# Introduction à la Recherche d'Information

> **i Info**
>
> **Concepts clés du chapitre :**
> - **Recherche d'Information (RI)** : méthodes pour acquérir, organiser, stocker et retrouver l'information pertinente
> - **Système de RI** : retrouve documents pertinents dans grande collection selon requête
> - **Pertinence** : notion subjective dépendant utilisateur et contexte
> - **Problème central** : comparer texte requête vs texte document pour trouver meilleur match
> - **Structuré vs non-structuré** : BDD (champs fixes) vs texte libre (focus RI)
> - **Évaluation** : **Précision** (% pertinents parmi retrouvés) et **Rappel** (% retrouvés parmi pertinents)

## Définitions et Concepts de Base

**Recherche d'Information (RI / IR)** :
- Ensemble des méthodes pour acquisition, organisation, stockage, recherche et sélection d'information pertinente
- Processus automatique répondant à requête utilisateur
- Examine collection de documents
- Retourne liste triée de documents pertinents

**Système de RI** :
- Permet de retrouver information pertinente par rapport à requête
- Travaille sur grande collection de documents

**Document** :
- Texte complet (livre, article)
- Partie de texte (chapitre, paragraphe)
- Page Web (HTML, PHP)

**Requête** :
- Exprime besoin d'information de l'utilisateur
- Formes : description textuelle, mots-clés, exemple de document similaire

**Types de besoins** :
- **Ponctuel** (adhoc) : recherche unique
- **Récurrent** : filtrage, alertes, flux RSS

## Notion de Pertinence

**Document pertinent** : contient l'information recherchée par l'utilisateur

**Notion complexe et subjective** dépendant de :
- **L'utilisateur** : son profil
- **Le contexte** : but, temps, lieu de la recherche

**Évaluation des systèmes RI** : basée sur cette notion de pertinence

## Information Structurée vs Non-Structurée

**Information structurée** :
- Organisée en champs
- Bases de données, requêtes SQL
- Correspondance exacte

**Information non-structurée** :
- **Focus principal de la RI**
- Texte libre
- Peut avoir parties structurées (auteur, date, titre)
- Exemple : "Retrouver les documents au sujet de scandales bancaires en Suisse"

## Problème Central de la RI

**Défi** : comparer texte requête avec texte document et déterminer bon match

**Difficultés** :
- Correspondance exacte des mots ne suffit pas
- Plusieurs façons d'écrire même concept
- Certains documents correspondent mieux que d'autres

**Exemple de problème** :
- Requête : "currency rate scam in Switzerland"
- Document : "Swiss bank scandal"
- Représentent-ils le même sujet ?

## Domaines d'Application

- **Moteurs de recherche Web** : Google
- **Recherche locale** : forums, blogs, sites de news
- **Réseaux sociaux** : Facebook, Twitter, Telegram
- **Sites internes d'entreprises**
- **Bibliothèques numériques**
- **Sites spécialisés** : médecine, droit, brevets
- **Recherche locale** : Mac OS Spotlight, Desktop search
- **Combiné avec** : recommandation (Netflix, Amazon), analyse big data

## Complexité de la Recherche

**Deux aspects** :
- **Utilisateurs** : Comment mieux exprimer besoins en requête ?
- **Système** : Comment mieux correspondre résultats à requête ?

**Focus des techniques RI** : amélioration pertinence en améliorant comportement système

## Évaluation de la Qualité

**Précision** :
- Fraction de documents récupérés correspondant au besoin d'information
- **P(pertinent | retrouvé)**

**Rappel** :
- Fraction de documents pertinents dans collection qui sont récupérés
- **P(retrouvé | pertinent)**

## Architecture Abstraite de RI

**Modèle général** :

```
Requête → Modèle de requête → Fonction de comparaison → Résultats
Documents → Modèle de document ↗
```

**Architecture complète** :

**Hors ligne** :
- Acquisition des documents (ex: web crawling)
- Fonction de représentation → Fichiers d'index

**En ligne** :
- Requête → Fonction de représentation → Représentation de requête
- Documents → Représentation de document
- Fonction de comparaison → Résultats

## Modèles de RI

**Modèles Classiques** :
- **Booléen** : opérateurs AND, OR, NOT
- **Vectoriel** : représentation par vecteurs
- **Probabiliste** : approche probabiliste

**Modèles Structurés** :
- Ensembles flous
- Booléen étendu
- Réseaux bayésiens
- Réseaux inférentiels

**Modèles Algébriques** :
- Vectoriel généraliste
- Latent Semantic Index
- Réseaux de neurones

**Types de Recherche** :
- **Adhoc** : recherche ponctuelle
- **Filtrage** : recherche continue
- **Navigationnel** : navigation guidée (plat, par structure, hypertext)

