Exact Nearest Neighbor Algorithms
Sliding sums

- Suppose we want to “smooth” a histogram, i.e. replace the values by the average over a window
  - How can we do this efficiently?

```plaintext
current_sum = window_sum

current_sum = window_sum + (-5) + (0)

current_sum = window_sum + (-2) + (3)
```
Sabermetrics

- One of the best players ever
  - .310 batting average
  - 3,465 hits
  - 260 home runs
  - 1,311 RBIs
  - 14x All-star
  - 5x World Series winner

- Who is the next Derek Jeter?

Derek Jeter

Sabermetrics

- Classic example of nearest neighbor application
- Hits, Home runs, RBIs, etc. are dimensions in “Baseball-space”
  - Every individual player has a unique point in this space
- Problem reduces to finding closest point in this space
POI Suggestions

- Simpler example, only 2d space
- Known set of interest points in a map - we want to suggest the closest
  - Note: we could make this more complicated; we could add dimensions for ratings, category, newness, etc.
- How do we figure out what to suggest?
  - Brute force: just compute distance to all known points and pick lowest
    - $O(n)$ in the number of points
    - Feels wasteful… why look at the distance to the Eiffel Tower when we know you’re in NYC?
  - Space partitioning
2d trees
2d trees
2d trees

![Diagram of 2d trees with marked points and connections.](image-url)
2d trees

[Diagram of 2d trees with coordinates marked: (4,3), (1,4), (6,2), (2,2), (3,5), (7,2), (8,5)]
2d trees
2d trees - Nearest Neighbor

Diagram showing a 2-dimensional tree structure with points and connections. The tree is composed of nodes labeled with coordinates: (1,4), (4,3), (6,2), (2,2), (3,5), (7,2), and (8,5). The points on the grid are marked with 'x' and lines connect the nodes in the tree structure.
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

Diagram showing a 2D space with points marked at (1,4), (2,2), (3,5), (4,3), (6,2), (7,2), and (8,5). A tree structure is also shown with nodes at the points mentioned.
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

![Diagram of 2d trees with nearest neighbor examples](image)
2d trees - Nearest Neighbor

Graph showing points in a 2D plane with coordinates and distances between points.
2d trees - Nearest Neighbor

Diagram showing a 2D tree structure with points marked on a grid and a corresponding tree structure to the right. Points include (1,4), (4,3), (6,2), (2,2), (3,5), (7,2), and (8,5). The distance is marked as $\sqrt{5}$.
2d trees - Nearest Neighbor

![Diagram showing 2D trees and nearest neighbor algorithm.](image-url)
2d trees - Nearest Neighbor

![Graph showing 2d trees and nearest neighbor points.](image)
2d trees - Nearest Neighbor

![Diagram of 2d trees - Nearest Neighbor]
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

(1,4) -> (2,2) <- (4,3) -> (8,5)
(3,5) -> (7,2)

√2
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

![Diagram of 2d trees and nearest neighbor search](image)
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

Diagram showing a 2D tree structure with points marked on a grid. The tree is used to illustrate the concept of nearest neighbor in 2D space.
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor
2d trees - Nearest Neighbor

(4,3) \sqrt{5}
(1,4)
(6,2) \sqrt{10}
(2,2)
(3,5)
(7,2)
(8,5) 3
2d trees - Nearest Neighbor

Diagram showing a 2D Cartesian plane with labeled points and a tree structure indicating nearest neighbor relationships.

- Points: (1,4), (6,2), (2,2), (3,5), (4,3), (7,2), (8,5)
- Distances: √5, √10, √17

The tree structure illustrates the nearest neighbor relationships between these points.
2d trees - Nearest Neighbor

![Diagram showing 2d trees with nearest neighbor examples and distances labeled with square roots, such as \( \sqrt{5} \) and \( \sqrt{17} \).]
2d trees - Nearest Neighbor

Diagram showing a 2D tree structure with points marked as 'x' and distances calculated with square roots. The tree is rooted at (4,3) and branches out to (1,4), (6,2), (2,2), (3,5), (7,2), and (8,5). Distances are calculated as √17, √5, √10, and 3.
2d trees - Nearest Neighbor

Diagram showing a 2D tree structure with points and distances.
2d trees - Nearest Neighbor

![Diagram showing 2D trees and nearest neighbor concept.](image)
2d trees - Nearest Neighbor

![Diagram of 2D tree structures with nearest neighbor points and distances.]

