Recommender Systems & Collective Intelligence

COMP47580

Dr. Michael O'Mahony

michael.omahony@ucd.ie

Overview

- Previously:
  - Non-personalised recommendation (average ratings, ad-hoc ranking metrics, product association, content-based)
  - Personalised recommendation (content-based)
- This topic – collaborative filtering approach to personalised prediction and recommendation
- Evaluation methodology and metrics
- Advantages and limitations

Recommender System Approaches

- Personalised recommender systems have been implemented using a variety of approaches:
  - Content-based recommenders
  - Collaborative filtering (CF)
  - Hybrid approaches
  - Cross-domain recommenders
  - Many others…
- This lecture – focus on CF:
  - Memory-based vs. model-based approaches

Collaborative Filtering (CF)

- CF – automates the “word-of-mouth” process
- Assists users to make choices based on the opinions of other users
- The underlying heuristic: people who agreed or disagreed on items in the past are likely to agree or disagree on future items
- Predictions and recommendations are made for users by combining the preferences of similar users in the system:
  - No content descriptions are required – recommendations are based only on user preferences
  - Can be applied to many domains – art, books, jokes, movies, music, web pages, blogs, hotels, restaurants …
  - Captures what people actually thought about items – thus, quality is considered
  - Can provide novel/serendipitious recommendations

Content-based Recommendation

- Items are recommended which are are similar in content to previously selected items
- Recommendations are based on a description of the content of items as opposed to users' opinions about items
- Comparisons between items are calculated over the features associated with each item:
  - For example, in the movie domain, features include actors, genres, director, plot summary…
  - Based on the range of feature values associated with a user's past choices, further movies with similar feature values are recommended
  - A *more like this* approach to recommendation
  - Recommendation quality? Diversity/novelty/serendipity?

Collaborative Filtering (CF)

- Let’s look at a toy example…
- Assume we have the following preference data…

	The Quiet Man	Casino	Star Wars	Top Gun	Dallas: The Movie
Eamon	3		3	4	2
Sharon	4	1	4	2	4
John	5	2		2	5
Trisha	2	5	3		1
Mike	?	2	4	1	5

- Q – based on this preference data, would user *Mike* like or dislike *The Quiet Man*?

Collaborative Filtering (CF)

- Let’s look at a toy example…
- Assume we have the following preference data…

	The Quiet Man	Casino	Star Wars	Top Gun	Dallas: The Movie
Eamon	3		3	4	2
Sharon	4	1	4	2	4
John	5	2		2	5
Trisha	2	5	3		1
Mike	?	2	4	1	5

- Q – based on this preference data, would user *Mike* like or dislike *The Quiet Man*?

Collaborative Filtering (CF)

- Let’s look at a toy example…
- Assume we have the following preference data…

	The Quiet Man	Casino	Star Wars	Top Gun	Dallas: The Movie
Eamon	3		3	4	2
Sharon	4	1	4	2	4
John	5	2		2	5
Trisha	2	5	3		1
Mike	?	2	4	1	5

- Q – based on this preference data, would user *Mike* like or dislike *The Quiet Man*?

Collaborative Filtering (CF)

- Typically supports users in two ways:
  - Make predictions for specific item(s) (prediction problem)
  - Recommend a ranked top-N list of items (top-N problem)
- Many CF algorithms proposed (user-based, item-based, matrix factorisation, other model-based approaches)…
- To begin, focus on user-based CF – similar to the toy example we looked at earlier…
- Remember – CF approaches operate over a set of user preferences; no content descriptions are required

User-based CF Algorithm

Algorithm:

```
1. Data Representation:
       Obtain preference data from users
       Construct (conceptually) a user-item matrix

2. Similarity Computation:
       Compute the similarity between the active user and all
       other users in the system

3. Neighbourhood Formation:
       Pick a subset of the system users on which predictions or
       recommendations are based

4. Make a prediction or generate a top-N list
```

Different approaches used in the above steps …

Data Representation

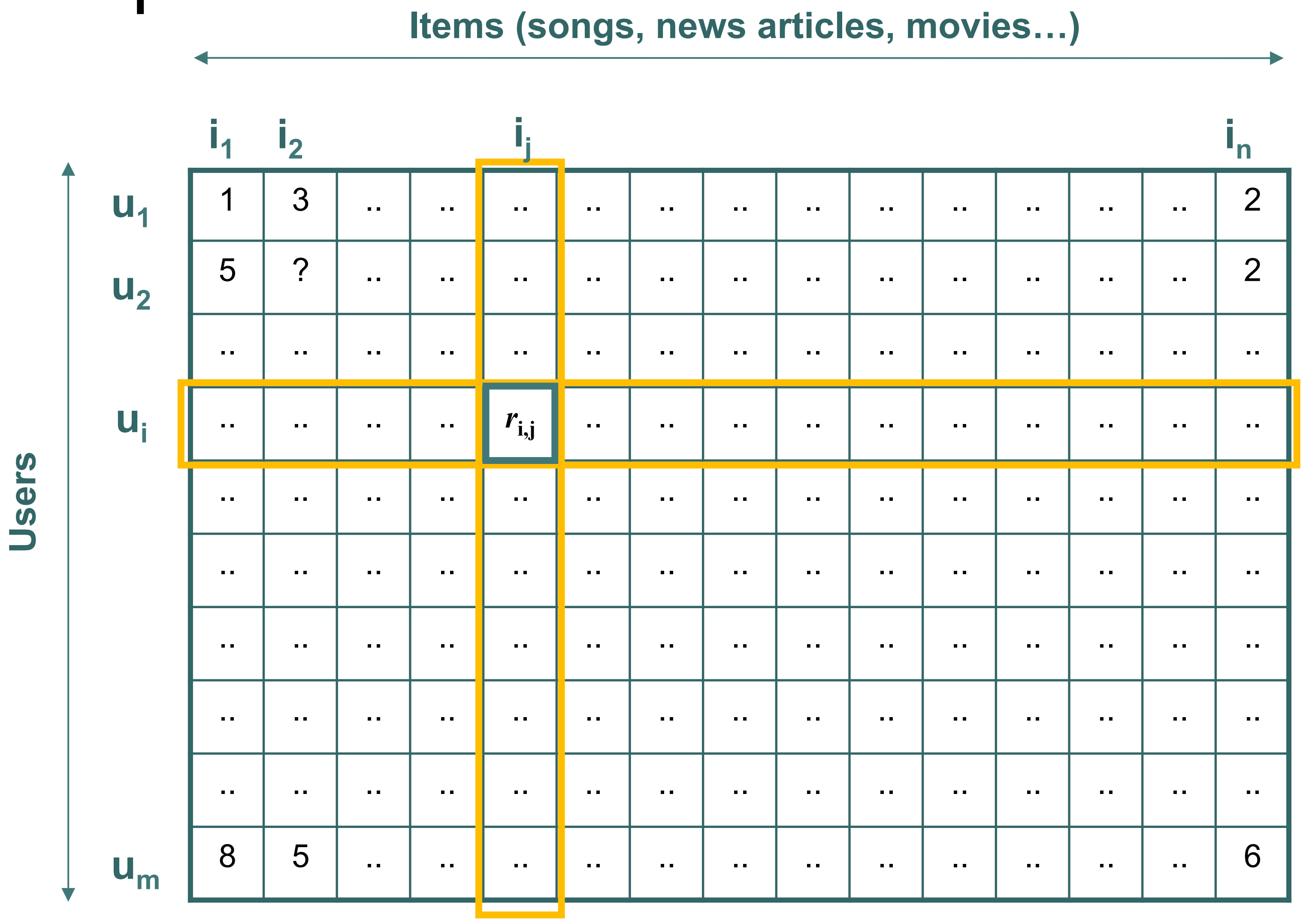
- Need to acquire (lots of) user preferences
- Types of ratings:
  - Scalar ratings: numerical ratings (e.g. 1 - 5 stars for movies) or ordinal ratings (e.g. strongly agree, agree, neutral, disagree, strongly disagree)
  - Binary ratings: e.g. votes up/down, like/dislike or agree/disagree
  - Unary ratings: e.g. likes, web pages visited, TV shows watched (note that not visiting a web page does not necessarily imply dislike)
- Ratings are gathered explicitly or implicitly

