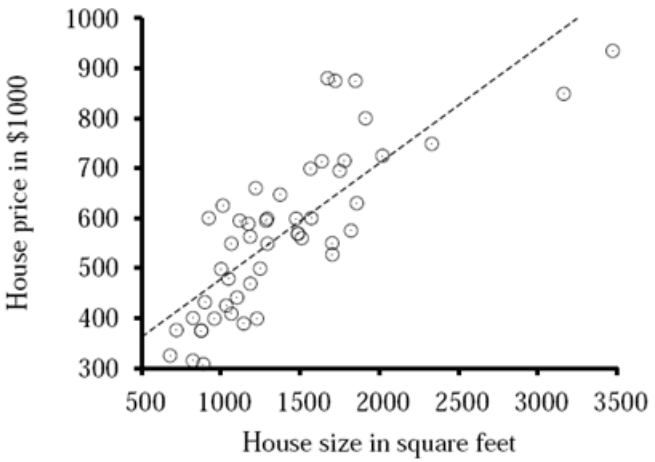
Learning from Examples

(Part B)

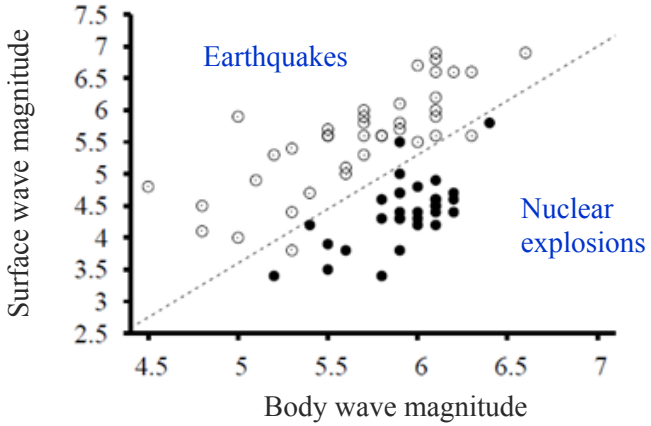
Contents

Regression and Classification with Linear Models

- Regression with a linear function
  - Univariate case
  - Multivariate case



- Linear functions turned into classifiers by applying
  - Hard threshold
  - Soft threshold



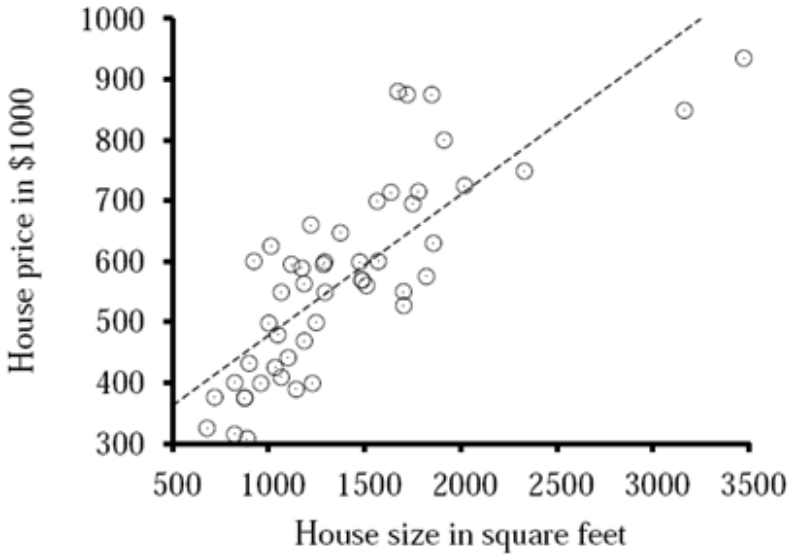
Univariate Linear Regression

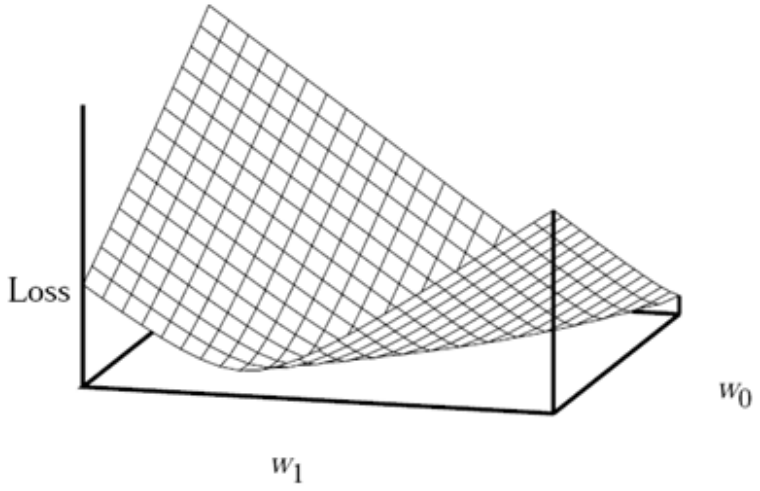
- The task of finding $h_{\mathbf{w}}$ that best fits the data

$$h_{\mathbf{w}}(x) = w_1 x + w_0$$

- Need to find the values of the weights $[w_0, w_1]$ that minimize the squared loss over all the training examples:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$





Univariate Linear Regression

- To minimize the sum $\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = -2\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0)) = 0$$

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = -2\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))x_j = 0$$

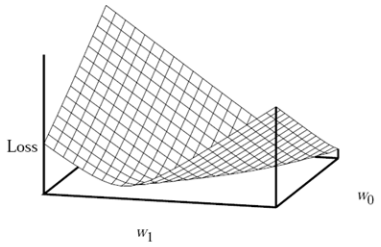
These equations have a unique solution:

$$w_0 = \left(\sum y_j - w_1\left(\sum x_j\right)\right)/N; \qquad w_1 = \frac{N\left(\sum x_j y_j\right) - \left(\sum x_j\right)\left(\sum y_j\right)}{N\left(\sum x_j^2\right) - \left(\sum x_j\right)^2}$$

