**Exercise 3**

**3.1. finish the costFunction of Logistic Regression in using vector multipe**

For regularized logistic regression, the cost function is defined as:

The code with matlab:

J= (-y'\*log(h) - (ones(m, 1)-y)'\*log(ones(m,1)-h))/m+ theta(2:end)'\*theta(2:end)\*lambda/(2\*m);

The partial derivative of regularized logistic regression cost for θj is as follow:

And the code is:

grad(2:end) = (h-y)’\*X(2:end,:)/m + lambda\*theta(2:end)/m;

h = sigmoid(X\*theta);

function g= sigmoid(z)

g = 1.0./(1.0 + exp(-z));

end;

**Ex 3.2 One-vs-All classification:**

Main purpose is implement one-vs-all classification problem.

根据给出X数据, 训练出一个θ, 它是一个(K\*(n+1))的矩阵, K表示classes的数量, θ(i, :)将给出属于i分类的几率, n表示特征值数量.

通过一个for循环, 训练每一个class, 将获得的单一θ组合, 获得all-theta.

for i = 1: num\_labels

initial\_theta = zeros(n+1, 1);

options = optimset('GradObj', 'on', 'MaxIter', 50);

initial\_theta = fmincg(@(t)(lrCostFunction(t, X, (y == i), lambda)),initial\_theta, options);

all\_theta(i,:) = initial\_theta';

end

对给予的数据进行预测, 给于它的预测结果, 同样可以通过一句代码完成:

[tmp,p] = max(X\*all\_theta',[],2);

%% 第一步X\*all\_theta’获得了一个(m\*K)的矩阵, m表示sample数量, K表示class的数量, 每一行的表示这个样品的属于任意1~K的几率, max函数可以获得其下标.

**Ex 3.3 Neural Networks**

复习如何实现前向扩展(feedforward propagation), 通过一个训练好的3层的神经网络, 预测给予sample的结果. 代码实现如下

a2 = sigmoid([ones(m,1), X]\*Theta1');

%%获得第二层的值

a3 = sigmoid([ones(m,1), a2]\*Theta2');

%%获得经过神经网络处理的数据的结果.

[tmp, p] = max(a3, [], 2);

%%根据属于每一个class的几率大小, 比较得到sample最大几率属于哪一个class

**Exercise 4**

**Ex 4.1 Feedforword and cost function**

Implement of cost function of neural network model, the formula is as follow:

m代表样本数量, K代表分类数目, 最终结果的结果为(m\*K)的矩阵, 上述的公式通过两次迭代统计了每一个点代表的cost.

**以下是比较蠢的两次迭代代码:**

for r = 1 : m

for c = 1 : num\_labels

cur\_y = c == y(r);

J = -cur\_y\*log(h2(r,c)) -(1-cur\_y) \*log(1-h2(r,c)) + J;

end

end

**稍微好一点的实现, 通过矩阵运算:**

J = 0;

for r = 1 : m

tmp\_y = zeros(1, num\_labels);

tmp\_y(y(r)) = 1;

J = J + (-tmp\_y\*log(h2(r,:)') - (1.-tmp\_y)\*log(1.-h2(r,:)'));

end

**Ex 4.2 Regularized cost function**

计算正则化之后的cost function, 各个层的theta, 除了bias theta,的平方的和. Formula is given as follow.

`

在正常的cost function的基础上添加regularized增加的cost即可.

J = J + (sum(sum(Theta1(:,[2:end]).\*Theta1(:,[2:end]))) + sum(sum(Theta2(:,[2:end]).\*Theta2(:,[2:end]))))\*lambda/(2\*m);

**Ex 4.3 Sigmoid gradient**

**求导可得, g**’(z**) = g(z)(1-g(z)):**

gg = sigmoid(z);

g\_grad = gg.\*(1.-gg);

**Ex 4.4 Random initialization**

神经网络的初始参数都为零或者不同的神经节点的参数一致, 否则会导致不同的节点获得的结果, 进行的下一步的优化都是一致的, 这样将导致同一层的多个节点称为摆设, 不能达到多点优化, 改进的效果.

