Homework 1: Vector Search

Submission Instructions

1. Submission Date: The assignment is due on 02.07.24, at 22:55.
2. Team Submission: Each team should consist of 3 students, unless otherwise approved.
3. Submission File Format: Each team should submit a zip file named `<ID1_ID2_ID3>`, where `ID<i>` represents the ID number of each of the 3 team members. The zip file must include the following files:
   - `part1.ipynb`
   - `part2.ipynb`
   - `Report.pdf`

   Note that the report should include a link to a GitHub repository, as detailed in the instructions in Section 3.
4. Report Language: The report (PDF file) must be written in English.
5. The deliverables that should be attached to the report must appear under a clear title, indicating the section number.
6. Code Compliance: Ensure that your code adheres to the provided template.
7. Questions: Any questions should be posted in the Moodle forum, and must be asked in English.
8. It is highly recommended to use the GPU server assigned for you. Please follow the instructions provided in the files `Students - How to work with your VM.pdf` and `installations.txt`.
9. If you struggle with package installations, please notify us as soon as possible via the forum.

Assignment Objectives

- Understand the difference between naive search methods and Faiss using visualization.
- Use data exploration and analysis to identify scenarios where a standard index might not perform well and develop strategies to handle such cases.
- Gain hands-on experience with RAG and VectorDB.
- Practice modular code writing.

1 Faiss

Introduction

Faiss (Facebook AI Similarity Search) is a library developed by Facebook AI Research that excels in efficient similarity search. It is built upon an optimized indexed structure to perform both exact and approximate nearest neighbor computations. As datasets grow in size and dimensionality, the time it takes to retrieve the nearest neighbors becomes a crucial factor. By evaluating the running times of different algorithms, as well as evaluating their performance, we can identify which are more suitable for large-scale applications.

Follow the instructions provided in the file `Part1.ipynb`. You can use the helper functions attached to the notebook, and add functions on your own in the designated code cells. Write your code as readable and modular as possible. The file `Part1.ipynb` with your completed code should be attached to the submitted zip file.

1.1 Running Times:

The deliverables that should appear in the report are:

- (1.1.1) A plot displaying the running time of the three methods as a function of the number of vectors in the index.
- (1.1.2) A plot displaying the running time of the three methods as a function of the dimensionality of the vectors.

1.2 Faiss LSH:

The deliverables that should appear in the report are:

- (1.2.1) A plot displaying the running time of Faiss LSH as a function of the number of vectors in the index.
- (1.2.2) A plot displaying the running time of Faiss LSH as a function of nbits.
- (1.2.3) A plot displaying the recall@k of Faiss LSH as a function of nbits.

2 Implementing an Index

Introduction

In this part of the assignment, you are required to implement an index from scratch. Understanding the characteristics of the data is a crucial factor in developing and applying tailored solutions, hence it is recommended to explore the characteristics of the given index vectors.

Follow the instructions provided in the file `Part2.ipynb`. You can use the helper functions attached to the notebook, and add functions on your own in the designated code cells. Write your code as readable and modular as possible. The file `Part2.ipynb` with your completed code should be attached to the submitted zip file. In this part you will use the files provided in the data directory. Please extract and place this directory in your main notebook path.

2.1 LSH vs. Naive Exhaustive Search

Follow the instructions provided in the notebook. The deliverables that should appear in the report are:

1. Running time of the `semi_optimized_exhaustive_search` function, used for ground trurth computation (wall time).
2. Running time of creating `faiss_lsh_index` (wall time).
3. Running time of `faiss_search` over `query_vectors` with `faiss_lsh_index` (wall time).
4. Recall@10 for `faiss_lsh_index`. Not that impressive, isn't it?

