Approximate Nearest Neighbors in Limited Space

Piotr Indyk
MIT

Tal Wagner
MIT

Introduction

What is the space complexity of the (Euclidean) Approximate Nearest Neighbor problem?

Problem: Compress a dataset \( X = \{x_1, \ldots, x_n\} \subset \mathbb{R}^d \) into a small size data structure (sketch) that can answer \((1 + \epsilon)\)-approximate nearest neighbor queries:

\[
\text{Given } y \in \mathbb{R}^d, \text{ return } i^* \in \{1, \ldots, n\} \text{ s.t. } \|y - x_{i^*}\| \leq (1 + \epsilon) \cdot \min_{i \in \{1, \ldots, n\}} \|y - x_i\|.
\]

Benefits of compression:

- **Time**: Speed-up linear scan of data.
- **Space**: Fit on memory-limited devices like GPUs (Johnson, Douze, Jégou (2017)).
- **Communication**: Facilitate distributed architectures.

Context:

- Nearest neighbor classifiers are popular in Machine Learning (e.g. Efros (2017)).
- Large body of empirical work on the above problem (see survey at Wong et al. (2016)).
- Yet, no better theoretical bounds than the dimension reduction theorem due to Johnson & Lindenstrauss (1984) were previously known.

Our Results

**Problem 1 – Approximate Nearest Neighbor:**

Answer query with success probability \( 1 - 1/n^\Omega(1) \).

<table>
<thead>
<tr>
<th>Method</th>
<th>Size in bits per point(^\ast)</th>
<th>What can it approximate?</th>
</tr>
</thead>
<tbody>
<tr>
<td>No compression</td>
<td>( O(d \log n) )</td>
<td>Distances between any ( y ) and all ( x \in X )</td>
</tr>
<tr>
<td>Johnson &amp; Lindenstrauss (1984)</td>
<td>( O\left(\frac{\log^2 n}{\epsilon^2}\right) )</td>
<td>Distances between any ( y ) and all ( x \in X )</td>
</tr>
<tr>
<td>Kushilevitz, Ostrovski, Rabani (2000)</td>
<td>( O\left(\frac{\log n}{\epsilon^2 \cdot \log R}\right) )</td>
<td>Distances between any ( y ) and all ( x \in X ), assuming ( |x - y| \in [r, R^r] )</td>
</tr>
<tr>
<td>Indyk &amp; Wagner (2017; 2018)</td>
<td>( O\left(\frac{\log n}{\epsilon^2}\right) )</td>
<td>Distances between all ( x, y \in X ), no out-of-sample query support</td>
</tr>
<tr>
<td>This work</td>
<td>( O\left(\frac{\log n}{\epsilon^2} \cdot \log(1/\epsilon)\right) )</td>
<td>Nearest neighbor of any ( y ) in ( X )</td>
</tr>
</tbody>
</table>

**Problem 2 – Approximate Distance Queries:**

Compress \( X \) such that for any query set \( Y \subset \mathbb{R}^d \) with \( q \) query points, the sketch can estimate all distances \( ||x - y|| \) for \( x \in X \) and \( y \in Y \), up to distortion \((1 \pm \epsilon)\).

<table>
<thead>
<tr>
<th>Reference</th>
<th># queries</th>
<th>Size in bits per point(^\ast)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Molinaro, Woodruff, Yaroslavtsev (2013)</td>
<td>( q \geq n )</td>
<td>( \Omega\left(\frac{\log^2 n}{\epsilon^2}\right) ) matches the Johnson-Lindenstrauss (1984) upper bound for ( q = n^{\Omega(1)} ).</td>
</tr>
<tr>
<td>This work</td>
<td>( 1 \leq q \leq n )</td>
<td>( O\left(\frac{\log n}{\epsilon^2} (\log q + \log(1/\epsilon))\right) )</td>
</tr>
</tbody>
</table>

\(\ast\) For simplicity, the bounds stated in this poster assume that all points coordinates in \( X \) are represented by \( O(\log n) \) bits. See the paper for the full dependence on all parameters.

Overview of Techniques

For this poster, we use a simplified sketch due to Indyk, Razenshteyn, Wagner (2017).

- Lossier than Indyk & Wagner (2017) by \( O(\log \log n) \), but simpler and captures main ideas.

The dataset \( X \) is represented by a hierarchical clustering tree.

Tree edges are annotated with binary precision bits of point coordinates in \( X \).

**How to compress the tree?**

**Prior work: “Bottom-out Compression”**

Remove every non-branching path from the tree, except its top edges.

- Stores most significant bits of each cluster.
- Preserves global cluster structure.

This preserves distances within \( X \):

- but not the nearest neighbor of a new query point \( y \):

This work: “Middle-Out Compression”

Remove every non-branching path from the tree, except its top and bottom edges.

- Also stores least significant bits of each cluster.
- Also preserves local cluster structure.

Overview of Analysis

Approximate nearest neighbor algorithm for a query point \( y \in \mathbb{R}^d \):

- Search for \( y \) down the tree, by the bits on the tree edges, until reaching a leaf.
- Return the point in \( X \) represented by that leaf.
- How to handle missing bits in the tree? **Guess they are the same as \( y \).**
- **Guessed right? Yay!** The algorithm learned the right absolute location of \( X \) from \( y \).

Ground truth \( X \) and \( y \)

Decompressed \( X \) and \( y \)

**Guessed wrong? It’s okay.** The algorithm doesn’t know it learned \( X \) wrong, but any point from now on is a good approximate nearest neighbor.

Ground truth \( X \) and \( y \)

Decompressed \( X \) and \( y \)

same up to small distortion

Decompressed \( X \) and \( y \)

algorithm will return an arbitrary point from this cluster as the nearest neighbor—this is okay

References


