Instance Based Learning

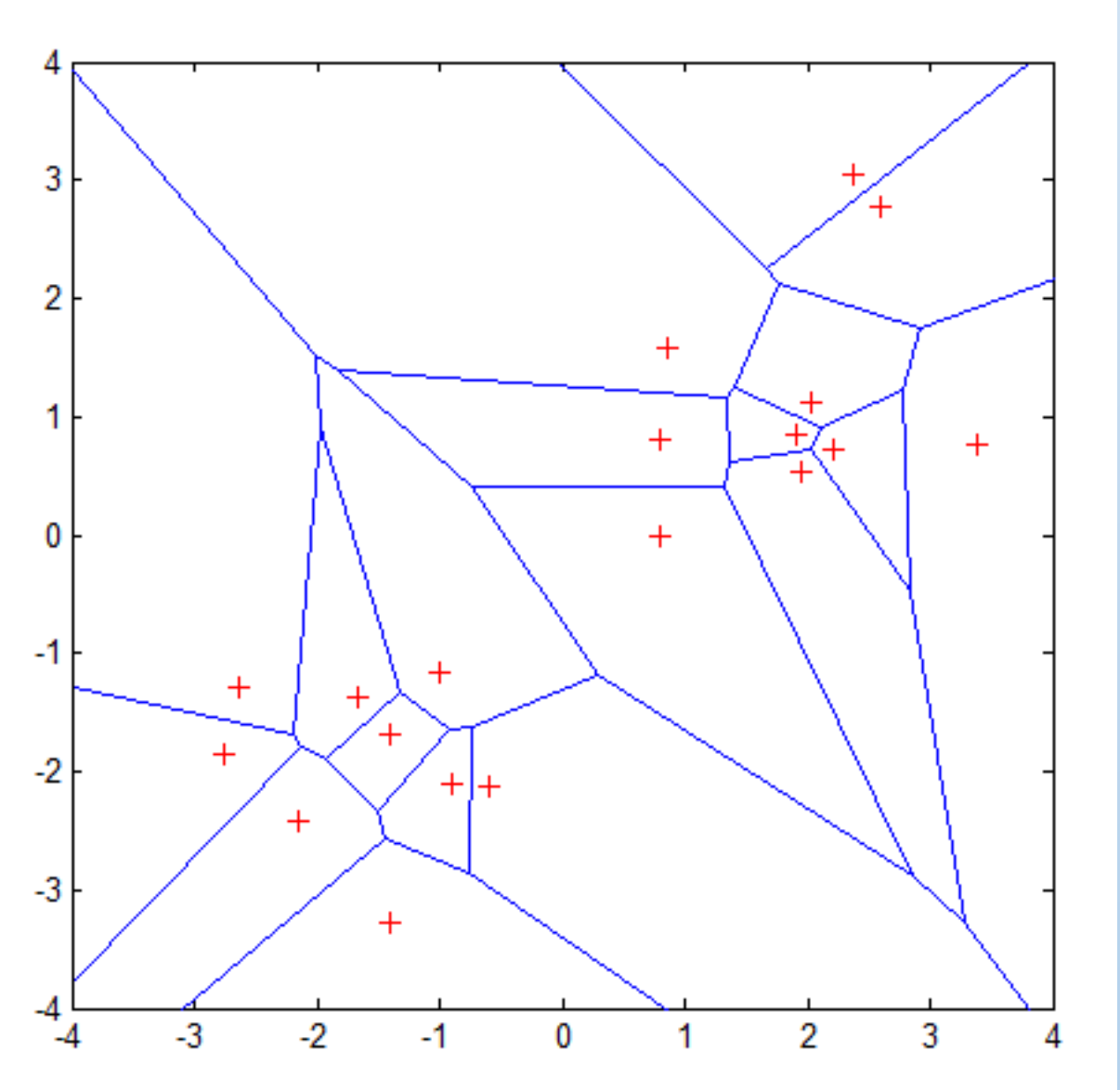
Introduction

- Instance-based learning is often termed *lazy* learning, as there is typically no “transformation” of training instances into more general “statements”
- Instead, the presented training data is simply stored and, when a new query instance is encountered, a set of similar, related instances is retrieved from memory and used to classify the new query instance
- Hence, instance-based learners never form an explicit general hypothesis regarding the target function. They simply compute the classification of each new query instance as needed

k-NN Approach

- The simplest, most used instance-based learning algorithm is the *k*-NN algorithm
- *k*-NN assumes that all instances are points in some *n*-dimensional space and defines neighbors in terms of distance (usually Euclidean in R-space)
- *k* is the number of neighbors considered

Graphic Depiction



Properties:

1) All possible points within a sample's Voronoi cell are the nearest neighboring points for that sample
2) For any sample, the nearest sample is determined by the closest Voronoi cell edge

