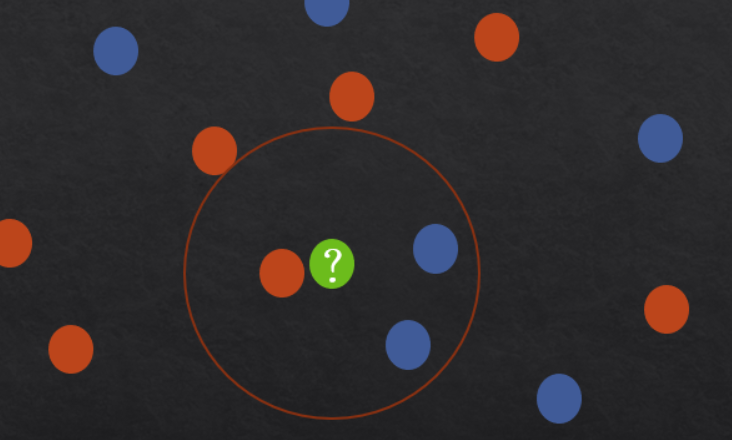
**21374389 牛鹏军 210711**

**理论分析**

|  |
| --- |
| 用C语言基于KNN算法实现Iris鸢尾花分类方法。要求用训练集  构建模型，用测试集进行验证，得出分类错误率，即  误分类实例数/测试实例总数的比率。  输入：IrisTrain.txt,IrisTest.txt  输出：ErrorRatio |

用KNN进行鸢尾花分类训练，令K=3，即将测试集中某组数据与训练集上的各组数据求欧氏距离，将测试集中的每组数据与训练集中各组数据的距离数据排序后，取出最小的三个距离对应的训练集的分类状况，取这三组中最多出现的标签，即为对测试集该组数据的预测情况。



