**NEURAL NETWORK**

**Groupmate’s:**

1-M. Umair Shahid (20F-0374)

2-Shahzaib Virk (20F-1081)

3-Mustafa Qureshi (20F-0344)

**KFoldCV.h:**

#include <map>

#include <vector>

#include <algorithm>

using namespace std;

class KFoldCV

{

public:

KFoldCV(map<int, int> indexLabel, int K);

virtual ~KFoldCV();

vector<int> getTrainingIndex(int K);

vector<int> getTestIndex(int K);

protected:

map<int, vector<int> > mapTraining;

map<int, vector<int> > mapTesting;

int numOfFold;

private:

};

**KFoldCV.cpp:**

#include "KFoldCV.h"

KFoldCV::KFoldCV(map<int, int> indexLabel, int K) {

numOfFold = K;

map<int, std::vector<int> > mapClass;

//Classify data for each class

for (map<int, int>::iterator it = indexLabel.begin(); it != indexLabel.end(); it++) {

mapClass[it->second].push\_back(it->first);

}

//Create data training and data testing for each class

for (map<int, vector<int> >::iterator it = mapClass.begin(); it != mapClass.end(); it++) {

const int size = mapClass[it->first].size();

//Randomize data

random\_shuffle(mapClass[it->first].begin(), mapClass[it->first].end());

//Distribute data for testing to each fold

vector<int> flag;

int foldNo = 0;

for (int i = 0; i < size; i++) {

flag.push\_back(++foldNo);

if (foldNo == numOfFold) {

foldNo = 0;

}

}

for (int i = 1; i <= numOfFold; i++) {

int idxFlag = 0;

while (idxFlag < size) {

if (flag[idxFlag] == i) {

mapTesting[i].push\_back(mapClass[it->first][idxFlag]);

}

else {

mapTraining[i].push\_back(mapClass[it->first][idxFlag]);

}

idxFlag++;

}

}

}

}

KFoldCV::~KFoldCV() {

}

vector<int> KFoldCV::getTrainingIndex(int K) {

return mapTraining[K];

}

vector<int> KFoldCV::getTestIndex(int K) {

return mapTesting[K];

}

**neuralclass.h:**

#include <vector>

#include <math.h>

#include <iostream>

using namespace std;

class NeuralClass

{

public:

NeuralClass();

double weights[30];

double bias = 1;

int hidenlayer[395];

static int hidencount;

int outputlayer[395];

static int outputcount;

void generateWeights();

double getHidden(int row,int \*\*firstlayer);

double sigmoidFunc(double y);

double finalResult(double out);

};

**neuralclass.cpp:**

#include "neuralclass.h"

NeuralClass::NeuralClass()

{}

int NeuralClass::hidencount = 0;

int NeuralClass::outputcount = 0;

void NeuralClass::generateWeights()

{

for (int i = 0; i < 30; i++)

{

//weights[i] = 10.0 + static\_cast<double>(rand()) / (static\_cast<double>(RAND\_MAX / (-10.0 - 5.0)));

weights[i] = -2.00 + (rand() % 5);

}

}

double NeuralClass::getHidden(int row, int\*\* firstlayer)

{

generateWeights();

double result = 0.0;

for (int i = 0; i < 29; i++)

{

result += (weights[i] \* (firstlayer[row][i]));

}

result += bias \* weights[29];

hidenlayer[hidencount] = result;

hidencount++;

return result;

}

double NeuralClass::sigmoidFunc(double y)

{

return 1 / (1 + exp(-1 \* y));

}

double NeuralClass::finalResult(double out)

{

double result = 0.0;

result = (out < 0.5 ? 0 : 1);

outputlayer[outputcount] = result;

outputcount++;

return out < 0.5 ? 0 : 1;

}

**main.cpp:**

#include "neuralclass.h"

#include <iostream>

#include "KFoldCV.h"

#include <map>

#include<fstream>

#include<string>

#include<cmath>

#include <cstdio>

using namespace std;

class DATA

{

public:

int skl, sex, age, add, Fsize, Pstatus, Medu, Fedu, Ttime, Stime,

fail, sklsup, famsup, paid, activities, nursery, higher, internet, romantic,

famrel, Ftime, Goout, Dalc, Walc, health, absences, G1, G2, G3, result;

