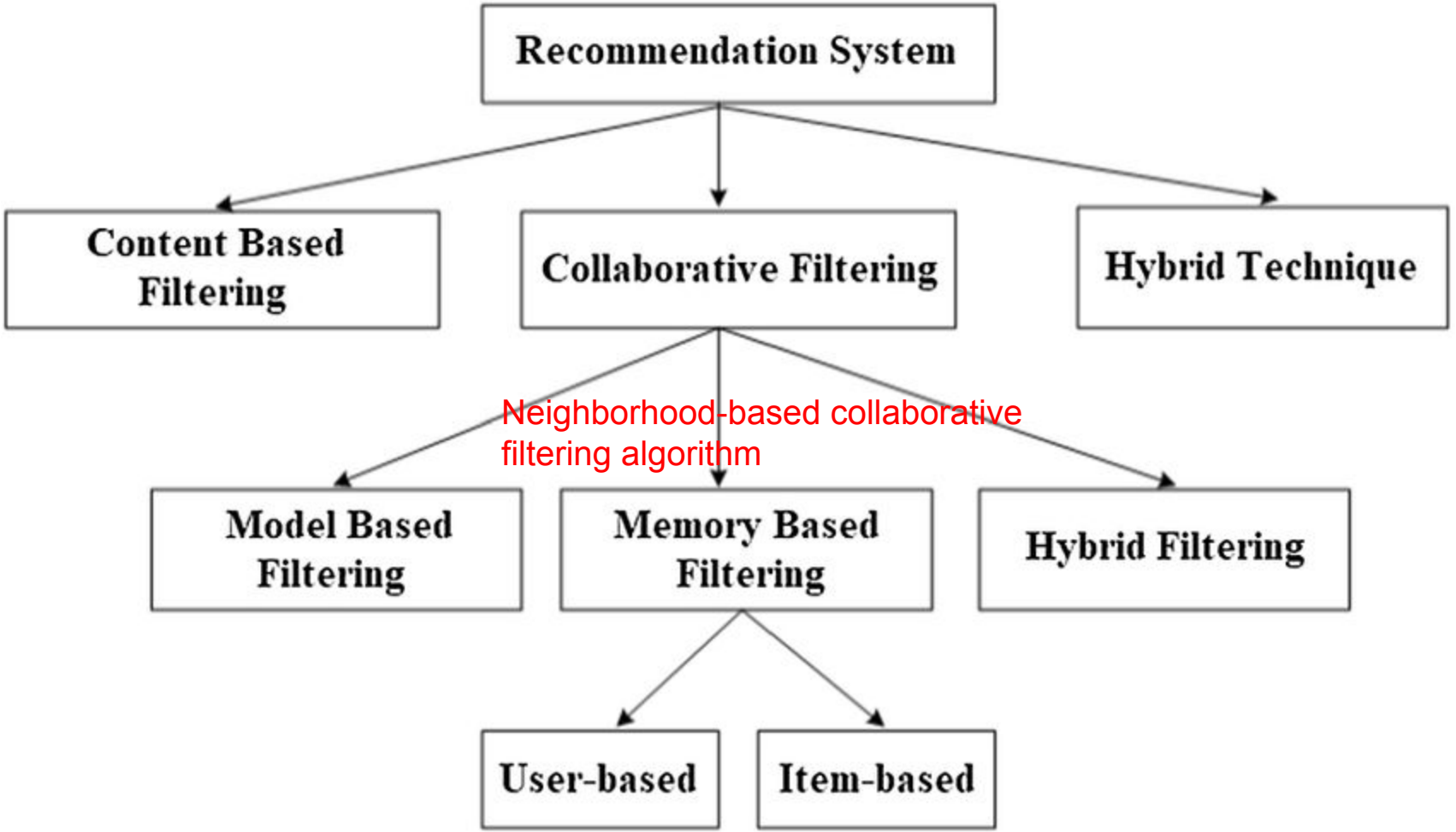
Collaborative Filtering

Architecture of Collaborative Filtering, User-based nearest neighbour recommendation, Item-based nearest neighbour recommendation, Model based and pre-processing based approaches, Clustering for recommendation system, Attacks on collaborative recommender systems, Advantages and
drawbacks of Collaborative Filtering.


Recommendation System
Content Based Filtering
Collaborative Filtering
Hybrid Technique
Neighborhood-based collaborative filtering algorithm
Model Based Filtering
Memory Based Filtering
Hybrid Filtering
User-based
Item-based

User-based collaborative filtering is a recommendation method that predicts a user's preferences by identifying similar users, known as a "peer group." The predicted ratings for target user (user A) are computed as a weighted average of the ratings given by this peer group to each item. The weights are based on the similarity between user A and other users in the group. This approach aims to recommend items to a user based on the preferences of users with similar tastes.

Neighborhood-based collaborative filtering algorithm

1. User-based collaborative filtering: In this case, the ratings provided by similar users to a target user A are used to make recommendations for A. The predicted ratings of A are computed as the weighted average values of these “peer group” ratings for each item.

Refer the last page for example

Ratings are predicted using the ratings of neighboring users, similarities among users (rows of ratings matrix),

2. Item-based collaborative filtering: In order to make recommendations for target item B, the first step is to determine a set S of items, which are most similar to item B. Then, in order to predict the rating of any particular user A for item B, the ratings in set S, which are specified by A, are determined. The weighted average of these ratings is used to compute the predicted rating of user A for item B.

Refer Notes doc for example

Ratings are predicted using the user’s own ratings on neighboring (i.e., closely related) items

we assume that the user-item ratings matrix is an incomplete m × n matrix

R = [r_{uj}] containing m users and n items.

ratings matrix is denoted by R, and it is an m × n matrix containing m users and n items. Therefore, the rating of user u for item j is denoted by [r_{uj}]

Neighborhood-based collaborative filtering algorithms can be formulated in one of two ways:

1. **Predicting the rating value of a user-item combination:** This is the simplest and most primitive formulation of a recommender system. In this case, the missing rating ruj of the user u for item j is predicted.

2. **Determining the top-k items or top-k users:** In most practical settings, the merchant is not necessarily looking for specific ratings values of user-item combinations. Rather, it is more interesting to learn the top-k most relevant items for a particular user, or the top-k most relevant users for a particular item. The problem of determining the top-k items is more common than that of finding the top-k users.

Ratings

Interval-based ratings mean using numbers to express how much you like or dislike something, but there are rules about the numbers. For example:

Numerical Values: You use numbers like 1, 2, 3, etc.
Defined Distances: The numbers have specific gaps between them, like going from 1 to 2 is considered the same as going from 4 to 5.
Equidistant: The gaps between the numbers are equal. So, the difference between 1 and 2 is the same as the difference between 3 and 4.

Imagine rating a movie from 1 to 5. If you give it a 4, it means you like it more than if you give it a 3. The idea is that the gaps between the numbers (1 to 2, 2 to 3, etc.) are the same - they're equal distances. This helps systems understand how much you like something compared to other things you've rated.

