```
//Deepak Shinde (Vyorius test )
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>
#include <algorithm>

using namespace std;

// Function to calculate cosine similarity between two users
double cosineSimilarity(const vector<int>& user1, const vector<int>& user2) {
int dotProduct = 0;
int magnitude1 = 0;
int magnitude2 = 0;

for (size_t i = 0; i < user1.size(); ++i) {
dotProduct += user1[i] * user2[i];
magnitude1 += user1[i] * user1[i];
magnitude2 += user2[i] * user2[i];
}

if (magnitude1 == 0 || magnitude2 == 0) return 0.0;

return dotProduct / (sqrt(magnitude1) * sqrt(magnitude2));
}

// Function to predict ratings for all unrated movies for a user
vector<double> predictRatings(const vector<vector<int>>& ratings, int userId) {
int numUsers = ratings.size();
int numMovies = ratings[0].size();

vector<double> similarity(numUsers, 0.0);
vector<double> predictedRatings(numMovies, 0.0);
vector<double> similaritySum(numMovies, 0.0);

// Calculate similarity with other users
for (int i = 0; i < numUsers; ++i) {
if (i != userId) {
similarity[i] = cosineSimilarity(ratings[userId], ratings[i]);
}
}

// Predict ratings for unrated movies
for (int i = 0; i < numUsers; ++i) {
if (i != userId) {
for (int j = 0; j < numMovies; ++j) {
if (ratings[userId][j] == 0 && ratings[i][j] > 0) {
predictedRatings[j] += similarity[i] * ratings[i][j];
similaritySum[j] += similarity[i];
}
}
}
}

for (int j = 0; j < numMovies; ++j) {
if (similaritySum[j] > 0) {
predictedRatings[j] /= similaritySum[j];
}
}

return predictedRatings;
}

// Function to recommend a ranked list of top N movies
vector<pair<int, double>> recommendMovies(const vector<double>& predictedRatings, int topN) {
vector<pair<int, double>> movieScores;

for (size_t i = 0; i < predictedRatings.size(); ++i) {
if (predictedRatings[i] > 0) {
movieScores.push_back({(int)i, predictedRatings[i]});
}
}

// Sort movies by their predicted ratings in descending order
sort(movieScores.begin(), movieScores.end(), [](const pair<int, double>& a, const pair<int, double>& b) {
return a.second > b.second;
});

// Return the top N recommendations
if (movieScores.size() > topN) {
movieScores.resize(topN);
}

return movieScores;
}

// Function to calculate RMSE
double calculateRMSE(const vector<vector<int>>& ratings, const vector<vector<int>>& testRatings) {
int count = 0;
double mse = 0.0;

for (size_t i = 0; i < ratings.size(); ++i) {
for (size_t j = 0; j < ratings[i].size(); ++j) {
if (testRatings[i][j] > 0) {
double predicted = ratings[i][j] > 0 ? ratings[i][j] : 0;
mse += pow(predicted - testRatings[i][j], 2);
count++;
}
}
}
```

```
return count > 0 ? sqrt(mse / count) : 0.0;
}

// Function to calculate MAE (Mean Absolute Error)
double calculateMAE(const vector<vector<int>>& ratings, const vector<vector<int>>& testRatings) {
int count = 0;
double mae = 0.0;

for (size_t i = 0; i < ratings.size(); ++i) {
for (size_t j = 0; j < ratings[i].size(); ++j) {
if (testRatings[i][j] > 0) {
double predicted = ratings[i][j] > 0 ? ratings[i][j] : 0;
mae += fabs(predicted - testRatings[i][j]);
count++;
}
}
}

return count > 0 ? mae / count : 0.0;
}

// Function to calculate Precision@N
double calculatePrecision(const vector<vector<int>>& testRatings, const vector<pair<int, double>>& recommendations,
int topN) {
int relevantCount = 0;
int recommendedCount = 0;

for (int i = 0; i < topN && i < recommendations.size(); ++i) {
int movieId = recommendations[i].first;
if (testRatings[0][movieId] > 0) { // Assuming the first user is the test user
relevantCount++;
}
recommendedCount++;
}

return recommendedCount > 0 ? (double)relevantCount / recommendedCount : 0.0;
}

// Function to calculate Recall@N
double calculateRecall(const vector<vector<int>>& testRatings, const vector<pair<int, double>>& recommendations, int
topN) {
int relevantCount = 0;
int relevantMoviesCount = 0;

// Count relevant movies in the test set
for (int i = 0; i < testRatings[0].size(); ++i) {
if (testRatings[0][i] > 0) { // Assuming the first user is the test user
relevantMoviesCount++;
}
}
```

```cpp
// Count how many of the relevant movies are in the top-N recommendations
for (int i = 0; i < topN && i < recommendations.size(); ++i) {
int movieId = recommendations[i].first;
if (testRatings[0][movieId] > 0) {
relevantCount++;
}
}

return relevantMoviesCount > 0 ? (double)relevantCount / relevantMoviesCount : 0.0;
}

int main() {
// Ratings Matrix (rows: users, columns: movies)
vector<vector<int>> ratings = {
{5, 4, 5, 0, 3}, // User 0
{4, 0, 0, 5, 1}, // User 1
{1, 1, 0, 0, 5}, // User 2
{0, 0, 5, 4, 0}, // User 3
{3, 4, 0, 0, 0} // User 4
};

// Example Test Data (for RMSE calculation)
vector<vector<int>> testRatings = {
{5, 4, 0, 0, 3}, // User 0
{4, 0, 0, 5, 1}, // User 1
{1, 1, 0, 3, 5}, // User 2
{2, 0, 0, 4, 3}, // User 3
{3, 4, 2, 0, 0} // User 4
};

int userId = 4; // User for whom we want recommendations,the specific user
int topN = 3; // Number of topN recommendations to fetch

vector<double> predictedRatings = predictRatings(ratings, userId);

cout << "Predicted ratings for User " << userId << ":\n";
for (size_t i = 0; i < predictedRatings.size(); ++i) {
cout << "Movie " << i << ": " << fixed << setprecision(2) << predictedRatings[i] << endl;
}

vector<pair<int, double>> recommendations = recommendMovies(predictedRatings, topN);

cout << "\nTop " << topN << " recommended movies for User " << userId << ":\n";
for (const auto& recommendation : recommendations) {
cout << "Movie " << recommendation.first << " with predicted rating " << fixed << setprecision(2) <<
recommendation.second << endl;
}

double mae = calculateMAE(ratings, testRatings);
cout << "\nMean Absolute Error (MAE): " << fixed << setprecision(4) << mae << endl;
```

```cpp
double precision = calculatePrecision(testRatings, recommendations, topN);
cout << "Precision@3: " << fixed << setprecision(4) << precision << endl;

double recall = calculateRecall(testRatings, recommendations, topN);
cout << "Recall@3: " << fixed << setprecision(4) << recall << endl;

double rmse = calculateRMSE(ratings, testRatings);
cout << "\nRoot Mean Square Error (RMSE): " << fixed << setprecision(4) << rmse << endl;

return 0;
}
```

Output:

```
Predicted ratings for User 4:

Movie 0: 0.00

Movie 1: 0.00

Movie 2: 5.00

Movie 3: 5.00

Movie 4: 2.85


Top 3 recommended movies for User 4:

Movie 2 with predicted rating 5.00

Movie 3 with predicted rating 5.00

Movie 4 with predicted rating 2.85


Mean Absolute Error (MAE): 0.6250

Precision@3: 0.3333

Recall@3: 0.3333

Root Mean Square Error (RMSE): 1.2748
```