ETIYA

Cognitus Team Artificial Intelligence Model Assignment

Exceed Everday

ETIYA

www.etiya.com

ETIYA

A short brief...

Efficient data processing, large language models (LLMs), generative AI, clustering, and semantic search have become more crucial than ever for various applications. All of these emerging applications rely on vector embeddings.

An embedding is a list of floating-point numbers (a vector) that captures semantic information related to the text it represents. The distance between two vectors measures their relationship, where small distances indicate high similarity and large distances indicate low similarity. For instance, semantic similarity can be determined using cosine similarity.

For instance, when a text is passed to the **all-MiniLM-L6-v2** model of the **Sentence Transformers** library, the embedding of this sentence is represented by a list of 384 numbers in a vector space (for example, [0.84, 0.42, ..., 0.02]). This is because this model maps the data into a 384-dimensional dense vector space.

These embeddings can be costly in terms of both time and computation. Therefore, there is a need for a specialized database to handle such data. A vector database indexes and stores vector embeddings for fast retrieval and similarity search, equipped with capabilities such as CRUD (Create, Read, Update, Delete) operations, metadata filtering, and horizontal scaling. This enables efficient management of vector embeddings, facilitating quick retrieval and similarity search.

Some of the popular vector databases are:

Qdrant, Pinecone, Chroma, Weaviate

Here we will focus on library ***Qdrant***. Let's get started.

Task

In order to understand your skills in vector database, we want you to extract the embeddings of the data given to you and add it to the Qdrant vector store. Then, we expect you to search and bring the closest points related to the given query in the vector store.

Dataset

You can work on any sentiment data you find on HuggingFace**.** Usually, this data consists of **sentence** and **label** variables.

So, how do you do this?

The addresses suggested below will generally be sufficient to complete this process:

- https://github.com/qdrant/qdrant
- https://github.com/qdrant/qdrant-client
- https://github.com/qdrant/fastembed
- https://github.com/qdrant/examples
- https://qdrant.tech/
- https://qdrant.tech/documentation/concepts/
- https://qdrant.tech/documentation/tutorials/

(You can complete the case flows by examining end-to-end code examples in existing tutorials.)

1. Convert your data to parquet format and perform all your operations through this parquet file.
2. If necessary, perform preprocessing (remove stopwords, remove punctuation, to lowercase, spellcheck etc.) on this data.
3. Install the requirements of the relevant library in your environment.
4. The easiest way to start using Qdrant is to run the Qdrant official container image. The latest versions are always available on DockerHub.
5. Create a qdrant client using ***QdrantClient*** *class*. To do this you need to set the parameters (host, port, prefer_grpc, grpc_port etc).
6. Create a collection using client's **create collection** function. With the Qdrant API, you can check the current status of the collection and create it if it does not exist, if it exists; you can delete and recreate it or you can keep going. In addition, think about **quantization** approach, but you do not implement it.
7. In the next stage, you should extract the embeddings of the given data through a transformer model and put them into the vector store. You can follow these steps:
    - In order to prevent large data from taking up an enormous amount of space on RAM and stopping the environment from running, that is, from causing the ram to crash, you should create a *generator* that will read the parquet file piece by piece, for example the batch size could be 128 or 256.

        (***hint***: *pyarrow.parquet; iter batches; batch to pandas; yield)*

- On each iteration of the loop, use the client's **upsert** function. This function takes the following parameters:

  ➔ Give any **collection name**.

  ➔ Give the **batch size**.

  ➔ You should give the **ids** as range.

  ➔ Think about the **shard key selector** parameter, but you do not implement it. It is important for this case study that you understand the logic. This approach is so important for us.

  ➔ Add data labels as **payload** this is called as **metadata**. In this way, since the label of each vector will be known, vectors can be filtered according to these labels. Additionally, think about client's **create payload index** function, but you do not implement it.

