Large Scale Machine Learning: k-NN, Perceptron & SVM

CS345a: Data Mining
Jure Leskovec and Anand Rajaraman
Stanford University

Supervised Machine Learning

- Would like to do prediction:
  learn a function: y = f(x)
- Where y can be:
  - Real: Regression
  - Categorical: Classification
  - More complex:
    - Ranking, Structured prediction, etc.
- Data is labeled:
  - Have many pairs (x,y)

Large Scale Machine Learning

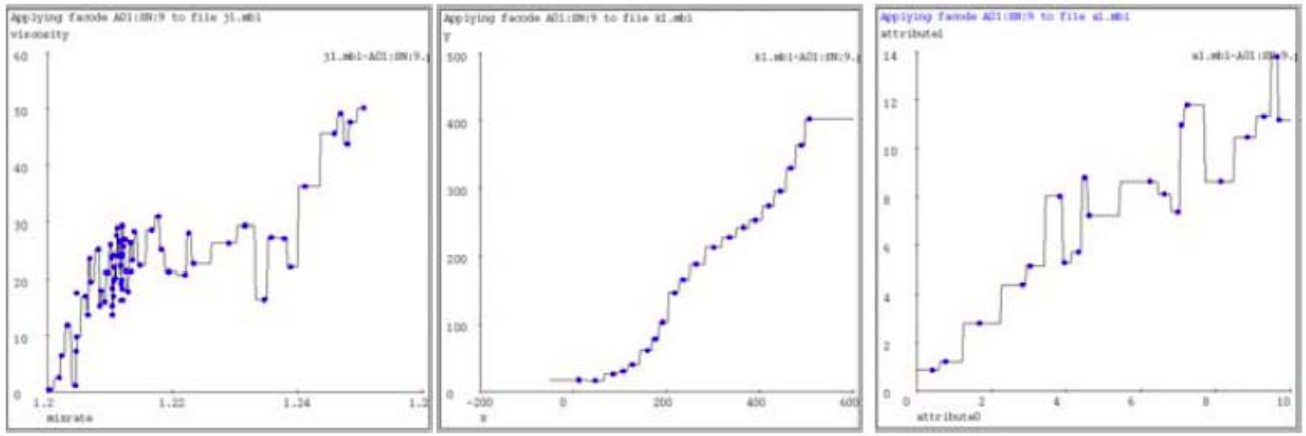
- We will talk about the following methods:
  - k-Nearest Neighbor (Instance based learning)
  - Perceptron algorithm
  - Support Vector Machines
  - Decision trees (lecture on Thursday by Sugato Basu from Google)
- **How to efficiently train (build a model)?**

Instance Based Learning

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

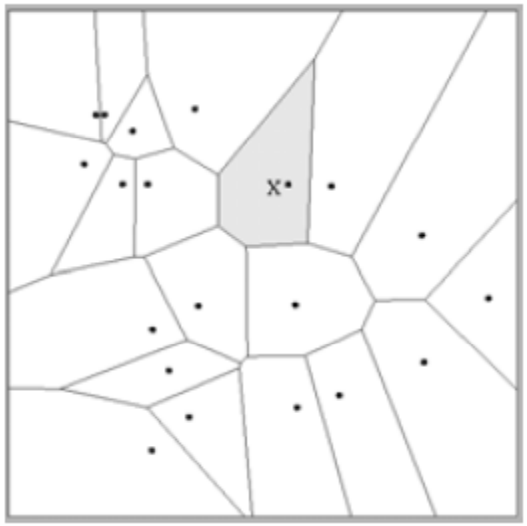
1-Nearest Neighbor

- To make things 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

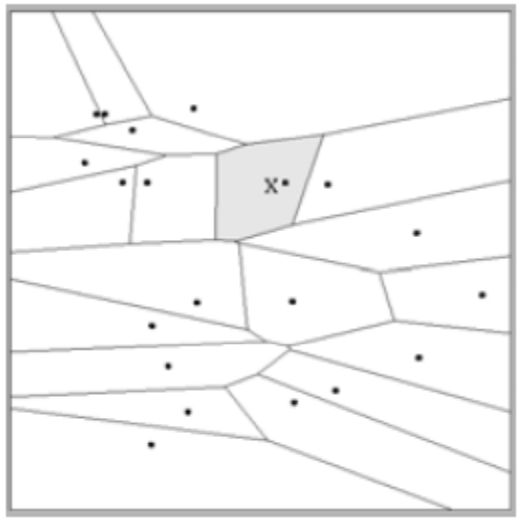


Distance metrics

- 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:



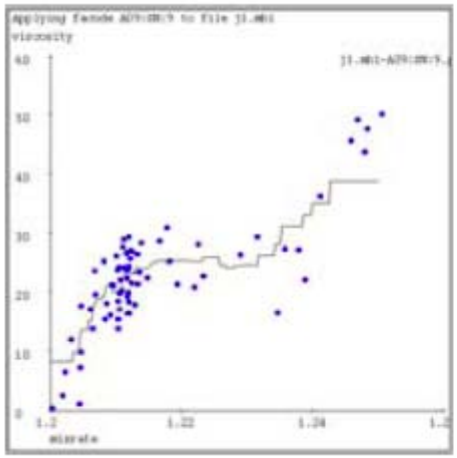
$d(xi,xj)=(x_{i1}-x_{j1})^2+(x_{i2}-x_{j2})^2$

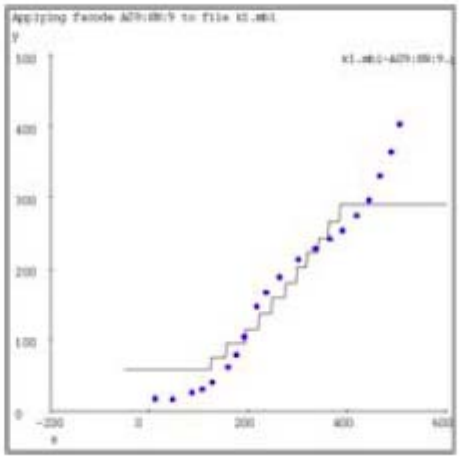


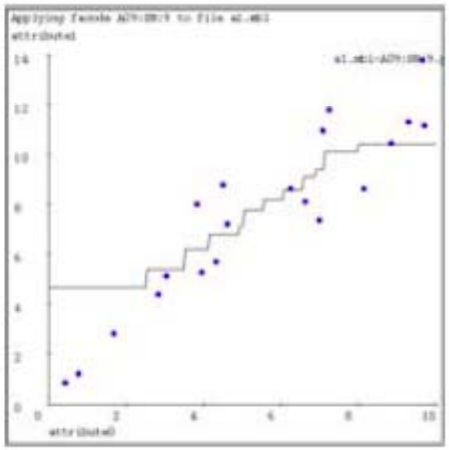
$d(xi,xj)=(x_{i1}-x_{j1})^2+(3x_{i2}-3x_{j2})^2$

