Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Applications

A core problem in the following applications:

- clustering, $k$-means algorithms
- information retrieval in databases
- information theory: vector quantization encoding
- classification in learning theory
- ...
Nearest Neighbors: Getting Started

▷ **Input:** a set of points (aka sites) $P$ in $\mathbb{R}^d$, a query point $q$

▷ **Output:** $nn(q, P)$, the point of $P$ nearest to $q$

$$d(q, P) = d(q, nn(q, P)).$$ (1)
The Euclidean Voronoi Diagram and its Dual the Delaunay Triangulation

Key properties:

- Voronoi cells of all dimensions
- Voronoi - Delaunay via the nerve construction
- Duality: cells of dim. $d - k$ vs cells of dimension $k$
- The empty ball property
Nearest Neighbors Using Voronoi Diagrams

- Nearest neighbor by walking
  - start from any point $p \in P$
  - while $\exists$ a neighbor $n(p)$ of $p$ in $Vor(P)$
    - closer to $q$ than $p$,
    - step to it: $p = n(p)$
  - done $nn(q) = p$

- Argument: the Delaunay neighborhood of a point is complete
  $Vor(p, P) =$ cell of $p$ in $Vor(P)$
  $N(p) =$ set of neighbors of $p$ in $Vor(P)$
  $N'(p) = \{p\} \cup N(p)$
  $Vor(p, N'(p)) = Vor(p, P)$

- Exercise: specify the algorithm using DT
The Nearest Neighbors Problem: Overview

▶ **Strategy:** preprocess point set $P$ of $n$ points in $\mathbb{R}^d$ into a data structure (DS) for fast nearest neighbor queries answer.

▶ **Ideal wish list:**
  - The DS should have linear size
  - A query should have sub-linear complexity i.e. $o(n)$
    - When $d = 1$: balanced binary search trees yield $O(\log n)$

▶ **Core difficulties:**
  - *Curse of dimensionality in $\mathbb{R}^d$: for high $d$, it is difficult to outperform the linear scan*
  - Interpretation: meaningfulness of distances in high dimensional spaces – distance concentration phenomena.
The Nearest Neighbors Problem: Elementary Options

▷ The trivial solution:
$O(dn)$ space, $O(dn)$ query time

▷ Voronoi diagram

d = 2, $O(n)$ space $O(\log n)$ query time

d > 2, $O\left(n^{\left\lceil \frac{d}{2}\right\rceil}\right)$ space

→ Under locally uniform condition on point distribution
the 1-skeleton Delaunay hierarchy achieves:
$O(n)$ space, $O(c^d \log n)$ expected query time.

▷ Spatial partitions based on trees
The Nearest Neighbors Problem: Variants

- **Variants:**
  - *k*-nearest neighbors: find the *k* points in *P* that are nearest to *q*
  - given *r* > 0, find the points in *P* at distance less than *r* from *q*
  - Various metrics
    - $L_2$, $L_p$, $L_\infty$
    - String: Hamming distance
    - Images, graphs: distance based on optimal transportation
    - Point sets: distances via optimal alignment
  - Non metric spaces – cf metric trees

- **Main contenders in metric spaces:**
  - Tree like data structures:
    - quad-trees – and its variant ANN
    - (randomized) kd-trees
    - *k*-means trees – partition derived from *k*-means with *k*=2
  - Locally Sensitive Hashing
Comparison and appetizer: setup

- **Contenders:** various hierarchical methods for approximate NN
  - randomized kd-trees: hierarchical partition with split direction chosen at random
  - k-means trees: hierarchical partition with split direction derived from k-means
  - ANN
  - LSH

- **Assessment for the accuracy of the approximation:** precision i.e. fraction of queries for which the correct NN is found

- **Two main questions addressed:**
  - Question 1: for a fixed database, which algorithm is best?
  - Question 2: are the performances stable when the size of the DB changes?

