**EX NO:01 DATE:**

**IMPLEMENT DATA SIMILARITY MEASURES USING PYTHON**

**AIM**

To implement data similarity measures using python.

**PROCEDURE**

1. Install the required libraries.

2. Create a Python script.

3. Calculate the Jaccard similarity between set1 and set2

4. Calculate the cosine similarity between the documents using cosine\_similarity from scikit-learn.

5. Calculate the Euclidean distance between point1 and point2 using the np.linalg.norm function from NumPy.

6. Save the script.

7. Run the script.

**PROGRAM**

import numpy as np

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

# Jaccard Similarity

def jaccard\_similarity(set1, set2):

intersection = len(set1.intersection(set2))

union = len(set1.union(set2))

return intersection / union

set1 = {1, 2, 3}

set2 = {2, 3, 4}

jaccard\_similarity\_value = jaccard\_similarity(set1, set2)

print(f"Jaccard Similarity: {jaccard\_similarity\_value}")

# Cosine Similarity

documents = ["This is a sample document.",

"Another example document.",

"A third document for testing."]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(documents)

cosine\_matrix = cosine\_similarity(X, X)

print("Cosine Similarity Matrix:")

print(cosine\_matrix)

# Euclidean Distance

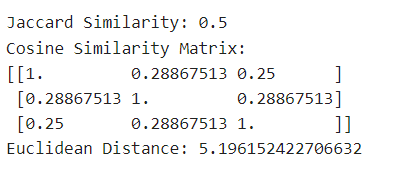
point1 = np.array([1, 2, 3])

point2 = np.array([4, 5, 6])

euclidean\_distance = np.linalg.norm(point1 - point2)

print(f"Euclidean Distance: {euclidean\_distance}")

**OUTPUT**

****

**RESULT**

Thus, the implementation of data similarity measures using python has been executed successfully.

**EX NO: 02 DATE:**

**IMPLEMENT DIMENSION REDUCTION TECHNIQUES FOR**

**RECOMMENDER SYSTEMS**

**AIM**

To implement dimension reduction techniques for recommender systems.

**PROCEDURE**

1. Create a user-item ratings matrix.
2. Perform SVD on the ratings matrix to reduce dimensions.
3. Initialize user and item matrices, specify iterations, and learning rate.
4. Update matrices using gradient descent and observed ratings.
5. Predict ratings using the learned user and item matrices.

**PROGRAM**

import numpy as np

# Sample user-item ratings matrix (user IDs on rows and item IDs on columns)

ratings = np.array([

[5, 4, 0, 0, 3],

[0, 4, 5, 0, 0],

[3, 0, 0, 0, 2],

[0, 0, 4, 5, 0]

])

# Singular Value Decomposition (SVD) for dimension reduction

# Perform SVD on the ratings matrix

U, S, VT = np.linalg.svd(ratings, full\_matrices=False)

# Select a reduced dimension (e.g., k) for dimension reduction

k = 2

# Create a k-rank approximation of the original matrix

U\_k = U[:, :k]

S\_k = np.diag(S[:k])

VT\_k = VT[:k, :]

# Reconstruct the ratings matrix with reduced dimensions

ratings\_reduced = np.dot(U\_k, np.dot(S\_k, VT\_k))

# Matrix Factorization for dimension reduction

# Number of users and items

num\_users, num\_items = ratings.shape

num\_features = 2 # Number of latent features (you can choose this)

# Randomly initialize user and item matrices

user\_matrix = np.random.rand(num\_users, num\_features)

item\_matrix = np.random.rand(num\_items, num\_features)

# Number of iterations for optimization

num\_iterations = 1000

learning\_rate = 0.01

for i in range(num\_iterations):

# Update user and item matrices using gradient descent

for user in range(num\_users):

for item in range(num\_items):

if ratings[user][item] > 0:

error = ratings[user][item] - np.dot(user\_matrix[user], item\_matrix[item])

for f in range(num\_features):

user\_matrix[user][f] += learning\_rate \* (2 \* error \* item\_matrix[item][f])

item\_matrix[item][f] += learning\_rate \* (2 \* error \* user\_matrix[user][f])

