Notes: Matrix Factorization Techniques for Recommender Systems

1. Recommender System Strategies

1.1 Content Filtering

Creates a profile for each user or product.

For example, a movie profile could include attributes regarding its **genre**, **actors**, **popularity**. A user profile might include **demographic** information. However, it requires gathering external information that might not be available or easy to collect.

1.2 Collaborative Filtering

Analyzes relationships between users and interdependencies among products to identify new user-item associations.

Doman free, but can address data aspects that are often elusive and difficult to profile using content filtering. Generally more accurate than content filtering, but suffers from **cold start** problem, and content filtering is superior in this situation.

The **2** primary areas of collaborative filtering are the **neighborhood methods** and **latent factor models**.

1.2.1 Neighborhood Methods

Centered on computing the relationships between items or users.

1. **Item-oriented** approach evaluates a user's preference for an item based on ratings of "neighboring" items by the same user. A product's neighbors are other products that tend to get similar ratings when rated by the same user.
2. **User-oriented** approach identifies like-minded users who can complement each others' ratings. Finds a user's neighbors who are users that also like the movies the user likes.

1.2.2 Latent Factor Models

Tries to explain the ratings by charactering both items and users on 20-100 factors inferred from the ratings patterns.

For movies, the discovered factors might measure obvious dimensions like genre, quirkiness or completely uninterpretable dimensions. For users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

2. Matrix Factorization Methods

Some of the most successful realizations of latent factor models are based on **matrix factorization**. Each item or user has a vector of factors. High correspondence between item and user factors leads to a recommendation.

The input data of recommender system is often a matrix where one dimension represents user and the other represents items of interest.

The most convenient data is high-quality **explicit feedback**: star ratings of stars, thumbs-up and thumbs-down. We call explicit feedback ratings. Usually it's a sparse matrix because a user is likely to only have rated a small percentage of all possible items.

When explicit feedback is not available, we can infer user preferences using **implicit feedback** based on users' past behaviors: purchase history, browsing history, search history. It usually denotes the presence or absence of an action, so a it's a densely filled matrix.

3. A Basic Matrix Factorization Model

Maps both users and items to a joint latent factor space of dimensionality f, then the user-item interactions are modeled as inner products.

Each item i is associated with a vector $q_i \in \mathbb{R}^f$, and each user u is associated with $p_u \in \mathbb{R}^f$. For i, the elements of q_i measure the extent to which the item possesses those factors. For u, the elements of p_u measure the extent of interest the user has in items that are high on the corresponding factors. Then user u's rating of item i is approximated by

$$\hat{r}_{ui} = q_i^\top p_u$$

This model is closely rated to **singular value decomposition**. But due to high portion of missing values in a sparse ratings matrix, conventional SVD is undefined. Moreover, only considering the known entries can easily lead to overfitting. Imputation is also not desirable.

Recently we mainly model directly the observed data only with regularization. The objective is minimizing the the regularized squared error on the set of known ratings:

$$\min_{q^*,p^*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^\top p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)$$

κ is the set of the (u,i) pairs for which r_{ui} is known (the training set). λ is the regularization coefficient, which is usually determined by cross-validation.

4. Learning Algorithms.

4.1 Stochastic Gradient Descent (SGD)

Loops through all ratings in the training set. For each training case, predicts r_{ui} and computes the error

$$e_{ui} = r_{ui} - q_i^\top p_u$$

The update rules are

$$q_i = q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u = p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

where γ is the learning rate and λ is the regularization coefficient.

Derivation of the Update Rule (Sketch):

$$\begin{aligned}
&\frac{\partial}{\partial q_i}((r_{ui} - q_i^\top p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)) \\
=&\frac{\partial}{\partial q_i}((e_{ui})^2 + \lambda(\|q_i\|^2 + \|p_u\|^2)) \\
=&2e_{ui}\frac{\partial e_{ui}}{\partial q_i} + \lambda \cdot q_i \\
=&-2e_{ui} \cdot p_u + \lambda \cdot q_i
\end{aligned}$$

Then we have the update rule

$$\begin{aligned} q_i &= q_i - \gamma \cdot \text{gradient} \\ &= q_i - \gamma \cdot (-2e_{ui} \cdot p_u + \lambda \cdot q_i) \\ &= q_i + \gamma(2e_{ui} \cdot p_u - \lambda \cdot q_i) \end{aligned}$$

4.2 Alternating Least Squares (ALS)

Both q_i and p_u are not unknown, so the objective function is not convex. We can fix one of the unknowns and the problem becomes quadratic. ALS rotates between fixing q_i's and p_u's.

In general SGD is easier and faster, but ALS is favorable in 2 cases:

1. Parallelization.
2. Implicit data. Since the matrix is dense, it is not practical to loop through all cases as SGD does.

5. Adding Biases

Much of the observed variation in rating values is due to effects with either users or items, known as **biases**. Some users tend to give higher ratings and some items tend to receive higher ratings.

The bias for r_{ui} is calculated as

$$b_{ui} = \mu + b_i + b_u$$

where μ is the overall average rating, and b_i and b_u are the observed deviations of u and i from the average.

Then we estimate r_{ui} as

$$\begin{aligned} \hat{r}_{ui} &= b_{ui} + q_i^\top p_u \\ &= \mu + b_i + b_u + q_i^\top p_u \end{aligned}$$

The new objective function now becomes

$$\min_{q^*,p^*,b^*} \sum_{(u,i)\in\kappa} (r_{ui} - \mu - b_u - b_i - q_i^\top p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2)$$

6. Additional Input Sources

Cold start problem: Many users supply very few ratings, making it difficult to reach general conclusions on their taste. We can incorporate additional sources about users: behavior information and purchase or browsing history.

$N(u)$ denotes the set of items for which user u expressed an implicit preference. Each item i is associated with a factor vector $x_i \in \mathbb{R}^f$. A user is characterized by

$$\sum_{i\in N(u)} x_i$$

Using normalization:

$$|N(u)^{-0.5}| \sum_{i\in N(u)} x_i$$

$A(u)$ describes u's attributes: gender, age group, zip code, income level. Factor vector y_a corresponds to each attribute to describe a user:

$$\sum_{a \in A(u)} y_a$$

Then we have the prediction function:

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^\top [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{a \in A(u)} y_a]$$

7. Temporal Dynamics

The following terms vary over time:

1. **Item bias** $b_i(t)$: An item's popularity might change over time.
2. **User bias** $b_u(t)$: Users change their baseline ratings over time.
3. **User preference** $p_u(t)$: users change their preferences over time as well as their perception of certain actors and directors.

Item characteristics q_i are static because items are static in nature.

Our prediction becomes

$$\hat{r}_{ui}(t) = \mu + b_i(t) + b_u(t) + q_i^\top p_u(t)$$

8. Inputs with Varying Confidence Levels

Not all observed ratings deserve the same weight or confidence. For example, massive adverting and adversarial users.

Another example is systems built around implicit feedback. A user's exact preference level is hard to quantify, so we only choose between "probably likes the product" and "probably not interested in the product". It is valuable to attach confidence scores with the estimated preferences. We can use available numerical values that describe the frequency of actions. A recurring event is more likely to reflect user opinion than a one-time event.

Let confidence be c_{ui}, we have the objective function

$$\min_{q^*, p^*, b^*} \sum_{(u,i) \in \kappa} c_{ui}(r_{ui} - \mu - b_u - b_i - q_i^\top p_u)^2 + \lambda(\|q_i\|^2 + \|p_u\|^2 + b_u^2 + b_i^2)$$

Ratings that have small confidence have small impact on the objective.