Fiche Documentaire du Code : *embedding*

Titre du projet :

Analyse personnalisée des articles scientifiques (à l'aide des LLM et d'un paradigme RAG**)**

Auteurs :

- **Mohamed Nassih** : Délégué de l'équipe – Responsable du génération des embeddings

Description du code :

Ce code génère des **embeddings** pour des articles scientifiques à l'aide de modèles de type *Transformer* comme **DistilBERT**, **GloVe** et **SentenceTransformers**. Ces embeddings sont ensuite stockés dans une base de données vectorielle Chroma, afin de permettre des recherches basées sur des similarités pour des requêtes utilisateurs.

Structure du code :

- **embedding.ipynb** : Ce notebook contient le processus de génération des embeddings pour les paragraphes d'articles segmentés. Il compare plusieurs modèles d'embeddings et stocke les résultats dans une base de données Chroma.

Structure des fichiers générés :

- **Embeddings/** : Répertoire (ou collection) où sont stockés les embeddings des articles.
- **chroma_db_persistent/** : Répertoire qui contient les embeddings stockés pour les recherches futures dans Chroma.

Instructions d'installation :

1. **Installer les bibliothèques nécessaires :** Les principales bibliothèques à installer sont transformers, sentence-transformers, torch, et chromadb :

```
pip install transformers sentence-transformers torch chromadb
```

2. **Configurer l'environnement :**
    - Vous devez avoir un environnement Python 3.10 ou supérieur.
    - Assurez-vous que Chroma est bien configuré pour stocker les embeddings de manière persistante.

Prérequis :

- **Python 3.10**
- **Chroma** : Une base de données vectorielle pour stocker et interroger les embeddings.
- **Modèles pré-entraînés** : Vous devez télécharger les modèles de type BERT ou SentenceTransformer depuis Hugging Face.

Fonctionnement du code :

1. **Chargement et Prétraitement des données** :
    - Les articles scientifiques sont chargés sous forme de texte.

- Chaque article est segmenté en paragraphes, ces segments sont ensuite transformés en vecteurs d'embeddings via les modèles d'apprentissage automatique.

2. **Comparaison des modèles d'embedding** :
   - Le notebook compare plusieurs modèles d'embeddings, notamment **DistilBERT**, **GloVe**, et **SentenceTransformers**.
   - Chaque modèle génère des embeddings pour les mêmes paragraphes, et les résultats sont comparés selon des mesures de similarité cosinus.
3. **Stockage dans Chroma** :
   - Les embeddings générés pour chaque paragraphe sont stockés dans une base de données Chroma pour être facilement récupérés lors de futures recherches.
4. **Interrogation des embeddings** :
   - Le notebook inclut un code permettant d'interroger les embeddings stockés dans Chroma, à partir d'une requête utilisateur sous forme de texte.

Entrées et sorties :

Entrées :

- Fichiers texte des articles segmentés, extraits lors de l'étape de segmentation automatique.
- Modèles d'embeddings pré-entraînés comme **DistilBERT**, **GloVe** ou **SentenceTransformers**.

Sorties :

- Fichiers d'embeddings stockés dans Chroma.
- Résultats des similarités cosinus entre les paragraphes et les requêtes utilisateurs.

Exemples d'utilisation :

Exemple de code pour générer des embeddings à partir d'un modèle comme DistilBERT :

```
from transformers import DistilBertTokenizer, DistilBertModel

tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = DistilBertModel.from_pretrained('distilbert-base-uncased')

def get_distilbert_embeddings(text):
    inputs = tokenizer(text, return_tensors='pt', max_length=512, truncation=True
    outputs = model(**inputs)
    return outputs.last_hidden_state.mean(dim=1).detach().numpy()
```

Ensuite, les embeddings sont stockés dans Chroma :

```
collection.add(
    ids=[f"{i}_{j}"],
    documents=[paragraph],
    embeddings=[embeddings],
    metadatas=[{"file_name": file_names[i], "paragraph_index": j}]
)
```

Tests et validation :

Les tests ont consisté à :

- Comparer les performances des modèles d'embedding (DistilBERT, GloVe, SentenceTransformers) à l'aide de similarités cosinus.
- Valider le stockage et la récupération des embeddings dans Chroma via des requêtes textuelles.

Problèmes connus :

- **Limite de mémoire** : Le processus de génération des embeddings peut consommer beaucoup de mémoire, surtout pour les articles volumineux.
- **Performances variables** : Les performances des modèles d'embeddings varient selon la nature des articles, et certains modèles comme GloVe peuvent être moins performants que des modèles plus modernes comme DistilBERT.