# Capstone Project - Netflix Recommendation Engine

By Noahchristy

Table of Contents

1. Problem Statement
2. Project Objective
3. Data Description
4. Data Pre-processing Steps and Inspiration
5. Choosing the Algorithm for the Project
6. Motivation and Reasons For Choosing the Algorithm
7. Assumptions
8. Model Evaluation and Techniques
9. Inferences from the Same
10. Future Possibilities of the Project
11. Conclusion
12. References

# Problem Statement

Customer Behaviour and it’s prediction lies at the core of every Business Model. From Stock Exchange, e-Commerce and Automobile to even Presidential Elections, predictions serve a great purpose. Most of these predictions are based on the data available about a person’s activity either online or in-person.

Netflix is all about connecting people to the movies they love. To help customers find those movies, they developed world-class movie recommendation system: CinematchSM. Its job is to predict whether someone will enjoy a movie based on how much they liked or disliked other movies. Netflix use those predictions to make personal movie recommendations based on each customer’s unique tastes.

Recommendation Engines are the much needed manifestations of the desired Predictability of User Activity. Recommendation Engines move one step further and not only give information but put forth strategies to further increase users interaction with the platform.

# Project Objective

1. Find out the list of most popular and liked genre
2. Create Model that finds the best suited Movie for one user in every genre.

3. Find what Genre Movies have received the best and worst ratings based on User Rating.

# Data Description

The data set which was provided contains the following information in columns and rows.

1. ID – Contains the separate keys for customer and movies.

2. Rating – A section contains the user ratings for all the movies.

3. Genre – Highlights the category of the movie.

1. Movie Name – Name of the movie with respect to the movie id

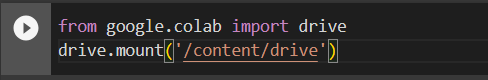
The data set is divided into two parts - the first set contains ID and ratings and the second set contains movie name and genre.

# Data Preprocessing Steps And Inspiration

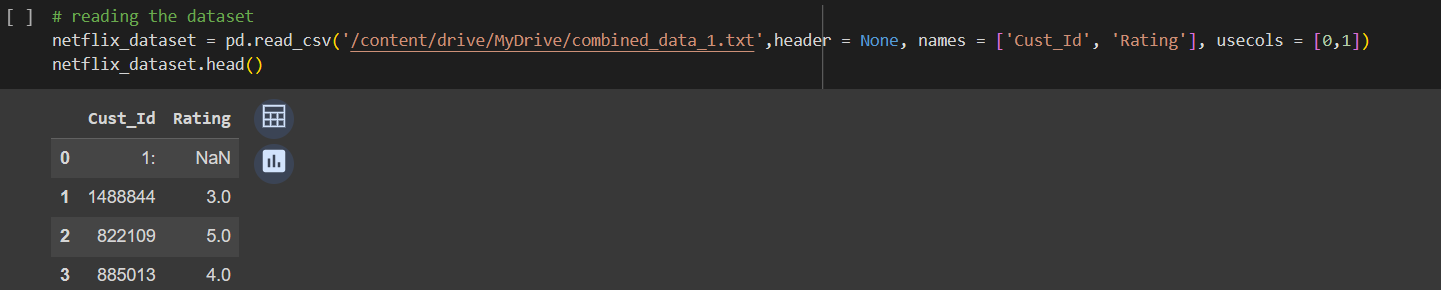
The preprocessing of the data included the following steps:

1. **First step:**

Importing the data from google drive to google colab.



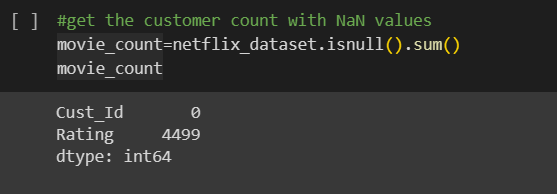
The code snippet utilizes the drive module from the google.colab library to mount Google Drive within a Google Colab notebook environment



The code snippet reads a dataset file named 'combined\_data\_1.txt' from the user's Google Drive using the pandas library in a Google Colab notebook

'/content/drive/MyDrive/combined\_data\_1.txt.': This is the file path to the dataset file stored in the user's Google Drive.

1. **Second step:**



# Counting NaN Values in Customer IDs:

# The code snippet calculates the count of NaN (Not a Number) values in the "Cust\_Id" column of the netflix\_dataset DataFrame.

