Task Overview

Task: to build an approximate K-NN Graph for a set of vectors. I.e., for each vector, find its approximate k nearest neighbors in a limited time (30 minutes).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Description</th>
<th>Size</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Bing queries encoded by Turing AGI v5</td>
<td>10^7</td>
<td>100</td>
</tr>
</tbody>
</table>

Evaluation Metric: Recall = number of true top 100 nearest neighbors / 100
Evaluation Environment: 32 CPU x 2.7 GHz, 64 GB Main Memory, 32 GB Storage, Ubuntu 20.04.5 LTS - no GPU

Solution Overview

We first use the faiss library [1] to obtain an initial KNN graph, and then use the pynndescent library [2] to refine it.

Our solution involved 2 major steps:
1. Construct an initial KNN graph.
2. Refine the initial KNN graph.

KNN Graph Initialization

Goal: A good trade-off between recall and runtime

Step:
1. Build: index=faiss.index_factory(......)
2. Train: index.train(X)
3. Add: index.add(X)
4. Search: D,I=index.search(X, k=340)

Parameter:
- index_string="IVF1100,PQ100x4fsr,RFlat"
- nprobe=77
- thread=16
- max_candidates=151

To partition the index into Voronoi cells is a popular approach, which reduces search space of our solution, and produces an approximate answer. (IVF)

Conclusion

1. The technical route of initializing the graph first and then refining it is fast and efficient.
2. The re-ranking of faiss and the searching of 340-NN make many reverse neighbors can be exploited to improve performance.
3. We achieve a good trade-off in the time distribution of initialization and refinement (24~25m on initializing and 4~5m on Refining). The iterative one-round NNDescent algorithm has the highest cost performance.

References: