Q1(b):

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”.

The output is as follows:

when labmda is 0, the accuracy is 0.852when labmda is 500, the accuracy is 0.873

when labmda is 1000, the accuracy is 0.881

when labmda is 1500, the accuracy is 0.881

when labmda is 2000, the accuracy is 0.882

when labmda is 2500, the accuracy is 0.882

when labmda is 3000, the accuracy is 0.884

when labmda is 3500, the accuracy is 0.884

when labmda is 4000, the accuracy is 0.885

when labmda is 4500, the accuracy is 0.885

when labmda is 5000, the accuracy is 0.885

when labmda is 5500, the accuracy is 0.884

when labmda is 6000, the accuracy is 0.886

when labmda is 6500, the accuracy is 0.887

when labmda is 7000, the accuracy is 0.887

when labmda is 7500, the accuracy is 0.887

when labmda is 8000, the accuracy is 0.887

when labmda is 8500, the accuracy is 0.887

when labmda is 9000, the accuracy is 0.887

when labmda is 9500, the accuracy is 0.886

when labmda is 10000, the accuracy is 0.885

when labmda is 10500, the accuracy is 0.884

when labmda is 11000, the accuracy is 0.884

when labmda is 11500, the accuracy is 0.884

when labmda is 12000, the accuracy is 0.884

when labmda is 12500, the accuracy is 0.885

when labmda is 13000, the accuracy is 0.885

when labmda is 13500, the accuracy is 0.884

when labmda is 14000, the accuracy is 0.884

when labmda is 14500, the accuracy is 0.885

when labmda is 15000, the accuracy is 0.884

when labmda is 15500, the accuracy is 0.884

when labmda is 16000, the accuracy is 0.885

when labmda is 16500, the accuracy is 0.884

when labmda is 17000, the accuracy is 0.883

when labmda is 17500, the accuracy is 0.883

when labmda is 18000, the accuracy is 0.884

when labmda is 18500, the accuracy is 0.884

when labmda is 19000, the accuracy is 0.884

when labmda is 19500, the accuracy is 0.884

when labmda is 20000, the accuracy is 0.884

the best labmda is 6500

the test accuracy 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;

temp.push\_back(1);

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

if (label == 5){

labels.push\_back(1);

} else {

labels.push\_back(0);

}

}

}

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

h.h:

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

#ifndef \_H\_H

#define \_H\_H

#include <Eigen/Dense>

#include "data.h"

class h{

double lambda;

Eigen::VectorXd w;

public:

h(double lambda, 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;

double sigmoid(double a){

return 1.0 / (1.0 + exp(-a));

}

h::h(double lambda, const data& D): lambda{lambda}{

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

int N = D.coordinates.size();

MatrixXd X(Dimen, N);

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

X.col(i) = D.coordinates.at(i);

}

w = VectorXd::Random(Dimen);

w /= 100;

for(int i = 0; i < 10; ++i){

MatrixXd R = MatrixXd::Zero(N,N);

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

R(i,i) = sigmoid(w.transpose() \* X.col(i)) \* (1 - sigmoid(w.transpose() \* X.col(i)));

}

MatrixXd H = X \* R \* X.transpose() + lambda \* MatrixXd::Identity(Dimen,Dimen);

VectorXd gradient = VectorXd::Zero(Dimen);

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

gradient.noalias() += (sigmoid(w.transpose() \* D.coordinates.at(i)) - D.labels.at(i)) \* D.coordinates.at(i);

}

gradient.noalias() += lambda \* w;

w.noalias() -= H.inverse() \* gradient;

}

};

int h::guess(VectorXd vec) const{

if ((- w.transpose()) \* vec >= 0){

return 0;

} else {

return 1;

}

}

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"

const double LAMBDA = 20000;

const int FOLD = 10;

int main(){

data\* train\_set = new data[FOLD];

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

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

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

train\_set[i-1].add(data(s1, s2));

}

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

double best\_accuracy = 0;

double best\_lambda = 0;

for(double lambda = 0; lambda <= LAMBDA; lambda += 500){

double accuracy = 0;

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

data learn;

for (int l = 0; l < FOLD; ++l){

if (l != i){

learn.add(train\_set[l]);

}

}

h hypothesis(lambda, learn);

accuracy += (1 - hypothesis.loss(train\_set[i]));

}

accuracy /= FOLD;

std::cout << "when labmda is " << lambda << ", the accuracy is " << accuracy << std::endl;

if (best\_accuracy < accuracy){

best\_lambda = lambda;

best\_accuracy = accuracy;

}

}

data all;

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

all.add(train\_set[i]);

}

h best\_h(best\_lambda, all);

double test\_accuracy = 1 - best\_h.loss(test\_data);

std::cout << "the best labmda is " << best\_lambda << std::endl;

std::cout << "the test accuracy is " << test\_accuracy << std::endl;

delete[] train\_set;

}

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

