Recommender Systems for E-commerce

A Project report Submitted to

Jawaharlal Nehru Technological University Hyderabad

In partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

16881A05N8	**Rishika Reddy Kota**
16881A05K6	**Gollamudi S V N Yashaswi**
17885A0529	**Dudam Manish**

Under the Guidance of

Ms S.Laxmi Sunaina

Assistant professor

Department of Computer Science and Engineering

VARDHAMAN COLLEGE OF ENGINEERING

(AUTONOMOUS)

(Affiliated to JNTUH, Hyderabad)

Shamshabad – 501 218, Hyderabad

2019 – 2020

CERTIFICATE

*This is to certify that the Mini Project report titled "**Recommender Systems for E-commerce**" is bonafide work done by*

16881A05N8 Rishika Reddy Kota
16881A05K6 Gollamudi S V N Yashaswi
17885A0529 Dudam Manish

*In the department of Computer Science and Engineering, **Vardhaman College of Engineering (AUTONOMOUS)**, Shamshabad is submitted to Jawaharlal Nehru Technological University Hyderabad, Hyderabad in partial fulfillment of the requirements for the award of B. Tech degree in Computer Science and Engineering during 2019-20.*

Verified by Coordinator

Signature of the Guide

Signature of the External Examiner

Signature of the Head of the Department

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We avail this opportunity to express our deep sense of gratitude and hearty thanks to management of Vardhaman College of Engineering, for providing congenial atmosphere and encouragement.

Our sincere thanks to **Dr Mallikharjuna Babu, Director & CEO**, for his encouragement.

We show gratitude to **Dr. S Sai Satyanarayana Reddy**, **Principal** for having provided all the facilities and support.

We would like to thank **Dr. Rajanikanth Aluvalu, Head of the Department,Computer Science and Engineering** or his expert guidance and encouragement at various levels of our Project.

We are thankful to our guide **Ms S.Laxmi Sunaina, Assistant Professor** for his sustained inspiring Guidance and cooperation throughout the process of this project. His wise counsel and suggestions were invaluable.

We express our deep sense of gratitude and thanks to all the **Teaching** and **Non-Teaching Staff** of our college who stood with us during the project and helped us to make it a successful venture.

We place highest regards to our **Parents**, our **Friends** and **Well-wishers** who helped a lot in making the report of this project.

16881A05N8 Rishika Reddy Kota
16881A05K6 Gollamudi S V N Yashaswi
17885A0529 Dudam Manish

DECLARATION BY THE CANDIDATE

We the undersigned solemnly declare that the project report RECOMMENDER SYSTEMS FOR E-COMMERCE is based on our work carried out during the course of our study under the supervision of Ms.S.Laxmi Sunaina, Assistant Professor. We assert the statements made and conclusions drawn are an outcome of our project work. We further certify that

I.The work contained in the report is original and has been done by team under the general supervision of our supervisor.

II.The work has not been submitted to any other Institution for any other degree/diploma/certificate in this university or any other University of India or abroad.

III.We have followed the guidelines provided by the college in writing the report.

IV.Whenever we have used materials (data, theoretical analysis, and text) from other sources, we have given due credit to them in the text of the report and giving their details in the references.

16881A05N8 Rishika Reddy Kota
16881A05K6 Gollamudi S V N Yashaswi
17885A0529 Dudam Manish

CONTENTS

ABSTRACT

Recommender Systems or Recommendation Systems is a subclass of information filtering systems that seeks to predict a “rating” or “preference” that a user would give to an item.
Recommender Systems are used in variety of areas, mostly used in e-commerce sites to filter the products according to the user’s interest.

Internet is speeding up and modifying the manner in which daily tasks such as online shopping, paying bills, etc are accomplished. To stay competitive, markets need to provide different products and services to different customers with different needs. Markets need to customize customers need while providing more options.

Recommender Systems allow rapid and automated customization and personalization of e-commerce sites. They allow the sites to generate more sales by tailoring to needs of the visitors and turning them in to customers.

Information based on user demographics, item attributes, and user preferences are gathered. From this dataset useful information is extracted which is referred as Knowledge Discovery in Dataset. Using an appropriate Recommendation System and Recommendation Techniques are used to generate recommendations for users.

1. INTRODUCTION

1.1 MOTIVATION

Recommender Systems enhances search options for users using E-commerce sites. A personalized website according to the interest of the user would be a better option than a website with no recommendations of user interests. Users would be more aligned towards websites with such improvements in E-commerce.

While Business bodies handling these sites have increased their profits and expanded their business of online marketing. Users more attracting towards sites profit E-commerce with huge amounts.

1.2 PROBLEM DEFINITION

E-commerce need to adopt to Recommender Systems, plain and similar options to every user would be of no use. Users in would eventually lose interest in E-commerce sites.

There are lot problems using non-recommended sites.

1. Since these sites lot of data presentation of data to the user would be an issue.

2. User search would be complex.

3. Cyber issues would matter.

4. Users eventually lose interest and E-commerce sites would be no longer be useful.

To solve such issues Recommender Systems is the solution.

1.3 OBJECTIVES

1. To Build an Algorithm for recommender system for a website

2. To suggest users with the products of their interests or mostly viewed

1.4 LIMITATIONS

1. Similar user ratings for an item are needed.

2. Since user ratings data is large, using collaborative filtering algorithm problems like sparsity arises.

3. Using Collaborative Algorithm problems like cold start arises.

1.5 ORGANIZATION OF DOCUMENTATION

In this project documentation, we have described about the problem definition and objective of the project initially, and followed by analysis, design, implementation and testing phases. Finally, the conclusion and future scope are described.

2.LITERATURE SURVEY

INTRODUCTION

In E-commerce information technology and internet increases day to day life very rapidly. E-commerce is the website for purchase and sale goods from it. User can attract more and more attention. In recent time Ecommerce takes place in the market platform and increase rapidly. In E-commerce, personalized product also helps to the user preference and their requirements. It provides product according to understand recommendation reduces time and cost for the users.

By using a Collaborative filtering algorithm, it is evaluated that unknown rating based on known entries with lowest accumulative error.

EXISTING SYSTEM

Existing systems does not have an in-built algorithm that would give personalized and customized websites that give recommendation of user interest. Lacking a recommender system, a site. It is just a same website for every user.

DISADVANTAGES OF EXISTING SYSTEM

- More time
- More money
- More manpower
- More work
- Less work results
- No accuracy

PROPOSED SYSTEM

The proposed system, Recommender Systems using collaborative filtering in which they discuss how people collaborate and filtrate relevant items to them and that of user's choice. The Filtering process included analyses of common properties among two or more users.

Recommender systems allow rapid and automated customization and personalization of e-commerce sites. They allow the sites to generate more sales by tailoring to the needs of the visitors and turning them into consumers, up-selling extra products by bundling closely related things together, and increasing customer loyalty.

