# SC310005 Artificial Intelligence

Lecture 9: Apriori Association Rules and Recommender Systems

teerapong.pa@chula.ac.th

# Reference


- https://www.r-bloggers.com/2019/05/practical-introduction-to-market-basket-analysis-asociation-rules/
- https://towardsdatascience.com/a-gentle-introduction-on-market-basket-analysis-association-rules-fa4b986a40ce
- https://bdi.or.th/big-data-101/what-is-association-rule/
- https://analyticsarora.com/what-is-association-rule-learning-machine-learning-interview-questions/
- https://rasbt.github.io/mlxtend/api_modules/mlxtend.frequent_patterns/association_rules/
- https://www.analyticsvidhya.com/blog/2021/07/recommendation-system-understanding-the-basic-concepts/

# What Is Association Rule Analysis ?

dataaspirant.com

# Market Basket Analysis – Association Rules

Market basket analysis uses association rule mining under the hood to identify products frequently bought together. Before we get into the nitty gritty of market basket analysis, let us get a basic understanding of association rule mining.

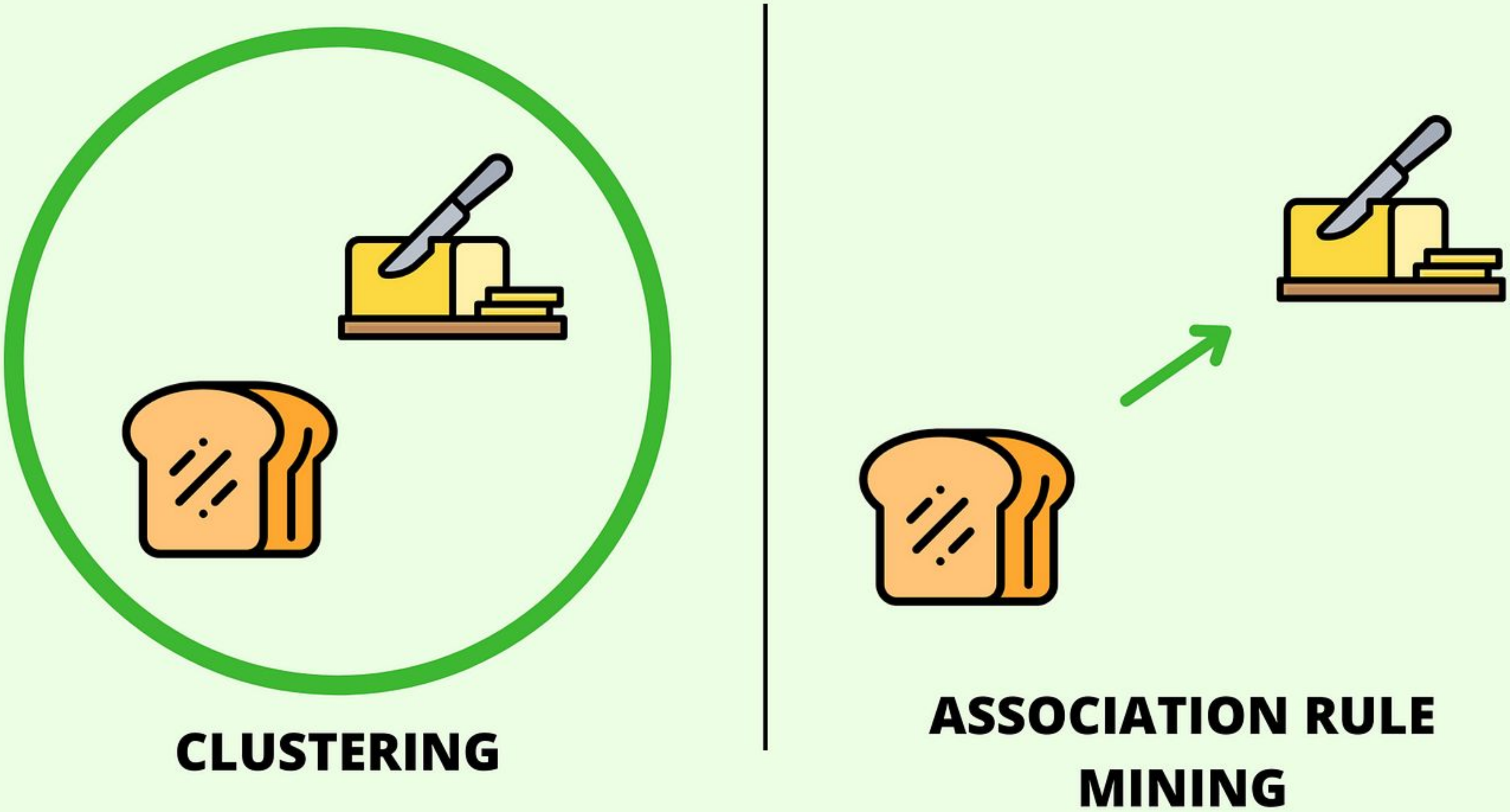
It finds association between different objects in a set.

In the case of market basket analysis, the objects are the products purchased by a customer and the set is the transaction. In short, market basket analysis

- is a unsupervised data mining technique
- that uncovers products frequently bought together
- and creates if-then scenario rules

# Market Basket Analysis – Association Rules

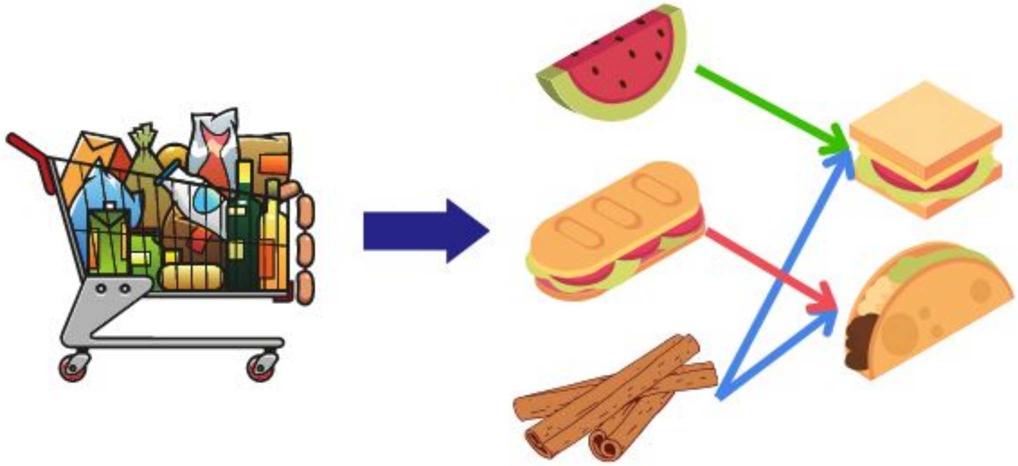
- is an association rule learning that is a rule-based machine learning method for discovering interesting relations between variables in large databases.
- So, Market basket analysis is a technique used by companies to find associations between products so that they can generate more revenue by presenting various related products to the customer


CLUSTERING
ASSOCIATION RULE MINING

# Clustering vs Association Rule Mining

Clustering techniques calculate clusters based on similarities whereas Association rule mining finds associations based on co-occurrences.

