**Recommendation System in Python**

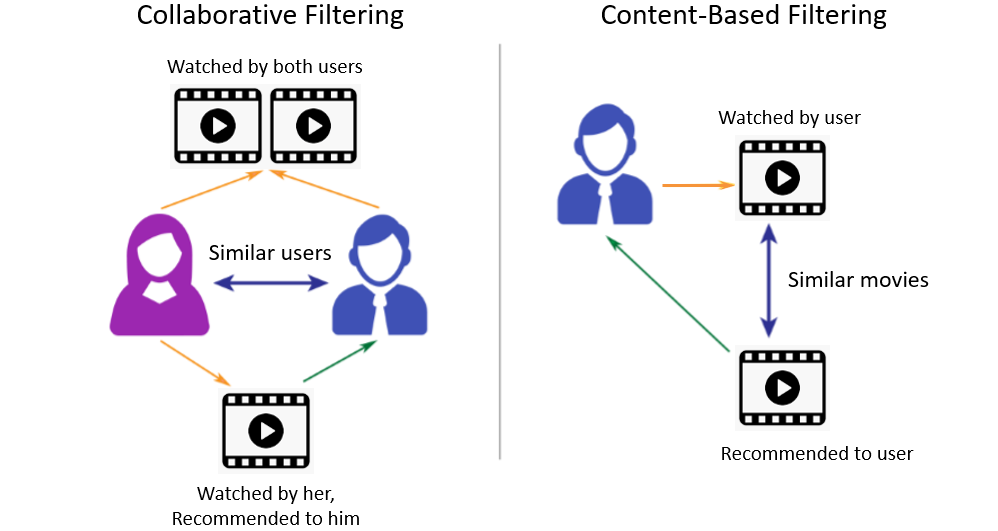
Generating value for businesses through data utilization and programming skills is at the core of both Data Science (DS) and Artificial Intelligence (AI). Industry pioneers like Netflix, Amazon, and Uber Eats have revolutionized the way people access products and services, enabling convenient experiences from the comfort of their homes with just a few clicks. These platforms leverage recommendation algorithms to enhance user satisfaction, offering personalized options tailored to individual interests and preferences. Python stands out as a crucial tool, providing a flexible and robust environment for developing and deploying cutting-edge recommendation systems.

In various applications, websites gather user data to predict their preferences, facilitating tailored content suggestions. Recommendation systems play a vital role in suggesting products and ideas aligned with a user’s unique perspective.

Python-based Recommendation Systems utilize a data-driven approach to deliver personalized suggestions to customers. Leveraging user data and algorithms, these systems forecast and recommend products, services, or content that users are likely to find compelling. They are particularly valuable in contexts where users face information overload, such as social media, streaming services, and e-commerce platforms. Python's extensive libraries and machine learning frameworks make it a preferred choice for building recommendation systems, with two primary approaches: content-based filtering (considering product attributes and user profiles) and collaborative filtering (based on user behavior and preferences). Hybrid strategies that combine these approaches are also gaining popularity. These systems enhance user experiences, increase engagement, and drive business growth.

**Recommender System is of different types:**

* **Content-Based Recommendation:** It is supervised machine learning used to induce a classifier to discriminate between interesting and uninteresting items for the user.
* **Collaborative Filtering:** Collaborative Filtering recommends items based on similarity measures between users and/or items. The basic assumption behind the algorithm is that users with similar interests have common preferences.

****

Content-Based Recommendation System: Content-based recommendation systems are a type of recommendation system commonly used in e-commerce, media streaming, and other online platforms to suggest items to users based on the attributes of those items. Unlike collaborative filtering methods, which rely on user interactions and similarities between users, content-based recommendation systems focus on the features of the items themselves.

Here's how they typically work:

**Item Representation:** Each item in the system is represented by a set of features or attributes. For example, in a movie recommendation system, attributes might include genre, director, actors, and plot keywords.

***User Profile*:** The system builds a profile for each user based on their preferences and interactions with items. This profile is often created by analyzing the items the user has interacted with in the past.

