

Machine Learning

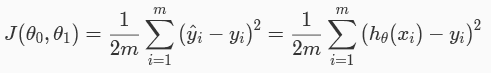
Coursera

2018.11

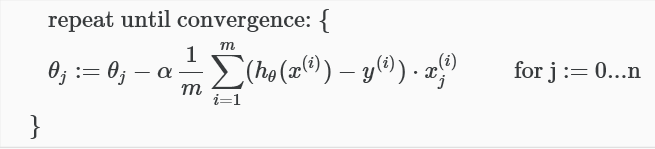
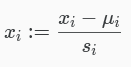
Introduction

1. 监督学习（supervised learning）：给出一堆已有数据和答案，让算法预测一组新数据的答案（一次决定）
2. 非监督学习（unsupervised learning）：只有一堆数据（找规律）；聚类
3. 强化学习（reinforcement learning）：奖惩机制；根据每次的结果自动进行优化（多次决定）

Linear Regression with One Variable

1. 代价函数（cost function）:
2. 梯度下降要注意“同时更新（simultaneously update）”的问题

Multivariate Linear Regression

1. 更新参数：
2. 特征缩放(feature scaling)：当各个特征的取值范围相差较大时，梯度下降算法收敛到最小值的速度很慢，可以通过特征缩放来加快梯度下降算法收敛的速度。一般应使特征取值范围为[-1,1]，但并非严格要求，相近即可。
3. 均值归一化（mean normalization）：
4. 

其中μi表示特征取值均值，si表示特征取值的极值或标准差

1. 通过作出 “J(θ)-迭代次数” 函数图像可以判断梯度下降算法是否正常工作。如果学习速度α选择得合适，每次迭代后J(θ)都会减小，如果J(θ)不随迭代次数单调递减，则应该适当缩小学习速度α。α也不宜过小，否则梯度下降算法的收敛速度会很慢。
2. 将已有的特征进行适当的组合得到新的特征有时可以达到很好的效果。
3. 正规方程：**θ = (XTX)-1XTY**

**X\*θ=Y（其中X表示特征取值矩阵，注意要加一项x0=1；Y表示相应特征取值的已知结果）。利用正规方程可以直接计算出θ0~θn的值。利用正规方程求解θ不必进行特征缩放。但是当n的值很大，即特征很多时，求解正规方程可能会很慢，此时可以考虑使用梯度下降方法。**

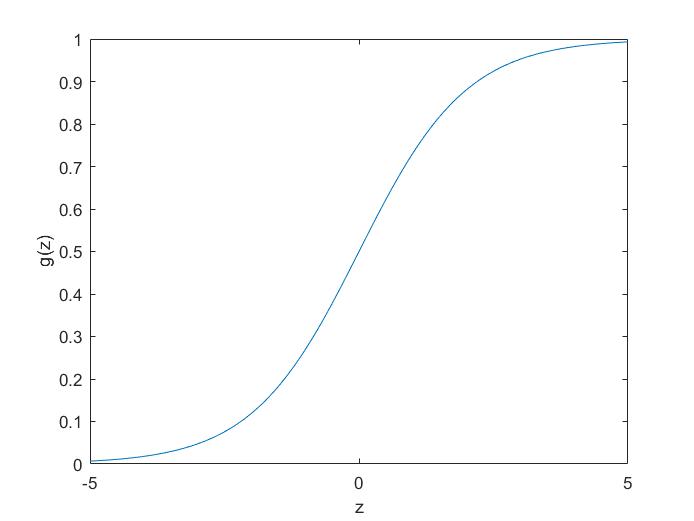
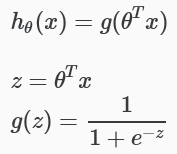
**8.**若X不可逆，有两种可能：采用的特征中有两个及以上特征具有相关性；特征太多而样本太少（如n=100, m=10）。解决方法分别是删除具有相关性的特征和删除一些特征以减少特征数量。

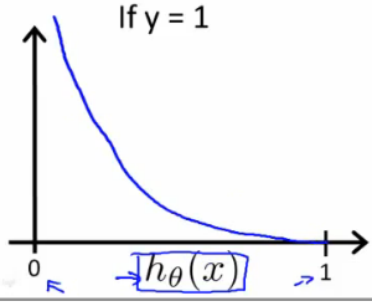
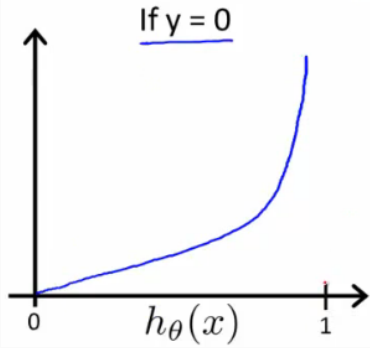
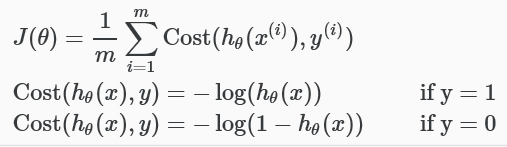
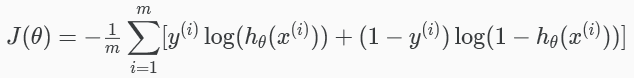
Octave/Matlab Tutorials

1. 指定小数位数：disp(sprintf(‘%0.2f’,a)； 使用format long 和 format short 命令可以调整默认显示小数位数
2. 等差行向量：v = 起始值:步长:结束值 如：v = 1:0.1:2
3. ones(m,n)：生成一个m行n列的矩阵，其中的元素全为1；类似地，zeros(m,n)可以生成一个m行n列的零矩阵；rand(m,n)可以随机生成一个m行n列的矩阵，元素数值均在0~1；randn(m,n)可以按照正态分布生成一个m 行n列的矩阵，其元素取值的服从正态分布，平均值为0
4. hist(m,n)：绘制柱状图，其中m为数据。例如一个一行a列的行向量，n为柱子的个数
5. eye(n)：生成一个n阶单位矩阵
6. size(A)：返回矩阵A的行数列数，返回的类型是一个一行两列的矩阵；size(A,1)和size(A,2)可以分别返回A矩阵的行数和列数；length(A)可以返回A矩阵的最大维数，即行数和列数中较大的那一个，但通常只对向量使用此函数，而不对一般矩阵使用
7. 使用cd “xx:\xxx\xxx”命令切换到指定文件夹后，可以用load example.dat命令或load(‘example.dat’)命令读取数据文件；who命令可以显示当前存储的全部变量；whos命令则不仅可以列出当前存储的全部变量，还可显示其详细信息；使用clear example命令可以删除example变量；使用save example.mat v命令可以把一个v变量存为一个名叫example.mat的文件，此命令会将数据以二进制格式存储，若想将数据存为普通的形式，可以使用命令save example.txt v –ascii
8. A(m,n)表示A矩阵第m行第n列的元素，A(m,:)表示A矩阵第m行的全部元素，A（:,n）表示A矩阵第n列的全部元素；A([m n],:)表示A矩阵的第m行和第n行的全部元素；A = [A,[a;b;c]]可以在A矩阵的右边增加一列[a;b;c]；A(:)可以将Ａ中的所有元素放入一个列向量；C = [A B]和C = [A;B]可以将A矩阵和B矩阵横向和竖向放在一起从而构成一个新矩阵
9. A .\* B表示A中的每一个元素与B矩阵对应位置上的元素相乘。点号通常表示位运算，所以类似地还有 A .^ 2 1 ./ A等操作
10. A’表示矩阵A的转置矩阵；magic(n) 幻方矩阵；find(a>b) 寻找指定条件元素；sum(A) 返回矢量各元素之和或矩阵每一列各元素之和；prod(A)返回矢量各元素之积或矩阵每一列各元素之积；floor(A)将A中的每一个元素缩小到小于等于该元素的最接近整数；ceil(A)将A中的每一个元素扩大到大于等于该元素的最接近整数；max(A,[],1)返回矩阵每一列的最大值，max(A,[],2)返回矩阵每一行的最大值；flipud(A)将A矩阵沿水平对称轴上下翻转
11. pinv(A)和inv(A)：求矩阵A的逆矩阵
12. plot()函数用来画二维图，有很多可选参数，详细信息可使用help plot查询；hold on命令可以保留当前已经画出的图形不被新的图形所覆盖；legend()函数用于在函数图像上添加图例；使用print –dpng ‘example.png’可以保存生成的图像，也可保存为其他格式的图片文件，详见help plot；subplot()函数可以将一张图片分为不同的格子，在每个格子里分别绘制不同的函数图像，便于相互对比；axis([a b c d])函数可以指定横坐标轴和纵坐标轴的范围（a,b表示横轴范围，c,d表示纵轴范围），还有其他参数可选
13. imagesc(A)可将A矩阵的数据以彩色方格图的形式呈现出来，颜色深浅代表值的大小，可以配合colorbar、colormap等命令使用

