Programming Exercise 8: Anomaly Detection and Recommender Systems

Machine Learning

Introduction

In this exercise, you will implement the anomaly detection algorithm and apply it to detect failing servers on a network. In the second part, you will use collaborative filtering to build a recommender system for movies. Before starting on the programming exercise, we strongly recommend watching the video lectures and completing the review questions for the associated topics.

To get started with the exercise, you will need to download the starter code and unzip its contents to the directory where you wish to complete the exercise. If needed, use the `cd` command in Octave/MATLAB to change to this directory before starting this exercise.

You can also find instructions for installing Octave/MATLAB in the "Environment Setup Instructions" of the course website.

Files included in this exercise

`ex8.m` - Octave/MATLAB script for first part of exercise
`ex8_cofi.m` - Octave/MATLAB script for second part of exercise
`ex8data1.mat` - First example Dataset for anomaly detection
`ex8data2.mat` - Second example Dataset for anomaly detection
`ex8_movies.mat` - Movie Review Dataset
`ex8_movieParams.mat` - Parameters provided for debugging
`multivariateGaussian.m` - Computes the probability density function for a Gaussian distribution
`visualizeFit.m` - 2D plot of a Gaussian distribution and a dataset
`checkCostFunction.m` - Gradient checking for collaborative filtering
`computeNumericalGradient.m` - Numerically compute gradients

`fmincg.m` - Function minimization routine (similar to fminunc)
`loadMovieList.m` - Loads the list of movies into a cell-array
`movie_ids.txt` - List of movies
`normalizeRatings.m` - Mean normalization for collaborative filtering
`submit.m` - Submission script that sends your solutions to our servers
[⋆] `estimateGaussian.m` - Estimate the parameters of a Gaussian distribution with a diagonal covariance matrix
[⋆] `selectThreshold.m` - Find a threshold for anomaly detection
[⋆] `cofiCostFunc.m` - Implement the cost function for collaborative filtering

⋆ indicates files you will need to complete

Throughout the first part of the exercise (anomaly detection) you will be using the script `ex8.m`. For the second part of collaborative filtering, you will use `ex8_cofi.m`. These scripts set up the dataset for the problems and make calls to functions that you will write. You are only required to modify functions in other files, by following the instructions in this assignment.

Where to get help

The exercises in this course use Octave[1] or MATLAB, a high-level programming language well-suited for numerical computations. If you do not have Octave or MATLAB installed, please refer to the installation instructions in the "Environment Setup Instructions" of the course website.

At the Octave/MATLAB command line, typing `help` followed by a function name displays documentation for a built-in function. For example, `help plot` will bring up help information for plotting. Further documentation for Octave functions can be found at the Octave documentation pages. MATLAB documentation can be found at the MATLAB documentation pages.

We also strongly encourage using the online **Discussions** to discuss exercises with other students. However, do not look at any source code written by others or share your source code with others.

[1] Octave is a free alternative to MATLAB. For the programming exercises, you are free to use either Octave or MATLAB.

1 Anomaly detection

In this exercise, you will implement an anomaly detection algorithm to detect anomalous behavior in server computers. The features measure the throughput (mb/s) and latency (ms) of response of each server. While your servers were operating, you collected $m = 307$ examples of how they were behaving, and thus have an unlabeled dataset $\{x^{(1)}, \ldots, x^{(m)}\}$. 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.

You will use a Gaussian model to detect anomalous examples in your dataset. You will first start on a 2D dataset that will allow you to visualize what the algorithm is doing. On that dataset you will fit a Gaussian distribution and then find values that have very low probability and hence can be considered anomalies. After that, you will apply the anomaly detection algorithm to a larger dataset with many dimensions. You will be using `ex8.m` for this part of the exercise.

The first part of `ex8.m` will visualize the dataset as shown in Figure 1.