**Similarity Calculation:** To recommend items to a user, the system calculates the similarity between the user's profile and each item in the catalog. This is usually done using similarity measures such as cosine similarity or Pearson correlation.

***Ranking and Recommendation*:** Finally, the system ranks the items based on their similarity to the user profile and recommends the top-ranked items to the user.

Advantages of content-based recommendation systems include:

***No Cold Start Problem*:** They can recommend items to new users based solely on the attributes of those items.

***Transparency*:** The recommendations are based on understandable features of the items, making the system more transparent to users.

Us*er Independence*: Recommendations are personalized to each user, so they don't rely on similarities between users.

However, content-based recommendation systems also have limitations:

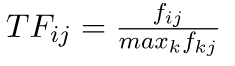
***Limited Serendipity*:** Since recommendations are based on the features of items the user has interacted with in the past, they may not introduce users to new or unexpected items.

***Limited Diversity*:** Recommendations may be biased towards items with similar attributes to those the user has already interacted with, leading to a lack of diversity in recommendations.

**Depend*ency on Feature Representation*:** The quality of recommendations depends heavily on the features used to represent items. If the features don't capture the essence of the items well, the recommendations may not be accurate.

Despite these limitations, content-based recommendation systems are widely used and can be effective in many scenarios, especially when there is rich metadata available for the items in the catalog.

**TF-IDF Vectorizer**

**Term Frequency(TF)**: Term frequency, or TF for short, is a key idea in information retrieval and natural language processing. It displays the regularity with which a certain term or word occurs in a text corpus or document. TF is used to rank terms in a document according to their relative value or significance.  
The term-frequency can be calculated by:  
  
where fij is the frequency of term(feature) i in document(item) j.   
For a variety of text analysis tasks, such as information retrieval, document classification, and sentiment analysis, the yielded TF value can be used to identify important terms in a document. It offers a framework for figuring out how relevant a word is in a particular situation.

**Inverse-document Frequency (IDF):** The measure known as Inverse Document Frequency (IDF) is employed in text analysis and information retrieval to evaluate the significance of phrases within a set of documents. IDF measures how uncommon or unique a term is in the corpus. To compute it, take the reciprocal of the fraction of documents that include the term and logarithmize it. Common terms have lower IDF values, while rare terms have higher values. IDF is an essential part of the TF-IDF (Term Frequency-Inverse Document Frequency) method, which uses it to assess the relative importance of terms in different documents. To improve information representation and retrieval from massive text datasets, IDF is used in tasks including document ranking, categorization, and text mining.  
The inverse-document frequency can be calculated with:  
  
where, ni number of documents that mention term i. N is the total number of docs.

A numerical statistic called Term Frequency-Inverse Document Frequency (TF-IDF) is employed in information retrieval and natural language processing. The term’s significance within a document is assessed in relation to a group of documents (the corpus). TF emphasizes terms with greater frequencies by measuring a term’s frequency of occurrence in a document. IDF evaluates a term’s rarity within the corpus, emphasizing terms that are distinct. A weighted score is produced for each term in a document by multiplying TF and IDF together to compute TF-IDF.

Therefore, the total formula is:



**User profile**

The user profile is a vector that describes the user preference. During the creation of the user’s profile, we use a utility matrix that describes the relationship between user and item. From this information, the best estimate we can decide which item the user likes, is some aggregation of the profiles of those items.

**Advantages and Disadvantages**

**Advantages:**

* No need for data on other users when applying to similar users.
* Able to recommend to users with unique tastes.
* Able to recommend new & popular items
* Explanations for recommended items.

**Disadvantages:**

* Finding the appropriate feature is hard.
* Doesn’t recommend items outside the user profile.

**Collaborative Filtering**

Collaborative filtering is based on the idea that similar people (based on the data) generally tend to like similar things. It predicts which item a user will like based on the item preferences of other similar users. Collaborative filtering uses a user-item matrix to generate recommendations. This matrix contains the values that indicate a user’s preference towards a given item. These values can represent either explicit feedback (direct user ratings) or implicit feedback (indirect user behavior such as listening, purchasing, watching).

