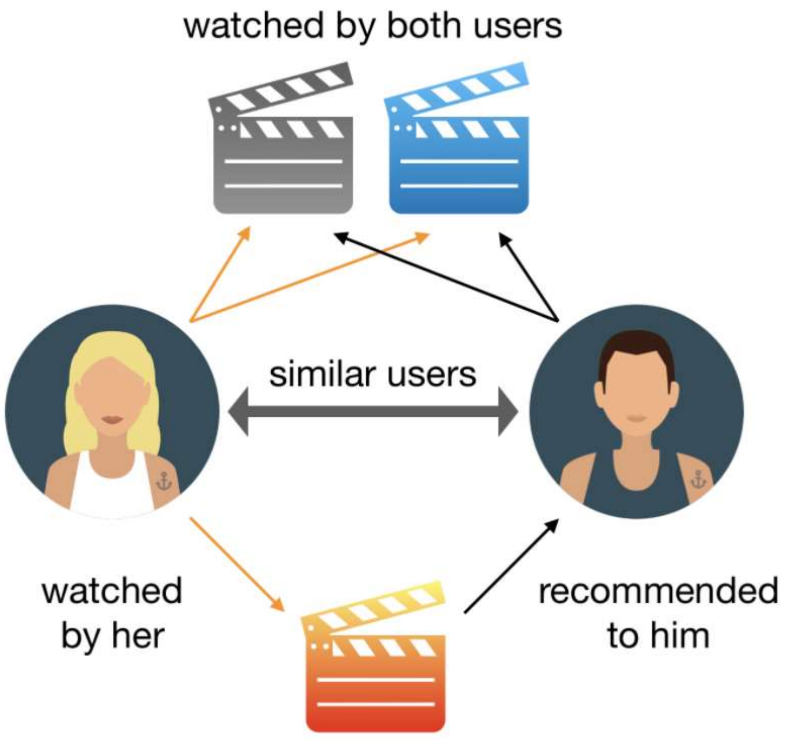
Graduate Certificate in Big Data Analytics

Personalised Recommender Systems

Dr. Barry Shepherd
Institute of Systems Science
National University of Singapore
Email: barryshepherd@nus.edu.sg



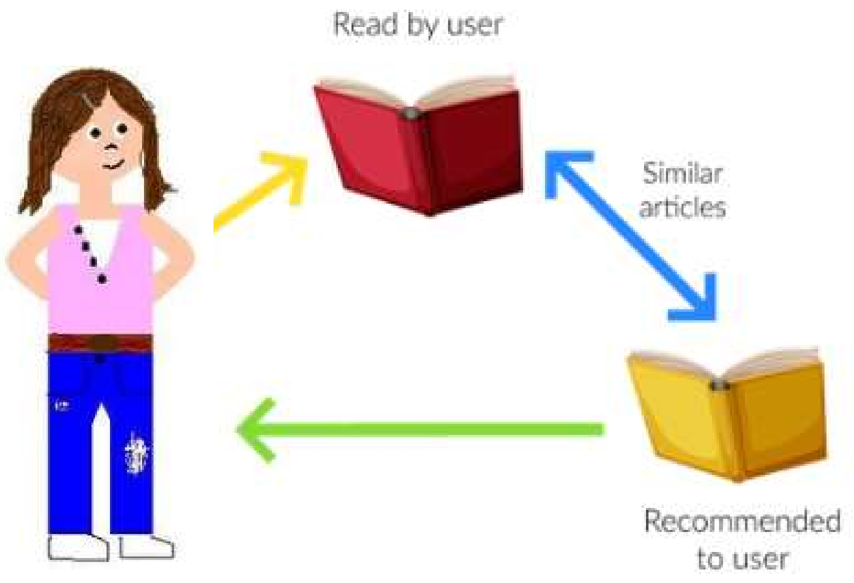
Topics

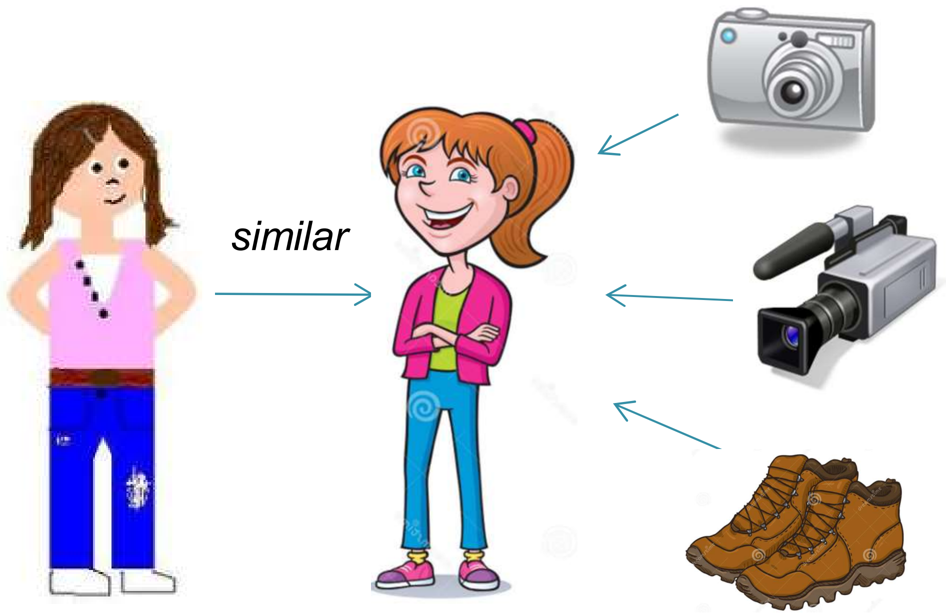
- Personalised Recommender System Approaches
- Content-Based Filtering (overview)
- Collaborative Filtering
- Evaluating Recommender Systems

Personalised Approaches

- Content-Based Filtering
  - Recommend based on what you have bought or viewed in the past
  - Match users directly to products & content
  - Commonly used for document recommendation: webpages, news articles, blogs etc.

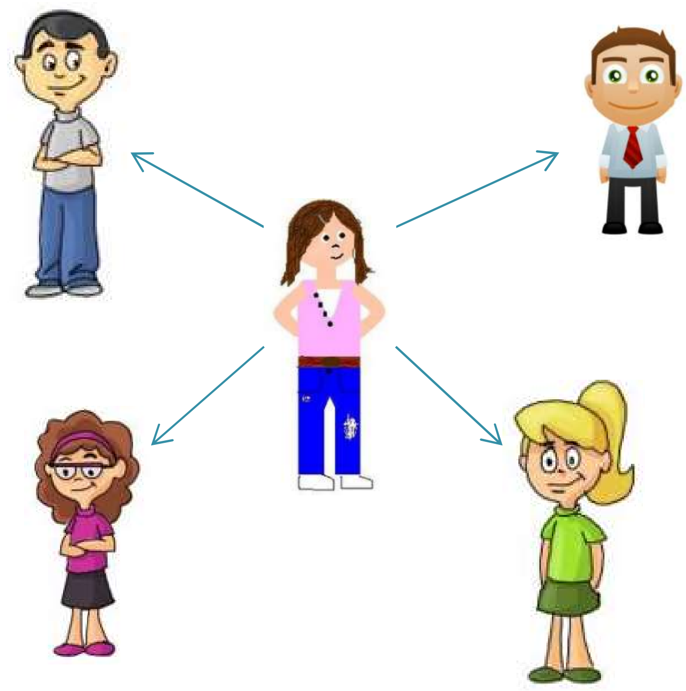
- Collaborative Filtering
  - Match users to people with similar tastes – recommend what they like
  - Commonly used in e-retail
  - Avoids the issue of users only being recommended more of what they already like (allows serendipity)





Other Personalised Approaches

- Community-Based/Social
  - Who are your friends and what do they like?
  - Users trust their friends opinions



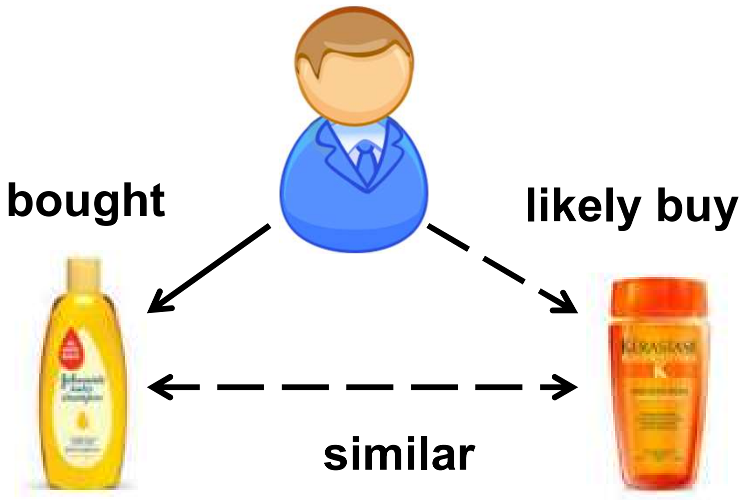
- Knowledge-Based
  - Often more appropriate for recommending products where many **user specific requirements and constraints** must be taken into account
    - E.g. cars, houses, computers, financial services
  - Main types:
    - Constraint-based
    - Case-Based
  - Issues
    - Acquiring the knowledge, the knowledge acquisition bottleneck

