University of Salford MANCHESTER

SCHOOL OF SCIENCE, ENGINEERING & ENVIRONMENT

Program: MSc of Data Science
Module: Big Data Tools and Techniques

Week 8

Recommender Systems in Apache Spark

2025

Expectations

1. Choose a quiet place to attend the class and please concentrate during the lecture
2. Don't raise hand or ask a question during the lecture
3. Put your questions in Padlet (not Teams' chat box) and I will review them in the due time (Padlet link is in BB, week 8, Lecture) – we will also end on a Q&A if there is time!
4. Turn off your mic during the lecture
5. We will have 5 mins break after the first hour of the lecture (please remind me)
6. Jisc code will be shared during the break time

Learning Outcomes

By the end of this lecture and workshop, you will be able to:

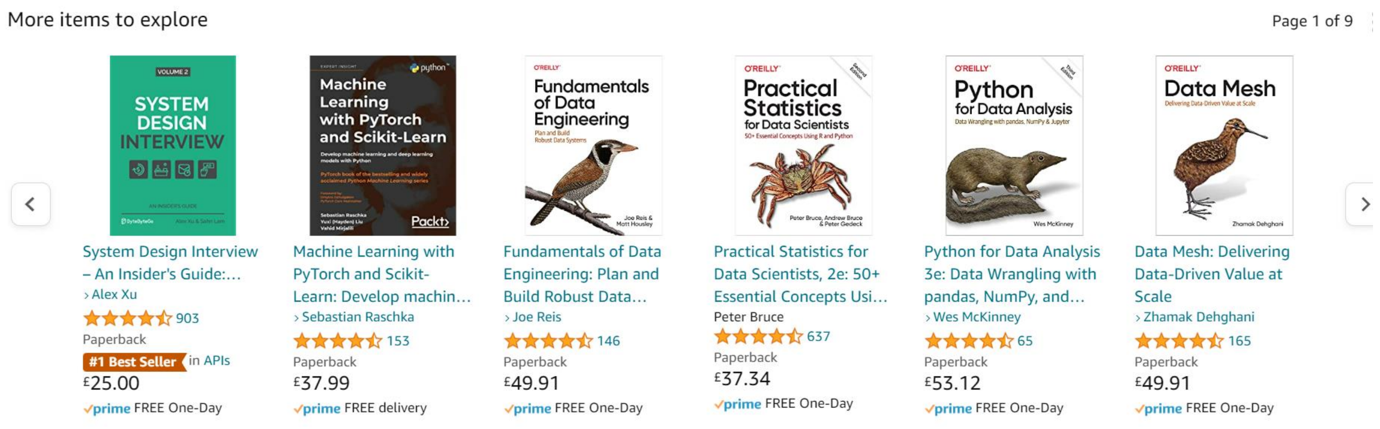
1. Describe the basic principles behind content based filtering and collaborative filtering recommender systems
2. Explain how matrix factorization is used in a collaborative filtering recommender system to find item and user embeddings
3. Use Apache Spark to apply ALS to build and test a recommender model

Pre-Class Activity

- It’s not too late to do the Pre-Class Activity!
- If you want to generate some personalised movie recommendations you should do this before the workshop ☺

Recommender Systems

Recommender systems are ubiquitous…

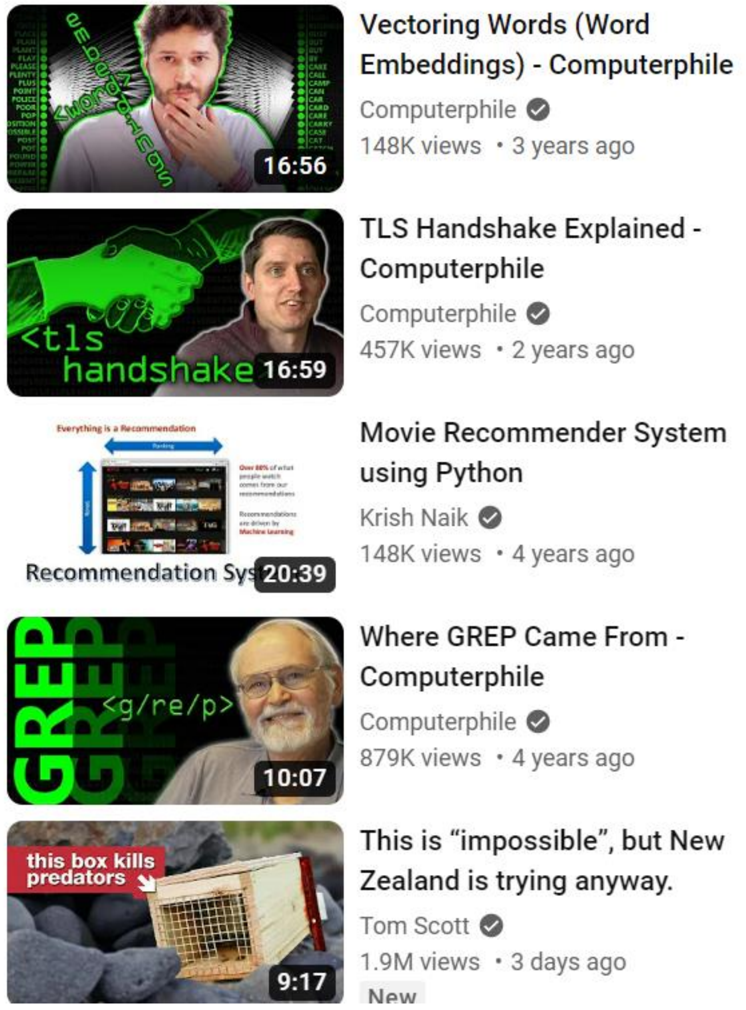


More items to explore

Page 1 of 9

System Design Interview – An Insider's Guide:…	Machine Learning with PyTorch and Scikit-Learn: Develop machin…	Fundamentals of Data Engineering: Plan and Build Robust Data…	Practical Statistics for Data Scientists, 2e: 50+ Essential Concepts Usi…	Python for Data Analysis 3e: Data Wrangling with pandas, NumPy, and…	Data Mesh: Delivering Data-Driven Value at Scale
› Alex Xu	› Sebastian Raschka	› Joe Reis	Peter Bruce	› Wes McKinney	› Zhamak Dehghani
903	153	146	637	65	165
Paperback	Paperback	Paperback	Paperback	Paperback	Paperback
#1 Best Seller in APIs					
£25.00	£37.99	£49.91	£37.34	£53.12	£49.91
prime FREE One-Day	prime FREE delivery	prime FREE One-Day	prime FREE One-Day	prime FREE One-Day	prime FREE One-Day

Think of Amazon, YouTube, Spotify…



Vectoring Words (Word Embeddings) - Computerphile
Computerphile
148K views • 3 years ago
16:56

TLS Handshake Explained - Computerphile
Computerphile
457K views • 2 years ago
16:59

Movie Recommender System using Python
Krish Naik
148K views • 4 years ago
20:39

Where GREP Came From - Computerphile
Computerphile
879K views • 4 years ago
10:07

This is "impossible", but New Zealand is trying anyway.
Tom Scott
1.9M views • 3 days ago
New
9:17

Recommender Systems

We’re covering them in this module because Apache Spark is a widely used framework to power recommendation engines.

