Information Retrieval Cheatsheet

Edoardo Riggio

December 18, 2021

Information Retrieval - SA. 2021
Computer Science
Università della Svizzera Italiana, Lugano

Contents

1 Introduction

1.1 Text Information Systems

Text Information Systems involve three main capabilities:

- **Text Retrieval**

  Information Retrieval is a field concerned with the structure, analysis, organization, storage searching, and retrieval of information.

- **Text Analysis**

  Analyze large amounts of text data in order to discover interesting patterns buried in text.

- **Text Organization**

  Annotate a collection of text documents with meaningful topical structures so that scattered information can be connected and navigated.

While text retrieval is part of information retrieval, text analysis and text organization are part of text mining.

Differently from queries done on DBMS, queries in search engines make use of natural language. It is much harder to compare the text query to the document text and determining what is a good match and what is not a good match. This is the core issue of information retrieval. There are many different ways of writing the same thing, thus an identical matching of words is not enough.

1.2 Relevance

A document is said to be relevant when it contains the information that a person was looking for when he/she submitted the query tot eh search engine.

In order to understand what the user is asking for in the query, we use something that is known as **NLP** (Natural Language Processing). NLP is concerned with developing techniques for enabling computers to understand the meaning of natural language text.

2 Text Access

Text data access is the foundation for text analysis. The general goal of text data access is to connect users with the right information at the right time.

Connection with users can be done in two ways:

- **Pull**

The user takes initiative in order to fetch relevant information from the system.

- **Push**

  The system takes initiative in order to offer relevant pieces of information to the users.

2.1 Access Mode: Pull vs Push

In **pull** mode, the user initiates the access process in order to find the relevant text data. When a user has such need, then this can be done in two different ways:

- **Querying**

  The user can use a query in order to obtain ad hoc information. This mode of research is done by using a few – yet specific – words.

- **Browsing**

  It is a way of accessing text data, and can be very useful for users when they do not know how to formulate an effective query.

In **push** mode, the system initiates the process to recommend a set of relevant information items to the user.

Broadly, there are two kinds of information needs:

- **Short-Term Needs**

  These needs are often associated with pull mode. This type of information need is temporary and usually satisfied through searching or browsing.

- **Long-Term Needs**

  These needs are often associated will pull mode. This type of information need can be better satisfied through filtering or recommendation of the system to the user.

Finally we have **browsing traces**. These traces happen whenever a user does any kind of browsing. They are used in order to model the behaviour of the user, and make more precise recommendations.

2.2 Search Engine Architecture

A **software architecture** consists of software components, the interfaces provided by those components, and the relationships between them.

The software architecture of a search engine is determined by the following requirements:

- **Effectiveness**
- **Efficiency**

An information retrieval process can be divided into four subprocesses:

1. **Indexing Process**

   This process is composed of several different stages. First we have **text acquisition**, in which the system identifies and stores documents for indexing.

   Second we have **text transformation**, in which the system transforms documents into index terms or features.

   Finally we have **index creation**, in which the system takes index terms and creates data structures to support fast searching.

2. **Query and Retrieval Process**

   It is composed of the following stages. **User interaction**, which supports the creation of a query and displays the results.

   Next we have **ranking and retrieval**, in which the query and indices are used in order to generate a ranked list of documents.

   Finally we have **evaluation**, which monitors and measures the effectiveness and efficiency of the information retrieval system.

3. **Relevance Feedback Process**

   It is composed of three parts. The first part is the **user evaluation**, where the user assesses the effectiveness of the system.
   Then we have **user feedback**, which supports refinement of the query and display of the result.
   Finally we have **ranking and retrieval**, where the system generates a ranked list of the documents.

2.3 Formal Definition of Information Retrieval

Information retrieval is composed of several elements, such as:

- **Vocabulary**

  Vocabulary is defined as:

  $$V = \{w_1, w_2, ..., w_n\}$$

  This represents the vocabulary of a language.

- **Document**

  A document is defined as:

  $$d_i = d_{i1}, ..., d_{iM}$$

  Where $d_{ij} \in V$.

- **Collection**

  A collection is defined as:

  $$C = \{d_1, ..., d_M\}$$

  this represents a collection of documents.

- **Query**

  A query is defined as:

  $$q = q_1, ..., q_M$$

  Where $q_j \in V$.

- **Relevant Documents**

  This set is defined as:

  $$R(q) \subseteq C$$

  And it is generally unknown and user-dependent. The query acts as a hint on which the document must be contained in $R(q)$.

2.4 How to Compute $R'(q)$

$R'(q)$ can be computed in one of two ways:

- **Document Selection**

  This can be described as:

  $$R'(q) = \{d \in C \mid f(d, q) = 1\}$$

  Where $f(d, q) \in \{0, 1\}$ and is an indicator function/binary classifier. Here the system must decide whether a document is relevant or not – i.e. **absolute relevance**. This is also known as **boolean retrieval**.

- **Document Ranking**

  This can be defined as:

  $$R'(q) = \{d \in C \mid f(d, q) > \Theta\}$$

  Where $f(d, q) \in \mathbb{R}$ is a relevance measure function, and Θ is a cutoff determined by the user. Here the system only decides if one document is more likely relevant than another – i.e. **relative measure**. This is also known as **ranked retrieval**.

3 Implementation of an IR System

An information retrieval system is mainly made up of four components:

1. **Tokenizer**

   This component takes in documents as raw strings and determines how to separate the large document into separate tokens.

2. **Indexer**

   This module processes documents and indexes them with appropriate data structures. This module can be ran offline.

3. **Scorer/Ranker**

   This module takes a query and returns a ranked list of documents.

3.1 Indexing

The main role of the indexer is to convert documents into data structures in order to enable fast search. The **inverted index** is the dominant indexing method for supporting basic search algorithms.

