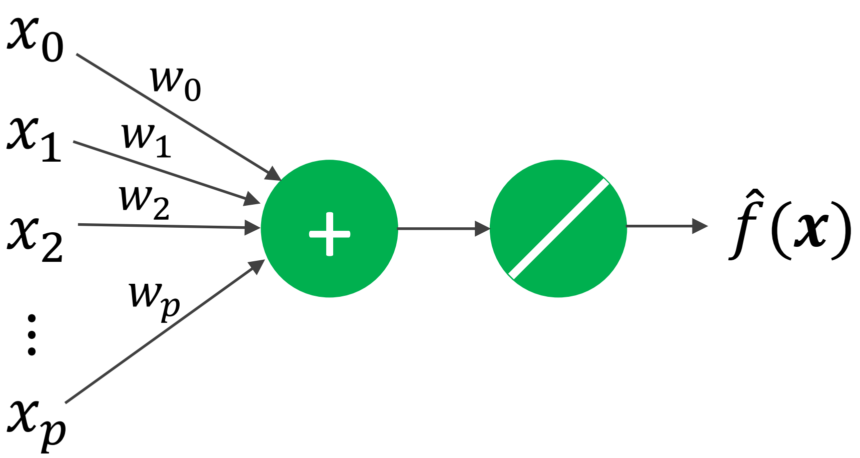
Linear models II

Lecture 05

Recap on linear models

Linear Regression

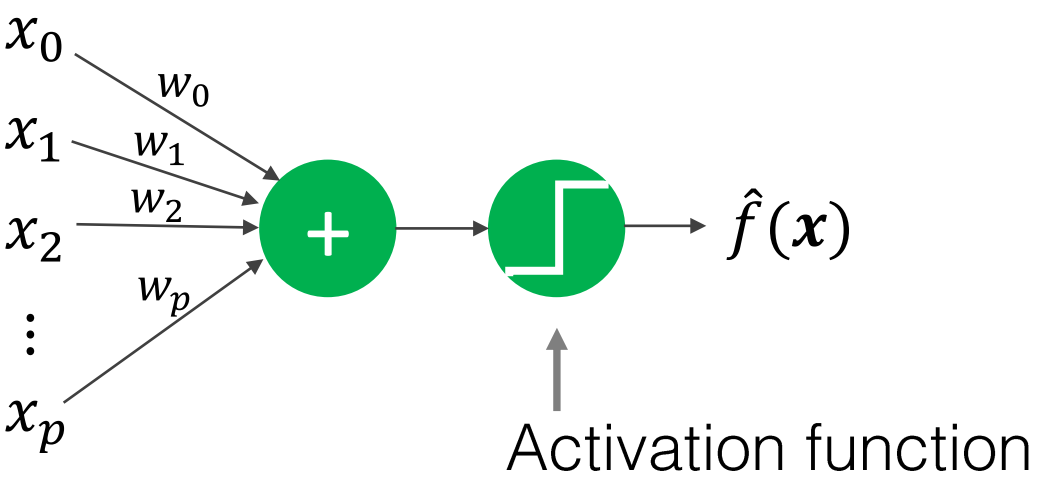
$$\hat{f}(\boldsymbol{x}) = \sum_{i=0}^{p} w_i x_i$$



Linear Classification

(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$



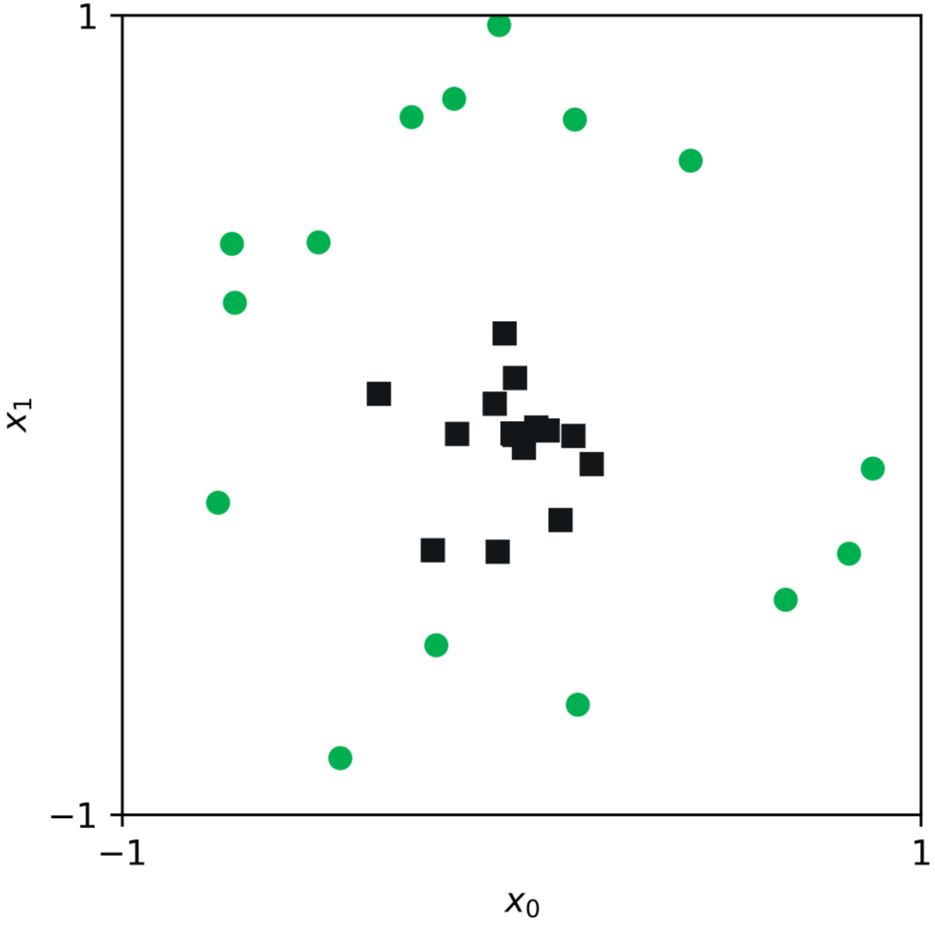
Source: Abu-Mostafa, Learning from Data, Caltech

Can I model nonlinear relationships?