# Predict ratings using the learned user and item matrices

ratings\_predicted = np.dot(user\_matrix, item\_matrix.T)

# The 'ratings\_predicted' matrix contains the predicted ratings

print("Original Ratings:")

print(ratings)

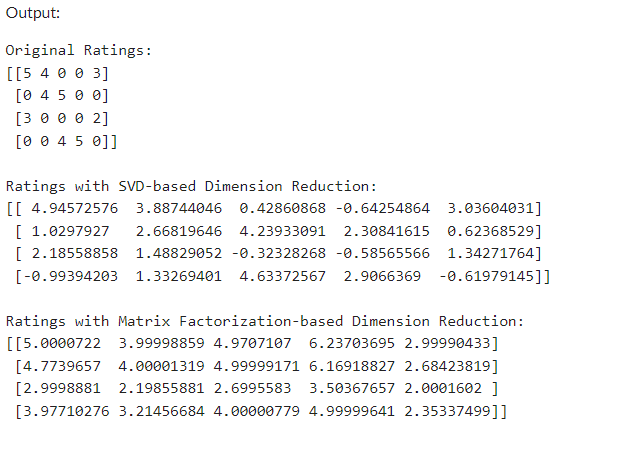
print("\nRatings with SVD-based Dimension Reduction:")

print(ratings\_reduced)

print("\nRatings with Matrix Factorization-based Dimension Reduction:")

print(ratings\_predicted)

**OUTPUT**

****

**RESULT**

Thus, the implementation of dimension reduction techniques for recommender systems has been executed successfully.

**EX NO: 03 DATE:**

**IMPLEMENT USER PROFILE LEARNING**

**AIM**

To implement user profile learning.

**PROCEDURE**

1. Define user-item interaction data with different ratings

2. Define item features with movie genres (example: two features for genres)

3. Implement user profile learning

4. Define a function to recommend items to a user based on user profile

5. Make personalized recommendations for a specific user.

6. Define movie genres

7. Print the personalized recommendations with genres for the user

**PROGRAM**

import numpy as np

# Define user-item interaction data with different ratings

user\_item\_matrix = np.array([

[4, 0, 0, 3, 0],

[0, 2, 0, 0, 4],

[0, 0, 3, 0, 0]

])

# Define item features with movie genres (example: two features for genres)

item\_features = np.array([

[0.7, 0.3], # Action

[0.3, 0.7], # Comedy

[0.6, 0.4], # Drama

[0.3, 0.7], # Sci-Fi

[0.4, 0.6], # Thriller

])

# Implement user profile learning

user\_profiles = np.dot(user\_item\_matrix, item\_features)

# Define a function to recommend items to a user based on user profile

def recommend\_items\_to\_user(user\_id, top\_n=4):

user\_profile = user\_profiles[user\_id]

item\_scores = np.dot(item\_features, user\_profile)

recommended\_items = np.arange(len(item\_features))

recommended\_items = [item for item in recommended\_items if user\_item\_matrix[user\_id][item] == 0]

recommended\_items = sorted(recommended\_items, key=lambda item: -item\_scores[item])[:top\_n]

return recommended\_items

# Make personalized recommendations for a specific user.

recommended\_items = recommend\_items\_to\_user(user\_id=0, top\_n=4)

# Define movie genres

movie\_genres = ["Action", "Comedy", "Drama", "Sci-Fi", "Thriller"]

# Print the personalized recommendations with genres for the user

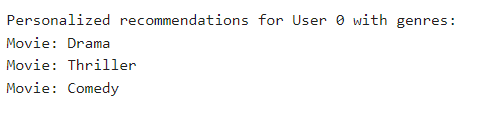
print(f"Personalized recommendations for User 0 with genres:")

for item in recommended\_items:

genre = movie\_genres[item]

print(f"Movie: {genre}")

**OUTPUT**

****

**RESULT**

Thus, the implementation of user profile learning has been executed successfully.

**EX NO: 04 DATE:**

**IMPLEMENT CONTENT-BASED RECOMMENDATION SYSTEMS**

**AIM**

To implement content-based recommendation systems.