This data structure is composed of two parts:

- **Lexicon**

  It is a table of search-specific information, such as document frequency and where to find in the postings the per-document term counts

- **Posting File**

  It is a mapping that goes from any term integer ID to a list of documents IDs and frequency information of the term in those documents.

Before obtaining the inverted index we need to pre-process the documents in order to extract only the features we are interested in.

3.2 Zipf's Law

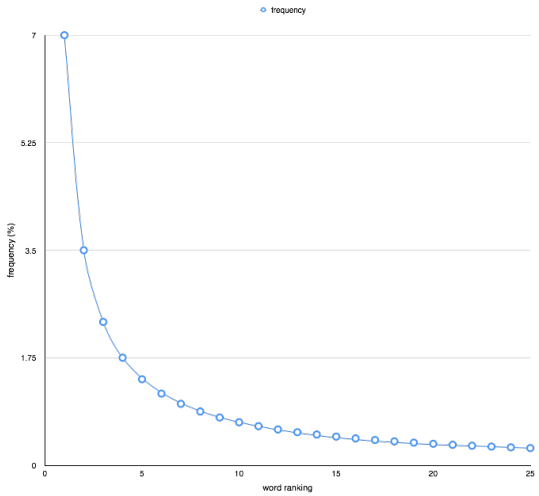
The distribution of words in documents is very skewed. This means that only a few words occur often, while many other words occur rarely.

Zipf's Law says that the rank r of a word times its frequency f, is approximately a constant k – assuming that the words are ranked in decreasing order of frequency. Thus we have the following formula:

$$r \cdot f \approx k$$

In the following case, for example, the words on the far right – with $x = [0, 7.5)$

– are high frequency words, thus useless, the words in the middle – with $x = [7.5, 15)$ – are intermediate frequency words, thus very useful, and the words in the far right – with $x = [15, 25]$ – are rare words, thus they might be useful.



3.3 Text Indexing

Text indexing can be divided into the following steps:

1. **Tokenization**
2. **Stopword Removal**
3. **Stemming**
4. **Detecting Phrases**
5. **POS Tagging and N-Grams**
6. **Processing Document Structure and Markup**
7. **Named Entity Recognition**
8. **Link Analysis**
9. **Build the Inverted Index**
10. **Compress the Inverted Index**

3.3.1 Tokenizing

To tokenize means to break down words into appropriate sequences of characters.

The first step is to use the **parser** in order to identify the appropriate parts of the document that need to be tokenized.

3.3.2 Stopping Removal

Some words have little to no meaning on their own and occur very frequently. These are treated as stopwords and removed. Although, sometimes, they could be important in combination with other words.

3.3.3 Stemming

Many morphological variations of words exist. In most cases, these have the same or very similar meanings. The goal of **stemmers** is to attempt to reduce morphological variations of words to a common stem.

For example, the words *consign*, *consigned*, *consigning* and *consignment* can all be reduced down to **consign**.

There are two main types of stemming approaches:

- **Algorithmic Approach**

  In this case it is a program that determines related words. This could give some false positives and many false negatives.

- **Dictionary-Based Approach**

  In this case a list of related words is used. Endings are removed based on a dictionary. This approach produces real words, not stems, and it's much more precise – but more expensive to run.

Some used stemmers are:

- **Porter Stemmer**

  This is an algorithm used since the 70s. It consists of a series of rules designed to remove the longest possible suffix from a word at each step. It produces stems, not real words. Sometimes this algorithm can be too aggressive or too weak. Porter2 stemmer was created in order to address some of the issues Porter had.

- **Krovertz Stemmer**

  This is a hybrid algorithmic-dictionary stemmer. The word is checked in a dictionary. If the word is present, then the word is either left alone or replaced with "exception", if the word is not present, the word is checked

for suffixes that could be removed. Finally, after the removal, the dictionary is checked again.

In this case words are generated, not stems. Furthermore, this stemmer has lower false positives, but higher false negatives.

3.3.4 POS Tagger and N-Grams

POS (Part Of Speech) tagging is the process of marking up a word in a text as corresponding to a particular part of speech, based on both its definition and its context. POS taggers use statistical models of text in order to predict syntactic tags of words.

Since POS tagging is too slow for some collections, **N-Grams** also exist. These are typically formed from overlapping sequences of words. Frequent N-Grams are more likely to be useful phrases. N-Grams follow Zipf's distribution.

3.3.5 Build the Inverted Index

In order to construct an inverted index, we use sort-based methods by following these steps:

1. Collect local tuples, such as term IDs, Doc IDs and frequency;
2. Sort the local tuples;
3. Perform pair-wise merge runs;
4. Output the inverted file.

While the dictionary part of the inverted index is of modest size, the postings part is huge and stored on disk.

3.4 Ranking Documents

The formula that is used in order to rank documents is the following.

$$f(q,d) = f_a(h(g(t_1,d,q),...,g(t_k,d,q)), f_d(d), f_q(q))$$

Where $f_d(q)$ and $f_q(q)$ are pre-computed. Moreover, a score accumulator is maintained for each d in order to compute h. Finally, for each query term t_i, the following inverted list is fetched.

$$\{(d_1,f_1),...,(d_n,f_n)\}$$

In order to improve the efficiency of the ranker, one could use caching, and keep only the most promising accumulators. There is no need for parallel processing.

4 Retrieval Models

The retrieval process is based on a retrieval model which matches a query with a document. There are at least two classes of retrieval models:

- **Set-Based Models**

  These are models like the Boolean model, and are defined by the following function:

  $$f(q, d) = \{0, 1\}$$

- **Similarity-Based Models**

  These are models such as the vector space model, the probabilistic model... They are defined by the following function:

  $$f(q, d) = \text{similarity}(q, d) = [0, \infty)$$

4.1 Examples of Retrieval Models

4.1.1 Boolean Model

A boolean model can only have two possible outcomes for query processing: true or false. It is an exact-match retrieval process, and the simplest for of ranking.

