A Recommendation System For Movies

Faculty of Engineering & Technology, Jadavpur University in the fulfilment of the requirements for the degree of Master of Computer Application

Submitted by

Mandakini Das

Registration Number: 133664 of 2015-16

Class Roll Number: 001510503002

Examination Roll No: MCA186002

Under the Supervision of

Dr. Sarmistha Neogy

Professor, Dept. of Computer Science & Engineering

Jadavpur University

Dept. of Computer Science & Engineering

Faculty of Engineering and Technology

Jadavpur University

May 2018

Declaration of Originality &
Compliance of Academics Ethics

I hereby declare that this thesis contains literature survey and original research work done by me, as part of my MCA studies. All information in this document have been obtained and presented in accordance with academic rules and ethical conduct.

Name	: **Mandakini Das**
Registration No	: 133664 of 2015-16
Class Roll No	: 001510503002
Examination Roll No	: MCA186002
Project Report Title	: A recommendation system for movies

Mandakini Das

Department of Computer Science & Engineering

Faculty of Engineering & Technology

Jadavpur University

To Whom It May Concern

This is to certify that MANDAKINI DAS, Registration Number: 133664 of 2015-16, Class Roll Number: 001510503002, Examination Roll Number: MCA186002, a student of MCA, from the Department of Computer Science & Engineering, under the Faculty of Engineering and Technology, Jadavpur University has done a project report under my supervision, entitled as "A recommendation system for movies". The thesis is approved for submission towards the fulfilment of the requirements for the degree of Master of Computer Application, from the Department of Computer Science & Engineering, Jadavpur University for the session 2017-18.

____

Dr. Sarmistha Neogy
(Supervisor)
Professor
Department of Computer Science and Engineering
Jadavpur University

____

Dr. Ujjwal Maulik
(Head of the Department)
Professor
Department of Computer Science and Engineering
Jadavpur University

____

Dr. Chiranjib Bhattacharjee
(Dean)
Professor
Faculty of Engineering and Technology
Jadavpur University

Certificate of Approval

(Only in case the project report is approved)

The forgoing thesis is hereby approved as a creditable study of an engineering subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite to the degree for which it has been submitted. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein, but approve this thesis only for the purpose for which it is submitted.

____________________________ ____________________________

Signature of the Examiner Signature of the Examiner

Date: _______________ Date: _______________

Acknowledgement

I would like to express my deepest gratitude to my advisor and guide Dr. Sarmistha Neogy, for her excellent guidance, care, patience, and providing me with an excellent atmosphere for doing research. I received a lot of encouragement and inspiration from her throughout the project. I am equally grateful to Dr. Ujjwal Maulik, Head of The Department, Computer Science & Engineering, Jadavpur University, for his support towards our department. Last but not the least, I would like to thank my parents and all respected teachers for their valuable suggestions and helpful discussions.

Regards,

Mandakini Das

Examination Roll No : MCA186002

Master of Computer Application

Jadavpur University

CONTENT

CHAPTER 1

INTRODUCTION

As we know that, the world is growing faster like never before. Everyone is rushing for their ultimate goals. This thirst results into the development in almost every sector. Online business is one of them. We people, don’t have time to shop from market and this is not the end. We don't even have time to choose the object from the collection. This created the embryo of online shopping, which nowadays, has become a huge tree with tons of branches.

As the online market grows exponentially, it’s obvious that competition will enter in this field also. Now, owners of their respective sites need to attract their users by providing attractive facilities. Recommender Engines is one of the facilities given to users.

Recommender engine is the most immediately recognizable machine learning technique in use today. We have seen services or sites that attempt to recommend books or movies or articles based on our past actions. They try to infer tastes and preferences and identify unknown items that are of interest [1].

Abstract Recommender Systems (RSs) are software tools and techniques providing suggestions for items to be of use to a user [2]. A recommender system is a technology that is deployed in the environment where items (products, movies, events, articles) are to be recommended to users (customers, visitors, app users, readers) or the opposite. Typically, there are many items and many users present in the environment making the problem hard and expensive to solve. Imagine a shop. Good merchant knows personal preferences of customers. Her/His high quality recommendations make customers satisfied and increase profits. In case of online marketing and shopping, personal recommendations can be generated by an artificial merchant: the recommender system .

This report presents an overview of recommendation systems and some examples of recommendation engines in Chapter 2. This report also contains details about different types of recommendation systems along with their advantages and disadvantages in Chapter 3. Here we have designed a basic model of simple recommendation system as well as collaborative filtering based recommender system on movie dataset. The details are discussed in Chapter 4. Chapter 5 concludes the work.

CHAPTER 2

RELATED WORKS

Recommender system is defined as a decision making strategy for users under complex information environments. Also, recommender system was defined from the perspective of E-commerce as a tool that helps users search through records of knowledge which is related to user's interest and preference. Recommender system was defined as a means of assisting and augmenting the social process of using recommendations of others to make choices when there is no sufficient personal knowledge or experience of the alternatives. Recommender systems handle the problem of information overload that users normally encounter by providing them with personalized, exclusive content and service recommendations [6].

Recommender systems have become increasingly popular in recent years, and are utilized in a variety of areas including movies, music, news, books, research articles, search queries, social tags, and products in general. There are also recommender systems for experts, collaborators, jokes, restaurants, garments, financial services, life insurance, online dating and Twitter pages.

2.1 Offline Recommendation Engines

In the external world, we can think of the people around us as recommendation engines.

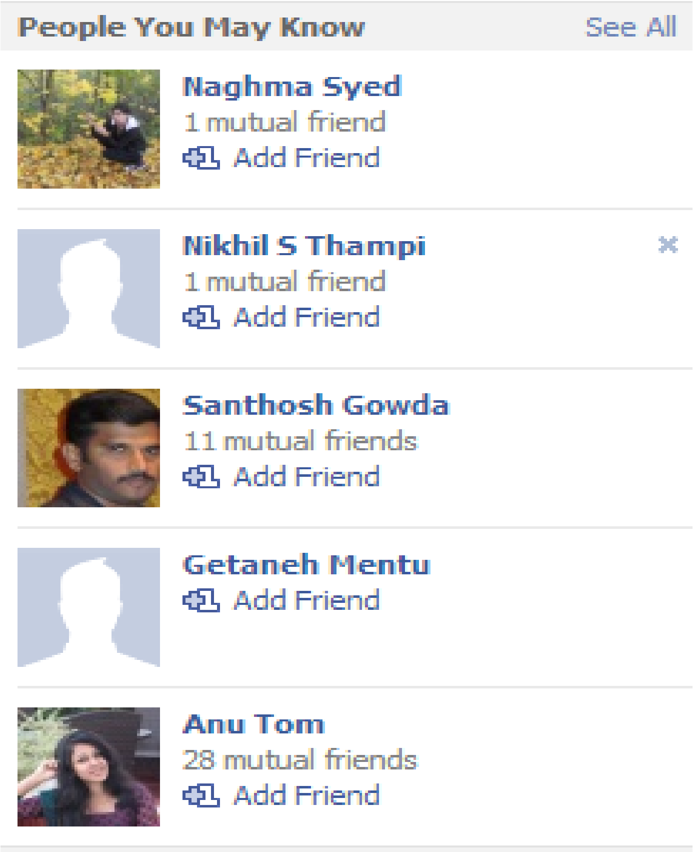
- **Your family and friends as clothes recommendation engines:** With the thousands of style options now available to us, we often rely on friends and family to recommend stores, styles and tell us what looks good on us.

- **Your Professors as book recommendation engines:** When want to research or better understand a concept, our Professors can lead us to the titles which best suit our needs
- **Your friends as movie recommendation engines:** If you have friends who know your cinematic tastes well, you're likely to trust their movie recommendations over a random stranger's picks.

All of these "offline recommenders" know something about *you*. They know your style, taste or area of study, and thus can make more informed decisions about what to recommendations would benefit you most. It is this personalisation- based on getting to "know" you- that online recommenders aim to emulate.

2.2 Online Recommendation Engines

- **Facebook:** *"People You May Know"*



People You May Know See All

Naghma Syed
1 mutual friend
Add Friend

Nikhil S Thampi
1 mutual friend
Add Friend

Santhosh Gowda
11 mutual friends
Add Friend

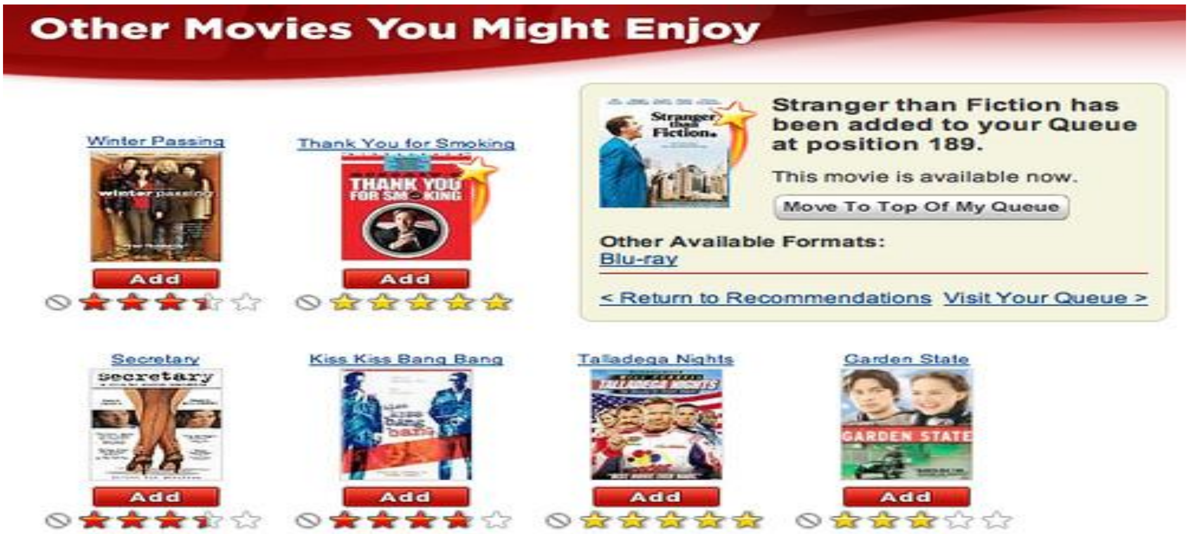
Getaneh Mentu
Add Friend

Anu Tom
28 mutual friends
Add Friend

https://www.google.co.in/search?q=fb+people+you+may+know&rlz=1C1CHZL_enIN709IN709&source=lnms&tbm=isch&sa=X&ved=2ahUKEwidxovdyoTbAhUBNI8KHdrtAS0Q_AUoAnoECAEQBA&biw=1360&bih=672#imgrc=kcjL8JfQPuTyDM:

Facebook uses a recommender system to suggest Facebook users you may know offline. The system is trained on personal data, mutual friends, where you went to school, places of work and mutual networks (pages, groups, etc.), to learn who might be in your offline & offline network.

