ICCS361 - week 5

# Recommender Systems

*Sunsern Cheamanunkul*

Adapted from lecture slides on recommender systems by Bing Liu from UIC

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
  - K-nearest neighbors
  - Association rules
  - Matrix factorization

# Introduction

- Recommender systems are widely used for recommending products and services to users.
- Good for two reasons:
  - Help <u>users</u> deal with the information overload by giving them recommendations of products, etc.
  - Help <u>businesses</u> to make more profits, i.e., selling more products.

# Example: Movie recommendation

- A set of users has initially rated some subset of movies (e.g., on the scale of 1 to 5) that they have already seen.
- These ratings serve as the input. The recommendation system uses these known ratings to predict the ratings that each user would give to those not rated movies by him/her.
- Recommendations of movies are then made to each user based on the predicted ratings.

# Other variations

- No rating but other attributes instead
  - Attributes about each user (e.g., age, gender, income, marital status, etc), and/or
  - Attributes about each movie (e.g., title, genre, director, leading actors or actresses, etc).
- The system will not predict ratings but predict the likelihood that a user will enjoy watching a movie.

# The Recommendation Problem

- We have a set of users $U$ and a set of items $S$ to be recommended to the users.
- Let $p$ be an utility function that measures the usefulness of item $s$ ($\in S$) to user $u$ ($\in U$), i.e.,
  - $p{:}U \times S \rightarrow R$, where $R$ is a totally ordered set (e.g., non-negative integers or real numbers in a range)
- Objective
  - Learn $p$ based on the past data
  - Use $p$ to predict the utility value of each item $s$ ($\in S$) to each user $u$ ($\in U$)

# Predictive Models

- Rating prediction – predict the rating score that a user is likely to give to an item (not seen or used before)
  - rating on an unseen movie. In this case, the utility of item *s* to user *u* is the rating given to *s* by *u*.
- Item prediction – predict a ranked list of items that a user is likely to buy or use.

# Two basic approaches

- Content-based recommendations:
  - The user will be recommended items similar to the ones the user preferred in the past
- Collaborative filtering:
  - The user will be recommended items that people with similar preferences liked in the past.
- Or, a combination of the two approaches..

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
  - K-nearest neighbors
  - Association rules
  - Matrix factorization

# Content-Based Recommendation

- Perform item recommendations by predicting the utility of items for a particular user based on how *"similar"* the items are to those that he/she liked in the past.
  - In a movie recommendation application, a movie may be represented by such features as specific actors, director, genre, subject matter, etc.
  - The user's interest or preference is also represented by the same set of features, called the **user profile**.

# Content-Based Recommendation

- Recommendations are made by comparing the *user profile* with *candidate items*
  - Note: they are presented using the same set of features.
- The top-*k* best matched or most similar items are recommended to the user.
- The simplest approach to content-based recommendation is to compute the similarity of the user profile with each item.

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
  - K-nearest neighbors
  - Association rules
  - Matrix factorization

# Collaborative filtering

- Collaborative filtering (CF) is one of the most studied and also the most widely-used recommendation approach in practice.
    - *k*-nearest neighbors
    - association rules
    - matrix factorization
- Key characteristic of CF
    - Predicts the utility of items for a user based on the items previously rated by <u>other like-minded users</u>.

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
  - K-nearest neighbors
  - Association rules
  - Matrix factorization

# K-Nearest Neighbors

- *k*NN (also known as the *memory-based approach*) utilizes the entire user-item database to generate predictions directly, i.e., no model building.
- This approach includes both
  - User-based methods
  - Item-based methods

# User-based kNN CF

- A user-based *k*NN collaborative filtering method consists of two primary phases:
  - the neighborhood formation phase and
  - the recommendation phase.
- Neighborhood formation — Figure out similar users
- Recommendation — Predict with the ratings given by the similar users weighted the similarity.

