**CSC478 Final Project Report**

**Team members: Shan Gao, Dachi Xu, Xingyue Duan**

In this project we used Movie Rating data from [www.movielens.org](http://www.movielens.org). It contains ratings on 1600+ movies by 1000 users. We applied most of KDD processes we learned from course CSC478 including data pre-processing and visualization, K-Nearest-Neighbor search, classification, categorization, clustering and item-based collaborative filtering. Tried to observe relationships inside the datasets and to do predictions and recommendations based on them.

The link to dataset is <http://facweb.cs.depaul.edu/mobasher/classes/ect584/data/movielens.zip>

# Data Set description:

|  |  |
| --- | --- |
| u.data | The full data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of the form: user id | item id | rating | timestamp.  The time stamps are unix seconds since 1/1/1970 UTC |
| u.info | The number of users, items, and ratings in the u data set. |
| u.item | Information about the items (movies); this is a tab separated list of the form:  movie id | movie title | release date | video release date | IMDb URL | unknown | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller | War | Western |  The last 19 fields are the genres, a 1 indicates the movie is of that genre, a 0 indicates it is not; movies can be in several genres at once. The movie ids are the ones used in the u.data data set. |
| u.genre | A list of the genres. |
| u.user | Demographic information about the users; this is a tab separated list of the form:  user id | age | gender | occupation | zip code  The user ids are the ones used in the u.data data set. |
| u.occupation | A list of the occupations. |
| ua.base  ua.test  ub.base  ub.test | The data sets ua.base, ua.test, ub.base, and ub.test split the u data into a training set and a test set with exactly 10 ratings per user in the test set. The sets ua.test and ub.test are disjoint. |
|  |  |

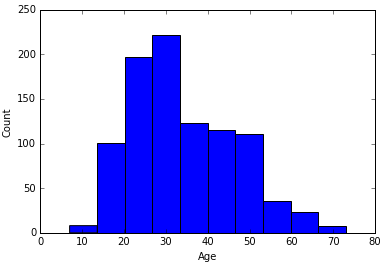
# Data Analysis:

## Section 1. Explore general characteristics of the dataset

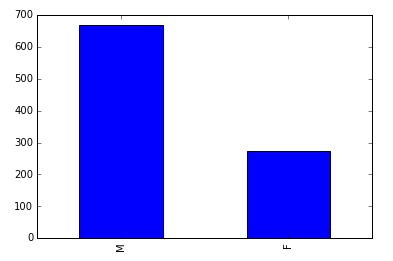
We loaded the data as Pandas data-frame, and explore the general characteristics of the data as a whole. We examined the means, standard deviations, and other statistics associated with the numerical attributes; show the distributions of values associated with categorical attributes. And we use histograms or bar-graphs to plot the distribution of movie and user data.

Observations:

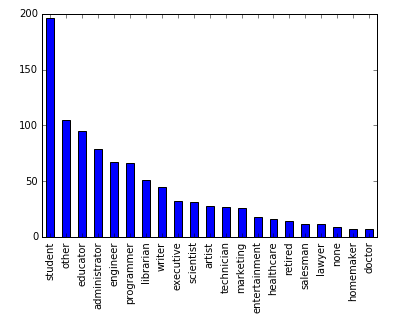
1. The mean age of all users is 34 and most of users are between 20 - 30 years old.



2. Number of male users are twice as many as female users.



3. Most of the users are students.



## Section 2. Correlation analysis

In this part we performed basic correlation analysis among the movie rating and user’s age attributes. Tried to find out any significant positive or negative correlations among them in each different genre. Also we constructed correlation matrix, used Matplotlib library and plotting capabilities of Pandas to create a scatter plot of the two attributes.

Observations:

1. The correlations between "user age" and "rating" across every genre is not strong.

2. Genres with highest correlations are:

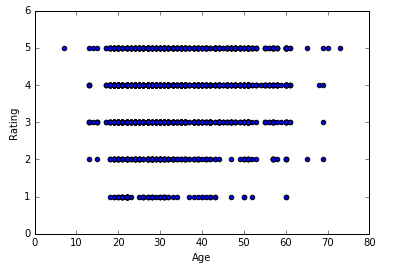
"Fantasy": 0.147

“Western": 0.131

“Film-Noir": 0.105

That means the elder the user, the more they love Fantasy, Western and Film-Noir movies.

