Exercise 3

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

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

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

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:

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

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

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), la
mbda)),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 c
lass 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:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right],$$

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.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right].$$

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(su
m(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,

$$\text{epsilon\_init} = \frac{\sqrt{6}}{\sqrt{L_{\text{in}}+L_{\text{out}}}},$$

L_{in} 和 L_{out} 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

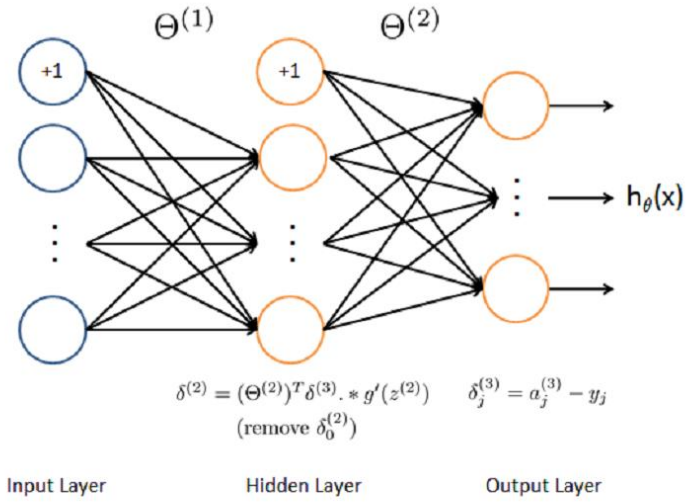


Figure 3: Backpropagation Updates.

δ_i^n表示第 n 层的第 i 项的"残差", $\delta_i^n = a_i^n - y_i^n = -(y_i^n - a_i^n) \cdot f'(z_i^n)$ -()

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

1. 对于所有 l，令 $\Delta W^{(l)} := 0$，$\Delta b^{(l)} := 0$ （设置为全零矩阵或全零向量）
2. 对于 $i = 1$ 到 m，
   a. 使用反向传播算法计算 $\nabla_{W^{(l)}} J(W,b;x,y)$ 和 $\nabla_{b^{(l)}} J(W,b;x,y)$。
   b. 计算 $\Delta W^{(l)} := \Delta W^{(l)} + \nabla_{W^{(l)}} J(W,b;x,y)$。
   c. 计算 $\Delta b^{(l)} := \Delta b^{(l)} + \nabla_{b^{(l)}} J(W,b;x,y)$。
3. 更新权重参数：

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$
$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

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 Theta_n 的偏导数.

$$\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n+\epsilon) - J(\theta_1,\theta_2,\theta_3,\ldots,\theta_n-\epsilon)}{2\epsilon}$$

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

$$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial\theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \qquad \text{for j} >= 0$$

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,

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

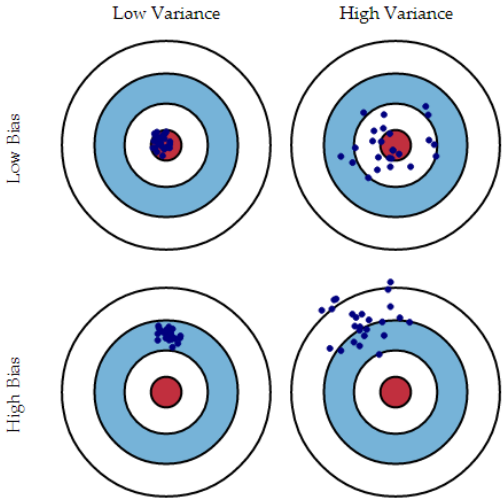
$$\frac{\partial J(\theta)}{\partial \theta_0} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \qquad \text{for } j = 0$$

$$\frac{\partial J(\theta)}{\partial \theta_j} = \left(\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}\right) + \frac{\lambda}{m}\theta_j \qquad \text{for } j \geq 1$$

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

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

Ex5.2 Bias-Variance Tradeoff

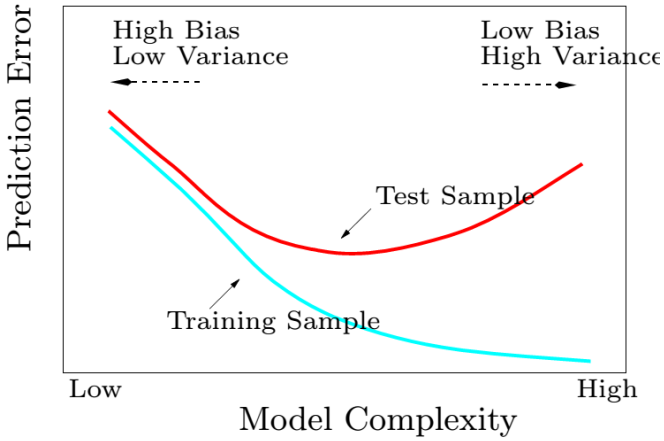


Ref.

