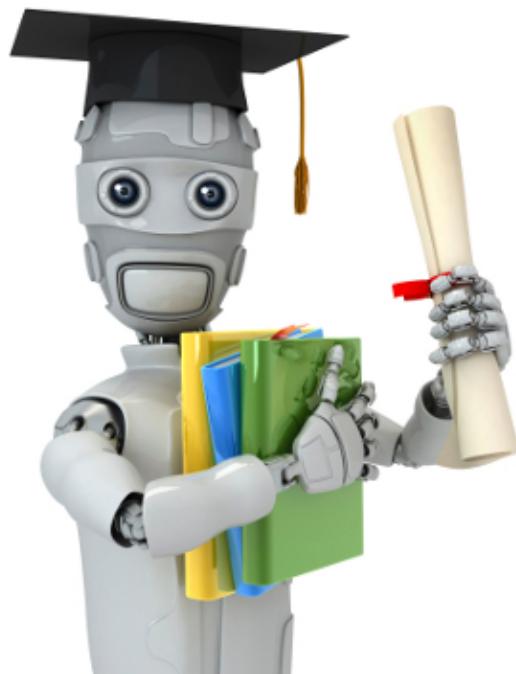


# COURSERA 机器学习课笔记

*by Prof. Andrew Ng*



Notes by Ryan Cheung

Ryanzjlib@gmail.com

Weibo@小小人\_V

## 目录

目录 .....	1
Week1 机器学习介绍 .....	7
1 机器学习介绍 .....	7
1.1 什么是机器学习? .....	7
1.2 监督学习 ( Supervised Learning ) .....	7
1.3 非监督学习 ( Unsupervised Learning ) .....	9
Week1 单变量线性回归 .....	11
2 单变量线性回归 ( Linear Regression with One Variable ) .....	11
2.1 模型表达 ( Model Representation ) .....	11
2.2 代价函数 ( Cost Function ) .....	12
2.3 梯度下降 ( Gradient Descent ) .....	13
2.4 对线性回归运用梯度下降法 .....	13
Week2 多变量线性回归 .....	15
3 多变量线性回归 ( Linear Regression with Multiple Variables ) .....	15
3.1 多维特征 ( Multiple Features ) .....	15
3.2 多变量梯度下降 ( Gradient descent for multiple variables ) .....	16
3.3 特征缩放 ( feature scaling ) .....	16
3.4 学习率(Learning rate) .....	17
Week2 多项式回归和正规方程 .....	19
4 多项式回归和正规方程 .....	19

4.1 多项式回归 ( Polynomial Regression ) .....	19
4.2 正规方程 ( Normal Equation ) .....	19
Week3 归一化 .....	21
5 逻辑回归 ( Logistic Regression ) .....	21
5.1 分类问题.....	21
5.2 分类问题建模.....	21
5.3 判定边界 ( Decision Boundary ) .....	23
5.4 代价函数.....	24
5.5 多类分类 ( Multiclass Classification ) .....	26
Week3 归一化 .....	28
6 归一化 ( Regularization ) .....	28
6.1 过拟合问题 ( The Problem of Overfitting ) .....	28
6.2 归一化代价函数 ( Regularization Cost Function ) .....	29
6.3 归一化线性回归 ( Regularized Linear Regression ) .....	30
6.4 归一化逻辑回归 ( Regularized Logistic Regression ) .....	31
Week4 神经网络 : 表达 .....	32
7 神经网络 : 表达 .....	32
7.1 非线性假设 ( Non-Linear Hypothesis ) .....	32
7.2 神经网络介绍.....	32
7.3 模型表达.....	33
7.4 神经网络模型表达.....	34
7.5 正向传播 (Forward Propagation) .....	35

7.6 对神经网络的理解.....	36
7.7 神经网络示例：二元逻辑运算符（Binary Logical Operators）.....	36
7.8 多类分类.....	37
Week5 神经网络：学习.....	39
8 神经网络：学习 .....	39
8.1 神经网络代价函数.....	39
8.2 反向传播算法（Backpropagation Algorithm）.....	39
8.3 梯度检验（Gradient Checking） .....	42
8.4 随机初始化（Random Initialization） .....	43
8.5 综合起来.....	43
Week6 机器学习应用建议.....	45
9 机器学习应用建议.....	45
9.1 决定下一步做什么.....	45
9.2 假设的评估（Evaluating a Hypothesis） .....	45
9.3 模型选择（交叉验证集） .....	46
9.4 偏倚和偏差诊断（Diagnosis Bias vs. Variance） .....	46
9.5 归一化与偏倚/偏差 .....	47
9.6 学习曲线（Learning Curves） .....	49
9.7 决定下一步做什么.....	50
Week6 机器学习系统设计.....	52
10 机器学习系统设计.....	52
10.1 首先要做什么.....	52

10.2 误差分析 ( Error Analysis ) .....	52
10.3 类偏斜的误差度量 ( Error Metrics for Skewed Classes ) .....	53
10.4 查全率和查准率之间的权衡.....	54
10.5 机器学习的数据.....	55
Week7 支持向量机.....	56
11 支持向量机 ( Support Vector Machine ) .....	56
11.1 优化目标 ( Optimization Objective ) .....	56
11.2 支持向量机判定边界 ( SVM Decision Boundary ) .....	58
11.3 核函数 ( Kernels ) .....	59
11.4 逻辑回归与支持向量机.....	63
Week8 聚类.....	64
12 聚类 ( Clustering ) .....	64
12.1 K-均值算法 .....	64
12.2 优化目标.....	66
12.3 随机初始化.....	66
12.4 选择聚类数.....	66
Week8 降维.....	67
13 降维 ( Dimensionality Reduction ) .....	67
13.1 动机一：数据压缩 ( Data Compression ) .....	67
13.2 动机二：数据可视化 ( Data Visualization ) .....	68
13.3 主要成分分析 ( Principal Component Analysis ) .....	68
13.4 主要成分分析算法.....	69

13.5 选择主要成分的数量.....	70
13.6 应用主要成分分析.....	71
Week9 异常检测 .....	73
14 异常检测 ( Anomaly Detection ) .....	73
14.1 密度估计 ( Density Estimation ) .....	73
14.2 高斯分布.....	74
14.3 异常检测.....	74
14.4 评价一个异常检测系统.....	75
14.5 异常检测与监督学习对比.....	75
14.6 选择特征.....	76
14.7 多元高斯分布(Multivariate Gaussian Distribution) .....	77
Week9 推荐系统 .....	80
15 推荐系统 ( Recommender Systems ) .....	80
15.1 问题形式化.....	80
15.2 基于内容的推荐系统 ( Content-based Recommendations ) .....	80
15.3 协同过滤算法 ( Collaborative Filtering Algorithm ) .....	82
15.4 均值归一化.....	83
Week10 大规模机器学习 .....	84
16 大规模机器学习 ( Large Scale Machine Learning ) .....	84
16.1 大型数据集的学习.....	84
16.2 随机梯度下降法 ( Stochastic Gradient Descent ) .....	84
16.3 微型批量梯度下降 ( Mini-Batch Gradient Descent ) .....	85

16.4 随机梯度下降收敛 ( Stochastic Descent Convergence ) .....	86
16.5 在线学习 ( Online Learning ) .....	87
16.6 映射化简和数据并行 ( Map Reduce and Data Parallelism ) .....	88
Week10 应用示例:图像文字识别.....	89
17 应用实例 : 图像文字识别 ( Photo OCR ) .....	89
17.1 问题描述和流程图 ( Problem description and pipeline ) .....	89
17.3 获得大量数据个人工数据.....	91
17.4 上限分析 ( Ceiling Analysis ) .....	92

## WEEK1 机器学习介绍

### 1 机器学习介绍

#### 1.1 什么是机器学习？

对于机器学习，并没有一个一致认同的定义，一个比较古老的定义是由 Arthur Samuel 在 1959 年给出的：

“机器学习研究的是如何赋予计算机在没有被明确编程的情况下仍能够学习的能力。  
(Field of study that ~~gives~~ gives computers the ability to learn without being explicitly programmed.)”

Samuel 编写了一个跳棋游戏的程序，并且让这个程序和程序自身玩了几万局跳棋游戏，并且记录下来棋盘上的什么位置可能会导致怎样的结果，随着时间的推移，计算机学会了棋盘上的哪些位置可能会导致胜利，并且最终战胜了设计程序的 Samuel.

另一个比较现代且形式化的定义是由 Tom Mitchell 在 1998 年给出的：

“对于某个任务 T 和表现的衡量 P，当计算机程序在该任务 T 的表现上，经过 P 的衡量，随着经验 E 而增长 我们便称计算机程序能够通过经验 E 来学习该任务。( computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.)”

在跳棋游戏的例子中，任务 T 是玩跳棋游戏，P 是游戏的输赢，E 则是一局又一局的游戏。

一些机器学习的应用举例：

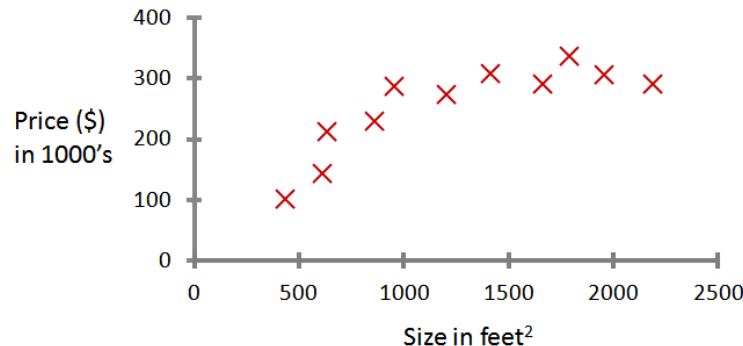
1. 数据库挖掘
2. 一些无法通过手动编程来编写的应用：如自然语言处理，计算机视觉
3. 一些自助式的程序：如推荐系统
4. 理解人类是如何学习的

#### 1.2 监督学习 ( SUPERVISED LEARNING )

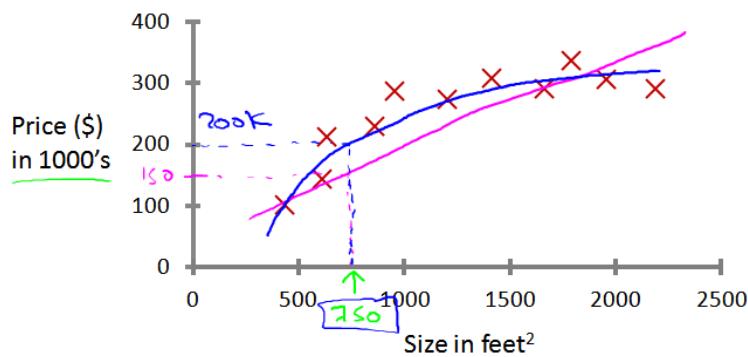
在课程稍后阶段我们再给监督学习一个更加正式的定义，现在我们从一个例子开始：

假设你有下面这些房价数据，图表上的每个实例都是一次房屋交易，横坐标为交易房屋的占地面积，纵坐标为房屋的交易价格。

Housing price prediction.



现在，假设你希望能够预测一个 750 平方英尺的房屋的交易价格可能是多少。一种方法是根据这些数据点的分布，画一条合适的直线，然后根据这条直线来预测。在房价预测这个例子中，一个二次函数可能更适合已有的数据，我们可能会更希望用这个二次函数的曲线来进行预测。

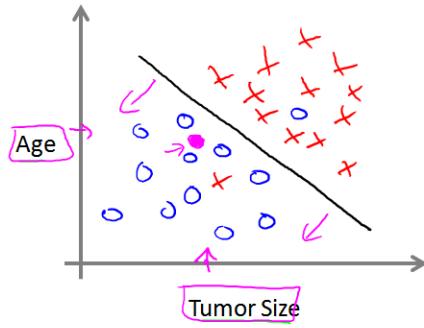


我们称这样的学习为监督学习。称其为监督式的学习，因为我们预先给了算法“正确结果”——即所有我们观察到的变量。

上面这个问题又称为回归问题 ( Regression )，因为我们能预测的结果是连续地值。

再来看另一种类型的监督学习问题：

假使你希望预测一个乳腺癌是否是恶性的，你现在有的数据是不同年龄的病人和她们身上肿瘤的尺寸以及这些肿瘤是否是恶性的。如果我们将这些信息绘制成一张 2D 图表，以横坐标为肿瘤的尺寸，以纵坐标为病人的年龄，以 O 代表良性肿瘤，以 X 代表恶性肿瘤。则我们的算法要学习的问题就变成了如何分割良性肿瘤和恶性肿瘤。



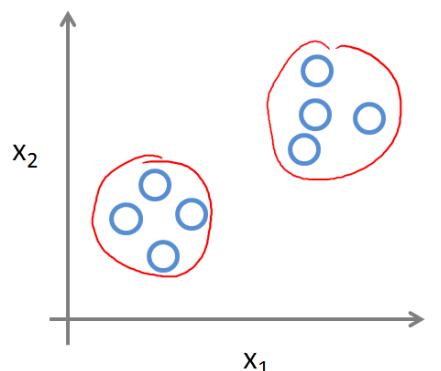
这样的问题是**分类问题** (Classification)，我们希望算法能够学会如何将数据分类到不同的类里。

上面的例子中我们只适用了两个**特征** (features) 来进行分类，现实中，我们会有非常多的特征，并且我们希望算法能够处理**无限多数量的特征**，在课程后面我们会介绍能够处理这样问题的算法，例如**支持向量机** (Support Vector Machine)。

### 1.3 非监督学习 (UNSUPERVISED LEARNING)

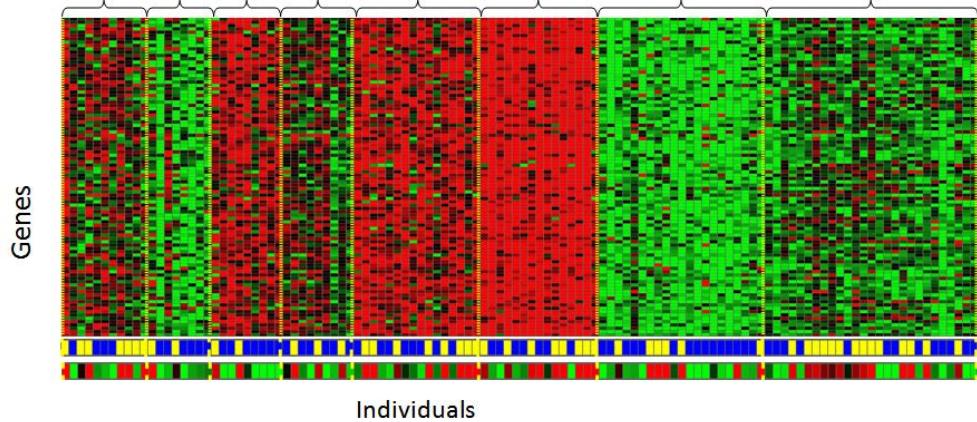
在监督学习中，无论是回归问题还是分类问题，我们的数据都具有一个结果 (房价问题中的房价，肿瘤问题中的良性与否)。

而在非监督学中，我们的现有数据中并没有结果，我们有的只是特征，因而**非监督学习要解决的问题是发现这些数据是否可以分为不同的组。**



非监督学习的一个例子是**聚类问题** (Clustering)，例如对一个大型的数据中心的网络传输数据情况进行分析，发现那些多数时候是在协作的计算机。

再一个例子，给定一些人和他们所有的基因，非监督学习可以根据是否具有某些基因而将这些人聚类：

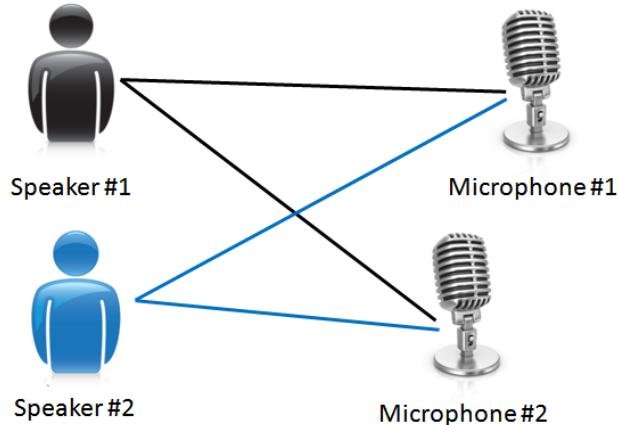


图中纵坐标为一个个人以及他们的基因，横坐标为各种类型的基因。

再一个非监督学习问题的例子是**鸡尾酒会问题** (Cocktail Party Problem)，在一个满是人的房间中，人们都在互相对话，我们使用一些麦克风录下房间中的声音，利用非监督学习算法来识别房间中某一个人所说的话。

鸡尾酒会问题的一个简化版本是一个房间中有两个人在同时在讲话，利用两个麦克风来录音。

## Cocktail party problem



下面这个只有一行的机器学习算法 (Octave) 可以非常漂亮地将两个人的说话给分离开来：

```
[W, s, v] = svd((repmat(sum(x.*x, 1), size(x, 1), 1).*x)*x');
```

## WEEK1 单变量线性回归

### 2 单变量线性回归 ( LINEAR REGRESSION WITH ONE VARIABLE )

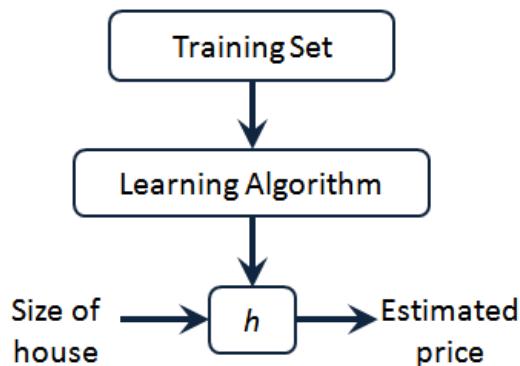
#### 2.1 模型表达 ( MODEL REPRESENTATION )

以之前的房屋交易问题为例，假使我们回归问题的训练集 ( Training Set ) 如下表所示：

Size in feet <sup>2</sup> (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

我们将要用来描述这个回归问题的标记如下：

- m 代表训练集中实例的数量
- x 代表特征/输入变量
- y 代表目标变量/输出变量
- (x,y) 代表训练集中的实例
- (x<sup>(i)</sup>,y<sup>(i)</sup>) 代表第 i 个观察实例
- h 代表学习算法的解决方案或函数也称为假设 ( hypothesis )



因而，要解决房价预测问题，我们实际上是要将训练集“喂”给我们的学习算法，进而学习得一个假设  $h$ ，然后将我们要预测的房屋的尺寸作为输入变量输入给  $h$ ，预测出该房屋的交易价格作为输出变量输出为结果。

那么，对于我们的房价预测问题，我们该如何表达  $h$ ？

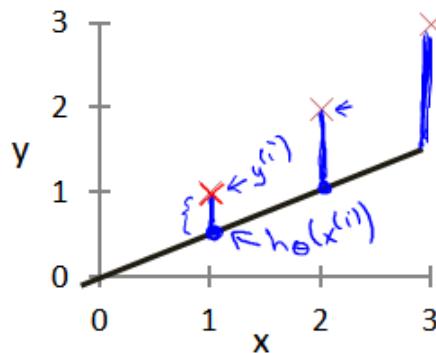
一种可能的表达方式为： $h_{\theta} = \theta_0 + \theta_1 x$

因为只含有一个特征/输入变量，因此这样的问题叫作单变量线性回归问题。

## 2.2 代价函数 ( COST FUNCTION )

我们现在要做的便是为我们的模型选择合适的参数 ( parameters )  $\theta_0$  和  $\theta_1$ ，在房价问题这个例子中便是直线的斜率和在  $y$  轴上的截距。

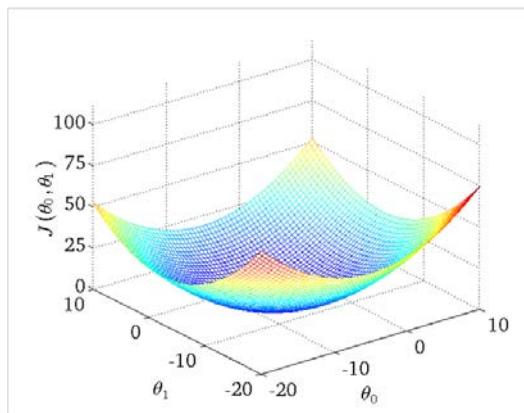
我们选择的参数决定了我们得到的直线相对于我们的训练集的准确程度，模型所预测的值与训练集中实际值之间的差距 ( 下图中蓝线所指 ) 就是建模误差 ( modeling error )。



我们的目标便是选择出可以使得建模误差的平方和能够最小的模型参数。

即使得代价函数  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$  最小。

我们绘制一个等高线图，三个坐标分别为  $\theta_0$  和  $\theta_1$  和  $J(\theta_0, \theta_1)$ ：

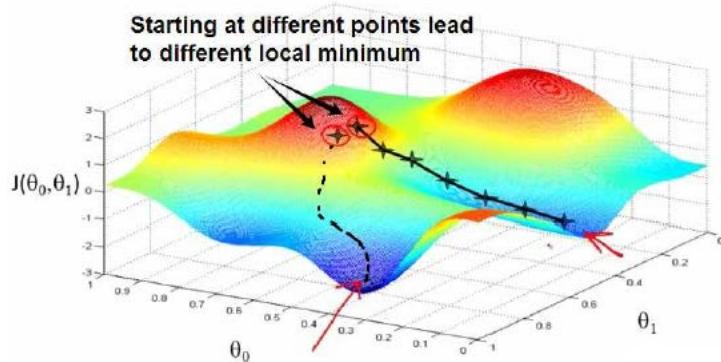


则可以看出在三维空间中存在一个使得  $J(\theta_0, \theta_1)$  最小的点。

### 2.3 梯度下降 ( GRADIENT DESCENT )

梯度下降是一个用来求函数最小值的算法，我们将使用梯度下降算法来求出代价函数  $J(\theta_0, \theta_1)$  的最小值。

梯度下降背后的思想是：开始时我们随机选择一个参数的组合 ( $\theta_0, \theta_1, \dots, \theta_n$ )，计算代价函数，然后我们寻找下一个能让代价函数值下降最多的参数组合。我们持续这么做直到到一个**局部最小值** ( local minimum )，因为我们并没有尝试完所有的参数组合，所以不能确定我们得到的局部最小值是否便是**全局最小值** ( global minimum )，选择不同的初始参数组合，可能会找到不同的局部最小值。



