MAXELLA APP

Hybrid Movies Recommender Using TensorFlow Recommender (TFRS)

Aktham Momani – Data Science: Advanced Machine Learning
Mentor: Kenneth Gil-Pasquel

Problem Statement

This Project is all about how to successfully formulate a recommendation engine, the difference between implicit and explicit feedback and how to build a movie recommendation system with TensorFlow and TFRS.

Context

Google/YouTube is all about connecting people to the movies/videos they love. To help customers find those movies, they developed world-class movie recommendation system called TensorFlow Recommender (TFRS). Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Google/YouTube uses those predictions to make personal videos recommendations based on each user's unique tastes.

Criteria Of Success

The ability to build a Movie Recommendation Engine with the highest possible retrieval accuracy (Predicting Movies) AND with the lowest Loss/RMSE (Ranking Movies)

Constraints

TensorFlow Recommender (TFRS) is a brand new package where there's very low used cases, so for the success of this project there will be plenty of research involved to successfully complete this project.

Key Data Source

TensorFlow Dataset (movie_lens/1m-ratings & movie_lens/1m-movies) and metadata_movies/credit from Movielens website.

Data Source

Because of the richness of the metadata in Tensorflow Movie Lens dataset, we have decided to choose 1 million Movie lens from TensorFlow to be our main dataset for this project. Also, we used both datasets from Movielens website: movies metadata & credits.

Features

- Config description: This dataset contains 1,000,085 anonymous ratings of approximately 3,619 movies made by 6,040 MovieLens users who joined MovieLens. Ratings are in whole-star increments. This dataset contains demographic data of users in addition to data on movies and ratings.
- This dataset is the largest dataset that includes demographic data from movie_lens.
- "user_gender": gender of the user who made the rating; a true value corresponds to male
- "bucketized_user_age": bucketized age values of the user who made the rating, the values and the corresponding ranges are:
- "movie_genres": The Genres of the movies are classified into 21 different classes as below:
- "user_occupation_label": the occupation of the user who made the rating represented by an integer-encoded label; labels are preprocessed to be consistent across different versions
- "user_occupation_text": the occupation of the user who made the rating in the original string; different versions can have different set of raw text labels
- "user_zip_code": the zip code of the user who made the rating.
- "release_date": This is the movie release date, in unix epoch (UTC - units of seconds) (int64).
- "director": This is the director of the movie.
- "start": This is the main star of the movie.

Data Wrangling

The Data wrangling step focuses on collecting our data, organizing it, and making sure it's well defined. For our project we have collected below datasets to have a good foundation so we can build a Deep Learning model with the best performance possible:

movie_lens/1m-ratings | movie_lens/1m-movies | movies_metadata.csv | credits.csv

Objective:

Wrangle above datasets to Convert 1 to 2 with more features and cleaned data:

1

	bucketized_user_age	movie_genres	movie_id	movie_title	timestamp	user_gender	user_id	user_occupation_label	user_occupation_text	user_rating	user_zip_code
0	25.0	[3, 4]	b'586'	b'Home Alone (1990)'	975897100	True	b'595'	6	b'executive/managerial'	4.0	b'10019'
1	35.0	[0, 1, 4, 14]	b'1197'	b'Princess Bride, The (1987)'	972790580	False	b'2804'	11	b'other/not specified'	5.0	b'46234'
2	25.0	[4, 14]	b'2502'	b'Office Space (1999)'	974757114	True	b'1457'	0	b'academic/educator'	4.0	b'95472'
3	18.0	[1, 3, 8]	b'2'	b'Jumanji (1995)'	965806208	True	b'3887'	17	b'college/grad student'	3.0	b'80513'
4	35.0	[10, 16]	b'1717'	b'Scream 2 (1997)'	976474533	True	b'329'	6	b'executive/managerial'	2.0	b'02115'

2

	bucketized_user_age	movie_genres	movie_id	movie_title	timestamp	user_gender	user_id	user_occupation_label	user_occupation_text	user_rating	user_zip_code	director	release_date	star
0	50	[7]	1251	Eight and half	974089380	False	2497	14	sales/marketing	3	37922	Federico Fellini	-217123200	Marcello Mastroianni
1	18	[7]	1251	Eight and half	986722200	True	671	17	college/grad student	5	61761	Federico Fellini	-217123200	Marcello Mastroianni
2	45	[7]	1251	Eight and half	960071880	False	5590	12	programmer	2	94117	Federico Fellini	-217123200	Marcello Mastroianni
3	25	[7]	1251	Eight and half	1011993120	True	1851	20	unemployed	5	59602	Federico Fellini	-217123200	Marcello Mastroianni
4	35	[7]	1251	Eight and half	963100320	False	5526	1	artist	5	27514	Federico Fellini	-217123200	Marcello Mastroianni

