Linear Regression and Classification

Outline

I. Line fitting and gradient descent

II. Multivariable linear regression

III. Linear classifiers

IV. Logistic regression

* Figures are from the textbook site.

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)$

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs

I. Linear Regression

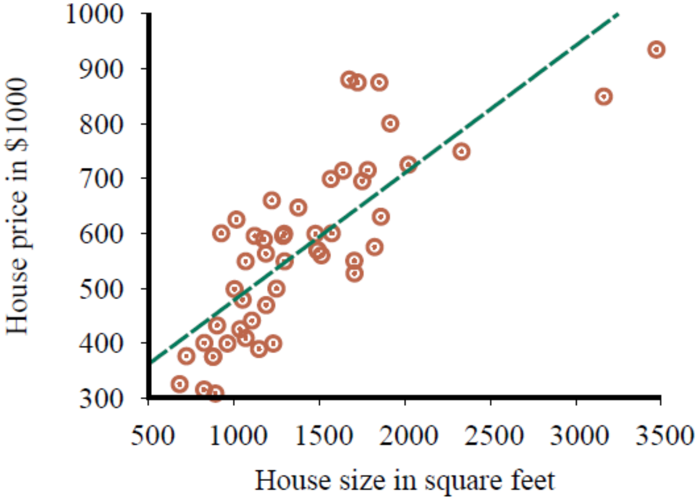
Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs outputs

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs outputs

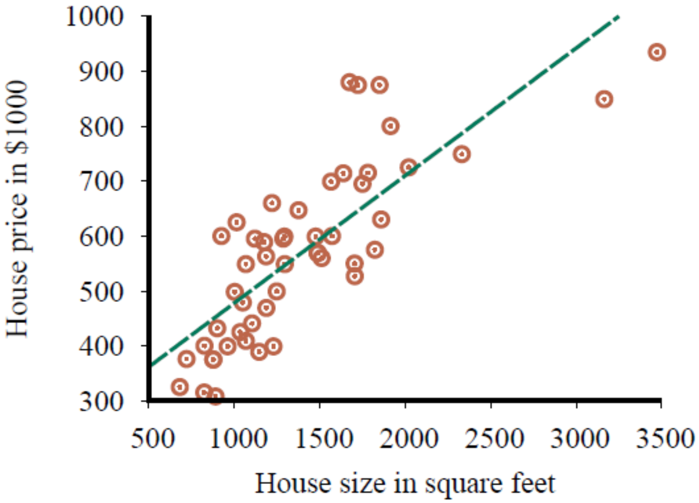


Houses for sale in Berkeley, CA

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs outputs



Houses for sale in Berkeley, CA

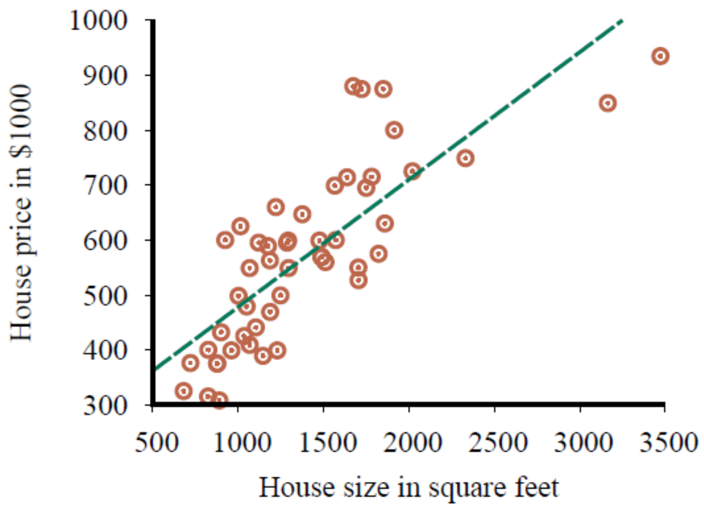
Hypothesis space: univariate linear functions.

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

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs outputs



Houses for sale in Berkeley, CA

Hypothesis space: univariate linear functions.

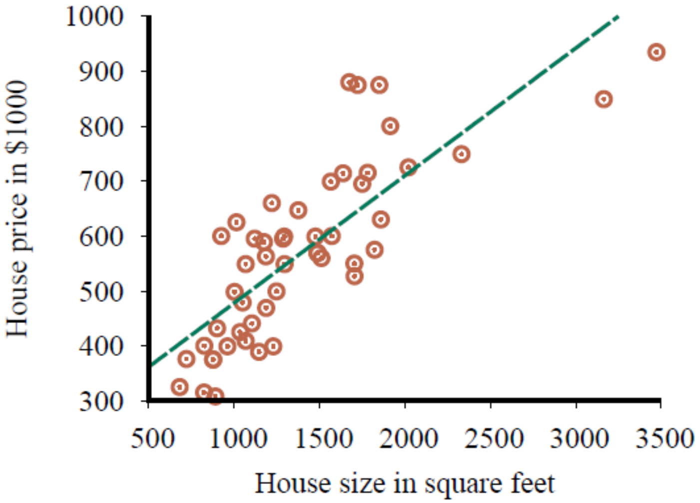
$$h_{\mathbf{w}}(x) \equiv w_1 x + w_0$$

(w_0, w_1)

I. Linear Regression

Data points: $(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)$

inputs outputs



Houses for sale in Berkeley, CA

Hypothesis space: univariate linear functions.

$$\boxed{h_{\boldsymbol{w}}(x) \equiv w_1 x + w_0}$$

(w_0, w_1)

Linear regression: Find the $h_{\boldsymbol{w}}$ that best fits the data.

Line Fitting

We find the weights (w_0, w_1) that minimizes the empirical loss.

Use the squared-error loss $L_2(y, h_{\boldsymbol{w}}) = (y - h_{\boldsymbol{w}})^2$, summed over all the points.

$$Loss(h_{\boldsymbol{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\boldsymbol{w}}(x_j))$$

$$= \sum_{j=1}^{N} \left(y_j - h_{\boldsymbol{w}}(x_j)\right)^2$$

Line Fitting

We find the weights (w_0, w_1) that minimizes the empirical loss.

Use the squared-error loss $L_2(y, h_{\mathbf{w}}) = (y - h_{\mathbf{w}})^2$, summed over all the points.

$$\boxed{Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j))}$$

$$= \sum_{j=1}^{N} \left(y_j - h_{\mathbf{w}}(x_j)\right)^2$$

$$= \sum_{j=1}^{N} \left(y_j - (w_1 x_j + w_0)\right)^2$$

Line Fitting

We find the weights (w_0, w_1) that minimizes the empirical loss.

Use the squared-error loss $L_2(y, h_{\boldsymbol{w}}) = (y - h_{\boldsymbol{w}})^2$, summed over all the points.

$$\mathit{Loss}(h_{\boldsymbol{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\boldsymbol{w}}(x_j))$$

$$= \sum_{j=1}^{N} \left(y_j - h_{\boldsymbol{w}}(x_j)\right)^2$$

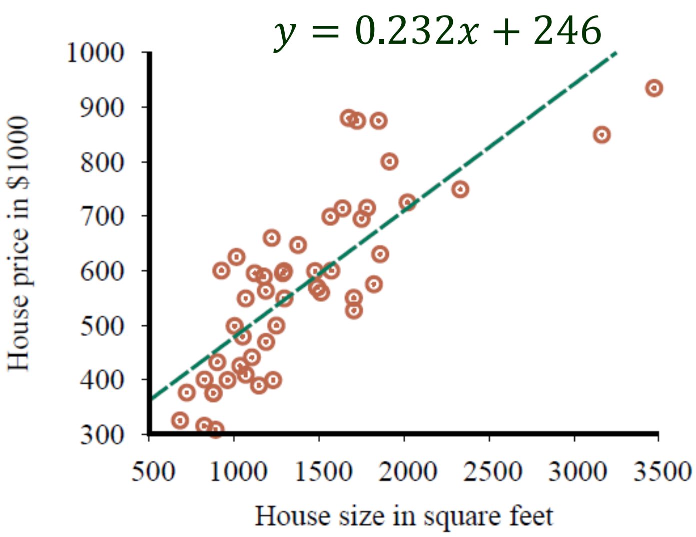
$$= \sum_{j=1}^{N} \left(y_j - (w_1 x_j + w_0)\right)^2$$

$$\boldsymbol{w}^* = \operatorname*{argmin}_{\boldsymbol{w}^*} \mathit{Loss}(h_{\boldsymbol{w}})$$

Vanishing of Partial Derivatives

At the minimizing $\boldsymbol{w}$, the gradient of $Loss(h_{\boldsymbol{w}})$ must vanish:

$$\nabla Loss(h_{\boldsymbol{w}}) = \left(\frac{\partial Loss}{\partial w_0}, \frac{\partial Loss}{\partial w_1}\right) = 0$$



Vanishing of Partial Derivatives

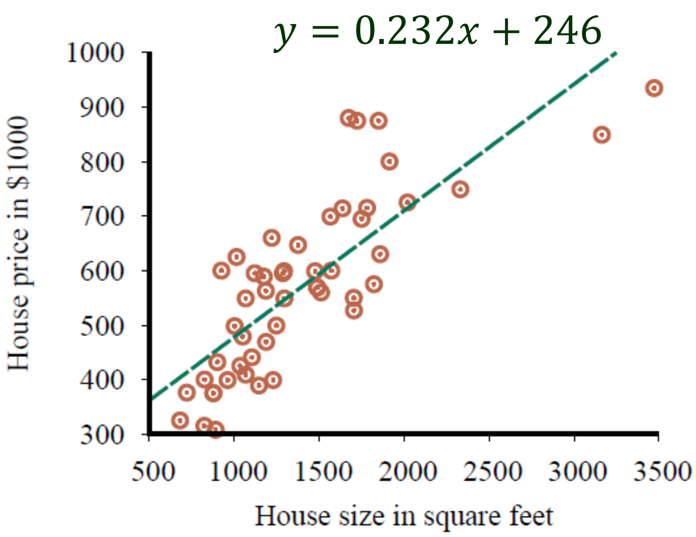
At the minimizing $\boldsymbol{w}$, the gradient of $Loss(h_{\boldsymbol{w}})$ must vanish:

$$\nabla Loss(h_{\boldsymbol{w}}) = \left(\frac{\partial Loss}{\partial w_0}, \frac{\partial Loss}{\partial w_1}\right) = 0$$

$$\Downarrow$$

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

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



Vanishing of Partial Derivatives

At the minimizing $\boldsymbol{w}$, the gradient of $Loss(h_{\boldsymbol{w}})$ must vanish:

$$\nabla Loss(h_{\boldsymbol{w}}) = \left(\frac{\partial Loss}{\partial w_0}, \frac{\partial Loss}{\partial w_1}\right) = 0$$

