Information Retrieval 24-25

Notes

University of Pisa

M.Sc. in Computer Science

Contents

Chapter 1

Introduction

Information retrieval is the process of finding relevant material of unstructured nature from large collections. The "material" is usually documents, web pages, or multimedia content. Originally, information retrieval was something only a few professionals interacted with. Nowadays, hundreds of millions of people engage with information retrieval systems when they, for example, use a web search engine or search through their email.

The two key aspects considered when evaluating the quality of an IR system are **effectiveness** and **efficiency**. The first refers to the capability of the system to produce a satisfactory result; the second refers to how quickly it does so. To guarantee a certain level of quality, some operations are done offline, such as document indexing, feature processing (e.g., term frequency, metadata), training of a learn-to-rank model to produce the order in which documents will be shown to the user, and so on. Still, many operations must be done on-line, such as query expansion and processing, index and feature lookup, and usage of the ranking funcion. If we consider the example of a search engine (SE), we are used to get back a response in a very short amount of time, despite the fact that in order to find the collection of documents presented to us, a lot of different operations must have been performed (i.e., the system must be efficient). Additionally, we also expect that those documents are the most relevant ones found in the collection, and that they are presented in the order of relevancy (the system must be effective). If those two ideas do not hold, we're unlikely to actually use the system for an extended amount of time.

The following chapters will go in detail about the different components of IR systems, and each will focus on how effectiveness and efficiency can be guaranteed. The key aspects that will be considered are:

- **Language properties:** how does language influence retrieval? What does it mean to retrieve a piece of text? How are documents scored and presented?

- **Auxiliary data structures:** e.g., inverted indexes;
- **Query processing:** how are queries expanded from the form provided by the user into one which can be "read" by the system?
- **Data storage and compression:** how can data be compressed efficiently?
- **Learning-to-rank models:** how is machine learning used in an IR system to produce a ranking (based on available ranked data)?
- **Neural IR:** how can Deep Learning and specifically Large Language Models be used in IR?

Chapter 2

Evaluation

To evaluate the quality of a IR system, say a SE, we may ask questions such as: how fast does it index a collection, how fast does it search (efficiency)? Or, does it recommend good related pages/products to buy to the user (effectiveness)? However, these questions alone do not provide any objective information about the intrinsic quality of the SE. We could say that a SE is "good" if it makes its users happy; but to measure this happiness, we can use different definitions: for example, how many times a search result is clicked, how long users stay on the same webpage, how often they return to use the SE. Since happiness by itself is impossible to measure, a commonly used proxy is **relevance** of search results.

2.1 Relevance

In order to measure relevance, three elements are needed:

- A benchmark document collection;
- A benchmark suite of queries;
- An assessment of either **relevant** or **non-relevant** for each query and document.

To construct the benchmark, we would have to analyze each possible pair of query and document and assign a relevance to it. Relevance assessment can be binary (relevant/not relevant), or multi-valued (0, 1, 2, 3 ...) for more nuance. Obviously, since assigning a relevance value to each query-document pair in a collection is way too expensive, a subset of the documents is used instead.

Assigning relevance must be done externally by humans. Some companies use crowdsourcing platforms (e.g., Amazon Mechanical Turk) to present pairs to low cost, not

highly qualified workers. This solution is cheap, but the outcome may not be as good as one produced by professionals.

But how are the queries defined? They must be relevant and suitable to the documents in the collection, and must be representative of user needs (i.e., they should resemble a normal query done by a person). One way to find them is to sample directly from existing query logs of the SE, if available. For classical, non-Web IR systems, these query logs may be nearly empty, as the query rate tends to be slow. In this case, experts may handcraft "user needs" and associated queries. Among popular public test collections is **TREC** (**Text REtrieval Conference**), where focus areas are called **tracks**; each track has a motivating use case, usually an abstraction of a user task. In practice, TREC consists of:

- A set of documents;
- A set of information needs (called **topics**);
- Relevance judgements that indicate which documents should be retrieved by which topics.

The result of a retrieval system executing a task on a test collection is called **run**. The technique first used to select the sample of documents to present to a human judge is **pooling**: the top results for a set of runs are combined to form a pool and only those documents are judged. Since this method automatically assumes that all unpooled documents are not relevant (so they remain unjudged), alternative methods have been investigated by TREC tracks to obtain judgements that support fair evaluation.