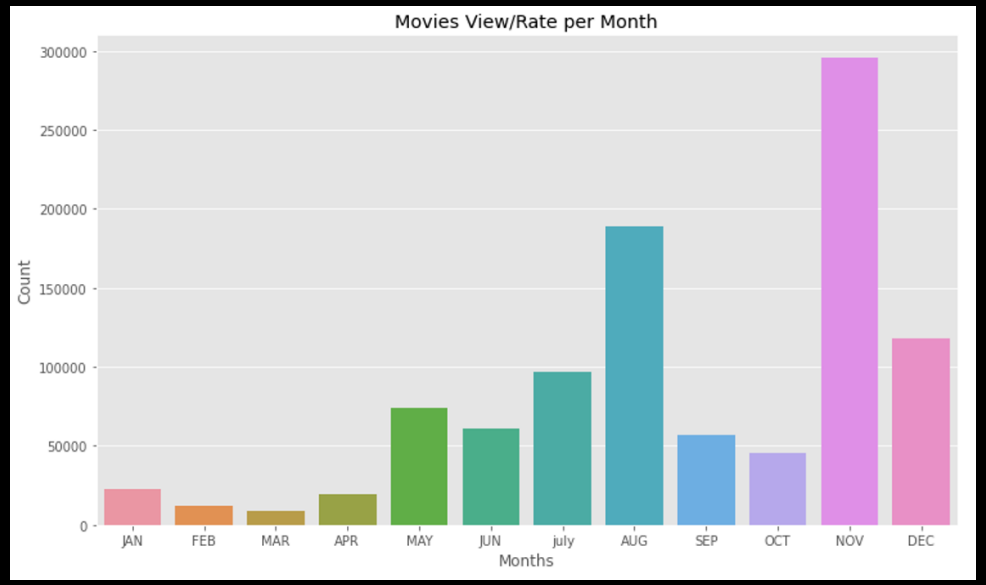
Exploratory Data Analysis EDA

Now that we've obtained, cleaned, and wrangled our dataset into a form that's ready for analysis, it's time to perform exploratory data analysis (EDA).

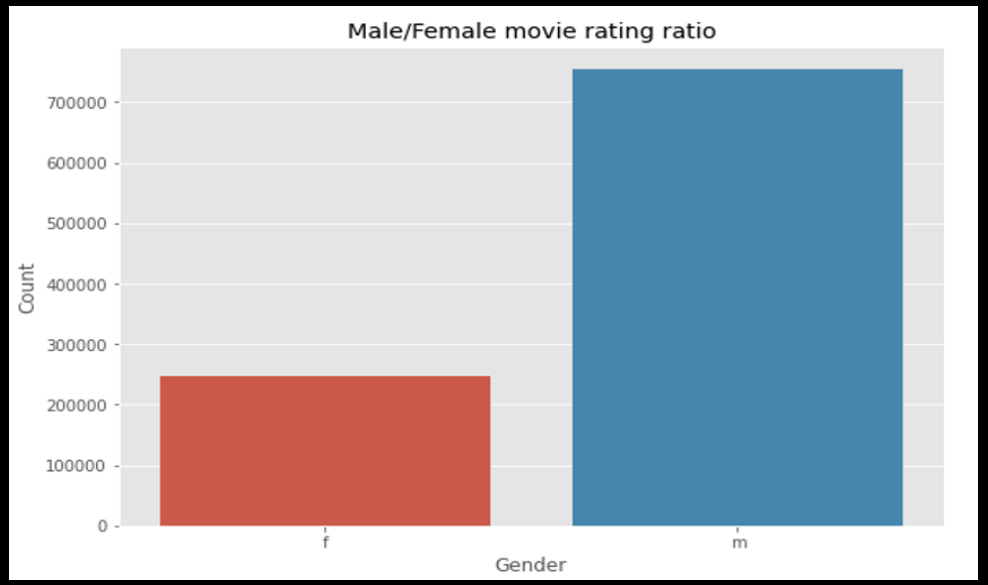
Initial Statistics Summary

	bucketized_user_age	timestamp	user_occupation_label	user_rating	user_zip_code
count	1000209.0	1000209.0	1000209.0	1000209.0	1000209.0
mean	29.7	972243695.4	11.1	3.6	54230.9
std	11.8	12152558.9	6.6	1.1	32090.6
min	1.0	956703932.0	0.0	1.0	231.0
25%	25.0	965302637.0	6.0	3.0	23185.0
50%	25.0	973018006.0	12.0	4.0	55129.0
75%	35.0	975220939.0	17.0	4.0	90004.0
max	56.0	1046454590.0	21.0	5.0	99945.0

