Wikipedia Game

ChatGPT Powered

By Max Miller and
Diego Lopez Ramos

What is Embedding

OpenAI's Embeddings are numerical representations of data, typically text, which capture semantic meaning so that similar concepts are positioned closer together in the embedding space. Essentially, an embedding is a way of converting text into a form that a machine can understand and process, preserving the nuanced relationships between words or sentences.

What is Spatial Distancing

Is the concept of measuring the distance between vectors in a multi-dimensional space. This distance reflects the semantic or contextual similarity between the data items (e.g., text) represented by these vectors. Understanding spatial distancing is crucial for many applications, such as information retrieval, recommendation systems, and clustering.

How Utilizing Both can be Beneficial

Semantic Search

By computing the distances or similarities between an input text vector and all other text vectors in a database, you can retrieve the most semantically related entries.

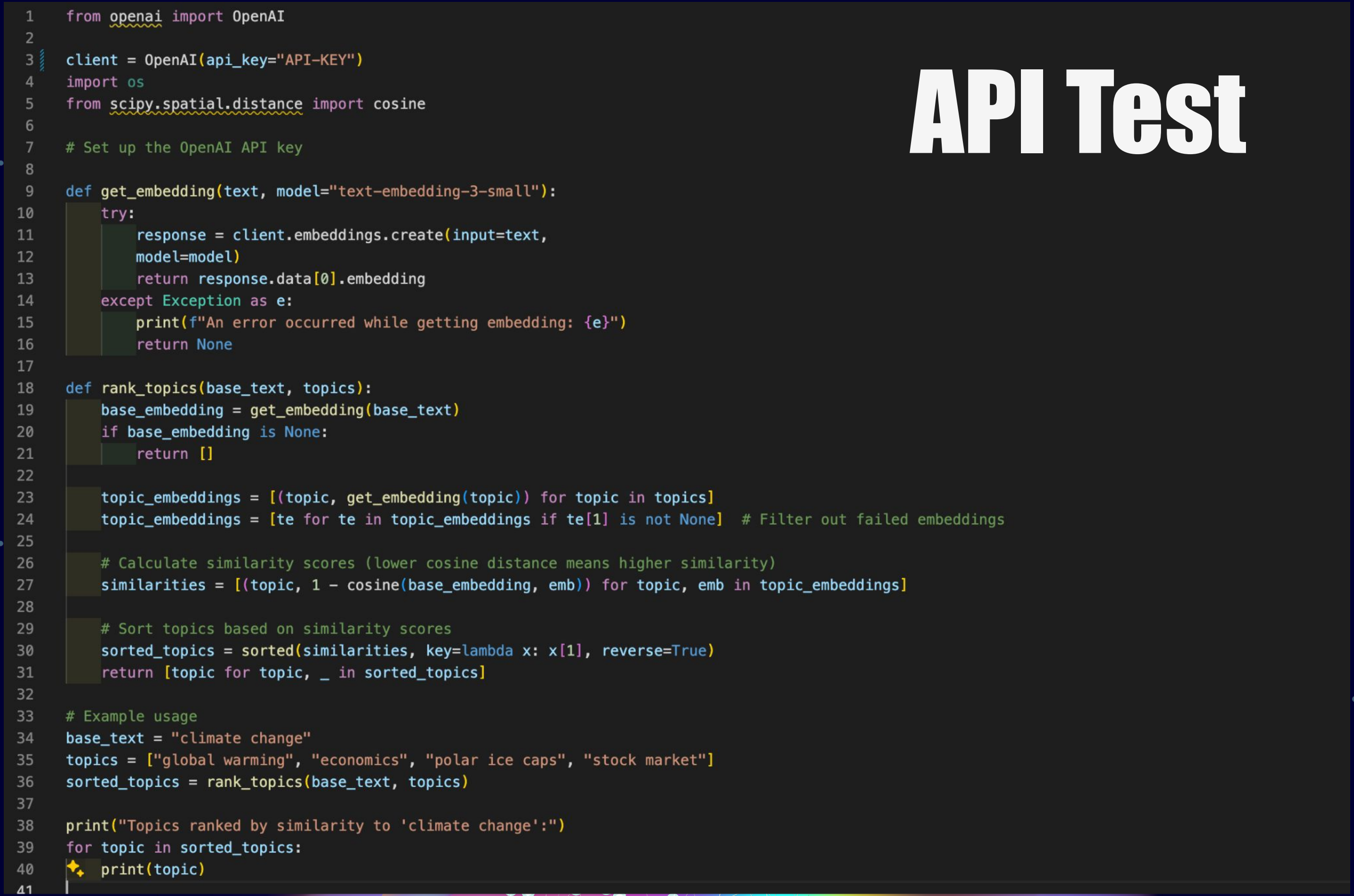
Content Recommendations

Similar to semantic search, by finding items (like articles, products, etc.) closest in the embedding space to a user's interests or past behaviors, systems can recommend new, relevant items.

Clustering

Groups of similar items can be formed by using clustering algorithms like k-means, which group data points (embeddings) that are close to each other in the embedding space.