- **Netflix:** *"Other Movies You Might Enjoy"*


Other Movies You Might Enjoy

Stranger than Fiction has been added to your Queue at position 189.

This movie is available now.

Move To Top Of My Queue

Other Available Formats:
Blu-ray

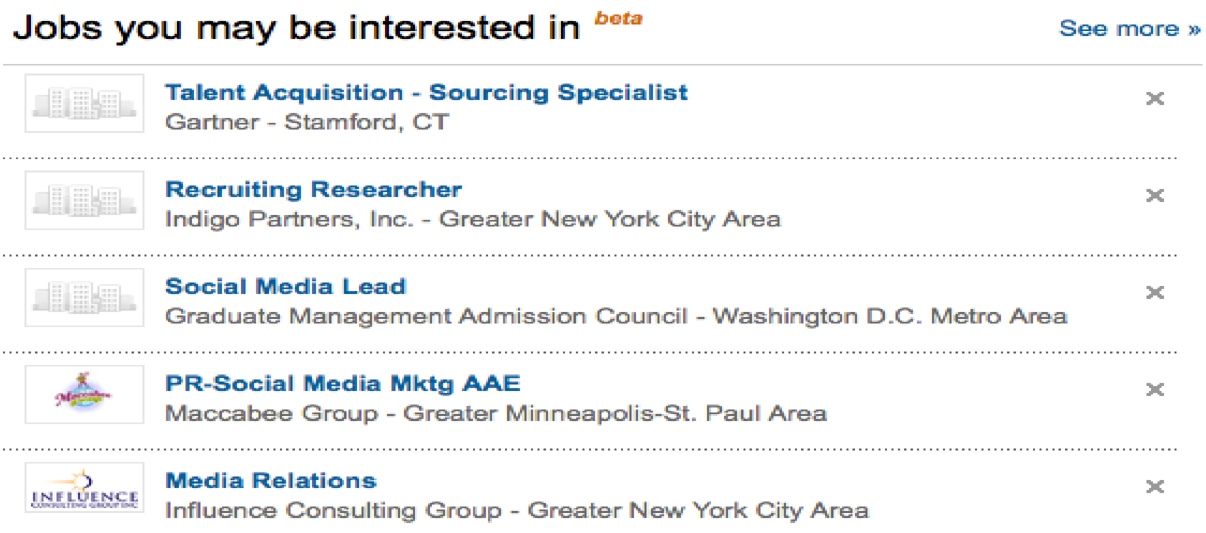
< Return to Recommendations Visit Your Queue >

https://www.google.co.in/search?q=netflix+other+movies+you+may+enjoy&rlz=1C1CHZL_enIN709IN709&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjqqqKPyoTbAhVGs48KHfzBBfgQ_AUoAnoECAAQBA&biw=1360&bih=672#imgrc=GCVkuG7FMBIjSM:

When a user fills out his Taste Preferences or rate movies and TV shows, he is helping Netflix to filter through the thousands of selections to get a better idea of what this user might like to watch. Factors that Netflix algorithm uses to make such recommendations include:

- The genre of movies and TV shows available
- User's streaming history, and previous ratings he has made.
- The combined ratings of all Netflix members who have similar tastes in titles to this user.

- **LinkedIn:** *"Jobs You May be Interested In"*


Jobs you may be interested in beta See more »

Talent Acquisition - Sourcing Specialist
Gartner - Stamford, CT

Recruiting Researcher
Indigo Partners, Inc. - Greater New York City Area

Social Media Lead
Graduate Management Admission Council - Washington D.C. Metro Area

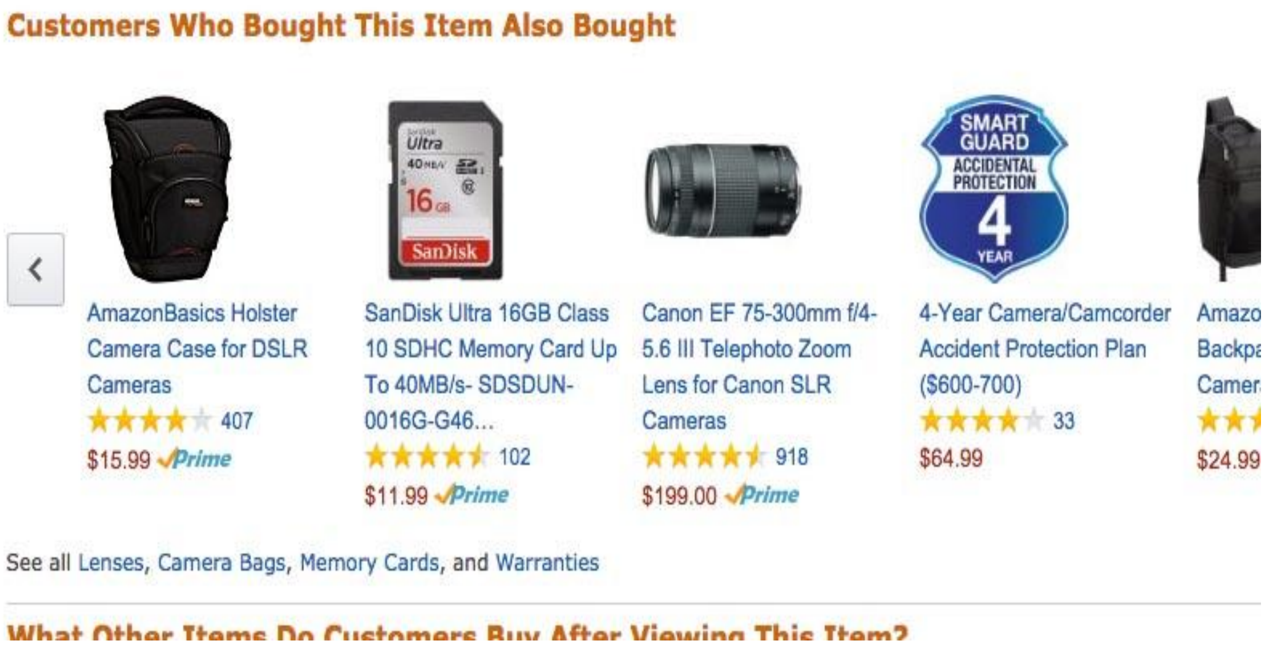
PR-Social Media Mktg AAE
Maccabee Group - Greater Minneapolis-St. Paul Area

Media Relations
Influence Consulting Group - Greater New York City Area

https://www.google.co.in/search?q=linkedin+jobs+you+may+be+interested+in&rlz=1C1CHZL_enIN709IN709&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiQ7sGtyITbAhWHpo8KHQLhBbMQ_AUoA3oECAAQBQ&biw=1360&bih=672#imgdii=gUZeHKvIWRksiM:&imgrc=7hv_MQ9-VolJ3M:

The Jobs You May Be Interested In feature shows jobs posted on LinkedIn that match your profile in some way. These recommendations are based on the titles and descriptions in your previous experience, and the skills other users have “endorsed”.

- **Amazon:** *“Customers Who Bought This Item Also Bought…”*



Customers Who Bought This Item Also Bought

AmazonBasics Holster Camera Case for DSLR Cameras
★★★★☆ 407
$15.99 ✓Prime

SanDisk Ultra 16GB Class 10 SDHC Memory Card Up To 40MB/s- SDSDUN-0016G-G46…
★★★★½ 102
$11.99 ✓Prime

Canon EF 75-300mm f/4-5.6 III Telephoto Zoom Lens for Canon SLR Cameras
★★★★½ 918
$199.00 ✓Prime

4-Year Camera/Camcorder Accident Protection Plan ($600-700)
★★★★☆ 33
$64.99

Amazo
Backpa
Camer
★★★
$24.99

See all Lenses, Camera Bags, Memory Cards, and Warranties

What Other Items Do Customers Buy After Viewing This Item?

SanDisk Ultra 40MB/s 16GB
SMART GUARD ACCIDENTAL PROTECTION 4 YEAR

https://www.google.co.in/search?q=amazon+customers+who+bought+this+also+bought&rlz=1C1CHZL_enIN709IN709&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjs7ZH7yITbAhXIL48KHYYND2IQ_AUoAXoECAAQAw&biw=1360&bih=672#imgrc=c6H03SOnYdLECM:

Amazon’s algorithm crunches data on all of its millions of customer baskets, to figure out which items are frequently bought together. This can lead to huge returns- for example, if a user is buying an electrical item, and sees a recommendation for the cables or batteries it requires beneath it, the user is very likely to purchase both the core product *and* the accessories from Amazon [11].

CHAPTER 3

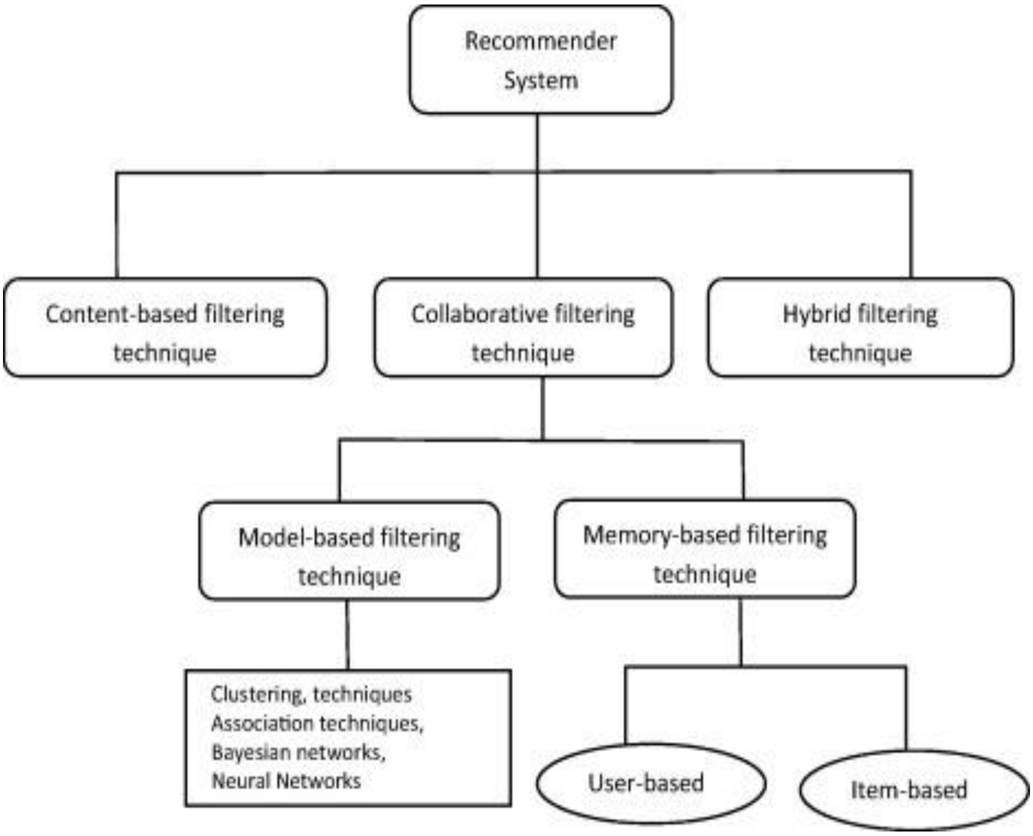
RECOMMENDER SYSTEM BASICS CONCEPT

Recommender systems are information filtering tools that are used to predict the rating for users and items, predominantly from big data to recommend their likes. Movie recommendation systems provide a mechanism to assist users in classifying users with similar interests. This makes recommender systems essentially a central part of websites and e-commerce applications.

