Hybrid Recommender system.

Phase I → Content Based Filtering

Phase II → Collaborative filtering

Phase III → Hybrid Recommender system

EDA, Data clearing, Songs → Transformations

↓

5, 10, 15, 20 → Streamlit ← Content Based

User engagement, User retention ↑

↓ Personalized ✓ → Content Based Filtering

↓ Diverse → Collaborative filtering

Collaborative Filtering → User history ← Content based

Collaboration → Between users → User Based CF

Between Items → Item Based CF

Songs artist, attributes, metadata →

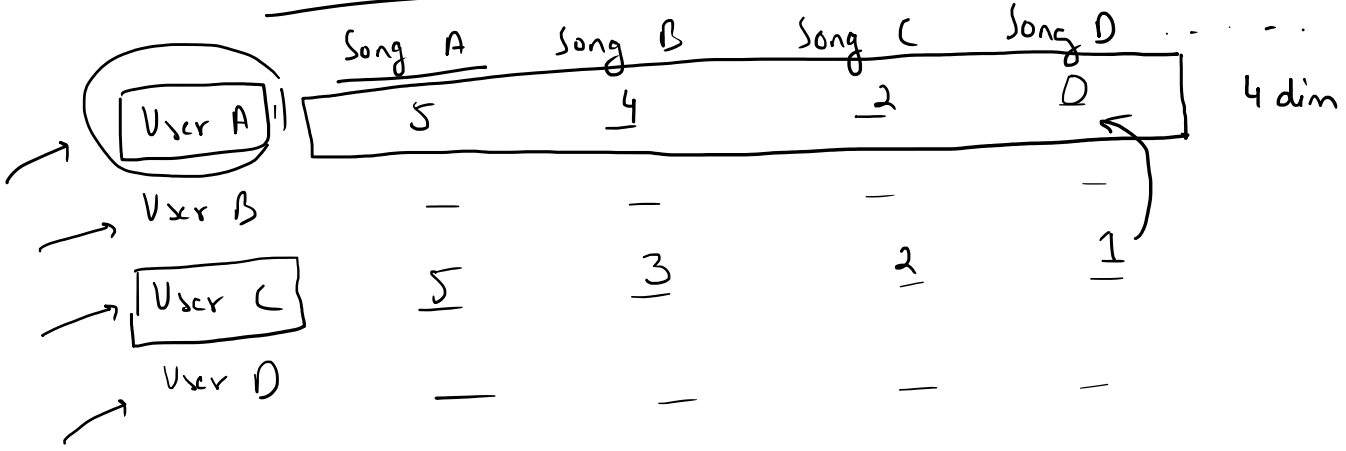
Song A [1 1 1 1 1 1] ← Songs artist, attributes, metadata

Similarity scores → Cosine similarity.

Users liking, Ratings, Playcount → Similarity

User Based CF → Similarity b/w users

Playcount → User history



Cosine similarity = Sort Similarity

↑ ↓

User Based Approach / CF → Items >> Users. ✓

Item Based Approach / CF → Users >> Items. ✓

	User X	User Y	User Z	User M	User N
Song A	—	—	—	—	—
Song B	—	—	—	—	—
Song C	—	—	—	—	—

Song C — — — — —

Song D — — — — —

Item - User Interaction Matrix → Item Based CF

Rows → Unique songs

Cols → Unique users.

Song → Vector Matrix

Similarity scores.

(5, 10)

(1, 10) X (10, 5)

(1, 5) → Recommend

Song A Song A

Song A Song B.

Previous

History Dataset

30,000 songs

track Id User Id → 9.7 lakhs Playcount

1 crore

User Item interaction matrix

(30k, 1million) → 2 byte np.int(16) (8)

30,000 x 1million → 30 Billion numbers.

→ 30 x 2 → 60 Billion Bytes.

interaction_db → 60 GB

Dask → Big Datasets → TBs

User (30K)

Song A ← 1million. 10000

Values → Zeros → sparse data. ← dense

CSR matrix ← cosine similarity

Non zero

Why Dask?

