**PROJECT 1:**

**TASK SELECTED:**

1. Build a recommender system that recommends books to read for every user based on their personal tastes and previous book ratings.

**CODE:**

import numpy as np

import pandas as pd

import os

from sklearn.datasets import load\_svmlight\_file

def calculate\_sparse\_cosine\_similarity(team\_32\_vector, team\_32\_matrix):

"""

Calculate cosine similarity between a single vector and all rows of a sparse matrix.

Parameters:

team\_32\_vector (csr\_matrix): A sparse vector (row) for comparison.

team\_32\_matrix (csr\_matrix): Sparse matrix where each row represents a different entity.

Returns:

numpy.ndarray: Array of cosine similarity scores for each row in the matrix.

"""

team\_32\_similarity = team\_32\_matrix.dot(team\_32\_vector.T).toarray().ravel()

team\_32\_vector\_norm = np.sqrt(team\_32\_vector.multiply(team\_32\_vector).sum())

team\_32\_row\_norms = np.sqrt(team\_32\_matrix.power(2).sum(axis=1)).A1

team\_32\_scores = team\_32\_similarity / (team\_32\_vector\_norm \* team\_32\_row\_norms + 1e-10)

return team\_32\_scores

def recommend\_global\_top\_books(team\_32\_sparse\_matrix, team\_32\_count=10):

"""

Recommend globally popular books based on average ratings across all users.

Parameters:

team\_32\_sparse\_matrix (csr\_matrix): Sparse matrix of user-item ratings.

team\_32\_count (int): Number of top books to return.

Returns:

list: Top books as tuples [(book\_id, average\_rating), ...].

"""

team\_32\_avg\_scores = team\_32\_sparse\_matrix.mean(axis=0).A1

team\_32\_top\_indices = np.argsort(team\_32\_avg\_scores)[-team\_32\_count:][::-1]

team\_32\_global\_books = [(idx, team\_32\_avg\_scores[idx]) for idx in team\_32\_top\_indices]

return team\_32\_global\_books

def generate\_book\_recommendations(team\_32\_sparse\_matrix, team\_32\_similar\_users, team\_32\_target\_user, team\_32\_top\_k=10):

"""

Generate book recommendations for a user using their top-K similar users.

Parameters:

team\_32\_sparse\_matrix (csr\_matrix): Sparse user-item ratings matrix.

team\_32\_similar\_users (list): List of tuples [(user\_id, similarity\_score), ...].

team\_32\_target\_user (int): ID of the user to recommend books for.

team\_32\_top\_k (int): Number of similar users to consider.

Returns:

list: Recommended books as tuples [(book\_id, predicted\_score), ...].

"""

team\_32\_user\_books = set(team\_32\_sparse\_matrix.getrow(team\_32\_target\_user).nonzero()[1])

team\_32\_all\_books = set()

for team\_32\_user\_id, \_ in team\_32\_similar\_users:

team\_32\_all\_books.update(team\_32\_sparse\_matrix.getrow(team\_32\_user\_id).nonzero()[1])

team\_32\_recommend\_pool = team\_32\_all\_books - team\_32\_user\_books

team\_32\_predicted\_scores = {}

for team\_32\_book in team\_32\_recommend\_pool:

team\_32\_total\_score = 0

team\_32\_weight\_sum = 0

for team\_32\_user\_id, team\_32\_similarity in team\_32\_similar\_users:

team\_32\_book\_rating = team\_32\_sparse\_matrix[team\_32\_user\_id, team\_32\_book]

if team\_32\_book\_rating > 0:

team\_32\_total\_score += team\_32\_similarity \* team\_32\_book\_rating

team\_32\_weight\_sum += team\_32\_similarity

if team\_32\_weight\_sum == 0:

team\_32\_predicted\_scores[team\_32\_book] = team\_32\_sparse\_matrix[:, team\_32\_book].mean()

else:

team\_32\_predicted\_scores[team\_32\_book] = team\_32\_total\_score / team\_32\_weight\_sum

team\_32\_personalized\_books = sorted(team\_32\_predicted\_scores.items(), key=lambda x: x[1], reverse=True)[:5]

if len(team\_32\_personalized\_books) < 5:

team\_32\_needed\_books = 5 - len(team\_32\_personalized\_books)

team\_32\_global\_books = recommend\_global\_top\_books(

team\_32\_sparse\_matrix,

team\_32\_count=team\_32\_needed\_books + len(team\_32\_personalized\_books)

)

team\_32\_existing\_books = {book\_id for book\_id, \_ in team\_32\_personalized\_books}

team\_32\_additional\_books = [

(book\_id, score) for book\_id, score in team\_32\_global\_books

if book\_id not in team\_32\_existing\_books

]

team\_32\_additional\_books = team\_32\_additional\_books[:team\_32\_needed\_books]

team\_32\_personalized\_books.extend(team\_32\_additional\_books)

return team\_32\_personalized\_books

def find\_top\_similar\_users\_sparse(team\_32\_matrix, team\_32\_user\_id, team\_32\_top\_k=10):

"""

Identify the top-K most similar users for a specified user based on cosine similarity.