A query is usually specified using boolean operators (such as AND, NOT, OR...), and is used in DBMSs.

Some advantages of boolean retrieval are:

- The result is predictable;
- Many different features can be incorporated;
- Efficient query processing.

While the disadvantages are:

- The effectiveness of the query solely depends on the user;
- Simple queries do not usually work well;
- Complex queries are difficult both to think and to write.

4.1.2 Ranked Retrieval

In ranked retrieval, documents are presented according to how much they match the query. In a good ranking function relevant documents should be ranked on top of non-relevant ones.

4.2 Designing Retrieval Models

All retrieval models are based on the assumption of using a **bag-of-words** representation of text. A bag-of-words is a model in which a text is represented as the multiset of its words, disregarding grammar and even word order.

In order to design a retrieval function, we require a computational definition of relevance. Moreover, we need to use features such as:

- **Term Frequency (TF)**

  This represents how many times does one term appear inside of each document.

- **Document Length**

  If a term occurs in a long document many times, it is not as significant as a term that occurs the same number of times inside a short document.

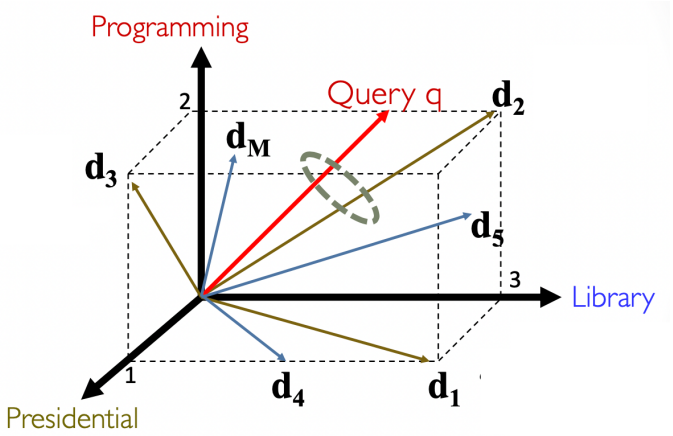
- **Document Frequency (DF)**

  This represents how often a term appears at least once in any document of the entire collection.

4.3 Vector Space Model

This is a simple, yet effective, way of designing ranking functions for information retrieval systems. This is a special case of similarity-based models, where we assume that relevance is roughly correlated to the similarity between a query and a document.

In this representation, each dimension of a highly dimensional space represents a term. We can plot the documents in the collection as vectors of term magnitude.



This is a framework, thus it needs to be defined. In order to use this framework we need the following:

- **Dimension Instantiation**

  The number of dimension can be defined by the number of words inside of a bag-of-words.

- **Vector Placement**

  In order to place the vector inside of the framework, we can use bit vectors. These bit vectors can only be either 0 or 1. We will have a 1 if the word w_i – one of the words of the axes – is present in the document or query, 0 otherwise.

- **Similarity Instantiation**

  The similarity between a query and a document can be defined as the dot product between these two vectors. In this case the formula would be:

$$\text{sim}(q, d) = \langle q, d \rangle = \sum_{i=1}^{N} x_i \cdot y_i$$

By using only these definitions and instantiations, the framework will be limited. For example, wouldn't it be correct to give more credit to terms that appear more times in a document or query? The current model does not allow it. This is why we can introduce some improvements to the current model, such as:

- **Term Frequency Vector**

  An improvement can be applied by representing a frequency vector. This vector will represent the number of times that the word w_i – one of the words of the axes – is contained in the query or in the document.

- **Term Frequency Weighting (TFW)**

  This improvement consists in using the same formula as the one used in the similarity instantiation, but with q and d represented by term frequency vectors.

- **Inverse Document Frequency**

  This improvement consists in giving a weight to words. The formula for computing y_i is given by:

$$y_i = \text{count}(w_i, q) * \log\left(\frac{M+1}{k}\right)$$

  Where $\text{count}(w_i, q)$ is the number of times the word w_i appears in document d, M is the total number of documents in a collection, and k is the total number of documents containing the word w.

The weight of each word inside of the collection of documents is computed by the second part of the previous formula, i.e.

$$\log\left(\frac{M+1}{k}\right)$$

4.3.1 TF Transformation

Since the inverse document frequency still has some problems, we can use TF transformation in order to normalize the number of times a word appears in a collection. This method is used in order to avoid the dominance of a single term above all the others.

This transformation, is mathematically defined as follows:

$$f(q,d) = \sum_{w \in q \cap d} \text{count}(w,q) \cdot \frac{(k+1)\cdot \text{count}(w,d)}{\text{count}(w,d)+k} \cdot \log\left(\frac{M+1}{df(w)}\right)$$

Where $k \geq 0$ is a constant, $w \in q \cap d$ indicates all matched query words in the document, and $df(w)$ indicates the document frequency of the word w.

4.3.2 Document Length Normalization

Since a long document has a higher chance of matching any query, it must be penalised – but not too much – by using a document length normalizer. In order to not over-penalise a long document, we say that a document with more words must be penalised more, while a document with more contents must be penalised less.

In order to normalize the document lengths, we use the average document length as a pivot. The formula of the document length normalizer is the following:

$$1 - b + b \cdot \left(\frac{|d|}{\text{avgdl}}\right)$$

Where $b \in [0,1]$ is a constant and *avgdl* is the average length of the documents in the collection.