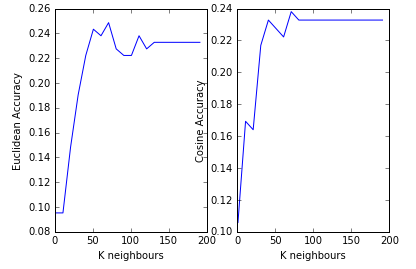
3. Another interesting finding is elder user tend to give higher ratings because most of the correlations are positive numbers except for Documentary and Horror genres.



## Section 3. Classification and Prediction

In this section, we experiment the movie data set with various classifiers, as well as with some of the model evaluation capabilities. The idea is to generate a user to movie rating matrix and use it to find similar users/groups to a given user. Then try to guess user's occupation.

* First we created our own KNN classifier. It allows input the training data matrix, the training labels, the user to be classified, the value of K, and returns the predicted occupation for the user and the top K neighbors. It works with Euclidean distance as well as Cosine Similarity. Then we created a function to compute the classification accuracy over the test data set (ratio of correct predictions to the number of test instances). This function calls the classifier function on all the test instances and in each case compares the actual test class label to the predicted class label. Then run the accuracy function on a range of values for K in order to compare accuracy values for different numbers of neighbors. We did this both using Euclidean Distance as well as Cosine similarity measure and tried to evaluate our classifiers on a range of values of K from 1 through 201 and present the results as a table or a graph.  
  Observation:

1. When k=71, we get highest accuracy for both Euclidean and Cosine distance.  
2. This predictive model is not very good because the relationship between user's rating and his/her occupation is not very strong.  


* Second, we created a classifier based on the Rocchio Method. The classifier takes as input the training data matrix, the training labels, and the user to be classified. It computes the prototype vectors for each of the occupation and measure Cosine similarity of the test instance to each prototype. The output includes the predicted class and the similarity values of the instance to each of the occupation prototypes.
* Third, we ran scikit-learn's KNN classifier on the test set. Generate the confusion matrix and visualize it using Matplotlib, as well as the classification report. Compute the average accuracy score. Experiment with different values of K and the weight parameter for KNN to see if we can improve accuracy.
* At last, we repeated the classification using scikit-learn's decision tree classifier and the naive Bayes classifier. For each model, we compared the average accuracy scores.

Summary of classification & prediction accuracy results:

Classifier Parameters Accuracy

1. Self-dev kNN k=71, Euclidean 24.9%

2. Self-dev Rocchio - 14.3%

3. Scikit-learn kNN k=31, distance 25.4%

4. Scikit-learn tree entropy, split=5 13.8%

5. Scikit-learn NB - 17.5%

## Section 4. Clustering

In this part, we tried to perform K-means clustering on the movie training data. Display the top 3 genres in each cluster, percentage of movies in the cluster in which the genre appear, and the size of the cluster. Using the cluster assignments as class labels, categorize each of the movie in the test data into each of the appropriate cluster. We used the kMeans module form Ch. 10 of MLA (we use the version provided by professor as it includes some corrections to the book version).

Observations:

1. When we group movies to 5 clusters, each cluster will have 1 or 2 dominated genres.

2. When we group movies to 10 clusters, each cluster will have 1 or 2 dominated genres but the percentage of the 1st one appears more than the 2nd one. (Stronger domination)

## Section 5. Item-Based collaborative filtering

Frist, we create a function "test". This function iterates over all users and for each performs cross-validation on movies (by calling the provided "cross\_validate\_user" function), and returns the error information necessary to compute Mean Absolute Error (MAE). Then we performed 5-fold cross-validation (20% test-ratio) and compute MAE results using standard item-based collaborative filtering (based on the rating prediction function "standEst").

The Mean Absoloute Error of function standEst is 0.7992

Second, we create a function "print\_most\_similar\_movies" which takes the movie ratings data, a query movie id, parameter k for the number of nearest neighbors, and a similarity metric function, and prints the names of the query movie as well as the names of the top k most similar movies based on user ratings.