Ref: Muja and Lowe, VISAPP 2009
Ref: O’Hara and Draper, Applications of Computer Vision (WACV), 2013
Main Contenders: Typical Results for Approximate NN

- DB used: Scale-Invariant Feature Transform (SIFT) for images: \( \{(x_i, y_i, \sigma_i)\} \)

**Question 1:** best algorithm

**Question 2 – for winners only**

i.e. for rand. kd-trees and k-means trees

**Take-home messages:**

- Randomized kd-trees and k-means trees win
- splits must exploit the variance in the dataset
- Speed-ups consistent when DB size increases

Ref: Muja and Lowe, VISAPP 2009
Ref: O’Hara and Draper, Applications of Computer Vision (WACV), 2013
Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Performances of geolocalization
a tale of data, features, and algorithms

▷ Source: Inria Colloquium talk by Alexei Efros, UC Berkeley, see https://iww.inria.fr/colloquium/fr/alexei-alyosha-efros-self-supervised-visual-learning-and-synthesis/

▷ Problem: geolocalize an image
  ▷ Solution one:
    ▷ DB of 6M images; (SIFT) features
    ▷ Answer: derived from the NN of the query image
  ▷ Solution two: DeepNet trained on DB of 91 M images
  ▷ Nb: correctness assessed at a given scale (in kilometers)
Localization from images: two (antipodal) strategies

Geolocation

im2gps, 2008
- Nearest Neighbors
- 6 million images

PlaNet, 2016
- Deep Net
- 91 million images
Performances: a matter of DB size

- **img2gps:**
  - Original img2gps: localization from NN in the database using simple SIFT, DB of 6.5 Flickr images
  - Revamped img2gps: more engineering on features but same DB size

- **img2gps versus Planet:**
  - Im2GPS: wins on city and region levels
  - PlaNet 6.2M: wins on on street, country and continent levels.

Ref: Weyand et al, ECCV 2016
The lesson: data, features, algorithms

Take home messages

- Do not underestimate the data
- DeepLand is not the only sweet spot...
Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
kd-tree for a collection of points (sites) \( P \)

**Definition:**

- A binary tree
- Any internal node implements a spatial partition induced by a hyperplane \( H \), splitting the point cloud into two equal subsets
  - right subtree: points \( p \) on one side of \( H \)
  - left subtree: remaining points
- The process halts when a node contains \( \leq n_0 \) points

Nb: the point realizing the median is stored in the node performing the split
kd-tree for a collection of points $P$

▷ Algorithm build_kdTree($S$)

\[ n \leftarrow \text{newNode} \]
\[ \text{if } |S| \leq n_0 \text{ then} \]
\[ \text{Store the point of } S \text{ into a container of } n \]
\[ \text{return } n \]
\[ \text{else} \]
\[ dir = \text{depth mod } d \]
Project the points of $S$ along direction $dir$
Compute the median $m$
\{Split into two equal subsets\}
\[ n.sample \leftarrow \text{sample } v \text{ realizing the median} \]
$L \leftarrow \text{point from } S \setminus \{v\} \text{ whose } dir\text{th coord is } < m$
$R \leftarrow \text{point from } S \setminus \{v\} \text{ whose } dir\text{th coord is } \geq m$
\[ n.left \leftarrow \text{build_kdTree}(L) \]
\[ n.right \leftarrow \text{build_kdTree}(R) \]
\[ \text{return } n \]
kd-tree: search

- **Main considerations:**
  - Exact versus approximate NN
  - No free lunch: complexity matters

- **Three main search strategies:**
  - (Approx.) the defeatist search: simple, but may fail
    (Nb: see later, distance concentration phenomena)
  - (Exact) the descending search: always succeeds, but may take time
  - (Exact) the priority search: strikes a compromise between the defeatist and descending strategies
kd-tree search: the defeatist search

▷ Key idea: recursively visit the subtree containing the query point
▷ Algorithm defeatist_search_kdTree: the defeatist search in a kd tree.