# Association Rule Learning



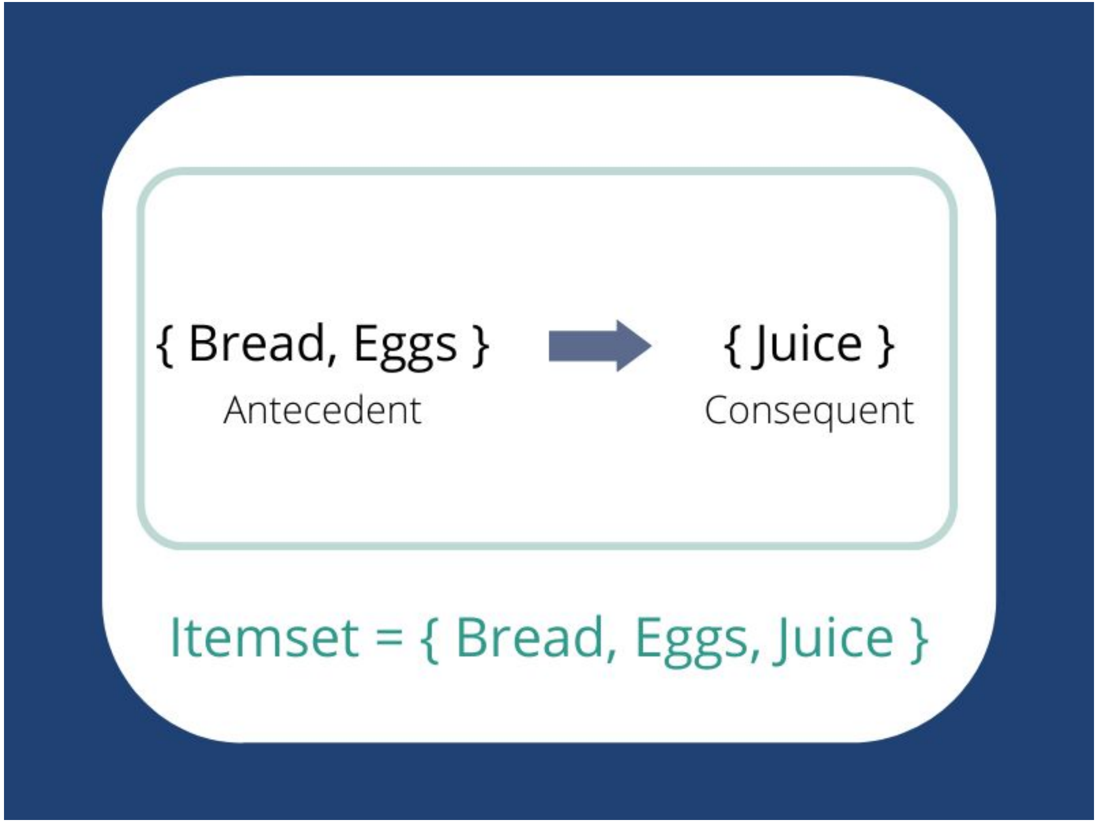
*"93% of people who purchased item A also purchased item B"*

# Why ?

Market basket analysis creates actionable insights for:

- designing store layout
- online recommendation engines
- targeted marketing campaign/sales promotion/email campaign
- cross/up selling
- catalogue design

# How do Association Rule Learning Algorithms work?



# How do Association Rule Learning Algorithms work?

Association rule learning algorithms work like conditional statements

(ex, if A then B). In this case, A is called the antecedent while B is called the consequent.

A, or the antecedent can be an item in your data while B is the result of the combination of antecedent(s).

B could take the form of an action, such as signaling a customer is likely to be a repeat buyer, or B could be another item in the dataset.

The algorithm differentiates random transactions from important patterns by using metrics like support, confidence and lift.

# How do Association Rule Learning Algorithms work?

- **Support** is the number of times an item was present in a transaction.
- **Confidence** is the number of times an item has been combined in a transaction.
- **Lift** is used to compare the number of times a rule was supposed to be obeyed to the number of times it actually obeyed.

Association rule learning algorithms work in two simple steps.

First, all frequent items in the dataset are found. Then, association rules from the frequent itemsets are generated using the support, confidence and lift threshold.

# What is the **Apriori** algorithm?

Apriori is an algorithm that finds all frequent items set in a dataset.

It finds items that are frequently transacted together whose support and confidence are above the minimum threshold.

In scenarios where there are so many items, Apriori helps with defining the rules for these items.

$$Rule:\ X \Rightarrow Y$$

$$Support = \frac{frq(X,Y)}{N}$$

$$Confidence = \frac{frq(X,Y)}{frq(X)}$$

$$Lift = \frac{Support}{Supp(X) \times Supp(Y)}$$


| Rule | Support | Confidence | Lift |
|---|---|---|---|
| $A \Rightarrow D$ | 2/5 | 2/3 | 10/9 |
| $C \Rightarrow A$ | 2/5 | 2/4 | 5/6 |
| $A \Rightarrow C$ | 2/5 | 2/3 | 5/6 |
| $B \& C \Rightarrow D$ | 1/5 | 1/3 | 5/9 |


| Rule | Support | Confidence | Lift |
|---|---|---|---|
| $A \Rightarrow D$ | 2/5 | 2/3 | 10/9 |
| $C \Rightarrow A$ | 2/5 | 2/4 | 5/6 |
| $A \Rightarrow C$ | 2/5 | 2/3 | 5/6 |
| $B \& C \Rightarrow D$ | 1/5 | 1/3 | 5/9 |

## Notes

# An example of Association Rules

**Assume there are 100 customers**

**10 of them bought milk, 8 bought butter and 6 bought both of them.**

**bought milk => bought butter**

- **support = P(Milk & Butter) =**
- **confidence = support/P(Butter) =**
- **lift = confidence/P(Milk) =**

