Search Engines

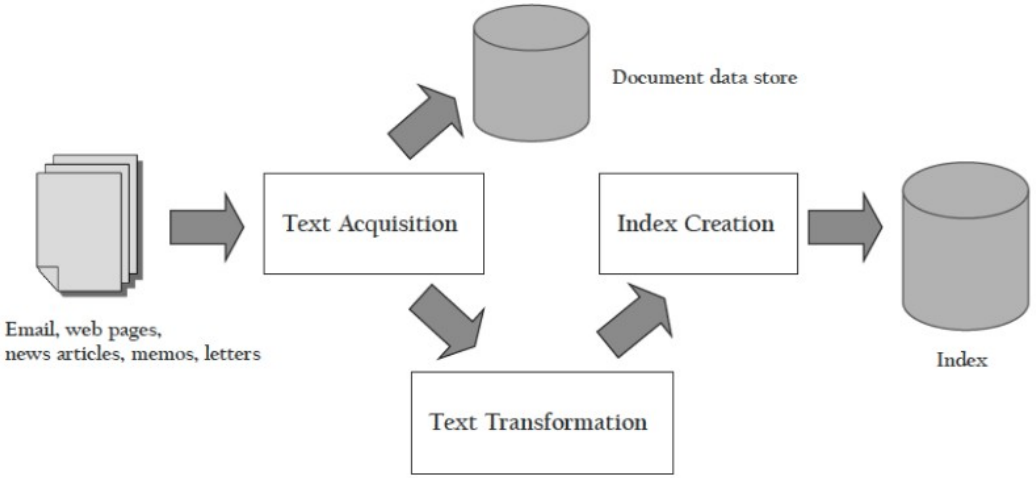
Goal : Find relevant information in a large collection of content

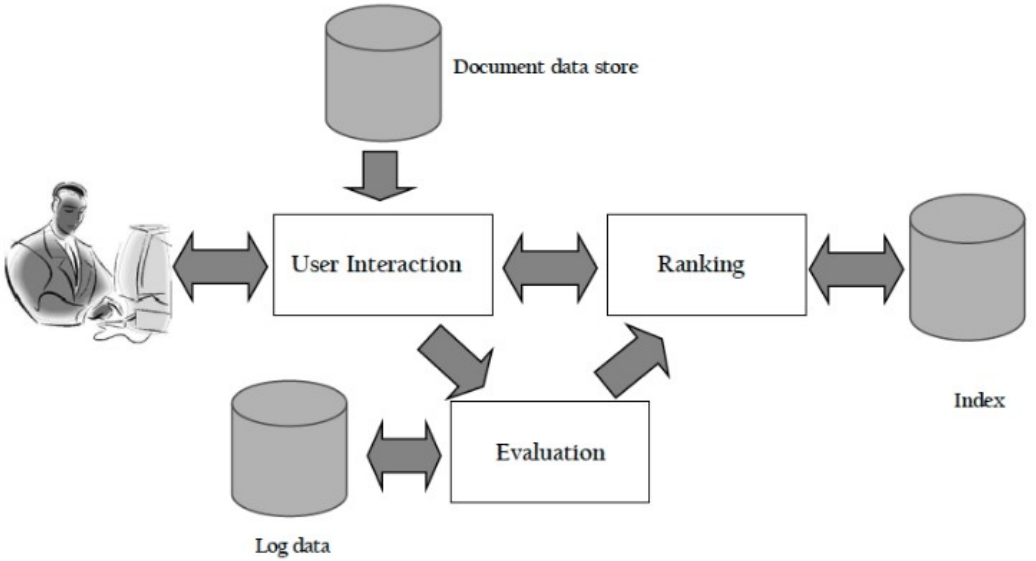
Applications :

- Web search (Google, …)
- File system search on your desktop
- Company level search engine (intranet, e-mails)