Require: Maintains $nn(q)$ of $q$, and $\tau = d(q, nn(q))$

$n \leftarrow \text{root}; \tau \leftarrow d(q, n.\text{sample})$

while $n \neq \text{NIL}$ do

Possibly update $nn(q)$ using $n.\text{sample}$, and $\tau$

if $q \in \text{Domain of } L$ then

defeatist_search_kdTree($n.\text{left}$)
endif

if $q \in \text{Domain of } R$ then

defeatist_search_kdTree($n.\text{right}$)
endif

▷ Complexity: assuming leaves of size $n_0$ – depth satisfies $2^h n_0 = n$

▷ search cost: $O(n_0 + \log(n/n_0))$

▷ Caveat: failure
kd-tree search: the exhaustive descending search

- Key idea: visit one or two subtree, depending on the distance $d(q, nn(q))$ computed

- Algorithm `descending_search_kdTree`: the descending search in a kd tree.

**Require:** Maintains $nn(q)$ of $q$, and $\tau = d(q, nn(q))$

**Require:** Uses the domain of a node $n$ (an intersection of half-spaces)

1. $n \leftarrow \text{root}$
2. $\tau \leftarrow d(q, n.sample)$
3. While $n \neq \text{NIL}$ do
   - Possibly update $nn(q)$ using $n.sample$
   - If $\text{Sphere}(q, \tau) \cap \text{Domain of } L$ then
     - `descending_search_kdTree(n.left)`
   - If $\text{Sphere}(q, \tau) \cap \text{Domain of } R$ then
     - `descending_search_kdTree(n.right)`

The value of $\tau$ ensures that the top cell will be visited.
kd-tree search: the priority search (idea)

- Priority search, key ideas:
  - Uses a priority queue to store nodes (regions), with a priority inversely proportional to the distance to \( q \).
  - Upon popping a node, the corresponding subtree is descended to visit the node closest to \( q \). Upon descending, \( nn(q) \) is updated.
  - While descending, the child not visited is possibly enqueued,
kd-tree search: priority search (algorithm)

- Uses a priority queue $Q$ to enumerate nodes by increasing distance to query $q$

**Ensure:** Maintains $nn(q)$ of $q$, and $\tau = d(q, nn(q))$

\[
\begin{align*}
  nn(q) & \leftarrow \text{root.sample} \\
  Q & .\text{insert(root)} \\
  \text{while True do} \\
  \quad \text{if } Q.\text{empty()} \text{ then} \\
  \quad \quad \text{return} \\
  \quad \quad \{ \text{Node with highest priority} \} \\
  \quad r & \leftarrow Q.\text{pop()} \\
  \quad \{ \text{The nearest box is too far wrt } nn(q) \} \\
  \quad \text{if } d(\text{bbox}(r), q) > \tau \text{ then} \\
  \quad \quad \text{return} \\
  \quad \quad \{ \text{Descend into box nearest to } q, \} \{ \text{and possibly enqueue the second node} \} \\
  \quad \text{for Nodes } n \text{ on the path from } r \text{ to the box nearest to } q \text{ do} \\
  \quad \quad \{ \text{Possibly update } nn(q) \text{ and } \tau \} \\
  \quad \quad d & \leftarrow d(q, n.\text{sample}) \\
  \quad \quad \text{if } d < \tau \text{ then} \\
  \quad \quad \quad nn(q) & \leftarrow n.\text{sample}; \tau \leftarrow d \\
  \quad \quad \{ \text{Possibly enqueue the second subtree} \} \\
  \quad \quad f & \leftarrow \text{brother of } n \\
  \quad \quad \text{if } d(\text{bbox}(f), q) \leq \tau \text{ then} \\
  \quad \quad \quad \{ \text{Insert with priority inverse to distance to } q \} \\
  \quad \quad \quad Q & .\text{insert}(f, 1/d)
\end{align*}
\]

NB: Enqueuing criterion can be adapted to report an $(1 + \varepsilon)$ approx. of the exact NN
kd-tree search: priority search (analysis)

