Supervised:Logistic Regession

Classification

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

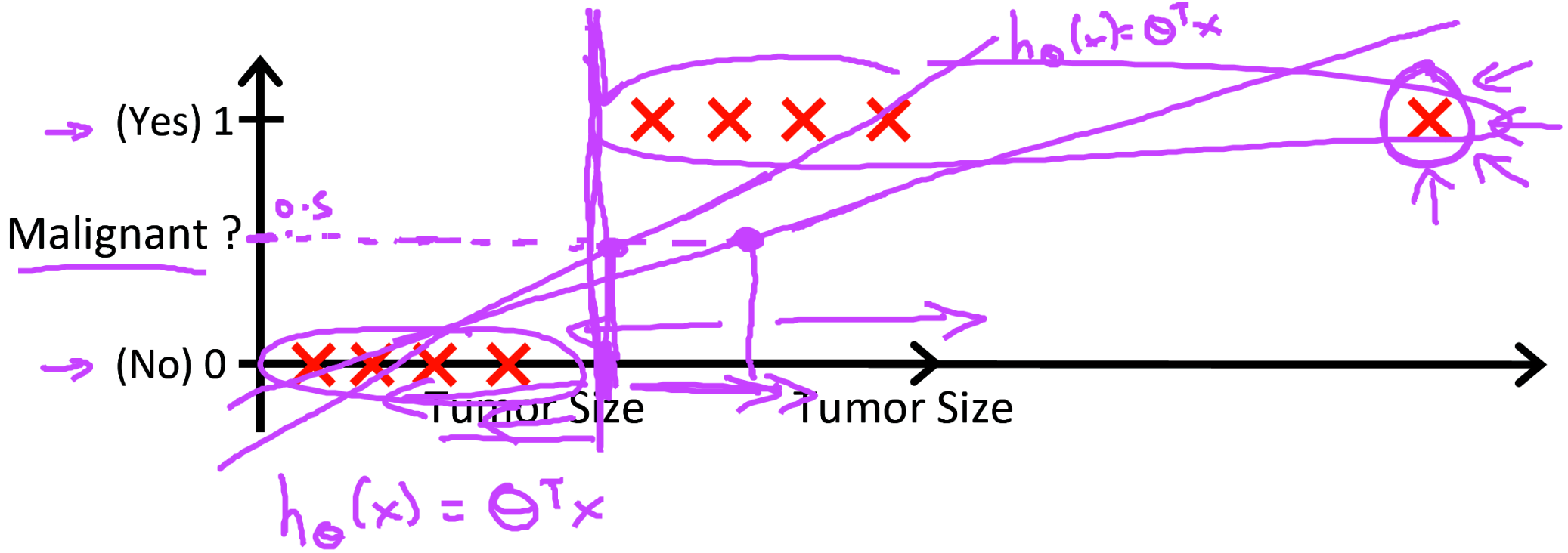
$y \in \{0, 1\}$

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

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

$y \in \{0, 1, 2, 3\}$

Classification



- Threshold classifier output $h_\theta(x)$ at 0.5:
  - If $h_\theta(x) \geq 0.5$, predict "y = 1"
  - If $h_\theta(x) < 0.5$, predict "y = 0"

Classification

Classification: $y = 0$ or 1

$h_\theta(x)$ can be > 1 or < 0

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Classification

Classification

Logistic Regression Model

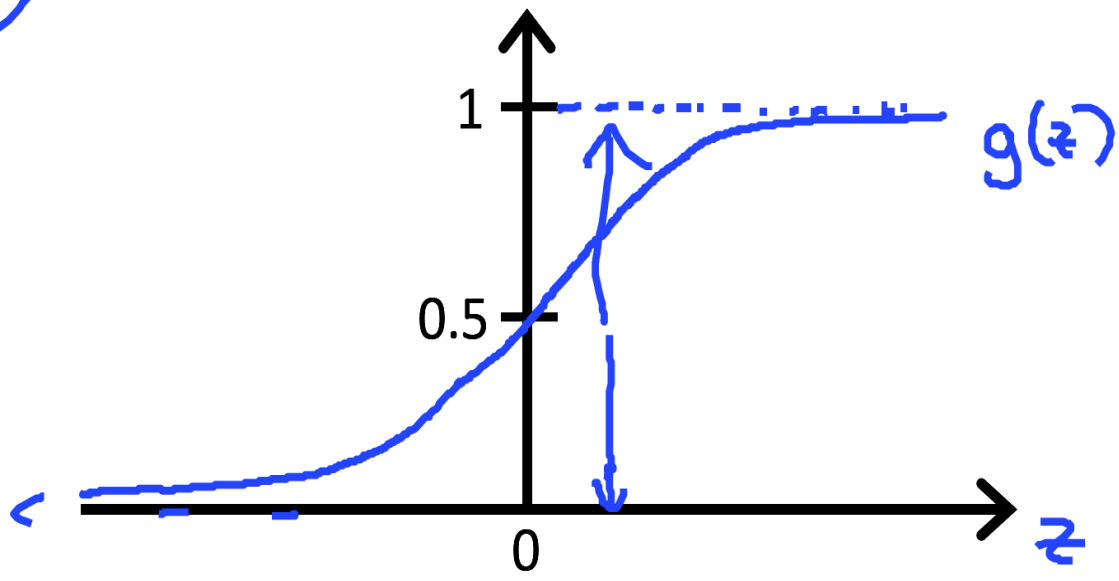
Want $0 \leq h_\theta(x) \leq 1$

$$h_\theta(x) = g(\theta^T x)$$

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

$z = \theta^T x$

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



- Sigmoid function
- Logistic function

Parameters θ.

Classification

Interpretation of Hypothesis Output

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$

Tell patient that 70% chance of tumor being malignant

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

"probability that y = 1, given x, parameterized by θ"

$y = 0$ or 1

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

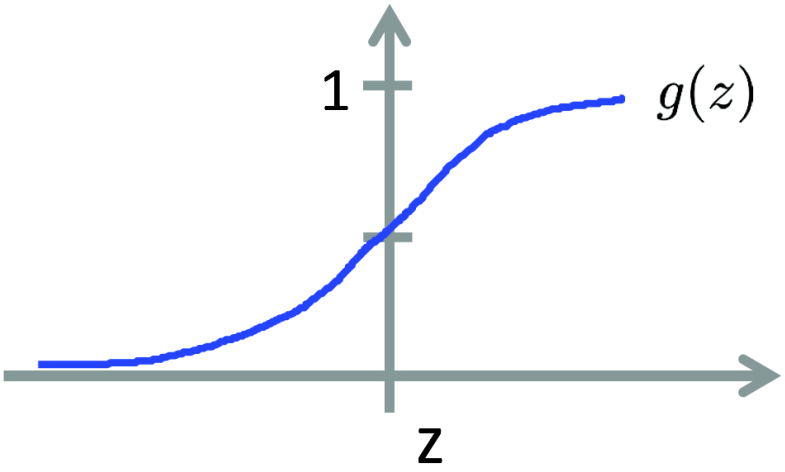
$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

Classification

Logistic regression

$$h_\theta(x) = g(\theta^T x)$$

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



Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$\theta^T x \geq 0$

$g(z) \geq 0.5$ when $z \geq 0$

$h_\theta(x) = g(\theta^T x)$

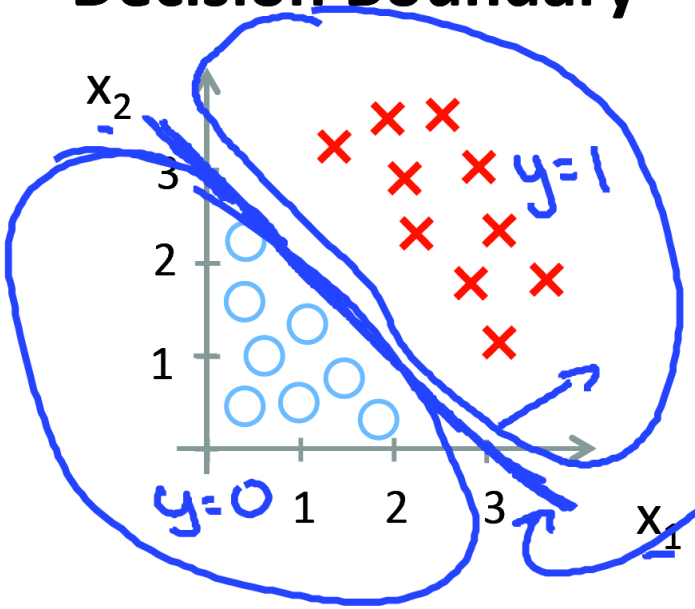
predict "$y = 0$" if $h_\theta(x) < 0.5$