# Neighborhood formation phase

- Let the record (or *profile*) of the target user be **u** (represented as a vector), and the record of another user be **v** ($\mathbf{v} \in T$).
- The similarity between the target user, **u**, and a neighbor, **v**, can be calculated using the **Pearson's correlation coefficient**:

$$sim(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i \in C} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sqrt{\sum_{i \in C} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2} \sqrt{\sum_{i \in C} (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})^2}},$$

# Recommendation Phase

- Use the following formula to compute the rating prediction of item *i* for target user **u**

$$p(\mathbf{u}, i) = \bar{r}_{\mathbf{u}} + \frac{\sum_{\mathbf{v} \in V} sim(\mathbf{u}, \mathbf{v}) \times (r_{\mathbf{v},i} - \bar{r}_{\mathbf{v}})}{\sum_{\mathbf{v} \in V} |sim(\mathbf{u}, \mathbf{v})|}$$

where $V$ is the set of $k$ similar users, $r_{\mathbf{v},i}$ is the rating of user **v** given to item *i*,

# Issue with the user-based kNN CF

- Lack of scalability
  - Requires the real-time comparison of the target user to all user records in order to generate predictions.
- A variation of this approach that remedies this problem is called **item-based CF**.

# Item-based CF

- The item-based approach works by comparing items based on their pattern of ratings across users. The similarity of items *i* and *j* is computed as follows:

$$sim(i,j) = \frac{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})(r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})}{\sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},i} - \bar{r}_{\mathbf{u}})^2} \sqrt{\sum_{\mathbf{u} \in U} (r_{\mathbf{u},j} - \bar{r}_{\mathbf{u}})^2}}$$

# Recommendation phase

- After computing the similarity between items we select a set of *k* most similar items to the target item and generate a predicted value of user **u**'s rating

$$p(\mathbf{u}, i) = \frac{\sum_{j \in J} r_{\mathbf{u},j} \times sim(i, j)}{\sum_{j \in J} sim(i, j)}$$

where *J* is the set of *k* similar items

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
    - K-nearest neighbors
    - Association rules
    - Matrix factorization

# Association rule-based CF

- Association rules obviously can be used for recommendation.
- Each transaction for association rule mining is the set of items bought by a particular user.
- We can find item association rules, e.g.,

  buy item X, buy item Y —> buy item Z
- Items can then be ranked by *confidence*.
- More details next time

# Overview

- Introduction
- Content-based recommendation
- Collaborative filtering-based recommendation
    - K-nearest neighbors
    - Association rules
    - Matrix factorization

# Matrix factorization

- The idea of **matrix factorization** is to decompose a matrix ***M*** into the product of several factor matrices, i.e.,

$$\boldsymbol{M} = \boldsymbol{F}_1 \boldsymbol{F}_2 \ldots \boldsymbol{F}_n$$

  where *n* can be any number, but it is usually 2 or 3.

# CF using matrix factorization

- Known for its superior performance both in terms of recommendation quality and scalability.
- Part of its success is due to the **Netflix Prize contest** for movie recommendation, which popularized a Singular Value Decomposition (SVD) based matrix factorization algorithm.

# Intuition

- Matrix factorization is based on the *latent factor model*.
  - Latent variables (also called features, aspects, or factors) are introduced to account for the underlying reasons of a user purchasing or using a product.
- The connections between the latent variables and observed variables (user, product, rating, etc.) are "*learned*" during the training
- Recommendations are made to users by computing their possible interactions with each product through these latent variables.

# Netflix Prize Contest

- In 2006, Netflix (movie streaming website) announced $1M award to whoever improve its recommender system's root mean square error (RMSE) performance by 10% or more.
- Training set was 100M movie ratings on a scale of 1 to 5, submitted by 500K users on 17K movies.
- Test set was about 3M ratings.
- Greatly impact the field of recommender systems and collaborative filtering.

# Netflix Prize Task

- Training data: Quadruples of the form
  (user, movie, rating, time)
  - For our purpose here, we only use triplets, i.e., (user, movie, rating)
  - For example, (132456, 13546, 4) means that the user with ID 132456 gave the movie with ID 13546 a rating of 4 (out of 5).
- Testing: predict the rating of each triplet:
  (user, movie, ?)

# Matrix Factorization

- The technique discussed here is based on the Singular Value Decomposition (SVD) method given by
  - Simon Funk at his blog site,
  - the derivation of Funk's method described by Wagman in the Netflix forums.
  - the paper by Takacs et al.
- The method was later improved by Koren et al., Paterek and several other researchers.

