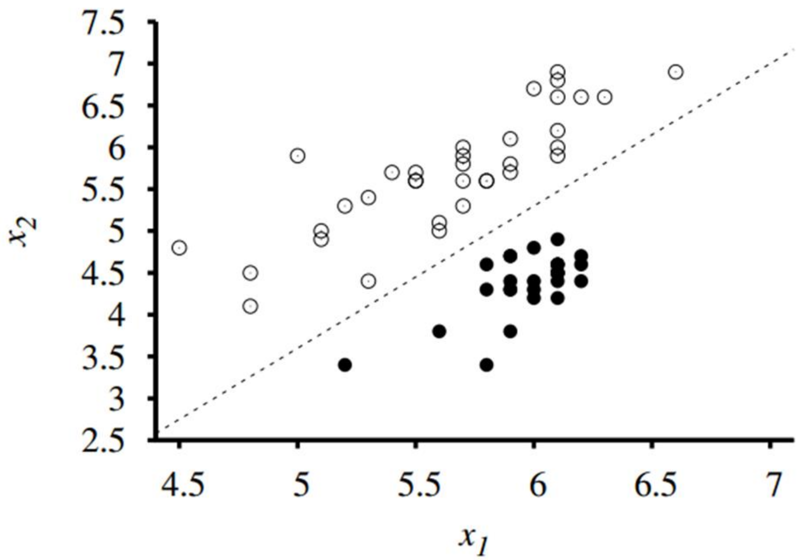
Logistic Regression

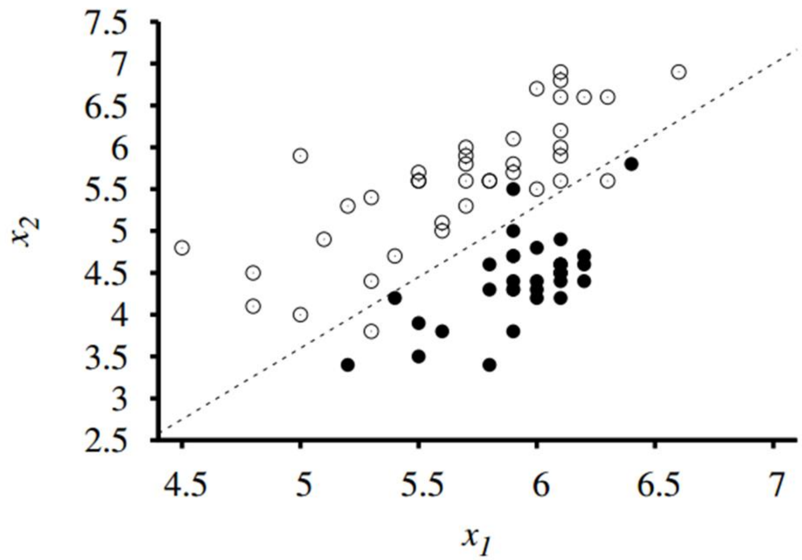
Abdus Salam Azad

Linear Functions as Classifier

- Linear functions can be used to do classification as well as regression



(a)



(b)