DATA()

{

skl = 0; sex = 0; age = 0;

add = 0; Fsize = 0; sklsup = 0;

Pstatus = 0; Medu = 0;

Fedu = 0; Ttime = 0;

Stime = 0; fail = 0;

famsup = 0; paid = 0;

activities = 0; nursery = 0;

higher = 0; internet = 0;

romantic = 0; famrel = 0; Ftime = 0;

Goout = 0; Dalc = 0; Walc = 0;

health = 0; absences = 0;

G1 = 0; G2 = 0; G3 = 0; result = 0;;

}

};

void DataSet(int& s, DATA\*& temp, int\*\*& arr)

{

ifstream read("data.txt");

//counting the number of Students stored in file

string t; int count = 0, count2 = 0;

while (!read.eof())

{

getline(read, t);

count2++;

}

read.close();

if (count2 == 0)

{

cout << "File Is Empty.\n";

system("pause");

return;

}

//storing DATA in array

temp = new DATA[count2];

arr = new int\* [count2];

for (int i = 0; i < count2; i++)

arr[i] = new int[30];

s = count2;

count = 0;

read.open("data.txt");

while (!read.eof())

{

read >> temp[count].skl >> temp[count].sex >> temp[count].age >> temp[count].add >> temp[count].Fsize >> temp[count].Pstatus

>> temp[count].Medu >> temp[count].Fedu >> temp[count].Ttime >> temp[count].Stime

>> temp[count].fail >> temp[count].sklsup >> temp[count].famsup >> temp[count].paid >> temp[count].activities

>> temp[count].nursery >> temp[count].higher >> temp[count].internet >> temp[count].romantic >> temp[count].famrel

>> temp[count].Ftime >> temp[count].Goout >> temp[count].Dalc >> temp[count].Walc >> temp[count].health

>> temp[count].absences >> temp[count].G1 >> temp[count].G2 >> temp[count].G3 >> temp[count].result;

count++;

}

read.close();

count = 0;

read.open("data.txt");

while (!read.eof())

{

read >> arr[count][0] >> arr[count][1] >> arr[count][2] >> arr[count][3] >> arr[count][4] >> arr[count][5] >> arr[count][6]

>> arr[count][7] >> arr[count][8] >> arr[count][9] >> arr[count][10] >> arr[count][11] >> arr[count][12] >> arr[count][13]

>> arr[count][14] >> arr[count][15] >> arr[count][16] >> arr[count][17] >> arr[count][18] >> arr[count][19] >> arr[count][20]

>> arr[count][21] >> arr[count][22] >> arr[count][23] >> arr[count][24] >> arr[count][25] >> arr[count][26] >> arr[count][27]

>> arr[count][28] >> arr[count][29];

count++;

}

read.close();

}

int predict(int\*\* arr, int row, float\* weight)

{

float activation = weight[0];

for (int i = 0; i < 29; i++)

{

activation += (arr[row][i] \* weight[i + 1]);

}

if (activation >= 0.0)

return 1;

else

return 0;

}

void Estimate\_W(int\*\* arr, float lRate, float epoch, float\*& weights)

{

weights = new float[30];

for (int i = 0; i < 30; i++)

{

weights[i] = 0.0;

}

for (int i = 0; i <= epoch; i++)

{

float count = 0.0;

for (int j = 0; j < 395; j++)

{

float prediction = predict(arr, j, weights);

float error = (arr[j][29]) - prediction;

count += pow(error, 2);

weights[0] = weights[0] + lRate \* error;

for (int k = 0; k < 29; k++)

{

weights[k + 1] = weights[k + 1] + lRate \* error \* arr[j][k];

}

}

cout << "epoch :: " << i << " Learning Rate :: " << lRate << " error :: " << count << endl;

}

}

void Estimate\_W\_Kfold(int\*\* arr, vector<int> d, float lRate, float epoch, float\*& weights)

{

cout << endl;

weights = new float[30];

for (int i = 0; i < 30; i++)

{

weights[i] = 0.0;

}

for (int i = 0; i <= epoch; i++)

{

float count = 0.0;

for (int j = 0; j < d.size(); j++)

{

float prediction = predict(arr, (d.at(j) - 1), weights);

float error = (arr[d.at(j) - 1][29]) - prediction;

count += pow(error, 2);

weights[0] = weights[0] + lRate \* error;

for (int k = 0; k < 29; k++)

