Machine Learning note - Stanford

Auther: Xingwei Chen

`MachineLearning` `Octave`

Basic Defination

1. The definition of machine learning:
   - A computer program
   - Learn from experience ***E*** with respect to some class of tasks ***T*** and performance measure ***P***
     - Example(playing checkers):
       - E: The experience of playing many games of checkers.
       - T: The task of playing checkers.
       - P: The probability that the program will win the next game.
2. Superivsed & Unsuperivsed Learning
   - Supervised learning: relationship between the input and output.
     - Regression: continuous
     - Classification: discrete
   - Unsupervised learning: donot need to know the results.
     - Clustering

Octave Tutorial

1. Basic Operations

```
help(object)
```

```
% #to make a comment
1~=2 #不等于
1&&0 #AND
1||0 #OR, 另一种写法:xor(1,0)
PS1(">> ") #将起始标识改为>>
```

2. Variables

```
a=3; #不加分号的话会直接print 'a=3'
disp(a); #只输出a的值
disp(sprintf('2 decimals: %0.2f',a)) #sprintf:输出strings
>>> 3.14
format long/short #默认的保留小数位（长/短）
```

3. Vector & Matrices

```
V = [1, 2, 3] # 1*3
V = [1; 2; 3] # 3*1
V=1:0.1:1.3 #start:step:end,输出1 1.1 1.2 1.3
V=1:6 #1 2 3 4 5 6

M=[1 2;3 4;5 6] #3*2
ones(2,3) #[1 1 1;1 1 1]
C=2*ones(2,3) #C=[2 2 2;2 2 2]

w=zeros(1,3) #w=0 0 0
I=eye(4) #4-class identity matrix
flipud(eye(4)) #右上左下
```

4. Random, distribution and plot

```
w=rand(3,3) #3*3 随机数矩阵
w=randn(3,3) #3*3正态分布数矩阵
w=-6+sqrt(10)*(randn(1,10000)); #注意是每个元素都+(-6)
hist(w) #柱形图
```

5. size and length

```
size(M) #返回行列数
    size(M,1) #矩阵M行数
    size(M,2) #矩阵M列数
length(M) #the number of elements
```

6. Save & Load & clean

```
cd 'c:/xxx/xxx' #设置路径
pwd #path way directory
ls #folders in the pwd
load("filename.dat")
addpath('newpath') #添加新路径

who #给出当前features的名字，类似dir()
whos #给出所有features的size/scale/class,类似head()

clear feature #clear a feature
clear #清空当前变量，无论who/whos都返回NULL

v=xxx(1:10) #选取某个feature xxx的前10个元素组成新feature

save xxx.mat v; #将feature v 保存在xxx.mat
save yyy.txt v -ascii; #save v as a text
```

7. manipulate

```
A=[1 2;3 4;5 6];
B=[10 11;12 13;14 15];

A(3,2) #6
A(3,:) #第三行的所有列值:5 6
A([1 3],:) #第一行+第三行
A(:,2)=[7;8;9] #赋值
A=[A,[100;101;102]]#add a new column
[A;[0 0 0 0]]#add a new row
A(:) #put all in A into a single vector
C=[A B] #横向组合，得到3*4矩阵
D=[A;B] #纵向组合，得到6*2矩阵
```

8. Computing the data

```
A*B #矩阵乘法
A .* B #同位置元素相乘，要求A与B的行列数相同
A .^2 #A中每个元素都平方
1 ./ v #v中每个元素取倒数
log(v)/exp(v)/abs(v)/-v
v+ones(3,1) #等价于v+1
A' '#转置
pinv(A) #逆矩阵

val=max(A) #给出A中的最大值
max(max(A)); max(A(:)); #均为等价式
max(A); #返回最大值所在行
[v1,v2]=max(A) #v1为最大，v2为第二大
max(A,[],1) #以一行向量的形式返回A每列的最大值，参数为2则返回行最大值

find(A<3) #filter: 返回所有小于3的元素
A=magic(3) #返回3阶魔方矩阵
sum()/prod() #累加/阶乘
floor()/ceil() #小于某数的最大整数/大于某数的最小整数
sum(A,1) #每列的sum
```

- 得到A的对角矩阵并对元素求和

```
B=A .* eye(9); #与维度相等的单位矩阵同位置元素相乘
sum(sum(B));
```

9. Plotting

- Basic operation

```
plot(x,y); #y可以为表达式
axis([0 1 -1 1]) #xrange(0,1),yrange(-1,1)
xlabel('X') #同理ylabel('Y')
legend('y1','y2') #添加标注
title('myplot')
cd 'path'; print -dpng 'myplot.png' #另存为
close #close the figure
```

- Multiple figure operation

```
#在不同的图层分别作图
figure(1); plot(x,y1);
```

```
figure(2); plot(x,y2); #分别做两个图

#在同一张图中作两条线
plot(x,y1);
hold on;
plot(x,y2,'r'); #可指定颜色

#在同一个图层中作两个图
subplot(1,2,1);plot(x,y1); #将图层一分为二，图表只占左半边，右边空白
subplot(1,2,2);plot(x,y1); #右边可以继续添加plot(x,y2)
clf; #清空图层
```

- 其他操作

```
imagesc(A) #矩阵A的每个元素用不同颜色的方格表示
```

10. Control Statements

- for loop

```
v=zeros(10,1)
for i=1:10,
    v(i)=2^i;
end;
```

- while loop

```
i=1;
while i<= 5,
    v(i)=100;
    i=i+1;
end;
```

- if/elseif/else/break

```
i=1;
while true,
    v(i)=999;
    i=i+1;
    if i==6,
        break;
    end; #the end of if
```

```
end; #the end of while

if xxx,
    disp('xxx');
elseif yyy,
    disp('yyy');
else
    disp('zzz');
end;
```

11. Define the function

```
#定义函数
function y =funcname(x)
y= x^2;

#调用函数
cd 'the function's path'
funcname(5);
```

- Example:

```
function [y1,y2]=funcsquare(x)
y1=x^2;
y2=x^3;
[a,b]=funcsquare(5);
```

12. Example: Compute cost function

```
function J=costFunction(X,y,theta)
    #X:design matrix
    #y:class labels

    m=size(X,1); #number of training examples
    predictions=X*theta; # pred of hypothesis on all m examples
    sqrErrors=(predictions-y) .^ 2; #squared errors

    J=1/(2*m) * sum(sqrErrors);

>>> 25,125
```

Line Regression

线性回归是一种参数学习方法: 根据已有的数据X来确定参数θ, 最终得到一个形如

$$J(\theta) = \theta^T X$$
$$\theta^T = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \dots \quad \theta_n]$$
$$X = \begin{bmatrix} X_1^{(1)} & X_2^{(1)} & \dots & X_n^{(1)} \\ X_1^{(2)} & X_2^{(2)} & \dots & X_n^{(2)} \\ \dots & \dots & \dots & \dots \\ X_1^{(3)} & X_2^{(3)} & \dots & X_n^{(3)} \end{bmatrix}$$

的式子, 常用于数据的预测.

1. Hypothesis Function:

Declare：m是样本的数量，n是数据的维度, x_i表示第i个维度,$x^{(i)}$表示第i个数据

$$\hat{y} = h_\theta(x) = \theta^T X = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

2. Cost Function

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

- 这里的cost function和统计学中的RSS(Root-Sum-Squares)的作用相似, 相当于所有点和拟合的曲线上对应点的变差值的平方和, 即模型的总误差
- Goal: Choose parameters $\theta_0, \theta_1, \dots, \theta_n$ to minimize$J(\theta_0, \theta_1, \dots \theta_n)$

3. 模型的求解

基本方法是根据数据集X来求取$J(\theta_0, \theta_1, \dots, \theta_n)$这里有两种求解的algorithm.

3.1 GD(Gradient Descent) algorithm

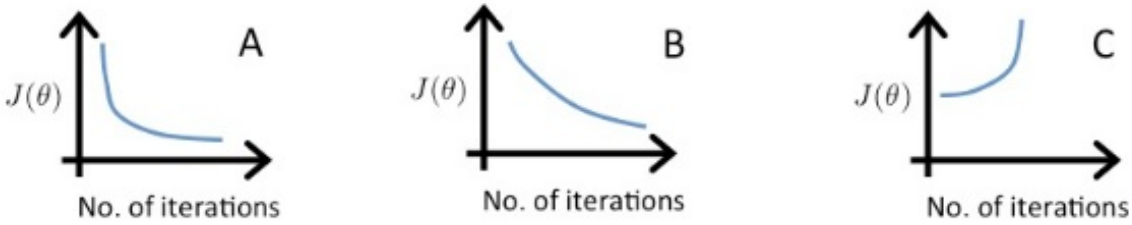
GD algorithm就是俗称的梯度下降法

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_1} J(\theta)$$

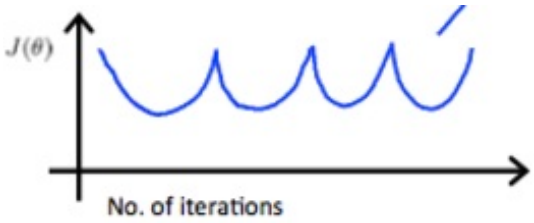
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta（x^{(i)}) - y^{(i)}) x_j^{(i)}$$

1. α的取值

α的取值太大那么下降速度将会太快, 如果α的取值太小则下降的速度会太慢



图A为合适的α取值, 图B为太小的α取值B， 图C为太大的α取值



这种情况是由于α的取值过大

通常我们找到一个会找到一个较大的α，然后按照3倍的速度进行减小直到得到一个合适的α(即converge)这里的合适与否是(根据我们能够接受的偏差量ϵ决定的, 通常情况下会让$\epsilon < 10^{-3}$

2. 求解θ

2.1 In mutli-variable linear regression:

$$\theta = \theta - (\alpha \frac{1}{m} \sum_{i=1}^{m} (X * \theta - y) .* X)^T$$

2.1 normal situation

Repeat{

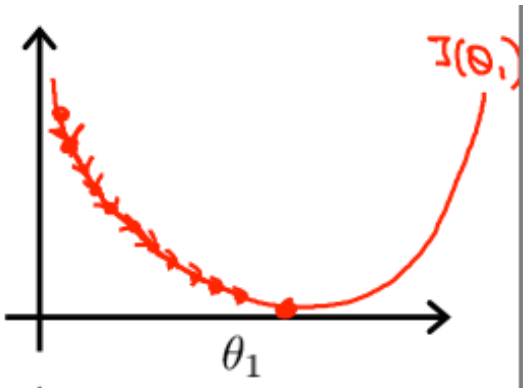
$$\theta_j = \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

}

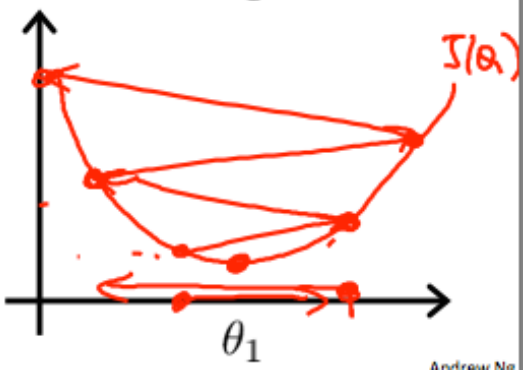
以one varible linear regression为例子

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.

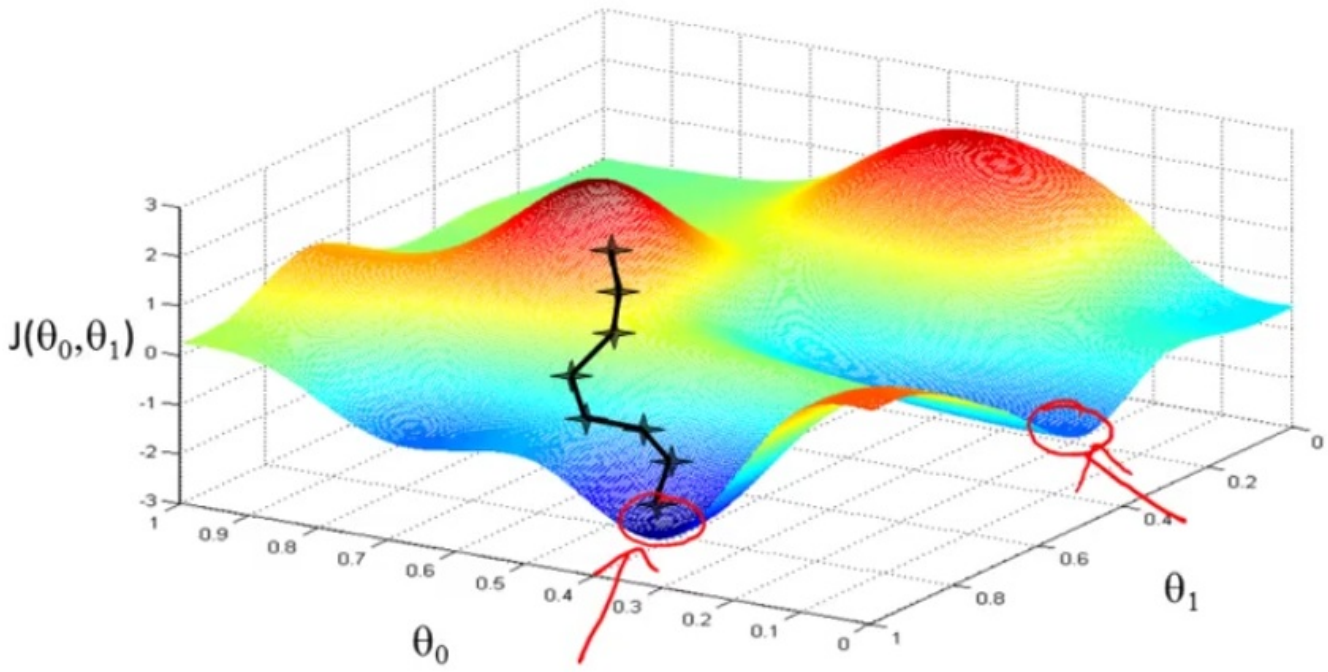


If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

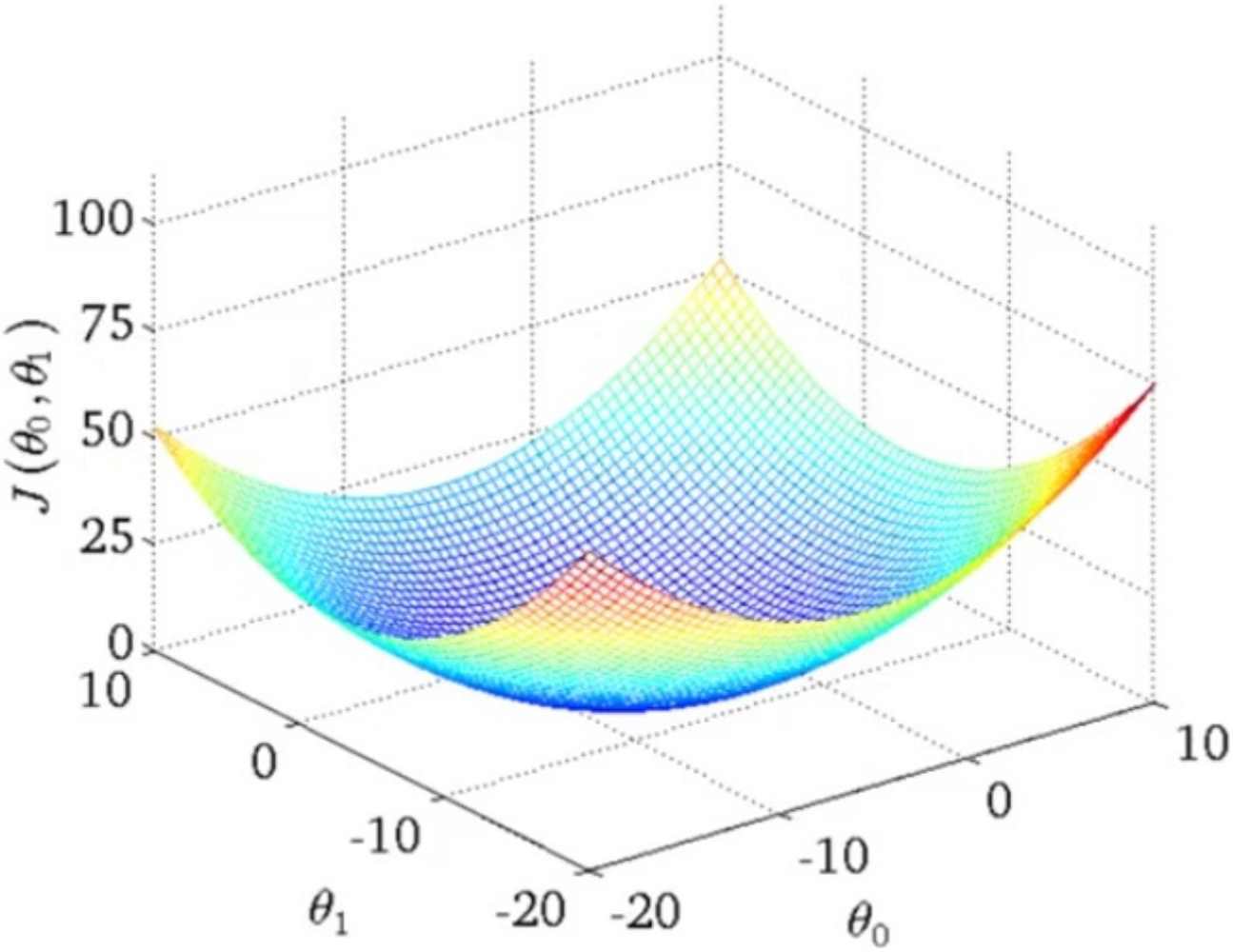


- 注意每次迭代，$\theta_j j = (1, 2, 3, \dots)$需要同时更新赋值
  - Example:
  - CORRECT:
    - $t_0 = \theta - \alpha \frac{d_J}{d_{\theta_0}}$
    - $t_1 = \theta - \alpha \frac{d_J}{d_{\theta_1}}$
    - $\theta_0 = t_0$
    - $\theta_1 = t_1$
  - WRONG:
    - $t_0 = \theta - \alpha \frac{d_J}{d_{\theta_0}}$
    - $\theta_0 = t_0$
    - $t_1 = \theta - \alpha \frac{d_J}{d_{\theta_1}}$
    - $\theta_1 = t_1$

3. GD只能得到一个局部最优解



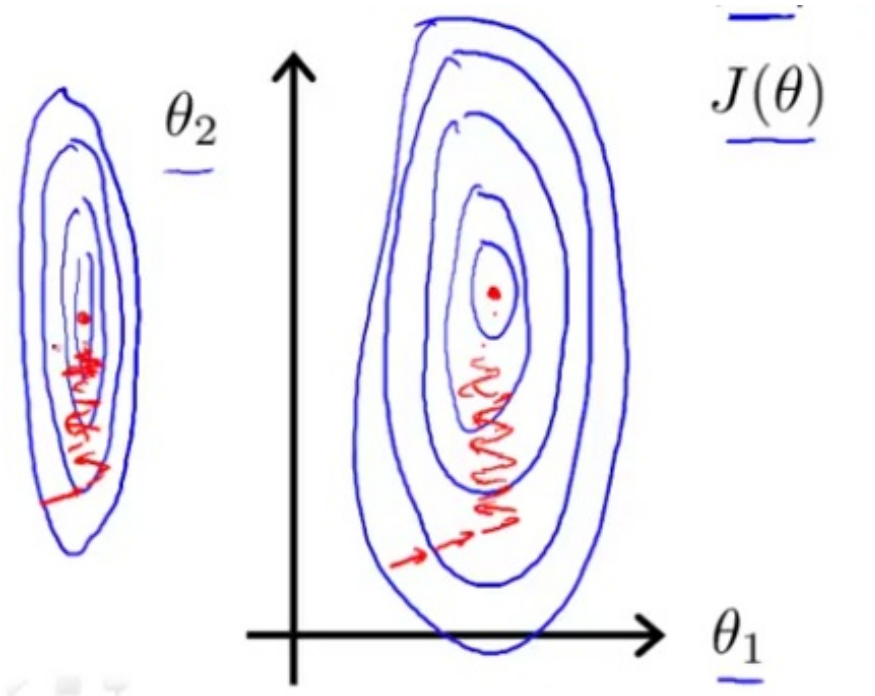
如上图所示梯度下降法可能会达到这两个点中的任意一个, 这两个点的斜率都为0. 但是再一些问题中, 如一元线性回归中局部最优解唯一, 即局部最优解就是全局最优解. 图新如下的凸函数(convex):



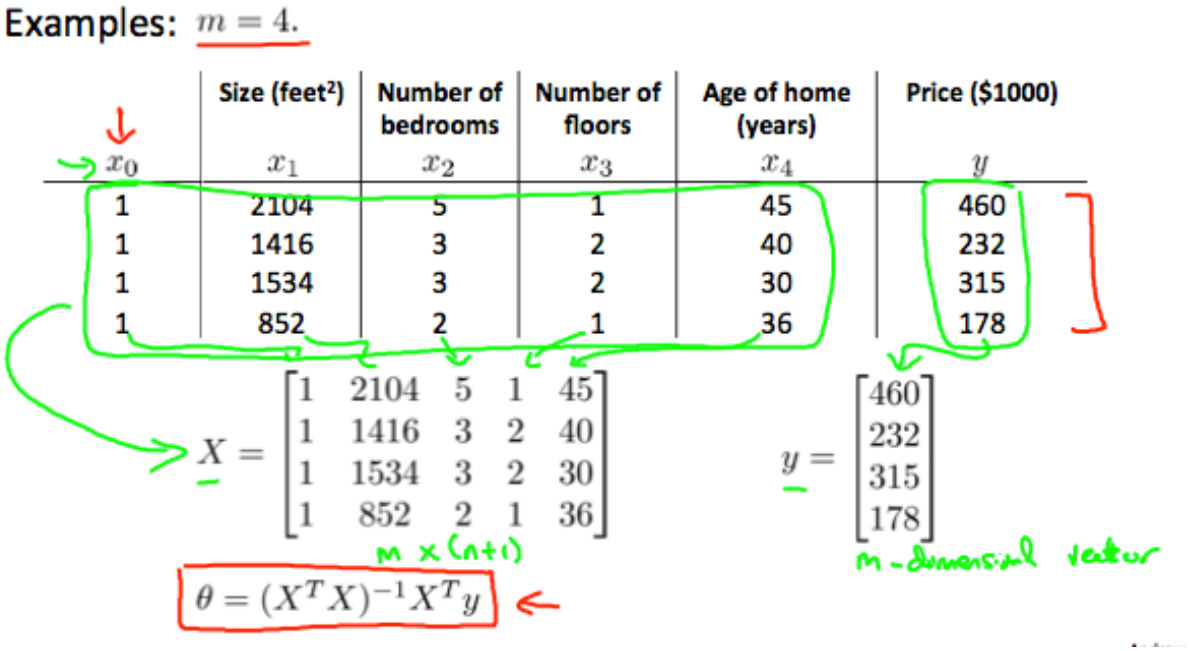
4. Feature Scaling

Idea: Make sure features are on a similar scale.

- 常用mean和range进行标准化$x_j^{(i)} = \frac{x_j^{(i)} - mean}{range}$
- 如果不进行标准化, 梯度下降的速度将会很慢, 如下图所示左右波动将会很剧烈.



3.2 Normal Regression



- Normal Regression: $\boldsymbol{\theta = (X^T X)^{-1} X^T y}$
  利用这个公式算出来的$\boldsymbol{\theta}$就是最优解.
- Normal Equation Noninvertibility：
  When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.' The 'pinv' function will give you a value of θ even if $\boldsymbol{X^T X}$ is not invertible.

如果$X^T X$不可逆, 原因可能为:

- **Redundant features**, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization"，即样本太少, 需要研究的特征太多

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

3.3 Compare about GD and NR:

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$
Works well when n is large	Slow if n is very large

当$m > 10^4$时一般就不用Normal Regression计算θ了.

3.4 Polynomial Regression

Logistic regression

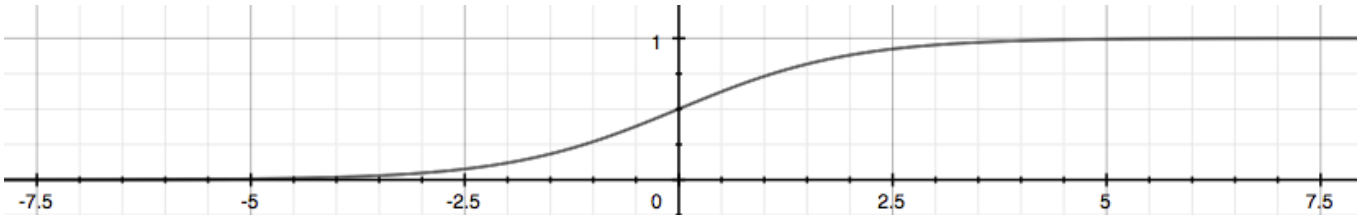
Logistic regression is a kind of classification algorithm.