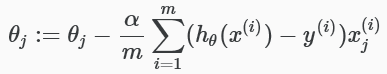
Logistic Regression

1. 逻辑函数（logistic function/sigmoid function）：

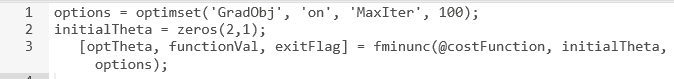
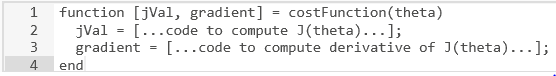


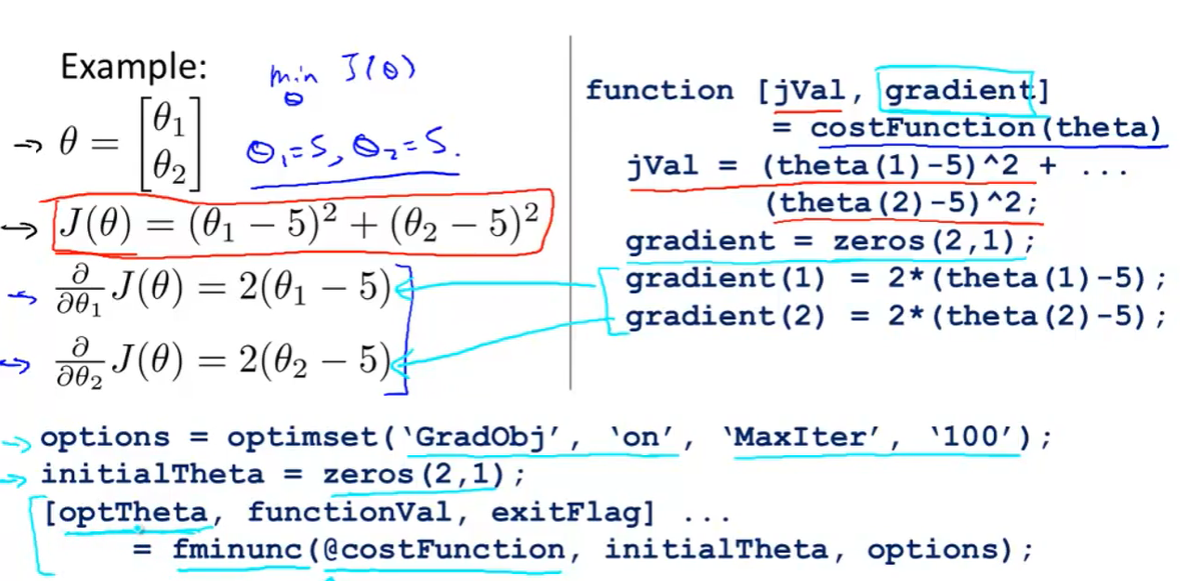
1. 逻辑函数h实际上表示的是y=1的概率，即h(x)=P(y=1|x,θ)，所以我们约定：当h>=0.5时，取y=1；当h<0.5时，取y=0。由h的定义和逻辑函数图像的特征可知，此约定等价于当 θTX>=0 时，取y=1；当 θTX<0 时，取y=0。θTX = 0的曲线或曲面称为决策边界（decision boundaries），它将平面或空间分为两部分，在这两部分中y分别取0和1。
2. 代价函数（cost function）：
3. 为了便于表述和编程，代价函数（cost function）也可写作一行，即：

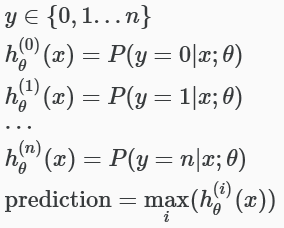
（实际应用时要注意向量化）

1. 与之对应地，θ的更新规则为：

值得注意的是，这个θ的更新规则形式上与线性回归的θ更新规则完全相同，但是其实其中的h(θ)定义并不相同。在实际应用此式时同样要注意向量化（vectorized）。

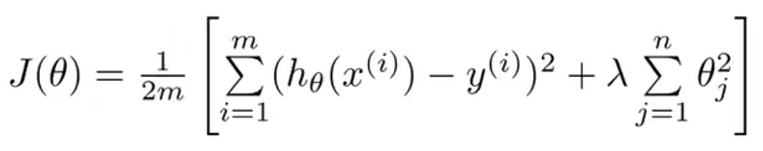
1. 与线性回归一样，在逻辑回归（logistic regression）中，特征缩放（feature scaling）同样可以加快梯度下降（gradient descent）的速度，当各个特征取值差异较大时，应使用特征缩放。
2. 当数据量很大时，梯度下降算法的速度往往不能令人满意，此时可以使用matlab内置的高级算法，如下

（需要注意的是fminuc函数要求initialTheta是一个大于等于二维的向量）。看一个例子：

1. 多类分类问题：当y的取值不止0和1，而是有n个，即我们想把数据分为n类时，可以分别对每一类应用逻辑函数，把那一类看作1，而其他所有类看作0。这样我们就可以用n个分类器（classifier）把数据分为n类，同时也得到了n个h(θ)。对于一组新的数据，判断它属于哪一类时就看它使哪一个h(θ)取得最大值。即：

Regularization

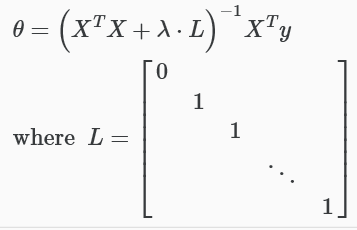
1. 当我们用一个比较简单的函数拟合训练数据时，往往很难得到较好的拟合结果，此时称之为欠拟合；而当我们用一个非常复杂的多项式拟合训练数据时，可能会得到一条通过所有数据点的曲线，但该曲线却不能对一组新的数据做出很好的预测，此时称之为过拟合。解决过拟合问题主要有两种方法：减少特征的数量或者进行正则化（regularization）。
2. 正则化的思路是减小高次项前的参数θj，从而使高次项对预测函数的影响大大减小，从而是预测函数趋向简单化而大大减轻过拟合的倾向。具体做法是，改变代价函数如下：



值得注意的是，Σ后θ的取值是从θ1开始的，并不包括θ0，这是一个惯例。此外，λ的合适取值对于正则化的效果十分重要。

1. 正则化方法既可用于线性回归也可用于逻辑回归。用于线性回归问题时，代价函数的表达式如上一条所示，对应的θ更新规则如下：

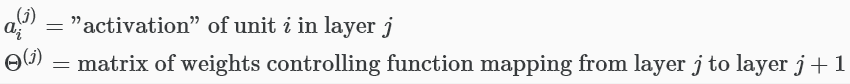
可以注意到，θ0的更新规则与其他θ不同，这是因为正则化的时候没有包括θ0。θj的更新方法也可写成下面这种形式： j∈{1,2,…,n}

同时，对于正规方程法，使用了正则化方法后，正规方程相应地变为下面这种形式：其中X为m×(n+1)的矩阵，L为(n+1)×(n+1)的矩阵。可以证明，加上λL这一项后，也解决了X可能不可逆的问题。

1. 对于逻辑回归问题，同样可以使用正则化方法以避免过拟合问题，相应的代价函数和θ的更新规则如下：

θj的更新规则看起来与线性回归一模一样，但是，如前所述，因为h(θ)的定义不同，所以他们的意义并不完全相同。而且，对代价函数和θ的更新规则做相应的改变以后，同样可以使用前面所讲的高级优化方法以提高速度。

