****

**《程序设计课程实践》设计文档**

# 第5章 第4次作业

**学号：22050924 22050938**

**姓名：许钟乐 傅来**

完成时间 2023 年 5 月

## 1、题目

鸢尾花分类

## 2、程序设计思路

本程序使用了KNN（K-最近邻）算法，使用了泛型编程的思想，实现了一个简易的使用欧几里得距离为权值的，KD-Tree为底层数据结构的分类器。实际使用时总数据范围不应大于1e8，输入数据维度不应多于1e3。使用KD-Tree与二叉堆优化查询，使得当且当以目标点为圆心，当前最大距离为半径的超球与分割点所分割的另一个超矩形存在交点或当前遍历点小于K个时才进行另一颗子树的查询，减少了查询的点的数量，实现了时间复杂度的优化。

## 3、程序源码

#include <bits/stdc++.h>  
#include "kdt.h"  
using namespace std;  
const int N = 4;  
using *dataset* = vector<pair<Point<N>, *string*>>;  
void readdata(*ifstream* &*fin*, *dataset* &*wow*)  
{  
 double s;  
 char tmp;  
 while (*fin* **>>** s **>>** tmp)  
 {  
 *string* id;  
 vector<double> ar;  
 ar.emplace\_back(s);  
 for (int i = 1; i < N; i++)  
 {  
 *fin* **>>** s **>>** tmp;  
 ar.emplace\_back(s);  
 }  
 *fin* **>>** id;  
 *wow*.emplace\_back(ar, id);  
 }  
}  
int main()  
{  
 *ifstream* fin1,fin2;  
 fin1.open("D:\\data.txt", *ios*::in);  
 if (!fin1.is\_open())  
 {  
 cout **<<** "data not found\n";  
 return 1;  
 }  
 *dataset* tmp;  
 readdata(fin1, tmp);  
 shuffle(tmp.begin(),tmp.end(), std::mt19937(std::random\_device()**()**));  
 *dataset* train(tmp.begin(),tmp.begin() **+** 74),query(tmp.begin()**+**75,tmp.end());  
 KDTree<N, *string*> kd(train);  
 cout **<<** "build complete\n";  
 for (int i = 1; i <= 15; i+= 2)  
 {  
 int res = 0;  
 for (auto &[poi, name] : query)  
 res += kd.KNNvalue(poi, i) **==** name;  
 cout **<<** "K = " **<<** i **<<** " " **<<** res \* 1.0 / query.size() \* 100 **<<** "%\n";  
 }  
 return 0;  
}

*//  
// Created by xzl on 2023/4/27.  
//*#ifndef **KNN\_\_KDT\_H\_**#define **KNN\_\_KDT\_H\_**#include <bits/stdc++.h>  
#include "Point.h"  
using std::*size\_t*,std::vector,std::pair;  
template<*size\_t* N,typename *T*>  
class KDTree  
{  
 private:  
 struct Node  
 {  
 Point<N> poi;  
 Node\* lson,\*rson;  
 int depth;  
 *T* val;  
 Node(const Point<N>& *poi*,int *dep*,*T w*):  
 poi(*poi*),lson(nullptr), rson(nullptr),depth(*dep*) ,val(*w*) {}  
 };  
 Node\* root;  
 *size\_t* sz,maxsz;  
 Node\* build(typename vector<pair<Point<N>,*T*>>::iterator *l*,  
 typename vector<pair<Point<N>,*T*>>::iterator *r*,int *dep*);  
 Node\* findNode(Node\* *cur*, const Point<N>& *poi*) const;  
 void NearestNeighbourdfs(const Node\* *cur*, const Point<N>& *key*, std::priority\_queue<pair<double,*T*>>& *q*) const;  
 Node\* deepcopyTree(Node\* *rt*);  
 void freeResource(Node\* *cur*);  
 public:  
 KDTree();  
 explicit KDTree(vector<pair<Point<N>,*T*>>& *poi*);  
 ~KDTree();  
 KDTree(const KDTree& *rhs*);  
 KDTree& operator**=**(const KDTree& *rhs*);  
 *size\_t* dimension() const;  
 *size\_t* size() const;  
 bool empty() const;  
 bool contain(const Point<N>& *poi*) const;  
 *T*& operator**[]**(const Point<N>& *poi*);  
 *T* KNNvalue(const Point<N>& *key*,*size\_t k*);  
 void insert(const Point<N>& *poi*,const *T*& *w*);  
};  
template<*size\_t* N, typename *T*>  
*T* KDTree<N, *T*>::KNNvalue(const Point<N> &*key*, *size\_t k*)  
{  
 std::priority\_queue<pair<double,*T*>> q;  
 maxsz = *k*;  
 if (empty())  
 return *T*();  
 NearestNeighbourdfs(root,*key*,q);  
 std::unordered\_map<*T*,int> cnt;  
 while(!q.empty())  
 {  
 cnt[q.top().second]++;  
 q.pop();  
 }  
 *T* res;  
 int max = 0;  
 for(const auto &it : cnt)  
 {  
 if (it.second > max)  
 {  
 res = it.first;  
 max = it.second;  
 }  
 }  
 return res;  
}  
template<*size\_t* N, typename *T*>  
void KDTree<N, *T*>::NearestNeighbourdfs(const KDTree::Node \**cur*, const Point<N> &*key*, std::priority\_queue<pair<double,*T*>> &*q*) const  
{  
 if (*cur* == nullptr)  
 return;  
 const Point<N>& curpoi = *cur*->poi;  
 *q*.push({dis(curpoi, *key*), *cur*->val});  
 if (*q*.size() > maxsz)  
 *q*.pop();  
 int dep = *cur*->depth;  
 bool left;  
 if (*key*[dep % N] < curpoi[dep % N])  
 {  
 NearestNeighbourdfs(*cur*->lson,*key*,*q*);  
 left = true;  
 }  
 else  
 {  
 NearestNeighbourdfs(*cur*->rson,*key*,*q*);  
 left = false;  
 }  
 if (*q*.size() < maxsz || fabs(*key*[dep % N] - curpoi[dep % N]) < *q*.top().first)  
 {  
 if (left)  
 NearestNeighbourdfs(*cur*->rson,*key*,*q*);  
 else  
 NearestNeighbourdfs(*cur*->lson,*key*,*q*);  
 }  
  