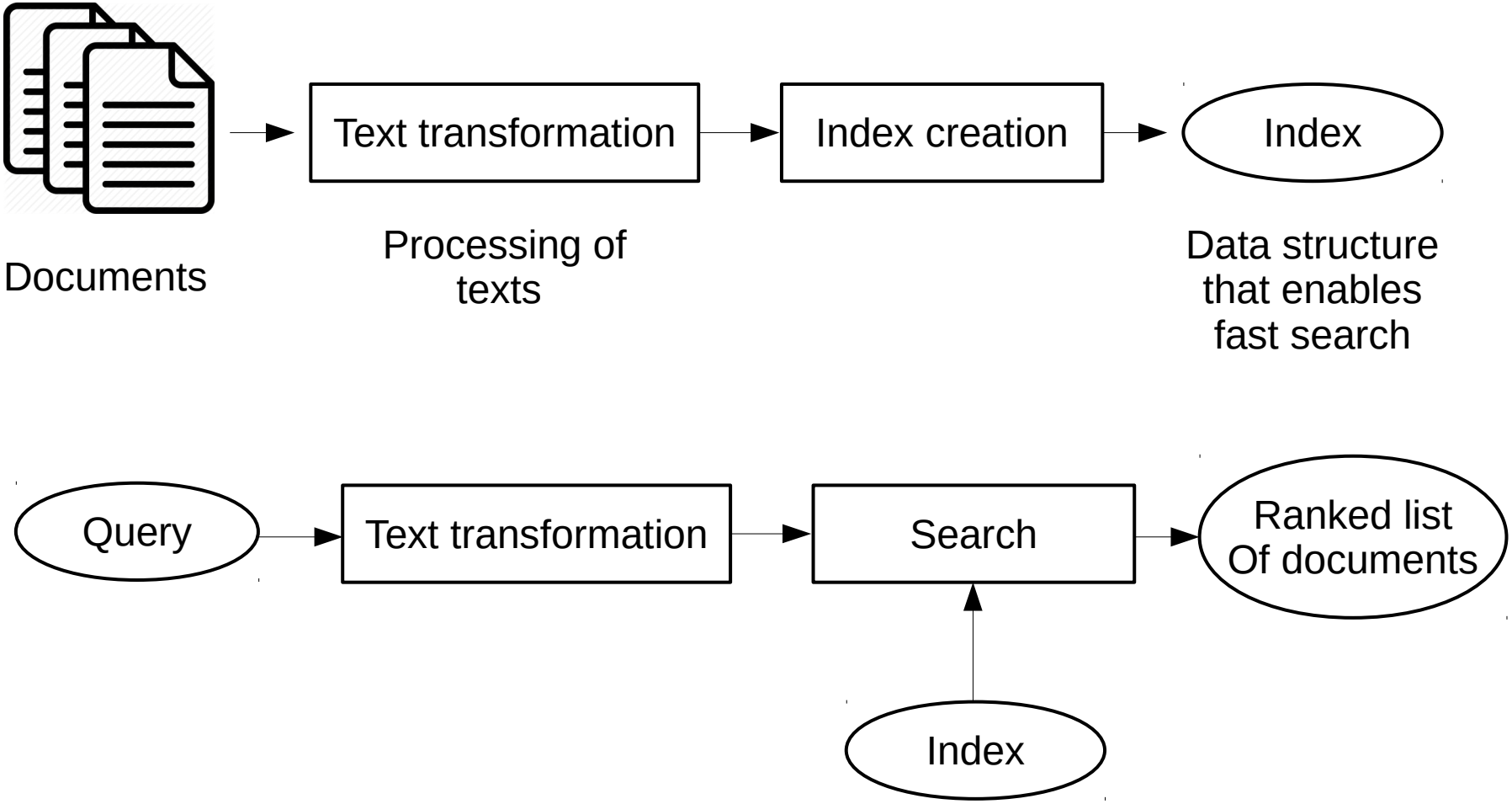
Project : Implement a search engine on text data and improvements or extension to image data.