1. Binary Classification

1.1 Logistic Regression Model

Logistic function 又叫 Sigmoid function， 很像统计学中假设检验的power function $E_\theta \varphi$

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

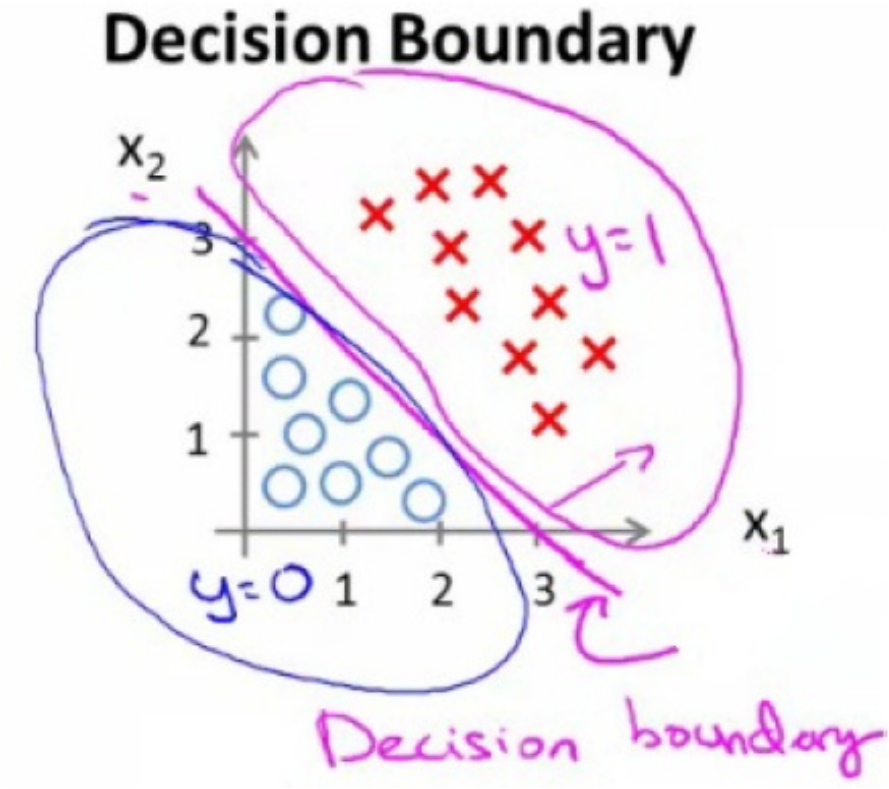


In this situation:

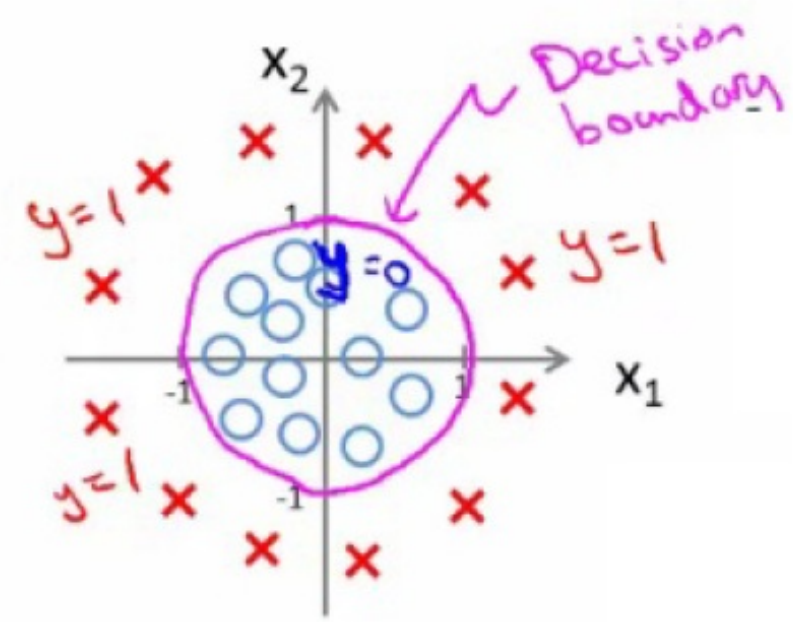
$$Desition(\theta^T x) = \begin{cases} 1 & \theta^T x \geq 0 \\ 0 & \theta^T x \leq 0 \end{cases}$$

本质上来讲$h_\theta(x) = P(y|x,\theta)$, probability that y, given x, parameterized by θ. 我们这里的分类focus on 0-1 classification.

- Desition boundary
  建立一个x1-x2的坐标系，确定分类边界



- $\theta = \begin{bmatrix} -3 & 0 & 0 \end{bmatrix}^T$
- Boundary: $x_1 + x_2 = 3$
- $x_1 = x_2 \leq 3 \rightarrow y = 1$



- $h_\theta = g(\theta * X)$
- $\theta = [-1 \quad 0 \quad 0 \quad 1 \quad 1]$
- Boundary: $x_1^2 + x_2^2 = 1$

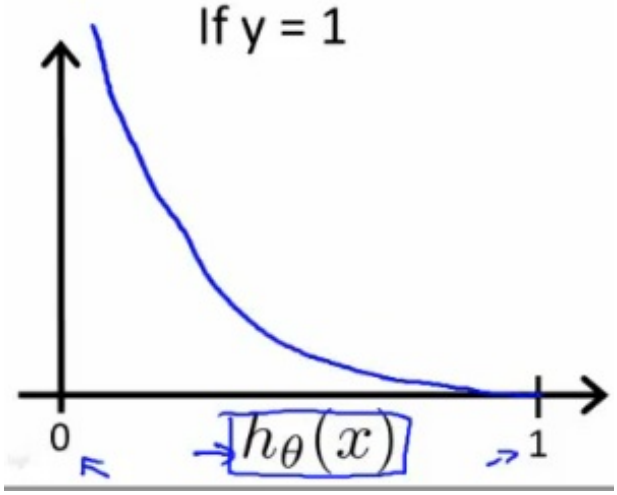
1.2 Cost function

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

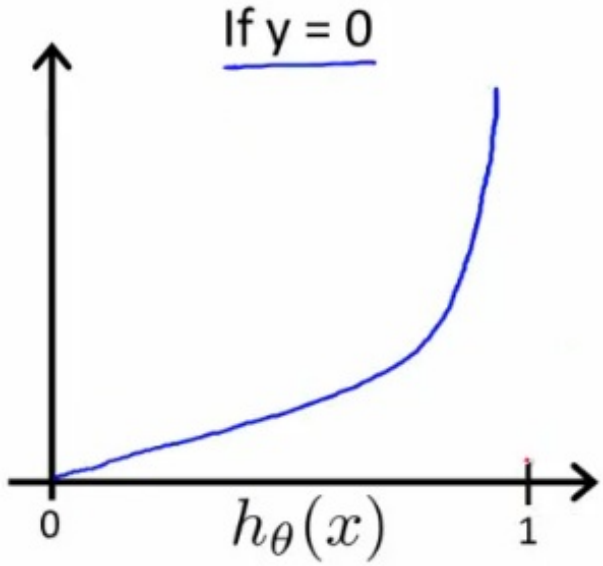
等价于:

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

When y = 1, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:



Similarly, when y = 0, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:



We can fully write out our entire cost function as follows:

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

A vectorized implementation is ;

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} * (-y^T \log(h) - (1-y)^T \log(1-h))$$

1.3 GD(Gradient Descent)

General form of gradient descent is:

Repeat{

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

}

we can work out the derivative part using calculus to get:

Repeat{

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

}

** About part is same with that in linear part**

The difference between them is that the $h_\theta(x)$ of LogRegression is $\frac{1}{1+e^{\theta^T x}}$ and the $h_\theta(x)$ of LogRegression is $\theta^T x$.

In this case, we can translate the GD formula into:

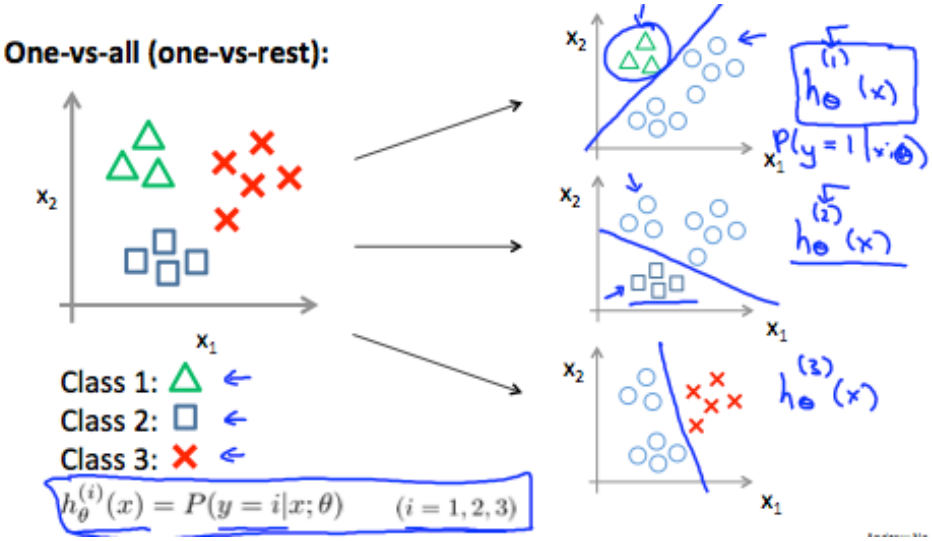
$$\theta = \theta - \frac{\alpha}{m} X^T(g(X\theta) - y)$$

Advanced Optimization

+ Conjugate gradient
+ BFGS
+ L-BFGS

不需要α,速度更快但较为复杂

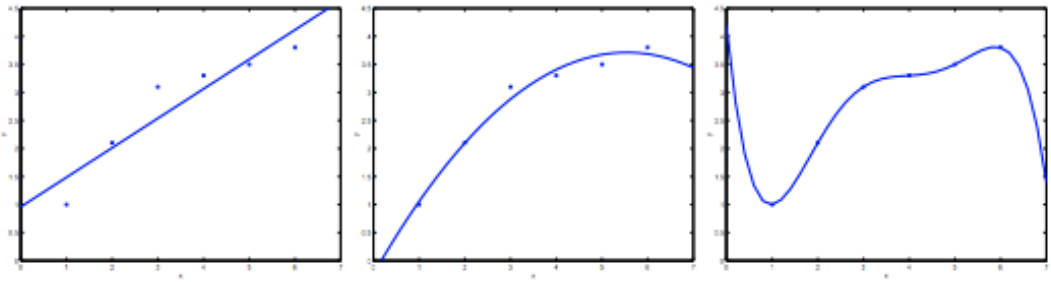
2. Multiclass Classification:



使用逻辑回归分类器$h_\theta^i(x)$来预测$y = i$的概率

- $y \in \{0, 1, 2, \ldots, n\}$
- $h_\theta^{(0)} = P(P(y = 0|x, \theta))$
- $h_\theta^{(1)} = P(P(y = 1|x, \theta))$
- ...
- $h_\theta^{(n)} = P(P(y = n|x, \theta))$
- 对于测试数据x, 将其归类为 $h_\theta^{(i)}$最大的那一类，本质上就是likelihood最大的那一类

3. Overfitting



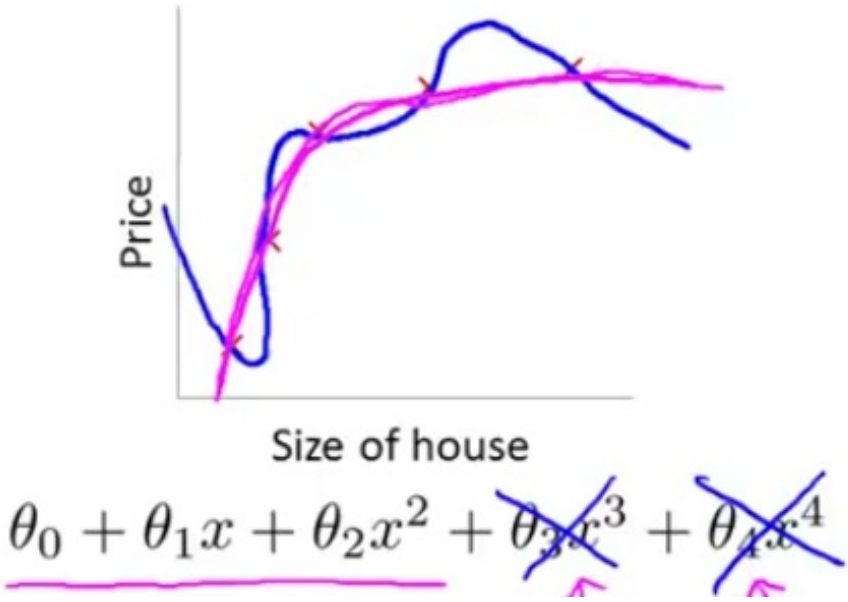
Underfitting: high bias (first graph above)
Overfitting: high variance (last graph above)

两种解决overfitting的方法:
1) Reduce the number of features:

- 只选取和response相关的一部分p预测因子,组成subset进行fit.

2) Regularization

- Shrinkage(regularization)
  - 不改变预测因子数量，但是部分预测因子系数为0；
  - 包括岭回归Ridge Regression以及Lasso dimension reduction
  - 通过对原有预测因子进行线性组合，得到一组简化的新预测因子
  - 使用新预测因子进行线性拟合，优化bias-variance trade-off
- Cost function of regularization



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

- λ: regularization parameter
  - λ:增加，系数模权重增加，相当于更多的被设置为0，模型被简化
  - 简化模型，Var逐渐降低，Bias逐渐升高

Regularized linear regression

对$\lambda\sum_{j=1}^{n}\theta_j^2$进行惩罚, 这一项是不考虑$\theta_0$的, 因为对常数项进行惩罚是没有意义的

Repeat{

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

$$\theta_0 = \theta_0 - \alpha[(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}) + \frac{\lambda}{m}\theta_j] \quad j \in \{1,2,\dots,n\}$$

}

Normal Equation:

$$\theta = (X^TX + \lambda L)^{-1}X^Ty$$

WHERE $L_{(n+1)*(n+1)} = \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \dots \\ & & & & 1 \end{bmatrix}$

Regularized Logistic Regression

格式类似线性回归,只是$h_\theta(x)$为sigmoid.

Neural Networks:Representation

参考资料1

4.1 Neural Networks-Model Representation

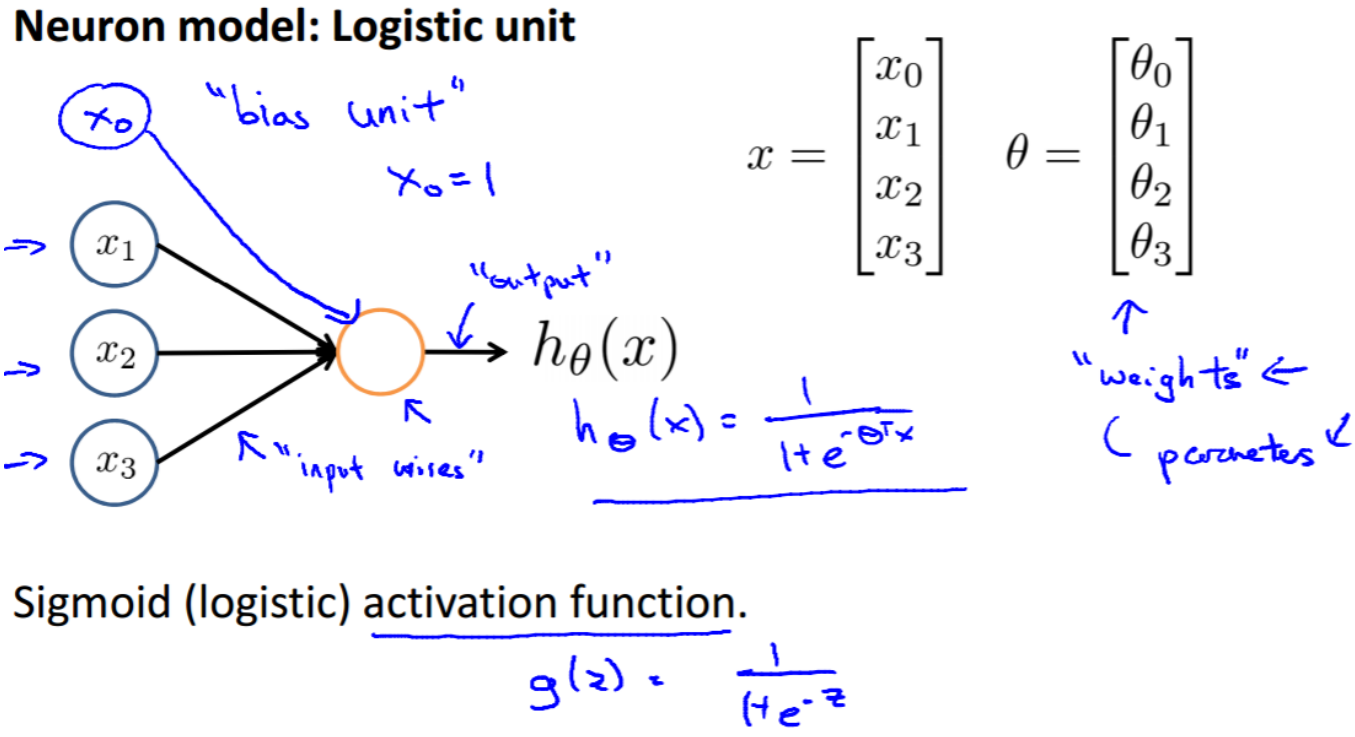
Neural Networks似乎可以看作是一个多层计算的模型, 它有一个输入层和一个输出层，中间包含一个以上的中间层。

可以解决维数较高的分类问题。由于随着维数的上升Logestic Regression 回归的计算量呈指数上升，因此高维的分类问题我们通过神经网络进行解决。如图像识别等

每一层神经元都进行一次计算，通过θ向量(权重)与前面神经元相乘得到新神经元，一直到最后

一层得出结果

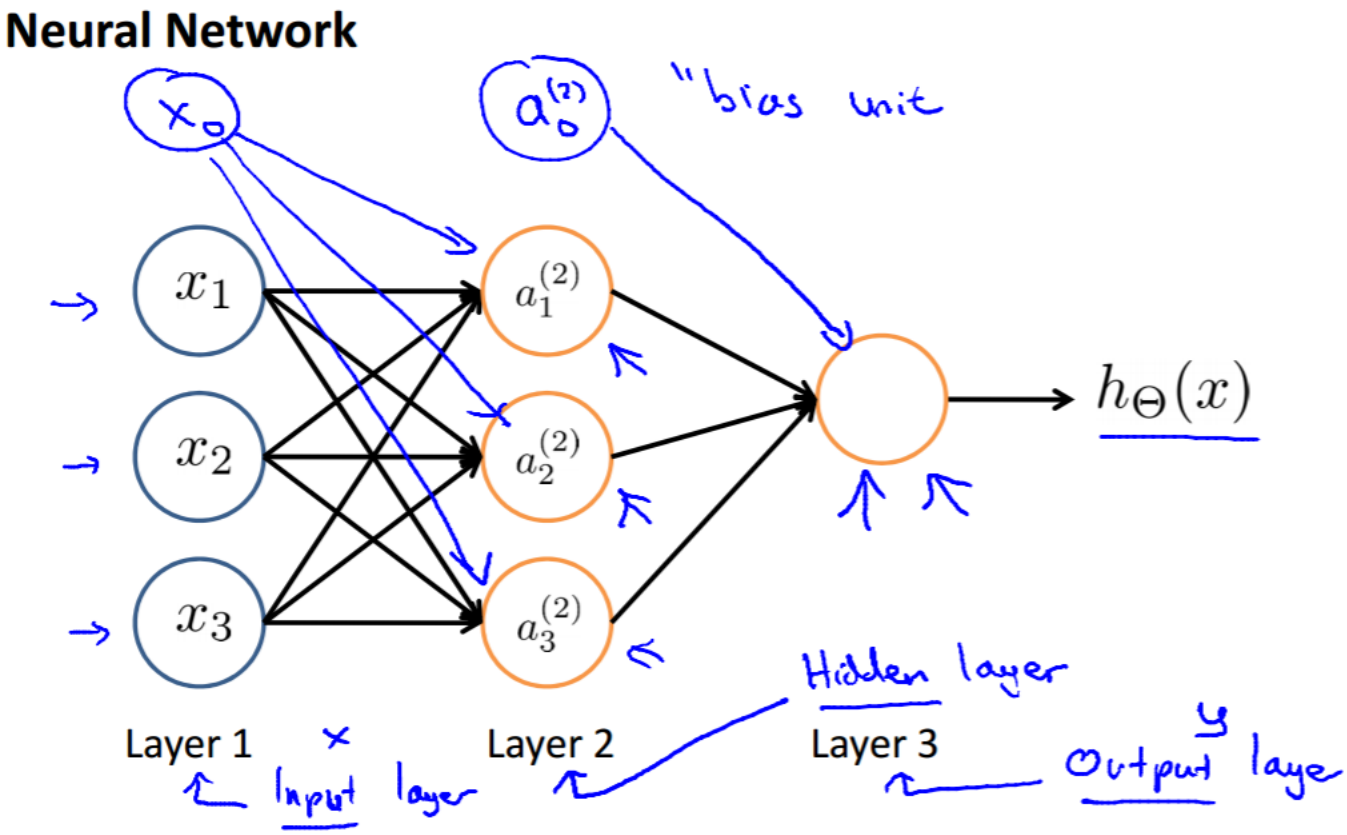
4.1.1 Neuron model: logistic unit



Sigmoid (logistic) activation function.

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

Neural Networks-Model 由许多的Neuron unit组成他们的上一层的所有数据x_1 x_n和x_0常数项组成的矩阵X权重矩阵θ的乘积构成了activation function的输入。再代入activation function计算得到这一层的神经元的数值, 作为下一层的输入。



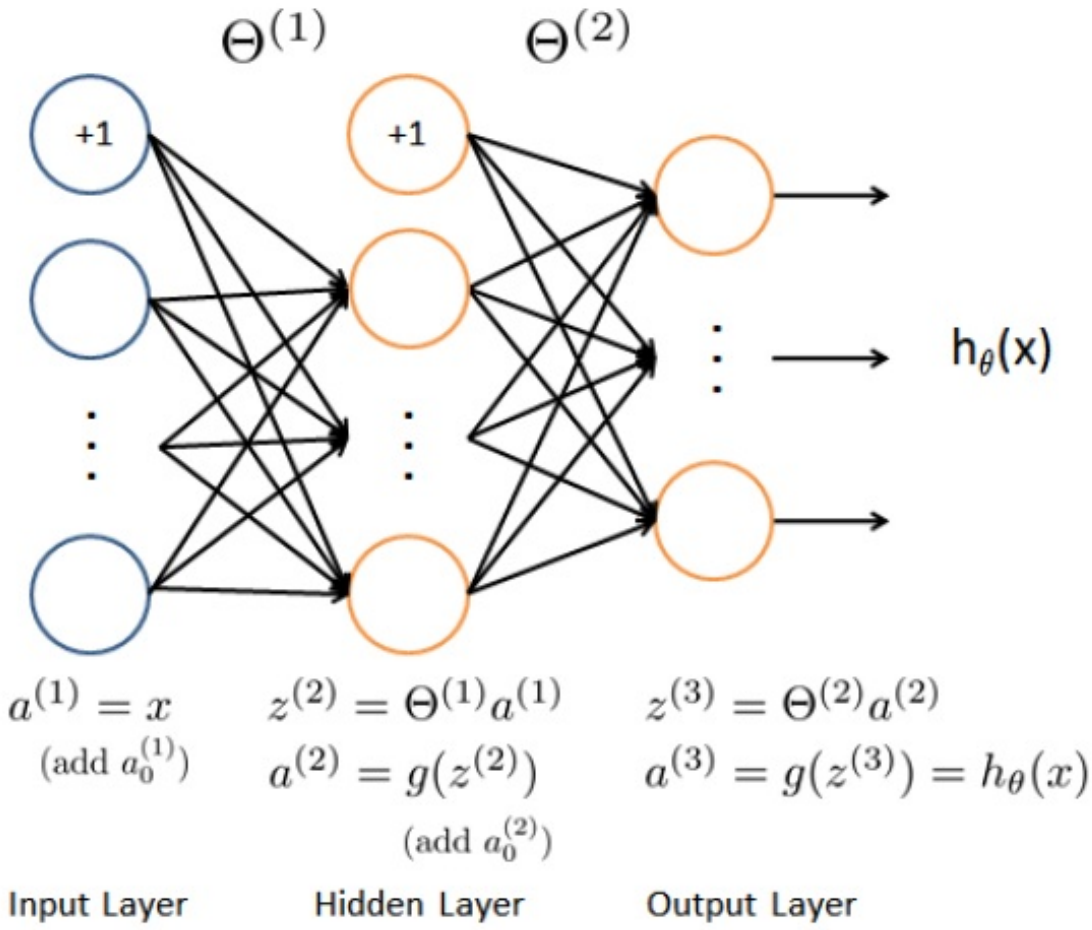
上图中的x_0和$a_0^{(2)}$不在模型中直接表示出来, 他们是一个修正unit，即常熟项。因此再计算的过程中数据矩阵应为$X = [ones(size(X)), X]$.