Data Representation

- Explicit Ratings
  - User supplies ratings – fairly precise method of gathering data; issues with reliability, changing preferences, malicious preferences…
  - Do not want to over-burden users – users may tire of providing ratings
  - Users need to be convinced there is a benefit in order to make the effort
  - CF algorithms need many ratings to function accurately

- Implicit Ratings
  - Not obtained directly from user; instead, ratings are “inferred” – e.g. a user’s browsing patterns (time spent on web page, a user’s bookmarks, links followed)
  - Removes “cost” of explicitly gathering ratings
  - Every user interaction can potentially contribute
  - Can be combined with explicit ratings
  - Implicitly gathered ratings – plentiful but noisy
- Ratings are “stored” in a user-item matrix

User-item Matrix



User Similarity

- How do we measure the similarity between users?
- A number of approaches have been considered…
- In most approaches, there needs to be an overlap between two users in order to compute their similarity:
  - If there is no overlap => no similarity computation is possible (in such cases, generally assume similarity is zero)
  - Small overlap => skewed results; an important limitation because in real world applications, the user-item matrix is sparse (many/most of the ratings are missing – e.g. consider Amazon…)
  - Basic approaches do not consider the importance of agreement on controversial items as opposed to agreement on universally-liked items
  - Large profiles (i.e. users who have rated many items) will have overlaps with many other profiles – problematic, can lead to such profiles being included in neighbourhoods "by default"
- There are various solutions to these problems…

Mean Squared Difference

- The Mean Squared Difference in the ratings between users *a* and *i* is computed as:

$$\mathrm{MSD}_{a,i} = \frac{\sum_{j \in I_a \cap I_i} (r_{a,j} - r_{i,j})^2}{|\{j : j \in I_a \cap I_i\}|}$$

where:

  - I_a is the set of items rated by user *a*
  - $r_{a,j}$ is the rating of user *a* for item *j*

- MSD computes the *difference* in ratings between two users; convert MSD into a *similarity* metric as follows:

$$w_{a,i} = 1 - \frac{\mathrm{MSD}_{a,i}}{(r_{max} - r_{min})^2}$$

where r_{min} and r_{max} are the minimum and maximum ratings, resp.

Mean Squared Difference

- Summations over co-rated items only − if there are no co-rated items, similarity is set to 0.
- Results in a value of [0,+1]
- Applicable for unary ratings?
- MSD assumes that users rate items according to similar distributions:
  - Problems occur when this assumption is violated
  - Critical vs. ‘generous’ raters? Critical raters rarely assign the maximum ratings to items…
  - So while differences in magnitudes between users’ ratings may exist, they may be highly correlated
  - Need a scale-invariant user similarity function to more accurately reflect rating patterns and to capture trends in users’ ratings

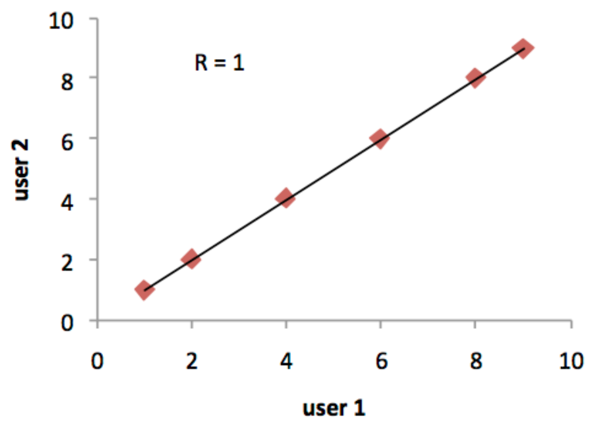
Pearson Correlation

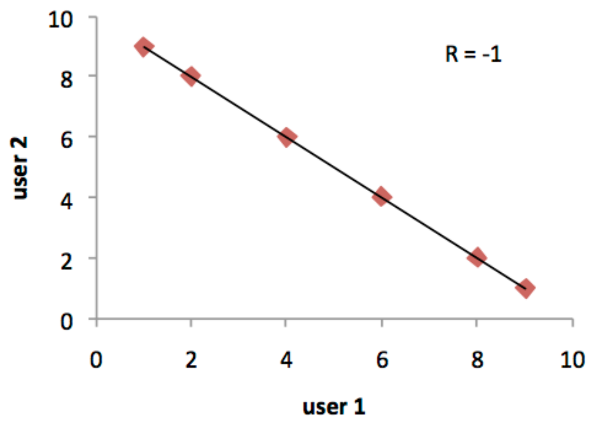
- Given by:

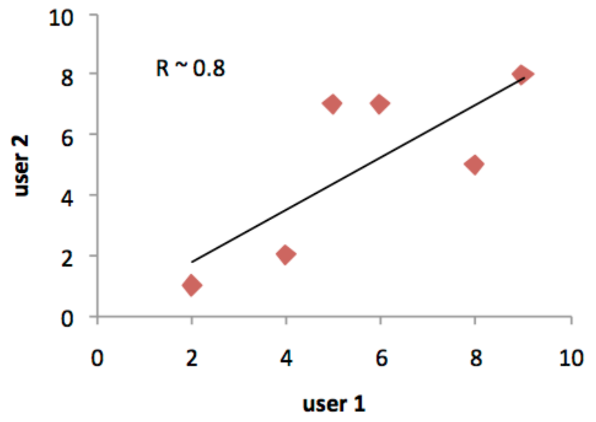
$$w_{a,i} = \frac{\sum_{j \in I_a \cap I_i} (r_{a,j} - \bar{r}_a)(r_{i,j} - \bar{r}_i)}{\sqrt{\sum_{j \in I_a \cap I_i} (r_{a,j} - \bar{r}_a)^2 \sum_{j \in I_a \cap I_i} (r_{i,j} - \bar{r}_i)^2}}$$