Pros and cons:

- nn always found
- linear storage
- nn often found at an early stage ... then time spent in useless recursion
- In the worst-case, all nodes are visited.
- Maintaining the priority queue Q has a cost

Variants and improvements:

- Initially the Q with all nodes from root to leaf containing the query
- Stopping the recursion once a fraction of nodes has been visited
- Backing up defeatist search with overlapping cells
- Combining multiple randomized kd-trees
References


Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Improvements aiming at fixing the defeatist search

- **Defeatist search:** (early) choice of one side is risky

- **Simple improvements:**
  - Use several trees, and pick the best neighbor(s)
  - Allow overlap between cells in a node: selected points stored twice → spill trees
  - Use randomization to obtain different partitions rescuing the defeatist search
    - different permutations of coordinate axis
    - directions aiming at maximizing the variance
  - Next: randomization captures information on directions carrying variance
Random projection trees (RPTrees)

Aka Random partition trees (RPTrees!)

- **kd-tree:** axis parallel splits

- Splitting along a random direction $U \in S^{d-1}$: project onto $U$ and split at the (perturbed) median

![Diagram of random projection trees]

- Resulting spatial partition

![Diagram of resulting spatial partition]
Random projection trees: generic algorithm with jitter

Below: version where one also jitters the median defining the split

Algorithm build_RPTree(S)

Ensure: Build the RPTree of a point set S

if \(|S| \leq n_0\) then
  \(n \leftarrow \text{newNode}\)
  Store \(S\) into \(n\)
  return \(n\)

Pick \(U\) uniformly at random from the unit sphere
Pick \(\beta\) uniformly at random from \([1/4, 3/4]\]
Let \(v\) be the \(\beta\)-fractile point on the projection of \(S\) onto \(U\)
\(\text{Rule}(x) = \text{left if } \langle x, U \rangle < v, \text{otherwise right}\)
\(\text{left\_tree} \leftarrow \text{build\_RPTree}\left(\{x \in S : \text{Rule}(x) = \text{left}\}\right)\)
\(\text{right\_tree} \leftarrow \text{build\_RPTree}\left(\{x \in S : \text{Rule}(x) = \text{right}\}\right)\)
return \((\text{Rule}(\cdot), \text{left\_tree}, \text{right\_tree})\)

Remark: RP trees have the following property – more later: diameter of the cells decrease down the tree at a rate depending on the intrinsic dimension of the data.
RPTrees: varying splits and their applications

- Various types of splits possible
  - Randomized partition tree:
    - exact split
  - Randomized partition tree:
    - perturbed split
  - Spill tree with overlapping split:
    - regular spill tree
    - virtual spill tree

NB: splits monitor the tree structure and the search route

Spill trees:
- Regular spill trees:
  overlapping cells yield redundant storage of points
- Virtual spill trees:
  median splits used – no redundant storage
  query routed in multiple leaves using overlapping splits

Summary: tree creation versus search

<table>
<thead>
<tr>
<th></th>
<th>Routing data</th>
<th>Routing queries (defeatist style)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP tree</td>
<td>Perturbed split</td>
<td>Perturbed split</td>
</tr>
<tr>
<td>Regular spill tree</td>
<td>Overlapping split</td>
<td>Median split</td>
</tr>
<tr>
<td>Virtual spill tree</td>
<td>Median split</td>
<td>Overlapping split</td>
</tr>
</tbody>
</table>
Regular spill trees: size

- **Tree depth:** assume that
  - the number of nodes transmitted to a son decreases by a factor at least \( \beta = 1/2 + \alpha \),
  - a leaf accommodates up to \( n_0 \) points

Then: the tree depth \( l \) satisfies \( \beta^l n \leq n_0 \) i.e. \( l = O(\log_{1/\beta} \frac{n}{n_0}) \).

