Name: **Salman Irfan**

Roll no: **2021-MSCS-511**

Course: **Artificial Neural Networks**

Submitted To: **Dr. Muhammad Awais Hassan**

Assignment 3

Part 1: -

C++ program take input two parameter $\theta 0$ and $\theta 1$ of a hypothesis function and display total cost based on the given data.

Source Code: -

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
using namespace std;

// global variables
const double ALPHA = 0.01; // learning rate
int n;                     // number of data points
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

int globalNumRows = 0;           // dynamically gets number of rows from file
int globalNumCols = 0;           // dynamically gets number of cols from file
double x[MAX_ROWS], y[MAX_COLS]; // input data
double theta0, theta1;           // hypothesis parameters
string globalFileName;
bool isFileExist = false;

double globalInputArray[MAX_ROWS];
double globalOutputArray[MAX_ROWS];

// input file from user
string chooseFile()
{
    string fileName;
    cout << "File to Train = ";
    cin >> fileName;
    return fileName;
}

// function to read csv file
bool readCsvFile(double data[MAX_ROWS][MAX_COLS], string fileName)
{
    int numRows = 0;
    int numCols = 0;
```

```
    ifstream file(fileName);
    if (file.is_open())
    {
        cout << "\033[32m" << fileName << " is opened successfully"
             << "\033[0m" << endl;
        string line; // storing each line of the file
        int row = 0; // to keep track the current row, by default representing
first row

        // getline (input stream, line by line, delimiter)
        while (getline(file, line) && row < MAX_ROWS)
        {
            stringstream ss(line); // Use stringstream to split each line into
fields
            string field;          // store each field / csv of the line
            int col = 0;           // to keep track the current col, by default
representing first col

            while (getline(ss, field, ',') && col < MAX_COLS)
            {                                  // read fields upto maximum number
of columns
                data[row][col] = stod(field); // storing the data in the
respective column
                col++;
            }
            row++;
            numRows++;
            if (col > numCols)
            {
                numCols = col;
                globalNumCols = numCols;
            }
        }
        file.close();
        isFileExist = true;
    }
    else
    { // if the file was not opened successfully
        cout << "\033[31m"
             << "Failed to open file: " << fileName << "\033[0m" << endl;
    }
    globalNumRows = numRows;
    return isFileExist;
}
```

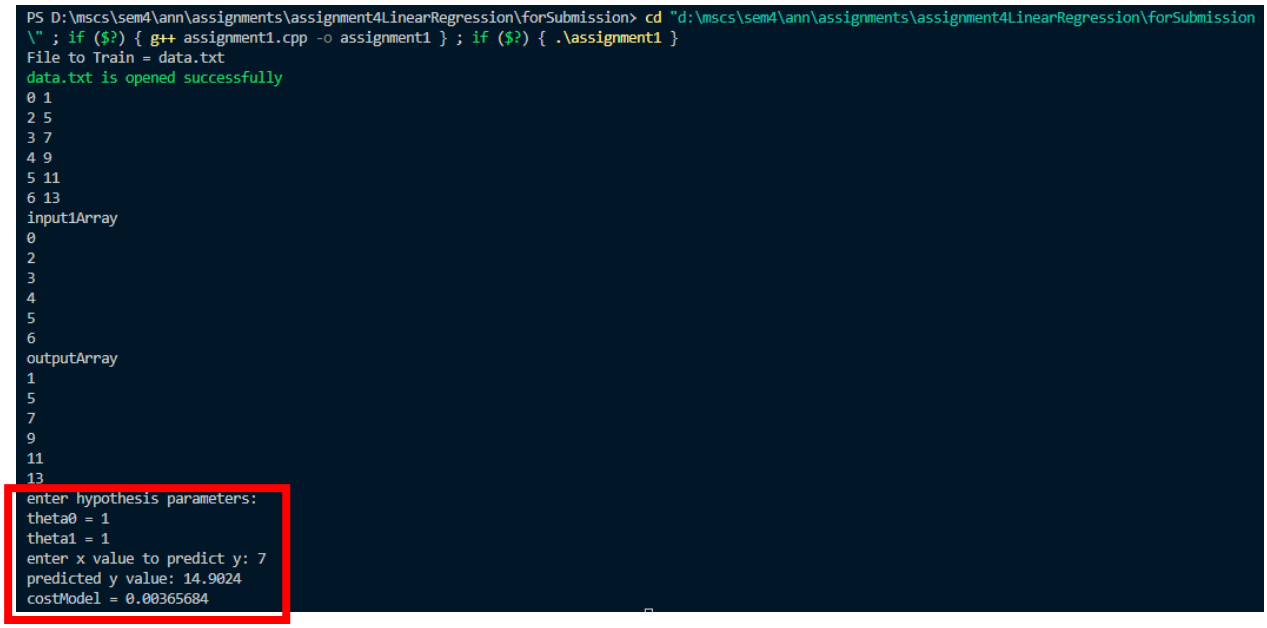
```
void printArray(double data[MAX_ROWS][MAX_COLS])
{
    for (int i = 0; i < globalNumRows; i++)
    { // for each row
        for (int j = 0; j < globalNumCols; j++)
        { // for each column
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
}