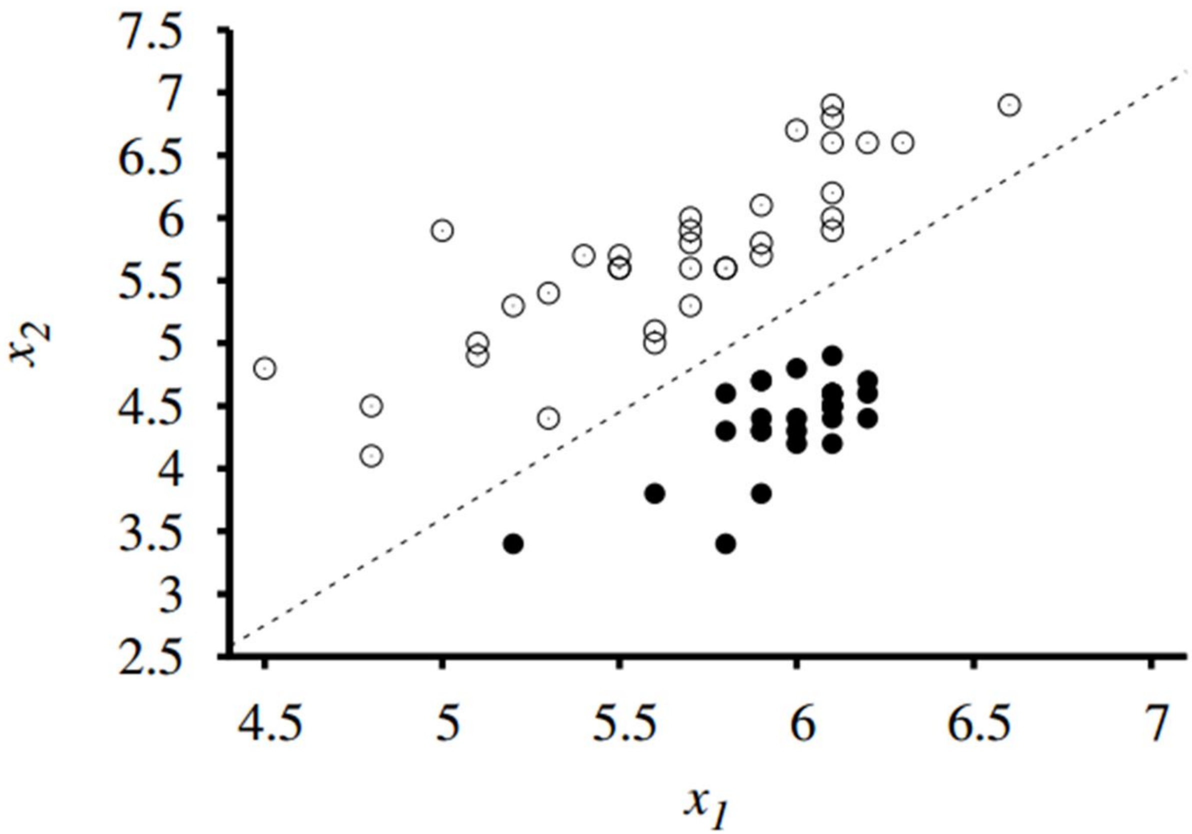
(a) Linearly Separable

(b) After addition of more data they are no longer Linearly Separable

Decision Boundary

- A **decision boundary** is a line (or a surface, in higher dimensions) that separates the two classes
- A linear decision boundary is called a **linear separator** and data that admit such a separator are called **linearly separable**

Linear Functions as Classifier

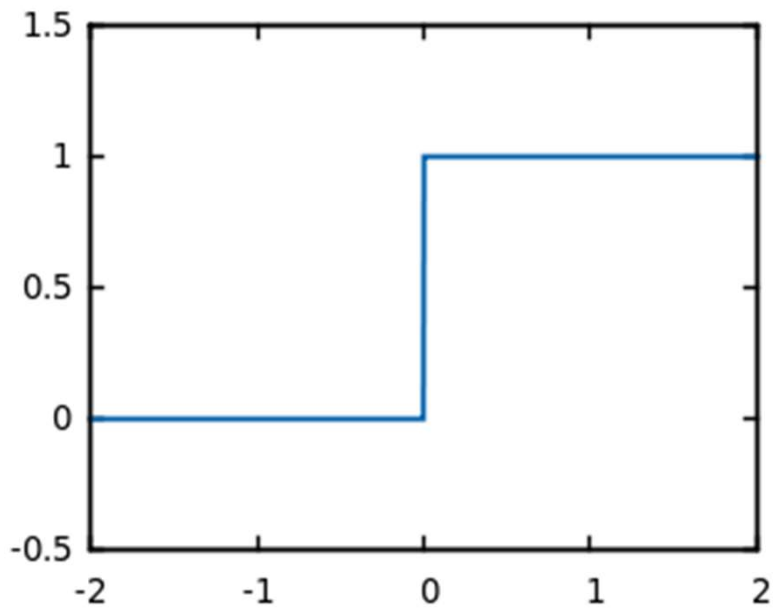


$h_{\mathbf{w}}(\mathbf{x}) = 1$ if $\mathbf{w} \cdot \mathbf{x} \geq 0$ and 0 otherwise.

Threshold Functions

- For convenience, we define a threshold function

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



Finding optimal weights

- We need to find appropriate weights to minimize the loss
- So we can apply Gradient Descent !!
  - NO !!!

