线性回归与线性分类

Linear Regression and Linear Classification

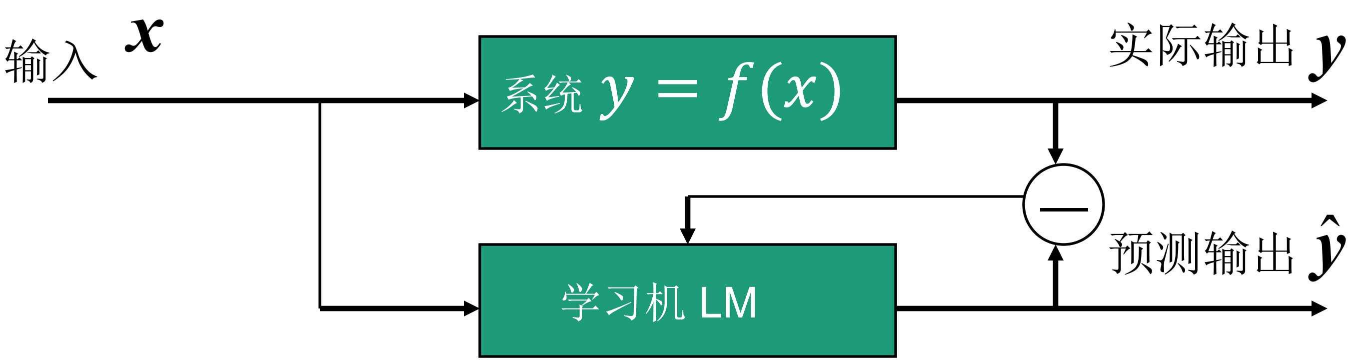
洪 波

Dept. of Biomedical Engineering

Tsinghua University

机器学习
Machine Learning

未知函数，未知分类器



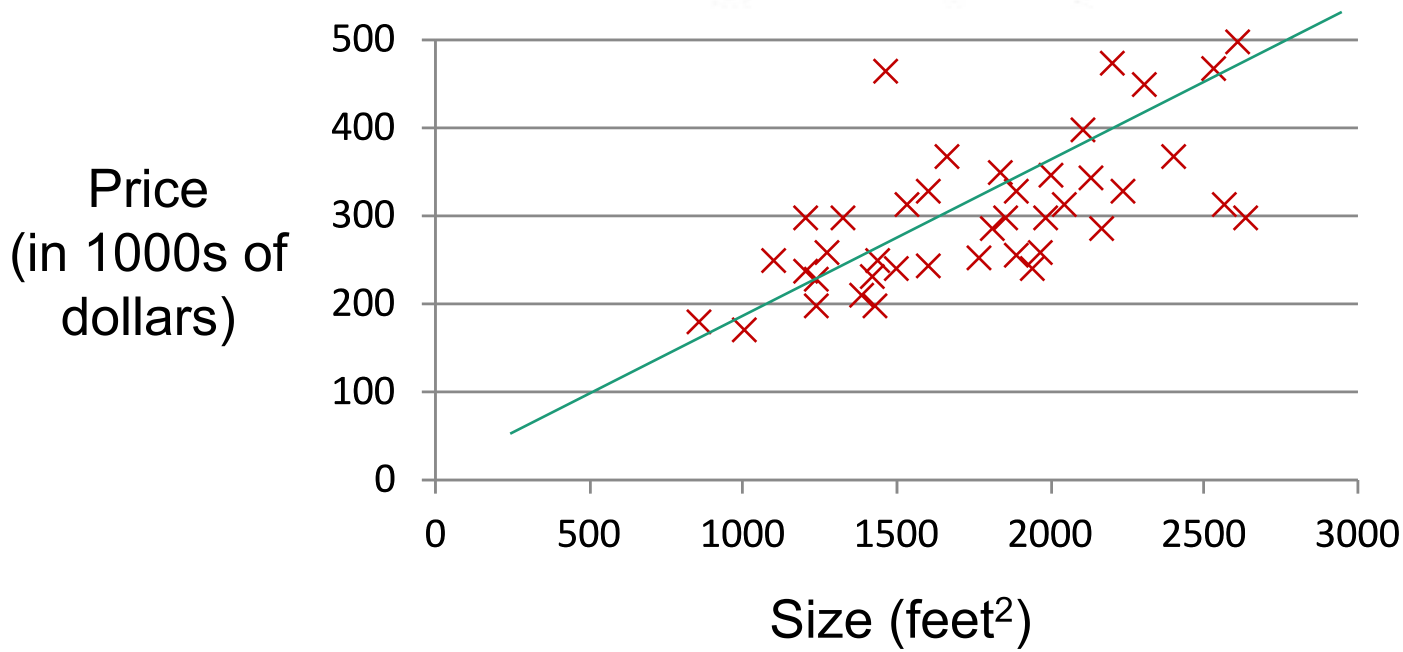
函数逼近，模式分类

“学习材料”：一组观测样本　$(x_1, y_1), (x_1, y_1), \cdots, (x_n, y_n)$

Linear Regression 线性回归

Linear Regression (Housing Prices Prediction)

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



Regression Problem: Predict real-valued output

Modified from Andrew Ng ML

Parameter optimization

Training Set

Size in feet^2 (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
…	…

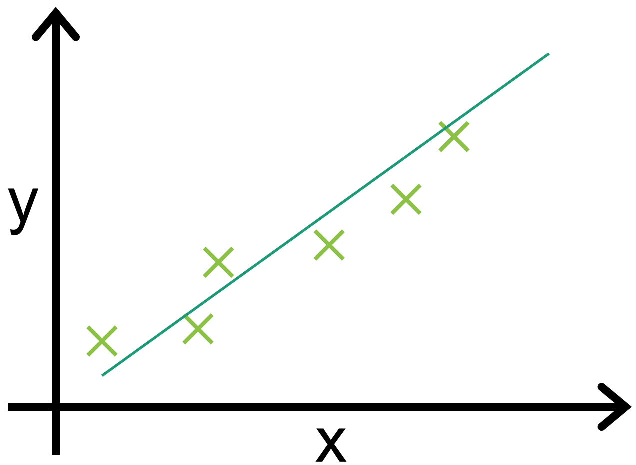
Hypothesis: $$h_\theta(x) = \theta_0 + \theta_1 x$$

θ_i's: Parameters

How to choose θ_i's ?

Modified from Andrew Ng ML

Cost Function 代价函数/目标函数



Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

代价函数 $$J(\boldsymbol{\theta}) = J(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1) = \frac{1}{2m}\sum_{i=1}^{m}(h(\boldsymbol{x}^{(i)}) - \boldsymbol{y}^{(i)})^2$$

简化模型 – 只考虑一个参数 θ_1

Hypothesis:

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

Parameters:

$$\theta_0, \theta_1$$

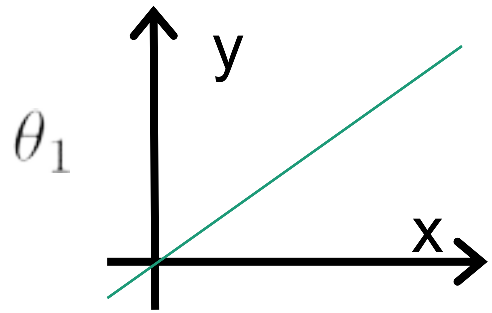
Cost Function:

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

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

<u>Simplified</u>

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



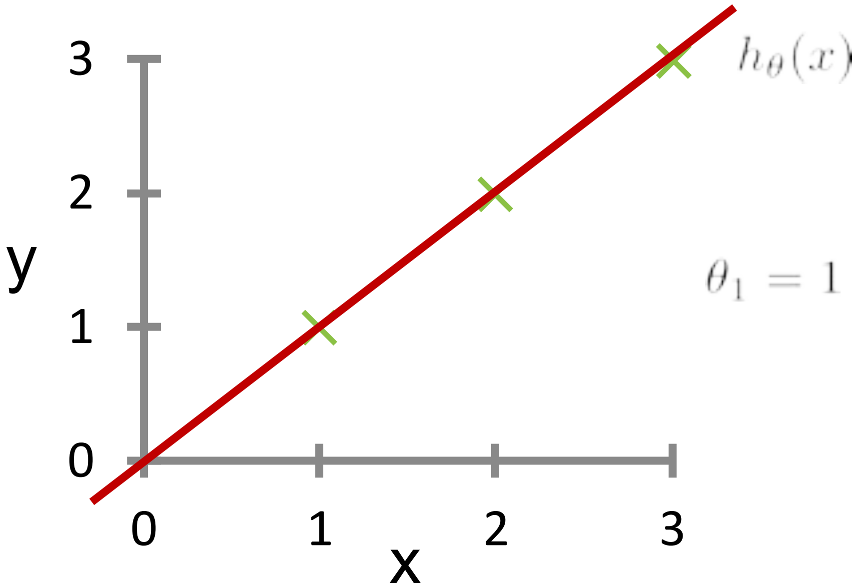
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

Modified from Andrew Ng ML

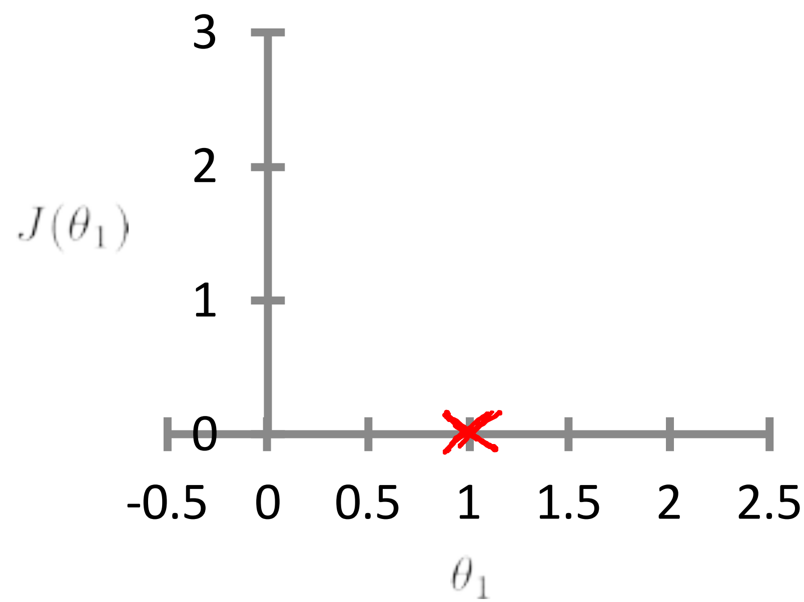
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)

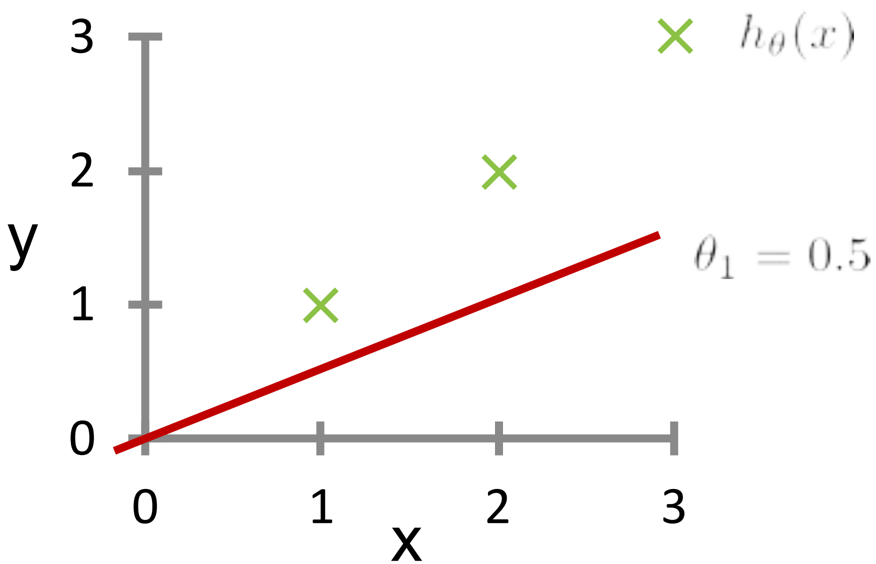


$$J(\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Modified from Andrew Ng ML

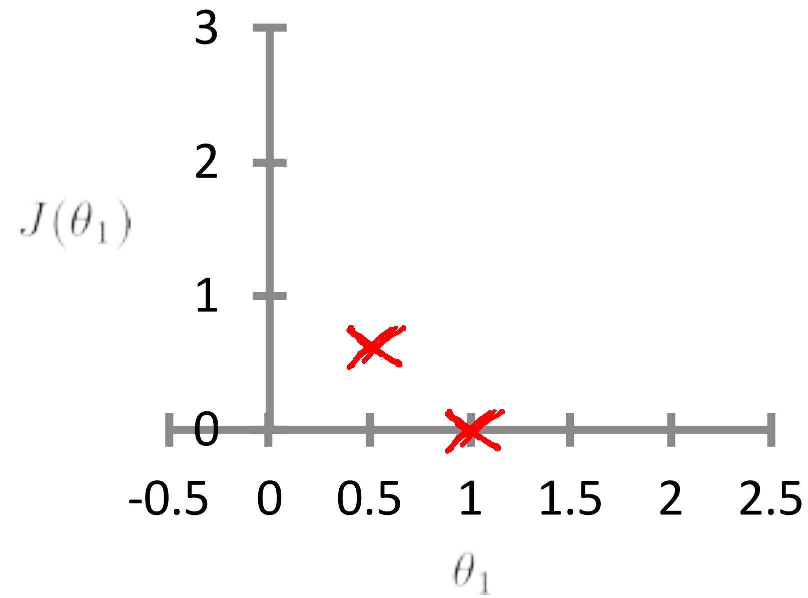
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)

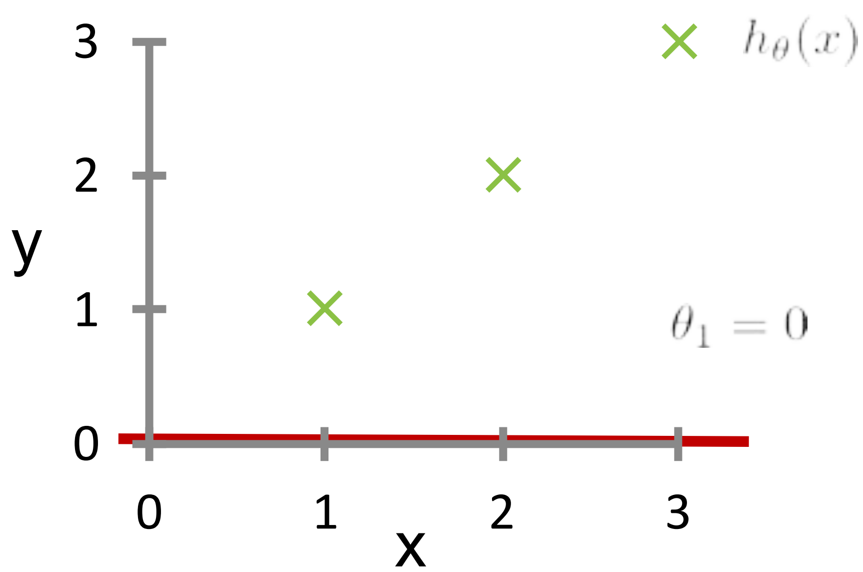


$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Modified from Andrew Ng ML

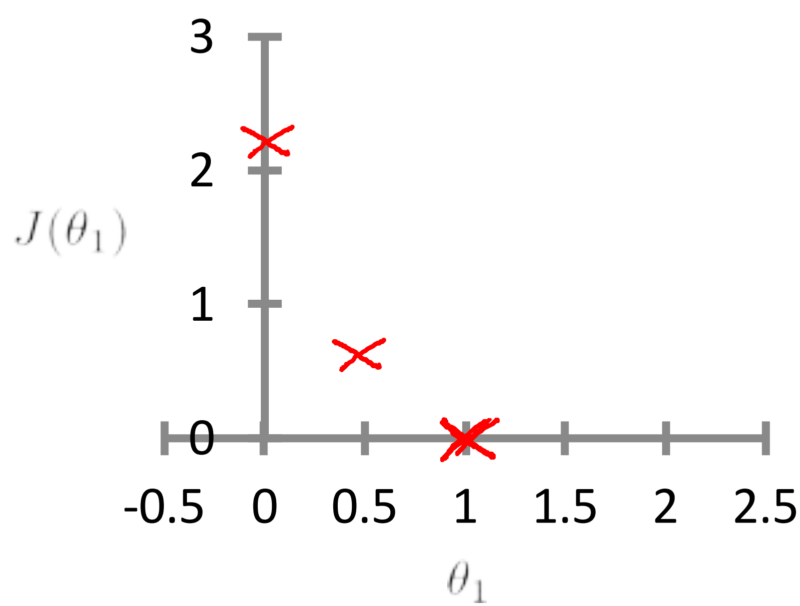
$h_\theta(x)$

(for fixed θ_1, this is a function of x)



