PES UNIVERSITY RR CAMPUS

ALGORITHMS FOR INFORMATION RETRIEVAL AND INTELLIGENT WEB

UE20CS332 - REPORT

ASSIGNMENT : 2

TITLE : Implementing a Recommender system .

CORPUS : Steam games Dataset.

1. PROBLEM STATEMENT :

A recommender system is an information filtering system that predicts and suggests items to users based on their past behavior and preferences. To implement a recommender system, you will need to choose a relevant corpus, which is a collection of documents or text that can be used for analysis.

Tasks:

1. Perform EDA on the data: The first step in implementing a recommender system is to perform exploratory data analysis (EDA) on the dataset. EDA helps to understand the structure and features of the data. You can use techniques such as data visualization, summary statistics, and correlation analysis to explore the data.

2. Preprocess the data: After performing EDA, the next step is to preprocess the data. Preprocessing involves cleaning and transforming the data into a format suitable for analysis. This may involve tasks such as removing duplicates, handling missing values, and tokenizing the text data.

3. Use neighborhood-based or model-based collaborative filtering for recommendation: Collaborative filtering is a technique used in recommender systems to predict user preferences by analyzing their past behavior and comparing it with that of similar users. Neighborhood-based collaborative filtering involves finding similar users and recommending items that they have liked. Model-based collaborative filtering involves building a model that can predict user preferences based on features such as item attributes and user demographics.

4. Content-based recommendation: Content-based recommendation is another technique used in recommender systems. It involves analyzing the features of items and recommending items that are similar to those that a user has liked in the past.

5. Analyze the results: After implementing the recommender system, you will need to analyze the results to determine its effectiveness. You can use techniques such as A/B testing and user surveys to evaluate the system's performance.

6. Use suitable evaluation metrics: To evaluate the performance of the recommender system, you will need to use suitable evaluation metrics such as precision, recall, and F1 score. These metrics help to measure the system's accuracy and effectiveness in recommending relevant items to users.

In conclusion, implementing a recommender system involves performing EDA, preprocessing the data, choosing a suitable recommendation technique such as neighborhood-based or model-based collaborative filtering, content-based recommendation, analyzing the results, and using suitable evaluation metrics to evaluate the system's performance.

2. INTRODUCTION :

The video game industry has experienced exponential growth in recent years, with the availability of online platforms making it easier for gamers to access a wide range of titles. However, with so many options available, it can be challenging for users to discover new games that align with their interests. This is where a recommender system can prove to be invaluable. In this project, we will explore the use of collaborative filtering and content-based recommendation methods to build a game recommendation system using the Steam dataset. Additionally, we will perform an analysis of the popularity of different game genres to gain insights into user preferences. Finally, we will evaluate the effectiveness of our recommender system using various metrics, including MAP, MAR, coverage, and novelty, and compare it to random and popularity-based recommenders. Overall, this project aims to provide a useful tool for gamers to discover new titles and enhance their gaming experience.

3. DATASET DESCRIPTION :

The dataset we are using is called "Game Recommendations on Steam," and it contains information related to video games on the Steam platform, including user data, game data, and interactions between users and games. The dataset includes three files:

1. **games.csv**: The file contains information about games and add-ons available on steam.

We have nearly 60k rows in this file.

It has column header as:

1. **app_id**: which has the id number of a product
2. **title**: product title is the second column which has the name of the product.
3. **date_release**: product release data is the third column which has information about the release date of that product(mean Game).

4. **Win**: Support windows which tells which operating system the product (Game) works fine. It is just a binary classification which is true or false.
5. **Mac**: Support MacOS which tells which operating system the product (Game) works fine. It is just a binary classification which is true or false.
6. **Linux**: Support Linux which tells which operating system the product (Game) works fine. It is just a binary classification which is true or false.
7. **rating**: Product Rating Category is the column where it tells about the rating the user has given to that particular product (Game). Here the rating will be in three different categories: Very positive, positive and Other.
8. **positive_ratio**: Ratio of positive feedback contains what is the ratio of users given positive feedback.
9. **user_reviews**: Amount of user left means the user who has left/ uninstalled the product.
10. **price_final**: Final price in US dollar $.

2. users.csv: The file contains about the users registered on stream.

The file has three columns that is user_id, products and reviews and it has 6.17M rows.

1. **user_id**: User's auto-generated ID.
2. **products**: Number of products in the user's library.
3. **reviews**: Number of reviews published

3. recommendation.csv: The file contains eight columns with 10M rows.

It has user reviews with product Id to user Id relations.

The file has column header as:

1. **app_id**: Native product Id on Steam.
2. **helpful**: how many users found a recommendation helpful.
3. **funny**: how many users found a recommendation funny.
4. **date**: Date of publishing.
5. **is_recommended**: Is the user recommending the product?
6. **hours**: How many hours played by the user.
7. **user_id**: User's auto-generated ID.
8. **review_id**: User's review auto-generated ID.

The dataset contains cleaned and preprocessed 10M+ samples of user recommendations (reviews) from a Steam Store - a leading online platform for purchasing and downloading video games, DLC, and other gaming-related content. Additionally, it contains detailed information about games and add-ons.

Overall, this dataset provides a rich source of data for building a game recommendation system on the Steam platform using various techniques such as collaborative filtering, content-based filtering, and hybrid methods.

Acknowledgements:

The dataset was collected from the Steam Official Store. All rights on the dataset thumbnail image belong to the Valve Corporation.

Inspiration:

Use this dataset to practice building a game recommendation system or performing an Exploratory Data Analysis on products from a Steam Store.

4. EDA (EXPLORATORY DATA ANALYSIS) :

In this phase we are doing various analysis on the steam games dataset we have downloaded.

At first, we append the metadata of this dataset with the existing dataset to get a dataframe. After that, these are the number of samples :-

Let's investigate how many samples in each table.

```
pd.DataFrame([
    ["Recommendations", len(recommendations.index)],
    ["Users", len(users.index)],
    ["Games", len(games.index)],
    ["Games metadata", len(games_metadata.index)]
], columns=["Dataframe", "Records"]).style.hide(axis="index")
```

Dataframe	Records
Recommendations	11265799
Users	6167727
Games	48318
Games metadata	48318

These are the tables with their data

Games table :-

```
games.head()
```

	app_id	title	date_release	win	mac	linux	rating	positive_ratio	user_reviews	price_final	price_original	discount	steam_deck
0	10090	Call of Duty: World at War	2008-11-18	True	False	False	Very Positive	92	37039	19.99	19.99	0.0	True
1	13500	Prince of Persia: Warrior Within™	2008-11-21	True	False	False	Very Positive	84	2199	9.99	9.99	0.0	True
2	22364	BRINK: Agents of Change	2011-08-03	True	False	False	Positive	85	21	2.99	2.99	0.0	True
3	113020	Monaco: What's Yours Is Mine	2013-04-24	True	True	True	Very Positive	92	3722	14.99	14.99	0.0	True
4	226560	Escape Dead Island	2014-11-18	True	False	False	Mixed	61	873	14.99	14.99	0.0	True

Recommendations table :-

```python
recommendations.head()
```

	app_id	helpful	funny	date	is_recommended	hours	user_id	review_id
0	975370	0	0	2022-12-12	True	36.3	21058	0
1	304390	4	0	2017-02-17	False	11.5	1153	1
2	1085660	2	0	2019-11-17	True	336.5	96124	2
3	703080	0	0	2022-09-23	True	27.4	98074	3
4	526870	0	0	2021-01-10	True	7.9	9634	4

Users table :-

Users table contains following attributes:

Primary key: user autogenerated ID;

Number of products purchased by the user;

Number of reviews published on Steam;

```python
users.head()
```

	user_id	products	reviews
0	6043251	156	1
1	3802782	329	3
2	2053748	176	2
3	2214328	98	2
4	2451716	144	3

We also confirmed that, there are no missing values in any of the tables.

