



ieeta instituto de engenharia electrónica e telemática de aveiro



universidade  
de aveiro

Departamento de Eletrónica, Telecomunicações e  
Informática

# **Machine Learning**

## **LECTURE 3: CLASSIFICATION - LOGISTIC REGRESSION**

**Petia Georgieva**  
**(petia@ua.pt)**

# **LOGISTIC REGRESSION - outline**

**1. Sigmoid function model (hypothesis)**

**2. Logistic regression cost function**

**3. Decision boundary**

**4. Nonlinearly separable data**

**3. Regularized logistic regression**

# CLASSIFICATION

Email: Spam /Not Spam ?

Tumour: Malignant /Benign ?

Online Transactions: Fraudulent (Yes /No) ?

## Binary classification:

$y = 1$ : “positive class” (e.g. malignant tumour)

$y = 0$ : “negative class” (e.g. benign tumour)

Threshold classifier output (probabilistic interpretation):

if  $h(x) \geq 0.5$ , predict “ $y=1$ ”

if  $h(x) < 0.5$ , predict “ $y=0$ ”

$0 \leq h(x) \leq 1$

**Multiclass classification** ( $m$  classes)  $\Rightarrow y = \{0, 1, 2, \dots\}$

Build  $m$  binary classifiers, for each classifier one of the classes has label 1 all other classes take label 0 .

# Logistic Regression

Given labelled data of  $m$  examples,  $n$  features

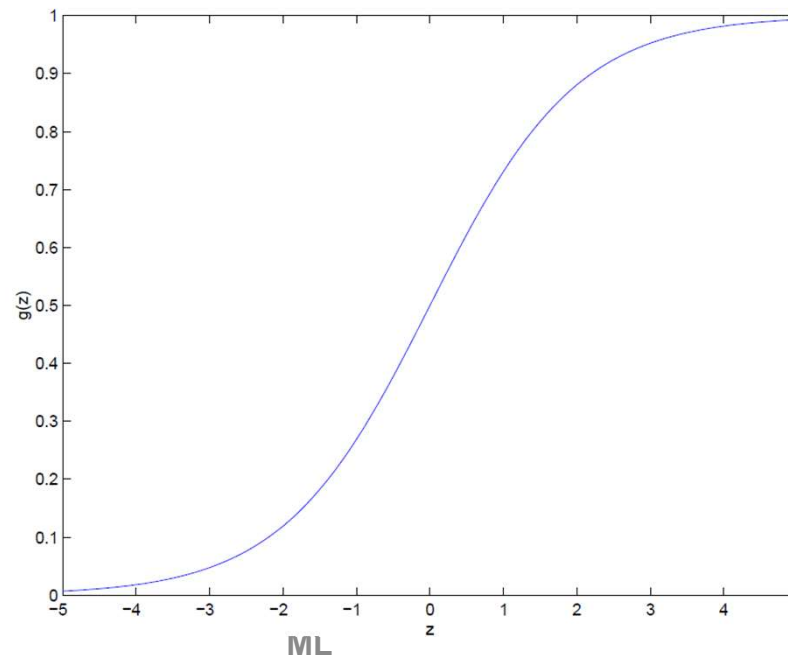
the labels are  $\{0, 1\} \Rightarrow$  binary classification

$x$  – vector of features;  $\theta$  – vector of model parameters;

$h(x)$  – logistic (sigmoid) function model (hypothesis)

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

**Logistic (sigmoid) function**



# Logistic Regression Cost Function

**Linear regression model** =>

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

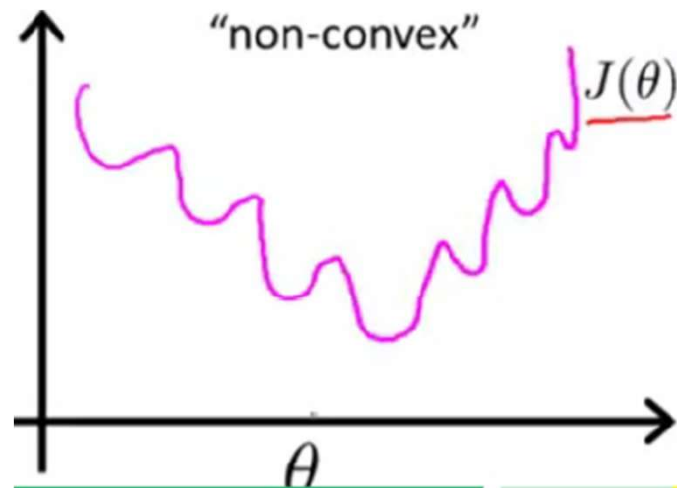
**Linear Regression cost function** =>

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

**Nonlinear logistic (sigmoid) model** =>

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

If we use the same cost function as with linear regression, but now the hypothesis is a nonlinear function,  $J(\theta)$  will be a non-convex function (has many local minima)=> **not efficient for optimization !**