**批量梯度下降** ( batch gradient descent ) 算法的公式为：

```
repeat until convergence {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$     (for  $j = 0$  and  $j = 1$ )  
}
```

其中  $\alpha$  是**学习率** ( learning rate )，它决定了我们沿着能让代价函数下降程度最大的方向向下迈出的步子有多大，在批量梯度下降中，我们每一次都同时让所有的参数减去学习速率乘以代价函数的导数。

### 2.4 对线性回归运用梯度下降法

对我们之前的线性回归问题运用梯度下降法，关键在于求出代价函数的导数，即：

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

$$j=0 \text{ 时} : \frac{\partial}{\partial \theta_0} J(\theta_0 \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$j=1 \text{ 时} : \frac{\partial}{\partial \theta_1} J(\theta_0 \theta_1) = \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)})$$

则算法改写成：

Repeat {

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

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

}

## WEEK2 多变量线性回归

### 3 多变量线性回归 ( LINEAR REGRESSION WITH MULTIPLE VARIABLES )

#### 3.1 多维特征 ( MULTIPLE FEATURES )

目前为止，我们探讨了单变量/特征的回归模型，现在我们对房价模型增加更多的特征，例如房间数楼层等，构成一个含有多个变量的模型，模型中的特征为  $(x_1, x_2, \dots, x_n)$ 。

Size (feet <sup>2</sup> )	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

增添更多特征后，我们引入一系列新的注释：

- $n$  代表特征的数量
- $x^{(i)}$  代表第  $i$  个训练实例，是特征矩阵中的第  $i$  行，是一个向量 ( vector )。
- $x_j^{(i)}$  代表特征矩阵中第  $i$  行的第  $j$  个特征，也就是第  $i$  个训练实例的第  $j$  个特征。

支持多变量的假设  $h$  表示为：

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

这个公式中有  $n+1$  个参数和  $n$  个变量，为了使得公式能够简化一些，引入  $x_0=1$ ，则公式转化为：

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

此时模型中的参数是一个  $n+1$  纬的向量，任何一个训练实例也都是  $n+1$  纬的向量，特征矩阵  $X$  的纬度是  $m \times n+1$ 。

因此公式可以简化为：

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

其中上标 T 代表矩阵转置。

### 3.2 多变量梯度下降 ( GRADIENT DESCENT FOR MULTIPLE VARIABLES )

与单变量线性回归类似，在多变量线性回归中，我们也构建一个代价函数，则这个代价函数是所有建模误差的平方和，即：

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

我们的目标和单变量线性回归问题中一样，是要找出使得代价函数最小的一系列参数。

多变量线性回归的批量梯度下降算法为：

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \dots, \theta_n)$$

}

即：

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

}

求导数后得到：

Repeat {

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

( simultaneously update  $\theta_j$   
for  $j=0,1,\dots,n$  )

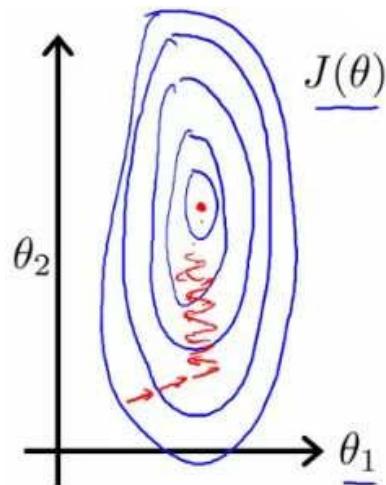
}

我们 开始随机选择一系列的参数直，计算所有的预测结果后，再给所有的参数一个新的直，如此循环直到收敛。

### 3.3 特征缩放 ( FEATURE SCALING )

在我们面对多维特征问题的时候，我们要保证这些特征都具有相近的尺度，这将帮助梯度下降算法更快地收敛。

以房价问题为例，假设我们使用两个特征，房屋的尺寸和房间的数量，尺寸的直为 0-2000 平方英尺，而房间数量的直则是 0-5，以两个参数分别为横纵坐标，绘制代价函数的等高线图能看出图像会显得很扁，梯度下降算法需要非常多次数的迭代才能收敛。



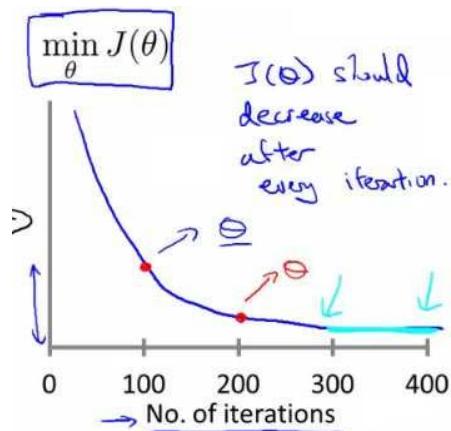
解决的方法是尝试将所有特征的尺度都尽量缩放到-1 到 1 之间。最简单的方法是令：

$$x_n = \frac{x_n - \mu_n}{s_n}$$

其中  $\mu_n$  是平均值， $s_n$  是标准差。

### 3.4 学习率(LEARNING RATE)

梯度下降算法收敛所需要的迭代次数根据模型的不同而不同，我们不能提前预知，我们可以绘制迭代次数和代价函数的图表来观测算法在何时趋于收敛。



也有一些自动测试是否收敛的方法，例如将代价函数的变化直与某个阀直（例如 0.001）进行比较，但通常看上面这样的图表更好。

梯度下降算法的每次迭代受到学习率的影响，如果学习率 $\alpha$ 过小，则达到收敛所需的迭代次数会非常高；如果学习率 $\alpha$ 过大，每次迭代可能不会减小代价函数，可能会越过局部最小值导致无法收敛。

通常可以考虑尝试这些学习率：

$$\alpha = 0.01, 0.3, 0.1, 0.3, 1, 3, 10$$

## WEEK2 多项式回归和正规方程

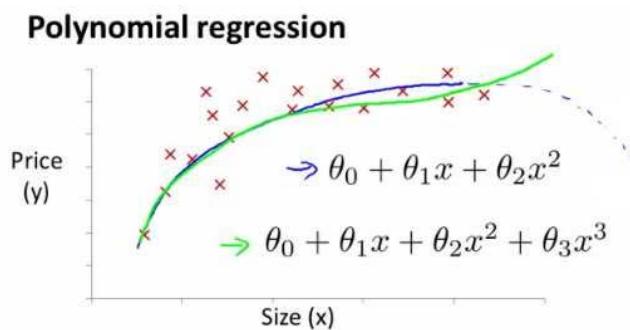
### 4 多项式回归和正规方程

#### 4.1 多项式回归 ( POLYNOMIAL REGRESSION )

线性回归并不适用于所有数据，有时我们需要曲线来适应我们的数据，比如一个二次方模型：

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$$

或者三次方模型： $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3$



通常我们需要先观察数据然后再决定准备尝试怎样的模型。

另外，我们可以令：

$$x_2 = x_2^2$$

$$x_3 = x_3^3$$

从而将模型转化为线性回归模型。注 如果我们采用多项式回归模型，在运行梯度下降算法前，特征缩放非常有必要。

#### 4.2 正规方程 ( NORMAL EQUATION )

到目前为止，我们都在使用梯度下降算法，但是对于某些线性回归问题，正规方程方法是更好的解决方案。

$$\frac{\partial}{\partial \theta_j} J(\theta_j) = 0$$

正规方程是通过求解下面的方程来找出使得代价函数最小的参数的：

假设我们的训练集特征矩阵为  $X$  (包含了  $x_0=1$ ) 并且我们的训练集结果为向量  $y$ , 则利用正规方程解出向量  $\theta = (X^T \times X)^{-1} \times X^T \times y$

上标 T 代表矩阵转置 , 上标-1 代表矩阵的逆。

以下表所示数据为例 :

$X(0)$	$X(1)$	$X(2)$	$X(3)$	$X(4)$	$y$
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

运用正规方程方法求解参数 :

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

在 Octave 中 , 正规方程写作 :  $\text{pinv}(X^T \times X) \times X^T \times y$

注 : 对于那些不可逆的矩阵 ( 通常是因为特征之间不独立 , 如同时包含英尺为单位的尺寸和米为单位的尺寸两个特征 , 也有可能是特征数量大于训练集的数量 ) , 正规方程方法是不能用的。

梯度下降与正规方程的比较 :

梯度下降	正规方程
需要选择学习率 $\alpha$	不需要
需要多次迭代	一次运算得出
当特征数量 $n$ 大时也能较好适用	如果特征数量 $n$ 较大则运算代价大 , 因为矩阵逆的计算时间复杂度为 $O(n^3)$
	通常来说当 $n$ 小于 10000 时还是可以接受的
适用于各种类型的模型	只适用于线性模型 , 不适合逻辑回归模型等其他模型

## WEEK3 归一化

### 5 逻辑回归 ( LOGISTIC REGRESSION )

#### 5.1 分类问题

在分类问题中，我们尝试预测的是结果是否属于某一个类（例如正确或错误）。分类问题的例子有：判断一封电子邮件是否是垃圾邮件；判断一次金融交易是否是欺诈等等。

我们从二元的分类问题开始讨论。

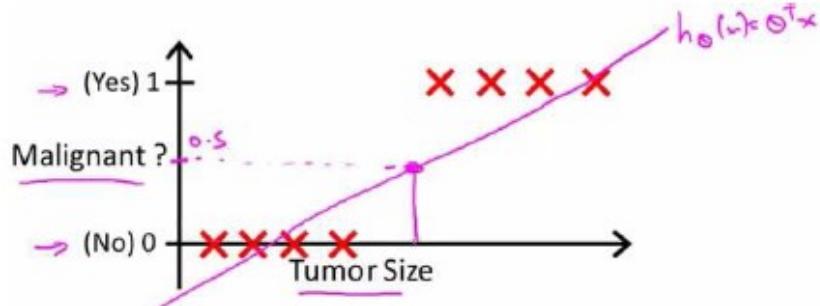
我们将**因变量**(dependant variable)可能属于的两个类分别称为**负向类** ( negative class ) 和**正向类** ( positive class )，则因变量

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

其中 0 表示负向类，1 表示正向类

#### 5.2 分类问题建模

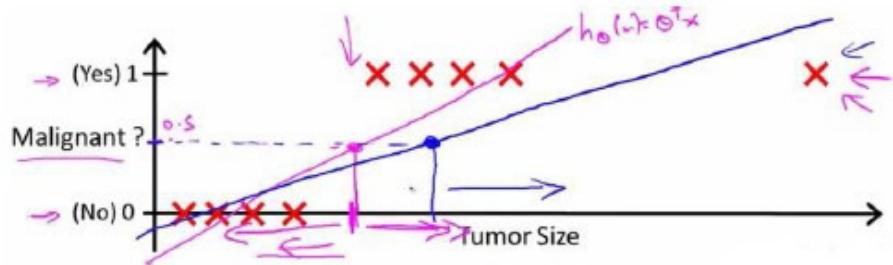
回顾在一开始提到的乳腺癌分类问题，我们可以用线性回归的方法求出适合数据的一条直线：



根据线性回归模型我们只能预测连续的值，然而对于分类问题，我们需要输出 0 或 1，我们可以预测：

- 当  $h_{\theta}$  大于等于 0.5 时，预测  $y=1$
- 当  $h_{\theta}$  小于 0.5 时，预测  $y=0$

对于上图所示的数据，这样的一个线性模型似乎能很好地完成分类任务。假使我们又观测到一个非常大尺寸的恶性肿瘤，将其作为实例加入到我们的训练集中来，这将使得我们获得一条新的直线。



这时，再使用 0.5 作为阈值来预测肿瘤是良性还是恶性便不合适了。可以看出，线性回归模型，因为其预测的值可以超越[0,1]的范围，并不适合解决这样的问题。

我们引入一个新的模型，逻辑回归，该模型的输出变量范围始终在 0 和 1 之间。

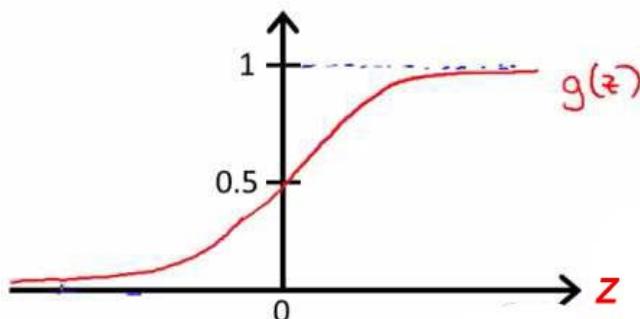
逻辑回归模型的假设是： $h_{\theta}(x) = g(\theta^T x)$

其中：

- $X$  代表特征向量
- $g$  代表逻辑函数 (logistic function) 是一个常用的逻辑函数为 S 形函数 (Sigmoid function)，公式为：

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

该函数的图像为：



合起来，我们得到逻辑回归模型的假设：

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

对模型的理解：

$h_{\theta}(x)$  的作用是 , 对于给定的输入变量 , 根据选择的参数计算输出变量 =1 的可能性( estimated probability ) 即  $h_{\theta}(x)=P(y=1|x;\theta)$

例如 , 如果对于给定的  $x$ , 通过已经确定的参数计算得出  $h_{\theta}(x)=0.7$  , 则表示有百分之 70 的几率  $y$  为正向类 , 相应地  $y$  为负向类的几率为  $1-0.7=0.3$ 。

### 5.3 判定边界 ( DECISION BOUNDARY )

在逻辑回归中 , 我们预测 :

- 当  $h_{\theta}$  大于等于 0.5 时 , 预测  $y=1$
- 当  $h_{\theta}$  小于 0.5 时 , 预测  $y=0$

根据上面绘制出的 S 形函数图像 , 我们知道当

- $z=0$  时  $g(z)=0.5$
- $z>0$  时  $g(z)>0.5$
- $z<0$  时  $g(z)<0.5$

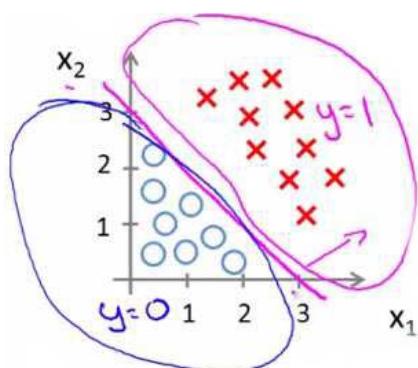
又  $z=\theta^T X$  , 即 :

- $\theta^T X$  大于等于 0 时 , 预测  $y=1$
- $\theta^T X$  小于 0 时 , 预测  $y=0$

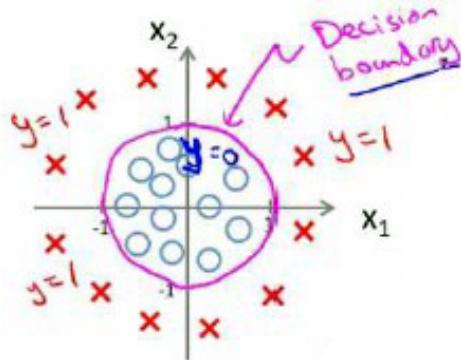
现在假设我们有一个模型 :  $h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$  并且参数  $\theta$  是向量  $[-3 \ 1 \ 1]$ 。

则当  $-3+x_1+x_2$  大于等于 0 , 即  $x_1+x_2$  大于等于 3 时 , 模型将预测  $y=1$ 。

我们可以绘制直线  $x_1+x_2=3$  , 这条线便是我们模型的分界线 , 将预测为 1 的区域和预测为 0 的区域分隔开。



假使我们的数据呈现这样的分布情况，怎样的模型才能适合呢？



因为需要用曲线才能分隔  $y=0$  的区域和  $y=1$  的区域，我们需要二次方特征：

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

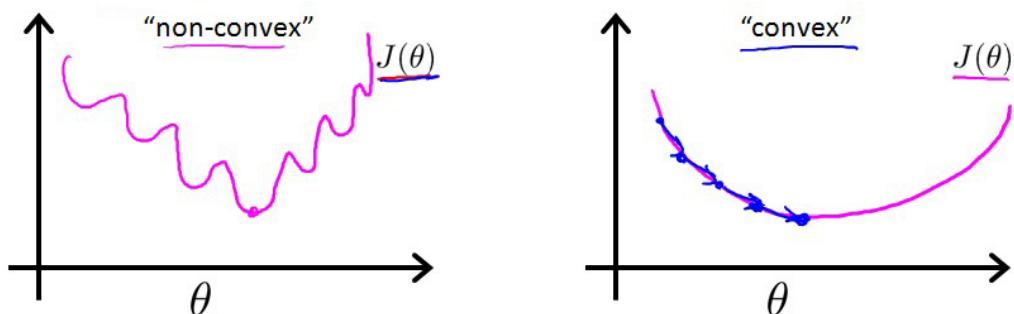
假设参数是  $[1 \ 0 \ 0 \ 1 \ 1]$ ，则我们得到的判定边界恰好是圆点在原点且半径为 1 的圆形。

我们可以用非常复杂的模型来适应非常复杂形状的判定边界。

#### 5.4 代价函数

对于线性回归模型，我们定义的代价函数是所有模型误差的平方和。理论上来说，我们也可以

对逻辑回归模型沿用这个定义，但是问题在于，当我们把  $h_{\theta}(x) = \frac{1}{1+e^{-\theta^T x}}$  带入到这样定义了的代价函数中时，我们得到的代价函数将是一个**非凸函数** ( non-convex function )。



这意味着我们的代价函数有许多局部最小值，这将影响梯度下降算法寻找全局最小值。

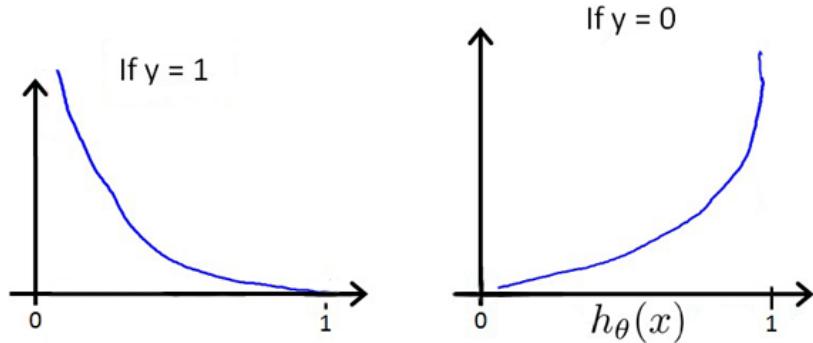
$$J(\theta) = \frac{1}{m} \sum_{i=1}^m Cost(h_{\theta}(x^{(i)}), y^{(i)})$$

因此我们重新定义逻辑回归的代价函数为：

其中

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

$h_\theta(x)$ 与  $\text{Cost}(h_\theta(x), y)$ 之间的关系如下图所示：



这样构建的  $\text{Cost}(h_\theta(x), y)$  函数的特点是：当实际的  $y=1$  且  $h_\theta$  也为 1 时误差为 0，当  $y=1$  但  $h_\theta$  不为 1 时误差随着  $h_\theta$  的变小而变大；当实际的  $y=0$  且  $h_\theta$  也为 0 时代价为 0，当  $y=0$  但  $h_\theta$  不为 0 时误差随着  $h_\theta$  的变大而变大。

将构建的  $\text{Cost}(h_\theta(x), y)$  简化如下：

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

带入代价函数得到：

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

在得到这样一个代价函数以后，我们便可以用梯度下降算法来求得能使代价函数最小的参数了。

算法为：

Repeat

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

} (simultaneously update all  $\theta_j$ )

求导后得到：

Repeat

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

(simultaneously update all )

}

注：虽然得到的梯度下降算法表面上看上去与线性回归的梯度下降算法一样，但是这里的  $h_\theta(x) = g(\theta^\top X)$  与线性回归中不同，所以实际上是不一样的。另外，在运行梯度下降算法之前，进行特征缩放依旧是非常必要的。

一些梯度下降算法之外的选择：

除了梯度下降算法以外还有一些常被用来令代价函数最小的算法，这些算法更加复杂和优越，而且通常不需要人工选择学习率，通常比梯度下降算法要更加快速。这些算法有：**共轭梯度** (Conjugate Gradient), **局部优化法**(Broyden fletcher goldfarb shann,BFGS)和**有限内存局部优化法**(LBFGS)

fminunc 是 matlab 和 octave 中都带的一个最小值优化函数，使用时我们需要提供代价函数和每个参数的求导，下面是 octave 中使用 fminunc 函数的代码示例：

```
function [jVal, gradient] = costFunction(theta)
    jVal = [... code to compute J(theta)...];
    gradient = [... code to compute derivative of J(theta)...];
end

options = optimset('GradObj', 'on', 'MaxIter', '100');

initialTheta = zeros(2, 1);

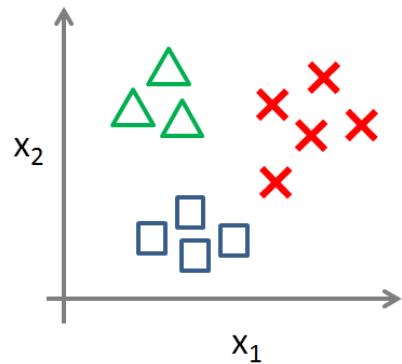
[optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
```

## 5.5 多类分类 ( MULTICLASS CLASSIFICATION )

多类分类问题中，我们的训练集中有多个类 ( $>2$ )，我们无法仅仅用一个二元变量 (0 或 1) 来做判断依据。例如我们要预测天气情况分四种类型：晴天、多云、下雨或下雪。

