summary

LAST TIME

$x \sim p(x)$ x: RANDOM VARIABLE in $\mathbb{R}^d$

- GOAL: given $\{x_i\}_{i=1}^N$, reconstruct $P_N(x) \approx p(x)$

- KERNEL DENSITY ESTIMATION (KDE)

$$P_N(z) = \frac{1}{N h^d} \sum_{i=1}^{N} k\left(\frac{z - x_i}{h}\right)$$

h: BANDWIDTH {THIS ASSUMES CONSTANT BANDWIDTH}

EXAMPLES

- $k(r) = \begin{cases} 1/2, & \|r\| < 1 \\ 0, & \text{otw} \end{cases}$, $r \in \mathbb{R}^d$

NAIVE

- h: $h(z) = d_k(z)$: dist

NEAREST NEIGHBOR

- $k(r) = (2\pi)^{-d/2} \exp\left(-r^T r\right)$

GAUSSIAN.

HYPERPARAMETER(S) : h, K

- choose BY . $\longrightarrow$ REGULARIZED LOG LIKEHOOD
  . $\longrightarrow$ EXPECTATION OF RMS (MISE)

SMALL $h, K \longrightarrow$ overfitting $\nearrow$ DOES NOT GENERALIZE WELL
$\rightarrow$ SMALL BIAS, LARGE VARIANCE

CAN ANALYTICALLY COMPUTE h_{opt}

- ISSUES WITH DIMENSIONALITY

  - PROBABILITY OF FINDING A POINT IN A VOLUME OF A CUBE OF LENGTH S.

  - $\frac{\|P_N(0) - p(0)\|}{\|p(0)\|}$ FOR NORMAL GAUSSIAN WITH $h_{opt} = \left(\frac{4}{d+2}\right)^{1/(d+4)} N^{-\frac{1}{d+4}}$

    FIND N so that error $< 10\%$ $d=1$ $d=5$ $d=10$.

IN PRACTICE : How to choose h. ?

SPLIT TRAINING SET 80% 20%
TRAIN TEST
X_N X_T

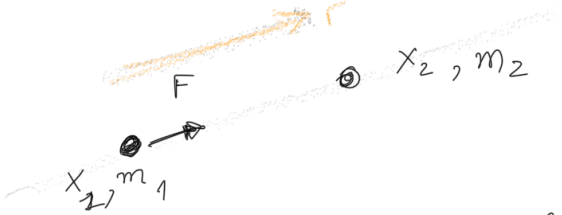
$K_{ij} = K(X_{T,i}, X_{N,j})$

FIND h (or K) such that
- log likelihood of $P_N(X_T \mid X_N, h)$ is max

$h = \arg\max_h \text{sum}\left(\log\left(K\left[\frac{i}{Nh}\right]\right)\right)$

PHYSICS

GRAVITATIONAL INTERACTIONS



$$r = x_2 - x_1$$

$$d = \frac{r}{\|r\|_2}$$

$$F = G\frac{m_1 m_2}{\rho^2} \quad \overset{G=1}{d=}, \quad F(x_1) = \frac{m_1 m_2}{\|r\|_2^2}\frac{r}{\|r\|_2}$$

IF WE HAVE N-OBJECTS

$$F_1 = m_1 \sum_{j=2}^{N} \frac{x_j - x_1}{\|x_1 - x_j\|^3} m_j = m_1 \sum_{j=2}^{N} k(x_1, x_j) m_j$$

Typically we have N particles

$$\begin{cases} m_i \frac{dV_i}{dt} = F_i = m_i \sum_{\substack{j=1 \\ j\neq i}}^{N} k(x_i, x_j) m_j \\ \vdots \\ \frac{dx_i}{dt} = V_i \end{cases}$$

Kernel matrix: $K_{ij} = k(x_i, x_j)$

THREE PROBLEMS

~ INNER PRODUCT SPACES & THE KERNEL TRICK

$$Ka = y \Rightarrow a = K^{-1}y \quad ; \quad \rightarrow \text{PREDICT}$$

~ PROBABILITY : $K1$

~ PHYSICS : Km

IN GENERAL DURING TRAINING :

$$K \in \mathbb{R}^{N \times N}$$

N : TRAINING SET

typically K is Semi PD.