% Randomly initialize the weights to small values

Epsilon\_init = 0.12;

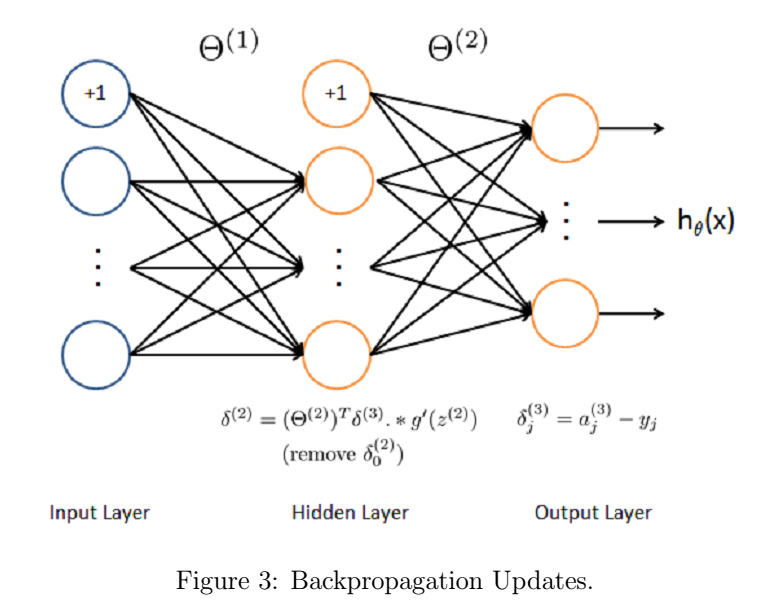
W = rand(L out, 1 + L in)\*2\*epsilon\_init – epsilon\_init;

Exercise给出如何实现, 重点应该是epsilon\_init的选择, 文中介绍了一种策略,

epsilon\_init = ,

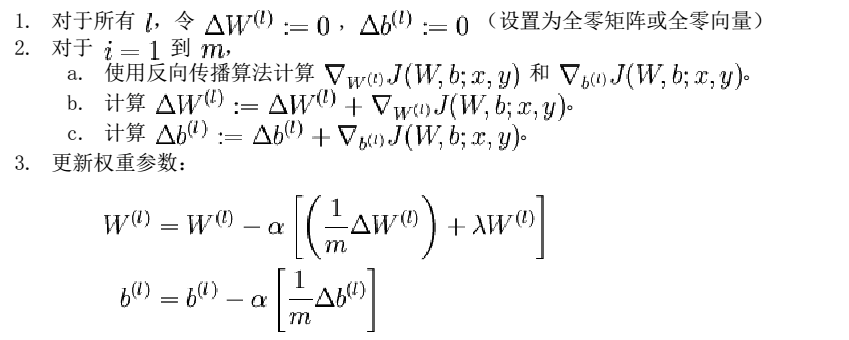
Lin和Lout分别为输入层节点的数据量和目的网络层的节点数目.

**Ex 4.5 Back progpagation**



表示第n层的第i项的”残差”, -()

当n为输出层时, 上述可以直接以上述公式获得输出层的残差. 当不为输出层时, 可以通过反向传播获得各个层的”残差”:



遍历所有的样品, 根据输出结果, 反向传播获得各层的残差, 利用残差结果计算得到各个θ的偏差, 遍历所有结果后, 获得各个θ的偏导数. 练习中并未对规则项进行重复梯度下降迭代.

for i =1: m

a1 = [ones(1,1), X(i,:)];

z2 = a1\*Theta1';

a2 = [ones(1,1), sigmoid(z2)];

z3 = a2\*Theta2';

a3 = sigmoid(z3);

