What to watch?

-Recommender Systems

By Sujay Jangam

BACKGROUND

1. What is a recommender system?
    a. A system that attempts to match users to items they may be interested in

2. Why do we even care about building a recommender system?
    a. 40% of app installs on Google Play (100 Million App downloads a day)
    b. 60% of watch time on YouTube (0.6 Billion hours a day)
    c. 35% of purchases on Amazon ($0.45 Billion revenue a day)
    d. 75% of movie watches on Netflix (114 Million hours a day)

*Stats as of 2020

PROBLEM STATEMENT

1. How can we implement our own recommender system to accurately predict the rating a user may give a movie?
2. The Dataset that I will use for this task is the Movielens, 100k Dataset, that was created in September 2018.
    a. This is the most recent dataset of this size
    b. The metric that will be used to evaluate the recommender system that is built is the “Root Mean Squared Error”.
    c. Bonus: Address the **COLD START** problem

TABLE OF CONTENTS

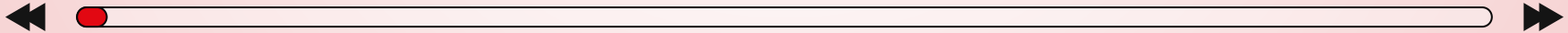
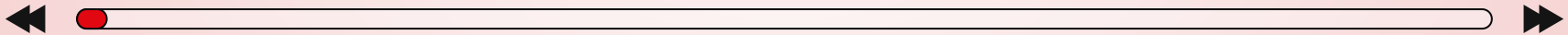


TABLE OF CONTENTS



01 Dataset 100k

Obtained from Movielens

Dataset

- Movies.csv → 9, 742 unique movies, with their genres
- Ratings.csv → 100, 836 ratings from 610 unique users across the 9,742 unique movies. Min 20 ratings per user
- Tags.csv → 3,683 instances of user generated tags
- Links.csv → Identifiers to link to other sources of movie data such as IMDB and TMDB databases

Merged DataFrame

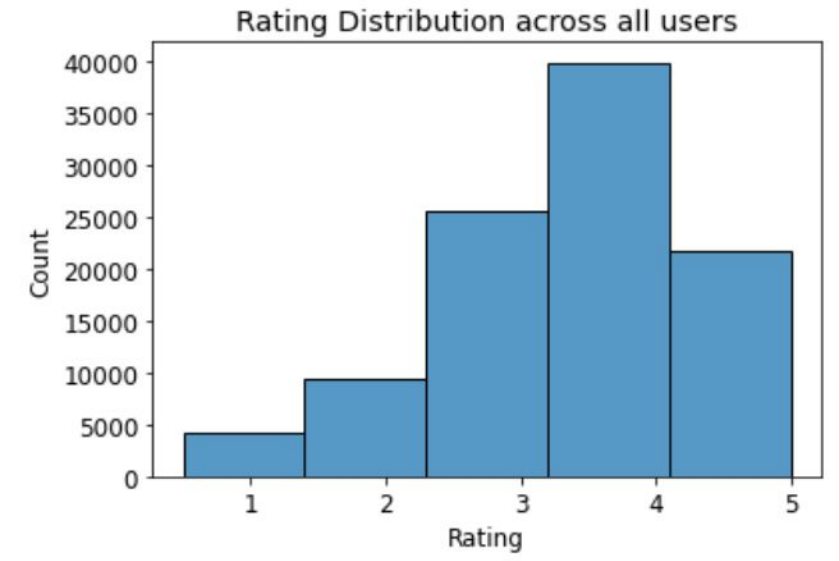
userId	movieId	rating	timestamp	title	genres
603	2961	1.0	963178906	Story of Us, The (1999)	Comedy Drama
232	64839	3.5	1234142304	Wrestler, The (2008)	Drama
191	110	3.0	829760897	Braveheart (1995)	Action Drama War
279	143859	3.0	1506395858	Hail, Caesar! (2016)	Comedy
528	5418	5.0	1391736760	Bourne Identity, The (2002)	Action Mystery Thriller