$\theta^T x < 0$

$g(z) < 0.5$ when $z < 0$

Classification

Decision Boundary



$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

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

-3 $\quad$ 1 $\quad$ 1

Decision boundary

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

$x_1 + x_2 \geq 3$

x_1, x_2

$h_\theta(x) = 0.5$

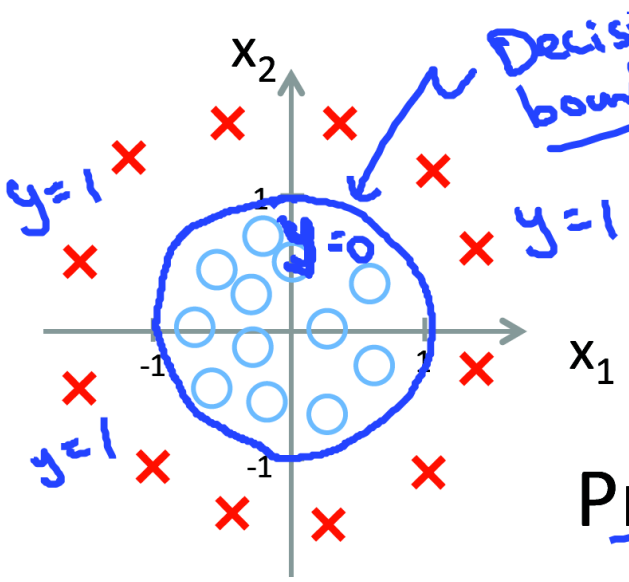
$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

$y = 0$

Classification

Non-linear decision boundaries



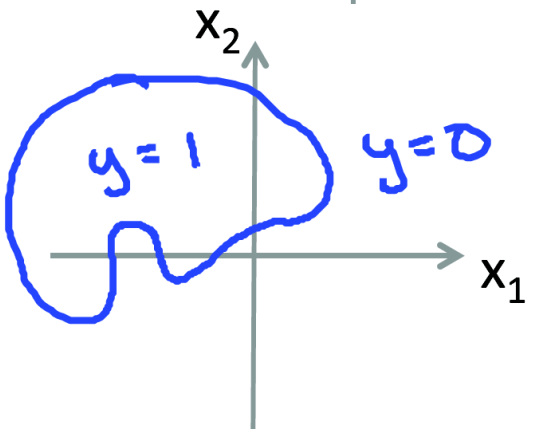
$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$$

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 \geq 1$



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

Classification

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix}$ $\mathbb{R}^{n+1}$ $x_0 = 1, y \in \{0, 1\}$

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

How to choose parameters θ ?

Classification:Cost function

Cost function

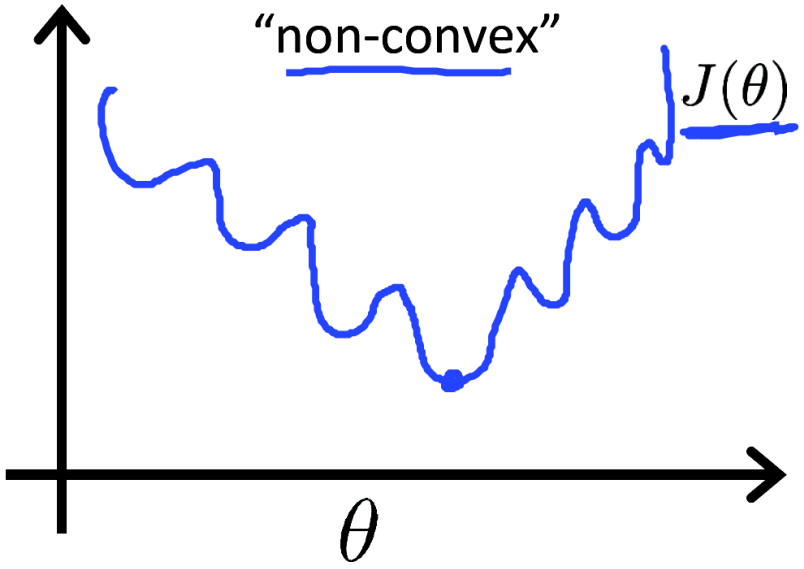
→ ~~Linear~~ regression: $J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

