UCONN
SCHOOL OF BUSINESS

# Movie Recommender Systems

**Final Project Report** - **OPIM5894 – Data Science with Python**

May 10th 2017

---

**Team PYBOTS**: Swati Arora, Karpagam Vinayagam, Ravi Srivastava, Prachi Gupta, Gaurav Vijay

# Contents

# Introduction

Ever wondered, “What algorithm Google uses to maximize its target advertisements revenue?”. What about the e-commerce websites which advocates you through options such as ‘people who bought this also bought this’. Or “How does Facebook automatically suggest us to tag friends in pictures”?

The answer is Recommendation Engines. With the growing amount of information on world wide web and with significant rise number of users, it becomes increasingly important for companies to search, map and provide their users with the relevant chunk of information according to their preferences and tastes.

Companies nowadays are building smart and intelligent recommendation engines by studying the past behavior of their users. Hence providing them recommendations and choices of their interest in terms of “Relevant Job postings”, “Movies of Interest”, “Suggested Videos”, “Facebook friends that you may know” and “People who bought this also bought this” etc.

## Why Recommender System?

Recommender systems are an important part of the information and e-commerce ecosystem. They represent a powerful method for enabling users to filter through large information and product spaces.

Historically, people have relied on recommendations and mentions from their peers or the advice of experts to support decisions and discover new material. They discuss the week’s blockbuster over the water cooler, they read reviews in the newspaper’s entertainment section, or they ask a librarian to suggest a book. They may trust their local theater manager or news stand to narrow down their choices, or turn on the TV and watch whatever happens to be playing.

These methods of recommending new things have their limits, particularly for information discovery. There may be an independent film or book that a person would enjoy, but no one in their circle of acquaintances has heard of it yet. There may be a new indie band in another city whose music will likely never cross the local critic’s radar. Computer-based systems provide the opportunity to expand the set of people from whom users can obtain recommendations. They also enable us to mine users’ history and stated preferences for patterns that neither they nor their acquaintances identify, potentially providing a more finely-tuned selection experience.

## What is Recommender System?

According to Wikipedia: *"Recommender systems or* **recommendation systems** *(sometimes replacing "system" with a synonym such as platform or engine) are a* **subclass of information filtering system** *that seek to predict the 'rating' or 'preference' that user would give to an item."*

Recommender systems are active information filtering systems which personalize the information coming to a user based on his interests, relevance of the information etc. They are simple algorithms which aim to provide the most relevant and accurate items to the user by filtering useful stuff from of a huge pool of information base. In other words, recommendation engines discover data patterns in the data set by learning consumers choices and produces the outcomes that co-relates to their needs and interests. Recommender systems are used widely for recommending movies, articles, restaurants, places to visit, items to buy etc. These systems have evolved to fulfill the natural dual need of buyers and sellers by automating the generation of recommendations based on data analysis.

Recommender systems differ in the way they analyze these data sources to develop notions of affinity between users and items which can be used to identify well-matched pairs. Collaborative Filtering systems analyze historical interactions alone, while Content-based Filtering systems are based on profile attributes; and Hybrid techniques attempt to combine both of these designs. The architecture of recommender systems and their evaluation on real-world problems is an active area of research.

The myriad approaches to Recommender Systems can be broadly categorized as:

• Collaborative Filtering (CF): In collaborative filtering system a user is recommended items based on the past ratings of all users collectively. It recommends items based on similarity measures between users and/or items. The items recommended to a user are those preferred by similar users.

• Content-based recommending: The content based system recommends items that are similar in content to items the user has liked in the past, or matched to attributes of the user. It examines properties of the items recommended. For instance, if a Netflix user has watched many cowboy movies, then recommend a movie classified in the database as having the "cowboy" genre.

• Hybrid approaches: These methods combine both collaborative and content based approaches.

### Applications of Recommendation Systems

The major areas of application of the recommender system are as follows:

1. Product Recommendations: Perhaps the most important use of recommendation systems is at on-line retailers. We have noted how Amazon or similar on-line vendors strive to present each returning user with some suggestions of products that they might like to buy. These suggestions are not random, but are based on the purchasing decisions made by similar customers or on other techniques we shall discuss in this chapter.