**Explicit Feedback:** The amount of data that is collected from the users when they choose to do so. Many of the times, users choose not to provide data for the user. So, this data is scarce and sometimes costs money.  For example, ratings from the user.

**Implicit Feedback:** In implicit feedback, we track user behavior to predict their preference.

**Example:**

Consider a user x, we need to find another user whose rating are similar to x’s rating, and then we estimate x’s rating based on another user.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | M\_1 | M\_2 | M\_3 | M\_4 | M\_5 | M\_6 | M\_7 |
| A | 4 |  |  | 5 | 1 |  |  |
| B | 5 | 5 | 4 |  |  | 5 |  |
| C |  |  |  | 2 | 4 |  |  |
| D |  | 3 |  |  |  |  | 3 |

Let’s create a matrix representing different user and products:

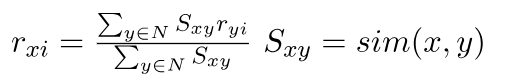
Consider two users x, y with rating vectors rx and ry. We need to decide a similarity matrix to calculate similarity b/w sim(x,y). THere are many methods to calculate similarity such as: Jaccard similarity, cosine similarity and pearson similarity. Here, we use centered cosine similarity/ pearson similarity, where we normalize the rating by subtracting the mean:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | M\_1 | M\_2 | M\_3 | M\_4 | M\_5 | M\_6 | M\_7 |
| A | 2/3 |  |  | 5/3 | -7/3 |  |  |
| B | 1/3 | 1/3 | -2/3 |  |  |  |  |
| C |  |  |  | -5/3 | 1/3 | 4/3 |  |
| D |  | 0 |  |  |  |  | 0 |

Here, we can calculate similarity: For ex: sim(A,B) = cos(rA, rB) = 0.09 ; sim(A,C) = -0.56. sim(A,B) > sim(A,C).

**Rating Predictions**

Let rx be the vector of user x’s rating. Let N be the set of k similar users who also rated item i. Then we can calculate the prediction of user x and item i by using following formula:



**Advantages and Disadvantages**

**Advantages:**

* No need for the domain knowledge because embedding are learned automatically.
* Capture inherent subtle characteristics.

**Disadvantages:**

* Cannot handle fresh items due to cold start problem.
* Hard to add any new features that may improve quality of model

**Imlementation of Recommendation System**

**FastAPI**

FastAPI is a modern, fast (hence the name), web framework for building APIs with Python. It's built on top of standard Python type hints and is designed to be easy to use, fast to run, and very performant. FastAPI leverages Python type hints to provide auto-generated interactive API documentation, making it easy to understand and work with APIs.

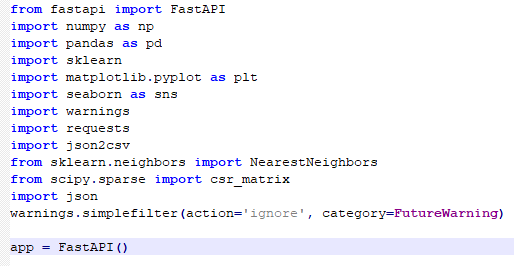
To run this API, you would typically save it to a Python file (e.g., `main.py`) and then use an ASGI server like uvicorn to serve it:

*uvicorn main:app --reload*

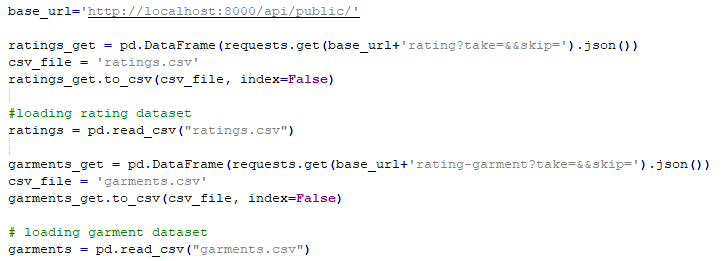
This command runs the FastAPI application in development mode (`--reload`), automatically reloading the server when code changes are detected.