In this project we propose a movie recommendation system, where user specific interests are taken into account, to determine recommendations.

A Recommendation System is composed of two modules: a database and a filtering technique. The database is responsible for storing the information about users, items and the associated ratings. The filtering technique is implemented by an algorithm.
There are three important types of recommender systems:

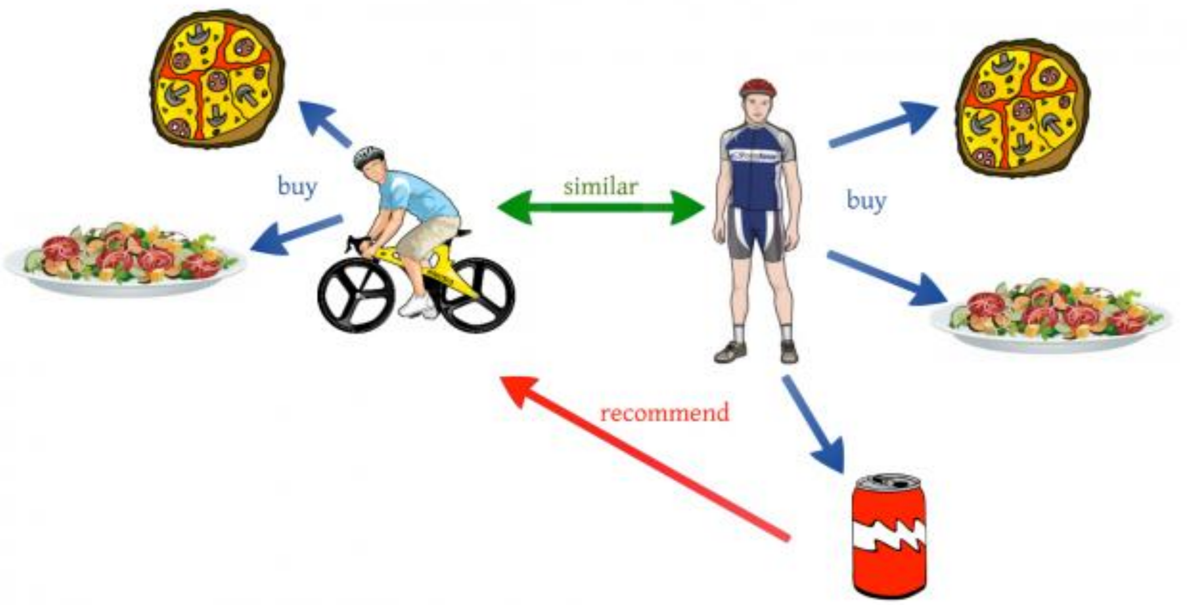
- Collaborative Filtering
- Content based Filtering
- Hybrid Filtering


Recommender System
Content-based filtering technique
Collaborative filtering technique
Hybrid filtering technique
Model-based filtering technique
Memory-based filtering technique
Clustering, techniques
Association techniques,
Bayesian networks,
Neural Networks
User-based
Item-based

https://www.sciencedirect.com/science/article/pii/S1110866515000341

3.1 COLLABORATIVE FILTERING

Collaborative filtering methods are based on collecting and analyzing a large amount of information on users' behaviours, activities or preferences and predicting what users will like based on their similarity to other users. A key advantage of the collaborative filtering approach is that it does not rely on machine analyzable content and therefore it is capable of accurately recommending complex items such as movies without requiring an "understanding" of the item itself. Many algorithms have been used in measuring user similarity or item similarity in recommender systems. For example, the k-nearest neighbour (k-NN) approach and the Pearson Correlation [11].



https://www.google.co.in/search?q=image+of+content+based+filtering&rlz=1C1CHZL_enIN709IN709&tbm=isch&source=iu&ictx=1&fir=hUs6sCIAzL5QRM%253A%252Cn_V7nX1sPBGkFM%252C_&usg=__Exn5uiVnNwZNvhkfFuYCSk77rE%3D&sa=X&ved=2ahUKEwiOuffht5LbAhWJQY8KHcGdAiIQ9QEwAnoECAEQMA#imgdii=_1ELzd__EjY4VM:&imgrc=hUs6sCIAzL5QRM:

For each user, recommender systems recommend items based on how similar users liked the item. Let's say Alice and Bob have similar interests in video games. Alice recently played and enjoyed the game Legend of Zelda: Breathe of the Wild. Bob has not played this game, but because the system has learned that Alice and Bob have similar tastes, it recommends this game to Bob. In addition to user similarity, recommender systems can also perform collaborative filtering using item similarity ("Users who liked this item also liked X") [10].

➢ WHY COLLABORATIVE FILTERING?

The main difference between collaborative filtering and content-based filtering is conceptual. Where content-based filtering is built around the attributes of a given object, collaborative filtering relies on the behavior of users. This approach has some distinct advantages over content-based filtering:

- **It benefits from large user bases.** Simply put, the more people are using the service, the better your recommendations will become, without doing additional development work or relying on subject area expertise.
- **It's flexible across different domains.** Collaborative filtering approaches are well suited to highly diverse sets of items. Where content-based filters rely on metadata, collaborative filtering is based on real-life activity, allowing it to make connections between seemingly disparate items (like say, an outboard motor and a fishing rod) that nonetheless might be relevant to some set of users (in this case, people who like to fish).
- **It produces more serendipitous recommendations.** When it comes to recommendations, accuracy isn't always the highest priority. Content-based filtering approaches tend to show users items that are very similar to items they've already liked, which can lead to filter bubble problems. By contrast, most users have interests that span different subsets, which in theory can result in more diverse (and interesting) recommendations.
- **It can capture more nuance around items.** Even a highly detailed content-based filtering system will only capture some of the features of a given item. By relying on actual human experience, collaborative filtering can sometimes recommend items that have a greater affinity with one another than a strict comparison of their attributes would suggest.

➢ TWO METHODS : USER - ITEM VS ITEM - ITEM

There are two approaches to collaborative filtering, one based on items, the other on users. Item-item collaborative filtering was originally developed by Amazon and draws inferences about the relationship between different items based on which items are purchased together. The more often two items (say, peanut butter and jelly) appear in the same shopping cart or user history, the "closer" they're said to be to one another. So, when someone comes and adds peanut butter to their cart, the algorithm will suggest things that are close, like jelly or white bread, over things that aren't, like motor oil.

User-item filtering takes a slightly different approach. Here, rather than calculating the distance between items, we calculate the distance between users based on their ratings (or likes, or whatever metric applies). When coming up with recommendations for a particular user, we then look at the users that are closest to them and then suggest items those users also liked but that our user hasn't interacted with yet. So, if you've watched and liked a certain number of videos on Facebook, Facebook can look at other users who liked those same videos and recommend one that they also liked but which you might not have seen yet.

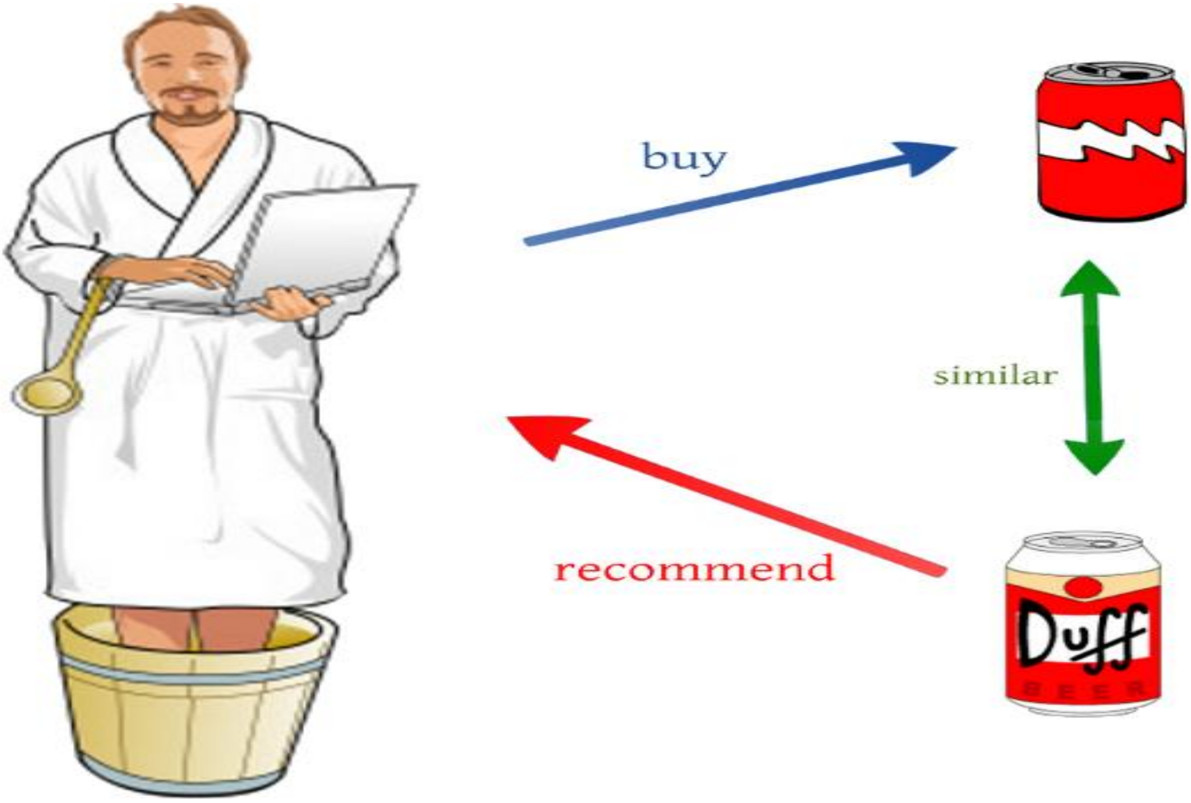
The important point here is that in both the examples above, the system has no idea why any of these items are related to one another, it only knows that they either show up in the same basket together, or that they're liked by people with similar preferences. In some cases, though, this can be a feature rather than a shortcoming, especially in cases where the items to be filtered are extremely heterogeneous, as in online retailers or social networks. (Note: This can also lead to some unanticipated situations, as when Amazon's algorithm began unintentionally suggesting drug paraphernalia to users who bought a particular scale.)

