Information Retrieval - Notes

Matteo Alberici

September - December 2021

Contents

1 Introduction

Information retrieval (**IR**) systems are used to harness big text data, which differs into three groups:

- **Structured data**: specified type of data (excel file: name, age, date ...)
- **Semi-structured data**: partially specified type of data (email: fields and body)
- **Unstructured data**: all types of data without a specified structure (book pages)

With **text retrieval**, it is possible to obtain small and relevant data from big text data, and with **text mining**, the small data received leads to knowledge.

1.1 Text Information System

Text Information system (**TIS**) involves three capabilities:

1. **Text retrieval**: "*information retrieval is a field concerned with the structure, analysis, organization, storage, searching, and retrieval of information.*" (Gerard Salton, 1968)
2. **Text analysis**: analyze a large amount of data to discover interesting patterns buried in the text
3. **Text organization**: annotate a collection of documents with meaningful topical structures so that scattered information can be connected and navigated

1.2 Database vs. IR Systems

Since databases records are made of well-defined semantics fields, it is easier to compare data than in IR system, in which it is not that easy to compare query text to a document, since having matching words is definitely not enough. **Relevance** represents a big issue in IR since many factors could determine what is relevant and what is not, such as the style, the novelty, the context.

1.3 Natural Language Processing

Natural language Processing (**NLP**) is concerned with techniques for enabling computers to understand the meaning of natural language text.

2 Text Data Access

2.1 Accessing Modalities

There exist different modalities to access text data:

- **Push mode**: the system provides information using knowledge about the user's stable need
- **Pull mode**: the system grabs and returns information selected ad hoc by the user
- **Querying**: the system returns documents concerning user's keywords used in queries
- **Browsing**: the system provides relevant information through which the user navigates, leaving a trace to model the user behavior

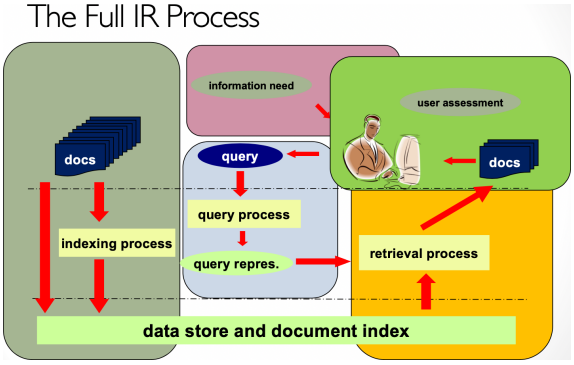
2.2 Search Engine Architecture

A software architecture consists of software components, their interfaces, and the relationships between them. The software architecture of a search engine is determined by its **effectiveness**, indicating the quality of its results, and its **efficiency**, meaning its response time and throughput. Search engines cache a document version when it is visited, and fetch it again when it is newly visited.

2.3 The IR Process

The **IR process** of a search engine is divided into sub-processes:

1. **Indexing process**: documents are indexed and stored offline
2. **Query process**: user's query is analyzed by the system
3. **Retrieval process**: queries representations are compared with indexed documents
4. **Relevance Feedback Process**: documents are returned to the user



2.3.1 Indexing process

The **indexing process** takes as input a document and outputs an index and the stored document:

1. **Text acquisition** (**Crawler**): identifies and stores documents for indexing
2. **Text transformation** (**Analysis**): transforms documents into index terms
3. **Index creation**: creates indexes from terms to support searching

2.3.2 Query and Retrieval Process

1. **User interaction**: supports the creation of queries and display of results
2. **Ranking and retrieval**: generates ranked lists of documents using queries and indexes
3. **Evaluation**: monitors and measures effectiveness and efficiency offline

Documents are stored to be compared with the query and to let the engine extract some snippets which will be displayed with the results.

2.3.3 Relevance Feedback Process

1. **User evaluation**: the user assesses the effectiveness of system
2. **User feedback**: supports refinement of query and of results display
3. **Ranking and retrieval**: the system regenerates lists of documents

2.4 IR semantics

Formal formulation of IR:

- **Vocabulary**: $V = \{w_1,\ w_2,\ \ldots,\ w_n\}$ of language
- **Document**: $d_i = \ d_{i1},\ \ldots,\ d_{im}$ where $d_{ij} \in V$
- **Collection**: $C = \{d_1,\ \ldots,\ d_m\}$
- **Query**: $q\ =\ q_1,\ \ldots,\ q_m$ where $q_i \in V$
- Set of relevant documents: $R(q) \subseteq C$
  - unknown and user-dependent
  - query is a "hint" on which document is in $R(q)$
- Task: compute $R'(q)$, an approximation of $R(q)$:
  - retrieval status value: an estimate R' of the real relevance R of docs to q

2.5 How To Compute $R'(q)$

- Strategy 1: **Document selection**
  - $R'(q) = \{d \in C \mid f(d,q) = 1\}$, where $f(d,q) \in \{0,1\}$ is an indicator function (binary classifier)
  - absolute relevance: the system must decide if a document is relevant or not
  - it is also called Boolean retrieval
- Strategy 2: **Document ranking**
  - $R'(q) = \{d \in C \mid f(d,q) > 0\}$, where $f(d,q) \in R$ is a relevance measure function: 0 is a cutoff determined by the user
  - relative relevance: system only needs to decide if one doc is more likely relevant than another
  - it is also called Ranked Retrieval

In document selection, the classifier is not very accurate due to over- or under-constrained queries and it does not give an order to the relevant documents, while ranking does.

2.6 Theoretical Justification for Ranking

Probability Ranking Principle [Robertson 77]: returning a ranked list of documents in descending order of probability that a document is relevant to the query is the optimal strategy under the following two assumptions:

1. a user would browse the results sequentially
2. the utility of a document is independent of the utility of any other document

3 Implementation

3.1 Document Indexing

Indexing converts documents to data structures to enable fast search. The inverted index is the dominating indexing method for supporting basic search algorithms, but before inverting the index we need to preprocess the documents to extract the relevant features.

3.1.1 Processing Text

It is necessary to convert words to index terms for many reasons such as matching the string typed by the user is too restrictive and not all words are of equal value. There is a huge variety of words used in text, but many **statistical** characteristics of word occurrences are predictable. The distribution of word frequencies is skewed since a few words occur very often and many words rarely or hardly ever occur.

3.1.2 Zipf's Law

Zipf's law states that "*the rank r of a word times its frequency f is approximately a constant (k)*".

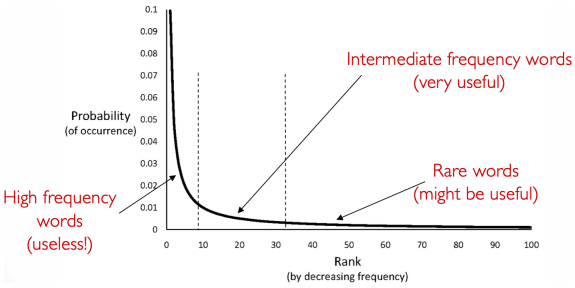


Figure 1: Zipf's law graph

3.2 Text Indexing: the Steps

Text indexing can be broken up into sequential 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 inverted index**
10. Compress inverted index

3.2.1 Tokenization

Tokenizing means breaking down words into appropriate sequences of characters following a set of rules such as changing upper- to lower-case. Example:

"*Bigcorp's 2007 bi-annual report showed profits rose* 10%"
becomes:
"*bigcorp 2007 annual report showed profits rose*"

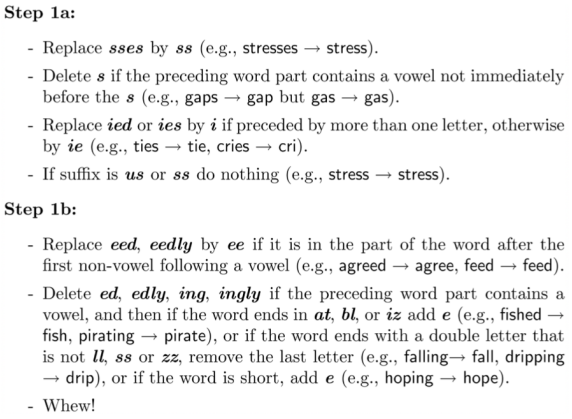
It is too simple for most search applications due to the amount of lost information. Small decisions in tokenizing can have a major impact on queries' effectiveness. The first step is to parse the document to find the elements to tokenize, identify them, and index every alphanumeric characters sequence terminated by a space or a special character.

3.2.2 Stopping

Since **function words** have little meaning on their own, they are treated as **stopwords** (i.e. removed) to reduce index space and improve effectiveness. Although, they can still be important in some combinations such as "*United States of America*". A stopword list can either be created from high-frequency words or be based on a standard one. The best policy is to index all words and make decisions about which ones to use at query time.

3.2.3 Stemming

There exist many morphological variations of words distinguished in **inflectional**, meaning plurals and tenses, and **derivational**, meaning making verbs nouns and more. **Stemmers** reduce these variations to a common **stem** through some techniques such as removing prefixes and suffixes to obtain a significant effectiveness improvement. There are two basic stemming approaches: the **algorithmic approach**, which uses programs to determine related words, and the **dictionary-based approach**, which uses lists of related words. The algorithmic stemmers remove "*s*" ending assuming plural or third person in verbs, but also obtaining many **false negatives**, such as "*supplies*" $\rightarrow$ "*supplie*", and **false positives**, such as "*ups*" $\rightarrow$ "*up*". The dictionary-based stemmers remove endings based on a dictionary obtaining real words instead of stems, having more precision concerning the algorithmic ones, but also being more expensive to run. The **Porter stemmer** is an algorithmic stemmer that removes the longest possible suffix at each step. Since it is sometimes too aggressive



Step 1a:

- Replace ***sses*** by ***ss*** (e.g., stresses → stress).
- Delete ***s*** if the preceding word part contains a vowel not immediately before the ***s*** (e.g., gaps → gap but gas → gas).
- Replace ***ied*** or ***ies*** by ***i*** if preceded by more than one letter, otherwise by ***ie*** (e.g., ties → tie, cries → cri).
- If suffix is ***us*** or ***ss*** do nothing (e.g., stress → stress).

Step 1b:

- Replace ***eed***, ***eedly*** by ***ee*** if it is in the part of the word after the first non-vowel following a vowel (e.g., agreed → agree, feed → feed).
- Delete ***ed***, ***edly***, ***ing***, ***ingly*** if the preceding word part contains a vowel, and then if the word ends in ***at***, ***bl***, or ***iz*** add ***e*** (e.g., fished → fish, pirating → pirate), or if the word ends with a double letter that is not ***ll***, ***ss*** or ***zz***, remove the last letter (e.g., falling→ fall, dripping → drip), or if the word is short, add ***e*** (e.g., hoping → hope).
- Whew!