Dataframe	% of missing values
Recommendations	0.000000
Users	0.000000
Games	0.000000

Now let us move on to the statistics :-

Base statistics

Before we proceed with the key investigations, we need to get descriptive statistics on numerical data.

```
games[["positive_ratio", "user_reviews", "price_final", "discount"]].describe(percentiles=[.1, .25, .5, .75, .9]).round(2).T
```

	count	mean	std	min	10%	25%	50%	75%	90%	max
positive_ratio	48318.0	76.92	18.18	0.0	51.0	67.00	81.00	91.00	96.00	100.00
user_reviews	48318.0	1767.66	38168.21	10.0	12.0	20.00	51.00	215.00	1125.00	6941137.00
price_final	48318.0	8.67	11.47	0.0	0.0	0.99	4.99	11.24	19.99	299.99
discount	48318.0	5.08	18.01	0.0	0.0	0.00	0.00	0.00	0.00	90.00

+ Code + Markdown

```
users[["products", "reviews"]].describe(percentiles=[.1, .25, .5, .75, .9]).round(2).T
```

	count	mean	std	min	10%	25%	50%	75%	90%	max
products	6167727.0	134.99	267.23	0.0	8.0	23.0	64.0	154.0	312.0	29308.0
reviews	6167727.0	1.83	2.33	0.0	1.0	1.0	1.0	2.0	3.0	271.0

```
recommendations[["helpful", "funny", "hours"]].describe(percentiles=[.1, .25, .5, .75, .9]).round(2).T
```

	count	mean	std	min	10%	25%	50%	75%	90%	max
helpful	11265799.0	3.48	56.40	0.0	0.0	0.0	0.0	0.0	3.0	36212.0
funny	11265799.0	1.18	40.23	0.0	0.0	0.0	0.0	0.0	0.0	28088.0
hours	11265799.0	147.70	210.06	0.0	5.3	16.2	54.5	181.1	455.5	999.9

Popularity Analysis:

In this analysis we will define "popular" as games having a high number of user reviews. So, games with higher number of reviews means it is much more popular than games with lower number of user reviews.

1. Top 10 popular games of 2022 with positive reviews

First, filtering games which were released in 2022 and have a positive_ratio more than 90. Then sorting the data by user_reviews and positive_ratio in descending order to get popular games that people liked the most.

	title	user_reviews	positive_ratio
9226	ELDEN RING	481754	91
3591	Raft	218598	93
2668	Vampire Survivors	175903	98
1351	Stray	101109	97
6587	God of War	65968	97
13119	Teardown	60815	96
12167	Marvel's Spider-Man Remastered	41232	96
7964	Cult of the Lamb	40135	93
8307	PowerWash Simulator	29465	97
10731	LEGO® Star Wars™: The Skywalker Saga	28451	92

2. Top 10 popular games of 2022 with mixed or lower reviews

Filtering data which has positive_ratio less than 70 and sorting by user_reviews and positive_ratio columns.

	title	user_reviews	positive_ratio
16364	Call of Duty®: Modern Warfare® II	316804	62
38840	Mirror 2: Project X	110981	26
2112	EA SPORTS™ FIFA 23	67263	50
12660	Warhammer 40000: Darktide	54741	54
19737	World War 3	49594	54
20602	The Cycle: Frontier	39025	61
393	Dread Hunger	38500	61
10539	The Callisto Protocol™	21737	62
28674	Call of Duty®: Warzone™ 2.0	19770	34
14964	Victoria 3	19223	67

3. Top 10 popular games of 2022 among mac apple users

Filtering data that was released in 2022, has positive_ratio more than 90 and are available for mac users.

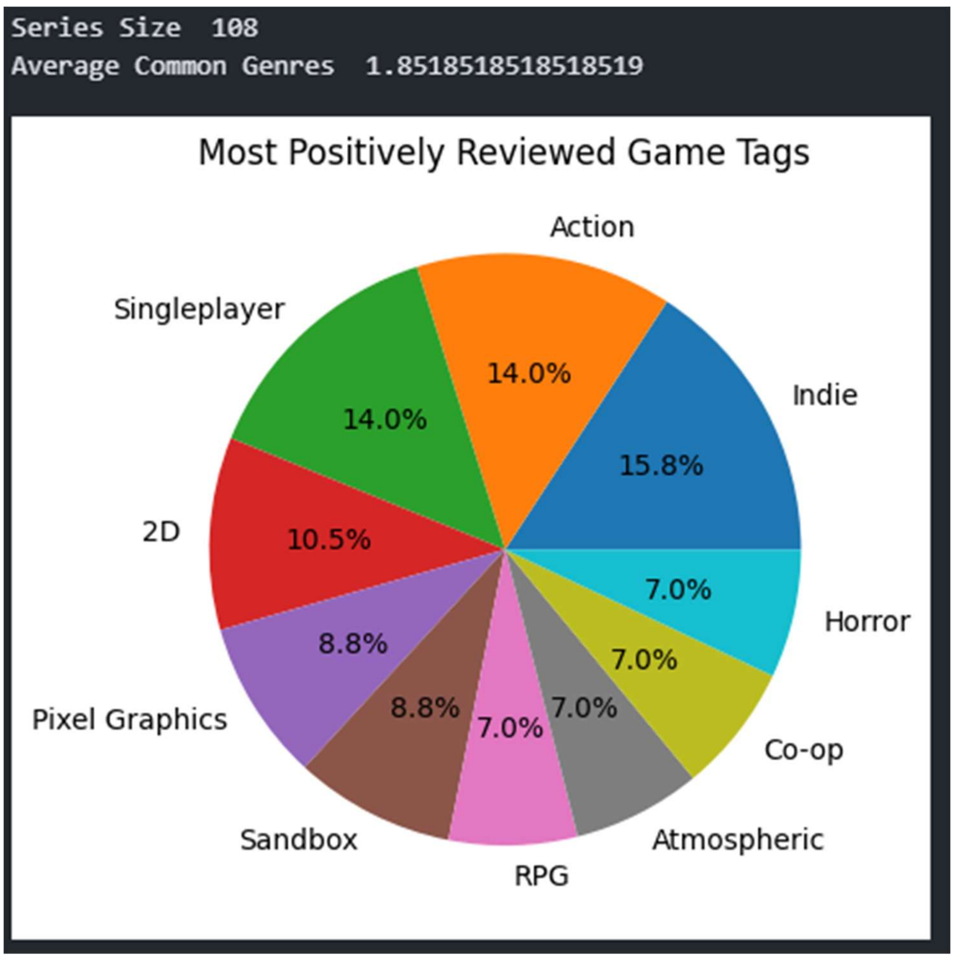
	title	user_reviews	positive_ratio
2668	Vampire Survivors	175903	98
7964	Cult of the Lamb	40135	93
2773	NEEDY STREAMER OVERLOAD	19517	95
18158	20 Minutes Till Dawn	18493	92
4267	The Stanley Parable: Ultra Deluxe	18171	94
17732	Stacklands	16917	96
3820	Rogue Legacy 2	11679	90
10504	Quaver	11567	92
28789	The Looker	10889	97
17413	Soulstone Survivors	8873	91

