Final Capstone Report

Movie Recommendation using Tensorflow Recommenders

Mohamed Ziane – Data Science Track – Machine Learning Specialty

Mentor: **Dipanjan Sarkar**

January 2022

Table of Contents

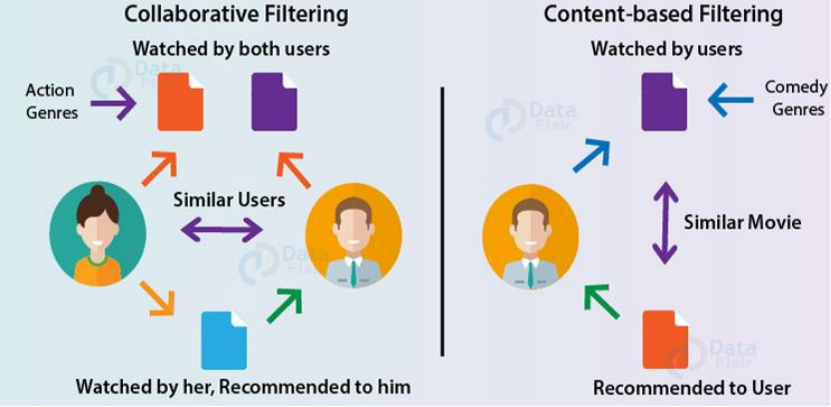
1. Introduction

Recommendation platforms (Youtube, Netflix, Disney Plus...etc) are becoming part of our lives from e commerce suggesting customers goods that could be of interest. Simply put, recommender systems are algorithms designed to suggest relevant items to users (can be movies to watch, articles to read, products to buy or literally anything else depending on the type of industry). Recommendation systems are paramount in some industries for they can produce significant income boost when efficient but also they can be a way to stand out to competitors. The gist aim is for those systems tp produce relevant suggestions for the collection of objects/products that may be of interest to users/customers. General well-known examples of leaders in this realm include suggestions from Amazon books or Netflix. The architecture of such recommendation engines depends on the domain and basic characteristics of the available data.

To help customers find those movies, Google created a movie recommendation system called TensorFlow Recommender (TFRS). Its goal is to predict if a user might enjoy a movie based on how much they liked or disliked other movies. Those predictions are then used to make personal videos recommendations based on each user's unique preferences.

There 3 types of recommendation engines:

- Collaborative filtering.
- Content based Filtering.
- Hybrid (Combination of Collaborative and Content based Filtering).



2. Objectives

This project aims to:

- Design a recommendation engine,
- Differentiate between implicit and explicit feedback and,
- Build a movie recommendation system with TensorFlow and TFRS.

Criteria Of Success

Generating a Movie Recommendation Engine with:

- The highest possible retrieval accuracy (i.e. Predicting Movies) and,
- With the lowest Loss/RMSE (Ranking Movies)

Constraints

TFRS can be classified as a relatively new package (several bugs have been reported depending on the version used). Based on my initial research and asking questions on official TFRS forums, I have decided not to use the latest version (2.7.0) but use an older version (2.5.0) instead that appears more stable for the type of work here

3. Datasets

100k Movielens from TensorFlow is our main dataset for this project. Also, we used both datasets from Movielens website: movies metadata & credits.

MovieLens possesses a set of movie ratings from the MovieLens website, a movie recommendation service. This dataset was collected and maintained by GroupLens , a research group at the University of Minnesota. There are 5 versions included: "25m", "latest-small", "100k", "1m", "20m". In all of the datasets, the movies data and ratings data are joined on "movieId". The 25m dataset, latest-small dataset, and 20m dataset contain only movie data and rating data. The 1m dataset and 100k dataset contain demographic data in addition to movie and rating data.

movie_lens/100k can be treated in two ways:

- It can be interpreted as expressing which movies the users watched (and rated), and which they did not. This is a form of *implicit feedback*, where users' watches tell us which things they prefer to see and which they'd rather not see.
- It can also be seen as expressing how much the users liked the movies they did watch. This is a form of *explicit feedback*: given that a user watched a movie, we can tell roughly how much they liked by looking at the rating they have given.

1) movie_lens/100k-ratings:

- Config description: This dataset contains 100,000 anonymous ratings of approximately 1,682 movies made by 943 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 second 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:
    - 1: "Under 18"
    - 18: "18-24"
    - 25: "25-34"
    - 35: "35-44"
    - 45: "45-49"
    - 50: "50-55"
    - 56: "56+"
- "movie_genres": The Genres of the movies are classified into 21 different classes as below:
    - 0: Action
    - 1: Adventure

    - 2: Animation
    - 3: Children
    - 4: Comedy
    - 5: Crime
    - 6: Documentary
    - 7: Drama
    - 8: Fantasy
    - 9: Film-Noir
    - 10: Horror
    - 11: IMAX
    - 12: Musical
    - 13: Mystery
    - 14: Romance
    - 15: Sci-Fi
    - 16: Thriller
    - 17: Unknown
    - 18: War
    - 19: Western
    - 20: no genres listed

- "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.

- Download size: 4.70 MiB

- Dataset size: 32.41 MiB

- Auto-cached (documentation): No

- Features:

```
FeaturesDict({
              'bucketized_user_age': tf.float32,
              'movie_genres': Sequence(ClassLabel(shape=(), dtype=tf.int64, num_classes=21)),
              'movie_id': tf.string,
              'movie_title': tf.string,
              'raw_user_age': tf.float32,
              'timestamp': tf.int64,
              'user_gender': tf.bool,
              'user_id': tf.string,
              'user_occupation_label': ClassLabel(shape=(), dtype=tf.int64, num_classes=22),
              'user_occupation_text': tf.string,
              'user_rating': tf.float32,
              'user_zip_code': tf.string,
             })
```

2) movie_lens/100k-movies

- Config description: This dataset contains data of approximately 1,682 movies rated in the 100k dataset.
- Download size: 4.70 MiB
- Dataset size: 150.35 KiB
- Auto-cached (documentation): Yes
- Features:

```
FeaturesDict({
              'movie_genres': Sequence(ClassLabel(shape=(), dtype=tf.int64, num_classes=21)),
              'movie_id': tf.string,
              'movie_title': tf.string,
             })
```

2) More information on TFRS

- TensorFlow Recommenders (TFRS) is a library for building recommender system models.
- It helps with the full workflow of building a recommender system: data preparation, model formulation, training, evaluation, and deployment.
- It's built on Keras and aims to have a gentle learning curve while still giving you the flexibility to build complex models.

TFRS makes it possible to:

- Build and evaluate flexible recommendation retrieval models.
- Freely incorporate item, user, and context information into recommendation models.
- Train multi-task models that jointly optimize multiple recommendation objectives.

TFRS is open source and available on **Github**.

To learn more, see the tutorial on how to build a movie recommender system, or check the API docs for the API reference.

4. Data Wrangling

a. Introduction

The **Data wrangling step** focuses on collecting or converting the data, organizing it, and making sure it's well defined. For our project we will be using the ***movie_lens/100k dataset*** from TensorFlow because it's a unique dataset with plenty of Metadata that's needed for this project. We'll focus in particular on:

- Cleaning NANs (If any), duplicate values (If any), wrong values and removing insignificant columns.
- Removing any special characters.
- Renaming some Column labels.
- Correcting some datatypes.

b. Objectives

- Changing the user_gender from booleans "Female" or "Male" to the following association: True:"Male", False:"Female"

- Removing the symbols: (b), (') and (").
- Dropping the following columns: "user_occupation_label" and "movie_genres".
- Changing "timestamp" which is in the unix epoch (units of seconds) to a datetime64 type.
- Fixing any wrong values in "user_zip_code" (removing any zipcode >5 characters & zip codes made out of letters)

5. Exploratory Data Analysis

a. Introduction & Objectives

The **Exploratory Data Analysis (EDA) Step** will focus on:

- To get familiar with the features in our dataset.
- Generally understand the core characteristics of our cleaned dataset.
- Explore the data relationships of all the features and understand how the features compare to the response variable.
- We will think about interesting figures and all the plots that can be created to help deepen our understanding of the data.
- We will be creating one feature that give us the year when the movie was released and will call it "movie_year_release".

b. Preliminary Statistics

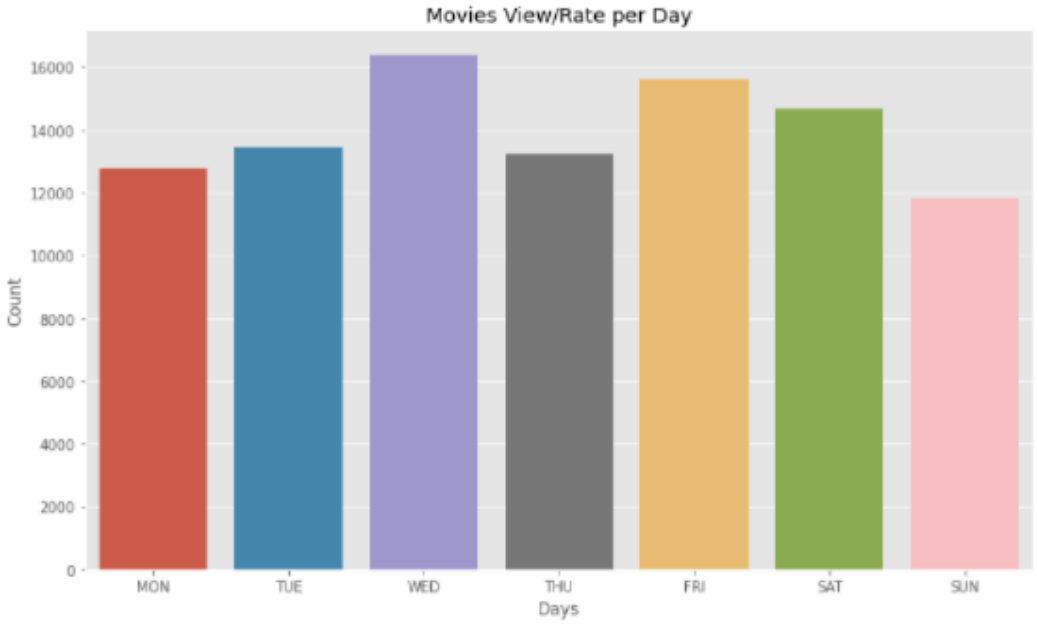
[418]:

	bucketized_user_age	movie_id	raw_user_age	user_id	user_occupation_label	user_rating	user_zip_code
count	97914.0	97914.0	97914.0	97914.0	97914.0	97914.0	97914.0
mean	29.2	425.0	33.0	461.5	11.4	3.5	52179.2
std	12.1	330.4	11.6	265.6	6.5	1.1	30976.4
min	1.0	1.0	7.0	1.0	0.0	1.0	0.0
25%	18.0	175.0	24.0	256.0	6.0	3.0	22902.0
50%	25.0	321.0	30.0	445.0	12.0	4.0	55106.0
75%	35.0	630.8	40.0	682.0	17.0	4.0	80913.0
max	56.0	1681.0	73.0	943.0	21.0	5.0	99835.0

