# **CSE 482 Class Project**

**Student Name**: Mounika Jetti

**Topic**: Development of a recommender system Web site

## **Movie Recommendation System**

Modern tech and retail companies make extensive use of intricate recommendation systems to enhance customer experience.

* 35% of Amazon's revenue is generated by it's recommendation system ([source](https://www.rejoiner.com/resources/amazon-recommendations-secret-selling-online)).
* For Netflix, 80% of movies that people watch are based on some sort of recommendation ([source](https://towardsdatascience.com/deep-dive-into-netflixs-recommender-system-341806ae3b48)).

The objective of a recommendation system is to tailor the experience of a service to suit the taste of a particular customer. This becomes even more relevant with online video content services like Netflix and Youtube with colossal digital libraries. For example, Youtube would offer a particular individual a unique view to its vast library personalized based on geography, language, etc.

**Primary Objective:** Our objective is to design a robust movie recommendation system using ratings data available at [MovieLens website](https://movielens.org/).

**Secondary Objective:**  Deal with the case when information regarding a user’s past behavior is not available, the cold-start problem.

## **Data Exploration and Visualization**

We will use the benchmark [MovieLens 25M dataset](https://grouplens.org/datasets/movielens/). We might choose to use a small sample of the dataset for some experiments as it was difficult to run collaborative filtering techniques on the full version.

The dataset contains 25 million ratings and one million tag applications across 62,000 movies. These data points were created by 162,000 users.

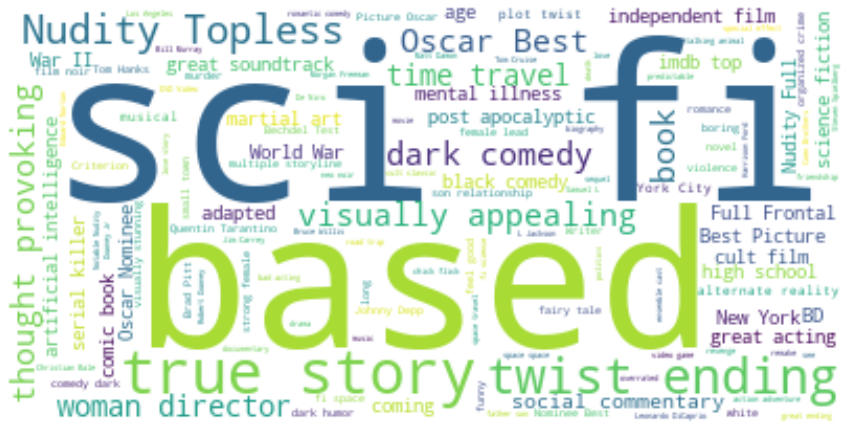
The datasets describe ratings and free-text tagging activities from [MovieLens](http://movielens.org), a movie recommendation service.

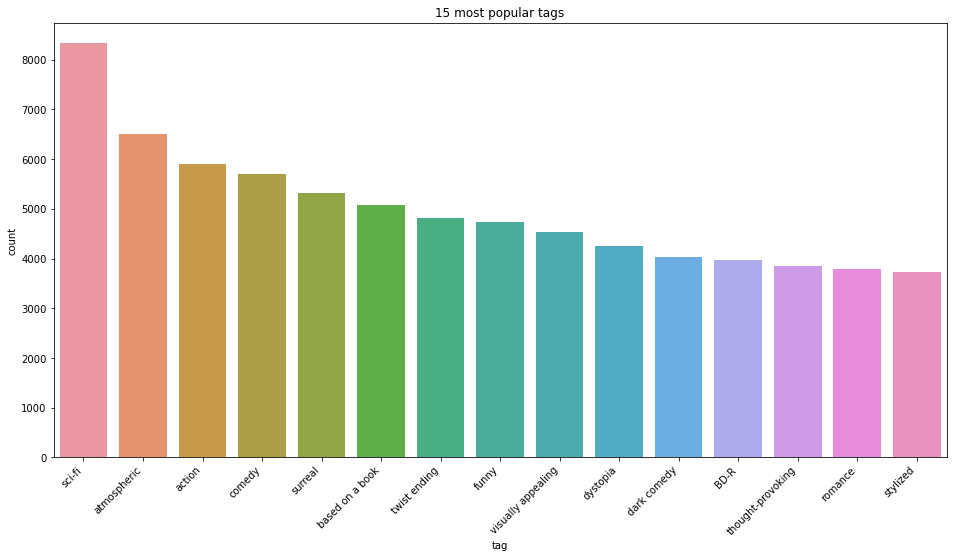
*Source: F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Transactions on Interactive Intelligent Systems (TiiS) 5, 4: 19:1–19:19.* [*https://doi.org/10.1145/2827872*](https://doi.org/10.1145/2827872)

The dataset is distributed among four csv files: links.csv, movies.csv, ratings.csv, tags.csv.

