Mistral AI Embeddings

Um Einbettungsbeispiele mit Mistral AI zu erstellen, können Sie deren Einbettungs-API nutzen, um numerische Darstellungen (Einbettungen) von Text zu generieren, die semantische Bedeutung erfassen. Diese Einbettungen sind nützlich für Aufgaben wie Ähnlichkeitserkennung, Clustering, Klassifizierung oder Suche. Im Folgenden werde ich demonstrieren, wie man Einbettungen für einen Satz von Sätzen generiert und deren Ähnlichkeiten mit dem Mistral AI Einbettungs-Endpoint berechnet. Dieses Beispiel baut auf der Struktur des bereitgestellten Codes auf, passt ihn jedoch speziell für Einbettungen an, wie in der Dokumentation beschrieben: Mistral AI Einbettungen.

Voraussetzungen

Bevor Sie das Beispiel ausführen, stellen Sie sicher, dass Sie Folgendes haben: 1. **API-Schlüssel**: Ein gültiger Mistral AI API-Schlüssel, der in der Umgebungsvariablen `MISTRAL_API_KEY` gespeichert ist. 2. **Abhängigkeiten**: Die erforderlichen Python-Pakete installiert. Sie können sie mit folgendem Befehl installieren: `bash pip install requests numpy`

Beispielcode

Hier ist ein vollständiges Python-Skript, das Einbettungen für drei Sätze generiert und deren paarweise Ähnlichkeiten berechnet:

```
import os
import requests
import numpy as np

def call_mistral_embeddings_api(texts, model="mistral-embed"):
    """
    Ruft die Mistral AI Einbettungs-API auf, um Einbettungen für eine Liste von Texten zu generieren.

    Args:
        texts (list): Liste von Zeichenketten, die eingebettet werden sollen.
        model (str): Das Einbettungsmodell, das verwendet werden soll (Standard: "mistral-embed").

    Returns:
        list: Liste von Einbettungsvektoren oder None, wenn die Anfrage fehlschlägt.
    """
    api_key = os.environ.get("MISTRAL_API_KEY")
    if not api_key:
        print("Fehler: Die Umgebungsvariable MISTRAL_API_KEY ist nicht gesetzt.")
```

```
        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"Mistral Einbettungs-API Fehler: Ungültiges Antwortformat: {response_json}")
            return None
    except requests.exceptions.RequestException as e:
        print(f"Mistral Einbettungs-API Fehler: {e}")
        if e.response:
            print(f"Antwortstatuscode: {e.response.status_code}")
            print(f"Antwortinhalt: {e.response.text}")
        return None

def calculate_similarity(emb1, emb2):
    """
    Berechnet die Ähnlichkeit zwischen zwei Einbettungen mithilfe des Skalarprodukts.

    Args:
        emb1 (list): Erster Einbettungsvektor.
        emb2 (list): Zweiter Einbettungsvektor.

    Returns:
        float: Ähnlichkeitswert (Skalarprodukt, äquivalent zur Kosinusähnlichkeit für normalisierte Vektoren)
```

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

if __name__ == "__main__":
    # Beispieltexte zum Einbetten
    texts = [
        "Ich liebe Programmieren in Python.",
        "Python ist eine großartige Programmiersprache.",
        "Das Wetter ist heute sonnig."
    ]

    # Generiere Einbettungen
    embeddings = call_mistral_embeddings_api(texts)
    if embeddings:
        # Drucke Einbettungsdimension
        print(f"Einbettungsdimension: {len(embeddings[0])}")

        # Berechne paarweise Ähnlichkeiten
        sim_12 = calculate_similarity(embeddings[0], embeddings[1])
        sim_13 = calculate_similarity(embeddings[0], embeddings[2])
        sim_23 = calculate_similarity(embeddings[1], embeddings[2])

