Information Retrieval

Definitions

Information Retrieval (IR). The process of finding *documents* from a given *document collection* that are *relevant* to the *user's query.*

Document. IR works with **document collections.** Each **document**, typically, is a **text** or can be viewed as one.

Document collections. The key assumption of IR is that **document collections** are large.

Note: This is not always the case. There are some specialized uses of IR techniques, where the document collections are on the order of tens or hunderds of documents, not hundreds of thousands/millions as is usual for traditional IR settings.

Queries. A **user query** is a formal representation of an **information need**. The two are not the same.

An **information need** is an articulated desire to obtain certain information (or to find documents that contain certain information). A **query** is a representation of an information need that can be processed by the specific Information Retrieval system.

Example. The user information need is to find all information about the University of Alabama's 1992 football season, in particular, to find the list of all games.

The user query to a search engine (google) can be: `"University of Alabama Crimson Tide football 1992 season games schedule"`.

(Given this query, Google returns the wikipedia page on the 1992 Alabama football season, with the list of games as the top page).

Query types. Different IR systems use different types of **queries**.

keyword queries: user expresses information need as one or more **keywords** (**terms**). The IR system searches for documents containing one or more of the terms specified by the user.

Boolean queries: user expresses information need as a Boolean expression connecting one or more keywords. The IR system searches for documents that match the boolean query.

phrase queries: user expresses information need in a form of a sequence of words/terms constituting a phrase. The IR system searches for documents that contain at least one instance of the phrase.

proximity queries: user expresses information need in a form of a sequence of words/terms constituting a phrase. The IR system searches for documents that contain all terms in the query close together in the text.

full document queries: user expresses information need in a form of an existing document from the document collection. The IR system returns the list of documents that are most similar to the query document.

natural language questions/queries: user expresses information need in a form of a question (text) in natural language. The IR system finds the documents in the collection that best answer the question (and occasionally, integrate the information obtained to provide a single cohesive answer to the question).

Ranking. IR systems typically **rank** answers to the user queries. The ranking is done according to the **computed relevance of each document to the query** as perceived by the IR system.

Relevance. A numeric measure determining whether a document should be returned as the answer to a user query. Typically, **relevance** ranges from 0 (completely irrelevant) to 1 (extremely relevant). Different IR models interpret **relevance** in different terms: probability that a user will find the document relevant; degree of similarity between document and query (or their representations), etc.

Architecture of an IR system

Figure 1 shows an architecture of a typical IR system.

A typical IR system consists of two major components:

Indexing system: the **off-line component** of an IR system. This component is responsible for processing the document collection and creating an **internal representation** of the documents that can be used by the **retrieval engine** to efficiently retrieve documents given queries.

Retrieval Engine: the **on-line component** of an IR system. The retrieval engine accepts queries from users, processes them using the internal representation