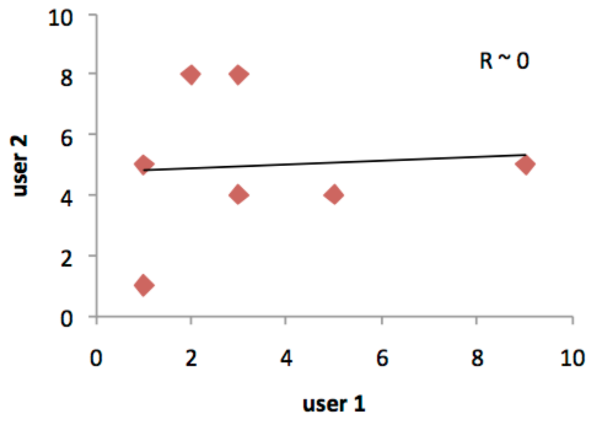
  where:
  - I_a is the set of items rated by user *a*
  - $r_{a,j}$ is the rating of user *a* for item *j*
  - $\bar{r}_a$ is the average rating of user *a*
- Summations over co-rated items only – if there are no co-rated items, similarity is set to 0.
- Results in a value of [-1,+1]
  - +1 indicates total agreement on co-rated items
  - 0 indicates no similarity between users
  - -1 indicates total disagreement
- Widely used similarity metric (applicable for unary ratings?)