02 EDA

Ratings, Genres, Titles and External Research

Ratings

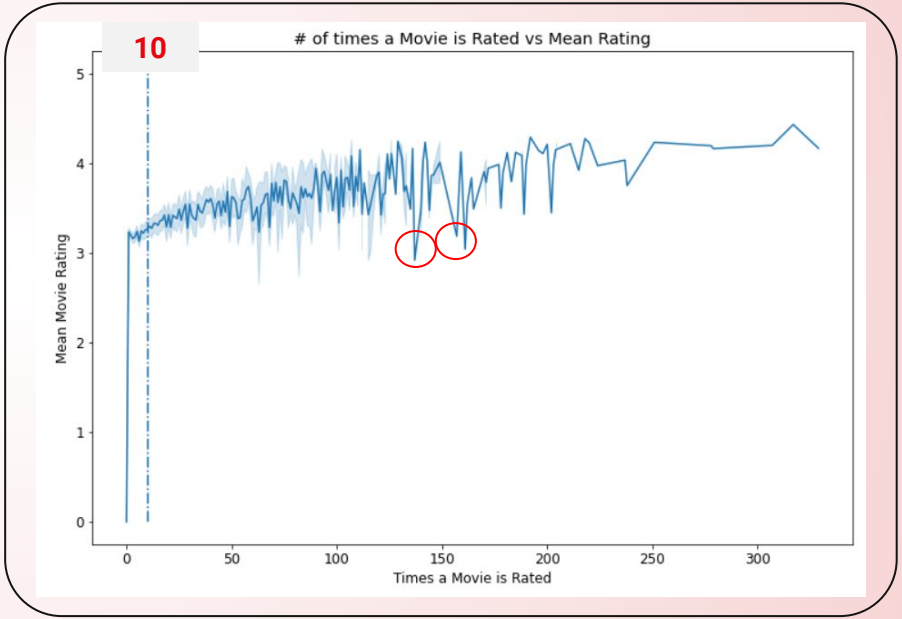
- Rating bias, possibly due to **Central Tendency Bias** or **Leniency Bias**
- Top user has rated **2,697** movies
- **Every** user has watched a movie only **once**.



Ratings

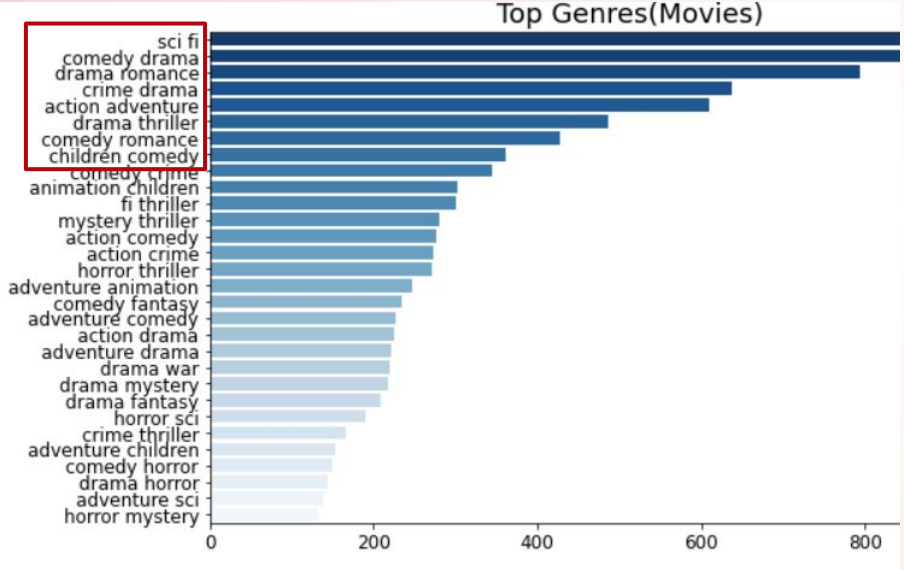
- Close to 3, 500 movies were rated only once
- As the number of times a movie is rated increases, the mean rating of the movie also increases
- Outliers are old box office hits (1990s), however sometimes, these movies pale compared to modern movies