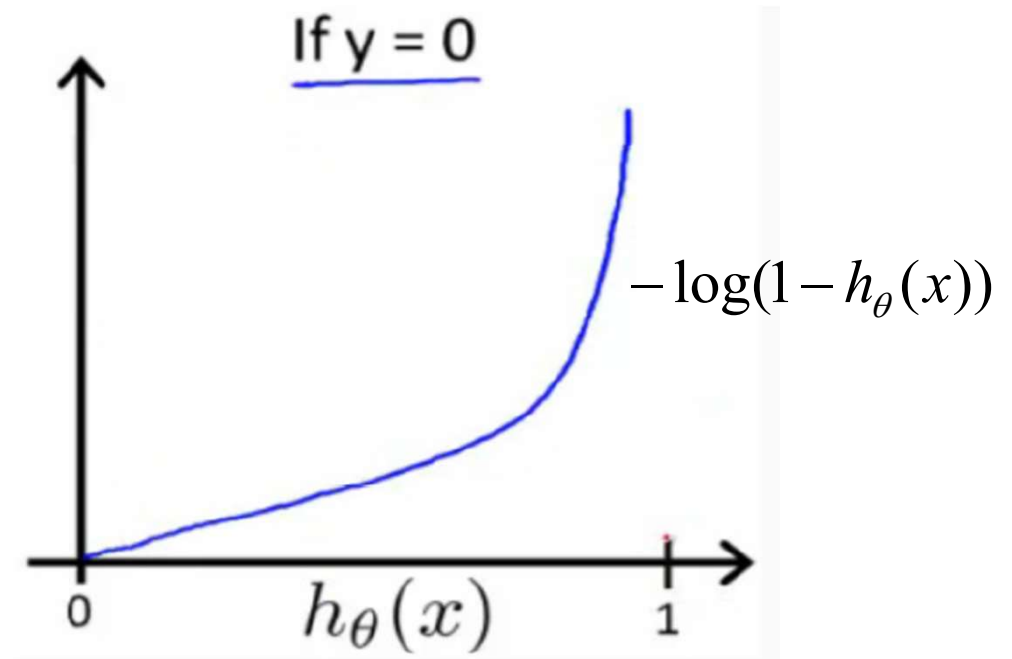
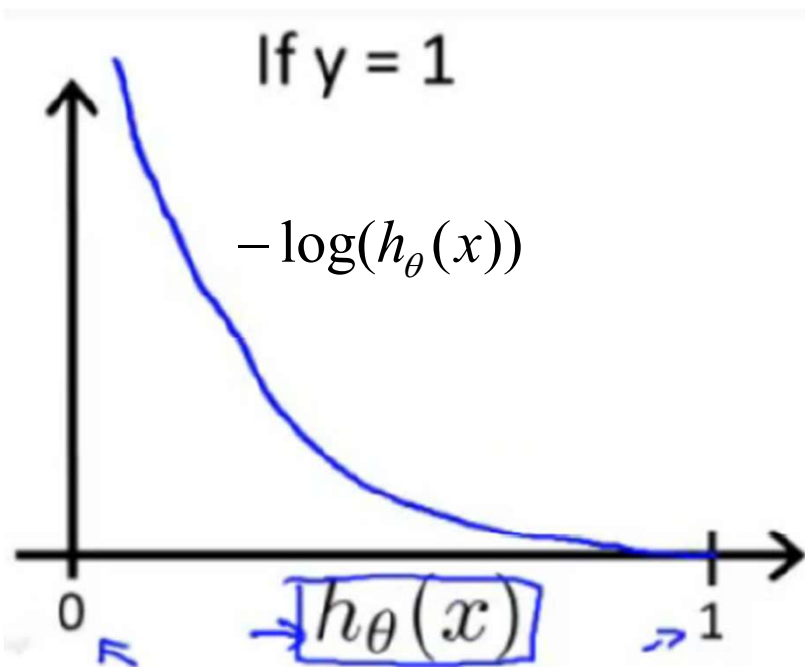
ML