Example:

对上面的Neural Network进行计算的过程如下

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

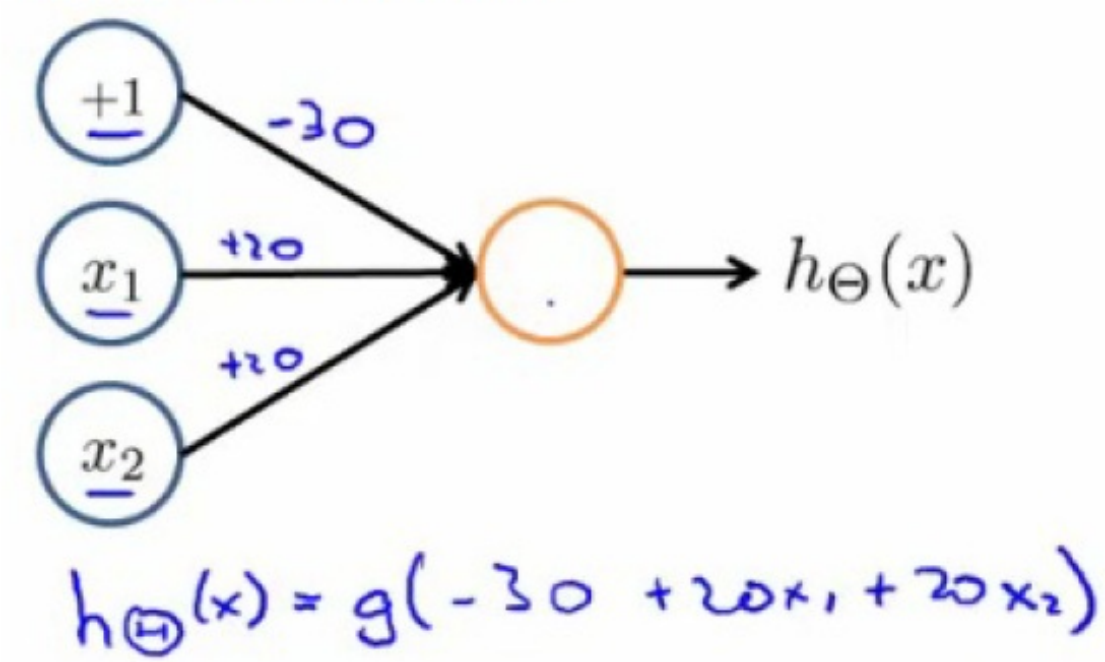
以层为单位的，通过矩阵运算进行计算，的神经网络计算过程如下图所示



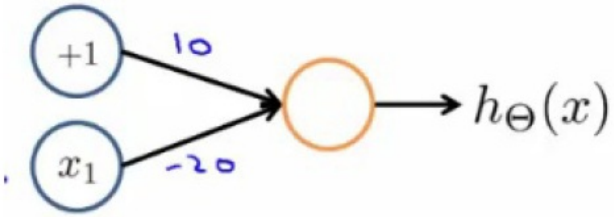
4.2 Application Example: 判断逻辑关系

- x1 And x

- $x = [1, x_1, x_2]^T$
- $\theta = [-30, 20, 20]$
- $h_\theta(x) = g(-30 + 20x_1 + 20x_2)$
- 根据真值表可得到结论AND

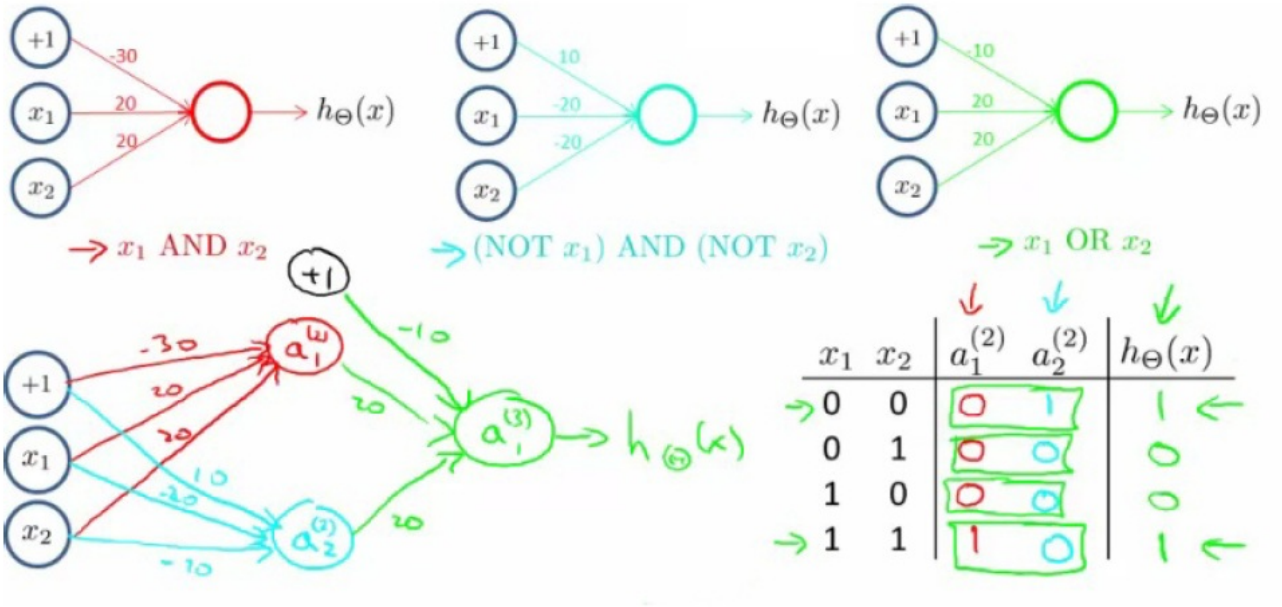


- Not x1
  - $x = [1, x_1]^T$
  - $\theta = [10, -20]$
  - $h_\theta(x) = g(10 - 20x_1)$
  - 根据真值表可得到结论NOT x_1



x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

- x1 ⊕ x2



$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \end{bmatrix} \rightarrow \left[a^{(3)} \right] \rightarrow h_\Theta(x)$$

$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

- 从第一层到第二层: $a^{(2)} = g(\Theta^{(1)} \cdot x)$

$$\Theta^{(2)} = \begin{bmatrix} -10 & 20 & 20 \end{bmatrix}$$

- 从第二层到第三层: $a^{(3)} = g(\Theta^{(2)} \cdot a^{(2)})$
- $h_\Theta(x) = a^{(3)}$, 根据真值表得到结论XNOR

4.3 Multiclass Classification

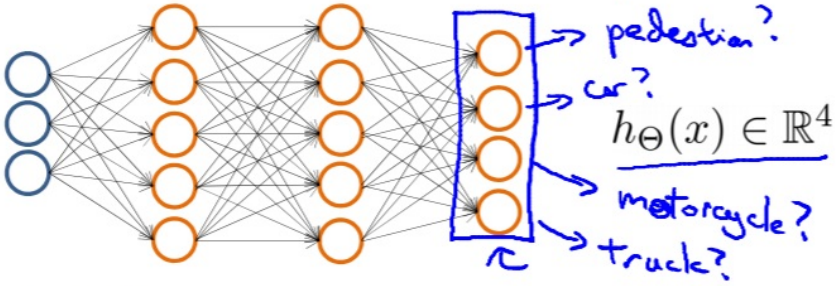
Multiple output units: One-vs-all.


Pedestrian


Car


Motorcycle

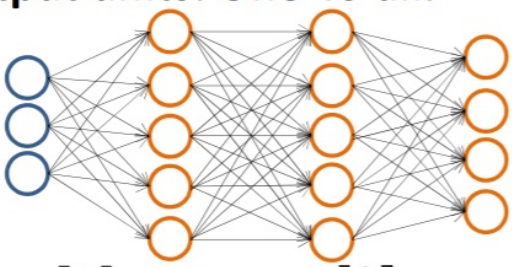

Truck



Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

Multiple output units: One-vs-all.



$h_\Theta(x) \in \mathbb{R}^4$

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian when car when motorcycle

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

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

Previously $y \in \{1, 2, 3, 4\}$

$h_\Theta(x^{(i)}) \approx y^{(i)}$, $\in \mathbb{R}^4$

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ \cdots \end{bmatrix} \rightarrow \begin{bmatrix} a_0^{(3)} \\ a_1^{(3)} \\ a_2^{(3)} \\ \cdots \end{bmatrix} \rightarrow \cdots \rightarrow \begin{bmatrix} h_\Theta(x)_1 \\ h_\Theta(x)_2 \\ h_\Theta(x)_3 \\ h_\Theta(x)_4 \end{bmatrix} \rightarrow$$

计算的结果是一个维度为mXn的矩阵(m为样本数，n为类别数)， 其中储存的是每一个样本为对应样本的概率， 最终取概率最大的那一类别最为最终的分类标签。

4.4 Cost Function and Backpropagation

Cost function for neural networks:

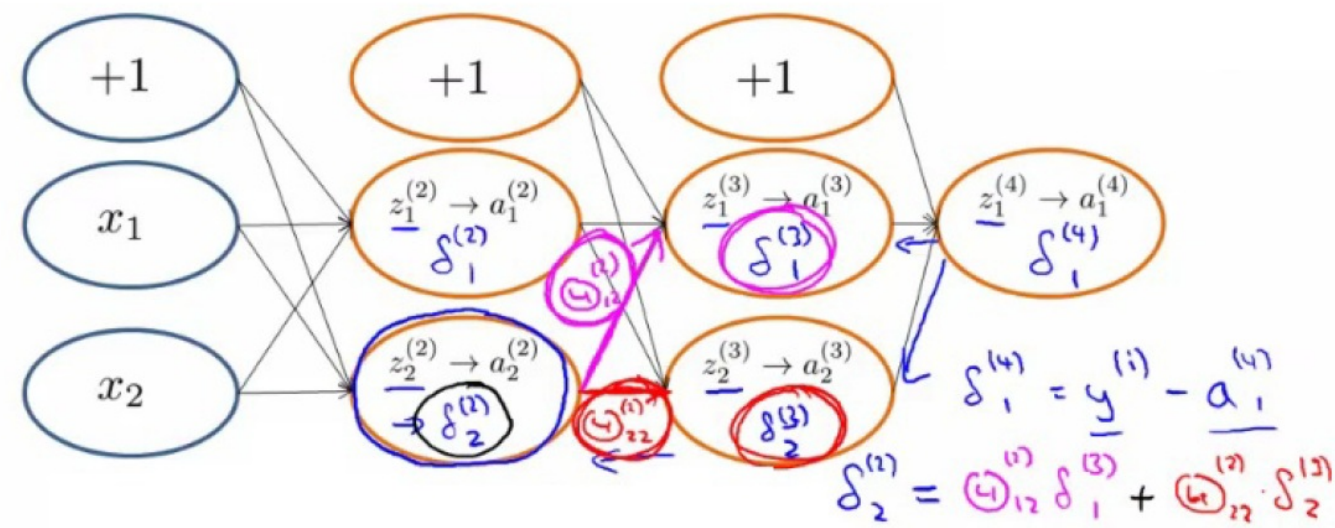
$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[y_k^{(i)}\log((h_\Theta(x^{(i)}))_k) + (1-y_k^{(i)})\log(1-(h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

套用了Logistic regression的cost function

- L = total number of layers in the network(input和output分别都算1层).
- s_l = number of units in layer l, 不包括bias unit $a_0^{(l)}$.
- K = number of output classe.

是用GD与之前的一样需要求出$J(\Theta)$, 并对其求偏导$\frac{dJ(\Theta)}{d\Theta_{j,i}^{(l)}}$由于直接计算过于复杂, 通常用Backpropagation进行优化

4.5 Backpropagation



- 后向传播是一种计算偏导数的有效算法, 思路如下:
  - 忽略bias项;
  - 给定一个样例(X, y)，首先进行前向传导，计算所有的节点值$a_j^{(l)}$，包括$h_\Theta(x)$的最终输出值;
  - 从后向前, 针对第l层的每一个节点j，计算残差$\delta_j^{(l)}$，该残差表明了当前节点对最终输出值的
    残差产生了多少影响;
    最后根据节点$a_j^{(l)}$和$\delta_j^{(l)}$得到偏导数;
- 计算过程：
  - 对于最后一层(输出层)，可以直接算出残差
    $\delta^{(L)} = a^{(L)} - y$
  - 从后向前, 逐层计算残差(到第二层为止):
    $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* g'(z^{(l)})$
  - 偏导数的求取,以这里取activation function为sigmoid为例:
    - $g(z) = \frac{1}{1+e^{-z}}$
    - 则偏导数可以表示为$g'(z) = \frac{e^{-z}}{(1+e^{-z})^2} = g(z)(1 - g(z))$
  - 综上所述, 可以得到残差计算公式为:
    $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$
    - 对于四层的神经网络:$\delta^{(4)} \to \delta^{(3)} \to \delta^{(2)}$
      - $\delta^{(4)} = a^{(4)} - y = h_\Theta(x) - y$
      - $\delta^{(3)} = ((\Theta^{(3)})^T \delta^{(4)}) .* a^{(3)} .* (1 - a^{(3)})$
      - $\delta^{(2)} = ((\Theta^{(3)})^T \delta^{(3)}) .* a^{(2)} .* (1 - a^{(2)})$

- 根据得到的残差和节点值，代入下式即可得到偏导:
  $\frac{dJ(\Theta)}{d\Theta_{j,i}^{(l)}} = \frac{1}{m}\sum_{t=1}^{m} a_j^{(t)(l)} \delta_i^{(t)(l+1)}$

- Backprogation Algorithm
  - Training set:$\{(x(1),y(1)),(x(2),y(2)),\dots(x(m),y(m))\}$
  - Initialization : set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j)
  - For training example $t=1,2,\dots,m$:
    - Set $a^{(1)} := x^{(t)}$
    - Perform forward propagation to get $a^{(2)},a^{(3)},\dots,a^{(L)}$
    - Using $y^{(t)}$,compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
    - Perform backword propagation to get $\delta^{(L-1)},\delta^{(L-2)},\dots,\delta^{(2)}$
    - Compute $\Delta_{i,j}^{(l)} = \Delta_{i,j}^{(l)} + a_j^{(l)}\delta^{(l+1)}$
  - 最终得到$D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}$

$$D_{i,j}^{(l)} := \begin{cases} \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}\right) & \text{if j=1} \\ \frac{1}{m}\Delta_{i,j}^{(l)} & \text{if j=0} \end{cases}$$

4.6 Implementation Note:

4.6.1 Unrolling Parameters into Vectors

将所有矩阵放入一个向量内: original matrices → unrolled vector

```
thetaVector = [Theta1(:); Theta2(:); Theta3(:);]
deltaVector = [ D1(:); D2(:); D3(:) ]
```

返回原来的矩阵: unrolled vector → original matrices

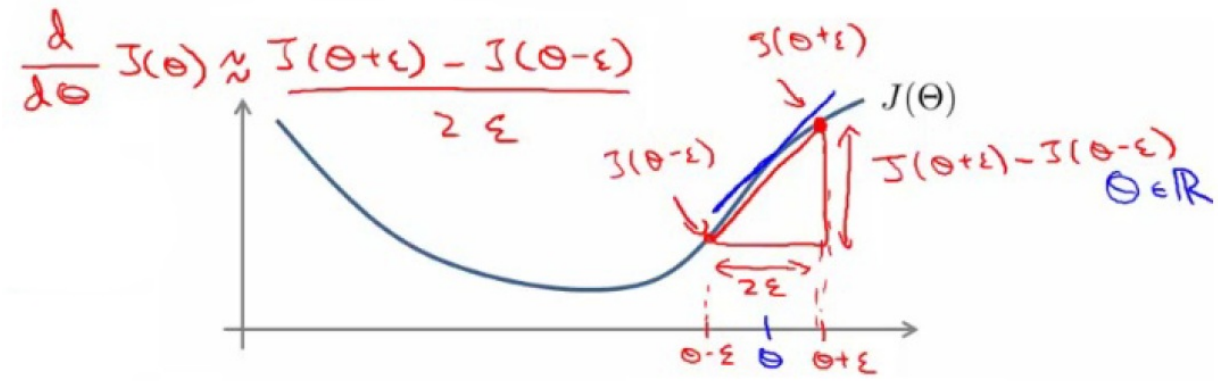
```
thetaVec=[Theta1(:);Theta2(:);Theta3(:)];#length=231
DVec=[D1(:);D2(:);D3(:)];
Theta1=reshape(thetaVec(1:110),10,11);
Theta2=reshape(thetaVec(111:220),10,11);
Theta3=reshape(thetaVec(221:231),1,11);
```

Learning Algorithm: initialTheta → optTheta

+ `function[jVal,gradient]=costFunction(thetaVector);`
+ Get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ from thetaVector;
+ Use FP & BP to compute $D^{(1)}, D^{(2)}, D^{(3)}, J(\Theta)$
+ Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec

4.6.2 Gradient Checking

- 可用于检验cost function是否bug-free
- 典型bug:
  - 缺位错误（Off-by-one error)：比如次循环，正确应该是`for(i=1;i<=m;i++)`,写成`for(i=1;i<m;i++)`就是缺位错误
- 使用步骤
  - 给定样本及参数初始值;
  - 使用BP计算`Gradient DVec(unrolled D(1),D(2),D(3))`
  - 使用NumericalGradientChecking计算近似的gradApprox;
  - 比较grad与gradApprox是否近似, 若不近似, + 说明出现bug需要检查cost函数的合理性
  - 若检查无误，在使用前屏蔽Gradient checking部分的代码(在训练过程中只用BP，否则严重影
    响速度)



- NumericalGradientChecking:

$$\frac{\partial J(\Theta))}{\partial \Theta} \approx \frac{J(\Theta+\epsilon)-J(\Theta-\epsilon)}{2\epsilon}$$

  - for $\Theta_j \in \{\Theta_1, \Theta_2, \ldots, \Theta_n\}$:

$$\frac{\partial J(\Theta))}{\partial \Theta_j} \approx \frac{J(\Theta_1, \Theta_2, \ldots, \Theta_j+\epsilon, \ldots \Theta_n)-J(\Theta_1, \Theta_2, \ldots, \Theta_j-\epsilon, \ldots \Theta_n}{2\epsilon}$$

  - 通过计算两边的接近程度来判定函数是否合理

- 代码实现:

```
for i=1:n,
    thetaPlus=theta;
    thetaPlus(i)=thetaPlus(i)+EPSILON;
    thetaMinus=theta;
    thetaMinus(i)=thetaMinus(i)-EPSILON;
    gradApprox(i)=(J(thetaPlus)-J(thetaMinus))/(2*EPSILON);
end;
```

4.6.3 Random Initialization

- 随机初始化参数θ
- 不使用Zero initialization(所有初始值为0)的原因:
  - 若$\theta_{i,j}^{(l)}=0$ for all l,i,j
  - 则$a_1^{(2)},a_2^{(2)}$的求取公式相同
  - 导致隐藏层中的权重相同$a_1^{(2)}=a_2^{(2)}$，不合理
  - 因此需要随机初始化参数来打破对称
- 随机初始化方法: 给定对称定义域$[-\varepsilon,\varepsilon]$, 在该定义域中随机取值

```
If the dimensions of Theta1 is 10x11, Theta2 is 10x11 and Theta3 is
1x11.

Theta1 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta2 = rand(10,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
Theta3 = rand(1,11) * (2 * INIT_EPSILON) - INIT_EPSILON;
```

- 注意随机初始化和梯度下降无关,不会互相影响

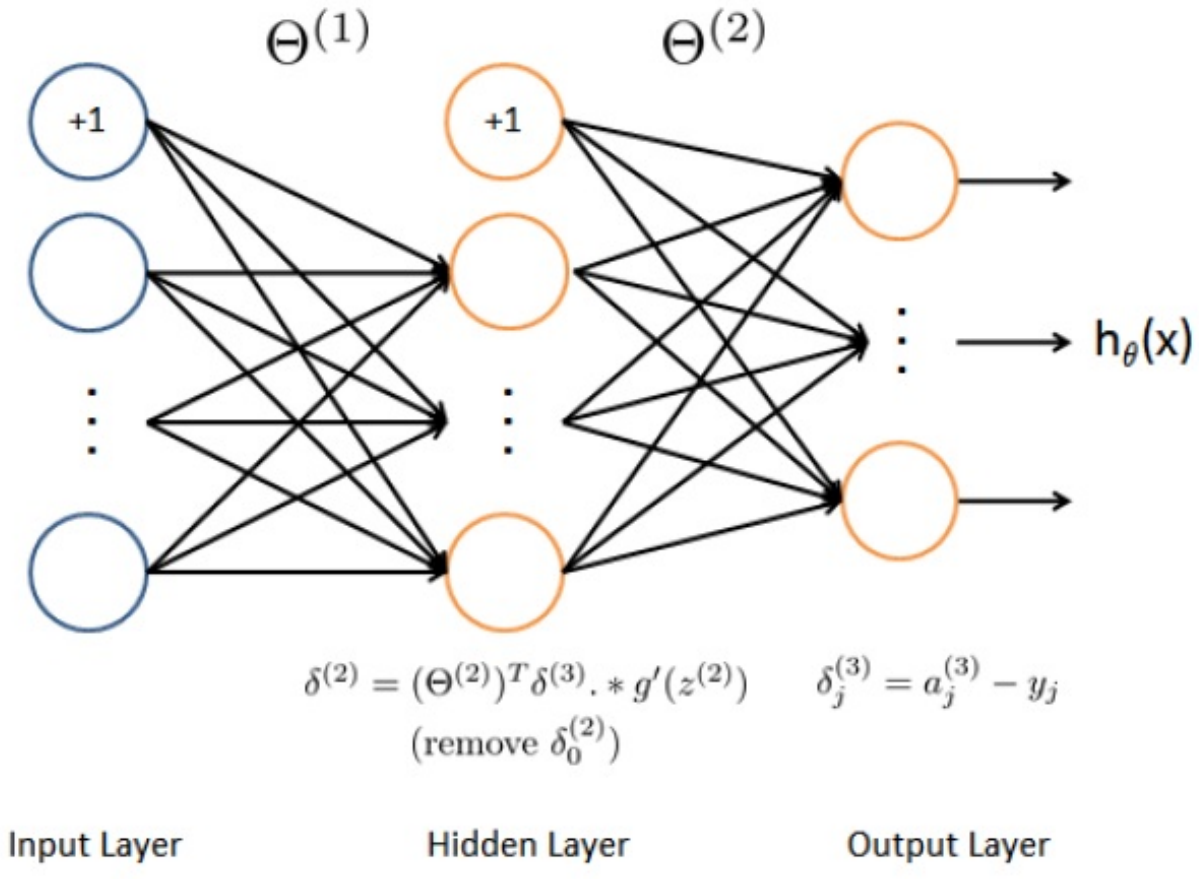
4.6.4 Summary

- 首先确定连接模式
  - 设置输入单元数: Dimension of features $x^{(i)}$;
  - 设置输出单元数: Number of classes;
  - 设置隐藏层数:
    - 默认为一层隐藏层;
    - 若不止一层，则需要确保每层的单元数S_l相同;
    - 一般情况下隐藏层单元数越多越好;
- 训练神经网络模型

- 随机初始化参数;
- 使用前向传播计算所有activation function即所有的$a_j^{(i)}$;
- 构建cost function;
- 使用backpropagation计算所有偏导数;
- 使用梯度检验校验backpropagation的计算结果, 则在使用模型时屏蔽这部分代码;
- 在后向传播的基础上, 使用梯度下降或其他方法, 使偏差即cost function $J(\Theta)$最小化;

```
for i = 1:m,
   Perform forward propagation and backpropagation using example (x(i), y(i))
   (Get activations a(l) and delta terms d(l) for l = 2,...,L
```

Example: Octave code of implement FeedForward and BackPropagation.




```
function [J grad] = nnCostFunction(nn_params, ...
                                   input_layer_size, ...
                                   hidden_layer_size, ...
                                   num_labels, ...
                                   X, y, lambda)
%NNCOSTFUNCTION Implements the neural network cost function for a two l
```

```
ayer
%neural network which performs classification
%   [J grad] = NNCOSTFUNCTON(nn_params, hidden_layer_size, num_labels, ...
%   X, y, lambda) computes the cost and gradient of the neural network. The
%   parameters for the neural network are "unrolled" into the vector
%   nn_params and need to be converted back into the weight matrices.
%
%   The returned parameter grad should be a "unrolled" vector of the
%   partial derivatives of the neural network.
%

% Reshape nn_params back into the parameters Theta1 and Theta2, the weight matrices
% for our 2 layer neural network
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
                 hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
                 num_labels, (hidden_layer_size + 1));

% Setup some useful variables
m = size(X, 1);

% You need to return the following variables correctly
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));

% ====================== YOUR CODE HERE ======================
% Instructions: You should complete the code by working through the
%               following parts.
%
% Part 1: Feedforward the neural network and return the cost in the
%         variable J. After implementing Part 1, you can verify that your
%         cost function computation is correct by verifying the cost
%         computed in ex4.m
%
% Part 2: Implement the backpropagation algorithm to compute the gradients
%         Theta1_grad and Theta2_grad. You should return the partial derivatives of
```

```
%         the cost function with respect to Theta1 and Theta2 in Theta1_grad and
%         Theta2_grad, respectively. After implementing Part 2, you can check
%         that your implementation is correct by running checkNNGradients
%
%         Note: The vector y passed into the function is a vector of labels
%               containing values from 1..K. You need to map this vector into a
%               binary vector of 1's and 0's to be used with the neural network
%               cost function.
%
%         Hint: We recommend implementing backpropagation using a for-loop
%               over the training examples if you are implementing it for the
%               first time.
%
% Part 3: Implement regularization with the cost function and gradients.
%
%         Hint: You can implement this around the code for
%               backpropagation. That is, you can compute the gradients for
%               the regularization separately and then add them to Theta1_grad
%               and Theta2_grad from Part 2.
%


%-------------------------------------Feedforward-------------------------------
a1 = [ones(m, 1) X];
z2 = a1 * Theta1';
a2 = sigmoid(z2);
a2 = [ones(m, 1) a2];
z3 = a2 * Theta2';
htheta = sigmoid(z3);

for k = 1:num_labels
    yk = y == k;
    hthetak = htheta(:, k);
    Jk = 1 / m * sum(-yk .* log(hthetak) - (1 - yk) .* log(1 - hthetak))
```

```
;
    J = J + Jk;
end

% implement the regularization
regularization = lambda / (2 * m) * (sum(sum(Theta1(:, 2:end) .^ 2)) + s
um(sum(Theta2(:, 2:end) .^ 2)));
J = J + regularization;

% ----------------------------------backpropagation---------------------
-----
for t = 1:m
    for k = 1:num_labels
        %let the column which equal to label k, equals to 1, other colum
ns equals to 0
        yk = y(t) == k;
        % htheta(t,k) is the probability calculated by Neural network th
at the t-th example
        % belong to k-th classes.
        delta_3(k) = htheta(t, k) - yk;
        % After this step we get the cost(bias) of the a3 layer
    end
    % Using the formula calculated the cost(bias) of the a2 layer
    size(Theta2)
    size(delta_3)
    delta_2 = Theta2' * delta_3' .* sigmoidGradient([1, z2(t, :)])';
    % remove the constant unit a2_0
    %
    % n = hidden_layer_size, the column is n+1 due to the ai_0 unit
    % theta_n_n+1 means n-th unit of layer to go, n+1-th unit of layer
from
    %
    % |theta_11 theta_12 ... theta_1_n+1|
    % |theta_21 theta_22  ... theta_2_n+1|
    % | ...     ...     ...
    % |theta_n1 theta_n2 ... theta_n_n+1|
    %
    delta_2 = delta_2(2:end);

    % change the
    Theta1_grad = Theta1_grad + delta_2 * a1(t, :);
    Theta2_grad = Theta2_grad + delta_3' * a2(t, :);
end

Theta1_grad = Theta1_grad / m;
Theta2_grad = Theta2_grad / m;
```

```
118.
119.   % implement the regularization
120.   Theta1_grad(:, 2:end) = Theta1_grad(:, 2:end) + lambda / m * Theta1(:, 2
       :end);
121.   Theta2_grad(:, 2:end) = Theta2_grad(:, 2:end) + lambda / m * Theta2(:, 2
       :end);
122.
123.
124.   % -------------------------------------------------------------
125.
126.   %
       =========================================================================
127.
128.   % Unroll gradients
129.   grad = [Theta1_grad(:) ; Theta2_grad(:)];
130.
131.
132.   end
```

Evaluating a Hypothesis

Once we have done some trouble shooting for errors in our predictions by:

- Getting more training examples(fixes high variance)
- Trying smaller sets of features(fixes high variance)
- Trying additional features(fixes high bias)
- Trying polynomial features(fixes high bias)
- Increasing λ(fixes high variance)
- decreasing λ(fixes high bias)

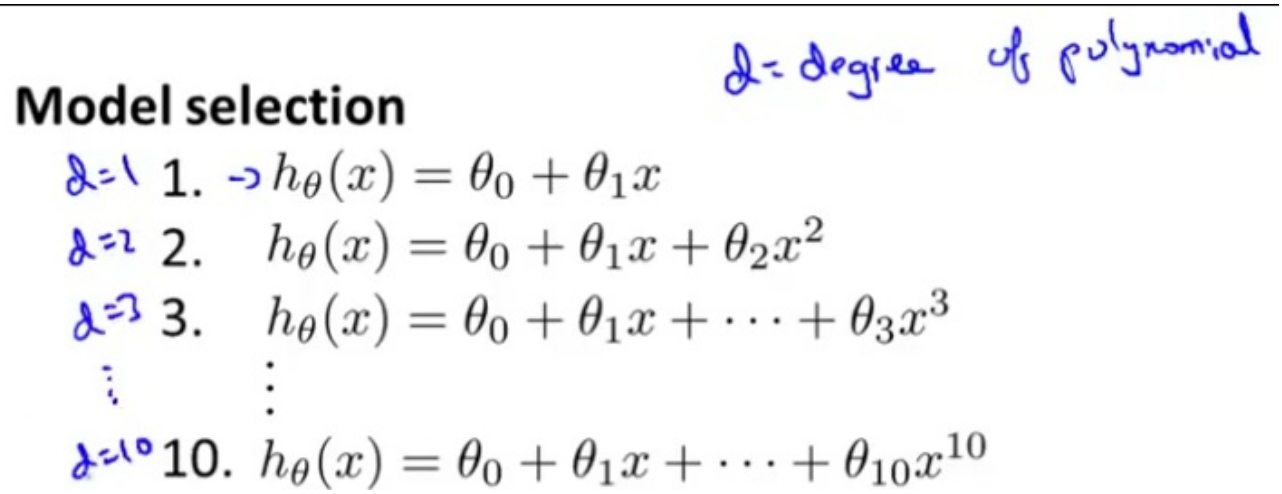
1. Divide the data set

A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a training set and a test set. Typically, the training set consists of 70 % of your data and the test set is the remaining 30 %.