Genre Popularity Analysis:

Based on this, we can see that some of the most popular games share common tags. However, some of these tags seem like they are very similar, things like Multiplayer, Co-op, and Online Co-op are three tags that one game could have. Some insights that we can find from this list is that Action is the most popular genre, with other notable tags such as Adventure, Simulation, FPS, RPG, and Strategy. Some of these tags describe features in a game rather than a genre, popular features seem to be Multiplayer, Open World, and First Person Shooter (FPS). The amount of multiplayer and singleplayer games are very close to being

even, therefore it does not seem like there is a strong preference between one or the other.
This list does have some flaws however. This list of games was gathered based on the amount of reviews a game has, whether the reviews were positive or not had no bearing. Now it could be worthwhile looking at some of the most positively reviewed games and comparing their tags to the most common ones that we have found here.

Most positively reviewed Game tags :-



Now we find a very interesting result. After taking the most positively reviewed games, the common tags look very different from before. We see now that instead of Action, Indie is the most popular tag on this list. Action is still a close second. We also see that singleplayer is more common now than multiplayer. New tags that were previously not as significant in the larger list, such as 2D, Pixel Graphics, Horror, Building, and Roguelike. From this we can see that indie games perform extremely well when they focus on creating a strong single player experience, with some of the other tags mentioned in the graph such as Sandbox and RPG. On the other hand, larger studios will have more success with making a game that is either singleplayer or multiplayer and it has elements which would make it an Adventure, FPS, and Open World.

5. PREPROCESSING AND HANDLING DOMAIN CHALLENGE :

As far as the preprocessing is concerned, the dataset is pre-cleaned with all the irrelevant words not present and null words not present .

Some of the other preprocessing done were, converting the dataset to dataframe, converting some member objects to the respective data type, etc…

Check missing values in tables

What is the percentage of missing values in a dataframe? Currently, there are no missing values in the dataset, so that we can proceed with further investigation.

```
[9]: def perc_na(df: pd.DataFrame) -> float:
         """Calculates a percentage of missing values in the dataframe."""
         return np.count_nonzero(df.isnull().values) / (df.shape[0] * df.shape[1]) * 100.0


     pd.DataFrame([
         ["Recommendations", perc_na(recommendations)],
         ["Users", perc_na(users)],
         ["Games", perc_na(games)]
     ], columns=["Dataframe", "% of missing values"]).style.hide(axis="index")
```

[9]:

Dataframe	% of missing values
Recommendations	0.000000
Users	0.000000
Games	0.000000

The date release field is object, so let's convert that to datetime.

```
[10]: games['date_release']=pd.to_datetime(games['date_release'])
```

Base statistics

Also, not all the data were loaded, but a subset in it is taken to accommodate the system requirements.

Thus, we begin by reading a CSV file containing recommendations. Then, we filter the recommendations based on the number of times a user has recommended an item. If a user has recommended an item less than 15 times, the recommendation is removed from the data.

Since the data is merely recommended = True/False, using this directly will lead to suboptimal results. For example, a person who played 1000/2000 hours of a game recommending against it should not be taken as negatively as a person recommending against it after 1 hour. Similarly, a person playing a game for 1000 hours should be taken as a higher positive recommendation than one playing for simply 1 or 2 hours.

Thus, a scaling function is defined and applied to each recommendation. This function scales the rating of each recommendation based on the number of hours played by the user. If the user has recommended an item, the rating is simply the number of hours played. If the user has not recommended an item, the rating is calculated using a logarithmic function.

After this, the data is transformed using a quantile cut to assign each rating to one of five labels, based on their rank within their user_id group.

[22]:
```python
import math
def scal_f(df):
  e = 2.718
  if(df['is_recommended']):
    return df['hours']
  else:
    return math.log2((1+df['hours'])) - 10
recommendations = recommendations[recommendations.groupby('user_id')['user_id'].transform('size') > 15]
recommendations
```

[22]:

	app_id	helpful	funny	date	is_recommended	hours	user_id	review_id
58	602960	0	0	2022-01-05	True	41.9	279151	58
138	570940	348	39	2021-12-11	False	73.3	2199166	138
139	397540	0	0	2020-07-18	True	86.3	2243392	139
141	39210	0	0	2021-04-09	True	123.1	2260360	141
151	1196590	0	0	2022-07-03	True	19.0	2575791	151
...	...	...	...	...	...	...	...	...
11265720	750920	32	0	2021-10-29	True	526.3	5583454	11265720
11265723	666140	7	0	2018-07-31	True	289.4	1153783	11265723
11265756	476600	2788	79	2017-11-03	False	31.9	5110523	11265756
11265770	640820	2	0	2021-12-02	True	69.9	3634187	11265770
11265782	512900	37	17	2022-07-06	True	52.4	5128580	11265782

645912 rows × 8 columns

[23]:
```python
recommendations['rating'] = recommendations.apply(scal_f, axis=1)
recommendations['rating'] = recommendations.groupby(['user_id'])['rating'].transform(lambda x: pd.qcut(x.rank(method='first'), 5,labels=range(1,6), duplicates='drop'))
recommendations
```

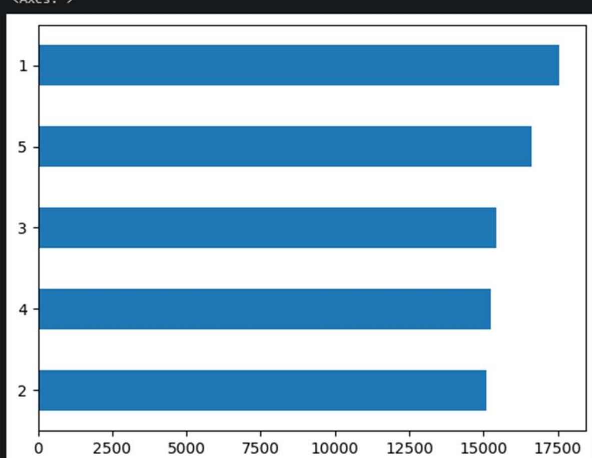
[23]:

	app_id	helpful	funny	date	is_recommended	hours	user_id	review_id	rating
58	602960	0	0	2022-01-05	True	41.9	279151	58	5
138	570940	348	39	2021-12-11	False	73.3	2199166	138	1
139	397540	0	0	2020-07-18	True	86.3	2243392	139	4
141	39210	0	0	2021-04-09	True	123.1	2260360	141	5
151	1196590	0	0	2022-07-03	True	19.0	2575791	151	4

This is the distribution of ratings after the transformation, which is even as expected.