Times a movie is rated vs Mean movie rating

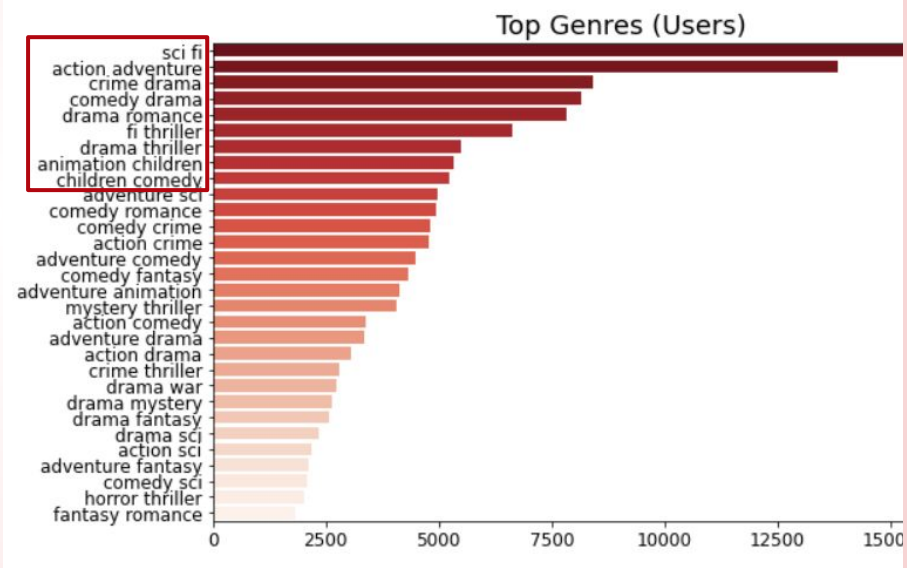


Genres (Bigrams)

Genres across movies (9k)

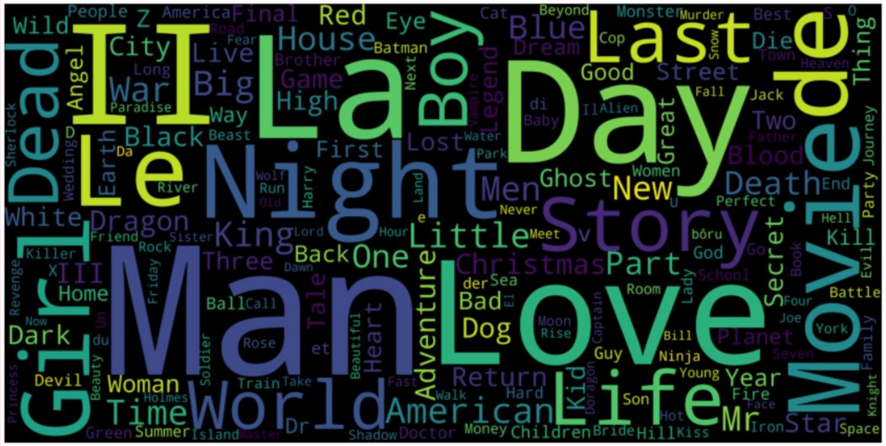


Genres across all users (100k)



Titles

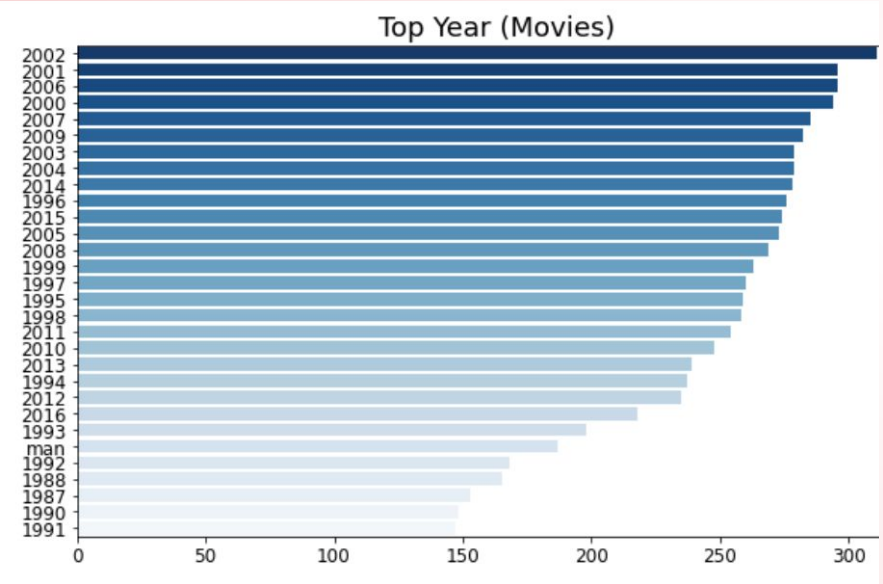
Titles across movies (9k)