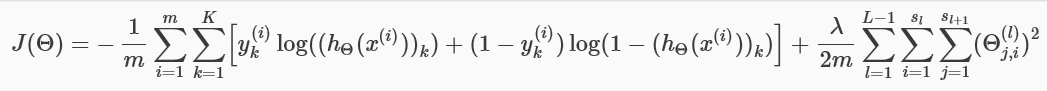
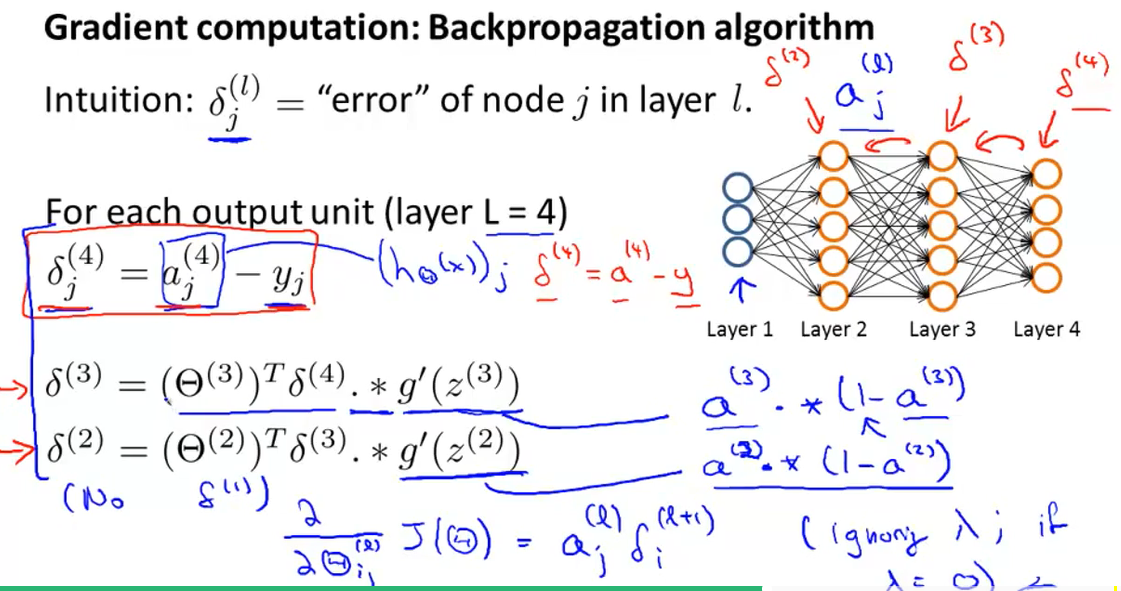
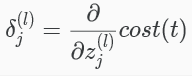
Neural Networks：Representation

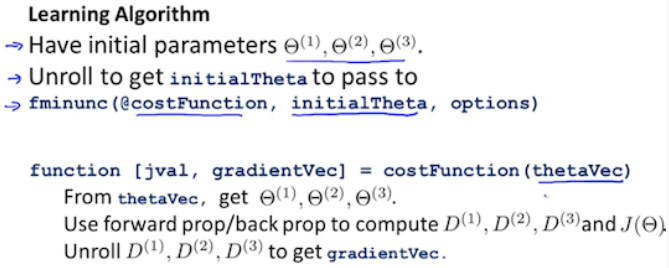
1. 

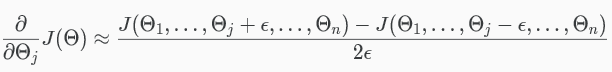
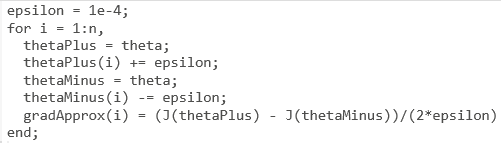
如果网络第j层有sj个单元，第j+1层有sj+1个单元，则Θ（j）的维度为sj+1×(sj+1)。此处的“+1”是由于有一个始终为1的x0(bias nodes)。

1. 定义，即zk(j)为第j层第k个单元的输入项（注意该式已经向量化），则。由此可以计算出第j层的值，添加一项偏置单元(bias nodes)后重复上述过程即可求得下一层的值，直到计算出最后的输出值。
2. 多元分类（Multi-class classification）问题，若共有N类，则输出层有N个神经元（units），分别代表N类。sigmoid函数计算出的是取相应值的概率，所以对于每一个example，它就应当属于输出层中的最大值所表示的类。

Neural Network：Learning

1. 神经网络的代价函数（cost function）为：
2. 反向传播算法（Backpropagation Algorithm）：为了最小化J(θ)，我们需要算出。由于J(θ)的形式很复杂，无法对其直接求导，所以我们用反向传播算法求导，其过程如下：用语言描述如下：对于给定的数据集，首先初始化Δ为0。对于数据集中的每一个example，使用前向传播(Forwardpropagation)算法将x代入输入层求得相应的输出层，即h(θ)，用上面第一张PPT上的表达式算出输出层的δ，即h(θ)与实际值之间的误差。然后利用输出层的δ反向计算出每一隐藏层(hidden layer)每一神经元的δ，即通过前向传播算法算出的值与期望值之间的误差，谓之反向传播。值得注意的是，反向传播最后只计算到第二层的δ，因为第一层的数据是数据集给定的，认为没有误差。然后更新Δ为Δ+aδ，即加上相应神经元前向传播算出的值与反向传播算出的误差之积。最后用算得的 Δ表示D，注意D的表示要分为两部分，因为正则化没有包含偏置单元（bias units）。可以证明，D就是所求的，即梯度下降中的梯度。然后便可通过梯度下降方法或其他高级优化方法算出使J(θ)最小的θ值。
3. 事实上，在反向传播算法中，每一个神经元的δ就是该神经元的代价函数对相应的z的偏导数，即
4. 反向传播算法的计算其实与前向传播算法相同，都是利用已经计算出的层的数据乘上相应的权重（weights，即Θ）从而得到下一层的数据。例如，对于一个四层神经网络，有：。
5. 一些高级优化算法（如fminunc()）常常要求参数为向量形式，而在神经网络中的Θ等参数一般为矩阵形式，所以如果要使用高级优化算法，需要进行向量化，具体方法如下：。这两行代码会Theta和D矩阵中的每一个元素取出，排列成一个很长的列向量，从而可以用于高级优化算法。而从该列向量中重新获得参数矩阵的方法如下（假设Theta1、Theta2、Theta3的维度分别为10×11，10×11，1×11）：

所以，利用高级优化算法计算代价函数（cost function）和梯度(gradient)的过程如下：

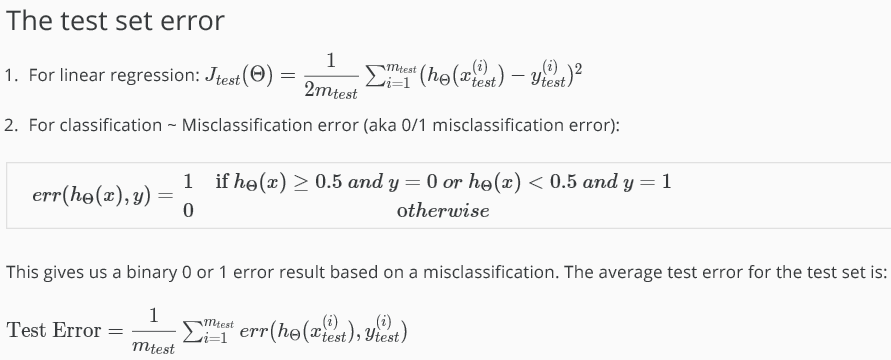
1. 虽然反向传播算法非常高效，但是由于它非常复杂，所以可能会出现一些奇怪的bug，为了验证所写的反向传播算法是否正确，可以采用梯度检验(gradient check)，其理论基础如下：可以注意到，其实就是导数的定义。ε通常取一个很小的值，但是也不宜过小，否则会出现数值计算上的问题，通常取10-4。具体实现过程如下：如果通过梯度检验算出的梯度与通过反向传播算法算出的梯度（即D）基本相等，则说明所写的反向传播算法是正确的。另一点值得注意的是，一旦确认所写的反向传播算法是正确的，就不应该再使用梯度检验，而是直接使用反向传播算法和给定的数据集训练模型，因为梯度检验的速度很慢，会影响训练效率。
2. 随机初始化：在神经网络中，将所有θ全部初始化为0是行不通的，因为这将导致隐藏层的神经元计算出相同的值，而且由此计算出的误差δ也是相同的，这种现象成为对称现象。打破对称的方法是把Θ矩阵初始化为一定范围内随机的值，具体做法如下（假设Theta的维度为10×11）：

需要注意，此处的epsilon与梯度检验中的epsilon没有任何关系。

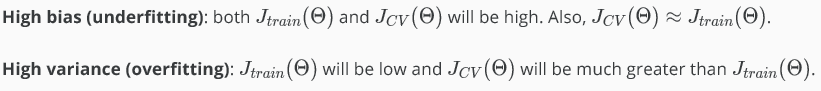
1. 通常只用一个隐藏层，如果使用多个隐藏层，一般应使每个隐藏层的神经元数目相同。隐藏层的神经元数目越多，得到的结果一般会越好，但是计算量也会大幅增加，应该综合考虑从而确定合适的神经元数目。
2. 训练一个神经网络一般分为以下六步：
3. 随机初始化权重（即Θ）；
4. 使用前向传播算法得到所有x对应的h(θ)；
5. 计算代价函数J(θ)；
6. 使用反向传播算法计算代价函数对θ的偏导数；
7. 使用梯度检验确认反向传播算法正确工作，然后关闭梯度检验；
8. 使用梯度下降或其他高级优化算法最小化代价函数，得到Θ。

