Nearest neighbors: performance

Lecture 7

by Marina Barsky

K-NN algorithm. Summary

- The training set ***is the*** model
- Advantages:
  - Building a classifier: zero work
  - Updating the model with every new record: zero work
  - Interpretable: we can justify our classification
  - Good for predicting numeric values (Regressor)
- Disadvantages:
  - The query is computationally expensive!

K-NN query performance: time and space

Space: O(*M x N*)

Running time: O(*M x N*)

- M – number of attributes
- N – total instances in the training set

K-NN performance improvements

Heuristic algorithms:

1. IB2: save memory, speed up classification
2. IB3: deal with noise

Data structures:

1. KD-tree
2. Ball-tree

---

*) IB – Instance Based

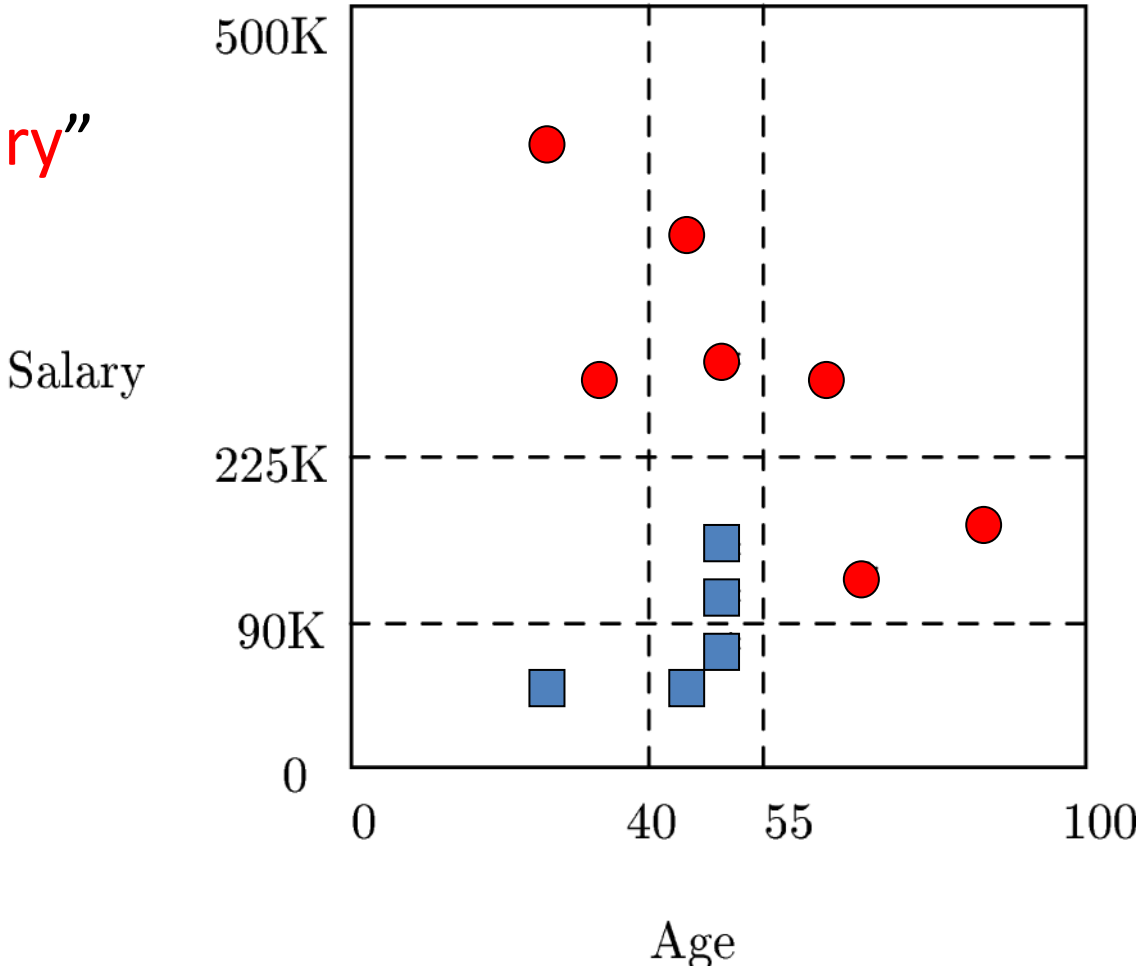
Algorithm example: IB2

main idea

- Work incrementally
- Only incorporate misclassified instances

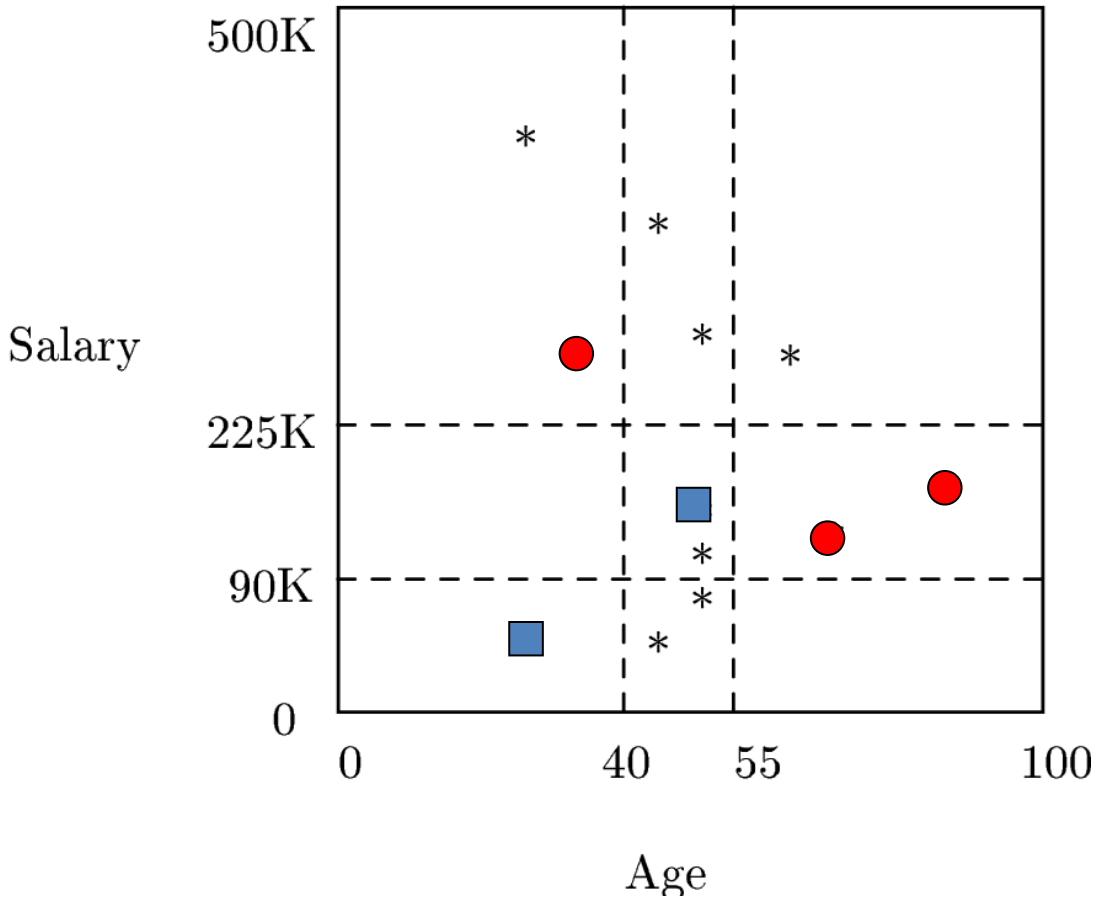
Example: IB2

Dataset:
"Who buys gold jewelry"