# Modèle de Recherche Booléen

> **i Info**
>
> **Concepts clés du chapitre :**
> - **Modèle booléen** : requêtes sont expressions booléennes (AND, OR, NOT), résultats binaires (correspond ou non)
> - **Matrice d'incidence** : matrice binaire terme-document (1 si terme présent, 0 sinon)
> - **Index inversé** : pour chaque terme, liste de tous documents le contenant (structure clé de la RI)
> - **docID** : identifiant unique de document
> - **Posting** : occurrence d'un terme (docID)
> - **Postings list** : liste des docIDs pour un terme donné
> - **Index positionnel** : stocke positions des termes dans documents (pour requêtes phrases et proximité)
> - **Complexité fusion** : O(x+y) où x, y = longueurs des listes à fusionner

## Principe du Modèle Booléen

**Définition** :
- Modèle le plus simple pour système de RI
- Requêtes = expressions booléennes
- Exemple : `Brutus AND Caesar AND NOT Calpurnia`
- Moteur renvoie résultats satisfaisant exactement l'expression

**Caractéristiques** :
- Documents vus comme **sacs de mots** (bag of words)
- Requêtes **précises** : un document correspond ou non (binaire)
- **Opérateurs** : AND, OR, NOT
- Outil commercial pendant trois décennies
- **Encore utilisé** : Email, catalogues bibliothèques, Mac OS X Spotlight

## Matrice d'Incidence Terme-Document

**Principe** :
- Indexer documents à l'avance pour éviter parcours linéaire
- **Matrice binaire** : entrée = 1 si terme apparaît dans document, 0 sinon

**Exemple** :

| Terme | Antony&Cleo | Julius | Tempest | Hamlet |
|---|---|---|---|---|

| | | | | |
|---|---|---|---|---|
| Antony | 1 | 1 | 0 | 0 |
| Brutus | 1 | 1 | 0 | 1 |
| Caesar | 1 | 1 | 0 | 1 |
| Calpurnia | 0 | 1 | 0 | 0 |

**Vecteur d'incidence** :
- Vecteur 0/1 pour chaque terme
- Exemple : Brutus = (110100)

**Traitement requête** `Brutus AND Caesar AND NOT Calpurnia` :
1. Prendre vecteurs : Brutus (110100), Caesar (110111), ¬Calpurnia (101111)
2. Opération **ET bit à bit** : `110100 AND 110111 AND 101111 = 100100`
3. **Résultats** : documents 1 et 4

**Problème** :
- Pour N=$10^6$ documents et M=500′000 termes → $5\times10^{11}$ entrées
- Matrice **extrêmement creuse** et inefficace

## Index Inversé

**Structure clé de la RI** :
- Pour chaque terme, stocker **liste de tous documents** le contenant
- Chaque document identifié par **docID**

**Exemple** :

| Terme | DocIDs |
|---|---|
| Brutus | 1 2 4 11 31 45 173 174 |
| Caesar | 1 2 4 5 6 16 57 132 |
| Calpurnia | 2 31 54 101 |

**Composants** :
- **Dictionnaire** : ensemble des termes
- **Posting** : occurrence (docID)
- **Postings list** : liste d'occurrences pour un terme
- **Doc. freq.** : fréquence documentaire

## Construction d'Index Inversé

**Étapes de traitement de texte** :
1. **Tokenisation** : couper en tokens de mots (gérer `John's`, `state-of-the-art`)
2. **Normalisation** : mapper texte et requêtes à même forme (`U.S.A.` = `USA`)
3. **Stemming** (racinisation) : réduire à racine (`kills`, `killed`, `killing`)
4. **Stop words** : omettre mots très courants (`the`, `a`, `to`, `of`)

**Étapes de l'indexeur** :
1. Créer séquence de paires (token modifié, docID)
2. **Trier** par termes puis par docID (étape primordiale)
3. Fusionner entrées multiples d'un terme dans un document
4. Diviser en dictionnaire et postings lists
5. Ajouter information de fréquence documentaire

**Coût de stockage** :
- Termes + Doc. freq + Listes de docIDs + Pointeurs

## Traitement des Requêtes Booléennes

**Requête AND** : `Brutus AND Calpurnia`
1. Localiser Brutus dans dictionnaire → récupérer postings list
2. Localiser Calpurnia dans dictionnaire → récupérer postings list
3. Fusionner (intersection) les deux listes

**Algorithme de fusion** :
- Parcourir deux listes simultanément
- **Complexité** : O(x+y) où x et y = longueurs des listes
- **Crucial** : fonctionne car listes sont triées
- **OR** : même complexité O(x+y)

**Exemple intersection** :

| Terme | DocIDs |
|---|---|
| Brutus | 1 2 4 11 31 45 173 174 |
| Calpurnia | 2 31 54 101 |
| **Résultat** | **2 31** |

## Requêtes de Phrases

**Problème** :
- Répondre à requêtes comme `"Applied Science University"` comme phrase exacte
- Ne suffit plus de stocker `<term : docs>`

**Solution 1 : Index Biword** :
- Indexer chaque **paire de termes consécutifs** comme phrase
- Texte : "Applied Science University"
- → Biwords : `applied science`, `science university`