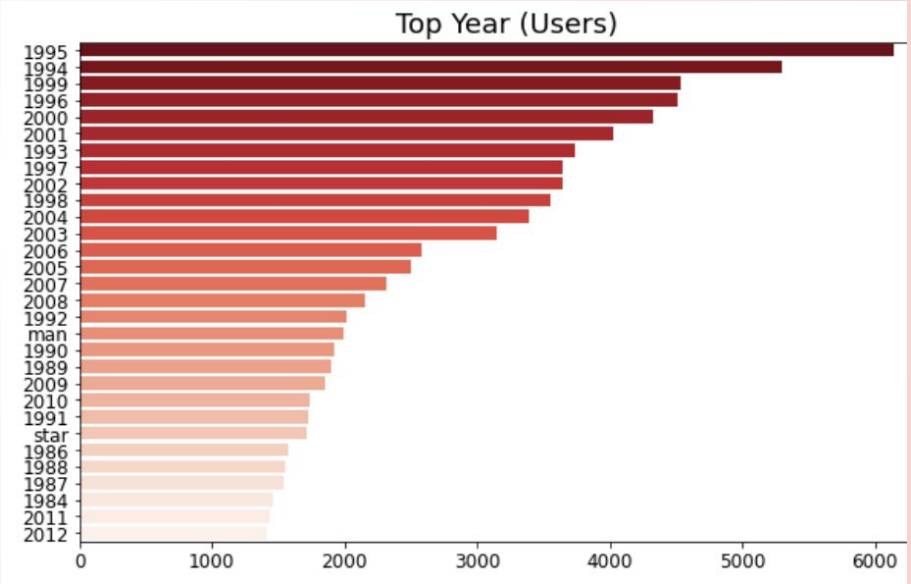
Titles across all users (100k)

Movie Year

Years across movies



Years across all users



Prediction Time

- Marketing Dive
  - 53% drop off in 3 Seconds
- Hubspot
  - Higher conversions < 2 Seconds
- Forbes
  - 1 Second is the magic number
- Advertising
  - Standard of 1000ms

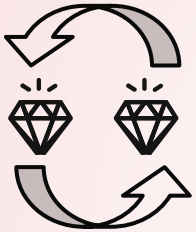
Benchmark

3 Seconds is the acceptable benchmark that we will accept

03 SurPRISE

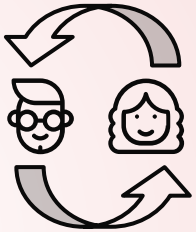
Modeling from the SurPRISE library

Simple Memory Based Collaborative Filtering



Item - Item CF Engine

RMSE: 3.652



User - User CF Engine

RMSE: 3.225
(benchmark to beat)

Base Models

- Agnostic across models
- “Baseline_only” model assumes ratings are normally distributed
- Selected 3 models to tune

Base Model Scores

model_name	model	test_rmse
base_svdpp	svdpp	0.866679
base_baseline_only	baseline	0.876241
base_svd	svd	0.879827
base_knn_baseline	knn_baseline	0.882639
base_knn_zscore	knn_zscore	0.903679
base_knn_means	knn_means	0.904506
base_slop	slope	0.909538
base_nmf	nmf	0.937035
base_cluster	cluster	0.954283
base_knn_basic	knn_basic	0.959823

Tuned Models

- "SVD+" model is the best
  - Pred_time = 5.2s
- "SVD" Model becomes our best model
  - Pred_time = 177ms (96% faster)

Tuned Model Scores

model_name	model	test_rmse
svdpp_tuned_2	svdpp	0.853009
svd_tuned_5	svd	0.861169
knn_als_tuned_5	knn_baseline	0.863791

Singular Value Decomposition (SVD)

