

# **Regression and Classification with Linear Models**

**CMPSCI 383  
Nov 15, 2011**

# Today's topics

---

- Learning from Examples: brief review
- Univariate Linear Regression
  - Batch gradient descent
  - Stochastic gradient descent
- Multivariate Linear Regression
  - Regularization
- Linear Classifiers
  - Perceptron learning rule
- Logistic Regression

# Learning from Examples (supervised learning)

---

Simplest form: learn a function from examples (*tabula rasa*)

$f$  is the target function

An example is a pair  $x, f(x)$ , e.g., 

$O$	$O$	$X$
	$X$	
$X$		

,  $+1$

Problem: find a(n) hypothesis  $h$   
such that  $h \approx f$   
given a training set of examples

(This is a highly simplified model of real learning:

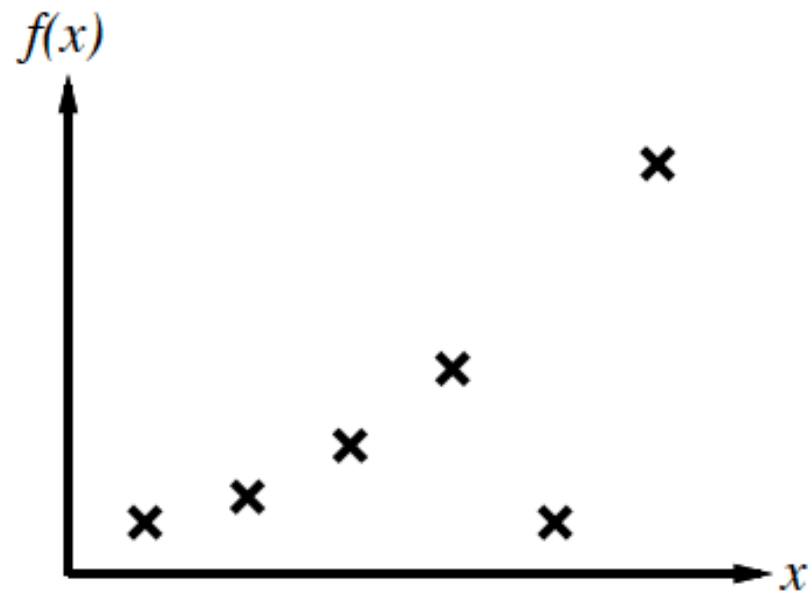
- Ignores prior knowledge
- Assumes a deterministic, observable “environment”
- Assumes examples are given
- Assumes that the agent wants to learn  $f$ —why?)