$\Downarrow$

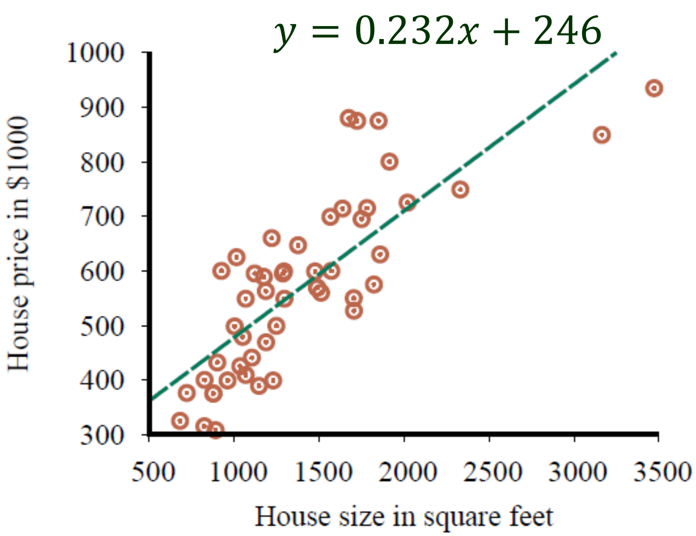
$$\frac{\partial Loss}{\partial w_0} = \frac{\partial}{\partial w_0}\sum_{j=1}^{N}\left(y_j - (w_1 x_j + w_0)\right)^2 = 0$$

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

$\Downarrow$

$$w_1 = \frac{N\sum_{j=1}^{N} x_j y_j - \left(\sum_{j=1}^{N} x_j\right)\cdot\left(\sum_{j=1}^{N} y_j\right)}{N\left(\sum_{j=1}^{N} x_j^2\right) - \left(\sum_{j=1}^{N} x_j\right)^2}$$

$$w_0 = \frac{1}{N}\left(\sum_{j=1}^{N} y_j - w_1\sum_{j=1}^{N} x_j\right)$$



Vanishing of Partial Derivatives

At the minimizing $\boldsymbol{w}$, the gradient of $Loss(h_{\boldsymbol{w}})$ must vanish:

$$\nabla Loss(h_{\boldsymbol{w}}) = \left(\frac{\partial Loss}{\partial w_0}, \frac{\partial Loss}{\partial w_1}\right) = 0$$

$$\Downarrow$$

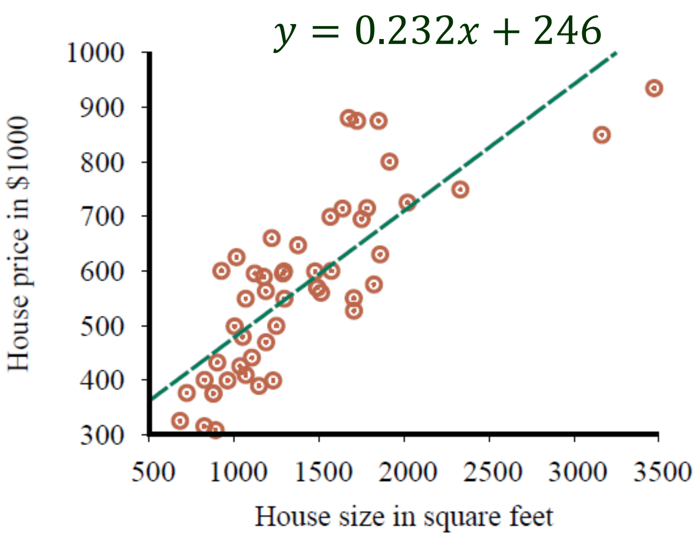
$$\frac{\partial Loss}{\partial w_0} = \frac{\partial}{\partial w_0}\sum_{j=1}^{N}\left(y_j - (w_1 x_j + w_0)\right)^2 = 0$$

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

$$\Downarrow$$

$$w_1 = \frac{N\sum_{j=1}^{N} x_j y_j - \left(\sum_{j=1}^{N} x_j\right)\cdot\left(\sum_{j=1}^{N} y_j\right)}{N\left(\sum_{j=1}^{N} x_j^2\right) - \left(\sum_{j=1}^{N} x_j\right)^2}$$

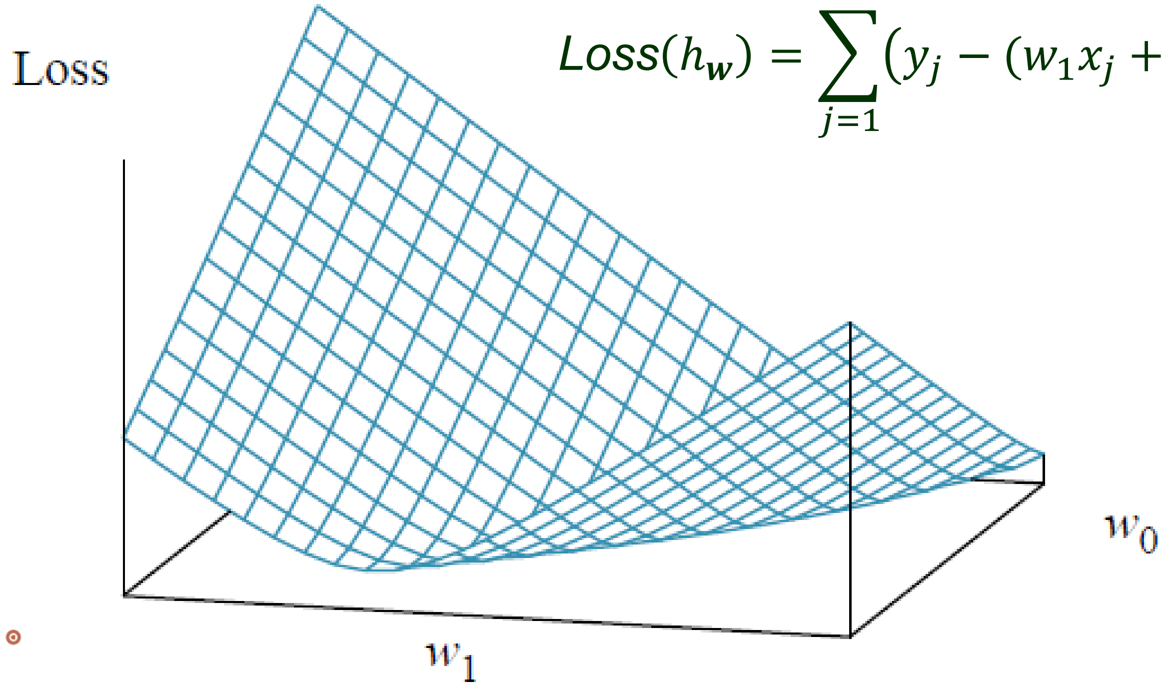
$$w_0 = \frac{1}{N}\left(\sum_{j=1}^{N} y_j - w_1\sum_{j=1}^{N} x_j\right)$$

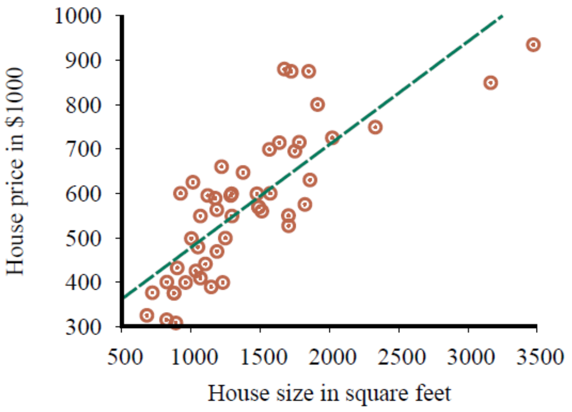


Note: the best-fit line does not minimize the sum of squares of distances of the data points to the line. It is inferior to a method used in computer vision for the purpose of extracting edges from an image. One reason is that the model cannot represent a vertical line.

Plot of the Loss Function

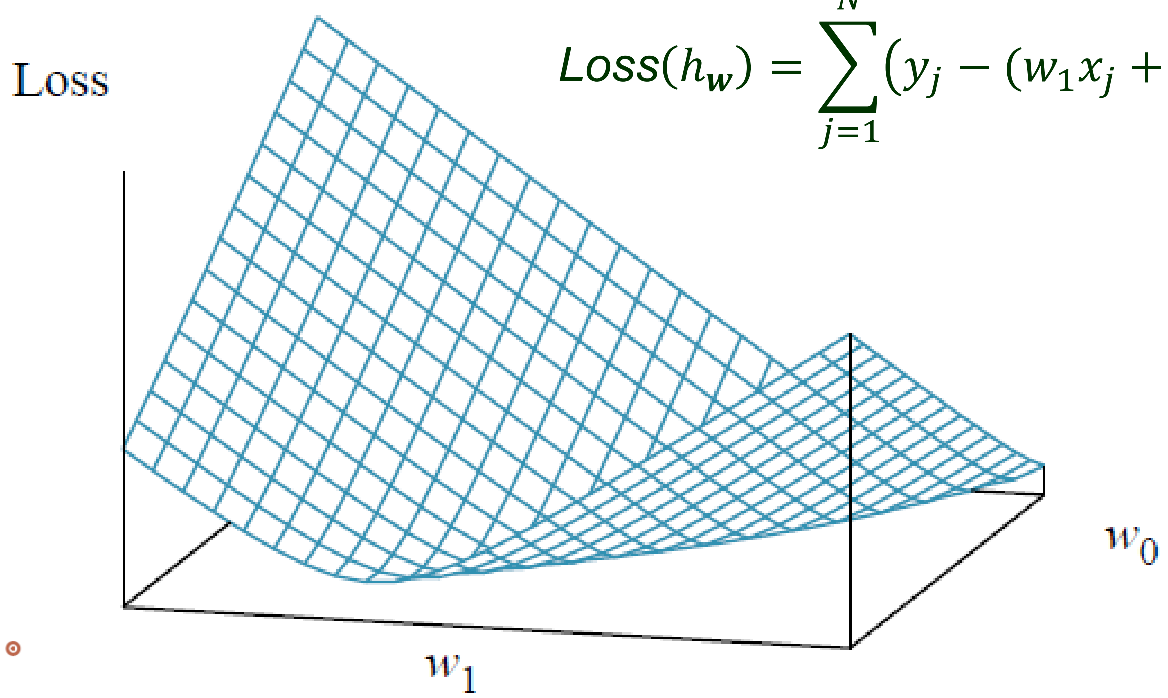
$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} \left(y_j - (w_1 x_j + w_0)\right)^2$$

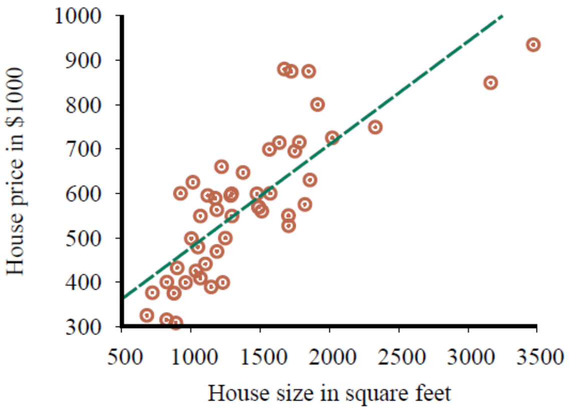




Plot of the Loss Function

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





- Convex function with no local minima.

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

Gradient Descent

- ♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.
- ♦ Instead, the method of gradient descent is used:
  - Start at a point $\boldsymbol{w}$ in the weight space.

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

- Start at a point $\boldsymbol{w}$ in the weight space.
- Compute an estimate of the gradient of the loss function.

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

- Start at a point $\boldsymbol{w}$ in the weight space.
- Compute an estimate of the gradient of the loss function.
- Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.

Gradient Descent

- ♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.
- ♦ Instead, the method of gradient descent is used:
  - Start at a point $\boldsymbol{w}$ in the weight space.
  - Compute an estimate of the gradient of the loss function.
  - Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.
  - Repeat until convergence on a point with (local) minimum loss.

