

Stanford Machine Learning Notes

Author : Yunqiu Xu

MachineLearning Python Statistics Matlab Coursera

主要参考资料:

[Coursera Stanford Machine Learning](#)

[Holehouse.org](#)

Week 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
- 栗子: 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

1.1 Supervised VS Unsupervised Learning

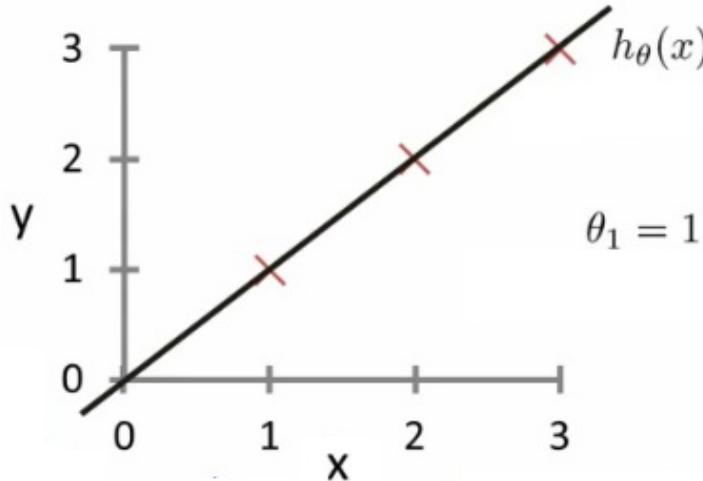
- Supervised learning: relationship between the input and output
 - Regression: continuous
 - Classification: discrete
- Unsupervised learning: do not need to know the results
 - e.g. Clustering
- Quiz 1:
 - different elements in ML;
 - Regression VS Classification;
 - Supervised and Unsupervised Learning;

1.2 LR with one variable

- 线性回归是一种参数学习方法: 有固定的明确的参数 θ , θ 确定后就不会改变

1.2.1 Hypothesis function:

$$\hat{y} = h_{\theta}(x) = \theta_0 + \theta_1 x$$



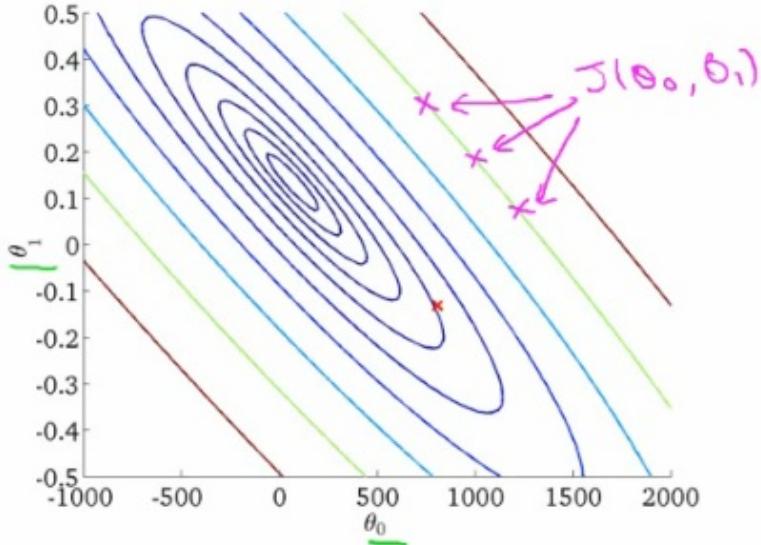
1.2.2 Cost Function $J(\theta_0, \theta_1)$

$$RSS = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

$$J(\theta_0, \theta_1) = \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2 = \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

- Similar to the average of RSS in Stanford Statistics I
- Goal : choose parameters θ_0, θ_1 to minimize $J(\theta_0, \theta_1)$
- 栗子: 等高线图
 - 等高线(年轮)图可用于描述 $J(\theta_0, \theta_1)$
 - θ_0 为横坐标, θ_1 为纵坐标
 - 年轮最小圈为线性回归最佳值

$J(\theta_0, \theta_1)$
 (function of the parameters θ_0, θ_1)



1.3 Gradient Descent

1.3.1 GD algorithm

- Repeat following loop until convergence

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

- j 为 0 或是 1
- $:=$ 为 赋值 符号
- 不断 梯度 下降 求 导 直到 左右 趋 同 (即 收 敛, 导 数 项 为 0)
- α : 学习速率, 用 于 控 制 梯 度 下 降 的 幅 度, 过 小 下 降 太 慢, 过 大 可 能 导 致 过 拟 合
- 随 着 不 断 优 化 赋 值, 导 数 变 化 幅 度 会 自 动 变 小, 不 需 要 时 刻 改 变 α
- 注意 每 次 迭 代, θ_0, θ_1 需 要 同 时 更 新 赋 值, 相 当 于 更 新 整 个 $\theta = \{\theta_0, \theta_1, \dots, \theta_n\}$ 向 量

- 考 虑 下 面 的 栗 子:

- 正 确 : 同 时 更 新 赋 值

- $t_0 := \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}$
- $t_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$
- $\theta_0 := t_0$
- $\theta_1 := t_1$

- 错误：

- $t_0 := \theta_0 - \alpha \frac{\partial J}{\partial \theta_0}$
- $\theta_0 := t_0$
- $t_1 := \theta_1 - \alpha \frac{\partial J}{\partial \theta_1}$
- $\theta_1 := t_1$

1.3.2 GD for linear regression

- Repeat following loop until convergence

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x_i) - y_i)$$

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

- Batch Gradient Descent: 每步梯度下降都使用全部数据, 可以和后面Stochastic GD对比

1.3.3 Convex Optimization

- GD只能得到极值而非最值;
- 为了保证得到全局最小值, cost function一定要为convex(凸函数);
- 凸函数充要条件: 二阶导数不小于0
- 对于凸优化问题, 局部最优解即是全局最优

1.5 局部加权线性回归LWLR

- 适合LR欠拟合的情形, 对特征集要求不是特别严格
- 给予每个($y_i - \theta_T x_i$)一定权重 w_i , 得到:

$$J = \sum_{i=1}^m w_i (y_i - \theta_T x_i)$$

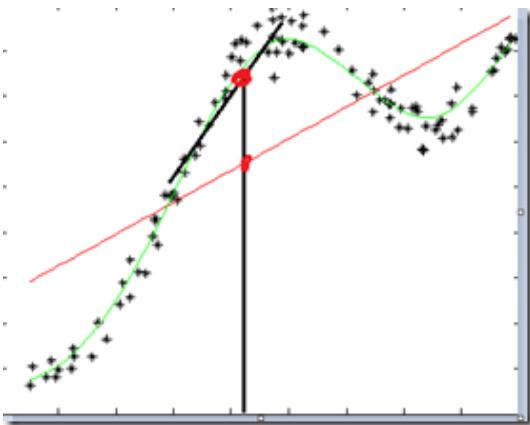
- w_i 可以使用高斯核函数进行表示
 - x 为预测值, x_i 为第i个训练数据
 - σ^2 控制衰减速率, 越大衰减越慢

$$w_i = \exp\left(-\frac{(x_i - x)^2}{2\sigma^2}\right)$$

- Normal Equation形式:

$$\theta = (X_T W X)^{-1} X_T W Y$$

- 下图中红线为普通线性回归, 黑线为局部加权



- + LWR与普通线性回归区别在于这是一种非参数学习方法
- + 每进行一次预测就会对 θ 进行更新
- + 要求一直保存训练集样本
- + 训练集容量较大时非参数学习效率较低

Week 2

2.1 Linear Regression with Multiple Variables

2.1.1 Multiple Features

- n: number of features
- m: number of observations
- $x^{(i)}$: 样本i的所有特征
- $x_j^{(i)}$: 样本i的第j个特征
- Hypothesis function :

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

- 注意这里假设 $x_0^i = 1$

$$h_{\theta}(x) = [\theta_0 \quad \theta_1 \quad \dots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} = \theta^T x$$

- 向量化表示: 每列都是一个样本, 样本特征表示为列向量

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

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

- Cost function

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

- Vectorized version:

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

2.1.2 GD for Multiple Variables

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

- 矩阵形式:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

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

2.1.3 Feature Normalization

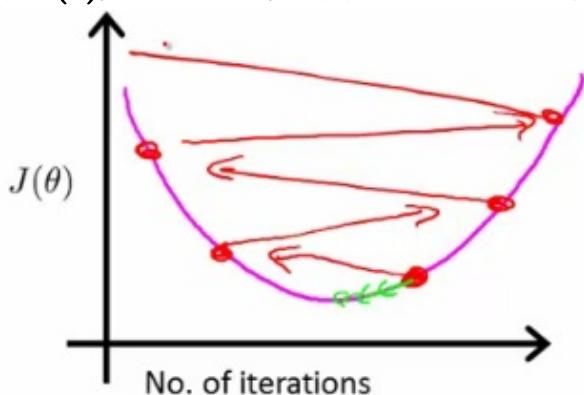
- Idea: make sure features are on a similar scale
- Goal: reduce the iteration
- 举个栗子：
 - $x_1 \in (100, 10000)$

- $x_2 \in (1, 100)$
- 通过正则化将 x_1, x_2 的范围都缩小到(0,1)
- Mean normalization:
 - $x_i := \frac{x_i - \mu_i}{S_i}$
 - $S_i = \max - \min$, 或者也可以为标准差

2.1.4 Learning Rate α



- α 的选择标准
 - 每次迭代后 $J(\theta)$ 应该下降, 如下图绿线
 - 若 $J(\theta)$, 如下图红线, 应使用更小的学习速率

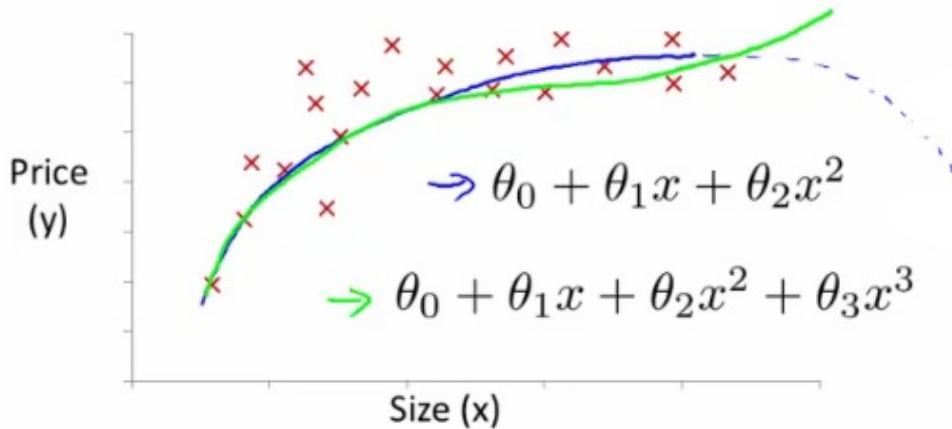


2.2 Features and Polynomial Regression

2.2.1 多项式回归

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \dots + \theta_n x_1^n$$

Polynomial regression



2.2.2 Normal Equation

- No need to do feature scaling

$$\theta = (X^T X)^{-1} X^T y$$

- $X_{m \times (n+1)}$: design matrix, 每一个 x_i 都是 $n+1$ 维度的特征向量的转置, $x_0^{(i)} = 1$
- $y_{m \times 1}$: m-dimentional vector
- 举个栗子: 4 observations + 5 features

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}; \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$\left(\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}^{-1} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \right)$$

2.2.3 GD VS NE

Gradient Descent	Normal Equation
need to choose α	no need to choose α
need to iterate	no need to iterate
works well even when n is large	slow with massive data

2.3 Octave Tutorial

2.3.1 Basic Operations

```

1. help(object)
2. % #to make a comment
3. 1~=2 #不等于
4. 1&&0 #AND
5. 1||0 #OR, 另一种写法:xor(1,0)
6. PS1(">> ") #将起始标识改为>>

```

2.3.2 Variables

```

1. a=3; #不加分号的话会直接print 'a=3'
2. disp(a); #只输出a的值
3. disp(sprintf('2 decimals: %0.2f',a)) #sprintf:输出string
4. >>> 3.14
5.
6. format long/short #默认的保留小数位（长/短）

```

2.3.3 Vectors & Matrices

```

1. V=[1 2 3] #1*3
2. V=[1;2;3] #3*1
3. V=1:0.1:1.3 #start:step:end,输出1 1.1 1.2 1.3
4. V=1:6 #1 2 3 4 5 6
5.
6. M=[1 2;3 4;5 6] #3*2
7. ones(2,3) #[1 1 1;1 1 1]
8. C=2*ones(2,3) #C=[2 2 2;2 2 2]
9.
10. w=zeros(1,3) #w=0 0 0
11. I=eye(4) #4-class identity matrix
12. flipud(eye(4)) #右上左下

```

2.3.4 Random, distribution and plot

```
1. w=rand(3,3) #3*3 随机数矩阵
```

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

2.3.5 Size & Length

```
1. size(M) #返回行列数  
2. size(M,1) #第一维数（此处为行数）  
3. size(M,2) #此处为列数  
4. length(M) #the number of elements
```

2.3.6 Save & Load

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

- 打开m文件的办法：切换到pwd，直接输入path并enter

```
1. who #给出当前features的名字，类似dir()  
2. whos #给出所有features的size/scale/class，类似head()  
3.  
4. clear feature #clear a feature  
5. clear #清空当前变量，无论who/whos都返回NULL  
6.  
7. v=xxx(1:10) #选取某个feature xxx的前10个元素组成新feature  
8.  
9. save xxx.mat v; #将feature v 保存在xxx.mat  
10. save yyy.txt v -ascii; #save v as a text
```

2.3.7 manipulate

```
1. A=[1 2;3 4;5 6];B=[10 11;12 13;14 15]  
2.  
3. A(3,2) #6  
4. A(3,:) #第三行的所有列值:5 6  
5. A([1 3],:) #第一行+第三行  
6. A(:,2)=[7;8;9] #赋值  
7. A=[A,[100;101;102]]#add a new column  
8. [A;[0 0 0 0]]#add a new row  
9. A(:) #put all in A into a single vector  
10. C=[A B] #横向组合，得到3*4矩阵  
11. D=[A;B] #纵向组合，得到6*2矩阵
```