$J(\theta_1)$

(function of the parameter θ_1)



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

Modified from Andrew Ng ML

回到两个参数的情况 θ_0, θ_1

模型假设: $h_\theta(x) = \theta_0 + \theta_1 x$

模型参数: θ_0, θ_1

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

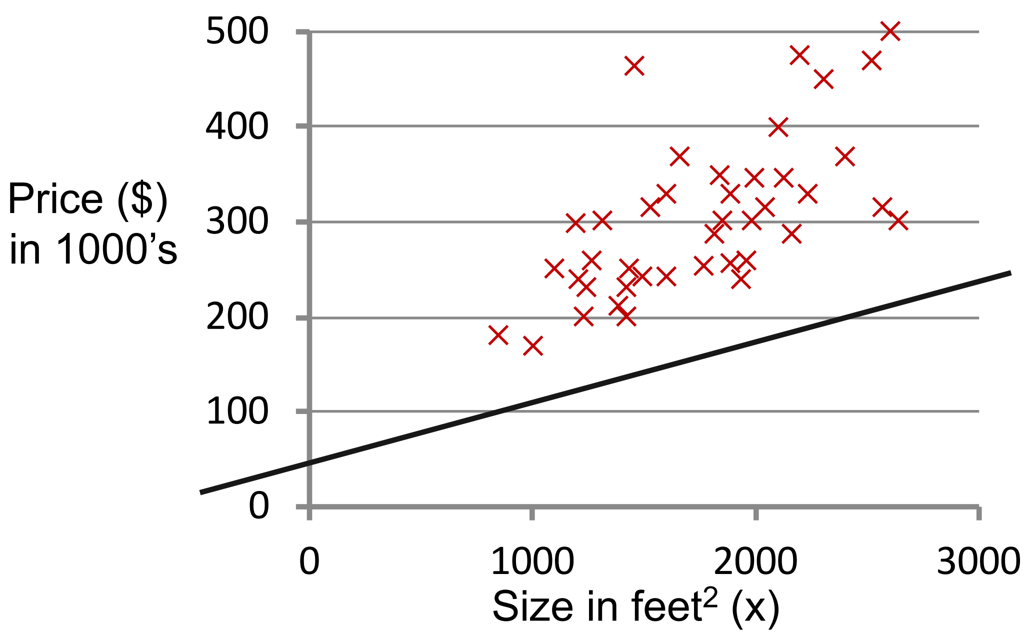
优化目标: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Modified from Andrew Ng ML

代价函数什么样？

$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$$h_\theta(x) = 50 + 0.06x$$

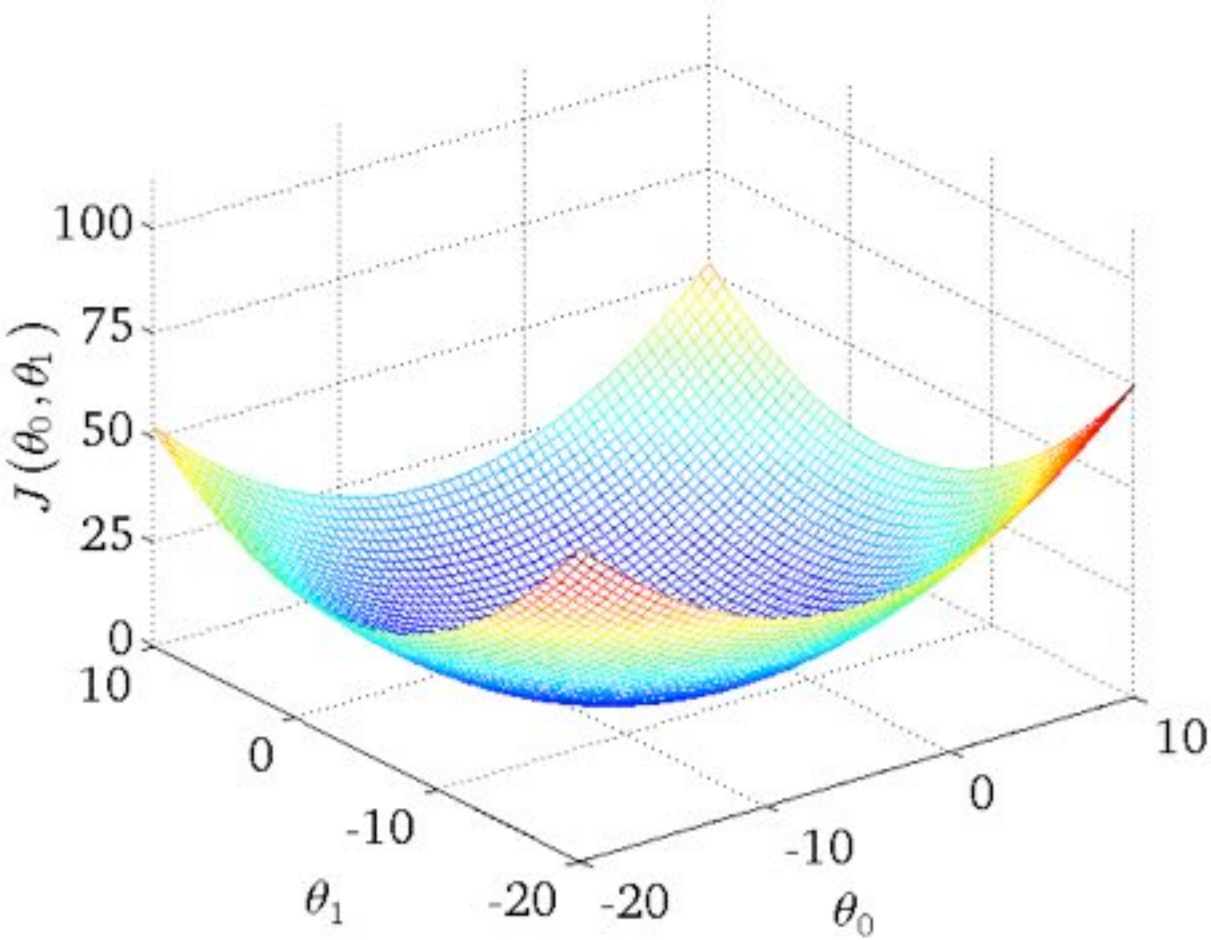
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

?

Modified from Andrew Ng ML

代价函数

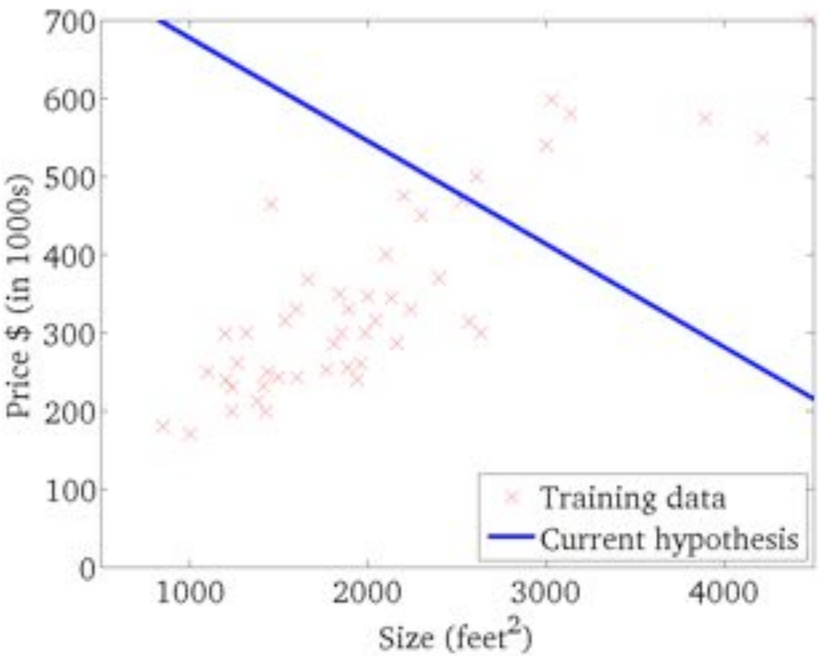


Modified from Andrew Ng ML

代价函数与模型参数

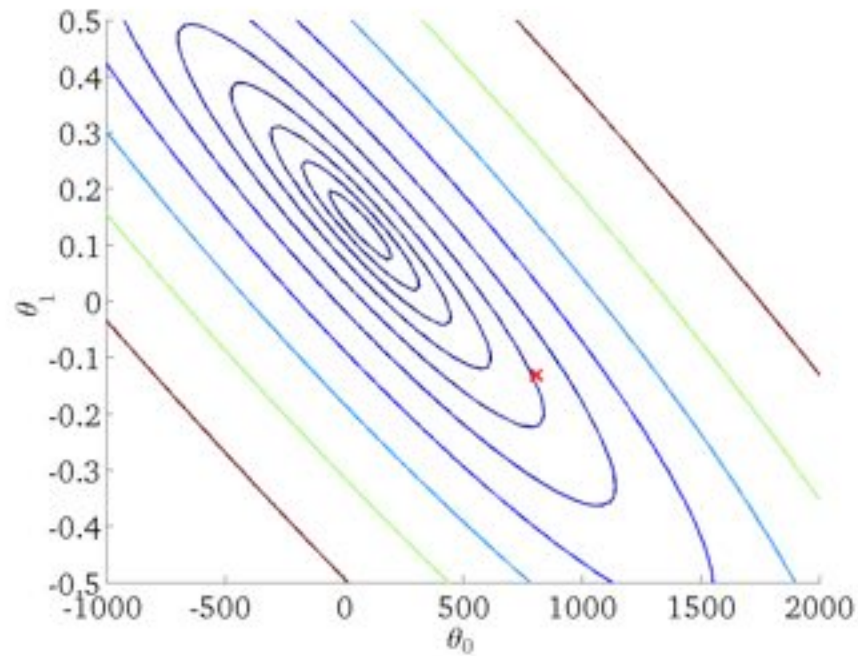
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)

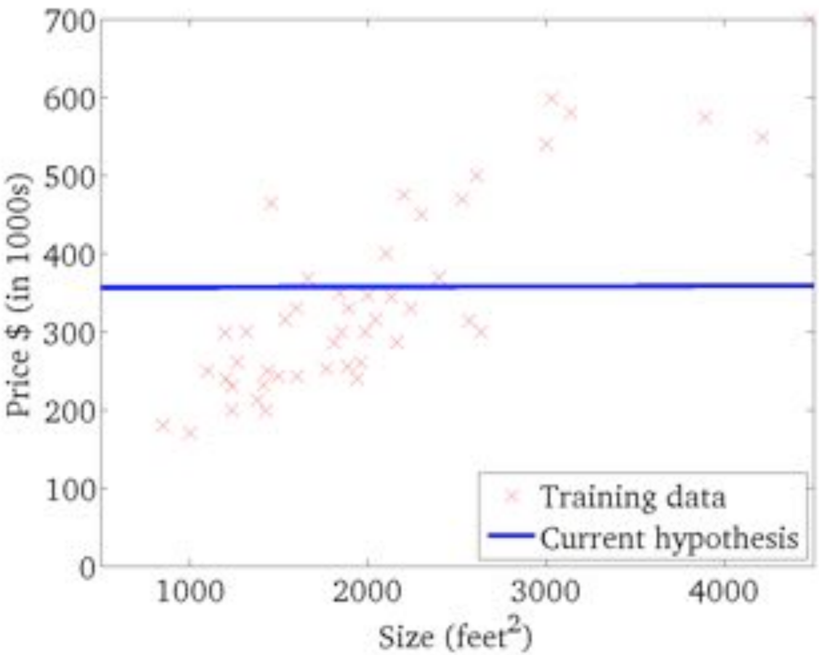


Modified from Andrew Ng ML

代价函数与模型参数

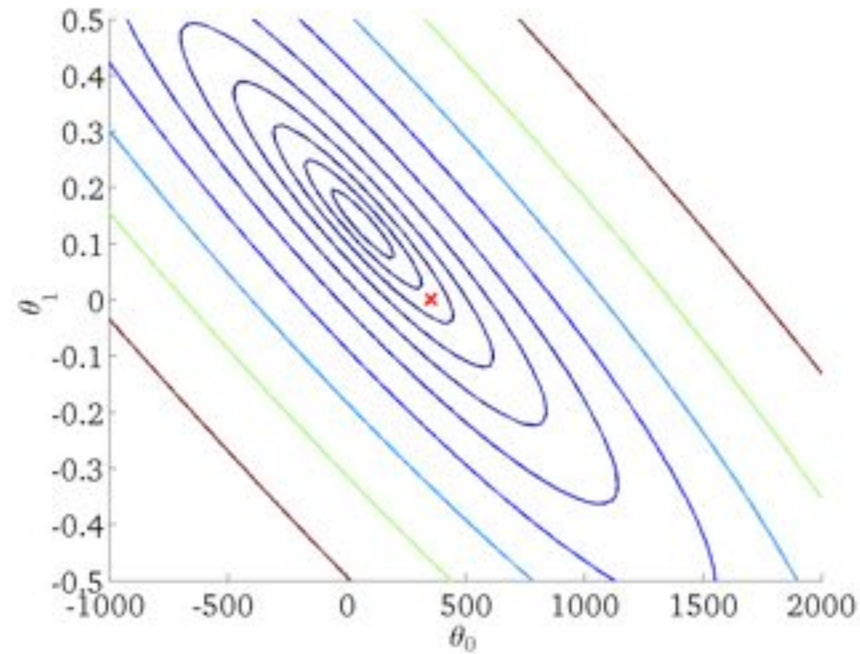
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

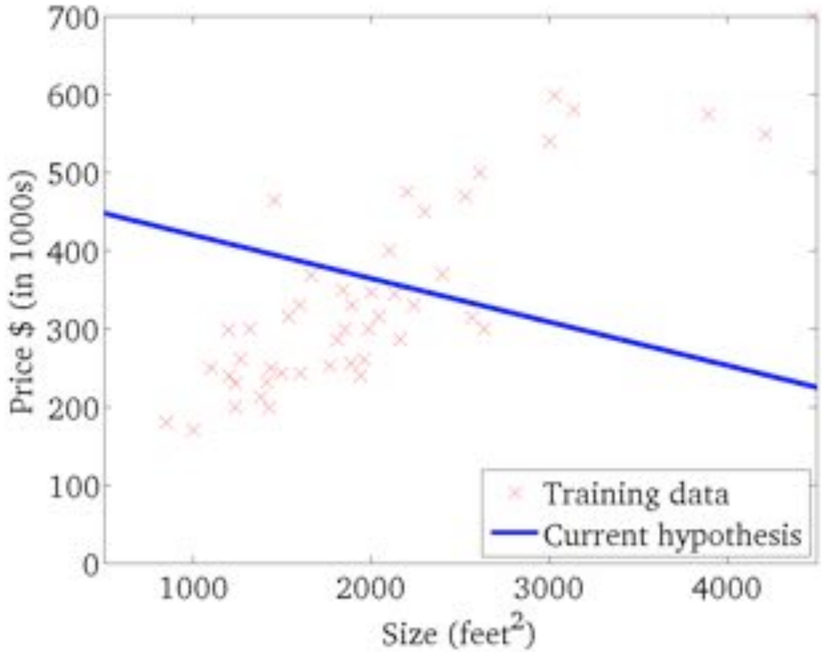


Modified from Andrew Ng ML

代价函数与模型参数

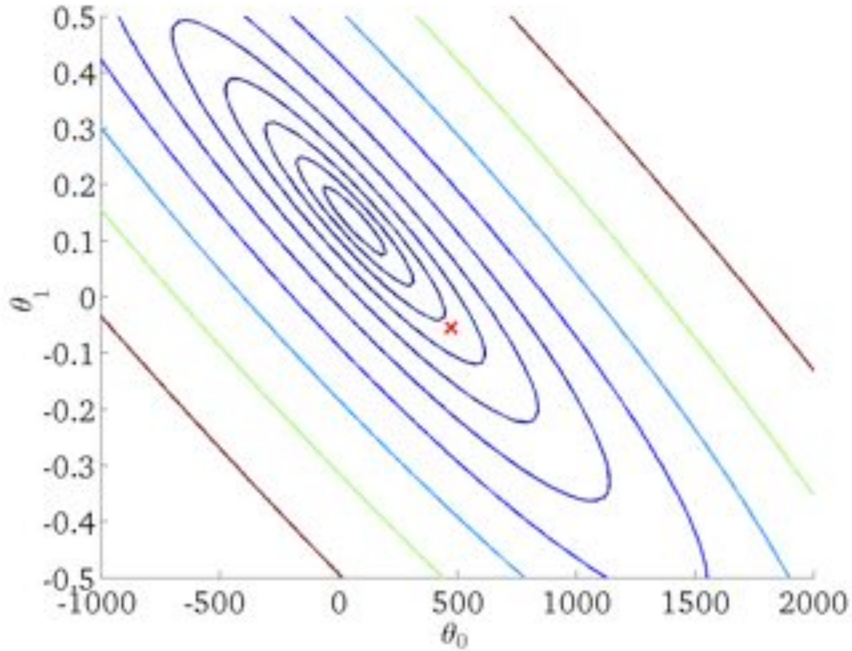
$$h_\theta(x)$$

(for fixed θ_0, θ_1, this is a function of x)



