Mistral AI Embeddings

Pour fournir des exemples d'intégration en utilisant Mistral AI, vous pouvez utiliser leur API d'intégration pour générer des représentations numériques (intégrations) de texte qui capturent le sens sémantique. Ces intégrations sont utiles pour des tâches telles que la détection de similarité, le regroupement, la classification ou la recherche. Ci-dessous, je vais démontrer comment générer des intégrations pour un ensemble de phrases et calculer leurs similarités en utilisant le point de terminaison d'intégration de Mistral AI. Cet exemple s'appuie sur la structure du code fourni mais l'adapte spécifiquement pour les intégrations, comme décrit dans la documentation : Mistral AI Embeddings.

Prérequis

Avant d'exécuter l'exemple, assurez-vous d'avoir : 1. **Clé API** : Une clé API Mistral AI valide stockée dans la variable d'environnement `MISTRAL_API_KEY`. 2. **Dépendances** : Les packages Python requis installés. Vous pouvez les installer avec : `bash pip install requests numpy`

Exemple de Code

Voici un script Python complet qui génère des intégrations pour trois phrases et calcule leurs similarités par paires :

```
import os
import requests
import numpy as np

def call_mistral_embeddings_api(texts, model="mistral-embed"):
    """
    Appeler l'API d'intégration Mistral AI pour générer des intégrations pour une liste de textes.

    Args:
        texts (list): Liste de chaînes de caractères à intégrer.
        model (str): Le modèle d'intégration à utiliser (par défaut : "mistral-embed").

    Returns:
        list: Liste de vecteurs d'intégration, ou None si la requête échoue.
    """
    api_key = os.environ.get("MISTRAL_API_KEY")
    if not api_key:
        print("Erreur : la variable d'environnement MISTRAL_API_KEY n'est pas définie.")
        return None
```

```
    url = "https://api.mistral.ai/v1/embeddings"
    headers = {
        "Content-Type": "application/json",
        "Accept": "application/json",
        "Authorization": f"Bearer {api_key}"
    }
    data = {
        "model": model,
        "input": texts
    }

    try:
        response = requests.post(url, headers=headers, json=data)
        response.raise_for_status()
        response_json = response.json()
        if response_json and "data" in response_json:
            embeddings = [item["embedding"] for item in response_json["data"]]
            return embeddings
        else:
            print(f"Erreur de l'API Mistral Embeddings : Format de réponse invalide : {response_json}")
            return None
    except requests.exceptions.RequestException as e:
        print(f"Erreur de l'API Mistral Embeddings : {e}")
        if e.response:
            print(f"Code de statut de la réponse : {e.response.status_code}")
            print(f"Contenu de la réponse : {e.response.text}")
        return None

def calculate_similarity(emb1, emb2):
    """
    Calculer la similarité entre deux intégrations en utilisant le produit scalaire.

    Args:
        emb1 (list): Premier vecteur d'intégration.
        emb2 (list): Deuxième vecteur d'intégration.

    Returns:
        float: Score de similarité (produit scalaire, équivalent à la similarité cosinus pour des vecteurs no
    """
```

```
    return np.dot(emb1, emb2)

if __name__ == "__main__":
    # Exemples de textes à intégrer
    texts = [
        "J'aime programmer en Python.",
        "Python est un excellent langage de programmation.",
        "Le temps est ensoleillé aujourd'hui."
    ]

    # Générer des intégrations
    embeddings = call_mistral_embeddings_api(texts)
    if embeddings:
        # Afficher la dimension de l'intégration
        print(f"Dimension de l'intégration : {len(embeddings[0])}")

        # Calculer les similarités par paires
        sim_12 = calculate_similarity(embeddings[0], embeddings[1])
        sim_13 = calculate_similarity(embeddings[0], embeddings[2])
        sim_23 = calculate_similarity(embeddings[1], embeddings[2])

        # Afficher les résultats
        print(f"\nRésultats de Similarité :")
        print(f"Texte 1 : '{texts[0]}'")
        print(f"Texte 2 : '{texts[1]}'")
        print(f"Texte 3 : '{texts[2]}'")
        print(f"\nSimilarité entre Texte 1 et Texte 2 : {sim_12:.4f}")
        print(f"Similarité entre Texte 1 et Texte 3 : {sim_13:.4f}")
        print(f"Similarité entre Texte 2 et Texte 3 : {sim_23:.4f}")
```

