

## **DS Experiment-8**

**Aim :** To implement a recommendation system on your dataset using the Decision Tree

**Theory :**

### **Types of Recommendation Systems**

A Recommendation System suggests relevant items to users based on their preferences, behavior, or other factors. There are several types of recommendation techniques:

#### ♦ **1. Content-Based Filtering**

- **Idea:** Recommends items similar to those the user has liked before.
- **Works on:** Item features (attributes such as brand, price, category).

##### **Example:**

- If a user buys a Samsung phone, they might be recommended another Samsung device based on brand preference.
- Uses techniques like TF-IDF (for text data), Cosine Similarity, Decision Trees, etc.

#### ♦ **2. Collaborative Filtering (CF)**

- **Idea:** Recommends items based on similar users' preferences.
- **Works on:** User interactions rather than item features.

##### **Example:**

- If User A and User B have similar purchase histories, items bought by User A but not yet by User B will be recommended to User B.
- Uses methods like User-Based CF and Item-Based CF.

### ♦ 3. Hybrid Recommendation System

- **Idea:** Combines Content-Based Filtering and Collaborative Filtering for better accuracy.  
**Example:**
- Netflix uses a hybrid approach, considering both user preferences and what similar users watch.

### ♦ 4. Knowledge-Based Recommendation

- **Idea:** Recommends items based on explicit domain knowledge rather than past user behavior.  
**Example:**
- A car recommendation system suggests vehicles based on engine type, price, and fuel efficiency, regardless of past purchases.

## 2. Recommendation System Evaluation Measures

Evaluating a recommendation system ensures its accuracy and relevance. Below are common evaluation metrics:

### ♦ 1. Accuracy-Based Metrics

#### (a) Precision:

- Measures how many of the recommended items are actually relevant.

- **Formula:**

$$Precision = \frac{Relevant\ Recommendations}{Total\ Recommendations}$$

- **Example:**

- If 5 out of 10 recommended items are relevant, Precision =  $5/10 = 0.5$  (50%).

**(b) Recall:**

- Measures how many of the relevant items are actually recommended.

- **Formula:**

$$Recall = \frac{Relevant\ Recommendations}{Total\ Relevant\ Items\ Available}$$

- **Example:**

- If a user liked 8 items, but only 5 were recommended, Recall =  $5/8 = 0.625$  (62.5%).

**(c) F1-Score:**

- A balance between Precision and Recall.

- **Formula:**

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

- Used when both Precision and Recall are important.

**(d) Accuracy:**

- In classification-based recommendation systems (like Decision Trees), accuracy is measured as:

$$Accuracy = \frac{Correct\ Predictions}{Total\ Predictions}$$

- In our Decision Tree model, if Accuracy = 1.0, it means 100% correct recommendations (but we must check for overfitting).

♦ **2. Ranking-Based Metrics**

These measure how well the recommendation system ranks items:

**(a) Mean Average Precision (MAP):**

- Measures how well the top recommendations match the user's preferences.

**(b) Normalized Discounted Cumulative Gain (NDCG):**

- Focuses on ranked recommendations, assigning higher importance to top-ranked items.

♦ **3. Diversity and Novelty Metrics**

- **Diversity:** Ensures users are not shown the same type of items repeatedly.
- **Novelty:** Measures if recommendations introduce new and unknown items.

## Implementation :

1.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

# Load dataset
file_path = "Dataset_Ds.csv"
df = pd.read_csv(file_path)

# Display basic info and first few rows
print(df.info())
print(df.head())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Region                 100000 non-null object
1   Country                100000 non-null object
2   Item Type              100000 non-null object
3   Sales Channel          100000 non-null object
4   Order Priority         100000 non-null object
5   Order Date             100000 non-null object
6   Order ID               100000 non-null int64
7   Ship Date              100000 non-null object
8   Units Sold             100000 non-null int64
9   Unit Price             100000 non-null float64
10  Unit Cost              100000 non-null float64
11  Total Revenue          100000 non-null float64
12  Total Cost             100000 non-null float64
13  Total Profit           100000 non-null float64
dtypes: float64(5), int64(2), object(7)
memory usage: 10.7+ MB
None
```

	Region	Country	Item Type \
0	Middle East and North Africa	Azerbaijan	Snacks
1	Central America and the Caribbean	Panama	Cosmetics
2	Sub-Saharan Africa	Sao Tome and Principe	Fruits
3	Sub-Saharan Africa	Sao Tome and Principe	Personal Care
4	Central America and the Caribbean	Belize	Household

  

	Sales Channel	Order Priority	Order Date	Order ID	Ship Date	Units Sold \
0	Online	C	10/8/2014	535113847	10/23/2014	934
1	Offline	L	2/22/2015	874708545	2/27/2015	4551
2	Offline	M	12/9/2015	854349935	1/18/2016	9986
3	Online	M	9/17/2014	892836844	10/12/2014	9118
4	Offline	H	2/4/2010	129280602	3/5/2010	5858

  

	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit
0	152.58	97.44	142509.72	91008.96	51500.76
1	437.20	263.33	1989697.20	1198414.83	791282.37
2	9.33	6.92	93169.38	69103.12	24066.26
3	81.73	56.67	745214.14	516717.06	228497.08
4	668.27	502.54	3914725.66	2943879.32	970846.34

The code loads the dataset `Dataset_Ds.csv` using Pandas and displays basic information about it. The `df.info()` function provides an overview of the dataset, including the number of entries, column names, data types, and memory usage. The `df.head()` function prints the first few rows to understand the dataset structure, which contains categorical (Region, Country, Item Type, etc.) and numerical (Unit Price, Total Revenue, Total Profit, etc.) features.

## 2.

