2003 Project Assignment

Supervisor: Doç.Dr.Teyfik Aytekin

Course Code & Title:CMP2003

Authors: Arda Baş 2102258

Deniz Yiğit Derviş 2102422

Ali Kaan Karagözgil 2102997

1.Introduction

Movie recommendation is a common task in the field of computer science and machine learning. One popular method for recommending movies is matrix factorization, which is a technique used to decompose a matrix into a product of two lower-rank matrices. This technique has been used successfully in many applications, including collaborative filtering for recommendation systems.

In this report, we will discuss how to implement a movie recommendation system using matrix factorization in C++. We will begin by discussing the basics of matrix factorization and how it can be used for recommendation systems. Next, we will go through the steps of implementing a movie recommendation system using matrix factorization in C++, including the necessary data structures and algorithms. Finally, we will conclude with a discussion of the advantages and limitations of using matrix factorization for movie recommendation.

Matrix factorization is a technique that decomposes a matrix A into the product of two lower-rank matrices, U and V, such that $A \approx UV$. The matrices U and V are called the "factor matrices." The rank of a matrix is the number of independent rows or columns in the matrix. For example, a matrix with rank 2 has two independent rows or columns, and a matrix with rank 3 has three independent rows or columns. Matrix factorization can be used for recommendation systems by representing users and items in a matrix, with the rows representing users and the columns representing items. The entries in the matrix are the ratings that users have given to the items. For example, if a user has given a rating of 5 to a movie, the entry in the matrix at the row corresponding to the user and the column corresponding to the movie would be 5.To implement a movie recommendation system using matrix factorization in C++, we will need to perform the following steps:

-Collect and preprocess the data: We will need a dataset of users, movies, and ratings. This data can be collected from online movie rating websites or from a survey. Before we can use the data for matrix factorization, we will need to preprocess it by removing any missing or incomplete entries and normalizing the ratings to a common scale.

-Initialize the factor matrices U and V: We will need to initialize the factor matrices U and V with random values. The size of the matrices will depend on the number of users and movies in the dataset.

-Iteratively update the factor matrices: We will then iteratively update the factor matrices U and V using an optimization algorithm such as gradient descent. The goal of the optimization is to find the values of U and V that minimize the difference between the predicted ratings and the actual ratings in the dataset.

-Make recommendations: Once we have obtained the factor matrices U and V, we can use them to make recommendations for movies to users. For a given user, we can find the movies that they have not yet rated and predict their ratings for those movies using the factor matrices. We can then recommend the movies with the highest predicted ratings to the user.

There are several advantages to using matrix factorization for movie recommendation. One advantage is that it can handle large datasets with many users and items efficiently. Additionally, matrix factorization can handle missing ratings and make recommendations for items that a user has not yet rated.

However, there are also some limitations to using matrix factorization for movie recommendation. One limitation is that it does not take into account the context or content of the movies, only the ratings. Additionally, matrix factorization can be sensitive to noise in the data and may produce poor recommendations if the factor matrices are not initialized or optimized correctly. In conclusion, matrix factorization is a useful technique for implementing a movie recommendation system in C++. It can handle large datasets with many users and items efficiently and make recommendations for items that a user has not yet rated. However, it is important to carefully preprocess the data and initialize and optimize the factor matrices in order to produce accurate recommendations. Overall, movie recommendation systems using matrix factorization can be a powerful tool for helping users discover new movies that they may enjoy. By implementing a recommendation system in C++ using matrix factorization, we can provide personalized recommendations to users based on their past ratings and the ratings of other users with similar tastes.

