Embedding Süreci ve FAISS Entegrasyonu

Bu projede ilk olarak *embedding* kavramını araştırmam gerekti. Başlangıçta bu terime dair hiçbir fikrim yoktu. Araştırmalarım sonucunda embedding işleminin, kelime ya da cümleleri sayısal vektörlere dönüştürme işlemi olduğunu öğrendim. Bu vektör temsilleri sayesinde, anlam olarak benzer cümlelerin sayısal olarak da birbirine yakın olduğu anlaşılabiliyor. Bu yaklaşımın temelinde ise vektörler arasındaki *dot product* (noktasal çarpım) işlemi yatıyor. Fizikte de bilindiği üzere, iki vektör aynı yönü gösterdiğinde dot product maksimum olur. Bu da doğal dilde anlamca benzer iki cümlenin vektör temsillerinin de benzer yönlerde olacağı anlamına gelir.

Bu işlemi gerçekleştirmek için `sentence-transformers` kütüphanesini kullandım. Bu kütüphane Hugging Face üzerine inşa edilmiş bir yapıdır. Embedding için `"sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2"` modelini seçtim. Bu modeli tercih etme sebebim, 50'den fazla dili desteklemesi; çünkü ileride sistemin çok dilli çalışabilmesi istenebilir.

Ardından, her biri `id`, `soru` ve `cevap` alanlarına sahip bir `dict` yapısında sorulardan oluşan bir liste oluşturdum. ID değerlerini, ileride spesifik sorulara ulaşmak için kullanabileceğimi düşündüm. Konu başlıklarını, aldığım makine öğrenmesi dersi doğrultusunda, bu alana uygun seçmeye çalıştım.

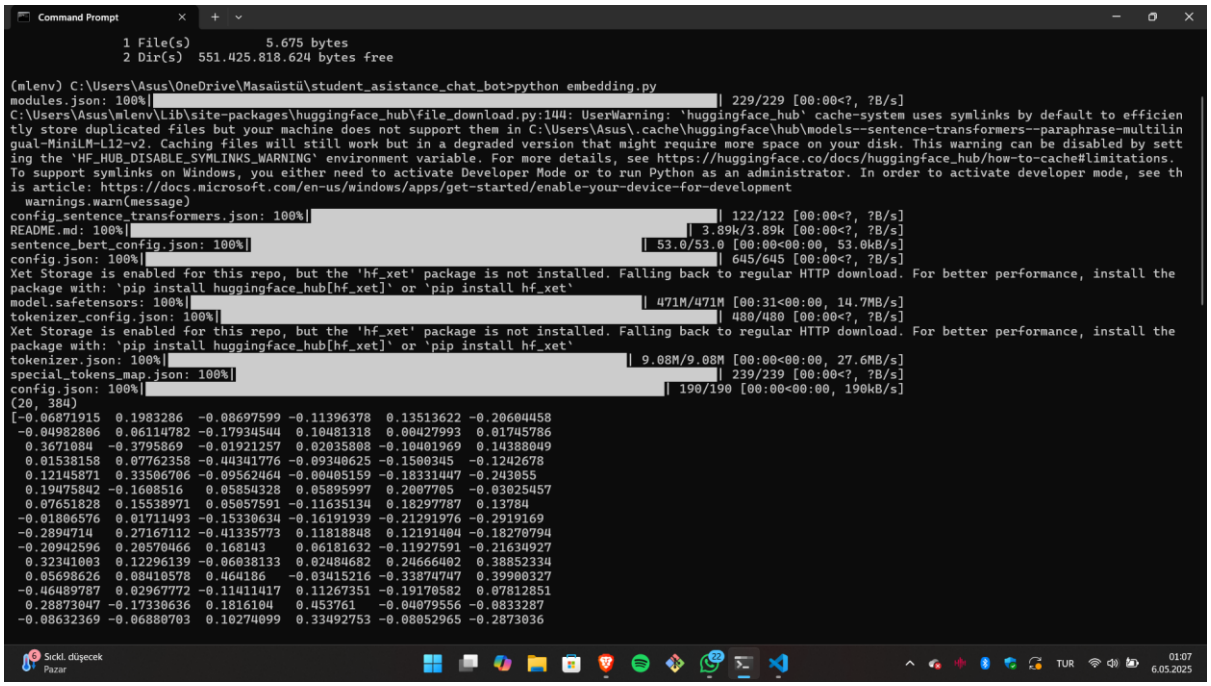
Bu aşamada embedding işlemi sorunsuz çalıştı ancak her çalıştırmada yeniden embedding hesaplamanın maliyetli ve zaman alıcı olduğunu fark ettim. Bu nedenle, bir defa oluşturduğum embedding'leri kaydedip gerektiğinde tekrar yükleyerek kullanabilmek istedim. Bunun için `NumPy`'nin `.npy` formatını ve Python'un `pickle` modülünü kullandım. `embedding` vektörlerini `.npy` dosyasına, soru metinlerini ise `pickle` dosyasına kaydettim.