FastAPI supports asynchronous request handlers, automatic validation of request parameters using Python type hints, automatic generation of OpenAPI (formerly known as Swagger) documentation, dependency injection, and much more. It's a powerful tool for building high-performance APIs with Python.

**Importing Libraries**

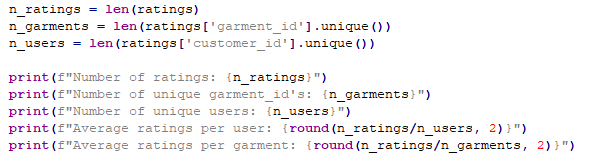
****

The Python environment for data analysis and visualization is initialized using this line of code. First, it imports essential libraries for data processing and visualization, including NumPy, Pandas, scikit-learn, Matplotlib, and Seaborn. It also sets up the code to suppress future warnings, so that cautions about upcoming library changes don’t clog the output and create a messier, less productive workspace. These preparatory actions create the framework for effective data exploration and analysis with the imported tools.



Two datasets are imported into this code to do a product recommendation study. User ratings for products are included in the first dataset, “ratings.csv,” which is kept in a Pandas DataFrame named ratings. The second dataset, called “garments.csv,” is put into a Pandas DataFrame called “garments” and contains garment metadata like names and price. In order to give a preliminary overview of the data and lay the groundwork for further analysis or recommendation system development, the code displays the first few rows of each DataFrame.

***Statistical Analysis of Ratings***



**Output:**

Number of ratings: 100836

Number of unique productId's: 9724

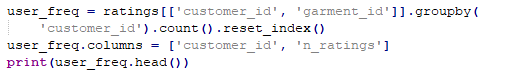
Number of unique users: 610

Average ratings per user: 165.3

Average ratings per garment: 10.37

This code computes and reports a number of crucial statistics for a garment ratings dataset. It counts the number of unique garment IDs (n\_ garments) and user IDs (n\_users) as well as the total number of ratings (n\_ratings). These metrics provide important information about the properties of the dataset, including its size and the variety of people and products inside it. To give a more complete picture of the distribution of ratings throughout the dataset, it also calculates and shows the average number of ratings for each user and each garment. Understanding the size and user interaction of the dataset requires knowledge of this information.

