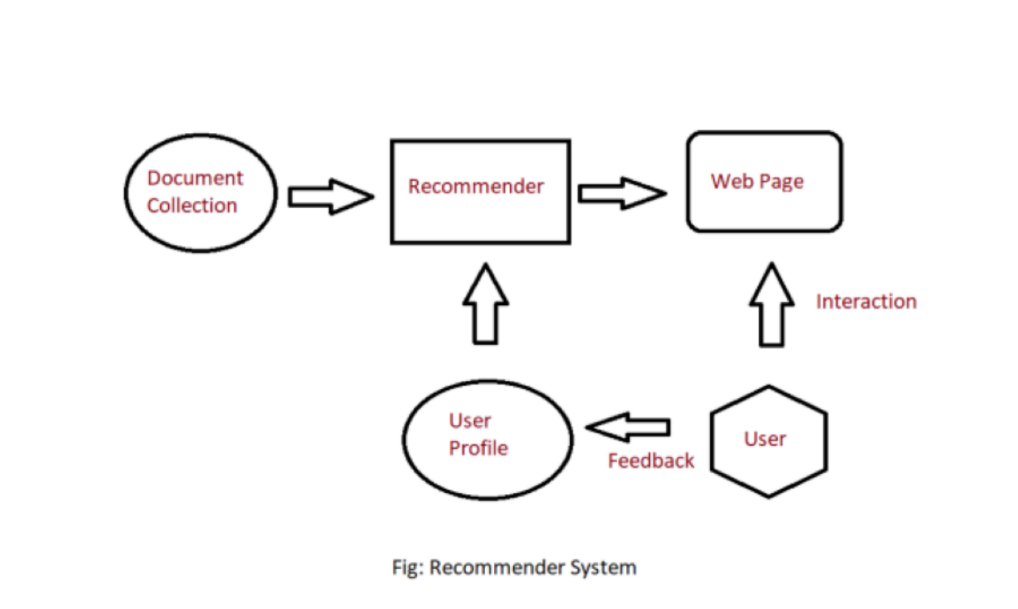
**Section 11.9**

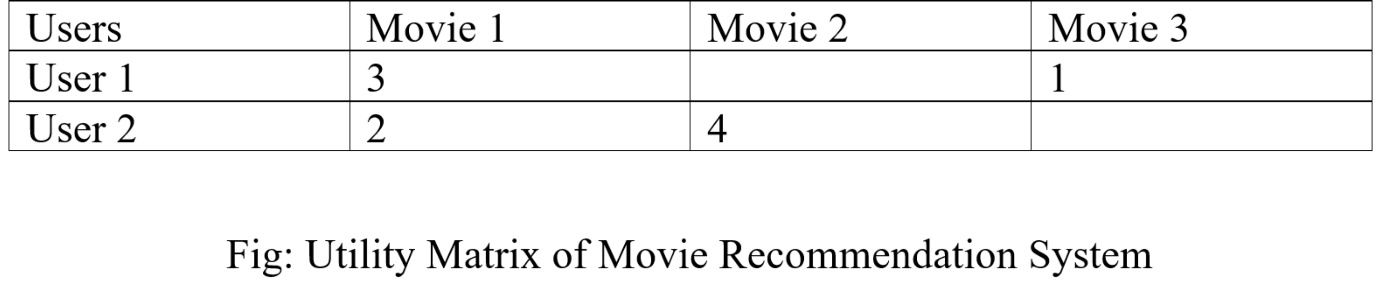
**CONTENT-BASED RECOMMENDER**

A Content-Based Recommender works on the data that collected from the user, either explicitly through rating/ feedback or implicitly by analyzing the links clicked. With data, a user profile is created. As the user provides more input or take more actions on the recommendation, the engine gets better and better.



**User Profile:** Technically, User Profile is used to create vectors that describe the user’s preference. Such a utility matrix describes the relationship between a user and the item. This information is used estimate the item that user likes.

