Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект по курсу “Дискретный анализ”

Студент: В. А. Петросян
Преподаватель: А. А. Журавлёв
Группа: М8О-208Б-17
Дата:
Оценка:
Подпись:

Москва
2019

# Курсовой проект по теме: “Classification and regression tree ”

Автоматическое предсказывание значения точки из пространства $R^m$ в  0 , 1

Запуск и параметры:

```
./prog learn    --input file1   --output file2  --height h
./prog classify --input file2   --output file3  --data file4
```

Пример использования в автоматическом режиме:

```
chmod +x ./work.sh
./work.sh
```

Пример использования в ручном режиме:

```
g++ TestGenerator.cpp
./a.out
make
./main learn dataWithValue tree 10
./main classify tree answer data
```

# Описание

## Постановка задачи

Задача состоит в коректном разделении пространства. Программа сначала обучается на каком-то наборе m-менрых векторов с известными значениями в этих точках. После программа снова запускается, но уже в режиме классификации , то есть при поступлении m-мерного вектора чисел программой будет сделано предположение о возможном значение в этой точке пространства. Ограничение на точность накладывает h максимальная высота дерева разбиения, которая не позволяет строить дерево разбиения глубже.

## Представление данных

```
class DataRepresent{

public:

    std::vector<double> space;
    bool value;
    DataRepresent(): value(true) , space(m){};

    DataRepresent &operator=(DataRepresent &ob){
        this->space = ob.space;
        this->value = ob.value;
        return * this;
    }
    friend std::ostream &operator<<(std::ostream &stream, DataRepresent &ob){
        for(int k = 0; k < m; ++k){
             stream << ob.space[k] << "\t";
        }
        stream << "\t\t" << ob.value << "\n";
        return stream;
    }
    DataRepresent& operator=(const DataRepresent& ob){
        for(int i = 0; i < m; ++i){
            this->space[i] = ob.space[i];
        }
        this->value = ob.value;
        return *this;
    }

};
```

## Обработка данных

Программа считывает данные и записывает их в вектор nums, который содержит в себе структуры DataRepresent . Каждая такая структура содержит координаты точки в виде обычного целочисленного вектора и значение true или false.

Далее начинается поиск по всем "измерениям"наилучшего разбиения с помощью энтропии. Мы стремимся к минимальной энтропии т.е. к нулевой. В конце очередной фазы алгоритма происходит разбиение по какому-то конкретному измерению и задача решается рекурсивно уже для двух подмассивов, пока не потребуется очередное разби-

ение или пока не достигнет максимальной высоты в конкретной ветке дерева.

Что такое энтропия? Это термин, который первыми ввели физики в рамках термодинамики. Они ввели ее как функцию состояния термодинамической системы.

Кроме физики, термин широко употребляется в математике: теории информации и математической статистике. В этих областях энтропия определяется статистически и называется информационной энтропией. Данное определение энтропии известно также как энтропия Шеннона. Энтропия может интерпретироваться как мера неупорядоченности некоторой системы.

Математический смысл информационной энтропии — это логарифм числа доступных состояний системы (основание логарифма может быть различным, но большим 1, оно определяет единицу измерения энтропии). В качестве функции от числа состояний выбрали именно логарифм , потому что он обеспечивает свойство аддитивности.

Информационная двоичная энтропия для независимых случайных событий X с n возможными состояниями, распределённых с вероятностями $p_i (i = 1, ..., n)$,

$$H(x) = -\sum_{i=1}^{n} p_i \log_2 p_i.$$

Проще говоря, энтропия означает меру сложности, хаотичности или неопределённости системы: чем меньше элементы системы подчинены какому-либо порядку, тем выше энтропия.

В своей программе я смотрю на величину:

```
( leftEntropy*leftAmount + rightEntropy*rightAmount )/totalAmount;
```

