Recommender Systems

Recommender systems are systems that are designed to recommend things to the user based on many different factors. These systems predict the most likely product that the users are most likely to purchase and are of interest to.

Companies like Netflix, Amazon, Instagram, etc. use recommender systems to help their users to identify the correct product or movies for them.

**Why recommender systems?**

* Increase in revenue based on personalization.
* Better user experience.
* More time spent on the platform
* Help websites improve user engagement.

**Recommender systems applications**

* Netflix to recommend movies
* E-commerce websites to recommend the products.
* Social media platforms to recommend feeds/ blogs/news/songs…etc. Ex: Instagram, Facebook.
* Food recommendations by Zomato, Swiggy.
* Songs recommendations by Spotify and Wynk music.
* Dating apps recommend people.

A few examples of how the recommendation system works

* Amazon Recommender System
  + Amazon sells virtually all categories of products such as books, CDs, software, electronics, and so on.
  + The recommendations on Amazon are provided on the basis of explicitly provided

ratings, buying behavior, and browsing behavior.

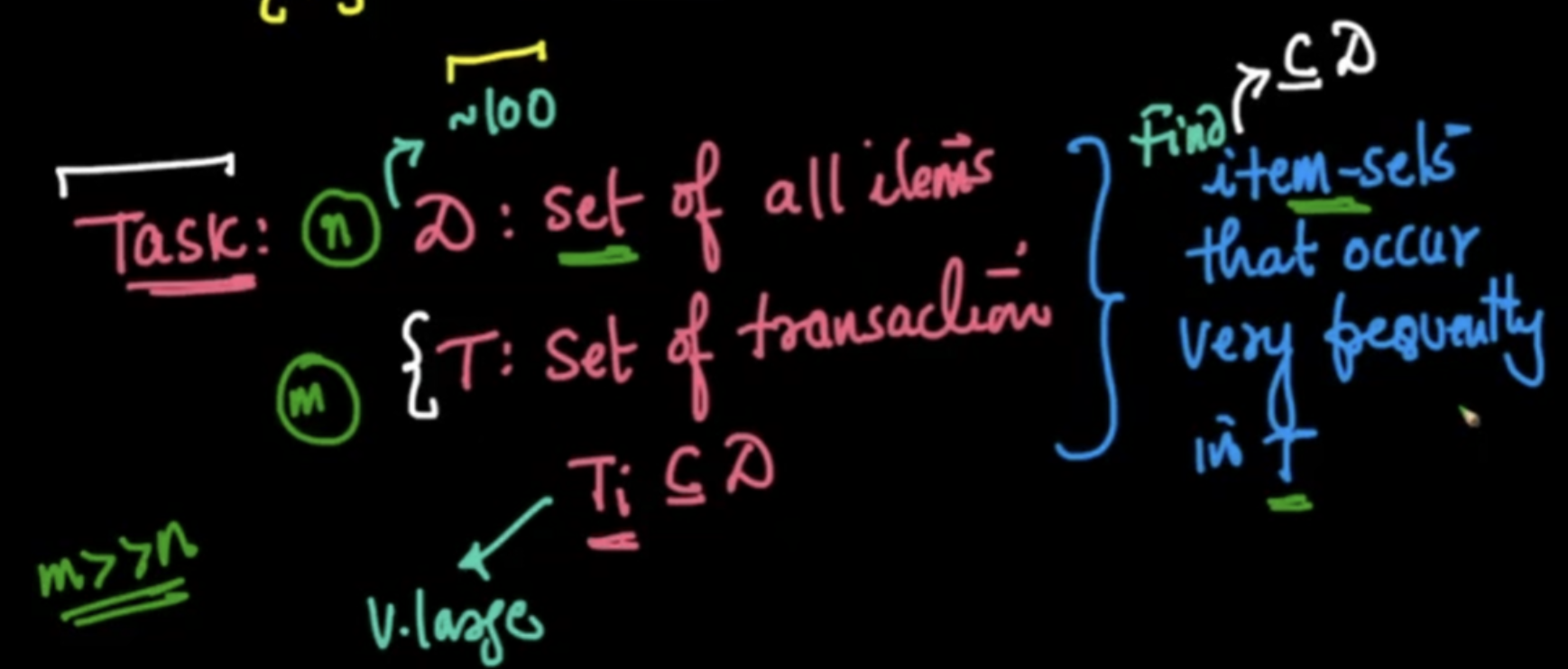
* + The ratings on Amazon are speciﬁed on a 5-point scale, with the lowest rating being 1-star, and the highest rating being 5-star.
  + The customer-speciﬁc buying and browsing data can be easily collected when users are logged in with an account authentication mechanism supported by Amazon.
  + Recommendations are also provided to users on the main Web page of the site, whenever they log into their accounts.
  + Based on the ratings given for each item amazon finds the similarity between items and recommends the items using collaborative filtering.
* Netflix Movie Recommender System
  + The recommendations system estimates the probability of a user watching a particular title based on the following factors:
    - Viewer interactions with Netflix services like viewer ratings, viewing history, etc.
    - Information about the categories, year of release, title, genres, and more.
    - Other viewers with similar watching preferences and tastes.
    - Time duration of a viewer watching a show
    - The device on which a viewer is watching.
    - The time of the day a viewer watches -This is because Netflix has the data that there is different viewing behavior based on the time of the day, the day of the week, the location, and the device on which a show or movie is viewed.
  + For every new subscriber, Netflix asks them to choose titles they would like to watch. These titles are used as the first step for personalized recommendations.
  + Later as viewers continue to watch over time the recommendations are powered by the titles they watched more recently along with other factors mentioned above.
  + To recommend titles for the users Netflix uses a various number of algorithms along with a content-based recommendation system.
* Google News Recommender System
  + The Google News Recommender system is able to recommend news to users based on their history of clicks. The clicks are associated with speciﬁc users based on identiﬁcation mechanisms enabled by Gmail accounts.
  + The act of a user clicking on a news article can be viewed as a positive rating for that article.
  + Collaborative recommendation algorithms are applied to the collected ratings so that inferences can be made about the personalized articles for speciﬁc users.