Collaborative Filtering

Collaborative filtering approach uses customer details, ratings, and reviews aggregated from all the customers to build recommendations. The strength of this approach is that it analyzes existing active customers with similar preferences and characteristics of the current customer to build the recommendations. The filtering method is achieved through a heuristic-based, a model-based method, or a hybrid model that combines characteristics from both heuristic and

model-based approaches. The heuristic based or memory-based collaborative filtering model takes in rating data, whether product was purchased or not, and duration of viewing products to calculate the recommendations. Active customers whose information is used is done by

selecting all the customers who are neighbors of the current customer using similarity measures including personal information, cosine metric, and Jaccard coefficient for binary data. Then, utilizing k-nearest neighbor classification method, prediction value is computed for each product that current customer has not viewed but the other active customers have. With the newly calculated set,

recommendation is created based on products with the highest scores. There are many different algorithms and technique that can be used in heuristic based collaborative filtering includes k-nearest neighbor algorithm, web mining algorithms, decision trees, and support vector machines. The model based collaborative filtering technique uses training data such as the active user's ratings and reviews to build a model using different data mining and machine learning algorithms. The model is then validated using the testing data and list of products and rating is predicted for them if customers have not given any rating to it yet or been exposed to it. While the heuristics-based model uses the entire database and the customers to create recommendations for the active customer, the model-based approach only relies on the active customer's information as the input. Techniques and algorithms from fields such as Bayesian model, clustering, association rules, artificial neural networks, linear regression, maximum entropy, latent semantic analysis, and Markov process can be used.

The Dataset

To experiment with recommendation algorithms, you'll need data that contains a **set of items** and a **set of users** who have reacted to some of the items.

The reaction can be **explicit** (rating on a scale of 1 to 5, likes or dislikes) or **implicit** (viewing an item, adding it to a wish list, the time spent on an article).

While working with such data, you'll mostly see it in the form of a **matrix** consisting of the reactions given by a set of users to some items from a set of items. Each row would contain the ratings given by a user, and each column would contain the ratings received by an item. A matrix with five users and five items could look like this:

	i_1	i_2	i_3	i_4	i_5
u_1	5		4	1	
u_2		3		3	
u_3		2	4	4	1
u_4	4	4	5		
u_5	2	4		5	2

The matrix shows five users who have rated some of the items on a scale of 1 to 5. For example, the first user has given a rating 4 to the third item.

In most cases, the cells in the matrix are empty, as users only rate a few items. It's highly unlikely for every user to rate or react to every item available. A matrix with mostly empty cells is called **sparse**, and the opposite to that (a mostly filled matrix) is called **dense**.

There are a lot of datasets that have been collected and made available to the public for research and benchmarking. Here's a list of high-quality data sources that you can choose from.

The best one to get started would be the MovieLens dataset collected by GroupLens Research. In particular, the MovieLens 100k dataset is a stable benchmark dataset with 100,000 ratings given by 943 users for 1682 movies, with each user having rated at least 20 movies.

This dataset consists of many files that contain information about the movies, the users, and the ratings given by users to the movies they have watched. The ones that are of interest are the following:

- **u.itefm:** the list of movies
- **u.data:** the list of ratings given by users

The file u.data that contains the ratings is a tab separated list of user ID, item ID, rating, and timestamp. The first few lines of the file look like this:

user_id	item_id	rating	timestamp
196	242	3	881250949
186	302	3	891717742
22	377	1	878887116
244	51	2	880606923
166	346	1	886397596

As shown above, the file tells what rating a user gave to a particular movie. This file contains 100,000 such ratings, which will be used to predict the ratings of the movies not seen by the users.

Steps in involved in Collaborative Filtering

To build a system that can automatically recommend items to users based on the preferences of other users, the first step is to find similar users or items. The second step is to predict the ratings of the items that are not yet rated by a user. So, you will need the answers to these questions:

- How do you determine which users or items are similar to one another?
- Given that you know which users are similar, how do you determine the rating that a user would give to an item based on the ratings of similar users?
- How do you measure the accuracy of the ratings you calculate?

The first two questions don't have single answers. Collaborative filtering is a family of algorithms where there are multiple ways to find similar users or items and multiple ways to calculate rating based on ratings of similar users. Depending on the choices you make, you end up with a type of collaborative filtering approach. You'll get to see the various approaches to find similarity and predict ratings in this article.

One important thing to keep in mind is that in an approach based purely on collaborative filtering, the similarity is not calculated using factors like the age of users, genre of the movie, or any other data about users or items. It is calculated only on the basis of the rating (explicit or implicit) a user gives to an item. For example, two users can be considered similar if they give the same ratings to ten movies despite there being a big difference in their age.

The third question for how to measure the accuracy of your predictions also has multiple answers, which include error calculation techniques that can be used in many places and not just recommenders based on collaborative filtering.

One of the approaches to measure the accuracy of your result is the Root Mean Square Error (RMSE), in which you predict ratings for a test dataset of user-item pairs whose rating values are already known. The difference between the known value and the predicted value would be the error. Square all the error values for the test set, find the average (or mean), and then take the square root of that average to get the RMSE.

Another metric to measure the accuracy is Mean Absolute Error (MAE), in which you find the magnitude of error by finding its absolute value and then taking the average of all error values.

You don’t need to worry about the details of RMSE or MAE at this point as they are readily available as part of various packages in Python, and you will see them later in the article.

Now let’s look at the different types of algorithms in the family of collaborative filtering.

How to Find similar users on the basis of ratings

The first category includes algorithms that are memory based, in which statistical techniques are applied to the entire dataset to calculate the predictions.

To find the rating **R** that a user **U** would give to an item **I**, the approach includes:

- Finding users similar to **U** who have rated the item **I**
- Calculating the rating **R** based the ratings of users found in the previous step

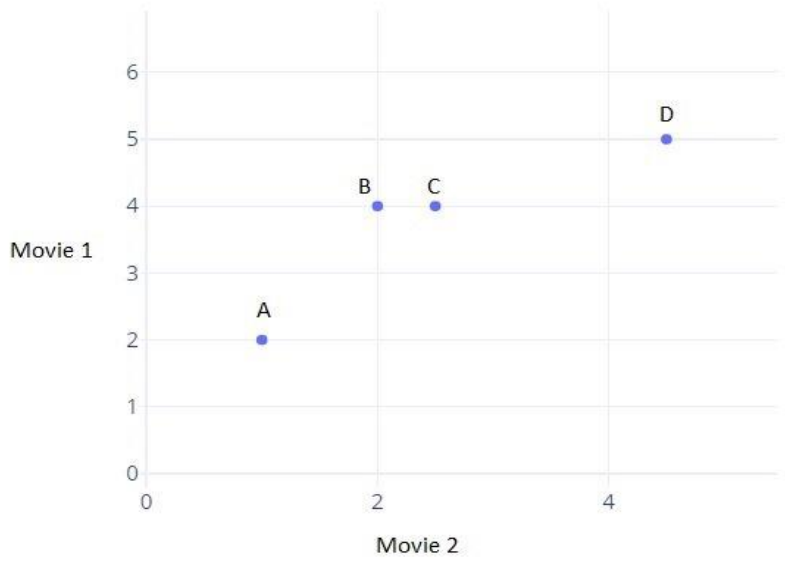
You’ll see each of them in detail in the following sections.