---

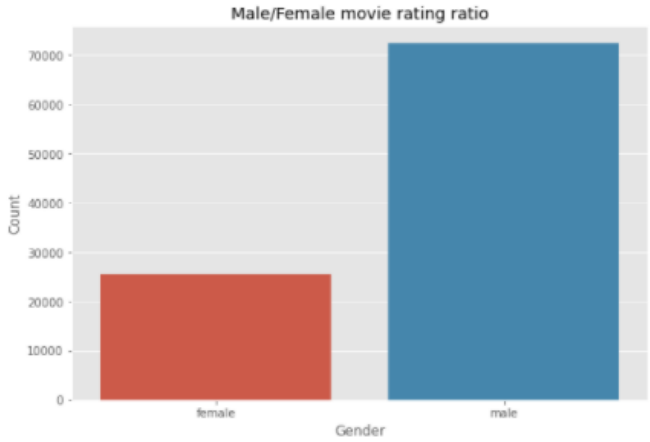
c. What is the most preferrable day to rate/watch movies?

Even though mid-week seems to slightly stand out (Wednesday), overall the distribution is almost equal and the day of the week does not appear to be a factor on to when users watches/rates movies



d. Who, among men and women, watches/rates more movies?

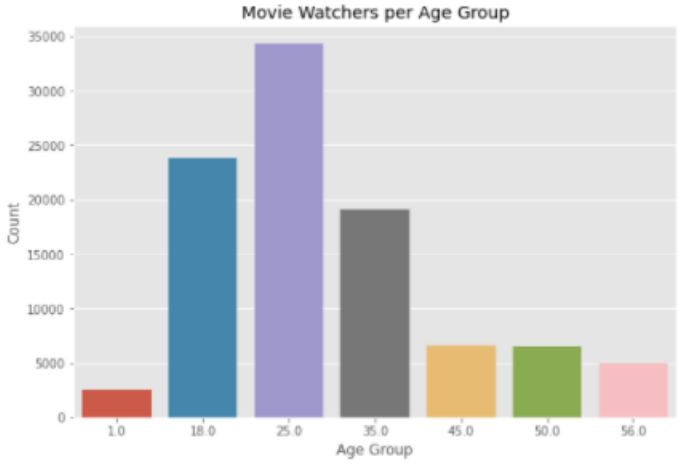
Looks like male users are rating more movies than females



e. What age group watches more movies?

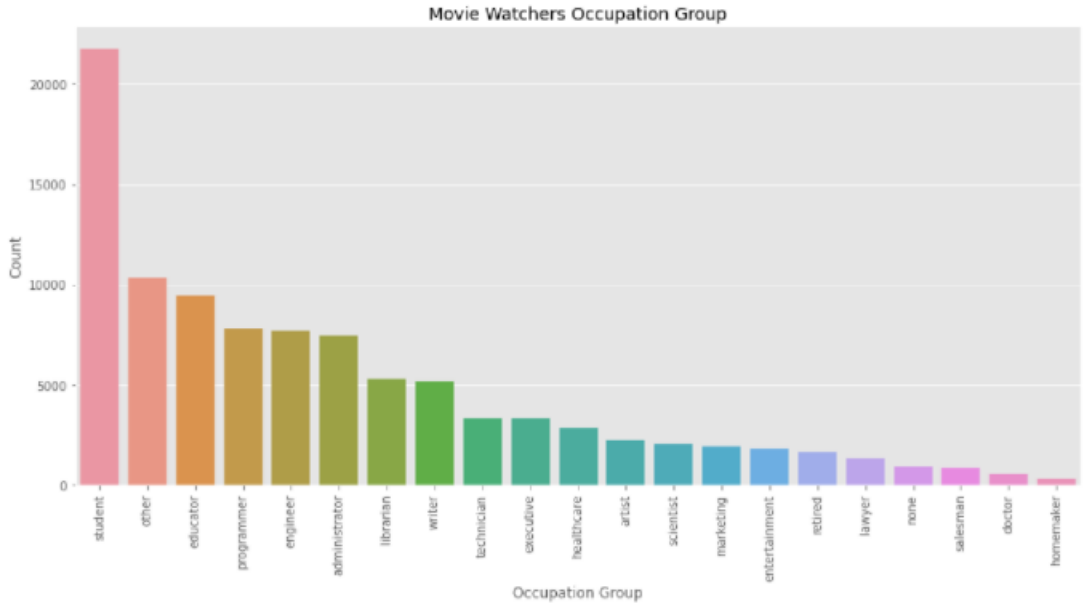
Group Ranking by age for watching/rating movies:

- 25-34
- 18-24
- 35-44



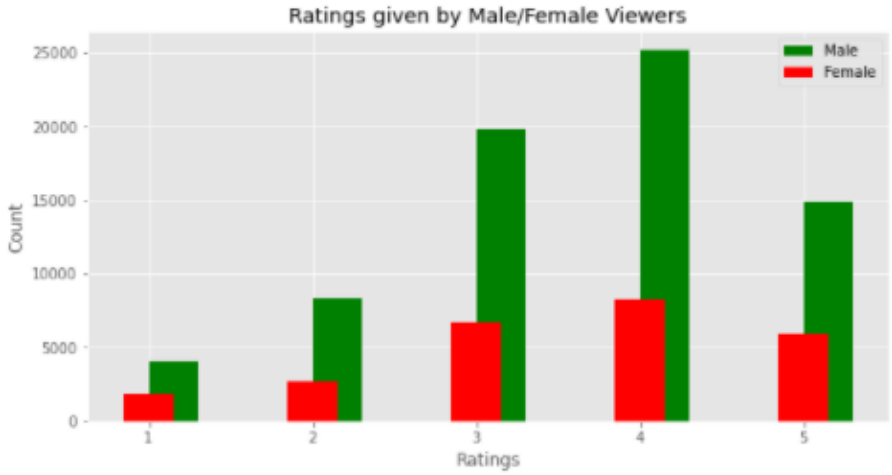
f. What kind of occupation do users have that watch/rate movies the most?

The 18-24 age group for students lead the way

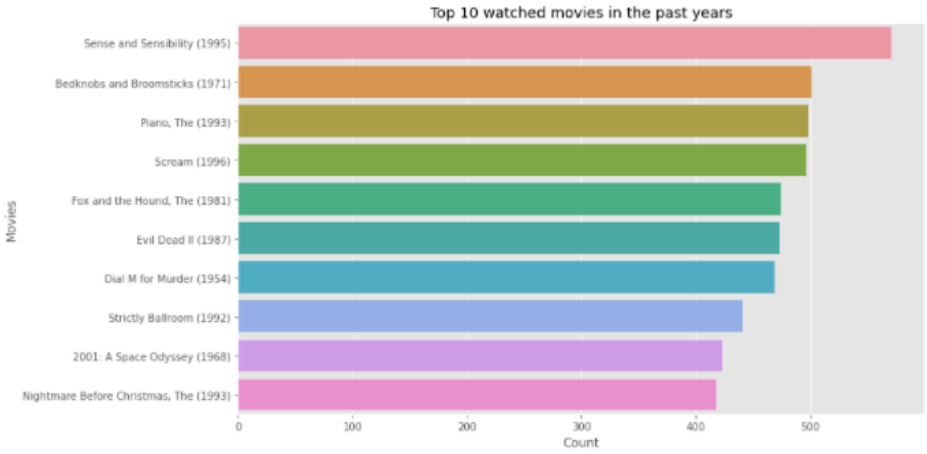


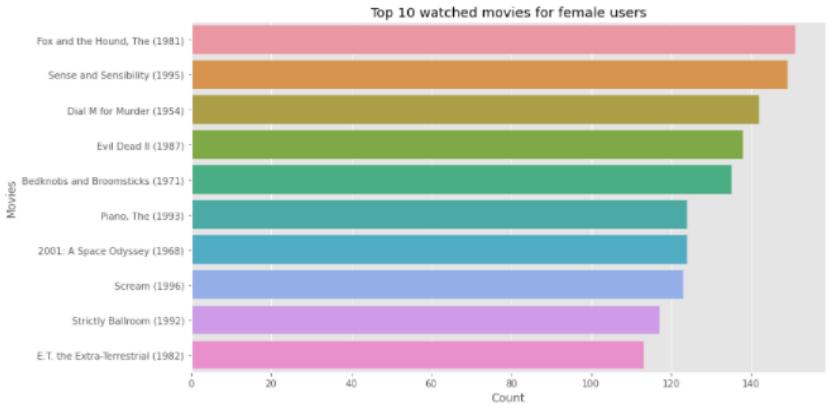
g. Let's have more insights between male and female users

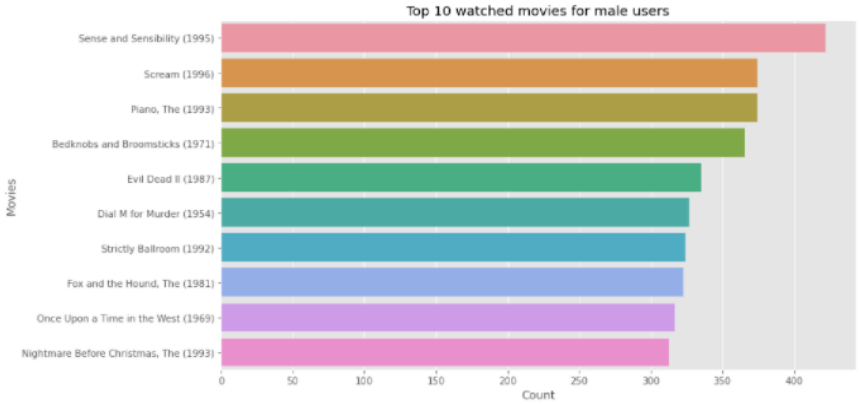
On a rating scale from 1 to 5, both male and female give more "4" ratings

