****

**《程序设计课程实践》设计文档**

# 第4章 第1次作业

**学号： 21052115**

**姓名： 姜曜楠**

完成时间 2022 年 4 月

## 1、题目

## 2、程序设计思路

依次需要进行的步骤有：

1. 数据读入：通过文件读取操作，将外部数据储存在储存数组，并及时检查读取正确性
2. 数据预处理：
   1. 针对housing.txt，主要是进行数据归一化处理，此处采用min-max标准化方法。归一化后，所有维度的数据均分布在0-1范围，且无量纲。
   2. 选择训练数据为前80%个数据，测试数据为后20%个数据
3. 模型训练和测试评估：
   1. 一元线性回归：
      1. 利用训练数据集训练单维度的一元线性回归模型Y = a + bX
      2. 利用测试数据集测试模型，并采用RMSE来评价模型
      3. 分别计算13个维度的一元线性回归模型，并采用Person系数来比较各维度与房价的相关度情况
      4. 采用最相关的维度重新构造一元线性回归模型
   2. 多元线性回归：
      1. 采用梯度下降算法和平方损失函数，利用训练数据集来训练多元线性回归模型，训练最大次数为MAXECHO
      2. 利用测试数据集测试模型，依旧采用RMSE评价模型
      3. 在一元线性回归基础上，寻找最相关的四个维度，重新构造新的多元线性回归模型
      4. 通过提高函数参数，提高内部函数的复用率

## 3、程序源码

#ifndef HOUSING\_H

#define HOUSING\_H

#include <stdbool.h>

#define DATAMAXN 506

#define DATATRAIN (int)(8 \* DATAMAXN / 10)

#define BEFOREMEDV 13

#define NBESTDIMENSION 4

enum dimensionToDigit

{

    CRIM,

    ZN,

    INDUS,

    CHAS,

    NOX,

    RM,

    AGE,

    DIS,

    RAD,

    TAX,

    PTRATIO,

    B,

    LSTAT,

    MEDV,

    INFOMAXN

};

typedef char \*string;

extern string digitToDimension[BEFOREMEDV];

typedef double Data\_T;

Data\_T dataAll[DATAMAXN][INFOMAXN];

/\* 函数定义 \*/

bool GetData(void);

void NormalizeData(void);

#endif

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <errno.h>

#include "housing.h"

#include "unary.h"

#include "multiple.h"

string digitToDimension[BEFOREMEDV] = {

    "CRIM",

    "ZN",

    "INDUS",

    "CHAS",

    "NOX",

    "RM",

    "AGE",

    "DIS",

    "RAD",

    "TAX",

    "PTRATIO",

    "B",

    "LSTAT",

};

int main(void)

{

    if (GetData())

    {

        NormalizeData();

        int dimensionAll[BEFOREMEDV];

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

        {

            dimensionAll[i] = i;

        }

        AllUnaryLinearReg(dimensionAll, BEFOREMEDV);

        int best[NBESTDIMENSION];

        GetNBestUnary(best, NBESTDIMENSION);

        system("pause");

        printf("\n\nAll dimension's ending:");

        MultipleLinearReg(dimensionAll, BEFOREMEDV);

        system("pause");

        printf("\n\n%d Best dimension's ending:", NBESTDIMENSION);

        MultipleLinearReg(best, NBESTDIMENSION);

        system("pause");

        return 0;

    }

    else

    {

        fprintf(stdout, "Error no.%d: %s\n", errno, strerror(errno));

        system("pause");

        exit(errno);

    }

}

bool GetData(void)

{

    FILE \*fp = fopen("D:\\VScode\\Program\\C\\Program\_latter\\Project4\\Entry\\housing.txt", "rb");

    if (fp == NULL)

    {

        fputs("Open error", stderr);

        return false;

    }

    char buffer[256];

    fgets(buffer, 256, fp);

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

    {

        fscanf(fp, "%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",

               &dataAll[i][CRIM], &dataAll[i][ZN], &dataAll[i][INDUS], &dataAll[i][CHAS], &dataAll[i][NOX], &dataAll[i][RM], &dataAll[i][AGE],

               &dataAll[i][DIS], &dataAll[i][RAD], &dataAll[i][TAX], &dataAll[i][PTRATIO], &dataAll[i][B], &dataAll[i][LSTAT], &dataAll[i][MEDV]);

    }

    fclose(fp);

    /\* 数据读取正确性检验 \*/

    // printf("\n%s", buffer);

    // for (int i = 0; i < DATAMAXN; i++)

    // {

    //     printf("\n%8.5lf %8.2lf %8.3lf %3d %8.4lf %8.4lf %8.2lf %8.4lf %3d %8.1lf %8.2lf %8.2lf %8.2lf %8.2lf",

    //            dataAll[i][CRIM], dataAll[i][ZN], dataAll[i][INDUS], dataAll[i][CHAS], dataAll[i][NOX], dataAll[i][RM], dataAll[i][AGE],