Figure 2: Porter stemmer example

or too weak, the **Porter2 stemmer** has been designed to address some of its issues and is used with other languages. The **Krovetz stemmer** is a hybrid algorithmic-dictionary-based stemmer:

1. check a word in a dictionary
    - if present, the word is either left alone or replaced with "exception"
    - if not present, the word is checked for suffixes
2. check again dictionary after removal

It has a lower false positive rate, but a higher false negative rate and is computationally expensive.

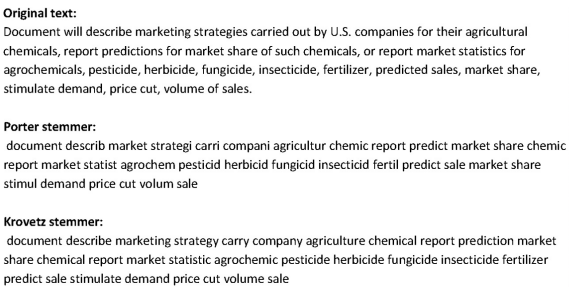


Figure 3: Stemmers comparison example

3.2.4 POS Tagger

Part-Of-Speech (**POS**) tagging is the process of marking up a word as corresponding to a particular part of speech based on its definition and context. POS

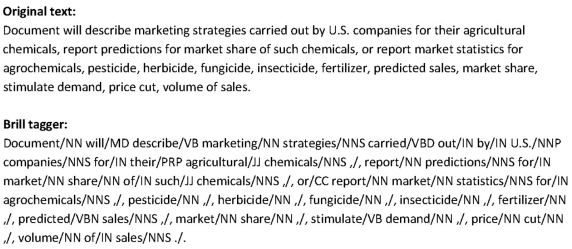


Figure 4: POS tagging example

taggers use statistical models of text to predict syntactic tags of words. This is the most accurate approach to identify the role of a word, but also the most expensive. Since it is too slow for collections, phrases are defined as sequences of **n-grams**.

3.2.5 Build Inverted Index

The data structures used for **inverted indexes** are either dictionaries, since they have a modest size and fast access, or postings since they have huge sizes. The main difficulty is building a huge index with limited memory. Inverted indexes can also support ranking algorithms and proximity matches, as shown in the following images:

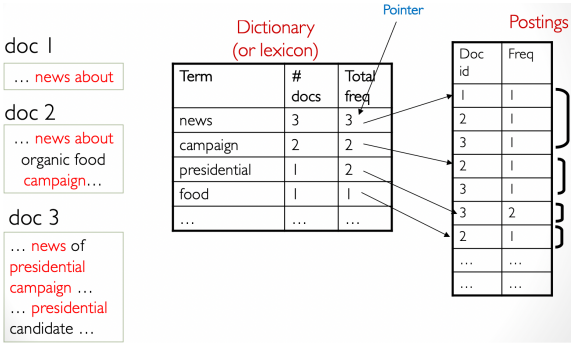


Figure 5: Inverted index example

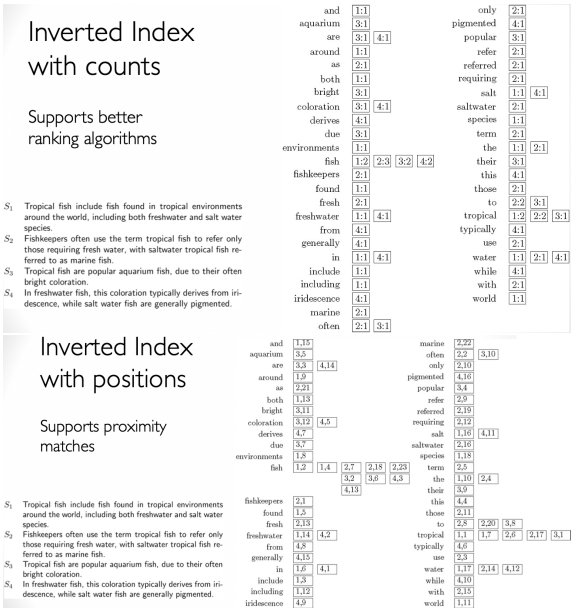


Figure 6: Better inverted indexes

to construct an inverted index:

1. Collect local tuples: $(termID,\ docID,\ freq)$
2. Sort local tuples to make runs
3. pair-wise merge runs
4. Output inverted file

During **distributed processing**, large numbers of inexpensive servers are used

rather than larger and more expensive machines, since they are cheaper, can be upgraded easily, and have fault tolerance.

3.3 Scoring Documents

A general algorithm for **scoring** documents can be defined as:

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

- Final score adjustment: $f_d(d)$ and $f_q(q)$ are precomputed
- Weighted aggregation h: maintain a score accumulator for each d
- For each query term t_i:
  1. Fetch the inverted list $\{(d_1,f_1),\ldots,(d_n,f_n)\}$
  2. For each entry (d_j,f_j), compute $g(t_i,d_j,q)$ and update the score accumulator for h
- Adjust the score to compute f_a and sort

4 Retrieval Models

Retrieval processes are based on **retrieval models** that match a query with a document. They offer a formalization of the concept of relevance, giving a computational definition for it. State of the art models are based on ranking functions relying on "bag of words" representation, terms frequency, documents frequency, and documents length. There exist at least two classes of retrieval models: the **set-based models**, where

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

and the **similarity-based models**, where

$$f(q,d) = similarity(q,d) = [0,\infty]$$

4.1 Boolean Model

The **Boolean model** is the most important set-based retrieval model. It has two possible outcomes for query processing: $true$ and $false$, which give an exact-match retrieval. Queries are usually specified using Boolean operators and sequences of queries are driven by the number of retrieved documents. Let's see an example for news about "*Lincoln*":

1. president AND lincoln
2. president AND lincoln $ANDNOT$ (automobile OR car)
3. president AND lincoln AND biography $ANDNOT$ (automobile OR car)
4. ...

The Boolean retrieval gives back predictable and easy to explain results, can incorporate many features, but its effectiveness depends entirely on the user and it may be difficult to think and write complex queries.

4.2 Ranked Retrieval

Ranked retrieval is better since documents are presented according to their match with the query:

- Query: $q = q_1, \ldots, q_m$ where $q_i \in V$
- Document: $d = d_1, \ldots, d_n$ where $d_i \in V$
- **Ranking function**: $f(q, d) \in \mathbb{R}$

Relevant documents should be ranked on top of non-relevant ones. The key challenge is to estimate the likelihood that a document d is relevant to a query q. There exist several ranked retrieval models:

- **Similarity-based models**: $f(d, q) = similarity(q, d) = [0, \infty]$
- **Probabilistic models**: $f(d, q) = p(R = 1|d, q)$, where $R \in \{0, 1\}$
- **Probabilistic inference model**: $f(q, d) = p(d \rightarrow q)$
- **Axiomatic model**: $f(d, q)$ must satisfy a set of constraints

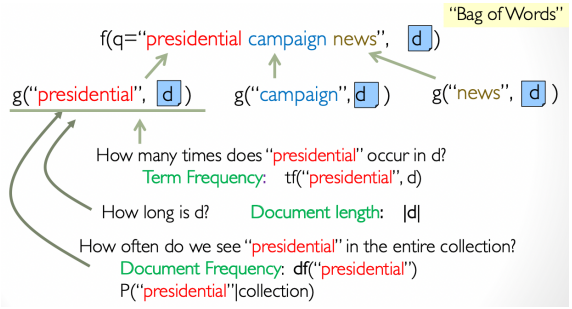


Figure 7: Common ideas of ranked retrieval models

4.3 Vector Space Model

The **Vector Space Model** (**VSM**) is a similarity-based model that represents a document or a query by a **term vector**, with each of them defining one dimension. A single term identifies a basic concept, while N terms define an **N-dimensional space**:

Query vector: $q = (x_1, \ldots, x_N)$, where $x_i \in \mathbb{R}$ is a query term weight
Document vector: $d = (y_1, \ldots, y_N)$, where $y_j \in \mathbb{R}$ is a document term weight

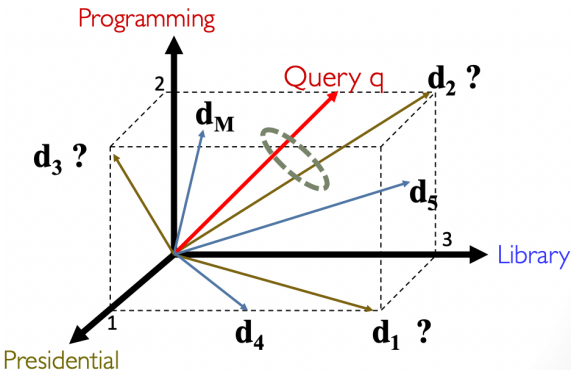


Figure 8: Vector Space Model representation

Basic concepts are assumed to be orthogonal, meaning that they are independent from one another. In the simplest VSM, the **similarity** Sim is obtained through the dot product of vectors q and d:

$$Sim(q,d) =< q,d >= \sum_{i=1}^{N} x_i \cdot y_i,$$

where x_i, $y_i \in 0,1$, since either a word is present or not in a document. Despite this, it has two problems: matching the same word more times deserves more credit and matching a word could be more important than matching another one.

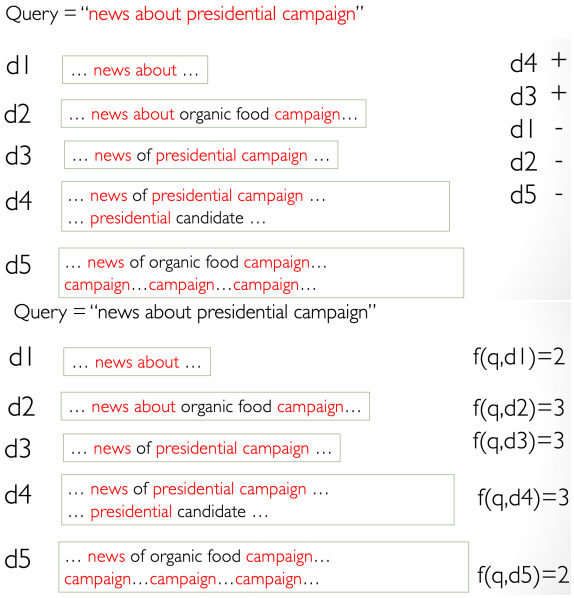


Figure 9: Ideal vs VSM ranking

to solve the first problem, the **term frequency weighting** must be introduced, defining as x_i the count of the word W_i in the query, and as y_i the count of the word w_i in the document.