**Requêtes longues** :
- Décomposer en requête booléenne
- `Western Switzerland Applied Science University`
- → `Western Switzerland AND Switzerland Applied AND Applied Science AND Science University`

**Problèmes** :
- **Faux positifs** possibles
- **Explosion d'index** : dictionnaire plus grand, beaucoup de biwords rares
- **Infaisable** pour plus de deux mots

**Solution 2 : Index Positionnel** :
- Stocker **position(s)** où les tokens apparaissent
- Format : `<term, nb_docs; doc1: pos1, pos2, ...; doc2: pos1, pos2, ...; etc.>`

**Exemple** :
```
<be: 993427;
  1: 7, 18, 33, 72, 86, 231;
  2: 3, 149;
  4: 17, 191, 291, 430, 434;
  5: 363, 367, ...>
```

**Traitement** :
- Algorithme de fusion récursif au niveau document
- Prend en compte **positions relatives**

**Requête "to be"** :
```
<to: 2: 1,17,74,222,551; 4: 8,16,190,429,433; ...>
<be: 1: 17,19; 4: 17,191,291,430,434; ...>
```
- Vérifier positions relatives valides (be suit to immédiatement)

## Requêtes de Proximité

**Principe** :
- Recherche avec **distance maximale** entre termes
- **Notation** : `/k` = "dans une distance max de k mots"
- **Exemple** : `Bank /3 scandal` (Bank et scandal séparés d'au plus 3 mots)
- **Seuls les index positionnels** peuvent traiter ces requêtes

## Taille des Index

**Index positionnel** :
- **2 à 4 fois plus large** qu'index non-positionnel
- **35-50%** du volume du texte original
- Dépend de la taille moyenne des documents :
  - **Page Web** : <1000 termes → 1 occurrence positionnelle
  - **Livre** : 100′000 termes → 100 occurrences positionnelles
- Peut être **compressé**
- **Standard aujourd'hui** malgré la taille (puissance des requêtes)

## Optimisations Combinées

**Pour phrases particulières fréquentes** :
- Exemples : "Twenty One Pilots", "Britney Spears", "The Who"
- **Combiner les deux approches** :
  - Stocker directement comme termes uniques
  - Éviter fusion répétée des postings lists

# Modèle Vectoriel

**i Info**

**Concepts clés du chapitre :**
- **Limitation modèle booléen** : "feast or famine" (0 ou milliers de résultats)
- **Besoin** : recherche avec classement par pertinence
- **Variables importantes** :
  - $\mathrm{tf}_{t,d}$ : nombre d'occurrences du terme $t$ dans document $d$
  - $\mathrm{df}_t$ : nombre de documents contenant terme $t$
  - $N$ : nombre total de documents dans collection
  - $\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$ : pouvoir discriminant d'un terme
  - $w_{t,d} = (1+\log \mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$ : poids TF-IDF
  - $q_i, d_i$ : poids terme $i$ dans requête et document
- **Similarité cosinus** : $\mathrm{Sim}(q,d) = \frac{q \cdot d}{|q|\,|d|}$ (varie de 0 à 1)
- **Norme L2** : $\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$ pour normaliser vecteurs

## Problèmes du Modèle Booléen

**Limitations** :
- **"Feast or famine"** : trop peu (0) ou trop de résultats (milliers)
- **AND** produit trop peu ; **OR** produit trop
- Difficile d'écrire requêtes booléennes
- Impossible de parcourir milliers de résultats
- Bon pour utilisateurs experts, pas pour la majorité

**Besoin** :
- Recherche avec **classement**
- Restituer documents les plus susceptibles d'être utiles
- **Ordonnés par pertinence**

## Coefficient Jaccard

**Formule** : mesure de chevauchement de deux ensembles

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

**Variables** :

- $|A \cap B|$ : nombre d'éléments communs
- $|A \cup B|$ : nombre d'éléments dans union

**Exemple** :
- R: "Cypher query optimization technique"
- D1: "database query optimization technique"
- JACCARD(R, D1) = 3/6 = 0.5

**Problèmes** :
- Ne tient pas compte de la **fréquence des termes** (répétitions ignorées)
- Ne tient pas compte de l'**impact des termes** (spécifiques vs génériques)
- Traite tous les termes également

## Fréquence des Termes (TF)

**TF (Term Frequency)** :
- $\mathrm{tf}_{t,d}$ = nombre de fois où terme $t$ apparaît dans document $d$

**Problème** :
- Fréquence brute inadéquate
- Document avec 10 occurrences n'est pas 10× plus pertinent qu'avec 1 occurrence
- Pertinence n'augmente pas proportionnellement

**Log-frequency** : pondération logarithmique

$$w_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{tf}_{t,d} \text{ si } \mathrm{tf}_{t,d} > 0 \\ 0 \text{ sinon} \end{cases}$$