Search engines





Search Engine Architecture



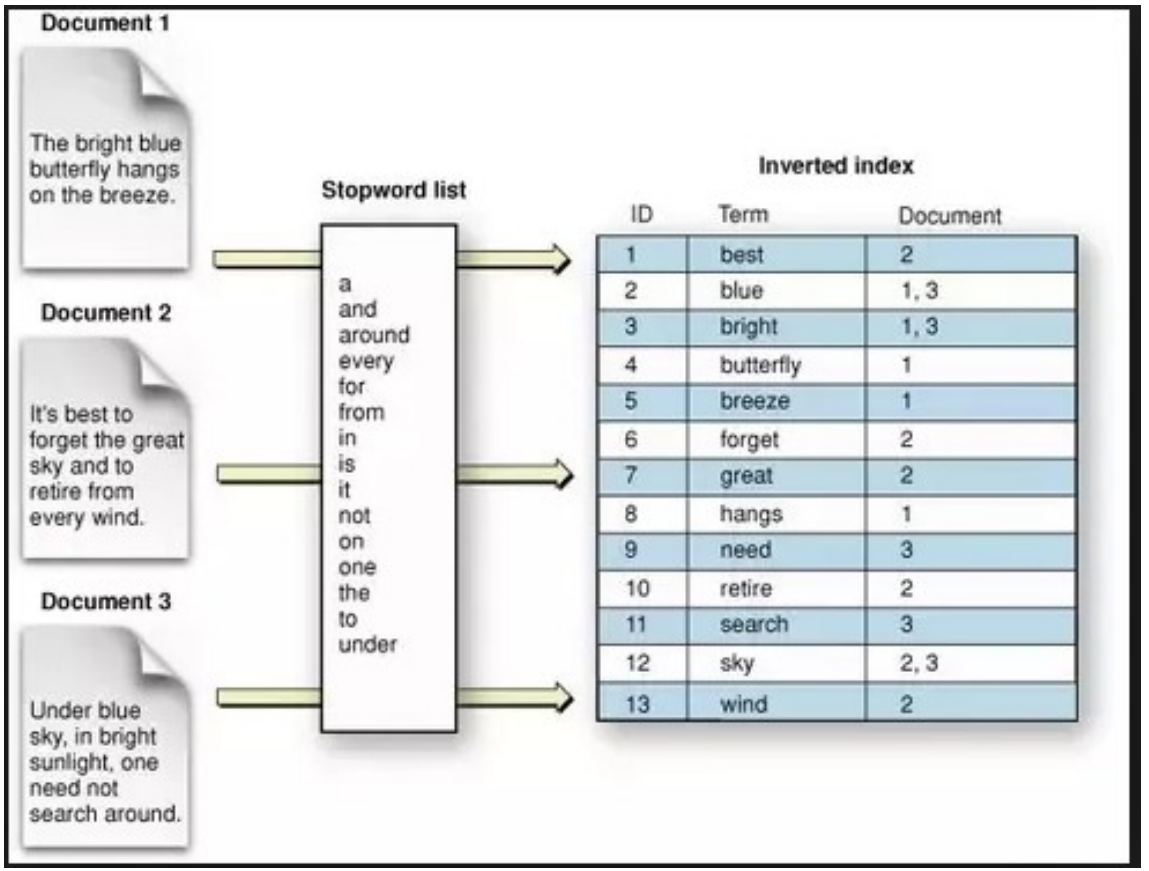
Text transformation

Goals: Clean text and separate each words

- Tokenization :
  - He wasn’t coming. → [‘He’, ‘was’, ‘n’t’, ‘coming’]
- Remove stop words :
  - A, is, the, to, be, … does not help search (non informative)
- Lemmatization :
  - Have, had, has, having → hav (same word corresponds to the same entry in the index)

Try on your own and use / compare with nltk library

Index creation



For each word:
Store the documents that contains it and how many times it appears in the document

