Week 8

FIT5202 Big Data Processing

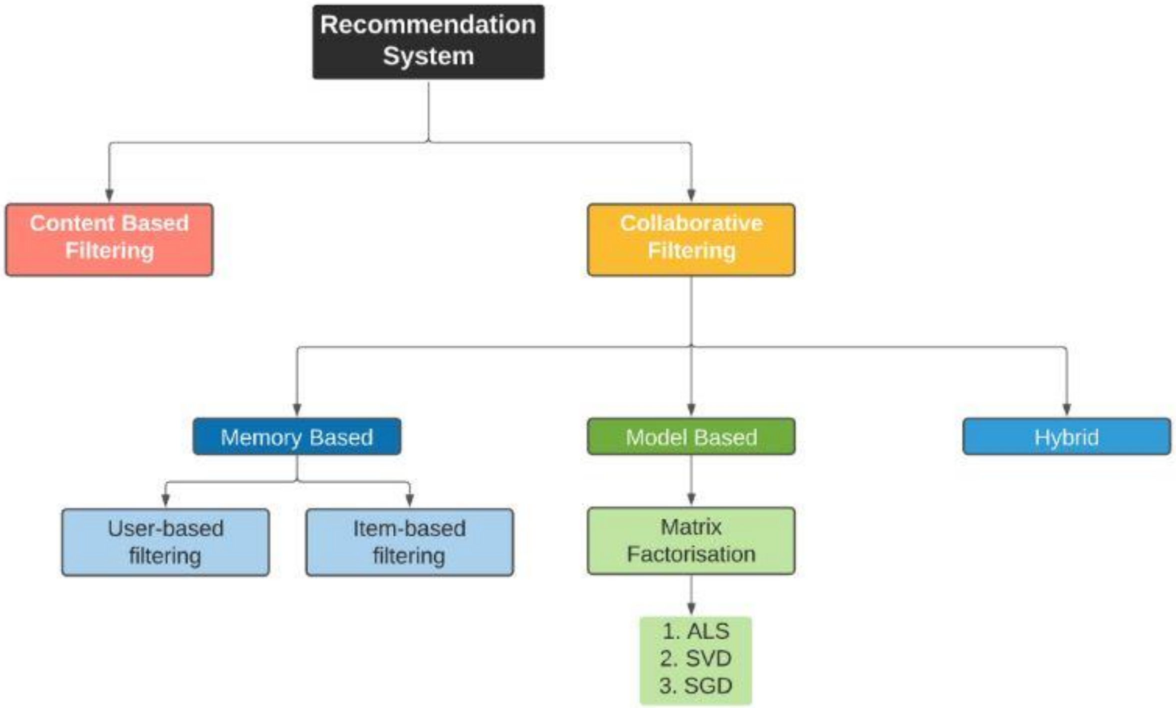
Collaborative Filtering using ALS

Week 8 Agenda

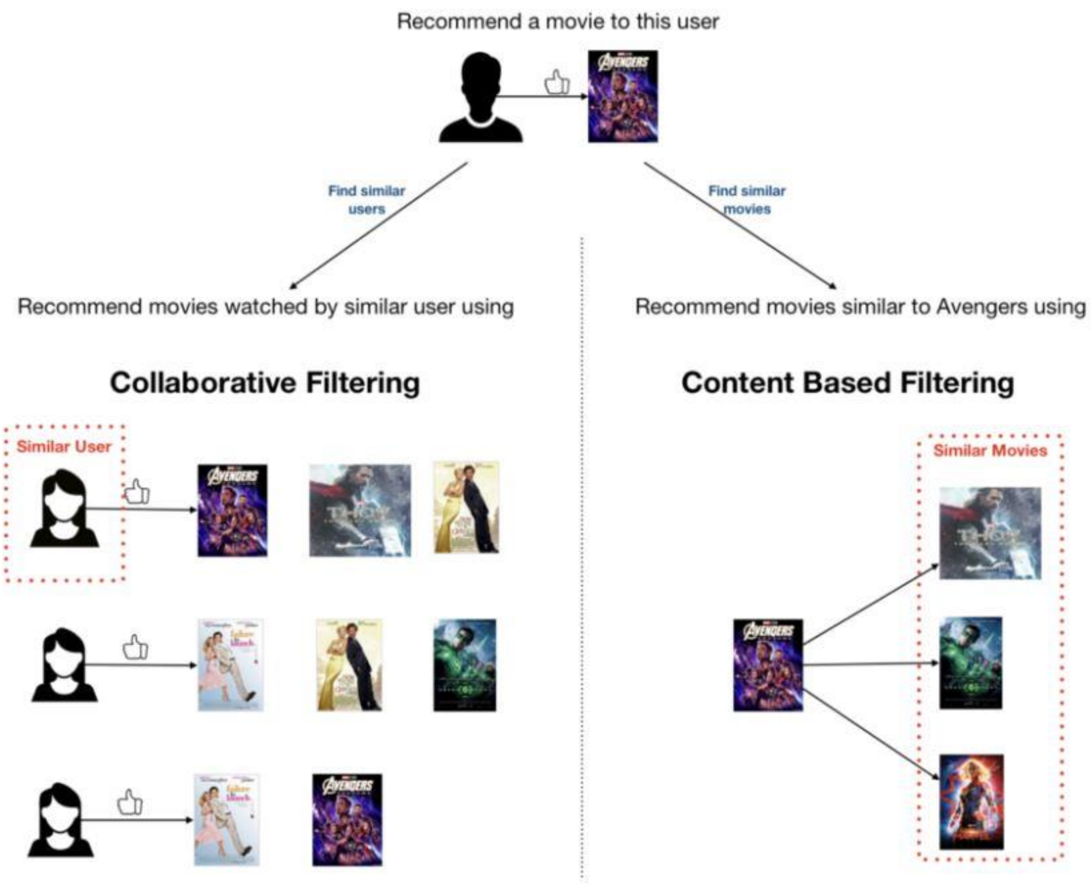
- Week 7 Review
    - K-means clustering
    - Model Selection
    - Model Persistence
- Collaborative Filtering
- Use case : Music Recommendation

