Instance-based learning: k-Nearest Neighbours

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

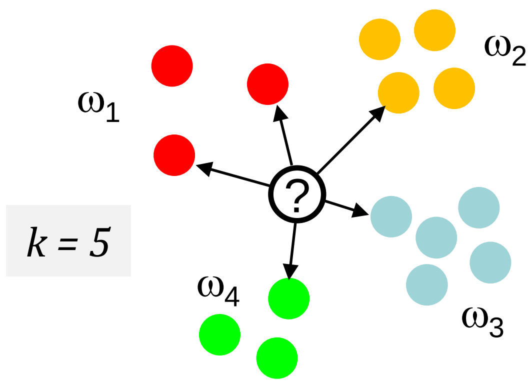
Alberto ORTIZ RODRÍGUEZ

Contents

- k-Nearest Neighbours classifier
- Supplementary material: Nearest-Neighbour Search & *k*-d trees
- Supplementary material: Condensed Nearest Neighbours
- Example of use

k-Nearest Neighbours classifier

- Supervised classification scheme based on the so-called **k-nearest neighbours rule**:
  - Given an unknown feature vector x and a distance d(x,y), e.g. Euclidean d(x,y) = $||x - y||$
    1. identify the k nearest neighbours (according to d) out of the N training samples
    2. out of these k samples, identify the number of patterns n_i that belong to every class ω_i, i = 1, 2, …, M
    3. assign x to the class ω_j such tat $n_j = \max\{n_1, n_2, \ldots, n_k\}$



$x \rightarrow \omega_1$

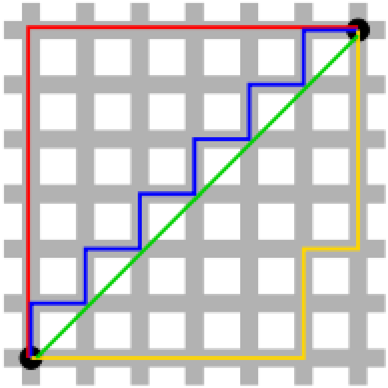
- kNN is considered a **lazy learning** algorithm
  - There is no training, or data abstraction/modeling, step
  - Defers data processing until it receives a request to classify an unlabelled sample
  - A priori, the full training dataset is needed
- There is a single parameter *k*

- **Examples of distance functions**:

  – weighted **L$_p$ metric** (or **Minkowski** measure)

$$d_p(a,b) = \left(\sum_{i=1}^{L} w_i |a_i - b_i|^p \right)^{\frac{1}{p}}, w_i \geq 0$$

$$d_2(a,b) = \sqrt{\sum_{i=1}^{L} (a_i - b_i)^2} \quad \text{(metric } L_2 \text{ or Euclidean distance)}$$



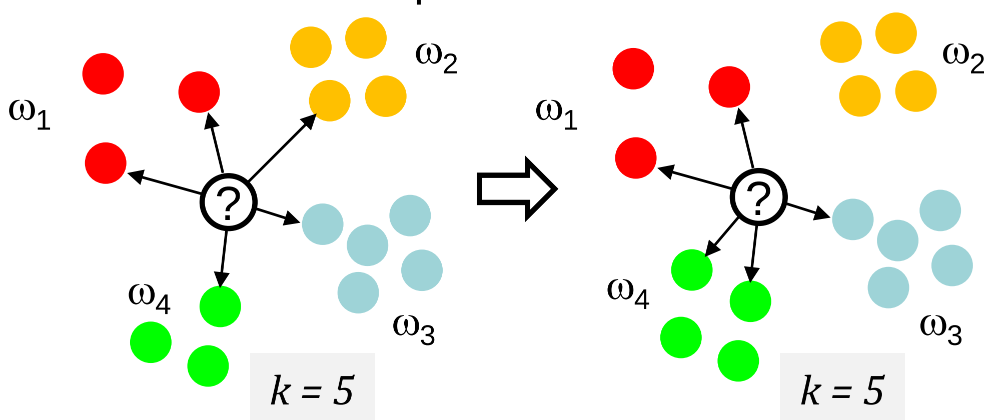
$$d_1(a,b) = \sum_{i=1}^{L} |a_i - b_i| \quad \text{(metric } L_1 \text{ or Manhattan distance or City Block distance)}$$

$$d_\infty(a,b) = \max_{1 \leq i \leq L} |a_i - b_i| \quad \text{(metric } L_\infty \text{ or Chebyshev distance)}$$

$$\left(= \lim_{p \to +\infty} d_p(a,b) \quad \text{if } w_i = 1\right)$$

k-Nearest Neighbours classifier

- To avoid ties, *k* is typically chosen **odd** for two-class problems, and, in general, not to be a multiple of the number of classes *M*



Given $N_k(x)$ the k-nearest neighbours of sample x and y_i the class labels:

$$n_1(x) = \sum_{i \in N_k(x)} I(y_i = 1) = 2$$

$$n_2(x) = \sum_{i \in N_k(x)} I(y_i = 2) = 0$$

$$n_3(x) = \sum_{i \in N_k(x)} I(y_i = 3) = 1$$

$$n_4(x) = \sum_{i \in N_k(x)} I(y_i = 4) = 2$$

$I(p)$ is the indicator function:

$$I(p) = \begin{cases} 1 & \text{if } p \text{ is true} \\ 0 & \text{otherwise} \end{cases}$$

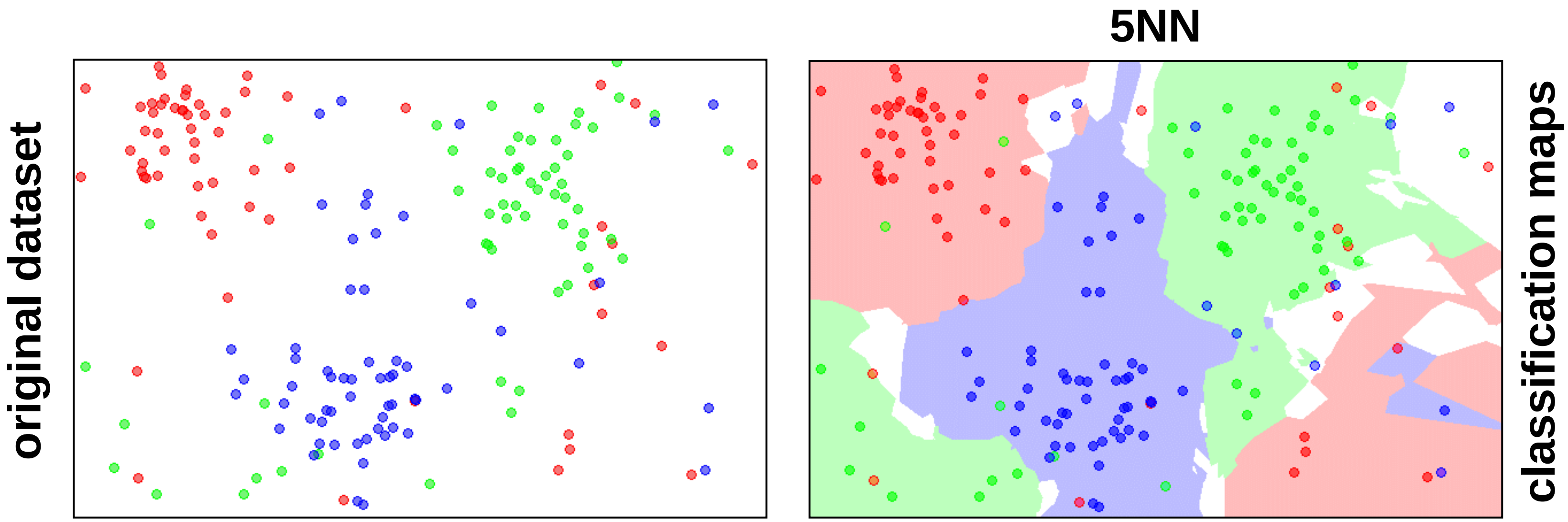
- Even with this, **ties** may arise: $n_1(x) = n_4(x)$
  - Ties may be broken **arbitrarily**
  - The unlabeled sample x may be assigned to the **class of the nearest neighbor**
    - consider all classes — $x \rightarrow \omega_3$
    - consider only the classes with the tying values — $x \rightarrow \omega_4$
  - Instead of every sample voting 1, make use of a **weighted vote** inversely proportional to the distance:

$$n_j(x) = \sum_{i \in N_k(x)} \frac{1}{1 + d(x_i, x)} I(y_i = j)$$

$x \rightarrow \omega_4$

$$x \rightarrow \omega_q\,, \quad q = \arg\max_j \; n_j$$

- An **example**: 3 classes, 60 samples/class, white = unclassified, i.e. kNN voting tied



k-Nearest Neighbours classifier

- Nearest neighbour is **competitive**:

Yann LeCunn – MNIST Digit Recognition
- Handwritten digits
- 28x28 pixel images: d = 784
- 60,000 training samples
- 10,000 test samples

(http://yann.lecun.com/exdb/mnist/)

	Test Error Rate (%)
Linear classifier (1-layer NN)	12.0
3-nearest-neighbors, Euclidean	5.0
3-nearest-neighbors, Euclidean, deskewed	2.4
1-NN, Tangent Distance, 16x16	1.1
1000 RBF + linear classifier (10 neurons)	3.6
SVM deg 4 polynomial	1.1
2-layer NN, 300 hidden units	4.7
2-layer NN, 300 HU, [deskewing]	1.6
LeNet-5, [distortions]	0.8
Boosted LeNet-4, [distortions]	0.7

- Other **theoretical results**:
  - e.g. error is less than twice the optimal classification error (Bayes error)

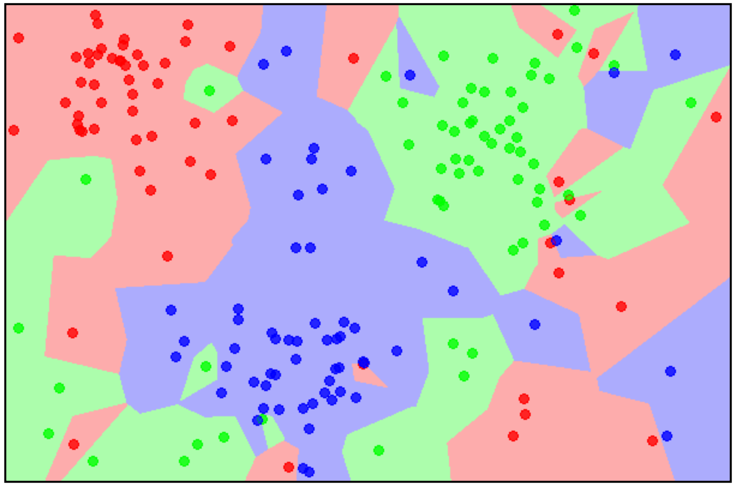
k-Nearest Neighbours classifier

- The simplest version of the algorithm is the **nearest neighbour classifier** (1NN)

Assign x to the class ω_j of its nearest neighbour

- Systematic application of the rule throughout the feature space gives rise to the **Voronoi tessellation/diagram**
  – shows the points of the feature space which are closest to every sample and therefore "inherit" its label

Voronoi tesselation



1NN

ties are less likelier !!

k-Nearest Neighbours classifier

- In the kNN algorithm, the **greatest effort** is placed on the **classification** not on the training:
  - Given a pattern **x** to classify, one has to calculate the distance $\mathbf{d(x,x_i)}$ from **x** to any of the **N** patterns $\mathbf{x_i}$ in the full dataset and keep the **k** nearest patterns according to **d** (**brute force** approach)
  - Great computational cost for N large: e.g. k = 1 & N = $10^6 \Rightarrow 10^6$ comparisons

- kNN and 1NN are similar in terms of efficiency
  - retrieving the k nearest neighbors is not much more expensive than retrieving a single nearest neighbor
  - k nearest neighbors can be maintained in a sorted queue

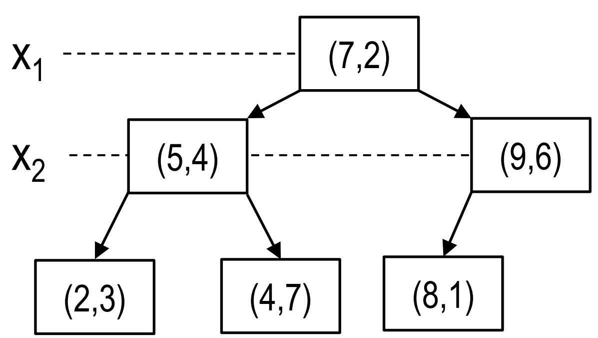
- A number of ways of reducing the search cost
  - Handle the dataset with the goal of reducing the search complexity, e.g. use a ***k*-d tree** or a **ball-tree**, use **approximate nearest neighbour search** (ANN), etc.
  - Reduce the size of the dataset without significantly altering the classification accuracy

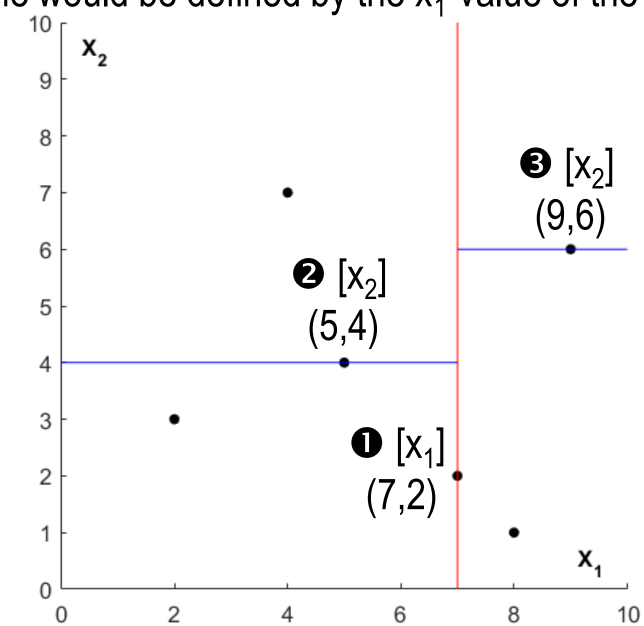
Contents

- k-Nearest Neighbours classifier
- Supplementary material: Nearest-Neighbour Search & *k*-d trees
- Supplementary material: Condensed Nearest Neighbours
- Example of use

- A ***k*-d tree** is a binary tree to store a set of k-dimensional points in an “ordered” way
- Every non-leaf node generates a **splitting hyperplane** along one axis, which divides the space into two parts, known as **half-spaces**
  - Points to the left of the hyperplane (for that axis) fall within the left subtree and points to the right of the hyperplane (for that axis) fall within the right subtree
  - The splitting axis is chosen such that **every node is associated with one of the k dimensions**, with the hyperplane perpendicular to that dimension's axis
    - e.g. if for a particular split the x_1 axis is chosen, all points in the subtree with a smaller x_1 value than the node’s x_1 value will appear in the left subtree and all points with larger x_1 value will be in the right subtree. In such a case, the hyperplane would be defined by the x_1-value of the node, and its normal would be the unit x_1-axis.