Gradient Descent

- For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.
- Instead, the method of gradient descent is used:
  - Start at a point $\boldsymbol{w}$ in the weight space.
  - Compute an estimate of the gradient of the loss function.
  - Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.
  - Repeat until convergence on a point with (local) minimum loss.

$\boldsymbol{w} \leftarrow$ any point in the parameter space
while not converged do
    for each w_i in $\boldsymbol{w}$ do

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

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

- Start at a point $\boldsymbol{w}$ in the weight space.
- Compute an estimate of the gradient of the loss function.
- Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.
- Repeat until convergence on a point with (local) minimum loss.

$\boldsymbol{w} \leftarrow$ any point in the parameter space
while not converged do
    for each w_i in $\boldsymbol{w}$ do

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

step size or *learning rate*

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

- Start at a point $\boldsymbol{w}$ in the weight space.
- Compute an estimate of the gradient of the loss function.
- Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.
- Repeat until convergence on a point with (local) minimum loss.

$\boldsymbol{w} \leftarrow$ any point in the parameter space
while not converged do
  for each w_i in $\boldsymbol{w}$ do

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

step size or *learning rate*

* Section 19.6.2 applies gradient descent to a quadratic loss function, which defeats the purpose since the gradient $\nabla Loss(h_{\boldsymbol{w}})$ is linear in $\boldsymbol{w}$ whose values can be easily determined from solving the linear system $\nabla Loss(h_{\boldsymbol{w}}) = 0$.

Gradient Descent

♠ For a complex loss function, vanishing of its gradient often results in a system of nonlinear equations in $\boldsymbol{w}$ that does not have a closed-form solution.

♦ Instead, the method of gradient descent is used:

- Start at a point $\boldsymbol{w}$ in the weight space.
- Compute an estimate of the gradient of the loss function.
- Move a small amount in the direction of the negative gradient, i.e., the steepest downhill direction.
- Repeat until convergence on a point with (local) minimum loss.

$\boldsymbol{w} \leftarrow$ any point in the parameter space
while not converged do
  for each w_i in $\boldsymbol{w}$ do

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

step size or *learning rate*

* Section 19.6.2 applies gradient descent to a quadratic loss function, which defeats the purpose since the gradient $\nabla Loss(h_{\boldsymbol{w}})$ is linear in $\boldsymbol{w}$ whose values can be easily determined from solving the linear system $\nabla Loss(h_{\boldsymbol{w}}) = 0$.

** To see how gradient descent works, see Section 4 of http://web.cs.iastate.edu/~cs577/handouts/nonlinear-program.pdf.

Multivariable Linear Regression

- An example is represented by an n-vector $\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,n})$.
- Hypothesis space:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

Multivariable Linear Regression

- An example is represented by an n-vector $\boldsymbol{x}_j = (x_{j,1}, \dots, x_{j,n})$.
- Hypothesis space:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

For convenience, we extend $\boldsymbol{x}$ by adding $x_0 = 1$ such that $\boldsymbol{x} = (1, x_1, \dots, x_n)$.

Multivariable Linear Regression

- An example is represented by an n-vector $\boldsymbol{x}_j = (x_{j,1}, \ldots, x_{j,n})$.
- Hypothesis space:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

For convenience, we extend $\boldsymbol{x}$ by adding $x_0 = 1$ such that $\boldsymbol{x} = (1, x_1, \ldots, x_n)$.

$$\Downarrow$$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

Multivariable Linear Regression

- An example is represented by an n-vector $\boldsymbol{x}_j = (x_{j,1}, \dots, x_{j,n})$.
- Hypothesis space:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = w_0 + w_1 x_1 + \cdots + w_n x_n = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$\Downarrow$$

For convenience, we extend $\boldsymbol{x}$ by adding $x_0 = 1$ such that $\boldsymbol{x} = (1, x_1, \dots, x_n)$.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{x}$$

- Best weight vector:

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmin}} \sum_j L_2(y_j, \boldsymbol{w} \cdot \boldsymbol{x}_j)$$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

$$0 = \nabla L(\boldsymbol{w}) = \nabla\big((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})\big)$$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

$$0 = \nabla L(\boldsymbol{w}) = \nabla\left((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})\right)$$

$$\Downarrow$$

$$\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} = 0$$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

$$0 = \nabla L(\boldsymbol{w}) = \nabla\big((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})\big)$$

$$\Downarrow$$

$$\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} = 0$$

$\boldsymbol{X}$ almost always has full rank since $m \gg n$

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \ldots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

$$0 = \nabla L(\boldsymbol{w}) = \nabla\big((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})\big)$$

$$\Downarrow$$

$$\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} = 0$$

$\Downarrow$ $\boldsymbol{X}$ almost always has full rank since $m \gg n$

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

Optimal Weights

- Write $\boldsymbol{w}$ as a column vector, i.e., $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$.
- Vector of m outputs: $\boldsymbol{y} = (y_1, y_2, \dots, y_m)^{\boldsymbol{T}}$.
- *Data matrix* $(m \times n)$: $\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix}$
- Predicted outputs: $\hat{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{w}$
- Loss over all the training data: $L(\boldsymbol{w}) = \| \hat{\boldsymbol{y}} - \boldsymbol{y} \|^2 = \| \boldsymbol{X}\boldsymbol{w} - \boldsymbol{y} \|^2$

$$0 = \nabla L(\boldsymbol{w}) = \nabla\left((\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})^T(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y})\right)$$

$$\Downarrow$$

$$\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y} = 0$$

$\Downarrow$ $\boldsymbol{X}$ almost always has full rank since $m \gg n$

$$\boxed{\boldsymbol{w}^* = \boldsymbol{w} = \underbrace{\left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\boldsymbol{T}}}_{\text{pseudoinverse of } \boldsymbol{X}}\boldsymbol{y}}$$

Regularization

Commonly applied on multivariable linear function to avoid overfitting.

$$\mathit{Cost}(h_{\mathbf{w}}) = \mathit{EmpLoss}(h_{\mathbf{w}}) + \lambda \mathit{Complexity}(h_{\mathbf{w}})$$

where

$$\mathit{Complexity}(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_{i=1}^{n} |w_i|^q$$

Regularization

Commonly applied on multivariable linear function to avoid overfitting.

$$Cost(h_{\mathbf{w}}) = EmpLoss(h_{\mathbf{w}}) + \lambda Complexity(h_{\mathbf{w}})$$

where

$$Complexity(h_{\mathbf{w}}) = L_q(\mathbf{w}) = \sum_{i=1}^{n} |w_i|^q$$

- L_1 $(q = 1)$ regularization tends to produce a sparse model (in which many weights are set to zero) because it takes the $w_0, w_1, \ldots, w_n$ axes seriously.

Regularization

Commonly applied on multivariable linear function to avoid overfitting.

$$\mathit{Cost}(h_{\boldsymbol{w}}) = \mathit{EmpLoss}(h_{\boldsymbol{w}}) + \lambda \mathit{Complexity}(h_{\boldsymbol{w}})$$

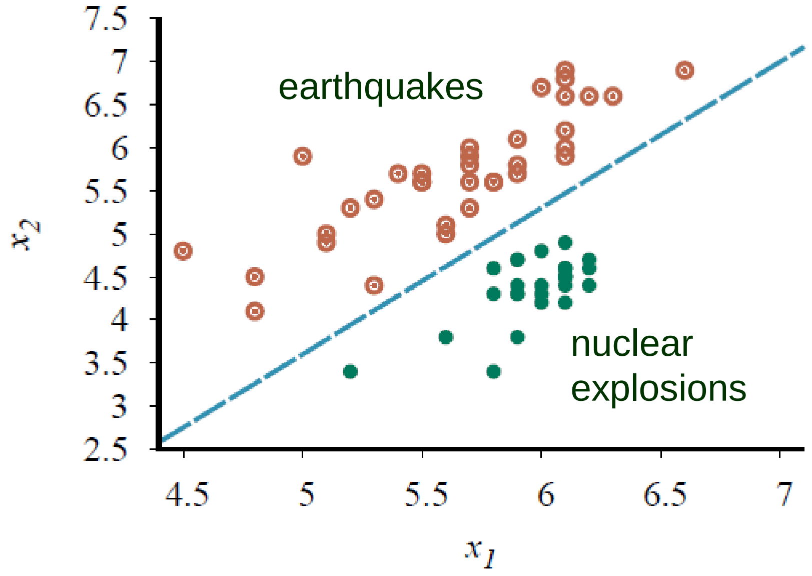
where

$$\mathit{Complexity}(h_{\boldsymbol{w}}) = L_q(\boldsymbol{w}) = \sum_{i=1}^{n} |w_i|^q$$

- ♦ L_1 $(q = 1)$ regularization tends to produce a sparse model (in which many weights are set to zero) because it takes the $w_0, w_1, \ldots, w_n$ axes seriously.

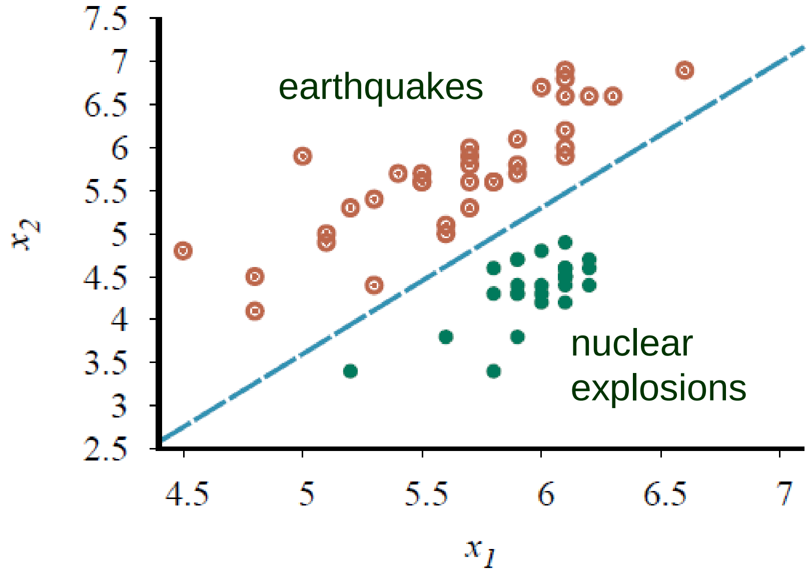
- ♠ L_2 $(q = 2)$ regularization takes the dimension axes arbitrarily.

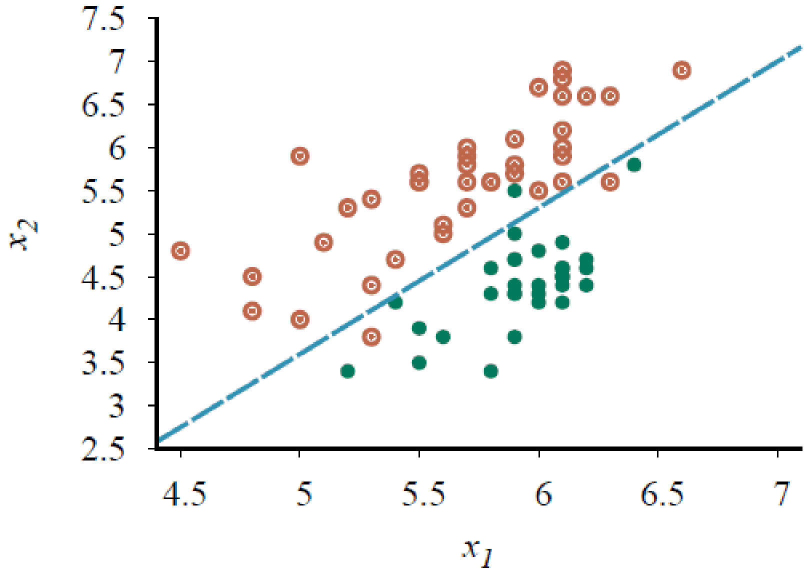
III. Linear Classifiers



Seismic data for earthquakes and nuclear explosions: x_1 and x_2 respectively refer to body and surface wave magnitudes computed from the seismic signal.