%%获得各个层的输入, 输出值

yy = zeros(num\_labels, 1);

yy(y(i)) = 1;

delta\_3 = (a3' - yy);

delta\_2 = Theta2'\*delta\_3.\*(sigmoidGradient([ones(1,1), z2]'));

%%获得各个层的残差

delta\_2 = delta\_2(2:end);

Theta1\_grad = Theta1\_grad + delta\_2\*a1;

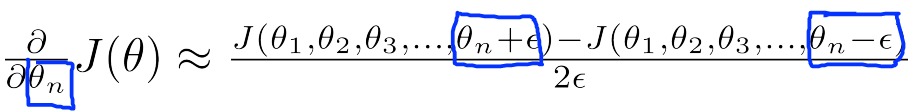
Theta2\_grad = Theta2\_grad + delta\_3\*a2;

%%利用残差结果, 得到该样品对总体偏差的影响.

end

Ex 4.6 Numerical estimation of gradients

为了验证我们得到的结果, exercise中提供了一个对gradient进行check的方法, 首先, 将层层之间转换的theta展开为一个长的向量THETA, matlab中可以通过[theta1(:); theta2(;)]完成. 初始化Theta为0, 然后通过以下公式, 得到Thetan的偏导数.



代码在exersize指导中以给出, 主要代码如下.

e = 1e-4;

for p = 1:numel(theta)

% Set perturbation vector

perturb(p) = e;

loss1 = J(theta - perturb);

loss2 = J(theta + perturb);

%% J()是costfunction函数的句柄.

% Compute Numerical Gradient

numgrad(p) = (loss2 - loss1) / (2\*e);

perturb(p) = 0;

end

Ex 4.7 Regularized Neural Networks

与线性回归和逻辑回归的正则化一样, 对应偏差权重的权重项不需要进行正则

可以直接得到其正则化后的权重偏导数

Theta1\_grad =Theta1\_grad/m + [zeros(size(Theta1,1), 1), Theta1(:, 2:end)]\*lambda/m;

Theta2\_grad = Theta2\_grad/m + [zeros(size(Theta2,1), 1), Theta2(:, 2:end)]\*lambda/m;

**Exercise 5. Regularized Linear Regression and Bias v.s.Variance**

**Ex 5.1 Regularized linear regression cost function and Regularized linear regression gradient**

**线性回归的损失函数和偏导数,**

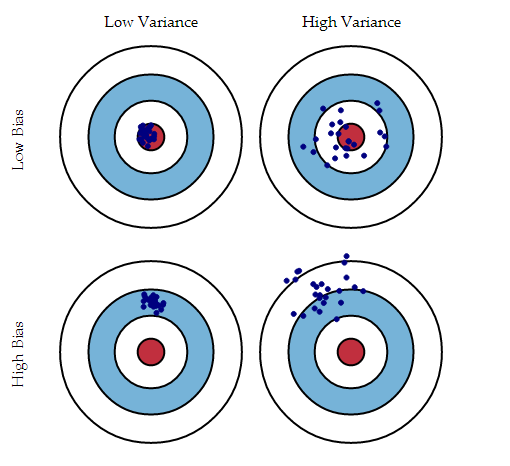
For

For

J = (X\*theta -y)'\*(X\*theta -y)/(2\*m) + theta(2:end)'\*theta(2:end)\*lambda/(2\*m);

grad = (X'\*(X\*theta - y))/m + [zeros(1,1); theta(2:end)]\*lambda/m;

