机器学习

课前准备：

1. 监督学习(supervised learning)

例子1：预测房价，根据给定的数据（房价，面积）拟合出关于房价的函数，然后根据这个函数估计新的房价。这个属于回归问题（regression）中的一种，意味着我们要得到一个连续属性的结果(continuous valued output)。

例子2：预测肿瘤，给定的数据是肿瘤大小和是否为恶性（Y/N）, 然后给定一个大小判断恶性肿瘤的概率。这个属于分类问题（classification），得到的是一个离散值(discrete valued output), 0或1(根据问题也可以有多个值，0,1,2….)。这个例子中只有一个特征(feature)（大小），我们也可以通过ML学习分类有无穷多特征的数据。

监督学习小总结：对于给定数据中的每一个值都有一个对应的正确答案(所谓的“训练集”)，算法根据这些来给出预测。

1. 非监督学习(unsupervised learning)

非监督学习没有“训练集”，所给的数据没有“正确答案”，需要我们在所给的数据(data set)中找到某种结构(structure)。需要找到算法自动的将数据集进行分类，这个“聚类(clustering)”算法是非监督学习的一种。

例子：鸡尾酒会问题(cocktail problem) 算法可以将两个不同的音源分开（两个语言话或是人声和背景音乐）

推荐使用Octave进行编程。

鸡尾酒问题的代码中用到函数svd，意思是奇异值分解，是用于解线性方程的。

非监督学习的特点是没有标签数据(labeled data), 由数据自己的特点进行分类。

机器学习的定义：

旧定义：the field of study that gives computers the ability to learn without being explicitly programmed.

新定义：A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

Supervised learning:

1. Regression problem: continuous output
2. Classification problem: discrete output

Unsupervised learning:

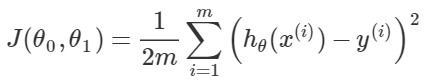
We can derive this structure by **clustering** the data based on relationships among the variables in the data

Preparation:

Linear regression with one variable:

We have: *hθ*(*x*)=*θ*0+*θ*1*x with x as input and y as output*

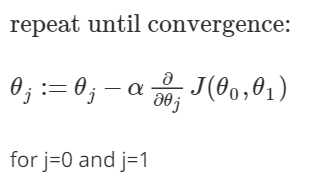
First we make our guess about the *θ*0 and *θ*1. Then we can measure our accuracy by “cost function”:



Also called as “squared error funcion” or “mean squared error”

Our goal is to minimize J

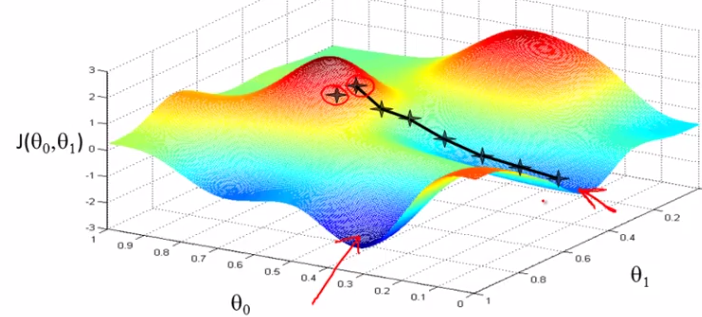
We use gradient descent(梯度下降法) to minimize J, we need to take derivative of each direction, in this case, *θ*0 and *θ*1



by reputation we can find *θ*0 and *θ*1 to minimize J

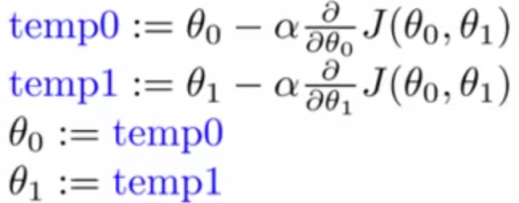
Gradient descent 的一个特点：

不同的出发点有可能得到不同的最优解



如图，如果我们选了右边那个起始点，那么最后会到达一个局部最低点，但不是整体的最优解。

还有一个小细节：



我们需要（simultaneous update）同步更新θ0 和 θ1，所以必须借助两个别的变量temp0和temp1

对于α 的选择，如果过小优化速度慢，如果过大可能会越过局部最低点，或有可能发散(divergent)

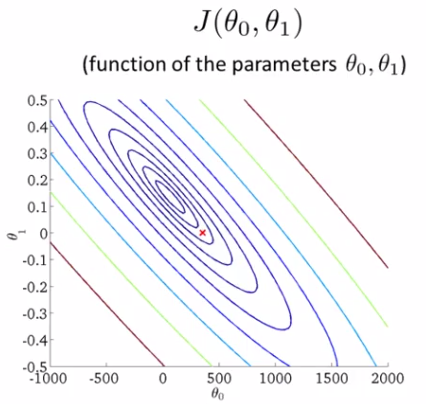
对于线性回归方程，J总是一个convex function（凸函数），没有局部最优解(local minumum)，只有全局最优解(global minimum)。