DURING TESTING

$$K \in \mathbb{R}^{M \times N}$$

M : TESTING SET

K CAN BE SPARSE ($\mathcal{O}(N)$ NON ZEROS)

- OR MOST OFTEN (IN ALL THESE APPLICATIONS)

DENSE $\frac{|K_{ij}|}{\max|K_{ij}|}$ significant.

- COMPLEXITY FOR DENSE MATRICES
  - MATVEC: $O(N^2)$
  - LINEAR SOLVE: $O(N^2 \sqrt{\text{cond}(K)})$
    $O(N^3)$ → CHOLESKY

Prohibitively expensive

$N = 10^9$ REQUIRES 10^5 GPUs RUNNING AT 3 TFLOPS.
[$ see's]

SCALABLE ALGORITHMS

- LOW-RANK APPROXIMATION
- BLOCK-LOW-RANK APPROXIMATION
- N-BODY ALGORITHMS
- PARALLELIZATION
- NEAREST NEIGHBORS.

Low-Rank Matrices

Suppose that $K \in \mathbb{R}^{M \times N}$ is low-rank (exact or numerically)

Let $U \Sigma V^T$ be its reduced SVD

- Every matrix has one
- Check properties of SVD

Then we say K has rank r if

$$K = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_r \\ | & & | \end{bmatrix}_{M \times r} \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}_{r \times r} \begin{bmatrix} — & v_1^T & — \\ & \vdots & \\ — & v_r^T & — \end{bmatrix}_{r \times N} = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

We say that K is approximately rank r if

$$\min_{\tilde{K}} \| K - \tilde{K} \|_2 \ll \sigma_r \quad \text{and } \tilde{K} \text{ has rank } r$$

Turns out r-truncation of SVD results in optimal approximation, $\|K - K_r\|_2 = \sigma_{r+1}$

spectral gap

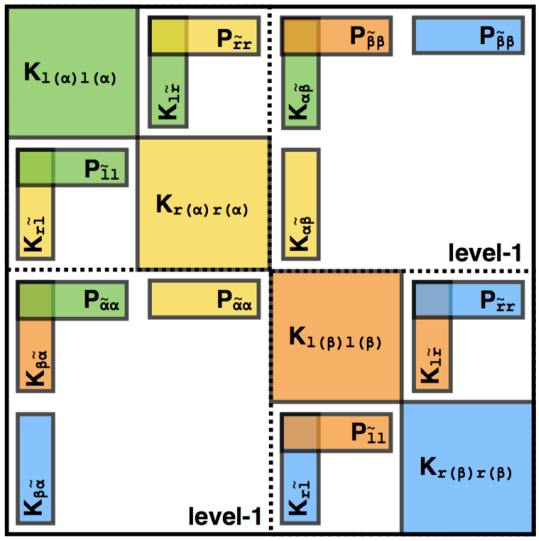
→ Complexity of matvec
→ Cost of computing SVD?

THE OTHER REGIME:
Truncated kernels, like the nearest neighbor:
These are sparse kernels.
Once we form the matrix, the matvec is O(Nk).
But then how do we discover the sparsity?

THE REALITY; KERNEL MATRICES ARE NEITHER LOW-RANK. NEITHER SPARSE. The best parameters typically result in matrices that their numerical rank is huge (so formally not low-rank).

HIERARCHICAL APPROXIMATIONS.

