TP3 - Moteur de recherche

Durée : 3 heures

Objectif

Développer un moteur de recherche qui utilise les index créés précédemment pour retourner et classer des résultats pertinents.

Données d'entrée

- Index créés au TP précédent (ou fournis) au format JSON
  - Le fichier products.jsonl a été modifié pour pouvoir créer des index plus intéressants :
    - Tous les documents ont maintenant une origine (si ils ont un dictionnaire `product_features`)
    - Le nom de la variante se trouve dans le titre après un tiret
    - Lorsque la description avait plus de deux phrases, un des phrases a été supprimée aléatoirement
- Un fichier json avec quelques synonymes de noms de pays

Étapes guidées

1. Lecture et préparation (15 min)
   - Charger les index
   - Mettre en place les fonctions de tokenization (identiques au TP précédent)
   - Implémenter la lecture des synonymes

2. Filtrage des documents (45 min)
   - Développer les fonctions de traitement de requête :
     - Tokenization
     - Normalisation
     - Augmentation de la requête par synonymes (applicable par exemple pour l'origine des produits)
   - Implémenter le filtrage des documents :
     - Créer une fonction qui vérifie si au moins un des tokens est présent
     - Créer une fonction qui vérifie la présence de tous les tokens, à l'exception des stopwords
   - NB: les stopwords ont été générés via la liste fournie par NLTK

```
6   import nltk
7   from nltk.corpus import stopwords
8   nltk.download("stopwords")
9   STOPWORDS = stopwords.words("english")
0
```

3. Ranking (30 min)
   - Analyser les données disponibles pour identifier les signaux pertinents
   - Implémenter la fonction bm25 ainsi qu'une fonction de match exact
   - Implémenter une fonction de scoring linéaire combinant :
     - Fréquence des tokens dans les documents
     - Présence dans le titre vs description
     - Les avis
     - Autres signaux identifiés comme pertinents
     - Utilisez l'information de position quand vous y avez accès
   - Ici vous êtes plutôt libres sur le choix des champs / features à utiliser.

4. Testing et optimisation (30 min)
    - Créer un jeu de requêtes test
    - Analyser les résultats
    - Ajuster les poids, features et paramètres
    - Documenter les choix et leurs impacts

Rappels de programmation:

- Une fonction ne fait qu'une action, si vous avec envie de nommer votre fonction do_something_and_do_something_else -> alors il vous faut deux fonctions
- Le nom d'une fonction commence toujours pas un verbe d'action
- Les noms des fonctions/variables doit être écrit en anglais, tout comme la documentation.

Livrable

- Formater les résultats en JSON avec pour chaque document:
    - Titre
    - URL
    - Description
    - Score de ranking
- Ainsi que des métadonnées
    - nombre total de documents
    - documents filtrés

Critères d'évaluation

- Qualité du filtrage
- Pertinence du ranking
- Originalité des signaux utilisés
- Qualité de l'analyse
- Documentation des choix

Conseils

- Commencer simple et itérer
- Tester avec des requêtes variées
- Documenter les observations
- Explorer les données pour identifier des signaux pertinents
- Réfléchir aux cas particuliers (requêtes longues, mots rares, etc.)