K-D Trees and KNN Searches

Or, “How Do We Figure Out What Nodes Are In Our Stencil?”

Naïve Nearest Neighbor Searches

- For every point
  - Find the distance to ever other point
  - Sort all those Distances
  - Take the points corresponding to the K smallest

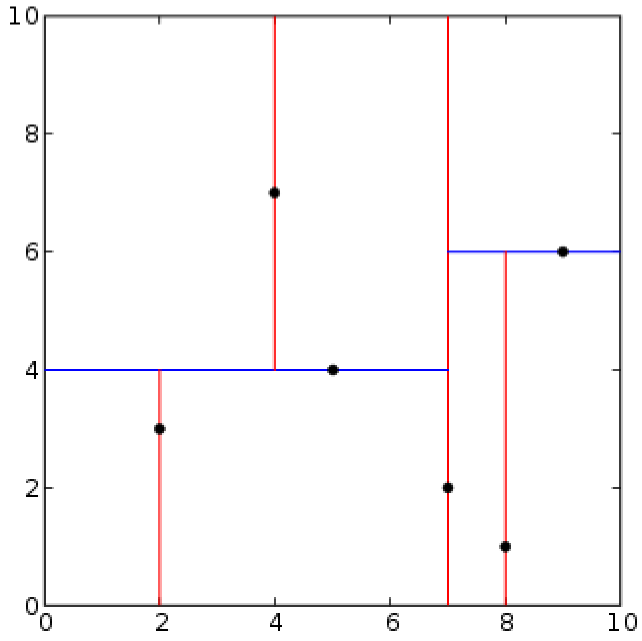
Naïve Nearest Neighbor Searches

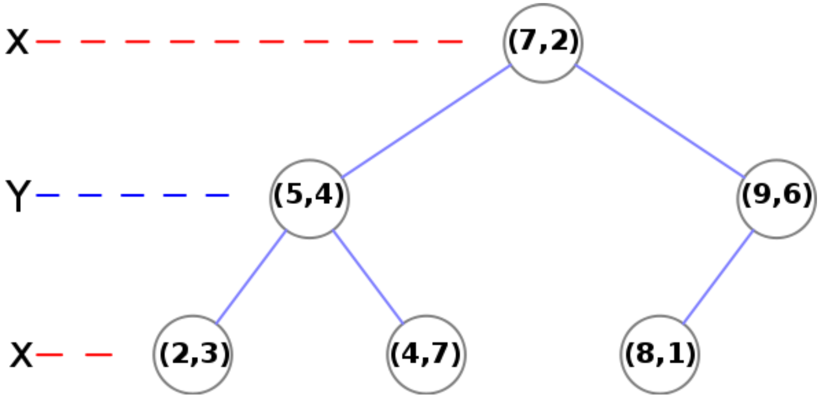
- For every point (N)
  - Find the distance to ever other point (N)
  - Sort all those Distances (N log N – N^2)
  - Take the points corresponding to the K smallest (1)

- Total time O(N^3)

KD Tree

- Bisecting structure
- Each branchpoint is the median in some dimension
  - One set of descendants are to one side, and one to the other
- Cycle the dimensions





Example via Wikipedia, calculated by users KiwiSunset and MYguel, 2006 and 2008, respectively

KD Tree Construction

- Median finding is expensive
  - O(N) or O(N log N)
  - Sometimes a random subset is sorted and used to serve as splitting planes, and the rest are just fitted in there
  - Lose balance guarantees (necessary for strict complexity proofs for some operations), but faster to construct
  - Often balanced in practice

KD Tree Construction

- Adding Elements
  - Can add elements dynamically, but it’s a bad idea to construct the original tree this way
  - Can break balance, and (AFAIK) not implemented in MATLAB
  - Can be helpful for “online” applications
  - Traverse down the tree, staying in a region where the new point should be located
    - When you reach a leaf go to one side or the other accordingly

Nearest Neighbor Search on a KD Tree

- For Each Point:
  - Start at the root
  - Traverse the Tree to the section where the new point belongs
  - Find the leaf; store it as the best
  - Traverse upward, and for each node;
    - If it's closer, it becomes the best
    - Check if there could be yet better points on the other side:
      - Fit a sphere around the point of the same radius as distance to current best
      - See if that sphere goes over the splitting plane associated with the considered branchpoint
    - If there could be, go down again on the other side. Otherwise, go up another level
- O(N log N)

K Nearest Neighbor Search on a KD Tree

- For Each Point:
  - Start at the root
  - Traverse the Tree to the section where the new point belongs
  - Find the leaf; store it as the first element in the “Best” queue
  - Traverse upward, and for each node;
    - Put it at the proper point of the “Best” queue
    - Check if there could be yet better points on the other side:
      - Fit a sphere around the point of the same radius as distance to last element in “Best” queue
      - See if that sphere goes over the splitting plane associated with the considered branchpoint
    - If there could be, go down again on the other side. Otherwise, go up another level
- A bit worse

KNN Complexity

- Building the Tree: O(D N log N), but constant can be large
- Searching the Tree: ~ log N per query point, so ~ N log N in total