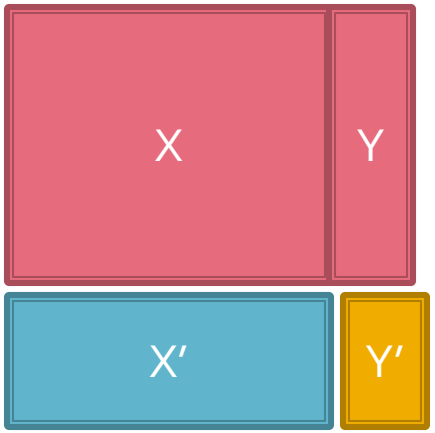
Large Scale Machine Learning: k-NN, Perceptron, Winnow

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
http://cs246.stanford.edu

Supervised Learning

- Would like to do prediction:
  estimate a function f(x) so that y = f(x)
- Where *y* can be:
  - Real number: Regression
  - Categorical: Classification
  - Complex object:
    - Ranking of items, Parse tree, etc.
- Data is labeled:
  - Have many pairs {(x, y)}
    - x … vector of real valued features
    - y … class ({+1, -1}, or a real number)



Training and test set

Large Scale Machine Learning

- We will talk about the following methods:
  - k-Nearest Neighbor (Instance based learning)
  - Perceptron algorithm
  - Support Vector Machines
  - Decision trees

- Main question:
  How to efficiently train
  (build a model/find model parameters)**?**

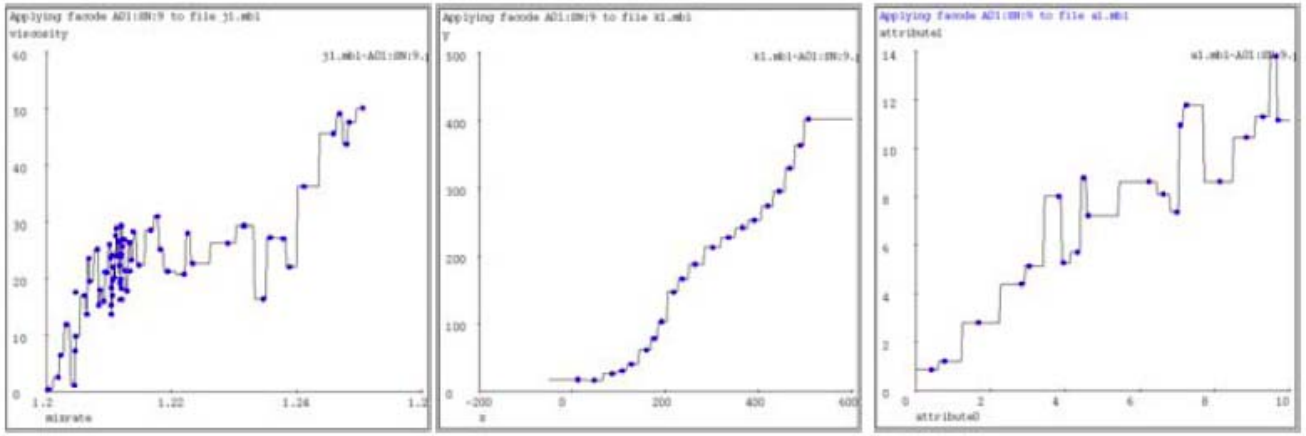
Instance Based Learning

- Instance based learning
- Example: Nearest neighbor
  - Keep the whole training dataset: {(x, y)}
  - A query example (vector) *q* comes
  - Find closest example(s) x*
  - Predict y*

  - Can be used both for regression and classification
    - Recommendation systems

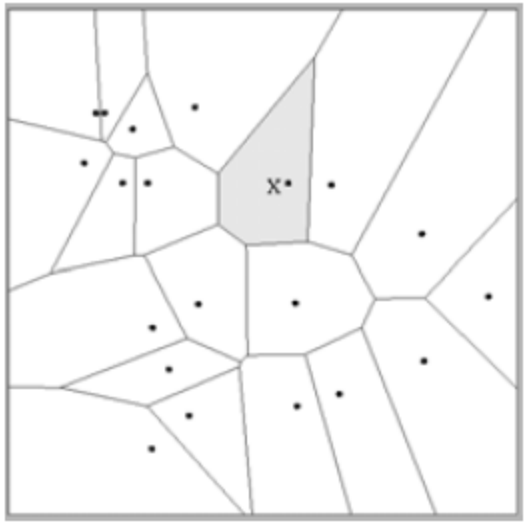
1-Nearest Neighbor

- To make Nearest Neighbor work we need 4 things:
  - Distance metric:
    - Euclidean
  - How many neighbors to look at?
    - One
  - Weighting function (optional):
    - Unused
  - How to fit with the local points?
    - Just predict the same output as the nearest neighbor

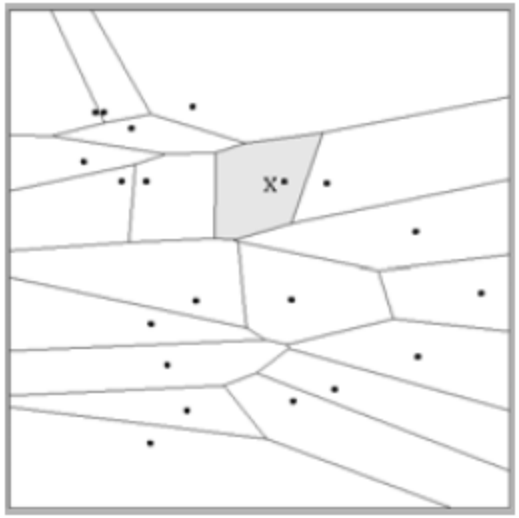


Distance metrics

- Suppose $x_1,\ldots, x_m$ are two dimensional:
  - $x_1=(x_{11},x_{12})$, $x_2=(x_{21},x_{22})$, …
- One can draw nearest neighbor regions:



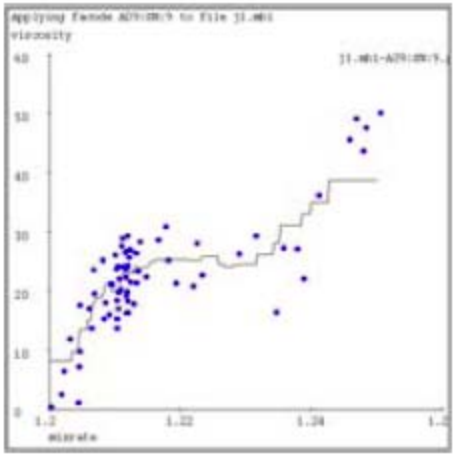
$d(x_i,x_j) = (x_{i1}-x_{j1})^2 + (x_{i2}-x_{j2})^2$

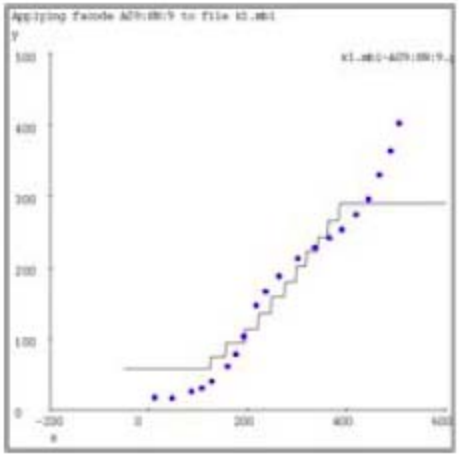


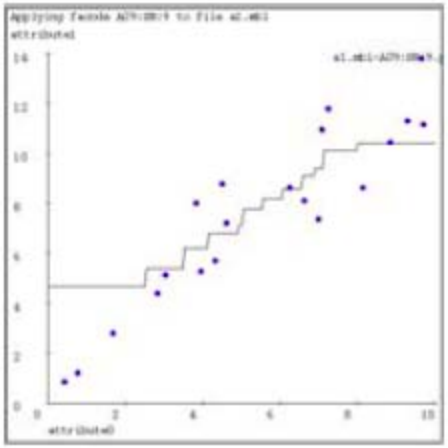
$d(x_i,x_j) = (x_{i1}-x_{j1})^2 + (3x_{i2}-3x_{j2})^2$

k-Nearest Neighbor

- Distance metric:
  - Euclidean
- How many neighbors to look at?
  - *k*
- Weighting function (optional):
  - Unused
- How to fit with the local points?
  - Just predict the average output among *k* nearest neighbors



