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:

o **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 :

To begin the analysis, essential Python libraries such as `pandas` and `numpy` were imported for data manipulation, along with various modules from `scikit-learn` for preprocessing, model training, clustering, dimensionality reduction, and ensemble learning. The dataset used for this project, titled "synthetic_cellphones_data_inr.csv", was loaded into a Pandas DataFrame using the `read_csv()` function. An initial data inspection was carried out using the `info()` method to review the structure of the dataset, confirm the absence of missing values, and verify that it contains 1000 records across 14 columns. Additionally, the `head()` function was used to preview the first five rows of the dataset and gain a basic understanding of its contents.

```
import numpy as np
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.cluster import KMeans
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import IsolationForest
from sklearn.decomposition import PCA
from sklearn.ensemble import VotingClassifier
```

```
# Load dataset
file_path = "/content/synthetic_cellphones_data_inr.csv"
df = pd.read_csv(file_path)

# Display initial dataset information
print("Dataset Info:")
print(df.info())  # Check column data types and missing values
print("\nFirst 5 rows of the dataset:")
print(df.head())  # Preview data
```

```
Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 14 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   cellphone_id      1000 non-null   int64
 1   brand             1000 non-null   object
 2   model             1000 non-null   object
 3   operating system  1000 non-null   object
 4   internal memory   1000 non-null   int64
 5   RAM               1000 non-null   int64
 6   performance       1000 non-null   float64
 7   main camera       1000 non-null   int64
 8   selfie camera     1000 non-null   int64
 9   battery size      1000 non-null   int64
 10  screen size       1000 non-null   float64
 11  weight            1000 non-null   int64
 12  price             1000 non-null   int64
```

2.

In the data preprocessing stage, the 'release date' column was converted into Unix timestamp format to facilitate numerical analysis. Subsequently, numerical and categorical columns were identified, and missing values were handled by imputing the mean for numerical attributes and the mode for categorical ones. Categorical variables were encoded using Label Encoding to transform them into numerical form. Finally, the numerical features were normalized using StandardScaler to ensure consistent scaling across attributes, which is essential for many machine learning algorithms.

```
# Convert 'release date' to numerical format (Unix timestamp)
df['release date'] = pd.to_datetime(df['release date'])
df['release date'] = df['release date'].astype(int) / 10**9  # Convert to seconds
```

```
[ ] # Identify numerical and categorical columns
    numeric_cols = ['internal memory', 'RAM', 'performance', 'main camera', 'selfie camera',
                    'battery size', 'screen size', 'weight', 'price', 'release date']
    categorical_cols = ['brand', 'model', 'operating system']
```

```
[ ] # Fill missing values
    df[numeric_cols] = df[numeric_cols].fillna(df[numeric_cols].mean())
    df[categorical_cols] = df[categorical_cols].fillna(df[categorical_cols].mode().iloc[0])
```

```
[ ] # Encode categorical columns
    for col in categorical_cols:
        df[col] = LabelEncoder().fit_transform(df[col])
```

```
[ ] # Normalize numerical features
    scaler = StandardScaler()
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
```

```
[ ] print("\nData after preprocessing:")
    print(df.head())  # Check transformed dataset
```

```
Data after preprocessing:
   cellphone id  brand  model  operating system  internal memory       RAM  \
```

```
Data after preprocessing:
   cellphone_id  brand  model  operating system  internal memory       RAM  \
0             0      0      0                 1         0.068869 -0.295823
1             1      2    182                 1        -0.679709  0.625744
2             2      1    102                 0        -0.679709 -0.295823
3             3      0     24                 0         0.068869 -1.217391
4             4      6    643                 1         0.068869  1.547312

   performance  main camera  selfie camera  battery size  screen size  \
0     0.935808    -1.299336       0.388087     -1.375113     0.476466
1     0.689773     0.211520       1.450693     -0.065272     0.862895
2     0.561835     1.489936      -1.028720     -0.831503     0.476466
3    -1.081677    -1.299336      -0.792586     -1.163709     0.862895
4    -0.638815     1.489936      -0.792586     -1.190458     1.635752

     weight     price  release date
0 -0.823815  1.510143      1.470020
1  0.839753 -1.532200     -1.013089
2  0.410445  0.531133      0.628666
3 -0.904311 -1.668716     -1.044250
4  0.732426 -0.894478      0.194188
```

3. Split the dataset

```
# Define features and target variable
X = df.drop(columns=['cellphone_id', 'price'])  # Features
y = df['price']  # Target variable

# Split data for training/testing (80% training, 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.

3.