Sample output:

Selected movie:

Star Wars (1977)

Top 5 Recommended movies are :

Return of the Jedi (1983)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Empire Strikes Back, The (1980)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Raiders of the Lost Ark (1981)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Indiana Jones and the Last Crusade (1989)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Toy Story (1995)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Selected movie:

Forrest Gump (1994)

Top 5 Recommended movies are :

E.T. the Extra-Terrestrial (1982)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Apollo 13 (1995)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dances with Wolves (1990)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Raiders of the Lost Ark (1981)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Back to the Future (1985)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

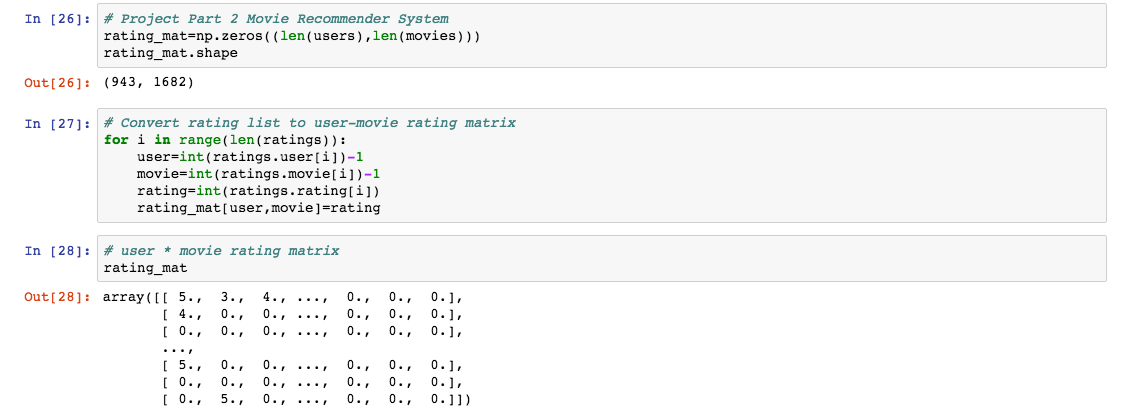
# Application development part 1. User based collaborative filtering recommender system

The structure of the project is organized as three parts, the first part is data analysis, the second part is application 1 (recommendation system using collaborative filtering with user-based), the third part is application 2(recommendation system using collaborative filtering with item-based). Below paragraph are mainly discussing the application 1.

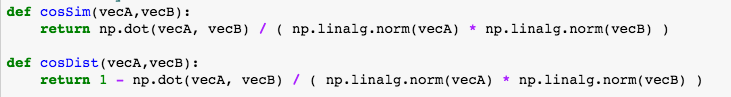
The purpose of application 1 is to apply k-nearest-neighbor algorithm on the user rating data and predict users rating for the selected movies. The overall workflow of this application is organized as follows:

1. Prepare the user rating data
2. Convert the original user rating data to rating matrix
3. Select user and movie for the rating prediction
4. Find the k nearest neighbors of the selected users based on the distances(cosine) of their rating vectors
5. Calculate the rating by averaging the rating for the selected movies from top k neighbors
6. Compare the prediction rating with the actual rating (if the user had rated it already)

As mentioned previously in the data analysis part, this dataset from movieLens contains 943 users and 1682 movies, the original format of rating data is (user – movie-rating), we converted the data to user x movie ratings matrix for the computation convenience.



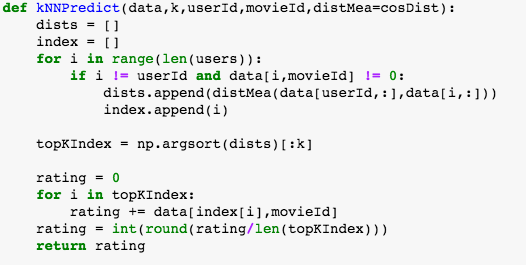
To calculate the similarity/distance between users, we adopted the cosine algorithm because it is compatible with the vector computation and would not be distorted by a scale problem (even this problem does not exhibit in the rating data)