**算法设计**

|  |
| --- |
|  |

**编程实现**

|  |
| --- |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <math.h>  #include <time.h>  #define TEST\_SIZE 26  #define TRAIN\_SIZE 124  #define FEATURE\_COUNT 4  #define K\_NEIGHBORS 3  typedef struct {  double features[FEATURE\_COUNT]; // 每种花的4个特征数据  char species[20]; // 存放花的种类  int label; // 用于设置标签 为了方便检测  } Iris;  typedef struct {  double value; // 距离数据  int label; // 用于绑定训练集标签  } Distance;  Iris testSet[TEST\_SIZE];  Iris forecastSet[TEST\_SIZE];  Iris trainSet[TRAIN\_SIZE];  Distance distances[TRAIN\_SIZE];  // 将花的种类转换为标签  void labelSpecies(char \*type, int \*label);  // 将数组随机重排  void shuffle(Iris iris[], int n);  // 从文件中加载数据集  void loadData(const char \*trainPath, const char \*testPath);  // 计算两个花之间的欧氏距离  double calculateDistance(const Iris \*iris1, const Iris \*iris2);  // 用于比较两个 Distance 结构的函数，用于排序  int compareDistances(const void \*d1, const void \*d2);  // 统计前 K 个最近邻居中出现次数最多的标签  int countMostFrequentLabel(Distance \*distances, int k);  // 函数用于返回指定标签的字符串表示  const char\* getSpeciesLabel(int label);  // 打印比较结果，包括原始标签、预测标签和准确率  void printResults(int k, int count);  int main() {  srand((unsigned int)time(NULL));  // 加载数据  loadData("IrisTrain.txt", "IrisTest.txt");  int count = 0;  // 对每个测试样本进行预测  for (int i = 0; i < TEST\_SIZE; i++) {  // 计算测试样本与训练集中每个样本的距离  for (int j = 0; j < TRAIN\_SIZE; j++) {  distances[j].value = calculateDistance(&testSet[i], &trainSet[j]);  distances[j].label = trainSet[j].label;  }  // 对距离进行排序  qsort(distances, TRAIN\_SIZE, sizeof(Distance), compareDistances);  // 统计最近的 K 个邻居中出现次数最多的标签  forecastSet[i].label = countMostFrequentLabel(distances, K\_NEIGHBORS);  // 检查预测结果是否正确  if (forecastSet[i].label == testSet[i].label) {  count++;  }  }  // 打印比较结果  printResults(K\_NEIGHBORS, count);  return 0;  }  void labelSpecies(char \*type, int \*label) {  if (strcmp(type, "Iris-setosa") == 0) \*label = 0;  else if (strcmp(type, "Iris-versicolor") == 0) \*label = 1;  else if (strcmp(type, "Iris-virginica") == 0) \*label = 2;  }  void shuffle(Iris iris[], int n) {  int i;  for (i = n - 1; i > 0; i--) {  int j = rand() % (i + 1);  Iris temp = iris[i];  iris[i] = iris[j];  iris[j] = temp;  }  }  void loadData(const char \*trainPath, const char \*testPath) {  FILE \*fpTrain = fopen(trainPath, "r");  FILE \*fpTest = fopen(testPath, "r");  char species[20];  double features[FEATURE\_COUNT];  int i, j;  if (!fpTrain || !fpTest) {  fprintf(stderr, "Error opening files.\n");  exit(1);  }  for (i = 0; i < TRAIN\_SIZE; i++) {  fscanf(fpTrain, "%lf,%lf,%lf,%lf,%s", &features[0], &features[1], &features[2], &features[3], species);  for (j = 0; j < FEATURE\_COUNT; j++) {  trainSet[i].features[j] = features[j];  }  labelSpecies(species, &trainSet[i].label);  }  for (i = 0; i < TEST\_SIZE; i++) {  fscanf(fpTest, "%lf,%lf,%lf,%lf,%s", &features[0], &features[1], &features[2], &features[3], species);  for (j = 0; j < FEATURE\_COUNT; j++) {  testSet[i].features[j] = features[j];  }  labelSpecies(species, &testSet[i].label);  }  fclose(fpTrain);  fclose(fpTest);  }  double calculateDistance(const Iris \*iris1, const Iris \*iris2) {  double sum = 0.0;  for (int i = 0; i < FEATURE\_COUNT; i++) {  sum += (iris1->features[i] - iris2->features[i]) \* (iris1->features[i] - iris2->features[i]);  }  return sqrt(sum);  }  int compareDistances(const void \*d1, const void \*d2) {  const Distance \*distance1 = (const Distance \*)d1;  const Distance \*distance2 = (const Distance \*)d2;  if (distance1->value > distance2->value) {  return 1;  } else if (distance1->value < distance2->value) {  return -1;  } else {  return 0;  }  }  int countMostFrequentLabel(Distance \*distances, int k) {  int labelCount[3] = {0};  for (int i = 0; i < k; i++) {  labelCount[distances[i].label]++;  }  int maxCount = labelCount[0];  int label = 0;  for (int i = 1; i < 3; i++) {  if (labelCount[i] > maxCount) {  maxCount = labelCount[i];  label = i;  }  }  return label;  }  const char\* getSpeciesLabel(int label) {  switch(label) {  case 0:  return "Iris-setosa";  case 1:  return "Iris-versicolor";  case 2:  return "Iris-virginica";  default:  return "Unknown";  }  }  void printResults(int k, int count) {  printf("Comparison Results for K = %d:\n", k);  for (int i = 0; i < TEST\_SIZE; i++) {  const char\* predictedSpecies = getSpeciesLabel(forecastSet[i].label);  const char\* trueSpecies = getSpeciesLabel(testSet[i].label);  const char\* correctness = (forecastSet[i].label == testSet[i].label) ? "Correct" : "Incorrect";  printf("%-20s%-20s%-10s\n", predictedSpecies, trueSpecies, correctness);  }  double accuracy = ((double)count / TEST\_SIZE) \* 100.0;  printf("Accuracy: %.2f%%\n", accuracy);  printf("ErrorRatio: %.2f%%\n", 100-accuracy);  printf("Correctly Classified Instances: %d out of %d\n", count, TEST\_SIZE);  } |

**测试分析**

|  |
| --- |
|  |

将两txt文件放在源代码同一目录下，训练集共有124组数据，测试集共有26组数据，最终测试集中25组成功预测，成功率为96.15%，失败率为3.85%，仅在第21组数据预测失败，成功率还是很可以的。

**结论**

KNN是一个经典的监督学习分类算法，在对简单数据分类时比较适用。