➢ CHALLENGES OF COLLABORATIVE FILTERING

- **Complexity and expense.** Collaborative filtering algorithms can run into scalability problems when the number of users and items gets too high (think in tens of millions of users and hundreds of thousands of items), especially when recommendations need to be generated in real-time online. Potential solution: This is where distributed clusters of machines running Hadoop or Spark come in handy. Depending on your project, it may also be possible to calculate relationships offline overnight by way of batch processing, which makes serving recommendations much quicker even if they're no longer being updated in real-time.
- **Data sparsity.** Many user signals are ambiguous. Just watching a video doesn't tell YouTube whether you liked that particular video or not, and just eating at a restaurant doesn't tell Yelp whether you liked it or not. That's why ratings are so important in collaborative-filtering systems. But users don't rate every item they interact with, and many users don't rate anything at all. Potential solution: Depending on the nature of the data, there may be proxy measures that can be used. Another common technique is to assume that missing reviews are equivalent to average reviews, though this is a very strong assumption in most cases.
- **The "cold start" problem.** As we've seen, collaborative-filtering can be a powerful way of recommending items based on user history, but what if there is no user history? This is called the "cold start" problem, and it can apply both to new items and to new users. Items with lots of history get recommended a lot, while those without never make it into the recommendation engine, resulting in a positive feedback loop. At the same time, new users have no history and thus the system doesn't have any good recommendations. Potential solution: Onboarding processes can learn basic info to jump-start user preferences, importing social network contacts [12].

3.2 CONTENT BASED FILTERING

Content-based filtering methods are based on a description of the item and a profile of the user's preference. In a content-based recommendation system, keywords are used to describe the items; beside, a user profile is built to indicate the type of item this user likes. In other words, these algorithms try to recommend items that are similar to those that a user liked in the past (or is examining in the present). In particular, various candidate items are compared with items previously rated by the user and the best-matching items are recommended. This approach has its roots in information retrieval and information filtering research [11].



https://www.google.co.in/search?q=image+of+content+based+filtering&rlz=1C1CHZL_enIN709IN709&tbm=isch&source=iu&ictx=1&fir=hUs6sCIAzL5QRM%253A%252Cn_V7nX1sPBGkFM%252C_&usg=__Exn5uiVnNwZNvhkfFuYCSk77rE%3D&sa=X&ved=2ahUKEwiOuffht5LbAhWJQY8KHcGdAiIQ9QEwAnoECAEQMA#imgrc=hUs6sCIAzL5QRM:

If companies have detailed metadata about each of your items, they can recommend items with similar metadata tags. For example, let's say I watch the show Bojack Horseman on Netflix. This show may have metadata tags of "Animated", "Comedy", and "Adult", so Netflix recommends other shows with these metadata tags, such as Family Guy. [10]

➢ WHY CONTENT-BASED FILTERING?

Collaborative filtering may be the state of the art when it comes to machine learning and recommender systems, but content-based filtering still has a number of advantages, especially in certain circumstances.

- **Results tend to be highly relevant.** Because 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 (think subscription and streaming media services).

- **Recommendations are transparent.** Another advantage is that the process by which any recommendation is generated can be made transparent, which may increase users' trust in their recommendations or allow them to tweak them. With collaborative-filtering, the process is more of a black box–the algorithm and users alike may not really understand why they're seeing the recommendations that are offered.
- **Users can get started more quickly.** Content-based filtering avoids the cold-start problem that often bedevils collaborative-filtering techniques. 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.
- **New items can be recommended immediately.** Related to the cold-start problem, another issue with collaborative-filtering is that new objects added to the library will have few (if any) interactions, which means they won't be recommended very often. Unlike collaborative-filtering systems, content-based recommenders don't require other users to interact with an object before it starts recommending it.

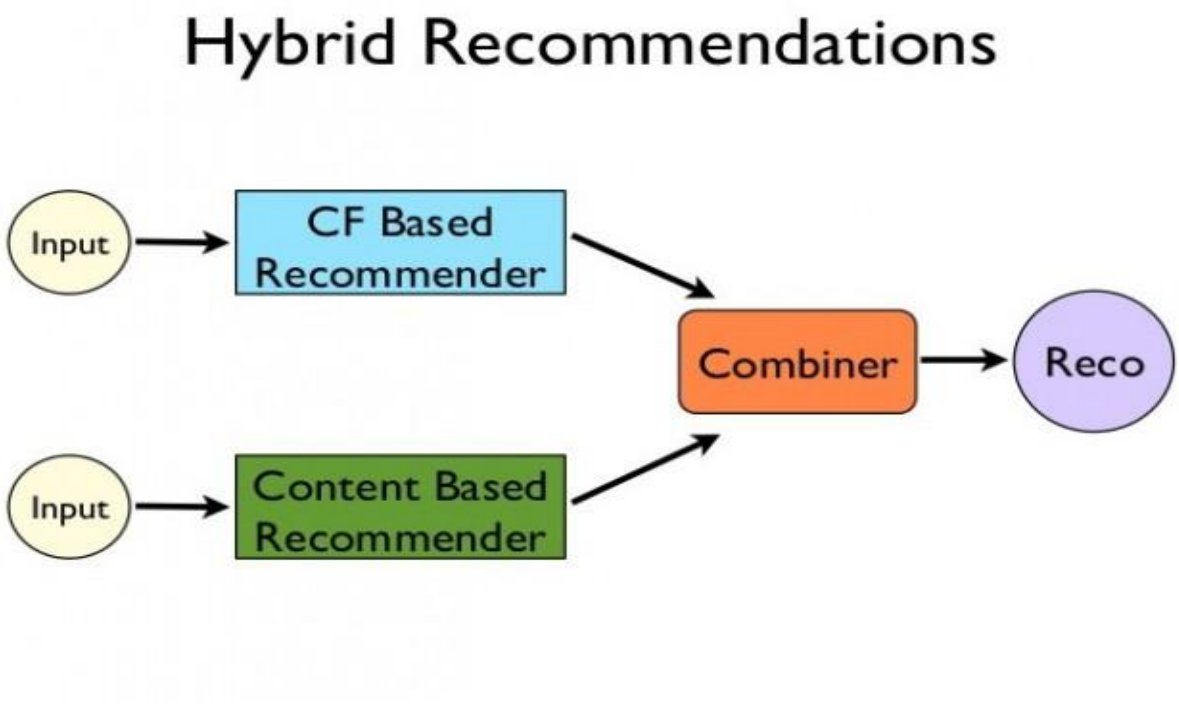
It's technically easier to implement. Compared to the sophisticated math involved in building a collaborative-filtering system, the data science behind a content-based system is relatively straightforward. The real work, as we've seen is in assigning the attributes in the first place.

➢ CHALLENGES OF CONTENT – BASED FILTERING

- **Lack of novelty and diversity.** Relevance is important, but it's not all .If you watched and liked *Star Wars*, the odds are pretty good that you'll also like *The Empire Strikes Back*, but you probably don't need a recommendation engine to tell you that. It's also important for a recommendation engine to come up with results that are novel (that is, stuff the user wasn't expecting) and diverse (that is, stuff that represents a broad selection of their interests).
- **Scalability is a challenge.** As we've seen, the key requirement when it comes to content-based filtering is exceptional domain-specific knowledge. Hiring subject-matter experts can be a labor-intensive and expensive process, making it impractical for many businesses who are just trying to build an MVP. Furthermore, manual tagging of attributes has to continue as new content is added.
- **Attributes may be incorrectly or inconsistently applied.** Content-based recommendations are only as good as the subject-matter experts who are tagging items. When you have hundreds of thousands (or millions) of items, it can be a challenge to ensure attributes are applied consistently or accurately [13].

3.3 HYBRID FILTERING

Recent research has demonstrated that a hybrid approach, combining collaborative filtering and content-based filtering could be more effective in some cases. Hybrid approaches can be implemented in several ways, by making content-based and collaborative-based predictions separately and then combining them, by adding content-based capabilities to a collaborative-based approach (and vice versa), or by unifying the approaches into one model. Several studies empirically compare the performance of the hybrid with the pure collaborative and content-based methods and demonstrate that the hybrid methods can provide more accurate recommendations than pure approaches. These methods can also be used to overcome some of the common problems in recommendation systems such as cold start and the sparsity problem.



http://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/

Netflix is a good example of a hybrid system. They make recommendations by comparing the watching and searching habits of similar users (i.e. collaborative filtering) as well as by offering movies that share characteristics with films that a user has rated highly (content-based filtering). [11]

CHAPTER 4

DESIGN AND IMPLENTATION OF A RECOMMENDER SYSTEM FOR MOVIE

4.1 DATA SET

We will be using the MovieLens dataset [5] for recommendation. It has been collected by the Group Lens research project at the University of Minnesota. The characteristics of the MovieLens 100K dataset are as follows:

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Each user has rated at least 20 movies.
- Simple demographic info for the uses(age, gender, occupation.zip)
- Genre information of movies.

This data is loaded into python. There are many files in ml-100k.zip file which we used. We loaded three important files. There is also a recommendation to read the readme document which gives a lot of information about difference files.