Basic Idea

- Using the second property, the *k*-NN classification rule is to assign to a test sample the majority category label of its *k* nearest training samples
- In practice, *k* is usually chosen to be odd, so as to reduce the probability of ties
- The *k* = 1 rule is generally called the nearest-neighbor classification rule – which we used to classify the MNIST data

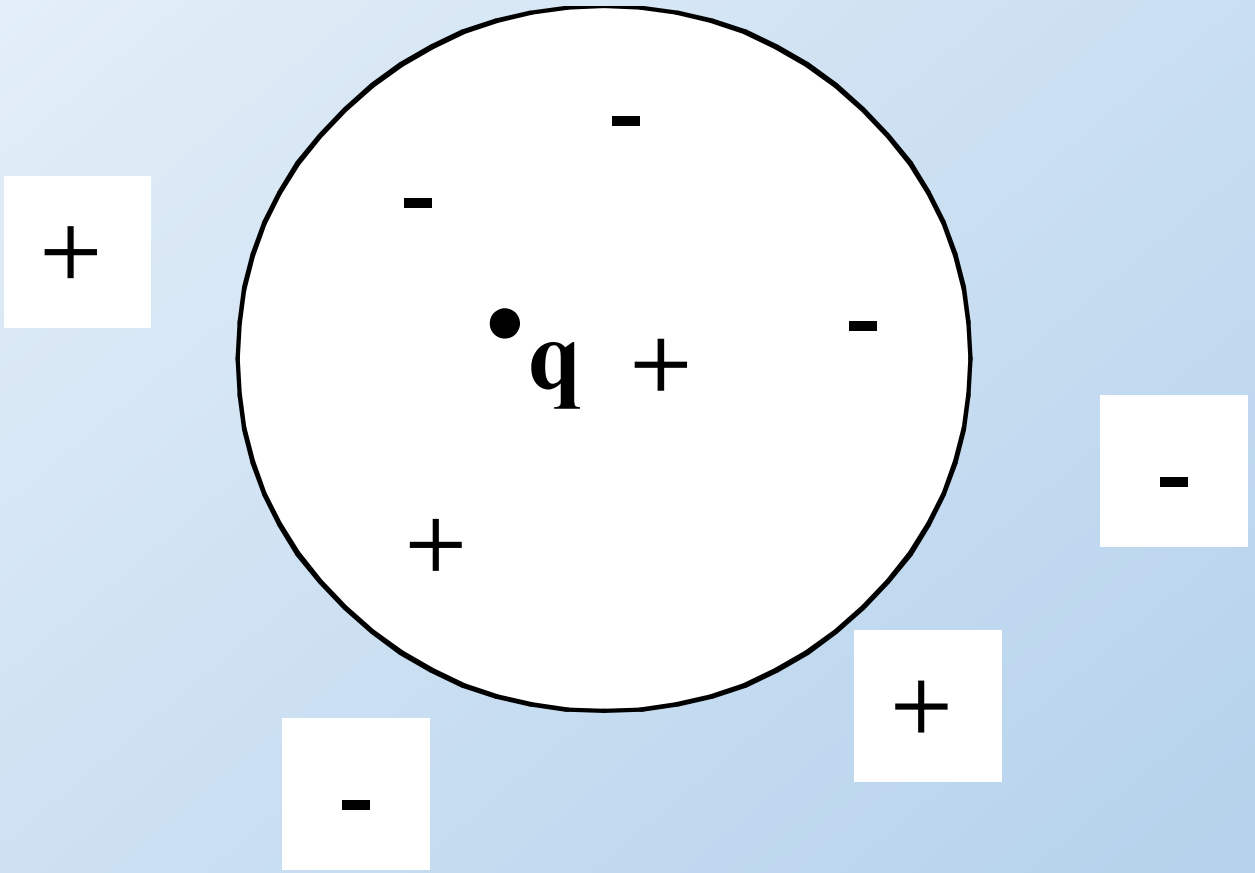
k-NN Algorithm

- For each training instance $t=(x, f(x))$
  - Add t to the set *Tr_instances*
- Given a query instance q to be classified
  - Let $x_1, \ldots, x_k$ be the k training instances in *Tr_instances* nearest to q
  - Return

$$\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$$

- Where V is the finite set of target class values, and $\delta(a,b)=1$ if $a=b$, and 0 otherwise (Kronecker function)
- Intuitively, the *k*-NN algorithm assigns to each new query instance the majority class among its *k* nearest neighbors

Simple Illustration



q is + under 1-NN, but – under 5-NN

Distance-weighted *k*-NN

- Replace $\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} d(v, f(x_i))$

by:

$$\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} \frac{1}{d(x_i, x_q)^2} d(v, f(x_i))$$

Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier

Standardization

- Transform raw feature values into z-scores

$$z_{ij} = \frac{x_{ij} - m_j}{s_j}$$

  - x_{ij} is the value for the i^{th} sample and j^{th} feature
  - m_j is the average of all x_{ij} for feature j
  - s_j is the standard deviation of all x_{ij} over all input samples
- Range and scale of z-scores should be similar (provided distributions of raw feature values are alike)

Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r\right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T Q(\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{m}\left(\sum_{i=1}^{m} (x_i - y_i) q_{ji}\right)(x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x} - \boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i}\left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}}\right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_x$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\,\mathrm{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

Figure 1. Equations of selected distance functions.
($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

Issues with Distance Metrics

- Most distance measures were designed for linear/real-valued attributes
- Two important questions in the context of machine learning:
  - How best to handle nominal attributes
  - What to do when attribute types are mixed

Some Remarks

- *k*-NN works well on many practical problems and is fairly noise tolerant (depending on the value of *k*)
- *k*-NN is subject to the curse of dimensionality (i.e., presence of many irrelevant attributes)
- *k*-NN needs adequate distance measure
- *k*-NN relies on efficient indexing

How is kNN Incremental?

- All training instances are stored
- Model consists of the set of training instances
- Adding a new training instance only affects the computation of neighbors, which is done at execution time (i.e., lazily)

Predicting Continuous Values (regression)

- Replace $\hat{f}(q) = \arg\max_{v \in V} \sum_{i=1}^{k} w_i d(v, f(x_i))$
- by: $\hat{f}(q) = \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$
- Note: unweighted corresponds to w_i=1 for all i

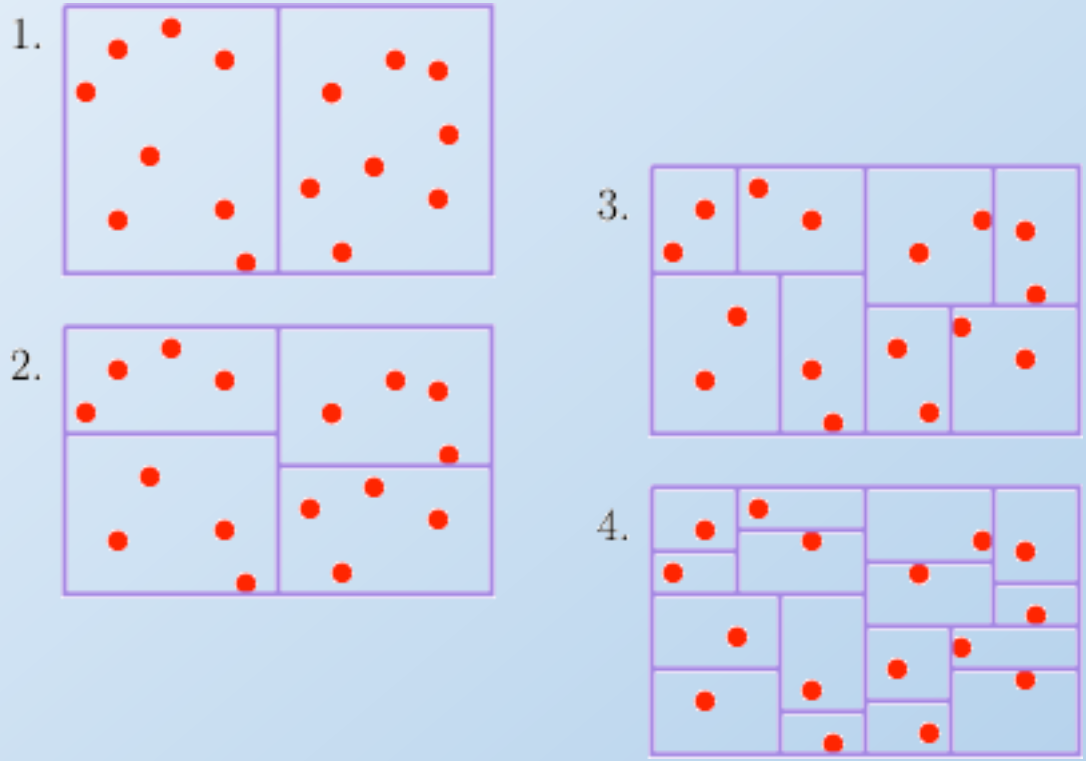
Speeding-up k-nn

- K-d trees
- Nearest-neighbor graph
- Geometric hashing

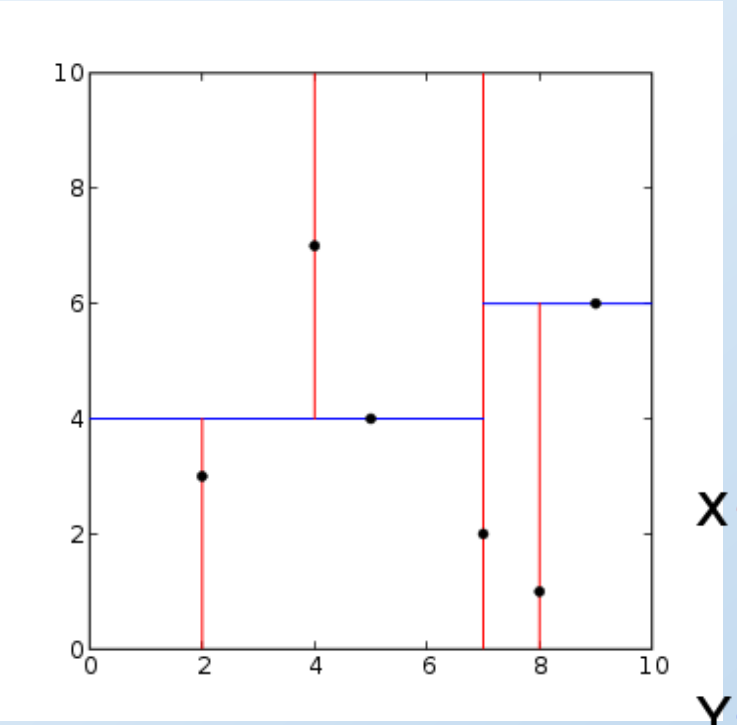
K-d tree

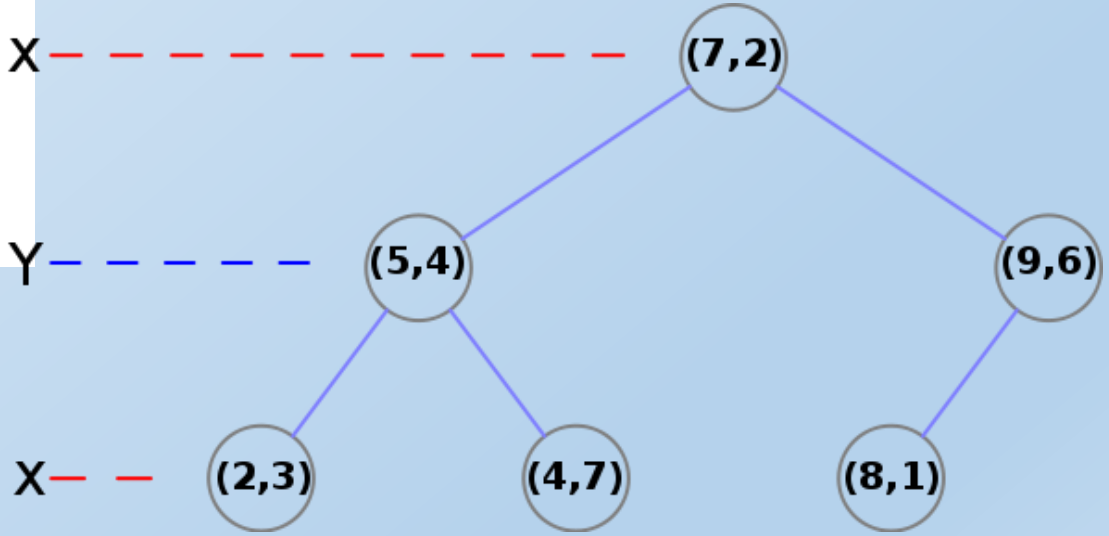
- Used for point location and multiple database queries, k – number of the attributes to perform the search
- Idea:
  - recursively split examples by the median value of an attribute
  - continue until there is one example per region
- Alternate attributes to split