The new procedure using these two sets is then:

Learn Θ and minimize $J_{train}(\Theta)$ using the training set
Compute the test set error $J_{test}(\Theta)$

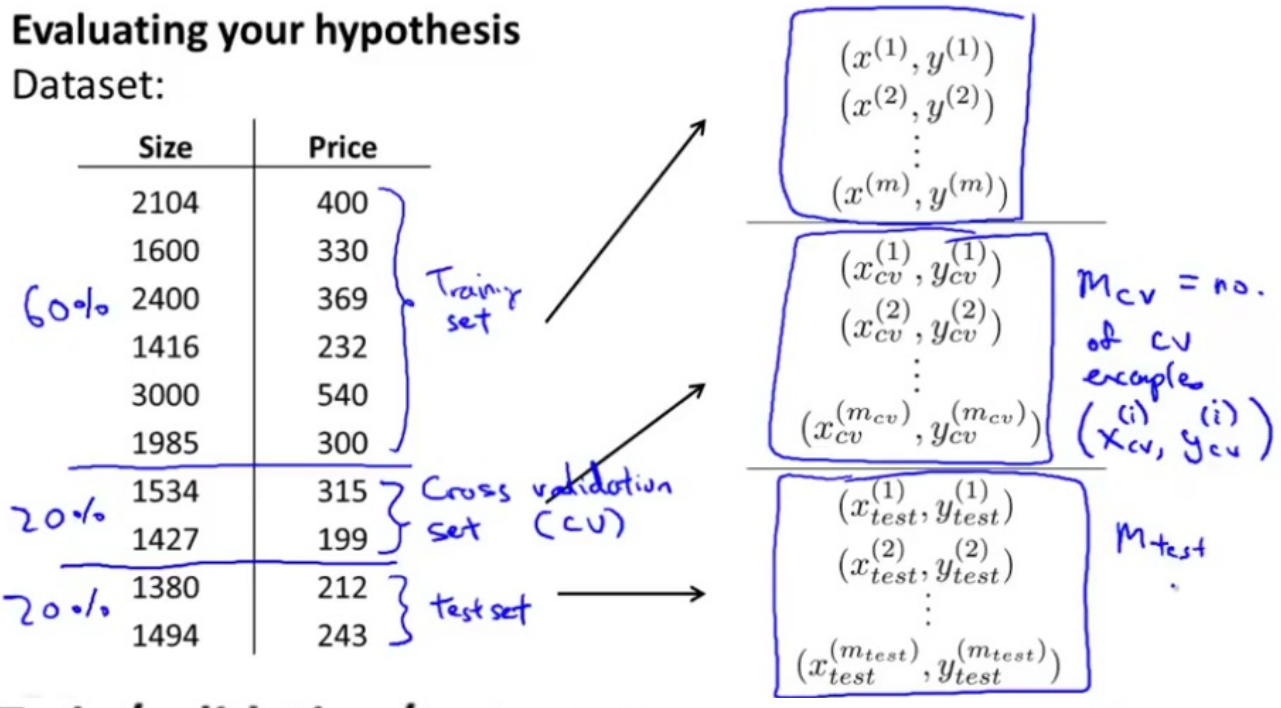
- The test set error
  - For linear regression:
    $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$
  - For classification
    $$err(h_\Theta(x), y) = \begin{cases} 1 & if (h_\Theta(x) \geq 0.5 \; and \; y = 0) \; or \; (h_\Theta < 0.5 \; and \; y = 1) \\ 0 & else \end{cases}$$
    $$TestError = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)})$$
    This gives us the proportion of the test data that was misclassified.
- Model selection



Evaluating your hypothesis

Dataset:

Size	Price	
2104	400	Training set (60%)
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	Cross validation set (CV) (20%)
1427	199	
1380	212	test set (20%)
1494	243	



Train/validation/test error

Training error:

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

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

Test error:

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

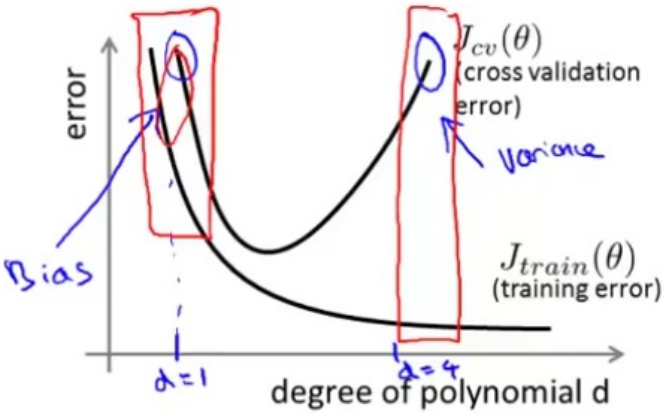
通过Training error: 计算得到10个degree各自的θ，根据validation Error选择$J_{validation}(\theta)$最小的模型，利用test error最终评估模型。

2. Variance and Bias

2.1 Variance and Bias的区别如下图所示:

Diagnosing bias vs. variance

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$J_{train}(\theta)$ will be low

$J_{cv}(\theta) \gg J_{train}(\theta)$

2.2 chosing the regularization parameter λ

Choosing the regularization parameter λ

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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

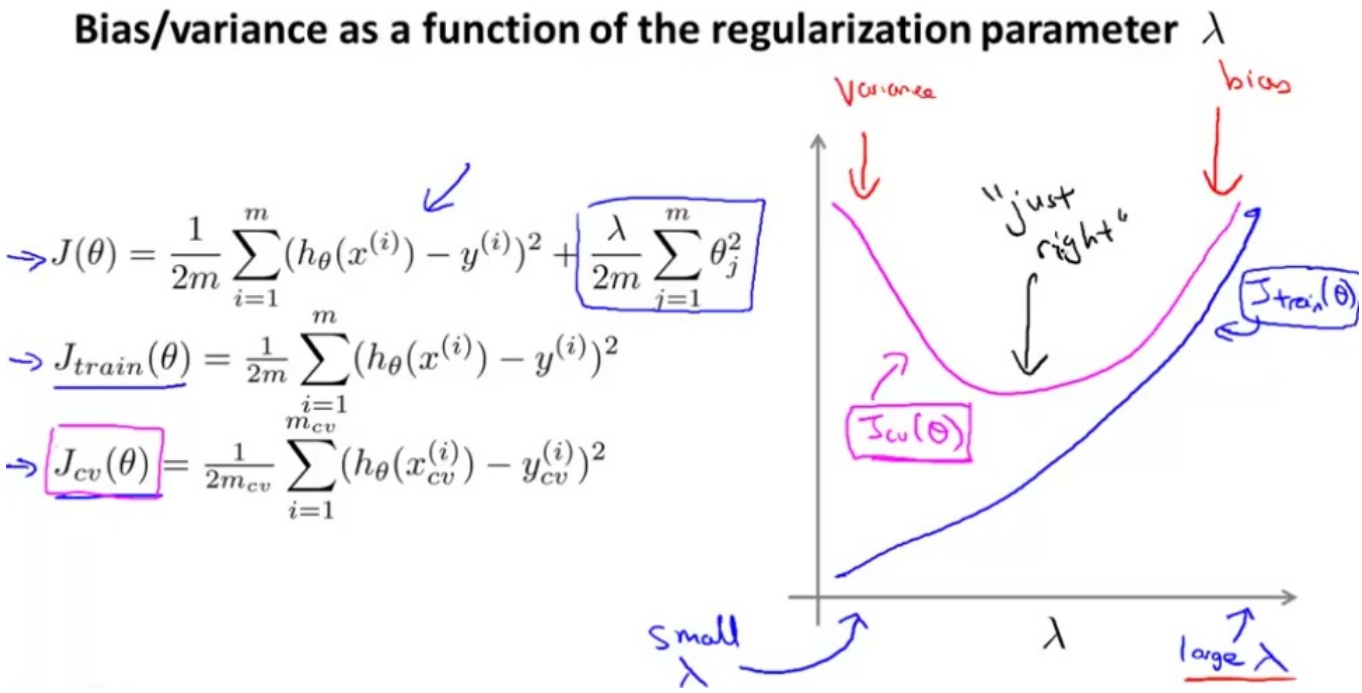
1. Try $\lambda = 0$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ → $\min_\theta J(\theta)$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $\theta^{(5)}$ → $J_{cv}(\theta^{(5)})$

⋮

12. Try $\lambda = 10$ → $\theta^{(12)}$ → $J_{cv}(\theta^{(12)})$

$10 \cdot 24$

Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$



随这λ的增大$J(\theta)$会对应增大，因为最终会趋于under fitting. **在计算$J(\theta)$的时候不加$lambda$, $lambda$只在计算得到theta时使用**

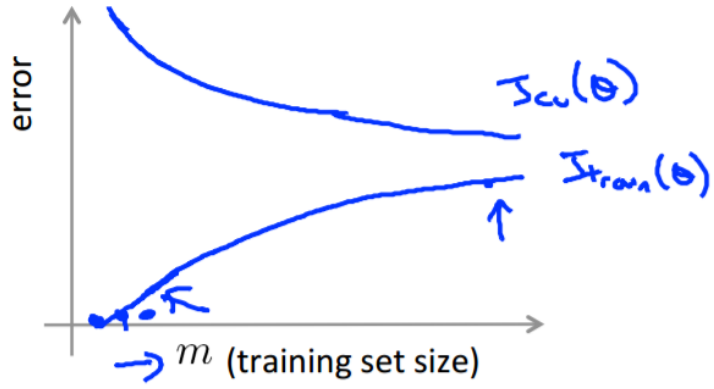
```
for i = 1: length(lambda_vec)
    lambda = lambda_vec(i);
    theta = trainLinearReg(X, y, lambda);
    error_train(i) =  linearRegCostFunction(X, y, theta, 0);
    error_val(i) = linearRegCostFunction(Xval, yval, theta, 0);
end
```

2.3 Learning curves

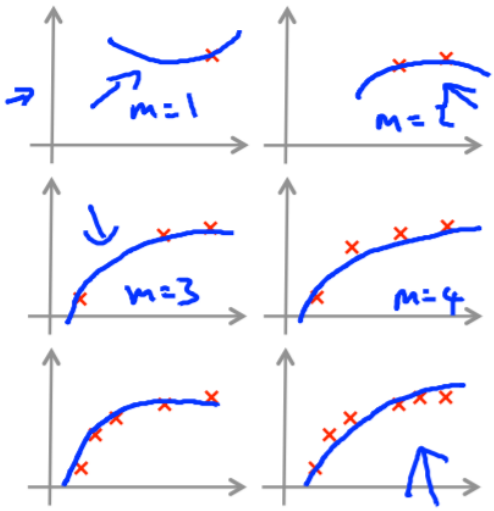
Learning curves

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

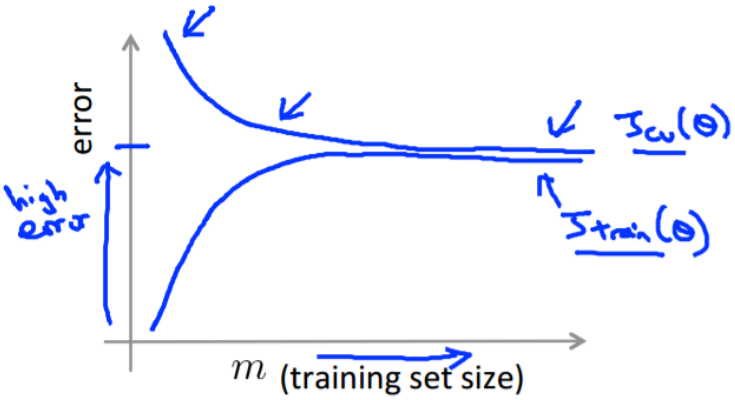
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



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

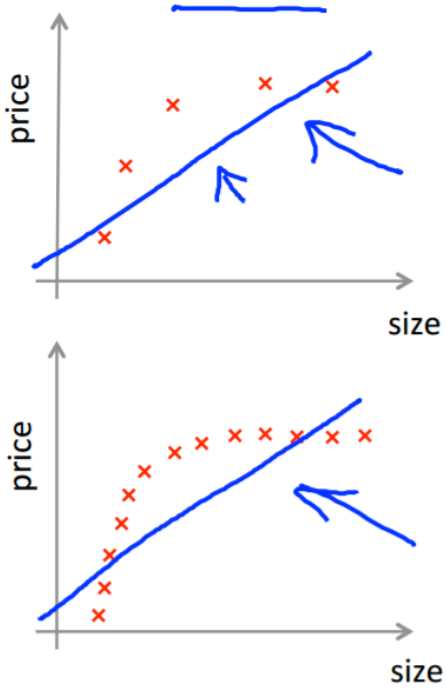


High bias

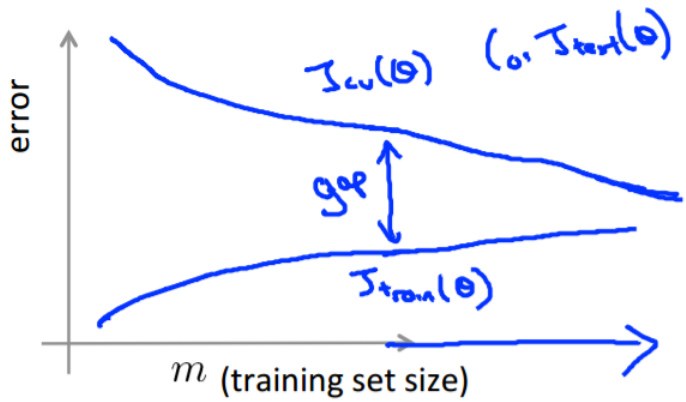


If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

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

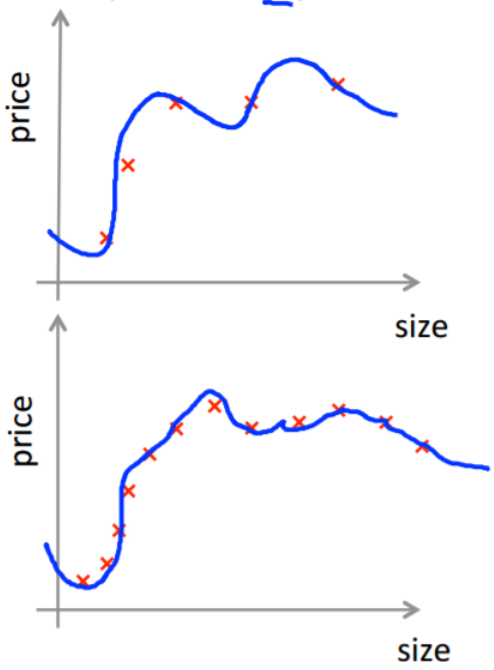


High variance



If a learning algorithm is suffering from high variance, getting more training data is likely to help.

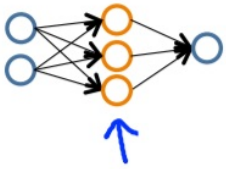
$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)



对于neural network而言， 增大λ可以有效的解决overfitting的问题, 对于neural network而言最大的问题是计算成本

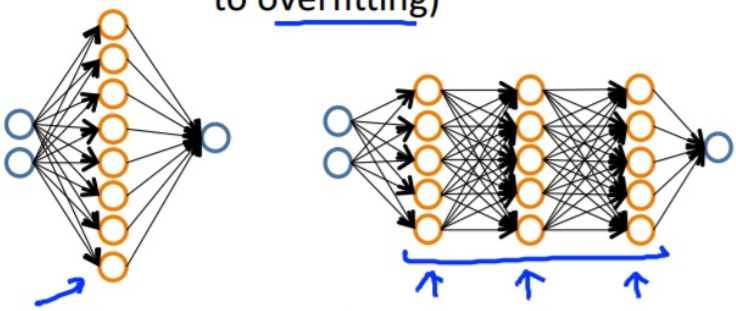
Neural networks and overfitting

"Small" neural network
(fewer parameters; more prone to underfitting)



Computationally cheaper

"Large" neural network
(more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

Small Neural Network --> fewer parameters --> underfitting
Large Neural Network --> more parameters -->overfitting
+ need to address overfitting

3. Machine Learning System Design

6.2.1 降低误差的几种方法

+ Collect more data;
+ Develop more sophiscated features;
+ Develop more sophiscated algorithms;

6.2.2 Error Analysis

+ Start with a simple algorithm: implement the model;
+ Plot learning curves : decide if more data/features may help;
+ Error analysis: test the model via CV and detect the shortcoming of the system ;

6.2.3 Error Metrics for Skewed Classes

+ P/N为预测结果，T/F为预测结果是否与实际相同

Error Matrix	**Actual 1**	**Actual 0**
Predicted 1	TP	FP
Predicted 0	FN	TN

- 几个指标:
  - Precision:预测值为1但实际值不确定, 这里求实际值也为1的比例
    - e.g. 诊断都患癌症, 求其中真正患癌症的比例

      $$Precision = \frac{TP}{TP+FP}$$
  - Recall(Sensitivity): 实际值为1,求其中预测值也为1的比例
    - e.g. 实际都患癌症, 求其中被诊断出来的比例

      $$Recall = \frac{TP}{TP+FB}$$
  - Specificity:实际值为0, 求其中预测结果也是0的比例

    $$Specificity = \frac{TN}{TN+FP}$$
  - Accuracy: 全体样本中被正确判断的比例

    $$Accuracy = \frac{TP+FN}{Total}$$
- Trading Off Precision and Recall
  - 确诊癌症需要有十足把握才决策:
    - high precision , low recall;
    - $h_\theta(x)$很高才会判定为1;
  - 避免癌症患者未被诊断:

        - high recall , low precision;
        - $h_\theta(x)$很低就会配判定为1；
    - 使用F score评估精确度和召回率
      $F = \frac{2PR}{TP+FB}$
    - 不使用average的原因: 若实际1/0的比例相差悬殊，average就会不合理
- Data for ML
    - 基于low bias algorithms 建立的模型适合大数据量 --> 不容易overfit
    - 若模型本身过于简单, 增加数据量并不能取得明显效果
    - 增大数据量对以下情况有帮助：
        - Features x contain sufficient information to predict y accurately(一个专家能否根据这些信息做出正确的判断或预测)
        - Train a learning algorithm with a large number of parameters (that is able to learn/represent fairly complex functions)
    - Threshold和Precision/Recall的关系：
        - Precision越高越不容易被判定为1
        - Recall越高越容易被判定为1

SVM

1. Model of SVM

SVM支持向量机是从logistic regression变形得到的
logistic regressioin的activation functioin 如下
$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$
obviously，当y=1且希望$h_\theta(x) \approx 1$, 则$\theta^T x$远大于0；同样，当y=0且希望$h_\theta(x) \approx 1$, 则$\theta^T x$远大于0

由于常数项的乘除不会改变极值点，即不会对θ的取值造成影响，因此可以通过将logistic function的cost function

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

乘上$\frac{m}{\lambda}$取$c = \frac{1}{\lambda}$可以得到support vector machine的cost function

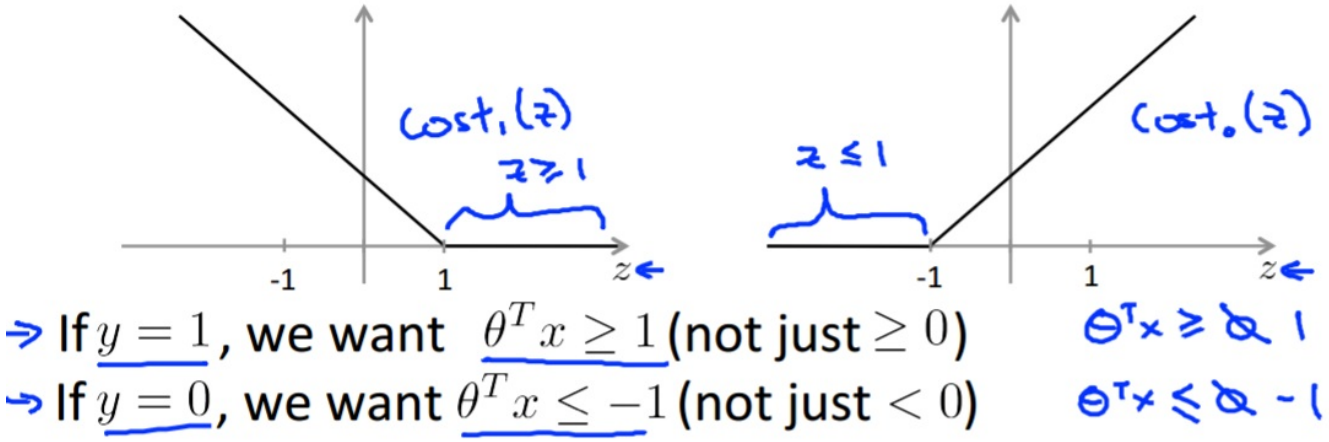
$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)})+(1-y^{(i)})cost_0(\theta^T x^{(i)})\right]+\frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

这里的两个cost函数的含义如下：

$$cost_1(\theta^T x^{(i)}) = -log(h_\theta(x^{(i)}))$$

$$cost_0(\theta^T x^{(i)}) = -log(1-h_\theta(x^{(i)}))$$

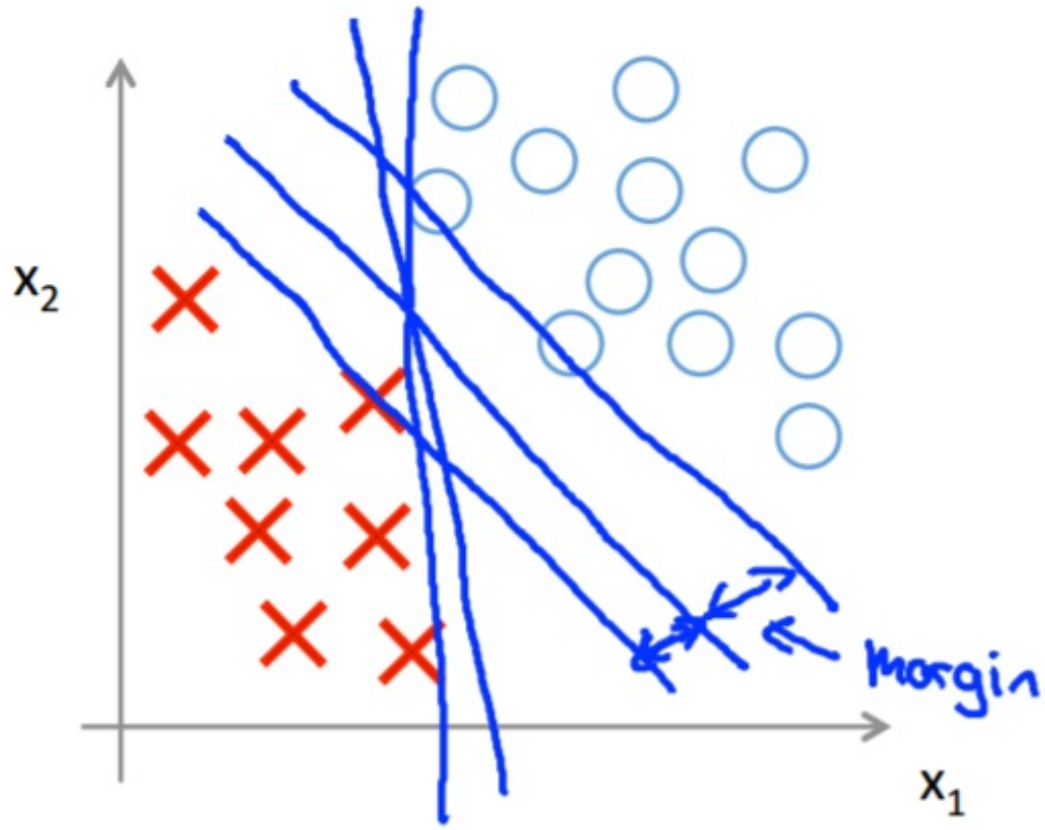
在SVM中对cost function进行修正得到新的cost function图像如下