- Linear regression problems with an L_2 loss function

$\rightarrow$ the loss function is convex in the weight space

$\rightarrow$ no local minima



Univariate Linear Regression

- To go beyond linear models where we often have no closed form solution, we apply gradient descent search in the weight space:

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in w do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

Here, α (step size) is called the learning rate:

- can be a constant, or
- can decay over time as the learning process proceeds

Univariate Linear Regression

- This update rule can also be used for univariate linear regression
  - Batch gradient descent
    - Cycle through all the training data for every step
    - Small α guarantees convergence (but slow)
  - Stochastic gradient descent
    - Take a step after each single example (can be used online)
    - No guarantee of convergence with fixed α (but fast)

Univariate Linear Regression

- In the case of one training example, (x, y):

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(x))^2$$

$$= 2(y - h_{\mathbf{w}}(x)) \times \frac{\partial}{\partial w_i} (y - (w_1 x + w_0))$$

Chain rule: $\frac{dL}{dw} = \frac{dL}{de} \cdot \frac{de}{dw}$

where $L = f(e)$ and $e = g(w)$

Therefore,

$$\frac{\partial}{\partial w_0} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)); \quad \frac{\partial}{\partial w_1} Loss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \times x$$

and the update rule (perceptron learning rule) becomes

$$w_0 \leftarrow w_0 + \alpha(y - h_{\mathbf{w}}(x)); \quad w_1 \leftarrow w_1 + \alpha(y - h_{\mathbf{w}}(x)) \times x$$

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- For N training examples (batch learning rule),

$$w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)); \quad w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$$

Multivariate Linear Regression

- Each example $\mathbf{x}_j$ is an n-element vector

$$h_{\mathbf{w}}(\mathbf{x}_j) = \mathbf{w} \cdot \mathbf{x}_j = \mathbf{w}^T \mathbf{x}_j = \sum_i w_i x_{j,i}$$

where $x_{j,0} = 1$ is a dummy input attribute

- The best weight vector minimizes loss over the examples:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j L_2(y_j, \mathbf{w} \cdot \mathbf{x}_j)$$

- The update rule for batch gradient descent is

$$w_i \leftarrow w_i + \alpha \sum_j x_{j,i}(y_j - h_{\mathbf{w}}(\mathbf{x}_j))$$

- The analytical solution is

$$\mathbf{w}^* = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

where $\mathbf{X}$ is the data matrix of inputs with one n-dimensional examples per row

Multivariate Linear Regression

- Overfitting by irrelevant attributes can be avoided by regularization

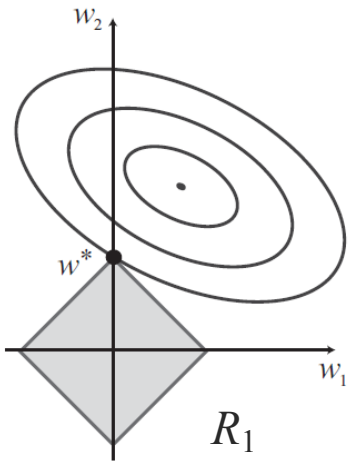
$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h)$$

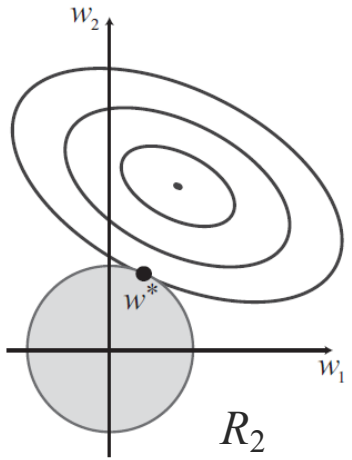
(With univariate linear regression we didn't have to worry about overfitting)

- A family of regularization functions for linear functions:

$$Complexity(h_{\mathbf{w}}) = R_q(\mathbf{w}) = \sum_i |w_i|^q$$

  - R_1 tends to produce a sparse model by setting some weights to zero (Using R_2 still allows an analytical solution but R_1 does not)





$$\frac{\partial}{\partial w_i} Cost(h_{\mathbf{w}}) = \frac{\partial}{\partial w_i} Loss(h_{\mathbf{w}}) + \lambda \frac{\partial}{\partial w_i}\left(\sum_j w_j^2\right)$$

(See page 5)

Minimize $EmpLoss(\mathbf{w})$ subject to the constraint $Complexity(\mathbf{w}) \le c$

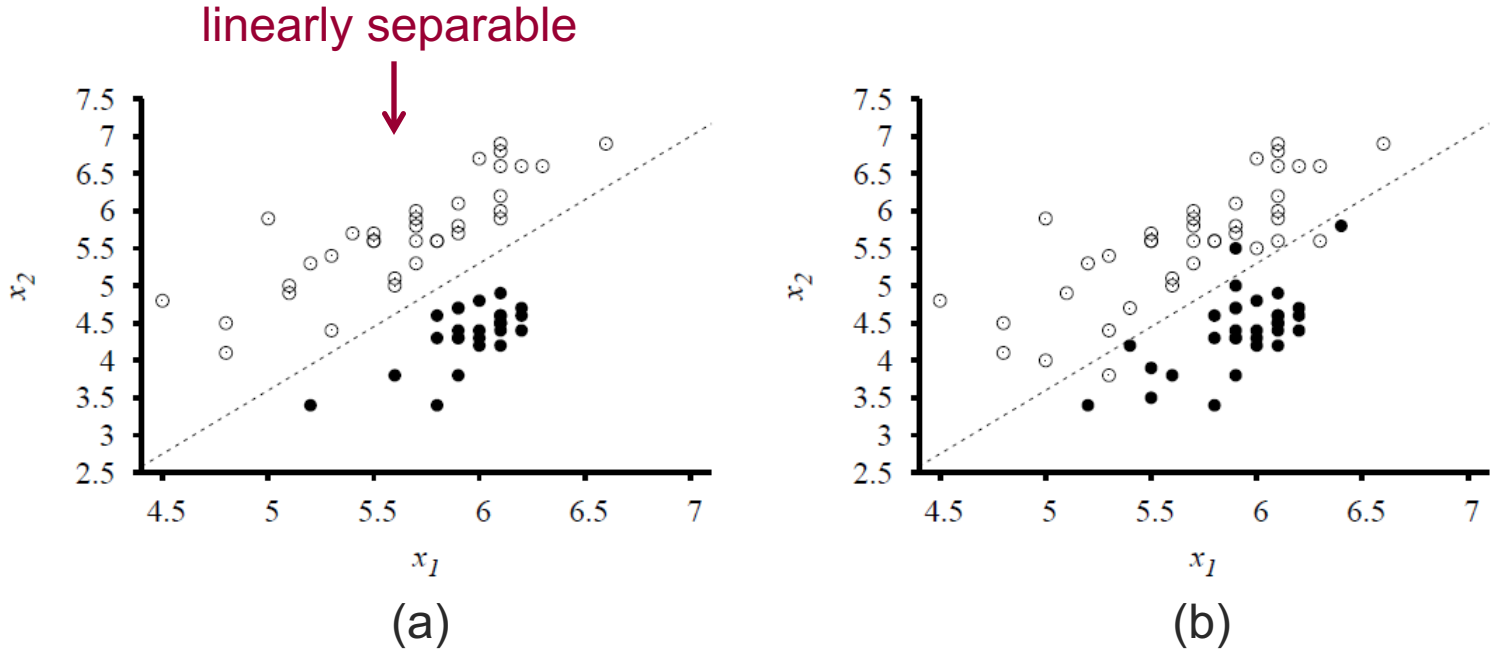
Linear Classifiers with a Hard Threshold