Tags seem to provide us a little bit insight about how the user felt about the movie.

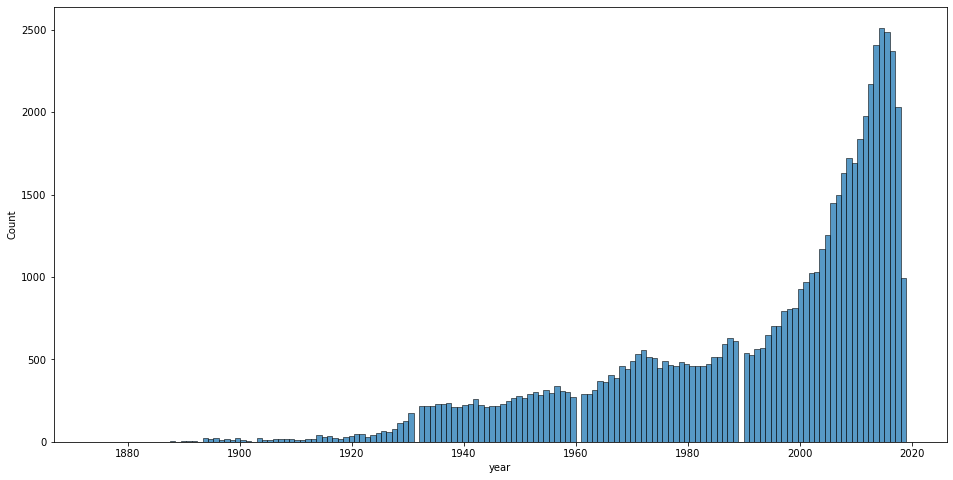
We can analyze these tags using NLP to further understand the overall sentiment about the movie.

****

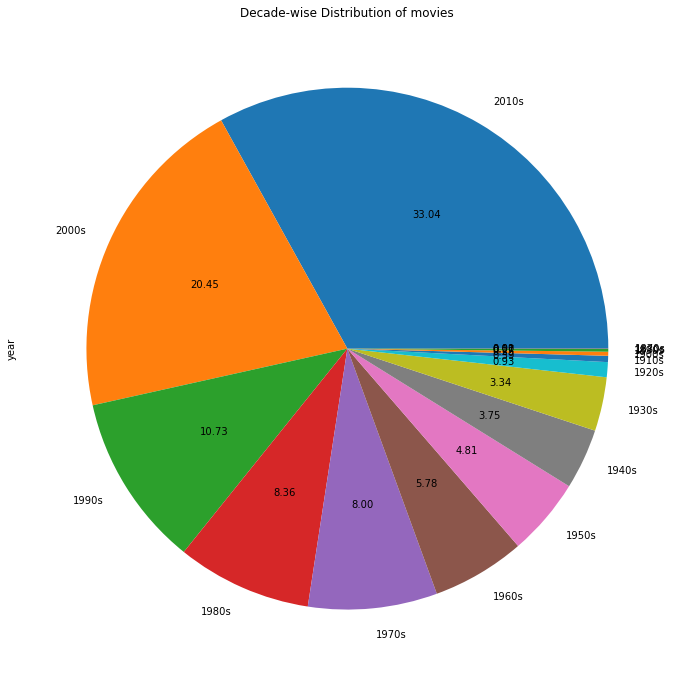


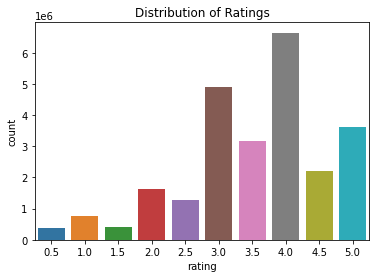
While I do believe that there is useful information available in tags data, I think utilizing it will require incorporating advanced NLP techniques which currently is outside the scope of this project. But will definitely revisit this in future iterations.