Limitations of linear decision boundaries

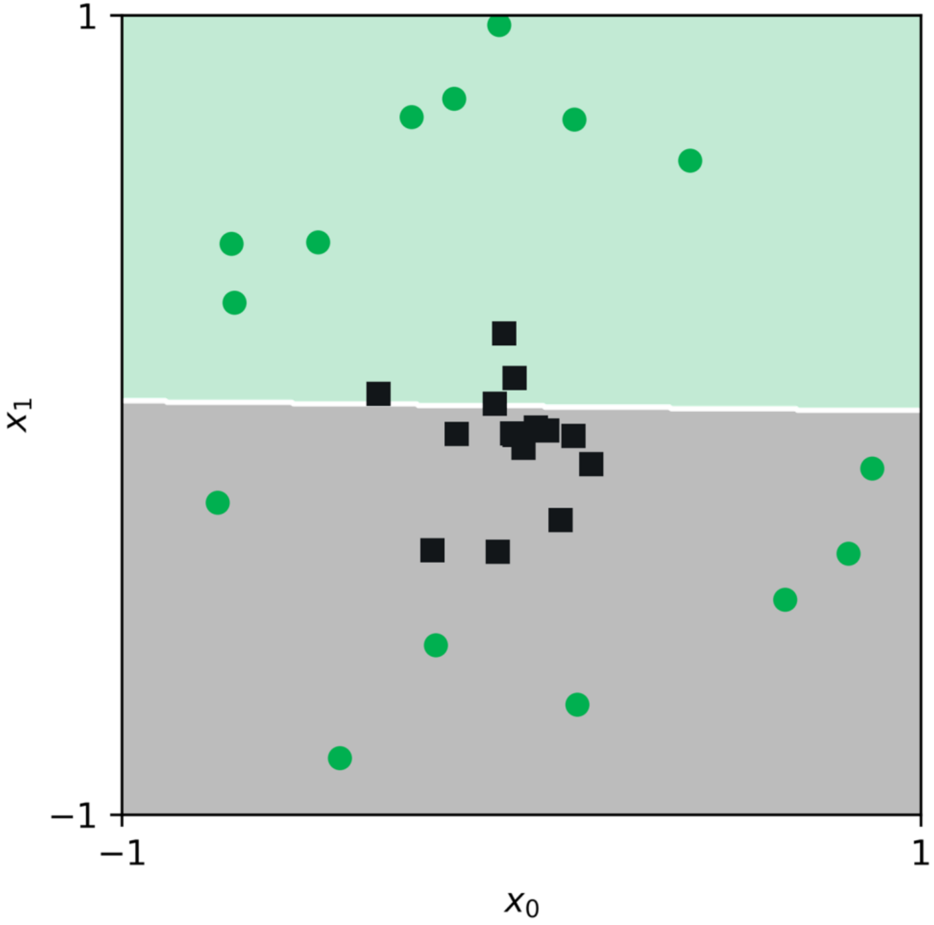
Original data

$$\boldsymbol{x}$$



Classify the features in this X-space

$$\hat{f}_{\boldsymbol{x}}(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{x})$$



Transformations of features

Consider a digits example…

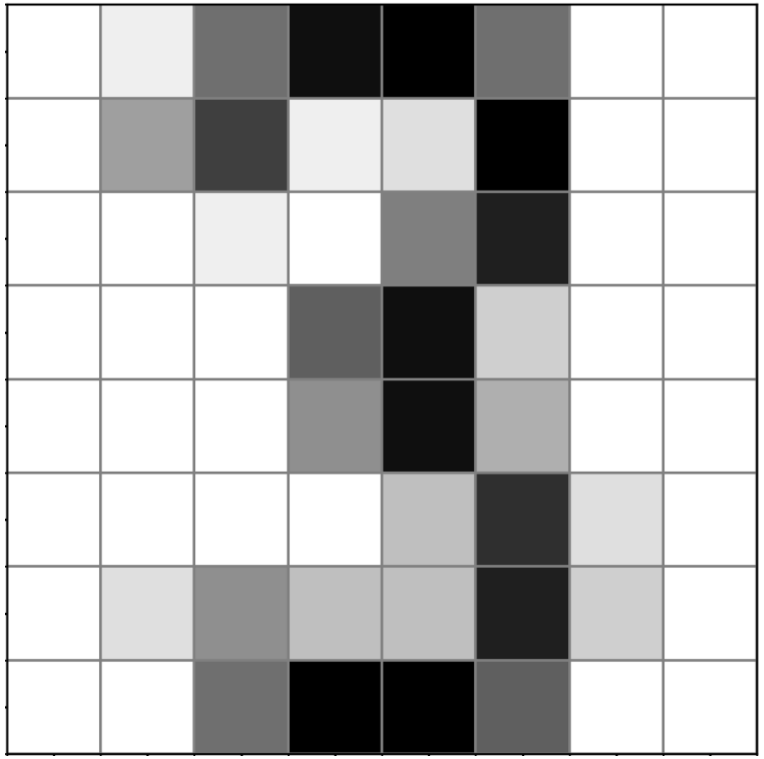
$$\boldsymbol{x} = [x_1, x_2, x_3, \ldots, x_{64}]$$

We could **create features** based on the raw features. For example:

$$\boldsymbol{z} = [x_1 x_2, x_3^2, \frac{x_{64}}{x_{42}}]$$

Which can be written simply as variables in a new feature space:

$$\boldsymbol{z} = [z_1, z_2, z_3]$$

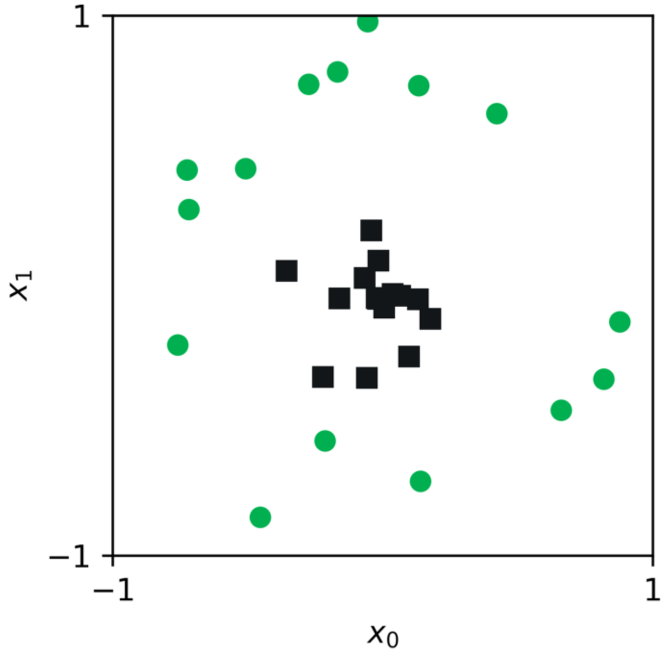


Source: Abu-Mostafa, Learning from Data, Caltech

1

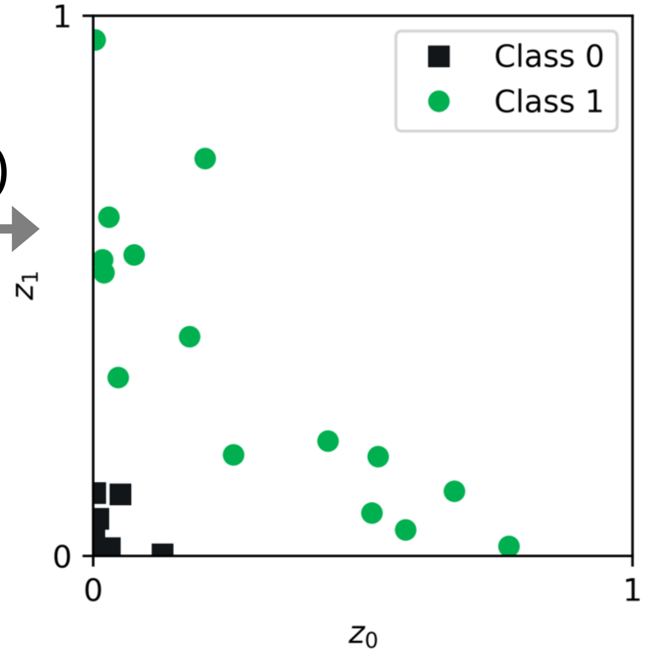
Original data

$\boldsymbol{x}$



transform the data

$$\boldsymbol{z} = \Phi(\boldsymbol{x})$$



2

This example transform is quadratic

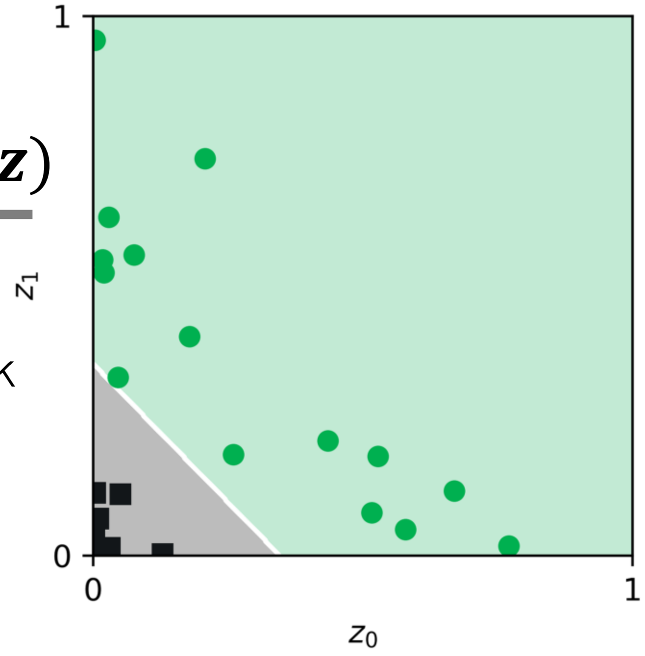
$$z_i = \Phi(x_i) = x_i^2$$

$$z_0 = x_0^2$$

$$z_1 = x_1^2$$

Classify the features in this Z-space

$$\hat{f}_z(\boldsymbol{z}) = \text{sign}(\boldsymbol{w}^T \boldsymbol{z})$$



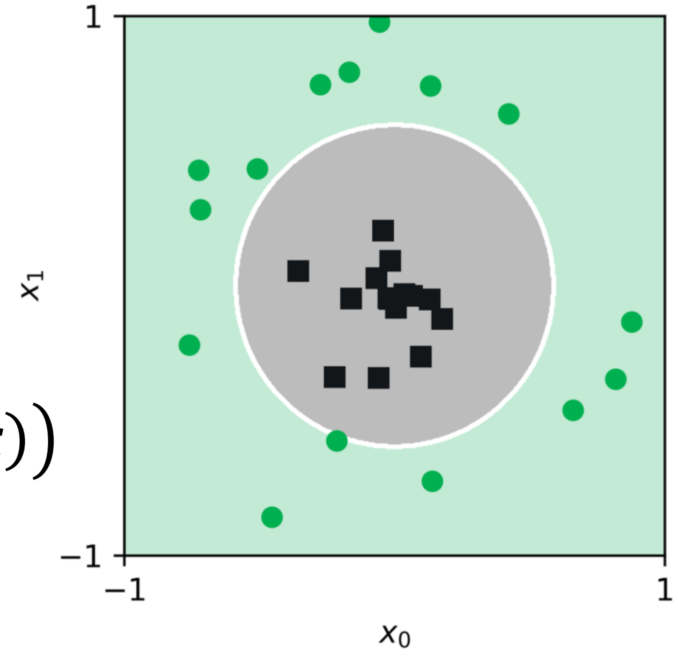
3

$$\boldsymbol{x} = \Phi^{-1}(\boldsymbol{z})$$

transform the data back

$$x_0 = z_0^{1/2}$$

$$x_1 = z_1^{1/2}$$



Predictions in the original X-space

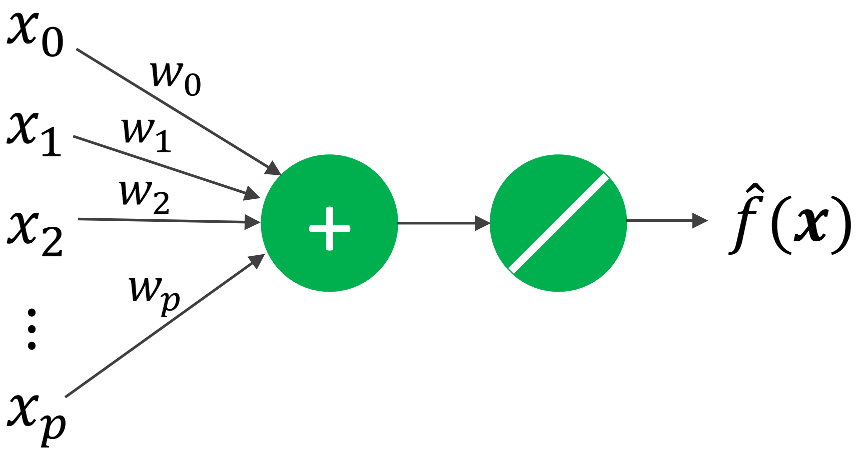
$$\hat{f}(\boldsymbol{x}) = \hat{f}_z\big(\Phi(\boldsymbol{x})\big)$$

4

Moving from regression to classification

Linear Regression

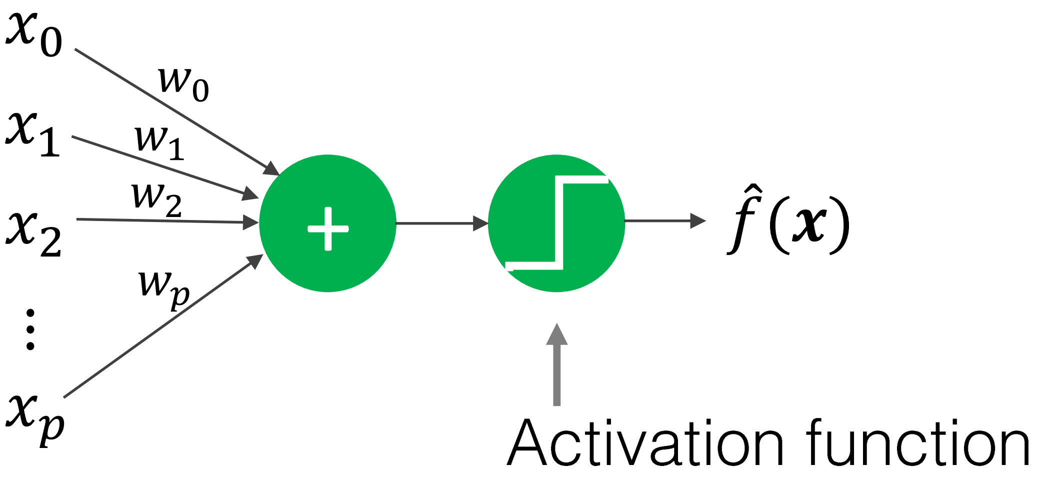
$$\hat{f}(\boldsymbol{x}) = \sum_{i=0}^{p} w_i x_i$$



