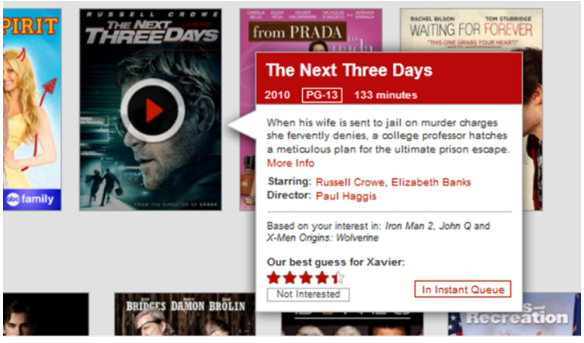
Chapter 6

Recommender Systems

What is a recommender system?
(And why do we need it?)



Source: http://techblog.netflix.com, 2012

Recommender System

A **recommender system** aims at **predicting** the **rating** or **preference** that a **user** would give to an **item** that she has not yet considered.

[1] Image source: Yasir72.multan [CC BY-SA 3.0], via Wikimedia Commons

Items and Ratings

Examples for the **items**:

- Songs, movies, books, news, blogs, products, people, venues,...

A **rating** can be expressed in various ways:

- **Interval-based ratings**: for example, by using a five-point (or five-star) scale. An important assumption is that the values explicitly define the distance between the ratings.
- **Ordinal ratings**: for example *disagree*, *neutral*, *strongly agree* etc. It is not assumed that the distance between any pair of adjacent rating values is the same.
- **Binary ratings**: only two options are present.
- **Unary ratings**: the user is allowed to specify a positive preference for an item, but there is no mechanism to specify a negative preference.
- **Continuous ratings**: the user can specify a rating on a continuous scale (for example, any real number between -10 and 10). This approach is used relatively rare.

Goals of recommender systems

General goals of recommender systems:

- Provide recommendations that are **relevant** for the user.
- Recommend items that a **novel** for the user, i.e., the user has not seen them in the past.
- In some cases, **serendipity** is also desirable. Such recommendations are truly surprising to the user and not just something she did not knew before.
- Provide recommendations that are **diverse**.

In general, the total number of items is large, but the number of items that have already been rated explicitly (or implicitly) by a user is small.
The central question is, how can we assess the utility of unrated items based on existing ratings of other items? As soon as the utility for all items has been computed, we can recommend the user the items with the "best" predicted rating.

Types of Recommender Systems

There are several **types of recommender systems**:

1. **Collaborative Filtering** Recommender System: recommend items to the user which have been preferred by other users with similar preferences in the past.
   - **Memory-based methods** (also called Neighborhood-based Collaborative Filtering)
     - **User-based Collaborative Filtering**
     - **Item-based Collaborative Filtering**
   - **Model-based methods**
2. **Content-based** Recommender Systems: recommend items to the user which are similar to the ones the user preferred in the past.
3. **Knowledge-based** Recommender Systems
4. **Hybrid** Recommender Systems

Collaborative Filtering

Collaborative filtering recommender systems can be subdivided into **memory-based** and **model-based** approaches.

- Memory-based approaches are also know as **neighborhood-based collaborative filtering**.

There are two primary types of neighborhood-based approaches:

1. **User-based collaborative filtering**: ratings provided by users that are similar to a target user u are used to make recommendations for u.
2. **Item-based collaborative filtering**: in order to predict the rating a user u would give to a particular item i, we first determine a set of other items S, which are most similar to i. Then a weighted average of the ratings in S is used to predict the rating that u will likely give to i.

The question is: how to determine the similarity between users and between items?

Neighborhood-based Collaborative Filtering

- The user-item ratings are usually expressed in an $m \times n$ **ratings matrix** $\mathbf{R}$, where m is the number of users in the system and n is the number of items.
- The ij-th element of $\mathbf{R}$ indicates the rating of user i for item j.
- Typically the ratings matrix is incomplete because not all users have rated all items.
- The goal of the recommender system is to predict the values for the empty cells.

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	7	6	7	4	5	4
User 2	6	7	?	4	3	4
User 3	?	3	3	1	1	?
User 4	1	2	2	3	3	4
User 5	1	?	1	2	3	3

User-based Collaborative Filtering

The idea behind user-based collaborative filtering is to identify other users that are **similar** to our **target user** (the user we want to predict ratings for). To determine the **neighborhood** of the target user, we need to compute her similarity to all other users.

- For the $m \times n$ ratings matrix $\mathbf{R}$ with m users and n items, let I_u denote the set of item indices for which ratings have been specified by user (row) u.
- For example[1], if the ratings of the first, third, and fifth items (columns) of user (row) u are specified (i.e. the user has rated the items before) and the remaining are missing, then we have $I_u = \{1, 3, 5\}$. Therefore, the set of items rated by both users u and v is given by $I_u \cap I_v$. For example, if user v has rated the first four items, then $I_v = \{1, 2, 3, 4\}$, and $I_u \cap I_v = \{1, 3, 5\} \cap \{1, 2, 3, 4\} = \{1, 3\}$. It is possible (and quite common) for $I_u \cap I_v$ to be an empty set because ratings matrices are generally sparse.
- The set $I_u \cap I_v$ defines the **mutually observed ratings**, which are used to compute the similarity between the uth and vth users for neighborhood computation.