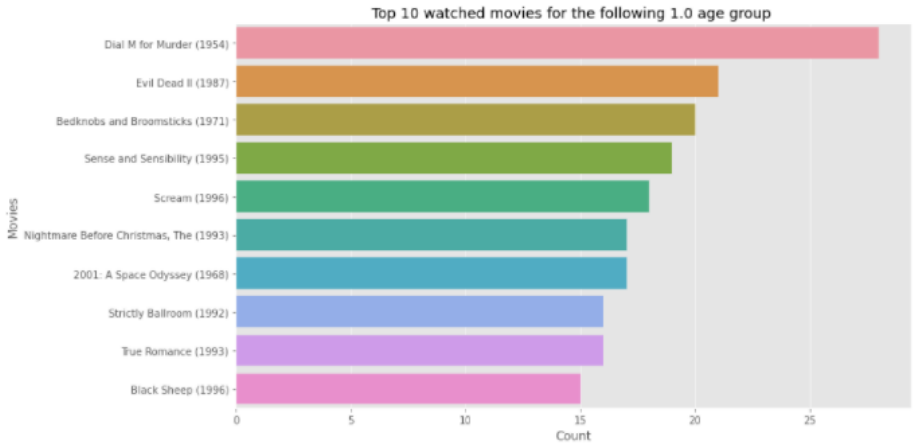


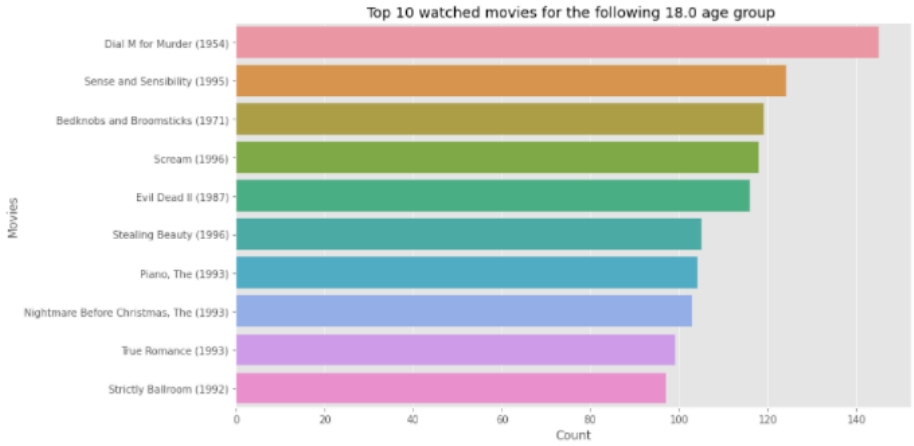
h. What are the most rated movies?

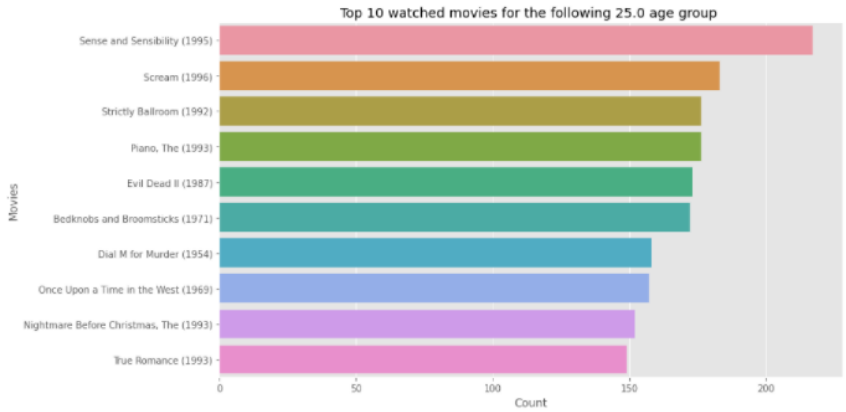


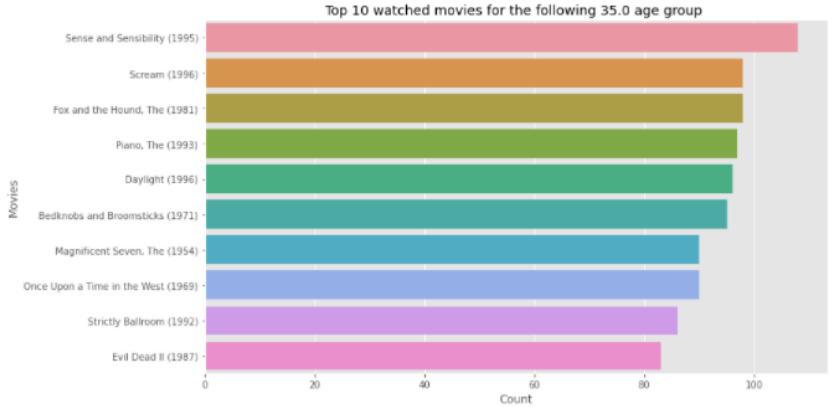


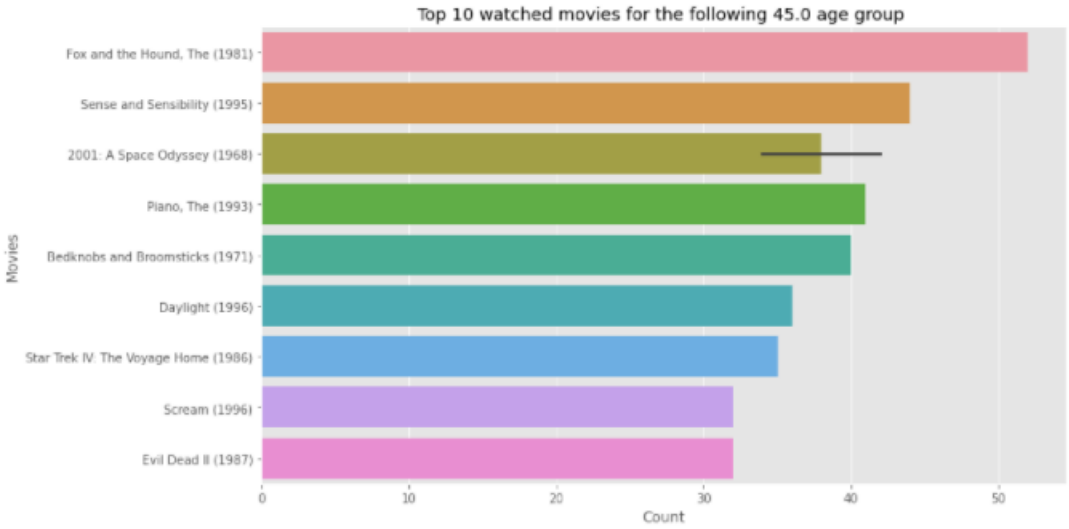


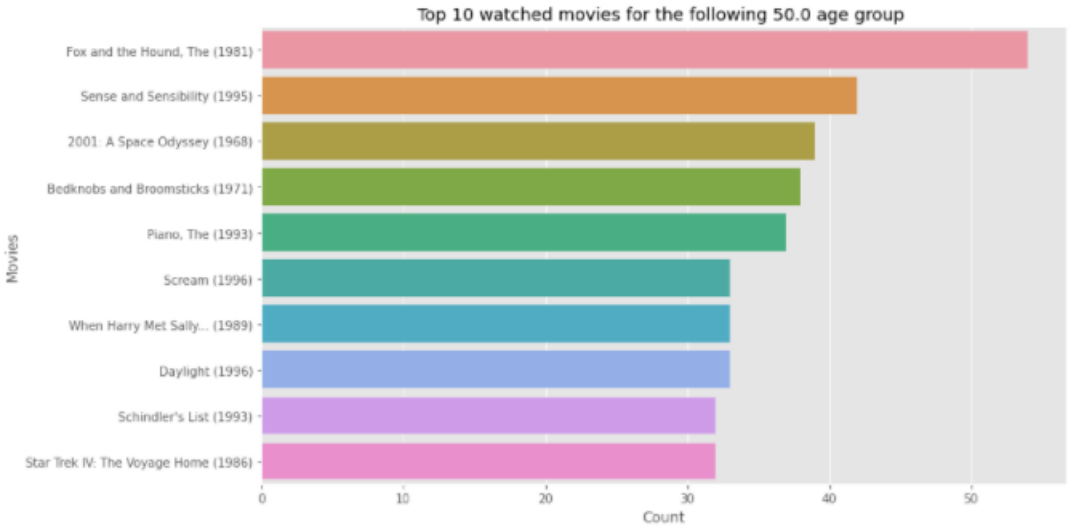

Top 10 watched movies for the following 1.0 age group
Dial M for Murder (1954)
Evil Dead II (1987)
Bedknobs and Broomsticks (1971)
Sense and Sensibility (1995)
Scream (1996)
Nightmare Before Christmas, The (1993)
2001: A Space Odyssey (1968)
Strictly Ballroom (1992)
True Romance (1993)
Black Sheep (1996)
Movies
0
5
10
15
20
25
Count