k-d tree

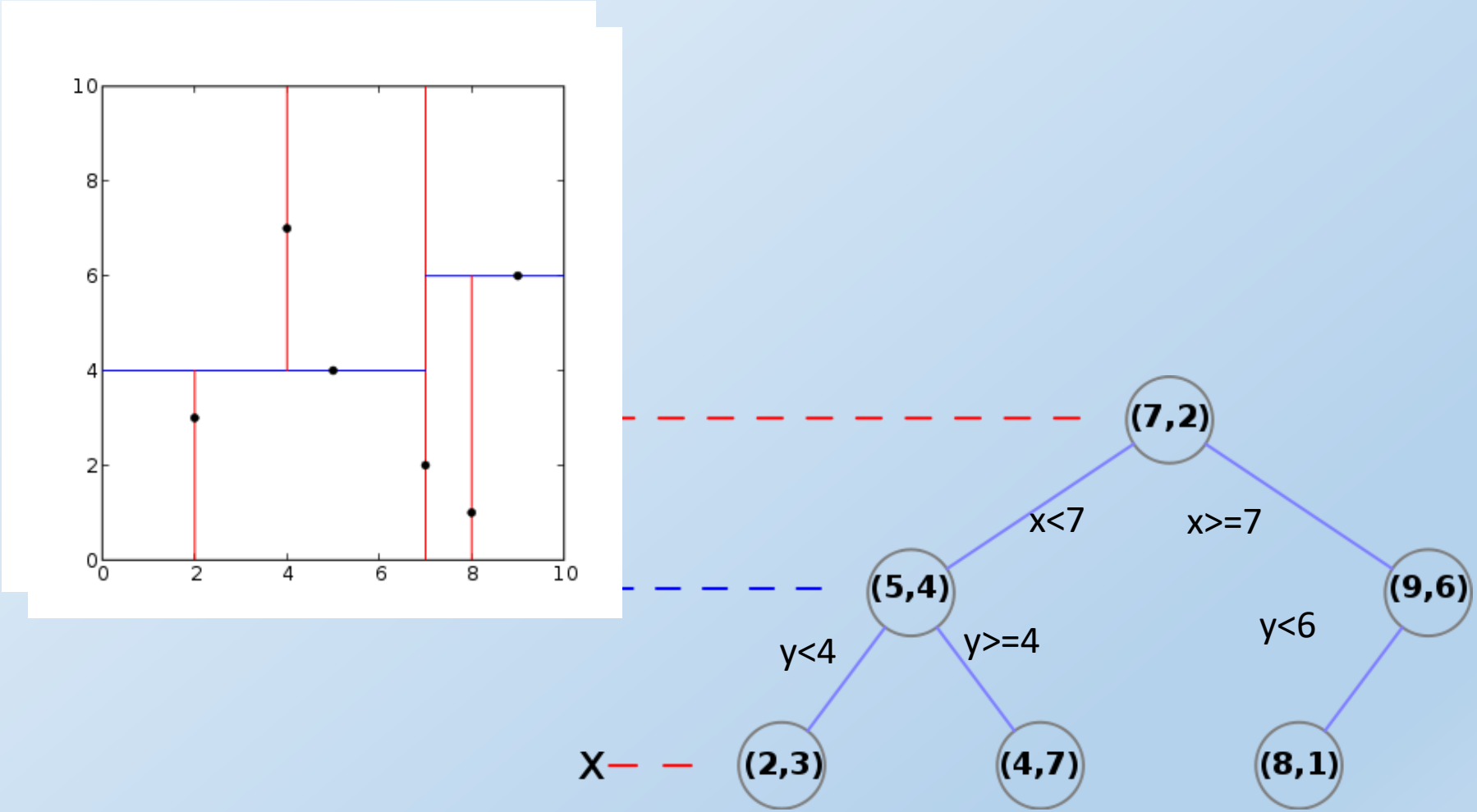


Kd tree example





Kd tree example



Construction

```
function kdtree (list of points pointList, int depth)
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;
    // Sort point list and choose median as pivot element
    select median by axis from pointList;
    // Create node and construct subtree
    node.data := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node
```

Finding the nearest neighbor

Given example x and a k-d tree

Descend to the leaf where x would be, finding the distance from x to every point in the path (there are log n points in the path) and storing the closest one

Traverse back from the leaf to the root

At every node, determine if it is possible to find the nearest neighbor in the subtree not traversed in the descent. If it’s possible to find the nearest neighbor in that subtree, apply search recursively to that subtree

- The savings come from NOT traversing the subtrees that were not visited in the descent

Example

Build a k-d tree with the following training examples:

t1: (1,3)

t2: (2,5)

t3: (3,4)

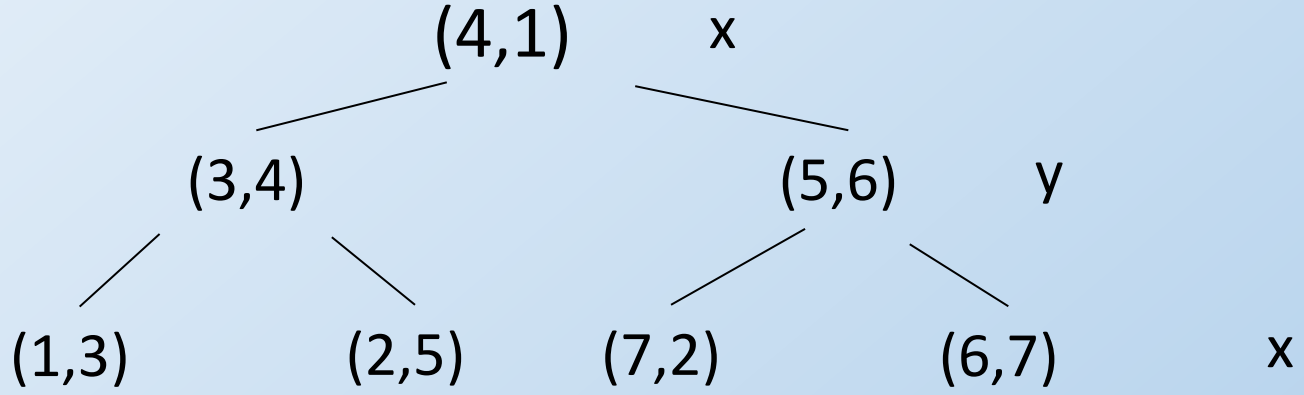
t4: (4,1)

t5: (5,6)