, где leftEntropy и rightEntropy вычисляются по формулам

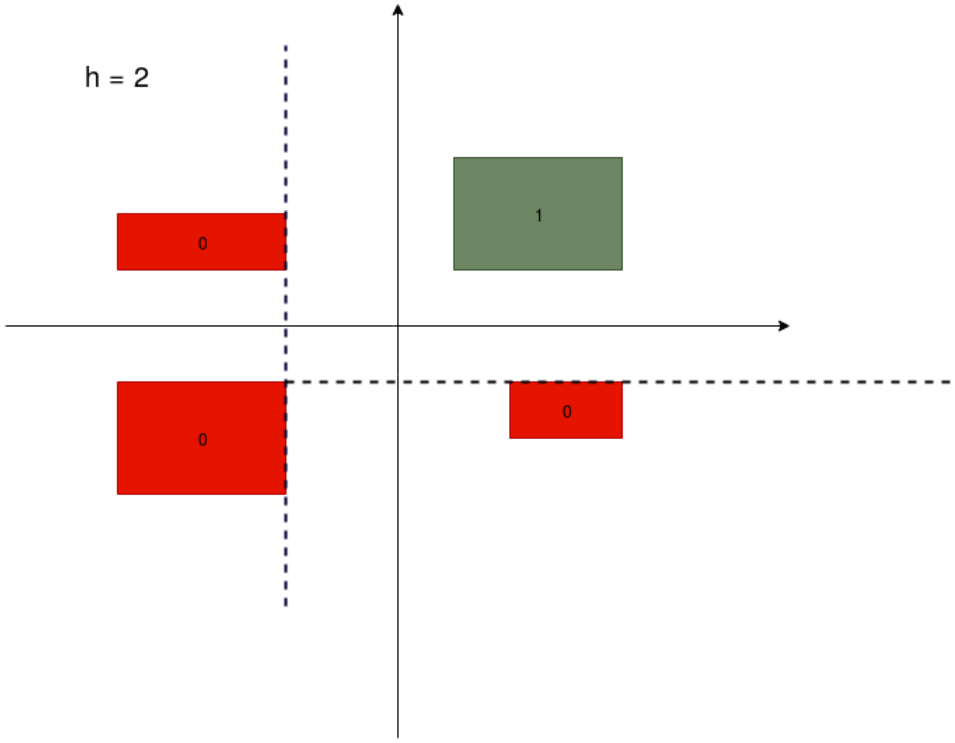
```
leftEntropy = Summand(leftZeroAmount,leftAmount) + Summand(leftUnitsAmount,leftAmount);
rightEntropy = Summand(rightZeroAmount,rightAmount) + Summand(rightUnitsAmount,rightAmou
```

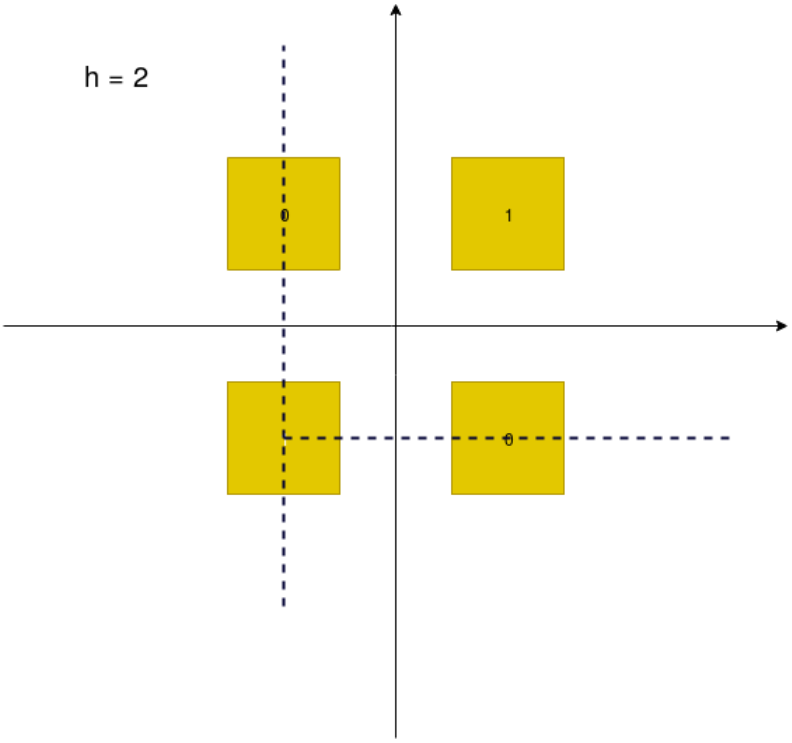
Стоит отметить, что есть другая техника разбиения пространства, которую я не решился реализовать. Конечно, я имею ввиду geany coefficient. Пытался в ней разобраться, искал в интернете ответы. Нашел пару статей, которые не до конца понял. В итоге наткнулся на видео из Google, где предлагалось решение похожей задачи именно через энтропию.