01 What is it?

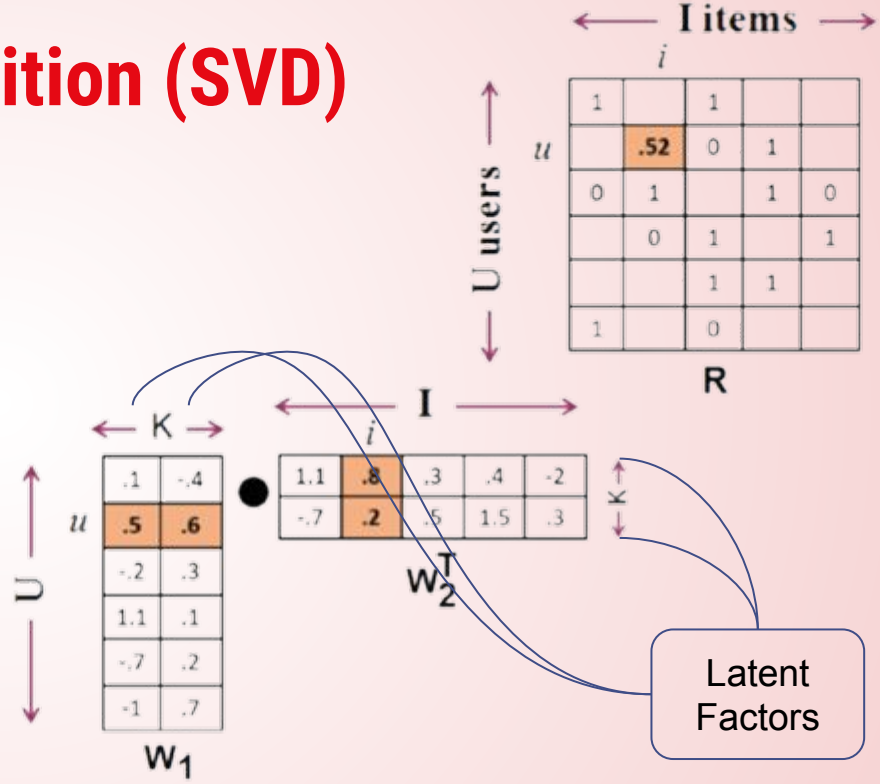
It is a matrix factorization algorithm.

02 How does the model work?

Transforms sparse data via factorization into two low rank matrix approximations.

03 Why is it useful?

Identifies underlying latent factors to predict user rating.



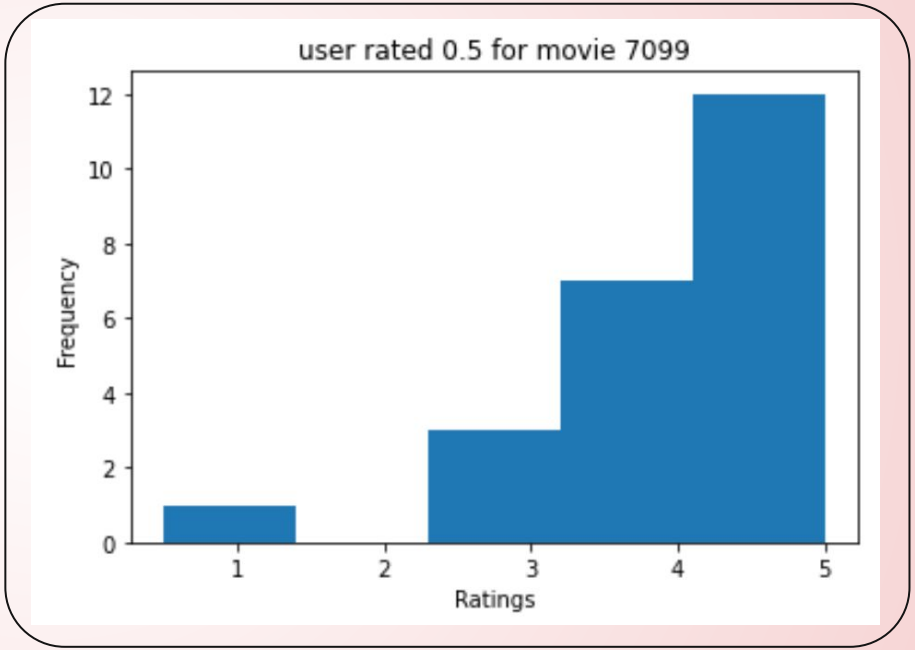
Error Analysis

userId	movieId	user_rating	pred_rating	pred_err
256	7099	0.5	4.723436	4.223436
441	527	0.5	4.721436	4.221436
393	53996	0.5	4.314065	3.814065
594	7115	0.5	4.295937	3.795937
34	110	0.5	4.282295	3.782295

Error Analysis

- Users were outliers when it came to prediction errors
- Movie 7099 had mean rating of 4, user rated 0.5
- Same pattern was observed across other movies

Distribution of Ratings for Movie 7099



Evaluation

SVD

Pros

Dimensionality Reduction

Removal of noise

Latent Underlying Factors

Successfully suggested movies of similar genres

Quick Output

The matrix factorization algorithm works very quickly, just in 117ms for a 100k dataset!