2. Movie Recommendations: Netflix offers its customers recommendations of movies they might like. These recommendations are based on ratings provided by users. The importance of predicting ratings accurately is so high, that Netflix offered a prize of one million dollars for the first algorithm that could beat its own recommendation system by 10%.

3. News Articles: News services have attempted to identify articles of interest to readers, based on the articles that they have read in the past. The similarity might be based on the similarity of important words in the documents, or on the articles that are read by people with similar reading tastes. The same principles apply to recommending blogs from among the millions of blogs available, videos on YouTube, or other sites where content is provided regularly.

## Project Proposal

The objective of the project is to create a robust recommender system which would recommend movies to users.

### Data Description

For building the movie recommendation system, we selected the dataset from Movie Lens website which has 1000209 ratings given by 6040 users on 3883 movies. Movie Lens dataset has three data files: Movies, Ratings, and Users. Movies data has information about the title of the Movie, Movie ID, and genres of the movie. Users data has user information such as User ID, Gender, Age, Occupation, and Zip code. Ratings data contains data containing all the ratings given by users for movies and the columns are User ID, Movie ID, Rating, and Timestamp.

Users data

| UserID | Age | Gender | Occupation | Zip code |
|---|---|---|---|---|

Ratings data

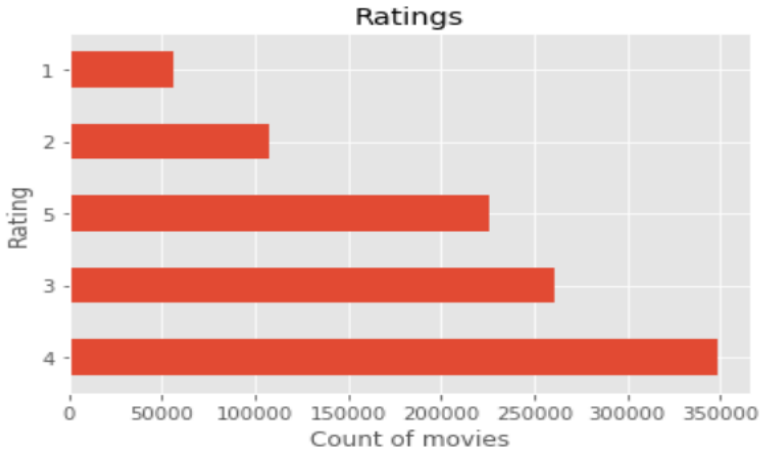
| UserID | MovieID | Rating | Timestamp |
|---|---|---|---|

Movie data

| MovieID | Title (Year) | Genres |
|---|---|---|

## Data Exploration

Movies have ratings from 1 to 5 and the mean rating is 3.58 for over 1 million observations. We observed the highest number of observations with rating 4 followed by 3 and 5. The average age of a user who rated a movie is 30.




From Movie data, we identified a total of 18 genres in the data and highest number of movies are from “Drama” and “Comedy” genres. Some other significant genres include Action, Romance, and Thriller. While the data had a total of 3883 movies, ratings were given only for 3706 movies. The data had no missing values. We used count() and isnull().sum() functions to check for missing values and used describe() function to get the summary statistics of the columns such as Rating and age.

## Data Preprocessing

We checked if the data contains any duplicates by using duplicated().any() function and found that the data has no duplicates and is very consistent. We identified that there were more than one genre for many movies in the Movies data in the format ‘Action|Adventure|Drama’ and we performed string splitting using str.split('|') function and expanded the dataset to identify that a movie can be associated with a maximum of 6 genres. We created a new dataset called movie genre with columns Movie ID, title

and the 18 uniquely identified genres. We populated the genre columns with 1 if the movie belongs to that category and 0 if the movie does not belong to that category.

| Genres |
|---|
| Animation\|Children's\|Comedy |
| Adventure\|Children's\|Fantasy |
| Comedy\|Romance |
| Comedy\|Drama |
| Comedy |

| Animation | Adventure | Comedy | Action | Drama |
|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |

# Types of Recommender Systems

## 1) Popularity Based Approach:

Some intuitive recommendations can be made by simpler approaches and popularity based is one such method where movies which are liked by most number of users will be recommended to all users. The most popular movies would be the same for each user since popularity is defined on the entire user pool. It is helpful in recommending movies when there is a new user with no user profile. This approach fails in most situations as there is no personalization involved and does not care about the users choices. By calculating the count of users who rated the movie and then sorting the user count based on Movie ID, we derived the most popular movies to be recommended. This approach identified American Beauty, Star Wars, Jurassic Park for recommending to all users. The popular movie list contains many action/adventure movies but it may not be liked by all the users.

| | MovieID | Title | Genres |
|---|---|---|---|
| 0 | 2858 | American Beauty (1999) | Comedy\|Drama |
| 1 | 260 | Star Wars: Episode IV - A New Hope (1977) | Action\|Adventure\|Fantasy\|Sci-Fi |
| 2 | 1196 | Star Wars: Episode V - The Empire Strikes Back... | Action\|Adventure\|Drama\|Sci-Fi\|War |
| 3 | 1210 | Star Wars: Episode VI - Return of the Jedi (1983) | Action\|Adventure\|Romance\|Sci-Fi\|War |
| 4 | 480 | Jurassic Park (1993) | Action\|Adventure\|Sci-Fi |

## 2) Collaborative Filtering based Recommender Systems

### a) Collaborative filtering – User Based

It is method of making automatic predictions about the interest of the user by collecting preferences from many users(collaborating). Note that these predictions are specific to the user, but uses

information gleaned from many users. (Wikipedia definition)

| User | Roman Holiday | Sabrina | Notorious | The Third Man |
|---|---|---|---|---|
| Swati | 1 | 1 | 5 | 5 |
| Ravi | 5 | | 1 | |
| Prachi | 5 | 5 | 1 | 2 |
| Gaurav | 1 | 1 | 5 | 5 |

**Figure: 1.1**

In User Based (User- User) Collaborative filtering, the recommendation is based on the similarity between the users. In the above Utility Matrix, we have users name as row index and movies as column, the cell contains the rating given by each user to the specific movies he has watched and rated. Now if we have to recommend a movie to Ravi we will find the group of similar users to Ravi and suggest the movies which was liked by other similar users. Just looking at the table, we can say that, both Prachi and Ravi rated Roman Holiday as 5 and Notorious as 1. Whereas, users Gaurav and Swati do not like Roman Holiday but given 5 rating to Notorious. Hence, Ravi is most likely to rate Sabrina better based on Prachi's rating to Sabrina and the similarity between Prachi and Ravi. The key is to find set of similar users or the neighborhood of User Ravi and to do that we need to define the notion of similarity between the users.

## Similarity Matrix:

For any recommender system, it is crucial to use the correct Similarity matrix to generate better results. In our project we focused on three main similarity matrices:

UTILITY MATRIX

| | HP1 | HP2 | HP3 | TW | SW1 | SW2 | SW3 |
|---|---|---|---|---|---|---|---|
| *A* | 4 | | | 5 | 1 | | |
| *B* | 5 | 5 | 4 | | | | |
| *C* | | | | 2 | 4 | 5 | |
| *D* | | 3 | | | | | 3 |

**Jaccard Similarity**

It is also known as intersection over Union. For the figure shown above, if we must find the Jaccard similarity index between user A and B, then the steps are as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

- Count the number of members which are shared between both sets A and B, the common movie they watched and rated(HP1) which is equal to 1.

- Count the total number of members in both sets (shared and un-shared) 5 in our case.
- Divide the number of shared members (1) by the total number of members (5).

Jaccard similarity between A and B, Sim(A,B)=1/5 whereas between A and C is Sim(A,C)= 2/4. Using this method tells us that User A and C are more similar than A and B. This is counter to the intuition we want to capture. So, we will abandon this notion to use Jaccard Similarity for this filtering. The disadvantage of using Jaccard similarity is that it ignores the rating value for each user to the movies. It just considers that user has rated the movie or not. Hence, in User Based Collaborative filtering this method cannot be used. We are using it for content based filtering which is discussed later.