1. Continuous ratings: Jester joke recommendation engine values -10-+10

The drawback of this approach is that it creates a burden on the user of having to think of a real value from an infinite number of possibilities. Therefore, such an approach is relatively rare

2. Interval-based ratings: numerical integer values from 1 to 5, from -2 to 2, or from 1 to 7. An important assumption is that the numerical values explicitly define the distances between the ratings, and the rating values are typically equidistant

3. Ordinal ratings: “Strongly Disagree,” “Disagree,” “Neutral,” “Agree,” and “Strongly Agree.”

   based on some characteristic or attribute

4. Binary ratings: like or dislike : you tube

5. Unary ratings: like : facebook

Predicting Ratings with Neighborhood-Based Methods

User-based models

$$similarity(A, B) = cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}$$

Table 2.1: User-user similarity computation between user 3 and other users

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6	Mean Rating	Cosine$(i, 3)$ (user-user)	Pearson$(i, 3)$ (user-user)
1	7	6	7	4	5	4			
2	6	7	?	4	3	4			
3	?	3	3	1	1	?			
4	1	2	2	3	3	4			
5	1	?	1	2	3	3			

Table 2.1: User-user similarity computation between user 3 and other users

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6	Mean Rating	Cosine$(i, 3)$ (user-user)	Pearson$(i, 3)$ (user-user)
1	7	6	7	4	5	4	5.5	0.956	0.894
2	6	7	?	4	3	4	4.8	0.981	0.939
3	?	3	3	1	1	?	2	1.0	1.0
4	1	2	2	3	3	4	2.5	0.789	-1.0
5	1	?	1	2	3	3	2	0.645	-0.817

2. Item-based models:

Table 2.2: Ratings matrix of Table 2.1 with mean-centering for adjusted cosine similarity computation among items. The adjusted cosine similarities of items 1 and 6 with other items are shown in the last two rows.

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	-1	0	1	1

2. Item-based models:

Table 2.2: Ratings matrix of Table 2.1 with mean-centering for adjusted cosine similarity computation among items. The adjusted cosine similarities of items 1 and 6 with other items are shown in the last two rows.

Item-Id ⇒ / User-Id ⇓	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	-1	0	1	1
Cosine$(1, j)$ (item-item)	1	0.735	0.912	-0.848	-0.813	-0.990
Cosine$(6, j)$	-0.990	-0.622	-0.912	0.829	0.730	1

Pearson correlation coefficient

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}}$$

r = correlation coefficient

x_i = values of the x-variable in a sample

$\bar{x}$ = mean of the values of the x-variable

y_i = values of the y-variable in a sample

$\bar{y}$ = mean of the values of the y-variable

User-Based Neighborhood Models

In order to determine the neighborhood of the target user i, her similarity to all the other users is computed. Therefore, a similarity function needs to be defined between the ratings specified by users. Such a similarity computation is tricky because different users may have different scales of ratings. One user might be biased toward liking most items, whereas another user might be biased toward not liking most of the items. Furthermore, different users may have rated different items. Therefore, mechanisms need to be identified to address these issues.

User-Based Neighborhood Models

I_u = {1, 3, 5}. Therefore, the set of items rated by both users u and v is given by I_u ∩ Iv. For example, if user v has rated the first four items, then I_v = {1, 2, 3, 4}, and $I_u \cap I_v$ = {1, 3, 5}∩{1, 2, 3, 4} = {1, 3}

One measure that captures the similarity Sim(u, v) between the rating vectors of two users u and v is the Pearson correlation coefficient. Because Iu ∩Iv represents the set of item indices for which both user u and user v have specified ratings, the coefficient is computed only on this set of items.

User-Based Neighborhood Models

The **first step** is to compute the mean rating μu for each user u using her specified ratings:

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \ldots m\}$$

Second step

Then, the Pearson correlation coefficient between the rows (users) u and v is defined as follows:

$$\text{Sim}(u, v) = \text{Pearson}(u, v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}} \tag{2.2}$$

User-Based Neighborhood Models

The Pearson coefficient is computed between the target user and all the other users. One way of defining the peer group of the target user would be to use the set of k users with the highest Pearson coefficient with the target. However, since the number of observed ratings in the top-k peer group of a target user may vary significantly with the item at hand, the closest k users are found for the target user separately for each predicted item, such that each of these k users have specified ratings for that item

User-Based Neighborhood Models

The main problem with this approach is that different users may provide ratings on different scales. One user might rate all items highly, whereas another user might rate all items negatively. The raw ratings, therefore, need to be mean-centered in row-wise fashion, before determining the (weighted) average rating of the peer group. The mean-centered rating s_{uj} of a user u for item j is defined by subtracting her mean rating from the raw rating r_{uj} .

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \ldots m\}$$

As before, the weighted average of the mean-centered rating of an item in the top-k peer group of target user u is used to provide a *mean-centered* prediction. The mean rating of the target user is then added back to this prediction to provide a *raw* rating prediction $\hat{r}_{uj}$ of target user u for item j. The hat notation "^" on top of r_{uj} indicates a *predicted* rating, as opposed to one that was already observed in the original ratings matrix. Let $P_u(j)$ be the set[1] of k closest users to target user u, who have specified ratings for item j. Users with very low or negative correlations with target user u are sometimes filtered from $P_u(j)$ as a heuristic enhancement. Then, the overall neighborhood-based *prediction function* is as follows:

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u, v)|} \tag{2.4}$$

This broader approach allows for a number of different variations in terms of how the similarity or prediction function is computed or in terms of which items are filtered out during the prediction process.

2.3.1.1 Similarity Function Variants

Several other variants of the similarity function are used in practice. One variant is to use the cosine function on the *raw* ratings rather than the mean-centered ratings:

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u \cap I_v} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} r_{vk}^2}} \tag{2.5}$$

In some implementations of the raw cosine, the normalization factors in the denominator are based on all the specified items and not the mutually rated items.

$$\text{RawCosine}(u, v) = \frac{\sum_{k \in I_u \cap I_v} r_{uk} \cdot r_{vk}}{\sqrt{\sum_{k \in I_u} r_{uk}^2} \cdot \sqrt{\sum_{k \in I_v} r_{vk}^2}} \tag{2.6}$$

In general, the Pearson correlation coefficient is preferable to the raw cosine because of the *bias adjustment* effect of mean-centering. This adjustment accounts for the fact that different users exhibit different levels of generosity in their global rating patterns.

The reliability of the similarity function $\text{Sim}(u, v)$ is often affected by the number of common ratings $|I_u \cap I_v|$ between users u and v. When the two users have only a small number of ratings in common, the similarity function should be reduced with a discount factor to de-emphasize the importance of that user pair. This method is referred to as *significance weighting*. The discount factor kicks in when the number of common ratings

2.3.2 Item-Based Neighborhood Models

In item-based models, peer groups are constructed in terms of *items* rather than *users*. Therefore, similarities need to be computed between items (or columns in the ratings matrix). Before computing the similarities between the columns, each row of the ratings matrix is centered to a mean of zero. As in the case of user-based ratings, the average rating of each item in the ratings matrix is subtracted from each rating to create a mean-centered matrix. This process is identical to that discussed earlier (see Equation 2.3), which results in the computation of mean-centered ratings s_{uj}. Let U_i be the indices of the set of users who have specified ratings for item i. Therefore, if the first, third, and fourth users have specified ratings for item i, then we have $U_i = \{1, 3, 4\}$.