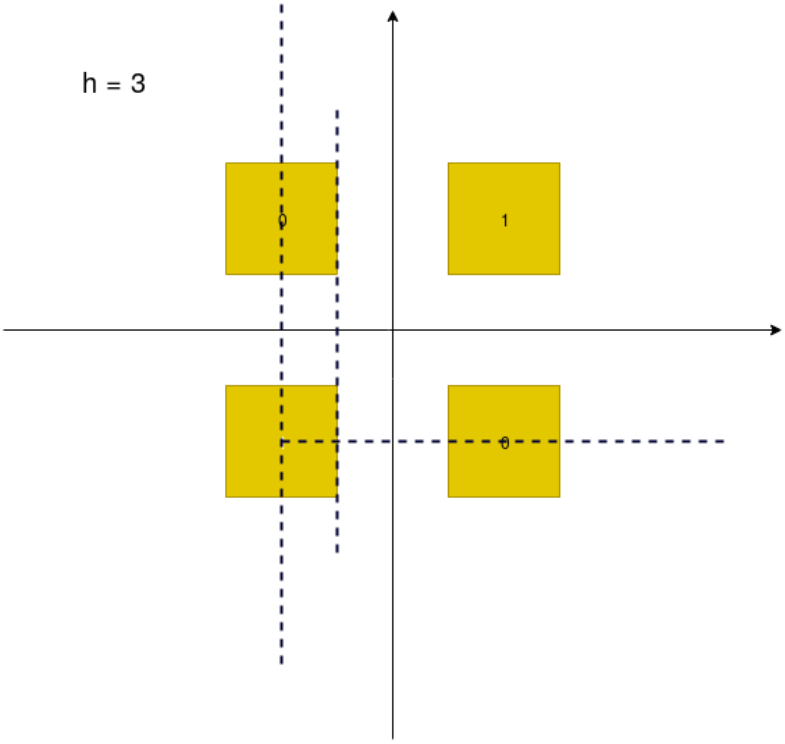
# Примеры классификации

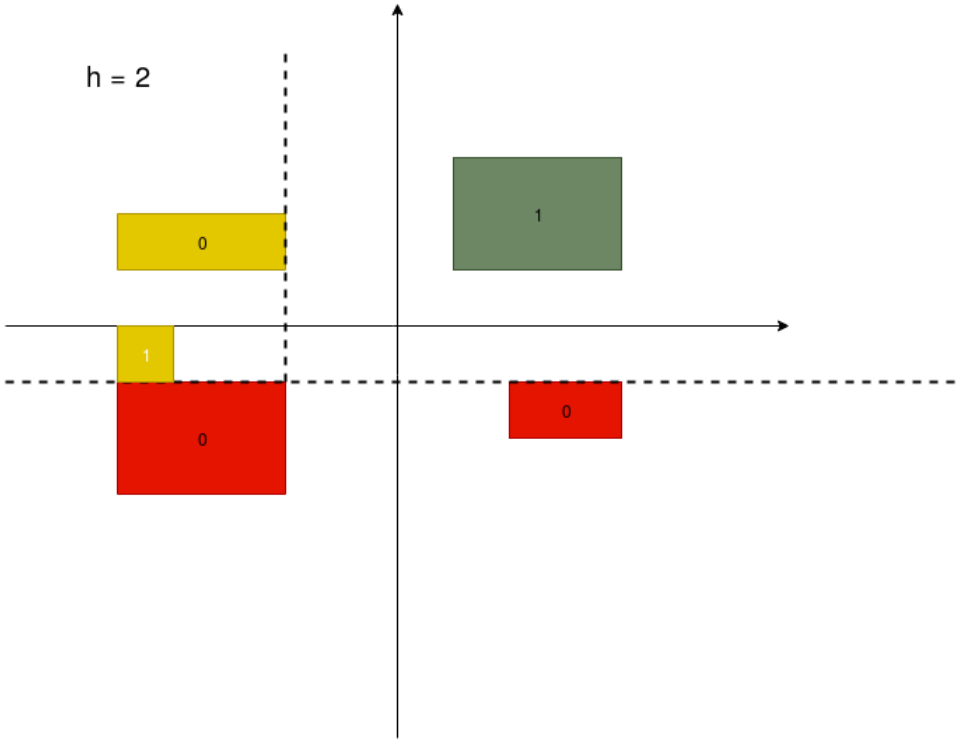
Алгоритм плохо справляется, когда точки идут очень плотно, а ограничение на высоту маленькое, хотя конечно все зависит от конкретного случая.