Notes

Association Rule

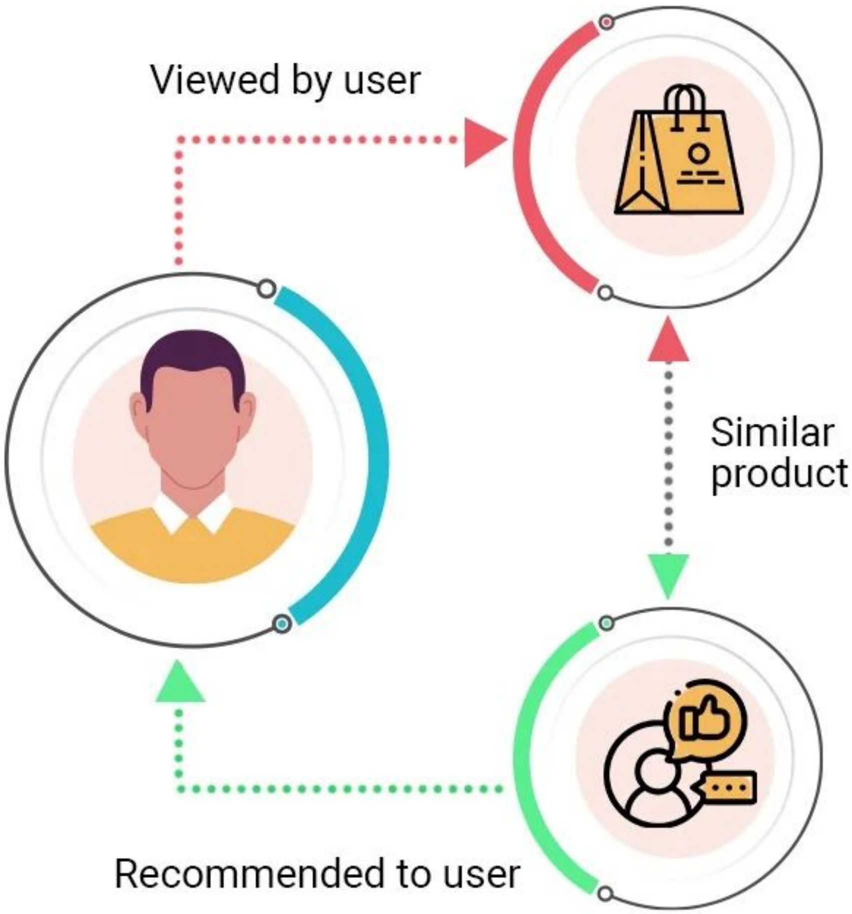
Mobile → Screen Guard

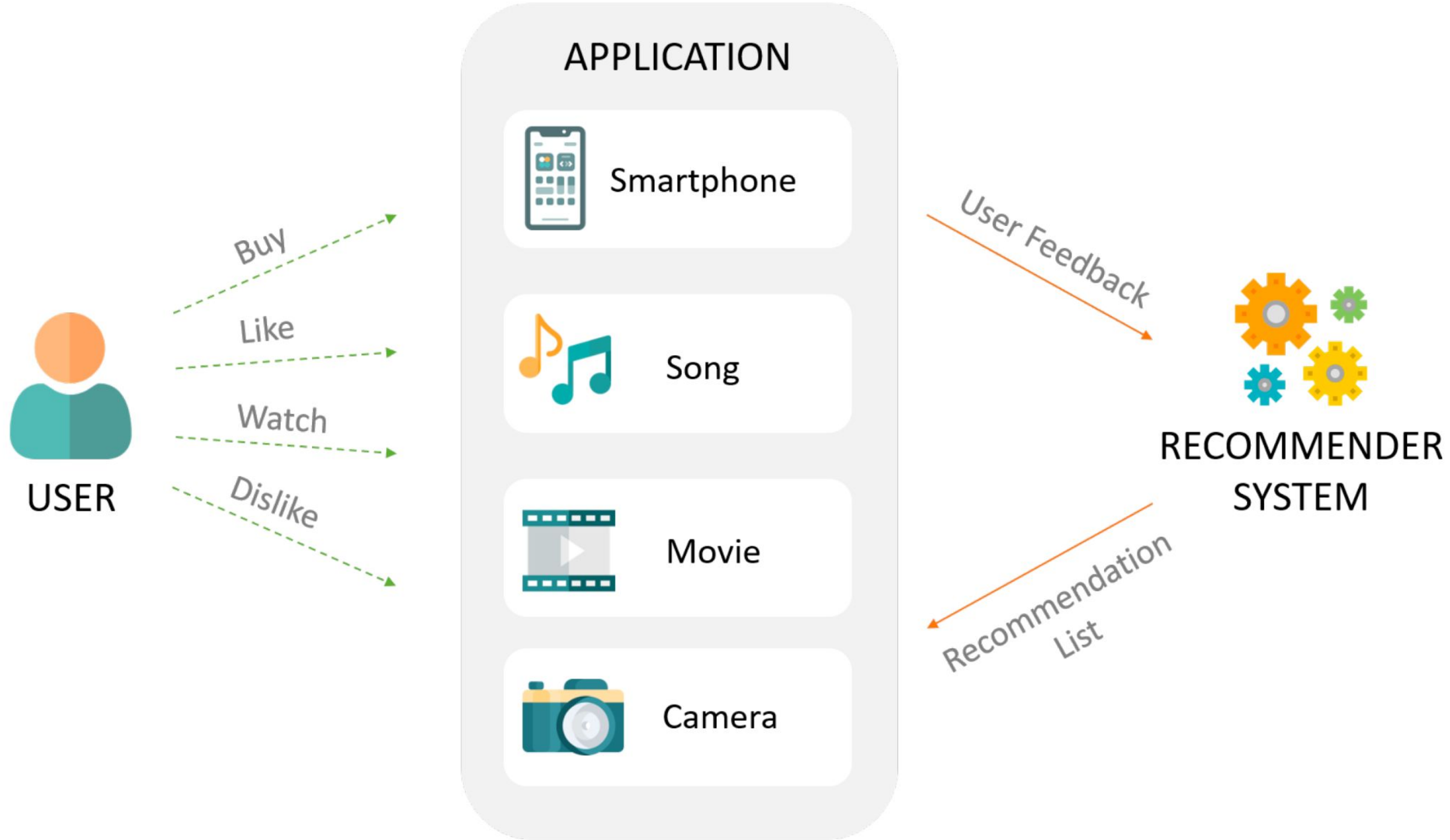
Total Transactions (N): 2000

| Item | Transactions |
|---|---|
| Mobile | 200 |
| Screen Guard | 160 |
| Mobile + Screen Guard | 120 |

Lift = Support (  +  ) / Support (  ) * Support (  )

$$= \frac{0.06}{(0.1 * 0.08)} = 7.5$$


Viewed by user
Similar product
Recommended to user


APPLICATION
Smartphone
Song
Movie
Camera
USER
Buy
Like
Watch
Dislike
User Feedback
RECOMMENDER SYSTEM
Recommendation List

# Recommendation System

A recommendation system is a subclass of Information filtering Systems that seeks to predict the rating or the preference a user might give to an item.

In simple words, it is an algorithm that suggests relevant items to users.

Eg: In the case of Netflix which movie to watch, In the case of e-commerce which product to buy, or In the case of kindle which book to read, etc.

# Use-Cases Of Recommendation System

There are many use-cases of it. Some are

**A. Personalized Content:** Helps to Improve the on-site experience by creating dynamic recommendations for different kinds of audiences like Netflix does.

**B. Better Product search experience:** Helps to categories the product based on their features. Eg: Material, Season, etc.

# Collaborative Based Filtering

Recommending the new items to users based on the interest and preference of other similar users is basically collaborative-based filtering. For eg:- When we shop on Amazon it recommends new products saying

**"Customer who bought this also brought"** as shown below.

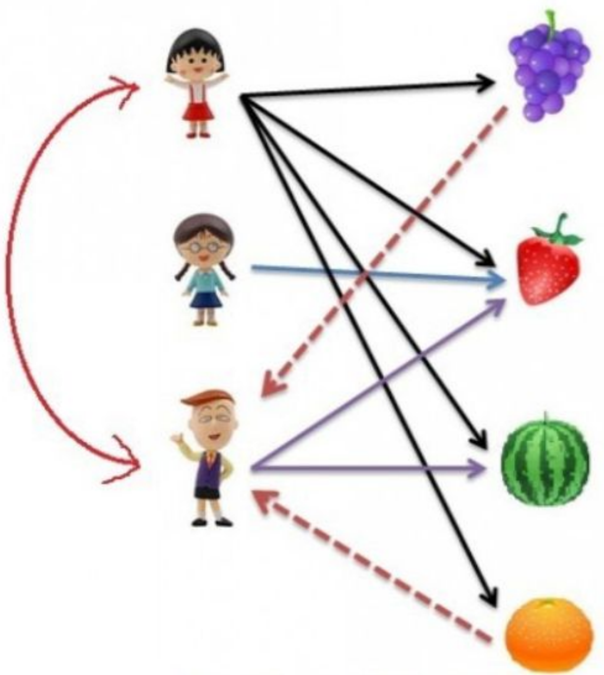

Source: https://www.amazon.com/OnePlus-Glacial-Unlocked-Android-Smartphone/dp/B08723759H/ref=sr_1_3?dchild=1&keywords=mobile&qid=1626099632&sr=8-3

# Collaborative Based Filtering

There are 2 types of collaborative filtering:-

A. User-Based Collaborative Filtering

Rating of the item is done using the rating of neighbouring users. In simple words, It is based on the notion of users’ similarity.