2.3.8 Computing the data

```
1. A*B #矩阵乘法
2. A .* B #同位置元素相乘, 要求A与B的行列数相同
3. A .^2 #A中每个元素都平方
4. 1 ./ v #v中每个元素取倒数
5. log(v)/exp(v)/abs(v)/-v
6. v+ones(3,1) #等价于v+1
7. A' '#转置
8. pinv(A) #逆矩阵
9.
10. val=max(A) #给出A中的最大值
11.     max(max(A)); max(A(:)); #均为等价式
12. max(A); #返回最大值所在行
13. [v1,v2]=max(A) #v1为最大, v2为第二大
14. 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
15.
16. find(A<3) #filter: 返回所有小于3的元素
17. A=magic(3) #返回3阶魔方矩阵
18. sum()/prod() #累加/阶乘
19. floor()/ceil() #小于某数的最大整数/大于某数的最小整数
20.     sum(A,1) #每列的sum
```

- 举个栗子：得到A的对角矩阵并对元素求和

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

2.3.9 Plotting Data

- 基本操作

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

- 多图操作

```
1. #在不同的图层分别作图
2. figure(1); plot(x,y1);
3. figure(2); plot(x,y2); #分别做两个图
4.
5. #在同一张图中作两条线
```

```
6. plot(x,y1);
7. hold on;
8. plot(x,y2, 'r'); #可指定颜色
9.
10. #在同一个图层中作两个图
11. 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
12. subplot(1,2,2);plot(x,y1); #右边可以继续添加plot(x,y2)
13. clf; #清空图层
```

- 其他操作

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

2.3.10 Control Statements

- for loop

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

- while loop

```
1. i=1;
2. while i<= 5,
3.     v(i)=100;
4.     i=i+1;
5. end;
```

- if/elseif/else/break

```
1. i=1;
2. while true,
3.     v(i)=999;
4.     i=i+1;
5.     if i==6,
6.         break;
7.     end; #the end of if
8. end; #the end of while
9.
10. if xxx,
11.     disp('xxx');
12. elseif yyy,
13.     disp('yyy');
14. else
```

```
15.      disp('zzz');
16. end;
```

2.3.11 Define the function

```
1. #定义函数
2. function y =funcname(x)
3. y= x^2;
4.
5. #调用函数
6. cd 'the function's path'
7. funcname(5);
```

- 栗子:

```
1. function [y1,y2]=funcsquare(x)
2.     y1=x^2;
3.     y2=x^3;
4. [a,b]=funcsquare(5);
5. >>> 25,125
```

2.3.12 Compute cost function

```
1. function J=costFunction(X,y,theta)
2. %X:design matrix
3. %y:class labels
4.
5. m=size(X,1); %number of training examples
6. predictions=X*theta; # pred of hypothesis on all m examples
7. sqrErrors=(predictions-y) .^ 2; #squared errors
8.
9. J=1/(2*m) * sum(sqrErrors);
```

Week 3

3.1 Logistic Regression

3.1.1 Classification and Representation

- Classification

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

- for multiclassess:

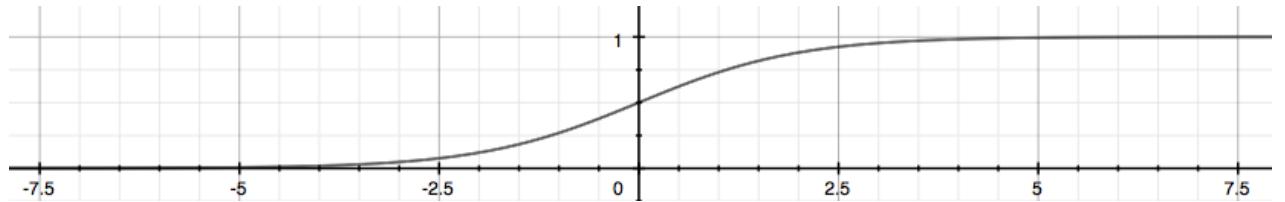
$$y \in \{0, 1, 2, 3, \dots\}$$

- Hypothesis Representation: Sigmoid function

◦ Sigmoid function 可以保证训练结果在 $[0, 1]$ 区间

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

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

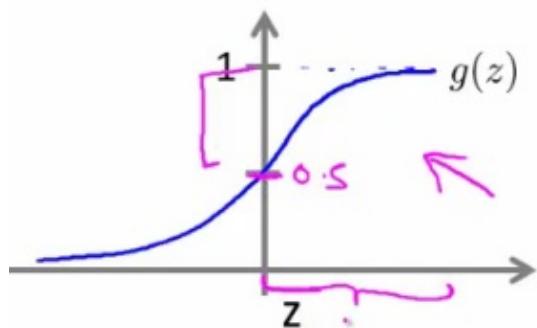


◦ 输出结果实际上为得到1的概率: likelihood estimation

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

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

- Decision Boundary: 建立一个 x_1 - x_2 的坐标系，确定分类边界



$$h_{\theta}(x) \geq 0.5 \rightarrow y = 1$$

$$h_{\theta}(x) < 0.5 \rightarrow y = 0$$

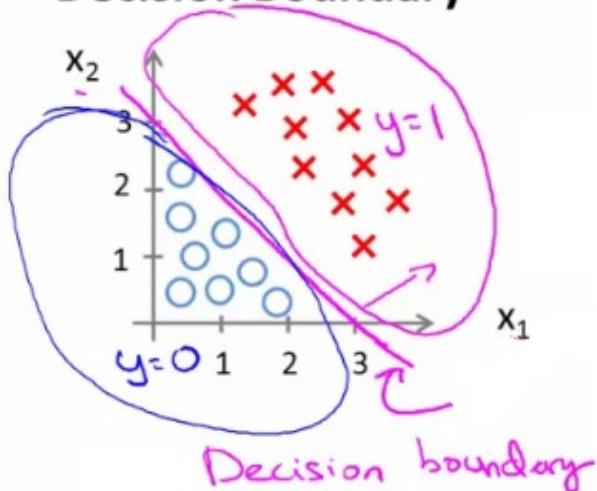
◦ 对于 $h_{\theta}(x) = g(\theta^T x)$

$$\theta^T x \geq 0 \rightarrow y = 1$$

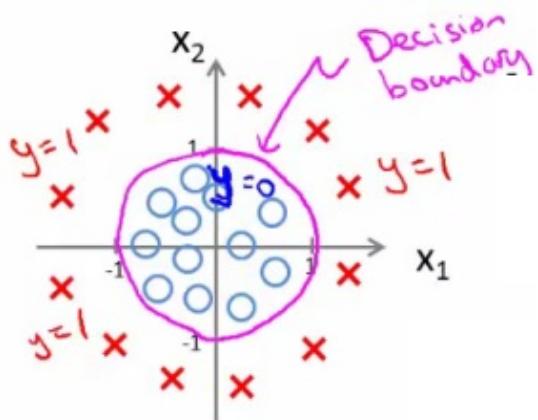
$$\theta^T x < 0 \rightarrow y = 0$$

◦ 栗子1: 线性

Decision Boundary



- $\theta = [-3 \ 0 \ 0]^T$
- Boundary: $x_1 + x_2 = 3$
- $x_1 + x_2 \geq 3 \rightarrow y = 1$
- 栗子2：非线性



- $h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2)$
- $\theta = [-1 \ 0 \ 0 \ 1 \ 1]^T$
- Boundary: $x_1^2 + x_2^2 = 1$

3.1.2 Cost Function

- 逻辑回归非线性，之前线性回归的cost function 无法得到convex, 需要进行转换

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

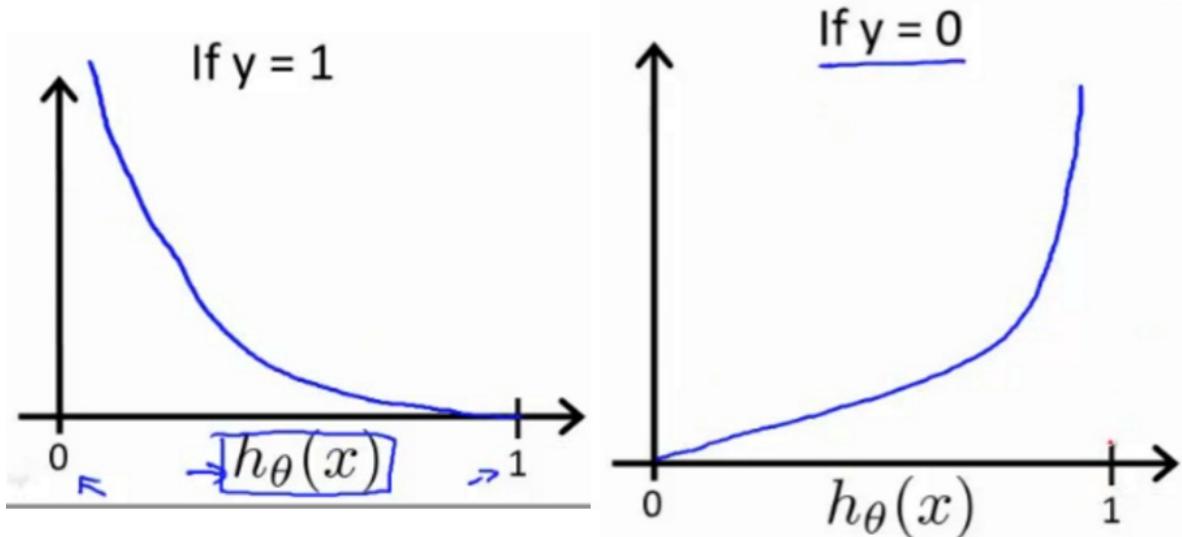
$$\text{cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y=1 \\ -\log(1 - h_\theta(x)) & \text{if } y=0 \end{cases}$$

- Simplified cost function: maximum likelihood function

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

- $h_\theta(x)$ 越接近 y , cost function 输出结果越小

- $\text{cost}(h_\theta(x), y) = 0$ if $h_\theta(x) = y$
- $\text{cost}(h_\theta(x), y) = \infty$ if $|h_\theta(x) - y| \rightarrow 1$



- Advanced Optimization

- Gradient descent 类似线性回归

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

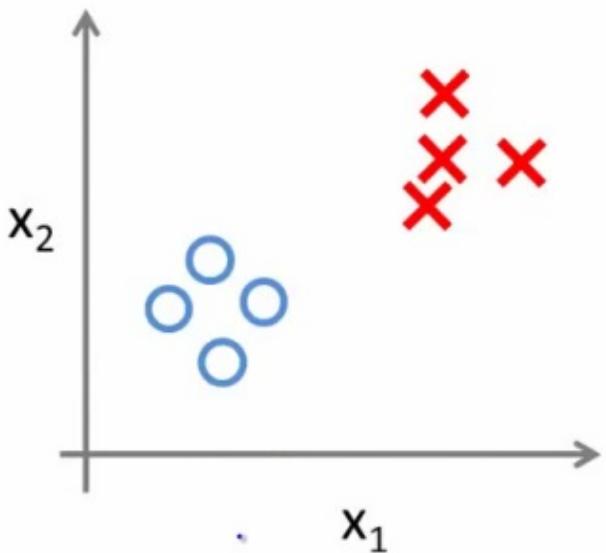
$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

- Conjugate gradient /BFGS / L-BFGS

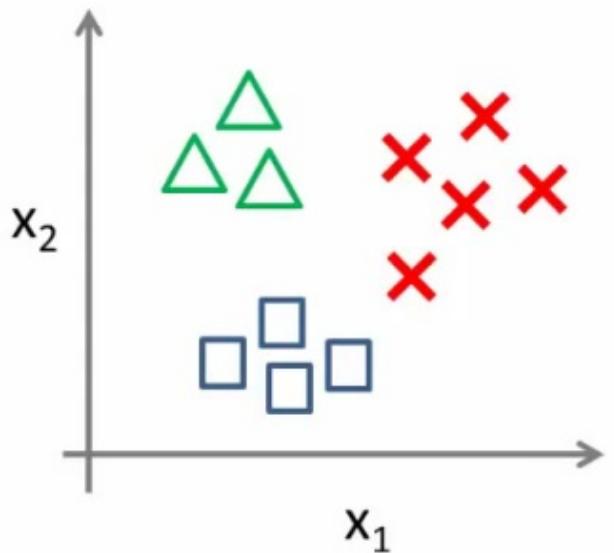
- 后三个不需要 α , 速度更快但更复杂

3.1.3 Multiclass Classification

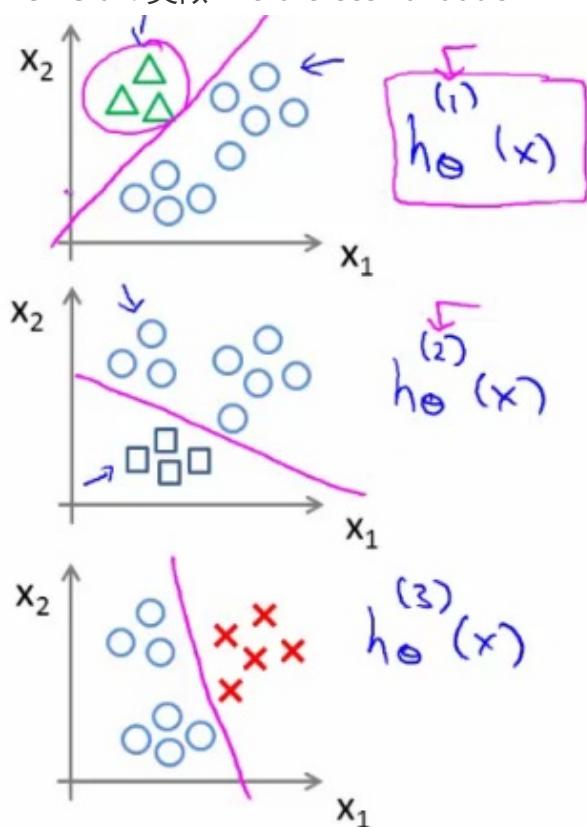
Binary classification:



Multi-class classification:



- One VS all: 类似k-fold cross validation



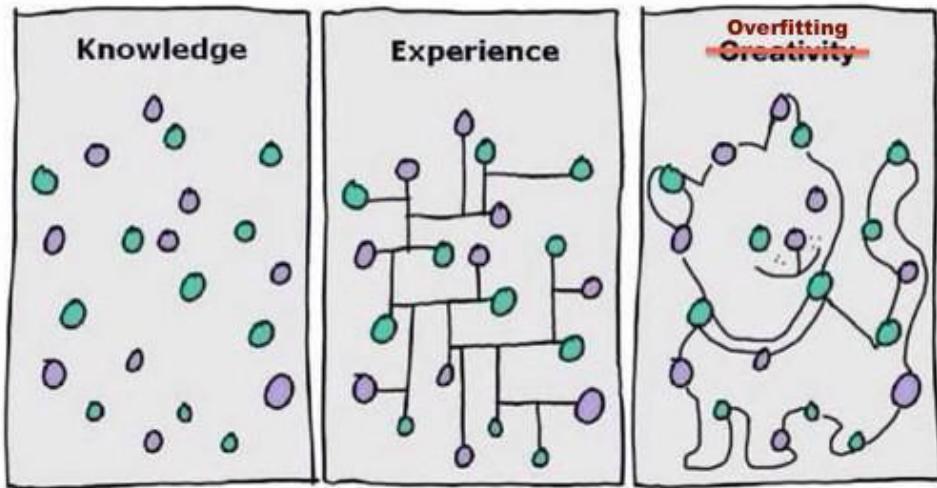
- 使用逻辑回归分类器 $h_{\theta}^{(i)}(x)$ 来预测 $y = i$ 的概率
 - $y \in \{0, 1, 2, \dots, n\}$
 - $h_{\theta}^{(0)}(x) = P(y = 0|x; \theta)$
 - $h_{\theta}^{(1)}(x) = P(y = 1|x; \theta)$

...