[1] Example adopted from: Recommender Systems The Textbook, C. C. Aggarwal, Springer, 2016

User-based Collaborative Filtering

Based on the set $I_u \cap I_v$ we can now calculate the **similarity** between users u and v.

- First, we compute the **mean rating** μ_u for every user u:

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

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

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

Repetition: Pearson's Correlation

Pearson's correlation coefficient ρ_{xy} is a measure of the linear dependence between two variables x and y, giving a value between -1 and $+1$ inclusive. It is defined as the *covariance* of the two variables divided by the product of their *standard deviations*. Writing the covariance and standard deviation in full gives

$$\rho_{xy} = \frac{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

where $\bar{x}$ and $\bar{y}$ denote the mean of x and y respectively. $1/n$ can be dropped:

$$\rho_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

User-based Collaborative Filtering

Let $P_u(j)$ be the set of users who are similar to the target user u (i.e. the users who are in the **neighborhood** of u) and who have rated item j before.

- The **predicted rating** $\hat{r}_{ui}$ of user u for item j is then given by

$$\hat{r}_{ui} = \mu_u + \frac{\sum_{v \in P_u(j)} sim(u, v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |sim(u, v)|}$$

User-based Collaborative Filtering: Example

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6	Mean rating	Pearson(i,3)
User 1	7	6	7	4	5	4	5.5	0.894
User 2	6	7	?	4	3	4	4.8	0.939
User 3	?	3	3	1	1	?	2	1.0
User 4	1	2	2	3	3	4	2.5	-1.0
User 5	1	?	1	2	3	3	2	-0.817

$$sim(1,3) = \frac{(6-5.5)(3-2)+(7-5.5)(3-2)+(4-5.5)(1-2)+(5-5.5)(1-2)}{\sqrt{0.5^2+1.5^2+(-1.5)^2+(-0.5)^2} * \sqrt{1^2+1^2+(-1)^2+(-1)^2}} \approx 0.894$$

- For user 3, the users 1 and 2 are most similar.

User-based Collaborative Filtering

In case we applied just *raw* ratings in the prediction function:

$$\hat{r}_{31} = \frac{0.894 * 7 + 0.939 * 6}{0.894 + 0.939} \approx 6.49$$

$$\hat{r}_{36} = \frac{0.894 * 4 + 0.939 * 4}{0.894 + 0.939} = 4$$

Taking into consideration the mean-centered ratings we obtain

$$\hat{r}_{31} = 2 + \frac{0.894 * 1.5 + 0.939 * 1.2}{0.894 + 0.939} \approx 3.35$$

$$\hat{r}_{36} = 2 + \frac{0.894 * (-1.5) + 0.939 * (-0.8)}{0.894 + 0.939} \approx 0.86$$

User-based Collaborative Filtering: Discussion

Improving the metrics / prediction function:

- Not all neighbor ratings might be equally "valuable"
  - Agreement on commonly liked items is not so informative as agreement on controversial items
  - Possible solution: give more weight to items that have a higher variance
- Value of number of co-rated items
  - Use "significance weighting", by e.g., linearly reducing the weight when the number of co-rated items is low
- Case amplification
  - Intuition: Give more weight to "very similar" neighbors, i.e., where the similarity value is close to 1.
- Neighborhood selection
  - Use similarity threshold or fixed number of neighbors

User-based Collaborative Filtering

- Scalability issues can arise with user-based collaborative filtering if there are many more users than items ($m >> n, m = |users|, n = |items|$):
  - Space complexity $O(m^2)$ when pre-computed
  - Time complexity for computing Pearson $O(m^2 n)$
- High sparsity leads to few common ratings between two users.
- In this case we can apply item-based collaborative filtering, which exploits relationships between items first, instead of relationships between users.

Item-based Collaborative Filtering

Another type of memory-based approach is called **item-based collaborative filtering**, which utilizes the similarity between *items* for calculating a rating prediction.

- The rating prediction for a target item of a target user is based on a set of other *items* that are similar to the target item.
- Item-to-item similarity is commonly based on measuring the **cosine similarity** between the rating vectors. The (raw) cosine similarity measures the similarity between two non-zero vectors in terms of the cosine of the angle between them.

$$cos(\vec{a}, \vec{b}) = \frac{\vec{a} \cdot \vec{b}}{||\vec{a}|| \cdot ||\vec{b}||}$$

or, using a different notation:

$$cos(\vec{a}, \vec{b}) = \frac{\sum_{i=1}^{n} a_i * b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \sqrt{\sum_{i=1}^{n} b_i^2}}$$

where a_i denotes the i-th element of vector $\vec{a}$.

Item-based Collaborative Filtering

- Considering only the raw cosine usually leads to inaccurate predictions because of the rating bias.