$X = \{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)\}$





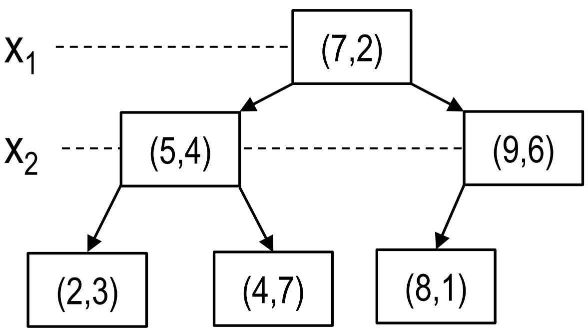
- There are many possible ways to choose axis-aligned splitting planes, and so there are many **different ways to construct *k*-d trees**
- The **canonical method** of *k*-d tree construction has the following constraints:
  - The **splitting axis changes sequentially** from level to level: first level – x_1, second level – x_2, etc. (start again with x_1 when the k-th level is reached)
  - Split is performed at the **median** of the subtree values for the chosen axis

X = {(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)}

(1) choose axis x_1 and take the median of
{2,5,9,4,8,7} → {2,4,5,**7**,8,9} → 7 → (**7**,2)
left subtree: {(2,3), (5,4), (4,7)}
right subtree: {(9,6), (8,1)}

(2) [L] choose axis x_2 and take the median of
{3,**4**,7} → 4 → (5,**4**)
left subtree: (2,3)
right subtree: (4,7)

(3) [R] choose axis x_2 and take the median of
{6,1} → {1,**6**} → 6 → (9,**6**)
left subtree: (8,1)



  - This method leads to a **balanced** *k*-d tree: all leaf nodes approx. equally closer to the root

k-d trees

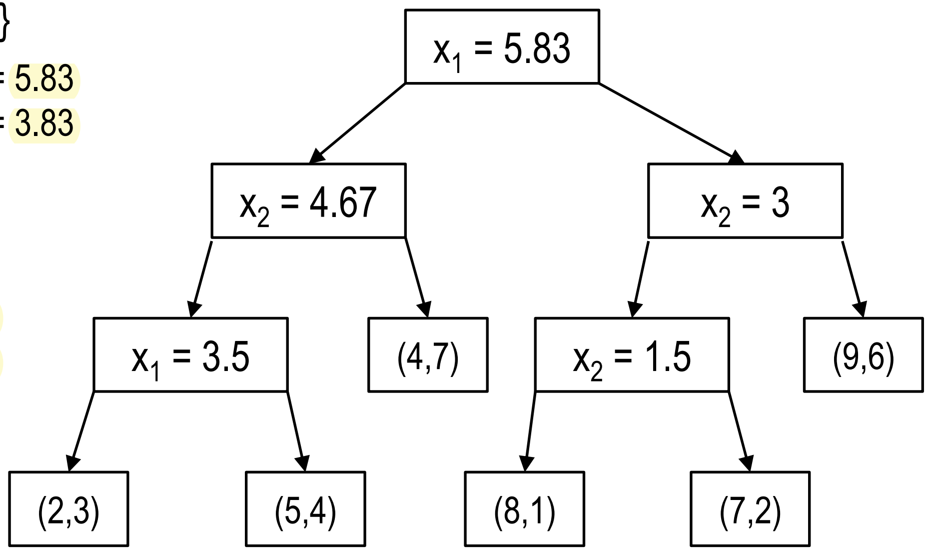
- Another popular method:
  - Choose the splitting axis according to the **spread** (var / range) of the data along each axis
  - Choose the split value as the **average** of values for the chosen axis and subtree

$X = \{(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)\}$

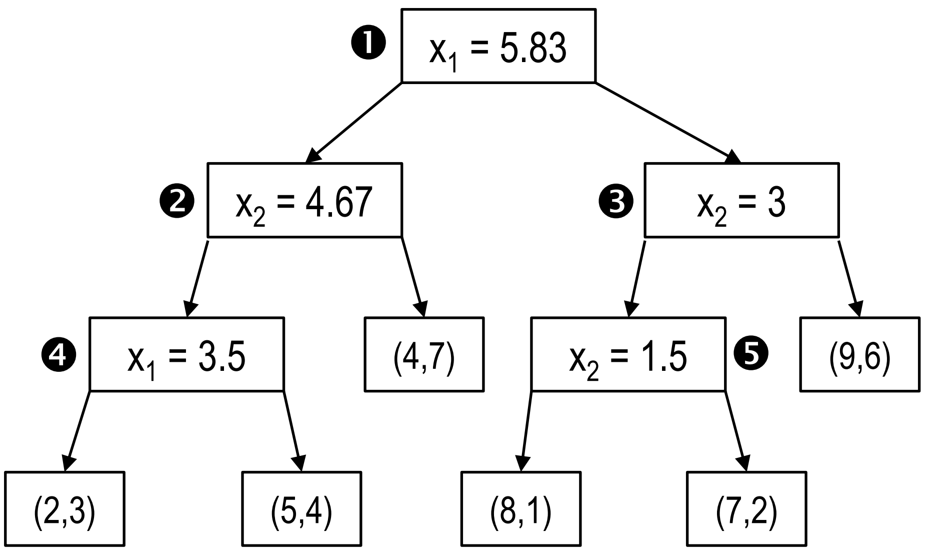
(1) $s_1 = \text{range}\{2,5,9,4,8,7\} = 9 - 2 = 7$, $\mu_1 = 5.83$
$s_2 = \text{range}\{3,4,6,7,1,2\} = 7 - 1 = 6$, $\mu_2 = 3.83$
choose axis x_1 and split at $x_1 = 5.83$
left subtree: {(2,3), (5,4), (4,7)}
right subtree: {(9,6), (8,1),(7,2)}

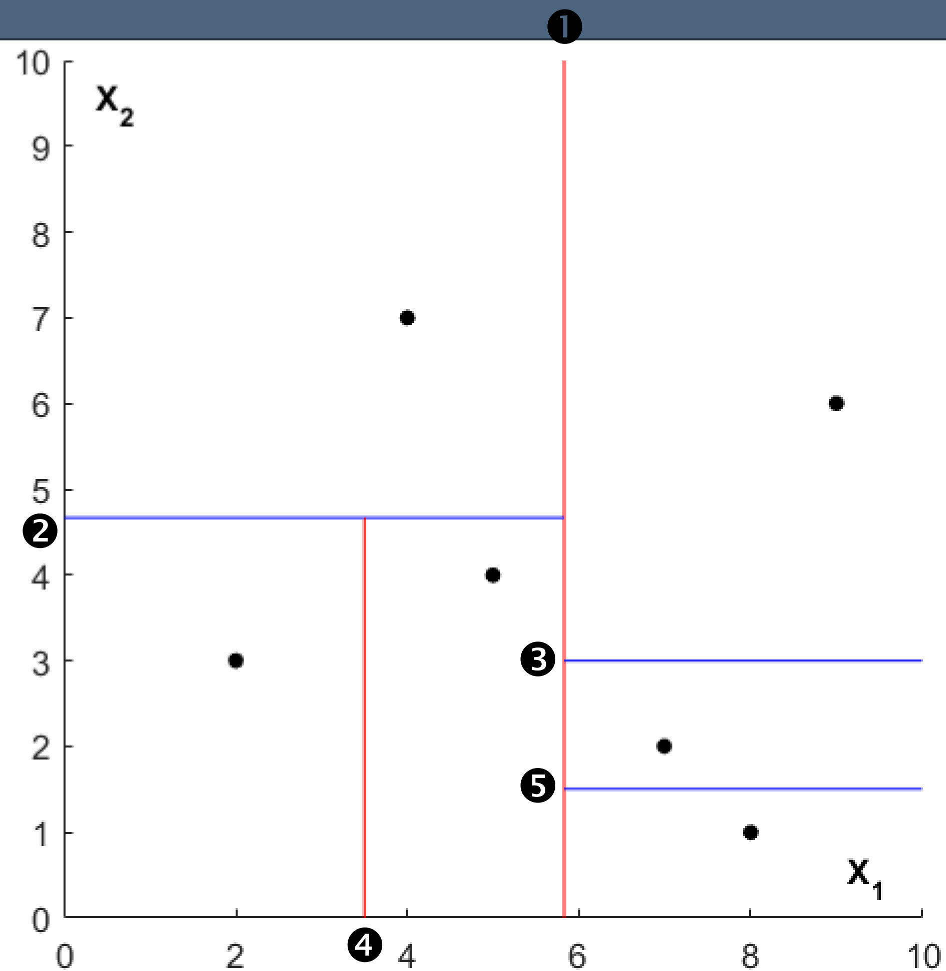
(2L) $s_1 = \text{range}\{2,5,4\} = 5 - 2 = 3$, $\mu_1 = 3.67$
$s_2 = \text{range}\{3,4,7\} = 7 - 3 = 4$, $\mu_2 = 4.67$
choose axis x_2 and split at $x_2 = 4.67$
left subtree: {(2,3),(5,4)}
right subtree: (4,7)

(3R) $s_1 = \text{range}\{9,8,7\} = 9 - 7 = 2$, $\mu_1 = 8$
$s_2 = \text{range}\{6,1,2\} = 6 - 1 = 5$, $\mu_2 = 3$
choose axis x_2 and split at $x_2 = 3$
left subtree: {(8,1), (7,2)}
right subtree: (9,6)



  - This method **avoids the sort** operation to find the median
  - Besides, **samples are stored only at the leafs**, not throughout the tree





- Growth of the tree can be stopped at any level: there can be **more than 1 sample per leaf**
- Growth can also continue until the **number of samples per leaf** is below a threshold

- **Nearest-neighbor search** (2nd k-d tree construction approach):

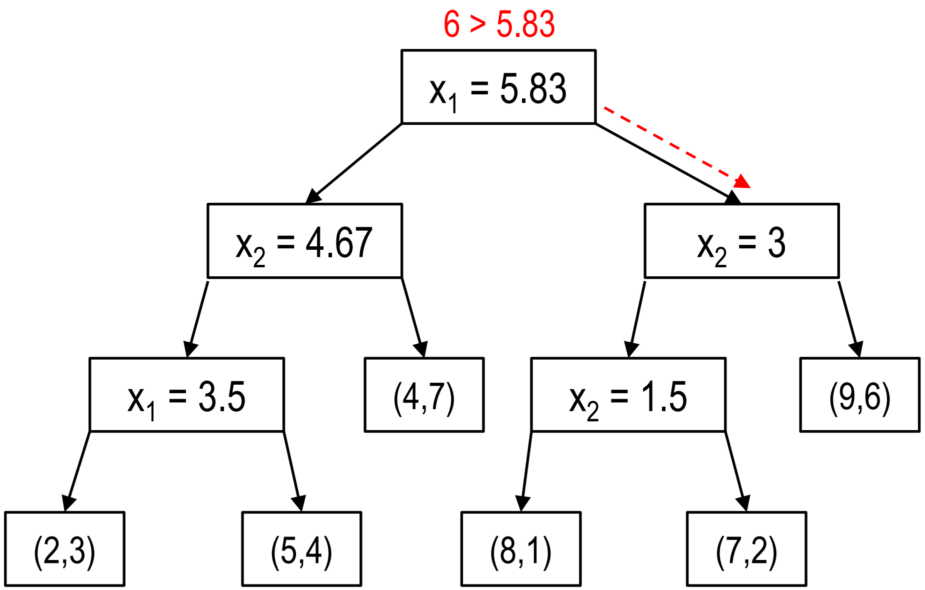
```
node {
int axis;      // splitting axis
real value;    // splitting value
node left;     // left subtree
node right;    // right subtree
kdpoint point; // if leaf node
}

NNS(q:in kdpoint, n:in node, b:inout point, r:inout real)
{
    if n.left = empty & n.right = empty then // it is a leaf node (only 1 sample/leaf)
        r_ = dist(q, n.point);
        if r_ < r then r = r_; b = n.point;  // update nearest
    else                                     // find first subtree to look into
        if q(n.axis) <= n.value then         // use splitting axis, visit subtrees in proper order
            NNS(q, n.left, b, r);            // look within left subtree
            if q(n.axis) + r >  n.value then NNS(q, n.right, b, r); // NN can be in right subtree
        else
            NNS(q, n.right, b, r);           // look within right subtree
            if q(n.axis) - r <= n.value then NNS(q, n.left, b, r);  // NN can be in left subtree
}

initial call: NNS(q, root, b, inf);
```

