

## İlk Kurulum

İlk olarak embedding yapacağım özelliğimi seçtim:

TokenType-> **exampleUsecase**

Github dosyamda Week 6 olarak tüm yolu ödev için 6. haftaya yüklüyorum. tokenType\_openai\_embedding.py içinde python ile openai embedding modelini kullanarak tokentype düğümüne ait exampleUsecase özelliğim için embedding uyguladım. Yorum satırıyla açıklamalarımı yaptım .

Daha sonra csv dosyamı Neo4j Aura' ya yüklüyorum.

GitHub CSV bağlantımı "raw" formatına çeviriyorum.:

<https://raw.githubusercontent.com/jaguuai/Homeworks/main/Week6/data/tokenType-exampleUsecase-embeddings.csv> . Daha sonrada yeni usecaseEmbedding özelliğimi TokenType düğümüne ekliyorum.

```
1 LOAD CSV WITH HEADERS
2 FROM 'https://raw.githubusercontent.com/jaguuai/Homeworks/main/Week6/data/tokenType-exampleUsecase-embeddings.csv'
3 AS row
4
5 MATCH (tt:TokenType {tokenId: toInteger(row.tokenTypeId)})
6
7 CALL db.create.setNodeVectorProperty(
8   tt,
9   'usecaseEmbedding',
10  apoc.convert.fromJsonList(row.embedding)
11 )
12
13 RETURN count(*)
14
```

Table RAW

count(\*)

1 4

Started stream

TokenType

TokenType

Properties Neighbors Relationships

Edit

TokenType

exampleUsecase	Backed by blockchain projects or real-world digital assets
tokenType	Blockchain Token
tokenId	4
usecaseEmbedding	-0.019299078732728958, -0.013489321805536747, -0.026417700573801994, -0.01708202250301838, -0.007626141421496868

Dahas sonra vektör indeks oluşturuyoruz. Tokentype için “tokenTypeExampleUcecaseIndex” adında index oluşturdum. IF NOT EXISTS ile aynı isimde avrşa tekrar olusturmasını engellliyorum. OpenAI'nin text-embedding-ada-002 modeli 1536 boyutlu vektör üretir.Bu yüzden-> vector.dimensions: 1536. cosine similarity ile benzerlik aramak için-> vector.similarity\_function: 'cosine'. Alternatif olarak euclidean veya dot da kullanılabilir ama cosine çoğu dil uygulaması için en uygundur.

```
1 CREATE VECTOR INDEX tokenTypeExampleUcecaseIndex IF NOT EXISTS
2 FOR (tt:TokenType)
3 ON tt.usecaseEmbedding
4 OPTIONS {indexConfig: {
5   'vector.dimensions': 1536,
6   'vector.similarity_function': 'cosine'
7 }}
```

Added 1 index

TokenType node'ları üzerinde, 1536 boyutlu usecaseEmbedding vektörlerini kullanarak cosine similarity ile benzerlik aramak için bir index oluşturun.

Oluşturduğum indeksin durumunu kontrol ediyorum:

```
neo4j$ SHOW INDEXES YIELD id, name, type, state, populationPercent WHERE type = "VECTOR"
```

id	name	type	state	populationPercent
20	"tokenTypeExampleUcecaseIndex"	"VECTOR"	"ONLINE"	100.0

Started stream

Neo4j'deki tokenTypeExampleUcecase adlı vektör index'i, sistem tarafından id: 20 olarak tanımlanmıştır. Bu index türü VECTOR olup, cosine benzerlik gibi vektör karşılaştırmalarında kullanılır. Şu anda state: ONLINE durumundadır, yani aktif ve sorgulara hazırdır. populationPercent: 100.0 değeri ise bu index'in veritabanındaki ilgili tüm TokenType düğümlerine başarıyla uygulanmış olduğunu gösterir. Bu sayede usecaseEmbedding alanı üzerinden benzerlik tabanlı aramalar yapılabilir.

