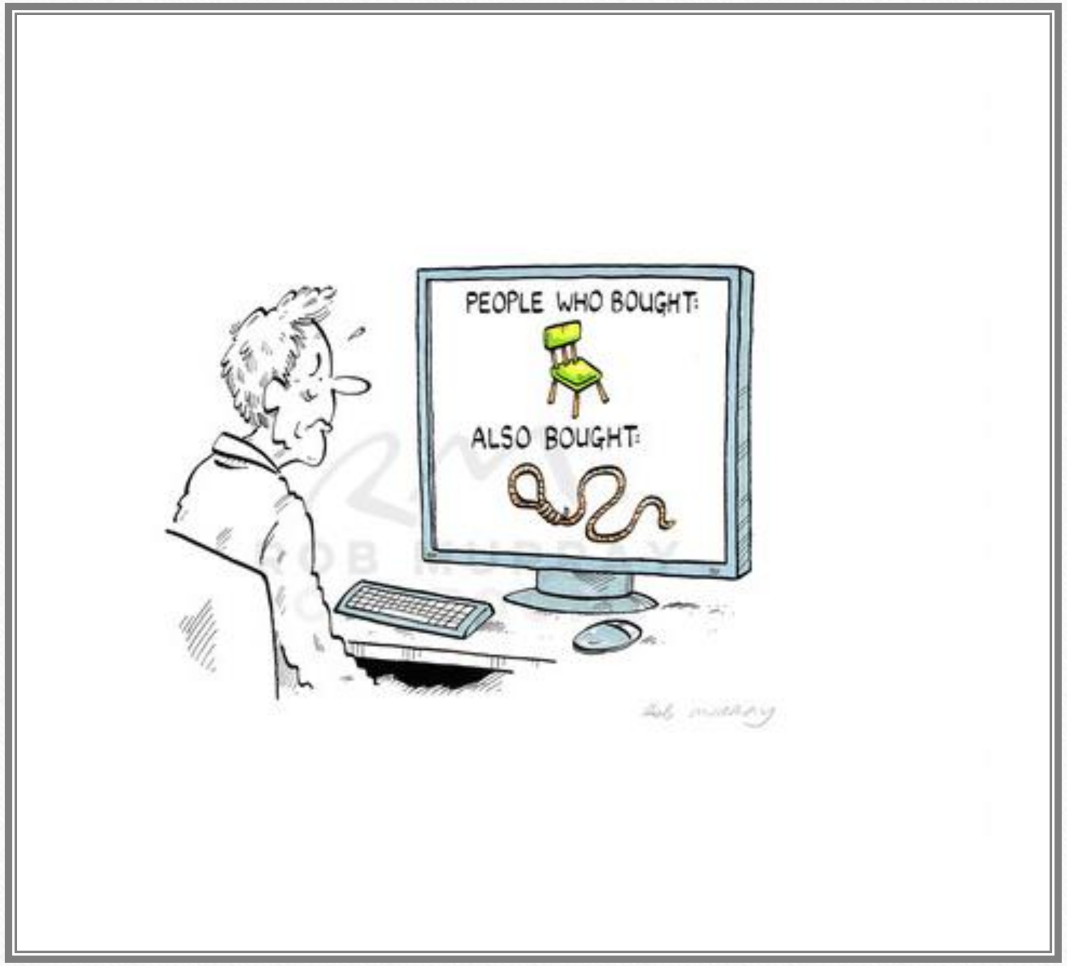


Source: Google

Movie Recommendation System Using Scala

Dinesh Varma Indukuri

MS Business Analytics, 2019

The University of Texas at Dallas

Source: Google

Types of Recommendation Systems

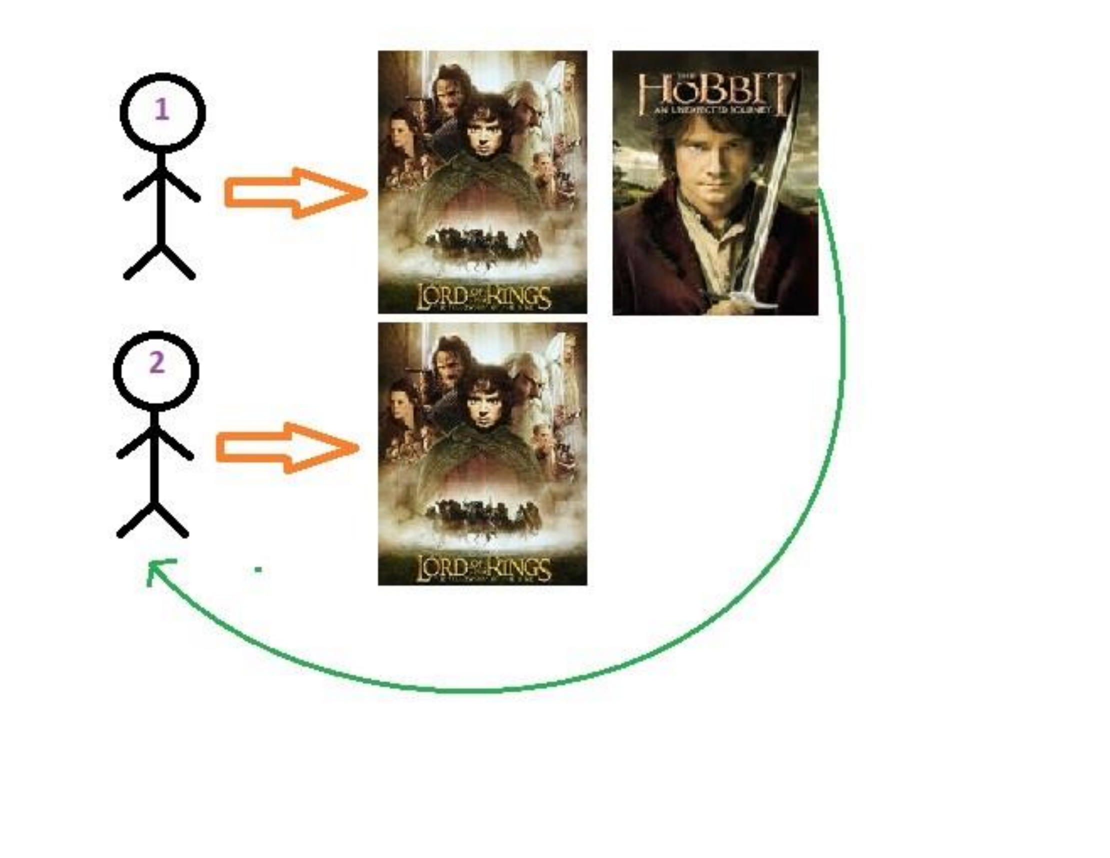
- Collaborative Recommender System
- Content based Recommender System
- Demographic based Recommender System
- Utility based Recommender System
- Knowledge based Recommender System
- Hybrid Recommender System

Collaborative Recommender System

A.) User Based Collaborative Filtering

- Hard Computations, Costly computational power for comparing and finding similarities

- Habits of people can be changed. Therefore making correct and useful recommendation can be hard in time



