Créer un moteur de recherche

Webmonster Antilles

v1.0.0

L'objectif

L'objectif

- Vous apprendre comment fonctionne la recherche
- Augmenter votre confort avec les autres moteurs
- **NE PAS** développer un autre moteur

Pourquoi se soucier de la recherche ?

Pourquoi la recherche interne ?

- Les crawlers standards doivent scraper du HTML.
- Vous connaissez le modèle de données mieux qu'eux
- Peut-être que ce n'est pas du tout une application web !

Notions fondamentales

Notions fondamentales

- Basé sur les documents
- Ne jamais regarder simplement une chaîne de caractères
- Index inversé
- Dédoublement
- N-gram
- Pertinence

Terminologie

Terminologie

- Moteur
- Document
- Corpus
- Stopword
- Racine
- Position
- Segments
- Pertinence
- Faceting
- Booster

Moteur

Moteur

La boîte noire à laquelle vous adressez une requête et dont vous obtenez les résultats.

Document

Document

Un blob (Binary large object) de texte
avec des métadonnées facultatives

Corpus

Corpus

La collection de tous les documents

Stopword

Stopword

Un mot court qui ne contribue pas à la pertinence et qui est généralement ignoré.

“et”, “un”, “le”, “mais”…

Dédoublement

Racine

Trouver la racine d'un mot

Segments

Segments

Données partagées stockant l'indice inversé.

Pertinence

Pertinence

Le ou les algorithmes utilisés pour classer les résultats en fonction de la requête.

Faceting

Faceting

Fournir le nombre de résultats parmi les résultats qui correspondent à certains critères.

“Forage en profondeur”

Booster

Booster

Améliorer artificiellement la pertinence de certains documents en fonction d'une condition.

Indexation

Indexation

Quatre composants principaux

- Réception et stockage des documents
- Tokenisation
- Génération de termes
- Indexation des termes

Documents

Documents

- PAS une ligne dans la BD
- Pensez à une masse de texte + métadonnées
- La qualité du texte est plus importante !
- Flat et non relationnel !
- Ne pas normaliser

Tokenisation

Tokenisation

En utilisant le blob de texte, vous :
- Fractionner chaque espace
- Minuscule
- Filtrer les stopwords
- Supprimer la ponctuation
- Etc.

Le but est de normaliser les tokens.

De petites unités atomiques cohérentes auxquelles nous pouvons attribuer un sens et avec lesquelles nous pouvons travailler.

Racine

Racine

- Pour éviter de rechercher manuellement dans tout le blob, vous tokenisez
- Davantage de post-traitement
- Ensuite ! vous trouvez le mot racine

Racine (exp)

Exemples :

“tester” => “test”
“chercheurs” => “chercheur” => “recherche”

Racine

Ceux-ci deviennent les termes de l'index inversé.

Lorsque vous faites la même chose à la requête, vous pouvez les faire correspondre

Racine

Cette méthode ne fonctionne que si vous connaissez la structure grammaticale de la langue.

Racine

La plupart sont spécifiques à l'anglais, mais d'autres langues sont disponibles.

Difficile de faire fonctionner avec du multilingue

Comment résoudre ce problème ?

Générons les termes sous un angle différent...

N-grams

N-grams

Résout certaines des lacunes de l'extraction avec de nouveaux compromis.

Passe une "fenêtre" sur les données tokenisées.

Ces fenêtres de données deviennent les termes de l'index.

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai”]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’, ‘iso’]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’, ‘iso’, ‘son’]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’, ‘iso’, ‘son’,

‘rou’]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’, ‘iso’, ‘son’,

‘rou’, ‘oug’]

N-grams

Exemple (gram taille de 3) :

maison rouge

[‘mai’, ‘ais’, ‘iso’, ‘son’,

‘rou’, ‘oug’, ‘uge’]

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

[‘mai’]

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

['mai', 'mais']

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

['mai', 'mais', 'maiso']

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

['mai', 'mais', 'maiso', 'maison']

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

['mai', 'mais', 'maiso', 'maison',

'rou']

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

['mai', 'mais', 'maiso', 'maison',

'rou', 'roug']

N-grams

Généralement utilisé avec des tailles de grammes multiples

Exemple (gram taille de 3 à 6) :

maison rouge

[‘mai’, ‘mais’, ‘maiso’, ‘maison’,

‘rou’, ‘roug’, ‘rouge’]

N-grams

Pour :

- Idéal pour l'autocomplétion (correspond à de petits fragments rapidement)
- Fonctionne dans toutes les langues (même asiatiques)

N-grams

Contre :