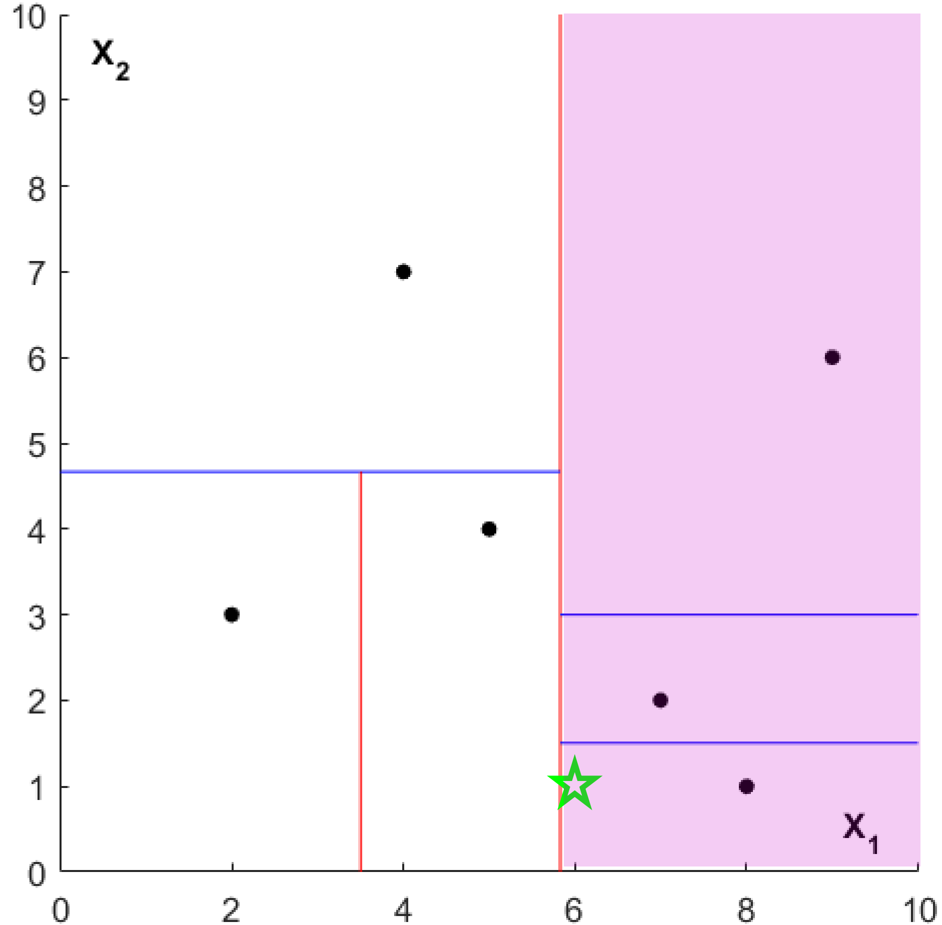
search pruning

- <u>remark</u>: algorithm slightly different when there are samples at intermediate nodes

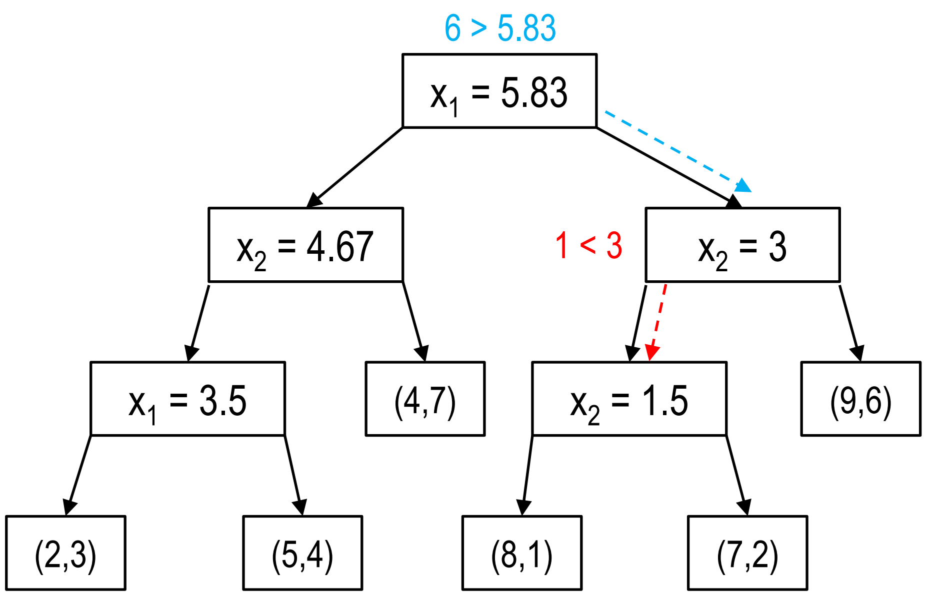
k-d trees

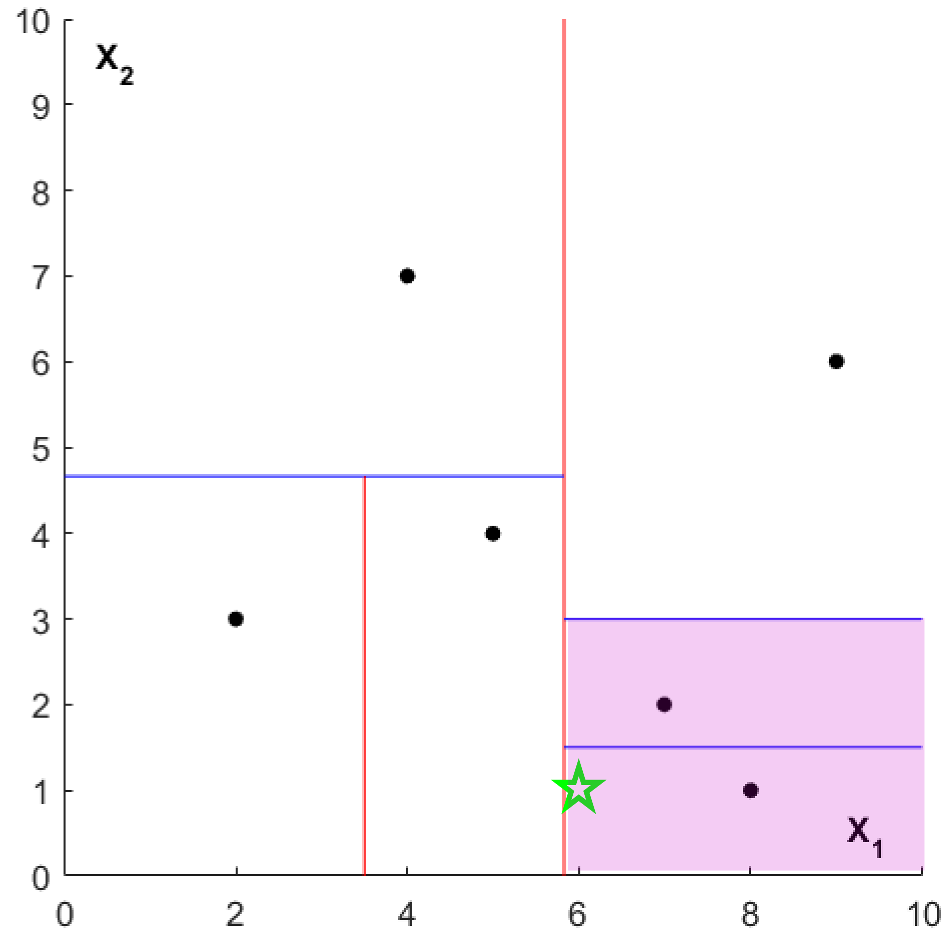
- q = (6,1)
- $r_0 = \infty$



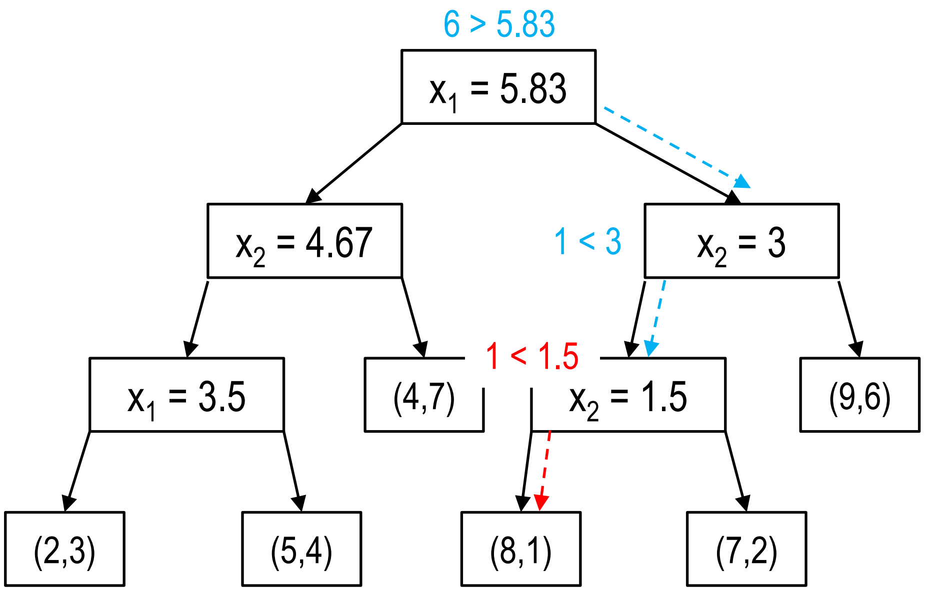


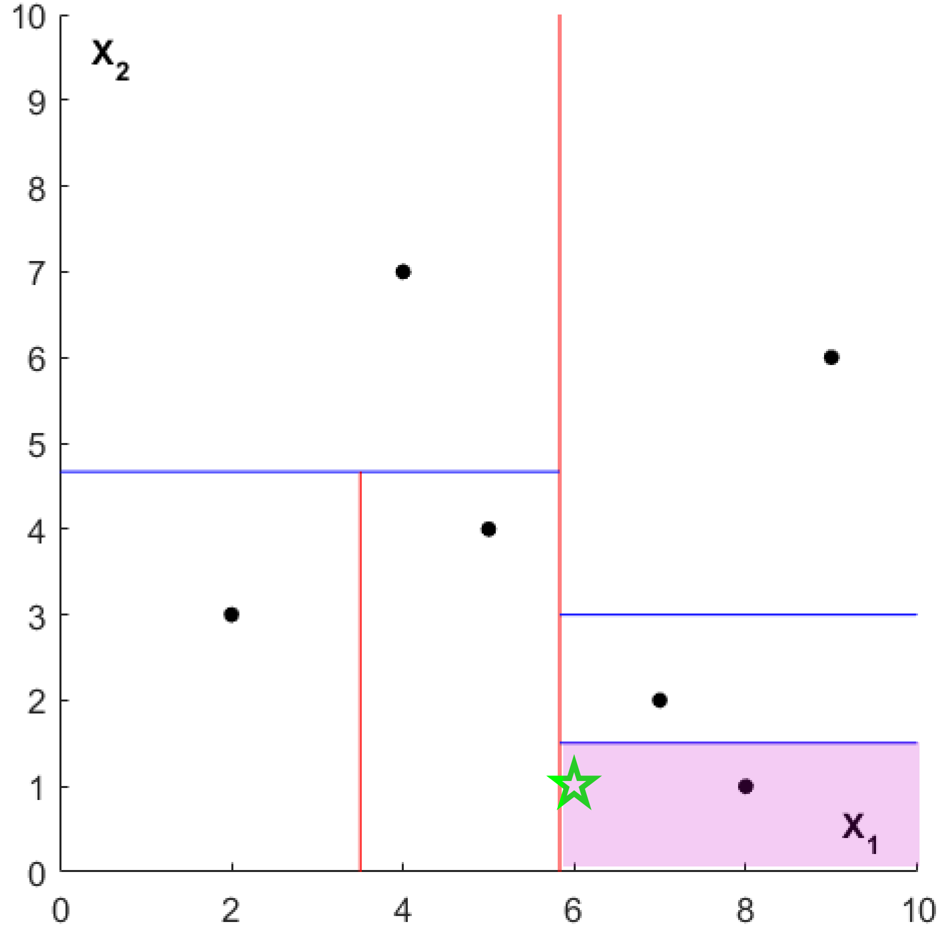
- $q = (6,1)$
- $r_0 = \infty$


6 > 5.83
x1 = 5.83
x2 = 4.67
1 < 3
x2 = 3
x1 = 3.5
(4,7)
x2 = 1.5
(9,6)
(2,3)
(5,4)
(8,1)
(7,2)