Then, the *adjusted* cosine similarity between the items (columns) i and j is defined as follows:

$$\text{AdjustedCosine}(i, j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}} \tag{2.14}$$

This similarity is referred to as the adjusted cosine similarity because the ratings are mean-centered before computing the similarity value. Although the Pearson correlation can also be used on the columns in the case of the item-based method, the adjusted cosine generally provides superior results.

First, the similarity between items are computed after adjusting for mean-centering. The mean-centered ratings matrix is illustrated in Table 2.2. The corresponding adjusted cosine similarities of each item to 1 and 6, respectively, are indicated in the final two rows of the table. For example, the value of the adjusted cosine between items 1 and 3, denoted by AdjustedCosine(1, 3), is as follows:

$$\text{AdjustedCosine}(1,3) = \frac{1.5 * 1.5 + (-1.5) * (-0.5) + (-1) * (-1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \cdot \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}} = 0.912$$

Other item-item similarities are computed in an exactly analogous way, and are illustrated in the final two rows of Table 2.2. It is evident that items 2 and 3 are most similar to item 1, whereas items 4 and 5 are most similar to item 6. Therefore, the weighted average of the *raw* ratings of user 3 for items 2 and 3 is used to predict the rating $\hat{r}_{31}$ of item 1, whereas the weighted average of the raw ratings of user 3 for items 4 and 5 is used to predict the rating $\hat{r}_{36}$ of item 6:

$$\hat{r}_{31} = \frac{3 * 0.735 + 3 * 0.912}{0.735 + 0.912} = 3$$

$$\hat{r}_{36} = \frac{1 * 0.829 + 1 * 0.730}{0.829 + 0.730} = 1$$

Efficient Implementation and Computational Complexity

- Neighborhood-based methods are always used to determine the best item recommendations for a target user or the best user recommendations for a target item.
- Neighborhood-based methods are always partitioned into an offline phase and an online phase.
- Neighborhood-based methods are always partitioned into an offline phase and an online phase. In the offline phase, the user-user (or item-item) similarity values and peer groups of the users (or items) are computed. For each user (or item), the relevant peer group is prestored on the basis of this computation. In the online phase, these similarity values and peer groups are leveraged to make predictions with the use of relationships

Time and space complexity

In the case of user-based methods, the process of determining the peer group of a target user may require $O(m \cdot n`)$ time. Therefore, the offline running time for computing the peer groups of all users is given by $O(m^2 \cdot n`)$. For item-based methods, the corresponding offline running time is given by $O(n^2 \cdot m`)$.

the space requirements of user-based methods are $O(m^2)$, whereas the space requirements of item-based methods are $O(n^2)$. Because the number of users is typically greater than the number of items, the space requirements of user-based methods are generally greater than those of item-based methods.

Clustering and Neighborhood-Based Methods

The main problem with neighborhood-based methods is the complexity of the offline phase, which can be quite significant when the number of users or the number of items is very large. For example, when the number of users m is of the order of a few hundred million, the $O(m^2 \cdot n')$ running time of a user-based method will become impractical even for occasional offline computation. Consider the case where $m = 10^8$ and $n' = 100$. In such a case, $O(m^2 \cdot n') = O(10^{18})$ operations will be required. If we make the conservative assumption that each operation requires an elementary machine cycle, a 10GHz computer will require 10^8 seconds, which is approximately 115.74 days. Clearly, such an approach will not be very practical from a scalability point of view.

clustering-based methods

The main idea of clustering-based methods is to replace the offline nearest-neighbor computation phase with an offline clustering phase. Just as the offline nearest-neighbor phase creates a large number of peer groups, which are centered at each possible target, the clustering process creates a smaller number of peer groups which are not necessarily centered at each possible target. The process of clustering is much more efficient than the $O(m^2 \cdot n`)$ time required for construction of the peer groups of every possible target. Once the clusters have been constructed, the process of predicting ratings is similar to the approach used in Equation

Therefore, the clusters are dependent on the representatives and vice versa. Such an interdependency is achieved with an iterative approach. We start with a set of representatives $\overline{Y_1} \ldots \overline{Y_k}$, which might be randomly chosen points generated in the range of the data space. We iteratively compute the cluster partitions using the representatives, and then recompute the representatives as the centroids of the resulting clusters. While computing the centroids, care must be taken to use only the observed values in each dimension. This two-step iterative approach is executed to convergence. The two-step approach is summarized as follows:

1. Determine the clusters $\mathcal{C}_1 \ldots \mathcal{C}_k$ by assigning each row in the $m \times n$ matrix to its closest representative from $\overline{Y_1} \ldots \overline{Y_k}$. Typically, the Euclidean distance or the Manhattan distance is used for similarity computation.

2. For each $i \in \{1 \ldots k\}$, reset $\overline{Y_i}$ to the centroid of the current set of points in $\mathcal{C}_i$.

Model-Based Collaborative Filtering

- Decision and Regression trees
- Rule based collaborative filtering
- Naive Baise collaborative filtering
- Using an Arbitrary Classification Model as a Black-Box .
- Latent Factor Models .

Decision and Regression Trees

Decision and regression trees are frequently used in data classification. Decision trees are designed for those cases in which the dependent variable is categorical, whereas regression trees are designed for those cases in which the dependent variable is numerical. Before discussing the generalization of decision trees to collaborative filtering, we will first discuss the application of decision trees to classification

Decision Trees for Recommendations: Think of the decision tree like a quiz, asking questions about the user's preferences to narrow down options.
Lists, Not Singles: Instead of recommending one perfect item, the end of the quiz has a list of suitable items. .

Decision and Regression Trees

- the decision tree produces lists of recommended items at its leaf nodes, instead of single items. This leads to reduced amount of search, when using the tree to compile a recommendation list for a user and consequently enables a scaling of the recommendation system.

Algorithm

- The decision tree forms a predictive model which maps the input to a predicted value based on the input's attributes. Each interior node in the tree corresponds to an attribute and each arc from a parent to a child node represents a possible value or a set of values of that attribute.

Decision tree: Imagine a flowchart with questions and answers like a choose-your-own-adventure book.
Predictive model: It uses this flowchart to predict something (the "predicted value") based on the information you provide (the "input").
Attributes: This is the information you provide, like your age, gender, or shopping preferences.
Interior node & arcs: These are like the questions and answers in the flowchart. Each question (node) focuses on a specific attribute, and the answers (arcs) lead to different parts of the flowchart based on your selection.
So, the decision tree acts like a map, using the information you give it (attributes) to guide it towards a predicted outcome (value). The questions (nodes) and answers (arcs) help navigate this map and arrive at the final prediction.

Building the tree: We start by making the tree's beginning (root node) and adding all the possible information (input set).
First question: We pick the most important question to ask at the start (attribute).
Making branches: For every possible answer (value) to the question, we create a new path (arc) and a smaller section (sub-node).

algo…

- The construction of the tree begins with a root node and the input set. An attribute is assigned to the root and arcs and sub-nodes for each set of values are created.

- The input set is then split by the values so that each child node receives only the part of the input set which matches the attribute value as specified by the arc to the child node.