$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



Modified from Andrew Ng ML

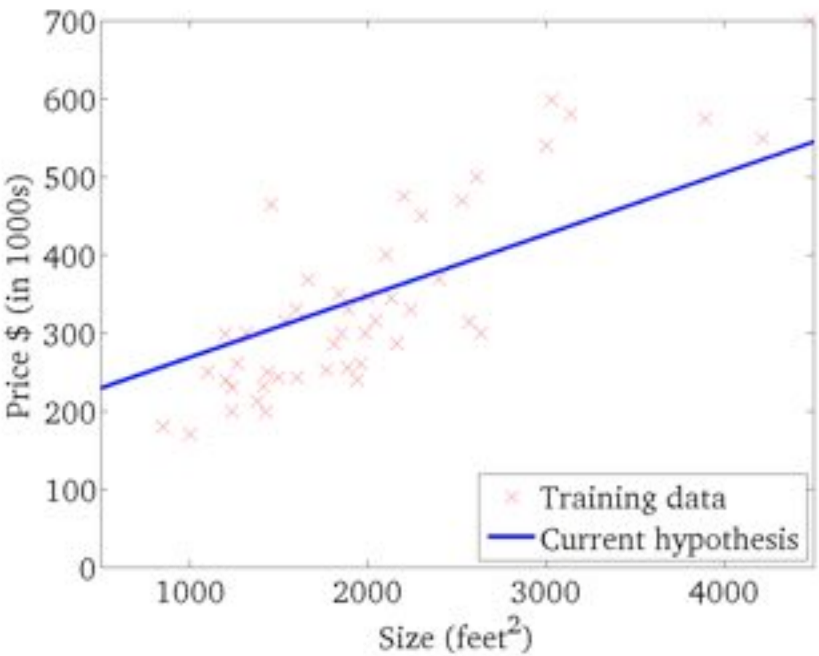
代价函数与模型参数

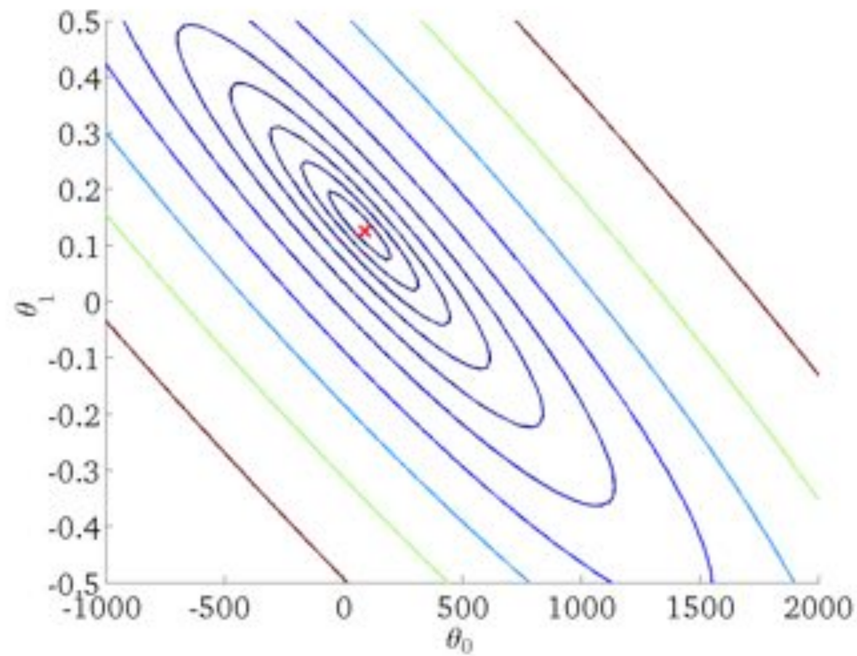
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

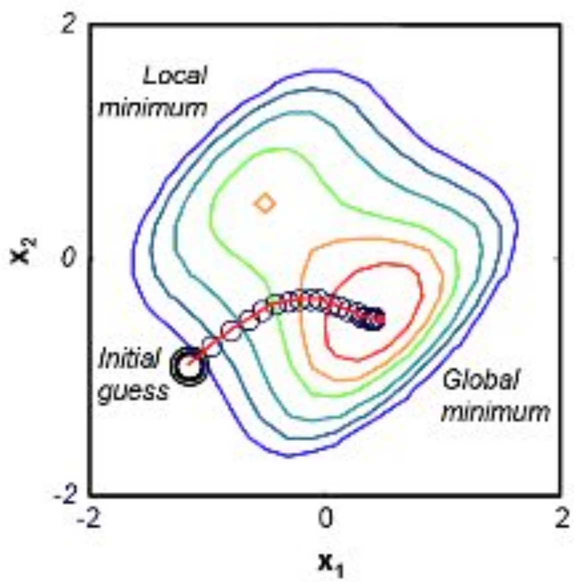




Modified from Andrew Ng ML

Gradient descent 梯度下降算法

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



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

$$\theta_0 = \theta_0 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(h(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}\right)$$

$$\theta_1 = \theta_1 - \frac{\alpha}{m}\sum_{i=1}^{m}\left(\left(h(\mathbf{x}^{(i)}) - \mathbf{y}^{(i)}\right)x_i\right)$$

Modified from Andrew Ng ML

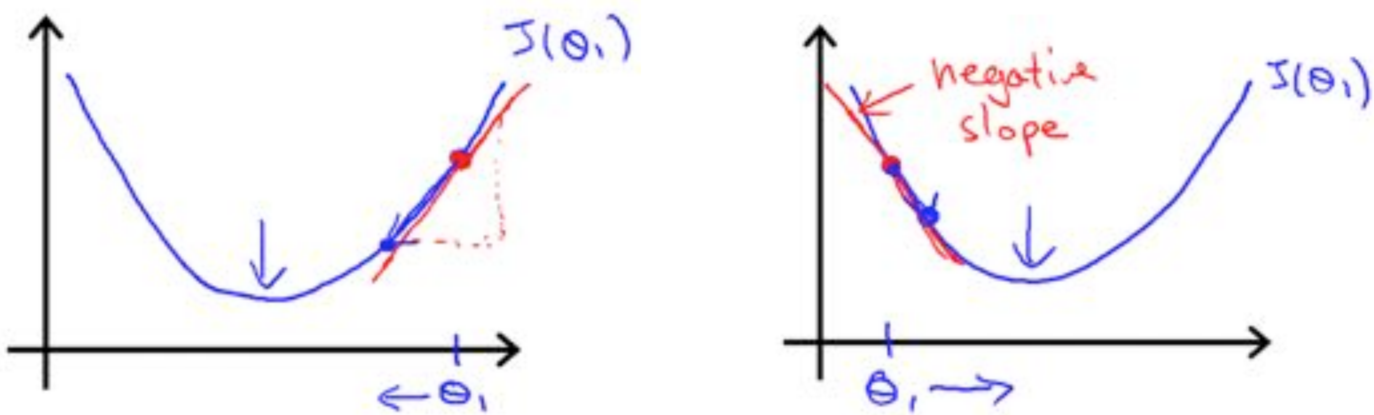
Gradient descent 梯度下降算法

repeat until convergence {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

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

}

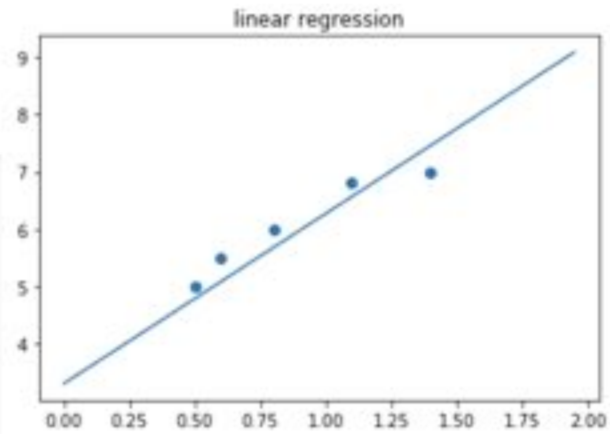


Modified from Andrew Ng ML

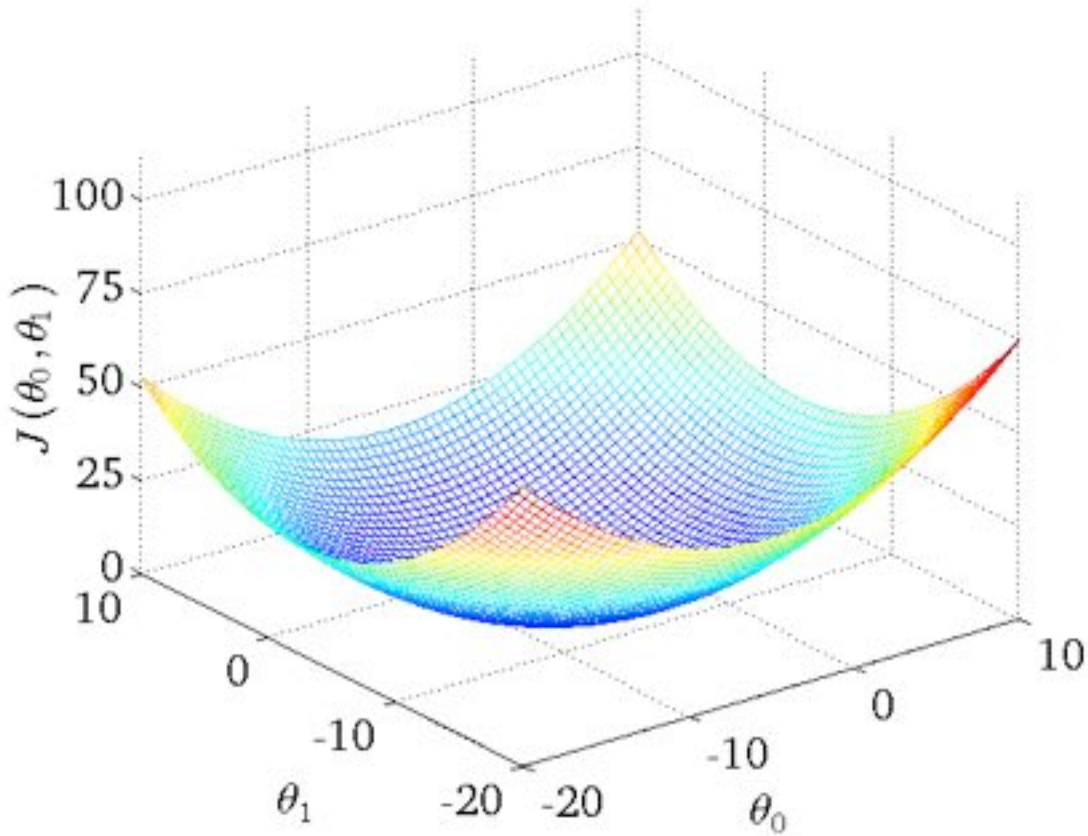
线性回归算法Python实现

Algorithm for Linear Regression

```
import numpy as np
wheat_and_bread = np.array([[0.5,5],[0.6,5.5],[0.8,6],[1.1,6.8],[1.4,7]])
sample_x=wheat_and_bread[:,0]
sample_y=wheat_and_bread[:,1]
def step_gradient(b_current, m_current, points, learningRate):
    b_gradient = 0
    m_gradient = 0
    N = float(len(points))
    for i in range(0, points.shape[0]):
        x = points[i][0]
        y = points[i][1]
        b_gradient += -(2/N) * (y - ((m_current * x) + b_current))
        m_gradient += -(2/N) * x * (y - ((m_current * x) + b_current))
    new_b = b_current - (learningRate * b_gradient)
    new_m = m_current - (learningRate * m_gradient)
    return [new_b, new_m]
def compute_value(b,m,point):
    return m*point+b
range_x=np.arange(0,2,0.05)
def gradient_descent_runner(points, starting_b, starting_m, learning_rate, num_iterations):
    b = starting_b
    m = starting_m
    plt.ion()
    for i in range(num_iterations):
        b, m = step_gradient(b, m, points, learning_rate)
        plt.cla()
        plt.scatter(sample_x,sample_y)
        plt.plot(range_x,compute_value(b,m,range_x))
        plt.pause(0.01)
    plt.ioff()
    plt.title('linear regression')
    plt.show()

    #return [b, m]
gradient_descent_runner(wheat_and_bread, 1, 1, 0.01, 100)
```