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



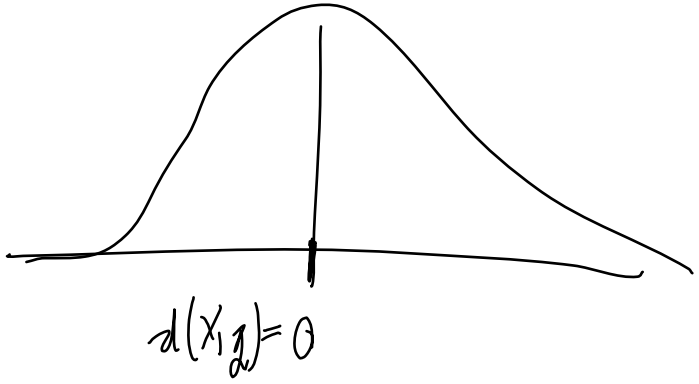


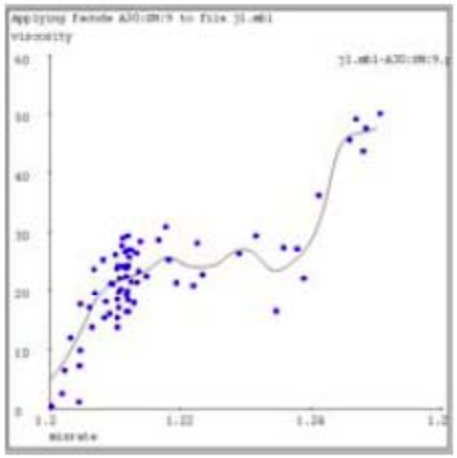


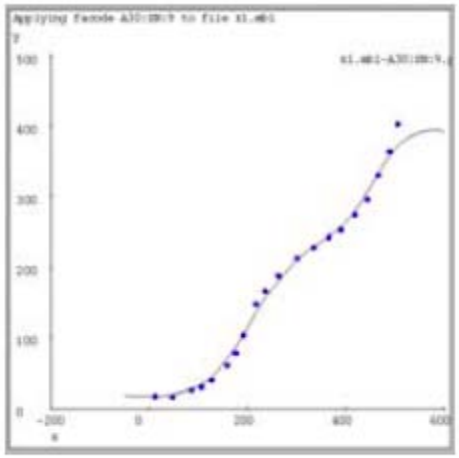
k=9

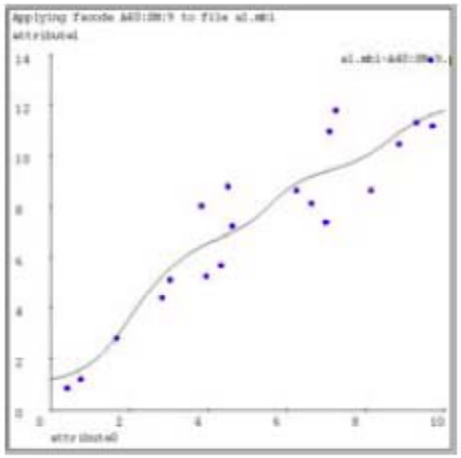
Kernel regression

- 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: $\Sigma w_i y_i / \Sigma w_i$



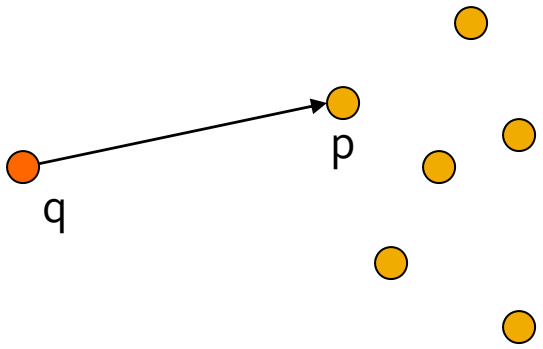






How to find nearest neighbors?

- 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

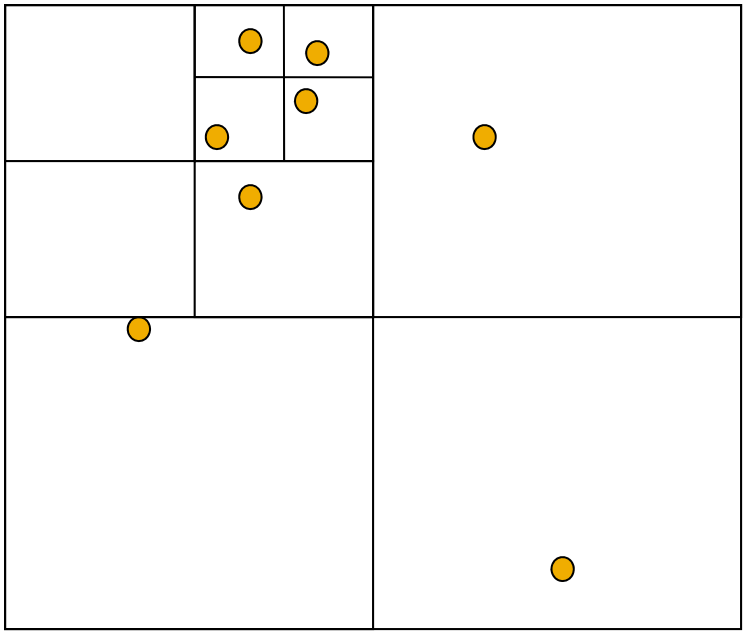


Algorithms for NN

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

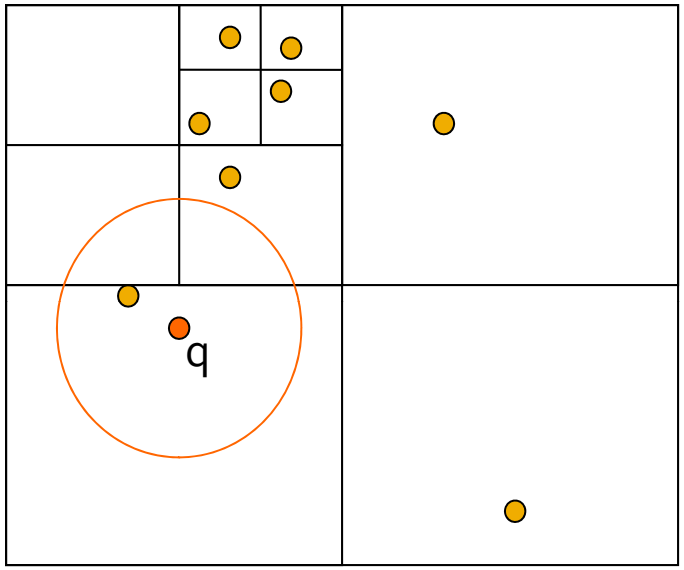
Quadtree (d=~3)

- 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)