Batch gradient descent: each step of gradient descent uses all the training examples

也有别的别的种类，比如每次只使用training set 的子集

除了这个方法外，还有别的方法来求出J的最小值，比如正规方程法(normal equation), 但是数据比较多的情况下GD法比较好。

We can use Contour plot to present J

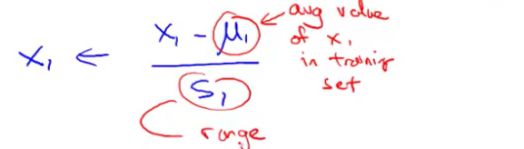


相同颜色的线条表示同一个cost function的值，最小值位于中间的圈那里，是一种表示J的平面方法。

多特征线性回归方程：

使用GD法的时候需要normalization，对于数值范围差的比较大的特征要进行scaling。

新的特征写成如下形式：



保证所有的特征在差不多的范围中，可以加快GD下降。

Normal equation

用这个方法只要一步就ok

 简直狂拽酷炫吊炸天

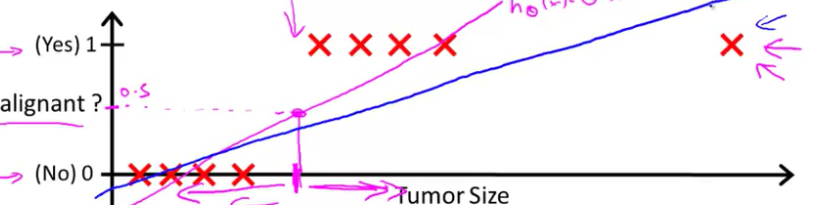
而且不用normalization，只有一个问题就是可能计算时间比较长。

注意training data中第一列要加ones(1,size(x)) 来得到cita0.

Classification problem

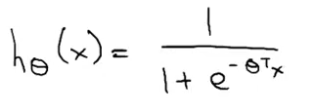
对于输出属于(0 or 1)问题

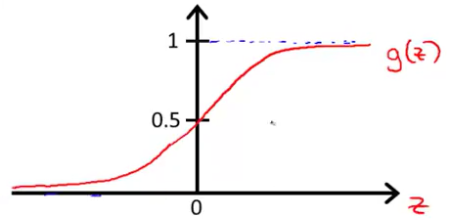
1. 一种处理方法，可以先用regression来拟合数据，然后定一个y轴的点作为阈值进行分类。但是会导致一个问题，如果数据较为离散，会对结果产生很大的影响：



我们使用logistic regression 来解决classification问题，这样同时可以保证输出的值在0到1之间

我们使用S形函数(sigmoid function) 或是逻辑函数(logistic function)来表示，如下(值永远在0到1之间)：





这个function的输出值将会被认为是y = 1的估计概率。

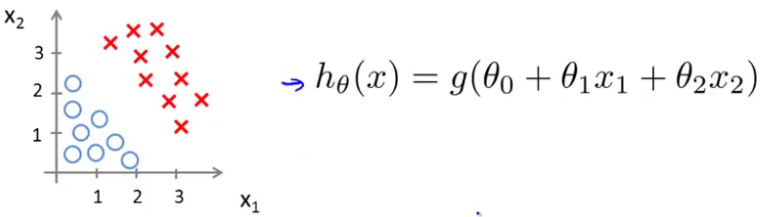


这个式子表示的是y = 0和y = 1是和cita有关的函数，两种情况概率加起来是1。

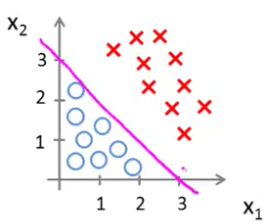
决策边界(decision boundary)