Similar to Pandas and numpy

dataframes Arrays

Easy to learn

Easy to implement

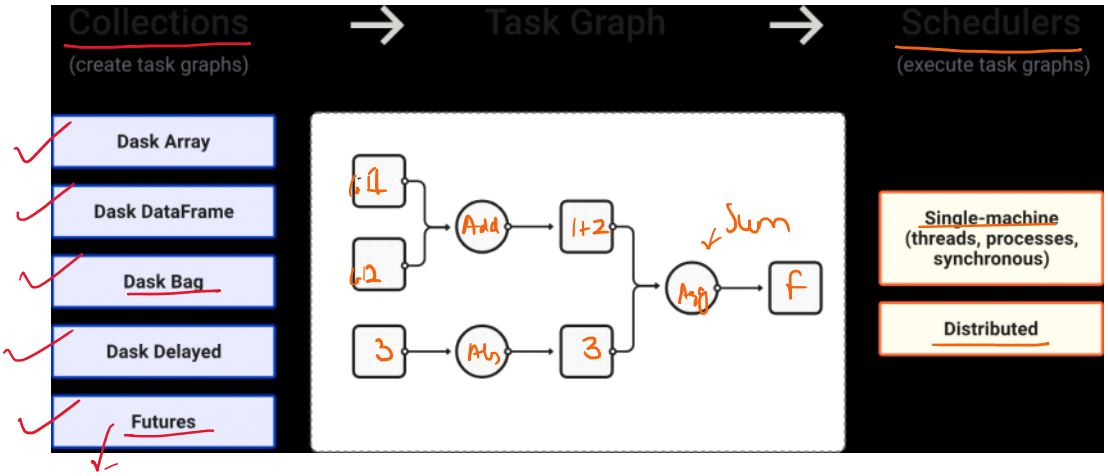
1) Familiarity

2) Parallel Computing → Multicore / Clusters.

3) Chunks

4) ML, DL, Boosting Algo

XgBoost, Light GBM



Clusters

Dask dataframe, Dask Arrays → Collections.

1) Client → User interact

2) Dask scheduler

3) Worker → Nodes on clusters. Cores on a CPU

→ Pandas code → task graphs

Parallelize

Computation Graphs.

Nodes

Functions, Operations.

Computation Graphs.

Functions, Operations.

1TB

Partitioning of data → Chunking

4 core CPU

4GB.

1GB. → 1024 chunks.

compute

Pandas, numpy → Graph

Chunks. → Sparse matrix

↓

Similarity Scores.

↓

Recommendations

Understand the flow

User history → 10 million rows.

trackId User Id Playcount

30k , 1 million → Item Based CF

Matrix → Interaction matrix

	User Id 1	User Id 2	3	4
Song A				
Song B				
Song C				

60 GB RAM → Chunking Dask

matrix → sparse matrix size ↓ zero

Cosine similarity.