**Cosine Similarity**

It is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them. The cosine of 0° is 1, and it is less than 1 for any other angle. In our dataset the ratings can be treated as vectors and we can compute the cosine similarity of the vectors.

$$Sim(A,B) = \cos(r_{A,}r_B) = 0.38 \qquad Sim(A,C) = \cos(r_{A,}r_C) = 0.32$$

This captures the fact that similarity between A- B is greater than A-C (but not that much). This still doesnot capture that A and C do not have similar likes and dislikes. To calculate the cosine similarity, we need to insert some value for unknown ratings and the simplest way is to treat them 0. The problem with Cosine similarity is it treat the missing ratings as negative. In our rating scale 0-5, 0 is the worst possible rating. So, we have assumed if A is not rated HP2 he would give it rating 0, which might be not the case.

**Centered Cosine Similarity**

To overcome the disadvantage of Cosine similarity, we are going to use Centered cosine similarity. The steps for this are:

- Normalize the ratings by subtracting the row mean or average rating given by user.
- Calculate each rating given by the user for the movies by subtracting the row mean except the missing ratings.
- If sum all the values of each row it is going to be 0. We have centered the rating for each user around 0. The positive rating specifies that user liked the movie more than average whereas the

negative rating means user liked the movie less than average. The magnitude of rating shows how much user liked or disliked the movie.

- Replace blank values with 0.
- Compute cosine similarity.

$$Sim(A,B) = \cos(r_A, r_B) = 0.09 \qquad Sim(A,C) = \cos(r_A, r_C) = -0.56$$

This captures the fact the similarity of user B with A and shows that A and C are very unlike each other. This is also called as **Pearson Correlation**.

Algorithm of User based Collaborative filtering:

1. Read Ratings Dataset

| | UserID | MovieID | Rating | Timestamp |
|---|---|---|---|---|
| 0 | 1 | 1193 | 5 | 978300760 |
| 1 | 1 | 661 | 3 | 978302109 |
| 2 | 1 | 914 | 3 | 978301968 |
| 3 | 1 | 3408 | 4 | 978300275 |
| 4 | 1 | 2355 | 5 | 978824291 |

2. Create Utility Matrix for data with User ID as rows and Movie ID as columns with each row containing the rating

```
# Preparing for Data Analysis
#user by Item utility matrix
movies_crosstab = pd.pivot_table(data=df, values='Rating', index='UserID', columns='MovieID')
movies_crosstab.head(10)
#movies_crosstab.count()
```

| MovieID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 3943 | 3944 | 3945 | 3946 | 3947 | 3948 | 3949 | 3950 | 3951 | 3952 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| UserID | | | | | | | | | | | | | | | | | | | | | |
| 1 | 5.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | 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 | 2.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 6 | 4.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 7 | NaN | NaN | NaN | NaN | NaN | 4.0 | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 8 | 4.0 | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 9 | 5.0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | 3.0 | NaN | NaN | NaN | NaN |
| 10 | 5.0 | 5.0 | NaN | NaN | NaN | NaN | 4.0 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | NaN | 4.0 | NaN | NaN | NaN | NaN |

10 rows × 3706 columns

3. Normalize the resulting matrix
4. Replace null values with 0

```
# Replacing NAN with 0
scale_crosstab[np.isnan(scale_crosstab)] = 0
scale_crosstab.head(6)
```

| UserID | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | ... | 6031 | 6032 | 6033 | 6034 | 6035 | 6036 | 6037 | 6038 | 6039 | 6040 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.811321 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | -1.901408 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 6 | -0.188679 | 0.0 | 0.0 | 0.0 | 0.0 | 0.000000 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