        # Zeige Ergebnisse an
        print(f"\nÄhnlichkeitsergebnisse:")
        print(f"Text 1: '{texts[0]}'")
        print(f"Text 2: '{texts[1]}'")
        print(f"Text 3: '{texts[2]}'")
        print(f"\nÄhnlichkeit zwischen Text 1 und Text 2: {sim_12:.4f}")
        print(f"Ähnlichkeit zwischen Text 1 und Text 3: {sim_13:.4f}")
        print(f"Ähnlichkeit zwischen Text 2 und Text 3: {sim_23:.4f}")
```

Ausführung

1. **API-Schlüssel setzen**:

   ```
   export MISTRAL_API_KEY="your_api_key_here"
   ```

2. **Speichern und Ausführen**: Speichern Sie das Skript (z.B. als `embedding_example.py`) und führen Sie es aus:

   ```
   python embedding_example.py
   ```

Erwartete Ausgabe

Angenommen, der API-Aufruf ist erfolgreich, sehen Sie eine Ausgabe wie diese (genaue Werte hängen von den zurückgegebenen Einbettungen ab):

```
Einbettungsdimension: 1024

Ähnlichkeitsergebnisse:
Text 1: 'Ich liebe Programmieren in Python.'
Text 2: 'Python ist eine großartige Programmiersprache.'
Text 3: 'Das Wetter ist heute sonnig.'

Ähnlichkeit zwischen Text 1 und Text 2: 0.9200
Ähnlichkeit zwischen Text 1 und Text 3: 0.6500
Ähnlichkeit zwischen Text 2 und Text 3: 0.6700
```

Erklärung

- **API-Endpoint**: Die Funktion `call_mistral_embeddings_api` sendet eine POST-Anfrage an `https://api.mistral.ai`, wobei eine Liste von Texten und das Modell `"mistral-embed"` übergeben werden. Die API gibt eine JSON-Antwort zurück, die Einbettungen unter dem Schlüssel `"data"` enthält.
- **Einbettungen**: Jede Einbettung ist ein 1024-dimensionaler Vektor (laut Mistral-Dokumentation), der den semantischen Inhalt des Eingabetexts darstellt. Die Einbettungen sind auf eine Norm von 1 normalisiert.
- **Ähnlichkeitsberechnung**: Da die Einbettungen normalisiert sind, entspricht das Skalarprodukt (`np.dot`) zwischen zwei Einbettungen ihrer Kosinusähnlichkeit. Höhere Werte deuten auf eine größere semantische Ähnlichkeit hin:
  - **Text 1 und Text 2**: Beide handeln vom Programmieren in Python, daher sollte ihre Ähnlichkeit hoch sein (z.B. ~0.92).
  - **Text 1 und Text 3**: Einer handelt vom Programmieren, der andere vom Wetter, daher sollte ihre Ähnlichkeit niedriger sein (z.B. ~0.65).
  - **Text 2 und Text 3**: Ähnliches Muster, niedrigere Ähnlichkeit aufgrund unterschiedlicher Themen.

Zusätzliche Anwendungsfälle

Einbettungen von Mistral AI können über die Ähnlichkeitserkennung hinaus angewendet werden: - **Clustering**: Gruppiere ähnliche Texte mit Algorithmen wie K-means (erfordert `scikit-learn`). `python from sklearn.cluster import KMeans kmeans = KMeans(n_clusters=2).fit(np.array(embeddings)) print(kmeans.labels_` - **Suche**: Einbetten Sie eine Abfrage und finden Sie den ähnlichsten Text: `python query = "Erzähl mir`

```
etwas über Python."  query_emb = call_mistral_embeddings_api([query])[0]  similarities = [calculate_similari
emb) for emb in embeddings]  most_similar = texts[np.argmax(similarities)]  print(f"Ähnlichster
Text: '{most_similar}'")
```
- **Klassifizierung**: Verwenden Sie Einbettungen als Merkmale in einem maschinellen Lernmodell.

Hinweise

- **Modell**: Das Modell `"mistral-embed"` ist für Einbettungen optimiert. Überprüfen Sie die Dokumentation auf Aktualisierungen oder zusätzliche Modelle.
- **Batching**: Die API verarbeitet mehrere Texte in einem Aufruf, was die Effizienz verbessert.
- **Erweiterte Anwendungen**: Für groß angelegte Ähnlichkeitssuchen integrieren Sie mit Vektordatenbanken wie Faiss oder Milvus.

Dieses Beispiel bietet eine praktische Einführung in die Verwendung von Mistral AI Einbettungen, die an die Struktur des bereitgestellten Chat-Abschlusscodes angepasst werden kann, indem der Endpunkt und das Datenformat geändert werden.