4.1.1 USERS :-

There are 943 rows as there are 943 users and 5 columns for 5 features for each namely their unique user_id, age, sex, occupation, zip_code.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 943 entries, 0 to 942
Data columns (total 5 columns):
user id       943 non-null int64
age           943 non-null int64
gender        943 non-null object
occupation    943 non-null object
zip code      943 non-null object
dtypes: int64(2), object(3)
memory usage: 36.9+ KB
None
```

4.1.2 RATINGS :-

There are 100000 ratings as there are 100K ratings for different users and movie combinations. Also each ratings has a timestamp associated with it.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 4 columns):
user id      100000 non-null int64
movie id     100000 non-null int64
rating       100000 non-null int64
timestamp    100000 non-null int64
dtypes: int64(4)
memory usage: 3.1 MB
None
```

4.1.3 ITEMS :-

This dataset contains attributes of 1682 movies. There are 24 columns out of which 19 specify the genre of a particular movie. The last 19 columns are for each genre and a value of 1 denotes that movie belongs to that genre and 0 otherwise.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1682 entries, 1 to 1682
Data columns (total 24 columns):
movie id              1682 non-null object
movie title           1681 non-null object
release date             0 non-null float64
video release date    1679 non-null object
IMDb URL              1682 non-null int64
unknown               1682 non-null int64
Action                1682 non-null int64
Adventure             1682 non-null int64
Animation             1682 non-null int64
Childrens             1682 non-null int64
Comedy                1682 non-null int64
```

```
Crime            1682 non-null int64
Documentary         1682 non-null int64
Drama             1682 non-null int64
Fantasy           1682 non-null int64
Film-Noir         1682 non-null int64
Horror           1682 non-null int64
Musical           1682 non-null int64
Mystery           1682 non-null int64
Romance             1682 non-null int64
Sci-Fi           1682 non-null int64
Thriller         1682 non-null int64
War              1682 non-null int64
Western            1682 non-null object
dtypes: float64(1), int64(19), object(4)
memory usage: 328.5+ KB
None
```

We also created a dataset by searching directors name of each movie in the internet, where we can see the name of directors of every movie. So in that dataset we have 25 columns where the last column is the names of movie directors.

4.2. IMPLEMENTATION

We have implemented movie recommendation system in python. We have used various library functions. We have used panda and used the scikit-learn library to split the dataset into testing and training.

- We have used **panda** which is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.
- We have used **NumPy** which is the fundamental package for scientific computing with Python.
- Also we have used **codecs** which defines a set of base classes which define the interface and can also be used to easily write your (? our) own **codecs** for use in Python.
- We have used standard **operators** as functions. For example, **operator.add(x, y)** is equivalent to the expression x+y. Many function names are those used for special methods, without the double underscores. For backward compatibility, many of these have a variant with the double underscores kept. The variants without the double underscores are preferred for clarity.
- We have also used **importlib.** The purpose this package is two-fold. One is to provide the implementation of the import statement (and thus, by extension, the __import__() function) in Python source code. This provides an implementation of **import** which is portable to any Python interpreter. This also provides an implementation which is easier to comprehend than one implemented in a programming language other than Python. Two, the components to implement **import** are exposed in this package, making it easier for users to create their own customised objects (known genericlly as an importer) to participate in the import process.
- We have also imported **Scripy.parse** which is SciPy 2-D sparse matrix package for numeric data.

4.3 WORK FLOW AND RESULTS

4.3.1 SIMPLE RECOMMENDATION:-

It is a generalized recommendation to every user, based on movie popularity and/or genre. The basic idea behind this system is that movies that are more popular and critically acclaimed will have a higher probability of being liked by the average audience. IMDB Top 250 is an example of this system.

The dataset has three CSV files named u.data, u.user, u.item.

- *25 most rated movies*

WORKFLOW:

i. Merge 3 dataset (movies , ratings , user)
ii. Name it lens
iii. Count number of times a movie is rated by users from lens
iv. Print top 25 most rated movie of the dataset lens

OUTPUT:

```
                 --*--25 MOST RATED MOVIES ARE --*--
Star Wars (1977)                        583
Contact (1997)                         509
Fargo (1996)                          508
Return of the Jedi (1983)                  507
Liar Liar (1997)                      485
English Patient, The (1996)                481
Scream (1996)                          478
Toy Story (1995)                       452
Air Force One (1997)                      431
Independence Day (ID4) (1996)              429
Raiders of the Lost Ark (1981)             420
Godfather, The (1972)                    413
Pulp Fiction (1994)                    394
Twelve Monkeys (1995)                     392
Silence of the Lambs, The (1991)           390
Jerry Maguire (1996)                    384
Chasing Amy (1997)                       379
Rock, The (1996)                        378
Empire Strikes Back, The (1980)            367
Star Trek: First Contact (1996)            365
Back to the Future (1985)                 350
Titanic (1997)                        350
Mission: Impossible (1996)                 344
Fugitive, The (1993)                     336
Indiana Jones and the Last Crusade (1989)    331
Name: title, dtype: int64
```

- *Highly rated movies*

WORKFLOW:

i. Group the movies by movie title from the dataset lens
ii. Evaluate mean [mean is average of a list of values] of each movie
iii. Store it as a dataset named movie_stats
iv. Sort the dataset movie_stats in descending order
v. Print head of this dataset

OUTPUT:

```
                  --*--HIGHLY RATED MOVIES--*--

                              rating

                                size mean

title

They Made Me a Criminal (1939)                  1  5.0

Marlene Dietrich: Shadow and Light (1996)       1  5.0

Saint of Fort Washington, The (1993)            2  5.0

Someone Else's America (1995)                   1  5.0

Star Kid (1997)                                 3  5.0
```

- Movies rated at least 100 times

WORKFLOW

i. Count the number of times a movie is rated on the dataset movie_stats
ii. Pick up those movies whose count in greater than or equal to 100
iii. Save these movie names in the dataset named atleast_100
iv. Find mean rating of the movies mentioned in the data set atleast_100
v. Sort them in descending order
vi. Print the head of the dataset

OUTPUT

```
             --*-- MOVIES RATED ATLEAST 100 TIMES --*--
                            rating
                              size     mean
title
Close Shave, A (1995)              112  4.491071
Schindler's List (1993)            298  4.466443
Wrong Trousers, The (1993)         118  4.466102
Casablanca (1942)                  243  4.456790
```

```
Shawshank Redemption, The (1994)        283  4.445230
Rear Window (1954)                      209  4.387560
Usual Suspects, The (1995)              267  4.385768
Star Wars (1977)                        583  4.358491
12 Angry Men (1957)                     125  4.344000
Citizen Kane (1941)                     198  4.292929
To Kill a Mockingbird (1962)            219  4.292237
One Flew Over the Cuckoo's Nest (1975)  264  4.291667
Silence of the Lambs, The (1991)        390  4.289744
North by Northwest (1959)               179  4.284916
Godfather, The (1972)                   413  4.283293
```

➢ Age group wise mean rating of movies

WORKFLOW

i. Create labels to name bins [bin is numeric variable converted into categorical variable]
ii. Split users into eight bins of ten years [0-9, 10-19, 20-29, etc]
iii. Create the bin to be exclusive of the max age in the bin
iv. Sort the dataset lens in descending order
v. Store top 50 movies in another dataset named most_50
vi. Find mean rating across the age group on the dataset most_50
vii. Show this data as a table

OUTPUT

```
                 --*-- AGEGROUP WISE MEAN RATING --*--
age_group                          0-9     10-19     20-29     30-39  \
title
E.T. the Extra-Terrestrial (1982)  0.0  3.680000  3.609091  3.806818
Empire Strikes Back, The (1980)    4.0  4.642857  4.311688  4.052083
English Patient, The (1996)        5.0  3.739130  3.571429  3.621849
Fargo (1996)                       0.0  3.937500  4.010471  4.230769
Forrest Gump (1994)                5.0  4.047619  3.785714  3.861702
Fugitive, The (1993)               0.0  4.320000  3.969925  3.981481
Full Monty, The (1997)             0.0  3.421053  4.056818  3.933333
Godfather, The (1972)              0.0  4.400000  4.345070  4.412844
Groundhog Day (1993)               0.0  3.476190  3.798246  3.786667
Independence Day (ID4) (1996)      0.0  3.595238  3.291429  3.389381

age_group                             40-49     50-59     60-69     70-79
title
E.T. the Extra-Terrestrial (1982)  4.160000  4.368421  4.375000  0.000000
Empire Strikes Back, The (1980)    4.100000  3.909091  4.250000  5.000000
English Patient, The (1996)        3.634615  3.774648  3.904762  4.500000
Fargo (1996)                       4.294118  4.442308  4.000000  4.333333
Forrest Gump (1994)                3.847826  4.000000  3.800000  0.000000
Fugitive, The (1993)               4.190476  4.240000  3.666667  0.000000
Full Monty, The (1997)             3.714286  4.146341  4.166667  3.500000
Godfather, The (1972)              3.929412  4.463415  4.125000  0.000000
Groundhog Day (1993)               3.851064  3.571429  3.571429  4.000000
Independence Day (ID4) (1996)      3.718750  3.888889  2.750000  0.000000
```

➢ Men and women most disagree on

WORKFLOW

i. Select average ratings of each movies rated by female and male separately
ii. Find difference between average ratings of male and female
iii. Print head of this dataset

OUTPUT

```
                  --*-- MEN AND WOMEN MOST DISAGREE ON --*--
sex                          F         M      diff
movie_id title
1        Toy Story (1995)   3.789916  3.909910  0.119994
2        GoldenEye (1995)   3.368421  3.178571 -0.189850
3        Four Rooms (1995)  2.687500  3.108108  0.420608
4        Get Shorty (1995)  3.400000  3.591463  0.191463
5        Copycat (1995)     3.772727  3.140625 -0.632102
```

➢ Finding mean rating of a particular movie

WORKFLOW

i. Import data files [u.item, u.user, u.data] onto data frames [item, users, data]
ii. Create a merged data frames [by merging item, users, data] based on similar column
iii. Name it *df*
iv. Group the movies based on their title
v. Name it *ratings total*
vi. Take the mean rating of each movie using the mean function
vii. Name it *ratings_mean*
viii. Convert ratings_total into data frame
ix. Print the top 5 mean rated movie
x. Take as input the name of a movie
xi. Print the mean rating of that particular movie from *ratings_total*

WORKFLOW

```
        rating              movie title  total ratings
1398  4.358491         Star Wars (1977)            583
333   3.803536           Contact (1997)            509
498   4.155512             Fargo (1996)            508
1234  4.007890  Return of the Jedi (1983)          507
860   3.156701        Liar Liar (1997)             485

WRITE A MOVIE NAME : Toy Story (1995)
Toy Story (1995)
        rating      movie title  total ratings
1523  3.878319  Toy Story (1995)            452
```

Finding mean rating of a particular movie genre

WORKFLOW

i. Import data files [u.item, u.user, u.data] onto data frames [movies, users, ratings]
ii. Create one merged Data Frame [by merging movies, ratings, users]
iii. Name it lens
iv. Reshape data frames by melting (Wide to long) selecting only needed fields
v. Name it mdf
vi. Filter for value=1 and the needed columns
vii. Run a pivoted aggregation on mdf
viii. Name it df
ix. Print first 5 elements of df
x. Take input genre name
xi. Print the mean ratings of movies that belongs to that particular genre

OUTPUT

```
genre
Action        3.518712
Adventure     3.527568
Animation     3.578192
Children's    3.372353
Comedy        3.449725
dtype: float64