To understand the concept of similarity, let’s create a simple dataset first.

The data includes four users **A**, **B**, **C**, and **D**, who have rated two movies. The ratings are stored in lists, and each list contains two numbers indicating the rating of each movie:

- Ratings by **A** are [1.0, 2.0].
- Ratings by **B** are [2.0, 4.0].
- Ratings by **C** are [2.5, 4.0].
- Ratings by **D** are [4.5, 5.0].

To start off with a visual clue, plot the ratings of two movies given by the users on a graph and look for a pattern. The graph looks like this:



In the graph above, each point represents a user and is plotted against the ratings they gave to two movies.

Looking at the distance between the points seems to be a good way to estimate similarity, right? You can find the distance using the formula for Euclidean distance between two points. You can use the function available in scipy as shown in the following program:

PYTHON SNIPPET:

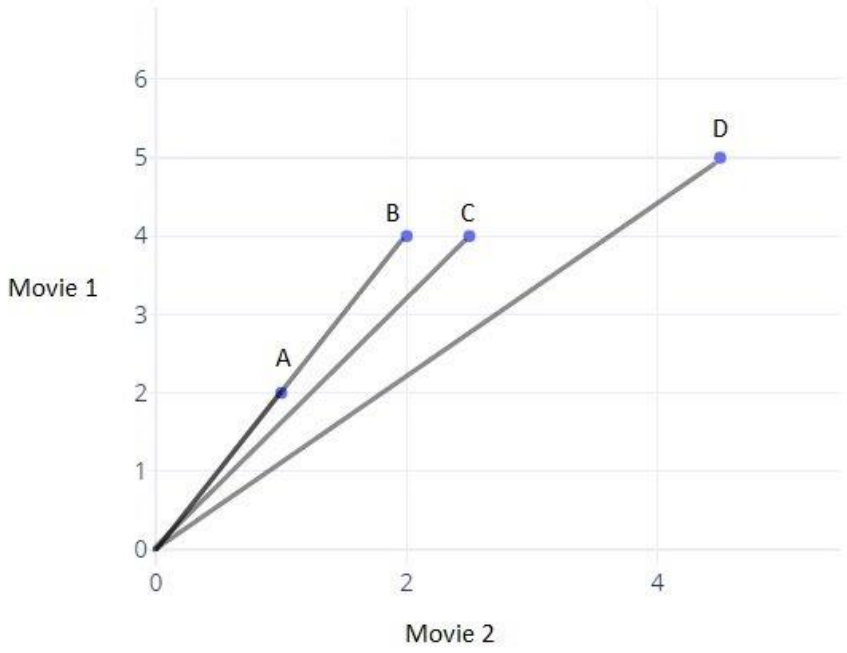
```
>>> from scipy import spatial

>>> a = [1, 2]
>>> b = [2, 4]
>>> c = [2.5, 4]
>>> d = [4.5, 5]

>>> spatial.distance.euclidean(c, a)
2.5
>>> spatial.distance.euclidean(c, b)
0.5
>>> spatial.distance.euclidean(c, d)
2.23606797749979
```

As shown above, you can use scipy.spatial.distance.euclidean to calculate the distance between two points. Using it to calculate the distance between the ratings of **A**, **B**, and **D** to that of **C** shows us that in terms of distance, the ratings of **C** are closest to those of **B**.

You can see that user **C** is closest to **B** even by looking at the graph. But out of **A** and **D** only, who is **C** closer to?

You could say **C** is closer to **D** in terms of distance. But looking at the rankings, it would seem that the choices of **C** would align with that of **A** more than **D** because both **A** and **C** like the second movie almost twice as much as they like the first movie, but **D** likes both of the movies equally.

So, what can you use to identify such patterns that Euclidean distance cannot? Can the angle between the lines joining the points to the origin be used to make a decision? You can take a look at the angle between the lines joining the origin of the graph to the respective points as shown:



The graph shows four lines joining each point to the origin. The lines for **A** and **B** are coincident, making the angle between them zero.

You can consider that, if the angle between the lines is increased, then the similarity decreases, and if the angle is zero, then the users are very similar.

To calculate similarity using angle, you need a function that returns a **higher similarity** or **smaller distance** for a lower angle and a **lower similarity** or **larger distance** for a higher angle. The cosine of an angle is a function that decreases from 1 to -1 as the angle increases from 0 to 180.

You can use the cosine of the angle to find the similarity between two users. The higher the angle, the lower will be the cosine and thus, the lower will be the similarity of the users. You can also inverse the value of the cosine of the angle to get the cosine distance between the users by subtracting it from 1.

scipy has a function that calculates the **cosine distance** of vectors. It returns a higher value for higher angle:

```
>>> from scipy import spatial
>>> a = [1, 2]
>>> b = [2, 4]
>>> c = [2.5, 4]
>>> d = [4.5, 5]

>>> spatial.distance.cosine(c,a)
0.004504527406047898

>>> spatial.distance.cosine(c,b)
0.004504527406047898

>>> spatial.distance.cosine(c,d)
0.015137225946083022

>>> spatial.distance.cosine(a,b)
0.0
```

The lower angle between the vectors of **C** and **A** gives a lower cosine distance value. If you want to rank user similarities in this way, use cosine distance.

Notice that users **A** and **B** are considered absolutely similar in the cosine similarity metric despite having different ratings. This is actually a common occurrence in the real world, and the users like the user **A** are what you can call **tough raters**. An example would be a movie critic who always gives out ratings lower than the average, but the rankings of the items in their list would be similar to the **Average raters** like **B**.

To factor in such individual user preferences, you will need to bring all users to the same level by removing their biases. You can do this by subtracting the average rating given by that user to all items from each item rated by that user. Here's what it would look like:

- For user **A**, the rating vector [1, 2] has the average 1.5. Subtracting 1.5 from every rating would give you the vector [-0.5, 0.5].
- For user **B**, the rating vector [2, 4] has the average 3. Subtracting 3 from every rating would give you the vector [-1, 1].