Start with a big box (root node). Put all the toys in this box.
Choose a question to ask (attribute). For example, "Is the toy red or blue?"
Create two smaller boxes (child nodes). Label one "Red" and the other "Blue."
Go through each toy in the big box. If the toy is red, put it in the red box. If it's blue, put it in the blue box.
Now, each small box (child node) only holds the toys (input set) that match the question's answer (attribute

- The process then repeats itself recursively for each child until splitting is no longer feasible.

Left Tree: Collaborative Filtering (CF)

Focus: User Preferences. The tree recommends movies based on whether other users with similar tastes liked " The Usual Suspects".
Key Questions: Did you like "The Godfather", "Pulp Fiction", etc.?
Outcome: A leaf node is marked 'L' (like) or 'D' (dislike), suggesting whether you'd enjoy "The Usual Suspects" based on your similarities with other users.

Key Differences

CF (Collaborative): Recommendations are driven by your similarity to other users.
CB (Content-Based): Recommendations are based on the movie's specific attributes and your past likes.
Which Is Better?

Neither is truly better. They have different strengths:

CF: Great for suggesting movies outside your usual genres, based on what similar people like.
CB: Good for finding movies similar to ones you already enjoy, focusing on specific features.

Right Tree: Content-Based (CB)

Focus: Movie Features. The tree recommends movies based on the characteristics of "The Usual Suspects".
Key Questions: Is it a crime drama? Who stars in it? When was it released?
Outcome: A leaf node is marked 'L' (like) or 'D' (dislike), suggesting whether you'd enjoy the movie based on its specific features.

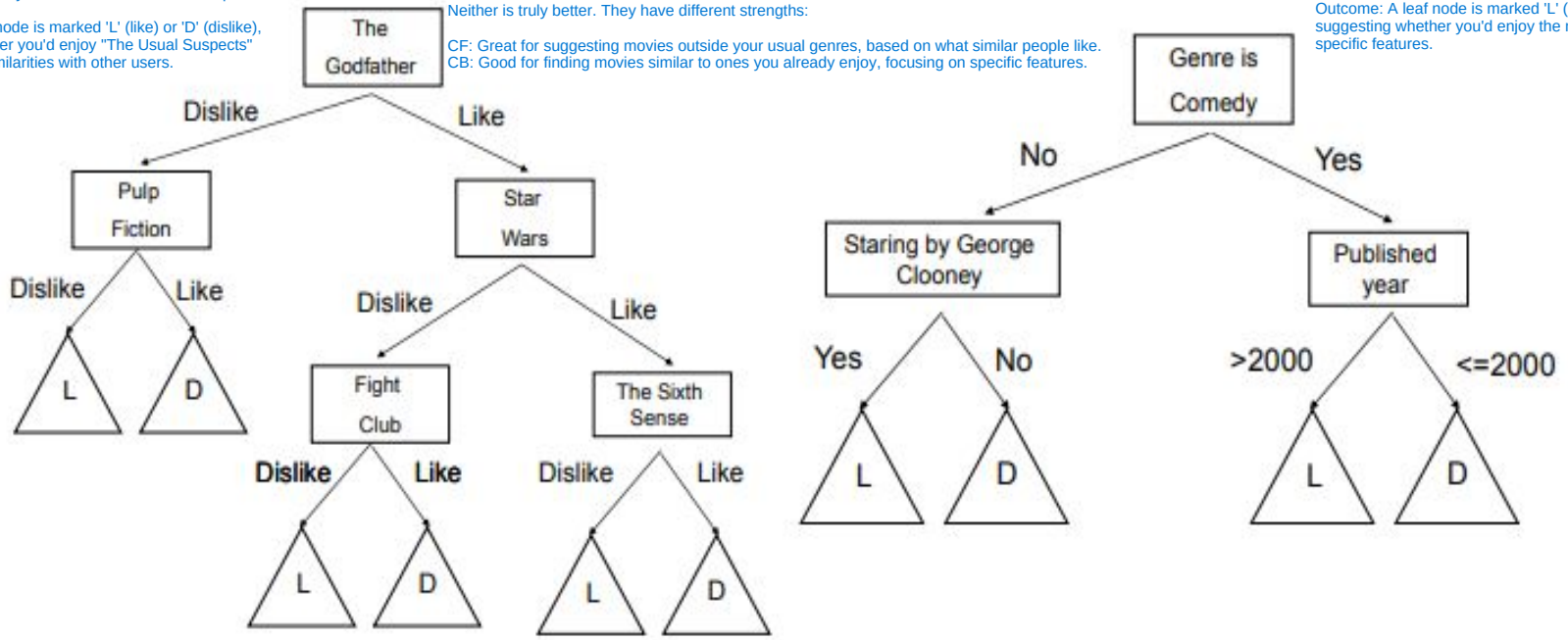


Figure 1: Left: A CF decision tree for whether users like the movie "The Usual Suspects" based on their preferences to other movies such as The Godfather, Pulp Fiction etc. A leaf labeled with "L" or "D" correspondingly indicates that the user likes/dislikes the movie "The Usual Suspects". Right: A CB decision tree for Bob.

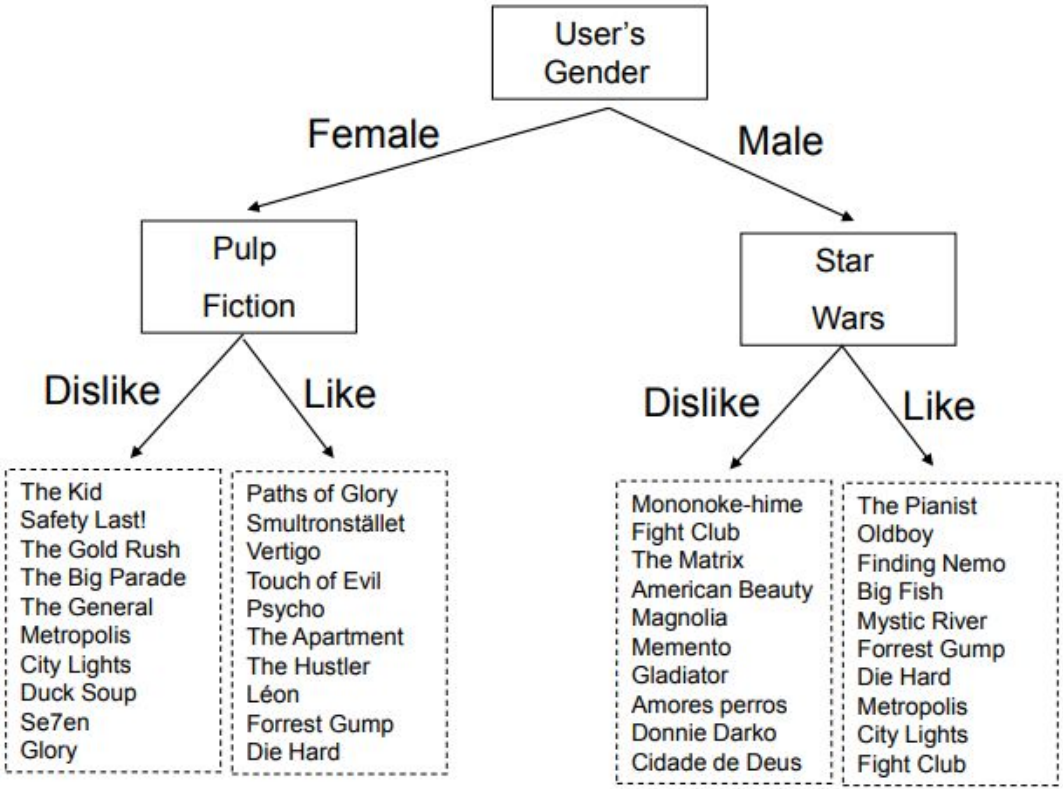


Figure 2: An example CF-CB hybrid decision tree

Consider the case in which we have an m×n matrix R. Without loss of generality, assume that the first (n − 1) columns are the independent variables, and the final column is the dependent variable. For ease in discussion, assume that all variables are binary. Therefore, we will discuss the creation of a decision tree rather than a regression tree