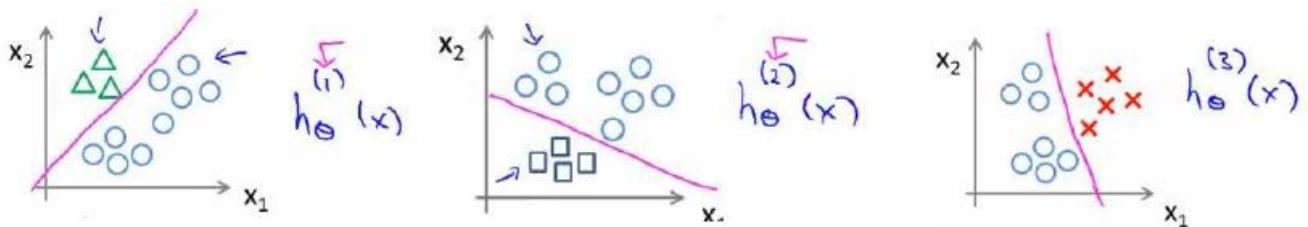
下面是一个多类分类问题可能的情况：

Multi-class classification:



一种解决这类问题的途径是采用**一对多** (One-vs-All) 方法。在一对多方法中，我们将多类分类问题转化成二元分类问题。为了能实现这样的转变，我们将多个类中的一个类标记为正向类 ( $y=1$ )，然后将其他所有类都标记为负向类，这个模型记作  $h_{\theta}^{(1)}(x)$ 。接着，类似地我们选择另一个类标记为正向类 ( $y=2$ )，再将其它类都标记为负向类，将这个模型记作  $h_{\theta}^{(2)}(x)$ ，依此类推。

最后我们得到一系列的模型简记为： $h_{\theta}^{(i)}(x)=p(y=i|x;\theta)$  其中  $i=(1,2,3,\dots,k)$



最后，在我们需要做预测时，我们将所有的分类机都运行一遍，然后对每一个输入变量，都选择最高可能性的输出变量。

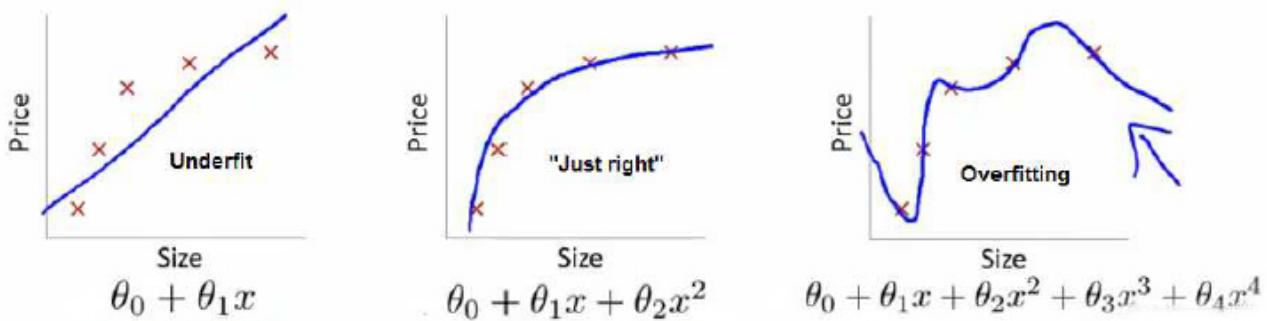
## WEEK3 归一化

### 6 归一化 ( REGULARIZATION )

#### 6.1 过拟合问题 ( THE PROBLEM OF OVERFITTING )

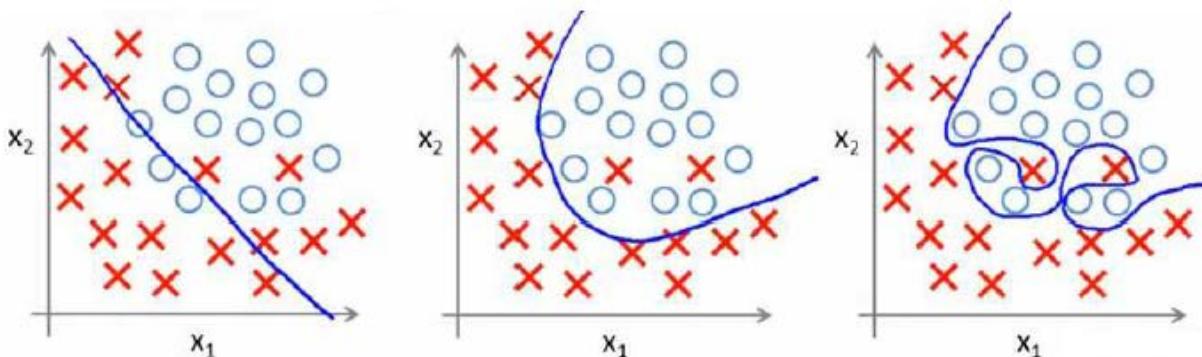
如果我们有非常多的特征，我们通过学习得到的假设可能能够非常好地适应训练集（代价函数可能几乎为 0），但是可能会不能推广到新的数据。

下图是一个回归问题的例子：



第一个模型是一个线性模型，低度拟合，不能很好地适应我们的训练集；第三个模型是一个四次方的模型，过度拟合，虽然能非常好地适应我们的训练集但在新输入变量进行预测时可能会效果不好；而中间的模型似乎最合适。

分类问题中也存在这样的问题：



问题是，如果我们发现了过拟合问题，应该如何处理？

1. 丢弃一些不能帮助我们正确预测的特征。

可以是手工选择保留哪些特征

或者使用一些模型选择的算法来帮忙（例如 PCA）

2. 归一化。

保留所有的特征，但是减少参数的**大小**（magnitude）。

## 6.2 归一化代价函数 ( REGULARIZATION COST FUNCTION )

上面的回归问题中如果我们的模型是：

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_3^3 + \theta_4 x_4^4$$

我们决定要减少 $\theta_3$ 和 $\theta_4$ 的大小 我们要做的便是修改代价函数 在其中 $\theta_3$ 和 $\theta_4$ 设置一点惩罚。这样做的话，我们在尝试最小化代价时也需要将这个惩罚纳入考虑中，并最终导致选择较小一些的 $\theta_3$ 和 $\theta_4$ 。修改后的代价函数如下：

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^m ((h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000\theta_3^2 + 10000\theta_4^2)$$

通过这样的代价函数选择出的 $\theta_3$ 和 $\theta_4$ 对预测结果的影响就比之前要小许多。

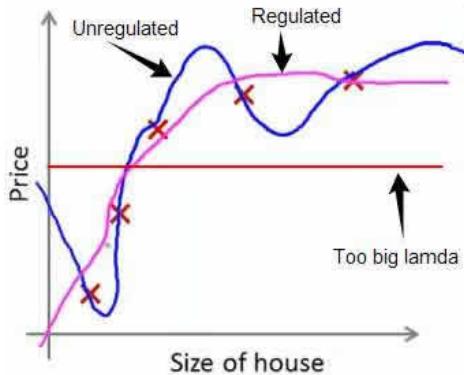
假如我们有非常多的特征，我们并不知道其中哪些特征我们要惩罚，我们将对所有的特征进行惩罚，并且让代价函数最优化的软件来选择这些惩罚的程度。这样的结果是得到了一个较为简单的能防止过拟合问题的假设：

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

其中 $\lambda$ 又称为**归一化参数** ( Regularization Parameter )。

注：根据惯例，我们不对 $\theta_0$ 进行惩罚。

经过归一化处理的模型与原模型的可能对比如下图所示：



如果选择的归一化参数 $\lambda$ 过大，则会把所有的参数都最小化了，导致模型变成 $h_\theta(x)=\theta_0$ 也就是上图中红色直线所示的情况，造成低度拟合。

### 6.3 归一化线性回归 ( REGULARIZED LINEAR REGRESSION )

归一化线性回归的代价函数为：

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

如果我们要使用梯度下降发令这个代价函数最小化，因为我们未对 $\theta_0$ 进行归一化，所以梯度下降算法将分两种情形：

*Repeat until convergence{*

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_0^{(i)}) \\ \theta_j &:= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m ((h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} + \frac{\lambda}{m} \theta_j) \\ &\quad \text{for } j=1,2,\dots,n \end{aligned}$$

}

对上面的算法中 $j=1,2,\dots,n$ 时的更新式子进行调整可得：

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

可以看出，归一化线性回归的梯度下降算法的变化在于，每次都在原有算法更新规则的基础上令 $\theta$ 值减少了一个额外的值。

我们同样也可以利用正规方程来求解归一化线性回归模型，方法如下所示：

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

图中的矩阵尺寸为  $n+1 \times n+1$ 。

#### 6.4 归一化逻辑回归 ( REGULARIZED LOGISTIC REGRESSION )

同样对于逻辑回归，我们也给代价函数增加一个归一化的表达式，得到：

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

要最小化该代价函数，通过求导，得出梯度下降算法为：

*Repeat until convergence {*

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

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

*for j=1,2,...n*

}

注：看上去同线性回归一样，但是知道  $h_\theta(x) = g(\theta^T x)$ ，所以与线性回归不同。

Octave 中，我们依旧可以用 fminuc 函数来求解代价函数最小化的参数，值得注意的是参数  $\theta_0$  的更新规则与其他情况不同。

## 7 神经网络：表达

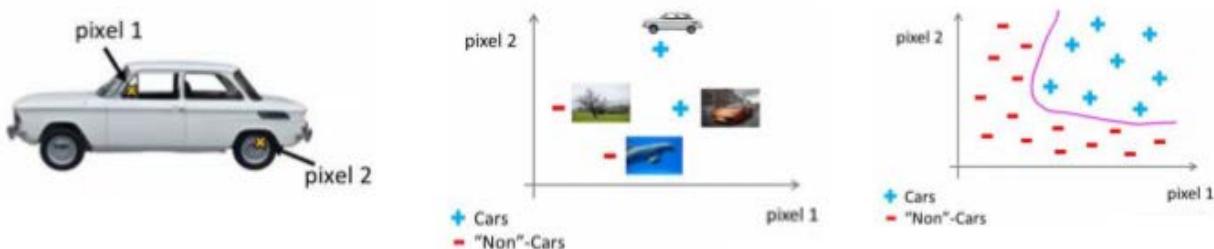
## 7.1 非线性假设 ( NON-LINEAR HYPOTHESIS )

之前我们已经看到过，使用非线性的多项式项能够帮助我们建立更好的分类模型。假设我们有非常多的特征，例如大于 100 个变量，我们希望用这 100 个特征来构建一个非线性的多项式模型，结果将是数量非常惊人的特征组合，即便我们只采用两两特征的组合

$(x_1x_2 + x_1x_3 + x_1x_4 + \dots + x_2x_3 + x_2x_4 + \dots + x_{99}x_{100})$ ，我们也会有接近 5000 个组合而成的特征。这对于一般的逻辑回归来说需要计算的特征太多了。

假设我们希望训练一个模型来识别视觉对象（例如识别一张图片上是否是一辆汽车），我们怎样才能这么做呢？一种方法是我们利用很多汽车的图片和很多非汽车的图片，然后利用这写图片上一个个像素的值（饱和度或亮度）来作为特征。

假如我们只选用灰度图片，每个像素则只有一个值（而非 RGB 值），我们可以选取图片上的两个不同位置上的两个像素，然后训练一个逻辑回归算法利用这两个像素的值来判断图片上是否是汽车：



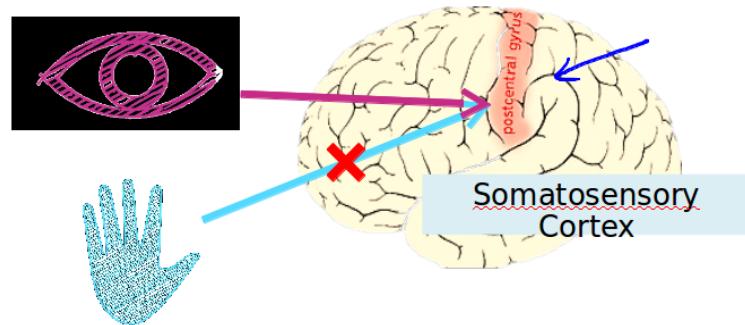
假使我们采用的都是  $50 \times 50$  像素的小图片，并且我们将所有的像素视为特征，则会有 2500 个特征，如果我们要进一步将两两特征组合构成一个多项式模型，则会有约  $2500^2/2$  个（接近 3 百万个）特征。普通的逻辑回归模型，不能有效地处理这么多的特征，这时候我们需要神经网络。

## 7.2 神经网络介绍

神经网络算法的来源：

神经网络算法源自对大脑的模仿。神经网络算法在八十年代被广为使用过，再之后便逐渐减少了使用，但是最近又开始变得流行起来，原因是神经网络是非常依赖计算能力的算法，随着新计算机性能的提高，算法又成为了有效的技术。

神经网络算法的目的是发现一个能模仿人类大脑学习能力的算法。研究表明，如果我们将视觉信号传给大脑中负责其他感觉的大脑皮层处，则那些大脑组织将能学会如何处理视觉信号。

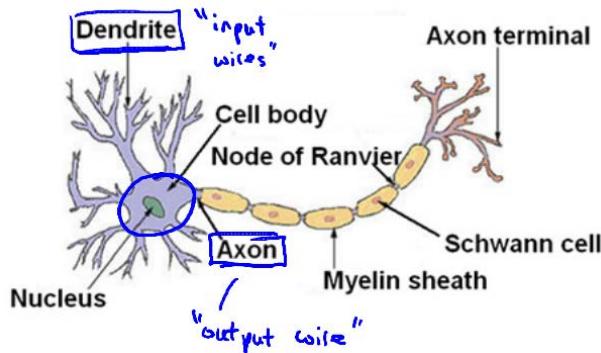


下面是一个让舌头学会如何去看的例子。在一个盲人的头顶配置一台低像素的照相机，然后将照片的像素转换为不同的电极，每个像素都按照亮度给赋予一个不同的电压值。结果随着实验进行，这个盲人开始能够利用舌头看见眼前的东西。

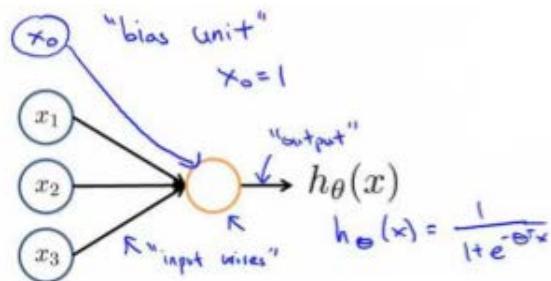


### 7.3 模型表达

为了构建神经网络模型，我们需要首先思考大脑中的神经网络是怎样的？每一个神经元都可以被认为是一个**处理单元/神经核** ( processing unit/ Nucleus )，它含有许多**输入/树突** ( input/Dendrite )，并且有一个**输出/轴突** ( output/Axon )。神经网络是大量神经元相互链接并通过电脉冲来交流的一个网络。



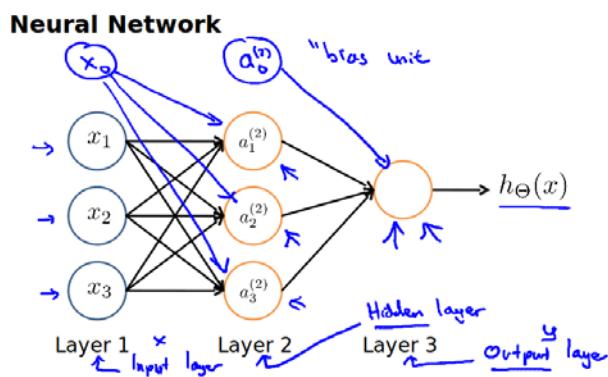
神经网络模型建立在很多神经元之上，每一个神经元又是一个个学习模型。这些神经元（也叫**激活单元**，activation unit）采纳一些特征作为输出，并且根据本身的模型提供一个输出。下图是一个以逻辑回归模型作为自身学习模型的神经元示例，在神经网络中，参数又可被成为**权重**（weight）。



Sigmoid (logistic) activation function.

## 7.4 神经网络模型表达

神经网络模型是许多逻辑单元按照不同层级组织起来的网络，每一层的输出变量都是下一层的输入变量。下图为一个3层的神经网络，第一层成为**输入层**（Input Layer），最后一层称为**输出层**（Output Layer），中间一层成为**隐藏层**（Hidden Layers）。我们为每一层都增加一个**偏倚单位**（bias unit）：



下面引入一些标记法来帮助描述模型：

- $a_i^{(j)}$  代表第  $j$  层的第  $i$  个激活单元。
- $\Theta^{(j)}$  代表从第  $j$  层映射到第  $j+1$  层时的权重的矩阵，例如  $\Theta^{(1)}$  代表从第一层映射到第二层的权重的矩阵。其尺寸为：以第  $j$  层的激活单元数量为行数，以第  $j+1$  层的激活单元数为列数的矩阵。例如：上图所示的神经网络中  $\Theta^{(1)}$  的尺寸为  $4 \times 3$ 。

对于上图所示的模型，激活单元和输出分别表达为：

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

上面进行的讨论中只是将特征矩阵中的一行（一个训练实例）喂给了神经网络，我们需要将整个训练集都喂给我们的神经网络算法来学习模型。

## 7.5 正向传播 (FORWARD PROPAGATION)

相对与使用循环来编码，利用向量化的方法会使得计算更为简便。以上面的神经网络为例，试着计算第二层的值：

$$g\left(\begin{bmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = g\left(\begin{bmatrix} \theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3 \\ \theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3 \\ \theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3 \end{bmatrix}\right) = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

我们令  $z^{(2)} = \Theta^{(1)}x$ ，则  $a^{(2)} = g(z^{(2)})$ ，计算后添加  $a_0^{(2)} = I$ 。

计算输出的值为：

$$g\left(\begin{bmatrix} \theta_{10}^{(2)} & \theta_{11}^{(2)} & \theta_{12}^{(2)} & \theta_{13}^{(2)} \end{bmatrix} \times \begin{bmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}\right) = g(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}) = h_\Theta(x)$$

我们令  $z^{(3)} = \Theta^{(2)}a^{(2)}$ ，则  $h_\Theta(x) = a^{(3)} = g(z^{(3)})$ 。

这只是针对训练集中一个训练实例所进行的计算。如果我们要对整个训练集进行计算，我们需要将训练集特征矩阵进行转置，使得同一个实例的特征都在同一列里。即：

$$z^{(2)} = \Theta^{(1)} \times X^T$$

$$a^{(2)} = g(z^{(2)})$$

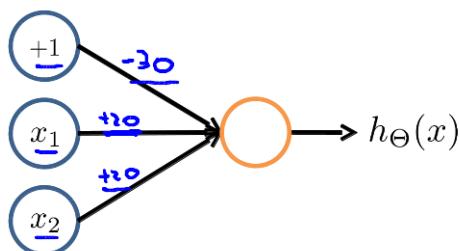
## 7.6 对神经网络的理解

本质上讲，神经网络能够通过学习得出其自身的一系列特征。在普通的逻辑回归中，我们被限制为使用数据中的原始特征  $x_1, x_2, \dots, x_n$ ，我们虽然可以使用一些二项式项来组合这些特征，但是我们仍然受到这些原始特征的限制。在神经网络中，原始特征只是输入层，在我们上面三层的神经网络例子中，第三层也就是输出层做出的预测利用的是第二层的特征，而非输入层中的原始特征，我们可以认为第二层中的特征是神经网络通过学习后自己得出的一系列用于预测输出变量的新特征。

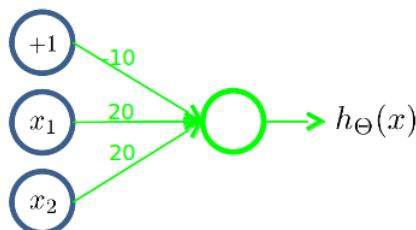
## 7.7 神经网络示例：二元逻辑运算符（BINARY LOGICAL OPERATORS）

当输入特征为布尔值（0 或 1）时，我们可以用一个单一的激活层可以作为二元逻辑运算符，为了表示不同的运算符，我们之需要选择不同的权重即可。

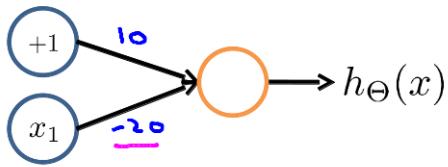
下图的神经元（三个权重分别为 -30, 20, 20）可以被视为作用同于逻辑与（AND）：



下图的神经元（三个权重分别为 -10, 20, 20）可以被视为作用等同于逻辑或（OR）：

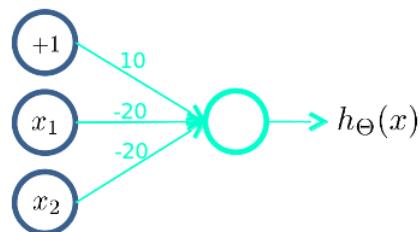


下图的神经元（两个权重分别为 10, -20）可以被视为作用等同于逻辑非（NOT）：

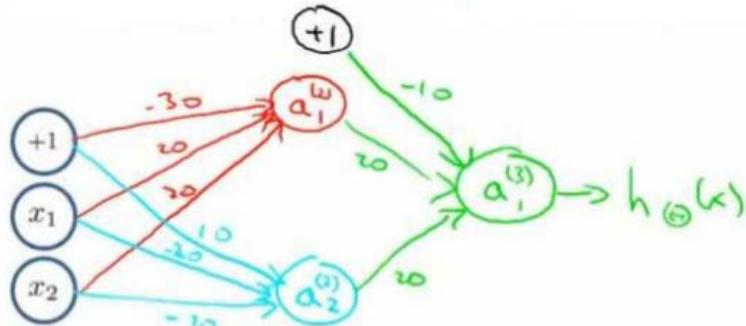


我们可以利用神经元来组合成更为复杂的神经网络以实现更复杂的运算。例如我们要实现 XNOR 功能（输入的两个值必须一样，均为 1 或均为 0），即  $XNOR = (x_1 \text{AND} x_2) \text{OR} ((\text{NOT}x_1) \text{AND} (\text{NOT}x_2))$

首先构造一个能表达 $(\text{NOT}x_1) \text{AND} (\text{NOT}x_2)$ 部分的神经元：



然后将表示 AND 的神经元和表示 $(\text{NOT}x_1) \text{AND} (\text{NOT}x_2)$ 的神经元以及表示 OR 的神经元进行组合：