ENTER A GENERE : Fantasy
3.27630383397
```

Finding top rated movies of a director

WORKFLOW

i. Set work directory to where the data is located
ii. Name the column headers for the datasets [u.item, u,user, u,data]
iii. Import the data files onto data frames
iv. Take input a director name
v. Print the name of those movies whose director name is equals to the input data

OUTPUT

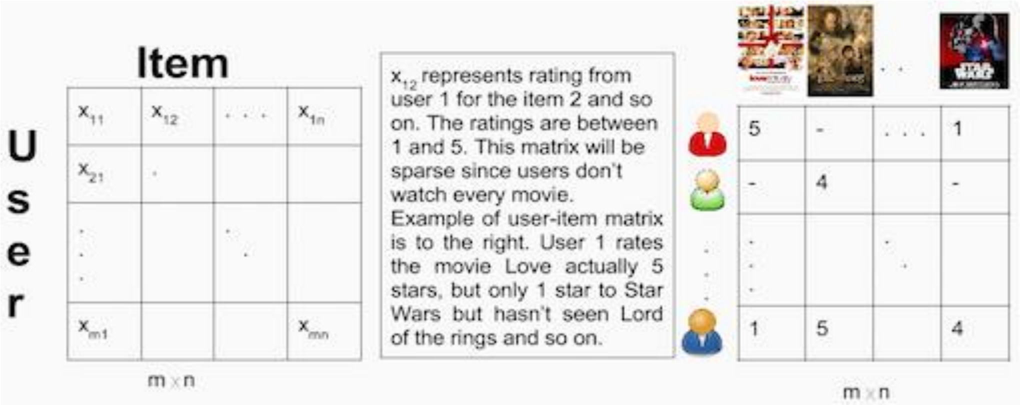
```
WRITE A DIRECTOR NAME : Martin Campbell
     movie id       movie title release date  video release date  \
1          2  GoldenEye (1995)  01-Jan-1995                 NaN
1415      1416  No Escape (1994)  01-Jan-1994                 NaN

                                              IMDb URL  unknown  Action  \
1     http://us.imdb.com/M/title-exact?GoldenEye%20(...        0       1
1415  http://us.imdb.com/M/title-exact?No%20Escape%2...       0      1

     Adventure  Animation  Childrens      ...       Film-Noir  Horror  \
1            1          0          0      ...               0       0
1415         0          0          0      ...               0       0

     Musical  Mystery  Romance  Sci-Fi  Thriller  War  Western  \
1          0        0        0       0         1    0        0
1415       0        0        0       1         0    0        0

           Director
1     Martin Campbell
1415  Martin Campbell

[2 rows x 25 columns]
```

4.3.2 MEMORY BASED COLABORATIVE FILTERING :-

Memory-based algorithms are easy to implement and produce reasonable prediction quality. This Collaborative Filtering approaches can be divided into two main sections: user-item filtering and item-item filtering. A user-item filtering takes a particular user, find users that are similar to that user based on similarity of ratings, and recommend items that those similar users liked. In contrast, item-item filtering will take an item, find users who liked that item, and find other items that those users or similar users also liked. It takes items and outputs other items as recommendations [18].

- Item-Item Collaborative Filtering: "Users who liked this item also liked …"
- User-Item Collaborative Filtering: "Users who are similar to you also liked …"

In both cases, we create a user-item matrix which we build from the entire dataset. Since we have split the data into testing and training we will need to create two 943 ×× 1682 matrices. The training matrix contains 75% of the ratings and the testing matrix contains 25% of the ratings. Example of user-item matrix:

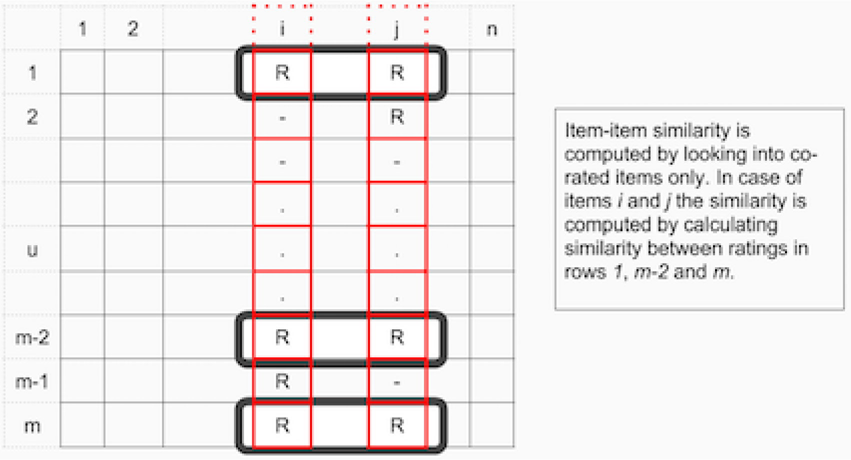


x_{12} represents rating from user 1 for the item 2 and so on. The ratings are between 1 and 5. This matrix will be sparse since users don't watch every movie. Example of user-item matrix is to the right. User 1 rates the movie Love actually 5 stars, but only 1 star to Star Wars but hasn't seen Lord of the rings and so on.

https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

After building the user-item matrix we calculate the similarity and create a similarity matrix.

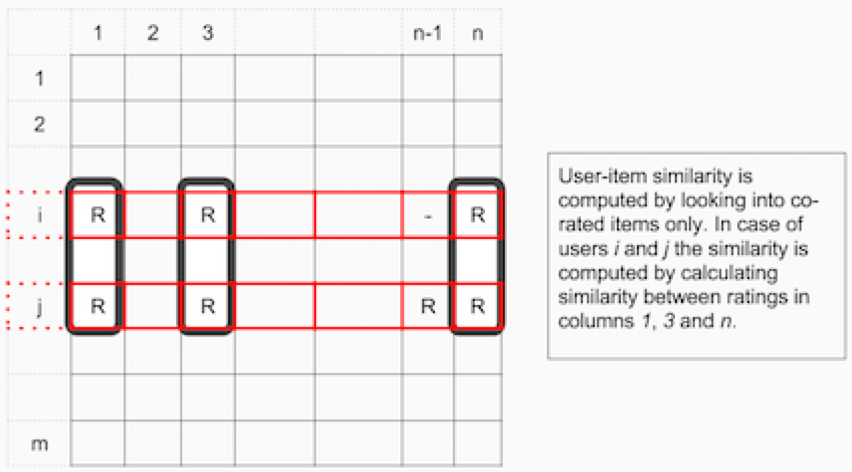
The similarity values between items in Item-Item Collaborative Filtering are measured by observing all the users who have rated both items.



Item-item similarity is computed by looking into co-rated items only. In case of items *i* and *j* the similarity is computed by calculating similarity between ratings in rows *1*, *m-2* and *m*.

https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

For User-Item Collaborative Filtering the similarity values between users are measured by observing all the items that are rated by both users.



User-item similarity is computed by looking into co-rated items only. In case of users *i* and *j* the similarity is computed by calculating similarity between ratings in columns *1*, *3* and *n*.

https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

A distance metric commonly used in recommender systems is cosine similarity, where the ratings are seen as vectors in *n*-dimensional space and the similarity is calculated based on the angle between these vectors. Cosine similarity for users *a* and *m* can be calculated using the formula below, where you take dot product of the user vector ***U**k* and the user vector ***Ua*** and divide it by multiplication of the Euclidean lengths of the vectors.

$$s_u^{cos}(u_k, u_a) = \frac{u_k \cdot u_a}{\|u_k\| \, \|u_a\|} = \frac{\sum x_{k,m} x_{a,m}}{\sqrt{\sum x_{k,m}^2 \sum x_{a,m}^2}}$$

To calculate similarity between items ***m*** and ***b*** you use the formula:

$$s_u^{cos}(i_m, i_b) = \frac{i_m \cdot i_b}{\|i_m\| \, \|i_b\|} = \frac{\sum x_{a,m} x_{a,b}}{\sqrt{\sum x_{a,m}^2 \sum x_{a,b}^2}}$$

Now we can make a prediction by applying following formula for user-based Collaborative Filtering:

$$\hat{x}_{k,m} = \bar{x}_k + \frac{\sum_{u_a} sim_u(u_k, u_a)(x_{a,m} - \bar{x}_{u_a})}{\sum_{u_a} |sim_u(u_k, u_a)|}$$

We can look at the similarity between users ***k*** and ***a*** as weights that are multiplied by the ratings of a similar user ***a*** (corrected for the average rating of that user). We need to normalize it so that the ratings stay between 1 and 5 and, as a final step, sum the average ratings for the user that we are trying to predict [18].

The idea here is that some users may tend always to give high or low ratings to all movies. The relative difference in the ratings that these users give is more important than the absolute values. To give an example: suppose, user ***k*** gives 4 stars to his favourite movies and 3 stars to all other good movies. Suppose now that another user ***t*** rates movies that he/she likes with 5 stars, and the movies he/she fell asleep over with 3 stars. These two users could have a very similar taste but treat the rating system differently.

When making a prediction for item-based CF we don't need to correct for users average rating since query user itself is used to do predictions.

$$\hat{x}_{k,m} = \frac{\sum\limits_{i_b} sim_i(i_m, i_b)(x_{k,b})}{\sum\limits_{i_b} |sim_i(i_m, i_b)|}$$

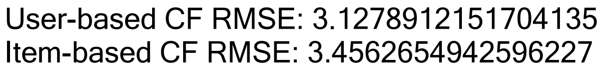
The most popular metric used to evaluate accuracy of predicted ratings is *Root Mean Squared Error (RMSE)*. [18]

$$RMSE = \sqrt{\frac{1}{N}\sum(x_i - \hat{x}_i)^2}$$

WORK FLOW

i. Read u.data file
ii. Convert it into dataframe named df
iii. Split the dataset into training and test dataset where percentage of test example (test_size) is 0.25
iv. Create two user-item matrices one for training another for testing
v. Calculate the cosine similarity of the train data (user_similarity, item_similarity)
vi. Make prediction for item-based CF using formulae(user_prediction, item_prediction)
vii. Evaluate accuracy of user-based CF and item-based CF using MSE function

OUTPUT