SETU FEEDBACK

Recommender Approaches

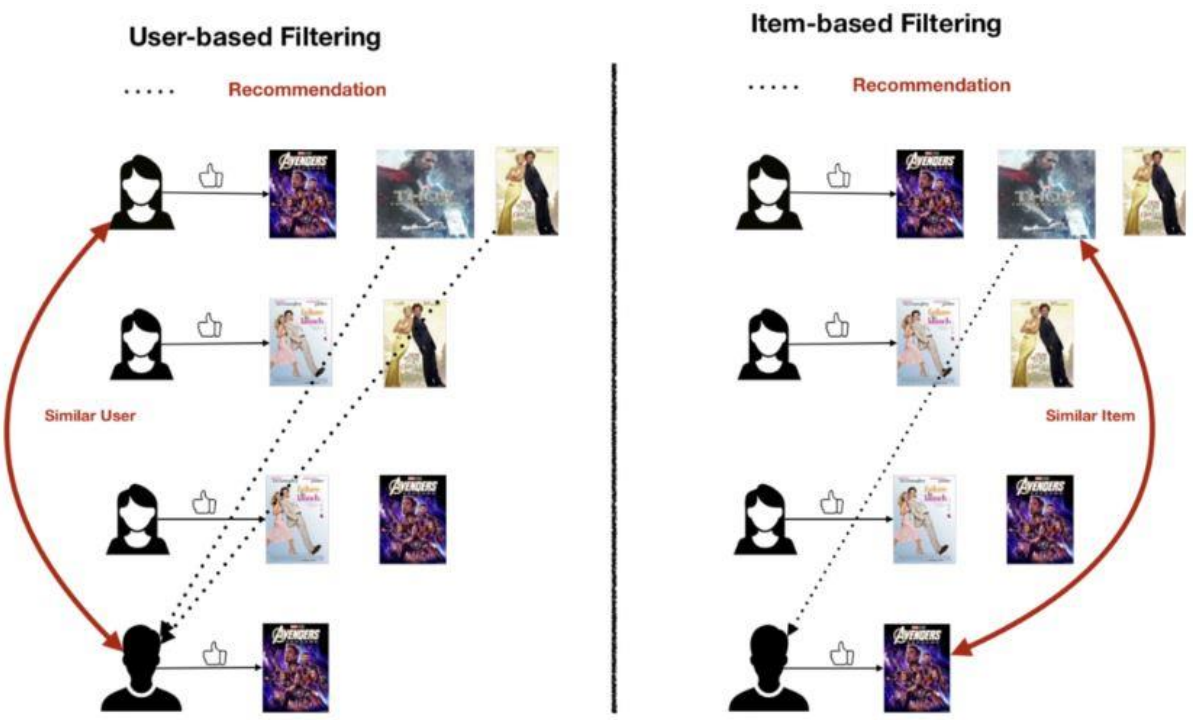


https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679



https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679

Memory-Based Approach

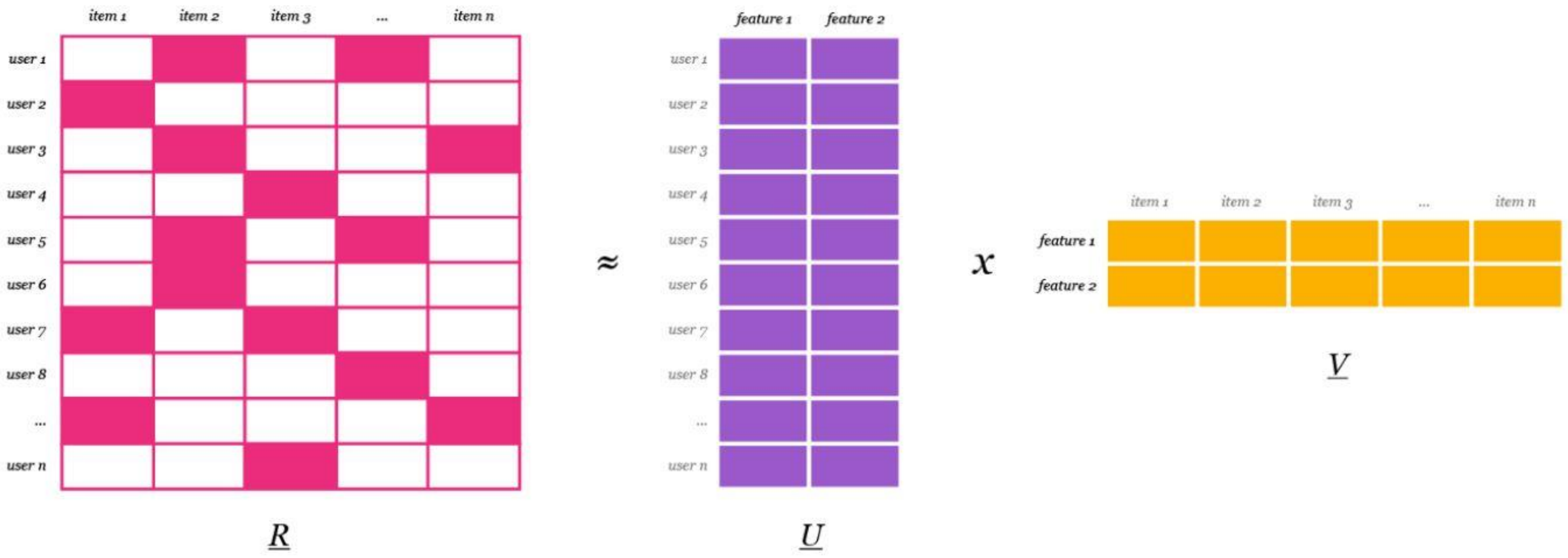


https://towardsdatascience.com/build-recommendation-system-with-pyspark-using-alternating-least-squares-als-matrix-factorisation-ebe1ad2e7679

