

Specialist Programme on Artificial Intelligence for IT & ITES Industry

Recommender Systems (part I)

Dr Barry Shepherd
barryshepherd@nus.edu.sg

Singapore e-Government Leadership Centre
National University of Singapore

© 2020 NUS. The contents contained in this document may not be reproduced in any form or by any means, without the written permission of ISS, NUS other than for the purpose for which it has been supplied.

Inspire *Lead* *Transform*

Agenda

- Scoping Recommender Systems
- Main types of Recommender System
 - Non-personalised recommendations
 - Content-Matching
 - Market Basket Analysis
 - Collaborative Filtering
 - Matrix Factorisation
 - Advanced Methods

What is a Recommender System?

Customers who bought this item also bought

 <p>Core Java Volume I-- Fundamentals (10th Edition) (Core Series) Cay S. Horstmann ★★★★★ 105 Paperback \$33.59 ✓prime</p>	 <p>Core Java for the Impatient Cay S. Horstmann ★★★★★ 24 Paperback \$30.65 ✓prime</p>	 <p>Java 8 in Action: Lambdas, Streams, and functional-style programming Raoul-Gabriel Urma ★★★★★ 80 Paperback \$36.72 ✓prime</p>	 <p>Effective Java (2nd Edition) Joshua Bloch ★★★★★ 233 Paperback \$41.79 ✓prime</p>	 <p>Java Concurrency in Practice Brian Goetz ★★★★★ 1 Paperback \$31.23 ✓prime</p>
---	---	--	---	--

A Common Definition

- A Recommender System aims to find and suggest items of likely interest based on the users' preferences
- Most modern Web apps have a recommender system
- Examples:
 - Netflix: TV shows and movies
 - Amazon: products
 - LinkedIn: jobs and colleagues
 - Last.fm: music artists and tracks
 - Facebook: friends



Scoping Recommender Systems

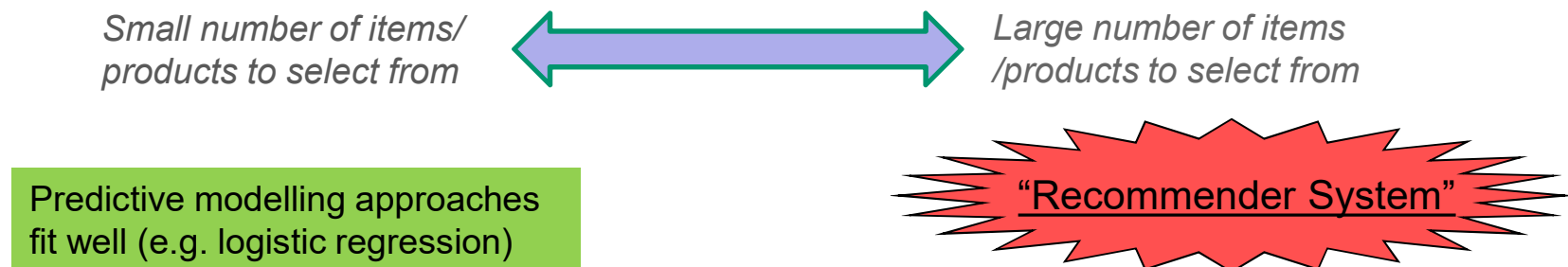
The phrase **Recommender System** is very broad – it *could be* used to describe a very wide range of applications, including common predictive modelling applications.

Just replace the word *predicting* with the word *recommending*!

For example:

- Recommending who to mail ads/promotions to (targeted marketing)
- Recommending the best drug or treatment for a patient
- Recommending the best business partner
- Recommending if a device/equipment needs preventative maintenance (T/F)
- Recommending which insurance claim to investigate further (possible fraud)

One distinguishing feature is the number of possible recommendations that can be made:



Scoping Recommender Systems

■ Recommending from a small set of products

- E.g. A bank wanting to promote its various insurance products (home, car, health, travel) to appropriate customers
- Typically done by customer segmentation and/or building a customised targeting model for each product
- Data rich: usually have customer account details + their transaction and activity records

■ Recommending from a large inventory of products

- E.g. On-line retailers like Amazon, Lazada, online newspapers and other content providers (their products are the various webpages, news articles etc)
- Data poor: often users are anonymous, e.g. have only info about their current browsing session

Predicting User Preferences

*A **recommender system** or a **recommendation system** is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item (Wikipedia)*

E.g. Amazon.com



12 million products

Amazon sells more than 12 million **products**.

In its quest to be all things to all people, **Amazon** has built an unbelievable catalog of more than 12 million **products**, books, media, wine, and services. If you expand this to **Amazon** Marketplace sellers, as well, the number is closer to more than 350 million **products**.

<http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/>

Each month **more than 197 million people** around the world get on their devices and visit Amazon.com.

That's more than the entire population of Russia.

<https://www.bigcommerce.com/blog/amazon-statistics/>

*"Judging by Amazon's success, the recommendation system works. The company reported a 29% sales increase to \$12.83 billion during its second fiscal quarter, up from \$9.9 billion during the same time last year. A lot of that growth arguably has to do with **the way Amazon has integrated recommendations** into nearly every part of the purchasing process..."*

E.g. Netflix

How important is Netflix's Recommender System?

80% of stream time is achieved through Netflix's recommender system, which is a highly impressive number. Moreover, Netflix believes in creating a user experience that will seek to improve retention rate, which in turn translates to savings on customer acquisition (estimated \$1B per year as of 2016).

<https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48>

Launched in 2006 with a \$1 million prize



- *Data* = 100 million movie ratings from 500,000 anonymous users on more than 17,000 movies.
- *Ratings* were on a scale 1 to 5
- *Data format* = (user, movie, rating, time)
- *Goal* = predict user ratings on a test set of 3 million
- *Winner* = first team that can beat their existing recommendation system performance by 10% (as measure by RMSE, root mean square error)

Congratulations!

The Netflix Prize sought to substantially improve the accuracy of predictions about how much someone is going to enjoy a movie based on their movie preferences.

On September 21, 2009 we awarded the \$1M Grand Prize to team "BellKor's Pragmatic Chaos". Read about [their algorithm](#), checkout team scores on the [Leaderboard](#), and join the discussions on the [Forum](#).

Data Sources For Recommender Systems

For a RS for a typical e-retailer, e-newspaper, media-streaming website etc:

- **Items** ~ the objects to be recommended and their properties
- **Users** ~ their demographics, purchases, account details, volunteered preferences
 - Often users are anonymous, via cookies only - hence user data is often limited / not available
- **Ratings** ~ interactions between the user/visitor and the website
 - **Explicit Ratings** – the user provides an opinion on an item using a rating scale, e.g. 1-5 (numerical) or “agree”, “neutral”, “disagree” etc. (ordinal)
 - **Implicit ratings** – derived from user actions (e.g. views, likes, buys)
 - **Text comments** - made by the user about the products
 - **Tags** – user assigns a tag to items, e.g. movie is “too long”, “bad acting”, etc.
- **Context**
 - Mostly used to filter recommendations
 - E.g. Is user a beginner or advanced camera user?
 - E.g. Is user looking for restaurants near their current location?



Explicit ratings are high value

Ratings Sparsity

People often do not bother to rate products!

■ In Netflix: **98.8 %** of the ratings are unknown

■ In Movielens: **95.7 %** of the ratings are unknown



Handling sparsity is a major issue when building Recommender Systems!

We usually don't recommend just one thing!

Recommended for you, Thomas

The image displays a grid of recommended items for a user named Thomas. The grid is organized into two rows and four columns. Each column represents a different category of items. The first row includes Literature & Fiction (books like 'Think and Grow Rich' and 'Calvino'), Exercise & Fitness Equipment (a yellow flower-shaped object, a black cone, a clock, and markers), Health, Fitness & Dieting Books (books like 'SUPERHUMAN FOCUS', 'FANTASTIC VOYAGE', and 'TURBO NINJA SUCCESS'), and Tableware (a silver thermos, a white bowl, and a box of instant noodles). The second row includes Prime Video (a movie poster for 'CATASTROPHE'), Coffee, Tea & Espresso (bags of coffee and tea), Biographies & Memoirs (books like 'SAM WATSON' and 'DESPOT OF POWER'), and Engineering Books (books like 'MONETIZING INNOVATION' and 'THE INNOVATION HUB').