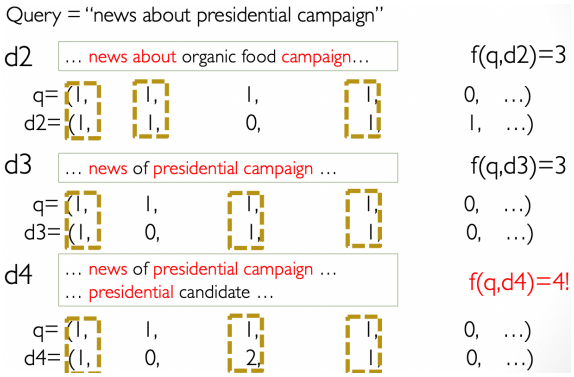


Figure 10: Term frequency vector representation

To solve the second problem, the **Inverse Document Frequency** (**IDF**), defining y_i as:

$$y_i = c(W_i, d) \cdot IDF(W_i)$$

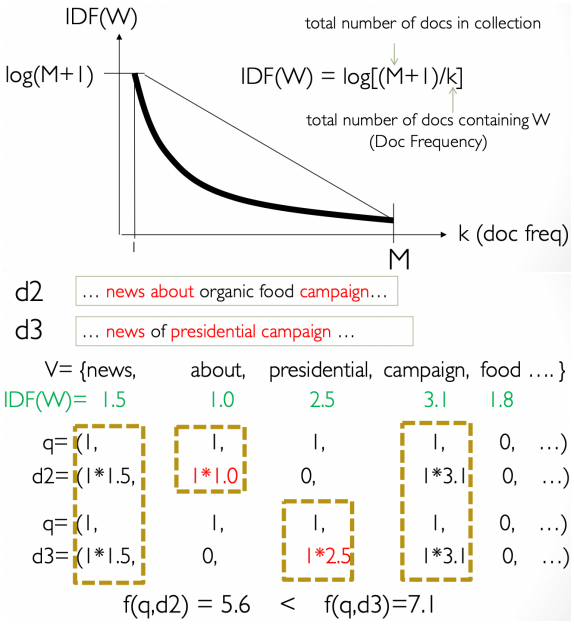


Figure 11: VSM with TF-IDF weighting

It has now a better weighting, but still wrong. to fix the results, the **TF Transformation** is needed, since we need a robust and effective transformation with an upper bound that avoids the dominance by one single term over all others. Finally, we obtain the following formula:

$$f(q,d) = \sum_{w \in q \bigcap d} c(w,q) \cdot \frac{(k+1 \cdot c(w,d))}{c(w,d)+k} \cdot log \frac{M+1}{df(w)}$$

A document can be long because it has more words or more content. Since long documents have a better chance to match any query, they must be penalized by **document length normalization** using a **pivoted length normalizer**, which is an average length used as a pivot.

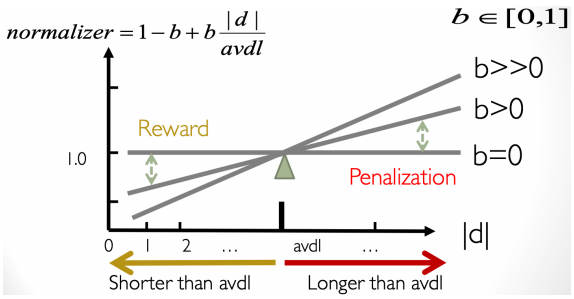


Figure 12: Pivoted length normalization representation

4.4 Probabilistic Models

Probabilistic models introduce the concept of **query likelihood**: if a user likes a document d, how likely would the user enter a query q for retrieving d? We define the **query likelihood ranking function** as follows:

$$f(d,q) = p(R = I|d,q) \approx p(q|d,R = I) \approx \frac{count(q,d,R = 1)}{count(q,d)}$$

to compute the probability that a user formulates the right query and the probability of text in general, we must use a **Language Model** (**LM**), which is a probability distribution over word sequences that quantifies the uncertainties in the natural language.
//Example:

- $p($"*Today is Wednesday*"$) \approx 0.001$
- $p($"*Today Wednesday is*"$) \approx 0.0000000000001$
- $p($"*The eigenvalue is positive*"$) \approx 0.00001$

It can perform different types of functions:

- **Speech recognition**: if we see "*John*" and "*feels*", how likely will we see "*happy*" as opposed to "*habit*" as the next word?
- **Text categorization**: if we see "*baseball*" three times and "*game*" once in an article, how likely is it about "*sports*"?
- **Information retrieval**: if a user is interested in sports, how likely would he use "*baseball*" in a query?

It can also be regarded as a probabilistic mechanism for generating text.

4.4.1 Unigram Language Model

The simplest LM is the **Unigram Language Model**, which generates each word independently through **word distribution**:

$$p(w_1, w_2, \ldots, w_N) = p(w_1) \cdot p(w_2) \cdot \ldots \cdot p(w_N)$$

//Example:

- $p($"*today is Wednesday*"$) = p($"*today*"$) \cdot p($"*is*"$) \cdot p($" *Wednesday*"$)$

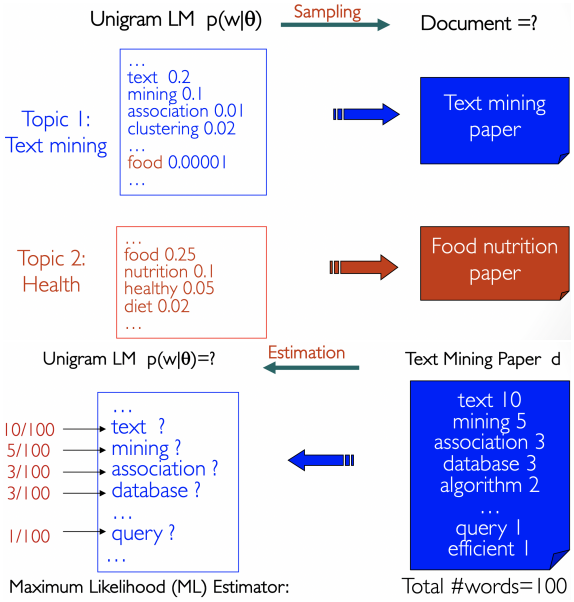


Figure 13: Unigram text generation and estimation

LMs are also used for **association analysis**, which checks what words are semantically related to a chosen one: the topic LM is **normalized** using a **background LM** B such as the general background English text:

$$\textbf{Normalized Topic LM}: p(w|"word")/p(w|B)$$

4.5 Query Generation by Sampling Words

If a user is thinking of a document, how likely would he pose a certain query?

$$p(q = \text{word\_1 word\_2} | d = \text{document}) = p(\text{word\_1}|d) \cdot p(\text{word\_2}|d) = \frac{c(word\_1, d)}{|d|} \cdot \frac{c(word\_2, d)}{|d|}$$

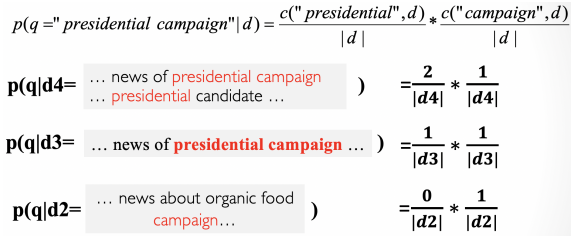


Figure 14: Query likelihood example

4.6 Smoothing

Smoothing is used to estimate the value of $p(w|d)$.

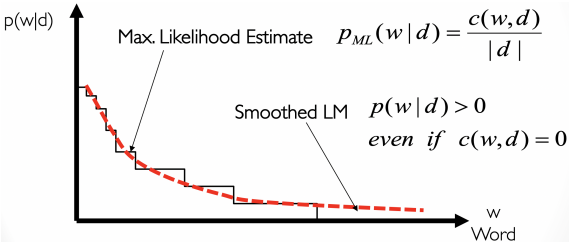


Figure 15: Smoothing graph

We discount the maximum likelihood estimation pML to save some probability to unseen words. Let the probability of an unseen word be proportional to its probability given by a reference LM:

$$p(w|d) = \{ p_{seen}(w|d) \text{ if } w \text{ is seen in } d \quad \textbf{Discounted ML} \{ \alpha_d p(w|C) \text{ otherwise} \quad \textbf{Collection language model}$$

Rewriting the ranking function with smoothing appears like this:

$$\sum_{w \in V,\ c(w,d)>0} c(w,d) \cdot log \frac{p_{seen}(w|d)}{\alpha_d(w|C)} + |q| \cdot log\alpha_d + \sum_{w \in V} c(w,q) \cdot logp(w|C)$$

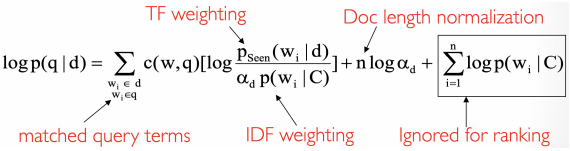


Figure 16: Rewriting representation

MISSING LAST 10 OF SMOOTHING.

5 Relevance Feedback

There exist three major forms of **feedback**:

- **Relevance**: user indicate expressively documents that are relevant to the query and make explicit judgments on the system results
- **Pseudo**: system assumes the top retrieved documents are relevant
- **Implicit**: system assumes relevance by monitoring what the user does

In **relevance feedback**, users make explicit relevance judgments on the system results.