Cons

Only Ids and ratings

“UserId”, “ItemId”, “Rating” Other features not used.

Transformed Data

Hard to “see” what the latent factors are.

Cold Start Problem

The algo cannot make predictions for new users. Resort to most popular movies.

Handling of Outlier Users

The algo gives error prone recommendations to outlier users.

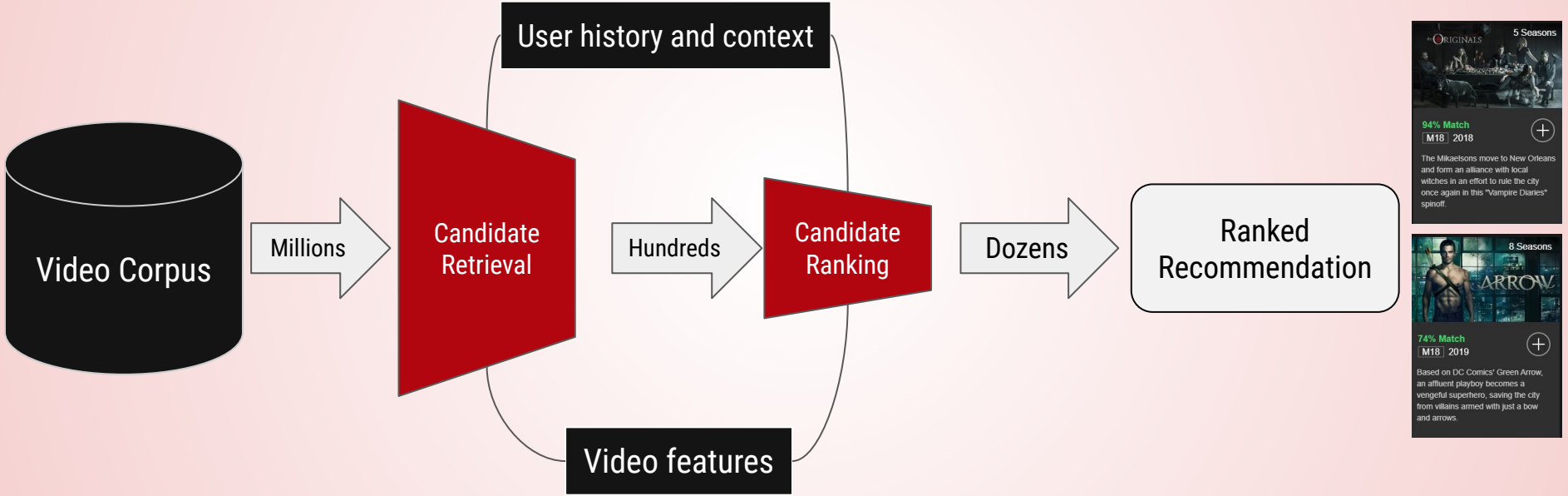
04 TensorFlow

Creating a Neural Network for Recommender System from the TensorFlow-Recommenders Library

Standard Practice

- Split the recommendation process into 2-3 parts
- Candidate Retrieval
  - List of items a **specific user** *might* like, filters out the noise
- Candidate Ranking
  - Ranking the items pulled out by Retrieval model to produce a list of personalized recommendations for that **specific user**