Are more common approach is therefore to apply an **adjusted cosine similarity** measure in order to calculate the similarity between two items i and j.
Let U_i be the set of users who have rated item i before, and let μ_u be the mean rating of a user u.

$$AdjustedCosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} (r_{ui} - \mu_u)(r_{uj} - \mu_u)}{\sqrt{\sum_{u \in U_i \cap U_j} (r_{ui} - \mu_u)^2} * \sqrt{\sum_{u \in U_i \cap U_j} (r_{uj} - \mu_u)^2}}$$

The cosine similarity is bounded between $[-1, 1]$. If both vectors are oriented in positive space then the outcome is bounded between $[0, 1]$.

Item-based collaborative filtering

Let $Q_t(u)$ be the set of top-k items most similar to item t for which user u has specified ratings.

- Then we can apply, for example, the following prediction function:

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} AdjustedCosine(j,t) * r_{uj}}{\sum_{j \in Q_t(u)} |AdjustedCosine(j,t)|}$$

Item-based Collaborative Filtering: Example

Consider the ratings as before for the user-based approach, however, this time with **mean-centered** ratings:

	Item 1	Item 2	Item 3	Item 4	Item 5	Item 6
User 1	1.5	0.5	1.5	-1.5	-0.5	-1.5
User 2	1.2	2.2	?	-0.8	-1.8	-0.8
User 3	?	1	1	-1	-1	?
User 4	-1.5	-0.5	-0.5	0.5	0.5	1.5
User 5	-1	?	-1	0	1	1
Cosine(1,j)	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

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

Items 2 and 3 are most similar to item 1. Items 4 and 5 are most similar to item 6.

Item-based Collaborative Filtering: Example

Using the AdjustedCosine similarity measure, we can now calculate the rating predictions, for example, for user 3:

- For predicting the rating of user 3 for item 1, we consider the two items 2 and 3 (because of high AdjustedCosine similarity with the target item 1):

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

- For predicting the rating of user 3 for item 6, we consider items 4 and 5 (because they have high similarity with target item 6):

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

Note that the predicted values are within the range of the original rating scale (1 to 7).

Item-based Collaborative Filtering

- Item-based collaborative filtering does not solve the scalability problem itself.
- However, we can calculate all pair-wise item similarities in advance, because item similarities are supposed to be more stable than user similarities.
- The neighborhood to be used at run-time is typically rather small, because only items are taken into account which the user has rated.

Memory requirements:

- In theory, up to n^2 pair-wise similarities need to be memorized (n = number of items).
- In practice, this is significantly lower (items with no co-ratings).
- Further reductions possible, for example:
  - Minimum threshold for co-ratings (items, which are rated at least by t users).
  - Limit the size of the neighborhood (might affect recommendation accuracy).

Discussion

General issues to consider:

- **New user problem**: the system first needs to learn the preferences of the user form his ratings.
- **New item problem**: items can only be recommended when a substantial number of users have rated it before.
- **Sparsity**: many items that have been rated by only few users each. Even if they got a high ranking, they would be recommended rarely. Also, a user whose preferences are unusual compared to all other users will get poor recommendations.

A solution to the problem of rating sparsity is to segment users based on their profile, for example age, gender, country, education (a.k.a. demographic filtering).

Model-based Collaborative Filtering

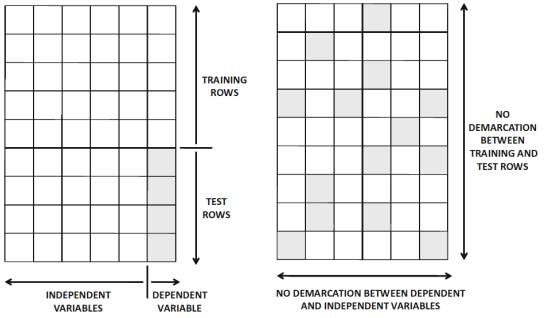
Neighborhood-based approaches for collaborative filtering are specific to the user and item that is being predicted. In contrast, **model-based collaborative filtering** relies on a summarized model of the data that is created beforehand during a training phase. This model is then used in the later prediction phase to efficiently calculate the predictions.

Advantages of model-based approaches[1]:

- Space-efficiency: Typically, the size of the learned model is much smaller than the original ratings matrix.
- Training speed and prediction speed: One problem with neighborhood-based methods is that the pre-processing stage is quadratic in either the number of users or the number of items. Model-based systems are usually much faster in the preprocessing phase of constructing the trained model.
- Avoiding overfitting: Overfitting is a serious problem in many machine learning algorithms, in which the prediction is overly influenced by random artifacts in the data. The summarization approach of model-based methods can often help in avoiding overfitting.

[1] Adopted from: Recommender Systems The Textbook, C. C. Aggarwal, Springer, 2016

Model-based Collaborative Filtering



(a) Classification (b) Collaborative filtering

Comparing the traditional classification problem with collaborative filtering. Shaded entries are missing and need to be predicted[1]

Model-based collaborative filtering methods include, for example, rule-based methods, decision trees, regression models, Bayes classifiers, support vector machines, or neural networks.

