**P2**

In this problem we will explore a real-world example to apply machine learning concepts to.

**Background**

GameHaven will be a social media platform enabling users to explore the board gaming hobby. GameHaven will track user interests and matching algorithms will determine new games that users are likely to enjoy and provide them with opportunities to find players and locations to play.

Games are described by many parameters that include primary categories and tags. Each game will be binned 1 through 5 in each of the following primary categories. Values for the game Chess will be provided as an example:

* Complexity (1, 2, 3, 4, 5) – Chess: 5, very complex
* Depth (1, 2, 3, 4, 5) – Chess: 5, very deep
* Speed (1, 2, 3, 4, 5) – Chess: 3, takes an average time to play a typical game
* Thematic (1, 2, 3, 4, 5) – Chess: 2, does not have a strong theme to it other than conflict
* Interaction (1, 2, 3, 4, 5) – Chess: 5, has very strong interaction with the opposing player
* Players (1, 2, 3, 4, 5) – Chess: 2, for 2 and only 2 players
* Mass (1, 2, 3, 4, 5) – Chess: 2, is a relatively small game
* Obscurity (1, 2, 3, 4, 5) – Chess: 1, one of the world’s most popular games

And tags are a list of attributes that a game either has or doesn’t: like dice, a space theme, drawing mechanics, bluffing etc.

Games are kept in a database that describe their vales for each of the categories and a list of applicable tags. Each time a player expresses “like” or “dislike” for a game, the users preferences associated with the attributes of the game are modified. Liked Chess? Your preference rating for the most complex category of games is increased.

While GameHaven is still in development, mock data has been generated to test the machine learning algorithms in the context of clustering users and matching games. It is believed that an unknown number of user clusters will develop. Example clusters might include groups of users that like: card games, party games, role-playing games, or “euro” games. To simulate these clusters, 4 patterns of users have been created as base preferences. The user profile data is a combination of the base pattern number, and variability is multiplied against the base. In other words, if the variability component were reduced to zero, the user data would be expressed by a rank-4 matrix.

**(please run section 2.1 of *Working with GameHaven Data.ipynb*)**

The of user preferences has been converted from .json placed into a taste profile matrix: users represent different rows, and each column represents a different preference. The first 40 columns are preferences for each of the 5 bins of eight categories, and the remaining 69 are preferences for the different game tags.

A lower rank model could be calculated offline and a low rank model could be used to provide the users a fast and responsive interface when interacting with the GameHaven platform. This project will look at two different mechanisms for creating a lower rank model of the user data. Both SVD and k-means will be used to create a model and the results will be compared.

**(please run section 2.2 of *Working with GameHaven Data.ipynb*)**

Section 2.2 runs a python SVD algorithm on the 500x109 user preference data. Using the results, rank(1) through rank (109) models of the data are compared to the original data. A plot is generated that shows the 2-Norm difference between each of the 109 different models and the original data.  
  
**Questions:**

What rank model would you recommend for this data and why?

It is expected that the real user data will have patterns, but not the four clear patterns that were used to generate the mock data. Would this change your answer in the previous question? How should the model rank be chosen if the results are a gradual continuum as the rank of the model is varied?

**(please run section 2.3 of *Working with GameHaven Data.ipynb*)**

If developing a model, it’s useful to check how well the model is fitting the data. Section 2.3 of the python code allows the comparison of specified users for different ranks of the model. Steps:

* Select ‘Model Rank’ and hit ‘Update Model’ button
* Select ‘User’ and hit ‘View User’ button
* Repeat as necessary

**Questions:**

Please experiment with different values of model rank and some different users. In the context of the curve shown in section 2.2, how does the proximity of values in side-by-side model comparisons

**(please run section 2.4 and 2.5 of *Working with GameHaven Data.ipynb*)**

The k-means algorithm was employed to create an alternate low rank model. Again, all 109 model ranks were created and the 2-norm of the model vs. data are plotted as a function of rank.

**Questions:**

Taking the SVD of the player data is computationally more complex than the k-means algorithm. Why does sweeping through 109 models take so much longer for k-means?

**(please run section 2.6 of *Working with GameHaven Data.ipynb*)**

A plot comparing the 2-norm of rank 1-109 models of both SVD and k-means are shown.  
  
**Questions**:  
Which technique creates a more accurate model of the data. Why?

The variable element of the mock data had to be greatly reduced to generate results that show strong results for a rank(4) model. What will this mean when GameHaven starts collecting data with a lot more noise?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

For the second part of this problem, please open testp2.m in Matlab.

One method we will use to interact with the data is Collaborative Filtering. Collaborative Filtering is a method used to make recommendations to a user based on their preferences and tastes compared with a data set of many users. This method is helpful for GameHaven to recommend board games to their users based on their tastes.

The initial dataset of GameHaven users and their properties is created from a pattern + randomization method. This ensures that data will be predictable but random across the users. Similarly a matrix of game data is created with the same attributes as the users.

From the game and user attribute matrices, a random sparse ratings matrix is generated. The rating is based on how well the user attributes match the game attributes. Additionally to mimic a real world scenario the data is sparse so there are very few initial ratings.

The collaborative filtering system demonstrated here is done in the following way:

1: Cluster users based on their characteristics

First use SVD to determine a good rank for the user matrix, and use that value for the number of clusters.

2: Create similarity matrix between users in the same cluster

Collaborative filtering uses a similarity function. The goal of a similarity function is to measure how similarity between vectors **x** and **y.** In our example we are using cosine similarity.

A similarity matrix **S** can be constructed from **X** where  where .

3: User similarity matrix to fill in missing data in the ratings matrix

The ratings matrix maps the ratings of different games by users, so its size is the number of games by number of users. **R(I,j)** is the rating that user **i** has for game **j.** As mentioned before, most of **R** is zeros since most users do not have ratings for most games. To estimate the ratings for a user based on the given data, we do the following:

1. Extract the ratings of all the players in the same cluster as the given user **R1 = R(user in cluster 1).**
2. Fill in ratings matrix using 3 different collaborative filtering methods as follows:
   1. Unweighted cluster average
   2. Weighted cluster average
   3. Unweighted full average
   4. Weighted full average
3. Compare result, error and timing across 4 methods.

4: Sort ratings matrix and provide user top 10 recommended games

Questions:

1. What is the best number of clusters to represent this data set?
2. Implement the similarity\_matrix.m function
3. What is the highest recommended game for user # 15 using each different method?
4. Compare the timing of each method. Which method would be best for scaling?