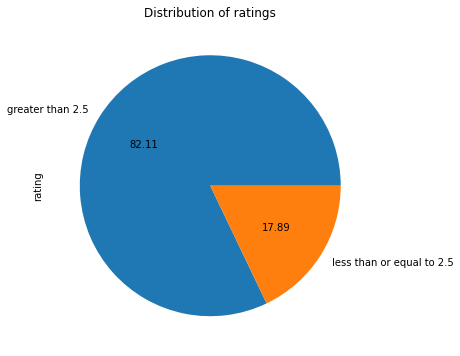
The graph shows that the number of movies released every year keeps on increasing with 2015 being the year with the highest number of releases at 2513 !!!



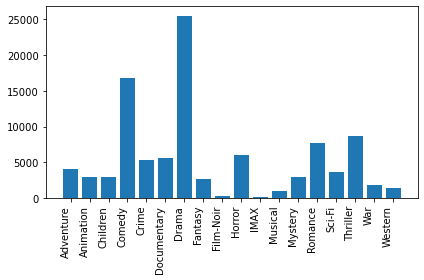
As expected, movies from 2000s and 2010s are going to dominate our dataset. This might add some bias in our engine towards movies from these 2 decades.

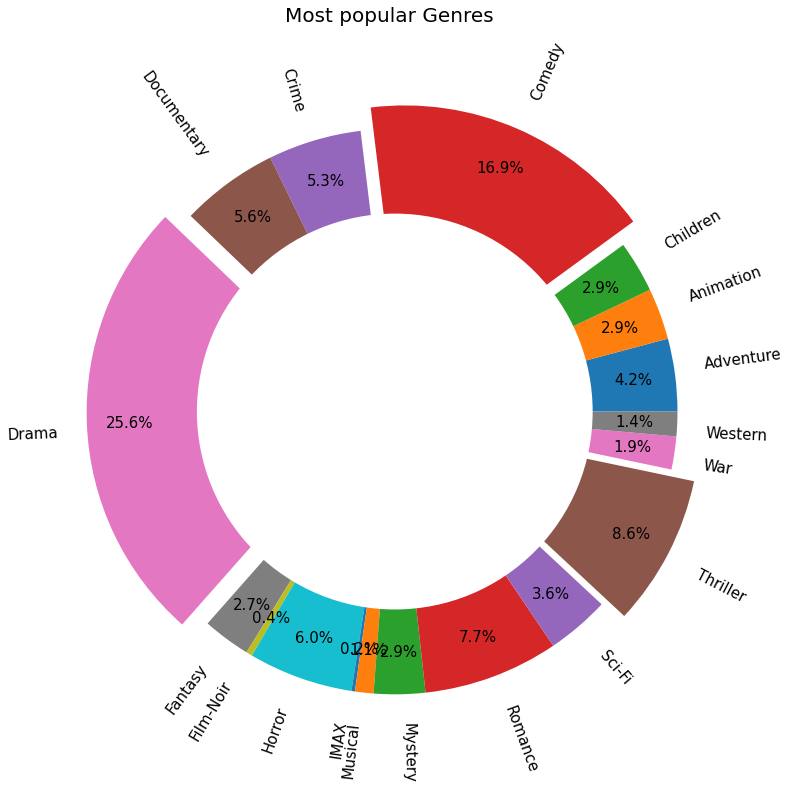






Drama is the most popular genre with 25,523 movies, followed by Comedy with 16,823 movies.





## **Popularity-based Recommendation Engine**

Probably the simplest and the most well used, popularity based recommendation systems recommend content/product based on trends in the overall database.

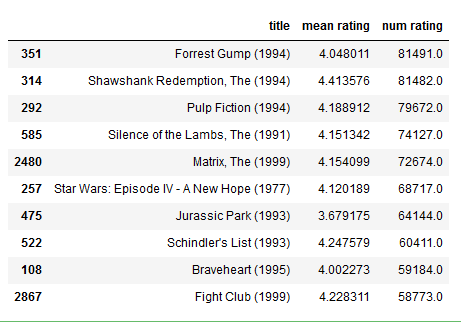
While this system does not tailor recommendations to a particular user, it provides a good solution to deal with the cold-start problem.