- The gradient is zero almost everywhere in weight space except at those points where $\mathbf{w} \cdot \mathbf{x} = 0$
  - at those points the gradient is undefined.

Perceptron Learning Rule

- Provided that the data are linearly separable, a simple update rule can be used
  - Perceptron Learning Rule

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

- Converges to a solution

- Does the rule ring any bell ?

Perceptron Learning Rule

- Provided that the data are linearly separable, a simple update rule can be used
  - Perceptron Learning Rule

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

- Converges to a solution

- It is the same updating rule as Linear Regression ;-)

Perceptron Learning Rule

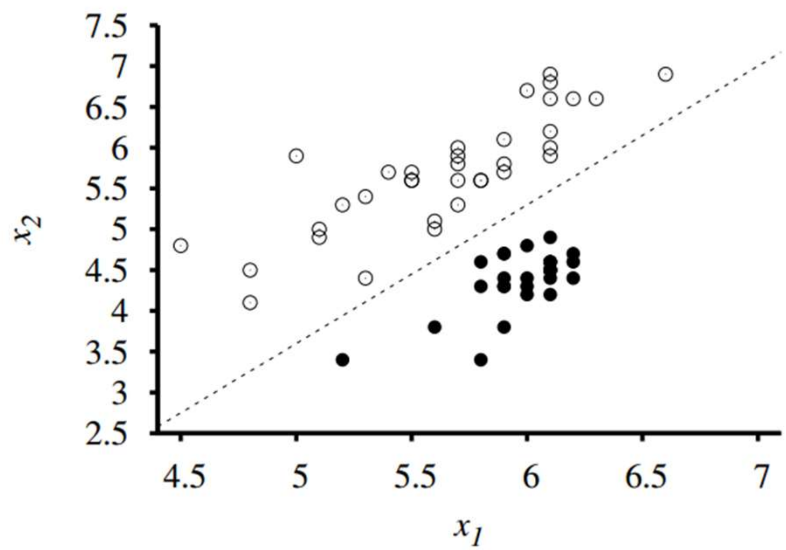
- We are dealing with binary classification
- There are three cases

- If the output is correct, i.e., $y = h_{\mathbf{w}}(\mathbf{x})$, then the weights are not changed.
- If y is 1 but $h_{\mathbf{w}}(\mathbf{x})$ is 0, then w_i is *increased* when the corresponding input x_i is positive and *decreased* when x_i is negative. This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ bigger so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 1.
- If y is 0 but $h_{\mathbf{w}}(\mathbf{x})$ is 1, then w_i is *decreased* when the corresponding input x_i is positive and *increased* when x_i is negative. This makes sense, because we want to make $\mathbf{w} \cdot \mathbf{x}$ smaller so that $h_{\mathbf{w}}(\mathbf{x})$ outputs a 0.

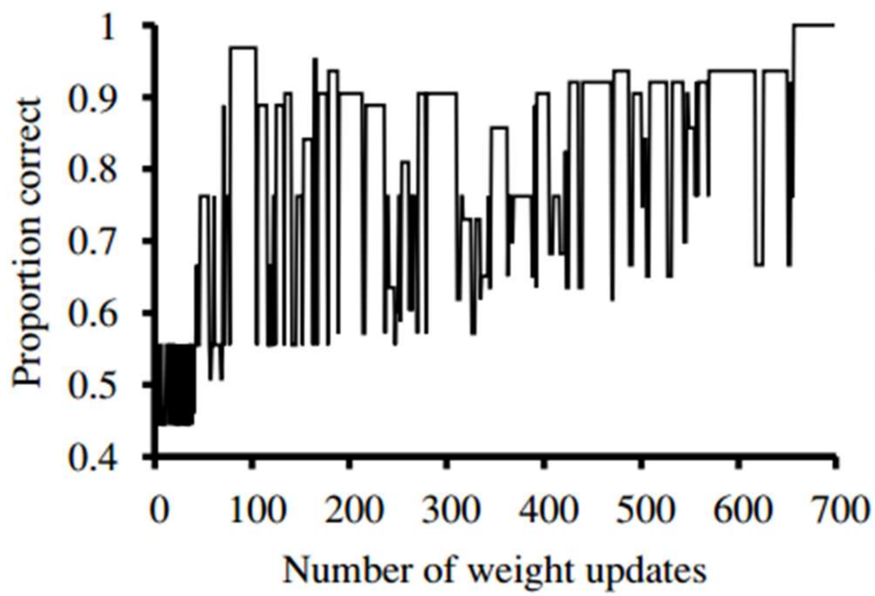
$$w_i \leftarrow w_i + \alpha \left(y - h_{\mathbf{w}}(\mathbf{x})\right) \times x_i$$