}  
template<*size\_t* N, typename *T*>  
void KDTree<N, *T*>::insert(const Point<N> &*poi*, const *T* &*w*)  
{  
 auto node = findNode(root,*poi*);  
 if (node == nullptr)  
 {  
 root = new Node(*poi*,0,*w*);  
 sz = 1;  
 return;  
 }  
 if (node->val == *w*)  
 {  
 node->val = *w*;  
 return;  
 }  
 int dep = node->depth;  
 auto Newnode = new Node(*poi*,dep + 1,*w*);  
 if (*poi*[dep % N] < node->poi[dep % N])  
 node->lson = Newnode;  
 else  
 node->rson = Newnode;  
 sz++;  
}  
template<*size\_t* N, typename *T*>  
*T* &KDTree<N, *T*>::operator**[]**(const Point<N> &*poi*)  
{  
 auto node = findNode(root,*poi*);  
 if (node != nullptr && node->poi == *poi*)  
 return node->val;  
 else  
 {  
 insert(*poi*,*T*());  
 node = findNode(node,*poi*);  
 return node->val;  
  
 }  
}  
template<*size\_t* N, typename *T*>  
typename KDTree<N,*T*>::Node \*KDTree<N, *T*>::findNode(KDTree::Node \**cur*, const Point<N> &*poi*) const  
{  
 if (*cur* == nullptr || *cur*->poi == *poi*)  
 return *cur*;  
 const Point<N>& curpoi = *cur*->poi;  
 int curdep = *cur*->depth;  
 if (*poi*[curdep % N] < curpoi[curdep % N])  
 return *cur*->lson == nullptr ? *cur* : findNode(*cur*->lson,*poi*);  
 else  
 return *cur*->rson == nullptr ? *cur* : findNode(*cur*->rson,*poi*);  
}  
template<*size\_t* N, typename *T*>  
bool KDTree<N, *T*>::contain(const Point<N> &*poi*) const  
{  
 auto node = findNode(root,*poi*);  
 return node != nullptr && node->poi == *poi*;  
}  
template<*size\_t* N, typename *T*>  
bool KDTree<N, *T*>::empty() const  
{  
 return sz == 0;  
}  
template<*size\_t* N, typename *T*>  
*size\_t* KDTree<N, *T*>::size() const  
{  
 return sz;  
}  
template<*size\_t* N, typename *T*>  
*size\_t* KDTree<N, *T*>::dimension() const  
{  
 return N;  
}  
template<*size\_t* N, typename *T*>  
KDTree<N, *T*>::KDTree():  
sz(0),root(nullptr) {}  
template<*size\_t* N, typename *T*>  
typename KDTree<N,*T*>::Node \*KDTree<N, *T*>::deepcopyTree(KDTree::Node \**rt*)  
{  
 if (*rt* == nullptr)  
 return nullptr;  
 auto nroot = new Node(\**rt*);  
 nroot->lson = deepcopyTree(*rt*->lson);  
 nroot->rson = deepcopyTree(*rt*->rson);  
 return nroot;  
}  
template<*size\_t* N, typename *T*>  
void KDTree<N, *T*>::freeResource(Node \**cur*)  
{  
 if (*cur* == nullptr)  
 return ;  
 freeResource(*cur*->lson);  
 freeResource(*cur*->rson);  
 delete *cur*;  
}  
template<*size\_t* N, typename *T*>  
typename KDTree<N,*T*>::Node \*KDTree<N, *T*>::build(typename vector<pair<Point<N>, *T*>>::iterator *l*,  
 typename vector<pair<Point<N>, *T*>>::iterator *r*,int *dep*)  
{  
 if (*l* >= *r*)  
 return nullptr;  
 auto axis = *dep* % N;  
 auto cmp = [&](const pair<Point<N>,*T*> &*lhs*,const pair<Point<N>,*T*>& *rhs* )  
 { return *lhs*.first[axis] < *rhs*.first[axis];};  
 auto mid = *l* + (*r* - *l*) / 2;  
 std::nth\_element(*l*,mid,*r*,cmp);  
 while(mid > *l* && mid->first[axis] == (mid - 1)->first[axis])  
 mid--;  
 auto Newnode = new Node(mid->first,*dep*,mid->second);  
 Newnode->lson = build(*l*,mid,*dep* + 1);  
 Newnode->rson = build(mid + 1,*r*,*dep* + 1);  
 return Newnode;  
}  
template<*size\_t* N, typename *T*>  
KDTree<N, *T*>::KDTree(vector<pair<Point<N>, *T*>> &*poi*)  
{  
 root = build(*poi*.begin(),*poi*.end(),0);  
 sz = *poi*.size();  
}  
template<*size\_t* N, typename *T*>  
KDTree<N, *T*>::KDTree(const KDTree& *rhs*)  
{  
 root = deepcopyTree(*rhs*);  
 sz = *rhs*.sz;  
}  
template<*size\_t* N, typename *T*>  
KDTree<N,*T*> &KDTree<N, *T*>::operator**=**(const KDTree &*rhs*)  
{  
 if (\**this* == *rhs*)  
 return \**this*;  
 freeResource(root);  
 root = deepcopyTree(*rhs*.root);  
 sz = *rhs*.sz;  
 return \**this*;  
}  
template<*size\_t* N,typename *T*>  
KDTree<N,*T*>::~KDTree()  
{  
 freeResource(root);  
}  
  