**Variables** :
- $w_{t,d}$ : poids du terme $t$ dans document $d$ (après pondération log)

**Exemples** : 0→0, 1→1, 2→1.3, 10→2, 1000→4

## Fréquence Documentaire (DF) et IDF

**DF (Document Frequency)** :
- $\mathrm{df}_t$ = nombre de documents contenant terme $t$
- $\mathrm{df}_t$ **élevé** → terme plus **général**
- $\mathrm{df}_t$ **bas** → terme plus **spécifique** (rare)

**IDF (Inverse Document Frequency)** : pouvoir discriminant

$$\mathrm{idf}_t = \log_{10}(N/\mathrm{df}_t)$$

**Variables** :
- $N$ : nombre total de documents dans collection
- $\mathrm{idf}_t$ : plus il est élevé, plus le terme est rare et discriminant

**Exemples** ($N = 1'000'000$) :
- calpurnia ($\mathrm{df} = 1$) : $\mathrm{idf} = 6$ (très rare, très discriminant)
- animal ($\mathrm{df} = 100$) : $\mathrm{idf} = 4$
- sunday ($\mathrm{df} = 1'000$) : $\mathrm{idf} = 3$
- fly ($\mathrm{df} = 10'000$) : $\mathrm{idf} = 2$
- under ($\mathrm{df} = 100'000$) : $\mathrm{idf} = 1$
- the ($\mathrm{df} = 1'000'000$) : $\mathrm{idf} = 0$ (pas discriminant)

**Propriétés** :
- **Une seule valeur d'IDF** par terme dans la collection
- Utilise **log** pour "atténuer" l'effet

## Pondération TF-IDF

**Formule** : meilleur schéma connu pour la RI

$$w_{t,d} = (1 + \log \mathrm{tf}_{t,d}) \times \log_{10}(N/\mathrm{df}_t)$$

**Variables** :
- $w_{t,d}$ : poids du terme $t$ dans document $d$
- Combine **TF** (importance locale) et **IDF** (importance globale)

**Propriétés** :
- Augmente avec nombre d'occurrences dans document (**TF**)
- Augmente avec rareté du terme dans collection (**IDF**)
- Note : "-" dans tf-idf est un trait d'union, pas un signe moins
- Noms alternatifs : tf.idf, tf × idf

**Évolution de la matrice** : binaire → fréquences → poids tf-idf

**Représentation document** :
- Vecteur $\boldsymbol{d}_j = (w_{1j}, w_{2j}, ..., w_{tj}) \in \mathbb{R}^{|V|}$
- $|V|$ : taille du vocabulaire

## Modèle Vectoriel : Principes

**Représentation** :
- **Documents** : vecteurs $|T|$-dimensionnels de poids tf-idf
- **Requêtes** : vecteurs similaires avec différents schémas de pondération
- Vecteurs très **creux** (la plupart des entrées nulles)
- **Modèle sac de mots** : ordre des mots non considéré

**Fonction de comparaison** :
- Classer documents selon **proximité à la requête** dans l'espace vectoriel

## Mesures de Similarité

**Distance euclidienne** : **MAUVAISE IDÉE**
- Grande pour vecteurs de longueurs différentes
- Distribution similaire → grande distance possible

**Angle entre vecteurs** : **BONNE IDÉE**
- Document $d$ et $d'$ (= $d$ + $d$) ont angle = 0° (similitude maximale)
- **Indépendant de la longueur** des vecteurs

**Similarité cosinus** : mesure de l'angle

$$\mathrm{Sim}(q, d) = \cos(q, d) = \frac{\boldsymbol{q} \cdot \boldsymbol{d}}{|\boldsymbol{q}|\ |\boldsymbol{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2} \sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

**Variables** :
- $q_i$ : poids terme $i$ dans requête
- $d_i$ : poids terme $i$ dans document
- $|\boldsymbol{q}|$, $|\boldsymbol{d}|$ : longueurs (normes L2) des vecteurs
- $|V|$ : taille du vocabulaire

**Propriétés** :
- Varie de **0** (aucune similarité) à **1** (identiques)
- **Insensible à la longueur** des documents

## Normalisation de Longueur

**Norme L2** :

$$\|\boldsymbol{x}\|_2 = \sqrt{\sum_i x_i^2}$$

**Fonction** :
- Calcule la **longueur** (magnitude) d'un vecteur
- Racine carrée de la somme des carrés des composantes

**Vecteur normalisé** :
- Diviser chaque composante par sa longueur
- Mappe vecteurs sur **sphère unitaire**
- Documents longs et courts ont poids de même ordre
- Document $d$ et $d'$ (annexé) ont vecteurs identiques après normalisation