- Linear decision boundary (linear separator):

  $\mathbf{w} \cdot \mathbf{x} = 0$ ($x_0 = 1$ is a dummy input)

- Classification hypothesis:

  $h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ and 0 otherwise

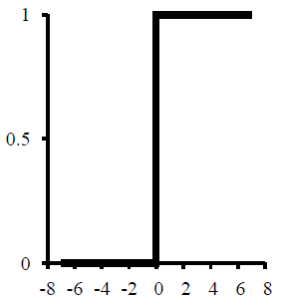


Linear Classifiers with a Hard Threshold

- An alternative mathematical form:

  $$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x})$$

  where $Threshold(z) = 1$ if $z \geq 0$ and 0 otherwise



- Minimize loss by gradient descent ??
  - Gradient is zero almost everywhere in the weight space
- Perceptron learning rule guarantees convergence for linearly separable data:

  $$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$ for a single example $(\mathbf{x}, y)$

- Note: both y and $h_{\mathbf{w}}(\mathbf{x})$ are either 0 or 1
  - No weight change for correct output
  - $y = 1, h_{\mathbf{w}}(\mathbf{x}) = 0$: w_i increased when $x_i > 0$, decreased when $x_i < 0$
  - $y = 0, h_{\mathbf{w}}(\mathbf{x}) = 1$: w_i decreased when $x_i > 0$, increased when $x_i < 0$

Linear Classifiers with a Hard Threshold

- Training curves by update rule applied one example at a time:

  (a) Linearly separable data of (a) on p. 11

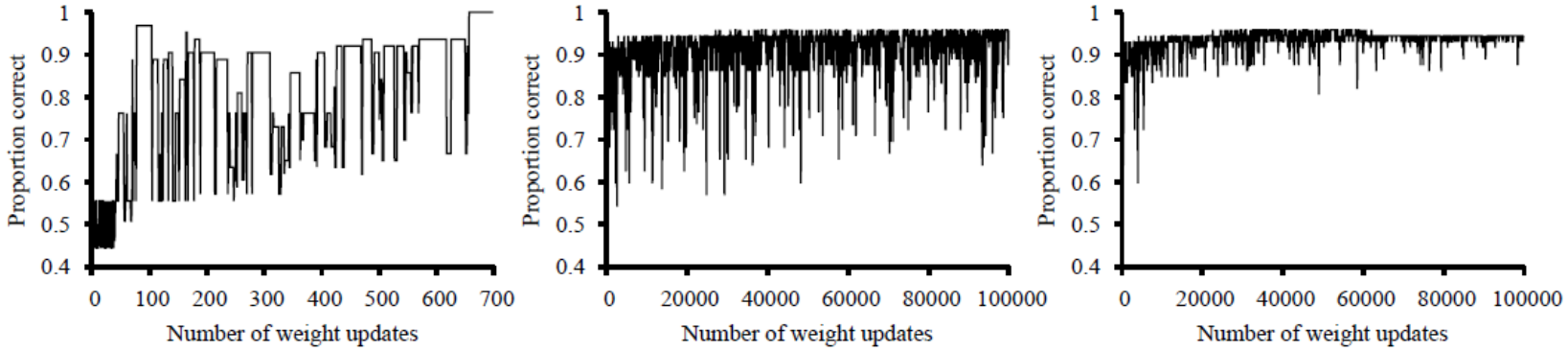
  657 steps to converge with 63 examples

  (b) Nonseparable data of (b) on p. 11

  Failing to converge even after 100,000 steps with fixed α

  (c) The same nonseparable data of (b)

  Not perfect after 100,000 steps but nearly converging to minimum-error solution with a learning rate schedule $\alpha(t) = 1000/(1000 + t)$

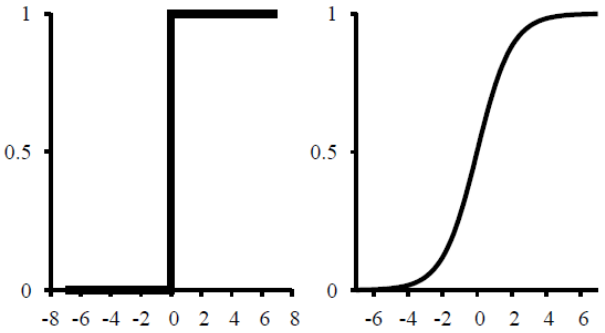


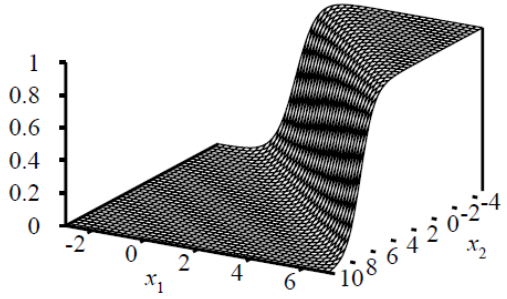
Linear Classification with Logistic Regression