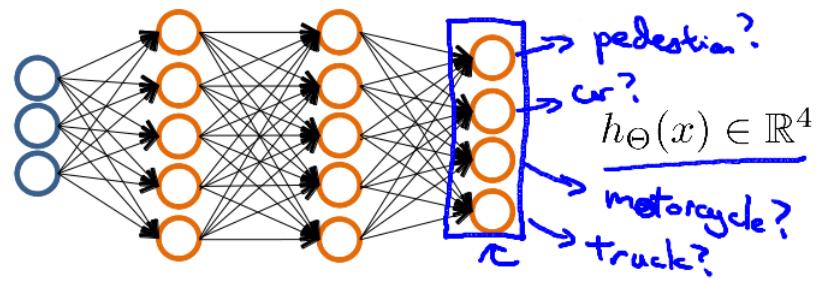


我们就得到了一个能实现 XNOR 运算符功能的神经网络。

## 7.8 多类分类

如果我们要训练一个神经网络算法来识别路人、汽车、摩托车和卡车，在输出层我们应该有 4 个值。例如，第一个值为 1 或 0 用于预测是否是行人，第二个值用于判断是否为汽车。

下面是该神经网络的可能结构示例：



神经网络算法的输出结果为四种可能情形之一：

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

## 8 神经网络：学习

## 8.1 神经网络代价函数

首先引入一些便于稍后讨论的新标记方法：

- $L$  代表一个神经网络中的层数
- $S_l$  代表第  $l$  层的处理单元（包括偏见单元）的个数。
- $S_L$  代表最后一层中处理单元的个数。
- $K$  代表我们希望分类的类的个数，与  $S_L$  相等。

我们回顾逻辑回归问题中我们的代价函数为：

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

在逻辑回归中，我们只有一个输出变量，又称标量（scalar），也只有一个因变量  $y$ ，但是在神经网络中，我们可以有很多输出变量，我们的  $h_\theta(x)$  是一个维度为  $K$  的向量，并且我们训练集中的因变量也是同样维度的一个向量，因此我们的代价函数会比逻辑回归更加复杂一些，为：

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

这个看起来复杂很多的代价函数背后的思想还是一样的，我们希望通过代价函数来观察算法预测的结果与真实情况的误差有多大，唯一不同的是，对于每一行特征，我们都会给出  $K$  个预测，基本上我们可以利用循环，对每一行特征都预测  $K$  个不同结果，然后在利用循环在  $K$  个预测中选择可能性最高的一个，将其与  $y$  中的实际数据进行比较。

归一化的那一项只是排除了每一层  $\theta_0$  后，每一层的  $\Theta$  矩阵的和。最里层的循环  $j$  循环所有的行（由  $s_{l+1}$  层的激活单元数决定），循环  $i$  则循环所有的列，由该层（ $s_l$  层）的激活单元数所决定。

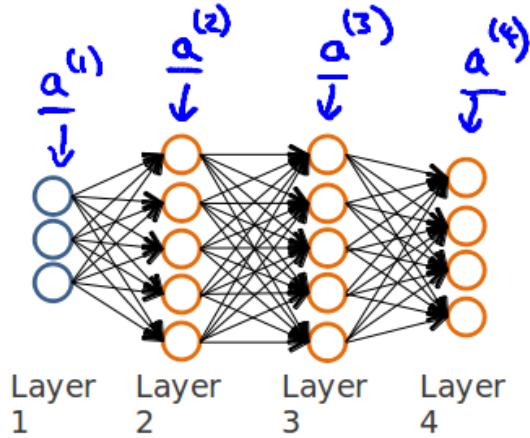
## 8.2 反向传播算法（BACKPROPAGATION ALGORITHM）

之前我们在计算神经网络预测结果的时候我们采用了一种正向传播方法，我们从第一层开始正向一层一层进行计算，直到最后一层的  $h_\theta(x)$ 。

现在，为了计算代价函数的偏导数  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ ，我们需要采用一种反向传播算法，也就是首先计算最后一层的误差，然后再一层一层反向求出各层的误差，直到倒数第二层。

以一个例子来说明反向传播算法。

假设我们的训练集只有一个实例  $(x^{(1)}, y^{(1)})$ ，我们的神经网络是一个四层的神经网络，其中  $K=4$ ,  $S_L=4$ ,  $L=4$ ：



我们从最后一层的误差开始计算，误差是激活单元的预测 ( $a_k^{(4)}$ ) 与实际值 ( $y_k$ ) 之间的误差，( $k=1:K$ )。我们用 $\delta$ 来表示误差，则：

$$\delta^{(4)} = a^{(4)} - y$$

我们利用这个误差值来计算前一层的误差：

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} * g'(z^{(3)})$$

其中  $g'(z^{(3)})$  是 S 形函数的导数， $g'(z^{(3)}) = a^{(3)} * (1 - a^{(3)})$ 。而  $(\Theta^{(3)})^T \delta^{(4)}$  则是权重导致的误差的和。

下一步是继续计算第二层的误差：

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} * g'(z^{(2)})$$

因为第一层是输入变量，不存在误差。我们有了所有的误差的表达式后，便可以计算代价函数的偏导数了，假设  $\lambda=0$ ，即我们不做任何归一化处理时有：

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{l+1}$$

重要的是清楚地知道上面式子中上下标的含义：

- $l$  代表目前所计算的是第几层
- $j$  代表目前计算层中的激活单元的下标，也将是下一层的第  $j$  个输入变量的下标。
- $i$  代表下一层中误差单元的下标，是受到权重矩阵中第  $i$  行影响的下一层中的误差单元的下标。

如果我们考虑归一化处理，并且我们的训练集是一个特征矩阵而非向量。在上面的特殊情况下，我们需要计算每一层的误差单元来计算代价函数的偏导数。在更为一般的情况下，我们同样需要计算每一层的误差单元，但是我们需要为整个训练集计算误差单元，此时的误差单元也是一个矩阵，我们用  $\Delta_{ij}^{(l)}$  来表示这个误差矩阵。第  $l$  层的第  $i$  个激活单元受到第  $j$  个参数影响而导致的误差。

我们的算法表示为：

```
for i=1:m {
    set a(i)=x(i)
    perform foward propagation to compute a(l) for l=1,2,3...L
    Using δ(L)=a(L)-yi
    perform back propagation to compute all previous layer error vector
    Δ(l)ij:=Δ(l)ij+a(l)jδ(l+1)i
}
```

即首先用正向传播方法计算出每一层的激活单元，利用训练集的结果与神经网络预测的结果求出最后一层的误差，然后利用该误差运用反向传播法计算出直至第二层的所有误差。

在求出了  $\Delta_{ij}^{(l)}$  之后，我们便可以计算代价函数的偏导数了，计算方法如下：

$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$
$$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} \text{ if } j = 0$$

在 Octave 中，如果我们要使用 fminuc 这样的优化算法来求解求出权重矩阵，我们需要将矩阵首先展开成为向量，在利用算法求出最优解后再重新转换回矩阵。

假设我们有三个权重矩阵，Theta1，Theta2 和 Theta3，尺寸分别为 10\*11，10\*11 和 1\*11

下面的代码可以实现这样的转换：

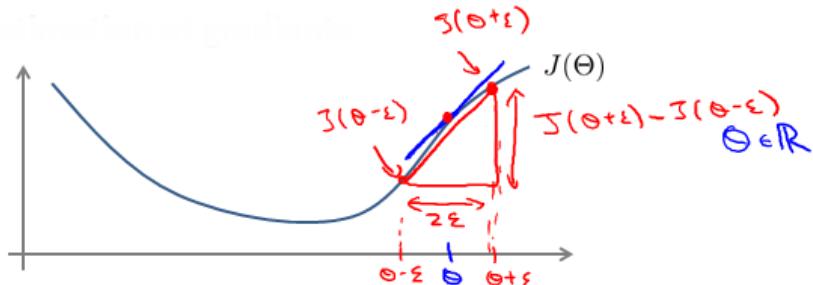
```
thetaVec = [Theta1(:) ; Theta2(:) ; Theta3(:)]  
... optimization using functions like fminuc...  
Theta1 = reshape(thetaVec(1:110, 10, 11);  
Theta2 = reshape(thetaVec(111:220, 10, 11);  
Theta1 = reshape(thetaVec(221:231, 1, 11);
```

### 8.3 梯度检验 ( GRADIENT CHECKING )

当我们对一个较为复杂的模型 ( 例如神经网络 ) 使用梯度下降算法时 , 可能会存在一些不容易察觉的错误 , 意味着 , 虽然代价看上去在不断减小 , 但最终的结果可能并不是最优解。

为了避免这样的问题 , 我们采取一种叫做**梯度的数值检验** ( Numerical Gradient Checking ) 方法。这种方法的思想是通过估计梯度值来检验我们计算的导数值是否真的是我们要求的。

对梯度的估计采用的方法是在代价函数上沿着切线的方向选择两个非常近的点然后计算两个点的平均值用以估计梯度。即对于某个特定的  $\theta$  , 我们计算出在  $\theta - \epsilon$  处和  $\theta + \epsilon$  的代价值 ( $\epsilon$  是一个非常小的值 , 通常选取 0.001 ) , 然后求两个代价的平均 , 用以估计在  $\theta$  处的代价值。



Octave 中代码如下 :

```
gradApprox = (J(theta + eps) - J(theta - eps)) / (2*eps)
```

当  $\theta$  是一个向量时 , 我们则需要对偏导数进行检验。因为代价函数的偏导数检验只针对一个参数的改变进行检验 , 下面是一个只针对  $\theta_1$  进行检验的示例 :

$$\frac{\partial}{\partial \theta_1} = \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

最后我们还需要对通过反向传播方法计算出的偏导数进行检验。

根据上面的算法，计算出的偏导数存储在矩阵  $D_{ij}^{(l)}$  中。检验时，我们要将该矩阵展开成为向量，同时我们也将  $\Theta$  矩阵展开为向量，我们针对每一个  $\theta$  都计算一个近似的梯度值，将这些值存储于一个近似梯度矩阵中，最终将得出的这个矩阵同  $D_{ij}^{(l)}$  进行比较。

```
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;

Check that gradApprox ≈ DVec
```

## 8.4 随机初始化 ( RANDOM INITIALIZATION )

任何优化算法都需要一些初始的参数。到目前为止我们都是初始所有参数为 0，这样的初始方法对于逻辑回归来说是可行的，但是对于神经网络来说是不可行的。如果我们令所有的初始参数都为 0，这将意味着我们第二层的所有激活单元都会有相同的值。同理，如果我们初始所有的参数都为一个非 0 的数，结果也是一样的。

我们通常初始参数为正负  $\epsilon$  之间的随机值，假设我们要随机初始一个尺寸为  $10 \times 11$  的参数矩阵，代码如下：

```
Theta1 = rand(10, 11) * (2*eps) - eps
```

## 8.5 综合起来

小结一下使用神经网络时的步骤：

网络结构：

第一件要做的事是选择网络结构，即决定选择多少层以及决定每层分别有多少个单元。

- 第一层的单元数即我们训练集的特征数量。
- 最后一层的单元数是我们训练集的结果的类的数量。
- 如果隐藏层数大于 1，确保每个隐藏层的单元个数相同，通常情况下隐藏层单元的个数越多越好。

我们真正要决定的是隐藏层的层数和每个中间层的单元数。

训练神经网络：

1. 参数的随机初始化
2. 利用正向传播方法计算所有的  $h_{\theta}(x)$
3. 编写计算代价函数  $J$  的代码
4. 利用反向传播方法计算所有偏导数
5. 利用数值检验方法检验这些偏导数
6. 使用优化算法来最小化代价函数

### 9 机器学习应用建议

#### 9.1 决定下一步做什么

假设我们需要用一个线性回归模型来预测房价，当我们运用训练好了的模型来预测未知数据的时候发现有较大的误差，我们下一步可以做什么？

1. 获得更多的训练实例——通常是有效的，但代价较大，下面的方法也可能有效，可考虑先采用下面的几种方法。
2. 尝试减少特征的数量
3. 尝试获得更多的特征
4. 尝试增加二项式特征
5. 尝试减少归一化程度 $\lambda$
6. 尝试增加归一化程度 $\lambda$

我们不应该随机选择上面的某种方法来改进我们的算法，而是运用一些机器学习诊断法来帮助我们知道上面哪些方法对我们的算法是有效的。

#### 9.2 假设的评估 ( EVALUATING A HYPOTHESIS )

##### 过拟合检验

为了检验算法是否过拟合，我们将数据分成训练集和测试集，通常用 70% 的数据作为训练集，用剩下 30% 的数据作为测试集。很重要的一点是训练集和测试集均要含有各种类型的数据，通常我们要对数据进行“洗牌”，然后再分成训练集和测试集。

##### 测试集评估

在通过训练集让我们的模型学习得出其参数后，对测试集运用该模型，我们有两种方式计算误差：

1. 对于线性回归模型，我们利用测试集数据计算代价函数  $J$

2. 对于逻辑回归模型，我们除了可以利用测试数据集来计算代价函数外：

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

还可以计算错误分类的比率，对于每一个测试集实例，计算：

$$err(h_{\theta}(x), y) = \begin{cases} 1 & \text{if } h(x) \geq 0.5 \text{ and } y = 0, \text{ or if } h(x) < 0.5 \text{ and } y = 1 \\ 0 & \text{Otherwise} \end{cases}$$

然后对计算结果求平均。

### 9.3 模型选择（交叉验证集）

假设我们要在 10 个不同次数的二项式模型之间进行选择：

1.  $h_{\theta}(x) = \theta_0 + \theta_1 x$
2.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$
3.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3$
- ⋮
10.  $h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10}$

显然越高次数的二项式模型越能够适应我们的训练数据集，但是适应训练数据集并不代表着能推广至一般情况，我们应该选择一个更能适应一般情况的模型。我们需要使用交叉验证集来帮助选择模型。

即：

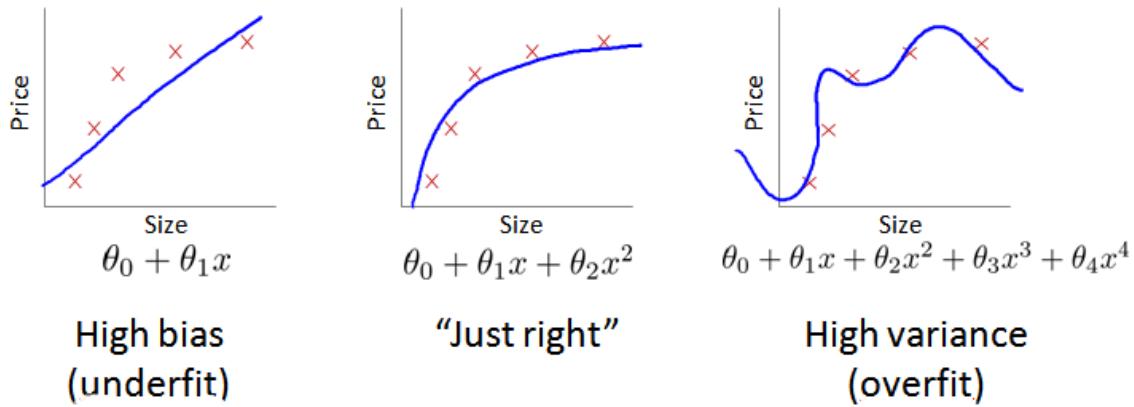
- 使用 60% 的数据作为训练集
- 使用 20% 的数据作为交叉验证集
- 使用 20% 的数据作为测试集

模型选择的方法为：

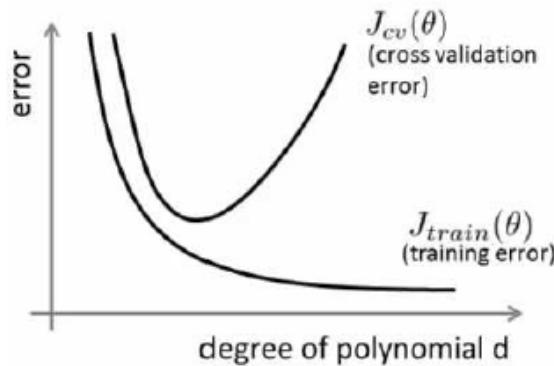
1. 使用训练集训练出 10 个模型
2. 用 10 个模型分别对交叉验证集计算得出交叉验证误差（代价函数的值）
3. 选取代价函数值最小的模型
4. 用步骤 3 中选出的模型对测试集计算得出推广误差（代价函数的值）

### 9.4 偏倚和偏差诊断（DIAGNOSIS BIAS VS. VARIANCE）

高偏倚和高偏差的问题基本上来说是低拟合和过拟合的问题。



我们通常会通过将训练集和交叉验证集的代价函数误差与多项式的次数绘制在同一张图表上来帮助分析：



- 对于训练集，当  $d$  较小时，模型拟合程度更低，误差较大；随着  $d$  的增长，拟合程度提高，误差减小。
- 对于交叉验证集，当  $d$  较小时，模型拟合程度低，误差较大；但是随着  $d$  的增长，误差呈现先减小后增大的趋势，转折点是我们的模型开始过拟合训练数据集的时候。

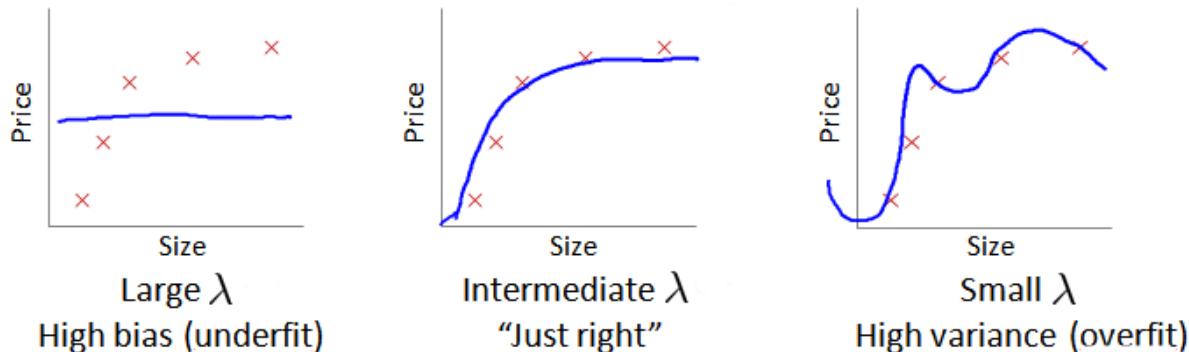
如果我们的交叉验证集误差较大，我们如何判断是偏倚还是偏差呢？

根据上面的图表，我们知道：

- 训练集误差和交叉验证集误差近似时：偏倚/低拟合
- 交叉验证集误差远大于训练集误差时：偏差/过拟合

## 9.5 归一化与偏倚/偏差

在我们在训练模型的过程中，一般会使用一些归一化方法来防止过拟合。但是我们可能会归一化的程度太高或太小了，即我们在选择 $\lambda$ 的值时也需要思考与刚才选择多项式模型次数类似的问题。



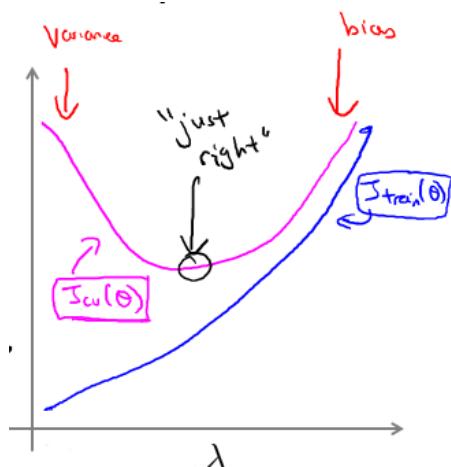
我们选择一系列的想要测试的 $\lambda$ 值，通常是 0-10 之间的呈现 2 倍关系的值（如：0,0.01,0.02,0.04,0.08,0.15,0.32,0.64,1.28,2.56,5.12,10 共 12 个）。

我们同样把数据分为训练集、交叉验证集和测试集。

选择 $\lambda$ 的方法为：

1. 使用训练集训练出 12 个不同程度归一化的模型
2. 用 12 个模型分别对交叉验证集计算的出交叉验证误差
3. 选择得出交叉验证误差最小的模型
4. 运用步骤 3 中选出模型对测试集计算得出推广误差

我们也可以同时将训练集和交叉验证集模型的代价函数误差与 $\lambda$ 的值绘制在一张图表上：

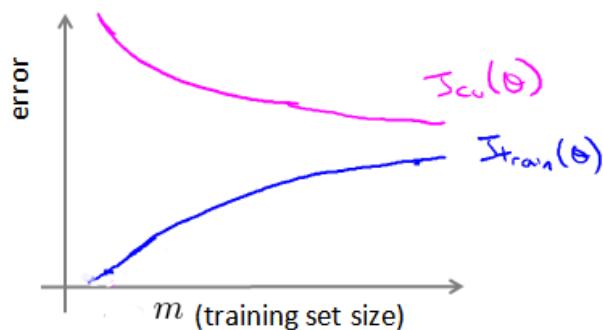


- 当 $\lambda$ 较小时，训练集误差较小（过拟合）而交叉验证集误差较大
- 随着 $\lambda$ 的增加，训练集误差不断增加（低拟合），而交叉验证集误差则是先减小后增加

## 9.6 学习曲线 ( LEARNING CURVES )

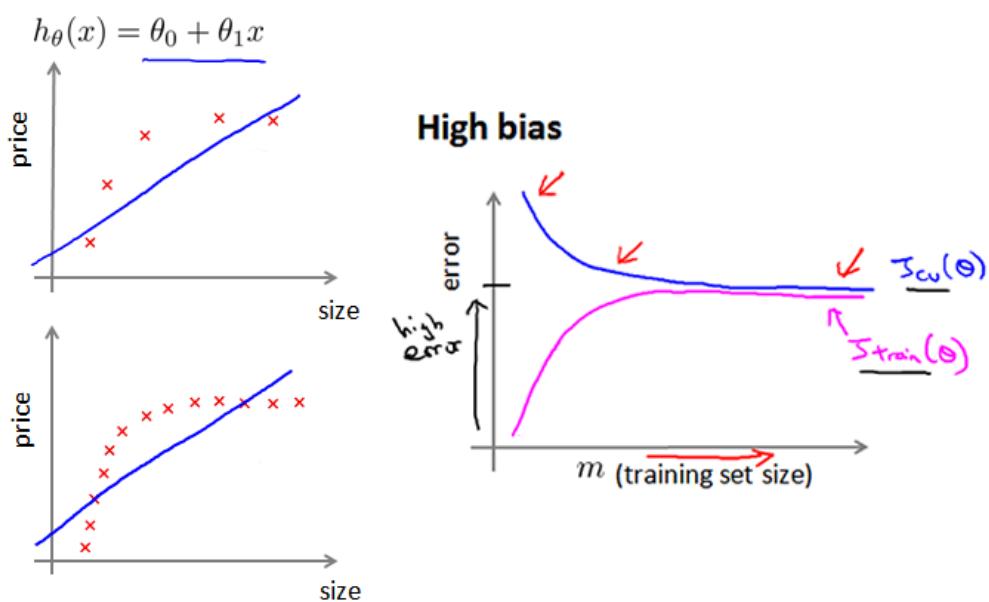
学习曲线是学习算法的一个很好的**合理检验** ( sanity check )。学习曲线是将训练集误差和交叉验证集误差作为训练集实例数量 ( $m$ ) 的函数绘制的图表。