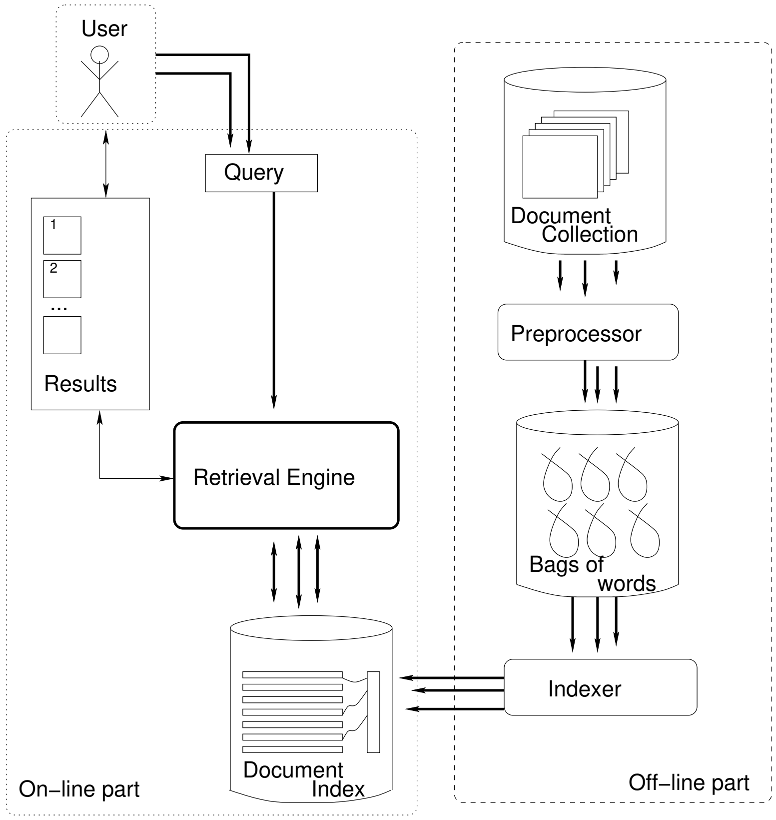

Figure 1: Architecture of an Information Retrieval system.

of the document collection consturcted by the **indexing system**, and delivers results back to the user. Retrieval engine can contain a **user feedback mechanism** for refining queries and returning better results.

All in all, from document collection and to the query answer, there are three main stages:

1. **Preprocessing.** Preprocessing involves a number of, typically, model-independent procedures that turn input documents from text into representations that **model-based indexer** can use to construct the **document collection index**. Typical preprocessing steps include:

   (a) **Parsing and tokenizing** input. On this stage individual keywords/terms, sentences, phrases and other constructs are identified. Also, parts of the input, not needed in the indexing process (e.g., HTML tags) are filtered out.

   (b) **Stopword removal.** Some words are ubiquitous, and their value as keywords is negligible. These words are identified and removed from the document representations.

   (c) **Stemming.** Each keyword is replaced with its **stem** - a substring that represents the *unchangable portion of the keyword*. This allows the IR system to treat words like ”computer”, ”computers”, ”computing”, ”computation” as the same keyword. [1]

[1] This is not always desirable. Google, for example does not stem keywords. However, in smaller document collections stemming can significantly improve the quality of retrieval.

2. **Indexing.** Indexing produces the **document collection index**: a collection of data structures representing the document collection in a way that (a) reflects the nature of the collection and (b) allows for efficient retrieval query processing.

   Indexing is **model-dependent**: different IR methods use different approaches to building the final document collection index.

3. **Retrieval engine.** The query engine has three roles (the third role is optional, but is commonly found):

   (a) **Management of user queries.** User queries are accepted, and, depending on their type, analyzed, parsed, and indexed.

   (b) **Query processing.** Parsed and indexed queries are compared using *retrieval algorithms*. Results are *retrieved*, *ranked* and provided to the user.

   (c) **Feedback processing.** (*optional*). User feedback regarding relevance of specific retrieved documents is accepted, analyzed, and used to refine the list of retrieved documents.

Information Retrieval Models

IR model is a way of representing documents and queries and computing the relevance between them.

Common Features

Most IR models share the following features.

Document collection. The input data for each IR model is a document collection $D = \{d_1, \ldots d_n\}$ of documents.

Vocabulary (corpus). The collection of **non-stop word** terms found in the documents from D is called the **vocabulary** or **corpus** of D. Given D, we denote the vocabulary of D as

$$V_D = \{t_1, \ldots t_M\}.$$

(where D is unique, we denote the vocabulary of D as simply V.) Each t_i is a **distinct term** (keyword) found in at least one document in D.

Bag of words representation. Each document $d_j \in D$ is represented as a **bag of words**, i.e., as an **unordered** collection of terms found in each document (**bag** means that the number of occurrences of each term may be taken into account).

The standard representation of **bag of words** is a **vector of keyword weights**: a vector which assigns each term $t_i \in V$ a **weight** based on its occurrence/non-occurrence in d_j.

As such, we view d_j as the vector

$$d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj}).$$

Here w_ij is the weight of term t_i in document d_j.

Different **IR models** represent **term weights** in different ways.

Boolean Model

Boolean Information Rertieval model is the simplest IR model. It has the following features:

Term weights. Given a document $d_j \in D$, it is represented as the vector of **binary keyword weights** $d_j = (w_{1j}, w_{2j}, w_{3j}, \ldots, w_{Mj})$, where $w_{ij} \in \{0, 1\}$ and

$$w_{ij} = \begin{cases} 1 & : & t_i \text{ appears in } d_j; \\ 0 & : & \text{otherwise}. \end{cases}$$

Queries. Boolean model accepts simple **keyword queries** and **Boolean queries**. A keyword query $q = \{t_{q1}, \ldots, t_{qs}\}$ is represented as a boolean expression $(t_{q1} \wedge t_{q2} \wedge \ldots \wedge q_{qs}$.