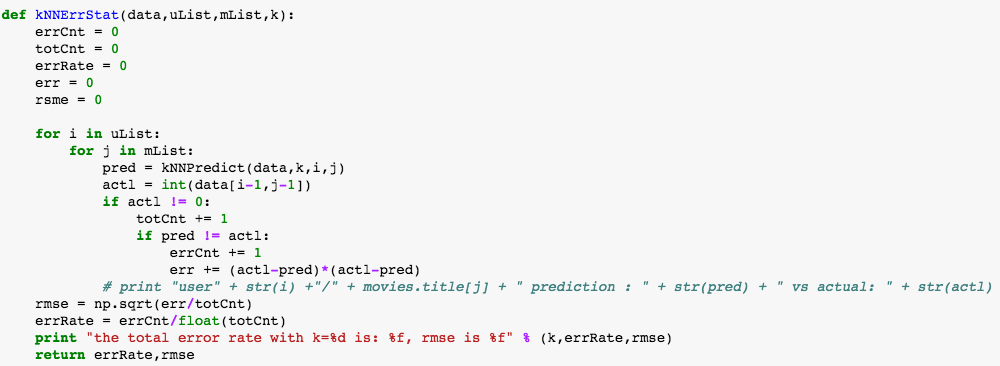
To have a glimpse of the user profile, we wrote a function to present a user’s age, gender, occupation and some of his/her favorite movies.



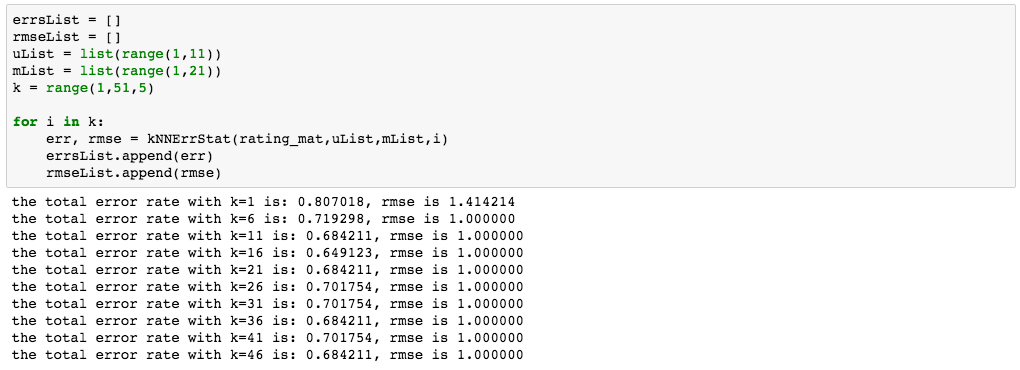
We have implemented the kNN algorithm, the input parameter will include the rating matrix, k number, user id, movie id and distance function (default distance function is cosine distance)

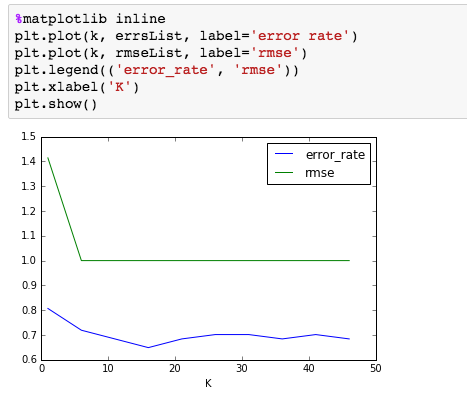


We also wrote a function to loop through different kNN parameters and keep the statistics of kNN error rate and rmse.



Below is a sample run of the application part 1 on the movieLens data, we selected 10 users (user 1 to 10) and 20 movies (movie 1 to 20) and we tried 10 different k (1 to 46, steps = 5).





As the test data indicates, that the accuracy went up as the k becomes larger, the RMSE also went lower as the k increased in the test run. When k = 1, the error rate is relatively high (80.7%) and the error rate went down to about 70% as the k increased, the best error rate is 64.91% when k = 16, the error rate fluctuated around 70% but remained stable even the k increased.