根据函数我们可以发现z大于0的时候g(z) 大于0.5。

举个栗子：

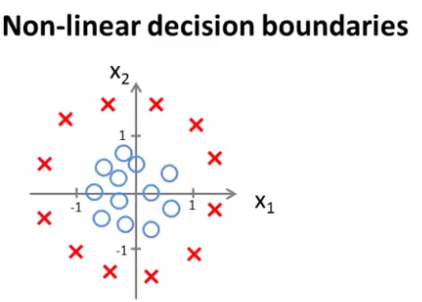


由上式可得，g(x) 中的x > 0 则有y = 1，所以我们可以得到x1 和 x2的线性规划区域：



这条区分线，就是决策边界。

再来一个难一点的栗子：

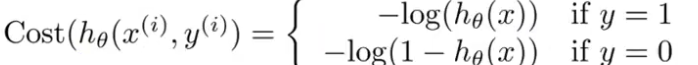


对于这种数据，我们可以用多项式来拟合，然后我们可以得到有多项式的决策边界，比如：



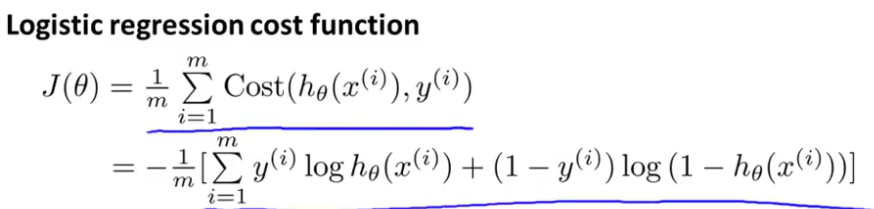
Cost function：

对于这个logistic regression 我们要用一个新的cost function来保证函数是凸函数 convex，并且没有局部最优解。

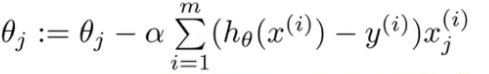


能保证和输出和y一样的时候cost function为0；输出相反的时候接近正无穷。

程序中我们可以写成一个形式，用y 和 (1-y) 来作为选择的条件：



然后和regression问题一样，我们用各个cita上的偏分来实现GD：



可以发现更新cita的式子和之前的regression一样的。

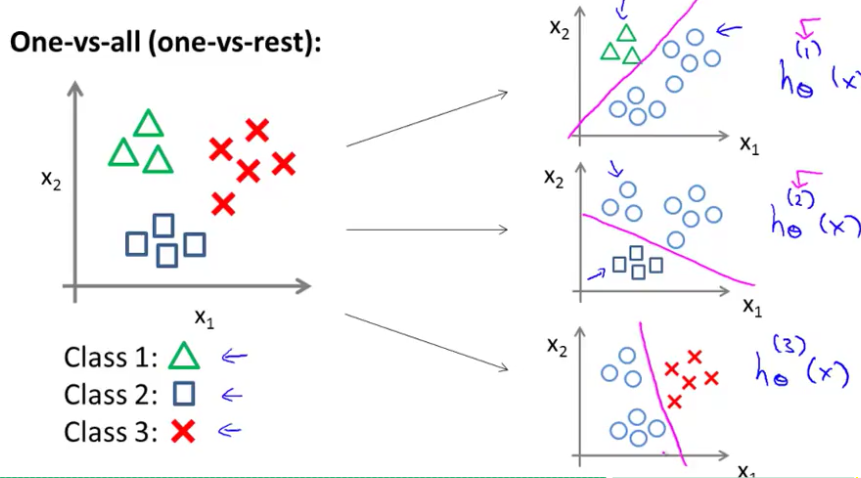
注意！logistic regression的cost function和linear regression不一样，不一样，不一样！！ 这里的h(x) 是那个sigmoid 函数的型式。怪不得老子做了6遍test还是没有满分。。。