Note that TREC does not contain any query, and only generic user needs. Participants are free to define (manually or automatically) actual queries for their specific IR system.

2.1.1 Measures

Evaluation of Unranked Retrieval Sets

Precision and **recall** are binary assessments commonly used to evaluate the effectiveness of an IR system. They are defined on the basis of a set of counts, described by the table below.

	Retrieved	Not retrieved
Relevant	TP	FN
Non-relevant	FP	TN

The two measures are then defined as:

- **Precision:** fraction of retrieved documents that are relevant.

$$Precision = \frac{TP}{TP + FN}$$

- **Recall:** fraction of relevant documents that are retrieved.

$$Recall = \frac{TP}{TP + FP}$$

The **F-Measure** (or **F-Score**) is another metric which condenses both precision and recall; it's calculated as the harmonic mean of the two:

$$F = 2\frac{Prec.Rec.}{Prec. + Rec.}$$

The harmonic mean is alway less or equal than the arithmetic/geometric mean: if the two values are very different, the harmonic mean is closer to their minimum. A weighted variant also exists; let α be the weight assigned to precision and β the weight assigned to recall, such that $\alpha = 1/(1 + \beta^2)$, **weighted F-Measure** is calculated as:

$$F_\beta = (\beta^2 + 1)\frac{Prec.Rec.}{\beta^2 Prec. + Rec.}$$

Evaluation of Ranked Retrieval Results

Precision, recall, and F-score are set-based methods, meaning that they are computed using an unordered set of documents. They can be extended to evaluate the ranked retrieval results returned by search engines. Some of these measures are **Mean Average Precision** (**MAP**), **Precision@K** (**P@K**), **Mean Reciprocal Rank** (**MRR**) for binary relevance, and **Normalized Discounted Cumulative Gain** (**NDCG**) for multiple levels of relevance.

Mean Average Precision Consider a set of documents returned by the SE, ordered by rank. Consider the indexes $K_1, \ldots, K_R$ at which recall increases. Precision is calculated at each K_i, considering only the documents with lower indexes, and the average across all K_i is calculated at the end. MAP is then calculated as the mean of the averages across multiple queries/rankings.

MAP is a macro-averaging measure; each query counts equally, even if only few documents are relevant for certain queries and many are relevant for other queries.

Precision@K MAP takes into account all documents returned by the query. However, especially in Web searches, the number of retrieved documents may be unknown or very high; what truly matters is how any good results are found in the first few pages. To calculate Precision@K, we set a rank threshold K, we compute the number of relevant documents among the top K ranking ones, and calculate P@K as:

$$P@K = \frac{\sum relevant\ documents\ in\ top\text{-}K}{K}$$

i.e., it is the fraction of relevant documents across the top-K retrieved ones. In a similar fashion, we can also calculate **Recall@K** as the fraction of relevant documents that are retrieved among the top-K.

P@K by itself does not average very well over a set of queries, since the number of relevant documents for a query affects the result. Anyway, it (along with **MAP@K**) are largely used for Web searches and recommender systems.

Mean Reciprocal Rank Suppose there is only a single relevant document for a given query. A way to approximate the rank of the correct answer could be to consider the search duration for the user: if he/she takes a long time to find the answer its ranking is low, if instead he/she finds it quickly it is ranked high.

Consider the rank position $rank_i$ of the first relevant document returned by a query $q_i \in Q$. The **Reciprocal Rank** is calculated as:

$$\mathbf{RR} = \frac{1}{rank_i}$$

MRR is the mean of the RRs across multiple queries.

Evaluation of Non-Binary Relevance

A popular measure used to evaluate web searches with non-binary relevance is **cumulative gain**, and in particular **Normalized Discounted Cumulative Gain** (**NDCG**). The idea is that the lower the rank of a relevant document is, the less it is useful for the user, since it is less likely to be visited.