Parameters:

team\_32\_matrix (csr\_matrix): Sparse matrix of user-item ratings.

team\_32\_user\_id (int): The user ID for whom similarity is calculated.

team\_32\_top\_k (int): The number of similar users to retrieve.

Returns:

list: Top-K similar users as tuples [(user\_id, similarity\_score), ...].

"""

team\_32\_user\_vector = team\_32\_matrix.getrow(team\_32\_user\_id)

team\_32\_similarities = calculate\_sparse\_cosine\_similarity(team\_32\_user\_vector, team\_32\_matrix)

team\_32\_similarities[team\_32\_user\_id] = -1

team\_32\_top\_indices = np.argsort(team\_32\_similarities)[-team\_32\_top\_k:][::-1]

team\_32\_similar\_users = [(index, team\_32\_similarities[index]) for index in team\_32\_top\_indices]

return team\_32\_similar\_users

def process\_user\_chunk\_and\_save(

team\_32\_data\_file, team\_32\_user\_map\_file, team\_32\_book\_map\_file,

team\_32\_titles\_file, team\_32\_result\_file, team\_32\_user\_group, team\_32\_num\_similar=10):

"""

Process a chunk of users and save their book recommendations to a file, including titles.

Parameters:

team\_32\_data\_file (str): Path to the LIBSVM file.

team\_32\_user\_map\_file (str): Path to the user mapping file.

team\_32\_book\_map\_file (str): Path to the book mapping file.

team\_32\_titles\_file (str): Path to the file containing book titles.

team\_32\_result\_file (str): Path to save the recommendations.

team\_32\_user\_group (list): List of user IDs to process.

team\_32\_num\_similar (int): Number of similar users to consider for recommendations.

Returns:

None

"""

team\_32\_matrix, \_ = load\_svmlight\_file(team\_32\_data\_file)

team\_32\_user\_map = pd.read\_csv(team\_32\_user\_map\_file)

team\_32\_book\_map = pd.read\_csv(team\_32\_book\_map\_file)

team\_32\_titles = pd.read\_csv(team\_32\_titles\_file, delimiter=';')

team\_32\_output = []

for team\_32\_user in team\_32\_user\_group:

team\_32\_user\_row = team\_32\_user\_map[team\_32\_user\_map['Mapped-ID'] == team\_32\_user]

if team\_32\_user\_row.empty:

continue

team\_32\_resolved\_user = team\_32\_user\_row['User-ID'].values[0]

team\_32\_similar\_users = find\_top\_similar\_users\_sparse(team\_32\_matrix, team\_32\_user, team\_32\_num\_similar)

team\_32\_recommendations = generate\_book\_recommendations(team\_32\_matrix, team\_32\_similar\_users, team\_32\_user)

for team\_32\_book\_id, team\_32\_score in team\_32\_recommendations:

team\_32\_book\_row = team\_32\_book\_map[team\_32\_book\_map['Mapped-ID'] == team\_32\_book\_id]

if team\_32\_book\_row.empty:

team\_32\_isbn = "Unknown ISBN"

else:

team\_32\_isbn = team\_32\_book\_row['ISBN'].values[0]

team\_32\_title\_row = team\_32\_titles[team\_32\_titles['ISBN'] == team\_32\_isbn]

if team\_32\_title\_row.empty:

team\_32\_book\_title = "Unknown Title"

else:

team\_32\_book\_title = team\_32\_title\_row['Title'].values[0]

team\_32\_output.append({

'User\_ID': team\_32\_resolved\_user,

'Book\_ID': team\_32\_isbn,

'Book\_Title': team\_32\_book\_title,

'Recommendation\_Score': team\_32\_score

})

team\_32\_df = pd.DataFrame(team\_32\_output)

if not os.path.isfile(team\_32\_result\_file):

team\_32\_df.to\_csv(team\_32\_result\_file, index=False)

else:

team\_32\_df.to\_csv(team\_32\_result\_file, mode='a', header=False, index=False)

input\_file = "user\_book\_matrix.libsvm"

output\_file = "output.libsvm"

with open(input\_file, "r") as infile:

lines = infile.readlines()

with open(output\_file, "w") as outfile:

for index, line in enumerate(lines):

features = line.strip()

if features:

label = index

libsvm\_line = f"{label} {features}"

outfile.write(libsvm\_line + "\n")

print(f"LIBSVM file created successfully: {output\_file}")

data\_file = "output.libsvm"

user\_mapping\_file = "user\_mapping.csv"

book\_mapping\_file = "book\_mapping.csv"

titles\_file = "Books.csv"

result\_file = "final\_recommendations.csv"

from sklearn.datasets import load\_svmlight\_file

import numpy as np

sparse\_matrix, \_ = load\_svmlight\_file(data\_file)

chunk\_size = 100

total\_users = sparse\_matrix.shape[0]

user\_chunks = [range(i, min(i + chunk\_size, total\_users)) for i in range(0, total\_users, chunk\_size)]

import os

if os.path.exists(result\_file):

os.remove(result\_file)

for i, user\_chunk in enumerate(user\_chunks):

try:

print(f"Processing chunk {i + 1}/{len(user\_chunks)} with {len(user\_chunk)} users...")