即，如果我们有 100 行数据，我们从 1 行数据开始，逐渐学习更多行的数据。思想是：当训练较少行数据的时候，训练的模型将能够非常完美地适应较少的训练数据，但是训练出来的模型却不能很好地适应交叉验证集数据或测试集数据。



如何利用学习曲线识别高偏倚/低拟合：

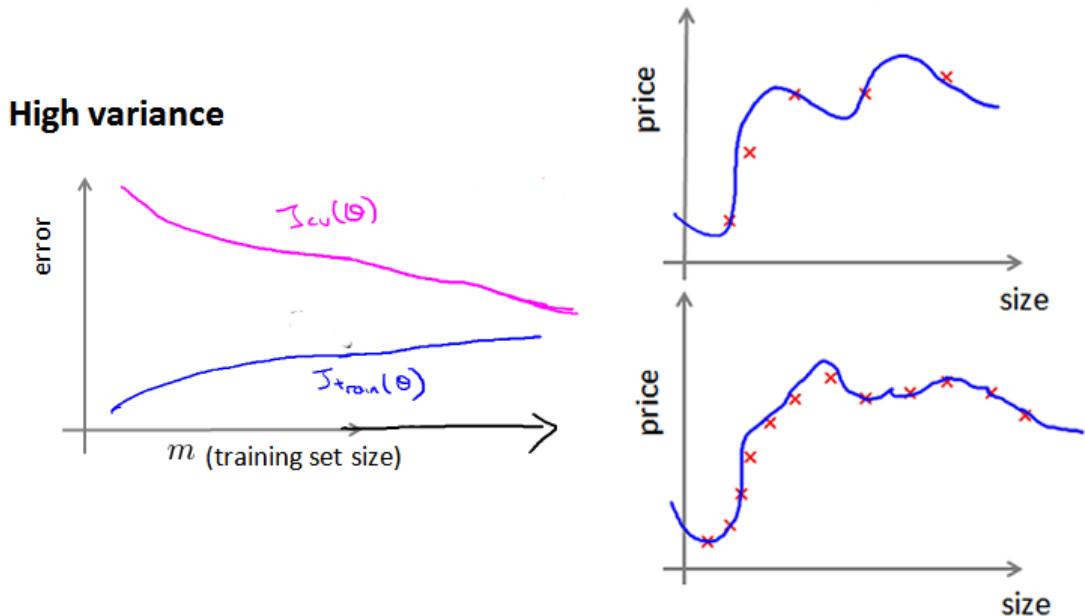
作为例子，我们尝试用一条直线来适应下面的数据，可以看出，无论训练集有多么大误差都不会有太大改观：



也就是说在高偏倚/低拟合的情况下，增加数据到训练集不一定能有帮助。

如何利用学习曲线识别高偏差/过拟合：

假设我们使用一个非常高次的多项式模型，并且归一化非常小，可以看出，当交叉验证集误差远大于训练集误差时，往训练集增加更多数据可以提高模型的效果。



也就是说在高偏差/过拟合的情况下，增加更多数据到训练集可能可以提高算法效果。

## 9.7 决定下一步做什么

回顾 1.1 中提出的六种可选的下一步，让我们来看一看我们在什么情况下应该怎样选择：

1. 获得更多的训练实例——解决高偏差
2. 尝试减少特征的数量——解决高偏差
3. 尝试获得更多的特征——解决高偏倚
4. 尝试增加二项式特征——解决高偏倚
5. 尝试减少归一化程度  $\lambda$ ——解决高偏倚
6. 尝试增加归一化程度  $\lambda$ ——解决高偏差

神经网络的偏倚和偏差

- 使用较小的神经网络，类似于参数较少的情况，容易导致高偏倚和低拟合，但计算代价较小
- 使用较大的神经网络，类似于参数较多的情况，容易导致高偏差和过拟合，虽然计算代价比较大，但是可以通过归一化手段来调整而更加适应数据。

通常选择较大的神经网络并采用归一化处理会比采用较小的神经网络效果要好。

对于神经网络中的隐藏层的层数的选择，通常从一层开始逐渐增加层数，为了更好地作选择，可以把数据分为训练集、交叉验证集和测试集，针对不同隐藏层数的神经网络训练神经网络，然后选择交叉验证集代价最小的神经网络。

### 10 机器学习系统设计

#### 10.1 首先要做什么

本周以一个垃圾邮件分类器算法为例进行讨论。

为了解决这样一个问题，我们首先要做的决定是如何选择并表达特征向量  $x$ 。我们可以选择一个由 100 个最常出现在垃圾邮件中的词所构成的列表，根据这些词是否有在邮件中出现，来获得我们的特征向量（出现为 1，不出现为 0），尺寸为  $100 \times 1$ 。

为了构建这个分类器算法，我们可以做很多事，例如：

1. 收集更多的数据，让我们有更多的垃圾邮件和非垃圾邮件的样本
2. 基于邮件的路由信息开发一系列复杂的特征
3. 基于邮件的正文信息开发一系列复杂的特征，包括考虑截词的处理
4. 为探测刻意的拼写错误（把 watch 写成 w4tch）开发复杂的算法

在上面这些选项中，非常难决定应该在哪一项上花费时间和精力，作出明智的选择比随着感觉走要更好。

#### 10.2 误差分析 ( ERROR ANALYSIS )

误差分析可以帮助我们系统化地选择该做什么。

构建一个学习算法的推荐方法为：

1. 从一个简单的能快速实现的算法开始，实现该算法并用交叉验证集数据测试这个算法
2. 绘制学习曲线，决定是增加更多数据，或者添加更多特征，还是其他选择
3. 进行误差分析：人工检查交叉验证集中我们算法中产生预测误差的实例，看看这些实例是否有某种系统化的趋势

以我们的垃圾邮件过滤器为例，误差分析要做的既是检验交叉验证集中我们的算法产生错误预测的所有邮件，看：

- 是否能将这些邮件按照类分组。例如医药品垃圾邮件，仿冒品垃圾邮件或者密码窃取邮件等。然后看分类器对哪一组邮件的预测误差最大，并着手优化。

- 思考怎样能改进分类器。例如，发现是否缺少某些特征，记下这些特征出现的次数。例如记录下错误拼写出现了多少次，异常的邮件路由情况出现了多少次等等，然后从出现次数最多的情况开始着手优化。

误差分析并不总能帮助我们判断应该采取怎样的行动。有时我们需要尝试不同的模型，然后进行比较，在模型比较时，用数值来判断哪一个模型更好更有效，通常我们是看交叉验证集的误差。

在我们的垃圾邮件分类器例子中，对于“我们是否应该将 discount/discounts/discounted/discounting 处理成同一个词？”如果这样做可以改善我们算法，我们会采用一些截词软件。误差分析不能帮助我们做出这类判断，我们只能尝试采用和不采用截词软件这两种不同方案，然后根据数值检验的结果来判断哪一种更好。

### 10.3 类偏斜的误差度量 ( ERROR METRICS FOR SKEWED CLASSES )

类偏斜情况表现为我们的训练集中有非常多的同一种类的实例，只有很少或没有其他类的实例。

例如我们希望用算法来预测癌症是否是恶性的，在我们的训练集中，只有 0.5% 的实例是恶性肿瘤。假设我们编写一个非学习而来的算法，在所有情况下都预测肿瘤是良性的，那么误差只有 0.5%。然而我们通过训练而得到的神经网络算法却有 1% 的误差。这时，误差的大小是不能视为评判算法效果的依据的。

#### 查准率 ( Precision ) 和查全率 ( Recall )

我们将算法预测的结果分成四种情况：

1. **正确肯定** ( True Positive, TP ) : 预测为真，实际为真
2. **正确否定** ( True Negative, TN ) : 预测为假，实际为真
3. **错误肯定** ( False Positive, FP ) : 预测为真，实际为假
4. **错误否定** ( False Negative, FN ) : 预测为假，实际为假

则：

- 查准率 =  $TP / (TP + FP)$

例，在所有我们预测有恶性肿瘤的病人中，实际上有恶性肿瘤的病人的百分比，越高越好。

- 查全率 =  $TP / (TP + FN)$

例，在所有实际上有恶性肿瘤的病人中，成功预测有恶性肿瘤的病人的百分比，越高越好。

这样，对于我们刚才那个总是预测病人肿瘤为良性的算法，其查全率是 0。

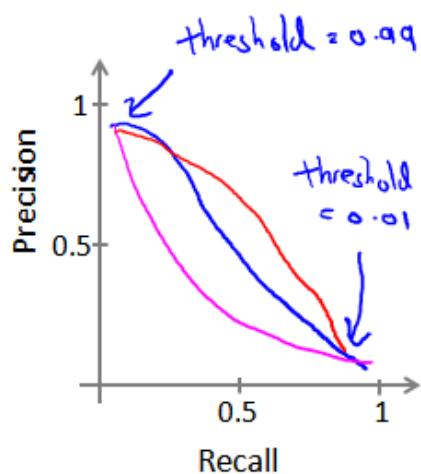
#### 10.4 查全率和查准率之间的权衡

继续沿用刚才预测肿瘤性质的例子。假使，我们的算法输出的结果在 0-1 之间，我们使用阀值 0.5 来预测真和假。

如果我们希望只在非常确信的情况下预测为真（肿瘤为恶性），即我们希望更高的查准率，我们可以使用比 0.5 更大的阀值，如 0.7, 0.9。这样做我们会减少错误预测病人为恶性肿瘤的情况，同时却会增加未能成功预测肿瘤为恶性的情况。

如果我们希望提高查全率，尽可能地让所有有可能是恶性肿瘤的病人都得到进一步地检查、诊断，我们可以使用比 0.5 更小的阀值，如 0.3。

我们可以将不同阀值情况下，查全率与查准率的关系绘制成图表，曲线的形状根据数据的不同而不同：



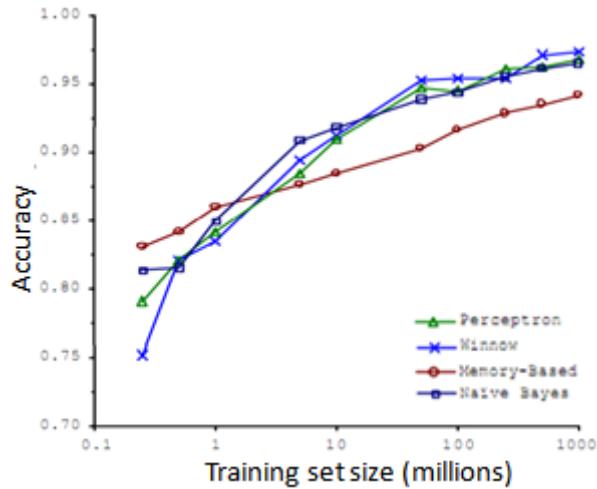
我们希望有一个帮助我们选择这个阀值的方法。一种方法是计算 **F<sub>1</sub> 值** (**F<sub>1</sub> Score**)，其计算公式为：

$$F_1 \text{ Score: } 2 \frac{PR}{P+R}$$

我们选择使得 F<sub>1</sub> 值最高的阀值。

## 10.5 机器学习的数据

Bank 和 Brill 的尝试通过机器学习算法来区分常见的易混淆的单词，他们尝试了许多种不同的算法，并发现，数据量非常大时，这些不同类型的算法效果都很好。我们下面希望探讨，什么时候我们会希望获得更多数据，而非修改算法。



通常情况下，首先思考这样一个问题，“在这些特征面前，一个真人专家是否能有信心地预测结果？”如果回答是肯定的，我们需要再思考我们的模型是怎样的。如果算法是高偏差的，且代价函数很小，那么增加训练集的数据量不太可能导致过拟合，可使得交叉验证误差和训练误差之间差距更小。这种情况下，考虑获得更多数据。

也可以这样来认识，我们希望我们的算法低偏倚，低偏差，我们通过选择更多的特征来降低偏倚，再通过增加数据量来降低偏差。

## WEEK7 支持向量机

### 11 支持向量机 ( SUPPORT VECTOR MACHINE )

支持向量机，SVM，是非常强大且流行的算法，在一些情况下，能面向一些复杂的非线性问题提供比逻辑回归或神经网络要更加简洁的解决方案。

#### 11.1 优化目标 ( OPTIMIZATION OBJECTIVE )

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

以逻辑回归为例展开讨论：回顾逻辑回归模型：

我们分  $y=1$  和  $y=0$  两种情况讨论：

- $y=1$  时，希望假设  $h_{\theta}(x)$  预测的值尽可能接近 1，即希望  $z=\theta^T x$  尽可能地大
- $y=0$  时，希望假设  $h_{\theta}(x)$  预测的值尽可能接近 0，即希望  $z=\theta^T x$  尽可能地小

从代价函数来看，回顾逻辑回归模型的代价函数为：

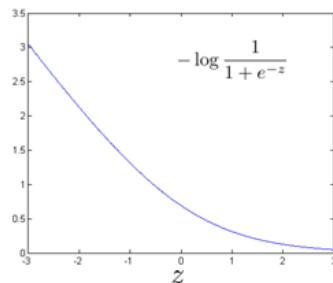
$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_{\theta}(x^{(i)})) \right]$$

针对任何训练集中任何一个实例，对总的代价的影响为：

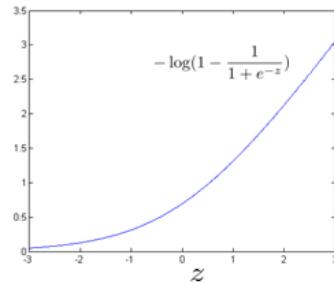
$$\begin{aligned} & -(y \log h_{\theta}(x) + (1 - y) \log(1 - h_{\theta}(x))) \\ \text{即：} & = -y \log \frac{1}{1 + e^{-\theta^T x}} - (1 - y) \log(1 - \frac{1}{1 + e^{-\theta^T x}}) \end{aligned}$$

为了使每一个实例造成的代价都尽可能地小，分  $y=1$  和  $y=0$  两种情况讨论，最佳的情况是代价为 0，但是由曲线可以看出，代价始终存在而非 0。

If  $y = 1$  (want  $\theta^T x \gg 0$ ):

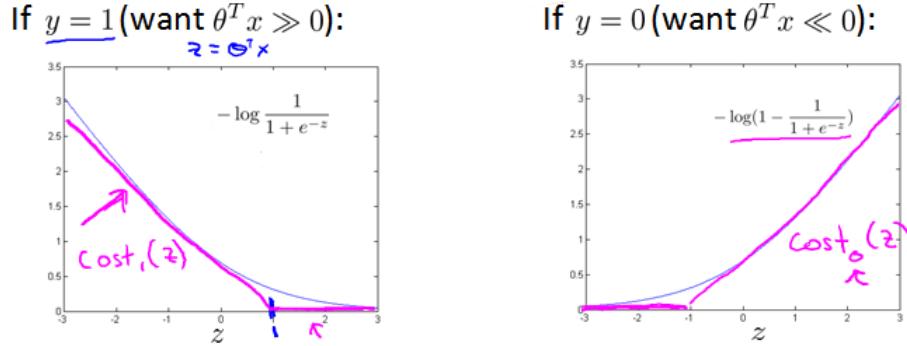


If  $y = 0$  (want  $\theta^T x \ll 0$ ):



在支持向量机中，我们将曲线的代价函数转变成由 2 条线段构成的折线：

- 当  $y=1$  时，我们希望构建新的代价函数如  $\text{cost}_1(z)$  所示，当  $z \geq 1$  时， $\text{cost}_1(z)=0$
- 当  $y=0$  时，我们希望构建新的代价函数如  $\text{cost}_0(z)$  所示，当  $z \leq -1$  时， $\text{cost}_0(z)=0$



用这两个新构建的代价函数代替原本逻辑回归的代价函数,得到：

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

对上面这个代价函数稍作调整：

1. 因为  $1/m$  实际上不影响最优化的结果，将其去掉。
2. 因为归一化参数  $\lambda$  控制的是归一化的这一项在整个代价函数中占的比例，对于支持向量机，我们想要控制的是新构建的代价函数部分，因此我们去掉  $\lambda$  的同时给第一项乘以一个常数  $C$ ，相当于我们将整个代价函数除以了  $\lambda$ ，且  $C=1/\lambda$ 。

我们依旧是希望能找出能使该代价函数最小的参数。注意，调整后的代价函数是一个**凸函数** (convex function)，而非之前逻辑回归那样的非凸函数，这意味着，求解的过程中，不会陷入局部最小值而错过全局最小值的情况：

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

最后，给出支持向量机的假设为：

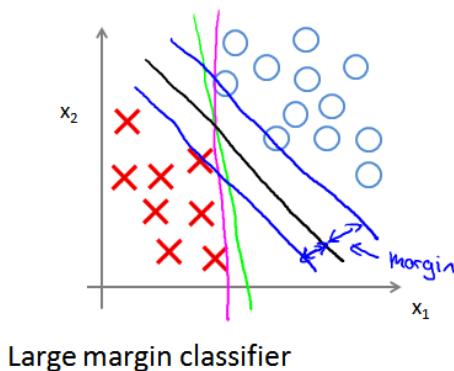
$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{if } \theta^T x < 0 \end{cases}$$

注意到，我们给出的支持向量机假设在预测时是以  $z$  与 0 的大小关系作为依据的，然而在训练函数时，我们是以正负 1 为依据的，这是支持向量机与逻辑回归的一个关键区别，且导致了下面要介绍的支持向量机的特性。

## 11.2 支持向量机判定边界 ( SVM DECISION BOUNDARY )

支持向量机有的时候也被称为**最大间隔分类器** ( Large Margin Classifier )，其原因是：支持向量机可以尝试发现一个与样本数据集之间有着最大间隔的判定边界。

下图是一个可以用直线来区分的分类问题示例，图中绿色和洋红色的两条线代表逻辑回归的判定边界，而黑色的线代表的是支持向量机的判定边界，从图上看出黑色的线似乎是更合理的，蓝色的两条线代表的是支持向量机的判定边界与样本数据之间的间隔。



下面我们思考一下支持向量机中归一化常数  $C$  的作用。

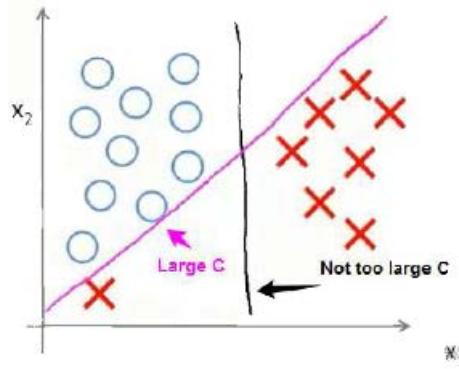
$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

假使我们选择的  $C$  是一个非常大的值，那么在让代价函数最小化的过程中，我们希望找出在  $y=1$  和  $y=0$  两种情况下都使得代价函数中左边的这一项尽量为零的参数。如果我们找到了这样的参数，则我们的最小化问题便转变成：

$$\min \frac{1}{2} \sum_{j=1}^n \theta_j^2 \quad s.t. \quad \begin{cases} \theta^T x^{(i)} \geq 1 & if \ y^{(i)} = 1 \\ \theta^T x^{(i)} \leq -1 & if \ y^{(i)} = 0 \end{cases}$$

这种情况下，我们得出的支持向量机判定边界是上面的黑线那样，具有尝试使得判定边界与样本数据间间隔最大的特性。

然而使得判定边界与样本数据之间间隔最大并不总是好事。假使，我们的数据集如下图所示：



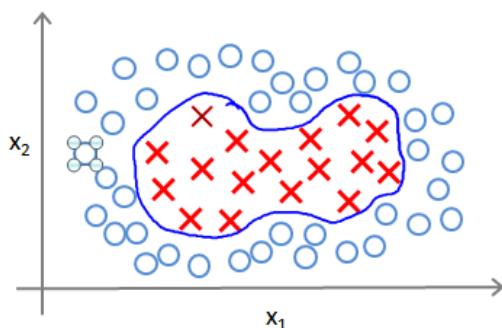
我们发现数据集中间有一个较为明显的异常值，如果我们在选取较大的 C，会导致得到的支持向量机判定边界为图中洋红色直线所示，似乎不是非常的合理。但如果我们选择的 C 较小，那么可能会获得图中黑色直线所示的判定边界。也就是说 C 值越小，支持向量机对异常值越不敏感。

回顾  $C=1/\lambda$ ，因此：

- C 较大时，相当于  $\lambda$  较小，可能会导致过拟合，高偏倚。
- C 较小时，相当于  $\lambda$  较大，可能会导致低拟合，高偏差。

### 11.3 核函数 ( KERNELS )

回顾我们之前讨论过可以使用高级数的多项式模型来解决无法用直线进行分隔的分类问题：

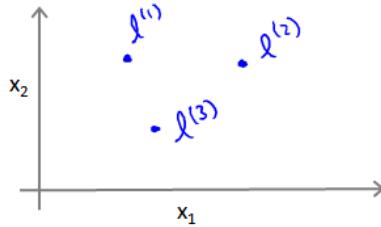


为了获得上图所示的判定边界，我们的模型可能是  $\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \dots$  的形式。我们可以用一系列的新的特征 f 来替换模型中的每一项。例如令：