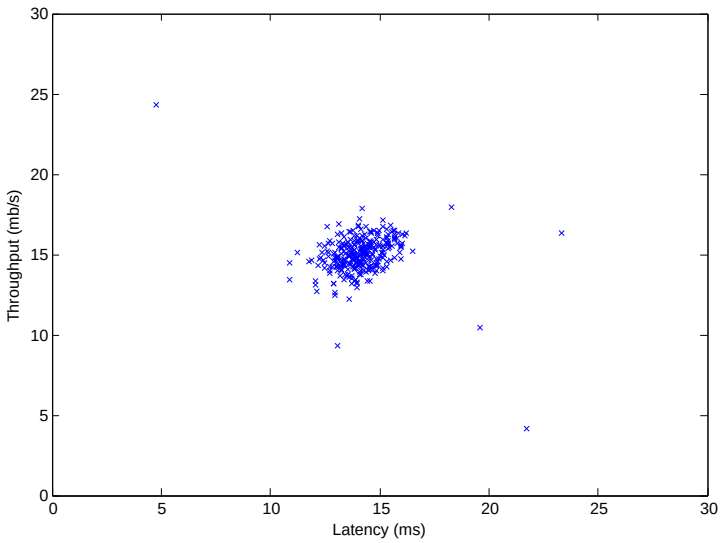

Figure 1: The first dataset.

1.1 Gaussian distribution

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 $x^{(i)} \in \mathbb{R}^n$), you want to estimate the Gaussian distribution for each of the features x_i. For each feature $i = 1 \ldots n$, 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).

The Gaussian distribution is given by

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

where μ is the mean and σ^2 controls the variance.

1.2 Estimating parameters for a Gaussian

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:

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

and for the variance you will use:

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

Your task is to complete the code in `estimateGaussian.m`. This function takes as input the data matrix `X` and should output an n-dimension vector `mu` that holds the mean of all the n features and another n-dimension vector `sigma2` that holds the variances of all the features. You can implement this using a for-loop over every feature and every training example (though a vectorized implementation might be more efficient; feel free to use a vectorized implementation if you prefer). Note that in Octave/MATLAB, the `var` function will (by default) use $\frac{1}{m-1}$, instead of $\frac{1}{m}$, when computing σ_i^2.

Once you have completed the code in `estimateGaussian.m`, the next part of `ex8.m` will visualize the contours of the fitted Gaussian distribution. You should get a plot similar to Figure 2. From your plot, you can see that most of the examples are in the region with the highest probability, while the anomalous examples are in the regions with lower probabilities.

You should now submit your solutions.

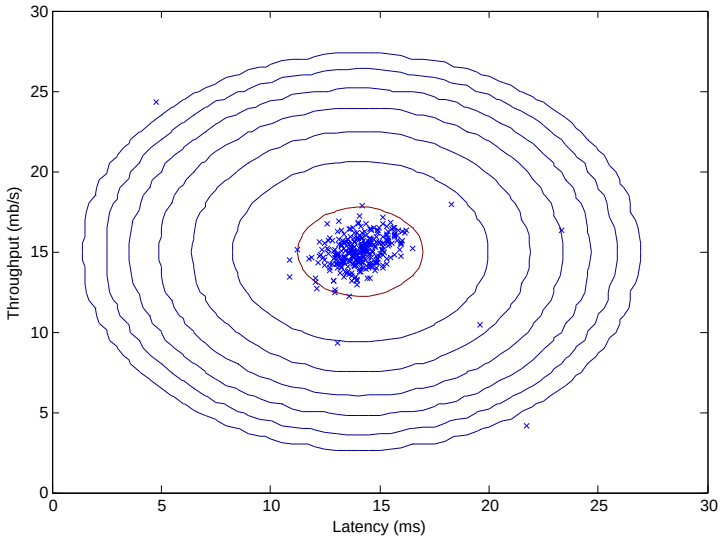

Figure 2: The Gaussian distribution contours of the distribution fit to the dataset.

1.3 Selecting the threshold, ε

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. 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 F_1 score on a cross validation set.