In the utility matrix, a particular value is assigned to each user-item pair. This value indicates the degree of preference. From the matrix, we can easily identify user preference relationship (as shown in figure).



If you observe carefully, some columns are blank in the matrix. This is because users do not give inputs every time. The objective of a recommendation system is not to fill the columns but to recommend a movie to the user which he/she will prefer. As per the given matrix, the recommender system will not suggest Movie 3 to User 2. Can you guess why?

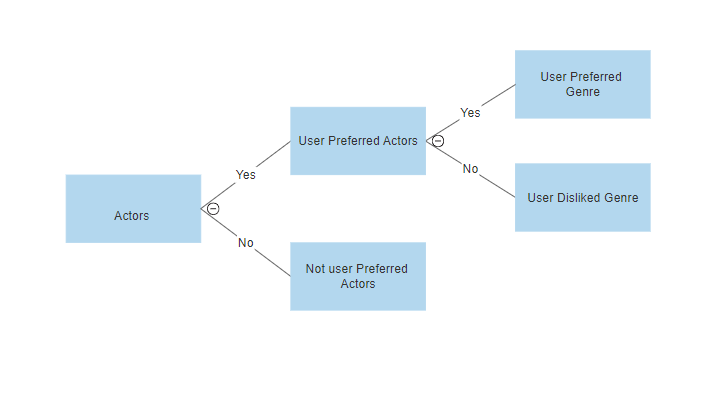
Look at Movie 1. User 1 and user 2 have given approximately the same ratings to movie 1. Continuing with that, Movie 3 has been rated low by User 1. This indicates that User 2 will also not like Movie 3.

**Item Profile:** Besides user profile, Content-Based Recommender also creates a profile for each item, which represents important features of that item. For example, if movie is an item, then its actors, director, release year, genre and rating from the IMDB (Internet Movie Database) are the features of interest.   
**Recommending Items to User Based on Content:**

**Method 1:** The cosine distance between the vectors of the item and the user is calculated to determine the item’s preference to the user. For example,

If the vector for a user has a positive number for actors, it indicates that the user will the movies in which that actor appears. A movie with actors which user likes will have a large positive fraction. This means that the cosine angle will be close to 0, indicating a small cosine distance between the vectors.

**Method 2:** Classification techniques like the Decision Tree can be used to identify whether a user wants to watch a movie or not. At each level of the tree, a certain condition is checked to refine the recommendation. For example:



**What are Recommender Systems?**

As a user, we often feel trapped with so many choices available to us. Whether we want to know which movie to watch, which dress to buy, which book to read, there are so many choices and we often get confused. ***To solve this problem, recommendation systems analyze likes and dislikes of users to help them make the best choice.***

Recommender systems, are information filtering system that are used to forecast the "rating" or "preference" that a user would assign to an item. They play an important role in online shopping, streaming services, social networking, and other platforms that requires personalized user experience. In these applications, recommendation systems evaluate user data, such as prior purchases, reviews, or browsing history, to find trends and preferences to recommend goods that the user may find interesting.

For example, we all have experienced that online e-commerce website like Amazon recommend products based on user’s browsing and purchase history. Similarly, music streaming services like Spotify, propose songs and artists based on your listening history.

**Types of Recommendation Systems**

There are broad categories of **Recommendation Systems-** collaborative filtering, content-based filtering, and hybrid systems.

**Collaborative filtering:** It works by evaluating user interactions and determining similarities between people (user-based) and things (item-based). This means that if User A and User B like the same movies, then User A may love to watch other movies enjoyed by User B. Thus, such a system forecast items which user may enjoy based on the preferences of other users who have similar likes.

Collaborative filtering analyses user interactions to identify similarities between individuals (user-based) and objects (item-based).

