# Machine Learning with Python

## Multivariate Linear Models For Regression And Classification

Dr. Aeshah Alsughayyir
Collage of Computer Science and Engineering
Taibah University

2021-2022

# Agenda

- Multivariant linear regression
  - Feature scaling
  - Polynomial Regression

- Logistic Regression: Binary-class classification
  - Sigmoid function
  - Decision boundary
  - Cost function

- Logistic Regression: Multi-class classification
  - SoftMax function

# Motivation

## Model against dataset

the housing price problem

| | area | price |
|---|---|---|
| 1 | | |
| 2 | | |
| ... | | |
| m | | |

$h_\theta(x) = \theta_0 + \theta_1 x_1$

| | area | age | price |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| ... | | | |
| m | | | |

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

| | area | .... | #Floors | price |
|---|---|---|---|---|
| 1 | | | | |
| 2 | | | | |
| ... | | | | |
| m | | | | |

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$

# Predict the house's price using many features

- In original version we had
  - X = house size, use this to predict
  - y = house price

- If in a new scheme we have more variables (such as number of bedrooms, number floors, age of the home)
  - $x_1$, $x_2$, $x_3$, $x_4$ are the four features
    - $x_1$ - size (feet squared)
    - $x_2$ - Number of bedrooms
    - $x_3$ - Number of floors
    - $x_4$ - Age of home (years)
  - y is the output variable (price)

**What can we do ?**

# Multivariate linear regression

$$\left.\begin{array}{r}\textbf{Multi−variante}\\ \textbf{Multi−features}\\ \textbf{Multi−variables}\end{array}\right\}\textbf{Linear regression}$$

# Multivariate linear regression

- *Multivariate linear regression* is the generalization of the univariate linear regression (seen earlier).
- As the name suggests, there are more than one independent variables, **x1,x2⋯,xn** and a dependent variable **y**.

**Notation**

- $x_1, x_2 \cdots, x_n$ denote the n features
- $y$ denotes the output variable to be predicted
- $n$ is number of features
- $m$ is the number of training examples
- $x^{(i)}$ is the $i^{th}$ training example
- $x_j^{(i)}$ is the $j^{th}$ feature of the $i^{th}$ training example

# Multivariate linear regression

- Multivariate Hypothesis

The hypothesis in case of univariate linear regression was,

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Extending the above function to multiple features, hypothesis of multivariate linear regression is given by,

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n \\ &= \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n, \text{ where } x_0 = 1 \\ &= \theta^T x, \text{ vectorizing above equation} \end{aligned} \tag{1}$$

- Where,
  - $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$ and $x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$

# Multivariate linear regression

- Multivariate Cost Function

The cost function for univariate linear regression was,

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Extending the above function to multiple features, the cost function for multiple features is given by,

$$\begin{aligned} J(\theta) &= \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left( \theta^T x^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{2m} \sum_{i=1}^{m} \left( \left( \sum_{j=0}^{n} \theta_j x_j^{(i)} \right) - y^{(i)} \right)^2 \end{aligned} \quad (2)$$

- Where $\theta$ is a vector give by $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$

# Multivariate linear regression

- Multivariate Gradient Descent

**Hypothesis:** $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

**Parameters:** $\theta_0, \theta_1, \ldots, \theta_n$

**Cost function:**

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

**Gradient descent:**

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

# Multivariate linear regression

- Multivariate Gradient descent

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n, \qquad \mathbf{x_0 = 1}$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left((\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n) - y^{(i)}\right)^2$$

$$\text{repeat until convergence} \left\{ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \right. \qquad (3)$$

$$\frac{\partial}{\partial \theta_0} J(\theta) = \quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = \quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) = \quad \theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...

Evaluating the partial derivative $\frac{\partial}{\partial \theta_j} J(\theta)$ gives,

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_j^{(i)} \qquad (4)$$

# Multivariate linear regression

- Multivariate Gradient descent

*Univariate Gradient decent*

**Gradient Descent**

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update $\theta_0, \theta_1$)

}

*Multivariate Gradient decent*

**New algorithm** $(n \geq 1)$:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

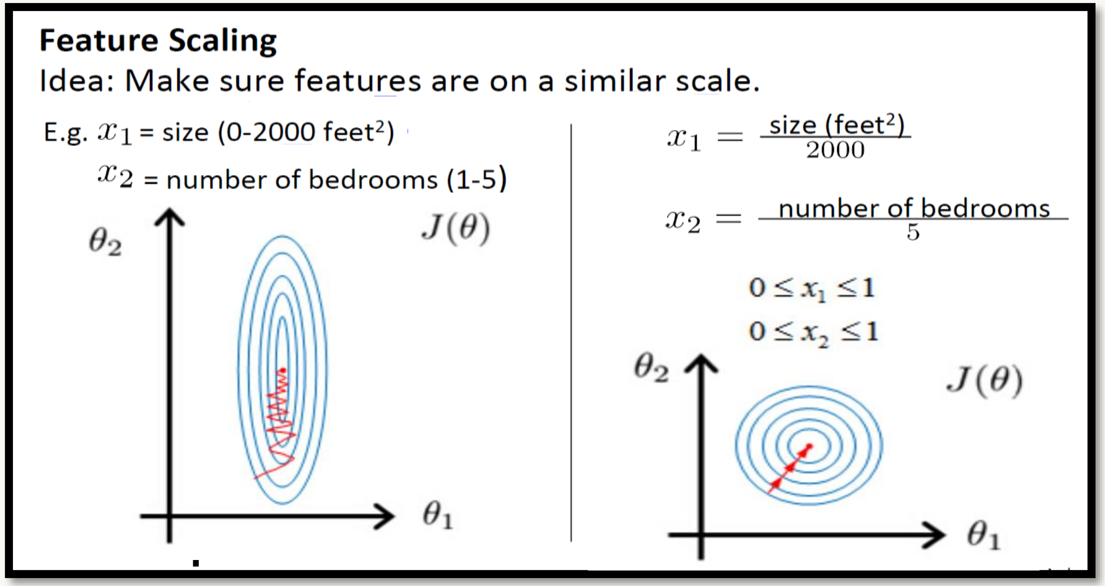
(simultaneously update $\theta_j$ for $j = 0, \ldots, n$)

}

# Feature Scaling

# Feature Scaling (Normalization)

- *Normalization* refers to normalizing the data dimensions so that they are of approximately the same scale.



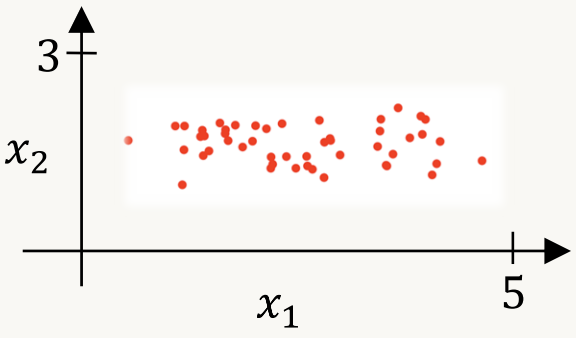

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

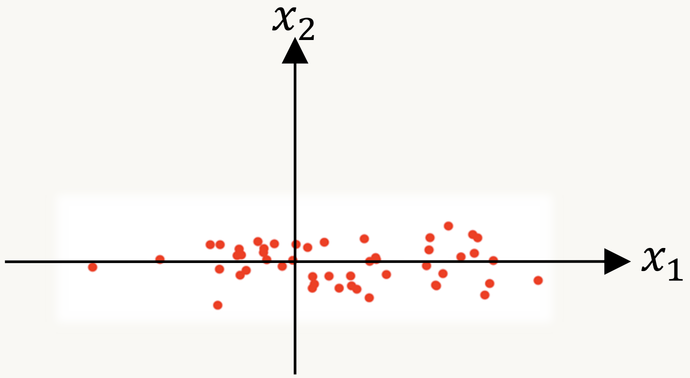
After **normalization** Contours become more like circles (as scaled between 0 and 1)

- As seen above, if the contours are ***skewed*** then learning steps would take longer to converge as the steps would be more prone to *oscillatory behavior* as shown in the left plot.
- Whereas if the features are properly scaled, then the plot is ***evenly distributed***, and the steps of gradient descent have better profile of convergence.

# Feature Scaling (Normalization)

- **Normalizing training sets**

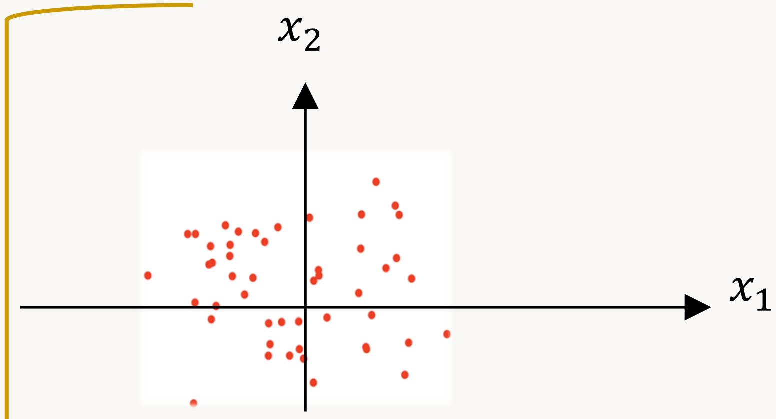



$$\mu_i = \frac{1}{m}\sum_{i=1}^{m}(x_i)$$

$$x_i = (x_i - \mu_i) \quad \forall i \in \{1,2,\cdots,n\}$$

- **Mean Normalization** is the most common form of preprocessing. It involves subtracting the mean across every individual *feature* in the data and has the geometric interpretation of *centering the cloud of data around the origin* along every dimension.
- **Not applied to the feature** $X_0$ , as it always =1

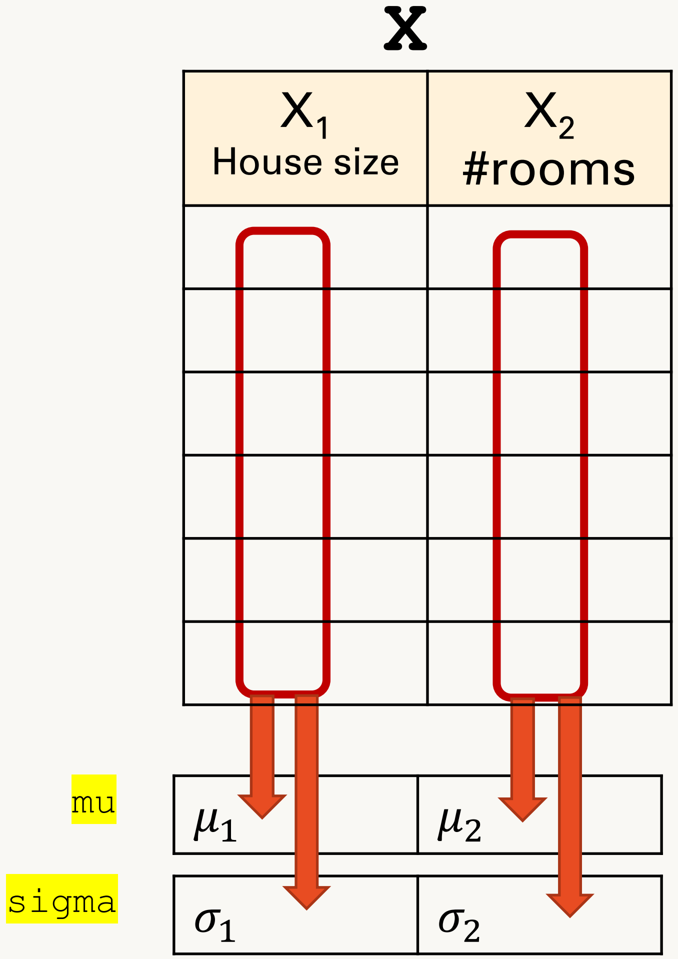



$$x_i = \frac{x_i - \mu_i}{S_i} \forall i \in \{1,2,\cdots,n\}$$

$$\text{standard deviation} = s_i = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_i)^2}$$

- $x_i$ is the feature value
- $\mu_i$ is the mean
- $S_i$ is the standard deviation or the range i.e., max−min

# Feature Normalization




$$x_i = \frac{x_i - \mu_i}{\sigma_i} \forall i \in \{1,2,\cdots,n\}$$

**X_norm**

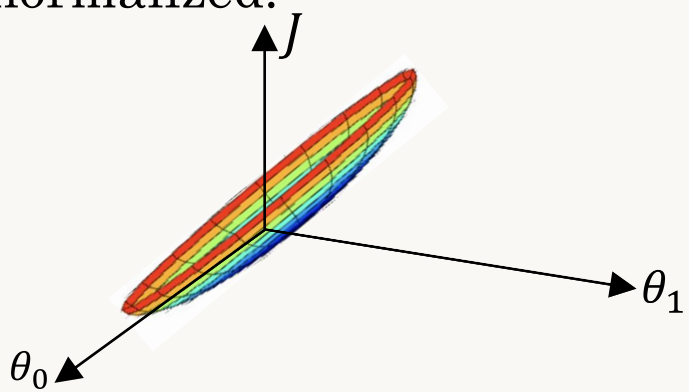
| $X_1$ | $X_2$ |
|---|---|
| $x_1^{(1)} = \frac{x_1^{(1)} - \mu_1}{\sigma_1}$ | $x_2^{(1)} = \frac{x_2^{(1)} - \mu_2}{\sigma_2}$ |
| $x_1^{(2)} = \frac{x_1^{(2)} - \mu_1}{\sigma_1}$ | $x_2^{(2)} = \frac{x_2^{(2)} - \mu_2}{\sigma_2}$ |
| ..... | ...... |

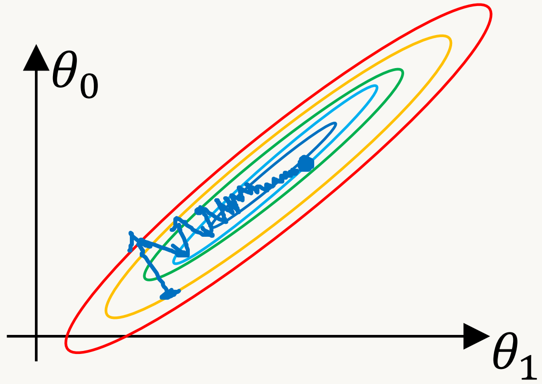
# Feature Normalization

- **Why normalize inputs?**

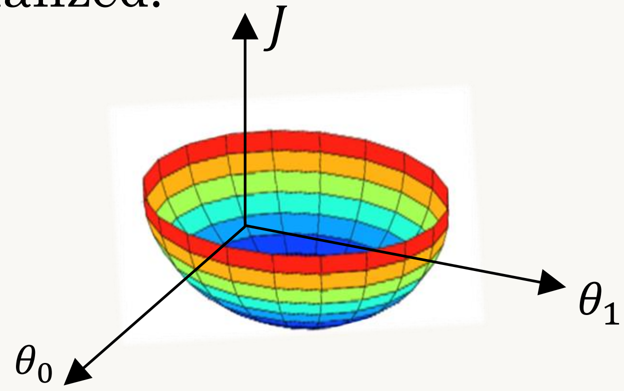
$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