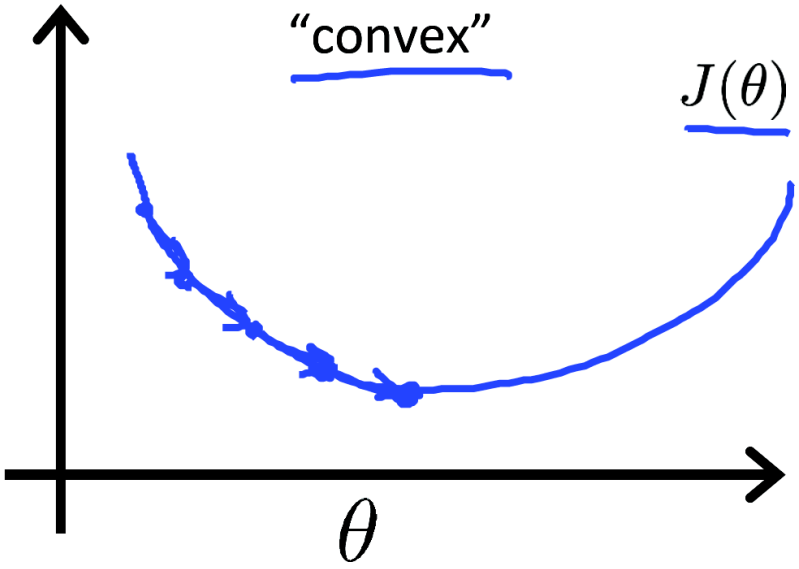
Logistic

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

$\text{Cost}(h_\theta(x^{(i)}), y)$

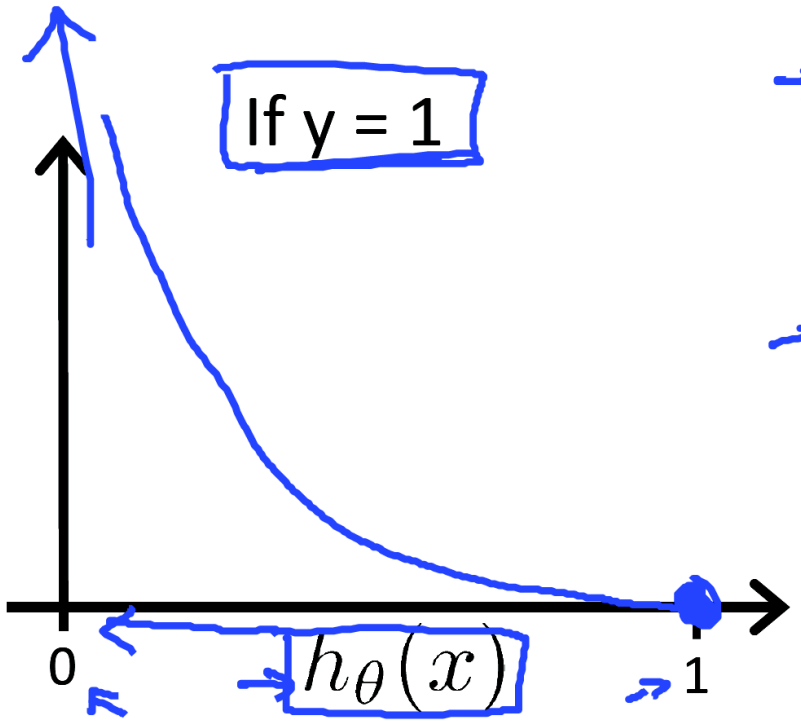
$\frac{1}{1 + e^{-\theta^T x}}$





Classification:Logistic Regression:Cost function

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

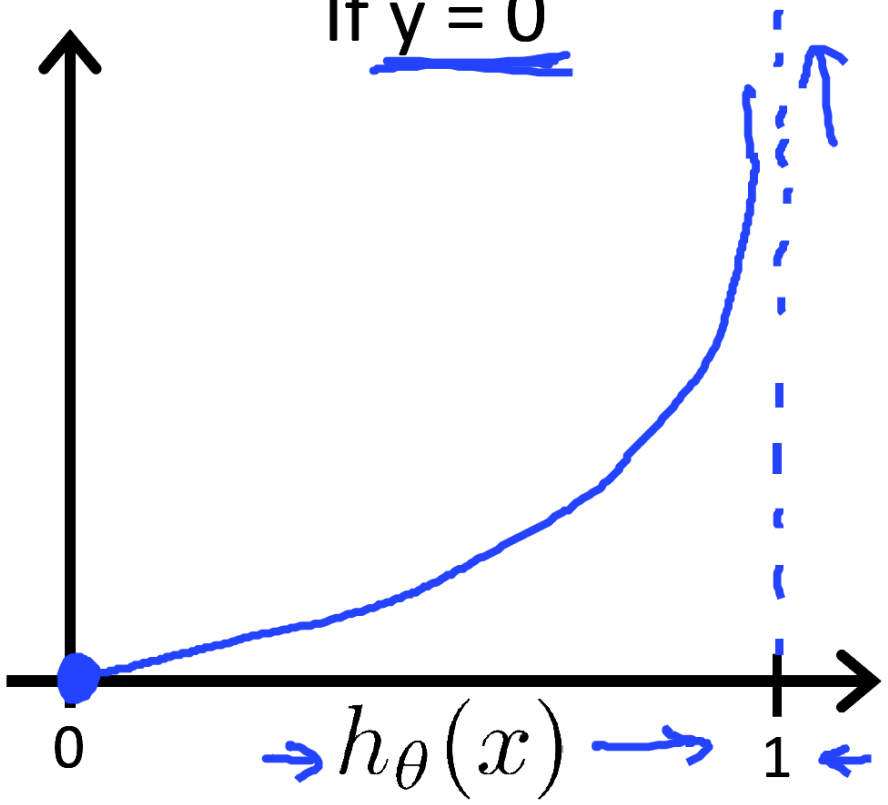


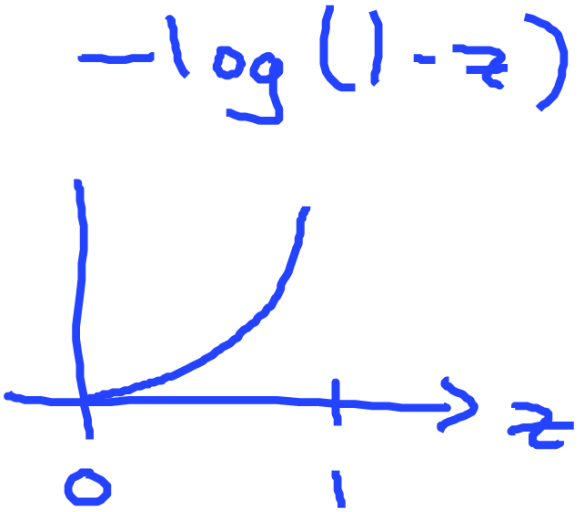
- $\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
  But as $h_\theta(x) \to 0$
  $Cost \to \infty$
- Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Classification:Logistic Regression:Cost function

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

If y = 0





Classification:Logistic Regression:Cost function

Logistic regression cost function

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

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

Note: $y = 0$ or 1 always

$$\text{Cost}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1-y) \log(1 - h_\theta(x))$$

If $y = 1$: $\text{Cost}(h_\theta(x), y) = -\log h_\theta(x)$

If $y = 0$: $\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$

Classification:Logistic Regression:Cost function

Logistic regression cost function

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

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

To fit parameters θ :

$$\min_\theta J(\theta)$$

Get θ

To make a prediction given new x:

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

$p(y=1 \mid x;\theta)$

Classification:Logistic Regression:Cost function

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

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

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

(simultaneously update all θ_j)

$\}$

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

Classification:Logistic Regression:Cost function

Gradient Descent

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

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

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$ for i = 0 to n

$h_\theta(x) = \theta^T x$

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

Algorithm looks identical to linear regression!

Classification:Logistic Regression:Cost function

$h_\theta x = g(\theta^T x)$

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

$Cost(h_\theta x, y) = -y \log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$

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

1. $d/dz(1/1+e^{-z})$ substitute $1+e^z = u$

$d/du(1/u) = d/du(u^{-1}) = -1u^{-2}$

2. $d/dz(1+e^{-z}) = -e^{-z}$

3. $d/dz(1/1+e^{-}) = dg/du \cdot du/dz$

$= -u^{-2} x - e^{-z}$

$= 1/u^2 x e^{-z}$

$= e^{-z}/(1+e^{-2})^2$

Derivatives cheat sheet

$d/dx\ 4x^3 = 3.4x^{3-1}$

$d/dx\ \sin(x) = \cos(x)$

$d/dx\ \cos(x) = -\sin(x)$

$d/dx\ \tan(x) = 1/\cos^2(x)$

$d/dx(e^x) = e^x$

$d/dx\ e^{-x} = -e^{-x}$

$d/dx\ \ln(x) = 1/x = x^{-1}$

$F(x) = f(g(x))$

$F'(x) = f'(g(x)) \cdot g'(x)$

Classification:Logistic Regression:Cost function

4. $e^{-z}/(1+e^{-z})^2 = \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} - \left(\frac{1}{1+e^{-z}}\right)^2$

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

$$= \boxed{g(z)(1-g(z))}$$

5. $\frac{\partial}{\partial\theta}(J\theta) = -1/m\left[\frac{\partial}{\partial\theta}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$

$$= -1/m\left[\sum_{i=1}^{m} y^{(i)} \frac{1}{h_\theta x^{(i)}} \frac{\partial}{\partial\theta} h_\theta(x^{(i)}) + 1-y^{(i)} \frac{1}{1-h_\theta x^{(i)}}\left(-\frac{\partial}{\partial\theta} h_\theta x^{(i)}\right)\right]$$

$$= -1/m\left[\sum_{i=1}^{m} y^{(i)} \frac{x_j^{(i)}}{h_\theta(x^{(i)})}(h_\theta x^{(i)})(1-h_\theta x^{(i)}) - (1-y^{(i)}) \frac{x_j^{(i)}}{1-h_\theta x^{(i)}}(h_\theta x^{(i)})(1-h_\theta x^{(i)})\right]$$

Classification:Logistic Regression:Cost function

4. $e^{-z}/(1+e^{-z})^2 = \frac{1+e^{-z}-1}{(1+e^{-z})^2} = \frac{1}{(1+e^{-z})} - \left(\frac{1}{1+e^{-z}}\right)^2$

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

$$= \boxed{g(z)(1-g(z))}$$

5. $\frac{\partial}{\partial\theta}(J\theta) = -1/m\left[\frac{\partial}{\partial\theta}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right]$

$$= -1/m\left[\sum_{i=1}^{m} y^{(i)}\frac{1}{h_\theta x^{(i)}}\frac{\partial}{\partial\theta} h_\theta(x^{(i)}) + 1-y^{(i)}\frac{1}{1-h_\theta x^{(i)}}\left(-\frac{\partial}{\partial\theta} h_\theta x^{(i)}\right)\right]$$

$$= -1/m\left[\sum_{i=1}^{m} y^{(i)}\frac{x_j^{(i)}}{h_\theta(x^{(i)})}(h_\theta x^{(i)})(1-h_\theta x^{(i)}) - (1-y^{(i)})\frac{x_j^{(i)}}{1-h_\theta x^{(i)}}(h_\theta x^{(i)})(1-h_\theta x^{(i)})\right]$$

Classification:Logistic Regression:Cost function

$$= -1/m \left[\sum y^{(i)} \cdot x_j^{(i)} \cdot (1 - h_\theta x^{(i)}) - (1-y) \cdot x_j^{(i)} \cdot h_\theta(x^{(i)}) \right]$$

$$= -1/m \left[\sum y^{(i)} x_j^{(i)} - y^{(i)} \cdot x_j^{(i)} h_\theta x - x_j^{(i)} \cdot h_\theta x^{(i)} + y^{(i)} \cdot x_j^{(i)} \cdot h_\theta x^{(i)} \right]$$

$$= -1/m \left[\sum y^{(i)} \cdot x_j^{(i)} - x_j^{(i)} \cdot h_\theta x^{(i)} \right]$$

$$= 1/m \sum_{i=1}^{m} (h_\theta x^{(i)} - y^{(i)}) x_j^{(i)}$$

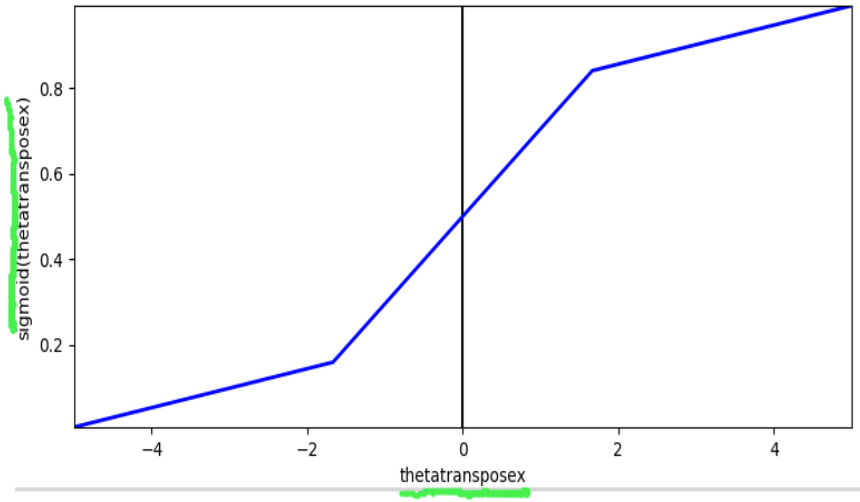
$$= \boxed{1/m \sum_{i=1}^{m} (\sigma(\theta^T x^{(i)}) - y^{(i)}) x_j^{(i)}}$$

Classification:Logistic Regression

```
def compute_cost(thetas,X,y):
    m = X.shape[0]
    print("training set size:\n",X.shape[0])
    costval=[]
    sigval=[]
    ttxvals=[]
    for theta in thetas:
        theta = np.reshape(theta,(len(theta),1))
        ttx=X.dot(theta)
        sig=sigmoid(X.dot(theta))
        sigval.append(sig)
        lg = -log(sig)
        oneminuslog=-log(1-sigmoid(X.dot(theta)))
        J=cost(X,y,theta)
        costval.append(J)
        if theta==1:
            print("X:\n",X)
            print("X.dot(theta)):\n",ttx)
            print("sig:",sig)
            plot(ttx,sig,"thetatransposex","sigmoid(thetatransposex)","")
            print("log:",lg)
            plot(sig,lg,"sigmoid(thetatransposex)","-log(sigmoid(thetatransposex))","")
            plot(ttx,lg,"thetatransposex","-log(sigmoid(thetatransposex))","")
            print("oneminuslog:",oneminuslog)
            plot(sig,oneminuslog,"sigmoid(thetatransposex)","-log(1-sigmoid(thetatransposex))","")
            plot(ttx,oneminuslog,"thetatransposex","-log(1-sigmoid(thetatransposex))","")
            print("Cost:",J)
    plot(thetas,costval,"theta","cost","")
```

```
def sigmoid(X):
    return 1 / (1 + np.exp(-X))
```