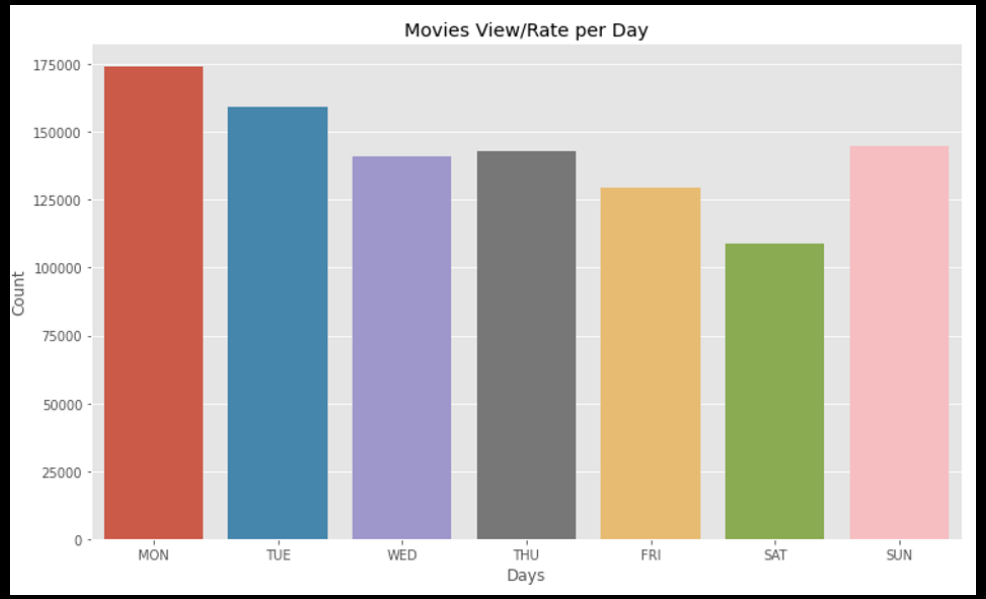
What is the preferable month of the year to rate/watch movies?



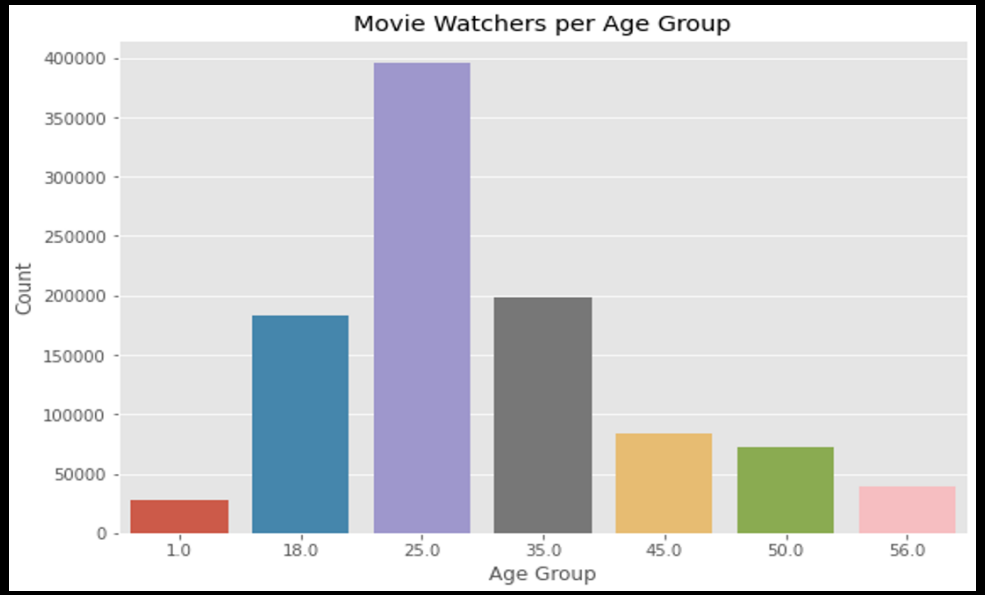
Who watches/rates more movies Men/Women?



