```
function [mu sigma2] = estimateGaussian(X)
%ESTIMATEGAUSSIAN This function estimates the parameters of a
%Gaussian distribution using the data in X
%   [mu sigma2] = estimateGaussian(X),
%   The input X is the dataset with each n-dimensional data point in one row
%   The output is an n-dimensional vector mu, the mean of the data set
%   and the variances sigma^2, an n x 1 vector
%

% Useful variables
[m, n] = size(X);

% You should return these values correctly
mu = zeros(n, 1);
sigma2 = zeros(n, 1);

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the mean of the data and the variances
%               In particular, mu(i) should contain the mean of
%               the data for the i-th feature and sigma2(i)
%               should contain variance of the i-th feature.
%
```

```
mu = ((1 / m) * sum(X))';
sigma2 = ((1 / m) * sum((X - mu') .^2))';
```

```
% =============================================================


end
```

```
function [bestEpsilon bestF1] = selectThreshold(yval, pval)
%SELECTTHRESHOLD Find the best threshold (epsilon) to use for selecting
%outliers
%   [bestEpsilon bestF1] = SELECTTHRESHOLD(yval, pval) finds the best
%   threshold to use for selecting outliers based on the results from a
%   validation set (pval) and the ground truth (yval).
%

bestEpsilon = 0;
bestF1 = 0;
F1 = 0;

stepsize = (max(pval) - min(pval)) / 1000;
for epsilon = min(pval):stepsize:max(pval)

    % ====================== YOUR CODE HERE ======================
    % Instructions: Compute the F1 score of choosing epsilon as the
    %               threshold and place the value in F1. The code at the
    %               end of the loop will compare the F1 score for this
    %               choice of epsilon and set it to be the best epsilon if
    %               it is better than the current choice of epsilon.
    %
    % Note: You can use predictions = (pval < epsilon) to get a binary vector

    %       of 0's and 1's of the outlier predictions
predictions = (pval < epsilon);
tp = sum((predictions == 1) & (yval == 1));
fp = sum((predictions == 1) & (yval == 0));
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

end
```

```
function [J, grad] = cofiCostFunc(params, Y, R, num_users, num_movies, ...
                                  num_features, lambda)
%COFICOSTFUNC Collaborative filtering cost function
%   [J, grad] = COFICOSTFUNC(params, Y, R, num_users, num_movies, ...
%   num_features, lambda) returns the cost and gradient for the
%   collaborative filtering problem.
%

% Unfold the U and W matrices from params
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);


% You need to return the following values correctly
J = 0;
X_grad = zeros(size(X));
Theta_grad = zeros(size(Theta));

% ====================== YOUR CODE HERE ======================
% Instructions: Compute the cost function and gradient for collaborative
%               filtering. Concretely, you should first implement the cost
%               function (without regularization) and make sure it is
```

```
%               matches our costs. After that, you should implement the
%               gradient and use the checkCostFunction routine to check
%               that the gradient is correct. Finally, you should implement
%               regularization.
%
% Notes: X - num_movies  x num_features matrix of movie features
%        Theta - num_users  x num_features matrix of user features
%        Y - num_movies x num_users matrix of user ratings of movies
%        R - num_movies x num_users matrix, where R(i, j) = 1 if the
%            i-th movie was rated by the j-th user
%
% You should set the following variables correctly:
%
%        X_grad - num_movies x num_features matrix, containing the
%                 partial derivatives w.r.t. to each element of X
%        Theta_grad - num_users x num_features matrix, containing the
%                     partial derivatives w.r.t. to each element of Theta
%
J_without_regularization = (1 / 2) * sum(sum(R .* (X * Theta' - Y) .^ 2));
rated_error = (X * Theta' - Y) .* R;
X_grad_without_regularization = rated_error * Theta;
Theta_grad_without_regularization = rated_error' * X;
J = J_without_regularization + (lambda / 2) * sum(sum(Theta .^2)) + (lambda / 2) * sum(sum(X .^2));
```

```
X_grad = X_grad_without_regularization + lambda * X;
Theta_grad = Theta_grad_without_regularization + lambda * Theta;
```

```
% =============================================================

grad = [X_grad(:); Theta_grad(:)];

end
```