// computes the predicted value of y for a given value of x
double predict(double xValue)
{
    double yPred = theta0 + theta1 * xValue; // hypothesis function
    return yPred;
}

// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost()
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]); // hypothesis function
        double diff = h - globalOutputArray[i];  // difference between hypothesis
and actual value
        cost += diff * diff;                     // squared difference
    }
    cost /= (2 * n); // average squared difference
    return cost;
}

// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost(double t0, double t1)
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = t0 + t1 * globalInputArray[i]; // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        cost += diff * diff;                      // squared difference
    }
```

```
    cost /= (2 * globalNumRows); // average squared difference
    return cost;
}

// updates the hypothesis parameters using gradient descent
void updateParameters()
{
    double theta0Sum = 0, theta1Sum = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]);  // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        theta0Sum += diff;                        // partial derivative of cost
with respect to theta0
        theta1Sum += diff * globalInputArray[i]; // partial derivative of cost
with respect to theta1
    }
    theta0 -= ALPHA * theta0Sum / globalNumRows; // update theta0
    theta1 -= ALPHA * theta1Sum / globalNumRows; // update theta1
}

// trains the hypothesis function using gradient descent
void train()
{
    for (int i = 0; i < MAX_ROWS; i++)
    {
        double prev_cost = computeCost(); // compute previous cost
        updateParameters();               // update hypothesis parameters
        double curr_cost = computeCost(); // compute current cost
        if (prev_cost - curr_cost < 1e-6)
        { // check for convergence
            break;
        }
    }
}

int main()
{
    string fileName = chooseFile();
    globalFileName = fileName;

    // storing data from file
    double data[MAX_ROWS][MAX_COLS];
    readCsvFile(data, fileName); // calling function for reading file
```

```
    printArray(data);

    if (isFileExist)
    {
        double inputArray[MAX_ROWS];
        cout << "input1Array \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            inputArray[i] = data[i][0];
            globalInputArray[i] = inputArray[i];
            cout << inputArray[i] << "\n";
        }

        double outputArray[MAX_ROWS];
        cout << "outputArray \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            outputArray[i] = data[i][1];
            globalOutputArray[i] = outputArray[i];
            cout << outputArray[i] << "\n";
        }
    }

    // read hypothesis parameters
    cout << "enter hypothesis parameters: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;


    // train the hypothesis function
    train();

    // test the hypothesis function
    double xValue;
    cout << "enter x value to predict y: ";
    cin >> xValue;
    double yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // compute cost
    double costModel = computeCost(theta0, theta1);
    cout<<"costModel = "<< costModel << "\n";
```