**Ex5.2 Bias-Variance Tradeoff**

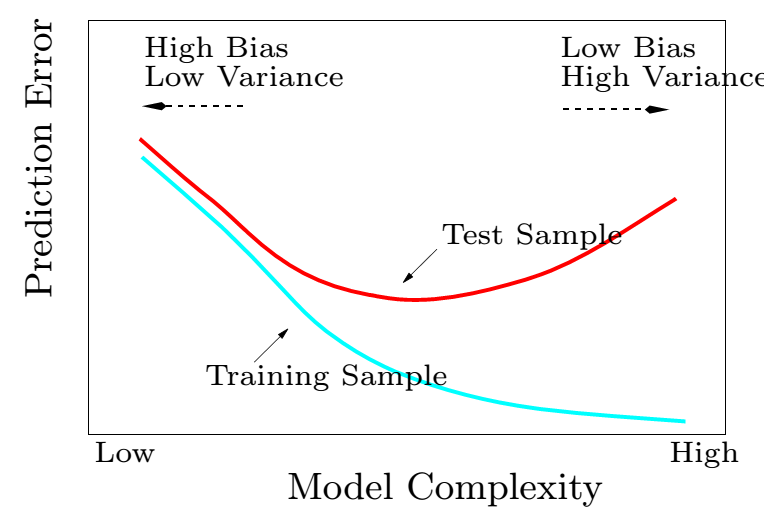
****

**Ref.**

**http://scott.fortmann-roe.com/docs/BiasVariance.html**

**http://en.wikipedia.org/wiki/Bias%E2%80%93variance\_tradeoff**

**与之对应的机器学习的模型复杂度对bias和variance的影响**



当模型复杂度越低, 其偏差越高, 而标准差越低, 而随着模型复杂度的上升, 偏差降低, 但是其标准差则会逐渐增大.

本真噪音是任何学习算法在该学习目标上的期望误差的下界；( 任何方法都克服不了的误差)

bias 度量了某种学习算法的平均估计结果所能逼近学习目标的程度；（独立于训练样本的误差，刻画了匹配的准确性和质量：一个高的偏差意味着一个坏的匹配）

variance 则度量了在面对同样规模的不同训练集时，学习算法的估计结果发生变动的程度。（相关于观测样本的误差，刻画了一个学习算法的精确性和特定性：一个高的方差意味着一个弱的匹配）

为了观察学习曲线—随着训练集增加, 训练误差和交叉检查集误差的大小, 定义了新一个新的学习曲线函数.

for i =1 :m

theta = trainLinearReg(X(1:i,:), y(1:i,:), lambda);

error\_train(i) = linearRegCostFunction(X(1:i,:), y(1:i,:), theta, lambda);

error\_val(i) = linearRegCostFunction(Xval, yval, theta, lambda);

end

为了生成高复杂度的模型, 生成了一个函数以生成高阶数据集

X\_poly = X(:);

for i = 2: p

X\_poly(:, end +1) = X\_poly(:,end).\*X(:);

end

生成高阶数据集带来的问题是数值过大, 需要重新normalization

mu = mean(X);

X\_norm = bsxfun(@minus, X, mu);

sigma = std(X\_norm);

X\_norm = bsxfun(@rdivide, X\_norm, sigma);

ex5脚本的训练结果表明, 在高模型复杂度下, 训练误差几乎为零, 而交叉检查集的误差随着训练集的增加而减小, 说明模型过拟合.

Ex 5.3. Adjust Regularization Parameter

实验, 通过调整正则项对模型的影响

m = length(lambda\_vec);

for i = 1: m

lambda = lambda\_vec(i);

theta = trainLinearReg(X, y, lambda);

error\_train(i) = linearRegCostFunction(X, y, theta, 0);

error\_val(i) = linearRegCostFunction(Xval, yval, theta, 0);

end

**Exercise 6 Support Vector Machines**

Exercise的svmtrain.m中已给出svm算法的matlab代码, 但是并未优化, 练习中的训练集数据量不大, 可以直接使用这个代码实现. 如果训练集很大的情况下, 更应该使用成熟的, 经优化的代码, 如libsvm库的代码, 它用多种语言, C++, python, matlab, Java实现了SVM.

Ex 6.1 Impact of Parameter C on the cost of the algorithm.

The formula of cost function of SVM is as follow:

参数*C*控制分类错误产生的损失的大小, C值越大, 表示错配的产生的损失越大, 则最终产生的模型中, 会尽量保证匹配正确.

