Міністерство освіти і науки України
Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення
Дисципліна: Алгоритми та структури даних

Лабораторна робота №5

Тема: «КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ ДЛЯ РЕАЛІЗАЦІЇ РЕКОМЕНДАЦІЙНИХ СИСТЕМ»

Виконав: ст. гр. КН-24
Куріщенко П. В.
Перевірив: викладач
Константинова Л. В.

Кропивницький 2026

Мета: Навчитися реалізовувати алгоритми колаборативної фільтрації.

Завдання: Реалізувати алгоритми колаборативної фільтрації на основі подоби користувачів та на основі подоби предметів. Як вхідні дані для алгоритмів використати відкритий набір даних MovieLens Datasets.

Повний лістинг програми міститься в git репозиторії: https://github.com/movavok/Algorithms-labs/tree/main/lab_5

Лістинг *dataset.h*:

```
#ifndef DATASET_H
#define DATASET_H

#include <iostream>
#include <unordered_map>
#include <string>
#include <sstream>
#include <fstream>

using namespace std;

class Dataset {
private:
    unordered_map<size_t, unordered_map<size_t, double>> m_userRatings;
    unordered_map<size_t, unordered_map<size_t, double>> m_movieRatings;

public:
    void load(const string& filename);

    const unordered_map<size_t, unordered_map<size_t, double>>&
getUserRatings() const {
        return m_userRatings;
    }

    const unordered_map<size_t, unordered_map<size_t, double>>&
getMovieRatings() const {
        return m_movieRatings;
    }
};

#endif // DATASET_H
```

Лістинг *dataset.cpp*:

```
#include "dataset.h"

void Dataset::load(const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
```

```
        cerr << "Не вдалося відкрити файл: " << filename << endl;
        return;
    }

    string line;
    getline(file, line);

    while (getline(file, line)) {
        stringstream steam(line);
        string user, movie, rating, timestamp;

        getline(steam, user, ',');
        getline(steam, movie, ',');
        getline(steam, rating, ',');
        getline(steam, timestamp, ',');

        size_t userId = stoul(user);
        size_t movieId = stoul(movie);
        double ratingValue = stod(rating);

        m_userRatings[userId][movieId] = ratingValue;
        m_movieRatings[movieId][userId] = ratingValue;
    }

    file.close();
    cout << "Завантажено " << m_userRatings.size() << " користувачів та " <<
m_movieRatings.size() << " фільмів.\n";
}
```

Лістинг *similarity.h*:

```
#ifndef SIMILARITY_H
#define SIMILARITY_H

#include "dataset.h"
#include <vector>

class Similarity {
public:
    static double pearsonUsers(size_t firstUserId, size_t secondUserId, const
Dataset&);
    static double pearsonMovies(size_t firstMovieId, size_t secondMovieId,
const Dataset&);

private:
    static double computePearson(const vector<pair<double, double>>&
commonRatings);
};


#endif // SIMILARITY_H
```

Лістинг *similarity.cpp*:

```
#include "similarity.h"
```

```cpp
double Similarity::computePearson(const vector<pair<double, double>>&
commonRatings) {
    int n = commonRatings.size();
    if (n == 0) return 0.0;

    double sumX = 0.0;
    double sumY = 0.0;
    double sumXSq = 0.0;
    double sumYSq = 0.0;
    double sumXY = 0.0;

    for (const auto& [x, y] : commonRatings) {
        sumX += x;
        sumY += y;
        sumXSq += x * x;
        sumYSq += y * y;
        sumXY += x * y;
    }

    double numer = sumXY - (sumX * sumY / n);

    double denom = sqrt((sumXSq - (sumX * sumX / n)) *
                        (sumYSq - (sumY * sumY / n)));

    if (denom == 0.0) return 0.0;

    return numer / denom;
}

double Similarity::pearsonUsers(size_t firstUserId, size_t secondUserId,
const Dataset& dataset) {
    const unordered_map<size_t, unordered_map<size_t, double>>& allUsers =
dataset.getUserRatings();

    if (!allUsers.count(firstUserId) || !allUsers.count(secondUserId))
        return 0.0;

    const unordered_map<size_t, double>& firstRatings =
allUsers.at(firstUserId);
    const unordered_map<size_t, double>& secondRatings =
allUsers.at(secondUserId);

    vector<pair<double, double>> commonRatings;

    for (const auto& [movieId, rating] : firstRatings)
        if (secondRatings.count(movieId))
            commonRatings.emplace_back(rating, secondRatings.at(movieId));

    return computePearson(commonRatings);
}

double Similarity::pearsonMovies(size_t firstMovieId, size_t secondMovieId,
const Dataset& dataset) {
    const unordered_map<size_t, unordered_map<size_t, double>>& allUsers =
dataset.getUserRatings();

    vector<pair<double, double>> commonRatings;

    for (const auto& [userId, ratings] : allUsers)
```

```
        if (ratings.count(firstMovieId) && ratings.count(secondMovieId))
            commonRatings.emplace_back(ratings.at(firstMovieId),
ratings.at(secondMovieId));

    return computePearson(commonRatings);
}
```