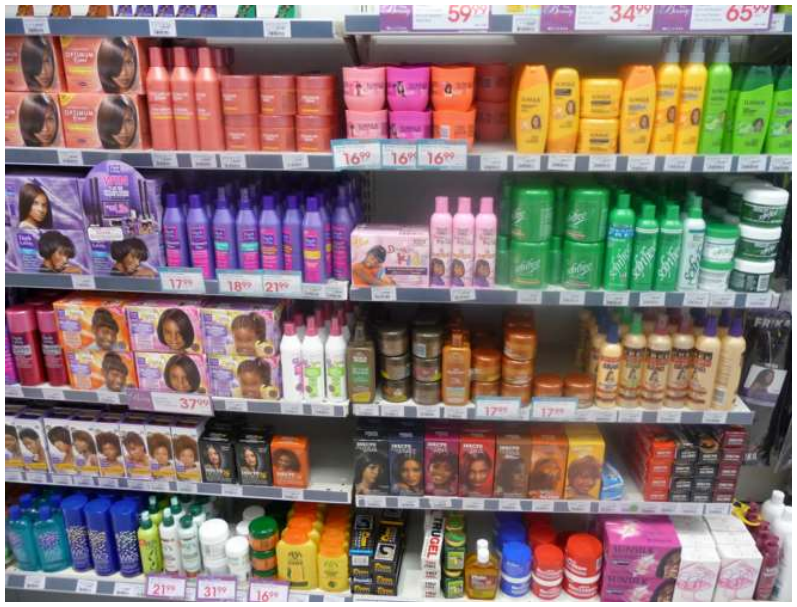
Content-Based Filtering

Match the attributes of a user with those of an item.

The main components are:

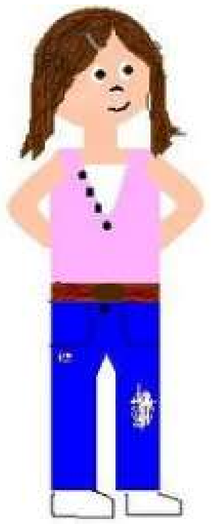
- **Content Analyser** ~ extracts structured product descriptions from a diversity of products
  - E.g. for movie: director, length, year, genre etc.
  - Data may already be available – e.g. provided by the manufacturer
  - **OR** extract using text mining -parse product descriptions / product webpages and extract features using keyword extraction
- **Profile Learner** ~ collects data about the user preferences and generalises into a user profile
- **Filtering** ~ matches the user profile against the product descriptions. Returns the products with the closest match





Content-Based Example

	Length	studio	Director	Lead Actor1	Genre	Date
movie1	90	Disney	Spielberg	Tom Hanks	Sci-fi	2014
movie2	110	MGM	Petersen	Brad Pitt	action	2012
movie3	120	Disney	Nolan	C. Bale	action	2008
…..						
movieN						



Which Movie(s) might our user like?

