МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

# Кафедра Інформатики

### ЗВІТ

### з лабораторної роботи № 4

з дисципліни: «[Машинне навчання](https://dl.nure.ua/course/view.php?id=4882)»

Виконав: Перевірила:

ст. гр. ІТІНФ-20-1 Шафроненко А.Ю.

Самченко С. О.

Харків 2022

**Мета:** Ознайомитись з побудовою дерев рішень та з дикретизацією атрибутів.

**Завдання:**

**Варіант №2:** Провести дискретизацію атрибутів А1 і А2 для такої вибірки:

|  |  |  |
| --- | --- | --- |
| Класс | А1 | А2 |
| + | 7 | 15 |
| – | 24 | 32 |
| + | 8 | 135 |
| – | 75 | 56 |
| – | 38 | 24 |
| + | 12 | 62 |

**Хід Роботи:**

Entropy (D) = p+ log2 p+ q log2 q

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1- Результати роботи

**Код програми:**

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

namespace MNtAD\_4

{

    public class DataSet

    {

        string classification;

        double a1;

        double a2;

        public string Classification { get { return classification; } set { classification = value; } }

        public double A1 { get { return a1; } set { a1 = value; } }

        public double A2 { get { return a2; } set { a2 = value; } }

        public DataSet(string classification, double a1, double a2)

        {

            this.classification = classification;

            this.a1 = a1;

            this.a2 = a2;

        }

        public DataSet(DataSet dataSet)

        {

            classification = dataSet.classification;

            a1 = dataSet.a1;

            a2 = dataSet.a2;

        }

        public string PrintDataset()

        {

            string result = classification + String.Format("\t{0:f3}", a1) + String.Format("\t{0:f3}", a2);

            return result;

        }

        public string PrintDataset(double constraint)

        {

            string result = classification + '\t';

            if (a1 < constraint) result += "+\t";

            else result += "-\t";

            result += String.Format("\t{0:f3}", a2);

            return result;

        }

        public string PrintDataset(double constraint1, double constraint2)

        {

            string result = classification + '\t';

            if (a1 < constraint1) result += "+\t";

            else result += "-\t";

            if (a2 < constraint2) result += "+\t";

            else result += "-\t";

            return result;

        }

    }

    class Program

    {

        List<DataSet> dataList = new List<DataSet>();

        public double Mean(double a,double b)

        {

            return (a + b) / 2;

        }

        public List<double> Threshold(List<DataSet> list, int atribute)

        {

            string inClassPrevious = "-";

            List<double> threshold = new();

            if (list[0].Classification == "+")

            {

                inClassPrevious = "+";

            }

            if (atribute == 1)

            {

                for(int i = 1; i < list.Count; ++i)

                {

                    if (inClassPrevious == list[i].Classification) continue;

                    else {

                        threshold.Add((list[i].A1 + list[i - 1].A1) / 2);

                        if (inClassPrevious == "+") inClassPrevious = "-";

                        else inClassPrevious = "+";

                    }

                }

            }

            if (atribute == 2)

            {

                for (int i = 1; i < list.Count; ++i)

                {

                    if (inClassPrevious == list[i].Classification) continue;

                    else

                    {

                        threshold.Add((list[i].A2 + list[i - 1].A2) / 2);

                        if (inClassPrevious == "+") inClassPrevious = "-";

                        else inClassPrevious = "+";

                    }

                }

            }

            return threshold;

        }

        public double FindD()

        {

            double positive = 0.0f;

            foreach (DataSet d in dataList)

            {

                if (d.Classification == "+") { positive++; }

            }

            return positive / dataList.Count;

        }

        public double FindD(double constraint, bool more, int atribute)

        {

            double positive = 0.0;

            double countResult = 0.0;

            foreach(DataSet d in dataList)

            {

                if (atribute == 1) {

                    if (more)

                    {

                        if (d.A1 > constraint)

                        {

                            countResult++;

                            if (d.Classification == "+")

                            {

                                positive++;

                            }

                        }

                    }

                    else

                    {

                        if (d.A1 < constraint)

                        {

                            countResult++;

                            if (d.Classification == "+")

                            {

                                positive++;

                            }

                        }

                    }

                }

                if (atribute == 2)

                {

                    if (more)

                    {

                        if (d.A2 > constraint)