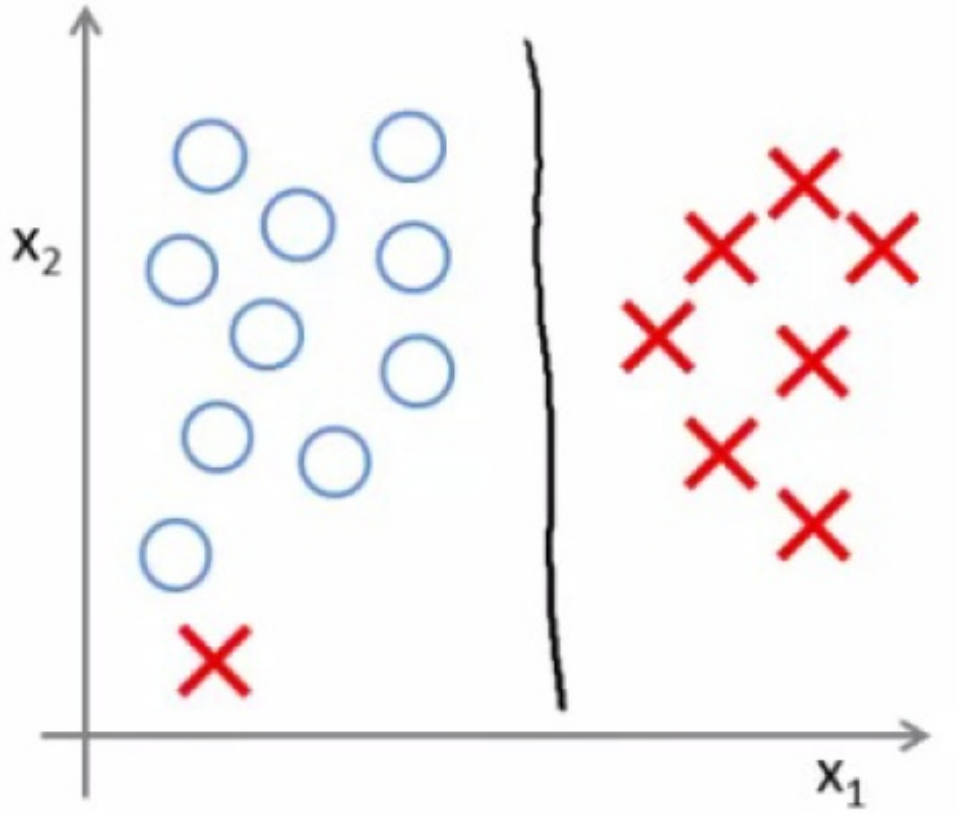
在SVM的cost function中C的取值应该要适当的小，如果C的取值过大Margin将会减小，即易受极端值的影响

2 Large Margin

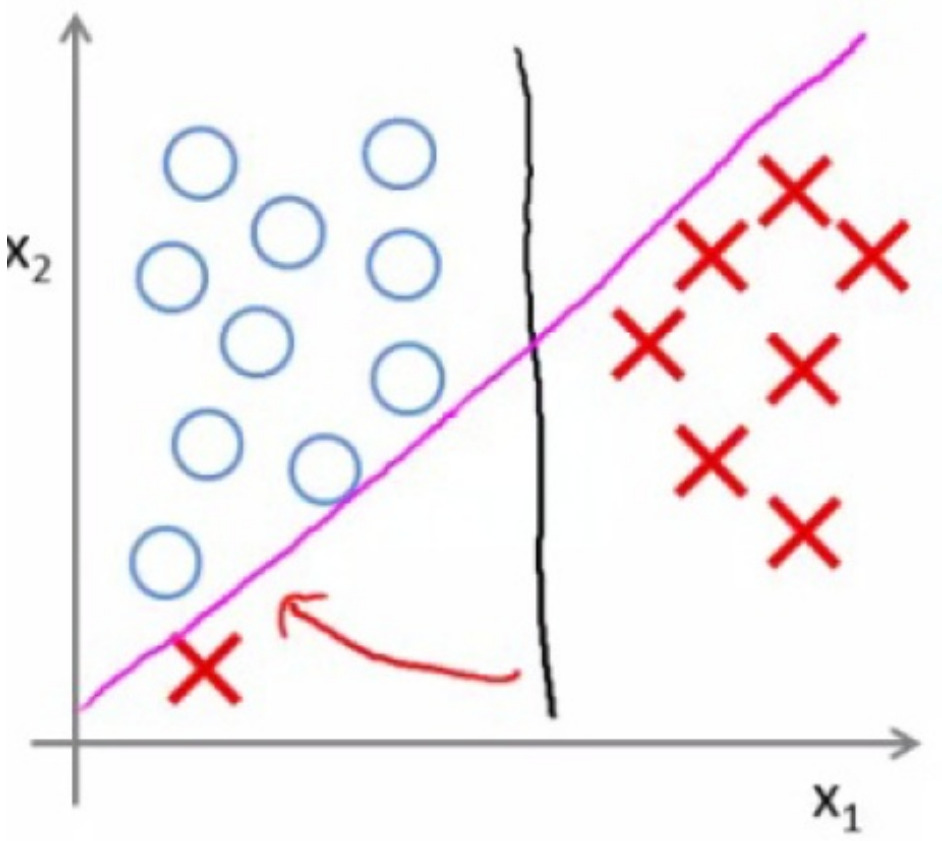
- 为了实现之前$\Theta^T x > 1, \Theta^T x < -1$cost为0的目标, SVM决策边界设置为
  - if y = 1, then $\Theta^T x > 1$(not just ≥ 0)
  - if y = 0, then $\Theta^T x < -1$(not just ≤ -1)
- margin: 两条边界线距离中间分界线的距离



- large margin

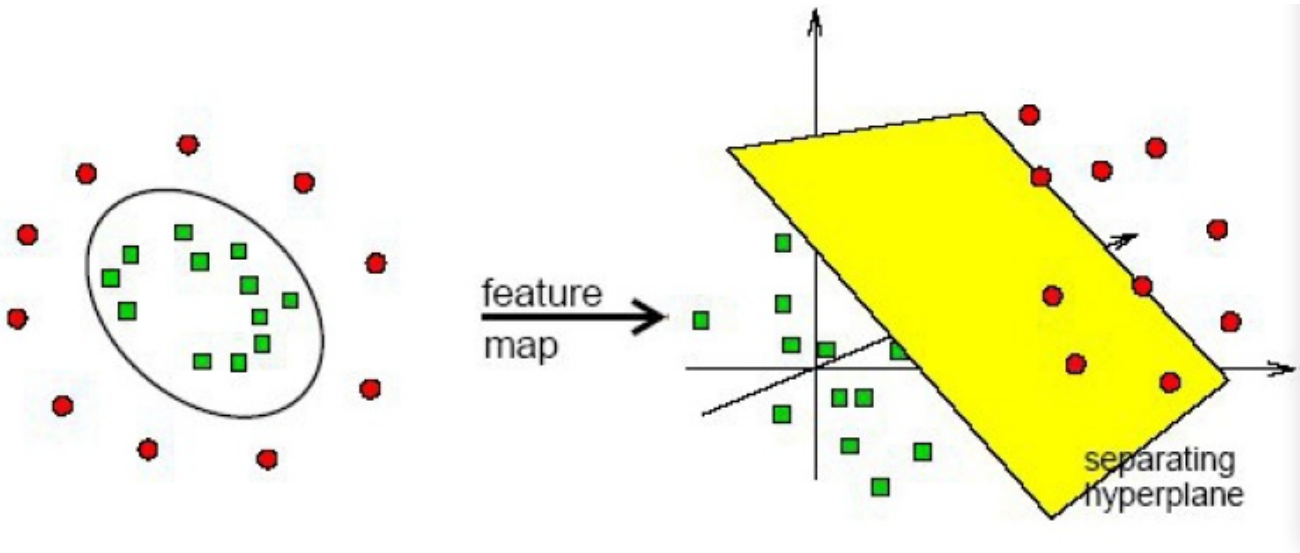


- small margin



3 kernel：

- 低维空间线性不可分的样本点再高维空间会变得线性可分
- Separation may be easier in higher dimensions



3.1 高斯核函数

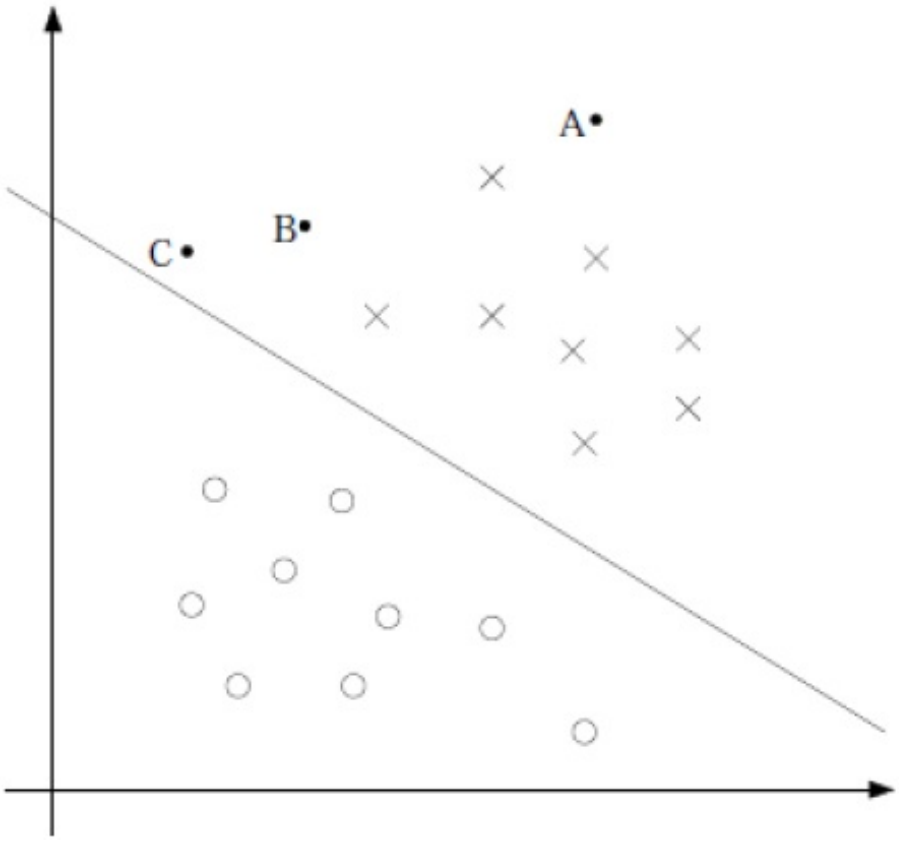
- 使用以下公式计算与landmarks $l^{(i)}$ 的相似度

$f_i = similarity(x, l^{(i)}) = exp(-\frac{||x-l^{(i)}||^2}{2\sigma^2})$

- $||x - l^{(i)}||^2 = \sum_{j=1}^{m} (x_j - l_j^{(i)})^2$ 欧式距离
- σ可以变化以得到不同变化幅度的形状
  - σ越大，var越小，变化越平缓

- if $x \approx l^{(i)} \rightarrow ||x - l^{(i)}||^2 \approx 0 \rightarrow f_i = 1$
- if $x \neq l^{(i)} \rightarrow ||x - l^{(i)}||^2 >> 0 \rightarrow f_i = 0$
- 根据这些相似度计算结果, 可将$h_\theta(x)$改写为:
  $h_\theta(x) = \theta^T f$
- Cost function最终可以被改写为
  $c\sum_{i=1}^{m} (y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)})) + \frac{1}{2}\sum_{j=1}^{m} \theta_j^2$

3.2 Example of using kernel



- 给定$\theta = [-0.5,\ 1,\ 1]$构成边界
- 假设C(x)离B(l)很近，则：
  - $f_1 \rightarrow 0$
  - $f_2 \rightarrow 1$

    - $\theta_0 + \theta_1 f_1 + \theta_2 f_2 \geq 0 \rightarrow$ Output 1
-
    - 假设A(x)离B(l)很近，则：
    - $f_1 \rightarrow 0$
    - $f_2 \rightarrow 1$
    - $\theta_0 + \theta_1 f_1 + \theta_2 f_2 \leq 0 \rightarrow$ Output 0

3.3 Kernel trick的步骤:

- Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(1)}, y^{(1)})$
- choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$
- Given training example $x^{(i)} \in \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
    - $f = [f_0 = 1, f_1, f_2, \ldots, f_m]$
    - $f_1^{(i)} = similarity(x^{(i)}, l^{(1)})$
    - $f_2^{(i)} = similarity(x^{(i)}, l^{(2)})$
    - ...
    - $f_i^{(i)} = similarity(x^{(i)}, l^{(i)}) = 1$
    - $f_m^{(i)} = similarity(x^{(i)}, l^{(m)})$
- 最后将$x^{(i)}$赋值为 $f^{(i)} = [f_0^{(i)} = 1, f_1^{(i)}, \ldots, f_m^{(i)}]$作为新特征向量, 代入cost function

$$J(\theta) = C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^{n} \theta_j^2$$

    - Summary:
    - 使用核函数将原始数据x变换到另一个特征空间f, 令其线性可分;
    - 在新特征空间f中使用SVM进行学习分类

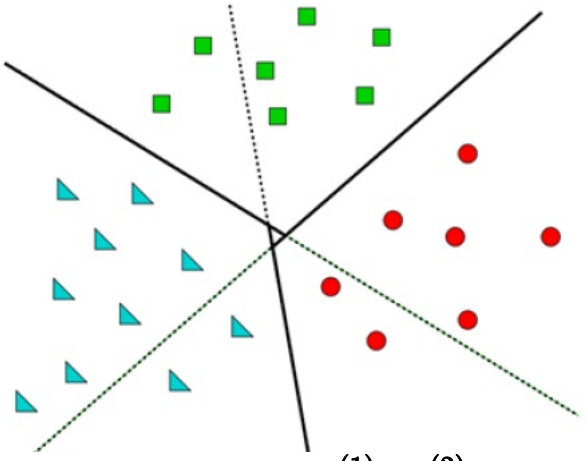
3.4 Parameters in SVM

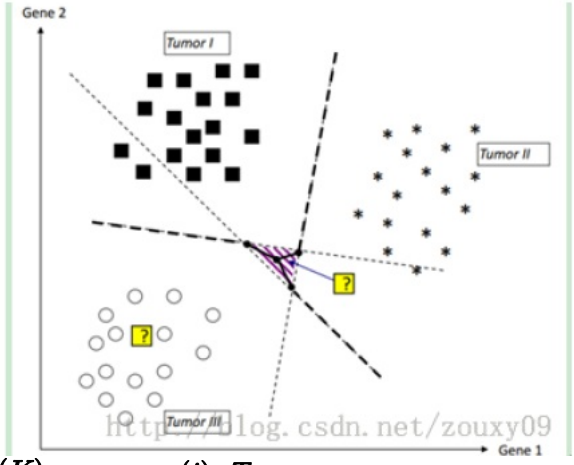
- $C = \frac{1}{\lambda}$: 与之前的规律刚好相反
    - Large C $\rightarrow$ complex model $\rightarrow$ high variance, low bias
    - Small C $\rightarrow$ simple model $\rightarrow$ low variance, high bias
- 高斯核函数中的σ^2:
    - Large σ^2 $\rightarrow$ smoother $\rightarrow$ low variance, high bias
    - Small σ^2 $\rightarrow$ steeper $\rightarrow$ high variance, low bias

4. SVM in practice

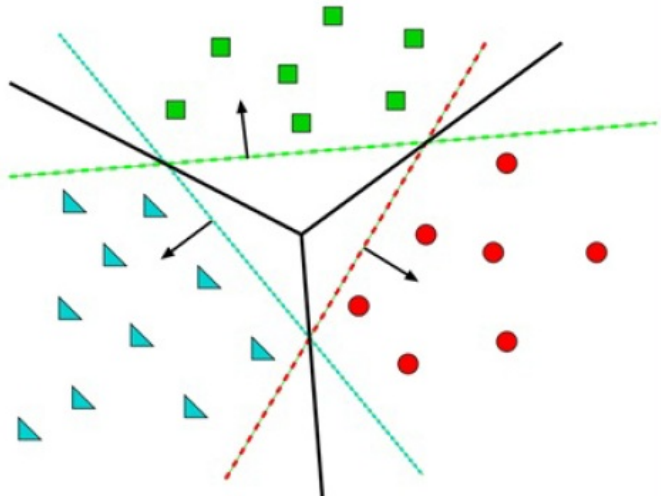
- SVM使用步骤:

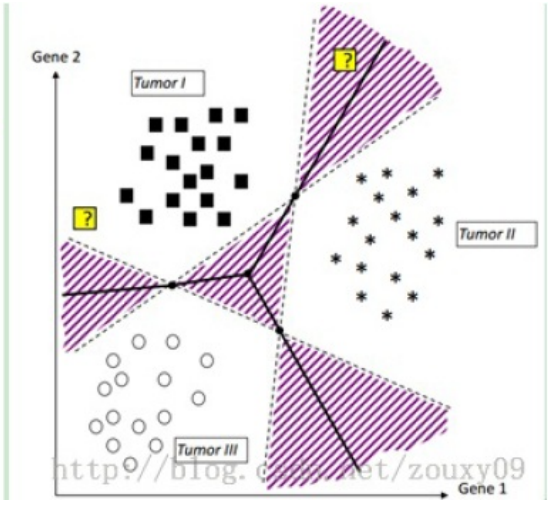
    - 选择参数C:
    - 选择核函数:
        - 样本量m小而特征量n多时选择线性分类器(无核)
        - 样本量m大而特征量n少时(线性不可分)选择高斯核
            - 需要选择σ^2
            - 需要进行featuring scaling
    - 使用训练集和CV集对C和核参数进行训练
- Multi-class Classification
    - 之前学习的为OVO: one versus one
        - Many SVM packages already have built-in mul-Iclass classification functionality.





    - OVA(one versus all): 得到$\Theta^{(1)}, \Theta^{(2)}, \ldots, \Theta^{(K)}$, 选取$(\Theta^{(i)})^T x$最大的class i





- SVM in R: `svm() in e1071 package`

```
svmfit=svm(y~.,data=dat,
    kernel="linear", #是no kernel
```

```
    #other kernels: polynomial/radial basis/tanh
    cost=10, #tuning parameter C
    scale=FALSE,
    type='c' #classification
    )
```

- SVM in Python: `svm.SVC() in sklearn package`

```
Support Vector Classification
2. from sklearn import svm
3. clf = svm.SVC(gamma=0.001, C=100.)
```

5.LR VS SVM

- declare :
  - n: number of features
  - m: number of training examples

Condition	SVM	LR
$n > m$	Linear kernel	√
small n, intermediate m	Gaussian kernel	
$n \ll m$	add more features, then linear kernel	add more features, then LR
classes are separable	√	
need to estimate probablities		√
nonlinear boundaries	√	

- Example:
  - n > m: n = 10000, m = 10-1000
  - samll n, intermediate m: n= 1-1000, m = 10-10000
  - n << m : n = 1-1000, m = 50000+
- Neural network适用于以上大部分情况,但速度慢且可能得到的只是局部最优(black box)

K-Mean clustering

1. K-Means Algorithm

- 基本思路
  - 预先确定cluster number K;
  - 随机指定K个点作为cluster centroids;
  - 根据其余点同质心的接近程度进行分类，得到K个聚类;
  - 每个聚类重新指定质心;
  - 重新聚类，迭代以上步骤直至每个聚类内的点都接近质心
- 基本变量：
  - K(number of clusters)
  - Training set{$x^{(1)}, x^{(2)}, \ldots, x^{(m)}$}
    - $x^{(1)} \in R^n$ (不包括x_0)
  - Randomly initialize K centroids:{$\mu_1, \mu_2, \ldots, \mu_k$}；
- Declare：
  - $c^{(i)}$ = 样本$x^{(i)}$当前被分到的cluster (1,2,...,K)的index
  - μ_k: cluster centroid k ($\mu_k \in R^n$:)
  - $\mu_{c^{(i)}}$: 样本$x^{(i)}$被分到的cluster (1,2,...,K)的index
  - Example:
    - $x^{(i)}$ = 5
    - $c^{(i)}$ = 5
    - $\mu_{c^{(i)}}$ = μ_5

Note: $||x^{(i)} - \mu_{c^{(i)}}||$为点$x^{(i)}$到簇中心$\mu_{c^{(i)}}$的距离

- **K-means optimization objective**:
  $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} ||x^{(i)} - \mu_{c^{(i)}}||^2$
  $min_{c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_k} J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_k)$
  最小化距离偏差的总和
- 算法思路
  随机初始化k个cluster centroids $\mu_1, \mu_2, \ldots \mu_k \in R^n$
  Repeat{
    for i = 1:m将每一个样本根据距离归类给一个簇中心
      $c^{(i)}$:= index(from 1 to K) of cluster centroid closest to $x^{(i)}$

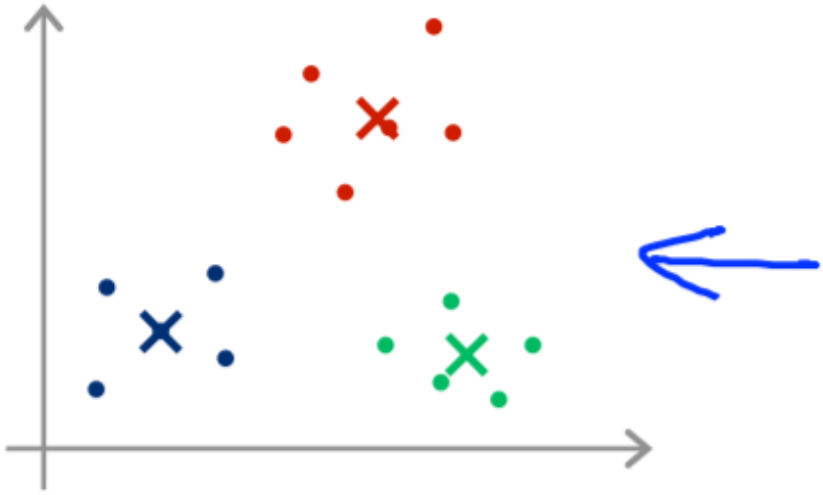
end
for k = 1:K根据分类的结果重新计算每个簇的簇中心
μ_k:= average (mean) of points assigned to cluster
end
}重复上述过程知道结果收敛为止

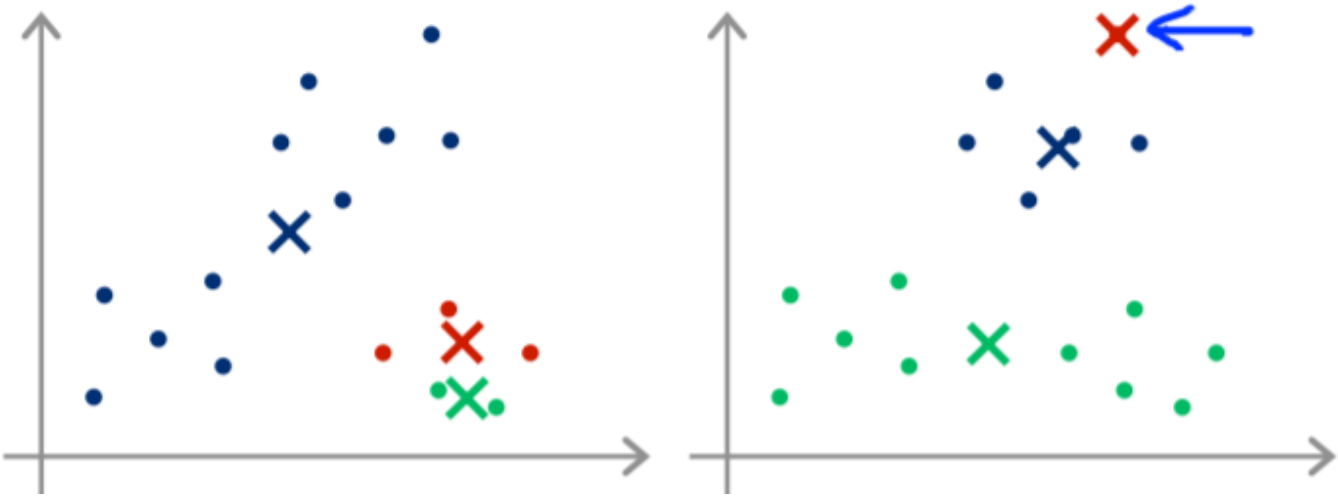
- 注意若存在空聚类,可以重新随机初始化或者直接删掉该聚类(K--)