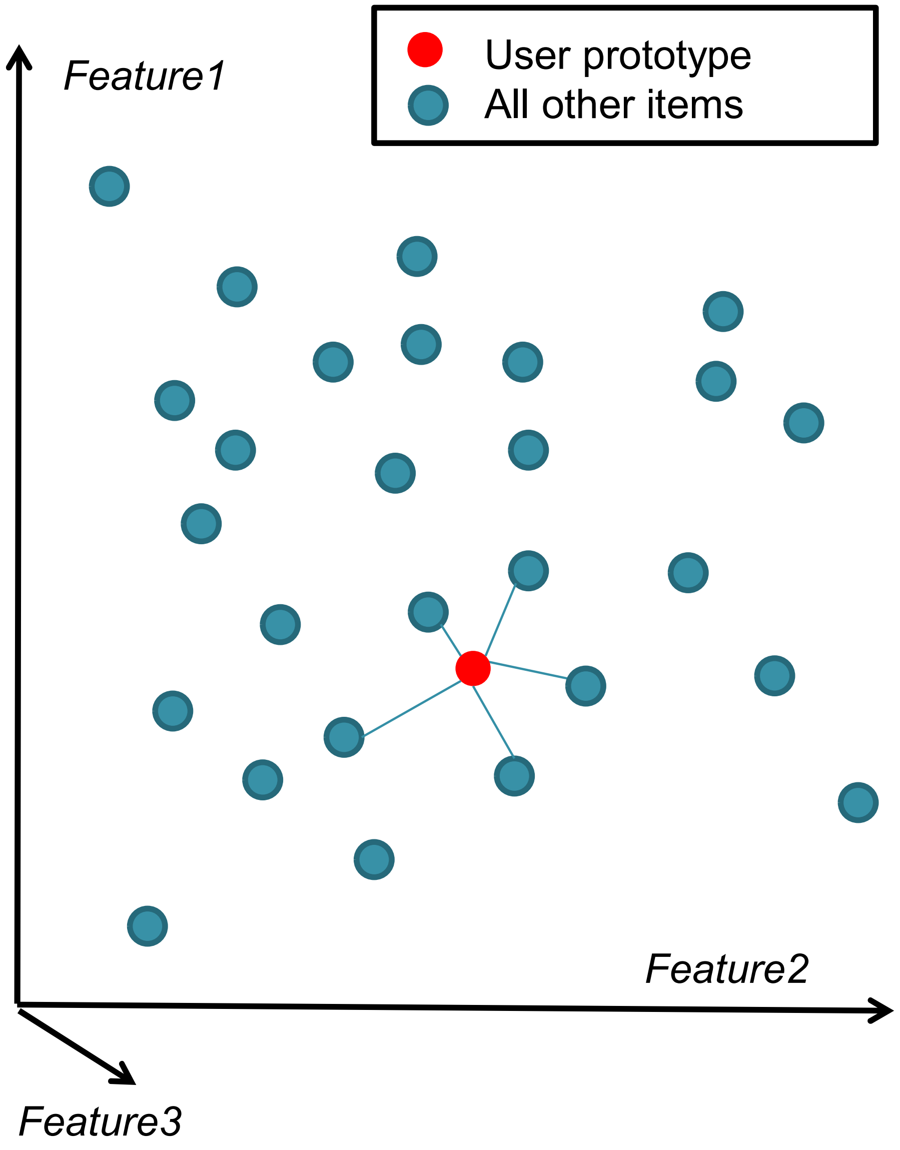
Method1: Neighbourhood approach

Method2: Classification approach

Neighbourhood Approach

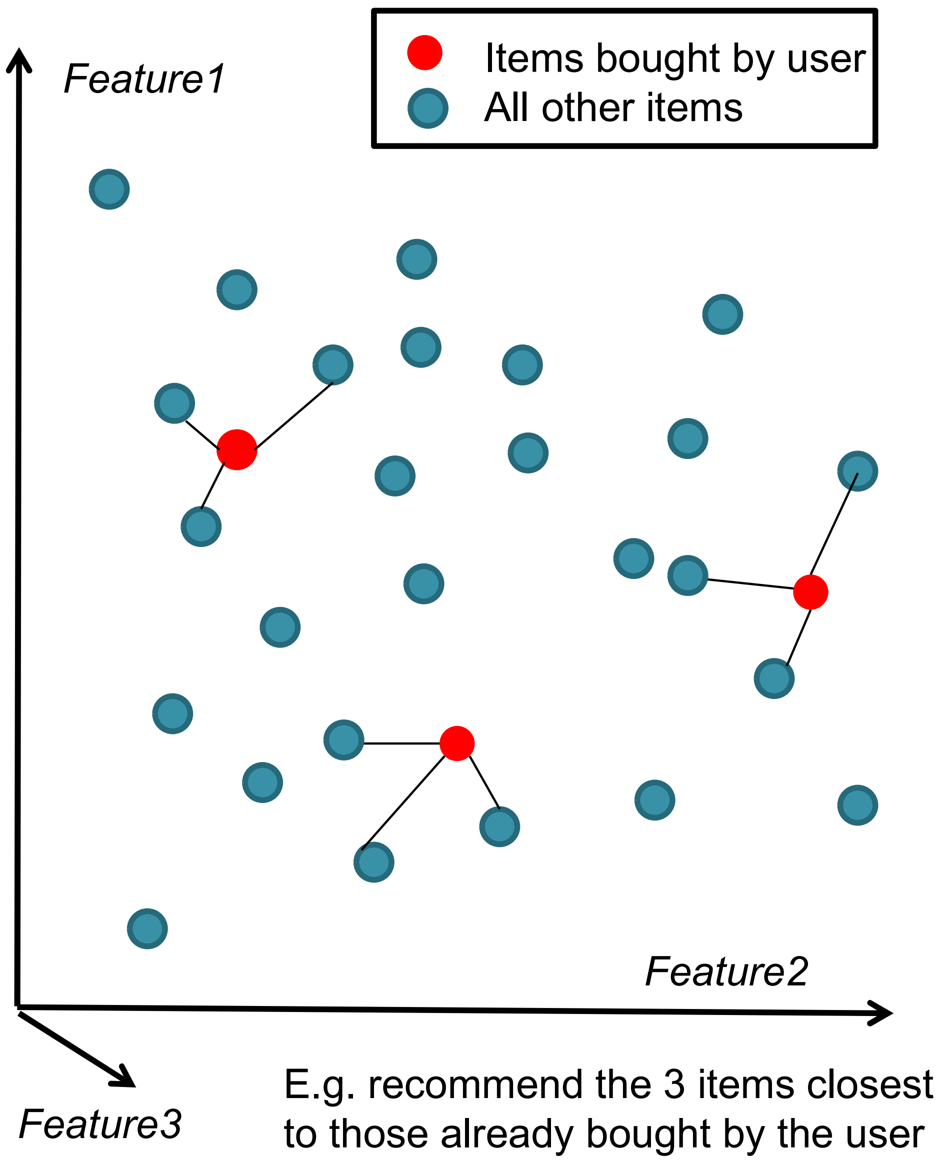
- Each item is represented by a set of features (a ***feature vector***)
- The user is represented by the same feature set, often obtained by:
  - The user selects preferences for the various features using pull-down menus
  - Or… computing an "average" vector from items already bought/liked by the user. This will represent a "typical" item for the user (often called a "prototype vector)
- Then use a distance (or similarity) measure to find the nearest items to the user
- The nearest items are then recommended – ranked by distance

**Also known as Vector-Space approach*



Neighbourhood Approach

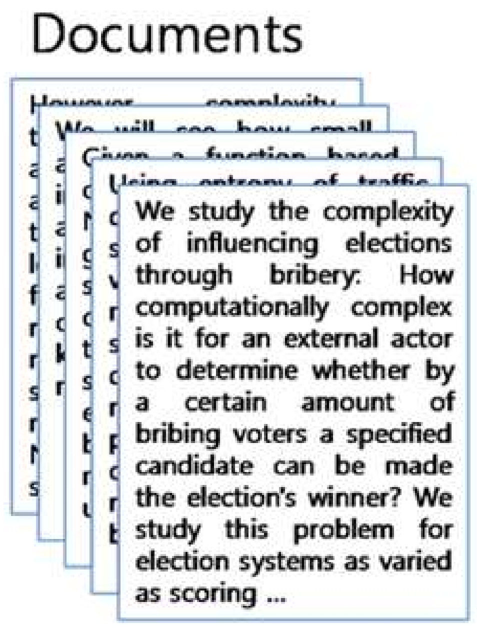
- If the retailer is selling a diverse set of products there is probably no single "typical" item for a user
- Instead, we can recommend items that are close to the items that the user has already expressed an interest in:
  - Previously purchased items
  - Viewed items
  - Searched for items (on the retailer's website)
  - Rated or 'liked' items
  - Etc.



E.g. recommend the 3 items closest to those already bought by the user

Deriving Product Features

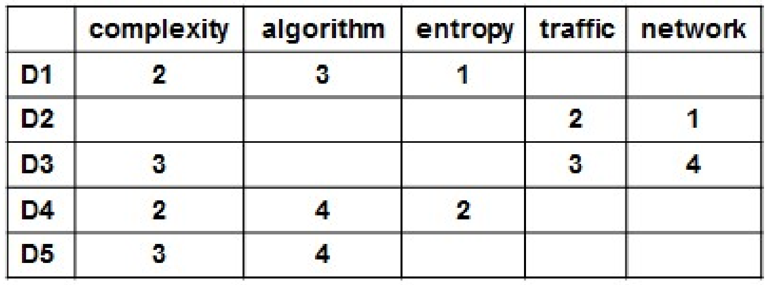
- Instead of explicitly defining product features we can extract text-based features from product documentation, product reviews, product webpages etc.
- E.g. The ***Bag-of-Words*** approach (BOW) represents a document by features that count the number of times each word (or term) occurs in the document



We study the complexity of influencing elections through bribery: How computationally complex is it for an external actor to determine whether by a certain amount of bribing voters a specified candidate can be made the election's winner? We study this problem for election systems as varied as scoring ...

Documents are pre-processed by operations like tokenization, stop-word removal, stemming

Terms can be single words or pairs, triples etc (N-grams)



	complexity	algorithm	entropy	traffic	network
D1	2	3	1		
D2				2	1
D3	3			3	4
D4	2	4	2		
D5	3	4			

Document-term matrix (DTM)

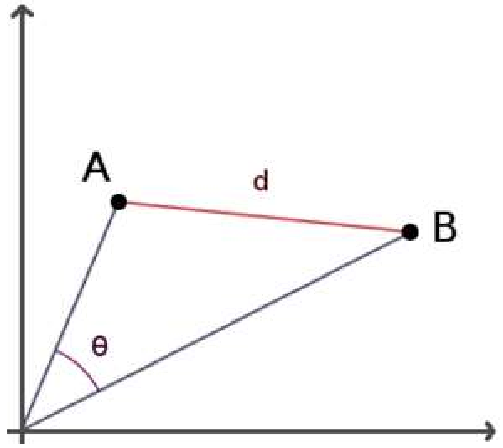
The term counts are usually weighted using TF-IDF

Similarity & Distance Measures

- Content-Based Recommender Systems typically use the cosine distance to measure the similarity between user and items
- In information retrieval systems* the similarity between two documents (A and B) is given by:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

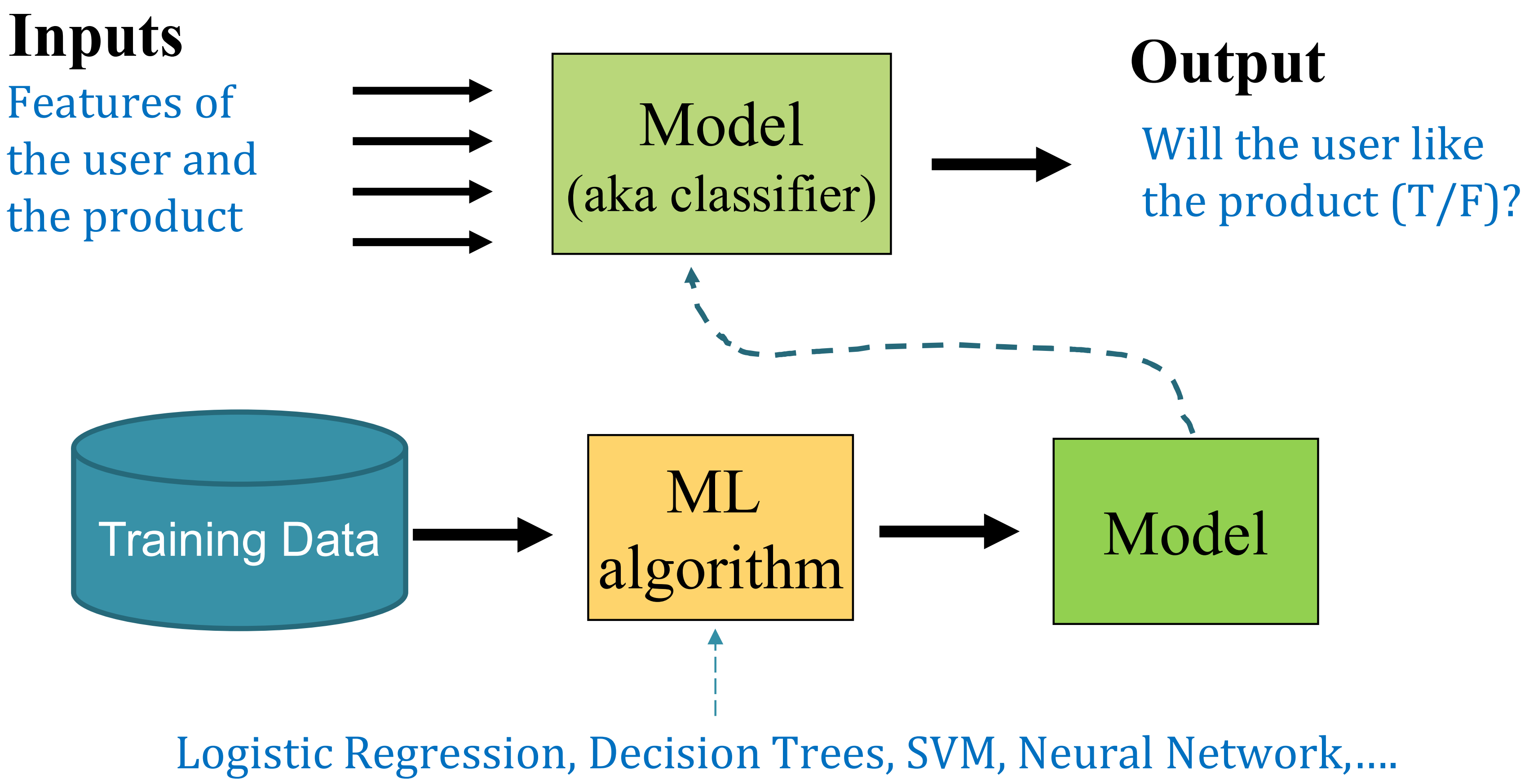
**Content-based recommender systems have their roots in Information Retrieval (IR). The goal of IR is to identify relevant documents in response to a user query*