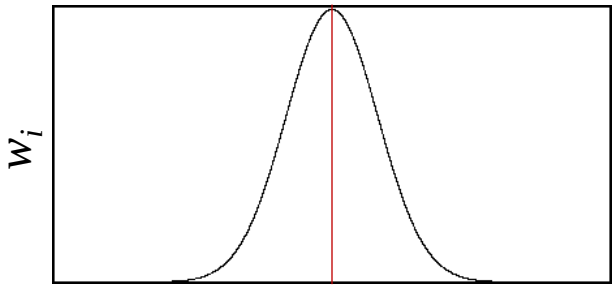




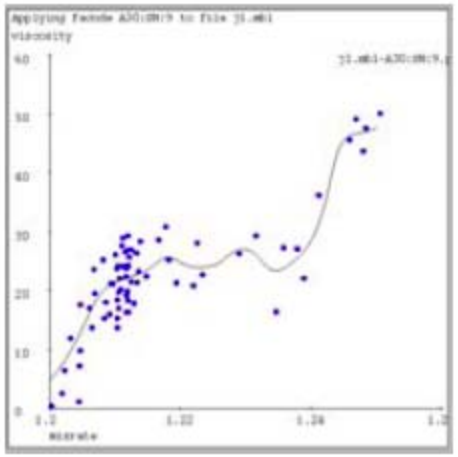
k=9

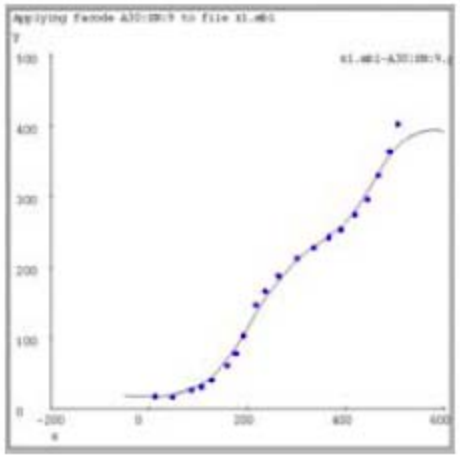
Kernel regression

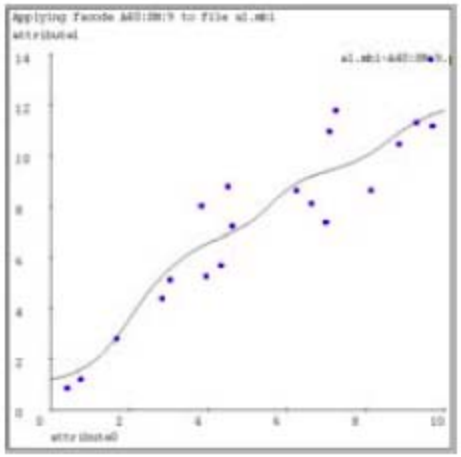
- Distance metric:
  - Euclidean
- How many neighbors to look at?
  - All of them (!)
- Weighting function:
  - $w_i = \exp(-d(x_i, q)^2/K_w)$
    - Nearby points to query q are weighted more strongly. K_w…kernel width.
- How to fit with the local points?
  - Predict weighted average: $\Sigma w_i y_i / \Sigma w_i$



$d(x_i, q) = 0$

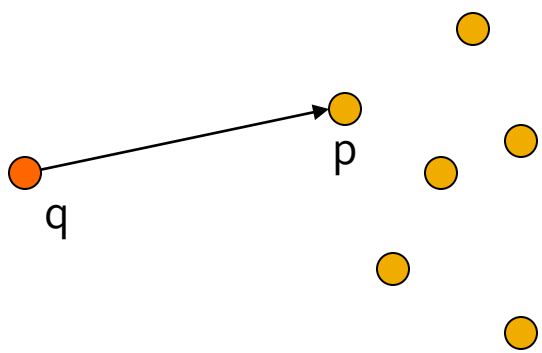






How to find nearest neighbors?

- Given: a set P of n points in R^d
- Goal: Given a query point q
  - NN: find the *nearest neighbor* p of q in P
  - Range search: find one/all points in P within distance r from q

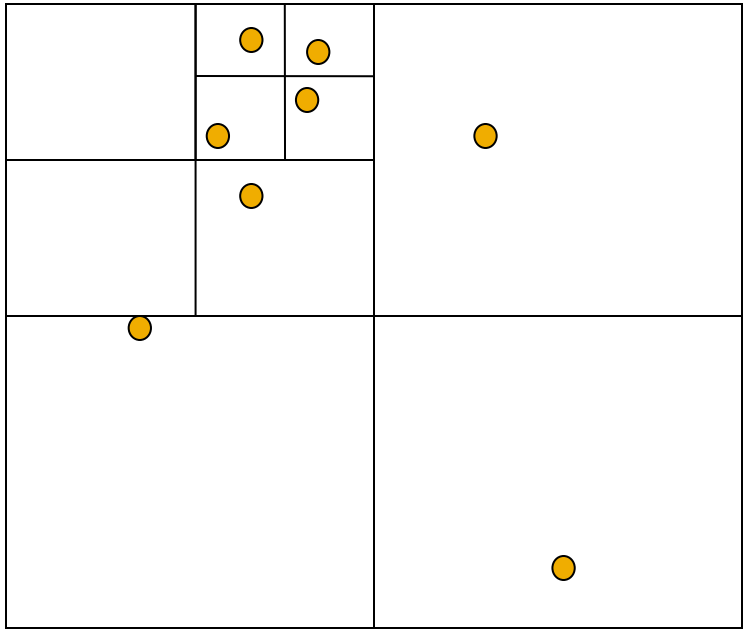


Algorithms for NN

- Main memory:
  - Linear scan
  - Tree based:
    - Quadtree
    - kd-tree
  - Hashing:
    - Locality-Sensitive Hashing
- Secondary storage:
  - R-trees

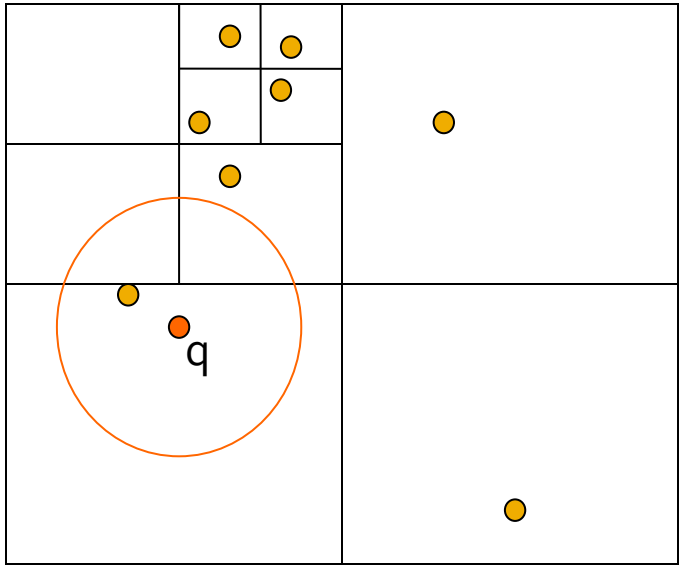
Quadtree (d=~3)

- Simplest spatial structure on Earth!
- Split the space into 2^d equal subsquares
- Repeat until done:
  - only one pixel left
  - only one point left
  - only a few points left
- Variants:
  - split only one dimension at a time
  - Kd-trees (in a moment)