```
    return 0;
}
```

Output: -



Part 2: -

Extend the previous assignment 01, now it should take parameters of two models and return the parameters of best model.

Source Code: -

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
using namespace std;

// global variables
const double ALPHA = 0.01; // learning rate
int n;                     // number of data points
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

int globalNumRows = 0;           // dynamically gets number of rows from file
int globalNumCols = 0;           // dynamically gets number of cols from file
double x[MAX_ROWS], y[MAX_COLS]; // input data
double theta0, theta1;           // hypothesis parameters


string globalFileName;
bool isFileExist = false;

double globalInputArray[MAX_ROWS];
double globalOutputArray[MAX_ROWS];

// input file from user
string chooseFile()
{
    string fileName;
    cout << "File to Train = ";
    cin >> fileName;
    return fileName;
}

// function to read csv file
bool readCsvFile(double data[MAX_ROWS][MAX_COLS], string fileName)
{
    int numRows = 0;
```

```
    int numCols = 0;
    ifstream file(fileName);
    if (file.is_open())
    {
        cout << "\033[32m" << fileName << " is opened successfully"
             << "\033[0m" << endl;
        string line; // storing each line of the file
        int row = 0; // to keep track the current row, by default representing
first row

        // getline (input stream, line by line, delimiter)
        while (getline(file, line) && row < MAX_ROWS)
        {
            stringstream ss(line); // Use stringstream to split each line into
fields
            string field;          // store each field / csv of the line
            int col = 0;           // to keep track the current col, by default
representing first col

            while (getline(ss, field, ',') && col < MAX_COLS)
            {                                  // read fields upto maximum number
of columns
                data[row][col] = stod(field); // storing the data in the
respective column
                col++;
            }
            row++;
            numRows++;
            if (col > numCols)
            {
                numCols = col;
                globalNumCols = numCols;
            }
        }
        file.close();
        isFileExist = true;
    }
    else
    { // if the file was not opened successfully
        cout << "\033[31m"
             << "Failed to open file: " << fileName << "\033[0m" << endl;
    }
    globalNumRows = numRows;
    return isFileExist;
}
```

```
void printArray(double data[MAX_ROWS][MAX_COLS])
{
    for (int i = 0; i < globalNumRows; i++)
    { // for each row
        for (int j = 0; j < globalNumCols; j++)
        { // for each column
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
}

// computes the predicted value of y for a given value of x
double predict(double xValue)
{
    double yPred = theta0 + theta1 * xValue; // hypothesis function
    return yPred;
}
// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost(double t0, double t1)
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = t0 + t1 * globalInputArray[i]; // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        cost += diff * diff;                      // squared difference
    }
    cost /= (2 * globalNumRows); // average squared difference
    return cost;
}

// updates the hypothesis parameters using gradient descent
void updateParameters()
{
    double theta0Sum = 0, theta1Sum = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]);  // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        theta0Sum += diff;                       // partial derivative of cost
with respect to theta0
```

```
        theta1Sum += diff * globalInputArray[i]; // partial derivative of cost
with respect to theta1
    }
    theta0 -= ALPHA * theta0Sum / globalNumRows; // update theta0
    theta1 -= ALPHA * theta1Sum / globalNumRows; // update theta1
}

// trains the hypothesis function using gradient descent
void train()
{
    for (int i = 0; i < MAX_ROWS; i++)
    {
        double prevCost = computeCost(theta0, theta1);
        updateParameters();
        double currCost = computeCost(theta0, theta1);
        if (prevCost - currCost < 1e-6)
        {
            break;
        }
    }
}


int main()
{
    string fileName = chooseFile();
    globalFileName = fileName;

    // storing data from file
    double data[MAX_ROWS][MAX_COLS];
    readCsvFile(data, fileName); // calling function for reading file
    printArray(data);

    if (isFileExist)
    {
        double inputArray[MAX_ROWS];
        cout << "input1Array \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            inputArray[i] = data[i][0];
            globalInputArray[i] = inputArray[i];
            cout << inputArray[i] << "\n";
        }

        double outputArray[MAX_ROWS];
```

```
        cout << "outputArray \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            outputArray[i] = data[i][1];
            globalOutputArray[i] = outputArray[i];
            cout << outputArray[i] << "\n";
        }
    }

    // read hypothesis parameters
    cout << "enter hypothesis parameters: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;

    double theta0ModelOne = theta0;
    double theta1ModelOne = theta1;

    // train the hypothesis function
    train();

    double xValue;
    cout << "enter x value to predict y: ";
    cin >> xValue;
    double yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // compute cost
    double costModel1 = computeCost(theta0, theta1);
    cout<<"costModel1 = "<< costModel1 << "\n";

    //  train another model
    cout << "enter hypothesis parameters for model two: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;
    double theta0ModelTwo = theta0;
    double theta1ModelTwo = theta1;

    // train the hypothesis function
    train();
```

```
    yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;



    // compute cost
    double costModel2 = computeCost(theta0, theta1);
    cout<<"costModel2 = "<< costModel2 << "\n";

    // deciding best model

    if (costModel1 < costModel2){
        cout << "the best model is model 1 with \ntheta0 = "<< theta0ModelOne <<
" and \ntheta1 = " << theta1ModelOne << "\n";
    }
    else{
        cout << "the best model is model 1 with \ntheta0 = "<< theta0ModelTwo <<
" and \ntheta1 = " << theta1ModelTwo << "\n";
    }

    return 0;
}
```

Output: -

```
\" ; if ($?) { g++ assignment2bestModel.cpp -o assignment2bestModel } ; if ($?) { .\assignment2bestModel }
File to Train = data.txt
data.txt is opened successfully
0 1
2 5
3 7
4 9
5 11
6 13
input1Array
0
2
3
4
5
6
outputArray
1
5
7
9
11
13
enter hypothesis parameters:
theta0 = 1
theta1 = 1
enter x value to predict y: 7
predicted y value: 14.9024
costModel1 = 0.00365684
enter hypothesis parameters for model two:
theta0 = 2
theta1 = 2
predicted y value: 14.5678
costModel2 = 0.0716336
the best model is model 1 with
theta0 = 1 and
theta1 = 1
```

Part 3: -

Extend the previous assignments, write a program that read the data and proposed the linear regression parameters.

(You can add range constrain on parameters).

Hint you need to change the first parameter value and keep the second parameter constant.