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,:), t
heta, lambda);
    error_val(i) = linearRegCostFunction(Xval, yval, theta, lambd
a);
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:

$$\min_{\theta} C\sum_{i}^{m}[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

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

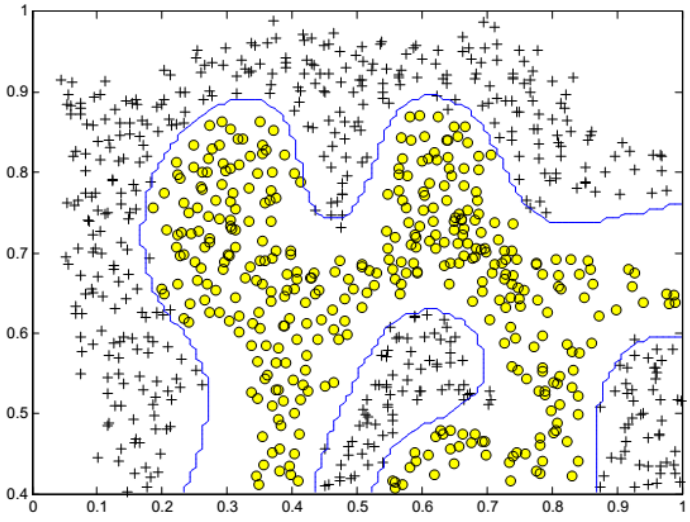
高斯核函数是 SVM 中最常用的核函数，它实际的意义是两个点在空间中的距离的函数.

$$K_{gaussian}(x^{(i)}, x^{(j)}) = exp\left(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2}\right) = exp\left(-\frac{\sum_{k=1}^{n}\left(x_k^{(i)} - x_k^{(j)}\right)^2}{2\sigma^2}\right)$$

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 σ.

利用给出的 svmtrain.m 和 svmpredict.m 脚本, 通过调整不同的 C 和σ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:

```
% Initialize centroids
centroids = kMeansInitCentroids(X, K);
for iter = 1:iterations
    % Cluster assignment step: Assign each data point to the
    % closest centroid. idx(i) corresponds to c^(i), the index
    % of the centroid assigned to example i
    idx = findClosestCentroids(X, centroids);

    % Move centroid step: Compute means based on centroid
    % assignments
    centroids = computeMeans(X, idx, K);
end
```

Ex7.1 Finding closest centroids

找到每一个数据集$x^{(i)}$的最近中心点$c^{(i)}$。

$$c^{(i)} := j \; that \; minimizes \; \left||x^{(i)} - \mu_j|\right|^2$$

```
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。

$$\mu_k := \frac{1}{|C_k|}\sum_{i \in C_k} x^{(i)}$$

```
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

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:

A. Preprocessing, normalize the dataset.

B. Obtain the covariance matrix

$$\Sigma = \frac{1}{m} X^T X$$

C. Use singular value decomposition the obtain the eigenvector

D. Compress the dataset the K dimensionality by dataset plus first K column of the eigenvector.
E. 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μ和σ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$p(x;\mu,\sigma^2)$；通过 cross validation 样本—包含异常样本数据，学习获得ϵ；对于待检测样本，如果$p(x^{test};\mu,\sigma^2)$的几率低于ϵ，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：

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

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:

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

其中协方差Σ，均值μ如下：

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

$$\Sigma=\frac{1}{\mathrm{m}}\sum_{i=1}^{m}\left(x^{(i)}-\mu\right)\left(x^{(i)}-\mu\right)^T$$

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μ和σ：

$$\mu=\frac{1}{m}\sum_{j=1}^{m}x_i^{(j)}$$

$$\sigma_i^2=\frac{1}{m}\sum_{j=1}^{m}\left(x_i^{(j)}-\mu_i\right)^2$$

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，ε

通过检查验证集的数据，获得最适的ε。以 F1 值作为检验标准，判断ε是否合理。

$$F_1 = \frac{2 \cdot \text{prec} \cdot \text{rec}}{\text{prec} + \text{rec}}$$

其中 prec 是 precision，精确度；而 rec 为 recall，回收率。

$$prec = \frac{tp}{tp + fp}$$

$$rec = \frac{tp}{tp + fn}$$

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n_f个特征量，$x \in \mathbb{R}^{n_f}$，而每位用户则同样一个参数$\theta^{\mathrm{i}}, \theta^{(i)} \in \mathbb{R}^{n_f}$,用户 i 对于某一部电影 j 的喜好可以表示为$y = \left(\theta^{(i)}\right)^T x^{(j)}$。推荐系统，通过已有的用户对电影的评分记录，学习$\theta$和$x$，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：

$$J\left(x^{(1)}, \ldots, x^{(n_m)}, \theta^{(1)}, .., \theta^{(n_u)}\right) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} \left(\left(\theta^{(j)}\right)^T x^{(i)} - y^{(i,j)}\right)^2$$

∑的下标表示，只计算$r(i,j) = 1$，也就是有评分记录的值。n_m是 movie 的数目，n_u是 user 的数目。用 matlab 实现如下：

```
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：

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

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

```

X_grad = (X*Theta'-Y).*R*Theta;

Theta_grad= ((X*Theta'-Y).*R)'*X;
```

8.7. Regularized cost function

正则化后的损失函数如下：

$$J\left(x^{(1)},\ldots,x^{(n_m)},\theta^{(1)},..,\theta^{(n_u)}\right)$$

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

$$+\left(\frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}\left(x_k^{(i)}\right)^2\right)$$

取偏导数：

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

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

用 matlab 实现如下：

```
J = sum(sum((X*Theta'-Y).*(X*Theta'-Y).*R))/2 + lambda/2*(sum(sum(Theta.*Theta)) + sum(sum(X.*X)));

X_grad = (X*Theta'-Y).*R*Theta + lambda*X;

Theta_grad= ((X*Theta'-Y).*R)'*X + lambda*Theta;
```