When the documents are similar the angle between their vectors is small and the cosine similarity approaches +1

Classification Approach

Use Supervised Machine Learning to build a classification model:



Inputs
Features of the user and the product
Model (aka classifier)
Output
Will the user like the product (T/F)?
Training Data
ML algorithm
Model
Logistic Regression, Decision Trees, SVM, Neural Network,....

[Building a recommender system via classification - Recommending Products | Coursera]

Classification Approach Example

To obtain the training data, get the users to label a sample of the items as like/dislike, or infer like/dislike from user behaviors, e.g. from past purchases:

*Product Features**

Length	Studio	Director	Genre
90	Disney	Spielberg	action
110	MGM	Petersen	romance
120	Disney	Nolan	scifi
100	Warner	Jackson	fantasy

+

*User Preferences & Demographics**

Age	Gender	Length	Director	Genre
34	M	90	Nolan	scifi
18	F	120	Scorsese	comedy
49	F	100	Jackson	action
64	F	90	Lucas	action

Target

Like
T
F
T
F

** Data pre-processing may be required before model building*

Example decision tree model:

```
If Length >=180
    If Genre == Action
        If Age < 30
            Then Like = True
        Else….
    Else …
Else…
```

Apply the model to all items not seen by the user. Recommend the items with the highest score (prediction confidence)

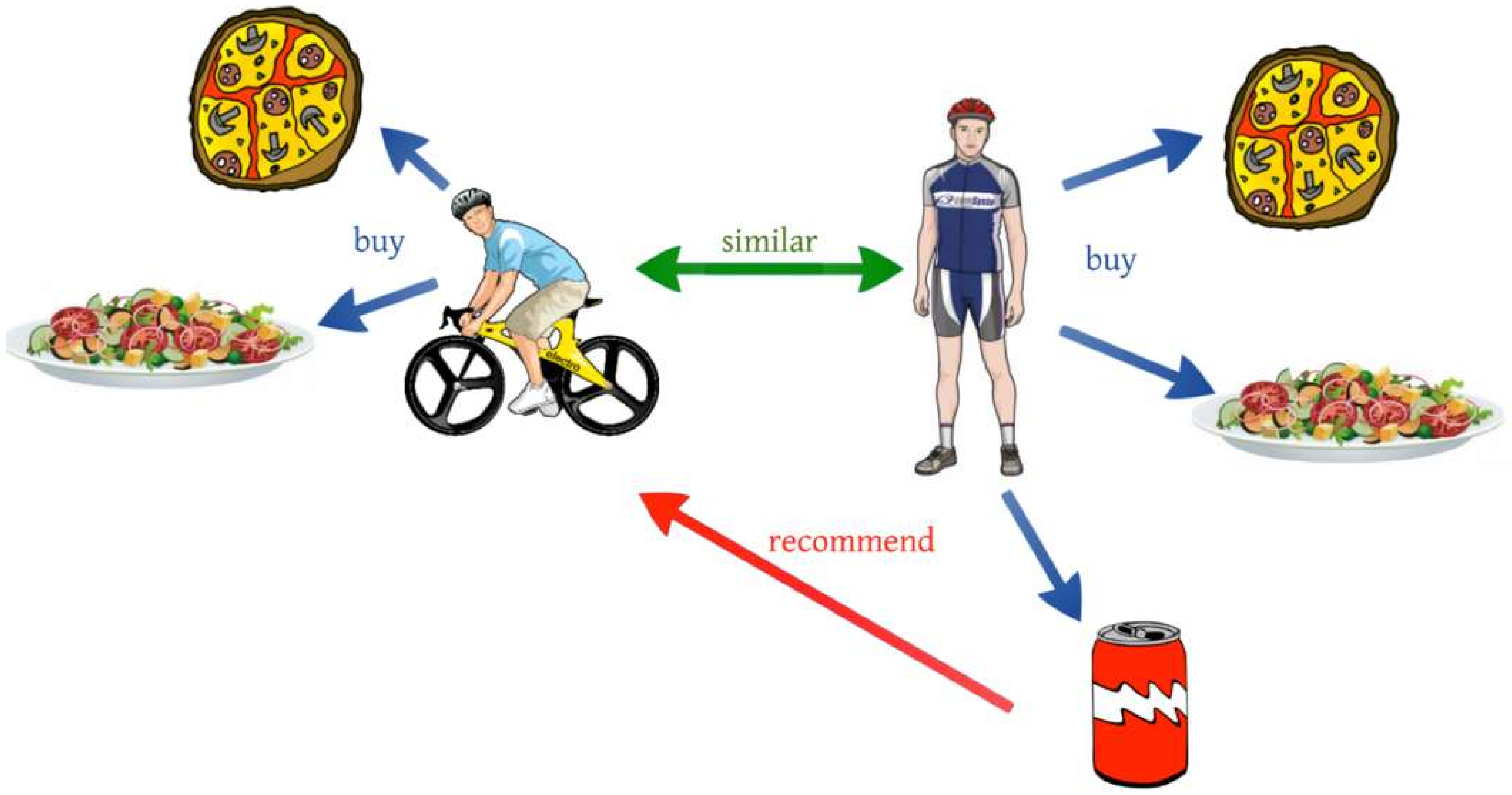
Can build a separate model for each user or item

Content-Based Example - Pandora

- Pandora is an online streaming music company,
- A team of musicologists annotates songs based on genre, rhythm, and progression. This data is transformed into a vector for comparing song similarity.
- Most songs are based on a feature vector of approximately 450 features, which were derived in a very long and arduous process
- Once this was done binary classification methods using more traditional machine learning techniques can be used to
  - Output a probability for a certain user to like a specific song based on a training set of their song listening history.
  - Recommend the songs with the greatest probability of being liked.
- This approach helps to promote the presentation of long tail music from unknown artists that might be a good fit for a particular user.

Collaborative Filtering (CF)

Match the target user to similar users and recommend what they like



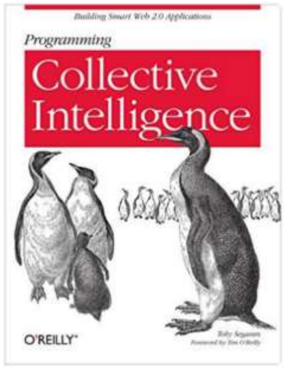
User-Based Collaborative Filtering

- We compute the similarity between users using the ***user-ratings matrix,*** a matrix of the user likes or interests in the various products/items on offer
- Ratings can be explicit (volunteered) or implicit (inferred from page-views, purchases..)
- Can use ***Cosine*** similarity, ***Euclidean*** similarity and ***Pearson*** similarity

User *	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	The Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby		4.5		4	1	

Ratings are numeric,
E.g. 1->5:
1 ~ do not like
5 ~ like very much

- ***Sparsity*** is usually an issue ~ lots of missing values!
  E.g. what is the similarity between Rose and Toby ?



