Homework #03 Answer

데이터사이언스를 위한 컴퓨팅 1 (2022년도 2학기, M3239.005500)

Due: 2022년 11월 1일 (화) 23시 59분

1 Read file, calculate correlation and run simple linear regression [100pts]

- Error for base case : -50 pts
- Error for test case : -15 pts for each function
- Wrong result (without error) : -5 pts for each function
- Wrong file format : -10 pts

Below is a sample answer.

Matrix.h

```
#pragma once

#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <typeinfo>
#include <algorithm>
#include <cmath>
using namespace std;
class DataFrame;

class Matrix{
    public:
        void input (vector<vector<double> >&);
        Matrix operator + (const Matrix&);
        Matrix operator - (const Matrix&);
        Matrix operator * (const Matrix&);
        Matrix Transpose();
        Matrix GetSubVectorbyColumn(int);
        void Print();
        int GetNumRow();
        int GetNumColumn();
        double GetVal(int, int);
        friend class DataFrame;
        vector<vector<double> > arr;
        int rows;
        int cols;
};

Matrix Cor(Matrix&, int method = 1);
Matrix SimpleLinearRegression(Matrix&, Matrix&);

class DataFrame{
    public:
        int ReadData(string, char, char, bool);
        Matrix GetMatrix(int[], int);
        vector<vector<string> > parsedCsv;
};
```

Matrix.cpp

```
#include "Matrix.h"

// Fill in here
#include <iostream>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <typeinfo>
#include <algorithm>
#include <cmath>
using namespace std;


int DataFrame::ReadData(std::string FileName, char sep, char comment, bool IsHeader){
    string line;
    ifstream myfile (FileName);
    bool header = IsHeader;
    char sepr = sep;

    if (myfile.is_open()){
        while (getline (myfile, line)){
            if (line[0] == comment) continue;
            stringstream lineStream(line);
            string cell;
            vector<string> parsedRow;
            while (getline(lineStream, cell, sepr)){
                    parsedRow.push_back(cell);
                }
            parsedCsv.push_back(parsedRow);
        }
    myfile.close();
    }

    if (header){
        for (int i = 0; i < parsedCsv.size(); i++){
            if (i == 0){
                parsedCsv.erase(parsedCsv.begin() + i);
            }
        }
    }
    return 0;
}

Matrix DataFrame::GetMatrix(int index[], int nColumn){
    int col_num = nColumn;
    vector<vector<double> > M;
    vector<double> C;
    for (int a = 0; a < parsedCsv.size(); a++){
        for (int b = 0; b < col_num; b++){
            C.push_back(stod(parsedCsv[a][index[b]]));
        }
        M.push_back(C);
        C.clear();
    }
    Matrix mat;
    mat.input(M);
    return mat;
}

void Matrix::input(vector<vector<double> >& A){
    rows = A.size();
    cols = A[0].size();
    vector<double> C;
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            C.push_back(A[i][j]);
        }
        arr.push_back(C);
        C.clear();
    }
}

Matrix Matrix::operator + (const Matrix& X){
    Matrix mat;
    vector<vector<double> > C(rows, vector<double>(cols));
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            C[i][j] = 0;
            C[i][j] += arr[i][j] + X.arr[i][j];
        }
    }
    mat.input(C);
    return mat;
}

Matrix Matrix::operator * (const Matrix& X){
    Matrix mat;
    vector<vector<double> > C(rows, vector<double>(X.cols));
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < X.cols; j++){
            C[i][j] = 0;
            for (int k = 0; k < cols; k++){
                C[i][j] += arr[i][k] * (X.arr[k][j]);
            }
        }
    }
    mat.input(C);
    return mat;
}

Matrix Matrix::operator - (const Matrix& X){
    Matrix mat;
    vector<vector<double> > C(rows, vector<double>(cols));
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            C[i][j] = 0;
            C[i][j] -= X.arr[i][j] - arr[i][j];
        }
    }
    mat.input(C);
    return mat;
}

Matrix Matrix::Transpose(){
    Matrix mat;
    vector<vector<double> > C (cols, vector<double>(rows));
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            C[j][i] = arr[i][j];
        }
    }
    mat.input(C);
    return mat;
}

Matrix Matrix::GetSubVectorbyColumn(int column){
    Matrix mat;
    vector<vector<double> > C (rows, vector<double>(1));
    for (int i = 0; i < rows; i++){
        int j = column;
        C[i][0] = arr[i][j];
        }
        mat.input(C);
    return mat;
}

void Matrix::Print(){
    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            cout << arr[i][j] << ' ';
        }
        cout << endl;
    }
}

int Matrix::GetNumRow(){
    return rows;
}

int Matrix::GetNumColumn(){
    return cols;
}

double Matrix::GetVal(int x, int y){
    return arr[x][y];
}

Matrix Cor(Matrix &mat, int method){
    Matrix cor_mat;
    int row = mat.GetNumRow();
    int col = mat.GetNumColumn();
    vector<double> sum_x (col);
    vector<double> squaresum_x (col);
    vector<vector<double> > sum_xy (col, vector<double>(col));
    vector<vector<double> > corr (col, vector<double>(col));
    int pair = row * (row - 1) / 2;

    if (method == 1){
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++){
                sum_x[j] += mat.arr[i][j];
                squaresum_x[j] += mat.arr[i][j] * mat.arr[i][j];
                for (int k = 0; k < col; k++){
                    sum_xy[j][k] += mat.arr[i][j] * mat.arr[i][k];
                }
            }
        }
        for (int i = 0; i < col; i++){
            for (int j = 0; j < col; j++){
                corr[i][j] = (row * sum_xy[i][j] - sum_x[i] * sum_x[j]) /
                             sqrt((row * squaresum_x[i] - sum_x[i] * sum_x[i])
                             * (row * squaresum_x[j] - sum_x[j] * sum_x[j]));
            }
        }
    }

    if (method == 2){

        for (int i = 0; i < col; i++){
            for (int j = 0; j <= i; j++){
                for (int k = 0; k < row; k++){
                    for (int p = 0; p < k; p++){
                        corr[i][j] += (mat.arr[k][i] - mat.arr[p][i])
                        * (mat.arr[k][j] - mat.arr[p][j]) >= 0 ? 1 : -1;
                        if (i != j){
                        corr[j][i] += (mat.arr[k][i] - mat.arr[p][i])
                        * (mat.arr[k][j] - mat.arr[p][j]) >= 0 ? 1 : -1;
                        }
                    }
                }
            }
        }
        for (int i = 0; i < col; i++){
            for (int j = 0; j < col; j++){
                corr[i][j] = corr[i][j] / pair;
            }
        }
    }

    cor_mat.input(corr);
    return cor_mat;
}

Matrix SimpleLinearRegression(Matrix &X, Matrix &Y){
    Matrix linear_mat;
    int n = X.GetNumRow();
    double sum_x = 0;
    double sum_y = 0;
    double sum_xy = 0;
    double squaresum_x = 0;
    vector<vector<double> > coeff (2, vector<double>(1));

    for (int i=0; i < n; i++){
        sum_x += X.arr[i][0];
        sum_y += Y.arr[i][0];
        sum_xy += X.arr[i][0] * Y.arr[i][0];
        squaresum_x += X.arr[i][0] * X.arr[i][0];
    }

    coeff[1][0] = (n * sum_xy - sum_x * sum_y) / (n * squaresum_x - sum_x * sum_x);
    coeff[0][0] = (sum_y * squaresum_x - sum_x * sum_xy)
                  / (n * squaresum_x - sum_x * sum_x);
    linear_mat.input(coeff);
    return linear_mat;
}
```