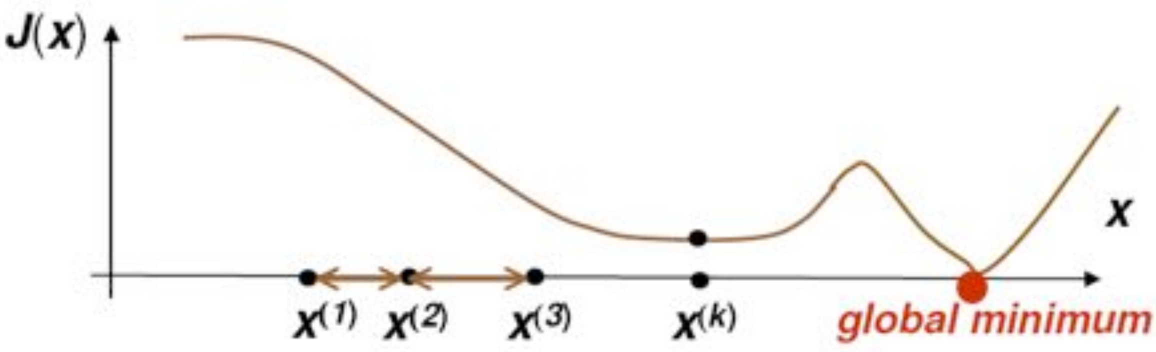
Convex Optimization 凸优化问题

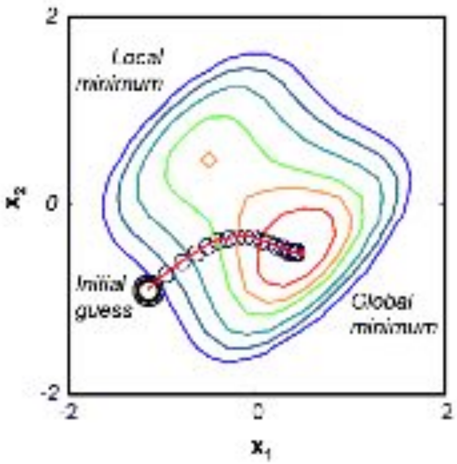


Modified from Andrew Ng ML

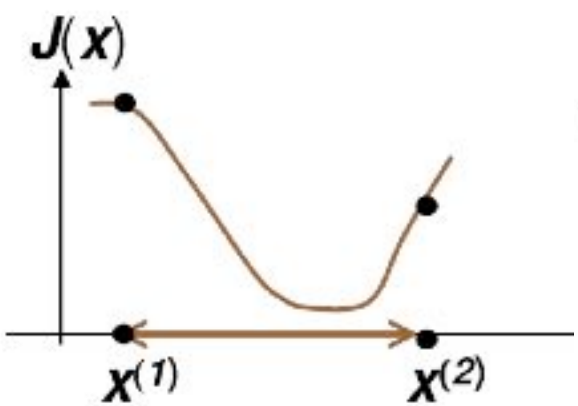
梯度下降法的两个问题

1. 如果不是凸优化问题，不能保证全局最优，会落入局部极值点

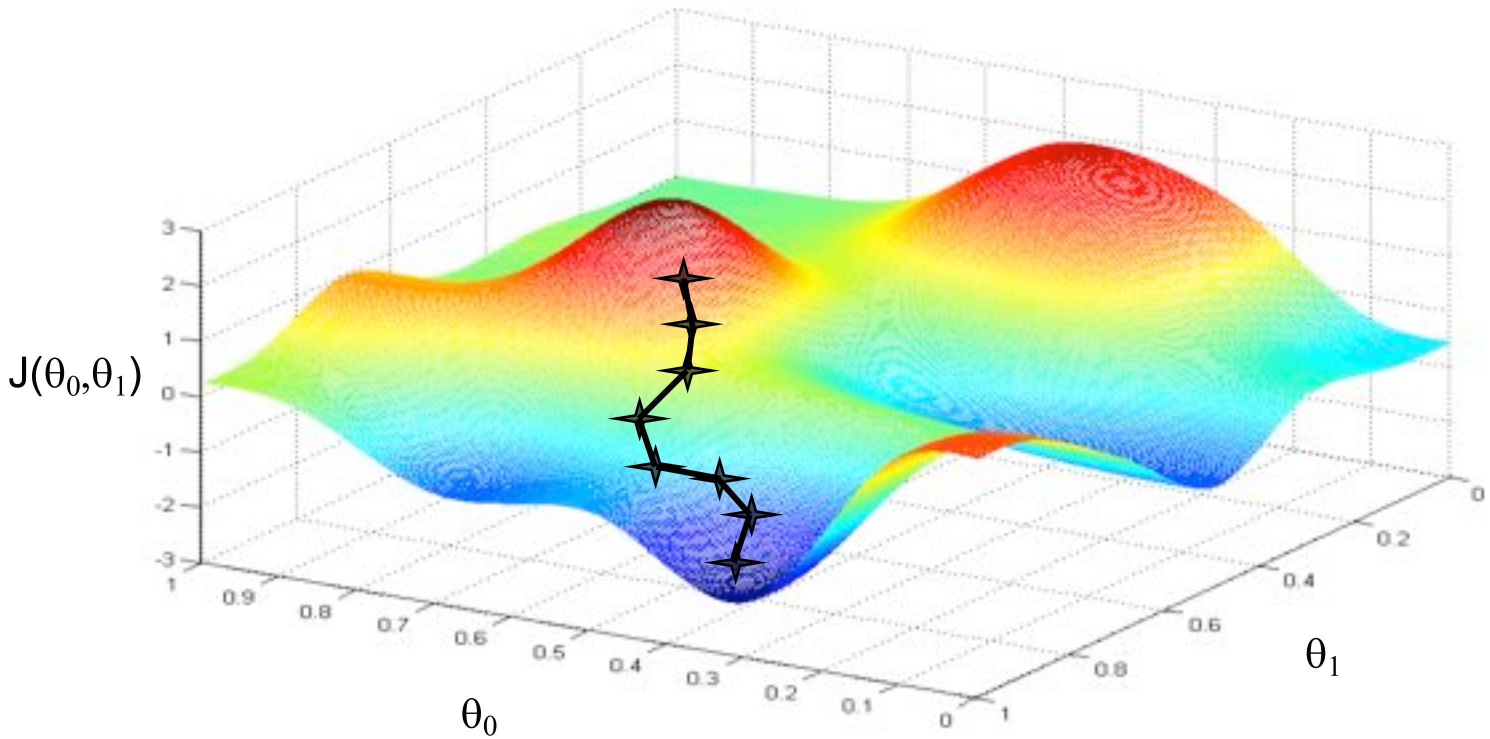




2. 学习率η不能太大，太小也不好

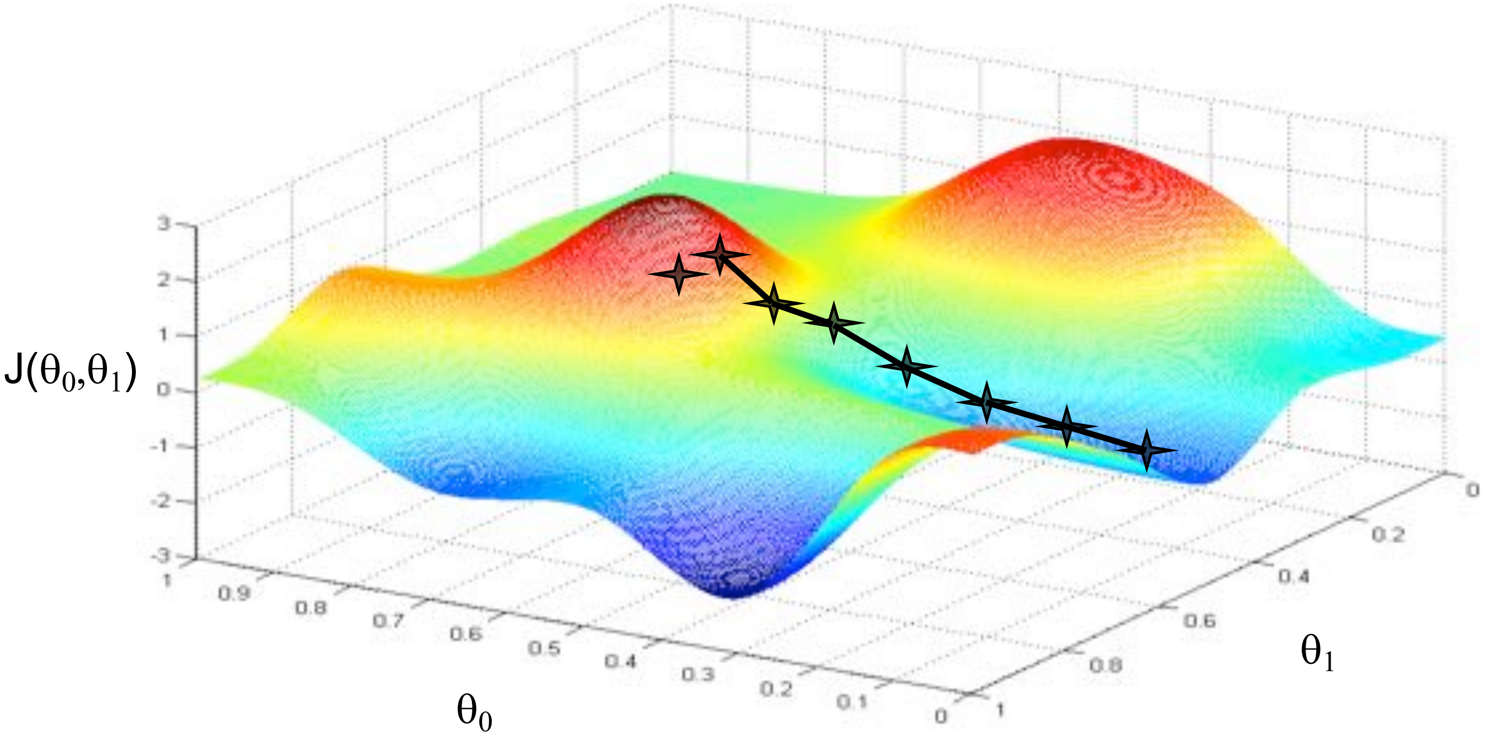


寻优结果对初始值的敏感性



Modified from Andrew Ng ML

寻优结果对初始值的敏感性



Modified from Andrew Ng ML

寻优算法 Advanced Optimization algorithm

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial\theta_j}J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
  梯度下降
- Conjugate gradient
  共轭梯度
- BFGS/L-BFGS
  拟牛顿法

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

Modified from Andrew Ng ML

多变量问题 Multiple features (variables)

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Modified from Andrew Ng ML

多变量线性回归

$$h_{\theta}(x)=\theta_0+\theta_1x_1+\theta_2x_2+\cdots+\theta_nx_n$$

$$h_{\theta}(x)=\begin{bmatrix}\theta_0 & \theta_1 & \cdots & \theta_n\end{bmatrix}\begin{bmatrix}x_0\\x_1\\\vdots\\x_n\end{bmatrix}=\theta^{\mathrm{T}}x$$

$$h_{\theta}(X)=X\theta$$

样本维数n

$$X=\begin{bmatrix}x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \cdots & x_n^{(1)}\\x_0^{(2)} & x_1^{(2)} & x_2^{(2)} & \cdots & x_n^{(2)}\\\vdots & \vdots & \vdots & \ddots & \vdots\\x_0^{(m)} & x_1^{(m)} & x_2^{(m)} & \cdots & x_n^{(m)}\end{bmatrix},\theta=\begin{bmatrix}\theta_0\\\theta_1\\\theta_2\\\vdots\\\theta_n\end{bmatrix}$$

样本个数m

在scikit-learn中，训练样本就是用m×n维的矩阵来表达的

多维情况的代价函数和梯度下降算法

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

$$\Downarrow$$

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

$$\boldsymbol{\theta}_j=\boldsymbol{\theta}_j-\alpha\frac{\partial}{\partial\boldsymbol{\theta}_j}J(\boldsymbol{\theta})$$

$$\boldsymbol{\theta}_j=\boldsymbol{\theta}_j-\frac{\alpha}{m}\sum_{i=1}^{m}\left(\left(h(\boldsymbol{x}^{(i)})-\boldsymbol{y}^{(i)}\right)\boldsymbol{x}_j^{(i)}\right)$$

算法伪代码

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 $j = 0, \ldots, n$)

Modified from Andrew Ng ML

梯度下降：从单变量到多变量

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 θ_0, θ_1)

}

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 for θ_j
$j = 0, \ldots, n$)

}

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

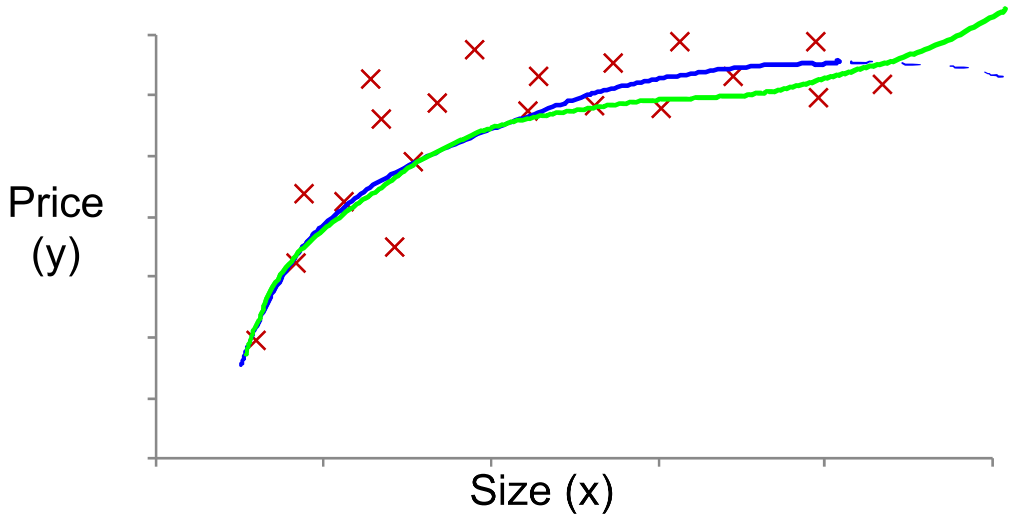
$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

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

$\ldots$

Modified from Andrew Ng ML

多项式回归 Polynomial regression



$$\theta_0 + \theta_1 x + \theta_2 x^2$$

$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned}$$

$$\begin{aligned} x_1 &= (size) \\ x_2 &= (size)^2 \\ x_3 &= (size)^3 \end{aligned}$$

Modified from Andrew Ng ML

示例：房价预测

·CRIM：城镇人均犯罪率。

·ZN：城镇超过25，000平方英尺的住宅区域的占地比例。

·INDUS：城镇非零售用地占地比例。

·CHAS：是否靠近河边，1为靠近，0为远离。

·NOX：一氧化氮浓度。

·RM：每套房产的平均房间个数。

·AGE：在1940年之前就盖好，且业主自住的房子的比例。

·DIS：与波士顿市中心的距离。

·RAD：周边高速公道的便利性指数。

·TAX：每10，000美元的财产税率。

·PTRATIO：小学老师的比例。

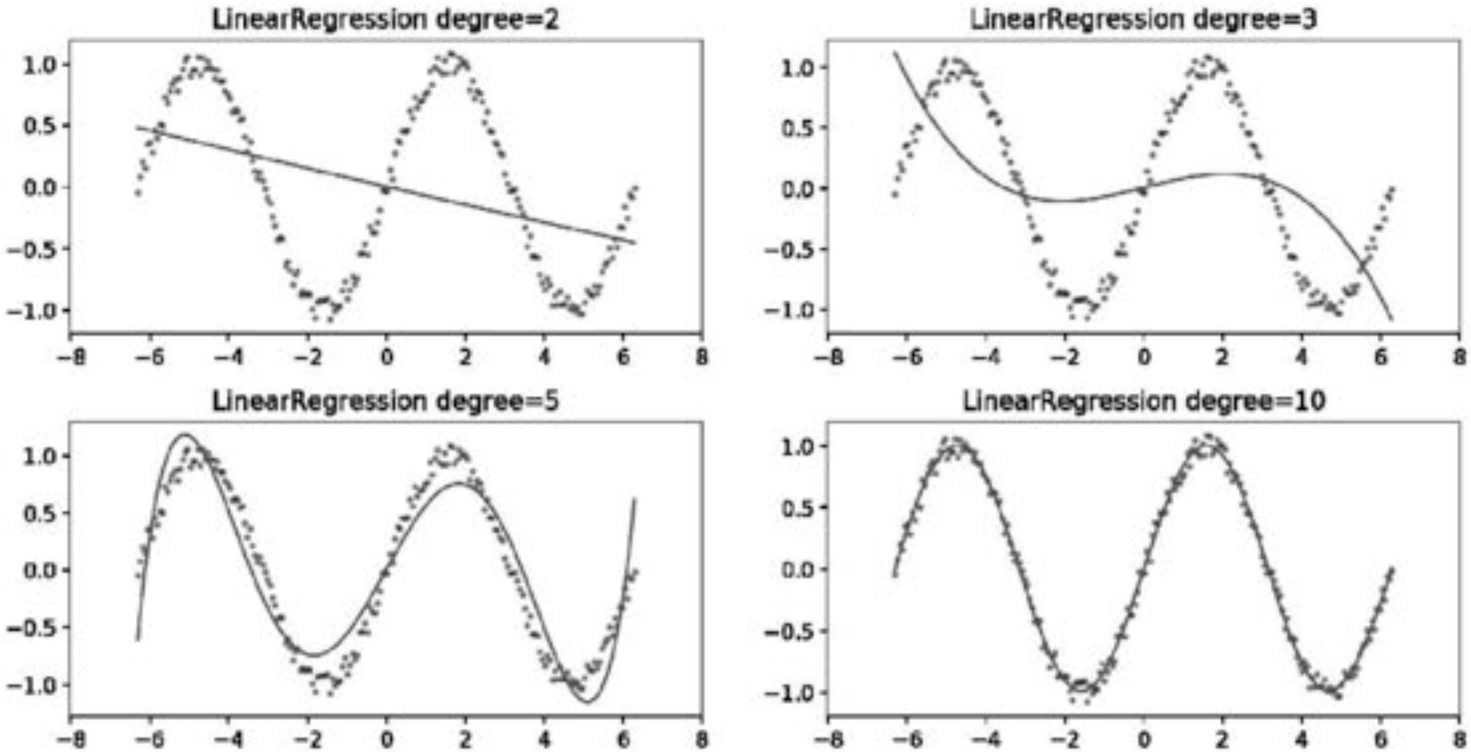
·B：城镇黑人的比例。

·LSTAT：地位较低的人口比例。

（506，13）

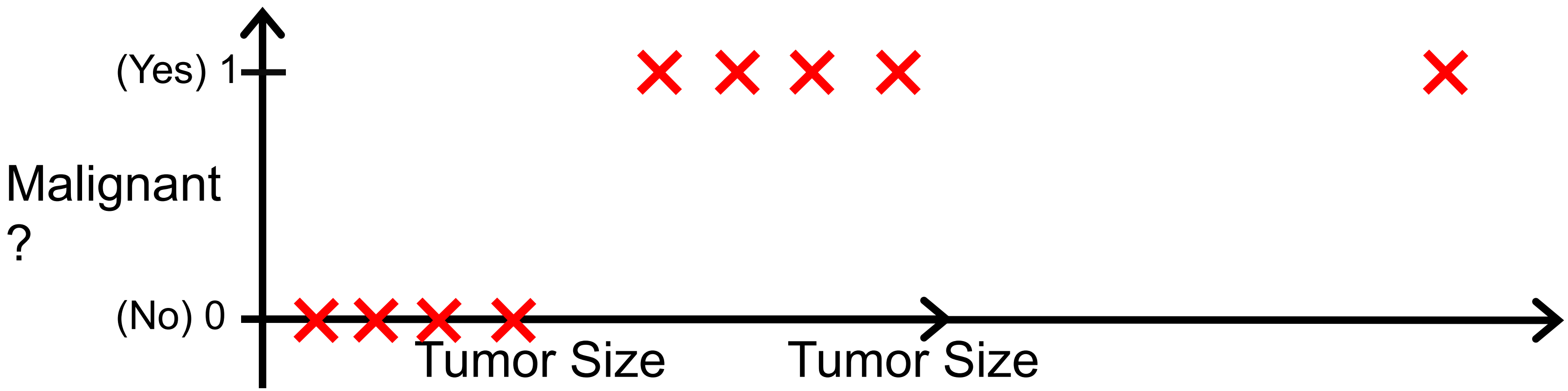
```
from sklearn.datasets import load_boston
boston = load_boston()
X = boston.data
y = boston.target
X.shape
```