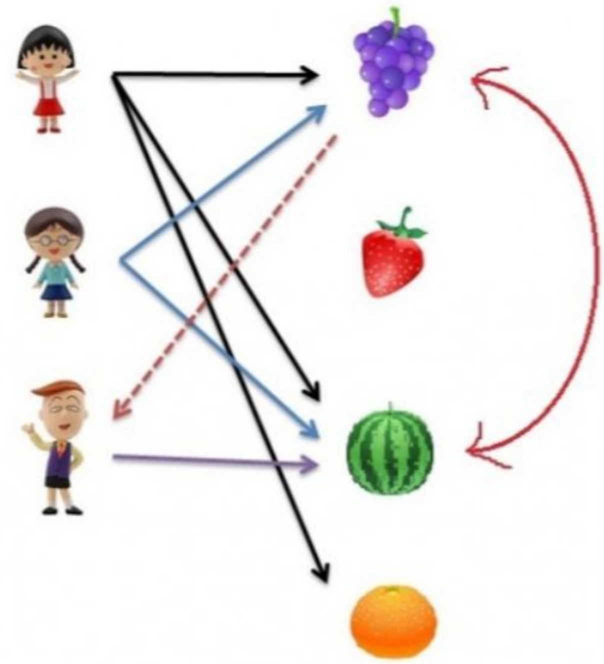
User-based filtering

# Collaborative Based Filtering

There are 2 types of collaborative filtering:-

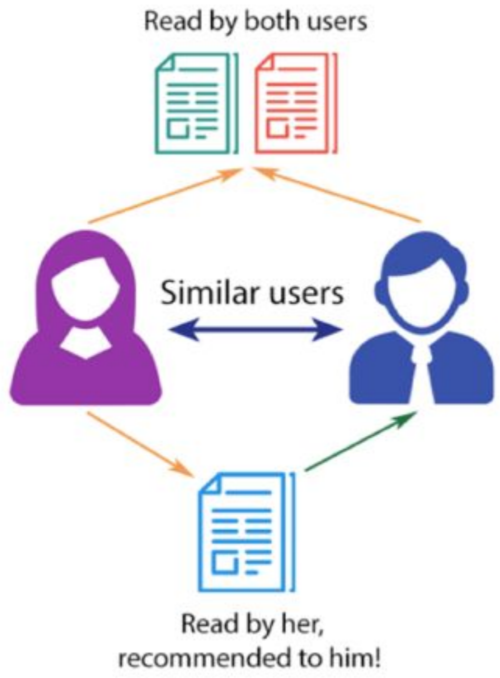
B. Item-Based Collaborative Filtering

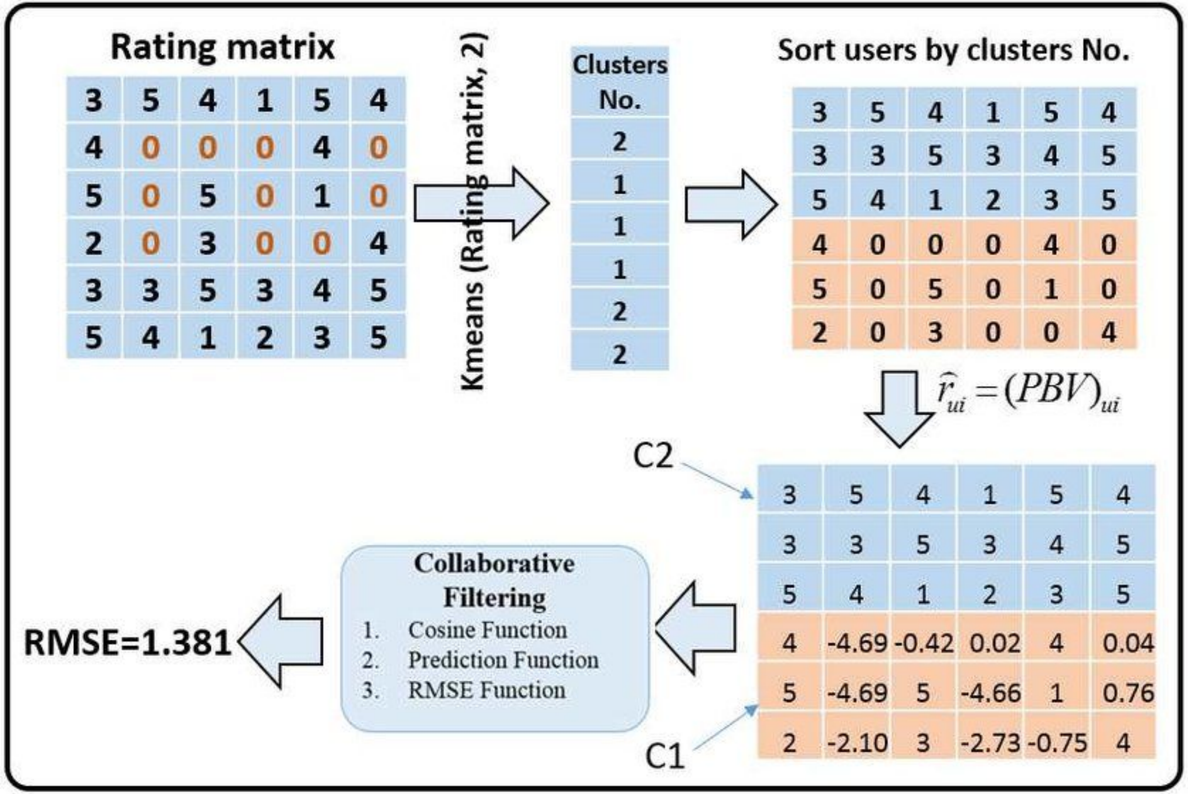
The rating of the item is predicted using the user’s own rating on neighbouring items. In simple words, it is based on the notion of item similarity.