**User-based Collaborative Filtering:** This technique predicts product(s) that may interest a user. It works by analyzing ratings provided to that product by other users having similar preferences. The steps involved include:

* **Find similarities between users and the target user. For this** an algorithm is used to analyze the ratings provided by both users to common goods.
* **Predict the missing rating of an item**: This is done using a weighted average method. In this technique, the ratings entered by similar users are given more weightage than the ratings entered by other users who have a different set of preferences.

**Item-Based Collaborative Filtering:** In this approach, things a user would enjoy is predicted based on their similarity. The steps performed during this type of filtering include,

* **Item to item similarity:**The cosine similarity is used to identify pairing of items that are similar.
* **Prediction Computation:** The missing rating for a product is then determined by analyzing ratings given by the user for similar products in the past. For this, a weighted sum of the ratings of other comparable goods is used.

User-based and item-based collaborative filtering may work on the same data.

**Method 2. Content-based filtering**

Content-based filtering, as the name suggests, is a technique used in recommender systems that suggest items that are similar to the items a user has shown interest in, based on the **attributes of the** item. Machine learning algorithms (like decision trees) are used to classify similar items based on features like genres, directors, or keywords associated with previously seen movies.

Content-based filtering is effective for organizations that offer a variety of goods, services, or information to provide personalized suggestions to users based on their previous behavior or explicit input. For example, if a user has given higher ratings to action movies, then the algorithm will recommend more action movies based on actors, directors, or keywords connected with previously liked movies.

To identify the relationship between user and item, vectors are used to represent information. Then, statistic metric dot product is used to observe how many features are active in both vectors at a moment. A high dot product indicates more common features and therefore higher similarity. Even the vector spacing method can be used make recommendations based on the distance between the user and item vectors.

Note that content-based filtering does not rely on data from other users. These recommendations are effective for people with specific taste or items with low user interaction data. so, the success of this technique depends on the quality of the item features available and the ability of the algorithm to capture the intricacies of human preferences.

**Hybrid systems:** Hybrid recommender systems are the most effective approach to developing a recommender system. These systems combine the best features of collaborative and content-based methods for more accurate and diversified recommendations. Hybrid systems often start with content-based filtering to study new users and then applies collaborative filtering when more interaction data becomes available.

Hybrid recommender systems can be categorized weighted, feature combination, cascade, feature augmentation, meta-level, switching, and mixed models.

As the name suggests, the feature combination method uses collaborative information in addition to feature data collected from content-based approaches. Similarly, the meta-level hybrid recommender system combines two recommender systems in such a way that the output of one becomes the input for the other.

Recommender systems are built by analyzing both explicit data, such as user ratings and reviews, as well as implicit data, including browsing history and click habits.

**Deep Neural Network Models for Recommendation Systems**

Deep learning techniques are used these days to develop sophisticated models that can capture complex patterns in user behavior and item features. Some commonly used deep learning models for recommendation include,

**Autoencoders:**Autoencoders are neural networks that learns from data to represent input efficiently. In recommender systems, autoencoders are used to create user-item interaction matrices. Such a matrix compresses user preferences into a smaller latent space in such a way, that the original user preferences are easily recovered from the compressed representation. While the encoder reduces the data's dimensionality, the decoder reconstructs it.

**Deep Neural Networks (DNNs):** In DNN, multiple layers of interconnected neurons are used. The input data is converted into a higher-level representation by each layer, to capture complex patterns. The relationships between users and items are represented using DNNs. For this, user demographics, item attributes, and past interactions between user and the item are considered to compute the probability of a user interacting with that item.

**Convolutional Neural Network (CNNs):** CNNs are extensively used to process images, videos, or any content where spatial or temporal patterns are important. CNNs can easily extract high-level features from visual content to recommend similar items based.

**Recurrent Neural Networks (RNNs):** RNNs are used to process sequential data in such a way that the output at each step is based on the previous input. RNNs are best-suited for session-based recommendations, where the order of interactions matters. In such a scenario, temporal dependencies in user behavior are modelled to get recommendations based on the sequence of actions taken by a user.