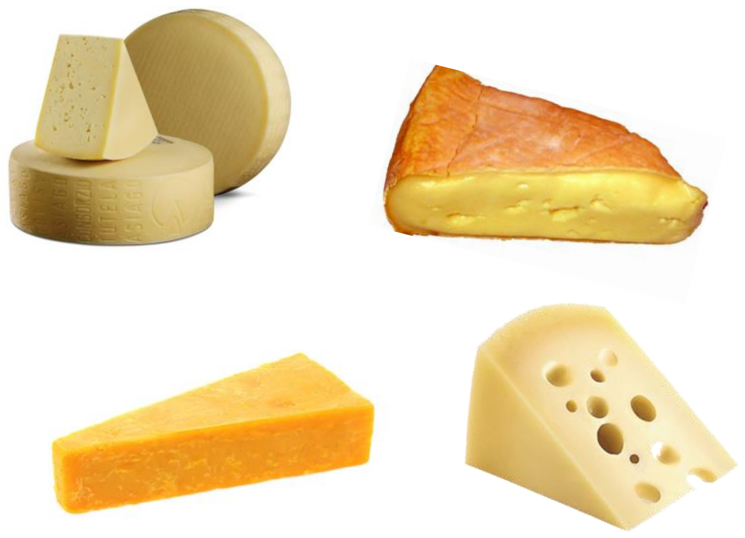
All make use of Apache Spark for their recommendation engines.

https://spark.apache.org/powered-by.html

Why is this a Big Data problem?

- **Volume:** Millions of users and products generate massive datasets.
- **Velocity:** New data is constantly generated as users interact with systems.
- **Variety:** Data comes from multiple sources (explicit ratings, purchase history, browsing behaviour, implicit interactions).

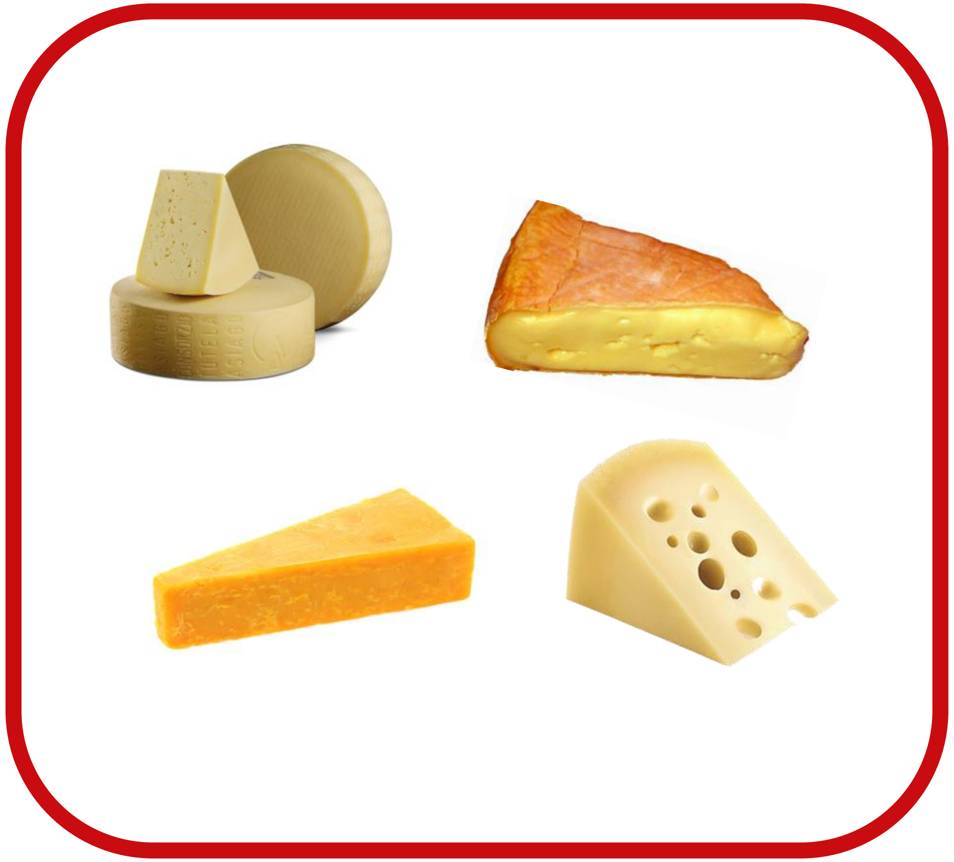
Making Recommendations



You own a shop and have noticed Alice comes in and buys cheese regularly.

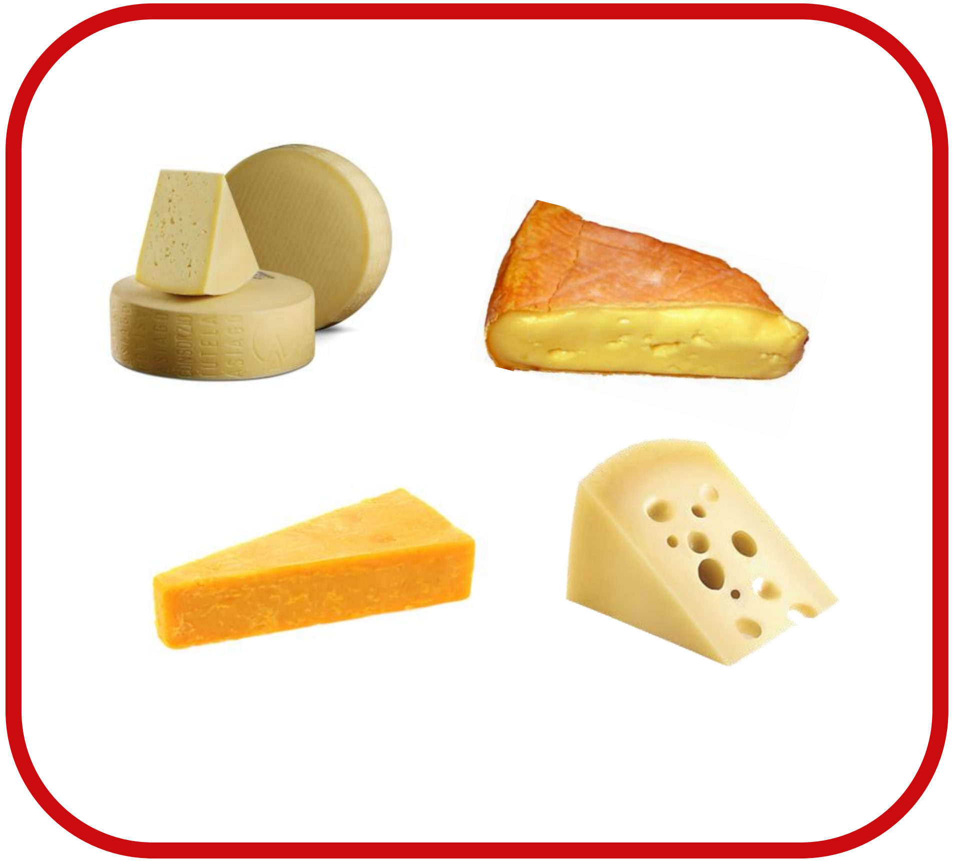
You want to recommend another product to her when she comes in….

Making Recommendations



How do you choose what you should recommend to her?