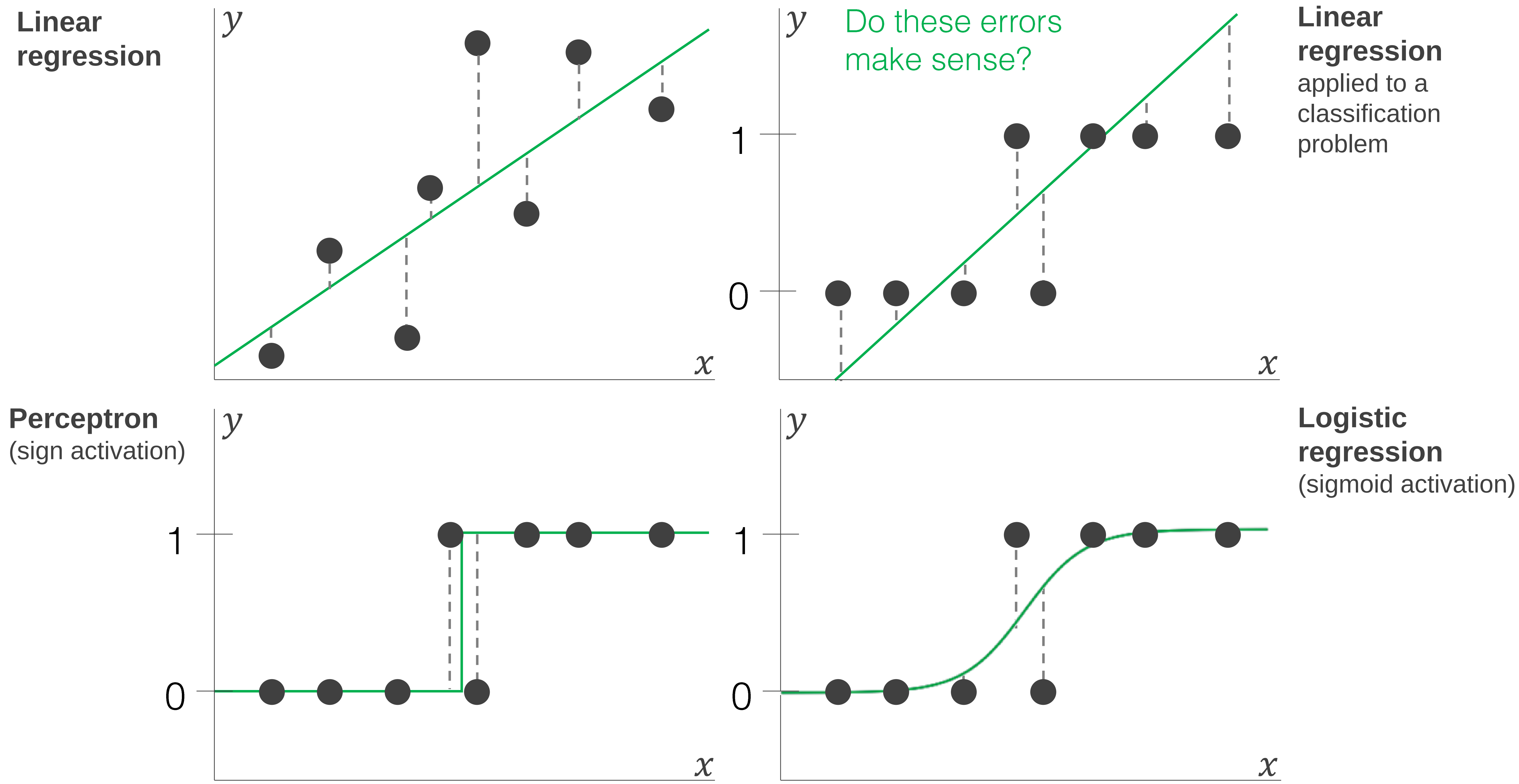
Linear Classification

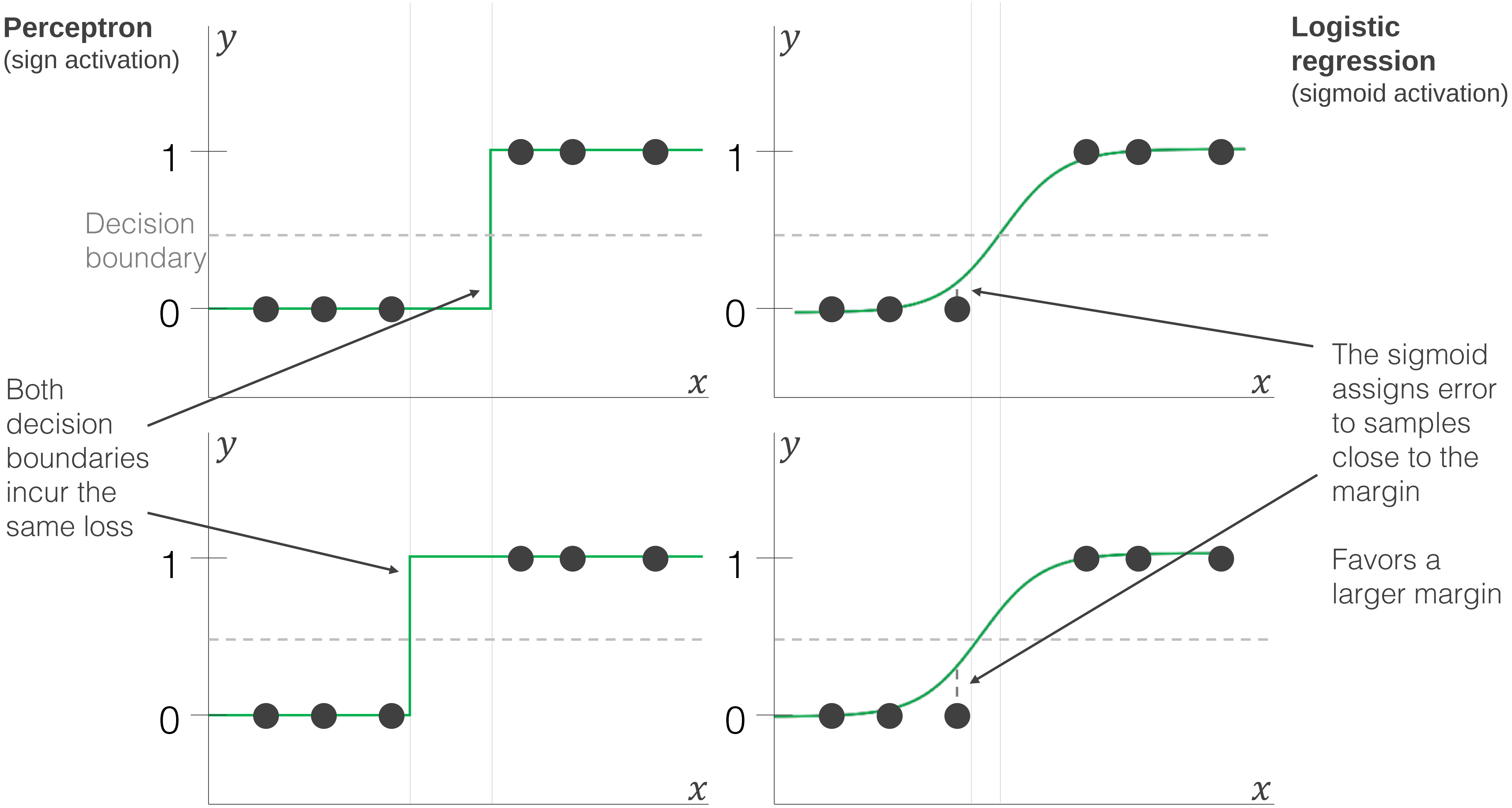
(perceptron)

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$



Source: Abu-Mostafa, Learning from Data, Caltech


Linear regression
Linear regression applied to a classification problem
Do these errors make sense?
Perceptron (sign activation)
Logistic regression (sigmoid activation)
y
x
1
0



Perceptron
(sign activation)
Logistic regression
(sigmoid activation)
y
1
0
x
Decision boundary
Both decision boundaries incur the same loss
The sigmoid assigns error to samples close to the margin
Favors a larger margin