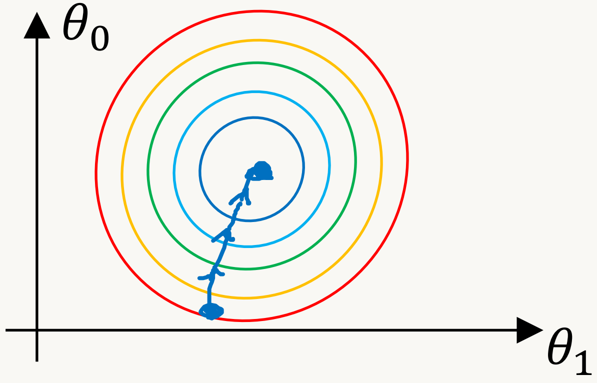
Unnormalized:



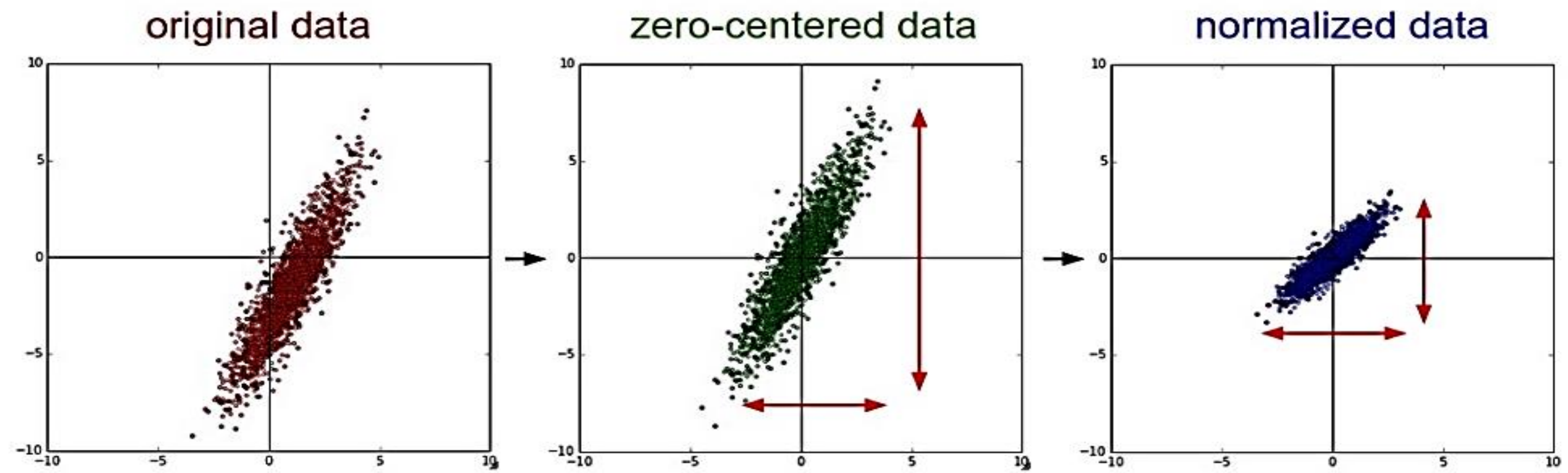

Normalized:



# Feature Normalization

- More illustrations




Common data preprocessing pipeline. **Left**: Original toy, 2-dimensional input data. **Middle**: The data is zero-centered by subtracting the mean in each dimension. The data cloud is now centered around the origin. **Right**: Each dimension is additionally scaled by its standard deviation. The red lines indicate the extent of the data - they are of unequal length in the middle, but of equal length on the right.

**Note**: The *feature normalization* must be applied to both instances from the **training** and **testing** sets

# More Tips

- <u>In Feature Engineering</u>

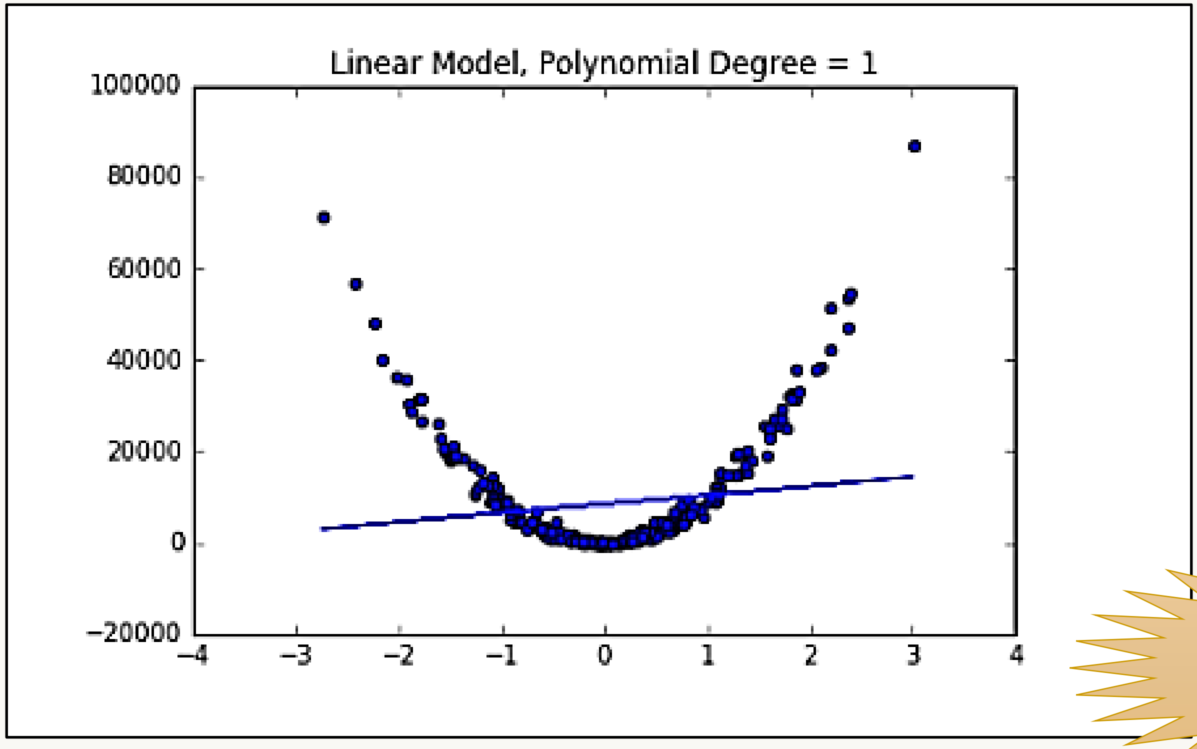
✓ Sometimes it might be fruitful to ***generate new features*** by combining the existing ones.

  - For example, given ***width*** and ***length*** of a property to predict ***price***

✓ It might be helpful to use ***area*** of the property, i.e., ***width * length*** as an additional feature.

# Polynomial Regression

# How does a linear model fit the data below ?





*Polynomial regression* is useful as it allows us to fit a model to nonlinear data.

# Polynomial regression

The concept of feature engineering can be used to achieve **polynomial regression**. Say the polynomial hypothesis chosen is,

$$h_\theta(x) = \theta_0 + \theta_1\, x + \theta_2\, x^2 + \cdots + \theta^n\, x^n$$

This function can be addressed as multivariate linear regression by substitution and is given by,

$$h_\theta(x) = \theta_0 + \theta_1\, x_1 + \theta_2\, x_2 + \cdots + \theta_n\, x_n$$

- Where $x_n = x^n$

# Polynomial regression (using multivariant regression )

Multi variant linear model

**Polynomial model**

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$$x_1 = (size)$$
$$x_2 = (size)^2$$
$$x_3 = (size)^3$$

Range →

| | |
|---|---|
| Size | 1-1000 |
| $size^2$ | 1-1000 000 |
| $size^3$ | 1- 1000 000 000 |

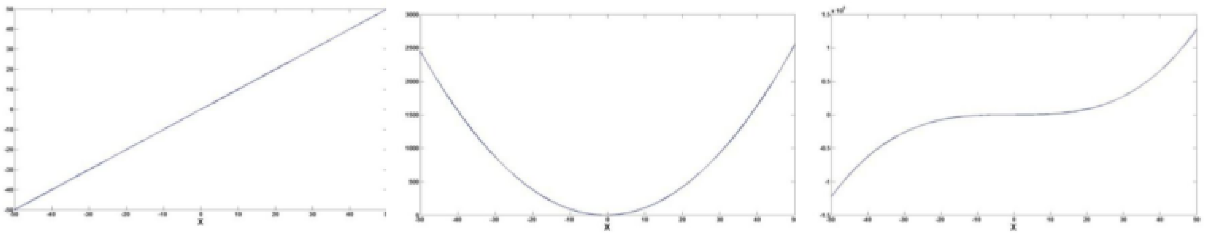

*Feature normalization* is very important as each feature has a different scale

**Note**:
If using features like this, then it is very important to ***apply feature scaling*** in order to avert issues related to feature range imbalance.

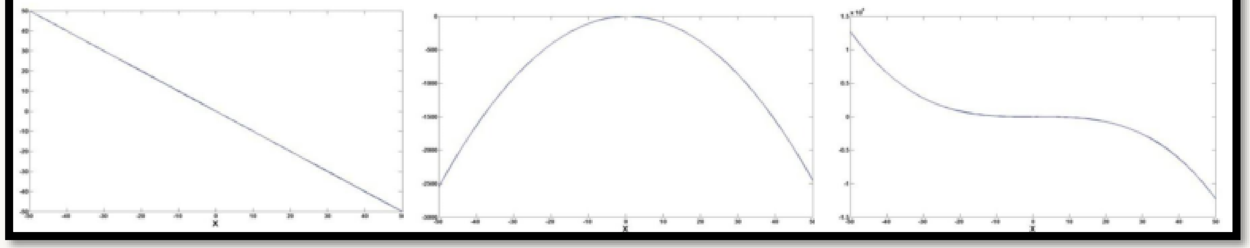
# Polynomial regression (Polynomial models examples)

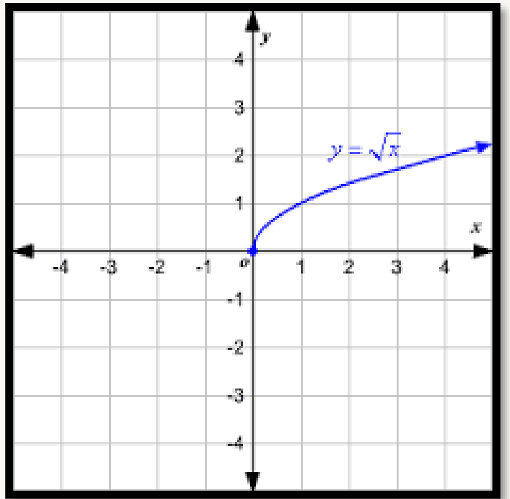
The sign of the coefficient for the highest order regressor determines the direction of the curvature

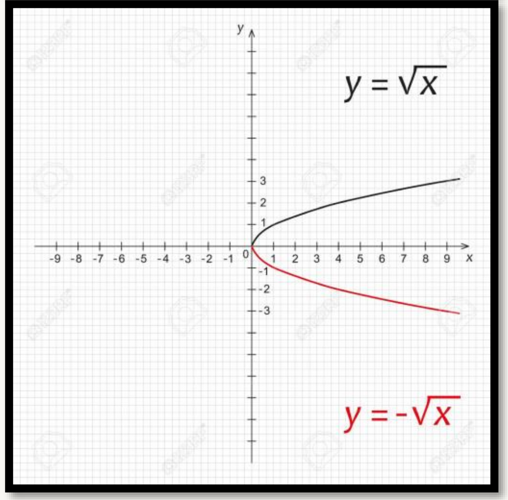
| Linear | Quadratic | Cubic |
|---|---|---|
| $Y' = 0 + 1X$ | $Y' = 0 + 1X + 1X^2$ | $Y' = 0 + 1X + 1X^2 + 1X^3$ |