# 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



# 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

**LogReg cost function combined into one expression :**  
***(also known as Cross-Entropy or Log Loss function)***

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

# Log Reg with gradient descent learning

**Initialize model parameters** (e.g.  $\theta = 0$ )

**Repeat until J converge** {

**Compute LogReg Model prediction** =>  
(different from linear regression model)

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

**Compute LogReg cost function** =>  
(different from linear regression cost function)

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

**Goal** =>

$$\min_{\theta} J(\theta)$$

**Compute cost function gradients** =>  
(same as linear regression gradients)

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

**Update parameters** =>  
(same as linear regression parameter update)

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

}



# Optimization algorithms

## 1. Gradient descent (learned in class)

## 2. Other optimization algorithms

- Conjugate gradient
- BFGS (Broyden–Fletcher–Goldfarb–Shanno)
- LM-BFGS Limited-memory BFGS (from the family of quasi-Newton methods)

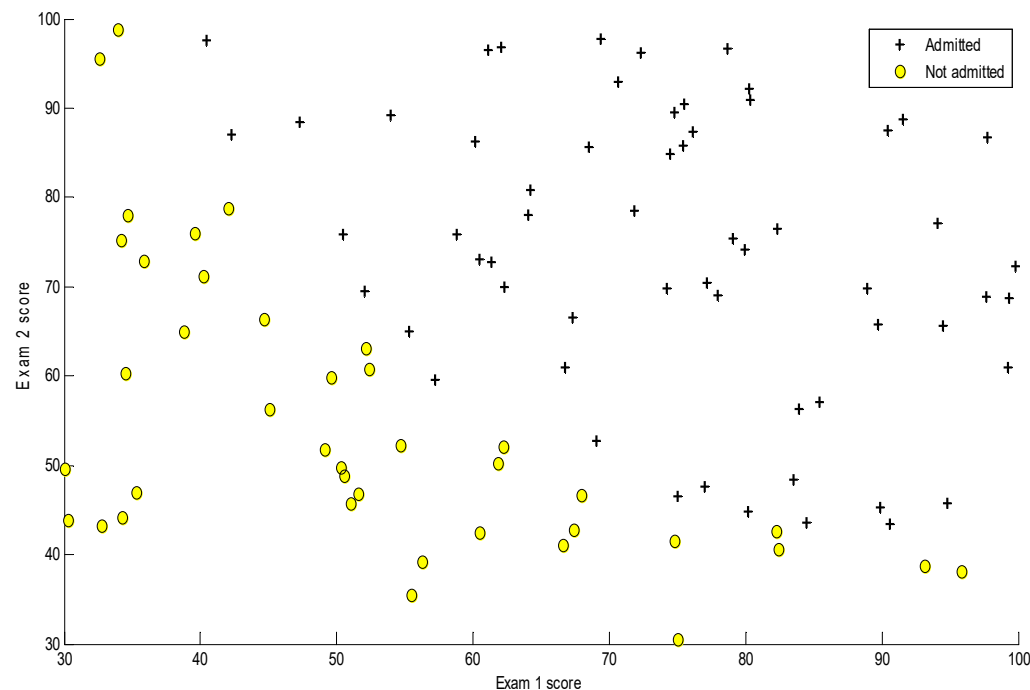
### **Advantages:**

- no need to manually choose the learning rate ( $\alpha$ )
- often faster than gradient descent

**Disadvantages:** more complex algorithms

# Logistic regression - example

Training data: applicant's scores on two exams and the admission decision (historical data). Build a logistic regression model to predict whether a student gets admitted into a university.