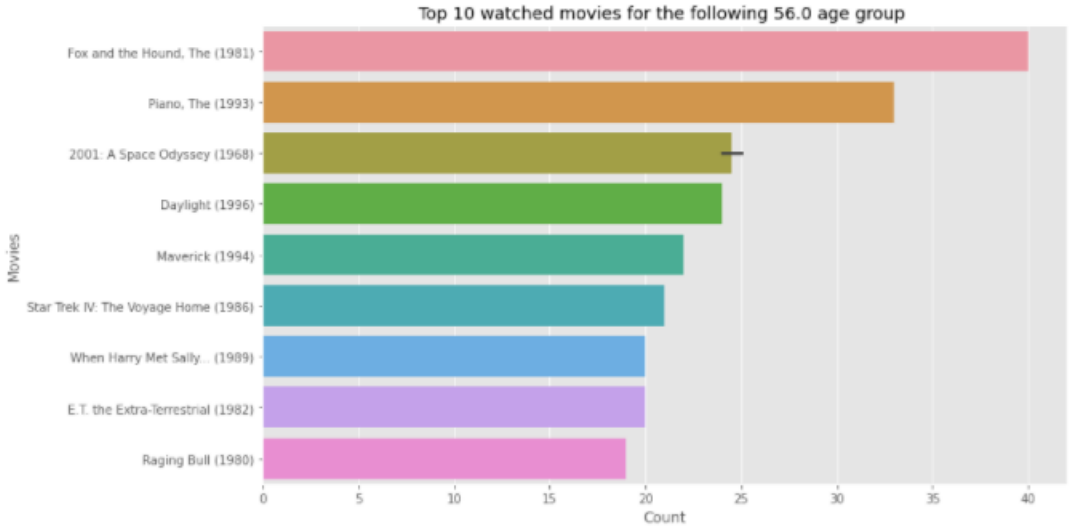

Top 10 watched movies for the following 18.0 age group
Dial M for Murder (1954)
Sense and Sensibility (1995)
Bedknobs and Broomsticks (1971)
Scream (1996)
Evil Dead II (1987)
Stealing Beauty (1996)
Piano, The (1993)
Nightmare Before Christmas, The (1993)
True Romance (1993)
Strictly Ballroom (1992)
Movies
0
20
40
60
80
100
120
140
Count


Top 10 watched movies for the following 25.0 age group
Sense and Sensibility (1995)
Scream (1996)
Strictly Ballroom (1992)
Piano, The (1993)
Evil Dead II (1987)
Bedknobs and Broomsticks (1971)
Dial M for Murder (1954)
Once Upon a Time in the West (1969)
Nightmare Before Christmas, The (1993)
True Romance (1993)
Movies
0
50
100
150
200
Count

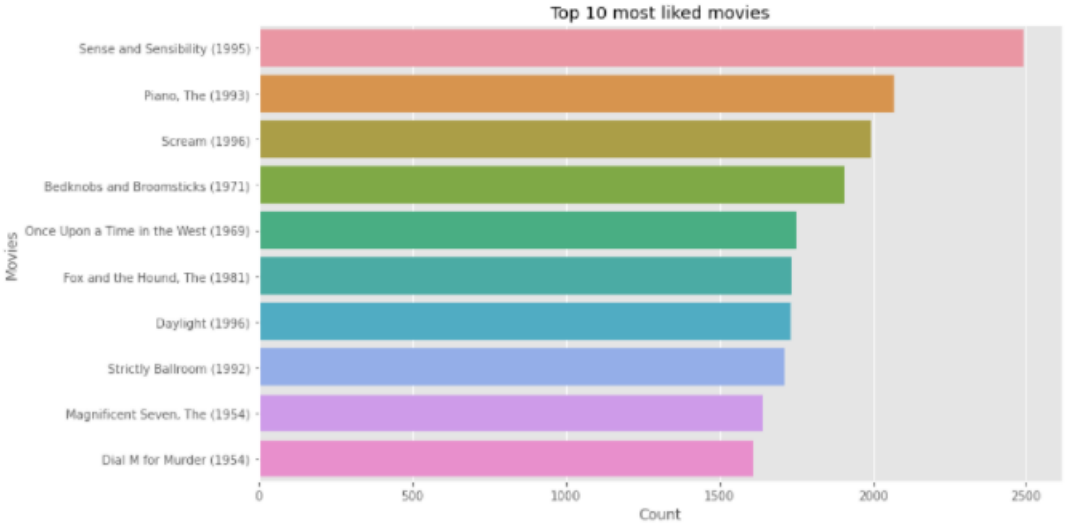

Top 10 watched movies for the following 35.0 age group
Sense and Sensibility (1995)
Scream (1996)
Fox and the Hound, The (1981)
Piano, The (1993)
Daylight (1996)
Bedknobs and Broomsticks (1971)
Magnificent Seven, The (1954)
Once Upon a Time in the West (1969)
Strictly Ballroom (1992)
Evil Dead II (1987)
Movies
0
20
40
60
80
100
Count

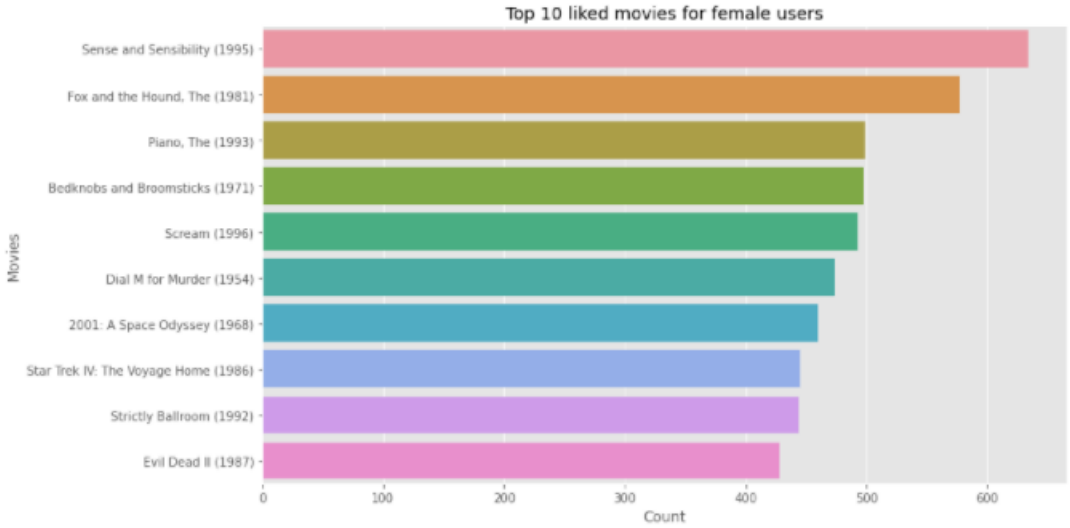

Top 10 watched movies for the following 45.0 age group
Fox and the Hound, The (1981)
Sense and Sensibility (1995)
2001: A Space Odyssey (1968)
Piano, The (1993)
Bedknobs and Broomsticks (1971)
Daylight (1996)
Star Trek IV: The Voyage Home (1986)
Scream (1996)
Evil Dead II (1987)
Movies
0
10
20
30
40
50
Count

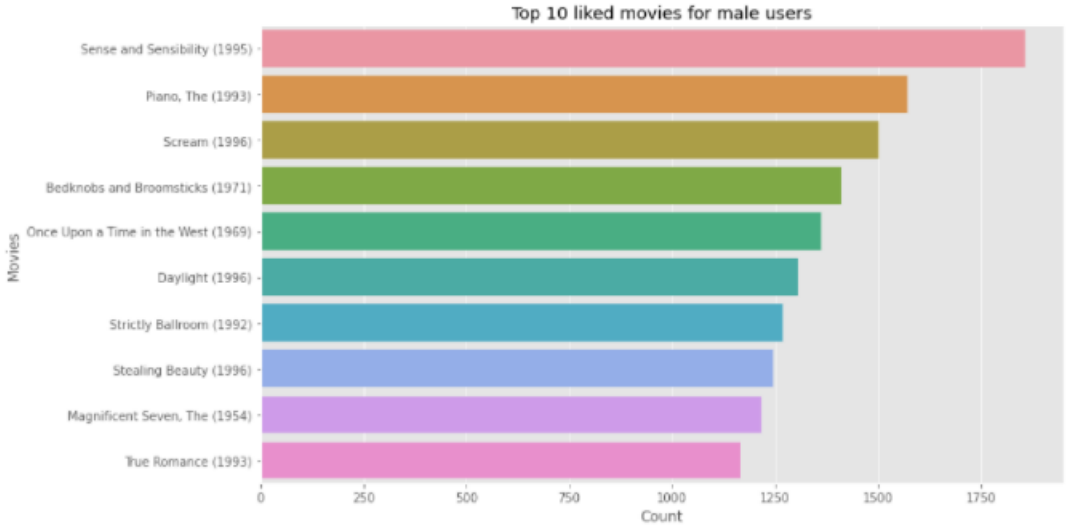

Top 10 watched movies for the following 50.0 age group
Fox and the Hound, The (1981)
Sense and Sensibility (1995)
2001: A Space Odyssey (1968)
Bedknobs and Broomsticks (1971)
Piano, The (1993)
Scream (1996)
When Harry Met Sally... (1989)
Daylight (1996)
Schindler's List (1993)
Star Trek IV: The Voyage Home (1986)
Movies
0
10
20
30
40
50
Count

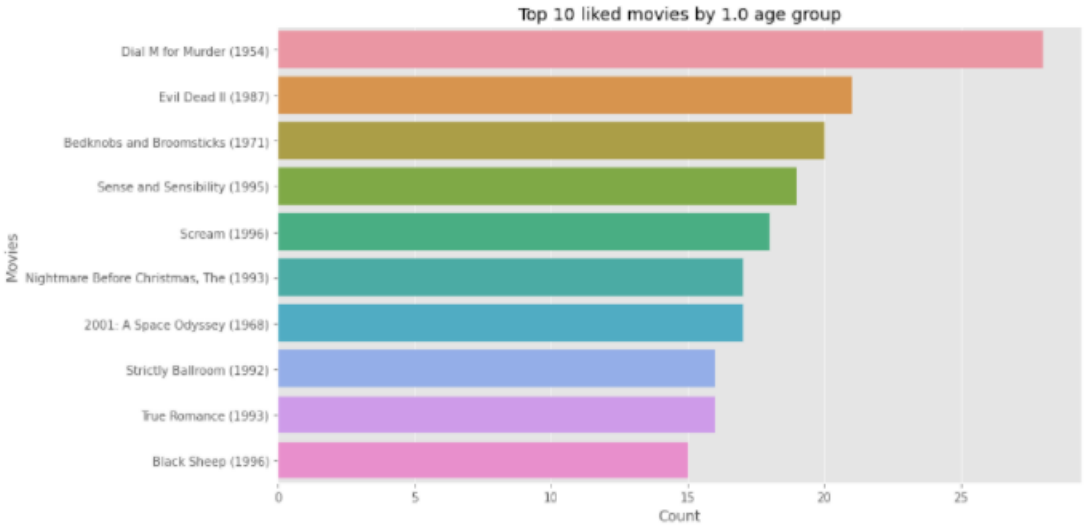

Top 10 watched movies for the following 56.0 age group
Fox and the Hound, The (1981)
Piano, The (1993)
2001: A Space Odyssey (1968)
Daylight (1996)
Maverick (1994)
Star Trek IV: The Voyage Home (1986)
When Harry Met Sally... (1989)
E.T. the Extra-Terrestrial (1982)
Raging Bull (1980)
Movies
0
5
10
15
20
25
30
35
40
Count

