‘Student Name: Panchakshari N Hiremath ID: 1016242440

CPS 580-01 - Artificial Intelligence

Spring 2020

Instructor: Dr. Ju Shen

Final Project - Gender prediction

Program Execution Instruction:

- Compile using g++ train.cpp in the terminal.
- Run using ./a.out Enter the number of training epochs and learning rate.
- The model will be saved as a txt file and the training accuracy will be displayed.

Logistic regression:

The model is saved as a struct in this program.

The data is loaded into 2 arrays, X for the height and weight and, Y for the label. Now the heights and weights are normalized by subtracting the mean and dividing by the standard deviation.

Logistic regression is performed on X with labels Y and gradient descent is used to learn the optimal parameters for the model. After training the model is saved as a text file.

When test.cpp is run, the model is loaded and evaluated on test.cpp.

Logistic regression model:

Evaluate the output of the model on the entire training set.

```
double* eval_full(model M,double** X,double *Y){
      for (int i = 0; i < N; i++)
    {
        Y[i] = 1/(1+exp(-(M.w0+M.w1*X[0][i]+M.w2*X[1][i])));
```

```
        //Logistic regression model
    }
    return Y;
}
```

Implementing the gradient descent procedure:

I have implemented the Gradient descent by taking partial derivatives.

```
for (int j = 0; j < N; j++)
        {
            //GRADIENT DESCENT BY TAKING PARTIAL DERIVATIVES
            w0 += -(Y[j]-Y_cap[j]);
            w1 += -(Y[j]-Y_cap[j])*X[0][j];
            w2 += -(Y[j]-Y_cap[j])*X[1][j];
        }
```

Parameter update consistency:

```
        M.w0 += training_rate*w0/N;
        M.w1 += training_rate*w1/N;
        M.w2 += training_rate*w2/N;
        Y = eval_full(M,X,Y);
```

Sorting training data:

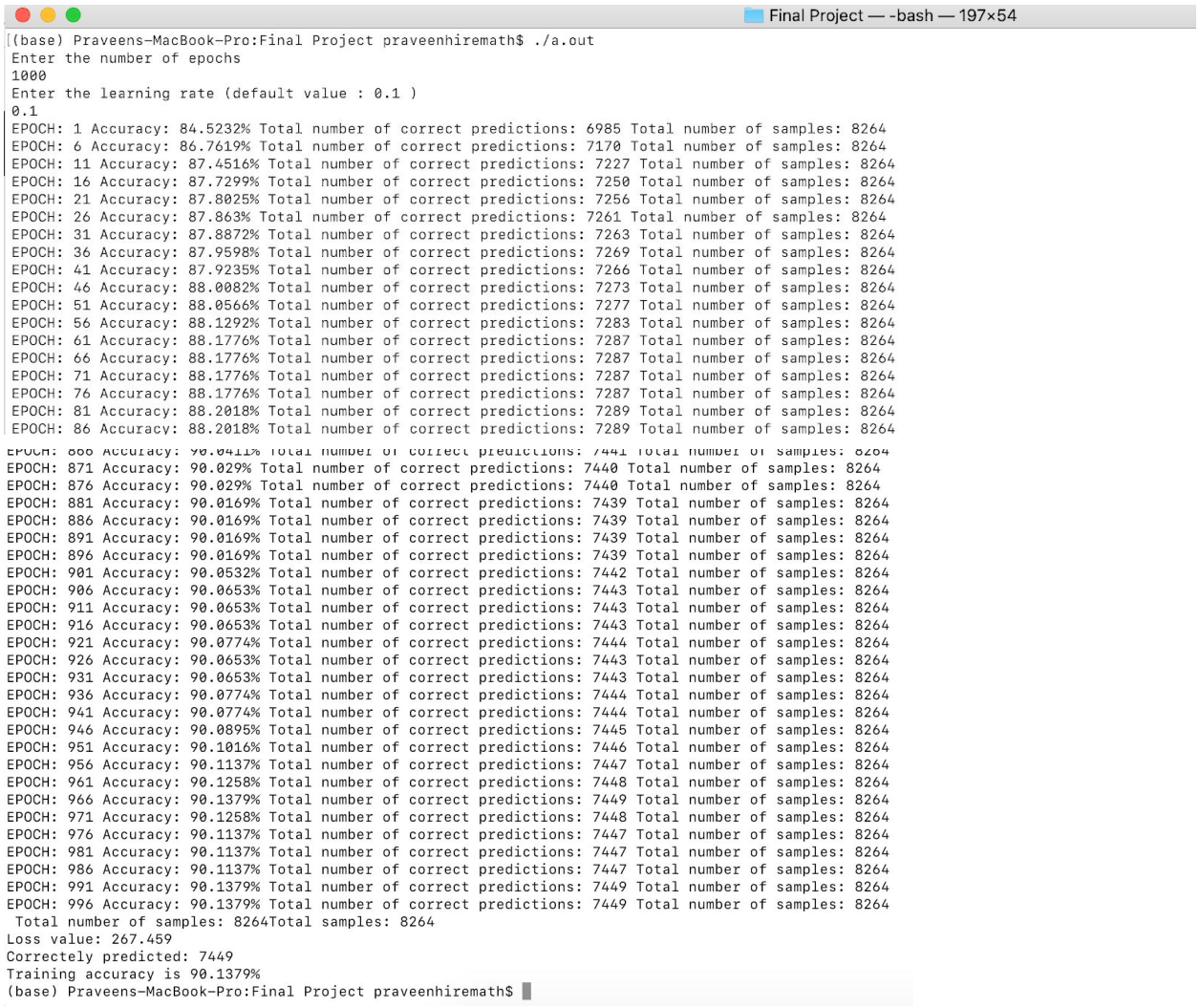
```
double **X = (double **)malloc(2 * sizeof(double *));
    for (int i=0; i<2; i++)
        X[i] = (double *)malloc(N * sizeof(double));
    double avg_height = 0.0;
    double avg_weight = 0.0;
    double std_height = 0.0;
    double std_weight = 0.0;
    int Y[N];
```