Pearson Correlation Examples

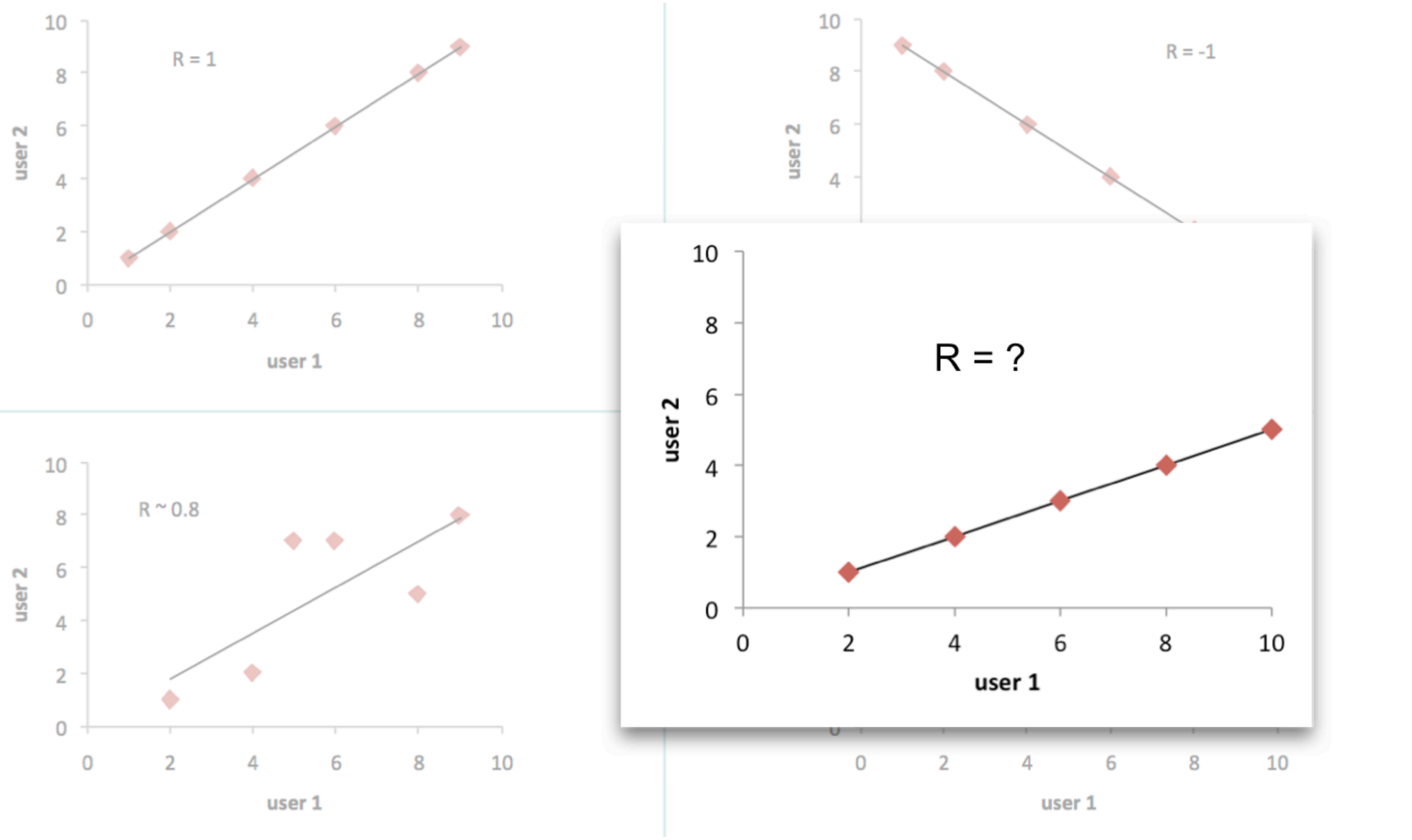




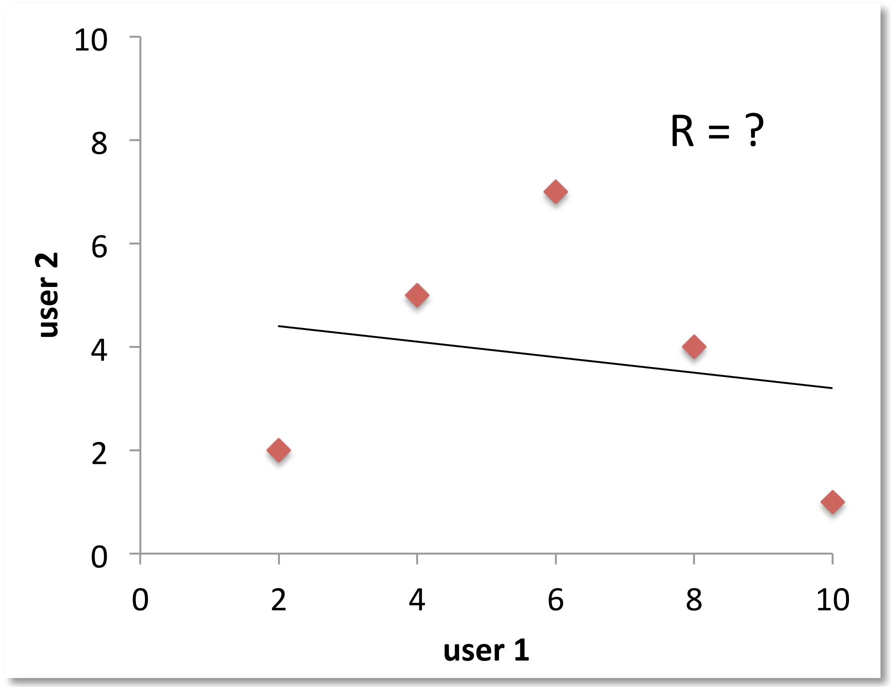




Pearson Correlation Examples

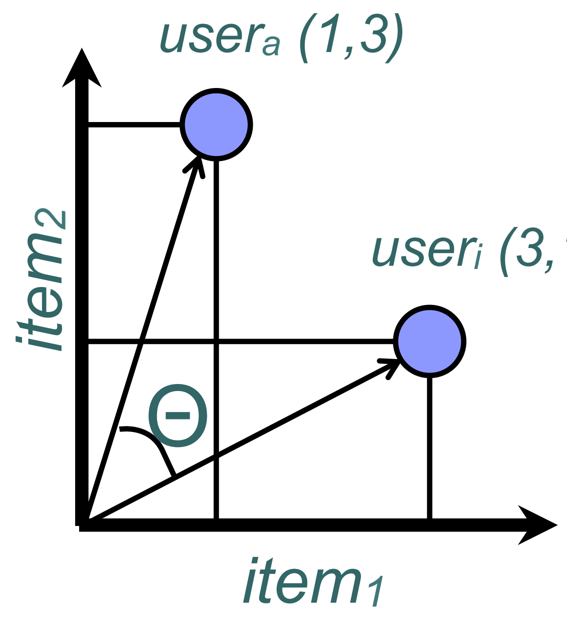


Pearson Correlation Examples



Cosine Similarity

- Users are represented as vectors in the *n*-dimensional item space
- Similarity is calculated as the cosine of the angle between the vectors



$$w_{a,i} = \frac{\sum_{j \in I_a \cap I_i} r_{a,j}\, r_{i,j}}{\sqrt{\sum_{k \in I_a} r^2_{a,k}}\ \sqrt{\sum_{k \in I_i} r^2_{i,k}}}$$

- If all $r_{i,j}$ >= 0, cosine results in a value of [0,+1]; otherwise [-1,+1]
- Normalisation – ensure than users who have rated many items are not *a priori* more similar to other users (important!)
- Applicable for unary ratings?

Extensions (1)

Significance Weighting

- Weights that are calculated over small numbers of co-rated items may provide an unreliable measure of the similarity between users
  - Consider Pearson correlation − what happens when there are only two common items?
- This extension modifies similarity weights based on the number of co-rated items (*n*) between users as follows:

$$w'_{a,i} = \begin{cases} w_{a,i} \times \frac{n}{N} & \text{if } n < N \\ w_{a,i} & \text{otherwise} \end{cases}$$

where *N* is a constant (e.g. *N* = 50)

- Improves reliability of similarity weights – similarity is modified as a function of the number of co-rated items

Extensions (1)

- Significance weighting – in the previous approach, the similarity between two users is reduced in all cases where the number of co-rated items is less than the threshold *N*
- What about users with small profiles (e.g. with less than 50 rated items)? Significance weighting will reduce similarities between such users even though they may be excellent neighbours…
- Modify similarity weights by the number of co-rated items (the intersection) between users divided by the union of items as follows:

$$w'_{a,i} = w_{a,i} \times J_{a,i}$$

where $J_{a,i} = \dfrac{|I_a \cap I_i|}{|I_a \cup I_i|}$ (Jaccard index)

Extensions (2)

Default Voting

- A key limitation is that user similarity is computed over co-rated items only
- Default voting is another approach to deal with this issue – by computing the similarity over the union rather then the intersection of users' ratings
- Can adopt a neutral default voting scheme, in which unrated items were assigned a neutral rating, indicating that the user had not yet rated these items
- Problematic if the intersection of items is small relative to the union, leads to spurious similarities…
- Other strategies – assign the average rating for unrated items computed over all users or over the most similar users who have rated it.

Extensions (3)

Case Amplification

- This extension is designed to emphasise weights that are close to 1 and to reduce the influence of lower weights.

$$w'_{a,i} = \begin{cases} w^{\rho}_{a,i} & \text{if } w_{a,i} \geq 0 \\ -[-w_{a,i}]^{\rho} & \text{if } w_{a,i} < 0 \end{cases}$$

where ρ is a constant (e.g. from the literature, $\rho = 2.5$)

- Increases the influence of more similar users on predictions and recommendations

Extensions (4)

Inverse User Frequency (IUF)

- Similar idea to inverse document frequency (IDF) term weighting in Information Retrieval:
  - Terms which are common across a collection do not distinguish between documents
- Likewise, popular items may be less helpful when computing similarities between users:
  - Give more weight to ratings for niche items
  - $\text{IUF}_k = \log(n/n_k)$, where n is the total number of users in the system and n_k is the number of users who rated item k
  - Thus popular items are assigned less weight in user similarity computations
- One approach to applying IUF is to multiply each rating by its IUF weight and use this product in place of ‘raw’ ratings in the similarity functions