Quadtree: Search

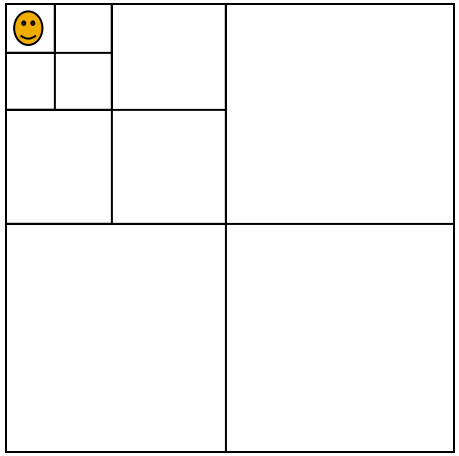
- Range search:
  - Put root node on the stack
  - Repeat:
    - pop the next node *T* from the stack
    - for each child *C* of *T*:
      - if *C* is a leaf, examine point(s) in *C*
      - if *C* intersects with the ball of radius *r* around *q*, add *C* to the stack



- Nearest neighbor:
  - Start range search with $r = \infty$
  - Whenever a point is found, update *r*
  - Only investigate nodes with respect to current *r*

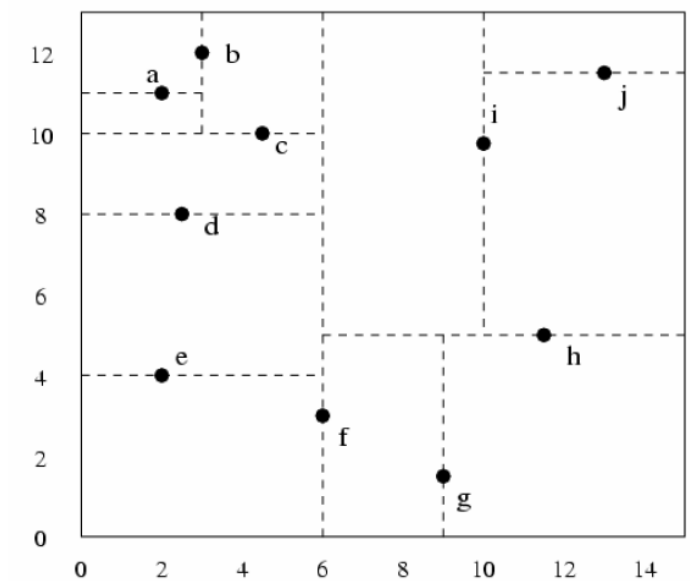
Problems with Quadtree

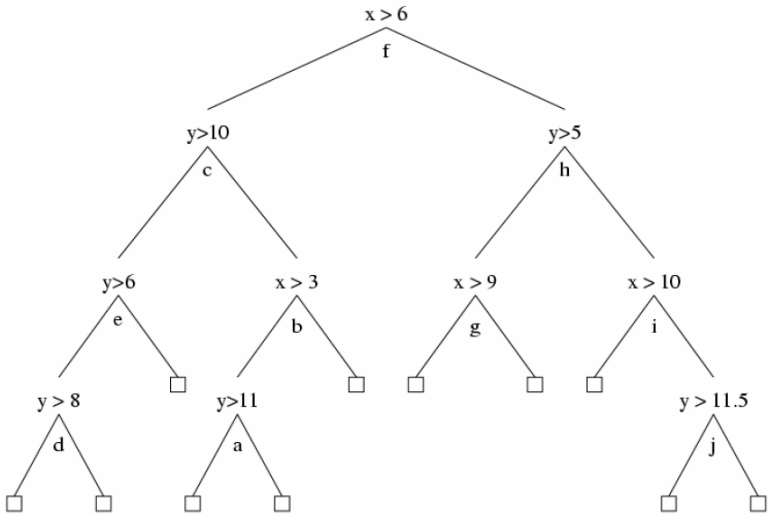
- Quadtrees work great for 2 to 3 dimensions
- Problems:
  - Empty spaces: if the points form sparse clouds, it takes a while to reach them
  - Space exponential in dimension
  - Time exponential in dimension, e.g., points on the hypercube



Kd-tree (d=~10)

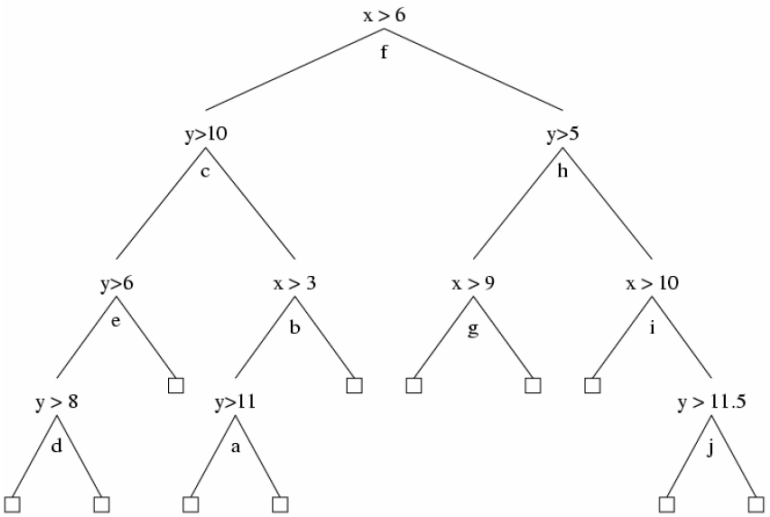
- Main ideas [Bentley '75] :
  - Only one-dimensional splits
  - Choose the split "carefully":
    - E.g., Pick dimension of largest variance and split at median (balanced split)
    - Do SVD or CUR, project and split
  - Queries: as for quadtrees
- Advantages:
  - no (or less) empty spaces
  - only linear space
- Query time at most:
  - Min[*dn*, exponential(*d*)]





Kd-tree (d=~10)

- Range search:
  - Put root node on the stack
  - Repeat:
    - pop the next node *T* from the stack
    - for each child *C* of *T*:
      - if *C* is a leaf, examine point(s) in *C*
      - if *C* intersects with the ball of radius *r* around *q*, add *C* to the stack