**Fig. 1 Scatter plot of training data**

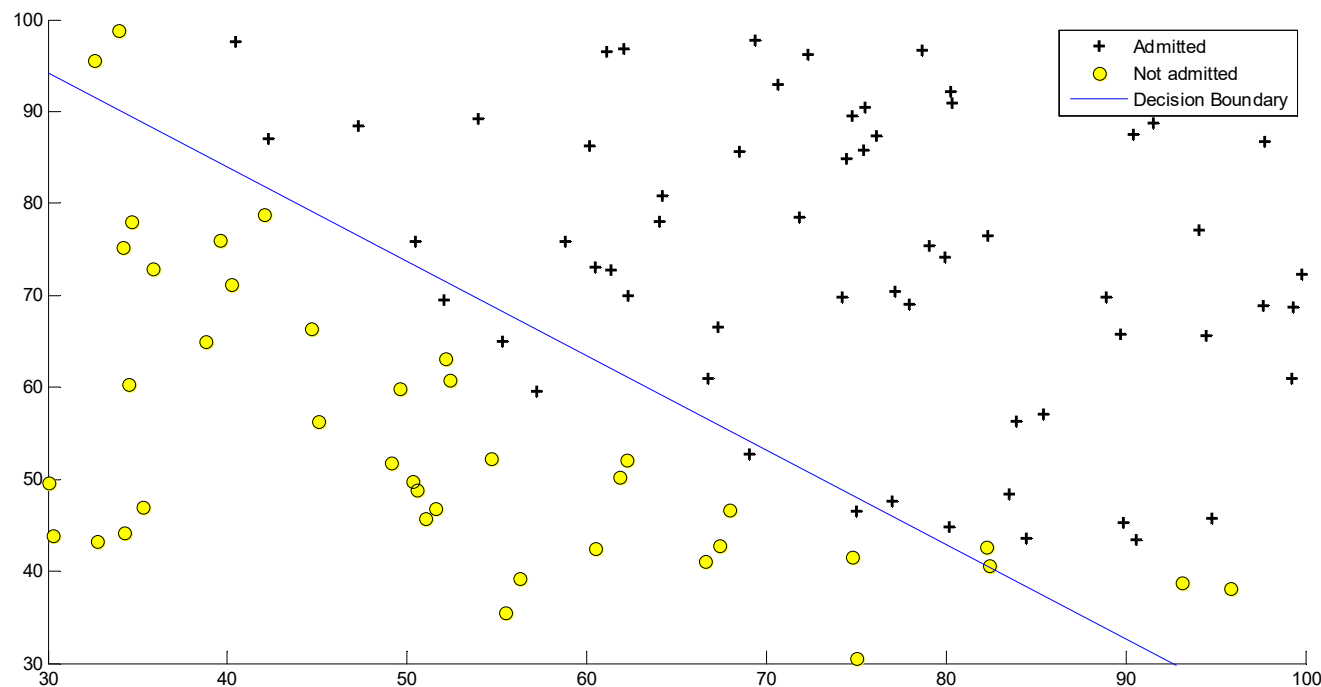
# Logistic regression - example

$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0 \Rightarrow$  decision boundary

*if*  $z > 0 \Rightarrow g(z) > 0.5 \Rightarrow$  predict class = 1

*if*  $z < 0 \Rightarrow g(z) < 0.5 \Rightarrow$  predict class = 0

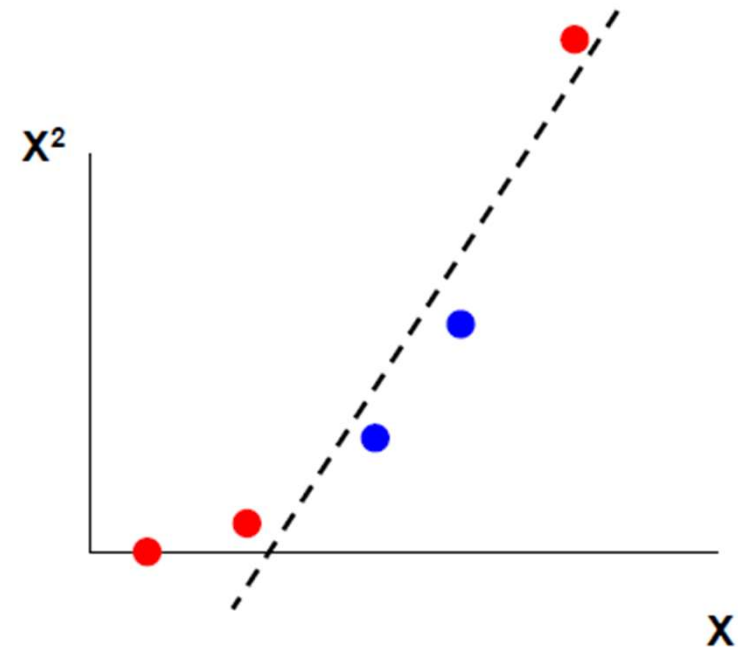
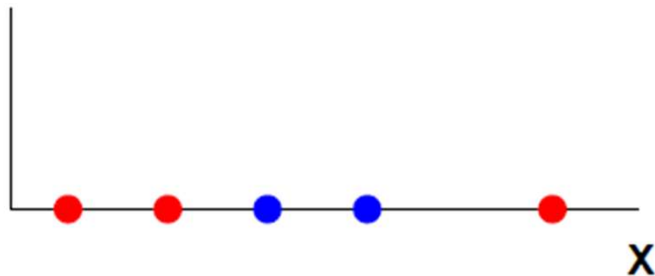
**Scatter plot of training data and linear decision boundary with the optimized parameters**



# Nonlinearly Separable Data

**Linear classifier cannot classify these examples.**

**And now ?**



$$z = \theta^T x = \theta_0 + \theta_1 x + \theta_2 x^2 = 0 \Rightarrow$$

Nonlinear decision boundary (in the original feature space  $x$ )