III. Linear Classifiers

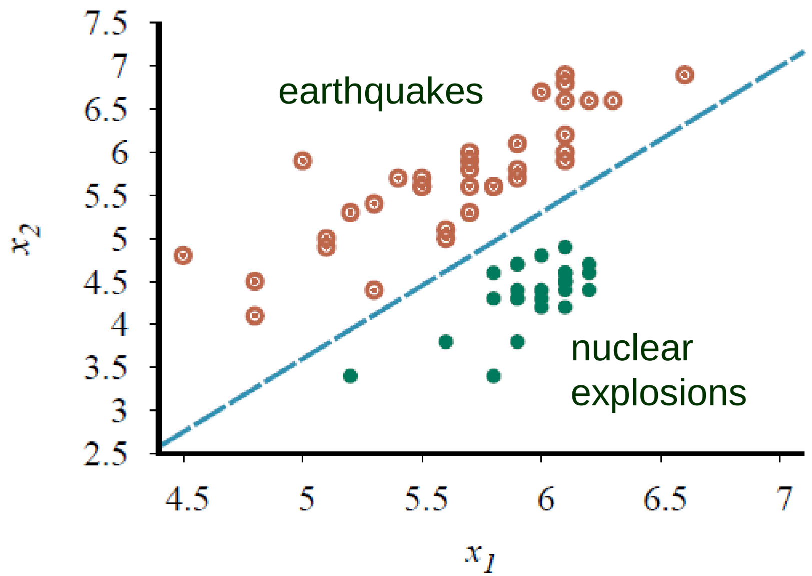




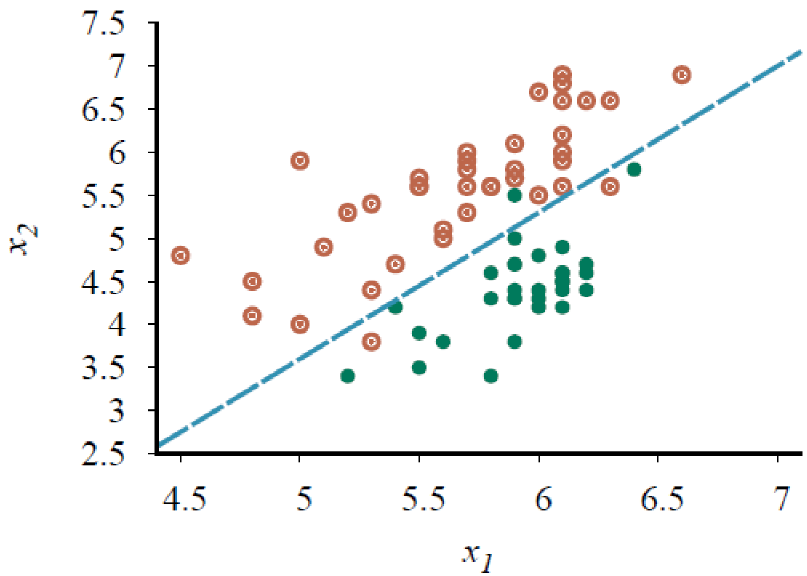
Seismic data for earthquakes and nuclear explosions: x_1 and x_2 respectively refer to body and surface wave magnitudes computed from the seismic signal.

Same domain with more data points.

III. Linear Classifiers



Seismic data for earthquakes and nuclear explosions: x_1 and x_2 respectively refer to body and surface wave magnitudes computed from the seismic signal.



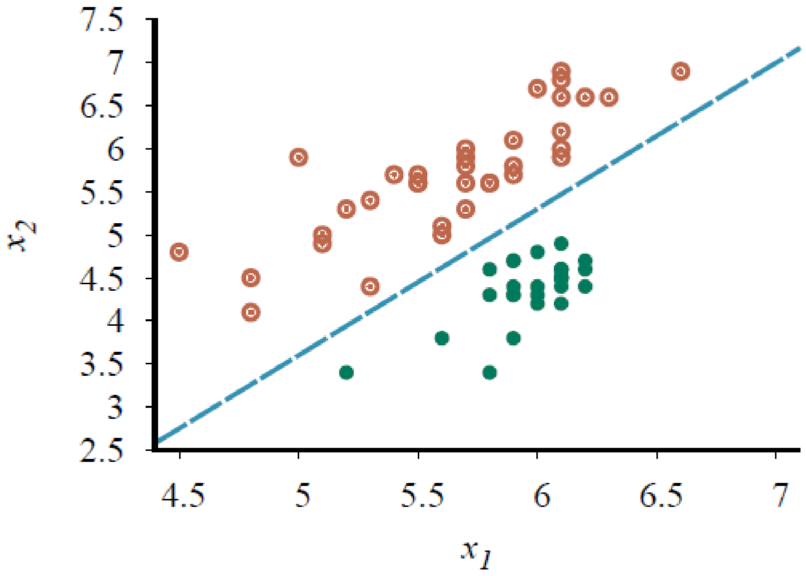
Same domain with more data points.

Task Learn a hypothesis that will take new (x_1, x_2) points and return 0 for earthquakes and 1 for explosions.

Linear Separator

A *decision boundary* is a line that separates two classes.

A *linear separator* is a linear decision boundary.

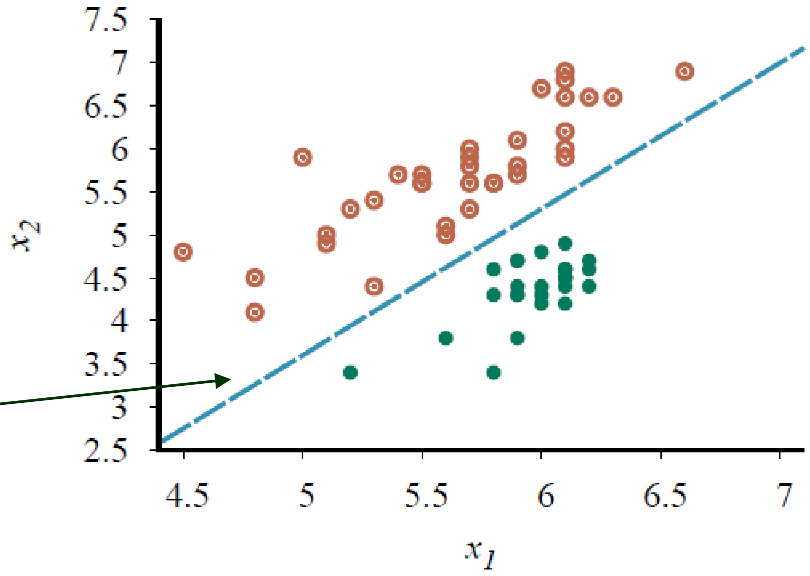


Linear Separator

A *decision boundary* is a line that separates two classes.

A *linear separator* is a linear decision boundary.

e.g., $-4.9 + 1.7x_1 - x_2 = 0$

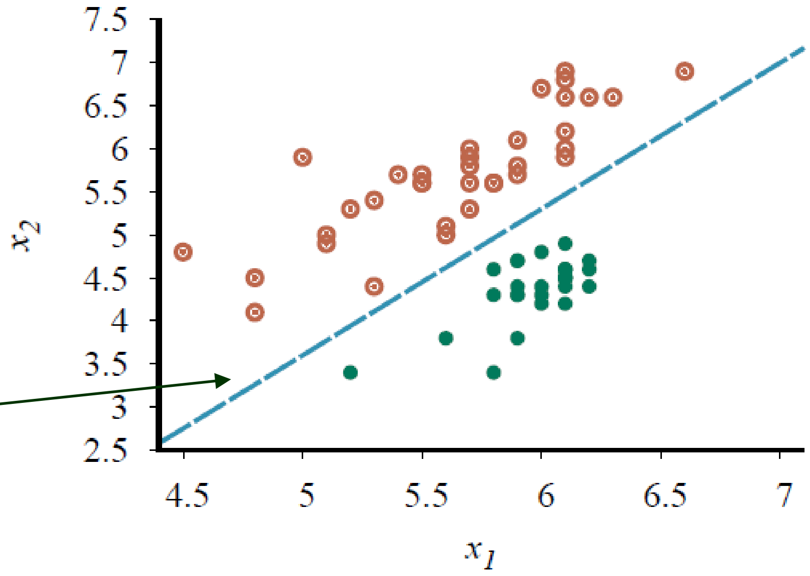


Linear Separator

A *decision boundary* is a line that separates two classes.

A *linear separator* is a linear decision boundary.

e.g., $-4.9 + 1.7x_1 - x_2 = 0$

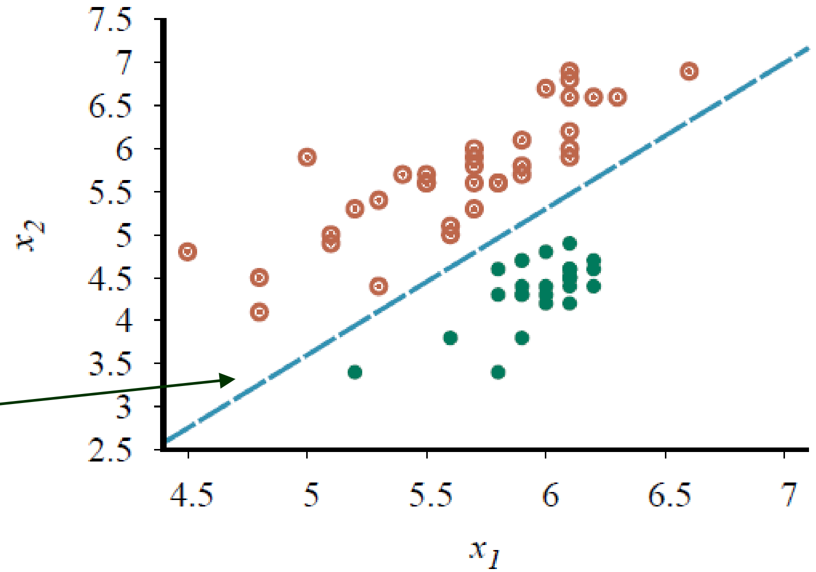


- $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\boldsymbol{T}}$
- $\boldsymbol{x} = (\underbrace{x_0}_{=1}, x_1, \ldots, x_n)$

Linear Separator

A *decision boundary* is a line that separates two classes.