For example, a model can recommend the top 10 most popular movies (movies with most number of ratings) in our database to a new user.

```

movie\_ratings.sort\_values(by=['num rating'], ascending=False).head(10)

```



As expected, all listed movies are internationally acclaimed hollywood classics. We can add more filters like that we want movies that were released after 2014.

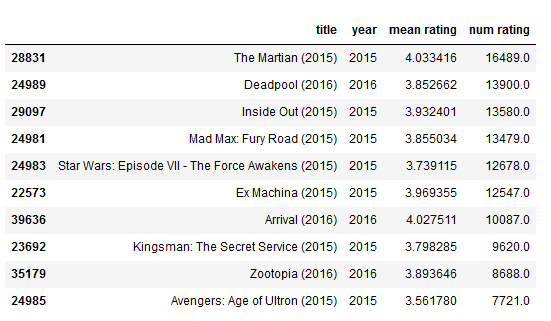
```

movie\_ratings = movies\_data[['title','year','mean rating','num rating']]

movie\_ratings = movie\_ratings[movie\_ratings['year']>2014]

movie\_ratings.sort\_values(by=['num rating'], ascending=False).head(10)

```



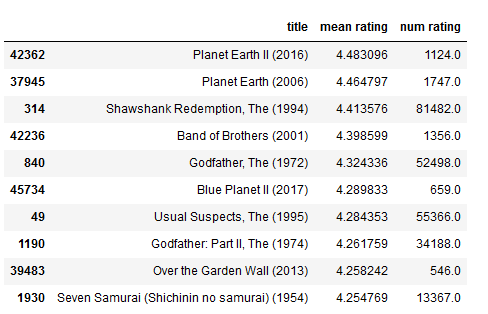
While these movies are rated quite high, they are not popular and only have 1 rating. This is not reliable information. We need to set a threshold of the minimum number of ratings a movie must have.

```

minimum\_num\_ratings = 500

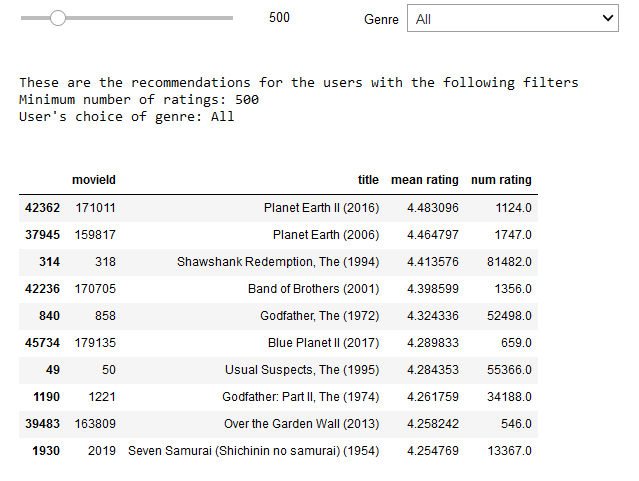
movie\_ratings[movie\_ratings['num rating']>minimum\_num\_ratings].sort\_values(by=['mean rating'], ascending=False).head(10)

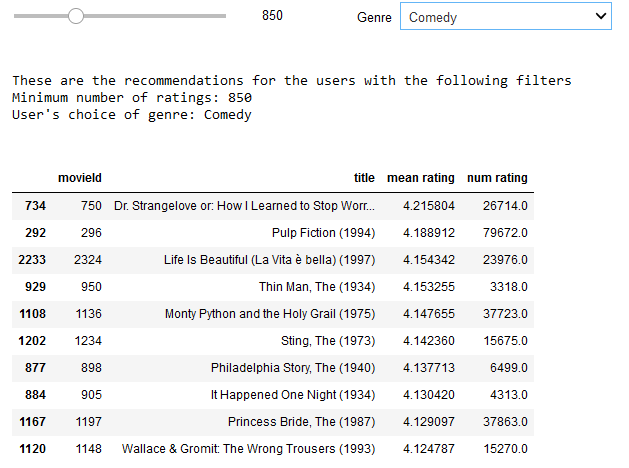
```



These recommendations seem a lot more robust now. These movies are both popular and well loved. Interestingly the top 2 movies are documentaries.