Item-based filtering

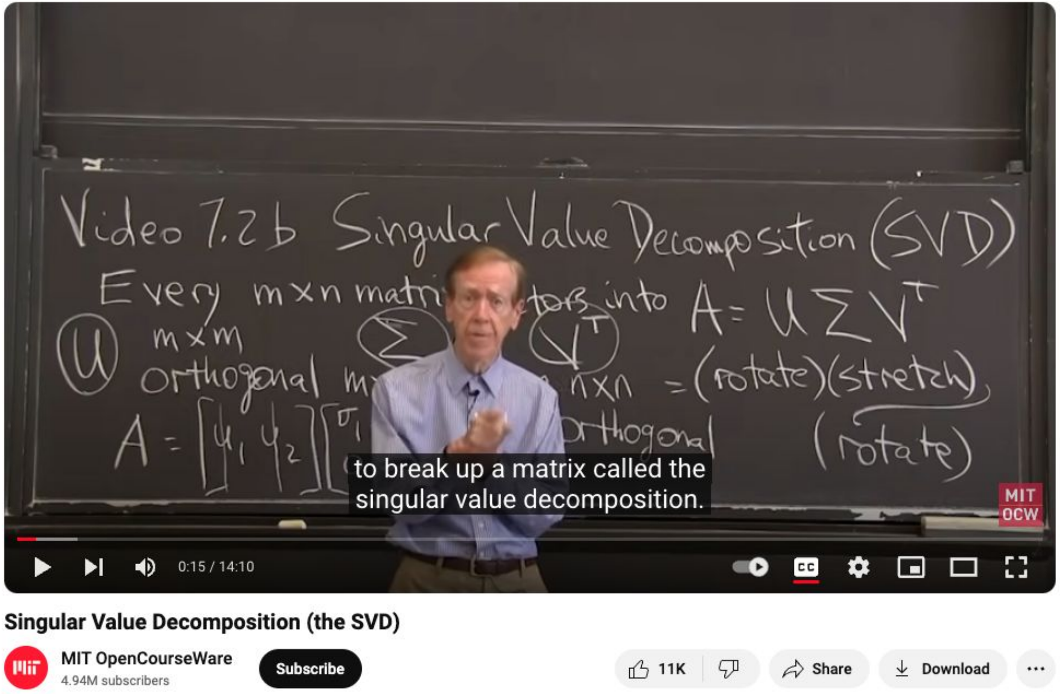
# Using Singular Value Decomposition (SVD)



# Using Singular Value Decomposition (SVD)

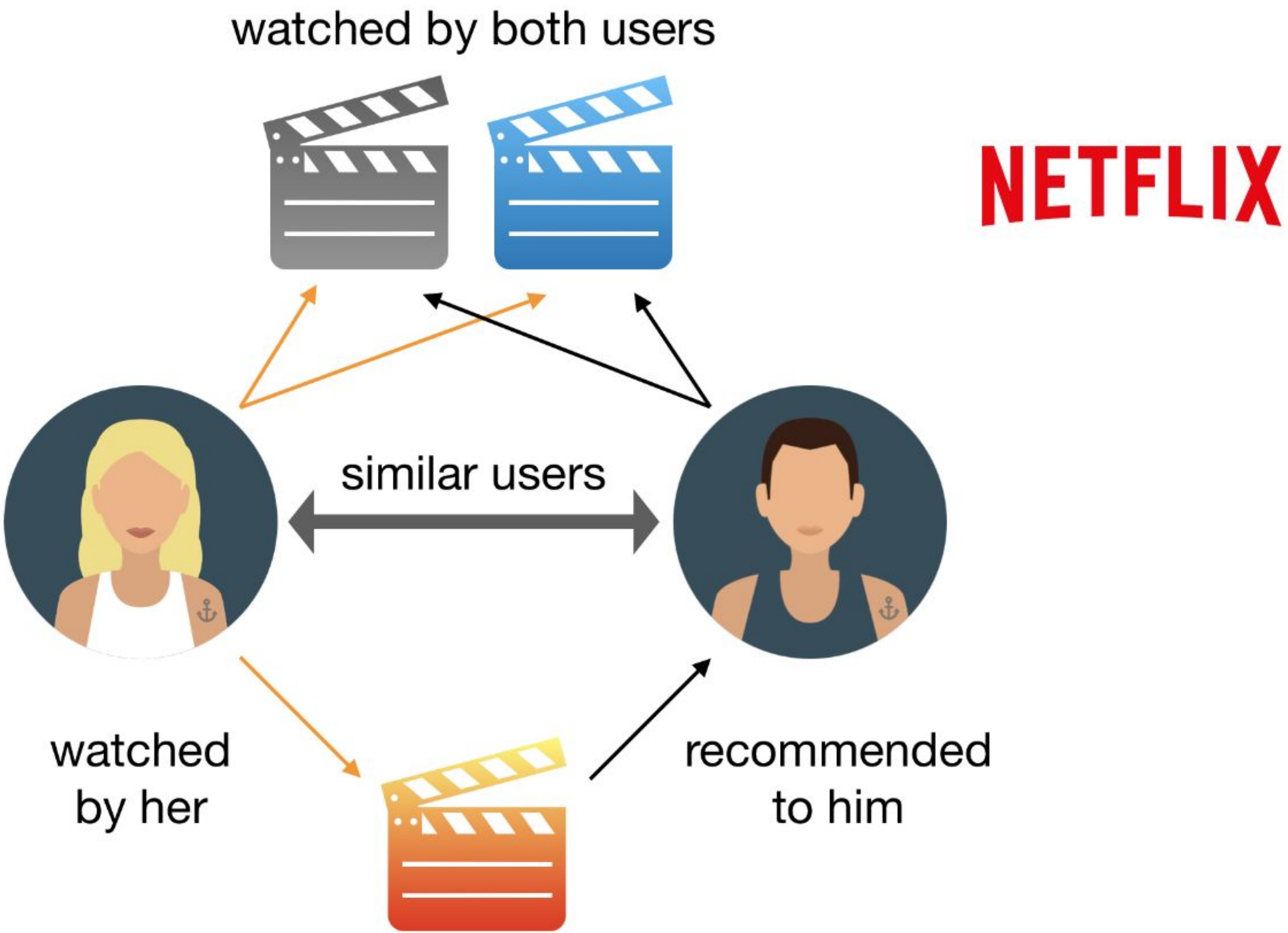
https://www.youtube.com/watch?v=mBcLRGuAFUk




**W. Gilbert Strang**

Strang in 2021

| | |
|---|---|
| **Born** | November 27, 1934 (age 89)<br>Chicago, Illinois |
| **Nationality** | American |
| **Alma mater** | Massachusetts Institute of Technology (BS)<br>Balliol College, Oxford (BA, MA)<br>University of California, Los Angeles (PhD) |
| **Awards** | Chauvenet Prize (1977) |


watched by both users
NETFLIX
similar users
watched
by her
recommended
to him

NETFLIX


Ratings
Searches
Watch History
Nature of Show
Re-watch Program
Device
Credit Calculation
Program Pause Time
NETFLIX
BIG DATA