**Types of recommender systems**

1. Apriori Algorithm
2. Content-based filtering system.
3. Collaborative-based filtering system.
4. Similarity-based filtering system.
5. Matrix factorization
6. Popularity-based recommender system.
7. Regression-based recommender system
8. Group-based recommender system

## **Market-Basket Analysis**

* Market basket analysis is used to analyze the combination of products that have been bought together.
* This is a technique used for purchases done by a customer. This identifies the pattern of frequent purchases of items by customers.
* This analysis can help to promote deals, offers, sale by the companies, and data mining techniques helps to achieve this analysis task.
* For Example, At huge superstores like Walmart, we will have products across many categories: Daliy essentials, Food products (like Milk, Butter, jam, bread, etc), Beauty Products, Toys, etc.
* Even then, we're looking at a **few hundreds** of products, or at the max, a few thousand, as opposed to the world of e-commerce where there may be lakhs or millions of products.
  + - Let n be the Total No of distinct products
    - Let's define D as the set of all the products we have, then, D={1, 2, 3, ..., n}
* Consider a customer that is done selecting the items they want, and are proceeding towards the billing counter. They present their **basket** full of products and the cashier scans the product bought and its quantity. This is called a **transaction**, denoted by T.
  + No of transactions m >> No of distinct items n



* This technique of analyzing transaction data to give recommendations to the customer is called **Market Basket Analysis**.
* Other applications of Market Basket Analysis:
  + **Bio-informatics**
    - If two chemical components c1 and c3 occur frequently within different proteins, then we can find out that perhaps there is some relation between these components.
    - If two gene sequences ATTC and AGTC occur frequently, in the sequence of some mammals, then we can find out that perhaps there is some relation between them.
  + **Medicine**
    - If we find that according to a doctor's prescription, medicines m1, m2, and m3 are being prescribed frequently, then it means that together they form some combination drug, which can cure a certain ailment.
  + **Finding similar webpages/web usage mining**
    - If in a single session many users are visiting the same webpages (w1, w2, w3), then perhaps they are related in nature

**Apriori Algorithm**

* Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. we can say that the apriori algorithm is an association rule learning that analyzes that people who bought product A also bought product B.
* It is a Frequency-based algorithm. Generally, the apriori algorithm operates on a database containing a huge number of transactions.

Ex: People who bought iPhones also bought Airpods.

**Examples** where the apriori algorithm can be used:

* Telecommunications
* Banking / Insurance
* Medical
* E-Commerce
* Retail

**To implement Apriori Algorithm in code:**

* We have a built-in function that implements apriori for us, under the `mlxtend.frequent\_patterns` library.
* We need to specify a minimum support threshold value, as a parameter of this function.
* Since our column names represent the items, we use `use\_colnames=True`
* This will give us the most frequent item sets, so we sort them in descending order also, using `sort\_values()`
* Using the Apriori algorithm, we get a sense of which item-sets are frequent.
* We have another tool in the Market Basket Analysis, called the **Association rule** using which we can find relations between these frequent item sets.

### **Association rules**

### Association Rules are widely used to analyze retail basket or transaction data and are intended to identify strong rules discovered in transaction data using measures of patterns, based on the concept of strong rules.