```
[2] # Convert Categorical Data to Numeric using Label Encoding

# Selecting categorical columns
categorical_cols = ['Region', 'Country', 'Item Type', 'Sales Channel', 'Order Priority']

# Apply Label Encoding
label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
    label_encoders[col] = le
```

This code converts categorical data into numerical values using Label Encoding. First, it selects categorical columns: 'Region', 'Country', 'Item Type', 'Sales Channel', and 'Order Priority'. Then, it applies `LabelEncoder()` to each column, transforming categorical values into unique numerical labels. The encoded values allow machine learning models to process the data efficiently.

3.

```
[3] # Convert Dates to Numerical Format

# Convert date columns to datetime format
df['Order Date'] = pd.to_datetime(df['Order Date'], format='%m/%d/%Y')
df['Ship Date'] = pd.to_datetime(df['Ship Date'], format='%m/%d/%Y')

# Feature: Days taken to ship the order
df['Shipping Days'] = (df['Ship Date'] - df['Order Date']).dt.days

# Drop unnecessary columns
df.drop(['Order Date', 'Ship Date', 'Order ID'], axis=1, inplace=True)

# Display processed data
print(df.head())
```

	Region	Country	Item Type	Sales Channel	Order Priority	Units Sold	\
0	4	9	10	1	0	934	
1	2	124	4	0	2	4551	
2	6	139	5	0	3	9986	
3	6	139	9	1	3	9118	
4	2	15	6	0	1	5858	

  

	Unit Price	Unit Cost	Total Revenue	Total Cost	Total Profit	\
0	152.58	97.44	142509.72	91008.96	51500.76	
1	437.20	263.33	1989697.20	1198414.83	791282.37	
2	9.33	6.92	93169.38	69103.12	24066.26	
3	81.73	56.67	745214.14	516717.06	228497.08	
4	668.27	502.54	3914725.66	2943879.32	970846.34	

  

	Shipping Days
0	15
1	5
2	40
3	25
4	29

This code converts date columns into a numerical format for better processing. The Order Date and Ship Date columns are first converted into datetime format. Then, a new feature Shipping Days is created by calculating the difference between Ship Date and Order Date. Unnecessary columns (Order Date, Ship Date, Order ID) are dropped to reduce redundancy. Finally, the processed dataset is displayed, showing encoded categorical values and numerical data ready for machine learning.

4.

```
[4] # Split Data into Training & Testing Sets

# Define features (X) and target variable (y)
X = df.drop(columns=['Item Type']) # All columns except 'Item Type'
y = df['Item Type'] # Target variable

# Split data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

This code splits a dataset into training and testing sets. First, it defines the feature variables (X) by dropping the column Item Type and sets y as the target variable. Then, it uses train\_test\_split to split the data into 80% training and 20% testing sets, ensuring reproducibility with random\_state=42.

5.

```
✓ [5] # Train the Decision Tree Model
0s

# Initialize Decision Tree model
dt_model = DecisionTreeClassifier(max_depth=5, random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

print("Model training complete!")
```

➞ Model training complete!

This code trains a Decision Tree model. It initializes a DecisionTreeClassifier with a maximum depth of 5 and random\_state=42 for reproducibility. The model is then trained using the fit method on the training data (X\_train, y\_train). A message confirms successful training.

6.

```
✓ [6] # Make Predictions & Evaluate Accuracy
0s

# Make predictions
y_pred = dt_model.predict(X_test)

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy:.2f}")
```

➞ Model Accuracy: 0.83

This code makes predictions and evaluates model accuracy. It uses the trained DecisionTreeClassifier to predict labels for X\_test. The accuracy is then calculated



using `accuracy_score(y_test, y_pred)`. The result obtained is 0.83 which means the model has an accuracy of 83%.

7.

```
[7] # Extracting a real row from dataset (row=3) and checking it to predict item type

new_order = df.iloc[2, :-1].values.reshape(1, -1)
recommended_item = dt_model.predict(new_order)
print(f"Recommended Item: {label_encoders['Item Type'].inverse_transform(recommended_item)[0]}")
```

Recommended Item: Fruits  
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but DecisionT  
warnings.warn(

A row from the dataset (excluding the "Item Type") is fed into the trained Decision Tree model. The model predicts the "Item Type", which is then decoded from its numerical representation. The output confirms the model accurately predicts "Fruits" for the given row.

## Conclusion

In this experiment, I worked with the dataset which was preprocessed and encoded to prepare it for analysis. After splitting the data into features (X) and labels (Y), I applied a Decision Tree model to predict product categories. Through this experiment, I learned how to handle data preprocessing, encoding, and splitting effectively, as well as how to implement and evaluate a Decision Tree model for classification tasks