- $h_{\theta}^{(n)}(x) = P(y = n|x; \theta)$
- 对于测试数据 x , 将其归类为 $h_{\theta}^{(i)}(x)$ 最大的那一类

3.2 Regularization

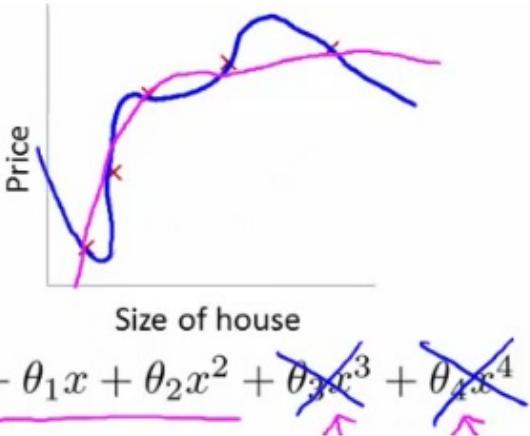
3.2.1 Overfitting and underfitting



- underfitting: high bias
- overfitting: high variance

3.2.2 Address overfitting

- Reduce the number of variables
 - Subset selection
 - 只选取和response相关的一部分p预测因子,组成subset进行fit
- 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



3.2.3 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 θ_j 被设置为0，模型被简化
 - 简化模型，Var逐渐降低，Bias逐渐升高

3.2.4 Regularized Linear Regression

- Gradient Descent
 - 由于 $x_0^{(i)} = 1$ 不变，因此 θ_0 不纳入岭回归范畴
 - $\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)})$
 - for $j \in \{1, 2, \dots, n\}$
 - $\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}) \right) + \frac{\lambda}{m} \theta_j \right]$
- Normal Equation (最小二乘法)

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

$$I_{n+1 \times n+1} = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

3.2.5 Regularized Logistic Regression

- 格式类似线性回归，只是 $h_\theta(x)$ 为sigmoid，此处略

3.3 批量梯度下降与随机梯度下降

- Batch GD:
 - 最小化所有训练样本的损失函数，使得最终求解的是全局的最优解
 - 每次迭代需要全部数据，计算量大

repeat{

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

(for every $j=0, \dots, n$)

}

- Stochastic GD:

- 每次只使用一个样本对 θ 求偏导更新
- 迭代次数为样本数
- 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

1. Randomly shuffle dataset ;

2. **repeat{**

for $i=1, \dots, m\{$

$$\theta_j' = \theta_j + (y^i - h_\theta(x^i)) x_j^i$$

(for $j=0, \dots, n$)

}

}

- 改进随机梯度下降：降低波动

- 随着迭代，不断更新(降低) α
- 每次迭代随机选择样本，本轮结束时将该样本删除
- BGD与SGD的折中:Mini-batch Gradient Descent(MBGD)
 - 每次选择 b 个样本进行更新

Say $b=10, m=1000.$

Repeat{

for $i=1, 11, 21, 31, \dots, 991\{$

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

 (**for every** $j=0, \dots, n$)

}

}

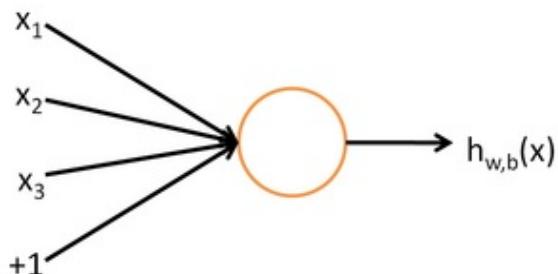
Week 4 Neural Networks:Representation

- 可参考[资料](#)

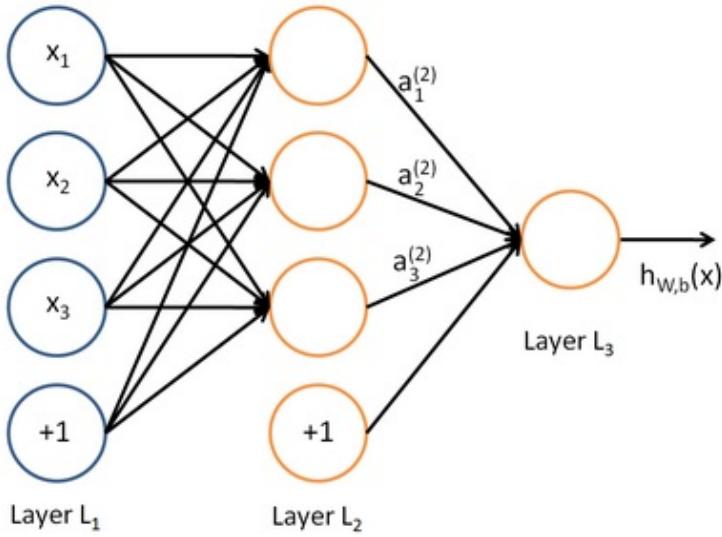
4.1 Neural Networks-Model Representation

4.1.1 多项式表示

- 一个简单的神经网络:



- 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



Layer L_1

Layer L_2

Layer L_3

- $+1$: 截距项 $x_0, a_0^{(2)}$
- $\Theta(j)$: 从第j层到+1层的权重向量
- n_l 为神经网络层数, L_j 为第j层
- $a_i^{(j)}$: 第j层的第i个activation function

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

4.1.2 向量化表示

- 使用 $z_k^{(j)}$ 表示j层i单元的输入加权和

$$z_k^{(j)} = \Theta_{k,0}^{(j-1)} x_0 + \Theta_{k,1}^{(j-1)} x_1 + \dots + \Theta_{k,m}^{(j-1)} x_m$$

- 使用向量可以将 x 以及 $z^{(j)}$ 表示为

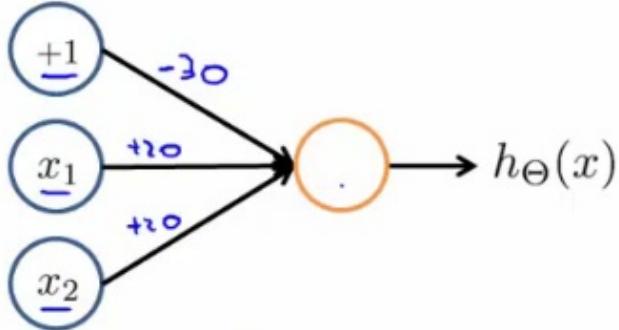
$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_m \end{bmatrix} z^{(j)} = \begin{bmatrix} z_0^{(j)} \\ z_1^{(j)} \\ \vdots \\ z_m^{(j)} \end{bmatrix}$$

- 因此可以得到 $z^{(j)} = \Theta^{(j-1)} a^{(j-1)}$, 即 $a_i^{(j)} = g(z_k^{(j)})$

4.2 Applications: 判断逻辑关系

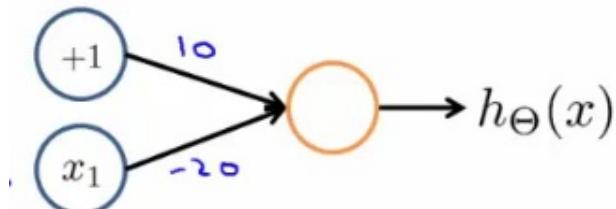
- 栗子1 : AND, 双层神经网络

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



$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$$

- 栗子2: NOT x_1
 - $x = [1, x_1]^T$
 - $\theta = [10, -20]$
 - $h_\theta(x) = g(10 - 20x_1)$
 - 根据真值表可得到结论NOT x_1



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

- 栗子3: XNOR, 三层神经网络

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

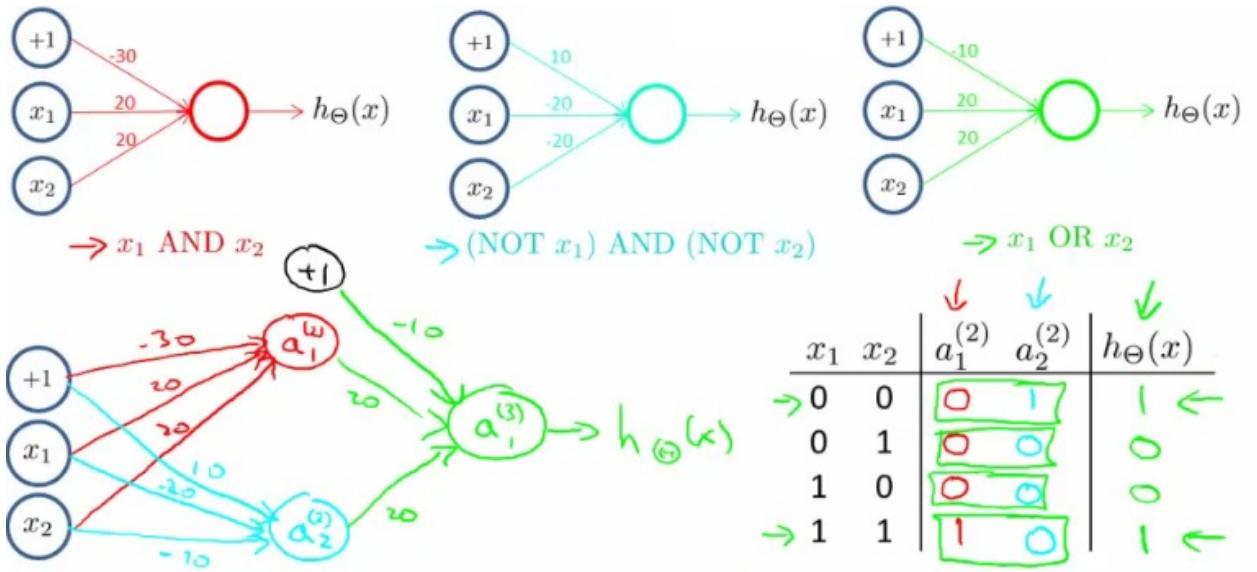
$$\Theta^{(1)} = \begin{bmatrix} -30 & 20 & 20 \\ 10 & -20 & -20 \end{bmatrix}$$

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

$$\Theta^{(2)} = [-10 \quad 20 \quad 20]$$

- 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

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

- 可以得到多个输出, 提升辨识度

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

Week 5 Neural Networks: Learning

5.1 Cost Function

- Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- L : number of layers (input + hidden + output)
- S_l : number of units in layer l , 不包括bias unit $a_0^{(l)}$

- 逻辑回归的cost function为:

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

- 套用到前向传播神经网络中:

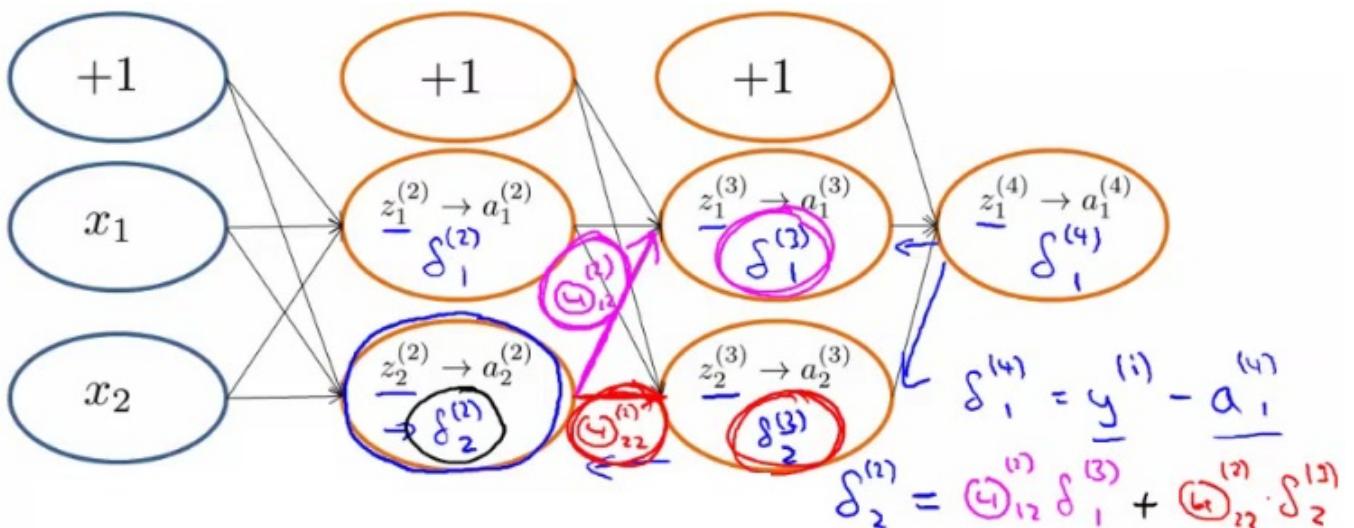
○ $\Theta_{j,i}^{(l)}$: l 层, i 行, j 列的系数向量

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

- 使用GD需要求出 $J(\Theta)$ 并对其求偏导 $\frac{\partial J(\Theta)}{\partial \Theta_{j,i}^{(l)}}$

- 该cost function过于复杂, 在下一节引入后向传播进行优化

5.2 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 (\mathbf{x}, \mathbf{y}) , 首先进行前向传导, 计算所有的激活值 $a_j^{(l)}$, 包括 $h_\Theta(\mathbf{x})$ 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;
 - 最后根据激活值和残差得到偏导数;
- 求取残差的方法:

- 对于最后一层(输出层) , 可以直接算出残差

$$\delta^{(L)} = a^{(L)} - y$$

- 得到前一层残差:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* g'(z^{(l)})$$

- 偏导数 $g'(z)$ 的求取,以逻辑回归为例:

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

- 则偏导数可表示为 $g'(z) = g(z)(1 - g(z))$

- 总结上述步骤,可使用以下公式计算残差:

$$\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$$

- 举个栗子, 对于4层神经网络: $\delta^{(4)} \rightarrow \delta^{(3)} \rightarrow \delta^{(2)}$

- $\delta^{(4)} = a^{(4)} - y = h_{\theta}(x) - y$
- $\delta^{(3)} = ((\theta^{(3)})^T \delta^{(4)}) .* a^{(3)} .* (1 - a^{(3)})$
- $\delta^{(2)} = ((\theta^{(2)})^T \delta^{(3)}) .* a^{(2)} .* (1 - a^{(2)})$

- 得到残差及激活值后根据以下公式求取偏导数:

$$\frac{\partial J(\Theta)}{\partial \Theta_{j,i}^{(l)}} = \frac{1}{m} \sum_{t=1}^m a_j^{(t)(l)} \delta_i^{(t)(l+1)}$$

- Backpropagation 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 backward 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 $D_{i,j}^{(l)} = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^{(l)}}$

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

5.3 Implementation Note: Unrolling Parameters into Vectors

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

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

- 返回原来的矩阵: unrolled vector -> original matrices

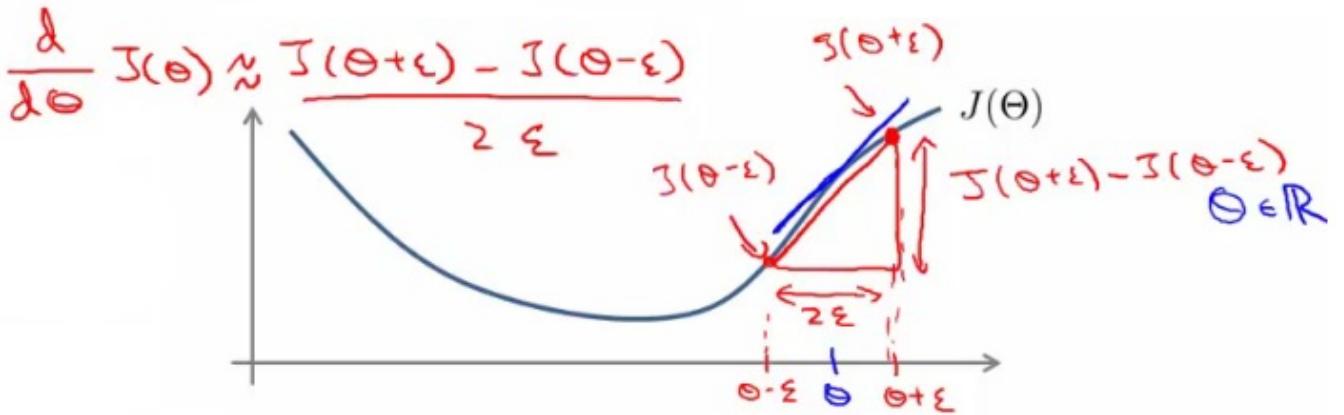
```
1. thetaVec=[Theta1(:);Theta2(:);Theta3(:)];#length=231
2. DVec=[D1(:);D2(:);D3(:)];
3. Theta1=reshape(thetaVec(1:110),10,11);
4. Theta2=reshape(thetaVec(111:220),10,11);
5. Theta3=reshape(thetaVec(221:231),1,11);
```

- 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

5.4 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 m 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
 - 未计算偏置项导致错误
- 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)



- Numerical Gradient Checking:

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

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

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

- 通过计算两边的接近程度来判定函数是否合理

- 代码实现:

```

1. for i=1:n,
2.     thetaPlus=theta;
3.     thetaPlus(i)=thetaPlus(i)+EPSILON;
4.     thetaMinus=theta;
5.     thetaMinus(i)=thetaMinus(i)-EPSILON;
6.     gradApprox(i)=(J(thetaPlus)-J(thetaMinus))/(2*EPSILON);
7. end;

```

5.5 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 $[-\epsilon, \epsilon]$, 在该定义域中随机取值

```

1. Theta1=rand(10,11)*(2*INIT_EPSILON)-INIT_EPSILON;