# Learning from Examples (supervised learning)

---

Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

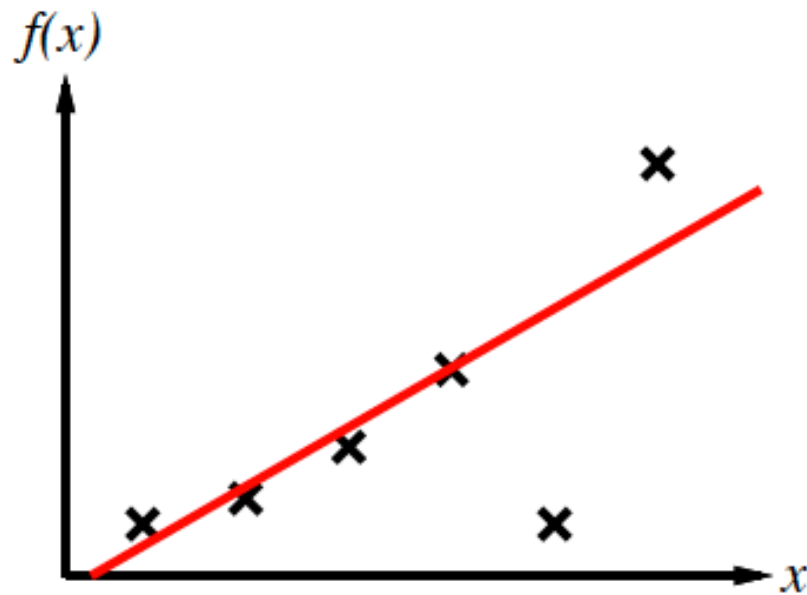


# Learning from Examples (supervised learning)

---

Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

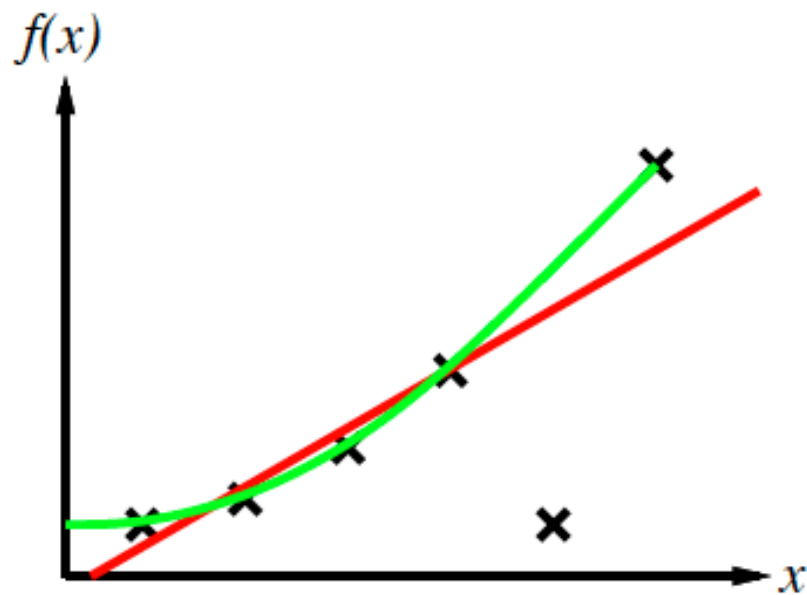


# Learning from Examples (supervised learning)

---

Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

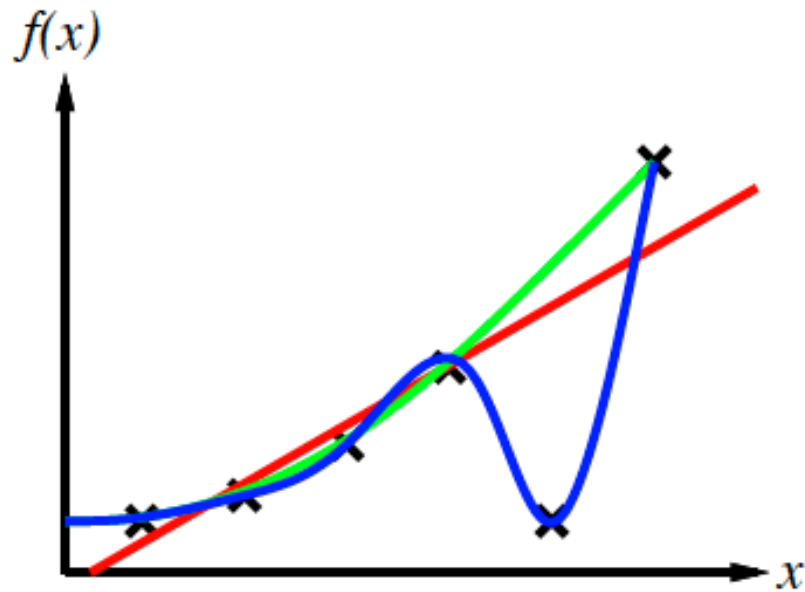


# Learning from Examples (supervised learning)

---

Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

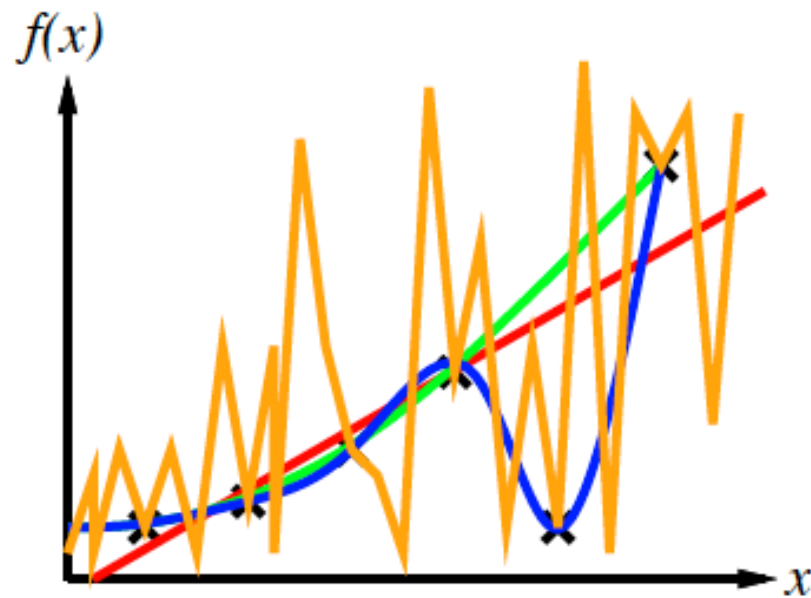


# Learning from Examples (supervised learning)

---

Construct/adjust  $h$  to agree with  $f$  on training set  
( $h$  is consistent if it agrees with  $f$  on all examples)

E.g., curve fitting:

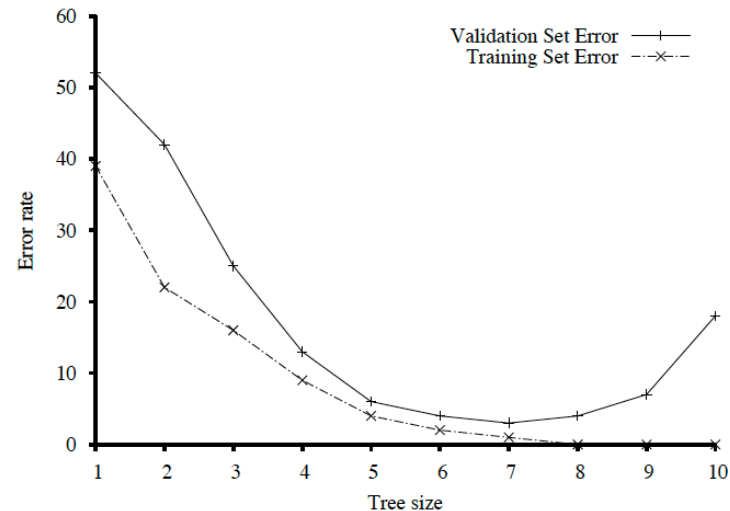




# Important issues

---

- Generalization
- Overfitting
- Cross-validation
  - Holdout cross validation
  - K-fold cross validation
  - Leave-one-out cross-validation
- Model selection