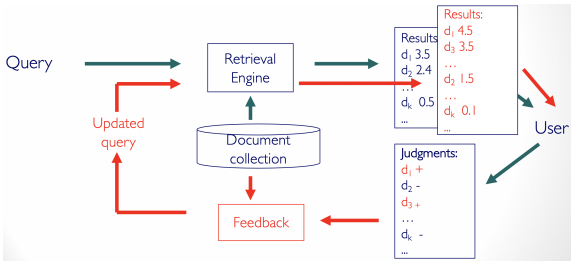


Figure 17: Relevance feedback representation

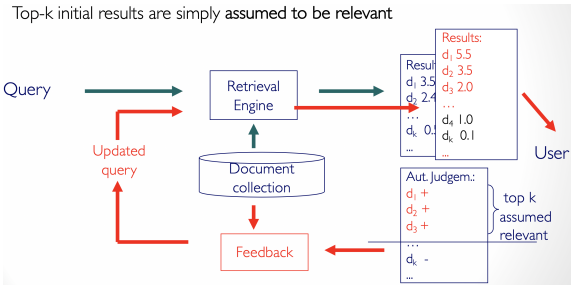


Figure 18: Pseudo feedback representation

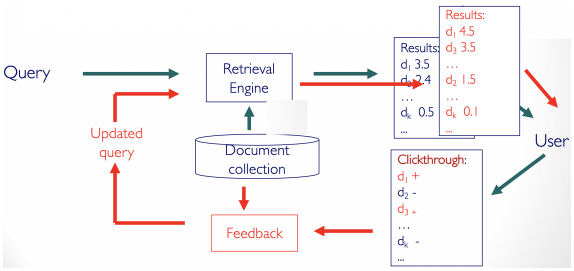


Figure 19: Implicit feedback representation

5.1 Relevance Feedback in Vector Space Model

A system learns from relevance feedback by **query expansion**, meaning adding new weighted terms, and **query modification**, meaning adjusting weights of old terms. In the vector space model, relevance feedback moves queries closer to relevant documents and maybe away from non-relevant ones.

5.2 Rocchio Feedback

The **Rocchio feedback** is based on the **optimal query**, which maximizes the difference between the relevant documents' average vector and the average vector representing the non-relevant ones. The resulting query is longer and expanded since frequent terms of relevant documents will be added to the modified query and the non-frequent ones in the non-relevant document will be removed. //Example: if *Doc3* is relevant, then the query vector moves in its direction, changing the relevance estimation of all the other documents.

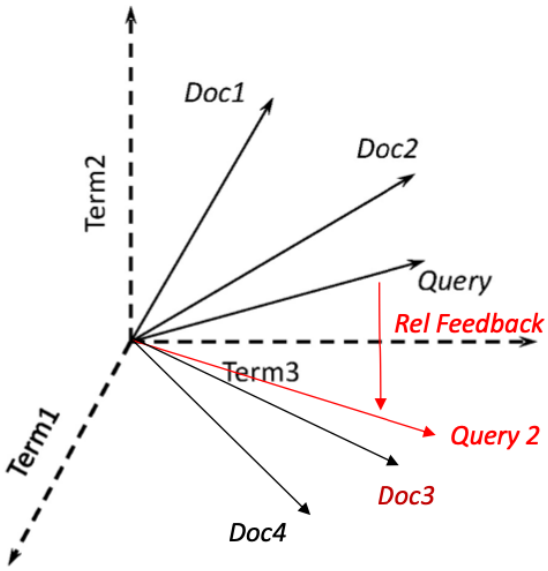


Figure 20: Rocchio algorithm illustration

The new query q'_j is a linear combination of the original query q_j, the average relevant documents, and the average non-relevant documents, governed by 3 parameters, $\alpha = 8$, $\beta = 16$, and $\gamma = 4$:

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

6 System Evaluation

Evaluation is concerned with assessing if the system carries out its tasks properly and it is the key to building effective and efficient search engines. It measures three different aspects:

- **Efficiency**: if the system carries out tasks with optimal use of its resources
- **Effectiveness**: if the system finds what we are searching for
- **Usability**: how useful is the system for real user tasks

6.1 Cranfield Evaluation Methodology

The **Cranfield evaluation methodology** is the primary approach to evaluate systems' effectiveness. The idea consists of building reusable test collections and defining measures of effectiveness using:

- sample collection of documents
- sample set of queries
- relevance judgments to obtain an ideal ranked list
- measures to quantify how well a system's result matches the ideal list

Obtaining relevance judgments is an expensive time-consuming process.

6.2 Effectiveness Measures for Retrieval Sets

The two main effectiveness measures of IR are **precision** and **recall**. Let's assume that A is a set of relevant documents and B is a set of retrieved documents:

$$\text{Recall} = \frac{|A \cap B|}{|A|}$$
$$\text{Precision} = \frac{|A \cap B|}{|B|}$$

	Relevant	Non-Relevant	
Retrieved	$A \cap B$	$\overline{A} \cap B$	B (=Rs)
Not Retrieved	$A \cap \overline{B}$	$\overline{A} \cap \overline{B}$	$\overline{B}$
	A (=Rd)	$\overline{A}$	

Figure 21: effectiveness measures

There exist two types of **classification errors**:

- False Positive - Type I error
  ratio of non-relevant documents that are retrieved

$$\textbf{Fallout} = \frac{|\bar{A} \cap B|}{|\bar{A}|}$$

- False Negative - Type II error
  ratio of non-retrieved relevant documents

$$\text{False Negative} = 1 - \text{ Recall}$$

The **F1 Score** is the **harmonic mean** of recall and precision, meaning that emphasizes the importance of small values whereas the arithmetic mean is more affected by large outliers:

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

It gives equal weight to P and R. Computing average precision values enables to compare ranking methods:

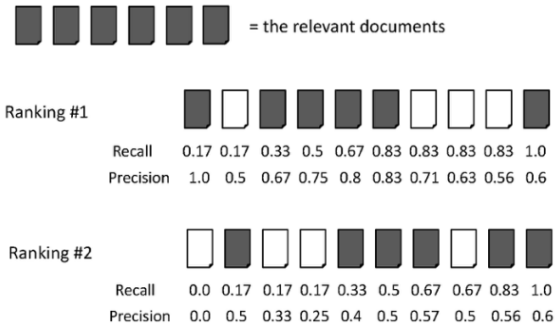


Figure 22: Average precision

Ranking #1: $(1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$
Ranking #2: $(0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$

Mean Average Precision (**MAP**) enables to summarise a system's performances and compare different ones:

1. calculate the average precision for each query
2. summarize rankings from multiple queries by averaging average precision

$$\text{mean average precision} = (0.78 + 0.52)/2 = .650$$

It requires many relevance judgments in the text collection.

6.3 Recall-Precision Graphs

Recall-precision graphs provide better summaries for comparison.

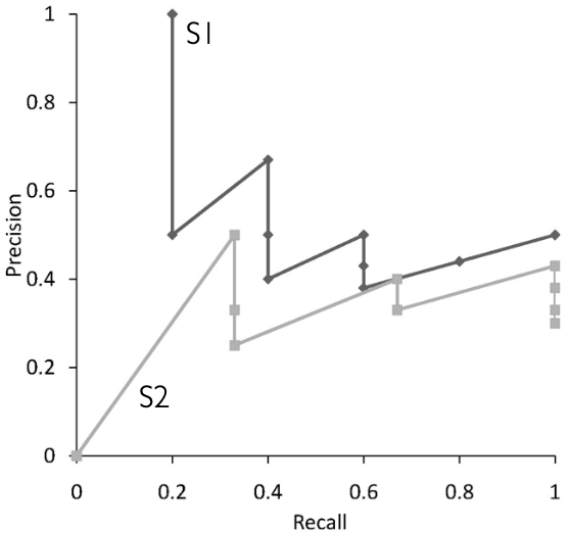


Figure 23: Recall-precision graph for the previous example

Although, in most cases, it is not possible to say which of the two systems is better.

6.4 Interpolation

We need to find performance measures at standard recall levels. Let's assume that S is the set of observed (R, P) points:

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

Interpolation defines precision at any recall level as the maximum precision observed in any recall-precision point at a higher level, producing a step function that makes the comparison easier.

6.5 Focusing on Top Documents

Users tend to look at only the top part of a ranked list. Since in this case recall is not appropriate, we should measure the system's performance in retrieving relevant documents at very high ranks. We can first compute the **reciprocal rank**, meaning the reciprocal of the rank at which the first relevant document is retrieved, then the **Mean Reciprocal Rank** (**RMM**), which is the average of the reciprocal ranks over a set of queries.

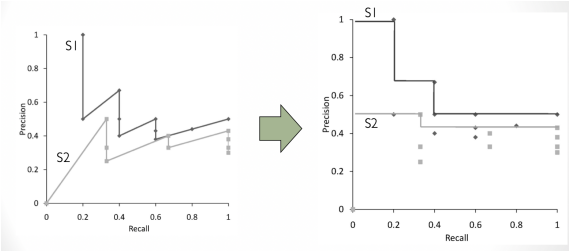


Figure 24: Interpolation effect

6.6 Multi-level Relevance Judgments

If there exist multi-level relevance judgments, we can compute the **Normalized Discounted Cumulative Gain** (**nDCG**):

1. measure the total utility of the top k documents to a user
2. discount the utility of a lowly ranked document
3. normalize to ensure comparability across queries

7 User Evaluation

User evaluation is used to test the quality of IR systems through user interactions. The user carries out searches in an artificial situation with a low volume of queries, due to the cost of user time.

7.1 Operational Evaluation

In **operational evaluation**, users are real users in real search situations and with real tasks on a real collection. It is more expensive and difficult to run since it requires lots of user time and evaluator care not to influence the user environment, but performs better tests in real situations. Operational evaluation needs a very careful design of the few variables under our control:

- system needs to be stable
- need to capture user actions non-intrusively
- need to avoid obstructing users' actions
- we are totally in the hands of users and their moods and personalities

7.2 User-oriented Evaluation

In **user-oriented evaluation**, the whole system is evaluated in a comparative way, such as through two interfaces to the same system, using objective measures, such as the time to search, and also using subjective measures, such as the ease of interaction. Although, the environment is less controlled: users have different knowledge, different abilities, and most importantly different definitions of relevance. Nevertheless, there exist things we can control, such as tasks, the time to search, and the given instructions. This evaluation is still experimental and needs to:

1. Decide aim of the evaluation
2. Formulate experimental hypotheses
3. Define experimental methodology
    - need to gather collections, set of people, **baseline** or **experimental** systems, search tasks, and comparison criteria
4. Carry out the experiment
5. analyze data

7.2.1 Decide Aim of Evaluation

Before starting with the evaluation, we must decide what are the features we intend to evaluate and what is the new feature we intend to achieve, meaning essentially what we want to know about the system. Finally, we must decide the aim of the experiment, such as:

"*evaluating the effect on X of changes in Y*".

7.2.2 Formulate Experimental Hypotheses

The aims of the experiment must be expressed as experimental hypotheses to be tested, such as:

"*the average searcher will use fewer queries to retrieve the same number of relevant pages using X than Y*".

Experimental hypotheses tell us some of the information we need to gather. For example, the hypotheses

"*X returns more relevant documents per query than Y*"

tell us we need to store the number of documents found and the number of relevant documents contained.

