**EX6 - Anomaly Detection**

%% ================== Part 1: Load Example Dataset ===================

load('ex8data1.mat');

plot(X(:, 1), X(:, 2), 'bx');

axis([0 30 0 30]);

xlabel('Latency (ms)');

ylabel('Throughput (mb/s)');

%% ================== Part 2: Estimate the dataset statistics ===========

[mu sigma2] = estimateGaussian(X);

p = multivariateGaussian(X, mu, sigma2);

visualizeFit(X, mu, sigma2);

xlabel('Latency (ms)');

ylabel('Throughput (mb/s)');

%% ================== Part 3: Find Outliers ===================

pval = multivariateGaussian(Xval, mu, sigma2);

[epsilon F1] = selectThreshold(yval, pval);

fprintf('Best epsilon found using cross-validation: %e\n', epsilon);

fprintf('Best F1 on Cross Validation Set: %f\n', F1);

outliers = find(p < epsilon);

hold on

plot(X(outliers, 1), X(outliers, 2), 'ro', 'LineWidth', 2, 'MarkerSize', 10);

hold off

%% ================== Part 4: Multidimensional Outliers ===================

load('ex8data2.mat');

[mu sigma2] = estimateGaussian(X);

p = multivariateGaussian(X, mu, sigma2);

pval = multivariateGaussian(Xval, mu, sigma2);

[epsilon F1] = selectThreshold(yval, pval);

fprintf('Best epsilon found using cross-validation: %e\n', epsilon);

fprintf('Best F1 on Cross Validation Set: %f\n', F1);

fprintf('# Outliers found: %d\n', sum(p < epsilon));

function [mu sigma2] = **estimateGaussian**(X)

%ESTIMATEGAUSSIAN This function estimates the parameters of a

%Gaussian distribution using the data in X

mu = mean(X)';

sigma2 = mean((bsxfun(@minus, X, mu')).^2)';

end

function p = **multivariateGaussian**(X, mu, Sigma2)

%MULTIVARIATEGAUSSIAN Computes the probability density function of the

%multivariate gaussian distribution.

k = length(mu);

if (size(Sigma2, 2) == 1) || (size(Sigma2, 1) == 1)

Sigma2 = diag(Sigma2);

end

X = bsxfun(@minus, X, mu(:)');

p = (2 \* pi) ^ (- k / 2) \* det(Sigma2) ^ (-0.5) \* ...

exp(-0.5 \* sum(bsxfun(@times, X \* pinv(Sigma2), X), 2));

end

function **visualizeFit**(X, mu, sigma2)

%VISUALIZEFIT Visualize the dataset and its estimated distribution.

[X1,X2] = meshgrid(0:.5:35);

Z = multivariateGaussian([X1(:) X2(:)],mu,sigma2);

Z = reshape(Z,size(X1));

plot(X(:, 1), X(:, 2),'bx');

hold on;

if (sum(isinf(Z)) == 0)

contour(X1, X2, Z, 10.^(-20:3:0)');

end

hold off;

end

function [bestEpsilon bestF1] = **selectThreshold**(yval, pval)

%SELECTTHRESHOLD Find the best threshold (epsilon) to use for selecting

%outliers

bestEpsilon = 0;

bestF1 = 0;

stepsize = (max(pval) - min(pval)) / 1000;

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

predicted\_anomalies = (pval<epsilon);

tp = sum(predicted\_anomalies==yval & yval==1);

fp = sum(predicted\_anomalies==1 & yval==0);

fn = sum(predicted\_anomalies==0 & yval==1);

prec = tp/(tp+fp);

rec = tp/(tp+fn);

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

if F1 > bestF1

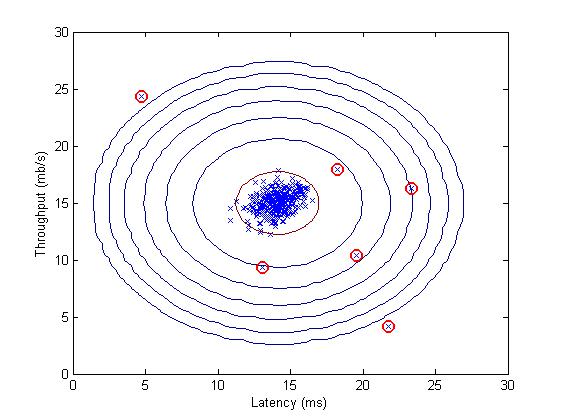
bestF1 = F1;

bestEpsilon = epsilon;

end

end

end



**EX6 - Recommender Systems**

%% ============== Part 1: Entering ratings for a New User ===============

movieList = loadMovieList(); %Loads names of movies corresponding to index

% Initialize my ratings

my\_ratings = zeros(1682, 1);

my\_ratings(1) = 4; my\_ratings(98) = 2; my\_ratings(7) = 3;

my\_ratings(12)= 5; my\_ratings(54) = 4; my\_ratings(64)= 5; my\_ratings(66)= 3;

my\_ratings(69) = 5; my\_ratings(183) = 4; my\_ratings(226) = 5; my\_ratings(355)= 5;

%% ================== Part 2: Learning Movie Ratings ====================

load('ex8\_movies.mat');

Y = [my\_ratings Y];

R = [(my\_ratings ~= 0) R];

[Ynorm, Ymean] = normalizeRatings(Y, R);

num\_users = size(Y, 2); num\_movies = size(Y, 1); num\_features = 10;

X = randn(num\_movies, num\_features);

Theta = randn(num\_users, num\_features);

initial\_parameters = [X(:); Theta(:)];

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

lambda = 10;

theta = fmincg (@(t)(cofiCostFunc(t, Y, R, num\_users, num\_movies, ...

num\_features, lambda)), ...

initial\_parameters, options);

X = reshape(theta(1:num\_movies\*num\_features), num\_movies, num\_features);

Theta = reshape(theta(num\_movies\*num\_features+1:end), ...

num\_users, num\_features);

fprintf('Recommender system learning completed.\n');

%% ================== Part 3: Recommendation for New User ====================

p = X \* Theta';

my\_predictions = p(:,1) + Ymean;

movieList = loadMovieList();

[r, ix] = sort(my\_predictions, 'descend');

fprintf('\nTop recommendations for you:\n');

for i=1:10

j = ix(i);

fprintf('Predicting rating %.1f for movie %s\n', my\_predictions(j), ...

movieList{j});

end

function [Ynorm, Ymean] = **normalizeRatings**(Y, R)

%NORMALIZERATINGS Preprocess data by subtracting mean rating for every

%movie (every row)

[m, n] = size(Y);

Ymean = zeros(m, 1);

Ynorm = zeros(size(Y));

for i = 1:m

idx = find(R(i, :) == 1);

Ymean(i) = mean(Y(i, idx));

Ynorm(i, idx) = Y(i, idx) - Ymean(i);

end

end

function [J, grad] = **cofiCostFunc**(params, Y, R, num\_users, num\_movies, ...

num\_features, lambda)

%COFICOSTFUNC Collaborative filtering cost function

X = reshape(params(1:num\_movies\*num\_features), num\_movies, num\_features);

Theta = reshape(params(num\_movies\*num\_features+1:end), ...

num\_users, num\_features);

J = 0.5\*sum(sum(((X\*Theta'-Y).\*R).^2)) + .../

lambda/2\*(sum(sum(Theta.^2))+sum(sum(X.^2)));

diff = X\*Theta'-Y;

X\_grad = (diff.\*R)\*Theta + lambda\*X;

Theta\_grad = (diff.\*R)'\*X + lambda\*Theta;

grad = [X\_grad(:); Theta\_grad(:)];

end