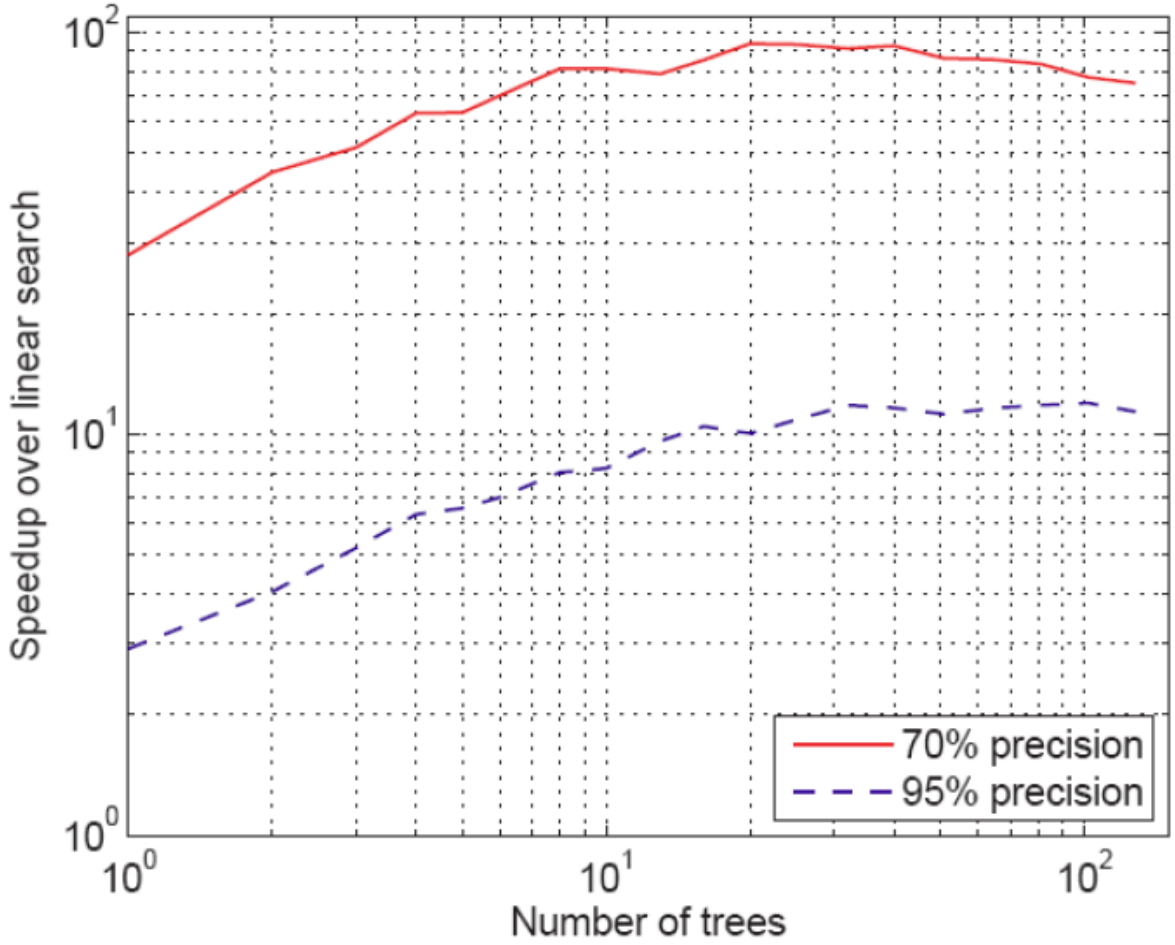


  - In what order we search the children?
    - **Best-Bin-First** (BBF), Last-Bin-First (LBF)

Randomized Kd-trees

- Performance of a single Kd-tree is low
- Randomized Kd-trees: Build several trees
  - Find top few dimensions of largest variance
  - Randomly select one of these dimensions; split on median
  - Construct many complete (i.e., one point per leaf) trees
  - Drawbacks:
    - More memory
    - Additional parameter to tune: number of trees
- Search
  - Descend through each tree until leaf is reached
  - Maintain a single priority queue for all the trees
  - For approximate search, stop after a certain number of nodes have been examined

Randomized Kd-trees

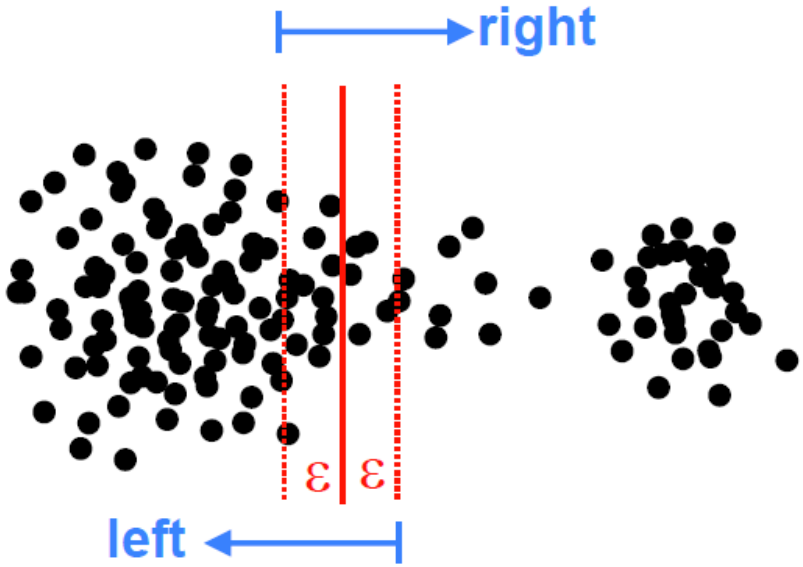


- d=128, n=100k

[Muja-Lowe, 2010]

Tricks for Trees: Spilling

- Overlapped partitioning reduces boundary errors
  - no backtracking necessary
- Spilling
  - Increases tree depth
    - more memory
    - slower to build
  - Better when split passes through sparse regions
  - Lower nodes may spill too much
    - hybrid of spill and non-spill nodes
  - Designing a good spill factor hard



Tricks for Trees

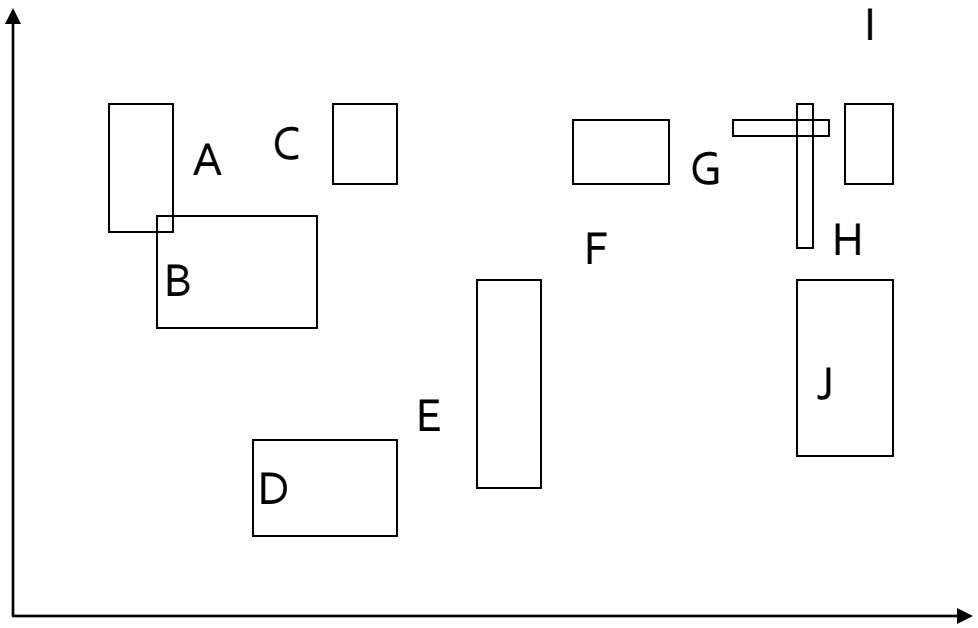
- For high dim. data, use randomized projections (CUR) or SVD
- Use Best-Bin-First (BBF)
  - Make a priority queue of all unexplored nodes
  - Visit them in order of their "closeness" to the query
    - Closeness is defined by distance to a cell boundary
- Space permitting:
  - Keep extra statistics on lower and upper bound for each cell and use triangle inequality to prune space
  - Use spilling to avoid backtracking
  - Use lookup tables for fast distance computation

R-trees (d=~20): Disk based

- "Bottom-up" approach [Guttman 84]
  - Start with a set of points/rectangles
  - Partition the set into groups of small cardinality
  - For each group, find minimum rectangle containing objects from this group (MBR)
  - Repeat
- Advantages:
  - Supports near(est) neighbor search (similar as before)
  - Works for points and rectangles
  - Avoids empty spaces