- also called the squashing function.




```
X: [  0. 3.33333333 6.66666667 10. ]
y: [ False False True True]
X:
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
X.dot(theta)):
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
sig: [[ 0.00669285]
      [ 0.1588691 ]
      [ 0.8411309 ]
      [ 0.99330715]]
log: [[ 5.00671535]
      [ 1.83967466]
      [ 0.17300799]
      [ 0.00671535]]
oneminuslog: [[ 0.00671535]
              [ 0.17300799]
              [ 1.83967466]
              [ 5.00671535]]
Cost: [ 0.08986167]
```

n input labels

scaled input

$\theta^T x$ for $\theta \to 1$

Sigmoid or σ

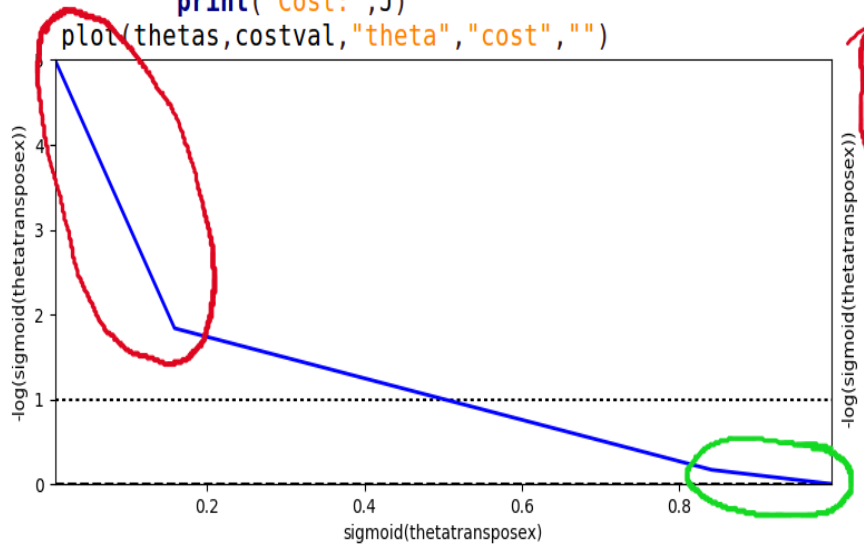
Classification:Logistic Regression

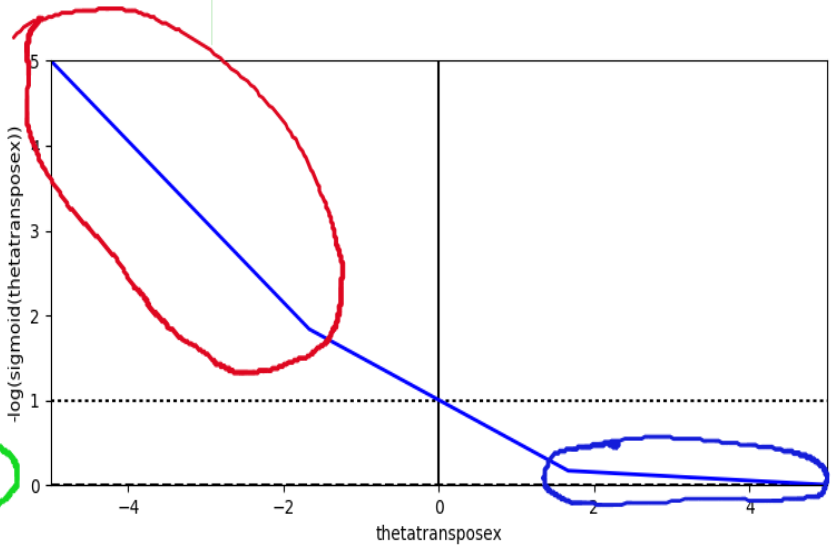
```
def compute_cost(thetas,X,y):
    m = X.shape[0]
    print("training set size:\n",X.shape[0])
    costval=[]
    sigval=[]
    ttxvals=[]
    for theta in thetas:
        theta = np.reshape(theta,(len(theta),1))
        ttx=X.dot(theta)
        sig=sigmoid(X.dot(theta))
        sigval.append(sig)
        lg = -log(sig)
        oneminuslog=-log(1-sigmoid(X.dot(theta)))
        J=cost(X,y,theta)
        costval.append(J)
        if theta==1:
            print("X:\n",X)
            print("X.dot(theta)):\n",ttx)
            print("sig:",sig)
            plot(ttx,sig,"thetatransposex","sigmoid(thetatransposex)","")
            print("log:",lg)
            plot(sig,lg,"sigmoid(thetatransposex)","-log(sigmoid(thetatransposex))","")
            plot(ttx,lg,"thetatransposex","-log(sigmoid(thetatransposex))","")
            print("oneminuslog:",oneminuslog)
            plot(sig,oneminuslog,"sigmoid(thetatransposex)","-log(1-sigmoid(thetatransposex))","")
            plot(ttx,oneminuslog,"thetatransposex","-log(1-sigmoid(thetatransposex))","")
            print("Cost:",J)
    plot(thetas,costval,"theta","cost","")
```

$$Cost(h_\theta x, y) = -y \log(h_\theta x) - (1-y)\log(1-h_\theta x)$$

- when $(Y=1)$, and if $h_\theta x(\sigma)$ is also 1 then the cost tends to 0, because the algo gets it right.
- As $\theta^T x \gg 0$, $h_\theta x \to 1$ and $-\log(h_\theta x) \to 0$
- when $(y=1)$ but if $h_\theta x \to 0$, the cost goes up or we penalize the algo

```
X: [  0. 3.33333333 6.66666667 10. ]
y: [ False False True True]
X:
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
X.dot(theta)):
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
sig: [[ 0.00669285]
      [ 0.1588691 ]
      [ 0.8411309 ]
      [ 0.99330715]]
log: [[ 5.00671535]
      [ 1.83967466]
      [ 0.17300799]
      [ 0.00671535]]
oneminuslog: [[ 0.00671535]
              [ 0.17300799]
              [ 1.83967466]
              [ 5.00671535]]
Cost: [ 0.08986167]
```