1	2	3	Dependent Variable
0	1	0	1
1	0	1	0
1	1	0	0
0	0	1	1
0	1	1	0

Decision trees work by splitting data into smaller subsets based on the values of different features (like age, income, location,

Gini Index: This is a measure of impurity in a dataset or a subset of data. It helps determine the best way to split the data at each step of the decision

A Smaller Gini Index is Better: A Gini Index value of 0 indicates perfect purity (all data points in the node belong to the same class). Conversely, larger Gini values mean greater impurity within a node.

The quality of the split can be evaluated by using the weighted average *Gini index* of the child nodes created from a split. If $p_1 \ldots p_r$ are the fractions of data records belonging to r different classes in a node S, then the Gini index $G(S)$ of the node is defined as follows:

$$G(S) = 1 - \sum_{i=1}^{r} p_i^2 \tag{3.1}$$

The Gini index lies between 0 and 1, with smaller values being more indicative of greater discriminative power. The overall Gini index of a split is equal to the weighted average of the Gini index of the children nodes. Here, the weight of a node is defined by the number of data points in it. Therefore, if S_1 and S_2 are the two children of node S in a binary decision tree, with n_1 and n_2 data records, respectively, then the Gini index of the split $S \Rightarrow (S_1, S_2)$ may be evaluated as follows:

$$\text{Gini}\,(S \Rightarrow [S_1, S_2]) = \frac{n_1 \cdot G(S_1) + n_2 \cdot G(S_2)}{n_1 + n_2} \tag{3.2}$$

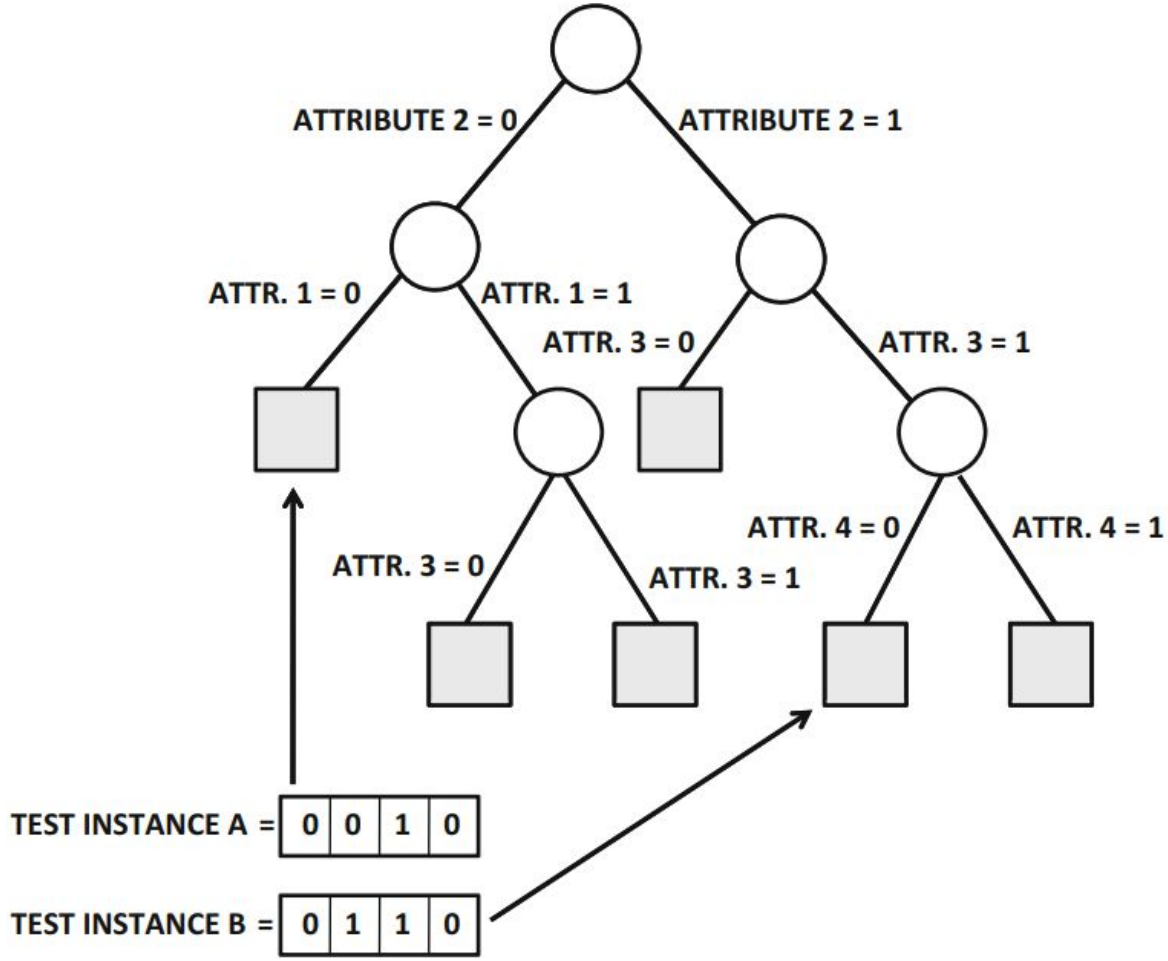


Figure 3.2: Example of a decision tree with four binary attributes

Weakness

- A major weakness in using decision trees as a prediction model in RS is the need to build a huge number of trees (either for each item or for each user).
- Moreover, the model can only compute the expected rating of a single item at a time. To provide recommendations to the user, we must traverse the tree(s) from root to leaf once for each item in order to compute its predicted rating.
- Only after computing the predicted rating of all items can the RS provide the recommendations (highest predicted rating items). Thus decision trees in RS do not scale well with respect to the number of items.

- The main challenge in extending decision trees to collaborative filtering is that the predicted entries and the observed entries are not clearly separated in column-wise fashion as feature and class variables. Furthermore, the ratings matrix is very sparsely populated, where the majority of entries are missing. This creates challenges in hierarchically partitioning the training data during the tree-building phase.