2.2 Custom Indexing Algorithm

Follow the instructions provided in the notebook. The deliverables that should appear in the report are:

1. A brief explanation (at most single A4 page) describing the index you implemented and the rationale behind it. You are allowed to add illustrative figures.
2. Running time of creating `custom_indexing_algorithm` (wall time). Note that it should be at most half (1/2) of the running time of `semi_optimized_exhaustive_search`, reported in Section 2.1.
3. Running time of searching over `query_vectors` with `custom_index_search` (wall time). Note that it should be at most third (1/3) of the running time of `semi_optimized_exhaustive_search`.
4. Recall@10 for `custom_index_search`. Note that it should be at least 0.8.

Some Notes about Section 2.2

- The code in the notebook should be documented, modular and well-structured. It is also part of your grade.
- A function cannot be longer than 15 lines. Refactor the code when it is possible. It is also part of your grade.
- You are allowed to add as many helper functions as you need, but you must use `custom_indexing_algorithm` and `custom_index_search`. You can change the signature of these functions if you want.
- You cannot use faiss, scipy, sklearn or any package allowing vector search optimizations, besides numpy.
- Experiment your solution with the additional two query-index set pairs, attached to the data directory. The performance requirements should also be satisfied over them, but there is no need to report these results.
- Note that the dimensionality of the vectors provided to you may differ from the vectors we use to test your solution.

3 Part 3: Pinecone VectorDB and RAG

In this part, you will explore the capabilities of Pinecone VectorDB and implement a basic Retrieval-Augmented Generation (RAG) pipeline. The goal is to enhance the performance of a standard Question Answering (QA) model by leveraging relevant documents from a vector database.

3.1 Task Description

1. **Dataset Selection:**
   - Find a dataset for which a standard QA model fails to accurately answer the questions, often resulting in hallucinations.
   - Ensure that the correct answers are available within a set of documents that will be stored in the Pinecone VectorDB.
2. **Building a RAG Pipeline:**
   - Implement a pipeline that retrieves relevant documents from the Pinecone VectorDB.
   - Integrate the retrieved documents with a generative model to form the RAG pipeline.
   - Anecdotally asses the RAG pipeline ability in answering the questions from the selected dataset.

3.1.1 Report

- Describe the dataset chosen and the rationale for its selection.
- Provide anecdote examples of cases where the standard QA model failed and the RAG pipeline succeeded.
- Describe the retrieval system and the prompts used in the RAG pipeline.
- Include any insights or observations on the effectiveness of the RAG pipeline in reducing hallucinations and improving answer accuracy. Again, note that we are not expecting a thorough evaluation; instead, focus on anecdotal insights.

3.1.2 Code

- Implement the entire RAG pipeline in Python scripts.
- Upload the code to a public GitHub repository. One team member should ensure that the repository is public and accessible. Add the link to the report.
- Ensure the code is modular and well-structured, covering all elements of the RAG pipeline, including:
  - Document reading and preprocessing.

  - Chunking documents for embedding generation.
  - Embedding generation and insertion into Pinecone VectorDB.
  - Retrieval of relevant documents.
  - Generating answers to given questions using the retrieved documents.
- Comment the code for clarity and maintain good coding practices.

A Accessing Jupyter Notebook Remotely

To access Jupyter Notebook remotely, follow these steps:

Step 1: Run Jupyter Notebook on the Server

Run the following command on putty, opened from the server:

```
jupyter notebook --no-browser --port=8888
```

Copy the first URL displayed in the command output (something like `http://localhost:8888/?token=c98ae3a6aea930892ac3f1b0242619c4753901fce430d75b`) without exiting the command. You can copy it using the right-click of your mouse.

Step 2: Set Up SSH Tunnel

Open the command prompt (cmd) on your local computer and run the following command:

```
ssh -N -f -L localhost:8889:localhost:8888 student@<dns_name>
```

Replace $< dns\_name >$ with the appropriate server address, copied from Azure Portal. When prompted, enter the password: `Technion2023!`

Step 3: Access Jupyter Notebook

In your local web browser, paste the URL copied from the server. Modify the port in the URL to match the first port number mentioned in the previous command (in this case, change it to 8889). Press Enter to access Jupyter Notebook remotely.