By doing this, you have changed the value of the average rating given by every user to 0. Try doing the same for users **C** and **D**, and you'll see that the ratings are now adjusted to give an average of 0 for all users, which brings them all to the same level and removes their biases.

The cosine of the angle between the adjusted vectors is called **centered cosine**. This approach is normally used when there are a lot of missing values in the vectors, and you need to place a common value to fill up the missing values.

Filling up the missing values in the ratings matrix with a random value could result in inaccuracies. A good choice to fill the missing values could be the average rating of each user, but the original averages of

user **A** and **B** are 1.5 and 3 respectively, and filling up all the empty values of **A** with 1.5 and those of **B** with 3 would make them dissimilar users.

But after adjusting the values, the **centered** average of both users is 0, which allows you to capture the idea of the item being above or below average more accurately for both users with all missing values in both user's vectors having the same value 0.

Euclidean distance and cosine similarity are some of the approaches that you can use to find users similar to one another and even items similar to one another. (The function used above calculates cosine distance. To calculate cosine similarity, subtract the distance from 1.)

How to Calculate the Ratings

After you have determined a list of users similar to a user **U**, you need to calculate the rating **R** that **U** would give to a certain item **I**. Again, just like similarity, you can do this in multiple ways.

You can predict that a user's rating **R** for an item **I** will be close to the average of the ratings given to **I** by the top 5 or top 10 users most similar to **U**. The mathematical formula for the average rating given by *n* users would look like this:

$$R_U = (\sum_{u=1}^{n} R_u)/n$$

This formula shows that the average rating given by the *n* similar users is equal to the sum of the ratings given by them divided by the number of similar users, which is *n*.

There will be situations where the *n* similar users that you found are not equally similar to the target user **U**. The top 3 of them might be very similar, and the rest might not be as similar to **U** as the top 3. In that case, you could consider an approach where the rating of the most similar user matters more than the second most similar user and so on. The weighted average can help us achieve that.

In the weighted average approach, you multiply each rating by a similarity factor(which tells how similar the users are). By multiplying with the similarity factor, you add weights to the ratings. The heavier the weight, the more the rating would matter.

The similarity factor, which would act as weights, should be the inverse of the distance discussed above because less distance implies higher similarity. For example, you can subtract the cosine distance from 1 to get cosine similarity.

With the similarity factor **S** for each user similar to the target user **U**, you can calculate the weighted average using this formula:

$$R_U = (\sum_{u=1}^{n} R_u * S_u)/(\sum_{u=1}^{n} S_u)$$

In the above formula, every rating is multiplied by the similarity factor of the user who gave the rating. The final predicted rating by user **U** will be equal to the sum of the weighted ratings divided by the sum of the weights.

With a weighted average, you give more consideration to the ratings of similar users in order of their similarity.

Now, you know how to find similar users and how to calculate ratings based on their ratings. There's also a variation of collaborative filtering where you predict

ratings by finding items similar to each other instead of users and calculating the ratings. You'll read about this variation in the next section.

With a weighted average, you give more consideration to the ratings of similar users in order of their similarity.

Now, you know how to find similar users and how to calculate ratings based on their ratings. There's also a variation of collaborative filtering where you predict ratings by finding items similar to each other instead of users and calculating the ratings. You'll read about this variation in the next section.

User-Based vs Item-Based Collaborative Filtering

The technique in the examples explained above, where the rating matrix is used to find similar users based on the ratings they give, is called user-based or user-user collaborative filtering. If you use the rating matrix to find similar items based on the ratings given to them by users, then the approach is called item-based or item-item collaborative filtering.

The two approaches are mathematically quite similar, but there is a conceptual difference between the two. Here's how the two compare:

- **User-based:** For a user **U**, with a set of similar users determined based on rating vectors consisting of given item ratings, the rating for an item **I**, which hasn't been rated, is found by picking out N users from the similarity list who have rated the item **I** and calculating the rating based on these N ratings.
- **Item-based:** For an item **I**, with a set of similar items determined based on rating vectors consisting of received user ratings, the rating by a user **U**, who hasn't rated it, is found by picking out N items from the similarity list that have been rated by **U** and calculating the rating based on these N ratings.

Item-based collaborative filtering was developed by Amazon. In a system where there are more users than items, item-based filtering is faster and more stable than user-based. It is effective because usually, the average rating received by an item doesn't change as quickly as the average rating given by a user to different items. It's also known to perform better than the user-based approach when the ratings matrix is sparse.

Although, the item-based approach performs poorly for datasets with browsing or entertainment related items such as MovieLens, where the recommendations it gives out seem very obvious to the target users. Such datasets see better results with matrix factorization techniques, which you'll see in the next section, or with hybrid recommenders that also take into account the content of the data like the genre by using content-based filtering.

You can use the library Surprise to experiment with different recommender algorithms quickly.

Using Python to Build Recommenders

There are quite a few libraries and toolkits in Python that provide implementations of various algorithms that you can use to build a recommender. But the one that you should try out while understanding recommendation systems is Surprise.

Surprise is a Python SciKit that comes with various recommender algorithms and similarity metrics to make it easy to build and analyze recommenders.

Here's how to install it using pip:

```
$ pip install numpy
$ pip install scikit-surprise
```

Here's how to install it using conda:

```
$ conda install -c conda-forge scikit-surprise
```

The Dataset module is used to load data from files, Pandas dataframes, or even built-in datasets available for experimentation. (MovieLens 100k is one of the built-in datasets in Surprise.) To load a dataset, some of the available methods are:

- Dataset.load_builtin()
- Dataset.load_from_file()
- Dataset.load_from_df()

- The Reader class is used to parse a file containing ratings. The default format in which it accepts data is that each rating is stored in a separate line in the order user item rating. This order and the separator can be configured using parameters:
- **line_format** is a string that stores the order of the data with field names separated by a space, as in "item user rating".
- **sep** is used to specify separator between fields, such as ','.
- **rating_scale** is used to specify the rating scale. The default is (1, 5).
- **skip_lines** is used to indicate the number of lines to skip at the beginning of the file. The default is 0.

Here's a program that you can use to load data from a Pandas dataframe or the from builtin MovieLens 100k dataset:

```
# load_data.py

import pandas as pd
from surprise import Dataset
from surprise import Reader

# This is the same data that was plotted for similarity earlier
# with one new user "E" who has rated only movie 1
ratings_dict = {
    "item": [1, 2, 1, 2, 1, 2, 1, 2, 1],
    "user": ['A', 'A', 'B', 'B', 'C', 'C', 'D', 'D', 'E'],
    "rating": [1, 2, 2, 4, 2.5, 4, 4.5, 5, 3],
}

df = pd.DataFrame(ratings_dict)
reader = Reader(rating_scale=(1, 5))

# Loads Pandas dataframe
data = Dataset.load_from_df(df[["user", "item", "rating"]], reader)
# Loads the builtin Movielens-100k data
movielens = Dataset.load_builtin('ml-100k')
```

In the above program, the data is stored in a dictionary that is loaded into a Pandas data frame and then into a Dataset object from Surprise.