7.2.3 Define experimental methodology

We need to gather many elements and have a clear idea of how we are going to use them:

- Collections: what are we searching for? is the data set accessible and big enough?
- People: who does the search? how much experience do they have?
- Systems: different systems, different versions of the same system, or a **baseline** system and an **experimental** system.
- Search tasks: are the tasks **given**, meaning more controlled and appropriate, or **real**, meaning more motivation for users?
  - Decision tasks: "*find the best museum in Rome*"
  - Background knowledge: "*find as much information as possible on mobile phone pricing schemes*"
  - Fact search: "*find Bill Gates email address*"
- Criteria for comparing data
  - Quantitative data: recall, precision, time to search (easier)
  - Qualitative data: usability, preference, learning (difficult)

  - Likert scale: qualitative data can be quantified

Very difficult	Difficult	No opinion	Easy	Very easy

Figure 25: Likert scale

7.2.4 Carry Out Experiment

Using experimental systems, be careful not to be too nice to subjects and give them too much help when searching. Also, it is important not to choose tasks and data that will do well on one system and badly on another one. The solutions are to adopt the proper attitude and to run pilot tests, meaning short experiment versions to debug.

7.2.5 Analyze Data

Proper methods of statistical data analysis need to be based on quantitative data since qualitative data are more difficult to analyze. In quantitative data analysis:

- Recall and precision evaluation?
  - Yes, if relevance is defined
  - No, otherwise
- Search success evaluation?
  - Give each person the same amount of minutes to search - how many tasks?
  - Give unlimited time to search - how long?
- Other measures?
  - Number of documents viewed
  - Number of queries run

In qualitative data analysis:

- Users' system perceptions
  - Think-aloud or semi-structured discussion with results interpretation
  - Questionnaires after each search and one after the whole experiment
- Tests for statistical significance
- Results variations
  - Statistical tests tell us whether the variations are natural
  - Yes, there is no difference in systems
  - No, let's make claims about systems

8 Web Crawling

A **Web crawler** finds, downloads, and caches web pages automatically, providing collections for indexing and searching. The process of a web crawler is the following:

1. The client program connects to a **domain name system** (**DNS**) server
2. The DNS server translates the hostname into an **internet protocol** (**IP**) address
3. The crawler attempts to connect to the server host using a specific port
4. The crawler sends an HTTP request to the webserver to request a page (***GET***)

More specifically, the actions performed by the crawler itself are the following:

1. The crawler starts with a set of **seeds**, which are set of **uniform resource locators** (**URLs**)
2. The seeds are added to a **frontier**, meaning a URL request queue
3. The crawler starts fetching and downloading pages from the request queue
4. The downloaded pages are parsed to find link tags that might contain other useful URLs to fetch
5. The new URLs are added to the frontier
6. Continue until no more URLs are found

The most common crawling strategy is the Breadth-First, but there exist some variations, such as the complete crawling, which crawls everything is found, and the focused crawling, which targets a subset of pages concerning a given query and relying on the fact that pages have links to other ones about the same topic. Sites that are difficult to find are referred to as the **deep Web**, which can be split up into three categories: the **private sites**, with no incoming links or with a required log in the **form results**, which can only be reached by entering some data into a form, and the **scripted pages**, which use a client-side language to generate links. Since crawlers spend a lot of time waiting for responses, they use **threads** to fetch hundreds of pages at once, potentially flooding sites. To avoid this problem, crawlers use some **politeness policies**. **Robots.txt** files are used to control them and have the following structure:

```
User-agent: *
Disallow: /private/
Disallow: /confidential/
Disallow: /other/
Allow: /other/public/
```

```
User-agent: FavoredCrawler
Disallow:

Sitemap: http://mysite.com/sitemap.xml.gz
```

8.1 MapReduce: Computation Pipeline

	Information Retrieval	Information Filtering
Documents	static collection	dynamic collection
Queries	short-lived queries	elaborated profiles
Needs	short term information	long term information

9 Recommender Systems

Recommender systems perform text access in a push mode. They can be distinguished into two types: **filtering systems** and **routing systems**. **Routing** determines the best order of documents to present to users.

9.1 Information Filtering

The following table shows the differences between IR and **Information Filtering** (**IF**): There exist three main types of IF, which differ in the way relevant documents are determined: content-based IF, collaborative IF, and use-based IF. Filtering answers to the question "*will user U like item X?*" in two different ways: with **item similarity**, looking at which items the user like and checking if an item is similar to them, performing content-based filtering, or with **user similarity**, looking at which users like an item and checking if a user is similar to them, performing collaborative filtering. User and item similarities can be combined.

9.2 Content-Based Filtering

Content-based filtering (**CB**) makes decisions for an individual user based on what it learned from his/her behavior, inferring interests from user feedback. Preferences can be based on the similarity between items.

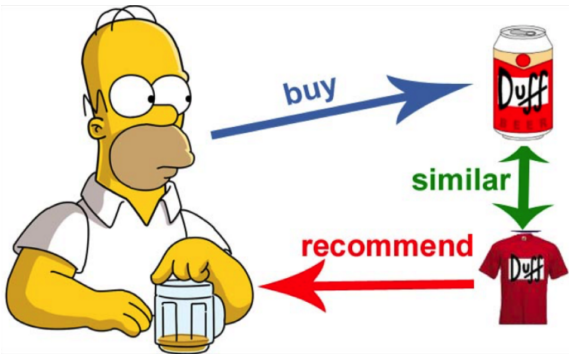


Figure 26: Content-based filtering general idea

There are three basic problems in CB filtering:

1. Filtering decision: binary classifier

2. Initialization: initialize the filter only based on the profile text

3. Learning: limited relevance judgments

9.3 Collaborative Filtering

Collaborative Filtering (**CF**) makes filtering decisions for an individual user based on the judgments of other users, inferring individual interests from similar users. It is based on **user profile comparison**, thus it needs a large number of users.

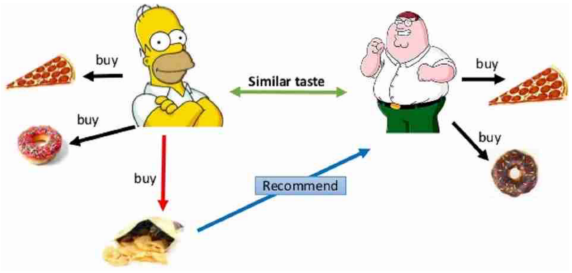


Figure 27: Collaborative filtering general idea

CF uses statistics to find correlations between profiles. There is no content-based matching since only documents identifiers are used. Users with the same interests will have similar preferences and users with similar preferences probably share the same interests.

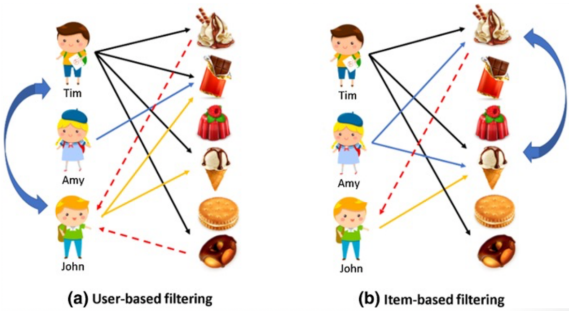


Figure 28: User-based filtering vs. item-based filtering

To improve user similarity measures, missing values are set to average ratings and **Inverse User Frequency** (**IUF**), similar to IDF, is used. There are still some problems to deal with:

- Size of data: we need to use very large sparse matrices to deal with large numbers of users and ratings

- Cold start: new documents cannot be recommended without previous use

In hybrid systems, new items are found by querying and browsing.

9.4 Use-based filtering

Use-based filtering identifies documents that are semantically relevant to a profile by inferring relevant documents from user actions. The following table shows the main differences between use-based and other types of filtering:

	Information Retrieval	Information Filtering
Documents	static collection	dynamic collection
Queries	short-lived queries	elaborated profiles
Needs	short term information	long term information

The system tracks the user's actions such as bookmarking, saving, printing, and the time spent on reading and scrolling. It then develops a profile and uses it to retrieve new pages. It still has a couple of problems: profiles can be very messy since user interaction styles changes concerning many factors, and can be intrusive since it is like someone is watching over the user's shoulders.

	IR	CBF	CF	UBF
Collection	Fixed (usually)	Dynamic (quickly, replacement)	Dynamic (slowly, usually extended)	Dynamic (slowly, usually extended)
Search statement	Query (short-lived, representation)	Profile (long-lived, representation)	Profile (long-lived, object ids)	Profile (long-lived, representation)
Information need change	Quick	Slow	Slow	Quick?
Matching	Content	Content	Id's	Content
Output	Ranked list of items	Items above threshold	Unseen similar items	Unseen similar items

Figure 29: Summary of filtering types

10 Text Data Analysis

Text retrieval and text analytics are combined to analyze big text data.

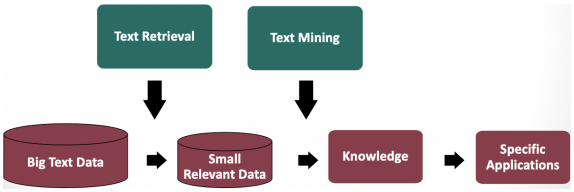


Figure 30: Text data analysis representation

10.1 Text Mining

Text mining turns data into high-quality information and actionable knowledge minimizing human effort and supplying knowledge for decision making. Text retrieval can be a preprocessor for text mining. The following is the landscape of text mining and analytics:

1. Natural language processing and text representation
2. World association mining and analysis
    - mining knowledge about language
3. Topic mining and analysis
    - mining content of text data
4. Opinion mining and sentiment analysis
    - mining knowledge about the observer
5. Text-based prediction (**predictive analytics**)
    - infer other real-world variables

10.2 Natural Language Content Analysis

Natural Language Processing (NLP) is at the foundation of text mining. Computers are far from being able to understand natural language. Since deep NLP requires common sense knowledge and inferences and only works for limited domains, shallow NLP based on statistical methods can be performed on a large scale and is more broadly applicable. The following scheme shows the actions performed during NLP:

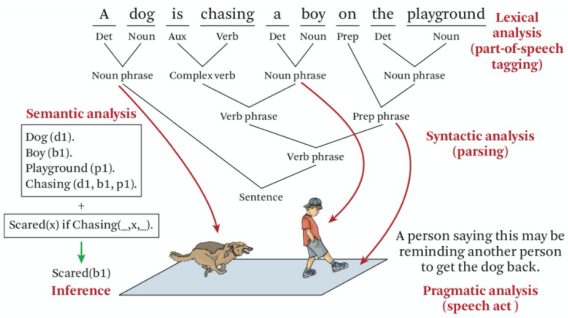


Figure 31: Basic concepts in NLP

In particular, **semantics analysis** is composed of two steps: entity/relation extraction and word sense and sentiment analysis.