2. 簇中心的初始化

- 满足簇的个数小于样本数K < m
- 从训练集中随机选择K个样本， 将这k个样本作为k个簇的初始化点

需要注意的是这样的初始化可能会导致求得的分类粗的最小距离为局部最优解





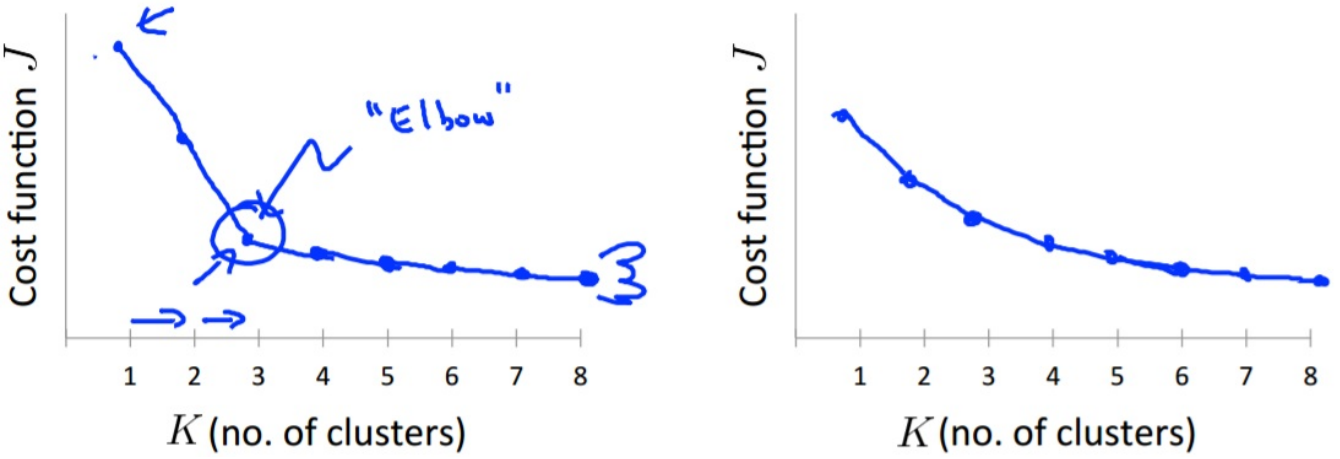
为了解决这一问题可以使用迭代多次求解的方法，迭代次数一般介于50-1000次之间

For i = 1:100{
  Ramdomly initialize K-means.
  Run K-means. Get $c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_k$.
  Compute cost function(distortion)
  $J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_k)$
}

选择cost functioin $J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_k)$最小的分类作为最终结果。这个方法只适用于K较小时最好是介于2-10之间。

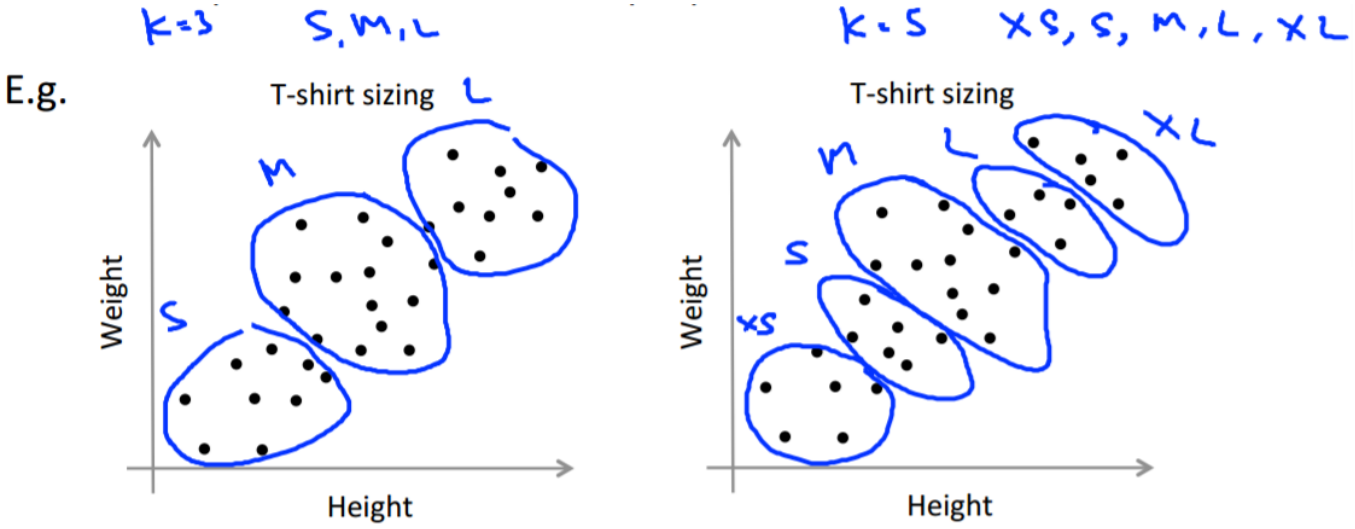
3. K的取值

方法1：



如上图所示绘制出cost function J和K的关系图选出拐点的K. 但是当情况如图右所示时则方法1失效

方法2： 根据实际需要选取合适的K



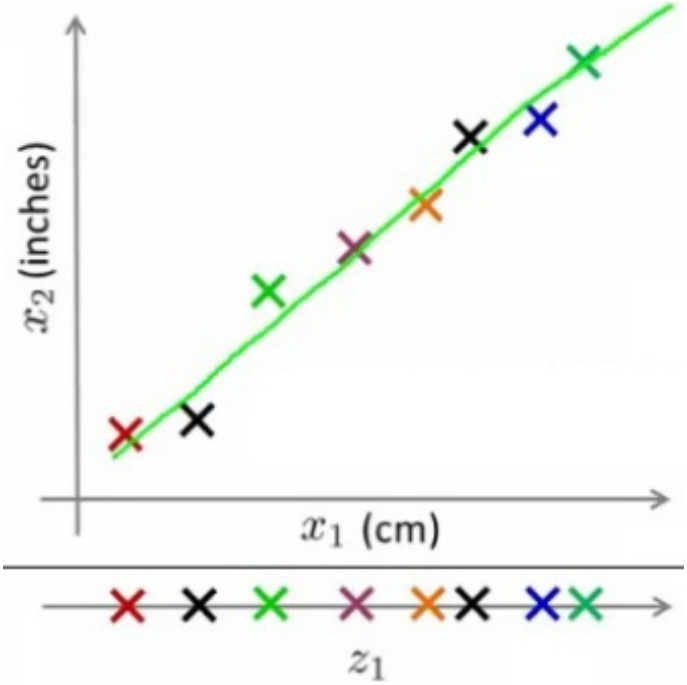
PCA

PCA主成分分析是一种常用的数据降维方法

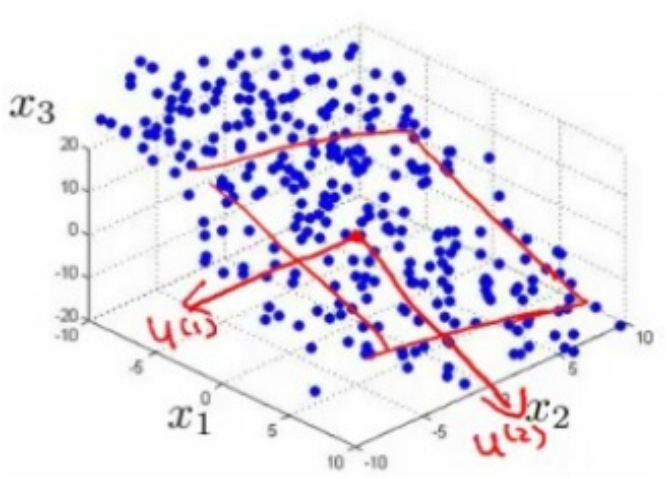
- 降低维度的目的:
  - 化简数据量, 优化算法效率
  - 使模型更易于可视化

1. PCA Problem Formulation

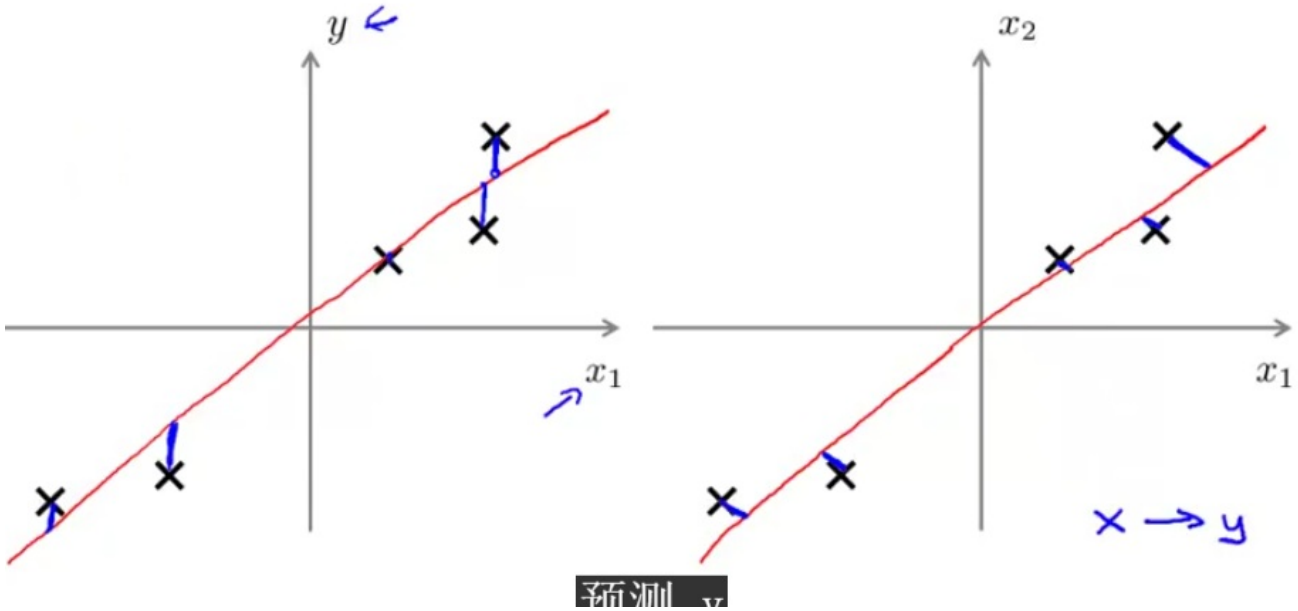
- PCA主要原理:
  - 先找方差最大的方向作为第一主成分
  - 在其法向量中寻找方差最大的作为第二主成分
  - 不断在之前所有主成分共有的法向量中寻找方差比较大的作为主成分
  - 最后得到K个主成分(K < N), 实现维度压缩
- PCA的目标: 最小化特征点到projection line的平均距离(projection error)
  - 注意得到的主成分线方向对降维没有影响(共线)
  - 将N个特征压缩到K个特征, 最小化projection error
  - 压缩后的各特征向量彼此正交
- Example：将二维数据压缩成一维



  - 首先进行feature scaling,令均值为0;
  - 使用PCA将所有特征点投射到一条直线上, 得到成分A;
  - 作该条直线的法向量,将所有点投射到该直线上, 得成分B;
  - 所有特征点到成分A的距离远小于到成分B的距离, 因此选取成分A作为第一主成分
- Example：将三维数据压缩成二维

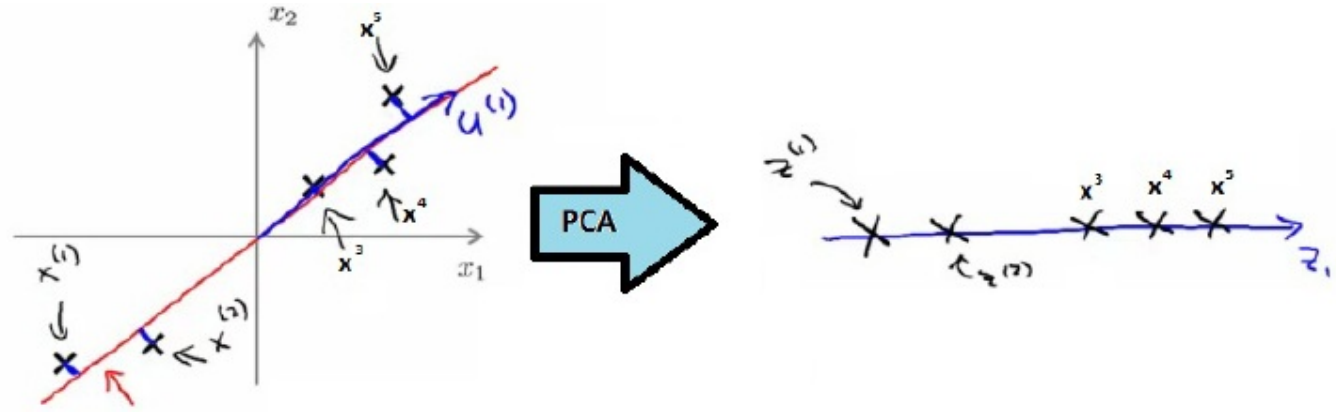


- PCA与LR的区别



  - Error
    - LR: squared error 为每个点到回归线的纵坐标距离差$y^{(i)} - h(y^{(i)})$
    - PCA:projection error为每个点到主成分线上投影点的欧氏距离 $\sqrt{\sum(Vec_A - Vec_B)^2}$ A为样本点， B为投影点
  - Axis
    - LR: $y - x$
    - PCA: $x_2 - x_1$, every y is treated equally
  - 若降维后的维度K依旧大于2, 最后得到的主成分是K条相互正交的直线
  - PCA不需要考虑x_0, 而LR需要单列出一列θ_0

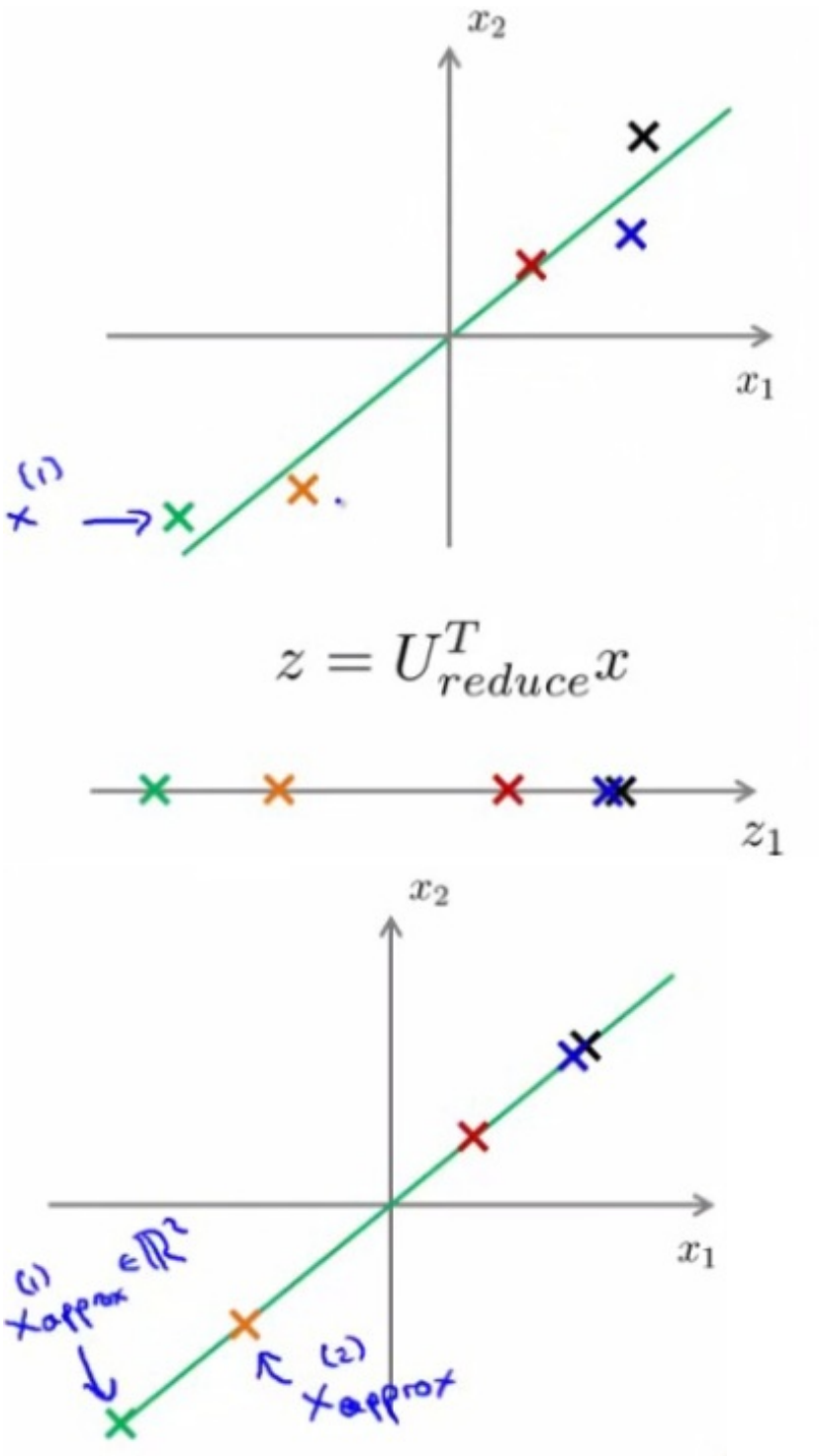
2. PCA Algorithm



- Preprocessing: feature scaling & mean normalization
  - Training set: $x^{(1)}, x^{(2)},\dots,x^{(m)}$, $x^{(i)}$ 为$n*1$维向量
  - $\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}$
  - mean normalization: x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}(necessary)
  - feature scaling: $x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{s_j}$(optional)
- PCA algorithm: $x^{(i)} \in R^2 \rightarrow z^{(i)} \in R$
  - 主要任务
    - 找到k个主成分：$u^{(1)}, u^{(2)}, \dots, u^{(i)}$
    - 将训练集投影为：$z^{(1)}, z^{(2)}, \dots, z^{(i)}$
  - Step 1:计算协方差矩阵
    - $\sum = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)})(x^{(i)})^T$
    - $(n*1)*(1*n)$
  - Step 2:计算得到协方差矩阵的特征向量
    - `[U, S, V] = svd(Sigma)`
    - SVD(singular value decomposition), 可以参考blog
  - Step 3:使用U矩阵的前k列计算z
    - $U = \{u^{(1)}, u^{(2)}, \dots, u^{(i)}\}$
    - 选取其中前k个元素作为主成分, 得到矩阵$U_{reduce, n*k} = U[:, 1:k]$
    - 计算投影后的新训练数据$z^{(i)} = U_{reduce}^T * x^{(i)}$

```
Sigma = (1/m) * XT * X; # covariance matrix
[U,S,V] = svd(Sigma); # projected directions
Ureduce = U(:,1:k); # take the first k directions
Z = X * Ureduce; # compute the projected data points
```

3. Reconstruction from Compressed Representation



- 将压缩后的k维数据返回到原来的n维
- $x_{approx}^{(i)} \approx U_{reduce} * z^{(i)}$
  - $(b * k) * (k * 1) \rightarrow n * 1$
- 降维后再增维还原，存在一定误差

- 还原后的增维点仍保持在PC线上, 而原数据点则不一定

4. Choosing the Number of Principle Components#K

- 根据(A)Average squared projection error: $\frac{1}{m}\sum_{i=1}^{k}||x^{(i)}-x_{approx}^{(i)}||^2$
- 以及(T)Total var: $\frac{1}{m}\sum_{i=1}^{k}||x^{(i)}||^2$
- 选取令$\frac{A}{T}$尽可能小的K: bao99%的variance(方差贡献率): $\frac{\frac{1}{m}\sum_{i=1}^{k}||x^{(i)}-x_{approx}^{(i)}||^2}{\frac{1}{m}\sum_{i=1}^{k}||x^{(i)}||^2}\leq 0.01$
- 用于选择k的算法:
  - Perform PCA with k=1;
  - Compute U_{reduce},z,x;
  - Check if $\frac{A}{T}\leq 0.01$;
  - If not, perform PCA with k=2,3,...
- 之前在`[U,S,V] = svd(Sigma)`中得到的S矩阵可用于简化
  - $S=diag[S_11,S_22,...S_nn]$
  - 可将$\frac{A}{T}$化简为: $\frac{\sum_{i=1}^{k}S_{ii}}{\sum_{i=1}^{n}S_{ii}}\geq 0.99$

5. Advice for Applying PCA

- PCA的常用用途为提升监督学习的效率: 消除无关的特征维度
  - 注意PCA过程仅能使用训练集进行压缩，得到$z^{(i)}$的可以在CV及测试集中使用
- 压缩数据: 减少内存及硬盘使用量
- 令数据更易于可视化: 压缩到二维或者三维
- PCA的不合理用途: prevent overfitting
  - 可以这样做, 但是不推荐, 应该使用regularization
  - 原因: 对于监督学习，regularization可以根据y进行调整
- 注意PCA不是机器学习模型的必要组成部分,设计ML系统前先考虑能否不用
  - 先用原数据设计,得不到期望结果时再考虑PCA

Anomaly detection

1.Problem Motivation

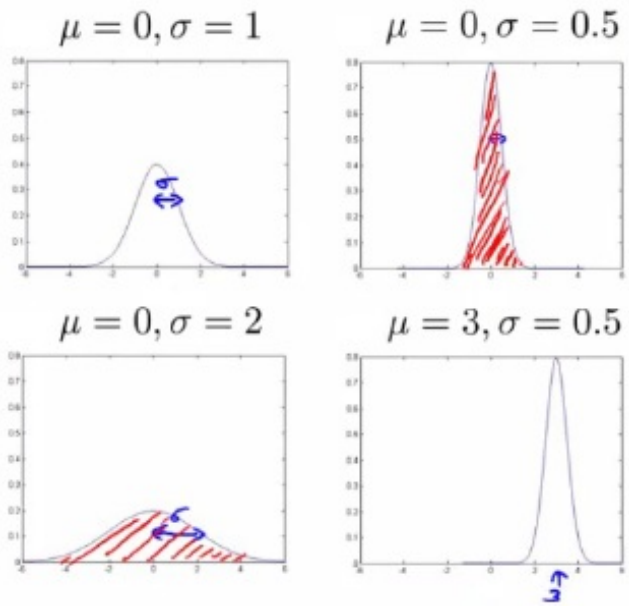
- 新样本出现在训练集样本越集中(密度大)的地方，为异常值的可能性越小
- Example： fraud detection
  - $x^{(i)}$: features of user i's activities
  - Model $p(x)$ from data
  - If $p(x) < \epsilon$: unusual users
  - Increase ϵ : high recall,low precision, 更容易被判定为不正常
  - Decrease ϵ : high precision,low recall, 十足把握才判断

2.Gaussian Distribution

- 高斯分布就是正态分布$N(\mu, \sigma^2)$
  - mean: $\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$
  - variance: $\sigma^2 = \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu)^2$
  - σ : standard deviation

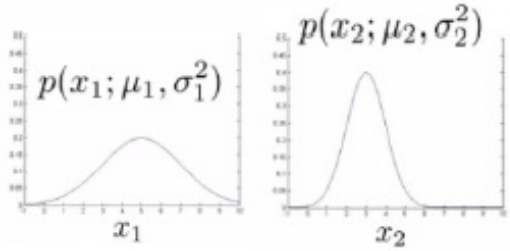
    $P(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$
    - 注意高斯分布函数与之前SVM中的高斯核不同

      Gaussian Kernel = $e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$
    - 高斯核中σ的变化趋势与高斯分布刚好相反
- 高斯分布与参数μ, σ^2的关系

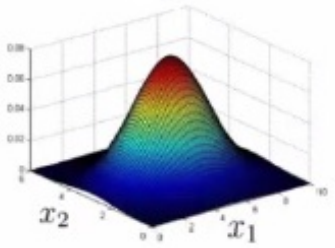


3. Algorithm

- Density estimation
  - Training set $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$:
  - $p(x) = \prod^n i = 1 p(x_j; \mu_j, \sigma_j^2)$: 不同维度密度函数的乘积
  - if $p(x) < \epsilon$, it's anomalous
- Example:
  - 可能存在异常的特征:
    - $x_1, \mu = 5, \sigma = 2$
    - $x_2, \mu = 3, \sigma = 1$
  - 计算x_1, x_2的高斯分布:



  - 在三维图中表述如下,其中高度为密度函数:



  - 使用两个数据点进行测试:
    - $p(x_{test}^1 = 0.436)$ 为正常值的可能性43.6%
      - Normal
    - $p(x_{test}^2 = 0.0021)$ 为正常值的可能性为0.21%
      - Anomalous

4. Developing and Evaluating an Anomaly Detection

- 典型的数据集分配:
  - Training set: 6000 good to compute $p(x)$
  - CV set: 2000个 good(y=0) 和 10个 anomalous(y=1)
  - Test set: 2000 good 和 10 anomalous
- Algorithm evaluation

- Fit model $p(x)$ on training set;
- For a CV or test example x:
  - if $p(x) < \epsilon \rightarrow anomaly \rightarrow y = 1$
  - if $p(x) \geq \epsilon \rightarrow anomaly \rightarrow y = 0$
- Possible evaluation metrics:
  - TP,TN,FP,FN
  - Precision/Recall
  - F_1 score
- Choose ϵ : CV