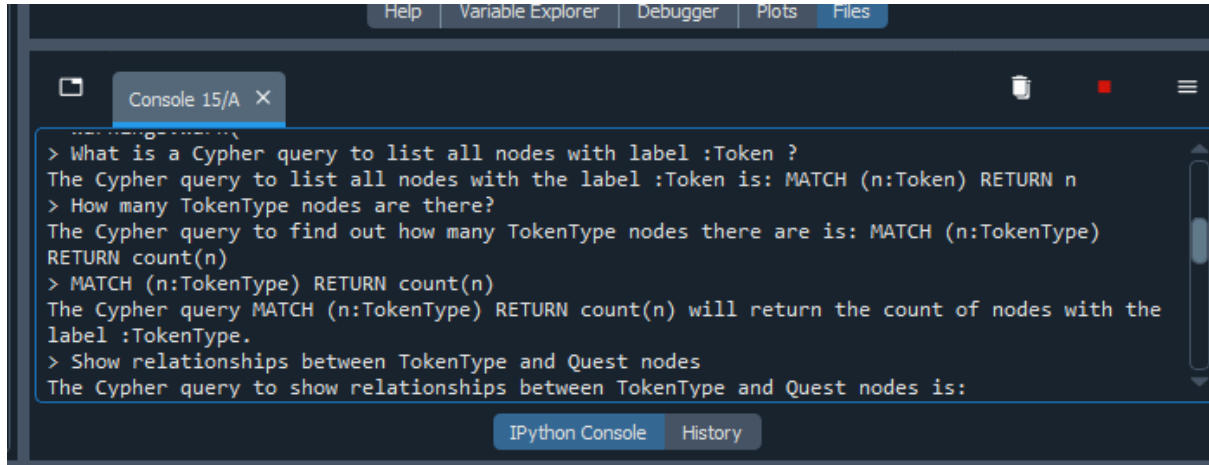
Vektör indeksimi kullanarak benzerlik araması yazıyorum. Normalde kendisi 1.0 gelmeliydi fakat ben vektörümü sadece 4 eleman üzerinden oluşturduğum için ufak fark var. Yine de en çok benzer kendisiyle olduğunu doğru şekilde gösteriyor.

```
1 MATCH (tt:TokenType {tokenType:"Security Token"})
2
3 CALL db.index.vector.queryNodes('tokenTypeExampleUcecaseIndex', 2,tt.usecaseEmbedding)
4 YIELD node, score
5
6 RETURN node.tokenType AS tokenType, node.exampleUcecase AS exampleUcecase, score
```

tokenType	exampleUcecase	score
"Security Token"	"Used as an investment tool in legal tech projects"	0.997161865234375
"Utility Token"	"Used as a payment tool in Web 3 projects"	0.930816650390625

Bu aşamaları deneyimledikten sonra örnek langchain dosyası üzerinden kurulum yapıyorum. Bu Python scripti, LangChain, Neo4j ve OpenAI GPT modellerini birleştirerek

etkileşimli bir Cypher asistanı oluşturur. Kullanıcı terminal üzerinden doğal dilde Neo4j'e dair sorular sorabilir (örneğin: "Show me all TokenTypes"), bu girişler LLM (GPT-3.5 veya GPT-4) tarafından işlenip uygun Cypher sorgularına dönüştürülür. Sohbet geçmişi ise Neo4j veritabanında saklanır. Bu sayede kullanıcı hem doğal dilde sorgular yazabilir, hem de Neo4j'e dair geçmişe bağlı, akıllı yanıtlar alabilir. Kod, "agent" ve "tool" mimarisiyle modüler şekilde çalışır ve oturum bazlı belleği destekler. SpyderIDE ile kodu çalıştırıp birkaç soru sordum. Yorum satırlarıyla beraber github Week6 ödev klasörüne `introduction_langchain.py` adıyla yükledim.



```
Help | Variable Explorer | Debugger | Plots | Files
Console 15/A X
> What is a Cypher query to list all nodes with label :Token ?
The Cypher query to list all nodes with the label :Token is: MATCH (n:Token) RETURN n
> How many TokenType nodes are there?
The Cypher query to find out how many TokenType nodes there are is: MATCH (n:TokenType)
RETURN count(n)
> MATCH (n:TokenType) RETURN count(n)
The Cypher query MATCH (n:TokenType) RETURN count(n) will return the count of nodes with the
label :TokenType.
> Show relationships between TokenType and Quest nodes
The Cypher query to show relationships between TokenType and Quest nodes is:
IPython Console History
```

**`pip install langchain` :LangChain'in ana çekirdek kütüphanesini yükler.**

- LLM (GPT, Claude, vs.) modelleriyle konuşabiliriz
- Agent, tool, chain gibi modülleri kullanabiliriz
- Prompt yönetimi, zincirleme işlem tanımı yapılabilir

**`pip install langchain-community langchain-neo4j langchainhub neo4j`**

<code>langchain-</code> <code>community</code>	LangChain'in <b>topluluk destekli tüm modül entegrasyonlarını</b> içerir (örnek: Redis, Pinecone, HuggingFace, vs.)
<code>langchain-neo4j</code>	Neo4j ile <b>LangChain arasında bağlantı</b> kurar. Sohbet geçmişi ve veri sorguları için kullanılır.
<code>langchainhub</code>	LangChain prompt şablonlarını indirebilmen için bir arayüz sunar (örneğin: <code>hub.pull("hwchase17/react-chat")</code> )
<code>neo4j</code>	Python'dan Neo4j veritabanına bağlanmanı sağlar ( <code>GraphDatabase.driver(...)</code> )

**`pip install openai langchain-openai`**

<code>openai</code>	<b>OpenAI API'sine</b> doğrudan bağlanmanı sağlar. (embedding, GPT modelleri, vs.)
---------------------	--

langchain-  
openai

OpenAI modellerini LangChain içinde **modül olarak kullanmanı sağlar.**