[1]Source: Recommender Systems The Textbook, C. C. Aggarwal, Springer, 2016

Association Rule Mining

Association Rule Mining is a technique to identify rulelike relationship patterns in large-scale transactions.

- An example rule could be: "If a customer purchases baby food then he or she also buys diapers in 70 percent of the cases".

Definition by Sarwar et al.:

- A (sales) transaction T is a subset of the set of available products $P = \{p_1, \ldots, p_m\}$ and describes a set of products that have been purchased together.
- Association rules are often written in the form $X \rightarrow Y$, with X and Y being both subsets of P and $X \cap Y = \emptyset$
- An association rule $X \rightarrow Y$ (e.g. baby food $\rightarrow$ diapers) expresses that whenever the elements of X (the rule body) are contained in transaction T, it is very likely that the elements in Y (the rule head) are elements of the same transaction.

In collaborative recommender systems, the goal of association rule mining is to detect rules such as *"If user Alice liked both Item* 5 *and Item* 7*, then Alice will most probably also like Item* 23*"*.

Association Rule Mining

The goal of rule mining algorithms such as *Apriori* (by Agrawal and Srikant, 1994) is to automatically detect such rules and *calculate a measure of quality for those rules*[1].

- The **support** of a rule $X \rightarrow Y$ is calculated as the percentage of transactions that contain all items of $X \cup Y$ with respect to the number of overall transactions (i.e., the probability of co-occurrence of X and Y in a transaction):

$$\text{Support}(X \rightarrow Y) = \frac{\text{Number of transactions containing } X \cup Y}{\text{Number of transactions}}$$

- The **confidence** is defined as the ratio of transactions that contain all items of $X \cup Y$ to the number of transactions that contain only X:

$$\text{Confidence}(X \rightarrow Y) = \frac{\text{Number of transactions containing } X \cup Y}{\text{Number of transactions containing } X}$$

[1] Adapted from: D. Jannach et al.: Recommender Systems An Introduction, Cambridge University Press, 2011

Association Rule Mining: Example

Assume a ratings matrix with a binary scale ("like" / "dislike"):

	Bread	Butter	Milk	Fish	Beef	Ham
User 1	1	1	1	0	0	0
User 2	0	1	1	0	1	0
User 3	1	1	0	0	0	0
User 4	1	1	1	1	1	1
User 5	0	0	0	1	0	1
User 6	0	0	0	1	1	1
User 7	0	1	0	1	1	0

It is evident that the columns of the table can be partitioned into two sets of closely related items. One of these sets is $\{Bread, Butter, Milk\}$, and the other set is $\{Fish, Beef, Ham\}$. These are the only itemsets with at least 3 items, which also have a support of at least 0.2.

Association Rule Mining

Finding association rules involves two steps:

1. Find all itemsets that satisfy a minimum support threshold s.
2. From each of these itemsets Z, all possible 2-way partitions $(X, Z - X)$ are used to create a potential rule $X \rightarrow Z - X$. Those rules satisfying the minimum confidence are retained.

The calculation of interesting association rules can be performed offline.

- At runtime, the following scheme can be used to compute recommendations for a user Alice:
  - Determine the set of $X \rightarrow Y$ association rules that are relevant for Alice that is, where Alice has bought (or liked) all elements from X.
  - Compute the union of items appearing in the consequent Y of those association rules that have not been purchased by Alice.
  - Sort the products according to the confidence of the rule that predicted them.
  - Return the first N elements of this ordered list as a recommendation.

Matrix factorization / Singular Value Decomposition

- Simply put, matrix factorization methods can be used in recommender systems to derive a set of **latent** (hidden) factors from the rating patterns and characterize both users and items by such vectors of factors.
- **Singular Value Decomposition (SVD)** was proposed by Deerwester et al. in 1990 as a method to discover latent factors in documents.
- The SVD theorem by Golub and Kahan (1965) states that a given matrix $\mathbf{M}$ can be decomposed into a product of three matrices as follows:

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$$

where $\mathbf{U}$ and $\mathbf{V}$ are called *left* and *right singular vectors* and the values of the diagonal of Σ are called the *singular values*.

Singular Value Decomposition: Example

Consider the following rating matrix:

	Item 1	Item 2	Item 3	Item 4
User 1	3	1	2	3
User 2	4	3	4	3
User 3	3	2	1	5
User 4	1	6	5	2

Because in this example the $4x4$ matrix $\mathbf{M}$ is quadratic, $\mathbf{U}, \Sigma$ and $\mathbf{V}$ are also quadratic.