**Pour vecteurs normalisés** :
- Similarité = **produit scalaire simple**
- Simplification du calcul

## Schémas de Pondération TF-IDF

**Trois composantes** :

**1. Term Frequency (TF)** :
- **n** (natural) : $\mathrm{tf}_{t,d}$ - fréquence brute
- **l** (logarithm) : $1 + \log \mathrm{tf}_{t,d}$ - pondération log
- **a** (augmented) : $0.5 + \frac{0.5 \cdot \mathrm{tf}_{t,d}}{\max_t \mathrm{tf}_{t,d}}$ - normalisé
- **b** (boolean) : $\{0 \text{ si } \mathrm{tf} > 0; 1 \text{ sinon}\}$ - binaire

**2. Document Frequency (DF)** :
- **n** (no) : 1 - pas d'IDF
- **t** (idf) : $\log(N/\mathrm{df}_t)$ - IDF standard
- **p** (prob idf) : $\max\{0, \log((N - \mathrm{df}_t)/\mathrm{df}_t)\}$ - IDF probabiliste

**3. Normalization** :
- **n** (none) : 1 - pas de normalisation
- **c** (cosine) : $1/\sqrt{w_1^2 + w_2^2 + ... + w_M^2}$ - norme L2
- **u** (pivoted unique) : $1/u$ - normalisation pivotée
- **b** (byte size) : $1/\mathrm{CharLength}^{\alpha}$, $\alpha < 1$ - par taille

**Schéma standard** (notation l-n-c / l-t-c) :
- **Document** : log tf, pas d'idf, normalisation cosinus **(l-n-c)**
- **Requête** : log tf, idf, normalisation cosinus **(l-t-c)**

## Implémentation du Modèle Vectoriel

**Implémentation naïve** : $O(|V| \cdot |D|)$ → **MAUVAIS**
1. Convertir tous documents en vecteurs tf-idf
2. Convertir requête en vecteur
3. Calculer score pour chaque document
4. Trier par score décroissant

**Implémentation pratique** : **utiliser index inversé**
- Documents sans mots-clés de requête n'affectent pas classement
- Identifier uniquement documents avec ≥1 mot-clé

**Étapes** :

**1. Prétraitement** :
- Tokenisation, stop words, stemming

**2. Indexation** : construire index inversé
- Entrée par mot du vocabulaire
- Liste documents avec TF
- Nombre total occurrences pour IDF
- **Complexité** : $O(mn)$ pour $m$ documents de $n$ tokens

**3. Recherche** : documents avec ≥1 mot de requête
- Calcul incrémental de similarité cosinus
- **Table de hachage** : docID → score cumulé

**4. Classement** : trier par similarité décroissante

**Accumulation term-at-a-time** :
- Calculer scores en ajoutant contributions des termes de requête
- Un terme à la fois

## Commentaires sur le Modèle Vectoriel

**Limitations** :
- Manque informations **syntaxiques** (structure phrase, ordre, proximité)
- Manque interprétation **sémantique** (sens des mots)
- Hypothèse **indépendance des termes** (ignore synonymie)
- Manque **contrôle booléen** (exiger présence d'un terme)
- Peut préférer doc avec A fréquent sans B, plutôt que doc avec A et B moins fréquents

**Avantages** :
- Approche **simple et mathématique**
- Prend en compte fréquences **locales** (tf) et **globales** (idf)
- **Correspondance partielle** et résultats classés
- **Fonctionne bien en pratique**
- **Implémentation efficace** pour grandes collections

# Extraction des Termes

**i Info**

**Concepts clés du chapitre :**
- **Pipeline** : Documents → Tokenizeur → Tokens → Modules Linguistiques → Indexeur → Index inversé
- **Token** : séquence de caractères candidat pour entrée d'index
- **Terme** : type de mot normalisé, entrée dans dictionnaire système RI
- **Tokenisation** : découper séquence caractères en tokens de mots
- **Normalisation** : mapper texte et requêtes sous même forme (accents, majuscules, points, traits d'union)
- **Stop words** : termes très courants, peu de valeur sémantique (the, a, to, of, …)
- **Stop list** : liste pour exclure mots courants du dictionnaire
- **Lemmatisation** : supprime fins flexionnelles, renvoie lemme (forme dictionnaire)
- **Stemming** : processus heuristique grossier, coupe extrémités des mots, dépend de la langue
- **Classes de stem** : groupe de mots transformés en même racine par algorithme stemming

## Pipeline d'Indexation

**Flux complet** :
`Documents → Tokenizeur → Tokens → Modules Linguistiques → Tokens modifiés → Indexeur → Index inversé`

**Exemple** :
```
"Friends, Romans, Countrymen"
→ [Friends, Romans, Countrymen]
→ [friend, roman, countryman]
```

## Tokenisation

**Définition** : découper séquence de caractères en **tokens de mots**

**Token** :
- Séquence de caractères
- Candidat pour entrée d'index après traitement supplémentaire

**Question clé** : quels sont les tokens valides à émettre ?

**Exemples de difficultés** :
- `John's` : un ou deux tokens ?
- `state-of-the-art` : un ou plusieurs tokens ?

## Normalisation des Termes

**Objectif** : normaliser mots du texte indexé et mots de requête sous **même forme** pour correspondance

**Terme** :
- Type de mot normalisé
- Entrée dans **dictionnaire du système RI**

**Classes d'équivalence** définies implicitement :
- **Suppression points** : `U.S.A.`, `USA` → `USA`
- **Suppression traits d'union** : `anti-discriminatory`, `antidiscriminatory` → `antidiscriminatory`