# Let's Code!

</>

Let's Code This Weekend

[1]

```
!pip install mlxtend  --quiet
!pip install scikit-surprise --quiet
```

[2]

```
!wget https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/marketbasket_dataset.csv
!wget https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/amazon_reviews_dataset.csv
!wget https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/amazon_reviews_dataset_toStudent.csv
```

```
--2024-01-26 10:34:27--  https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/marketbasket_dataset.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.110.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.110.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 710518 (694K) [text/plain]
Saving to: ‘marketbasket_dataset.csv.2’

marketbasket_datase 100%[===================>] 693.87K  --.-KB/s    in 0.04s

2024-01-26 10:34:27 (15.5 MB/s) - ‘marketbasket_dataset.csv.2’ saved [710518/710518]

--2024-01-26 10:34:27--  https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/amazon_reviews_dataset.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.108.133, 185.199.109.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 407164 (398K) [text/plain]
Saving to: ‘amazon_reviews_dataset.csv.2’

amazon_reviews_data 100%[===================>] 397.62K  --.-KB/s    in 0.03s

2024-01-26 10:34:28 (11.2 MB/s) - ‘amazon_reviews_dataset.csv.2’ saved [407164/407164]

--2024-01-26 10:34:28--  https://raw.githubusercontent.com/kaopanboonyuen/SC310005_ArtificialIntelligence_2023s1/main/dataset/amazon_reviews_dataset_toStudent.csv
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 814483 (795K) [text/plain]
Saving to: ‘amazon_reviews_dataset_toStudent.csv.1’

amazon_reviews_data 100%[===================>] 795.39K  --.-KB/s    in 0.05s

2024-01-26 10:34:28 (17.0 MB/s) - ‘amazon_reviews_dataset_toStudent.csv.1’ saved [814483/814483]
```

amazon.com

```
[36] import warnings
     warnings.filterwarnings("ignore")
```

```
[37] import pandas as pd
     from mlxtend.frequent_patterns import apriori
     from mlxtend.frequent_patterns import association_rules

     import matplotlib.pyplot as plt
     import seaborn as sns
```

```
[38] # Load the dataset
     df = pd.read_csv('marketbasket_dataset.csv')
```

```
[39] df.shape
```

```
(21293, 4)
```

```
[40] # Step 2: Remove rows where ITEM=='NONE'
     df = df[df['Item'] != 'NONE']
     df.head()
```

| | Date | Time | Transaction | Item |
|---|---|---|---|---|
| **0** | 2016-10-30 | 09:58:11 | 1 | Bread |
| **1** | 2016-10-30 | 10:05:34 | 2 | Scandinavian |
| **2** | 2016-10-30 | 10:05:34 | 2 | Scandinavian |
| **3** | 2016-10-30 | 10:07:57 | 3 | Hot chocolate |
| **4** | 2016-10-30 | 10:07:57 | 3 | Jam |

Notes

```python
# Step 3: Perform Market Basket Analysis with Apriori Algorithm
# Convert the data into a transaction format
basket = df.groupby(['Transaction', 'Item'])['Item'].count().unstack().reset_index().fillna(0).set_index('Transaction')
basket
```

| Item | Adjustment | Afternoon with the baker | Alfajores | Argentina Night | Art Tray | Bacon | Baguette | Bakewell | Bare Popcorn | Basket | ... | The BART | The Nomad | Tiffin | Toast | Truffles | Tshirt | Valentine's card | Vegan Feast | Vegan mincepie | Victorian Sponge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Transaction** | | | | | | | | | | | | | | | | | | | | | |
| **1** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **2** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **3** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **4** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **5** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9680** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **9681** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **9682** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **9683** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **9684** | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | ... | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

9465 rows × 94 columns

[42]

```python
# Convert the counts into binary values (0 or 1)
basket_sets = basket.applymap(lambda x: 1 if x > 0 else 0)
basket_sets
```

| Item | Adjustment | Afternoon with the baker | Alfajores | Argentina Night | Art Tray | Bacon | Baguette | Bakewell | Bare Popcorn | Basket | ... | The BART | The Nomad | Tiffin | Toast | Truffles | Tshirt | Valentine's card | Vegan Feast | Vegan mincepie | Victorian Sponge |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Transaction** | | | | | | | | | | | | | | | | | | | | | |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **5** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **9680** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9681** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| **9682** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9683** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **9684** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[43]

```
# Find frequent itemsets with minimum support of 0.01
frequent_itemsets = apriori(basket_sets, min_support=0.01, use_colnames=True)
frequent_itemsets
```

| | support | itemsets |
|---|---|---|
| 0 | 0.036344 | (Alfajores) |
| 1 | 0.016059 | (Baguette) |
| 2 | 0.327205 | (Bread) |
| 3 | 0.040042 | (Brownie) |
| 4 | 0.103856 | (Cake) |
| ... | ... | ... |
| 56 | 0.023666 | (Toast, Coffee) |
| 57 | 0.014369 | (Sandwich, Tea) |
| 58 | 0.010037 | (Coffee, Bread, Cake) |
| 59 | 0.011199 | (Pastry, Coffee, Bread) |
| 60 | 0.010037 | (Cake, Coffee, Tea) |

61 rows × 2 columns

[44]

```
# Extract association rules
association_rules_df = association_rules(frequent_itemsets, metric="lift", min_threshold=0.5)
association_rules_df
```

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (Bread) | (Alfajores) | 0.327205 | 0.036344 | 0.010354 | 0.031644 | 0.870657 | -0.001538 | 0.995145 | -0.180870 |
| 1 | (Alfajores) | (Bread) | 0.036344 | 0.327205 | 0.010354 | 0.284884 | 0.870657 | -0.001538 | 0.940818 | -0.133570 |
| 2 | (Coffee) | (Alfajores) | 0.478394 | 0.036344 | 0.019651 | 0.041078 | 1.130235 | 0.002264 | 1.004936 | 0.220910 |
| 3 | (Alfajores) | (Coffee) | 0.036344 | 0.478394 | 0.019651 | 0.540698 | 1.130235 | 0.002264 | 1.135648 | 0.119574 |
| 4 | (Bread) | (Brownie) | 0.327205 | 0.040042 | 0.010777 | 0.032935 | 0.822508 | -0.002326 | 0.992651 | -0.242849 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 69 | (Cake, Tea) | (Coffee) | 0.023772 | 0.478394 | 0.010037 | 0.422222 | 0.882582 | -0.001335 | 0.902779 | -0.119934 |
| 70 | (Coffee, Tea) | (Cake) | 0.049868 | 0.103856 | 0.010037 | 0.201271 | 1.937977 | 0.004858 | 1.121962 | 0.509401 |
| 71 | (Cake) | (Coffee, Tea) | 0.103856 | 0.049868 | 0.010037 | 0.096643 | 1.937977 | 0.004858 | 1.051779 | 0.540090 |
| 72 | (Coffee) | (Cake, Tea) | 0.478394 | 0.023772 | 0.010037 | 0.020981 | 0.882582 | -0.001335 | 0.997149 | -0.203223 |
| 73 | (Tea) | (Cake, Coffee) | 0.142631 | 0.054728 | 0.010037 | 0.070370 | 1.285822 | 0.002231 | 1.016827 | 0.259266 |

74 rows × 10 columns

[45]

```python
# Step 5: Filter association rules with two items in antecedents and consequents
# Filter association rules where both antecedents and consequents contain two items each
filtered_rules = association_rules_df[
    (association_rules_df['antecedents'].apply(lambda x: len(x)) == 2) &
    (association_rules_df['consequents'].apply(lambda x: len(x)) == 1)
]

# Step 6: Analyze and Interpret the Results
print("Filtered Association Rules:")
filtered_rules
```

