RI et graphe de données - TP Base de données vectorielle

L'objectif de ce TP est de préparer développé une base de données vectorielle d'embeddings sémantique dynamique d'extrait de documents texte.

Complément de cours

Dans le TP précédent, nous avons mis en œuvre un moteur de recherche basé sur un modèle booléen et un modèle vectoriel classique (tf.idf).

Dans ce TP, nous allons introduire une approche plus moderne de la recherche d'information, fondée sur les représentations vectorielles (*embeddings*). Ces représentations sont obtenues à l'aide de modèles de langage, qui permettent de projeter des textes dans un espace vectoriel sémantique.

Dans cet espace :

— deux textes proches sémantiquement ont des vecteurs proches ;
— la recherche d'information devient un problème de **similarité vectorielle**.

Dans un environnement professionnel, les embeddings sont souvent calculés avec des modèles lourds (utilisant PyTorch ou TensorFlow), nécessitant un GPU. Dans le cadre de ce TP, nous utilisons la bibliothèque `fastembed` qui ne nécessite pas de GPU et qui est assez légère et rapide à installer.

La bibliothèque `fastembed` repose principalement sur la classe `TextEmbedding`. Elle permet de charger un modèle d'embeddings et de calculer les vecteurs associés à des textes. Elle s'utilise de la manière suivante :

```
from fastembed import TextEmbedding

modele = TextEmbedding(
    model_name="sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"
)
```

Le paramètre `model_name` permet de choisir le modèle utilisé. Dans ce TP, nous utilisons un modèle multilingue compact, adapté à la langue française et à une exécution sur CPU.

La méthode principale de cette classe est `embed`. Elle permet de transformer un ou plusieurs textes en vecteurs numériques.

```
vecteurs = modele.embed(["bonjour le monde", "formation informatique"])
```

Cette méthode prend en entrée une liste de textes. Elle retourne un itérable de vecteurs (souvent converti en liste) représentants les textes dans l'espace vectoriel sémantique.

Description de l'application

Le code implémente une mini base de données vectorielle permettant :

— de collecter des documents Web ;
— de les transformer en représentations vectorielles ;

— de rechercher les passages les plus pertinents pour une requête.

Utilisation du programme principal

Le script principal est `main.py`. Il propose plusieurs commandes :

— `recuperer URL` : télécharge une page Web et stocke son contenu textuel dans le répertoire `corpus` ;
— `parcourir URL N` : explore un site Web à partir d'une URL de départ et récupère jusqu'à N pages ;
— `indexer` : construit la base vectorielle à partir des documents du corpus ;
— `rechercher "requête"` : recherche les passages les plus pertinents dans la base.

Le fonctionnement global est le suivant :

1. récupération de documents (recuperer ou parcourir) ;
2. stockage sous forme de fichiers JSON dans le répertoire `corpus` ;
3. découpage des documents en fragments (chunks) ;
4. transformation en vecteurs (embeddings) ;
5. stockage dans une base vectorielle dans le répertoire `db` ;
6. recherche par similarité.

Description des modules

Le projet est structuré en plusieurs modules du package `base_de_donnees_vectorielle` :

— `modele.py` : définition des structures de données (Document, ChunkVecteur, ResultatRecherche) ;
— `recuperation.py` : téléchargement des pages Web, extraction du texte et parcours de site ;
— `chargeur_corpus.py` : lecture des documents stockés dans le répertoire `corpus` ;
— `decoupage.py` : découpage des documents en fragments (chunks) de taille fixe avec recouvrement ;
— `embeddings.py` : calcul des représentations vectorielles des textes ;
— `stockage.py` : sauvegarde et chargement de la base vectorielle dans le répertoire `db` ;
— `similarite.py` : calcul de la similarité cosinus entre vecteurs ;
— `requeteur.py` : recherche des chunks pertinents dans la base ;

Travail à réaliser

L'objectif de ce TP est de compléter certaines parties du code afin d'obtenir un moteur de recherche vectoriel fonctionnel. Les modules à compléter sont `similarite.py`, `embeddings.py` et `requeteur.py`.

Ce TP utilise `uv` pour la gestion des dépendances. Il nécessite une version de python supérieur ou égale à 3.13.2.

Testez votre programme en récupérant, indexant et recherchant des informations sur le site de description de notre formation.