6 rows × 6040 columns

5. Calculate Cosine Similarity and sort the values in descending order to find similar Users

## Finding similar users

```
recc_user = similar_User.sort_values(1, ascending=False)
recc_user.head()
recc_user = recc_user.head(11)
recc_user['corr_user'] = recc_user.index
recc_user
```

| | 1 | corr_user |
|---|---|---|
| 1 | 1.000000 | 1 |
| 1180 | 0.319034 | 1180 |
| 2831 | 0.315454 | 2831 |
| 2766 | 0.313474 | 2766 |
| 5130 | 0.285433 | 5130 |
| 4595 | 0.285348 | 4595 |
| 2543 | 0.284072 | 2543 |
| 266 | 0.283534 | 266 |
| 439 | 0.276575 | 439 |
| 4361 | 0.269354 | 4361 |
| 4702 | 0.267480 | 4702 |

6. Take the top 10 users and the movies they watched
7. Recommend the best rated movies by the Users similar to User A

## Reccomending top rated Movies to the User

```
summary = pd.merge(recc_movies, movies_df,on='MovieID')
summary
summary.sort_values('Rating_x', ascending=False)[['MovieID','Title','Genres','directorName','Rating_x']].head(10)
```

| | MovieID | Title | Genres | directorName | Rating_x |
|---|---|---|---|---|---|
| 0 | 1250 | Bridge on the River Kwai, The (1957) | Drama\|War | David Lean | 5 |
| 201 | 1302 | Field of Dreams (1989) | Drama | Phil Alden Robinson | 5 |
| 203 | 1304 | Butch Cassidy and the Sundance Kid (1969) | Action\|Comedy\|Western | George Roy Hill | 5 |
| 205 | 1304 | Butch Cassidy and the Sundance Kid (1969) | Action\|Comedy\|Western | George Roy Hill | 5 |
| 206 | 648 | Mission: Impossible (1996) | Action\|Adventure\|Mystery | J.J. Abrams | 5 |
| 208 | 3873 | Cat Ballou (1965) | Comedy\|Western | Elliot Silverstein | 5 |
| 210 | 3196 | Stalag 17 (1953) | Drama\|War | Billy Wilder | 5 |
| 212 | 3363 | American Graffiti (1973) | Comedy\|Drama | George Lucas | 5 |
| 216 | 2716 | Ghostbusters (1984) | Comedy\|Horror | Ivan Reitman | 5 |
| 222 | 1951 | Oliver! (1968) | Musical | Carol Reed | 5 |

## b) Collaborative Filtering Item Based Recommender System

Item-item collaborative filtering, or item-based is a form of collaborative filtering for recommender systems based on the similarity between items calculated using people's ratings of those items. Item-item collaborative filtering was invented and used by Amazon.com in 1998.

Earlier collaborative filtering systems based on rating similarity between users (known as user-user collaborative filtering) had two major problems:

- **Scalability:** The computation increases as the number of users increases. User-based methods work fine for thousands of users, but scalability gets to be a problem when we have a million users.
- **Sparsity:** Most recommendation systems have many users and many products but the average user rates a small fraction of the total products.

Algorithm Used to Implement Item Based Filtering: Weighted Slope One

**Weighted Slope One** is a family of algorithms used for collaborative filtering. Arguably, it is the simplest form of non-trivial item-based collaborative filtering based on ratings. Their simplicity makes it especially easy to implement them efficiently while their accuracy is often on par with more complicated and computationally expensive algorithms.

Here's the basic idea on the working of algorithm.

Suppose Ravi gave a rating of 3 to Toy Story and rating of 4 to Troy. Swati gave a rating of 4 to Toy Story. We'd like to predict how Swati would rate Troy. In table form the problem might look like this:

| User/Movies | Toy Story | Troy |
|---|---|---|
| Ravi | 3 | 4 |
| Swati | 4 | ? |

To guess what Swati might rate Troy we could reason as follows. Ravi rated Troy one whole point better than Toy Story. We can predict then that Swati would rate Troy one point higher so we will predict that Swati will give it a rating '5'.

The functioning of Slope-One algorithm takes place in two parts.

- Calculation of deviation matrix or deviation between every pair of items.
- Making predictions using deviations matrix and user ratings.