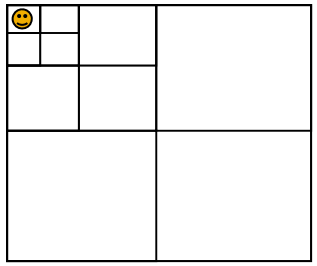


Quadtree: Search

- 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*

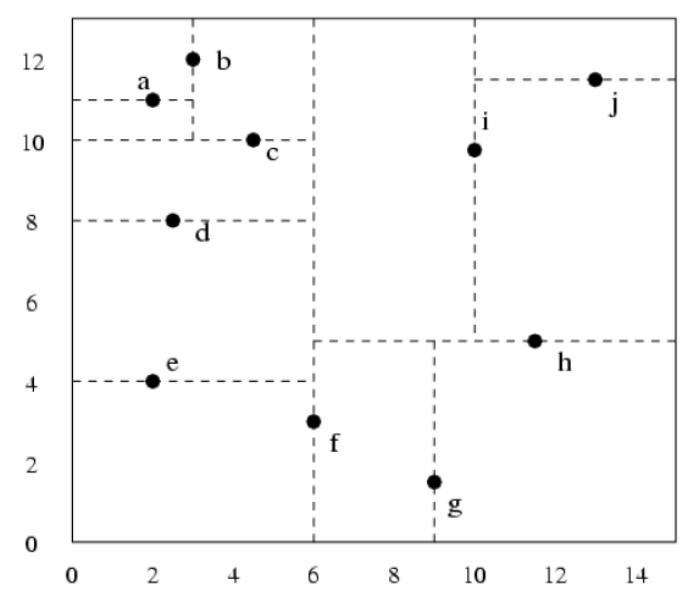


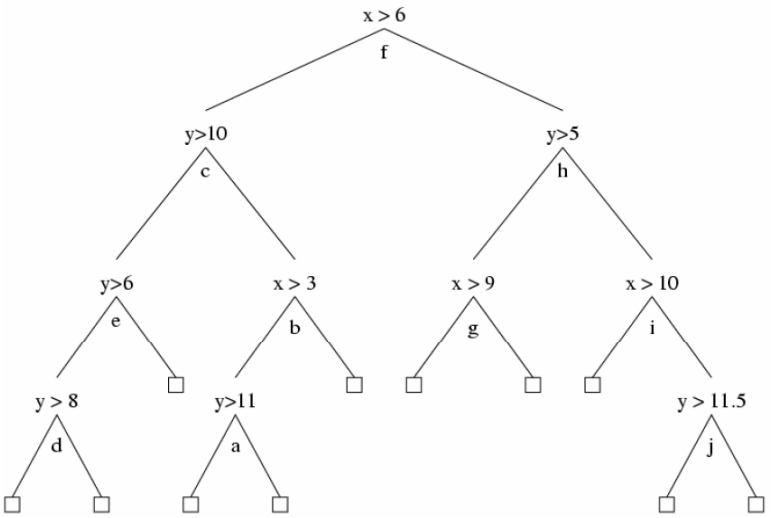
- Great in 2 or 3 dimensions
- Nodes have 2^d parents
- Space issues:



Kd-tree (d=~10)

- Main ideas [Bentley '75] :
  - Only one-dimensional splits
  - Choose the split "carefully" (many variations)
  - Queries: as for quadtrees
- Advantages:
  - no (or less) empty spaces
  - only linear space
- Query time at most:
  - Min[*dn*, exponential(*d*)]



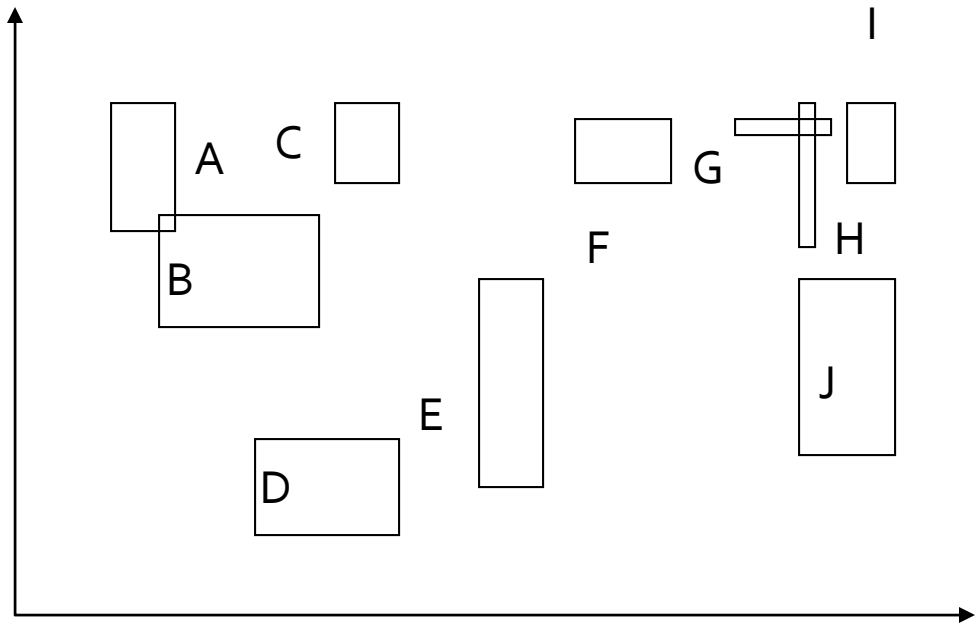


R-trees (d=~20)

- "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

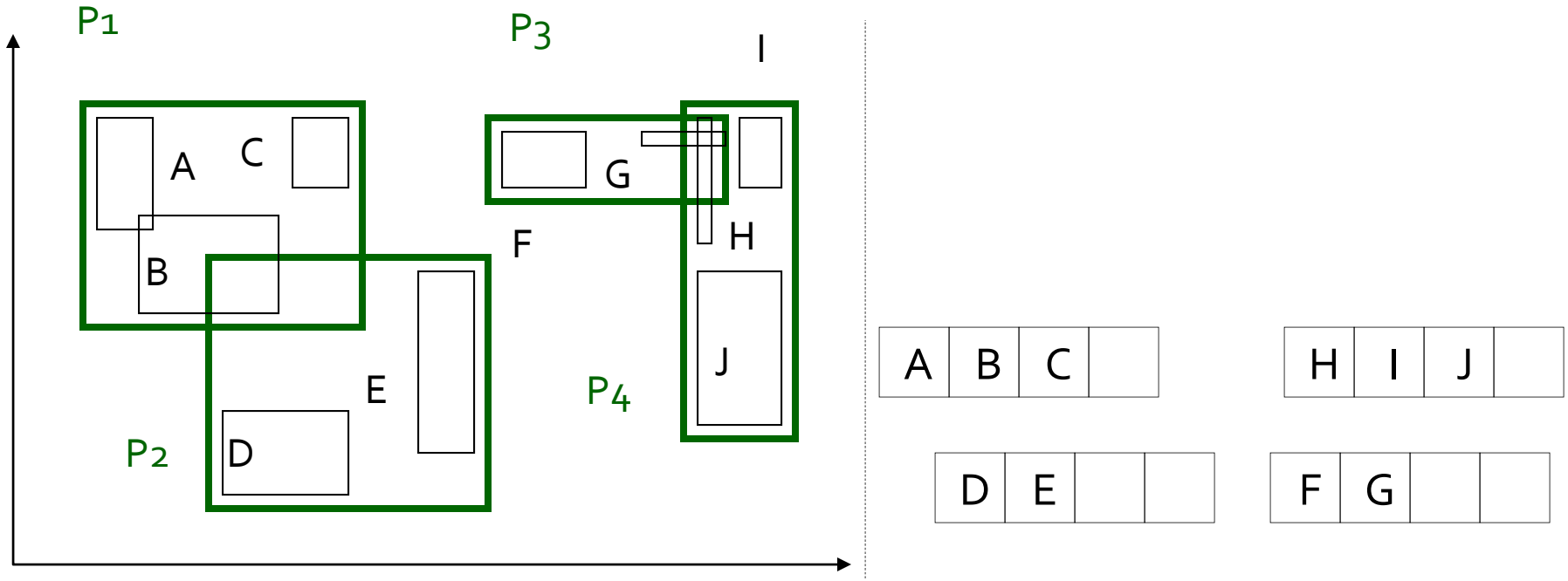
R-trees (1)