**Principe critique** :
- Traitement **cohérent** requête/documents
- Petits détails n'ont pas forcément effet global sur performances

## Formes Courantes de Normalisation

**Accents** :
- Ex français : `résumé` vs `resume`
- Ex allemand : `Tuebingen` vs `Tübingen`
- **Critère** : comment utilisateurs écrivent leurs requêtes ?
- Souvent sans accents → préférable de normaliser à terme désaccentué
- Ex : `Tuebingen`, `Tübingen`, `Tubingen` → `Tubingen`

**Minuscule/Majuscule** :
- **Stratégie commune** : réduire toutes lettres en minuscules
- **Exceptions** : majuscule au milieu de phrase (`General Motors`, `Bush` vs `bush`)
- **Heuristique anglais** :
  - Convertir en minuscules mots en début de phrase
  - Convertir en minuscules titres entièrement en majuscule
  - Mots en majuscule au milieu de phrase laissés en majuscule

## Stop Words (Mots Vides)

**Définition** :
- Termes apparaissant très couramment dans texte
- Ex : `a`, `an`, `and`, `are`, `as`, `at`, `be`, `by`, `for`, `from`
- **Peu de valeur sémantique**
- **30% des postings** pour les 30 premiers mots

**Stop list** : liste pour exclure mots les plus courants du dictionnaire

**Avantages suppression** :
- **Économie espace** d'index
- **Traitement plus rapide** des requêtes

**Quand supprimer ?** :
- **OUI** pour comparaison documents (clustering, classification) :
  - Peu de sémantique
  - Vecteurs plus petits → comparaison plus rapide
- **NON** pour applications de recherche (tendance actuelle) :
  - Requêtes phrases : "King of Denmark"
  - Titres : "The father", "To be or not to be"
  - Relations : "flights to London" vs "flights from London"

## Stemming et Lemmatisation

**Problème** : différentes formes grammaticales d'un mot
- **Flexion** : `organize`, `organizes`, `organizing`
- **Dérivation** : `democracy`, `democratic`, `democratization`

**Objectif** : réduire formes flexionnelles/variantes à forme de base
- `have`, `had`, `having` → `have`
- `car`, `cars`, `car's`, `cars'` → `car`
- `the boy's cars had different colors` → `the boy car have different color`

**Lemmatisation** :
- Supprime uniquement **fins flexionnelles**
- Renvoie **forme de base/dictionnaire** (lemme)
- Utilise vocabulaire et analyse morphologique

**Stemming (Racinisation)** :
- Processus **heuristique grossier**
- Coupe extrémités des mots
- Suppression affixes dérivatifs
- **Dépend de la langue**
- Ex : `automate(s)`, `automatic`, `automation` → `automat`
- Ex : `compressed` et `compression` → `compress`

## Stemming sur Documents et Requêtes

**Principe** :
- Si stemming appliqué lors indexation → doit aussi s'appliquer sur requêtes

**Problème** : peut réduire exactitude des résultats
- Ex : `Python Programming` → `Python program` (perte de précision)

**Alternative : stemming basé sur requêtes** :
- Décision au moment de la requête plutôt que pendant indexation
- Si pas assez de résultats → étendre avec variantes de termes ayant même racine
- Ex : `Python Programming` étendu avec mots ayant racine de `program`, pas réduit
- **Attention** : pas besoin d'appliquer stemming sur documents dans ce cas
- Stocker groupes de mots avec même racine

## Classes de Stem

**Définition** : groupe de mots transformés en même racine par algorithme de stemming

**Exemples** (Porter stemmer sur TREC News) :
```
/bank: banked, banking, bankings, banks
/ocean: oceaneering, oceanic, oceanics, oceanization, oceans
/polic: polical, polically, police, policeable, policed,
        policement, policer, policers, polices, policial,
        policically, policier, policiers, policies, policing,
        policization, policize, policly, policy, policying, policys
```

**Problèmes** :
- Classes **trop grandes** et manquant d'exactitude
- Ex : `police` et `policy` dans même classe (sémantiquement différents)

**Amélioration** : utiliser statistiques de cooccurrence de termes
- Analyse statistique identifie et élimine termes avec faible cooccurrence relative

## Impact sur la Performance

**Stemming sur documents** :
- **Réduit taille de l'index**
- Impacte scores de similarité requête/document

**Suppression stop words** :
- Impacte scores de similarité requête/document
- **Réduit** nombre de documents pertinents pour requêtes phrases (problème !)