- De nombreux autres termes dans l'index
- La qualité initiale peut en souffrir

N-grams

```
min_gram = 3
max_gram = 6
terms = {}

for position, token in enumerate(tokens):
    for window_length is range(min_gram, min(max_gram + 1, len(token))):
        gram = token[:window_length]
        terms.setdefault(gram, set([]))
        terms[gram].add(position)
```

Index inversé

Index inversé

- Le cœur du moteur
- Comme un dictionnaire
- Les clés sont importantes (termes de tous les documents)
- Stocke la position et les identifiants des documents

Index inversé

Les clés sont des termes
Les valeurs sont les ‘documents’ + [position]

```
index = {
    'blob': {
        'document-1524' : [3],
    },
    'text': {
        'document-1524' : [5, 10],
    }
}
```

Segments

Segments

- Il existe de nombreuses façons de procéder
- Beaucoup suivent **Lucene** *
- Nous allons **tricher** et adopter une approche légèrement plus simple.

Lucene est une bibliothèque open source écrite en Java qui permet d'indexer et de chercher du texte .

Segments

- Fichiers plats
- Clés hachées
- Toujours triés
- Utiliser JSON pour les données de position/document

Segments

```
import hashlib
import json
import os

def make_segment_same(term, length = 6):
    hashed = hashlib.md5(term).hexdigest()
    return "{0}.index".format(hashed[:length])

def save_segment(term, term_info):
    seg_name = make_segment_same(term)

    with open(seg_name, 'a') as seg_file:
        seg_file.write("{0}\t{1}\n".format(term, json.dumps(term_info)))
```

Recherche

Recherche

Trois composants principaux

- Analyseur de requêtes
- Lecteur d'index
- Révélation des scores

Analyseur de requêtes

Analyseur de requêtes

Analyse de la structure

Traitez les éléments de la même manière que vous avez préparé le document.

Analyseur de requêtes

```
STOP_WORDS = set([
  'un', 'une', 'et', 'à', 'comme', 'mais', 'par', 'ou', 'où', 'pour', 'si', 'dans', 'es', 'ca',
  'cela', 'si', 'non', 'il', 'elle', 'les', 'le', 'mon', 'ton', 'son', 'ma', 'ta', 'sa', 'mes',
  'ses', 'leurs', 'puis', 'ensuite', 'enfin', 'aussi', 'même', 'encore', 'car'

])

def parse_query(query):
    tokens = [token for token in query.split() if not token in STOP_WORDS]
    return make_ngrams(tokens)
```

Lecteur d’index

Lecteur d’index

Par terme, hacher le terme pour obtenir le bon fichier.

Passez en revue et collectez tous les résultats des positions et des documents.

Lecteur d’index

```
def load_segment(term):
    seg_name = make_segment_same(term)

    if not os.path.exists(seg_name):
        return {}

    with open(seg_name, 'r') as seg_file:
        for line in seg_file:
            seg_term, term_info = line.rstrip().split('\t', i)
            if seg_name == term:
                return json.loads(term_info)
    return ()
```

Lecteur d’index

```
def collect_results(terms):
    matches = {}
    for term in terms:
        matches[term] = load_segment(term)
    return matches
```

Révélation des scores

Révélation des scores

Réorganiser la collection de documents en fonction de leur adéquation avec la requête.

De nombreux choix pour la méthode de pondération :

- Okapi BM25
- Phased
- Google PageRank

Révélation des scores

```
def bm25_relevance(terms, matches, current_docs, total_docs, b = 0, k = 1.2):
    score = b

    for term in terms:
        idf = math.log((total_docs = matches[term] + 1) / matches[term]) / math.log(1.0 + total_docs)
        score = score + current_docs[term] + idf / (current_docs[term] + k)

    return 0.5 + score / (2 * len(terms))
```

Thèmes avancés

Faceting

Faceting

- Pour un champ donné, rassemblez tous les termes
- Compter la longueur des identifiants uniques (document)
- Mettre en ordre décroissant

Booster

Booster

- Pendant le processus de notation.
- Si une condition est remplie, modifiez la note en conséquence.

Plus comme ceci...

Plus comme ceci

- Rassembler tous les termes d'un document donné
- Triez en fonction du nombre de fois qu'un document est vu dans l'ensemble.
- Il s'agit d'une vue simpliste
- Des solutions plus complètes utilisent le NLP (traitement automatique des langues) pour améliorer la qualité

Merci !

Rédaction : Webmonster Antilles

Chef de projet : **XavierS**

Inspiration : **R3tr0_**

Source : Daniel Lindsley - Lawrence, KS