| | | |
|---|---|---|
| $Y' = 0 + -1X$ | $Y' = 0 + 1X + -1X^2$ | $Y' = 0 + 1X + 1X^2 + -1X^3$ |

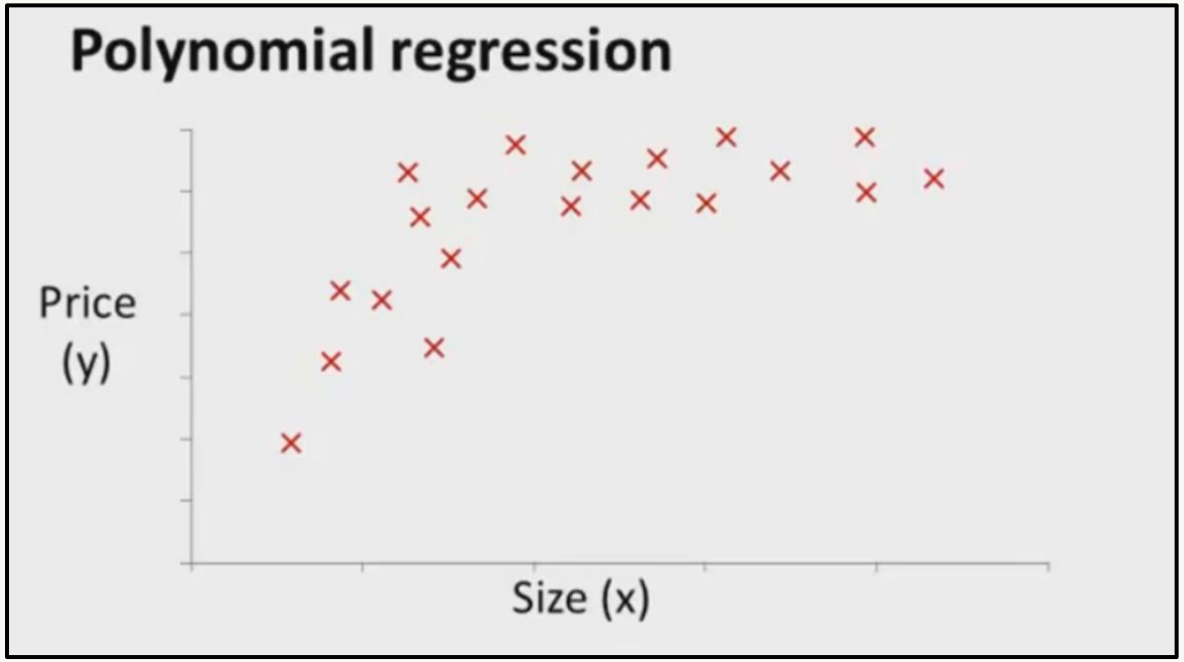




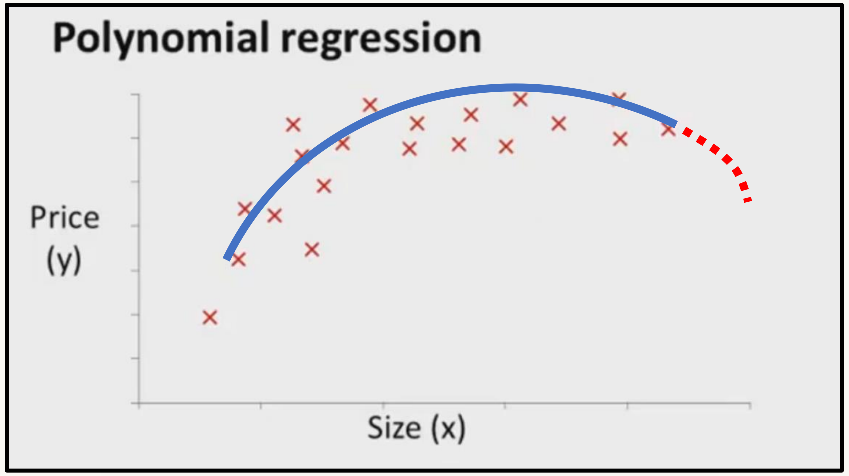
# Polynomial regression

- **Which model is the best ?**



# Polynomial regression

- **Which model is the best ?**

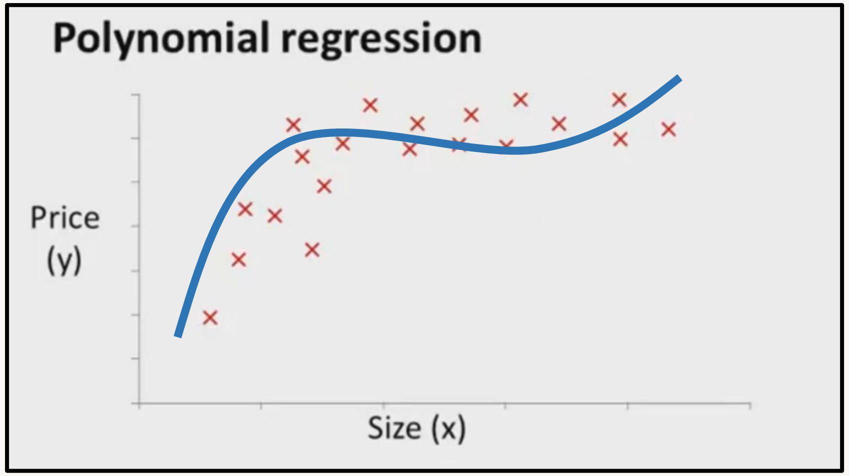



$$h\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

**Not accurate**: the price should be increased when size increases

# Polynomial regression

- **Which model is the best ?**

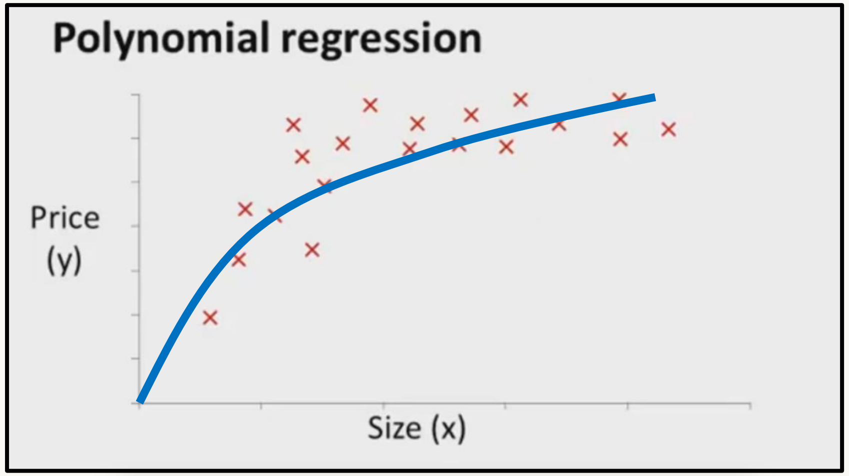



$$h\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

**Overfitting**: lack of generalization

# Polynomial regression

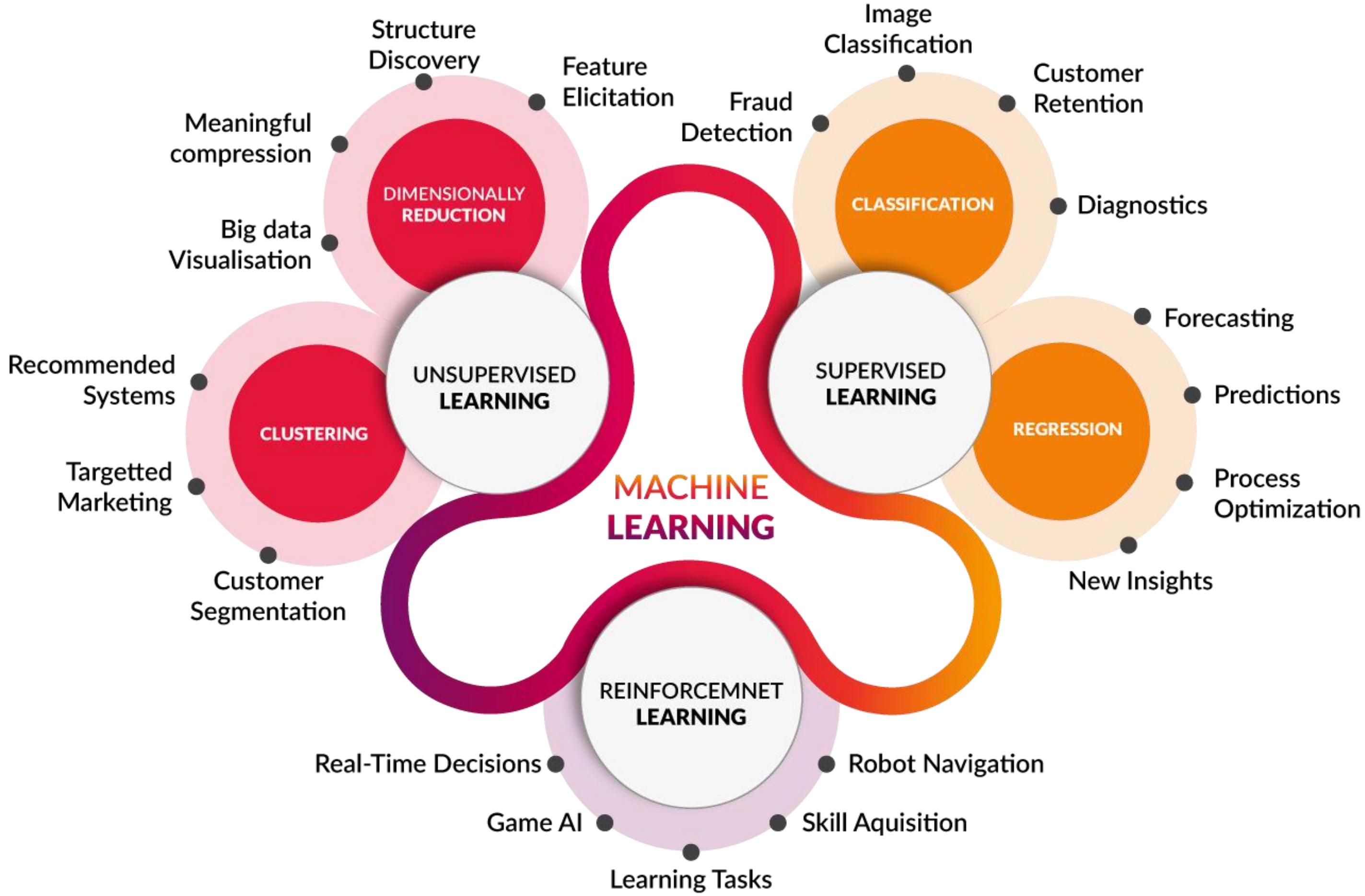
- **Which model is the best ?**




$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 \sqrt{x}$$

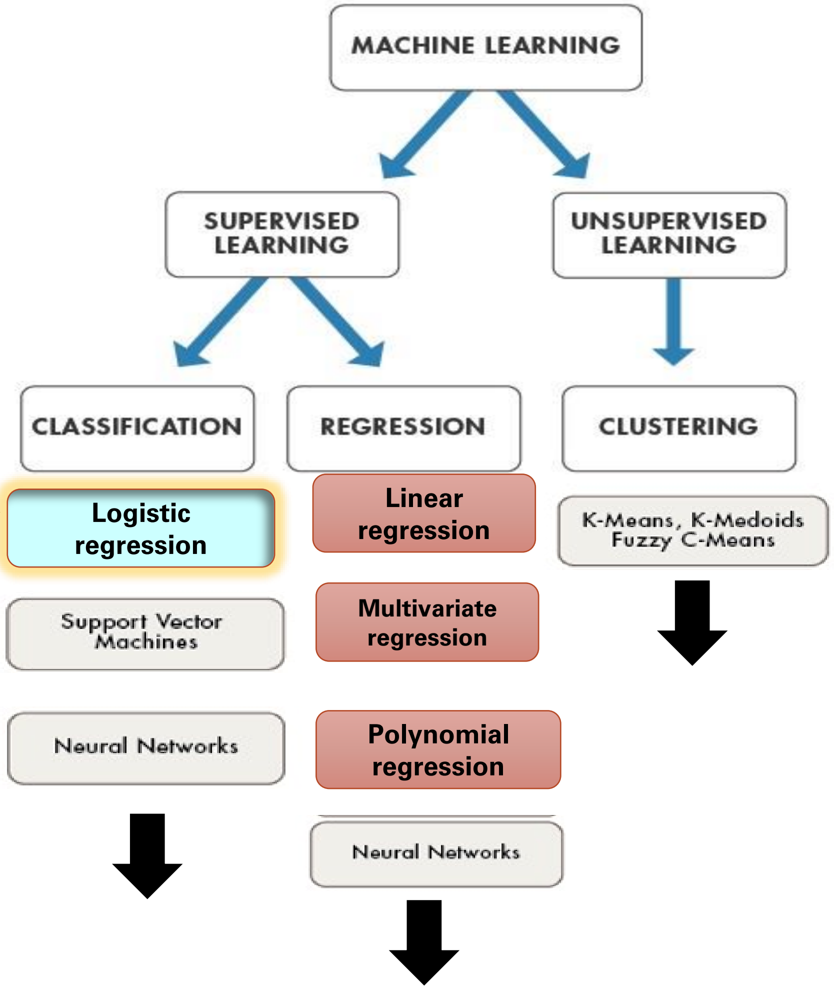
**Seems to be good**:
a non-decreasing function as opposed to quadratic function which comes back down

# Logistic Regression Model

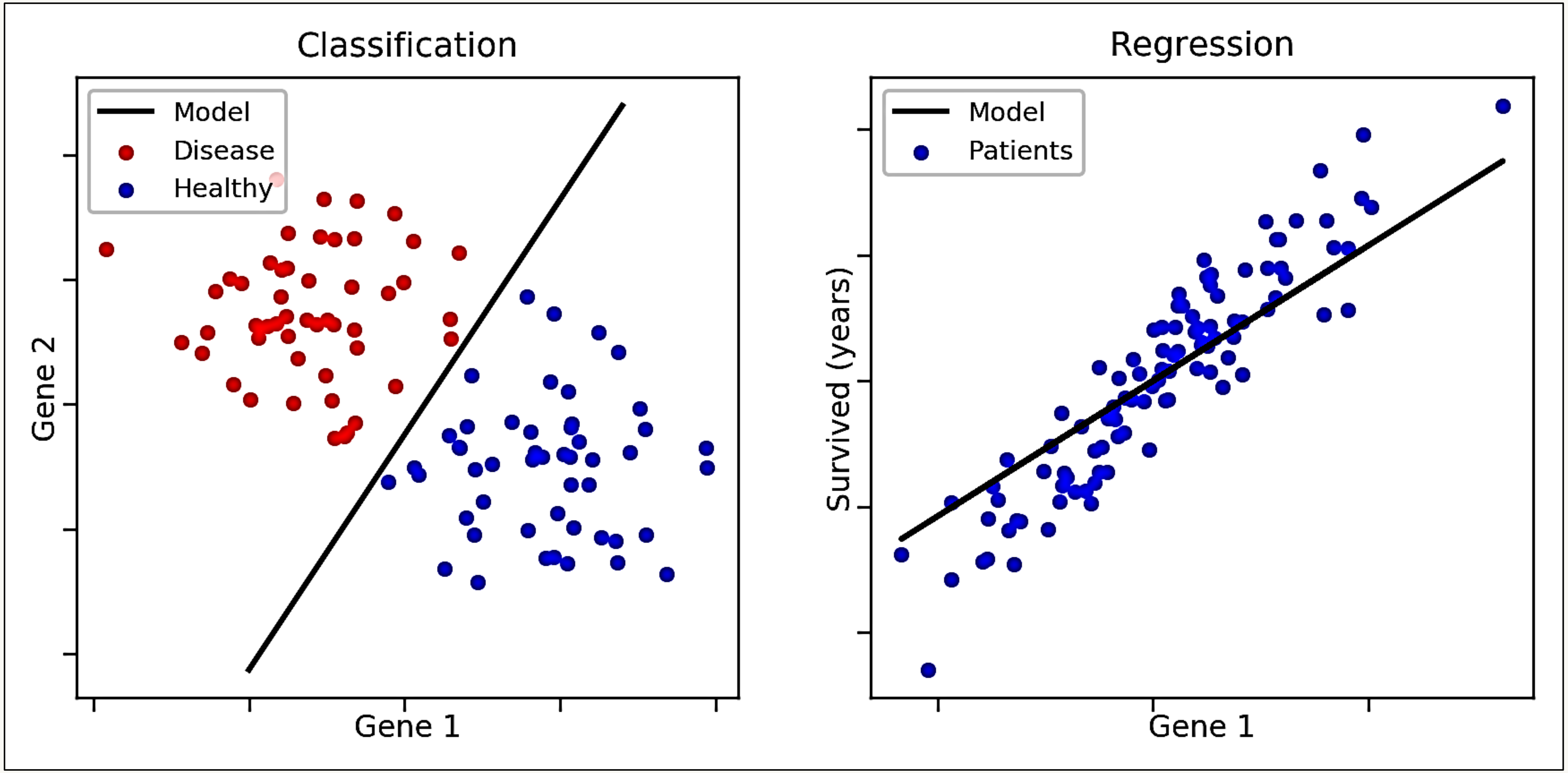

MACHINE LEARNING
UNSUPERVISED LEARNING
DIMENSIONALLY REDUCTION
Structure Discovery
Feature Elicitation
Meaningful compression
Big data Visualisation
CLUSTERING
Recommended Systems
Targetted Marketing
Customer Segmentation
SUPERVISED LEARNING
CLASSIFICATION
Image Classification
Customer Retention
Fraud Detection
Diagnostics
REGRESSION
Forecasting
Predictions
Process Optimization
New Insights
REINFORCEMNET LEARNING
Real-Time Decisions
Robot Navigation
Game AI
Skill Aquisition
Learning Tasks

# Note:

*Logistic regression* is used for classification not for regression (prediction) like linear/polynomial regression.