**Attention Mechanisms:** Attention mechanisms facilitate focusing on the most relevant parts of the input data. Dynamic weights are assigned to different parts of the input to highlight the most important features. In recommendation systems, features or interactions that are prominently associated with a user’s preferences are identified and prioritized by attention mechanisms to get more accurate predictions.

**Importance of Recommendation Systems**

As stated earlier, recommender systems are now an important part of digital platforms. Such systems help to improve user experiences, drive engagement, and provide decision-making tools. Recommender systems are often used as information filtering tools to give users information that is most relevant to them. Some of the benefits of these systems are given below.   
**Faster Decision-making:** Recommender systems improve decision-making process. And this in turn boost sales, enhances customer loyalty and overall happiness and lower transaction costs.

**Personalized user experience:** Personalized customer experience provides highly relevant and valuable suggestions to the user thereby improving their relationship with the business.

**Increase engagement:** By providing customers information, goods, or services that they are likely to be interested in, recommender system increases user engagement.

**Recommender System using the K-Nearest Neighbors (KNN) Algorithm**

KNN, being a simple, non-parametric, and instance-based learning algorithm is often used for classification and regression tasks. As you recall, KNN finds the closest neighbours. The same concept can be used to identify either nearest users or items based on a similarity metric. Based on the preferences of the neighbours, recommendations are provided to the users.

We know that the user-based collaborative filtering approach recommends items to a user by finding similar users (neighbours) who have similar preferences and the item-based collaborative filtering recommends items based on the similarity between items. That is, by identifying items similar to those the user has liked in the past. With this recapitulation, let us see the steps to build a KNN-Based Recommender System.

Data that describes information about user interactions with items is collected. This data includes features like ratings, purchases, or clicks.

The collected data is pre-processed to handle missing values, normalize ratings, and transform it into a user-item matrix.

The similarity between users or items using a similarity metric is calculated. Common metrics that are often used at this stage are,

* **Cosine Similarity, which m**easures the cosine of the angle between two vectors.
* **Pearson Correlation that computes** the linear correlation between two vectors.
* **Jaccard Similarity that calculates** the similarity between two sets.

For each user or item, find the K-nearest neighbors are identified based on the computed similarity scores. For this, the similarity scores are sorted and then the top K neighbors are selected.

Based on the preferences of the K-nearest neighbors, recommendations are made for the user.

To make recommendations, aggregate the ratings or interactions of the neighbors. Then, recommend items with the highest aggregated scores.

|  |
| --- |
| **Code Implementation of building Recommender Systems using KNN**  **import** **numpy** **as** **np**  **from** **sklearn.neighbors** **import** NearestNeighbors  # rows represent users and columns represent items, *0 means no rating*  user\_item\_matrix = np.array([  [4, 0, 0, 5, 1],  [5, 5, 4, 0, 0],  [0, 0, 0, 2, 4],  [0, 3, 0, 0, 5],  [5, 0, 4, 0, 0]  ])  *# Normalize the matrix by subtracting the mean rating of each user*  mean\_user\_rating = np.mean(user\_item\_matrix, axis=1).reshape(-1, 1)  normalized\_matrix = user\_item\_matrix - mean\_user\_rating  *# Fit the KNN model*  knn = NearestNeighbors(metric='cosine', algorithm='brute')  knn.fit(normalized\_matrix)  *# Find the k nearest neighbors for a target user (e.g., user index 0)*  target\_user\_index = 0  distances, indices = knn.kneighbors(normalized\_matrix[target\_user\_index].reshape(1, -1), n\_neighbors=3)  *# Aggregate ratings from the nearest neighbors*  neighbors\_ratings = user\_item\_matrix[indices.flatten()]  predicted\_ratings = neighbors\_ratings.mean(axis=0)  *#* identify items that the target user has not rated (i.e., entries with a value of 0)  unrated\_items = np.where(user\_item\_matrix[target\_user\_index] == 0)[0]  *# Recommend items with the highest predicted ratings that the target user hasn't rated*  recommended\_items = unrated\_items[np.argsort(predicted\_ratings[unrated\_items])[::-1]]  print(f"Recommended items for user **{**target\_user\_index**}**: **{**recommended\_items**}**")  **Output:**  Recommended items for user 0: [2 1] |