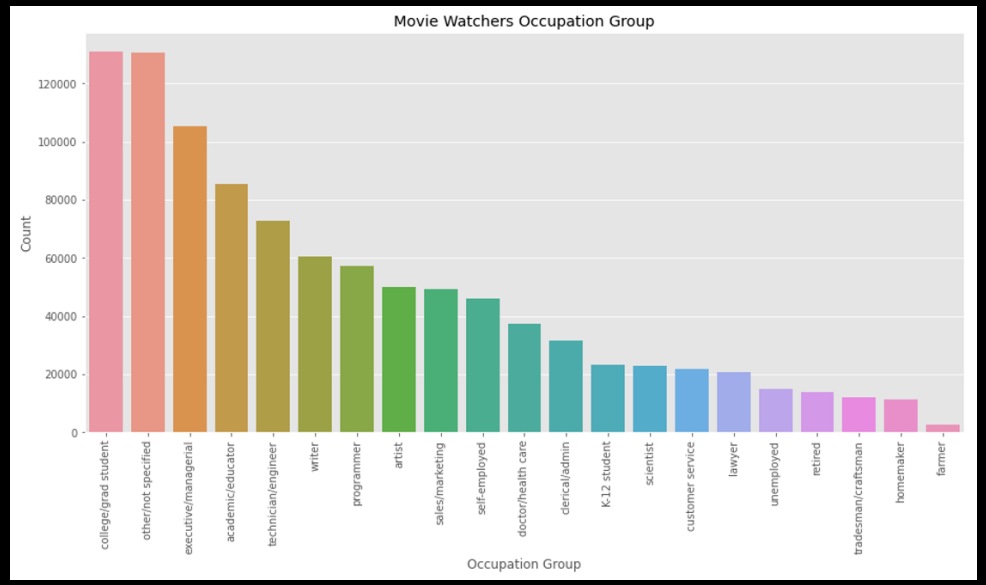
What is the preferable day of the week to rate/watch movies?



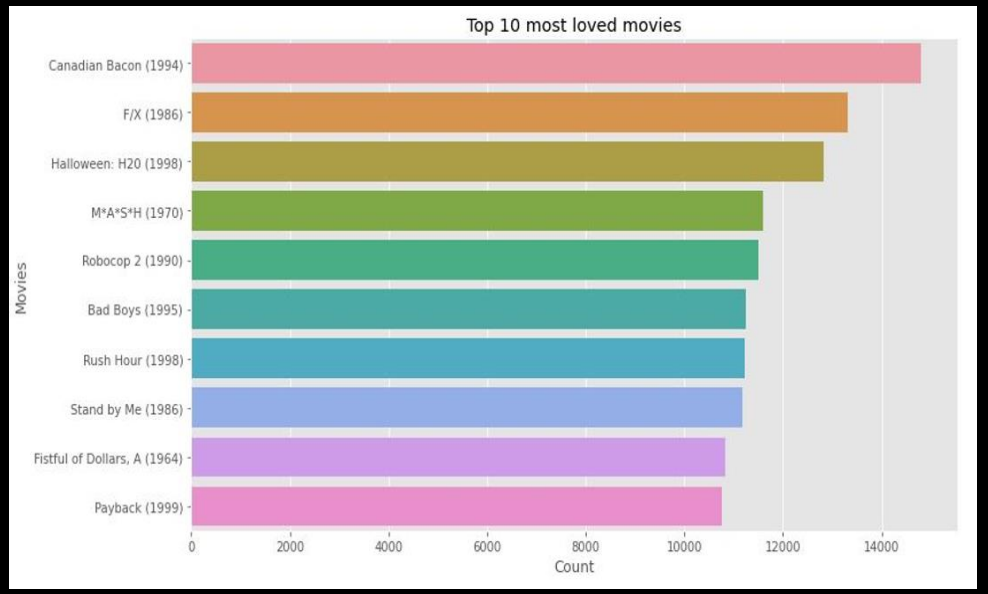
What age group watches more movies?



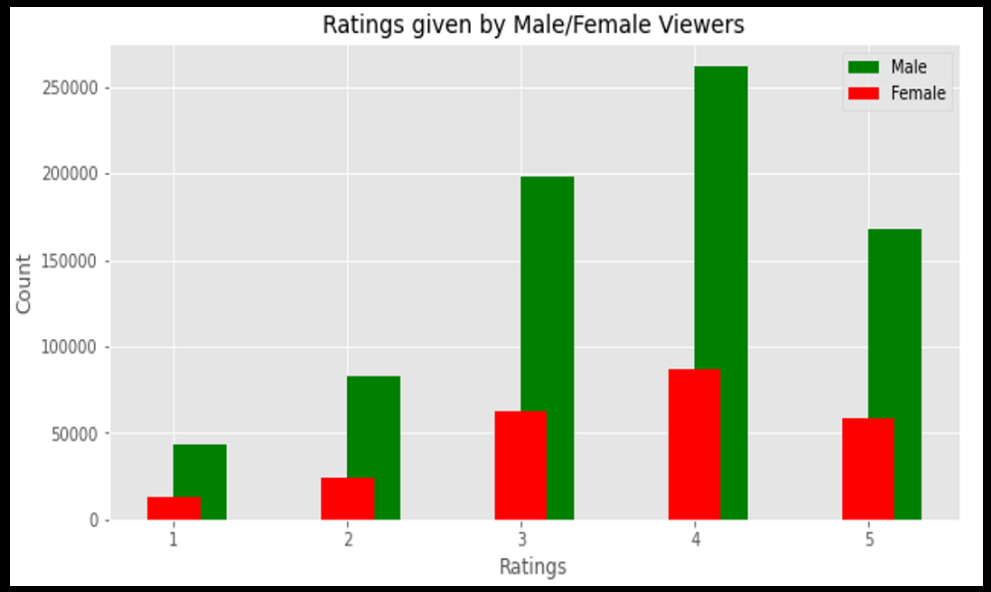
What kind of occupants watches/rates more movies?