Implicit vs Explicit Feedback

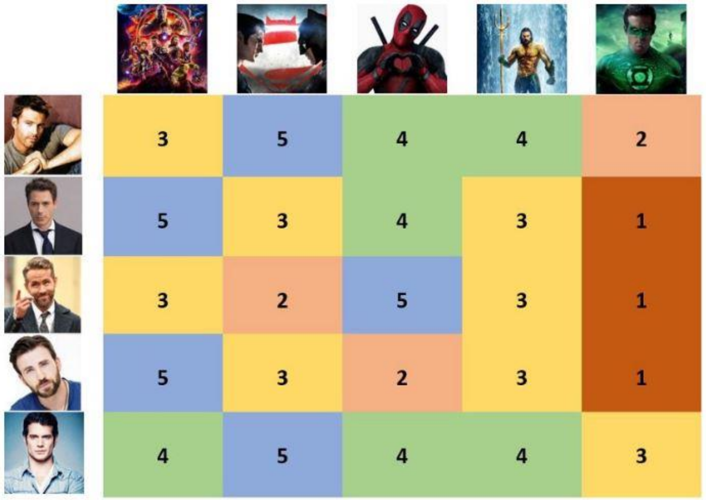
- **Explicit** :
  - when we have **some sort of Rating** (i.e. **users provide items' rating explicitly**)
- **Implicit** :
  - data is gathered from user behaviour, e.g. how many times a song is played or a movie is watched.
  - **Advantage** : more data
  - **Disadvantage** : Noisy data, negative preferences are not known

Matrix Factorization

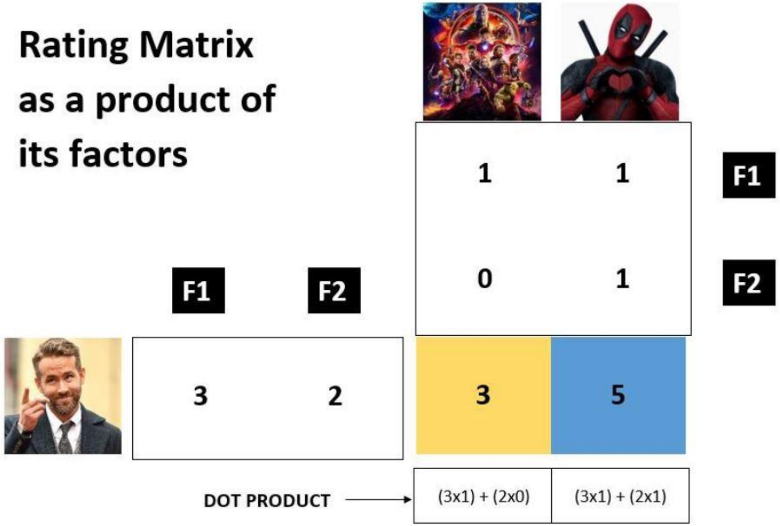


https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

Matrix Factorization – with Explicit Rating



Rating Matrix



https://medium.com/sfu-cspmp/recommendation-systems-collaborative-filtering-using-matrix-factorization-simplified-2118f4ef2cd3

Matrix Factorization – with Implicit Feedback

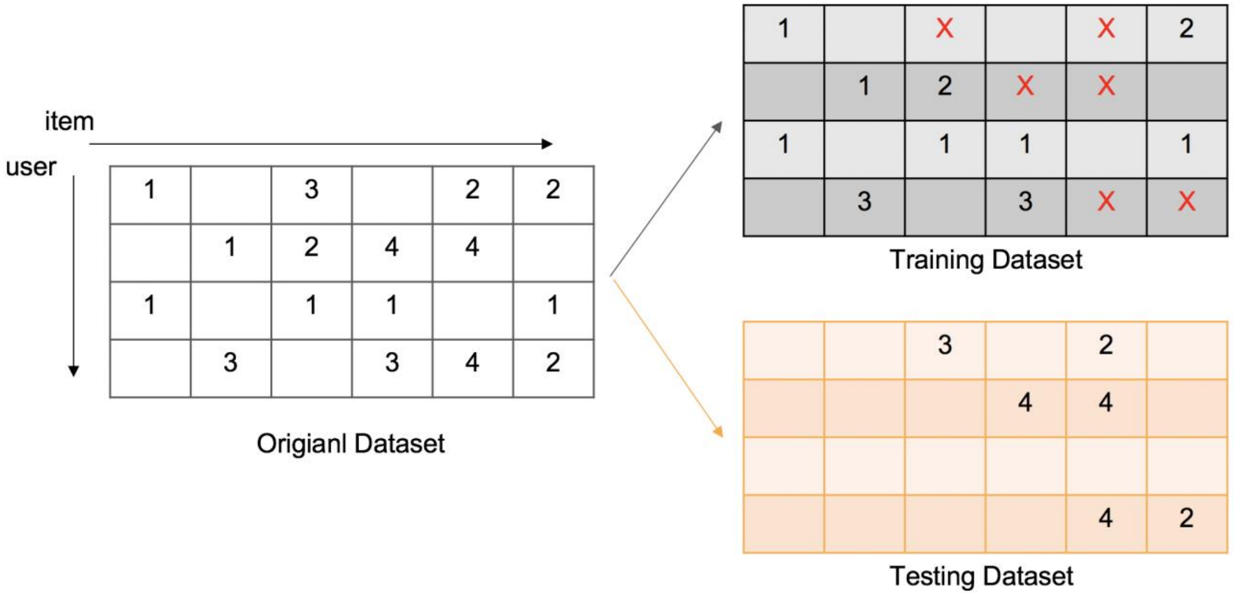
https://developers.google.com/machine-learning/recommendation/collaborative/matrix