Query processing. A document vector d_j is relevant to query q (keyword or boolean) if $d_j \models q$, i.e., if q is **satisfied by** or is **true on** d_j.

This reduces **query processing** in boolean model to *satisfaction checking* or **boolean expression evaluation** on specific instances.

Document retrieval. Each document satisfying q is **retrieved**. Retrieved documents are **not ranked**: they are either retrieved (match the query) or not retrieved (do not match the query).

Pros. Simplicity. Efficiency. Familiarity.

Cons. No ranking. Require **exact match** (rarely useful on practice). Exact match of keywords (no synonyms, no related terms).

Vector Space Model

Overview. **Vector Space Model** represents keyword weights on the scale from 0 to 1 and represents queries in a way, similar to documents. It uses **cosine similarity** to compute the relevance between a document and a query and uses the relevance value to rank the results.

Term frequency. Given a document $d_j \in D$ and a term $t_i \in V$, the **term frequency (TF)** f_{ij} of t_i in d_j is the number of times t_i occurs in d_j. For a document d_j, we can construct its vector of term frequencies

$$f_{d_j} = (f_{1j}, f_{2j}, \ldots, f_{Mj}).$$

Normalized term frequency. Term frequencies are commonly manipulated to provide for a better representation of the document. Two manipulation techniques used are **thresholding** and **normalization**.

Given a **threshold value** α, we set **term frequency** f'_{ij} to be

$$f'_{ij} = \begin{cases} f_{ij} & : \ f_{ij} < \alpha; \\ \alpha & : \ f_{ij} \geq \alpha \end{cases}$$

(i.e., we discount any further occurrences of the terms in document beyond a certain **threshold** α number of occurrences).

Given a vector f_{d_j} of (possibly thresholded) term frequencies, we compute **normalized term frequencies** tf_{ij} as follows:

$$tf_{ij} = \frac{f_{ij}}{\max(f_{1j}, f_{2j}, \ldots, f_{Mj})}.$$

Document frequency (DF). Given a term $t_i \in V$, its **document frequency**, df_i is defined as the **number of documents in which** t_i **occurs**:

$$df_i = |\{d_j \in D | f_{ij} > 0\}|.$$

Inverse document frequency (IDF). Given a term $t_i \in V$, its **inverse document frequency (IDF)** is computed as

$$idf_i = \log \frac{n}{df_i}.$$

TF-IDF keyword weighting schema. Given a document d_j and a term t_i,

$$w_{ij} = tf_{ij} \cdot idf_i = \frac{f_{ij}}{\max(f_{1j}, \ldots, f_{Mj})} \cdot \log_2 \frac{n}{df_i}.$$

Intuition. The idea behind the **tf-idf weighting schema** is straightforward. The importance of a keyword to a document is measured using two rules:

1. The more often the keyword appears in a document, the more important it is.
2. The less frequently the keyword appears in the document collection, the more important its occurence is in each document.

Term frequency (or normalized term frequency) captures the first rule. **Inverse document frequency** captures the second rule.

Query weighting schemas. Given a query $q = (w_{1q}, \ldots, w_{Mq})$, the query term weights can be determined using the exact **TF-IDF weighting schema**:

$$w_{iq} = tf_{iq} \cdot idf_i.$$

However, if q is short (contains relatively few terms in comparison with documents from D), some slight adjustments can be adopted:

$$w_{iq} = (0.5 + 0.5 \cdot tf_{iq}) \cdot idf_i.$$

Other simple weighting schemas. Some other simple term weighting schemas:

Normalized Term Frequency (TF): $w_{ij} = tf_{ij}$.

Inverse Document Frequency (IDF): $w_{ij} = idf_i$.

(note, when d_j has no repeated keywords, TF-IDF weighting schema converges to simple IDF weighting. This is common in document collections that consist of small documents: e.g., lists of quotations.)