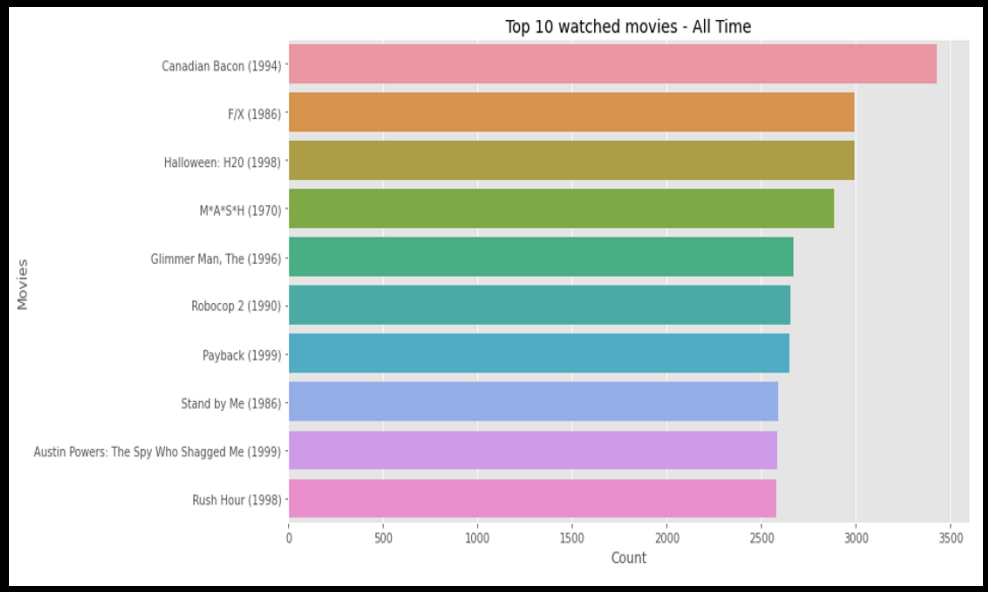
What are the most loved Movies?



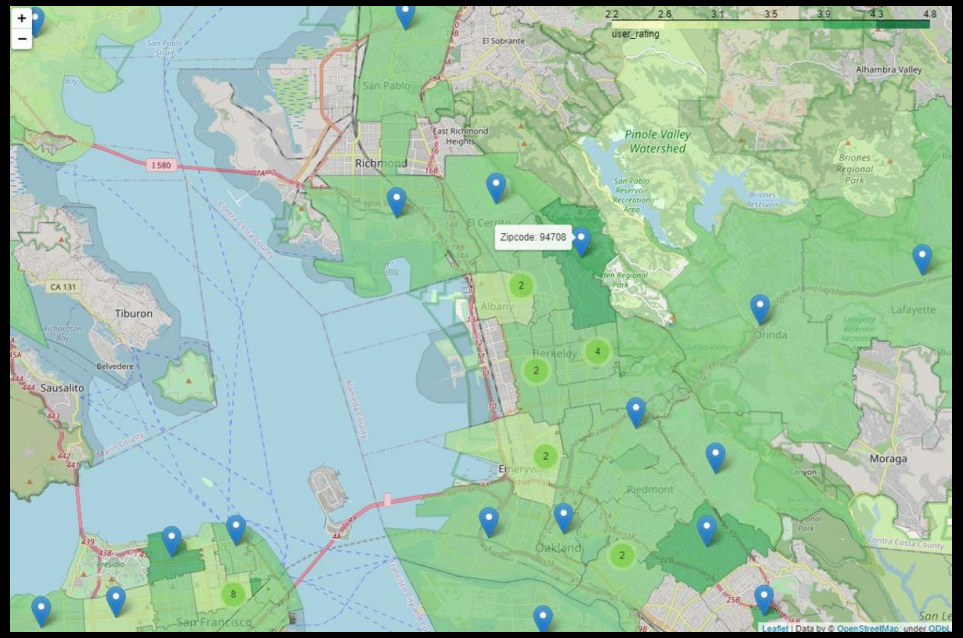
How much rating people give mostly per Gender?



What are the most rated movies?