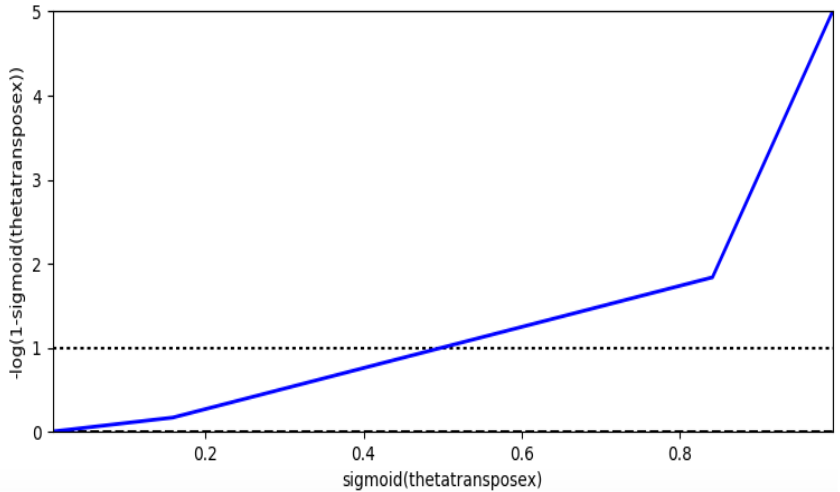
Classification:Logistic Regression

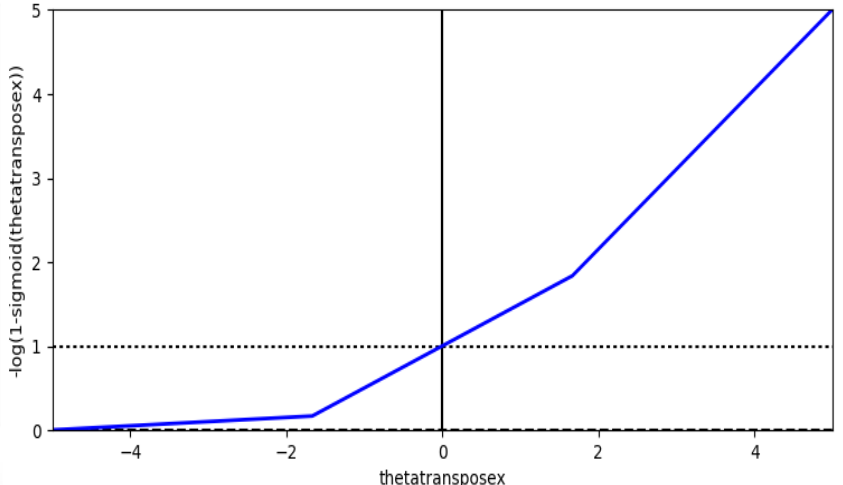
```
def compute_cost(thetas,X,y):
    m = X.shape[0]
    print("training set size:\n",X.shape[0])
    costval=[]
    sigval=[]
    ttxvals=[]
    for theta in thetas:
        theta = np.reshape(theta,(len(theta),1))
        ttx=X.dot(theta)
        sig=sigmoid(X.dot(theta))
        sigval.append(sig)
        lg = -log(sig)
        oneminuslog=-log(1-sigmoid(X.dot(theta)))
        J=cost(X,y,theta)
        costval.append(J)
        if theta==1:
            print("X:\n",X)
            print("X.dot(theta)):\n",ttx)
            print("sig:",sig)
            plot(ttx,sig,"thetatransposex","sigmoid(thetatransposex)","")
            print("log:",lg)
            plot(sig,lg,"sigmoid(thetatransposex)","-log(sigmoid(thetatransposex))","")
            plot(ttx,lg,"thetatransposex","-log(sigmoid(thetatransposex))","")
            print("oneminuslog:",oneminuslog)
            plot(sig,oneminuslog,"sigmoid(thetatransposex)","-log(1-sigmoid(thetatransposex))","")
            plot(ttx,oneminuslog,"thetatransposex","-log(1-sigmoid(thetatransposex))","")
            print("Cost:",J)
    plot(thetas,costval,"theta","cost","")
```

$$Cost(h_\theta x, y) = -y \log(h_\theta x) - (1-y)\log(1-h_\theta x)$$

- when $(y=0)$...

```
X: [  0. 3.33333333 6.66666667 10
y: [ False False True True]
X:
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
X.dot(theta)):
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
sig: [[ 0.00669285]
      [ 0.1588691 ]
      [ 0.8411309 ]
      [ 0.99330715]]
log: [[ 5.00671535]
      [ 1.83967466]
      [ 0.17300799]
      [ 0.00671535]]
oneminuslog: [[ 0.00671535]
              [ 0.17300799]
              [ 1.83967466]
              [ 5.00671535]]
Cost: [ 0.08986167]
```



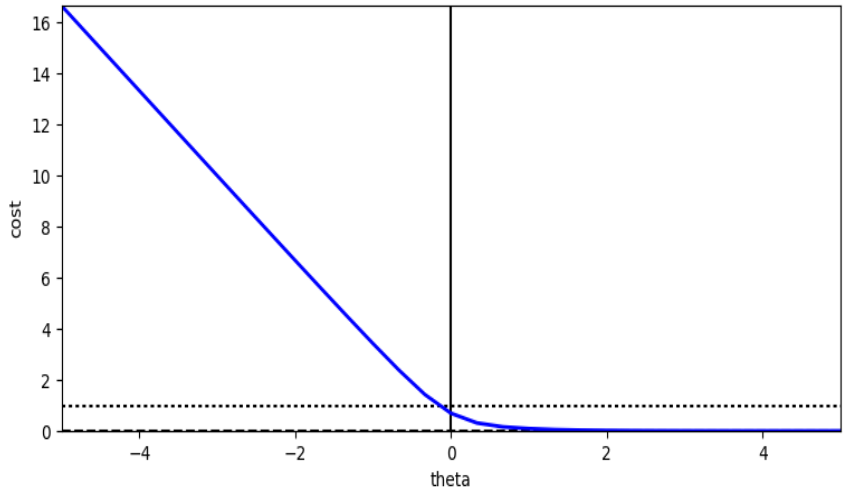


Classification:Logistic Regression

```
def compute_cost(thetas,X,y):
    m = X.shape[0]
    print("training set size:\n",X.shape[0])
    costval=[]
    sigval=[]
    ttxvals=[]
    for theta in thetas:
        theta = np.reshape(theta,(len(theta),1))
        ttx=X.dot(theta)
        sig=sigmoid(X.dot(theta))
        sigval.append(sig)
        lg = -log(sig)
        oneminuslog=-log(1-sigmoid(X.dot(theta)))
        J=cost(X,y,theta)
        costval.append(J)
        if theta==1:
            print("X:\n",X)
            print("X.dot(theta)):\n",ttx)
            print("sig:",sig)
            plot(ttx,sig,"thetatransposex","sigmoid(thetatransposex)","")
            print("log:",lg)
            plot(sig,lg,"sigmoid(thetatransposex)","-log(sigmoid(thetatransposex))","")
            plot(ttx,lg,"thetatransposex","-log(sigmoid(thetatransposex))","")
            print("oneminuslog:",oneminuslog)
            plot(sig,oneminuslog,"sigmoid(thetatransposex)","-log(1-sigmoid(thetatransposex))","")
            plot(ttx,oneminuslog,"thetatransposex","-log(1-sigmoid(thetatransposex))","")
            print("Cost:",J)
    plot(thetas,costval,"theta","cost","")
```

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

```
def cost(X,y,theta):
    m = X.shape[0]
    return (1./m) * (-transpose(y).dot(log(sigmoid(X.dot(theta))))
        - transpose(1-y).dot(log(1-sigmoid(X.dot(theta)))))
```




```
X: [  0. 3.33333333 6.66666667 10. ]
y: [ False False True True]
X:
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
X.dot(theta)):
 [[-5.        ]
  [-1.66666667]
  [ 1.66666667]
  [ 5.        ]]
sig: [[ 0.00669285]
      [ 0.1588691 ]
      [ 0.8411309 ]
      [ 0.99330715]]
log: [[ 5.00671535]
      [ 1.83967466]
      [ 0.17300799]
      [ 0.00671535]]
oneminuslog: [[ 0.00671535]
              [ 0.17300799]
              [ 1.83967466]
              [ 5.00671535]]
Cost: [ 0.08986167]
```

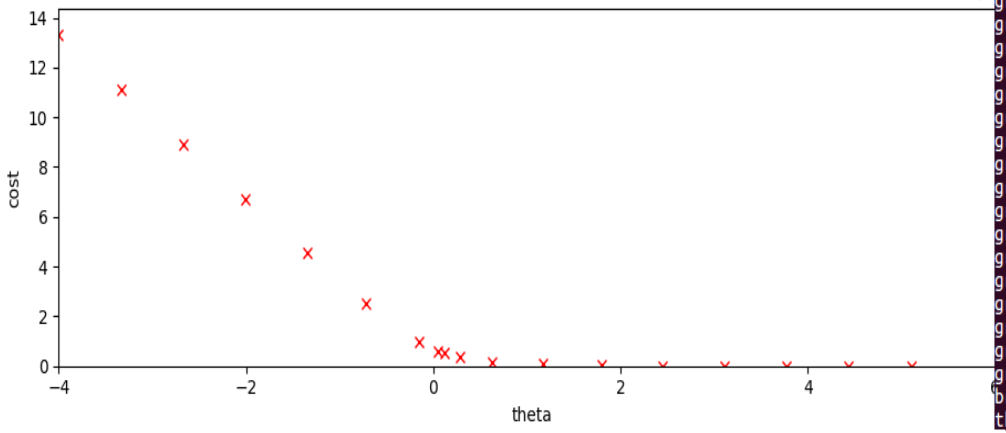
$\{ \quad \} / m$

Classification:Logistic Regression:BGD

```
def gradient(X,y,theta):
    m = len(X)
    return 1/m * X.T.dot(np.sum(sigmoid(X.dot(theta))-y,axis=1).reshape(-1,1))

def plot_gradient_descent(theta, eta, theta_path=None):
    m = len(tumor_size_scaled)
    n_iterations = 20
    costvalprev=0.0
    change=0.0
    for iteration in range(n_iterations):
        y_predict=sigmoid(thetaTransposeX(tumor_size_scaled,theta))
        costval=cost(tumor_size_scaled,mal,theta)
        if iteration < iterations_to_track:
            if theta_path_bgd_10 is not None:
                theta_path_bgd_10.append(theta)
            if cost_path_bgd_10 is not None:
                cost_path_bgd_10.append(costval)
            if y_predict_bgd_10 is not None:
                y_predict_bgd_10.append(y_predict)
        gradients = gradient(tumor_size_scaled,mal,theta)
        if gradients > 0: #going up so reverse direction
            theta = theta + eta * gradients
        else:
            theta = theta - eta * gradients
        if iteration !=0:
            if (costvalprev-costval< .0004):
                print("breaking:")
                break
        costvalprev=costval
        print("gradient:",gradients," theta:",theta," costval:",costval)
```

gradient $\Rightarrow \partial/\partial\theta_j J(\theta) = \frac{1}{m}\sum_{i=1}^{m}(\sigma(\theta^T x^i) - y^i)x^i_j$

- convergence rule
- going up or down the slope.

