Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Институт информационных технологий

Факультет компьютерных технологий

Специальность ПОИТ

**Контрольная Работа**

По дисциплине «Методы и алгоритмы принятия решений»

Лабораторная работа 2

Выполнил: студент гр. 881072 Пискарёв К.А.

Проверил: Бакунов А.М.

Минск 2020

**ЛАБОРАТОРНАЯ РАБОТА №2**

**РАСПОЗНАВАНИЕ ОБРАЗОВ НА ОСНОВЕ САМООБУЧЕНИЯ**

Цель работы: изучить особенности распознавания образов в самообучающихся системах и научиться классифицировать объекты с помощью алго- ритма максимина.

Порядок выполнения работы

1. Изучение теоретической части лабораторной работы.

2. Реализация алгоритма максимина.

3. Защита лабораторной работы.

По сравнению с методами контролируемого обучения алгоритмы самообучения отличаются большей неполнотой информации. В этих алгоритмах не известны ни классы, ни их количество, ни признаки. Необходимым минимумом информации для классификации объектов являются сами образы и их признаки, без этого не выполняется ни один алгоритм. В обучении «без учителя» алгоритм самостоятельно определяет классы, на которые делится исходное множество данных, и одновременно определяет присущие им признаки. Для разделения данных используется следующий универсальный критерий. Процесс организуется так, чтобы среди всех возможных вариантов группировок найти такой, когда группы обладают наибольшей компактностью.

В качестве примера метода распознавания образов, использующего процедуру самообучения, рассмотрим алгоритм максимина.

Исходные данные – число образов, которые нужно разделить на классы. Количество образов предлагается брать в диапазоне от 1000 до 100 000.

Признаки объектов задаются случайным образом, это координаты векторов.

Цель и результат работы алгоритма – исходя из произвольного выбора максимально компактно разделить объекты на классы, определив ядро каждого класса.

**ДЕМОНСТРАЦИЯ РАБОТЫ ПРОГРАММЫ**

После запуска программы отображается форма, представленная на рисунке 1.

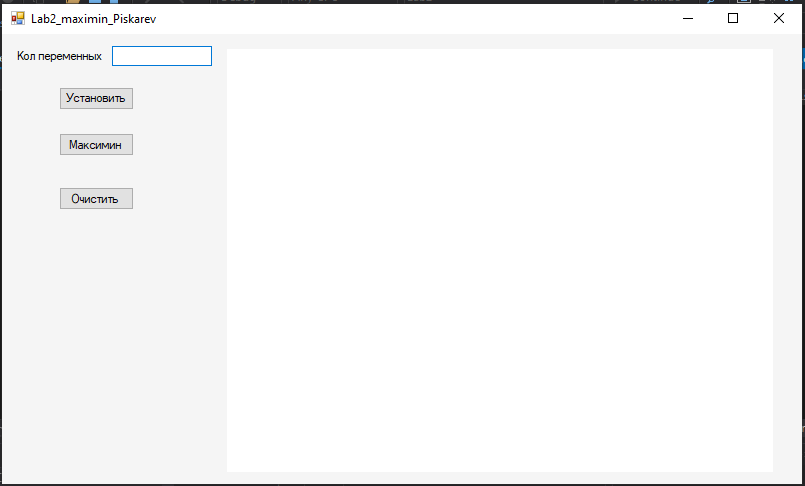


Рисунок 1

Требуется заполнить количество переменных установить, для первичной инициализации. Результат представлен на рисунке 2.

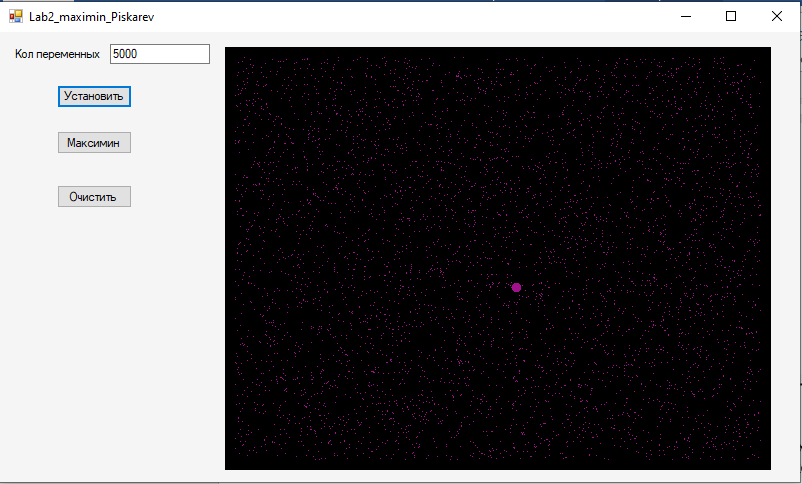


Рисунок 2

Для запуска алгоритма нажать на кнопку Максимин и после этого будет добавлен новый кластер согласно алгоритму. Пример представлен на рисунке 3.