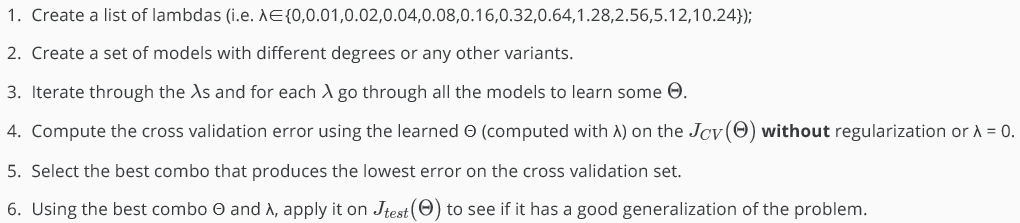
通常用for循环来对每一个example使用前向传播和反向传播算法以求得a、δ和Δ。另外，由于神经网络的代价函数不是一个凸函数，所以有一定概率最后会收敛到一个局部最小值而非全局最小值（但是AndrewNg说通常情况下结果还是不错的。。。）。

Advice for Applying Machine Learning

1. 为了评估训练出的模型的性能，通常我们将已有的数据集分为两部分：训练集（training set，70%）和测试集（test set，30%）。定义测试集的误差如下：测试集的误差大小一定程度上就可以反映模型的性能。
2. 模型选择：在进行模型选择时，通常将已有数据分为三部分：训练集（training set，60%），验证集（cross validation set，20%），测试集（test set，20%）。用训练集数据将各模型训练好后（线性回归等方式），用验证集数据求得各个模型的误差，选择误差最小的模型。测试集数据用于评估选择的模型的泛化误差（generalization error），即模型对新数据的误差情况。
3. 当模型的效果不理想时，通过比较训练集误差和验证集误差的大小，我们可以判断模型是欠拟合（即高偏差）还是过拟合（即高方差）：

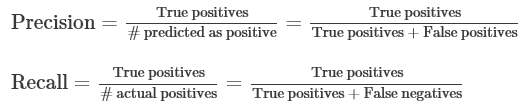
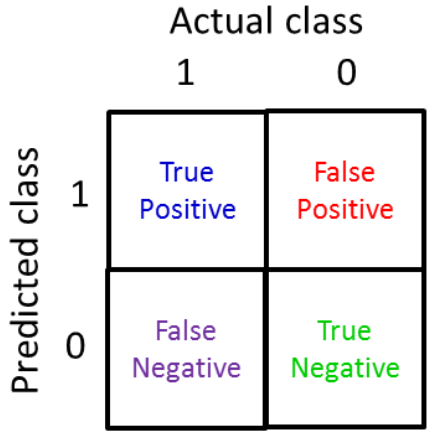
高偏差（high bias（underfitting））:训练集误差和验证集误差都很大；

高方差（high variance（overfitting））：训练集误差很小，验证集误差远大于训练集误差

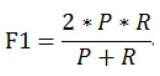
1. 前面已经说到，为了解决过拟合问题，我们使用了正则化（regularization），但是正则化方法中λ的取值直接关系到最终的效果，求取合适的λ值的步骤如下：从小到大取一系列的λ值，依次带入代价函数（cost function）从而得到不同的θ，即不同的模型，用验证集数据分别计算出各个模型的验证集误差（验证集误差公式不需要正则化），使验证集误差最小的λ值就是最合适的λ。（没错，就是列一大堆λ然后一个一个试( ╯□╰ ) 。把若干个要试的λ列成一个向量然后用一个for循环计算出每一个λ对应的验证集误差，作出验证集误差对λ的图像即可比较直观地找到合适的λ。）
2. 学习曲线，即训练集误差Jtrain和验证集误差Jcv，可以帮助我们判断模型处于高偏差还是高方差。当模型处于高偏差时：若训练集数据少，则训练集误差很小而验证集误差很大；当训练集数据逐渐增多时，训练集误差和验证集误差会逐渐接近并且最终都比较大；对于高偏差的模型，增加训练数据对于提高模型性能没有什么帮助。当模型处于高方差时：若训练集数据少，则训练集误差很小而验证集误差很大；当训练集数据逐渐增多时，训练集误差会逐渐变大而验证集误差会逐渐变小，但训练集误差始终小于验证集误差而且它们之间的差值始终很大；对于高方差模型，增加训练数据对于提高模型的性能有一定的帮助。
3. 小结：采用更多训练样本有助于解决高方差问题；减少特征数量有助于解决高方差问题；增加特征有助于解决高偏差问题；增加多项式特征有助于解决高偏差问题；减小λ有助于解决高偏差问题；增大λ有助于解决高方差问题。

Machine Learning System Design

1. AndrewNg说当面对几种可能的方法不知道选哪种时，最好对不同的方法都写个简单的模型大致看一下效果，而不是随便选一种然后一条路走到黑。对于模型的性能评价要有一个数值化的标准。
2. 偏斜类（skewed classes）是指在分类时，两类的占比相差很大，比如1：0 = 99.5：0.5。在这样的情况下，即使我们直接无脑预测y = 1，也能获得99.5%的准确率，显然，对于偏斜类，把预测准确率作为评估模型性能的标准就不太合适。因此，定义了两个新的指标来评估模型处理偏斜类问题时的性能：精确率（Precision）和召回率（recall）。它们的定义如下：

 从定义可以看出，精确率和召回率都是越高越好。值得注意的是，在偏斜类中，我们 通常将占比较小的那一类定为1。

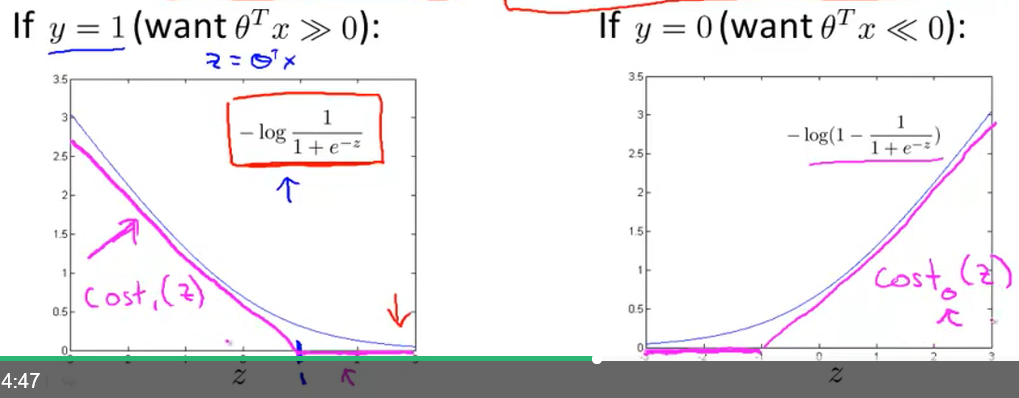
1. 在逻辑回归问题中，通过调整预测0和1的临界值（一般设为0.5），可以获得不同的精确率和召回率，到底需要高精确率还是高召回率要根据具体的情况而定。为了评估两个有不同精确率和召回率的模型的性能，定义一个新的指标：F值（F score），或称F1值（F1 score）。其定义如下：

F值越大，就意味着精确率和召回率较均衡，通常就认为是一个较好的模型。

Support Vector Machines

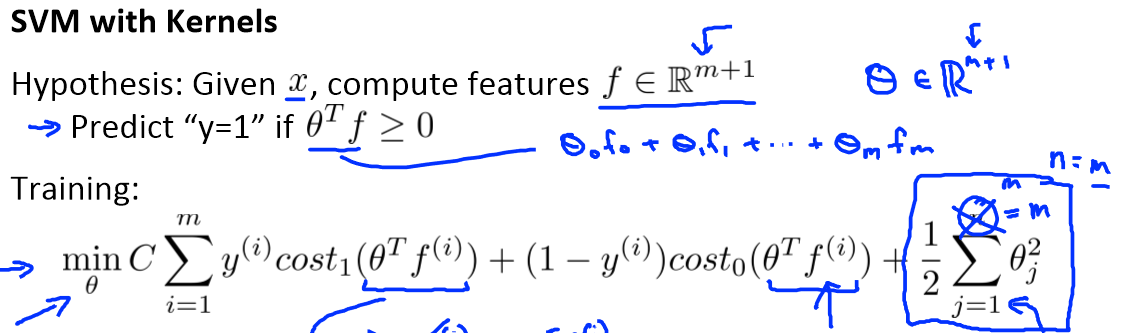
1. 将逻辑回归的代价函数做一点变化就可以的得到支持向量机（support vector machine，SVM）的代价函数，如下图所示，从中可以发现，去掉了m，并且去掉了第二项的λ而在第一项前添加了一个常数C，也就是从约束第二项变为约束第一项。