- Problems with linear threshold function:
  - Learning with perceptron rule is very unpredictable because $h_{\mathbf{w}}(\mathbf{x})$ is discontinuous and not differentiable
  - Only $0/1$ predictions even for boundary examples
- We can replace the threshold function with the logistic function

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

  - Output is interpreted as probability of belonging to class 1





For the data of (b) on p. 11

Linear Classification with Logistic Regression

- Logistic regression uses gradient descent:

Chain rule: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

where $z = f(y)$ and $y = g(x)$

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

$$= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i} (y - g(\mathbf{w} \cdot \mathbf{x})) \quad (g\text{: logistic function})$$

$$= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i} \mathbf{w} \cdot \mathbf{x}$$

$$= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i$$

Note that the derivative of the logistic function satisfies $g' = g(1 - g)$,

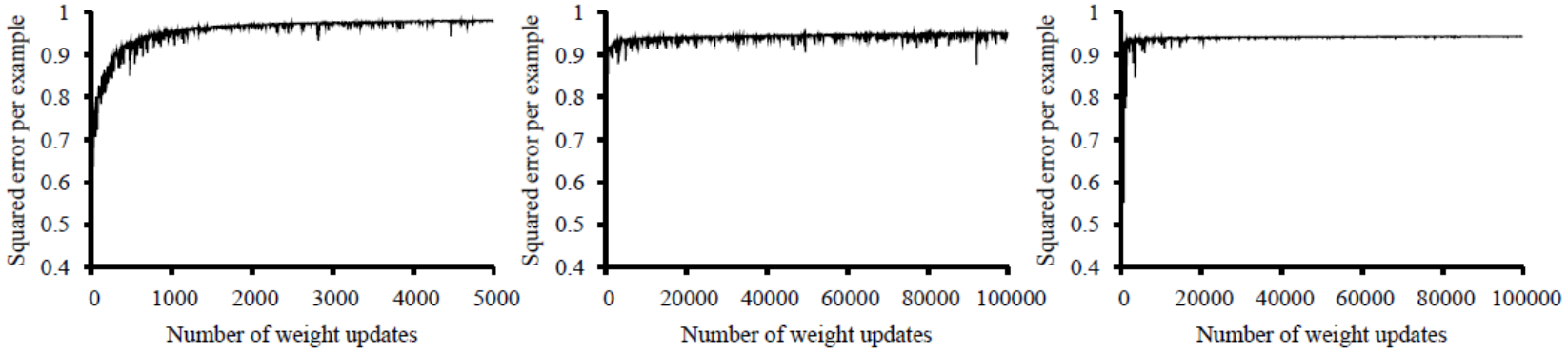
$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

So the update rule is

$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

Linear Classification with Logistic Regression

- Training curves by update rule applied one example at a time:
  - (a) Linearly separable data
    slower to converge but more predictable
  - (b) Nonseparable data with fixed α, and (c) with scheduled α converges far more quickly and reliably



(a) (b) (c)

Nonparametric Models

- Parametric model:
  - Summarizes data with a set of parameters of fixed size
  - Assumes that the data is drawn from a model of known form

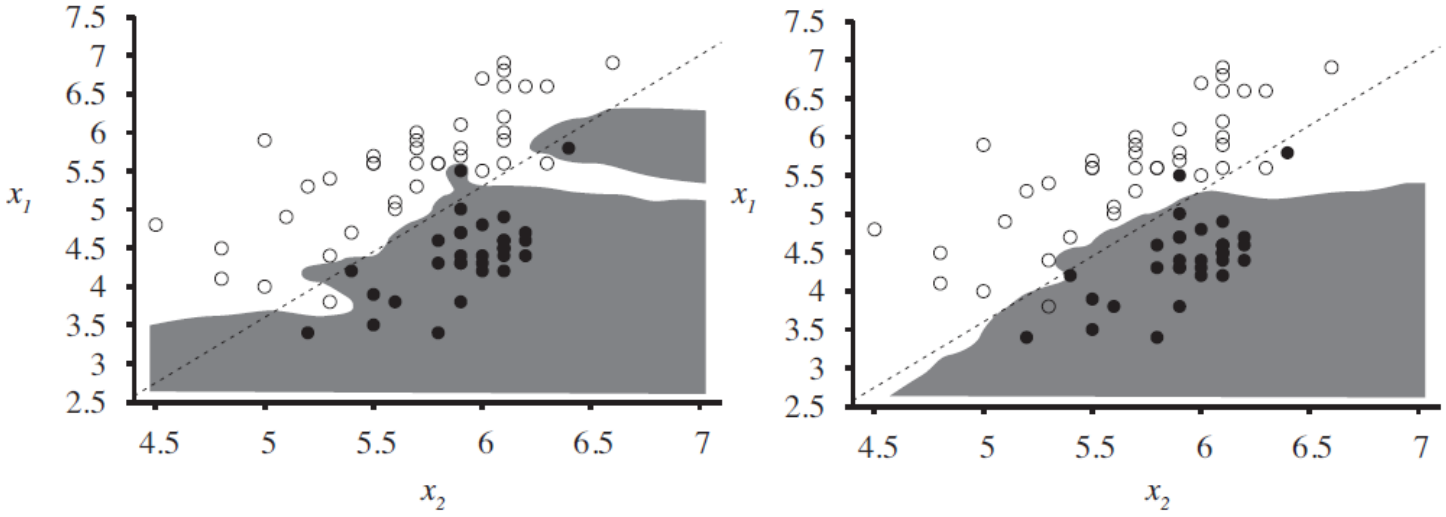
    E.g., linear regression
- Nonparametric model:
  - When the data cannot be characterized by a bounded set of parameters
  - We let the data speak for themselves

    (especially when a large volume of data are available)