Part 1: Computing Deviation

The first step is to compute the deviations. The average deviation of an item *i* with respect to item *j* is:

$$dev_{i,j} = \sum_{u \in S_{i,j}(X)} \frac{u_i - u_j}{card(S_{i,j}(X))}$$

where card(S) is how many elements are in S and X is the entire set of all ratings. So card($S_{j,i}$(X)) is the number of people who have rated both j and i. Let's consider the deviation of Toy Story with respect to Jumanji. In this case, card($S_{j,i}$(X)) is 2—there are 2 people (Ravi and Swati) who rated both Toy Story and Jumanji. The $u_j - u_i$ numerator is (that user's rating for Toy Story) minus (that user's rating for Jumanji). So, the deviation is:

| User/Movies | Toy Story | Jumanji | Nixon |
|---|---|---|---|
| Ravi | 4 | 3 | 4 |
| Swati | 5 | 2 | ? |
| Gaurav | ? | 3.5 | 4 |
| Prachi | 5 | ? | 3 |

$$\boldsymbol{dev_{ToyStory,Jumanji}} = \frac{4-3}{2} + \frac{5-2}{2} \boldsymbol{= 2}$$

So, the deviation of ToyStory to Jumanji is 2 meaning that on average users have rated Jumanji 2 better than ToyStory.

Similarly, we will calculate deviation among other movies. Our final deviation matrix is show below.

| Movies | Toy Story | Jumanji | Nixon |
|---|---|---|---|
| Toy Story | 0 | 2 | 1 |
| Jumanji | -2 | 0 | -0.075 |
| Nixon | -1 | 0.75 | 0 |

Part 2: Making predictions with Weighted SlopeOne

Once our deviation matrix is ready we can use that collection to make predictions. The formula to make predictions is:

$$\boldsymbol{P^{wS1}(u_j) = \frac{\sum_{i \in S(u)-\{j\}} (Dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u)-\{j\}} c_{j,i}}}$$

where

$c_{j,i}$ = card($S_{j,i}(X)$)

PwS1(u)j means our prediction using Weighted Slope One of user u's rating for item j. So, for example $\boldsymbol{P^{wS1}}$ ($Swati_{Nixon}$) means our prediction for what Swati would rate Nixon.

Let's say I am interested in answering that question: How might Swati rate Nixon?

| User/Movies | Toy Story | Jumanji | Nixon |
|---|---|---|---|
| Swati | 5 | 2 | ? |

User Rating Matrix

| User/Movies | Toy Story | Jumanji | Nixon |
|---|---|---|---|
| Ravi | 4 | 3 | 4 |
| Swati | 5 | 2 | ? |
| Gaurav | ? | 3.5 | 4 |
| Prachi | 5 | ? | 3 |

Deviation Matrix

| Movies | Toy Story | Jumanji | Nixon |
|---|---|---|---|
| Toy Story | 0 | 2 | 1 |
| Jumanji | -2 | 0 | -0.075 |
| Nixon | -1 | 0.75 | 0 |

Our algorithm will work as per the below steps:

**Numerator**

1. Swati has rated Toy Story and gave it a rating 5—that is the $u_i$.

2. The deviation of Nixon with respect to Toy Story is -1 — this is the $dev_{j,i.}$

3. $dev_{j,i}$ + $u_i$ then is 4.

4. Looking at user rating matrix we see that there were two people (Ravi and Prachi) that rated both Toy Story and Nixon so $c_{j,i}$ = 2

5. So ($dev_{j,i}$ + $u_i$) $c_{j,i}$ = 4 x 2 = 8

6. Swati has rated Jumanji and gave him a 2.

7. The deviation of Nixon with respect to Jumanji is 0.75

8. $dev_{j,i}$ + $u_i$ then is 2.75

9. Two people rated both Nixon and Jumanji so ($dev_{j,i}$ + $u_i$) $c_{j,i}$ = 2.75 x 2 = 5.5

10. We sum up steps 5 and 9 to get 13.5 for the numerator

**Denominator**

11. Dissecting the denominator we get something like for every movie that Swati has rated, sum the cardinalities of those movies (how many people rated both that Movie and Nixon). So, Swati has rated Toy Story and the cardinality of Toy Story and Nixon (that is, the total number of people that rated both of them) is 2. Swati has rated Jumanji and its cardinality is also 2. So, the denominator is 4.