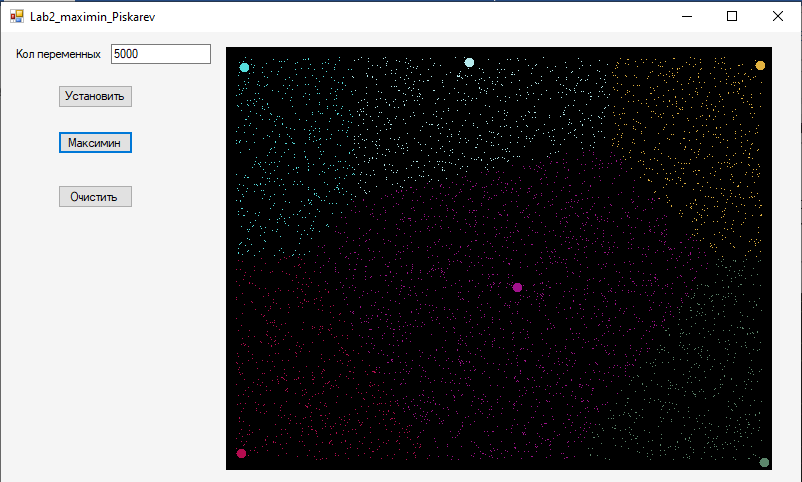
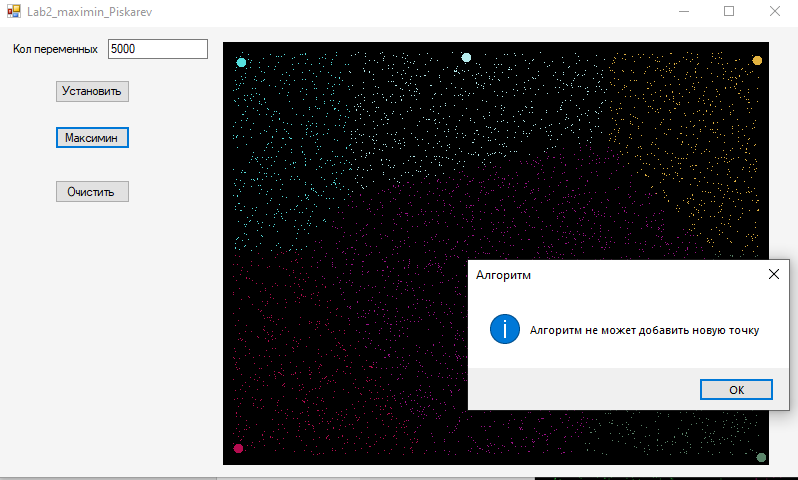


Рисунок 3

Если больше нет возможности согласно условиям добавить новый кластер отображается ошибка. Результат представлен на рисунке 4.



**ПРИЛОЖЕНИЕ**

Код программы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab1

{

static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public class KPoint

{

public int X { get; set; }

public int Y { get; set; }

public int Klaster { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public class Kernel: ICloneable

{

public int Klaster { get; set; }

public Color Color { get; set; }

public KPoint KPoint { get; set; }

public bool Equals(Kernel b)

{

return this.KPoint.X == b.KPoint.X && this.KPoint.Y == b.KPoint.Y;

}

public Object Clone()

{

return new Kernel()

{

Color = this.Color,

Klaster = this.Klaster,

KPoint = new KPoint()

{

X = this.KPoint.X,

Y = this.KPoint.Y,

Klaster = this.KPoint.Klaster,

}

};