4.4 Probabilistic Models

In probabilistic models, we define the ranking function based on the probability that a given document d is relevant to a query q.

$$p(R = 1 \mid d, q)$$

Where $R \in \{0,1\}$ is a binary random variable that denotes relevance. In order to define relevance in a model such as this, we use the following formula.

$$p(R = 1 \mid q, d) = \frac{\text{count}(R = 1, d, q)}{\text{count}(d, q)}$$

4.5 Query Likelihood Retrieval Model

In query likelihood models, our assumption is that the probability of relevance can be approximated by the probability of a query given a document and a relevance.

$$p(q \mid d, R = 1)$$

This captures the following probability: if a user likes document d, how likely would the user enter query q in order to retrieve document d? In this model we assume that the user imagines some ideal document, and generates a query based on that ideal document.

4.6 Statistical Language Model

A statistical language model is used in order to compute the probability of text. For example:

$$\begin{aligned} p(\text{"}\textit{Today is Wednesday}\text{"}) &\approx 0.001 \\ p(\text{"}\textit{Today Wednesday is}\text{"}) &\approx 0.0000000000001 \\ p(\text{"}\textit{The eigenvalue is positive}\text{"}) &\approx 0.00001 \end{aligned}$$

This probability distribution over word sequences is highly context- and user-dependent. This is also called a generative model.

Language model is useful to quantify the uncertainties in natural language. It can be particularly useful in:

- **Speech Recognition**
- **Text Categorization**
- **Information Retrieval**

4.6.1 Unigram Language Model

Unigram language model is the simplest form of a language model. It generates text by generating each word independently. Thus:

$$p(w_1, w_2, ..., w_n) = p(w_1) + p(w_2) + ... + p(w_n)$$

In order to compute a more accurate estimation of unigrams, we could use topic representation. To do so we get:

- **Background Language Model**

  This LM is composed of terms contained in a general background English text. Mathematically this is equal to:

$$p(w \mid B)$$

- **Topic Language Model**

  This LM is composed of terms contained in a collection based on a topic – for example all words that appear in computer science papers. Mathematically this is equal to:

$$p(w \mid C)$$

Given these two LMs, we can compute a normalized LM which can then be used with the original text. The normalized LM is given by:

$$\frac{p(w \mid T)}{p(w \mid B)}$$

4.6.2 Query Generation

We can consider $p(q \mid d)$ to be made up of independent terms. This way the probability becomes a product of the probabilities of of each query word in each document's language model. In order to formally state this, we consider a query q that contains the words:

$$q = w_1, w_2, ..., w_n$$

Such that $|q| = n$. The scoring or ranking function will thus be:

$$p(q \mid d) = p(w_1 \mid d) \cdot p(w_2 \mid d) \cdot ... \cdot p(w_n \mid d)$$

In practice, we score the document for this query by using the logarithm of the query likelihood.

$$\begin{aligned} \text{score}(q, d) &= \log p(q \mid d) \\ &= \sum_{i=1}^{n} \log p(w_i \mid d) \\ &= \sum_{w \in V} \text{count}(w, q) \cdot \log p(w \mid d) \end{aligned}$$

Where V is the vocabulary.

4.6.3 Smoothing the Document LM

In order to estimate the $p(w \mid d)$ in the formula above, we can use the MLE (Maximum Likelihood Estimator). This will make sure that the curve given by $p(w \mid d)$ is smoother.

$$p_{ML}(w \mid d) = \frac{\text{count}(w, d)}{|d|}$$

Now $p(w \mid d)$ will be always positive, even if $count(w, d) = 0$. Now, we also need to consider what probability should be assigned to unseen words. This is

the equation used in order to smooth the curve:

$$p(w \mid d) = \begin{cases} p_{seen}(w \mid d) & \text{if } w \text{ is seen in } d \\ \alpha_d \cdot p(w \mid C) & \text{otherwise} \end{cases}$$

Where $p_{seen}(w \mid d)$ is the discounted MLE, and C is the collection language model. Finally we can rewrite the ranking function accounting for smoothing. The components of the new function are the following:

- **Query Words Matched in *d***

  The formula is:

$$\sum_{w \in V,\ \text{count}(w,d)>0} \text{count}(w,q) \cdot \log(p_{seen}(w \mid d))$$

- **Query Words not Matched in *d***

  The formula is:

$$\sum_{w \in V,\ \text{count}(w,d)=0} \text{count}(w,q) \cdot \log(\alpha_d \cdot p(w \mid C))$$

- **All Query Words**

  The formula is:

$$\sum_{w \in V} \text{count}(w,q) \cdot \log(\alpha_d \cdot p(w \mid C))$$

An the final equation will be:

$$\begin{aligned}
\text{score}(q,d) &= \sum_{w \in V} \text{count}(w,q) \cdot \log p(w \mid d) \\
&= \sum_{w \in V,\ \text{count}(w,d)>0} \text{count}(w,q) \cdot \log(p_{seen}(w \mid d)) \\
&\quad + \sum_{w \in V,\ \text{count}(w,d)=0} \text{count}(w,q) \cdot \log(\alpha_d \cdot p(w \mid C)) \\
&= \sum_{w \in V} \text{count}(w,q) \cdot \log(\alpha_d \cdot p(w \mid C)) \\
&\quad - \sum_{w \in V,\ \text{count}(w,d)>0} \text{count}(w,q) \cdot \log(\alpha_d \cdot p(w \mid C)) \\
&= \sum_{\substack{w \in d \\ w \in q}} \text{count}(w,q) \cdot \log\left(\frac{p_{seen}(w \mid d)}{\alpha_d \cdot p(w \mid C)}\right) + n \log(\alpha_d) + \sum_{w \in V} \log(p(w \mid C))
\end{aligned}$$

By rewriting the formula, the following features are now incorporated in the ranking function:

- **Matched Query Terms**

  This can be seen in:

  $$\sum_{\substack{w \in d \\ w \in q}} \dots$$

- **TF Weighting**

  This can be seen in:

  $$p_{seen}(w \mid d)$$

- **IDF Weighting**

  This can be seen in:

  $$\alpha_d \cdot p(w \mid C)$$

- **Document Length Normalization**

  This can be seen in:

  $$n \log(\alpha_d)$$

Other smoothing methods can be used, two of the main ones are:

- **Linear Interpolation Smoothing**

  Which is also known as Jelinek-Mercer Smoothing.

