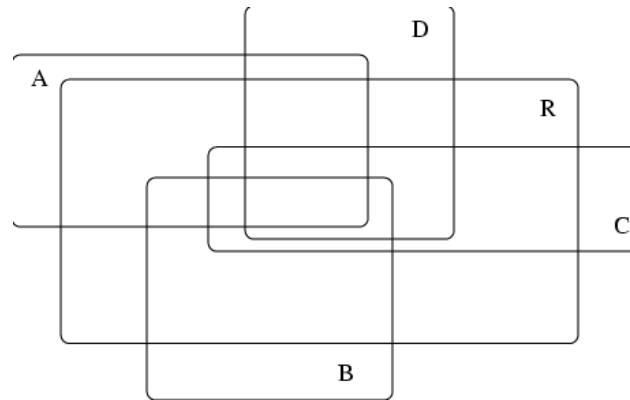
The contents of the document pool are sorted into random order to prevent sequence biasing effects discussed above.

The relevance of each document in the pool is then judged manually.

Since pooling only involves the assessment of the relevance of a subset of the document set, the relevance set developed using pooling is *only an approximation* of the *true complete relevance set*,

which will be a *subset* of the relevant documents in the collection, which *will remain unknown*.

Pooling



Let R represent all the relevant documents for a particular request r_x contained within this document set.

Let A, B, C and D represent documents retrieved by each of four different IR systems for request r_x . For each IR system this may include all documents retrieved or only those above a selected cutoff point in a ranked output list.

Thus different IR systems generally contribute different retrieved documents to the pool, which may or may not be relevant documents.

Pooling

For pooling to be effective, the IR systems used to create the pool should be as diverse as possible. This increases the potential that as many relevant documents as possible will be retrieved by at least one of them.

Of course, without manually judging the relevance of every document in the collection, **we do not know the true composition of R** , and we know that it is not feasible to make this complete assessment of the documents.

The approximate computation of R is carried out by **manually** judging the relevance of the pool of documents retrieved by A, B, C and D.

As shown, there may be some relevant documents not in the pooled relevance set because they have not been retrieved by any of the systems.

Pooling

Thus, using this method **we can never be sure that all relevant documents have been found** and the documents listed in our set R **will always only be an approximation** of the true membership of R , although hopefully a good one!

Our approximation of R is used to evaluate the Precision and Recall metrics for each request as described earlier.

Interactive Evaluation of IR Systems

Can the methods of IR system evaluation that we have studied be adapted to a more user-inclusive perspective to evaluate operational interactive IR systems?

Major problem: If we are dealing with a user's ASK rather than a request:

- An ASK is not directly visible to us ... and we cannot capture it.
- The ASK exists only for a short space of time, and it is likely to be changed by interacting with the system and by examining documents.
- Hence, we cannot hope to put the original ASK to more than one system in order to compare different systems ... or to get extensive relevance judgements for documents which may resolve the ASK.

Interactive evaluation is hard! - and is not explored further in this module.

IR Evaluation Workshops

This evaluation methodology of test collection development using pooling, and precision and recall based evaluation metrics, is actively used within a number of international IR evaluation workshops.

The best known and longest running is the annual Text REtrieval Conference (TREC) workshop details at `trec.nist.gov`.

Registered participants are given an agreed set of documents and retrieval requests, they submit their retrieval output for this data collection by a strict deadline, and performance is then computed for agreed evaluation metrics after pooling the results from the participants.

TREC is ongoing with new tasks being developed each year, e.g. web retrieval, question-answering, video retrieval.

TREC

The aims of TREC:

- To encourage IR research based on standard large-scale collections.
- To increase communication between industry, academia and government creates an open forum for exchange of research ideas.
- To speed the transfer of knowledge from research laboratories into commercial products.
- To develop and make available new evaluation techniques for use in academia and industry to encourage the development of new and more effective IR technologies.

TREC

Other evaluation benchmarks include CLEF, NTCIR, FIRE, MediaEval.

Most of leading international IR researchers are active participants in TREC or one or more of the other evaluation workshops.