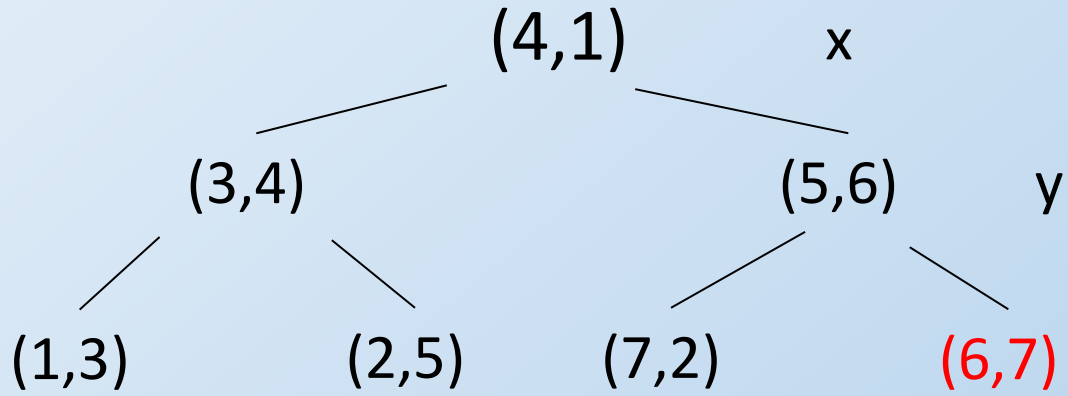
t6: (6,7)

t7: (7,2)

Example

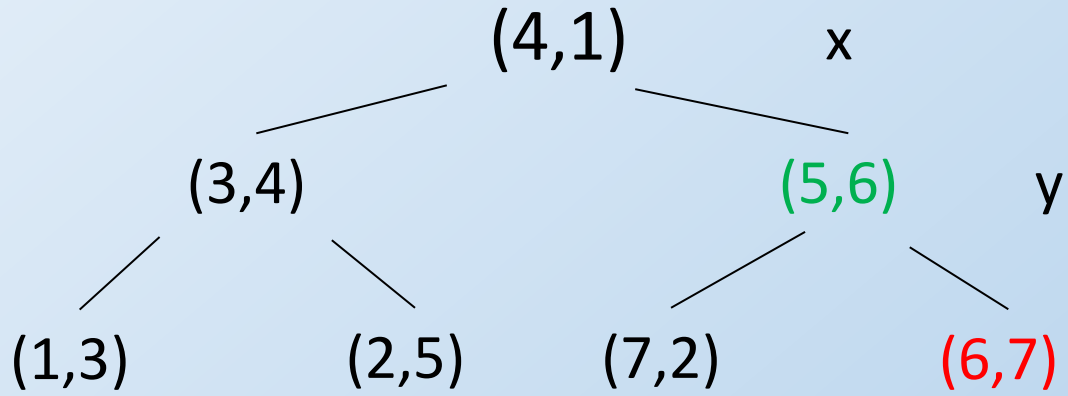


Example



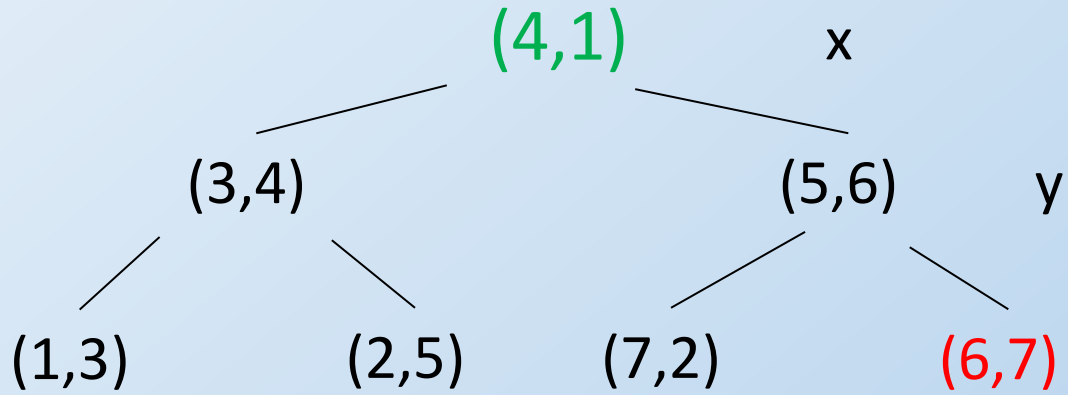
Find the nearest neighbor of (7,10)
After descending, the nearest neighbor found so far is (6,7) (dist = 4)