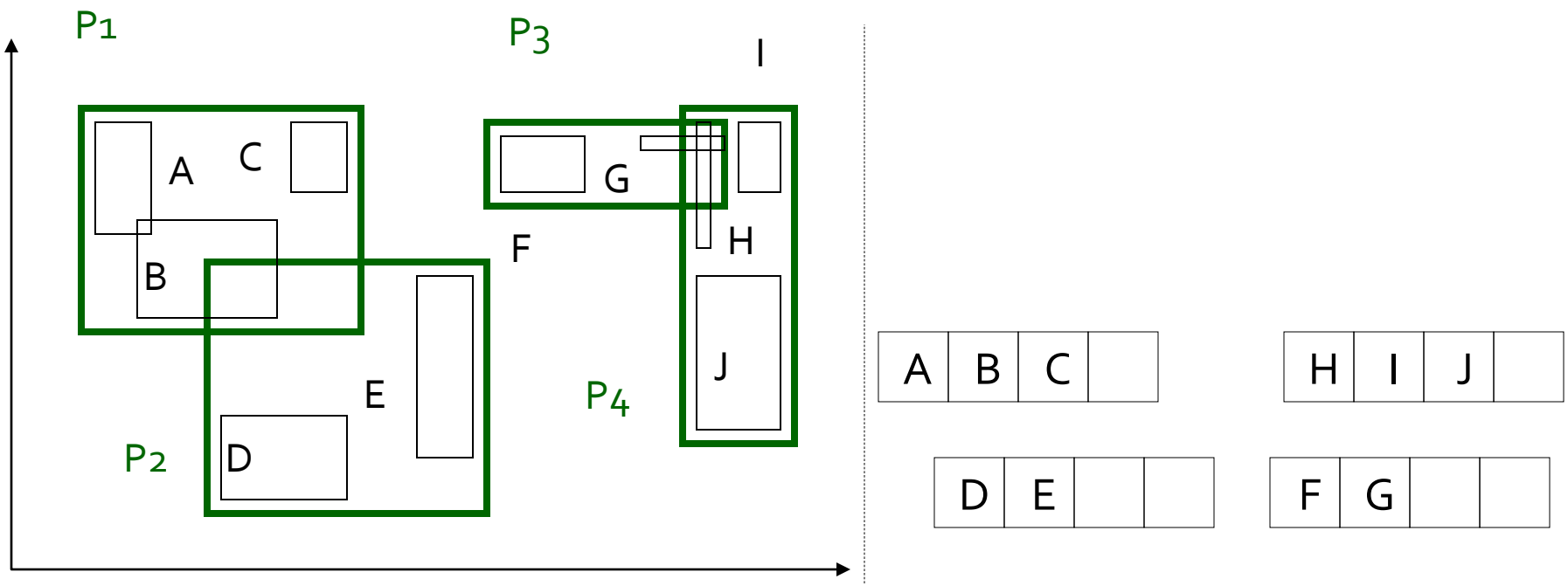
R-trees (1)

- R-trees with fan-out 4:
  - group nearby rectangles to parent MBRs



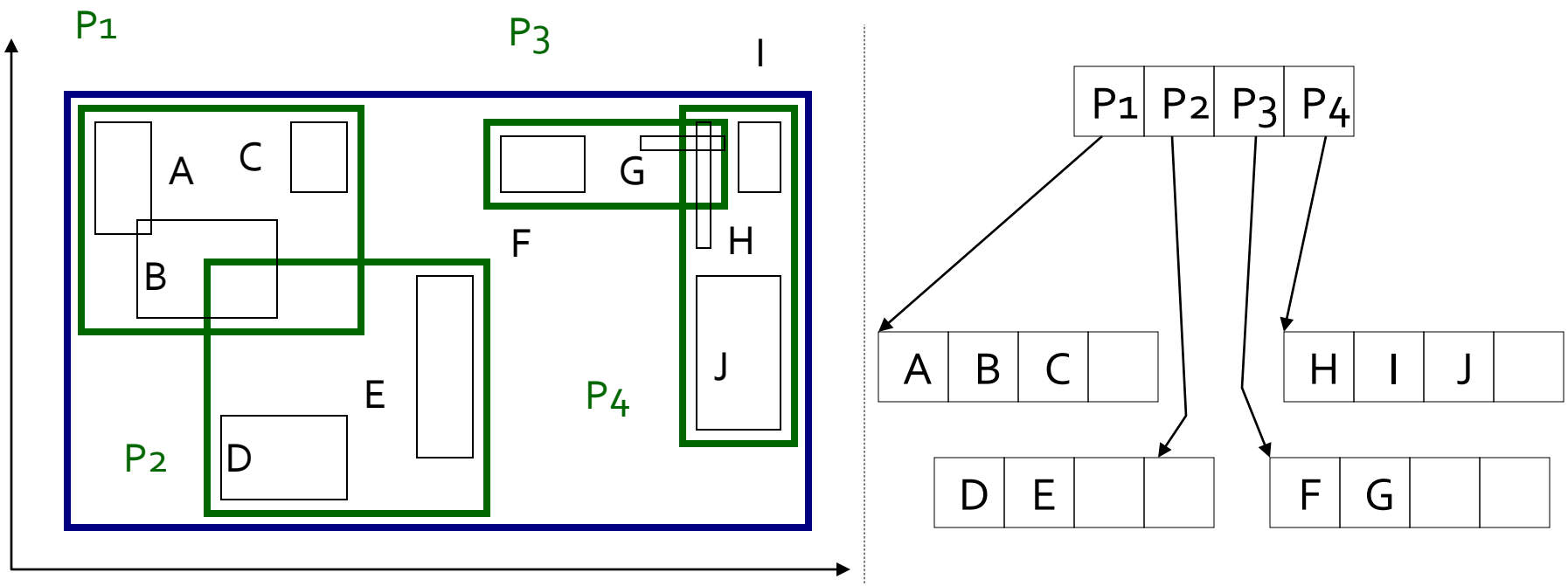
R-trees (2)

- R-trees with fan-out 4:
  - every parent node completely covers its 'children'



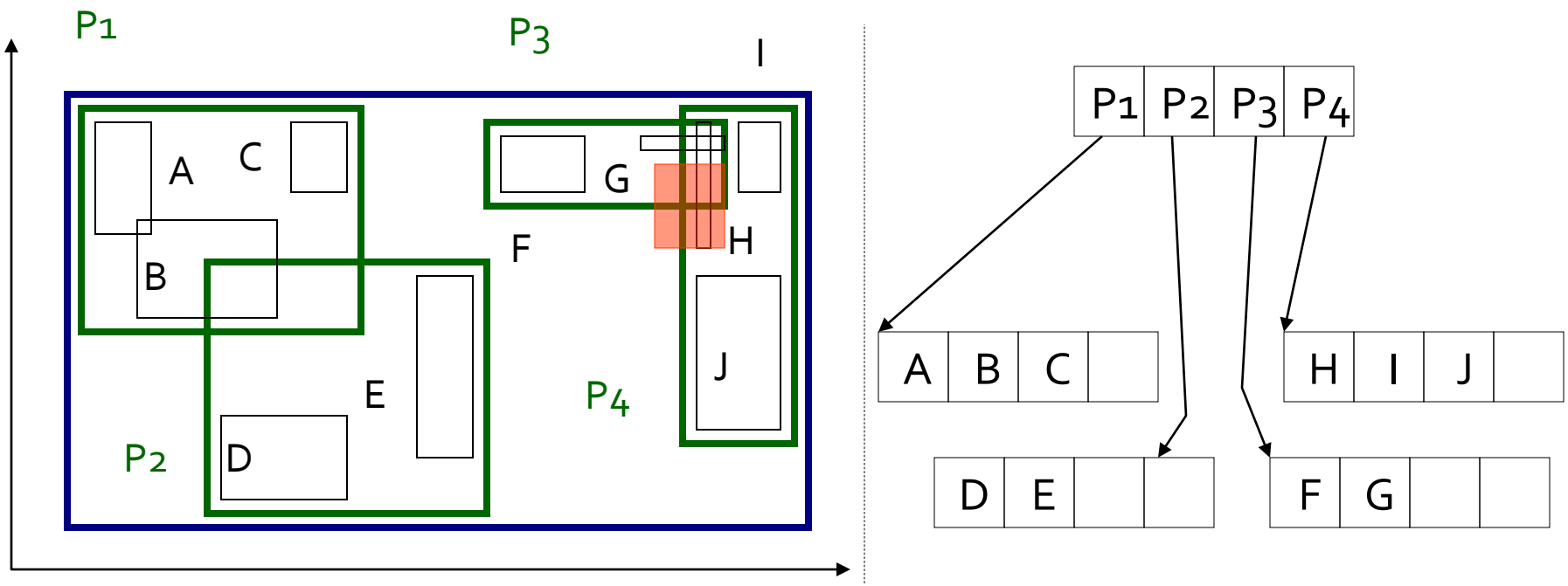
R-trees (3)

- R-trees with fan-out 4:
  - every parent node completely covers its 'children'