示例：多项式拟合正弦函数



Logistic Regression 分类器

从回归到分类



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

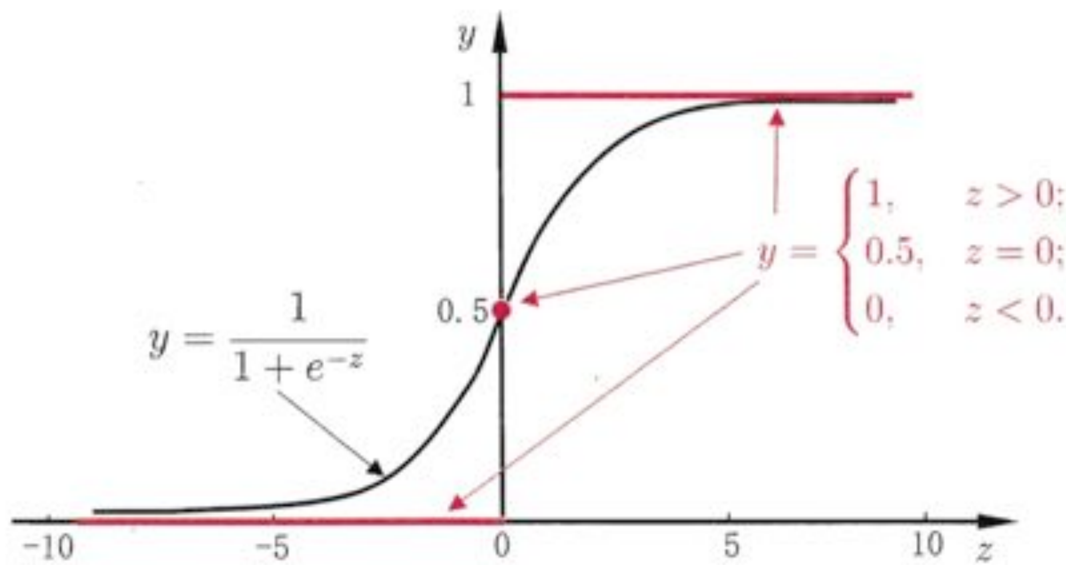
If $h_\theta(x) < 0.5$, predict "y = 0"

Modified from Andrew Ng ML

对数几率回归模型 Logistic Regression Model

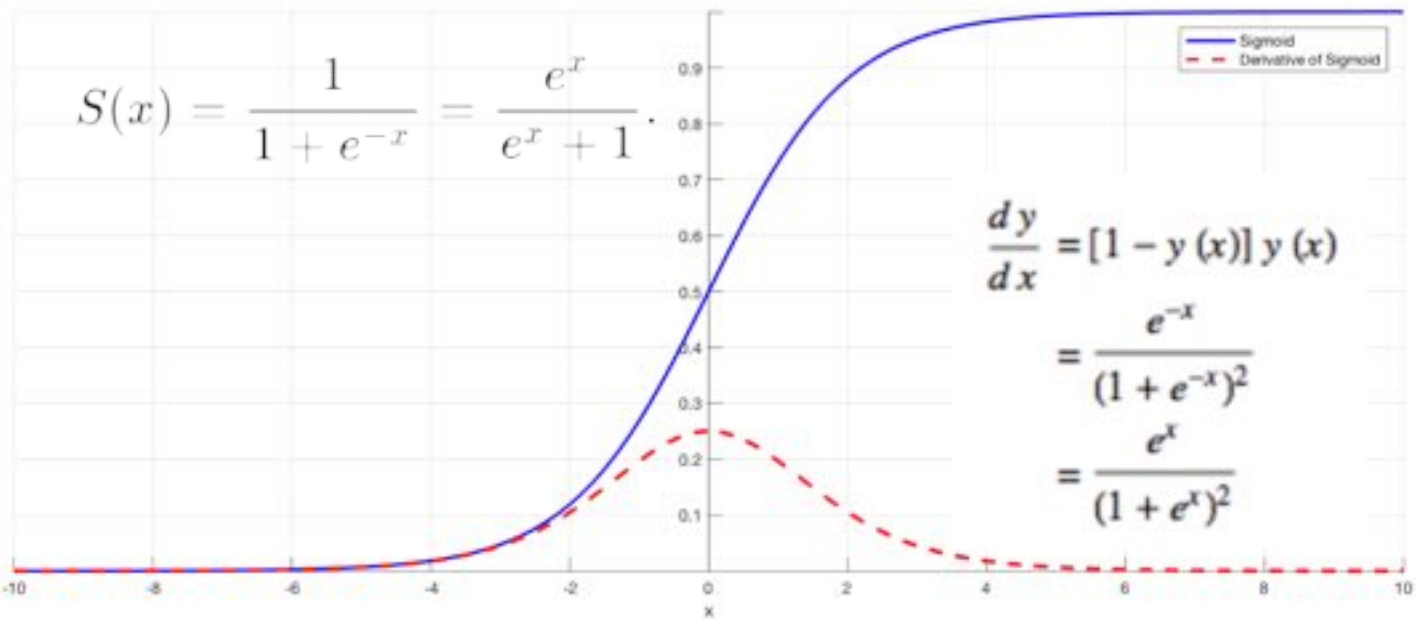
$$h_\theta(x) = \theta^T x \quad \Rightarrow \quad h_{\boldsymbol{\theta}}(x) = g(z) = g(\boldsymbol{\theta}^{\mathrm{T}} x) = \frac{1}{1+\mathrm{e}^{-\boldsymbol{\theta}^{\mathrm{T}} x}}$$

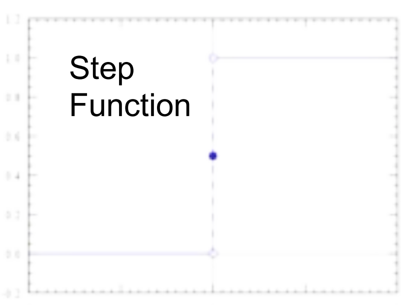
Sigmoid function
Logistic function

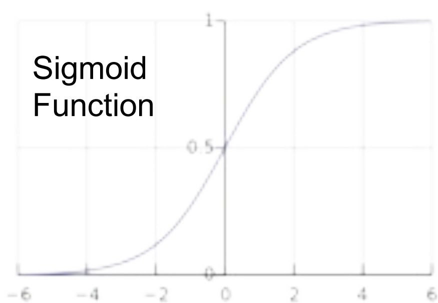


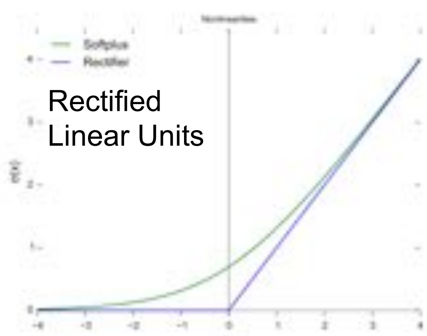
Modified from 周志华

Sigmoid function and beyond









https://github.com/Kulbear/deep-learning-nano-foundation/wiki/ReLU-and-Softmax-Activation-Functions

模型输出的含义
Interpretation of Hypothesis Output

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

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

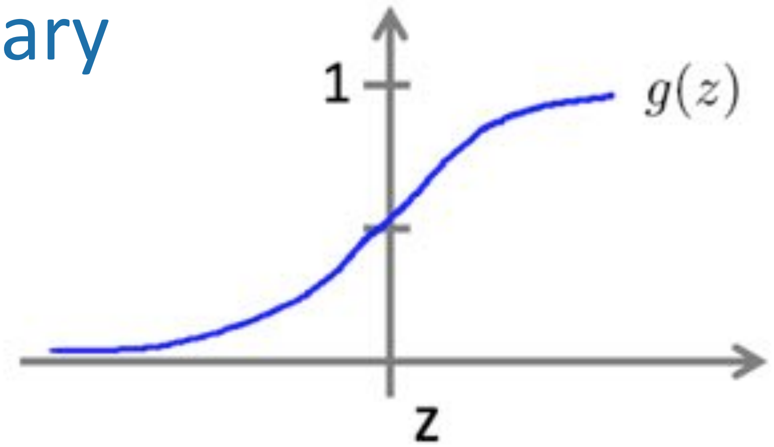
$h_\theta(x) = 0.7$ 70% 概率是恶性肿瘤

Probability that y = 1, given x, parameterized by θ

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

Modified from Andrew Ng ML

决策边界 Decision boundary



Logistic regression

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

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

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

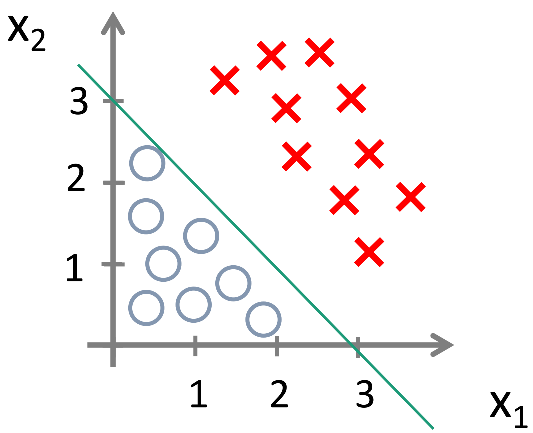
$$\theta^T X \geq 0$$

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

$$\theta^T X < 0$$

Modified from Andrew Ng ML

决策边界 Decision boundary

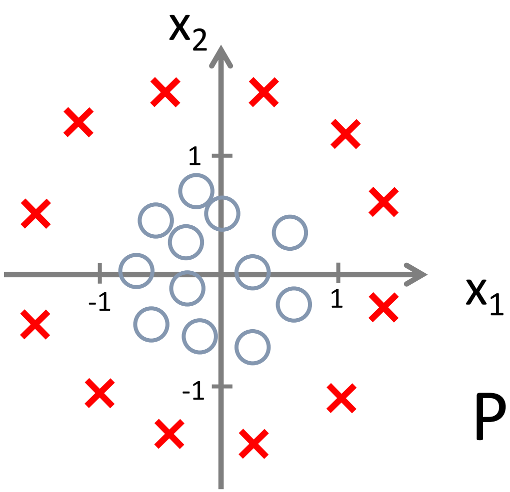


$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

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

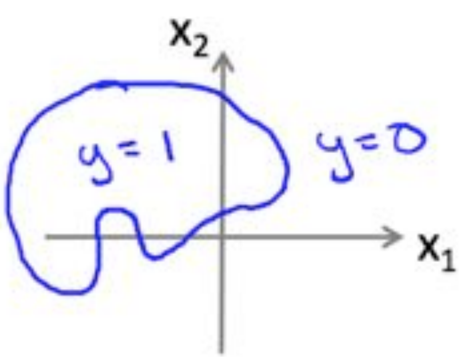
Modified from Andrew Ng ML

非线性决策边界
Non-linear decision boundaries



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

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



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

Modified from Andrew Ng ML

优化问题 Optimization

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

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

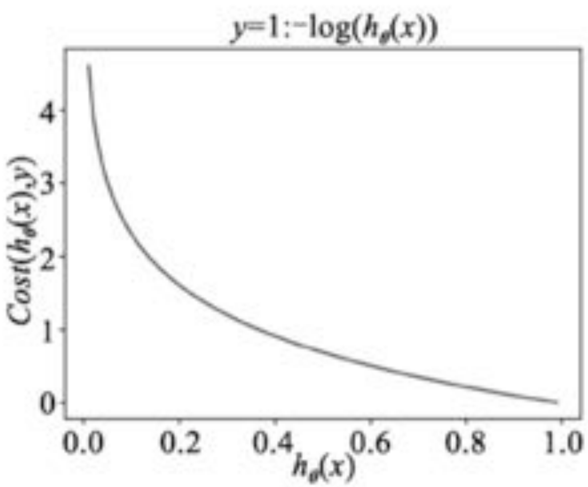
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

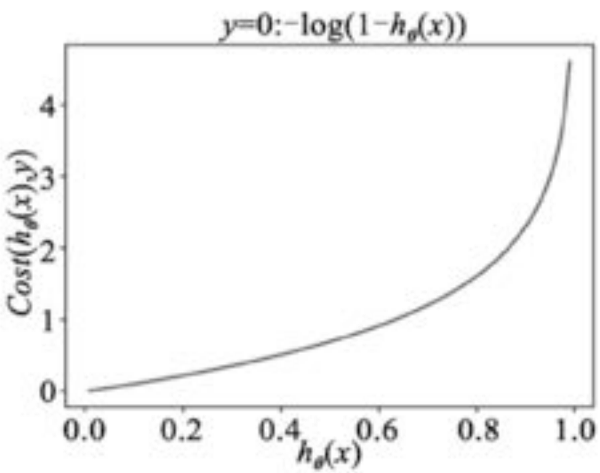
How to choose parameters θ ?

Modified from Andrew Ng ML

代价函数 Cost Function 😆

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





当y=1时，随着$\boldsymbol{h_\theta}$ (x) 的值（预测为1的概率）越来越**大**，预测值越来越接近真实值，其成本越来越小。

当y=0时，随着$\boldsymbol{h_\theta}$ (x) 的值（预测为0的概率）越来越**大**，预测值越来越偏离真实值，其成本越来越大。

Modified from Andrew Ng ML

代价函数 Cost Function

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

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

由于是离散值，当y=1时，1-y=0，上式的后半部分为0；当y=0时，上式的前半部分为0。因此下式与分开表达的成本计算公式等价。

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

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

Modified from Andrew Ng ML

寻优算法：梯度下降法 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 $\{$

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

(simultaneously update all θ_j)

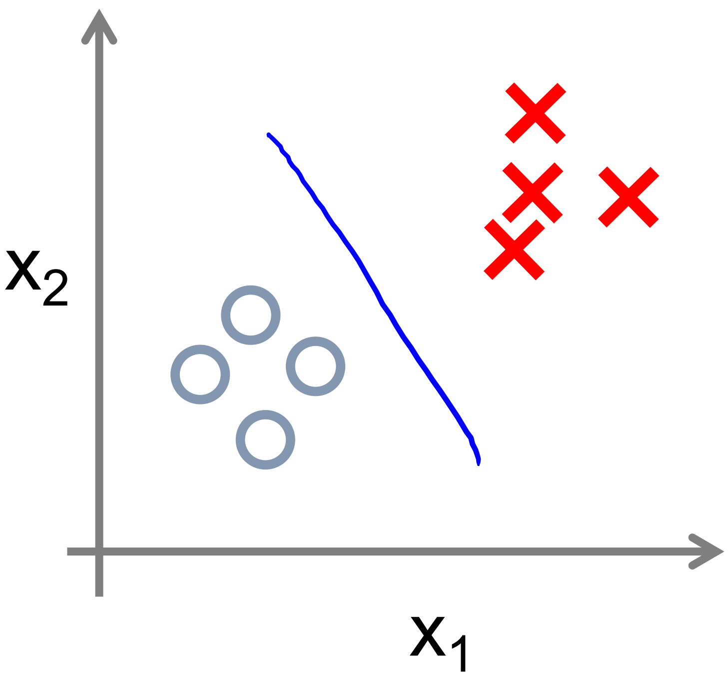
$\}$

Algorithm looks identical to linear regression!