### There are various metrics in place to help us understand the strength of the association between an antecedent and consequent:

* **Working of association rule:**
  + Consider we have our set of items: D ={ 1, 2, 3, ..., n}
  + Let's define X and Y as another set of items, as follows:
    - X= {1,2,3}
    - Y = {4,6}
  + If the item-set {1, 2, 3, 4, 6} is a **frequent itemset**, then according to the **Association Rule** of Market Basket Analysis, we can say that people who buy X, have a very high likelihood to buy Y also.
  + This can be written as X -> Y and **read as "If X, then Y".**
* For example, **If** a person buys beer, **then** there is a high tendency of buying diapers.
* X → Y is not the same as Y → X
* When we say, People buying beer, have a high tendency of buying diapers, that **does not imply** that people buying diapers have a high tendency of buying beers.
* To set these apart, we have the following terminologies in place:
  + **Antecedent (If):** The items on the LEFT ie., the item which the customer buys.
  + **Consequent (Then):** The items on the RIGHT ie., the item which the customer follows to buy.
* There are a couple of different metrics to know how strongly two item sets are associated:
  + Support
  + Confidence
  + Lift
  + Leverage
  + Conviction

**1. Support:**

It is calculated to check how popular a given item is. It is measured by the proportion of transactions in which an item set appears. It is also used to measure abundance or frequency.

Drawback: If ‘x’ is very popular then many items will have high support w.r.t to x.

**2. Confidence:**

It is calculated to check how likely item X is purchased when item Y is purchased. This is measured by the proportion of transactions with item X, in which item Y also appears.

Drawback: If ‘y’ is very popular then it will have high confidence w.r.t to many items.

**3. Lift:**

It is calculated to measure how likely item Y is purchased when item X is purchased while controlling for how popular item Y is.

* lift(X→Y) = 1 if X and Y are independent
* lift(X→Y) < 1, unlikely to be bought together**.**
* lift(X → Y) > 1, likely to be bought together
* lift value ranges from 0 to infinity.
* Gives Bi-directional recommendations.

**Steps to implement the apriori algorithm:**

### Create a frequency table of all the items that occur in all transactions.

1. Create a pivot matrix representing 1 if the item is present and 0 if the item is not present.
2. Encode the matrix, return 0 for all the items that have the value less than or equal to 0 and return 1 for all the values that are greater than or equal to 1.
3. Use the library mlxtend.frequent\_patterns and import apriori.
4. Calculate frequent itemsets using the metric min\_support.
5. Use the library mlxtend.frequent\_patterns and import association\_rules
6. With the help of association\_rules select the appropriate metric (ex: support, confidence, lift … default = confidence) to recommend the items.

**Advantages:**

1. It is the most simple and easy to understand and implement.
2. It doesn’t require labeled data as it is completely unsupervised and hence this can be used for many different situations as we can find unlabeled data quite often.
3. It is used to calculate large item sets.

**Disadvantages:**

1. Apriori algorithm is an expensive method to find support since the calculation has to pass through the whole database.
2. It is computationally expensive.
3. Complexity grows exponentially.
4. Cold start problem.

## **Content-Based recommender System**

* Content-based filtering makes recommendations based on the similarity of items. It uses item features to recommend other items similar to what the user likes, based on their previous actions or explicit feedback.
* Content-based filtering does not require other users' data during recommendation to one user.
* Content-based recommender systems try to match users to items that are similar to what they have liked in the past. This similarity is not necessarily based on rating correlations across users but on the basis of the *attributes*  of the objects liked by the user
* At the most basic level, content-based systems are dependent on two sources of data:
  + The ﬁrst source of data is a description of various items in terms of content-centric attributes. An example of such a representation could be the text description of an item by the manufacturer.
  + The second source of data is a user proﬁle, which is generated from user feedback about various items. The user feedback might be explicit or implicit. Explicit feedback may correspond to ratings, whereas implicit feedback may correspond to user actions.
* The cold-start problem, which describes the difficulty of making recommendations when the users or the items are new.
* **How can we overcome the cold start problem?**
  + A basic idea would be to recommend the most popular / frequently bought items using a frequency-based approach. But this is a very vague approach.
* Consider the case of a new user.
  + Even though we do not have any information regarding this user's interactions with different items, we do have other additional information about this new user.
  + **Location**
    - This can be used to get an idea of the items used/purchased by other users in that area.
    - A swiggy recommender engine can make an assumption that Idli-Dosa is more probable to be liked by a user residing in Southern India.
    - We can get the location of users from IP Addresses.
    - Most platforms do ask for your location before letting you sign up.
  + **Gender**
    - Useful in recommending clothes and accessories.
  + **Age**
  + **Type of Credit Card**
    - This too can help get a lot of information about the data, like their spending habits, their credit limit, the brand of credit card, etc.
  + **The device being used to access the platform**
    - We can assume that a user using Apple Macbook would have more spending power than a user using a cheap Chinese smartphone.
* We form a new d'-dimensional vector that holds all this data, and then use **user-user similarity** on it, and recommend accordingly.
* This is known as **User-user similarity-based Content Filtering** Recommender systems.
* Consider that there is a new product on Amazon, and though there is no data about user ratings, we still have additional information like
  + **Product Description**
    - This would potentially be stored as a BoW(Bag of Words).
  + **Price**
  + **Category of product**
    - Like electronics, clothing, sports, etc…and so on.
* We form a new d-dimensional vector that holds all this data, and then use **item-item similarity** on it, and recommend accordingly.
* Hence this is called **item-item similarity-based content filtering** Recommender systems.
* We can recommend this new item to those users who bought similar items from the same categories until we have sufficient information.
  + This additional information is known as **metadata**.
* This process of finding user-user or item-item similarities, using metadata in order to recommend items to users is called a **Content-based Recommendation system**.