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

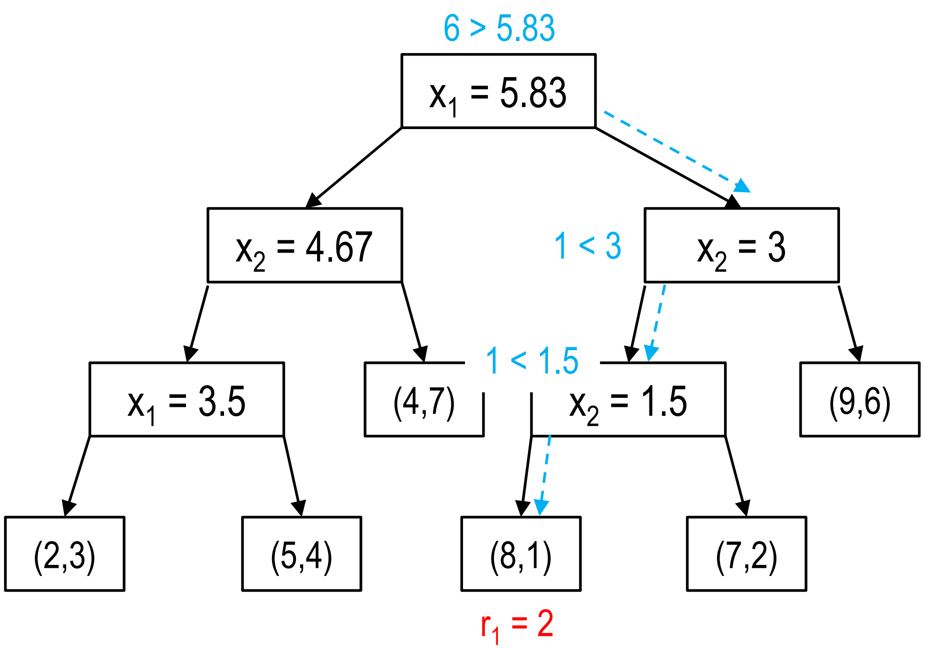
- $q = (6,1)$
- $r_0 = \infty$

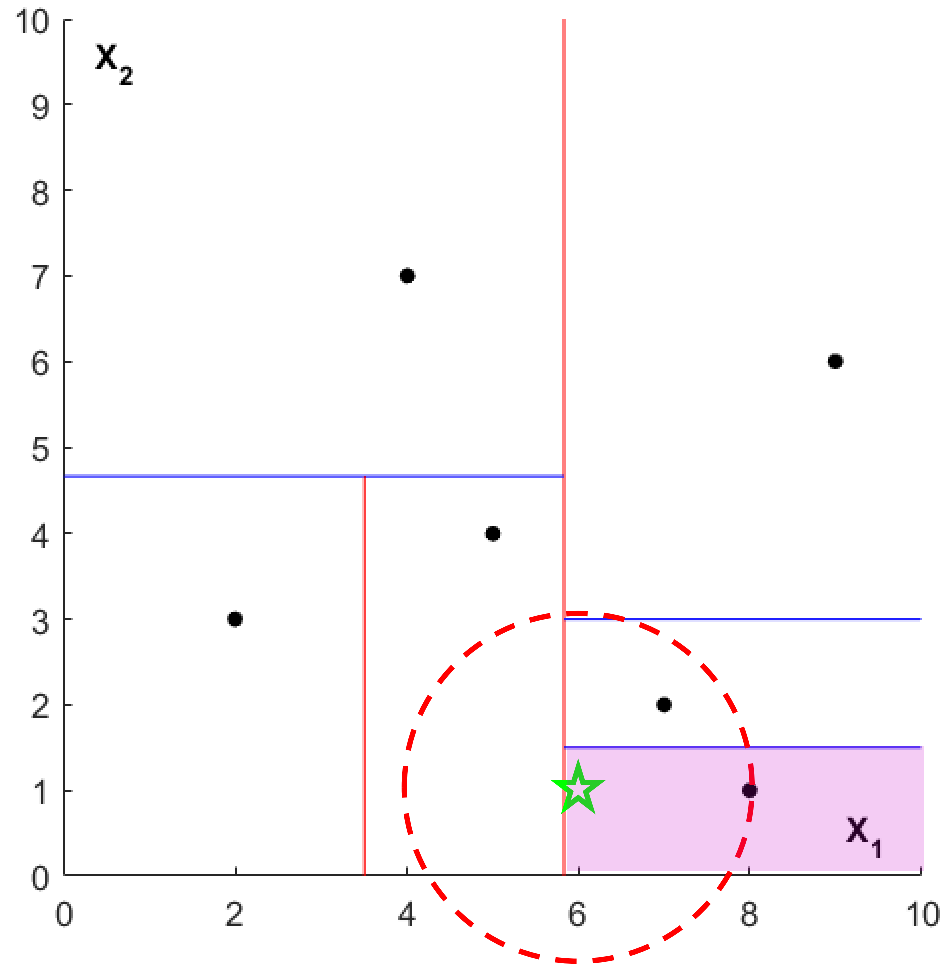

6 > 5.83
x1 = 5.83
x2 = 4.67
1 < 3
x2 = 3
x1 = 3.5
(4,7)
1 < 1.5
x2 = 1.5
(9,6)
(2,3)
(5,4)
(8,1)
(7,2)


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

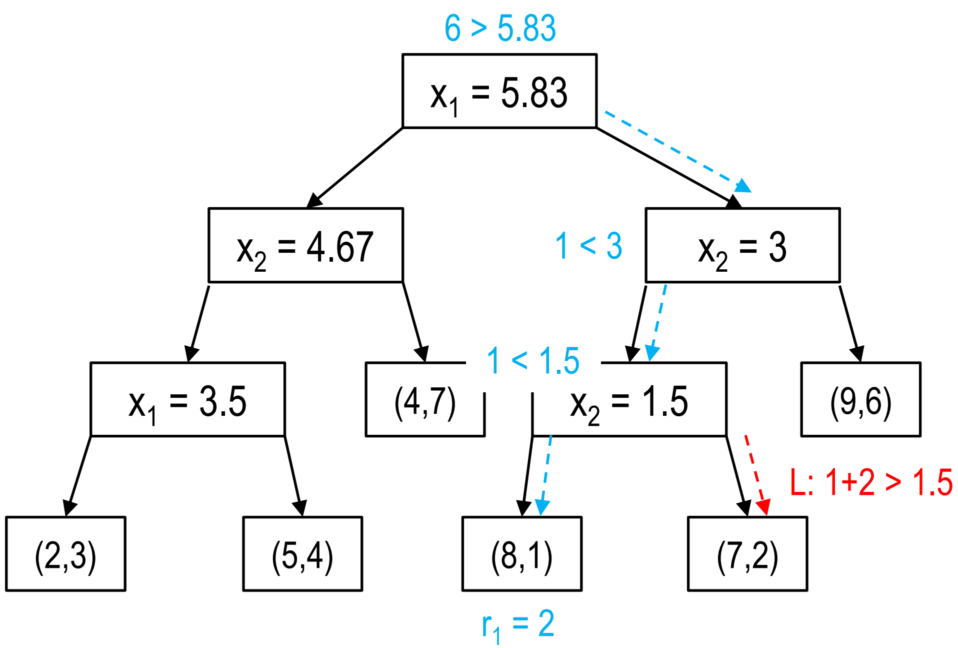
k-d trees

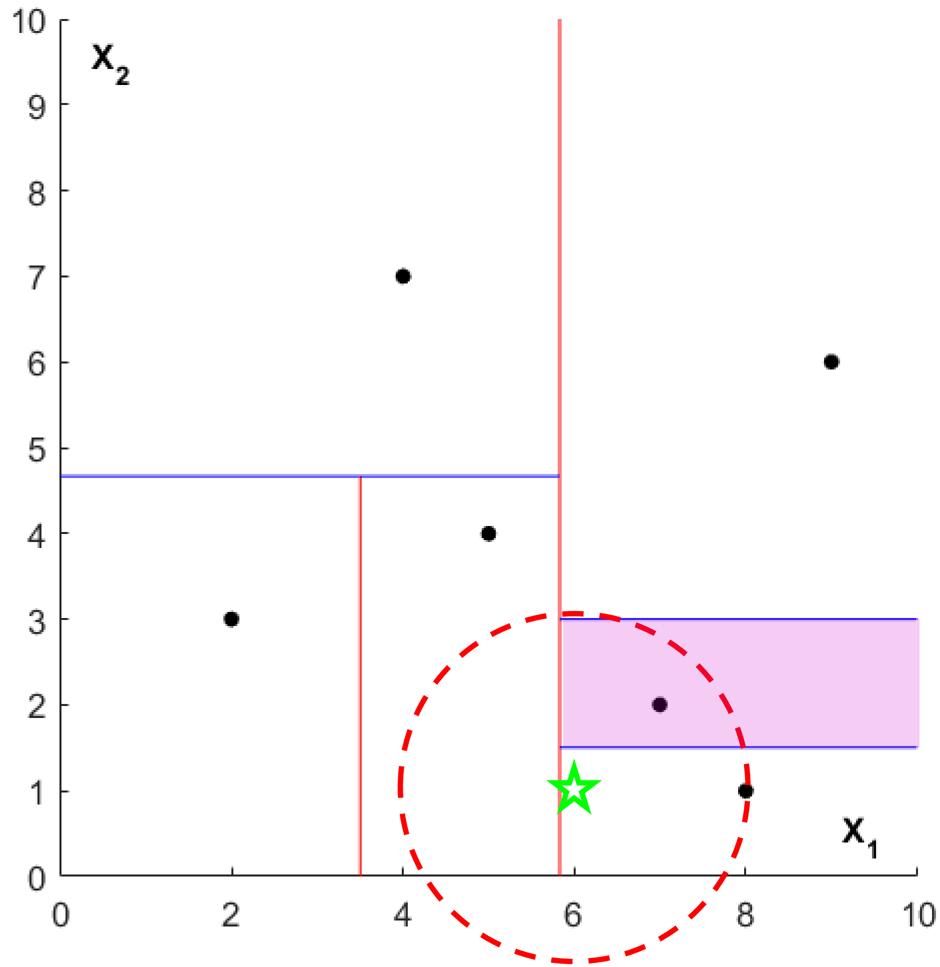
- $q = (6,1)$
- $r_0 = \infty$



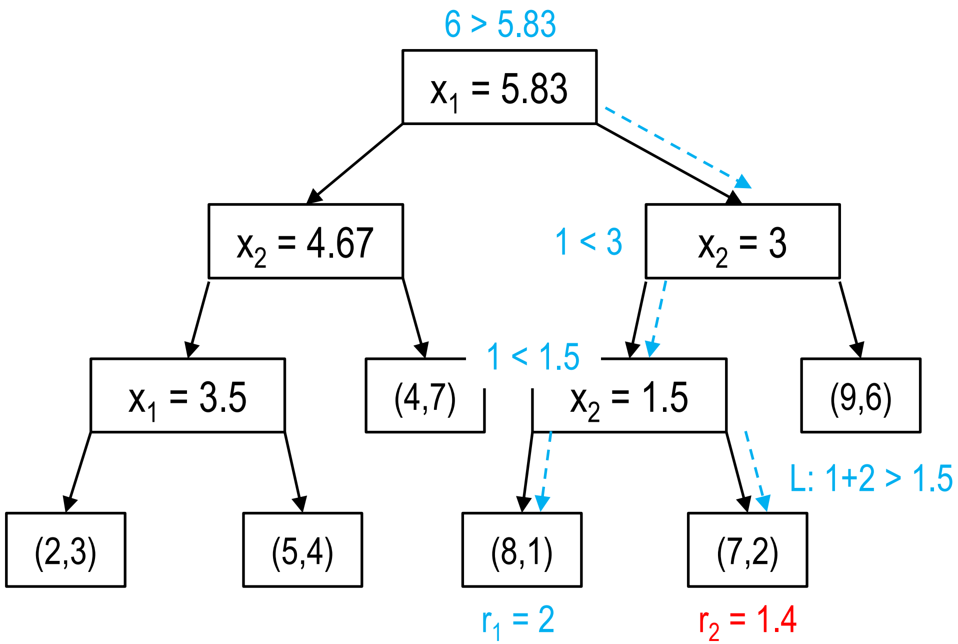


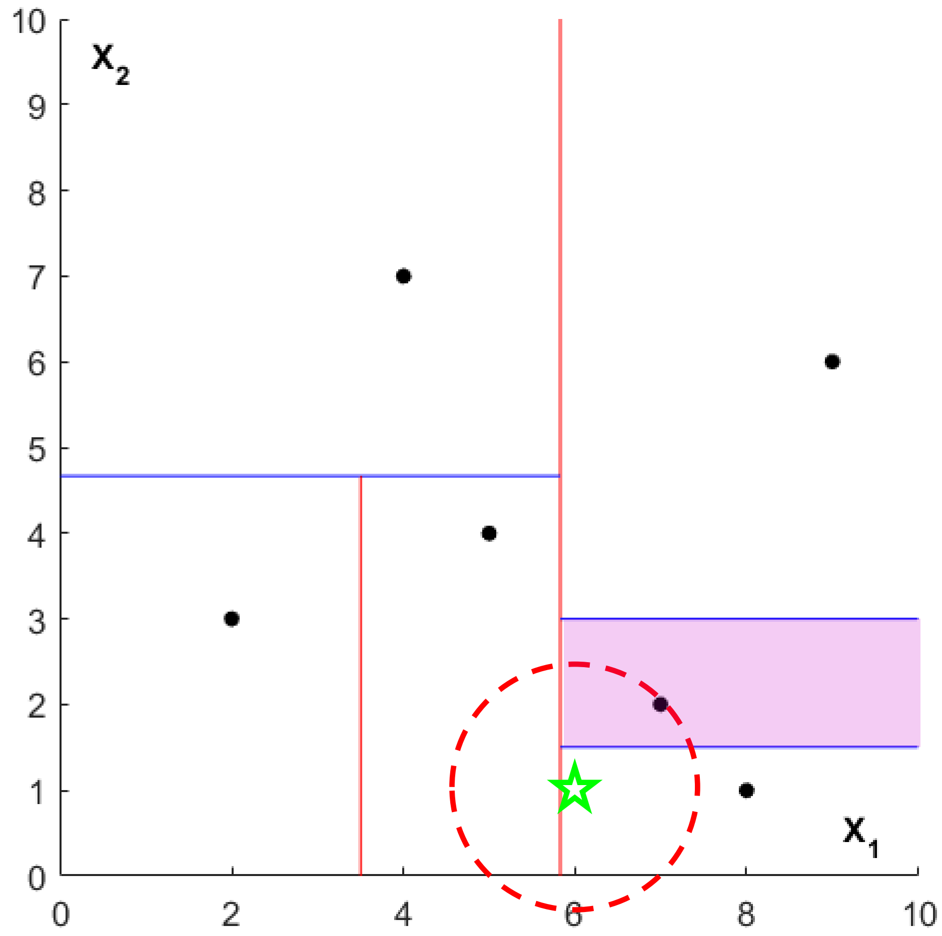
- $q = (6,1)$
- $r_0 = \infty$


6 > 5.83
x1 = 5.83
x2 = 4.67
1 < 3
x2 = 3
x1 = 3.5
(4,7)
1 < 1.5
x2 = 1.5
(9,6)
L: 1+2 > 1.5
(2,3)
(5,4)
(8,1)
(7,2)
r1 = 2