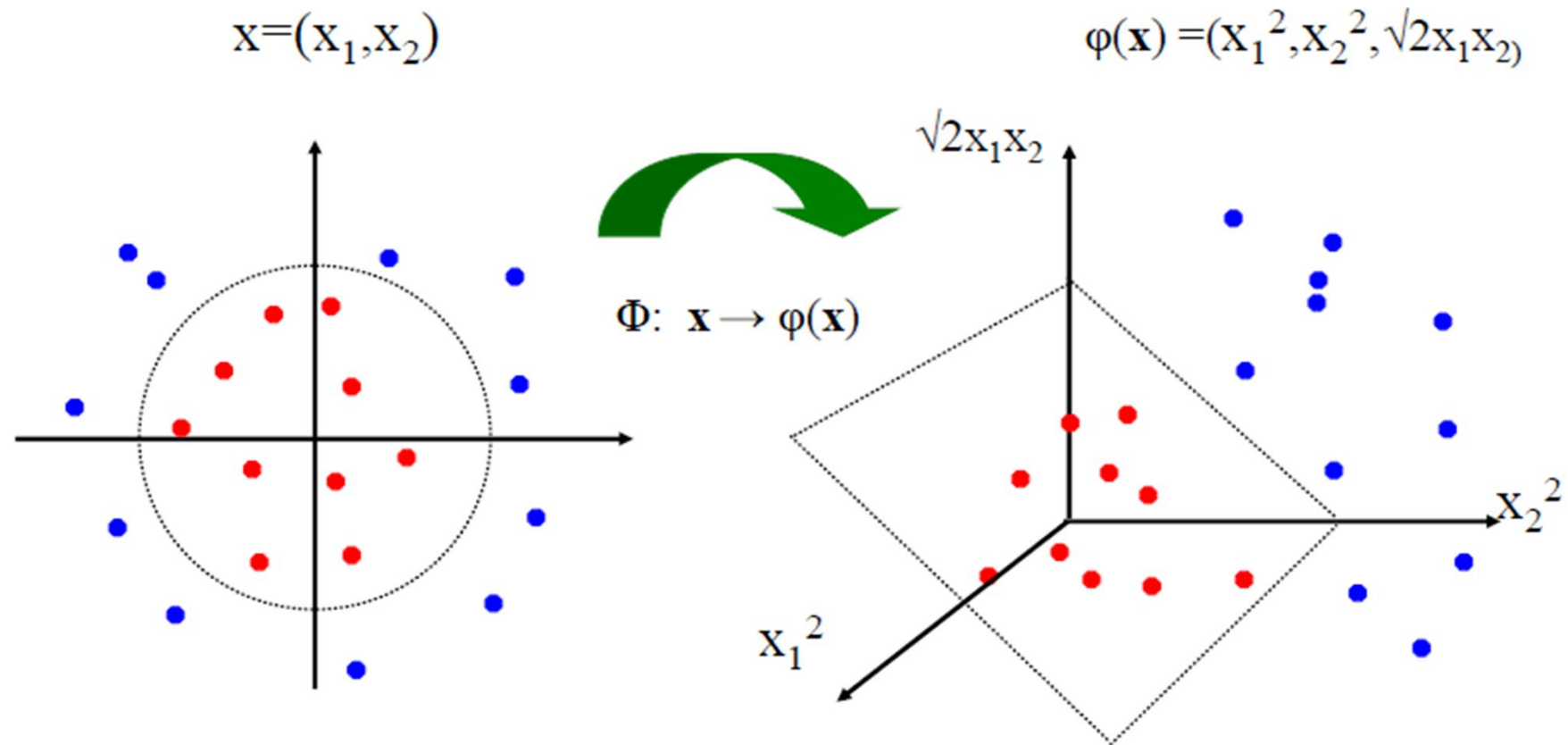
Linear decision boundary (in the extended feature space  $x, x^2$ )