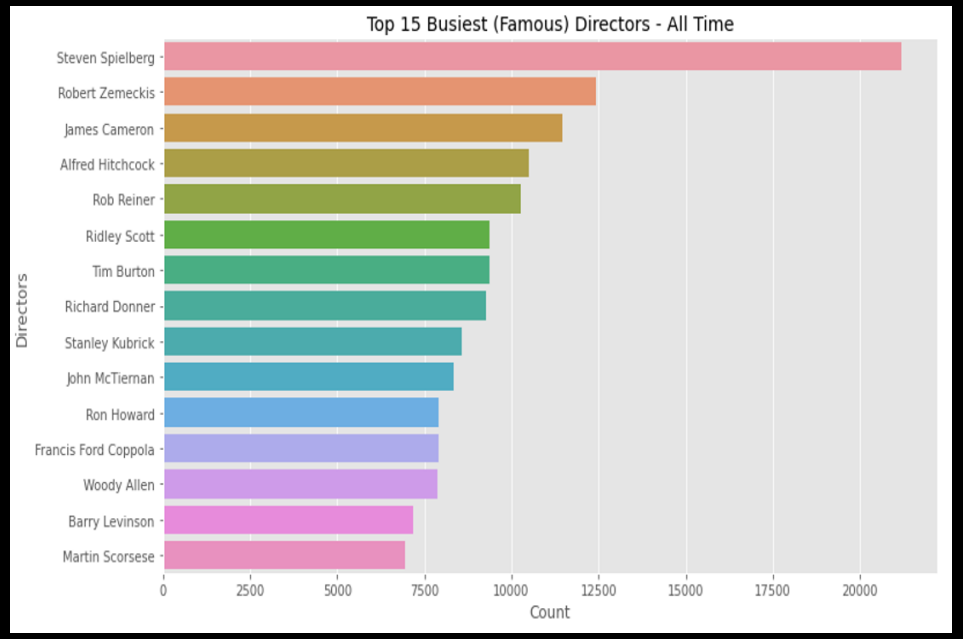


Is there any relations between the user rating and location?

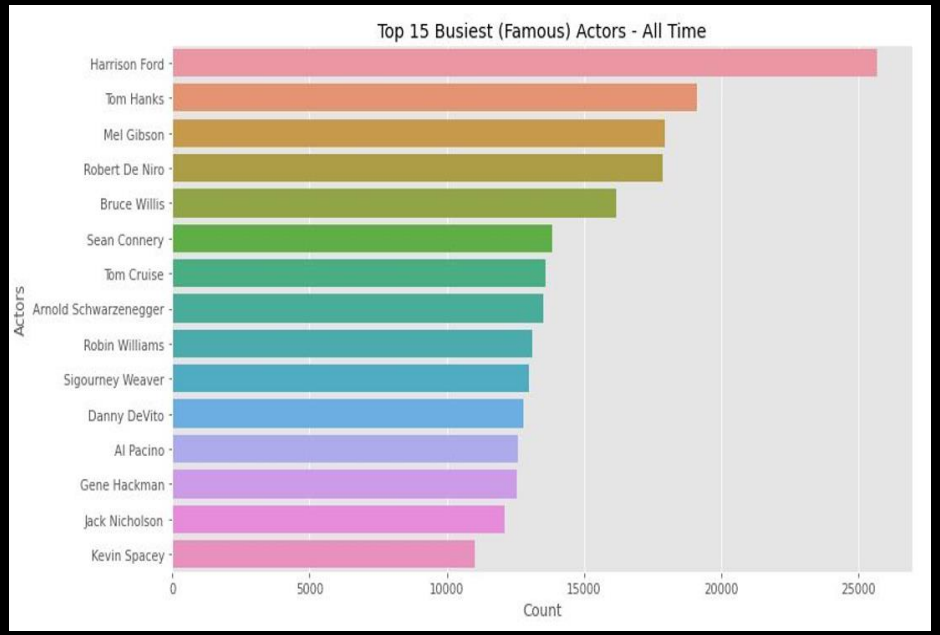


What are the worst movies per rating??