[25]:
```python
recommendations.rating.value_counts().sort_values().plot(kind = 'barh')
```

[25]: <Axes: >



The slight bias towards ratings 1 and 5 can be explained by the law of extremes: people who really dislike a game are more likely to vent their anger/annoyance by posting a bad review. Similar for people who really like a game, this accounts for the relative increase in the amount of 1 or 5 reviews.

6. METHODOLOGY (NEIGHBORHOOD BASED OR MODEL BASED COLLABORATIVE FILTERING) :-

Collaborative filtering is a technique used by recommender systems to recommend items to users based on their similarities with other users. Neighborhood-based collaborative filtering is a type of collaborative filtering that uses similarities between items or users to make recommendations. In this approach, the system identifies similar items or users to the target user, and recommends items that are liked by similar users or items.

```
class KNNWithMeans(SymmetricAlgo):
    def __init__(self, k=40, min_k=1, sim_options={}, verbose=True, **kwargs):

        SymmetricAlgo.__init__(self, sim_options=sim_options, verbose=verbose, **kwargs)

        self.k = k
        self.min_k = min_k

    def fit(self, trainset):

        SymmetricAlgo.fit(self, trainset)
        self.sim = self.compute_similarities()

        self.means = np.zeros(self.n_x)
        for x, ratings in self.xr.items():
            self.means[x] = np.mean([r for (_, r) in ratings])

        return self

    def estimate(self, u, i):

        if not (self.trainset.knows_user(u) and self.trainset.knows_item(i)):
            raise PredictionImpossible("User and/or item is unknown.")

        x, y = self.switch(u, i)

        neighbors = [(x2, self.sim[x, x2], r) for (x2, r) in self.yr[y]]
        k_neighbors = heapq.nlargest(self.k, neighbors, key=lambda t: t[1])

        est = self.means[x]

        # compute weighted average
        sum_sim = sum_ratings = actual_k = 0
        for (nb, sim, r) in k_neighbors:
            if sim > 0:
                sum_sim += sim
                sum_ratings += sim * (r - self.means[nb])
                actual_k += 1

        if actual_k < self.min_k:
            sum_ratings = 0

        try:
            est += sum_ratings / sum_sim
        except ZeroDivisionError:
            pass  # return mean

        details = {"actual_k": actual_k}
        return est, details
```

The KNNWithMeans class is a subclass of SymmetricAlgo that implements a basic collaborative filtering algorithm, taking into account the mean ratings of each user.

Finally, the KNNWithMeans class is defined, which is a neighborhood-based collaborative filtering algorithm. This class defines a fit() method, which computes similarities between items or users, and an estimate() method, which uses these similarities to make recommendations for a given user. The algorithm computes the weighted average of ratings from k similar users to recommend items to the target user.

7. RESULTS FOR COLLABORATIVE FILTERING :

A collaborative filtering algorithm, taking into account the mean ratings of each user

> k Nearest ,i.e,games most similar to that of the game (argument passed by the user) the user needs.

```
print_k_nearest_apps(recommendations, games, 'Rust')
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.

The 10 nearest neighbors of Rust are:
Grand Theft Auto V
Space Engineers
VRChat
BeamNG.drive
RimWorld
Metro Exodus
Red Dead Redemption 2
Call of Duty®: Black Ops III
The Witcher® 3: Wild Hunt
Call of Duty®: Modern Warfare® II
```

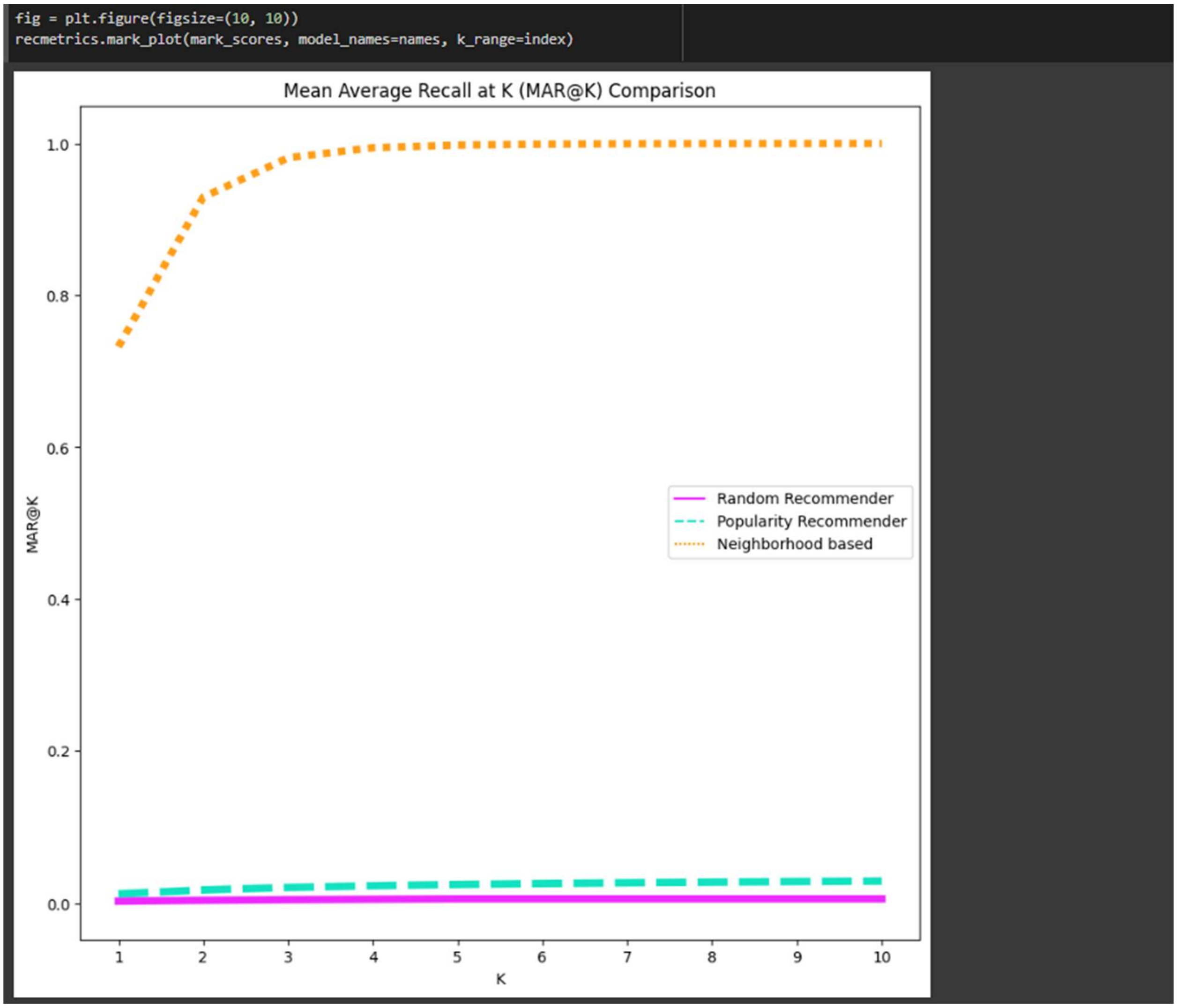
```
print_k_nearest_apps(recommendations, games, 'Team Fortress 2')
```

```
Estimating biases using als...
Computing the pearson_baseline similarity matrix...
Done computing similarity matrix.

The 10 nearest neighbors of Team Fortress 2 are:
Borderlands 2
Stellaris
Killing Floor 2
Grand Theft Auto IV: The Complete Edition
Sonic Adventure 2
Shadow of the Tomb Raider: Definitive Edition
Chicken Invaders Universe
Slime Rancher
Call of Duty®: Modern Warfare® 3
Friday the 13th: The Game
```