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^{\mathbf{T}}$$

$$\mathbf{U} = \begin{pmatrix} -0.35 & -0.36 & 0.29 & -0.80 \\ -0.56 & -0.08 & 0.62 & 0.52 \\ -0.44 & -0.56 & -0.65 & 0.21 \\ -0.59 & -0.73 & -0.28 & -0.17 \end{pmatrix}$$

$$\Sigma = \begin{pmatrix} 12.22 & 0 & 0 & 0 \\ 0 & 4.92 & 0 & 0 \\ 0 & 0 & 2.06 & 0 \\ 0 & 0 & 0 & 0.29 \end{pmatrix}$$

$$\mathbf{V}^T = \begin{pmatrix} -0.43 & -0.53 & -0.52 & -0.50 \\ -0.49 & 0.53 & 0.40 & -0.55 \\ 0.55 & -0.41 & 0.48 & -0.53 \\ 0.51 & 0.50 & -0.56 & -0.38 \end{pmatrix}$$

Singular Value Decomposition: Example

The main point of the SVD is that we can approximate the full matrix by observing only the most important features, i.e., those with the largest singular values.

- For example, consider the projection of $\mathbf{U}$, $\mathbf{V^T}$ and Σ in the two-dimensional space:

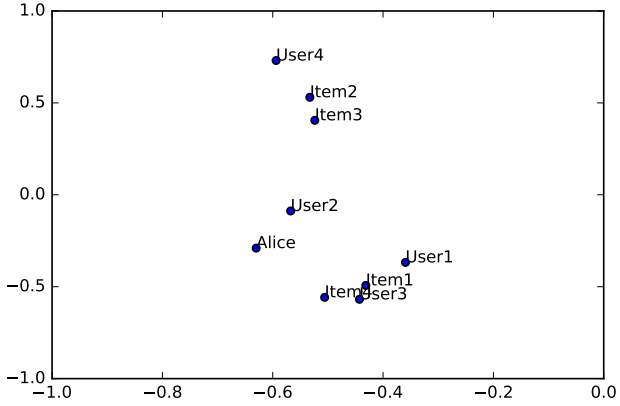
$$\mathbf{U}_2 = \begin{pmatrix} -0.35 & -0.36 \\ -0.56 & -0.08 \\ -0.44 & -0.56 \\ -0.59 & -0.73 \end{pmatrix} \quad \Sigma_2 = \begin{pmatrix} 12.22 & 0 \\ 0 & 4.92 \end{pmatrix} \quad \mathbf{V}_2^T = \begin{pmatrix} -0.43 & -0.53 & -0.52 & -0.50 \\ -0.49 & 0.53 & 0.40 & -0.55 \end{pmatrix}$$

- Matrix $\mathbf{U}$ and $\mathbf{V}$ correspond to the **latent user and item factors**.
- Although in this example we cannot observe any clusters of users, we can see that the items from $\mathbf{V}$ build two groups.

Singular Value Decomposition: Example

Consider a target user *Alice* with a rating vector $r_{Alice} = [5, 3, 4, 4]$ for the four items. We can find out where Alice would be positioned in this two-dimensional space by calculating

$$r_{Alice2D} = r_{Alice} \times \mathbf{V}_2 \times \Sigma_2^{-1} \approx [-0.63, 0.29]$$



Naive Bayes Collaborative Filtering

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

- We treat items as features and users as instances.
- The main challenge is that any feature (item) can be the target class, and we have to deal with incomplete feature variables.

Assume a small number of l distinct ratings $v_1, \ldots, v_l$ such as *like, neutral, dislike*, and an $m \times n$ rating matrix where the (u, j)th element is denoted by r_{uj}. Furthermore, let I_u be the set of items that have been rated by user u.
The goal of the Bayes classifier is to predict the unobserved rating r_{uj}.

- r_{uj} can take any one of the discrete possibilities in $\{v_1, \ldots, v_n\}$.
- We would like to determine the **probability** that r_{uj} takes on any of these values **conditional on the observed ratings in I_u**.

In other words, we want to determine the **probability** $P(r_{uj} = v_s | \text{Observed ratings in } I_u)$ for each value of s in $\{1, \ldots, l\}$.

Naive Bayes Collaborative Filtering

We can simplify this expression by using the well-known **Bayes theorem**:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

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

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

- We would like to determine the value of s for which the value of $P(r_{uj} = v_s | \text{Observed ratings in } I_u)$ is as large as possible.
- Because the denominator of the right hand side is independent of s, we can express the equation in terms of a constant of proportionality:

$$P(r_{uj} = v_s | \text{Observed ratings in } I_u) \propto P(r_{uj} = v_s) * P(\text{Observed ratings in } I_u | r_{uj} = v_s)$$

Naive Bayes Collaborative Filtering

- The value of $P(r_{uj} = v_s)$, also called the **prior probability**, is estimated to the fraction of users that have specified the rating v_s for item j (users who haven't rated j are ignored here).
- $P(\text{Observed ratings in } I_u | r_{uj} = v_s)$ is estimated with the use of the **naive assumption** (i.e., conditional independence between ratings). Conditional independence says that the ratings of a user for various items I_u are independent from each other, *conditional* of the fact that the value of r_{uj} was observed to be v_s. We can express this mathematically as follows:

$$P(\text{Observed ratings in } I_u | r_{uj} = v_s) = \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)$$

- $P(r_{uk} | r_{uj} = v_s)$ is estimated as the fraction of users that have specified the rating of r_{uk} for the kth item, *given that they have specified the rating of their jth item to v_s*.