***User Rating Frequency***

****

**Output:**

userId n\_ratings

0 1 232

1 2 29

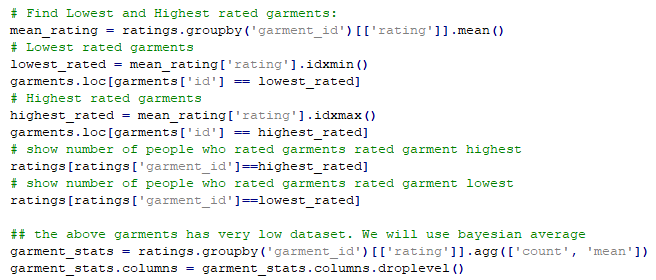
2 3 39

3 4 216

4 5 44

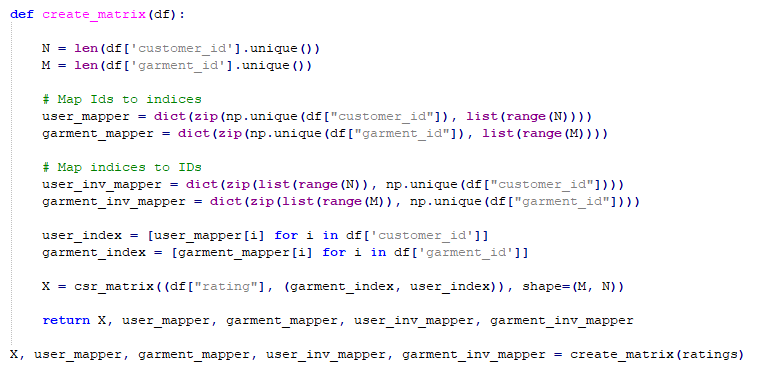
The garment ratings dataset’s user-specific statistics are computed and shown in this code segment. After classifying the data according to user IDs, it calculates the total number of ratings each user has submitted and saves the results in a new DataFrame named user\_freq. With ‘userId’ denoting the user ID and ‘n\_ratings’ the number of ratings the user has contributed, the columns are suitably labeled. To facilitate additional user-based analysis and the creation of recommendation systems, this user-level frequency information is crucial for comprehending user engagement and activity inside the rating dataset. The first few rows of this DataFrame are shown for a brief summary of user-specific rating counts by the print(user\_freq.head()) line.

***Garment Rating Analysis***



To determine which garments in the dataset have the lowest and highest ratings, this algorithm analyzes garment reviews. It determines the average ratings for every film, making it possible to identify which ones have the lowest and greatest average ratings. Subsequently, the algorithm accesses and presents the information about these films from the’garments’ dataset. It also sheds light on the popularity and audience involvement of the garment by displaying the number of users who rated both the highest and lowest-ranked ones. This gives insights into user engagement. Bayesian averages may offer more accurate quality ratings for films with a small number of ratings.

**User-Item Matrix Creation**



A user-item matrix is a basic data structure in recommendation systems, and it is created by the code that is given. This is how it operates:

To find the number of unique users and unique videos in the dataset, N and M are computed.

There are four dictionaries produced:

**user\_mapper:** Maps distinct user IDs to indexes (user ID 1 becomes index 0 for example).

**garment\_mapper:** Converts distinct garment IDs into indices (garment ID 1 becomes index 0 for example).

**user\_inv\_mapper:** Reverses user\_mapper and maps indices back to user IDs.

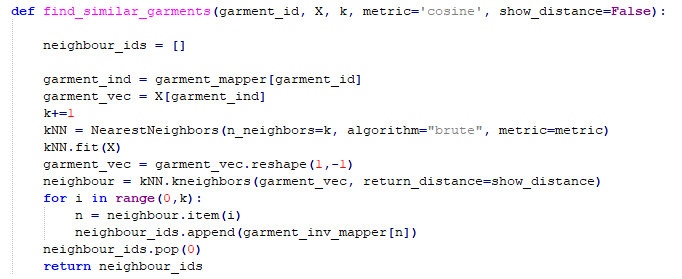
**garment\_inv\_mapper:** Reverses garment\_mapper by mapping indices to garment IDs.

To map the real user and garment IDs in the dataset to their matching indices, the lists user\_index and garment\_index are generated.

A sparse matrix X is created using the SciPy function csr\_matrix. The user and garment indices that correspond to the rating values in the dataset are used to generate this matrix. The form of it is (M, N), where M denotes the quantity of distinct films and N denotes the quantity of distinct consumers.

To put it another way, this code makes it easy to do calculations and create recommendation systems based on the structured representation of user ratings for garments in the data.