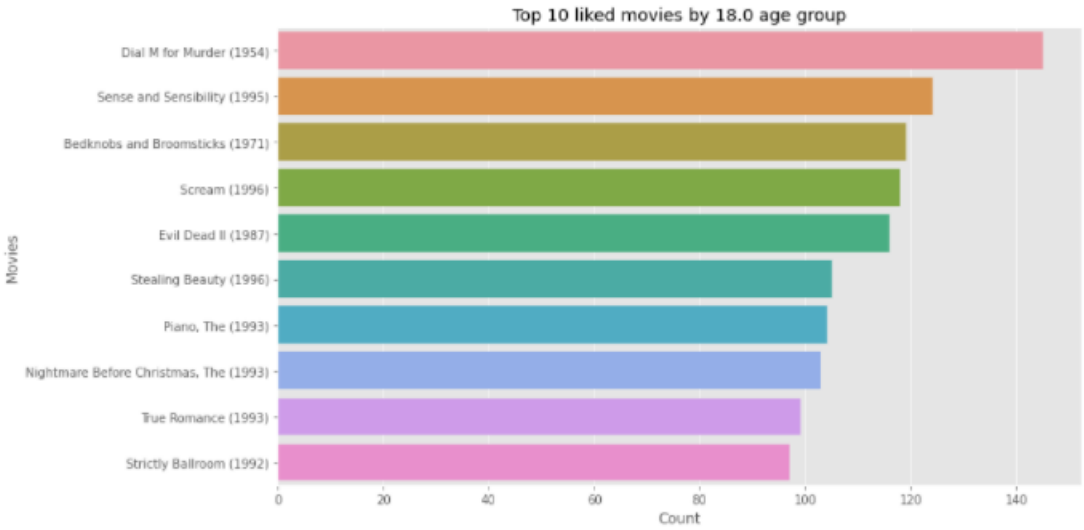
i. What are the most liked movies?

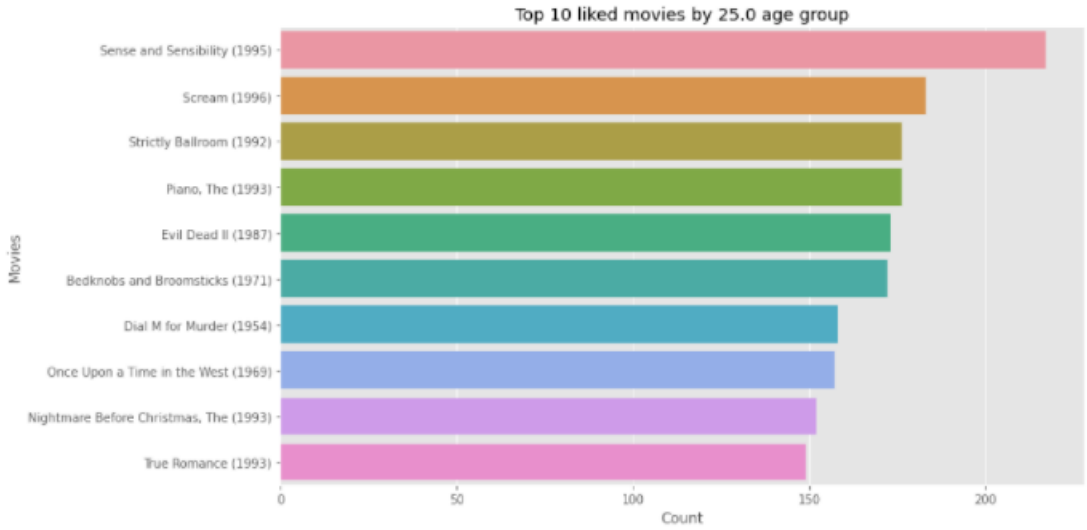


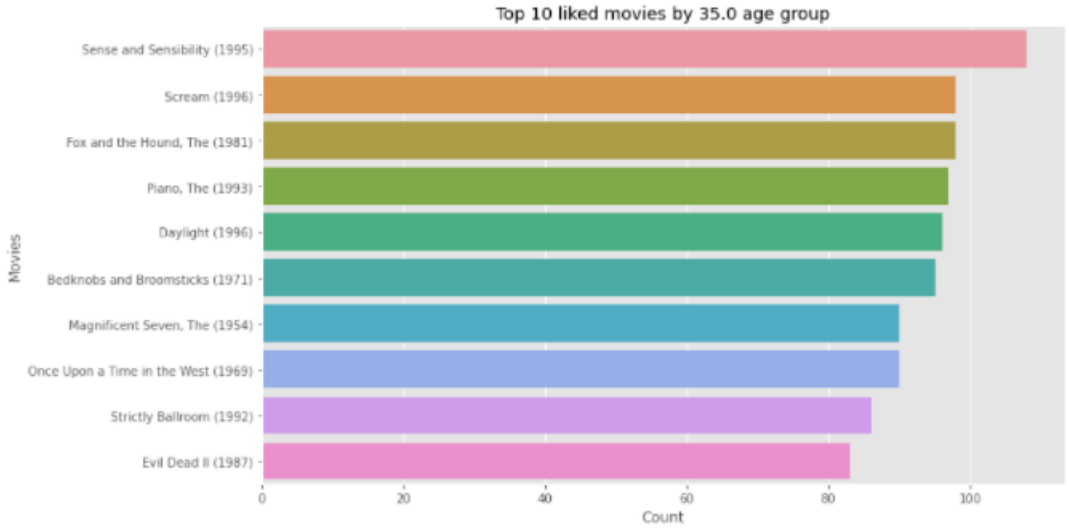


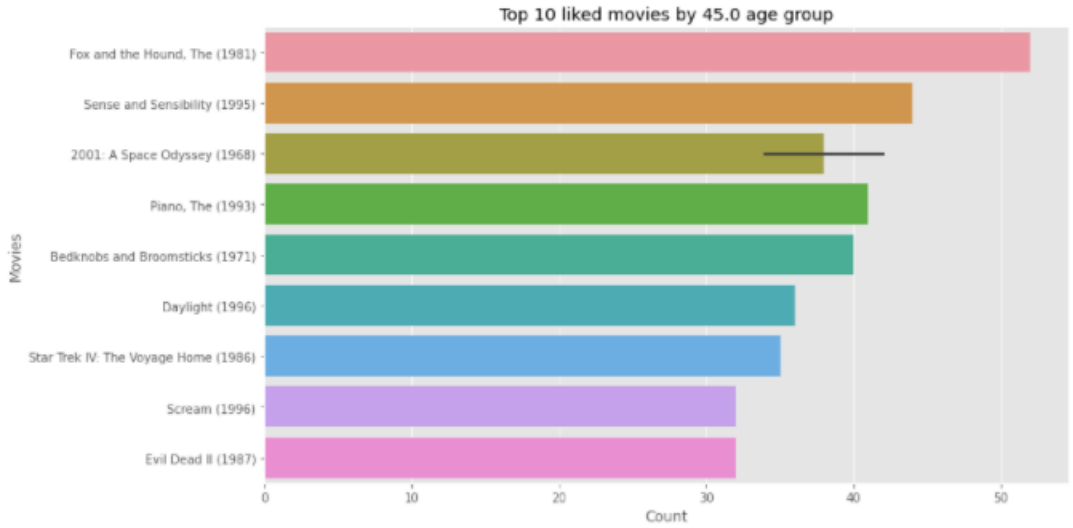


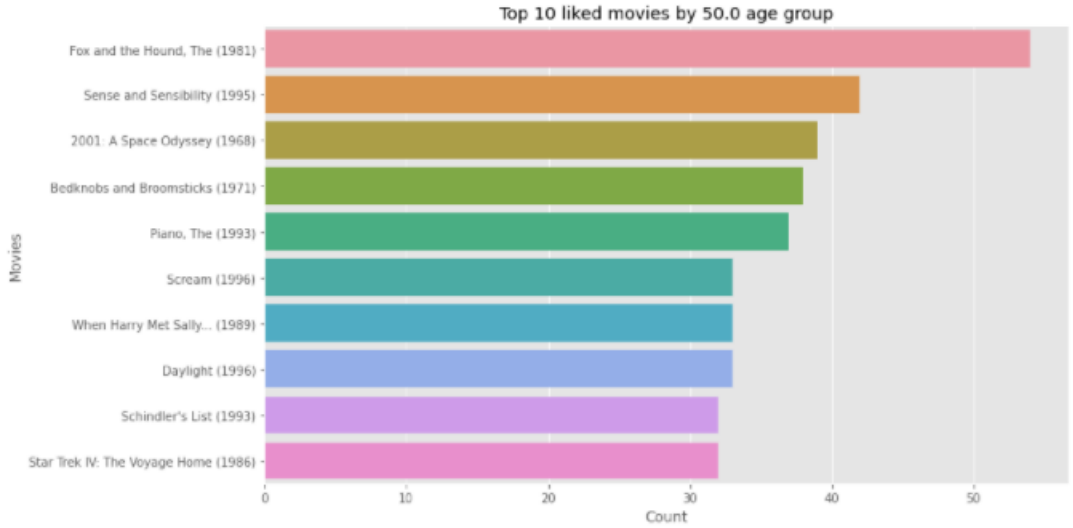

Top 10 liked movies by 1.0 age group
Dial M for Murder (1954)
Evil Dead II (1987)
Bedknobs and Broomsticks (1971)
Sense and Sensibility (1995)
Scream (1996)
Nightmare Before Christmas, The (1993)
2001: A Space Odyssey (1968)
Strictly Ballroom (1992)
True Romance (1993)
Black Sheep (1996)
Movies
0
5
10
15
20
25
Count