$f_1 = x_1, f_2 = x_2, f_3 = x_1 x_2, f_4 = x_1^2, f_5 = x_2^2 \dots$  得到  $h_{\theta}(x) = f_1 + f_2 + \dots + f_n$ 。然而，除了对原有的特征进行组合以外，有没有更好的方法来构造  $f_1, f_2, f_3$ ？

我们可以利用核函数来计算出新的特征。

给定一个训练实例  $x$ ，我们利用  $x$  的各个特征与我们预先选定的地标 ( landmarks )  $l^{(1)}, l^{(2)}, l^{(3)}$  的近似程度来选取新的特征  $f_1, f_2, f_3$ 。



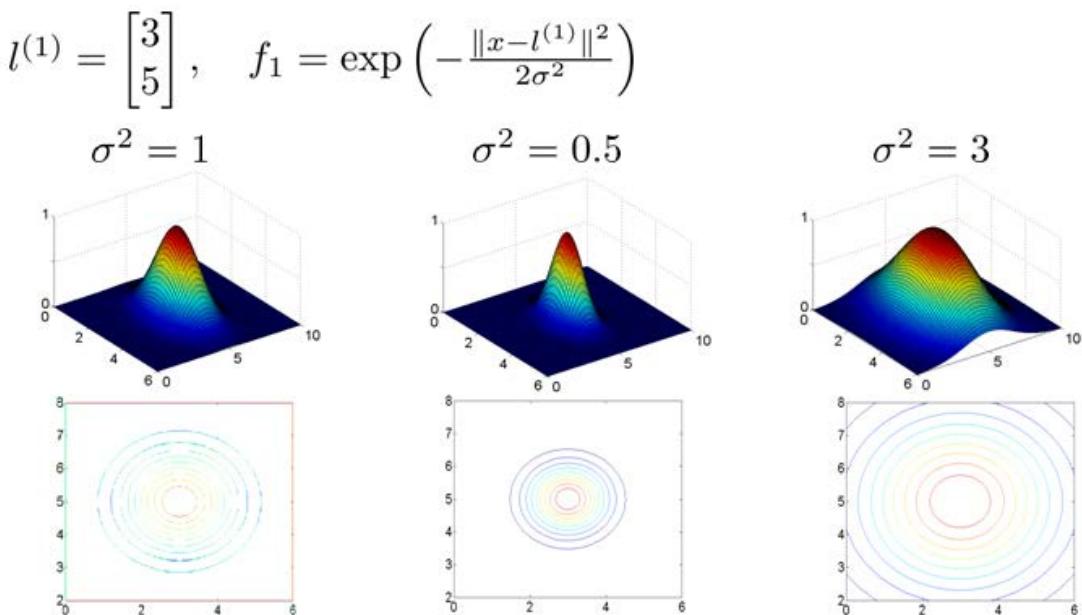
例如：

$$f_1 = \text{similarity}(x, l^{(1)}) = e\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

其中： $\|x - l^{(1)}\|^2 = \sum_{j=1}^n (x_j - l_j^{(1)})^2$ ，为实例  $x$  中所有特征与地标  $l^{(1)}$  之间的距离的和。上例中的  $\text{similarity}(x, l^{(1)})$  就是核函数，具体而言，这里是一个**高斯核函数** ( Gaussian Kernel )。注：这个函数与正态分布没什么实际上的关系，只是看上去像而已。

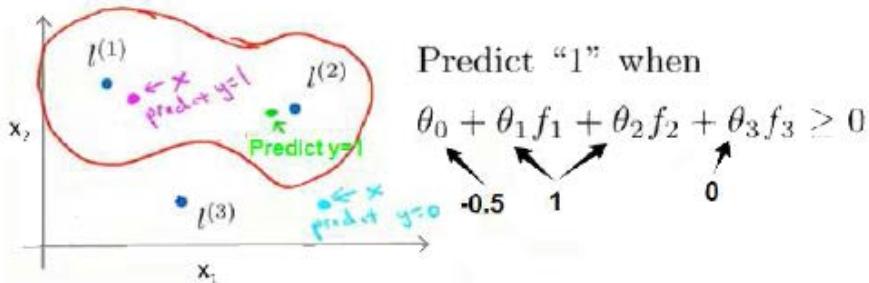
这些地标的作用是什么？如果一个训练实例  $x$  与地标  $L$  之间的距离近似于 0，则新特征  $f$  近似于  $e^{-0}=1$ ，如果训练实例  $x$  与地标  $L$  之间距离较远，则  $f$  近似于  $e^{(-\text{一个较大的数})}=0$ 。

假设我们的训练实例含有两个特征  $[x_1 \ x_2]$ ，给定地标  $l^{(1)}$  与不同的  $\sigma$  值，见下图：



图中水平面的坐标为  $x_1, x_2$  而垂直坐标轴代表  $f$ 。可以看出，只有当  $x$  与  $|^{(1)}$  重合时  $f$  才具有最大值。随着  $x$  的改变  $f$  值改变的速率受到  $\sigma^2$  的控制。

在下图中，当实例处于洋红色的点位置处，因为其离  $|^{(1)}$  更近，但是离  $|^{(2)}$  和  $|^{(3)}$  较远，因此  $f_1$  接近 1，而  $f_2, f_3$  接近 0。因此  $h_\theta(x) = \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 > 0$ ，因此预测  $y=1$ 。同理可以求出，对于离  $|^{(2)}$  较近的绿色点，也预测  $y=1$ ，但是对于蓝绿色的点，因为其离三个地标都较远，预测  $y=0$ 。



这样，图中红色的封闭曲线所表示的范围，便是我们依据一个单一的训练实例和我们选取的地标所得出的判定边界，在预测时，我们采用的特征不是训练实例本身的特征，而是通过核函数计算出的新特征  $f_1, f_2, f_3$ 。

如何选择地标？

我们通常是根据训练集的数量选择地标的数量，即如果训练集中有  $m$  个实例，则我们选取  $m$  个地标，并且令： $|^{(1)}=x^{(1)}, |^{(2)}=x^{(2)}, \dots, |^{(m)}=x^{(m)}$ 。这样做好处在于：现在我们得到的新特征是建立在原有特征与训练集中所有其他特征之间距离的基础之上的，即：

$$f^{(i)} = \begin{bmatrix} f_0^{(i)} = 1 \\ f_1^{(i)} = sim(x^{(i)}, l^{(1)}) \\ f_2^{(i)} = sim(x^{(i)}, l^{(2)}) \\ f_i^{(i)} = sim(x^{(i)}, l^{(i)}) = e^0 = 1 \\ \vdots \\ f_m^{(i)} = sim(x^{(i)}, l^{(m)}) \end{bmatrix}$$

下面我们将核函数运用到支持向量机中，修改我们的支持向量机假设为：

- 给定  $x$ ，计算新特征  $f$ ，当  $\theta^T f >= 0$  时，预测  $y=1$ ，否则反之。

相应地修改代价函数为：

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

在具体实施过程中，我们还需要对最后的归一化项进行些微调整，在计算  $\sum_{j=1}^{n=m} \theta_j^2 = \theta^T \theta$  时，我们用  $\theta^T M \theta$  代替  $\theta^T \theta$ ，其中  $M$  是根据我们选择的核函数而不同的一个矩阵。这样做的原因是简化计算。

理论上讲，我们也可以在逻辑回归中使用核函数，但是上面使用  $M$  来简化计算的方法不适用于逻辑回归，因此计算将非常耗费时间。

在此，我们不介绍最小化支持向量机的代价函数的方法，你可以使用现有的软件包（如 liblinear, libsvm 等）。在使用这些软件包最小化我们的代价函数之前，我们通常需要编写核函数，并且如果我们使用高斯核函数，那么在使用之前进行特征缩放是非常必要的。

另外，支持向量机也可以不使用核函数，不使用核函数又称为**线性核函数** ( linear kernel )，当我们不采用非常复杂的函数，或者我们的训练集特征非常多而实例非常少的时候，可以采用这种不带核函数的支持向量机。

下面是支持向量机的两个参数  $C$  和  $\sigma$  的影响：

- $C$  较大时，相当于  $\lambda$  较小，可能会导致过拟合，高偏倚
- $C$  较小时，相当于  $\lambda$  较大，可能会导致低拟合，高偏差
- $\sigma$  较大时，导致高偏倚
- $\sigma$  较大时，导致高偏差

在高斯核函数之外我们还有其他一些选择，如：

- 多项式核函数 ( Polynomial Kernel )
- 字符串核函数 ( String kernel )
- 卡方核函数 ( chi-square kernel )
- 直方图交集核函数 ( histogram intersection kernel )
- etc...

这些核函数的目标也都是根据训练集和地标之间的距离来构建新特征，这些核函数需要满足 Mercer's 定理，才能被支持向量机的优化软件正确处理。

## 多类分类问题

假设我们利用之前介绍的一对多方法来解决一个多类分类问题。如果一共有  $k$  个类，则我们需要  $k$  个模型，以及  $k$  个参数向量  $\theta$ 。我们同样也可以训练  $k$  个支持向量机来解决多类分类问题。但是大多数支持向量机软件包都有内置的多类分类功能，我们只要直接使用即可。

### 11.4 逻辑回归与支持向量机

从逻辑回归模型，我们得到了支持向量机模型，在两者之间，我们应该如何选择呢？

下面是一些普遍使用的准则：

- 如果相较于  $m$  而言， $n$  要大许多，即训练集数据量不够支持我们训练一个复杂的非线性模型，我们选用逻辑回归模型或者不带核函数的支持向量机。
- 如果  $n$  较小，而且  $m$  大小中等，例如  $n$  在 1-1000 之间，而  $m$  在 10-10000 之间，使用带高斯核函数的支持向量机。
- 如果  $n$  较小，而  $m$  较大，例如  $n$  在 1-1000 之间，而  $m$  大于 50000，则使用支持向量机会非常慢，解决方案是创造、增加更多的特征，然后使用逻辑回归或不带核函数的支持向量机。

值得一提的是，神经网络在以上三种情况下都可能会有较好的表现，但是训练神经网络可能非常慢，选择支持向量机的原因主要在于它的代价函数是凸函数，不存在局部最小值。

### 12 聚类 ( CLUSTERING )

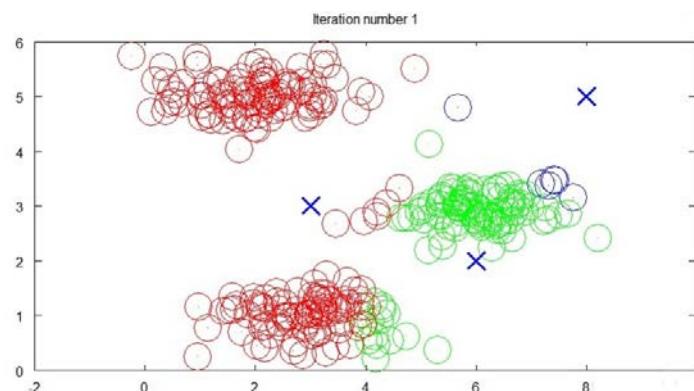
#### 12.1 K-均值算法

K-均值是最普及的聚类算法，算法接受一个未标记的数据集，然后将数据聚类成不同的组。

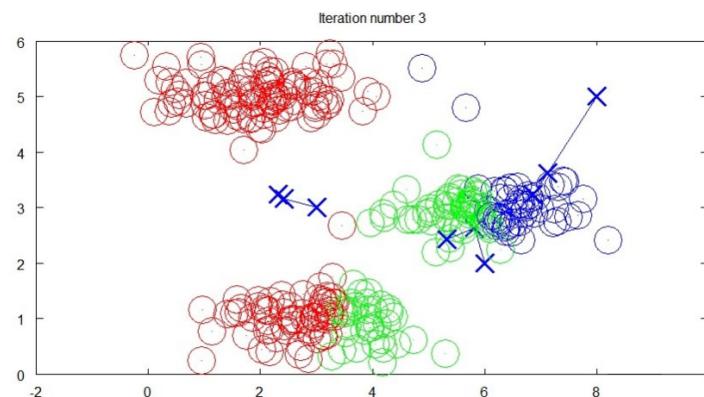
K-均值是一个迭代算法，假设我们想要将数据聚类成 n 个组，其方法为：

1. 首先选择 K 个随机的点，称为**聚类中心** ( cluster centroids )
2. 对于数据集中的每一个数据，按照距离 K 个中心点的距离，将其与距离最近的中心点关联起来，与同一个中心点关联的所有点聚成一类
3. 计算每一个组的平均值，将该组所关联的中心点移动到平均值的位置
4. 重复步骤 2-4 直至中心点不再变化

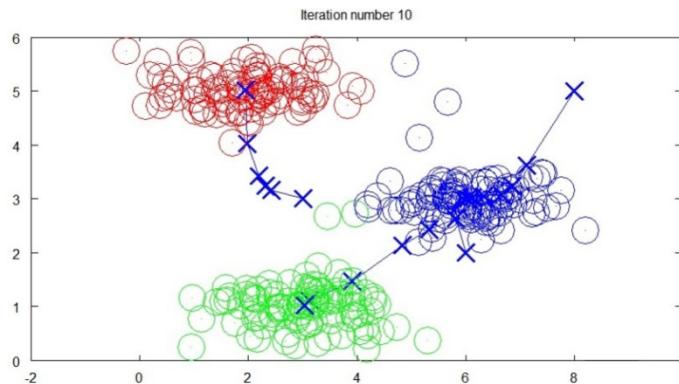
下面是一个聚类示例：



迭代 1 次



迭代 3 次



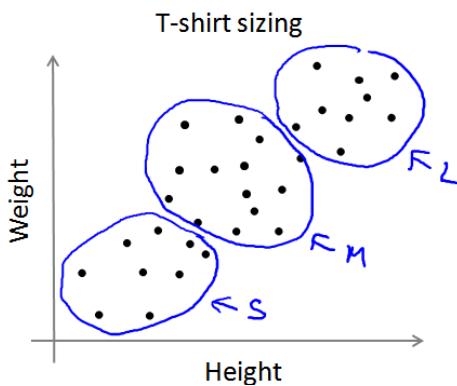
迭代 10 次

用  $\mu_1, \mu_2, \dots, \mu_k$  来表示聚类中心，用  $c^{(1)}, c^{(2)}, \dots, c^{(m)}$  来存储与第  $i$  个实例数据最近的聚类中心的索引，K-均值算法的伪代码如下：

```
Repeat {  
for i = 1 to m  
     $c^{(i)} := \text{index (from 1 to } K \text{) of cluster centroid closest to } x^{(i)}$   
for k = 1 to K  
     $\mu_k := \text{average (mean) of points assigned to cluster } k$   
}
```

算法分为两个步骤，第一个 for 循环是赋值步骤，第二个 for 循环是聚类中心的移动。

K-均值算法也可以很便利地用于将数据分为许多不同组，即使在没有非常明显区分的组群的情况下也可以。下图所示的数据集包含身高和体重两项特征构成的，利用 K-均值算法将数据分为三类，用于帮助确定将要生产的 T-恤衫的三种尺寸。



## 12.2 优化目标

K-均值最小化问题，是要最小化所有的数据点与其所关联的聚类中心点之间的距离之和，因此 K-均值的代价函数（又称畸变函数 Distortion function）为：

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

其中  $\mu_{c^{(i)}}$  代表与  $x^{(i)}$  最近的聚类中心点。

我们的优化目标便是找出使得代价函数最小的  $c^{(1)}, c^{(2)}, \dots, c^{(m)}$  和  $\mu_1, \mu_2, \dots, \mu_K$ ：

$$\min_{\substack{c^{(1)}, \dots, c^{(m)}, \\ \mu_1, \dots, \mu_K}} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

回顾刚才给出的 K-均值迭代算法，我们知道，第一个循环是用于减小  $c^{(i)}$  引起的代价，而第二个循环则是用于减小  $\mu_i$  引起的代价。迭代的过程一定会是每一次迭代都在减小代价函数，不然便是出现了错误。

## 12.3 随机初始化

在运行 K-均值算法的之前，我们首先要随机初始化所有的聚类中心点，下面介绍怎样做：

1. 我们应该选择  $K < m$ ，即聚类中心点的个数要小于所有训练集实例的数量
2. 随机选择  $K$  个训练实例，然后令  $K$  个聚类中心分别与这  $K$  个训练实例相等

K-均值的一个问题在于，它有可能会停留在一个局部最小值处，而这取决于初始化的情况。

为了解决这个问题，我们通常需要多次运行 K-均值算法，每一次都重新进行随机初始化，最后再比较多次运行 K-均值的结果，选择代价函数最小的结果。这种方法在  $K$  较小的时候（2-10）还是可行的，但是如果  $K$  较大，这么作也可能不会有明显地改善。

## 12.4 选择聚类数

没有所谓最好的选择聚类数的方法，通常是需要根据不同的问题，人工进行选择的。选择的时候思考我们运用 K-均值算法聚类的动机是什么，然后选择能最好服务于该目的聚类数。

例如，我们的 T-恤制造例子中，我们要将用户按照身材聚类，我们可以分成 3 个尺寸 S,M,L 也可以分成 5 个尺寸 XS,S,M,L,XL，这样的选择是建立在回答“聚类后我们制造的 T-恤是否能较好地适合我们的客户”这个问题的基础上作出的。

## WEEK8 降维

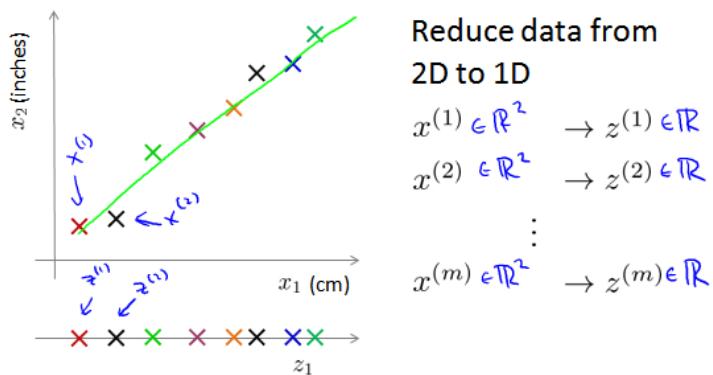
### 13 降维 ( DIMENSIONALITY REDUCTION )

#### 13.1 动机一：数据压缩 ( DATA COMPRESSION )

通过几个例子来介绍降维。

将数据从二维降至一维：

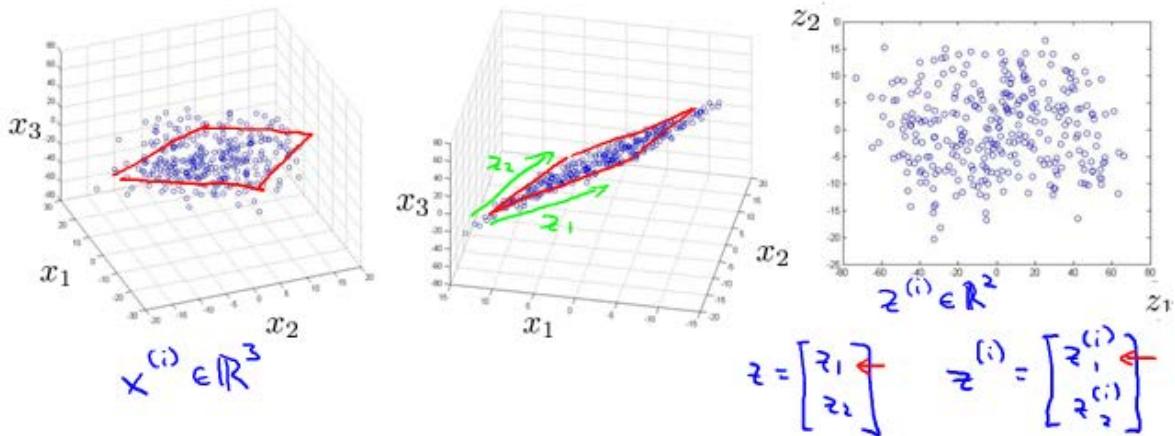
假使我们要采用两种不同的仪器来测量一些东西的尺寸，其中一个仪器测量结果的单位是英寸，另一个仪器测量的结果是厘米，我们希望将测量的结果作为我们机器学习的特征。现在的问题的是，两种仪器对同一个东西测量的结果不完全相等（由于误差、精度等），而将两者都作为特征有些重复，因而，我们希望将这个二维的数据降至一维。



具体做法是，我们找出一条合适的直线，然后将所有的数据点都投影到该直线上，然后用  $z^{(i)}$  标识，这样我们便完成了二维数据  $x^{(i)}$  向一维数据  $z^{(i)}$  的映射。这样新的得到的特征只是原有特征的近似，但是这样做将我们的存储、内存占用量减半，并且使我们要使用这些数据的算法可以运行得更快。

将数据从三维降至二维：

这个例子中我们要将一个三维的特征向量降至一个二维的特征向量。过程是与上面类似的，我们将三维向量投射到一个二维的平面上，强迫使得所有的数据都在同一个平面上，降至二维的特征向量。



这样的处理过程可以被用于把任何维度的数据降到任何想要的维度，例如将 1000 维的特征降至 100 维。

### 13.2 动机二：数据可视化 ( DATA VISUALIZATION )

在许多及其学习问题中，如果我们能将数据可视化，我们便能寻找到一个更好的解决方案，降维可以帮助我们。

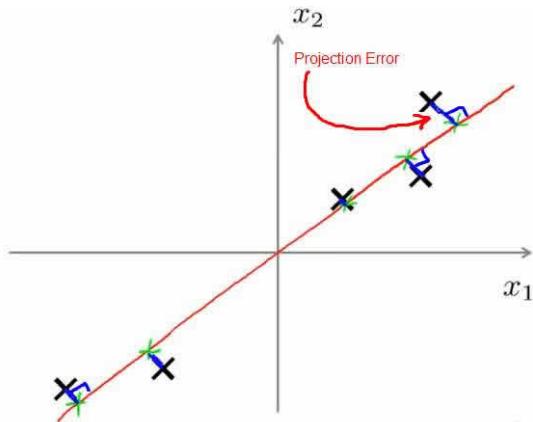
假使我们有有关于许多不同国家的数据，每一个特征向量都有 50 个特征（如，GDP，人均 GDP，平均寿命等）。如果要将这个 50 维的数据可视化是不可能的。使用降维的方法将其降至 2 维，我们便可以将其可视化了。

