Programming Exercise 8
Anomaly Detection and Recommender Systems

Machine Learning (Stanford University)

September 2020

This exercise implements:

(1) Anomaly Detection algorithm and apply it to detect failing servers on a network
(2) Collaborative Filtering to build a recommender system for movies

1 Anomaly Detection

Instruction:
In this exercise, you will implement an anomaly detection algorithm to detect anomalous behavior in server computers.

Dataset
(2) Two Features from each server

1. throughput (mb/s)
2. latency (ms)

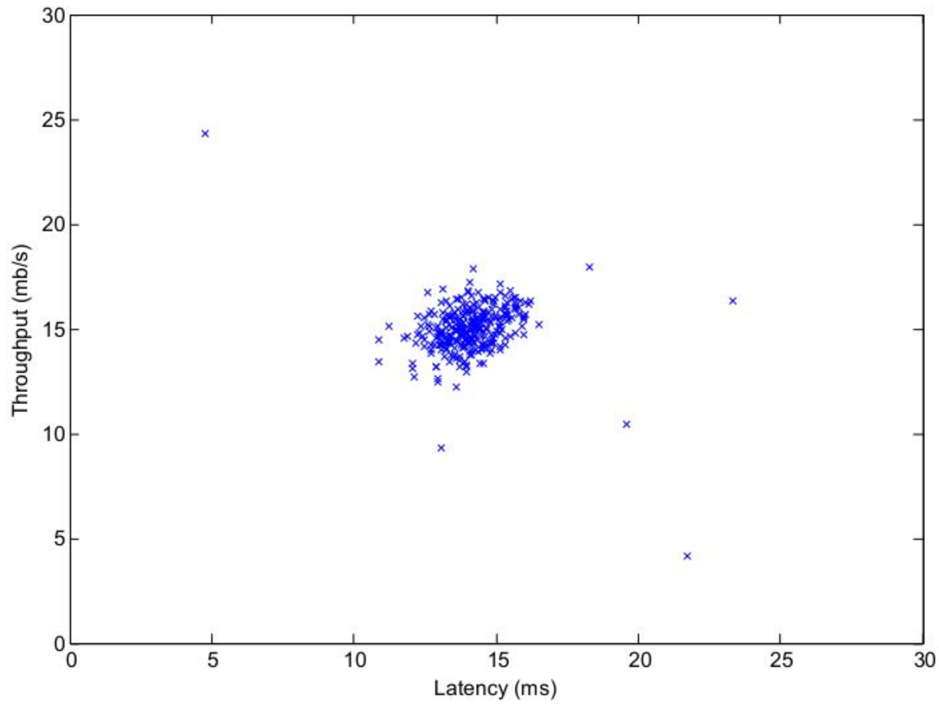
m = 307 examples where unlabeled dataset $\{x^{(1)}, \dots, x^{(m)}\}$

You suspect that the vast majority of these examples are “normal” (non-anomalous) examples of the servers operating normally, but there might also be some examples of servers acting anomalously within this dataset.

Use a Gaussian model to detect anomalous examples in your dataset.

(1) 2D dataset
    - allow you to visualize what the algorithm is doing
    1. fit a Gaussian distribution
    2. find values that have very low probability ~ an anomaly
(2) Use (1) and apply it to a larger dataset with many dimensions

Visualizing the first dataset



1.1 Gaussian distribution

Instruction:
To perform anomaly detection, you will first need to fit a model to the data's distribution.

Given a training set $\{x^{(1)}, ..., x^{(m)}\}$ (where $x^{(i)} \in R^n$), you want to estimate the Gaussian distribution for each of the features x i . For each feature i = 1 . . . n, you need to find parameters μ_i and σ_i^2 that fit the data in the i-th dimension $\{x_i^{(1)}, ..., x_i^{(m)}\}$ (the i-th dimension of each example).