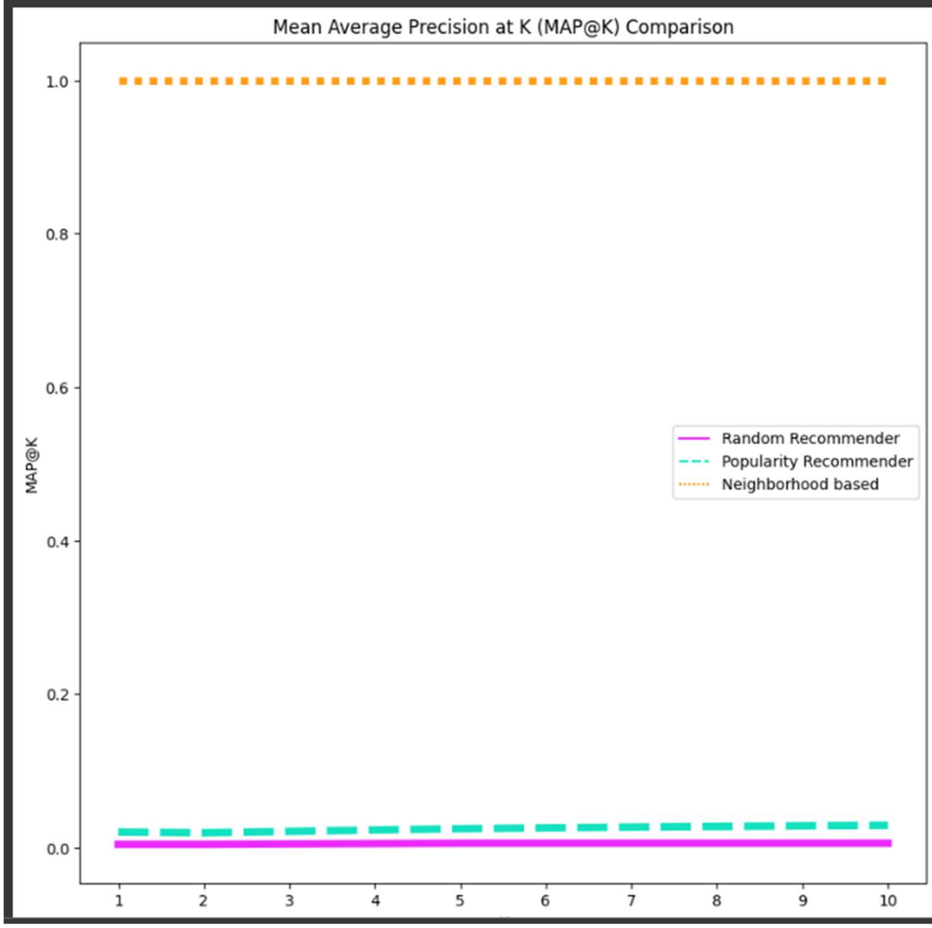
8. EVALUATION METRICS for NEIGHBORHOOD BASED COLLABORATIVE FILTERING:

> We have Performed Mean Average Recall Calculation on a Random Recommender system, a Popularity recommender system and our neighborhood based recommender model.



> We have also Performed Mean Average Precision Calculation on a Random Recommender system, a Popularity recommender system and our neighborhood based recommender model

```
fig = plt.figure(figsize=(10, 10))
recmetrics.mapk_plot(mark_scores, model_names=names, k_range=index)
```

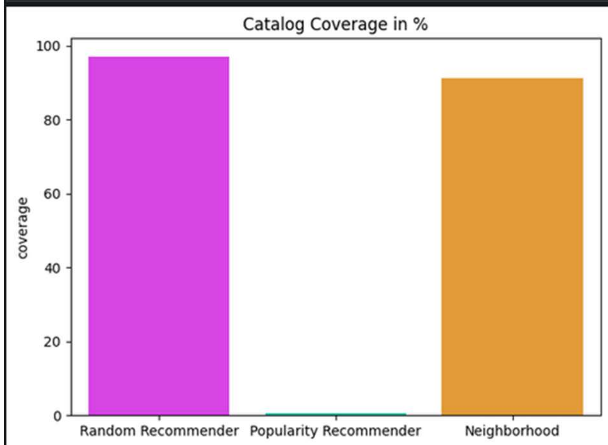


Calculation for coverage(percentage of items that are recommended by the system) and novelty(a measure of newness or distinctness) is done for the same.

```
# N=100 observed recommendation lists
random_cat_coverage = recmetrics.catalog_coverage(ran_recs, catalog, 100)
pop_cat_coverage = recmetrics.catalog_coverage(pop_recs, catalog, 100)
cf_cat_coverage_n = recmetrics.catalog_coverage(cf_recs_n, catalog, 100)
```

```
# plot of prediction coverage
coverage_scores = [random_coverage, pop_coverage, cf_coverage_n]
model_names = ['Random Recommender', 'Popularity Recommender', 'Neighborhood']

fig = plt.figure(figsize=(7, 5))
recmetrics.coverage_plot(coverage_scores, model_names)
```




```
nov = recommendations.app_id.value_counts()
pop = dict(nov)
```

```
random_novelty,random_mselfinfo_list = recmetrics.novelty(ran_recs, pop, len(users), 10)
pop_novelty,pop_mselfinfo_list = recmetrics.novelty(pop_recs, pop, len(users), 10)
cf_novelty_n,cf_mselfinfo_list = recmetrics.novelty(cf_recs_n, pop, len(users), 10)
```

```
print(random_novelty, pop_novelty, cf_novelty_n)
```

```
7.807899792586648 13.27177957652021 18.940192493145776
```

As expected a random recommender achieves near 100% coverage since it recommends anything regardless of any measure. Popularity based recommender just recommends the most popular 10 to everyone so it suffers. Our model achieves a good score on this metric, nearly 90% coverage.

For reference:

Random recommender: recommends a random sample of apps

Popularity: Always recommend the top k items in terms of occurrences

Neighborhood based: Our KNN model, collaborative

9. CONTENT BASED FILTERING :

Content-based filtering is a type of recommendation system that uses the attributes or features of an item to recommend other similar items to users. In content-based filtering, the system learns the user's preferences based on their interactions with items and then recommends similar items that match those preferences.

In the context of Steam games dataset, content-based filtering recommends games to users based on the attributes or features of the games.