Naive Bayes Collaborative Filtering

By plugging in the estimation of the prior probability $P(r_{uj} = v_s)$ and the previous equation, it is possible to obtain an estimate of the **posterior probability** of the rating of item j for user u as follows:

$$P(r_{uj} = v_s | \text{Observed ratings in } I_u) \propto P(r_{uj} = v_s) * \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)$$

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

$$\hat{r}_{uj} = \text{argmax}_{v_s} P(r_{uj} = v_s) * \prod_{k \in I_u} P(r_{uk} | r_{uj} = v_s)$$

Content-based Recommendations

Content-based recommender systems rely on a **user's own ratings** and try to identify new items that are **content-wise similar** to the items the user has liked before. Here, the content-wise similarity is defined on the attributes of items and *not* based on rating preferences of other users. Examples:

- Books from similar authors and from similar subject matters of books you have bought before.
- Movie recommendations based on the genre and actors in movies you have watched in the past.
- People recommendations based on common interests with people you already are already connected to.

In general, each content-based recommender system relies on two concepts: user profile and item profile.

- The **user profile** comprises of previous explicit or implicit feedback the user has generated about various items.
- The **item profile** is a set of content-centric attributes for a particular item, for example, a genre, a textual description, or a list of keywords.

Content-based Recommender Systems

Limitations of content-based recommendations:

- **Limitations in item analysis**: Given that the set of items is usually large, all items must be available in a data format that can be automatically parsed in order to extract the features. This is often easy for text documents, but harder for audio or video streams etc.
- **Equality of features**: If two items are represented by the same set of features, it is not possible anymore to distinguish between the two.
  - Example: a gossip news article and a scientific paper that contain a similar set of keywords.

Content-based Recommender Systems

Limitations of content-based recommendations:

- **Overspecialization**: can occur when only items with a very high degree of utility are recommended. Then, if a user has never rated items from a certain group of items, he will not get any recommendations for them.
  - Example: a user who has never rated a French restaurant won't get any recommendation for even the greatest cuisine around the corner.

  A solution to the overspecialization problem is to introduce some randomness in order to increase the diversity of recommendations and to present a range of options to the user.
- **New user problem**: a new user with no or very few existing ratings will not receive accurate recommendations.

Content-based Recommender Systems

- Let N be the number of all documents (items) D that can be recommended.
- Let $f_{t,d}$ be the number of times that **term** t occurs in **document** d.

The **term frequency** is defined as:

$$\mathrm{TF}(t,d) = \frac{f_{t,d}}{max_x\{f_{x,d}\}}$$

(We divide the raw frequency by the maximum raw frequency of any term in the document)
The **inverse document frequency** measures how common or rare a term is across all documents:

$$\mathrm{IDF}(t) = log\frac{N}{|\{d \in D : t \in d\}|}$$

The **term frequency-inverse document frequency (TF-IDF)** is defined as:

$$\text{TF-IDF}(t,d) = \mathrm{TF}(t,d) * \mathrm{IDF}(t)$$

Content-based Recommender Systems

TF-IDF is one possible measure to create individual **item profiles**:

$$\text{Profile}(d) = \Big(\text{TF-IDF}(t_1, d), \text{TF-IDF}(t_2, d), \ldots, \text{TF-IDF}(t_j, d)\Big)$$

where j is the number of terms in d.

- Given such vectors of TF-IDF values, we can then determine the similarity between two items, for example, by calculating the cosine between the vectors.
- This requires, of course, that there there is a set of common terms that appears in both documents.
- In practice, it is very common (and advisable) to preprocess documents before calculating TF-IDFs. Common steps are, for example:
  - Removing **stop words**, i.e., very common terms such as "a", "the", "I", "it", etc.
  - **Stemming**, i.e., reducing all words to their stems.

Source: Recommender Systems - The Textbook, C. C. Aggarwal, Springer, 2016

Content-based recommendations

Suppose we would like to perform an automated **feature selection** among all possible terms, based on how discriminative a term is *for a user*. One possible approach is to study the **Gini index** for each term.

- In the following, we treat all ratings of a user as categorial values.
- Let v be the number of possible ratings. For example, $v = 5$ in a 5-star rating scheme.
- Let t be a particular term and let $p_1(t), \ldots, p_v(t)$ be the fraction of the items rated at each of these v possible values.

Then, the **Gini index** of the term t is defined as

$$Gini(t) = 1 - \sum_{i=1}^{v} p_i(t)^2$$

- The value of $Gini(t)$ lies in the range $(0, 1 - \frac{1}{t})$.
- Smaller values indicate **greater discriminative power**.

Knowledge-based Recommender Systems

Knowledge-based recommender systems rely on explicitly soliciting user requirements for items.

- Knowledge-based recommender systems are well suited to the recommendation of items that are not bought on a regular basis (for example, house, car, etc.).

There are two types of knowledge-based recommender systems:

1. **Constraint-based recommender systems**: users specify requirements or constraints (for example, lower or upper limits) on the item.
2. **Case-based recommender systems**: specific cases are specified by the user as targets or anchor points.

