Movie Recommendation System on Apache Spark

Raghad Rowaida

Why recommendation system

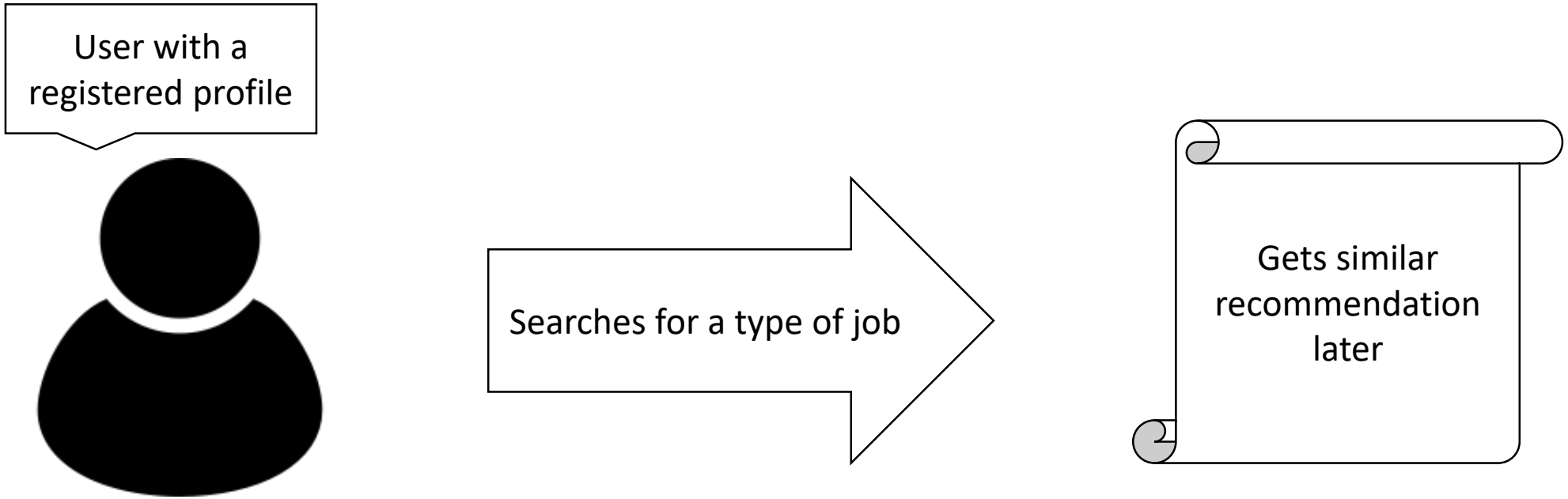
- Internet is filled with data
- Takes time to search for desired item
- Recommendation system makes searching easier, and keep user hooked to the system
- Saves a lot of time

Recommendation System Principles

- Content based recommendation system
- Collaborative Filtering based recommendation system
- Hybrid recommendation system