> Screenshot of the games dataframe:

games_df

	app_id	title	win	mac	linux	user_reviews	price_final	steam_deck	tags
0	10090	Call of Duty: World at War	1	0	0	37039	19.99	1	[Zombies, World War II, FPS, Multiplayer, Acti...
1	13500	Prince of Persia: Warrior Within™	1	0	0	2199	9.99	1	[Action, Adventure, Parkour, Third Person, Gre...
2	22364	BRINK: Agents of Change	1	0	0	21	2.99	1	[Action]
3	113020	Monaco: What's Yours Is Mine	1	1	1	3722	14.99	1	[Co-op, Stealth, Indie, Heist, Local Co-Op, St...
4	226560	Escape Dead Island	1	0	0	873	14.99	1	[Zombies, Adventure, Survival, Action, Third P...
...	...	...	...	...	...	...	...	...	...
48313	1803600	Gone Rogue	1	0	0	65	12.49	1	[Stealth, Action, Real Time Tactics, Isometric...
48314	1811990	Wildfrost	1	0	0	1819	19.99	1	[Card Battler, Difficult, Roguelike Deckbuilde...
48315	1868120	BLACK ACE	1	0	0	76	0.00	1	[FPS, Third-Person Shooter, Multiplayer, Shoot...
48316	2206390	BandRoll	1	0	0	26	1.59	1	[3D Platformer, Rhythm, Parkour, 3D, Atmospher...
48317	2330750	Cataclysm: Dark Days Ahead	1	0	1	130	19.99	1	[Traditional Roguelike, Sandbox, RPG, Zombies,...

48318 rows × 9 columns

> We Totally have 441 genre Labels.

```python
from collections import Counter
genres_counts = Counter(g for genres in meta_df['tags'] for g in genres)
print(f"There are {len(genres_counts)} genre labels.")
genres_counts
```

```
There are 441 genre labels.
Counter({'Zombies': 1374,
         'World War II': 634,
         'FPS': 3046,
         'Multiplayer': 6727,
         'Action': 21373,
         'Shooter': 4318,
```

> Dimensions of our game features cosine similarity matrix: (48318, 48318)

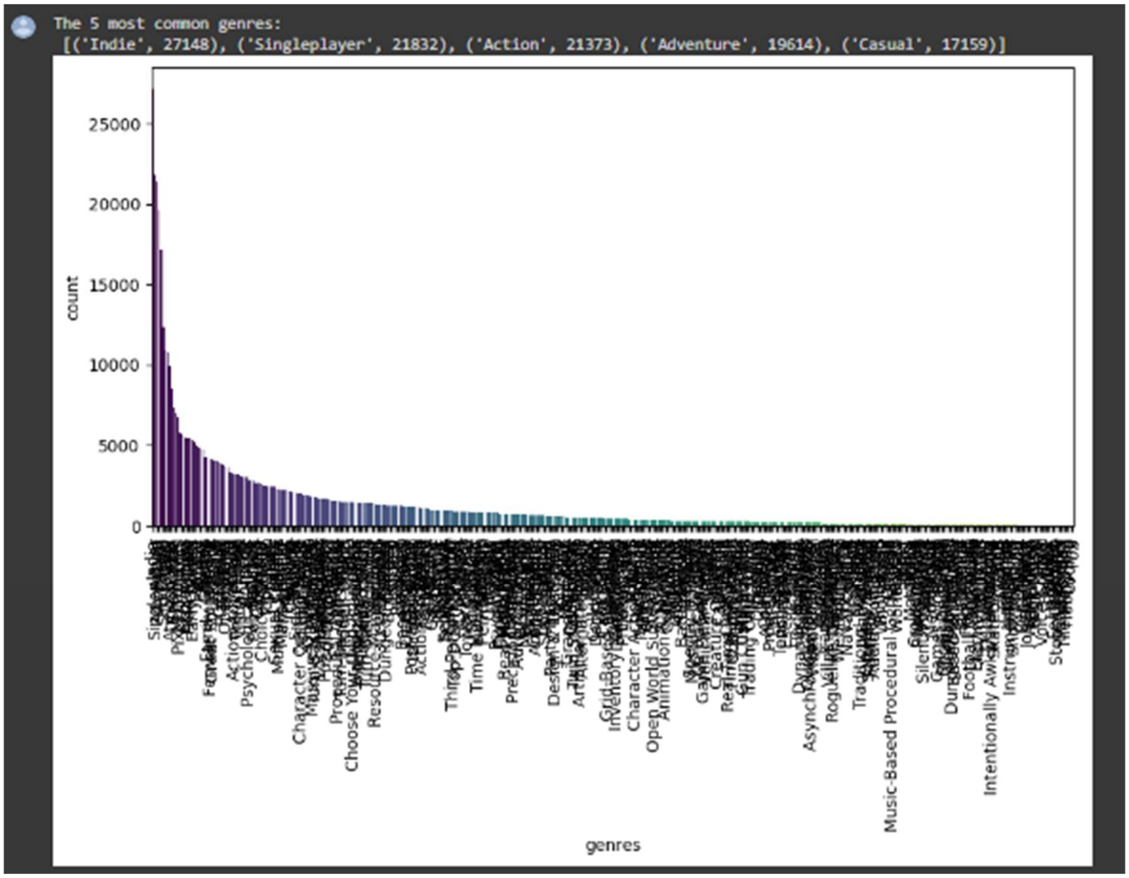
```python
from sklearn.metrics.pairwise import cosine_similarity
tmp = games_df.drop(columns=['app_id', 'title'], axis=1)
cosine_sim = cosine_similarity(tmp, tmp)
print(f"Dimensions of our game features cosine similarity matrix: {cosine_sim.shape}")
```

```
Dimensions of our game features cosine similarity matrix: (48318, 48318)
```

10. <u>RESULTS FOR CONTENT FILTERING :</u>

Printing the 5 most common game genres :

- **'Indie'**,**'Singleplayer'**, '**Action**',**'Adventure'**,**'Casual'** are the top 5 most common game genres according our analysis



Final Result:

- The First Argument passed through the get_recs function is the number of recommendations the user would like to receive.
- The Second Argument is the app_id of the game the person likes.

```
get_recs(10, 1)
get_recs(4, 300)
```

```
Because you liked 1    Prince of Persia: Warrior Within™
Name: title, dtype: object:
1                   Prince of Persia: Warrior Within™
2                             BRINK: Agents of Change
4                                  Escape Dead Island
6                                        METAL SLUG 3
8                        Mount & Blade II: Bannerlord
9     Men of War: Assault Squad 2 - Deluxe Edition u...
10                  Hyperdimension Neptunia Re;Birth1
11                                The Sum of All Fears
12                                          Cold Fear™
13                       LEGO® Harry Potter: Years 1-4
Name: title, dtype: object
Because you liked 300    The Witch & The 66 Mushrooms
Name: title, dtype: object:
1    Prince of Persia: Warrior Within™
2              BRINK: Agents of Change
4                   Escape Dead Island
6                         METAL SLUG 3
Name: title, dtype: object
```

11. Hybrid Recommender System :

This Hybrid Recommender model allows the users to provide their game genre, price range, and OS preferences, and generates initial recommendations using content-based filtering.

Collaborative filtering is then used to refine the recommendations based on the user's further preferences and likings.