Relevance computation. Vector space retrieval traditionally uses the **cosine similarity** to compute relevance:

$$sim(d_j, q) = cos(d_j, q) = \frac{d_j \cdot q}{||d_j|| \cdot ||q||} = \frac{\sum_{i=1}^{M} w_{ij} \cdot w_{iq}}{\sqrt{\sum_{i=1}^{M} w_{ij}^2 \cdot \sum_{i=1}^{M} w_{iq}^2}}.$$

Modifications of Vector Space Retrieval

Various modifications have been proposed both the **TF-IDF weighting schema** and to the **relevance computation.**

Okapi. **Okapi** is a family of vector space retrieval methods that try to compensate for the disparity in the size of the query and the size of the document being queried.

These methods can be considered as either modifying the term weights or modifying the relevance computation function.

The following formula shows how the relevance compuation is modified:

$$okapi(d_j, q) = \sum_{t \in d_j, q} \ln \frac{n - df_i + 0.5}{df_i + 0.5} \times \frac{(k_1 + 1) \cdot f_{ij}}{k_1 \cdot (1 - b + b \cdot \frac{dl_j}{avdl}) + f_{ij}} \times \frac{(k_2 + 1) \cdot f_{iq}}{k_2 + f_{iq}}$$

Here,

dl_j:	length of document d_j (in bytes)	
$avdl$:	average length (in bytes) of a document in D	
k_1:	normalization parameter for d_j	$1.0 - 2.0$
b:	normalization parameter for document length	usually 0.75
k_2	normalization parameter for query q	$1 - 1000$

Pivoted normalization weighting (pnw).

$$pnw(d_j, q) = \sum_{t_i \in d_j, q} \frac{1 + \ln(1 + \ln(f_{ij}))}{(1 - s) + s \cdot \frac{dl_j}{avdl}} \times f_{iq} \times \ln \frac{n + 1}{df_i}.$$

Here, s is the document length normalization parameter, usually set to 0.2.

Evaluation of IR Systems

Notation. Let $D = \{d_1, \ldots, d_n\}$ be a document collection. Let q be a query, and let $S_q = \{d'_1, \ldots, d'_k\}$ be a **list of documents** retrieved by some IR system S. The documents are ranked in the order their of percieved relevance to the query:

$$sim(d'_1, q) \geq sim(d'_2, q) \geq \ldots \geq sim(d'_k, q).$$

We also assume that if for some $d_j \in D$, $sim(d_j, q) > 0$ then $d_j \in s(q)$, i.e., the list $s(q)$ contains all documents deemed relevant by the system.

Given a query q, let $D_q = \{d^*_1, \ldots, d^*_l\}$ be the **set of documents from** D **that are truly relevant to query** q. This set is often referred to as **"golden standard"**.

Precision. The **precision** of IR system S on query q is the percentage of **retrieved** documents that are **relevant**:

$$precision = \frac{|D_q \cap S_q|}{|S_q|}.$$

Precision answers the question *what percentage of retrieved documents is relevant?*

Recall. The **recall** of an IR system S on query q is the precentage of **relevant** documents that are **retrieved** by S:

$$recall = \frac{|D_q \cap S_q|}{|D_q|}.$$

Recall answers the question *what percentage of relevent documents is retrieved?*

F-measure. The **f-measure** is the harmonic mean of **precision** and **recall**.

$$f_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}.$$

More generally, for $\beta \neq 0$,

$$f_\beta = \frac{(1 + \beta^2) \cdot precision \cdot recall}{\beta^2 \cdot precision + recall}.$$

Recall and precision at rank i. In many retrieval systems, the entire set S_q is either unobservable (too many answers) or is never used.

(e.g., most people concentrate on the first 1–2 pages of links returned by Google.)

We can measure the accuracy of the retrieval using **recall** and **precision at rank** measures.

Notation. Let $S^i_q = \{d'_1, \ldots, d'_i\}$ denote the list of top i retrieved documents for query q.

Recall at rank i. Recall at rank i is the recall achieved by considering only the set S^i_q as the query answer:

$$recall(i) = \frac{|S^i_q \cap D_q|}{|D_q|}.$$

Precision at rank i. Precision at rank i is the precision achieved by considering only the set S_q^i as the query answer:

$$precision(i) = \frac{|S_q^i \cap D_q|}{|S_q^i|}.$$

Average precision. **Average precision** is the mean of the precision over all ranks i:

$$p_{avg} = \frac{1}{|S_q|} \cdot \sum_{i=1} |S_q| precision(i).$$

Precision-Recall Curve. The plot showing the values of **precision at rank** i and **recall at rank** i (plotted against each other).