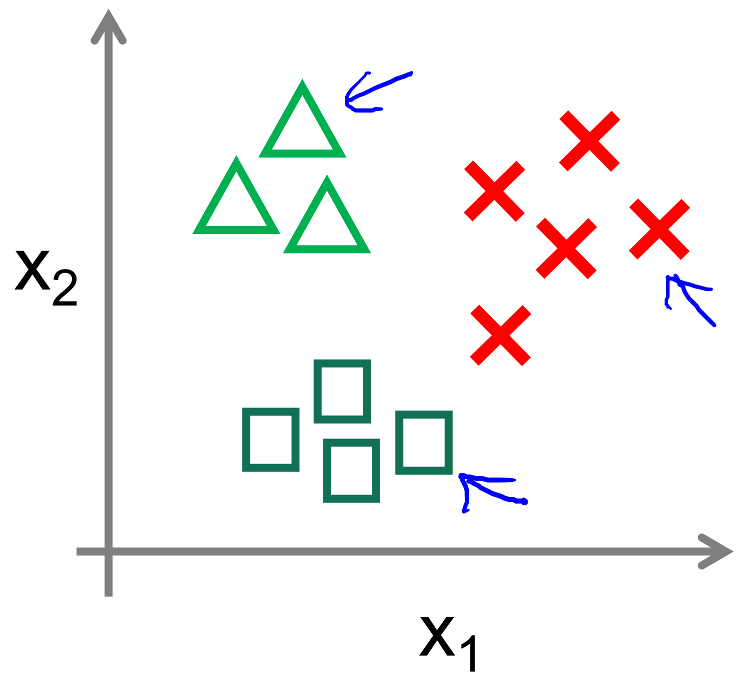
Modified from Andrew Ng ML

多分类问题 Multi-class classification

Binary classification:

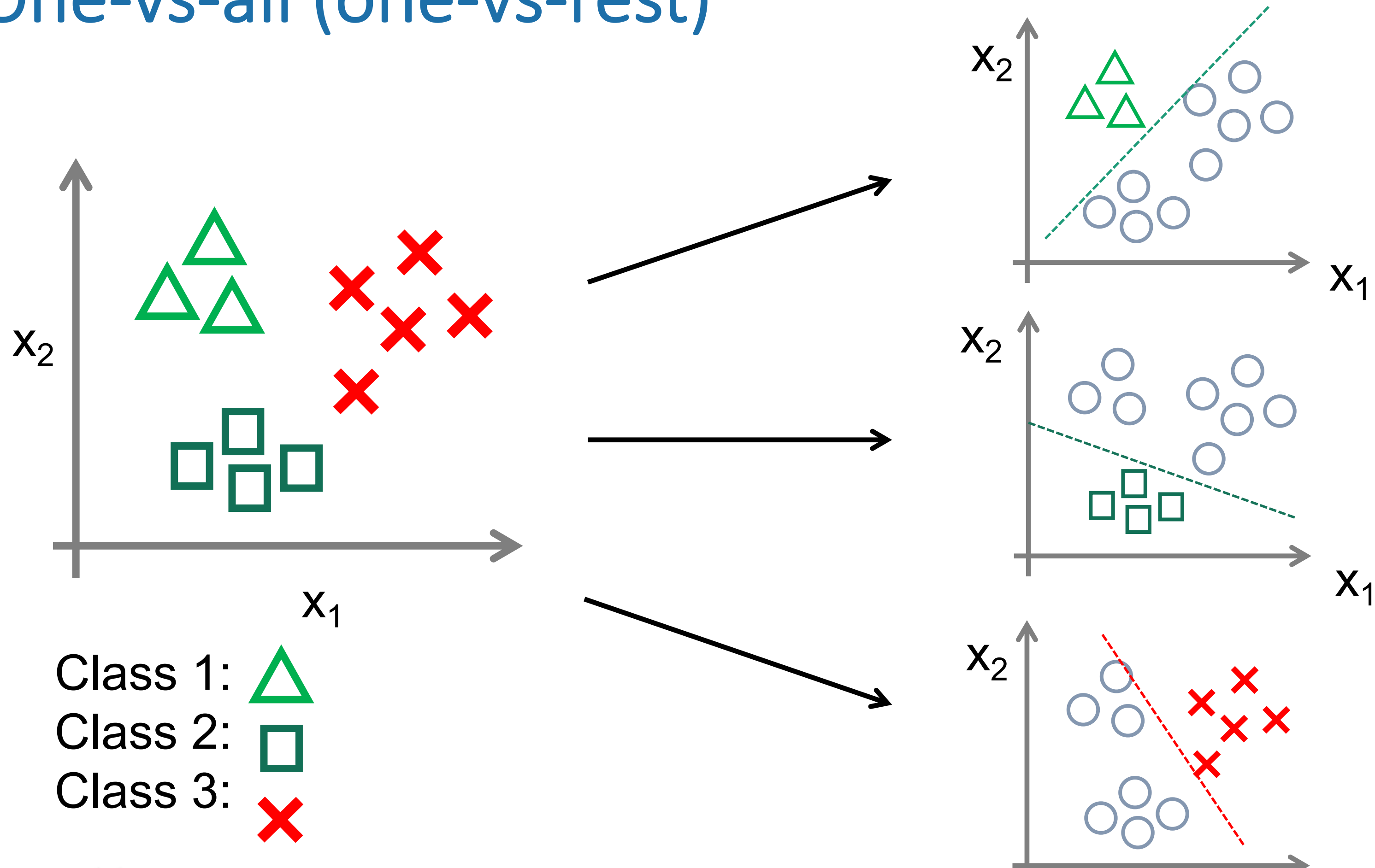


Multi-class classification:



Modified from Andrew Ng ML

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



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

$$\boxed{\max_i h_\theta^{(i)}(x)}$$

Modified from Andrew Ng ML

示例：乳腺癌检测

·radius：半径，即病灶中心点离边界的平均距离。

·texture：纹理，灰度值的标准偏差。

·perimeter：周长，即病灶的大小。

·area：面积，也是反映病灶大小的一个指标。

·smoothness：平滑度，即半径的变化幅度。

·compactness：密实度，周长的平方除以面积的商，再减1，即 $\frac{perimeter^2}{area} - 1$。

·concavity：凹度，凹陷部分轮廓的严重程度。

·concave points：凹点，凹陷轮廓的数量。

·symmetry：对称性。

·fractal dimension：分形维度。

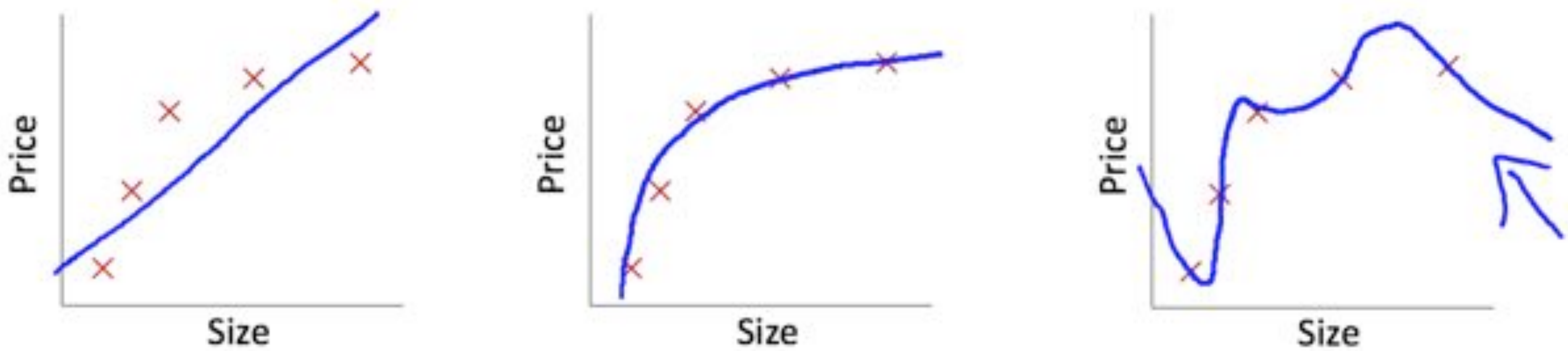
```
# 载入数据
from sklearn.datasets import load_breast_cancer

cancer = load_breast_cancer()
X = cancer.data
y = cancer.target
print('data shape: {0}; no. positive: {1}; no. negative: {2}'.format(
    X.shape, y[y==1].shape[0], y[y==0].shape[0]))
print(cancer.data[0])
```

上述代码输出结果如下：

```
data shape: (569, 30); no. positive: 357; no. negative: 212
[  1.79900000e+01   1.03800000e+01   1.22800000e+02   1.00100000e+03
   1.18400000e-01   2.77600000e-01   3.00100000e-01   1.47100000e-01
   2.41900000e-01   7.87100000e-02   1.09500000e+00   9.05300000e-01
   8.58900000e+00   1.53400000e+02   6.39900000e-03   4.90400000e-02
   5.37300000e-02   1.58700000e-02   3.00300000e-02   6.19300000e-03
   2.53800000e+01   1.73300000e+01   1.84600000e+02   2.01900000e+03
   1.62200000e-01   6.65600000e-01   7.11900000e-01   2.65400000e-01
   4.60100000e-01   1.18900000e-01]
```

Overfitting and Regularization
过拟合与正则化

Example: Linear regression (housing prices)

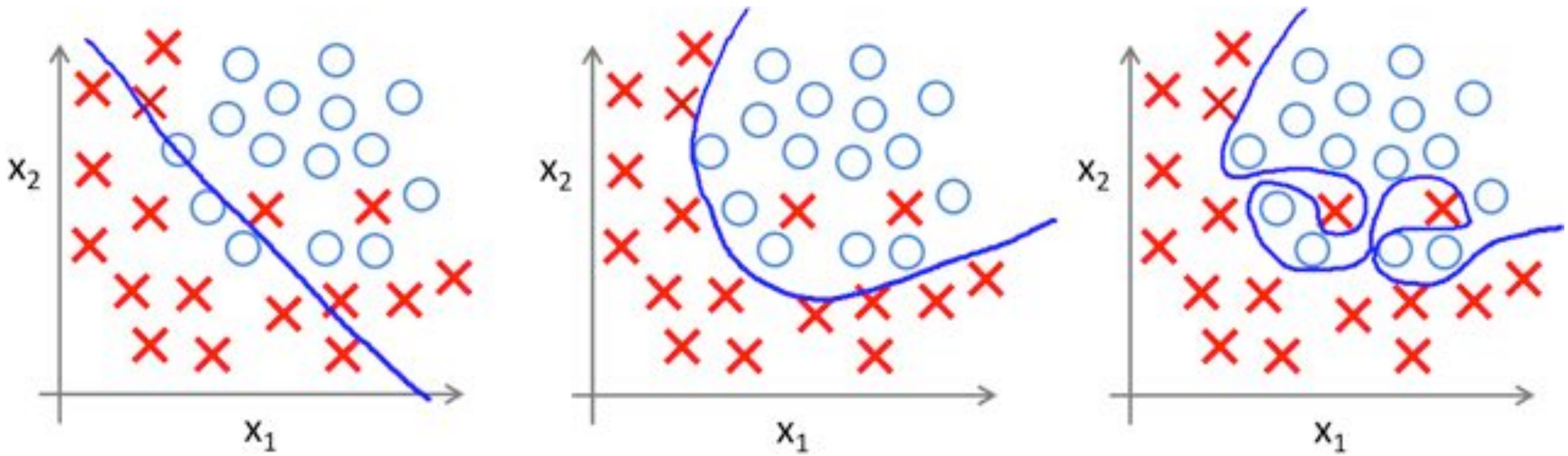


$\theta_0 + \theta_1 x$ $\qquad$ $\theta_0 + \theta_1 x + \theta_2 x^2$ $\qquad$ $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Modified from Andrew Ng ML

Example: Logistic regression



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

(g = sigmoid function)

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$$

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \ldots)$$

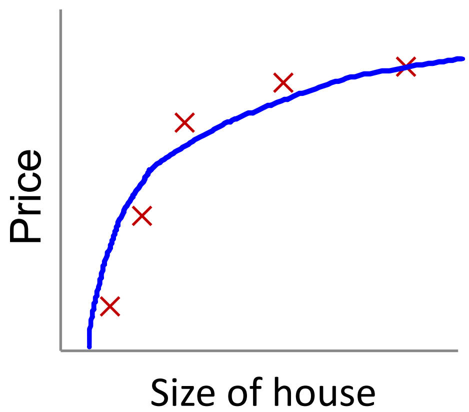
Modified from Andrew Ng ML

如何解决过拟合问题？

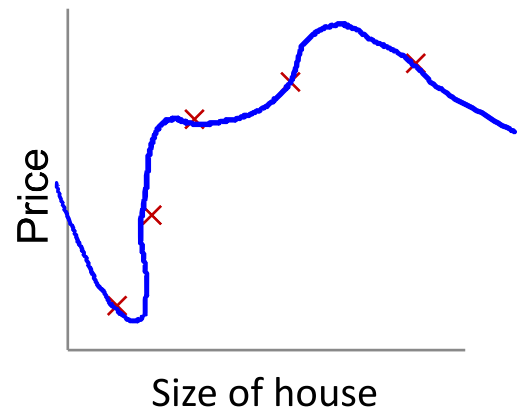
1. Reduce number of features
   - Manually select which features to keep.
   - Model selection algorithm (later in course).
2. Regularization 正则化
   - Keep all the features, but reduce magnitude/values of parameters θ_j.
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.

Modified from Andrew Ng ML

正则化 Regularization



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Suppose we penalize and make θ_3, θ_4 really small.

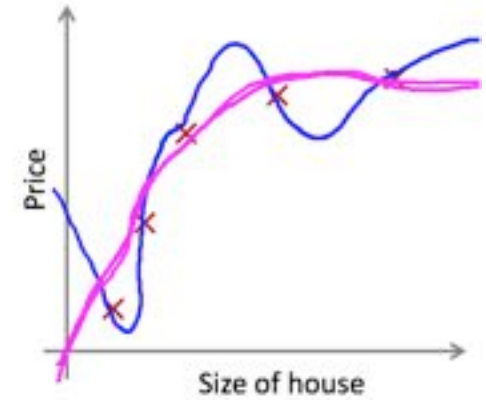
$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Modified from Andrew Ng ML

正则化 Regularization

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

– "Simpler" hypothesis
– Less prone to overfitting



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

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

正则项

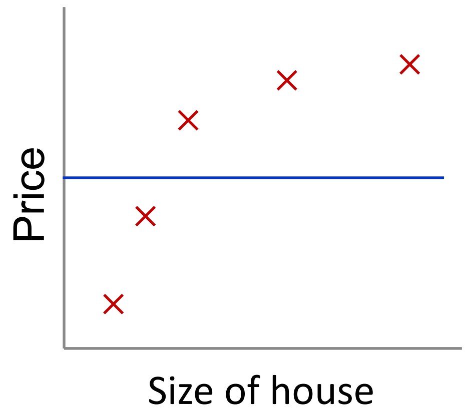
Modified from Andrew Ng ML

过度正则化导致欠拟合

In regularized linear regression, we choose θ to minimize

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

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Modified from Andrew Ng ML

线性回归的正则化梯度下降算法

Linear regression - Gradient descent with regularization

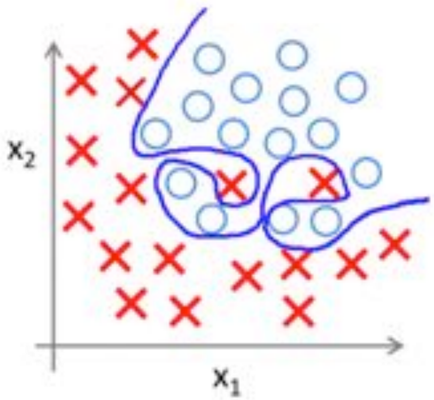
代价函数 $J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$

梯度 $\frac{\partial J(\theta)}{\partial \theta_j}$

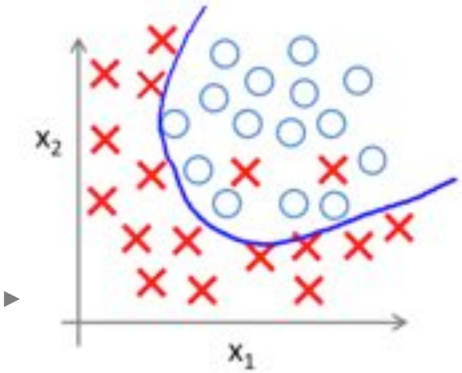
$$\theta_j = \theta_j - \alpha \boxed{\frac{1}{m}\sum_{i=1}^{m}\left[\left(\left(h(x^{(i)}) - y^{(i)}\right)x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j\right]}$$

$$\theta_j = \theta_j\underline{(1 - \alpha\frac{\lambda}{m})} - \alpha\frac{1}{m}\sum_{i=1}^{m}\left(\left(h(x^{(i)}) - y^{(i)}\right)x_j^{(i)}\right)$$

对数几率回归的正则化
Logistic regression with regularization



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



代价函数

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

正则化

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

Modified from Andrew Ng ML

对数几率回归的正则化梯度下降算法

代价函数

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

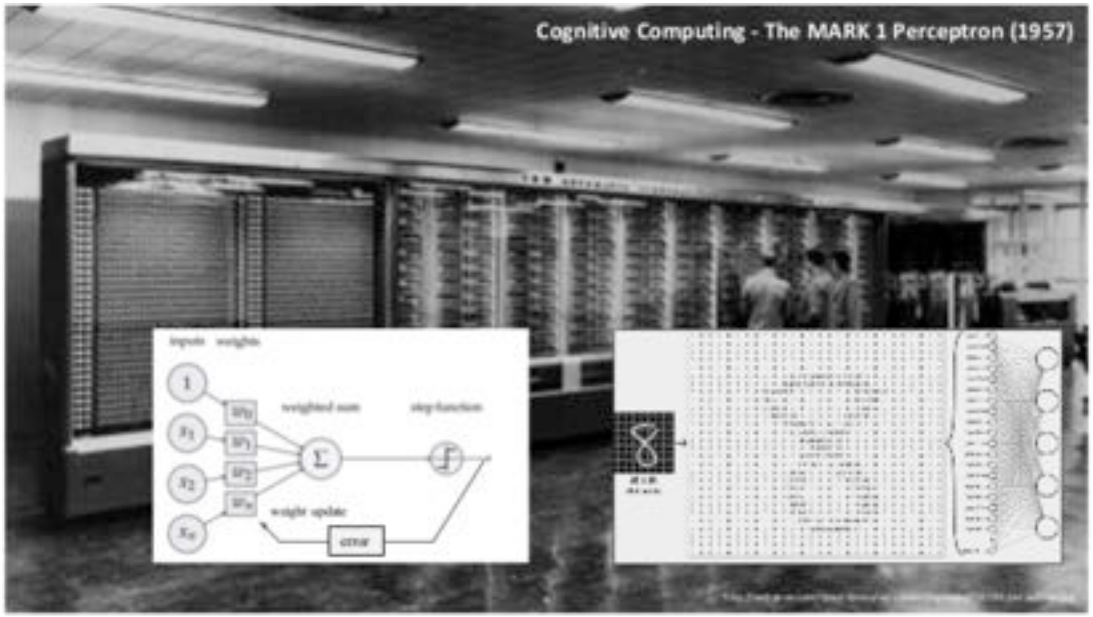
梯度 $\frac{\partial J(\theta)}{\partial \theta_j}$