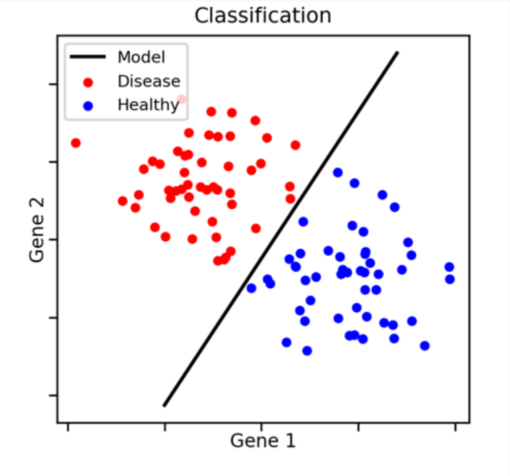


# Classification vs. Prediction (regression)



# Classification vs. Prediction (regression)

- Classification:
  - predicts categorical class labels
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Prediction:
  - models continuous-valued functions, i.e., predicts unknown or missing values



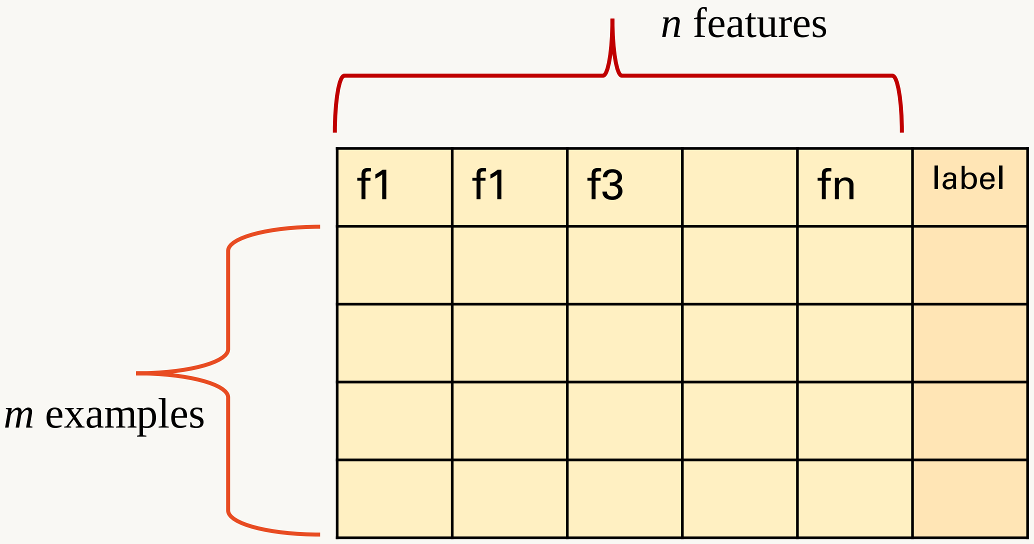

# Classification applications

- Typical Applications of Classification
  - credit approval
  - target marketing
  - medical diagnosis
  - treatment effectiveness analysis

- **Email**: Spam / Not Spam?
- **Online Transactions**: Fraudulent (Yes / No)?
- **Tumor**: Malignant / Benign ?

$label \in \{0,1\}$

0: "Negative Class" (e.g., benign tumor)
1: "Positive Class" (e.g., malignant tumor)




Training dataset

# Note:

- The output is the class label ➔ 0 or 1
- Our hypothesis should produce zero or one
- ML job is to find the weights for the features for a function where its output is zero or one
- And we need a model where: $0 \leq h_\theta(x) \leq 1$

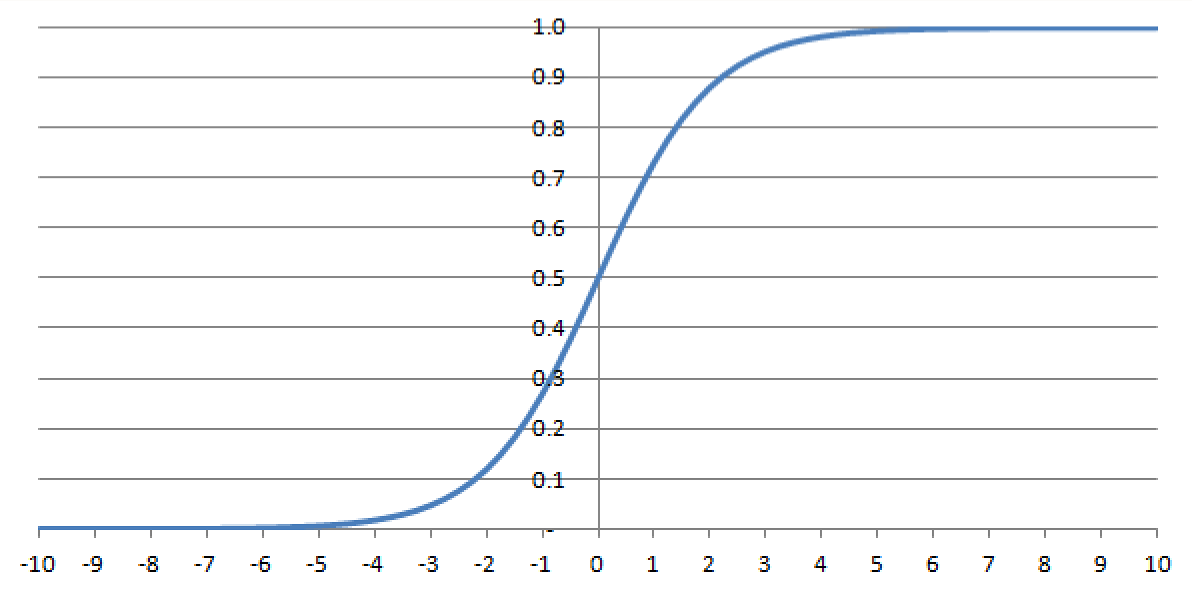
**Suggestion is to use the** *sigmoid function*

# Sigmoid Function or Logistic Function.

- *Plot of the sigmoid function* is given below which shows no matter what the value of ***z***, the function returns a value between 0 and 1

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

$g(z)$




$z$

# Logistic Regression

- for $0 \leq h_\theta(x) \leq 1$ is to be true, there is a need of **squashing function,** i.e., a function which limits the output of hypothesis between given range.
- For logistic regression **sigmoid function is used as the squashing function**.
- The hypothesis for logistic regression is give by: $h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

# Logistic Regression (Cont.)

- *The value of hypothesis* $h_\theta(x)$ *is interpreted as the probability that the input x belongs to class y=1, i.e., probability that y=1, given x, parametrized by θ.*

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

It can be mathematically represented as,

$$h_\theta(x) = P(y = 1|x;\theta)$$

The fundamental properties of probability holds here, i.e.,

$$P(y = 0|x;\theta) + P(y = 1|x;\theta) = 1$$

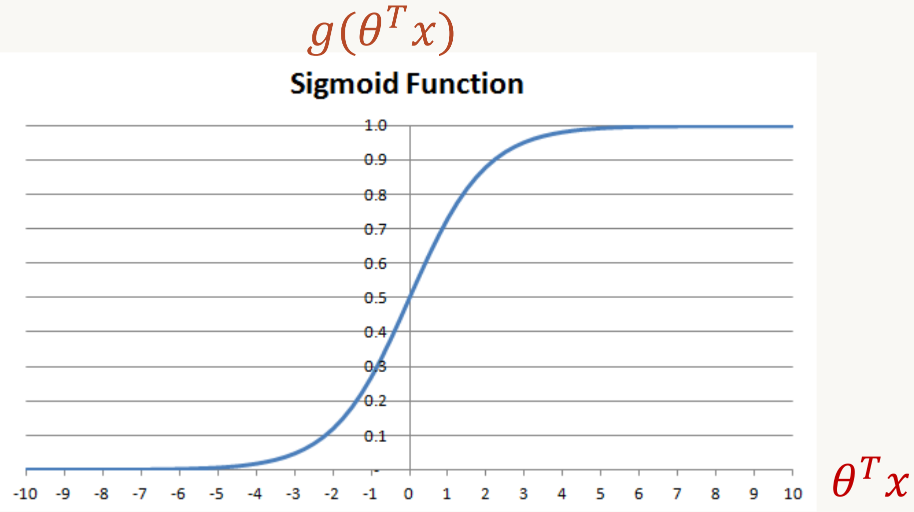
# Decision Boundary

# Decision Boundary

- For the given hypothesis of logistic regression, say *δ=0.5* is chosen as the **threshold for the binary classification.**

$label \in \{0,1\}$ 0: "Negative Class" 1: "Positive Class"

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$




$$\text{predict } y = 1, \text{ if } \theta^T x \geq 0$$
$$\text{predict } y = 0, \text{ if } \theta^T x < 0$$

This is the decision boundary

# Linear Decision Boundary

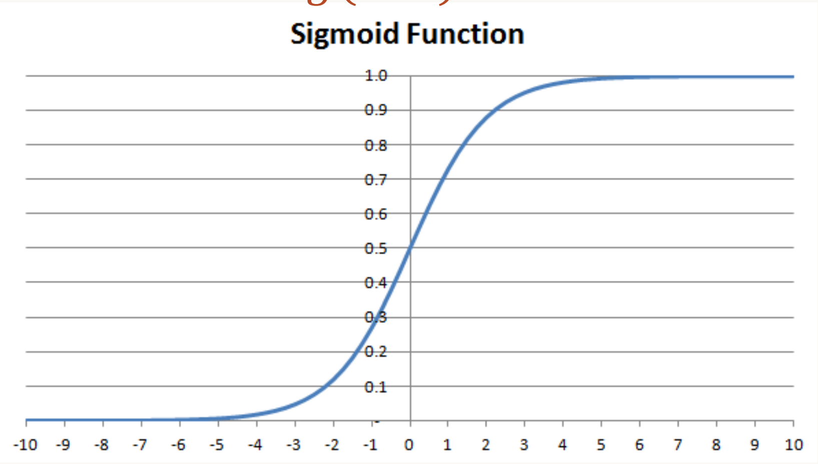
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

*In this example*
$\theta^T x = -12 + x_1 + x_2$

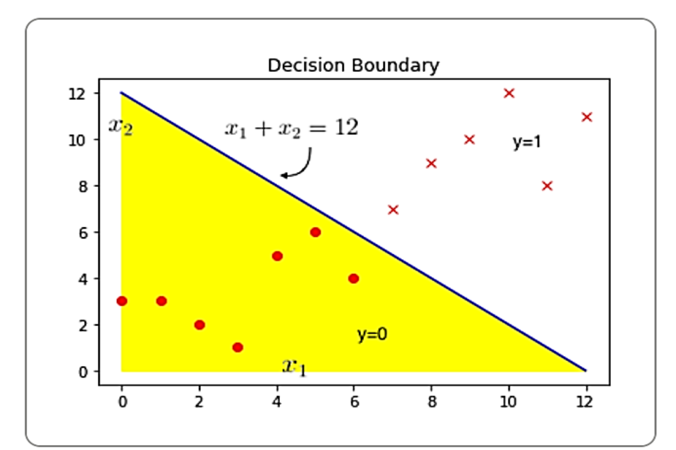
predict $y = 1$, if $\theta^T x \geq 0$
predict $y = 0$, if $\theta^T x < 0$

predict $y = 1$, if $-12 + x_1 + x_2 \geq 0$ or $x_1 + x_2 \geq 12$
predict $y = 0$, if $-12 + x_1 + x_2 < 0$ or $x_1 + x_2 < 12$

$g(\theta^T x)$



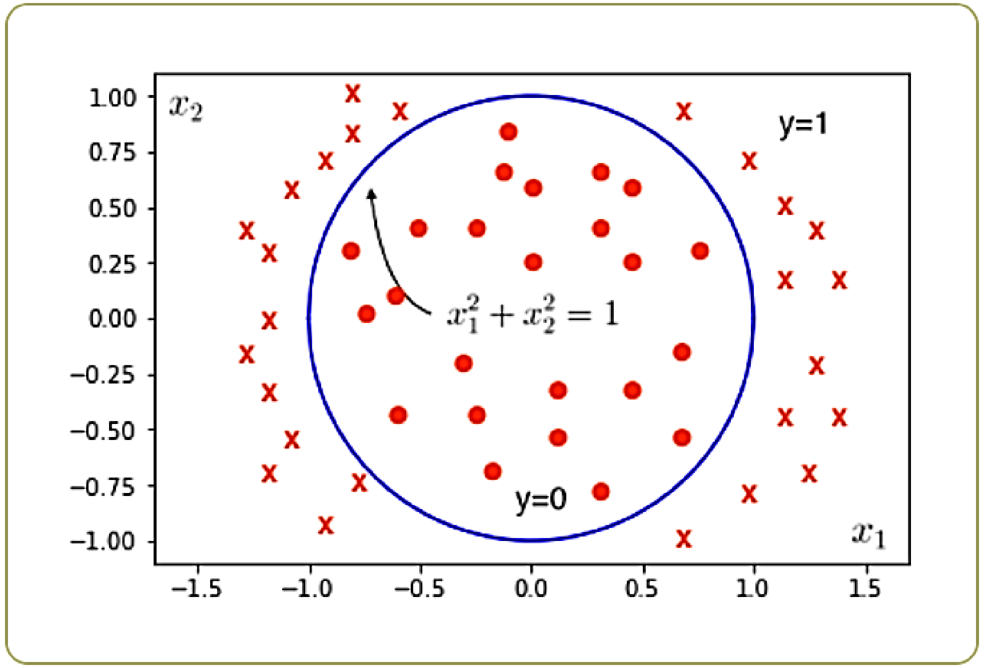

$\theta^T x$



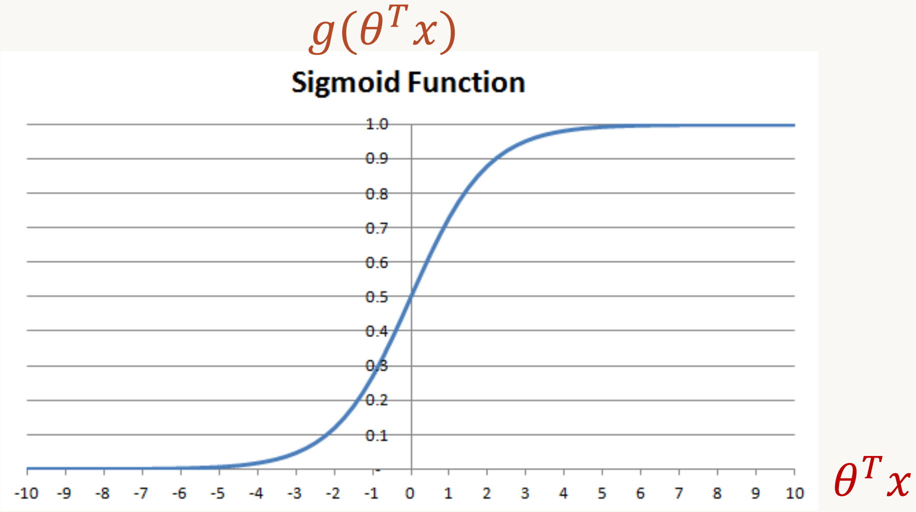

*Decision Boundary*

# Nonlinear Decision Boundary

- It is possible to achieve *non-linear decision boundaries* by using the higher order polynomial terms and can be incorporated in a way similar to how multivariate linear regression handles polynomial regression.