A *linear separator* is a linear decision boundary.



e.g., $-4.9 + 1.7x_1 - x_2 = 0$

- $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\boldsymbol{T}}$
- $\boldsymbol{x} = (\underbrace{x_0}_{=1}, x_1, \ldots, x_n)$
- *Classification hypothesis*:

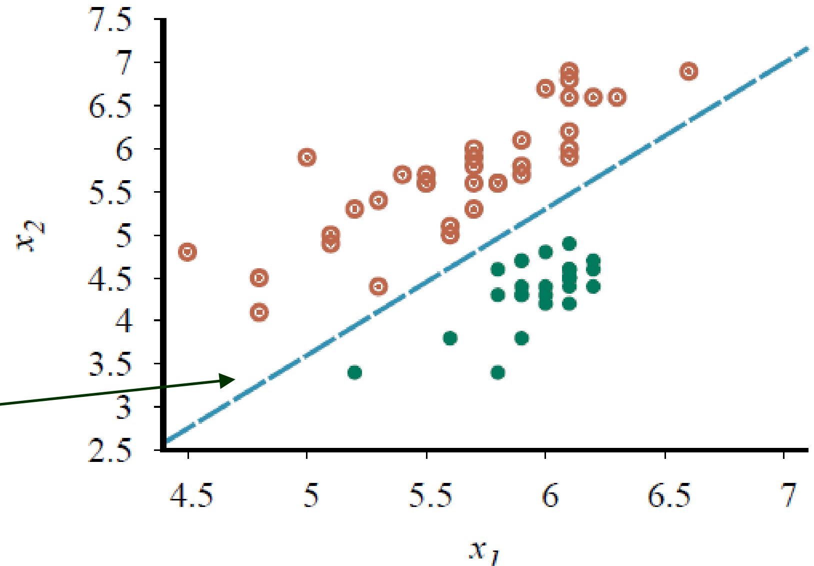
$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

Linear Separator

A *decision boundary* is a line that separates two classes.

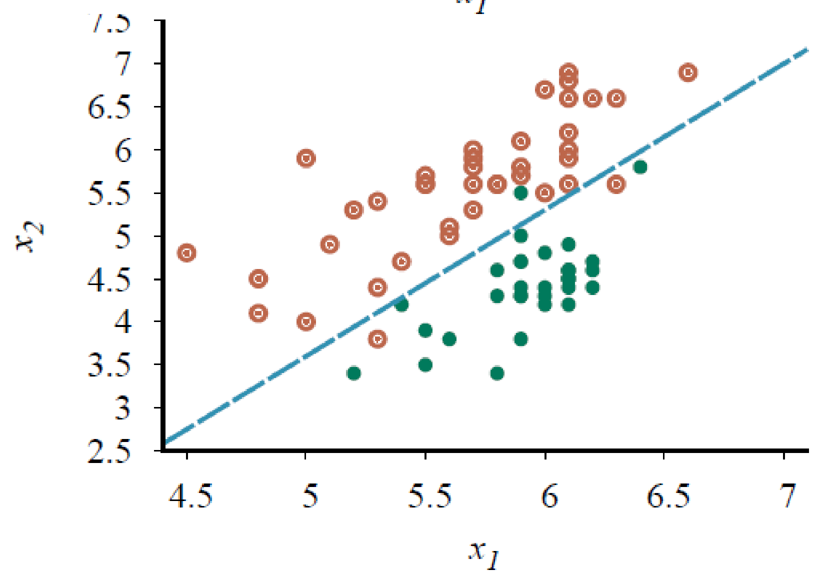
A *linear separator* is a linear decision boundary.

e.g., $-4.9 + 1.7x_1 - x_2 = 0$



- $\boldsymbol{w} = (w_0, w_1, \ldots, w_n)^{\boldsymbol{T}}$
- $\boldsymbol{x} = (\underset{= 1}{x_0}, x_1, \ldots, x_n)$
- *Classification hypothesis*:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

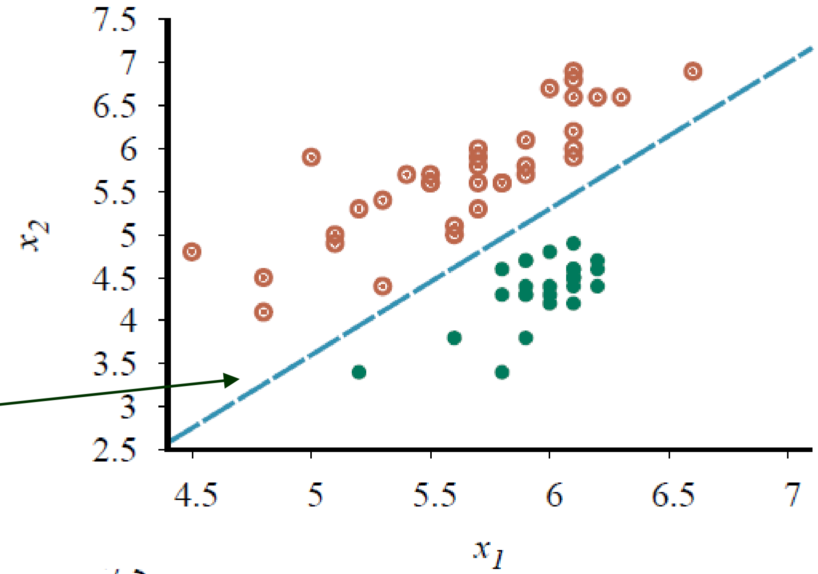


Linear Separator

A *decision boundary* is a line that separates two classes.

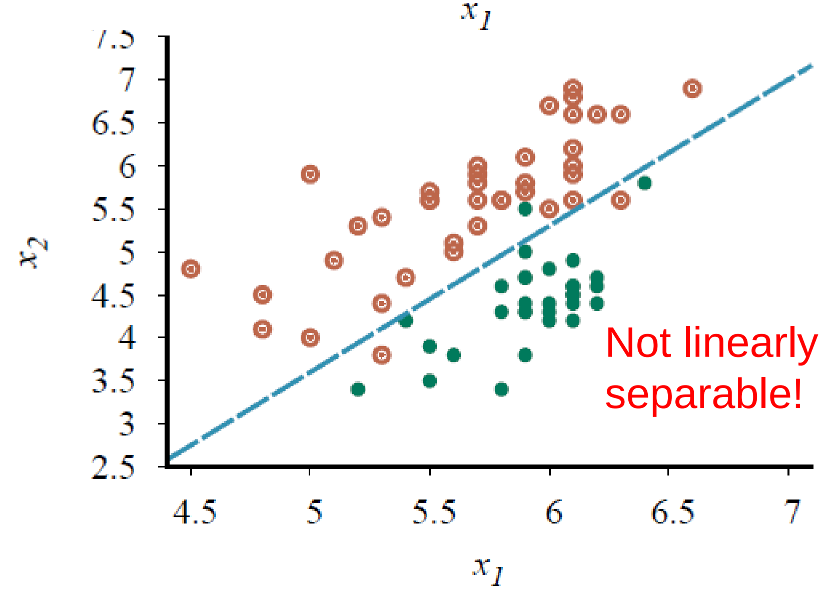
A *linear separator* is a linear decision boundary.

e.g., $-4.9 + 1.7x_1 - x_2 = 0$



- $\boldsymbol{w} = (w_0, w_1, \dots, w_n)^{\boldsymbol{T}}$
- $\boldsymbol{x} = (\underbrace{x_0}_{=1}, x_1, \dots, x_n)$
- *Classification hypothesis*:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$



Learning Rule

- ♠ Gradient $\nabla h_{\boldsymbol{w}}$ either vanishes or is undefined.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

Learning Rule

♠ Gradient $\nabla h_{\boldsymbol{w}}$ either vanishes or is undefined.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

♦ Use the *perceptron learning rule* (essentially borrowed from gradient descent):

$$w_i \leftarrow w_i + \alpha(y_j - h_{\boldsymbol{w}}(\boldsymbol{x}_j))x_{j,i}$$ on a single example $(\boldsymbol{x}_j, y)$

Learning Rule

♠ Gradient $\nabla h_{\boldsymbol{w}}$ either vanishes or is undefined.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

♦ Use the *perceptron learning rule* (essentially borrowed from gradient descent):

$$w_i \leftarrow w_i + \alpha(y_j - h_{\boldsymbol{w}}(\boldsymbol{x}_j))x_{j,i}$$ on a single example $(\boldsymbol{x}_j, y)$

- $y_j = h_{\boldsymbol{w}}(\boldsymbol{x}_j)$. The output is correct, so no change of weights.

Learning Rule

♠ Gradient $\nabla h_{\boldsymbol{w}}$ either vanishes or is undefined.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

♦ Use the *perceptron learning rule* (essentially borrowed from gradient descent):

$$w_i \leftarrow w_i + \alpha(y_j - h_{\boldsymbol{w}}(\boldsymbol{x}_j))x_{j,i}$$ on a single example $(\boldsymbol{x}_j, y)$

- $y_j = h_{\boldsymbol{w}}(\boldsymbol{x}_j)$. The output is correct, so no change of weights.
- $y_j = 1$ but $h_{\boldsymbol{w}}(\boldsymbol{x}_j) = 0$. w_i is increased if $x_{j,i} > 0$ and decreased if $x_{j,i} < 0$. In both situations, $\boldsymbol{xw}$ increases with the intention to output 1.

Learning Rule

- ♠ Gradient $\nabla h_{\boldsymbol{w}}$ either vanishes or is undefined.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$

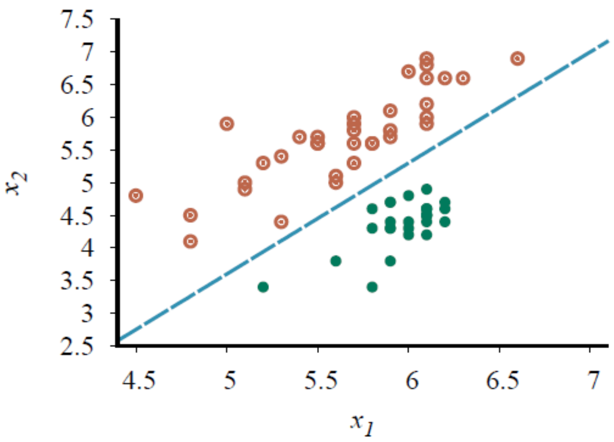
- ♦ Use the *perceptron learning rule* (essentially borrowed from gradient descent):

$$w_i \leftarrow w_i + \alpha(y_j - h_{\boldsymbol{w}}(\boldsymbol{x}_j))x_{j,i}$$ on a single example $(\boldsymbol{x}_j, y)$

- $y_j = h_{\boldsymbol{w}}(\boldsymbol{x}_j)$. The output is correct, so no change of weights.
- $y_j = 1$ but $h_{\boldsymbol{w}}(\boldsymbol{x}_j) = 0$. w_i is increased if $x_{j,i} > 0$ and decreased if $x_{j,i} < 0$. In both situations, $\boldsymbol{xw}$ increases with the intention to output 1.
- $y_j = 0$ but $h_{\boldsymbol{w}}(\boldsymbol{x}_j) = 1$. w_i is decreased if $x_{j,i} > 0$ and increased if $x_{j,i} < 0$. In both situations, $\boldsymbol{xw}$ decreases with the intention to output 0.