interaction matrix ← dask

↓ sparse matrix

Column list
Index
large ↑

Integers. ← less space
faster

Songs datasets → 50k ← Pandas

User history → Dask dataframe

{

No. of unique songs → 30k

No. of unique users → 1 million

Unique track ids

song dataset → track Id , name, artist, url
3k ↓
Streamlit

Interaction matrix → Indexes → track id

Columns → User ids

Values → Playcount

track id User id Playcount

integers encode dask specific

Chunk → 9 divide → 10 million → User, track

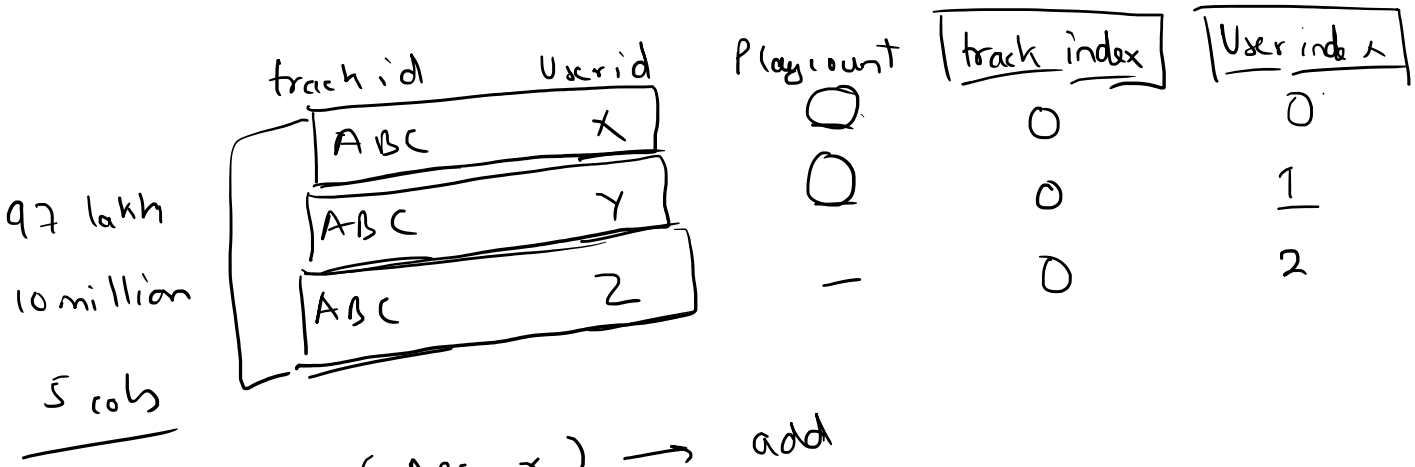
track id 1 - - - - - - - - - - - 1 million

dask → Unique songs > Categories
Unique users

categorize → [30k] → 0, 1, 2 encode index

[1 million]

User history → add new cols.



(ABC, X) → add

(ABC, Y) → add

(ABC, Z) → add.