2.Our Own Approach

```cpp
#include <iostream>
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>
#include <map>
#include <queue>
#include <fstream>
#include <map>
#include <string>
#include <vector>
#include <sstream>
#include <iomanip>

using namespace std;
class MF_ALS {
public:
    int n_users_;
    int n_items_;
    int n_factors_;
    double lr_;
    double reg_;
    vector<vector<double>> user_factors_;
    vector<vector<double>> item_factors_;

    //constructor
    MF_ALS(int n_users, int n_items, int n_factors, double lr, double reg)
    :n_users_(n_users),
    n_items_(n_items),
    n_factors_(n_factors),
    lr_(lr),
    reg_(reg){

        for (int i = 0; i < n_users_; i++) {
                    vector<double> user_factors;
                    for (int j = 0; j < n_factors_; j++) {
                        user_factors.push_back(rand() / (RAND_MAX + 1.0));

                    }
            user_factors_.push_back(user_factors);
                }
                for (int i = 0; i < n_items_; i++) {
                    vector<double> item_factors;
                    for (int j = 0; j < n_factors_; j++) {
```

```cpp
void read_input(MF_ALS& model, const std::string& filename) {
        usersSum[user] = usersSum[user]+rating;
        itemSum[item] = itemSum[item] + rating;
    }

        file.close();
            // Train model on each rating
        //double total_squared_error = 0.0;
    double total_absolute_error = 0.0;
        int num_ratings = 0;
    for (const auto& [user, user_ratings] : ratings) {
        for (const auto& [item, rating] : user_ratings) {
            model.train(user, item, rating);
            double error = abs(rating - model.predict(user, item));
            total_absolute_error += error;
            num_ratings++;
            /*double prediction = model.predict(user, item);
            double squared_error = (prediction - rating) * (prediction - rating);
            total_squared_error += squared_error;
            num_ratings++;*/
        }
    }
        double MAE = total_absolute_error / num_ratings;
        //double mse = total_absolute_error / num_ratings;
        double rmse = sqrt(MAE);
       // std::cout << "RMSE: " << rmse << std::endl;

}


int main() {
    // Initialize model
        MF_ALS model(100000, 10000, 15, 0.01, 0.01);

        // Read in ratings data from file and train model
    read_input(model, "train.csv");

std::ifstream test_file("test.csv");
    std::string test_line;
    while (std::getline(test_file, test_line)) {
        int id, user, item;
        std::istringstream iss(test_line);
        iss.ignore();
        iss >> id;
```

```cpp
class MF_ALS {
    double predict(int user, int item) {
            return 0.0;
        }
        double prediction = 0;
        for (int i = 0; i < n_factors_; i++) {
            prediction += user_factors_[user][i] * item_factors_[item][i];
        }
        return prediction;
    }

    void train(int user, int item, double rating) {
        if (user >= n_users_ || item >= n_items_) {
            return;
        }
        double prediction = predict(user, item);
        double error = rating - prediction;
        // Update user and item factors using gradient descent
        for (int i = 0; i < n_factors_; i++) {
            double user_factor = user_factors_[user][i];
            double item_factor = item_factors_[item][i];
            user_factors_[user][i] += lr_ * (error * item_factor - reg_ * user_factor);
            item_factors_[item][i] += lr_ * (error * user_factor - reg_ * item_factor);
        }
    }
};
int usersSum[1000000] = {0};
 int itemSum[100000] = {0};
void read_input(MF_ALS& model, const std::string& filename) {
    std::map<int, std::map<int, double>> ratings;
    std::ifstream file(filename);
    std::string line;
    while (std::getline(file, line)) {
    int user;
    int item;
    double rating;
    std::istringstream iss(line);
    iss.ignore();
    iss >> user;
    iss.ignore(); // ignore the ',' character
    iss >> item;
    iss.ignore(); // ignore the ',' character
    iss >> rating;
    ratings[user][item] = rating;
```

```cpp
int main() {
    while (std::getline(test_file, test_line)) {
        int id, user, item;
        std::istringstream iss(test_line);
        iss.ignore();
        iss >> id;
        iss.ignore();
        iss >> user;
        iss.ignore();
        iss >> item;
        double prediction = model.predict(user, item);
        std::cout << id << "," <<setprecision(50)<<prediction << std::endl;
    }
    test_file.close();

    std::cout << "Top 10 highest rated users are:" << std::endl;
        for (int i = 0; i < 10; i++){
            int max_val = 0, max_ind = 0;
            for (int j = 0; j < 1000000; j++){
                if (usersSum[j] > max_val){
                    max_val = usersSum[j];
                    max_ind = j;
                }
            }
            std::cout << "User " << max_ind << ": " << max_val << std::endl;
            usersSum[max_ind] = 0;
        }

      std::cout << std::endl;

      std::cout << "Top 10 highest rated movies are:" << std::endl;
        for (int i = 0; i < 10; i++){
            int max_val = 0, max_ind = 0;
            for (int j = 0; j < 100000; j++){
                if (itemSum[j] > max_val){
                    max_val = itemSum[j];
                    max_ind = j;
                }
            }
            std::cout << "Movie " << max_ind << ": " << max_val << std::endl;
            itemSum[max_ind] = 0;
        }

}
```

Matrix factorization is a technique used to predict ratings or preferences of users for items in a collaborative filtering system. Collaborative filtering systems are used to recommend items to users based on the preferences of other users. These systems can be used to recommend movies, songs, products, or any other type of item. The goal of matrix factorization is to decompose a matrix of user-item ratings into two lower rank matrices such that the product of these two matrices approximates the original matrix. These two matrices represent the preferences of the users and the characteristics of the items, respectively. The ALS algorithm is an iterative method for solving matrix factorization problems. It alternates between fixing the user matrix and solving for the item matrix, and vice versa. This is done until convergence, which occurs when the difference between the approximated and original matrices is below a certain threshold. The MF_ALS class in the provided code is an implementation of matrix factorization using the ALS algorithm. It has several member variables:

n_users_: the number of users in the dataset

n_items_: the number of items in the dataset

n_factors_: the number of factors to use in the matrix factorization

lr_: the learning rate used in the gradient descent optimization

reg_: the regularization term used in the gradient descent optimization

user_factors_: a matrix of factors representing the preferences of the users

item_factors_: a matrix of factors representing the characteristics of the items

The MF_ALS class has three member functions: a constructor, a predict function, and a train function. The constructor initializes the user_factors_ and item_factors_ matrices with random values. The predict function takes in a user and an item and returns the predicted rating for that user-item pair by taking the dot product of the corresponding rows in the user_factors_ and item_factors_ matrices. The train function takes in a user, item, and rating, and updates the user_factors_ and item_factors_ matrices using gradient descent to minimize the prediction error. The prediction error is calculated as the difference between the actual rating and the predicted rating. The read_input function reads in ratings data from a file and trains the model on each rating using the train function. It also calculates the mean absolute error (MAE) of the model's predictions on the ratings data. The MAE is a measure of the average magnitude of the prediction errors, and is calculated as the sum of the absolute errors divided by the number of ratings. A lower MAE indicates a better fit of the model to the data.

3.Summary

In summary, the provided code is an implementation of matrix factorization using the ALS algorithm for predicting movie ratings. The user-item matrix is decomposed into two lower rank matrices representing the preferences of the users and the characteristics of the movies. The model is trained on a set of user-item ratings and can be used to predict the ratings for movies that a user has not yet rated. The ALS algorithm is used to optimize the model parameters and improve the prediction accuracy.