Literature & Fiction
42 ITEMS

Exercise & Fitness Equipment
8 ITEMS

Health, Fitness & Dieting Books
37 ITEMS

Tableware
10 ITEMS

Prime Video – Unlimited Streaming for Prime Members
12 ITEMS

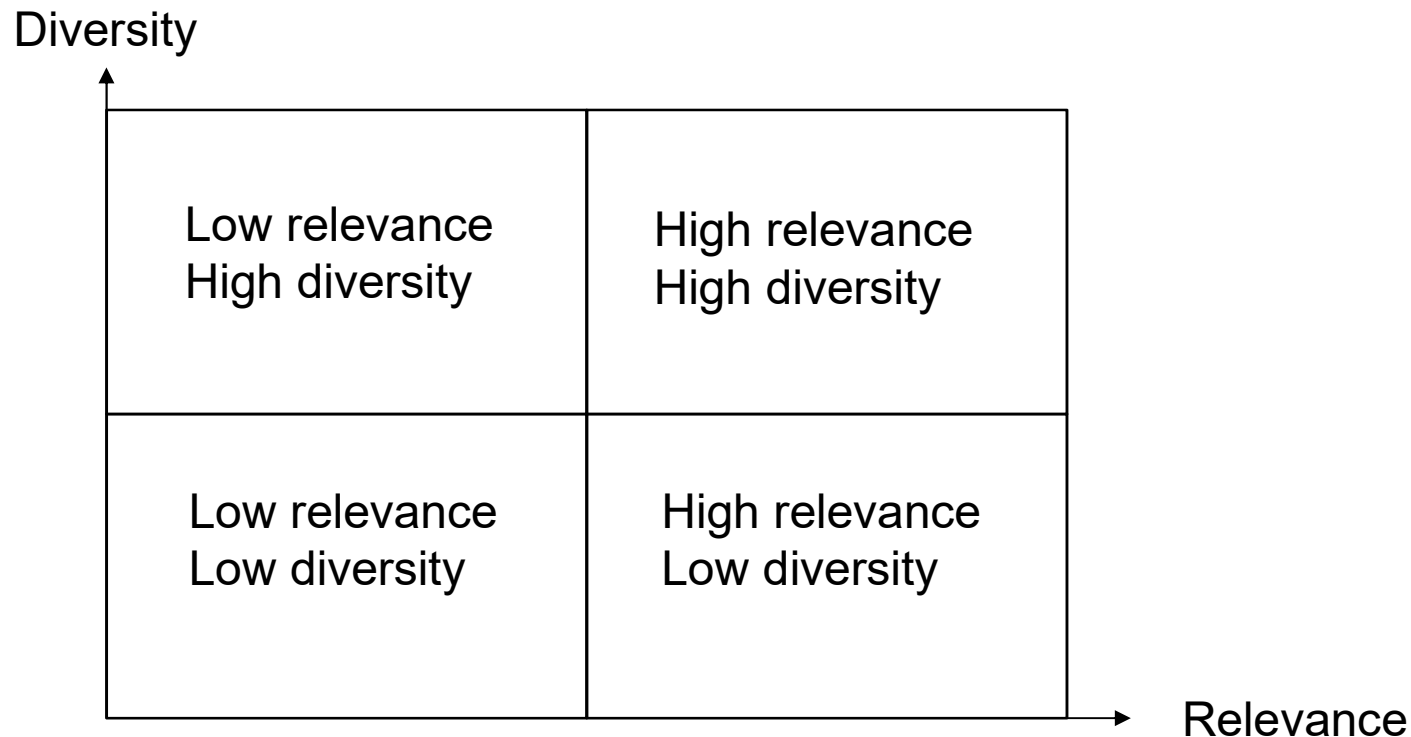
Coffee, Tea & Espresso
66 ITEMS

Biographies & Memoirs
17 ITEMS

Engineering Books
7 ITEMS

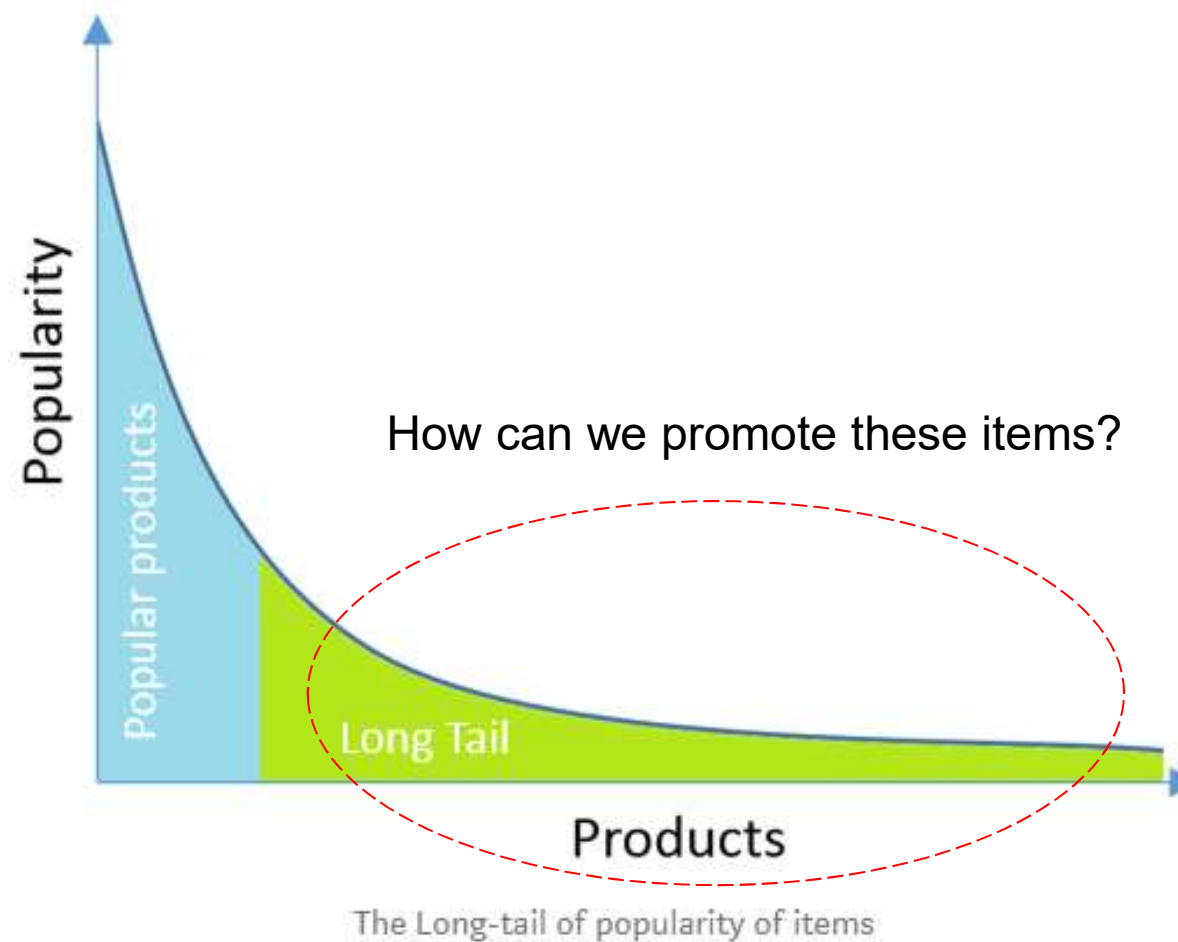
Diversity is important too!

- Unlike predictive modelling, accuracy or precision is not the only criteria for success



Where do you think your recommendations should be?

Recommender Systems & the Long Tail



Recommender Systems & the Long Tail

- Solving the long-tail problem was one of the early promises of Recommender Systems...



<http://www.wired.com/2004/10/tail>

Recommender System Approaches

- Simple – no personalised (best selling items)
- More of the same (content-based)
- Common combinations (market basket analysis)
- What do people similar to you like? (collaborative filtering)

Non-Personalised

- Best Selling Items
- Most viewed items
- Current Trending Items



Amazon.com recommends category top sellers for shoppers looking to try the new and latest products. 'Best-selling' adds a social proof element to the recommendation, that 'other people are doing it and so should you'. Best sellers from a specific product category help people find popular products and buy from new categories they may never have purchased from before, which opens up someone to a whole new range of up-sell and cross-sell opportunities.

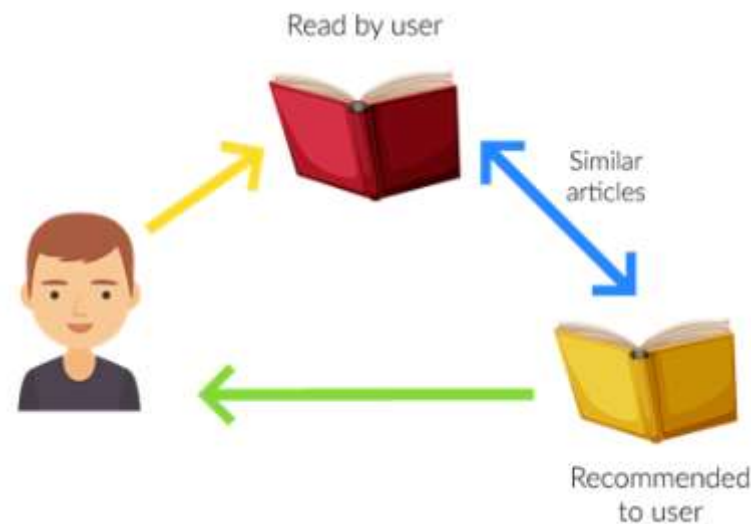
<http://rejoiner.com/resources/amazon-recommendations-secret-selling-online/>

Content-Based Filtering

- One of the earliest Recommender System approaches

Show you more of what you like!

- Has roots in information retrieval systems
- Good for content recommendations



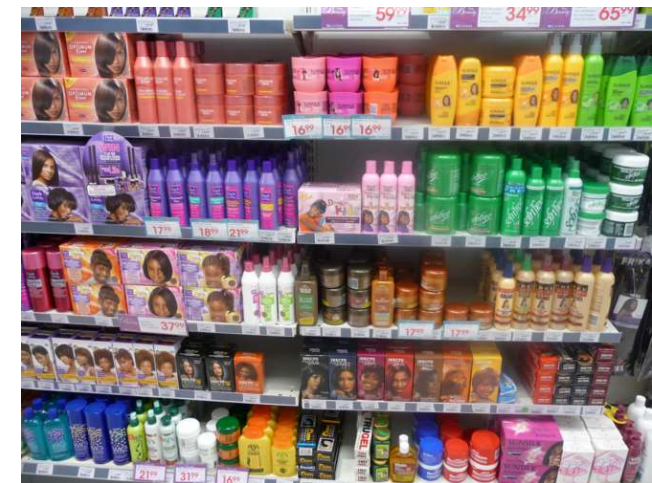
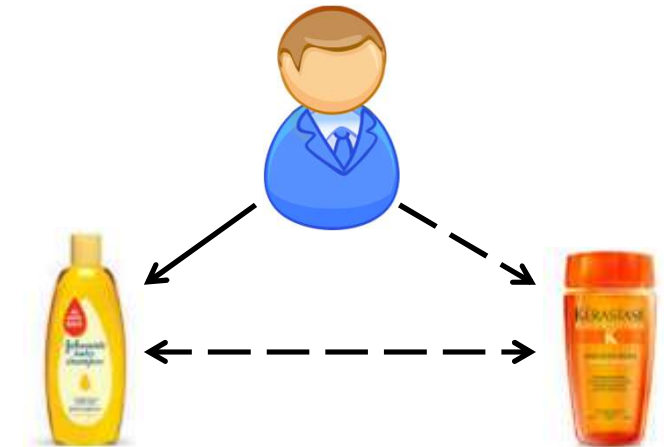
- Big disadvantage ~ get recommended more of the same, little diversity

Content-Based Recommenders

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

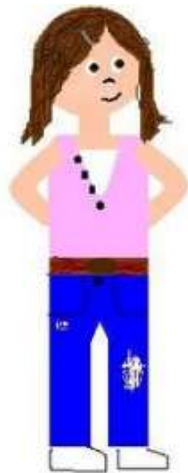
Main components are:

- **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. structured data provided by the manufacturer
 - **OR** extract using text mining - parse product descriptions (e.g. 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
 - Usually a relevance score is output for each product indicating the relevance (match) of the product to the user
 - The products are then ranked by relevance and the top N form the recommendations



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?