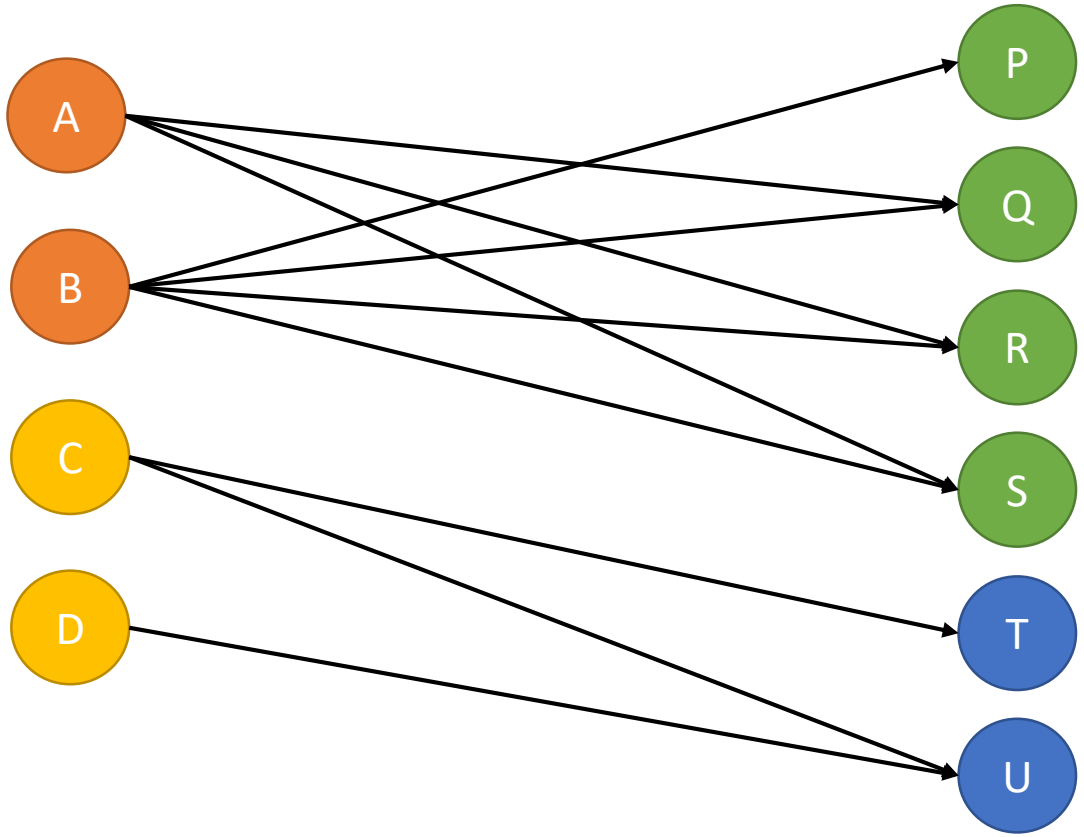
Content based recommendation system

- Analyze a user's data and history to make similar type of recommendation.



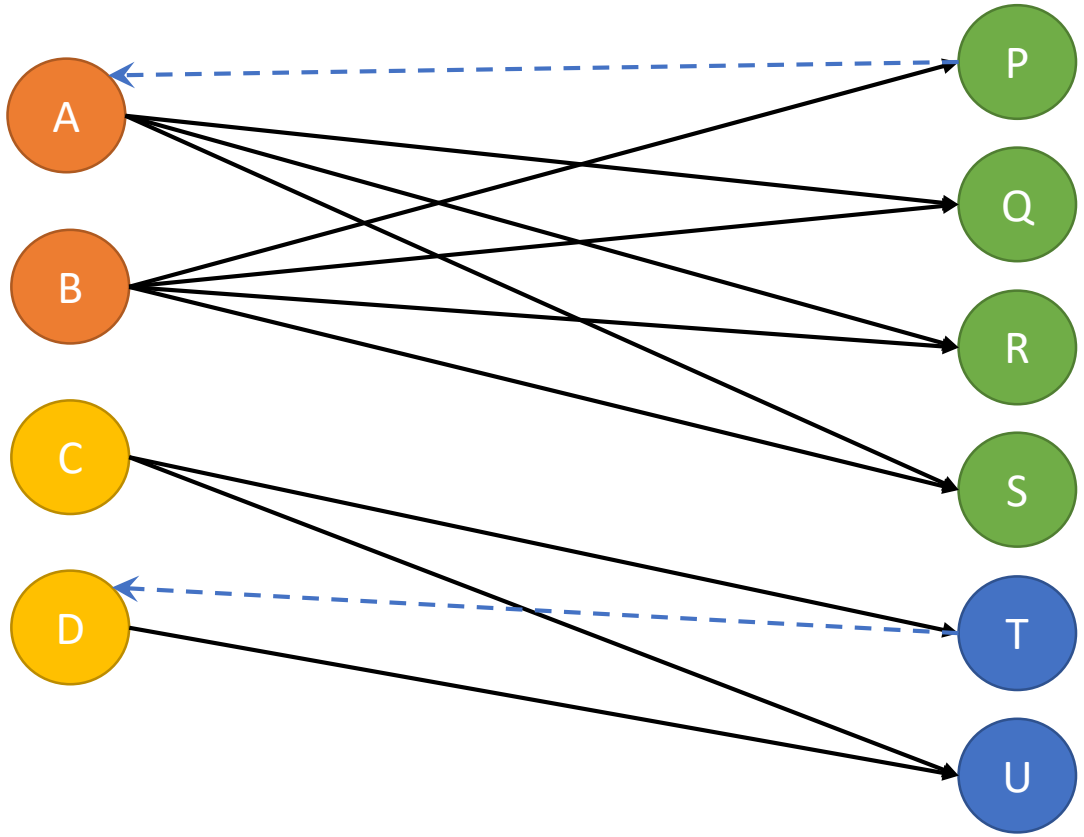
Collaborative Filtering recommendation system

- Use user's rating to find out people of similar taste. Recommend a new item which received high rating from people of similar taste.