Algorithms Based on K-Nearest Neighbor's (k-NN)

The choice of algorithm for the recommender function depends on the technique you want to use. For the memory-based approaches discussed above, the algorithm that would fit the bill is Centered k-NN because the algorithm is very close to the centered cosine similarity formula explained above. It is available in Surprise as KNNWithMeans.

To find the similarity, you simply have to configure the function by passing a dictionary as an argument to the recommender function. The dictionary should have the required keys, such as the following:

- **name** contains the similarity metric to use. Options are cosine, msd, pearson, or Pearson baseline. The default is msd.
- **user_based** is a Boolean that tells whether the approach will be user-based or item-based. The default is True, which means the user-based approach will be used.
- **min_support** is the minimum number of common items needed between users to consider them for similarity. For the item-based approach, this corresponds to the minimum number of common users for two items.

The following program configures the KNNWithMeans function:

```
# recommender.py

from surprise import KNNWithMeans

# To use item-based cosine similarity
sim_options = {
    "name": "cosine",
    "user_based": False,  # Compute  similarities between items
}
algo = KNNWithMeans(sim_options=sim_options)
```

The recommender function in the above program is configured to use the cosine similarity and to find similar items using the item-based approach.

To try out this recommender, you need to create a Trainset from data. Trainset is built using the same data but contains more information about the data, such as the number of users and items (n_users, n_items) that are used by the algorithm. You can create it either by using the entire data or a part of the data. You can also divide the data into folds where some of the data will be used for training and some for testing.

Here's an example to find out how the user **E** would rate the movie 2:

```
>>> from load_data import data
>>> from recommender import algo

>>> trainingSet = data.build_full_trainset()

>>> algo.fit(trainingSet)
Computing the cosine similarity matrix...
Done computing similarity matrix.
<surprise.prediction_algorithms.knns.KNNWithMeans object at 0x7f04fec56898>

>>> prediction = algo.predict('E', 2)
>>> prediction.est
4.15
```

The algorithm predicted that the user **E** would rate the movie 4.15, which could be high enough to be shown as a recommendation.

You should try out the different k-NN based algorithms along with different similarity options and matrix factorization algorithms available in the Surprise library. Try them out on the MovieLens dataset to see if you can beat some benchmarks. The next section will cover how to use Surprise to check which parameters perform best for your data.

3. ANALYSIS

Hardware Requirement

Processor	:	Intel Core Duo 2.0 GHz or more
RAM	:	1 GB or More
Hard disk	:	80GB or more
Monitor	:	15” CRT, or LCD monitor
Keyboard	:	Normal or Multimedia
Mouse	:	Compatible mouse

3.2 Software Requirement

Front End	:	Python3
Back End	:	Anaconda
Operation System	:	Windows XP with server pack 2 Or Windows Vista

Software Development Process

Life Cycle Used to develop this Project

Life cycle used **----** **SDLC**

The Systems Development Life Cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project from an initial feasibility study through maintenance of the completed application. Various SDLC methodologies have been developed to guide the processes involved including the waterfall model (the original SDLC method), rapid application development (RAD), joint application development (JAD), the fountain model and the spiral model. Mostly, several models are combined into some sort of hybrid methodology.

Feasibilities study

The first step is to study whether or not it is practical to development the software or whether or not it is worth carrying on with the requirement analysis. The following questions should be answered in this step:

a) Does the software contribute to the overall objective of the organization?

b) Can the software be implemented using current technology and within given cost and schedule constraints?

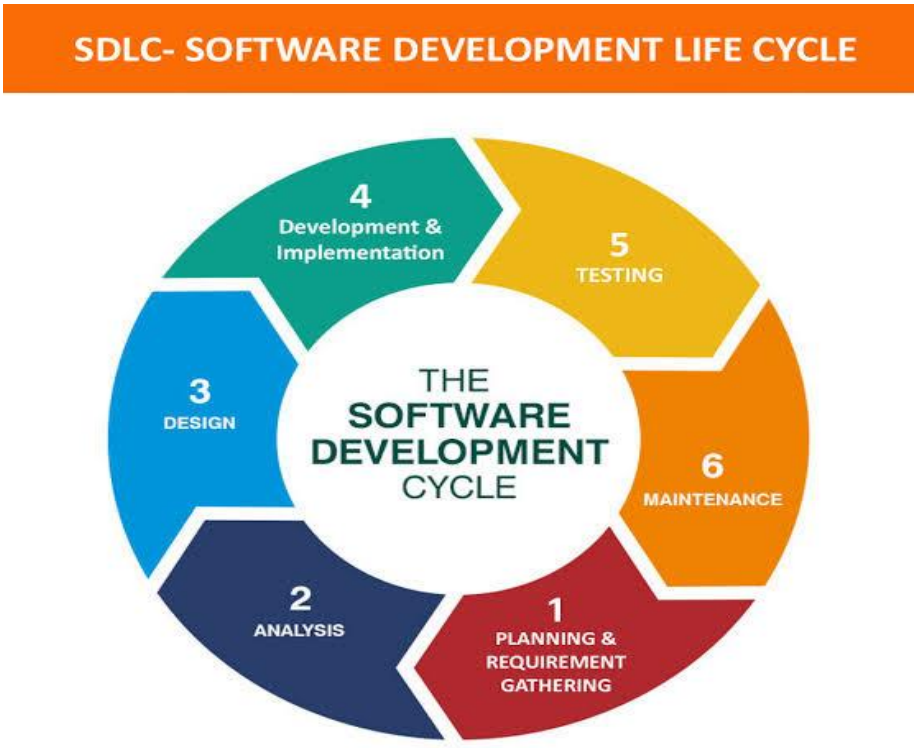
c) Can the software be integrated with other software in the application domain

A widely used model to document requirement is called use case, which describe the interaction of one or several scenarios. A use case usually consists of the following parts. A description of what the software and users except when the scenario starts.

a) A description of the normal flow of events in the scenario

b) A description of what can go wrong and how this is handled, and

c) A description of the system state when the scenario finishes.



ECONOMIC ANALYSIS

Among the most important information contained in feasibility study is Cost Benefit Analysis and assessment of the economic justification for a computer-based system project. Cost Benefit Analysis delineates costs for the project development and weighs them against tangible and intangible benefits of a system. Cost Benefits Analysis is complicated by the criteria that vary with the characteristics of the system to be developed, the relative size of the project and the expected return on investment desired as part of company's strategic plan. In addition, many benefits derived from a computer-based system are intangible (e.g. better design quality through iterative optimization, increased customer satisfaction through programmable control etc.) As this is an in-house project for the company, to be used for its own convenience and also it is not that big a project. So neither it requires a huge amount of money nor any costly tools or infrastructure need to be set up for it.