*Non-Linear Decision Boundary*



# Nonlinear Decision Boundary (Cont.)

Say, the hypothesis of the logistic regression has higher order polynomial terms, and is given by,

$\boldsymbol{\theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2}$

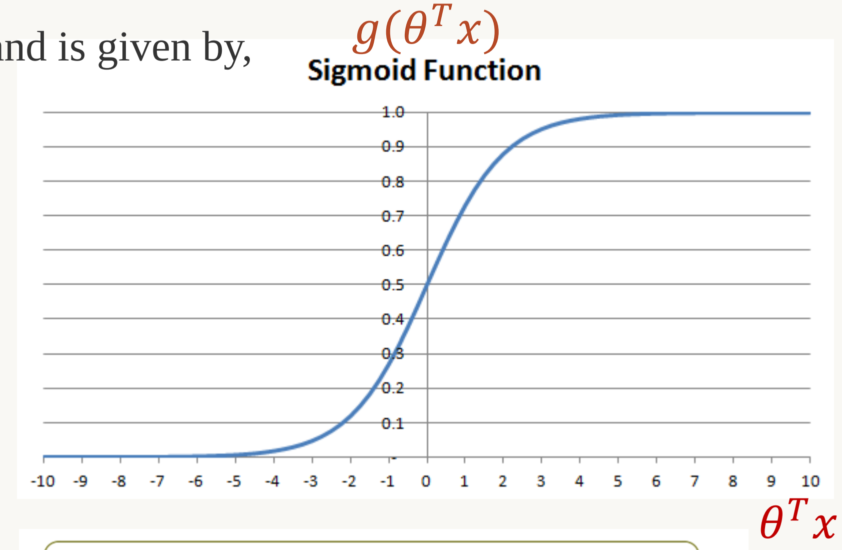
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$

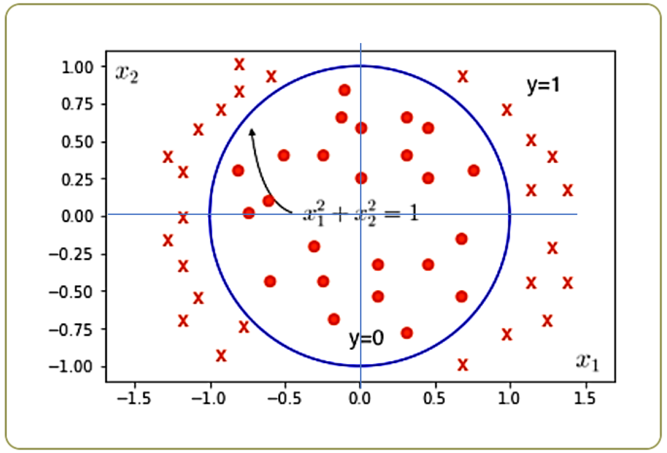
let optimal θ given below would form an optimal decision boundary
$\theta^T$ = [-1 0 0 1 1]

Substituting $\theta^T x = -1 + x_1^2 + x_2^2$

Decision boundary is $x_1^2 + x_2^2 = 1$

predict $y = 1$, if $-1 + x_1^2 + x_2^2 \geq 0$ or $x_1^2 + x_2^2 \geq 1$
predict $y = 0$, if $-1 + x_1^2 + x_2^2 < 0$ or $x_1^2 + x_2^2 < 1$

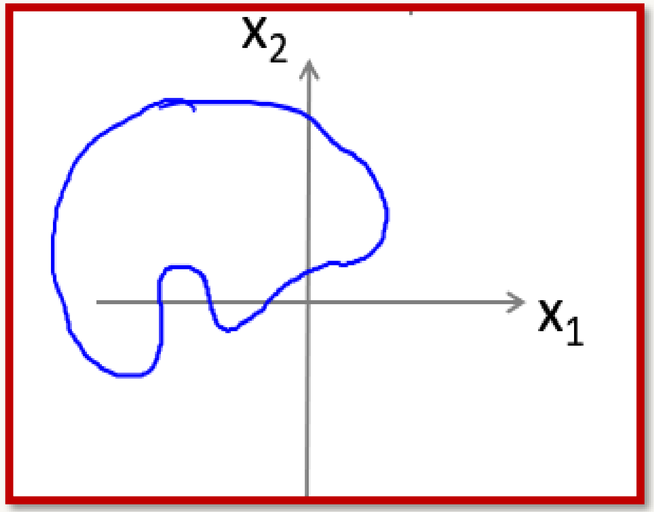


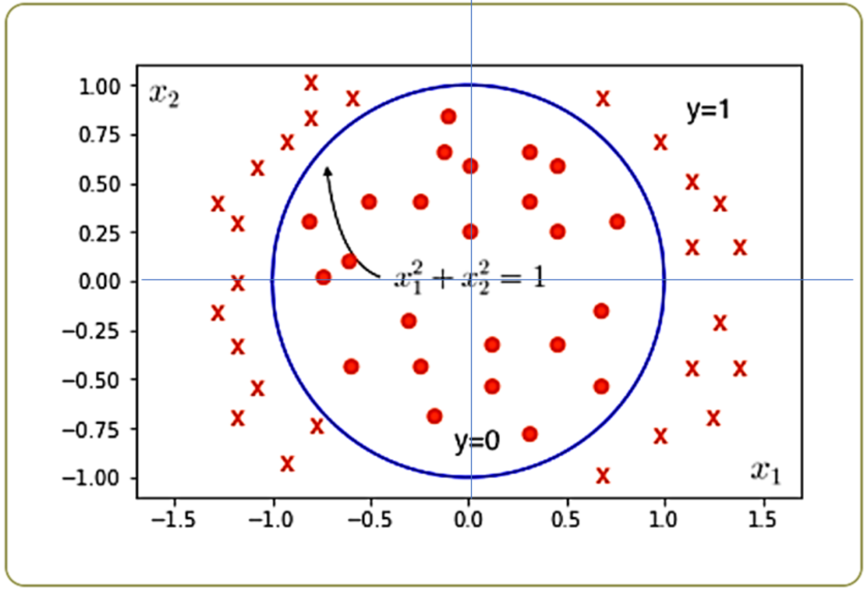

*Non-Linear Decision Boundary*

# Nonlinear Decision Boundary (Cont.)

- As the order of features is increased more and more **complex decision** boundaries can be achieved by logistic regression. *Be aware of overfitting !!*




$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$




*Non-Linear Decision Boundary*

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

**Gradient Descent is used to search for the best parameter values of θ**
that make the decision boundary

# Logistic Regression Cost Function

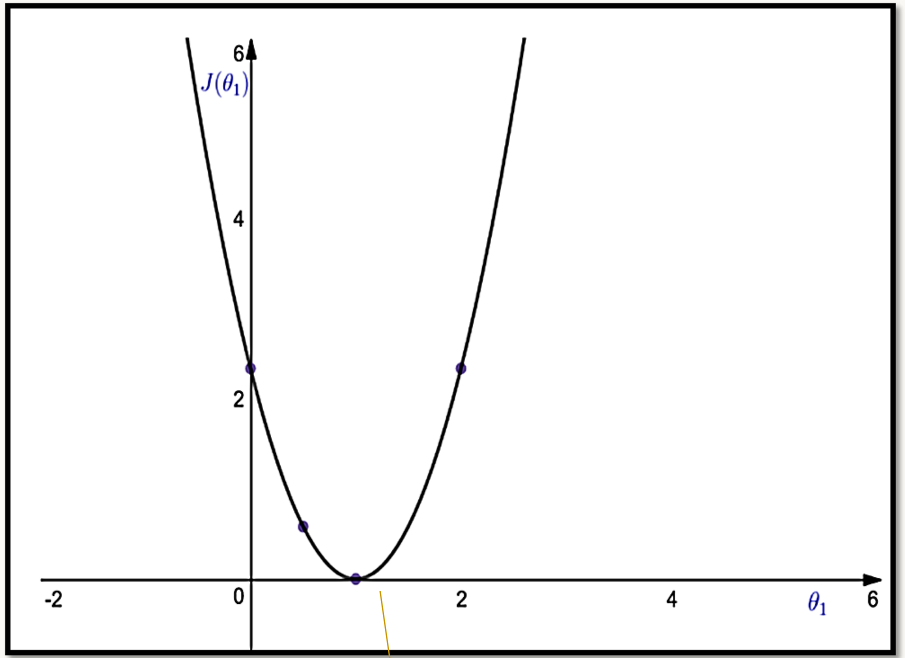
# Logistic Regression (Cost Function)

**Recall that previously in linear regression:**

It can be seen in Mulivariate Linear Regression that the cost function for the linear regression is given by,

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$
$$= \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$



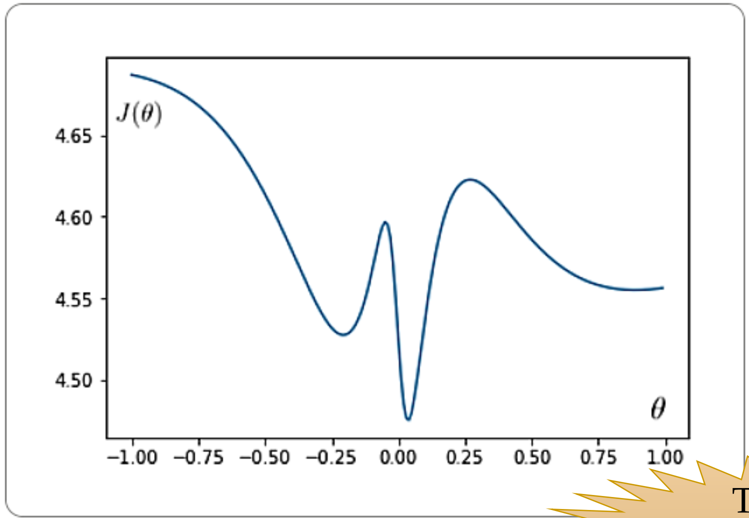

The Cost function is convex

# Logistic Regression (Cost Function) (Cont.)

- The same cost function of multivariate regression would not work well for the logistic regression because the hypothesis for logistic regression is the complex sigmoid function
- Below, gives **non-convex** curve with many **local minima** as shown in the plot below.
- So, gradient descent will not work properly for such a case and therefore it would be very difficult to minimize this function.

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left( h_\theta(x^{(i)}) - y^{(i)} \right)^2$$
$$= \frac{1}{m} \sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$



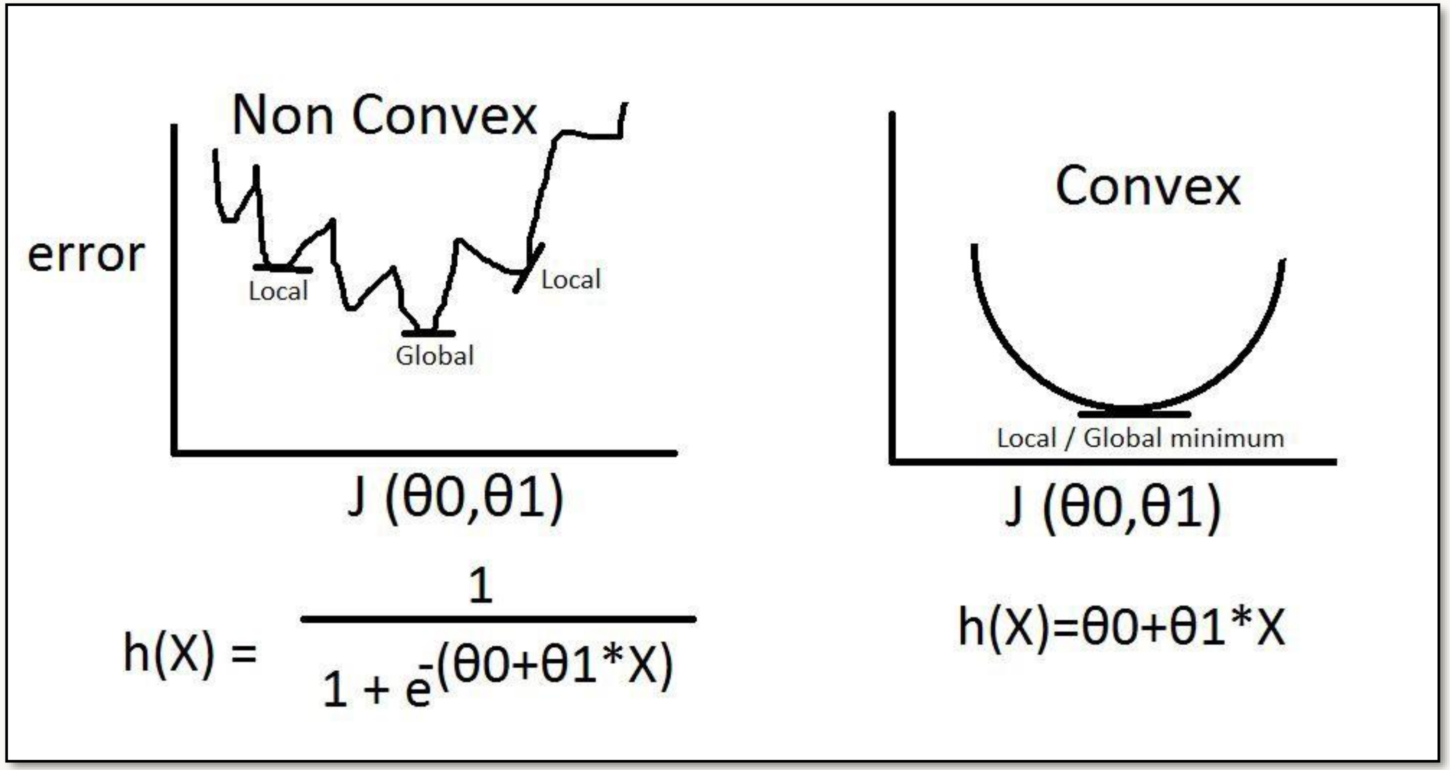

Non Convex Cost

This cost function is NOT convex

We need to use another cost function which is convex!