Comment Exécuter

1. **Définir la Clé API** :

   ```
   export MISTRAL_API_KEY="votre_clé_api_ici"
   ```

2. **Enregistrer et Exécuter** : Enregistrez le script (par exemple, sous le nom `embedding_example.py`) et exécutez-le :

   ```
   python embedding_example.py
   ```

Sortie Attendue

En supposant que l'appel API réussit, vous verrez une sortie similaire à celle-ci (les valeurs exactes dépendent des intégrations retournées) :

```
Dimension de l'intégration : 1024

Résultats de Similarité :
Texte 1 : 'J'aime programmer en Python.'
Texte 2 : 'Python est un excellent langage de programmation.'
Texte 3 : 'Le temps est ensoleillé aujourd'hui.'

Similarité entre Texte 1 et Texte 2 : 0.9200
Similarité entre Texte 1 et Texte 3 : 0.6500
Similarité entre Texte 2 et Texte 3 : 0.6700
```

Explication

- **Point de Terminaison de l'API** : La fonction `call_mistral_embeddings_api` envoie une requête POST à `https://api.mistral.ai/v1/embeddings`, en passant une liste de textes et le modèle `"mistral-embed"`. L'API retourne une réponse JSON contenant les intégrations sous la clé `"data"`.
- **Intégrations** : Chaque intégration est un vecteur de 1024 dimensions (selon la documentation de Mistral), représentant le contenu sémantique du texte d'entrée. Les intégrations sont normalisées à une norme de 1.
- **Calcul de Similarité** : Puisque les intégrations sont normalisées, le produit scalaire (`np.dot`) entre deux intégrations équivaut à leur similarité cosinus. Des valeurs plus élevées indiquent une similarité sémantique plus grande :
  - **Texte 1 et Texte 2** : Les deux concernent la programmation en Python, donc leur similarité devrait être élevée (par exemple, ~0.92).
  - **Texte 1 et Texte 3** : L'un concerne la programmation, l'autre la météo, donc leur similarité devrait être plus faible (par exemple, ~0.65).
  - **Texte 2 et Texte 3** : Même schéma, similarité plus faible en raison des sujets différents.

Cas d'Utilisation Supplémentaires

Les intégrations de Mistral AI peuvent être appliquées au-delà de la détection de similarité : - **Regroupement** : Regrouper des textes similaires en utilisant des algorithmes comme K-means (nécessite `scikit-learn`). `python from sklearn.cluster import KMeans kmeans = KMeans(n_clusters=2).fit(np.array(embeddi print(kmeans.labels_)` - **Recherche** : Intégrer une requête et trouver le texte le plus similaire : `python`

```
query = "Parlez-moi de Python."  query_emb = call_mistral_embeddings_api([query])[0]  similarities
= [calculate_similarity(query_emb, emb) for emb in embeddings]  most_similar = texts[np.argmax(similarities)]
print(f"Texte le plus similaire : '{most_similar}'")
```
- **Classification** : Utiliser les intégrations comme caractéristiques dans un modèle d'apprentissage automatique.

Notes

- **Modèle** : Le modèle `"mistral-embed"` est optimisé pour les intégrations. Consultez la documentation pour les mises à jour ou les modèles supplémentaires.
- **Traitement par lots** : L'API traite plusieurs textes en une seule requête, améliorant ainsi l'efficacité.
- **Applications Avancées** : Pour une recherche de similarité à grande échelle, intégrez avec des bases de données vectorielles comme Faiss ou Milvus.

Cet exemple fournit une introduction pratique à l'utilisation des intégrations Mistral AI, adaptable à la structure du code de complétion de chat fourni en changeant le point de terminaison et le format des données.