# Movie Recommendations

## Objective

The objective of this project is to build a recommendation system for the Movielens data set. Given the data set containing users to movie ratings mappings, recommendations for new movies to watch are made for users. This is done by predicting the ratings a user would give to movies that they haven't watched and picking the top candidates with the highest ratings.

## Uses

Recommender systems are used in a wide variety of areas including movies, books. music, apparel, consumer products, mobile apps, games etc. They help the users discover items matching their taste that they may not have found otherwise.

## Benefits

Following are some of the benefits of recommender systems to businesses:

- Revenue:

Making personalized recommendations based on the user's tastes results in a higher conversion rate and an increased revenue due to higher average spend per order and higher number of items per order.

- Improved customer experience:

The customer's shopping experience becomes more efficient and satisfying when relevant products are recommended to them. They tend to return at a later date to buy more of the recommended items. This also results in customer loyalty towards the business.

- Personalization:

For an online business, knowing customers' preferences and making personalized promotional offers and email campaigns aid in improving brand affinity and customer satisfaction.

## Dataset

The dataset used for this project will be the Movielens 100K dataset. This is a stable benchmark dataset that is publicly available. The ratings are made on a 5 star scale with half star increments.

# Proposed approaches

## Collaborative filtering

Collaborative filtering works by using similarities between users or items to make recommendations.

### User based collaborative filtering:

With this technique, recommendations are made by forming user neighborhoods. Users of the same neighborhood share common preferences.

From the user to movie matrix, this method aims at finding users most similar to the active user. The distance or similarity is determined using different measures - Euclidean distance, Pearson correlation, Cosine similarity etc. Based on this, a user to user weighted ratings matrix can be formed.

### Item based collaborative filtering:

This technique makes recommendations based on relationships between items that are rated similarly by users. Based on the items the user has liked, other items in the neighborhood of already rated items are recommended.

### Challenges with collaborative filtering:

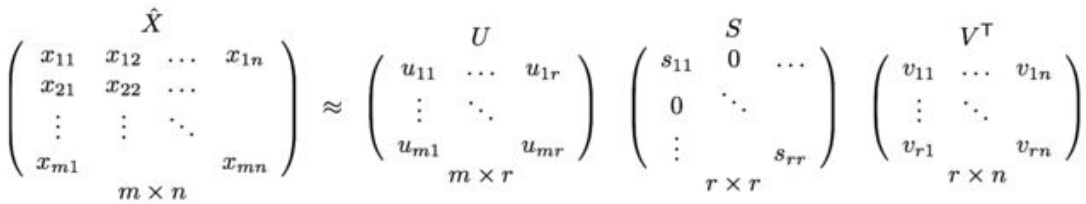
Data sparsity
Not enough ratings in the system

Cold start
New user or new item

Scalability
Very large number of users and movies

## Singular Value Decomposition (SVD)

The SVD method uses Matrix Factorization to extract latent features. This results in dimensionality reduction. The matrix factorization is done on the user-items ratings matrix.



Singular Matrix Decomposition(http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/images/svd2.png)

X denotes the utility matrix,
U is a left singular matrix, representing the relationship between users and **latent factors.**
S is a diagonal matrix describing the strength of each latent factor
V transpose is a right singular matrix, indicating the similarity between items and latent factors.

For the capstone project, we'll be using the SVD implementation from scikit surprise. **Surprise** is a Python **scikit** for building and analyzing recommender systems.

To estimate the unknown ratings, the algorithm internally minimizes the regularized squared error using Stochastic Gradient Descent.

Following is the documentation page for the implementation we use
https://surprise.readthedocs.io/en/stable/matrix_factorization.html#surprise.prediction_algorithms.matrix_factorization.SVD

## Data wrangling:

The dataset used for the capstone project is the MovieLens 100K dataset.

After loading the csv data into a pandas dataframe, I checked the dimensions using shape attribute of the dataframe. I checked for any null values using the info method and found out that all columns had non null values for the ratings and movies data sets. Since this is a reference dataset for movie recommendations, the source data seems pretty clean.

```
ratings.shape
```

```
(100004, 4)
```

```
ratings.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100004 entries, 0 to 100003
Data columns (total 4 columns):
userId       100004 non-null int64
movieId      100004 non-null int64
rating       100004 non-null float64
timestamp    100004 non-null int64
dtypes: float64(1), int64(3)
memory usage: 3.1 MB
```

```
movies.shape
```

```
(9125, 3)
```

```
movies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9125 entries, 0 to 9124
Data columns (total 3 columns):
movieId    9125 non-null int64
title      9125 non-null object
genres     9125 non-null object
dtypes: int64(1), object(2)
memory usage: 213.9+ KB
```