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



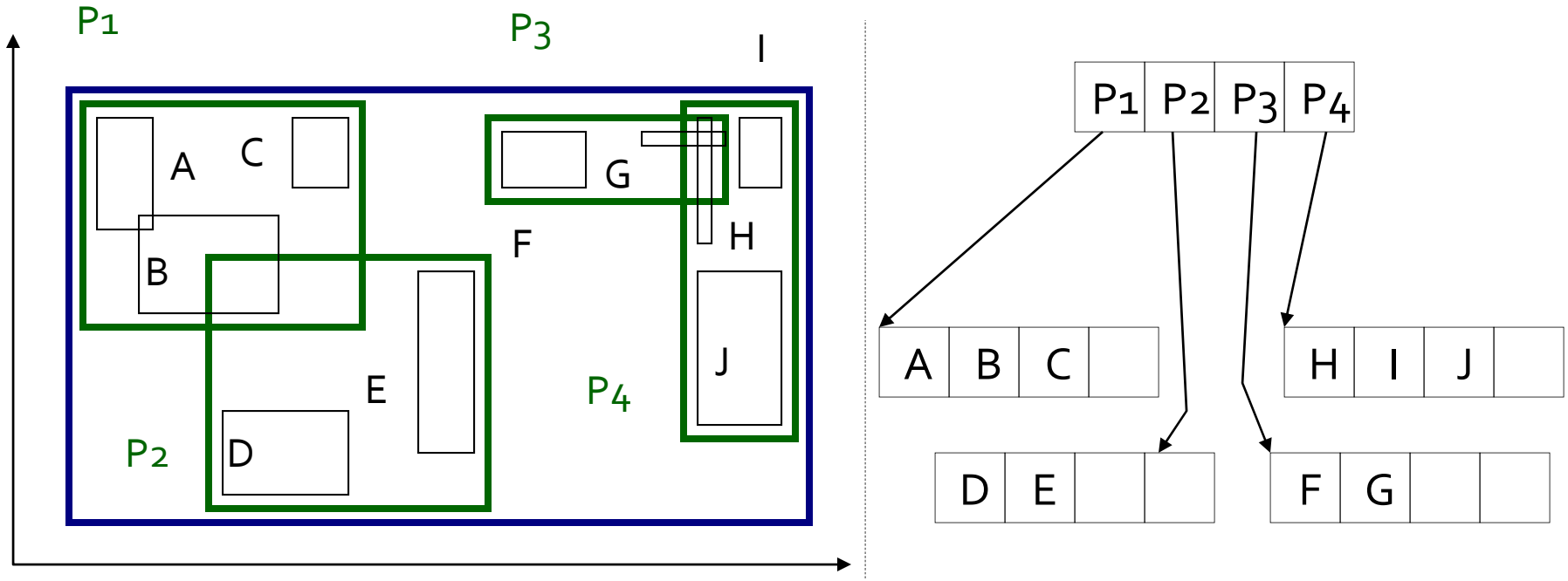
R-trees (2)

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



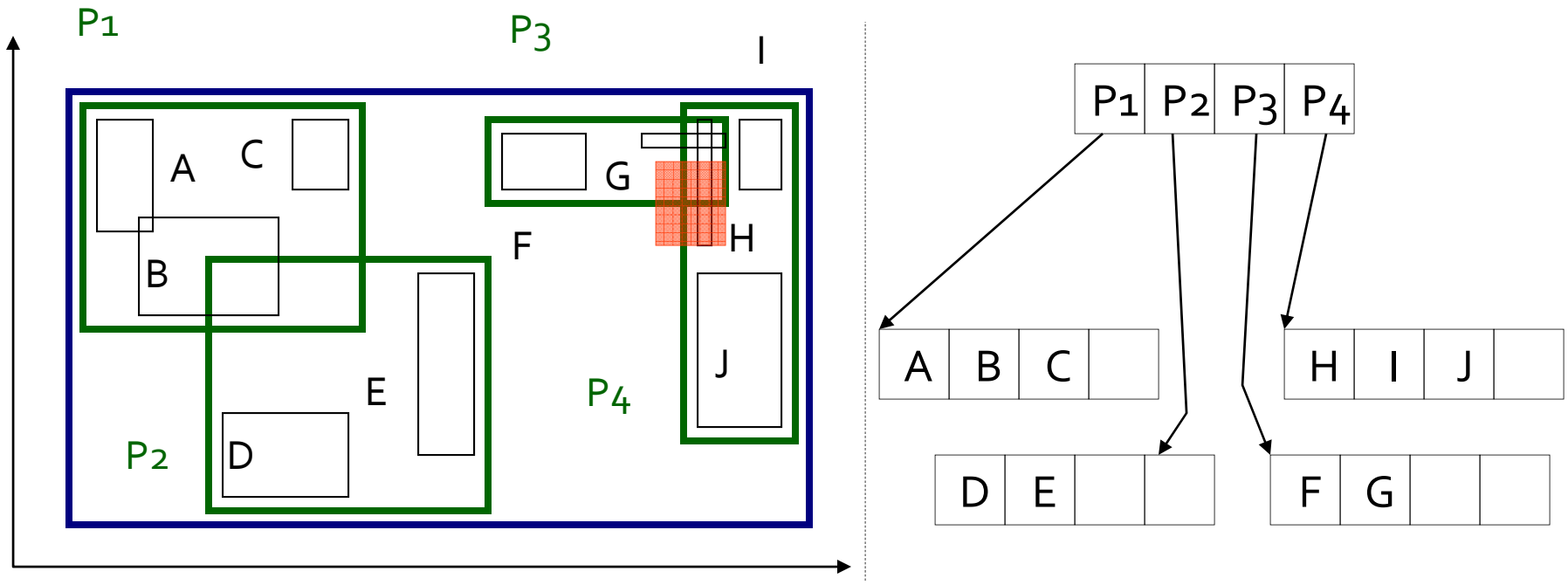
R-trees (3)