Source: Recommender Systems - The Textbook, C. C. Aggarwal, Springer, 2016

Constraint-based Recommender Systems

Three types of input for a constrained-based recommender system:

1. Inherent **properties of the user** (e.g, demographics, risk profiles) and specific **requirements in the product**.
2. **Knowledge bases**, which map user attributes to products attributes. For example, a user who is searching for a house and who has a family size of 5 will likely require ≥ 3 bedrooms.
3. The **product catalog**, which contains all the item attributes.

Question: how to handle empty result sets?

Case-based Recommender Systems

In **case-based recommender systems** a similarity function is used to retrieve the examples that are *most similar* to the **user-specified target**.

- After the results have been presented to the users, feedback is typically enabled through the use of **critiques**.
- The user may specify a directional critique (e.g., "cheaper") or a replacement critique (e.g., "different color").
- **Compound critiques** allow the user to specify multiple modifications, which are hidden behind informal descriptions that the user can understand (e.g., "classier", "roomier", "cheaper", "sportier").

The point in conversational critiquing is that when a user might wish to have a "classier" car, but they might not be easily able to concretely express it in terms of the product features such as the interior structure of the car. On the other hand, a qualification such as classier is more intuitive, and it can be encoded in terms of the product features by a domain expert. This interactive process is designed to help them learn the complex product space in an intuitive way.

Summary of the types of recommender systems

1. **Collaborative Filtering** Recommender Systems
   - **Memory-based methods**
     - **User-based Collaborative Filtering**
     - **Item-based Collaborative Filtering**
   - **Model-based methods**
2. **Content-based** Recommender Systems
   - Feature extraction techniques
   - Content-based learning of user-profiles
   - Filtering and recommendation
3. **Knowledge-based** Recommender Systems
   - Constraint-based Recommender Systems
   - Case-based Recommender Systems
4. **Hybrid** Recommender Systems

Evaluating Recommender Systems

In order to assess the quality of a recommender system (which might internally combine multiple types of recommender techniques), we need to set some **evaluation goals** first.

- For example, **accuracy** is one important measure, however, it does not capture other important characteristics such as serendipity, novelty of items, or coverage.

The evaluation can be performed either by means of **online methods**, such as A/B testing, or via **offline methods**, which rely on historical data sets.

- Online methods allow to measure the true effectiveness of a recommender system, for example by quantifying the achieved conversion rate.
- However, compared to offline methods, online methods are typically more difficult to apply, as they require direct user participation.

Evaluating Recommender Systems

- **Mean Absolute Error:**

$$\text{MAE} = \frac{1}{|U|} \sum_{u \in U} \frac{1}{|I_u|} \sum_{i \in I_u} |\hat{r}_{ui} - r_{ui}|$$

where U is the set of users who have rated at least one item, I_u is the set of items rated by user u, r_{ui} is the rating of user u for item i, and $\hat{r}_{ui}$ is the predicted rating of user u for item i.

- **Root Mean Squared Error:**

$$\text{RMSE} = \sqrt{\frac{1}{|U|} \sum_{u \in U} \frac{1}{|I_u|} \sum_{i \in I_u} (\hat{r}_{ui} - r_{ui})^2}$$

The RMSE disproportionately penalizes large errors in the predictions, while the MAE does not.

Evaluating Recommender Systems

		Reality	
		Actually Good	Actually Bad
Prediction	Rated good	true positive (tp)	false positive (fp)
	Rated bad	false negative (fn)	true negative (tn)

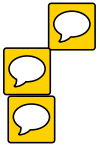
- **Precision**: a measure of exactness, determines the fraction of relevant items retrieved out of all items retrieved

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} = \frac{|\text{good movies recommended}|}{|\text{all recommendations}|}$$

For example: the proportion of recommended movies that are actually good

- **Recall**: a measure of completeness, determines the fraction of relevant items retrieved out of all relevant items

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}} = \frac{|\text{good movies recommended}|}{|\text{all good movies}|}$$



For example: the proportion of all good movies recommended

Attacks on Recommender Systems

(Monetary) value of being in recommendation lists

- Individuals may be interested to push some items by manipulating the recommender system.
- Individuals might be interested to decrease the rank of other items.
- Some simply might may want to sabotage the system.

Manipulation of the "Internet opinion"

- Malevolent users try to influence behavior of recommender systems.
- System should include a certain item very often/seldom in its recommendation list.

A simple strategy?

- (Automatically) create numerous fake accounts / profiles.
- Issue high or low ratings to the "target item".
  - Will not work for neighbor-based recommenders.
  - More elaborate attack models required.
  - Goal is to insert profiles that will appear in neighborhood of many.

Source: Recommender Systems - An Introduction, D. Jannach et al., Cambridge University Press, 2011

Attacks on Recommender Systems

Characterization of profile insertion attacks:

- **Push attack**: increase the prediction value of a target item.
- **Nuke attack**: decrease the prediction value of a target item.
- Make the recommender system **unusable** as a whole.