Precision at recall levels. Sometimes, it makes sense to consider the following collection of precision metrics. We select values $0 < r_1 < r_2 < r_3 < \ldots r_s \leq 1$. For each value r_j from the list, we find the set $S_q^i \subseteq S_q$, such that $recall(i) \geq r_j$ while $recall(i-1) < r_j$ (i.e. the smallest list of ranked retrieved documents that achieves the recall of r_j).

The precision $precision(r_j)$ at recall level r_j is defined as $precision(i)$ for such S_q^i. Precision-recall curve can be used to plot precision at recall levels.

Multiquery measures. Average Expected Precision. IR system behavior on individual queries may vary. It is important to be able to summarize IR system accuracy in general, i.e., *over multiple queries*.

Let $Q = \{q_1, \ldots q_L\}$ be a list of queries and $S_{q_1}, \ldots, S_{q_L}$ be the ranked lists of retrieved documents. **Average precision over Q at recall level** r_j is defined as

$$\hat{precision}_Q(r_j) = \frac{1}{|Q|} \cdot \sum_{i=1}^{|Q|} precision_{q_i}(r_j).$$

Let $S_q(r_j)$ be the set $S_q^{s_q} = \{d_1^q, \ldots, d_{s_q}^q\} \subseteq S_q$ such that $recall(S_q^{s_q} \geq r_j$ but $recall(S_q^{s_q-1}) < r_j$.

Overall precision at recall level r_j is defined as

$$precision_Q(r_j) = \frac{\sum_{i=1}^{|Q|} |S_{q_i}(r_j) \cap D_{q_i}|}{\sum_{i=1}^{|Q|} |S_{q_i}(r_j)|}.$$

Preprocessing

Preprocessing involves a number of **model-independent** activities: i.e., tasks that need to be performed in order to represent documents from D as **bags of words**.

These tasks include **parsing**, **stopword removal** and **stemming**.

Parsing

Parsing is the only preprocessing activity that is dependent on the input format.

The **parsing** process reads one-by-one each document d_j from the document collection D. The following tasks are typically performed:

- The input document is **tokenized**. That is, the **parser** detects word boundaries, punctuation and other features of the document.
- The input document is **filtered** if necessary. Some documents are provided to the IR system in a form of `HTML` or `XML` files. Other documents may contain formatting instructions, tables, images, and other features that either require **deep parsing** or need to be/ can be ignored by the rest of the preprocessing and indexing components.

  (For example, if the documents are provided in HTML form, many HTML tags can be removed from the document file.)

Stopword Removal

Stopwords. A **stopword** is a word that is found in colloquial speech/literature/specific document collection *so often, that it does not carry any specific meaning, nor does it possess any discriminating ability.*

Various lists of stopwords exist. Typical stopwords are pronouns ("I", "you", "they", "them", "his"), common verbs ("do", "be", "am", "are", "have" "had"), propositions and connectives: ("in", "onto", "and", "or", "of", "from").

Some document collection acquire stopwords that are specific to them. E.g., a collection of Computer Science articles may need to declare "computer" to be a stopword.

Stopword removal. Stopwords cause *false positives*, so it ia important to get rid of them. The traditional mechanism is as follows:

- For each token of type word read from the document, check if the word belongs to the list of known stopwords.
- If it belongs to the list of stopwords, remove it from the stream, consider next word.
- If is **does not belong** to the list of stopwords, pass the token onto the next step of processing.

Stemming

Stemming is the process of replacing a word with its **stem** or **root**[2].

Stemming procedures are unique for each language. In English, **Porter Algorithm**[1] is considered to be the traditional **stemming technique**. The specifics of the algorithm can be found in [1], as well as at

[2] Technically, a stem of a word is more than just a root. Stemming algorithms keep prefixes intact.

http://tartarus.org/∼martin/PorterStemmer/

The algorithm itself is described at

http://tartarus.org/∼martin/PorterStemmer/def.txt

The algorithm is organized as a series of rule applications. Given a word, on each step of the series the algorithm checks whether a rule is applicable. A rule identifies the ending of the current word, and replaces it (occasionally, simply removes it) with a shorter ending.

References

[1] M.F. Porter, 1980, An algorithm for suffix stripping, Program, 14(3) pp 130–137.