Q1(a):

The matrix part is done using library [Eigen 3.3.7](http://eigen.tuxfamily.org/index.php?title=Main_Page).

C++ code, compiled using “g++14 -std=c++17 -I ./eigen3/ \*.cpp”, run by “./a.out”.

I combined the learning data set together and learned data once.

“data” is responsible for reading and storing input data, “h” is responsible for computing hypothesis from data and doing test over data, “main” performs the learning and testing.

To display the results better, I transposed the output vector to row vectors so that they can fit in one screen. The output is as follows:

pi is: 0.5

mu1 is: 4.846 4.902 8.28 9.858 10.68 9.342 6.198 4.602 4.978 5.998 10.498 9.662 8.448 7.66 5.322 4.412 4.438 6.36 10.02 6.84 5.6 4.764 4.436 4.366 4.758 6.392 10.126 9.514 7.924 6.302 4.532 4.64 4.812 5.274 7.596 7.948 8.518 7.668 5.634 4.226 4.612 4.718 4.924 5.762 7.362 8.612 5.764 4.408 4.29 5.038 7.032 7.762 8.782 7.908 5.516 4.548 4.456 4.758 8.606 10.226 8.702 6.198 5.002 4.944

mu2 is: 5.158 4.954 5.654 9.55 8.618 5.482 5.478 4.896 4.782 4.75 8.194 10.272 7.03 5.272 4.864 4.736 4.864 5.058 9.814 8.438 4.92 5.068 4.762 4.882 4.686 5.76 10.572 8.02 6.306 5.47 5.154 4.664 4.988 6.278 10.67 9.814 9.532 8.804 5.84 4.588 4.832 5.732 10.402 9.132 7.756 9.324 8.338 4.732 4.704 5.334 9.224 9.968 7.304 9.668 8.934 4.968 4.932 5.002 5.84 9.202 10.99 9.72 6.41 5.186

sigma(diagonal) is: 40.0697 39.8221 43.3939 42.7667 40.4308 42.4314 43.9622 35.7604 37.64 39.7447 41.8872 42.2509 43.9802 40.3992 39.5939 37.2223 35.9498 40.8265 42.7855 46.7263 39.7868 38.7918 35.4976 36.4921 36.5594 40.4464 40.6215 46.5407 45.7253 43.4879 37.2876 36.9888 37.8723 42.1378 41.9909 44.3984 45.4073 45.2837 42.0512 32.9046 37.2726 39.7133 40.3273 44.528 46.2997 43.6842 43.728 35.2429 35.6571 40.4655 44.1844 44.5242 46.087 44.6127 43.8117 38.7293 37.5637 37.5607 45.2406 42.5901 42.5655 43.9982 42.6719 40.0801

w is: -0.0154362 0.0161948 0.0393757 -0.00457235 0.0619536 0.153649 0.0191925 0.055181 0.0868793 -0.0168463 0.120623 0.00442281 0.0259352 0.15032 -0.00108072 -0.0418335 0.0720875 0.0453185 0.0106403 -0.0269965 -0.00676469 -0.05822 -0.0418545 0.0276396 0.0818634 0.0226895 -0.0212766 0.0179884 0.0921044 -0.00838512 -0.0758181 -0.00457054 0.086915 -0.012448 -0.0607205 -0.0338675 -0.0283565 -0.0884285 -0.0411873 -0.0351354 -0.025256 0.00682692 -0.2129 -0.0724104 -0.00984092 0.0316057 0.0265646 -0.0510064 0.0227854 -0.0958112 -0.0755238 -0.0469715 0.0615342 -0.0353748 -0.0678257 -0.00184471 -0.0423693 -0.0734694 0.087231 0.0529176 -0.00980751 -0.0598071 -0.00544649 0.00300349

w0 is: 0.0986343

the accuracy on the test set is : 0.890909

The code is as follows:

data.h:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef \_DATA\_H#define \_DATA\_H

#include <string>

#include <vector>

#include <utility>

#include <Eigen/Dense>

struct data{

data();

std::vector<Eigen::VectorXd> coordinates;

std::vector<int> labels;

data(int a);

data(std::string input, std::string target);

void add(const data& other);

};

#endif

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

data.cpp:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <sstream>#include <string>

#include <vector>

#include <utility>

#include <fstream>

#include "data.h"

#include <Eigen/Dense>

using namespace std;

data::data(){}

void data::add(const data& other){

coordinates.insert(coordinates.end(), other.coordinates.begin(), other.coordinates.end());

labels.insert(labels.end(), other.labels.begin(), other.labels.end());