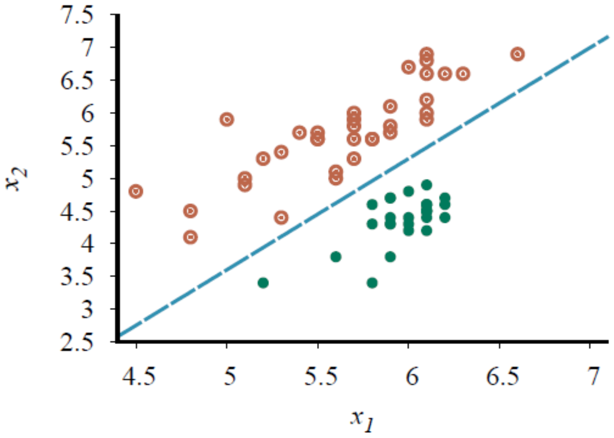
Training Curves for Perceptron Learning

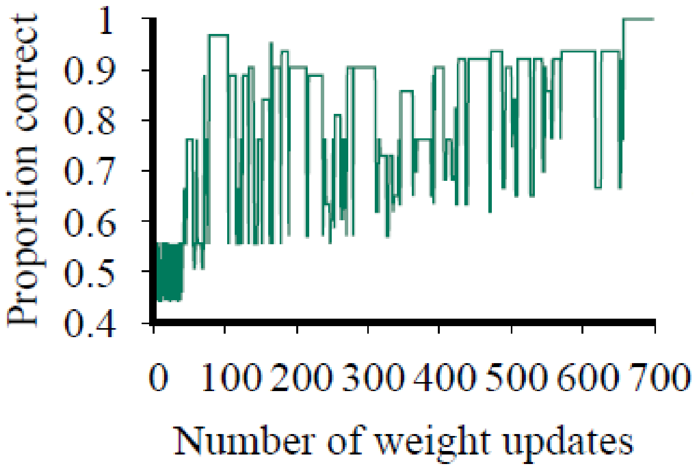
- The learning rule is applied one example at a time.
- A *training curve* measures the classifier performance on a fixed training set as learning proceeds one example at a time on the same set.



Training Curves for Perceptron Learning

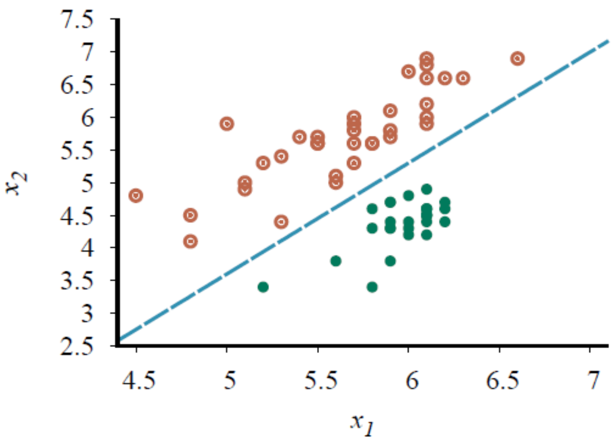
- The learning rule is applied one example at a time.
- A *training curve* measures the classifier performance on a fixed training set as learning proceeds one example at a time on the same set.

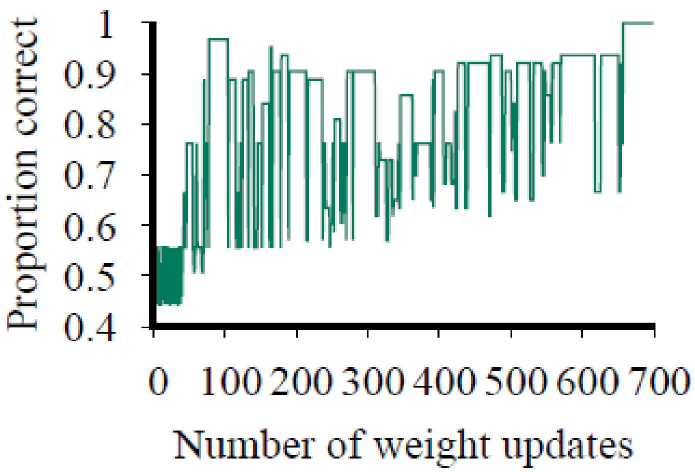




Training Curves for Perceptron Learning

- The learning rule is applied one example at a time.
- A *training curve* measures the classifier performance on a fixed training set as learning proceeds one example at a time on the same set.

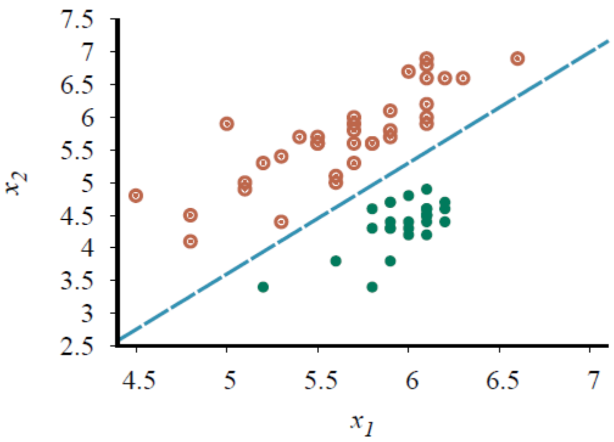


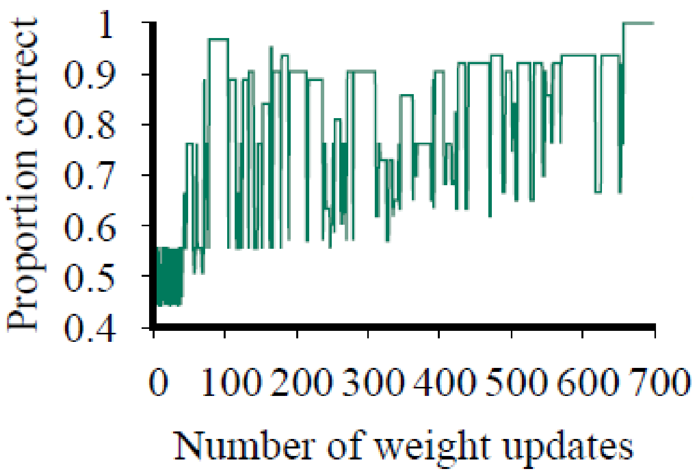


- 657 steps before convergence

Training Curves for Perceptron Learning

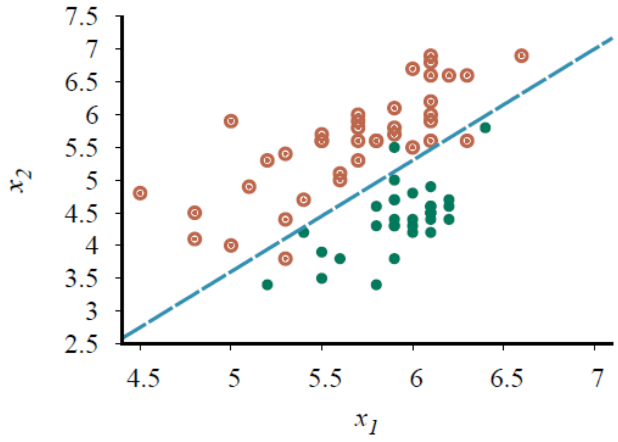
- The learning rule is applied one example at a time.
- A *training curve* measures the classifier performance on a fixed training set as learning proceeds one example at a time on the same set.



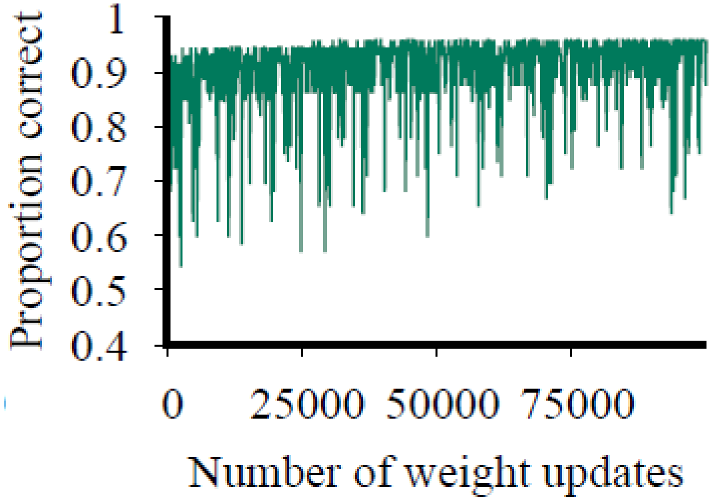


- 657 steps before convergence
- 63 examples, each used 10 times on average

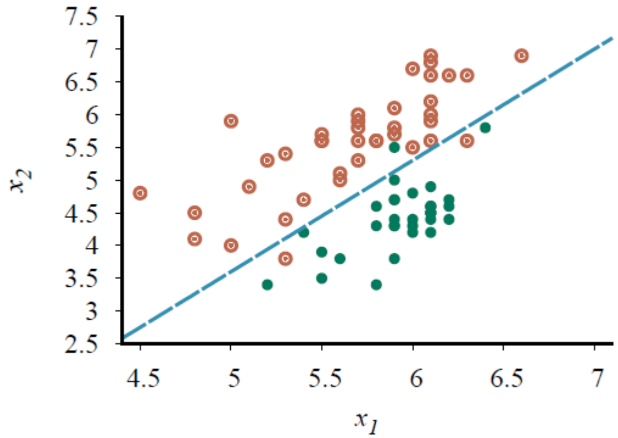
Training Curves (cont'd)



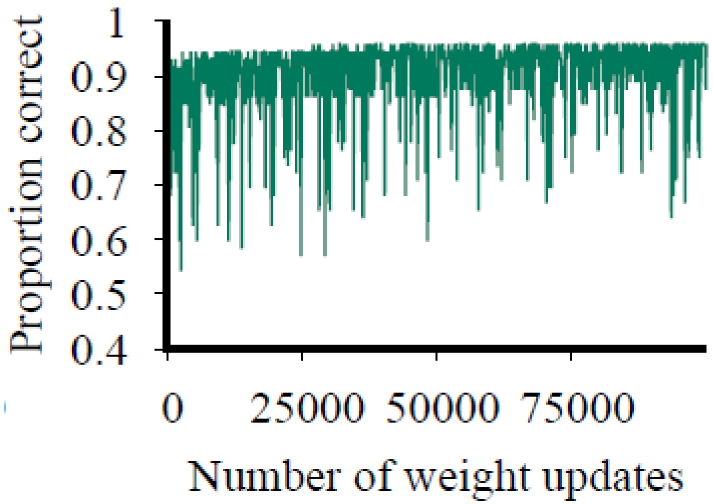
Data not linearly separable.



Training Curves (cont'd)

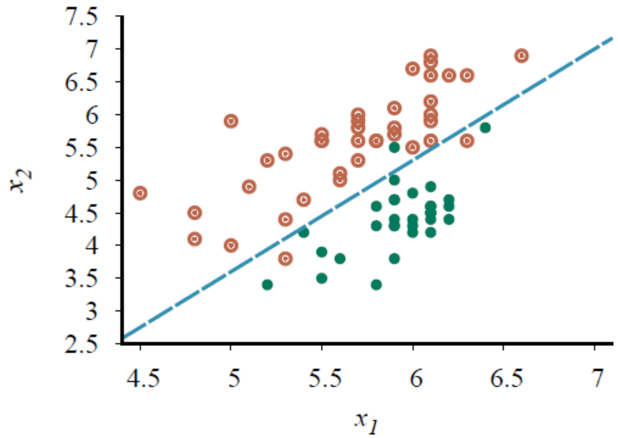


Data not linearly separable.