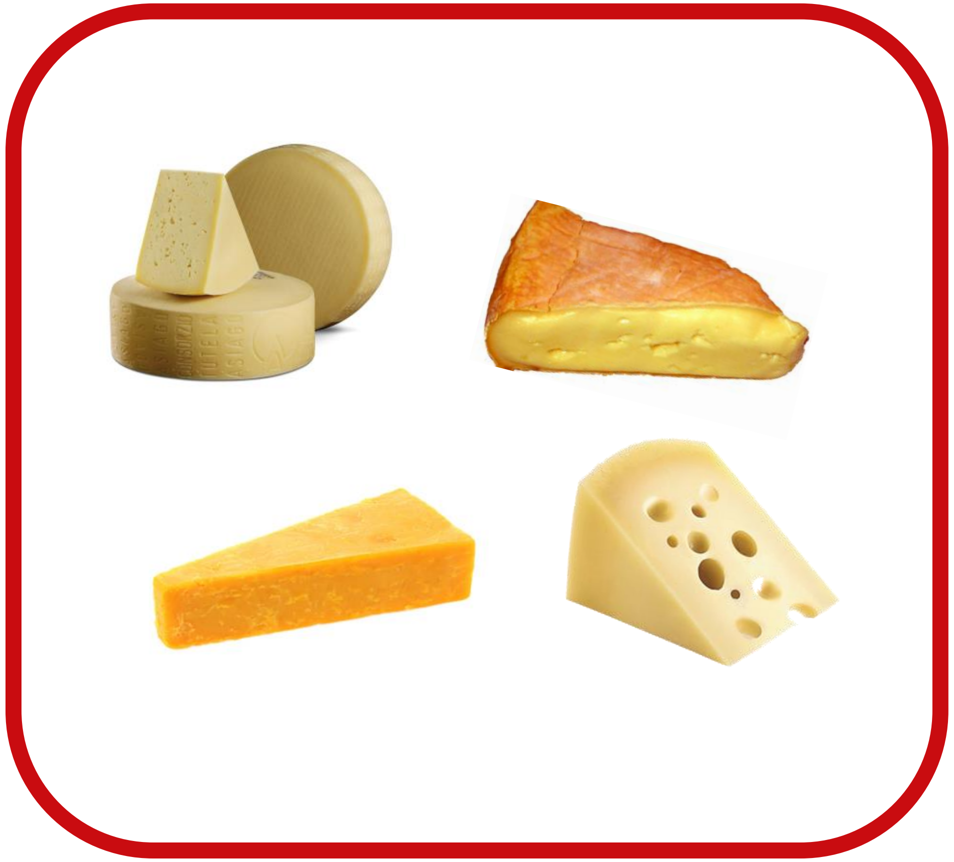
Making Recommendations



OPTION 1

Recommend another cheese as that's similar to what she has been purchasing already!

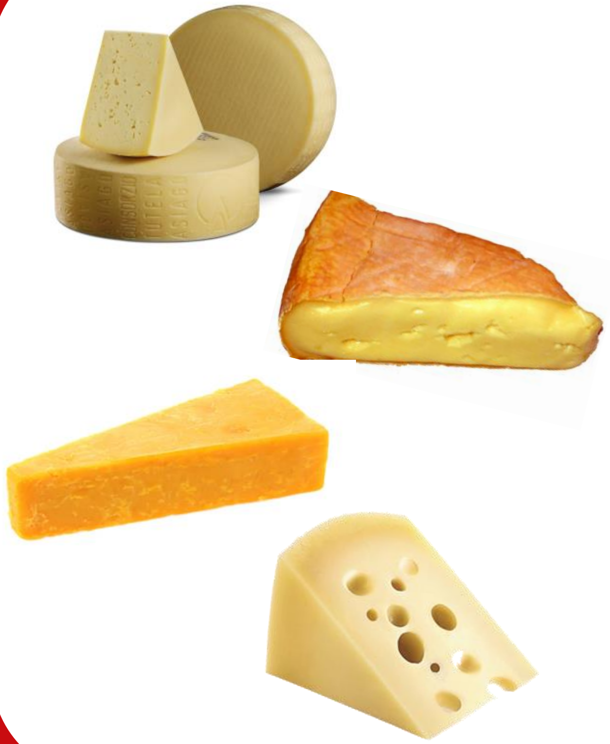
Making Recommendations



OPTION 2

You've noticed Bob also buys cheese regularly, but he normally buys red wine too. Maybe you can recommend that to her?

Making Recommendations



OPTION 1

Content based filtering

OPTION 2

Collaborative filtering

Types of Recommender System

- **Content based filtering** uses some measure of similarity to identify items which are similar to what a user has liked previously

- **Collaborative filtering** uses similarities between users and items **simultaneously** to generate recommendations

Content Based Filtering

- Content based filtering uses some measure of similarity to identify items which are similar to what a user has liked previously
- It uses user profile information and metadata on the items to make recommendations using a measure of similarity
- For example, the first time you logon to Netflix they have no data on you. So they ask you to rate a couple of films you have watched and then uses this to create a watchlist
  - E.g. if you enjoyed *Harry Potter* and *Narnia,* it can identify you like fantasy movies and recommend *Lord of the Rings*

Content Based Filtering

- Content-Based Filtering recommends items similar to those a user has liked before.
- It uses item metadata (e.g., genre, ingredients, features) to compute similarity.
- Each item is represented as a vector in a feature space.
- The system finds similar items by computing distances between these vectors.

Content Based Filtering

- Example: A movie recommendation system
- Each movie is represented as a vector (e.g., [Action, Romance, Sci-Fi])
- User preferences are also a vector based on past ratings.
- Similarity (e.g., cosine similarity) is used to find the closest match.

This requires us to decide what features should be included in the feature space!

Content Based Filtering Advantages

- You don’t need existing data from other users – the recommendations are specific to the current user
- This might make this approach particularly useful for new businesses without existing user data, or for new users
- The model can capture the specific interests of the individual user

Content Based Filtering Disadvantages

- The recommendations are purely based on similarity to a specific users previous preferences so the recommendations might seem obvious and don't help them discover new interests (think of the cheese and wine example)
- The model relies on us 'hand-engineering' the right features on which to judge similarity – which takes time and expertise
  - As an example, imagine that in the example on the previous slide, Netflix used 'Film Run Time' instead of 'Genre' to judge similarity. It would probably not give good recommendations! (This isn't a particularly realistic example, but the point is we need to understand the domain to know what the important features are!)

Types of Recommender System

- **Content based filtering** uses some measure of similarity to identify items which are similar to what a user has liked previously

- **Collaborative filtering** uses similarities between users and items **simultaneously** to generate recommendations

User-Item Matrix

We can start with a feedback matrix where each row is a user and each column is an item.

If the user has provided a rating then we populate the relevant entry.