Hierarchical matrices



$$\begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} = \begin{bmatrix} G_{11} & 0 \\ 0 & G_{22} \end{bmatrix} + \begin{bmatrix} 0 & G_{12} \\ G_{21} & 0 \end{bmatrix}$$

$$D + UV$$

$$/$$

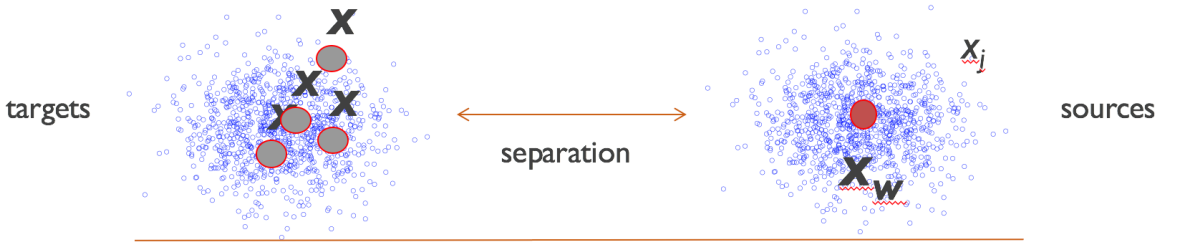
$$D + UV$$

rank "s"

$$\mathcal{O}(N^2) \rightarrow \mathcal{O}(N \log N)$$

NBODY 101

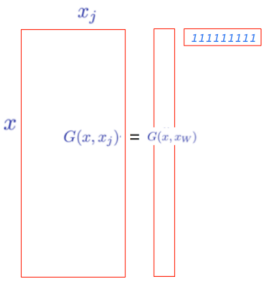
Idea I: Far-field interactions



x: Target, x_j: sources
w_j: weights

$u(x) = \sum_j G(x, x_j) w_j$

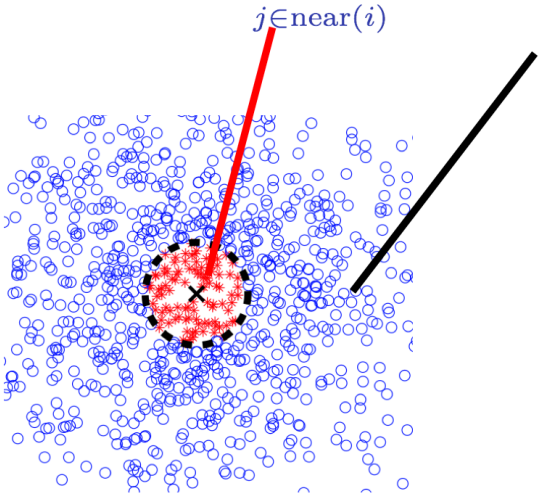
1. compute $W = \sum_i w_j$
2. choose x_W
3. $u(x) \approx G(x, x_W) W$



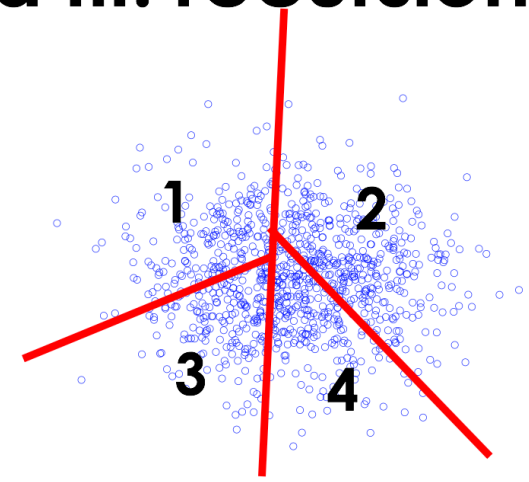
LOW RANK APPROXIMATION
$O(N^3) \rightarrow O(N)$