Filtered Association Rules:

| | antecedents | consequents | antecedent support | consequent support | support | confidence | lift | leverage | conviction | zhangs_metric |
|---|---|---|---|---|---|---|---|---|---|---|
| **56** | (Coffee, Bread) | (Cake) | 0.090016 | 0.103856 | 0.010037 | 0.111502 | 1.073621 | 0.000688 | 1.008606 | 0.075356 |
| **57** | (Cake, Coffee) | (Bread) | 0.054728 | 0.327205 | 0.010037 | 0.183398 | 0.560497 | -0.007870 | 0.823895 | -0.453411 |
| **58** | (Cake, Bread) | (Coffee) | 0.023349 | 0.478394 | 0.010037 | 0.429864 | 0.898557 | -0.001133 | 0.914880 | -0.103617 |
| **62** | (Pastry, Coffee) | (Bread) | 0.047544 | 0.327205 | 0.011199 | 0.235556 | 0.719901 | -0.004357 | 0.880109 | -0.290026 |
| **63** | (Pastry, Bread) | (Coffee) | 0.029160 | 0.478394 | 0.011199 | 0.384058 | 0.802807 | -0.002751 | 0.846843 | -0.201920 |
| **64** | (Coffee, Bread) | (Pastry) | 0.090016 | 0.086107 | 0.011199 | 0.124413 | 1.444872 | 0.003448 | 1.043749 | 0.338354 |
| **68** | (Cake, Coffee) | (Tea) | 0.054728 | 0.142631 | 0.010037 | 0.183398 | 1.285822 | 0.002231 | 1.049923 | 0.235157 |
| **69** | (Cake, Tea) | (Coffee) | 0.023772 | 0.478394 | 0.010037 | 0.422222 | 0.882582 | -0.001335 | 0.902779 | -0.119934 |
| **70** | (Coffee, Tea) | (Cake) | 0.049868 | 0.103856 | 0.010037 | 0.201271 | 1.937977 | 0.004858 | 1.121962 | 0.509401 |

[46]

```python
# Calculate Support
# Calculate support for each item (without using libs)
support = df['Item'].value_counts(normalize=True)
support
```

```
Coffee            0.266787
Bread             0.162140
Tea               0.069976
Cake              0.049983
Pastry            0.041742
                    ...
Bacon             0.000049
Gift voucher      0.000049
Olum & polenta    0.000049
Raw bars          0.000049
Polenta           0.000049
Name: Item, Length: 94, dtype: float64
```

[47]

```python
# Calculate Lift (without using libs)

# Calculate lift between pairs of items
unique_items = df['Item'].unique()
lift_data = []

for i in range(len(unique_items)):
    for j in range(i+1, len(unique_items)):
        item1 = unique_items[i]
        item2 = unique_items[j]

        # Calculate support for item1 and item2
        support_item1 = support[item1]
        support_item2 = support[item2]

        # Calculate support for item1 and item2 occurring together
        support_both = len(df[(df['Item'] == item1) & (df['Item'] == item2)]) / len(df)

        # Calculate lift
        lift = support_both / (support_item1 * support_item2)

        lift_data.append({'Item1': item1, 'Item2': item2, 'Support_Item1': support_item1,
                          'Support_Item2': support_item2, 'Support_Both': support_both, 'Lift': lift})
```

[48]

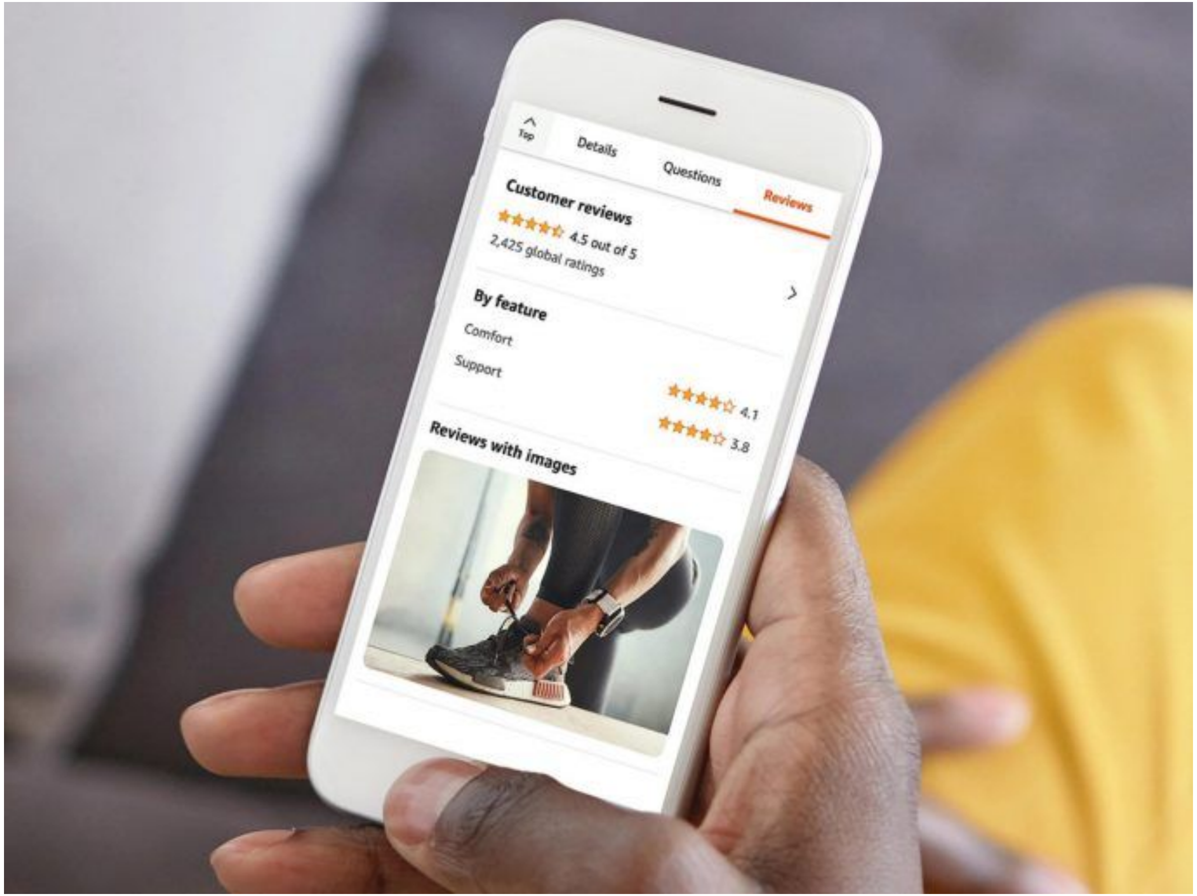
```python
# Convert lift data to DataFrame
lift_df = pd.DataFrame(lift_data)
lift_df
```

| | Item1 | Item2 | Support_Item1 | Support_Item2 | Support_Both | Lift |
|---|---|---|---|---|---|---|
| 0 | Bread | Scandinavian | 0.162140 | 0.013508 | 0.0 | 0.0 |
| 1 | Bread | Hot chocolate | 0.162140 | 0.028771 | 0.0 | 0.0 |
| 2 | Bread | Jam | 0.162140 | 0.007266 | 0.0 | 0.0 |
| 3 | Bread | Cookies | 0.162140 | 0.026332 | 0.0 | 0.0 |
| 4 | Bread | Muffin | 0.162140 | 0.018043 | 0.0 | 0.0 |
| ... | ... | ... | ... | ... | ... | ... |
| 4366 | Cherry me Dried fruit | Raw bars | 0.000146 | 0.000049 | 0.0 | 0.0 |
| 4367 | Cherry me Dried fruit | Tacos/Fajita | 0.000146 | 0.000536 | 0.0 | 0.0 |
| 4368 | Mortimer | Raw bars | 0.000244 | 0.000049 | 0.0 | 0.0 |
| 4369 | Mortimer | Tacos/Fajita | 0.000244 | 0.000536 | 0.0 | 0.0 |
| 4370 | Raw bars | Tacos/Fajita | 0.000049 | 0.000536 | 0.0 | 0.0 |