IB2 example

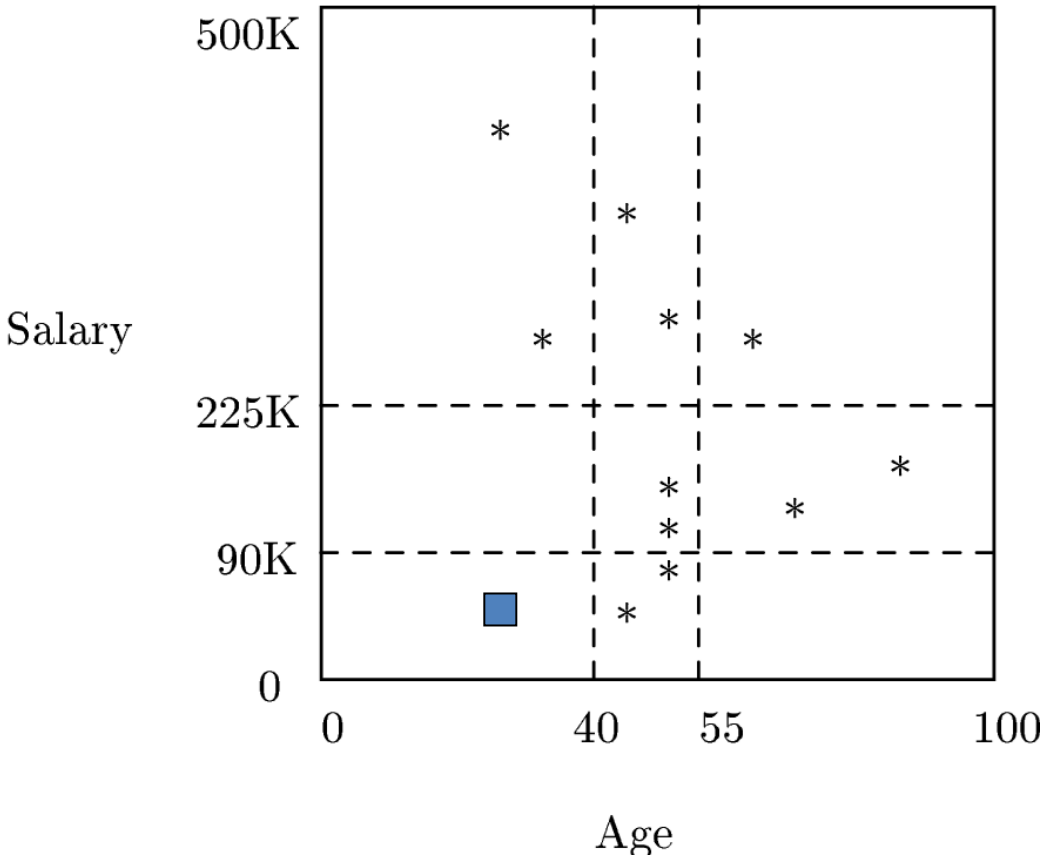
- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - (30,260,yes)
  - (50,75,no)
  - (50,120,no)
  - (70,110,yes)
  - (25,400,yes)
  - (50,100,no)
  - (45,350,yes)
  - (50,275,yes)
  - (60,260,yes)



IB2 output: We memorize only these 5 points.

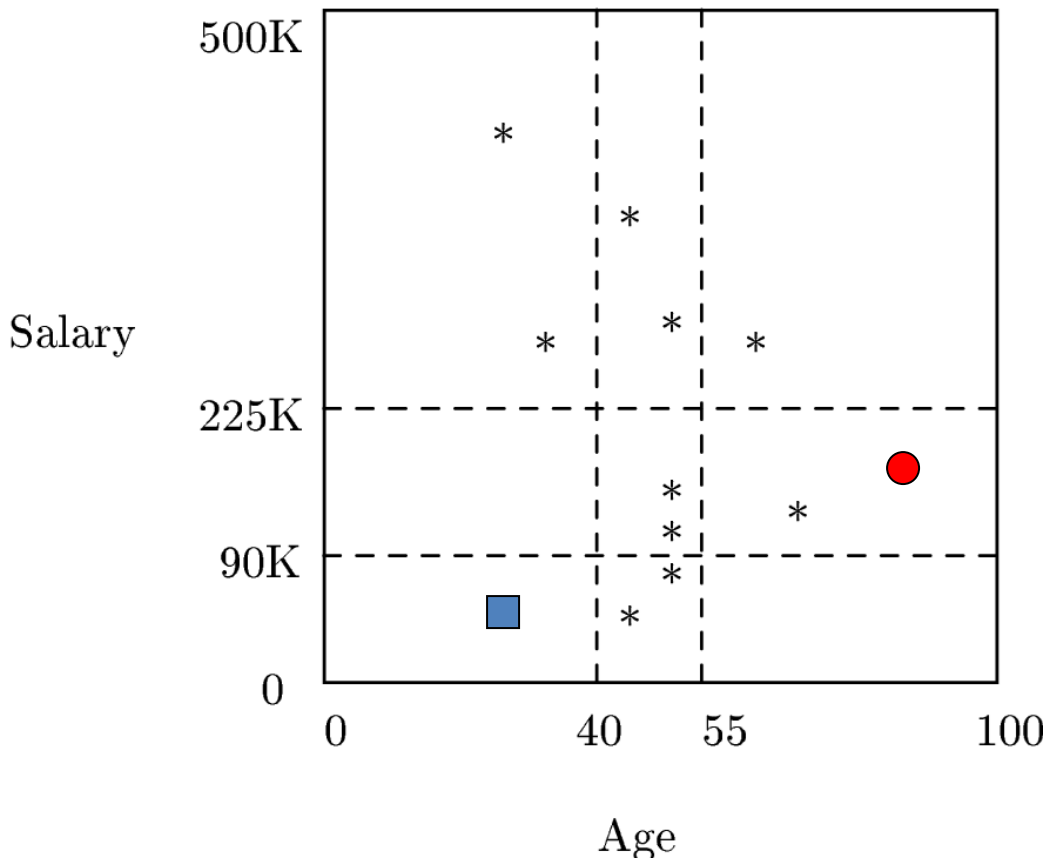
IB2 example

- Data:
  - (25,60,no)