The top 15 busiest (Famous!) Directors

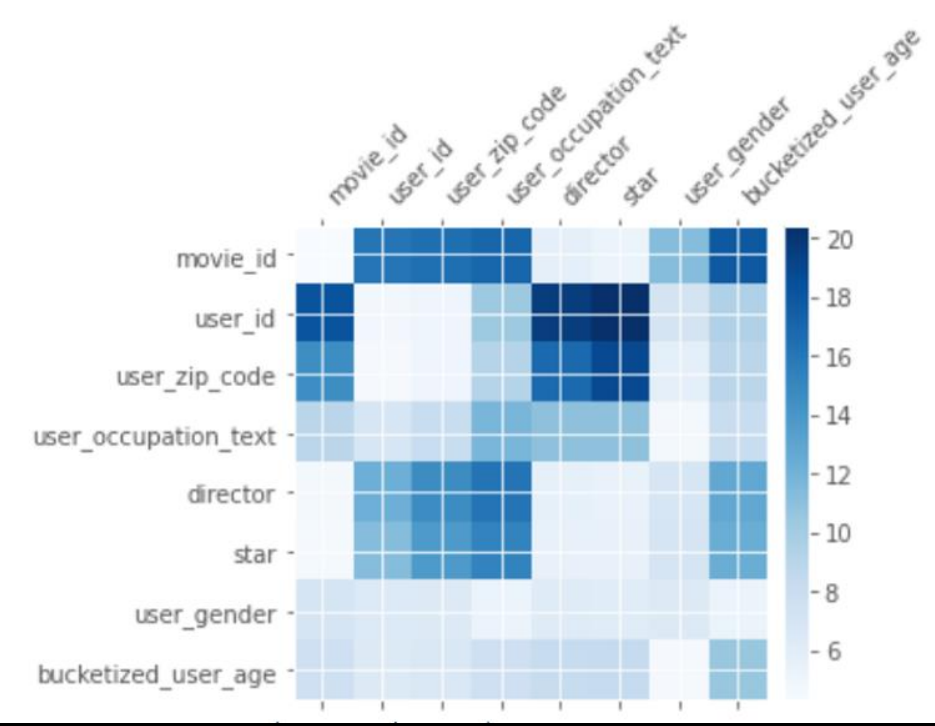


The top 15 busiest (Famous!) Actors



Machine Learning Modeling: TensorFlow Recommenders (TFRS)

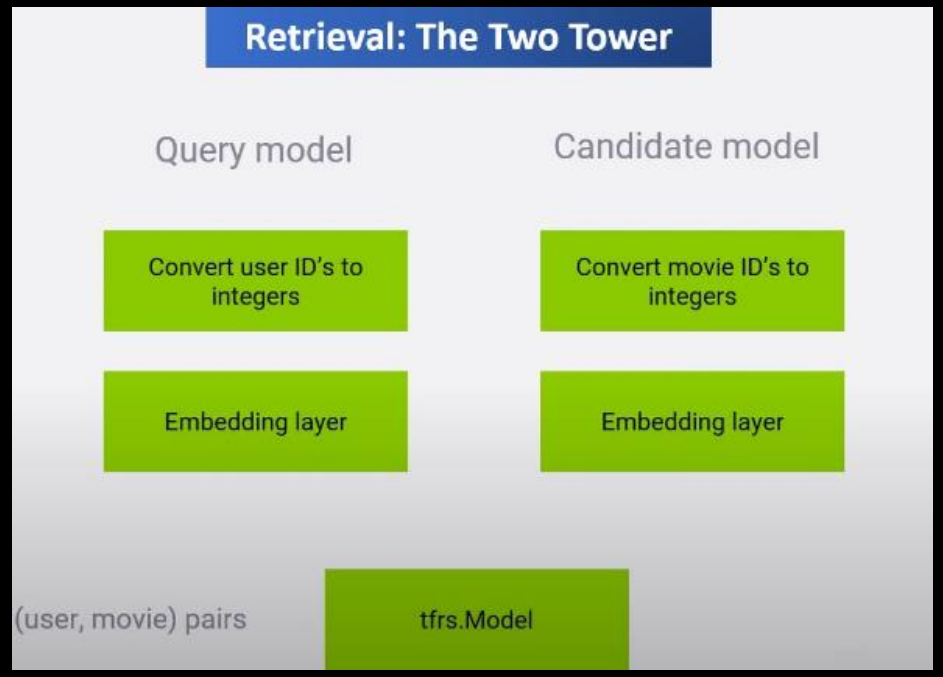
Deep and Cross Network (DCN-v2) For feature Importance



One of the nice things about DCN is that we can visualize the weights from the cross network and see if it has successfully learned the important feature process.
As shown above, the stronger the interaction between two features is. In this case, the feature cross of user ID and movie ID, director, star are of great importance

Machine Learning Modeling: TensorFlow Recommenders (TFRS)

First Stage: Retrieval (The Two Towers Model)



Tuning Summary:

- As we can see below, we managed to improve accuracy and reduce loss by:
- Increase embedding_dimension from 32-> 64
- keeping learning_rate 0.1
- As a result, loss was reduced from 903.56 (Baseline) to 846.05 and top_10_accuracy was improved from 3.2% to 4.3%.

Tuning	top_1_accuracy	top_5_accuracy	top_10_accuracy	top_50_accuracy	top_100_accuracy	loss
Baseline Model: embedding_dimension 32 -> 64, learning_rate 0.1, epochs=5	0.002739	0.016660	0.032353	0.125389	0.207780	903.56
Model_1: embedding_dimension 32 -> 64, learning_rate 0.1 , epochs=5	0.002999	0.020739	0.039571	0.143078	0.231983	874.51
Model_2: embedding_dimension 64, learning_rate 0.01, epochs=5	0.003095	0.021155	0.040475	0.145901	0.235966	868.06
Model_3: embedding_dimension 64, learning_rate 0.001, epochs=5	0.002995	0.021155	0.040491	0.145969	0.236346	867.46
Model_4: embedding_dimension 64, learning_rate 0.1, epochs=32	0.003347	0.022155	0.043127	0.159742	0.258966	846.05