You should now complete the code in `selectThreshold.m`. For this, we will use a cross validation set $\{(x_{\text{cv}}^{(1)}, y_{\text{cv}}^{(1)}), \ldots, (x_{\text{cv}}^{(m_{\text{cv}})}, y_{\text{cv}}^{(m_{\text{cv}})})\}$, 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 $p(x_{\text{cv}}^{(i)})$. The vector of all of these probabilities $p(x_{\text{cv}}^{(1)}), \ldots, p(x_{\text{cv}}^{(m_{\text{cv}})})$ is passed to `selectThreshold.m` in the vector `pval`. The corresponding labels $y_{\text{cv}}^{(1)}, \ldots, y_{\text{cv}}^{(m_{\text{cv}})}$ 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 $p(x) < \varepsilon$, then it is considered to be an anomaly. The function should also return the F_1 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 F_1 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, fp, fn). 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 $x_{\text{cv}}^{(i)}$ an anomaly, and 0 otherwise. You can then, for example, compute the number of false positives using: `fp = sum((cvPredictions == 1) & (yval == 0))`.

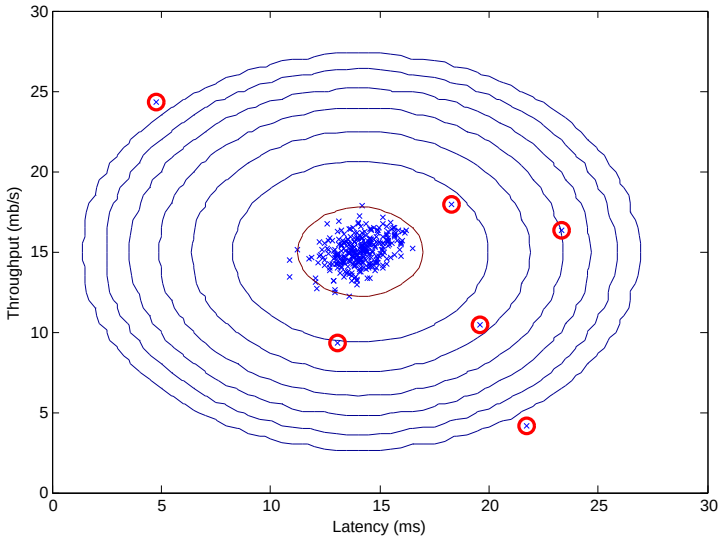

Figure 3: The classified anomalies.

Once you have completed the code in `selectThreshold.m`, the next step in `ex8.m` will run your anomaly detection code and circle the anomalies in the plot (Figure 3).

You should now submit your solutions.

1.4 High dimensional dataset

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 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.

---

2 Recommender Systems

In this part of the exercise, you will implement the collaborative filtering learning algorithm and apply it to a dataset of movie ratings.[2] This dataset consists of ratings on a scale of 1 to 5. The dataset has $n_u = 943$ users, and $n_m = 1682$ movies. For this part of the exercise, you will be working with the script `ex8_cofi.m`.

In the next parts of this exercise, you will implement the function `cofiCostFunc.m` that computes the collaborative fitlering objective function and gradient. After implementing the cost function and gradient, you will use `fmincg.m` to learn the parameters for collaborative filtering.

2.1 Movie ratings dataset

The first part of the script `ex8_cofi.m` will load the dataset `ex8_movies.mat`, providing the variables `Y` and `R` in your Octave/MATLAB environment.

The matrix Y (a `num_movies` $\times$ `num_users` matrix) stores the ratings $y^{(i,j)}$ (from 1 to 5). The matrix R is an binary-valued indicator matrix, where $R(i,j) = 1$ if user j gave a rating to movie i, and $R(i,j) = 0$ 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.

To help you understand the matrix `Y`, the script `ex8_cofi.m` will compute the average movie rating for the first movie (Toy Story) and output the average rating to the screen.