***Garment Similarity Analysis***

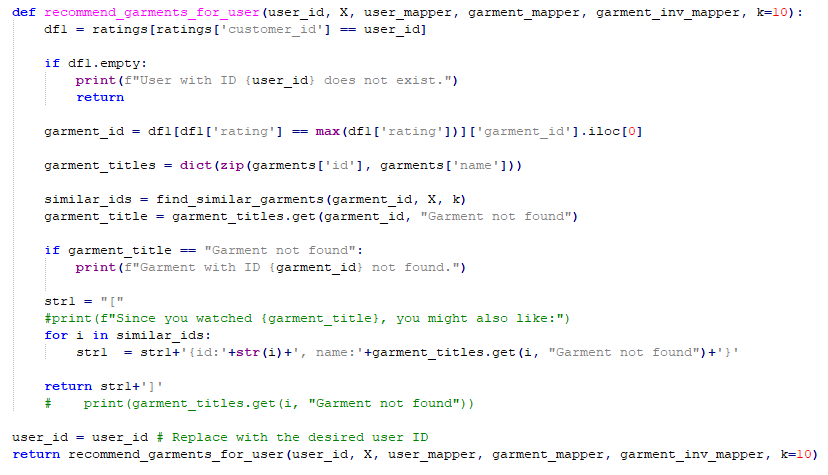


The provided code defines a function, “find\_similar\_garments,” which uses the k-Nearest Neighbors (KNN) algorithm to identify garments that are similar to a given garment. The function takes inputs such as the target garment ID, a user-item matrix (X), the number of neighbors to consider (k), a similarity metric (default is cosine similarity), and an option to show distances between garments. The function begins by initializing a blank list to hold the IDs of films that are comparable. It takes the target garment’s index out of the garment\_mapper dictionary and uses the user-item matrix to acquire the feature vector that goes with it. Next, the KNN model is configured using the given parameters.

The distances and indices of the k-nearest neighbors to the target garment are calculated once the KNN model has been fitted. Using the garment\_inv\_mapper dictionary, the loop retrieves these neighbor indices and maps them back to garment IDs. Since it matches the desired garment, the first item in the list is eliminated. The code ends with a list of related garment titles and the title of the target film, suggesting garments based on the KNN model.

Garment Recommendation with respect to Users Preference

Create a function to recomment the garments based on the user preferences.



The function accepts the following inputs: dictionaries (user\_mapper, garment\_mapper, and garment\_inv\_mapper) for mapping user and garment IDs to matrix indices; the user\_id for which recommendations are desired; a user-item matrix X representing garment ratings; and an optional parameter k for the number of recommended garments (default is 10).

It initially filters the ratings dataset to see if the user with the given ID is there. It notifies the user that the requested person does not exist and ends the function if the user does not exist (the filtered DataFrame is empty).

The code, if it exists, designates the garment that has received the highest rating from that particular user. It finds the garmentId of this garment and chooses it based on the highest rating.

With information from the garments dataset, a dictionary called garment\_titles is created to map garment IDs to their titles. The function then uses find\_similar\_garments to locate films that are comparable to the garment in the user-item matrix that has the highest rating (denoted by garment\_id). It gives back a list of comparable garment IDs.

The code searches the garment titles dictionary for the title of the highest-rated film, and if the film is not found, it sets the title to “Garment not found.” When a garment title is retrieved as “Garment not found,” it means that the highest-rated film (based on garment\_id) is not present in the dataset. If the garment is located, the customer is presented with recommendations for other garments based on the highest rated film. The list of comparable garment IDs is iterated over, and the titles are printed. When a garment isn’t discovered in the dataset, the default message is “Garment not found.”

The function handles situations where the user or garment doesn’t exist in the dataset and is intended to suggest garments for a particular user based on their highest-rated film. The code calls the function with the necessary parameters and sets the user\_id to a specific user to show how to utilize the method.Reccomment the garments



**Output:**

"[{id:12, name:Product 12}{id:11, name:Product 11}{id:3, name:Product 3}{id:7, name:Product 7}{id:9, name:Product 9}{id:8, name:Product 8}{id:6, name:Product 6}{id:2, name:Product 2}{id:1, name:Product 1}{id:5, name:Product 5}]"

****

**Output:**

User with ID 2300 does not exist.

**Conclusion**

In conclusion, developing a Python recommendation system allows for the creation of tailored content recommendations that improve user experience and take into account user preferences. Through the utilization of collaborative filtering, content-based filtering, and hybrid techniques, these systems are able to offer customized recommendations to consumers for content, garments, or items. These systems use sophisticated methods such as closest neighbors and matrix factorization to find hidden patterns in item attributes and user behavior. Recommendation systems are able to adjust and get better over time thanks to the combination of machine learning and data-driven insights. In the end, these solutions are essential for raising consumer satisfaction, improving user engagement, and propelling corporate expansion in a variety of industries.