Fast $k$-Nearest Neighbour Search via Prioritized DCI

Ke Li

Jitendra Malik
Introduction

• The method of $k$-nearest neighbours is a fundamental building block of many machine learning methods.

• Problem definition: Given a database of $n$ points and the query, find the $k$ points that are closest to the query.
Notions of Dimensionality

• The hardness of a dataset can be characterized using two notions of dimensionality.
  – Ambient dimensionality: the dimensionality of the space that contains the data points.
  – Intrinsic dimensionality: can be roughly thought of as the dimensionality of the data manifold.
Intrinsic Dimensionality

• Definition:

A dataset $D$ has intrinsic dimensionality\(^1\) $d'$ if for all $r > 0$, $\alpha > 1$ and $p$ such that $|B_p(r)| \geq k$,

$$|B_p(\alpha r)| \leq \alpha^{d'} |B_p(r)|$$

\(^1\)This is also known as the expansion dimension or the KR-dimension.
Intrinsic Dimensionality

- A $d'$-dimensional uniform grid $\mathbb{Z}^{d'}$ has intrinsic dimensionality $d'$. 
Intrinsic Dimensionality

- If it were embedded in a higher-dimensional space, it would retain its intrinsic dimensionality.
Intrinsic Dimensionality

• Equivalently:

\[
\log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r)
\]

  – Plot \(\log_2 |B_p(\alpha r)|\) against \(\log_2 (\alpha r)\)

  – Maximum slope upper bounds the intrinsic dimensionality.
Intrinsic Dimensionality

• Equivalently:

\[ \log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r) \]

– Plot \( \log_2 |B_p(\alpha r)| \) against \( \log_2 (\alpha r) \)

– Maximum slope upper bounds the intrinsic dimensionality.
Intrinsic Dimensionality

- Equivalently:
  \[ \log_2 |B_p(\alpha r)| \leq d' \log_2 (\alpha r) + (\log_2 |B_p(r)| - d' \log_2 r) \]

- Plot \( \log_2 |B_p(\alpha r)| \)
  against \( \log_2 (\alpha r) \)

- Maximum slope upper bounds the intrinsic dimensionality.
Why is High Dimensionality Hard?

\[ d' = 1 \]
Why is High Dimensionality Hard?

\[ d' = 1 \]
Why is High Dimensionality Hard?

\[ d' = 2 \]
Why is High Dimensionality Hard?

\[ d' = 2 \]
Why is High Dimensionality Hard?

\[ d' = 3 \]
Why is High Dimensionality Hard?

\[ d' = 3 \]
Why is High Dimensionality Hard?

\[ d' = 3 \]

- The number of nearby points could grow exponentially in intrinsic dimensionality.
History

• The problem of nearest neighbour search was formalized by Cover & Hart (1967) and Minsky & Papert (1969) in their seminal book, *Perceptrons.*
History

• The problem of nearest neighbour search was formalized by Cover & Hart (1967) and Minsky & Papert (1969) in their seminal book, *Perceptrons*.

“We conjecture that even for the best possible $A_{\text{file}} - A_{\text{find}}$ pairs, ... for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of the memory.”

p. 223
The problem of nearest neighbour search was formalized by Cover & Hart (1967) and Minsky & Papert (1969) in their seminal book, *Perceptrons*. We conjecture that even for the best possible $A_{\text{file}} - A_{\text{find}}$ pairs, ... for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of the memory.

In other words, even for the best choice of dataset and queries, when the dataset is large and high-dimensional, doing substantially better than exhaustive search is conjectured to be impossible.

“... for large data sets with long word lengths there are no practical alternatives to large searches that inspect large parts of the memory.”

p. 223
The Curse of Dimensionality

- **k-d tree** (Bentley, 1975)
- **X-tree** (Berchtold et al., 1996)
- **R-tree** (Guttman, 1984)

- Exact deterministic algorithms based on space partitioning.
- Query time exponential in the ambient dimensionality.

Fast $k$-Nearest Neighbour Search via Prioritized DCI
The Curse of Dimensionality

- Exact deterministic algorithm.
- Query time \textit{polynomial} in ambient dimensionality.
- Space complexity \textit{exponential} in ambient dimensionality.

(Meiser, 1993)
The Curse of Dimensionality

- LSH introduced the idea of randomization.
- Approximate randomized algorithm based on space partitioning.
- Query time is $O(dn^\rho)$, where $\rho \approx 1/(1 + \epsilon)^2$.


- k-d tree
- R-tree
- Meiser X-tree
- LSH (Indyk & Motwani, 1998)
- Euclidean LSH (Datar et al., 2004)
- (Andoni & Indyk, 2006)
The Curse of Dimensionality

- Exact randomized algorithms based on space partitioning.
- Query time \textit{exponential} in intrinsic dimensionality.