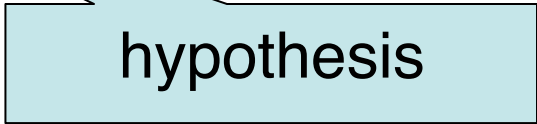
# Recall Notation

---

$(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$       training set

Where each  $y_j$  was generated by  
an unknown function  $y = f(\mathbf{x})$

Discover a function  $h$  that best  
approximates the true function  $f$



hypothesis

# Loss Functions

---

Suppose the true prediction for input  $\mathbf{x}$  is  $f(\mathbf{x}) = y$

but the hypothesis gives  $h(\mathbf{x}) = \hat{y}$

$$L(\mathbf{x}, y, \hat{y}) = \text{Utility}(\text{result of using } y \text{ given input } \mathbf{x}) \\ - \text{Utility}(\text{result of using } \hat{y} \text{ given input } \mathbf{x})$$

Simplified version:  $L(y, \hat{y})$

Absolute value loss:  $L_1(y, \hat{y}) = |y - \hat{y}|$

Squared error loss:  $L_2(y, \hat{y}) = (y - \hat{y})^2$

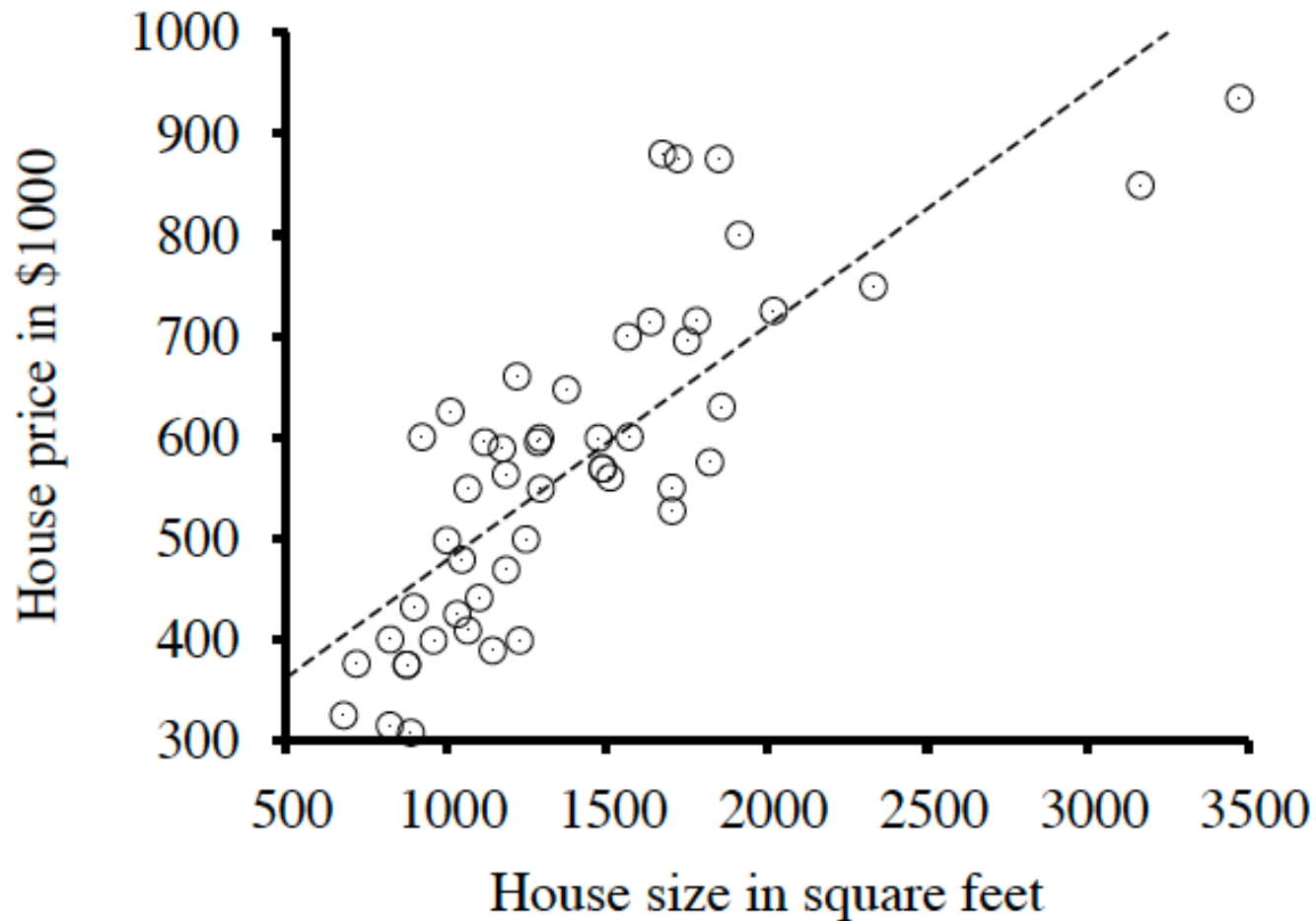
0/1 loss:  $L_{0/1}(y, \hat{y}) = 0$  if  $y = \hat{y}$ , else 1

Generalization loss: expected loss over all possible examples

Empirical loss: average loss over available examples

# Univariate Linear Regression

---



# Univariate Linear Regression contd.

---

$$\mathbf{w} = [w_0, w_1] \quad \text{weight vector}$$

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

Find weight vector that minimizes empirical loss,  
e.g.,  $L_2$ :

