1 Recommendation System

1.1 Short Theory

A recommender system or a recommendation system is a subclass of information filtering system that seeks to predict the “rating” or “preference” that a user would give to an item. Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, online dating, and webpages.

Let n_u be the number of users and n_m be the number of items. Denote n_r is number of rattings given by the users towards the items. The recommendation system tries to find M missing ratings: $M = n_m n_u - n_r$. Many methods can be used to predict the missing values such as *Matrix Factorization*, *Content-based recommendation*, *User-based recommendation*. In this exercise, we focus on *Collaborative Filtering*.

Collaborative Filtering tries to learn latent representations of users and items. Two approaches exist:

- Learn the representations of users and items separately. This is a process with two steps. First, we learn the users’ representation and next we address the items’ representation. The process is repeated until convergence.
- The representatios of users and items can be learned in a joint process.

In this exercise, we follow the second approach. Let us define the cost function for the collaborative filtering algorithm as follows:

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

In equation (1), $\theta^1, \ldots, \theta^{(n_u)}$ represents users' feature, and $x^{(1)}, \ldots, x^{(n_m)}$ represents items' feature. y is the ground truth (i.e. known ratings) and r is the set of the known ratings' indices. The cost function above can be minimized using gradient descent. The detail of the collaborative filtering includes several steps as follows:

- Step 1: Initialize $\theta^{(j)}$ and $x^{(i)}$ to small random values.
- Step 2: Minimize the cost function using gradient descent.

$$\begin{aligned} x_k^{(i)} &:= x_k^{(i)} - \alpha \left(\sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta_k^{(j)} + \lambda x_k^{(i)} \right) \\ \theta_k^{(j)} &:= \theta_k^{(j)} - \alpha \left(\sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda \theta_k^{(j)} \right) \end{aligned} \tag{2}$$

- Step 3: Repeat step 2 until convergence, or until a certain number of iterations has beed performed.

Having obtained the representations of users and items, we can make prediction of user i to item j by

$$rate(i,j) = (\theta^{(j)})^T x^{(i)}$$

1.2 Python Exercise

In this exercise, we will implement the collaborative filtering algorithm. We are going to use a standard dataset for studying recommendation system named "MovieLens100K". The dataset contains 943 users and 1682 movies. There are $100,000$ integer ratings, with values in range $[1,5]$. For your convenience, the dataset has been split into a training and a testing set.

Step 1: Load dataset and create mask matrix. The loading procedure has been done for you while implementing function *create_mask* is your task. This function outputs a matrix of the same size with the input matrix. Value of an element in the output matrix indicates whether the corresponding rating exists (1) or not (0).

Step 2: Implement cost and gradient functions. You will implement functions *compute_cost* and *compute_gradient*. You can refer to equations (1) and (2) for this step.

Step 3: Learn the representations of users and movies. In this step, you need to use the update rule to learn Θ and X. Beaware that these representations are updated at the same time.

Step 4: Make predction. You will use the learned features to make prediction on the testing set, which has been kept separately. You will need to make a new mask matrix for the testing set. We will use two criteria to evaluate your implementation. Concretely, we employ *Root Mean Squared Error* (RMSE) and *Mean Absolute Error* (MAE) for evaluating your learned models. The values for RMSE and MAE on the testing set shoule be around 0.98 and 0.76.