优化进阶

先写一个程序return cost function和对应的gradient

然后用自带的fminunc函数来进行计算找到最佳的cita值。

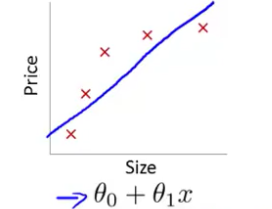
一对多分类算法（one-vs-all）多类别分类问题。

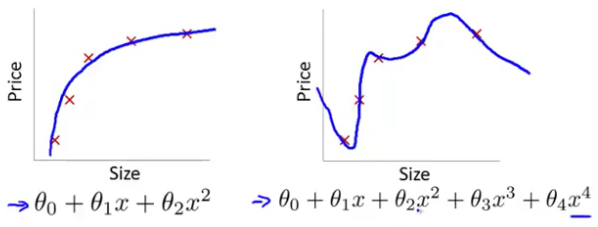


对这个例子来说，可以将问题转化为二元逻辑回归问题，分3种情况，每种情况中另外两种class归为一类进行分类。

Overfitting problem 过度拟合

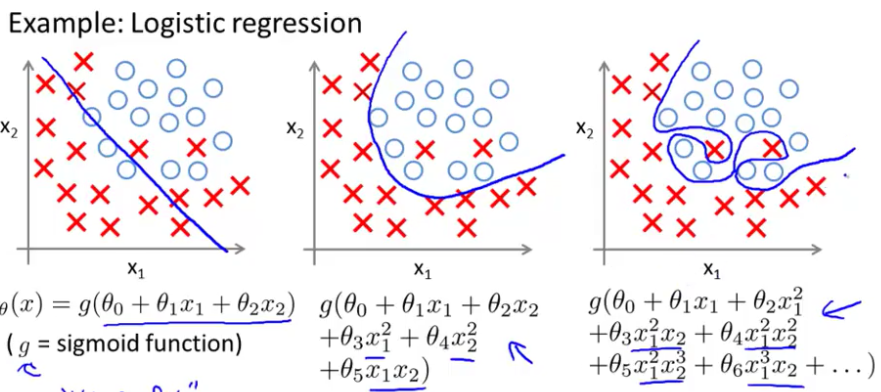
Underfitting 欠拟合导致 high bias 误差过大，如：





过度拟合则是最后一张图，虽然和training data很吻合，但是拥有过多的feature，不能正确的预测新的值，我们也称为 high variance

Logistic regression 中也会碰到这个问题



最后一张图就是过度拟合

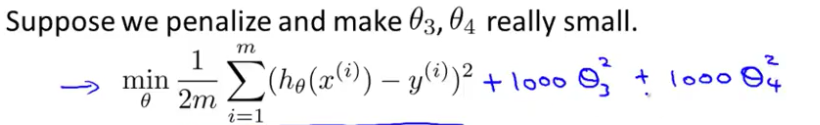
比较容易出现于feature较多而training data 较少的情况

Options to resolve：

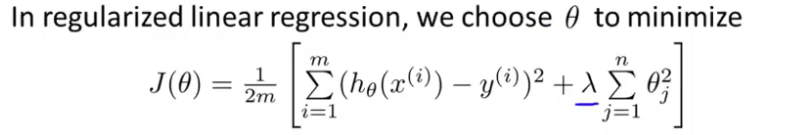
1.人为减少变量的数量或是用模型选择算法(model selection algorithm)

2. 正则化(regularization) 保留所有变量，但是选择不同的系数cita

正则化过程

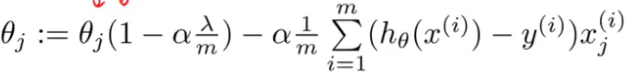


这样我们可以得到一个相对平滑的曲线，所以正则化的要点是在cost function中加上一项和系数有关的式子：



然后它的系数由一个lambda决定。注意cita0不参与到这个正则化中。所以在MATLAB的程序中，正则化的循环要从2开始。

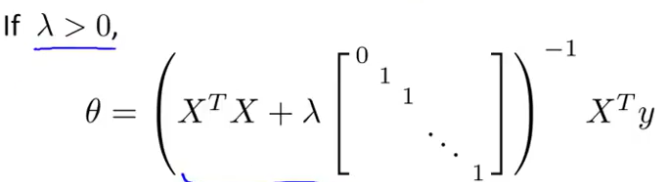
线性回归gradient的正则化：



加上一项后的偏分形式，注意j从1开始，因为cita0没有正则化的那项。

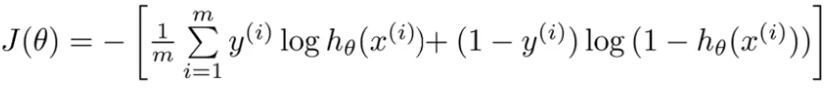
Normal equation 也可以加入正则化的部分。

 变为



在MATLAB中可以用pinv来进行计算，即使不可逆也可以得到一个伪逆矩阵。但是加上正则化的话则没有这个问题了，括号中一定是可逆的。

逻辑回归原cost function：



然后和之前的方法一样，在后面加上项，然后系数是lanmda / 2m, 如下：



Gradient：也要分为两部分，因为cita0没有正则化



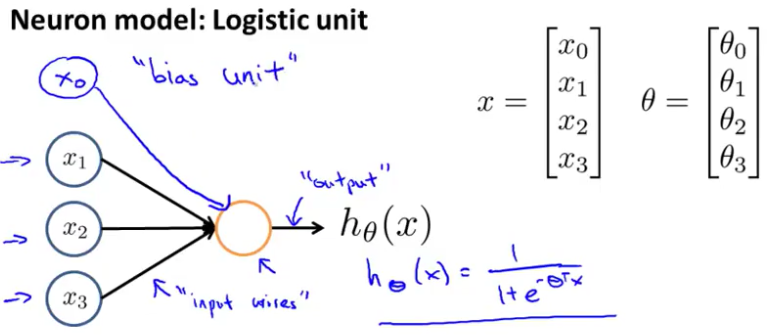


神经网络 neural networks

Non-linear classification

比较适用于当特征很大时（>10000000）(比如分辨照片)

One learning algorithm hypothesis: 模拟大脑用的单一学习算法

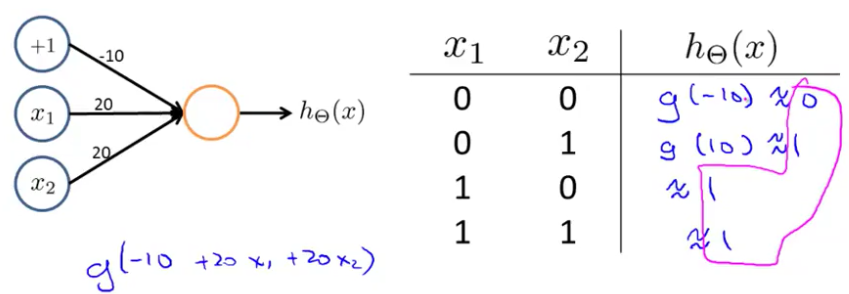


神经网络中术语Weight 就是 parameter

传递cita的维度是 S j+1 \* (Sj + 1)