# Application development part 2. Item based collaborative filtering recommender system

## Introduction

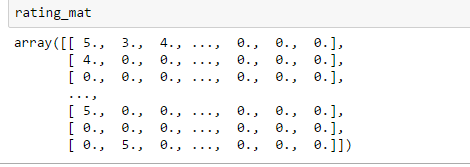
This is an Item-based collaborate filter system application, which has 3 main functions: “***rec\_most\_similar\_movies***” --- Given a movie, recommend the top n similar movies

”***PredictRateForUser***”--- Given an unrated movie, predict the rating for a specific user

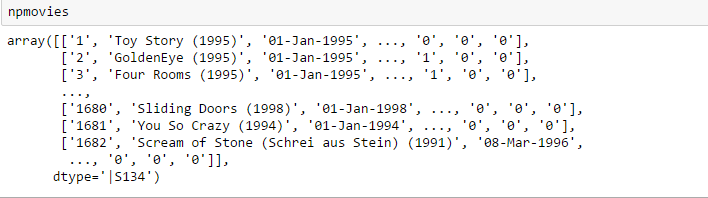
“***PredictRateForUserMovie***” --- Given a user, find all his/her unrated movies and predict the ratings

## Parameters:

This application will take generally 3 parameters, which are movie rating table (***rating\_mat***), movie database (***npmovies***), and similarity function (**c*osSim2, pearsSim, ecludSim***) to use. The movie rating table is a table contains lists of ratings for each user and is got from “u.data “, which contains user id and the user’s rating for each movie.

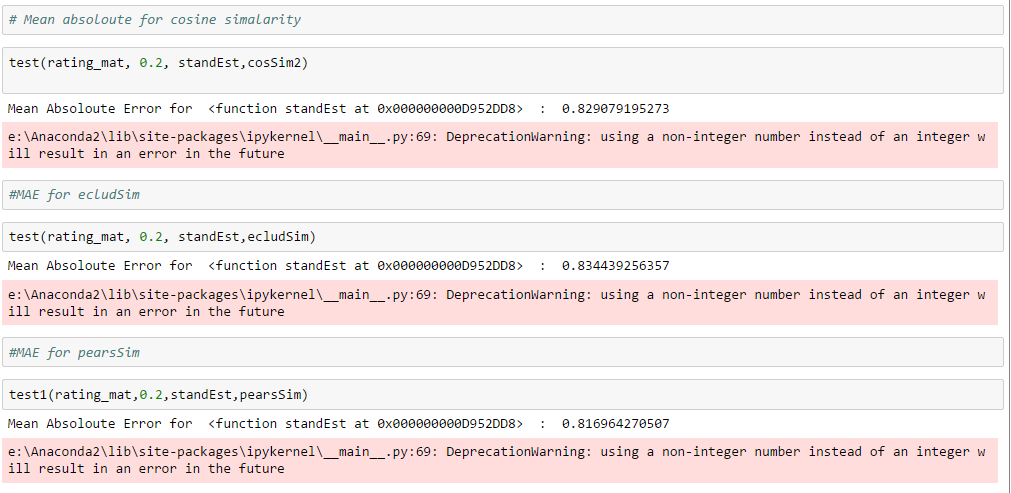
Movie rating table:

Movie database:



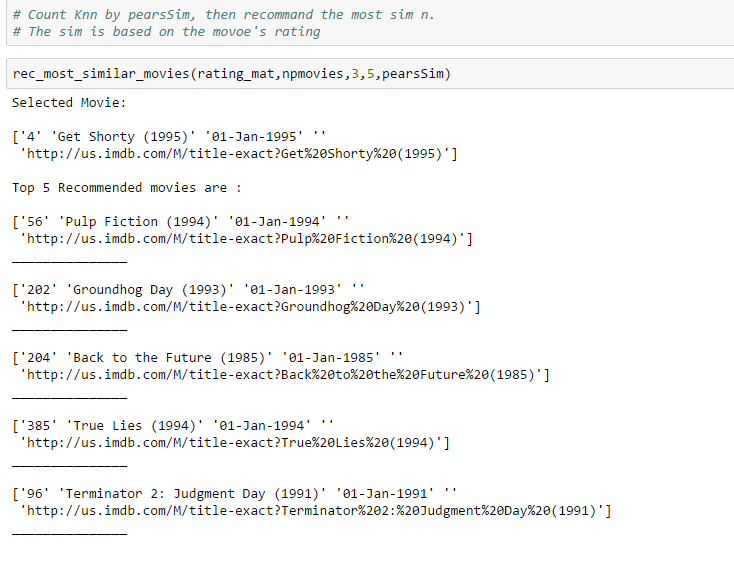
## Function Details

To build this application, we first import “itemBasedRec.py”, and then built a test function to test the mean absolute error for each similarity mothed, the results are:



From the test result we found that although pears method seems slight better (0.817 vs 0.834 and 0.829), there is not a significant difference using anyone of those three similarity method. Then we build the function ***rec\_most\_similar\_movies,*** which will recommend top n similar movies for that user. This function takes 5 parameters: the movie rating table, movie database, the movie used as simple ,number of movie user wants to be recommend, and the similarity method. The basic idea is KNN search. We find the top n similar movies, and recommend them to that user.