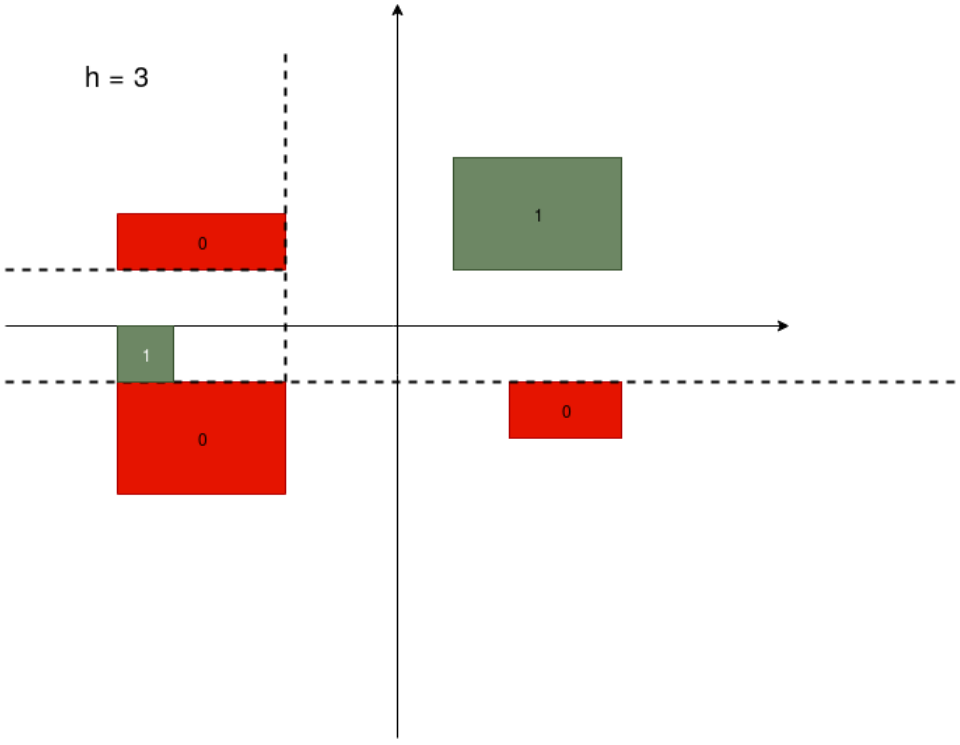
Зеленый цвет - область только с единицами.
Красный цвет - область только с нулями.
Желтый цвет - область в которой как единицы так и нули.