Genre can also be used for further finetuning. While eventually we would want to understand a user's preferred genre by building a taste profile, currently we will allow the user to make a selection. Whatever we have seen so far can be combined together to build a very simple popularity based recommendation engine. Here we will allow the user to set different filters like minimum number of ratings and genre.





Observations for Popularity based

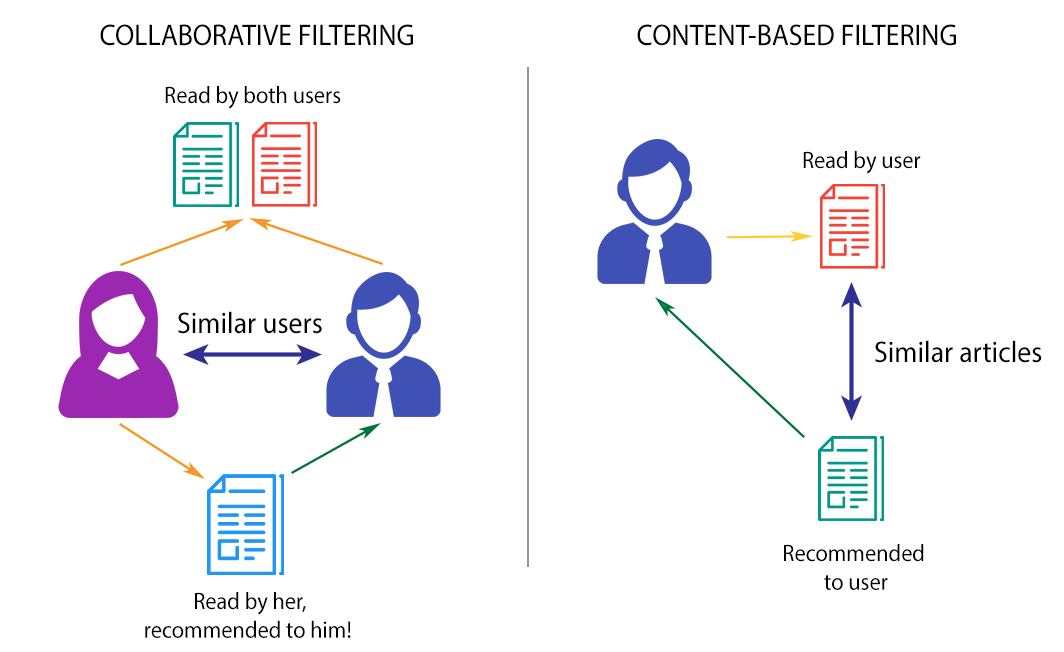
## **Collaborative Filtering**

In content based filtering, we look at the movies that a user has liked in the past and then recommend him/her similar movies.

In collaborative filtering, we take into accounts a user's past. Let's say User A likes three movies - Iron Man, Avengers, and Captain America. User B also likes three movies - Iron Man, Avengers, and Thor.

Then the assumption here is that User A and B are similar or have similar interests and we can recommend Thor to User A, whereas Captain America to User B.

This is how collaborative filtering is performed.



Most modern recommendation systems use a hybrid of content and collaborative based model. But, in our case as we do not contain a lot of metadata about the movie (cast, director, etc), we chose to focus on collaborative filtering.

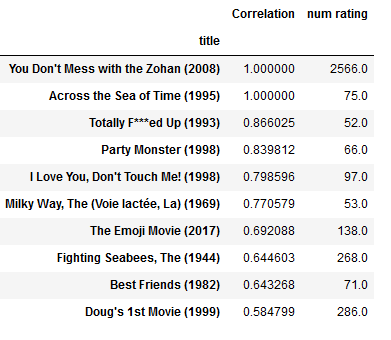
There are two kinds of collaborative filtering: **Item-based** and **User-based**.

### **Item based Collaborative Filtering using Distance Metrics**

In Item based collaborative filtering, we explore the relationship between the items. So, we will look for items that have received similar ratings in our database.

We can use multiple distance/similarity metrics to find the relationship between the movies. We will first use correlation in order to find the similarity between the movies.