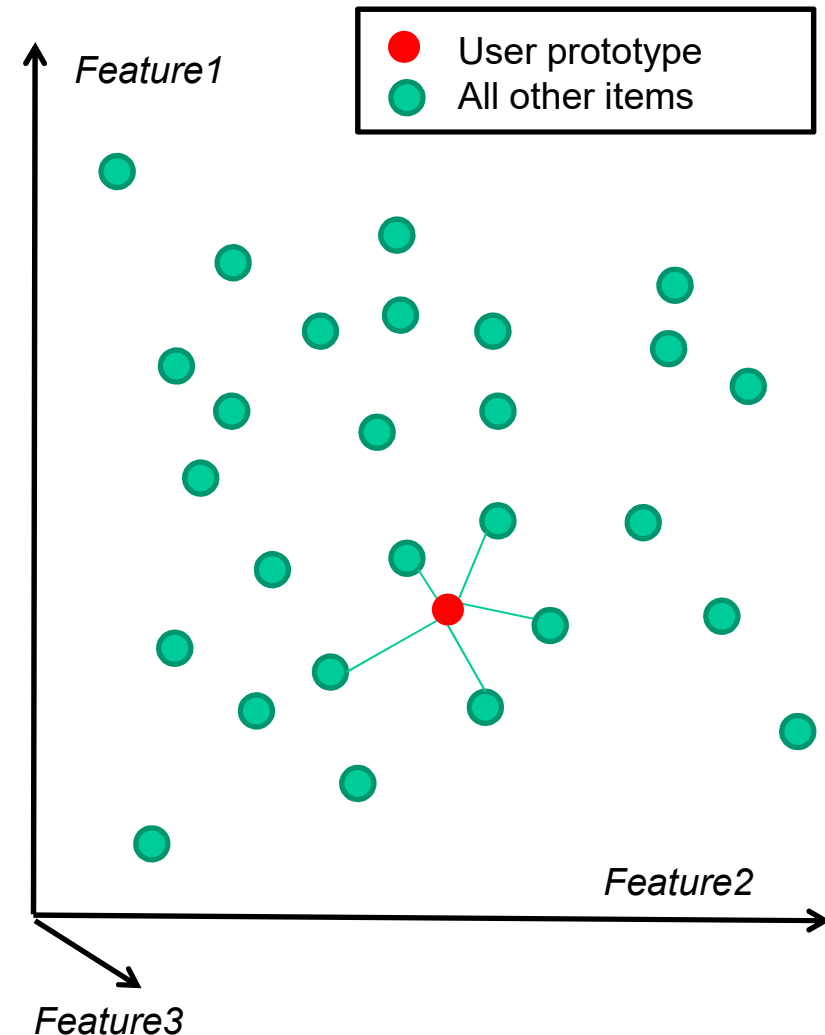
Method1: Neighbourhood approach

Method2: Predictive Modelling approach

Method1: Neighbourhood Approach

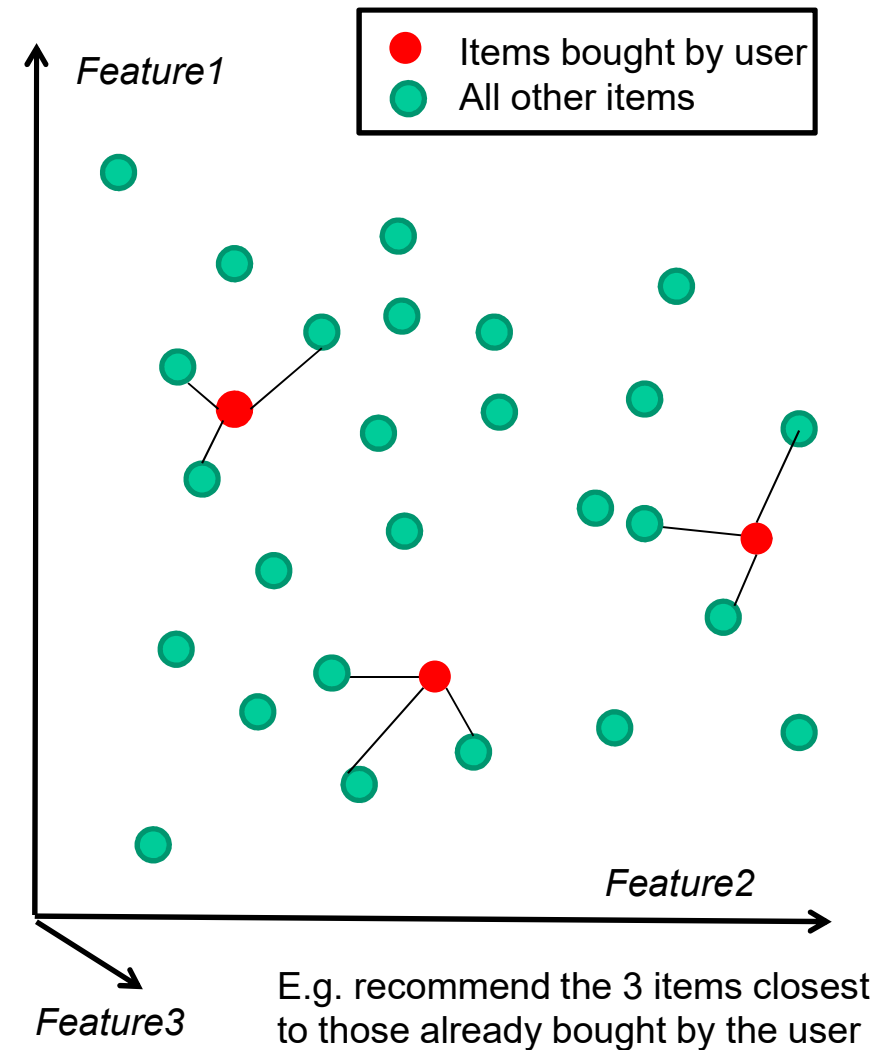
- Each item is represented by a set of features (a **feature vector**)
- The user is represented by the same feature set, often obtained by:
 - The user selects preferences for the various features using pull-down menus
 - Computing an “average” vector from items already bought/liked by the user. This 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 their distance

**Also known as Vector-Space approach*



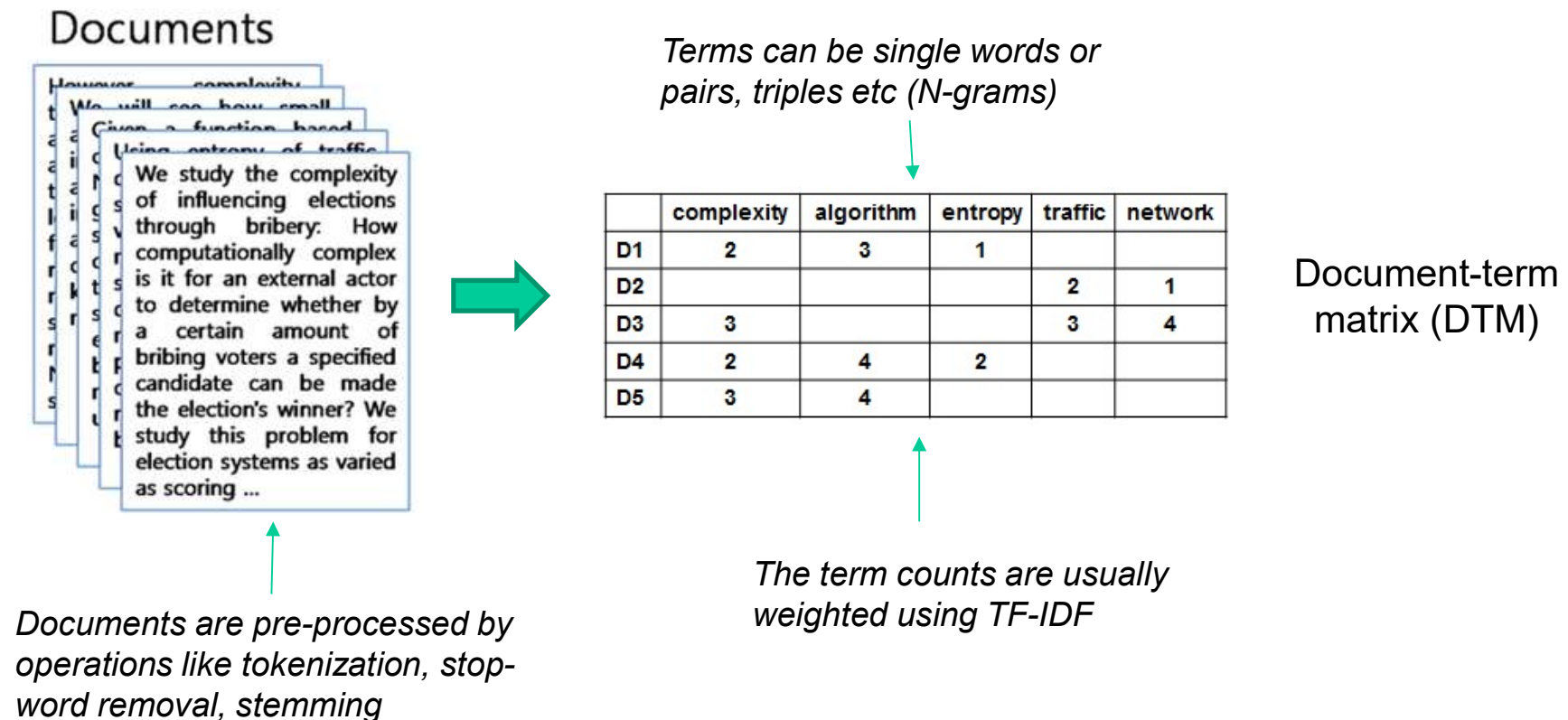
Method1: Neighbourhood Approach

- If the retailer is selling a diverse product range 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 bought or has viewed on an website (*implicit feedback*)
- Or we can recommend items that are close to the items that the user has “liked” or rated highly (*explicit feedback*)



Text-Based 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



TF-IDF Weighting

■ Term Frequency (TF)

- TF = number of occurrences of a term in a document
- More occurrences suggest more importance

■ Inverse Document Frequency (IDF)

- A measure of how much information a term provides
- When a term occurs in many documents it's considered unimportant
- $DF = \text{number documents that contain the term} / \text{number of documents}$
- $IDF = \text{inverse document frequency} = \log(1/DF)$

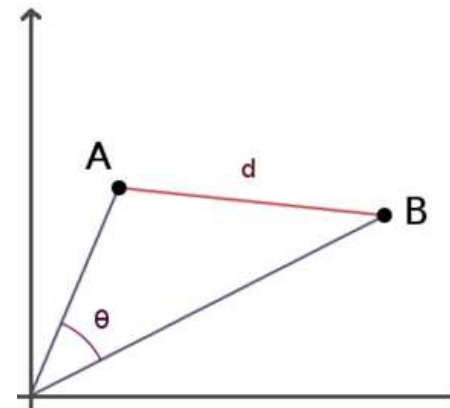
■ $TF\text{-}IDF = TF * IDF$

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:

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

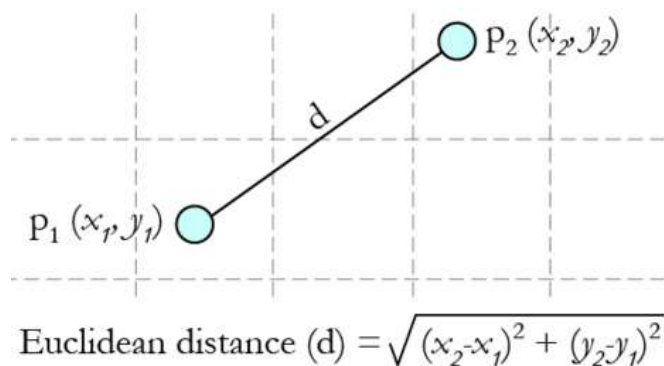
**Content-based recommender systems have their roots in Information Retrieval (IR). The goal of IR is to identify relevant documents in response to a user query*



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

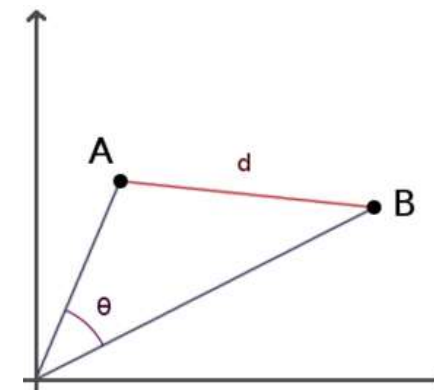
Why Cosine Similarity?

- For general classification problems, the distance metric used in k-NN systems is commonly the Euclidean distance



Can convert to “similarity” using (for example):
similarity = $1/(1 + \text{distance})$

- If the vectors are documents then Cosine Similarity is often better because it ignores the magnitude of the vectors (only considers angle)

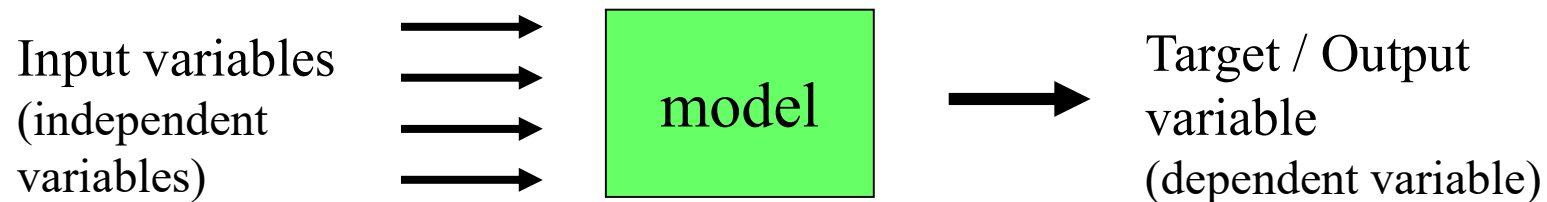


E.g. if one doc has word “disease” 10 times and another has it 100 times then using cosine they will still be similar but they will be different using Euclidean distance

E.g. Two docs with similar word profiles

	w1	w2	w3	w4	w5	w6
Item1	0	10	20	100	200	400
Item2	0	100	200	1000	2000	4000
				Eucl Dst	Eucl Sim	Cosine
				4129.2	0.0002	1.0000

Method2: Predictive Modelling Approach



- Event Prediction - will something happen?
 - E.g. will customer respond to a campaign? (Response Modelling)
 - E.g. will customer buy a product?
 - Can use: Logistic Regression, Decision Trees, Neural Networks,....
- Value Prediction - predicting the value of something
 - E.g. how much money will a person spend?
 - Can use: Linear or Polynomial regression, Neural Networks,...

Method2: Predictive Modelling Approach

The user rates a sample of the items (explicitly or implicitly) as like/dislike:

	Length	Studio	Director	Lead Actor1	Genre	Date	Likes
movie1	90	Disney	Spielberg	Tom Hanks	action	2009	T
movie2	110	MGM	Petersen	Brad Pitt	action	2004	F
movie3	120	Disney	Nolan	C. Bale	Sci-fi	2008	F
movie4							T
movie5							F

E.g. a decision tree model:

If we have sufficient labelled items then we can use supervised learning to build a predictive model for that user

```

If Length < 100
  If Genre == Action
    If Date < 2006
      Then Likes = True
    Else....
  Else ...
Else...
  
```

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

Requires one model per user or one model per 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.

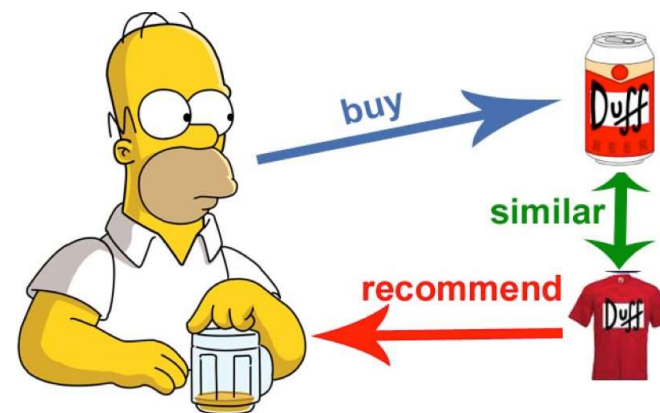
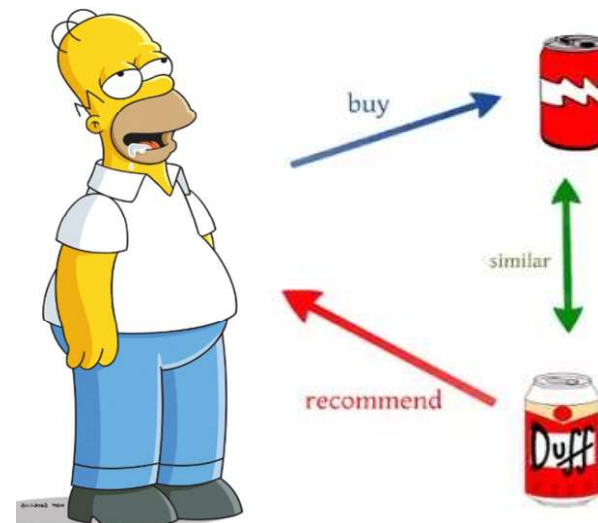
Pros & Cons of Content-Based Filtering

Pros

- Can recommend new items even if there are no ratings provided by users
- Can explain why recommendations are made by referencing the deciding item features
- Can recommend new items not yet rated by any users

Cons

- Hard to recommend items not similar to those already seen by the user
- Getting an adequate set of product descriptor features can be hard
- Can be hard to get the user preferences against the product features. Users don't generally have the patience to specify all of their preferences



Market Basket Analysis (MBA)

- Ignore the properties (features) of the items
- Look at what items get bought together?



- Simple approach ~ co-occurrence counting = count how often items occur together in the same basket

Recommendations from Purchase Baskets

Customers who bought this item also bought

 <p>Core Java Volume I-- Fundamentals (10th Edition) (Core Series) Cay S. Horstmann ★★★★★ 105 Paperback \$33.59 ✓prime</p>	 <p>Core Java for the Impatient Cay S. Horstmann ★★★★★ 24 Paperback \$30.65 ✓prime</p>	 <p>Java 8 in Action: Lambdas, Streams, and functional-style programming Raoul-Gabriel Urma ★★★★★ 80 Paperback \$36.72 ✓prime</p>	 <p>Effective Java (2nd Edition) Joshua Bloch ★★★★★ 233 Paperback \$41.79 ✓prime</p>	 <p>Java Concurrency in Practice Brian Goetz ★★★★★ 1 Paperback \$31.23 ✓prime</p>
---	---	--	---	--

Recommendations from View Baskets

People who viewed this item also viewed

 <p>PERSONALISE MUG WITH ANY NAME ANY NAME COSTA COFFEE £0.99</p>	 <p>Retro Personalised "Robert" coffee mug £1.00</p>	 <p>KEEP CALM MUG PERSONALISE YOUR KEEP CALM MUG ON BOTH SIDES KEEP CALM AND CARRY ON PERSONALISED... £3.29</p>
--	--	--

Co-occurrence Counting

E.g. pages visited on a news site:

session1: **home**, **news**, sport
 session2: finance, news
 session3: fashion, home
 session4: **news**, finance, **home**
 session5: sport, home, finance
 session6: fashion, **home**, **news**
 session7: **home**, finance, **news**, sport

A frequent item-set is {home, news}

Co-occurrence counts for pairs :

	home	sport	news	fashion	finance
home	6	3	4	2	3
sport		3	2	0	2
news			5	1	3
fashion				2	0
finance					4

Symmetrical matrix - Items in each basket have no temporal ordering, we only need consider the green

- Items that occur together (are in the same basket) are called an item-set.
- If an item-set has a large count (a frequent item-set) then there is a potential association between the items in the item set

Association Rule Learning

- Co-occurrence counting is not viable for large datasets and large item-sets
- Algorithms such as Apriori (Agrawal et al, '93) use heuristics to reduce the combinatorial size of the search space
 - If an item-set is frequent, then all of its subsets must also be frequent
 - The user specifies minimum rule support and rule confidence
- The found associations are expressed as rules, e.g.

session1: home, news sport
session2: finance, news
session3: fashion, home
session4: news, finance home
session5: sport, home, finance
session6: fashion home, news
session7: home, finance news sport

Possible Rules	Itemset count
{home} → {news},	4
{home, news} → {finance},	2
{news} → {home, sport},	2

Note1: rules indicate co-occurrence, not causality or a sequence over time

Issues with Association Mining

- Can often generate a huge number of rules, often trivial and with repetition:

If coffee and milk then sugar
If milk and sugar then coffee
If sugar and coffee then milk