Throughout this part of the exercise, you will also be working with the matrices, `X` and `Theta`:

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

The i-th row of `X` corresponds to the feature vector $x^{(i)}$ for the i-th movie, and 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, $x^{(i)} \in \mathbb{R}^{100}$ and $\theta^{(j)} \in \mathbb{R}^{100}$. Correspondingly, `X` is a $n_m \times 100$ matrix and `Theta` is a $n_u \times 100$ matrix.

[2] MovieLens 100k Dataset from GroupLens Research.

2.2 Collaborative filtering learning algorithm

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

The collaborative filtering algorithm in the setting of movie recommendations considers a set of n-dimensional parameter vectors $x^{(1)}, ..., x^{(n_m)}$ and $\theta^{(1)}, ..., \theta^{(n_u)}$, where the model predicts the rating for movie i by user j as $y^{(i,j)} = (\theta^{(j)})^T x^{(i)}$. Given a dataset that consists of a set of ratings produced by some users on some movies, you wish to learn the parameter vectors $x^{(1)}, ..., x^{(n_m)}, \theta^{(1)}, ..., \theta^{(n_u)}$ that produce the best fit (minimizes the squared error).

You will complete the code in `cofiCostFunc.m` to compute the cost function and gradient for collaborative filtering. Note that the parameters to the function (i.e., the values that you are trying to learn) are `X` and `Theta`. In order to use an off-the-shelf minimizer such as `fmincg`, the cost function has been set up to unroll the parameters into a single vector `params`. You had previously used the same vector unrolling method in the neural networks programming exercise.

2.2.1 Collaborative filtering cost function

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

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

You should now modify `cofiCostFunc.m` to return this cost in the variable `J`. Note that you should be accumulating the cost for user j and movie i only if $R(i, j) = 1$.

After you have completed the function, the script `ex8_cofi.m` will run your cost function. You should expect to see an output of 22.22.

You should now submit your solutions.

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.

2.2.2 Collaborative filtering gradient

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:

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

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

Note that the function returns the gradient for both sets of variables by unrolling them into a single vector. After you have completed the code to compute the gradients, the script `ex8_cofi.m` will run a gradient check (`checkCostFunction`) to numerically check the implementation of your gradients.[3] If your implementation is correct, you should find that the analytical and numerical gradients match up closely.

You should now submit your solutions.

[3] This is similar to the numerical check that you used in the neural networks exercise.

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 Tip: To perform the vectorization, you might find this helpful: You should come up with a way to compute all the derivatives associated with $x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}$ (i.e., the derivative terms associated with the feature vector $x^{(i)}$) at the same time. Let us define the derivatives for the feature vector of the i-th movie as:

$$(\texttt{X}_{\texttt{grad}}(\texttt{i},:))^T = \begin{bmatrix} \frac{\partial J}{\partial x_1^{(i)}} \\ \frac{\partial J}{\partial x_2^{(i)}} \\ \vdots \\ \frac{\partial J}{\partial x_n^{(i)}} \end{bmatrix} = \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta^{(j)}$$

To vectorize the above expression, you can start by indexing into `Theta` and `Y` to select only the elements of interests (that is, those with $r(i,j) = 1$). Intuitively, when you consider the features for the i-th movie, you only need to be concern about the users who had given ratings to the movie, and this allows you to remove all the other users from `Theta` and `Y`.

Concretely, you can set `idx = find(R(i, :)==1)` to be a list of all the users that have rated movie i. This will allow you to create the temporary matrices $\texttt{Theta}_{\texttt{temp}} = \texttt{Theta}(\texttt{idx},:)$ and $\texttt{Y}_{\texttt{temp}} = \texttt{Y}(\texttt{i},\texttt{idx})$ that index into `Theta` and `Y` to give you only the set of users which have rated the i-th movie. This will allow you to write the derivatives as:

$$\texttt{X}_{\texttt{grad}}(\texttt{i},:) = (\texttt{X}(\texttt{i},:) * \texttt{Theta}_{\texttt{temp}}^{\texttt{T}} - \texttt{Y}_{\texttt{temp}}) * \texttt{Theta}_{\texttt{temp}}.$$

(Note: The vectorized computation above returns a row-vector instead.)

