# 22C:16 (CS:1210) Project 2

## Introduction

As we browse the web, we are constantly being recommended stuff – videos, news articles, books, movies, music, etc. We have now come to routinely expect recommendations from popular websites such as Amazon, Netflix, Pandora, youTube, New York Times, etc. These recommendations are made possible by *recommender systems* that these websites run. Recommender systems contain a lot of computer science under the hood and this project aims to give you a glimpse of this. A successful recommendation system has to, by some means, be able to predict a user's tastes and make recommendations accordingly. Over the years, Netflix has invested a lot of resources into improving its movie recommender system – some of you may have heard of the *Netflix prize.* Between 2006 and 2009, Netflix ran a competition, with a prize of one million dollars to the team that could take a given dataset of over 100 million movie ratings and return recommendations that were 10% more accurate than those offered by the company's existing recommender system. This competition did a lot to energize the search for new and more accurate algorithms and better implementations of these algorithms. In this programming project, you are asked to build a simple recommender system for movies.

One, relatively new approach to coming up with recommendations is called *collaborative filtering.* In this approach, recommendations are made on the basis of "collaboration" among many users. Typically, such a system will collect numeric ratings (e.g., from 1 through 5 with 1 being worst and 5 being best) from users on a collection of items (e.g., movies). To determine what to recommend for a user "Alice," the recommender system looks at all users who have tastes similar to Alice's tastes and uses the items they have liked as a source of recommendations for Alice. How does the system figure out who has tastes similar to Alice? It simply uses past ratings to do this – those users who have rated items in a manner similar to Alice are considered to have tastes similar to Alice.

For this programming project we provide to you a dataset that contains 100,000 movie ratings. These are real ratings by real people gathered by the Group Lens research group at the University of Minnesota (see `http://www.grouplens.org/`). The dataset is made available with permission from Group Lens. Here is a nice summary of the data set from the `README` file accompanying the data set.

> The data was collected through the MovieLens web site (movielens.umn.edu) during the seven-month period from September 19th, 1997 through April 22nd, 1998. This data has been cleaned up – users who had less than 20 ratings or did not have complete demographic information were removed from this data set.

The ultimate goal of your program is to take a user (specified by an ID) and make movie recommendations for this user based on the rating history of this user and all the others in the provided data set.

## Data Files

The dataset is available in the directory

`http://homepage.cs.uiowa.edu/ sriram/16/spring13/projects/project2/dataset/`

and consists of a `README` file along with 6 other text files. The data in all 6 files is quite clearly explained in the `README` file, in the section titled "Detailed Descriptions of Data Files." So read the `README` file carefully first and then look through the data files to get a sense of how they are organized.

## Stage 1 (Due on Monday, 4/29)

In the first stage of the project (due on Friday, 4/26) your program is expected to read from the given files store the information in appropriate data structures, and answer simple queries about the data. Specifically, it is required to create five lists – a user list, a movie list, two ratings list, and a genre list – these are described below in detail.

- Write a function with the header:

  ```
  def createUserList():
  ```

  that reads from the file `u.user` and returns a list containing all of the demographic information pertaining to the users. Suppose we call this function as `userList = createUserList()`. Then `userList` should contain as many elements as there are users and information pertaining to the user with ID $i$ should appear in slot $i - 1$ in `userList`. Furthermore, each element in `userList` should be a dictionary with keys "`age`", "`gender`", "`occupation`", and "`zip`". The values corresponding to these keys should simply be appropriate values read from the file `u.user`. For example, the first line in `u.user` is

  ```
  1|24|M|technician|85711
  ```

  and therefore `userList[0]` should be the dictionary

  ```
  {"age":24, "gender":"M", "occupation":"technician", "zip":85711}
  ```

  Thus `userList` is a list with 943 dictionaries, each dictionary containing 4 keys.