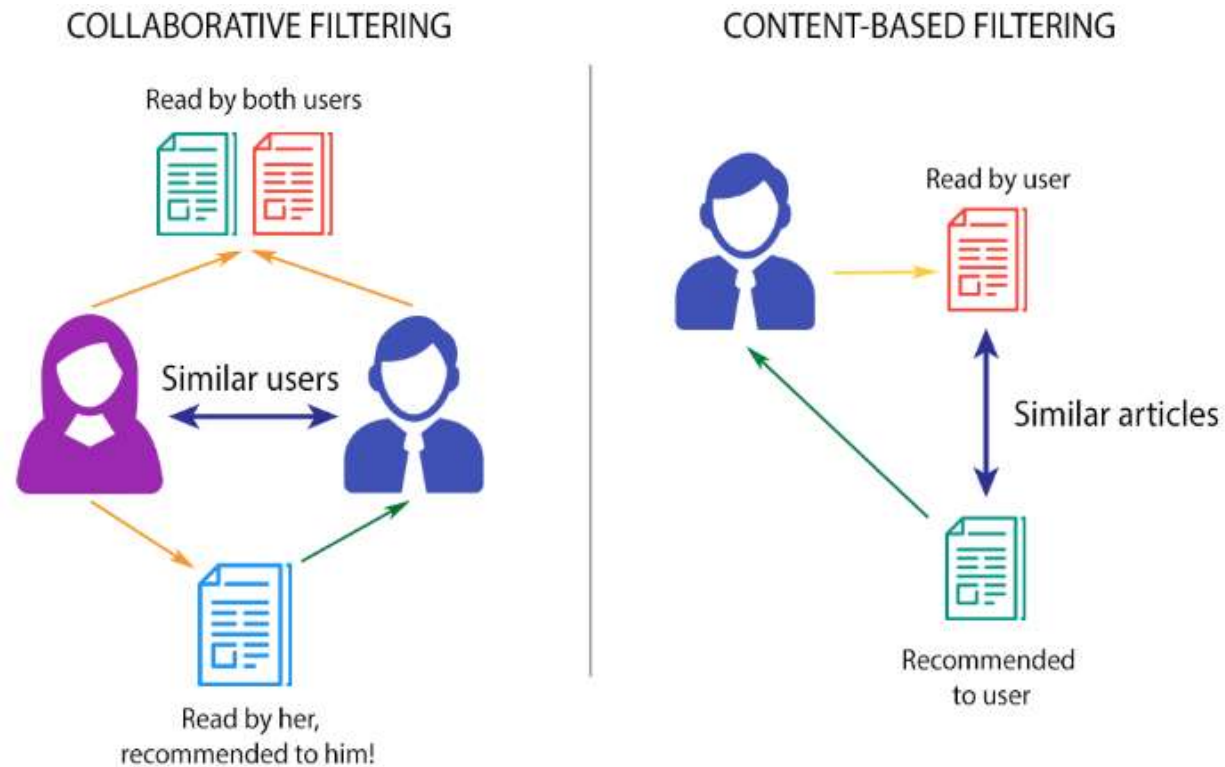
- We can filter the rules by “rule support” and “rule confidence” to select only those with high support and confidence
 - But this may still require analysts to make (subjective) decisions about which rules are valid and important
- Very granular product inventories may cause association overload, or hide associations
- Categorisation may help, e.g.

pineapple pizza, salami pizza, veggie pizza → pizza

240gm Haagen Daz, 420gm Haagen Daz → Haagen Daz

Collaborative Filtering

- No need to use the properties of the items
- Only look at how the users view, buy or rate the items
- Allows much greater diversity in recommendations



Measuring User Similarity

- How do we find similar users? What user data can we use?
- Most common data used is the user-ratings matrix, a matrix of the user likes or interests in the various products/items on offer



John 	5	1	3	5
Tom 	?	?	?	2
Alice 	4	?	3	?

Ratings can be explicit (directly given) or implicit (inferred from page-views, purchases etc.)

Finding Similar Users

- How to compute distances/similarities between entries in the ratings matrix?
- Sparsity (many missing values) is a problem
- E.g. what is the similarity between John and Tom ?

				
John 	5	1	3	5
Tom 	?	?	?	2
Alice 	4	?	3	?

Common Metrics:

- Euclidean Distance
- Cosine Similarity
- Pearson Correlation

Finding Similar Users

- 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

$$\text{Sim}(\mathbf{u}, \mathbf{v}) = \frac{\sum_{i \in C} (r_{u,i} - \bar{r}_u)(r_{v,i} - \bar{r}_v)}{\sqrt{\sum_{i \in C} (r_{u,i} - \bar{r}_u)^2} \sqrt{\sum_{i \in C} (r_{v,i} - \bar{r}_v)^2}}$$

$r_{u,i}$ = rating of user u on item i
 $r_{v,i}$ = rating of user v on item i
 \bar{r}_u, \bar{r}_v = mean ratings of user's u, v
 C is the set of all co-rated items

User	movie1	movie2	movie3	movie4	movie5	movie6	Mean?
target	1	2			3	2	2
1	2	5	4		5	1	3.4
2	2	3	5	4		2	3.2
3			5	3		4	4

```
> users
      m1 m2 m3 m4 m5 m6
target  1  2 NA NA  3  2
u1       2  5  4 NA  5  1
u2       2  3  5  4 NA  2
u3      NA NA  5  3 NA  4
```

```
> cor(t(users), use="pairwise.complete.obs")
      target      u1      u2      u3
target 1.0000000 0.5940885 0.5000000      NA
u1      0.5940885 1.0000000 0.6454972 1.0000000
u2      0.5000000 0.6454972 1.0000000 0.3273268
u3      NA 1.0000000 0.3273268 1.0000000
```

Making a Recommendation

- Once we have the similarities between the target user and the other users then we compute a recommendation based on the preferences of other users. One method is to sum the preferences of these other users weighted by their similarity to the target (weighted average)

$$\text{Predicted rating } (u,i) = \bar{r}_u + \frac{\sum_{v \in V} \text{sim}(u,v) * (r_{v,i} - \bar{r}_v)}{\sum_{v \in V} |\text{sim}(u,v)|} \quad \left\{ \begin{array}{l} V \text{ is the set of} \\ \text{other users} \end{array} \right.$$

User	similarity	m1	m2	m3	m4	m5	m6	Mean
target	1	-1	0	?	?	1	0	2
user1	0.59	-1.4	1.6	0.6		1.6	-2.4	3.4
user2	0.5	-1.2	-0.2	1.8	0.8		-1.2	3.2
user3	NA			1	-1		0	4
weighted average				$\frac{0.59*0.6 + 0.5*1.8}{0.59+0.5} = 1.15$	$\frac{0.5*0.8}{0.5} = 0.8$			
predicted rating				$2 + 1.15 = 3.15$	$2 + 0.8 = 2.8$			

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:

$$\frac{r_{ui} - \bar{r}_u}{\sigma_u}$$

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

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

- 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.

A Simple Example*

The raw ratings....

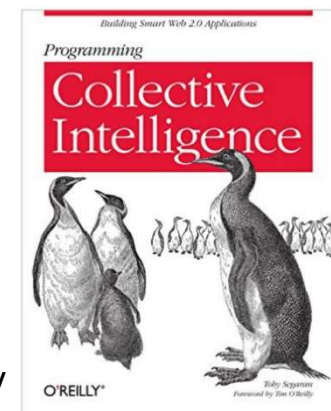
User	Lady In Water	Snakes On A Plane	Just My Luck	Superman	You Me And Dupree	The Night Listener
Rose	2.5	3.5	3	3.5	2.5	3
Seymour	3	3.5	1.5	5	3.5	3
Philips	2.5	3		3.5		4
Puig		3.5	3	4	2.5	4.5
LaSalle	3	4	2	3	2	3
Matthews	3	4		5	3.5	3
Toby		4.5		4	1	

The goal is to make movie recommendations for Toby.

Do this by predicting his likely rating for his unseen movies:

Lady in Water, Just My Luck, The Night Listener

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



(I) Calculate Toby's Similarity to Others

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	

The Pearson implementation in the book and also in R (the cor() fn) works with sparse data by only considering the “pairwise complete” items.

Hence:

- $\text{Mean}(\text{Rose}) = (3.5+3.5+2.5)/3 = 3.16$, $\text{Mean}(\text{Toby}) = (4.5+4+1)/3 = 3.16$
- $\text{Sim}(\text{Toby}, \text{Rose}) = \frac{(3.5-3.16)*(4.5-3.16) + (3.5-3.16)*(4-3.16) + (2.5-3.16)*(1-3.16)}{\sqrt{((3.5-3.16)^2+(3.5-3.16)^2+(2.5-3.16)^2) * ((4.5-3.16)^2+(4-3.16)^2+(1-3.16)^2)}} = 0.99$

(2) Compute Weighted Avg of Neighbour Ratings

Critic	Similarity	Night	S.xNight	Lady	S.xLady	Luck	S.xLuck
Rose	0.99	3.0	2.97	2.5	2.48	3.0	2.97
Seymour	0.38	3.0	1.14	3.0	1.14	1.5	0.57
Puig	0.89	4.5	4.02			3.0	2.68
LaSalle	0.92	3.0	2.77	3.0	2.77	2.0	1.85
Matthews	0.66	3.0	1.99	3.0	1.99		
Total			12.89		8.38		8.07
Sim. Sum			3.84		2.95		3.18
Total/Sim. Sum			3.35		2.83		2.53

The similarity of the other users to Toby (this was calculated using the Pearson Coefficient)

All neighbours with similarity > 0 are included in the weighted average (avoid having to select a value for K)

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

Note that, for simplicity, only the raw ratings are used here

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 $\sim O(M \cdot N)$ where $M = \text{\#users}$, $N = \text{\#items}$
 - In practice $\sim O(M + N)$ since most users have rated/purchased few items
- Possible solutions
 - Item-Item Collaborative Filtering
 - Dimensionality Reduction
 - Streaming Collaborative Filtering

Another way to Scale Collaborative Filtering

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

- Compute the distance between items i and j using the adjusted cosine similarity*:

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

{

$r_{u,i}$ = rating of user u on item i
 $r_{u,j}$ = rating of user u on item j
 \bar{r}_u = mean rating of user u
 U = set of all users

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

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

{

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

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

Item-Based Recommendations for Toby

↓ Movies seen by Toby → Movies not seen by Toby

Movie	Rating	Night	R.xNight	Lady	R.xLady	Luck	R.xLuck
Snakes	4.5	0.182	0.818	0.222	0.999	0.105	0.474
Superman	4.0	0.103	0.412	0.091	0.363	0.065	0.258
Dupree	1.0	0.148	0.148	0.4	0.4	0.182	0.182
Total		0.433	1.378	0.713	1.764	0.352	0.914
Normalized			3.183		2.473		2.598

Toby's ratings
for the movies
he has seen

The similarities between **Night Listener** and the movies seen by Toby – these are derived from the pre-computed item-item similarity matrix (using Euclidean similarity)

$$(\text{= } 1.378/0.433)$$

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**

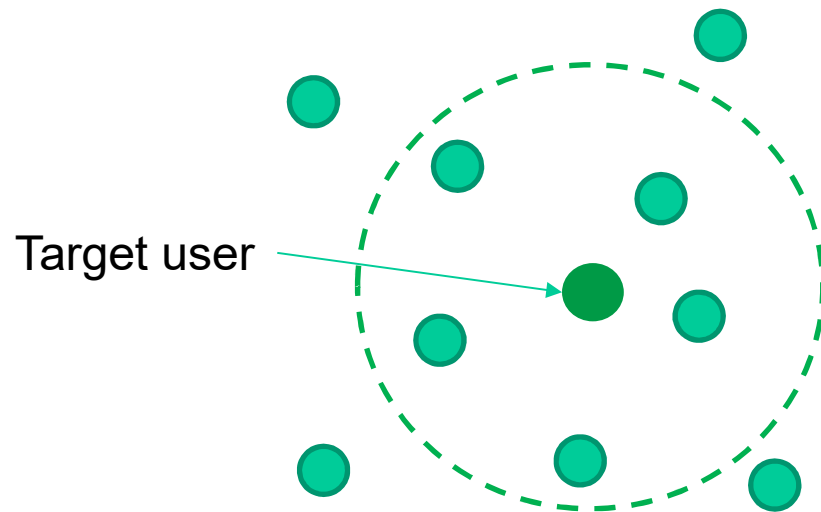
Which is Best: User-Based or Item-Based?