5. Anomaly Detection vs Supervised Learning

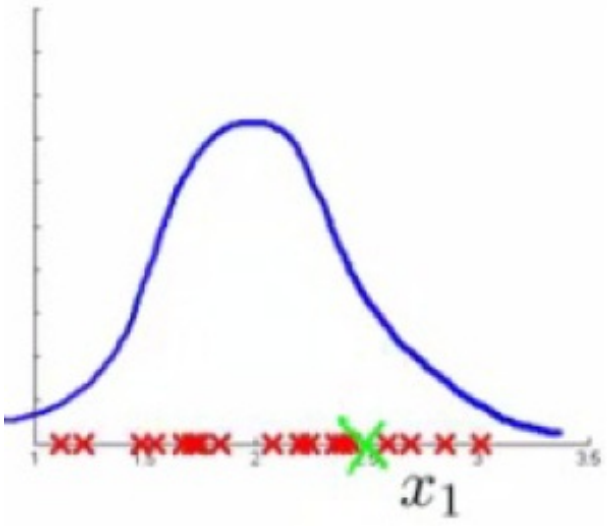
Item	Anomaly Detection	Supervised Learning
Proportion	Negative数据远多于Positive	Negative和Positive 差不多,或者没什么严格要求
Type	多种异常, 很难学习并预测	可以从训练集预测得到Positive数据的类型
Application	检测异常,系统监控	垃圾分类, 天气预测

- Example:
  - Anomaly detection
    - Fraud detection
    - Manufacture
    - Monitoring
  - Supervised learning
    - Email spam classification
    - Weather prediction
    - Cancer classification

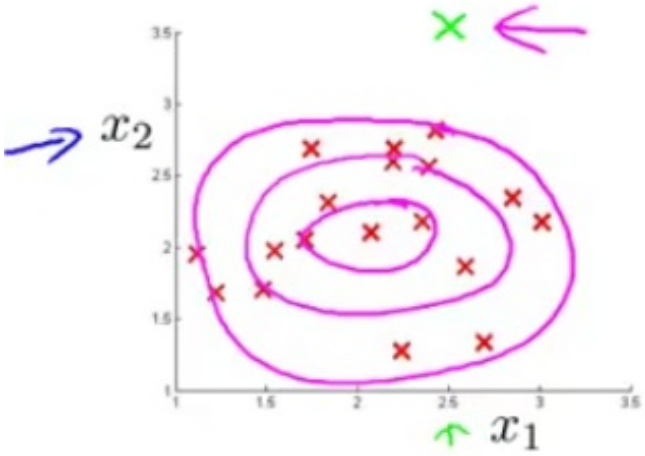
6. Choosing What Features to Use

- Non-gaussian features: make them like gaussian
  - $x \rightarrow log(x + c)$
  - $x \rightarrow x^c$
- Error analysis for anamoly detection:

- Goal: more normal data x(y=0),less anomalous data x(y=1)
- Most conditions: p(x) is large for both normal and anomalous examples

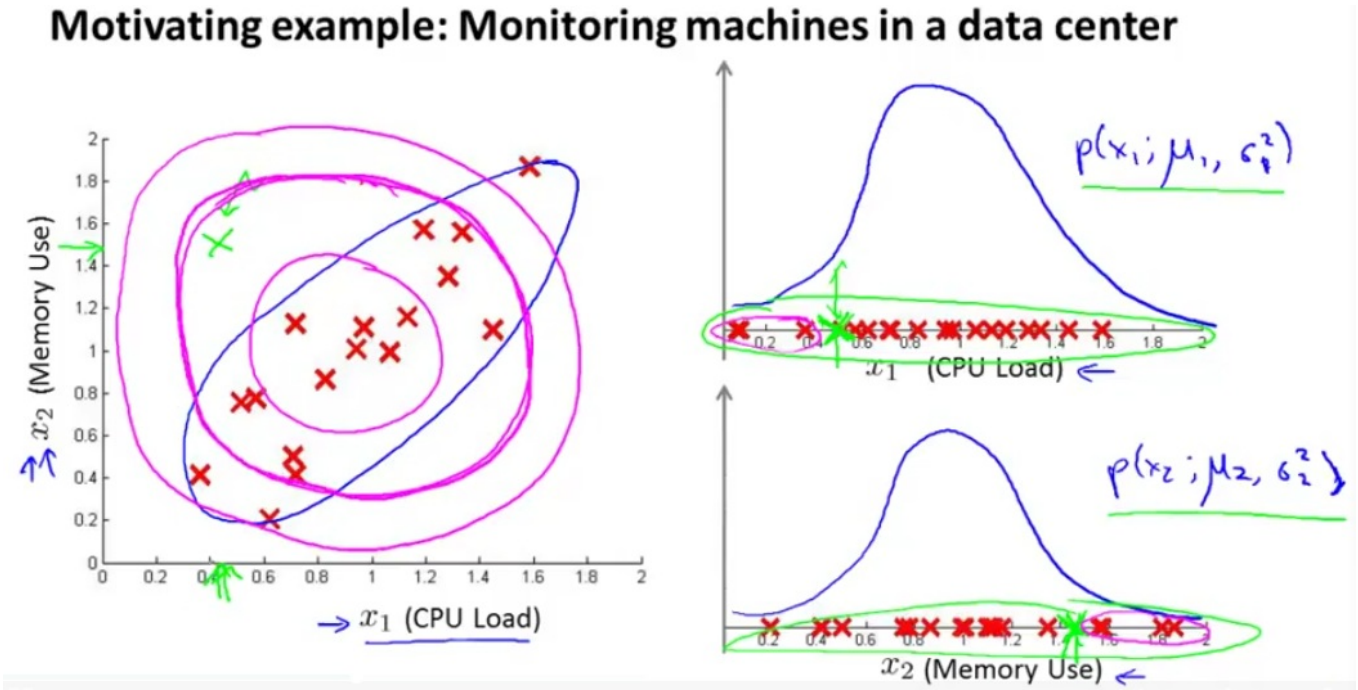


- 成因: 维度过低导致异常跟正常情况混在一起,难以区分
- 解决措施：增加维度，例如引入多项式



7. Multivariate Gaussian Distribution

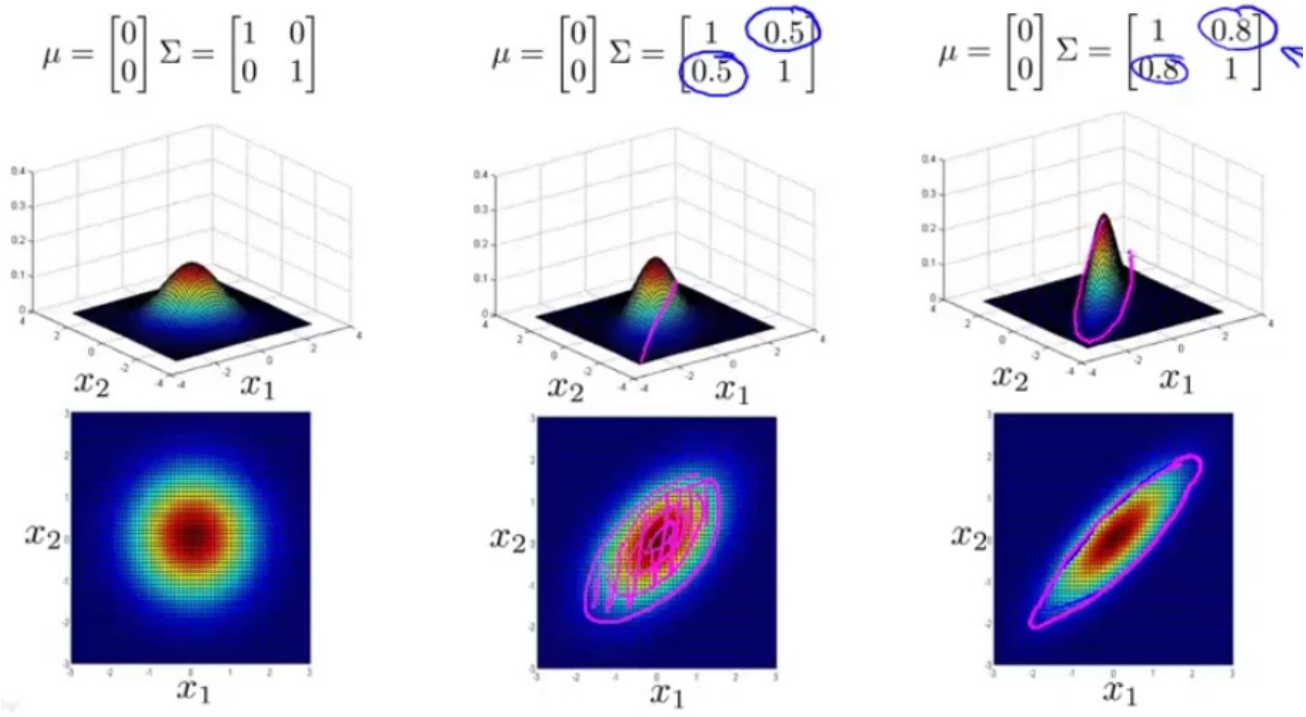
当出现以下情况时， 我们需要使用MUltivariate Gaussian Distribution来进行anomaly dectection

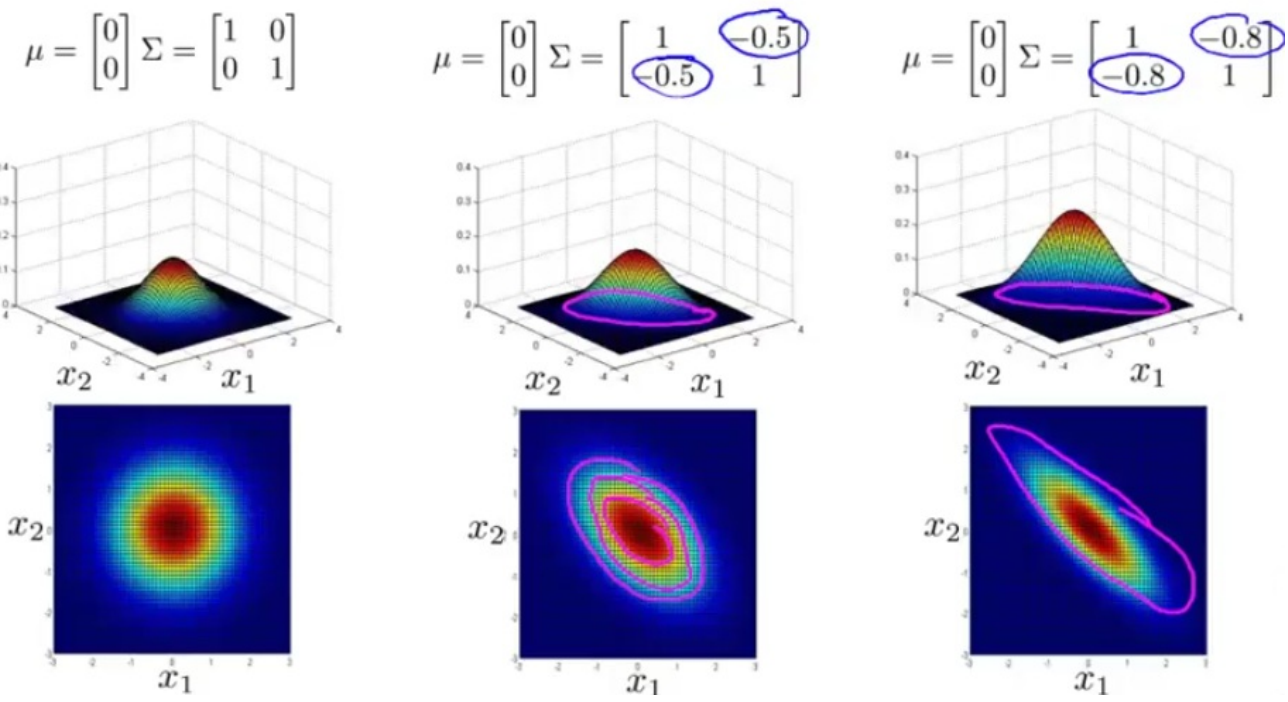


- 使用以下公式一次性获得所有$p(x)$

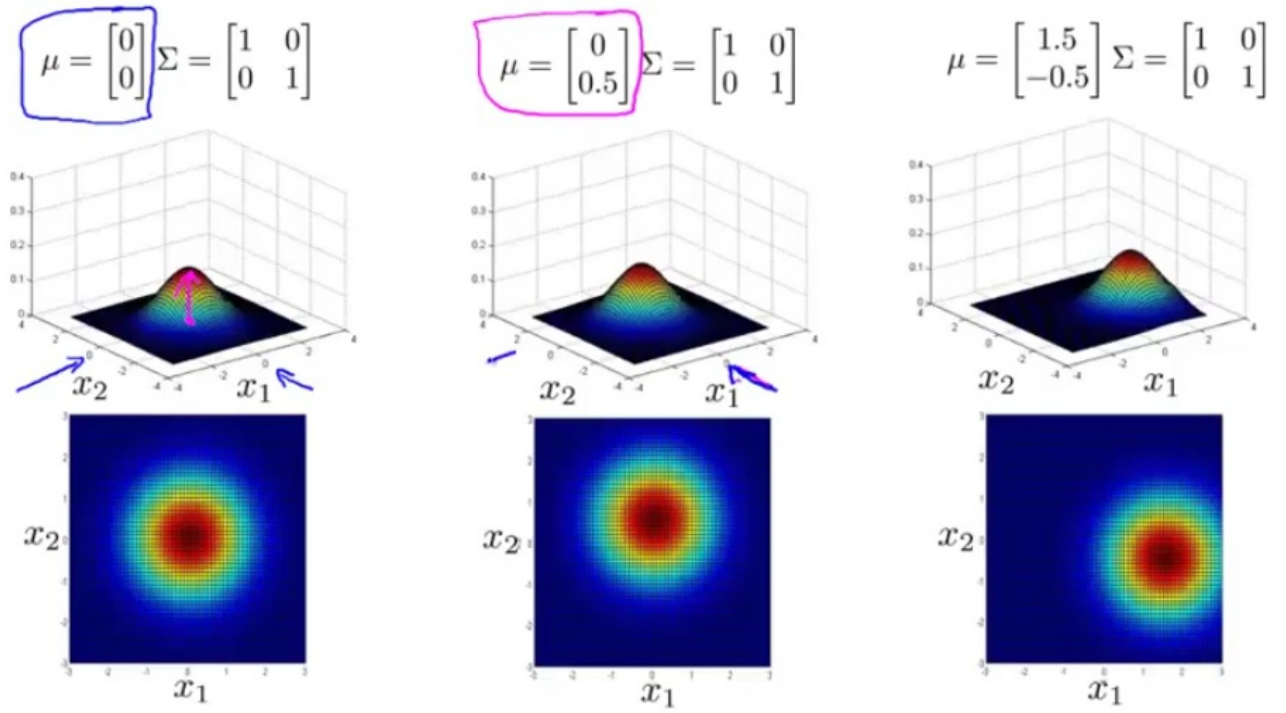
$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} exp(-1/2(x-\mu)^T\Sigma^{-1}(x-\mu))$$

  - $\sum = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)}-\mu)(x^{(i)}-\mu)^T$ 是协方差矩阵
  - $\mu = \frac{1}{m}\sum_{i=1}^{m}x^{(i)}$ 是均值
- 调整密度图形的形状





- 调整峰的位置



8. Original model VS Multivariate Gaussian

Item	Original model	Multivariate Gaussian
$p(x)$	$\prod_{j=1}^{n} p(x_j;\mu_j,\sigma_j^2)$	$P(x;\mu,\Sigma)$
检测异常方式	手动创建特征检测	自动捕捉特征间的联系
是否易于构建	易于构建	不易构建
对样本量的要求	无要求	样本量m必须大于特征量n,否则Σ不可逆

- 一般用MG需要$m \geq 10n$

9. 注意事项

- Anomaly detection的适用情况;
- Increase/decrease ε对异常检测的影响;
- 异常检测与监督学习的差别;
- 注意训练模型p(x)时不使用任何异常数据;

Reconmmender System

1. Problem Formulation

- Example:电影评分系统
  - $n_u = 4$: 用户数量
  - $n_m = 5$: 电影数量
  - $r(i,j) = 1$: 用户j对电影i进行了评分
  - y(i, j): 用户j对电影i的评分,只有$r(i,j)$为真时才会出现
  - $x^{(i)}$: 电影i的特征向量
  - $\theta^{(j)}$: 用户j的参数向量

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9

2. Content Based Recommendations

- 电影有两个特征, Romance(x_1), Action(x_2), 这里增加一个额外特征x_0
  - 因此电影i的特征向量$x_{(i)}$可表示为$3*1$向量

    $$x_{(i)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$$
  - Recommending process:
    - For each user j , learn a parameter $\theta^{(j)} \in R^3$
    - Predict j as rating movie i with $(\theta^{(j)})^T x^{(i)}$
    - 这里$\theta^{(j)}$指用户j对两类电影的爱好
    - 注意θ不那么容易看出来，需要通过学习得到
- Example: 预测user 1对movie 3的评分
  - $\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$
  - $x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$
  - 预测评分为:

    $(\theta^{(1)})^T x^{(3)} = (0*1) + (5*0.99) + (0*0) = 4.95$
- 根据以下公式得到$\theta^{(j)}$:

$$min_{\theta^{(1)},\theta^{(2)},\dots,\theta^{(n_u)}} = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r(i,j)=1}(\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(\theta_k^{(j)})^2$$

- 梯度下降, 和LR的差别只有不包含 $\frac{1}{m}$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(j)} \quad \text{for } k = 0$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(j)} + \lambda \theta_k^{(j)} \right) \quad \text{for } k \neq 0, \text{这个其实是通用式}$$

3. Collaborate Filtering

- 对于并不知道待估计的样本的特征向量x的情况:
  - 猜一个$\boldsymbol{\theta}$,根据$\boldsymbol{\theta}$求$\boldsymbol{x}$

    Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, estimate $x^{(1)}, \ldots, x^{(n_m)}$:

    $$\min_{x^{(1)},\ldots,x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$
  - 根据$\boldsymbol{x}$优化$\boldsymbol{\theta}$

    Given $x^{(1)}, \ldots, x^{(n_m)}$, estimate $\theta^{(1)}, \ldots, \theta^{(n_u)}$:

    $$\min_{\theta^{(1)},\ldots,\theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$
  - 再用$\boldsymbol{\theta}$反过来优化$\boldsymbol{x}$
  - 不断循环来优化$\boldsymbol{\theta}$ and$\boldsymbol{x}$
- Example:

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
Love at last	5	5	0	0	?	?
Romance forever	5	?	?	0	?	?
Cute puppies of love	?	4	0	?	?	?
Nonstop car chases	0	0	5	4	?	?
Swords vs. karate	0	0	5	?	?	?

  - 从图中可以看出Alice和Bob给了前三部爱情片高分，后两部动作片低分，因此他们偏好Romance,同样的我们可以发现 Carol和Dave偏好Action
  - 假设$\boldsymbol{\theta}$为

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

  - 例如, 对于电影"Love at last",Alice和Bob偏好而Carol和Dave不喜欢
    - $(\theta^{(1)})^T x^{(1)} \approx 5$
    - $(\theta^{(2)})^T x^{(1)} \approx 5$
    - $(\theta^{(3)})^T x^{(1)} \approx 0$
    - $(\theta^{(4)})^T x^{(1)} \approx 0$
    - 因此可以推测$x^{(1)} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$
  - 同理可以根据已有θ推定其他电影的类型

- 协同过滤算法Collaborative filtering algorithm:

$$J(x,\theta) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^{n} (\theta_k^{(j)})^2$$

  - 不再使用刚刚x,θ交替优化的过程, 而是同时最小化
  - 随机(打破对称)初始化$x^{(1)}, x^{(2)}, \ldots, x^{(n_m)}$, $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(n_u)}$
  - 使用梯度下降优化$J(x, \theta)$

$$\begin{aligned} x_k^{(i)} &:= x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right) \\ \theta_k^{(j)} &:= \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \end{aligned}$$

  - 对于给定θ的用户,使用训练得到的x计算$\theta^T x$作为预测评分

4. Vectorization: Low Rank Matrix Factorization

- $Y = X\Theta^T$ 两个向量相乘构造预测评分矩阵: 每个用户对每部电影的评分

$$\begin{bmatrix} (\theta^{(1)})^T(x^{(1)}) & (\theta^{(2)})^T(x^{(1)}) & \dots & (\theta^{(n_u)})^T(x^{(1)}) \\ (\theta^{(1)})^T(x^{(2)}) & (\theta^{(2)})^T(x^{(2)}) & \dots & (\theta^{(n_u)})^T(x^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ (\theta^{(1)})^T(x^{(n_m)}) & (\theta^{(2)})^T(x^{(n_m)}) & \dots & (\theta^{(n_u)})^T(x^{(n_m)}) \end{bmatrix}$$

- 根据每部电影的特征向量$x^{(i)}$寻找相关联的电影:
  - 相似度计算: $||x^{(i)} - x^{(j)}||$
- Mean Normalization
  - Example: 如果预测一个从未给任何电影做过评分的用户的评分, 不能将该用户的初始评分都设为0

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

  - 正确做法为正态化, 首先求取各行已知数据的平均值

$$\mu_i = \frac{\sum_{j:r(r,j)=1} Y_{i,j}}{\sum_j r(r,j)}$$

$$\mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix}$$

  - 然后将Y矩阵每个元素减去平均值,变换为

$$Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

  - 根据这个矩阵Y，学习θ和x
  - 最后预测结果为:$(\theta^{(j)})^T x^{(i)} + \mu_i$, 把减去的均值加回去

    - 例如对于User 5:
      $\theta^{(5)} = [0,\ 0]^T \rightarrow (\theta^{(5)})^T x^{(i)} = 0$再加上μ后即可得到预测评分数据
    - 注意不用再进行scaling(除以range或标准差): all the scales are similar
- 使用情景：
    - 用户购买了某个产品是否有其他的产品可以推荐给他

Large Scale Machine learning

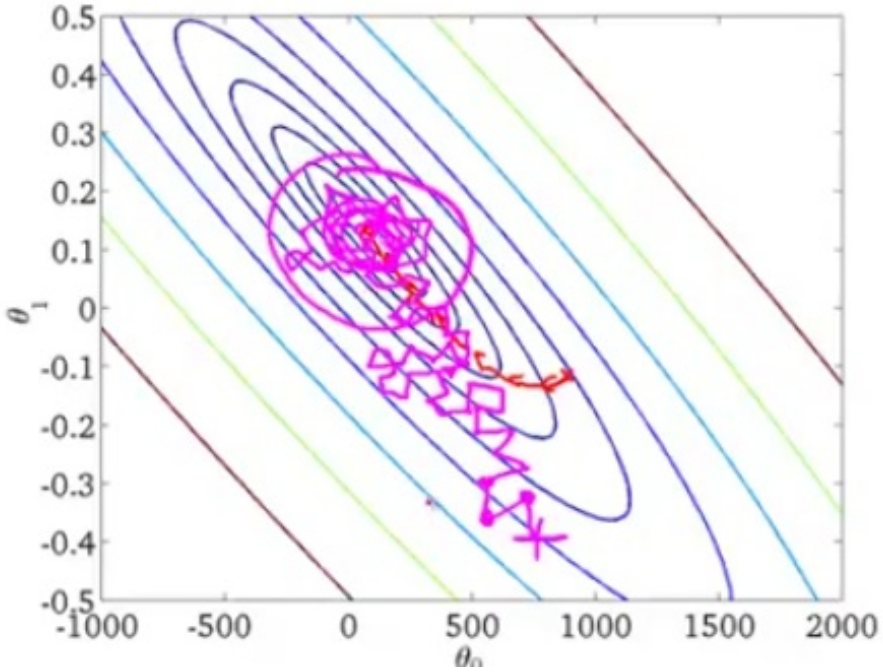
1. GD with large data set

- 对于High Variance模型: 大样本量效果更好
- 对于High Bias模型: 增大样本量没什么效果, 不如增加特征量让模型更复杂
- 使用Mini-Batch GD 和 Stochastic GD的动机
    - 每次迭代需要计算所有的样本，计算量大

2. Stochastic GD

- 首先要Randomly shaffle dataset
- for循环每次只使用一个样本对求偏导更新(循环次数为样本数)，外层的repeat对这个过程进行重复迭代
- 整体方向为全局最优,但不是每次迭代都为全局最优
- 最终的结果在全局最优解附近
- 外层的循环(当m很大的时候)一般为1-100次，具体视m的大小和下降的情况决定

repeat{
  for i =1:m{
    $\theta_j = \theta_j + (y(i) - h_\theta(x^{(i)}))x_j^{(i)}$
  }
}
(for every j = 0,1,...,n)



其下降过程会像粉色线一样

3. BGD与SGD的折中:Mini-batch Gradient Descent(MBGD)

- 首先要Randomly shaffle dataset
- 每次处理 b examples，外层的repeat对这个过程进行重复迭代

repeat{

for i = 1, 11+b, ..m-b+1{

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

}

}

(for every j = 0,1,...,n)

- Example:
  - 每次选择b个样本进行更新

    Say b = 10, m = 1000

    Repeat{

    for i = 1, 11, 21, 31, .. , 991{

    $$\theta_j = \theta_j - \frac{\alpha}{10}\sum_{k=1}^{i+9}(h_\theta(x^{(k)} - y(k)))x_j^{(k)}$$