从上式也可以观察到，逻辑回归代价函数中的对数项变为了两个重新定义的函数cost1和cost0，其定义如下图所示，实际上就是用两段折线取代原来的对数曲线。

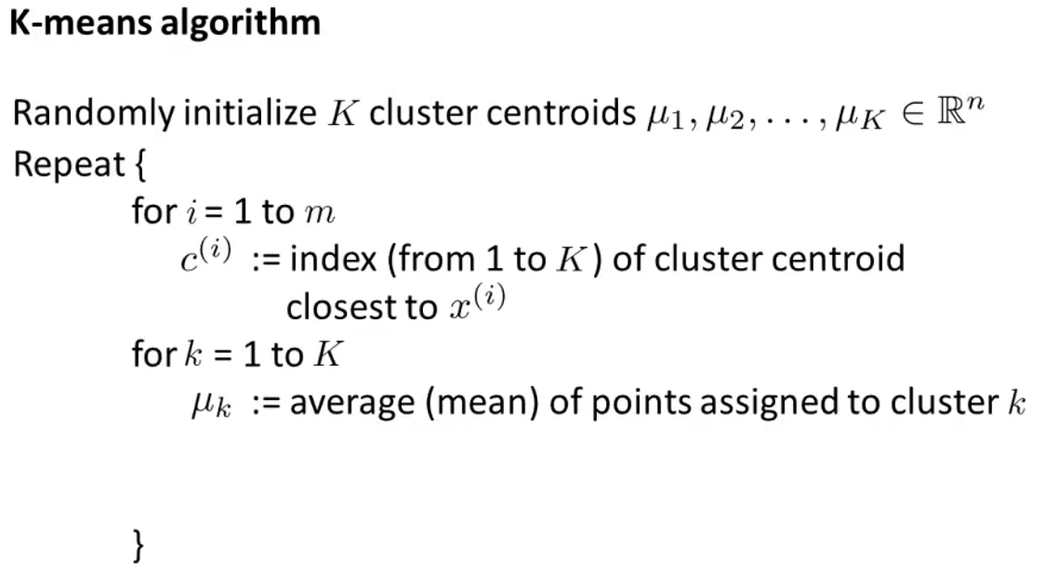
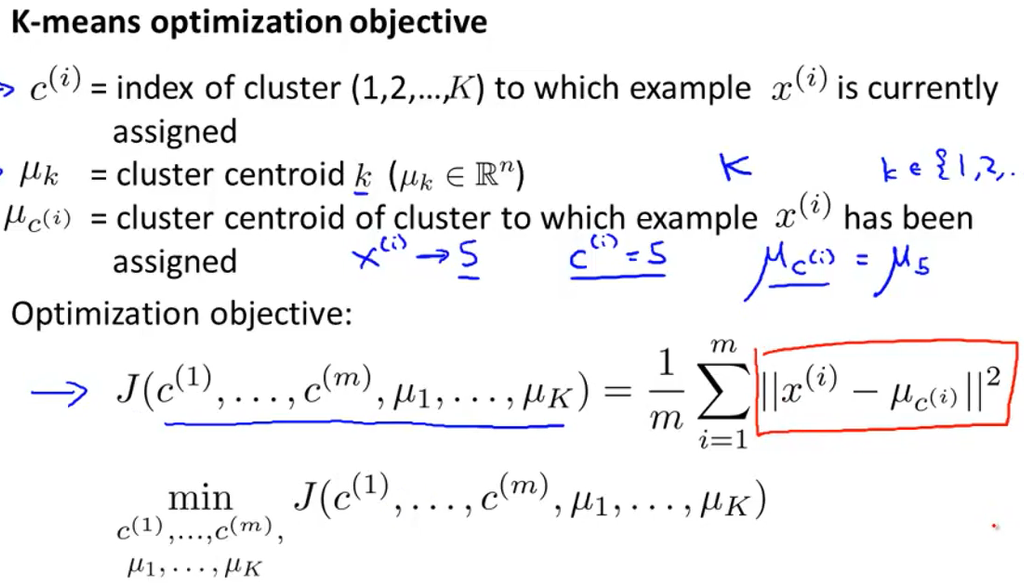
值得注意的是，支持向量机输出的h不再是y=1的概率。

1. 支持向量机又被称为最大间隔分类器，由支持向量机得到的决策边界（decision boundary）离正负样本都比较远，也就是说，能够比较好地把正负样本分开，得到较好的决策边界。当支持向量机代价函数中的C的取值很大时，容易受到异常值的干扰，但是只要适当减小C的值就可以大大降低这种干扰。
2. 为了拟合一个非线性的决策边界，在支持向量机中，我们定义新的特征f1,f2,f3……,其具体定义过程为：首先手动地取三个和x同维度的向量，然后定义f如下：

f1=exp(-)。其中‖m‖表示向量的范数，即向量的长度。这个定义f的函数就叫做核函数（kernels），上述这种核函数称为高斯核函数，其几何意义可以理解为向量x和l的“相似度”，x和l越接近，f的取值越靠近1，越远则f的取值越靠近0。σ是核函数的参数，其取值影响f由1趋近0的速度。σ2的值越大，f趋近0的速度越快；σ2的值越小，f趋近0的速度越慢。

1. 通过上面的描述可以知道，选择合适的l非常重要，那么如何选择合适的l呢？答案是直接把每一个x，即每一个example作为选择的l，有多少个x就选择多少个l，然后分别计算f作为新的特征。当θTf>=0时，就预测y=1。如下图所示：值得注意的是，最小化代价函数以求出θ的过程可以直接利用已有的一些高级工具。
2. 支持向量机的代价函数中的常数C类似于正则化逻辑回归代价函数中的1/λ，所以当C的取值很大时，容易造成低偏差高方差（过拟合）；当C的取值很小时，容易造成高偏差低方差（欠拟合）。另一个重要的常数是核函数中的σ2，当σ2的取值很大时，容易造成高偏差低方差（欠拟合）；当σ2的取值很小时，容易造成低偏差高方差（过拟合）。
3. 由于在支持向量机中，我们使用高级工具来计算θ，因此我们只需要选择C的取值以及选择合适的核函数。常用的核函数有线性核函数（即不使用核函数）和高斯核函数。在使用高斯核函数时，需要选择合适的σ2的值以及进行特征缩放。许多高级SVM软件包都内置了多类分类问题，可以直接使用。
4. 当特征数很多而训练样本较少时，通常使用逻辑回归或者线性核函数的SVM（太复杂很容易过拟合）；当特征数较少而训练样本数目适中时，可以使用高斯核函数的SVM；当特征数较少而训练样本很多时，可以人为地构造一些特征然后使用逻辑回归或者线性核函数的SVM。神经网络可以适用于上述的各种情形，但是有时训练起来会比较慢。

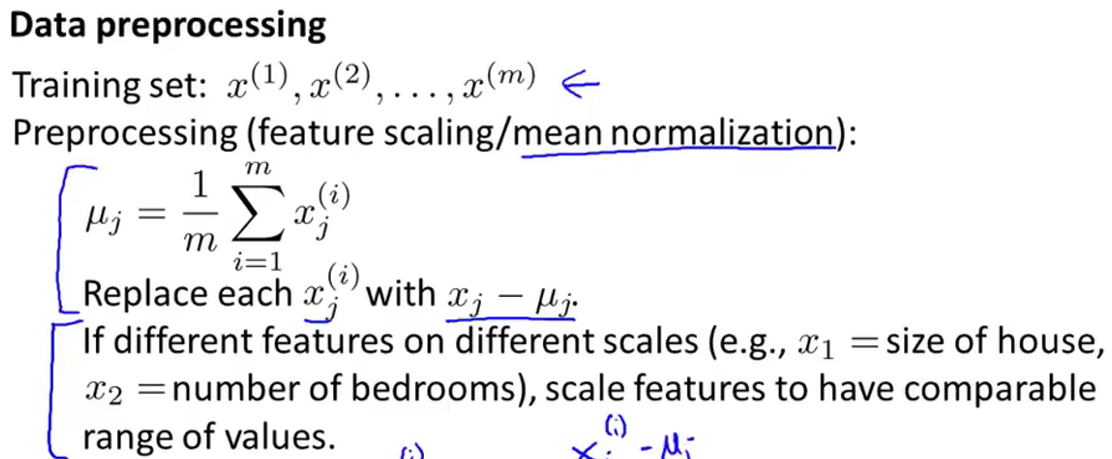
Unsupervised Learning

1. K均值算法（K-Means Algorithm）：K均值算法可以把一堆数据聚类成K个簇，其具体过程为：先随机选择K个聚类中心（cluster centroids）,然后按照各个数据点离聚类中心的距离把所有数据点分为K个簇（把每一个数据点归到离它最近的聚类中心所代表的簇里），这个过程称为簇分配；然后，把每一个聚类中心移动到它这一簇中所有数据点的平均位置，这个过程称为移动聚类中心。重复上述过程直到收敛就可以把所有数据划分为K个簇。（上述中的数据均为n维向量，“数据点”和“聚类中心”也是n维空间中的点）。
2. 如果簇分配后某个聚类中心一个数据点都没能分配到，通常我们就直接舍弃这个聚类中心。但这样最终就只能得到K-1个簇，如果一定要得到K个簇，则需要重新随机初始化一个聚类中心。
3. 和之前讲到的线性回归、逻辑回归等一样，K均值算法也有一个优化目标（optimization objective），或者说代价函数。它可以帮助我调试程序，保证算法正确运行，并且有利于找到更好的簇以及避免局部最优解。其代价函数如下（这个代价函数也被称为失真函数（distortion function））：