10 million track ID User Id index Playcount

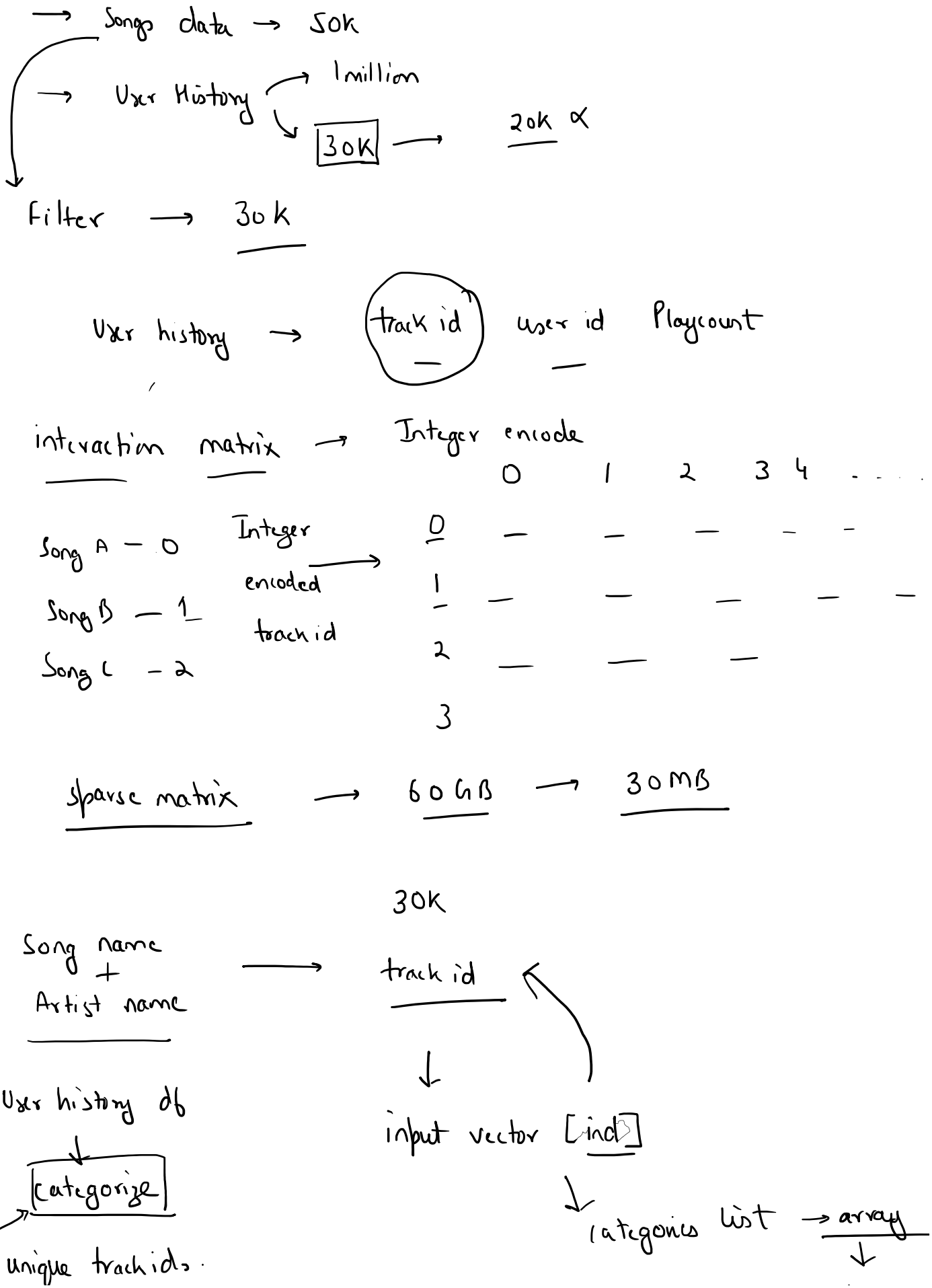
Sparse matrix

	User 0	User 1	User 2
track 0	—	—	—
track 1	—	—	—
2	—	—	—
3	—	—	

Sparse matrix

Building Pipeline

09 January 2025 15:29



categories ...

↓
save

Unique trackids.

↓ 30K

([trackid1 2 3 4 5]) ✓

[0 1 2 3 4 ...]

Cosine similarity (input vector , interaction matrix)

→ Sort Descending → Top k similarities

↓

indexes.

trackid → index → input vector → top k indexes.

↓

top k track ids.

array [top k indices]

↓

top k track ids → Songs dataset → 30k

↓

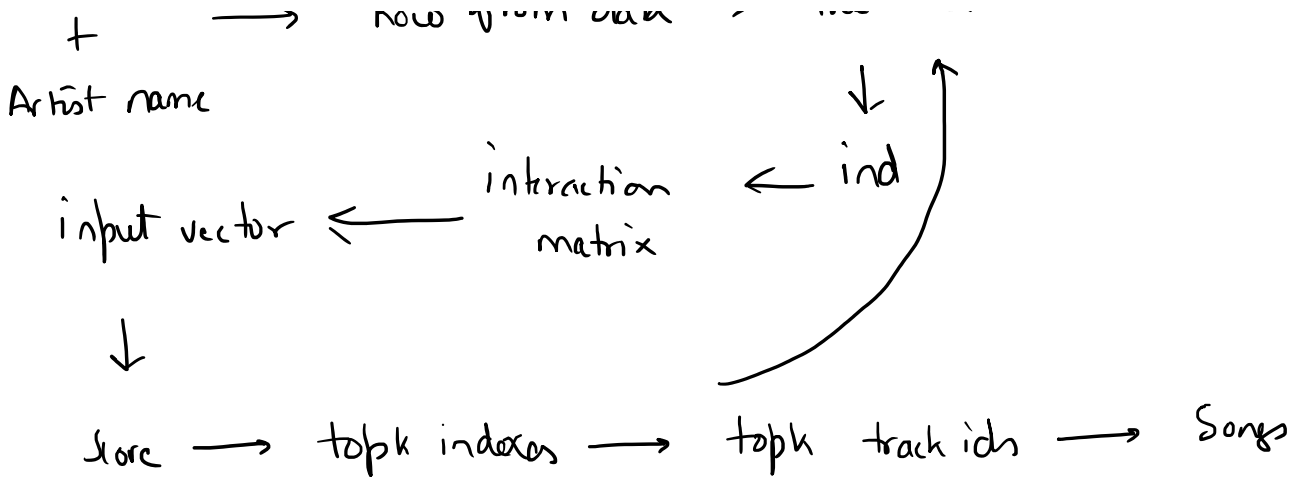
name , artist , url ← get the song details

Collaborative filtering → (Song name , artist name , 30k songs data, → 8 MB

categories array , interaction matrix)

↓ small ↓ 30 MB

Song name + → Row from data → track id


+
Artist name
interaction matrix
input vector
ind
store
topk indexes
topk track ids
Songs