This approach provides more personalized and relevant recommendations to users.

airiw-2-final.ipynb

[102]:
```
def read_item_names(app_df):
    """Read the u.item file from dataset and return two
    mappings to convert raw ids into movie names and movie names into raw ids.
    """
    app = app_df
    rid_to_name = dict(zip(app['app_id'], app['title']))
    name_to_rid = dict(zip( app['title'],app['app_id']))

    return rid_to_name, name_to_rid

def get_k_nearest_apps(algo, app_df, app_name):

  # Read the mappings raw id <-> movie name
  rid_to_name, name_to_rid = read_item_names(app_df)

  app_raw_id = name_to_rid[app_name]
  app_inner_id = algo.trainset.to_inner_iid(app_raw_id)

  app_neighbors = algo.get_neighbors(app_inner_id, k=3)

  # Convert inner ids of the neighbors into names.
  app_neighbors = (
      algo.trainset.to_raw_iid(inner_id) for inner_id in app_neighbors
  )
  app_neighbors_names = (rid_to_name[rid] for rid in app_neighbors)

  return app_neighbors, app_neighbors_names
```

[75]:
```
import math
def sim_p(n_x, yr, min_support):

    prods = np.zeros((n_x, n_x), np.double)
    # number of common ys
    freq = np.zeros((n_x, n_x), np.int_)
    # sum (r_xy ^ 2) for common ys
    sqi = np.zeros((n_x, n_x), np.double)
    # sum (r_x'y ^ 2) for common ys
    sqj = np.zeros((n_x, n_x), np.double)
    # the similarity matrix
    sim = np.zeros((n_x, n_x), np.double)

    min_sprt = min_support

    for y, y_ratings in yr.items():
        for xi, ri in y_ratings:
            for xj, rj in y_ratings:
                freq[xi, xj] += 1
                prods[xi, xj] += ri * rj
                sqi[xi, xj] += ri**2
                sqj[xi, xj] += rj**2

    for xi in range(n_x):
        sim[xi, xi] = 1
        for xj in range(xi + 1, n_x):
            if freq[xi, xj] < min_sprt:
                sim[xi, xj] = 0
            else:
                denum = math.sqrt(sqi[xi, xj] * sqj[xi, xj])
                sim[xi, xj] = prods[xi, xj] / denum

            sim[xj, xi] = sim[xi, xj]

    return np.asarray(sim)
```

```
class HybridModel(SymmetricAlgo):

    def __init__(self, k=40, min_k=1, sim_options={}, verbose=True, **kwargs):

        SymmetricAlgo.__init__(self, sim_options=sim_options, verbose=verbose, **kwargs)
        self.k = k
        self.min_k = min_k
        self.liked = []
    def fit(self, trainset):

        SymmetricAlgo.fit(self, trainset)
        self.sim = self.compute_similarities()

        return self
    def compute_similarities(self):
        """Build the similarity matrix.

        Returns:
            The similarity matrix."""

        construction_func = {
            "msd": sim_p,
        }

        if self.sim_options["user_based"]:
            n_x, yr = self.trainset.n_users, self.trainset.ir
        else:
            n_x, yr = self.trainset.n_items, self.trainset.ur

        min_support = self.sim_options.get("min_support", 1)

        args = [n_x, yr, min_support]

        name = self.sim_options.get("name", "msd").lower()
        if name == "pearson_baseline":
            shrinkage = self.sim_options.get("shrinkage", 100)
            bu, bi = self.compute_baselines()
            if self.sim_options["user_based"]:
                bx, by = bu, bi
            else:
                bx, by = bi, bu

            args += [self.trainset.global_mean, bx, by, shrinkage]

        try:
            if getattr(self, "verbose", False):
                print(f"Computing the {name} similarity matrix...")
            sim = construction_func[name](*args)
            if getattr(self, "verbose", False):
                print("Done computing similarity matrix.")
            return sim
        except KeyError:
            raise NameError(
                "Wrong sim name "
                + name
                + ". Allowed values "
                + "are "
                + ", ".join(construction_func.keys())
                + "."
            )
    def estimate(self, u, i):

        if not (self.trainset.knows_user(u) and self.trainset.knows_item(i)):
```

```python
        except KeyError:
            raise NameError(
                "Wrong sim name "
                + name
                + ". Allowed values "
                + "are "
                + ", ".join(construction_func.keys())
                + "."
            )
    def estimate(self, u, i):

        if not (self.trainset.knows_user(u) and self.trainset.knows_item(i)):
            raise PredictionImpossible("User and/or item is unknown.")

        x, y = self.switch(u, i)

        neighbors = [(self.sim[x, x2], r) for (x2, r) in self.yr[y]]
        k_neighbors = heapq.nlargest(self.k, neighbors, key=lambda t: t[0])

        # compute weighted average
        sum_sim = sum_ratings = actual_k = 0
        for (sim, r) in k_neighbors:
            if sim > 0:
                sum_sim += sim
                sum_ratings += sim * r
                actual_k += 1

        if actual_k < self.min_k:
            raise PredictionImpossible("Not enough neighbors.")

        est = sum_ratings / sum_sim

        details = {"actual_k": actual_k}
        return est, details
    def get_df_cond():
        print("Enter your maximum price in dollars.")
        max_p = float(input())
        print("Enter some tags you like. Seperate tags by commas.")
        tags = input().split(',')
        print("Do you want to be able to play it on Windows? Enter 1 for yes and 0 for doesn't matter")
        w = int(input())
        print("Do you want to be able to play it on Linux? Enter 1 for yes and 0 for doesn't matter")
        l = int(input())
        print("Do you want to be able to play it on MacOS? Enter 1 for yes and 0 for doesn't matter")
        m = int(input())
        print("Do you want to be able to play it on SteamDeck? Enter 1 for yes and 0 for doesn't matter")
        sd = int(input())
        return [max_p, tags, w, l, m, sd]
    def add_liked(self, item):
        self.liked.append(item)
```

```python
def narrow():
    meta_df = games_metadata.drop(columns = 'description', axis = 1)
    games = pd.read_csv('SteamDataset/games.csv')
    l = HybridModel.get_df_cond()
    cond = (games['price_final'] <= l[0])
    result = games[cond].app_id
    games = games_df
    cond = (games.app_id.isin(result)) & (games['win']==l[2] if l[3]==1 else True) & (games['linux']==l[3] if l[3]==1 else True) & (games['mac']==l[4] if l[4]==1 else True) & (games['steam_deck']==l[5] if l[5]==1 else True)
    result = games[cond].app_id
    cond = (meta_df.app_id.isin(result)) & (meta_df.tags.map(set(l[1]).issubset)) & (meta_df.app_id.isin(recommendations.app_id))
    result = meta_df[cond].app_id
    return result
```

```python
[78]: def train_hybrid(algo, q):
          reader = Reader(rating_scale=(1, 5))
          df = recommendations.loc[recommendations.app_id.isin(q)]
          print(df)
          data = Dataset.load_from_df(df[["user_id", "app_id", "rating"]], reader)
          trainset = data.build_full_trainset()
          algo.fit(trainset)
```

```python
[79]: algo = HybridModel(sim_options = {
          "name": "msd"
      })
      n_id = narrow()
      train_hybrid(algo, n_id)
```

```
Enter your maximum price in dollars.
 25
Enter some tags you like. Seperate tags by commas.
 Action,Zombies
Do you want to be able to play it on Windows? Enter 1 for yes and 0 for doesn't matter
 1