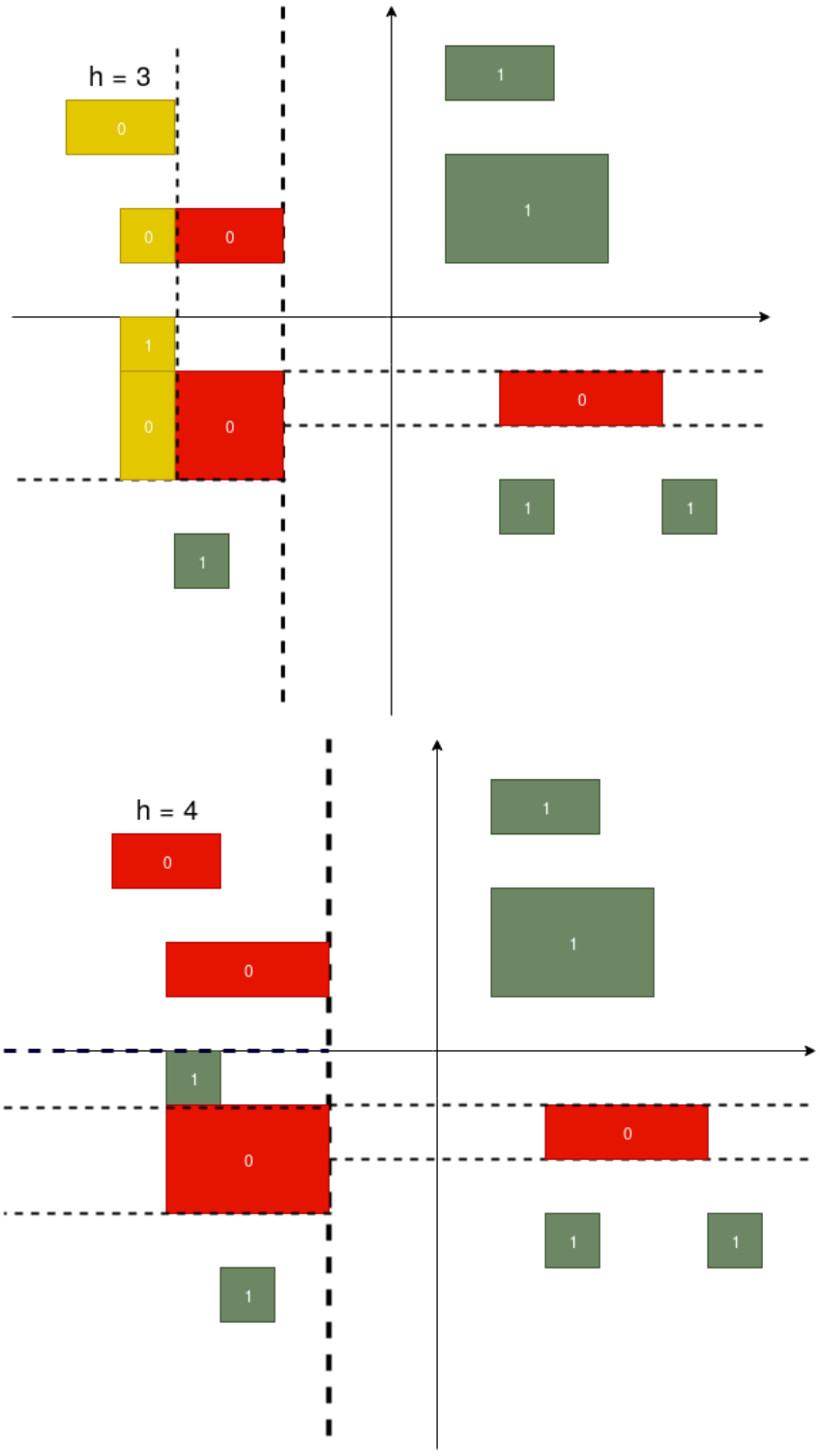


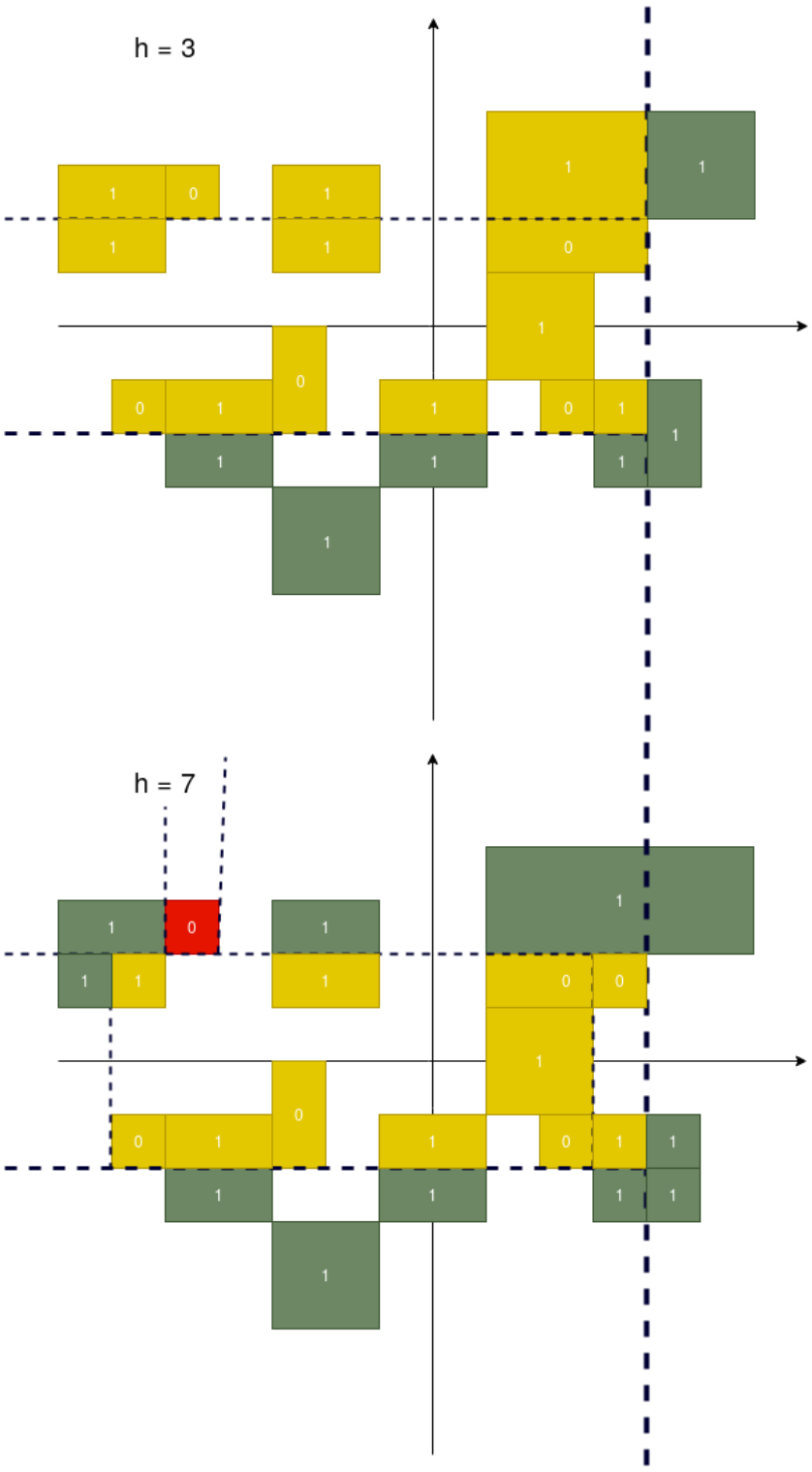

h = 2
0
1
0


h = 3
0
1
0


h = 2
0
1
1
0
0


h = 3
0
1
1
0
0


h = 3
0
0
0
1
0
0
1
1
1
1
h = 4
0
0
1
0
1
0
1
1


h = 3
h = 7

# Исходный код

```
// learn  --input file1  --output file2  --height int h
// classify --input file2 --output file3  --data file4
// Time complexity should be O( n*m*h*log(n) )

#include <iostream>
#include <vector>
#include <fstream>
#include <cstdlib>
#include <cmath>
#include <algorithm>
#include <map>
#include "DataRepresent.h"
#include "BinarSearchTree.h"

const int AMORTIZATION = 1000;

double Summand(int nk,int a){
    if( nk == 0){
        return 0;
    }
    double answer = (double)nk*(log2(nk) - log2(a))/a;
    return answer;
}

void DivByEntropy(double &minEntropy,const std::vector<DataRepresent> &nums,const std::v

    double current;
    double entropyValueNew = INFINITY;
    int zeros = 0;
    int units = 0;
    int leftZeroAmount = 0;
    int leftUnitsAmount = 0;

    std::map<int,int> itCount;
    int maximum = 0;

    for(int i = startPos; i <= endPos; ++i){
        if( nums[order[i]].value == false ){
            ++zeros;
        }
```

```
    else{
        ++units;
    }
    int cnt = ++itCount[ nums[order[i] ].space[dimension] ];
    if( cnt > maximum ){
        maximum = cnt;
    }
}
int totalAmount = units + zeros;
if( maximum == totalAmount){
    minEntropy = INFINITY;
    return;
}
// [startPos,endPos]
for(int i = startPos; i < endPos; ++i){

    current = nums[order[i]].space[dimension];
    int bnd = itCount.find(current)->second;
    for( int count = 0; count < bnd ; ++count){
        if( nums[order[i + count]].value == false){
            ++leftZeroAmount;
        }
        else{
            ++leftUnitsAmount;
        }
    }
    i = i + bnd - 1;
    if( i == endPos){
        return;
    }
    int rightUnitsAmount = units - leftUnitsAmount;//can be 0
    int rightZeroAmount = zeros - leftZeroAmount;//can be 0
    int leftAmount = leftZeroAmount + leftUnitsAmount;//never can be 0
    int rightAmount = rightZeroAmount + rightUnitsAmount;//never can be 0

    double leftEntropy;
    double rightEntropy;
    leftEntropy = Summand(leftZeroAmount,leftAmount) + Summand(leftUnitsAmount,leftA
    if( leftEntropy != 0 ){
        leftEntropy *= -1;
    }
    rightEntropy = Summand(rightZeroAmount,rightAmount) + Summand(rightUnitsAmount,r
```

```
        if( rightEntropy != 0 ){
            rightEntropy *= -1;
        }
        entropyValueNew = ( leftEntropy*leftAmount + rightEntropy*rightAmount )/totalAmo
        if ( entropyValueNew <= minEntropy ){
            minEntropy = entropyValueNew;
            index = i;
            splitLine = current;
            ifBiggerVal = AMORTIZATION * rightUnitsAmount / rightAmount;
            ifSmallerVal = AMORTIZATION * leftUnitsAmount / leftAmount;
        }
    }
    return;

}


void BuildCART(TTree* ptr,std::vector<DataRepresent> &nums,std::vector<int> &order,int s

    if ( startPos < endPos ){

        double minEntropy = INFINITY;
        int index = startPos;
        int dimension;
        double line = 10;
        int ifSmallerVal = 0;
        int ifBiggerVal = 0;

        double bestEntropy = INFINITY;
        int bestIndex = index;
        int bestDimension;
        double bestLine;
        int bestIfBiggerVal;
        int bestIfSmallerVal;

        for(int i = 0; i < m && ( bestEntropy != 0 ) ; ++i){