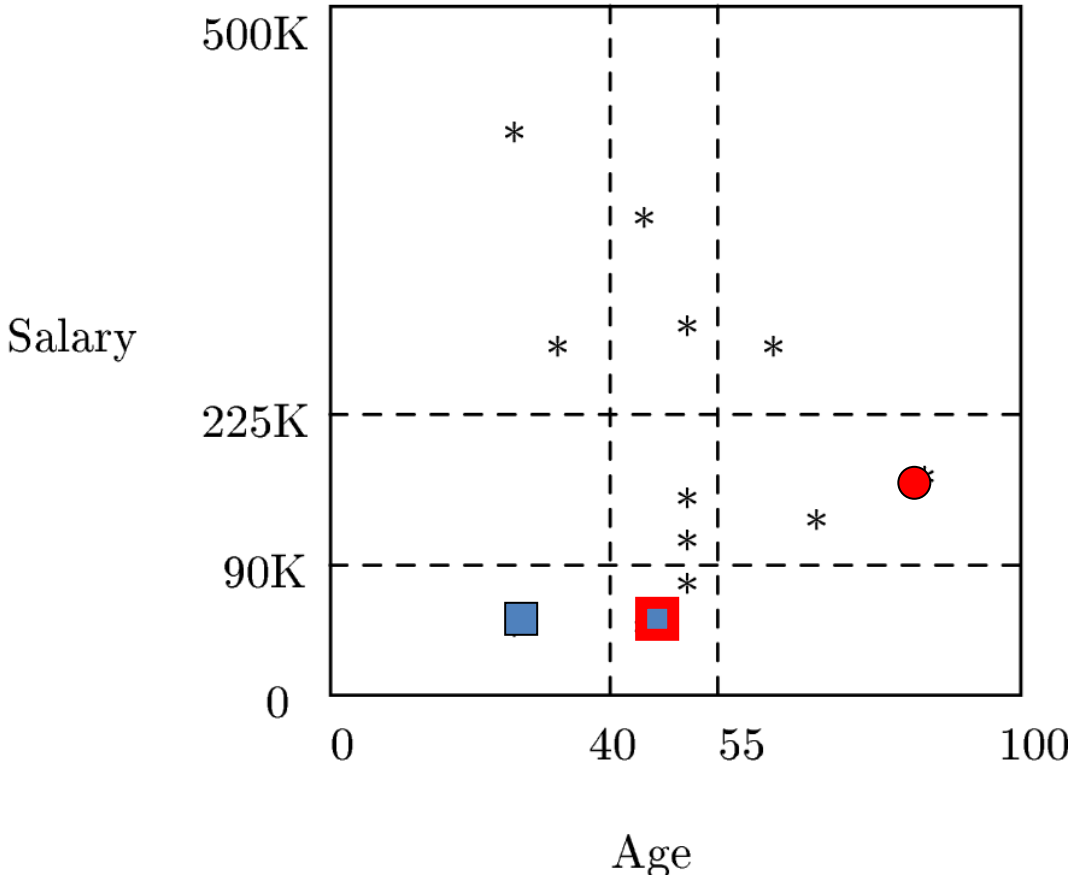
IB2 example

- Data:
  - (25,60,no)
  - *(85,140,yes)*



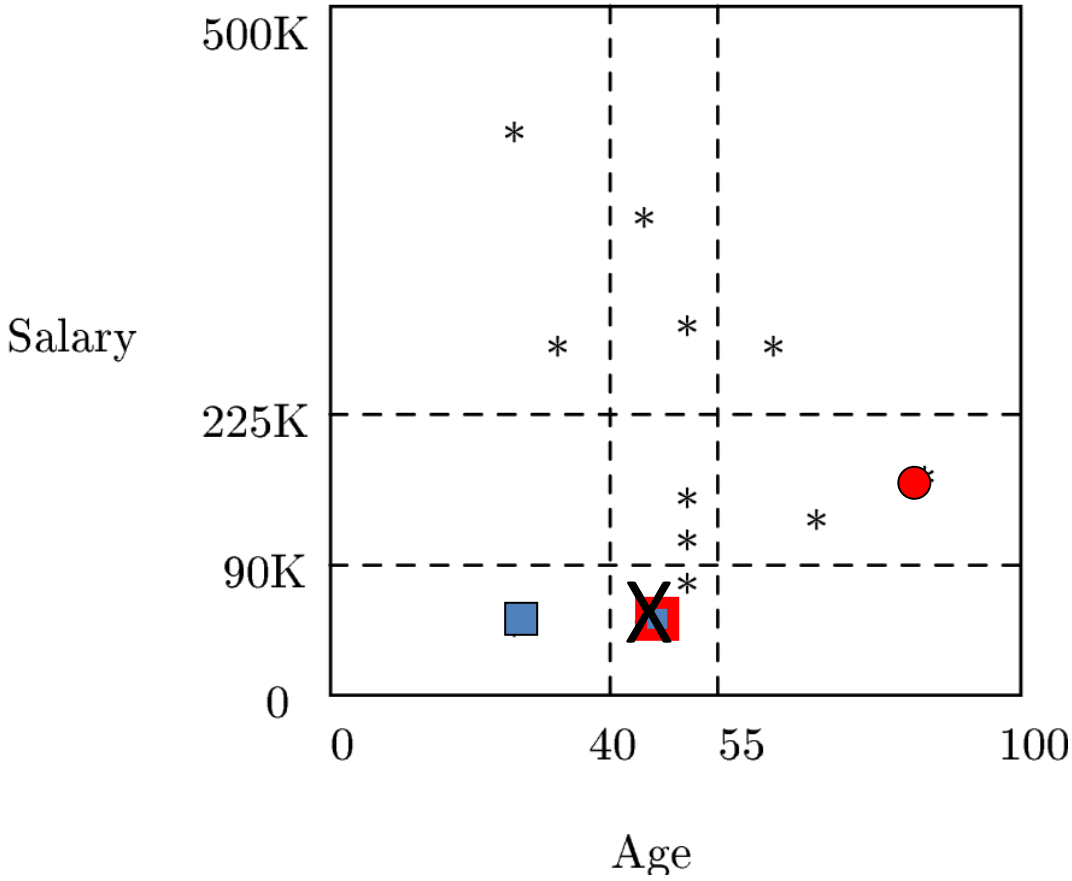
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - *(45,60,no)*



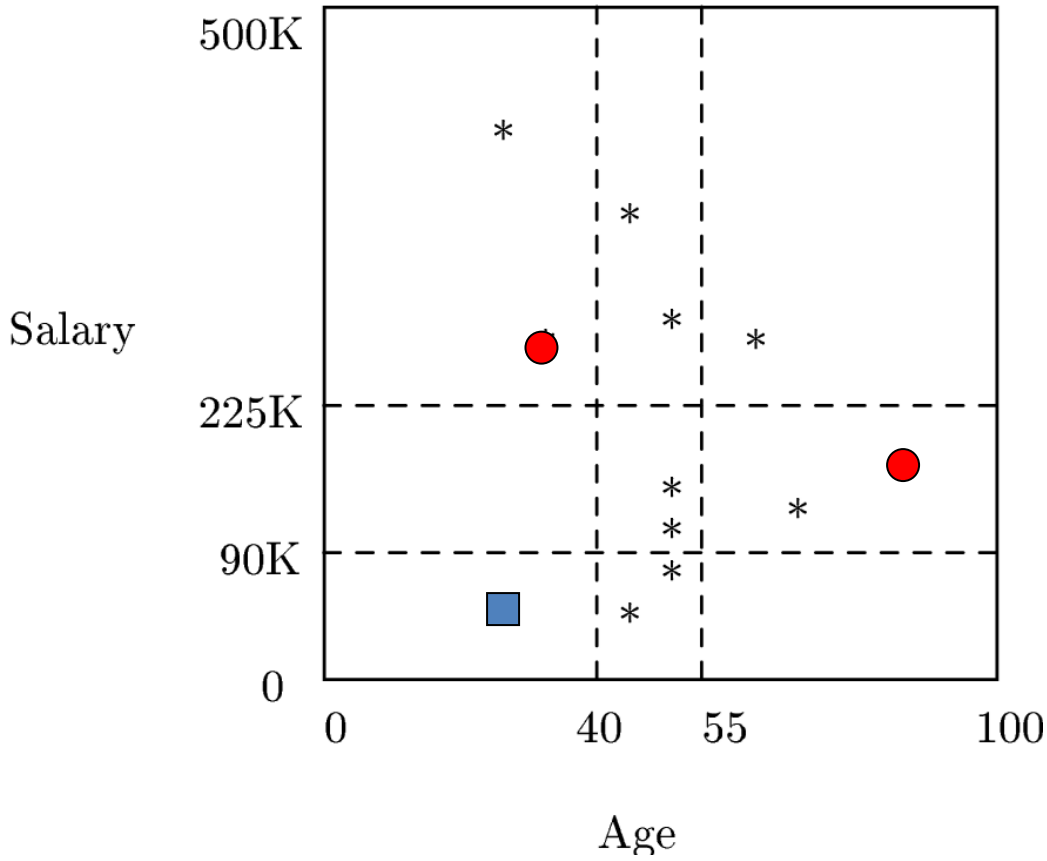
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - *(45,60,no)*