There is no technical difference between push and nuke attacks, nevertheless Push and Nuke attacks are not always equally effective.
Differentiation factors between attacks:

- Where is the focus of an attack? Only on particular users and items?
- Targeting a subset of items or users might be less suspicious.
- More focused attacks may be more effective (attack profile more precisely defined).

Attacks on Recommender Systems

Classification criteria for recommender system attacks include:

- **Cost**
  - How costly is it to make an attack?
  - How many profiles have to be inserted?
  - Is knowledge about the ratings matrix required? (Usually it is not public, but estimates can be made)
- **Algorithm dependability**
  - Is the attack designed for a particular recommendation algorithm?
- **Detectability**
  - How easy is it to detect the attack?

Attacks on Recommender Systems

General scheme of an attack profile:

Item 1	...	Item K	...	Item L	...	Item N	Target
r_1	...	r_k	...	r_l	...	r_n	?
selected items			filler items		unrated items		

Attack models mainly differ in the way the profile sections are filled.

Attacks on Recommender Systems

Random attack model

- Take random values for both the selected items and the filler items. Typical distribution of ratings is known, e.g., for the movie domain (average 3.6, standard deviation around 1.1).
- Idea: generate profiles with "typical" ratings so they are considered as neighbors to many other real profiles.
- High/low ratings for target items.
- Limited effect compared with more advanced models.

Item 1	...	Item K	...	Item L	...	Item N	Target
r_1	...	r_k	...	r_l	...	r_n	?
selected items			filler items		unrated items		

Attacks on Recommender Systems

Average attack model

- Use the individual item's rating average for both selected items and filler items.
- Intuitively, there should be more neighbors.
- Additional cost involved: find out the average rating of an item.
- More effective than Random Attack in user-based CF, but additional knowledge is required.
- It is quite easy to determine average rating values per item vValues explicitly provided when item is displayed).

Item 1	...	Item K	...	Item L	...	Item N	Target
r_1	...	r_k	...	r_l	...	r_n	?
selected items			filler items		unrated items		

Attacks on Recommender Systems

Bandwagon attack model

- Exploits additional information about the community ratings.
- Simple idea: add profiles that contain high ratings for "blockbusters" (in the selected items) and use random values for the filler items.
- Will intuitively lead to more neighbors because popular items will have many ratings and rating values are similar to many other user-profiles.
- Example: Injecting a profile with high rating values for the Harry Potter series.
- Low-cost attack, set of top-selling items/blockbusters can be easily determined.
- Does not require additional knowledge about mean item ratings.

Item 1	...	Item K	...	Item L	...	Item N	Target
r_1	...	r_k	...	r_l	...	r_n	?
selected items			filler items		unrated items		

Attacks on Recommender Systems

Segment attack model

- Designing an attack that aims to push item A.
- Find items that are similar to target item, these items probably liked by the same group of people.
- Identify subset of user community that is interested in items similar to A.
- Inject profiles that have high ratings for fantasy novels and random or low ratings for other genres. Thus, item will be pushed within the relevant community.
- For example: Push the new Harry Potter book
  - Attacker will inject profile with positive ratings for other popular fantasy books. Harry Potter book will be recommended to typical fantasy book reader.
- Additional knowledge (e.g. genre of a book) is required

Attacks on Recommender Systems

Special nuke attacks

- **Love/hate** attack
  - Target item is given the minimum value.
  - Filler items are given the highest possible rating value
  - Serious effect on systems recommendations when goal is to nuke an item. Other way around (push an item) it is not effective.
- **Reverse bandwagon**
  - Associate target item with other items that are disliked by many people.
  - Selected item set is filled with minimum ratings.

Item 1	...	Item K	...	Item L	...	Item N	Target
r_1	...	r_k	...	r_l	...	r_n	?
selected items			filler items		unrated items		

Attacks on Recommender Systems

Effectiveness of attacks:

- Effect depends mainly on the attack size (number of fake profiles inserted)
- User-based recommenders:
  - Bandwagon / Average Attack: bias shift of 1.5 points on a 5-point scale at 3% attack size.
  - Average Attack slightly better but requires more knowledge.
  - 1.5 points shift is significant; 3% attack size means inserting, e.g., 30,000 profiles into one-million rating database.
- Item-based recommenders:
  - Far more stable; only 0.15 points prediction shift achieved.
  - Exception: Segment attack successful (was designed for item-based method).
- Hybrid recommenders and other model-based algorithms cannot be easily biased (with the described/known attack models)

Attacks on Recommender Systems

Countermeasures (examples):

- Use model-based or hybrid algorithms.
  - More robust against profile injection attacks.
  - Accuracy comparable with accuracy of memory-based approaches.
  - Less vulnerable.
- Increase profile injection costs.
- Use captchas.
- Use low-cost manual insertion.
- Use statistical attack detection methods:
  - Detect groups of users who collaborate to push/nuke items.
  - Monitor development of ratings for an item, such as changes in average rating, or changes in rating entropy, or use time-dependent metrics (bulk ratings).
- Use machine-learning methods to discriminate real from fake profiles.