            dimension = i;
            sort(order.begin() + startPos,order.begin() + endPos,[&](int lhs,int rhs){ r
            DivByEntropy(minEntropy,nums,order,startPos,endPos,dimension,line,ifBiggerVa
            if( minEntropy < bestEntropy ){
                bestIndex = index;
```

```
            bestEntropy = minEntropy;
            bestDimension = dimension;
            bestLine = line;
            bestIfBiggerVal = ifBiggerVal;
            bestIfSmallerVal = ifSmallerVal;
        }

    }

    ptr->coordinate = bestDimension;
    ptr->position = bestLine;
    ptr->ifBiggerValue = bestIfBiggerVal;
    ptr->ifSmallerValue = bestIfSmallerVal;
    ptr->left = nullptr;
    ptr->right = nullptr;

    if( startPos < bestIndex && bestIndex < endPos ){
        ++hight;
        if( hight < hightBound){
            TTree * left = nullptr;
            TTree * right = nullptr;
            if( bestIfBiggerVal != 1000 && bestIfBiggerVal != 0){
                right = (TTree *)malloc(sizeof(TTree));
            }
            if( bestIfSmallerVal != 1000 && bestIfSmallerVal != 0){
                left = (TTree *)malloc(sizeof(TTree));
            }

            sort(order.begin() + startPos,order.begin() + endPos,[&](int lhs,int rhs
            ptr->left = left;
            ptr->right = right;
            int leftHight = hight;
            int rightHight = hight;
            if( bestIfBiggerVal != 1000 && bestIfBiggerVal != 0){
                BuildCART(right,nums,order,bestIndex + 1,endPos,rightHight);
            }
            if( bestIfSmallerVal != 1000 && bestIfSmallerVal != 0){
                BuildCART(left,nums,order,0,bestIndex,leftHight);
            }
        }
    }
```

```
    }
    else{
        free(ptr);
        ptr = nullptr;
        std::cout << "Error startPos >= endPos\n" << startPos << " " << endPos << "\n";
    }

    return;
}


int SearchValue(std::vector<double> &answer,TTree *node){

    TTree * tmp = node;
    if( tmp != nullptr){

        if( answer[tmp->coordinate] <= tmp->position ){
            if(tmp->left != nullptr){
                SearchValue(answer,tmp->left);
            }
            else{
                return tmp->ifSmallerValue;
            }
        }
        else{
            if( tmp->right != nullptr){
                SearchValue(answer,tmp->right);
            }
            else{
                return tmp->ifBiggerValue;
            }
        }

    }

}

void PrintTree(TTree *node){

    if (node != nullptr){

        PrintTree(node->left);
```

```
        std::cout << node->coordinate << "\t" << node->position << "\t" << node->ifBigge

        PrintTree(node->right);
    }

}

void SerializeTree(TTree *node,std::ofstream& DTfout){
if( DTfout.is_open() ){

if( node != nullptr ){

            if( node->left != nullptr){
                DTfout << 1 << " ";
                SerializeTree(node->left,DTfout);
            }
            else{
                DTfout << 0 << " ";
            }

            DTfout << *node << " ";

            if( node->right != nullptr){
                DTfout << 1 << " ";
                SerializeTree(node->right,DTfout);
            }
            else{
                DTfout << 0 << " ";
            }
            (void*) node;
            free( node );
        }

}
    else{
        std::cout << "Can't open file in SerializeTree function\n";
    }


}
```

```
TTree * DeserializeTree(std::ifstream& DTfin){

if( DTfin.is_open() ){