Source: Kaggle.com

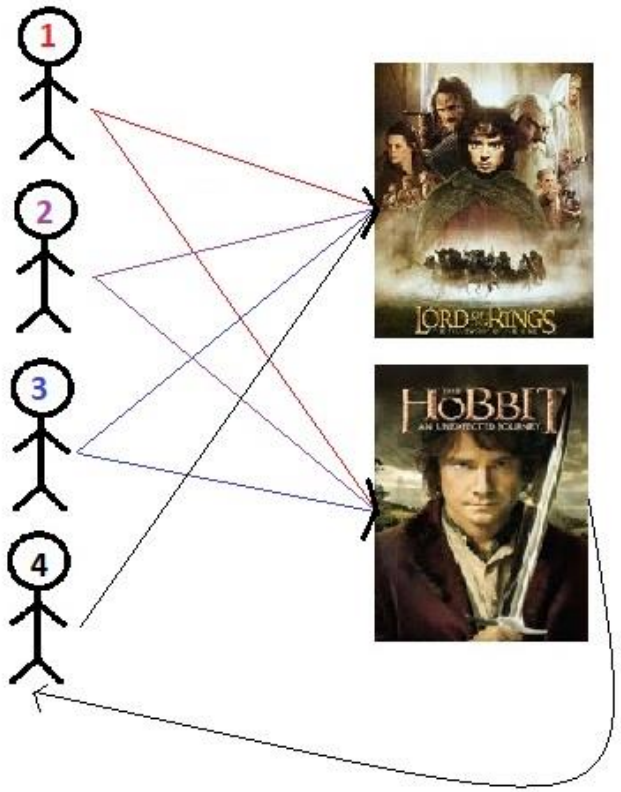
Collaborative Recommender System

B.) Item Based Collaborative Filtering

- Instead of finding relationship between users, used items like movies or stuffs are compared with each others.

- Similarities between lord of the rings and hobbit movies because both are liked by three different people.

- If the similarity is high enough, we can recommend hobbit to other people who only watched lord of the rings movie



Source: Kaggle.com

How to recommend movies?

a.) Correlation

- Recommend "Headless Body in Topless Bar (1995)" movie to people who watched "Bad Boys (1995)"

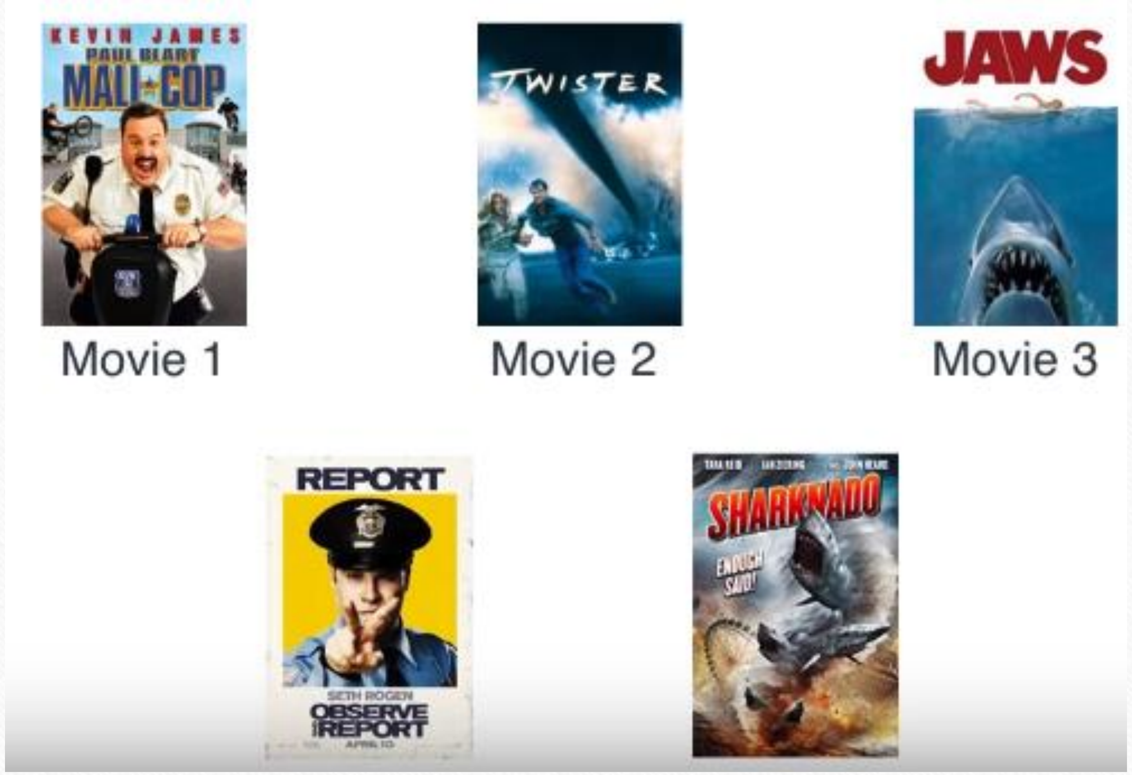
Title	Correlation
Bad Boys(1995)	1.000
Headless body in Topless Bar (1995	0.7237
Last Summer in the Hamptons (1995)	0.6075
Two Bits (1995)	0.5070
Shadows (Cinenie) (1998)	0.49418