这样做的问题在于，降维的算法只负责减少唯独，新产生的特征的意义就必须由我们自己去发现了。

### 13.3 主要成分分析 ( PRINCIPAL COMPONENT ANALYSIS )

主要成分分析是最常见的降维算法。

在 PCA 中，我们要做的是找到一个**方向向量** ( Vector direction )，当我们把所有的数据都投射到该向量上时，我们希望投射平均均方误差能尽可能地小。方向向量是一个经过原点的向量，而投射误差是从特征向量向该方向向量作垂线的长度。

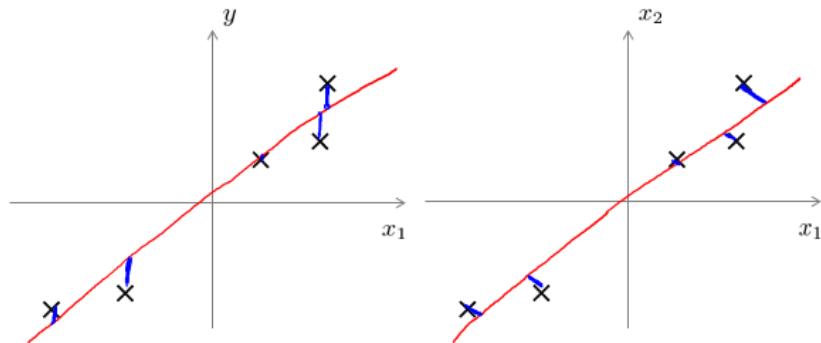


下面给出主要成分分析问题的描述：

- 问题是将  $n$  维数据降至  $k$  维
- 目标是找到向量  $u^{(1)}, u^{(2)}, \dots, u^{(k)}$  使得总的投射误差最小

主要成分分析与线性回顾的比较：

主要成分分析与线性回归是两种不同的算法。主要成分分析最小化的是投射误差而线性回归尝试的是最小化预测误差。线性回归的目的是预测结果，而主要成分分析不作任何预测。



上图中，左边的是线性回归的误差，右边则是主要成分分析的误差。

#### 13.4 主要成分分析算法

第一步是均值归一化。我们需要计算出所有特征的均值，然后令  $x_j = x_j - \mu_j$ 。如果特征是在不同的数量级上，我们还需要将其除以标准差  $\sigma^2$ 。

第二步是计算协方差矩阵 ( covariance matrix )  $\Sigma$  :

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)}) (x^{(i)})^T$$

第三步是计算协方差矩阵的**特征向量** ( eigenvectors ) :

在 Octave 里我们可以利用**奇异值分解** ( singular value decomposition ) 来求解 ,  $[U, S, V] = svd(\sigma)$ 。

对于一个  $n \times n$  维度的矩阵 , 上式中的  $U$  是一个具有与数据之间最小投射误差的方向向量构成的矩阵。如果我们希望将数据从  $n$  维降至  $k$  维 , 我们只需要从  $U$  中选取前  $K$  个向量 , 获得一个  $n \times k$  维度的矩阵 , 我们用  $U_{reduce}$  表示 , 然后通过如下计算获得要求的新特征向量  $z^{(i)}$  :

$$z^{(i)} = U_{reduce}^T \times x^{(i)}$$

其中  $x$  是  $n \times 1$  维的 , 因此结果为  $k \times 1$  维度。

注 , 我们不对偏倚特征进行处理。

### 13.5 选择主要成分的数量

主要成分分析是减少投射的平均均方误差 :

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

训练集的方差为 :

$$\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$$

我们希望在平均均方误差与训练集方差的比例尽可能小的情况下选择尽可能小的  $K$  值。

如果我们希望这个比例小于 1% , 就意味着原本数据的偏差有 99% 都保留下来了 , 如果我们选择保留 95% 的偏差 , 便能非常显著地降低模型中特征的维度了。

我们可以先令  $K=1$  , 然后进行主要成分分析 , 获得  $U_{reduce}$  和  $z$  , 然后计算比例是否小于 1%。如果不是的话再令  $K=2$  , 如此类推 , 直到找到可以使得比例小于 1% 的最小  $K$  值 ( 原因是各个特征之间通常情况存在某种相关性 ) 。

还有一些更好的方式来选择  $K$  , 当我们在 Octave 中调用 “svd” 函数的时候 , 我们获得三个参数 :  $[U, S, V] = svd(\sigma)$ 。

$$S = \begin{bmatrix} S_{11} & & & \\ & S_{22} & & \\ & & S_{33} & \\ & & & \ddots \\ & & & S_{nn} \end{bmatrix}$$

其中的  $S$  是一个  $n \times n$  的矩阵，只有对角线上有值，而其它单元都是 0，我们可以使用这个矩阵来计算平均均方误差与训练集方差的比例：

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

也就是：

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

在压缩过数据后，我们可以采用如下方法来近似地获得原有的特征：

$$x_{approx}^{(i)} = U_{reduce} z^{(i)}$$

### 13.6 应用主要成分分析

假使我们正在针对一张  $100 \times 100$  像素的图片进行某个计算机视觉的机器学习，即总共有 10000 个特征。

1. 第一步是运用主要成分分析将数据压缩至 1000 个特征
2. 然后对训练集运行学习算法
3. 在预测时，采用之前学习而来的  $U_{reduce}$  将输入的特征  $x$  转换成特征向量  $z$ ，然后再进行预测

注，如果我们有交叉验证集合测试集，也采用对训练集学习而来的  $U_{reduce}$ 。

错误的主要成分分析情况：

一个常见错误使用主要成分分析的情况是，将其用于减少过拟合（减少了特征的数量）。这样做非常不好，不如尝试归一化处理。原因在于主要成分分析只是近似地丢弃掉一些特征，它并

不考虑任何与结果变量有关的信息，因此可能会丢失非常重要的特征。然而当我们进行归一化处理时，会考虑到结果变量，不会丢掉重要的数据。

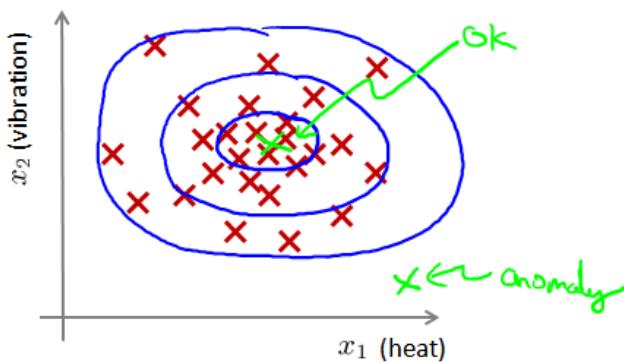
另一个常见的错误是，默认地将主要成分分析作为学习过程中的一部分，这虽然很多时候有效果，最好还是从所有原始特征开始，只在有必要的时候（算法运行太慢或者占用太多内存）才考虑采用主要成分分析。

## 14 异常检测 ( ANOMALY DETECTION )

异常检测是一个非监督学习算法，用于发现可能不应该属于一个已定义的组中的数据。

## 14.1 密度估计 ( DENSITY ESTIMATION )

给定数据集  $x_{(1)}, x_{(2)}, \dots, x_{(m)}$ ，我们假使数据集是正常的，我们希望知道新的数据  $x_{(test)}$  是不是异常的，即这个测试数据不属于该组数据的几率如何。我们所构建的模型应该能根据该测试数据的位置告诉我们其属于一组数据的可能性  $p(x)$ 。



上图中，在蓝色圈内的数据属于该组数据的可能性较高，而越是偏远的数据，其属于该组数据的可能性就越低。

这种方法称为密度估计，表达如下：

$$\text{if } p(x) \begin{cases} \leq \epsilon & \text{anomaly} \\ > \epsilon & \text{normal} \end{cases}$$

异常检测主要用来识别欺骗。

例如在线采集而来的有关用户的数据，一个特征向量中可能会包含如：用户多久登录一次，访问过的页面，在论坛发布的帖子数量，甚至是打字速度等。尝试根据这些特征构建一个模型，可以用这个模型来识别那些不符合该模式的用户。

再一个例子是检测一个数据中心，特征可能包含：内存使用情况，被访问的磁盘数量，CPU的负载，网络的通信量等。根据这些特征可以构建一个模型，用来判断某些计算机是不是有可能出错了。

## 14.2 高斯分布

回顾高斯分布的基本知识。

通常如果我们认为变量  $x$  符合高斯分布  $\text{D}x \sim N(\mu, \sigma^2)$  则其概率密度函数为：

$$P(x, \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

我们可以利用已有的数据来预测总体中的  $\mu$  和  $\sigma^2$  的计算方法如下：

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

注：机器学习中对于方差我们通常只除以  $m$  而非统计学中的  $(m-1)$ 。

## 14.3 异常检测

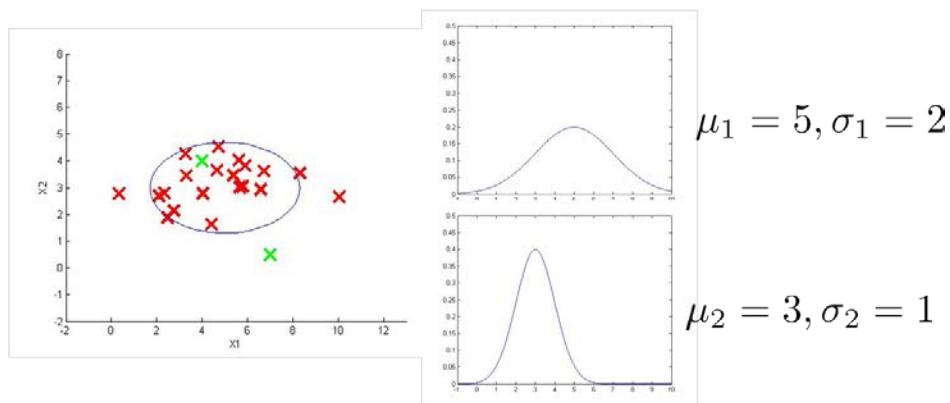
对于给定的数据集  $x_{(1)}, x_{(2)}, \dots, x_{(m)}$ ，我们要针对每一个特征计算  $\mu$  和  $\sigma^2$  的估计值。

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

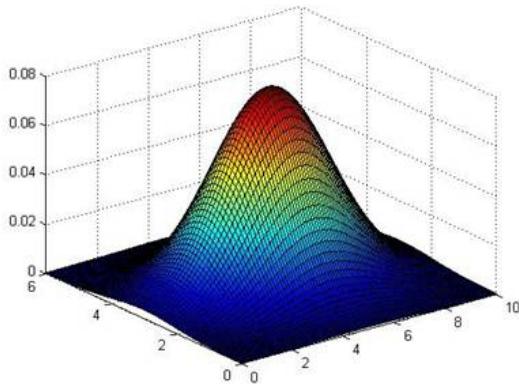
一旦我们获得了平均值和方差的估计值，给定新的一个训练实例，根据模型计算  $p(x)$ ：

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

下图是一个由两个特征的训练集，以及特征的分布情况：



下面的三维图表表示的是密度估计函数，z 轴为根据两个特征的值所估计  $p(x)$  值：



我们选择一个 $\varepsilon$ ，将  $p(x)=\varepsilon$ 作为我们的判定边界，当  $p(x)>\varepsilon$ 时预测数据为正常数据，否则则为异常。

#### 14.4 评价一个异常检测系统

异常检测算法是一个非监督学习算法，意味着我们无法根据结果变量  $y$  的值来告诉我们数据是否真的是异常的。我们需要另一种方法来帮助检验算法是否有效。当我们开发一个异常检测系统时，我们从带标记（异常或正常）的数据着手，我们从其中选择一部分正常数据用于构建训练集，然后用剩下的正常数据和异常数据混合的数据构成交叉检验集和测试集。

例如：我们有 10000 台正常引擎的数据，有 20 台异常引擎的数据。

我们这样分配数据：

- 6000 台正常引擎的数据作为训练集
- 2000 台正常引擎和 10 台异常引擎的数据作为交叉检验集
- 2000 台正常引擎和 10 台异常引擎的数据作为测试集

具体的评价方法如下：

1. 根据测试集数据，我们估计特征的平均值和方差并构建  $p(x)$  函数
2. 对交叉检验集，我们尝试使用不同的 $\varepsilon$ 值作为阀值，并预测数据是否异常，根据 F1 值或者查准率与查全率的比例来选择 $\varepsilon$
3. 选出 $\varepsilon$ 后，针对测试集进行预测，计算异常检验系统的 F1 值或者查准率与查全率之比

#### 14.5 异常检测与监督学习对比

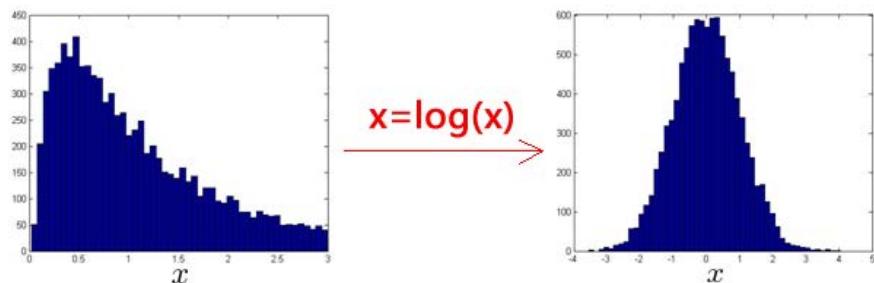
之前我们构建的异常检测系统也使用了带标记的数据，与监督学习有些相似，下面的对比有助于选择采用监督学习还是异常检测：

异常检测	监督学习
非常少量的正向类（异常数据 $y=1$ ），同时有大量的正向类和负向类 大量的负向类 ( $y=0$ )	
许多不同种类的异常，非常难根据非常少量的正向类数据来训练算法。	有足够的正向类实例，足够用于训练算法，未来遇到的正向类实例可能与训练集中的非常近似。
未来遇到的异常可能与已掌握的异常非常的不同。	
例如： <ul style="list-style-type: none"><li>1. 欺诈行为检测</li><li>2. 生产（例如飞机引擎）</li><li>3. 检测数据中心的计算机运行状况</li></ul>	例如： <ul style="list-style-type: none"><li>1. 邮件过滤器</li><li>2. 天气预报</li><li>3. 肿瘤分类</li></ul>

## 14.6 选择特征

对于异常检测算法，我们使用的特征是至关重要的，下面谈谈如何选择特征：

异常检测假设特征符合高斯分布，如果数据的分布不是高斯分布，异常检测算法也能够工作，但是最好还是将数据转换成高斯分布，例如使用对数函数  $x = \log(x+c)$ ，其中  $c$  为非负常数；或者  $x=x^c$ ， $c$  为 0-1 之间的一个分数，等方法。



误差分析：

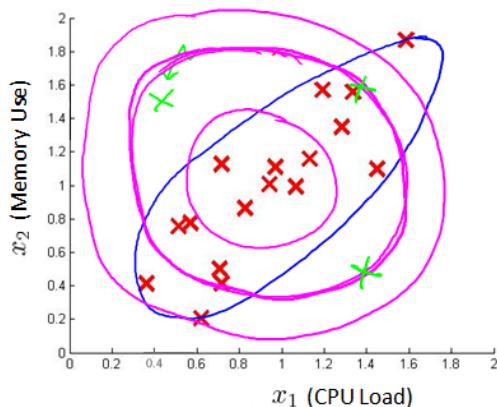
一个常见的问题是，一些异常的数据可能也会有较高的  $p(x)$  值，因而被算法认为是正常的。这种情况下误差分析能够帮助我们，我们可以分析那些被算法错误预测为正常的数据，观察能否找出一些问题。我们可能能从问题中发现我们需要增加一些新的特征，增加这些新特征后获得的新算法能够帮助我们更好地进行异常检测。

我们通常可以通过将一些相关的特征进行组合，来获得一些新的更好的特征（异常数据的该特征值异常地大或小），例如，在检测数据中心的计算机状况的例子中，我们可以用 CPU 负载与网络通信量的比例作为一个新的特征，如果该值异常地大便有可能意味着该服务器是陷入了一些问题中。

#### 14.7 多元高斯分布(MULTIVARIATE GAUSSIAN DISTRIBUTION)

假使我们有两个相关的特征，而且这两个特征的值域范围比较宽，这种情况下，一般的高斯分布模型可能不能很好地识别异常数据。其原因在于，一般的高斯分布模型尝试的是去同时抓住两个特征的偏差，因此创造出一个比较大的判定边界。

下图中是两个相关特征，洋红色的线（根据  $\epsilon$  的不同其范围可大可小）是一般的高斯分布模型获得的判定边界，很明显绿色的 X 所代表的数据点很可能是异常值，但是其  $p(x)$  值却仍然在正常范围内。多元高斯分布将创建像图中蓝色曲线所示的判定边界。



在一般的高斯分布模型中，我们计算  $p(x)$  的方法是：

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

通过分别计算每个特征对应的几率然后将其累乘起来，在多元高斯分布模型中，我们将构建特征的协方差矩阵，用所有的特征一起来计算  $p(x)$ 。

我们首先计算所有特征的平均值，然后再计算协方差矩阵：

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T = \frac{1}{m} (X - \mu)^T (X - \mu)$$

注：其中  $\mu$  是一个向量，其每一个单元都是原特征矩阵中一行数据的均值。

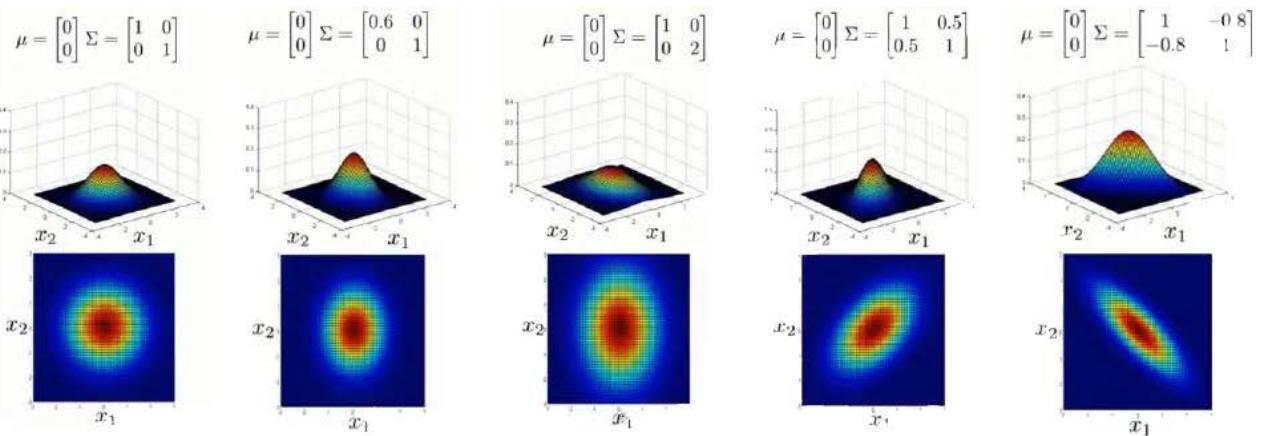
最后我们计算多元高斯分布的  $p(x)$ ：

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

其中：

- $|\Sigma|$  是定矩阵，在 Octave 中用 `det(sigma)` 计算
- $\Sigma^1$  是逆矩阵

下面我们来看看协方差矩阵是如何影响模型的：



上图是 5 个不同的模型，从左往右依次分析：

1. 是一个一般的高斯分布模型
2. 通过协方差矩阵，令特征 1 拥有较小的偏差，同时保持特征 2 的偏差
3. 通过协方差矩阵，令特征 2 拥有较大的偏差，同时保持特征 1 的偏差
4. 通过协方差矩阵，在不改变两个特征的原有偏差的基础上，增加两者之间的正相关性
5. 通过协方差矩阵，在不改变两个特征的原有偏差的基础上，增加两者之间的负相关性

多元高斯分布模型与原高斯分布模型的关系：

可以证明的是，原本的高斯分布模型是多元高斯分布模型的一个子集，即像上图中的第 1、2、3，3 个例子所示，如果协方差矩阵只在对角线的单位上有非零的值时，即为原本的高斯分布模型了。

原高斯分布模型和多元高斯分布模型的比较：

原高斯分布模型	多元高斯分布模型
不能捕捉特征之间的相关性但可以通过将特征进行组合的方法来解决	自动捕捉特征之间的相关性
计算代价低，能适应大规模的特征	计算代价较高
训练集较小时也同样适用	必须要有 $m > n$ ，不然的话协方差矩阵不可逆的，通常需要 $m > 10n$ 另外特征冗余也会导致协方差矩阵不可逆

原高斯分布模型被广泛使用着，如果特征之间在某种程度上存在相互关联的情况，我们可以通过构造新特征的方法来捉补这些相关性。

如果训练集不是太大，并且没有太多的特征，我们可以使用多元高斯分布模型

## 15 推荐系统 ( RECOMMENDER SYSTEMS )

## 15.1 问题形式化

我们从一个例子开始定义推荐系统的问题。

假使我们是一个电影供应商，我们有 5 部电影和 4 个用户，我们要求用户为电影打分。

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	6
Romance forever	5	?	0	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	0	0
Swords vs. karate	0	0	0	0

前三部电影是爱情片，后两部则是动作片，我们可以看出 Alice 和 Bob 似乎更倾向与爱情片，而 Carol 和 Dave 似乎更倾向与动作片。并且没有一个用户给所有的电影都打过分。我们希望构建一个算法来预测他们每个人可能会给他们没看过的电影打多少分，并以此作为推荐的依据。

下面引入一些标记：

- $n_u$  代表用户的数量
- $n_m$  代表电影的数量
- $r(i,j)$  如果用户  $i$  给电影  $j$  评过分则  $r(i,j)=1$
- $y^{(i,j)}$  代表用户  $i$  给电影  $j$  的评分
- $m_j$  代表用户  $j$  评过分的电影的总数

