Benchmarking Nearest Neighbor Search: Influence of LID and Visualization of Results

Martin Aumüller and Matteo Ceccarello
IT University of Copenhagen

May 4, 2019

http://ann-benchmarks.com/edml19
https://github.com/erikbern/ann-benchmarks/
**$k$-Nearest Neighbor Problem**

- **Preprocessing**: Build DS for set $S$ of $n$ data points
- **Task**: Given query point $q$, return $k$ closest points to $q$ in $S$
Nearest neighbor search on words

• GloVe: learning algorithm to find vector representations for words

• *GloVe.twitter* dataset: **1.2M words**, vectors trained from **2B tweets**, **100 dimensions**

• Semantically similar words: nearest neighbor search on vectors

https://nlp.stanford.edu/projects/glove/
GloVe Examples

“calgary”
- “edmonton”
- “winnipeg”
- “vancouver”
- “ottawa”
- “toronto”

“canada”
- “europe”
- “uk”
- “australia”
- “ireland”
- “usa”

“maple”
- “walnut”
- “syrup”
- “leaf”
- “pine”
- “bourbon”
Basic Setup

• Data is described by **high-dimensional feature vectors**

• **Exact similarity search is difficult** in high dimensions

• data structures and algorithms suffer
  • **exponential dependence** on dimensionality
  • in time, space, or both

Curse of Dimensionality
Why is Exact NN difficult?

- Choose \( n \) random points from \( N(0, 1/d)^d \), for large \( d \)
- Choose a random query point
- nearest and furthest neighbor basically at same distance
Performance on GloVe

Recall-Queries per second (1/s) tradeoff - up and to the right is better
2 Implementations

ANNOY (Bernhardsson, 2015)

HNSW (Malkov & Yashunin, 2016)
ANNOY

• 2 parameters:
  • # of trees
  • # candidates collected
Benchmarking Approach

Architecture

Preprocessing Phase
- train(X)
- Algorithm
  - Measure index build time/size

Query Phase
- query(q, k)
- getAdditional()
  - "candidates": 245, ...
- Queries left?
  - Yes
  - Experiments left?
  - No
  - Save results

Measure query time
Configuration

float:
  any:
    annoy:
      constructor: Annoy
      base_args: ["@metric"]
    run_groups:
      one_or_two_hundred_trees:
        args: [[100, 200], [100, 200, 400, 1000]]
      four_hundred_trees:
        args: [400, [1000, 2000, 4000, 10000]]
Configuration

float:
any:
  annoy:
    constructor: Annoy
    base-args: ["@metric"]
  run-groups:
    one-or-two-hundred-trees:
      args: [[100, 200], [100, 200, 400, 1000]]
    four-hundred-trees:
      args: [400, [1000, 2000, 4000, 10000]]

Annoy("euclidean", 100, 100)
Annoy("euclidean", 100, 200)
Annoy("euclidean", 100, 400)
Annoy("euclidean", 100, 1000)
Annoy("euclidean", 200, 100)
Annoy("euclidean", 200, 200)
Annoy("euclidean", 200, 400)
Annoy("euclidean", 200, 1000)
Annoy("euclidean", 400, 1000)
Annoy("euclidean", 400, 2000)
Annoy("euclidean", 400, 4000)
Annoy("euclidean", 400, 10000)
Visualizing runs

Evaluation approach: plot pareto frontier
Focus of the Presentation

• Q1: Can we design query workloads of different difficulty for real-world datasets?

• Q2: What is hidden in standard performance/quality plots?
Local Intrinsic Dimensionality

- Introduced by (Houle, 2013)
- Local measure of dimensionality
- Defined with respect to a query point

$$ID_r(q) = \lim_{\varepsilon \to 0} \frac{\log(\frac{\text{blue}}{\text{pink}})}{\log(1 + \varepsilon)}$$

- Estimation via MLE (Amsaleg et al., 2015)

$$\widehat{ID}_k(q) = \left(\frac{1}{k} \sum \ln \frac{r_i}{r_k}\right)^{-1}$$
LID Distribution

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Data Points</th>
<th>Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT [9]</td>
<td>1 000 000</td>
<td>128</td>
</tr>
<tr>
<td>MNIST</td>
<td>65 000</td>
<td>784</td>
</tr>
<tr>
<td>Fashion-MNIST [19]</td>
<td>65 000</td>
<td>784</td>
</tr>
<tr>
<td>GLOVE [17]</td>
<td>1 183 514</td>
<td>100</td>
</tr>
<tr>
<td>GLOVE-2M [17]</td>
<td>2 196 018</td>
<td>300</td>
</tr>
<tr>
<td>GNEWS [16]</td>
<td>3 000 000</td>
<td>300</td>
</tr>
</tbody>
</table>
Hypotheses

- LID reflects difficulty to answer individual queries on a single, fixed data set.
- LID distribution predicts performance between different data sets.
Preparation of Data Sets

• Choose 10,000 query points with
  • smallest LID (‘easy’)
  • largest LID (‘hard’)
  • around median LID (‘middle’)

![Graph showing Local Intrinsic Dimensionality for MNIST dataset]
EVALUATION
Adaptiveness

• *k*-NN search algorithms are usually *not* adaptive
Evaluation Beyond Averages
Distribution of Individual Recall
Distribution of Individual Query Times

Annoy

HNSW

1 / query time
Conclusion/Open Problems

• LID: nice tool to prepare workloads of varying difficulty
• Interactive visualization and evaluation

• LID as a tool to adapt query algorithms?
• Other dimensionality measures as difficulty estimators?

http://ann-benchmarks.com/edml19
https://github.com/erikbern/ann-benchmarks/