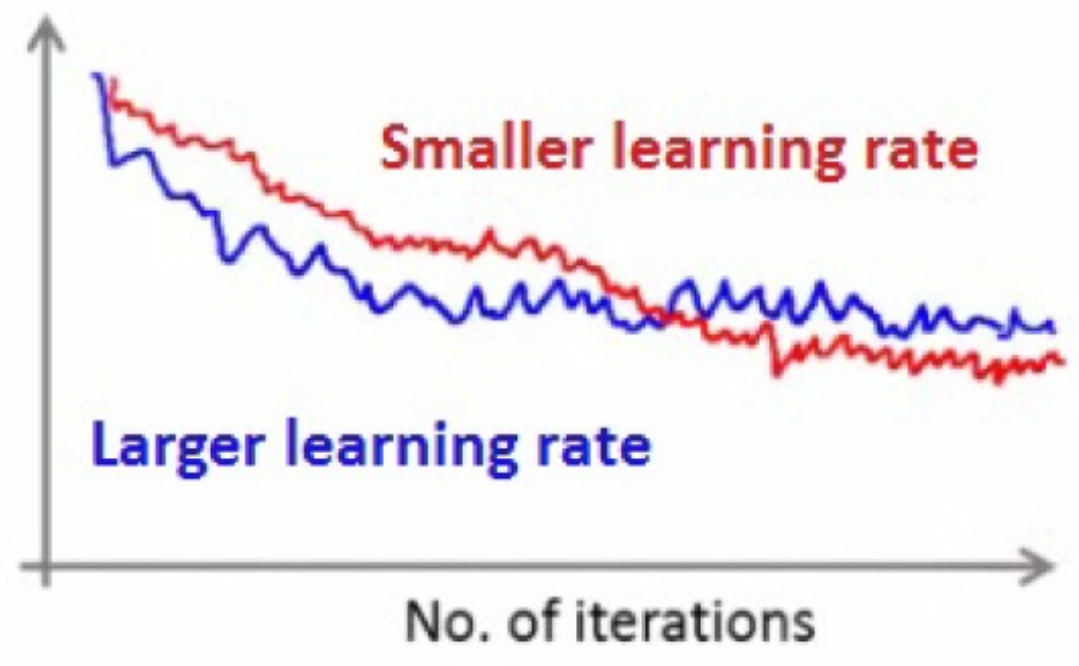
    }

}
(for every j = 0,1,...,n)

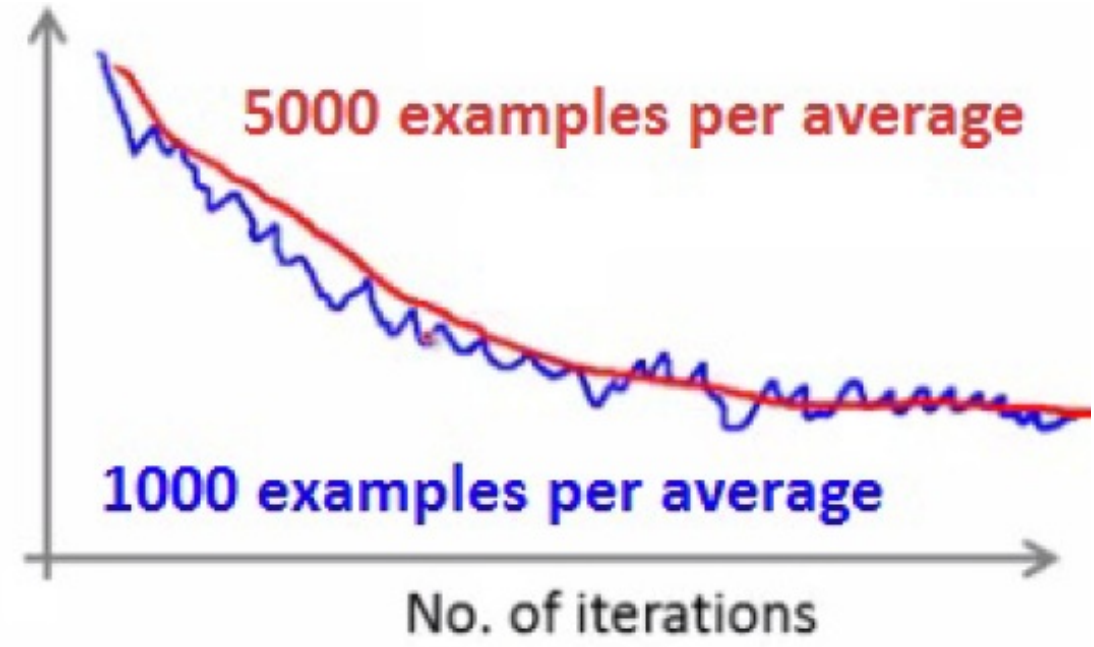
- SGD VS BGD:
  - SGD:
    - 每次最小化1条样本的损失函数;
    - 不是每次迭代得到的损失函数都向着全局最优方向;
    - 大方向正确，结果在全局最优解附近;
  - BGD:
    - 每次最小化所有样本的损失函数;
    - 得到全局最优解;
  - Mini-Batch GD
    - 每次最小化b条样本的损失函数
  - SGD相比BGD更适合处理大量数据

4. Stochastic GD Convergence and choose α: 选择合适的学习速率并检查是否发生了梯度上升

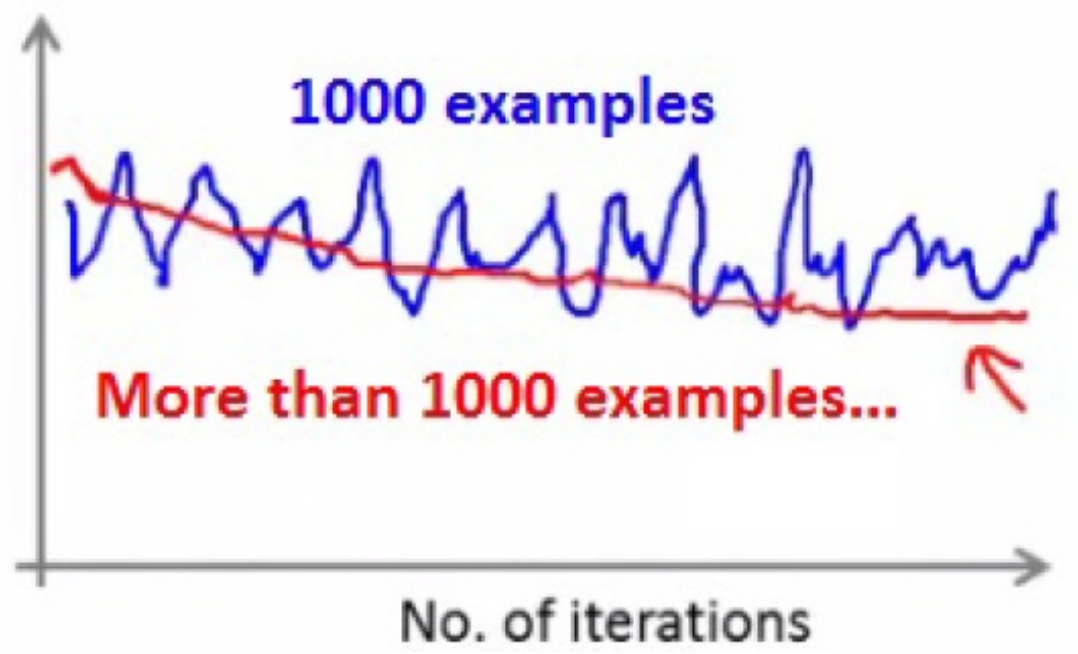
- BGD:
  - Plot J as a function of the number of iterations;
  - $J = \frac{1}{2m} \sum (h - y)^2$
- SGD:
  - $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)} - y^{(i)})^2)$
  - 在更新前先计算$cost(\theta, (x^{(i)}, y^{(i)}))$
  - 每进行1000次迭代, 给出这1000个样本的平均cost，检查随机梯度下降是否在收敛
- 几种情况:
  - 更小的学习速率α: 震荡较小



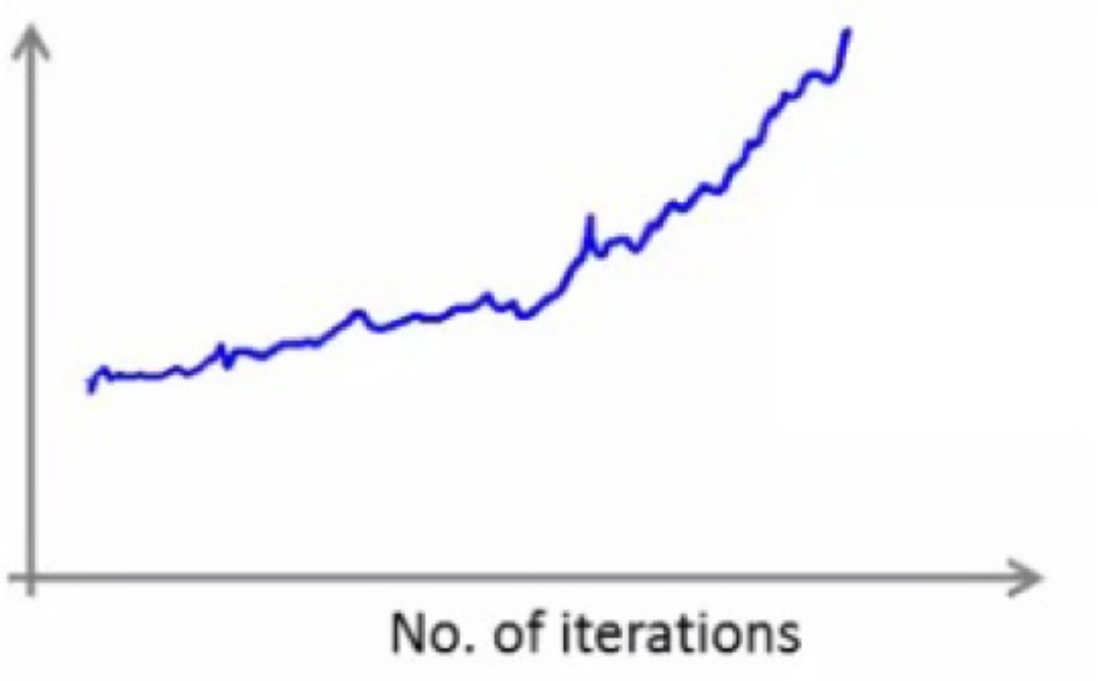
- 平均样本越大,曲线越平滑



- 噪音过大, 可使用更大的样本量进行调节, 得到cost function的实际趋势



- 曲线上升: 需要使用更小的学习速率曲线



- 改进随机梯度下降:降低波动
  - 随着迭代，不断更新(降低)α，这样可以使本来在全局最优附近震荡的cost function趋于全局最优解

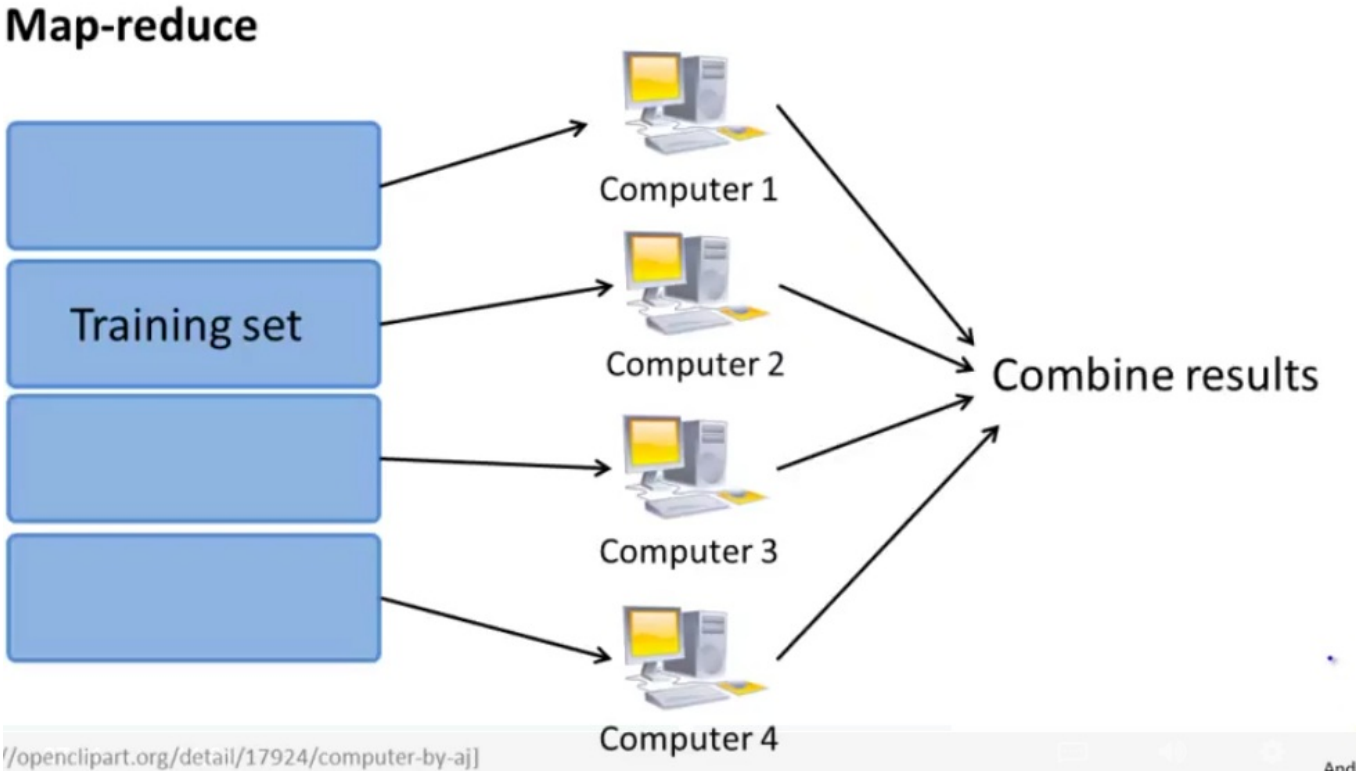
    $\alpha = \frac{cost_1}{iterationNumber + cost_2}$
    - cost1和cost2需要自行确定，iterationNumber是已经计算的样本数， 但是cost1和cost2的确定需要花费大量的时间

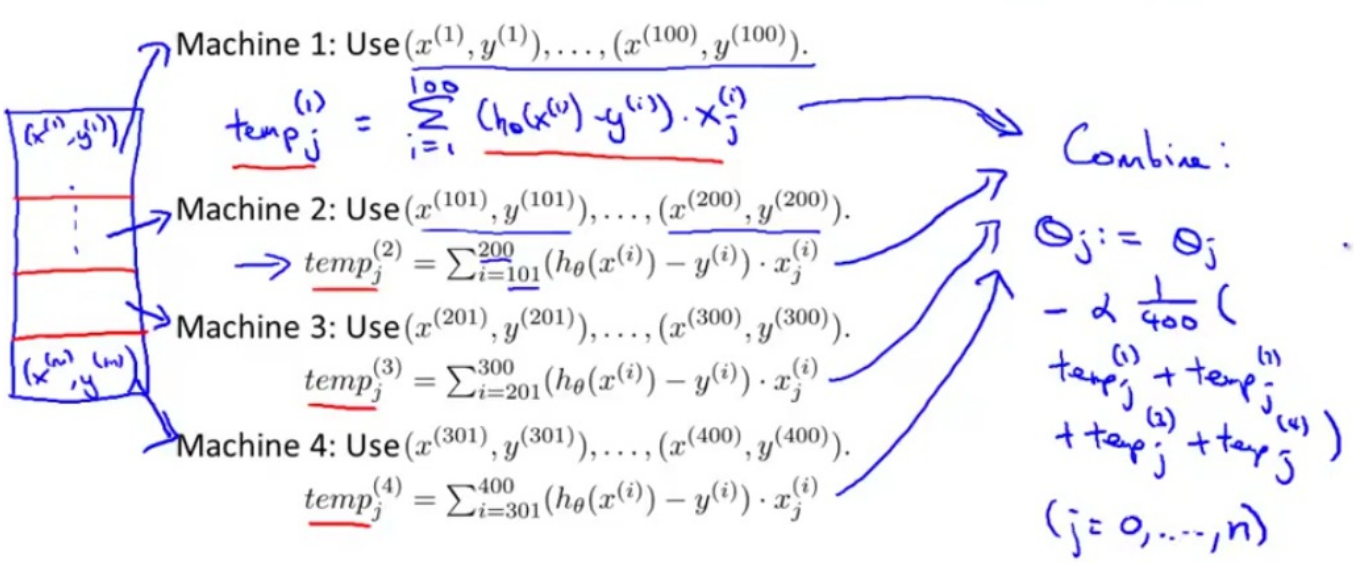
5. Online Learning

- 根据新的样本，边学习，边给出结果，类似于SGD
- 不需要保存所有数据，用后即丢弃
- Goal: learn $p(y=1|x;\theta)$ to optimize
- Repeat forever{
  Get (x,y) corresponding to user
  Update θ using (x,y):
  for $f \in \{0,1,\dots,n\}$:
  $\theta_j = \theta_j - \alpha(h_\theta(x)-y)x_j \ (j=0,1,\dots,n)$
  }
- 优势在于可以随着用户偏好的改变而改变

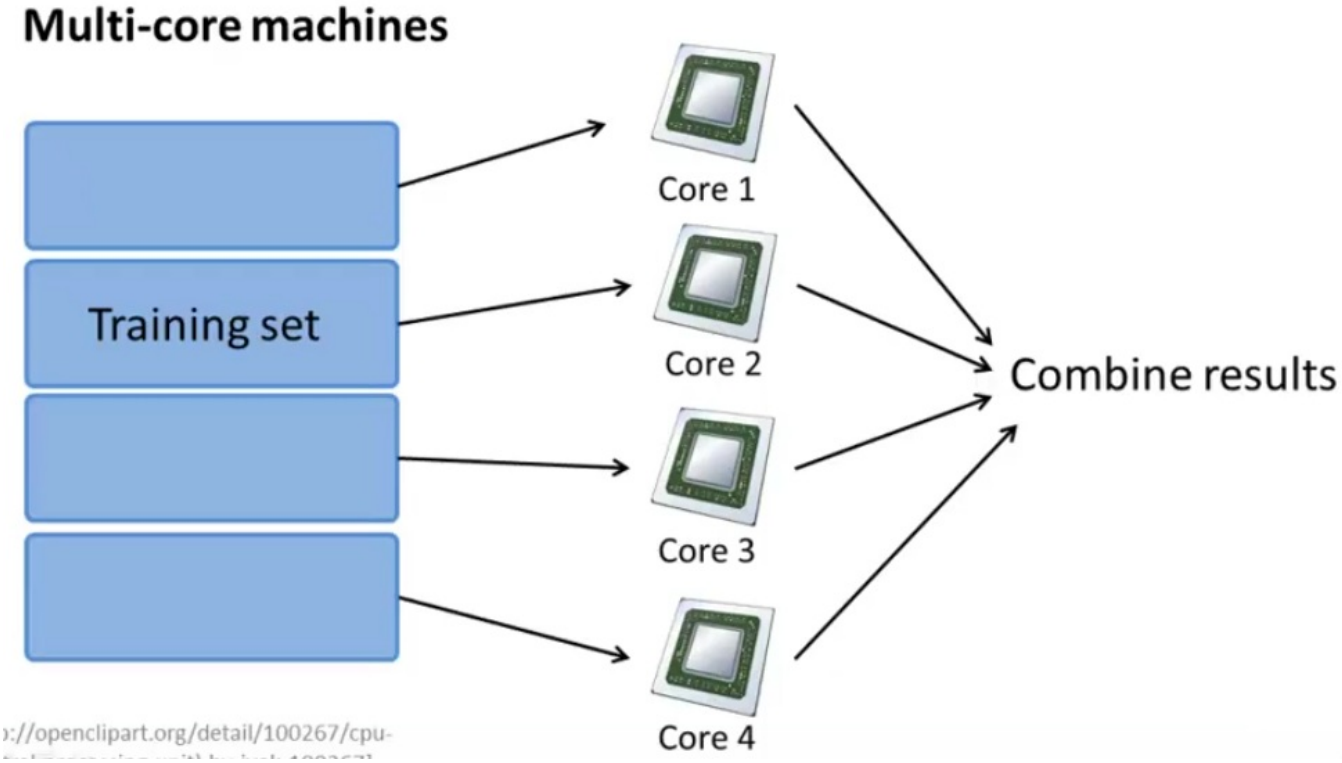
6. Map Reduce and Data Parallelism

多台机器:





单机:



- 其中使用的GD，近似于批量梯度下降法
- Example: devide and combine
  训练集
  - 训练集$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(400)}, y^{(400)})\}$
  - 将训练集分为4个子集, 每个子集100个元素
  - 使用4台机器分别进行训练得到四个J
  - 组合: $\theta_j = \theta_j - \frac{\alpha}{400} (J_1 + J_2 + J_3 + J_4)$

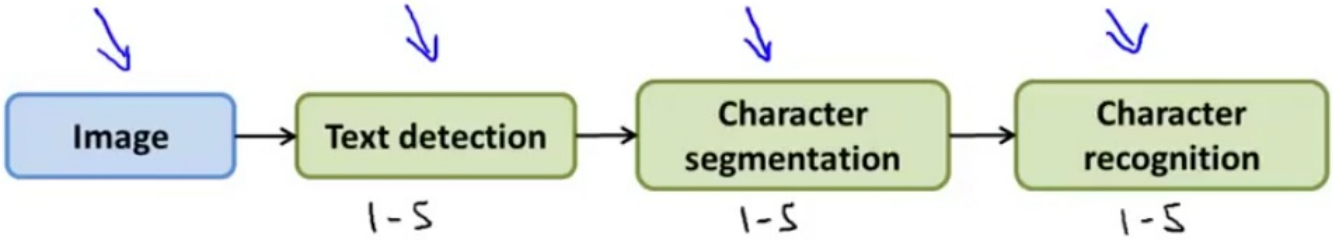
- MR适用情况: Model can be expressed as computing sums of functions over the training set
    - 例如线性回归和逻辑回归
    - BGD可以使用MR而SGD不行

        1. 注意事项
- SGD迭代时不减反增;
- SGD VS BGD;
    - 随机初始化
    - SGD并不能保证每次都收敛，只是总体方向正确
- Online learning
- 可以使用Map-Reduce的算法
- 使用MR前,需要先确定如何利用分开后再组合的数据展示算法

Application Example: Photo OCR

1. Pipeline

一个拥有许多阶段和成分的系统， 其中一些用到了机器学习。
* Example: Photo OCR pipeline



2. Sliding window detection

- 首先利用许多positive样本(y=1)和negative样本(y=0)训练一个神经网络或其他supervised算法
- 每次的移动量叫做step-size/stride，通常选择4个像素
- 用于每次扫描的window是不断增大的， 但是window中提出的图片提取后会被转换成82*36(训练所用标准)再带入训练好的神经网络(或其他算法)

- 用sliding window detection 可以完成 Text detection 和 Character segmentation， 但是他们各自都需要一个神经网络算法或其他算法对window所获取的图片进行分类

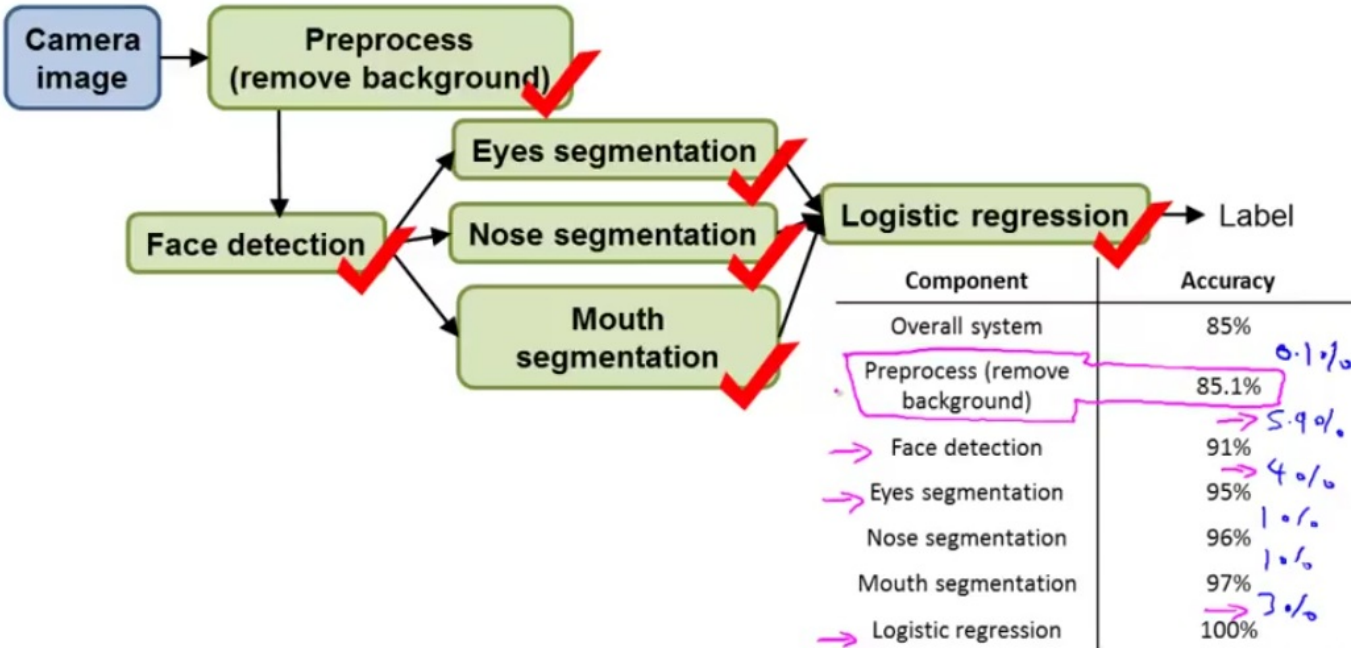
3. Get more data

- 先确保分类器是低bias得到， 否则再多的数据也没有用
- 主要方法:
  - Artificial data synthesis
    - Distortion data
      - Audio: 背景噪音， 低清晰度等
      - text： 扭曲文字，换用其他字体等
    - 通常再数据中加入purely random/meaningless noise是没什么作用的，如高斯噪音等
  - Collect/ label it yourself可以算算大概耗时，有时候是个好办法
  - Crowd source(E.G. Amazon Mechanical Turk)再网络上用低价雇人帮你label

Ceiling analysis

根据修正pipeline中不同component所能获得的增益大小选择优先需要优化的算法(利用增加数据集先进行测试增益)

Another ceiling analysis example



Component	Accuracy
Overall system	85%
Preprocess (remove background)	85.1%
Face detection	91%
Eyes segmentation	95%
Nose segmentation	96%
Mouth segmentation	97%
Logistic regression	100%

0.1%
5.9%
4%
1%
1%
3%