Source Code: -

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
using namespace std;

// global variables
const double ALPHA = 0.01; // learning rate
int n;                     // number of data points
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

int globalNumRows = 0;           // dynamically gets number of rows from file
int globalNumCols = 0;           // dynamically gets number of cols from file
double x[MAX_ROWS], y[MAX_COLS]; // input data
double theta0, theta1;           // hypothesis parameters


string globalFileName;
bool isFileExist = false;

double globalInputArray[MAX_ROWS];
double globalOutputArray[MAX_ROWS];

// input file from user
string chooseFile()
{
    string fileName;
    cout << "File to Train = ";
    cin >> fileName;
    return fileName;
}

// function to read csv file
bool readCsvFile(double data[MAX_ROWS][MAX_COLS], string fileName)
{
```

```
    int numRows = 0;
    int numCols = 0;
    ifstream file(fileName);
    if (file.is_open())
    {
        cout << "\033[32m" << fileName << " is opened successfully"
             << "\033[0m" << endl;
        string line; // storing each line of the file
        int row = 0; // to keep track the current row, by default representing
first row

        // getline (input stream, line by line, delimiter)
        while (getline(file, line) && row < MAX_ROWS)
        {
            stringstream ss(line); // Use stringstream to split each line into
fields
            string field;          // store each field / csv of the line
            int col = 0;           // to keep track the current col, by default
representing first col

            while (getline(ss, field, ',') && col < MAX_COLS)
            {                                  // read fields upto maximum number
of columns
                data[row][col] = stod(field); // storing the data in the
respective column
                col++;
            }
            row++;
            numRows++;
            if (col > numCols)
            {
                numCols = col;
                globalNumCols = numCols;
            }
        }
        file.close();
        isFileExist = true;
    }
    else
    { // if the file was not opened successfully
        cout << "\033[31m"
             << "Failed to open file: " << fileName << "\033[0m" << endl;
    }
    globalNumRows = numRows;
    return isFileExist;
```

```
}

void printArray(double data[MAX_ROWS][MAX_COLS])
{
    for (int i = 0; i < globalNumRows; i++)
    { // for each row
        for (int j = 0; j < globalNumCols; j++)
        { // for each column
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
}

// computes the predicted value of y for a given value of x
double predict(double xValue)
{
    double yPred = theta0 + theta1 * xValue; // hypothesis function
    return yPred;
}
// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost(double t0, double t1)
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = t0 + t1 * globalInputArray[i]; // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        cost += diff * diff;                      // squared difference
    }
    cost /= (2 * globalNumRows); // average squared difference
    return cost;
}

// updates the hypothesis parameters using gradient descent
void updateParameters()
{
    double theta0Sum = 0, theta1Sum = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]);  // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
```

```
        theta0Sum += diff;                          // partial derivative of cost
with respect to theta0
        theta1Sum += diff * globalInputArray[i]; // partial derivative of cost
with respect to theta1
    }
    theta0 -= ALPHA * theta0Sum / globalNumRows; // update theta0
    theta1 -= ALPHA * theta1Sum / globalNumRows; // update theta1
}

// find the linear regression parameters with a range constraint of 1 to 10
void findLinearRegressionParameters() {
    double suggestedCost = INFINITY; // initialize to positive infinity
    double suggestedTheta0, suggestedTheta1;
    for (int i = 1; i <= 10; i++) { // range constraint of 1 to 10
        double theta0 = i; // first parameter value
        double theta1 = 0; // keep the second parameter constant at 0
        double cost = computeCost(theta0, theta1);
        if (cost < suggestedCost) {
            suggestedCost = cost;
            suggestedTheta0 = theta0;
            suggestedTheta1 = theta1;
        }
    }
    cout << "suggested model: theta0 = " << suggestedTheta0 << ", theta1 = " <<
suggestedTheta1 << endl;
}

// trains the hypothesis function using gradient descent
void train()
{
    for (int i = 0; i < MAX_ROWS; i++)
    {
        double prevCost = computeCost(theta0, theta1);
        updateParameters();
        double currCost = computeCost(theta0, theta1);
        if (prevCost - currCost < 1e-6)
        {
            break;
        }
    }
}


int main()
{
```

```
    string fileName = chooseFile();
    globalFileName = fileName;

    // storing data from file
    double data[MAX_ROWS][MAX_COLS];
    readCsvFile(data, fileName); // calling function for reading file
    printArray(data);

    if (isFileExist)
    {
        double inputArray[MAX_ROWS];
        cout << "input1Array \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            inputArray[i] = data[i][0];
            globalInputArray[i] = inputArray[i];
            cout << inputArray[i] << "\n";
        }

        double outputArray[MAX_ROWS];
        cout << "outputArray \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            outputArray[i] = data[i][1];
            globalOutputArray[i] = outputArray[i];
            cout << outputArray[i] << "\n";
        }
    }

    // read hypothesis parameters for model one
    cout << "enter hypothesis parameters: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;

    double theta0ModelOne = theta0;
    double theta1ModelOne = theta1;