Alice	5		2	
Bob		4		
Carole	5			1
Toni		2		3

Explicit vs Implicit Ratings

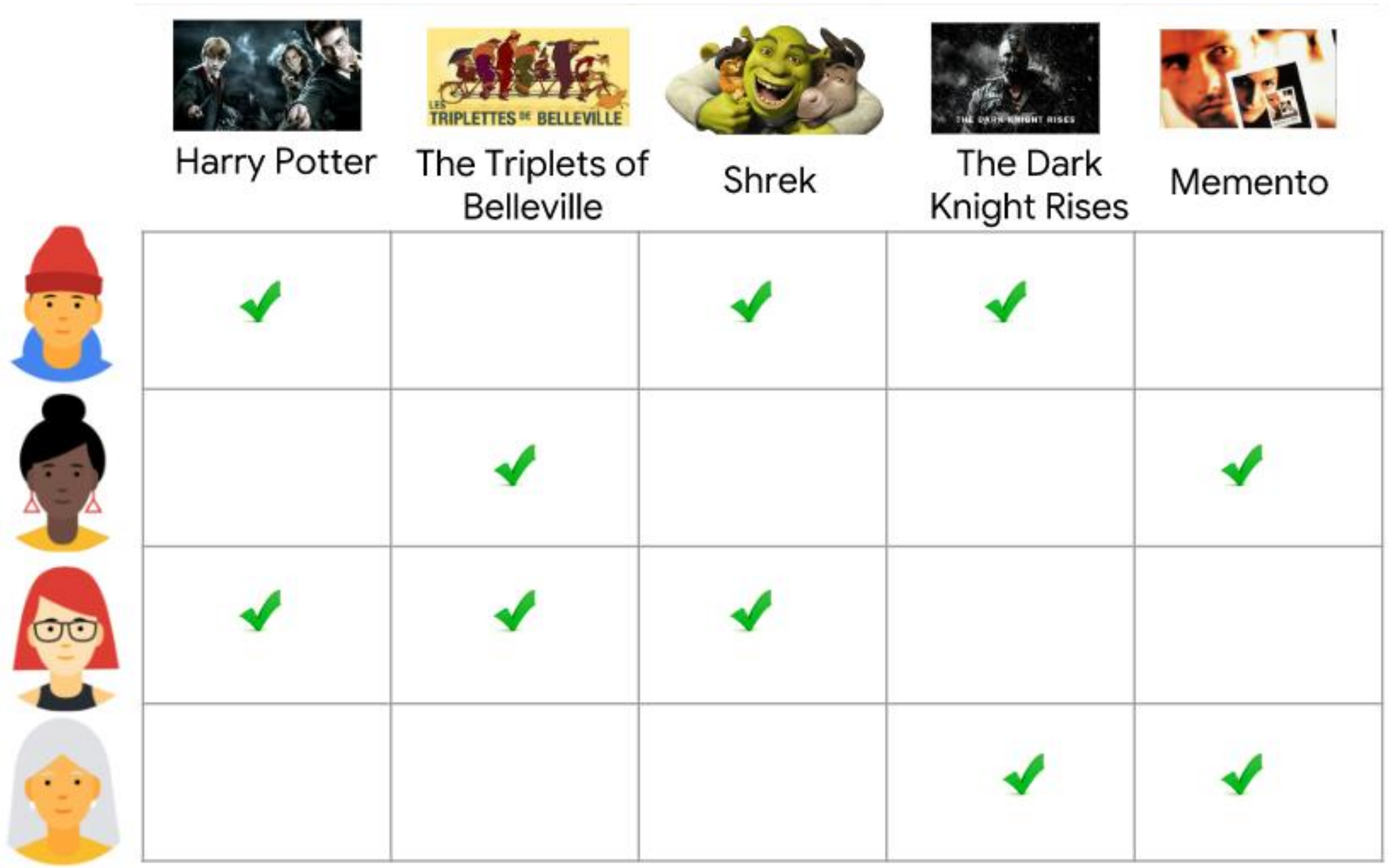
- Explicit Ratings are where users directly provide feedback on items.
- Examples:
  - Star ratings (e.g., giving a movie 4 out of 5 stars on Netflix).
  - Thumbs up/down (e.g., YouTube's like/dislike button).
  - Written reviews (e.g., product reviews on Amazon).
- More accurate and reliable since users express their true preferences.
- Often sparse, as many users do not provide explicit feedback.

Explicit vs Implicit Ratings

- **Implicit Ratings** are inferred from user behaviour rather than direct feedback.
- Examples:
  - Time spent watching a video (e.g., if a user watches 90% of a movie, it's likely they enjoyed it).
  - Clicks on products (e.g., frequently clicking on a product page suggests interest).
  - Purchase history (e.g., buying an item may indicate preference).
- More abundant since user behaviour is always being tracked.
- Less reliable—just because someone clicked on something doesn't mean they liked it.

User-Item Matrix

We can start with a feedback matrix where each row is a user and each column is an item…



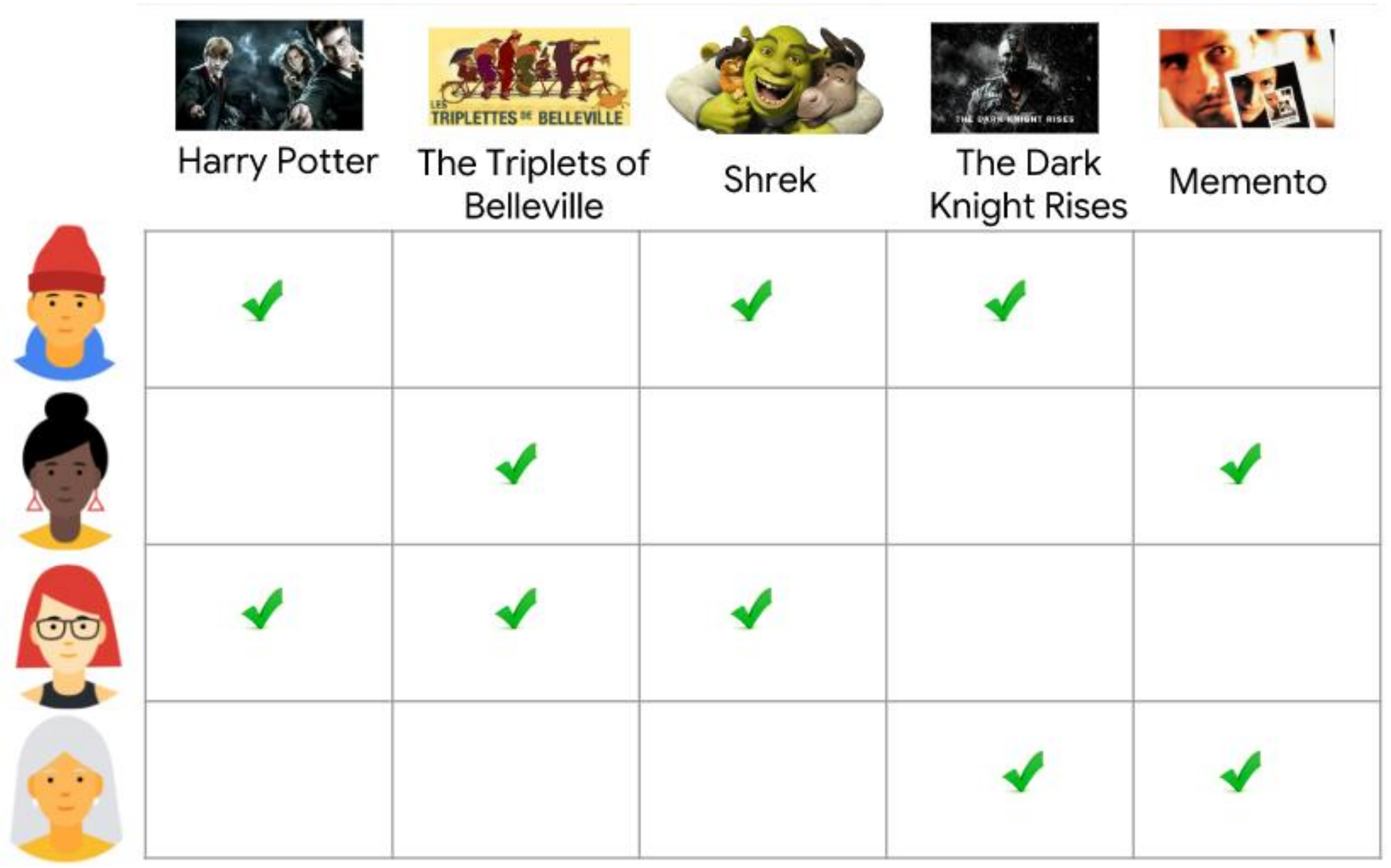
	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

User-Item Matrix

In a real-world example this matrix is:

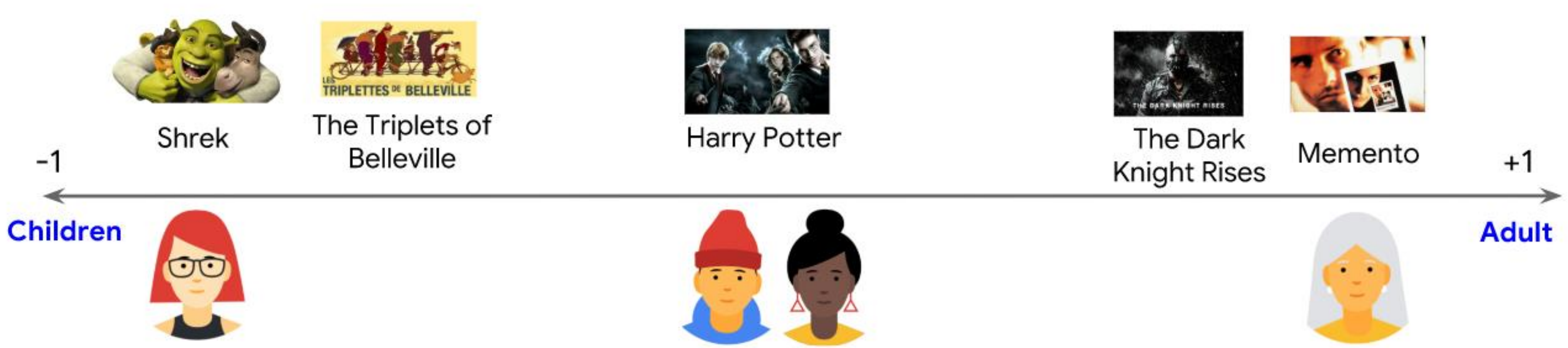
- Very big: e.g. Netflix has 230m subscribers and ~17,000 titles, which gives a matrix with around 3.9 trillion cells.
- Very sparse: e.g., most users have only seen or rated a tiny fraction of the 17,000 titles



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

1 Dimensional Embedding

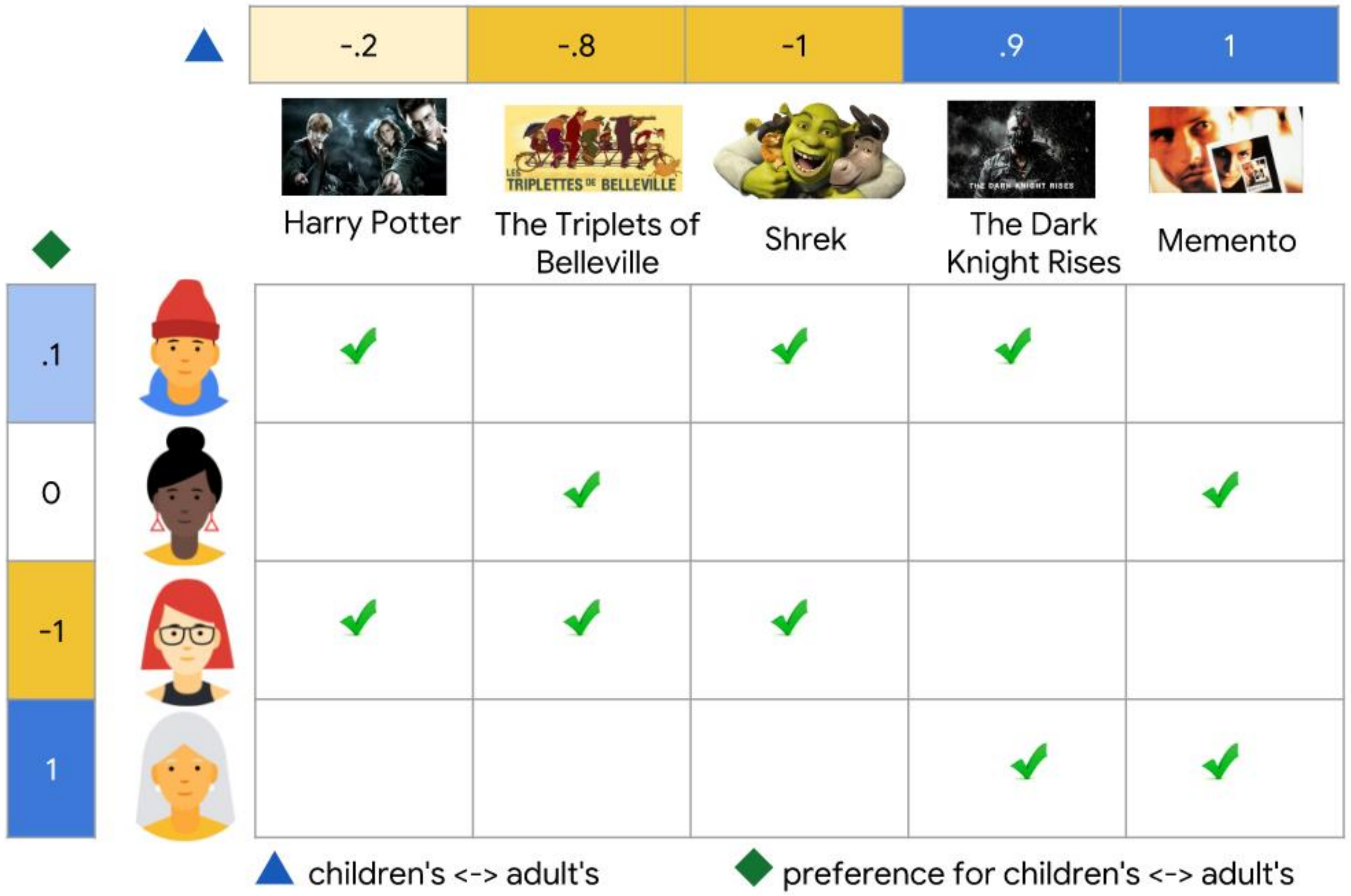


Can we capture these film preferences by assigning each movie a number between -1 and 1 to represent whether it's a children's movie or an adult movie?

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

1 Dimensional Embedding

When we multiply the user embedding and the movie embedding the values for the movies the individual likes should be closer to 1.



◆		Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
▲		-.2	-.8	-1	.9	1
.1		✓		✓	✓	
0			✓			✓
-1		✓	✓	✓		
1					✓	✓

▲ children's <-> adult's ◆ preference for children's <-> adult's

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

2 Dimensional Embedding

Since we couldn't fully capture the user preferences in one dimension, lets try two dimensions…