- In general...
- If $\#users \gg \#items$ (e.g. Amazon) then Item-Item may be better
- If $\#users \ll \#items$ (e.g. recommending web content) then User-User may be better
- Workshop I
 - Compare User-based versus Item-based CF
 - Compare different similarity measurements
 - Compare normalised versus non-normalised data

Nearest Neighbours vs All Neighbours?

■ Restricting the neighbours to the nearest ones ?

- Nearest K (small K or large K?)
- Similarity $\geq M$ (what M?)

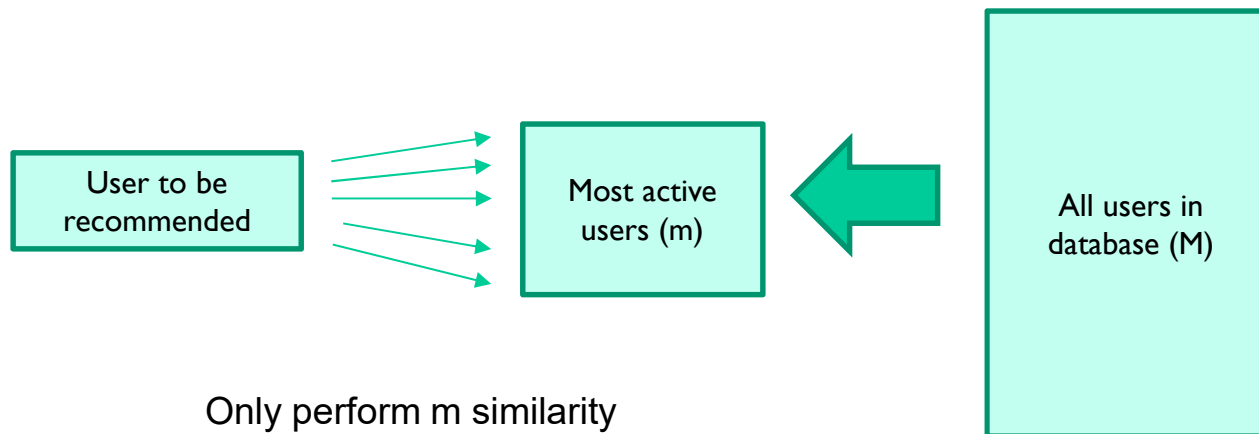


- No clear winner
- Depends also on operational issues, size & sparsity of data etc.
- E.g. for sparse data there may be no ratings for item X in the K nearest neighbours

Predicted rating = weighted average rating of the K nearest neighbours

Data Reduction

- Massive datasets increase computational load
- For huge datasets, data sampling may be effective in certain situations
- User Sampling:
 - Reduce the number of customers
 - Random sampling, or business directed, e.g. select only the most active customers

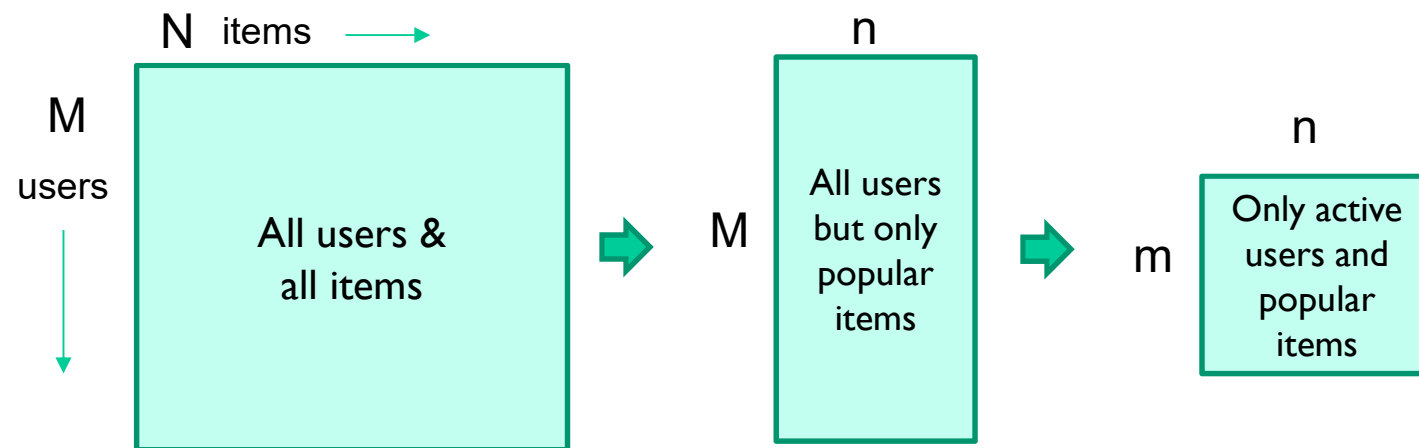


Only perform m similarity
computations ($m \ll M$)

Data Reduction

■ Item Sampling

- Large item sets can also cause huge computational overheads
- Reduce the number of items to recommend from
- Random sampling or business directed, e.g. select only the most popular items

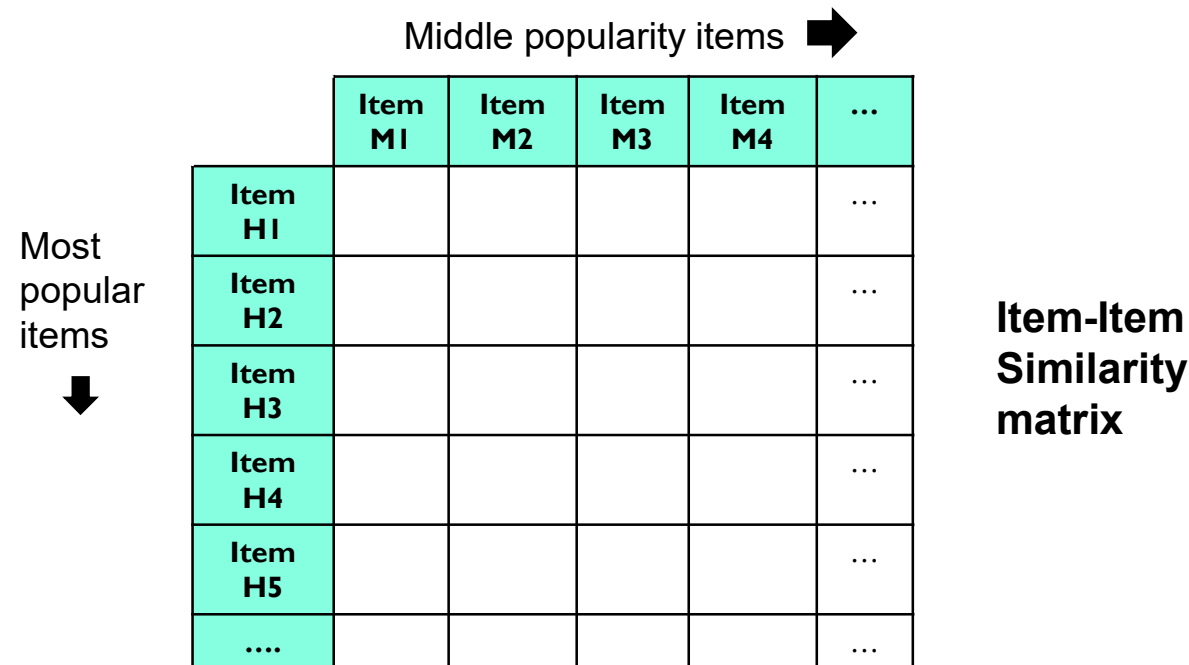


■ Item Categorisation

- Can dramatically reduce the data size and may improve accuracy
- But.... less granular recommendations might not satisfy users