We will pick the 2008 comedy movie - 'You Don't Mess with the Zohan (2008)'.



Most of these recommendations don't make any sense to me. I tried different distance metrics but none of them resulted in better results. To improve results in item based aprroach, we need to add more information about the items.

## **User based Collaborative Filtering**

Finally we are going to use our much discussed user based collaborative filtering where we try to find similar users.

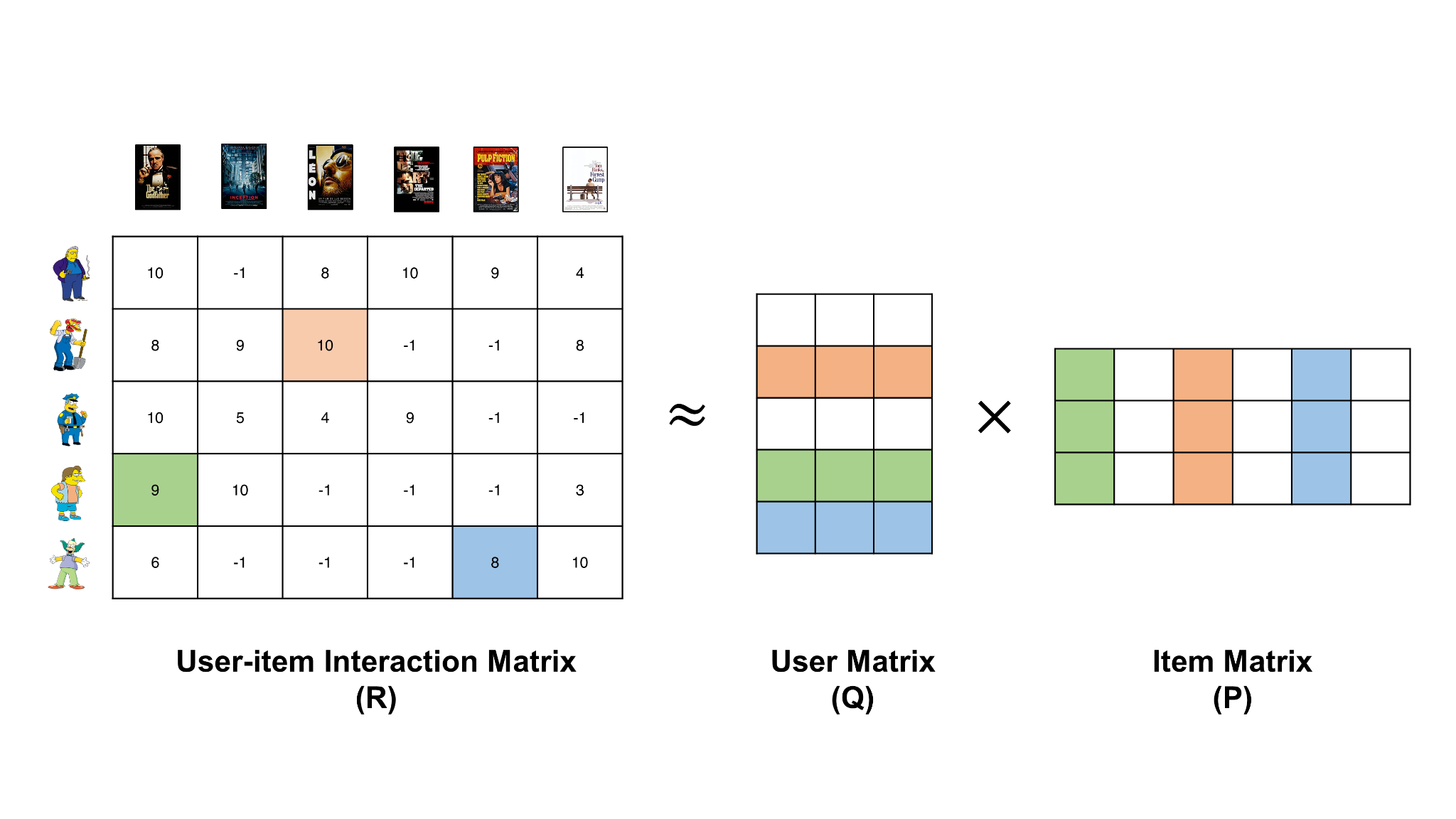
We will try two different kinds of algorithms here: Nearest Neighbour and Matrix Factorization.