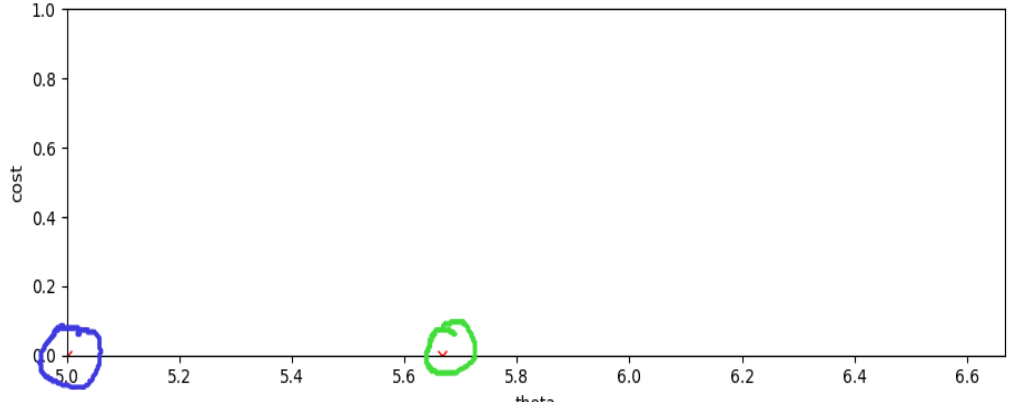



```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/logisticregression/logsiticregressiongraph2.py
thetainit: [[-4.]]
gradient: [[-6.66242993]]  theta: [[-3.33375701]]  costval: [ 13.33396924]
gradient: [[-6.65383834]]  theta: [[-2.66837317]]  costval: [ 11.11445126]
gradient: [[-6.62806817]]  theta: [[-2.00556636]]  costval: [ 8.90039917]
gradient: [[-6.5524319]]  theta: [[-1.35032317]]  costval: [ 6.70261041]
gradient: [[-6.33731371]]  theta: [[-0.7165918]]  costval: [ 4.55173898]
gradient: [[-5.62129158]]  theta: [[-0.15446264]]  costval: [ 2.53464648]
gradient: [[-2.05356512]]  theta: [[ 0.05089387]]  costval: [ 0.99110364]
gradient: [[ 0.70340617]]  theta: [[ 0.12123449]]  costval: [ 0.6128099]
gradient: [[ 1.63848263]]  theta: [[ 0.28508275]]  costval: [ 0.51625916]
gradient: [[ 3.45052092]]  theta: [[ 0.63013485]]  costval: [ 0.34948867]
gradient: [[ 5.39207964]]  theta: [[ 1.16934281]]  costval: [ 0.17096596]
gradient: [[ 6.22228]]  theta: [[ 1.79157081]]  costval: [ 0.06802139]
gradient: [[ 6.50516429]]  theta: [[ 2.44208724]]  costval: [ 0.02469341]
gradient: [[ 6.61065606]]  theta: [[ 3.10315284]]  costval: [ 0.00846787]
gradient: [[ 6.64785935]]  theta: [[ 3.76793878]]  costval: [ 0.0028289]
gradient: [[ 6.66043297]]  theta: [[ 4.43398208]]  costval: [ 0.00093592]
gradient: [[ 6.66460989]]  theta: [[ 5.10044307]]  costval: [ 0.00030861]
breaking:
theta best: [[ 5.76704197]]
```

Classification:Logistic Regression:BGD

```
def gradient(X,y,theta):
    m = len(X)
    return 1/m * X.T.dot(np.sum(sigmoid(X.dot(theta))-y,axis=1).reshape(-1,1))

def plot_gradient_descent(theta, eta, theta_path=None):
    m = len(tumor_size_scaled)
    n_iterations = 20
    costvalprev=0.0
    change=0.0
    for iteration in range(n_iterations):
        y_predict=sigmoid(thetaTransposeX(tumor_size_scaled,theta))
        costval=cost(tumor_size_scaled,mal,theta)
        if iteration < iterations_to_track:
            if theta_path_bgd_10 is not None:
                theta_path_bgd_10.append(theta)
            if cost_path_bgd_10 is not None:
                cost_path_bgd_10.append(costval)
            if y_predict_bgd_10 is not None:
                y_predict_bgd_10.append(y_predict)
        gradients = gradient(tumor_size_scaled,mal,theta)
        if iteration !=0:
            if (costvalprev-costval < .0004):
                print("breaking:")
                break
        if gradients > 0: #going up so reverse direction
            theta = theta + eta * gradients
        else:
            theta = theta - eta * gradients
        costvalprev=costval
        print("gradient:",gradients," theta:",theta," costval:",costval)
```

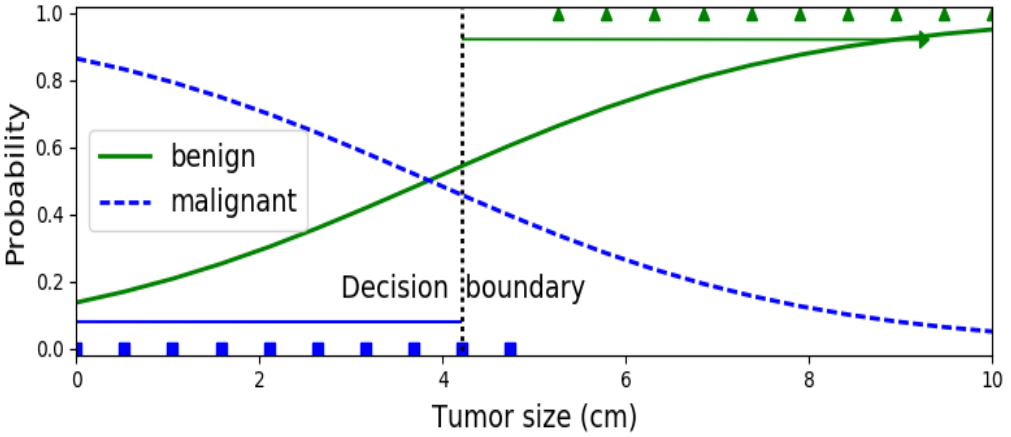
- theta init = 5
- theta best = 5.66



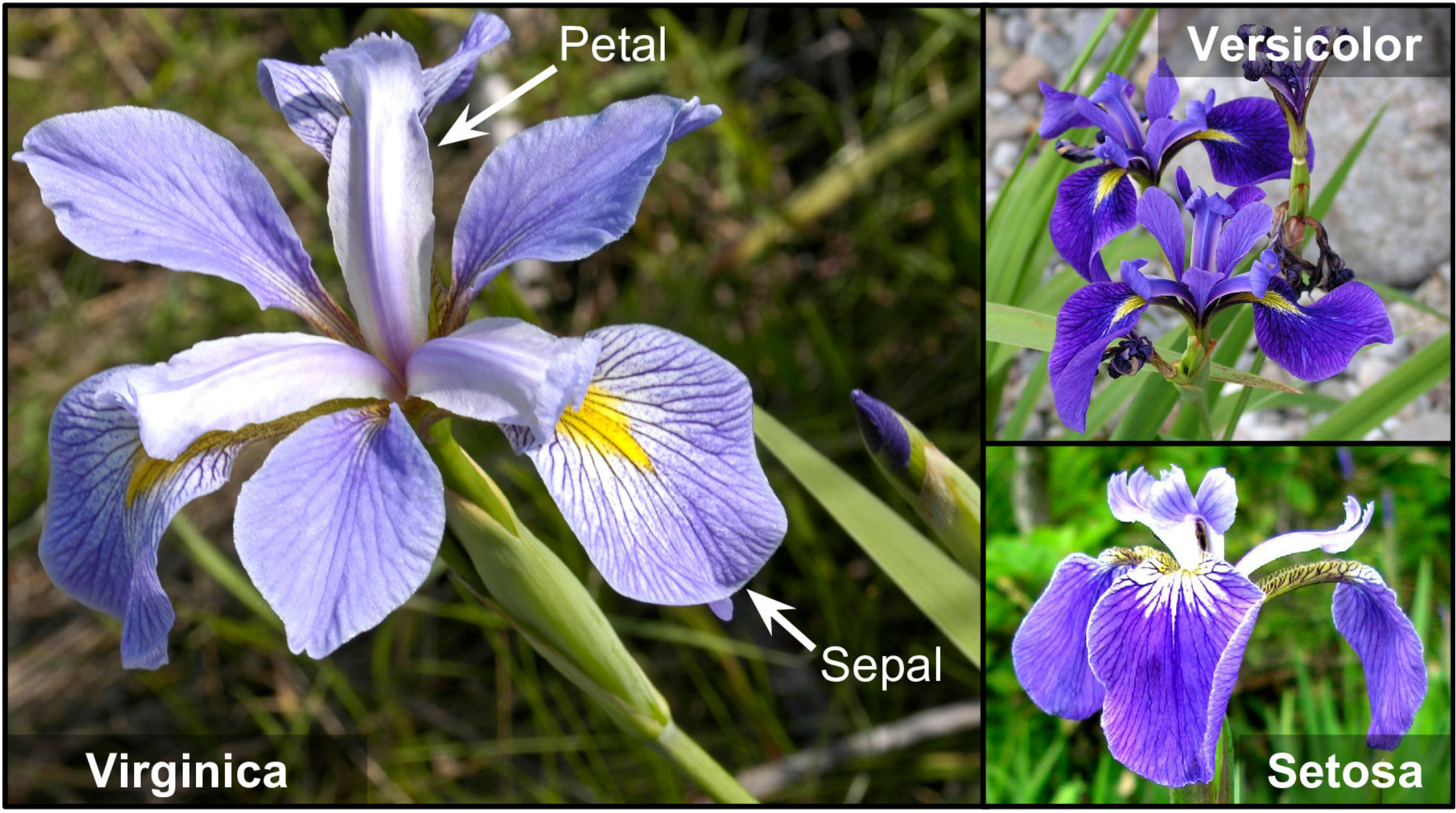

```
(ml_home) mohit@nomind:~/Work/ArtificialIntelligence$ ./ML/logisticregression/logsiticregressiongraph2.py
thetainit: [[ 5.]]
gradient: [[ 6.66586563]]  theta: [[ 5.66658656]]  costval: [ 0.00012017]
breaking:
theta best: [[ 5.66658656]]
```