        int value;
        DTfin >> value;

        if ( value == 0){
            TTree * ptr = nullptr;
            return ptr;
        }
        if( value == 1){

            TTree * left = DeserializeTree(DTfin);

            TTree * ptr = (TTree *)malloc(sizeof(TTree));

            DTfin >> ptr->coordinate >> ptr->position >> ptr->ifSmallerValue >> ptr->ifB

            TTree * right = DeserializeTree(DTfin);

            ptr->left = left;
            ptr->right = right;
            return ptr;
        }

}
    else{
        std::cout << "Can't open file in DeserializeTree function\n";
    }

}


void CleanTree(TTree *node){

    if( node != nullptr ){

        if( node->left != nullptr){
            CleanTree(node->left);
        }
```

```
        if( node->right != nullptr){
            CleanTree(node->right);
        }

        (void*) node;
        free( node );
    }

}

std::ifstream fin;
std::ofstream fout;
std::ifstream fdata;

int main(int argc, char const * argv[]) {

    if ( argc <= 1 ){
        std::cout << "Inject in coommand line \n./program learn --input --output --hight
        std::cout << "or\n";
        std::cout << "./program classify --input --output --data\n";
        return 1;
    }

    std::string arg = argv[1];
    bool okIn = false;

    std::string firstInput;
    std::string firstOutput;
    int hight;

    std::string secondInput;
    std::string secondData;
    std::string secondOutput;

    if ( arg == "learn" && argc == 5 ){
        okIn = true;
        firstInput = argv[2];
        firstOutput = argv[3];
        hight = atoi(argv[4]);
        fin.open(firstInput);
        fout.open(firstOutput);
        hightBound = hight;
```

```
}
if( arg == "classify" && argc == 5 ){
    okIn = true;
    secondInput = argv[2];
    secondOutput = argv[3];
    secondData = argv[4];
    fin.open(secondInput);
    fout.open(secondOutput);
    fdata.open(secondData);
}

if ( okIn == false ){
    std::cout << "Inject in coommand line \n./program learn --input --output --hight
    std::cout << "or\n";
    std::cout << "./program classify --input --output --data\n";
    return 1;
}
if ( !fout ){
    std::cout << "Can't open file\n";
    return 1;
}
if ( !fin ){
    std::cout << "Can't open file\n";
    return 1;
}

if ( arg == "learn"){

    fin >> n >> m;
    std::vector<DataRepresent> nums(n);
    srand(2);
    std::vector<int> order(n);
    for(int i = 0; i < n; ++i){
        for(int k = 0; k < m; ++k){
            fin >> nums[i].space[k];
        }
        fin >> nums[i].value;
        order[i] = i;
    }
    TTree *myCart = (TTree *)malloc(sizeof(TTree));;
    hight = 0;
    BuildCART(myCart,nums,order,0,n - 1,hight);
```

```
        fout << 1 << " ";
        SerializeTree(myCart,fout);
    }

if( arg == "classify"){

        fdata >> n >> m;
        std::vector<DataRepresent> nums(n);
        for(int i = 0; i < n; ++i){
            for(int k = 0; k < m; ++k){
                fdata >> nums[i].space[k];
            }
            nums[i].value = 2;
        }

        TTree *ptr = nullptr;
        ptr = DeserializeTree(fin);
        if( ptr == nullptr){
            std::cout << "Problem with tree\n";
        }

        for(int i = 0; i < n; ++i){
            int chisl = SearchValue(nums[i].space,ptr);
            double ans = (double)chisl / AMORTIZATION;
            fout << ans << "\n";
        }

        CleanTree(ptr);

    }

    return 0;
}
```