*if*  $z > 0 \Rightarrow g(z) > 0.5 \Rightarrow \text{predict class} = 1$

*if*  $z < 0 \Rightarrow g(z) < 0.5 \Rightarrow \text{predict class} = 0$

# Nonlinearly Separable Data

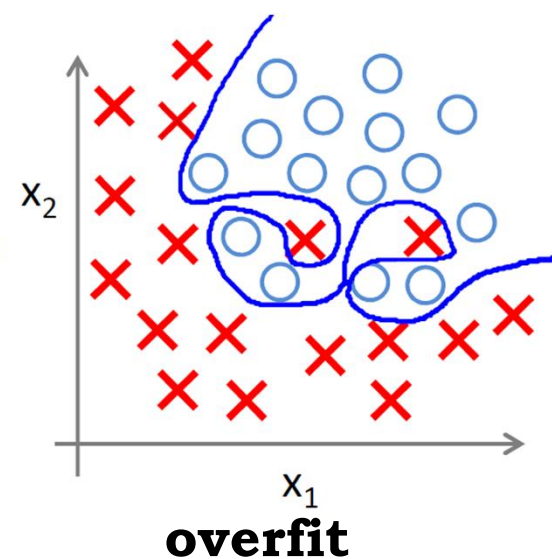
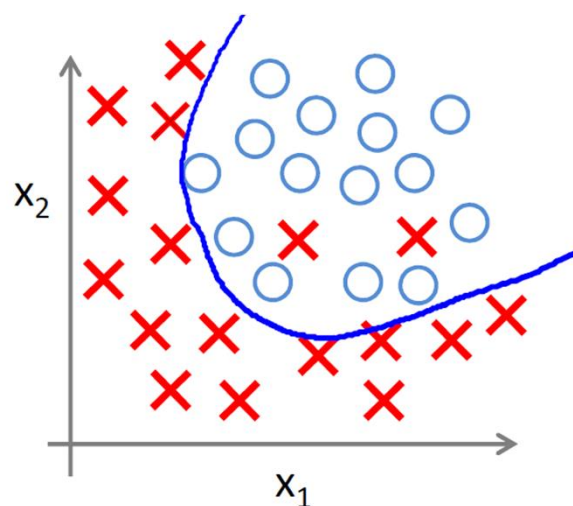
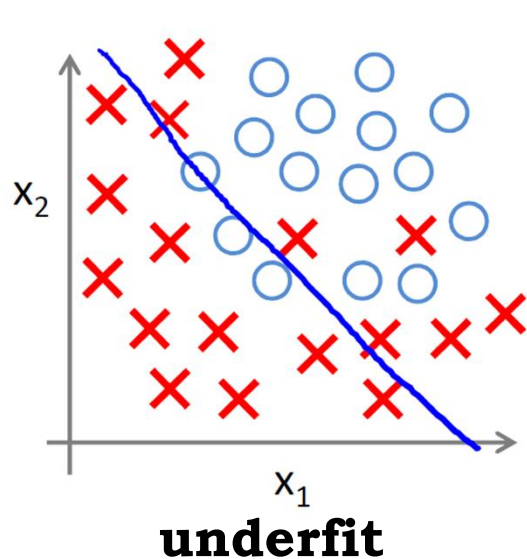
- The original input space ( $\mathbf{x}$ ) can be mapped to some higher-dimensional feature space ( $\phi(\mathbf{x})$ ) where the training set is separable:



This slide is courtesy of [www.iro.umontreal.ca/~pift6080/documents/papers/svm\\_tutorial.ppt](http://www.iro.umontreal.ca/~pift6080/documents/papers/svm_tutorial.ppt)

# Overfitting problem

**Overfitting:** If we have too many features, the learned hypothesis (model) may fit the training data very well but fail to generalize to new examples.



# Regularization

Regularization is a popular method in ML to prevent overfitting by reducing the model parameters  $\theta$  towards zero

## 1 Ridge Regression

- Keep all the features, but reduces the magnitude of  $\theta$ .
- Works well when each of the features contributes a bit to predict  $y$ .

## 2 Lasso Regression

- May shrink some coefficients of  $\theta$  to exactly zero.
- Serve as a feature selection tools (reduces the number of features).

# Regularized Logistic Regression

**Unregularized Log Reg cost function:**

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

**Regularized Log Reg cost function (ridge regression)**

*$\lambda$  is the regularization parameter (hyper-parameter) that needs to be selected*

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m [-y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

Small  $\lambda \Rightarrow$  lower bias, higher variance

High  $\lambda \Rightarrow$  higher bias, lower variance



# Regularized Logistic Regression

**Unregularized cost function gradients (for all parameters  $j=0,1, \dots,n$ )**

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

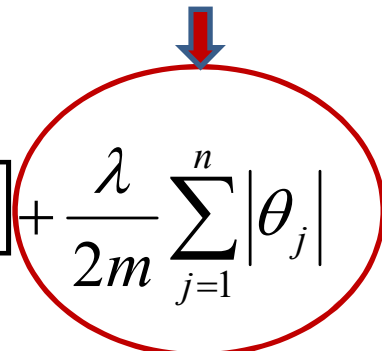
**Regularized cost function gradient for  $j=0$  (no change)**

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

**Regularized cost function gradients for  $j=1,2,\dots,n$**

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

# Regularization: Lasso Regression

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[ -y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n |\theta_j|$$