Training Curve

- We inspect the training curve
- Each weight update is done based on one random example like SGD
- A linearly separable data is used

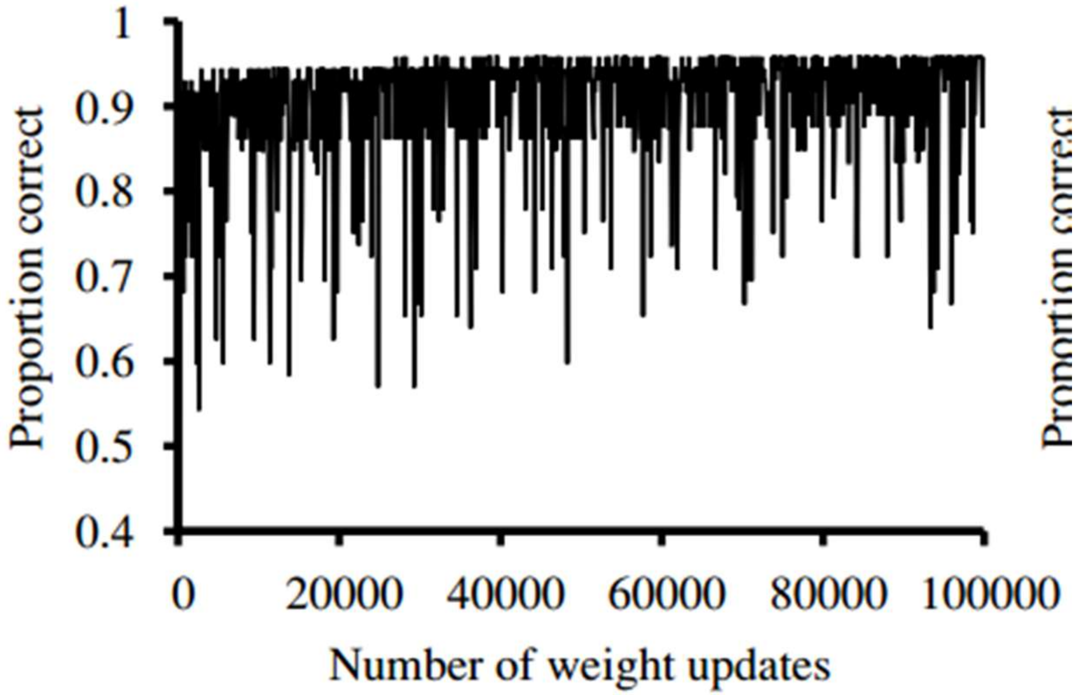


Training Curve



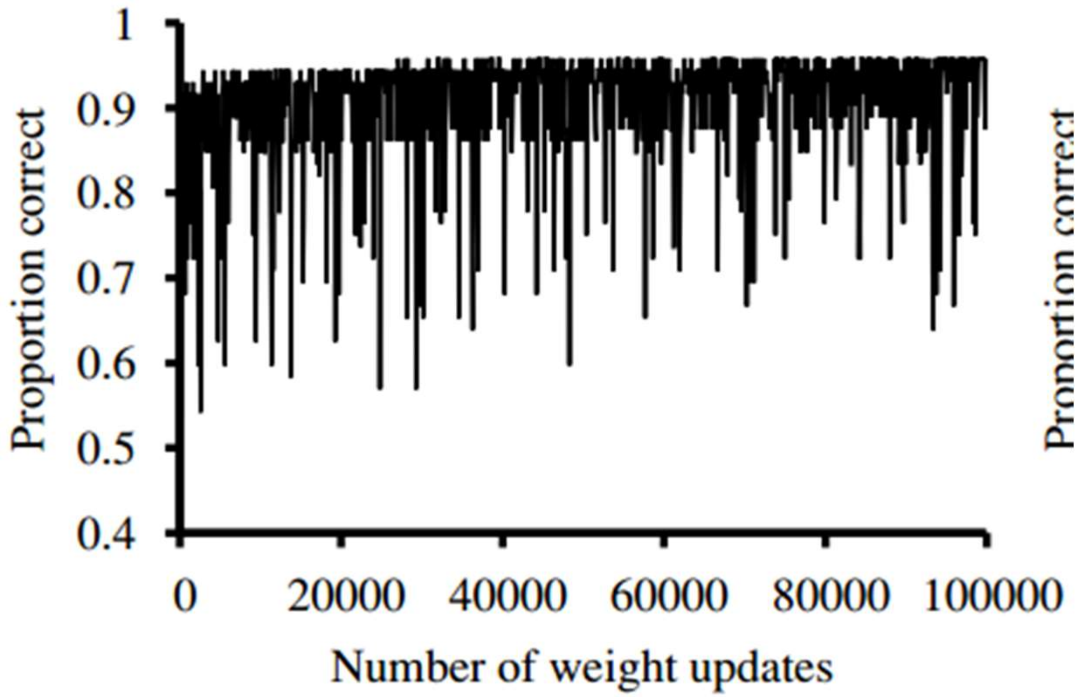
- The curve shows the update rule converging to a zero-error linear separator.
- The "convergence" process isn't exactly pretty, but it always works.
- This particular run takes 657 steps to converge, for a data set with 63 examples, so each example is presented roughly 10 times on average.