- **Bayesian Smoothing**

  Which is also known as Dirichlet Prior Smoothing

5 Feedback Models

There are three major forms of feedback:

- **Relevance Feedback**

  The user expressively indicates the documents that are relevant to the query.

- **Pseudo Feedback**

  The system assumes the top retrieved documents as relevant.

- **Implicit Feedback**

  The system monitors what the user does and based on that it makes some assumptions on the document relevance.

5.1 Relevance Feedback

The user can make judgements about whether each returned document is useful or not. With relevance feedback we can perform two operations on the query:

- **Query Expansion**

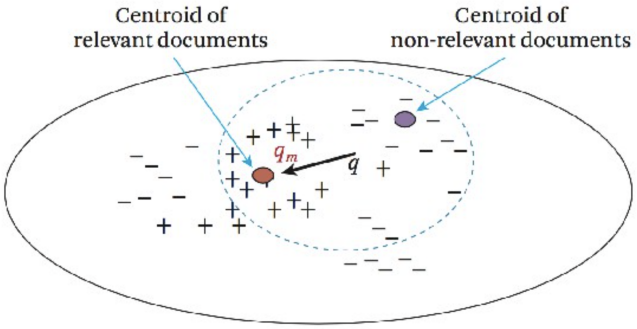
  It is when new weighted terms are added.

- **Query Modification**

  It is when the weights of existing terms are adjusted.

5.2 Rocchio Feedback

The Rocchio Feedback is a relevance feedback algorithm applied to the VSM (Vector Space Model). The following is a representation of the Rocchio Feedback.



The Rocchio algorithm was invented in 1971 and revolves around the concept of an **optimal query**. The optimal query maximises the difference between the average vector representing the relevant documents and the average vector representing the non-relevant documents.

The modified query results in a longer expanded query. This is because the terms that occur frequently in the relevant documents will be added to the modified query and those that occur frequently in non-relevant documents will be removed.

The Rocchio feedback formula is the following:

$$q'_j = \alpha \cdot q_j + \beta \cdot \frac{1}{|R|} \cdot \sum_{D_i \in R} d_{ij} - \gamma \cdot \frac{1}{|NR|} \cdot \sum_{D_i \in NR} d_{ij}$$

Where α is a constant of value 8, β is a constant of value 16, γ is a constant of value 4, R is the group of relevant documents – $|R|$ is the size of that group, and NR is the group of non-relevant groups – $|NR|$ is the size of that group.

6 System Evaluation

Evaluation is concerned with assessing if the system carries out its tasks properly. This is the key to building an effective and efficient search engine.

In an evaluation assessment, three measurements are checked:

- **Efficiency**

  It measures if the information retrieval system carries out its tasks with an optimal use of its resources.

- **Effectiveness**

  It measures if the information retrieval system finds what the user wants to find.

- **Usability**

  It measures how useful is the system for real user tasks.

Furthermore, these measurements are usually carried out in a controlled laboratory environment.

6.1 The Cranfield Evaluation Methodology

This is the primary approach to an evaluation of an information retrieval system's effectiveness. The main idea of this test is to build reusable test collections and to define measures of effectiveness. In order to do this, we need:

- A sample collection of documents;
- A sample collection of queries/topics;
- Relevance judgements;
- Measures to quantify how well a system's result matches the ideal ranked list.

6.2 TREC

TREC (Text REtrieval Conference) is an evaluation forum that started in 1992 in order to encourage research in information retrieval evaluation and to compare information retrieval systems.

6.3 Effectiveness

The two main measurements to evaluate effectiveness in an information retrieval system are **precision** and **recall**.

Precision is a measurement in which we simply compute to what extent all the retrieval results are relevant. On the other hand, **recall** measures the completeness of coverage of relevant documents in the retrieval system.

6.3.1 Classification Errors

There are two main classification error in information retrieval systems, these are:

- **False Positive - Type I Error**

  It is the ratio of non-relevant documents that have been retrieved.

- **False Negative - Type II Error**

  It is the ratio of the relevant documents that have not been retrieved.

6.3.2 F1 Score

The F1 score is the harmonic mean of recall and precision. Its formula is the following:

$$F = \frac{1}{\frac{1}{2}\left(\frac{1}{R} + \frac{1}{P}\right)} = \frac{2RP}{R+P}$$

Where R is the recall and P is the precision of the information retrieval system.

6.4 Computing the Performance of a System

6.4.1 Mean Average Precision

In order to properly compute the performance of a system we need to analyse several queries. To compute the average precision for each query, and to summarize the rankings from multiple queries, we use the **Mean Average Precision (MAP)**.

The MAP measurement assumes that the user is interested in finding many relevant documents for each query. This requires many relevance judgements in the text collection.

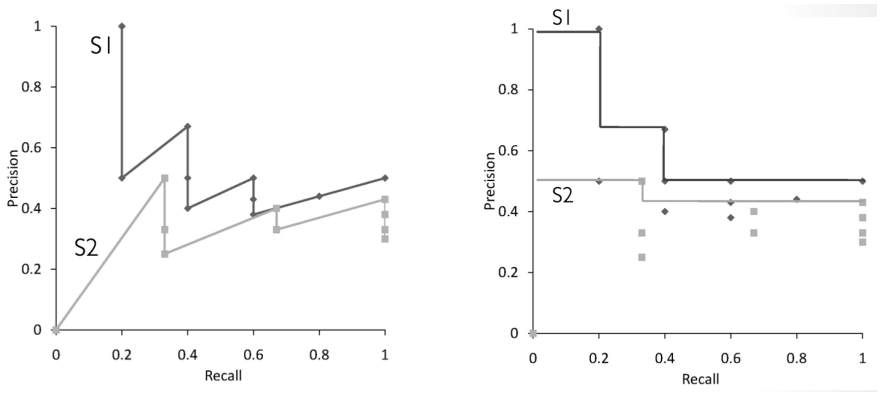
In order to obtain the MAP, first we compute the average precision of each query, and then we compute the mean of these means.