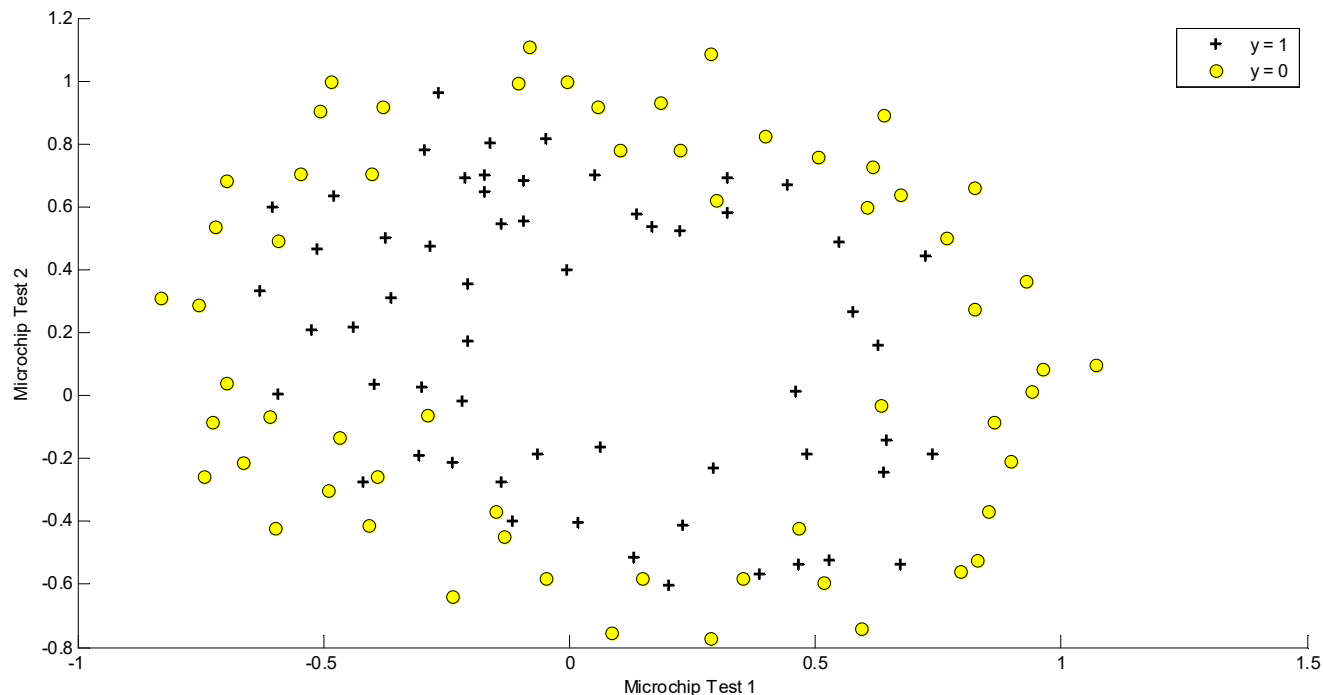
Ridge Regression shrinks  $\theta$  towards zero, but never equal to zero => all features are included in the model no matter how small are the coefficients.

Lasso Regression is able to shrink coefficients to exactly zero => reduces the number of features. This makes Lasso Regression useful in cases with high dimension and helps with model interpretability.

Lasso Regression involves absolute values (not differentiable) => computing is difficult. Various algorithms are available in sklearn Python library.

# Regularized Log Reg -example

Predict whether microchips from a fabrication plant passes quality assurance (QA). During QA, each microchip goes through various tests to ensure it is functioning correctly. Suppose we have the test results for some microchips on two different tests. From these two tests, we would like to determine whether the microchips should be accepted ( $y=1$ ) or rejected ( $y=0$ ).



# Regularized Log Reg -example

Dataset is not linearly separable  $\Rightarrow$  logistic regression will only be able to find a linear decision boundary. One way to fit the data better is to create more features. For example add polynomial terms of  $x_1$  and  $x_2$ .

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1x_2^5 \\ x_2^6 \end{bmatrix}$$