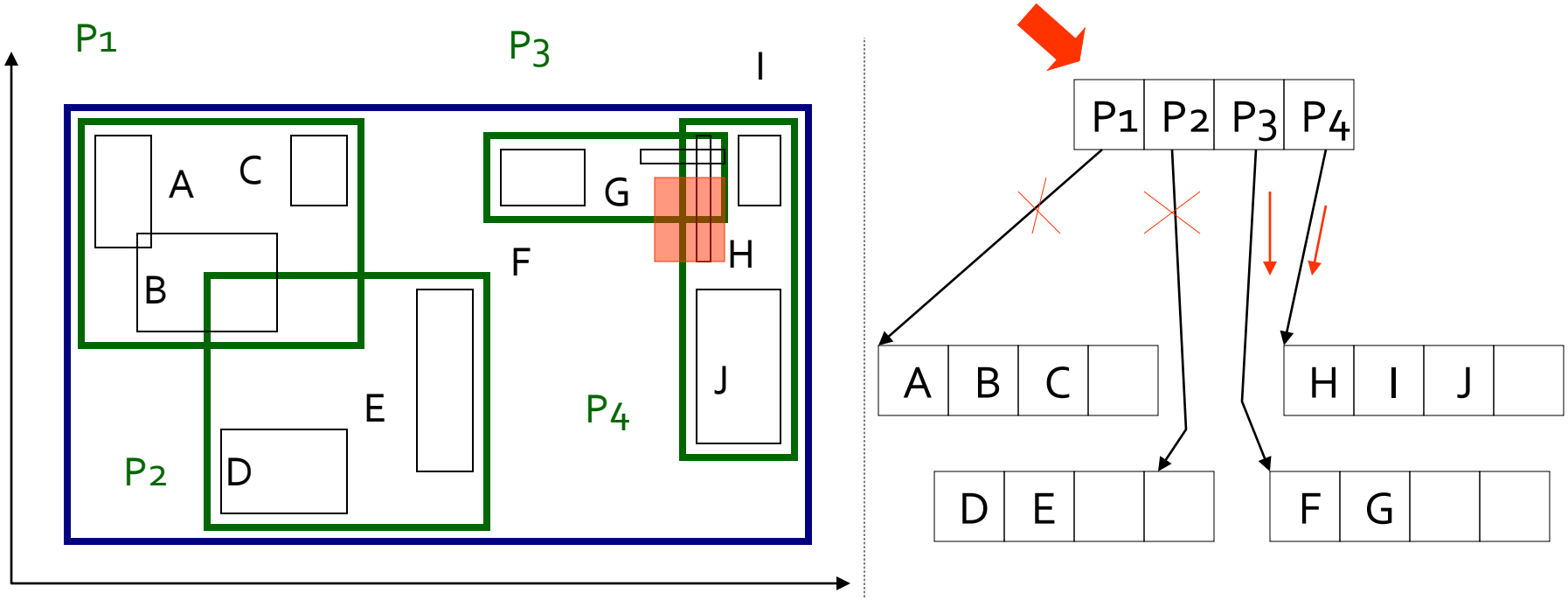
R-trees: Range search

- Example of a range search query



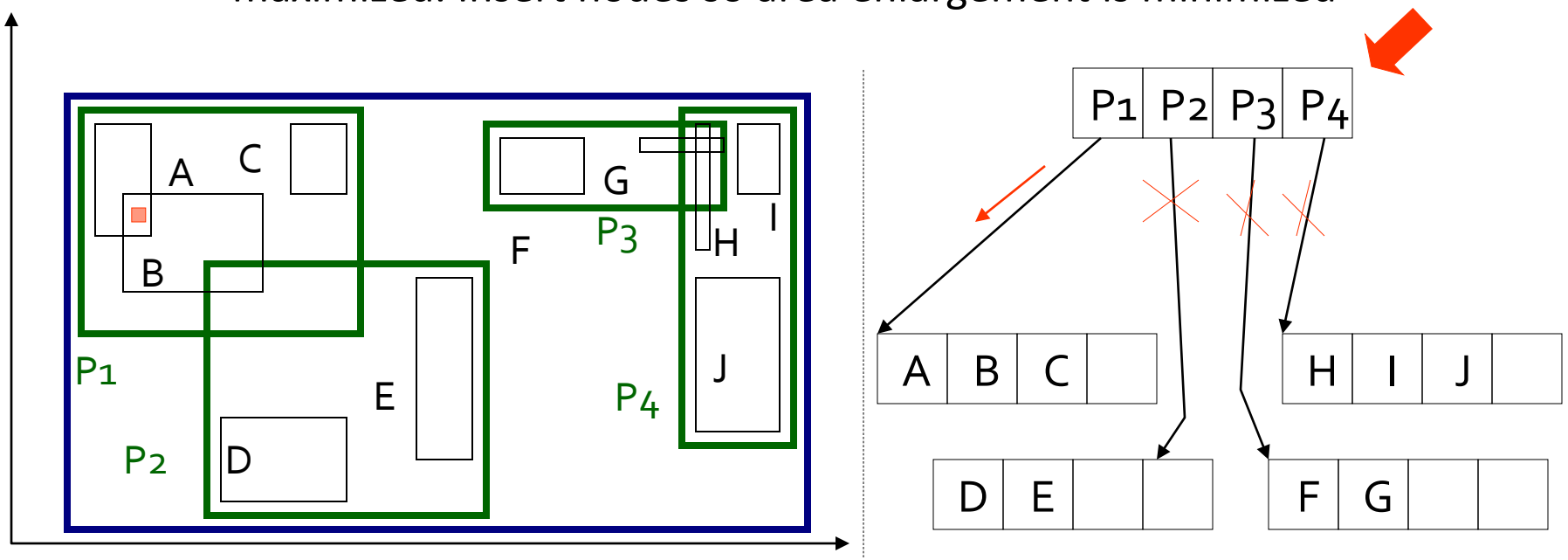
R-trees: Range search

- Example of a range search query



R-trees: Insertion

- Insertion of point *x*:
  - Find MBR intersecting with *x* and insert
  - If a node is full, then a split:
    - Linear – choose far apart nodes as ends. Randomly choose nodes and assign them so that they require the smallest MBR enlargement
    - Quadratic – choose two nodes so the dead space between them is maximized. Insert nodes so area enlargement is minimized



Linear scan: VA-file

- Approach [Weber, Schek, Blott'98]
  - In high-dimensional spaces, all tree-based indexing structures examine large fraction of leaves
  - If we need to visit so many nodes anyway, it is better to scan the whole data set and avoid performing seeks altogether
  - 1 seek = transfer of few hundred KB

VA-file

- Natural question:
  How to speed-up linear scan?

- Answer: Use approximation
  - Use only *i* bits per dimension (and speed-up the scan by a factor of 32/*i*)
  - Identify all points which <u>could</u> be returned as an answer
  - Verify the points using original data set

(1958)
F. Rosenblatt

The perceptron: a probabilistic model for information storage and organization in the brain

Psychological Review 65:386–408

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational mem-

Perceptron & Winnow

Linear models: Perceptron

- Example: Spam filtering

	viagra	learning	the	dating	nigeria	*spam*?
$\vec{x}_1 = ($	1	0	1	0	0)	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0)	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1)	$y_3 = 1$