Cost function:

```
double Cost(double *Y,int *Y_cap)
{
 static double sum = 0.0;
    for (int i = 0; i < N; i++)
    {
        sum -= Y_cap[i] * log(Y[i]) + (1-Y_cap[i]) * log(1-Y[i]);
    }
    return sum;
}
```

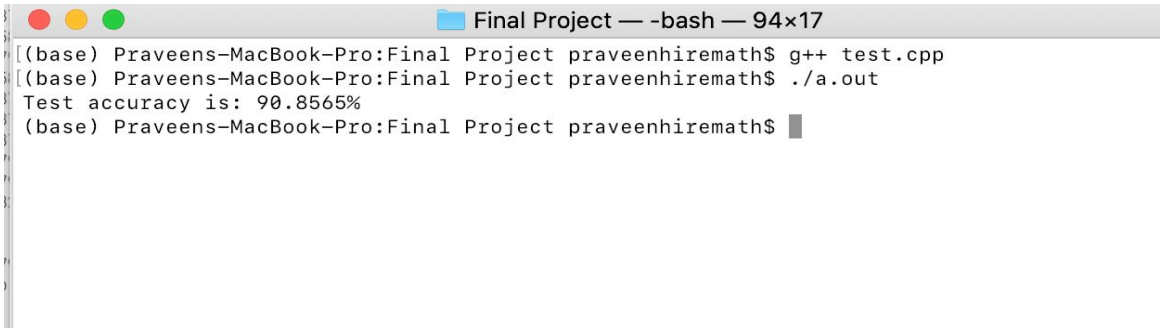
Output :

Training:



Testing:

I will create a module.txt when I run the train.cpp and the test results are shown below.



Source Code :