Classification:Logistic Regression(scikit):descision boundary

```
tumor_size=np.linspace(0.0, 10.0, num=20)
print("\nX:",tumor_size)
tumor_size_new=tumor_size.reshape(-1,1)
print("\nx_new:",tumor_size_new)

mal=(tumor_size>5)
print("\nmal:",mal)


log_reg = LogisticRegression()
log_reg.fit(tumor_size_new, mal)
print(log_reg)
print(log_reg.coef_)
print(log_reg.intercept_)
X_test = np.linspace(0.0, 10.0, 20).reshape(-1, 1)
mal_proba = log_reg.predict_proba(X_test)
decision_boundary = X_test[mal_proba[:, 1] >= 0.5][0]
print("decision_boundary:",decision_boundary)
```

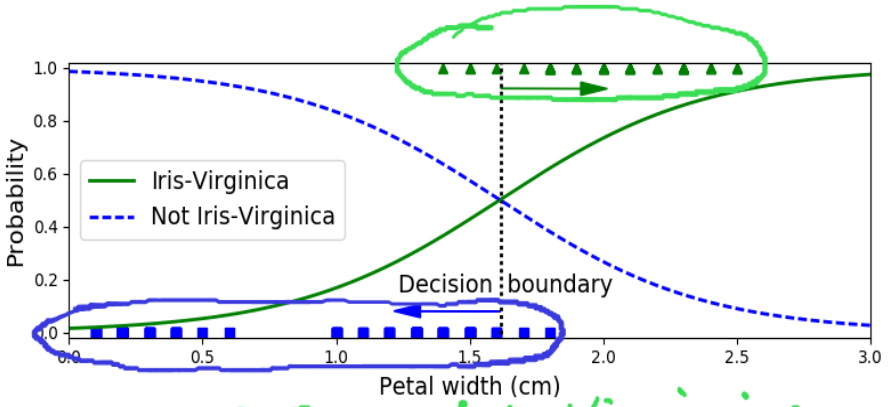


Classification:Logistic Regression(scikit):descision boundary



Classification:Logistic Regression(scikit):descision boundary

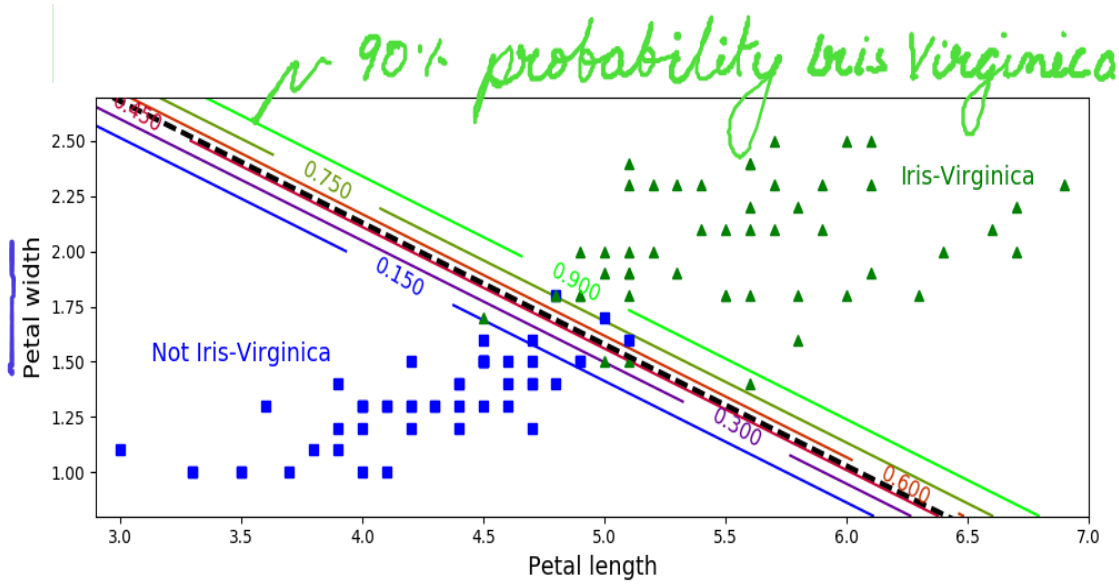
```
iris = datasets.load_iris()
print(list(iris.keys()))
print(iris.DESCR)


X = iris["data"][:, 3:]  # petal width
y = (iris["target"] == 2).astype(np.int)  # 1 if Iris-Virginica, else 0

log_reg = LogisticRegression()
log_reg.fit(X, y)
X1=np.linspace(0, 3, 1000);

X_new = np.linspace(0, 3, 1000).reshape(-1, 1)

y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]
print("decision_boundary:",decision_boundary)
```



- 1.4-2.5 cm Iris Virginica
- 0.1-1.8 cm other specie
- Descision Boundary at 1.6. and the algorithm is not so sure around that mark.
- may be adding more features will help.

Classification:Logistic Regression(scikit):linear descision boundary

```
iris = datasets.load_iris()

X = iris["data"][:, (2, 3)]  # petal length, petal width
y = (iris["target"] == 2).astype(np.int)

log_reg = LogisticRegression(C=10**10)
log_reg.fit(X, y)

x0, x1 = np.meshgrid(
        np.linspace(2.9, 7, 500).reshape(-1, 1),
        np.linspace(0.8, 2.7, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
y_proba = log_reg.predict_proba(X_new)
plt.figure(figsize=(10, 4))
plt.plot(X[y==0, 0], X[y==0, 1], "bs")
plt.plot(X[y==1, 0], X[y==1, 1], "g^")
zz = y_proba[:, 1].reshape(x0.shape)
contour = plt.contour(x0, x1, zz, cmap=plt.cm.brg)

left_right = np.array([2.9, 7])
boundary = -(log_reg.coef_[0][0] * left_right + log_reg.intercept_[0]) / log_reg.coef_[0][1]

plt.clabel(contour, inline=1, fontsize=12)
plt.plot(left_right, boundary, "k--", linewidth=3)
plt.text(3.5, 1.5, "Not Iris-Virginica", fontsize=14, color="b", ha="center")
plt.text(6.5, 2.3, "Iris-Virginica", fontsize=14, color="g", ha="center")
plt.xlabel("Petal length", fontsize=14)
plt.ylabel("Petal width", fontsize=14)
plt.axis([2.9, 7, 0.8, 2.7])
```

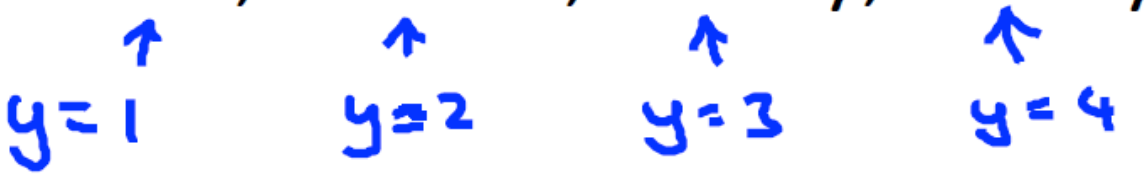


- $C = 1/\lambda$ controls regularization
- LR adds l_2 penalty by default.