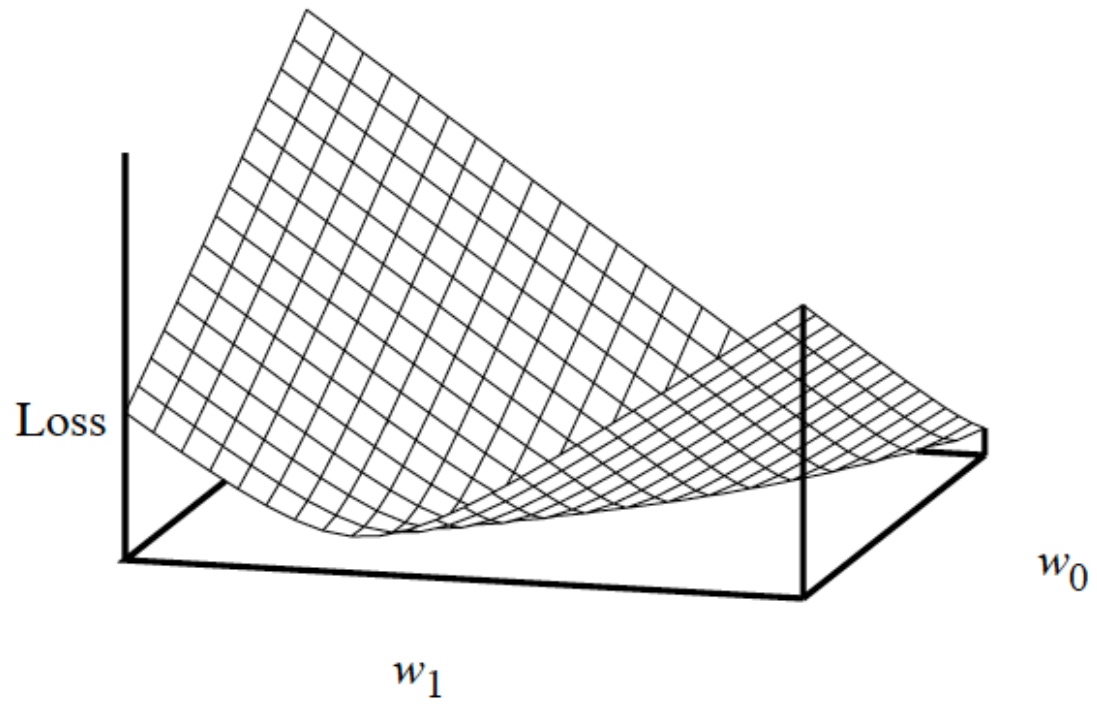
$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^N L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^N (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^N (y_j - (w_1 x_j + w_0))^2$$

i.e., find  $\mathbf{w}^*$  such that

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} Loss(h_{\mathbf{w}})$$

# Weight Space

---



## Finding $w^*$

---

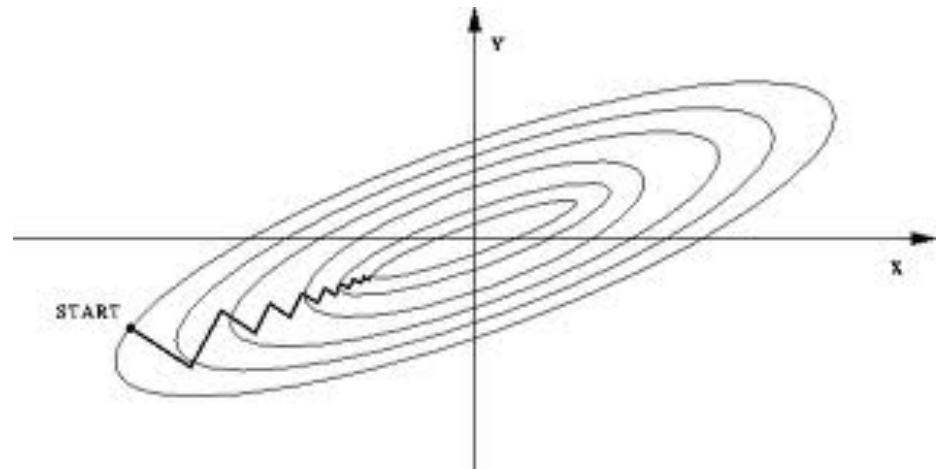
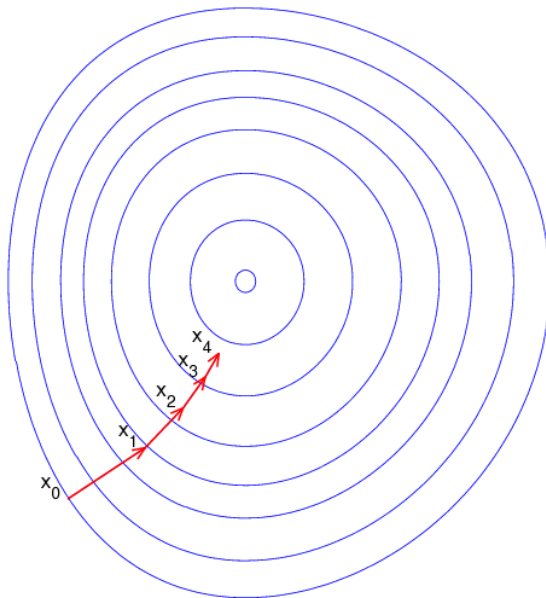
Find weights such that:

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

# Gradient Descent

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

step size or  
learning rate





## Gradient Descent contd.

---

For one training example  $(x, y)$ :

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

For  $N$  training examples:

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

batch gradient descent

stochastic gradient descent: take a step for one training example at a time

## The Multivariate case

---

$$h_{sw}(\mathbf{x}_j) = w_0 + w_1x_{j,1} + \cdots + w_nx_{j,n} = w_0 + \sum_i w_i x_{j,i}$$

Augmented vectors: add a feature to each  $\mathbf{x}$  by tacking on a 1:  $x_{j,0} = 1$

Then:

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

And batch gradient descent update becomes:

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

## The Multivariate case contd.

---

Or, solving analytically:

Let  $\mathbf{y}$  be the vector of outputs for the training examples

$\mathbf{X}$  data matrix: each row is an input vector

Solving this for  $\mathbf{w}^*$ :  $\mathbf{y} = \mathbf{X}\mathbf{w}$

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

pseudo inverse

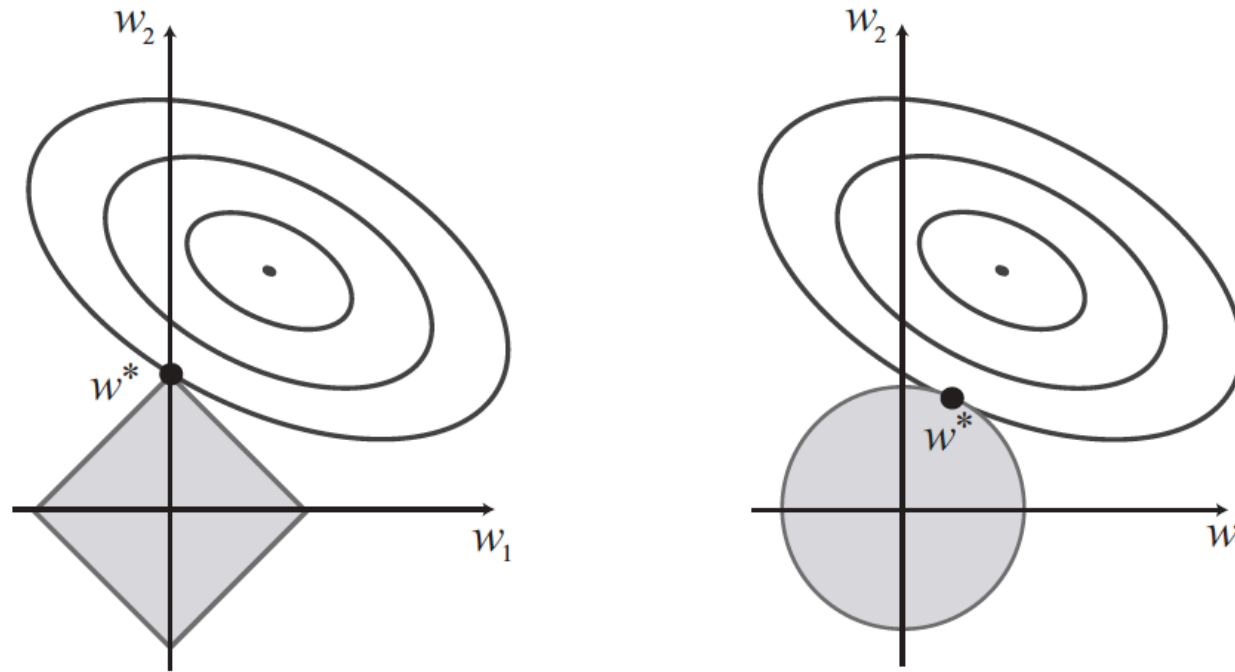
# Regularization

---

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

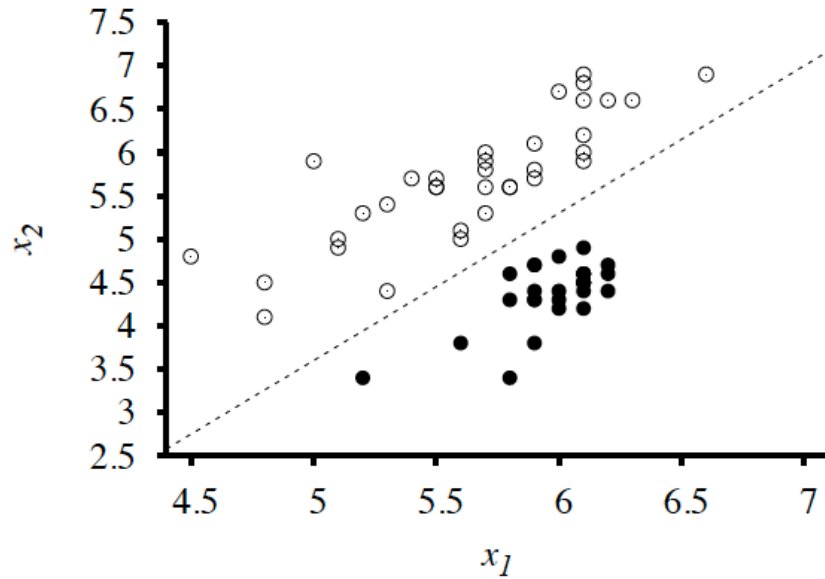
$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_i |w_i|^q$$