**PROCEDURE**

1. Obtain a movie dataset with genres.

2. Create a DataFrame to store the movie data.

3. Create a user profile with genre preferences (1 for liked genres, 0 for disliked genres).

4. Apply one-hot encoding to movie genres (represent as 0 or 1).

5. Calculate cosine similarity between the user profile and movies.

6. Add similarity scores to the movie DataFrame.

7. Sort movies by similarity in descending order for recommendations.

8. Display top recommended movies based on user profile similarity.

**PROGRAM**

import pandas as pd

from sklearn.metrics.pairwise import cosine\_similarity

# Sample dataset (you should replace this with your movie dataset)

data = {

'MovieID': [1, 2, 3, 4, 5],

'Title': ['Movie A', 'Movie B', 'Movie C', 'Movie D', 'Movie E'],

'Genres': ['Action|Adventure', 'Comedy|Romance', 'Action|Sci-Fi', 'Drama', 'Horror|Thriller']

}

# Create a DataFrame from the dataset

movies\_df = pd.DataFrame(data)

# Create a user profile for a specific user (1 for liked genres, 0 for disliked genres)

user\_profile = {

'Action': 1,

'Adventure': 0,

'Comedy': 1,

'Romance': 0,

'Sci-Fi': 1,

'Drama': 0,

'Horror': 0,

'Thriller': 0

}

# Apply one-hot encoding to genres

genres = movies\_df['Genres'].str.get\_dummies('|')

movies\_df = pd.concat([movies\_df, genres], axis=1)

# Drop the original 'Genres' column

movies\_df.drop(columns=['Genres'], inplace=True)

# Calculate cosine similarity between user profile and movies

similarity\_scores = cosine\_similarity([list(user\_profile.values())], movies\_df.iloc[:, 2:])

# Add similarity scores to the movies DataFrame

movies\_df['Similarity'] = similarity\_scores[0]

# Sort the movies by similarity in descending order

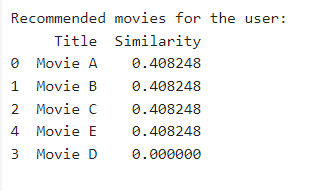
recommended\_movies = movies\_df.sort\_values(by='Similarity', ascending=False)

# Display the top recommended movies for the specific user

print(f"Recommended movies for the user:")

print(recommended\_movies[['Title', 'Similarity']])

**OUTPUT**

****

**RESULT**

Thus, the implementation of content-based recommendation systems has been executed successfully.

**EX NO: 05 DATE:**

**IMPLEMENT COLLABORATIVE FILTER TECHNIQUES**

**AIM**

To implement collaborative filter techniques.

**PROCEDURE**

1. Import the necessary libraries.

2. Define the sample user-item ratings dataset.

3. Convert the data into a Pandas DataFrame.

4. Define a function to perform user-based collaborative filtering.

5. Calculate the cosine similarity between users using the DataFrame.

6. Find the index of the target user in the DataFrame columns.

7. Get the similarity scores of the target user with other users.

8. Find the indices of users most similar to the target user, excluding the user itself.

9. Identify items rated by the target user.

10. Initialize an array to store recommendations.

11. Loop through the similar users and their ratings.

12. Calculate the recommendation score for items rated by similar users but not by the target user.

13. Sort the recommendations by their scores in descending order.

14. Get the top N recommendations.

15. Example usage: Specify the target user and the number of recommendations.

16. Display the recommended item indices.

**PROGRAM**

import numpy as np

import pandas as pd

from sklearn.metrics.pairwise import cosine\_similarity

# Sample user-item ratings dataset

data = {

'User1': [4, 5, 0, 0, 1],

'User2': [5, 4, 1, 0, 0],

'User3': [0, 0, 5, 4, 1],

'User4': [0, 0, 4, 5, 0],

'User5': [1, 0, 0, 0, 5]

}

# Convert the data into a DataFrame

df = pd.DataFrame(data)

# Function to calculate user-based collaborative filtering recommendations

def user\_based\_collaborative\_filtering(df, target\_user, num\_recommendations=5):

# Calculate the cosine similarity between users

similarity = cosine\_similarity(df)

# Find the index of the target user

user\_index = df.columns.get\_loc(target\_user)