Example



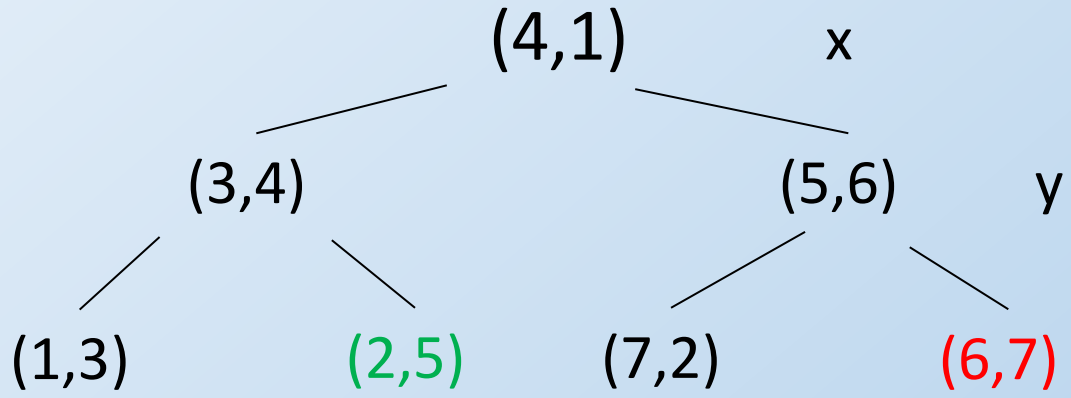
Backtraking
Can we have examples in the left subtree of (5,6) with a distance to (7,10) of less than 4?
No! (since |10-6| =4)

Example



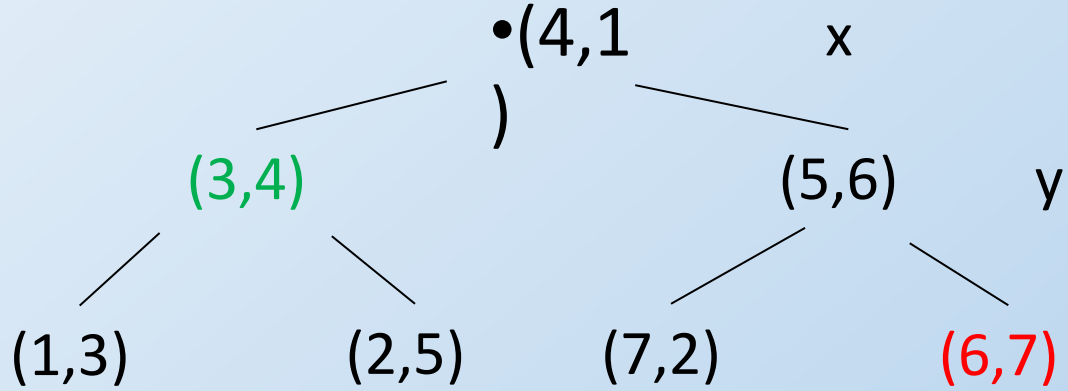
Backtraking
Can we have examples in the left subtree of (4,1) with a distance to (7,10) of less than 4?
Yes! (since |7-4| =3) (point (4,10) could be there)

Example



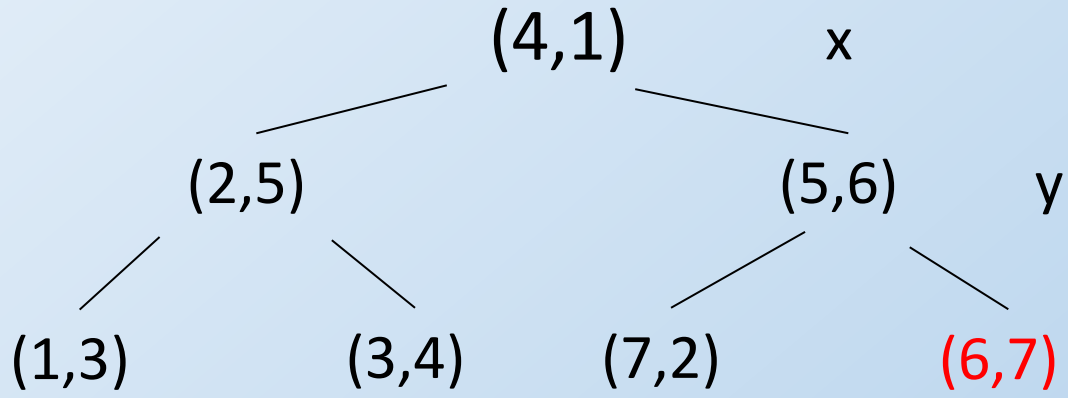
Descent
Nothing found in the descent that is closer