11 Opinion Mining and Sentiment Analysis

Opinion mining consists of analyzing what others think about something. It is hard to retrieve documents that report opinions and not only facts since they can be expressed in different ways, in different media, from different users, and can be hidden in vast amounts of data. Moreover, opinions are not verifiable and can be false: **fake news** consists of false or misleading information which damages the reputation of someone or something. Fake news generates **misled opinions**.
Opinion analysis is the computational treatment of the subjectivity in text and tries to design algorithms to detect and analyze opinions in text.

11.1 Opinion Analysis

An **opinion** is a subjective statement describing what an **opinion holder** believes or thinks about an **opinion target** and cannot be proved right or wrong. It can also be defined with the following **quintuple**:

$$o = (e_i,\ a_{ij},\ oo_{ijkl},\ h_k,\ t_l), \text{ where}$$

- e_i: the name of an entity
- a_{ij}: an aspect of that entity
- oo_{ijkl}: the orientation of the opinion about that aspect
- h_k: the opinion holder
- t_l: the time when the opinion is expressed by the holder

In particular, the orientation oo_{ijkl} can either be positive, negative, or neutral and can be expressed with different intensities.

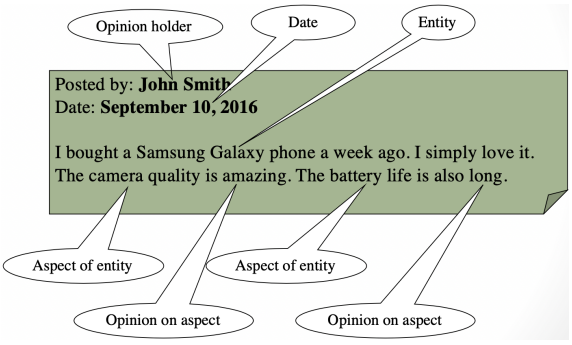


Figure 32: Example of an opinion

11.2 Opinion Analysis Levels

Opinion analysis can be divided into the following three different **levels** of analysis:

A **Document level**

- Assumptions:
  - a document focuses on a single object
  - a document contains an opinion from a single opinion holder
- Classifies a whole document as expressing a positive or negative sentiment
  - Task 1: identify if the document is opinionated
  - Task 2: determine the polarity of the document
- Example: "*I bought a Samsung Galaxy phone a week ago. I simply love it. The camera quality is amazing*".

B **Sentence level**

- Assumption: a sentence contains only one opinion
- Determines whether a sentence expresses a positive, negative, or neutral opinion
  - Task 1: identify if the sentence is opinionated
  - Task 2: determine the polarity of the sentence
- Example: "*This is a beautiful shirt*".

C **Aspect level**

- Performs finer-grained analysis and determines the opinion expressed towards different aspects
  - Task 1: identify and extract the entity aspects
  - Task 2: determine the polarity of the opinions of each aspect
  - Task 3: group aspects synonyms

11.3 Sentiment Analysis Approaches

There exist three classes of **sentiment analysis approaches**: unsupervised, supervised, and semi-supervised.

11.3.1 Unsupervised Approaches

In **unsupervised approaches**, also known as lexicon-based approaches, no training data is required and rely on external lexical resources that associate a polarity score to each term, defining the content sentiment.
The **sentiment lexicons** are lists of words and expressions used to express opinions. There are three ways to build lexicons: manual lexicons, dictionary-based

lexicons, and corpus-based lexicons, and there exist three different types of lexicons: plain lists, meaning lists of positives and negatives, scores, depending on the intensity of the sentiment, and real numbers, where a word has a value based on the sentiment. They have some advantages, but also some disadvantages:

- Advantages:
  - fairly accurate independently of the medium
  - no need for training corpus
  - easily extended to new domains with additional words
  - easy to rationalize prediction output and to explain them
- Disadvantages:
  - costly to implement
  - in-domain ML models underperform if compared to well-trained ones
  - sensitive to affective dictionary coverage

11.3.2 Supervised Approaches

Supervised approaches use classical ML algorithms to extract characteristic patterns for each category building a predictive model. These approaches mostly focus on better modeling the documents.

11.3.3 Semi-Supervised Approach

Semi-supervised approaches perform the following steps:

1. Get manually annotated documents from a specific domain
2. Train any standard classifier using bag-of-words as features
3. Apply trained classifier to test corpus or application

In these approaches, the features to focus on must be carefully specified. It has one important advantage and some disadvantages:

- Advantage:
  - Tend to attain good predictive accuracy
- Disadvantages"
  - Need for a very specific training corpus:
    - automated extraction
    - crowdsourcing the annotation process
  - Domain sensitivity

11.4 Opinion Retrieval

Opinion retrieval consists of retrieving documents that express an opinion about a topic. The following table compares IR with OR:

	Information Retrieval	Opinion Retrieval
Search for	factual topics	opinions and opinionated topics
Rank according to	topical relevance score	relevance to topic and content of opinion

Opinion retrieval is composed of retrieval and opinion analysis and performs the following steps:

1. Rank documents based on their topical relevance
2. Use off-the-shelf retrieval systems and weighting models
3. Re-rank results by applying heuristics for detecting opinions

The following formula computes a document's relevance score:

$$score(Q, D) = (1 - a) \cdot relevance\_score(Q, D+) + a \cdot opinion\_score(Q, D)$$

12 Text Clustering

Text clustering groups similar objects together depending on their natural structure. An important issue of clustering consists of determining what is similar: users define the **clustering bias** for assessing similarity. Objects that do not appear in clusters are called **outliers**. In text retrieval, clustering is used on terms, sentences, websites, and so on to let us get a sense of the overall content of a collection. It is used to link similar text objects and to generate a hierarchy with sub-clusters.

12.1 Clustering Techniques

Clustering is unsupervised, but there exist some predefined criteria on which objects should go together. It can be divided into two main categories: **similarity-based clustering**, which differs in agglomerative clustering and divisive clustering, and **model-based clustering**, where the data is generated from a mixture of component models.

12.1.1 Similarity-based Clustering

Similarity-based clustering provides a clustering bias to measure similarity between two text objects, finding an optimal partitioning to maximize intra-group similarity and minimize inter-group similarity. There are two strategies for obtaining optimal clustering: progressively construct a hierarchy of clusters either in a bottom-up way, gradually grouping objects into larger clusters, or in a top-down way, gradually partition the data into smaller clusters; or start with an initial tentative clustering and iteratively improve it. Two representative similarity-based methods are the Hierarchical Agglomerative Clustering (HAC) and the k-means, and both of them require a symmetric and normalized similarity function. The **Hierarchical Agglomerative Clustering** (**HAC**) gradually groups objects in a bottom-up fashion to form a hierarchy until some stopping criterion is met.

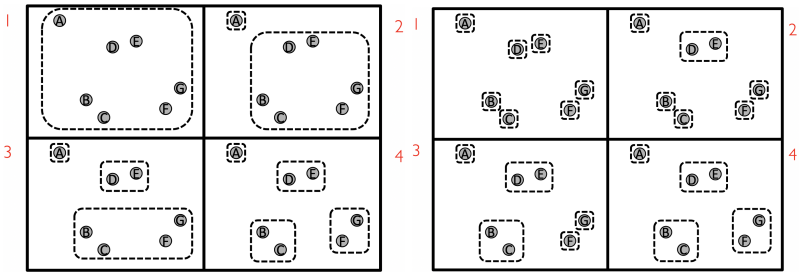


Figure 33: Hierarchical Divisive [left] and Hierarchical Agglomerative [right] clusterings

12.2 Clustering Strategies

- **Single-link**
  - creates "loose" clusters and is sensitive to outliers
  - $cost(C_i, C_j) = min\{dist(X_i, X_j) \ | X_i \in C_i, X_j \in C_j\}$
- **Complete-link**
  - creates "tight" clusters and is sensitive to outliers
  - $cost(C_i, C_j) = max\{dist(X_i, X_j) \ | X_i \in C_i, X_j \in C_j\}$
- **Average-link**
  - clusters "in between" the above two and is insensitive to outliers
  - $cost(C_i, C_j) = \dfrac{\sum_{X_i \in C_i,\ X_j \in C_j} dist(X_i, X_j)}{|C_i| \cdot |C_j|}$
- **Average group linkage**: $cost(C_i, C_j) = dist(\mu_{C_i}, \mu_{C_j})$

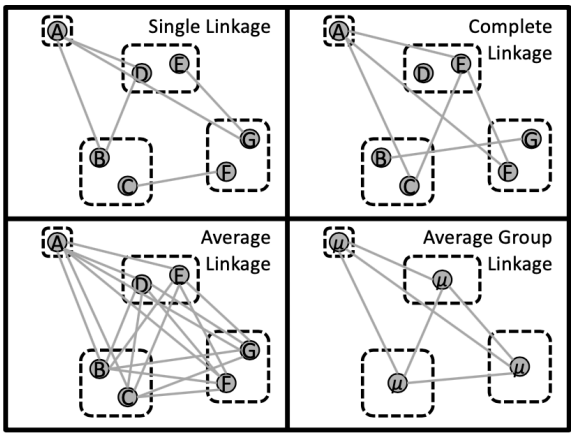


Figure 34: Clustering strategies

12.3 Cluster Representative

To represent clusters, we use **clustroids**, which are those "points that are closest to all other points", and **centroids**, which are the average of all points in the cluster. "Closest" means that the point has the smallest maximum distance to the others, the smallest average distance to the others, or the smallest sum of squares of distances to the others. There exist different approaches to determine the nearness of clusters:

1. Treat clustroids as centroids and measure the distance between them

2. Compute the intercluster distances, meaning the minimum of the distances between any two points, one from each cluster

3. Analyze the cohesion of clusters and merge clusters whose union is most cohesive

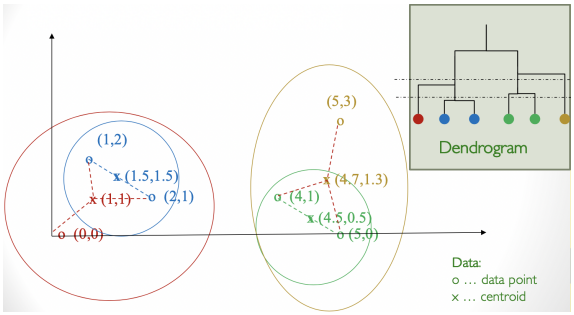


Figure 35: Example

12.4 K-Means clustering

K-means clustering picks a value for k, which is the number of clusters and initializes those clusters by picking one point for each of them as far away as possible from the previous points. Then, it populates clusters as follows:

1. Place each point in the cluster with the nearest centroid

2. Update the locations of centroids of the clusters

3. Reassign all points to their closest centroid

4. Repeat steps 2 and 3 until convergence, meaning until points don't move between clusters and centroids stabilize

to choose the best value for k, we must try different values looking at the change in the average distance to clustroids as they increase: the average falls rapidly until the right k is chosen, then changes a little.