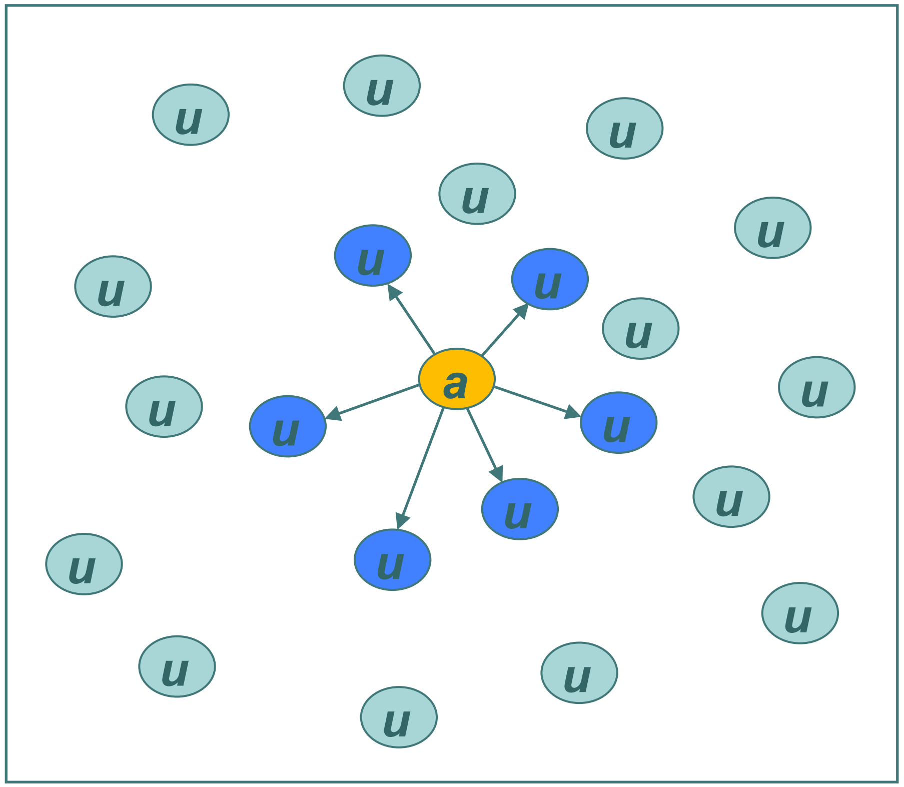
Neighbourhood Formation

- Concerns the process by which those users (neighbours) that contribute to predictions and recommendations are selected
- Different approaches have implications for algorithm performance (e.g. accuracy, coverage..., more later)

k Nearest-neighbour (kNN)

- Forms a neighbourhood of size *k* by simply picking the *k* users with the highest similarity to the active user
- Neighbourhood size is application dependent => typically chosen by experiment
  - Picking a large *k* can lead to reduced accuracy by diluting the influence of the most similar neighbours
  - Picking a small *k* can result in poor accuracy for those users with few/no close neighbours

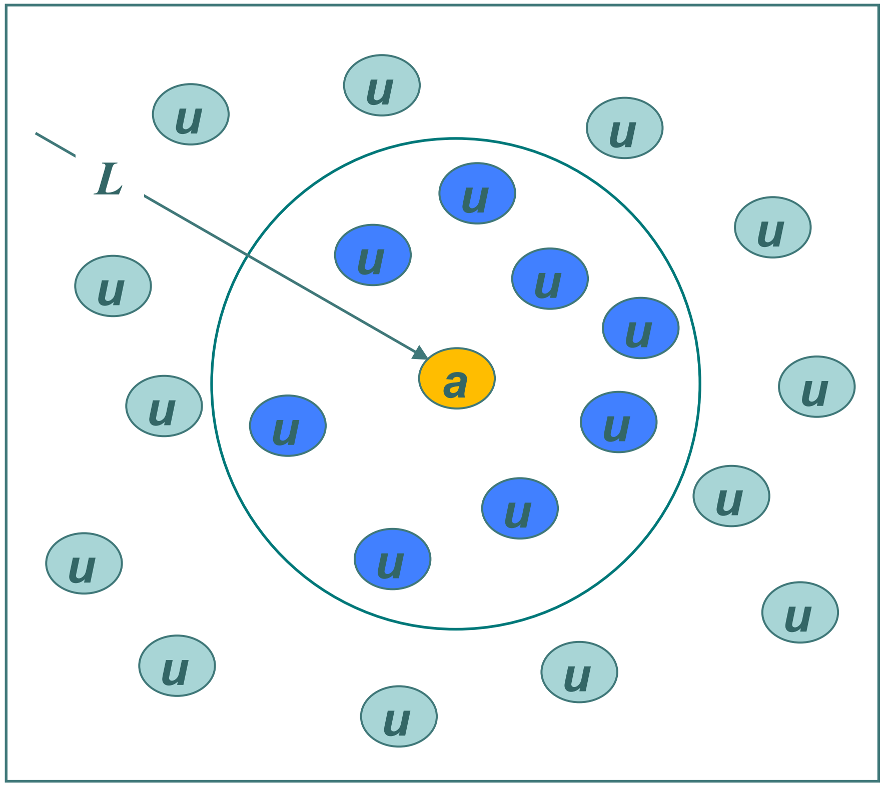
$k = 6$



Similarity Thresholding

- Users whose similarities with the active user exceed a threshold *L* are selected as neighbours
- Motivation => restrict neighbourhood to the highest value neighbours
- Setting a high threshold can result in:
  - Good accuracy for users with close neighbours
  - However, for users without close neighbours – neighbourhoods may not be formed (no user satisfies the criterion) => no recommendations
- Setting low thresholds results in large-sized neighbourhoods and thereby nullifies the purpose of the scheme

Select all u s.t. $w_{a,i} > L$



Combined kNN and Similarity Threshold

- This scheme combines the features of kNN and thresholding, where neighbourhoods of at most size *k* are formed with users whose similarity to the active user exceeds the threshold value
- In previous work (MovieLens), neighbourhoods with *k* = 20 and an threshold of *L* = 0.1 were found to provide good results…
- Parameters are domain dependant…

Predictions & Recommendations

- Recommender systems can be used to:
  - Compute predictions
  - Generate a tanked top-N list of items
- Various approaches that have been applied to the above

Making Predictions (1)

Simple Average Approach

- Compute the mean of the neighbours' ratings for the target item:

$$p_{a,j} = \frac{\sum_{i=1}^{n} r_{i,j}}{n}$$

where $p_{a,j}$ is the prediction for the active user *a* on item *j* , $r_{i,j}$ is the rating of neighbour *i* for item *j*, and *n* is the number of neighbours

- But this approach does not take neighbour similarity into account...
- Neighbours which are less similar to the active user have the same influence on the predicted rating as neighbours which are very similar to the active user…
- How can this approach be improved?

Making Predictions (2)

Weighted Average Approach

- Compute a weighted average of the neighbours’ ratings for the target item (better):

$$p_{a,j} = \frac{\sum_{i=1}^{n} w_{a,i} \; r_{i,j}}{\sum_{i=1}^{n} |w_{a,i}|}$$

  where $p_{a,j}$ is the prediction for the active user *a* on item *j* and $w_{a,i}$ is the similarity between users *a* and *i*

- The assumption inherent in this approach is that all users rate items according to (approximately) the same distribution.
- However, this is not a safe assumption to make, e.g. some users may be more critical when assigning ratings than others
- This observation has lead to the introduction of the next approach…

Making Predictions (3)

Deviation from User Mean Approach

- Some users may be very critical, rarely assigning the top rating to any item:
  - Such users should rarely receive maximum predictions
- In contrast, other users may generally give higher ratings to items:
  - These users would expect to receive high or maximum predictions for the majority of liked items
- Key point: rating distributions of users are centered around different points
- ***Resnick's*** algorithm – deviation from user mean approach (normalisation) – target item is assigned a rating that is an adjusted form of the active user's mean rating:

$$p_{a,j} = \bar{r}_a + \frac{\sum_{i=1}^{n} w_{a,i} \left(r_{i,j} - \bar{r}_i\right)}{\sum_{i=1}^{n} \left| w_{a,i} \right|}$$

Normalisation

Approaches:

- Mean centering – for each user, subtract the user's mean rating from all ratings assigned by the user
- z-score normalisation − takes into account the spread in ratings across the scale. For each user, perform mean centering and then divide by the standard deviation of ratings (experiments indicate only small improvements in prediction accuracy observed using this approach...)
- Normalise by item mean ratings (more later)

Making Recommendations

- Form a candidate set of items for recommendation by taking the union of all the items that have been rated by the active user's neighbours.
- From this set, those items that have already been rated by the active user may be excluded (depending on domain)
- The remaining items can be ranked according to, for example:
  - **Frequent Item** – the percentage of neighbours who have rated the item (subject to a minimum threshold rating)
  - **Liked Item** – the mean item rating across neighbours
  - **Predicted Rating** – make a prediction for each of the candidate items
- The top-*N* items with the highest scores are returned as recommendations

Performance Evaluation

- There are many possible approaches to recommendation...
- Need to perform rigorous evaluation using systematic scientific experiments...
- How do we know if the recommender delivers high quality recommendations?
- What do we mean by high quality?