Sigmoid function

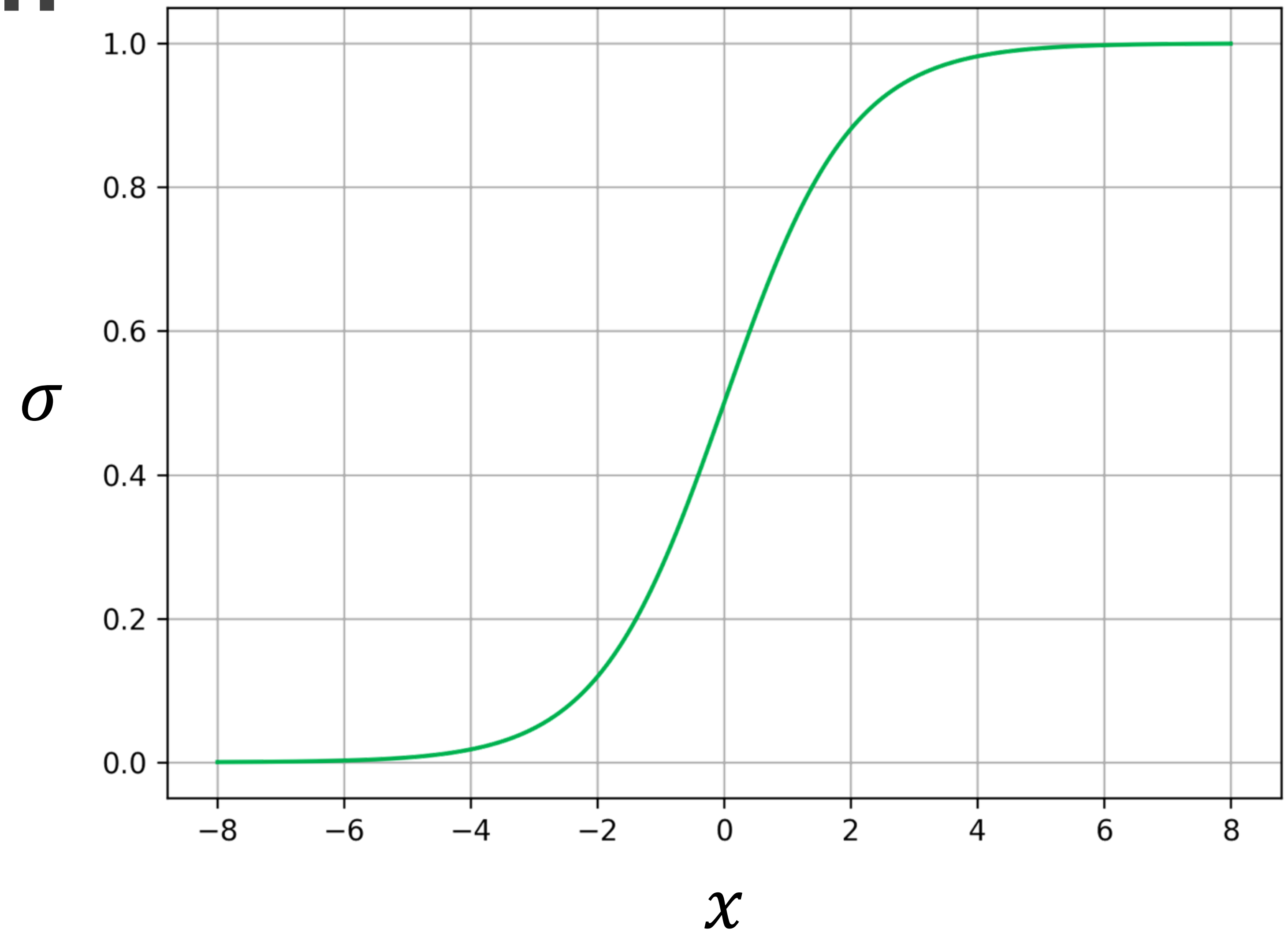
Definition

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Useful properties

$$\sigma(-x) = 1 - \sigma(x)$$

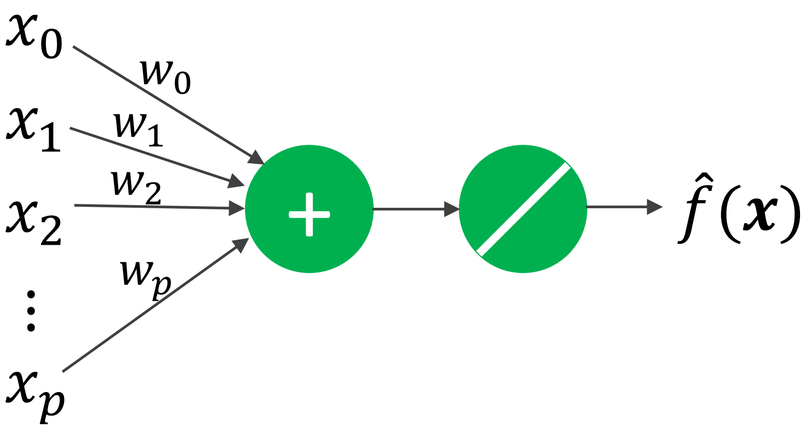
$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1 - \sigma(x))$$



Moving from regression to classification

Linear Regression

$$\hat{f}(\boldsymbol{x}) = \sum_{i=0}^{p} w_i x_i$$

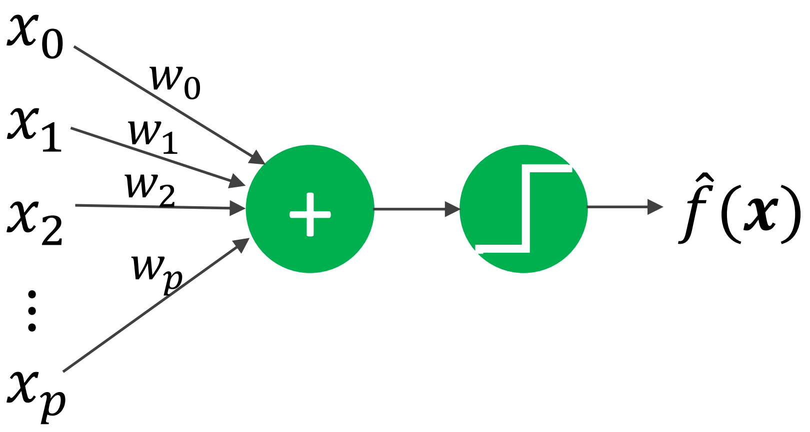


Linear Classification

Perceptron

$$\hat{f}(\boldsymbol{x}) = sign\left(\sum_{i=0}^{p} w_i x_i\right)$$

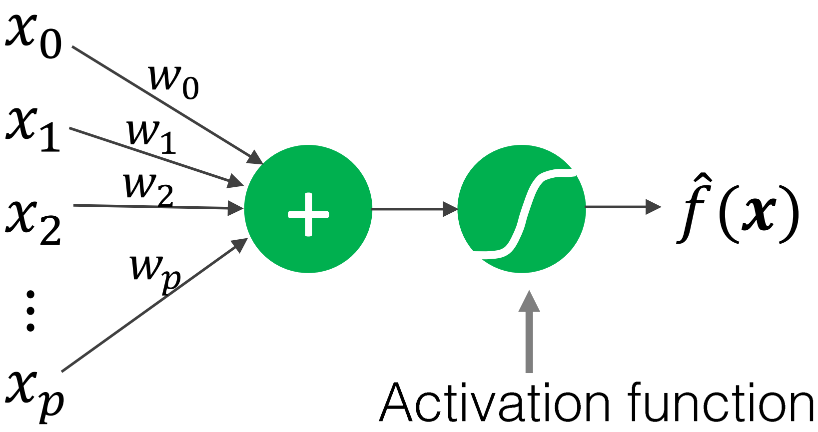
$$sign(x) = \begin{cases} 1 & x > 0 \\ -1 & \text{else} \end{cases}$$