**Why is it called a "Content-based" Recommendation**

* Because we are not using the purchase data for finding similarities and then recommending.
* Instead, we are using **features extracted from**/provided in the **content** of the user/item to form a d-dimensional vector representing the user/item metadata.
* The point of content-based filtering is that we have to know the content of both the users and the items.

**Advantages:**

1. The model doesn't need any data about other users, since the recommendations are specific to this user. This makes it easier to scale to a large number of users.
2. No cold start problem.
3. No sparsity problem.
4. The model can capture the specific interests of a user and can recommend niche items that very few other users are interested in.

**Disadvantages:**

1. Since the feature representation of the items is hand-engineered to some extent, this technique requires a lot of domain knowledge.
2. It always recommends items related to the same categories, and never recommends anything from other categories.

## **Collaborative Filtering System**

* To address some of the limitations of content-based filtering, collaborative filtering uses similarities between users and items simultaneously to provide recommendations.
* This system can filter out items that a user might like on the basis of reactions by similar users and makes recommendations based on user interactions.
* This filtering works on the assumption that people who like similar things have similar tastes. There are different approaches we can adopt to implement CF recommender systems
* Collaborative filtering methods have been applied to many different kinds of data including
  + sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors;
  + financial data, such as financial service institutions that integrate many financial sources;
  + electronic commerce and web applications where the focus is on user data, etc.
* The workflow of a collaborative filtering system is:

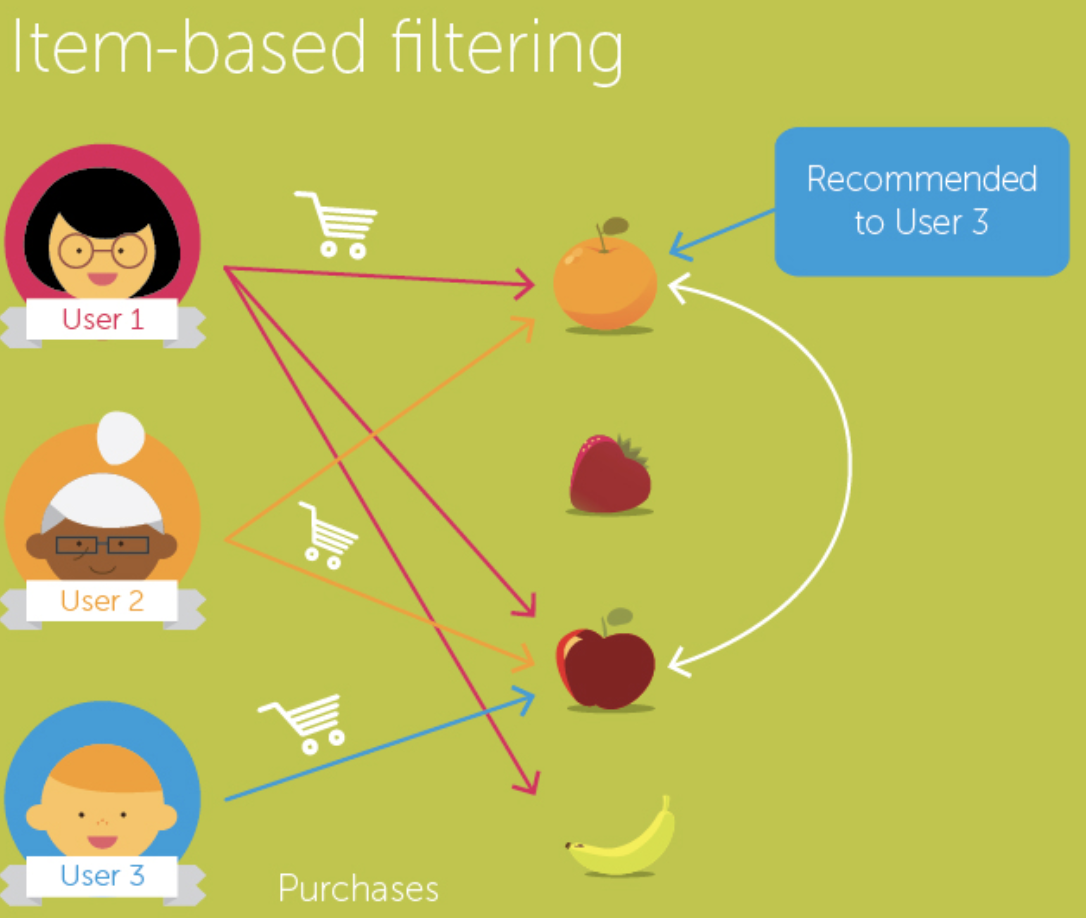
1. A user expresses his or her preferences by rating items (e.g. books, movies, or music recordings) of the system. These ratings can be viewed as an approximate representation of the user's interest in the corresponding domain.
2. The system matches this user's ratings against other users and finds the people with the most "similar" tastes.
3. With similar users, the system recommends items that similar users have rated highly but are not yet been rated by this user (presumably the absence of rating is often considered as the unfamiliarity of an item)