title	Ace Ventura: When Nature Calls (1995)	Across the Sea of Time (1995)	Amazing Panda Adventure, The (1995)	American President, The (1995)	Angela (1995)	Angels and Insects (1995)	Anne Frank Remembered (1995)	Antonia's Line (Antonia) (1995)	Assassins (1995)	Babe (1995)	Bad Boys (1995)
userId											
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	3.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	5.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
8	1.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
10	NaN	NaN	NaN	4.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
11	3.5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3.0

How to recommend movies?

b.) Matrix Factorization

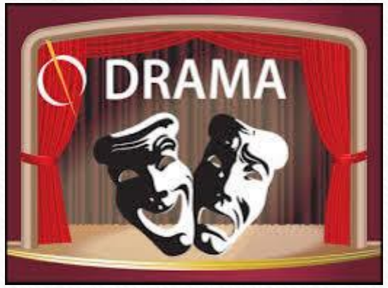
	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

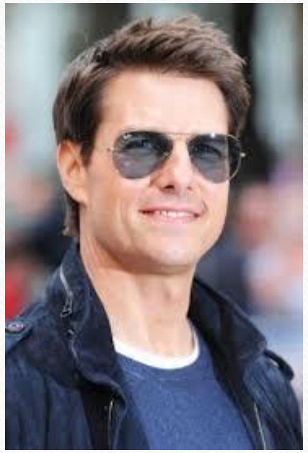


Reference: Luis Serrano

Matrix Factorization

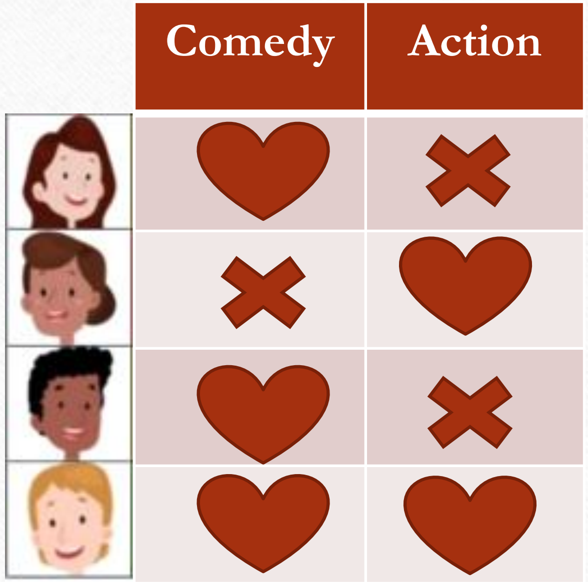
Factorization: 6 * 4 = 24





Matrix Factorization

Feature	M1	M2	M3	M4	M5
Comedy	3	1	1	3	1
Action	1	2	4	1	3



	M1	M2	M3	M4	M5
	3	1	1	3	1
	1	2	4	1	3
	3	1	1	3	1
	4	3	5	4	4

Reference: Luis Serrano

Data

Data Source: Movie Lens Dataset (20 M)

- Movie.csv
- Rating.csv

Environment

Data Bricks Community Edition

- Spark 2.4.4
- Scala 2.11
- Cluster: Memory = 6 GB, Cores = 0.88

Movie

movieId	title	genres
1	Toy Story (1995)	Adventure|Animati...
2	Jumanji (1995)	Adventure|Childre...
3	Grumpier Old Men ...	Comedy|Romance
4	Waiting to Exhale...	Comedy|Drama|Romance
5	Father of the Bri...	Comedy
6	Heat (1995)	Action|Crime|Thri...
7	Sabrina (1995)	Comedy|Romance
8	Tom and Huck (1995)	Adventure|Children
9	Sudden Death (1995)	Action
10	GoldenEye (1995)	Action|Adventure|...