#endif *//KNN\_\_KDT\_H\_*

#ifndef **KNN\_POINT\_H\_**#define **KNN\_POINT\_H\_**#include <bits/stdc++.h>  
using std::*size\_t*;  
template<*size\_t* N>  
class Point  
{  
 private:  
 double data[N];  
 public:  
 typedef double\* *iterator*;  
 typedef const double\* *const\_iterator*;  
 *size\_t* size() const;  
 double& operator**[]**(*size\_t i*);  
 double operator**[]**(*size\_t i*) const;  
 *iterator* begin();  
 *iterator* end();  
 *const\_iterator* begin() const;  
 *const\_iterator* end() const;  
 Point(){};  
 Point(const std::vector<double>& *ar*);  
};  
template<*size\_t* N>  
Point<N>::Point(const std::vector<double> &*ar*)  
{  
 for(int i=0;i<N;i++)  
 data[i] = *ar***[**i**]**;  
}  
template<*size\_t* N>  
double dis(const Point<N>&*lhs*,const Point<N>& *rhs*);  
template <*size\_t* N>  
bool operator**==**(const Point<N>& *lhs*, const Point<N>& *rhs*);  
template <*size\_t* N>  
bool operator**!=**(const Point<N>& *lhs*, const Point<N>& *rhs*);  
  
  
template<*size\_t* N>  
double dis(const Point<N>&*lhs*,const Point<N>& *rhs*)  
{  
 double res = 0;  
 for(auto i =0;i<N;i++)  
 res += (*lhs*[i] - *rhs*[i]) \* (*lhs*[i] - *rhs*[i]);  
 return sqrt(res);  
}  
  
template <std::*size\_t* N>  
bool operator**==**(const Point<N>& *lhs*, const Point<N>& *rhs*)  
{  
 return std::equal(*lhs*.begin(), *lhs*.end(), *rhs*.begin());  
}  
  
template <std::*size\_t* N>  
bool operator**!=**(const Point<N>& *lhs*, const Point<N>& *rhs*)  
{  
 return !(*lhs* == *rhs*);  
}  
template<*size\_t* N>  
typename Point<N>::*const\_iterator* Point<N>::end() const  
{  
 return data + size();  
}  
template<*size\_t* N>  
typename Point<N>::*iterator* Point<N>::end()  
{  
 return data + size();  
}  
template<*size\_t* N>  
typename Point<N>::*const\_iterator* Point<N>::begin() const  
{  
 return data;  
}  
template<*size\_t* N>  
typename Point<N>::*iterator* Point<N>::begin()  
{  
 return data;  
}  
template<*size\_t* N>  
double Point<N>::operator**[]**(*size\_t i*) const  
{  
 return data[*i*];  
}  
template<*size\_t* N>  
double &Point<N>::operator**[]**(*size\_t i*)  
{  
 return data[*i*];  
}  
template<*size\_t* N>  
*size\_t* Point<N>::size() const  
{  
 return N;  
}  
  
#endif *//KNN\_\_POINT\_H\_*

## 4、运行截图





