Building a Semantic Search Engine with Vectorized Databases
Project Details

Deliverables:

The index files for four databases, which will be used to evaluate the project.The databases are provided to you with varying sizes: 20M, 15M, 10M, and 1M. More details can be found in the Evaluation.ipynb.

System Constraints:

DB size/ Constraint	1 M Row	10 M Row	15 M Row	20 M Row
Peak RAM Usage (on retrieval phase)	20 MB	50 MB	50 MB	50 MB
Time Limit (on retrieval phase)	3 sec	6 sec	8 sec	10 sec
Score (Min accepted)	-5000	-5000	-5000	-5000
Index Size (Max accepted)	50 MB	100 MB	150 MB	200 MB

Notebook and Code Walkthrough:

First you'll fork this repo: https://github.com/farah-moh/vec_db

In the repo you'll find a vec_db.py file. This file contains the implementation for your vector index. It has multiple functions:

1. **generate_database():**
   This function generates a random database of a given size. You should run this to generate the databases for the specified sizes in the project requirements. It then calls a build_index() function to build the index on the generated data.
2. **_build_index():**
   This function is empty. You should add your index building logic here.
3. **_write_vectors_to_file():**
   This is a utility function that is called by generate_database() function to write the randomly generated data on disk. It uses a memory mapped implementation which enables you to read specific rows from the database without reading the whole file on memory.
4. **_get_num_records():**
   This is a utility function that calculates the number of records in the database.
5. **insert_records():**
   This function inserts new records in the data file. It extends the size of the current file and adds the new data. It then calls the _build_index() function again to rebuild the index. If your index supports insertions, you should call the insert function of your index instead (but handling insertions is not required).
6. **get_one_row():**

This function reads one row from the data file given its index. It doesn't load the whole file on memory, but only the needed row.

7. **get_all_rows():**
   This function returns all the database rows. Take care as it will load the whole file on memory.
8. **retrieve():**
   This function is the main retrieval function in your index. It returns the top K vectors given a query vector. The current implementation is a brute-force KNN solution (very slow). You should replace it with your retrieval logic, **without changing the function name or parameters.**
9. **_cal_score():**
   This is a utility function that calculates the cosine similarity between two vectors. This is the similarity measure that we will be using.

There's also a evaluation.py file which has the following functions:

1. **run_queries():**
   This function will run a given number of queries against your index and return your result and the ground-truth (actual top K nearest vectors).
2. **eval():**
   This function evaluates your results against the ground-truth. Maximum score is 0.
3. In the **main** function you can try generating some data and running some queries to test your retrieval recall.

Then you can use this notebook to evaluate your code:
https://colab.research.google.com/drive/1NDjJl623MuTBXJcVvtd09zIW-zF5sjIZ#scrollTo=hV2Nc_f8Mbqh&uniqifier=1

The notebook clones your repository and loads your index files, guiding you through the evaluation process. Follow the steps in the notebook to ensure your code is evaluated correctly.

We'll agree on a common seed for the database so everyone has the same data. On the submission day, the query seed number will be changed.

Important: The __init__() and retrieve() functions must retain the same name and parameters as in the repository, as it will be called during evaluation. You are free to restructure the code and add additional functions, but make sure your code runs without errors in the notebook.