1. Mixing Predicted and Observed Values:

Traditional decision trees rely on having clear separation between features (independent variables) and class variables (dependent variable). These are usually represented in separate columns of the data matrix.
In CF, however, the data itself is the rating matrix, which combines both observed ratings made by users for items and predicted ratings the model needs to learn.
This fusion of observed and predicted values makes it difficult to directly apply traditional decision tree algorithms.
2. Sparsity of the Rating Matrix:

The rating matrix is typically very sparse, meaning most entries are missing because users don't rate every item.
This sparsity poses challenges during hierarchical partitioning in the decision tree building process.
Decision trees typically split data based on features, but when most entries are missing, splitting becomes unreliable and can lead to inaccurate predictions.
These two challenges raise difficulties in adapting conventional decision trees for collaborative filtering tasks.

the concept of association rules emerged from the desire to uncover relationships between items in supermarket data. This involved discovering frequently purchased items together, which laid the groundwork for analyzing customer behavior and purchase patterns.
Bridging the Gap: This idea of identifying relationships between items became a springboard for developing collaborative filtering techniques.
CF aims to predict items a user might like based on their past preferences and the preferences of similar users. This essentially involves finding connections between users and items based on past interactions, similar to uncovering links between items in supermarket data.
Shared Focus: Both association rules and CF share a core focus on identifying patterns and relationships within data.
While association rules primarily focus on uncovering connections between items, CF utilizes these connections between users and items to make personalized recommendations.

Rule-Based Collaborative Filtering

- The relationship between association rules and collaborative filtering is a natural one because the association rule problem was first proposed in the context of discovering relationships between supermarket data.

Definition 3.3.3 (Association Rules) *A rule $X \Rightarrow Y$ is said to be an association rule at a minimum support of s and minimum confidence of c, if the following two conditions are satisfied:*

1. *The support of $X \cup Y$ is at least s.*
2. *The confidence of $X \Rightarrow Y$ is at least c.*

an association rule is a statement that reveals a relationship or pattern between different items or events within a dataset. It typically follows the format: "If X, then Y," where:

X represents a set of items or conditions (called the antecedent).
Y represents another item or condition (called the consequent).
The association rule essentially suggests that the presence of X increases the likelihood of Y occurring. For example, an association rule might state: "If a customer buys bread, then they are also likely to buy milk."

Key Properties of Association Rules:

Support: This metric indicates the frequency of both the antecedent and consequent occurring together within the dataset. It reflects how common the association is.
Confidence: This metric represents the probability of the consequent (Y) occurring, given that the antecedent (X) is already present. It reflects the strength of the association.

Therefore, such rules could appear as follows:

$$(Item = Bread, Rating = Like) \Rightarrow (Item = Eggs, Rating = Like)$$

$$(Item = Bread, Rating = Like) \text{ AND } (Item = Fish, Rating = Dislike) \Rightarrow (Item = Eggs, Rating = Dislike)$$

Naive Bayes Collaborative Filtering

We will assume that there are a small number of distinct ratings, each of which can be treated as a categorical value. Therefore, the orderings among the ratings will be ignored in the following discussion. For example, three ratings, such as Like, Neutral, and Dislike, will be treated as unordered discrete values. In the case where the number of distinct ratings is small, such an approximation can be reasonably used without significant loss in accuracy.

The naive Bayes model is a generative model, which is commonly used for classification.

Assume that there are l distinct values of the ratings, which are denoted by v1 ...vl. As in the case of the other models discussed in this chapter, we assume that we have an m × n matrix R containing the ratings of m users for n items. The (u, j)th entry of the matrix is denoted by r_{uj} .

Consider the uth user, who has specified ratings for the set of items I_u. In other words, if the uth row has specified ratings for the first, third, and fifth columns, then we have $I_i = \{1, 3, 5\}$. Consider the case where the Bayes classifier needs to predict the unobserved rating r_{uj} of user u for item j. Note that r_{uj} can take on any one of the discrete possibilities in $\{v_1 \ldots v_l\}$. Therefore, we would like to determine the probability that r_{uj} takes on any of these values *conditional on the observed ratings in* I_u. Therefore, for each value of $s \in \{1 \ldots l\}$, we would like to determine the probability $P(r_{uj} = v_s | \textit{Observed ratings in } I_u)$. This expression appears in the form $P(A|B)$, where A and B are events corresponding to the value of r_{uj}, and the values of the observed ratings in I_u, respectively. The expression can be simplified using the well-known Bayes rule in probability theory:

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)} \tag{3.3}$$

Therefore, for each value of $s \in \{1 \ldots l\}$, we have the following:

$$P(r_{uj} = v_s | \textit{Observed ratings in } I_u) = \frac{P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)}{P(\textit{Observed ratings in } I_u)}$$

This estimate of the posterior probability of the rating r_{uj} can be used to estimate its value in one of the following two ways:

1. By computing each of the expressions on the right-hand side of Equation 3.7 for each $s \in \{1 \ldots l\}$, and determining the value of s at which it is the largest, one can determine the most likely value $\hat{r}_{uj}$ of the missing rating r_{uj}. In other words, we have:

$$\begin{aligned}\hat{r}_{uj} &= \text{argmax}_{v_s} P(r_{uj} = v_s | \textit{Observed ratings in } I_u) \\ &= \text{argmax}_{v_s} P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)\end{aligned}$$

   Such an approach, however, treats a rating purely as a categorical value and ignores all ordering among the various ratings. When the number of possible ratings is small, this is a reasonable approach to use.

2. Rather than determining the rating that takes on the maximum probability, one can estimate the predicted value as the weighted average of all the ratings, where the weight of a rating is its probability. In other words, the weight of the rating v_s is proportional to the value of $P(r_{uj} = v_s | \textit{Observed ratings in } I_u)$, as computed in Equation 3.7. Note that the constant of proportionality in the equation is irrelevant for computing the weighted average. Therefore, the estimated value $\hat{r}_{uj}$ of the missing rating r_{uj} in the matrix R is as follows:

$$\begin{aligned}\hat{r}_{uj} &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s | \textit{Observed ratings in } I_u)}{\sum_{s=1}^{l} P(r_{uj} = v_s | \textit{Observed ratings in } I_u)} \\ &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)}{\sum_{s=1}^{l} P(r_{uj} = v_s) \cdot P(\textit{Observed ratings in } I_u | r_{uj} = v_s)} \\ &= \frac{\sum_{s=1}^{l} v_s \cdot P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)}{\sum_{s=1}^{l} P(r_{uj} = v_s) \cdot \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)}\end{aligned}$$

   This approach is preferable when the granularity of the ratings distribution is greater.

pre-processing based approaches,

1. (Iterative step 1): Use algorithm A to estimate the missing entries of each column by setting it as the target variable and the remaining columns as the feature variables. For the remaining columns, use the current set of filled in values to create a complete matrix of feature variables. The observed ratings in the target column are used for training, and the missing ratings are predicted.

2. (Iterative step 2): Update all the missing entries based on the prediction of algorithm A on each target column.