* *Example taken from "Programming Competitive Intelligence*", O'Reilly

Measuring User Similarity

User	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby		4.5		4	1	

Options for handling missing values:

1. Only consider items where both users have given a rating ('pairwise complete')

$$\text{CosineSim(Toby, Rose)} = \frac{3.5*4.5 + 3.5*4 + 2.5*1}{\text{sqrt}(3.5^2+3.5^2+2.5^2)*\text{sqrt}(4.5^2+4^2+1^2)} = 0.95$$

2. Assume missing values are zeros

$$\text{CosineSim(Toby, Rose)} = \frac{2.5*0 + 3.5*4.5 + 3*0 + 3.5*4 + 2.5*1 + 3*0}{\text{sqrt}(2.5^2+3.5^2+3^2+3.5^2+2.5^2+3^2)*\text{sqrt}(0^2+4.5^2+0^2+4^2+1^2+0^2)} = 0.71$$

Predicting a Users Rating for an Item

Once we have the similarities between the target user and the other users, we predict the target user's rating for each of their unseen (unrated) items. To do this we average the ratings given by the other users on an unseen item, weighted by their similarity to the target (i.e. compute the weighted average rating)

$$\text{Predicted rating } (\mathbf{u},\mathrm{i}) = \frac{\Sigma_{v \in \boldsymbol{V}} \; \mathrm{sim}(\mathbf{u},\mathbf{v}) * r_{\mathbf{v},\mathrm{i}}}{\Sigma_{v \in \boldsymbol{V}} \; |\, \mathrm{sim}(\mathbf{u},\mathbf{v}) \,|}$$

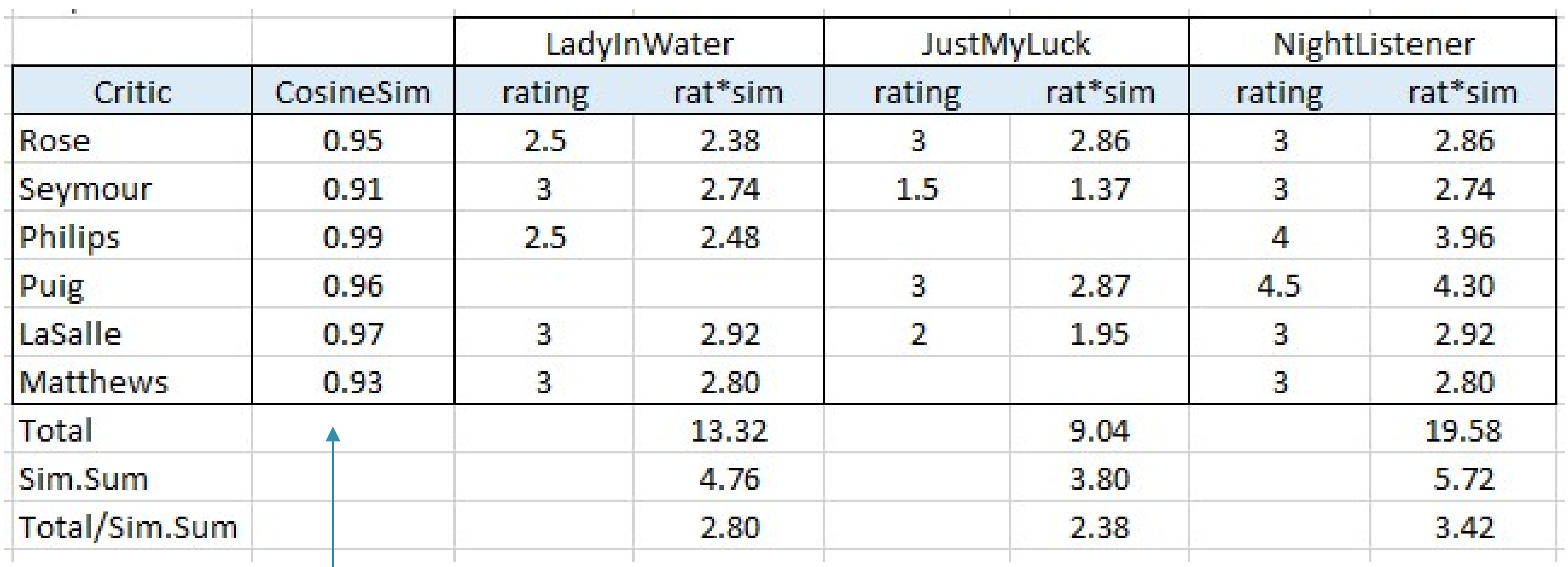
V is the set of other users

User	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	The Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby	?	**4.5**	**?**	**4**	**1**	?

target

To recommend a movie to Toby, we predict the likely rating for his unseen movies by computing the weighted average of the other users ratings on these movies. Then recommend the movie with the highest predicted rating

Computing the Weighted Average Rating



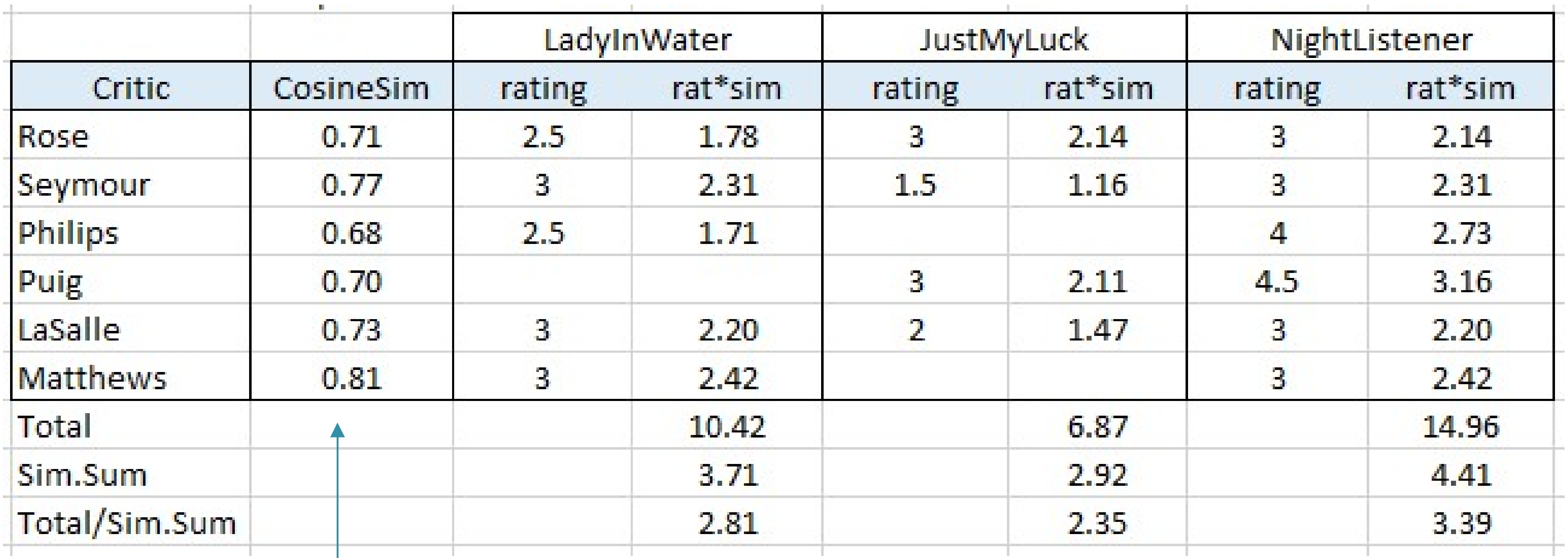
		LadyInWater		JustMyLuck		NightListener	
Critic	CosineSim	rating	rat*sim	rating	rat*sim	rating	rat*sim
Rose	0.95	2.5	2.38	3	2.86	3	2.86
Seymour	0.91	3	2.74	1.5	1.37	3	2.74
Philips	0.99	2.5	2.48			4	3.96
Puig	0.96			3	2.87	4.5	4.30
LaSalle	0.97	3	2.92	2	1.95	3	2.92
Matthews	0.93	3	2.80			3	2.80
Total			13.32		9.04		19.58
Sim.Sum			4.76		3.80		5.72
Total/Sim.Sum			2.80		2.38		3.42

The similarity of the other users to Toby (calculated using Cosine Similarity & 'pairwise complete')

The weighted average of the other users ratings for each unseen movie.

The best movie to recommend to Toby is hence 'The Night Listener'

Ignoring Pairwise Complete*



		LadyInWater		JustMyLuck		NightListener	
Critic	CosineSim	rating	rat*sim	rating	rat*sim	rating	rat*sim
Rose	0.71	2.5	1.78	3	2.14	3	2.14
Seymour	0.77	3	2.31	1.5	1.16	3	2.31
Philips	0.68	2.5	1.71			4	2.73
Puig	0.70			3	2.11	4.5	3.16
LaSalle	0.73	3	2.20	2	1.47	3	2.20
Matthews	0.81	3	2.42			3	2.42
Total			10.42		6.87		14.96
Sim.Sum			3.71		2.92		4.41
Total/Sim.Sum			2.81		2.35		3.39

The similarity of the other users to Toby (calculated using Cosine Similarity)

The weighted average of the other users ratings for each unseen movie.