                        {

                            countResult++;

                            if (d.Classification == "+")

                            {

                                positive++;

                            }

                        }

                    }

                    else

                    {

                        if (d.A2 < constraint)

                        {

                            countResult++;

                            if (d.Classification == "+")

                            {

                                positive++;

                            }

                        }

                    }

                }

            }

            if (countResult != 0) {

                return positive / countResult;

            }

            else {

                return 0;

            }

        }

        public double Entropy(double p)

        {

            if (p == 1.0 || p == 0.0) return 0;

            double q = 1.0 - p;

            return - p \* Math.Log2(p) - q \* Math.Log2(q);

        }

        public double Gain(double constraint, int attribute)

        {

            double D = FindD();

            double dMore, dLess;

            if (attribute == 1) {

                dMore = FindD(constraint, true, 1);

                dLess = FindD(constraint, false, 1);

            }

            else

            {

                dMore = FindD(constraint, true, 2);

                dLess = FindD(constraint, false, 2);

            }

            double result = Entropy(D) - ((dLess \* Entropy(dLess) / D) + (dMore \* Entropy(dMore) / D));

            return result;

        }

        static void Main(string[] args)

        {

            Program program = new();

            string path = @"D:\work\data.txt";

            using (StreamReader sr = new StreamReader(path, System.Text.Encoding.Default))

            {

                string line;

                while ((line = sr.ReadLine()) != null)

                {

                    string[] s = line.Split("\t");

                    DataSet dataSet = new DataSet(s[0], double.Parse(s[1]), double.Parse(s[2]));

                    program.dataList.Add(dataSet);

                }

            }

            List<DataSet> dataSets1 = program.dataList.OrderByDescending(set => set.A1).ToList();

            Console.WriteLine("Base dataset");

            foreach (DataSet d in program.dataList)

            {

                Console.WriteLine(d.PrintDataset());

            }

            Console.WriteLine("First dataset:");

            foreach (DataSet d in dataSets1)

            {

                Console.WriteLine(d.PrintDataset());

            }

            Console.WriteLine("Threshold1:");

            List<double> threshold1 = program.Threshold(dataSets1, 1);

            for (int i = 0; i < threshold1.Count; ++i)

            {

                Console.WriteLine($"D{i + 1} = {threshold1[i]}");

            }

            List<double> gainForEachThreshold1 = new();

            for (int i = 0; i < threshold1.Count; ++i)

            {

                gainForEachThreshold1.Add(program.Gain(threshold1[i], 1));

                Console.WriteLine($"Gain {i + 1} = {gainForEachThreshold1[i]}");

            }

            double firstMaxThreshold = threshold1[gainForEachThreshold1.IndexOf(gainForEachThreshold1.Max())];

            Console.WriteLine("Dataset after first algorithm step:");

            foreach (DataSet d in program.dataList)

            {

                Console.WriteLine(d.PrintDataset(firstMaxThreshold));

            }

            List<DataSet> dataSets2 = program.dataList.OrderByDescending(set => set.A2).ToList();

            Console.WriteLine("Second dataset:");

            foreach (DataSet d in dataSets2)

            {

                Console.WriteLine(d.PrintDataset());

            }

            Console.WriteLine("Threshold2:");

            List<double> threshold2 = program.Threshold(dataSets2, 2);

            for (int i = 0; i < threshold2.Count; ++i)

            {

                Console.WriteLine($"D{i + 1} = {threshold2[i]}");

            }

            List<double> gainForEachThreshold2 = new();

            for (int i = 0; i < threshold2.Count; ++i)

            {

                gainForEachThreshold2.Add(program.Gain(threshold2[i], 2));

                Console.WriteLine($"Gain {i + 1} = {gainForEachThreshold2[i]}");

            }

            double secondMaxThreshold = threshold2[gainForEachThreshold2.IndexOf(gainForEachThreshold2.Max())];

            Console.WriteLine("Result dataset");

            foreach (DataSet d in program.dataList)

            {

                Console.WriteLine(d.PrintDataset(firstMaxThreshold, secondMaxThreshold));

            }

        }

    }

}

**Висновок:**

У ході лабораторної роботи я реалізував розрахунок дерева рішення за допомогою критеріїв виходу інформації та порогів. Було створено програму для розрахунку результатів алгоритму.