**Challenges with Collaborative Filtering**

* Cold Start problem:
  + As collaborative filtering methods recommend items based on users past preferences, new users will need to rate a sufficient number of items to enable the system to capture their preferences accurately and thus provides reliable recommendations.
  + Similarly, new items also have the same problem. When new items are added to the system, they need to be rated by a substantial number of users before they could be recommended to users who have similar tastes to the ones who rated them.
* Sparsity:
  + If the user/rating matrix is sparse, it is not easy to find the users that have rated the same items, which brings about challenges in the performance of the recommendation.
* Popularity Bias:
  + Not possible to recommend items to users with unique tastes.
  + Popular items are recommended more to the users as more data is available on them
  + This may begin a positive feedback loop not allowing the model to recommend items with less popularity to the users
* Shilling attacks
  + In a recommendation system where everyone can give the ratings, people may give many positive ratings for their own items and negative ratings for their competitors.
  + It is often necessary for collaborative filtering systems to introduce precautions to discourage such manipulations.
* Scalability
  + Collaborative filtering models are computationally expensive

### **(a). Item - Item- Based recommender System**

* Item-item collaborative filtering, or item-based, or item-to-item, is a form of [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) for [recommender systems](https://en.wikipedia.org/wiki/Recommender_systems) based on the similarity between the items are calculated.
* Build an item-item matrix determining relationships between pairs of items.
* Infer the tastes of the current user by examining the matrix and matching that user's data.
* “Users who liked this item also liked…”



Ex:

* User 1 bought an Orange, Strawberry, and Apple.
* User 2 bought Orange and Apple.
* User 3 purchases Apple so we calculate the similarity between the fruits and recommend Orange.

To calculate the similarity between two items, we use hamming distance.

* **Hamming Distance**: Hamming distance is a metric for comparing two binary data strings. While comparing two binary strings of equal length, Hamming distance is the number of bit positions in which the two bits are different.

The Hamming distance between two strings, a and b is denoted as d(a,b).

Ex: We have two strings in binary data

Str1 = [0,0,1,0,0,1]

Str2 = [0,1,1,0,1,1].

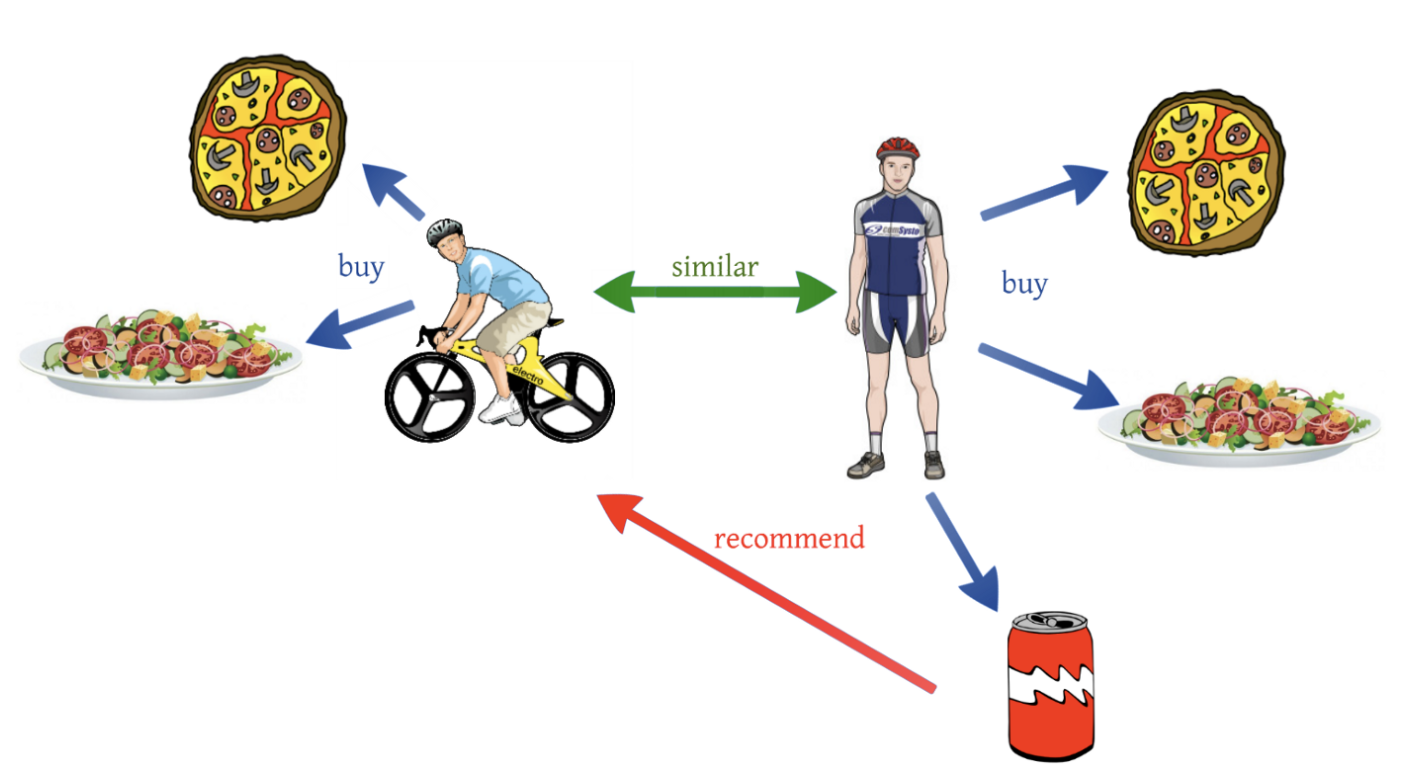
So, there are two positions in which bits are different hence the distance will be 2.