Gaussian Distribution

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

1.2 Estimating parameters for a Gaussian

Instruction:

You can estimate the parameters, (μ_i , σ_i^2), of the i-th feature by using the following equations.
To estimate the mean, you will use:

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

for the variance you will use:

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

Implementation:

estimateGaussian.m

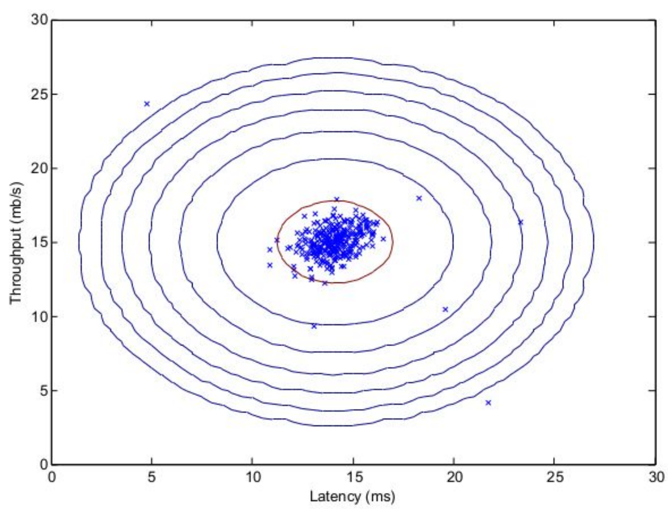
```
function [mu sigma2] = estimateGaussian(X)
%ESTIMATEGAUSSIAN This function estimates the parameters of a
%Gaussian distribution using the data in X
%   [mu sigma2] = estimateGaussian(X),
%   The input X is the dataset with each n-dimensional data point in one row
%   The output is an n-dimensional 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);

mu = (1/m)*sum(X(:,1:n))'
disp(X)
sigma2 = (1/m)*sum((X(:,1:n) - mu').^2)'

end
```



The Gaussian distribution contours of the distribution fit to the dataset

Remarks:
Now that you have estimated the Gaussian parameters, you can investigate which examples have a very high probability given this distribution and which examples have a very low probability. The low probability examples are more likely to be the anomalies in our dataset.

1.3 Selecting the threshold, ε

Instruction:
One way to determine which examples are anomalies is to select a threshold based on a cross validation set. In this part of the exercise, you will implement an algorithm to select the threshold ε using the F1 score on a cross validation set.

You should now complete the code in selectThreshold.m. For this, we will use a cross validation set $\{(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(mcv)}, y_{cv}^{(mcv)})\}$, where the label y = 1 corresponds to an anomalous example, and y = 0 corresponds to a normal example. For each cross validation example, we will compute $p(x_{cv}^{(1)})$. The vector of all of these probabilities $p(x_{cv}^{(1)}), \ldots, p(x_{cv}^{(mcv)})$) is passed to selectThreshold.m in the vector pval. The corresponding labels $(y_{cv}^{(1)}, \ldots, y_{cv}^{(mcv)})$ is passed to the same function in the vector yval.

The function selectThreshold.m should return two values; the first is the selected threshold ε. If an example x has a low probability $p(x) < ε$, then it is considered to be an anomaly. The function should also return the F1 score, which tells you how well you're doing on finding the ground truth anomalies given a certain threshold. For many different values of ε, you will compute the resulting F1 score by computing how many examples the current threshold classifies correctly and incorrectly.

The F_1 score is computed using precision ($prec$) and recall (rec):

$$F_1 = \frac{2 \cdot prec \cdot rec}{prec + rec}, \tag{3}$$

You compute precision and recall by:

$$prec = \frac{tp}{tp + fp} \tag{4}$$

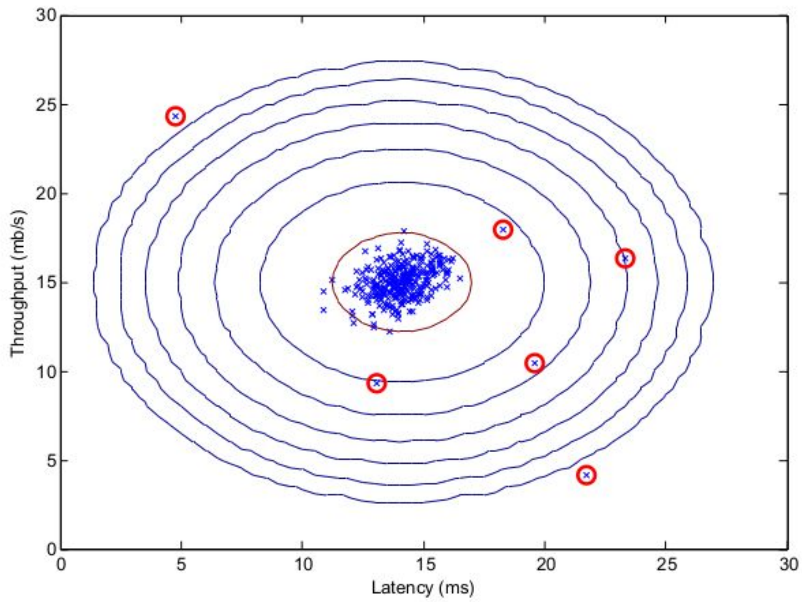
$$rec = \frac{tp}{tp + fn}, \tag{5}$$

where

- tp is the number of true positives: the ground truth label says it's an anomaly and our algorithm correctly classified it as an anomaly.
- fp is the number of false positives: the ground truth label says it's not an anomaly, but our algorithm incorrectly classified it as an anomaly.
- fn is the number of false negatives: the ground truth label says it's an anomaly, but our algorithm incorrectly classified it as not being anomalous.

In the provided code selectThreshold.m, there is already a loop that will try many different values of ε and select the best ε based on the F 1 score. You should now complete the code in selectThreshold.m. You can implement the computation of the F1 score using a for-loop over all the cross validation examples (to compute the values tp, f p, f n). You should see a value for epsilon of about 8.99e-05.

Implementation Note: In order to compute tp, fp and fn, you may be able to use a vectorized implementation rather than loop over all the examples. This can be implemented by Octave/MATLAB's equality test between a vector and a single number. If you have several binary values in an n-dimensional binary vector $v \in \{0,1\}^n$, you can find out how many values in this vector are 0 by using: `sum(v == 0)`. You can also apply a logical `and` operator to such binary vectors. For instance, let `cvPredictions` be a binary vector of the size of your number of cross validation set, where the i-th element is 1 if your algorithm considers $x_{cv}^{(i)}$ an anomaly, and 0 otherwise. You can then, for example, compute the number of false positives using: `fp = sum((cvPredictions == 1) & (yval == 0))`.



The classified anomalies

Implementation:

selectThreshold.m

```
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)
  cvPredictions = (pval < epsilon);

  tp = sum((cvPredictions == 1) & (yval == 1));
  fp = sum((cvPredictions == 1) & (yval == 0));
  fn = sum((cvPredictions == 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
```

1.4 High dimensional dataset

Instructions:
The last part of the script ex8.m will run the anomaly detection algorithm you implemented on a more realistic and much harder dataset. In this dataset, each example is described by 11 features, capturing many more properties of your compute servers.

The script will use your code to estimate the Gaussian parameters (μ_i and σ_i^2), evaluate the probabilities for both the training data X from which you estimated the Gaussian parameters, and do so for the cross-validation set Xval. Finally, it will use selectThreshold to find the best threshold ε. You should see a value epsilon of about 1.38e-18, and 117 anomalies found.

Result run:

```
fBest epsilon found using cross-validation: 1.377229e-18
Best F1 on Cross Validation Set:  0.615385
   (you should see a value epsilon of about 1.38e-18)
   (you should see a Best F1 value of 0.615385)
# Outliers found: 117
```

2 Recommender Systems

Instruction:
In this part of the exercise, you will implement the collaborative filtering learning algorithm and apply it to a dataset of movie ratings.

2.1 Movie ratings dataset

Dataset:
- consists of ratings on a scale of 1 to 5
- n_u = 943 users
- n_m = 1682 movies

- Matrix Y
    - (a num movies × num users matrix) stores the ratings $y^{(i,j)}$ (from 1 to 5)
- Matrix R
    - binary-valued indicator matrix
    - R(i, j) = 1 if user j gave a rating to movie i
    - R(i, j) = 0 otherwise

The objective of collaborative filtering is to predict movie ratings for the movies that users have not yet rated, that is, the entries with R(i, j) = 0. This will allow us to recommend the movies with the highest predicted ratings to the user.