Sonraki araştırmamda bu embedding vektörleriyle benzerlik karşılaştırmaları yapmanın birkaç yolu olduğunu öğrendim. Bunlar arasında FAISS ve ChromaDB gibi araçlar yer alıyordu. İlk olarak ChromaDB'yi denemek istedim çünkü daha basit görünüyordu; fakat ortamımda çalışmadığı için FAISS'e yöneldim. FAISS, daha karmaşık bir yapı gibi görünmesine rağmen, vektörler için yüksek performanslı bir indeksleme ve arama imkânı sunuyor.

FAISS kullanırken, öncelikle sistemde daha önce oluşturulmuş embedding, FAISS index ve metadata dosyalarının varlığını `os.path.exists` yöntemiyle kontrol eden bir `if-else` yapısı kurdum. Eğer bu dosyalar mevcut değilse, embedding hesaplaması yapılır ve dosyalar oluşturulur; mevcutsa doğrudan bu dosyalar yüklenerek sistem başlatılır.

Bu işlem sırasında embedding'ler `.npy`, indeks `.index` ve metadata ise `.pkl` formatında kaydediliyor.

Son olarak basit bir query ("Eksik veriler nasıl doldurulur?") cümlesi ile FAISS arama fonksiyonunu test ettim. Sistem, bu sorguya en çok benzeyen soruları başarıyla getirerek doğru çalıştığını gösterdi. Aldığım çıktıları aşağıya ekliyorum:




```
(mlenv) C:\Users\Asus\OneDrive\Masaüstü\student_asistance_chat_bot>python embedding_chromadb.py
✅ Kod başladı.
📥 Embedding'ler FAISS'e eklendi.
🔍 En yakın sonuçlar:
- Eksik verilerle başa çıkmak için hangi yöntemleri kullanabilirim? (ID: 1)
- Sürekli veri nedir ve nasıl işlenir? (ID: 12)
- Veri kümesindeki aykırı değerleri nasıl tespit edebilirim? (ID: 2)
```

```
(base) PS C:\Users\Asus\OneDrive\Masaüstü\student_asistance_chat_bot> python embedding_chromadb.py
✅ Kod başladı.
🆕 İlk çalıştırma: veriler oluşturuluyor ve kaydediliyor...
✅ Embedding'ler ve index kaydedildi.

🔍 En yakın sonuçlar:
- Eksik verilerle başa çıkmak için hangi yöntemleri kullanabilirim? (ID: 1)
- Sürekli veri nedir ve nasıl işlenir? (ID: 12)
- Veri kümesindeki aykırı değerleri nasıl tespit edebilirim? (ID: 2)
(base) PS C:\Users\Asus\OneDrive\Masaüstü\student_asistance_chat_bot> python embedding_chromadb.py
✅ Kod başladı.
📂 Kayıtlı veriler bulundu, yükleniyor...

🔍 En yakın sonuçlar:
- Eksik verilerle başa çıkmak için hangi yöntemleri kullanabilirim? (ID: 1)
- Sürekli veri nedir ve nasıl işlenir? (ID: 12)
- Veri kümesindeki aykırı değerleri nasıl tespit edebilirim? (ID: 2)
```