  - Typically, the variation across runs is very large.

What if the data is not linearly separable ??



The perceptron learning rule fails to converge even after 10,000 steps in this example

What if the data is not linearly separable ??

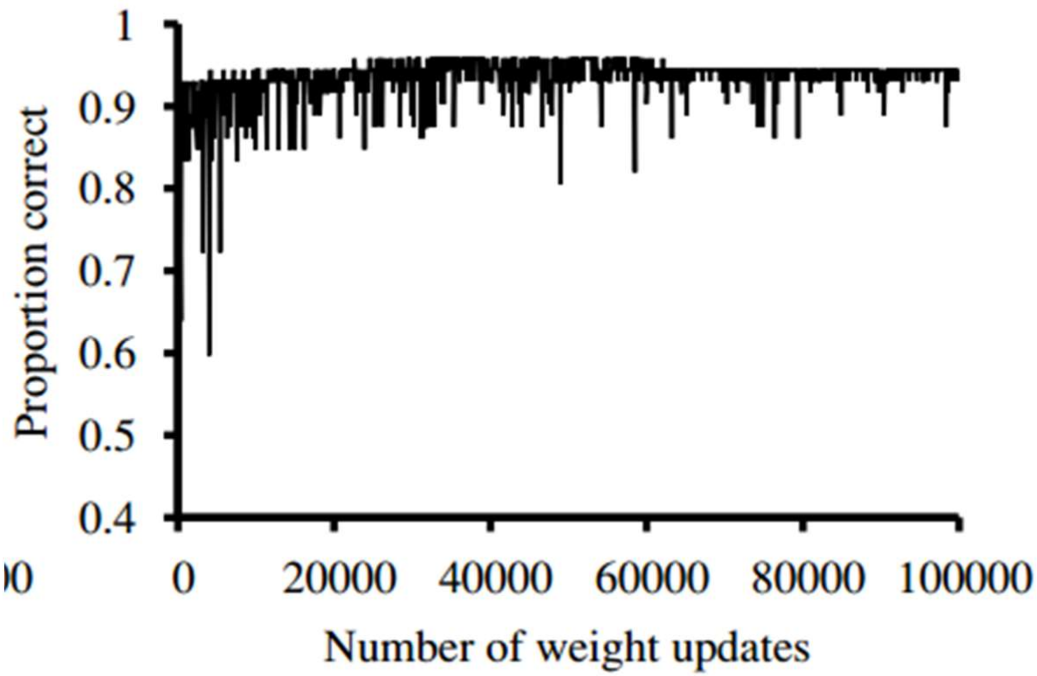


- Even though it hits the minimum-error solution many times, the algorithm keeps changing the weights.
- In general, the perceptron rule may not converge to a stable solution for fixed learning rate α

What if the data is not linearly separable ??