- 2 features on a contour plot.
- dashed line is the DB.
- each parallel line represents a specific probability. From (15% – 90%)

Multiclass classification

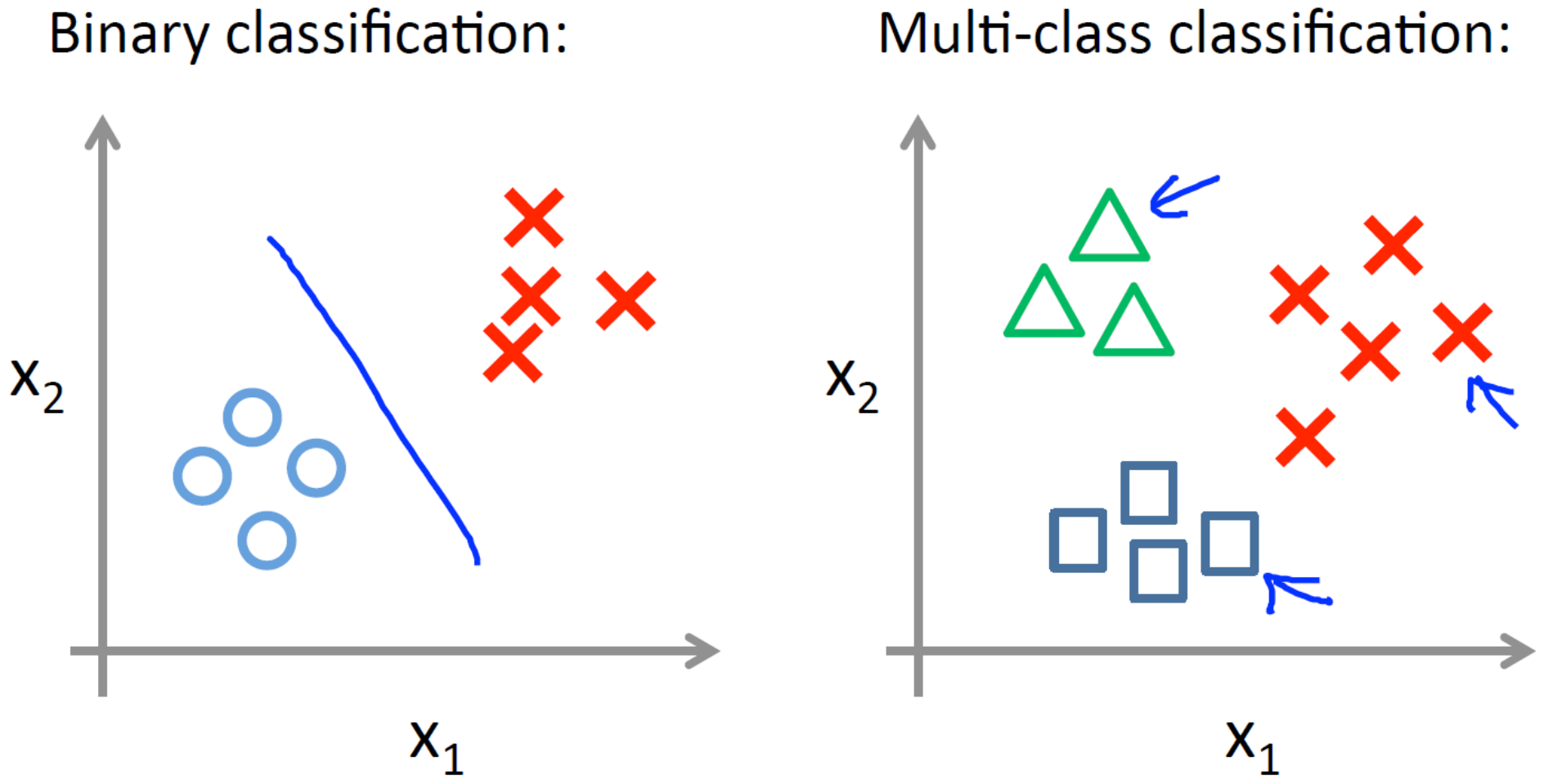
Email foldering/tagging: Work, Friends, Family, Hobby



Medical diagrams: Not ill, Cold, Flu

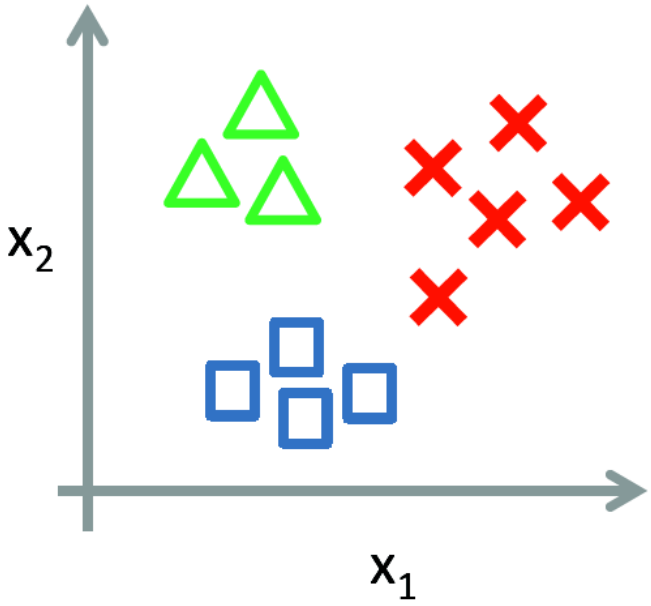
Weather: Sunny, Cloudy, Rain, Snow

Multiclass Classification



Multiclass Classification

One-vs-all (one-vs-rest):

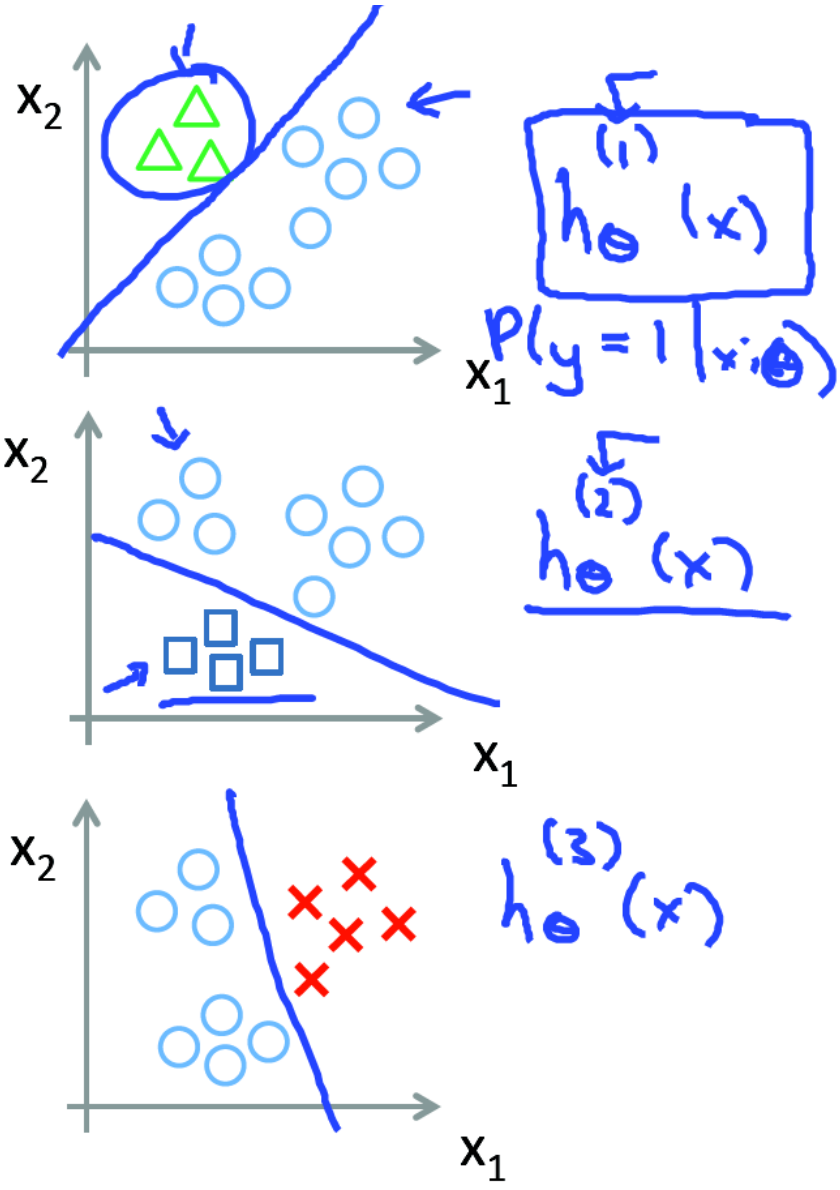


Class 1: △

Class 2: □

Class 3: ×

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$



Multiclass Classification:Softmax Regression

```
fl=fileloader("input")
iris = datasets.load_iris()
X = iris["data"][:, (2, 3)]  # petal length, petal width
y = iris["target"]

softmax_reg = LogisticRegression(multi_class="multinomial", solver="lbfgs", C=10)
softmax_reg.fit(X, y)
x0, x1 = np.meshgrid(
        np.linspace(0, 8, 500).reshape(-1, 1),
        np.linspace(0, 3.5, 200).reshape(-1, 1),
    )
X_new = np.c_[x0.ravel(), x1.ravel()]
y_proba = softmax_reg.predict_proba(X_new)
y_predict = softmax_reg.predict(X_new)

zz1 = y_proba[:, 1].reshape(x0.shape)
zz = y_predict.reshape(x0.shape)

plt.figure(figsize=(10, 4))
plt.plot(X[y==2, 0], X[y==2, 1], "g^", label="Iris-Virginica")
plt.plot(X[y==1, 0], X[y==1, 1], "bs", label="Iris-Versicolor")
plt.plot(X[y==0, 0], X[y==0, 1], "yo", label="Iris-Setosa")

from matplotlib.colors import ListedColormap
custom_cmap = ListedColormap(['#fafab0','#9898ff','#a0faa0'])
```