Machine Learning Modeling: TensorFlow Recommenders

Second Stage: Ranking

Tuning Summary

As shown below, we managed to reduce RMSE and loss from 93% to 86.89% and 81% to 64% respectively using below:

- Increased embedding_dimension from 32 ==> 64.
- Increased epochs from 3 ==> 32
- Increase Dense Layers from 2 ==> 4
- Adding Dropout + Adding Max Norm

Ranking Tuning	RMSE	loss
Baseline Model: 2 Dense Layers + embedding_dimension = 32 + epochs = 3	0.931866	0.810238
Model 1: 2 Dense Layers + embedding_dimension = 64 + epochs = 32	0.898368	0.853799
Model 2: Deeper: 3 Dense Layers + embedding_dimension = 64 + epochs = 32	0.873792	0.805714
Model 3: Deeper: 4 Dense Layers + embedding_dimension = 64 + epochs = 32	0.868916	0.65914
Model 4: Deeper: 4 Dense Layers + embedding_dimension = 64 + epochs = 32 + Adding Dropout + Adding Max Norm	0.866493	0.643070

Machine Learning Modeling: TensorFlow Recommenders

Multi-Task Model (Joint): The Two Towers + Ranking Model

Due to the important and the high possibility of this model:

- We'll focusing in using the important features as shown from DCN-v2 (Figure 25): "user_occupation_text", "user_gender", "director", "star", "bucketized_user_age",
- Normalize all Numerical Features: timestamps, user_age, user_gender.
- Reconfigured all Movie Lens Classes (TensorFlow Recommenders) to accommodate the new embedding design due to new features, having deeper neural networks and adding regularization to help overfitting:
  - class UserModel
  - class QueryModel
  - class MovieModel
  - class CandidateModel
  - class MovielensModel

Models	Retrieval top-100 accuracy	Ranking RMSE
Model 1: Rating-specialized model	0.031	0.940
Model 2: Retrieval-specialized model	0.158	3.868
Model 3: Joint model	0.158	0.974
Model 4: Joint model (Tuned)	0.129	0.992

Next Step:

As I mentioned, there's huge potential in this model before we give up, so we'll complete:

- More Hyperparameters fine tuning.
- Add the missing sequential features like movie_genres and age (Model failed), so there will be more research in this part.

TensorFlow Recommender (TFRS) overall Models Performance (TFRS)

Retrieval Model

Tuning	top_1_accuracy	top_5_accuracy	top_10_accuracy	top_50_accuracy	top_100_accuracy	loss
Baseline Model: embedding_dimension 32 -> 64, learning_rate 0.1, epochs=5	0.002739	0.016660	0.032353	0.125389	0.207780	903.56
Model_1: embedding_dimension 32 -> 64, learning_rate 0.1 , epochs=5	0.002999	0.020739	0.039571	0.143078	0.231983	874.51
Model_2: embedding_dimension 64, learning_rate 0.01, epochs=5	0.003095	0.021155	0.040475	0.145901	0.235966	868.06
Model_3: embedding_dimension 64, learning_rate 0.001, epochs=5	0.002995	0.021155	0.040491	0.145969	0.236346	867.46
Model_4: embedding_dimension 64, learning_rate 0.1, epochs=32	0.003347	0.022155	0.043127	0.159742	0.258966	846.05

Ranking Model

Ranking Tuning	RMSE	loss
Baseline Model: 2 Dense Layers + embedding_dimension = 32 + epochs = 3	0.931866	0.810238
Model 1: 2 Dense Layers + embedding_dimension = 64 + epochs = 32	0.898368	0.853799
Model 2: Deeper: 3 Dense Layers + embedding_dimension = 64 + epochs = 32	0.873792	0.805714
Model 3: Deeper: 4 Dense Layers + embedding_dimension = 64 + epochs = 32	0.868916	0.65914
Model 4: Deeper: 4 Dense Layers + embedding_dimension = 64 + epochs = 32 + Adding Dropout + Adding Max Norm	0.866493	0.643070

Multi-Task Model (Joint)

Models	Retrieval top-100 accuracy	Ranking RMSE
Model 1: Rating-specialized model	0.031	0.940
Model 2: Retrieval-specialized model	0.158	3.868
Model 3: Joint model	0.158	0.974
Model 4: Joint model (Tuned)	0.129	0.992

Future Work

- More research is needed to optimize existing Tensorflow Recommenders Classes to accommodate the extra features created in this project
- More fine tune is needed in order a better retrieval prediction and low RMSE Ranking from the Joint Model (Multi-Task)
- Explore and check Listwise ranking and see the benefits if any.
- Building Maxilla App