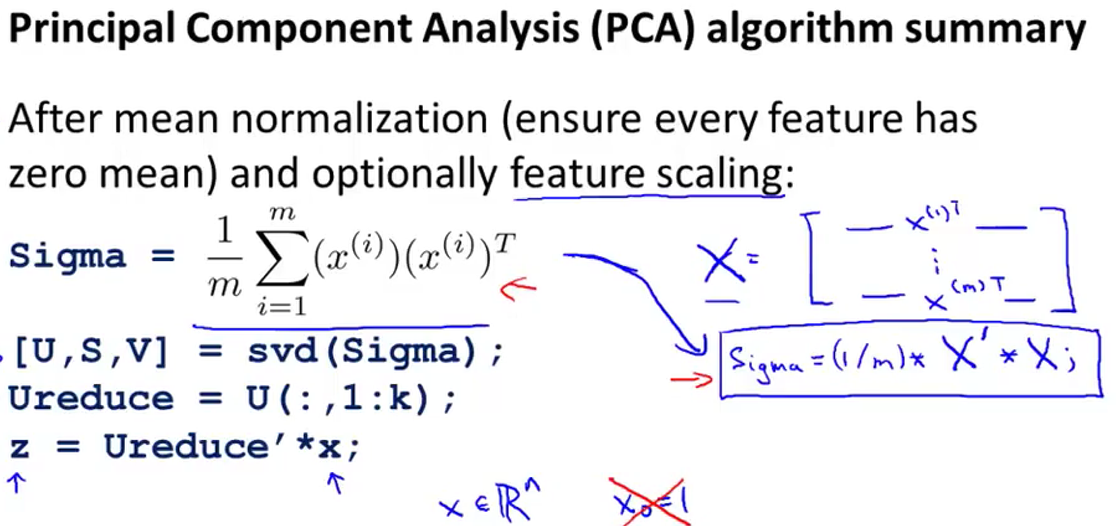
可以看到，这个代价函数的参数为c和μ。事实上，数学上可以证明，簇分配的过程就是固定μ而优化c，移动聚类中心的过程就是固定c而优化μ。

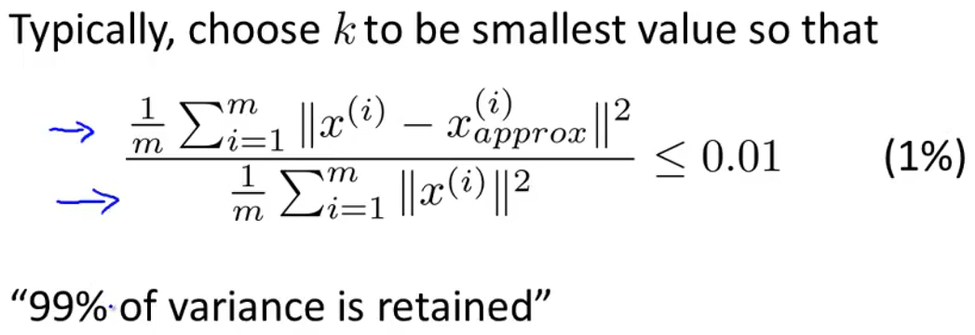
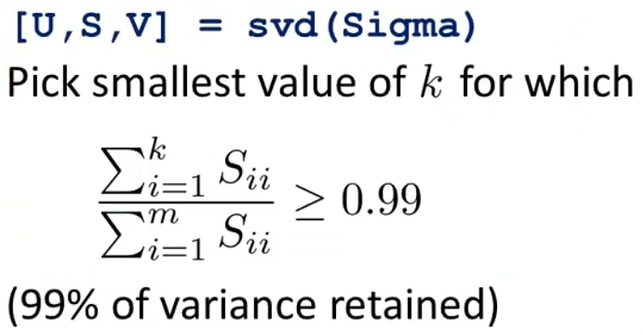
1. K均值算法中非常重要的一步是随机初始化K个聚类中心，通常的做法是这样的：从训练集中随机选取K个数据作为聚类中心。
2. 由于聚类中心初始化情况的不同，K均值算法也容易出现局部最优解（local optimal），为了找到一个相对较好的解，通常的做法是：连续做几十次K均值算法，每次随机选择聚类中心，并且计算代价函数，从而可以得到几十个不同的聚类结果，其中代价函数最小的聚类结果就是最好的结果。
3. 至于如何选择合适的聚类个数，事实上没有自动化选择的方法，最常用的方法是手动选择。手动选择也有两种方式，第一种称为肘部法则（elbow method）：作出代价函数对不同聚类数目的函数图像，即J-K图，选择函数图像上明显的拐点（肘部）对应的聚类数目。然而肘部法则并不好用，因为很多时候作出的图像上并没有明显的拐点。第二种方式就是根据你后续处理需要的聚类个数来决定。