The result of the function is like:

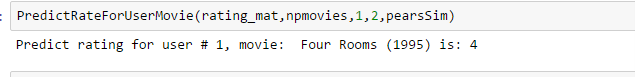


The ***PredictRateForUser*** function will automatically predict all unrated movie for a user, and the prediction is based by that user’s rating on most similar movies. This function takes 4 parameters:

The movie rating table, movie database, user id, and the similarity function. And the output of this function is:



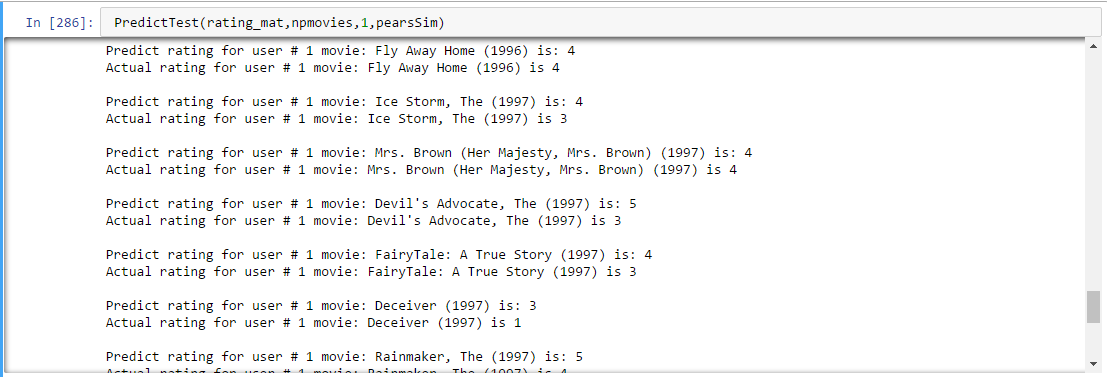
Function ***PredictRateForUserMovie*** is pretty much like ***PredictRateForUser,*** but specified which movie the user want to predict:



## Test

In the test part we built 2 test functions, which are ***PredictTest*** and ***PredictTest2***. The idea is to use the same way to predict the rated movie for a specific user, and compare the predicted result with actual rating. And for accuracy, we used the distance between the predicted rating and actual rating.

Example for PredictTest:



Next, we take 20% or the total user as test target (189 users), and use ***PredictTest2*** to test the accuracy with different similarity method, the result is:



This result is consist with the MAE test result. We can see that all three similarity method gives the accuracy around 0.79, and pears similarity gives the highest accuracy, with is 0.7964 and just slightly higher than other 2 methods.

The overall test results are around 78% and are acceptable.

# Appendix:

In addition to the report above, several supplement materials are included:

* Data\_Analysis.ipynb, Data\_Analysis.html (The ipython notebook for data analysis part)
* app1.ipynb, app1.html (The ipython notebook for the demonstration of the user based collaborative filtering system)
* app1.py (The interactive python program that user can test on the user based collaborative filtering system recommender system)
* app1\_readme.txt (Instructions on how to run the ipython notebook and interactive python program for user based collaborative filtering system)
* app2.ipynb, app2.html (The ipython notebook for the demonstration of the item based collaborative filtering system)
* app2.py (The interactive python program that user can test on the item based collaborative filtering system recommender system)
* app2\_readme.txt (Instructions on how to run the ipython notebook and interactive python program for item based collaborative filtering system)

# References:

1. CSC478 course material: <http://facweb.cs.depaul.edu/mobasher/classes/csc478/>
2. Machine Learning In Action
3. Michael J. Pazzani , Daniel Billsus: Content-based Recommendation Systems, <http://www.fxpal.com/publications/FXPAL-PR-06-383.pdf>
4. Wikipedia on Recommendation System: <https://en.wikipedia.org/wiki/Recommender_system>