Discounted Cumulative Gain uses graded relevance (or **gain**) as a measure of usefulness; it is accumulated starting at the top of the ranking and may also be discounted (= reduced) at lower ranks. The typical discount is $\frac{1}{\log(rank)}$. The formula used to calculate DCG is:

$$DCG = r_1 + \frac{r_2}{\log_2 2} + \frac{r_3}{\log_2 3} + \cdots + \frac{r_n}{\log_2 n} =$$
$$= r_1 + \sum_{i=2}^{n} \frac{r_i}{\log_2 i}$$

This case uses base 2 for the logarithm, but any other base can be used as well. As for the other measures, it can be calculated only for the p top ranking documents instead of the whole set of retrieved ones.

An alternative formulation makes it so that high relevance judgements become much more important:

$$DCG = \sum_{i=1}^{n} \frac{2 * r_i - 1}{\log_2(1+i)}$$

This variant is used by some web search companies.

NDCG is the normalized version of DCG, and is calculated as the ratio between the DGC of a response and the ideal DGC of a perfect ranking; the prefect ranking would be one that first returns all the docuents with the highest relevance level, then the next highest relevance level, and so on.

$$NDCG = \frac{DCG}{Ideal\ DGC}$$

NDCG takes values between 0 and 1.

Chapter 3

Efficient Algorithms for Modern CPUs

Modern IR systems must manage billions of documents and queries, so we need efficient algorithms that can handle such amounts of data, as well as scale across global infrastructures. Efficiency does not only mean a better user experience, but also a reduction in costs, both for computing and cooling systems. Additionally, efficient systems consume less energy, and therefore are more environmentally sustainable.

Computers use several layers of cache on their chips to speed up RAM and disk access time. When an instruction or a block of data must be read, it is also stored accordingly into the cache(s), since they are faster to access. The smaller the cache, the faster the access time. Ideally, programs should take in consideration how cache layers are used and exploit temporal and spatial locality.

- **Temporal locality** states that when a block of data has been accessed, it is likely to be accessed again very soon. Cache replacement policies such as LRU make sure to keep the data that is most likely to be needed.
- **Spatial locality** states that when a block of data has been accessed, it is likely that nearby memory locations will also be accessed in the near future.

  Normally, when a location is accessed, a larger chunk of memory is read (a cache line of 64 bytes). **Hardware prefetchers** can observe the behaviour of a program and prefetch data if repetitive patterns of cache misses appear.

3.1 Parallelism

Another interesting thing to consider is **parallelism**. Parallelism can speed up a CPU through:

- **Pipelining**, which overlaps the execution of multiple instructions so that different parts of the CPU are kept busy at the same time;
- **Superscalar processors**, which have multiple execution units that process independent operations simultaneously;
- **SIMD**, which are a special kind of instructions executing the same operation on more data at the same time.

Pipelining When an instruction is executed, it actually goes through multiple stages on the CPU. The most simple pipelining is a 5-stage one: fetch, decode, execute, load/store, write. The time needed to move an instruction from one stage to the other defines the **clock time**, so it is chosen to accomodate the longest possible operation (usually memory access).

Modern high-performance CPUs have multiple pipeline stages, usually 10-20, but may be more. This means that the latency to execute something simple like an `add` operation may need up to 20 or more cycles. **Latency** is the total time that an operation passes in the pipeline, while **throughput** of a CPU is the number of instructions that are completed and exit the pipeline per unit of time.

Pipeline hazards are situations where the next instruction cannot go forward in the pipeline in the next clock cycle. This can happen because of:

- **Structural hazards**, when one or more instructions must wait because another one further in the pipeline is using a component. Thiese hazards are unavoidable.
- **Data hazards**, when an instruction must wait for an operand to be computed from a previous step. They can be avoided by restructuring computation;
- **Control hazards**, when the CPU cannot tell which branch in an `if-else` statement it must choose. Normally, it will choose randomly and keep loading instructions into the pipeline, and, if the choice turns out to be wrong, the pipeline will be flushed to accomodate the correct branch (thus wasting some cycles).

Regarding the last type, thankfully CPUs have **branch predictors** capable of guessing which branch is the more likely to be picked by observing past behaviour. The branch predictor is capable of noticing particular patterns, such as a condition holding true/false for several loop iterations, or a condition alternating between true/false between successive instructions.

Superscalar Processing In superscalar processing, multiple instructions are dispatched do different execution units on the core (each core has several specialized ALUs). This type of parallelism is also called **instruction-level parallelism**.