k-Nearest Neighbors (kNN) is used to form clusters of similar users based on common movie ratings. Finally, predictions are made using the average rating of top-k nearest neighbors. We will try three kNN based algorithms:

* KNNBasic
* KNNWithMeans
* KNNWithZScore

Matrix Factorization techniques find latent features that are then used to compute similarity between users and items. We will try two matrix factorization based algorithms:

* SVD
* SVDpp



To make our life easier, we will make use of the really awesome [surprise library](https://surpriselib.com/). It allows us to build complex recommendation engine pipelines effortlessly.

We will evaluate top models from each list and then do grid search on them to search for the best hyperparameters.

We will run a 5-fold cross validation to test accuracy for each of the five algorithms.

We chose two standard errors as our evaluation metrics:

* **Mean Absolute Error (MAE)** computes the average of all the the absolute value differences between the true and the predicted rating.
* **Root Mean Square Error (RMSE)** computes the mean value of all the differences squared between the true and the predicted ratings and then proceeds to calculate the square root out of the result.

```

# Running 5-fold cross validation to test accuracy of the algorithm

knn\_basic = cross\_validate(KNNBasic(), data, cv=5, n\_jobs=5, verbose=True)

knn\_means = cross\_validate(KNNWithMeans(), data, cv=5, n\_jobs=5, verbose=True)

knn\_z = cross\_validate(KNNWithZScore(), data, cv=5, n\_jobs=5, verbose=True)

svd = cross\_validate(SVD(), data, cv=5, n\_jobs=5, verbose=True)

svd\_pp = cross\_validate(SVDpp(), data, cv=5, n\_jobs=5, verbose=True)

```

An avergae MAE of 0.6861 for SVD indicates an average absolute error of 0.6861 between the true and predicted ratings. We will try to reduce this error further by tuning hyperparameters

Both the Matrix Factorization algorithms seem to do much better for both the metrics. We will focus our attention on them for hyperparameter optimization.

While SVDpp had the better performance in terms of error rate, it is very time consuming to train. A grid search on SVDpp lasts many days. So, I chose to optimize the SVD model for number of epochs, learning rate and regularization using grid search.

The best accuracy was received for the following hyperparameter configuration:

SVD with number of epochs = 50, learning rate = 0.02, and regularization = 0.1

**RMSE:** 0.8507

**MAE:** 0.6517

While our SVD based collaborative filtering model works well, it suffers from one issue: the **cold-start** problem.

The **cold-start** problem occurs when you don't have enough past information about a particular user. Imagine signing up to Amazon as a new customer. When you just sign up, amazon does not have any information about you and thus, cannot make intelligent recommendations to you based on your purchases.

This is why Amazon would often use other information like geography to recommend products to you.

We will use the **popularity based recommendation engine to resolve the cold-start** problem. This will be used for new users with no ratings in the dataset.

For old users, the SVD model will be used to calculate expected ratings for all the movies in the dataset and then return the top 5 movies with the highest expected ratings.

To add more customization, we are adding the possibility to mention preferred genre and also give the minimum number of ratings.

We will also filter out movies that the user has already watched.

Here is the dashboard:

## **Conclusion**

Recommendation engines are a data filtering tool that help recommend the most relevant items to a particular user. Nowadays, all types of companies utilize very complex machine learning based recommendation systems with great success. Today, we tried building one such recommendation engine for movies.

For this purpose, we made use of the famous MovieLens database. We tried different recommendation engines like popularity based, content based and collaborative filtering. After testing all the models for multiple accuracy metrics, we ended up with a hybrid system.

This hybrid engine used a popularity based model to solve the cold-start problem and then the predictions were made using an SVD based collaborative filtering model.

# **Future Work**

A lot more can be done with this project.

We analysed the tag information to a certain extent in this report. Even though the results weren't very encouraging, I think we can make some use of that information. We will need to incorporate NLP in our system though.

Unfortunately, we couldn't make use of the full MovieLens dataset as we had limited computational power. We can think of a distributed system or different algorithms using which we can utilize all the data that we have.

We also need to scrape more information about the movies like cast, director, etc from the internet.