# 

# Calculating the Number of Unique Customers:

# The code snippet calculates the number of unique customers in the Netflix dataset by using the nunique() method on the "Cust\_Id" column. Here's a breakdown of the code:

# 

# Subtracting the movies count from both customer count and rating count variables.

# 

# Counting Ratings for Each Star Level:

# The code snippet groups the Netflix dataset by the "Rating" column and calculates the count of each distinct rating (1, 2, 3, 4, and 5 stars).

# 

# 

# The above mentioned code and graph will visually show us on the all the details movies count ratings and customers.

# **Third Step:**

# In this step we are in the need to separate customer ids according to the movies which they have rated.

# 

# Adding Movie ID Column and Calculating Null Values in Ratings:

# The code snippet adds a new column "Movie\_ID" to the netflix\_dataset DataFrame and creates a new DataFrame df\_nan to check for null values in the "Rating" column.

# 

# **Fourth Step:**

# Creating Numpy Array for Movie IDs:

# 

# The code snippet creates a Numpy array (movie\_np) that assigns movie IDs based on the position of null values in the "Rating" column of the Netflix dataset.

# a loop to iterate through the DataFrame df\_nan and assigns movie IDs based on the position of null values in the "Rating" column. The np.full() function is used to create arrays filled with the current movie ID.

# The resulting arrays are appended to the movie\_np array.

# The last record is handled separately, accounting for the length of the remaining dataset.

# After the execution of this code, the movie\_np array will contain movie IDs corresponding to the positions of null values in the "Rating" column

# **Fifth Step:**

# Filtering Users and Movies Based on Ratings Count:

# 

# 

# The code filters out users who have rated fewer movies and movies that have been rated less frequently by creating a summary DataFrame named dataset\_movie\_summary

# netflix\_dataset.groupby('Movie\_Id'): This groups the dataset by the "Movie\_Id" column.

# agg(f): The agg() function is used to apply the specified aggregation functions (f) to each group. In this case, it calculates the count and mean for each movie.

# 

# The code snippet calculates the 70th percentile (quantile) of the "count" column.This value can be used as a threshold for filtering out less popular movies based on the number of ratings they have received.

# Creating a Movie Rating Count Benchmark:

# The code snippet creates a benchmark for the minimum number of ratings a movie should have to be considered in the analysis.

# 

# The round() function is used to round the calculated value to the nearest whole number, ensuring that the benchmark is a whole number.

# Creating a List of Movies to Drop Based on Benchmark:

# 

# The code snippet creates a list of movie IDs (drop\_movie\_list) that do not meet the specified benchmark for the minimum number of ratings.

# Creating Customer Summary for Activity:

# The code snippet creates a summary DataFrame (dataset\_cust\_summary) based on customer activity, aggregating the "Rating" column.

# 

# This groups the dataset by the "Cust\_Id" column and selects the 'Rating' column for aggregation.

# **Sixth Step:**

# The code snippet calculates a benchmark for the minimum number of ratings a customer should have given to be considered active in the analysis.

# 

# This calculates the 70th percentile of the "count" column in th dataset\_cust\_summary DataFrame, representing the minimum number of ratings a customer should have.

# The resulting drop\_cust\_list is a list of customer IDs that you may consider dropping from your analysis due to their lower rating activity.

# Filtering Customers and Movies Based on Benchmarks:

# The code snippet filters out customers and movies that fall below the specified benchmarks for activity.

# 

# Trimming the Dataset Based on Filtered Lists:

# 

# by removing rows corresponding to customers and movies that fall below the specified benchmarks for activity.

# After running this code, netflix\_dataset will contain only the rows corresponding to customers and movies that meet the specified benchmarks for activity

# **Seventh Step:**

# Preparing Dataset for Singular Value Decomposition (SVD):

# The code snippet prepares the netflix\_dataset DataFrame for Singular Value Decomposition (SVD) by converting it into a sparse matrix using a pivot table.

# 

# This creates a pivot table (df\_p) from the netflix\_dataset DataFrame, where the rows represent customers, columns represent movies, and the values are the ratings.

# Now here we import another dataset which contains movie\_id, year and name for further model building and evaluation.

# 

# This reads the movie titles dataset from the specified CSV file, considering only columns with indices 0, 1, and 2, and assigning column names 'Movie\_Id', 'Year', and 'Name' respectively

# Choosing the Algorithm For the Project

Description for the Singular value Decomposition(SVD) algorithm for the project.