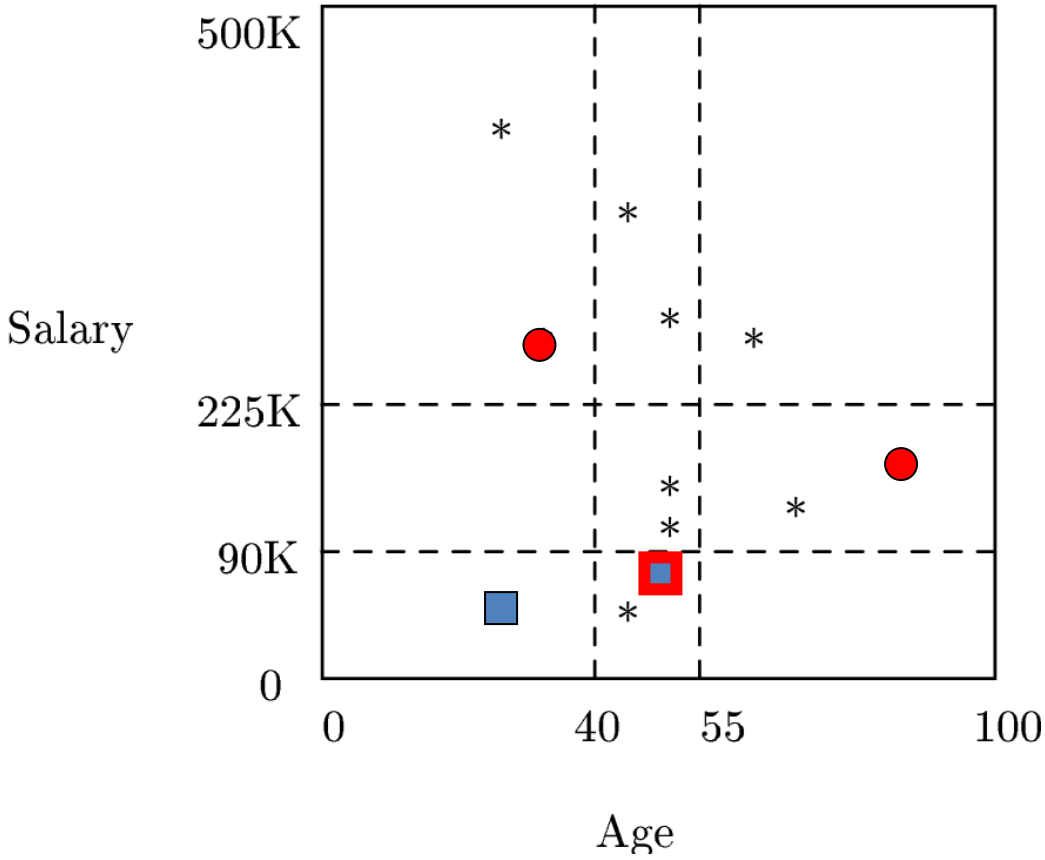
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - *(30,260,yes)*



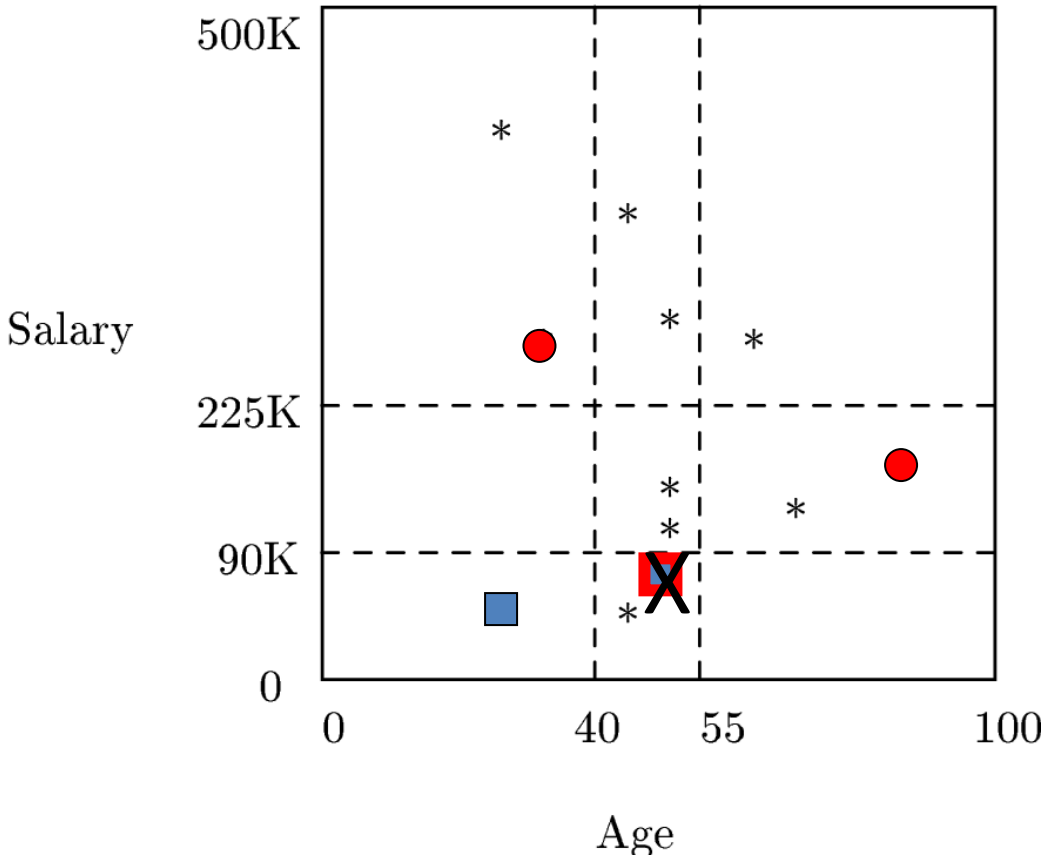
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - (30,260,yes)
  - *(50,75,no)*