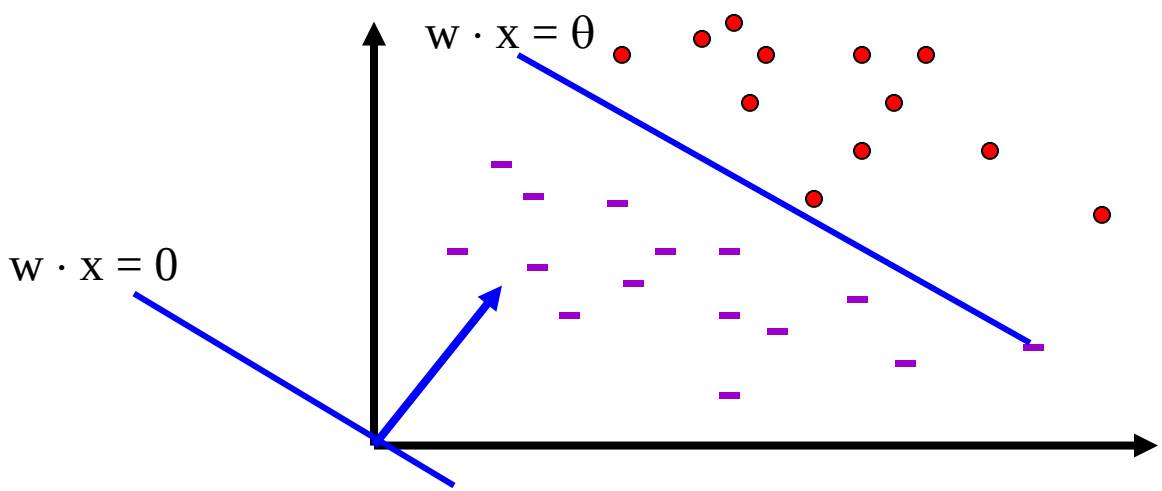
- Instance space X:
  - Binary feature vectors *x* of word occurrences
  - *d* features (words + other things, d~100,000)
- Class Y:
  - *y*: Spam (+1), Ham (-1)

Linear models for classification

- Binary classification:

$$f(x) = \begin{cases} 1 & \text{if} \quad w_1 x_1 + w_2 x_2 + \ldots w_n x_n \geq \theta \\ 0 & \text{otherwise} \end{cases}$$

- Input: Vectors x_i and labels y_i
- Goal: Find vector $w = (w_1, w_2, \ldots, w_n)$
  - Each w_i is a real number



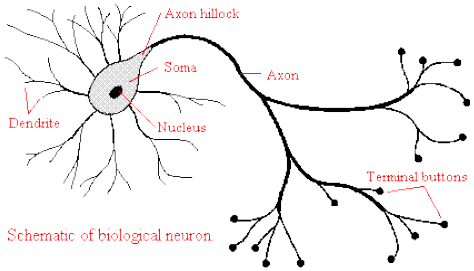
Note:

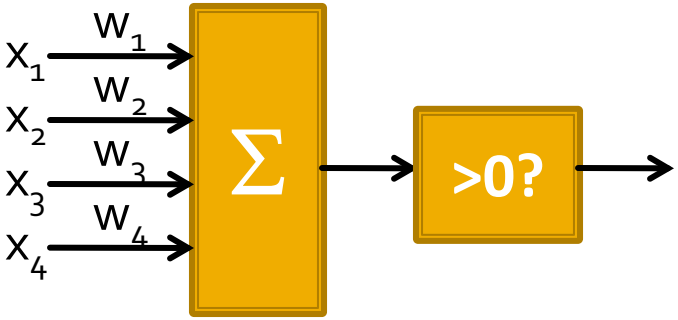
$$\mathbf{x} \Leftrightarrow \langle \mathbf{x}, 1 \rangle \quad \forall \mathbf{x}$$

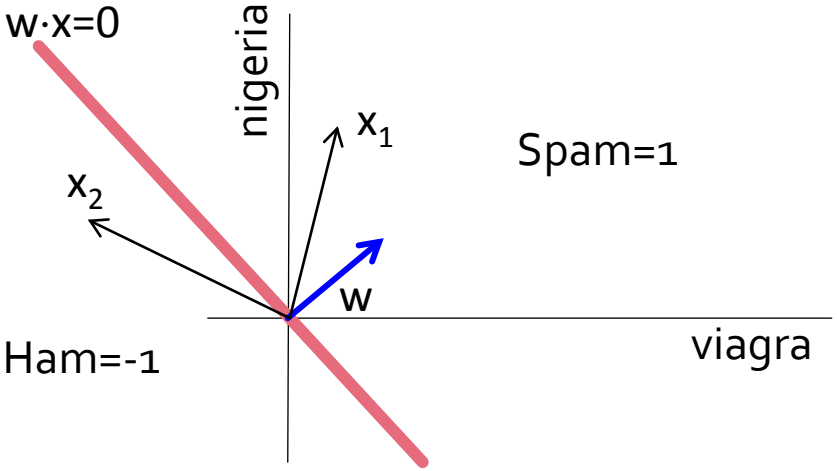
$$\mathbf{w} \Leftrightarrow \langle \mathbf{w}, -\theta \rangle$$

Perceptron [Rosenblatt '57]

- (very) Loose motivation: Neuron
- Inputs are feature values
- Each feature has a weight w_i
- Activation is the sum:
  - $f(x) = \Sigma_i w_i \cdot x_i = w \cdot x - \theta$
- If the f(x) is:
  - Positive: predict +1
  - Negative: predict -1



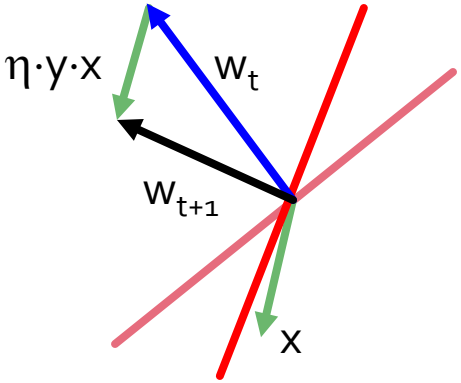




Perceptron: Estimating w

- Perceptron: $y' = \text{sign}(w \cdot x)$
- How to find parameters w?
  - Start with $w_0 = 0$
  - Pick training examples x one by one (from disk)
  - Predict class of x using current weights
    - $y' = \text{sign}(w_t \cdot x)$
  - If y' is correct (i.e., y=y'):
    - No change: $w_{t+1} = w_t$
  - If y' is wrong: adjust w

    $w_{t+1} = w_t + \eta \cdot y \cdot x$
    - where:
      - η is the learning rate parameter
      - x is the training example
      - y is true class label ({+1, -1})

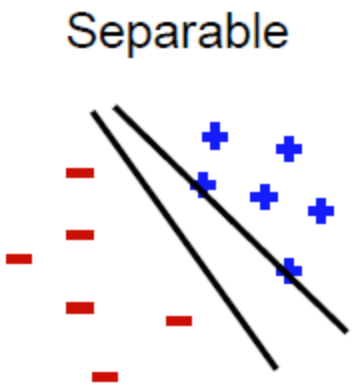


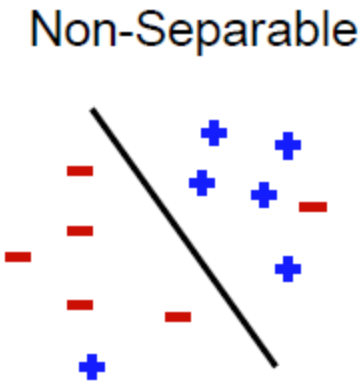
Perceptron convergence

- Perceptron Convergence Theorem:
  - If there exist a set of weights that are consistent (i.e., the data is linearly separable) the perceptron learning algorithm will converge
- How long would it take to converge?
- Perceptron Cycling Theorem:
  - If the training data is not linearly separable the perceptron learning algorithm will eventually repeat the same set of weights and therefore enter an infinite loop
- How to provide robustness, more expressivity?