## 15.2 基于内容的推荐系统 ( CONTENT-BASED RECOMMENDATIONS )

在一个基于内容的推荐系统算法中，我们假设对于我们希望推荐的东西有一些数据，这些数据是有关这些东西的特征。

在我们的例子中，我们可以假设每部电影都有两个特征，如  $x_1$  代表电影的浪漫程度， $x_2$  代表电影的动作程度。

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

则每部电影都有一个特征向量，如  $x^{(1)}$  是第一部电影的特征向量为 [0.9 0]。

下面我们要基于这些特征来构建一个推荐系统算法。

假设我们采用线性回归模型，我们可以针对每一个用户都训练一个线性回归模型，如  $\theta^{(1)}$  是第一个用户的模型的参数。

于是，我们有：

- $\theta^{(j)}$  用户 j 的参数向量
- $x^{(i)}$  电影 i 的特征向量

对于用户 j 和电影 i，我们预测评分为： $(\theta^{(j)})^T(x^{(i)})$

### 代价函数

针对用户 j，该线性回归模型的代价为预测误差的平方和，加上归一化项：

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

其中  $i:r(i,j)$  表示我们只计算那些用户 j 评过分的电影。在一般的线性回归模型中，误差项和归一化项应该都是乘以  $1/2m$ ，在这里我们将 m 去掉。并且我们不对偏倚项  $\theta_0$  进行归一化处理。

上面的代价函数只是针对一个用户的，为了学习所有用户，我们将所有用户的代价函数求和：

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

如果我们要用梯度下降法来求解最优解，我们计算代价函数的偏导数后得到梯度下降的更新公式为：

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

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

### 15.3 协同过滤算法 ( COLLABORATIVE FILTERING ALGORITHM )

在之前的基于内容的推荐系统中，对于每一部电影，我们都掌握了可用的特征，使用这些特征训练出了每一个用户的参数。相反地，如果我们拥有用户的参数，我们可以学习得出电影的特征。

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

但是如果我们既没有用户的参数，也没有电影的特征，这两种方法都不可行了。协同过滤算法可以同时学习这两者。

我们的优化目标便改为同时针对  $x$  和  $\theta$  进行。

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

对代价函数求偏导数的结果如下：

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

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

注：在协同过滤从算法中，我们通常不使用偏倚项，如果需要的话，算法会自动学得。

协同过滤算法使用步骤如下：

1. 初始  $x^{(1)}, x^{(2)}, \dots, x^{(n_m)}$ ,  $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(n_u)}$  为一些随机小值
2. 使用梯度下降算法最小化代价函数
3. 在训练完算法后，我们预测  $(\theta^{(j)})^T (x^{(i)})$  为用户  $j$  给电影  $i$  的评分

通过这个学习过程获得的特征矩阵包含了有关电影的重要数据，这些数据不总是人能读懂的，但是我们可以用这些数据作为给用户推荐电影的依据。

例如，如果一位用户正在观看电影  $x^{(i)}$ ，我们可以寻找另一部电影  $x^{(j)}$ ，依据两部电影的特征向量之间的距离  $\|x^{(i)} - x^{(j)}\|$  的大小。

## 15.4 均值归一化

让我们来看下面的用户评分数据：

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)	$Y =$
Love at last	5	5	0	0	?	$\begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \end{bmatrix}$
Romance forever	5	?	?	0	?	$\begin{bmatrix} ? & 4 & 0 & ? & ? \end{bmatrix}$
Cute puppies of love	?	4	0	?	?	$\begin{bmatrix} 0 & 0 & 5 & 4 & ? \end{bmatrix}$
Nonstop car chases	0	0	5	4	?	$\begin{bmatrix} 0 & 0 & 5 & 0 & ? \end{bmatrix}$
Swords vs. karate	0	0	5	?	?	

如果我们新增一个用户 Eve，并且 Eve 没有为任何电影评分，那么我们以什么为依据为 Eve 推荐电影呢？

我们首先需要对结果 Y 矩阵进行均值归一化处理 将每一个用户对某一部电影的评分减去所有用户对该电影评分的平均值：

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix} \quad \mu = \begin{bmatrix} 2.5 \\ 2.5 \\ 2 \\ 2.25 \\ 1.25 \end{bmatrix} \rightarrow Y = \begin{bmatrix} 2.5 & 2.5 & -2.5 & -2.5 & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

然后我们利用这个新的 Y 矩阵来训练算法。

如果我们要用新训练出的算法来预测评分，则需要将平均值重新加回去，预测  $(\Theta^{(j)})^T(x^{(i)}) + \mu_i$

对于 Eve，我们的新模型会认为她给每部电影的评分都是该电影的平均分。

## 16 大规模机器学习 ( LARGE SCALE MACHINE LEARNING )

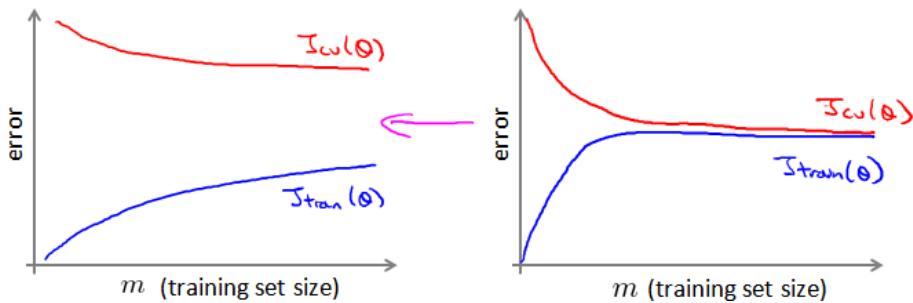
## 16.1 大型数据集的学习

如果我们有一个低偏倚的模型，增加数据集的规模可以帮助你获得更好的结果。

我们应该怎样应对一个有 100 万条记录的训练集？

以线性回归模型为例，每一次梯度下降迭代，我们都需要计算训练集的误差的平方和，如果我们的学习算法需要有 20 次迭代，这便已经是非常大的计算代价。

首先应该做的事是去检查一个这么大规模的训练集是否真的必要，也许我们只用 1000 个训练集也能获得较好的效果，我们可以绘制学习曲线来帮助判断。



## 16.2 随机梯度下降法 ( STOCHASTIC GRADIENT DESCENT )

如果我们一定需要一个大规模的训练集，我们可以尝试使用随机梯度下降法来代替批量梯度下降法。

在随机梯度下降法中，我们定义代价函数为一个单一训练实例的代价：

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

随机梯度下降算法为：

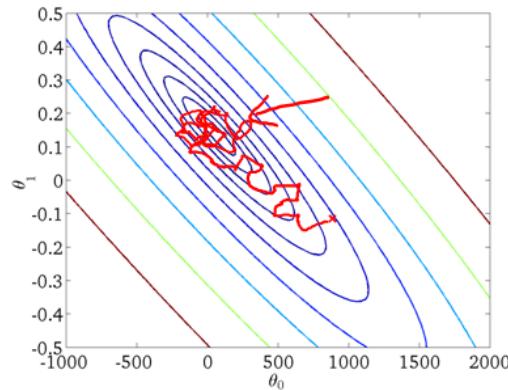
首先对训练集随机“洗牌”，然后：

```

Repeat (usually anywhere between 1-10) {
    for i=1:m {
         $\theta_j := \theta_j - \alpha(h_{\theta}(x^{(i)}) - y^{(i)})x_j^{(i)}$ 
        (for j=0:n)
    }
}

```

随机梯度下降算法在每一次计算之后便更新参数 $\Theta$ ，而不需要首先将所有的训练集求和，在梯度下降算法还没有完成一次迭代时，随机梯度下降算法便已经走出了很远。但是这样的算法存在的问题是，不是每一步都是朝着“正确”的方向迈出的。因此算法虽然会逐渐走向全局最小值的位置，但是可能无法站到那个最小值的那一点，而是在最小值点附近徘徊。



### 16.3 微型批量梯度下降 ( MINI-BATCH GRADIENT DESCENT )

微型批量梯度下降算法是介于批量梯度下降算法和随机梯度下降算法之间的算法，每计算常数 $b$ 次训练实例，便更新一次参数 $\Theta$ 。

```

Repeat {
    for i=1:m {
         $\theta_j := \theta_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} (h_{\theta}(x^{(k)}) - y^{(k)})x_j^{(k)}$ 
        (for j=0:n)

        i += 10;
    }
}

```

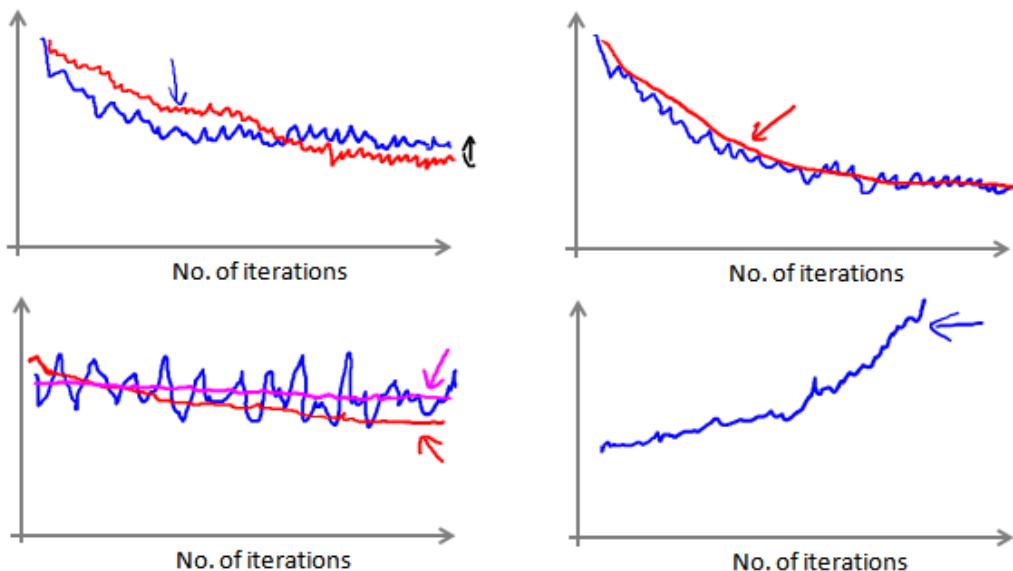
通常我们会令 $b$ 在2-100之间。这样做的好处在于，我们可以用向量化的方式来循环 $b$ 个训练实例，如果我们用的线性代数函数库比较好，能够支持平行处理，那么算法的总体表现将不受影响（与随机梯度下降相同）。

## 16.4 随机梯度下降收敛 ( STOCHASTIC DESCENT CONVERGENCE )

现在我们介绍随机梯度下降算法的调试，以及学习率 $\alpha$ 的选取。

在批量梯度下降中，我们可以令代价函数 $J$ 为迭代次数的函数，绘制图表，根据图表来判断梯度下降是否收敛。但是，在大规模的训练集的情况下，这是不现实的，因为计算代价太大了。

在随机梯度下降中，我们在每一次更新 $\Theta$ 之前都计算一次代价，然后每 $X$ 次迭代后，求出这 $X$ 次对训练实例计算代价的平均值，然后绘制这些平均值与 $X$ 次迭代的次数之间的函数图表。



当我们绘制这样的图表时，可能会得到一个颠簸不平但是不会明显减少的函数图像（如上面左下图中蓝线所示）。我们可以增加 $X$ 来使得函数更加平缓，也许便能看出下降的趋势了（如上面左下图中红线所示）；或者可能函数图表仍然是颠簸不平且不下降的（如洋红色线所示），那么我们的模型本身可能存在一些错误。

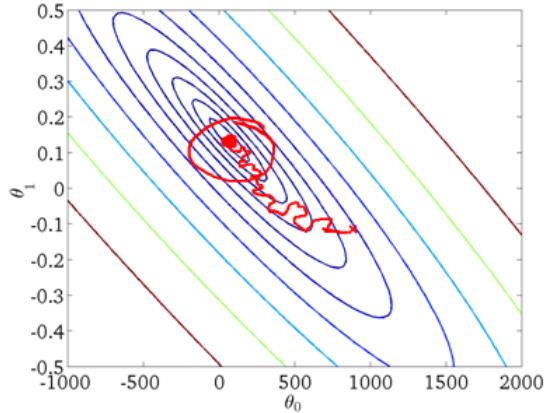
如果我们得到的曲线如上面右下方所示，不断地上升，那么我们可能会需要选择一个较小的学习率 $\alpha$ 。

我们也可以令学习率随着迭代次数的增加而减小，例如令：

$$\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$$

随着我们不断地靠近全局最小值，通过减小学习率，我们迫使算法收敛而非在最小值附近徘徊。

但是通常我们不需要这样做便能有非常好的效果了，对 $\alpha$ 进行调整所耗费的计算通常不值得



## 16.5 在线学习 ( ONLINE LEARNING )

在线学习算法指的是对数据流而非离线的静态数据集的学习。许多在线网站都有持续不断的用户流，对于每一个用户，网站希望能在不将数据存储到数据库中便顺利地进行算法学习。

假使我们正在经营一家物流公司，每当一个用户询问从地点 A 至地点 B 的快递费用时，我们给用户一个报价，该用户可能选择接受 ( $y=1$ ) 或不接受 ( $y=0$ )。

现在，我们希望构建一个模型，来预测用户接受报价使用我们的物流服务的可能性。因此报价是我们的一个特征，其他特征为距离，起始地点，目标地点以及特定的用户数据。模型的输出是  $p(y=1)$ 。

在线学习的算法与随机梯度下降算法有些类似，我们对单一的实例进行学习，而非对一个提前定义的训练集进行循环。

```

Repeat forever (as long as the website is running) {
    Get (x, y) corresponding to the current user
     $\theta_j := \theta_j - \alpha(h_\theta(x) - y)x_j$ 
    (for j=0:n)
}

```

一旦对一个数据的学习完成了，我们便可以丢弃该数据，不需要再存储它了。这种方式的好处在于，我们的算法可以很好的适应用户的倾向性，算法可以针对用户的当前行为不断地更新模型以适应该用户。

每次交互事件并不只产生一个数据集，例如，我们一次给用户提供 3 个物流选项，用户选择第 2 项，我们实际上可以获得 3 个新的训练实例，因而我们的算法可以一次从 3 个实例中学习并更新模型。

## 16.6 映射化简和数据并行 ( MAP REDUCE AND DATA PARALLELISM )

映射化简和数据并行对于大规模机器学习问题而言是非常重要的概念。之前提到，如果我们用批量梯度下降算法来求解大规模数据集的最优解，我们需要对整个训练集进行循环，计算偏导数和代价，再求和，计算代价非常大。如果我们能够将我们的数据集分配给多台计算机，让每一台计算机处理数据集的一个子集，然后我们将计所的结果汇总在求和。这样的方法叫做映射简化。

具体而言，如果任何学习算法能够表达为，对训练集的函数的求和，那么便能将这个任务分配给多台计算机（或者同一台计算机的不同 CPU 核心），以达到加速处理的目的。

例如，我们有 400 个训练实例，我们可以将批量梯度下降的求和任务分配给 4 台计算机进行处理：

Machine no	Samples	Computation	Result
1	1-100	$temp^{(1)} = \sum_{i=1}^{100} (h_\theta(x^i) - y^i)x_j^i$	
2	101-200	$temp^{(2)} = \sum_{i=101}^{200} (h_\theta(x^i) - y^i)x_j^i$	
3	201-300	$temp^{(3)} = \sum_{i=201}^{300} (h_\theta(x^i) - y^i)x_j^i$	
4	301-400	$temp^{(4)} = \sum_{i=301}^{400} (h_\theta(x^i) - y^i)x_j^i$	$\theta_j = \theta_j - \alpha \frac{1}{400} (temp^{(1)} + temp^{(2)} + temp^{(3)} + temp^{(4)})$

很多高级的线性代数函数库已经能够利用多核 CPU 的多个核心来平行地处理矩阵运算，这也是算法的向量化实现如此重要的缘故（比调用循环快）。

## WEEK10 应用示例:图像文字识别

### 17 应用实例：图像文字识别（PHOTO OCR）

#### 17.1 问题描述和流程图（PROBLEM DESCRIPTION AND PIPELINE）

图像文字识别应用所作的事是，从一张给定的图片中识别文字。这比从一份扫描文档中识别文字要复杂的多。



为了完成这样的工作，需要采取如下步骤：

1. 文字侦测 (Text detection) —— 将图片上的文字与其他环境对象分离开来
2. 字符切分 (Character segmentation) —— 将文字分割成一个个单一的字符
3. 字符分类 (Character classification) —— 确定每一个字符是什么

可以用任务流程图来表达这个问题，每一项任务可以由一个单独的小队来负责解决：



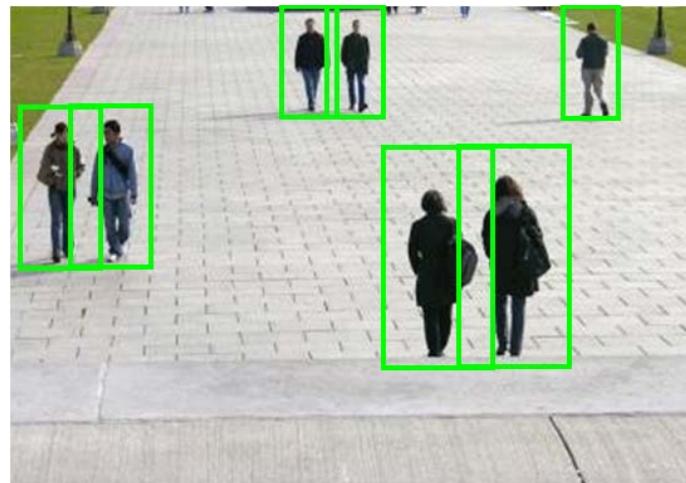
#### 17.2 滑动窗口（SLIDING WINDOWS）

滑动窗口是一项用来从图像中抽取对象的技术。

假使我们需要在一张图片中识别行人，首先要做的是用许多固定尺寸的图片来训练一个能够准确识别行人的模型。然后我们以之前训练识别行人的模型时所采用的图片尺寸在我们要进行行人识别的图片上进行剪裁，然后将剪裁得到的切片交给模型，让模型判断是否为行人，然后在

图片上滑动剪裁区域重新进行剪裁，将新剪裁的切片也交给模型进行判断，如此循环直至将图片全部检测完。

一旦完成后，我们按比例放大剪裁的区域，再以新的尺寸对图片进行剪裁，将新剪裁的切片按比例缩小至模型所采纳的尺寸，交给模型进行判断，如此循环。

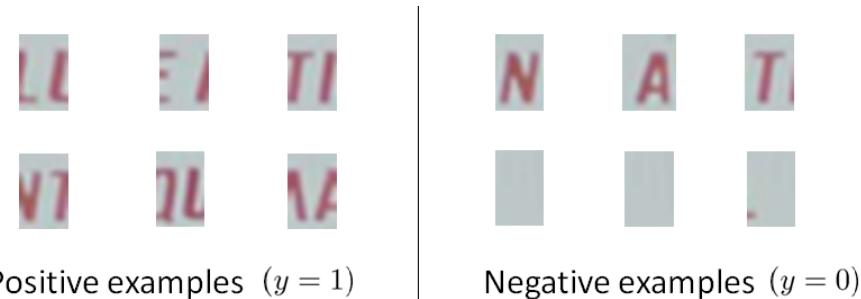


滑动窗口技术也被用于文字识别，首先训练模型能够区分字符与非字符，然后，运用滑动窗口技术识别字符，一旦完成了字符的识别，我们将识别得出的区域进行一些扩展，然后将重叠的区域进行合并。接着我们以宽高比作为过滤条件，过滤掉高度比宽度更大的区域（认为单词的长度通常比高度要大）。下图中绿色的区域是经过这些步骤后被认为是文字的区域，而红色的区域是被忽略的。



以上便是文字侦测阶段。

下一步是训练一个模型来完成将文字分割成一个个字符的任务，需要的训练集由单个字符的图片和两个相连字符之间的图片来训练模型。



模型训练完后，我们仍然是使用滑动窗口技术来进行字符识别。



以上便是字符切分阶段。

最后一个阶段是字符分类阶段，利用神经网络、支持向量机或者逻辑回归算法训练一个分类器即可。

### 17.3 获得大量数据个人工数据

如果我们的模型是低偏倚的，那么获得更多的数据用于训练模型，是能够有更好的效果的。问题在于，我们怎样获得数据，数据不总是可以直接获得的，我们有可能需要人工地创造一些数据。

以我们的文字识别应用为例，我们可以字体网站下载各种字体，然后利用这些不同的字体配上各种不同的随机背景图片创造出一些用于训练的实例，这让我们能够获得一个无限大的训练集。这是从零开始创造实例。

另一种方法是，利用已有的数据，然后对其进行修改，例如将已有的字符图片进行一些扭曲、旋转、模糊处理。只要我们认为实际数据有可能和经过这样处理后的数据类似，我们便可以用这样的方法来创造大量的数据。

有关获得更多数据的几种方法：

1. 人工数据合成
2. 手动收集、标记数据
3. 众包

#### 17.4 上限分析 ( CEILING ANALYSIS )

在机器学习的应用中，我们通常需要通过几个步骤才能进行最终的预测，我们如何能够知道哪一部分最值得我们花时间和精力去改善呢？这个问题可以通过上限分析来回答。

回到我们的文字识别应用中，我们的流程图如下：



流程图中每一部分的输出都是下一部分的输入，上限分析中，我们选取一部分，手工提供 100% 正确的输出结果，然后看应用的整体效果提升了多少。

假使我们的例子中总体效果为 72% 的正确率。

如果我们令文字侦测部分输出的结果 100% 正确，发现系统的总体效果从 72% 提高到了 89%。这意味着我们很可能会希望投入时间精力来提高我们的文字侦测部分。

接着我们手动选择数据，让字符切分输出的结果 100% 正确，发现系统的总体效果只提升了 1%，这意味着，我们的字符切分部分可能已经足够好了。

最后我们手工选择数据，让字符分类输出的结果 100% 正确，系统的总体效果又提升了 10%，这意味着我们可能也应该投入更多的时间和精力来提高应用的总体表现。

Component	Accuracy
Overall system	72%
Text detection	89%
Character segmentation	90%
Character recognition	100%

Handwritten annotations in blue ink show downward arrows and percentage values: 17% between Overall system and Text detection, 1% between Text detection and Character segmentation, and 10% between Character segmentation and Character recognition.

The END