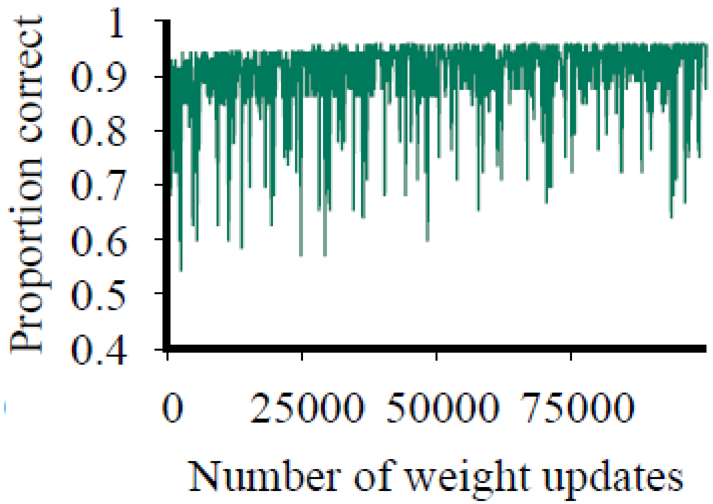


- Fails to converge after 10,000 steps.

Training Curves (cont'd)

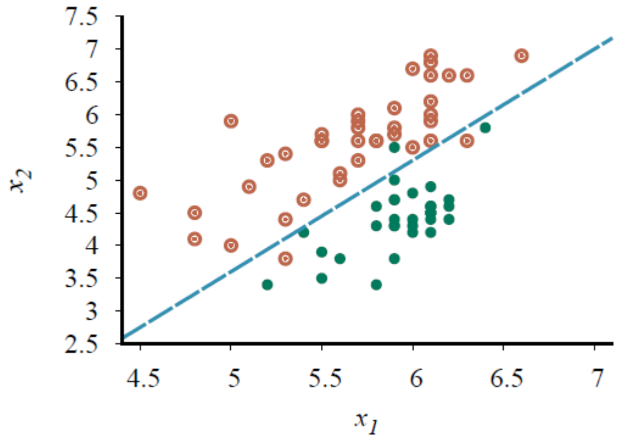


Data not linearly separable.

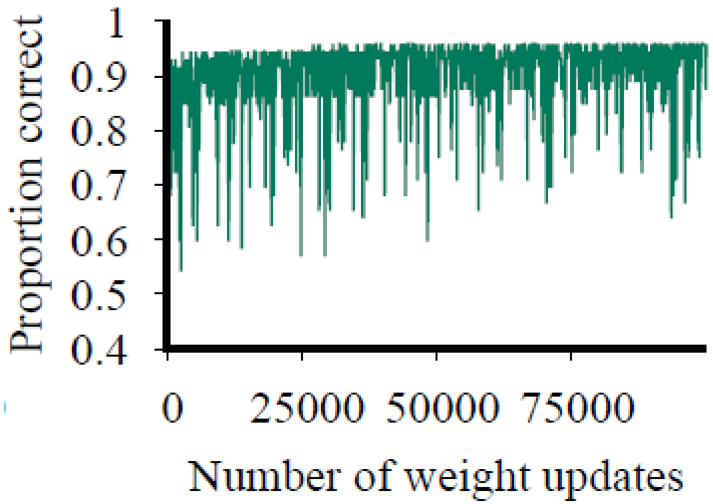


- Fails to converge after 10,000 steps.
- Let α decay as $O(1/t)$ where $t = \#$ iterations.

Training Curves (cont'd)

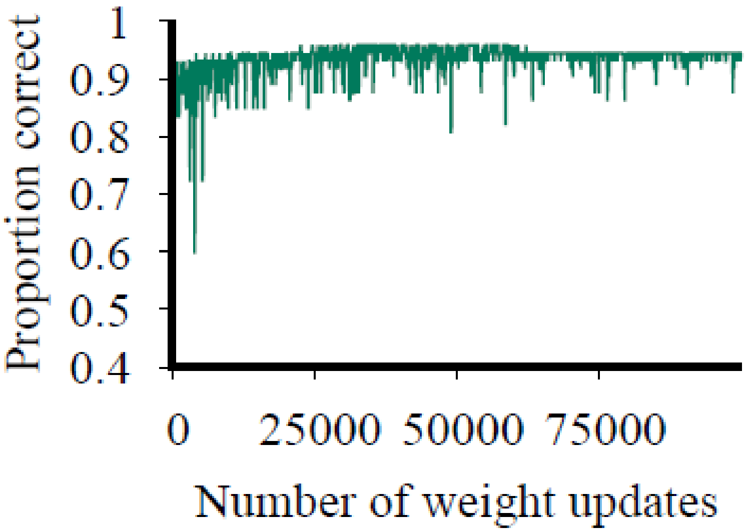


Data not linearly separable.



- Fails to converge after 10,000 steps.
- Let α decay as $O(1/t)$ where $t = \#$ iterations.

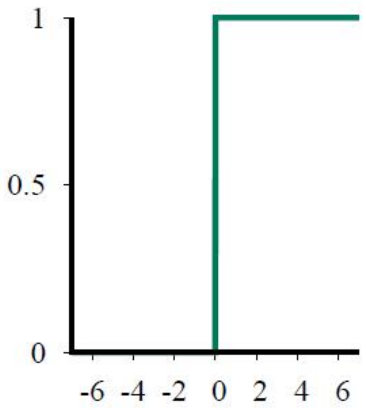
e.g., $\alpha(t) = 1000/(1000 + t)$



IV. Logistic Function

- Current hypothesis function is not continuous, let alone differentiable.
- This makes learning with the perceptron rule very unpredictable.
- It would be better if some examples could be classified as unclear borderline cases.

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$



IV. Logistic Function

♠ Current hypothesis function is not continuous, let alone differentiable.

♠ This makes learning with the perceptron rule very unpredictable.

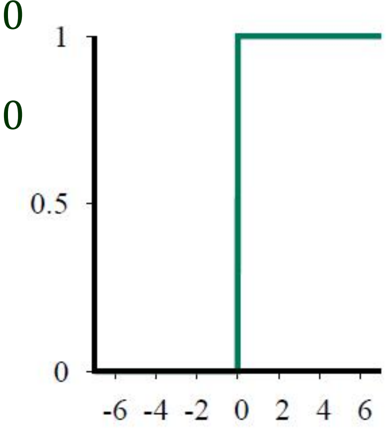
♠ It would be better if some examples could be classified as unclear borderline cases.

♦ Use a continuous, differential function to soften the threshold

Logistic function:

$$\mathit{Logistics}(z) = g(z) = \frac{1}{1+e^{-z}}$$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$



IV. Logistic Function

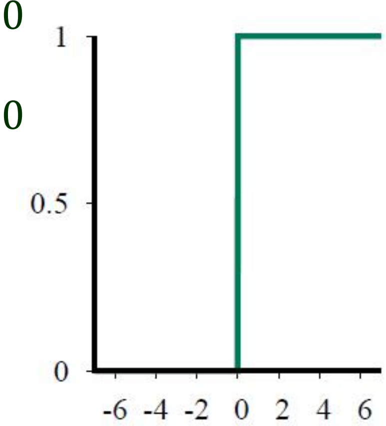
- Current hypothesis function is not continuous, let alone differentiable.
- This makes learning with the perceptron rule very unpredictable.
- It would be better if some examples could be classified as unclear borderline cases.

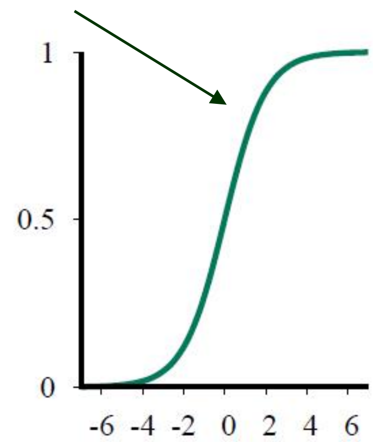
- Use a continuous, differential function to soften the threshold

Logistic function:

$$Logistics(z) = g(z) = \frac{1}{1+e^{-z}}$$

$$h_w(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$





IV. Logistic Function

- Current hypothesis function is not continuous, let alone differentiable.
- This makes learning with the perceptron rule very unpredictable.
- It would be better if some examples could be classified as unclear borderline cases.

- Use a continuous, differential function to soften the threshold

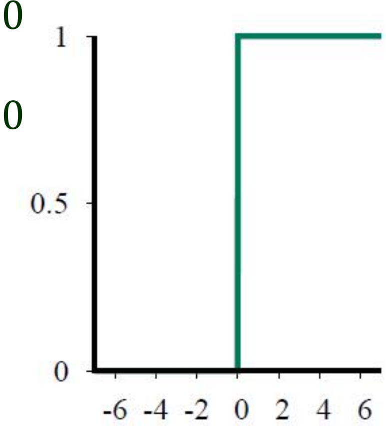
Logistic function:

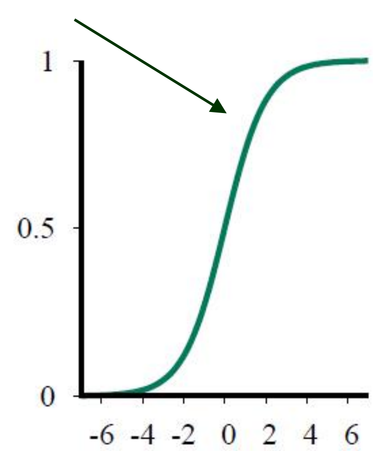
$$\mathit{Logistics}(z) = g(z) = \frac{1}{1+e^{-z}}$$

Hypothesis function:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{x}\boldsymbol{w}) = \frac{1}{1+e^{-\boldsymbol{x}\boldsymbol{w}}}$$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{x}\boldsymbol{w} \geq 0 \\ 0 \text{ if } \boldsymbol{x}\boldsymbol{w} < 0 \end{cases}$$





IV. Logistic Function

- Current hypothesis function is not continuous, let alone differentiable.
- This makes learning with the perceptron rule very unpredictable.
- It would be better if some examples could be classified as unclear borderline cases.

- Use a continuous, differential function to soften the threshold

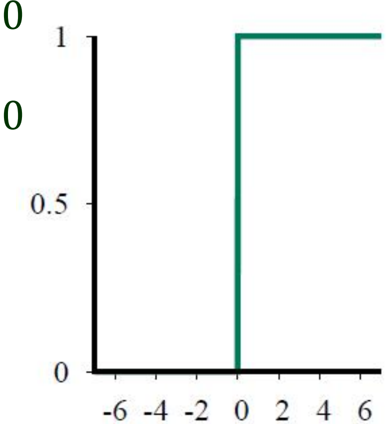
Logistic function:

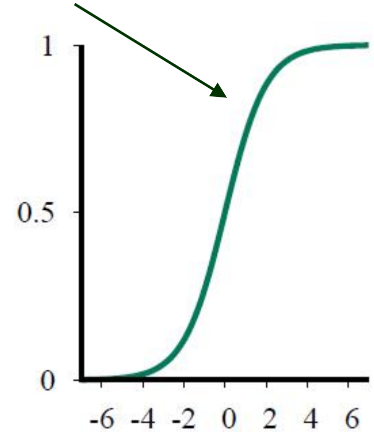
$$Logistics(z) = g(z) = \frac{1}{1+e^{-z}}$$

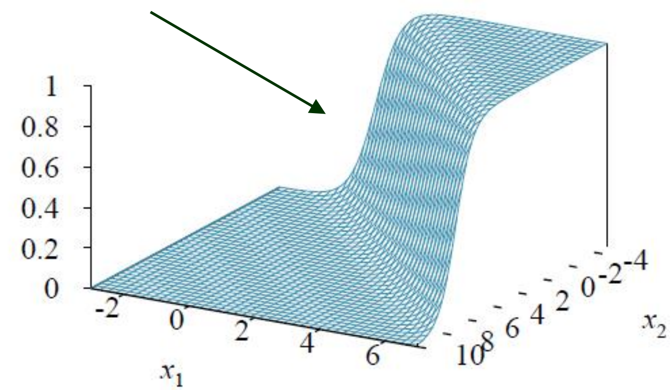
Hypothesis function:

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = Logistics(\boldsymbol{xw}) = \frac{1}{1+e^{-\boldsymbol{xw}}}$$

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \boldsymbol{xw} \geq 0 \\ 0 \text{ if } \boldsymbol{xw} < 0 \end{cases}$$







Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \textit{Logistics}(\boldsymbol{xw})$ to minimize loss on a data set.

$$\textit{Logistics}(z) = g(z) = \frac{1}{1 + e^{-z}}$$

Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{xw})$ to minimize loss on a data set.

