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

Master IAII
Pr. Sara QASSIMI

Tutorial: Building a Item-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

In this step, we determine the number of users and items (movies) in the dataset. We also set parameters and initialize lists for storing data.

```
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 M > 2000:
  print("N =", N, "are you sure you want to continue?")
  print("Comment out these lines if so...")
  exit()

K = 20
limit = 5
neighbors = []
averages = []
deviations = []
```

Step 3: Finding Neighbors

In this step, we compute similarities between items based on user ratings and identify the most similar items for each item. This helps us create a neighborhood of similar items.

```
for i in range(M):
  # Find users who rated item i
  users_i = movie2user[i]
  # ... calculate item's average rating and deviations
  sl = SortedList()
  for j in range(M):
    # Calculate common users who rated both items i and j
    # ... calculate item 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

This step involves defining a function to predict user ratings for item-user pairs. We iterate through the training and test data to calculate predictions, and then we compute the Mean Squared Error (MSE) to evaluate the system's performance.

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

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

test_predictions = []
test_targets = []
# ... similar loop for the 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

Lastly, we showcase how to generate item recommendations for a subset of users and save these 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_items = []
    for item_idx in range(M):
        # ... calculate prediction for this user-item pair
        recommended_items.append((item_idx, prediction, actual_rating))

    recommendations.append((user_idx, recommended_items))

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