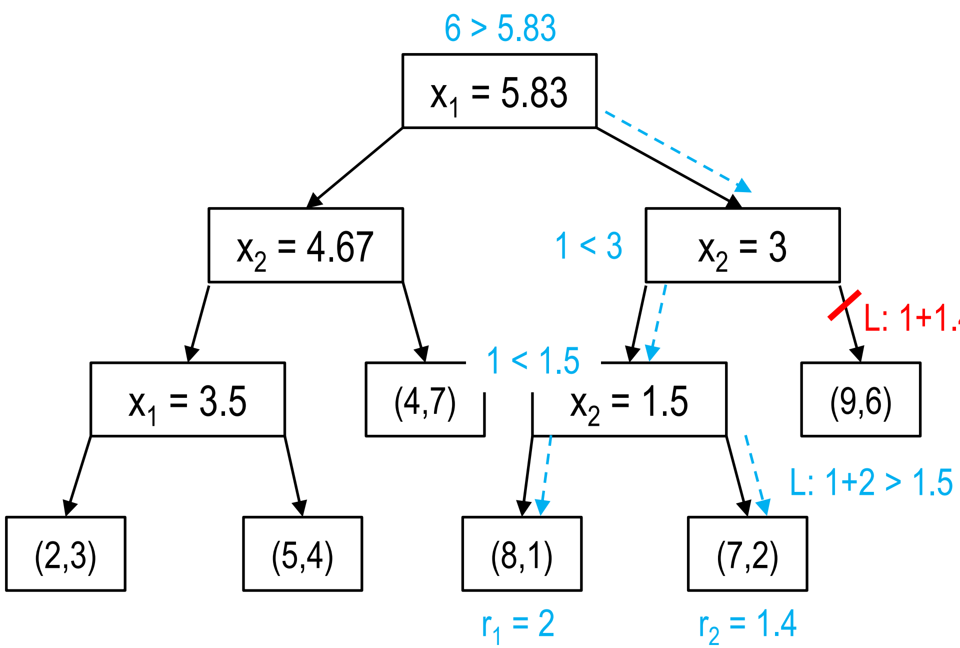

x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

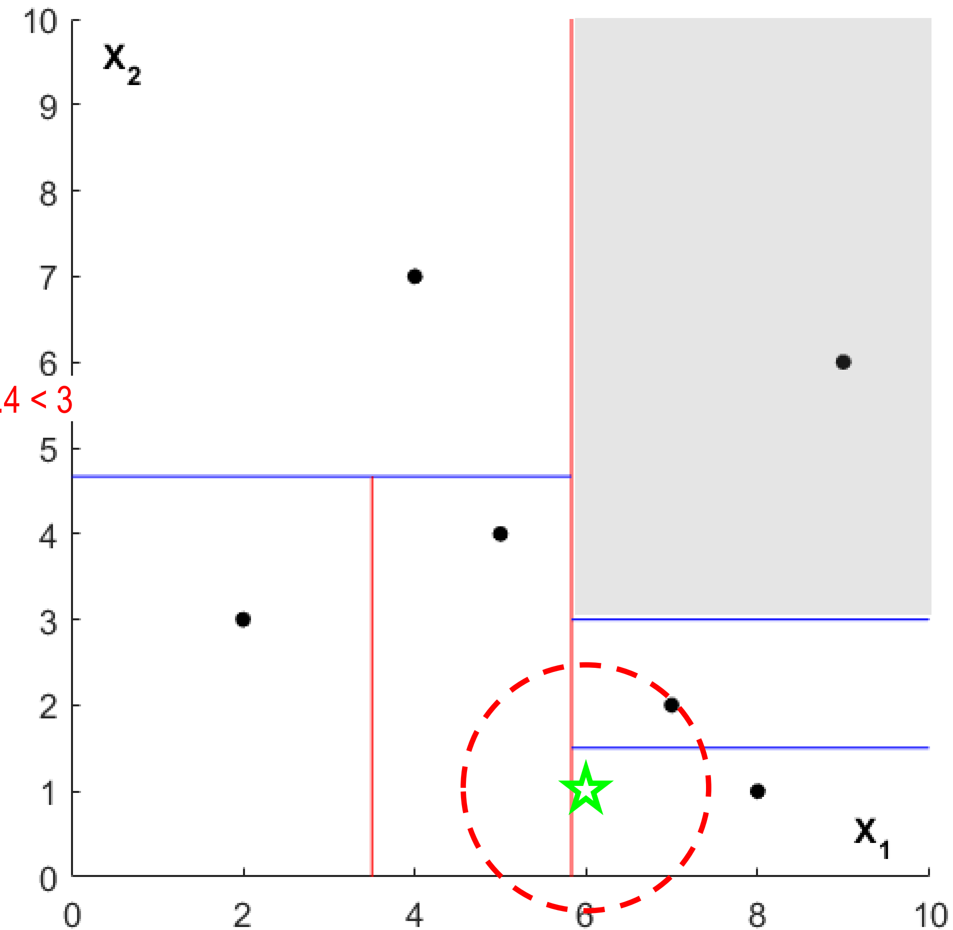
- $q = (6,1)$
- $r_0 = \infty$


6 > 5.83
x1 = 5.83
x2 = 4.67
1 < 3
x2 = 3
x1 = 3.5
(4,7)
1 < 1.5
x2 = 1.5
(9,6)
L: 1+2 > 1.5
(2,3)
(5,4)
(8,1)
(7,2)
r1 = 2
r2 = 1.4


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

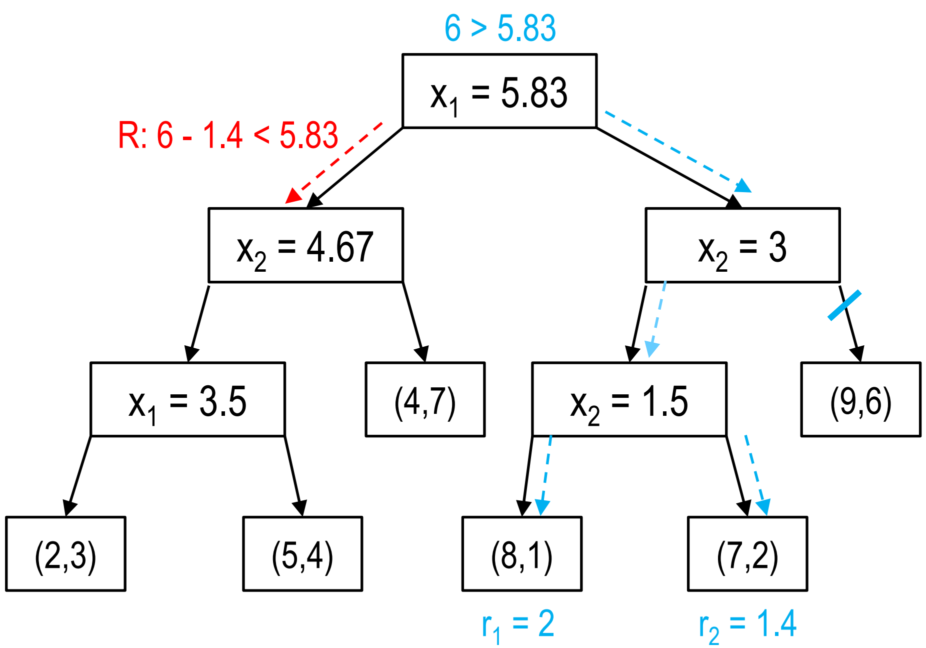
- $q = (6,1)$
- $r_0 = \infty$

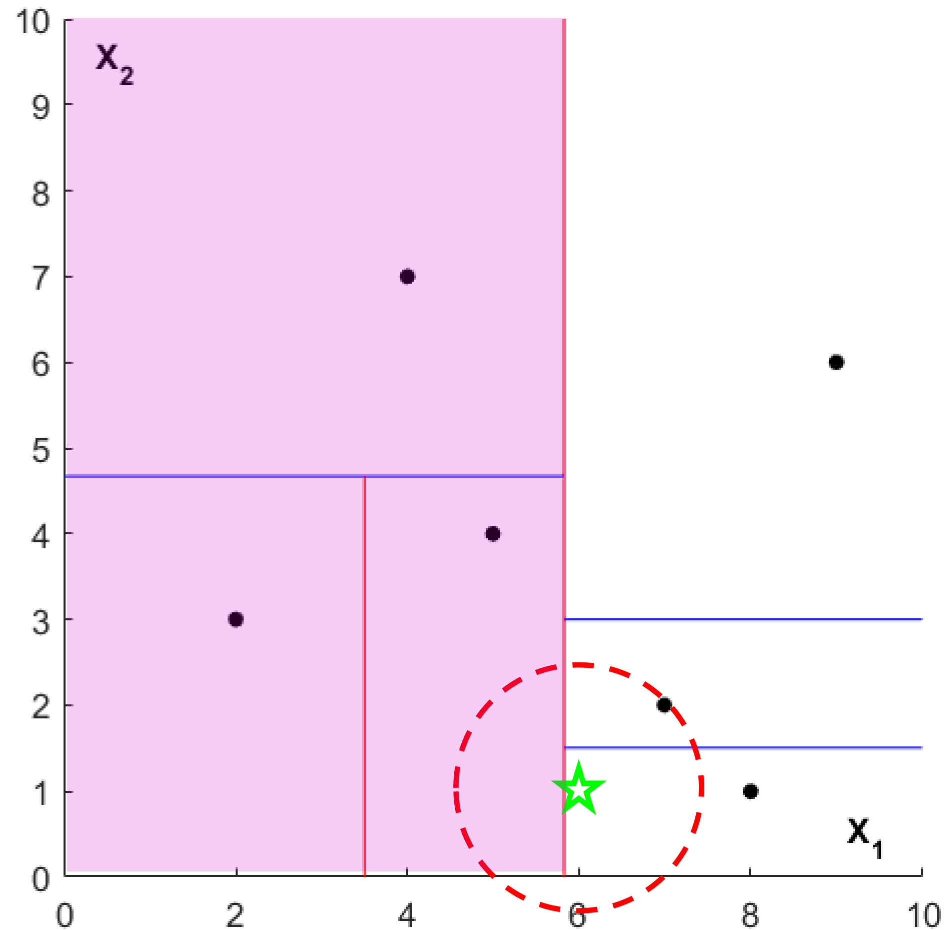

6 > 5.83
x1 = 5.83
x2 = 4.67
1 < 3
x2 = 3
L: 1+1.4 < 3
x1 = 3.5
(4,7)
1 < 1.5
x2 = 1.5
(9,6)
L: 1+2 > 1.5
(2,3)
(5,4)
(8,1)
(7,2)
r1 = 2
r2 = 1.4