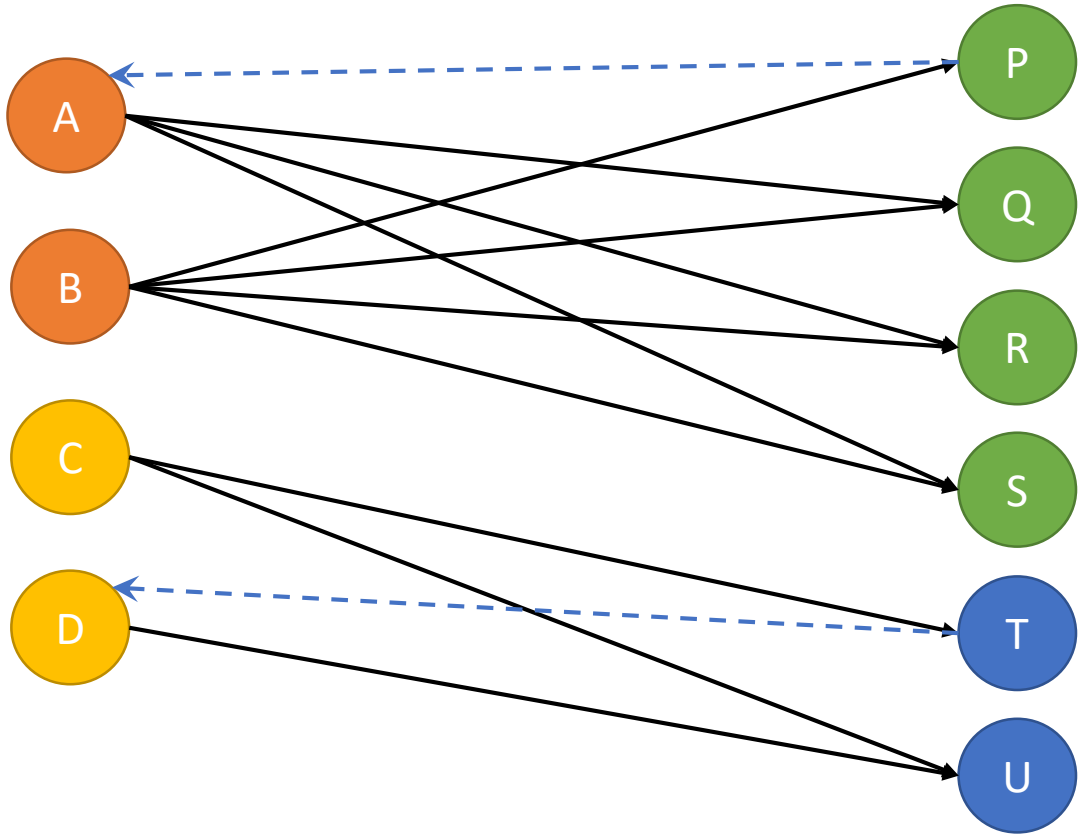
Collaborative Filtering recommendation system

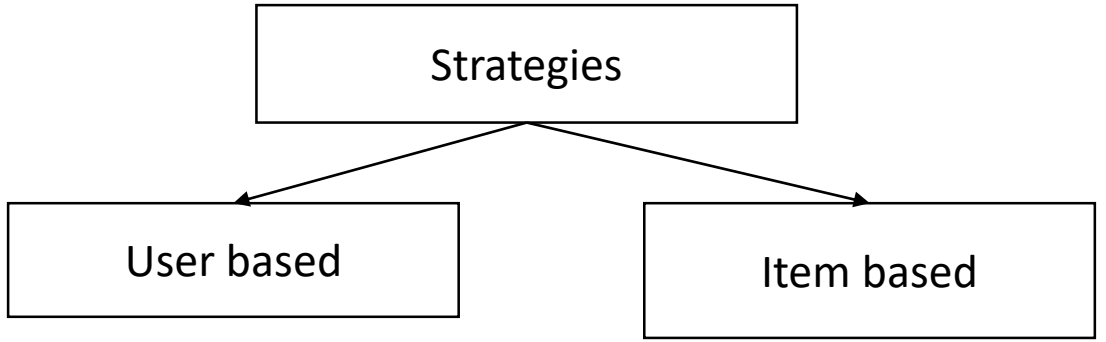
- Use user's rating to find out people of similar taste. Recommend a new item which received high rating from people of similar taste.