Idea II: Near-/Far-field split

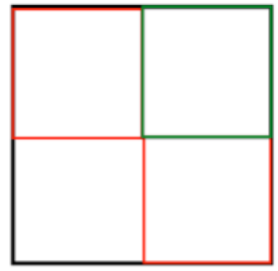
$$u_i = \sum_{\substack{j=1 \\ j \neq i}}^{N} G(x_i, x_j) w_j = \sum_{j \in \text{near}(i)} G_{ij} w_j + \sum_{j \in \text{far}(i)} G_{ij} w_j$$

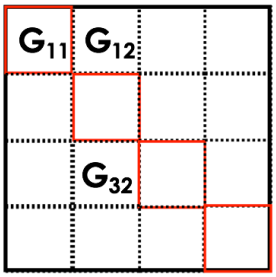


Idea III: recursion



R^d

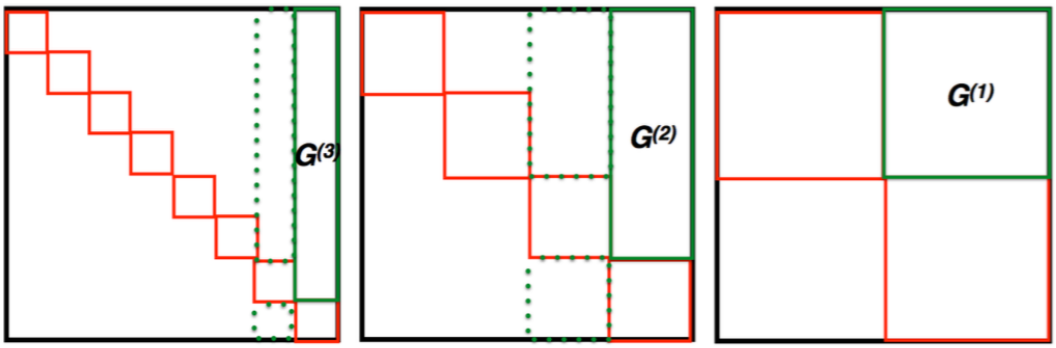




Matrix partitioning

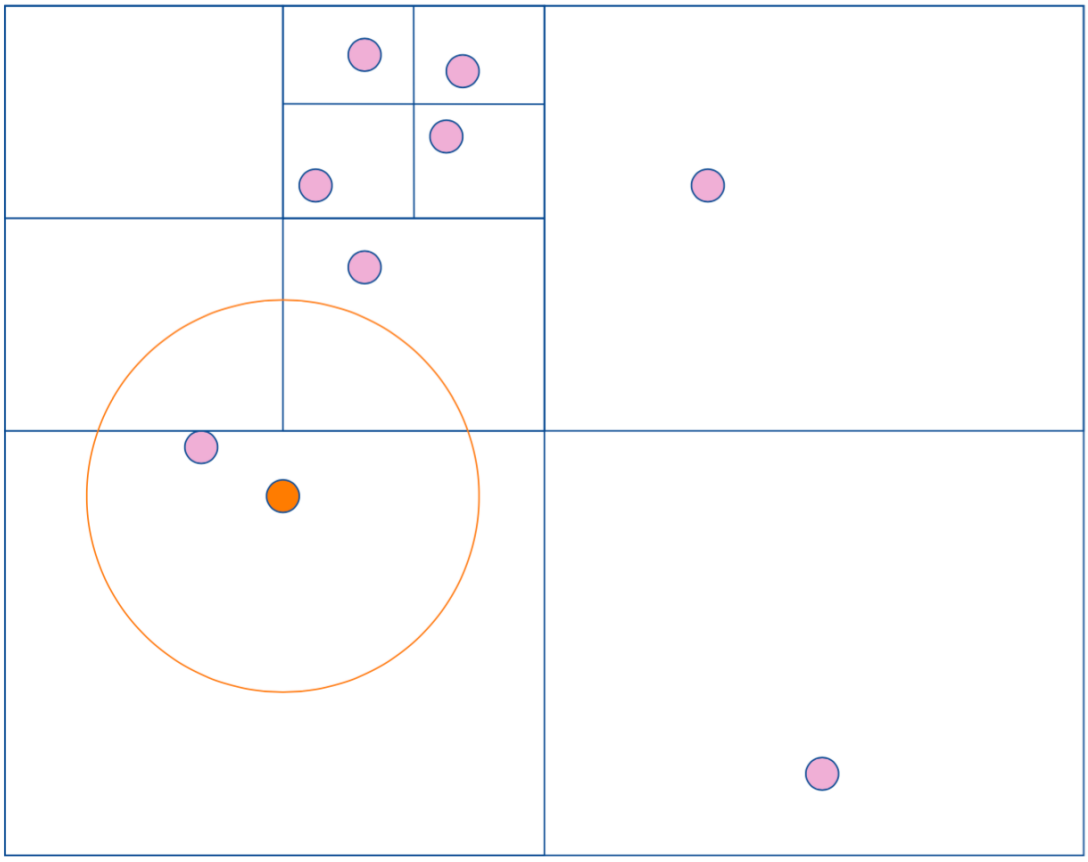
Algebraic view

$$u_i = \sum_{j \in \text{diagonal}(i)} G_{ij} w_j + \sum_{j \in \text{off-diagonal}(i)} G_{ij} w_j$$



(a) **Level 3.** (b) **Level 2.** (c) **Level 1.**

The NEAREST NEIGHBORS PROBLEM:
DIRECT EVALUTION of distances O(N) complexity



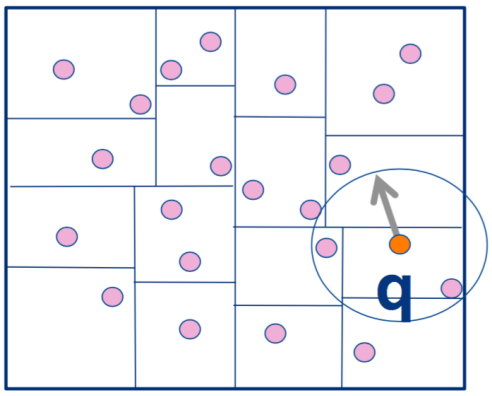
Exact search O(n) per query point

In low dimensions can be O(1) per query point

- Preprocess points to spatial data structure
  - pruning => O(1) cost

PRUNING

1. Construct tree (group points)
2. Insert *q* into tree
3. Find nearest neighbor in the leaf
4. Range search and pruning