  ➔ The last parameter is **vectors**. You need a transformer model to extract vectors. You can use the SentenceTransformer library to do that. Example embedding model should be all-MiniLM-L6-v2. Vectors are uploaded to the vector store according to the model's name. Since this is an overlooked situation, you can use the code below as a hint:

```
>>>
        vectors = {
          embedding_model_name: [
            arr.tolist()
            for arr in model.encode(
              sentences=data, # piece of data on each iteration
              batch_size=batch_size,
              normalize_embeddings=True,
            )
          ]
        }
```

  Think about fastembed, but you do not implement it. It is important for speed and memory. This approach is also important for us. For detailed information click here.

8. Yes, well done! You have almost finished the case! We only have one more little task remaining:

- ✘ Imagine you have a query (sentence). You want to to search and bring the closest points related to the given query in the vector store. For instance, if you are working on sentiment data, this will be the closest points to the given query, such as positive, negative and neutral vectors.
- ✘ Do this by using the client's **search** function. Provide the necessary parameters such as *collection name*, *query vector*, *query filter*, *limit*, *score threshold*. Remember that you need to convert the given query into a vector with the transformer model and give it to the query vector parameter with the **models.NamedVector(name, vector)** class.
- ✘ Also with the *query filter* parameter, return only the points with the payload you want on an example, for instance, only positive points or neutral points.
- ✘ Use *limit* and *score threshold* parameters on an example.
- ✘ Show all of these on examples using the **recommend** function. Do not forget to use *positive*, *negative* and *using* parameters.
- ✘ Think about **search batch** and **recommend batch** functions, but you do not implement it.

Expected Results

- ✘ Make sure that the codes you write comply with *PEP8*, *Flake8* and code standards. For example, sorting imports, giving type hints, writing docstrings, writing return types, etc.
- ✘ Do not use the client's **add** function to upload vectors into qdrant store. Especially, use client's **create collection** and **upsert functions**.

  (***hint***: *the source code client's add function)*
- ✘ If you use constant values in your codes, please create a **constants.py** file.
- ✘ Make your project a python package. Create a **pyproject.toml** file and make your package settings through this file (do not forget to put the relevant libraries here as well). Create **.gitignore** and **.env** files and make the necessary settings.
- ✘ Share your work with us through a GitHub repo, but It is important that you commit your codes and send them to the GitHub repo as you develop. We do not prefer you to send your work all at once at the end of the case.
- ✘ Give instructions on how to run your code if anything else is needed.

ETIYA

Notes

- ✘ You must implement your solution using Python 3.11.8+
- ✘ You are free to choose any library of frameworks and any editor.
- ✘ If you are not familiar with Python code standards, please do short research and remember that you can solve these problems with some plugins available on VS Code (Ruff plugin may be a good choice for all of them).
- ✘ Please write your codes clean, legible and understandable. Manage your functions or classes through separate modules (suitable for micro service architecture).
- ✘ It is a plus to use the pydantic library for validation of some parameters.
- ✘ Logging is a plus. You can use the loguru library.
- ✘ You do not have to do everything that is asked of you. What is important is what you can do in a short time on a subject you do not know.
- ✘ If you feel comfortable, we recommend you use English everywhere. If there is an interview about your work after you complete your work, it will be in Turkish.

You have **7 DAYS** to fulfill the assignment. Please, do not hesitate to contact us (**uguray.durdu@etiya.com**) in case of any questions or suggestions.

We will support you in the points where you are stuck and cannot solve it even if you try, we do not recommend you to get help from anyone because we will ask you to tell us about the improvements you have made at the end of the given time and we will ask some questions about them. Please!

- ✘ We do not recommend that you spend any effort more than 7 days on this study.
- ✘ Don't try to do it perfectly.
- ✘ Don't waste your time dealing with unwanted and extra things.
- ✘ Don't put yourself under stress.
- ✘ Try to understand the logic of the things you do, do not memorize.
- ✘ Please, do not share this case study on any platform, do not put it in any GitHub repo!

Good luck :)