- **Tree size:**
  \[
  n_02^l = n_02^{\log_{1/\beta} \frac{n}{n_0}}.
  \]

- **Examples:**
  - \( \alpha = 0.05 \): \( O(n^{1.159}) \).
  - \( \alpha = 0.1 \): \( O(n^{1.357}) \).
Spill trees: compromising Storage vs NN searches

- Spill trees: overlapping splits yield superlinear storage
- Yet, search mail fail too:

\[ p_1 = nn(q) \in \text{Left} \quad q \in \text{right} \]

\[ p_1 = nn(q): \text{routed in left subtree only} \]

\[ \quad \text{query point } q: \text{routed in right subtree only} \]
Failure of the defeatist search

- **Goal:** probability that a defeatist search does not return the exact nearest neighbor(s)?

- **The event to be analyzed, denoted** $\textbf{Err}$:
  - $k = 1$: the NN query does not return $p_{(1)}$
  - $k > 1$: the NN query does not return $p_{(1)}, \ldots, p_{(k)}$
Qualifying the hardness of nearest neighbor queries

▶ Notations:
  ▶ Dataset $P = p_1, \ldots, p_n$
  ▶ Sorted dataset wrt $q$: $p(1), \ldots, p(n)$

$$
\Phi(q, P) = \frac{1}{n} \sum_{i=2}^{n} \frac{\|q - p(1)\|_2}{\|q - p(i)\|_2}.
$$

▶ Extreme cases:
  ▶ $\Phi \sim 0$: $p_1$ isolated, finding it should be easy
  ▶ $\Phi \sim 1$: points equidistant from $q$; finding $p(1)$ should be hard

▶ Rationale: in using RPT and spill trees with the defeatist search, the probability of success should depend upon $\Phi$. 
Generalizations of the function $\Phi$

- **Rationale:** function $\Phi$ shall be used for nodes containing a subset of the database.

- For a cell containing $m$ points – evaluate the remaining points in that cell:

  $$\Phi_m(q, P) = \frac{1}{m} \sum_{i=2}^{m} \| q - p(i) \|_2. \quad (3)$$

- If one is interested in the $k$ nearest neighbors – evaluate the remaining points too:

  $$\Phi_{k,m}(q, P) = \frac{1}{m} \sum_{i=k+1}^{m} \frac{\| q - p(1) \|_2 + \cdots + \| q - p(k) \|_2}{\| q - p(i) \|_2}. \quad (4)$$
Theoretical results on the performances

Agenda for the next lecture:

- RPTrees: success/failure probability to report NN
- Random projections and adaptation to intrinsic dimension
- NN, distances and concentration phenomena
Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
A geometric distance: the Hausdorff distance

▷ Hausdorff distance. Consider a metric space \((M, d)\). The Hausdorff distance of two non-empty subsets \(X\) and \(Y\) is defined by

\[
d_H(X, Y) = \max(H(X, Y), H(Y, X)), \quad \text{with } H(X, Y) = \sup_{x \in X} \inf_{y \in Y} d(x, y). \tag{5}
\]

Note that the one-sided distance is not symmetric, as seen on Fig. 1.

▷ Rmk. For closed set, the min distance is realized: inf becomes min; sup becomes max.

Figure: The one-sided Hausdorff distance is not symmetric
Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Comparing Histograms

▷ Bin-to-bin methods:

\[
d(H, K) = \sum_i | h_k - k_k |
\]  

→ overestimates the distance since neighboring bins are not considered.

▷ Mixing (e.g. quadratic) methods:

\[
d^2(H, K) = (h - k)^t A (h - k)
\]  

→ underestimates distances: tends to accentuate the similarity of color distributions without a pronounced mode.

▷ Illustrations:
Transport Plan Between Two Weighted Point Sets

- Weighted point sets:
  
  \[ P = \{(p_1, w_{p_1}), \ldots, (p_m, w_{p_m})\} \quad \text{and} \quad Q = \{(q_1, w_{q_1}), \ldots, (q_n, w_{q_n})\}. \quad (8) \]

  NB: nodes from \(P\) (resp. \(Q\)): production (resp. demand) nodes

  Shorthand for the sum of masses: \(W_P = \sum_i w_{p_i}, \ W_Q = \sum_j w_{q_j}\).