# Simon Funk's method

- Differs from the standard SVD in that the singular value matrix is not used and combined into the other two matrices.
- Let R be a matrix containing 500K x 17K = 8.5 billion entries. Each non-empty entry $r_{ij}$ represents a movie rating of user i on movie j.
- His SVD method decomposes R into two matrices: U (user-aspect) and M (movie-aspect) so that

$$R \approx U^\top M$$

where $\boldsymbol{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_I]$ and $\boldsymbol{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_J]$

# Simon Funk's method

- Use $K$ = 90 latent aspects ($K$ needs to be set by cross-validation).
- Each movie will be described by only 90 aspect values indicating how much that movie exemplifies each aspect.
- Each user is also described by 90 aspect values indicating how much he/she prefers each aspect.

# Simon Funk's method

- To combine these together into a rating, we simply multiply each user aspect by the corresponding movie aspect, and then sum them up to give a rating to indicate how much that user likes that movie:

  $\boldsymbol{U} = [\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_I]$ and $\boldsymbol{M} = [\mathbf{m}_1, \mathbf{m}_2, \ldots, \mathbf{m}_J]$

- Or,

$$r_{ij} \approx \mathbf{u}_i^T \mathbf{m}_j = \sum_{k=1}^{K} u_{ki} \times m_{kj}$$

# Simon Funk's method

- SVD is a mathematical way to find these two smaller matrices which minimizes the resulting approximation error, the mean square error (MSE).
- We can use the resulting matrices ***U*** and ***M*** to predict the ratings in the test set.

$$p_{ij} = \sum_{k=1}^{K} u_{ki} \times m_{kj}$$

# SVD calculation

- The matrix R is huge but extremely sparse.
  - Total 8.5 billion cells
  - Only 100 million non-empty cells
- Traditional SVD methods wouldn't work
- Simon Funk proposed a simple incremental method for doing SVD based on the idea of stochastic gradient descent.

# Simon Funk's method

- To minimize the error, the gradient descent approach is used.
- Let the error $e_{ij} = r_{ij} - p_{ij}$
- For gradient descent, we take the partial derivative of the square error with respect to each parameter, i.e. with respect to each $u_{ki}$ and $m_{kj}$.

$$\frac{\partial (e_{ij})^2}{\partial u_{ki}} = 2e_{ij} \frac{\partial e_{ij}}{\partial u_{ki}}$$

# Simon Funk's method

Since $r_{ij}$ in Equation (12) is given in the training data and is a constant, then we have

$$\frac{\partial e_{ij}}{\partial u_{ki}} = -\frac{\partial p_{ij}}{\partial u_{ki}}. \tag{14}$$

Now since $p_{ij}$ is just a sum over $K$ terms (one for each singular vector), and only one of them is a function of $u_{ki}$, namely the term $u_{ki} \times m_{kj}$. Its derivative with respect to $u_{ki}$ is just $m_{kj}$, and the derivatives of all the other terms are zero. Thus, for the single rating by user $i$ for item $j$, and one singular vector $k$, we have

$$\frac{\partial (e_{ij})^2}{\partial u_{ki}} = 2e_{ij}(-m_{kj}) = -2(r_{ij} - p_{ij})m_{kj}. \tag{15}$$

# Simon Funk's method

If you follow the same procedure to take the partial derivative with respect to $m_{kj}$, we get

$$\frac{\partial(e_{ij})^2}{\partial m_{kj}} = 2e_{ij}(-u_{ki}) = -2(r_{ij} - p_{ij})u_{ki} . \tag{16}$$

When using gradient descent, one uses a parameter $\gamma$ called the **learning rate** as a multiplier on the gradient to use as the step to add to the parameter, so we get the following gradient descent update rule:

$$u_{ki}^{t+1} = u_{ki}^{t} - \gamma \frac{\partial(e_{ij})^2}{\partial u_{ki}} = u_{ki}^{t} + 2\gamma(r_{ij} - p_{ij})m_{kj}^{t} . \tag{17}$$

# The final update rules

- By the same reasoning, we can also compute the update rule for $m_{kj}$.
- Finally, we have both rules

$$u_{ki}^{t+1} = u_{ki}^{t} + 2\gamma(r_{ij} - p_{ij})m_{kj}^{t}. \tag{18}$$

$$m_{kj}^{t+1} = m_{kj}^{t} + 2\gamma(r_{ij} - p_{ij})u_{ki}^{t}. \tag{19}$$

# Further improvements

- Regularization term
- Variable learning rate
- Data preprocessing
  - Identify and remove outliers
  - Normalization
- Time information for each rating was also added later.