Single Instruction, Multiple Data SIMD instructions operate n special registers that hold 128, 256, or even 512 bits. Data in registers is divided into blocks of 8, 16, 32, or 64 bits.

SIMD instructions can be used in two ways: either through **auto-vectorization**, meaning that the compiler automatically converts scalar operations into SIMD ones, or they are explicitly used by the programmer in the code. In the latter case, there is an higher level of control over which operations are optimized (since the automatic conversion done by the compiler can only optimize simpler instructions), but obviously requires detailed hardware knowledge.

Chapter 4

Natural Language

Natural language is the language used by humans to communicate with each other. Languages are defined on the basis of thousands of words, a complex syntax, and a mostly compositional semantic. Language is also often ambiguous; the same word may have different meanings depending on the context of the sentence, or the same sentence may be interpreted in different ways.

Natural Language Understanding has the aim of building machines capable of receiving and giving information using natural language, like a human would. Natural language processing is said to be an "AI-complete" problem, meaning that it is a problem only solvable using AI, and that if we were to find a solution, then we'd have a solution for any other problem. In practice, natural language processing is used in many applications, such as chatbots, search engines, machine translation, and personal assistants (such as Alexa by Amazon, or Google Home by Google). It can be used with different data types: tabular data (to construct and interpret search queries), graphs (to represent the content of each node and link, for example ina social media graph), and images/video data (to describe the content of the image/video and to retrieve or generate similar images/videos).

Information Retrieval is strictly connected and overlapping with the fields of Natural Language Processing and Machine Learning:

- NLP methods are often built on top of IR or ML methods;
- ML uses IR measures to define the goals of the learned models, and assumes language can be manipulated via NLP.

4.1 NLP Pipelining

A **processing pipeline** is a sequence of preprocessing steps aimed at transforming the raw text input into a format that can be effectively used as input to the chosen machine learning model(s). Some of the key steps in the pipeline are:

- **Tokenization**: it identifies the words (tokens) in the text. Popular libraries provide "language aware" tokenization, i.e., it isolates tokens differently depending on the language or context of the text. After this step, a **vocabulary** of the terms used can be constructed.
- **Sentence splitting**: it isolates whole sentences from the text, so that they can be analyzed individually. This is not always an easy task, as punctuation marks can often be used for uses other than sentence separation (e.g., in acronyms, numbers, initials, etc.).
- **Stemming and lemmatization**: both aim at reducing words to their roots; stemming does it by applying a set of language-dependent transformation rules to find the stem of a word, while lemmatization actually tries to find the root of the word in the vocabulary. The difference between the two can be shown with an example: the word "cars" is both stemmed and lemmatized to "car"; the word "was" may be stemmed to "wa", but lemmatized to "be".

Raw text can be transformed in different formats depending on the task to be solved. Common models are bag-of-words and n-grams.

Bag-of-words represents a text as the list of unique words appearing in it. It loses any information about word frequency and order, requiring external data structures to store this information; on the other hand, it is easy to implement and calculate. The set of all distinct extracted words can be called *dictionary*, *vocabulary*, or **feature set/space**, since each word becomes a feature of the new representation.

N-grams features are sequences of n words which capture word order. They do not correspond to a specific token in the text, and are instead obtained as a combination of them. For example, in the text:

"the quick brown fox jumps over the lazy dog"

we can find 2-grams such as:

'the quick", 'quick brown', 'brown fox', ... , 'the lazy', 'lazy dog'

Different libraries have specific formats to identify n-grams.

4.2 Zipf's Law

Zipf's Law is an empirical law used to model the frequency of words in a text. It is based in the observation that the most common word in a text is usually twice as frequent as the second most common one, three times as frequent as the third most common, and so on. Specifically, the law states that the frequency of a word is proportional to the inverse of the rank:

$$frequency = \frac{1}{(rank + b)^a}$$

where $a \approx 1$, $b \approx 2.7$.

The **principle of least effort** is a theory that states that animals, people, and even well-designed machines naturally choose the path of least effort, meaning the one that requires the least amount of work to reach a goal. In the case of human communication, both speaker and listener in a conversation will abide by this principle. The speaker tends to use a small vocabulary of common words, while the listener tends to prefer longer, rarer words. Zipf's Law can be seen as the result to the compromise between the two.