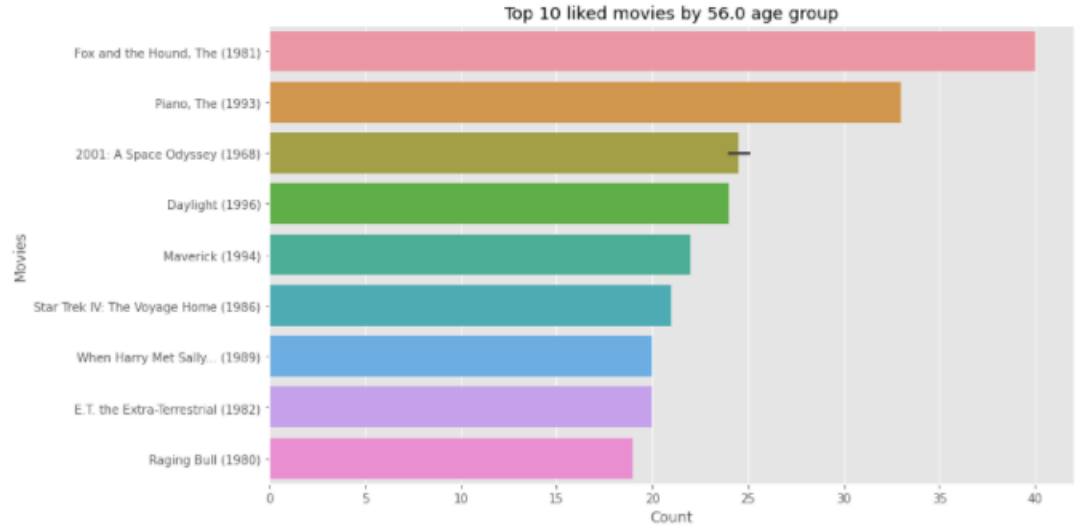

Top 10 liked movies by 18.0 age group
Dial M for Murder (1954)
Sense and Sensibility (1995)
Bedknobs and Broomsticks (1971)
Scream (1996)
Evil Dead II (1987)
Stealing Beauty (1996)
Piano, The (1993)
Nightmare Before Christmas, The (1993)
True Romance (1993)
Strictly Ballroom (1992)
Movies
0
20
40
60
80
100
120
140
Count


Top 10 liked movies by 25.0 age group
Sense and Sensibility (1995)
Scream (1996)
Strictly Ballroom (1992)
Piano, The (1993)
Evil Dead II (1987)
Bedknobs and Broomsticks (1971)
Dial M for Murder (1954)
Once Upon a Time in the West (1969)
Nightmare Before Christmas, The (1993)
True Romance (1993)
Movies
0
50
100
150
200
Count


Top 10 liked movies by 35.0 age group
Sense and Sensibility (1995)
Scream (1996)
Fox and the Hound, The (1981)
Piano, The (1993)
Daylight (1996)
Bedknobs and Broomsticks (1971)
Magnificent Seven, The (1954)
Once Upon a Time in the West (1969)
Strictly Ballroom (1992)
Evil Dead II (1987)
Movies
0
20
40
60
80
100
Count


Top 10 liked movies by 45.0 age group
Fox and the Hound, The (1981)
Sense and Sensibility (1995)
2001: A Space Odyssey (1968)
Piano, The (1993)
Bedknobs and Broomsticks (1971)
Daylight (1996)
Star Trek IV: The Voyage Home (1986)
Scream (1996)
Evil Dead II (1987)
Movies
0
10
20
30
40
50
Count

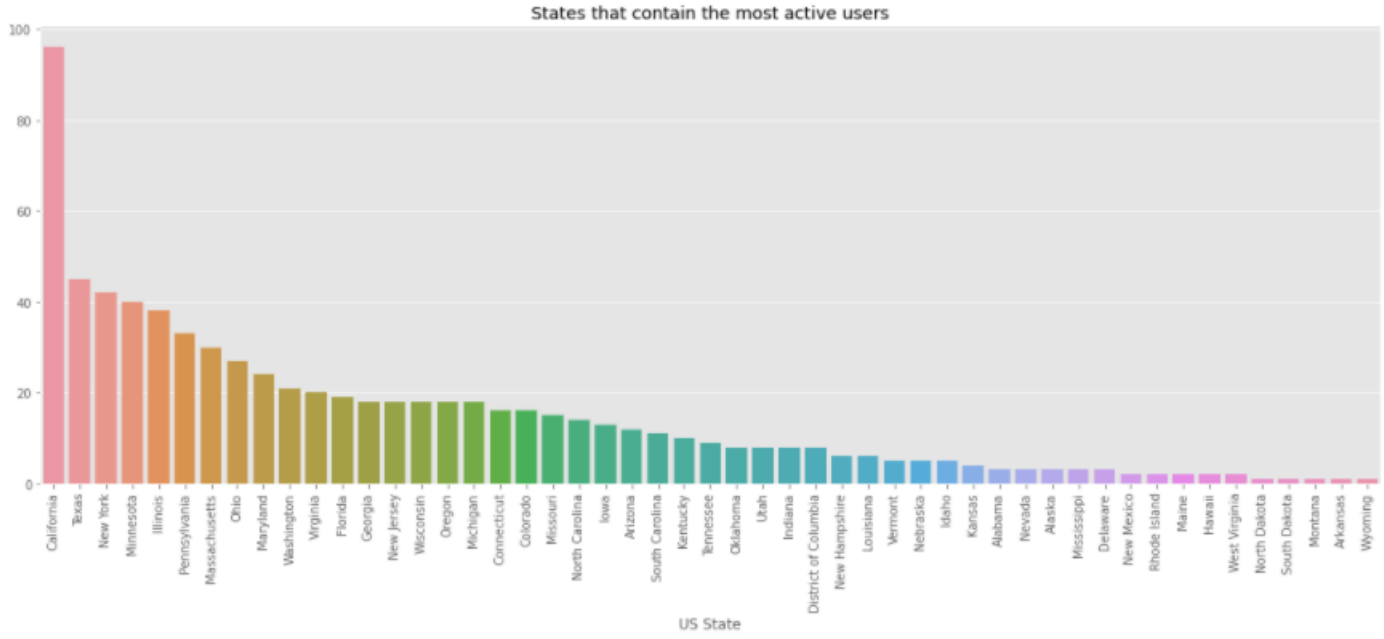

Top 10 liked movies by 50.0 age group
Fox and the Hound, The (1981)
Sense and Sensibility (1995)
2001: A Space Odyssey (1968)
Bedknobs and Broomsticks (1971)
Piano, The (1993)
Scream (1996)
When Harry Met Sally... (1989)
Daylight (1996)
Schindler's List (1993)
Star Trek IV: The Voyage Home (1986)
Movies
0
10
20
30
40
50
Count


Top 10 liked movies by 56.0 age group
Fox and the Hound, The (1981)
Piano, The (1993)
2001: A Space Odyssey (1968)
Daylight (1996)
Maverick (1994)
Star Trek IV: The Voyage Home (1986)
When Harry Met Sally... (1989)
E.T. the Extra-Terrestrial (1982)
Raging Bull (1980)
Movies
0
5
10
15
20
25
30
35
40
Count

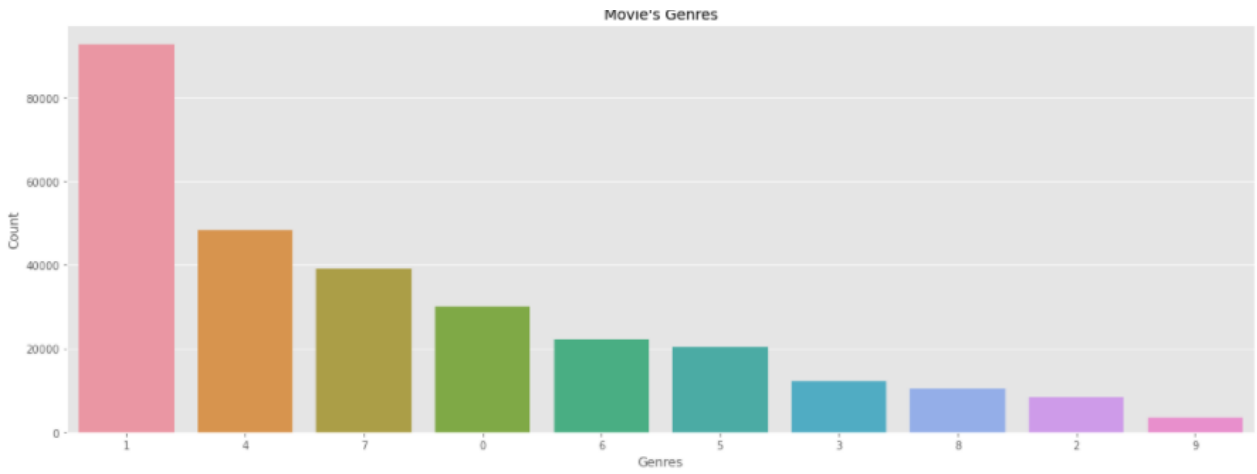
j. What are the worst movies per rating?

k. Is there any relationship between the user's rate and their geographical location?