    E.g., instance-based (memory-based) learning

Nearest Neighbor Models

- Given a query $\mathbf{x}_q$, find the k nearest neighbors $NN(k, \mathbf{x}_q)$
  - Classification: plurality vote of $NN(k, \mathbf{x}_q)$
  - Regression: mean or median of $NN(k, \mathbf{x}_q)$ or
    solve a linear regression problem on $NN(k, \mathbf{x}_q)$
  - Can use cross-validation to select the best k



Overfitting with $k = 1$ O.K. with $k = 5$

Nearest Neighbor Models

- Distance metric: Minkowski distance (L^p norm)

$$L^p(\mathbf{x}_j, \mathbf{x}_q) = \left(\sum_i \left| x_{j,i} - x_{q,i} \right|^p \right)^{1/p}$$

  - $p = 2$: Euclidean distance
  - $p = 1$: Manhattan distance
  - $p = 1$ with Boolean attributes: Hamming distance

- Normalization:

$$x_{j,i} \rightarrow (x_{j,i} - \mu_i)/\sigma_i$$

  - μ_i: mean of the values in the ith dimension
  - σ_i: standard deviation of the values in the ith dimension

Nearest Neighbor Models

- Curse of dimensionality: nearest neighbors are not very near!

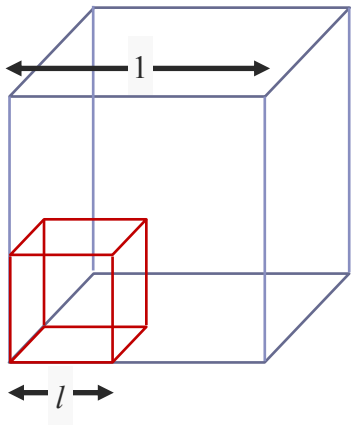
  l: average side length of a neighborhood

  l^n: volume of the neighborhood hypercube

  - If N points are uniformly distributed in the full hypercube of volume 1,

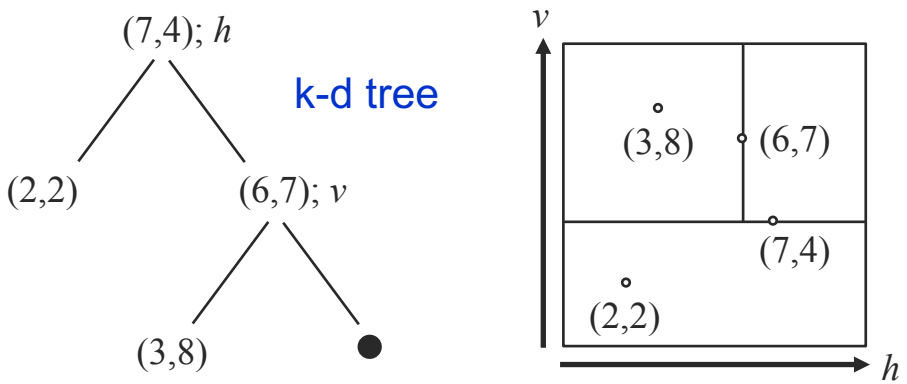
    $$l^n/1 = k/N \rightarrow l = (k/N)^{1/n}$$

  - E.g., let $k = 10$ and $N = 1{,}000{,}000$
    - $n = 3 \rightarrow l = 0.02$
    - $n = 17 \rightarrow l = 0.5$
    - $n = 200 \rightarrow l = 0.94$



Nearest Neighbor Models

- Time complexity:
  - $O(N)$ with a sequential table
  - $O(\log N)$ with a binary tree → k-d tree
  - $O(1)$ with a hash table → locality-sensitive hashing



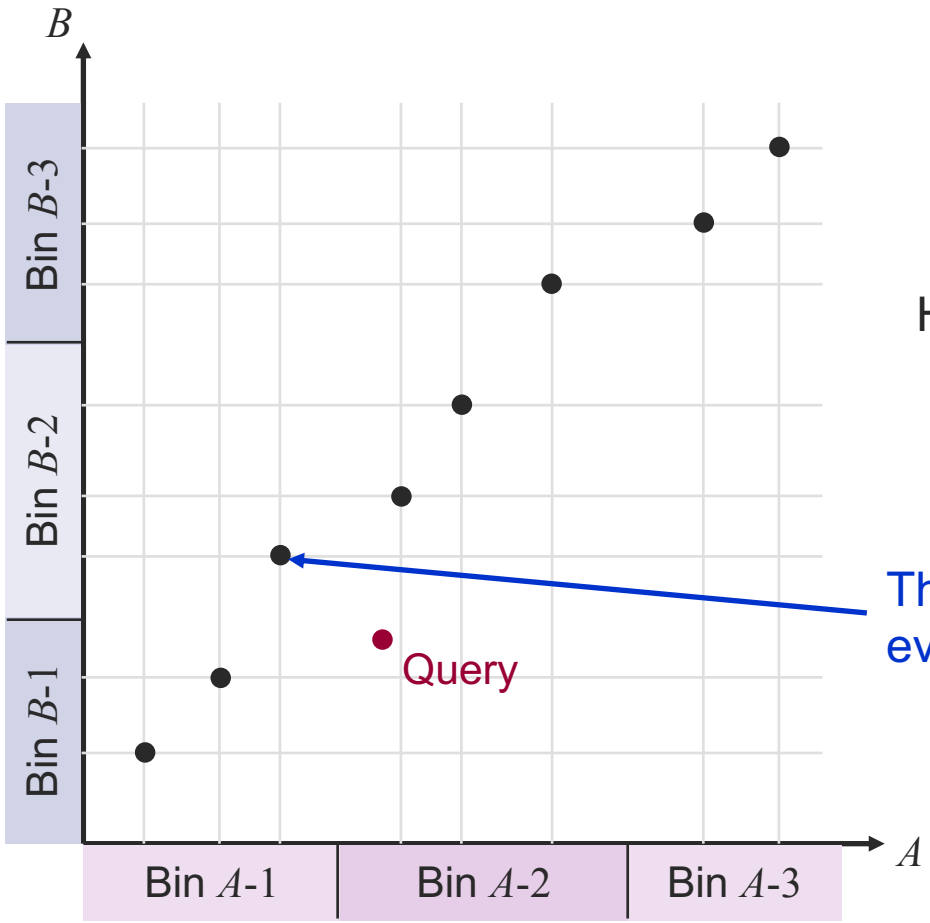
- Lazy learning:
  - All the real works are done at the time of classifying a new instance (not at the time of memorizing)
  - But tailor-made predictions can be made