4371 rows × 6 columns

Notes

amazon.com



Top
Details
Questions
Reviews
Customer reviews
4.5 out of 5
2,425 global ratings
By feature
Comfort
Support
4.1
3.8
Reviews with images

```python
from surprise import Dataset, Reader, SVD
from surprise.model_selection import train_test_split
from surprise import accuracy
```

```python
# Step 1: Load and Prepare Data

# Load the dataset
df = pd.read_csv('amazon_reviews_dataset.csv')
```

```python
df.head()
```

| | userID | productId | Rating | timestamp |
|---|---|---|---|---|
| 0 | A361M14PU2GUEG | B000LRMS66 | 5.0 | 1324944000 |
| 1 | A1YEPFLLH42OU1 | B007WTAJTO | 4.0 | 1354147200 |
| 2 | A1SB9BNNGKNX2Z | B000652M6Y | 5.0 | 1294704000 |
| 3 | A2GOHNFBHUU3UI | B000MWC0IG | 5.0 | 1310428800 |
| 4 | A7QMQBGJ2TCQG | B001GKPOJK | 5.0 | 1272153600 |

```python
df.columns
```

```
Index(['userID', 'productId', 'Rating', 'timestamp'], dtype='object')
```

Notes

[54]
```python
# Assuming the dataset has four columns: 'userId', 'productId', 'Rating', 'timestamp'
reader = Reader(rating_scale=(1, 5))
data = Dataset.load_from_df(df[['userID', 'productId', 'Rating']], reader)
```

[55]
```python
# Step 2: Train-Test Split
trainset, testset = train_test_split(data, test_size=0.2, random_state=2024)
```

[56]
```python
# Step 3: Choose a Recommender Algorithm and Train the Model
# Example: Using the Singular Value Decomposition (SVD) algorithm
model = SVD()
model.fit(trainset)
```

```
<surprise.prediction_algorithms.matrix_factorization.SVD at 0x7f77f5e1e470>
```

[57]
```python
# Step 4: Evaluate the Recommender System
predictions = model.test(testset)
accuracy.rmse(predictions)
```

```
RMSE: 1.0311
1.0311149980732
```

```python
df[df['userID']=='A7QMQBGJ2TCQG']
```

| | userID | productId | Rating | timestamp |
|---|---|---|---|---|
| 4 | A7QMQBGJ2TCQG | B001GKPOJK | 5.0 | 1272153600 |
| 185 | A7QMQBGJ2TCQG | B000VS8GSE | 5.0 | 1271894400 |
| 821 | A7QMQBGJ2TCQG | B000P9ELC4 | 5.0 | 1210723200 |
| 990 | A7QMQBGJ2TCQG | B001FA0FTK | 5.0 | 1248566400 |
| 2232 | A7QMQBGJ2TCQG | B00009R8RS | 5.0 | 1213833600 |
| 2318 | A7QMQBGJ2TCQG | B0011N180Q | 5.0 | 1210723200 |
| 3587 | A7QMQBGJ2TCQG | B002U6KT8U | 5.0 | 1271462400 |
| 6191 | A7QMQBGJ2TCQG | B000J509CK | 5.0 | 1205107200 |
| 6641 | A7QMQBGJ2TCQG | B000HJPK2C | 5.0 | 1288569600 |
| 8414 | A7QMQBGJ2TCQG | B000NZKX4K | 5.0 | 1201564800 |
| 8488 | A7QMQBGJ2TCQG | B000BYCKU8 | 5.0 | 1205107200 |
| 9217 | A7QMQBGJ2TCQG | B001D7REIK | 1.0 | 1259798400 |
| 9682 | A7QMQBGJ2TCQG | B00004ZCJE | 5.0 | 1200268800 |

Notes

```
[59]
# Step 5: Generate Recommendations
# Let's generate recommendations for a specific user (e.g., userId = 1)
user_id = 'A7QMQBGJ2TCQG'
user_products = df[df['userID'] == user_id]['productId'].tolist()
user_products
```

```
['B001GKPOJK',
 'B000VS8GSE',
 'B000P9ELC4',
 'B001FA0FTK',
 'B00009R8RS',
 'B0011N180Q',
 'B002U6KT8U',
 'B000J509CK',
 'B000HJPK2C',
 'B000NZKX4K',
 'B000BYCKU8',
 'B001D7REIK',
 'B00004ZCJE']
```

```
[60]
already_reviewed = {product: rating for (user, product, rating) in
                    df[df['userID'] == user_id][['userID', 'productId', 'Rating']].values}
already_reviewed
```

```
{'B001GKPOJK': 5.0,
 'B000VS8GSE': 5.0,
 'B000P9ELC4': 5.0,
 'B001FA0FTK': 5.0,
 'B00009R8RS': 5.0,
 'B0011N180Q': 5.0,
 'B002U6KT8U': 5.0,
 'B000J509CK': 5.0,
 'B000HJPK2C': 5.0,
 'B000NZKX4K': 5.0,
 'B000BYCKU8': 5.0,
 'B001D7REIK': 1.0,
 'B00004ZCJE': 5.0}
```

Notes

[61]

```python
# Generate recommendations for the user
recommendations = {}
for product_id in df['productId'].unique():
    if product_id not in user_products:
        product_rating = model.predict(user_id, product_id).est
        recommendations[product_id] = product_rating
```

[62]

```python
# Get the top N recommendations
top_recommendations = sorted(recommendations.items(), key=lambda x: x[1], reverse=True)[:10]
top_recommendations
```

```
[('B000N99BBC', 5),
 ('B009JBFOZW', 4.99767846125369),
 ('B00829THK0', 4.984688265783457),
 ('B000EVSLRO', 4.948017716606849),
 ('B000BQ7GW8', 4.947694302761051),
 ('B007WTAJTO', 4.945560891166697),
 ('B00EMHVVNM', 4.93783389499693),
 ('B0079M711S', 4.9357330601513585),
 ('B00829TIEK', 4.935434233789874),
 ('B009YT6PPC', 4.928635402124242)]
```

```python
print("Top Recommendations for User", user_id)
for product_id, rating in top_recommendations:
    print("Product ID:", product_id, "Rating:", rating)
```

```
Top Recommendations for User A7QMQBGJ2TCQG
Product ID: B00AZFMORW Rating: 3.79830527450249
Product ID: B002SQK2F2 Rating: 3.8785099459668455
Product ID: B003VYH1UE Rating: 3.896061271368228
Product ID: B004YIZWXO Rating: 3.921719057114392
Product ID: B00009RU8K Rating: 3.930042996241758
Product ID: B00AVST3HC Rating: 3.986037354412681
Product ID: B0037N9922 Rating: 3.9952031082028565
Product ID: B0038JE07O Rating: 3.9952656188273226
Product ID: B009LPV766 Rating: 3.99864311810838
Product ID: B00904JILO Rating: 4.00371229030489
```

Notes

# Week 9: Assignment

## Build a recommender system using SVD techniques.

This assignment aims to build a recommender system using the Amazon Review Dataset and analyze the recommendations made for specific users.