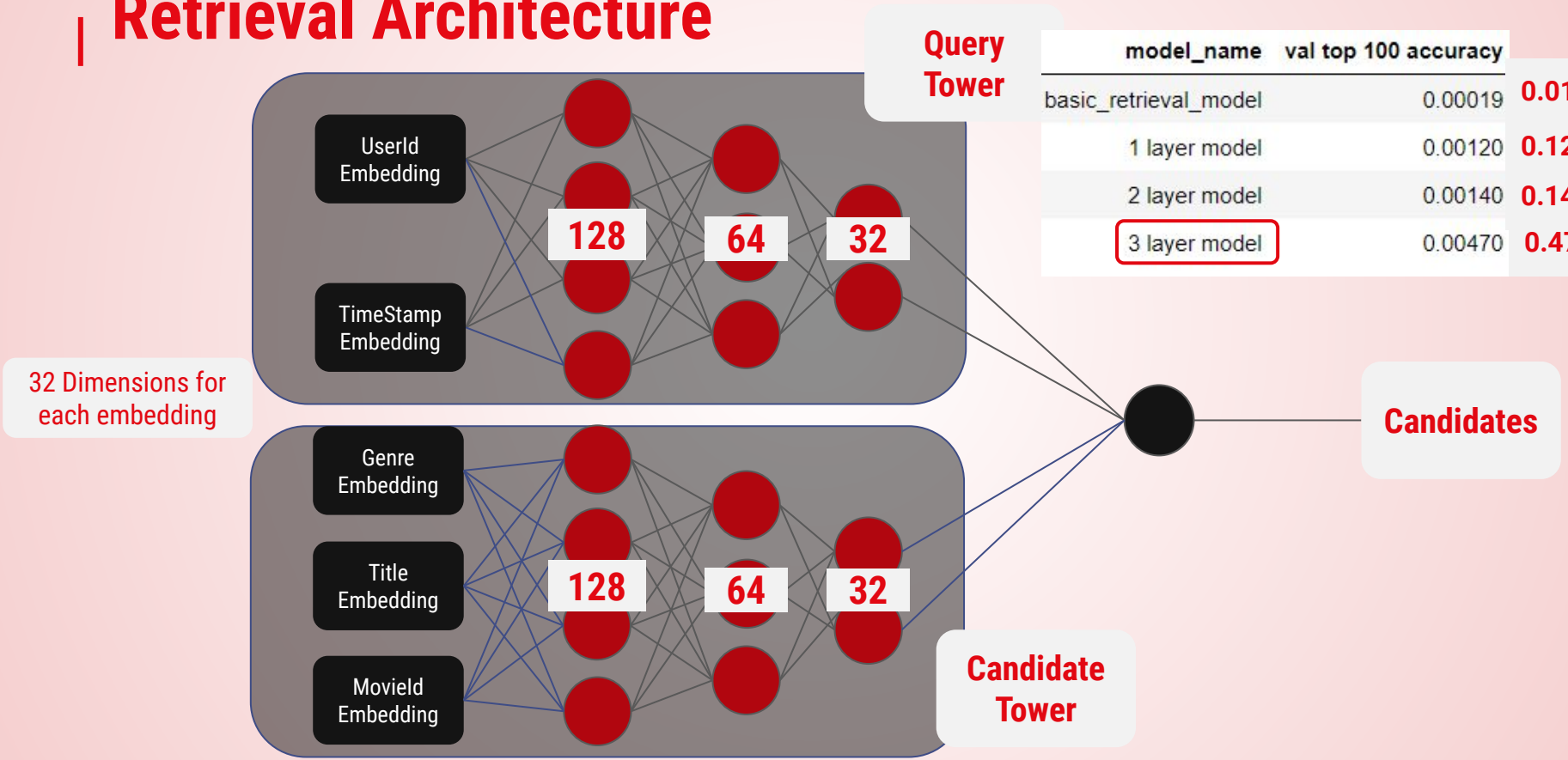
Overall Model Flow



Retrieval Model

- Further Split into 2 sub-models
- Query Tower
  - Computes query representation that produces embedding for each user
- Candidate Tower
  - Computes candidate representation that produces an embedding for each item
- Outputs are multiplied to give query-candidate affinity score
- Based on implicit feedback of having watched the movie