Dimensionality Reduction

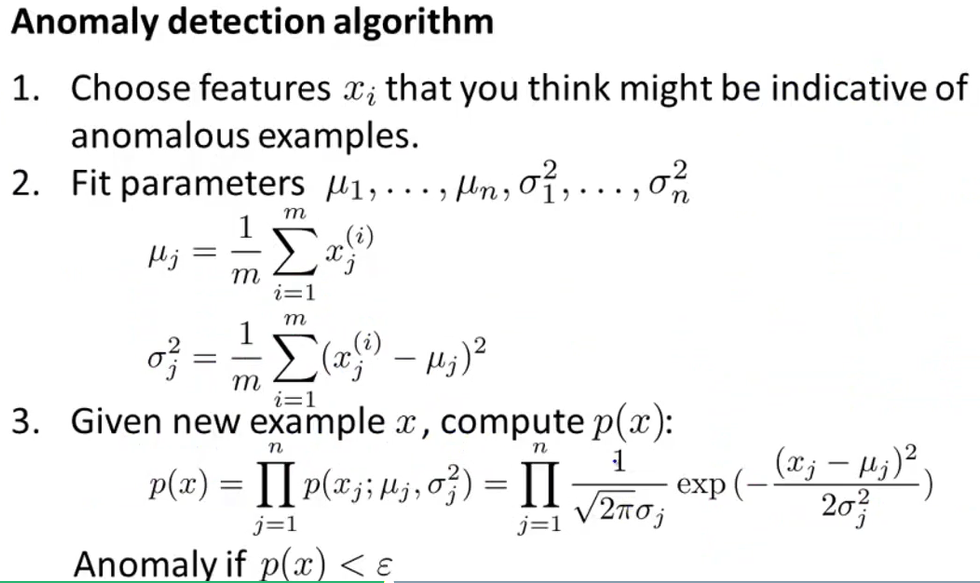
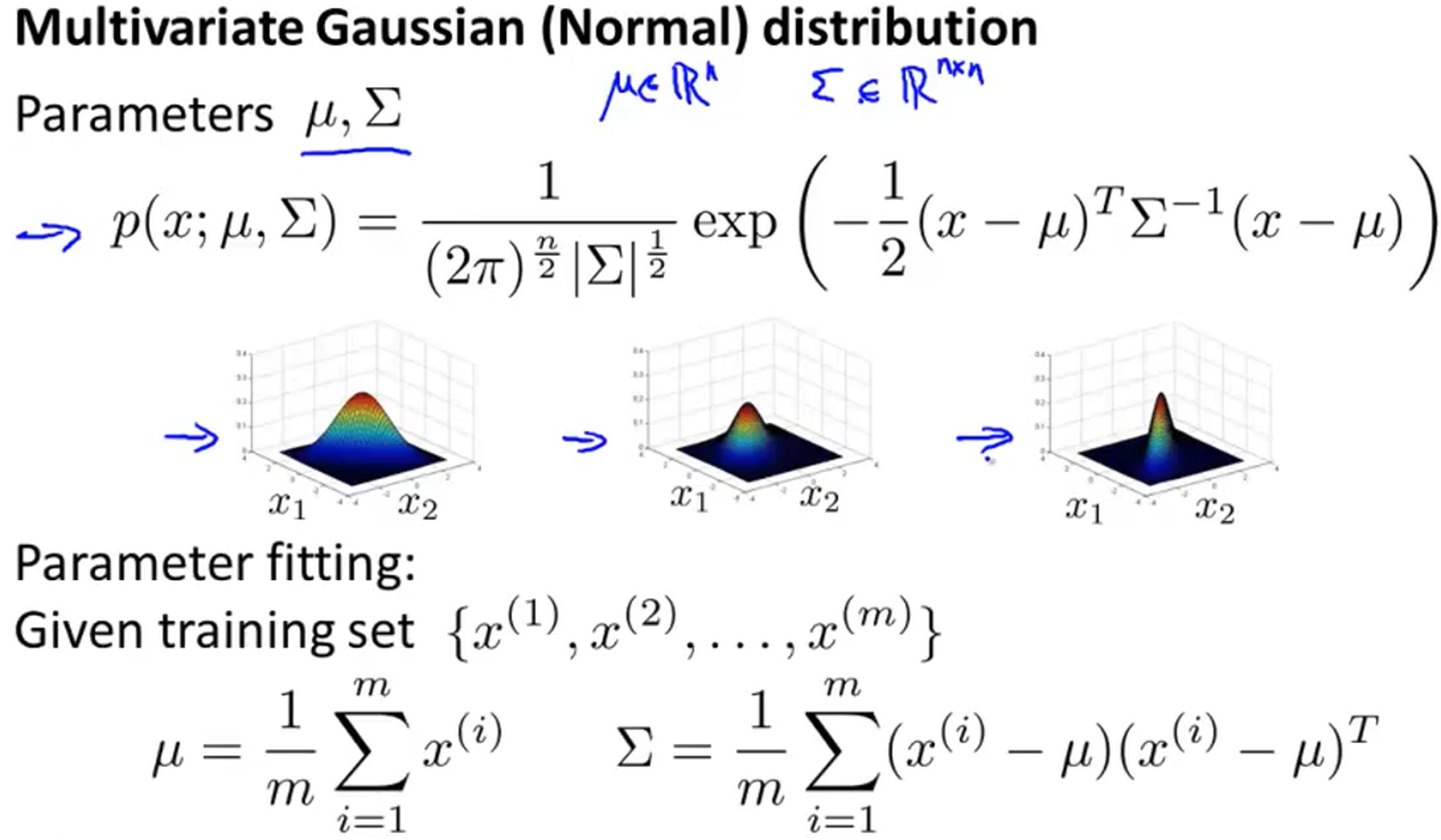
1. 降维不仅可以压缩数据，使其占用的计算机内存更少，还可以提高算法的运行效率。降维就是把高维空间的数据投影到一个低维的空间里。
2. 主成分分析（principal component analysis，PCA）就是找到一个合适的低维空间，使高维空间中的点到其相应的低维空间中的投影点的距离（即投影误差）的平方和最小。
3. 在运行PCA算法前，首先要对数据进行预处理，即进行特征缩放（feature scaling）和均值归一化（mean normalization）：

然后，计算出协方差矩阵Sigma，其定义如下图所示。再然后，利用奇异值分解函数svd计算出矩阵U，矩阵U的前k列就是我们需要的低维空间，最后用矩阵U的转置乘以x就得到了x在低维空间中的投影点 。具体过程如下：



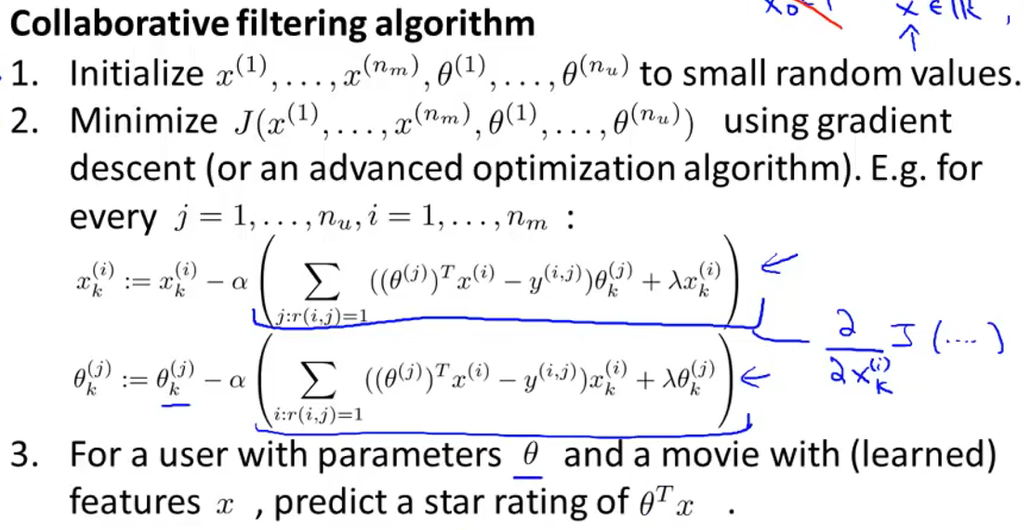
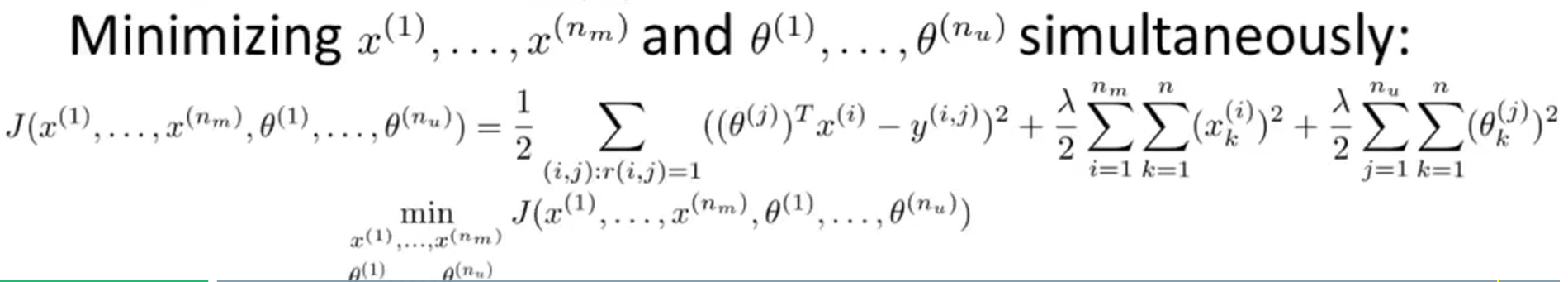
1. 如果已知低维空间中的数据，想要将其投影回高维空间，只需利用x=Ureduce\*z即可。由于投影误差的存在，这样得到的x与未降维前原始高维空间中的x有一定的误差。但是，由于PCA算法最小化了投影误差的平方和，所以这种误差并不是很大。
2. PCA算法中很重要的一点是选择合适的k值，即低维空间的维度，通常的选择方法如下：此时也称为99%的差异性被保留，保留多少差异性可以根据需要调整。上式只是定义，具体的实现方法如下：在调用svd函数时我们得到了三个矩阵：U、S、V，在计算Ureduce时只用到了U矩阵，在选择k的时候则需要用到S矩阵。S矩阵是一个对角矩阵，对角线以外的元素均为零，利用S矩阵便可以很容易的计算出上式中不等号左侧那个复杂的式子，经过一定的形式变换，可以得到下面的式子：
3. PCA可以压缩数据以节省存储空间，也可以把高维数据降维到二维或三维以实现数据可视化，还可以在监督学习中对高维的训练数据进行降维以提高监督学习算法的速度，但PCA不适合用来避免过拟合，避免过拟合应该使用正则化。

