# CSE 258 Assignment 1

**Feng Jiang**

**Kaggle: Doraemon**

**PID: A53248625**

November 21, 2017

**Visit prediction: Use collaborative filtering to recommend items for similar users. Rating prediction: Implement latent factor model to predict the rating of a given user.**

## 1 Visit prediction

In the visit prediction task, we were given a set of user-item visit pairs to make true-false binary prediction for unknown user-item pairs. I implemented the collaborative filtering algorithm on the user, that is, given a user-item pair, if this item is visited by users that are very similar to the given user, I predict the instance to be true, otherwise, false. The similarity here is defined as the Jaccard similarity.

$$Jaccard(A, B) = \frac{|A \bigcap B|}{|A \bigcup B|}$$

where $|A \bigcap B|$ is the length of the intersection of user A's and B's categories and $|A \bigcup B|$ is the length of the union of user A's and B's categories. The final categorization accuracy I achieved on Kaggle is 0.720004.

### 1.1 Algorithm implementation and assumptions

The training set and validation set were generated as specified in homework 3. The actual implementation of the algorithm is described in algorithm 1.

The reason that I used categories in similarity computation is that people tend to repeatedly visit places that are of some specific kinds. Foodies tend to explore different restaurants. People interested in fashion might frequently visit clothing stores. If they have established certain preference to some kinds of places as indicated in the training set, they might seldom deviate from this pattern. So I used categories to measure people's

**Algorithm 1** collaborative filtering

```
1: procedure TRAINING
2:     store the categories of items that each user visited
3:     store each user's visited items as liked items, if the item's ratings is not below the user's average rating by 1.0
4:     setup a threshold to represent how similar two users need to be to be considered as "very" similar
5: for each user cur_user:
6:     compute his(her) Jaccard similarity with all the other users temp_user in the training set
7:     if similarity is above the threshold then
8:         add temp_user's liked items to cur_user's potential items set
9: procedure PREDICTING
10: for each user item pair:
11:     predict ← false
12:     if user is seen before then
13:         if the item is user's potential items then
14:             predict ← true
15:     else
16:         if the item is popular item (as in hw3) then
17:             predict ← true
```

similarity. Another legitimate criteria for similarity could be the actual items that users visit. But I thought in real life, the chance that different users happened to buy the same item might be too small, so the user-item pair matrix might be too sparse. And my latter experiment on visited items didn't greatly improve my performance.
My another assumption is that if two users have similar taste, what one user like might also be attractive for another user. So I used collaborative filtering.
The last assumption is that the rating of a user reflected whether he(she) like the item or not. I don't want to make recommendation to his similar user if the original user even don't like the items himself. So if the rating of an item is below the average rating of this user by one point, I won't recommend it to his similar user.

### 1.2 Tuning and experiment

I tuned for the similarity threshold for users using 0.3, 0.2, 0.175, 0.15, 0.1. The prediction accuracy on my validation peak when it was set to be 0.175. I also tried build a "two-way" collaborative filtering, where besides the collaborative filtering with users, I also did the same with items and combine the results to gether to make prediction. However, the false positive rate was too high and it didn't improve on the simple one. Finally, I also tried use visited items instead of categories to calculate users' similarities. However, the results were also not improved.

## 2 Rating prediction

In the rating prediction task, we were given users with their reviews including rating for items and we needed to build model to predict users' ratings on unknown items. I built the latent factor model to perform the task.

$$f(u,i) = \alpha + \beta_u + \beta_i$$

where $f(u,i)$ is the rating for a user item pair, $\alpha$ is the rating bias, $\beta_u$ is how much certain user tend to rate above the mean, and $\beta_i$ is who much certain item tend to be rated above the mean. The model was updated by changing $\alpha$, $\beta_u$, and $\beta_i$ to minimize the objective function:

$$\sum_{u,i}(\alpha + \beta_u + \beta_i - R_{u,i})^2 + \lambda[\sum_u(\beta_u^2) + \sum_i(\beta_i^2)]$$

This is a convex optimization problem so during training, we solve for the closed form solution until convergence to update $\alpha$, $\beta_u$, $\beta_i$:

$$\alpha = \frac{\sum_{u,i}(R_{u,i} - \beta_u - \beta_i)}{N_{train}}$$

$$\beta_u = \frac{\sum_{i \in I_u}(R_{u,i} - \alpha - \beta_i)}{\lambda + |I_u|}$$

$$\beta_u = \frac{\sum_{u \in U_i}(R_{u,i} - \alpha - \beta_u)}{\lambda + |U_i}$$

Although this model doesn't incorporate the interaction between user and item, it achieved a better performance than other more complex models as my experiment showed. My final (root) mean-squared error on Kaggle was 0.75764.

### 2.1 Algorithm implementation and assumptions

The training set and validation set were generated as specified in homework 3. The actual implementation of the algorithm is described in algorithm 2.

**Algorithm 2** latent factor model

```
1: procedure TRAINING
2:     initialize $\alpha$ to be the mean of the rating, $\beta_u$ and $\beta_i$ to be vector of random number within -5. to 5.
3:     while not converge do
4:         update $\alpha$, $\beta_u$ and $\beta_i$ using the rule described in previous section
5: procedure PREDICTING
6:     $f(u,i) = \alpha + \beta_u + \beta_i$
```

The assumption I was making here was that we could make prediction on the rating of an unknown user-item pair's based only on the past rating records of the user and item, while the features of items and users and the interaction between users and items were not as important. This simple model actually worked pretty well as I will be pointing out below.

### 2.2 Tuning and experiment

I tuned for $\lambda$ which determined the impact of the regularization term. $\lambda$ value of 1, 2, 3, 4, 5 was tried and it was observed that as I decreased $\lambda$, the final mean square error (MSE) on validation set was increased while the performance on Kaggle test set was decreased. The reason might be that the model overfit with a small $\lambda$. I also tried implemented a more complex version of latent factor model:

$$f(u,i) = \alpha + \beta_u + \beta_i + \gamma_i\gamma_u$$

where $\gamma_i$ and $\gamma_u$ are $k$ dimensional representation of items and users' features. So their dot product will theoretically model the interaction between them. However, trying $k$ with value 2 and 5 had close the performance as the simple version on the Kaggle test set but didn't improve, although the MSE on my validation set was increased. Maybe the same issue with overfitting was a problem with k-latent factor model.