The best movie to recommend to Toby is hence 'The Night Listener'

*many/most CF libraries ignore pairwise complete

User Similarity using Pearson Correlation

- Users may use different ratings to quantify the same level of appreciation for an item. This can create bias. E.g. one user may rate a movie as 5 and the other 4, yet they both like it equally well
- **Pearson Correlation Coefficient** is similar to cosine similarity but eliminates this bias by subtracting each users mean rating from their individual ratings. Value ranges from -1 to +1

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

$r_{u,i}$ = rating of user u on item i
$r_{v,i}$ = rating of user v on item i

$\overline{r}_u$, $\overline{r}_v$ = mean ratings of user's u, v

C is the set of all co-rated items

Other Normalisation Methods

Z-score normalization

- Divide the user *mean-centered* rating by the user's rating standard deviation. Hence the normalised rating for user *u* on movie *i* is: $\dfrac{r_{ui} - \bar{r}_u}{\sigma_u}$

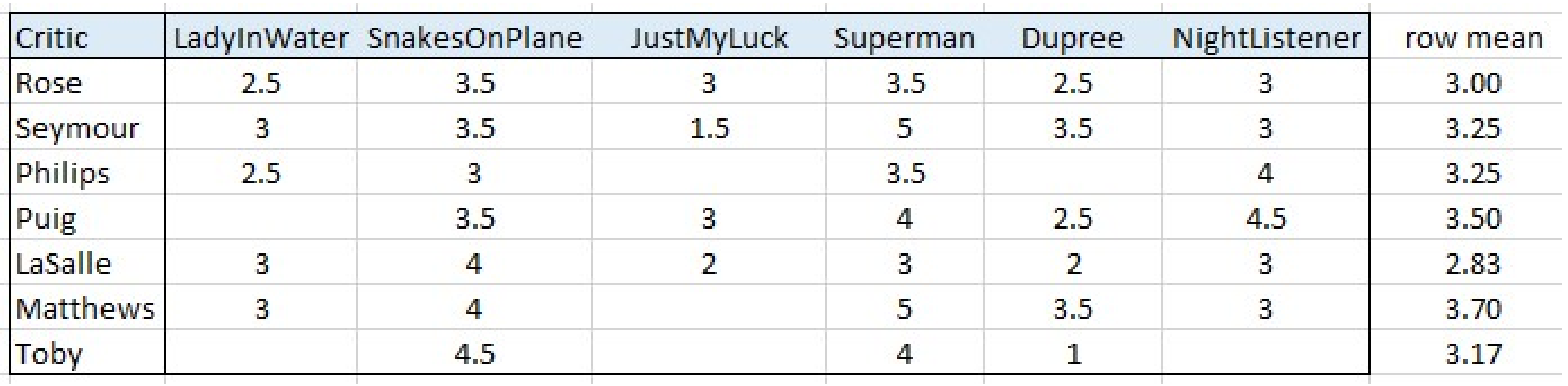
- The predicted rating for an unrated movie *i* for user *u* is then:

$$\hat{r}_{ui} = \bar{r}_u + \sigma_u \frac{\sum\limits_{v \in \mathcal{N}_i(u)} w_{uv} (r_{vi} - \bar{r}_v)/\sigma_v}{\sum\limits_{v \in \mathcal{N}_i(u)} |w_{uv}|}$$

- Z-score normalisation is useful if the rating scale has a ***wide range of discrete values or is continuous***.
- But, because the ratings are divided and multiplied by possibly very different standard deviation values, *Z*-score can be more sensitive than mean-centering and can often predict ratings that are outside the rating scale.

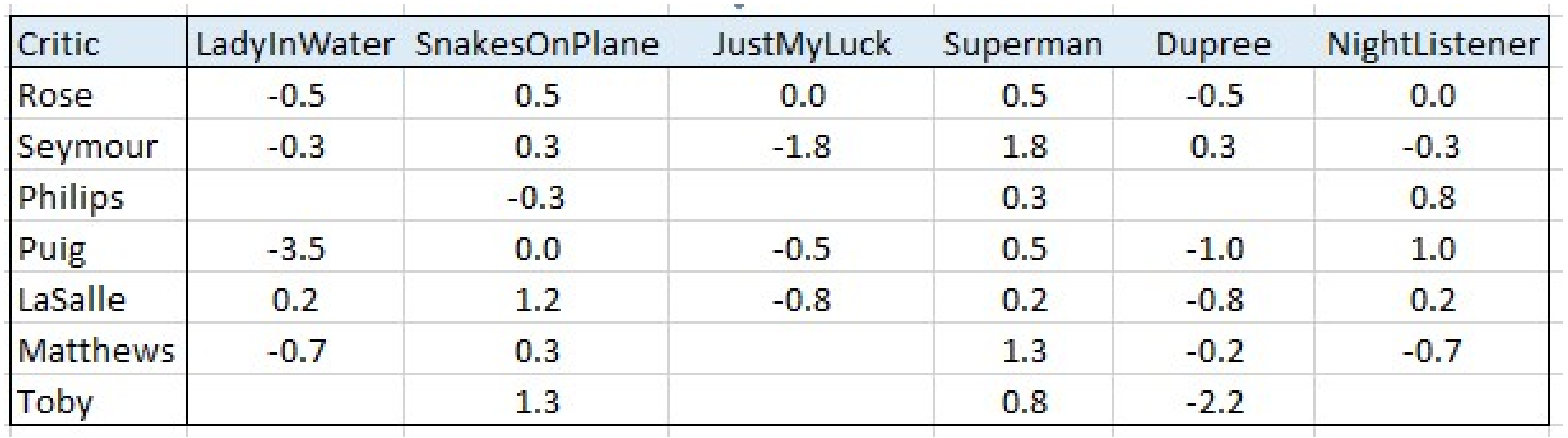
Pre-Normalisation

- We can choose to normalise all of the data in advance…



Critic	LadyInWater	SnakesOnPlane	JustMyLuck	Superman	Dupree	NightListener	row mean
Rose	2.5	3.5	3	3.5	2.5	3	3.00
Seymour	3	3.5	1.5	5	3.5	3	3.25
Philips	2.5	3		3.5		4	3.25
Puig		3.5	3	4	2.5	4.5	3.50
LaSalle	3	4	2	3	2	3	2.83
Matthews	3	4		5	3.5	3	3.70
Toby		4.5		4	1		3.17

Mean centering (subtract user's mean)



Critic	LadyInWater	SnakesOnPlane	JustMyLuck	Superman	Dupree	NightListener
Rose	-0.5	0.5	0.0	0.5	-0.5	0.0
Seymour	-0.3	0.3	-1.8	1.8	0.3	-0.3
Philips		-0.3		0.3		0.8
Puig	-3.5	0.0	-0.5	0.5	-1.0	1.0
LaSalle	0.2	1.2	-0.8	0.2	-0.8	0.2
Matthews	-0.7	0.3		1.3	-0.2	-0.7
Toby		1.3		0.8	-2.2	

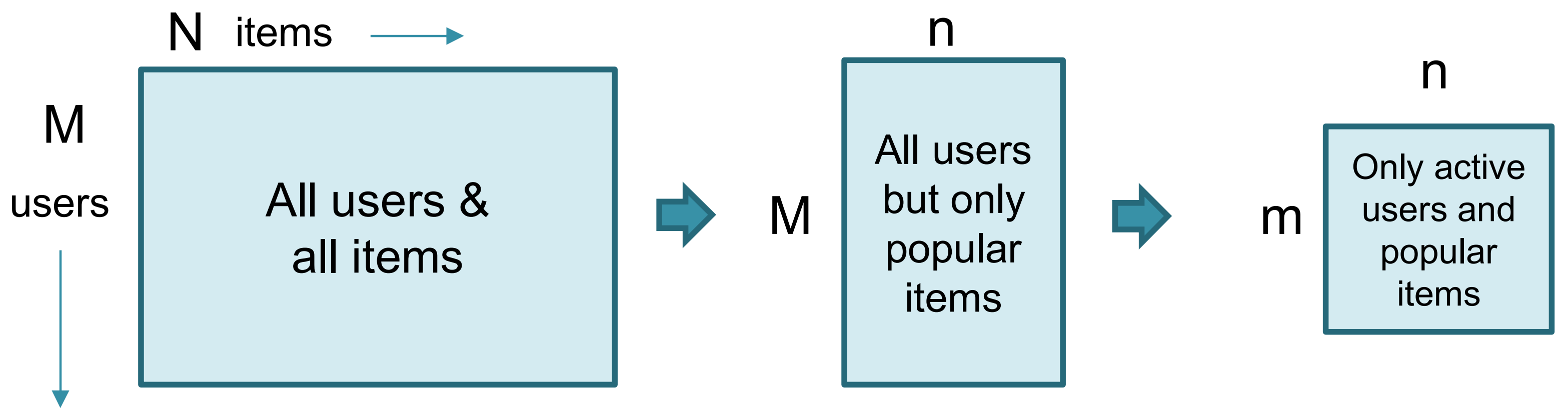
…..then proceed exactly as before

User-Based CF: Scalability Issues

- User-Based Collaborative Filtering does not scale well
  - User likes/interests may change often hence similarities between users are best computed at the time of the recommendation – this becomes computationally prohibitive for large numbers of users
- Computational load
  - Worse case ~ O(M*N) where M = #users, N = #items
  - In practice ~ O(M + N) since most users have rated/purchased few items
- Possible solutions
  - Dimensionality / Data Reduction
  - Item-Item Collaborative Filtering

Data Reduction for CF

- Massive datasets increase computational load
- For huge datasets, data sampling may be effective in certain situations
  - Reduce the number of users and/or items
  - Business directed sampling, e.g. select only active users, popular items
  - Reduce the number of items by grouping,
    e.g. tuna pizza, salami pizza, vegetarian pizza -> pizza


N items
M users
All users & all items
n
M
All users but only popular items
n
m
Only active users and popular items

Another way to Scale Collaborative Filtering

- Item-Based Collaborative filtering was developed by Amazon ~ 1998

Industry Report

Amazon.com Recommendations

Item-to-Item Collaborative Filtering

Greg Linden, Brent Smith, and Jeremy York • *Amazon.com*

http://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf

Item-Based Collaborative Filtering

- Transpose the user-rating matrix, compute distances between items not users
- Faster than User-Based – there are usually far fewer items than users AND the similarities between items change much less frequently than between users hence all item-item distances can be pre-computed
- A well known application of this method is Amazon's recommendation engine

item	user1	user2	user3	user4	….	userN
movie1	2	5	4			
movie2		3			1	
…..						

