Lab Summary: Getting Started with Vector Search and Embeddings

Lab Info

Title: Getting Started with Vector Search and Embeddings
Learning Path: Advanced: Generative AI for Developers
Platform: Google Cloud Skills Boost
Duration: 1 hour 30 minutes

Lab Objectives

- Create a Vertex AI Notebook instance and run the lab notebook.
- Generate text embeddings using a pre-trained model.
- Upload the embeddings to Google Cloud Storage.
- Create and deploy a vector search index using Matching Engine.
- Run semantic queries to retrieve the most relevant results.

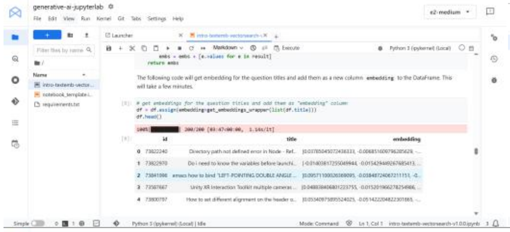
Lab Content (Detailed)

Task 1 – Open the notebook in Vertex AI Workbench

I began by launching a Vertex AI Workbench environment and cloning the provided lab notebook.
This step was essential to execute the code and interact with the cloud-based embedding and vector tools.

Task 2 – Generate embeddings

I used Vertex AI's Embeddings model to convert question titles into numerical representations (embeddings).
Then I saved the embeddings in a JSON file and uploaded it to a newly created Google Cloud Storage bucket.
This process involved running predefined cells in the notebook and using Python to assign the embeddings column to the dataframe.



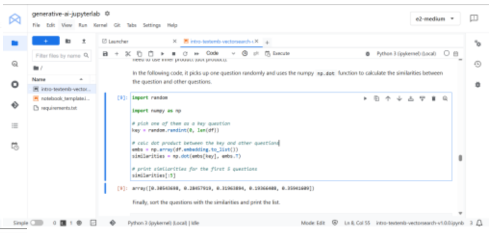
```
        embs = embs + [e.values for e in result]
    return embs
```

The following code will get embedding for the question titles and add them as a new column `embedding` to the DataFrame. This will take a few minutes.

```
# get embeddings for the question titles and add them as "embedding" column
df = df.assign(embedding=get_embeddings_wrapper(list(df.title)))
df.head()
```

```
100%|██████████| 200/200 [03:47<00:00,  1.14s/it]
```

```
         id                                            title                                            embedding
0  73822240      Directory path not defined error in Node - Ref...  [0.037050450724383333, -0.008851600796285629, -...
1  73822970  Do i need to know the variables before launchi...   [-0.014038172550499944, -0.015429449267685413, ...
2  73841996  emacs how to bind "LEFT-POINTING DOUBLE ANGLE ...  [0.09571100026369095, -0.038487240672111151, -0...
3  73587667     Unity XR Interaction Toolkit multiple cameras ...  [0.048838406801223755, -0.015201966278254986, ...
4  73800797     How to set different alignment on the header o...  [0.05534087589555240225, -0.051422204822301665, -...
```

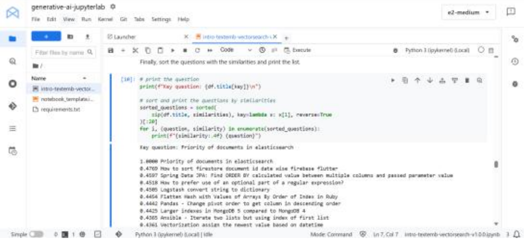


In the following code, it picks up one question randomly and uses the numpy `np.dot` function to calculate the similarities between the question and other questions.

```
import random

import numpy as np

# pick one of them as a key question
key = random.randint(0, len(df))

# calc dot product between the key and other questions
embs = np.array(df.embedding.to_list())
similarities = np.dot(embs[key], embs.T)

# print similarities for the first 5 questions
similarities[:5]
```

```
array([0.30563608, 0.28457919, 0.31963804, 0.19366408, 0.35941609])
```

Finally, sort the questions with the similarities and print the list.