Properties of Perceptron

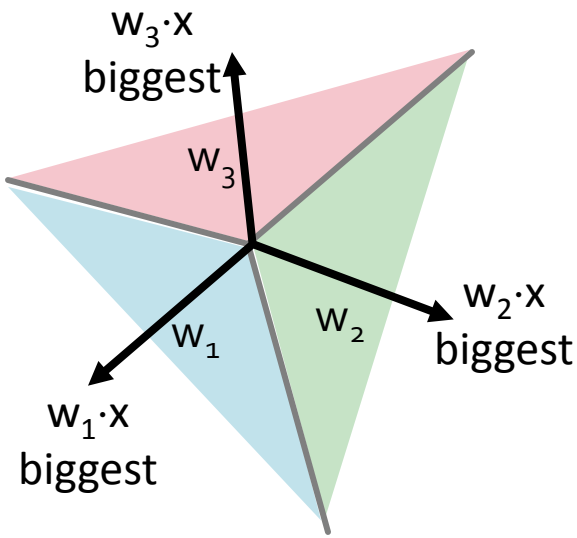
- Separability: some parameters get training set perfectly
- Convergence: if training set is separable, perceptron will converge (binary case)
- Mistake bound: number of mistakes < $1/\gamma^2$





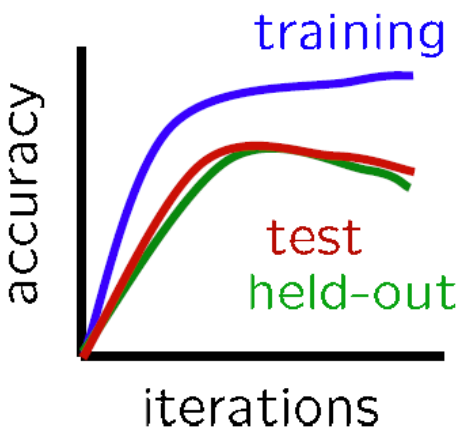
Multiclass Perceptron

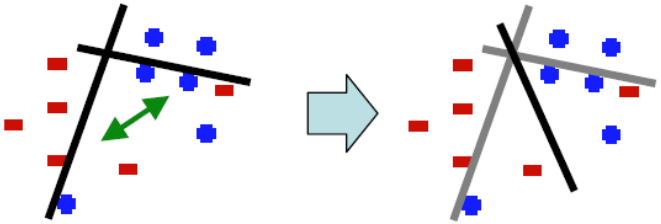
- If more than 2 classes:
  - Weight vector w_c for each class
  - Calculate activation for each class
    - $f(x,c) = \sum_i w_{c,i} \cdot x_i = w_c \cdot x$
  - Highest activation wins:
    - $c = \arg\max_c f(x,c)$

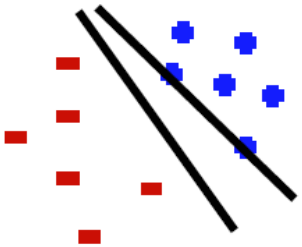


Issues with Perceptrons

- Overfitting:
- Regularization: if the data is not separable weights dance around
- Mediocre generalization:
  - Finds a "barely" separating solution







Winnow algorithm

- Winnow algorithm
  - Similar to perceptron, just different updates

Initialize : $\theta = n; \; w_i = 1$

Prediction is 1 iff $w \bullet x \geq \theta$

If no mistake : do nothing

If $f(x) = 1$ **but** $w \bullet x < \theta$ **,** $w_i \leftarrow 2w_i$ **(if** $x_i = 1$**) (promotion)**

If $f(x) = 0$ **but** $w \bullet x \geq \theta$ **,** $w_i \leftarrow w_i/2$ **(if** $x_i = 1$**) (demotion)**

  - Learns linear threshold functions

Extensions: Winnow

- Algorithm learns monotone functions
- For the general case:
  - Duplicate variables:
    - To negate variable x_i, introduce a new variable $x_i' = -x_i$
    - Learn monotone functions over 2n variables
  - Balanced version:
    - Keep two weights for each variable; effective weight is the difference

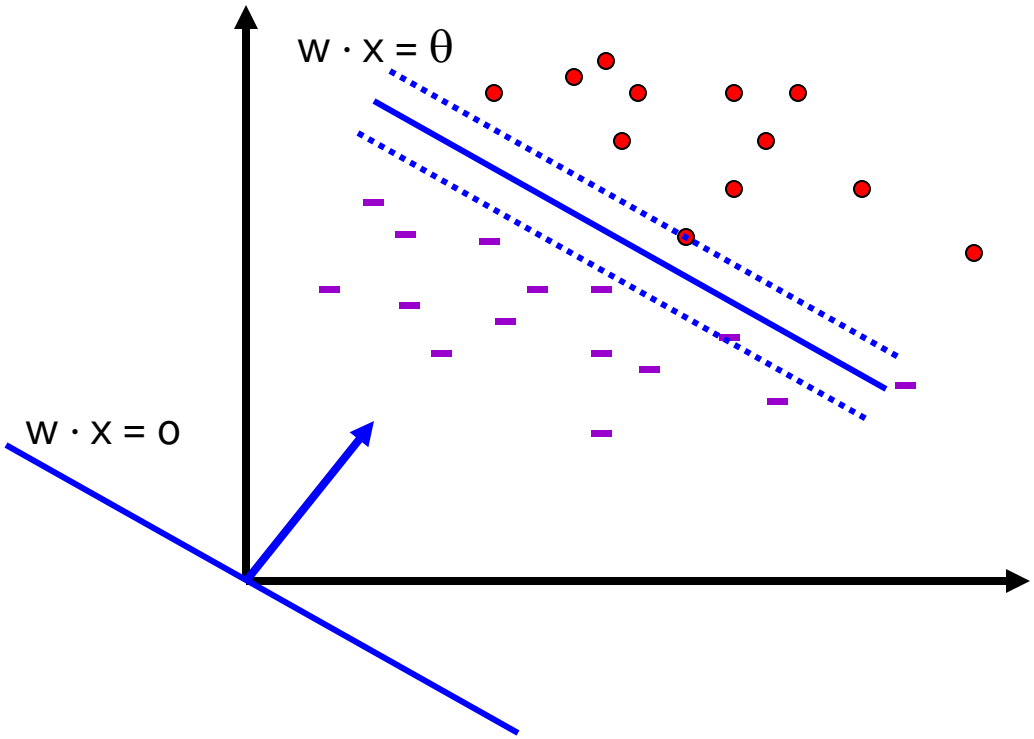
Update Rule :

If $f(x) = 1$ but $(w^+ - w^-) \bullet x \le \theta$, $\quad w_i^+ \leftarrow 2w_i^+ \quad w_i^- \leftarrow \frac{1}{2} w_i^-$ where $x_i = 1$ (promotion)

If $f(x) = 0$ but $(w^+ - w^-) \bullet x \ge \theta$, $\quad w_i^+ \leftarrow \frac{1}{2} w_i^+ \quad w_i^- \leftarrow 2w_i^-$ where $x_i = 1$ (demotion)

Extensions: Thick Separator

- Thick Separator (aka Perceptron with Margin)
  (Applies both for Perceptron and Winnow)
  - Promote if:
    - $w \cdot x > \theta + \gamma$
  - Demote if:
    - $w \cdot x < \theta - \gamma$



Note: γ is a functional margin. Its effect could disappear as *w* grows. Nevertheless, this has been shown to be a very effective algorithmic addition.

Summary of Algorithms

Examples : $x \in \{0,1\}^n$; **Hypothesis : $w \in R^n$**

Prediction is 1 iff $w \bullet x \geq \theta$

- Additive weight update algorithm [Perceptron, Rosenblatt, 1958]

$w \leftarrow w + \eta_i y_j x_j$

If Class = 1 but $w \bullet x \leq \theta$, $w_i \leftarrow w_i + 1$ (if $x_i = 1$) (promotion)

If Class = 0 but $w \bullet x \geq \theta$, $w_i \leftarrow w_i - 1$ (if $x_i = 1$) (demotion)

- Multiplicative weight update algorithm [Winnow, Littlestone, 1988]

$w \leftarrow w\, \eta_i \exp\{y_j x_j\}$

If Class = 1 but $w \bullet x \leq \theta$, $w_i \leftarrow 2w_i$ (if $x_i = 1$) (promotion)

If Class = 0 but $w \bullet x \geq \theta$, $w_i \leftarrow w_i/2$ (if $x_i = 1$) (demotion)

Perceptron vs. Winnow

- **Perceptron**
- Online: can adjust to changing target, over time
- Advantages
  - Simple
  - Guaranteed to learn a linearly separable problem
- Limitations
  - only linear separations
  - only converges for linearly separable data
  - not really "efficient with many features"

- **Winnow**
- Online: can adjust to changing target, over time
- Advantages
  - Simple
  - Guaranteed to learn a linearly separable problem
  - **Suitable for problems with many irrelevant attributes**
- Limitations
  - only linear separations
  - only converges for linearly separable data
  - not really "efficient with many features"