# Get the similarity scores of the target user with other users

user\_similarity = similarity[user\_index]

# Find the indices of users most similar to the target user

similar\_users\_indices = np.argsort(user\_similarity)[::-1][1:] # Exclude the user itself

# Find items rated by similar users but not by the target user

items\_rated\_by\_target\_user = df[target\_user] > 0

recommendations = np.zeros(df.shape[1])

for index in similar\_users\_indices:

similar\_user\_id = df.columns[index]

user\_ratings = df[similar\_user\_id]

similar\_user\_similarity = user\_similarity[index]

for item in range(df.shape[1]):

if not items\_rated\_by\_target\_user[item] and user\_ratings[item] > 0:

recommendations[item] += user\_ratings[item] \* similar\_user\_similarity

# Sort the recommendations by the scores in descending order

recommended\_items = np.argsort(recommendations)[::-1]

# Get the top N recommendations

top\_recommendations = recommended\_items[:num\_recommendations]

return top\_recommendations

# Example usage

target\_user = 'User1'

recommended\_items = user\_based\_collaborative\_filtering(df, target\_user, num\_recommendations=3)

# Display the recommended item indices

print(f"Recommended items for {target\_user}: {recommended\_items}")

**OUTPUT**

****

**RESULT**

Thus, the implementation of collaborative filter techniques has been executed successfully.

**EX NO: 07 DATE:**

**IMPLEMENT ACCURACY METRICS LIKE RECEIVER OPERATING CHARACTERISTIC CURVES**

**AIM**

To implement accuracy metrics like Receiver Operating Characteristic curves.

**PROCEDURE**

1. Example user-item ratings (can be a user-item matrix)

2. Define a relevance threshold (e.g., ratings >= 3 are relevant)

3. Convert ratings to binary relevance (1 for relevant, 0 for non-relevant)

4. Example predicted scores (confidence scores) for items

5. Flatten the binary ratings and predicted scores

6. Calculate ROC curve

7. Calculate AUC

8. Calculate Mean Average Precision (MAP)

8.1. Calculate average precision for a single user

8.2. Calculate mean average precision for all users

9. Example actuals for MAP

10. Example predicted lists for MAP

11. Calculate Normalized Discounted Cumulative Gain (NDCG)

11.1. Calculate discounted cumulative gain at k

11.2. Calculate normalized discounted cumulative gain at k

12. Example actual and predicted for NDCG

13. Calculate Precision at K and Recall at K

13.1. Calculate precision at k

13.2. Calculate recall at k

14. Example values for Precision at K and Recall at K

15. Print all metrics

16. Plot ROC curve

**PROGRAM**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import roc\_curve, auc

# Example user-item ratings (can be a user-item matrix)

ratings = np.array([[4, 5, 0, 1, 0],

[0, 2, 3, 0, 0],

[4, 0, 0, 0, 1],

[0, 0, 2, 4, 0]])

# Define a relevance threshold (e.g., ratings >= 3 are relevant)

threshold = 3

# Convert ratings to binary relevance (1 for relevant, 0 for non-relevant)

binary\_ratings = (ratings >= threshold).astype(int)

# Example predicted scores (confidence scores) for items

predicted\_scores = np.array([[0.8, 0.9, 0.2, 0.6, 0.1],

[0.1, 0.5, 0.7, 0.3, 0.2],

[0.9, 0.1, 0.3, 0.4, 0.8],

[0.2, 0.4, 0.6, 0.8, 0.1]])

# Flatten the binary ratings and predicted scores

binary\_ratings\_flat = binary\_ratings.flatten()

predicted\_scores\_flat = predicted\_scores.flatten()

# Calculate ROC curve

fpr, tpr, thresholds = roc\_curve(binary\_ratings\_flat, predicted\_scores\_flat)

# Calculate AUC

roc\_auc = auc(fpr, tpr)

# Calculate Mean Average Precision (MAP)

def average\_precision(actual, predicted):

precision = 0.0

num\_hits = 0

for i, p in enumerate(predicted):

if p in actual and p not in predicted[:i]:

num\_hits += 1

precision += num\_hits / (i + 1)

if not actual:

return 0.0

return precision / len(actual)

def mean\_average\_precision(actuals, predicted\_list):

average\_precisions = [average\_precision(a, p) for a, p in zip(actuals, predicted\_list) if a]

return sum(average\_precisions) / len(average\_precisions)