TECHNICAL ANALYSIS

During technical analysis, the technical merits of the system are studied and at the same time collecting additional information about performance, reliability, maintainability and predictability.

Technical analysis begins with an assessment of the technical viability of the proposed system.

What technologies are required to accomplished system function and performance?

How will these obtained from technical analysis form the basis for another go/no-go decision on the test system? If the technical risk is severe, if models indicate that the desired function can not be achieved, if the pieces just won't fit together smoothly-it's back to the drawing board.

SYSTEM ANALYSIS

System analysis is the process of studying the business processors and procedures, generally referred to as business systems, to see how they can operate and whether improvement is needed.

This may involve examining data movement and storage, machines and technology used in the system, programs that control the machines, people providing inputs, doing the processing and receiving the outputs.

CONSTRAINTS AND LIMITATIONS

The constraints and limitation within a system are the drawbacks that occur during the implementation of the system. These limitations and constraints can crop up in almost every system; the most important fact is to find a way to overcome these problems.

4. DESIGN

Use cases: Elicit requirement from users in meaningful chunks. Construction planning is built around delivering some use cases n each interaction basis for system testing.

Class diagrams: shows static structure of concepts, types and class. Concepts how users think about the world; type shows interfaces of software components; classes show implementation of software components.

Interaction diagrams: shows how several objects collaborate in single use case.

Package diagram: show group of classes and dependencies among them.

State diagram: show how single object behaves across many use cases.

Activity diagram: shows behavior with control structure. Can show many objects over many uses, many object in single use case, or implementations methods encourage parallel behavior, etc.

The end-product of this project is a comprehensive tool that can parse any vb.net program and extract most of the object oriented features inherent in the program such as polymorphism, inheritance, encapsulation and abstraction.

What is UML?

UML stands for Unified Modeling Language is the successor to the wave of Object Oriented Analysis and Design (OOA&D) methods that appeared in the late 80's. It most directly unifies the methods of Booch, Rumbaugh (OMT) and Jacobson. The UML is called a modeling language, not a method. Most methods consist at least in principle, of both a modeling language and a process. The Modeling language is that notation that methods used to express design.

Notations and meta-models:

The notation is the graphical stuff; it is the syntax of the modeling language. For instance, class diagram notation defines how items are concepts such as class, association, and multiplicity is represented. These are:

Class Diagram

The class diagram technique has become truly central within object-oriented methods. Virtually every method has included some variation on this technique. Class diagram is also subject to the greatest range of modeling concept. Although the basic elements are needed by everyone, advanced concepts are used less often. A class diagram describes the types of objects in the systemand the various kinds of static relationship that exist among them. There are two principal kinds of static relationship:

- Association
- Subtype

Class diagram also show the attributes and operations of a class and the constraints that apply to the way objects are connected.

Association: Association represent between instances of class. From the conceptual perspective, association represents conceptual relations between classes. Each association has two roles. Each role is a direction on the association. A role also has multiplicity, which is a indication of how many object may participate in the given relationship.

Generalization: A typical example of generalization evolves the personal and corporate customer of a business. They have differences but also many similarity. The similarities can be placed in generalization with personal customer and corporate customer sub type.

Aggregation: aggregation is the part of relationship. It is like saying a car has engine and wheels as its parts. This sounds good, but difficult thing is considering, what is the difference is aggregation and association.

Interaction: interaction diagrams are models that describes how groups of objects collaboration in some behavior.

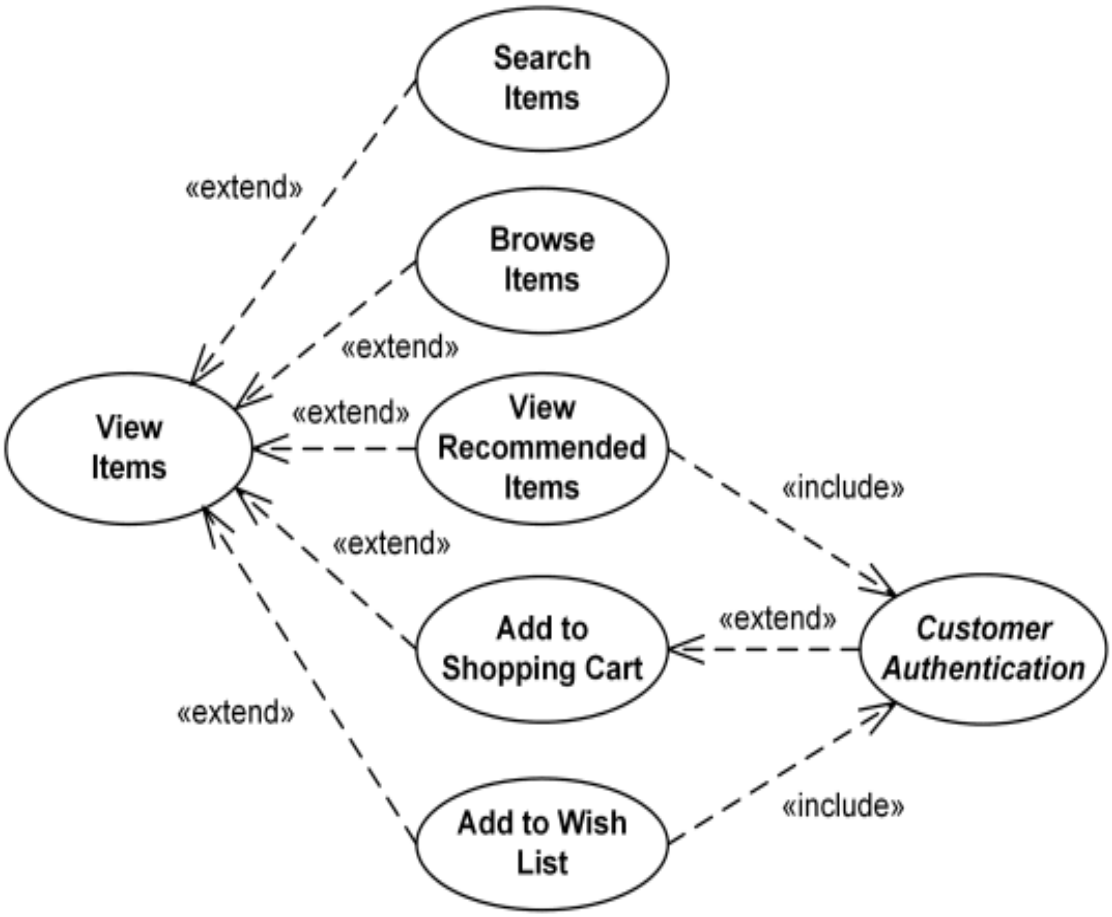
Typically, an interaction diagram captures the behavior a single use cases. The diagram shows a number of example objects and the messages that are passed between these objects in use cases. These are following approaches with simple use case that exhibits the following behavior.