# L<sub>1</sub> vs. L<sub>2</sub> Regularization

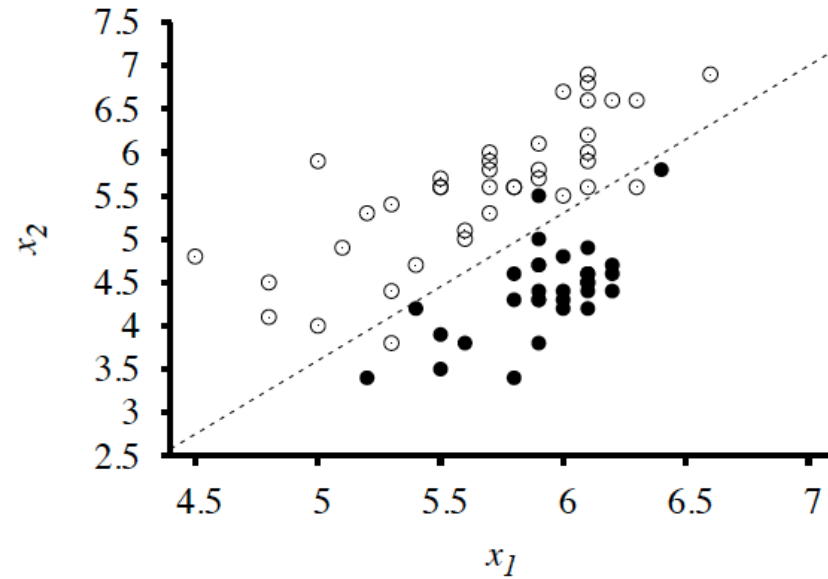


**Figure 18.14** FILES: figures/diamond.eps (Wed Nov 4 14:45:53 2009). Why  $L_1$  regularization tends to produce a sparse model. (a) With  $L_1$  regularization (box), the minimal achievable loss (concentric contours) often occurs on an axis, meaning a weight of zero. (b) With  $L_2$  regularization (circle), the minimal loss is likely to occur anywhere on the circle, giving no preference to zero weights.

# Linear Classification: hard thresholds



(a)



(b)

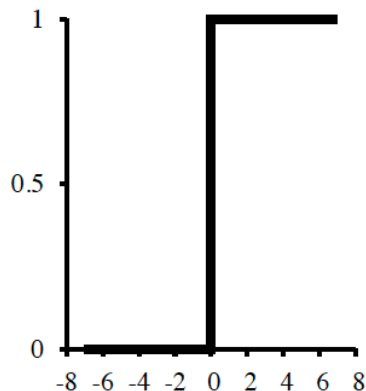
**Figure 18.15** FILES: . (a) Plot of two seismic data parameters, body wave magnitude  $x_1$  and surface wave magnitude  $x_2$ , for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East (?). Also shown is a decision boundary between the classes. (b) The same domain with more data points. The earthquakes and explosions are no longer linearly separable.

# Linear Classification: hard thresholds contd.

---

- Decision Boundary:
  - In linear case: linear separator, a hyperplane
- Linearly separable:
  - data is linearly separable if the classes can be separated by a linear separator
- Classification hypothesis:

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



# Perceptron Learning Rule

---

For a single sample  $(\mathbf{x}, y)$  :

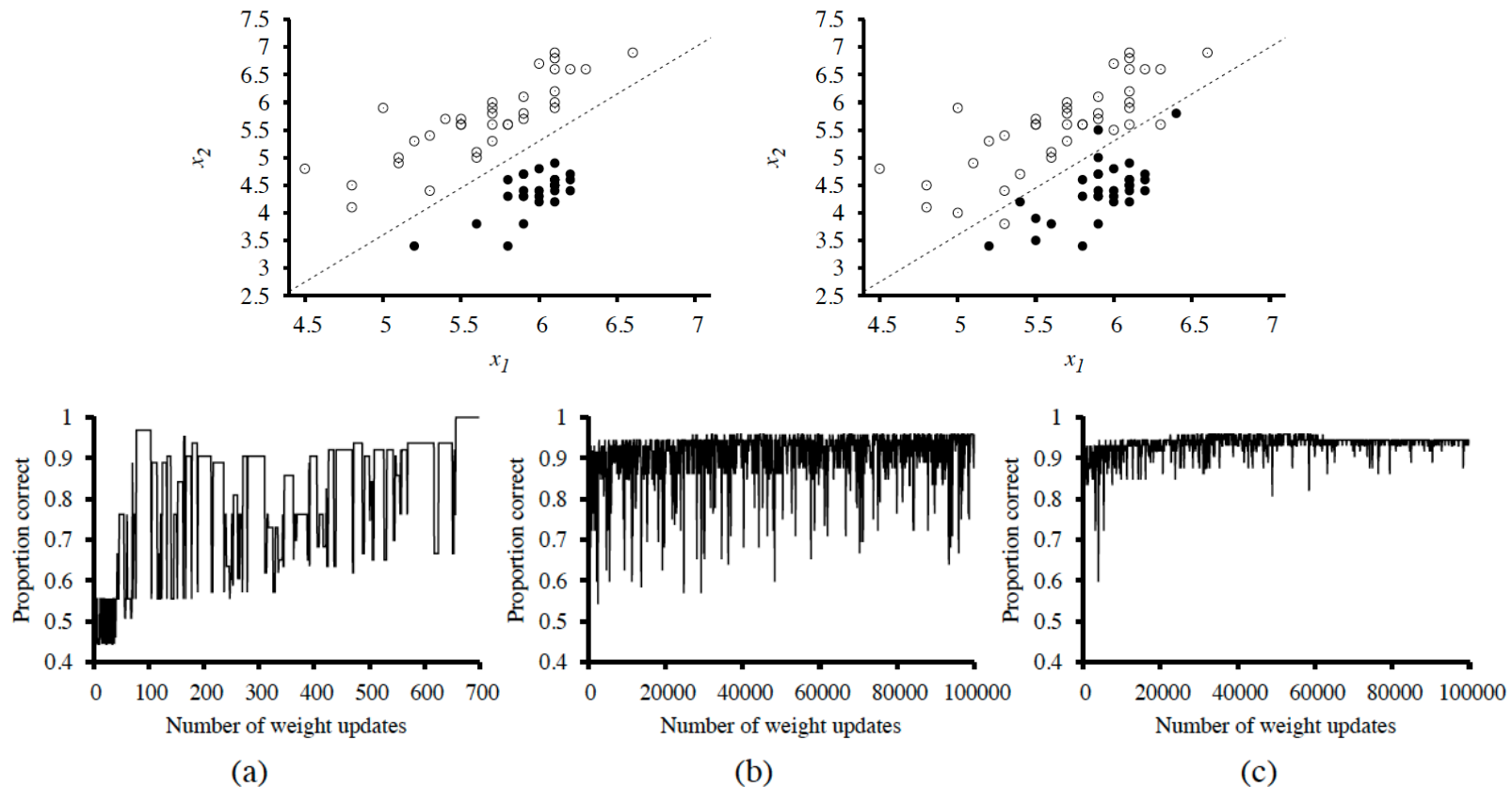
$$w_i \leftarrow w_i + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_i$$