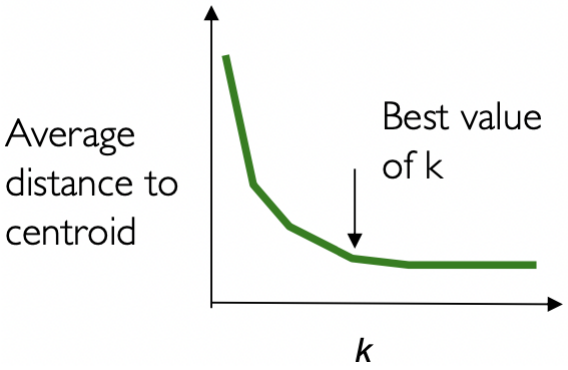


Figure 36: Best $k-$value selection

There exist many variations of $k-$means clustering: in the k**-nearest neighbor clustering**, k is the number of elements in the cluster.

12.5 Evaluation of Clustering

Evaluation of clustering analyzes three measures based on objects and purposes:

1. **Coherence**: how similar objects in a cluster are
2. **Separation**: how far objects in different clusters are
3. **Utility**: how useful the discovered clusters are

The most important determinant of effectiveness is a good similarity measure. There exist two types of evaluation: direct evaluation and indirect evaluation. Direct evaluation measures the **closeness** of a result to the ideal one and has a bias imposed by the human assessors. Its procedure is the following:

1. Given a test set, humans create an ideal clustering result known as the "*gold standard*"
2. Use a system to produce clusters from that test set
3. Quantify the similarity between the system-generated clusters and the gold standard ones

Indirect evaluation measures the **usefulness** of a result and has a bias imposed by the intended application. Its procedure is the following:

1. Create a test set to quantify the performance of any system
2. Choose a baseline system to compare with
3. Add a clustering algorithm to the baseline system
4. Compare the performance of the clustering system and the baseline one

13 Text Categorization

In **text categorization**, every object is put in a category, while in **text classification** each object obtains a label to identify its class. Given that, classification and categorization have the same meaning. While classification answers the question: "*what class does this item belong to?*", and is a supervised learning task, clustering answers the question: "*how can I group this set of items?*" and is an unsupervised learning task.

13.1 Classification

Classification is the task of automatically applying labels to items and is useful for search-related tasks such as spam detection and sentiment classification. to classify a measure about something, we use the following procedure:

1. Identify the set of features indicative of the measure
2. Extract the features from that something
3. Combine the shreds of evidence from the features into a hypothesis
4. Classify the item based on the evidence

13.2 Text Categorization

Given a set of predefined categories, possibly forming a hierarchy, and a training set of labeled text objects, we need to classify a text object into one or more of the categories. Categories can be either internal, characterizing a text object, or external, characterizing an entity associated with the text object.

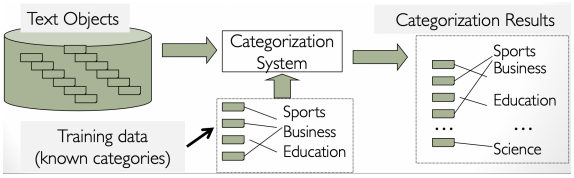


Figure 37: Text categorization example

Text categorization is used to enrich text representation, extending it in multiple levels and facilitating aggregation of text content, and also to infer properties of entities associated with text data.

13.3 Variants of Problem Formulation

- **Binary categorization**: splits into two categories
  - Retrieval: $\{relevant - doc,\ non - relevant - doc\}$

  - Spam filtering: $\{spam,\ non-spam\}$
  - Opinion: $\{positive,\ negative\}$
- $K-$**category categorization**: splits into more categories
  - Topic categorization: $\{sports,\ science,\ travel,\ business,\ \ldots\}$
  - Email routing: $\{folder1,\ folder2,\ folder3,\ \ldots\}$
- **Hierarchy categorization**: splits into categories forming a hierarchy
- **Joint categorization**: performs many categorization tasks in a joint manner

13.4 Text Categorization System

A classification system has three development phases: training, testing, and validation. Such a system must be trained to work properly. Training involves presenting examples of objects with labels to classify, which is done by a combination of two components: a **feature extractor**, which extracts features associated with the label, and a **machine learning algorithm**, to learn how to best associate features with a class.

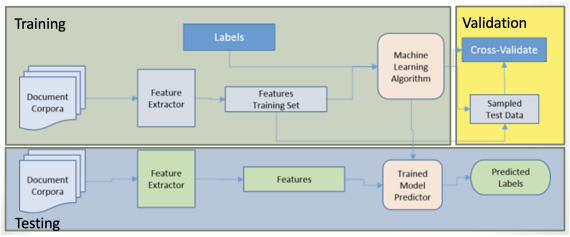


Figure 38: Categorization system structure

14 Multimedia Information Retrieval

14.1 Audio Indexing and Retrieval

Audio is any signal whose frequencies range is in the human audible range and is normally displayed as a waveform with several physical characteristics used to describe it. Most audio characteristics have no semantic content, thus are only used for indexing and retrieving.

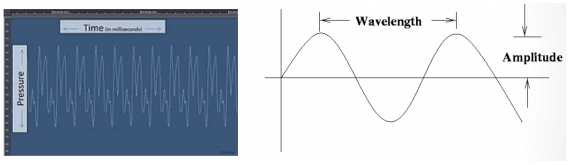


Figure 39: Waveform characteristics example

From the physical characteristics, we can mathematically derive some acoustic features such as pitch and loudness. These features are functions changing over time which must be **sampled** with the following procedure:

1. Compute acoustic features at a certain time interval
2. For each feature, derive the average value μ, the variance σ, and the autocorrelation ρ, meaning the measure of the function smoothness

$$s = (\ (\mu_l, \sigma_l, \rho_l),\ (\mu_{br}, \sigma_{br}, \rho_{br}),\ \ldots,\ (\mu_h, \sigma_h, \rho_h)$$

3. Weight the values of the features by amplitude

The more points we store, the more precise and expensive the sampling is. Audio retrieving has the following problems:

1. Identification of the most useful acoustic features
2. Representation of the acoustic features
3. Matching of acoustic features

The first approaches were based on the Vector Space Model with vectors representing the weighted presence or absence of acoustic features, while nowadays they are based on advanced matching techniques using neural networks and deep learning.
If audio has been indexed as features vectors, then retrieval can be carried out by comparing them with a mathematical model using a query with the covariance matrix R^{-1}:

$$sim(s, q) = ((s - q)^t \cdot R^{-1} \cdot (s - q))$$

14.2 Speech Indexing and Retrieval

Indexing of spoken document is concerned with capturing the semantics of "what has been said" and is performed starting with the following approaches:

1. Controlled vocabulary indexing
2. Ranked retrieval based on associated text
3. Social filtering based on other users' ratings
4. Automatic feature-based speech indexing

The choice of the features to detect depends on the application:

- Identity: speaker identification and segmentation → sound
- Language: language, dialect, accent → sound and word spotting
- Measurable characteristics: emotions, environment → sound
- **Speech content**: phonemes, n-best recognition → **speech**

Speech is richer than text since it holds some information such as the language, the speaker, and the emotions, thus speech recognition is complex.

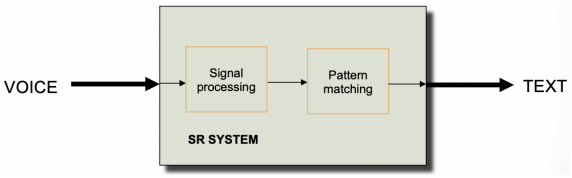


Figure 40: Speech recognition as a black box

The long-term goal of speech recognition systems is to retrieve speaker independent, continuous speech with an almost infinite vocabulary size. Speech recognition is difficult because of many reasons:

- Word boundaries
- Adjacent phonemes influence each other
- The same phoneme cannot be repeated twice without changing its physical representation due to different acoustic conditions
- Ambiguity in phonemic to orthographic transcription
- Out-of-vocabulary words

While speech recognition is almost a solved problem, Spoken Document Retrieval (SDR) is not. The approaches to SDR are the LVSR approach, the subwords units approach, and the word spotting approach.

14.2.1 LVSR Approach

The **LVSR** approach creates transcripts of spoken documents using retrieval methods, transforming SDR into textual IR. Thus, it is a valid approach only with a perfect transcription system. It still has the same problem as general IR.

14.2.2 Subwords Units Approach

The **subwords units** approach analyze either syllable sequences or phoneme sequences. In the first case, syllable sequences are treated as indexing features and the procedure is the following:

- Indexing:
  - Determine a possible set of features from training text
  - Select a subset to use as indexing features
  - Perform feature spotting on spoken documents
- Query: decompose the query and extract the indexing features
- Retrieval: apply $ff(s_j, d_i) \cdot idf(s_j)$ weighting and cosine similarity

On the other side, the procedure is the following:

- Phoneme sequences are treated as indexing features:
  - Select variable-length sequences
  - Perform phonetic recognition on spoken documents
  - Process the output to derive possible sequences
  - Select a subset to use as indexing features
- Query: maps the query into phonemes and extract the indexing features
- Retrieval:
  - Apply $ff(s_j, d_i) \cdot idf(s_j)$ weighting and cosine
  - Matching indexing features in queries and documents

14.2.3 Word Spotting Approach

The **word spotting** approach is used for topic identification, topic spotting, and fast identification of potentially relevant segments of streaming media. A small set of keywords is treated as indexing features, thus it requires knowledge about document content and queries:

- Indexing:
  - Select a set of keywords to spot
  - Perform keyword spotting on the spoken documents

  - Select the documents based on the presence of keywords combinations
- Query: select the combination of keywords
- Retrieval:
  - Apply $ff(s_j, d_i) \cdot idf(s_j)$ weighting and cosine
  - Pattern recognition system

14.3 Image Indexing and Retrieval

There exist three main approaches in **image indexing and retrieval**:

- Keyword based: manual, semi-manual, and automatic
- Visual properties based: fully automatic
- Concept based: mostly manual

14.3.1 Keyword Based Approach

In the **keyword based** approach, images are annotated using keywords. Since images descriptions are textual, we could use standard IR techniques. Keywords are assigned to an image by analyzing the text associated with it. This approach has some problems: manual annotation is expensive and subjective and low level visual properties are almost impossible to index.