# Выводы

Курсовой проект потребовал приложить не мало усилий и времени. Для написания работающего CART потребовалось изучить существенное количество дополнительных источников. Честно говоря, поставленная задача с самой первой секунды меня заинтересовала. В области машинного обучения на момент выполнения задания я совсем ничего не знал. Пока искал информацию для своей задачи в интеренете посмотрел десяток обучающих видео. После выполнения работы сложилась некая примерная картина, что такое ML и "с чем его едят".

Не стоит забывать про математическую часть моей задачи. С энтропией я познакомился на лекции по физики, когда мы изучали Термодинамику. Забавно, ведь тогда я не мог себе представить, что она мне пригодиться именно в программировании. Пойзже я вспомнил про Теорему Шенонна и картинка потихоньку сложилась.

Немного о задачах и трудностях, с которыми я столкнулся в ходе написания программы. В начале пришлось написать свою сортировку TimSort = (MergeSort + InsertionSort) , так как я не догадался использовать лямбда функцию. В дальнейшем ее заменил на std::sort из соображений эффективности. Для взаимодействия с пользователем возникла задача обработки ключей командной строки. В процессе разработки программы я расширил свои навыки владения языком C++ и узнал много нового о стандартной библиотеке шаблонов STL, научился писать дерево решений, освежил свои навыки работы с динамической памятью.

Здорово,что такая интересная задача как классификация пространства может быть решена "ручками"на C++,а не просто и бездумно "Black-box"библиотекой из Python для ML !