- \textit{k-d} tree (1975)
- R-tree (1984)
- Meiser X-tree (1993)
- Spill tree (Liu et al., 2004)
- Virtual spill tree (Dasgupta & Sinha, 2015)
- RP tree (Dasgupta & Freund, 2008)

Fast $k$-Nearest Neighbour Search via Prioritized DCI
The Curse of Dimensionality

- Exact algorithms based on local search and coarse-to-fine.
- Query time exponential in intrinsic dimensionality.

Fast k-Nearest Neighbour Search via Prioritized DCI
The Curse of Dimensionality

- Our contribution: a new family of exact randomized algorithms, known as Dynamic Continuous Indexing.
The Curse of Dimensionality

- Query time *linear* in ambient dimensionality and *sublinear* in intrinsic dimensionality.
- Space complexity independent of ambient or intrinsic dimensionality.
The Curse of Dimensionality

- $O(d'^d + \log n)$
- $O(2^{3d'} \log n)$
- $O(d' \max(\log n, n^{1-1/d'})$)
- $O(d' \max(\log n, n^{1-m/d'}) + (m\log m)\max(\log n, n^{1-1/d'})$)

Intrinsic Dimensionality ($d'$)

Query Time Complexity

K&R, Nav. Net, Cover Tree

Spill Tree, RP Tree
The Curse of Dimensionality

- K&R, Nav. Net, Cover Tree
- Spill Tree, RP Tree
- DCI

Query Time Complexity vs. Intrinsic Dimensionality ($d'$)

- $O(d'^d + \log n)$
- $O(2^{3d'} \log n)$
- $O(d \max(\log n, n^{1-1/d'})$
- $O(d \max(\log n, n^{1-m/d'}) + (m \log m) \max(\log n, n^{1-1/d'})$
The Curse of Dimensionality

Fast $k$-Nearest Neighbour Search via Prioritized DCI

Query Time Complexity

Intrinsic Dimensionality ($d'$)

- $O(d'^d + \log n)$
- $O(2^{3d'} \log n)$
- $O(d\max(\log n, n^{1-1/d'}))$
- $O(d\max(\log n, n^{1-m/d'}) + (m\log m)\max(\log n, n^{1-1/d'}))$
Our Approach

• Key difference from prior methods: Dynamic Continuous Indexing (DCI) avoids *space partitioning*.

• Space partitioning is a divide-and-conquer strategy that underlies most existing methods, including $k$-d trees and locality-sensitive hashing (LSH).
  
  — It works by partitioning the space into discrete cells and keeping track of points contained in each.

• We conjecture that the curse of dimensionality stems from the inherent deficiencies of space partitioning.
The Case Against Space Partitioning
$k$-d tree

Fast $k$-Nearest Neighbour Search via Prioritized DCI
$k$-d tree

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Fast $k$-Nearest Neighbour Search via Prioritized DCI
$k$-d tree

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Fast $k$-Nearest Neighbour Search via Prioritized DCI
$k$-d tree

Fast $k$-Nearest Neighbour Search via Prioritized DCI
$k$-d tree

Fast $k$-Nearest Neighbour Search via Prioritized DCI
$k$-d tree

- Either the number or volume of the cells must grow exponentially in dimensionality.
$k$-d tree

- Either the number or volume of the cells must grow exponentially in dimensionality.
- “Field of view” limited to cell containing the query.
$k$-d tree
• The number of neighbouring cells that must be searched grows exponentially in the dimensionality in the worst case.
LSH

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Fast $k$-Nearest Neighbour Search via Prioritized DCI
Fast $k$-Nearest Neighbour Search via Prioritized DCI
LSH

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Fast \( k \)-Nearest Neighbour Search via Prioritized DCI
LSH

- Searching over only the points in the cell containing the query would lead to the incorrect result.
• As the ratio of surface area to volume grows in dimensionality, the number of overlapping grids grows in dimensionality.
LSH

Fast $k$-Nearest Neighbour Search via Prioritized DCI
LSH

Fast $k$-Nearest Neighbour Search via Prioritized DCI
• Inefficient when query lies in denser regions.
Fast k-Nearest Neighbour Search via Prioritized DCI
Fast \( k \)-Nearest Neighbour Search via Prioritized DCI
LSH

• Returns no points when query lies in sparser regions.
• Returns no points when query lies in sparser regions.
• This partitioning is unsuitable for datasets with large variations in density.
Prioritized DCI
Prioritized DCI
Prioritized DCI

Project all data points along a random direction.
Prioritized DCI

Project all data points along multiple random directions.
Prioritized DCI

Project the query along each projection direction.