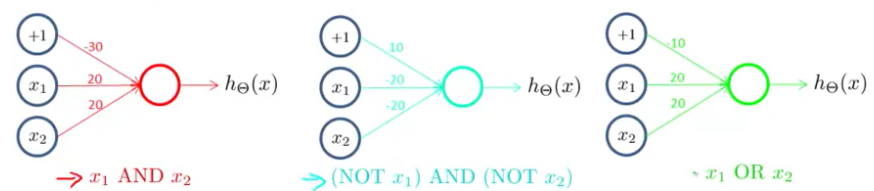
用向量化来简便计算。

用神经网络不同的weight，我们可以得到不同的逻辑结果：

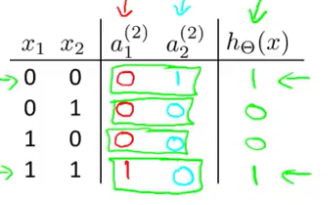


比如上图我们可以得到x1和x2是 or 的逻辑关系

几个逻辑关系的例子：



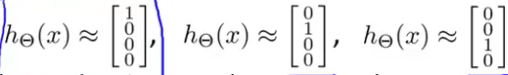
现在我们想达成一个 x1 XNOR x2的一个逻辑关系，x1 和 x2一样时输出1，不一样为0. 我们必须用两层逻辑组合才能达到这个目的：



a1 和x1 x2之间是and的关系；a2 和 x1 x2之间是非and非的关系，最后h(x) 和x1 x2之间是or的关系，x1 x2 和最后的h之间可得 XNOR关系。

多分类问题(multiclass calssification)

遇到有超过2个类别的问题我们可以希望结果输出如下结果：



来表示不同的类别，其中输出的h(x) 不一定恰好是1，因为用到了sigmoid函数，所以是约等于1。

同时我们要修改training data中的y，变为：



XOR 两者不同时为1 ，否则为0

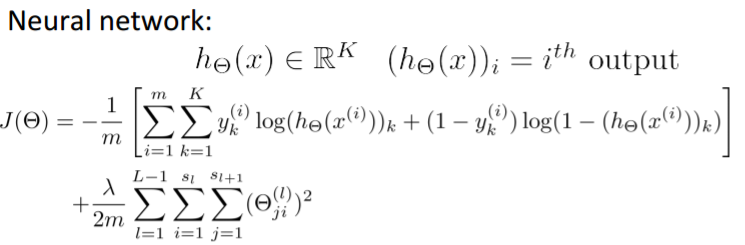
XAND 两者不同时为0，否则为1

NAND 两者都为1时输出0， 和AND反着的

Week5

神经网络的cost function

因为一般都处理oneversusall的问题，所以这里需要加上一个h(x)的k维度



K是output的种类，比如辨认0到9，那么k就等于10

想最小化J需要计算i和j两个方向上的Theta的gradient

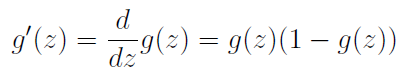
我们先用forward propagation从training data X算到神经网络最后一层，然后我们需要得到从第二层到最后一层的error，现在使用back propagation从后往前算

假设一共三层，首先：



注意这里的y要变成[1,0,0],[0,1,0]....这种格式





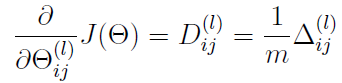
以此类推得到每层的error，注意error都是每层的unit数目K\*1

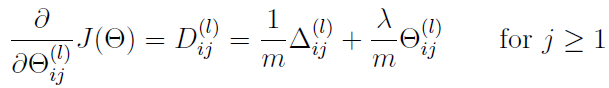
然后用公式：



注意每层的delta的维度和Theta样，比如这里就是26x1x1x401 = 26x401的维度

最后因为这个delta是m个x累加的，所以得到最后的gradient需要：



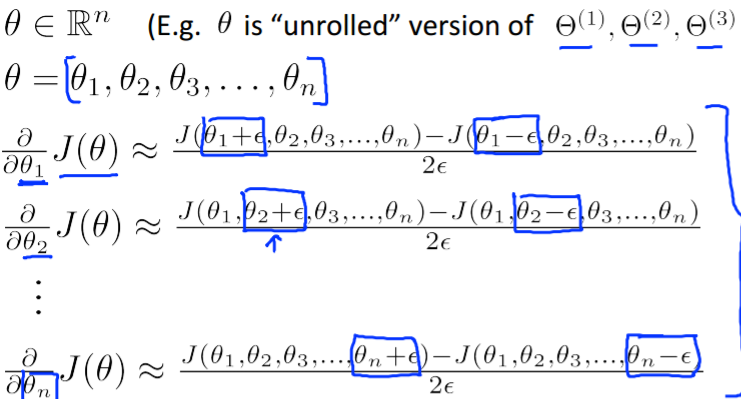


正则化的时候注意第一行的bias不用正则化

一些小技巧：

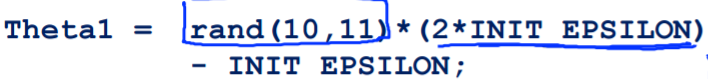
可以用一个点两边的极小值做斜线来获得近似的斜率。Gradient checking