Issues with regular decompositions

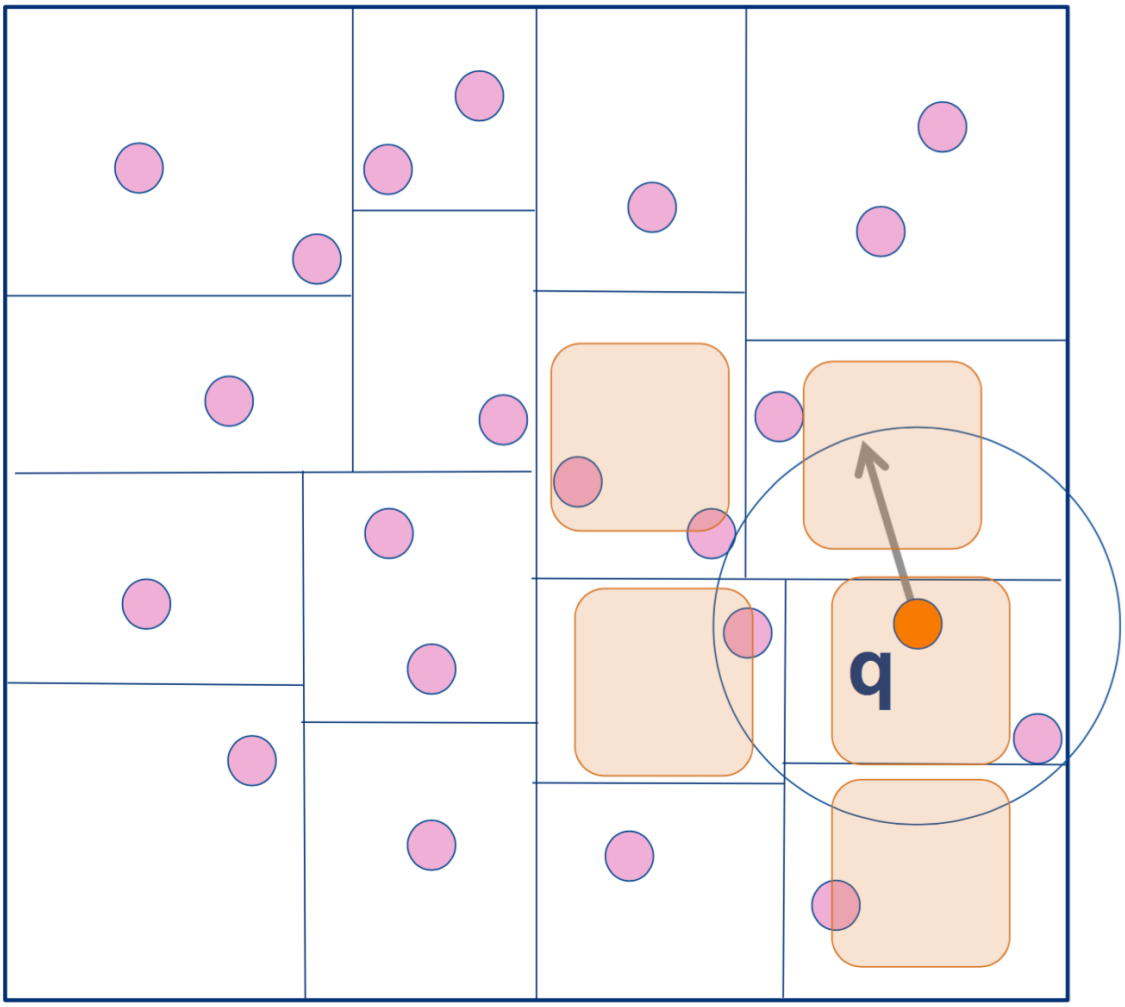
nearest-neighbor problems

high-dimensional problems

- 2^d per division

alternative data structures

- popular: KD-tree



Range query (q, R)

Traverse tree bottom-up

- if node is a leaf
  - do direct range query on the points
- visit node
  - If BoundingBox(node) INTERSECTION Ball(q, R) == Empty
    - TERMINATE RECURSION
  - else visit children

Nearest neighbor query(q)

leaf_q = insert_point(q, root)

- like quadtrees
- traverse tree top down and insert point
- p=find nearest neighbor in "leaf_q" using direct search
- R_initial = distance(leaf_q, p)
- traverse tree top down using range search with R_initial and updating R_initial if a point p' closer to q is found

Back to NN-search: some negative results, d>>3

Main problem: r-search or k-search require ball around query point to use for pruning.

In high dimensions (say d>100) pruning is negligible and the complexity is O(n) per query point

- For uniform or normal distributions, exact searches, and any tree that uses convex grouping (Weber, Shek, Blott '98)

$O(n^2)$ for *all nearest neighbor problems*

Consider all nearest neighbors

Iterate

- choose rotation
- rotate coordinates
- build KDT tree
- find NNs of each point in its leaf
- merge with existing NNs set of each point

1: $K = \emptyset, \mathcal{T} =$`root`
2: $Q =$ RANDOMROTATION
3: $p = Qr$
4: NODESPLIT($\mathcal{T}, p$)
5: $F =$LEAFLOCALSEARCH($\mathcal{T}$)
6: $K =$UPDATENEIGHBORS(K,F)
7: `if` CONVERGED(K,F) `return`
8: GOTO 2