    //            dataAll[i][DIS], dataAll[i][RAD], dataAll[i][TAX], dataAll[i][PTRATIO], dataAll[i][B], dataAll[i][LSTAT], dataAll[i][MEDV]);

    // }

    return true;

}

void NormalizeData(void) // x = (x - min)/(max - min)

{

    Data\_T max[INFOMAXN] = {0};

    Data\_T min[INFOMAXN] = {0};

    for (int dimension = 0; dimension < INFOMAXN; dimension++)

    {

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

        {

            if (dataAll[i][dimension] > max[dimension])

            {

                max[dimension] = dataAll[i][dimension];

            }

            if (dataAll[i][dimension] < min[dimension])

            {

                min[dimension] = dataAll[i][dimension];

            }

        }

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

        {

            dataAll[i][dimension] = (dataAll[i][dimension] - min[dimension]) / (max[dimension] - min[dimension]);

        }

    }

}

#ifndef UNARY\_H

#define UNARY\_H

#include "housing.h"

/\* 函数定义 \*/

void UnaryLinearReg(int dimension);

void AllUnaryLinearReg(int \*dimension, int N);

void GetNBestUnary(int \*best, int number);

#endif

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include "unary.h"

typedef struct data

{

    double x; // one dimension from[CRIM, ZN, INDUS, CHAS, NOX, RM, AGE, DIS, RAD, TAX, PTRATIO, B, LSTAT]

    double y; // MEDV

} Data;

Data data[DATAMAXN];

double a; // Y = a + bX

double b; // Y = a + bX

double sumx;

double sumy;

double sumxx;

double sumxy;

double sumyy;

double r[BEFOREMEDV];

bool flag[BEFOREMEDV];

int bestDimension;

string correlateStrength[5] = {

    "Extremely weak",   // |r| < 0.2    -> 0

    "Weak",             // |r| < 0.4    -> 1

    "Medium level",     // |r| < 0.6    -> 2

    "Strong",           // |r| < 0.8    -> 3

    "Extremely strong", // |r| < 1.0    -> 4

};

/\* 局部函数定义 \*/

void initUnary(int dimension);

void processUnary(int dimension);

void deviationUnary(int dimension);

void assessUnary(int dimension);

void bestUnary(void);

/\* 接口实现 \*/

void UnaryLinearReg(int dimension)

{

    initUnary(dimension);

    processUnary(dimension);

    deviationUnary(dimension);

    assessUnary(dimension);

}

void AllUnaryLinearReg(int \*dimension, int N)

{

    for (int i = 0; i < N; i++)

    {

        UnaryLinearReg(dimension[i]);

    }

    bestUnary();

}

void GetNBestUnary(int \*best, int number)

{

    for (int i = 0; i < number; i++)

    {

        best[i] = 0;

    }

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

    {

        flag[i] = false;

    }

    for (int i = 0; i < number; i++)

    {

        bestDimension = 0;

        for (int dimension = 1; dimension < BEFOREMEDV; dimension++)

        {

            if (flag[dimension] == true)

            {

                continue;

            }

            double tempR = r[dimension] > 0 ? r[dimension] : -r[dimension];

            double BestR = r[bestDimension] > 0 ? r[bestDimension] : -r[bestDimension];

            if ((tempR - BestR) > 0.001)

            {

                bestDimension = dimension;

            }

        }

        best[i] = bestDimension;

        flag[bestDimension] = true;

        printf("\nBest[%d]: %s", i + 1, digitToDimension[best[i]]);

    }

}

/\* 局部函数实现 \*/

void initUnary(int dimension)

{

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

    {

        data[i].x = dataAll[i][dimension];

        data[i].y = dataAll[i][MEDV];

    }

    a = b = sumx = sumy = sumxx = sumxy = sumyy = 0;

}

void processUnary(int dimension)

{

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

    {

        sumx += data[i].x;

        sumy += data[i].y;

        sumxx += data[i].x \* data[i].x;

        sumxy += data[i].x \* data[i].y;

        sumyy += data[i].y \* data[i].y;

    }

    b = (sumxy \* DATATRAIN - sumx \* sumy) / (sumxx \* DATATRAIN - sumx \* sumx);

    a = sumy / DATATRAIN - b \* sumx / DATATRAIN;

    printf("\nPrediction(%7s And MENV):\tY = %+4.2lfX %+4.2lf", digitToDimension[dimension], b, a);

}

void deviationUnary(int dimension)

{

    /\*

        RMSE = sqrt(E[(obs-model)^2]) = sqrt(sum((obs - model)^2) / n)

     \*/

    double sum\_ObsAndModel = 0;

    for (int i = DATATRAIN + 1; i < DATAMAXN; i++)

    {

        double model = a + b \* data[i].x;

        sum\_ObsAndModel += pow((data[i].y - model), 2);