# Évaluation des Systèmes de RI

**i Info**

**Concepts clés du chapitre :**
- **Matrice de confusion** : tp (vrais positifs), fp (faux positifs), fn (faux négatifs), tn (vrais négatifs)
- **Précision** : $\text{Précision} = \frac{\text{tp}}{\text{tp} + \text{fp}}$ = P(pertinent|retrouvé)
- **Rappel** : $\text{Rappel} = \frac{\text{tp}}{\text{tp} + \text{fn}}$ = P(retrouvé|pertinent)
- **F-mesure** : $F_\beta = (1 + \beta^2)\frac{(\text{P} \cdot \text{R})}{\beta^2 \cdot \text{P} + \text{R}}$ où $F_1$ est moyenne harmonique de P et R
- **Average Precision (AP)** : $\text{AP} = \frac{\sum_k (\text{P}(k) \cdot \text{rel}(k))}{\text{nb docs pertinents}}$ où rel(k)=1 si doc rang k pertinent
- **MAP** : $\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AP}(q)}{Q}$ = moyenne des AP sur Q requêtes
- **P@K** : précision aux K premiers documents (K=5, 10, 20 typiquement)
- **R-Précision** : $\frac{\text{nb docs pertinents dans top-R}}{R}$ où R = nb total docs pertinents pour requête
- **Courbe Rappel-Précision interpolée** : $p_{\text{interp}}(r) = \max_{r' \geq r} p(r')$ avec points standard 0.0 à 1.0
- **Collection de tests** : documents + requêtes + jugements de pertinence (benchmark)

## Pourquoi Évaluer ?

**Nombreuses alternatives de conception** :
- Utiliser listes de stop words ?
- Appliquer stemming ?
- Utiliser pondération idf ?
- Quel schéma de pondération ?

**Choix** :
- Dépend de l'**application** et des **données**
- Comment savoir quelles techniques sont efficaces ?

## Dimensions d'Efficacité

**Performance système** :
- **Vitesse de recherche**
- **Vitesse d'indexation** (documents/heure)
- **Indexation incrémentale** (ex: +10K articles/jour)
- **Évolutivité** (5M → 20M documents)

**Qualité de recherche** :
- **Satisfaction** des utilisateurs
- **Pertinence** des résultats
- Réponses rapides mais inutiles → utilisateur mécontent

## Collection de Tests

**Composants nécessaires** pour mesure standard :
1. **Collection de documents de référence**
2. **Suite de requêtes de référence**
3. **Jugements de pertinence** pour chaque paire (requête, document)
   - Souvent **binaires** (Pertinent vs Non Pertinent)

**Exemples de collections publiques** :
- **CACM** : titres/abstracts Communications ACM 1958-1979
- **AP** : dépêches Associated Press 1988-1990 (TREC)
- **GOV2** : pages Web domaine .gov 2004 (TREC topics 701-850)

**Utilisation** : comparer systèmes A et B avec requêtes, documents et jugements **standardisés**

## Précision et Rappel

**Définitions fondamentales** :

$$\text{Précision} = \text{P}(\text{pertinent}|\text{retrouvé}) = \frac{\text{tp}}{\text{tp} + \text{fp}}$$

$$\text{Rappel} = \text{P}(\text{retrouvé}|\text{pertinent}) = \frac{\text{tp}}{\text{tp} + \text{fn}}$$

**Matrice de confusion** :

| | Pertinent | Non Pertinent |
|---|---|---|
| Retrouvé | tp | fp |
| Non Retrouvé | fn | tn |

**Formules en mots** :
- **Précision** = (# documents pertinents retrouvés) / (# total documents retrouvés)
- **Rappel** = (# documents pertinents retrouvés) / (# total documents pertinents dans collection)

**Exemple** :
- Système renvoie 8 docs pertinents + 10 non pertinents
- 20 pertinents au total dans collection
- **Précision** = 8/18 ≈ 0.44
- **Rappel** = 8/20 = 0.40

## Pourquoi pas Accuracy ?

**RI comme classification** :
- Documents en deux classes (pertinents/non pertinents)
- Accuracy = fraction classifications correctes

**Problèmes avec accuracy** :
- >**99.9%** documents non pertinents pour une requête
- Système maximisant accuracy → tout étiqueter non pertinent
- **Totalement insatisfaisant** pour RI
- Utilisateurs veulent voir documents, tolérance aux faux positifs

**Précision et rappel** :
- Focus sur **vrais positifs** (tp)
- Quel % documents pertinents trouvés ?
- Combien de faux positifs renvoyés ?