Transposition of the user rating matrix

Example Item-Item Similarity Matrix

	item1	item2	item3	item4	item5	item6	…
item1	1	0.90	0.61	0.13	0.68	0.75	…
item2		1	0.58	0.78	0.12	0.29	…
item3			1	0.55	0.32	0.69	…
item4				1	0.11	0.98	…
item5					1	0.75	…
item6						1	…
….							…

The matrix is symmetrical. Similarity is measured from 0 to 1 or -1 to 1, diagonals hence take the value 1

Typically precomputed, the frequency of updating depends on type of item (e.g. fast or slow seller, speed of new releases) – weekly or monthly updates may suffice.

Item-Based Collaborative Filtering

- Predict user ***u***'s rating for an item ***i*** using:

$$\text{Predicted rating } (\mathbf{u},i) = \frac{\sum_{j \in J} \text{sim}(i,j) * r_{\mathbf{u},j}}{\sum_{j \in J} |\text{sim}(i,j)|}$$

j is the set of K similar items to ***i*** (among items that u has rated)

- Instead of using Pearson, compute the distance between items ***i*** and ***j*** using the *adjusted cosine similarity**:

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

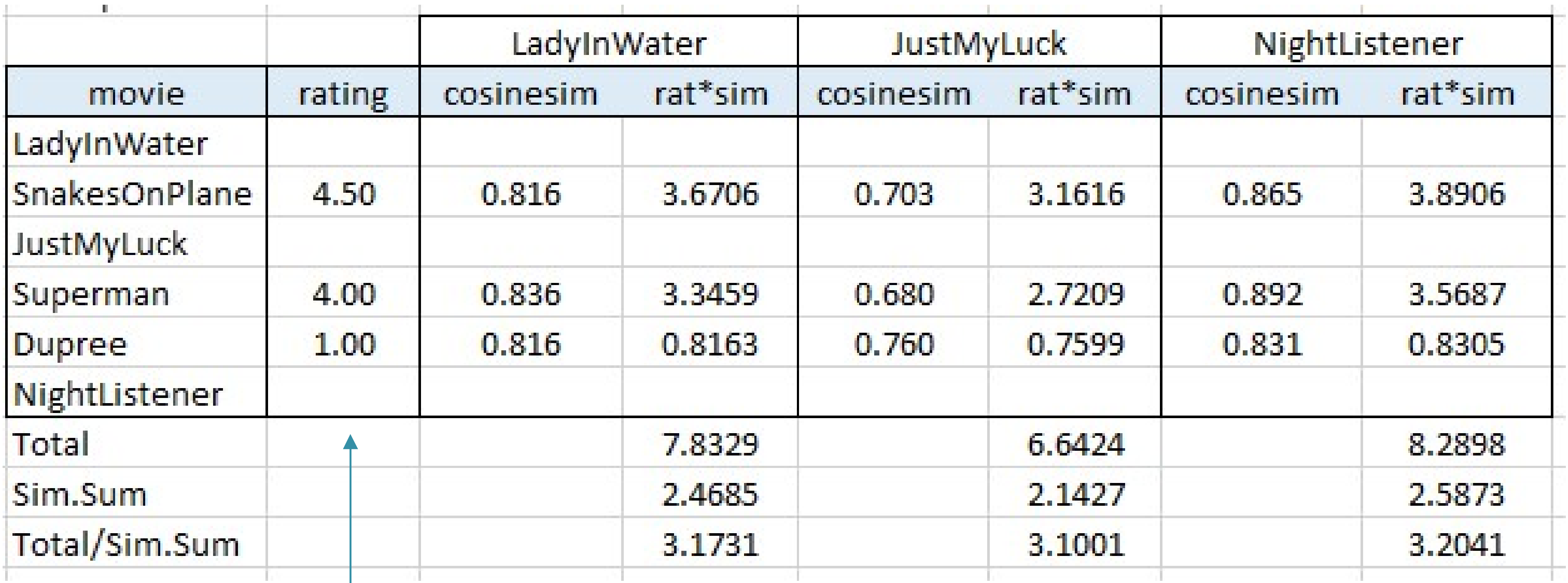
$r_{u,i}$ = rating of user u on item i
$r_{u,j}$ = rating of user u on item j

$\overline{r}_u$ = mean rating of user u

U = set of all users

** Other similarity measures can be used (e.g. Cosine, Euclidean Similarity)*

Item-Based Recommendations for Toby



		LadyInWater		JustMyLuck		NightListener	
movie	rating	cosinesim	rat*sim	cosinesim	rat*sim	cosinesim	rat*sim
LadyInWater							
SnakesOnPlane	4.50	0.816	3.6706	0.703	3.1616	0.865	3.8906
JustMyLuck							
Superman	4.00	0.836	3.3459	0.680	2.7209	0.892	3.5687
Dupree	1.00	0.816	0.8163	0.760	0.7599	0.831	0.8305
NightListener							
Total			7.8329		6.6424		8.2898
Sim.Sum			2.4685		2.1427		2.5873
Total/Sim.Sum			3.1731		3.1001		3.2041

Toby's ratings for the movies he has seen

Toby's estimated rating for ***Night Listener*** is obtained by computing his average rating for all of his seen movies – weighted by their distance to ***Night Listener***

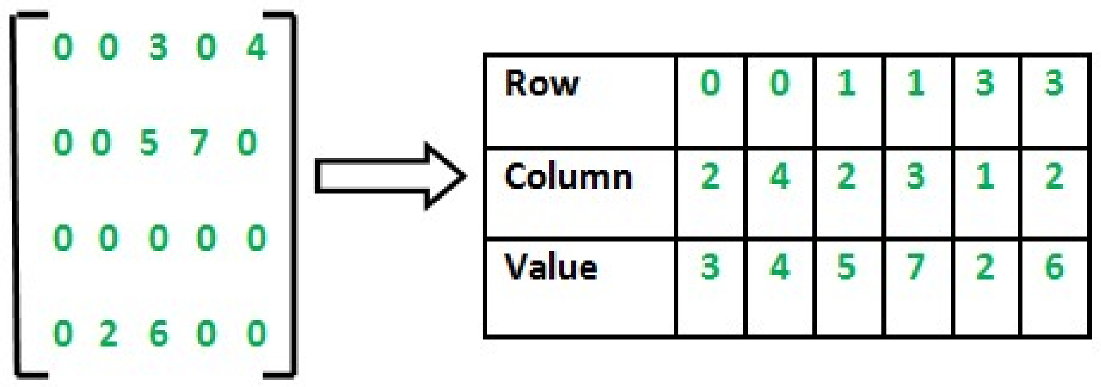
The best movie to recommend to Toby is hence 'The Night Listener'

Which is Best: User-Based or Item-Based?