Locality-Sensitive Hashing

- Approximate near-neighbors problem:
  - Given a data set of example points and a query point $\mathbf{x}_q$, find, with high probability, an example point (or points) that is near $\mathbf{x}_q$
- Locality-sensitive hash:
  - A hash table can be created by projecting the data onto a line and discretizing the line into hash bins
  - We choose l random projections and create l hash tables, $g_1(\mathbf{x}), \ldots, g_l(\mathbf{x})$
  - Given a query point $\mathbf{x}_q$, we fetch the set of points in bin $g_j(\mathbf{x}_q)$ for each j, and union them into a set of candidate points C
  - We find the k closest points from C by computing the actual distance to $\mathbf{x}_q$

Locality-Sensitive Hashing

Example:



Hash function:

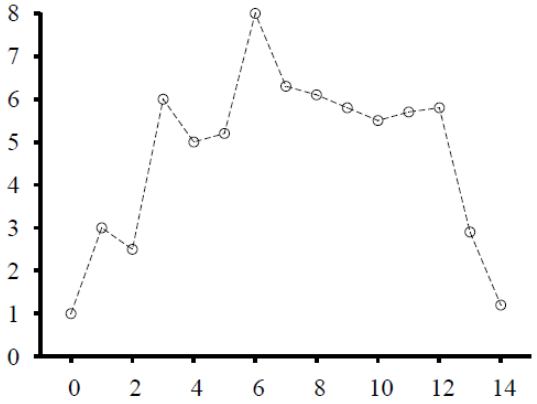
If $x_{k-1} \leq X < x_k$, then Bin X-k

This point cannot be a candidate even if it is closest to the query

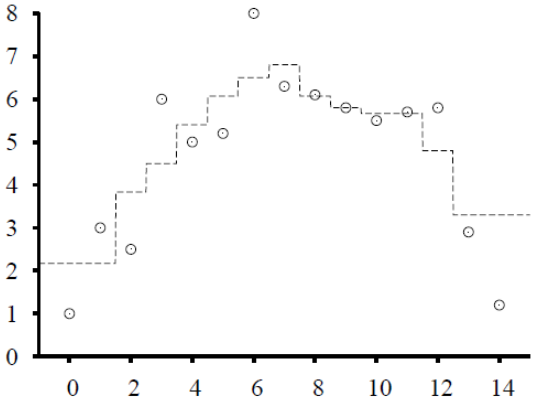
Nonparametric Regression

- k-nearest-neighbors regression:
  - k-NN average: $h(x) = \sum (y_j / k)$
    - Poor estimate at outlying points
      (evidence comes from one side, ignores trend)
  - k-NN linear regression
    - Finds best line through k examples
    - Captures trend at outliers
- Locally weighted regression:
  - Avoids discontinuities in $h(x)$
  - Examples are weighted by a kernel function
  - Weight decreases gradually as the distance to the query point increases

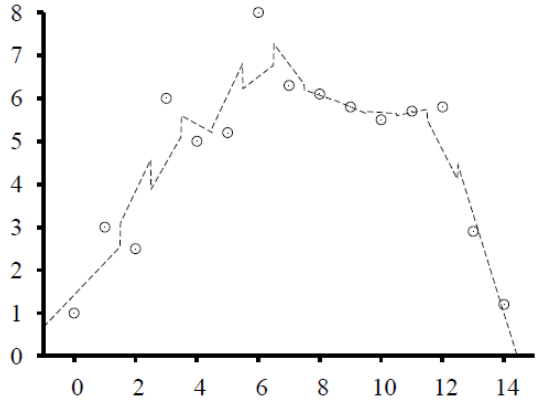
Nonparametric Regression



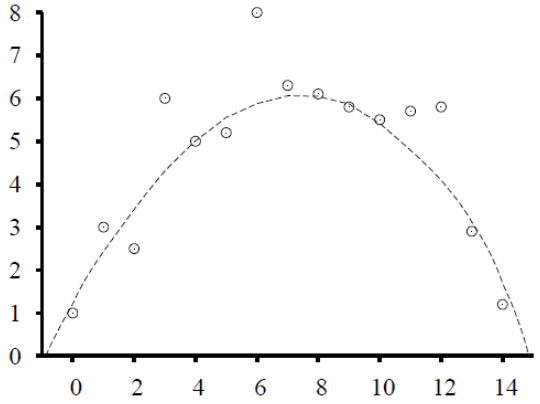
(a) connect the dots



(b) 3-NN average



(c) 3-NN linear regression

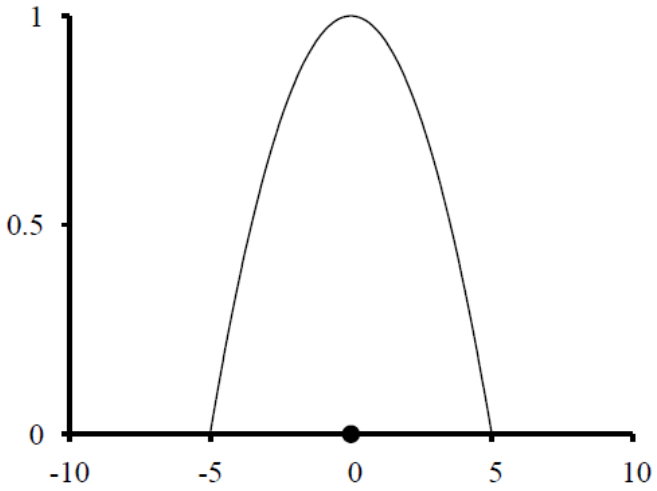


(d) locally weighted regression (quadratic kernel, $k = 10$)