Objects can send a message to another. Each message is checks with given stock item. There are two diagrams: Sequence and Collaboration diagram.

Package Diagram: One of the oldest questions in software methods is: how do you break down a large system into smaller systems? It becomes difficult to understand and the changes we make to them.

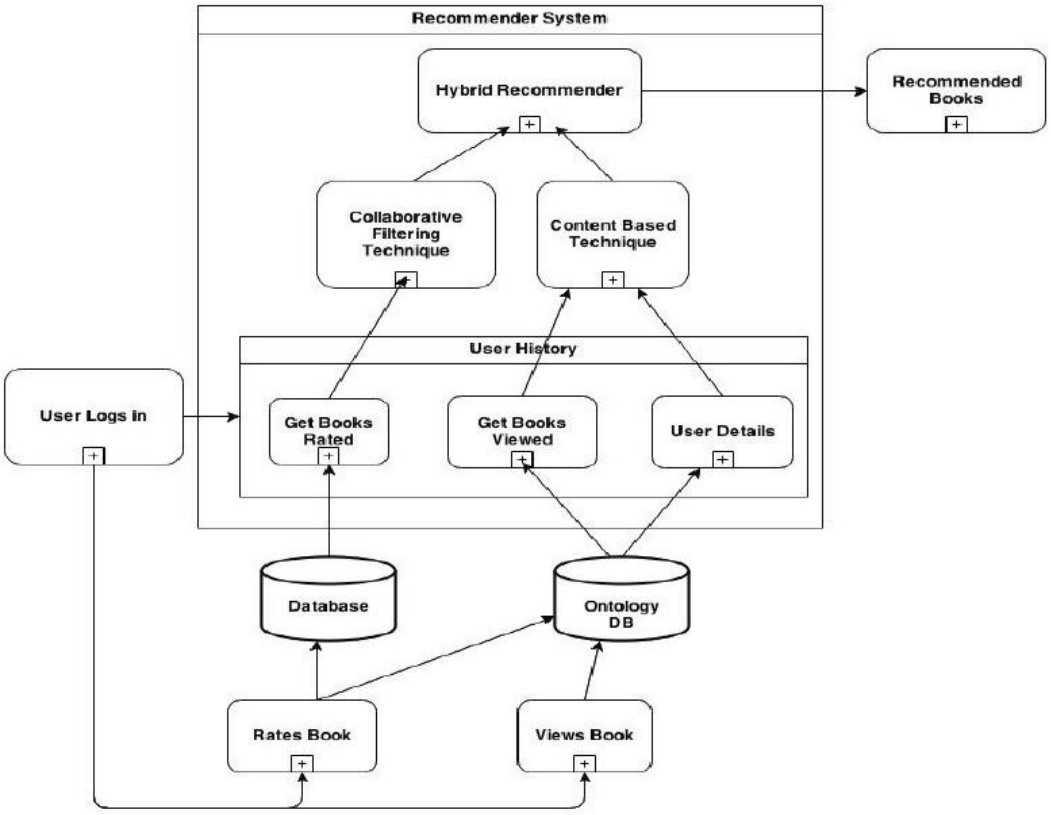
USECASE DIAGRAM:

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform. In this context, a "system" is something being developed or operated, such as a web site. The "actors" are people or entities operating under defined roles within the system



ACTIVITY DIAGRAM:

An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed. We can depict both sequential processing and concurrent processing of activities using an activity diagram. They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system. An activity diagram is a behavioral diagram. An activity diagram is very similar to a flowchart.



5. IMPLEMENTATION & RESULTS

Importing Surprise to Python Code

Python is used to write to execute the algorithm.

Surprise package is imported to use KNN algorithm

```
In [20]: import pandas as pd

In [16]: from surprise import Dataset
         #from surprise import Reader

In [17]: data = Dataset.load_builtin('ml-100k')
```

Training Algorithm

k-nearest neighbor algorithm is trained using a subset of 'm1-100k' dataset.

```
In [18]: from surprise import KNNBasic

In [19]: sim_options = {
             "name": "cosine",
             "user_based": False,  # Compute  similarities between items
         }

In [6]: algo = KNNBasic(sim_options=sim_options)

In [7]: from surprise.model_selection import train_test_split

In [8]: trainset, testset = train_test_split(data, test_size=.25)

In [9]: algo.fit(trainset)
        Computing the cosine similarity matrix...
        Done computing similarity matrix.

Out[9]: <surprise.prediction_algorithms.knns.KNNBasic at 0x4a353c8>
```

Calculating RMSE value

Root mean square is calculated for algorithm built to know accuracy.

```
In [11]: from surprise import accuracy

In [12]: accuracy.rmse(predictions)

         RMSE: 1.0300

Out[12]: 1.0299783766095976
```

Predictions

```
In [13]: predictions = algo.fit(trainset).test(testset)

         Computing the cosine similarity matrix...
         Done computing similarity matrix.

In [14]: predictions

Out[14]: [Prediction(uid='2', iid='301', r_ui=4.0, est=3.8464581809974905, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='795', iid='97', r_ui=2.0, est=3.2250366648836786, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='106', iid='1242', r_ui=4.0, est=3.8771634423227708, details={'actual_k': 30, 'was_impossible': False}),
          Prediction(uid='210', iid='174', r_ui=5.0, est=4.150554452355137, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='85', iid='317', r_ui=3.0, est=3.473712401735669, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='794', iid='181', r_ui=4.0, est=4.3459895487295075, details={'actual_k': 26, 'was_impossible': False}),
          Prediction(uid='312', iid='222', r_ui=3.0, est=4.375432610403982, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='588', iid='402', r_ui=5.0, est=3.5220592872632976, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='62', iid='704', r_ui=2.0, est=3.349031975001474, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='493', iid='69', r_ui=5.0, est=4.048639840249818, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='535', iid='921', r_ui=4.0, est=4.199541258841566, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='711', iid='566', r_ui=2.0, est=3.574777038835242, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='311', iid='845', r_ui=4.0, est=3.722516192703832, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='151', iid='356', r_ui=2.0, est=4.101159857126149, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='347', iid='286', r_ui=3.0, est=3.7274821444187753, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='320', iid='403', r_ui=4.0, est=4.051411498208947, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='740', iid='269', r_ui=4.0, est=3.342489740732474, details={'actual_k': 15, 'was_impossible': False}),
          Prediction(uid='234', iid='596', r_ui=2.0, est=3.2492895639752697, details={'actual_k': 40, 'was_impossible': False}),
          Prediction(uid='429', iid='101', r_ui=4.0, est=3.199370133906652, details={'actual_k': 40, 'was_impossible': False}),
```