# Example actuals for MAP

actuals = [[1, 2, 3], [1, 4], [2, 4, 5]]

# Example predicted lists for MAP

predicted\_list = [[1, 2, 3, 4, 5], [1, 2, 3], [1, 2, 3, 4]]

map\_score = mean\_average\_precision(actuals, predicted\_list)

# Calculate Normalized Discounted Cumulative Gain (NDCG)

def dcg\_at\_k(ranks, k):

return np.sum([(2 \*\* rank - 1) / np.log2(rank + 1) for rank in ranks[:k]])

def ndcg\_at\_k(actual, predicted, k):

actual\_ranks = [predicted.index(a) + 1 if a in predicted else 0 for a in actual]

ideal\_ranks = sorted(actual\_ranks, reverse=True)

return dcg\_at\_k(actual\_ranks, k) / dcg\_at\_k(ideal\_ranks, k)

# Example actual and predicted for NDCG

actual = [1, 2, 3]

predicted = [1, 3, 2, 4, 5]

k = 3

ndcg\_score = ndcg\_at\_k(actual, predicted, k)

# Calculate Precision at K and Recall at K

def precision\_at\_k(actual, predicted, k):

if k == 0:

return 0.0

intersection = len(set(predicted[:k]) & set(actual))

return intersection / k

def recall\_at\_k(actual, predicted, k):

if len(actual) == 0:

return 0.0

intersection = len(set(predicted[:k]) & set(actual))

return intersection / len(actual)

# Example values for Precision at K and Recall at K

k = 3

precision\_at\_k\_score = precision\_at\_k(actual, predicted, k)

recall\_at\_k\_score = recall\_at\_k(actual, predicted, k)

# Print all metrics

print("AUC:", roc\_auc)

print("Mean Average Precision (MAP):", map\_score)

print(f"NDCG at K ({k}):", ndcg\_score)

print(f"Precision at K ({k}):", precision\_at\_k\_score)

print(f"Recall at K ({k}):", recall\_at\_k\_score)

# Plot ROC curve

plt.figure()

plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

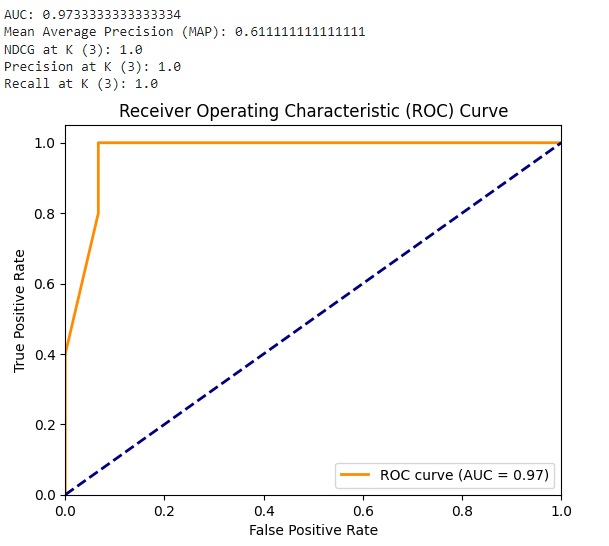
plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic (ROC) Curve')

plt.legend(loc='lower right')

plt.show()

**OUTPUT**

****

**RESULT**

Thus, the implementation accuracy metrics like Receiver Operated Characteristic curves has been executed successfully.

**EX NO: 06 DATE:**

**CREATE AN ATTACK FOR TAMPERING WITH RECOMMENDER SYSTEMS**

**AIM**

To create an attack for tampering with recommender systems.