- If the output is correct, i.e.,  $y = h_{\mathbf{w}}(\mathbf{x})$ , then the weights don't change
- If  $y = 1$  but  $h_{\mathbf{w}}(\mathbf{x}) = 0$ , then  $w_i$  is *increased* when  $x_i$  is positive and *decreased* when  $x_i$  is negative.
- If  $y = 0$  but  $h_{\mathbf{w}}(\mathbf{x}) = 1$ , then  $w_i$  is *decreased* when  $x_i$  is positive and *increased* when  $x_i$  is negative.

Perceptron Convergence Theorem: For any data set that's linearly separable and any training procedure that continues to present each training example, the learning rule is guaranteed to find a solution in a finite number of steps.

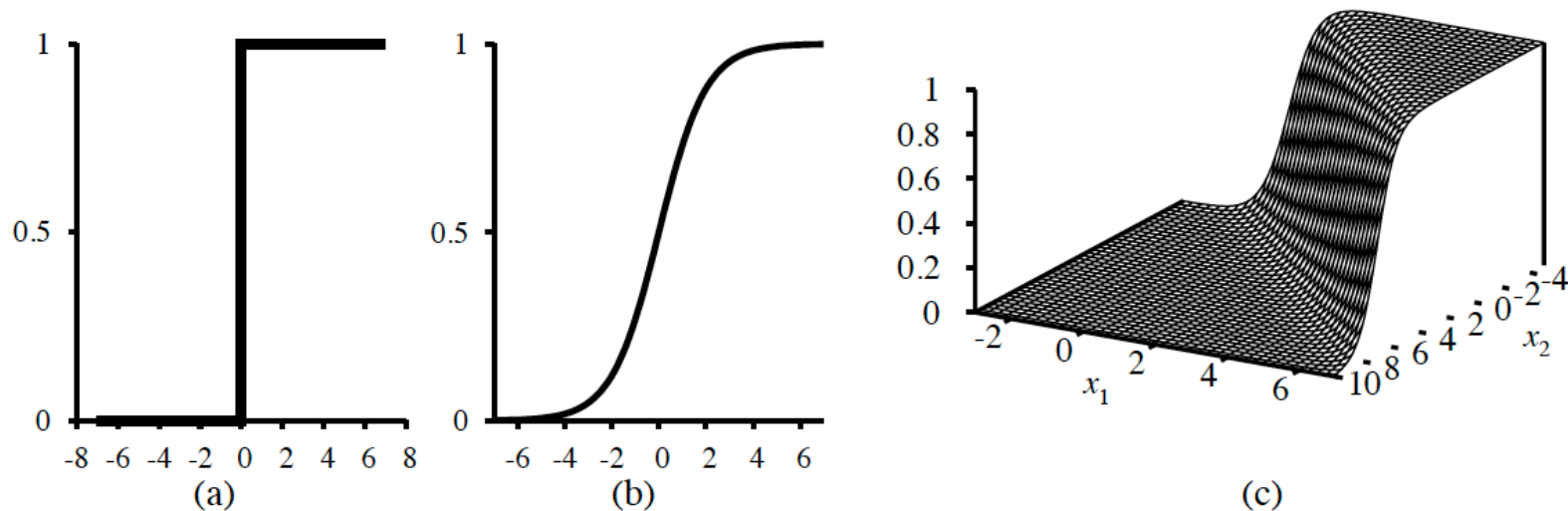


# Perceptron Performance



**Figure 18.16** FILES: . (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 18.14(a). (b) The same plot for the noisy, non-separable data in Figure 18.14(b); note the change in scale of the  $x$ -axis. (c) The same plot as in (b), with a learning rate schedule  $\alpha(t) = 1000/(1000 + t)$ .

