****

**《程序设计课程实践》设计文档**

# 第5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

构建结构体struct iris，其内部成员为double类型的数组（用来存放四个维度数据），int类型的变量（用来存放花名编号）；

然后依次需要进行的步骤有：

1. 数据读入：通过GetData函数读入文本数据，并将花名按照要求转化为对应的数字编号，存储在结构体数组Data中
2. 数据预处理：调用Preprocess函数
   1. 对读入的数据进行前后次序的随机打乱（通过调用函数shuffle()）
   2. 对前四个维度进行min-max标准化（通过调用函数normalizate()）
   3. 切分数据为训练数据集X和测试数据集为Y，比例大致为3：2（通过调用函数divideXY()），且保持Data数组不变
   4. 将测试数据集上的花名编号初始化为RAW值（通过调用initY()）
3. 实现KNN分类算法
   1. 遍历测试数据集
   2. 对于测试数据Y[i]，通过调用getKMinDistance函数找到训练数据集中离Y[i]最近的K个数据的下标，存储着closeIndex数组中（此处K取5）
   3. 通过调用getMostSpecies函数，得到closeIndex数组中出现次数最多的花名编号，并赋值给Y[i].species
4. 计算准确度：
   1. 遍历测试数据集，将其species与Data数组中对应位置的species进行比较，统计正确预测个数
   2. 利用(double)count / (DATAMAXN - DATATRAIN)计算准确度
5. 为减少偶然性，执行EPCHO次程序，求得平均准确度并输出

## 3、程序源码

#ifndef IRIS\_H

#define IRIS\_H

#include <stdbool.h>

#define DATAMAXN 150

#define DATATRAIN (int)(DATAMAXN \* 3 / 5)

#define EPCHO 10000

#define K 5

enum property

{

    sepalLength,

    sepalWidth,

    petalLength,

    petalWidth,

    dimensionN,

};

/\* 函数定义 \*/

bool GetData(void);

void Preprocess(void);

void KNN(void);

double GetAccuracy(void);

#endif

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <error.h>

#include <math.h>

#include <time.h>

#include "iris.h"

#define NSPECIES 3

#define RAW -1

typedef char \*string;

struct iris

{

    double data[dimensionN];

    int species;

} Data[DATAMAXN], X[DATATRAIN], Y[DATAMAXN - DATATRAIN];

double MaxValue[dimensionN], MinValue[dimensionN];

string speciesname[NSPECIES] = {

    "Iris-setosa",

    "Iris-versicolor",

    "Iris-virginica",

};

/\* 局部函数定义 \*/

void shuffle(void);

void normalizate(void);

void divideXY(void);

void initY(void);

void getKMinDistance(int indexY, int \*closeIndex);

int getMostSpecies(int \*closeIndex);

/\* 测试函数 \*/

/\* 准确率不高，最高才到达0.7，正常KNN应该在0.9 \*/

int main(void)

{

    if (!GetData())

    {

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

        system("pause");

        exit(errno);

    }

    double accuracy = 0;

    for (int i = 0; i < EPCHO; i++)

    {

        Preprocess();

        KNN();

        accuracy += GetAccuracy();

    }

    printf("\n\nTest %d times -> Average accuracy: %.2lf%%", EPCHO, 100 \* accuracy / EPCHO);

    system("pause");

    return 0;

}

/\* 接口函数实现 \*/

bool GetData(void)

{

    char irisname[20];

    FILE \*fp = fopen("data.txt", "rb");

    if (fp == NULL)

    {

        fputs("Open error", stderr);

        return false;

    }

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

    {

        fscanf(fp, "%lf,%lf,%lf,%lf,%s", &Data[i].data[sepalLength], &Data[i].data[sepalWidth],

               &Data[i].data[petalLength], &Data[i].data[petalWidth], irisname);

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

        {

            if (strcmp(irisname, speciesname[j]) == 0)

            {

                Data[i].species = j + 1;

                break;

            }

        }

    }

    fclose(fp);

    fp = NULL;

    return true;

}

void Preprocess(void)

{

    shuffle();

    normalizate();

    divideXY();

    initY();

}

void KNN(void)

{

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

    {

        int closeIndex[K];

        getKMinDistance(i, closeIndex);

        Y[i].species = getMostSpecies(closeIndex);

    }

}

double GetAccuracy(void)

{

    int count = 0;

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

    {

        if (Y[i].species == Data[i + DATATRAIN].species)

        {

            count++;

        }

    }

    return (double)count / (DATAMAXN - DATATRAIN);

}

/\* 局部函数实现 \*/

void shuffle(void)

{

    srand(time(0));

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

    {

        int index1 = rand() % DATAMAXN;

        int index2 = rand() % DATAMAXN;

        while (index2 == index1)

        {

            index2 = rand() % DATAMAXN;

        }

        struct iris temp = Data[index1];

        Data[index1] = Data[index2];

        Data[index2] = temp;

    }

}

void normalizate(void) // min-max normalizate

{

    /\*

        Search MaxValue & MinValue of train data in every dimension

        And test will still use them

        Last one is {1,2,3}, which seems not to need noramalization

    \*/

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

    {

        MaxValue[dimension] = Data[0].data[dimension];

        MinValue[dimension] = Data[0].data[dimension];

    }

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

    {

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

        {

            if (Data[i].data[dimension] > MaxValue[dimension])

            {

                MaxValue[dimension] = Data[i].data[dimension];

            }

            if (Data[i].data[dimension] < MinValue[dimension])

            {

                MinValue[dimension] = Data[i].data[dimension];

            }

        }

    }

    /\* min-max normalization \*/

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

    {

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

        {

            Data[i].data[dimension] =

                (Data[i].data[dimension] - MinValue[dimension]) / (MaxValue[dimension] - MinValue[dimension]);

        }

    }

}

void divideXY(void)

{

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

    {

        if (i < DATATRAIN)

        {

            X[i] = Data[i];

        }

        else

        {

            Y[i - DATATRAIN] = Data[i];

        }

    }

}

void initY(void)

{

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

    {

        Y[i].species = RAW;

    }

}

void getKMinDistance(int indexY, int \*closeIndex)

{

    double distance[K];

    for (int k = 0; k < K; k++)

    {

        double minDistance = 10000;

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

        {

            double sumDistance = 0;

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

            {

                sumDistance += pow(Y[indexY].data[dimension] - X[i].data[dimension], 2);

            }

            // check one has been

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

            {

                if (sumDistance - distance[check] < 0.00000001 &&

                    sumDistance - distance[check] > -0.00000001)

                {

                    sumDistance = 10000;

                    break;

                }

            }

            // when it's Ok, change minDistance and record i as index of array X

            if (sumDistance < minDistance)

            {

                minDistance = sumDistance;

                closeIndex[k] = i;

            }

        }

        distance[k] = minDistance;

    }

}

int getMostSpecies(int \*closeIndex)

{

    int count[NSPECIES + 1]; // count[0]不用

    memset(count, 0, sizeof(count));

    for (int k = 0; k < K; k++)

    {

        count[X[closeIndex[k]].species]++;

    }

    int mostIndex = 0;

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

    {

        if (count[i] > count[mostIndex])

        {

            mostIndex = i;

        }

    }

    return mostIndex;

}

## 4、运行截图