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

$$\theta_j=\theta_j\underline{(1-\alpha\frac{\lambda}{m})}-\alpha\frac{1}{m}\sum_{i=1}^{m}\left(\left(h(x^{(i)})-y^{(i)}\right)x_j^{(i)}\right)$$

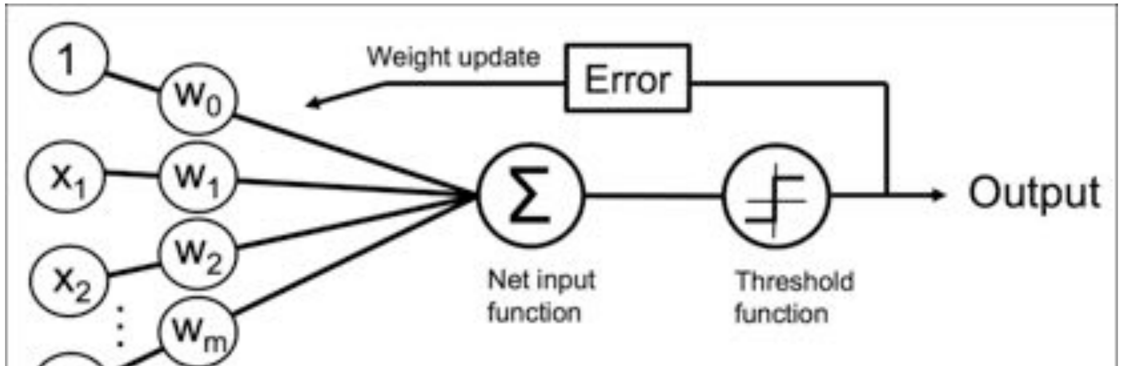
感知器Perceptron

感知机 Perceptron



Frank Rosenblatt, 1957

感知机 Perceptron



$$y = f(\sum_{i=1}^{m} \boldsymbol{w}_i \cdot \boldsymbol{x}_i) = f(\boldsymbol{W}^T \boldsymbol{X})$$

Frank Rosenblatt, 1957

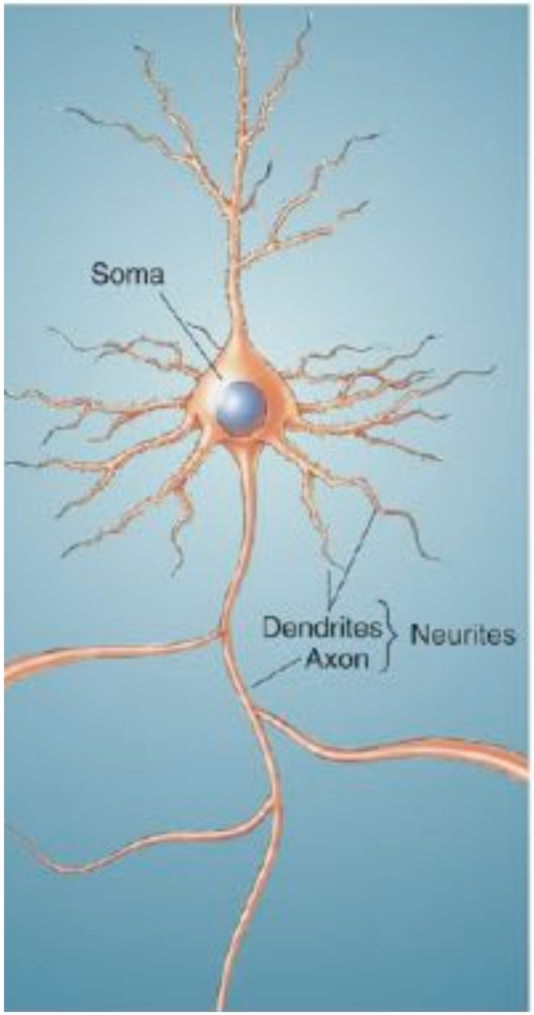
- Rosenblatt (1958) introduced the perceptron, the simplest form of neural network
- The perceptron is a single neuron with adjustable synaptic weights and a threshold activation function
- Rosenblatt also developed an error-correction rule to adapt these weights (the perceptron learning rule)
- He also proved that if the (two) classes were linearly separable, the algorithm would converge to a solution (the perceptron convergence theorem)

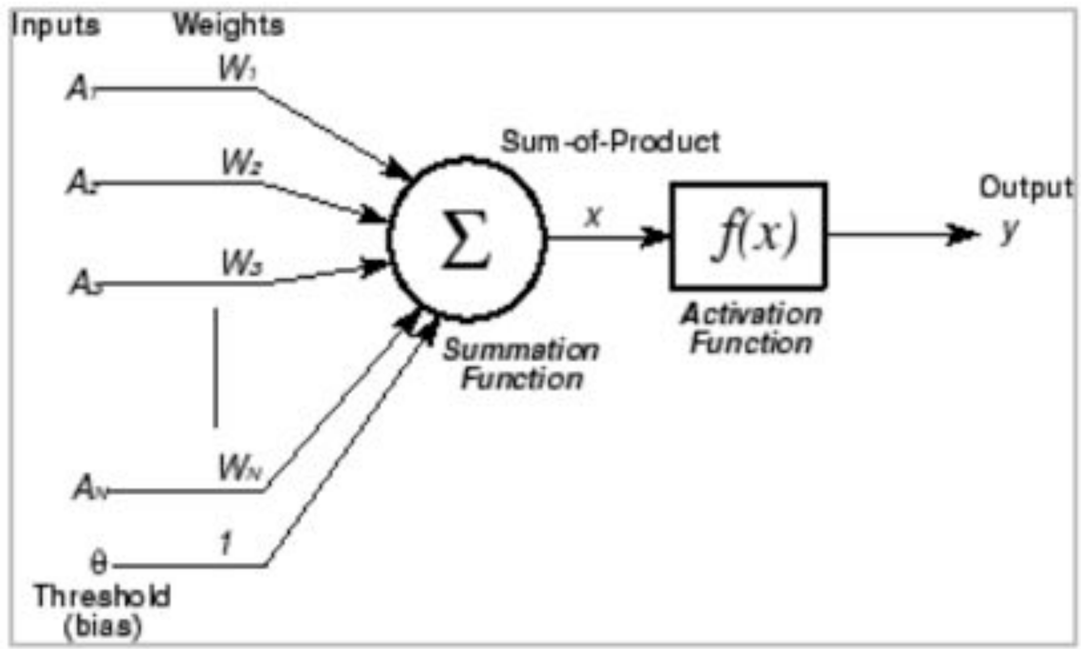
感知机 Perceptron

The perceptron, invented in the late 1950s, was considered a **paradigm shift**. For the first time, a machine could be taught to perform certain tasks using examples. This surprising invention was almost immediately followed by an equally surprising theoretical result, the perceptron convergence theorem, which states that a machine executing the perceptron algorithm can effectively produce a decision rule that is compatible with its training examples. A youthful wave of optimism took over the research community. Although it only dealt with a very specific category of tasks, namely pattern recognition tasks, the perceptron was widely publicized as the forerunner of more general learning machines.

Léon Bottou, 1988

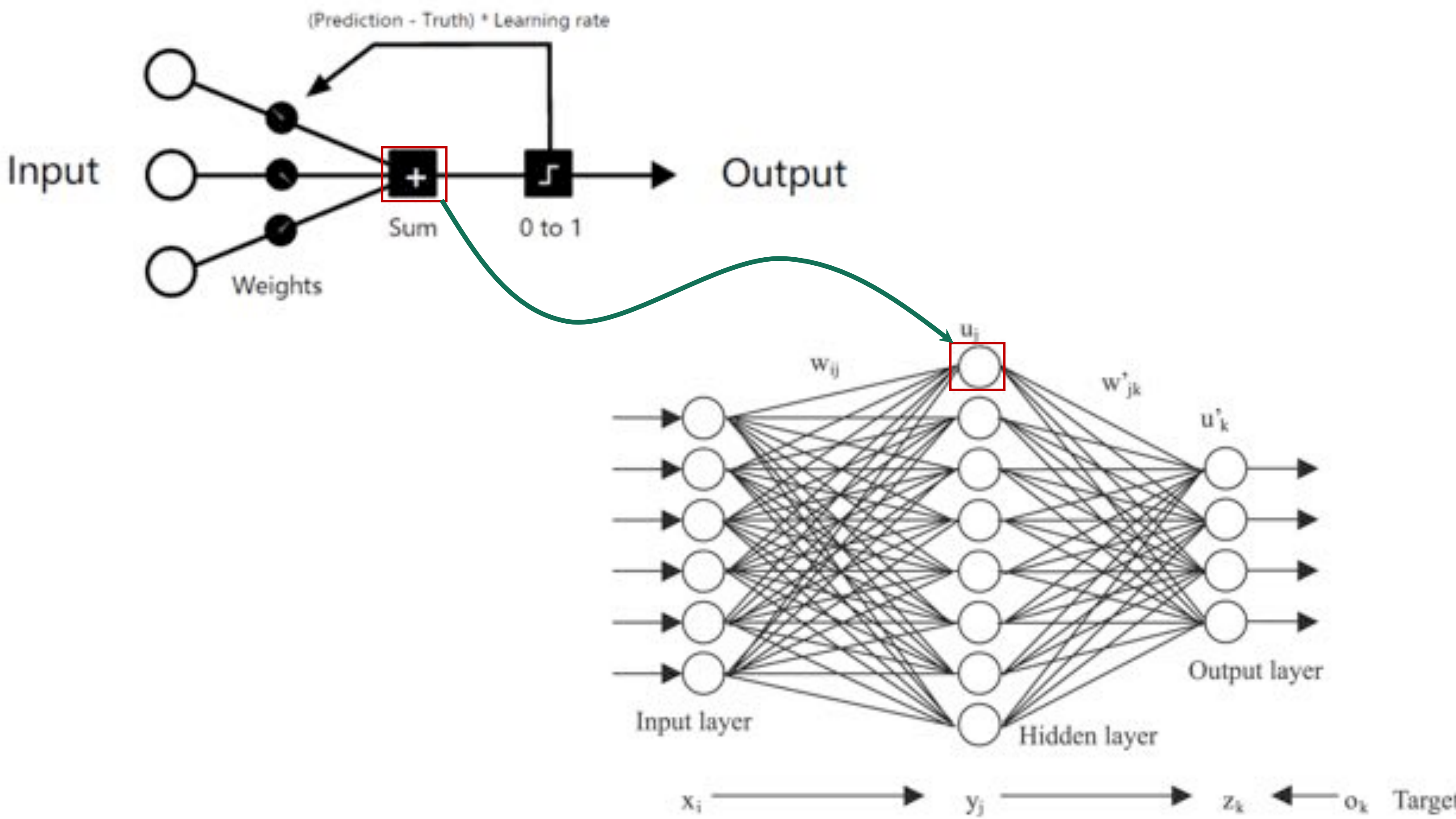
神经细胞的MP模型





McCulloch–Pitts 1943

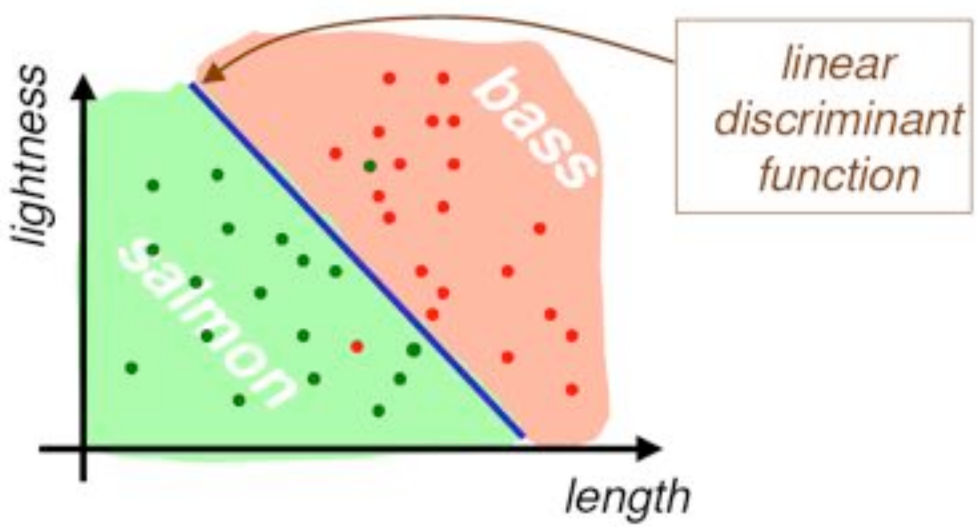
从感知机到神经网络


(Prediction - Truth) * Learning rate
Input
Output
Sum
0 to 1
Weights
u_j
w_{ij}
w'_{jk}
u'_k
Input layer
Hidden layer
Output layer
x_i
y_j
z_k
o_k
Target

线性分类器 Linear Classifier

- No probability distribution (no shape or parameters are known)
- Only labeled data available

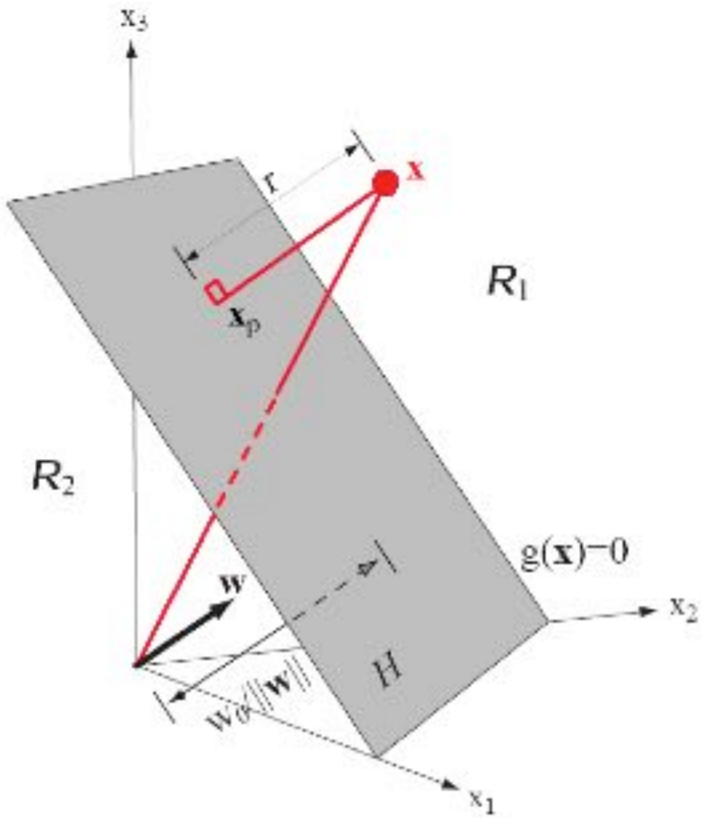
- The shape of discriminant functions is assumed to be linear



$$g(x) = w^T x + w_0 \Leftrightarrow \begin{cases} g(x) > 0 & x \in \omega_1 \\ g(x) < 0 & x \in \omega_2 \end{cases}$$

- **Need to estimate parameters of the linear function**

分类超平面 Hyperplane



$$g(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0$$

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$$

$$g(X) = W^{\mathrm{T}}(X_p + r\frac{W}{\|W\|}) + w_0 = W^{\mathrm{T}}X_p + w_0 + r\frac{W^{\mathrm{T}}W}{\|W\|} = r\|W\|$$

$$r = \frac{g(\mathbf{x})}{\|\mathbf{w}\|}$$

- The orientation of the surface is determined by the vector **w**
- The location of the surface is determined by the bias w_0
- **The discriminant function *g*(x) is proportional to the signed distance from x to the hyperplane**

线性判别函数的增广形式

$$g(x) = w^T x + w_0 \Leftrightarrow \begin{cases} g(x) > 0 & x \in \omega_1 \\ g(x) < 0 & x \in \omega_2 \end{cases}$$

$$g(\mathbf{x}) = w_0 + \sum_{i=1}^{d} w_i x_i = \sum_{i=0}^{d} w_i x_i$$

$$\mathbf{a} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix} = \begin{bmatrix} w_0 \\ \\ \mathbf{w} \\ \\ \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{bmatrix} = \begin{bmatrix} 1 \\ \\ \mathbf{x} \\ \\ \end{bmatrix}$$