}

data::data(string input, string target){

ifstream in1(input);

ifstream in2(target);

string line;

while(getline(in1,line)){

vector<int> temp;

//data

while (line.find(",") != string::npos){

istringstream line\_stream(line);

int x;

line\_stream >> x;

temp.push\_back(x);

line = line.substr(line.find(",") + 1);

}

istringstream line\_stream(line);

int x;

line\_stream >> x;

temp.push\_back(x);

Eigen::VectorXd vec(temp.size());

for(uint i = 0; i < temp.size(); ++i){

vec(i) = temp.at(i);

}

coordinates.push\_back(vec);

//label

int label;

in2 >> label;

//setup

labels.push\_back(label);

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

h.h:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#ifndef \_H\_H#define \_H\_H

#include <Eigen/Dense>

#include "data.h"

class h{

double pi;

Eigen::VectorXd mu1;

Eigen::VectorXd mu2;

Eigen::MatrixXd sigma;

Eigen::VectorXd w;

double w0;

public: h(const data& D);

//given a coordinate, predict its label.

int guess(Eigen::VectorXd vec) const;

//given a set of coordinates and its labels, output the propability that it guesses wrongly.

double loss(const data& D) const;

};

#endif

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

h.cpp:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <vector>#include <map>

#include <iostream>

#include <cmath>

#include <Eigen/Dense>

#include "h.h"

#include "data.h"

using namespace Eigen;

h::h(const data& D){

int Dimen = D.coordinates.at(0).size();

int N = D.labels.size();

int N1 = 0;

int N2 = 0;

for(const auto& label: D.labels){

if (label == 5){

++N1;

} else {

++N2;

}

}

pi = 1.0 \* N1 / N;

mu1 = VectorXd::Zero(Dimen);

mu2 = VectorXd::Zero(Dimen);

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

if (D.labels.at(i) == 5){

mu1 += D.coordinates.at(i);

} else {

mu2 += D.coordinates.at(i);

}

}

mu1 /= N1;

mu2 /= N2;

Eigen::MatrixXd S1(Dimen,Dimen);

Eigen::MatrixXd S2(Dimen,Dimen);

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

if (D.labels.at(i) == 5){

S1 += (D.coordinates.at(i) - mu1) \* ((D.coordinates.at(i) - mu1).transpose());

} else {

S2 += (D.coordinates.at(i) - mu2) \* ((D.coordinates.at(i) - mu2).transpose());

}

}

S1 /= N1;

S2 /= N2;

sigma = 1.0 \* N1 / N \* S1 + 1.0 \* N2 / N \* S2;

MatrixXd INVERSE = sigma.inverse();

w = INVERSE \* (mu1 - mu2);

w0 = -0.5 \* mu1.transpose() \* INVERSE \* mu1;

w0 += 0.5 \* mu2.transpose() \* INVERSE \* mu2;

w0 += log(pi/(1-pi));

std::cout << "pi is: " << pi << std::endl << std::endl;

std::cout << "mu1 is: " << mu1.transpose() << std::endl << std::endl;

std::cout << "mu2 is: " << mu2.transpose() << std::endl << std::endl;

std::cout << "sigma(diagonal) is: " << sigma.diagonal().transpose() << std::endl << std::endl;

std::cout << "w is: " << w.transpose() << std::endl << std::endl;

std::cout << "w0 is: " << w0 << std::endl << std::endl;

};

int h::guess(VectorXd vec) const{

if (1 / (1 + exp(-(w.transpose() \* vec + w0))) >= 0.5){

return 5;

} else {

return 6;

}

}

double h::loss(const data& D) const{

int total = 0;

int right = 0;

auto it = D.labels.begin();

for(const auto& coords: D.coordinates){

if (guess(coords) == \*it){

++right;

}

++it;

++total;

}

return 1 - 1.0 \* right / total;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

main.cpp:

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#include <string>#include <iostream>

#include "data.h"

#include "h.h"

int main(){

data train\_set;

for (int i = 1; i <= 10; ++i){

std::string s1("trainData" + std::to\_string(i) + ".csv");

std::string s2("trainLabels" + std::to\_string(i) + ".csv");

train\_set.add(data(s1, s2));

}

data test\_data("testData.csv", "testLabels.csv");

h hypothesis(train\_set);

std::cout << "the accuracy on the test set is : " << 1 - hypothesis.loss(test\_data) << std::endl;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*