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



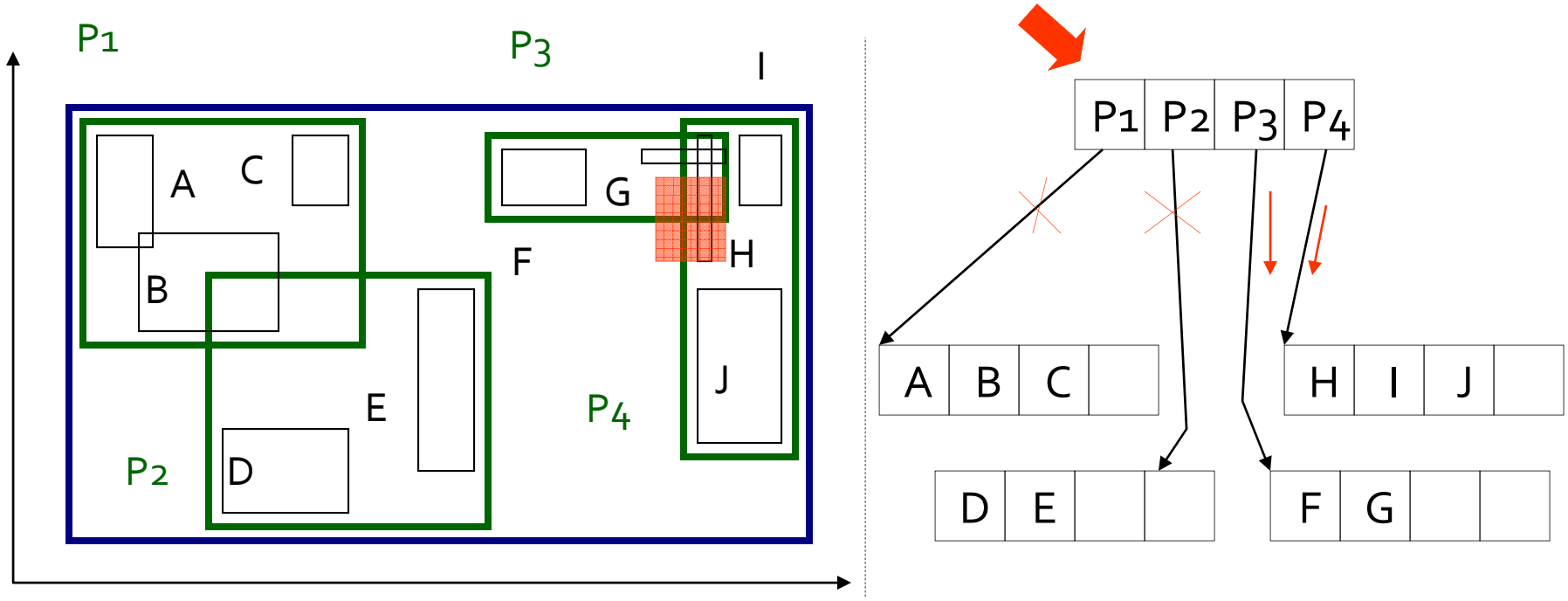
R-trees: Range search

- Example of a range search query



R-trees: Range search

- Example of a range search query



Linear models: Perceptron

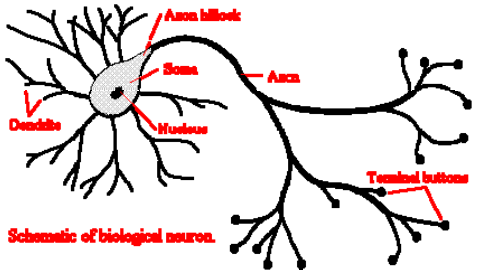
- Example: Spam filtering

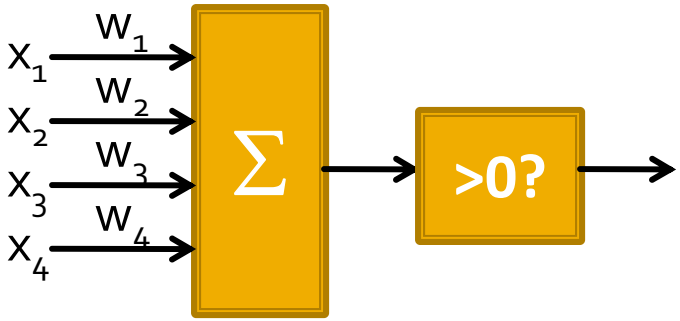
	viagra	learning	the	dating	nigeria	*spam?*
$\vec{x}_1 = ($	1	0	1	0	0 $)$	$y_1 = 1$
$\vec{x}_2 = ($	0	1	1	0	0 $)$	$y_2 = -1$
$\vec{x}_3 = ($	0	0	0	0	1 $)$	$y_3 = 1$

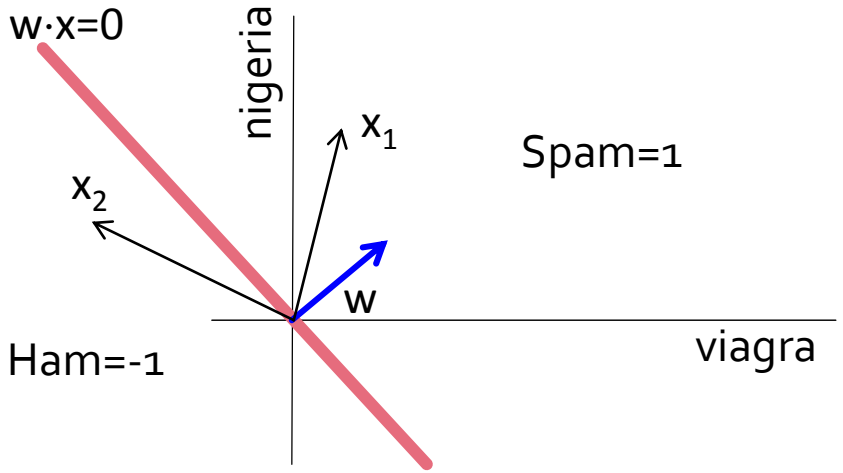
- Instance space X:
  - Feature vector of word occurrences (binary, TF-IDF)
  - d features (d~100,000)
- Class Y:
  - Spam (+1), Ham (-1)

Perceptron [Rosenblatt '57]

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

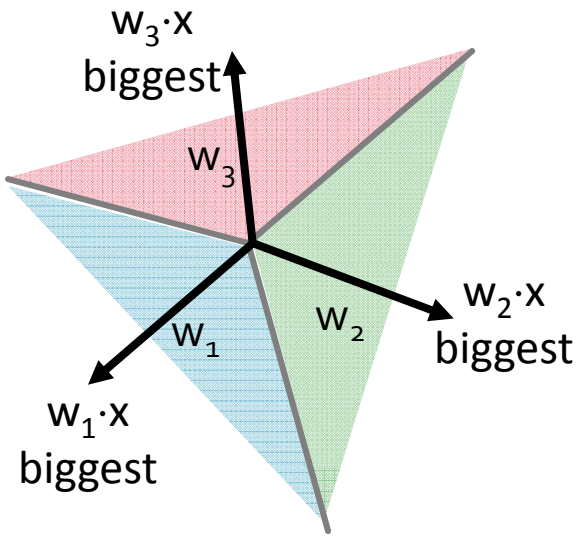






Multiclass Perceptron

- 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:
    - $c = \arg\max_c f(x,c)$



Learning the model

- Define a model:

  Perceptron: $y = \text{sign}(w \cdot x)$

- Define a loss function:

  $L(w) = -\sum_i y_i \cdot w \cdot x_i$

- Minimize the loss:
  - Compute gradient L'(w) and optimize:

    $w_{t+1} = w_t - \eta_t \cdot L'(w) = w_t - \lambda_t \cdot \sum_i^m dL(y_i \cdot w \cdot x_i)/dw$

    (Batch gradient descent)