Logistic Regression

$$\hat{f}(\boldsymbol{x}) = \sigma\left(\sum_{i=0}^{p} w_i x_i\right)$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Source: Abu-Mostafa, Learning from Data, Caltech

We take steps to fit our model

1. Define a cost function for measuring the fit

2. Optimize the cost function by adjusting model parameters

   a. Calculate the gradient

   b. Set the gradient to zero

   c. Solve for the model parameters

We COULD use the same cost function

Assume the cost function is mean square error

$$C(\boldsymbol{w}) \triangleq E_{in}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}\left(\hat{f}(\boldsymbol{x}_n, \boldsymbol{w}) - y_n\right)^2$$

$$\hat{f}(\boldsymbol{x}_n, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T\boldsymbol{x}_n)$$

Plug in our model

$$C(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(\sigma(\boldsymbol{w}^T\boldsymbol{x}_n) - y_n)^2$$

Calculate the gradient

$$\nabla_w C(\boldsymbol{w}) = \frac{2}{N}\sum_{n=1}^{N}[\sigma(\boldsymbol{w}^T\boldsymbol{x}_n) - y_n]\sigma(\boldsymbol{w}^T\boldsymbol{x}_n)[\boldsymbol{1} - \sigma(\boldsymbol{w}^T\boldsymbol{x}_n)]\boldsymbol{x}_n$$

Set the gradient to zero and solve for $\boldsymbol{w}$

$$\nabla_w C(\boldsymbol{w}) = \boldsymbol{0}$$

But we don’t for logistic regression...

There’s a more appropriate cost function to use for classification...

Refresher: Maximum Likelihood Estimation

We purchase a set of 1,000 identical scratch tickets and want to determine the underlying probability of each of them being a winner

Assume we have $N = 1{,}000$ **independent** Bernoulli random variables

$$P(X = 1) = p$$
$$P(X = 0) = 1 - p$$

Goal: find the value of p that maximizes the likelihood of our data

Goal: find the value of p that maximizes the likelihood of our data

$$P(X = 1) = p$$
$$P(X = 0) = 1 - p$$

For a **single observation**, the likelihood is:

$$L(x_i) = P(x_i|p) = p^{x_i}(1-p)^{1-x_i}$$

For a **multiple independent observations**, the likelihood is:

$$L(\boldsymbol{x}) = P(\boldsymbol{x}|p) = \prod_{i=1}^{N} P(x_i|p)$$

$$= p^{\sum x_i}(1-p)^{N-\sum x_i}$$

Goal: find the value of p that maximizes the likelihood of our data

$$P(\boldsymbol{x}|p) = p^{\sum x_i}(1-p)^{N-\sum x_i}$$

Maximizing the likelihood is equivalent to maximizing the log-likelihood

$$\ln[P(\boldsymbol{x}|p)] = \ln[p^{\sum x_i}(1-p)^{N-\sum x_i}]$$

$$\ln[P(\boldsymbol{x}|p)] = \ln(p)\sum_{i=1}^{N} x_i + \ln(1-p)\left[N - \sum_{i=1}^{N} x_i\right]$$

We take the **derivative of this log likelihood and set it to zero**, then solve for p

Goal: find the value of p that maximizes the likelihood of our data

We take the derivative of this log likelihood and set it to zero, then solve for p

$$\ln[P(\boldsymbol{x}|p)] = \ln(p)\sum_{i=1}^{N} x_i + \ln(1-p)\left[N - \sum_{i=1}^{N} x_i\right]$$

$$\frac{\partial \ln[P(\boldsymbol{x}|p)]}{\partial p} = \frac{\sum_{i=1}^{N} x_i}{p} - \frac{N - \sum_{i=1}^{N} x_i}{1-p} = 0$$

This results in our estimate being the mean of our observations:

$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Another interpretation of logistic regression

Our model: $\hat{y} = \hat{f}(\boldsymbol{x}) = \sigma(\boldsymbol{w}^T\boldsymbol{x})$

$$\sigma(\boldsymbol{w}^T\boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{w}^T\boldsymbol{x}}}$$

Logistic regression models the **probability that features belong to a class**

$$P(y_i = 1|\boldsymbol{x}_i) = \sigma(\boldsymbol{w}^T\boldsymbol{x}_i)$$

$$P(y_i = 0|\boldsymbol{x}_i) = 1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_i)$$

The interpretation of the Likelihood

The probability of observing the class labels $y_1, y_2, \dots, y_N$ corresponding to $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N$

The likelihood for **one observation**:

$$P(y_i|\boldsymbol{x}_i) = P(y_i = 1|\boldsymbol{x}_i)^{y_i} P(y_i = 0|\boldsymbol{x}_i)^{1-y_i}$$

The likelihood for **all observations**:

$$P(\boldsymbol{y}|\boldsymbol{X}) = P(y_1, y_2, \dots, y_N|\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N) = \prod_{i=1}^{N} P(y_i|\boldsymbol{x}_i)$$

Source: Malik Magdon-Ismail, Learning from Data

The likelihood for all observations:

$$P(\boldsymbol{y}|\boldsymbol{X}) = \prod_{i=1}^{N} P(y_i|\boldsymbol{x}_i) = \prod_{i=1}^{N} P(y_i = 1|\boldsymbol{x}_i)^{y_i} P(y_i = 0|\boldsymbol{x}_i)^{1-y_i}$$

$$= \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} [1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)]^{1-y_i}$$

This is the quantity we optimize

(to be precise, this is the negative of the cost function)

We can take the **logarithm**, then the **gradient**, then set equal to zero…

$$C(\boldsymbol{w}) = \prod_{i=1}^{N} \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)^{y_i} [1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)]^{1-y_i}$$

$$= \prod_{i=1}^{N} \hat{y}_i^{\,y_i} [1 - \hat{y}_i]^{1-y_i} \qquad \text{assuming} \quad \hat{y}_i \triangleq \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)$$