{

weights[k + 1] = weights[k + 1] + lRate \* error \* arr[d.at(j) - 1][k];

}

}

cout << "epoch :: " << i << " Learning Rate :: " << lRate << " error :: " << count << endl;

}

}

int main()

{

int size, \*\* arr;

DATA\* obj;

DataSet(size, obj, arr);

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << " 'Neural Network' " << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

getchar();

NeuralClass NN;

for (int i = 0; i < 395; i++)

{

double y = NN.getHidden(i, arr);

cout << "Student " << (i + 1) << " ESTIMATION : " << NN.finalResult(NN.sigmoidFunc(y)) << " Result : " << arr[i][29] << endl;

//NN.finalResult(NN.sigmoidFunc(y));

}

cout << " !! Network Created !!" << endl;

cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

int prediction;

/\*float weights[31] = { -2.1,0.200,-0.27,0.29,-0.203,0.230,-0.290,0.2,-0.212,-0.256,-0.288,0.15,0.243,-0.20067,0.27895,0.2123,-0.2153,0.237,-0.21,-0.29,0.219,-0.2501,0.2001,-0.203,-0.298,0.2,0.2,0.2,-0.2,-0.27,0.22 };

cout << "--------------------------------------------Prediction-------------------------------------" << endl;

cout << " :: Before Training Weights :: " << endl << endl;

getchar();

for (int i = 0; i < size; i++)

{

prediction = predict(arr, i, weights);

cout << "Student " << (i + 1) << " predection = " << prediction << " Actual Result = " << obj[i].result << endl;

}\*/

getchar();

cout << "------------------------------------!Training Weights!------------------------------------" << endl;;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

float\* predicted\_weights;

cout << "-----------------------------------------------------------------------------------------" << endl;

Estimate\_W(arr, 0.1, 9, predicted\_weights);

cout << "--- 'PREDICTED WEIGHTS' ---" << endl;

for (int i = 0; i < 30; i++)

{

cout << predicted\_weights[i] << " ";

}

cout << "\n-----------------------------------------------------------------------------------------" << endl;

cout << endl << endl;

//getchar();

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "--------------------------------------------Prediction-------------------------------------" << endl;

cout << " :: After Training Whole DATA :: " << endl << endl;

getchar();

for (int i = 0; i < size; i++)

{

prediction = predict(arr, i, predicted\_weights);

cout << "Student " << (i + 1) << " predection = " << prediction << " Actual Result = " << obj[i].result << endl;

}

//getchar();

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << "-------------------------------------------K-Fold CV----------------------------------------" << endl;

map<int, int> Total\_index;

for (int i = 1; i <= 395; i++)

Total\_index[i] = 0;

// Get index-data for experiment

KFoldCV cv(Total\_index, 5);

for (int i = 1; i <= 5; i++)

{

getchar();

cout << "\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

cout << " !! Experiment !! " << i << endl;

cout << "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~" << endl;

getchar();

cout << "\nTraining DATA Indexes ::\n " << endl;

vector<int> data = cv.getTrainingIndex(i);

for (int i = 0; i < data.size(); i++)

{

cout << data.at(i) << ",";

}

cout << endl;

Estimate\_W\_Kfold(arr, data, 0.1, 5, predicted\_weights);

cout << "\n--- 'PREDICTED WEIGHTS' ---" << endl;

for (int i = 0; i < 30; i++)

{

cout << predicted\_weights[i] << " ";

}

cout << "\n\nTesting DATA Indexes :: ";

cout << endl;

vector<int> data2 = cv.getTestIndex(i);

for (int i = 0; i < data2.size(); i++)

{

cout << data2.at(i) << ",";

}

cout << "\n\n:: Testing DATA :: " << endl << endl;

for (int i = 0; i < data2.size(); i++)

{

prediction = predict(arr, (data2.at(i) - 1), predicted\_weights);

cout << "Student " << (data2.at(i)) << " predection = " << prediction << " Actual Result = " << obj[(data2.at(i) - 1)].result << endl;

}

cout << "\n-----------------------------------------------------------------------------------------" << endl;

}

system("pause>0");

return 0;

}

**OutPut:**