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

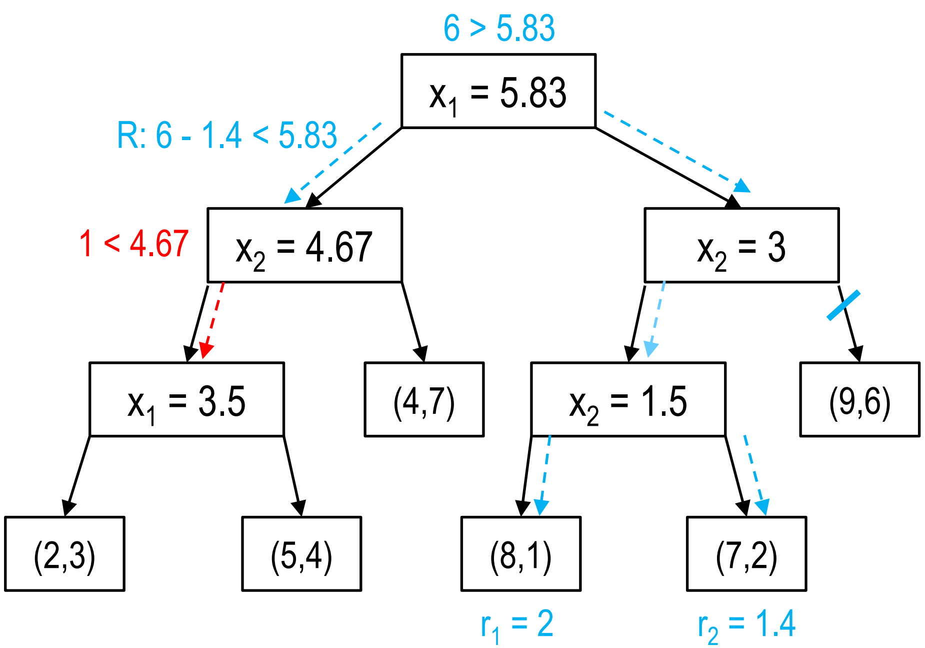
k-d trees

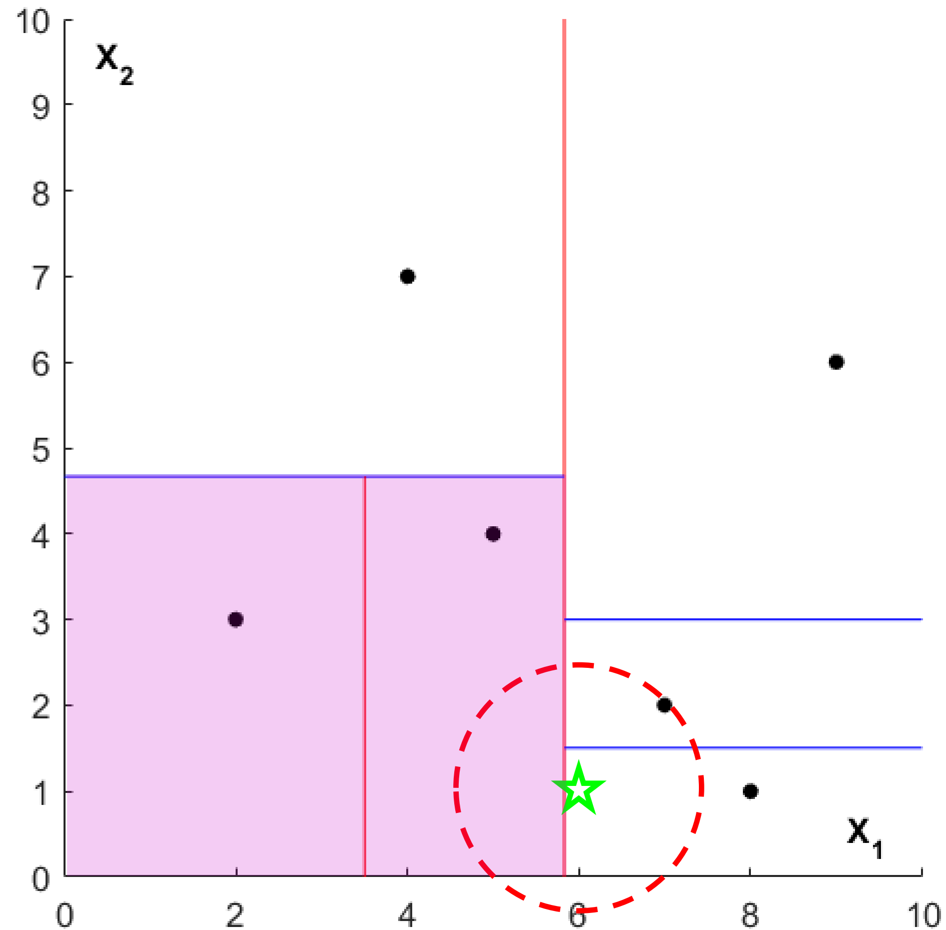
- $q = (6,1)$
- $r_0 = \infty$



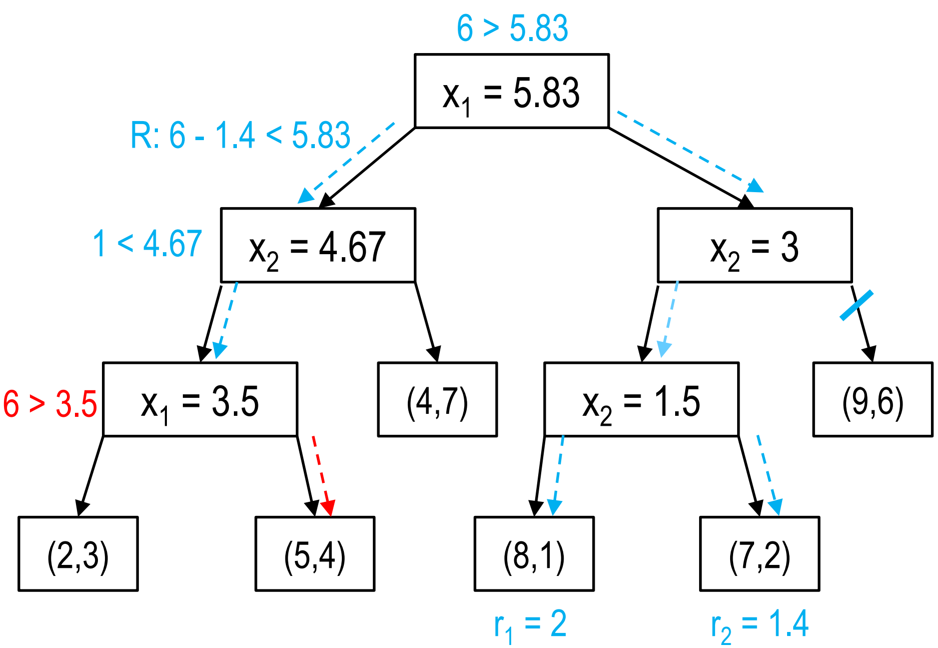


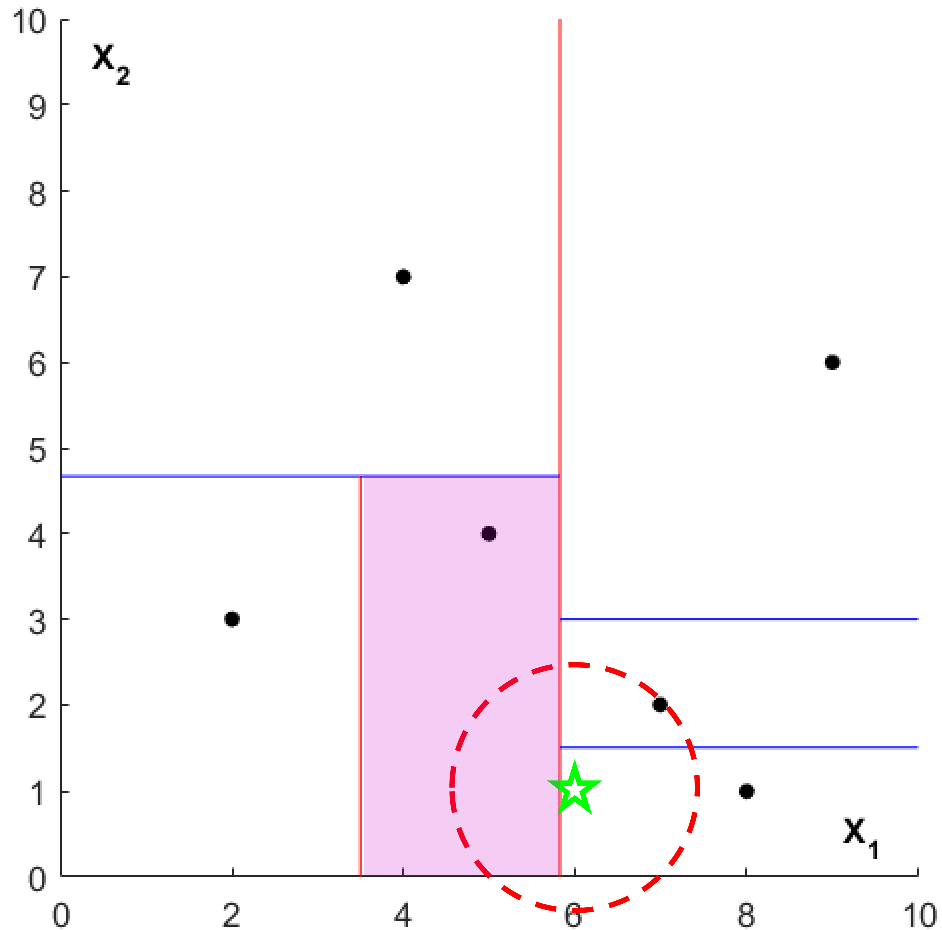
- $q = (6,1)$
- $r_0 = \infty$


6 > 5.83
x1 = 5.83
R: 6 - 1.4 < 5.83
1 < 4.67
x2 = 4.67
x2 = 3
x1 = 3.5
(4,7)
x2 = 1.5
(9,6)
(2,3)
(5,4)
(8,1)
(7,2)
r1 = 2
r2 = 1.4


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

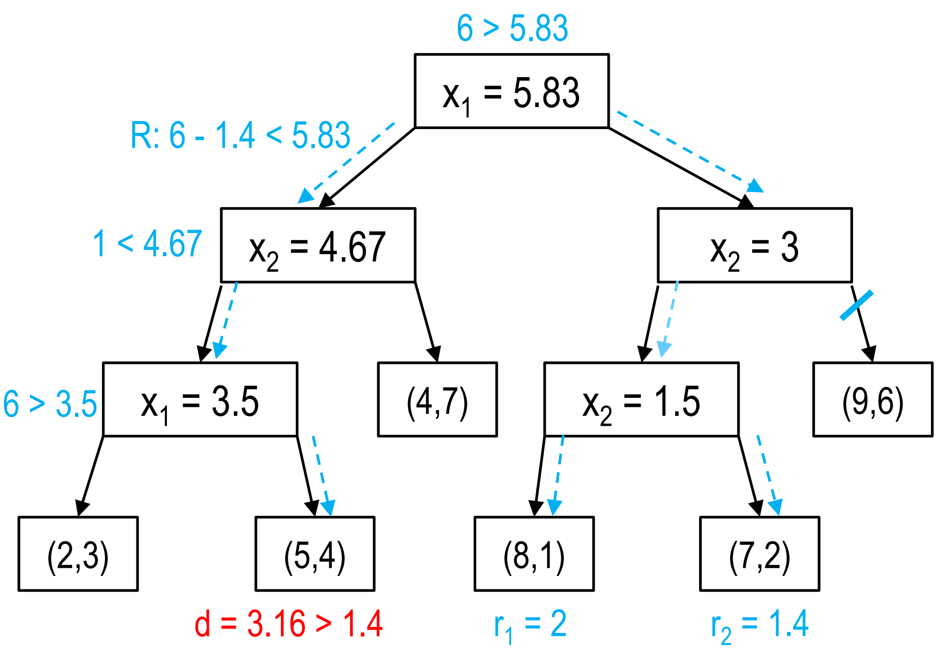
- $q = (6,1)$
- $r_0 = \infty$

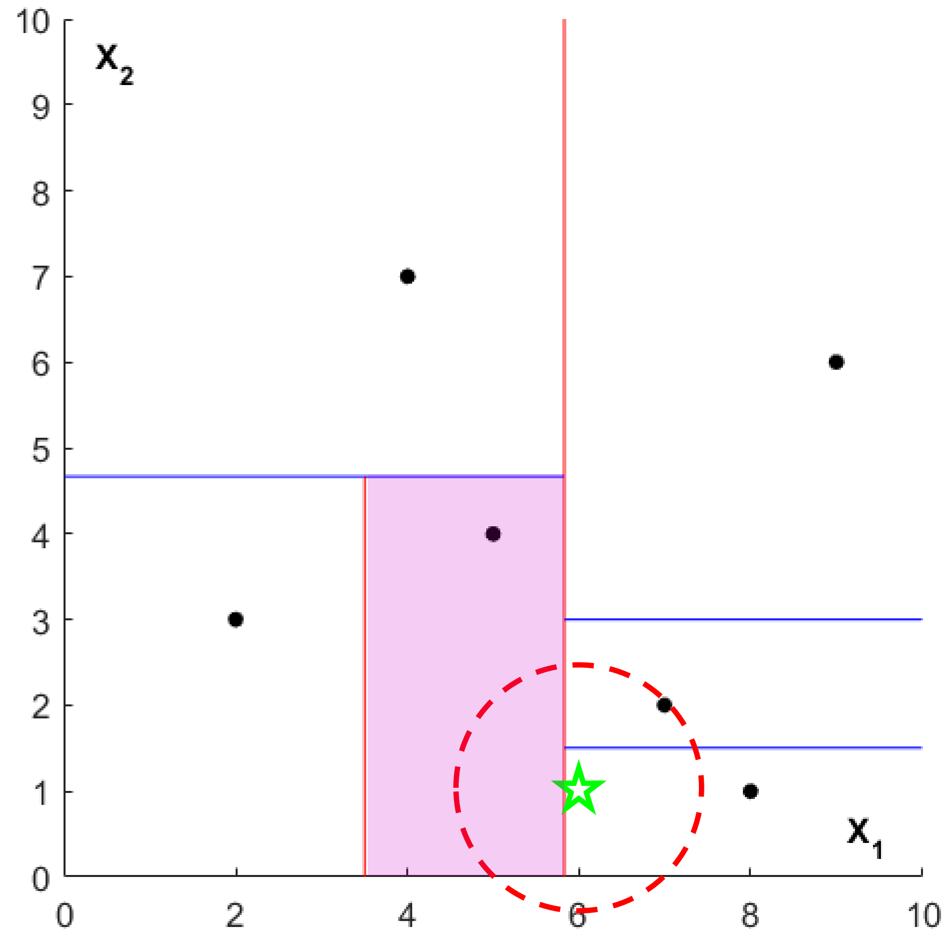

6 > 5.83
x1 = 5.83
R: 6 - 1.4 < 5.83
1 < 4.67
x2 = 4.67
x2 = 3
6 > 3.5
x1 = 3.5
(4,7)
x2 = 1.5
(9,6)
(2,3)
(5,4)
(8,1)
(7,2)
r1 = 2
r2 = 1.4