6.4.2 Interpolation of Recall-Precision Graphs

In order to better understand a recall-precision graph, we use the following formula:

$$P(R) = \max\{P' : R' \geq R \wedge (R', P') \in S\}$$

Where S is the set of observed (R, P) points. This formula defines the precision at any recall level as the maximum precision observed in any point in S at a higher recall level. This will produce a step function that is both easier to understand and compare than the original, and allows us to define precision when the $R = 0$.



Here the figure on the *left* represents the original recall-precision graph, while the one on the *right* represents the image on the left after we apply interpolation.

7 User Evaluation

In user evaluation, we test the user interaction in addition to the quality of the information retrieval system. The quality of the interface of the search engine cannot be tested without the user involvement.

In this evaluation we have a low volume of queries and artificial tasks – meaning that the user carries out the search, but in an artificial situation.

7.1 Experimental Design

This is an experimental evaluation, thus it needs to:

1. **Decide the aim of the evaluation**

   We need to decide which are the features that we need to evaluate, and what is the goal of this new feature.

2. **Formulate experimental hypotheses**

   The aims of the experiment need to be expressed as experimental hypotheses to be tested. These hypotheses also tell us some of the information we will need to gather.

3. **Define an experimental methodology**

   We need to gather:

   - **Collections**

     What are we searching?
   - **People**

     Who does the searching?
   - **Systems**

     Usually either two or more systems or two or more versions of the same machines are compared.
   - **Search Tasks**

     Search tasks can be given or can be real. In order to test the user on these tasks, we use an **experimental matrix** – for example a Latin Square Design. Tasks can also be: decision tasks, background knowledge or fact search.

4. **Define criteria for data comparison**

   While quantitative data can be easy to compare, qualitative data is more difficult to compare. In order to compare the latter kind of data, we can use a Likert scale.

5. **Perform experiments**

   Try not to introduce any bias in the experiment, this could alter the results and make the whole output of the experiment unusable.

6. **Analyse the obtained data**

   Use proper methods of statistical data analysis on quantitative data.

7.2 Qualitative and Quantitative Data

Quantitative data measures the search effectiveness. On the other hand, **qualitative data** measures the user's perception of a system. Qualitative data is assessed by using tests for statistical significance, such as: Mann-Whitney test, T-test, Chi-squared tests...

7.3 Operational Evaluation

It is similar to user evaluation, but in this case it deals with real users in real search situations and with real tasks on a real collection.

This is an expensive and difficult-to-run test, but it is a very good test of the system in a real situation.

8 Web Search Engine

A web search engine is divided into three main parts: crawling, indexing and ranking.

8.1 Web Crawling

Find and download pages automatically. The web provides for the collection, which is way too big and constantly growing.

A **web crawler** is a software that crawls web pages and puts them into cache. The following is the process of crawling.

1. Connect to a DNS server
2. The DNS server translates the hostname into an IP address
3. There is an attempt from the scraper to connect to the host using a specific port
4. A `GET` request is sent to the web server

8.1.1 Focused Crawling

Here the crawler attempt to download only the pages that are about a specific topic. This method relies on the fact that pages about a topic tend to have links to other pages on the same topic.

The crawler uses a **text classifier** in order to determine whether a page is on topic.

8.1.2 Deep Web

The 96% of available websites are in the deep web. These websites can be divided into three main groups:

- **Private websites**

  These are websites that might require the user to login, or simply websites that do not have incoming links.

- **Form Results**

  These are websites that can be reached only after entering some data inside of a form.

- **Scripted Pages**

  These are pages that use client-side programming languages to generate links.

8.1.3 Politeness Policies

In order not to flood web servers with thousands of requests all at the same time, politeness policies are used by web crawlers. These are, for example:

- Limiting the number of requests per second
- Respecting the `robots.txt` file of the website

8.2 Ranking Algorithms

Ranking algorithms are used in order to find which websites are the most authoritative and generally better. One way of performing such ranking is by using the famous PageRank algorithm.

PageRank is used to compute the "popularity" of the given webpages. It does so by considering the websites' inlinks.

The **Random Surfer Model** is the basic concept behind PageRank. I uses the following algorithm:

1. Choose a random number r between 0 and 1
2. If $r < \alpha$, then go to a random page
3. If $r \geq \alpha$, then click a link at random on the current page
4. Start again from the beginning

More generally, PageRank is defined by the following formula:

$$PR(u) = \frac{\lambda}{N} + (1 - \lambda) \cdot \sum_{v \in B_v} \frac{PR(v)}{L_v}$$

Where B_v is the set of pages that point to u, L_v is the number of outgoing links from page v, N is the number of pages, and λ is a constant which is typically set to 0.15.

Interesting pages detected by the algorithm fall into two classes:

- **Authorities**

  These are pages that contain useful information. In practice, these pages have many incoming links.

- **Hubs**

  These are pages that list a number of authorities. In practice, these pages have many outgoing links to different authorities.

Authorities are ranked based on the sum of the votes that each Hub has, while the Hubs are ranked based on the sum of votes of the Authorities they are pointing to.

9 Recommender Systems

Information filtering deals with dynamic collections, and generates long and elaborated profiles and filters. It also serves long term information needs. On the other hand, **information retrieval** deals with static collections. In this case it server short-lived queries.

Another comparison is between **information filtering** and **routing**. While **information filtering** chooses documents to show based on a binary decision. **routing** relies on ranking in order to determine the best results. In the first case, we have the problem of documents being too much or too few.