If we take the log of both sides:

$$\log C(\boldsymbol{w}) = \log \left[\prod_{i=1}^{N} \hat{y}_i^{\,y_i} [1 - \hat{y}_i]^{1-y_i} \right] = \sum_{i=1}^{N} \log(\hat{y}_i^{\,y_i} [1 - \hat{y}_i]^{1-y_i})$$

$$= \sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

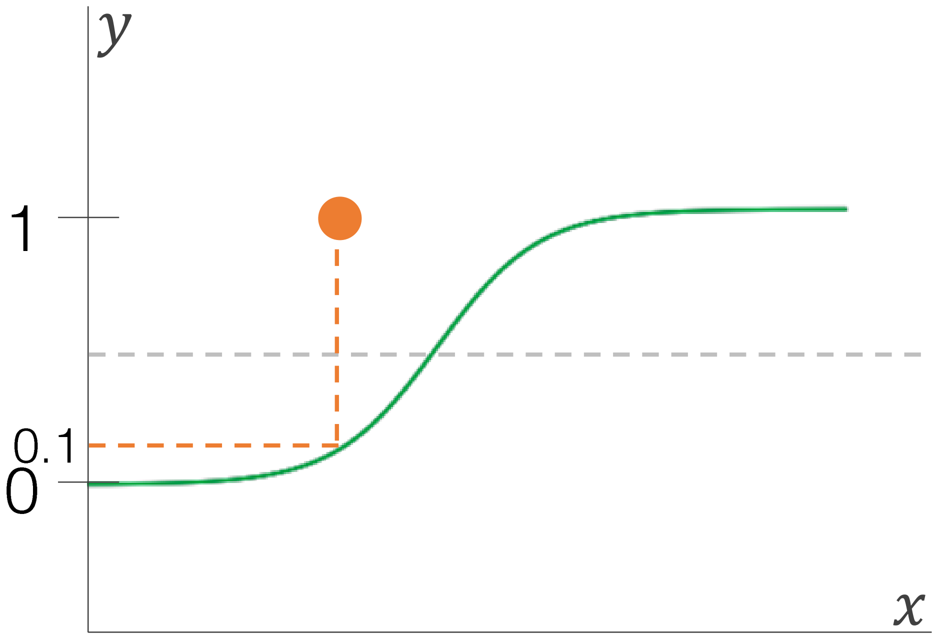
This is cross entropy

(to be precise, cross entropy is typically defined as the average of the negative of this quantity)

Mean Square Error vs Cross Entropy

$$\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$



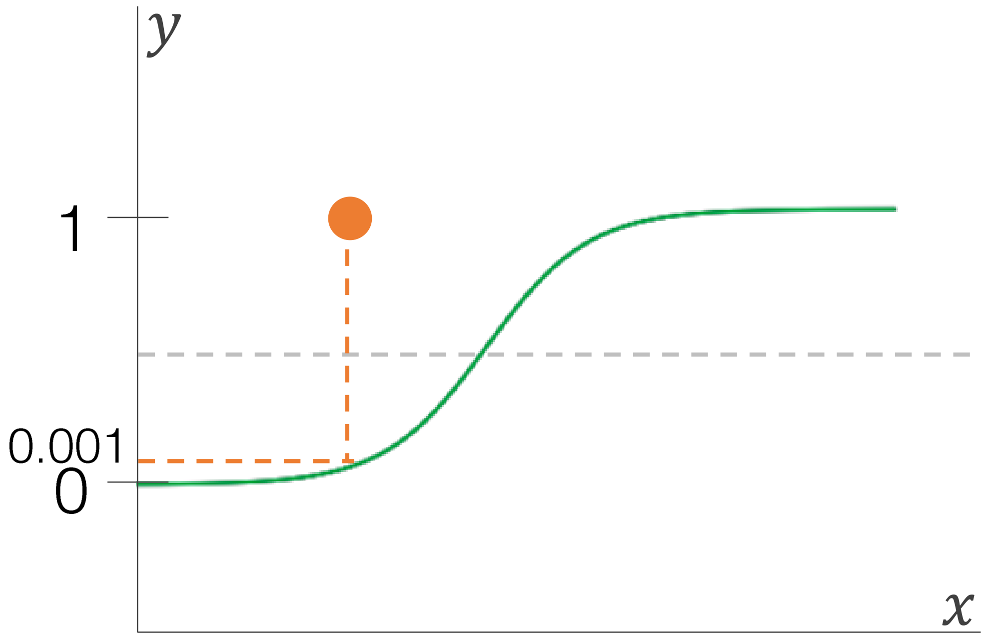
$$\begin{aligned} C_{MSE} &= (\hat{y}_i - y_i)^2 \\ &= (0.1 - 1)^2 \\ &= 0.81 \end{aligned}$$

$$\begin{aligned} C_{CE} &= -[y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)] \\ &= -[(1)\log(0.1) + (0)\log(0.9)] \\ &= 2.30 \end{aligned}$$

Mean Square Error vs Cross Entropy

$$\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2$$

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)$$



$$\begin{aligned} C_{MSE} &= (\hat{y}_i - y_i)^2 \\ &= (0.001 - 1)^2 \\ &= 0.998 \end{aligned}$$

$$\begin{aligned} C_{CE} &= -[y_i \log(\hat{y}_i) + (1-y_i)\log(1-\hat{y}_i)] \\ &= -[(1)\log(0.001) + (0)\log(0.999)] \\ &= 6.91 \end{aligned}$$

These costs functions are not solvable in closed form.