I have chosen the SVD algorithm for this project for the following reasons:

1. Dimensionality Reduction:

SVD inherently performs dimensionality reduction by retaining only the top-k singular values and associated vectors. This helps in reducing the sparsity of the user-item matrix and focusing on the most relevant information, making computations more efficient.

1. Handling Missing Values (Imputation):

SVD can handle missing values in the user-item matrix, which is common in recommendation systems due to the sparse nature of user ratings. The factorization process effectively imputes missing values, providing predictions for items that users have not rated.

1. Improved Prediction Accuracy:

SVD often leads to improved prediction accuracy compared to simpler methods. The latent factors discovered by SVD capture underlying patterns in user preferences and item characteristics, allowing for more accurate predictions of user ratings.

1. Collaborative Filtering:

SVD is a collaborative filtering technique that leverages the preferences and behaviors of users to make recommendations. This approach is well-suited for platforms like Netflix, where user interactions with content play a crucial role in generating accurate recommendations.

1. Scalability:

Once the factorization is performed, predicting ratings for new items or users becomes computationally efficient. This scalability is essential for large-scale recommendation systems like Netflix, where the user and item databases are extensive

# Assumptions

The following assumptions were made in order to create the SVD model for Netflix recommendation engine project.

1. Linear Relationships:

SVD assumes linear relationships between users and items. It assumes that users can be represented as linear combinations of latent factors and that items can similarly be expressed in terms of these factors. This assumption may oversimplify the underlying user preferences and item characteristics.

1. Collaborative Filtering Assumption:

SVD relies on the collaborative filtering assumption that users who agreed in the past will agree in the future. This may not always hold true, especially when user preferences evolve.

1. Independence of Ratings:

SVD assumes that user ratings are independent of each other. This means that the rating a user gives to one item is not influenced by the ratings they give to other items. In reality, user preferences may be influenced by previous ratings and interactions.

1. Similar users like similar items:

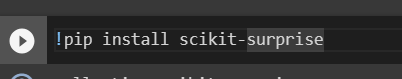
The model assumes that users who have similar ratings for some items will also have similar ratings for other items. This assumption is the basis of collaborative filtering, which is a type of recommendation system that works by finding similar users and recommending items that they have rated highly.

It's essential to be aware of these assumptions and consider their implications when applying SVD to a recommendation system.

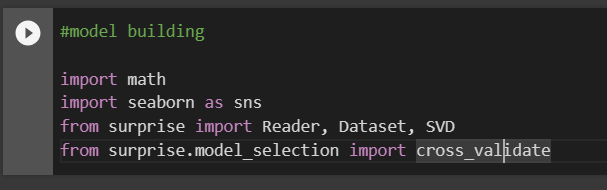
# Model Evaluation and Technique

The following techniques and steps were involved in the evaluation of the model

1. Importing all the required packages:



This command will install the scikit-surprise library, which is a Python library specifically designed for building and analyzing recommendation systems.



This imports the required modules from scikit-surprise:

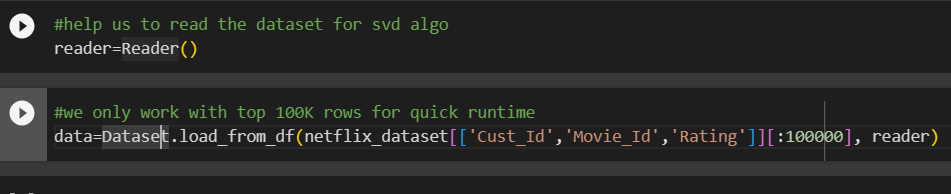
Reader: Helps in parsing the file or dataframe containing ratings.

Dataset: Represents the dataset, and can be used to load data from various sources.

SVD: Stands for Singular Value Decomposition, which is a collaborative filtering algorithm for recommendation.

cross\_validate: Provides functionality for cross-validation.

1. Read the dataset into SVD algo:



# svd = SVD(): This initializes an instance of the SVD model.

# 

# measures=['RMSE', 'MAE'] indicates the evaluation metrics to be used, which are Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE). cv=3 specifies 3-fold cross-validation.

# During cross-validation, the dataset is split into k folds (in this case, 3 folds), and the model is trained and evaluated k times, each time using a different fold as the test set and the remaining folds as the training set. This helps to assess the model's performance on different subsets of the data.