Nonparametric Regression

- Kernel function:
  - Should be symmetric around 0, have a maximum at 0
  - Area under the kernel must be bounded
  - The shape does not matter much, kernel width is more important (underfitting vs. overfitting)
  - Best kernel width can be chosen by cross-validation

A quadratic kernel,
$\mathcal{K}(x) = \max(0,\ 1 - (2|x|/k)^2)$,
with kernel width $k = 10$,
centered on the query point $x = 0$



Nonparametric Regression

- Given a query point $\mathbf{x}_q$, we solve the following problem using gradient descent:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathrm{x}_j))(y_j - \mathbf{w} \cdot \mathbf{x}_j)^2$$

Then, the answer is $h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q$

  - New regression problem should be solved for every query point, but with just a few points

Ensemble Learning

- Idea:
  - Select a collection of hypotheses and combine their predictions
  - Misclassification is much less likely than by a single hypothesis

- Multiple, diverse predictive models are constructed from adapted versions of the training data
  - Resampled
  - Reweighted
- The predictions of these models are combined in some way
  - Averaging
  - Voting (possibly weighted)

Bagging

- Diverse models are generated on different random samples of the original data set (Bagging: Bootstrap aggregating)
  - Samples of the same size as the original data are taken uniformly with replacement → called bootstrap samples
  - Some of the original data will be missing and some others will be duplicated
    - Probability of a data point being not selected for a bootstrap sample of size n: $(1 - 1/n)^n$

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n = \frac{1}{e} \approx 0.368$$

- Bagging improves unstable learning schemes:
  - Performance is improved in almost all cases if the learning scheme is unstable (e.g., decision tree)

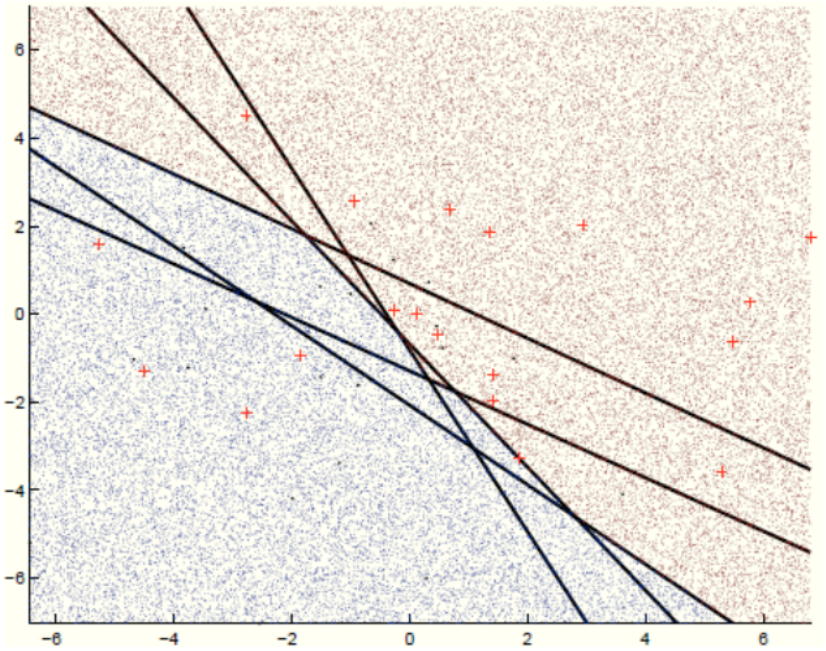
Bagging

function BAGGING(D, K, L) **returns** a set of unweighted hypotheses
  inputs: data set D; ensemble size K; learning algorithm L.
  output: ensemble of models whose predictions are to be combined by voting or averaging
  for $k = 1$ **to** K **do**
    build a bootstrap sample D_k from D by sampling $|D|$ data points with replacement;
    run L on D_k to produce a model M_k;
  end
  return $\{M_k \mid 1 \leq k \leq K\}$

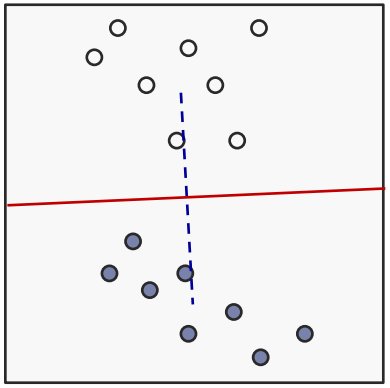
- To combine the predictions from different models
  - Voting/averaging
  - Each model receives equal weight

Bagging

Example: Five basic linear classifiers on bootstrap samples from 20 positive and 20 negative examples



The decision boundary is a perpendicular bisector of the line between the positive and negative centers of mass



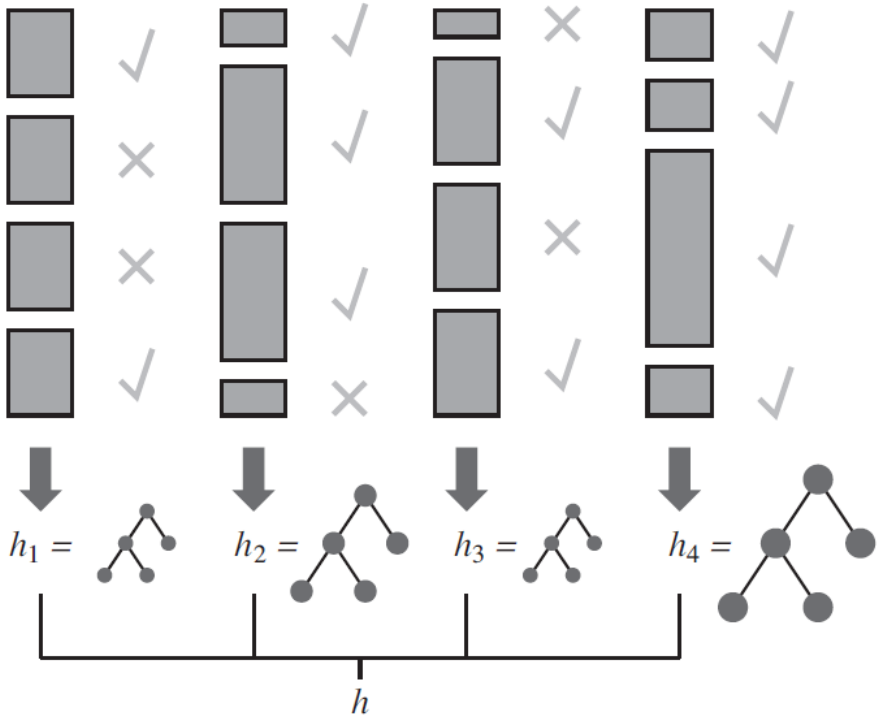
The decision rule is majority vote, leading to a piecewise linear decision boundary