There are three different types of information filtering:

- Content-based information filtering
- Collaborative/Social information filtering
- Use-base information filtering

9.1 Content-Based Information Filtering

This types of filtering makes decisions for individual users based on what the system learned the user likes or dislikes. Interest and preferences are inferred based on previous user feedback.

Some of the downsides of these techniques are:

- The filtering decision is binary
- The initialization phase is based only on the profile text or very few initial examples
- There is a limited relevance judgement ("yes" documents)

9.2 Collaborative Filtering

In this case the filtering decisions for an individual user are based on the judgements of other users. Interest and preference are inferred from that of other similar users.

This kind of filtering is based in user profile comparison, thus cannot work with only one user.

Furthermore, collaborative filtering will make several assumptions, such as:

- Users with the same interests will have the same preferences
- Users with the same preferences will likely have the same interests

- A sufficiently large number of users preferences is available

Some possible improvements are:

- Dealing with missing values by setting defaults
- Using Inverse User Frequency

Some problems, instead, are:

- The size of the data needs to be very big
- There is a problem known as Cold Start. This happens when new documents cannot be recommended because there are no previous references

9.3 Use-Based Information Filtering

This method identifies which documents are semantically relevant to a user profile by inferring relevance from user behaviour.

Some problems with this method are:

- User profiles can be messy
- It can be very intrusive and not respect the user's right to privacy

10 Text Clustering

Text clustering is used in order to group similar objects together by discovering their natural structure.

In order to assess similarity between objects, the user must define the perspective – i.e. a bias.

10.1 Clustering in Text Retrieval

Clusters can be created based on *concepts*, *topics* or *themes*. It can be divided into two main categories:

- **Similarity-Based Clustering**

  Which can be either *agglomerative* or *divisive*.

- **Model-Based Clustering**

  Data is assumed to have been generated from a finite mixture of component models.

This is an unsupervised method of grouping objects together.

10.1.1 Similarity-Based Clustering

In order to perform similarity-based clustering, we need to:

- Provide a clustering bias
- Find the optimal partitioning in order to maximize intra-group similarity, and at the same time minimize inter-group similarity

In order to obtain the optimal clusterings, we need to follow this algorithm:

1. Progressively construct a hierarchy of clusters using one of the following two methods:
   - **Agglomerative Method**

     Gradually group similar objects into larger clusters.
   - **Divisive Method**

     Gradually partition the data into smaller clusters.
2. Start with an initial tentative clustering and iteratively improve it

10.1.2 Agglomerative Hierarchical Clustering

The algorithm for this type of clustering method is the following:

1. Assume a similarity function
2. Gradually group objects in a bottom-up fashion in order to form a group hierarchy
3. Stop whenever some stopping criterion is met

10.2 Clustering Strategies

Several clustering strategies exist, these are:

- **Single-Linkage**

  It creates loose clusters and is based on individual decisions, thus being sensitive to outliers. The clustering cost of this strategy is defined by the following formula:

  $$Co = \min\{\ d(x_i, x_j) \mid x_i \in C_i, x_j \in C_j\ \}$$

  Where Co is the cost, $d(x_i, x_j)$ is the distance between the two elements, and C is a cluster.

- **Complete-Linkage**

  It creates tight clusters and is based on individual decisions, thus being sensitive to outliers. The clustering cost of this strategy is defined by the following formula:

  $$Co = \max\{\ d(x_i, x_j) \mid x_i \in C_i, x_j \in C_j\ \}$$

  Where Co is the cost, $d(x_i, x_j)$ is the distance between the two elements, and C is a cluster.

- **Average-Linkage**

  It creates clusters in between the two clusters above and is based on group decisions, thus not being sensitive to outliers. The clustering cost of this strategy is defined by the following formula:

  $$Co = \frac{\sum_{x_i \in C_i, x_j \in C_j} d(x_i, x_j)}{|C_i|\ |C_j|}$$

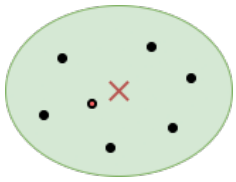
  Where Co is the cost, $d(x_i, x_j)$ is the distance between the two elements, and C is a cluster.

10.3 Cluster Representative

In order to represent a cluster of many points, we use something called a **clustroid**, which is a point that is closer to all other points.

A **centroid**, instead, is the average of all data points in a cluster. This means that it is an artificial point – that can correspond to a real point in the cluster.

The following is a graphical representation of a cluster:



Where the *cluster* is the green oval, the red x is the *centroid*, the black dot is a regular *data point*, and the black dot with the red dot inside it is a *clustroid*.

"Closer to" – relative to data points – can be defined in several different ways:

- The smallest maximum distance to other points
- The smallest average distance to other points
- The smallest sum of squares of distance to other points

The "nearness" of clusters, on the other hand, can be determined using several different approaches:

- Treat the clustroids as if they were centroids and measure the distance between the two clusters
- Use the **intercluster distance** by involving the minimum of the distances between any two points – one from each cluster
- Use the **cohesion of clusters** notion by involving the maximum distance of the cluster to another cluster's clustroid. After doing so, merge the clusters whose union is the most cohesive

10.4 K-Means Clustering

In order to initialize this method, we do the following:

- Assume that we are working in an Euclidean space
- Pick a value for the number of clusters, k
- Initialise the clusters by picking one point per cluster

After having initialized the clusters we can proceed to populate them:

1. For each point, place it in the cluster whose current centroid is the nearest
2. After all points have been assigned, update the locations of the centroids
3. Reassign all points to their closest centroid
4. Repeat steps 2 and 3 until convergence, meaning that the points do not move any more between clusters

10.5 Clustering Evaluation

The evaluation of clustering aims at maximizing the following aspects:

- **Coherence**

  How similar objects in the same cluster are.

- **Separation**

  How far away objects in different clusters are.