```
# Train Linear Regression model
regressor = LinearRegression()
regressor.fit(X_train, y_train)

# Predict on test data
y_pred_regression = regressor.predict(X_test)
print("\nRegression Model trained.")
```

```
Regression Model trained.
```

```
# Convert price into categories
df['price_category'] = pd.qcut(df['price'], q=3, labels=['Low', 'Mid', 'High'])

# Split classification dataset
y_class = df['price_category']
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(X, y_class, test_size=0.2, random_state=42)

# Train Random Forest Classifier
classifier = RandomForestClassifier(n_estimators=100, random_state=42)
classifier.fit(X_train_class, y_train_class)

# Predict classification results
y_pred_classification = classifier.predict(X_test_class)
print("\nClassification Model trained.")
```

```
Classification Model trained.
```

```
# Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X)
print("\nClustering Model applied.")
```

```
Clustering Model applied.
```

```
# Train Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train_class, y_train_class)
print("\nDecision Tree Model trained.")
```

```
Decision Tree Model trained.
```

```
# Apply Isolation Forest for Anomaly Detection
anomaly_detector = IsolationForest(contamination=0.05, random_state=42)
df['Anomaly'] = anomaly_detector.fit_predict(X)
print("\nAnomaly Detection Model applied.")
```

```
Anomaly Detection Model applied.
```

```
# Apply PCA for Dimensionality Reduction
pca = PCA(n_components=5)
X_pca = pca.fit_transform(X)
print("\nDimensionality Reduction (PCA) applied.")
```

```
Dimensionality Reduction (PCA) applied.
```

```
# Train an Ensemble Model using Voting Classifier
ensemble_model = VotingClassifier(
    estimators=[
        ('rf', RandomForestClassifier(n_estimators=100, random_state=42)),
        ('dt', DecisionTreeClassifier(random_state=42))
    ], voting='hard'
)
ensemble_model.fit(X_train_class, y_train_class)
print("\nEnsemble Model trained.")
```

```
print(recommend_smartphone(ram=8, battery=5000, camera=48, price=30000, brand="OnePlus"))
```

```
No smartphones found with the given criteria.
```

```
print(recommend_smartphone(ram=6, battery=5727, camera=108, price=11500, brand="OnePlus"))
```

```
No smartphones found with the given criteria.
```

```
print(recommend_smartphone(ram=8, battery=4000, camera=48, price=40000, brand="Samsung"))
```

```
       brand              model  RAM  battery size  main camera  price
968  Samsung  Samsung Model 968   16          5708          108  23821
511  Samsung  Samsung Model 511   16          4738           48  36520
637  Samsung  Samsung Model 637   12          5082           64  36520
```

In this project, multiple machine learning models and techniques were implemented on a synthetic cellphone dataset to derive valuable insights and patterns. Initially, a **Linear Regression model** was trained to predict mobile phone prices based on various numerical features. To categorize the prices into discrete labels ('Low', 'Mid', 'High'), a **Random Forest Classifier** and a **Decision Tree Classifier** were employed.

For unsupervised learning, **K-Means Clustering** was used to group phones with similar characteristics, while **Isolation Forest** was applied for **anomaly detection**, helping identify outliers in the dataset. To reduce dimensionality and improve computational efficiency, **Principal Component Analysis (PCA)** was utilized.

Lastly, **cosine similarity** was calculated using normalized feature vectors to determine the degree of similarity between different mobile phones, which can be useful for recommendation or comparison systems.

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 Regression clustering, classification ,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