Do you want to be able to play it on Linux? Enter 1 for yes and 0 for doesn't matter
 0
Do you want to be able to play it on MacOS? Enter 1 for yes and 0 for doesn't matter
 0
Do you want to be able to play it on SteamDeck? Enter 1 for yes and 0 for doesn't matter
 0
           app_id  helpful  funny        date  is_recommended  hours  user_id  \
8891663     21690        0      0  2016-01-19            True   72.4  1866494
9916394    254700        0      0  2017-10-18           False    1.1   113880
11210190   418370       61     28  2017-07-29            True   41.8  4867281
11147316      500        0      0  2021-04-19            True    0.4   131015
8805000   1180380        0      0  2021-12-20            True    9.3  2776701
...           ...      ...    ...         ...             ...    ...      ...
9992490     17470        0      0  2022-08-16            True   10.0  4580536
11220641   418370       32      3  2018-02-05            True   35.4  6153452
5639684    242760        0      3  2020-04-25            True   27.4  3199846
9415306   1621070        0      0  2022-08-07            True    7.6    21903
11197407   555160       35     19  2018-12-11            True   19.7  6102118

          review_id rating
8891663     8891663      4
9916394     9916394      2
11210190   11210190      5
11147316   11147316      2
8805000     8805000      2
...             ...    ...
9992490     9992490      1
11220641   11220641      3
5639684     5639684      3
9415306     9415306      3
11197407   11197407      3

[2427 rows x 9 columns]
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
con1, con2 = read_item_names(games.loc[games.app_id.isin(n_id)])
import random
seed = random.choice(list(n_id))
y = get_k_nearest_apps(algo, games.loc[games.app_id.isin(n_id)], con1[seed])
for i in y[0]:
    print(con1[i])
print("Enter a list of numbers which you liked!")
z = list(map(int, input().split(',')))
o = []
c = 0
y = get_k_nearest_apps(algo, games.loc[games.app_id.isin(n_id)], con1[seed])
for i in y[0]:
    #print(c, i)
    if c in z:
        o.append(i)
    c += 1
o
```

```
Left 4 Dead 2
Dead Island Definitive Edition
Dead Space (2008)
DESOLATE
Guns Gore and Cannoli 2
Enter a list of numbers which you liked!
 1,2
[383150, 17470]
```

```
for i in o:
    algo.add_liked(i)
algo.liked
```

```
[383150, 17470]
```

```
def get_final_rec(algo, n_id):
    liked = algo.liked
    d = {}
    for i in liked:
        for j in get_k_nearest_apps(algo, games.loc[games.app_id.isin(n_id)], con1[i]):
            if(j not in d.keys()):
                d[j] = 1
            else:
                d[j] += 1
    return (max(d, key = d.get))
con1, con2 = read_item_names(games.loc[games.app_id.isin(n_id)])
print("You final recommendations are: ")
for i in get_final_rec(algo, n_id):
    if i not in o:
        print(con1[i])
```

```
You final recommendations are:
resident evil 4 (2005)
Left 4 Dead 2
DESOLATE
Guns Gore and Cannoli 2
```

```
#Second example
algo = HybridModel(sim_options = {
    "name": "msd"
})
n_id = narrow()
train_hybrid(algo, n_id)
```

```
Enter your maximum price in dollars.
 100
```

```
Enter your maximum price in dollars.
 100
Enter some tags you like. Seperate tags by commas.
 Stealth
Do you want to be able to play it on Windows? Enter 1 for yes and 0 for doesn't matter
 1
Do you want to be able to play it on Linux? Enter 1 for yes and 0 for doesn't matter
 1
Do you want to be able to play it on MacOS? Enter 1 for yes and 0 for doesn't matter
 0
Do you want to be able to play it on SteamDeck? Enter 1 for yes and 0 for doesn't matter
 0
           app_id  helpful  funny        date  is_recommended  hours  user_id  \
11257581   512900       30      0  2022-06-11            True  117.0  4045888
1095926    218620        0      0  2016-04-14            True  220.1  1913978
10708984  1545990        0      0  2022-11-14            True    7.3   461206
4750630    218620       13      0  2016-11-24            True  764.7  6098857
3987452    218620        2      0  2019-09-06            True   48.4  3725012
...           ...      ...    ...         ...             ...    ...      ...
2025139    218620        0      0  2015-08-11            True  252.9  4915124
7393959    412020        2      0  2020-07-27            True    3.1  3725434
8929572    238320        0      0  2020-11-14            True    1.5  3840238
2812591    218620       76      0  2015-10-15           False  250.7  4017204
3744710    218620        3      0  2014-02-27            True  385.2   566826

          review_id rating
11257581   11257581      5
1095926     1095926      5
10708984   10708984      1
4750630     4750630      5
3987452     3987452      5
...             ...    ...
2025139     2025139      5
7393959     7393959      2
8929572     8929572      1
2812591     2812591      2
3744710     3744710      5

[1575 rows x 9 columns]
Computing the msd similarity matrix...
Done computing similarity matrix.
```

```
con1, con2 = read_item_names(games.loc[games.app_id.isin(n_id)])
seed = random.choice(list(n_id))
y = get_k_nearest_apps(algo, games.loc[games.app_id.isin(n_id)], con1[seed])
for i in y[0]:
    print(con1[i])
print("Enter a list of numbers which you liked!")
z = list(map(int, input().split(',')))
o = []
c = 0
y = get_k_nearest_apps(algo, games.loc[games.app_id.isin(n_id)], con1[seed])
for i in y[0]:
    #print(c, i)
    if c in z:
        o.append(i)
    c += 1
for i in o:
    algo.add_liked(i)
con1, con2 = read_item_names(games.loc[games.app_id.isin(n_id)])
print("You final recommendations are: ")
```

```
    algo.add_liked(i)
con1, con2 = read_item_names(games.loc[games.app_id.isin(n_id)])
print("You final recommendations are: ")
for i in get_final_rec(algo, n_id):
    if i not in o:
        print(con1[i])
```

```
PAYDAY 2
Alien: Isolation
Outlast
Enter a list of numbers which you liked!
 1
You final recommendations are:
Metro Exodus
Penumbra Overture
Oddworld: New 'n' Tasty
```

12. CONCLUSION

In conclusion, our analysis of the Steam game dataset has led to the implementation of several recommendation models that can be used to personalize game recommendations for users. We started by performing exploratory data analysis on the dataset, which helped us understand the distribution of features and identify potential issues in the data. We then performed pre-processing on the dataset, which included cleaning and transforming the data into a format that could be used for analysis.

Next, we implemented three different recommendation models on the dataset: content-based filtering, collaborative-based filtering, and a hybrid recommender system. The content-based filtering model used the attributes of games to recommend similar games to users, while the collaborative-based filtering model used user-item interactions to make recommendations. The hybrid recommender system combined the strengths of both models to provide more personalized recommendations.

Overall, our analysis and implementation of different recommender models on the Steam game dataset have shown the potential of using recommendation systems to personalize game recommendations for users. Our models can help improve the user experience on the platform by providing tailored recommendations and promoting game discovery. However, there is still room for improvement in terms of increasing the diversity of recommendations and addressing the cold-start problem for new users.