- A metric \(d(\cdot, \cdot)\): distance between two points \(d_{ij} = d(p_i, q_j)\).

- A transport plan: is a set of non-negative flows \(f_{ij}\) circulating on the edges of the bipartite graph \(P \times Q\).

Figure: Transport plan between two weighted point sets
The Earth Mover Distance: Definition

- **Optimization problem:**

  Minimize:  \[ C_{EMD-LP} = \sum_{ij} f_{ij} d_{ij} \]  under the constraints:

  \[
  \begin{cases} 
  (C1) f_{ij} \geq 0 \\
  (C2) \sum_j f_{ij} \leq w_{p_i}, \forall i \\
  (C3) \sum_i f_{ij} \leq w_{q_j}, \forall j \\
  (C4) \sum_i \sum_j f_{ij} = \min(W_P, W_Q) 
  \end{cases}
  \]  (10)

  These constraints read as follows:
  - (C1) Flows are positive
  - (C2,C3) A node cannot export (resp. receive) more than its weight.
  - (C4) The total flow neither exceeds the production nor the demand.

- **Earth mover distance:** defined from the cost by

  \[ d_{EMD-LP} = \frac{C_{EMD-LP}}{\sum_{ij} f_{ij}} = \frac{C_{EMD-LP}}{\min(W_P, W_Q)} \]  (11)

- **Advantages:**
  - Applies to signatures in general, the histograms being a particular case.
  - Embeds the notion of nearness, via the metric in the ground space.
  - Allows for partial matches. See however, the comment in section ??.
  - Easy to compute: linear program.

- **Ref:** Rubner, Tomasi, Guibas, IJCV, 2000
The Earth Mover Distance: Main Properties

▶ Theorems:
  ▶ Computed in polynomial time in the \# of variables
  ▶ Number of edges carrying flow is \( \leq n + m - 1 \)
  ▶ If \( W_P = W_Q \) and \( d(\cdot, \cdot) \) is a metric: EMD is also a metric

▶ Rmk: entropy regularized EMD distances, aka Sinkhorn distances, yield iterative algorithms – faster than LP solving. See refs. by M. Cuturi et al.
Application to image retrieval

- **Image coding, two options:**
  - convert image to histogram using a fixed binning of the color space; mass of bin: num. of pixel within it.
  - cluster pixels (say with k-means): mass of cluster is the fraction of pixels assigned to it

- **Search on DB of 20,000 images:** (a) $L_1$ (d) Quadratic form (e) EMD
Mallow’s Distance – p-th Wasswerstein metric

▷ Consider: two RV in $\mathbb{R}^d$: $X \sim P$, $Y \sim Q$.

▷ Mallows distance between $X$ and $Y$: minimum of expected difference between $X$ and $Y$ over all joint distributions $F$ for $(X, Y)$, such that the marginal of $F(X, \cdot)$ is $P$ and that of $F(\cdot, Y)$ is $Q$ (aka coupling):

$$M_p(X, Y)^p = \min_{F \in \mathcal{F}} \mathbb{E}_F[\|X - Y\|^p] : (X, Y) \sim F, X \sim P, Y \sim Q \}. \quad (12)$$

▷ Discrete setting: $P$ and $Q$

$$P = \{(x_1, w_{p_1}), \ldots, (x_m, w_{p_m})\} \quad (13)$$
$$Q = \{(y_1, w_{q_1}), \ldots, (y_n, w_{q_n})\}. \quad (14)$$

▷ Joint distribution is specified by probabilities on all pairs i.e. $F = \{f_{ij}\}$, and the fact that it respects the marginals yields:

$$\sum_j f_{ij} = p_i, \quad \sum_i f_{ij} = q_j, \quad \sum_{ij} f_{ij} = 1. \quad (15)$$