- Nearest neighbors:
  - (4,3) to (1,4) distance: √5
  - (1,4) to (6,2) distance: √10
  - (3,5) to (2,2) distance: √18
  - (7,2) to (8,5) distance: 3√18
  - (2,2) to (3,5) distance: 2√17
2d trees

- **Construction complexity**: $O(n \log n)$
  - Complication: how do you decide how to partition?
  - Requires you be smart about picking pivots

- **Adding/Removing element**: $O(\log n)$
  - This is because we know it's balanced from the median selection
  - ...except adding/removing might make it unbalanced -- there are variants that handle this

- **Nearest Neighbor**
  - Average case: $O(\log n)$
  - Worst case: $O(n)$
    - Not great… but not worse than brute force

- Can also be used effectively for range finding
$k$-d trees

- 2d trees can be extended to $k$ dimensions
**$k$-d trees**

- Same algorithm for nearest neighbor!
  - Remember sabermetrics!
  - ...except there’s a catch

- Curse of Dimensionality
  - The higher the dimensions, the “sparser” the data gets in the space
  - Harder to rule out portions of the tree, so many searches end up being fancy brute forces
  - In general, $k$-d trees are useful when $N \gg 2^k$
Sabermetrics (reprise)

- Finding single neighbor could be noise-prone
  - Are we sure this year’s “Derek Jeter” will be next year’s too?
  - What if there are lots of close points... are we sure that the relative distance matters?
  - Could ask for set of most likely players

- Alternate question: will a player make it to the Hall of Fame?
  - Still k-dimensional space, but we’re not comparing with individual point
  - Am I in the “neighborhood” of Hall of Famers?
  - Classic example of “classification problem”
**k-Nearest Neighbors (kNN)**

- New plan: find the *k* closest points
  - Each can “vote” for a classification
  - …or you can do some other kind of averaging

- Can we modify our k-d tree NN algorithm to do this?
  - Track *k* closest points in max-heap (priority queue)
    - Keep heap at size *k*
  - Only need to consider *k*’th closest point for tree pruning
Voronoi Diagrams

- **Useful visualization of nearest neighbors**
  - Good when you have a known set of comparison points

- **Wide ranging applications**
  - Epidemiology
    - Cholera victims all near one water pump
  - Aviation
    - Nearest airport for flight diversion
  - Networking
    - Capacity derivation
  - Robotics
    - Points are obstacles, edges are safest paths

Voronoi Diagrams

- Also helpful for visualizing effects of different distance metrics

Euclidean distance

Manhattan distance

Voronoi Diagrams

- Polygon construction algorithm is a little tricky, but conceptually you can think of expanding balls around the points.

**k-means Clustering**

- **Goal:** Group \( n \) data points into \( k \) groups based on nearest neighbor

**Algorithm:**

1. Pick \( k \) data points at random to be starting “centers,” call each center \( c_i \)
2. For each node \( n \), calculate which of the \( k \) centers is the nearest neighbor and add it to set \( S_i \)
3. Compute the mean of all points in \( S_i \) to generate a new \( c_i \)
4. Go back to (2) and repeat with the new centers, until the centers converge
Notice: the algorithm basically creates Voronoi diagrams for the centers!

**k-means clustering**

- Does this always converge?
  - Depends on distance function. Generally yes for Euclidean
  - Converges quickly in practice, but worst case can take an exponential number of iterations

- Does it give the optimal clustering?
  - NO! Well, at least not always.
Other space partitioning data structures

- **Leaf point k-d trees**
  - Only stores points in leaves, but leaves can store more than one point
  - Split space at the middle of longest axis
  - Effectively “buckets” points - can be used for approximate nearest neighbor

- **Quadtrees**
  - Split space into quadrants (i.e. every tree node has four children)
  - Quadrant can only contain at most $q$ nodes
    - If there are more than $q$, split that quadrant again into quadrants
  - Applications
    - Collision detection (video games)
    - Image representation/processing (transforming/comparing/etc. nearby pixels)
    - Sparse data storage (spreadsheets)
  - Octrees are extension to 3d