# Logistic Regression (Cost Function) (Cont.)

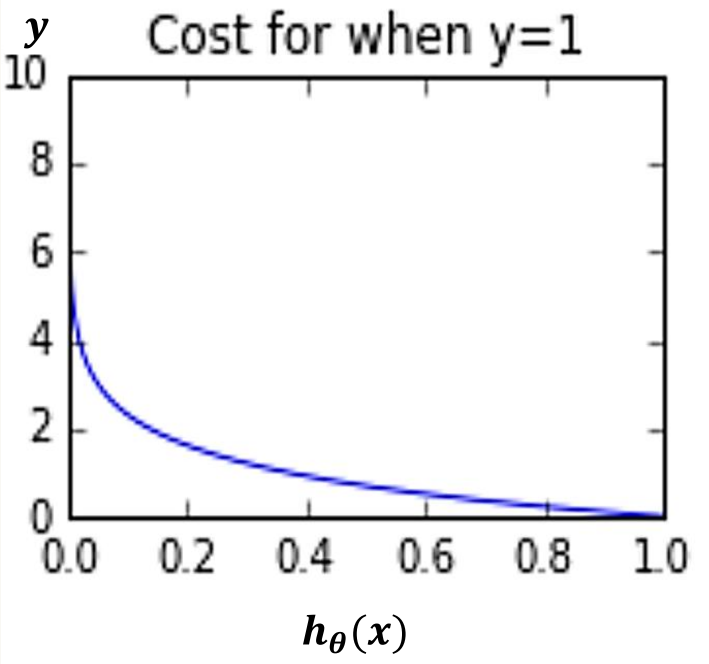
- **More illustration:**

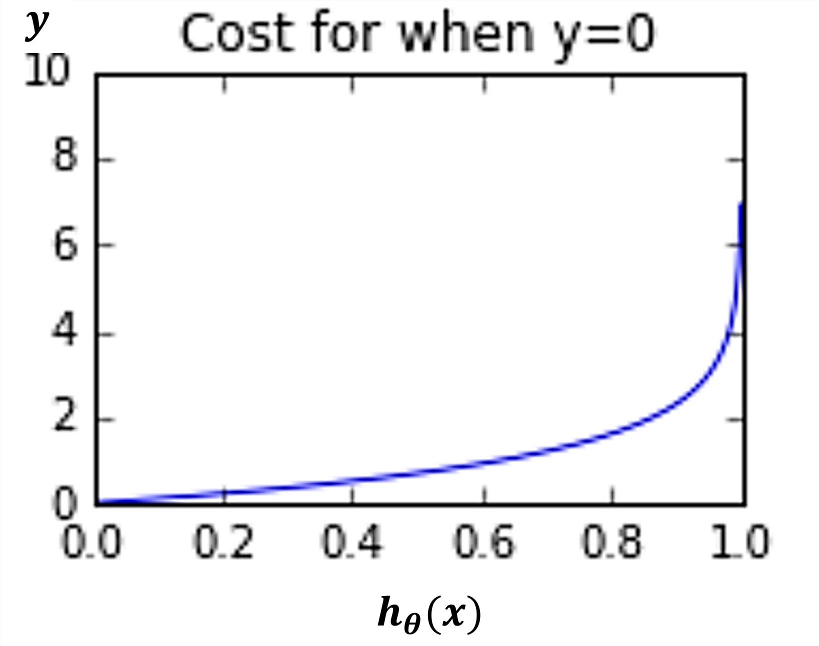


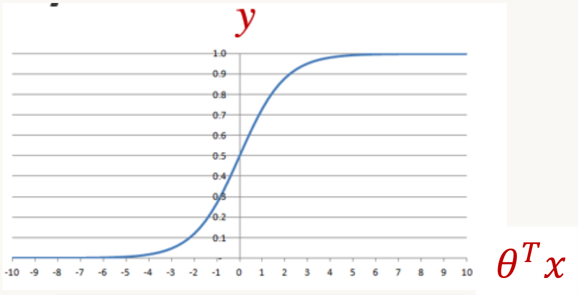
# Logistic Regression (Cost Function) (Cont.)

So, **cost function for logistic regression** is given by,

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \text{ if } y = 1 \\ -log(1 - h_\theta(x)) \text{ if } y = 0 \end{cases}$$



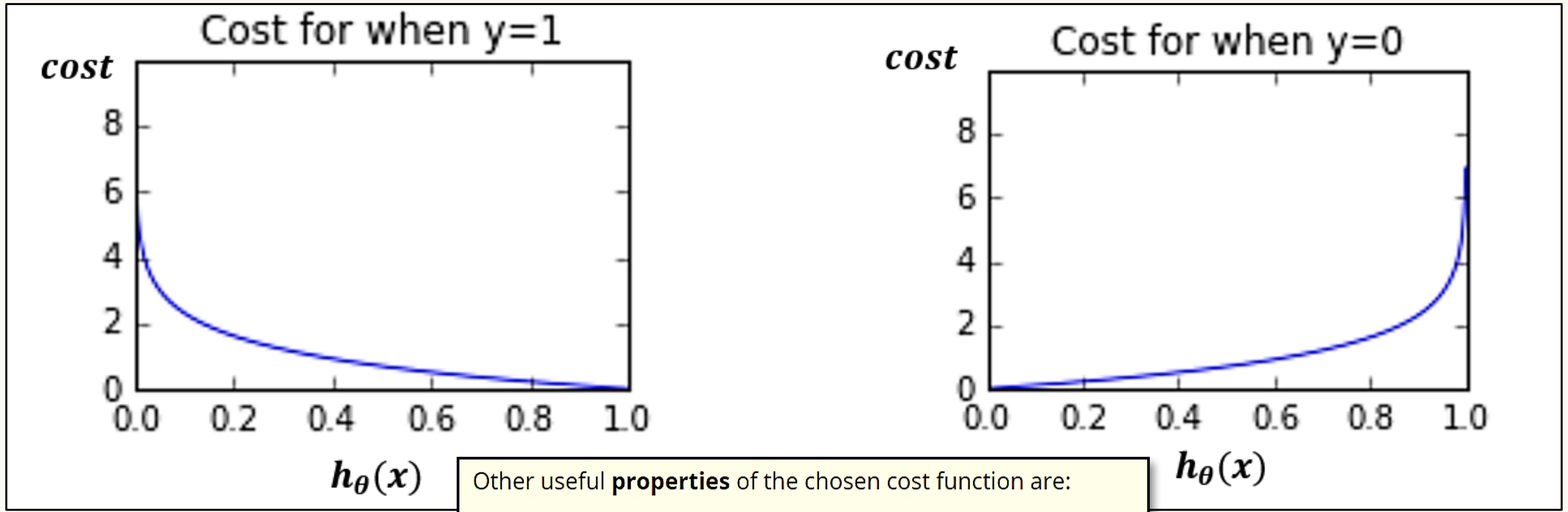

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

It is clear that new *cost function* *can be minimized because its convex*.

# Logistic Regression (Cost Function) (Cont.)

- *This cost function is reached at using the* ***principle of maximum likelihood expectation****.*

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \text{ if } y = 1 \\ -log(1 - h_\theta(x)) \text{ if } y = 0 \end{cases}$$



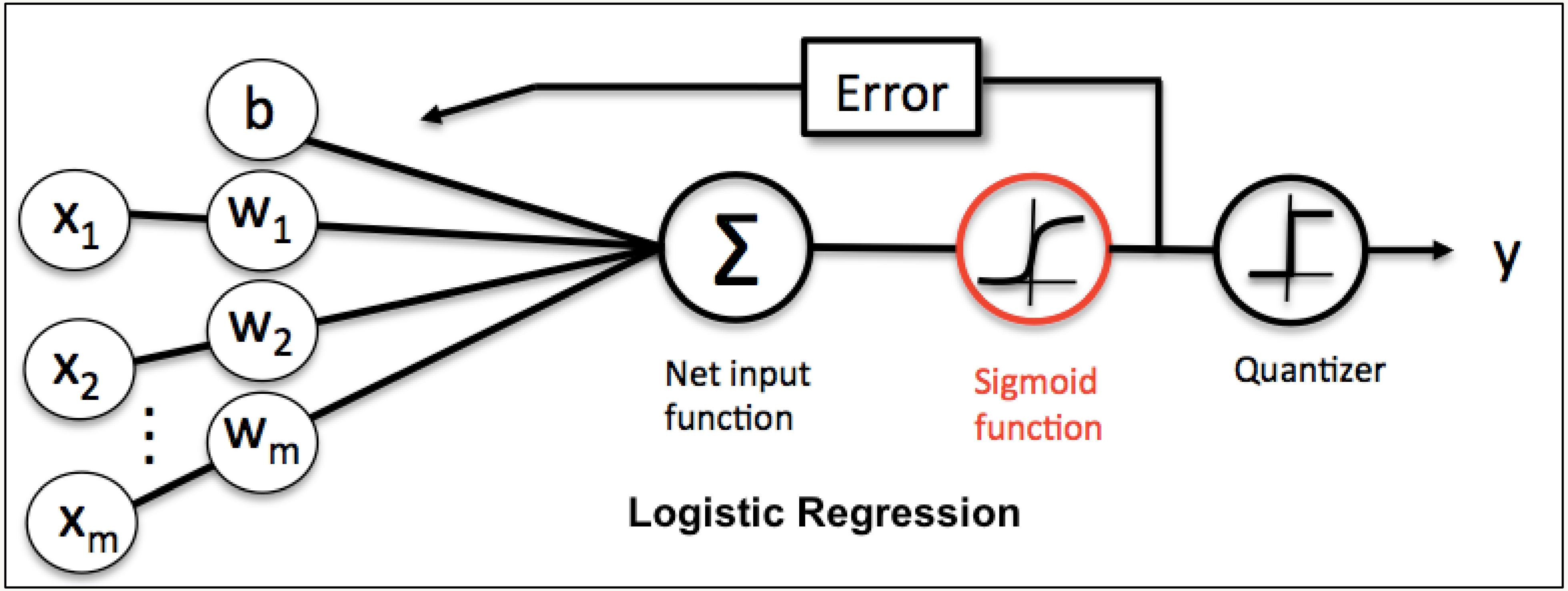

Other useful **properties** of the chosen cost function are:

- if y = 1 and
  - h(x) = 1, then Cost = 0
  - h(x) $\rightarrow$ 0, then Cost $\rightarrow \infty$
- if y = 0 and
  - h(x) = 0, then Cost = 0
  - h(x) $\rightarrow$ 1, then Cost $\rightarrow \infty$

*Y* is the actual value
*h(x)* is the predicted value

# Logistic Regression (Cost Function) (Cont.)

**Summary:**



# Gradient Descent for Logistic Regression

# Gradient Descent for Logistic Regression

$$J(\theta) = -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

$$Cost(h_\theta(x), y) = \begin{cases} -log(h_\theta(x)) \text{ if } y = 1 \\ -log(1 - h_\theta(x)) \text{ if } y = 0 \end{cases}$$

Want $\min_\theta J(\theta)$ :

Repeat $\{ \ \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta) \ \}$ (simultaneously update all $\theta_j$)

Repeat $\{ \ \theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \ \}$ (simultaneously update all $\theta_j$)

Algorithm looks identical to linear regression!

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \text{ and } x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

**Note:** *Feature Scaling* is as important for logistic regression as it is for linear regression as it helps the process of gradient descent.

QUICK TIP

# Gradient Descent for Logistic Regression (Cont.)

## Advanced Optimization

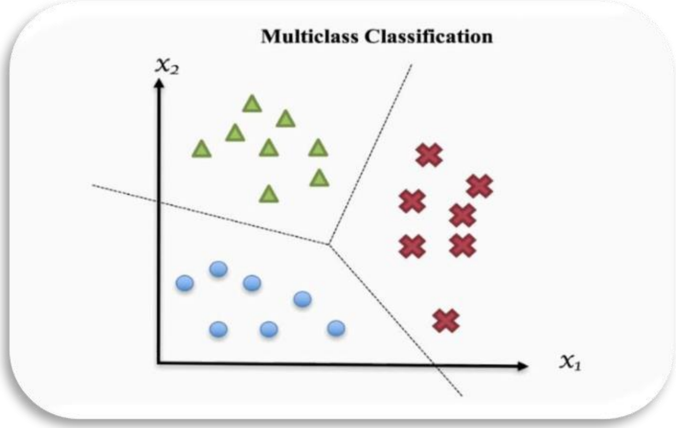
Given the functions for calculation of $J(\theta)$ and $\frac{\partial}{\partial\theta}J(\theta)$ one can apply one of the many **optimization techniques other than gradient descent**:

- Conjugate Descent
- BFGS
- L-BFGS

| Advantage | Disadvantages |
|---|---|
| No need to manually pick $\alpha$ | More complex |
| Often faster than gradient descent | Harder to debug |

*These algorithms automatically find out the best α value.*

# Multiclass Logistic Regression



# Multiclass Logistic Regression

- Multiclass logistic regression is an extension of the binary classification making use of the **one-vs-all** or **one-vs-rest** classification strategy.