12. So our prediction of how well Swati will like Nixon is 13.5/4 = 3.375

**Algorithm Implementation in Python:**

1. In order to implement slope one algorithm, we used **Surprise** (Simple Python Recommendation System Engine) package.
2. We divided our data into training and test dataset with 900K records in training dataset and 100K in test (training: test 90:10).
3. RMSE and MAE were used as evaluation parameters to analyze the performance of the algorithm. (RMSE:0.002, MAE:0.71)
4. In order to recommend top five movies for a user, we first identified the movies which user has not watched in the past.
5. Predicted rating for unwatched movies were sorted in descending order and top five movies were selected.

## 3) Content based recommendation system

**What is it?**

Content based recommender systems are active information filtering systems which personalize the information coming to a user based on his interests, relevance of the information etc. Such systems are used widely for recommending movies, articles, restaurants, places to visit, items to buy etc. Content-based recommendation systems try to recommend items similar to those a given user has liked in the past.

**How does it work?**

A content based recommender works with data that the user provides, for example a rating, which reveals the user's interests. Based on that data, a user profile is generated, which is then used to make suggestions to the user. As the user provides more inputs or takes actions on the recommendations, the engine becomes more and more accurate.

Advantages of content based filtering

- **Relevant results**: Since content-based recommendations rely on characteristics of objects themselves, they are likely to be highly relevant to a user's interests. This makes them especially valuable for organizations with massive libraries of a single type of content.
- **Quick start for user**: Content-based filtering avoids the cold-start problem. While the system still needs some initial inputs from users to start making recommendations, the quality of those early recommendations is likely to be much higher than with a system that only becomes robust after millions of data points have been added and correlated.
- **Immediate start for items:** Another issue with collaborative-filtering is that new objects added to the library will not have any interactions, which means they won't be recommended. Content-based recommenders don't require other users to interact with an object before it starts recommending it.
- **Easier to implement**: The data science behind a content-based system is relatively straightforward. The real work is in assigning the attributes in the first place.

**Limitations of this system**

Content based recommenders are not good at capturing inter-dependencies or complex behaviors. For example: I might like English movies only when they are available with subtitles or dubbed in my language, but not otherwise. This type of information cannot be captured by these recommenders.

Algorithm

1. Bring down 5 point rating scale of user to a binary 1 (like) and 0 (dislike)
2. Break down movies into binary (1/0) availability of genres
3. Create user profile by dot product of user preference and item profile created in steps 1 and 2.
4. Calculate Jaccard similarity coefficient
5. Recommend movies.

The following section explains the steps and python coding in detail.

**Python code Explanation**

We will now explain the actual steps in making a content based recommendation system.

**Step 1 - Feature Engineering**

Here we are interested in the feature whether user liked the movie or disliked the movie. For that we will convert the rating given by the user to like or dislike. Rating greater than 3 would be categorized as 'like' or '1' and less than or equal to 3 would be categorized as 'dislike' or '0'.

The following code was used to convert the rating

```
#converting teh rating to 1 and -1 if the rating is above 3 that means user liked the ovie otherwise user did not like
for i in range(0, len(binaryRatings)):
    if(binaryRatings.iat[i,2]>3):
        binaryRatings.iloc[i,2]=1
    else :
        binaryRatings.iloc[i,2]=-1
```

**Step 2 - Creation of Item Profile**

Here, we break down each movie (item) into availability of various genres. For example, if a particular movie has Action, Comedy, Sci-Fi genres, then it will have values of '1' against each genre and '0' against other genres.

```
#convert the data from long to wide format
ratmat=binaryRatings.pivot(index='UserId',columns='MovieID',values='Rating').fillna(0)
```

| | MovieID | Action | Adventure | Animation | Children's | Comedy | Crime | Documentary | Drama | Fantasy | Film-Noir | Horror | Musical | Mystery | Romance | Sci-Fi | Thriller |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 2 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 3 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 3 | 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Step 3 - Creation of User Profile**

**User**

To create the user profile, dot product of the movie genre matrix and binary rating matrix, is taken.