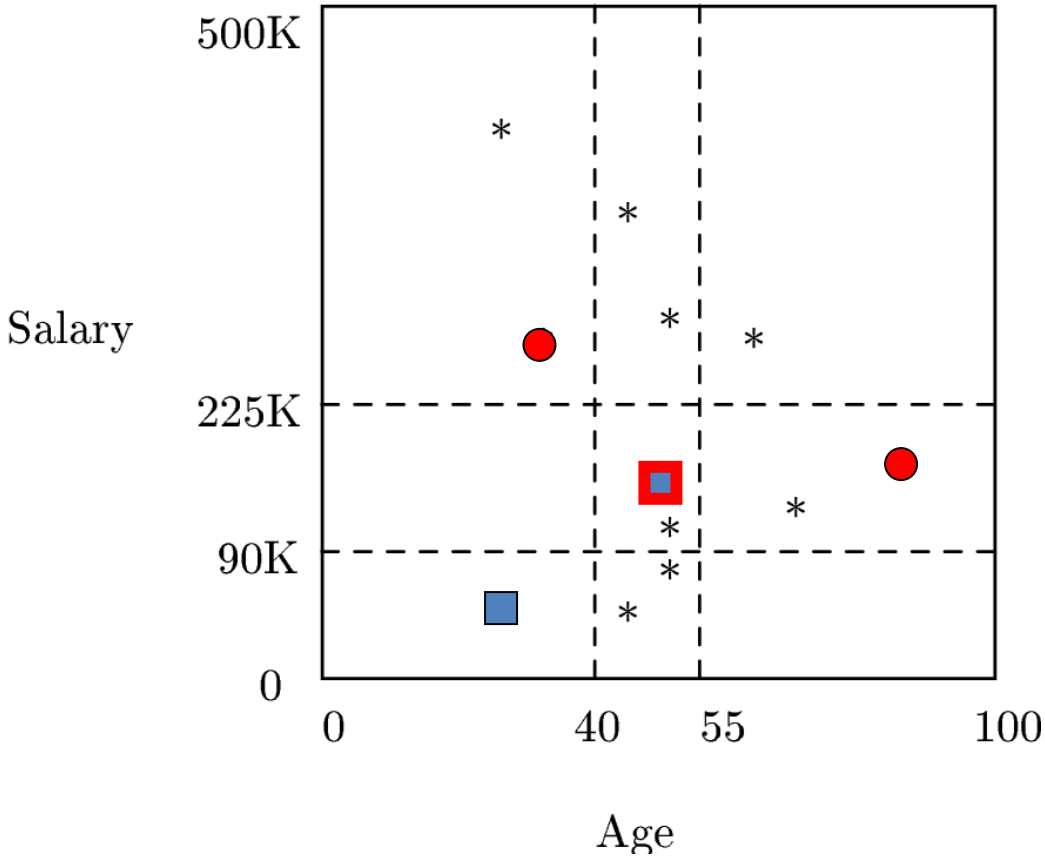
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - (30,260,yes)
  - *(50,75,no)*



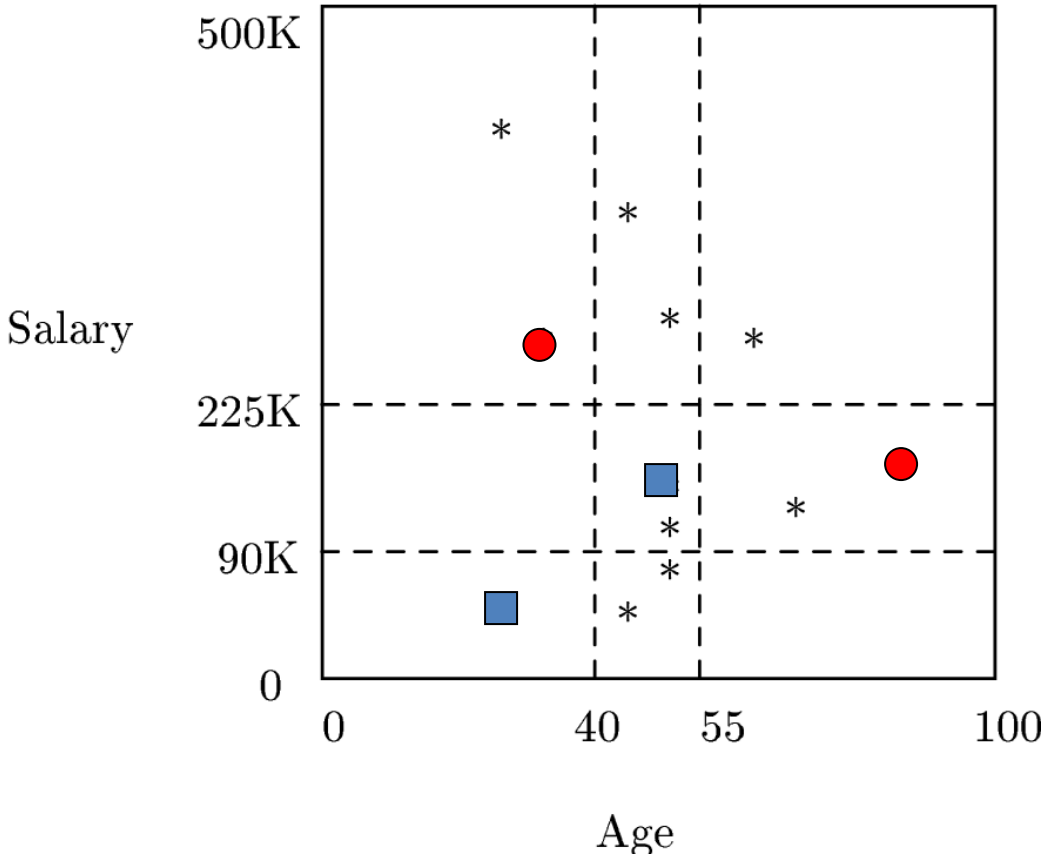
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - (30,260,yes)
  - (50,75,no)
  - *(50,120,no)*



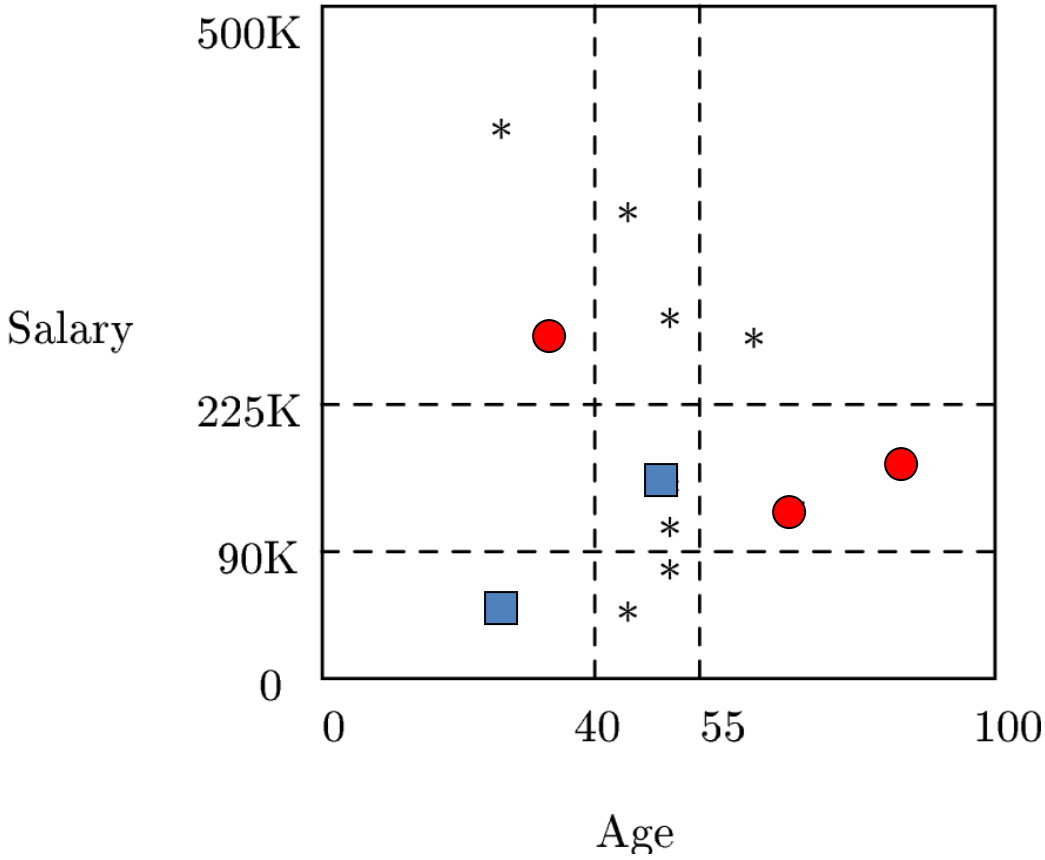
IB2 example

- Data:
  - (25,60,no)
  - (85,140,yes)
  - (45,60,no)
  - (30,260,yes)
  - (50,75,no)
  - *(50,120,no)*



IB2 example

- Continuing in a similar way, we finally get a smaller set to memorize.
  - The colored points are the ones that get memorized.