Still apply gradient descent.

$$\mathit{Logistics}(z) = g(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial}{\partial w_i} \mathit{Loss}(\boldsymbol{w}) = \frac{\partial}{\partial w_i} \left(y - h_{\boldsymbol{w}}(\boldsymbol{x})\right)^2$$

Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{x}\boldsymbol{w})$ to minimize loss on a data set.

Still apply gradient descent.

$\mathit{Logistics}(z) = g(z) = \dfrac{1}{1 + e^{-z}}$

$$\frac{\partial}{\partial w_i} \mathit{Loss}(\boldsymbol{w}) = \frac{\partial}{\partial w_i} \left(y - h_{\boldsymbol{w}}(\boldsymbol{x})\right)^2$$
$$\vdots$$
$$= -2(y - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot g'(\boldsymbol{x}\boldsymbol{w}) \cdot x_i$$

Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{xw})$ to minimize loss on a data set.

Still apply gradient descent.

$$\mathit{Logistics}(z) = g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial}{\partial w_i} \mathit{Loss}(\boldsymbol{w}) = \frac{\partial}{\partial w_i}\left(y - h_{\boldsymbol{w}}(\boldsymbol{x})\right)^2$$

$$\vdots$$

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

$$\begin{aligned} g'(\boldsymbol{xw}) &= g(\boldsymbol{xw})(1 - g(\boldsymbol{xw})) \\ &= h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \end{aligned}$$

Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{x}\boldsymbol{w})$ to minimize loss on a data set.

Still apply gradient descent.

$$\mathit{Logistics}(z) = g(z) = \frac{1}{1+e^{-z}}$$

$$\frac{\partial}{\partial w_i}\mathit{Loss}(\boldsymbol{w}) = \frac{\partial}{\partial w_i}\left(y - h_{\boldsymbol{w}}(\boldsymbol{x})\right)^2$$

$$\vdots$$

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

$$g'(\boldsymbol{x}\boldsymbol{w}) = g(\boldsymbol{x}\boldsymbol{w})(1 - g(\boldsymbol{x}\boldsymbol{w}))$$

$$= h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}))$$

$$= -2(y - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot x_i$$

Logistic Regression

Fit the model $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathit{Logistics}(\boldsymbol{xw})$ to minimize loss on a data set.

Still apply gradient descent.

$$\mathit{Logistics}(z) = g(z) = \frac{1}{1 + e^{-z}}$$

$$\frac{\partial}{\partial w_i} \mathit{Loss}(\boldsymbol{w}) = \frac{\partial}{\partial w_i} \left(y - h_{\boldsymbol{w}}(\boldsymbol{x})\right)^2$$

$$\vdots$$

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

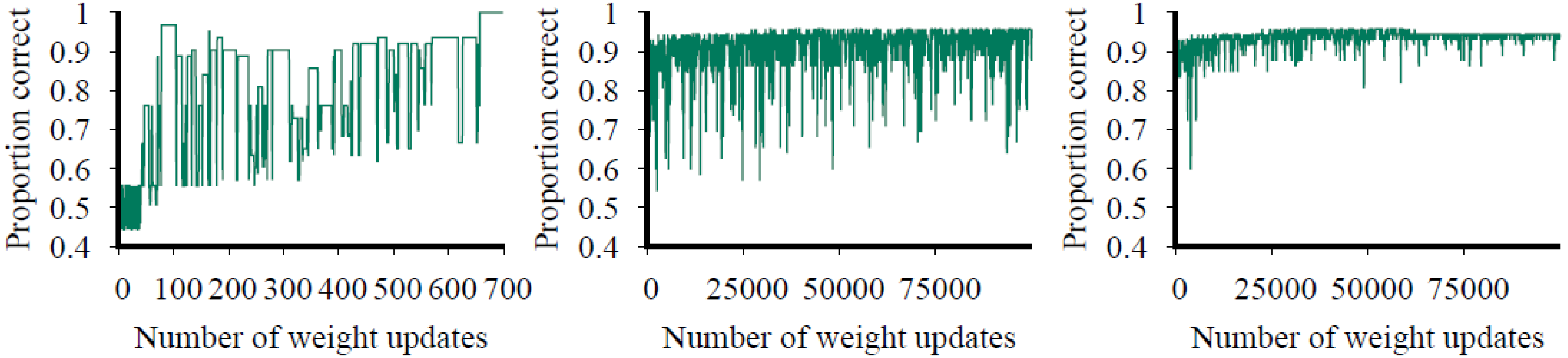
$$g'(\boldsymbol{xw}) = g(\boldsymbol{xw})(1 - g(\boldsymbol{xw}))$$
$$= h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x}))$$

$$= -2(y - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot h_{\boldsymbol{w}}(\boldsymbol{x})(1 - h_{\boldsymbol{w}}(\boldsymbol{x})) \cdot x_i$$

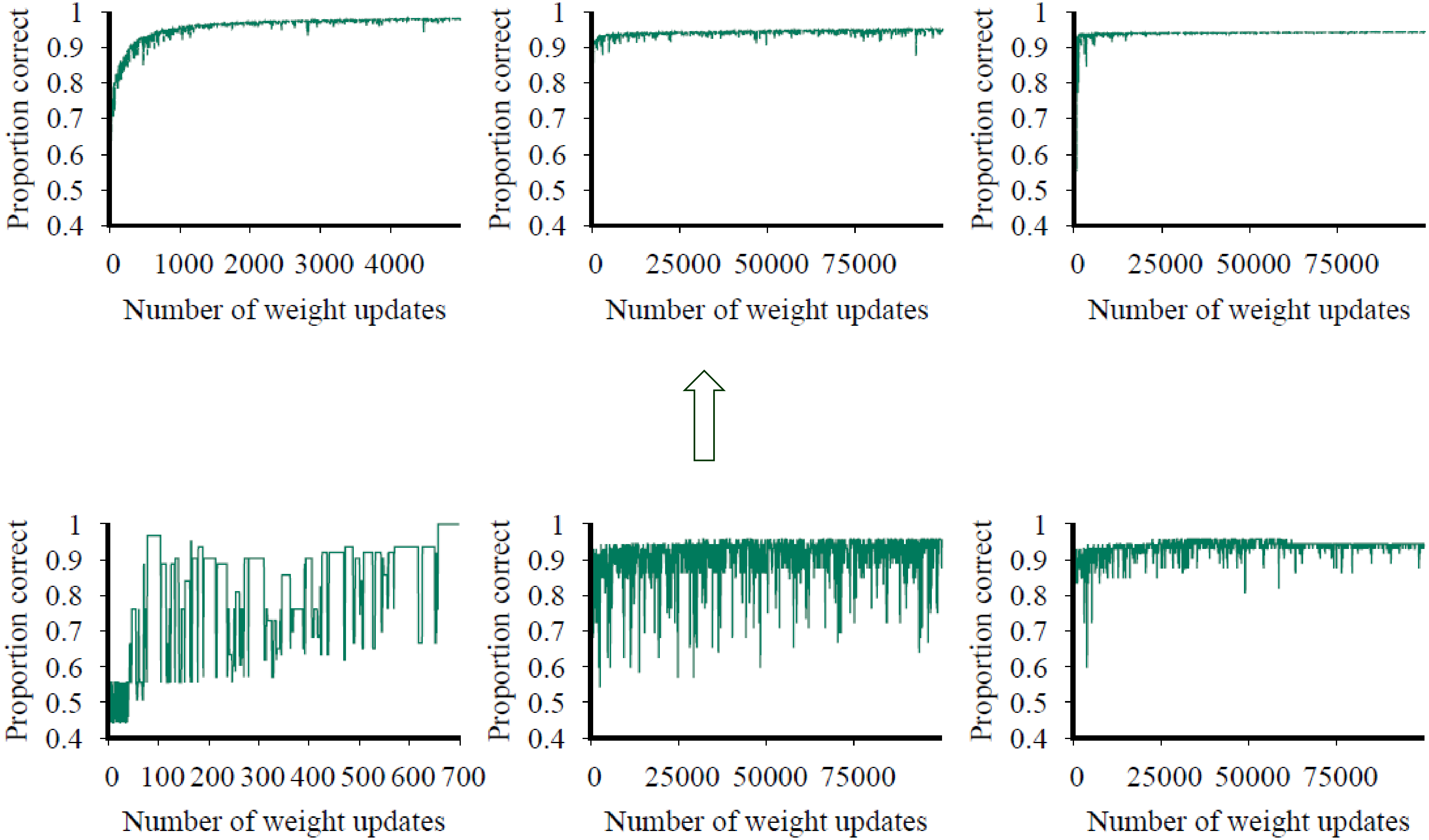
Weight update:

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

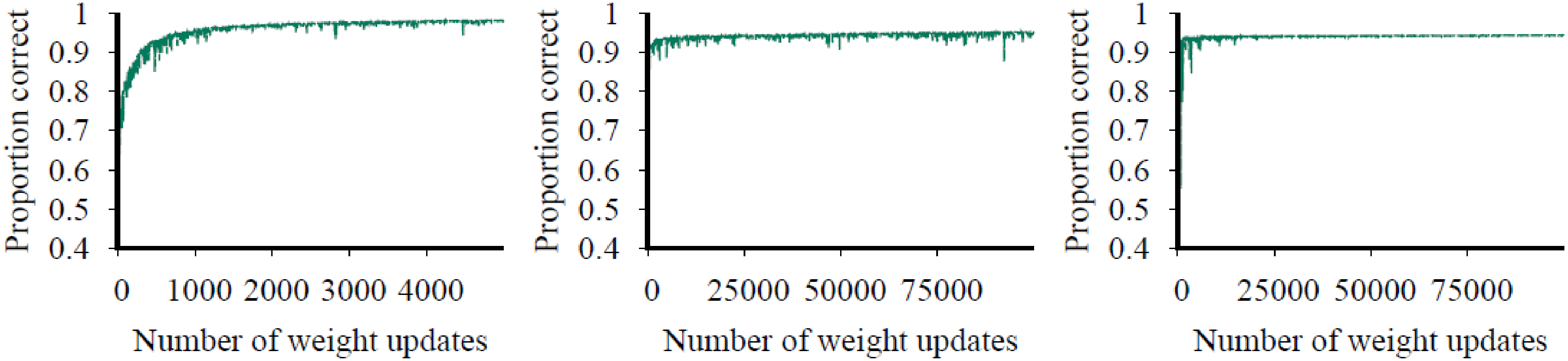
Improvements on Training Results



Improvements on Training Results



Improvements on Training Results



Logistic regression converges
far more quickly and reliably.