Performance Evaluation

- Live-User Analysis vs Off-line Evaluation
  - Analysis of real usage and prediction/recommendation feedback. A/B testing to evaluate different algorithms. Facilitates a holistic approach to system evaluation.
  - Offline evaluations are best suited to testing core recommendation algorithm components by using existing user-rating datasets.

- Synthetic vs Natural Data Sets
  - Natural data sets based on historical live-user data (e.g. MovieLens, EachMovie, BookCrossing, Netflix data sets), which contain real user profiles with genuine ratings.
  - Synthetic data sets − not derived from live users. Artificially created with specific properties, often to test particular aspects of an algorithm that might not be facilitated by existing natural data sets.
  - Data set characteristics – size, sparsity, diversity, etc.

Evaluation Methodologies

- Repeated Random Sub-Sampling
  - Randomly split the data into training and test sets; e.g. 5x 80/20 splits.
  - For each split, evaluate the test data based on predictions/recommendations made using the training data only.
  - Average evaluation results across the splits.
- K-Fold Cross Validation
  - Randomly partition the data set into K subsamples.
  - Of the K subsamples, a single subsample is used as the test data, with the remaining K-1 subsamples as training data.
  - Repeat K times (the folds), using each subsample in turn as the test set.
  - Average evaluation results across the folds.
- Leave-One-Out
  - Select a single observation from the data set (e.g. a single user-item-rating) as the test data with the remaining data as the training set (useful when the quantity of data available is limited).
  - Repeat for each of the observations and average over the individual tests.

Evaluation Metrics

Prediction Accuracy:

- Mean Absolute Error (MAE):
  - $p_{a,j}$ and $r_{a,j}$ are the predicted and actual ratings for user *a*, item *j*, resp.
  - *T* is the test set.
  - Calculated over test set ratings.

$$\text{MAE} = \frac{1}{|T|} \sum_{(a,j) \in T} |p_{a,j} - r_{a,j}|$$

- Root Mean Squared Error (RMSE):
  - Penalises larger errors over smaller errors.
  - Calculated over test set ratings.

$$\text{RMSE} = \sqrt{\frac{1}{|T|} \sum_{(a,j) \in T} (p_{a,j} - r_{a,j})^2}$$

Evaluation Metrics

Recommendation Accuracy:

- Precision and Recall:
  - Precision can be seen as a measure of exactness or fidelity, whereas Recall is a measure of completeness.
  - Precision (*Pr*) represents the probability that a recommended item is relevant.
  - Recall (*Re*) represents the probability that a relevant item is recommended.

- F_1 Measure:
  - Precision and recall are often conflicting metrics. For example, increasing the number of recommendations is likely to improve recall, but reduce precision.
  - To resolve this conflict, the F_1 measure, which combines the precision and recall metrics, can be used.

$$Pr = \frac{|T \cap R|}{|R|} \qquad Re = \frac{|T \cap R|}{|T|} \qquad F_1 = \frac{2 \times Pr \times Re}{Pr + Re}$$

(where, for a given user, *T* is the test set and *R* is the recommended set)

Performance Evaluation

Other Considerations:

- **Coverage** – one variation: the percentage of users for whom predictions/ recommendations can be made − if a neighbourhood cannot be formed, then *personalised* predictions/recommendations cannot be made. (Other variations include recommendation coverage and item-space coverage.)
- **Novelty & Serendipity** – recommend items the user is unaware of but would like
- **Diversity** – ensuring that top-N lists are not comprised of only “similar” items
- **Learning Rate** – how quickly CF becomes an effective predictor of taste as more data is collected
- **Confidence** – describes the CF system’s ability to evaluate the likely quality of predictions
- **Site performance metrics** – tracking system throughput (items purchased or downloaded, the number of active users etc.)
- **Robustness** – are CF systems vulnerable to manipulation? (**Yes!**)

CF – Advantages

Advantages

- **Quality & Taste**:
  - People have the ability to analyse information based on quality and taste
  - Since CF operates on user preferences, it inherits this ability
  - Other filtering techniques are less able to quantify the quality that is inherent in certain items – e.g. a content-based search may return relevant items, but some of these may be of poor quality...

- **Item Features**:
  - CF algorithms make predictions / recommendations based solely on user preferences
  - Do not rely on any item features that need to be pre-determined and extracted prior to filtering
  - CF algorithms can be readily implemented in complex domains which contain items that are difficult to analyse by automated processes

CF – Advantages

- **Serendipitous Recommendations**:
  - CF algorithms have the ability to provide serendipitous recommendations – i.e. where recommended items are not known to the user but which are of interest

<u>Limitations</u>

- **Cold-Start Problem**:
  - CF algorithms rely on relationships between users and items in order to make predictions / recommendations
  - In recently commissioned applications, such relationships may not yet exist or may be based on only limited quantities of data
  - Can seed system with ratings from review sites, bestseller lists, box office returns etc.