    // train the hypothesis function
    train();

    double xValue;
    cout << "enter x value to predict y: ";
    cin >> xValue;
```

```
    double yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // compute cost
    double costModel1 = computeCost(theta0, theta1);
    cout<<"costModel1 = "<< costModel1 << "\n";

    // find the linear regression parameters
    findLinearRegressionParameters();

    //  train another model two
    cout << "enter hypothesis parameters for model two: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;
    double theta0ModelTwo = theta0;
    double theta1ModelTwo = theta1;

    // train the hypothesis function
    train();


    yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // find the linear regression parameters
    findLinearRegressionParameters();

    // compute cost
    double costModel2 = computeCost(theta0, theta1);
    cout<<"costModel2 = "<< costModel2 << "\n";

    // deciding best model

    if (costModel1 < costModel2){
        cout << "the best model is model 1 with \ntheta0 = "<< theta0ModelOne <<
" & \ntheta1 = " << theta1ModelOne << "\n";
    }
    else{
        cout << "the best model is model 2 with \ntheta0 = "<< theta0ModelTwo <<
" & \ntheta1 = " << theta1ModelTwo << "\n";
    }

    return 0;
```

```
}
```

Output: -

```
PS D:\mscs\sem4\ann\assignments\assignment4LinearRegression\f
\" ; if ($?) { g++ assignment3RegressionParameters.cpp -o ass
File to Train = data.txt
data.txt is opened successfully
0 1
2 5
3 7
4 9
5 11
6 13
input1Array
0
2
3
4
5
6
outputArray
1
5
7
9
11
13
```

```
enter hypothesis parameters:
theta0 = 1
theta1 = 1
enter x value to predict y: 7
predicted y value: 14.9024
costModel1 = 0.00365684
suggested model: theta0 = 8, theta1 = 0
enter hypothesis parameters for model two:
theta0 = 2
theta1 = 2
predicted y value: 14.5678
suggested model: theta0 = 8, theta1 = 0
costModel2 = 0.0716336
the best model is model 1 with
theta0 = 1 &
theta1 = 1
```

Part 4: -

Calculate the runtime of algorithm.

Source Code: -

```cpp
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <chrono> // for seconds counting
using namespace std;
using namespace std::chrono;

// global variables
const double ALPHA = 0.01; // learning rate
int n;                     // number of data points
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

int globalNumRows = 0;           // dynamically gets number of rows from file
int globalNumCols = 0;           // dynamically gets number of cols from file
double x[MAX_ROWS], y[MAX_COLS]; // input data
double theta0, theta1;           // hypothesis parameters

string globalFileName;
bool isFileExist = false;

double globalInputArray[MAX_ROWS];
double globalOutputArray[MAX_ROWS];

// input file from user
string chooseFile()
{
    string fileName;
    cout << "File to Train = ";
    cin >> fileName;
    return fileName;
}

// function to read csv file
bool readCsvFile(double data[MAX_ROWS][MAX_COLS], string fileName)
{
    int numRows = 0;
    int numCols = 0;
    ifstream file(fileName);
```

```
    if (file.is_open())
    {
        cout << "\033[32m" << fileName << " is opened successfully"
             << "\033[0m" << endl;
        string line; // storing each line of the file
        int row = 0; // to keep track the current row, by default representing
first row

        // getline (input stream, line by line, delimiter)
        while (getline(file, line) && row < MAX_ROWS)
        {
            stringstream ss(line); // Use stringstream to split each line into
fields
            string field;          // store each field / csv of the line
            int col = 0;           // to keep track the current col, by default
representing first col

            while (getline(ss, field, ',') && col < MAX_COLS)
            {                                  // read fields upto maximum number
of columns
                data[row][col] = stod(field); // storing the data in the
respective column
                col++;
            }
            row++;
            numRows++;
            if (col > numCols)
            {
                numCols = col;
                globalNumCols = numCols;
            }
        }
        file.close();
        isFileExist = true;
    }
    else
    { // if the file was not opened successfully
        cout << "\033[31m"
             << "Failed to open file: " << fileName << "\033[0m" << endl;
    }
    globalNumRows = numRows;
    return isFileExist;
}

void printArray(double data[MAX_ROWS][MAX_COLS])
```

```
{
    for (int i = 0; i < globalNumRows; i++)
    { // for each row
        for (int j = 0; j < globalNumCols; j++)
        { // for each column
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
}

// computes the predicted value of y for a given value of x
double predict(double xValue)
{
    double yPred = theta0 + theta1 * xValue; // hypothesis function
    return yPred;
}
// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost(double t0, double t1)
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = t0 + t1 * globalInputArray[i]; // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        cost += diff * diff;                      // squared difference
    }
    cost /= (2 * globalNumRows); // average squared difference
    return cost;
}