Rating

userId	movieId	rating	timestamp
1	2	3.5	2005-04-02 23:53:47
1	29	3.5	2005-04-02 23:31:16
1	32	3.5	2005-04-02 23:33:39
1	47	3.5	2005-04-02 23:32:07
1	50	3.5	2005-04-02 23:29:40
1	112	3.5	2004-09-10 03:09:00
1	151	4.0	2004-09-10 03:08:54
1	223	4.0	2005-04-02 23:46:13
1	253	4.0	2005-04-02 23:35:40
1	260	4.0	2005-04-02 23:33:46

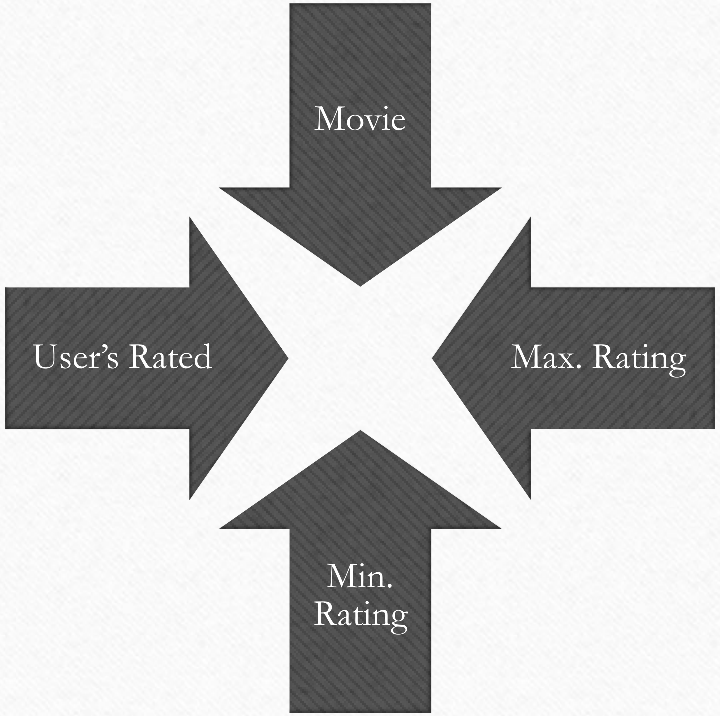
Data Statistics

```
/* Count of ratings, users, movies */
val numRatings = rating.count()
val numUsers = rating.select(rating.col("userId")).distinct().count()
val numMovies = rating.select(rating.col("movieId")).distinct().count()
println("Got " + numRatings + " ratings from " + numUsers + " users on " + numMovies + " movies.")
```

Ratings	Users	Movies
200,00,263	1,38,493	26,744

Inferential Statistics

```
/* Get the max, min ratings along with the count of users who have rated a movie */
val results = spark.sql("select movies.title, movierates.maxr, movierates.minr, movierates.cntu "
+ "from(SELECT ratings.movieId,max(ratings.rating) as maxr,"
+ "min(ratings.rating) as minr,count(distinct userId) as cntu "
+ "FROM ratings group by ratings.movieId) movierates "
+ "join movies on movierates.movieId=movies.movieId "
+ "order by movierates.cntu desc")
results.show()
```



title	maxr	minr	cntu
Pulp Fiction (1994)	5.0	0.5	67310
Forrest Gump (1994)	5.0	0.5	66172
Shawshank Redempt...	5.0	0.5	63366
Silence of the La...	5.0	0.5	63299
Jurassic Park (1993)	5.0	0.5	59715
Star Wars: Episod...	5.0	0.5	54502
Braveheart (1995)	5.0	0.5	53769
Terminator 2: Jud...	5.0	0.5	52244
Matrix, The (1999)	5.0	0.5	51334
Schindler's List ...	5.0	0.5	50054
Toy Story (1995)	5.0	0.5	49695
Fugitive, The (1993)	5.0	0.5	49581
Apollo 13 (1995)	5.0	0.5	47777
Independence Day ...	5.0	0.5	47048
Usual Suspects, T...	5.0	0.5	47006
Star Wars: Episod...	5.0	0.5	46839
Batman (1989)	5.0	0.5	46054
Star Wars: Episod...	5.0	0.5	45313