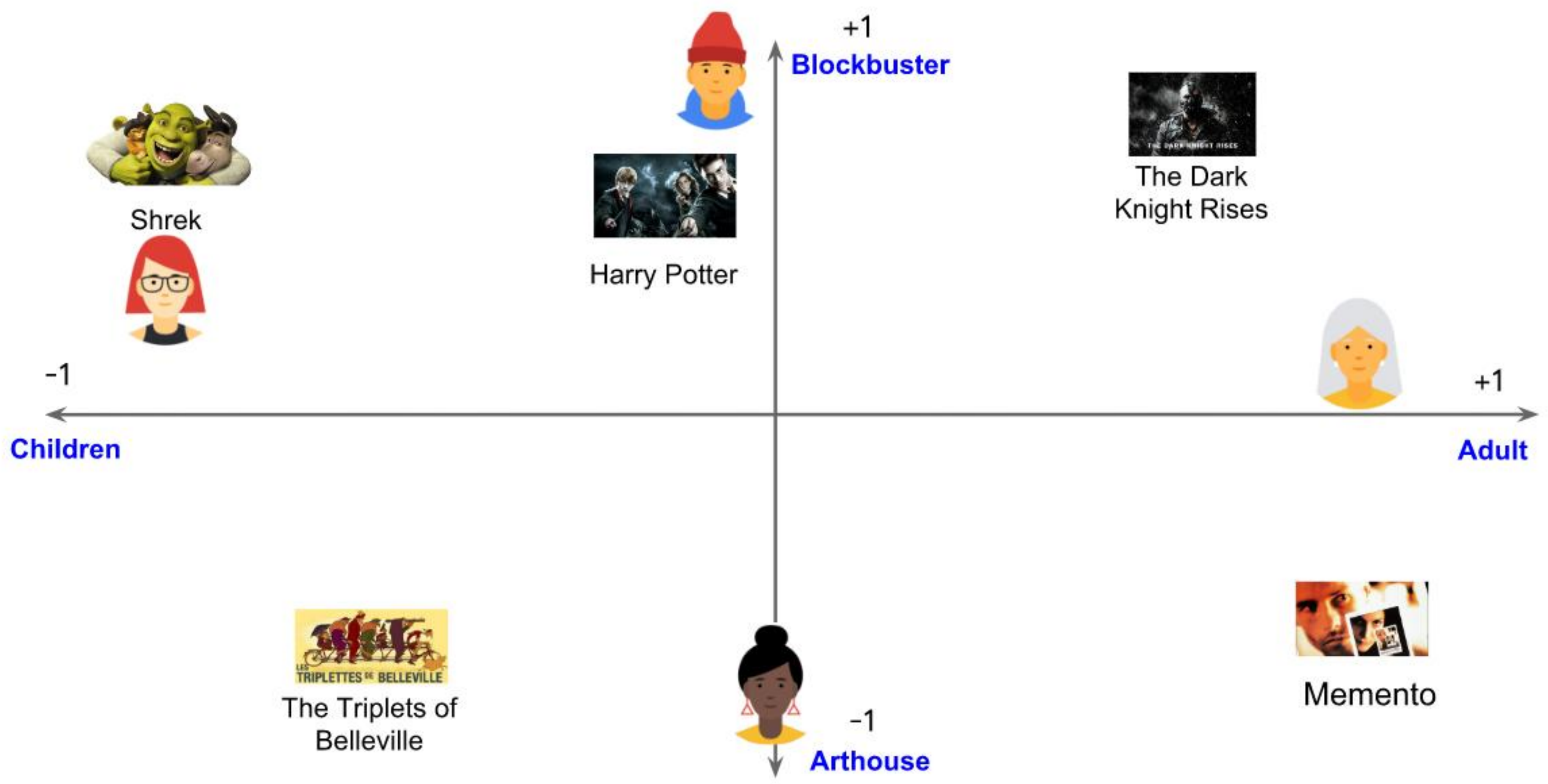
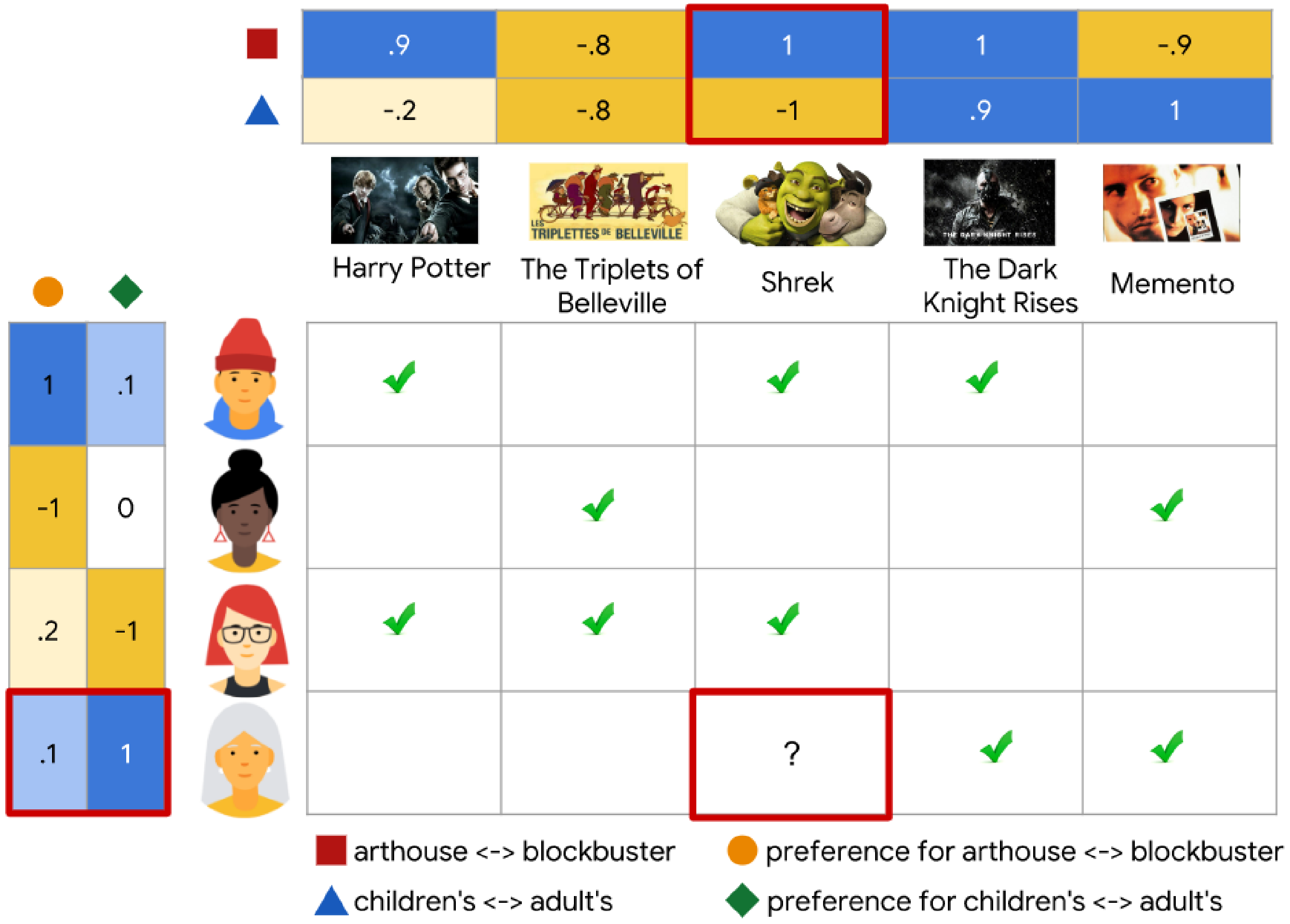


Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

2 Dimensional Embedding

This now captures the user preferences for all users better than before!



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
🟥	.9	-.8	1	1	-.9
▲	-.2	-.8	-1	.9	1

●	◆	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
1	.1	✓		✓	✓	
-1	0		✓			✓
.2	-1	✓	✓	✓		
.1	1			?	✓	✓

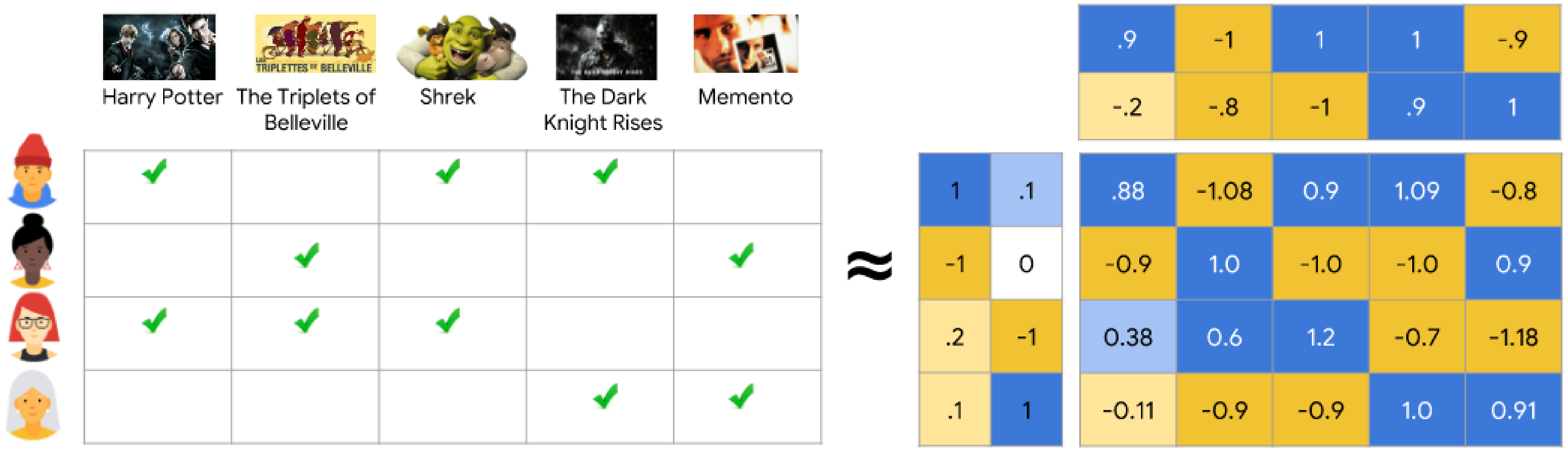
🟥 arthouse <-> blockbuster
▲ children's <-> adult's
● preference for arthouse <-> blockbuster
◆ preference for children's <-> adult's