// updates the hypothesis parameters using gradient descent
void updateParameters()
{
    double theta0Sum = 0, theta1Sum = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]);  // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        theta0Sum += diff;                        // partial derivative of cost
with respect to theta0
        theta1Sum += diff * globalInputArray[i]; // partial derivative of cost
with respect to theta1
```

```
    }
    theta0 -= ALPHA * theta0Sum / globalNumRows; // update theta0
    theta1 -= ALPHA * theta1Sum / globalNumRows; // update theta1
}

// find the linear regression parameters with a range constraint of 1 to 10
void findLinearRegressionParameters()
{
    double suggestedCost = INFINITY; // initialize to positive infinity
    double suggestedTheta0, suggestedTheta1;
    int iterations = 0;
    auto old = steady_clock::now(); // start timer
    for (int i = 1; i <= 10; i++)
    {                       // range constraint of 1 to 10
        double theta0 = i; // first parameter value
        double theta1 = 0; // keep the second parameter constant at 0
        double cost = computeCost(theta0, theta1);
        iterations++;
        if (cost < suggestedCost)
        {
            suggestedCost = cost;
            suggestedTheta0 = theta0;
            suggestedTheta1 = theta1;
        }
    }
    auto dur = steady_clock::now() - old; // stop timer
    auto duration = duration_cast<seconds>(dur).count();
    // don't know the details about chrono header, auto keyword, steady_clock,
    // followed this link
    // https://www.youtube.com/watch?v=QYaQStudgnE
    // from 9 minutes

    cout << "\n";
    cout << "total iterations = " << iterations << ", duration = " << duration <<
" seconds" << endl;
    cout << "suggested parameters = theta0 = " << suggestedTheta0 << ", theta1 =
" << suggestedTheta1 << endl;
}

// trains the hypothesis function using gradient descent
void train()
{
    for (int i = 0; i < MAX_ROWS; i++)
    {
        double prevCost = computeCost(theta0, theta1);
```

```
        updateParameters();
        double currCost = computeCost(theta0, theta1);
        if (prevCost - currCost < 1e-6)
        {
            break;
        }
    }
}

int main()
{
    string fileName = chooseFile();
    globalFileName = fileName;

    // storing data from file
    double data[MAX_ROWS][MAX_COLS];
    readCsvFile(data, fileName); // calling function for reading file
    printArray(data);

    if (isFileExist)
    {
        double inputArray[MAX_ROWS];
        cout << "input1Array \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            inputArray[i] = data[i][0];
            globalInputArray[i] = inputArray[i];
            cout << inputArray[i] << "\n";
        }

        double outputArray[MAX_ROWS];
        cout << "outputArray \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            outputArray[i] = data[i][1];
            globalOutputArray[i] = outputArray[i];
            cout << outputArray[i] << "\n";
        }
    }

    // read hypothesis parameters for model one
    cout << "enter hypothesis parameters: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
```

```
    cin >> theta1;

    double theta0ModelOne = theta0;
    double theta1ModelOne = theta1;

    // train the hypothesis function
    train();

    double xValue;
    cout << "enter x value to predict y: ";
    cin >> xValue;
    double yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // compute cost
    double costModel1 = computeCost(theta0, theta1);
    cout << "costModel1 = " << costModel1 << "\n";

    // find the linear regression parameters
    findLinearRegressionParameters();

    //  train another model two
    cout << "enter hypothesis parameters for model two: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;
    double theta0ModelTwo = theta0;
    double theta1ModelTwo = theta1;

    // train the hypothesis function
    train();

    yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // find the linear regression parameters
    findLinearRegressionParameters();

    // compute cost
    double costModel2 = computeCost(theta0, theta1);
    cout << "costModel2 = " << costModel2 << "\n";

    // deciding best model
```

```
    if (costModel1 < costModel2)
    {
        cout << "the best model is model 1 with \ntheta0 = " << theta0ModelOne <<
" & \ntheta1 = " << theta1ModelOne << "\n";
    }
    else
    {
        cout << "the best model is model 2 with \ntheta0 = " << theta0ModelTwo <<
" & \ntheta1 = " << theta1ModelTwo << "\n";
    }

    return 0;
}
```

Output: -

```
PS D:\mscs\sem4\ann\assignments\assignment4LinearReg
\" ; if ($?) { g++ assignment4RegressionRunTime.cpp
File to Train = data.txt
data.txt is opened successfully
0 1
2 5
3 7
4 9
5 11
6 13
input1Array
0
2
3
4
5
6
outputArray
1
5
7
9
11
13
```