    }

    double rmse = sqrt(sum\_ObsAndModel / (DATAMAXN - DATATRAIN - 1)); // 分母确保方差估计量的无偏性

    printf("\nRMSE:\t%.4lf", rmse);

}

void assessUnary(int dimension) // using the Pearson correlation coefficient

{

    /\*

        r = Cov(X,Y)/ sqrt(Var[X]\*Var[Y])

        Cov(X,Y) = E[XY] - E[X]E[Y] = sumxy/n - (sumx/n)\*(sumy/n)

        Var[X] = E[X^2] - (E[X])^2 = sumxx/n - (sumx/n)^2

    \*/

    double covxy = sumxy / DATATRAIN - sumx \* sumy / pow(DATATRAIN, 2);

    double varx = sumxx / DATATRAIN - sumx \* sumx / pow(DATATRAIN, 2);

    double vary = sumyy / DATATRAIN - sumy \* sumy / pow(DATATRAIN, 2);

    r[dimension] = covxy / sqrt(varx \* vary);

    printf("\nPerson:\t%+4.2lf\nExtent:\t%s", r[dimension], correlateStrength[abs((int)(r[dimension] \* 100)) / 20]);

}

void bestUnary(void)

{

    bestDimension = 0;

    for (int dimension = 1; dimension < BEFOREMEDV; dimension++)

    {

        double tempR = r[dimension] > 0 ? r[dimension] : -r[dimension];

        double BestR = r[bestDimension] > 0 ? r[bestDimension] : -r[bestDimension];

        if ((tempR - BestR) > 0.001)

        {

            bestDimension = dimension;

        }

    }

    printf("\n\n\nBest prediction:(%s And MENV): Y = %+.2lfX%+.2lf", digitToDimension[bestDimension], b, a);

    printf("\nPerson:\t%+.2lf\nExtent:\t%s", r[bestDimension], correlateStrength[abs((int)(r[bestDimension] \* 100)) / 20]);

}

#ifndef MULTIPLE\_H

#define MULTIPLE\_H

#include "housing.h"

/\* 函数定义 \*/

void MultipleLinearReg(int \*dimension, int N);

#endif

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <math.h>

#include "multiple.h"

#define MAXECHO 10000

Data\_T W[BEFOREMEDV + 1];

Data\_T prediction[DATAMAXN];

double alpha = 0.0008;

/\* 局部函数定义 \*/

void initMultiple(void);

void processMultiple(int \*dimension, int N);

void deviationMultiple(int \*dimension, int N);

/\* 接口实现 \*/

void MultipleLinearReg(int \*dimension, int N)

{

    initMultiple();

    processMultiple(dimension, N);

    deviationMultiple(dimension, N);

}

/\* 局部函数实现 \*/

void initMultiple(void)

{

    memset(W, 0, sizeof(W));

}

void processMultiple(int \*dimension, int N)

{

    double grad[BEFOREMEDV + 1];

    double loss;

    for (int echo = 0; echo < MAXECHO; echo++)

    {

        loss = 0;

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

        {

            prediction[i] = W[0] \* 1;

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

            {

                prediction[i] += W[dimension[j] + 1] \* dataAll[i][dimension[j]];

            }

            loss += pow(prediction[i] - dataAll[i][MEDV], 2) / 2;

        }

        loss /= DATATRAIN;

        // printf("\nTrain loss: %.3lf", loss);

        if (loss < 0.001)

        {

            break;

        }

        for (int i = 0; i < BEFOREMEDV + 1; i++)

        {

            grad[i] = 0;

        }

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

        {

            grad[0] += (prediction[i] - dataAll[i][MEDV]) \* 1;

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

            {

                grad[dimension[j] + 1] += (prediction[i] - dataAll[i][MEDV]) \* dataAll[i][dimension[j]];

            }

        }

        for (int i = 0; i < BEFOREMEDV + 1; i++)

        {

            W[i] -= alpha \* grad[i];

        }

    }

    printf("\nTrain loss: %.3lf", loss);

}

void deviationMultiple(int \*dimension, int N)

{

    /\*

        RMSE = sqrt(E[(obs-model)^2]) = sqrt(sum((obs - model)^2) / n)

     \*/

    double sum\_ObsAndModel = 0;

    for (int i = DATATRAIN + 1; i < DATAMAXN; i++)

    {

        prediction[i] = W[0] \* 1;

        for (int j = 0; j < N; j++)

        {

            prediction[i] += W[dimension[j] + 1] \* dataAll[i][dimension[j]];

        }

        sum\_ObsAndModel += pow((dataAll[i][MEDV] - prediction[i]), 2);

    }

    double rmse = sqrt(sum\_ObsAndModel / (DATAMAXN - DATATRAIN - 1)); // 分母确保方差估计量的无偏性

    printf("\nRMSE: %.3lf", rmse);

}

## 4、运行截图