```

```
2. Theta2=rand(10,11)*(2*INIT_EPSILON)-INIT_EPSILON;
3. 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

5.7 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激活函数;
 - 构建cost function;
 - 使用后向传播计算所有偏导数;
 - 使用梯度检验判断cost function是否合理, 若合理, 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最小化 $J(\Theta)$;

Week 6

6.1 Advice for Applying Machine Learning

6.1.1 Evaluate a Learning Algorithm

- Evaluating a Hypothesis
 - Split the data as training set and testing set
 - Learn Θ and minimize $J_{train}(\Theta)$ from training data;
 - Compute test error $J_{test}(\Theta)$ from test set;
- Test set error
 - 对于线性回归, 可用以下公式表示误差

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

- 对于分类问题，可用以下公式表示misclassification error

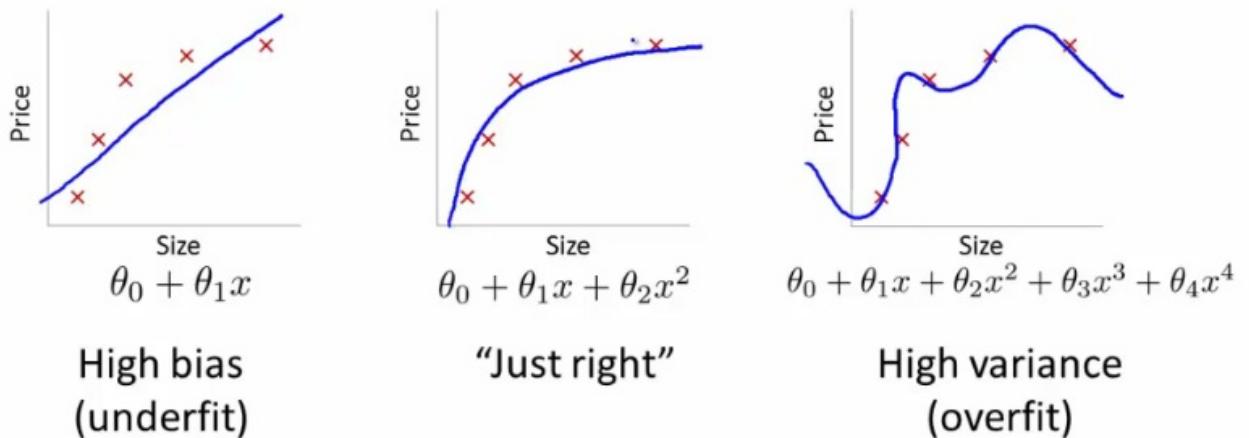
$$Err(h_{\Theta}(x), y) := \begin{cases} 1 & \text{if } h_{\Theta}(x) \geq 0.5, y = 0 \text{ or } h_{\Theta}(x) < 0.5, y = 1 \\ 0 & \text{No misclassifying} \end{cases}$$

- 分类问题平均误差为：

$$J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} Err(h_{\Theta}(x_{test}^{(i)}), y_{test}^{(i)})$$

6.1.2 Model Selection and Train/Validation/Test Sets

- Model Selection: 这里的栗子是多项式回归
 - d : degree of polynomial (the number of features)
 - $\Theta^{(d)}$: degree为 d 的多项式的参数向量
 - e.g. $\Theta^{(2)} = [\theta_0, \theta_1, \theta_2]$
 - $J_{test}(\Theta^{(d)})$: degree为 d 的多项式的test error
 - 模型选择的目标在于寻找其中test error最小的模型



- Train/Validation/Test Sets

- 还是使用之前的cost function来计算误差，只是样本量m不同
- 需要将样本划分成三个集合，例如：

- Train set: 60%
- CV set: 20%
- Test set: 20%

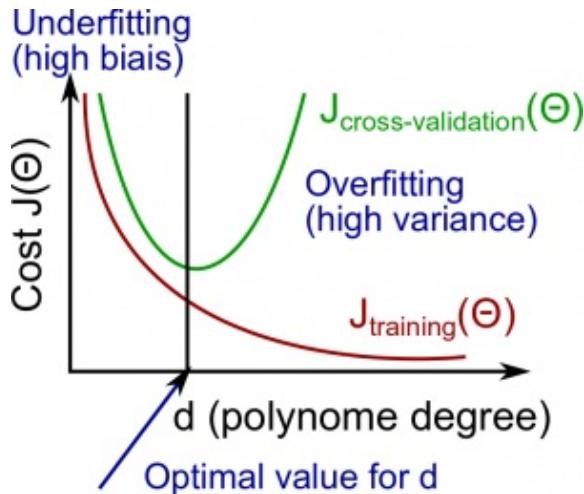
- 模型选择的过程：

- $\min(J_{Train})$
- 得到每个多项式的 Θ 向量
- $\min(J_{CV})$

- 选取 J_{CV} 最小的模型导入测试集进行测试
- 注意一般情况下 $J_{CV} > J_{Test}$, 原因是多fit了一个参数d

6.1.3 Diagnosing Bias vs. Variance

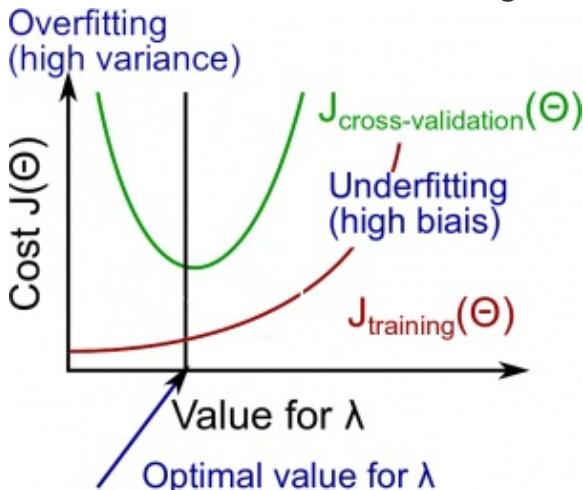
- Underfitting : d过小, 模型过于简单, high bias
- Overfitting : d过大, 模型过于复杂, high variance



- 随着 d 增加 : J_{Train} 降低, J_{CV} 以及 J_{Test} 先降低后升高
 - 先High Bias, 之后High Variance
 - High Bias阶段 : $J_{Train} \approx J_{CV}$ 且都很高
 - High Variance阶段 : $J_{Train} \ll J_{CV}$

6.1.4 Regularization and Bias/Variance

- 先通过使用不同的 λ 得到不同的 $J(\Theta)$ 及对应的 Θ 向量;
- 通过 CV 对 Θ 向量进行评估, 从而选取合适的 λ
- λ 增加会导致模型变简单(underfitting): J_{Train} 不断升高, J_{CV} 先降低后升高

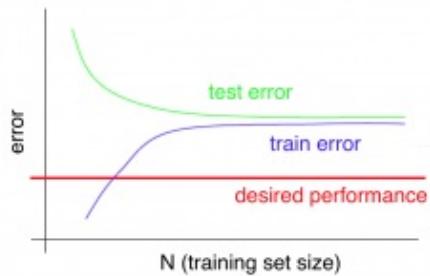


6.1.5 Learning Curve

- 学习曲线是样本量和误差的关系图

More on Bias vs. Variance

Typical learning curve for high bias(at fixed model complexity):

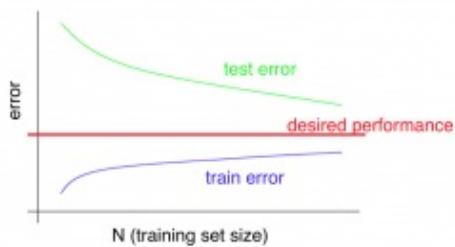


- 若模型本身high bias --> 仅仅增加样本量意义不大

- 随着样本量 m 增加， J_{Train} 不断增大， J_{CV} 不断降低
- 最终 J_{Train} 与 J_{CV} 趋同，不会交叉

More on Bias vs. Variance

Typical learning curve for high variance(at fixed model complexity):



- 若模型本身high variance --> 增加样本量有一定帮助

- 随着样本量 m 增加， J_{Train} 不断增大， J_{CV} 不断降低，但速度很慢
- 需要大大增加样本量才能勉强趋同

6.1.6 Diagnosing Neural Networks

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

6.1.7 Typical rules

Item	Performance
Larger training set	fix high var
Reducing additional features	fix high var
Getting additional features	fix high bias
Adding polynomial features	fix high bias
Larger λ	fix high var

Item	Performance
Smaller λ	fix high bias
Larger NN	easier to overfitting
Smaller NN	easier to underfitting

6.1.8 Quiz

- 因为样本量更大的缘故, 训练集一般从一开始的表现就优于测试集
- 训练时只会用到训练集和CV集, 无论何时都不会用到测试集

6.2 Machine Learning System Design

6.2.1 降低误差的几种方法

- Collect more data;
- Develop more sophisticated features;
- Develop more sophisticated 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

$$Recall = \frac{TP}{TP + FN}$$

- 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

$$Accuracy = \frac{TP + TN}{Total}$$

6.2.4 Trading Off Precision and Recall

- 确诊癌症需要有十足把握才决策:
 - high precision , low recall;
 - $h_{\theta}(x) > 0.9$ 才会判定为1;
- 避免癌症患者未被诊断:
 - high recall , low precision;
 - $h_{\theta}(x)$ 很低就会被判定为1
- 使用F score评估精确度和召回率

$$F = \frac{2PR}{P + R}$$

- 不使用accuracy的原因: 若实际1/0的比例相差悬殊, accuracy不合理

6.2.5 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

6.2.6 Quiz

- 复习准确度/精确度/召回率/F等指标的计算
- 增大数据量对以下情况有帮助 :
 - Features x contain sufficient information to predict y accurately