Stochastic Gradient Descent

- Stochastic gradient descent:
  - Examples are drawn from a finite training set
  - Pick random example x_j and update

$$w_{t+1} = w_t - \eta_t \cdot dL(w \cdot x_j, y_j)/dw$$

$O(1 \cdot d)$

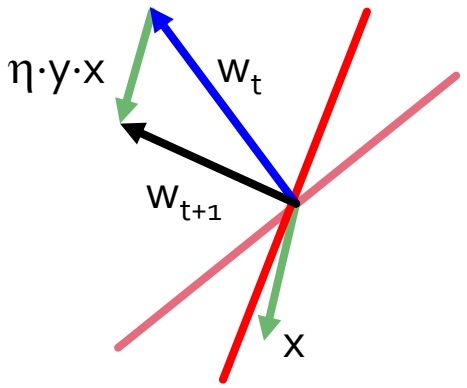
	Cost per iteration	Time to reach accuracy ρ	Time for optimization error $<\varepsilon$
GD	$O(m \cdot d)$	$O(m \cdot \kappa \cdot d \cdot \log(1/\rho))$	$O(\kappa \cdot d^2/\varepsilon \cdot \log^2(1/\varepsilon))$
2nd order GD	$O(d(d+m))$	$O(m \cdot d \cdot \log\log(1/\rho))$	$O(d^2/\varepsilon \cdot \log(1/\varepsilon) \cdot \log\log(1/\varepsilon))$
Stochastic GD	$O(d)$	$O(\kappa \cdot d/\rho)$	$O(\kappa \cdot d/\varepsilon)$

[Bottou-LeCun '04]

m… number of examples
d… number of features
κ… condition number

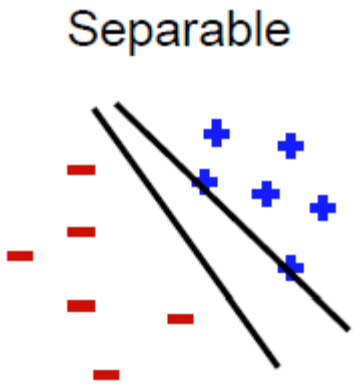
Perceptron: Estimating w

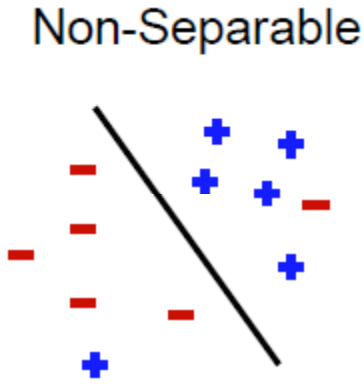
- Start with w=0
- Pick training examples x one by one
- Predict class of x using current weights
  - y' = sign(w·x)
- If y' is correct:
  - no change
- If y' is wrong: adjust w
  - $w_{t+1} = w_t + \eta \cdot y \cdot x$
    - η is the learning rate parameter
    - x is the training example
    - y is true class label



Properties of Perceptron

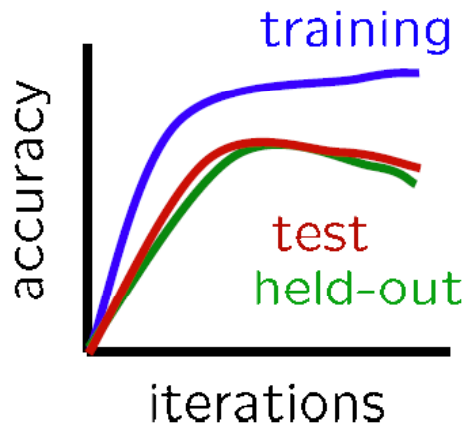
- Separability: some parameters get training set perfectly
- Convergence: if training set is separable, perceptron will converge (binary case)
- Mistake bound: number of mistakes (binary case) related to the margin or degree of separability γ:
  - mistakes < $1/\gamma^2$

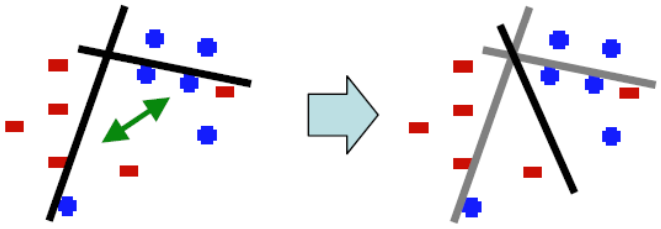


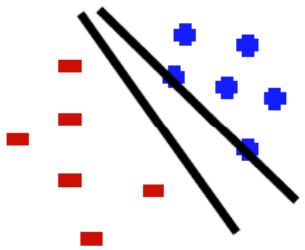


Issues with Perceptrons

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

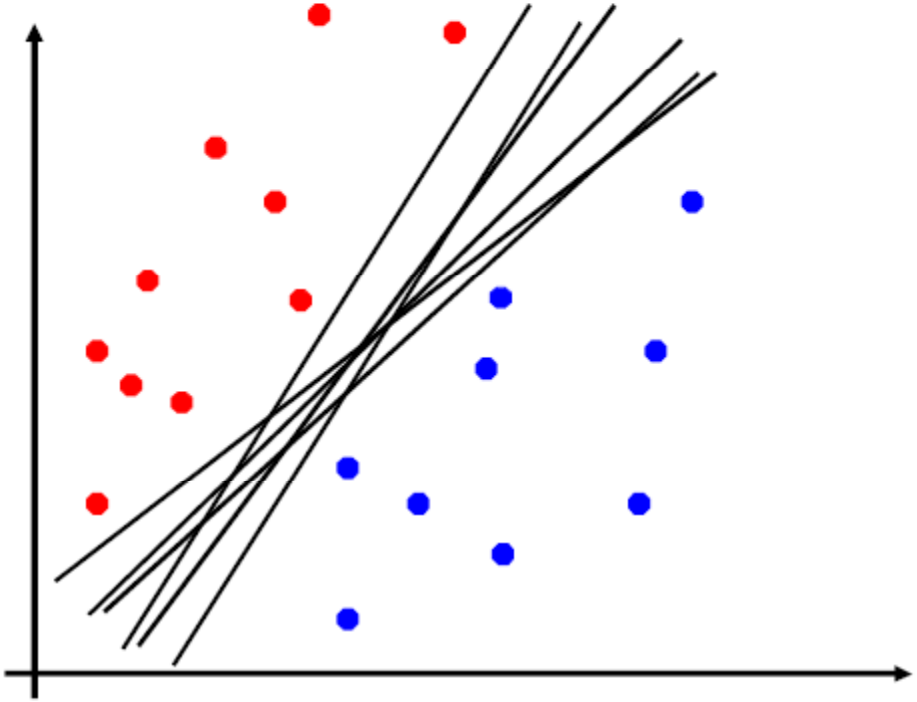






Support Vector Machines

- Which is best linear separator?