- The perceptron rule may not converge to a stable solution for fixed learning rate α
- If α decays as $O(1/t)$ where t is the iteration number, then the rule can be shown to converge to a minimum-error solution when examples are presented in a **random sequence**.
- The minimum-error solution is NP-hard, so one expects that many presentations of the examples will be required for convergence to be achieved

What if the data is not linearly separable ??



with a learning rate schedule

$\alpha(t) = 1000/(1000 + t)$

Perceptron Learning Rule

- The hard nature of the threshold causes some problems:
  - The hypothesis $h_{\mathbf{w}}(\mathbf{x})$ is not differentiable
  - It is a discontinuous function of its inputs and its weights
- This makes learning with the perceptron rule very unpredictable

Perceptron Learning Rule

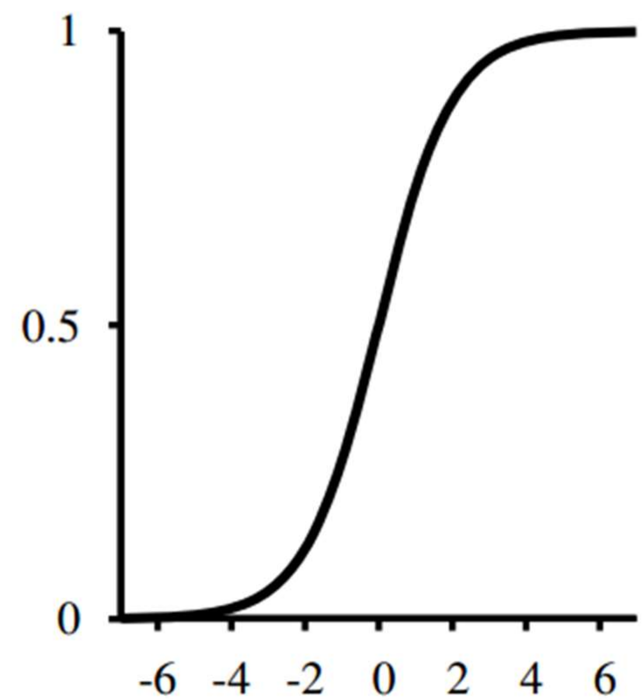
- The linear classifier always announces a completely confident prediction of 1 or 0
  - even for examples that are very close to the boundary

Linear Classification with Logistic Regression – Softening the hard threshold

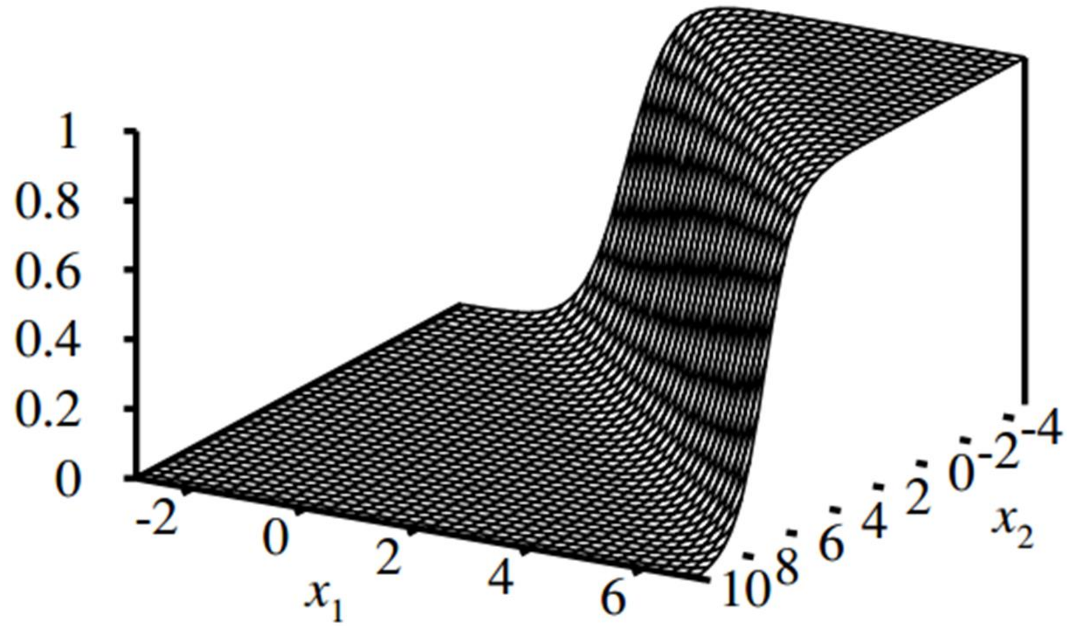
- The logistic function
  - Also called sigmoid function