- Write a function with the header:

  ```
  def createMovieList():
  ```

  that reads from the file `u.item` and returns a list containing all of the information pertaining to movies given in the file. Suppose we call this function as `movieList = createMovieList()`. Then `movieList` should contain as many elements as there are movies and information pertaining to the movie with ID $i$ should appear in slot $i - 1$ in `movieList`. Furthermore, each element in `movieList` should be a dictionary with keys "`title`", "`release date`", "`video release date`", "`IMDB url`", and "`genre`". The values corresponding to these keys should simply be appropriate values read from the file `u.item`. For example, the first line in `u.list` is

```
1|Toy Story (1995)|01-Jan-1995||http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)|0|0|0|1|1|1|0|0|0|0|0|0|0|0|0|0|0|0|0
```

  and therefore `movieList[0]` should be the dictionary

  ```
  {"title":"Toy Story (1995)", "release date":"01-Jan-1995", "video release date":"",
  "IMDB url":"http://us.imdb.com/M/title-exact?Toy%20Story%20(1995)",
  "genre":[0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0]}
  ```

  Note that the value associated with the key "`genre`" is a length-19 list of zeroes and ones.

- Write a function with the header:

  ```
  def createRatingsList(numUsers, numMovies):
  ```

  that reads from the file `u.data` and returns two lists containing all 100,000 ratings provided in `u.data`. The function takes as arguments the number of users and the number of movies in the data set. Suppose we call this function as `[rLu, rLm] = createRatingsList(numUsers, numMovies)`. Then `rLu` is a list, with one element per user, of all the ratings provided by each user. Similarly, `rLm` is a list, with one element per movie, of all the ratings received by each movie. In particular, the ratings provided by user with ID $i$ should appear in slot $i-1$ in `rLu` and the ratings received by movie with ID $i$ should appear in slot $i-1$ in `rLm`. We explain `rLu` a little bit more; `rLm` is quite similar. The ratings, by a particular user, appear as a dictionary whose keys are IDs of movies that this user has rated and whose values are corresponding ratings. For example, the user with ID 1 has rated movie 61 and given it the rating 4. Hence the key-value pair `61:4` should appear in `ratingsList[0]`. In `rLm`, the ratings received by a movie appear as a dictionary whose keys are IDs of users who have rated that movie. Note that at this point we are ignoring the time stamp values provided in the `u.data` file.

- Write a function with the header:

  ```
  def createGenreList():
  ```

  that reads from the file `u.genre` and returns the list of movie genres listed in the file. The genres should appear in the order in which they are listed in the file.

After creating these data structures you are required to write a bunch of functions to answer simple queries. Below we specify the headers of the required functions along with documentation describing what the function is expected to do.

```
# returns the mean rating provided by user with id u. The second argument is
# the ratings list containing ratings per user.
def meanUserRating(u, userRatings):

# returns the mean rating received by a movie with id u. The second argument is
# the ratings list containing ratings per movie. Only movies that have received
# at least 10 ratings are include in the mean computation.
def meanMovieRating(u, movieRatings):

# Given a genre ID and a movie list, this function returns the list of all movie titles in the given
# genre, sorted in alphabetical order.
def moviesInGenre(genre, movieList):

# This function takes a genre ID, movie list and the ratings list containing ratings per movies.
# It returns a list of tuples containing movies titles and associated mean ratings. This list should
# only contain movie titles of the given genre that have received at least 20 ratings and should be
# sorted in decreasing order of mean rating received.
def popularMoviesInGenre(genre, movieList, movieRatings):

# This function takes a genre ID, movie list and the ratings list containing ratings per movies.
# It returns the mean rating for the given genre, which is the mean taken over all ratings of all
movies in the genre.
def meanGenreRating(genre, movieList, movieRatings):

# This function takes a genre list, movie list and the ratings list containing ratings per movies.
# It returns a sorted list of genres, sorted in decreasing order of mean rating.
def popularGenres(genreList, movieList, movieRatings):
```

## Stage 2 (Due on Friday, 5/10)

### Under Construction

Requires the implementation of two recommendation algorithms followed by the evaluation of these algorithms. Hide 20% of the data chosen at random. Make predictions and compare with actual data.