Building the model

```
val als = new ALS().
  setMaxIter(5).
  setRegParam(0.01).
  setUserCol("userId").
  setItemCol("movieId").
  setRatingCol("rating")
println(als.explainParams())
```

Important hyper parameters in Alternating Least Square (ALS)

- maxIter : Max. interations (defaults to 10)
- Rank: number of latent/hidden factors (defaults to 10)
- regParam: Regularization Parameter (λ) (defaults to 1.0)

Model Prediction

```
/* Prediction */
val predictions = model.transform(testing)
predictions.show(10)
```

```
+------+-------+------+-------------------+----------+
|userId|movieId|rating|          timestamp|prediction|
+------+-------+------+-------------------+----------+
| 96393|    148|   3.0|2000-09-28 19:41:30| 2.9136493|
| 20132|    148|   3.0|2002-05-19 02:36:33| 3.1030006|
| 22884|    148|   3.0|1999-12-11 21:31:08|  2.344263|
| 10303|    148|   3.0|1999-10-25 13:16:01| 3.2844634|
| 44979|    148|   3.0|1996-04-29 11:43:40| 2.0735006|
| 13170|    148|   3.0|1998-01-23 03:08:11|0.69044816|
| 32882|    148|   3.0|1996-07-06 23:27:53|  2.788623|
|  5585|    148|   3.0|1996-06-05 02:11:17| 3.9919806|
| 36445|    148|   4.5|2014-12-23 18:15:55| 2.3047786|
| 94994|    148|   4.0|1996-06-01 20:44:37| 3.7862895|
+------+-------+------+-------------------+----------+
```

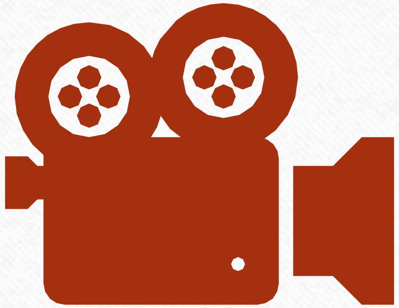
Model Evaluation

```
/* Model Evaluation */

import org.apache.spark.ml.evaluation.RegressionEvaluator
val evaluator = new RegressionEvaluator().
  setMetricName("rmse").  // root mean squared error
  setLabelCol("rating").
  setPredictionCol("prediction")
val rmse = evaluator.evaluate(predictions)
println(s"Root-mean-square error = $rmse")
```

```
Root-mean-square error = 0.8127186759177553
```

Reference: Collaborative Filtering



Top 10 movie recommendations for each user

```
-------------------------------------------------------------------------------+
|userId|recommendations
|
+------+-------------------------------------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------+
|148   |[[128366, 12.673134], [54067, 12.200273], [106503, 10.693678], [5402, 10.441715], [87884, 10.206536], [50482, 9.962639], [106624, 9.9
04217], [94011, 9.634692], [5270, 9.558779], [82261, 9.410857]]          |
|463   |[[96255, 11.256444], [128366, 9.451611], [112907, 9.414341], [74406, 9.414341], [116704, 9.414341], [86451, 9.201364], [73529, 9.1022
49], [115008, 8.596032], [74159, 8.462144], [78196, 8.407199]]           |
|471   |[[96255, 10.7220125], [69442, 9.918955], [34373, 9.503464], [92169, 9.364761], [77412, 9.211686], [60356, 8.864539], [70804, 8.86178
8], [74159, 8.825016], [70806, 8.437598], [76873, 8.356776]]              |
```

Reference: Collaborative Filtering

Any Queries?