## Précision vs Rappel

**Importance relative** dépend du contexte :

**Haute précision** (internautes typiques) :
- Chaque résultat **première page** pertinent
- Pas d'intérêt pour tous les documents pertinents

**Haut rappel** (chercheurs professionnels - juristes, analystes) :
- Obtenir **tous** documents pertinents
- Tolérance résultats faible précision

**Compromis** :
- Précision et rappel **inversement proportionnels**
- Rappel = 1 facile (récupérer tous documents) → précision très faible
- **Rappel** : fonction non décroissante du nb documents récupérés
- **Précision** : diminue généralement quand nb documents augmente

## Mesure F

**Moyenne harmonique pondérée** :

$$F_\beta = (1 + \beta^2)\frac{(\text{Précision} \cdot \text{Rappel})}{\beta^2 \cdot \text{Précision} + \text{Rappel}}$$

**Paramètre $\beta$** :
- Importance relative rappel vs précision
- $\beta > 1$ accentue **rappel**
- $\beta < 1$ accentue **précision**

**$F_1$ couramment utilisé** : moyenne harmonique de précision et rappel

$$F_1 = \frac{2 \cdot \text{Précision} \cdot \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

**Moyenne harmonique** :
- Souligne importance **petites valeurs**
- vs moyenne arithmétique influencée par valeurs aberrantes

**Relation moyennes** : Q > A > G > H (Quadratic > Arithmetic > Geometric > Harmonic)

## Évaluation Résultats Classés

**Problème** : Précision, Rappel, F sont basés sur **ensembles non ordonnés**

**Solution** : étendre pour résultats classés
- Ensembles appropriés = **k premiers documents** récupérés
- Différencier classements avec même nb documents pertinents

**Trois approches de résumé** :
1. **Average Precision (AP) et MAP** : moyenne précisions aux positions où document pertinent récupéré
2. **Graphique Rappel-Précision interpolé** : précision à niveaux rappel standard 0.0 à 1.0
3. **Précision à rangs fixes** : P@K et R-Précision

## Average Precision (AP)

**Principe** : tenir compte valeurs précision aux points où **nouveau document pertinent** retourné

**Formule** :

$$\text{AP} = \frac{\sum_k (\text{Précision}(k) \cdot \text{rel}(k))}{\#\text{ total documents pertinents}}$$

où $\text{rel}(k) = 1$ si document à rang $k$ est pertinent, 0 sinon

**Exemple** :
```
Ranking #1: Précisions = [1.00, 0.50, 0.67, 0.75, 0.80, ...]
AP = (1.00 + 0.50 + 0.67 + 0.75 + 0.80 + 0.50) / 6 = 0.70
```

**Note importante** : diviser par nombre **total** documents pertinents dans collection (pas seulement ceux retrouvés)

## Mean Average Precision (MAP)

**Définition** : **moyenne des AP** sur plusieurs requêtes

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AP}(q)}{Q}$$

où $Q$ = nombre de requêtes

**Exemple avec 2 requêtes** :
- Query 1 : AP = 0.70
- Query 2 : AP = 0.48
- **MAP** = (0.70 + 0.48) / 2 = 0.59

**Utilité** : comparer **performances globales** d'un système à un autre

## Courbe Rappel-Précision

**Construction** : pour chaque document retourné, marquer coordonnées **(rappel, précision)**

**Interpolation** : lisser la courbe

$$p_{\text{interp}}(r) = \max_{r' \geq r} p(r')$$

où $p(r')$ = précision observée à rappel $r'$

**Propriétés** :
- Précision interpolée = **précision maximale** observée à niveau rappel ≥ actuel
- Produit **fonction d'étape**
- Définit précision au rappel 0.0

**Précision moyenne interpolée 11 points** :
- Points rappel standard : **0.0, 0.1, 0.2, 0.3, …, 1.0**
- Calculer précisions interpolées pour chaque requête
- Faire **moyenne** sur plusieurs requêtes
- Tracer graphique en joignant points moyens

**Comparaison systèmes** : courbe la plus proche du **coin supérieur droit** = meilleures performances

## Mesures Focalisées sur Meilleurs Documents

**Contexte** :
- Utilisateurs regardent seulement **haut de la liste**
- Certaines tâches : un seul document pertinent (navigation, question-réponse)
- Rappel pas approprié

**Précision @ rang K (P@K)** :
- K typiquement **5, 10, 20**
- **Facile** à calculer et comprendre
- Pas sensible aux positions < K

**R-Précision** :
- Précision à la **R-ième** position
- R = nombre documents pertinents pour cette requête dans collection
- Exemple : si 6 docs pertinents totaux, R-Précision = précision aux 6 premiers rangs

**Formule** :

$$\text{R-Précision} = \frac{\#\text{ docs pertinents dans top-R}}{R}$$

## Résumé Évaluation

**Principes** :
- **Aucune mesure** n'est LA seule bonne mesure
- Choisir mesures **adaptées à la tâche**
- Utiliser **combinaison de mesures**
- Évaluer différents aspects efficacité système
- Analyser performances **requêtes individuelles**

**Mesures principales** :
- **Précision/Rappel** : mesures de base
- **F-mesure** : compromis unique
- **MAP** : performance moyenne multi-requêtes
- **P@K** : focus sur top résultats
- **R-Précision** : adaptatif au nb documents pertinents
- **Graphiques interpolés** : visualisation performances