|  |  |
| --- | --- |
| **Advantages**:   * Simple to implement * Easy to understand. * Effective for small to medium-sized datasets. * No need for model training; works on instance-based learning. | **Disadvantages**:   * Computationally expensive for large datasets. * Performance degrade if used with sparse data. |

|  |
| --- |
| **Create a Recommender System for Book Ratings Dataset (**[book review](https://drive.google.com/file/d/1lvWO2ewsoGM56-_Y1F4pW9Mcc_QPm8yr/view?usp=sharing)) **using Pyspark.**  #importing the required pyspark library  from pyspark.sql import SparkSession  from pyspark.ml.evaluation import RegressionEvaluator  from pyspark.ml.recommendation import ALS  #Setup Spark Session  spark = SparkSession.builder.appName('Recommender').getOrCreate()  #CSV file can be downloaded from the link mentioned above.  data = spark.read.csv('book\_ratings.csv',  inferSchema=True,header=True)  #data.describe().show()  # Random split into train\_data and test\_data  train\_data, test\_data = data.randomSplit([0.8, 0.2])  # Build the recommendation model using ALS on the training data  als = ALS(maxIter=5,  regParam=0.01,  userCol="user\_id",  itemCol="book\_id",  ratingCol="rating")  #Fitting the model on the train\_data  model = als.fit(train\_data)  # Evaluate the model by computing the RMSE on the test data  predictions = model.transform(test\_data)  #Displaying predictions calculated by the model  predictions.show(5)  #Printing and calculating RMSE  evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",predictionCol="prediction")  rmse = evaluator.evaluate(predictions)  print("Root-mean-square error = " + str(rmse))  #Filtering user with user id "5461" with book id on which it has given the reviews  user1 = test\_data.filter(test\_data['user\_id']==5461).select(['book\_id','user\_id'])  #Displaying user1 data  user1.show(5)  recommendations = model.transform(user1)  #Displaying the predictions of books for user1  recommendations.orderBy('prediction',ascending=False).show(5)  spark.stop()  OUTPUT  +-------+-------+------+----------+  |book\_id|user\_id|rating|prediction|  +-------+-------+------+----------+  | 1| 588| 5| 3.804705|  | 1| 5461| 3| 4.053508|  | 1| 17984| 5| 4.1971784|  | 6685| 5015| 3| 2.5951052|  | 3373| 25405| 4| 3.014865|  +-------+-------+------+----------+  only showing top 5 rows  Root-mean-square error = nan  +-------+-------+  |book\_id|user\_id|  +-------+-------+  | 1| 5461|  | 7| 5461|  | 9| 5461|  | 16| 5461|  | 47| 5461|  +-------+-------+  only showing top 5 rows  +-------+-------+----------+  |book\_id|user\_id|prediction|  +-------+-------+----------+  | 66| 5461| 5.3337426|  | 58| 5461| 4.886608|  | 869| 5461| 4.855506|  | 354| 5461| 4.7570896|  | 661| 5461| 4.662834|  +-------+-------+----------+  only showing top 5 rows |

The above code implements collaborative filtering using library Spark MLlib and the method, **Alternating Least Squares**. Some important parameters that apply to the MLlib implementation are,

* The number of blocks used to parallelize computation is numBlocks (set to -1 to auto-configure).
* rank is the number of latent factors in the model.
* iteration is the number of iterations to execute
* lambda specifies the regularisation parameter in ALS
* implicitPrefs is used to determine whether to use the ALS variation tailored for implicit feedback data or the explicit feedback.
* The implicit feedback variant of ALS has a parameter called alpha that controls the initial level of confidence in preference observations.