Data Sampling & the Long Tail

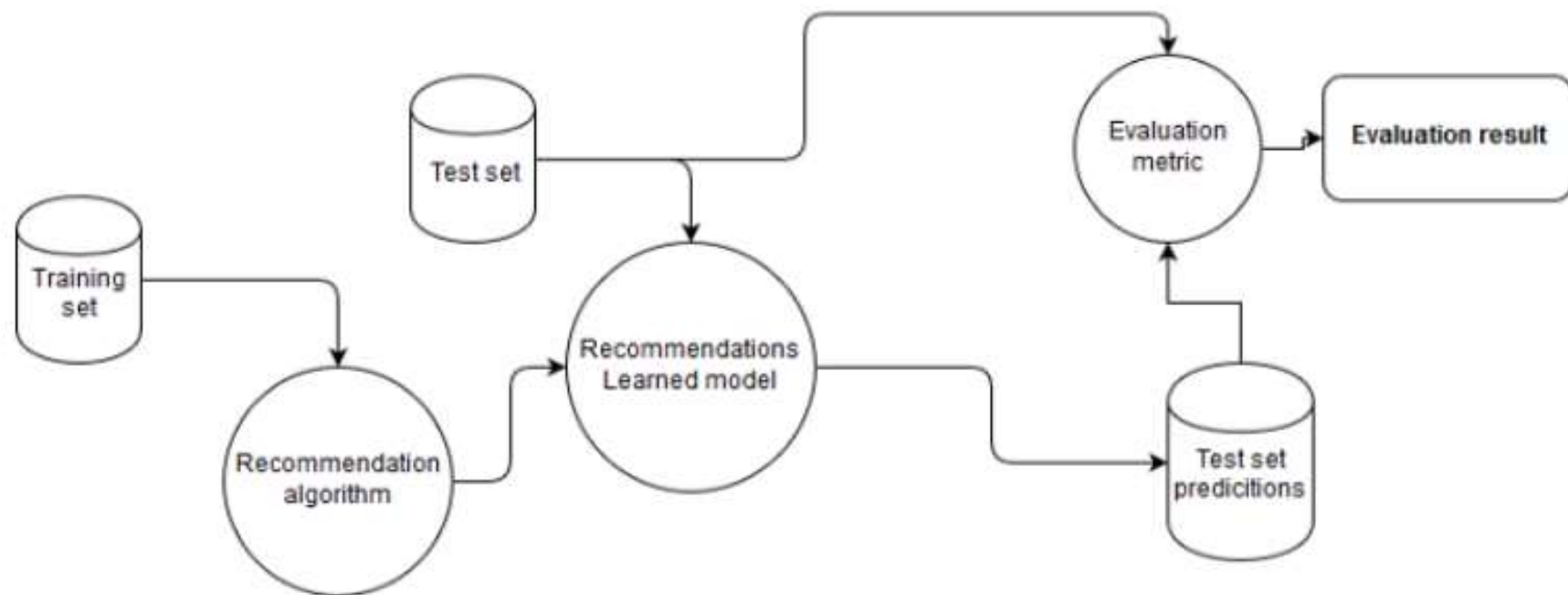
- Only considering popular items may cause us to only recommend items in the short head not the long tail.
- Other strategies are possible, e.g.
 - E.g. For user-based CF use the reduced (popular) item set for computing the user-user similarity, then recommend using the full item set
 - E.g. For item-based CF, compute only similarities between very-popular and middle-popular items.



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?

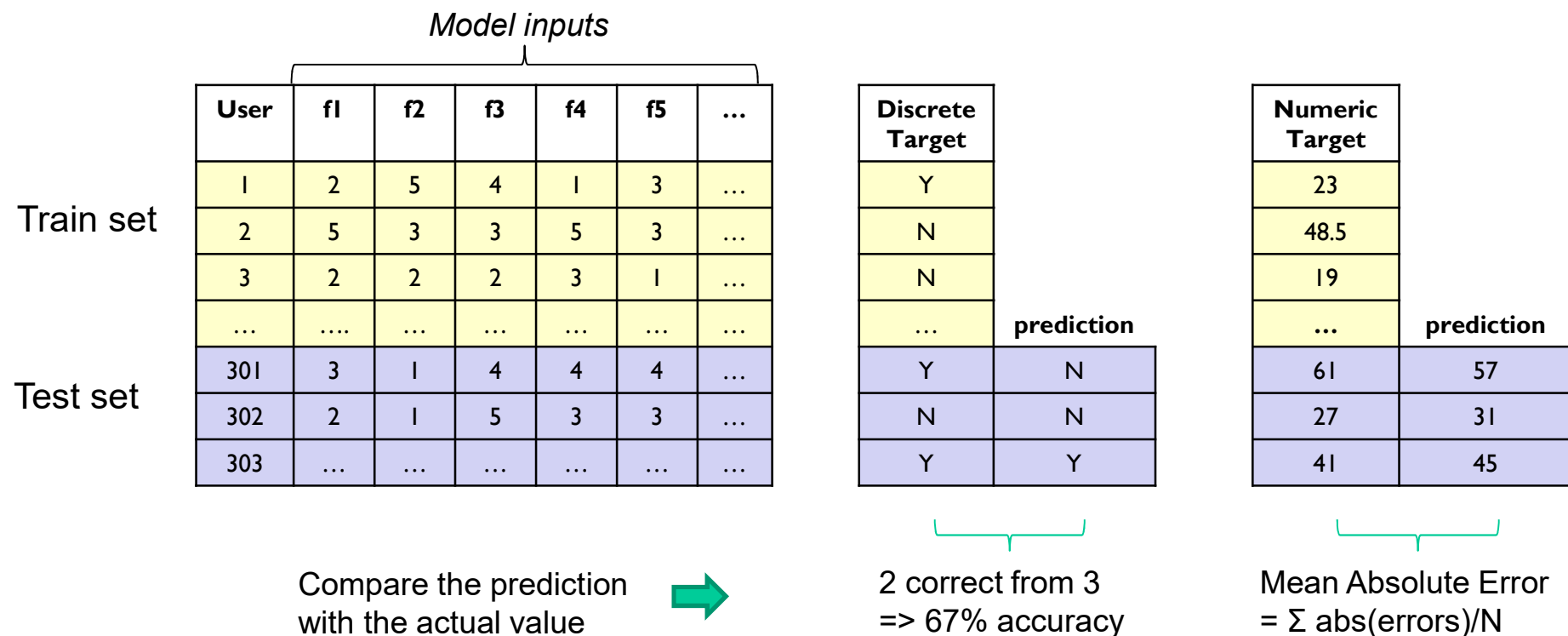
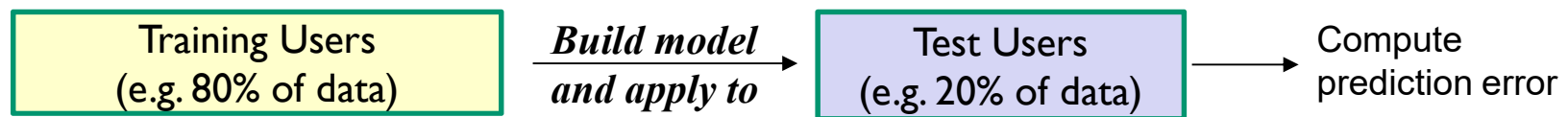
Evaluating Recommender Systems



Recommendation system workflow

Evaluating Recommender Systems

- Using historical data we can test the accuracy of the rating prediction*
- Common test for a Predictive Model (e.g. logistic regression):




Measuring the accuracy of the rating prediction

- For ratings data there is no target variable!
- Instead we hide, in turn, each rating made by each test user and then predict the rating for the hidden item, we can then compute:

$$\text{Error} = \text{actual rating} - \text{predicted rating}$$

User	movie1	movie2	movie3	movie4	movie5	movie6	etc....
Test user	2	5	4		3	1	



For each movie rated by the test user:

- Set the movie rating to blank (NA) – but keep a copy
- Make a prediction for that movie using the training data
- Compare the prediction with actual rating:

$$\text{error} = \text{abs}(\text{predicted rating} - \text{actual rating})$$
- Increment a count of the number of tests
- Restore the blank movie rating

At the end compute the MAE (mean absolute error)


$$\text{MAE} = \frac{\sum \text{abs(errors)}}{\# \text{ tests}}$$

Measuring the accuracy of the rating prediction

- A common variant is to split the data into training & test sets by randomly assigning individual ratings to the test set, e.g. assign ~ 30% to the test set

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

	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

Train set (after deleting test ratings)

Test set

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

Working with Implicit Ratings

■ Issues with Explicit Ratings

- Very sparse - users tend to be lazy, often don't bother to rate
- Users may not always tell the truth? E.g. Influenced by peer/friend opinions (all of my friends liked that movie so maybe it was good after all!)
- May not be available at all if no system in place to collect them
- Watching what the user actually does (e.g. what they view or buy) may be more reliable / accurate than ratings



- Buying a product
- Viewing a (product) page
- Clicking on a link
- Time spend looking at a page
- Repeat visits
- Referring a page to others

BUT...

- Buying something doesn't always mean liking something
- Could be a mistake buy or impulse buy or mistaken page-click etc.
- I may be buying for someone else using my own account

Working with Implicit Ratings

- Visits to webpages is a commonly used implicit feedback signal:
 - Repeat visits to a page implies liking the page/product (more repeats => more likes)
 - More time on page implies liking the page/product (longer duration => more like)

User (or Session)	page1	page2	page3	page4	page5	page6
1	2	5	4		3	1
2		3		5	3	1
3			5	3		

*Implicit
ratings
matrix*

E.g. Assume the ratings here refer to the time spent on page (normalised to 1-5)

- This looks similar to regular (explicit) ratings. Can we proceed as before?
 - Yes, we can try. But, since all users with implicit ratings like the page to some degree, if we predict the implicit rating we are predicting the amount of like and not like/dislike



Implicit ratings are often treated as Binary

- Its common to treat ANY page view (or other 'like' behaviour) as a like, else don't know

User (or Session)	page1	page2	page3	page4	page5	page6
1	2	5	4		3	1
2		3		5	3	1
3			5	3		



User (or Session)	page1	page2	page3	page4	page5	page6
1	1	1	1		1	1
2		1		1	1	1
3			1	1		

- But now we can't use Cosine Similarity (or similar) since result will always be 1

$$\text{Cosine Similarity} = (1*1 + 1*1 + \dots) / \sqrt{(1^2+1^2 + \dots)*(1^2+1^2 + \dots)} = N/N = 1$$

$$\text{Euclidean Similarity} = 1 / (1 + \sqrt{(1-1)^2 + (1-1)^2 + \dots}) = 1/1 = 1$$

- To apply Cosine or Euclidean we must assume the NA's => 0 (don't like)

Binary Ratings: Jaccard Similarity

- Measures the similarity between two sets
- Makes no assumptions about the missing values

$$\text{Sim}_{\text{jaccard}}(A,B) = |A \cap B| / |A \cup B|$$

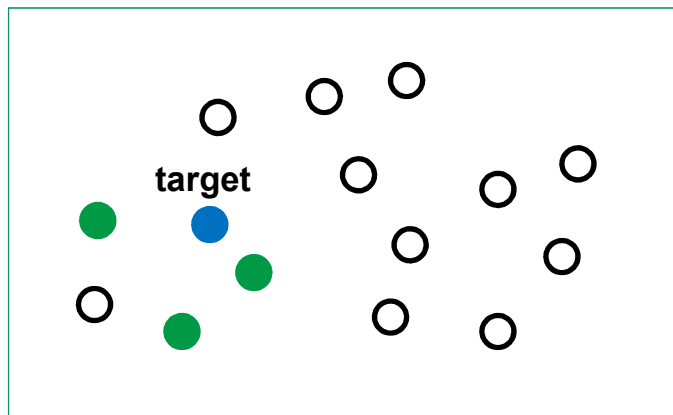
User	b1	b2	b3	b4	b5	b6	b7	b8	b9	b10
U1	1	1	1				1			
U2		1			1	1	1	1		

What is the Jaccard similarity for (u1,u2)?