We need a new approach…

Gradient descent

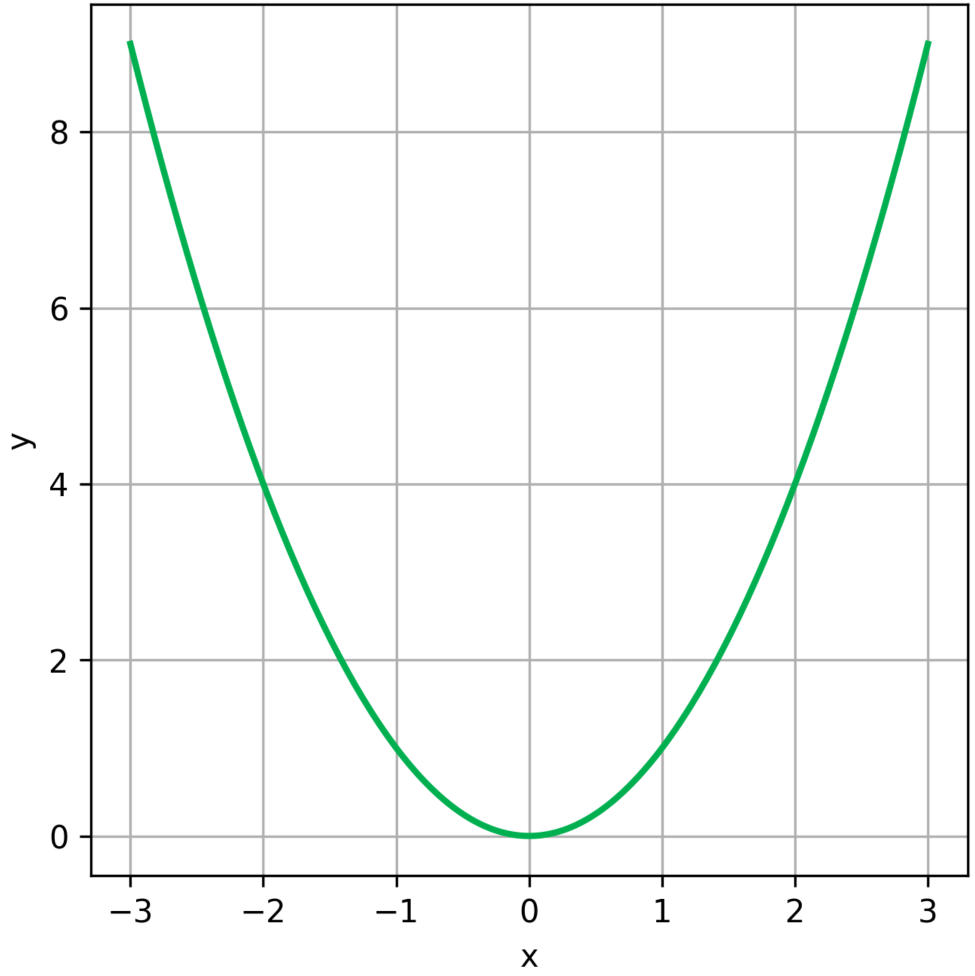
Minimize $y = x^2$

We start at a point and want to "roll" down to the minimum

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \eta \boldsymbol{v}$$

Learning rate

Direction to move in



Gradient descent

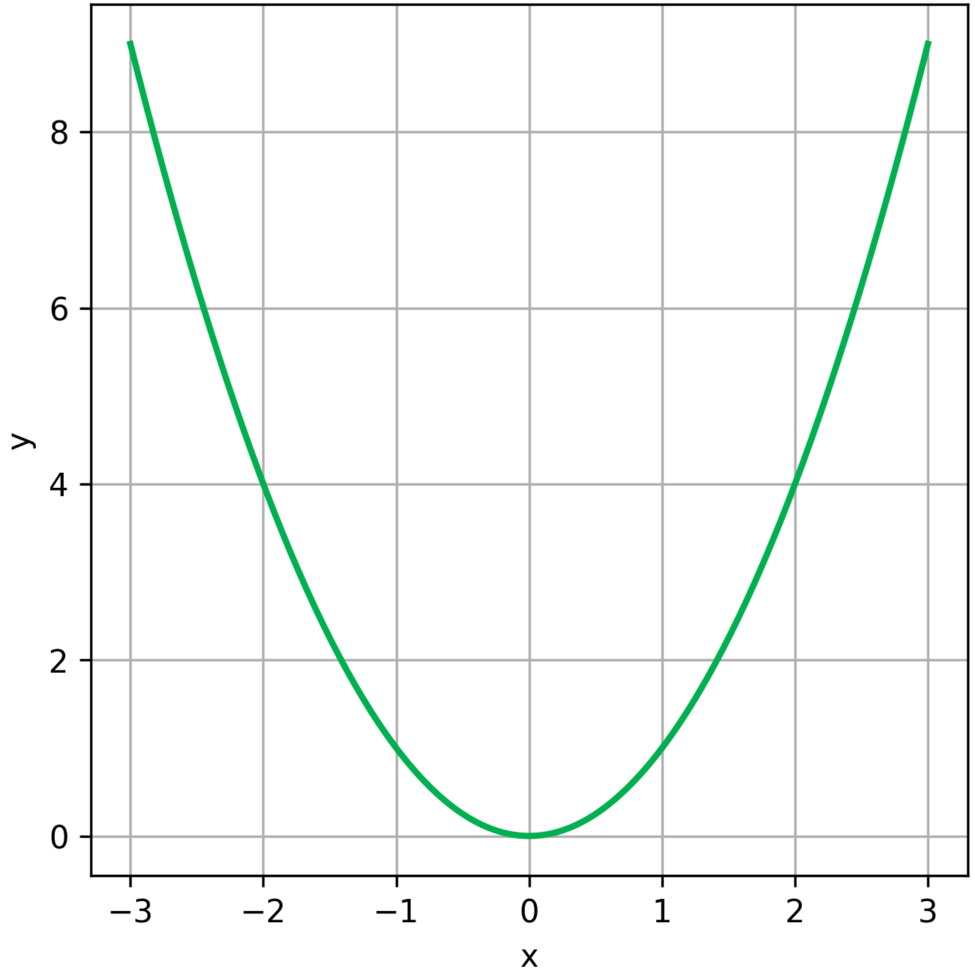
Minimize $f(x) = x^2$

The gradient points in the direction of steepest **positive** change

$$\frac{df(x)}{dx} = 2x$$

We want to move in the **opposite** direction of the gradient

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - \eta \nabla f\left(\boldsymbol{x}^{(i)}\right)$$



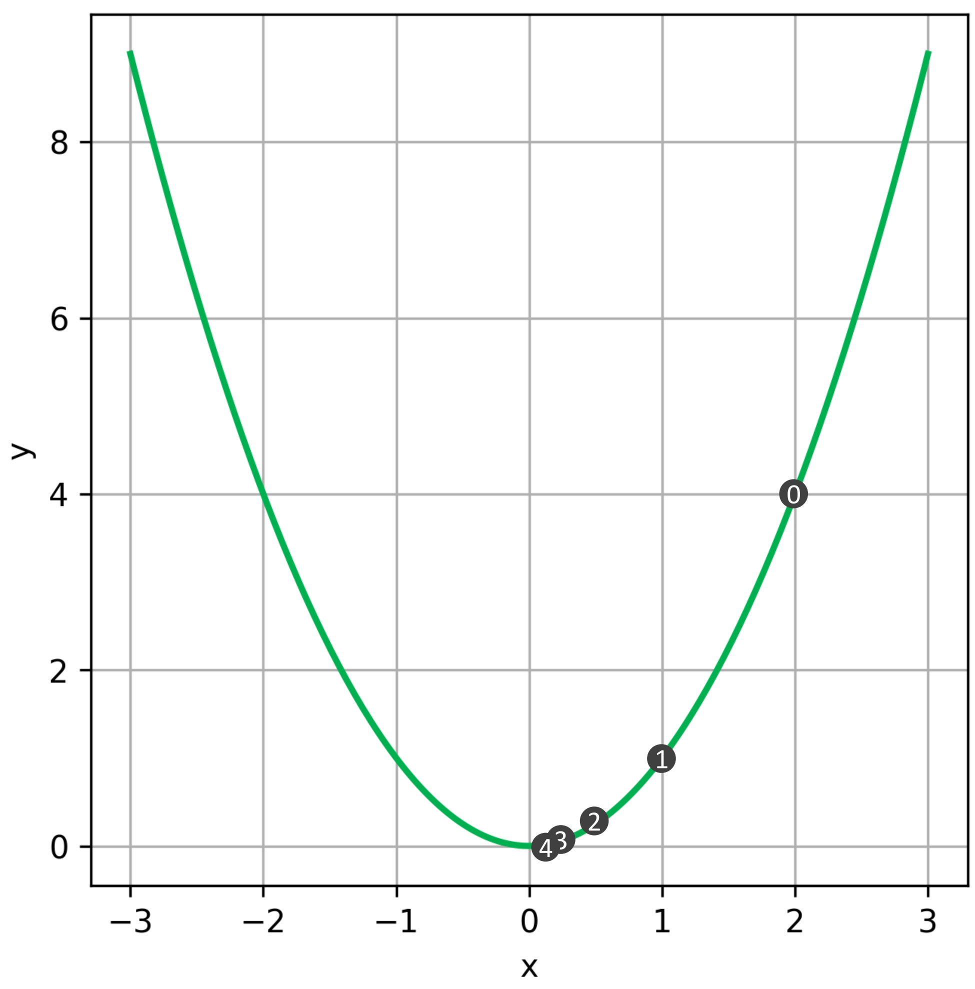
Gradient descent

Minimize $f(x) = x^2$

Assume $x^{(0)} = 2$ and $\eta = 0.25$

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - (0.25)(2\boldsymbol{x}^{(i)})$$
$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} - (0.5)\boldsymbol{x}^{(i)}$$

i	$x^{(i)}$	$y^{(i)}$
0	2	4
1	1	1
2	0.5	0.25
3	0.25	0.0625
4	0.125	0.0156



Takeaways

Transformations of features (**feature extraction**) may help to overcome nonlinearities

Logistic regression is much better suited for classification than linear regression

Logistic regression parameters must be estimated iteratively, and a method for that optimization is **gradient descent**

Gradient descent can be used for **cost function optimization** and there are a number of variants