After you have vectorized the computations of the derivatives with respect to $x^{(i)}$, you should use a similar method to vectorize the derivatives with respect to $\theta^{(j)}$ as well.

2.2.3 Regularized cost function

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. After you are done, the script `ex8_cofi.m` will run your regularized cost function, and you should expect to see a cost of about 31.34.

You should now submit your solutions.

2.2.4 Regularized gradient

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:

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

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

This means that you just need to add $\lambda x^{(i)}$ to the `X_grad(i,:)` variable described earlier, and add $\lambda\theta^{(j)}$ to the `Theta_grad(j,:)` variable described earlier.

After you have completed the code to compute the gradients, the script `ex8_cofi.m` will run another gradient check (`checkCostFunction`) to numerically check the implementation of your gradients.

You should now submit your solutions.

2.3 Learning movie recommendations

After you have finished implementing the collaborative filtering cost function and gradient, you can now start training your algorithm to make movie

recommendations for yourself. In the next part of the `ex8_cofi.m` script, you can enter your own movie preferences, so that later when the algorithm runs, you can get your own movie recommendations! We have filled out some values according to our own preferences, but you should change this according to your own tastes. The list of all movies and their number in the dataset can be found listed in the file `movie_idx.txt`.

2.3.1 Recommendations

```
Top recommendations for you:
Predicting rating 9.0 for movie Titanic (1997)
Predicting rating 8.9 for movie Star Wars (1977)
Predicting rating 8.8 for movie Shawshank Redemption, The (1994)
Predicting rating 8.5 for movie As Good As It Gets (1997)
Predicting rating 8.5 for movie Good Will Hunting (1997)
Predicting rating 8.5 for movie Usual Suspects, The (1995)
Predicting rating 8.5 for movie Schindler's List (1993)
Predicting rating 8.4 for movie Raiders of the Lost Ark (1981)
Predicting rating 8.4 for movie Empire Strikes Back, The (1980)
Predicting rating 8.4 for movie Braveheart (1995)

Original ratings provided:
Rated 4 for Toy Story (1995)
Rated 3 for Twelve Monkeys (1995)
Rated 5 for Usual Suspects, The (1995)
Rated 4 for Outbreak (1995)
Rated 5 for Shawshank Redemption, The (1994)
Rated 3 for While You Were Sleeping (1995)
Rated 5 for Forrest Gump (1994)
Rated 2 for Silence of the Lambs, The (1991)
Rated 4 for Alien (1979)
Rated 5 for Die Hard 2 (1990)
Rated 5 for Sphere (1998)
```

Figure 4: Movie recommendations

After the additional ratings have been added to the dataset, the script will proceed to train the collaborative filtering model. This will learn the parameters `X` and `Theta`. To predict the rating of movie i for user j, you need

to compute $(\theta^{(j)})^T x^{(i)}$. The next part of the script computes the ratings for all the movies and users and displays the movies that it recommends (Figure 4), according to ratings that were entered earlier in the script. Note that you might obtain a different set of the predictions due to different random initializations.

Submission and Grading

After completing various parts of the assignment, be sure to use the `submit` function system to submit your solutions to our servers. The following is a breakdown of how each part of this exercise is scored.

Part	**Submitted File**	**Points**
Estimate Gaussian Parameters	`estimateGuassian.m`	15 points
Select Threshold	`selectThreshold.m`	15 points
Collaborative Filtering Cost	`cofiCostFunc.m`	20 points
Collaborative Filtering Gradient	`cofiCostFunc.m`	30 points
Regularized Cost	`cofiCostFunc.m`	10 points
Gradient with regularization	`cofiCostFunc.m`	10 points
Total Points		100 points

You are allowed to submit your solutions multiple times, and we will take only the highest score into consideration.