```
User-based CF RMSE: 3.1278912151704135
Item-based CF RMSE: 3.4562654942596227
```

4.3.3 MODEL BASED COLABORATIVE FILTERING

Model-based Collaborative Filtering is based on matrix factorization (MF) which has received greater exposure, mainly as an unsupervised learning method for latent variable decomposition and dimensionality reduction. Matrix factorization is widely used for recommender systems where it can deal better with scalability and sparsity than Memory-based Collaborative Filtering. The goal of MF is to learn the latent preferences of users and the latent attributes of items from known ratings (learn features that describe the characteristics of ratings) to then predict the unknown ratings through the dot product of the latent features of users and items. When you have a very sparse matrix, with a lot of dimensions, by doing matrix factorization you can restructure the user-item matrix into low-rank structure, and you can represent the matrix by the multiplication of two low-rank matrices, where the rows contain the latent vector. You fit this matrix to approximate your original matrix, as closely as possible, by multiplying the low-rank matrices together, which fills in the entries missing in the original matrix [9].

A well-known matrix factorization method is Singular value decomposition (SVD). Collaborative Filtering can be formulated by approximating a matrix **X** by using singular value decomposition. The general equation can be expressed as follows:

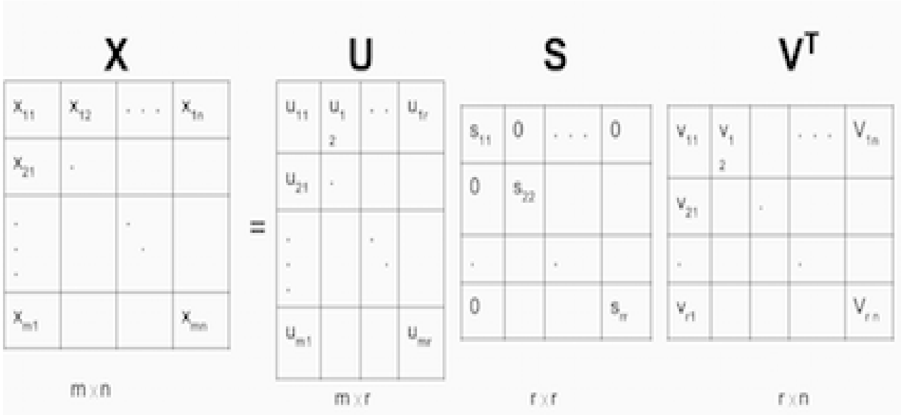
X=**U**x**S**x**VT**

Given an **m**x**n** matrix **X**:

- **U** is an **m**x**r** orthogonal matrix
- **S** is an **r**x**r** diagonal matrix with non-negative real numbers on the diagonal
- **VT** is an **r**x**n** orthogonal matrix

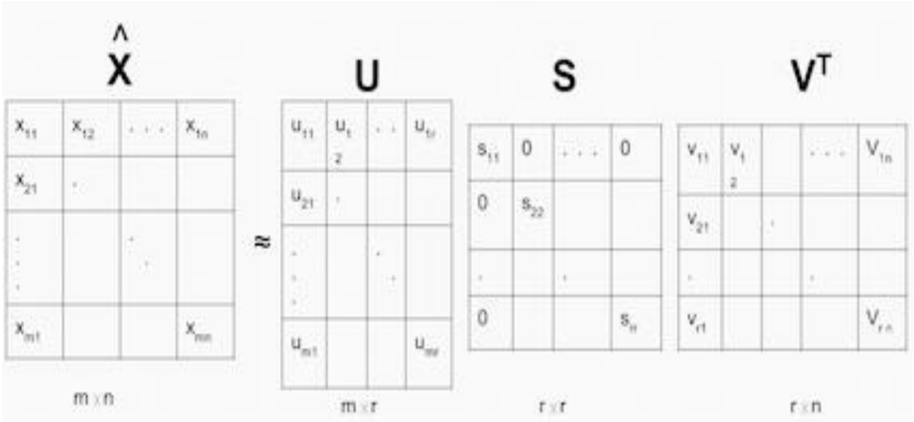
Elements on the diagnoal in **S** are known as singular values of **X**.

Matrix **X** can be factorized to **U**, **S** and **V**. The **U** matrix represents the feature vectors corresponding to the users in the hidden feature space and the **V** matrix represents the feature vectors corresponding to the items in the hidden feature space.



https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

Now we can make a prediction by taking dot product of **U**, **S** and **VT**.



https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html

Just as its name suggest matrix factorization is to, obviously, factorize a matrix, i.e. to find out two (or more) matrices such that when you multiply them you will get back the original matrix. matrix factorization can be used to discover latent features underlying the interactions between two different kinds of entities. (Of course, you can consider more than two kinds of entities and you will be dealing with tensor factorization, which would be more complicated.) And one obvious application is to predict ratings in collaborative filtering.

In a recommendation system such as Netflix or MovieLens , there is a group of users and a set of items (movies for the above two systems). Given that each users have rated some items in the system, we would like to predict how the users would rate the items that they have not yet rated, such that we can make recommendations to the users. In this case, all the information we have about the existing ratings can be represented in a matrix. Assume now we have 5 users and 10 items, and ratings are integers ranging from 1 to 5, the matrix may look something like this (a hyphen means that the user has not yet rated the movie):

	D1	D2	D3	D4
U1	5	3	-	1
U2	4	-	-	1
U3	1	1	-	5
U4	1	-	-	4
U5	-	1	5	4

Hence, the task of predicting the missing ratings can be considered as filling in the blanks (the hyphens in the matrix) such that the values would be consistent with the existing ratings in the matrix.

The intuition behind using matrix factorization to solve this problem is that there should be some latent features that determine how a user rates an item. For example, two users would give high ratings to a certain movie if they both like the actors/actresses of the movie, or if the movie is an action movie, which is a genre preferred by both users. Hence, if we can discover these latent features, we should be able to predict a rating with respect to a certain user and a certain item, because the features associated with the user should match with the features associated with the item.

In trying to discover the different features, we also make the assumption that the number of features would be smaller than the number of users and the number of items. It should not be difficult to understand this assumption because clearly it would not be reasonable to assume that each user is associated with a unique feature (although this is not impossible). And anyway if this is the case there would be no point in making recommendations, because each of these users would not be interested in the items rated by other users. Similarly, the same argument applies to the items [9].

➢ MATHEMATICS OF MATRIX FACTORIZATION

Having discussed the intuition behind matrix factorization, we can now go on to work on the mathematics. Firstly, we have a set U of users, and a set D of items. Let **R** of size **|U|**x**|D|** be the matrix that contains all the ratings that the users have assigned to the items. Also, we assume that we would like to discover K latent features. Our task, then, is to find two matrices **P** (a **|U|xK** matrix) and **Q** (a **|D|xK** matrix) such that their product approximates **R**

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q}^T = \hat{\mathbf{R}}$$

In this way, each row of $\mathbf{P}$ would represent the strength of the associations between a user and the features. Similarly, each row of $\mathbf{Q}$ would represent the strength of the associations between an item and the features. To get the prediction of a rating of an item **$\mathbf{d_j}$** by **$\mathbf{U_i}$** , we can calculate the dot product of the two vectors corresponding to **$\mathbf{U_i}$** and **$\mathbf{d_j}$** :

$$\hat{r}_{ij} = p_i^T q_j = \sum_{k=1}^{k} p_{ik} q_{kj}$$

Now, we have to find a way to obtain $\mathbf{P}$ and $\mathbf{Q}$. One way to approach this problem is the first initialize the two matrices with some values, calculate how `different' their product is to $\mathbf{M}$, and then try to minimize this difference iteratively. Such a method is called gradient descent, aiming at finding a local minimum of the difference [9].

The difference here, usually called the error between the estimated rating and the real rating, can be calculated by the following equation for each user-item pair:

$$e_{ij}^2 = (r_{ij} - \hat{r}_{ij})^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik} q_{kj})^2$$

Here we consider the squared error because the estimated rating can be either higher or lower than the real rating.

To minimize the error, we have to know in which direction we have to modify the values of $\mathbf{p}_{ik}$ and $\mathbf{q}_{kj}$. In other words, we need to know the gradient at the current values, and therefore we differentiate the above equation with respect to these two variables separately:

$$\frac{\partial}{\partial p_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(q_{kj}) = -2e_{ij}q_{kj}$$
$$\frac{\partial}{\partial q_{ik}} e_{ij}^2 = -2(r_{ij} - \hat{r}_{ij})(p_{ik}) = -2e_{ij}p_{ik}$$

Having obtained the gradient, we can now formulate the update rules for both $\mathbf{p}_{ik}$ and $\mathbf{q}_{kj}$:

$$p'_{ik} = p_{ik} + \alpha\frac{\partial}{\partial p_{ik}} e_{ij}^2 = p_{ik} + 2\alpha e_{ij}q_{kj}$$
$$q'_{kj} = q_{kj} + \alpha\frac{\partial}{\partial q_{kj}} e_{ij}^2 = q_{kj} + 2\alpha e_{ij}p_{ik}$$

Here, $\boldsymbol{\alpha}$ is a constant whose value determines the rate of approaching the minimum. Usually we will choose a small value for $\boldsymbol{\alpha}$, say 0.0002. This is because if we make too large a step towards the minimum we may run into the risk of missing the minimum and end up oscillating around the minimum.

we are not really trying to come up with **P** and **Q** such that we can reproduce **R** exactly. Instead, we will only try to minimise the errors of the observed user-item pairs. In other words, if we let **T** be a set of tuples, each of which is in the form of **(u_i,d_j,r_{ij}),** such that T contains all the observed user-item pairs together with the associated ratings, we are only trying to minimise every $\mathbf{e}_{ij}$ for **(u_i,d_j,r_{ij})εT**. (In other words, **T** is our set of training data.) As for the rest of the unknowns, we will be able to determine their values once the associations between the users, items and features have been learnt.

Using the above update rules, we can then iteratively perform the operation until the error converges to its minimum. We can check the overall error as calculated using the following equation and determine when we should stop the process [9].

$$E = \sum_{(u_i,d_j,r_{ij})\in T} e_{ij} = \sum_{(u_i,d_j,r_{ij})\in T} (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2$$

- REGULARIZATION

The above algorithm is a very basic algorithm for factorizing a matrix. There are a lot of methods to make things look more complicated. A common extension to this basic algorithm is to introduce regularization to avoid overfitting. This is done by adding a parameter $\boldsymbol{\beta}$ and modify the squared error as follows:

$$e_{ij}^2 = (r_{ij} - \sum_{k=1}^{K} p_{ik}q_{kj})^2 + \frac{\beta}{2}\sum_{k=1}^{K}(||P||^2 + ||Q||^2)$$

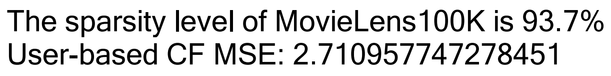
In other words, the new parameter $\boldsymbol{\beta}$ is used to control the magnitudes of the user-feature and item-feature vectors such that **P** and **Q** would give a good approximation of R without having to contain large numbers. In practice, **β** is set to some values in the range of 0.02. The new update rules for this squared error can be obtained by a procedure similar to the one described above. The new update rules are as follows [9].

$$p'_{ik} = p_{ik} + \alpha\frac{\partial}{\partial p_{ik}}e_{ij}^2 = p_{ik} + \alpha(2e_{ij}q_{kj} - \beta p_{ik})$$
$$q'_{kj} = q_{kj} + \alpha\frac{\partial}{\partial q_{kj}}e_{ij}^2 = q_{kj} + \alpha(2e_{ij}p_{ik} - \beta q_{kj})$$

WORK FLOW of finding *root meansquare error*

i. Read in the file which contains the full dataset
ii. Split the dataset into training (train_data_matrix) and test (test_data_matrix) dataset where percentage of test example (test_size) is 0.25
iii. Find sparsity level of movie lens dataset
iv. Choose k=20
v. Find SVD components (**s,u,vt**) from test matrix
vi. Make a prediction (X_predict) by taking dot product of **s,u,vt**
vii. Calculate root mean square value between X_predict and test_data_matrix

OUTPUT