 - 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

Week 7 Support Vector Machine

- 参考资料: [JerryLead](#) , [pluskid](#)等人的blog

7.1 Optimization Objective: 将逻辑回归改造为SVM

- 逻辑回归的cost function :

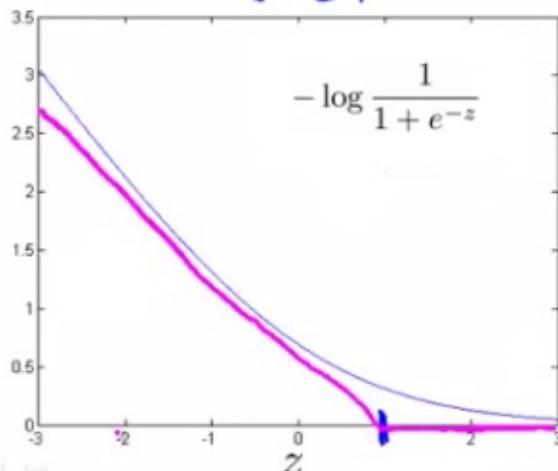
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

- 引入 $cost_1$ 与 $cost_0$ 函数:

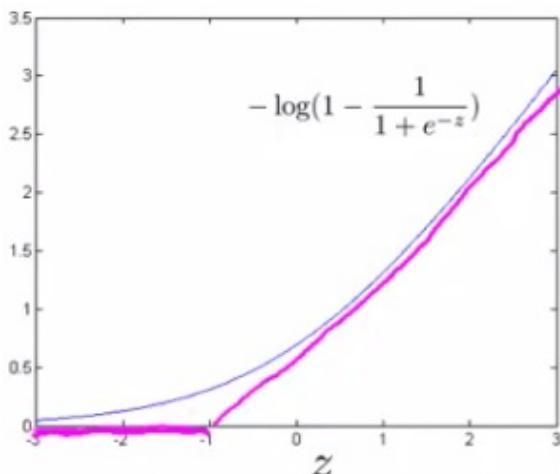
$$cost_1(\theta^T x^{(i)}) = -\log(h_\theta(x^{(i)}))$$

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

- $\theta^T x^{(i)} > 1 \rightarrow \text{Outputs } 0$



- $\theta^T x^{(i)} < -1 \rightarrow \text{Outputs } 0$



- 令 $z = \theta^T x$, 则:

$$cost_1(z) = \max(0, k(1 - z))$$

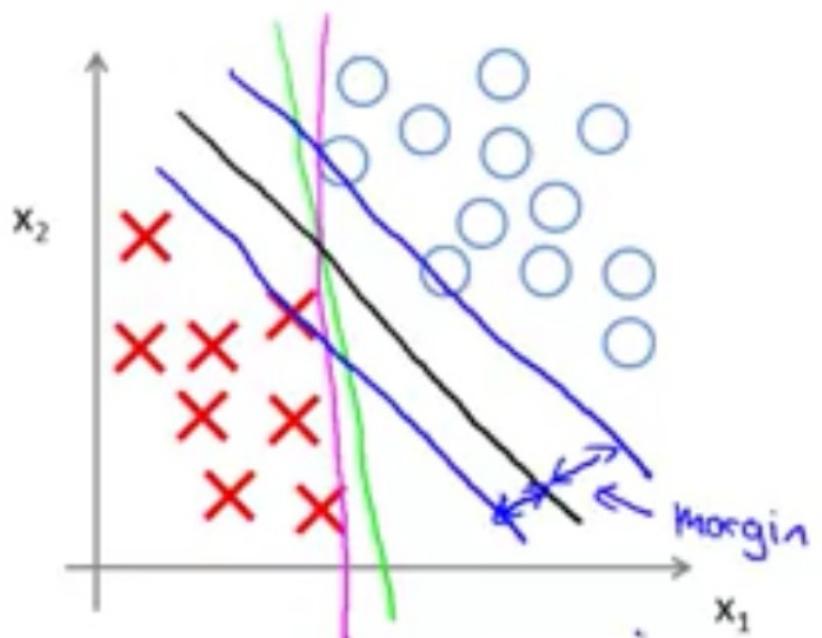
$$cost_0(z) = \max(0, k(1 + z))$$

- k 为常数项, 用于控制斜率
- 将逻辑回归改造为:
$$J(\theta) = \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{\lambda}{2} \sum_{j=1}^n \theta_j^2$$
 - cost_1 与 cost_0 与样本量无关, 因而不需要添加 m 项
 - 例如: 对于 $(u - 5)^2 + 1$ 与 $10(u - 5)^2 + 10$, 得到最小值时的 u 并不会因为乘以 10 而变化
- 使用新参数 $C = \frac{1}{\lambda}$, 得到如下 SVM cost function:
$$J(\theta) = C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

7.2 Large Margin Intuition

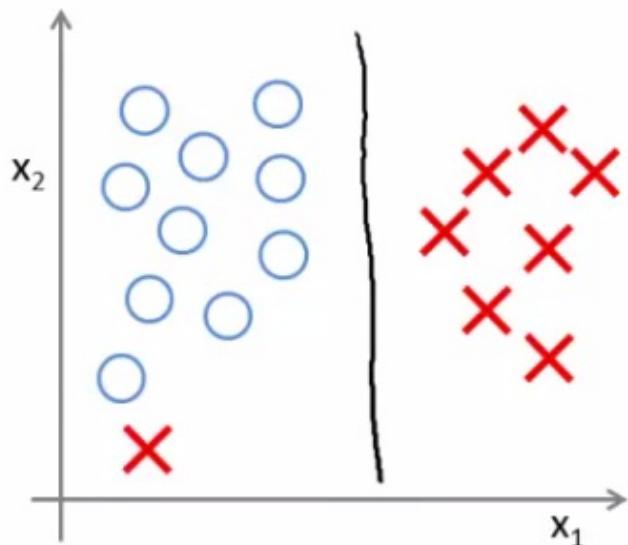
- 为了实现之前 $\Theta^T x > 1$ 及 $\Theta^T x < -1$ 时都输出 0 的目标, SVM 决策边界设置为
 - if $y = 1$, then $\Theta^T x \geq 1$ (not just ≥ 0);
 - if $y = 0$, then $\Theta^T x \leq -1$ (not just ≤ 0);
- margin: 两条边界线距离中间分界线的距离

SVM Decision Boundary: Linearly separable case

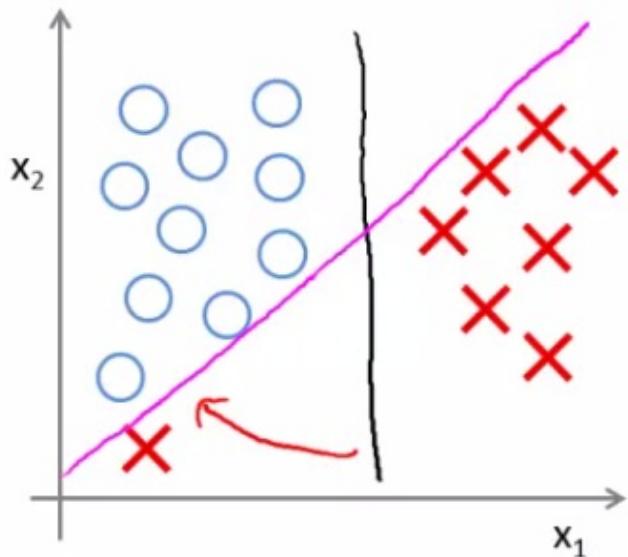


Large margin classifier

- large margin:



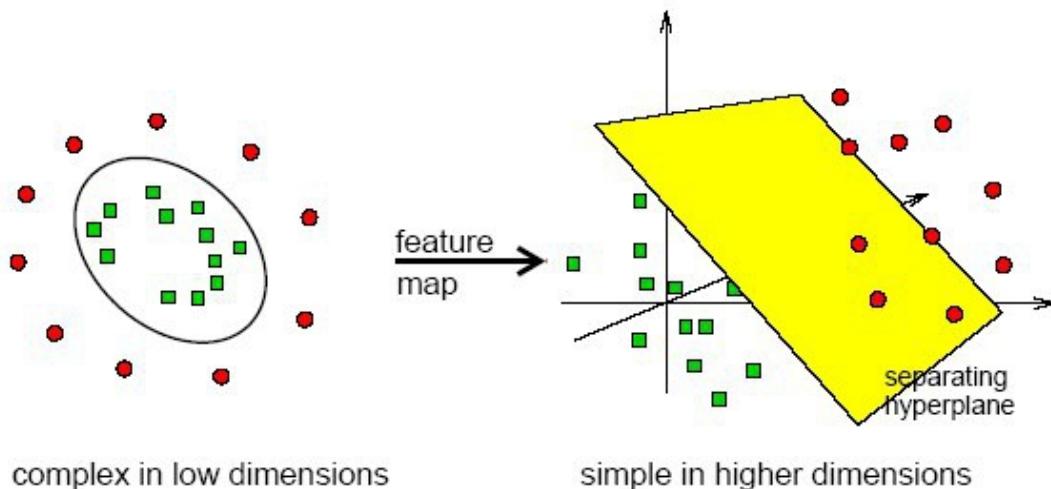
- small margin:



7.3 Kernels

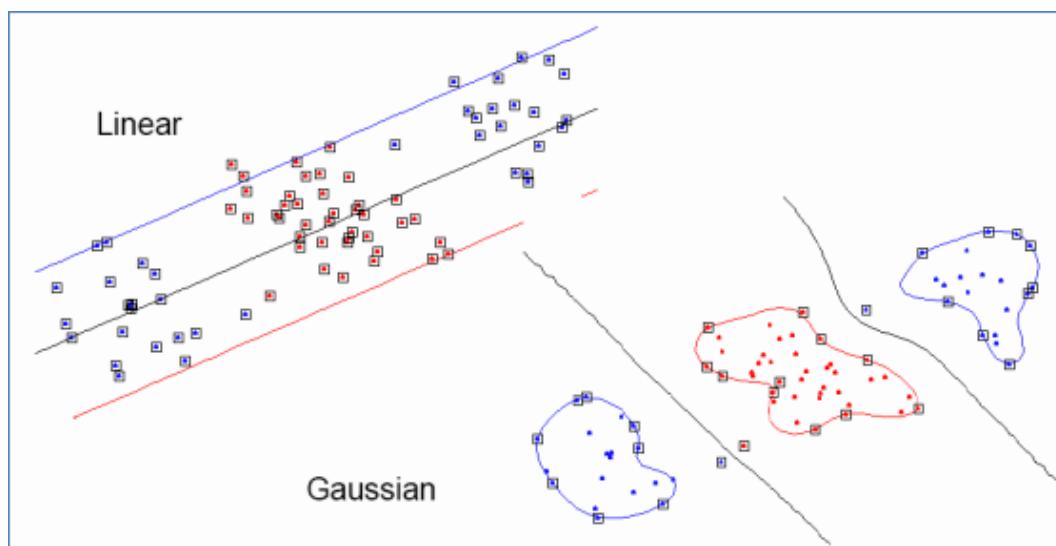
- 低维空间线性不可分的样本在高维空间会变得可分

Separation may be easier in higher dimensions



- $K(x_1, x_2)$ 可用于计算两个样本在高维空间的内积 $f(x_1)^T * f(x_2)$
 - 省去在高维空间进行繁琐的多项式计算

7.3.1 用于计算相似度的高斯核



- 使用以下公式计算 x 与 landmarks $l^{(i)}$ 的相似度

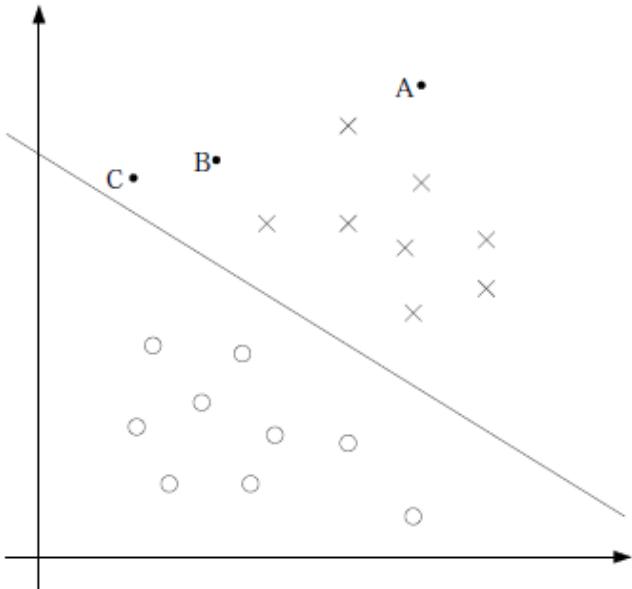
$$f_i = \text{similarity}(x, l^{(i)}) = \exp\left(-\frac{\|x - l^{(i)}\|^2}{(2\sigma^2)}\right)$$

- $\|x - l^{(i)}\|^2 = \sum_{j=1}^m (x_j - l_j^{(i)})^2$: 欧氏距离
 - 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 $x \approx l^{(i)} \rightarrow \|x - l^{(i)}\|^2 \approx 0 \rightarrow f_i \approx 1$
 - if $x \neq l^{(i)} \rightarrow \|x - l^{(i)}\|^2 \gg 0 \rightarrow f_i \approx 0$

- 根据这些相似度计算结果, 可将 $h_{\Theta}(x)$ 改写为:

$$h_{\Theta}(x) = \Theta_0 f_0 + \Theta_1 f_1 + \Theta_2 f_2 + \dots$$

7.3.2 使用Kernel进行预测的栗子



- 给定 $\theta = [-0.5, 1, 1]$ 构成决策边界 l
- 假设点C离 l 很近, 则:
 - $f_1 \rightarrow 0$
 - $f_2 \rightarrow 1$
 - $\theta_0 + \theta_1 f_1 + \theta_2 f_2 \geq 0 \rightarrow \text{Output } 1$
- 假设点A离 l 很远, 则:
 - $f_1 \rightarrow 0$
 - $f_2 \rightarrow 0$
 - $\theta_0 + \theta_1 f_1 + \theta_2 f_2 \approx 0 \rightarrow \text{Output } 0$

7.3.3 SVM with Kernels

- Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$
- Choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$
- Given training example $x^{(i)} \in \{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
 - $f = [f_0 = 1, f_1, f_2, \dots, f_m]$
 - $f_1^{(i)} = \text{similarity}(x^{(i)}, l^{(1)})$
 - $f_2^{(i)} = \text{similarity}(x^{(i)}, l^{(2)})$
 - ...
 - $f_1^{(i)} = \text{similarity}(x^{(i)}, l^{(i)})$ 这个完全相似
 - ...

- $f_m^{(i)} = \text{similarity}(x^{(i)}, l^{(m)})$
- 最后将 $x^{(i)}$ 赋值为 $f^{(i)} = [f_0^{(i)} = 1, f_1^{(i)}, \dots, f_m^{(i)}]$ 作为新特征向量，代入 cost function

$$J(\theta) = C \sum_{i=1}^m \left[y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

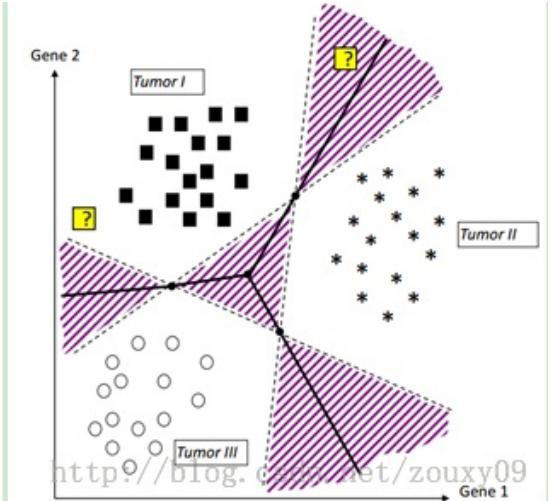
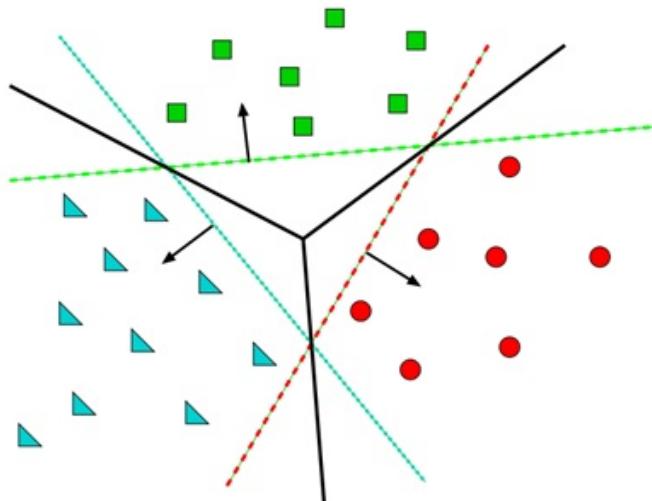
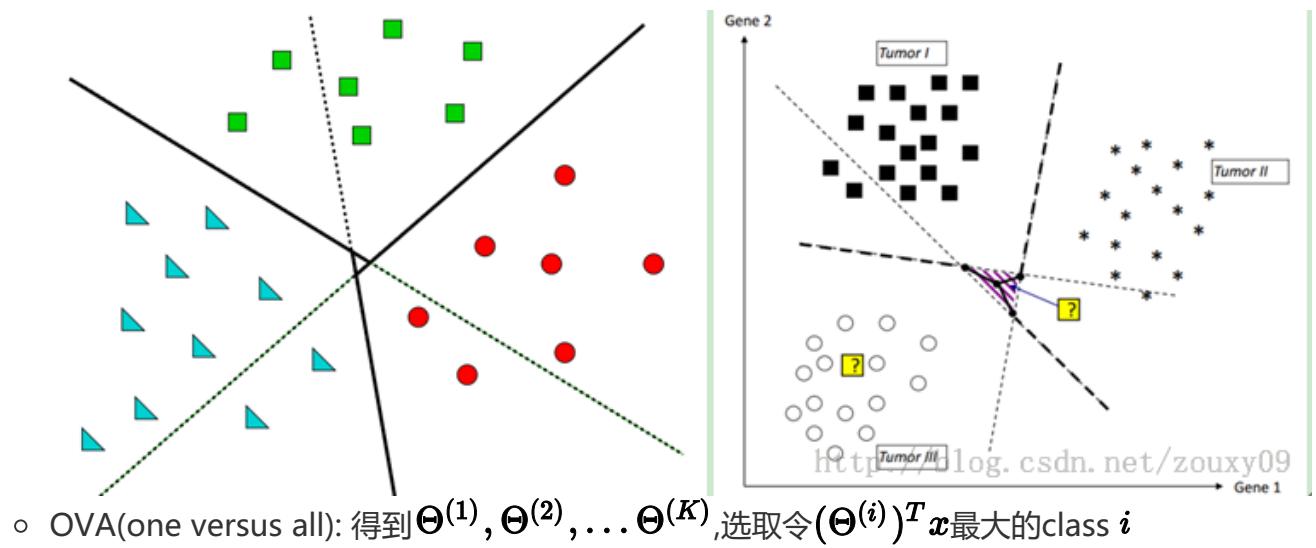
- 总结下 Kernel trick 的步骤：
 - 使用核函数将原始数据 x 变换到另一个特征空间 f , 令其线性可分;
 - 在新特征空间 f 中使用 SVM 进行学习分类

7.3.4 Parameters in SVM

- $C = \frac{1}{\lambda}$: 与之前 λ 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 $\sigma^2 \rightarrow$ smoother \rightarrow low variance, high bias
 - Small $\sigma^2 \rightarrow$ steeper \rightarrow high variance, low bias

7.4 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



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