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

  - The model is now called logistic regression
- Has more desirable mathematical properties



Logistic Regression



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

Plot of a logistic regression hypothesis $h_{\mathbf{w}}(\mathbf{x})$ = Logistic($\mathbf{w} \cdot \mathbf{x}$)
for the same linearly non-separable data

Logistic Regression

- Play with the logistic function with this online tool
  - https://www.desmos.com/calculator/bgontvxotm

Logistic Regression

- The output, being a number between 0 and 1, can be interpreted as a *probability* of belonging to the class labeled 1.
- The hypothesis forms a soft boundary in the input space
  - gives a probability of 0.5 for any input at the center of the boundary region
  - approaches 0 or 1 as we move away from the boundary

Computing the gradients

- Gradient descent computation is now possible
  - So is SGD
- Our hypotheses no longer output just 0 or 1, we will use the L2 loss function

Computing the gradients

- We'll use g to stand for the logistic function

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w} \cdot \mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i \ .
\end{aligned}
$$

Computing the gradients

- We'll use g to stand for the logistic function

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\mathbf{w}) &= \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x})) \times \frac{\partial}{\partial w_i}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times \frac{\partial}{\partial w_i}\mathbf{w} \cdot \mathbf{x} \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x})) \times g'(\mathbf{w} \cdot \mathbf{x}) \times x_i \, .
\end{aligned}
$$

Computing the gradients

The derivative g' of the logistic function satisfies $g'(z) = g(z)(1 - g(z))$, so we have

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

so the weight update for minimizing the loss is

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

Logistic Regression

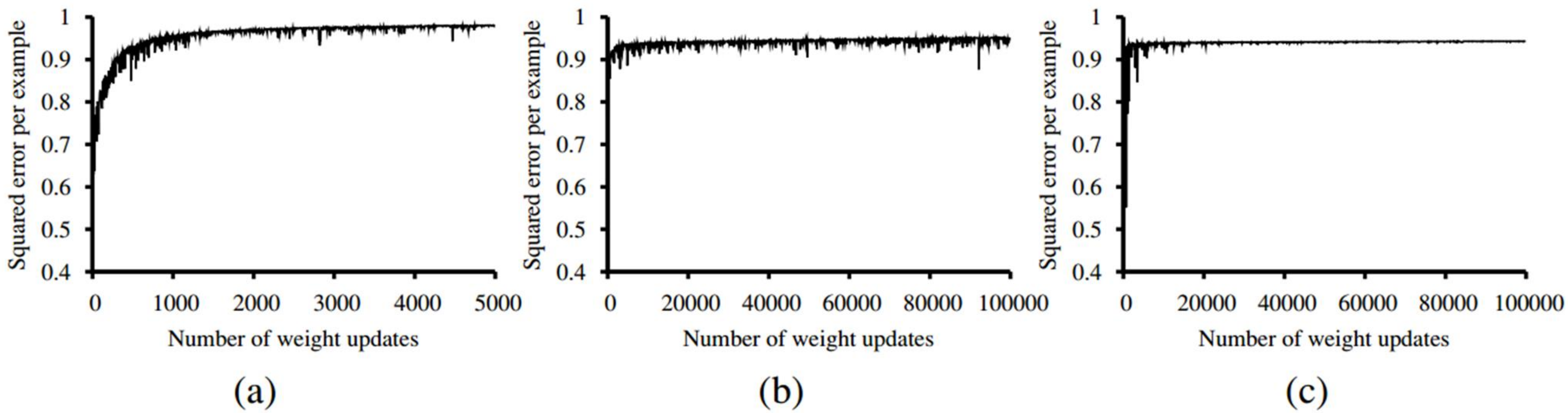


Figure 18.18 Repeat of the experiments in Figure 18.16 using logistic regression and squared error. The plot in (a) covers 5000 iterations rather than 1000, while (b) and (c) use the same scale.

(a) Linearly separable data. (b) and (c) non separable data. (c) uses varying learning rate

Resource

- Russel & Norvig
  - Chapter 18: