МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

# Кафедра Інформатики

### ЗВІТ

### з лабораторної роботи № 1

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 заняття:**

Засвоїти основні складові навчальної вибірки (приклади, атрибути, значення атрибутів, цільовий атрибут), повторити основні статистичні характеристики, необхідні для попередньої обробки навчальних вибірок.

**Завдання:**

В ході лабораторної роботи необхідно написати 4 процедури попередньої обробки вибірок:

* процедура доступу до довільного елемента вибірки;
* процедура випадкового сортування записів вибірки;
* процедура обчислення статистичних характеристик для значень кожного атрибуту (мат. очікування та дисперсії);
* процедура призначення номерів класів для значень цільового атрибуту, яка замінює символьні значення класів їх порядковими номерами.

**Хід Роботи:**

Був обраний набір даних Balance Scale Data Set в якому присутні наступні атрибути:

**Attribute Information:**

1. Class Name: 3 (L, B, R)  
2. Left-Weight: 5 (1, 2, 3, 4, 5)  
3. Left-Distance: 5 (1, 2, 3, 4, 5)  
4. Right-Weight: 5 (1, 2, 3, 4, 5)  
5. Right-Distance: 5 (1, 2, 3, 4, 5)

Для обробки даних було створено класс BalanceScale в якому було 5 атрибутів з яких атрубут name має символьного типу та процедуру перетворення на чисельного значення, та інші атрибути чисельного типу.

Процедура доступа реалізована так що з загальної вибірки по списку обмежень для конкретних атрибутів відбираються тільки ті значення які відповідають всім вимогам.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 1 - Процедура доступу до елементу вибірки

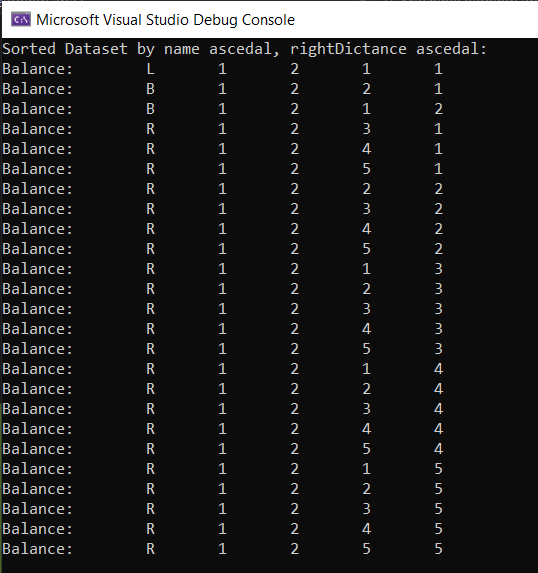
Сортування відбувається за вказаною кількістю атрибутів та типом сортування для кожного з них(по зростанню, по спаду) :

Рис. 2 - Процедура випадкового сортування вибірки;

З відібраних процедурою доступу набору даних для кожного атрибуту в наборі даних підраховується математичне очікування, дисперсія та середньо-квадратичне відхилення за формулами стастики.

Изображение выглядит как текст

Автоматически созданное описание

Рис. 3 - Процедура підрахунку мат. очікування и дисперсії та середньо-квадратичного відхилення

В останній процедурі Для кожного підкласу атрибуту Name створюється чисельний еквівалент та значення замінюється цим числом.

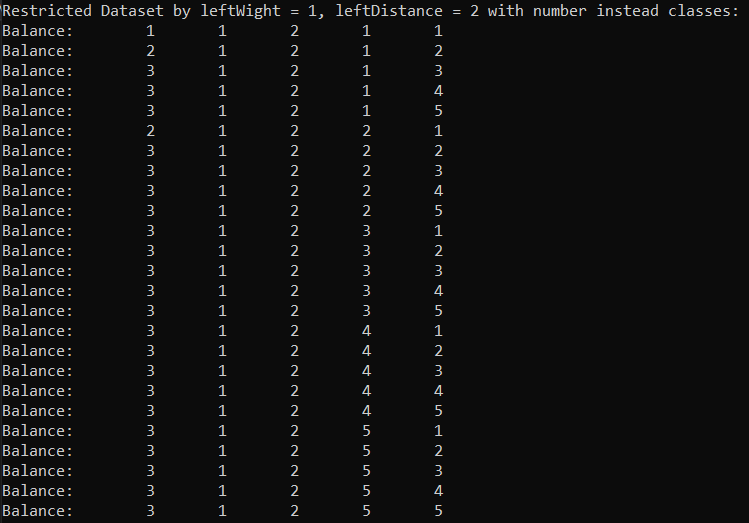


Рис. 4 - Процедура присвоєння номерів класів цільовим значенням атрибутів, що замінюють значення символів класів їх серійними номерами.

**Код програми:**

using System;

using System.Collections.Generic;

using System.IO;

namespace MachineLearningLaba1

{

    public class BalanceScale {

        char name;

        int leftWeight;

        int leftDistance;

        int rightWeight;

        int rightDistance;

        public char Name {

            get

            {

                return name;

            }

            set

            {

                name = value;

            }

        }

        public int LeftWeight

        {

            get

            {

                return leftWeight;

            }

            set

            {

                leftWeight = value;

            }

        }

        public int LeftDistance

        {

            get

            {

                return leftDistance;

            }

            set

            {

                leftDistance = value;

            }

        }

        public int RightWeight

        {

            get

            {

                return rightWeight;

            }

            set

            {

                rightWeight = value;

            }

        }

        public int RightDistance

        {

            get

            {

                return rightDistance;

            }

            set

            {

                rightDistance = value;

            }

        }

        public BalanceScale(char name, int leftWeight, int leftDistance,int rightWeight, int rightDistance) {

            this.name = name;

            this.leftWeight = leftWeight;

            this.leftDistance = leftDistance;

            this.rightWeight = rightWeight;

            this.rightDistance = rightDistance;

        }

        public void PrintBalanceScale()

        {

            Console.WriteLine("Balance:\t" + name + '\t' + leftWeight + '\t' + leftDistance + '\t' + rightWeight + '\t' + rightDistance);

        }

        public int NumberByName() {

            if(name == 'L')

            {

                return 1;

            }

            else if (name == 'B')

            {

                return 2;

            }

            else

            {

                return 3;

            }

        }

        public char NameByNumber(int value)

        {

            if (value == 1)

            {

                return 'L';

            }

            else if (value == 2)

            {

                return 'B';

            }

            else

            {

                return 'R';

            }

        }

        public int GetValueOfAtribute(string atribute)

        {

            switch (atribute)

            {

                case "name":

                    return NumberByName();

                case "leftWeight":

                    return leftWeight;

                case "leftDistance":

                    return leftDistance;

                case "rightWeight":

                    return rightWeight;

                case "rightDistance":

                    return rightDistance;

                default:

                    return 0;

            }

        }

        public void SetValueOfAtribute(string atribute, int value)

        {

            switch (atribute)

            {

                case "name":

                    name = NameByNumber(value);

                    break;

                case "leftWeight":

                    leftWeight = value;

                    break;

                case "leftDistance":

                    leftDistance = value;

                    break;

                case "rightWeight":

                    rightWeight = value;

                    break;

                case "rightDistance":

                    rightDistance = value;

                    break;

            }

        }

    }

    class Program

    {

        List<BalanceScale> balanceScales = new List<BalanceScale>();

        private void FindElementsUsingCategory(Dictionary<string, int> constraint)

        {

            List<BalanceScale> result = new List<BalanceScale>();

            foreach (BalanceScale balanceScale in balanceScales)

            {

                bool flag = true;

                foreach (string atribute in constraint.Keys)

                {

                    if (!balanceScale.GetValueOfAtribute(atribute).Equals(constraint[atribute]))

                    {

                        flag = false;

                    }

                }

                if (flag) {

                    result.Add(balanceScale);

                }

            }

            balanceScales = result;

        }

        private void PrintElements()

        {

            foreach (BalanceScale balance in balanceScales)

            {

                balance.PrintBalanceScale();

            }

        }

        private void BubleSortForBalanceScale(string atribute, int startIndex, int endIndex, bool asc)

        {

            for (int i = startIndex; i < endIndex; i++)

            {

                for (int j = startIndex; j < endIndex; j++)

                {

                    if (asc)

                    {

                        if (balanceScales[j].GetValueOfAtribute(atribute) > balanceScales[j+1].GetValueOfAtribute(atribute))

                        {

                            BalanceScale bl = balanceScales[j + 1];

                            balanceScales.RemoveAt(j + 1);

                            balanceScales.Insert(j + 1, balanceScales[j]);

                            balanceScales.RemoveAt(j);

                            balanceScales.Insert(j, bl);

                        }

                    }

                    else

                    {

                        if (balanceScales[j].GetValueOfAtribute(atribute) < balanceScales[j + 1].GetValueOfAtribute(atribute))

                        {

                            BalanceScale bl = balanceScales[j + 1];

                            balanceScales.RemoveAt(j + 1);

                            balanceScales.Insert(j + 1, balanceScales[j]);

                            balanceScales.RemoveAt(j);

                            balanceScales.Insert(j, bl);

                        }

                    }

                }

            }

        }

        private int LastRepeadingIndex(string atribute,int j)

        {

            int result = balanceScales.Count - 1;

            for (int i = j; i < balanceScales.Count - 1; ++i)

            {

                if(balanceScales[i].GetValueOfAtribute(atribute) == balanceScales[i + 1].GetValueOfAtribute(atribute))

                {

                    continue;

                }

                else

                {

                    result = i;

                    break;

                }

            }

            return result;

        }

        public void SortElements(string[] atributes, bool[] asc)

        {

            BubleSortForBalanceScale(atributes[0],0,balanceScales.Count-1, asc[0]);

            for (int countAtributes = 1; countAtributes < atributes.Length; ++countAtributes) {

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

                {

                    int lastIndex = LastRepeadingIndex(atributes[countAtributes - 1], i);

                    if (lastIndex <= i)

                    {

                        continue;

                    }

                    else

                    {

                        BubleSortForBalanceScale(atributes[countAtributes],i, lastIndex, asc[countAtributes]);

                    }

                }

            }

        }

        //Statistics

        public double MathematicalExpectation(string atribute) {

            double result = 0;

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

            {

                result += balanceScales[i].GetValueOfAtribute(atribute);

            }

            return result / balanceScales.Count;

        }

        public double Dispertion(string atribute)

        {

            double result = 0;

            double expectation = MathematicalExpectation(atribute);

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

            {

                result += Math.Pow(balanceScales[i].GetValueOfAtribute(atribute) - expectation, 2);

            }

            return result / (balanceScales.Count - 1);

        }

        public double StandardDeviation(string atribute)

        {

            return (double) Math.Sqrt(Dispertion(atribute));

        }

        public void PrintStatisticalCharacteristics() {

            string[] atributes = { "name", "leftWeight", "leftDistance", "rightWeight", "rightDistance" };

            Console.WriteLine("Statistics characteristics:");

            for (int i = 0; i < atributes.Length; ++i)

            {

                double mathExpectation = MathematicalExpectation(atributes[i]);

                string output = "Atribute " + atributes[i] + ":\t";

                output += " X = " + mathExpectation;

                output += " D = " + Dispertion(atributes[i]);

                output += " S = " + StandardDeviation(atributes[i]);

                Console.WriteLine(output);

            }

        }

        static void Main(string[] args)

        {

            List<BalanceScale> originalScale = new List<BalanceScale>();

            Program program = new Program();

            string[] atributes = { "name", "leftWeight", "leftDistance", "rightWeight", "rightDistance" };

            string path = @"D:\work\balance-scale.data";

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

            {

                string line;

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

                {

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

                    BalanceScale bs = new BalanceScale(s[0].ToCharArray()[0], int.Parse(s[1]), int.Parse(s[2]),

                        int.Parse(s[3]), int.Parse(s[4]));

                    originalScale.Add(bs);

                }

                program.balanceScales.AddRange(originalScale);

                Dictionary<string, int> requiredElements = new Dictionary<string, int>();

                requiredElements.Add(atributes[1], 1);

                requiredElements.Add(atributes[2], 2);

                program.FindElementsUsingCategory(requiredElements);

                program.PrintStatisticalCharacteristics();

                program.FindElementsUsingCategory(requiredElements);

                string[] test = new string[] { atributes[0], atributes[4] };

                bool[] ascFull = new bool[] { true, true };

                program.SortElements(test,ascFull);

                program.PrintElements();

            }

        }

    }

}

**Висновок:** У ході лабораторної роботи я засвоїв основні складові навчальної вибірки, повторив основні статистичні характеристики, необхідні для попередньої обробки навчальних вибірок, а саме такі як математичне очікування, дисперсія та середньоквадратичне очікування. Також було реалізовано програму за допомогою якої можна відбирати, сортувати та обробляти данні за допомогою процедур, які реалізовані як методи класів що забезпечує цілісність та безпечність використання даних, а для атрибутів з нечисленними можливість застосування нечіткої логіки.