API Test



```python
from openai import OpenAI

client = OpenAI(api_key="API-KEY")
import os
from scipy.spatial.distance import cosine

# Set up the OpenAI API key

def get_embedding(text, model="text-embedding-3-small"):
    try:
        response = client.embeddings.create(input=text,
        model=model)
        return response.data[0].embedding
    except Exception as e:
        print(f"An error occurred while getting embedding: {e}")
        return None

def rank_topics(base_text, topics):
    base_embedding = get_embedding(base_text)
    if base_embedding is None:
        return []

    topic_embeddings = [(topic, get_embedding(topic)) for topic in topics]
    topic_embeddings = [te for te in topic_embeddings if te[1] is not None]  # Filter out failed embeddings

    # Calculate similarity scores (lower cosine distance means higher similarity)
    similarities = [(topic, 1 - cosine(base_embedding, emb)) for topic, emb in topic_embeddings]

    # Sort topics based on similarity scores
    sorted_topics = sorted(similarities, key=lambda x: x[1], reverse=True)
    return [topic for topic, _ in sorted_topics]

# Example usage
base_text = "climate change"
topics = ["global warming", "economics", "polar ice caps", "stock market"]
sorted_topics = rank_topics(base_text, topics)

print("Topics ranked by similarity to 'climate change':")
for topic in sorted_topics:
    print(topic)
```

Code Explanation

What it does

- Embeddings are generated from both the Wikipedia text from the first page and the Wikipedia topic
- Calculates cosine similarity between topics and base text
- Next topic is chosen based on highest ranking similarity

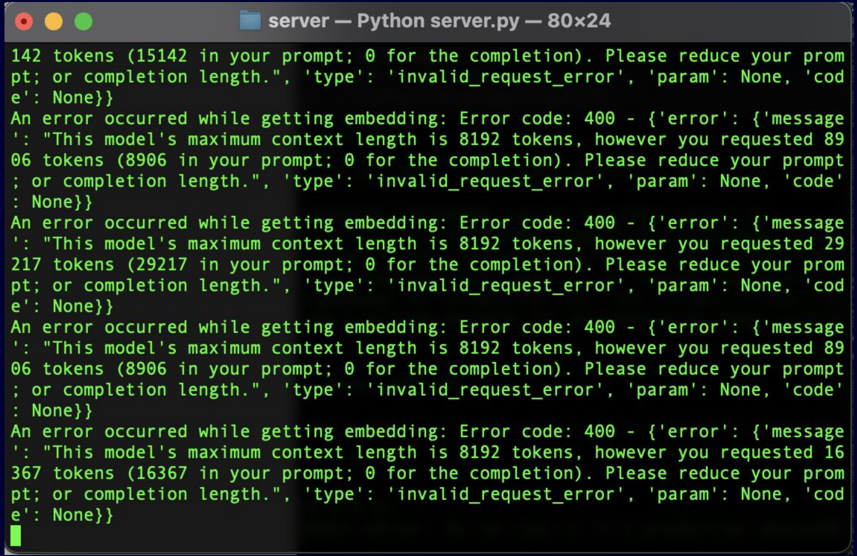
How it works

- get_embedding function uses OpenAI API to generate an embedding given text
- rank_topics creates a list of topics and calculates cosine similarity between base and topic text, then sorts topics by distance
- get_links finds the next link using rank_topics

Roadblocks

- Implementing rank_topics()
- Max token length limitations
- Getting stuck

  at the beginning



```
142 tokens (15142 in your prompt; 0 for the completion). Please reduce your prom
pt; or completion length.", 'type': 'invalid_request_error', 'param': None, 'cod
e': None}}
An error occurred while getting embedding: Error code: 400 - {'error': {'message
': "This model's maximum context length is 8192 tokens, however you requested 89
06 tokens (8906 in your prompt; 0 for the completion). Please reduce your prompt
; or completion length.", 'type': 'invalid_request_error', 'param': None, 'code'
: None}}
An error occurred while getting embedding: Error code: 400 - {'error': {'message
': "This model's maximum context length is 8192 tokens, however you requested 29
217 tokens (29217 in your prompt; 0 for the completion). Please reduce your prom
pt; or completion length.", 'type': 'invalid_request_error', 'param': None, 'cod
e': None}}
An error occurred while getting embedding: Error code: 400 - {'error': {'message
': "This model's maximum context length is 8192 tokens, however you requested 89
06 tokens (8906 in your prompt; 0 for the completion). Please reduce your prompt
; or completion length.", 'type': 'invalid_request_error', 'param': None, 'code'
: None}}
An error occurred while getting embedding: Error code: 400 - {'error': {'message
': "This model's maximum context length is 8192 tokens, however you requested 16
367 tokens (16367 in your prompt; 0 for the completion). Please reduce your prom
pt; or completion length.", 'type': 'invalid_request_error', 'param': None, 'cod
e': None}}
```

What’s next?

- Implement Sorting
- Adding backtracking
- Experimenting with conversational models of OpenAI (ChatGPT 4)

Thanks for listening!