- In general…
- If #users >> #items (e.g. Amazon) then Item-Item may be better
- If #users << #items (e.g. recommending web content) then User-User may be better
- In workshop2 we:
  - Compare User-based versus Item-based CF
  - Compare different similarity measurements
  - Compare normalised versus non-normalised data
  - Explore handling large datasets using data reduction

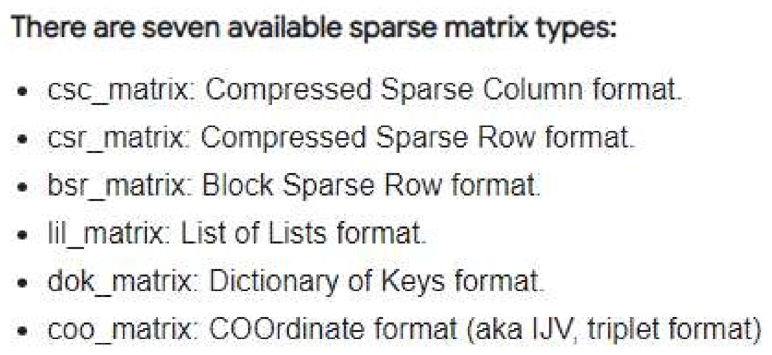
Efficient Storage: Sparse vs Dense Matrices

- Ratings data is often very sparse, storing as a dense matrix can be impractical
- E.g. if 5M users, 500K items and everyone rates ~10 items (hence ~ 50M ratings)
  - Dense ratings matrix = 5M * 0.5M = 2,500,000M cells (~ 10TB if stored as float)
- Sparse matrix representation stores only the actual values (plus row & col indexes)*
  - Sparse ratings matrix ~ (approx.) 3 * 50M = 150M values



$$\begin{bmatrix} 0 & 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 7 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 6 & 0 & 0 \end{bmatrix} \Rightarrow$$

Row	0	0	1	1	3	3
Column	2	4	2	3	1	2
Value	3	4	5	7	2	6



There are seven available sparse matrix types:

- csc_matrix: Compressed Sparse Column format.
- csr_matrix: Compressed Sparse Row format.
- bsr_matrix: Block Sparse Row format.
- lil_matrix: List of Lists format.
- dok_matrix: Dictionary of Keys format.
- coo_matrix: COOrdinate format (aka IJV, triplet format)

How big would the item-item similarity matrix be, and would a sparse matrix representation be useful for storing it?

Evaluating Recommender Systems

- How can we test a Recommender System before going live?
- How accurate does a Recommendation System Need to be?
- How can we estimate / measure the success of a Recommender System?
- What makes one Recommender System better than another?

Testing Rating Predictions

- Split the data into training & test sets by randomly assigning individual ratings to the test set, e.g. assign 30% to the test set.
- Predict the ratings in the test set and compare against the known values to get the ***Mean Absolute Error*** (MAE) averaged over all test predictions

All data

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1		3	4.5		2	4		5		1
u2	4	3.5		2	3		5		4.5	
u3		4	3		2	2.5	5		4	4

▢ =Test set

Train set

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1		3			2	4				1
u2				2	3				4.5	
u3		4	3			2.5	5			4

$$\textbf{MAE} = \frac{\underline{\Sigma \text{ abs(actual rating – prediction)}}}{\text{\# tests}}$$

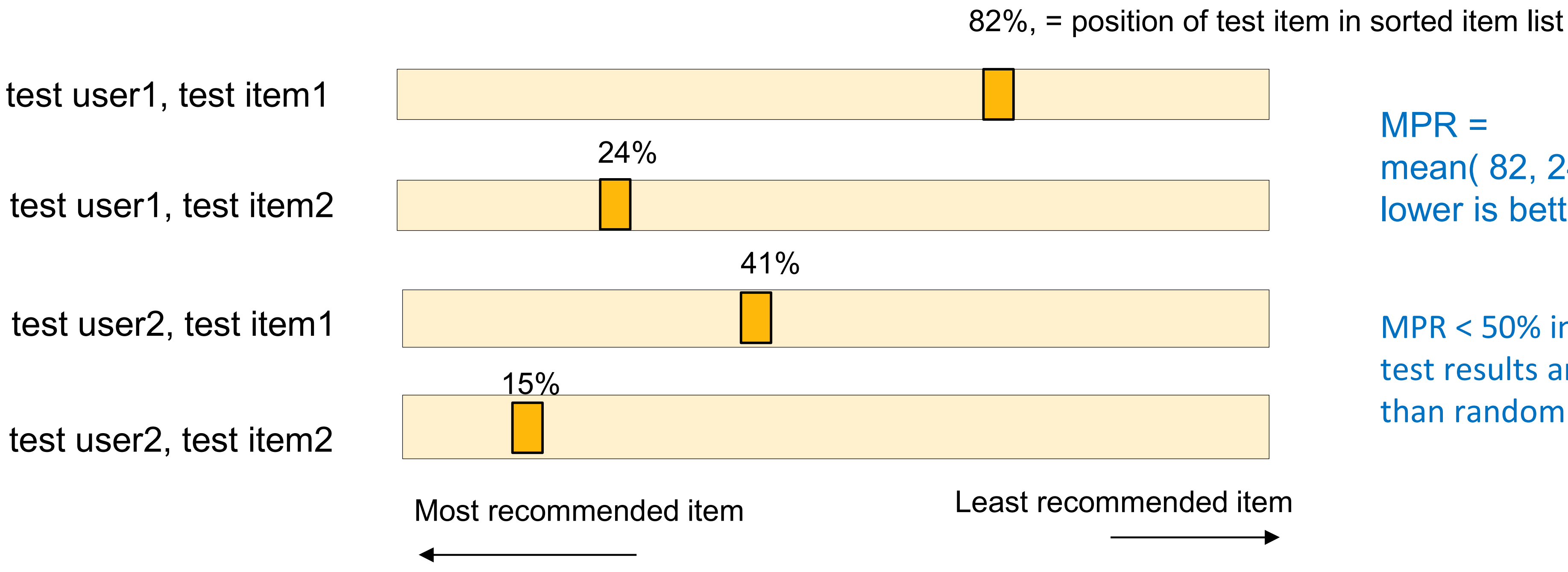
Test set

	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
u1			4.5					5		
u2	4	3.5					5			
u3					2				4	

Mean Percentage Ranking (MPR)

- What does a MAE of (say) 0.78 mean in practice? Are the recommendations good? What if there is no rating prediction, hence no MAE? (e.g. content-based filtering)
- MPR offers a way to estimate the effectiveness of the recommendations rather than the accuracy of the rating prediction

Consider only test items that are liked, e.g. have rating >=4 (you decide this threshold)
For each test item, predict its % position in the ranked list of all items *(ranked by their decreasing predicted rating, or any other scoring mechanism)*

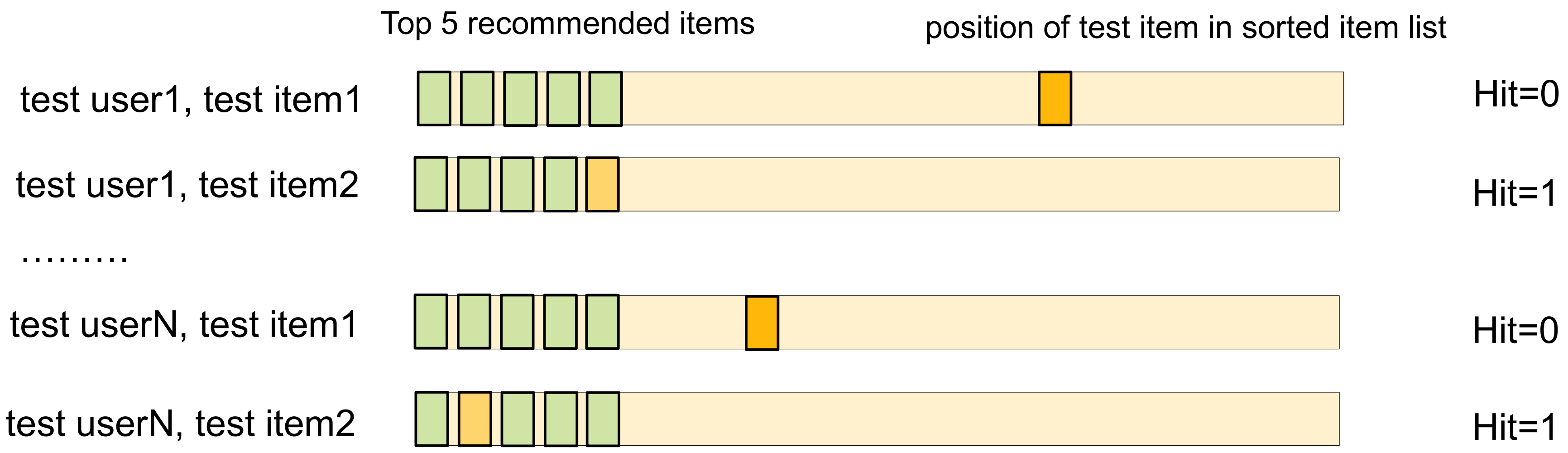


MPR =
mean(82, 24, 41,15)
lower is better!

MPR < 50% indicates test results are better than random guessing

Estimating Lift over Random

- Consider only test items that are liked
- For each test item, select the top-K items in the sorted recommendation list
- Count how many times a test item appeared in the top-K (call this the #hits)
- Compare with how many would be expected if random recommendations are made
- E.g. if top-K = 5:



Hit% = (#hits) / (#recommendations made)

Lift = (hit% using model) / (hit% using random)

***For large item sets you may need large test sets (or large K) to get significant results!*

Workshop 2

- Experiment with user-based and item-based CF
- Use Movielens, Jester & Bookcrossing datasets
- Consider Explicit Ratings only