ON THE IMPEDANCE MISMATCH OF VECTOR DATA: VECTOR DATABASES Vs. RELATIONAL DATABASES

OBJECTIVES

In the history of databases, it is not the first time that a new model challenges the relational model. This, indeed, happened in the late 70s and early 80s when the object-oriented programming languages appeared. NOSQL questions the relational model as the unique solution to represent data in data management systems. We aim at learning that different data models yield different performance depending on the nature of the data (the closer the nature of the data to the internal data representation in the database, the better).

Specifically, in this lab we will exemplify the impedance mismatch on vector data by comparing the storage and processing of embeddings in a relational database and a vector database (native database for vector data).

We summarize the objectives as follows:

- Learn the concept of impedance mismatch
- Learn about the vector data impedance mismatch by experimenting with embeddings generated and processed in a relational and a vector database
- Understand the relevance of new data models and their role in closing the impedance mismatch gap in modern data applications

REQUIRED KNOWLEDGE

This laboratory session does not require any previous knowledge.
Your lab mates for this session will be those of the team creation 1 event. You have until the deadline specified in the event to create your own team. Otherwise, you will be randomly assigned to a team.

TOOLS

- PostgreSQL (and optionally, its Pgvector extension)
- Chroma
- A neural network embedding model: the most significant application of word embeddings is to encode words for use as input to complex neural networks that try to understand the meanings of entire sentences, or even paragraphs. One such class of networks are called *transformer neural networks*. Two famous transformer networks are BERT from Google and GPT3 from OpenAI. BERT now handles many Google searches. However, they require GPUs to run. A powerful yet lightweight alternative is all-MiniLM-L6-v2. You are suggested to consider for this lab this or any other alternative lightweight transformer model).
- A corpus: bookCorpus

ACTIVITIES TO DO FOR THIS LAB

Important: you may need to use a few new technologies in this lab. Please, carefully read the links provided and the additional material you may need to setup these technologies in your computer.

The programming language for this lab is Python.

- Get familiar with HuggingFace: https://huggingface.co/docs/hub/en/index
- Get familiar on how to develop with Transformers with HuggingFace: https://huggingface.co/docs/transformers/en/quicktour (you are recommended to stay with lightweight transformers such as all-MiniLM-L6-v2)
- Download the bookCorpus (available in HuggingFace) and generate different chunks from it (depending on the available memory of your computer). Do not try to do the lab with the complete corpus. It would be too slow. A reasonable amount of data would be around 10k sentences

Get familiar how to create embeddings from the bookCorpus with the chosen transformer.

<u>PostgreSQL</u>

- Setup PostgreSQL locally in your computer (https://www.postgresqltutorial.com/postgresql-getting-started/) and use a connector to tap into PostgreSQL from Python: e.g., psycopg2: https://www.postgresqltutorial.com/postgresql-python/connect/. You **CANNOT** use Pgvector for this part
- [P0] Load into PostgreSQL (using the datatypes that you find optimal for it) the bookCorpus chunks you created. Split them per sentences when loading them into PostgreSQL
- [P1] Create a Python script that connects to your database, generates the embeddings for each sentence, and store the embeddings
- [P2] For 10 different sentences (clearly identify them), compute the top-2 most similar sentences (among all other sentences) for each of them using two different distance metrics
- Use the time library in Python to measure the performance of the following aspects:
  - Compute the minimum, maximum, standard deviation and average time for storing the textual data
  - Compute the minimum, maximum, standard deviation and average time for storing the embeddings
  - Compute the minimum, maximum standard deviation and average time when computing the top-2 similar sentences for the 10 chosen sentences
- [PQ1] Then, answer the following questions:
  - Are the insertion times for text and embeddings stable? What are your conclusions on this matter?
  - Are the querying times stable when computing the similarities? Did you find differences between the two distance metrics you used? What are the conclusions on this matter?

    - Can you think of any available insertion method, data structure or indexing technique available in PostgreSQL (**Pgvector is still not allowed to be used yet**) that would improve the performance of these operators?

Chroma

- Setup Chroma and learn how to connect to it via Python: https://docs.trychroma.com/getting-started, https://docs.trychroma.com/#python
- [C0] Load into Chroma the same chunk of data you loaded into PostgreSQL. The same sentence split must be used
- [C1] Create a Python script that connects to your database, generates the embeddings for each sentence, and store the embeddings
- [C2] Repeat the same querying you did for PostgreSQL now for Chroma. That is, choose the same 10 different sentences (clearly identify them) and compute the top-2 most similar sentences (among all other sentences) for each of them using two different distance metrics (ideally, use the same distance metrics as you used in PostgreSQL)
- Repeat the measures you did for PostgreSQL but now for Chroma. That is:
    - Compute the minimum, maximum, standard deviation and average time for storing the textual data
    - Compute the minimum, maximum, standard deviation and average time for storing the embeddings
    - Compute the minimum, maximum standard deviation and average time when computing the top-2 similar sentences for the 10 chosen sentences
- [CQ1] Then, answer the following questions:
    - Are the insertion times for text and embeddings stable? What are your conclusions on this matter?
    - Are the querying times stable when computing the similarities? Did you find differences between the two distance metrics you used? Is it possible, in Chroma, to measure the insertion of text and embeddings creation separately? What are the conclusions on this matter?
- Can you think of any available insertion method, data structure or indexing technique available in Chroma that would improve the performance of these operators?

Pgvector

Optionally, you can repeat the experiments you did for PostgreSQL and Chroma with Pgvector (https://github.com/pgvector/pgvector). Install the PostgreSQL extension (a native extension for vector data in PostgreSQL) and measure the same times. What are the main differences between using Chroma and Pgvector? What are your conclusions? (i.e., pros and cons for each)

DELIVERABLES

You must deliver 1 document and 6 python scripts (9 if you do the optional part with Pgvector). Namely:

- An explanatory document,
- The PostgreSQL scripts: [P0], [P1] and [P2]
- The Chroma scripts: [C0], [C1] and [C2]
- The Pgvector scripts: [G0], [G1] and [G2]

The document must be 10 pages long (12 if you do the optional part), at most, and include the following information:

- A link to a public Github or repository where to find the chunk of data you used for your experiments,
- The document must be divided in 3 (4 if you do the optional part) parts:
    - PostgreSQL: explain the main decisions you made when using PostgreSQL (specially, from the impedance mismatch point of view). That is, explain any decision you made when creating the PostgreSQL scripts that may impact performance and the number of code lines and calls made. Then, answer the questions posed for [PQ1]
    - Chroma: explain the main decisions you made when using Chroma (specially, from the impedance mismatch point of view). That is, explain any decision you made when creating the Chroma scripts that may impact performance and the number of code lines and calls made. Then, answer the questions posed for [CQ1]
    - Pgvector: If you did the optional part, include yet another section for Pgvector with the same information.
    - Discussion: from an impedance mismatch perspective, what is the difference between using PostgreSQL and Chroma? What are the pros and cons for each system? If you used the optional part, include Pgvector in the discussion.

EVALUATION

The PostgreSQL and Chroma parts are evaluated, each, with 3p. The evaluation considers your code and explanation (in the document) about the impedance mismatch generated in each solution. The discussion between them both is 3p (4p if Pgvector is included). The optional part on Pgvector (its scripts and questions) is evaluated with 2p. Thus, without the optional part, you can get, at most, 9 out 10 in the lab mark. If you do the optional part you can score 12/10 (rounded up to 10 if your mark is higher).

In the evaluation, the rationale and the discussion about the impedance mismatch you carry out in the document has a high impact on the final mark. Being precise and concise in your answers is also a quality aspect to be considered.