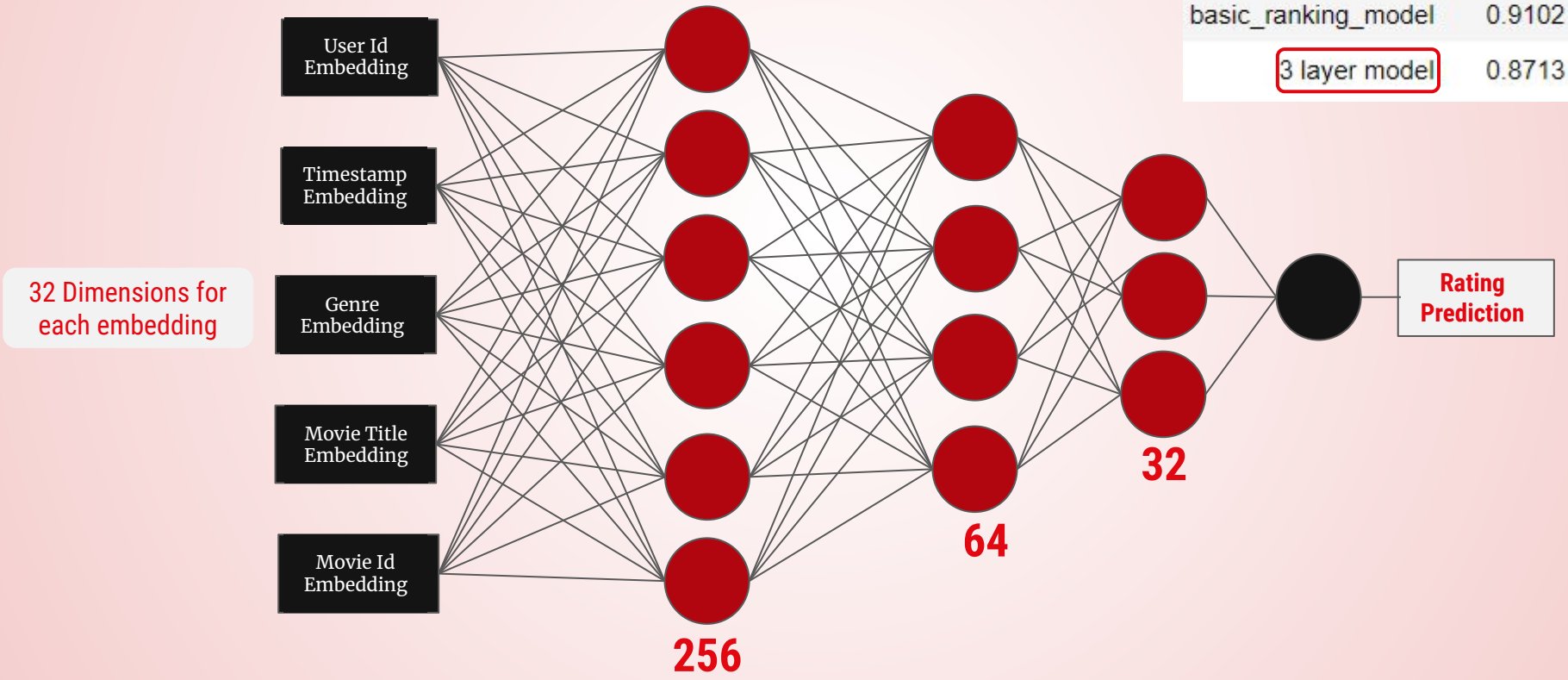
Retrieval Architecture



model_name	val top 100 accuracy	
basic_retrieval_model	0.00019	0.019%
1 layer model	0.00120	0.12%
2 layer model	0.00140	0.14%
3 layer model	0.00470	0.47%

Ranking Architecture

model_name	val rmse
basic_ranking_model	0.9102
3 layer model	0.8713



Evaluation

TFRS

Total Pred Time:
1.5s (avg)

Pros

Feature Acceptance

We can use many more features

Add Unknown Tokens

The model can handle new users, no cold start problem.

RMSE Score

With just adding more features, the NN has reached close to our SVD RMSE.

Cons

Hard to train

Complicated to set up and not easy to understand.

Time taken to train

For a 100k dataset, took a total of 6-8 hrs

Deeper models

Careful hyperparameter tuning needs to be done in order to get good performance

05 Conclusion

Next Steps

TFRS

Features

Non History features (e.g. Age, Occupation, Year of movie)

Tuning

Tuning using Keras Tuner Library

Architecture

Extreme Multiclass Classification

SVD

New Models

SurPRISE library is not going to be updated

CFSVD

SVD with contextual features, research

Tuning

More detailed tuning of “deeper” hyperparameters

MY Learnings

Never Enough Time

Important to work within a scope, it is easy to fall into a “black hole”

Tenacity & Independence

I was able to solve the issues with TFRS, (there were loads). Managed to self learn it.

Take a break!

A calm mind is going to give you solutions much quicker than a panicked mind.

THANK YOU!

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon and infographics & images by Freepik