**PROCEDURE**

1. Import necessary libraries

2. Load the Movie ratings dataset with Dask

3. Sample a smaller subset of the data

4. Convert 'Rating' to float32

5. Create a sparse user-item matrix

6. Initialize the recommender system with the user-item matrix

7. Implement the tampering attack

8. Initialize the recommender system

9. Display some details from the original recommender system

10. Display a subset of the original user-item matrix

11. Execute the tampering attack

12. Display some details from the tampered recommender system

13. Display a subset of the tampered user-item matrix

**PROGRAM**

import pandas as pd

import numpy as np

import dask.dataframe as dd

from scipy.sparse import coo\_matrix

from scipy.sparse.linalg import svds

data = dd.read\_csv('Netflix\_Dataset\_Rating.csv/Netflix\_Dataset\_Rating.csv')

data = data.sample(frac=0.01, random\_state=42).compute()

data['Rating'] = data['Rating'].astype(np.float32)

user\_item\_matrix = coo\_matrix((data['Rating'], (data['User\_ID'], data['Movie\_ID'])))

class RecommenderSystem:

def \_\_init\_\_(self, user\_item\_matrix):

self.user\_item\_matrix = user\_item\_matrix

def get\_weights(self):

\_, \_, vt = svds(self.user\_item\_matrix.astype(float), k=20)

return vt.T

def set\_weights(self, weights):

self.user\_item\_matrix = coo\_matrix(weights @ weights.T)

def tampering\_attack(recommender\_system, attack\_strength):

weights = recommender\_system.get\_weights()

num\_dimensions = len(weights)

magnitude = attack\_strength / num\_dimensions

modified\_weights = weights + np.random.normal(0, magnitude, size=weights.shape)

recommender\_system.set\_weights(modified\_weights)

recommender\_system = RecommenderSystem(user\_item\_matrix)

original\_weights = recommender\_system.get\_weights()

print("\nDetails from the original recommender system:")

print("Original weights shape:", original\_weights.shape)

print("Original weights mean:", np.mean(original\_weights))

print("Original weights standard deviation:", np.std(original\_weights))

print("\nOriginal user-item matrix (subset):")

print("Non-zero entries in the user-item matrix:", user\_item\_matrix.getnnz())

print("Maximum value in the user-item matrix:", user\_item\_matrix.max())

print("Minimum value in the user-item matrix:", user\_item\_matrix.min())

attack\_strength = 0.5

tampering\_attack(recommender\_system, attack\_strength)

tampered\_weights = recommender\_system.get\_weights()

print("\nDetails from the tampered recommender system:")

print("Tampered weights shape:", tampered\_weights.shape)

print("Tampered weights mean:", np.mean(tampered\_weights))

print("Tampered weights standard deviation:", np.std(tampered\_weights))

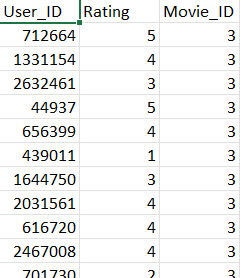
print("\nTampered user-item matrix (subset):")

print("Non-zero entries in the user-item matrix:", recommender\_system.user\_item\_matrix.getnnz())

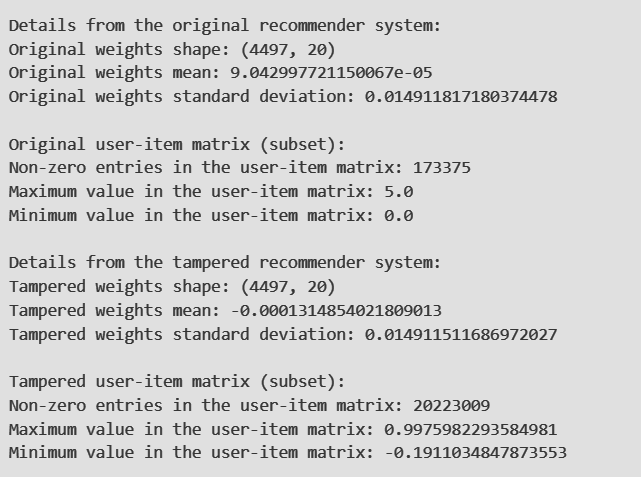
print("Maximum value in the user-item matrix:", recommender\_system.user\_item\_matrix.max())

print("Minimum value in the user-item matrix:", recommender\_system.user\_item\_matrix.min())

**DATASET**

****

**OUTPUT**



**RESULT**

Thus, the creation of an attack for tampering with recommender systems has been executed successfully.