# Building the recommendation algorithm.

# First we take user 712664 and we try to recommend some movies based on the past data.

# 

# 

# first we will make a shallow copy of the movie\_titles.csv file so that we can change

# the values in the copied dataset, not in the actual dataset.

# 

# Now, you have a copy named user\_712664 that you can modify without affecting the original df\_title DataFrame. If you plan to customize or update information related to user 712664, you can make changes to user\_712664 while keeping the original data intact in df\_title.

# Resetting tha index:

# This method resets the index of the DataFrame. The reset\_index() function is used to move the current index (which might be a multi-level index) to columns and generate a default integer index.

# DataFrame will have a default integer index, and the previous index (potentially Movie\_Id) will become a regular column in the DataFrame.

# 

# filtering the user\_712664 DataFrame to exclude movies that were identified earlier and placed in the drop\_movie\_list.

# 

# This condition checks whether each movie ID in the user\_712664 DataFrame is in the drop\_movie\_list. After running this code, the user\_712664 DataFrame will contain only the rows corresponding to movies that were not in the drop\_movie\_list

# Prediction:

# the SVD model (svd) to estimate scores for movies in the user\_712664 DataFrame for user 712664.

# 

# user\_712664 should have a new column ('Estimate\_Score') containing the estimated scores for user 712664 for each movie in the DataFrame.

# sorting the user\_712664 DataFrame based on the 'Estimate\_Score' column in ascending order and then printing the sorted DataFrame.

# 

# It sorts the DataFrame based on the values in the 'Estimate\_Score' column in ascending order. It prints the sorted DataFrame, showing the movies in the order of increasing estimated scores.

# 

# re-sorting the user\_712664 DataFrame based on the 'Estimate\_Score' column, this time in descending order (highest scores first), and then printing the top 10 rows.

# 

# It sorts the DataFrame based on the values in the 'Estimate\_Score' column in descending order, so the movies with the highest estimated scores come first.

# you'll have the user\_712664 DataFrame sorted in descending order by estimated scores, and you'll see the top 10 recommended movies

# Inferences from the Project

based on a typical recommendation system project using collaborative filtering and Singular Value Decomposition (SVD), here are some potential inferences you might draw:

* User Preferences:

The model has learned latent features that capture user preferences. This understanding is derived from the estimated scores for movies, indicating the likelihood of a user enjoying a particular movie.

* Movie Popularity:

The recommendations are influenced by the popularity of movies within the dataset. Movies with higher estimated scores might be popular or have broader appeal.

* Personalization:

The model provides personalized recommendations for each user based on their historical ratings. This personalization is a key strength of collaborative filtering.

* Model Performance:

Assess the model's performance metrics (e.g., RMSE, MAE) from the cross-validation. Lower RMSE and MAE values generally indicate better predictive accuracy

* Movie Filtering:

The process of filtering out less active or less popular movies (using drop\_movie\_list) affects the recommendations. Evaluate whether this filtering improves the quality of recommendations and reduces noise.

# Future Possibilities

The future possibilities are:

* Hybrid Models:

Combine collaborative filtering (SVD) with other recommendation techniques, such as content-based filtering or hybrid models. This can potentially improve recommendation accuracy and address limitations inherent in collaborative filtering.

* Advanced Collaborative Filtering Models:

Explore more advanced collaborative filtering models, such as matrix factorization variants (e.g., alternating least squares), deep learning-based collaborative filtering, or neural collaborative filtering.

* Dynamic Recommendations:

Implement a system for real-time or dynamic recommendations that adapt to users' immediate preferences, taking into account recent interactions.

* Explanability:

Enhance model interpretability to provide users with explanations for why certain recommendations are made. This can improve user trust and understanding of the system.

# Conclusion

This Netflix recommendation engine project leveraged collaborative filtering, specifically Singular Value Decomposition (SVD), to generate personalized movie recommendations for users based on historical ratings.

In conclusion, this Netflix recommendation engine project provided valuable insights into the application of collaborative filtering, specifically SVD, for generating personalized movie recommendations. The project laid the foundation for future enhancements and explored avenues for overcoming limitations, emphasizing the importance of continuous evaluation, experimentation, and consideration of ethical implications in recommendation system development.

# References

# Intellipaat Lectures: Revised the lectures related to this project several time.

# Kaggle website: Verified sample code snippets related to this project.

# Analytics vidhya website: Revised the SVD algorithm.