How does Item-based work?

* Firstly, we compute similarities between items using hamming distance.
* Secondly, based on the computed similarities, items similar to those already consumed/rated are looked at and recommended accordingly.

### **(b). User - User- Based recommender System**

* User-user collaborative filtering, or user-based, or user-to-user, is a form of [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) for [recommender systems](https://en.wikipedia.org/wiki/Recommender_systems) based on the similarity between the users calculated.
* Look for users who share the same rating patterns as the active user (the user whom the prediction is for).
* Use the ratings from those like-minded users found in step 1 to calculate a prediction for the active user.
* “ Users who are similar to you also liked…”



Ex:

* There is user 1 who bought pizza, salad, and soft drink and there is user 2 who bought pizza and salad. So, we calculate the similarity between the two users and recommend a soft drink to user 2.

To calculate the similarity between two items, we use euclidean distance or cosine-similarity.

* **Euclidean Distance**: The Euclidean distance is already familiar, the Euclidean distance is calculated between two 2-dimensional vectors x = (x1, x2) and y = (y1, y2) is given by:
* **Cosine Similarity**: Cosine similarity is the cosine of the angle between two vectors and it is used as a distance evaluation metric between two points in the plane.

Cosine similarity ranges from -1 to 1.

1 indicates the items are the same whereas -1 represents the compared items are dissimilar.

How User-based works?

* Firstly, we compute similarities between users using euclidean distance or cosine similarity.
* Secondly, based on the computed similarities between the users, the items of similar users are recommended.