Initialization

Filling Missing Entries: Begin by filling in missing entries of the matrix. This can be done with:

Row or Column Averages: Calculate the average for each row (user) or column (item) and use it to fill missing entries within the row/column.
Simple Collaborative Filtering Algorithm: Employ a basic user-based or item-based CF technique to give initial estimates for missing entries.
(Optional) User-Bias Removal: This step aims to account for differences in rating tendencies among users:

Centering Rows: For each row (user), subtract the user's average rating from their individual ratings. This centers their ratings around 0.
Storing User Bias: Store each user's average rating, as this bias will need to be added back in later.
Key Points About Initialization:

"Artificial Values": The filled-in values during the initialization process are merely estimates and likely have biases.
User Bias: Even if you remove user bias, the simple filling strategies will likely introduce other biases.
Two-Step Iterative Refinement:

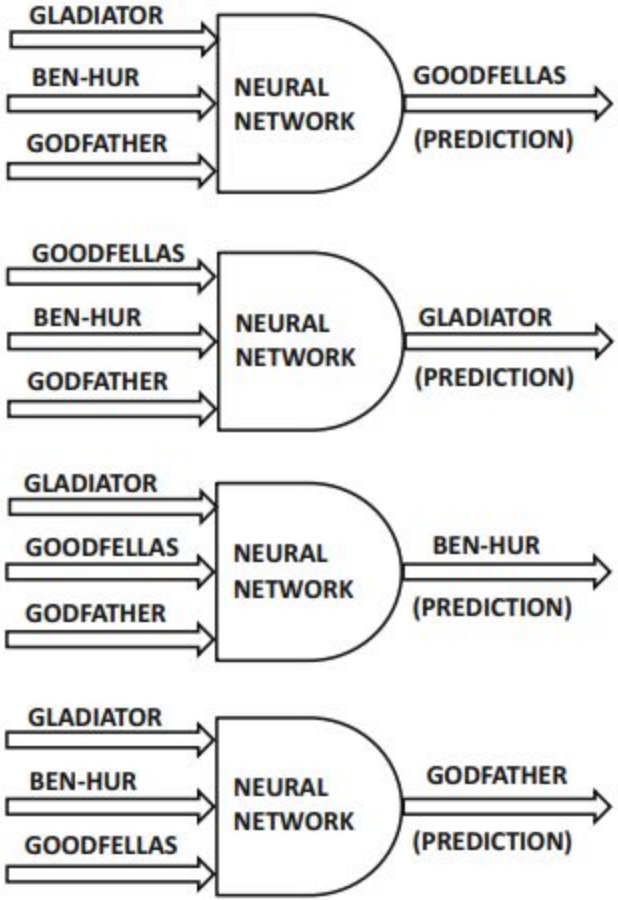
This aims to reduce the biases introduced during initialization and improve the accuracy of predictions:

Step 1: Train your primary classification/regression model (algorithm A) using the complete matrix (with filled-in values). Make predictions for the originally missing entries that were filled in the initialization phase.
Step 2: Replace the "artificial" values in the matrix with the predictions generated by algorithm A in step 1.
Iteration: Repeat step 1 and 2. Each iteration potentially improves the predictions, reducing the bias introduced initially.

Re-Adding User Bias:

If you opted for user-bias removal in initialization, after the iterative process you need to add back the stored user biases to your final predictions.
Why Does This Work?

Your main modeling algorithm (A) can likely handle some inaccuracies in the data.
Iteratively replacing filled-in values with the main algorithm's own predictions progressively refines the matrix, reducing the impact of initial crude estimations.



Consider an arbitrary classification/regression modeling algorithm A, which is designed to work with a completely specified matrix.

The first step is to initialize the missing entries in the matrix with row averages, column averages, or with any simple collaborative filtering algorithm.

For example, one might use a simple user-based algorithm for the initialization process. As an optional enhancement, one might center each row of the ratings matrix as a preprocessing step to remove user bias.

In this case, the bias of each user needs to be added back to the predicted values in a post-processing phase. Removing user bias during pre-processing often makes the approach more robust. If the user bias is removed, then the missing entries are always filled in with row averages, which are 0. These simple initializations and bias removal methods will still lead to prediction bias, when one attempts to use the artificially filled in values as training data. Then, the bias in the predicted entries can be iteratively reduced by using the following two-step iterative approach:

Sparsity: Recommender systems typically deal with sparse rating matrices, meaning most entries are missing due to users not rating all items.
LFMs excel at handling sparsity: They can learn underlying factors or latent variables that capture the relationships between users and items, even with missing data.
Better Predictions: By capturing these underlying factors, LFMs can make more accurate predictions about user preferences for unrated items compared to simpler methods like filling missing entries with averages.
Leveraging Dimensionality Reduction:
High-Dimensional Data: The rating matrix, with each user and item representing a dimension, can be considered high-dimensional data.
Dimensionality Reduction Techniques: These techniques, like matrix factorization (MF), aim to represent the data in a lower number of dimensions while preserving the essential information.

Latent factor models

Benefits in Recommender Systems: By reducing dimensionality, LFMs can:
Reduce the impact of noise in the data.
Improve the efficiency of calculations for making recommendations.
Capture the underlying structure of user-item relationships even with sparse data.
Similarities to Other Areas: As you mentioned, dimensionality reduction methods are valuable tools in various data analysis fields beyond recommender systems. They are used to: Visualize high-dimensional data in lower dimensions, Identify patterns and structures & Reduce computational complexity by working with a smaller number of dimensions.

Latent factor models are considered to be state-of-the-art in recommender systems. These models leverage well-known dimensionality reduction methods to fill in the missing entries. Dimensionality reduction methods are used commonly in other areas of data analytics to represent the underlying data in a small number of dimensions.

Overall, Latent Factor Models are powerful tools that leverage the strengths of dimensionality reduction to effectively address the challenges of sparse data and make accurate recommendations in recommender systems.

Challenges of High Dimensionality: Analyzing and handling high-dimensional data can be challenging due to:

Increased complexity: As dimensionality increases, the complexity of calculations and algorithms also increases.
Curse of dimensionality: With more dimensions, the amount of data needed to accurately model the relationships between variables grows exponentially.

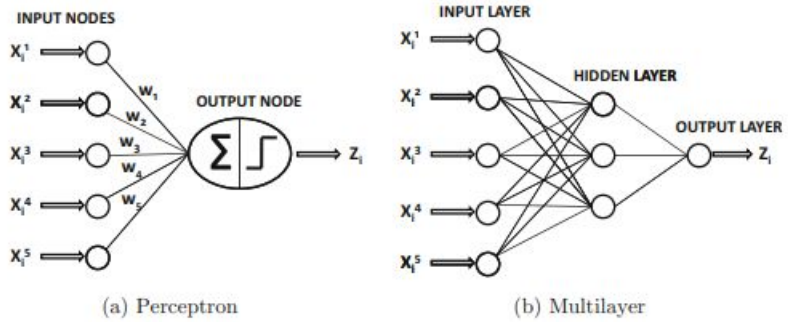
The key idea in dimensionality reduction methods is that the reduced, rotated, and completely specified representation can be robustly estimated from an incomplete data matrix.