不过这个是在使用数据训练之前，训练的时候要turn off这个checking

一开始Theta的初始值不能全部设为0，可以随机设为+-epsilon



一般步骤：

1. 获得随机的初始Theta

2. Forward propagation

3. 计算cost function J

4. Backprop计算gradient

5. 用gradient checking来保证gradient计算正确

6. 优化Theta，注意如果使用MATLAB中的fmincg等方法，要把Theta都展开成一个1xN的vector而不是数组

Week6

一个ml算法如果误差较大，一般两种原因，high bias or high variance

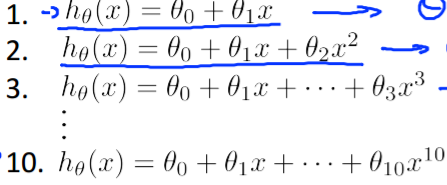
要学会如何测试并且解决问题

一般一开始将data分为三部分，60% training data, 20% cross validation data

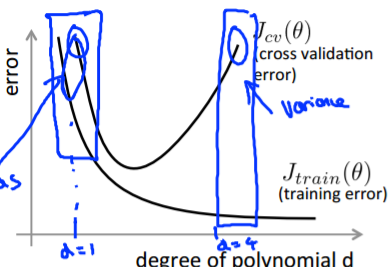
和20% test data.

用training data测试不同的特征和参数，用cv data来得出准确值进而选出合适的算法。 最后用test data来得到一个预测全新数据时的准确率。注意要再分出一个cv data是因为在用cv data算准确度并选择哪个算法的时候就已经将cv的数据fit in一次了，所以必须再用一组全新的数据才能得到准确度。

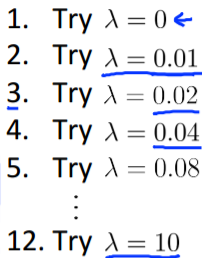
比如选择x的次方



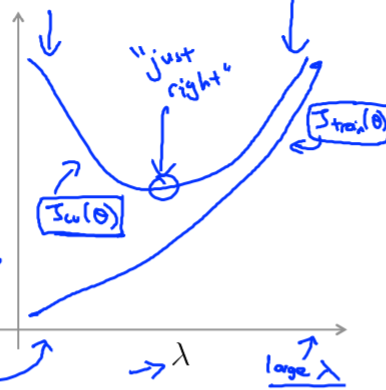
可以将这十个h(x)的cv的cost算出来然后选出小的那个就是最佳的model



可以看到一般trianing data的error是越来越小的，但是到后面就会有过拟合的问题了，overfitting, 所以cv的error有一个极小值后开始变大。

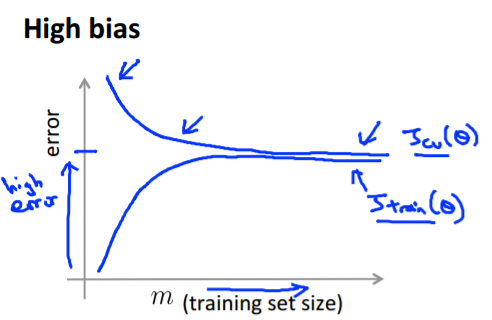


选出最佳的lambda和上面的方法差不多

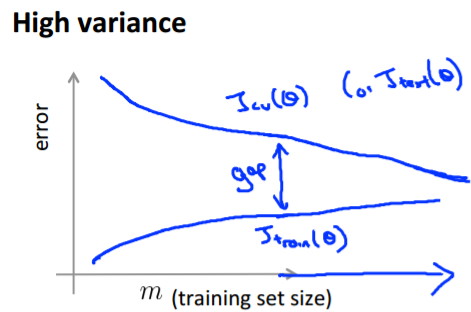


曲线不同，因为lambda = 0的时候是overfitting的所以training data的error是变大的，

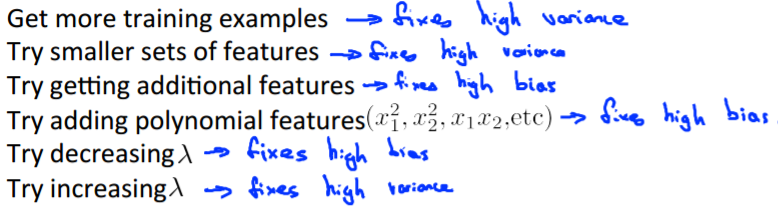
Cv的error同样有一个最小值



如果有High bias问题，那么随着训练数据的增加training和cv的error会趋于一致，但是都比较大，可以得出结论如果是high bias问题的话单纯增加训练数据不能解决问题