process\_user\_chunk\_and\_save(

team\_32\_data\_file=data\_file,

team\_32\_user\_map\_file=user\_mapping\_file,

team\_32\_book\_map\_file=book\_mapping\_file,

team\_32\_titles\_file=titles\_file,

team\_32\_result\_file=result\_file,

team\_32\_user\_group=user\_chunk,

team\_32\_num\_similar=10

)

except Exception as e:

print(f"Error processing chunk {i + 1}: {e}")

**SCREENSHOT:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**CODE EXPLANATIONS:**

Here's a breakdown of the code, with an explanation of each function, its inputs, outputs, and the task it performs:

**1. calculate\_sparse\_cosine\_similarity**

**Inputs:**

* team\_32\_vector: A sparse vector (row) representing a user's ratings.
* team\_32\_matrix: A sparse matrix where each row represents another user's ratings.

**Outputs:**

* team\_32\_scores: An array of cosine similarity scores between the input vector and each row in the matrix.

**Task:** This function calculates the **cosine similarity** between a given user vector and all other user vectors in the matrix using the formula:

cosine similarity= dot product of vectors​/ ∥vector1∥⋅∥vector2∥

It returns the similarity scores as an array, which can be used to find the most similar users.

**2. find\_top\_similar\_users\_sparse**

**Inputs:**

* team\_32\_matrix: Sparse matrix of user-item ratings.
* team\_32\_user\_id: ID of the user for whom similar users are identified.
* team\_32\_top\_k: Number of top similar users to retrieve.

**Outputs:**

* team\_32\_similar\_users: A list of tuples [(user\_id, similarity\_score), ...] representing the most similar users.

**Task:** This function calculates the cosine similarity between a specific user and all other users. It identifies the **top K similar users** with the highest similarity scores, excluding the target user. The value of k is equal to 10 in this case

**3. recommend\_global\_top\_books**

**Inputs:**

* team\_32\_sparse\_matrix: A sparse matrix of user-item ratings.
* team\_32\_count: Number of top books to return (default is 10).

**Outputs:**

* team\_32\_global\_books: A list of tuples [book\_id, average\_rating), ...] representing the most popular books and their average ratings.

**Task:** This function calculates the **average rating** for each book across all users and sorts them in descending order. It selects the top books based on the average ratings, representing globally popular recommendations.

**4. generate\_book\_recommendations**

**Inputs:**

* team\_32\_sparse\_matrix: Sparse user-item ratings matrix.
* team\_32\_similar\_users: List of tuples [(user\_id, similarity\_score), ...] representing similar users.
* team\_32\_target\_user: ID of the target user for whom recommendations are generated.
* team\_32\_top\_k: Number of similar users to consider for recommendations.

**Outputs:**

* team\_32\_personalized\_books: A list of recommended books as tuples [(book\_id, predicted\_score), ...].

**Task:** This function predicts personalized book recommendations for a user. It:

1. Identifies books that similar users have rated but the target user has not.
2. Calculates predicted scores for these books using weighted averages of ratings by similar users. The below formula to used to select the top 5 books based on weighted average.

A mathematical equation with numbers and symbols

Description automatically generated

1. If the denominator is zero then the predicted scores as set as the global average rating for the book.
2. If fewer than 5 books are found, it fills the list with globally popular books.

**5. process\_user\_chunk\_and\_save**

**Inputs:**

* team\_32\_data\_file: Path to the LIBSVM file containing sparse user-item ratings.
* team\_32\_user\_map\_file: CSV file mapping internal user IDs to external IDs.
* team\_32\_book\_map\_file: CSV file mapping internal book IDs to ISBNs.
* team\_32\_titles\_file: CSV file containing book titles and their ISBNs.
* team\_32\_result\_file: Path to save the recommendations.
* team\_32\_user\_group: List of user IDs to process.
* team\_32\_num\_similar: Number of similar users to consider for recommendations.

**Outputs:**

* None (saves recommendations to the specified file).

**Task:** This function processes a batch of users to generate personalized book recommendations. It:

1. Reads sparse user-item ratings, user mappings, and book details.
2. Finds top similar users for each target user in the batch.
3. Generate personalized recommendations for each user.
4. Resolves book details (e.g., ISBN and titles) and saves the results to a CSV file.

**Main Script**

1. Convert the user\_book\_matrix.libsvm dataset with the user\_id labels so that it can be directly loaded using load\_svmlight\_file
2. **Load the dataset:** The LIBSVM file is read into a sparse matrix, representing user-item ratings.
3. **Divide users into chunks:** The users are divided into smaller groups (chunk\_size = 100) for batch processing.
4. **Process each chunk:** For each chunk, the script:
   * Finds similar users.
   * Generates recommendations.
   * Resolves book titles and saves results to a file.

This script is designed to build a personalized recommendation system based on collaborative filtering using sparse cosine similarity, with fallback global recommendations for diversity.