**Drawbacks**:

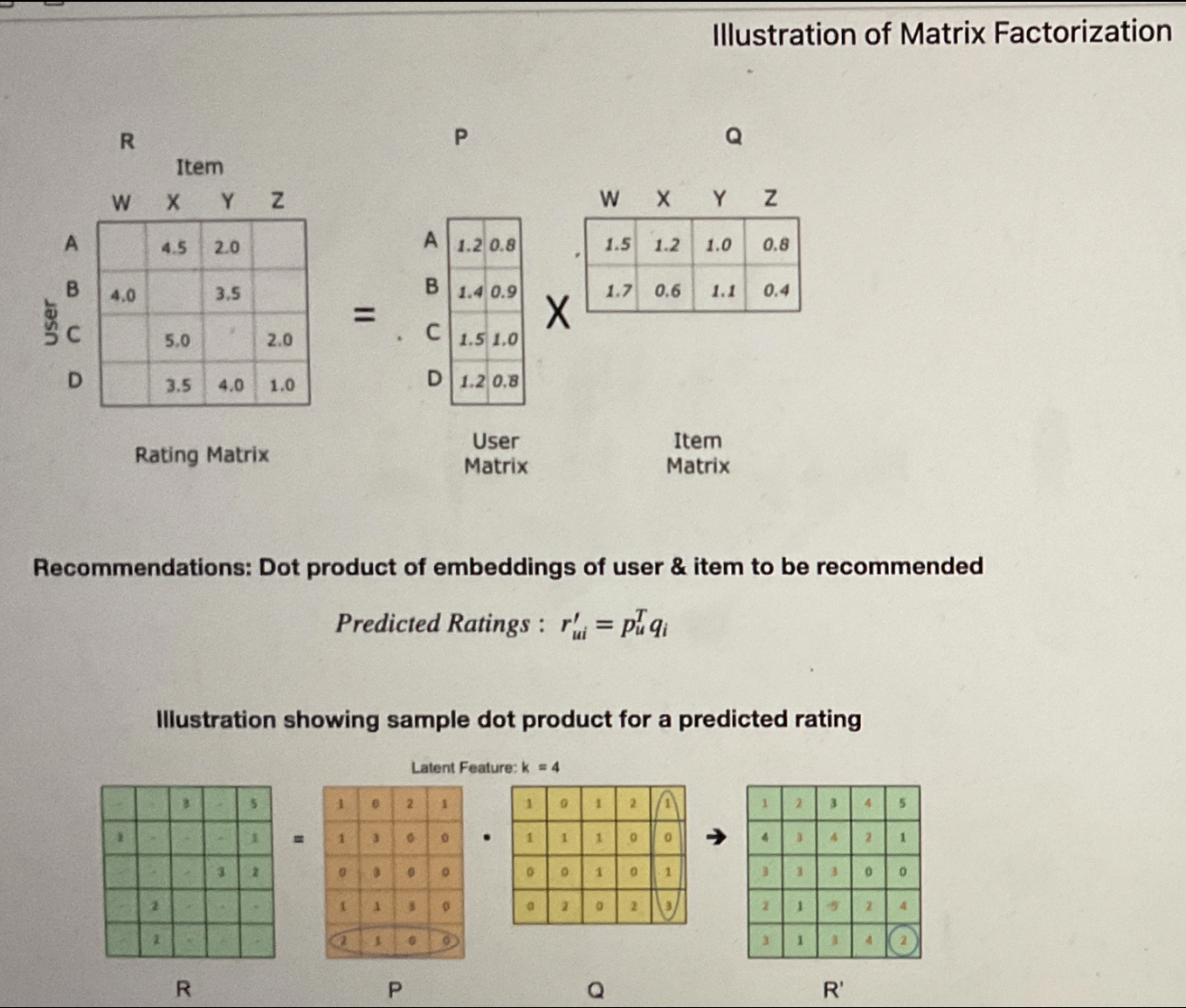
* Data Sparsity: In the case of a large number of items, the number of items a user has rated reduces to a tiny percentage making the correlation coefficient less reliable.
* User profiles change quickly and the entire system model has to be recomputed which is both time and computationally expensive.
* To overcome these drawbacks we use an item-item-based recommender system.

### **(c). User - Item Based recommender System**

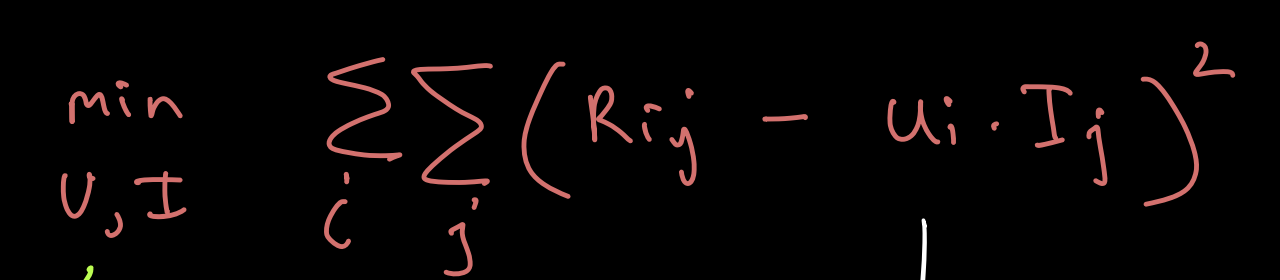
* User-Item collaborative filtering is a form of [collaborative filtering](https://en.wikipedia.org/wiki/Collaborative_filtering) for [recommender systems](https://en.wikipedia.org/wiki/Recommender_systems) based on the similarity between the users and items is calculated. Data contains a set of users and items and ratings/reactions in the form of a user-item interaction matrix.
* If the user-item interaction matrix is sparse it is not easy to find the users that have rated the same items.
* To deal with this sparsity problem we use matrix factorization.

## **Matrix Factorization**

* Matrix factorization is a way to generate latent features when multiplying two different kinds of entities. Collaborative filtering is the application of matrix factorization to identify the relationship between items and user entities.
* With the input of users' ratings on the items, we would like to predict how the users would rate the items so the users can get the recommender based on the prediction.
* Since not every user gives ratings to all the items, there are many missing values in the matrix and it results in a sparse matrix.
* Hence, the null values not given by the users would be filled with 0 such that the filled values are provided for the multiplication.