CF – Limitations

- **Early Rater Problem**:
  - This issue applies to predictions that are sought for items that are new or only recently added to a system.
  - For such items, predictions are problematic, since there are no (or few) ratings contained in the system on which to calculate predictions
  - Similarly, newly registered users are problematic for even established systems, until they have rated at least some of the available items
- **Sparsity Problem**:
  - Typically, recommender system datasets are very sparse, because users will have assigned ratings to only a relatively small number of items (<< 1%).
  - This is particularly true for e–commerce systems, where very large numbers of items are frequently offered for sale (e.g. consider Amazon.com)
  - Thus it may not be possible to make accurate (or any) predictions / recommendations for certain users and items

CF – Limitations

- **Scalability**
  - User-based CF algorithms suffer from serious scalability problems:
    - Computations grow with both the number of users and items
    - With potentially millions of users and items, typical real-world systems running user-based CF algorithms will suffer serious scalability problems
  - Solutions have been proposed to improve scalability performance:
    - Model-based approaches
    - Cluster the database into groups of like-minded users from which all neighbours are subsequently drawn; probabilistic models; reduced dimensional representation of the user-item space, etc.
    - Matrix factorisation approaches, item-based CF…

Item-based CF

- Bottleneck in user-based CF: the search for neighbours (in real time) among a large user population of potential neighbours
- Item-based CF – avoid bottleneck by finding relationships between *items* as opposed to between *users*
- Item-item similarities – more “stable” than user-user similarities:
  - In some domains, the set of items is relatively static compared to the user base (the latter changes more frequently – new users arrive, users leave)
  - For “established” items, i.e. those with a significant number of ratings, the addition of a new rating for an item will not significantly affect item-item similarities...
  - …Do not need to re-compute item-item similarities every time a new rating is recorded => re-compute similarities offline (e.g. every 24 hours) => improved scalability
- Item-based CF:
  - Intuition: users are interested in items similar to those previously experienced
  - Compute similarities between items (in place of similarities between users), and recommend items that are similar to items the user has already rated…

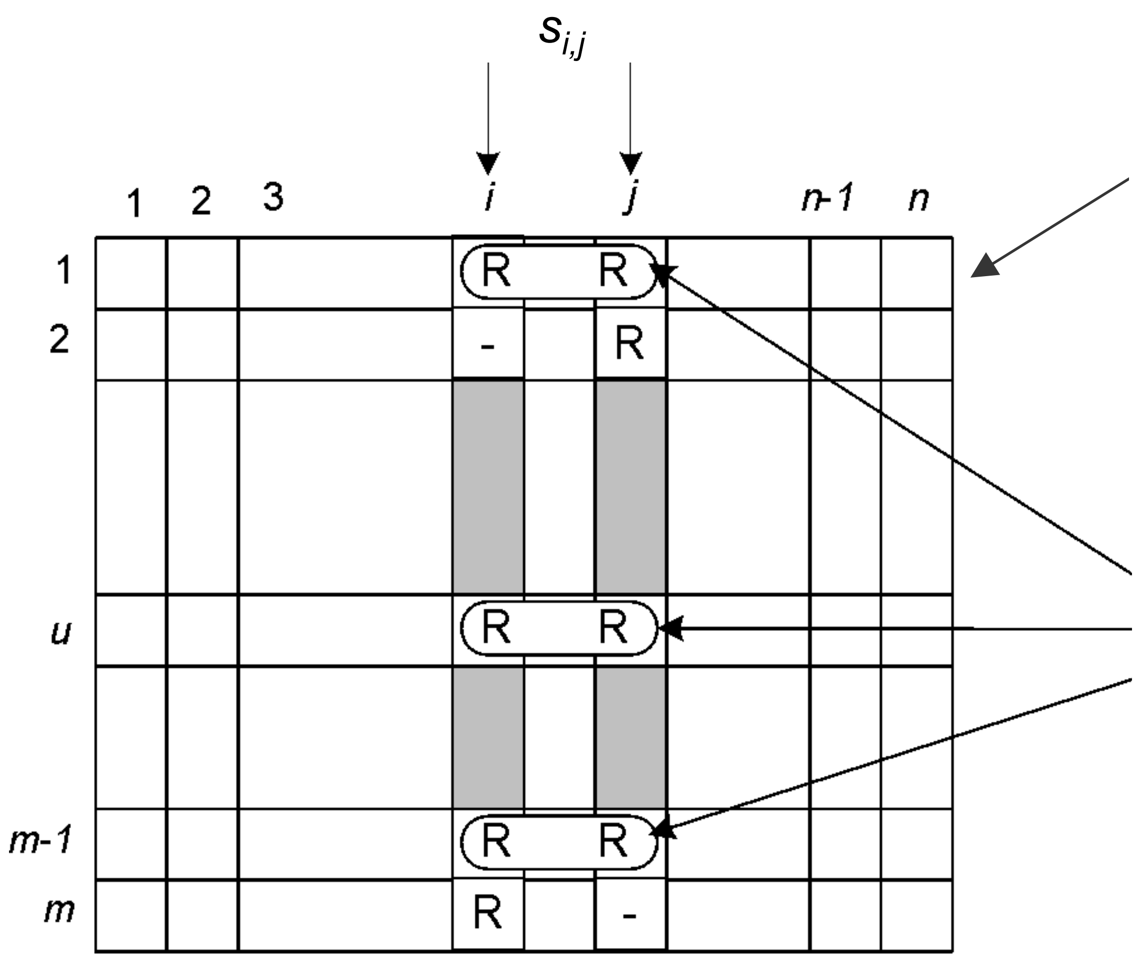
Item-based CF Algorithm

Algorithm:

```
1. Data Representation:
       Obtain preference data from users
       Construct (conceptually) a **user-item matrix**

2. Similarity Computation:
       Compute the pairwise similarities between all items

3. Neighbourhood Formation:
       For each item *i*, create an item neighbourhood consisting of
       a subset of other items which have the highest similarity
       to item *i*

4. Make a prediction or generate a top-N list
```

Different approaches used in the above steps …

Item Similarity



User-item matrix

The similarity ($s_{i,j}$) between two items *i* and *j* is calculated over the set of users which have rated both items

In this example, $s_{i,j}$ is calculated over the set of users {*1*, *u*, *m-1*}

Item Similarity

- Approaches include:
  - Pearson correlation
  - Cosine similarity
  - Note – in the above, similarities are computed across *items* instead of *users* (i.e. across the columns of the user-item matrix)

- Another approach is Adjusted Cosine similarity (**Sarwar 2001**):
  - Takes into account that users rate items according to different distributions…
  - Let *U* denote the set of users which have rated both items *i* and *j*; similarity is given by:

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

Neighbourhood Formation

- Identify the most similar items to each item in the system
- Approaches:
  - k nearest neighbour (kNN) – for each item *i*, the neighbourhood consists of the *k* most similar items to item *i*
  - Similarity thresholding – for each item *i*, the neighbourhood consists of all items with similarity (to item *i*) in excess of a threshold, *T*
  - Combined kNN and similarity thresholding – for each item *i*, the neighbourhood consists of (up to) the *k* most similar items with similarity (to item *i*) in excess of a threshold, *T*
- Parameters are domain-dependent. As with user-based CF, the different approaches have implications for algorithm performance…

Making Predictions (1)

Simple Average Approach

- A prediction for user *a* on item *j* is given by:

$$p_{a,j} = \frac{\sum_{i \in I_a \cap \mathcal{N}_j} r_{a,i}}{|\{i : i \in I_a \cap \mathcal{N}_j\}|}$$

  where $r_{a,i}$ is the rating for user *a* on item *i*, I_a is the set of items rated by user *a*, and $\mathcal{N}_j$ is the neighbourhood of item *j*