x2
x1
0
1
2
3
4
5
6
7
8
9
10
0
2
4
6
8
10

k-d trees

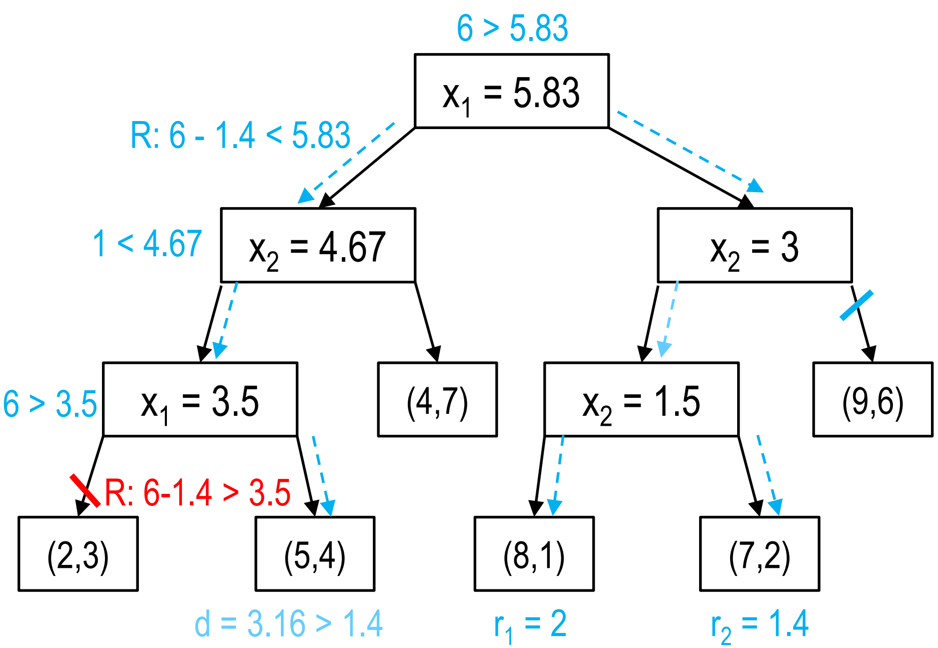
- $q = (6,1)$
- $r_0 = \infty$

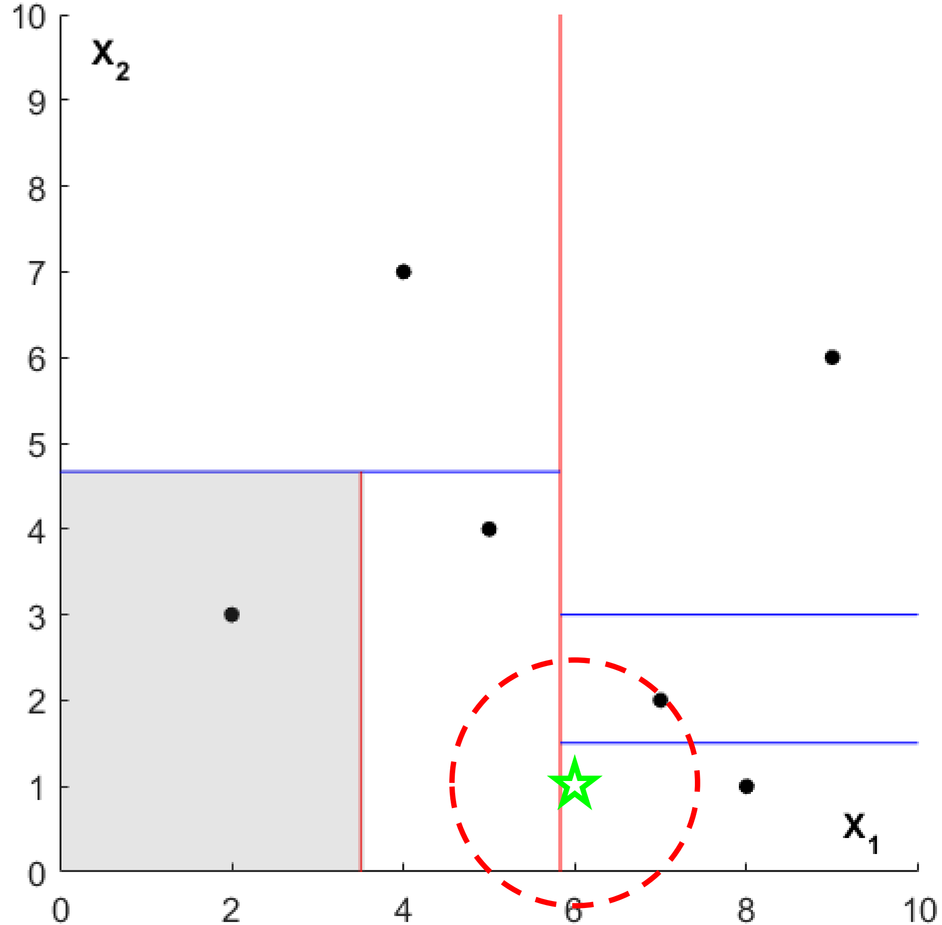




k-d trees

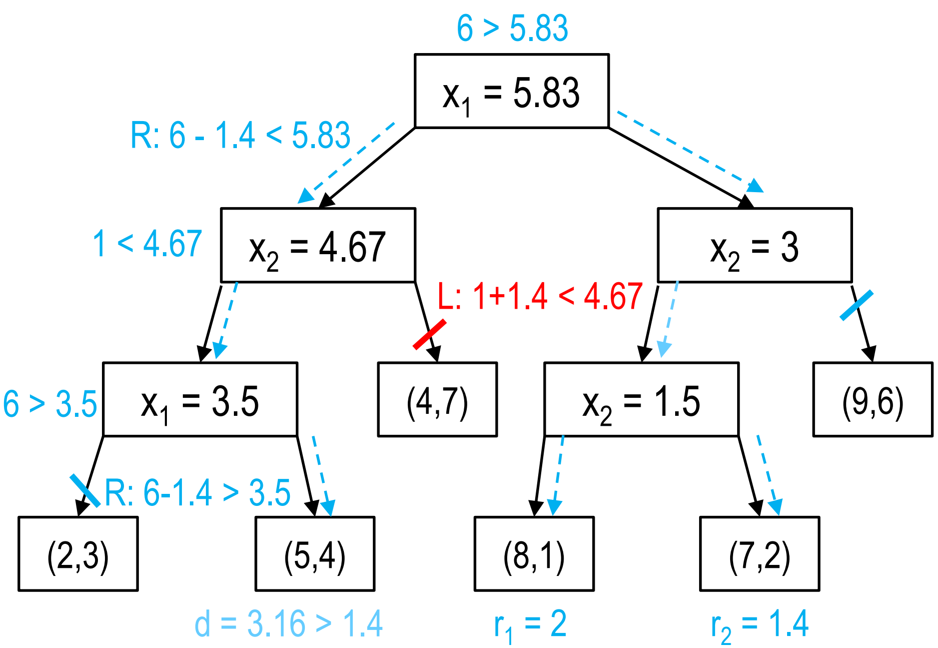
- $q = (6,1)$
- $r_0 = \infty$

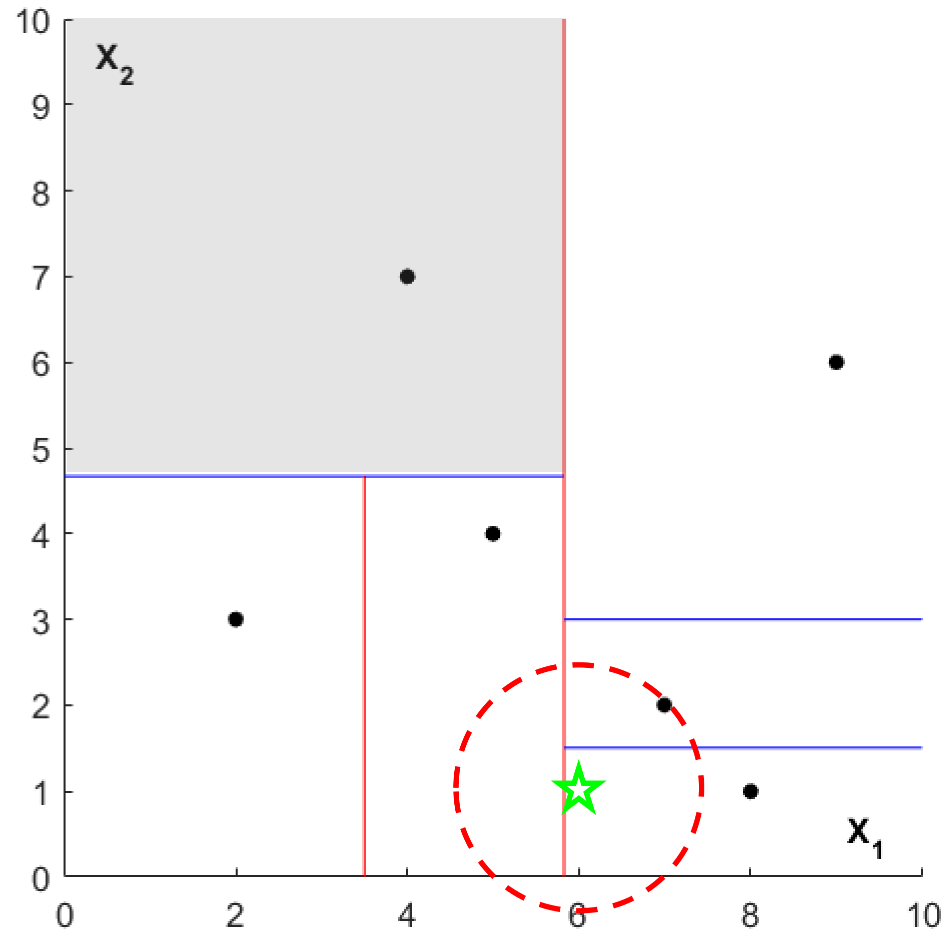




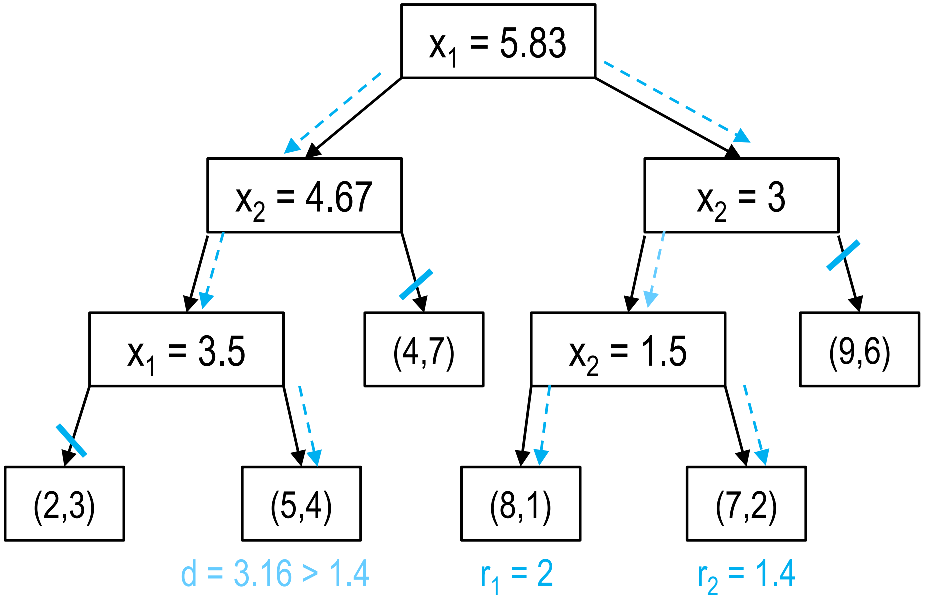
k-d trees

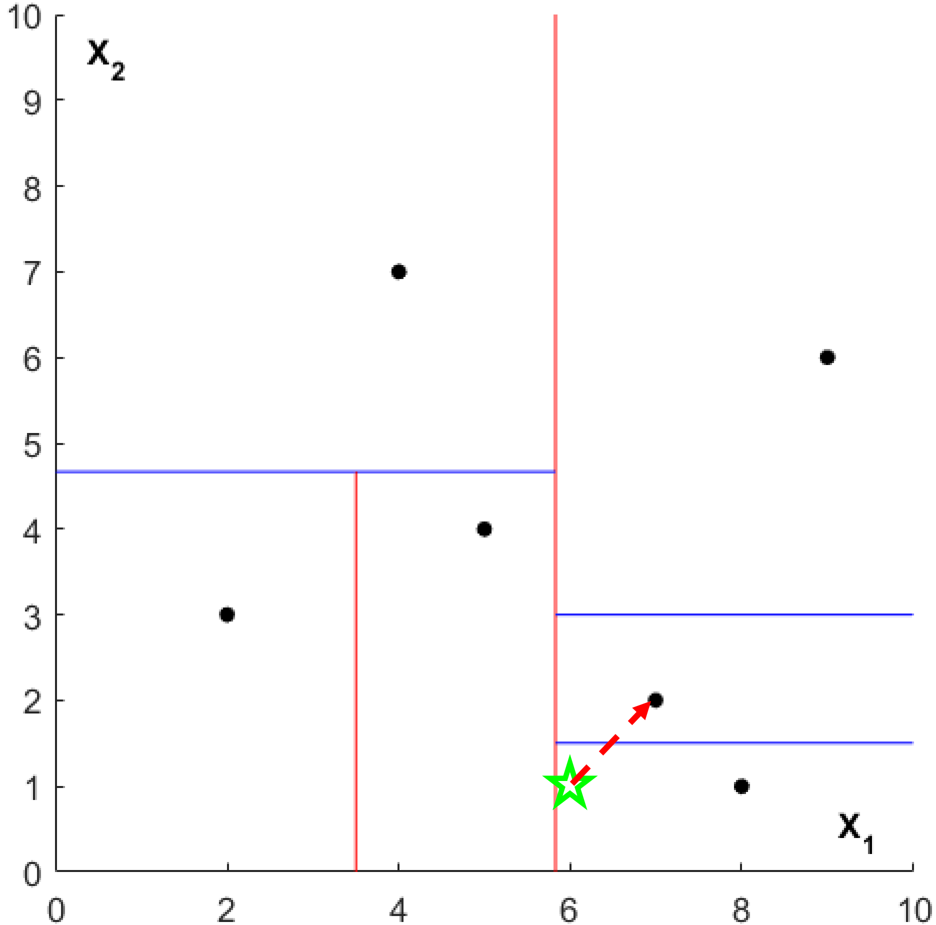
- $q = (6,1)$
- $r_0 = \infty$





- $q = (6,1) \rightarrow NN = (7,2)$, dist = 1.4
- $r_0 = \infty$

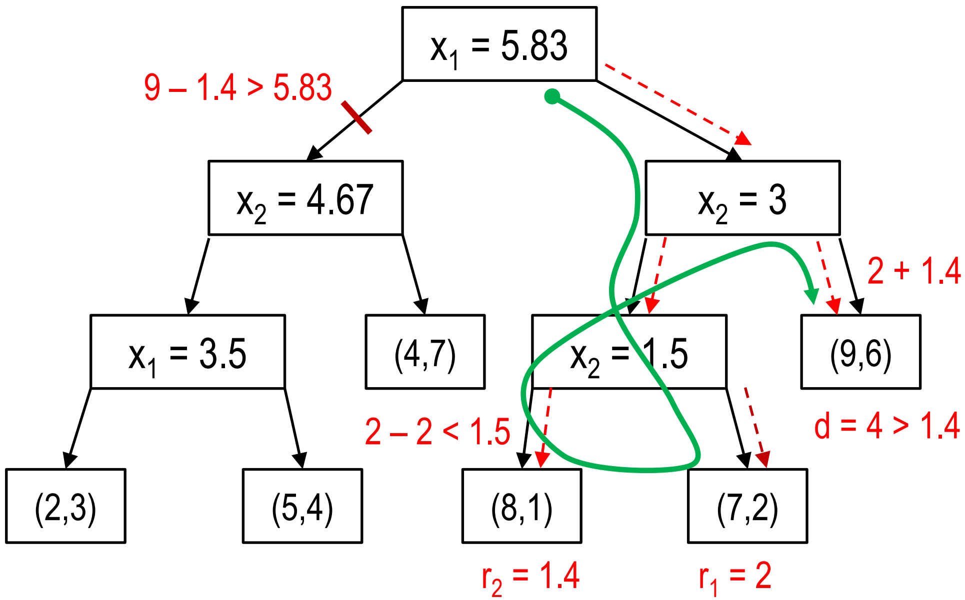


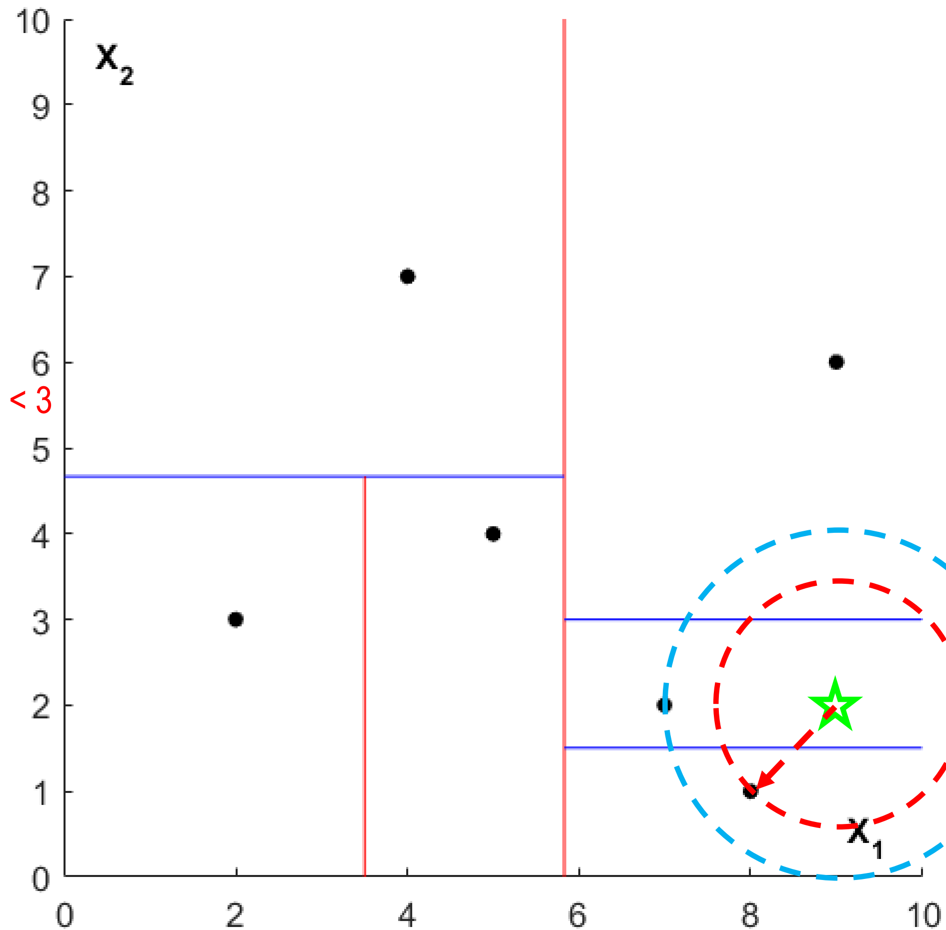


- 11 nodes in the tree, 8 visited, 3 distance calculations

k-d trees

- **Another example**: q = (9,2)





- NN = (8,1), dist = 1.4
- 11 nodes in the tree, 6 visited
  2 distance calculations

- A **larger example**:
  - N = 10^6 samples randomly generated
  - searching for q = (0.29514, 0.897237, 0.941998) randomly generated
  - found NN = (0.296093, 0.896173, 0.948082) at distance 0.00624896
  - visited 44 nodes
- For **1NN**, time **complexity** is O(N) in the worst case, but on average is $O(\log_2 N)$
- If there are **multiple samples at the leaves**, then the nearest neighbor must be searched among all of them:

```
node {
int axis;          // splitting axis
real value;        // splitting value
node left;         // left subtree
node right;        // right subtree
kdpoint points[m]; // if leaf node, a maximum of m nodes
}
nearest(q:in kdpoint, n:in node, b:inout point, r:inout real)
{
    if n.left = empty & n.right = empty then // it is a leaf node (m samples/leaf)
        for every sample i at node n
            r_ = dist(q, n.points[i]);
            if r_ < r then r = r_; b = n.points[i];  // update nearest
    [...]
}
```

k-d trees

- The algorithm can be **extended** in several ways by simple modifications:
  - To search for the **k nearest neighbours**
    - maintain k current best instead of just one
    - prune a branch search only when k points have been found and the branch cannot have points closer than any of the k current bests
    - For **kNN** , time **complexity** is O(kN) in the worst case, but on average is $O(k \log_2 N)$
  - It can also be converted to **approximate nearest neighbour** search to run faster, e.g.
    - set an upper bound on the number of points to examine in the tree, or
    - interrupt the search based upon a real time clock (may be more appropriate in hardware implementations)

- Some recommendations:
  - For **small datasets** and **reduced dimensionality**, **brute force** performs well
  - If data is **sparse with reduced dimensionality** (< 20), ***k*-d tree** performs better than ball-trees