如果是high variance的话注意cv和training的error相对会有一个更大的差距，但是随着m的增大能慢慢的趋于一致，所以这个情况是可以增大m来提高accuracy的



一些常见的手段，注意最后两个不要搞混了

一些构建ml project的经验：

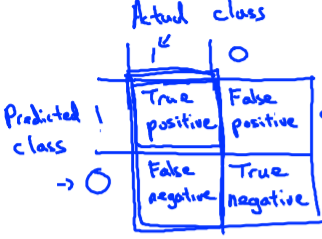
邮件分类系统，检查关键词来进行classification

一般步骤，先快速的设计一个相关的算法得到结果，然后根据结果和对应的learning curve来决定修改提高的部分。然后检查cv中出错的样例来决定要不要额外的增加算法。

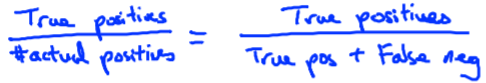
比如有的spam邮件故意拼错sale或discount等敏感词来躲避分类，这样就可以考虑加入一个单词过滤系统。

癌症检查系统例子(skewed class)，这里真正得癌症的人很少，所以就算是99%的准确率也不能说明是一个好的算法，所以要用另一种方式来测试准确率

使用precision和recall两个比率







Precision是预测是1中的准确率

Recall是现实中是1的中被预测的概率

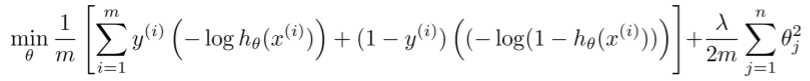
注意要把概率小的(这里就是得cancer)设为1，并且分子永远是true positive，所以如果永远返回0的话那么precision = 0，recall也是0

根据实际情况可以调整threshold，比如不是以0.5作为分界点，选择0.7可以增加precision降低recall，反过来0.3降低precision增加recall

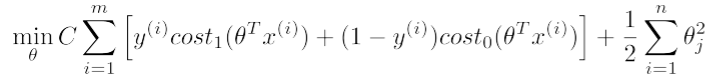
一般可以用F score = 2\*P\*R / (P + R) 来测试一个应用于skewed class算法的准确度。P和R分别是precision和recall

Week 7

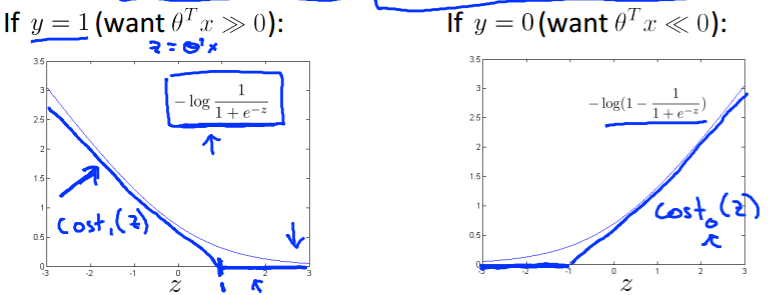
之前遇到的logistic regression:



支持状态机 support vector machin(SVM):

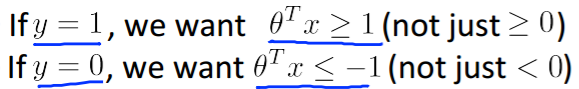


将原来function中的h(x)进行了替换，



然后去掉常数项m，C = 1/lambda

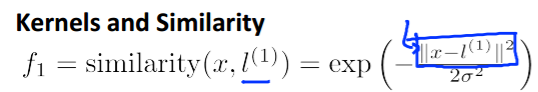
从上面的两张图可以看出来，使用SVM可以更有效的区别y = 0 and y = 1两种情况



所以可以得到一条更为理想的boundary，即分隔线会和两边的数据保持一定的margin，

当然同时我们要谨慎选择C来保证不要overfitting

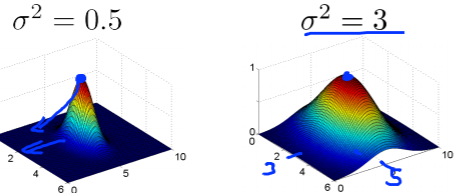
核函数，kernel：



用于比较x 和 l(landmark) 相似度的一个方程。如果两个很接近，那么f1会约等于1

如果差的比较大就会约等于0

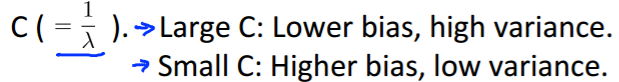
可以比较一下不同sigma的值对应的不同函数图像：

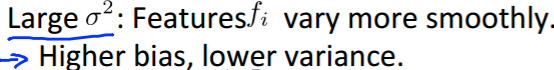


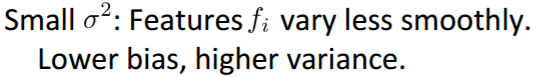
我们应用的时候把所有training data都选为landmark，所以一共是m个landmark