Fast search:
Given the words in the query, you can access immediately the relevant documents

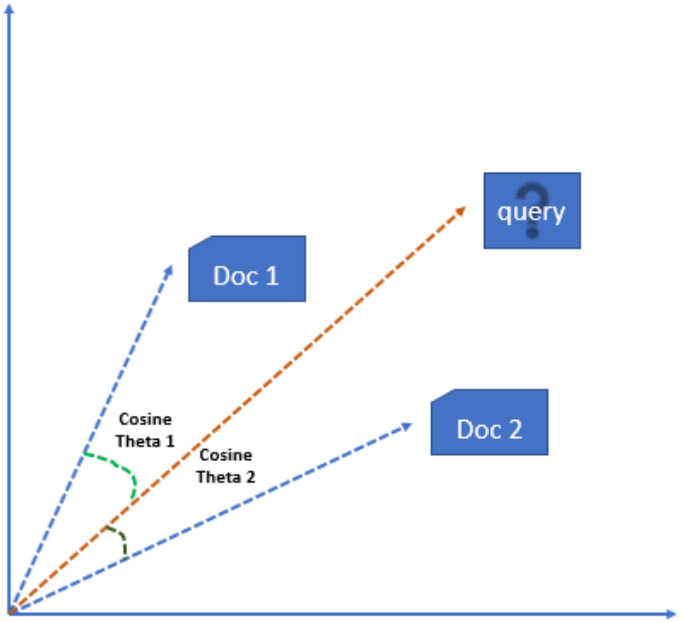
Need a function to save and load the index → Avoid to create it every time.

Search Methods

- **Boolean model:** Returns the documents that satisfies the boolean expression
  - Ex : (football ou rugby) et france

  Problem: No scores and ranking

- **Vector space model:**
  - Use similarity between representations of documents



Search methods

Representation : TF-IDF

Term Frequency – Inverse Document Frequency

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Cosine Similarity :

Angle between the 2 representation vectors

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

A : TF-IDF of the query

B : TF-IDF of the documents

Search methods

- Main question : How to use the index to make fast searches

- Term at a time method (accumulator):

football	Doc1 (1.5)	Doc3 (0.6)
match	Doc2 (3.5)	Doc3(0.1)

Term 1: score[doc1] = 1.5 et score[doc2] = 0.6
Term 2: score[doc1] = 1.5, score[doc2] = 3.5 et score[doc3] = 0.1

Accumulateur pour $\sum A_i . B_i$

Search methods

- Document at a time method (index fusion)

football: doc1(0.5), doc3(0.7), doc6(0.3)
match: doc1(1.2), doc4(1.5)
league: doc2(2.1), doc3(0.6)

étape 1 : doc1(1.7)
étape 2 : doc1(1.7), doc2(2.1)
étape 3 : doc1(1.7), doc2(2.1), doc3(1.3)
étape 4 : doc1(1.7), doc2(2.1), doc3(1.3), doc4(1.5)
étape 5 : doc1(1.7), doc2(2.1), doc3(1.3), doc4(1.5), doc6(0.6)

Sort the final index obtained and return the results

Methods to implement

- Preprocessing of text (try on your own or use nltk)
- Index creation from texts
- Index saving and loading
- 2 search methods shown in the slides
- Tests for the methods created