Boosting

- Start with a normal training set and learn h_1
- Increase the weights of the misclassified examples and learn h_2
- Continue until K hypotheses are generated
- The final ensemble hypothesis is a weighted-majority combination of all the K hypotheses
  - Each weighted according to its performance on the training set

- ADABOOST algorithm:
  - Given a weak learning algorithm (slightly better than random guessing), the ensemble classifies the training data perfectly for large enough K

Boosting



- The height of the rectangle (example) corresponds to the weight
- The checks and crosses indicate whether the example was classified correctly by the current hypothesis
- The size of the decision tree indicates the weight of that hypothesis in the final ensemble

Boosting

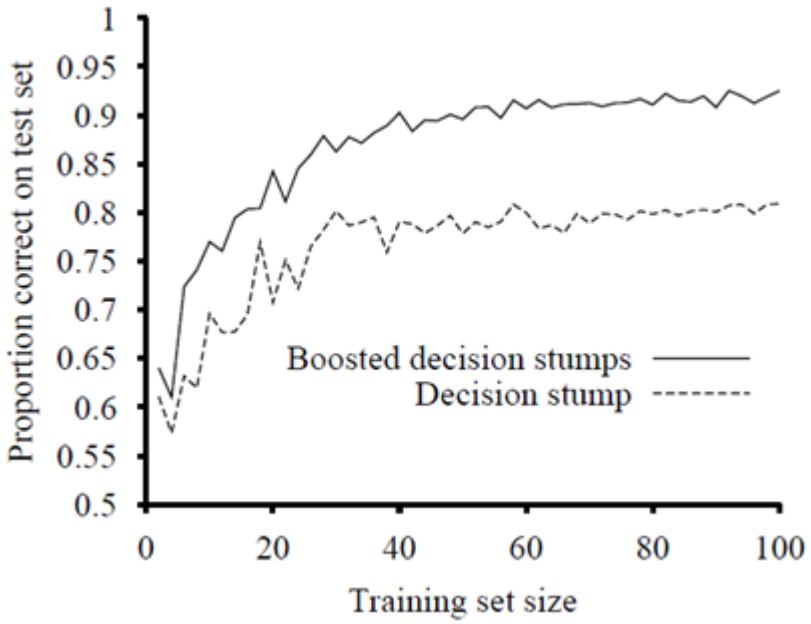
function ADABOOST(D, K, L) **returns** a weighted-majority hypothesis
  inputs: data set D of size N; ensemble size K; learning algorithm L.
  local variables: **w**, a vector of N example weights, initially $1/N$
    h, a vector of K hypotheses
    z, a vector of K hypothesis weights
  for $k = 1$ **to** K **do**
    $\mathbf{h}[k] \leftarrow L(D, \mathbf{w})$
    $error \leftarrow 0$
    for $j = 1$ to N **do**
      if $\mathbf{h}[k](x_j) \neq y_j$ **then** $error \leftarrow error + \mathbf{w}[j]$
    for $j = 1$ to N **do**
      if $\mathbf{h}[k](x_j) = y_j$ **then** $\mathbf{w}[j] \leftarrow \mathbf{w}[j] \cdot error/(1 - error)$
    $\mathbf{w} \leftarrow$ NORMALIZE(**w**)
    $\mathbf{z}[k] \leftarrow \log(1 - error)/ error$
  return WEIGHTED-MAJORITY(**h**, **z**)

	$error = 0.1$	$error = 0.2$
$error/(1 - error)$	1/9	2/8

Boosting

Example:

(a) Boosting (with $K = 5$) improves the performance of decision stumps (decision trees with depth 1) from 81% to 93% on 100 training examples



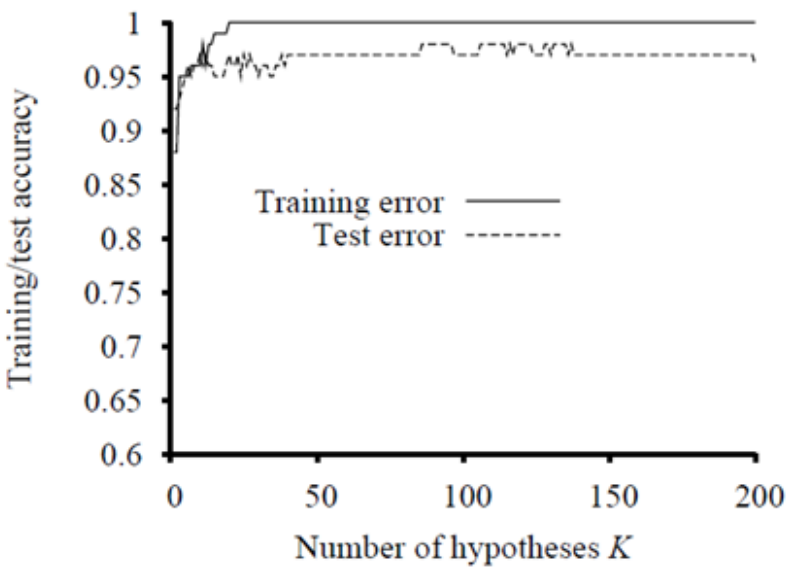
Boosting

Example:

(b) The training error reaches zero when $K = 20$

The test error continues to decrease long after the training error has reached zero (Accuracy 0.95 at $K = 20 \rightarrow 0.98$ at $K = 137$)

(against Ockham's razor?)



- Boosting is effective in increasing the margins of examples, even if they are already on the correct side of the decision boundary