This is the final answer.
I.e. we memorize only these 5 points.

IB2 summary

- Work incrementally
- Only incorporate misclassified instances

- Problem: noisy data might get incorporated

Data structure example: KD-tree

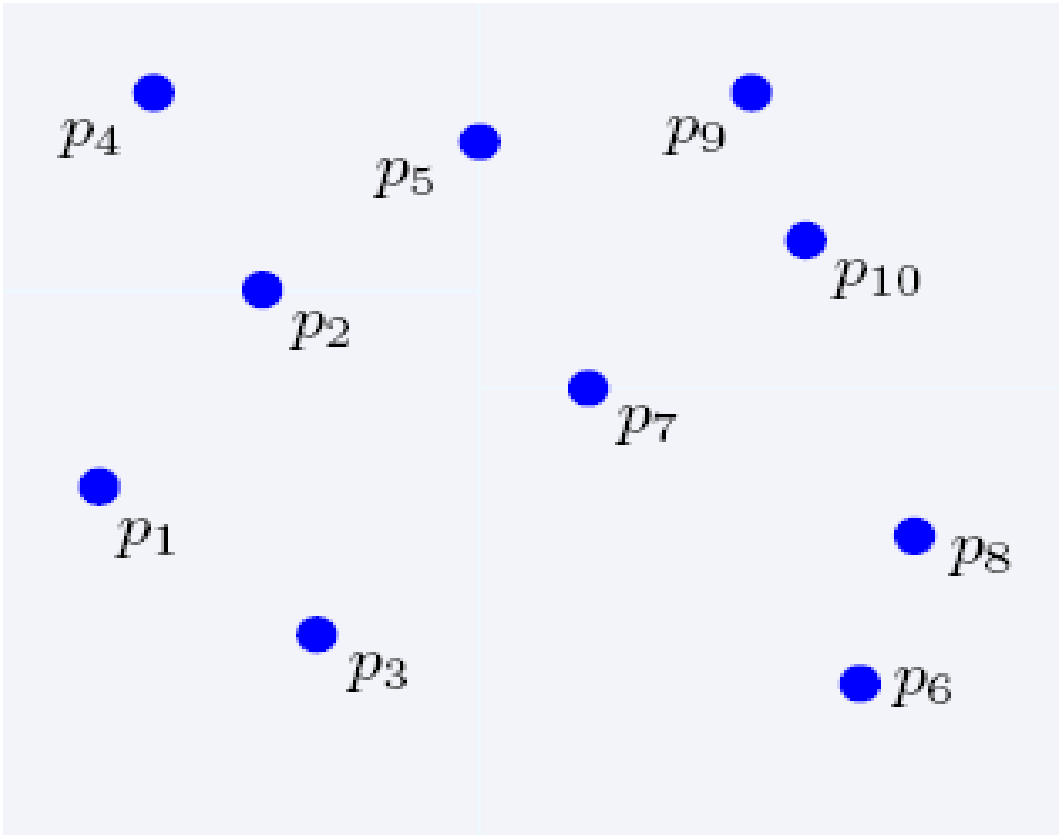
To find nearest neighbors quickly, use a special type of a binary tree: KD-tree

- At the root of the tree we split the set of points into two subsets of same size by a hyperplane vertical to x1-axis (first dimension)
- At the children of the root, the partition is based on the second dimension: x2
- At depth d, we start all over again by partitioning on the first coordinate
- The recursion stops until there is only one point left, which is stored as a leaf

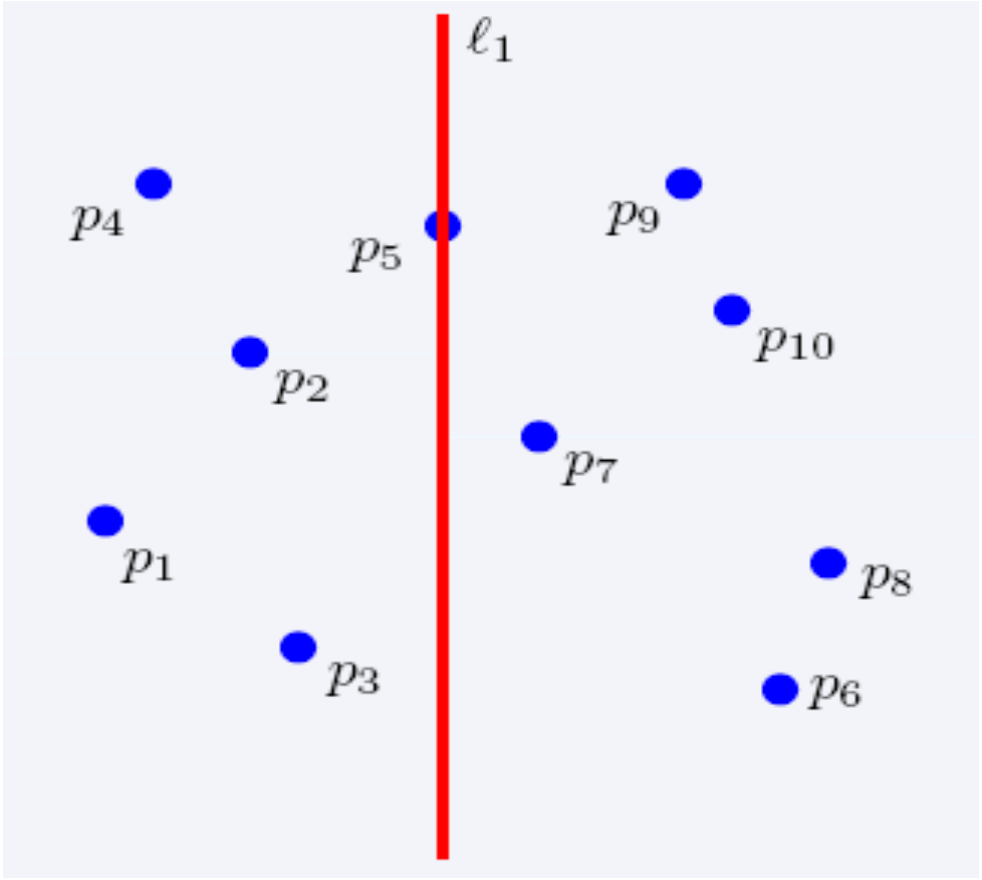
2-dimensional kd-trees

- Algorithm:
  - Choose **x** or **y** coordinate (alternate)
  - Choose the median of the coordinate: this defines a horizontal or vertical line
  - Recurse on both sides
- We get a binary tree:
  - Size **O(n)**
  - Depth **O(logn)**
  - Construction time **O(nlogn)**