通过修改C值的大小, 发现本例给出的训练集中, C值在1时, 生成的Decision Boundary是介于两个类中间, 但是却导致了一个异常值不能被正确分类, 当C值在100时, 生成的Decision Boundary保证了所有的训练集的正确分类. 而C值在0.01时, 生成的Decision Boundary完全偏离了两个集合的分界线.

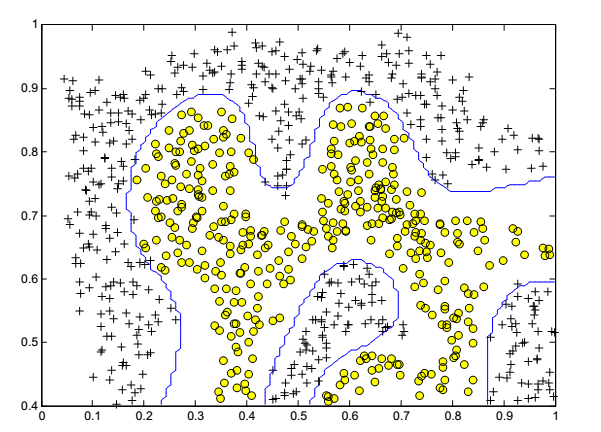
Ex 6.2 Gaussian Kernels

高斯核函数是SVM中最常用的核函数, 它实际的意义是两个点在空间中的距离的函数.

使用matlab实现以上Gaussian Kernel函数

sim = exp(-(x1 - x2)'\*(x1 - x2)/(2\*sigma^2));

ex6的脚本将根据Gaussian Kernel函数和svmtrain.m和svmpredict.m给出数据集Dataset2的boundary.



Ex 6.3 Choice of Parameter C and .

利用给出的svmtrain.m和svmpredict.m脚本, 通过调整不同的C和参数验证交叉检测数据集的正确率, 给出最优的参数组合.

给出的比较愚笨的matlab的实现.

res = zeros(m, m);

yval = yval(:);

for i = 1:m

C = candicate(i);

for j = 1: m

sigma = candicate(j);

model = svmTrain(X, y, C, @(x1, x2) gaussianKernel(x1, x2, sigma));

pred = svmPredict(model, Xval);

pred = pred(:);

res(i, j) = mean(double(pred == yval));

end

end

[p1, p1] = max(max(res'));

[p2, p2] = max(max(res));

Exercise 7: K-means Clustering and Principal Component Analysis

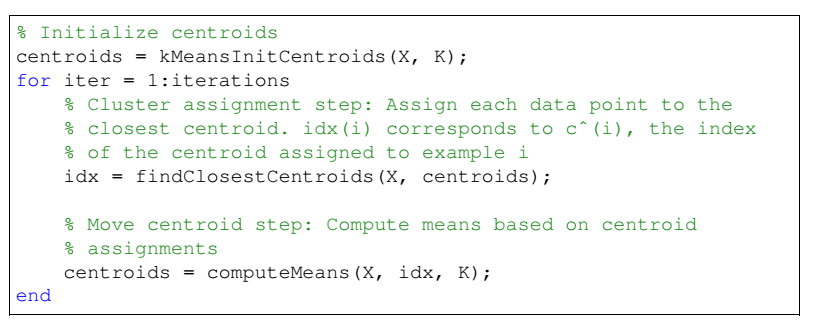
给予一个数据集，按要求对其进行分类cluster，但是并没有给出正确的学习方法应该给出的值，即成为非监督型学习。

K-means方法被用于非监督型学习。学习方法假设数据集被分为K个集合。

K-means algorithm：

1. 随机选取K个点分别为K个数据集合的中心。
2. 计算每个数据集合点到K数据中心的距离，取最近的集合中心，将数据分类到该数据中心。
3. 对每个集合中心，计算中心到所有的属于这个中心的数据集的向量之和，取平均值，得到集合中心的位移向量，以这个向量对集合中心进行位移。
4. 迭代步骤1-3。