## Intuition

Given a classification problem with **n** distinct classes, train n classifiers, where each classifier draws a decision boundary for one class vs all the other classes. Mathematically,

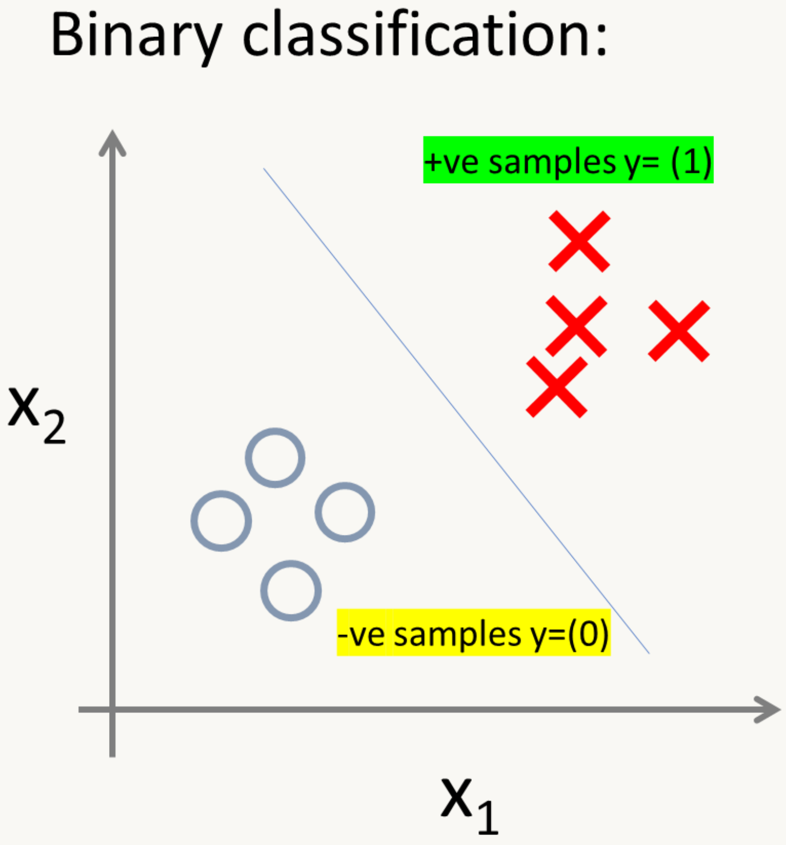
$$h_\theta^{(i)}(x) = P(y = i|x; \theta)$$

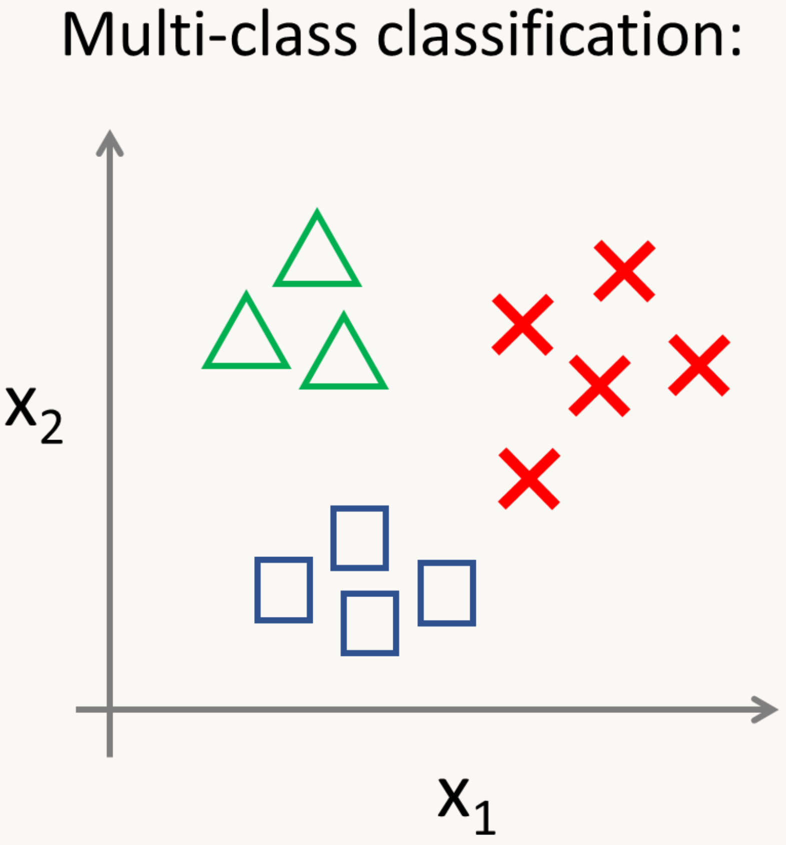
# Multiclass Logistic Regression (Cont.)

For Example:

- Email foddering/tagging: Work, Friends, Family, Hobby
  y=1 y=2 y=3 y=4
- Medical diagrams: Not ill, Cold, Flu
  y=1 y=2 y=3
- Weather: Sunny, Cloudy, Rain, Snow
  y=1 y=2 y=3 y=4

# Multiclass Logistic Regression (Cont.)



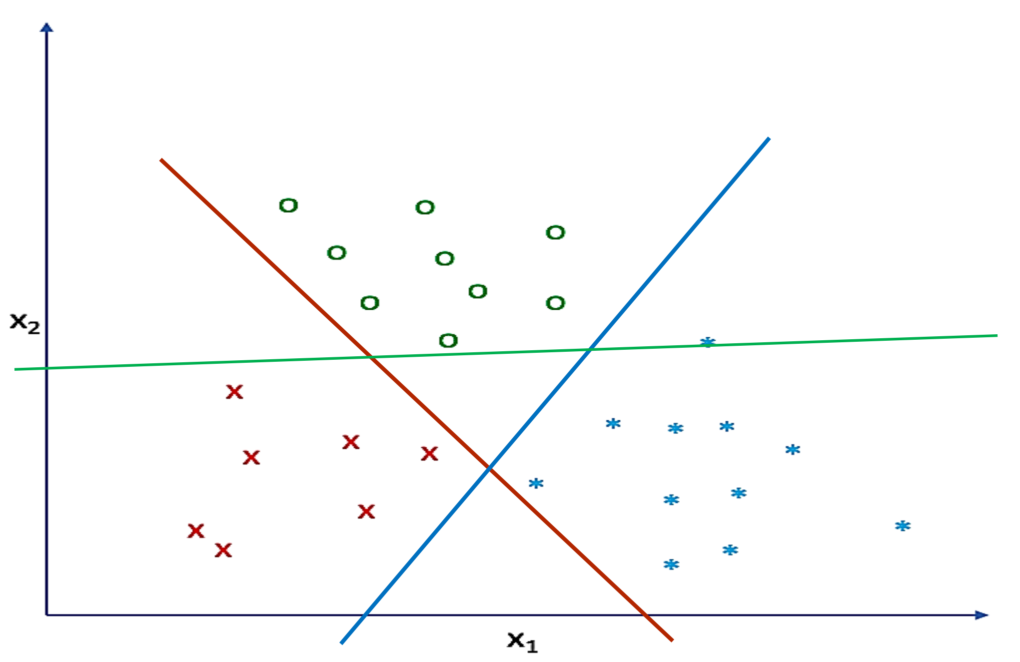



# Multiclass Logistic Regression (Cont.)

$$x \in \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad y \in \{1, 2, \dots, k\}$$

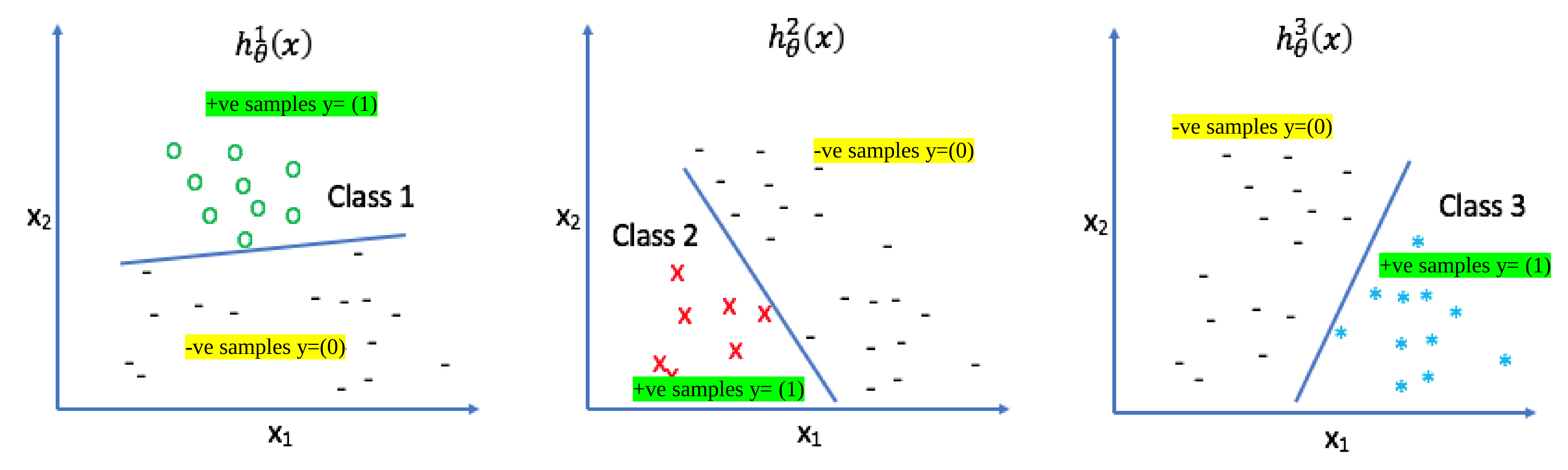
For a dataset with $k$ classes, you'd train a collection of $k$ classifiers.