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

Collaborative Filtering

- In this example, we hand-engineered the features (dimensions) – i.e., we picked the features and chose a value for each user and movie to try and represent them as we thought best.
- When we carry out collaborative filtering, we want to learn these embeddings **automatically.**

Collaborative Filtering



	Harry Potter	The Triplets of Belleville	Shrek	The Dark Knight Rises	Memento
	✓		✓	✓	
		✓			✓
	✓	✓	✓		
				✓	✓

≈

1	.1
-1	0
.2	-1
.1	1

.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

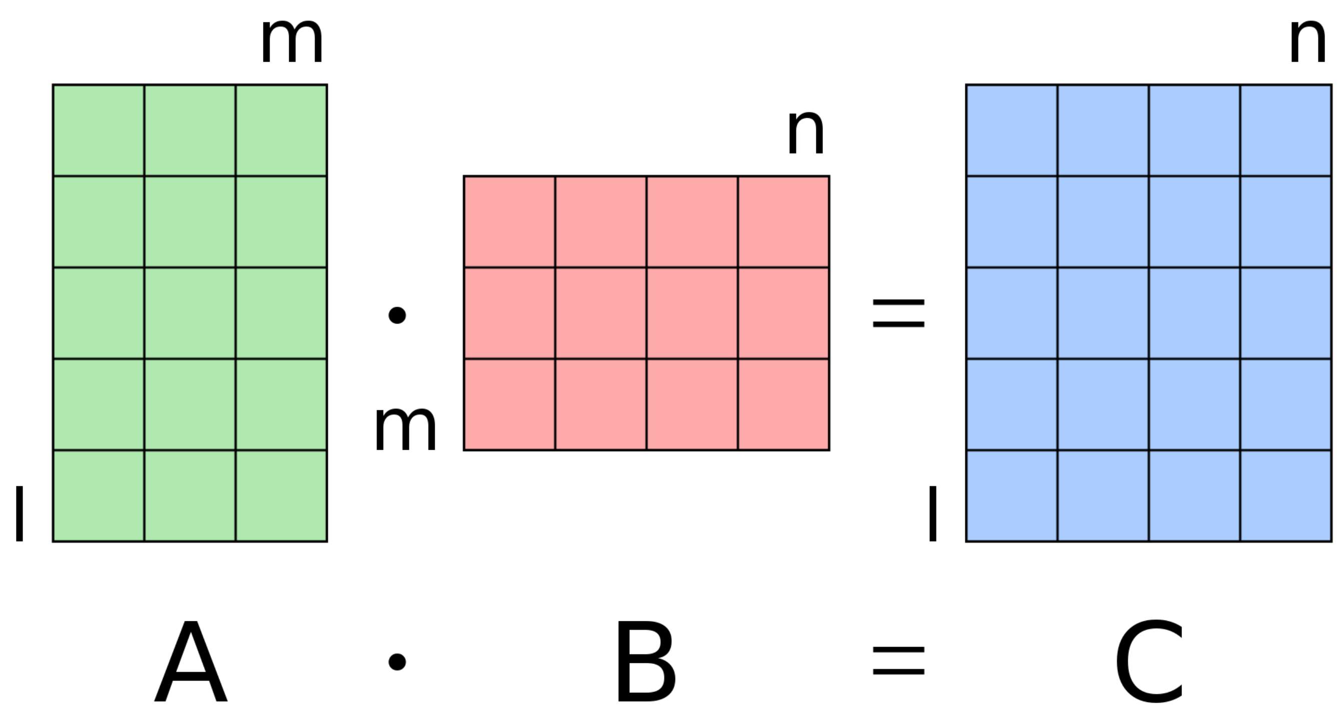
.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

Matrix Multiplication

Multiplying an $I \times m$ matrix by a $m \times n$ matrix gives us an $I \times n$ matrix.

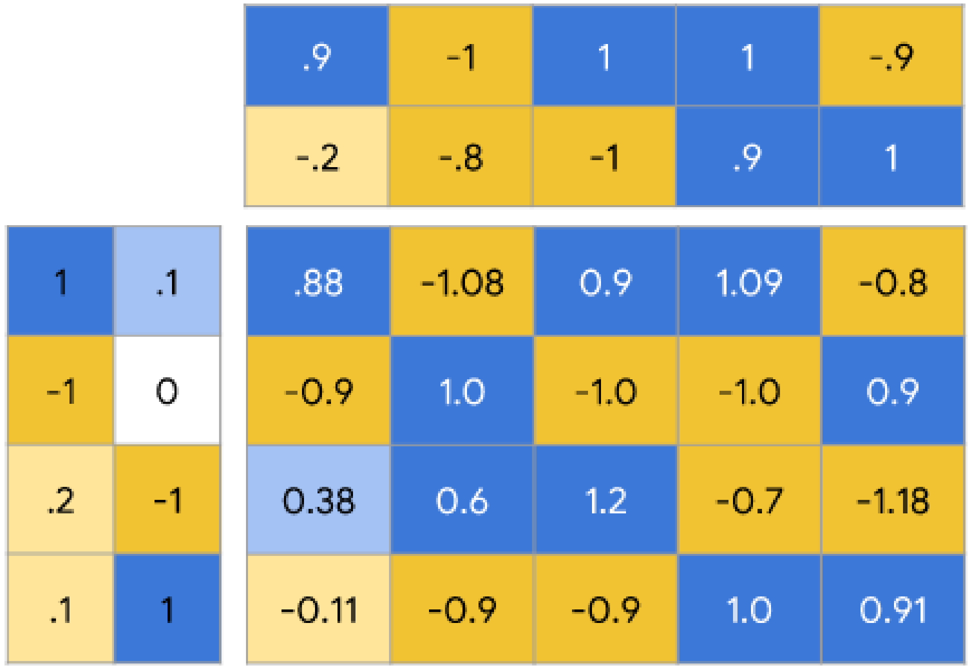
In the example, *I* is the number of users, *n* is the number of items and *m* is the number of **latent** (hidden) **factors** which we are using in our embedding.



Matrix Factorization

Reminder: Factorization is breaking a number down into smaller numbers that multiplied together give you the original number – e.g., factors of 15 include 3 and 5.

Matrix factorization involves finding two lower dimensional matrices which when multiplied give you the original matrix (or in practise, a good approximation of the original matrix.)