Alternating Least Square (ALS) – Implicit Rating

- **Confidence:** $c_{ui} = 1 + \alpha r_{ui}$
  - Quantify confidence of how much user *u* ***likes*** the item *i* of the user from the implicit rating data ***r***
- **Alpha** α
  - The rate (linear scaling) of confidence increases
- **Optimizing alternately to find U, V**
  - Randomly initialize U and V
  - Iterating the following steps:
    - Fixing U → Optimizing V
    - Fixing V → Optimizing U

https://medium.com/radon-dev/als-implicit-collaborative-filtering-5ed653ba39fe

Train/Test Split



Evaluation metrics

For explicit feedback

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (Predicted_i - Actual_i)^2}{N}}$$

RMSE may not be appropriate for measuring prediction for implicit feedback

user_id	artist_id	playcount	prediction
1001440	463	2	-0.6025843
1046559	463	782	0.6918464
1059765	463	793	-0.045939725
1024631	833	5	0.8736501
2010008	833	185	1.0421734
1029563	833	3	0.38790843
2010008	2366	4	0.16086206
2023686	3175	1	0.19943924
2102019	1004021	28	0.043972284
1059765	1007972	21	0.46731347
1024631	1012617	1	0.03206493
1024631	1014191	3	0.38790256
2023686	1014191	3	0.16714399
2023686	1014690	2	0.24097718
1017610	1019303	68	0.42512476
1024631	1028228	1	0.1952564
1059637	1048726	1	0.0038849264
2069889	1048726	2	-0.032158498
1072684	1076507	2	0.97082245
2023686	1084951	1	-0.027778534

only showing top 20 rows

For implicit feedback

ROEM (Rank Ordering Error Metric)

$$\overline{rank} = \frac{\sum_{u,i} r_{ui}^t rank_{ui}}{\sum_{u,i} r_{ui}^t} \qquad (8)$$

Lower values of $\overline{rank}$ are more desirable, as they indicate ranking actually watched shows closer to the top of the recommendation lists. Notice that for random predictions, the expected value of $rank_{ui}$ is 50% (placing i in the middle of the sorted list). Thus, $\overline{rank} \geqslant 50\%$ indicates an algorithm no better than random.

user_id	artist_id	playcount	prediction	percent_rank
1059637	1233770	613	6.226111	0.0
2020513	754	993	4.6966453	2.032520325203252E-4
1072684	1330	54	4.677941	4.065040650406504E-4
1059334	228	12	4.4809046	6.097560975609756E-4
1059637	1000130	19129	3.7980416	8.130081300813008E-4
2069889	1000263	177	3.6094182	0.0010162601626016261
1070641	1004294	2	3.5892177	0.0012195121951219512
1007308	393	12	3.5190763	0.0014227642276422765
2023686	1285410	3	3.4317646	0.0016260162601626016
1047812	718	10	3.3937335	0.0018292682926827
1031009	4163	17	3.271598	0.0020325203252032522
1055449	407	39	3.1267304	0.0022357723577235773
1055449	1194	119	3.109819	0.0024390243902439024
2023686	2884	1	3.0556219	0.0026422764227642275
1058890	1233770	38	3.0266361	0.00284552845528453
2023686	1270	26	3.0264745	0.00304878048780478
2005710	1001412	1575	2.981335	0.0032520325203252032
2062243	1000323	241	2.9571671	0.0034552845528455283
1059637	1000926	1	2.9293735	0.00365853658536854
2023686	1002262	39	2.9215589	0.0038617886178861 79

only showing top 20 rows

From paper 'Collaborative Filtering for Implicit Feedback Datasets'

Cold-Start Problem

- ❑ Cold-start: New users will have no to little information about them to be compared with other users.