```
The sparsity level of MovieLens100K is 93.7%
User-based CF MSE: 2.710957747278451
```

WORK FLOW of Matrix Factorization

i. Build movie dicitionary (movies_dict) with line no as numpy movie id, its actual movie id as the key
ii. Read data from ratings file where each line of i/p file represents one tag applied to one movie by one user, which has the following format: user Id, movie Id, tag, timestamp
iii. Return a numpy array named numpy_arr
iv. Create P an initial matrix of dimension N x K, where is n is no of users and k is hidden latent features
v. Create Q an initial matrix of dimension M x K, where M is no of movies and K is hidden latent features
vi. Initialize steps variable which is the maximum number of steps to perform the optimisation, hard coding the values
vii. Initialize alpha variable the learning rate, hard coding the values
viii. Initialize beta variable the regularization parameter, hard coding the values
ix. For each user, each item calculate the error of the element, second norm of P and Q for regularization, sum of norms
x. Compute the gradient from the error of each user, each item
xi. Compute total error
xii. Predict numpy array of users and movie ratings

WORK FLOW of Recommendation

i. Read the rating file for the missing
ii. Get the mapping between movie names, actual movie id and numpy movie id
iii. Build predicted numpy movie id from the saved predicted matrix of user and movie ratings
iv. Create a dictionary of unrated movies for each user
v. Recommend top 25 unrated movies based on their the predicted score

OUTPUT

```
Top 25 movies recommendation for the user 1
Letters from Iwo Jima (2006) with Movie rating value 9.360346877845387
Eddie Murphy Raw (1987) with Movie rating value 9.236018165660075
Dear Wendy (2005) with Movie rating value 9.203698523462286
Talking About Sex (1994) with Movie rating value 8.956160676896394
Love! Valour! Compassion! (1997) with Movie rating value 8.947410581308626
Brave (2012) with Movie rating value 8.934428106940143
Wild at Heart (1990) with Movie rating value 8.929597840739088
Taking Chance (2009) with Movie rating value 8.865345347150067
Wonderland (1999) with Movie rating value 8.858284085776976
Goldfinger (1964) with Movie rating value 8.85507043606452
Advantageous (2015) with Movie rating value 8.841811638780884
Hurt Locker  The (2008) with Movie rating value 8.827438714746878
Character (Karakter) (1997) with Movie rating value 8.814686428454374
Man from Elysian Fields  The (2001) with Movie rating value 8.804616845028592
Elite Squad (Tropa de Elite) (2007) with Movie rating value 8.800233480476749
Andalusian Dog  An (Chien andalou  Un) (1929) with Movie rating value 8.782505251474689
All About the Benjamins (2002) with Movie rating value 8.773890359394672
Tortilla Soup (2001) with Movie rating value 8.758542219008639
Irma la Douce (1963) with Movie rating value 8.745150922721505
Lion in Winter  The (1968) with Movie rating value 8.73719699262219
Mystery Science Theater 3000: The Movie (1996) with Movie rating value
8.730271624625018
Buried (2010) with Movie rating value 8.710702189343554
Ride the High Country (1962) with Movie rating value 8.709931728714334
Ice Harvest  The (2005) with Movie rating value 8.700350486013615
Musketeer  The (2001) with Movie rating value 8.666684616297236


Top 25 movies recommendation for the user 2
Star Maker  The (Uomo delle stelle  L') (1995) with Movie rating value 10.13880104020052
Haunting  The (1999) with Movie rating value 10.09639976129478
Dersu Uzala (1975) with Movie rating value 9.986568480207652
Gran Torino (2008) with Movie rating value 9.633308061739772
Dot the I (2003) with Movie rating value 9.593458722493194
Love! Valour! Compassion! (1997) with Movie rating value 9.578484363839507
Repulsion (1965) with Movie rating value 9.525752185140883
No Country for Old Men (2007) with Movie rating value 9.489700718703212
Steam: The Turkish Bath (Hamam) (1997) with Movie rating value 9.464359147721858
Insurgent (2015) with Movie rating value 9.455806020643394
Great Santini  The (1979) with Movie rating value 9.431101171056332
Bloodsport 2 (a.k.a. Bloodsport II: The Next Kumite) (1996) with Movie rating value
9.427678213008964
Dear Wendy (2005) with Movie rating value 9.322087684615997
```

```
Factotum (2005) with Movie rating value 9.312970454745962
To Catch a Thief (1955) with Movie rating value 9.25597696293851
Collapse (2009) with Movie rating value 9.20992322565514
Darkness (2002) with Movie rating value 9.177683020889742
Sword in the Stone  The (1963) with Movie rating value 9.16489314214196
Upstream Color (2013) with Movie rating value 9.136938290335344
Taking Chance (2009) with Movie rating value 9.113214062168026
Green Hornet  The (2011) with Movie rating value 9.107034466975747
Herbie Rides Again (1974) with Movie rating value 9.100971983950002
Dog Day Afternoon (1975) with Movie rating value 9.094729114359401
Midnight Cowboy (1969) with Movie rating value 9.088105650764051
Shower (Xizao) (1999) with Movie rating value 9.081735532919996


Top 25 movies recommendation for the user 3
Talking About Sex (1994) with Movie rating value 10.204998351509273
Suriyothai (a.k.a. Legend of Suriyothai  The) (2001) with Movie rating value
9.947882319852024
Everyone Says I Love You (1996) with Movie rating value 9.865275533587132
Can't Buy Me Love (1987) with Movie rating value 9.856880582067522
Safe Conduct (Laissez-Passer) (2002) with Movie rating value 9.784257297830154
Death in Brunswick (1991) with Movie rating value 9.784003891609235
Poison Ivy II (1996) with Movie rating value 9.72720674424106
Low Life (1994) with Movie rating value 9.721290686061655
Jules and Jim (Jules et Jim) (1961) with Movie rating value 9.67255094340262
Making Plans for Lena (Non ma fille  tu n'iras pas danser) (2009) with Movie rating value
9.617833833089582
Forbidden Planet (1956) with Movie rating value 9.59833795485311
Lion in Winter  The (1968) with Movie rating value 9.490113114953642
Vertigo (1958) with Movie rating value 9.489959791757553
Eddie Murphy Raw (1987) with Movie rating value 9.478330514789848
Wild at Heart (1990) with Movie rating value 9.462678235822004
Trip to the Moon  A (Voyage dans la lune  Le) (1902) with Movie rating value
9.413287809691331
Sentinel  The (2006) with Movie rating value 9.39122760646115
Great Santini  The (1979) with Movie rating value 9.357351563700892
Evita (1996) with Movie rating value 9.332073442188022
Letters from Iwo Jima (2006) with Movie rating value 9.326944306809107
Alice in Wonderland (1951) with Movie rating value 9.321912754319111
No Country for Old Men (2007) with Movie rating value 9.318230556471313
Joe Kidd (1972) with Movie rating value 9.311247251165236
Wonderland (1999) with Movie rating value 9.288325162517323
Buried (2010) with Movie rating value 9.280888142401098
```

CHAPTER 5

CONCLUDING REMARKS

Here we traversed through the process of making a basic recommendation engine in python. We stated by understanding the fundamentals of recommendations. Then we went on to load the Movie lens data set for the purpose of experimentation.

Subsequently we made a first model as a simple popularity model in which the most popular movies are recommended for users. They can also find mean rating of a particular movie or search for top rated movies of particular genre or search for the movies by director name. But this lacked personalization. So we made another model based on collaborative filtering and content based collaborative filtering.

We found that the root means square error is less in model based filtering **(2.71)** than both user based collaborative filtering **(3.12)** and item based collaborative filtering **(3.45)**. The important aspect is that the Collaborative Filtering model uses data (user_id, movie_id, rating) to learn the latent features. If there is little amount of data available model-based CF model will predict poorly, since it will be more difficult to learn the latent features.

Models that use both ratings and content features are called Hybrid Recommender Systems where both Collaborative Filtering and Content-based Models are combined. Hybrid recommender systems usually show higher accuracy than Collaborative Filtering or Content-based Models on their own: they are capable to address the cold-start problem better since if there is no ratings for a user or an item, one could use the metadata from the user or item to make a prediction.

We would like to propose another type of recommendation algorithm where factors like ratings, type of movie watched, age, occupation can be used to group users into "clusters" with similar viewing habits. A customer can belong to multiple clusters. Based on the cluster, we can then identify the movie characteristics that would be most appealing to the user. And we can recommend the user the top rated movie within that cluster.

REFERENCE

1. Gaurav Arora, Ashish Kumar, Gitanjali Sanjay Devre, Prof. Amit Ghumare , (2014) , MOVIE RECOMMENDATION SYSTEM BASED ON USERS' SIMILARITY , 4(3) , 765-766
2. Richhi F, Rokach L, Shapira B, Recommender Systems Handbook, DOI 10.1007/978-0-387-85820-3_1, © Springer Science+Business Media, LLC 2011, http://www.inf.unibz.it/~ricci/papers/intro-rec-sys-handbook.pdf
3. Manoj Kumar, D.K.Yadav, Ankur Sing, Vijay Kr. Gupta, 920150, International Journal of Computer Applications: A Movie Recommender System: MOVREC, 123(3), 7-10
4. Bhumika Bhatt, Prof. Premal J Patel, Prof. Hetal Gaudani, (2014), A Review Paper on Machine Learning Based Recommendation System, 2(4), 3955-3956
5. http://files.grouplens.org/datasets/movielens/ml-100k.zip
6. https://www.sciencedirect.com/science/article/pii/S1110866515000341
7. http://beyondvalence.blogspot.in/2014/09/python-and-pandas-part-2-movie-ratings.html
8. https://www.datacamp.com/community/tutorials/recommender-systems-python
9. http://www.quuxlabs.com/blog/2010/09/matrix-factorization-a-simple-tutorial-and-implementation-in-python/
10. https://blog.dominodatalab.com/recommender-systems-collaborative-filtering/
11. http://dataconomy.com/2015/03/an-introduction-to-recommendation-engines/
12. https://www.upwork.com/hiring/data/how-collaborative-filtering-works/
13. https://www.upwork.com/hiring/data/what-is-content-based-filtering/
14. http://www.cs.carleton.edu/cs_comps/0607/recommend/recommender/memorybased.html
15. https://acodeforthought.wordpress.com/2016/12/26/building-a-simple-recommender-system-with-movie-lens-data-set/
16. https://www.analyticsvidhya.com/blog/2016/06/quick-guide-build-recommendation-engine-python/
17. http://enhancedatascience.com/2017/04/22/building-recommender-scratch/
18. https://cambridgespark.com/content/tutorials/implementing-your-own-recommender-systems-in-Python/index.html
19. Arpita Jain, Santosh K. Viswakarma, International Journal of Computer Applications, (2017), 169(6)