NONLINEAR decision boundary  $\Rightarrow$

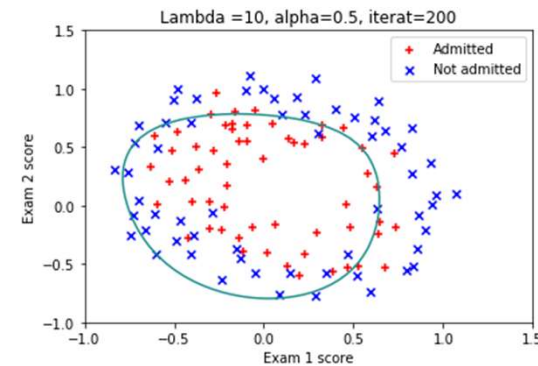
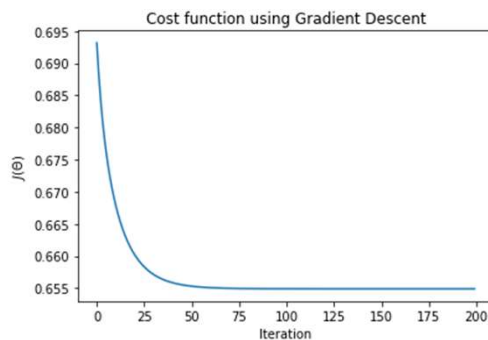
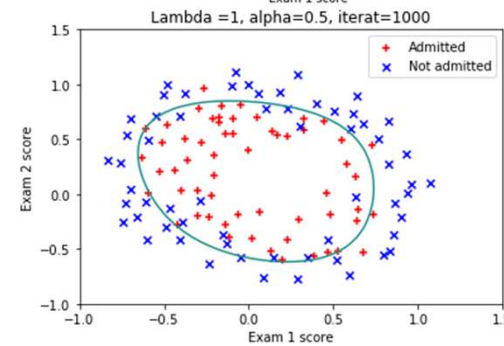
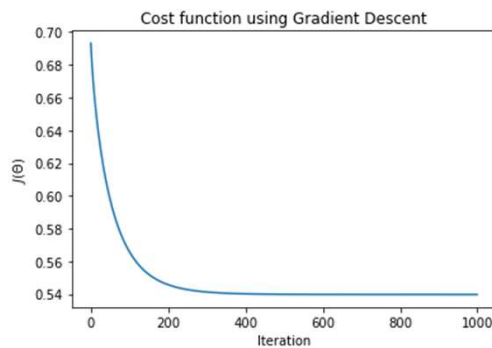
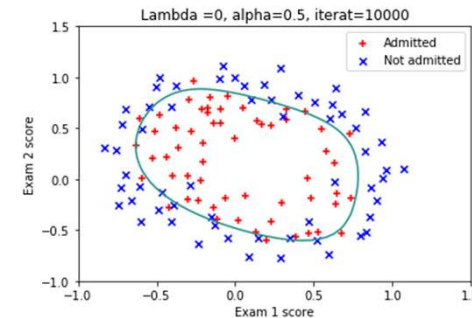
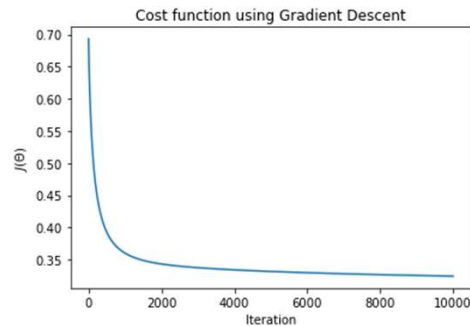
$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \dots \theta_{28} x_2^6 = 0$$

if  $z > 0 \Rightarrow g(z) > 0.5 \Rightarrow$  predict class = 1

if  $z < 0 \Rightarrow g(z) < 0.5 \Rightarrow$  predict class = 0

# Regularized Log Reg -example

Accuracy on training data: :84.75% ( $\lambda=0$ ) | 83.90 % ( $\lambda=1$ ) | 71.2 % ( $\lambda=10$ ) )



# Multiclass Classification

Exs. *Email classification* (work, friends, family)

*Medical diagnosis* (not ill, cold, flu) ; *Weather* (sunny, cloudy, rain, snow)

## One-versus-all strategy :

For K classes train K binary classifiers:

for  $c=1:K$

    Make  $y_{\text{binary}}=1$  (only for examples of class  $c$ )

$y_{\text{binary}}=0$  (for examples of all other classes)

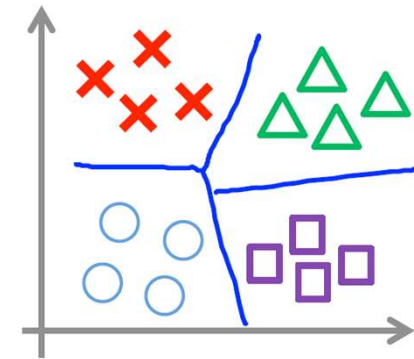
$\theta$ =Train classifier with training data  $X$  and output  $y_{\text{binary}}$ .

    Save the learned parameters of all classifiers in one matrix

    where each row is the learned parameters of one classifier:

$\theta_{\text{all}}(c,:)=\theta$

end



New example: winner-takes-all strategy, the binary classifier with the highest output score assigns the class.