- Matrix X and Theta

$$\mathtt{X} = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(n_m)})^T — \end{bmatrix}, \quad \mathtt{Theta} = \begin{bmatrix} — (\theta^{(1)})^T — \\ — (\theta^{(2)})^T — \\ \vdots \\ — (\theta^{(n_u)})^T — \end{bmatrix}$$

  - i-th row of X corresponds to the feature vector x (i) for the i-th movie
  - the j-th row of Theta corresponds to one parameter vector $\theta^{(j)}$, for the j-th user
  - Both $x^{(i)}$ and $\theta^{(j)}$ are n-dimensional vectors
  - For the purposes of this exercise, you will use n = 100, and therefore, $x^{(i)} \in$ R 100 and $\theta^{(j)} \in$ R 100
  - Correspondingly, X is a n m × 100 matrix and Theta is a n u × 100 matrix

2.2 Collaborative filtering learning algorithm

Instruction:
Now, you will start implementing the collaborative filtering learning algorithm. You will start by implementing the cost function (without regularization).

2.2.1 Collaborative filtering cost function

The collaborative filtering cost function (without regularization) is given by

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

Implementation Note: We strongly encourage you to use a vectorized implementation to compute J, since it will later by called many times by the optimization package `fmincg`. As usual, it might be easiest to first write a non-vectorized implementation (to make sure you have the right answer), and the modify it to become a vectorized implementation (checking that the vectorization steps don't change your algorithm's output). To come up with a vectorized implementation, the following tip might be helpful: You can use the `R` matrix to set selected entries to 0. For example, `R .* M` will do an element-wise multiplication between `M` and `R`; since `R` only has elements with values either 0 or 1, this has the effect of setting the elements of `M` to 0 only when the corresponding value in `R` is 0. Hence, `sum(sum(R.*M))` is the sum of all the elements of `M` for which the corresponding element in `R` equals 1.

Implementation:

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

2.2.2 Collaborative filtering gradient

Instruction:

Now, you should implement the gradient (without regularization). Specifically, you should complete the code in cofiCostFunc.m to return the variables X grad and Theta grad. Note that X grad should be a matrix of the same size as X and similarly, Theta grad is a matrix of the same size as Theta. The gradients of the cost function is given by:

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

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

> **Implementation Note:** You can get full credit for this assignment without using a vectorized implementation, but your code will run much more slowly (a small number of hours), and so we recommend that you try to vectorize your implementation.
>
> To get started, you can implement the gradient with a for-loop over movies (for computing $\frac{\partial J}{\partial x_k^{(i)}}$) and a for-loop over users (for computing $\frac{\partial J}{\partial \theta_k^{(j)}}$). When you first implement the gradient, you might start with an unvectorized version, by implementing another inner for-loop that computes each element in the summation. After you have completed the gradient computation this way, you should try to vectorize your implementation (vectorize the inner for-loops), so that you're left with only two for-loops (one for looping over movies to compute $\frac{\partial J}{\partial x_k^{(i)}}$ for each movie, and one for looping over users to compute $\frac{\partial J}{\partial \theta_k^{(j)}}$ for each user).

Implementation:

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

2.2.3 Regularized cost function

Instruction:

The cost function for collaborative filtering with regularization is given by

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

You should now add regularization to your original computations of the cost function, J.

Implementation:

```
J = J + (lambda/2)*(sum(sum(Theta.^2))) + (lambda/2)*(sum(sum(X.^2)));
```

2.2.4 Regularized gradient

Instruction:
Now that you have implemented the regularized cost function, you should proceed to implement regularization for the gradient. You should add to your implementation in cofiCostFunc.m to return the regularized gradient by adding the contributions from the regularization terms. Note that the gradients for the regularized cost function is given by:

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

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

Implementation:

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

Whole cofiFunc.m

cofiCostFunc.m

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

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

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

X_grad = X_grad + lambda*X;
Theta_grad = Theta_grad + lambda*Theta;

% =============================================================

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

end
```

3 Scores

```
==                                   Part Name |     Score | Feedback
==                                   --------- |     ----- | --------
==             Estimate Gaussian Parameters |  15 /  15 | Nice work!
==                         Select Threshold |  15 /  15 | Nice work!
==             Collaborative Filtering Cost |  20 /  20 | Nice work!
==         Collaborative Filtering Gradient |  30 /  30 | Nice work!
==                         Regularized Cost |  10 /  10 | Nice work!
==                     Regularized Gradient |  10 /  10 | Nice work!
==                                   --------------------------------
==                                             | 100 / 100 |
```