Stopwords are the most common words in a language. In the English lanaguage, stopwords include "the", "a", "to". They can be removed from the text without losing too much information. Different libraries and tools provide their own list of stopwords depending on the application; for example, MySQL specifies words such as "appreciate" or "unfortunately", since it's used for sentiment analysis.

When it comes to rare words, it is not necessarily true that uncommon words are the most informative or useful. If a word is so rare that it appears very few times in very few documents, it may actually be of little help for future retrieval: it could be a typo, or a sort of artificial identifier associated to the document (e.g., a slug in a url). Removing these rare words can help make it faster to process indexed data, and requires less space.

4.3 Vector Space Model

Words can be represented as $|F|$-dimensional vectors obtained through **one-hot encoding**, where F is the set of distinct features (words, n-grams, etc.) in the collection. The relevance of a feature f in the document d is indicated as $w_{f,d}$.

A document can then be represented as the weighted sum of all the vectors of its features:

$$v(d) = \sum_{f \in d} w_{f,d} v(f)$$

These vector representations are usually sparse (most of their terms are equal to 0). Weights can be assigned in different ways. Common methods are binary weights (bag-of-words), term frequency (tf), and tf-idf.

To find matches between a query and a document, we can compare the terms appearing in both: using a simple boolean operator (AND/OR), we can check wheter a certain document contains only/all the terms in the query and return them to the user. This is a very simple methodology, but it does not produce a ranking and does not take into account the frequency of words.

4.3.1 Vector Space Ranking

In **ranked retrieval**, the system reorders the (top k) documents in the collection for a given query. Instead of defining binary queries with a specific language, natural language is used instead. To rank documents, we need some way to assign a **score** to each document given a query that measures how well they match.

A possible scoring is given by **Jaccard's coefficient**:

$$jaccard(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

where A and B are sets (in our case, the query and the document). This coefficient is always a value between 0 and 1, where 0 means no overlap and 1 means perfect overlap. While a better alternative than basic binary scoring, it is based on the assumpion that its operators are sets, so it does not consider term frequency to calculate the score. The more frequent is the term in the document, the higher the score; the rarer the term in a collection, the more informative it is (with the exceptions mentioned above).

We introduce two data structures to then define more sophisticated scoring measures: the incidence matrix and the count matrix.
The **incidence matrix** is used to represent the presence of absence of terms in documents. Each row corresponds to a word, and each column corresponds to a document.

	d_1	d_2	d_3	d_4
w_1	1	0	1	0
w_2	0	1	0	1
w_3	1	1	0	0
w_4	0	0	1	1

Table 4.1: Example of an incidence matrix.

The **count matrix** keeps track of the number of occurrences of a term in a document, so each column of the matrix is a count vector, and each count vector is a multiset.

	d_1	d_2	d_3	d_4
w_1	32	0	5	0
w_2	0	15	0	13
w_3	2	1	0	0
w_4	0	0	10	2

Table 4.2: Example of a count matrix.

Note that this representation still does not consider the order in which words appear in the documents.

The **term frequency** $tf_{t,d}$ of a term t in a document d is simply the number of times t appears in d. Raw term frequency cannot be used as is, however, since relevancy is not linearly proportional to it. A document with 10 occurrences of a term is more relevant than a document with only 1 occurrence, but it is not 10 times more relevant. **Log-frequency weight** is a common way to normalize term frequency:

$$w_{t,d} = \begin{cases} 1 + \log_{10}(tf_{t,d}) & tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

The score of a document-query pair is then calculated by summing the weights of all the terms appearing in both:

$$score(q, d) = \sum_{t \in q \cap d} w_{t,d}$$

The score is always 0 if none of the terms of the query appear in the document.

However, we previously mentioned that informativeness also depends on how frequent the word is in the entire collection. If a term is globally rare, documents containing that word will be more relevant for queries asking for it. To capture this aspect, **document frequency** df_t is used: it is defined as the number of documents containing term t. Since df_t measures the inverse of the informativeness of a word, **inverse document frequency** (idf_t) is used instead.

Bibliography