$$h_\theta^1(x), h_\theta^2(x), \dots h_\theta^k(x)$$



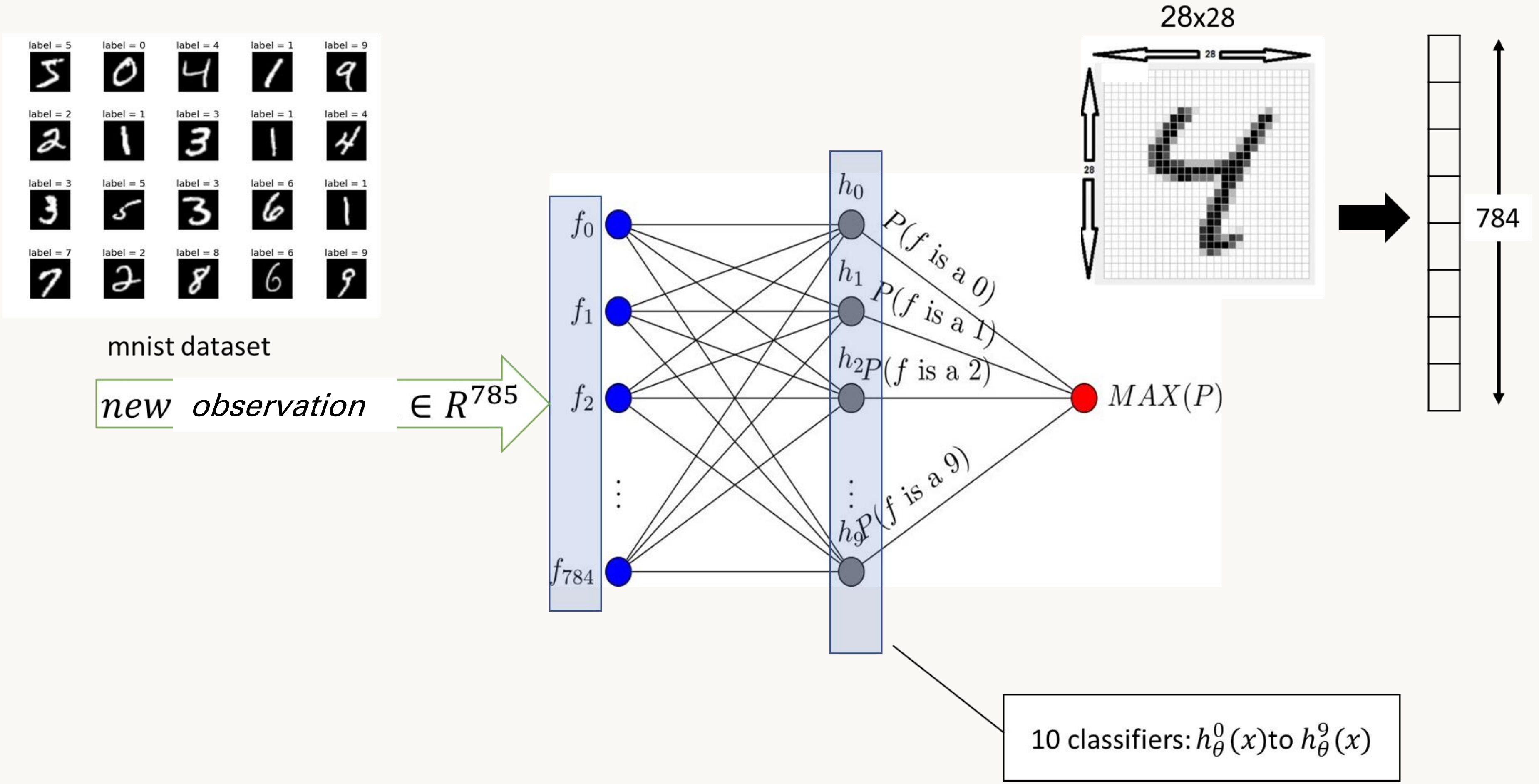

We have three classes

# Multiclass Logistic Regression (Cont.)

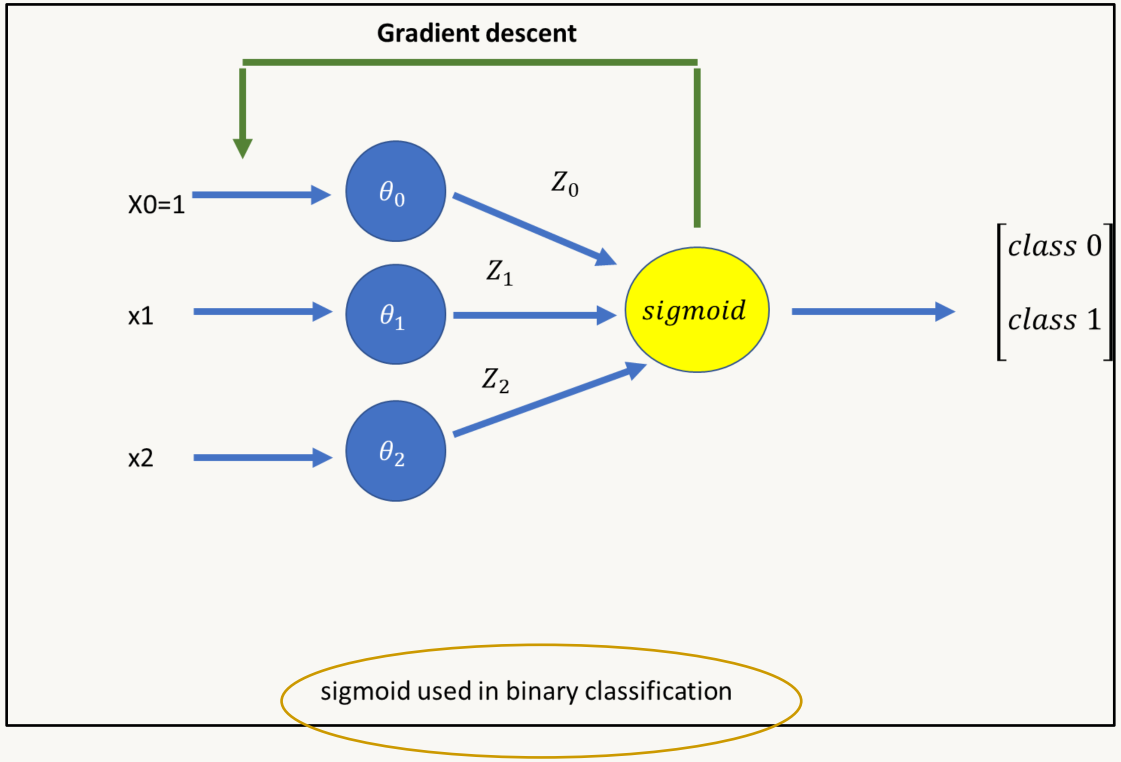


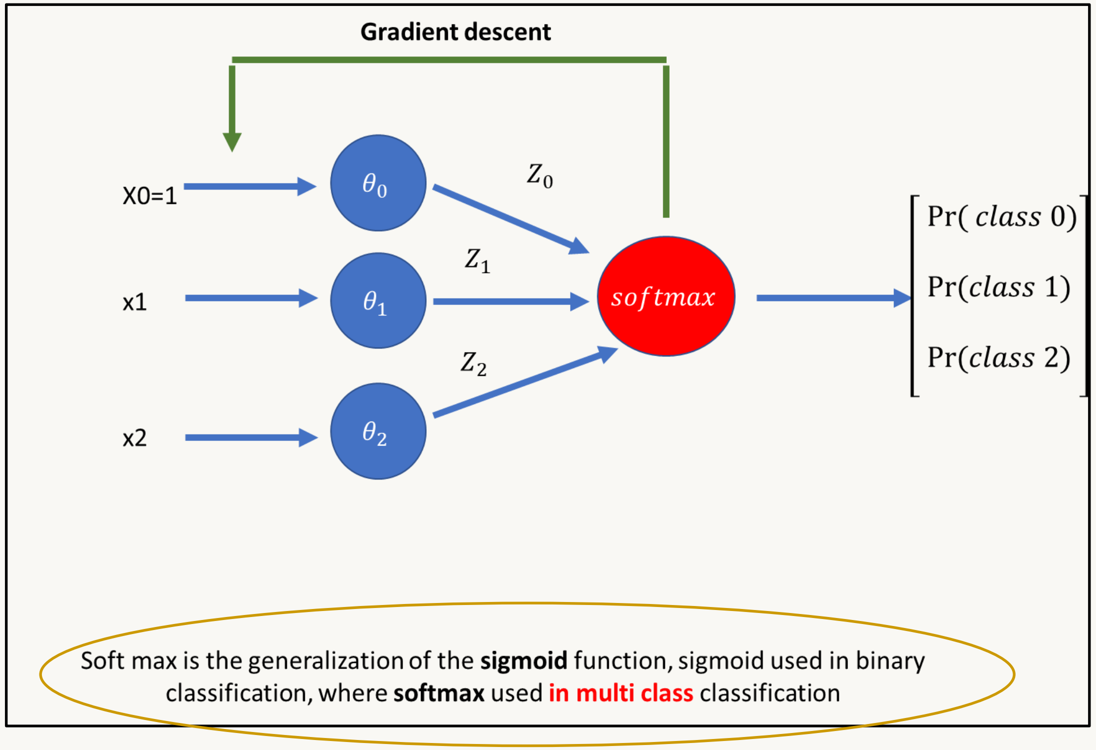

- Each classifier $h^i_\theta(x)$ returns the probability that an observation belongs to **class i.**
- All we have to do in order to predict the class of an ***observation*** is to select the class of whichever classifier returns the **highest probability**.

# Multiclass Logistic Regression (Cont.)



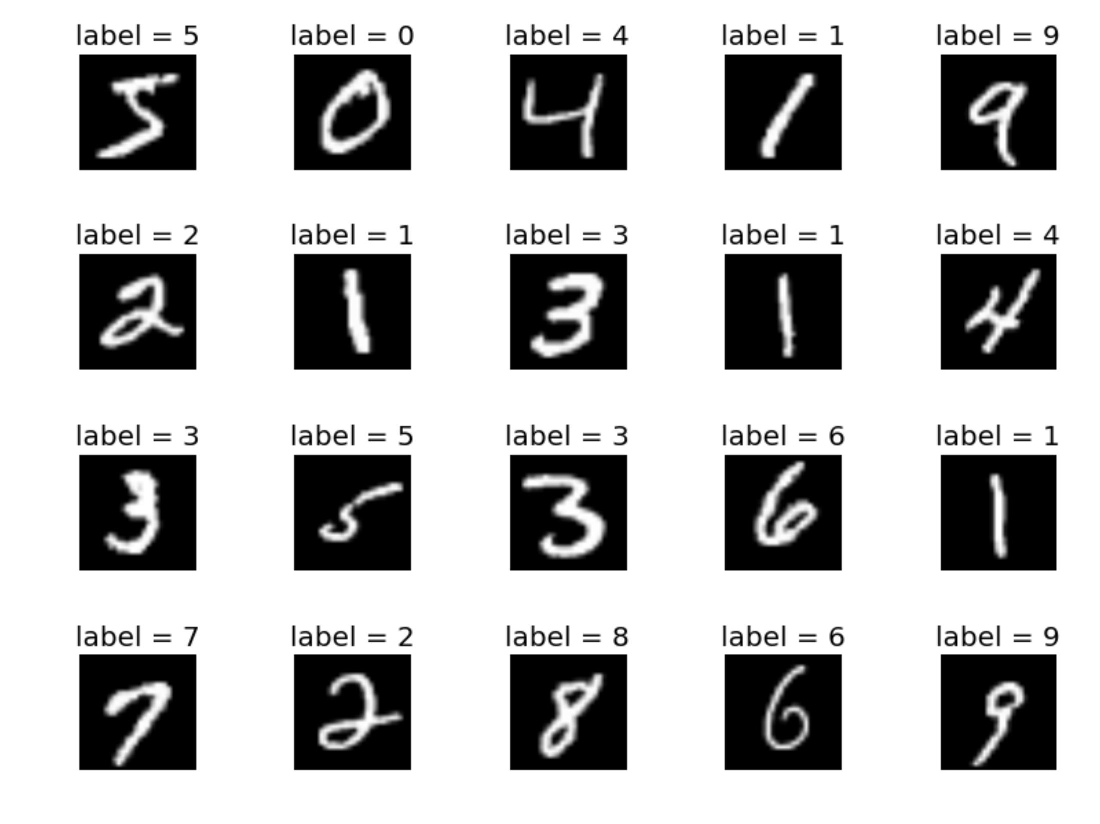
# Multiclass Logistic Regression (Squashing Function)



# Multiclass Logistic Regression (Cont.)

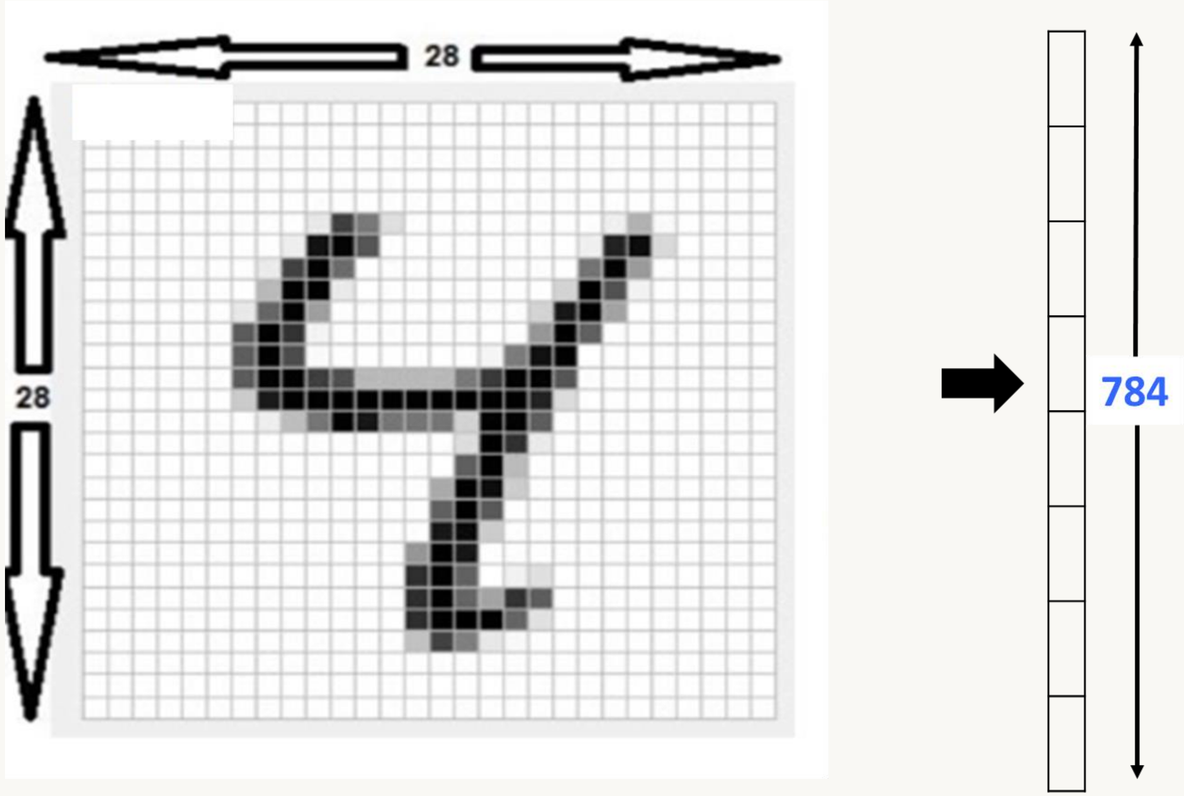
For Example:




mnist dataset

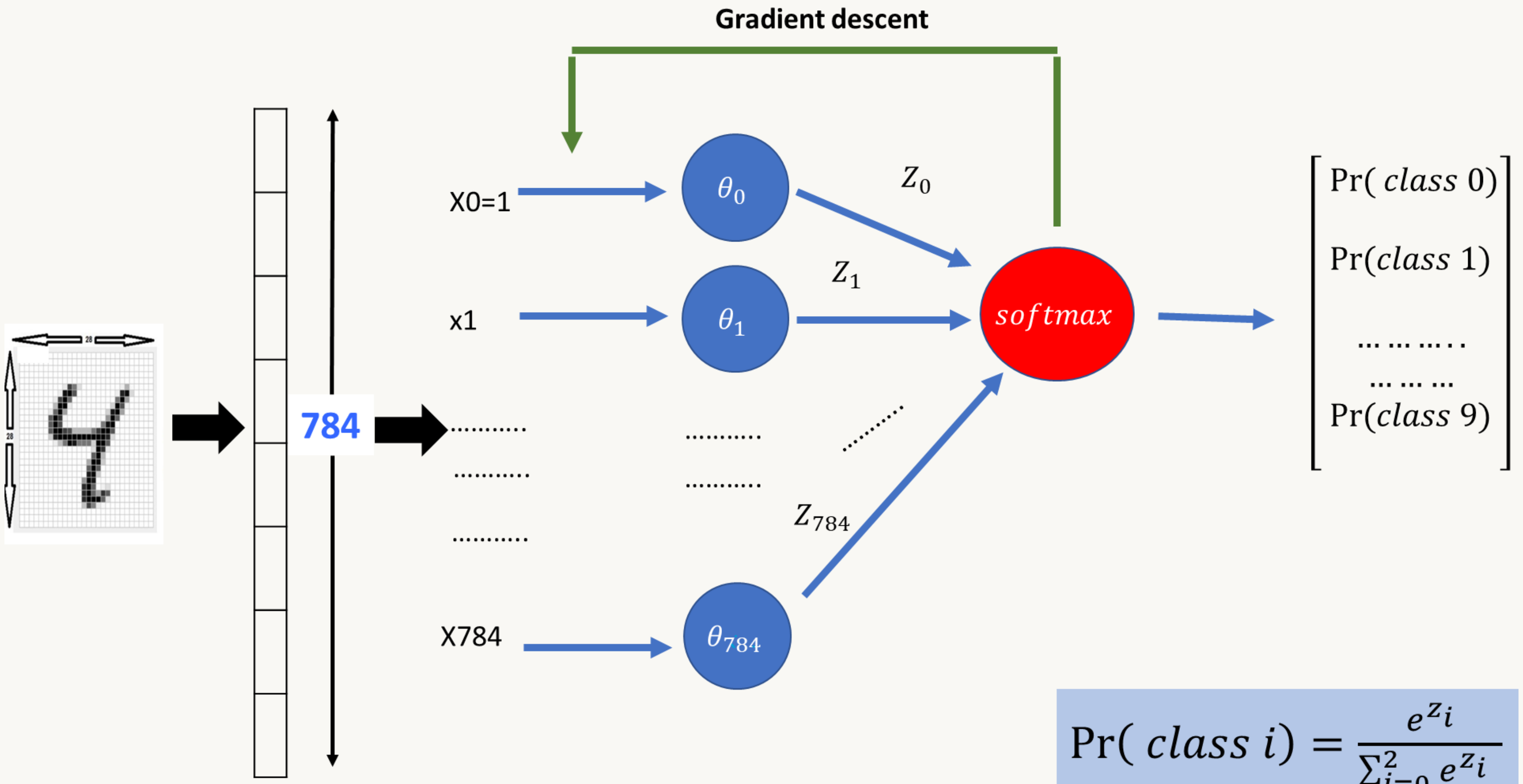
# Multiclass Logistic Regression (Cont.)

For Example:



# Multiclass Logistic Regression (Cont.)

For Example:




$$\Pr(\ class\ i) = \frac{e^{z_i}}{\sum_{i=0}^{2} e^{z_i}}$$

# Summary

**Implementation Note:** In the multivariate case, the cost function can also be written in the following vectorized form:

$$J(\theta) = \frac{1}{2m}(X\theta - \vec{y})^T (X\theta - \vec{y})$$

where

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix} \qquad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}.$$

The vectorized version is efficient when you're working with numerical computing tools like Octave/MATLAB. If you are an expert with matrix operations, you can prove to yourself that the two forms are equivalent.

- Multi-class Classification (Useful videos):
  - ✓ https://www.youtube.com/watch?v=LLux1SW--oM
  - ✓ https://www.youtube.com/watch?v=ueO_Ph0Pyqk

***Any Question?***