```

1.  svmfit=svm(y~,data=dat,
2.                kernel="linear", #是no kernel
3.                #other kernels: polynomial/radial basis/tanh
4.                cost=10, #tuning parameter C
5.                scale=FALSE,
6.                type='c' #classification
7.      )

```

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

```

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

```

7.5 LR VS SVM

- 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 probabilities		✓
nonlinear boundaries	✓	

+ Neural network适用于以上大部分情况,但难以优化且可能得到的只是局部最优(black box)

7.6 Quiz

- C 和 σ^2 的影响;
- m 和 n 的影响;
- Gaussian kernel $\in [0,1]$;
- Gaussian kernel前应进行feature scaling;
- OVO / OVA

Week 8

8.1 Clustering

8.1.1 Unsupervised Learning: Introduction

- Training set: data without labels
- Application of clustering:
 - Market segmentation;
 - Social network analysis;
 - Astronomical data analysis

8.1.2 K-Means Algorithm

- 基本思路

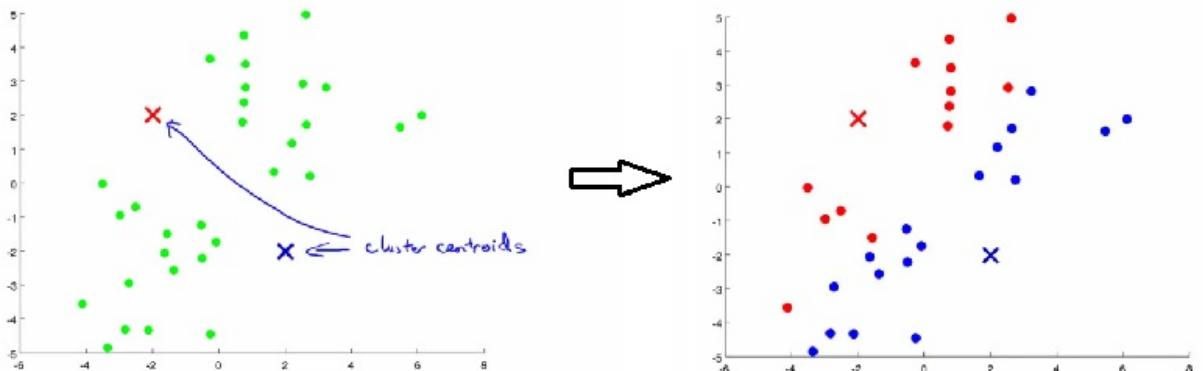
- 预先确定cluster number K ;
- 随机指定 K 个点作为cluster centroids;
- 根据其余点同质心的接近程度进行分类，得到 K 个聚类;
- 每个聚类重新指定质心;
- 重新聚类，迭代以上步骤直至每个聚类内的点都接近质心

- 基本变量:

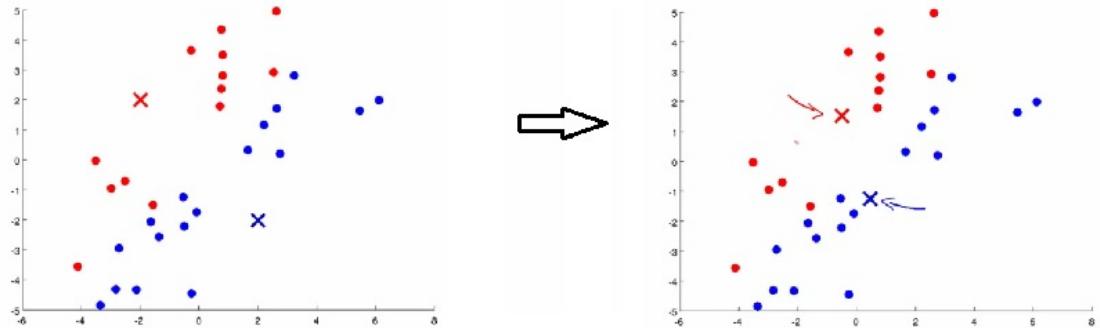
- The number of clusters: K ;
- Training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, 排除 $x_0 = 1$
- Randomly initialize K centroids: $\{\mu_1, \mu_2, \dots, \mu_K\}$;

- Repeat:

- for $i = 1, 2, \dots, m$: 将每个观察值都归类给对应的质心
 - $c^{(i)} := \text{the index}(1 \text{ to } K) \text{ of cluster closest to } x^{(i)}$
 - $c^{(i)} = \operatorname{argmin}_k \|x^{(i)} - \mu_k\|^2$ 欧氏距离平方
 - e.g. 假若最接近 $x^{(2)}$ 的是第5个质心 : $c^{(2)} = 5$



- for $k = 1, 2, 3, \dots, K$: 重新指定聚类质心
 - $\mu_k := \text{average(mean)} \text{ of points in cluster } k$
 - $\mu_k = \frac{1}{n} [x^{(k_1)} + x^{(k_2)} + \dots + x^{(k_n)}]$
 - 其中这 n 个 x 为在前一个循环被归类到cluster k 的训练数据



- 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--)

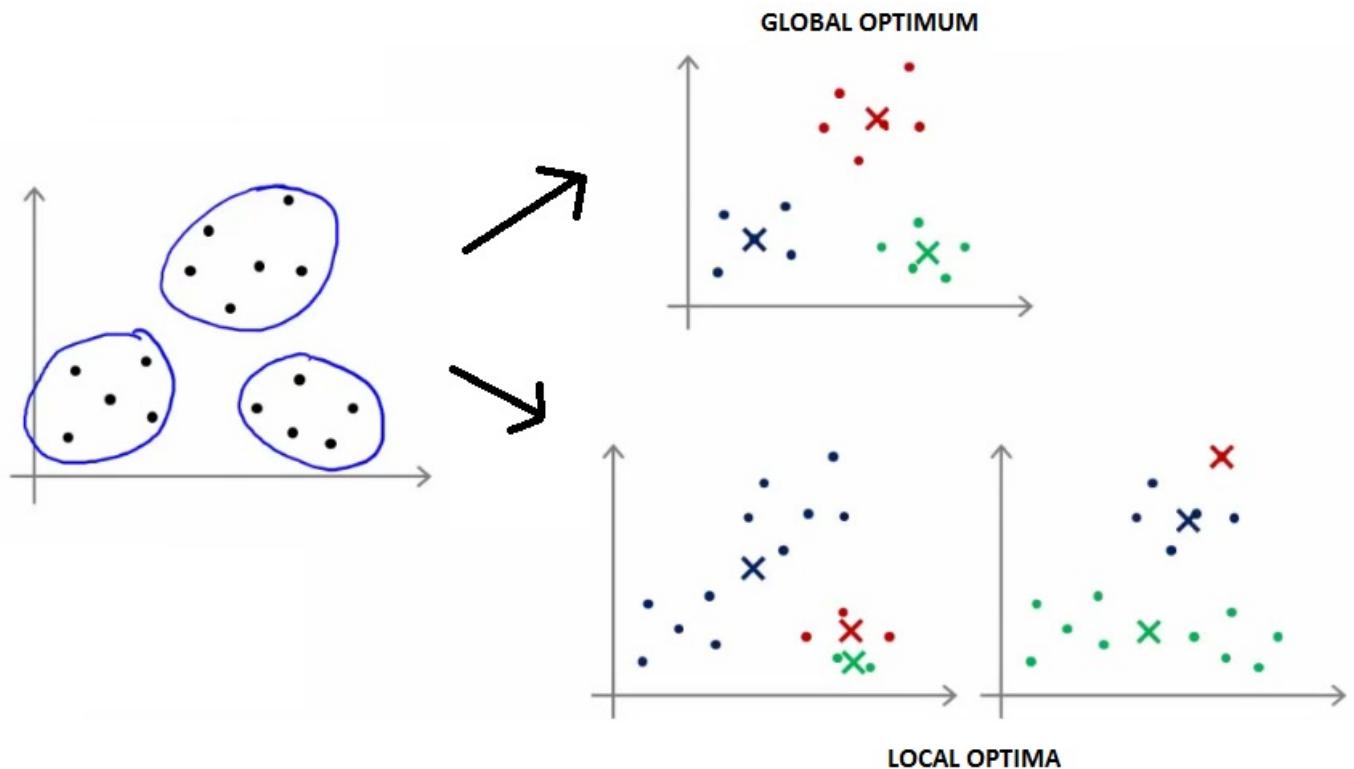
8.1.3 Optimization Objective

- $\mu_{c^{(i)}} : x^{(i)} = 5 \rightarrow c^{(i)} = 5 \rightarrow \mu_{c^{(i)}} = \mu_5$
- Cost function:

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

- In cluster assignment step, $\{\mu_1, \dots, \mu_K\}$ are fixed, the goal is to minimize J with $\{c^{(1)}, \dots, c^{(m)}\}$
- In move centroid step, the goal is to minimize J with $\{\mu_1, \dots, \mu_K\}$
- 注意在K-means中cost function是不断下降的

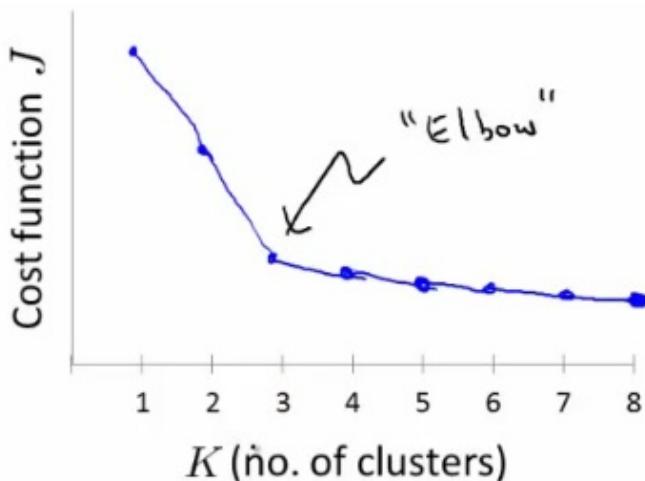
8.1.4 Random Initialization



- 注意K-means有可能得到的只是局部最优
 - 解决措施: 计算得到100个cost function ,然后取其中最小的 J
- for $i = 1, 2, \dots, 100$: 先得到100个cost function
 - Randomly initialize K-means;
 - Run K0means to get $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots \mu_K$;
 - Compute cost function: $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots \mu_K)$
- Pick clustering with lowest cost J

8.1.5 Choosing the Number of Clusters

- $K < m$ 避免产生空聚类
- 随机选择质心



- Elbow method:
 - 随着K增加 , cost J不断下降;
 - Elbow为下降速率骤然降低的点, 在这之后J之后仍下降 , 但速度缓慢
 - 因此可以通过观察Elbow来寻找适合的K
- 如果随着K增加, J不减反增, 说明J得到的是bad local optima, 需要重新初始化质心
- 有时Elbow并不适用, e.g. J下降全程十分平缓

8.1.6 Quiz

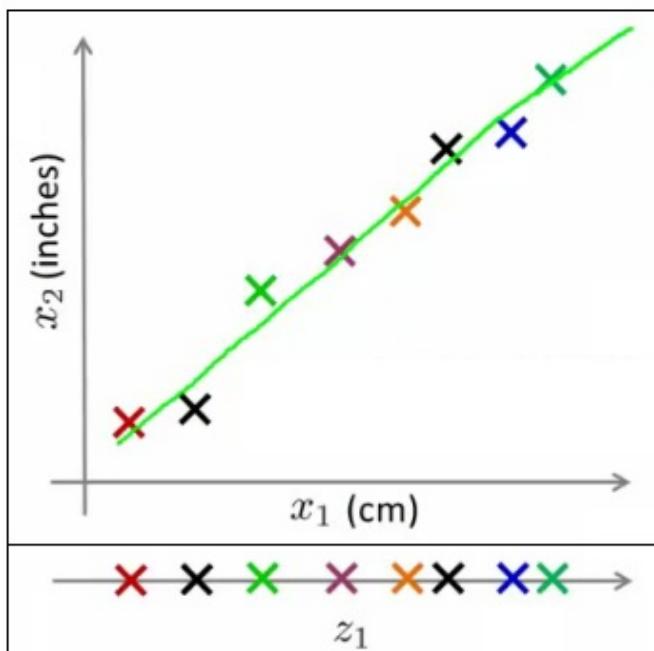
- 区分监督与非监督学习;
- $x^{(i)}, c^{(i)}, \mu_k$ 的关系;
- K-Means的预处理和循环处理步骤;
- 随机初始化的作用;

8.2 Dimensionality Reduction

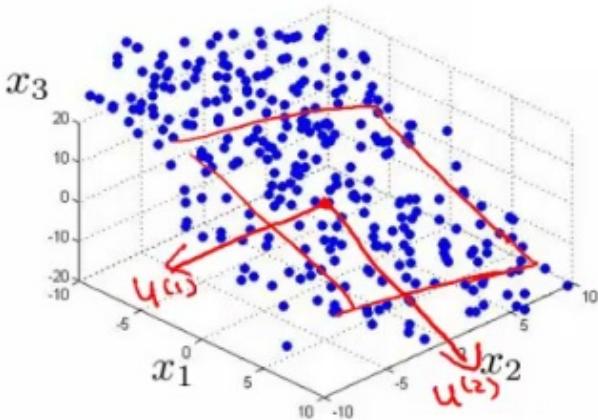
- 降低维度的目的:
 - 化简数据量, 优化算法效率
 - 使模型更易于可视化

8.2.1 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
- 这里举一个将二维数据压缩为一维的栗子:
 - 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



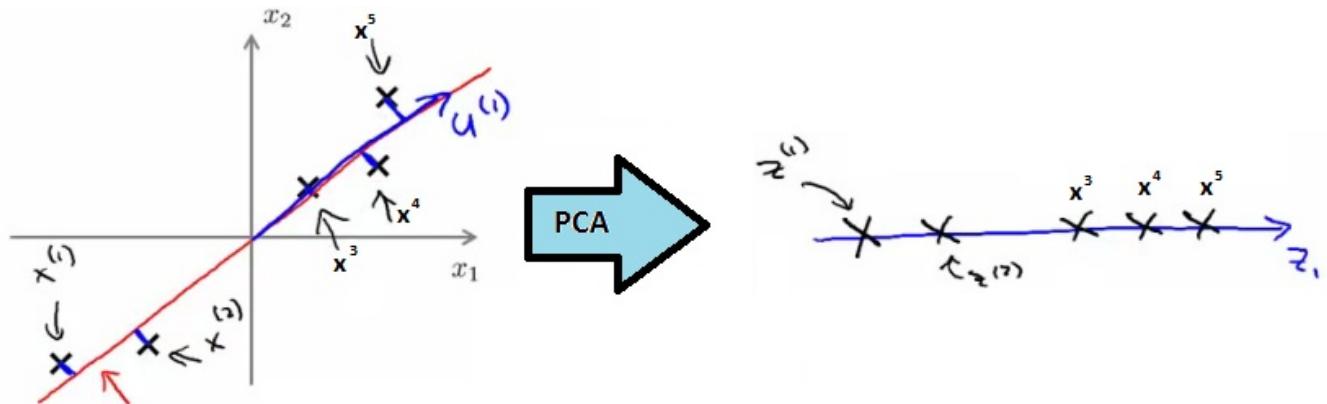
- 再举一个将三维数据压缩为二维的栗子:



- PCA与LR的不同之处：
 - Error
 - LR: **squared error** 为每个点到主成分线的纵坐标距离 $y^{(i)} - h(x^{(i)})$
 - PCA: **projection error** 为每个点到主成分线上投影点的欧氏距离

$$\sqrt{\sum (Vec_A - Vec_B)^2}$$
 - 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

8.2.2 PCA Algorithm



- Preprocessing: feature scaling & mean normalization
 - Training set: $x^{(1)}, x^{(2)}, \dots, x^{(m)}$, $x^{(i)}$ 为 $n \times 1$ 向量
 - $\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$
 - $x_j^{(i)} := x_j^{(i)} - \mu_j$, 此为必要步骤
 - $x_j^{(i)} := \frac{x_j^{(i)} - \mu_j}{s_j}$, 此为可选步骤

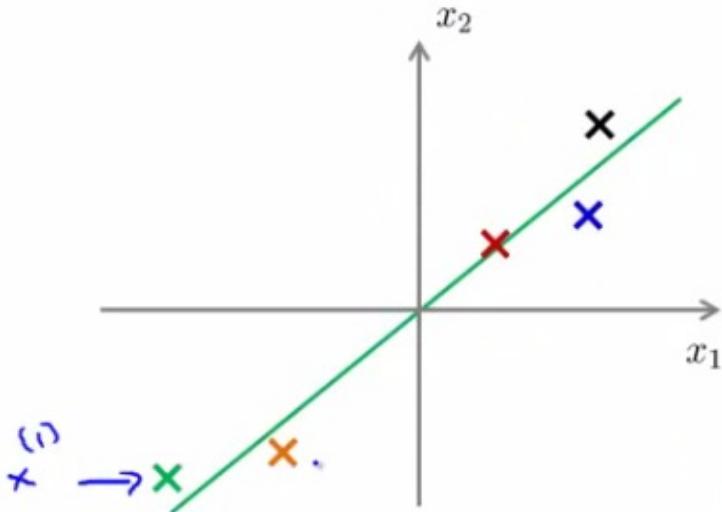
- PCA algorithm: $x^{(i)} \in R^2 \rightarrow z^{(i)} \in R$
 - 主要任务:
 - 找到k个主成分: $u^{(1)}, u^{(2)}, \dots, u^{(k)}$
 - 将之前的训练集投影为: $z^{(1)}, z^{(2)}, \dots, z^{(m)}$
 - Step 1 : 计算得到协方差矩阵
 - $\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)}) (x^{(i)})^T$
 - $(n \times 1) \times (1 \times n)$
 - Step 2 : 计算得到协方差矩阵的特征向量
 - $[U, S, V] = \text{svd}(\Sigma)$
 - SVD(singular value decomposition), 可以参考[链接](#)
 - Step 3 : 使用U矩阵的前k列计算z
 - $U = \{u^{(1)}, u^{(2)}, \dots, u^{(n)}\}$ 为 $n \times n$ 矩阵
 - 选取其中前k个元素作为主成分, 得到矩阵 $U_{reduce, n \times k} = U[:, 1:k]$
 - 计算投影后的新训练数据 $z^{(i)} = U_{reduce}^T * x^{(i)}$