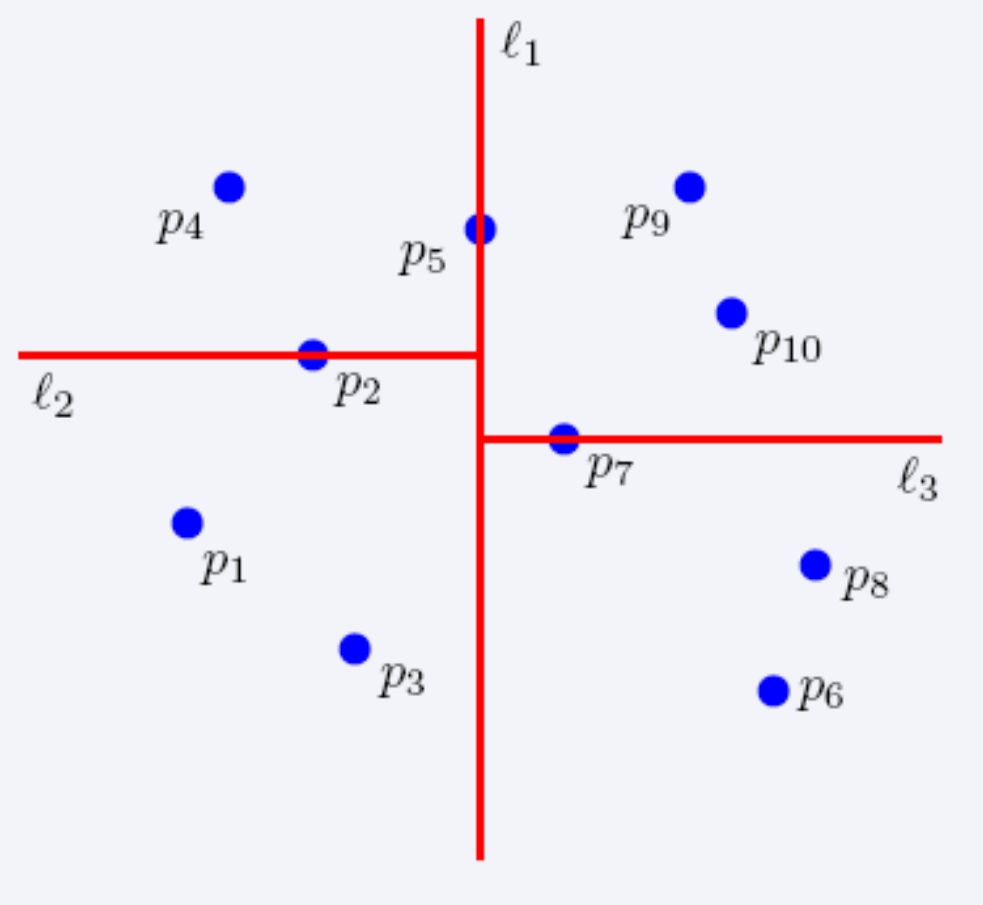
Construction of kd-trees



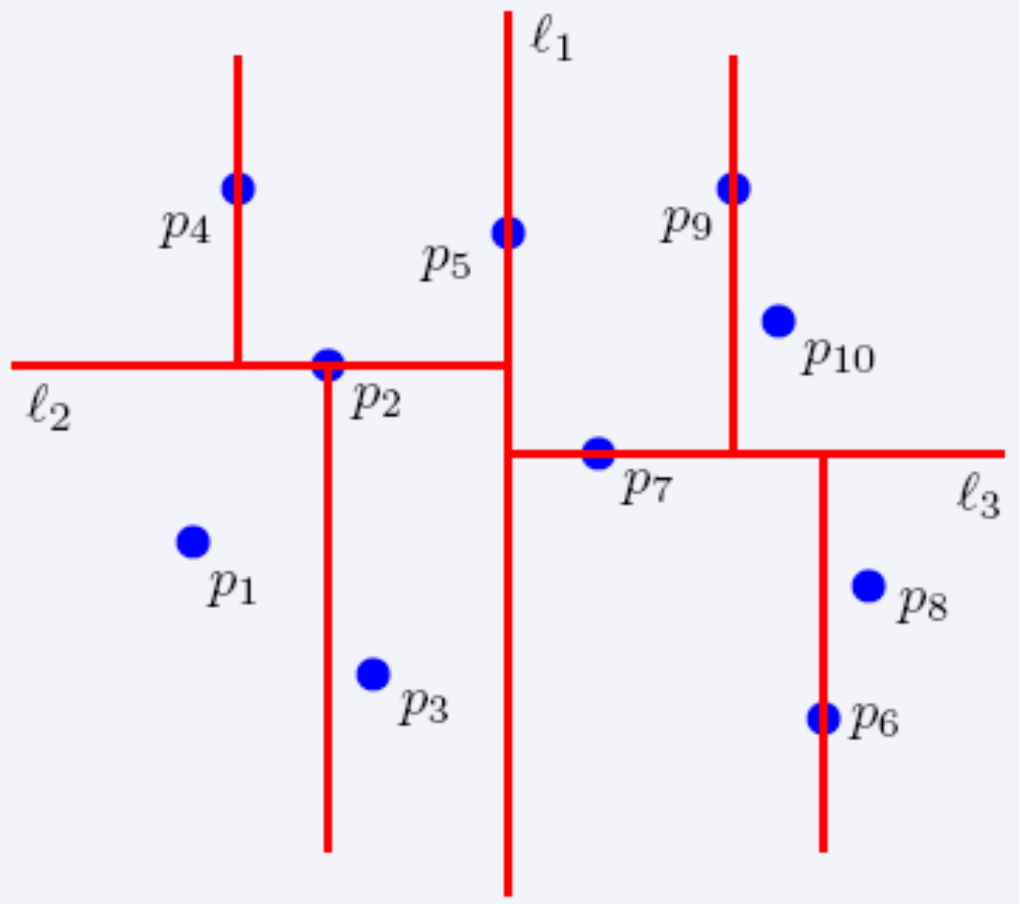
Construction of kd-trees



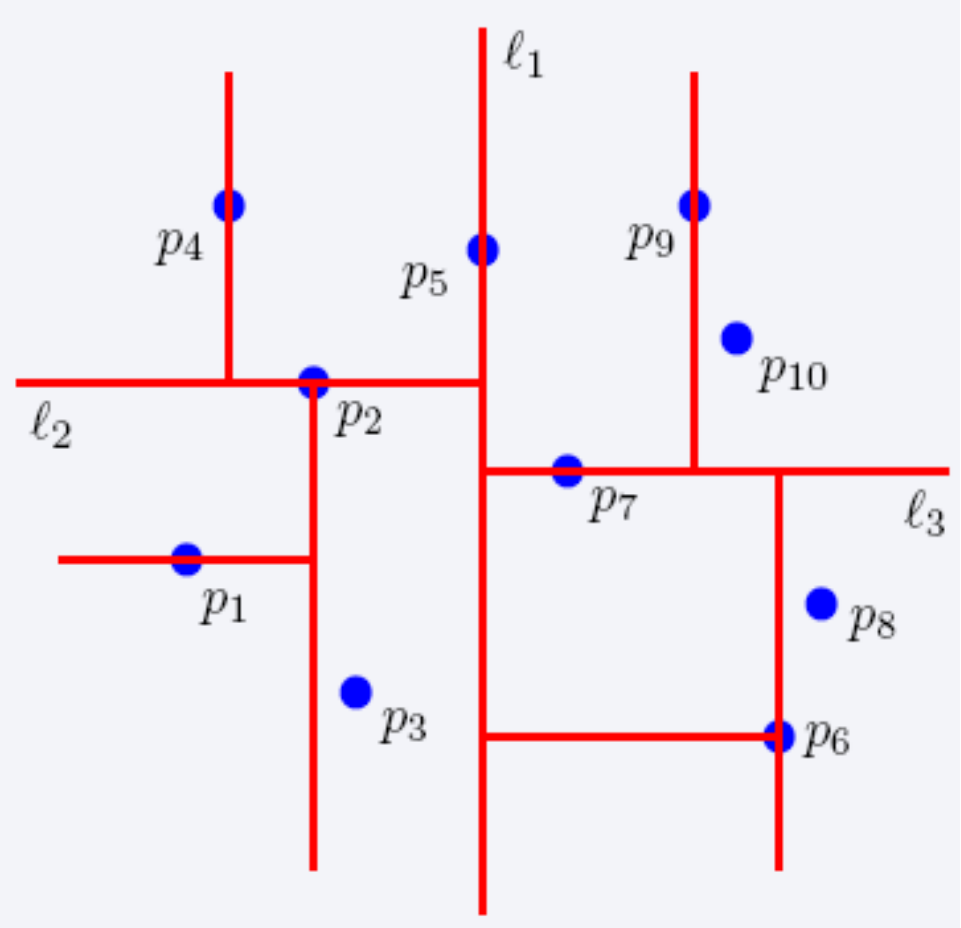
Construction of kd-trees



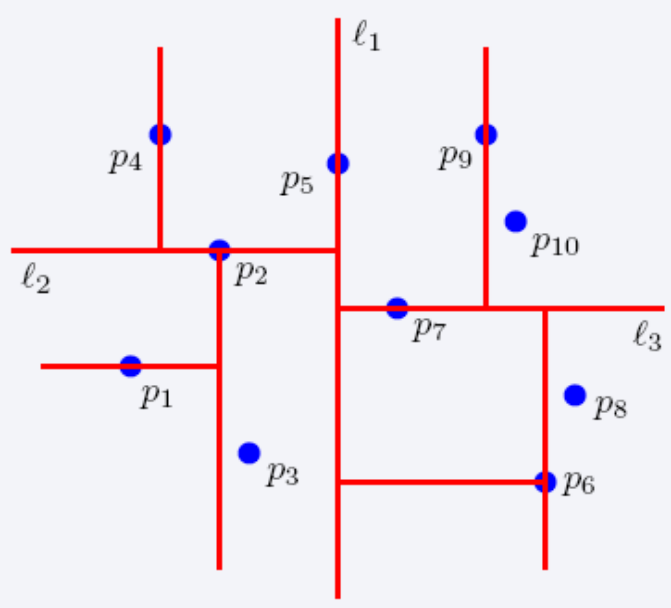
Construction of kd-trees

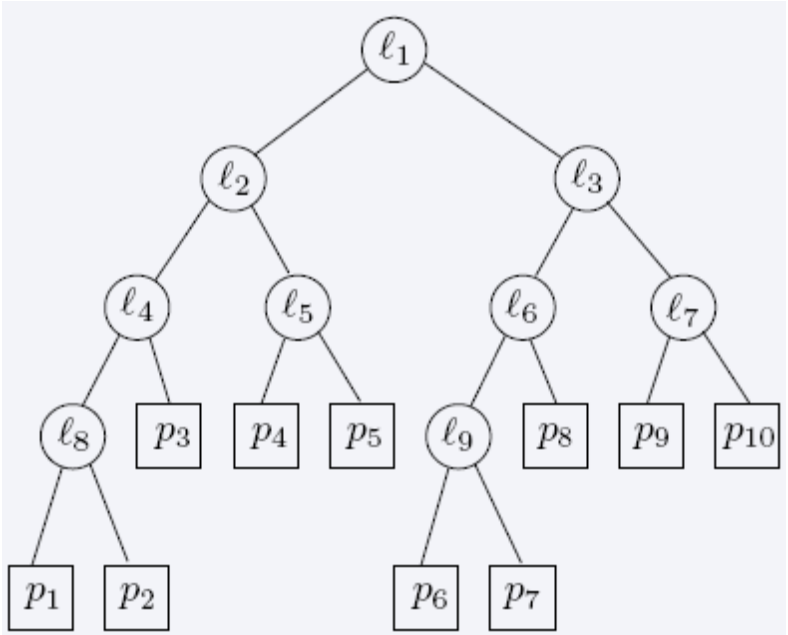


Construction of kd-trees