# Linear Classification with Logistic Regression



**Figure 18.17** FILES: . (a) The hard threshold function  $Threshold(z)$  with 0/1 output. Note that the function is nondifferentiable at  $z = 0$ . (b) The logistic function,  $Logistic(z) = \frac{1}{1+e^{-z}}$ , also known as the sigmoid function. (c) Plot of a logistic regression hypothesis  $h_{\mathbf{w}}(\mathbf{x}) = \text{Logistic}(\mathbf{w} \cdot \mathbf{x})$  for the data shown in Figure 18.14(b).

An important function!

# Logistic Regression

---

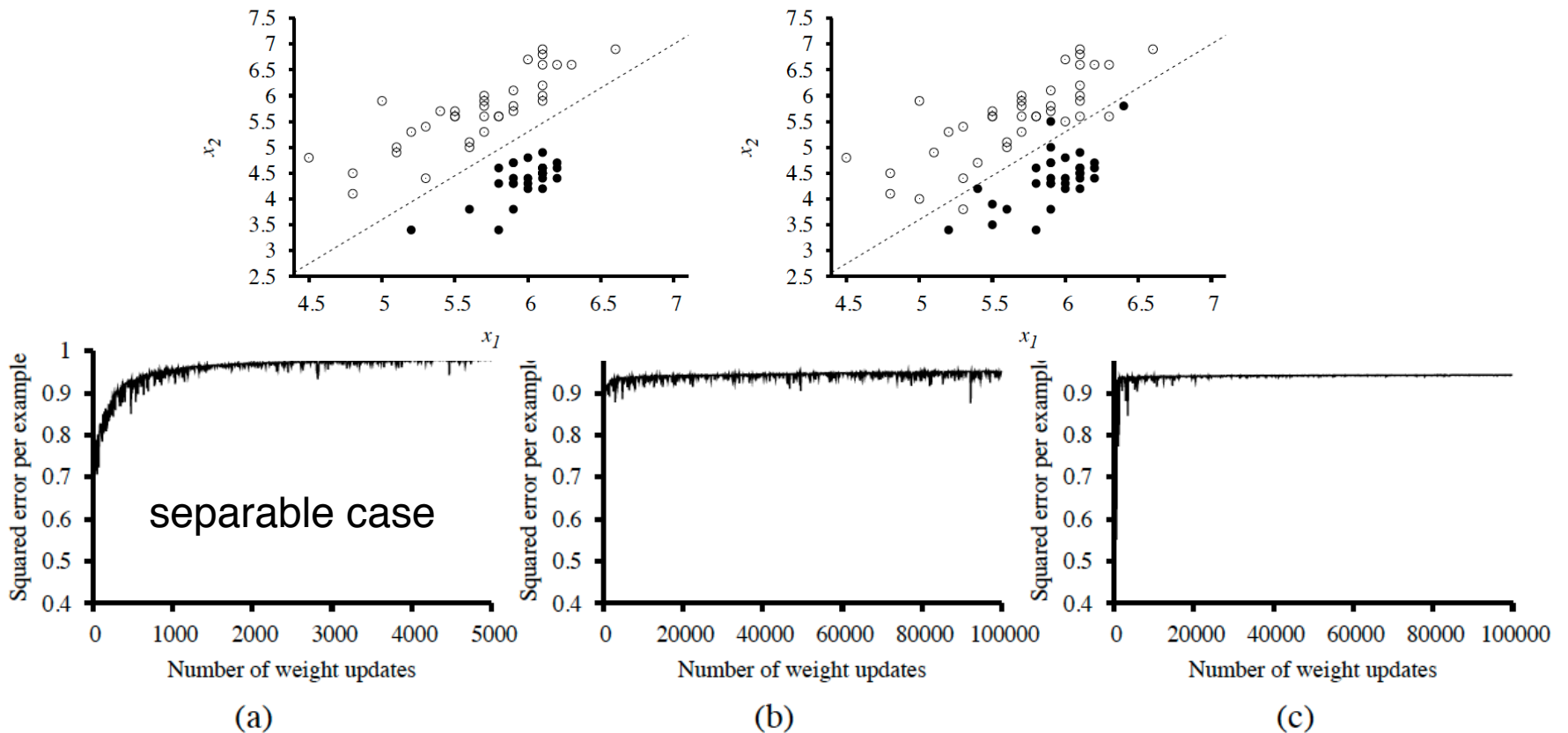
$$h_{\mathbf{w}}(\mathbf{x}) = \text{Logistic}(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

For a single sample  $(\mathbf{x}, y)$  and  $L_2$  loss function :

$$w_i \leftarrow w_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) \underbrace{h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))}_{\text{derivative of logistic function}} x_i$$

derivative of logistic function

# Logistic Regression Performance



**Figure 18.18** FILES: . Repeat of the experiments in Figure 18.15 using logistic regression and squared error. The plot in (a) covers 5000 iterations rather than 1000, while (b) and (c) use the same scale.

# Summary

---

- Learning from Examples: brief review
  - Loss functions
  - Generalization
  - Overfitting
  - Cross-validation
  - Regularization
- Univariate Linear Regression
  - Batch gradient descent
  - Stochastic gradient descent
- Multivariate Linear Regression
  - Regularization
- Linear Classifiers
  - Perceptron learning rule
- Logistic Regression

## Next Class

---

- Artificial Neural Networks, Nonparametric Models, & Support Vector Machines
- Secs. 18.7 – 18.9