```
enter hypothesis parameters:
theta0 = 1
theta1 = 1
enter x value to predict y: 7
predicted y value: 14.9024
costModel1 = 0.00365684

total iterations = 10, duration = 0 seconds
suggested parameters = theta0 = 8, theta1 = 0
enter hypothesis parameters for model two:
theta0 = 2
theta1 = 2
predicted y value: 14.5678

total iterations = 10, duration = 0 seconds
suggested parameters = theta0 = 8, theta1 = 0
costModel2 = 0.0716336
the best model is model 1 with
theta0 = 1 &
theta1 = 1
```

Part 5: -

Repeat the Experiment for following ranges of parameters and fill the table.

Range	Number of Iterations	Time in Seconds
1 to 10		
1 to 100		
1 to 1000		
1 to 10000		

Source Code: -

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <cmath>
#include <chrono> // for seconds counting
using namespace std;
using namespace std::chrono;

// global variables
const double ALPHA = 0.01; // learning rate
int n;                     // number of data points
const int MAX_ROWS = 100;
const int MAX_COLS = 100;

int globalNumRows = 0;           // dynamically gets number of rows from file
int globalNumCols = 0;           // dynamically gets number of cols from file
double x[MAX_ROWS], y[MAX_COLS]; // input data
double theta0, theta1;           // hypothesis parameters

string globalFileName;
bool isFileExist = false;

double globalInputArray[MAX_ROWS];
double globalOutputArray[MAX_ROWS];

// input file from user
string chooseFile()
```

```
{
    string fileName;
    cout << "File to Train = ";
    cin >> fileName;
    return fileName;
}

// function to read csv file
bool readCsvFile(double data[MAX_ROWS][MAX_COLS], string fileName)
{
    int numRows = 0;
    int numCols = 0;
    ifstream file(fileName);
    if (file.is_open())
    {
        cout << "\033[32m" << fileName << " is opened successfully"
             << "\033[0m" << endl;
        string line; // storing each line of the file
        int row = 0; // to keep track the current row, by default representing
first row

        // getline (input stream, line by line, delimiter)
        while (getline(file, line) && row < MAX_ROWS)
        {
            stringstream ss(line); // Use stringstream to split each line into
fields
            string field;          // store each field / csv of the line
            int col = 0;           // to keep track the current col, by default
representing first col

            while (getline(ss, field, ',') && col < MAX_COLS)
            {                                  // read fields upto maximum number
of columns
                data[row][col] = stod(field); // storing the data in the
respective column
                col++;
            }
            row++;
            numRows++;
            if (col > numCols)
            {
                numCols = col;
                globalNumCols = numCols;
            }
        }
```

```
        file.close();
        isFileExist = true;
    }
    else
    { // if the file was not opened successfully
        cout << "\033[31m"
             << "Failed to open file: " << fileName << "\033[0m" << endl;
    }
    globalNumRows = numRows;
    return isFileExist;
}

void printArray(double data[MAX_ROWS][MAX_COLS])
{
    for (int i = 0; i < globalNumRows; i++)
    { // for each row
        for (int j = 0; j < globalNumCols; j++)
        { // for each column
            cout << data[i][j] << " ";
        }
        cout << endl;
    }
}

// computes the predicted value of y for a given value of x
double predict(double xValue)
{
    double yPred = theta0 + theta1 * xValue; // hypothesis function
    return yPred;
}
// computes the cost of the hypothesis function with parameters theta0 and theta1
double computeCost(double t0, double t1)
{
    double cost = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = t0 + t1 * globalInputArray[i]; // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        cost += diff * diff;                      // squared difference
    }
    cost /= (2 * globalNumRows); // average squared difference
    return cost;
}
```

```
// updates the hypothesis parameters using gradient descent
void updateParameters()
{
    double theta0Sum = 0, theta1Sum = 0;
    for (int i = 0; i < globalNumRows; i++)
    {
        double h = predict(globalInputArray[i]);  // hypothesis function
        double diff = h - globalOutputArray[i];   // difference between
hypothesis and actual value
        theta0Sum += diff;                        // partial derivative of cost
with respect to theta0
        theta1Sum += diff * globalInputArray[i]; // partial derivative of cost
with respect to theta1
    }
    theta0 -= ALPHA * theta0Sum / globalNumRows; // update theta0
    theta1 -= ALPHA * theta1Sum / globalNumRows; // update theta1
}

// find the linear regression parameters with a range constraint of 1 to
max_theta0
void findLinearRegressionParameters(int range) {
    double suggestedCost = INFINITY; // initialize to positive infinity
    double suggestedTheta0, suggestedTheta1;
    int iterations = 0;
    auto old = steady_clock::now(); // start timer
    for (int i = 1; i <= range; i++) { // range constraint of 1 to range
        for (int j = 0; j <= range; j++) {