Next we'll check whether we see what we expect from a data analysis (of user ratings) standpoint. The ratings data is the most important for our analysis. It contains only the movies users actually rated. Therefore it makes sense that it doesn't contain any null values. When we create a pivot table out of the ratings data as the next step, we see that all movies a given user hasn't yet rated show up as NaN values. A lot of such values are expected as this is supposed to be a sparse matrix.

```
user_ratings = pd.pivot_table(ratings, index='movieId', columns='userId', values='rating')
```

```
user_ratings.head()
```

| userId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 670 | 671 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | | | | | | | | |
| 1 | NaN | NaN | NaN | NaN | NaN | NaN | 3.0 | NaN | 4.0 | NaN | ... | NaN | 4.0 | 3.5 | NaN | NaN | NaN | NaN | NaN | 4.0 | 5.0 |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 5.0 | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | NaN | NaN | NaN | NaN | 4.0 | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |

5 rows × 671 columns

In a later step, in the process of getting recommendations for a specific user from the movies that user has not rated, we get the unrated movies. At this step, the timestamp column in the resulting dataframe doesn't provide any useful information and is considered as noise. In this case, we drop the irrelevant column from the dataframe.

```
# User ratings for movies not rated by user 665
movies_unrated = ratings[rating_665[ratings.movieId].isnull().values]
movies_unrated = movies_unrated.drop(['timestamp'],axis=1)
movies_unrated.head()
```

| | userId | movieId | rating |
|---|---|---|---|
| 0 | 1 | 31 | 2.5 |
| 3 | 1 | 1129 | 2.0 |
| 4 | 1 | 1172 | 4.0 |
| 5 | 1 | 1263 | 2.0 |
| 6 | 1 | 1287 | 2.0 |

## Inferential Statistics:

For making movie recommendations for a specific user as part of the capstone project, one of the approaches involves performing User-based Collaborative Filtering.

This involves using a similarity measure to determine the correlation among users. Cosine similarity and Pearson correlation are two popular measures used by recommendation systems for this purpose. Recent implementations tend to use pearson correlation as it has been found to provide the best results.

The following formula is used to determine correlation between user i and user k.

$$u_{ik} = \frac{\sum_j (v_{ij} - v_i)(v_{kj} - v_k)}{\sqrt{\sum_j (v_{ij} - v_i)^2 \sum_j (v_{kj} - v_k)^2}}$$

Pearson Correlation (https://goo.gl/y93CsC)

After reading the user ratings data, we create a pivot table to create a 2D matrix between users and the movies with the known ratings as values. This is a sparse matrix as the number of movies a given user has rated tends to be a very low proportion of the total number of movies.

```
user_ratings = pd.pivot_table(ratings, index='movieId', columns='userId', values='rating')
```

```
user_ratings.head()
```

| userId | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 662 | 663 | 664 | 665 | 666 | 667 | 668 | 669 | 670 | 671 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| movieId | | | | | | | | | | | | | | | | | | | | | |
| 1 | NaN | NaN | NaN | NaN | NaN | NaN | 3.0 | NaN | 4.0 | NaN | ... | NaN | 4.0 | 3.5 | NaN | NaN | NaN | NaN | NaN | 4.0 | 5.0 |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | 5.0 | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | NaN | NaN | NaN | NaN | 4.0 | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN |

5 rows × 671 columns

A single series from this dataframe is extracted for a specific user (the active user). The pearson correlation is then calculated between that user's rating data and the entire user ratings dataframe.

```
rating_665 = user_ratings[665]
```

```
rating_665.head()
```

```
movieId
1    NaN
2    3.0
3    3.0
4    NaN
5    3.0
Name: 665, dtype: float64
```

```
corr_665 = user_ratings.corrwith(rating_665)
```

The pandas DataFrame object has a corrwith method that performs this operation. Inspecting the results of this operation gives correlation between the active user and every other user. The correlation coefficient values range from -1 to +1 with +1 being given for users with most similar ratings and movie preferences as the active user.

```
corr_665 = corr_665.sort_values(ascending=False)
```

```
corr_665.head()
```

```
userId
272    1.0
665    1.0
158    1.0
540    1.0
135    1.0
dtype: float64
```

```
corr_665.sort_values(ascending=True).head()
```

```
userId
482   -1.000000
403   -1.000000
331   -1.000000
301   -1.000000
490   -0.891902
dtype: float64
```

We then filter out the movies the active user has already rated and come up with rating predictions for the active user by using the pearson correlation coefficient values. This involves giving higher weightage for similar users' ratings than for users' with opposing movie preferences.

For the capstone project, we also come up with recommendations based on Singular Value Decomposition (SVD) method. This method reduces the dimensionality by doing matrix factorization to extract latent features. This method typically performs very well for recommender systems.

