Building a Semantic Search Engine with Text Embeddings for Clothing Products

1. Objective

- Learn to generate text embeddings using Azure OpenAI's **"text-embedding-3-small"** model.
- Build a semantic search engine that finds the most similar clothes based on product descriptions.
- Use cosine distance to measure similarity between embeddings.
- Practice handling embeddings and performing vector similarity search in Python.

2. Problem Statement

- Online stores require effective search and recommendation systems to help customers find relevant products.
- Traditional keyword matching often fails to capture semantic similarities.
- This exercise guides you to build a semantic search engine by embedding product descriptions and user queries as vectors and finding the closest matches.

3. Inputs / Shared Artifacts

- A sample dataset of clothing products (generated within your script) with fields such as:
    - `title`
    - `short_description`
    - `price`
    - `Category`
- Azure OpenAI Resource:

    - API key, endpoint URL, and deployment name (to be set as environment variables).
- Example snippet of generated data:

```
products = [
    {
        "title": "Classic Blue Jeans",
        "short_description": "Comfortable blue denim jeans with a
relaxed fit.",
        "price": 49.99,
        "category": "Jeans"
    },
    {
        "title": "Red Hoodie",
        "short_description": "Cozy red hoodie made from organic
cotton.",
        "price": 39.99,
        "category": "Hoodies"
    },
    # ... add more products
]
```

- Python environment with OpenAI Python client installed (`openai` or `OpenAI`).
- `scipy` or `numpy` for cosine similarity calculation (`scipy.spatial.distance.cosine`).

4. Expected Outcome

- A Python script that:
    - Creates embeddings for all products' descriptions using Azure OpenAI's **"text-embedding-3-small"** model.
    - Accepts a user input query (product description or search text).
    - Generates an embedding for the input query.
    - Computes cosine distances between the query embedding and product embeddings.
    - Returns and prints the top N most similar products ranked by similarity.

5. Concepts Covered

- Using Azure OpenAI's text embedding models for semantic understanding.
- Handling batched embedding requests (if implemented).
- Computing cosine similarity to find nearest neighbors in vector space.
- Basic data structuring and search logic for recommendation engines.
- Integration with Azure OpenAI API via official Python client.

6. Example: Step-by-Step Instructions with Code

```
from openai import AzureOpenAI

from scipy.spatial.distance import cosine

# Step 1: Setup AzureOpenAI
client = AzureOpenAI(
    api_version="2024-07-01-preview",
    azure_endpoint=os.getenv("AZURE_OPENAI_ENDPOINT"),
    api_key=os.getenv("AZURE_OPENAI_API_KEY"),
)



# Step 2: Sample product data
products = [
    {
        "title": "Classic Blue Jeans",
        "short_description": "Comfortable blue denim jeans with a
relaxed fit.",
        "price": 49.99,
        "category": "Jeans"
    },
    {
        "title": "Red Hoodie",
        "short_description": "Cozy red hoodie made from organic
cotton.",
        "price": 39.99,
```

```
        "category": "Hoodies"
    },
    {
        "title": "Black Leather Jacket",
        "short_description": "Stylish black leather jacket with a slim
fit design.",
        "price": 120.00,
        "category": "Jackets"
    },
    # Add more products as needed
]

# Step 3: Function to get embeddings from Azure OpenAI
def get_embedding(text):
    response = client.embeddings.create(
        model="text-embedding-3-small",
        input=text
    )
    embedding = response.data[0].embedding
    return embedding

# Step 4: Generate embeddings for all product descriptions
for product in products:
    product["embedding"] = get_embedding(product["short_description"])

# Step 5: Accept user input (query)
query = "warm cotton sweatshirt"

# Step 6: Get embedding for the user query
query_embedding = get_embedding(query)

# Step 7: Compute cosine similarity between query and each product
def similarity_score(vec1, vec2):
    return 1 - cosine(vec1, vec2)  # cosine returns distance; 1 -
distance = similarity

scores = []
for product in products:
```

```
    score = similarity_score(query_embedding, product["embedding"])
    scores.append((score, product))

# Step 8: Sort products by similarity descending
scores.sort(key=lambda x: x[0], reverse=True)

# Step 9: Display top matches
print(f"Top matching products for query: '{query}'\n")
for score, product in scores[:3]:  # top 3 results
    print(f"Title: {product['title']}")
    print(f"Description: {product['short_description']}")
    print(f"Price: ${product['price']}")
    print(f"Category: {product['category']}")
    print(f"Similarity Score: {score:.4f}\n")
```

7. Final Submission Checklist

- Submit your complete Python script with clear comments.
- Include your sample product data embedded in the script.
- Write a brief README or report including:
    - How embeddings were created and used.
    - Explanation of cosine similarity for ranking.
    - Any challenges or limitations encountered.