كليـــــة العلـــــوم
والتقنيات - مراكش
FACULTE DES SCIENCES
ET TECHNIQUES - MARRAKECH

Master IAII
Pr. Sara QASSIMI

Tutorial: Building a User-Based Collaborative Filtering Recommendation System

Step 1: Imports and Data Loading

Let's start by importing the required libraries and loading the preprocessed data. These libraries will enable us to manipulate and analyze the data efficiently.

```
# Import necessary libraries
from __future__ import print_function, division
from builtins import range, input
import pickle
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from datetime import datetime
from sortedcontainers import SortedList

# Load data from pickled files
with open('user2movie.json', 'rb') as f:
  user2movie = pickle.load(f)
# ... similar loading for other data files
```

Step 2: Data Preprocessing and Initialization

We determine the number of users and movies in the dataset, and then perform some checks. This step helps us prepare the required variables and parameters for further analysis.

- N is the number of users, so np.max(list(user2movie.keys())) gives us the highest user ID.
- m1 is the maximum movie ID found in movie2user, and m2 is the maximum movie ID found in usermovie2rating_test. We take the maximum of these two to ensure we consider all movies.

```
N = np.max(list(user2movie.keys())) + 1
m1 = np.max(list(movie2user.keys()))
m2 = np.max([m for (u, m), r in usermovie2rating_test.items()])
M = max(m1, m2) + 1
print("N:", N, "M:", M)

if N > 10000:
  print("N =", N, "are you sure you want to continue?")
  exit()

K = 25
limit = 5
neighbors = []
averages = []
deviations = []
```

Step 3: Finding Neighbors

This step involves iterating through each user to find their nearest neighbors. We calculate the Pearson correlation coefficient between users and store the most similar users as neighbors.

```
for i in range(N):
  # Find movies rated by user i
  movies_i = user2movie[i]
  # ... calculate user's average rating and deviations
  sl = SortedList()
  for j in range(N):
    # Calculate common movies between users i and j
    # ... calculate user j's average rating and deviations
    # ... calculate correlation coefficient w_ij
    # ... insert w_ij into the sorted list of neighbors
    # ... truncate the list to maintain K neighbors
  neighbors.append(sl)
```

Step 4: Prediction and MSE Calculation

In this step, we define a function to predict ratings for user-movie pairs. We iterate through training and test data to calculate predictions, and then we compute the Mean Squared Error (MSE) to evaluate the performance of our recommendation system.

```
def predict(i, m):
  # Calculate the weighted sum of deviations for prediction
  # ... iterate through neighbors and calculate prediction
  return prediction

train_predictions = []
train_targets = []
# ... loop through user-movie pairs in training set
# ... calculate predictions and save predictions and targets

test_predictions = []
test_targets = []
# ... similar loop for test set

# Calculate Mean Squared Error (MSE)
def mse(p, t):
  # ... calculate MSE
  return mse

print('train mse:', mse(train_predictions, train_targets))
print('test mse:', mse(test_predictions, test_targets))
```

Step 5: Generating Recommendations

Finally, we demonstrate how to generate movie recommendations for a subset of users and save the recommendations to a CSV file.

```python
user_indices_for_recommendation = np.random.choice(N, size=10, replace=False

recommendations = []
for user_idx in user_indices_for_recommendation:
    recommended_movies = []
    for movie_idx in range(M):
        # ... calculate prediction for this user-movie pair
        recommended_movies.append((movie_idx, prediction, actual_rating))

    recommendations.append((user_idx, recommended_movies))

# Save recommendations to a CSV file
# ... create a DataFrame and save to CSV
```