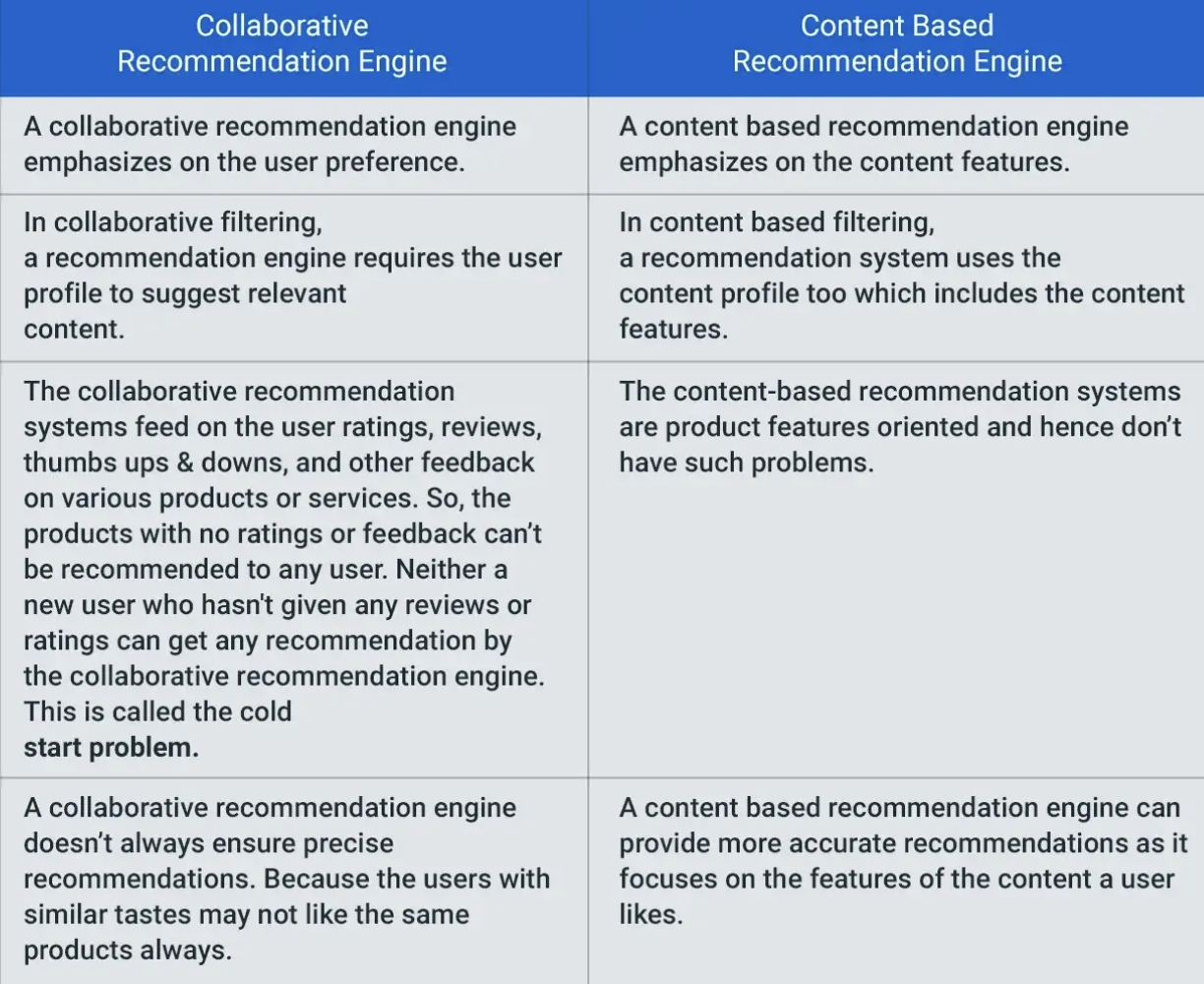
****

* Matrix P represents the association between a user and the features while matrix Q represents the association between an item and the features.
* We can get the prediction of a rating of an item by the calculation of the dot product of the two vectors.
* To get two entities of both P and Q, we need to initialize the two matrices and calculate the difference and we minimize the difference through the iterations. The method is called gradient descent, aiming at finding a local minimum of the difference.



* The user-item interaction matrix is very sparse since every user does not rate all items. The missing entries in the matrix would be replaced by the dot product of the factor matrices. Therefore, we know what to recommend to the users with the unseen movies based on the prediction.

**Collaborative filtering vs Content-based recommender system**

****

**Advantages of CF:**

1. No domain knowledge necessary
   * We don't need domain knowledge because the embeddings are automatically learned.
2. Serendipity
   * The model can help users discover new interests. In isolation, the ML system may not know the user is interested in a given item, but the model might still recommend it because similar users are interested in that item.
3. The system doesn't need contextual features.

**Disadvantages of CF:**

1. Fails in case of cold start: the model requires enough other users already in the system to find a good match.
2. Collaborative filtering models are computationally expensive.
3. It's a bit difficult to recommend items to users with unique tastes.