In [14]:
```
predictions
```

```
Prediction(uid='504', iid='98', r_ui=5.0, est=3.799231333168856, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='450', iid='393', r_ui=4.0, est=3.674023783529555, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='495', iid='1188', r_ui=5.0, est=3.8259164104921495, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='385', iid='656', r_ui=5.0, est=3.599815623923977, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='846', iid='505', r_ui=5.0, est=3.0479391481044384, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='588', iid='220', r_ui=5.0, est=3.8235836735294053, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='864', iid='234', r_ui=4.0, est=4.024368845118488, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='222', iid='829', r_ui=3.0, est=2.45, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='591', iid='25', r_ui=4.0, est=3.8545351990685255, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='259', iid='168', r_ui=5.0, est=3.8502328908568475, details={'actual_k': 38, 'was_impossible': False}),
Prediction(uid='387', iid='188', r_ui=5.0, est=3.5473738772772485, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='228', iid='272', r_ui=5.0, est=3.142155355134251, details={'actual_k': 14, 'was_impossible': False}),
Prediction(uid='268', iid='218', r_ui=2.0, est=3.12241802060291, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='851', iid='412', r_ui=2.0, est=3.275976710357323, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='44', iid='82', r_ui=4.0, est=3.824778336275371, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='184', iid='192', r_ui=4.0, est=3.873543288443668, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='541', iid='924', r_ui=5.0, est=3.6240147888747374, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='43', iid='411', r_ui=3.0, est=3.6751039366746885, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='612', iid='476', r_ui=3.0, est=3.2240236905409185, details={'actual_k': 19, 'was_impossible': False}),
```

In [14]:
```
predictions
```

```
Prediction(uid='524', iid='855', r_ui=4.0, est=3.549505993788338, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='494', iid='194', r_ui=4.0, est=3.9531473870131535, details={'actual_k': 34, 'was_impossible': False}),
Prediction(uid='316', iid='292', r_ui=4.0, est=3.6006958366258806, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='405', iid='443', r_ui=4.0, est=1.225, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='537', iid='718', r_ui=4.0, est=2.875, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='429', iid='441', r_ui=3.0, est=2.9705162727074303, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='640', iid='231', r_ui=5.0, est=4.323976037942228, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='171', iid='245', r_ui=3.0, est=3.3400944619856068, details={'actual_k': 18, 'was_impossible': False}),
Prediction(uid='94', iid='293', r_ui=4.0, est=3.6479912283666245, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='527', iid='956', r_ui=4.0, est=3.5993856542095632, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='7', iid='418', r_ui=4.0, est=4.198050562858097, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='921', iid='760', r_ui=2.0, est=3.2450925480627872, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='473', iid='137', r_ui=4.0, est=3.954622923750732, details={'actual_k': 23, 'was_impossible': False}),
Prediction(uid='387', iid='48', r_ui=4.0, est=3.899898926671864, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='592', iid='735', r_ui=5.0, est=3.3233681559132178, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='260', iid='350', r_ui=4.0, est=3.9509896402819495, details={'actual_k': 16, 'was_impossible': False}),
Prediction(uid='617', iid='573', r_ui=4.0, est=2.478874159087827, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='263', iid='568', r_ui=4.0, est=4.200516475890386, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='389', iid='657', r_ui=5.0, est=4.0982586260614955, details={'actual_k': 40, 'was_impossible': False}),
Prediction(uid='189', iid='503', r_ui=3.0, est=3.9732616741223534, details={'actual_k': 40, 'was_impossible': False}),
```

6. TESTING & VALIDATION

Introduction:

The development of software systems involves a series of production activities where opportunities for injection of human fallibilities are enormous. Errors may begin to occur at the very inception of the process where the objectives may be erroneously or imperfectly specified, as well as in later design and development stages. Because of the human inability to perform and communicate with perfection, software development is accompanied by a quality assurance activity.

TESTING TECHNIQUES

Testing is the process of executing a program with the intention of finding errors. The various test strategies used for testing the software are as follows.

- Unit Testing
- Integration Testing
- Validation Testing
- System Testing

Unit Testing:

Unit testing focuses on verification effort on the smallest unit of the software design module. The main goal is to make sure that every source statement and the logic path has been executed correctly at least once. The output of this stage is the source code.

Integration Testing:

In Integration testing, we find errors that have occurred during the integration. After testing each module, which is then integrated into subsystems and then to form the entire system on which integration testing is performed. The goal of testing is to detect design errors while focusing on testing the interconnection between modules.

Validation Testing:

This testing concentrates on confirming that the software is error-free in all respects. All the specified validations are verified and the software is subjected to hard-core testing. It also aims at determining the degree of deviation that exists in the software designed from the specification; they are listed out and are corrected.

System Testing:

In this testing, the system is tested for the errors after coupling all the modules together. The system is tested against the specified requirements to see if all the requirements are met and the system performs as specified by the requirements.

Test cases:

A test case is a software testing document, which consists of the event, action, input, output, expected result, and actual result. Larger test cases may also contain prerequisite states or steps, and descriptions. A test case should also contain a place for the actual result. These steps can be stored in a word processor document, spreadsheet, database or another common repository.

TEST CASES:

1. Different CV’s are derived from the trainset and given to the algorithm, based on rmse value accuracy is test.

2. Less is the rmse value more probable are the predictions.

```
In [11]: from surprise import accuracy

In [12]: accuracy.rmse(predictions)

         RMSE: 1.0300

Out[12]: 1.0299783766095976
```

7. CONCLUSION

Recommender systems allow e-commerce sites to be highly customizable for the user and buyer. They allow companies to better understand their users, provide personalized stores, and in turn increase customer satisfaction and loyalty. They are implemented by utilizing various existing data mining tool sand adapting them to current needs.

Collaborative filtering allows the active user to get recommendation based on products that users with similar interest have purchased and rated positively, and by using the active user's previous ratings and transaction history to build a model that provides a new set of similar products. Content based filtering compares the user's personal profile and preferences with the database to find products that are of interest and align with the active user and present them.

This also poses many challenges which include cold start, handling anonymous users, creating a social recommender system that can accommodate more than one active user, handling various different data sources and scalability with increased data.

8. REFERENCES

- ✓ Surprise document

  https://surprise.readthedocs.io/en/stable/getting_started.html

- ✓ Article on recommender systems

  https://realpython.com/build-recommendation-engine-collaborative-filtering/

- ✓ SimilarityModule
  https://surprise.readthedocs.io/en/stable/similarities.html#module-surprise.similarities

- ✓ Data set

  https://grouplens.org/datasets/movielens/