```

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

```

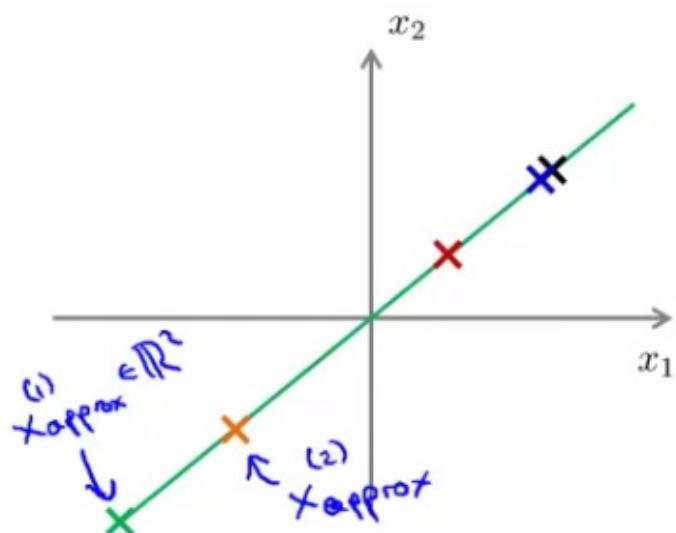
8.2.3 Reconstruction from Compressed Representation



$$z = U_{reduce}^T x$$



- 将压缩后的k维数据返回到原来的n维
- $x_{approx}^{(1)} \approx U_{reduce} * z^{(1)}$
 - $(n \times k) \times (k \times 1) \rightarrow n \times 1$
- 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



8.2.4 Choosing the Number of Principle Components #K

- 根据(A)Average squared projection error: $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$
- 以及(T)Total var : $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$
- 选取令 $\frac{A}{T}$ 尽可能小的k : 消除99%的variance

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

- 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 $\frac{A}{T}$
 - $S = diag[S_{11}, S_{22}, \dots, S_{nn}]$
 - 可将 $\frac{A}{T}$ 化简为:

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^n S_{ii}} \geq 0.99$$

8.2.5 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，得到的 $z^{(i)}$ 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

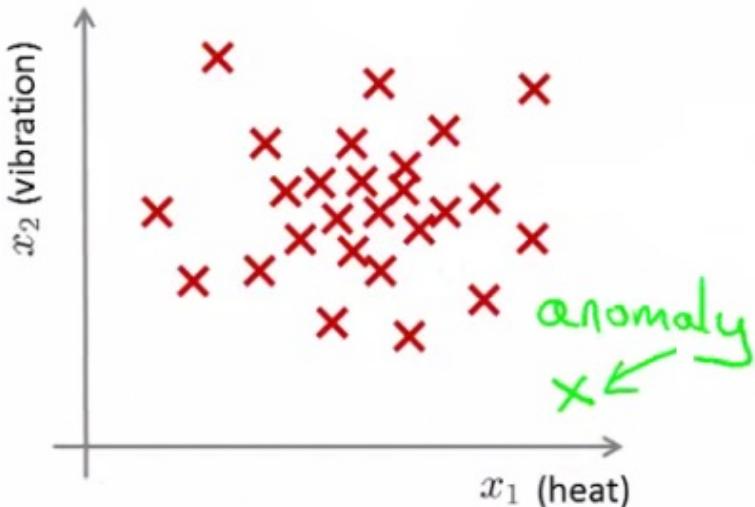
8.2.6 Quiz

- 同一条PC line包含两个向量
- K selection:
 - retain variance
 - $\frac{A}{T}$
- x与z的相互转化
- 注意PCA与上一节的K-Means的区别,PCA不需要多级随机初始化

- mean normalization--> necessary
- feature scaling--> optional
- PCA的应用

Week 9

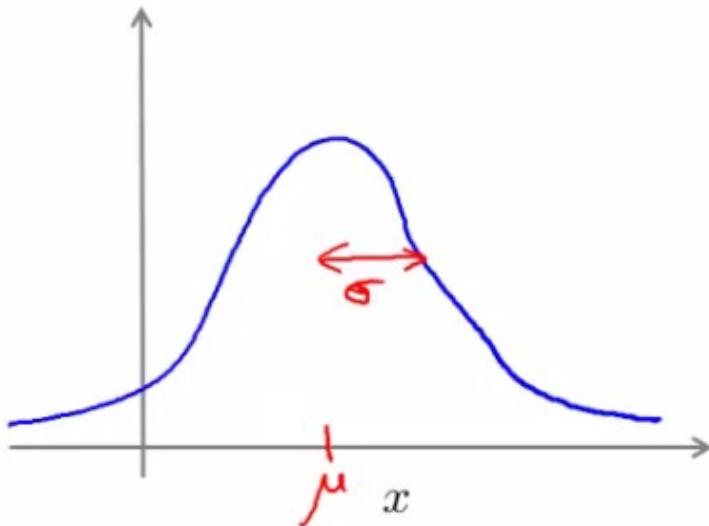
9.1 Anomaly Detection



9.1.1 Problem Motivation

- 新样本出现在训练集样本越集中(密度大)的地方，出错可能性越小
- 举个栗子: fraud detection
 - $\mathbf{x}^{(i)}$: features of user i 's activities
 - Model $p(\mathbf{x})$ from data
 - If $p(\mathbf{x}) < \epsilon$: 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

9.1.2 Gaussian Distribution



- $x \sim N(\mu, \sigma^2)$

- $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$: mean
- $\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$: Var
- σ : standard deviation

$$P(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

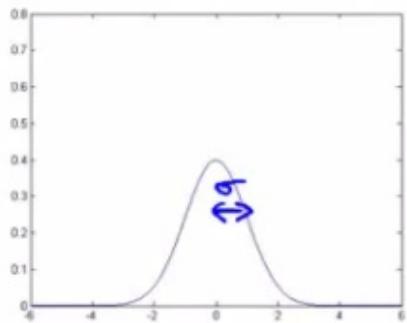
- 注意高斯分布函数与之前SVM中的高斯核不同

$$\text{Gaussian kernel} = e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

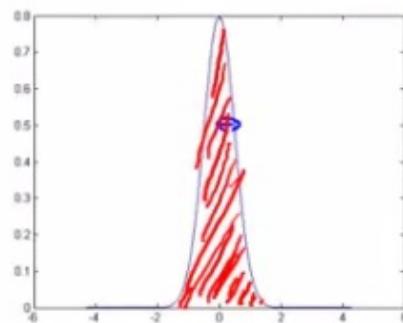
- 高斯核中σ的变化趋势与高斯分布刚好相反

- 一些高斯分布函数的栗子:

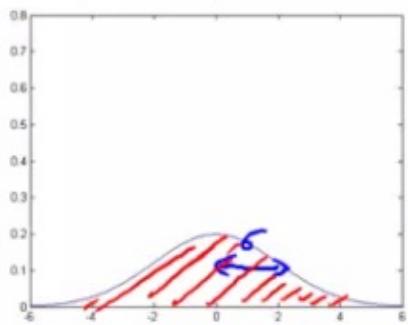
$$\mu = 0, \sigma = 1$$



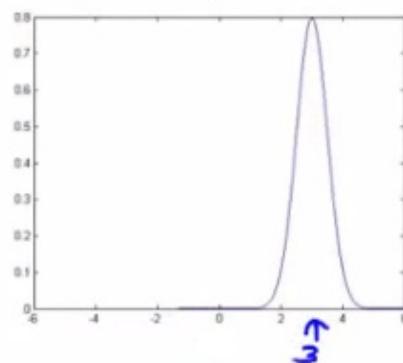
$$\mu = 0, \sigma = 0.5$$



$$\mu = 0, \sigma = 2$$

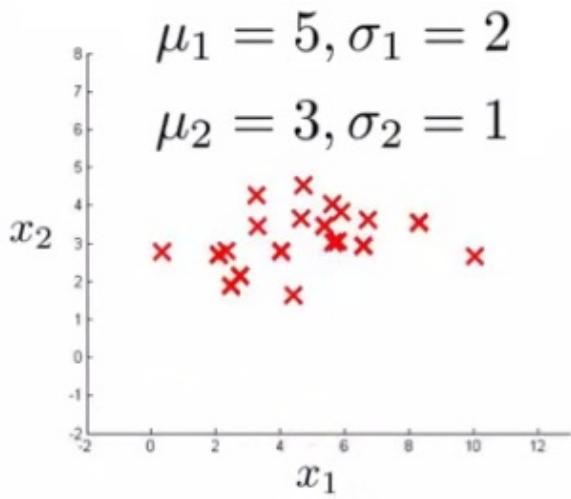


$$\mu = 3, \sigma = 0.5$$

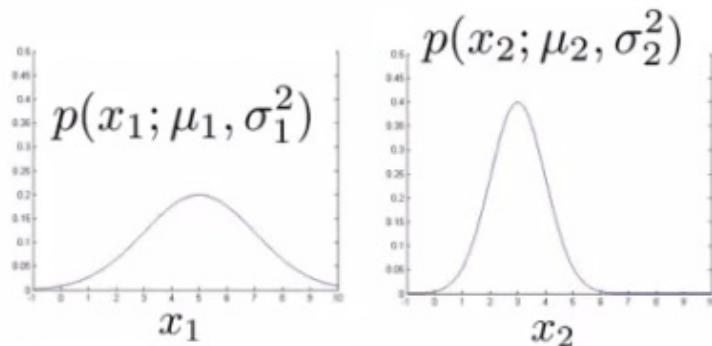


9.1.3 Algorithm

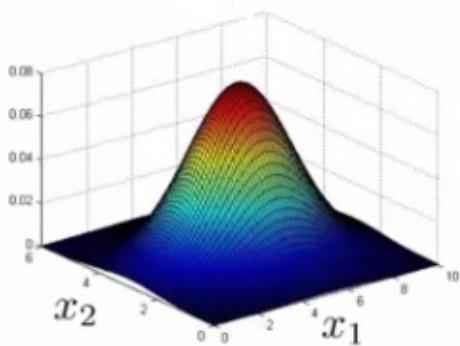
- Density estimation
 - Training set: $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
 - $p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$: 不同维度密度函数的乘积
- Anomaly detection algorithm
 - Choose the feature $x^{(i)}$ which is likely to be anomalous;
 - Fit parameters $\mu_1, \mu_2, \dots, \mu_n, \sigma_1^2, \sigma_2^2, \dots, \sigma_n^2$;
 - Compute $p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi\sigma_j}} e^{-\frac{1}{2}(\frac{x_j-\mu_j}{\sigma_j})^2}$
 - If $p(x) < \epsilon$, it's anomalous
- 举个栗子:
 - 可能存在异常的特征:
 - $x_1, \mu = 5, \sigma = 2$
 - $x_2, \mu = 3, \sigma = 1$
 - 构建以下系统:



- 计算 x_1 和 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:

- $p(x_1^1_{\text{test}}) = 0.436 \geq \text{epsilon}$ (~40% chance it's normal)
 - Normal
 - $p(x_2^2_{\text{test}}) = 0.0021 < \text{epsilon}$ (~0.2% chance it's normal)
 - Anomalous

9.1.4 Developing and Evaluating an Anomaly Detection

- 典型的数据集分配:

- Training set: 6000 good to compute $p(\mathbf{x})$
- CV set: 2000 good($y=0$) + 10 anomalous($y=1$)

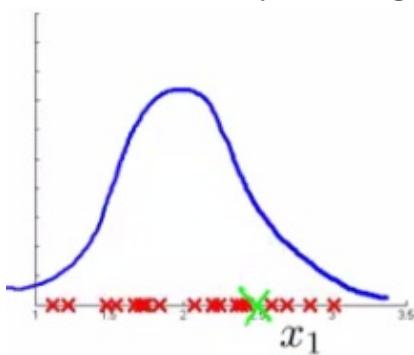
- Test set: 2000 good + 10 anomalous
- Algorithm evaluation
 - Fit model $p(x)$ on training set;
 - For a CV or test example x:
 - if $p(x) < \epsilon \rightarrow \text{anomaly} \rightarrow y = 1$
 - if $p(x) \geq \epsilon \rightarrow \text{normal} \rightarrow y = 0$
 - Possible evaluation metrics:
 - TP,TN,FP,FN
 - Precision/Recall
 - F_1 score
 - 不能用Accuracy: 不同features可能会出极端结果
 - Choose ϵ : CV

9.1.5 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

9.1.6 Choosing What Features to Use

- Non-gaussian features: make them like gaussian
 - $x \rightarrow \log(x + c)$
 - $x \rightarrow \sqrt{x}$
- Error analysis for anomaly detection:
 - Goal: more normal data ($y=0$), less anomalous data ($y=1$)
 - 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

9.1.7 Multivariate Gaussian Distribution

- 使用以下公式一次性获得所有 $p(x)$

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

- 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

9.1.8 Quiz

- 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;

9.2 Recommender Systems

9.2.1 Problem Formulation

- 举个栗子：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

9.2.2 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 $x_0 = 1$

- 因此电影i的特征向量 $\mathbf{x}^{(i)}$ 可表示为 3×1 向量, 例如

$$\mathbf{x}^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$$

- Recommending process:

- For each user j , learn a para $\theta^{(j)} \in R^3$;
- Predict j as rating movie i with $(\theta^{(j)})^T \mathbf{x}^{(i)}$
- 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

- 举个栗子: 预测 user 1 对 movie 3 的评分

- $\theta^{(1)}$: 这个在之后会解释

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

- $\mathbf{x}^{(3)}$:

$$\mathbf{x}^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix}$$

- 因此预测评分为:

$$(\theta^{(1)})^T \mathbf{x}^{(3)} = (0 * 1) + (5 * 0.99) + (0 * 0) = 4.95$$

- 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 $\frac{1}{m}$

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

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

9.2.3 Collaborate Filtering

- 对于并不知道特征向量x的情况:

- 猜解 θ , 根据 θ 求 x

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

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

- 根据 x 优化 θ

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

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

- 再用 θ 反过来优化 x...

- 这里还是用之前电影的栗子

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 & Bob 偏好 Romance, Carol & Dave 偏好 Action
- 假定 θ 已知

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

- 例如, 对于电影"Love at last", Alice&Bob偏好而Carol&Dave不喜欢
 - $(\theta^{(1)})^T x^1 \approx 5$
 - $(\theta^{(2)})^T x^1 \approx 5$
 - $(\theta^{(3)})^T x^1 \approx 0$
 - $(\theta^{(4)})^T x^1 \approx 0$
 - 我们可以推测 $x^{(1)} = [1 \ 1 \ 0]^T$
- 同理可以根据已有 θ 推定其他电影的类型
- 协同过滤算法:

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

- 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 $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}, \theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$
- 使用梯度下降优化 $J(x, \theta)$

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

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

- 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

9.2.4 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
 - 举个栗子: 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	?	?

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

- 正确做法为正规范化, 首先求取各行已知数据的平均值

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

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

- 然后将Y矩阵每个元素减去平均值, 变换为

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

- 最后预测结果为: $(\theta^{(j)})^T x^{(i)} + \mu_i$
- 例如对于User 5:
 - $\theta^{(5)} = [0 \ 0]^T \rightarrow (\theta^{(5)})^T x^{(i)} = 0$, 但加上 μ 后即可得到预测评分数据
- 注意不用再进行scaling: all the scales are similar

9.2.5 Quiz

- 注意J的几种表示形式;
- 适合使用推荐系统的情况, 注意推荐系统一定要有他人信息;
- 协同过滤算法
 - 注意初始化方式
 - 和线性回归对比不需要 $\frac{1}{m}$

- 除了梯度下降, 还可以使用其他算法进行优化
- 未评分数据($r(i, j) = 0$), 依旧可以使用协同过滤
- 求取 $r(i, j) == 1$ 数据的加和

```

1. total=sum(C.*R);
2. #等价于
3. total=sum(C(R==1));

```

Week 10 Large Scale ML

10.1 GD with Large Datasets

- 复习下WEEK 6样本量与误差的关系
- 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

10.2 Stochastic GD: 可以参考3.3

- 相比BGD更适合处理大量数据
- SGD VS BGD:
 - SGD:
 - 最小化每条样本的损失函数;
 - 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:
 - 最小化所有样本的损失函数;
 - 得到全局最优解;
- Mini-Batch GD
 - BGD: use all m examples in each iteration
 - SGD: use 1 example in each iteration
 - Mini BGD : use b examples a time
- Stochastic GD Convergence : 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;
 - $J = \frac{1}{2m} \sum (h - y)^2$

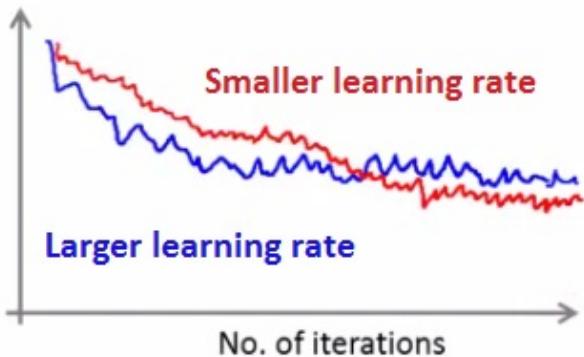
- SGD:

- $cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$
- 在更新 θ 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

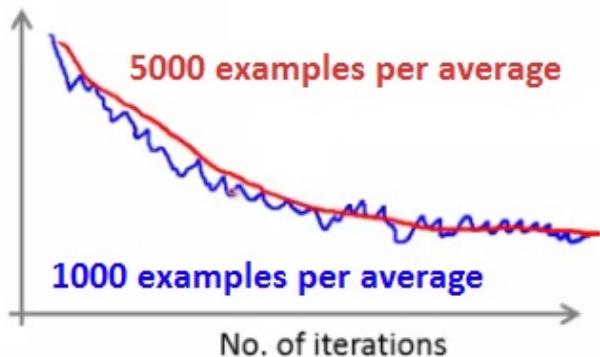
$$\alpha = \frac{cost1}{iterationNumber + cost2}$$

- 几种情况:

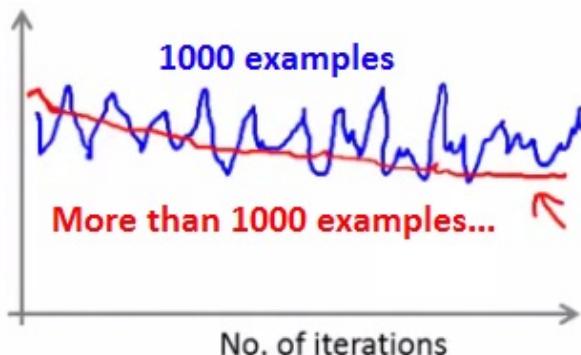
- 更小的学习速率: 震荡较小



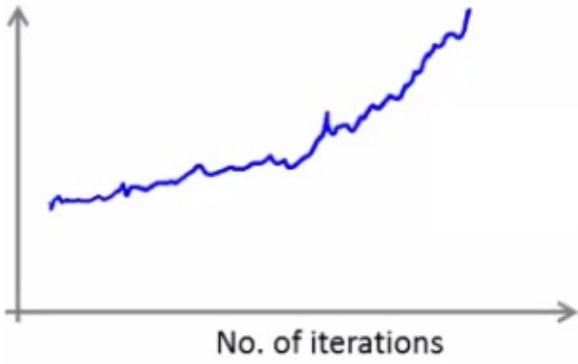
- 平均样本越大, 曲线越平滑



- 噪音过大, 可使用更大的平均样本量进行调节



- 曲线上升: 需要使用更小的学习速率



10.3 Online Learning

- 根据新的样本，边学习，边给出结果
- 不需要保存所有数据，用后即丢弃
- Goal: learn $p(y = 1|x; \theta)$ to optimize
- Repeat:
 - Get (x, y) corresponding to user
 - Update θ using (x, y) :
 - for $j \in \{0, 1, \dots, n\}$:
 - $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j$

10.4 Map Reduce and Data Parallelism

- 复习之前在python中学的map函数和reduce函数
 - map: 对每个元素使用特定函数
 - reduce: 迭代调用序列中的元素
- 举个栗子: devide and combine
 - 训练集 $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(400)}, y^{(400)})\}$
 - 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

10.5 Quiz

- 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

Week 11 Application Example: Photo OCR

- 这里给出了OCR的栗子, 我先行略过咯 :D