            double theta0 = i; // first parameter value
            double theta1 = j; // second parameter value
            double cost = computeCost(theta0, theta1);
            iterations++;
            if (cost < suggestedCost) {
                suggestedCost = cost;
                suggestedTheta0 = theta0;
                suggestedTheta1 = theta1;
            }
        }
    }
    auto dur = steady_clock::now() - old; // stop timer
    auto duration = duration_cast<seconds>(dur).count();
    // don't know the details about chrono header, auto keyword, steady_clock,
    // followed this link
    // https://www.youtube.com/watch?v=QYaQStudgnE
    // from 9 minutes
    cout << "Range 1 to " << range << ": ";
```

```
    cout << "total iterations = " << iterations << ", duration = " << duration <<
" seconds" << endl;
    cout << "suggested model: theta0 = " << suggestedTheta0 << ", theta1 = " <<
suggestedTheta1 << endl;
}
// trains the hypothesis function using gradient descent
void train()
{
    for (int i = 0; i < MAX_ROWS; i++)
    {
        double prevCost = computeCost(theta0, theta1);
        updateParameters();
        double currCost = computeCost(theta0, theta1);
        if (prevCost - currCost < 1e-6)
        {
            break;
        }
    }
}

int main()
{
    string fileName = chooseFile();
    globalFileName = fileName;

    // storing data from file
    double data[MAX_ROWS][MAX_COLS];
    readCsvFile(data, fileName); // calling function for reading file
    printArray(data);

    if (isFileExist)
    {
        double inputArray[MAX_ROWS];
        cout << "input1Array \n";
        for (int i = 0; i < globalNumRows; i++)
        {
            inputArray[i] = data[i][0];
            globalInputArray[i] = inputArray[i];
            cout << inputArray[i] << "\n";
        }

        double outputArray[MAX_ROWS];
        cout << "outputArray \n";
        for (int i = 0; i < globalNumRows; i++)
        {
```

```
            outputArray[i] = data[i][1];
            globalOutputArray[i] = outputArray[i];
            cout << outputArray[i] << "\n";
        }
    }

    // read hypothesis parameters for model one
    cout << "enter hypothesis parameters: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;

    double theta0ModelOne = theta0;
    double theta1ModelOne = theta1;

    // train the hypothesis function
    train();

    double xValue;
    cout << "enter x value to predict y: ";
    cin >> xValue;
    double yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // compute cost
    double costModel1 = computeCost(theta0, theta1);
    cout << "costModel1 = " << costModel1 << "\n";

    // find the linear regression parameters for various ranges
    findLinearRegressionParameters(10);
    findLinearRegressionParameters(100);
    findLinearRegressionParameters(1000);
    findLinearRegressionParameters(10000);

    //  train another model two
    cout << "enter hypothesis parameters for model two: " << endl;
    cout << "theta0 = ";
    cin >> theta0;
    cout << "theta1 = ";
    cin >> theta1;
    double theta0ModelTwo = theta0;
    double theta1ModelTwo = theta1;

    // train the hypothesis function
```

```
    train();

    yPred = predict(xValue);
    cout << "predicted y value: " << yPred << endl;

    // find the linear regression parameters for various ranges
    findLinearRegressionParameters(10);
    findLinearRegressionParameters(100);
    findLinearRegressionParameters(1000);
    findLinearRegressionParameters(10000);

    // compute cost
    double costModel2 = computeCost(theta0, theta1);
    cout << "costModel2 = " << costModel2 << "\n";

    // deciding best model

    if (costModel1 < costModel2)
    {
        cout << "the best model is model 1 with \ntheta0 = " << theta0ModelOne <<
" & \ntheta1 = " << theta1ModelOne << "\n";
    }
    else
    {
        cout << "the best model is model 2 with \ntheta0 = " << theta0ModelTwo <<
" & \ntheta1 = " << theta1ModelTwo << "\n";
    }

    return 0;
}
```

Output: -

```
File to Train = data.txt
data.txt is opened successfully
0 1
2 5
3 7
4 9
5 11
6 13
input1Array
0
2
3
4
5
6
outputArray
1
5
7
9
11
13
```

```
enter hypothesis parameters:
theta0 = 1
theta1 = 1
enter x value to predict y: 7
predicted y value: 14.9024
costModel1 = 0.00365684
Range 1 to 10: total iterations = 110, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 100: total iterations = 10100, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 1000: total iterations = 1001000, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 10000: total iterations = 100010000, duration = 5 seconds
suggested model: theta0 = 1, theta1 = 2
enter hypothesis parameters for model two:
theta0 = 2
theta1 = 2
predicted y value: 14.5678
Range 1 to 10: total iterations = 110, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 100: total iterations = 10100, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 1000: total iterations = 1001000, duration = 0 seconds
suggested model: theta0 = 1, theta1 = 2
Range 1 to 10000: total iterations = 100010000, duration = 5 seconds
suggested model: theta0 = 1, theta1 = 2
costModel2 = 0.0716336
the best model is model 1 with
theta0 = 1 &
theta1 = 1
```