Augmented feature vector

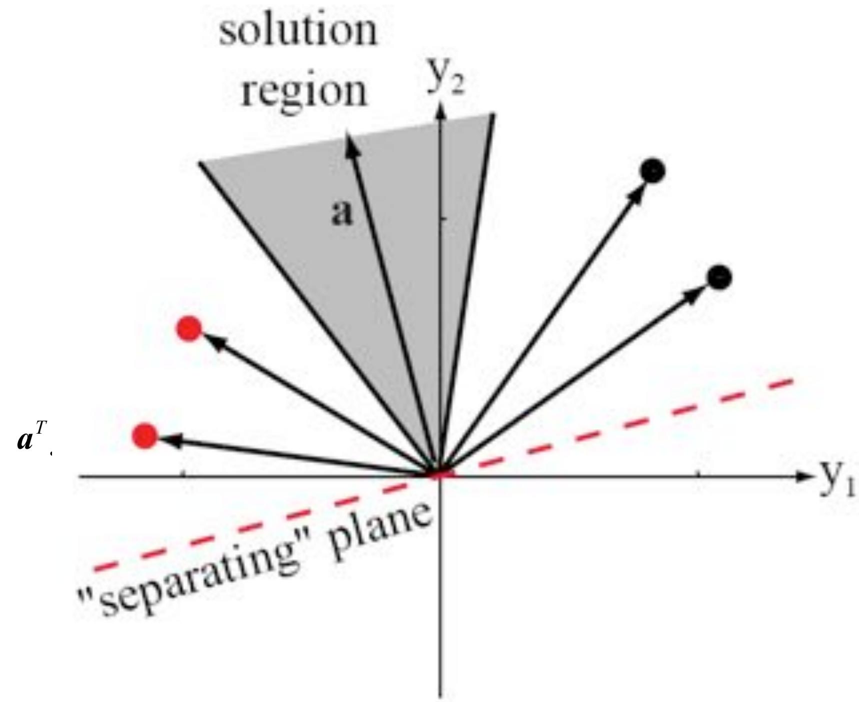
$$g(x) = a^T y \begin{cases} > 0 & x \in \omega_1 \\ < 0 & x \in \omega_2 \end{cases}$$

样本规范化 Normalization

$$g(x) = a^T y \begin{cases} > 0 & x \in \omega_1 \\ < 0 & x \in \omega_2 \end{cases}$$

$$y \leftarrow [-y] \quad \forall y \in \omega_2$$

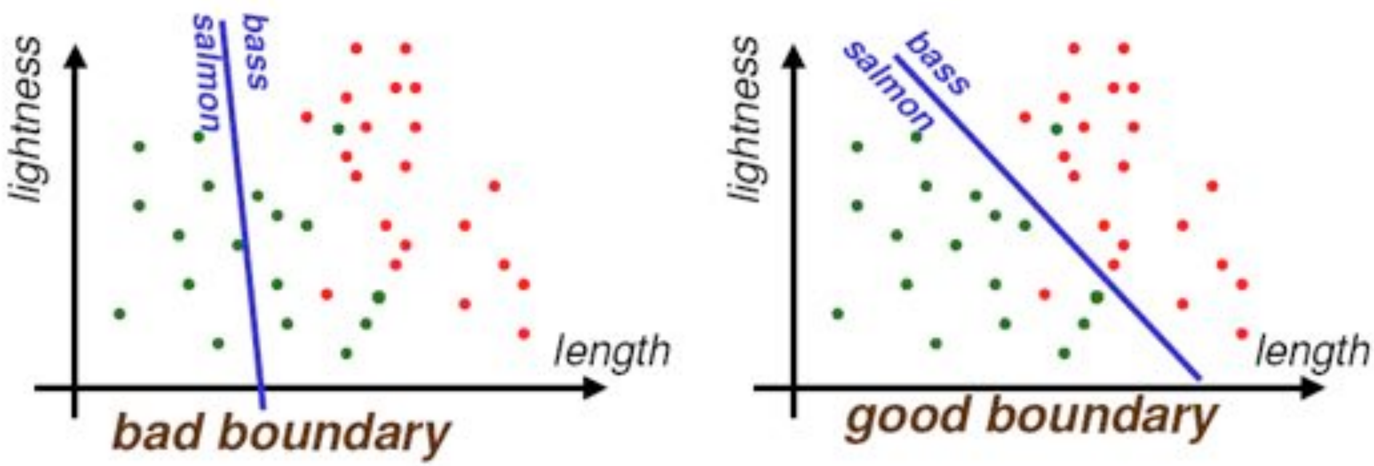
$$a^T y > 0 \quad \forall y$$



- 向量*a* 的取值范围如何确定？
- 怎样的*a*是最优的？

准则函数 Criterion Function

- **Which decision boundary is better（怎样的a是最优的）？**
- **The task is to find the best parameter of the line to separate the two classes with least error.**

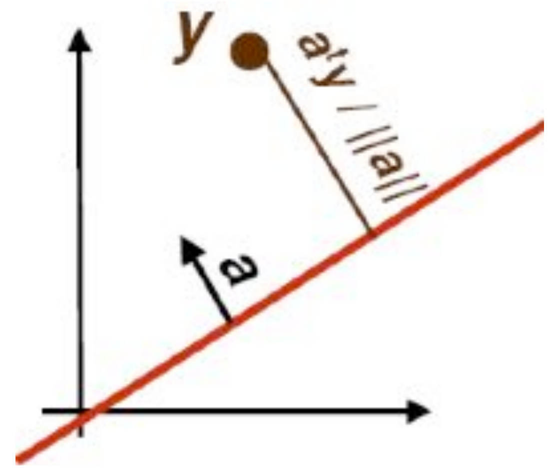


- **Need criterion function J(a) which is minimized when a is a solution vector**

感知器准则 Perceptron Rule

- 更好的选择: ***Perceptron*** criterion function

$$J_p(a) = \sum_{y \in Y_M} (-a^t y)$$



其中Y_M是错分样本集

$$Y_M(a) = \{ sample \ \ y_i \ \ s.t. \ \ a^t y_i < 0 \}$$

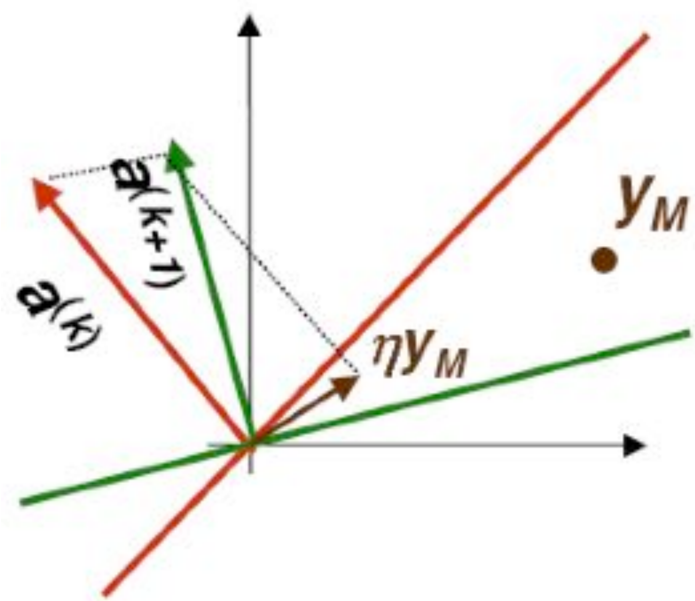
- $J_p(a)$ 是错分样本到决策面的距离之和的 $||a||$倍？
- $J_p(a)$是分段线性的，所以可用梯度下降法求解最小值

感知器准则下的梯度下降法

优化目标是最小化<u>错分样本</u>的负判决值之和

$$J_P(\mathbf{a}) = \sum_{y \in Y_M} \left(-\mathbf{a}^T y\right)$$

$$\nabla_a J_P(\mathbf{a}) = \sum_{y \in Y_M} (-y)$$



$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta \sum_{y \in Y_M(k)} y$$

Batch Update 成批更新

$$\mathbf{a}(k+1) = \mathbf{a}(k) + \eta \cdot y^k$$

Single-sample Update

单样本逐个更新

感知器准则算法伪代码

Batch Update

begin initialize $\mathbf{a}, \eta(\cdot)$, criterion $\theta, k = 0$

  do $k \leftarrow k + 1$

    $\mathbf{a} \leftarrow \mathbf{a} + \eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y}$

  until $\eta(k) \sum_{\mathbf{y} \in \mathcal{Y}_k} \mathbf{y} < \theta$

 return $\mathbf{a}$

end

Single-sample Update

begin initialize $\mathbf{a}, k = 0$

  do $k \leftarrow (k + 1) \mathrm{mod} n$

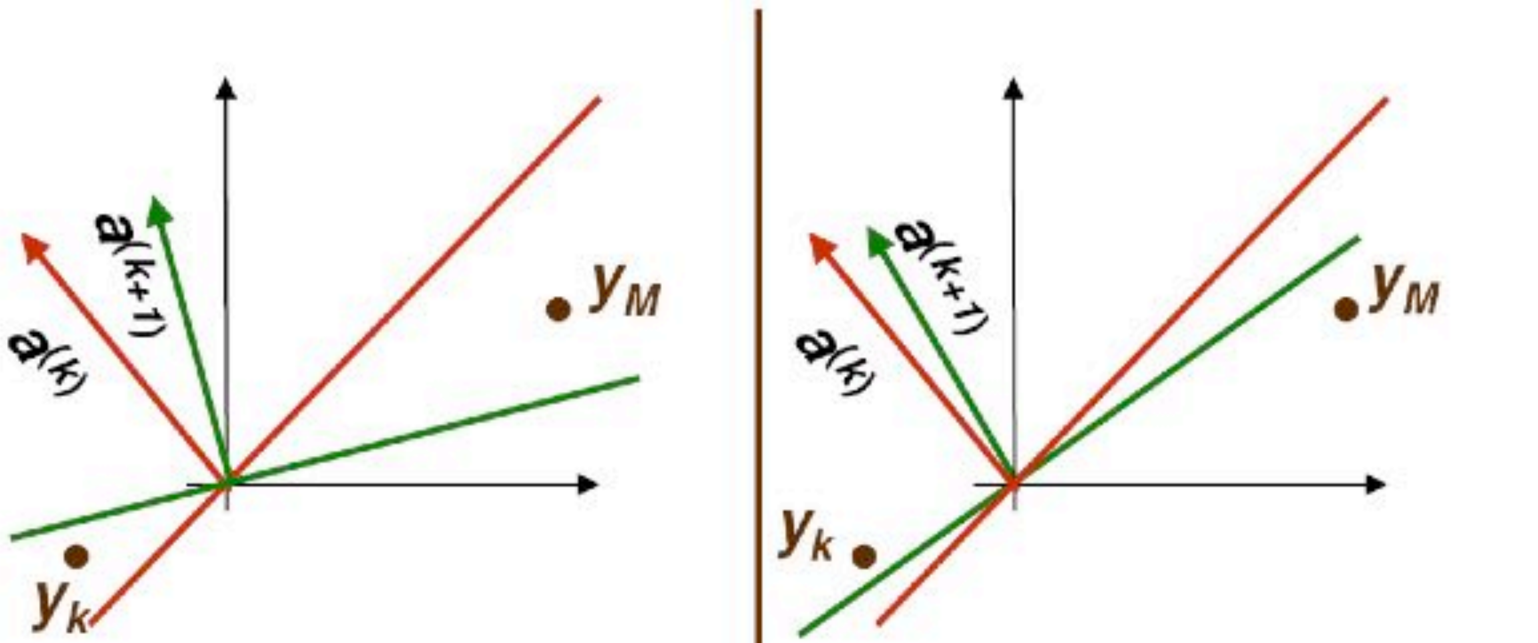
    if $\mathbf{y}_k$ is misclassified by $\mathbf{a}$ **then** $\mathbf{a} \leftarrow \mathbf{a} - \mathbf{y}_k$

  until all patterns properly classified

 return $\mathbf{a}$

end

Learning rate in single-sample update



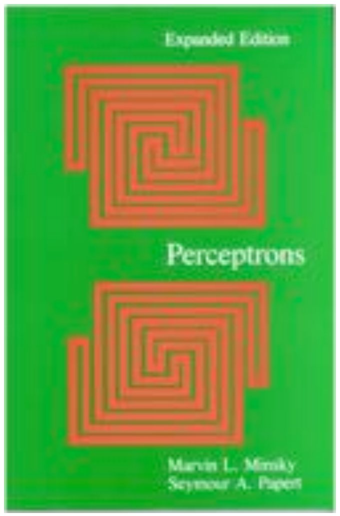
η is too large, previously correctly classified sample $\boldsymbol{y}_k$ is now misclassified

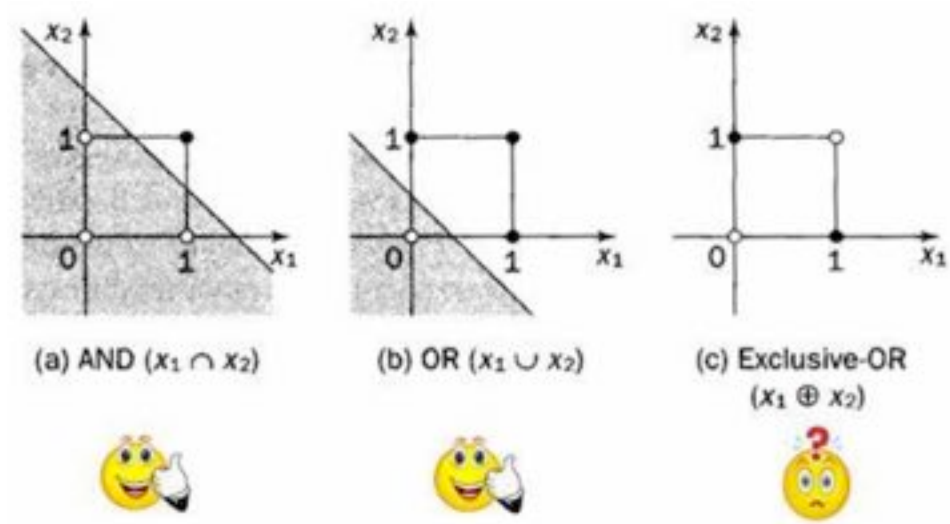
η is too small, $\boldsymbol{y}_M$ is still misclassified

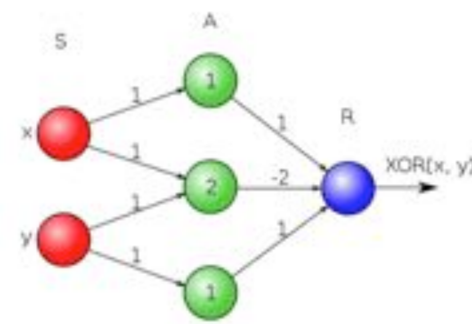
Perceptron Learning Algorithm: A Graphical Explanation Of Why It Works

https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

XOR problem killed Perceptron



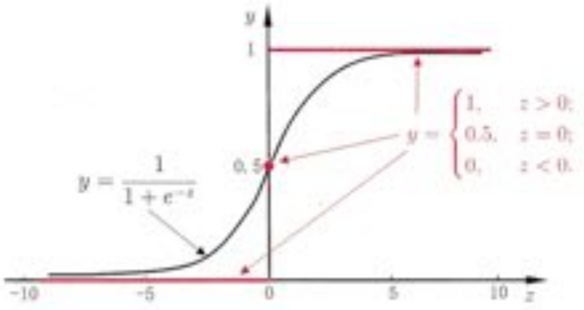




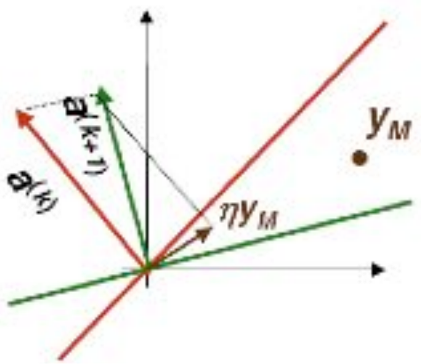
Minsky and Papert, 1969

总结：线性回归与线性分类

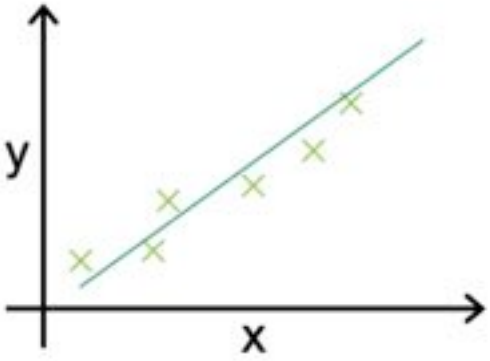
对数几率回归分类器
Logistic regression



感知机线性分类Perceptron



线性回归
Linear regression



机器学习的核心方法

- ☑ 模型选择 Model selection
- ☑ 代价函数 Cost function
- ☑ 优化算法 Optimization algorithm
- ☑ **正则化 Regularization**
- ☑ **核函数升维 Kernel trick**