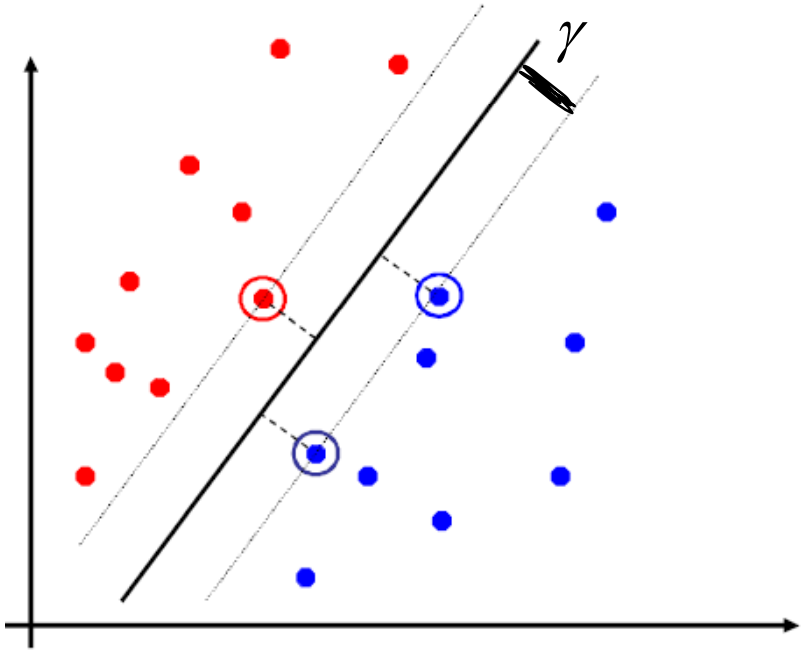


Support Vector Machine

- Maximize the margin:
  - Good according to intuition, theory & practice

$$\max_w \gamma$$

$$s.t. \forall i, y_i \cdot x_i \cdot w \geq \gamma$$



  - Since:

$$\gamma = \frac{1}{\sqrt{w \cdot w}}$$

$$\min_w \| w \|^2$$

$$s.t. \forall i, y_i \cdot x_i \cdot w \geq 1$$

SVM with "hard" constraints

Support Vector Machines

- If not separable introduce slack variables ξ:

$$\underset{\mathbf{w},\xi_i \geq 0}{\text{argmin}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$

$$\text{s.t.} \quad \forall i, \ y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \geq 1 - \xi_i$$

- Or in the "natural" form:

$$\underset{\mathbf{w}}{\text{argmin}}\ f(\mathbf{w}) \qquad \text{where:}$$

$$f(\mathbf{w}) \overset{\text{def}}{=} \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{Regularization term}} + \underbrace{\frac{1}{m}\sum_{i=1}^{m}\max\{0, 1 - y_i\langle \mathbf{w}, \mathbf{x}_i\rangle\}}_{\text{Empirical loss}}$$

SVM: How to estimate w

- Use quadratic solver:
  - Minimize quadratic function
  - Subject to linear constraints

$$\underset{\mathbf{w},\xi_i \geq 0}{\text{argmin}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i$$
$$\text{s.t.} \quad \forall i,\ y_i\langle \mathbf{w}, \mathbf{x}_i\rangle \geq 1 - \xi_i$$

- Stochastic gradient descent:
  - Minimize:

$$f(\mathbf{w}) \overset{\text{def}}{=} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{m}\sum_{i=1}^{m}\max\{0, 1 - y_i\langle \mathbf{w}, \mathbf{x}_i\rangle\}$$

  - Update:

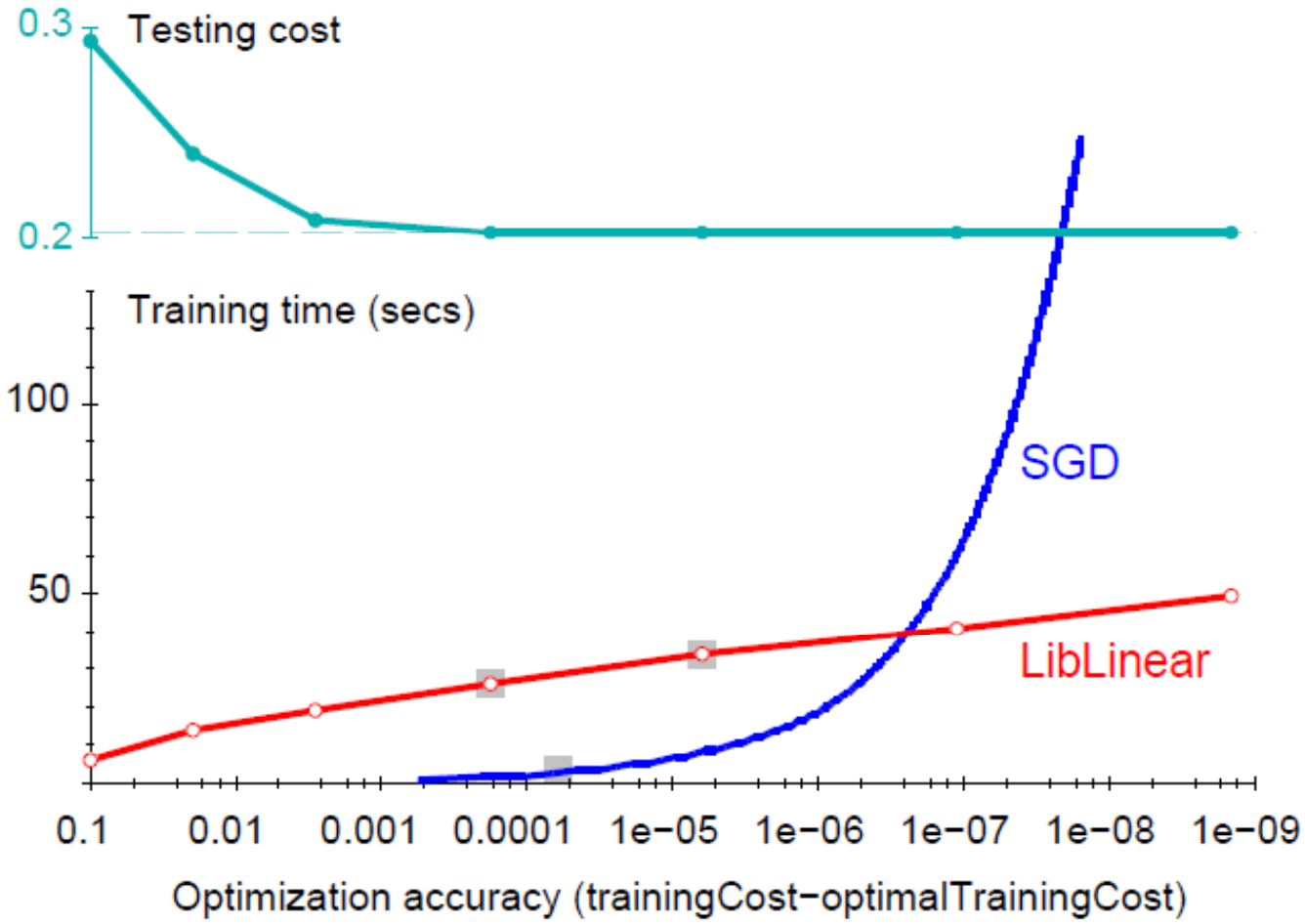
$$w \leftarrow w - \eta_t f'(w) = w - \eta_t\left(\lambda w + \frac{\partial L(wx_t, y_t)}{\partial w}\right)$$

Example: Text categorization

- Example by Leon Bottou:
  - Reuters RCV1 document corpus
  - m=781k training examples, 23k test examples
  - d=50k features
- Training time:

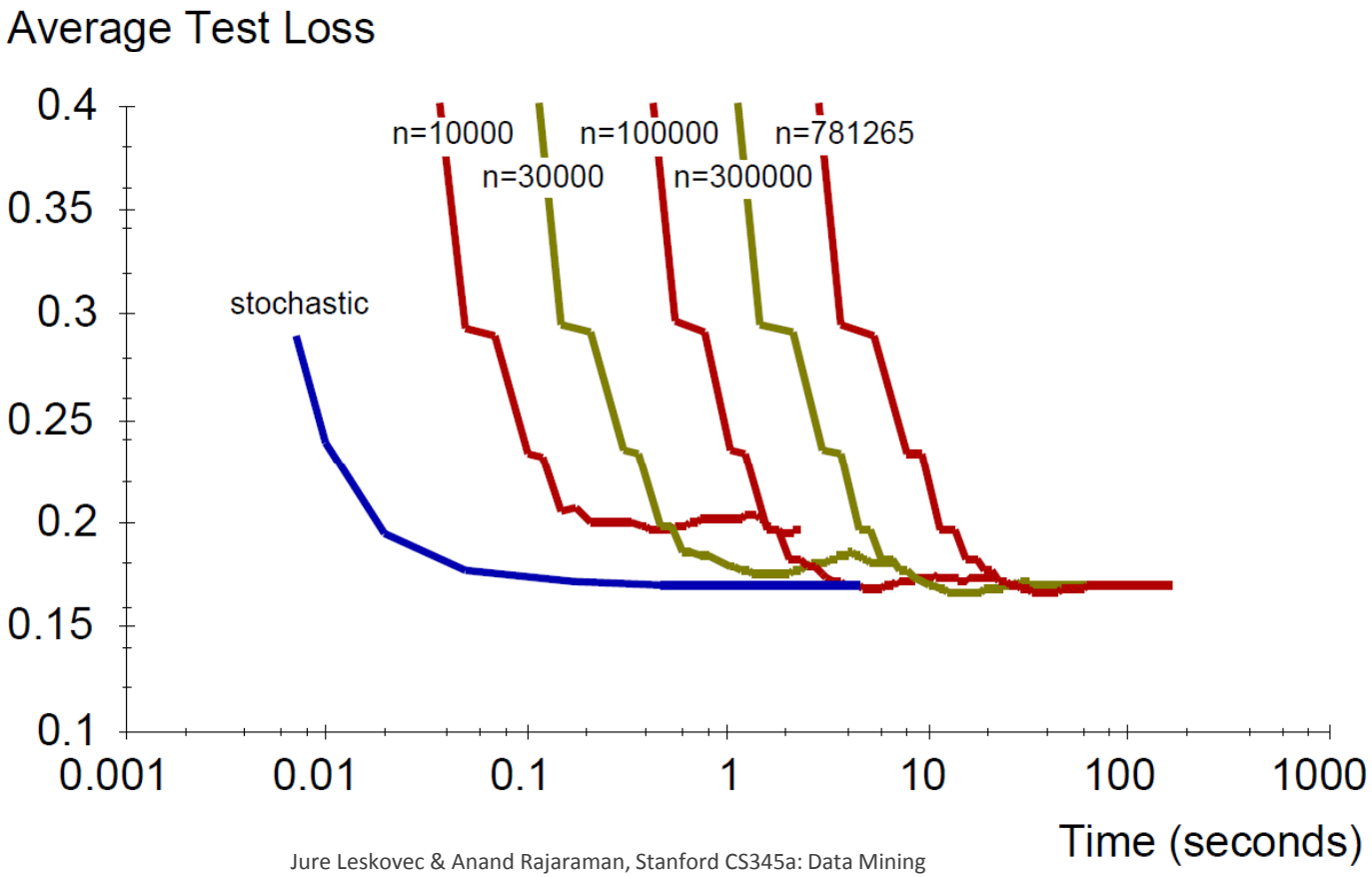
	Training Time	Primal cost	Test Error
SVMLight	23,642 secs	*0.2275*	*6.02%*
SVMPerf	66 secs	0.2278	6.03%
SGD	1.4 secs	*0.2275*	*6.02%*

Optimization accuracy



Subsampling

- What if we subsample the dataset?
  - SGD on full dataset vs.
  - Conjugate gradient on n training examples



Practical considerations

- Need to choose learning rate η:

$$w_{t+1} \leftarrow w_t - \eta_t L'(w)$$

- Leon suggests:
  - Select small subsample
  - Try various rates η
  - Pick the one that most reduces the loss
  - Use η for next 100k iterations on the full dataset

Practical considerations

- Stopping criteria:
  How many iterations of SGD?
  - Early stopping with cross validation
    - Create validation set
    - Monitor cost function on the validation set
    - Stop when loss stops decreasing
  - Early stopping a priori
    - Extract two disjoint subsamples A and B of training data
    - Determine the number of epochs k by training on A, stop by validating on B
    - Train for k epochs on the full dataset

Practical considerations

- Kernel function: $K(x_i,x_j)=\phi(x_i) \cdot \phi(x_j)$
- Does the SVM kernel trick still work?
- Yes (but not without a price)
  - Represent w with its kernel expansion:

    $\Sigma_i \alpha_i \cdot \phi(x_i)$
  - Usually:

    $dL(w)/dw = - \mu \cdot \phi(x_j)$
  - Then update w at epoch *t* by combining α:

    $\alpha_t = (1- \eta \cdot \lambda) \alpha_t + \mu \cdot \lambda$

PEGASOS

[Shalev-Shwartz et al. ICML '07]

INPUT: training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$,
regularization parameter λ
number of iterations T

$|A_t| = S$
Subgradient method

$|A_t| = 1$
Stochastic gradient

INITIALIZE: Choose $\mathbf{w}_1$ s.t. $\|\mathbf{w}_1\| \leq 1/\sqrt{\lambda}$

FOR $t = 1, 2, \ldots, T$

Subgradient

Choose $A_t \subseteq S$

$A_t^+ = \{(\mathbf{x}, y) \in A_t : y\langle \mathbf{w}_t, \mathbf{x}\rangle < 1\}$

$\nabla_t = \lambda \mathbf{w}_t - \frac{\eta_t}{|A_t|} \sum_{(\mathbf{x},y)\in A_t^+} y\,\mathbf{x}$

$\eta_t = \frac{1}{t\,\lambda}$

$\mathbf{w}'_t = \mathbf{w}_t - \eta_t \nabla_t$

Projection

$\Leftarrow \mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|\mathbf{w}'_t\|}\right\} \mathbf{w}'_t$

OUTPUT: $\mathbf{w}_{T+1}$

[Shalev-Shwartz et al. ICML '07]

Run-Time of Pegasos

- Choosing $|A_t|=1$ and a linear kernel over R^n
- Theorem [Shalev-Shwartz et al. '07]:
  - Run-time required for Pegasos to find ε accurate solution with prob. >1-δ

$$\tilde{O}\left(\frac{n}{\delta\,\lambda\,\epsilon}\right)$$

- Run-time depends on number of features n
- Does not depend on #examples m
- Depends on "difficulty" of problem (λ and ε)