Train.cpp :

```
#include<iostream>
#include<fstream>
#include<string>
#include<math.h>
#define N 8264 //Number of training examples
using namespace std;


struct model
{
    double w0;
    double w1;
    double w2;
};

double Cost(double *Y,int *Y_cap)
{
    static double sum = 0.0;
    for (int i = 0; i < N; i++)
    {
        sum -= Y_cap[i] * log(Y[i]) + (1-Y_cap[i]) * log(1-Y[i]);//LOSS
FUNCTION BINARY CROSS ENTROPY LOSS

    }
    return sum/N;

}

double* eval_full(model M,double** X,double *Y)//Evaluate the output of the
model on the entire training set
{

    for (int i = 0; i < N; i++)
    {
        Y[i] = 1/(1+exp(-(M.w0+M.w1*X[0][i]+M.w2*X[1][i])));//Logistic
regression model

    }
    return Y;
}
```

```
double accu(double *Y,int *Y_cap)
{
    double sum = 0.0;
    for (int i = 0; i < N; i++)
    {
        sum += (int)(((int)(Y[i]>=0.5))==Y_cap[i]);

    }
    return (100*sum/N);

}
model train(model M,double training_rate,int epochs,double **X,int *Y_cap)
{

    double *Y = new double[N];
    Y = eval_full(M,X,Y);


    for (int i = 0; i < epochs; i++)
    {
        double w0 = 0,w1 = 0,w2 = 0;

        for (int j = 0; j < N; j++)
        {
            //GRADIENT DESCENT BY TAKING PARTIAL DERIVATIVES
            w0 += -(Y[j]-Y_cap[j]);
            w1 += -(Y[j]-Y_cap[j])*X[0][j];
            w2 += -(Y[j]-Y_cap[j])*X[1][j];
        }

        //PARAMETER UPDATE CONSISTENCY
        M.w0 += training_rate*w0/N;
        M.w1 += training_rate*w1/N;
        M.w2 += training_rate*w2/N;
        Y = eval_full(M,X,Y);

        double cost = Cost(Y,Y_cap);
        if(i%5==0)
        {
            cout<<"EPOCH: "<<(i+1);
            cout<<" Accuracy: "<<accu(Y,Y_cap)<<"%";
            cout<<" Total number of correct predictions:
"<<(N*accu(Y,Y_cap)/100);
            cout<<" Total number of samples: "<<N;
            cout<<endl;
```

```
        }

    }
    cout<<" Total number of samples: "<<N;
    return M;




}

int evaluate(model M,double height,double weight)
{
    double test = 1/(1+exp(-(M.w0+M.w1*height+M.w2*weight)));
    if (test >= 0.5)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int main()
{
    ifstream file("train.txt");
    string str;

    //double **X = new double[2][N];
    double **X = (double **)malloc(2 * sizeof(double *));
    for (int i=0; i<2; i++)
        X[i] = (double *)malloc(N * sizeof(double));
    double avg_height = 0.0;
    double avg_weight = 0.0;
    double std_height = 0.0;
    double std_weight = 0.0;
    int Y[N];//Storing the training data

    getline(file, str);//remove the first line which is the data
description
    int i = 0;
    while (getline(file, str))
    {
        Y[i] = (int)(str[0]-48);
```

```
        int a = str.size();
        int b;
        for (int j = 2; j < a; j++)
        {
            if(str[j]==' ')
            {
                b = j;
                break;
            }
        }
        X[0][i] = stod(str.substr(2,b-1));
        X[1][i] = stod(str.substr(b+1,a-b));
        avg_height += X[0][i];
        avg_weight += X[1][i];
        i++;
    }
    avg_weight = avg_weight/N;
    avg_height = avg_height/N;

    for (int i = 0; i < N; i++)
    {
        std_height += (X[0][i]-avg_height)*(X[0][i]-avg_height);
        std_weight += (X[1][i]-avg_weight)*(X[1][i]-avg_weight);
    }
    std_weight = sqrt(std_weight/N);
    std_height = sqrt(std_height/N);


    for (int i = 0; i < N; i++)
    {
        X[0][i] = (X[0][i]-avg_height)/std_height;
        X[1][i] = (X[1][i]-avg_weight)/std_weight;
    }
       model M;
    M.w0=M.w1=M.w2=0.1;

    int epoch;
    double learning_rate;
    cout<<"Enter the number of epochs"<<endl;
    cin >> epoch ;
    cout<<"Enter the learning rate (default value : 0.1 )"<<endl;
    cin >> learning_rate ;
```

```
    M = train(M,learning_rate,epoch,X,Y);
    double *Y_t = new double[N];
    Y_t = eval_full(M,X,Y_t);

    double su = 0;
    for (int i = 0; i < N; i++)
    {
        int t = (Y_t[i]>=0.5)?1:0;
        if (t==Y[i])
        {
            su++;
        }



    }

    cout<<"Total samples: "<<N<<endl;
    cout<<"Loss value: "<<Cost(Y_t,Y)<<endl;
    cout<<"Correctely predicted: "<<su<<endl;
    cout<<"Training accuracy is "<<(su*100)/N<<"%"<<endl;

    ofstream file_m;
    file_m.open("model.txt");
    file_m << to_string(M.w0);
    file_m << "\n";
    file_m << to_string(M.w1);
    file_m << "\n";
    file_m << to_string(M.w2);
    file_m.close();

}
```

Test.cpp

```
#include<iostream>
#include<fstream>
#include<string>
#include <stdlib.h>
#include<math.h>
using namespace std;

struct model
{
    double w0;
```

```
    double w1;
    double w2;
};
int evaluate(model M,double height,double weight)
{
    double test = 1/(1+exp(-(M.w0+M.w1*height+M.w2*weight)));
    if (test >= 0.5)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
int main()
{
    ifstream file("model.txt");
    string str;
    model M;
    getline(file, str);
    M.w0 = atof(str.c_str());
    getline(file, str);
    M.w1 = atof(str.c_str());
    getline(file, str);
    M.w2 = atof(str.c_str());

    ifstream file1("test.txt");


    getline(file1, str);//remove the first line which is the data
description
    double su = 0.0;
    int n = 0;
    while (getline(file1, str))
    {
        int y = (int)(str[0]-48);

        int a = str.size();
        int b;
        for (int j = 2; j < a; j++)
        {
            if(str[j]==' ')
            {
                b = j;
```

```
                break;
            }
        }
        double avg_height = 66.4388;
        double std_height = 3.86941;
        double avg_weight = 161.957;
        double std_weight = 32.2341;
        double heig = stod(str.substr(2,b-1));
        double weig = stod(str.substr(b+1,a-b));
        heig = (heig-avg_height)/std_height;
        weig = (weig-avg_weight)/std_weight;
        int y1 = evaluate(M,heig,weig);
        if (y1==y)
        {
            su++;
        }
        n++;


    }
    cout<<"Test accuracy is: "<<(su*100)/n<<"%"<<endl;

}
```