California and Texas lead the way. Are the students the group who's also causing such high rates? Indeed, students are the group driving the rating in California



I. What is the most popular genre in our dataset?



We can infer from the above graph that the 3 top most popular Genres are Adventure, Comedy and Drama:

The Genres of the movies are classified into 21 different classes as below:

- 0: Action
- 1: Adventure
- 2: Animation
- 3: Children
- 4: Comedy
- 5: Crime
- 6: Documentary
- 7: Drama
- 8: Fantasy
- 9: Film-Noir
- 10: Horror
- 11: IMAX
- 12: Musical
- 13: Mystery
- 14: Romance
- 15: Sci-Fi
- 16: Thriller
- 17: Unknown
- 18: War
- 19: Western
- 20: no genres listed

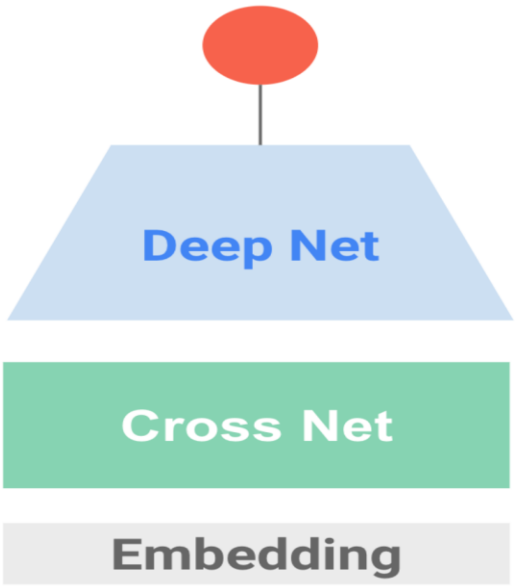
6. Machine Learning Modeling using Tensorflow Recommenders (TFRS)

a. Introduction

For this final part, we will be dealing with:

- The Feature importance using Deep and Cross Network (DCN-v2)
- Training multiple TensorFlow Recommenders.
- Applying hyperparameters tuning where applicable to ensure every algorithm will result in the best prediction possible.
- Finally, evaluating these Models.

b. Features Importance using Deep & Cross Network (DCN-V2)



b.1. Deep and cross network (DCN) came out of Google Research, and is designed to learn explicit and bounded-degree cross features effectively

- Large and sparse feature space is extremely hard to train.
- Oftentimes, we needed to do a lot of manual feature engineering, including designing cross features, which is very challenging and less effective.
- Whilst possible to use additional neural networks under such circumstances, it's not the most efficient approach. DCN is specifically designed to tackle all of the above challenges.

b.2. Feature Cross

Let's say we're building a recommender system to sell a blender to customers. Then our customers' past purchase history, such as purchased apples and purchased recipes books, or geographic features are single features. If one has purchased both apples and recipes books, then this customer will be more likely to click on the recommended blender. The combination of purchased apples and the purchased recipes books is referred to as feature cross, which provides additional interaction information beyond the individual features.

b.3. Cross Network

In real world recommendation systems, we often have large and sparse feature space. So, identifying effective feature processes in this setting would often require manual feature engineering or exhaustive search, which is highly inefficient. To tackle this issue, Google Research team has proposed DCN. It starts with an input layer, typically an embedded layer, followed by a cross network containing multiple cross layers that models explicitly feature interactions, and then combines with a deep network that models implicit feature interactions. The deep network is just a traditional multilayer construction. But the core of DCN is really the cross network. It explicitly applies feature crossing at each layer. And the highest polynomial degree increases with layer depth.

b.4. Deep & Cross Network

There are a couple of ways to combine the cross network and the deep network:

- Stack the deep network on top of the cross network.
- Place deep & cross networks in parallel.

b.5. Model Structure

We first train a DCN model with a stacked structure, that is, the inputs are fed to a cross network followed by a deep network.

b.6. Model construction

- The model architecture we will be building starts with an embedded layer, which is fed into a cross network followed by a deep network.
- The embedded dimension is set to 32 for all the features.
- Please note that we could also have used different embedded sizes for different features.

b.7. Model Training

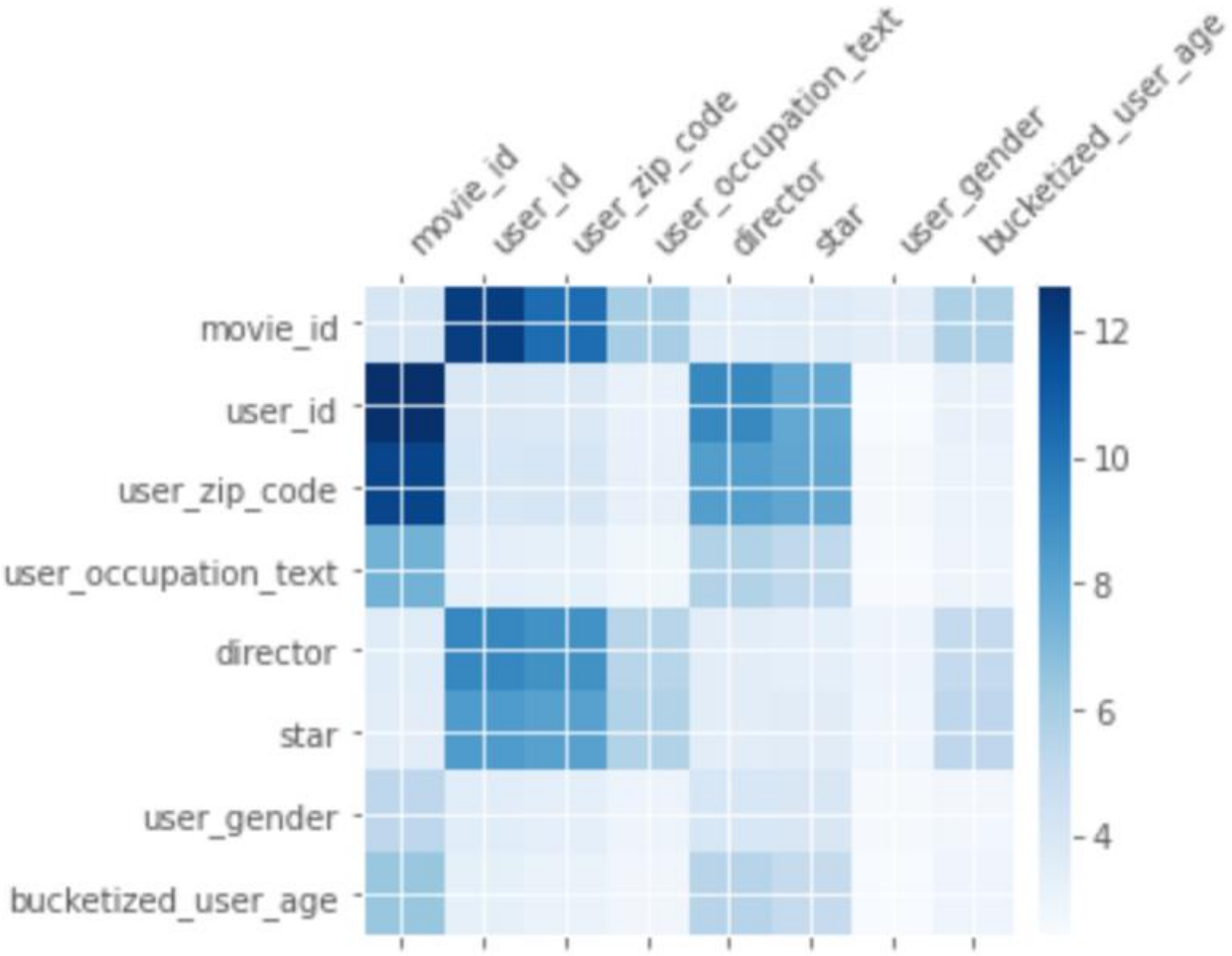
We set some hyper-parameters for the models.

- Note that these hyper-parameters are set globally for all the models for demonstration purpose.
- If you want to obtain the best performance for each model, or to conduct a fair comparison amongst models, I would suggest you to fine-tune the hyper-parameters.
- Remember that the model architecture and optimization schemes are intertwined.

b.8. DCN (stacked)

From the below graph, we can visualize the weights from the cross network and see if it has successfully learned the important feature process.

As shown above, for instance, the feature cross of user ID and movie ID are of great importance.



b.9. Low-rank DCN

To reduce the training and running cost, we leverage low-rank techniques to approximate the DCN weight matrices.

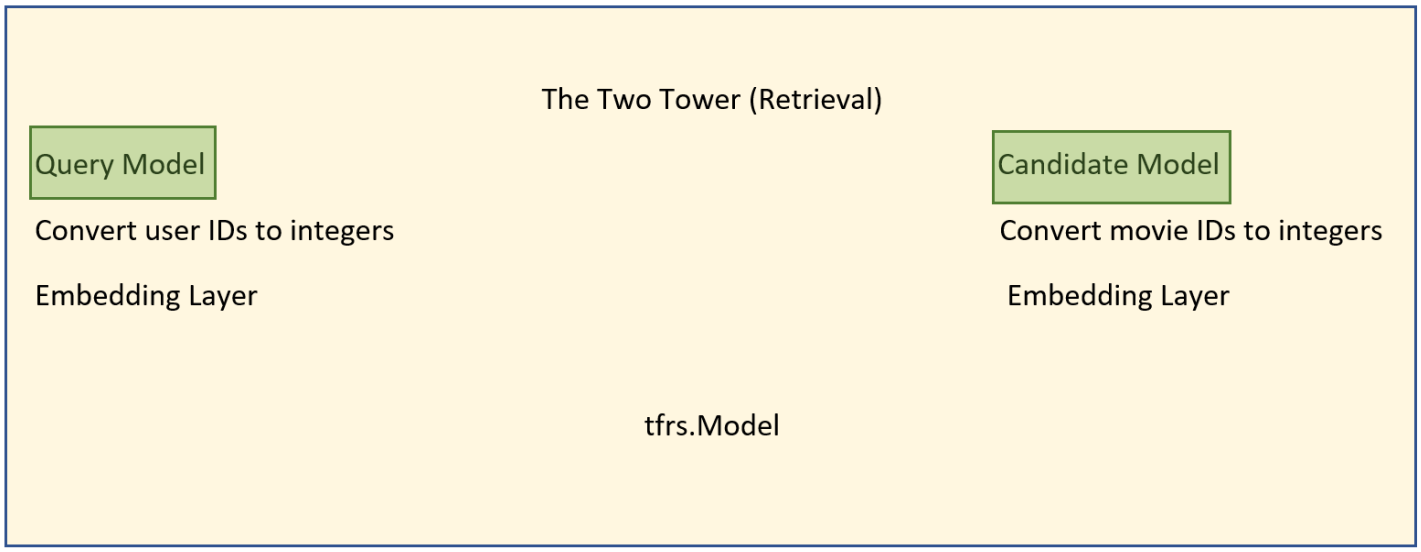
- The rank is passed in through the projection_dim argument: a smaller projection_dim results in a lower cost.
- Note that projection_dim needs to be smaller than the (input size)/2 to reduce the cost. In practice, we've observed that using a low-rank DCN with a rank of (input size)/4 consistently preserved the accuracy of a full-rank DCN.