  - As the number of neighbours **k increases**, the **query time** of both *k*-d trees and ball-trees **increases**

Contents

Condensed Nearest Neighbour

- Another alternative to decrease the cost of an NN search is to **reduce the size of the training set** without significantly altering the classification accuracy.
- **Condensed Nearest Neighbour** (CNN):

Use a set of prototypes $U \subseteq X$ to classify, instead of the full set X

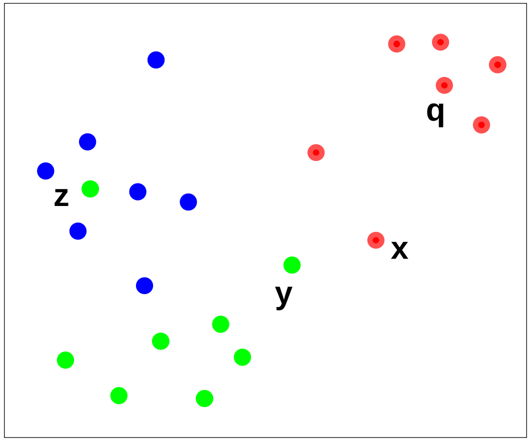
stage 1: discard outliers

1. Remove outliers: go through X, removing each point in turn, checking whether it is recognized as the correct class; if not, then it is an ***outlier*** and it is removed from X

stage 2: choose prototypes

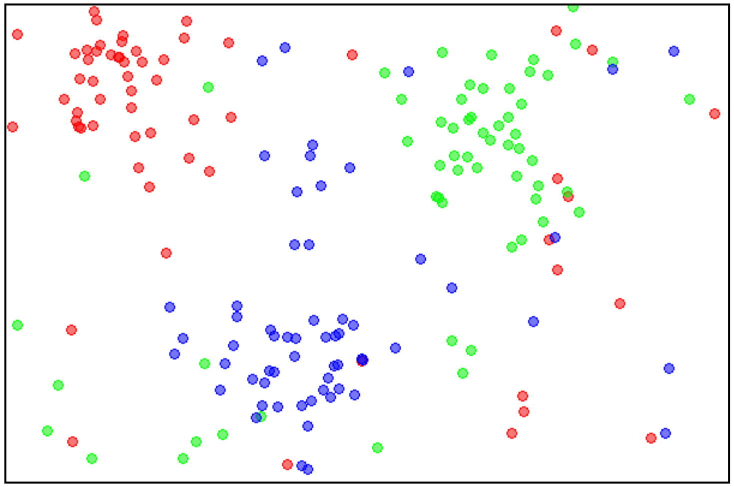
2. U = {random point from X}
3. Build the prototype set U: go through X, picking any point and checking whether it is recognized as the correct class according to U and 1-NN; if it is, then it is an ***absorbed*** point (i.e. *interior* point); if not, it is **transferred to the *prototype*** set U (i.e. *border* point)
4. Repeat 3 until no more prototypes are transferred to U

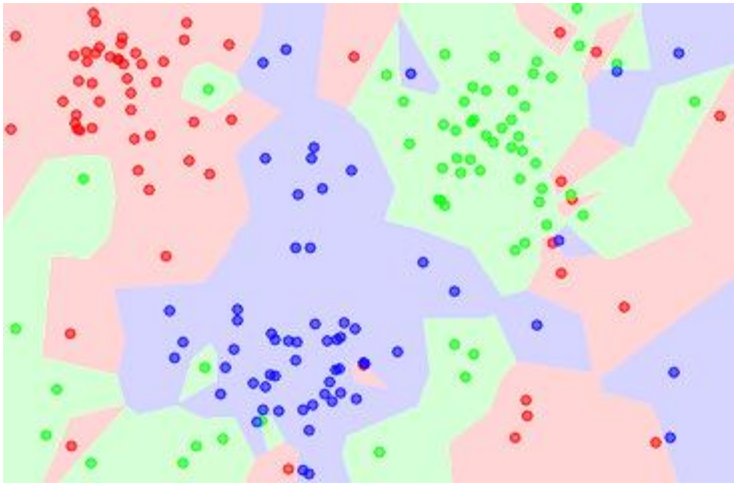
z = outlier
x, y = prototypes
q = absorbed

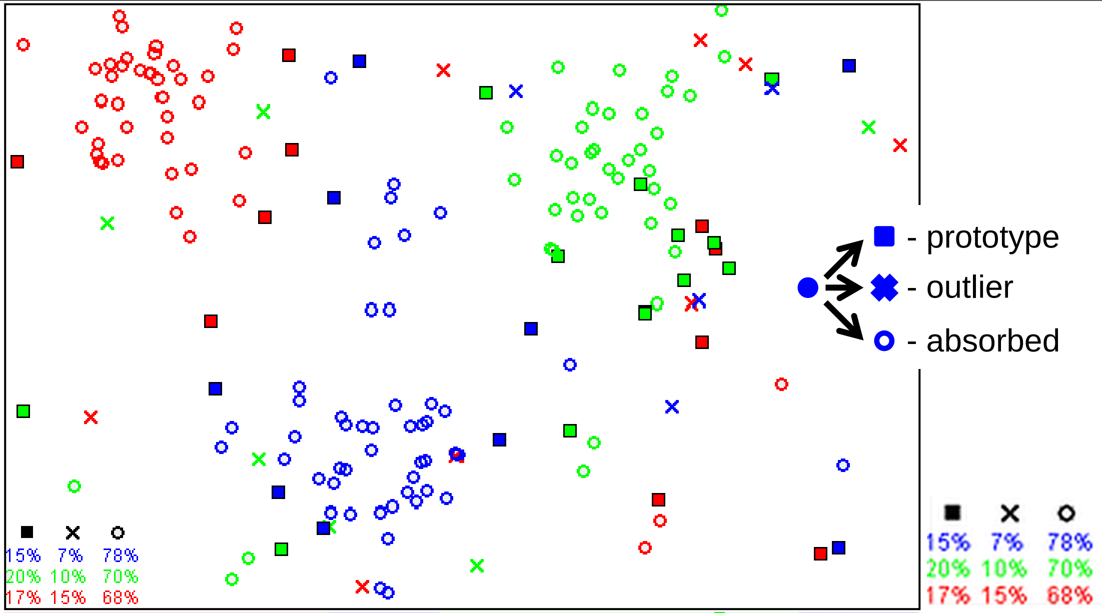


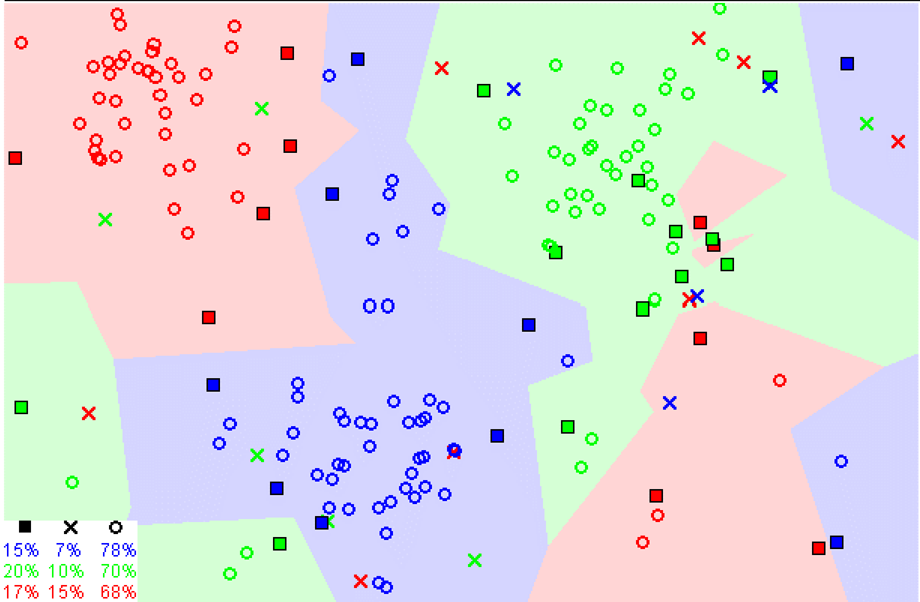
- An **example** (cntd.)

original dataset









Contents

- k-Nearest Neighbours classifier
- Supplementary material: Nearest-Neighbour Search & *k*-d trees
- Supplementary material: Condensed Nearest Neighbours
- Example of use

- Example with **scikit-learn**:

```
model = KNeighborsClassifier (n_neighbors, weights, algorithm,
                                   leaf_size, metric, p)

n_neighbours: number of neighbours to be used for prediction, e.g. 5
weights: weight function used in prediction, posible values
•   'uniform': all points of the k-neighbourhood weigh the same
•   'distance': points are weighted by the inverse of their distance
algorithm: 'auto', 'ball_tree', 'kd_tree', 'brute'
leaf_size: size of the leaf nodes for ball trees and kd-trees, e.g. 30
metric: 'euclidean', 'manhattan', 'chevyshev', 'minkowski'
p: power for the Minkowski metric, e.g. 2
```

```
from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n_neighbors=5, weights='uniform', metric =
'euclidean')
clf.fit(X_train,y_train)
y_test = clf.predict(X_test)
```

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

Instance-based learning: k-Nearest Neighbours

Universitat de les Illes Balears
Departament de Ciències Matemàtiques i Informàtica

11752 Aprendizaje Automático
11752 Machine Learning
Máster Universitario
en Sistemas Inteligentes

Alberto ORTIZ RODRÍGUEZ