Homework 1: Vector Search

Submission Instructions

1. Submission Date: The assignment is due on 06.05.24, at 22:55.
2. Team Submission: only pairs, unless otherwise approved in advance.
3. Submission File Format: Each team should submit a zip file named `<ID1_ID2>`, where `ID<i>` represents the ID number of each of the two team members. The zip file must include the following files:
    - `part1.ipynb`
    - `part2.ipynb`
4. Code Compliance: Ensure that your code adheres to the provided template.
5. Questions: Any questions should be posted in the Moodle forum, and must be asked in English.
6. It is highly recommended to use the server assigned for you. Please follow the instructions provided in the files `Students - How to work with your VM.pdf` and `installations.txt`.
7. If you struggle with package installations, please notify us as soon as possible via the forum.

1 Faiss

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 notebook are:

- (1.1.1) A plot displaying a running time comparison between `naive_exhaustive_search` and `faiss_flatL2` as a function of the number of vectors in the index.
- (1.1.2) A plot displaying a running time comparison between `optimized_exhaustive_search` and `faiss_flatL2` as a function of the number of vectors in the index.
- (1.1.3) A plot displaying a running time comparison between `optimized_exhaustive_search` and `faiss_flatL2` as a function of the dimensionality of the vectors.

1.2 Faiss LSH:

The deliverables that should appear in the notebook 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. 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

In this part you are asked only the run the existing code, without implementing anything new. The relevant code cells comprises the following:

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 relevant results should appear in the designated code cells:

1. Wall time for generating `custom_indexing_algorithm`.
   It should be at most half ($\leq \frac{1}{2}$) the running time of `semi_optimized_exhaustive_search`, as reported in Section 2.1.
2. Wall time for searching over `query_vectors` using `custom_index_search`.
   It should be at most a third ($\leq \frac{1}{3}$) of the running time of `semi_optimized_exhaustive_search`.
3. Recall@10 of the `custom_index_search` function.

Notes on Section 2.2

- Your code must be documented, modular, and well-structured. This will be graded.
- Each function must not exceed 15 lines. Refactor your code where necessary. This is also part of your grade.
- You may add as many helper functions as needed. However, you must implement and use `custom_indexing_algorithm` and `custom_index_search`. You may modify their signatures if desired.

- You are not allowed to use `faiss`, `scipy`, `sklearn`, or any other library that provides optimized vector search. Only `numpy` is permitted.
- Note that the dimensionality of the vectors provided to you may differ from those used for evaluation.
- Grading for the wall time of `custom_indexing_algorithm` is based on the ratio between its runtime and that of `semi_optimized_exhaustive_search`:
  - If the ratio ≥ 1: grade is 0.
  - If the ratio ≤ 0.5: grade is 100.
  - Otherwise: grade is $100 \times \left(1 - \frac{\text{ratio}-0.5}{0.5}\right)$, i.e., scaled linearly between 100 (at ratio 0.5) and 0 (at ratio 1).

  The same scheme applies separately to both the indexing and search runtime requirements. Recall@10 will be graded relative to the baseline results provided in the notebook.
- Grading for the wall time of `custom_index_search` is based on the ratio between its runtime and that of `semi_optimized_exhaustive_search`:
  - If the ratio ≥ 1: grade is 0.
  - If the ratio $\leq \frac{1}{3}$: grade is 100.
  - Otherwise: grade is $100 \times \left(1 - \frac{\text{ratio}-\frac{1}{3}}{\frac{2}{3}}\right)$, which linearly interpolates between 100 (at ratio $\frac{1}{3}$) and 0 (at ratio 1).
- Grading for Recall@10 is based on the achieved recall score:
  - If Recall@10 ≥ 0.25: grade is 100.
  - If Recall@10 ≤ 0.15: grade is 0.
  - Otherwise: grade is $100 \times \left(\frac{\text{recall}-0.15}{0.10}\right)$, which linearly interpolates between 0 (at recall 0.15) and 100 (at recall 0.25).

  The top 3 student pairs with the highest Recall@10 scores – among those who satisfy both runtime ratio constraints (a grade of 100 for these components) – will receive a bonus to their total assignment grade: +3 points for 1st place, +2 points for 2nd, and +1 point for 3rd.

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: `Technion2024!`

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.