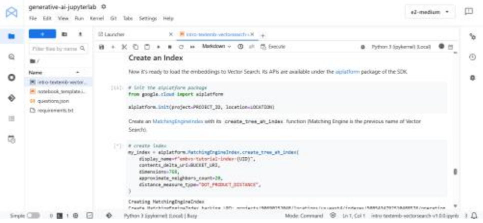
Finally, sort the questions with the similarities and print the list.

```
# print the question
print(f"Key question: {df.title[key]}\n")

# sort and print the questions by similarities
sorted_questions = sorted(
    zip(df.title, similarities), key=lambda x: x[1], reverse=True
)[:20]
for i, (question, similarity) in enumerate(sorted_questions):
    print(f"{similarity:.4f} {question}")
```

```
Key question: Priority of documents in elasticsearch

1.0000 Priority of documents in elasticsearch
0.4769 How to sort firestore document id date wise firebase flutter
0.4597 Spring Data JPA: Find ORDER BY calculated value between multiple columns and passed parameter value
0.4518 How to prefer use of an optional part of a regular expression?
0.4505 Logstash convert string to dictionary
0.4454 Flatten Hash with Values of Arrays By Order of Index in Ruby
0.4442 Pandas - Change pivot order to get column in descending order
0.4425 Larger indexes in MongoDB 5 compared to MongoDB 4
0.4365 Ansible - Iterate two lists but using index of first list
0.4361 Vectorization assign the newest value based on datetime
```

Task 3 – Create and deploy an index

Using the `aiplatform` library, I created a Matching Engine Index by calling the `create_tree_ah_index` function.
Key parameters like dimensions (768), distance measure (DOT_PRODUCT_DISTANCE), and approximate neighbors count (20) were configured.
The deployment process took about 20–30 minutes to complete.



Create an Index

Now it's ready to load the embeddings to Vector Search. Its APIs are available under the aiplatform package of the SDK.

```
# init the aiplatform package
from google.cloud import aiplatform

aiplatform.init(project=PROJECT_ID, location=LOCATION)
```

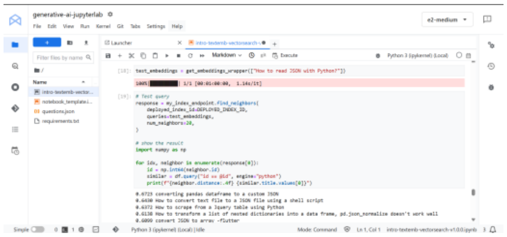
Create an MatchingEngineIndex with its `create_tree_ah_index` function (Matching Engine is the previous name of Vector Search).

```
# create index
my_index = aiplatform.MatchingEngineIndex.create_tree_ah_index(
    display_name=f"embvs-tutorial-index-{UID}",
    contents_delta_uri=BUCKET_URI,
    dimensions=768,
    approximate_neighbors_count=20,
    distance_measure_type="DOT_PRODUCT_DISTANCE",
)
```

```
Creating MatchingEngineIndex
```

Task 4 – Run a query

I randomly selected a question from the dataset to act as a key query, then computed the similarity between it and other questions using NumPy's dot product.
Finally, I sorted and printed the questions based on similarity scores, showing the closest semantic matches.
This highlighted how vector search can go beyond keyword matching by understanding the meaning behind text.



```
test_embeddings = get_embeddings_wrapper(["How to read JSON with Python?"])
```

```
100%|██████████| 1/1 [00:01<00:00,  1.14s/it]
```

```
# Test query
response = my_index_endpoint.find_neighbors(
    deployed_index_id=DEPLOYED_INDEX_ID,
    queries=test_embeddings,
    num_neighbors=20,
)

# show the result
import numpy as np

for idx, neighbor in enumerate(response[0]):
    id = np.int64(neighbor.id)
    similar = df.query("id == @id", engine="python")
    print(f"{neighbor.distance:.4f} {similar.title.values[0]}")
```

```
0.6723 converting pandas dataframe to a custom JSON
0.6630 How to convert text file to a JSON file using a shell script
0.6372 How to scrape from a Jquery table using Python
0.6138 How to transform a list of nested dictionaries into a data frame, pd.json_normalize doesn't work well
0.6099 convert JSON to array -flutter
```

What I Learned

- I learned how to generate embeddings from text and represent data semantically.
- I used Google Cloud Storage to manage and access my JSON files in the notebook.
- I successfully created and deployed a vector search index with Google’s Matching Engine.
- I practiced querying the index using similarity scores to retrieve semantically relevant results.
- I gained practical experience in building intelligent search systems without writing APIs or backend code.
- This lab enhanced my understanding of how vector search can be applied in real-world AI applications such as chatbots, recommendation engines, and semantic retrieval.