Anomaly Detection

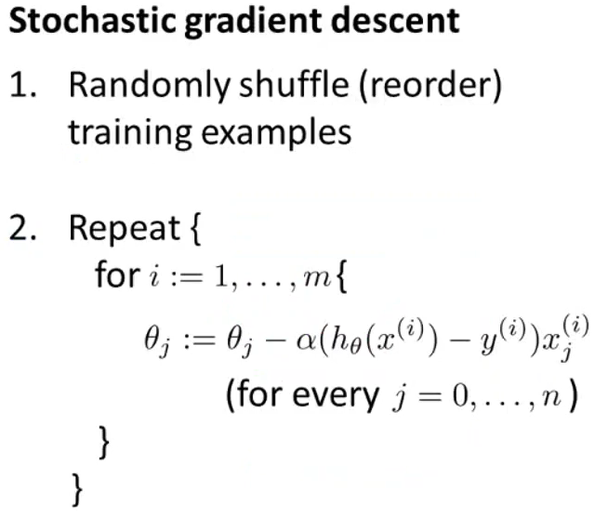
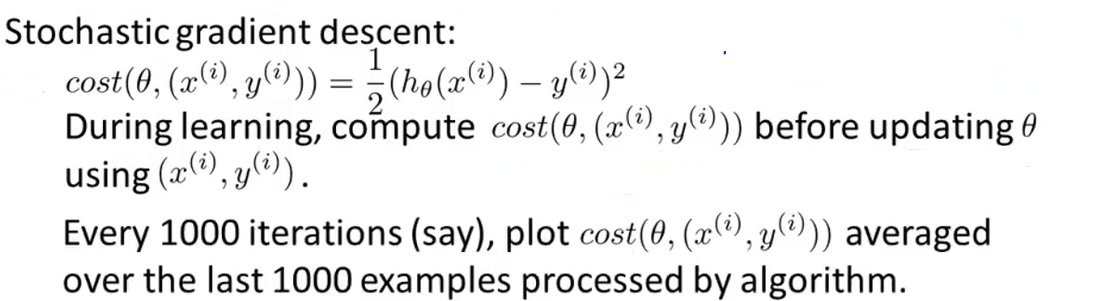
1. 所谓异常检测就是利用一些已经知道的正常状态的样本建立一个模型，从而判断一个新的样本是否正常，当算法的输出结果小于一个特定的阈值ε时，就认为该样本异常。
2. 高斯分布，也称正态分布，有两个参数：均值μ和方差σ2。
3. 若已知m个数据，每个数据有n个特征，假设这n个特征相互独立且都服从高斯分布，则可以得到n个高斯分布函数，将这n个高斯分布函数相乘就得到了异常检测的模型。具体过程如下：
4. 评估异常检测算法时，常用的方法是：用无标签的数据构成训练集（最好全是正常样本，但有少量异常样本也行），用带标签的数据构成交叉验证集和测试集。用训练集训练模型，然后用交叉验证集确定合适的阈值ε。值得注意的是，在异常检测问题中，通常情况下正常样本的数目远远多于异常样本，也就是说数据是非常偏斜的（skewed）。因此不应当用准确率来评估模型的效果，而应当使用精确率、召回率以及F值等指标。
5. 在评估异常检测算法时，我们使用了带标签的数据。那么，既然有带标签的数据，为什么不用监督学习而要使用异常检测算法呢？原因在于，在异常检测问题中，异常样本的数量相对于正常样本的数量来说非常少，在这种情况下异常检测算法比起逻辑回归等监督学习算法有更好的表现。
6. 在进行异常检测的时候，其实有些特征并不一定服从高斯分布，所以我们可以在选择特征的时候画出特征的直方图，观察一下它是否大致服从高斯分布。若服从，则可以直接使用；若不服从，则最好对其进行一些处理，例如取对数、取平方根或者开三次方等等，使其看起来大致服从高斯分布，然后用处理后的特征代替原特征，这样可以使算法的效果更好。
7. 在上面所讲的异常检测算法中，我们假设样本的各个特征相互独立，这样的得到的高斯分布是一个标准的钟形曲线（若只有两个特征则等高线图是一圈一圈的圆）。事实上，很多时候各特征之间并不是相互独立的，这时可以用到多元高斯分布。多元高斯分布不对每个特征分别构建一个高斯分布函数，而是只用一个高斯分布函数（若只有两个特征则等高线是一圈一圈的椭圆），这样可以反映出特征之间的相关性，使模型更准确。多元高斯分布函数具体如下：此处的Σ即协方差矩阵，与PCA中的协方差矩阵相同。
8. 通常来讲，使用普通的异常检测算法就够了。如果有明显相关性的特征，可以手动构造新特征以达到多元高斯分布的效果。而且，普通的异常检测算法也适用于样本数量（m）很少的情况，而多元高斯分布则要求样本数（m）必须大于特征数（n），事实上，只有在样本数量比特征数量大得多的时候使用多元高斯分布才有较好的效果。

1.

Recommender Systems

1. 基于内容的推荐：在这个电影推荐的例子中，我们可以根据电影的内容与各种主题（爱情、科幻、战争、动作等）的相关度抽象出一个特征变量x，然后对每一个用户做线性回归，以该用户已评分的电影为训练集得到该用户对应的θ，从而可以预测该用户对未观看过的电影的评分，进而推荐该用户可能喜欢的电影。
2. 协同过滤（collaborative filtering）：如果知道用户对不同主题电影的喜恶程度（即基于内容的推荐中的θ），同时也知道用户对某部电影的评分，就可以推测出这部电影的特征向量x，然后又可以根据基于内容的推荐对θ进行修正。由于每位用户都对多部电影进行了评分，每部电影也被多为用户评分，所以可以循环进行上述过程，不断地修正电影的特征变量x和用户的θ，从而达到较好的效果。
3. 也可以不反复循环，直接把x和θ综合到一起，同时优化这两个参数，也就是协同过滤算法，具体如下：其中J为代价函数，同时包括了x和θ这两个参数
4. 当我们计算出了各电影的特征向量x，就可以给用户推荐类似的电影，衡量两部电影相似度的指标是，这个值越小，两部电影就越相似。
5. 均值归一化（mean normalization）：计算每部电影的平均得分μ（总得分/评分人数），用每个人的评分减去相应的μ得到新的数据集，然后用这个新的数据集来进行协同过滤算法训练出θ和x。值得注意的是，此时用户对一部未观看过的电影的评分预测不再是θTx，而是θTx+μ。使用均值归一化的原因在于，如果有一位新用户尚未对任何电影进行评分，则不使用均值归一化的话计算出的这位用户的θ将会是[0;0;…0]，使用均值归一化可以避免这一问题。

Large Scale Machine Learning

1. 前面所讲的梯度下降算法也称为批量梯度下降（batch gradient descent），可以注意到，在批量梯度下降算法中，每一次计算梯度都需要读入全部的数据并求和，当数据量非常大时，这样做的效率就会很低。因此，对于大量数据，有另一种梯度下降算法，称为随机梯度下降（stochastic gradient descent）。随机梯度下降算法每次只需要读取一组数据，但是循环次数较多，不过还是比批量梯度下降算法快得多。不过随机梯度下降最后并不会恰好收敛到全局最小值，而是在全局最小值附近的一个范围内震荡。其具体过程如下：
2. 还有一种介于批量梯度下降和随机梯度下降之间的算法，称为小批量梯度下降（mini-batch gradient descent）。这种算法和前两种梯度下降算法的不同之处在于，每次计算梯度所需读取的数据既不是m个也不是1个，而是b个（b通常取2—100）。也就是把所有数据分为若干组，每组b个，每次读取一组。如果进行较好的向量化，小批量梯度下降甚至可以比随机梯度下降更快。具体过程如下：
3. 检查随机梯度下降是否在逐渐收敛的方法：
4. Map Reduce：在处理大量数据时，如果算法中遇到很大数量的样本的求和运算，可以把样本分为若干份，分别交给若干个计算机进行运算，然后再把各个计算机返回的数据相加即可，这样可以大大提高算法执行的速度。

Application Example：Photo OCR

1. 流水线（pipeline）：当面对一个复杂的机器学习问题（比如OCR）时，可以把整个问题分解为若干个模块，然后逐个解决或者分工合作，称之为流水线。
2. 滑动窗（sliding window）：当我们需要在一张图片里找到某个目标（比如人物或者文字）时，可以用一个矩形从图片的一个角开始逐渐滑过整张图片，对每一次滑动，可以用一个监督学习算法判断当前的矩形框里是否有我们想找的目标。
3. 只要我们有一个低偏差的算法，那么大量的学习数据通常都能使整个模型的性能得到很大的提升，所以应当想办法获得尽可能多的学习数据。有时也可以利用已有的少量训练数据人工合成更多的数据，甚至对于一些无标签数据手动打上标签。
4. 上限分析（ceiling analysis）：对于一个机器学习应用流水线，我们可以人为地将某个模块的准确率提高到100%，来看看这样做对于整个模型的性能有多大的提升，从而更好地分配资源，将更多的精力花在对模型性能提升最大的模块上。