.9	-1	1	1	-.9
-.2	-.8	-1	.9	1

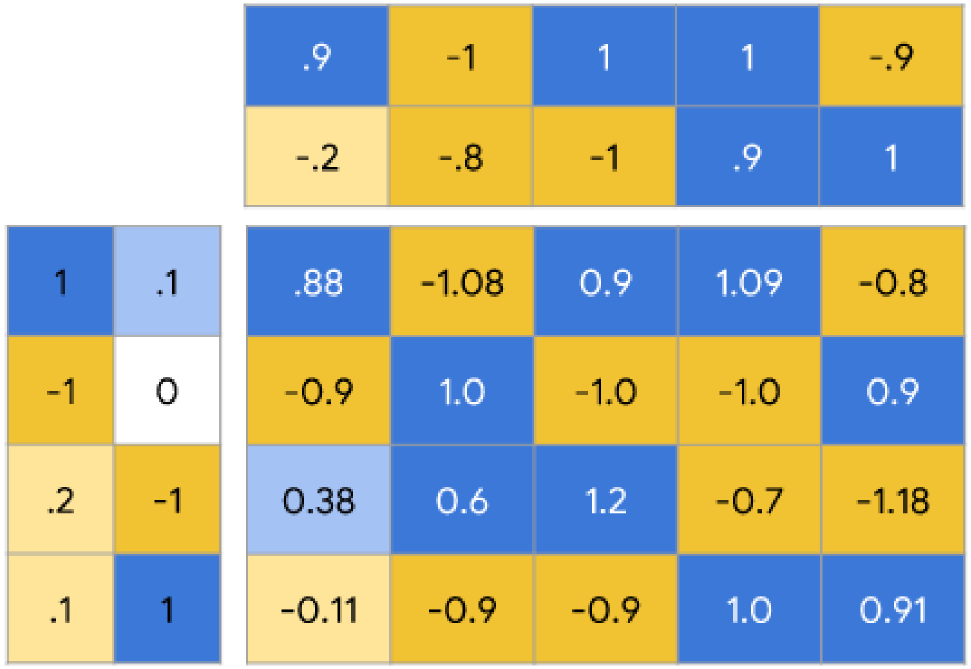
1	.1
-1	0
.2	-1
.1	1

.88	-1.08	0.9	1.09	-0.8
-0.9	1.0	-1.0	-1.0	0.9
0.38	0.6	1.2	-0.7	-1.18
-0.11	-0.9	-0.9	1.0	0.91

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

Matrix Factorization

- We represent users and items in a lower-dimensional space, where similar users and similar items have similar representations.
- Example: Instead of storing a massive 230M × 17,000 user-item matrix (3.9 trillion cells!), we learn compact k-dimensional embeddings (e.g., k=50).
- Where no data exists for a user-item combination, multiplying the user and item embeddings estimates the missing rating, allowing us to make recommendations.



$$\begin{bmatrix} .9 & -1 & 1 & 1 & -.9 \\ -.2 & -.8 & -1 & .9 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & .1 \\ -1 & 0 \\ .2 & -1 \\ .1 & 1 \end{bmatrix} \quad \begin{bmatrix} .88 & -1.08 & 0.9 & 1.09 & -0.8 \\ -0.9 & 1.0 & -1.0 & -1.0 & 0.9 \\ 0.38 & 0.6 & 1.2 & -0.7 & -1.18 \\ -0.11 & -0.9 & -0.9 & 1.0 & 0.91 \end{bmatrix}$$

Image from Google, licensed under Creative Commons: https://developers.google.com/machine-learning/recommendation/collaborative/basics

Alternating Least Squares (ALS)

- To carry out matrix factorization we can use an algorithm called Alternating Least Squares (ALS)
- The goal is to find two matrices such that their product is approximately equal to the original matrix of users and products
- The algorithm tries to minimise an **objective function.** In simple terms, this provides a measure of the error (i.e., the difference between the actual values in the user-item matrix and the values we get by multiplying the user and item embedding matrices.) We'll come back to this on the next slide

Objective Function

- The objective function which the ALS algorithm tries to minimise is:

$$L = \underbrace{\sum_{u,i \in S} (r_{ui} - \mathbf{x}_u \mathbf{y}_i^T)^2}_{\text{This is the sum of the squared errors for all ratings in the dataset}} + \underbrace{\lambda \left(\sum_u \|\mathbf{x}_u\|^2 + \sum_i \|\mathbf{y}_i\|^2 \right)}_{\text{This is a regularization term which we introduce to prevent overfitting. We control this with the hyperparameter } \lambda \text{ (lambda)}}$$

ALS Algorithm

- ALS alternates between updating user embeddings and item embeddings.
- Given an initial random guess for user and item embeddings:
  - Fix user embeddings and solve for item embeddings.
  - Fix item embeddings and solve for user embeddings.
  - Repeat until convergence.
- The objective function minimises the squared error (already mentioned in Slide 33).
- Why alternating?
  - A direct solution is infeasible due to the size and sparsity of real-world data.

Overfitting in ALS

- Overfitting occurs when a model learns patterns that are too specific to the training data.
- This makes the model perform well on training data but poorly on new (unseen) data.
- Causes of overfitting: Too many parameters (complex models) and not enough data.
- Regularization (e.g., the lambda parameter in ALS) helps prevent overfitting

Evaluating our Recommender

- We can evaluate our recommender system by comparing the ratings it predicts against the true ratings (where we have them)
- To do this we can use a metric called the **Root Mean Square Error:**

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}}$$

Where y_i is the *i*-th rating and $\hat{y}_i$ is the prediction for this rating.

Evaluating our Recommender Example

y	$\hat{y}$	$(\hat{y} - y)^2$
4	1.9	4.41
3	3.2	0.04
5	6.1	1.21
2	1.1	0.81
TOTAL		6.47

So the RMSE is the square root of (6.47 / 4) = 1.27

The RMSE is in the same unit as y so we can interpret it as the average difference between the values predicted and the actual values.

ALS in MLlib

- ALS is implemented for us in Spark MLlib so we will be using this implementation in the workshop this afternoon.
- We have a number of hyperparameters we can set, including:
    - rank – the number of latent factors (the number of dimensions in our embedding)
    - implicitPrefs – whether we are dealing with explicit ratings or implicit feedback (explicit is the default)
    - coldStartStrategy – how to deal with users or items which were not in the training set when it comes to making predictions (we discuss the cold start problem shortly)
    - regParam – specifies the regularization parameter, which we use to prevent overfitting

Collaborative Filtering Advantages

- We don’t need any domain knowledge to train the model because the embeddings are automatically learned
- The model can help users discover new interests – the model can recommend items because similar users are interested in that item
- The system only needs the feedback matrix to generate the model – i.e., we don’t need any contextual features

Collaborative Filtering Disadvantages

- You need existing data on items or users for a collaborative filtering model
- If an item or user is not seen at the point when the model is being trained, it cannot create an embedding for it. The model can therefore not be queried for that item or user.
  - This is known as the **cold start problem**

Cold Start Problem

- ALS requires users and items to have interaction history, so new users or items can't be embedded.
- Strategies:
  - Use content-based features (e.g., metadata for new movies).
  - Hybrid models (combine ALS with deep learning).
  - Use default embeddings for new items/users.

Candidate Generation & Ranking

- In practice, a recommender system might use multiple candidate generators (e.g., a content-based filtering and collaborative filtering models)
- The recommendation candidates from multiple sources can be combined into a pool of candidates and these can then be scored or ranked by another model to decide which ones to display and in which order

Example Pipeline

An example pipeline may consist of the following stages:

1. Data Collection: Capture user interactions (clickstream data).
2. Feature Engineering: Extract relevant features (e.g., time spent on content) as implicit ratings.
3. Batch Processing: Use Spark SQL to aggregate data and train an ALS model.
4. Batch Predictions: Store recommendations in a NoSQL database for fast retrieval.
5. Streaming Processing: Use Spark Streaming to process new user interactions which will then be used for periodic re-training

Quick Quiz!

www.menti.com

5680 5331

Exploring Further…

For more information on how Apache Spark is used in industry for recommendations:

https://www.youtube.com/watch?v=cxtDrBJjTBY

“Learning to Rank with Apache Spark: A Case Study in Production ML with Adam Davidson & Anna Bladzichz”

Learning Outcomes

By the end of this lecture and workshop, you will be able to:

1. Describe the basic principles behind content based filtering and collaborative filtering recommender systems
2. Explain how matrix factorization is used in a collaborative filtering recommender system to find item and user embeddings
3. Use Apache Spark to apply ALS to build and test a recommender model