▷ Functional to be minimized becomes:

$$M_p(X, Y)^p = \mathbb{E}_F[\|X - Y\|^p] = \sum_{ij} f_{ij} \|x_i - y_j\|^p. \quad (16)$$
Mallow’s Distance versus EMD – Example with $p = 1$

- Mallows’ distance ($W_P = W_Q = 1$):
  $$M_p = \frac{1}{4} \times 0 + \frac{1}{4} \times 1 + \frac{1}{4} \times 1 + \frac{1}{4} \times 0$$

- EMD, assuming uniform weights on all points, ie $W_P = 2$ and $W_Q = 4$:
  $$EMD = 0$$ since a flow of 2 units satisfies all constraints.

![Diagram](image.png)

**Figure:** Mallows’s distance
References


Nearest Neighbors Algorithms in Euclidean and Metric Spaces: Algorithms and Data Structures

Introduction

Intermezzo: data vs algorithms

kd-trees and basic search algorithms

kd-trees and random projection trees: improved search algorithms

Important metrics: geometry based

Important metrics: the Earth Mover Distance

Metric trees and variants
Metric spaces

**Definition 1.** A *metric space* is a pair \((M, d)\), with \(d : M \times M \to \mathbb{R}^+\), such that:

- **(1) Positivity:** \(d(x, y) \geq 0\)
- **(1a) Self-distance:** \(d(x, x) = 0\)
- **(1b) Isolation:** \(x \neq y \implies d(x, y) > 0\)
- **(2) Symmetry:** \(d(x, y) = d(y, x)\)
- **(3) Triangle inequality:** \(d(x, y) \leq d(x, z) + d(y, z)\)

**Product metric.** Assume that for some \(k > 1\):

\[M = M_1 \times \cdots \times M_k.\]  

(17)

and that each \((M_i, d_i)\) is a metric space. For \(p \geq 1\), the *product metric* is:

\[d(x, y) = \left( \sum_{k=1}^{k} d_i(x_i, y_i)^p \right)^{1/p}\]  

(18)

Some particular cases are:

- **\((M_i = \mathbb{R}, d_i = | \cdot |)\):** \(L_p\) metrics.
- **\(p = 1, d_i = \text{uniform metric}:** Hamming distance.
Using the triangle inequality

**Lemma 2.** For any three points $p, q, s \in M$, for any $r > 0$, and for any point set $P \subset M$, one has:

$$|d(q, p) - d(p, s)| \leq d(q, s) \leq d(q, p) + d(p, s)$$  \hspace{1cm} (19)

$$d(q, s) \geq d_P(q, s) := \max_{p \in P} |d(q, p) - d(p, s)|$$  \hspace{1cm} (20)

$$\begin{cases} 
  d(p, s) > d(p, q) + r \Rightarrow d(q, s) > r \\
  d(p, s) < d(p, q) - r \Rightarrow d(q, s) > r.
\end{cases}$$  \hspace{1cm} (21)

*Figure: Lower bound from the triangle inequality, see Lemma 2*
Metric tree: definition

Definition:

- A binary tree
- Any internal node implements a spherical cut defined by the distance $\mu$ to a pivot $v$
  - right subtree: points $p$ such that $d(pivot, p) \geq \mu$
  - left subtree: points $p$ such that $d(pivot, p) < \mu$

Figure: Metric tree for a square domain (A) One step (B) Full tree
Metric tree: construction

- Recursively construction:
  - Choose a pivot, ideally inducing a partition into subsets of the same size
  - Assign points to subtrees and recurse
  - Complexity under the balanced subtrees assumption: $O(n \log n)$.