We come up with evaluation metrics for our recommender. For this purpose, we compute accuracy metrics - RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error).

We also calculate some Top N metrics to evaluate the relevance of top N recommendations provided to the user. In the real world, these metrics are more fitting for recommender systems.

Following are the metrics we compute for our system.

- Hit Rate - Number of successful hits from Top N / Number of users
- Average Reciprocal Hit Rate - Cumulative value of 1/rank / Number of users
- Cumulative Hit Rate - How often we recommended a movie the user actually liked (ignore low ratings)
- Hit Rate by Rating value - Break down of hit rate by rating value

## Evaluating the Recommendation system using metrics:

The metrics relevant for a given system depend on the requirements and the UI design of the recommender.

While considering the above top N metrics, if the layout allows for simultaneous display of all top N recommendations, then hit rate might be the best.

If on the other hand, the UI displays the recommendations one by one and the user has to scroll through to get to the lower ranked ones within the top N, then it might help to reward recommendations the users end up watching with that are ranked towards the top and penalize for any item watched that are ranked lower among the N movies displayed. For this scenario, the ARHR (Average Reciprocal Hit Rank) might be the best to use.

Cumulative Hit Rate is a special case of hit rate with a threshold using which the lower ranked ratings among the hits are discarded. This is when we want to make sure that the recommender system comes up with movies that the user actually likes.

Hit Rate by rating value give a break down of ratings for the hits.

One key point to note is that for top N metrics, their values by themselves may not help us come to conclusion about the effectiveness of the system. Due to high sparsity that is very common for user ratings, the observed values can be low. Practically, these values are most useful when comparing multiple recommender implementations.

# Observed metrics for the capstone project:

## Accuracy metrics

The metrics were calculated using scikit surprise library methods.

Direct accuracy metrics for RMSE and MAE
**RMSE: 0.9033701087151802**
**MAE: 0.6977882196132263**

5 fold cross validation results for RMSE and MAE

| | Fold 1 | Fold 2 | Fold 3 | Fold 4 | Fold 5 | Mean | Std |
|---|---|---|---|---|---|---|---|
| **RMSE (testset)** | **0.8953** | **0.8950** | **0.8974** | **0.8953** | **0.9027** | **0.8972** | **0.0029** |
| **MAE (testset)** | **0.6899** | **0.6892** | **0.6919** | **0.6900** | **0.6927** | **0.6907** | **0.0013** |

## Top N metrics

Following were the observed Top N metrics

**Hit Rate: 0.029806259314456036**

**ARHR (Average Reciprocal Hit Rank): 0.0111560570576964**

**cHR (Cumulative Hit Rate, rating >= 4): 0.04960835509138381**

**rHR (Hit Rate by Rating value):**
**3.5 0.017241379310344827**
**4.0 0.0425531914893617**
**4.5 0.020833333333333332**
**5.0 0.06802721088435375**

The top N metrics above appear to be on the lower side due to the sparsity of user ratings data.

It is best to combine metrics for more insight. In our case, the hit rate is 3% but the cumulative hit rate (cHR) is 5%. This is good as the cHR discards low ranked left out ratings

## A/B Testing:

A recommender system with great accuracy and top N metrics isn't still guaranteed to be successful if rolled out. To a real world recommender, these metrics by themselves are not enough.

Online A/B tests and multivariate tests are extremely important to evaluate real world impact and tune the recommender system. Good A/B tests help companies to invest time and resources only on systems that actually work in the real world.

A/B tests typically come up with estimates for CTR (Click Through Rate), CR (Conversion rate) and ROI (Return on Investment). These estimates can help the stakeholders make the final decision on whether to switch the implementation and which new implementation to choose as the case may be.

## Summary:

Explored / learned the following during the course of my Capstone project

- Multiple approaches to coming up with recommendations - User based CF and SVD.
- Compute Pearson correlation among users to determine similarity for User based Collaborative Filtering.
- The importance of ranking based metrics for recommender systems in the real world in addition to accuracy metrics.
- Interpret the metrics for recommender systems and pick the relevant metrics for a given use case.
- Accuracy and top N metrics can be used to narrow down the selections to a single implementation or a few. Online A/B tests are the actual indicators of the success of a recommender system.

## Next Steps:

I would like to explore the following further

- Ways to transform the user ratings matrix to reduce the effects of sparsity.
- Deep Learning based approaches to building recommender systems.

## Links & References:

- My github repo with capstone project resources and other springboard submissions. https://github.com/lakshv77/springboard/
- Helpful post on overview and comparison of different approaches to implement recommendation systems. https://towardsdatascience.com/various-implementations-of-collaborative-filtering-100385c6dfe0
- Top N metrics evaluation was based on the material from this online course. https://www.udemy.com/building-recommender-systems-with-machine-learning-and-ai/