需要实现K-means算法的matlab版本:



Ex7.1 Finding closest centroids

找到每一个数据集的最近中心点。

for i = 1: n

tmp\_centroid = centroids;

tmp = bsxfun(@minus, tmp\_centroid, X(i,:));

res = zeros(K, 1);

for j =1 : K

res(j) = tmp(j,:)\*tmp(j,:)';

end

[idx(i),idx(i)] = min(res);

end

Ex7.2 Computing centroid means

计算每一个数据中心点的位移向量：所有的归属于数据中心到所有归属该分类的数据集的向量之和，再取平均值。

count = zeros(K, 1);

for i =1 :m

centroids(idx(i),:) = centroids(idx(i),:) + X(i,:);

count(idx(i),1) = count(idx(i),1) + 1;

end

centroids = bsxfun(@rdivide, centroids, count);

Ex7.2 Random initialization

K-means方法受限于最初的K中心数据点的选择，可能得到一个局部最优解，因而需要通过多次随机选择K中心数据点的，获得一个全局最优解。

R = randperm(size(X, 1));

Initial\_centroid = X(R(1:K), :);

Ex 7.3 Image compression with *K*-means

imread可以用于多种通用的图像格式，jpeg，png，tiff，bmp等等，获得一个3维的矩阵，前面两个维度是图像上像素点的位置，第3维表示的像素的颜色的位，存储位的格式视图像格式而定。

Exercise中实现的K-means方法是基于2维数据，因为首先利用reshpae函数，将图片存储于2维矩阵。之后调用K-means方法，获得点数据集的分类信息。利用这些点的分类信息，将每个点赋值为它最近的分类中心的点。

Ex 7.4 Principle Component Analysis

PCA，主成分分析，可用于将高维的数据降维，同时尽量保持数据特征。主要在其他学习算法前应用与数据，用于数据降维，提高运算速度，或者将数据降维到2/3维，以实现可视化。

协方差矩阵，一个表示多个向量之间协方差的矩阵。

计算包含多个向量的二维矩阵，首先需要将数据均一化，exercise里调用了featureNormalize.m。

mu = mean(X);

X\_norm = bsxfun(@minus, X, mu);

sigma = std(X\_norm);

X\_norm = bsxfun(@rdivide, X\_norm, sigma);

获得的均一化的数据，已经完成了减去平均值的部分，可以通过以下公式直接获得协方差矩阵。之后调用svd方法，进行奇异值分解。

%%sigma表示原始数据的协方差矩阵。

sigma = (1/m)\*(X'\*X);

%%svd是matlab的奇异值分解方法。U是特征矩阵，包含了需要的主成分，而S包含了一个对角矩阵

[U,S,V] = svd(sigma);

Ex7.5 Dimensionality Reducation with PCA

利用奇异值分解获得特征矩阵，以特征矩阵对原数据进行降维。

Z=X\*U(:,1:K);

U\_reduce = U(:,1:K);

Z = X\*U\_reduce;

Ex7.6 Reconstructing an approximation of the data

利用特征矩阵，将降维后的数据还原。

U\_reduce = U(:,1:K);

X\_rec = Z\*U\_reduce';

**Summary of PCA method:**

1. Preprocessing, normalize the dataset.
2. Obtain the covariance matrix
3. Use singular value decomposition the obtain the eigenvector
4. Compress the dataset the K dimensionality by dataset plus first K column of the eigenvector.
5. If it is necessary, reconstruction the data by plus the compressed dataset and transposed matrix of first K column of the eigenvector.