14.3.2 Visually Based Approach

Usually referred to as Content Based Image Retrieval (CBIR), the **visually based** approach computes the similarity between query and documents on visual features such as texture and shape which are automatically or semi-automatically detected. Images are represented using **feature vectors**:

$$If_i = (If_{i1}, \ldots, If_{in})$$

and queries are represented with the same set:

$$Qf_i = (Qf_{i1}, \ldots, Qf_{in}).$$

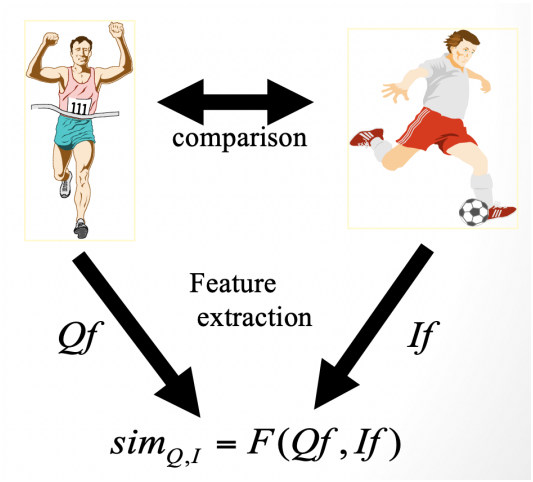


Figure 41: Matching example

This approach uses a form of query-by-example with a specific similarity function which should be a good approximation of human perception and should have properties to speed up computations. Let's see how to handle each feature set:

- **Color based retrieval**
  1. Represent the image as a rectangular pixel raster
  2. Represent each pixel as a quantified color
  3. Count the number of pixels in each color bin
  4. Compute vector similarity (normalized inner product)
- **Texture matching**

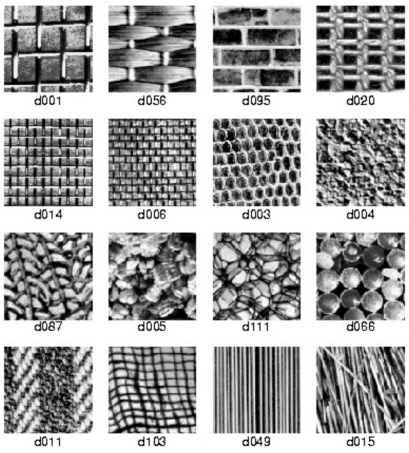


Figure 42: Texture matching example

- **Image segmentation** (color and texture characterize objects and not images)
    1. Segment at color and texture discontinuities
    2. Represent shape and orientation of objects
    3. Represent relative positions of objects
    4. Perform rotation-invariant and scale-invariant matching

The most common example of image segmentation is the **blobworld**, in which complex queries are formulated based on the selection of visual features.

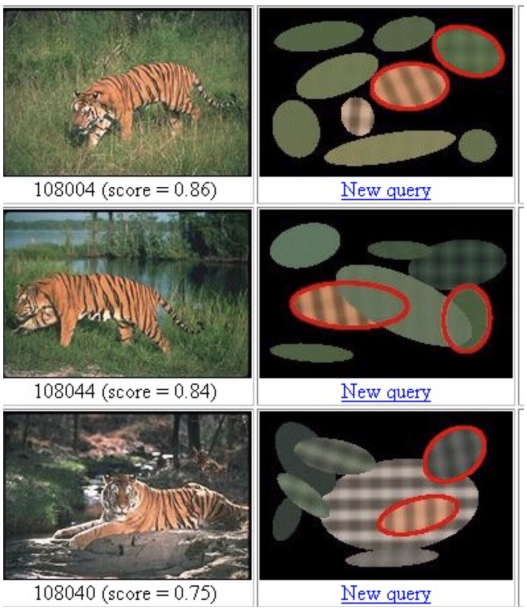


Figure 43: Blobworld segmentation example

14.3.3 Concept Based Approach

In **concept based** approach, the system automatically captures the most important conceptual features of an image. More specifically, the system assigns index terms to a part of the image. This approach has a couple of problems: the automatic concept assignment is imprecise and ambiguous and the manual concept assignment is highly subjective. Moreover, knowledge of the application domain is required.

Figure 44: Concept based approach

14.4 Video Indexing and Retrieval

Video indexing and retrieval can be divided into three processes:

1. Video segmentation
2. Non-sequential browsing
3. Content-based indexing

14.4.1 Video segmentation

Video segmentation splits a video into scenes, then into shots, and finally into frames. A **shot** is a sequence of frames recorded in one camera operation, and a scene is a collection of semantically related shots.

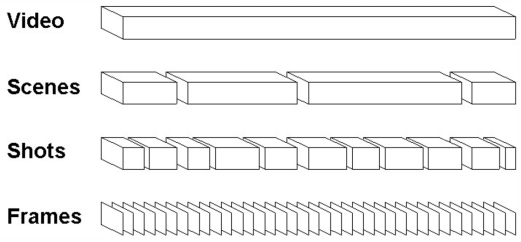


Figure 45: Video segmentation representation

Shot Boundary Detection (**SBD**) takes all the frames extracted from a video and then creates shots depending on those frames. This is a complex technique since it is hard and expensive to determine the many input parameters and the accuracy varies from 20% to 80%

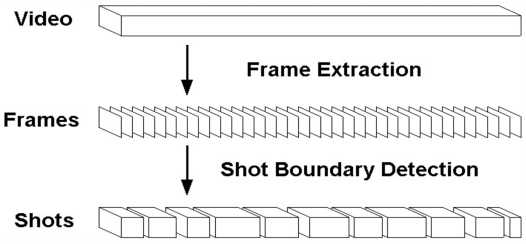


Figure 46: SBD representation

The easiest SBD technique is performed via frame similarity:

1. Take two frames in the input
2. Create the color histograms

3. Concatenate the histograms to form the frame color signature
4. Use cosine similarity measure to get frames similarity

14.4.2 Non-sequential Browsing

Since video browsing, meaning the visual inspection of videos, is time-consuming, videos are structured in terms of shots and scenes to **browse non-sequentially**. A scene hierarchy allows browsing and retrieving at various semantic levels. The video structuring procedure is the following:

1. Segment the video
2. Group shots into scenes
3. Generate a scene hierarchy according to the content

The size and shape of a scene tree are determined by the complexity of the video.

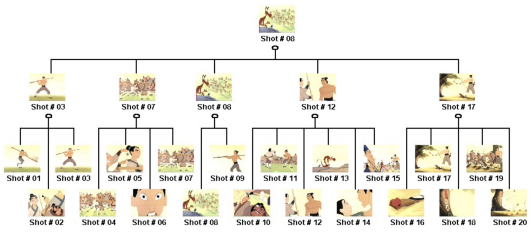


Figure 47: Scene tree and keyframes

14.4.3 Content-based Indexing

Since a video is a combination of lots of features, in **content-based indexing** we must combine different detectors such as text features, visual features, and audio features. The following are some techniques that can be combined for content-based indexing:

- Indexing by color: images containing the same colors
- Indexing by shape: images containing the same shapes
- Indexing by content: images containing the same content but with different shapes and colors

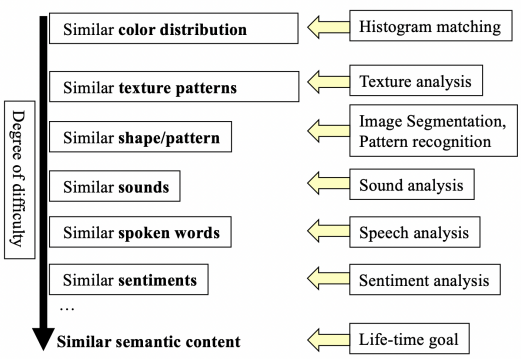


Figure 48: Combination of different features

15 Appendix A - Introduction to Scrappy

15.1 Web Crawling and Scraping

Web crawling consists of automatically downloading a web page's data, extracting the hyperlinks it contains and following them, while **Web scraping** consists of automatically downloading a Web page's content and extracting specific information from it.
With too many requests the server could become overloaded, banning the crawler IP address, but there exist some precaution steps:

- respect robots.txt files, which contain information on what is allowed to crawl and what's not
- limit the number of requests per second
- identify yourself and use proxies

15.2 What is Scrapy

Scrapy is an application framework for crawling websites and extracting structured data, with a large variety of features:

- cookies and session handling
- HTTP compression, authentication, caching
- user-agent spoofing
- robots.txt files
- crawl depth restrictions

15.2.1 Installation Guide

1. `python3 -m venv my\_env`
2. `source my\_env/bin/activate`
3. `pip install Scrapy`

15.2.2 Commands

- `scrapy startproject *project_name*`
  : creates a new project
- `scrapy runspider *spider_name* -o *output_file*`
  : run the spider
- `view(response)`
  : opens the web page in the browser

16 Appendix B - Introduction to Solr

Solr is an open-source search platform and the most used IR library.

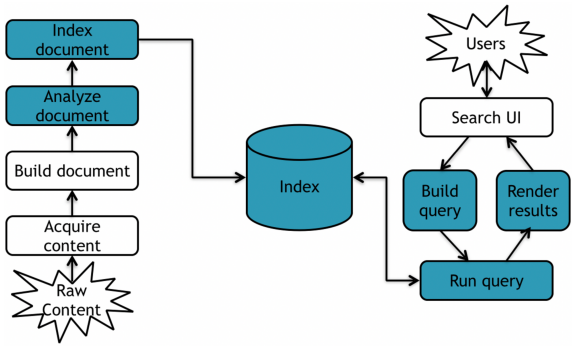


Figure 49: Solr working

16.1 Commands

- `bin/solr start -e cloud`
  : creates a new server
- `bin/post -c <collectionName> <docsCollection>/*`
  : index documents
- `view(response)`
  : opens the web page in the browser