对于每个x 我们可以得出f1 f2....fm，和所有landmark的相似度，然后再用f(i)代替x在cost function中进行训练

注意两个重要参数的影响：







C比较好理解一点，sigma越大，f函数越平缓，会造成高偏差，而过小的话会造成overfitting

除了高斯kernel外还有别的kernel，但是不是那么常用，如果不选用高斯kernel那么就要linear kernel。

注意在使用SVM with Gaussian kernel的时候还是要注意feature scaling的问题，不要使得一个feature的数值范围过大，不然这个feature会得到过多的偏重。

Logistic 和 SVM的选择要看数据的特征数n和training data的数量m

如果n较大，m较小，或者n较小，m较大。那么比较适合logistic regression或者不要kernel的SVM，如果n比较小而m在10到10000这个区间，那么比较适合SVM wiht kernel。

Week 8

Unsupervised learning: Clustering

K-means算法

一开始先随机选择K个聚合点

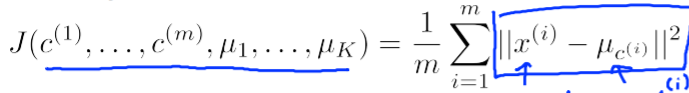
然后将所有m个点分配给最近的那个聚合点

然后根据每个分组再算出这个组中所有点的平均位置，也就是聚合点的新位置

然后在重复上上个步骤，再将点分配给新的聚合点

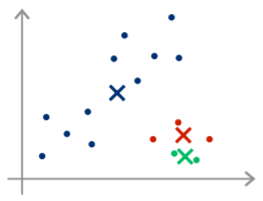
如果中途有聚合点没有分配到点，那么可以直接删去，最后结果是K - 1个聚合，也可以额外再初始化一个聚合点来弥补这个删去的聚合点

代价函数：distortion cost function



注意这个式子中的 是x(i) 分配到的那个聚合点的位置

注意K-means算法可能得到局部最优，比如：

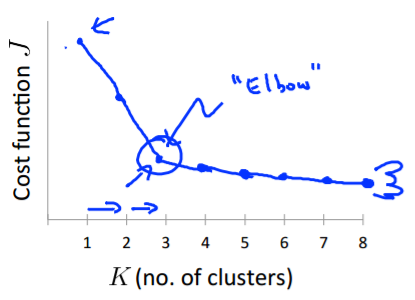


所以一开始选择聚合点的时候要注意random initialization

最好在1到m个数据点中选择尽可能比较远的K个初始点

在选择K的值时候，有一个肘部方法可以帮助选择：

画出J cost function 和 K的图，然后看有没有拐点，比如：



可以看到K = 3的时候有一个明显的拐点，说明3个聚类是一个比较好的选择

如果没有明显拐点的话就要结合具体目的来选择K了

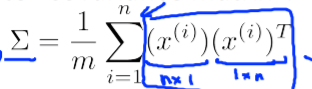
压缩维度

如果两个维度上的数据十分相关的话可以考虑把这两个维度上的数据压缩到一个维度上来1节省内存空间2加快学习算法的运行

一般使用PCA算法(principal component algorithm)

使用的时候我们要计算project error，做原来数据点到投射点的垂线来计算距离得出误差，这个和linear regression还是不同的，首先没有y，其次是做到projection的垂线

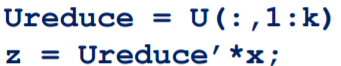
过程：

原始数据是n\*1的，首先得到一个n\*n的covariance matrix，

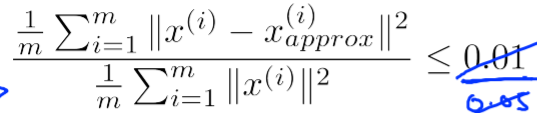
然后使用奇异值分解(singular value decomposition)



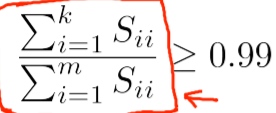
在其中U选出前K的vector组成想要的降维数组然后再乘原始n\*1的数据可得降维度后的数据



然后想得到原来的数据只要再乘降维数组的逆即可

选择降维的K的时候一般从保留百分之多少的差异开始，

如果是0.01的话表示保留了99%以上的特征，这个数值还有一个更方便的算法：



使用svd分解中的S矩阵即可以快速的得出两个维度数据的相似度，然后可以从K = 1开始迭代，知道相似度达到要求，一般90%到99%是比较普遍的选择

使用PCA建议

一开始需要得到原始X和压缩Z之间的映射关系，也就是压缩矩阵U，这个U从training data中得到，然后直接应用到cv和test数据上。

不要使用PCA来处理overfitting，虽然可能会有效果，但是有可能会丢弃有用的数据，而且一般使用正则化就是很好的方法了。

不要一开始就想着使用PCA，一般是发现feature过大，原始数据处理太慢才要使用PCA