Ex7.7 PCA on faces

利用PCA method对高维的image进行降维，以一张32×32×X的图片为例，通过reshape函数，生成1024×X的二维矩阵，

Exercise 8: Anomaly Detection and Recommender Systems

8.1. Anomaly detection

异常检测，通过给予的训练样本，学习得到变量和的（基于样本服从高斯分布的假设），获得样本特征变量和该样本属于正常的几率；通过cross validation样本—包含异常样本数据，学习获得；对于待检测样本，如果的几率低于，则判定样本异常。

异常检测，类似于supervised learning中的logistic regression，目的都是判断样本是否属于一个类。在以下情况下，往往采用anomaly detection而不是supervised learning：异常样品过少，导致supervised learning不能建立正确的模型，大量的negative样本，Anomalies的类型各种各样，不能通过已有的异常样本学习到所有的异常，并且将来出现的各种异常可能与已知的所有异常都不一致。

常用anomaly detection的案例：欺诈检测，生产产品检测、数据中心的机器状态检测……

对于一些non-gaussian的特征结果，可以通过取对数，取指数等方法获得类似高斯分布的数据。

本章节的假设样本都符合高斯分布：

考虑到样本之间可能的联系，如果使用多变量高斯分布，则有:

其中协方差，均值如下：

相对于多变量高斯分布，原始的简单模型计算简单，但是可能需要手动增加变量之间联系的特征量。而多变量高斯模型能够自动获取特征量之间的关系，然后由于计算一个矩阵的逆矩阵，在特征量比较大的情况下，这个计算是十分耗时的。**采用多变量模型的一个前提是样本量必须大于特征量m>n，否则获得的协方差矩阵不可逆。**

8.2. Implement estimateGaussian.m

为了实现高斯函数，需要从数据集中获得参数和：

Matlab的std函数的原型是s = std(X,flag,dim)，flag=0表示结果除以m-1，flag=1表示结果除以m。

mu=mean(X,1);

tmp = bsxfun(@minus, X, mu);

sigma2 = sum(tmp.\*tmp, 1)./m;

mu = mu';

8.3. Selecting the threshold，

通过检查验证集的数据，获得最适的。以F1值作为检验标准，判断是否合理。

其中prec是precision，精确度；而rec为recall，回收率。

tp为true positive，预测结果为true，实际结果也为true；

fp为false positive，预测结果为true，实际结果为negtive；

fn为true negative，预测结果为false，实际结果为positive；

for epsilon = min(pval):stepsize:max(pval)

predictions = pval < epsilon;

fp = sum(predictions == 1 & yval == 0);

tp = sum(predictions == 1 & yval == 1 );

fn = sum(predictions == 0 & yval ==1 );

prec = tp/(tp+fp);

rec = tp/(tp + fn);

F1 = 2\*prec\*rec/(prec+rec);

if F1 > bestF1

bestF1 = F1;

bestEpsilon = epsilon;

end

end

8.4. Recommender Systems

推荐系统，以movie评分为例，一部电影可以分解为个特征量，，而每位用户则同样一个参数，，用户i对于某一部电影j的喜好可以表示为。推荐系统，通过已有的用户对电影的评分记录，学习和，从而用于判别未知的用户对电影的评分。

8.5 Collaborative filtering cost function

没有正则项的协同过滤算法的损失函数的公式如下：

的下标表示，只计算，也就是有评分记录的值。是movie的数目，是user的数目。用matlab实现如下：

1. J = sum(sum((X\*Theta'-Y).\*(X\*Theta'-Y).\*R))/2;

8.6 Collaborative filtering gradient

实现协同过滤损失函数的偏导数（无正则项），公式如下：

1. X\_grad = (X\*Theta'-Y).\*R\*Theta;
2. Theta\_grad= ((X\*Theta'-Y).\*R)'\*X;

8.7. Regularized cost function

正则化后的损失函数如下：

取偏导数：

用matlab实现如下：

|  |
| --- |
|  |

1. J = sum(sum((X\*Theta'-Y).\*(X\*Theta'-Y).\*R))/2 + lambda/2\*(sum(sum(Theta.\*Theta)) + sum(sum(X.\*X)));
2. X\_grad = (X\*Theta'-Y).\*R\*Theta + lambda\*X;
3. Theta\_grad= ((X\*Theta'-Y).\*R)'\*X + lambda\*Theta;