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Prioritized DCI

Find the closest point to the query along each projection direction and add them to the frontier.
Prioritized DCI

Compare their projected distances to the query.
Prioritized DCI

Visit the point with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

Find the next closest point along the projection direction that has just been processed and add it to the frontier.
Prioritized DCI

Compare projected distances of points on the frontier and visit the one with the shortest projected distance.
Prioritized DCI

There is now a point that has been visited along all projection directions. We add it to the candidate set.
Prioritized DCI

Visit the next point.
Prioritized DCI

Visit the next point.

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Prioritized DCI

Visit the next point.
Prioritized DCI

Visit the next point and add it to the candidate set.
Prioritized DCI

Visit the next point and add it to the candidate set.
Prioritized DCI

Perform exhaustive search over candidate points and return $k$ points that are closest to the query.
Intuition

• Points are added to the candidate set in the order of their maximum projected distance to the query.
• Maximum projected distance is a lower bound on the true distance.
• As the number of projection directions increases, this lower bound approaches the true distance.

\[
\max_j \{ |\langle p^i, u_j \rangle - \langle q, u_j \rangle| \} = \max_j \{ |\langle p^i - q, u_j \rangle| \} \leq \|p^i - q\|_2
\]

where \( \|u_j\|_2 = 1 \quad \forall j \)
Lemma

• We derived the following lemma, which may be of independent interest:
• For any set of events $E_1, \ldots, E_n$, the probability that at least $m$ of them occur is at most:

\[ \frac{1}{m} \sum_{i=1}^{n} \Pr(E_i) \]

• When $m = 1$, this reduces to the union bound.

(Proof is in the “Fast k-Nearest Neighbour Search via Prioritized DCI” paper, though two students, Eric Xia and Zipeng Qin, later came up with simpler proofs, one using measure theory, and one using Markov’s inequality.)
Complexity

• Construction Time: $O(m(dn + n \log n))$
• Query Time: $O(dk \max(\log(n/k), (n/k)^{1-m/d'})) + mk \log m(\max(\log(n/k), (n/k)^{1-1/d'})))$
• Insertion Time: $O(m(d + \log n))$
• Deletion Time: $O(m \log n)$
• Space: $O(mn)$

where $m \geq 1$ is the number of projection directions chosen by the user.
Complexity

- **Construction Time:** \( O(m(dn + n \log n)) \)
- **Query Time:** \( O\left(\frac{dk \max(\log(n/k), (n/k)^{1-m/d'}) + nk \log m(\max(\log(n/k), (n/k)^{1-1/d'})))}{m(d + \log n)}\right) \)
- **Insertion Time:** \( O(m) \)
- **Deletion Time:** \( O(m \log n) \)
- **Space:** \( O(mn) \)

where \( m \geq 1 \) is the number of projection directions chosen by the user.

*Linear* dependence on ambient dimensionality

*Sublinear* dependence on intrinsic dimensionality
Complexity

- **Construction Time:** $O(m(dn + n \log n))$
- **Query Time:** $O(dk \max(\log(n/k), (n/k)^{1 - m/d'})) + mk \log m(\max(\log(n/k), (n/k)^{1 - 1/d'})))$
- **Insertion Time:** $O(m(d + \log n))$
- **Deletion Time:** $O(m \log n)$
- **Space:** $O(mn)$

where $m \geq 1$ is the number of projection directions chosen by the user.

A linear increase in intrinsic dimensionality can be mostly counteracted with a linear increase in the number of projection directions.
Experiments
Query Time on CIFAR-100

\[
\text{approximation ratio} = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}
\]

Fast \(k\)-Nearest Neighbour Search via Prioritized DCI
Query Time on MNIST

Approximation ratio = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}

Fast k-Nearest Neighbour Search via Prioritized DCI
Query Time on MNIST

Fast k Nearest Neighbour Search via Prioritized DCI

approximation ratio = \frac{\text{distance to retrieved nearest neighbours}}{\text{distance to true nearest neighbours}}
Space Efficiency on CIFAR-100

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Space Efficiency on MNIST

![Bar chart showing memory usage for different techniques: LSH, PQ, DCI (m=15, L=3), Prioritized DCI (m=15, L=3), Prioritized DCI (m=10, L=2). The chart illustrates a 21x improvement in efficiency compared to LSH.](image-url)
For More Details...

Fast $k$-Nearest Neighbour Search via Dynamic Continuous Indexing
Ke Li, Jitendra Malik
ICML, 2016

Fast $k$-Nearest Neighbour Search via Prioritized DCI
Ke Li, Jitendra Malik
ICML, 2017