Example



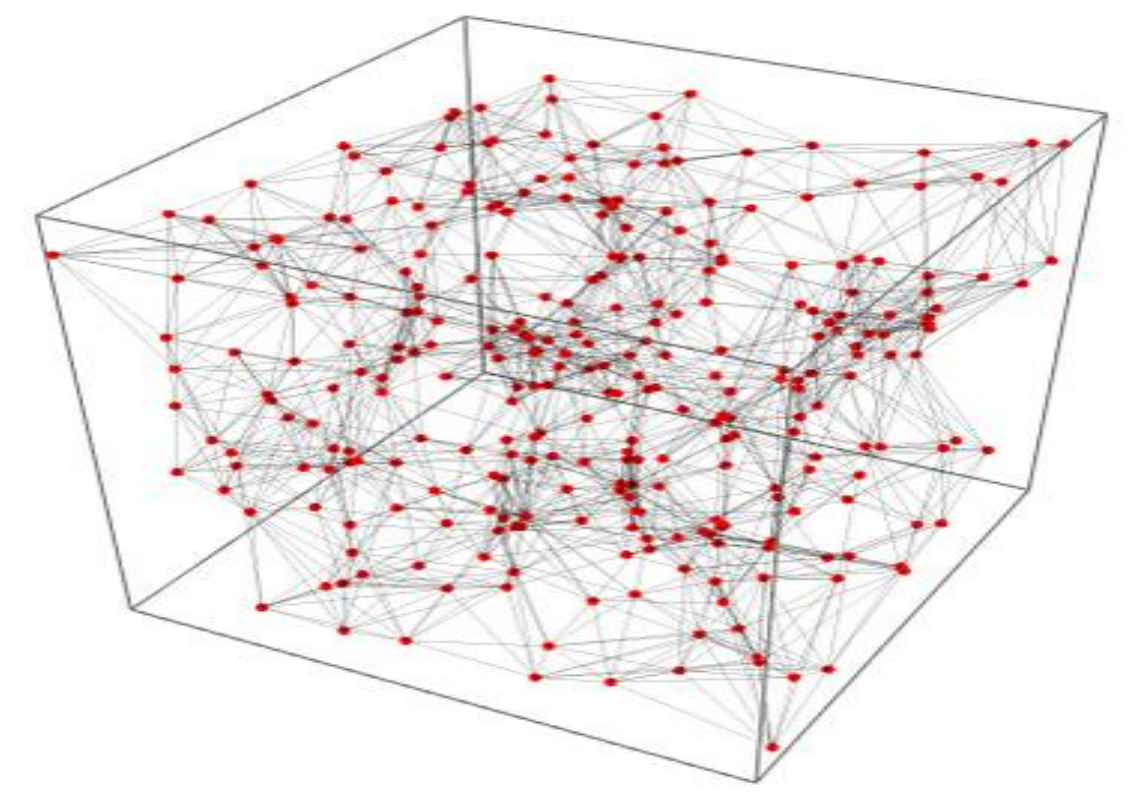
Backtracking
Can we have examples in the left subtree of (3,4) with a distance to (7,10) of less than 4?
No! (since |10-4| = 6)

Example



Done
(6,7) is the nearest neighbor of (7,10)

Nearest neighbor graph



Nearest neighbor graph

Approximation algorithm

Idea:

- Build nearest neighbor graph G of training set
  - Every training example is a vertex
  - Each vertex is connected to its n nearest neighbors

- To find the nearest neighbors of a query point
  - Start at a random vertex in G
  - Search graph, keeping track of the nearest neighbor found until no vertex with smaller distance can be found

Nearest neighbor graph search

Input: x, G, k, r, t

Output: k-nearest neighbors of x

N = {}

for i = 1 to r

  let Y_0 be a random vertex in G

  for j = 1 to t

    N = N + G[Y_{j-1}] – add to N the nearest neighbors of Y_{j-1}

    Y_j = vertex in G[Y_{j-1}] that is closest to x

nn = k points in N that are closest to x

return nn

Nearest neighbor graph search

Input: x, G, k, r, t

Output: k-nearest neighbors of x

N = {}

for i = 1 to r

  let Y_0 be a random vertex in G

  for j = 1 to t

    N = N + G[Y_{j-1}] – add to N the nearest neighbors of Y_{j-1}

    Y_j = vertex in G[Y_{j-1}] that is closest to x

nn = k points in N that are closest to x

return nn

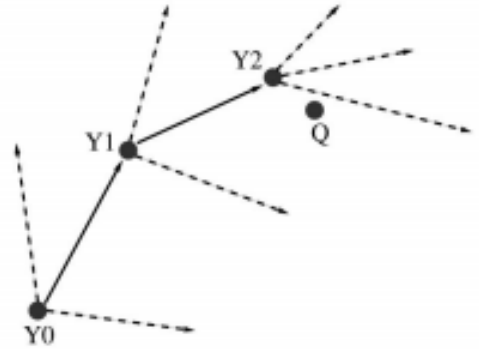


Figure 2: The GNNS Algorithm on a simple nearest neighbor graph.

Nearest neighbor graph

- Can significantly speed up nearest neighbor search when training set is large and it's feasible to spend time preprocessing (building the graph is time consuming)
- Search is probabilistic, no guarantee of finding actual nearest neighbors
- Tradeoff between quality and execution time
  - More random starts, more neighbors per vertex -> better results, higher running time
- In some languages such as Python, where numeric computation can be implemented efficiently, time savings are more limited