Robust Estimation: This estimation involves finding the underlying factors or latent variables that capture the relationships between users and items. These latent factors can be estimated even with missing data, making LFMs effective for recommender systems.

$$z_i = \text{sign}\{\overline{W} \cdot \overline{X_i} + b\} \quad (3.9)$$

b is bias

Using a Neural Network as a Black-Box



(a) Perceptron (b) Multilayer

Figure 3.3: Single and multilayer neural networks

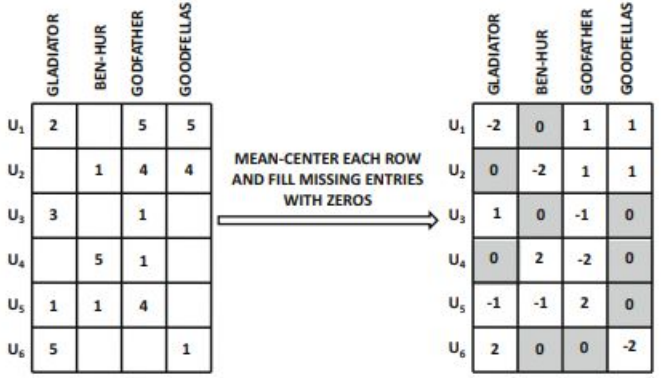


Figure 3.4: Pre-processing the ratings matrix. Shaded entries are iteratively updated.

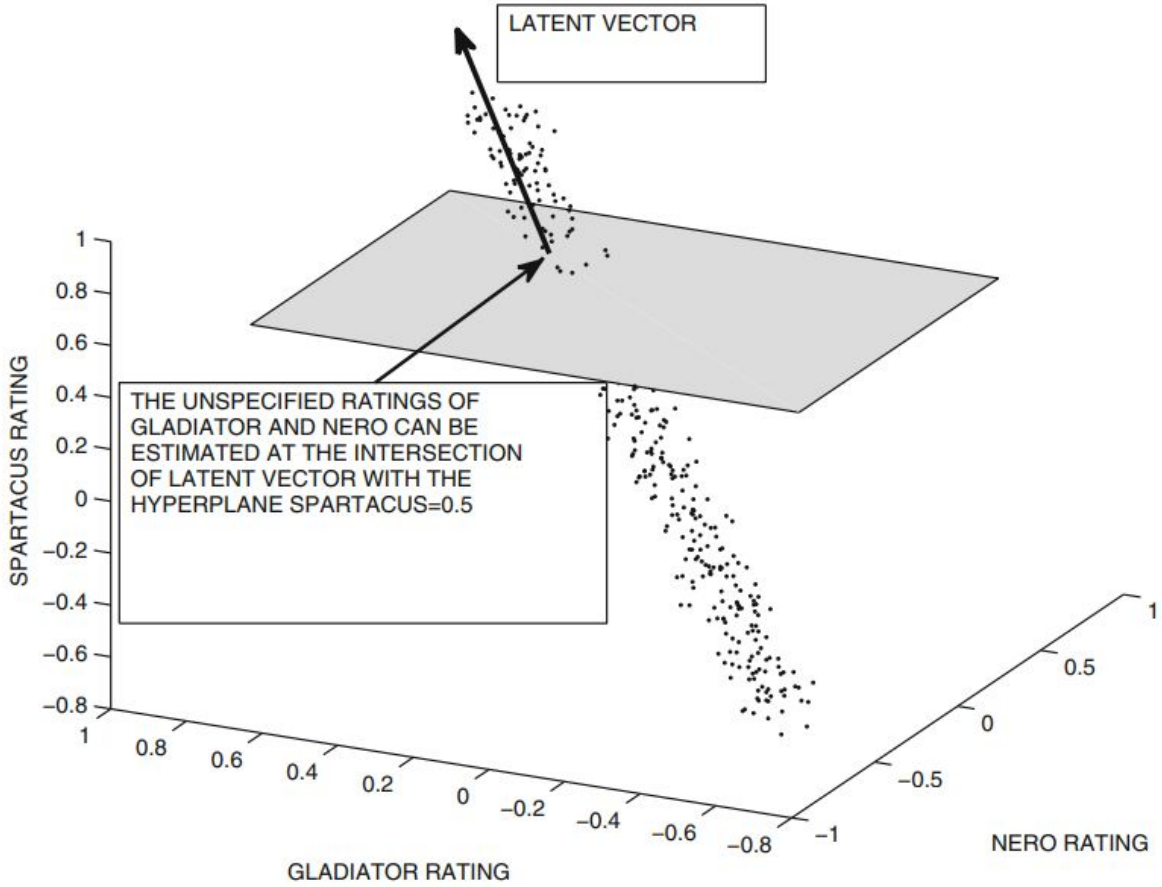


Figure 3.6: Leveraging correlation-based redundancies in missing data estimation for a user whose only specified rating is a value of 0.5 for the movie *Spartacus*

ATTACKS

Unfortunately, there are significant motivations for participants to submit incorrect feedback about items for personal gain or for malicious reasons:

• The manufacturer of an item or the author of a book might submit fake (positive) reviews on Amazon in order to maximize sales. Such attacks are also referred to as product push attacks.

• The competitor of an item manufacturer might submit malicious reviews about the item. Such attacks are also referred to as nuke attacks.

By creating a concerted set of fake feedbacks from many different users, it is possible to change the predictions of the recommender system. Such users become shills in the attack process. Therefore, such attacks are also referred to as shilling attacks.

if you post a positive review online of your best friend's terrible taxi service

Table 12.1: A naive attack: injecting fake user profiles with a single pushed item

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	?	?	7	?	?
Fake-2	?	?	6	?	?
Fake-3	?	?	7	?	?
Fake-4	?	?	6	?	?
Fake-5	?	?	7	?	?

Table 12.2: Slightly better than naive attack: injecting fake user profiles with a single pushed item and random ratings on other items

Item ⇒ / **User ⇓**	1	2	3	4	5
John	1	2	1	6	7
Sayani	2	1	2	7	6
Mary	1	1	?	7	7
Alice	7	6	5	1	2
Bob	?	7	6	2	1
Carol	7	7	6	?	3
Fake-1	2	4	7	6	1
Fake-2	7	2	6	1	5
Fake-3	2	1	7	6	7
Fake-4	1	7	6	2	4
Fake-5	3	5	7	7	4

Let's consider a simple example to illustrate user-based collaborative filtering:

1. **User Ratings:**
   - User A has rated movies X, Y, and Z.
   - User B has rated movies X, Y, and W.
   - User C has rated movies Y, Z, and W.

2. **Similarity Calculation:**
   - Compute the similarity between User A and Users B and C. Let's say the similarity scores are:
     - Similarity(A, B) = 0.8
     - Similarity(A, C) = 0.6

3. **Peer Group Selection:**
   - Choose Users B and C as the peer group for User A.

4. **Weighted Average Rating Calculation:**
   - Predict the rating for movie W for User A using the weighted average:
     $$\text{Predicted Rating}_A(W) = \frac{(0.8 \times \text{Rating}_B(W)) + (0.6 \times \text{Rating}_C(W))}{0.8 + 0.6}$$

5. **Top-N Recommendations:**
   - Recommend the movies with the highest predicted ratings to User A.

In this example, User A's predicted rating for movie W is influenced by the ratings of Users B and C, with weights determined by their similarity to User A. This process helps recommend items to User A based on the preferences of similar users.