b.10. DNN (Cross Layer = False)

DCN RMSE mean: 0.9811,

- stdv: 0.0241
- DCN (low-rank) RMSE mean: 0.9666, stdv: 0.0128
- DNN RMSE mean: 0.9531, stdv: 0.0115

c. The Two-Tower & Ranking Models: Baseline vs. Tuned Joint Model



c.1. Real-world recommender systems are often composed of two stages:

- *The retrieval stage* (Selects recommendation candidates): is responsible for selecting an initial set of hundreds of candidates from all possible candidates.
    - The main objective of this model is to efficiently weed out all candidates that the user is not interested in. Because the retrieval model may be dealing with millions of candidates, it has to be computationally efficient.
- *The ranking stage* (Selects the best candidates and rank them): takes the outputs of the retrieval model and fine-tunes them to select the best possible handful of recommendations.
    - Its task is to narrow down the set of items the user may be interested in to a shortlist of likely candidates.

c.2. Retrieval models are often composed of two sub-models:

The retrieval model embeds user ID's and movie ID's of rated movies into embedded layers of the same dimension:

- A query model computing the query representation (normally a fixed-dimensionality embedding vector) using query features.
- A candidate model computing the candidate representation (an equally-sized vector) using the candidate features.
- As shown below, the two models are multiplied to create a query-candidate affinity scores for each rating during training. If the affinity score for the rating is higher than other candidates, then we can consider the model to be reasonable.

c.3. Embedded layer Magic

As discussed above, we might think of the embedded layers as just a way of encoding a way of forcing the categorical data into some sort of a standard format that can be easily fed into a neural network and usually that's how it's used but embedded layers are more than that!

The way they're working under the hood is every unique id is being mapped to a vector of n dimensions, but it's going to be like a vector of 32 floating point values and we can think of this as a position in a 32-dimensional space that represents the similarity between one user id and another or between one movie id and another so by using embeddied layers in this way we're getting around that whole problem of data sparsity and sparse vectors and at the same time, we're getting a measure of similarity so it's a very simple way of getting recommendation candidates.

The outputs of the two models are then multiplied together to give a query-candidate affinity score, with higher scores expressing a better match between the candidate and the query.

In this Model, we built and trained such a two-tower model using the Movielens dataset (100k Dataset):

- Getting our data and splitting it into a training and test set.
- Implementing a retrieval model.
- Fitting and evaluating it.

c.4. The Ranking

The ranking stage takes the outputs of the retrieval model and fine-tunes them to select the best possible handful of recommendations.

- Its task is to narrow down the set of items the user may be interested in to a shortlist of likely candidates.

c.5. Case 1: Baseline

There are two critical parts to multi-task recommenders:

- They optimize for two or more objectives, and so have two or more losses.
- They share variables between the tasks, allowing for transfer learning.

Now, let's define our models as before, but instead of having a single task, we will have two tasks: one that predicts ratings, and one that predicts movie watches.

Putting it together

- We put it all together in a model class.
- The new component here is that - since we have two tasks and two losses - we need to decide on how important each loss is. We can do this by giving each of the losses a weight, and treating these weights as hyperparameters.
    - If we assign a large loss weight to the rating task, our model is going to focus on predicting ratings (but still use some information from the retrieval task); if we assign a large loss weight to the retrieval task, it will focus on retrieval instead.

c.5.1. Rating-specialized model

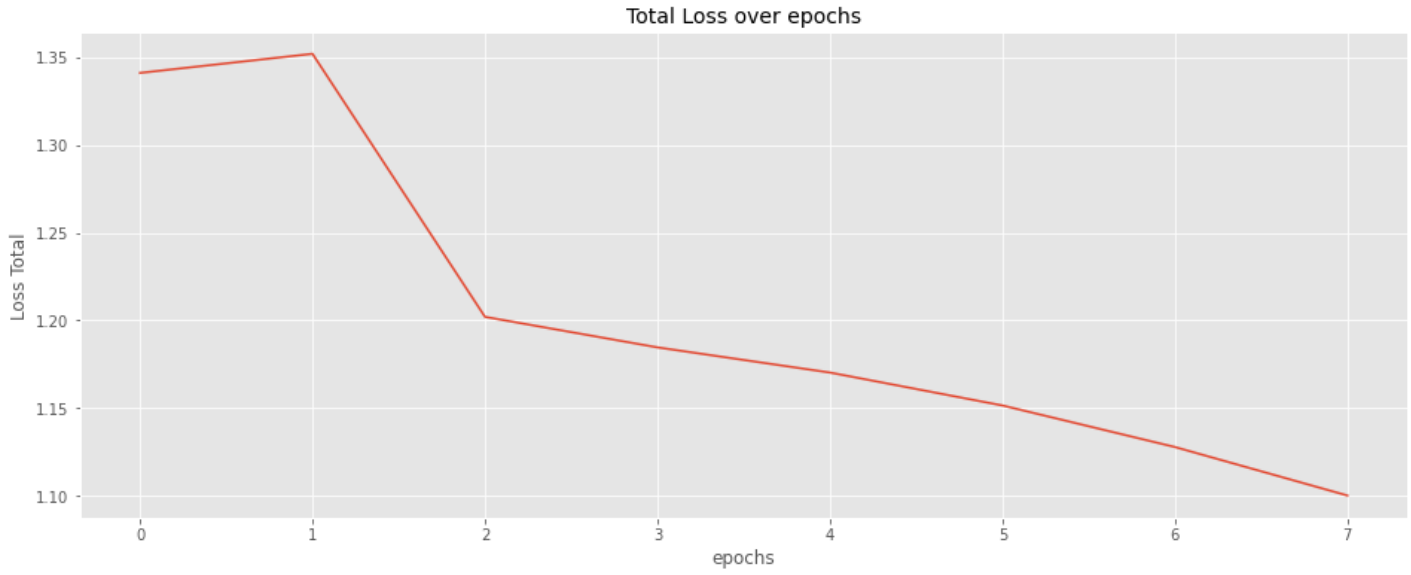
Depending on the weights we assign, the model will encode a different balance of the tasks. Let's start with a model that only considers ratings.

As the model trains, the loss is falling and a set of top-k retrieval metrics is updated.

- These tell us whether the true positive is in the top-k retrieved items from the entire candidate set.
- For example, a top-5 categorical accuracy metric of 0.2 would tell us that, on average, the true positive is in the top 5 retrieved items 20% of the time.

11/11 [==============================] - 2s 110ms/step - root_mean_squared_error: 1.0711 - factorized_top_k/top_1_categorical_accuracy: 0.0723 - factorized_top_k/top_5_categorical_accuracy: 0.1022 - factorized_top_k/top_10_categorical_accuracy: 0.1153 - factorized_top_k/top_50_categorical_accuracy: 0.1852 - factorized_top_k/top_100_categorical_accuracy: 0.2359 - loss: 1.1437 - regularization_loss: 0.0000e+00 - total_loss: 1.1437 Retrieval top-100 accuracy: 0.236. Ranking RMSE: 1.071.

We get a good RMSE but with poor prediction



c.5.2. Retrieval-specialized model

Let's now try a model that focuses on retrieval only.

11/11 [==============================] - 2s 134ms/step - root_mean_squared_error: 3.6912 - factorized_top_k/top_1_categorical_accuracy: 3.1586e-04 - factorized_top_k/top_5_categorical_accuracy: 6.7683e-04 - factorized_top_k/top_10_categorical_accuracy: 0.0013 - factorized_top_k/top_50_categorical_accuracy: 0.0229 - factorized_top_k/top_100_categorical_accuracy: 0.0578 - loss: 15094.9468 - regularization_loss: 0.0000e+00 - total_loss: 15094.9468 Retrieval top-100 accuracy: 0.058. Ranking RMSE: 3.691.

We get a less impressive RMSE coupled with a poor prediction too

c.5.3. Joint Model: Baseline

Let's now train a model that assigns positive weights to both tasks.

Models	Retrieval top-10 accuracy	Retrieval top-100 accuracy	Ranking RMSE
Model: Rating-specialized model	0.1153	0.236	1.071
Model: Retrieval-specialized model	0.0013	0.058	3.691
Model: Joint model: Baseline	0.1474	0.285	1.087

The joint model seems to provide an overall better prediction than the other independent models What can we improve upon the joint model?

- Adding more features
- Optimize Embedding
- embedding_dimension
- epochs= 8 to 16
- Learning rate

c.6. Case 2: Tuned Joint Model

As before, this task's goal is to predict which movies the user will or will not watch.

Those folowwing classes were re-configured to accomodate the new embedded design due to new features, having deeper neural networks and adding regularization to help overfitting:

- class UserModel
- class QueryModel
- class MovieModel
- class MovieModel
- class CandidateModel
- class MovielensModel

Let's now train a model that assigns positive weights to both tasks.

Models	Retrieval top-10 accuracy	Retrieval top-100 accuracy	Ranking RMSE
Model: Rating-specialized model	0.1153	0.236	1.071
Model: Retrieval-specialized model	0.0013	0.058	3.691
Model: Joint model: Baseline	0.1474	0.285	1.087
Model: Tuned Joint model (Best Model for teh Recommendation)	0.9544	0.96	1.11

The Case 2 results look very promising even though more fine-tuning can be done to further reduce the Ranking RMSE

7.Conclusion & Future Work

- Further optimizing existing Tensorflow Recommenders Classes to accommodate extra features.
- Additional fine-tuning needed to enhance retrieval prediction and reduce RMSE Ranking from the Joint Baseline Model (Multi-Task)
- Building an application