Collaborative Filtering recommendation system

- Use user's rating to find out people of similar taste. Recommend a new item which received high rating from people of similar taste.





Hybrid recommendation system

- Applying a combination of both approaches for recommendation.

Appropriate choice for movie recommender

- Content based

Appropriate choice for movie recommender

- ~~Content based~~
- Collaborative filtering based
  - Item based
  - User based

Appropriate choice for movie recommender

- ~~Content based~~
- Collaborative filtering based
  - ~~Item based~~
  - User based

Collaborative Filtering: Algorithm

1. Represent the dataset as a Matrix of U x I dimension where U – number of user, I number of item (movies) and cell (u,i) – User u's rating of item i.
2. Apply Matrix Factorization technique of Machine Learning to decompose the input matrix into user-rating, and item-rating submatrix.
3. Recommend item to user based on the latent rating above a threshold.

Matrix Factorization: What is it?

	M1	M2	M3	M3
U1		4.5	2	
U2	4		3.5	
U3		5		2
U4		1	4	1

=

	M1	M2	M3	M4
LF1	1.5	1.2	1	0.8
LF2	1.7	0.6	1.1	0.4

✖

	LF1	LF2
U1	1.2	0.8
U2	1.4	0.9
U3	1.5	1
U4	1.2	0.8

$$\tilde{r}_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i}$$

Matrix Factorization: Why?

userId	4	5	10	14	15	18	19	26	31	34	...	283199	283204	283206	283208	283210	283215	283219	283222	283224
movieId																				
1	4.0	NaN	5.0	4.5	4.0	NaN	NaN	NaN	5.0	NaN	...	5.0	NaN	NaN	4.5	NaN	4.0	4.0	NaN	NaN
2	4.0	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	4.0
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	2.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN
6	4.5	NaN	NaN	NaN	NaN	3.0	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	5.0	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	4.0	NaN	NaN	NaN	NaN	3.0	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0
11	3.5	NaN	NaN	NaN	NaN	NaN	4.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN
12	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
13	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
14	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0
15	NaN	NaN	NaN	NaN	NaN	NaN	3.0	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Matrix Factorization: How?

- Select number of latent factor
- Apply gradient descent (or any other training routine) to minimize the objective function:

$$\underset{H,W}{\arg\min} \|R - \tilde{R}\|_{\mathrm{F}} + \alpha\|H\| + \beta\|W\|$$

Matrix Factorization: Alternating Least Square

- Implemented in Apache Spark ML
- Runs in parallel

Alternating Least Square (ALS)

- ALS minimizes **two loss functions alternatively**
- First holds user matrix fixed and runs gradient descent with item matrix; then it holds item matrix fixed and runs gradient descent with user matrix
- ALS runs its gradient descent in **parallel** across multiple partitions of the underlying training data from a cluster of machines

Dataset

- MovieLens 100K Dataset
- 100,000 ratings (1-5) from 943 users on 1682 movies
- Each user has rated at least 20 movies.

Implementation

- OpenStack configuration with 32 GB disk space, 8 GB RAM, 4 vCPUper node
- Used Apache spark 2.4.4 version
- Used special libraries, such as: org.apache.spark.mllib.recommendation
- Spark cluster model: single master, and single slave
- Average time to run ALS for 100k rating data: 10.86 second

Sample output

- Top 10 movie recommendation for user ID 789 from the ml-100K dataset.

```
(Godfather, The (1972),5.0)
(Trainspotting (1996),5.0)
(Dead Man Walking (1995),5.0)
(Star Wars (1977),5.0)
(Swingers (1996),5.0)
(Leaving Las Vegas (1995),5.0)
(Bound (1996),5.0)
(Fargo (1996),5.0)
(Last Supper, The (1995),5.0)
(Private Parts (1997),4.0)
19/12/04 02:01:13 INFO SparkContext: Invoking stop() from shutdown hook
```

Participation session

1. What type of recommendation algorithm ‘job portals’ uses, and why?

2. Why would we use user based CF, instead of item based CF?

3. How Matrix Factorization enhance CF algorithm performance?

Thank you