Лістинг *recommender.h*:

```
#ifndef RECOMMENDER_H
#define RECOMMENDER_H

#include "similarity.h"

class Recommender {
public:
    Recommender(Dataset& data) : m_data(data) {}

    vector<pair<size_t, double>> recommendUserBased(size_t userId, size_t
amount) {
        return recommend(userId, amount, Similarity::pearsonUsers);
    }

    vector<pair<size_t, double>> recommendItemBased(size_t userId, size_t
amount) {
        return recommend(userId, amount, Similarity::pearsonMovies);
    }

private:
    Dataset& m_data;

    static bool compareDesc(const pair<size_t, double>& a, const pair<size_t,
double>& b) {
        return a.second > b.second;
    }

    template<typename SimilarityFunc>
    vector<pair<size_t, double>> recommend(size_t userId, size_t
recommendedAmount, SimilarityFunc similarity) {
        const unordered_map<size_t, unordered_map<size_t, double>>& allUsers
= m_data.getUserRatings();
        if (!allUsers.count(userId)) return {};

        const unordered_map<size_t, double>& userRated = allUsers.at(userId);
        unordered_map<size_t, double> scores;
        unordered_map<size_t, double> similarSums;

        for (const auto& [candidateId, _] : allUsers) {
            if (candidateId == userId) continue;

            double sim = similarity(userId, candidateId, m_data);
            if (sim <= 0.0) continue;

            for (const auto& [movieId, rating] : userRated) {
                scores[movieId] += rating * sim;
                similarSums[movieId] += sim;
            }
        }
```

```
        vector<pair<size_t, double>> recommendations;
        for (const auto& [id, total] : scores)
            recommendations.emplace_back(id, total / similarSums[id]);

        sort(recommendations.begin(), recommendations.end(), compareDesc);

        if (recommendations.size() > recommendedAmount)
            recommendations.resize(recommendedAmount);

        return recommendations;
    }
};

#endif // RECOMMENDER_H
```

Лістинг *main.cpp*:

```
#include <iostream>

#include "recommender.h"

using namespace std;

int main() {
    Dataset dataset;
    dataset.load("../../data/ratings.csv");

    Recommender rec(dataset);

    size_t userId = 1;
    size_t recommendedAmount = 10;

    cout << "Введіть номер користувача: ";
    cin >> userId;
    cout << "Введіть кількість рекомендацій: ";
    cin >> recommendedAmount;

    vector<pair<size_t, double>> userRecs = rec.recommendUserBased(userId,
recommendedAmount);
    cout << "Рекомендації на основі схожих користувачів для користувача №" <<
userId << ":" << endl;
    for (const auto& [movieId, score] : userRecs)
        cout << "Фільм №" << movieId << " -> передбачувана оцінка: " << score
<< endl;

    cout << endl;

    vector<pair<size_t, double>> itemRecs = rec.recommendItemBased(userId,
recommendedAmount);
    cout << "Рекомендації на основі схожих фільмів для користувача №" <<
userId << ":" << endl;
    for (const auto& [movieId, score] : itemRecs)
        cout << "Фільм №" << movieId << " -> передбачувана оцінка: " << score
<< endl;

    return 0;
}
```

Результат виконання:

```
Завантажено 610 користувачів та 9724 фільмів.
Введіть номер користувача: 101
Введіть кількість рекомендацій: 10
Рекомендації на основі схожих користувачів для користувача №101:
Фільм №2997 -> передбачувана оцінка: 5
Фільм №2712 -> передбачувана оцінка: 5
Фільм №3174 -> передбачувана оцінка: 5
Фільм №2959 -> передбачувана оцінка: 5
Фільм №1093 -> передбачувана оцінка: 5
Фільм №1719 -> передбачувана оцінка: 5
Фільм №2395 -> передбачувана оцінка: 5
Фільм №2599 -> передбачувана оцінка: 5
Фільм №2318 -> передбачувана оцінка: 5
Фільм №2858 -> передбачувана оцінка: 4

Рекомендації на основі схожих фільмів для користувача №101:
Фільм №2997 -> передбачувана оцінка: 5
Фільм №2712 -> передбачувана оцінка: 5
Фільм №3174 -> передбачувана оцінка: 5
Фільм №2959 -> передбачувана оцінка: 5
Фільм №1093 -> передбачувана оцінка: 5
Фільм №1719 -> передбачувана оцінка: 5
Фільм №2395 -> передбачувана оцінка: 5
Фільм №2599 -> передбачувана оцінка: 5
Фільм №2318 -> передбачувана оцінка: 5
Фільм №2858 -> передбачувана оцінка: 4
```

Висновки: У ході роботи було реалізовано алгоритми колаборативної фільтрації для побудови системи рекомендацій фільмів на основі набору даних MovieLens. Зокрема, було розроблено два підходи: user-based (на основі схожих користувачів) та item-based (на основі схожих фільмів), із використанням коефіцієнта кореляції Пірсона для обчислення схожості.