---

**Algorithm 1** Algorithm build\_MetricTree($S$)

```plaintext
{build\_MetricTree($S$)}
if $S = \emptyset$ then
  return NIL
$n \leftarrow newNode$
Draw at random $Q \subset S$ and $v \in Q$
$n.pivot \leftarrow v$
$\mu \leftarrow \text{median} \{d(v, p), p \in Q \setminus \{v\}\}$
{The pivot splits points into two subsets}
$L \leftarrow \{s \in S \setminus \{p\} | d(s, v) < \mu\}$
$R \leftarrow \{s \in S \setminus \{p\} | d(s, v) \geq \mu\}$
{For each subtree: min/max distances to points in that subtree}
$n.(d_1, d_2) \leftarrow (\text{min}, \text{max}) \text{ of distances } d(v, p), p \in L$
$n.(d_3, d_4) \leftarrow (\text{min}, \text{max}) \text{ of distances } d(v, p), p \in R$
{Recursion}
$n.L \leftarrow \text{build\_MetricTree}(L)$
$n.R \leftarrow \text{build\_MetricTree}(R)$
```
Searching a metric tree: algorithm

**Algorithm 2**

```plaintext
search_MetricTree( T, q )
```

{Note of T is denoted n}

```plaintext
nn(q) ← ∅
τ ← ∞
if n = NIL then
    return
{Check whether the pivot is the nn}
    l ← d(q, n.pivot)
if l < τ then
    nn(q) ← n.pivot
    τ ← l
{Dilate the distance intervals for left and right subtrees}
    I_l ← [n.d_1 - τ, n.d_2 + τ]
    I_r ← [n.d_3 - τ, n.d_4 + τ]
if l ∈ I_l then
    search_MetricTree(n.L, q)
if l ∈ I_r then
    search_MetricTree(n.R, q)
```
Searching a metric tree: correctness – pruning lemma

Lemma 3. Consider the intervals associated with a node, as defined in Algorithm 1, that is \( l_i \leftarrow [n.d_1 - \tau, n.d_2 + \tau] \) \( l_r \leftarrow [n.d_3 - \tau, n.d_4 + \tau] \). Then:
(1) If \( l \not\in l_i \), the left subtree can be pruned.
(2) If \( l \not\in l_r \), the left subtree can be pruned.

Proof.
We prove (1), as condition (2) is equivalent. Let us denote \( l_L = [d_1, d_2] \). Since \( l = d(v, q) \not\in l_i \), we have \( d(v, q) < d_1 - \tau \) and \( d(v, q) > d_2 + \tau \). We analyze these two conditions in turn.

\( \triangleright \) Condition on the right hand side. By definition of \( d_2 \), with \( v \) the pivot, we have:
\[
\forall p \in L : d(v, q) > d(v, p) + \tau.
\]
Using the triangle inequality for \( d(v, q) \) yields
\[
d(v, p) + d(p, q) \geq d(v, q) > d(v, p) + \tau \Rightarrow d(q, p) > \tau.
\]

\( \triangleright \) Mutatis mutandis. \( \square \)
Metric tree: choosing the pivot

- By the pruning lemma: for small $\tau$ and if $q$ is picked uniformly at random, the measure of the boundary of the spheres of radius $d_1, \ldots, d_4$ determines the probability that no pruning takes place.
  $\Rightarrow$ pick the pivot so as to minimize this measure.

- Example in 2D: 3 choices for the pivot, so as to split the unit square (mass: 1) into two regions of equal size (mass: 1/2)

- Choice of pivots (illustrated using $\mu$ (rather than the $d_i$s):
  - Best pivot: $p_c$
  - Worst pivot: $p_m$

Figure: Metric trees: minimizing the measure of boundaries.
From metric trees to metric forests

Search options:
▶ (I) The exact search, based on the pruning lemma.
▶ (II) The defeatist style search: visit one subtree only

Compromising speed versus accuracy
▶ (I) Exact, but possibly costly if little/no pruning occurs. Worst-case: linear time.
▶ (II) Faster, but error prone.
▶ Compromise: using a forest of trees rescues erroneous branching decisions in the course of the defeatist search.

Figure: Metric forest
References