- **Utility**

  How useful the discovered clusters are for an application.

In order to perform a good evaluation of clusterings, we need to proceed as follows:

1. Given a predefined test set, have humans create an ideal clustering result
2. Use a system to produce clusters from the same test set
3. Quantify the similarity between the two clusterings

11 Text Categorization

In text categorization object are put into classes – or categories.

Given a set of predefined categories and a training set of labelled text objects, the task is to classify a text object int one or more of the categories.

Categories can be of two different types:

- **Internal**

  They characterize a text object.

- **External**

  They characterize an entity associated with the text object.

11.1 Manual Categorization Method

In this case the category of an object is determined based on rules carefully designed to reflect domain knowledge about the problem.

This method is labour intensive, and does not scale up well. It's also not robust, this is because it can't handle uncertainties in rules.

11.2 Automatic Categorization Methods

Manual categorization method limitations can be fixed with the use of machine learning.

Machine learning is used in order to learn *soft* rules for categorization based on training data. All of the methods rely in discriminative features of text objects in order to distinguish categories. Moreover, they also adjust the weights on features in order to minimize errors.

11.3 Generative Classifiers

In this case the algorithm learn what the data looks like in each category. In order to model the classifiers, we can use the Bayes Rule.

11.4 Discriminative Classifiers

In this case the algorithm learns what features separate each category.

In order to do so, we can use the Support Vector Machine or the k-Nearest Neighbors.

11.5 Feature Selection

A feature is an attribute of an object that we wish to classify.

Feature selection methods reduce the number of features by only choosing the most useful.

11.6 Information Gain

This is a commonly used feature selection measure based on information theory. The model is trained with the top k best features – with k usually being a small number.

11.7 Naïve Bayes Classifier

This is a typical example of a **generative classifier**. It is a probabilistic model based on Bayes' Rule. The formula of this model is:

$$P(C \mid D) = \frac{P(D \mid C) \cdot P(C)}{\sum_{c \in C} P(D \mid C = c) \cdot P(C = c)}$$

Where C is a random variable corresponding to the class, and D is a random variable corresponding to the input – for an example document.

Documents are classified according to the following formula:

$$\begin{aligned} Class(d) &= \arg\max_{c \in C} P(c \mid d) \\ &= \arg\max_{c \in C} \frac{p(d \mid c) \cdot P(c)}{\sum_{c \in C} P(d \mid c) \cdot P(c)} \end{aligned}$$

Where $P(d \mid c)$ is the probability that document d is observed given the class c, and $P(c)$ is the probability of observing class c.

11.7.1 Estimating P(c)

This term is estimated as the proportion of training documents in class c. The formula for this is:

$$P(c) = \frac{N_c}{N}$$

Where N_c is the number of training documents in class c, and N is the total number of training documents.

11.7.2 Estimating P(d | c)

This term is estimated depending on the event space used to represent the documents. The event space is the set of all possible outcomes for a given random variable.

11.7.3 Multiple Bernoulli Event Space

This is a natural way of modelling distributions over binary vectors – which indicate whether a term is present or not inside of a document. The formula for this event space is the following:

$$f(k; p) = p^k(1-p)^{1-k}$$

On the other hand, if we want to check the frequency of a term in a document, we would have to use **multinomial distribution**.

11.7.4 Multinomial Distribution

By using this distribution, the way $P(d \mid c)$ is computed changes, In this case we would have:

$$P(d \mid c) \propto \prod_{w \in v} P(w \mid c)^{tf_{w,d}}$$

Where $P(w \mid c)$ can be either a **Laplacian Smoothed (Jelinek-Mercer) Estimate**, which formula is the following:

$$P(w \mid c) = \frac{tf_{w,c} + 1}{|c| + |V|}$$

Or a **Collection Smoothed (Diricklet) Estimate**, which formula is the following:

$$P(w \mid c) = \frac{tf_{w,c} + \mu \frac{cfw}{|c|}}{|c| + \mu}$$

11.8 Support Vector Machine

This is an example of a discriminative model based on geometrical principles.

The goal here is to, given a set of inputs labelled + and −, find the best hyperplane that separates these two groups.

A hyperplane is a generalization of a line to higher dimensions. It is defined by the vector w. The best hyperplane – in this case – is the one with the maximum margin between the two classes. This margin can be computed by using the following formula:

$$Margin(w) = \frac{|w \cdot x^{-}| + |w \cdot w^{+}|}{||w||}$$

It is generally assumed that $|w \cdot x^{-}| + |w \cdot w^{+}| = 1$. Thus, in order to find the largest margin, we must maximize the following:

$$Margin(w) = \frac{2}{||w||}$$

11.8.1 The Kernel Trick

Since data cannot always be linearly-separable, we can transform it in order to make it linearly-separable. Computing vector math in very high dimensional spaces is costly.

For this reasons we use something known are the kernel trick. It basically allows for very high-dimensional dot products to be computed efficiently.

11.8.2 Non-Binary Classification with SVMs

There are two ways to do it:

- **One Versus All**

  Train the model as a "class c versus not class c" SVM for every class. If there are k classes, k classifiers must be trained. The items are classified based on the following formula:

  $$Class(x) = \arg\max_{c} w_c \cdot x$$

- **One Versus One**

  A binary classifier is trained for every pair of classes, Thus

  $$k\frac{k-1}{2}$$

  Classifiers must be trained. This method is computationally expensive for large values of k.

11.9 K-Nearest Neighbors

The goal of this algorithm is to find k examples in the training set that are the most similar to the text object to be classified, by performing a search for every testing instance.

After this has been done, we assign the category that is most common in these neighbor text objects.

11.10 Classifier Evaluation

In order to evaluate classifiers, we can use one of the following metrics:

- Accuracy
- Precision
- Recall
- F-Measure
- ROC Curve Analysis