- The prediction is calculated by simply taking the average of a subset of the user's own ratings – the subset is the intersection between the user's previously rated items and the items in the neighbourhood of *j*
- However this approach does not take item similarity into account… …the ratings for the user's items which are more similar to *j* should be afforded more influence on the prediction…

Making Predictions (2)

Weighted Average Approach

- A prediction for user *a* on item *j* is given by:

$$p_{a,j} = \frac{\sum_{i \in I_a \cap \mathcal{N}_j} s_{i,j} \, r_{a,i}}{\sum_{i \in I_a \cap \mathcal{N}_j} s_{i,j}}$$

  where $s_{i,j}$ is the similarity between items *i* and *j*

- As before, the prediction is calculated by taking the average of a subset of the user’s own ratings – here the average is weighted by item similarity
- However this approach does not take into account biases in item ratings…

Making Predictions (3)

Deviation from Item Mean Approach

- As per the previous approach (i.e. calculates a weighted average), but also takes into account the differences between the user's ratings for items compared to the mean ratings for items:

$$p_{a,j} = \bar{r}_j + \frac{\sum_{i \in I_a \cap \mathcal{N}_j} s_{i,j} \left(r_{a,i} - \bar{r}_i\right)}{\sum_{i \in I_a \cap \mathcal{N}_j} s_{i,j}}$$

where $\bar{r}_j$ is the mean rating for item *j*

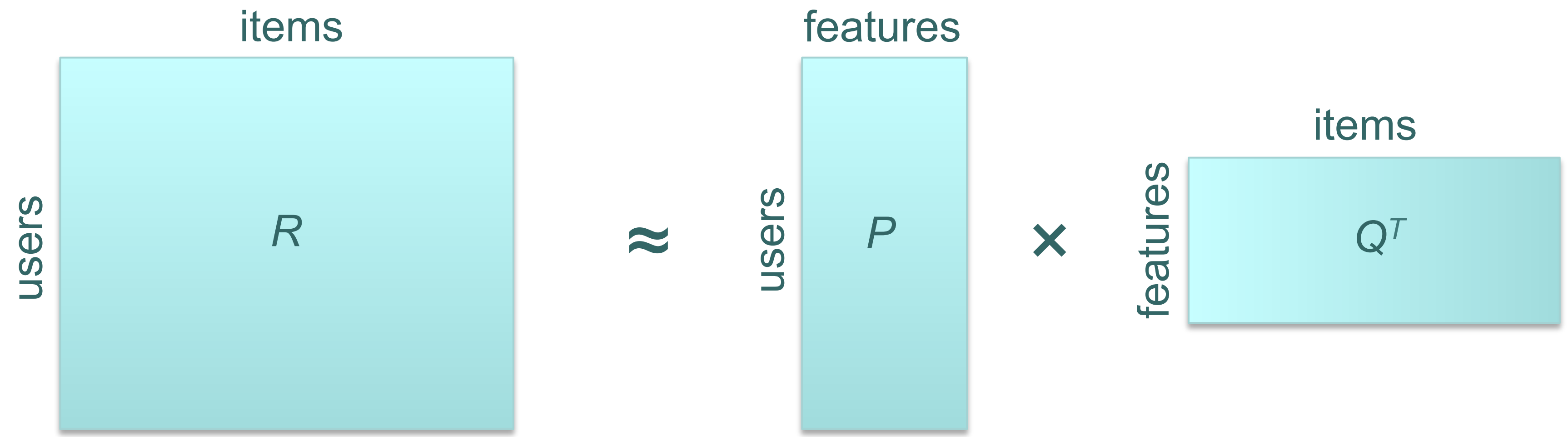
- Analogous to ***Resnick's*** user-based deviation from user mean algorithm.

Matrix Factorisation

- Matrix factorisation collaborative filtering approaches assume that there exists some *latent features* which capture user preferences.
- For example, users may assign high ratings to a particular movie if they like the movie’s director, or the movie’s actors, or the movie’s genre…
- The goal is to discover these latent features and use them to predict a user’s rating for an item by matching the features associated with the user to those of the item.
- But we do not know what these latent features are…
  How can we proceed?

Matrix Factorisation

- Let *R* denote the user-item matrix (a |*U*| x |*I*| matrix)
- Find two matrices *P* (|*U*| x *K* matrix) and *Q* (|*I*| x *K* matrix) such that their product approximates *R* using *K* latent features



$$R \approx P \times Q^T = \hat{R}$$

Matrix Factorisation

- What do matrices *P* and *Q* represent?
  - Each row of *P* represents the strength of the association between a user and each of the *K* features.
  - Each row of *Q* represents the strength of the association between an item and each of the *K* features.
- Once *P* and *Q* are computed, a predicted rating for user *i*, item *j* is given by the dot product of the two vectors corresponding to user *i* and item *j*:

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{K} p_{ik}\, q_{kj}$$

Obtaining Matrices *P* and *Q*

- One way is the populate *P* and *Q* with initial values, compute how different their product is to *R*, and then minimise this difference iteratively.

- The difference is given by the error between the predicted and actual ratings.

- For each rating in the training set, the error is computed as:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \textstyle\sum_{k=1}^{K} p_{ik}\, q_{kj})^2 \qquad \textbf{(1)}$$

- The error over all ratings in the training *T* is given by:

$$E = \textstyle\sum_{(u_i, d_j, r_{ij}) \in T} e_{ij}^2 = \sum_{(u_i, d_j, r_{ij}) \in T} (r_{ij} - \sum_{k=1}^{K} p_{ik}\, q_{kj})^2$$

Obtaining Matrices *P* and *Q*

- Gradient descent − to minimise the error, we need to determine in which direction we have to modify the values – i.e. need to know the *gradient* at the current values of $p_{i\,k}$ and $q_{k\,j}$

- Differentiate Equation (1) with respect to these two variables:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}\, q_{kj}$$

$$\frac{\partial}{\partial q_{kj}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}\, p_{ik}$$

- Having obtained the gradient, we can now formulate the update rules for both $p_{i\,k}$ and $q_{k\,j}$ (α is the learning rate ~0.001)

$$p'_{ik} = p_{ik} + \alpha \frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij}\, q_{kj}$$

$$q'_{kj} = q_{kj} + \alpha \frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}\, p_{ik}$$

Obtaining Matrices *P* and *Q*

- For each factor, iterate over each rating in the training set (updating the values of $p_{i\,k}$ and $q_{k\,j}$ each time using the update rules) until the error is minimised.

- The above is an example of one approach, many variations exist. See e.g. (**Ott 2008)** for more details.

- Matrix factorisation approaches perform well in practice. A matrix factorisation approach was used (along with other approaches) by the winning team in the Netflix prize.

Summary

- Personalised vs. non personalised recommenders
- Collaborative filtering − personalised approach to prediction and recommendation
- Different approaches – user-based, item-based and matrix factorisation collaborative filtering algorithms
- Evaluation methodologies and metrics
- Advantages and limitations