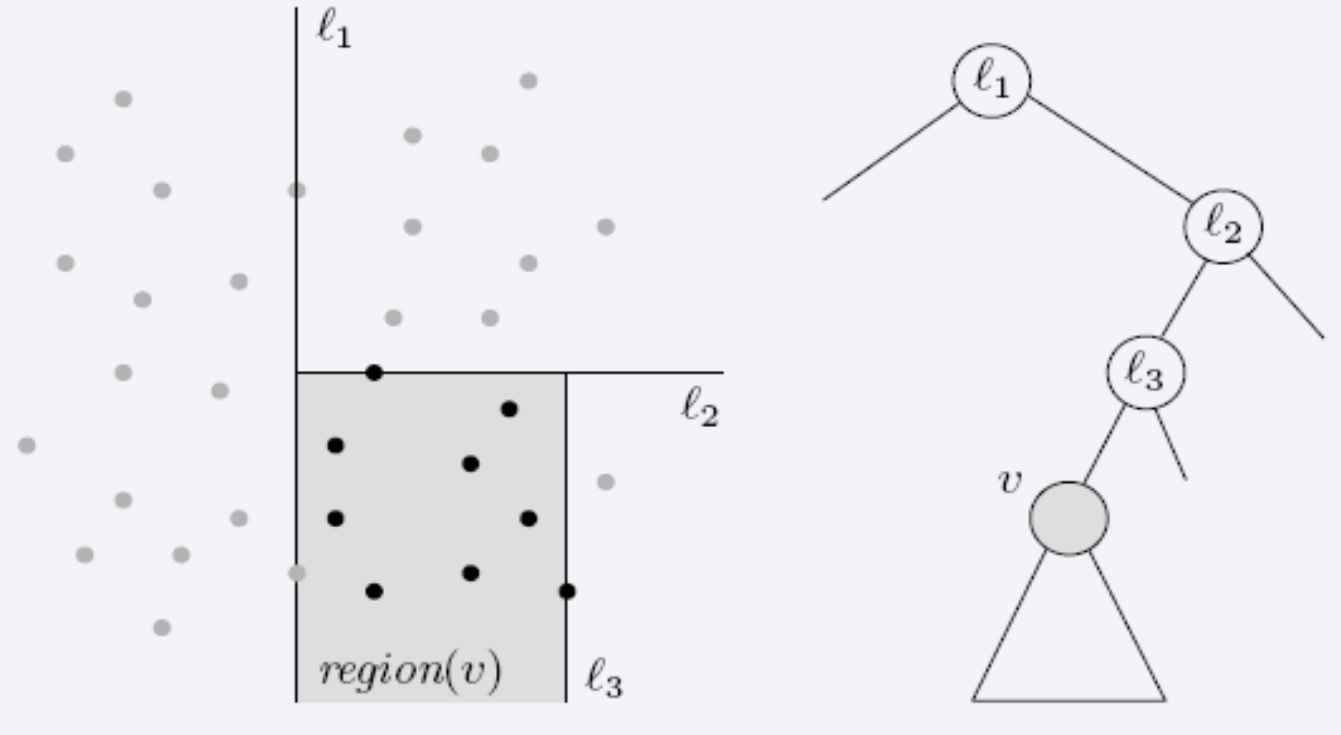


The complete kd-tree





Region of node **v**



Region(v) : the subtree rooted at **v** stores the points in black dots

d-dimensional kd-trees

- A data structure to support range queries in $\mathbf{R^d}$

- Preprocessing time: **O(nlogn)**
- Space complexity: **O(n)**
- Query time: $\mathbf{O(n^{1-1/d}+k)}$