A user profiles shows the aggregated inclination of each user towards movie genres. Each column represents a unique userId, and positive values shows a preference towards a certain genre.

```
from scipy import linalg,dot
userprof = dot(ratmat1,itemprof)/linalg.norm(ratmat1)/linalg.norm(itemprof)
userprof
```

The values were again simplified into a binary matrix — positive values were mapped to 1 to represent likes, negative values were mapped to 0 to represent dislikes.

```
for i in range(0, userprof_df.shape[0]):
    for j in range(0,userprof_df.shape[1]):
        if(userprof_df.iat[i,j]>0):
            userprof_df.iloc[i,j]=1
        else :
            userprof_df.iloc[i,j]=0
```

**Step 4 - Calculation of Jaccard Similarity**

In this Assume that users like similar items, and retrieve movies that are closest in similarity to a user's profile, which represents a user's preference for an item's feature.

```
# Method to compute Jaccard similarity index between two sets
def compute_jaccard(user1_vals, user2_vals):
    intersection = user1_vals.intersection(user2_vals)
    union = user1_vals.union(user2_vals)
    jaccard = len(intersection)/float(len(union))
    return jaccard
```

**Step 5 – Recommending top 5 movie to a particular user**

```
#Recommending these 5 movies to the user id1
df[4441].nlargest(5)
```

```
title
Lady and the Tramp (1955)     0.888889
Little Mermaid, The (1989)    0.888889
Pocahontas (1995)             0.833333
Goofy Movie, A (1995)         0.833333
Aladdin (1992)                0.833333
Name: 4441, dtype: float64
```

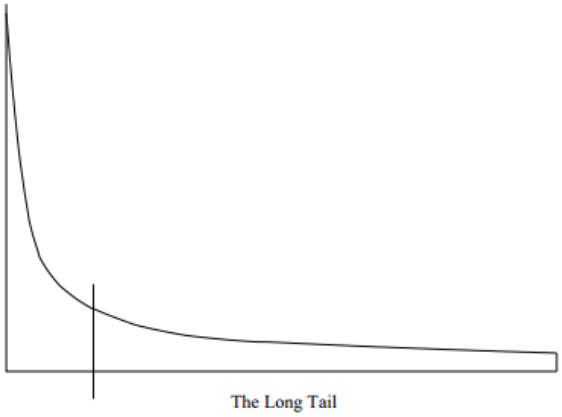
# Next Step

## Long Tail

It's not always possible to display all resources in a physical delivery system which are constrained in space. Example, a bookstore has limited racks to show all books, but on the other hand an online store can show all as its virtual, such as Amazon! The same holds good for virtual reading materials such as e-newspapers. They contain almost everything a physical paper can't due to fixed number of pages.

This reduced space in the physical world helps make physical world recommendations easier. It's not possible to tailor a physical store to everyone's needs. The choices displayed are usually decided by aggregating the likes of a vast pool of customers over time. This very difference between the physical store and the online commercial world is called the long tail phenomenon.

In the below figure, the vertical axis represents popularity (the number of times an item is chosen). The items are ordered on the horizontal axis according to their popularity. Physical institutions provide only the most popular items to the left of the vertical line, while the corresponding on-line institutions provide the entire range of items: the tail as well as the popular items.



The Long Tail

The long-tail phenomenon forces on-line institutions to recommend items to individual users. It is not possible to present all available items to the user, the way physical institutions can. Neither can we expect users to have heard of each of the items they might like. There are several researches going on in this field to improve the existing recommendation algorithms and overcome the problem of long tail.

# References


1. Analytics Vidhya: https://www.analyticsvidhya.com/blog/2015/08/beginners-guide-learn-content-based-recommender-systems/
2. Springer: https://link.springer.com/chapter/10.1007/978-0-387-85820-3_3
3. Upwork: https://www.upwork.com/hiring/data/what-is-content-based-filtering/
4. http://www.statisticshowto.com/jaccard-index/