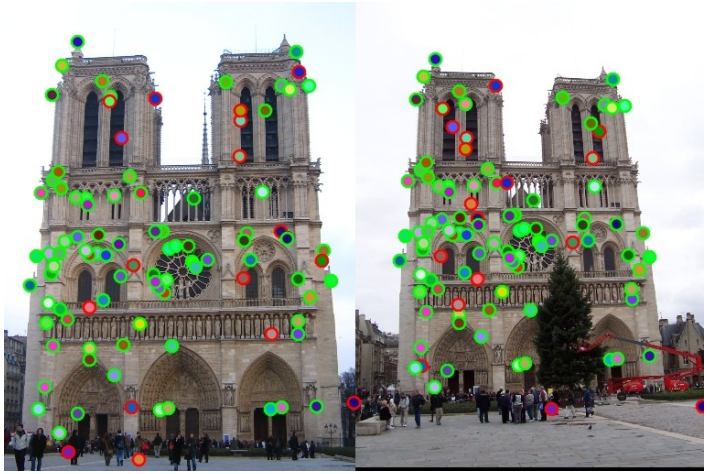
Improvements

- Parrallel index creation using Spark and a sorting and merging algorithm
- Compression of the index using gap compression and variable length codes
- Extension to image search

Image search

1- Local descriptors (SIFT)

- Robust points detection (repeatability)
- Computation of the descriptors (histogram of gradients) which are scale and rotation invariant



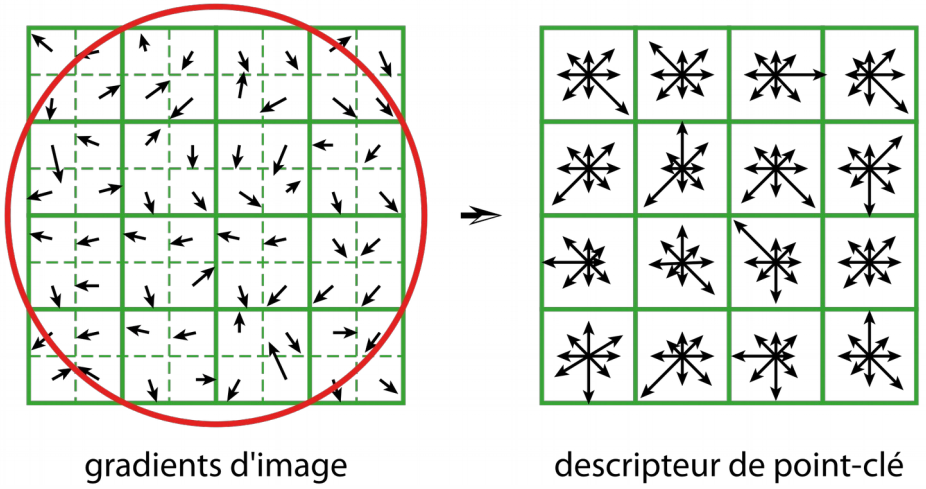
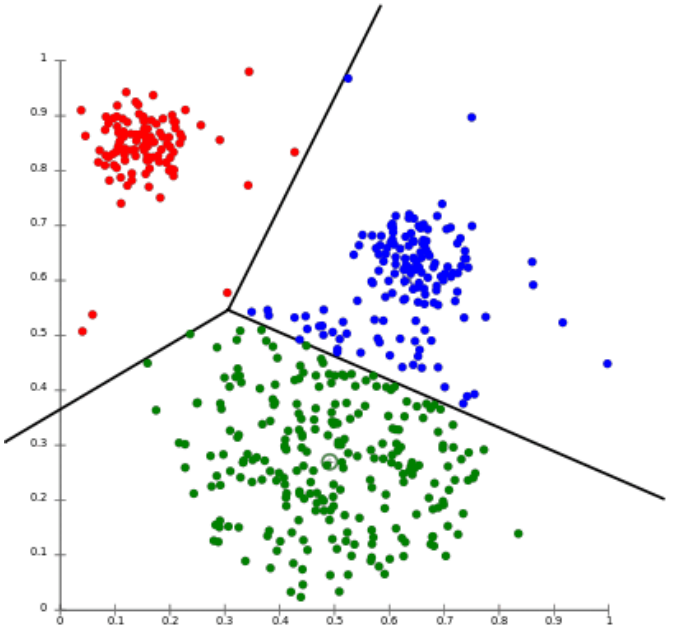


Image search

2- Discretization



- Clustering : group similar representations
- SIFT representation → Cluster (= visual word) (K-means)
- Use the same search algorithm as for text with visual words