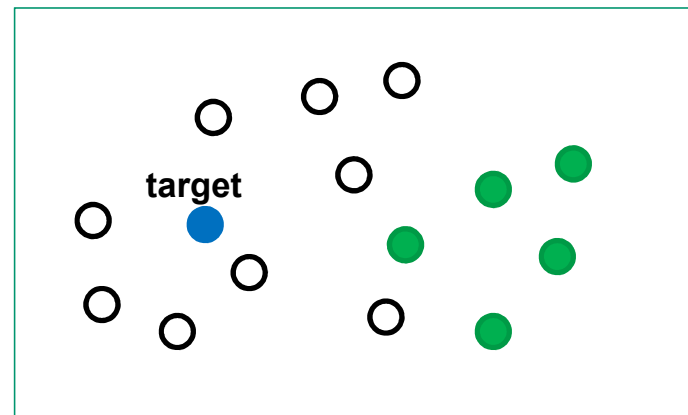
Binary Ratings: Making Recommendations

- **Issue:** For an unseen item (X), the weighted average rating of the target user's neighbor's on that item is always 1 (the weighted average of many 1's is 1) . So how do we rank all of the unseen items in order to make a recommendation?
- **Possible Solution:** rank unseen items using the average similarity of the target user to the users who liked the item
 - Rational: high similarity suggests that the target user may also like X

Users plotted in feature space...



Target is likely to “like” X



Target is less likely to “like” X

- Target user
- Likes X
- Unknown for X

Later we examine MF approaches exist for handling Implicit Ratings

Testing when using Implicit Ratings

- If we treat the implicit ratings the same as explicit ratings
 - We can predict the implicit rating and hence compute its MAE
- If we treat the implicit ratings differently from explicit ratings
 - E.g. treat as binary like signals (like/unknown)
 - We cannot measure MAE since there is no predicted rating

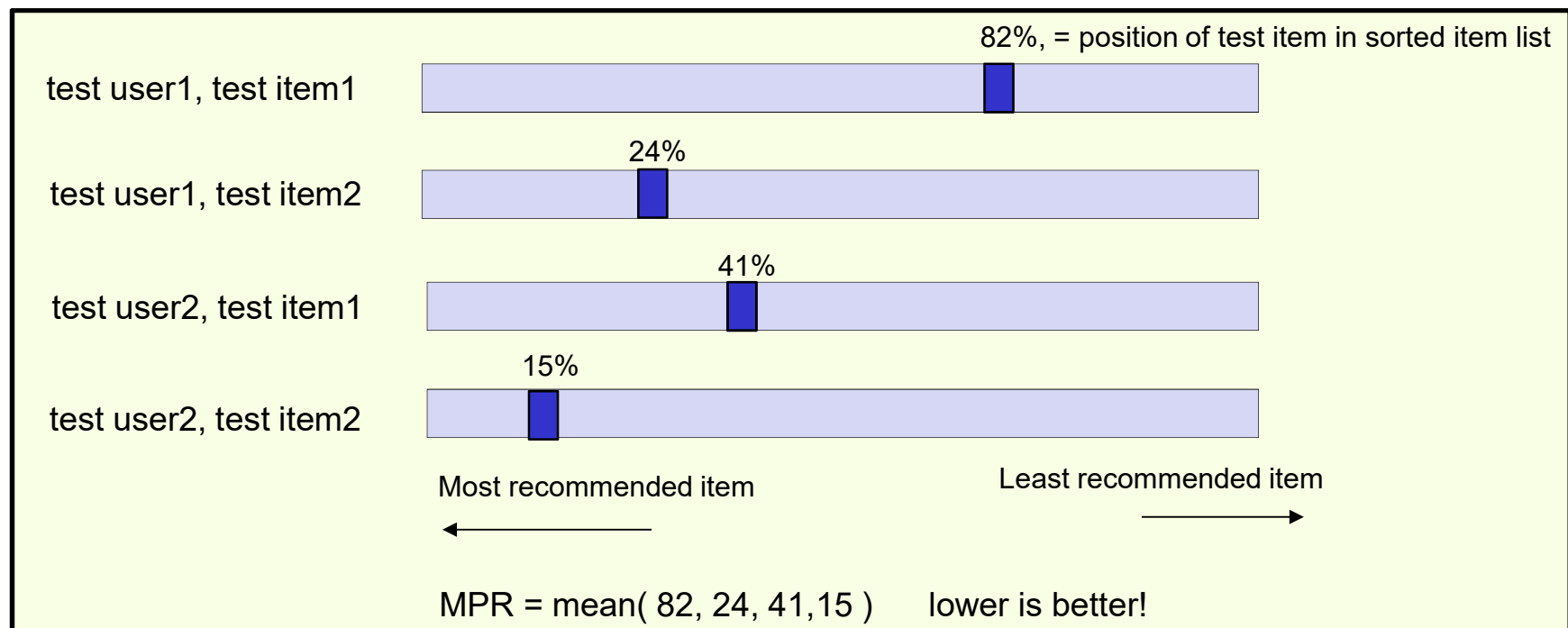


What does a MAE of (say) 1.18 mean in practice anyway?

We really need to know how accurate the **recommendations** are

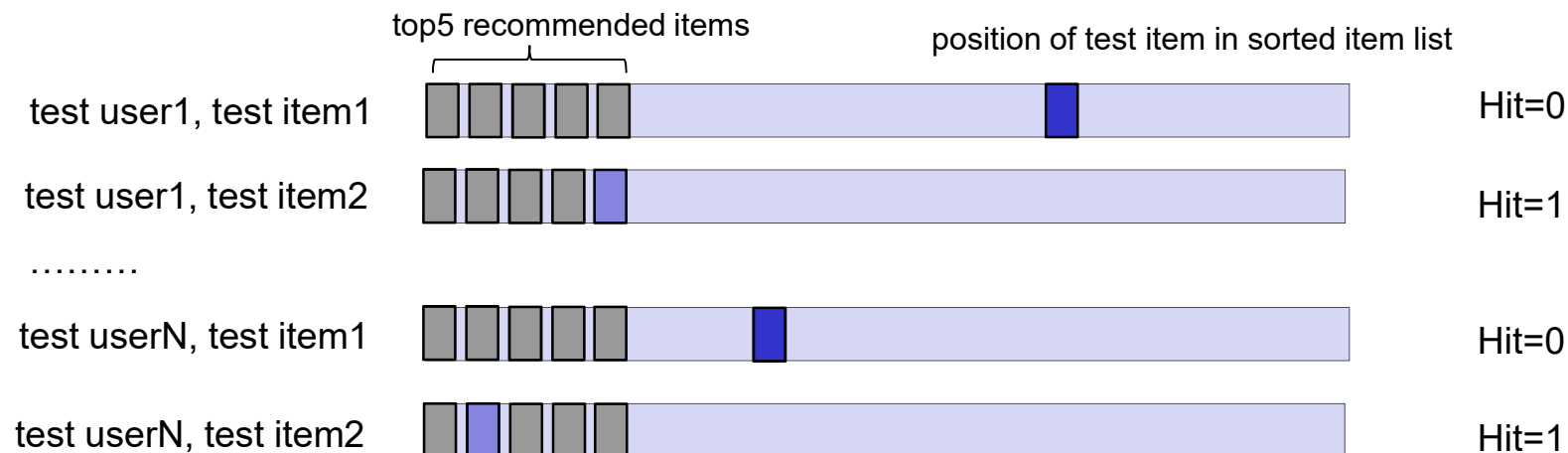
Mean Percentage Ranking (MPR)

- Test set = as before, but only consider items that are **liked** (e.g. have rating ≥ 4)
- For each test item, predict its position (as a %) in the ranked list of all items (ranked by their decreasing likeability by the user, e.g. their predicted rating)
- Since we know the item is liked we hope its predicted rank is low (to the left), e.g. Rank=1 \Rightarrow top of list \Rightarrow top recommendation
- $MPR < 50\%$ indicates test results are better than random guessing



Estimating Lift

- Make recommendations for each test user by selecting the top-K items in the sorted list.
- Count how many times a test item appeared in the recommendations (call this the #hits).
- Compare with how many would be expected if random recommendations are made
- E.g. if top-K = 5:



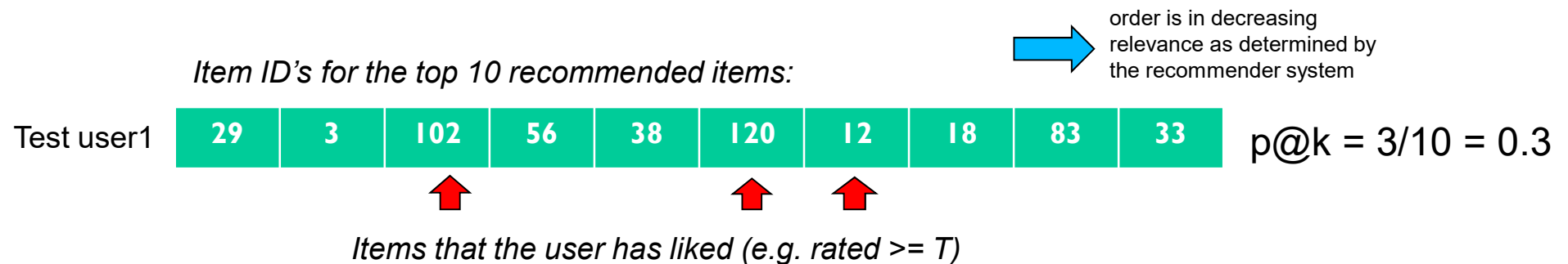
$$\text{Hit\%} = (\text{\#hits}) / (\text{\#recommendations made})$$

$$\text{Lift} = (\text{hit\% using model}) / (\text{hit\% using random})$$

*****For large item sets you may need large test sets to get significant results!***

Precision at K (P@K)

- Precision at K is the proportion of recommended items in the top-K set that are relevant
- An item is relevant if the user is known to like it (e.g. actual rating ≥ 4)
- E.g. if $K = 10$...



Precision@k = (# of recommended items @k that are relevant) / (# of recommended items @k)

*When computing $p@k$ for a user from a test set, we remove from the recommended item list all items not in the test set for that user (these are the items that we do not know the actual rating for)

https://medium.com/@m_n_malaeb/recall-and-precision-at-k-for-recommender-systems-618483226c54