- ❑ Cold starts occur when we attempt to predict a rating for users and/or items in the test dataset that were not present during training the model

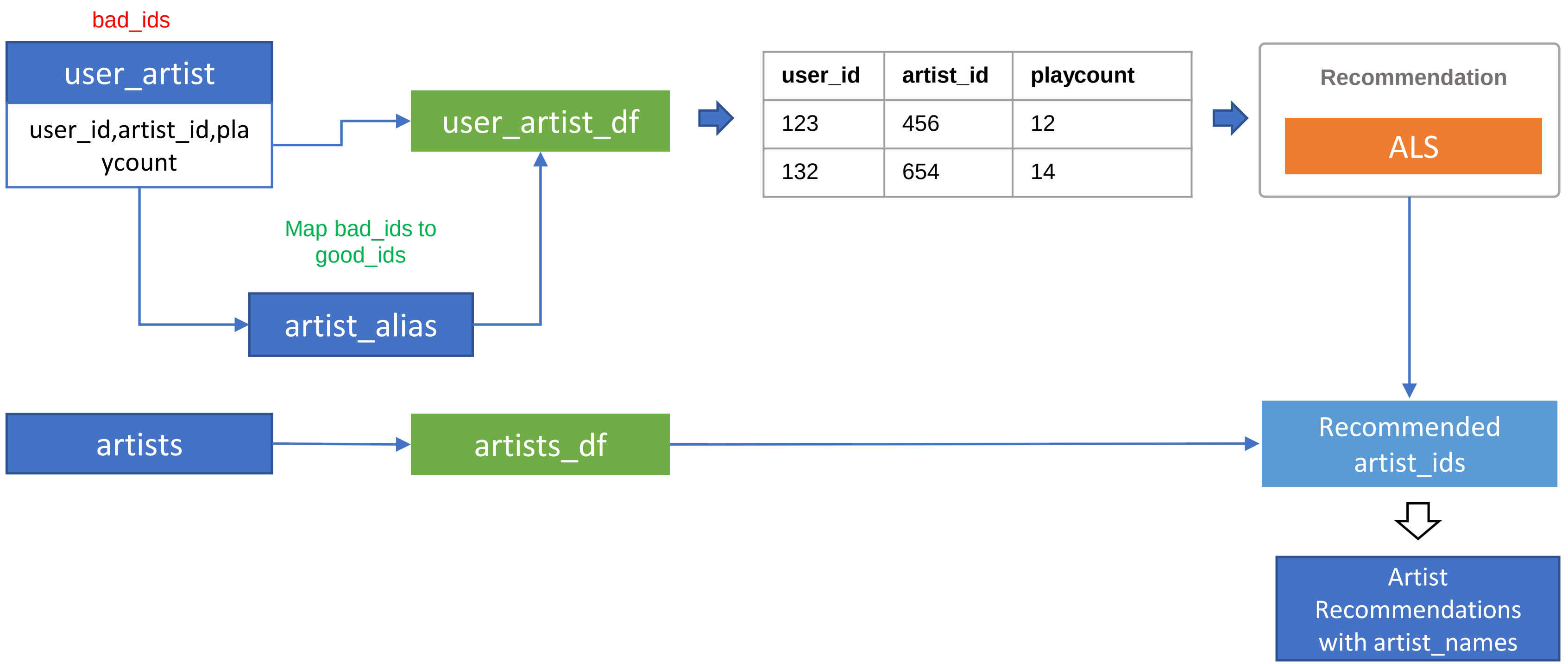
Two strategies for handling this problem:

"NaN" - return an empty variable.

- Spark assigns NaN predictions during ALSModel.transform when a user and/or item factor is not present in the model.
- In development however, this result prevents us from calculating a performance metric to evaluate the system.

"drop" - this option simply removes the row/column from the predictions that contain NaN values. Our result will therefore only contain valid numbers that can be used for evaluation.

Use Case : Music Recommendation



user_id	artist_id	playcount
123	456	12
132	654	14

Thank You!

See you next week.