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Lab1

{

public class KDistance

{

public Kernel Kernel { get; set; }

public double Distance { get; set; }

}

}

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab1

{

public class Maximin

{

private const int MIN\_KPOINTS\_COUNT = 10;

private const int MAX\_KPOINTS\_COUNT = int.MaxValue;

private const int MIN\_KLASTER\_COUNT = 2;

private const int MAX\_KLASTER\_COUNT = 256;

private const int DEFAULT\_KLASTER = 0;

private const int INDENT = 10;

private Object lockObject = new Object();

private int kpoints\_count, k\_count, xWidth, yHeight = 0;

private bool isMaximin;

private Color[] colors;

private KPoint[] kPoints;

private Kernel[] OldKernels, NewKernels;

private Random rand;

private BufferedGraphics bufferedGraphics;

public Maximin(BufferedGraphics bufferedGraphics)

{

this.bufferedGraphics = bufferedGraphics;

rand = new Random();

}

public void KInit(int kpoint, int width, int height)

{

kpoints\_count = kpoint;

k\_count = 1;

isMaximin = true;

xWidth = width;

yHeight = height;

colors = new Color[100];

OldKernels = new Kernel[100];

NewKernels = new Kernel[100];

kPoints = new KPoint[kpoints\_count];

for (var i = 0; i < kpoints\_count; i++)

{

kPoints[i] = new KPoint();

kPoints[i].X = INDENT + rand.Next(xWidth - 2 \* INDENT);

kPoints[i].Y = INDENT + rand.Next(yHeight - 2 \* INDENT);

kPoints[i].Klaster = DEFAULT\_KLASTER;

}

for (var i = 0; i < k\_count; i++)

{

colors[i] = NewColor();

OldKernels[i] = NewKernels[i] = new Kernel();

OldKernels[i].KPoint = NewKernels[i].KPoint = kPoints[rand.Next(kpoints\_count)];

OldKernels[i].Color = NewKernels[i].Color = colors[i];

OldKernels[i].Klaster = NewKernels[i].Klaster = i;

}

Draw();

}

public void ClearBox()

{

bufferedGraphics.Graphics.Clear(Color.Black);

bufferedGraphics.Render();

}

public void MaxMinDo()

{

if (isMaximin)

{

double[] maxDistances = new double[kpoints\_count];

int[] newKernelIndexes = new int[kpoints\_count];

double maxDistance = 0;

int newKernelIndex = 0;

for (var i = 0; i< k\_count; i++)

{

double maxDistanceInClass = 0;

int newKernelIndexInClass = 0;

for (int j = 0; j < kpoints\_count; j++)

if (kPoints[j].Klaster == i &&

EvklidDistance(kPoints[j], NewKernels[i].KPoint) > maxDistanceInClass)

{

maxDistanceInClass = EvklidDistance(kPoints[j], NewKernels[i].KPoint);

newKernelIndexInClass = j;

}

maxDistances[i] = maxDistanceInClass;

newKernelIndexes[i] = newKernelIndexInClass;

if (maxDistances[i] > maxDistance)

{

maxDistance = maxDistances[i];

newKernelIndex = newKernelIndexes[i];

}

}

double sumPairsDistance = 0;

int pair\_count = 0;

for(int i = 0; i< k\_count; i++)

{

for (int j = 0; j < k\_count; j++)

{

if (i != j)

{

sumPairsDistance+= EvklidDistance(NewKernels[i].KPoint, NewKernels[j].KPoint);

pair\_count++;

}

}

}

if ((maxDistance> 0 && (sumPairsDistance == 0 || pair\_count ==0)) || maxDistance > (sumPairsDistance / (pair\_count \* 2))){

colors[k\_count] = NewColor();

OldKernels[k\_count] = NewKernels[k\_count] = new Kernel();

OldKernels[k\_count].KPoint = NewKernels[k\_count].KPoint = kPoints[newKernelIndex];

OldKernels[k\_count].Color = NewKernels[k\_count].Color = colors[k\_count];

OldKernels[k\_count].Klaster = NewKernels[k\_count].Klaster = k\_count;

k\_count++;

ResetPoints();

Draw();

}

else

{

ShowError();

}

}

else

{

ShowError();

}

}

private void ShowError()

{

MessageBox.Show("Алгоритм не может добавить новую точку", "Алгоритм", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

private void ResetPoints()

{

Dictionary<Kernel, List<KPoint>> kernelPoints = new Dictionary<Kernel, List<KPoint>>();

for (var i = 0; i < k\_count; i++)

{

OldKernels[i] = (Kernel)NewKernels[i].Clone();

kernelPoints.Add(NewKernels[i], new List<KPoint>());

}

Parallel.ForEach(kPoints,

new ParallelOptions() { MaxDegreeOfParallelism = kPoints.Count() },

(kPoint) =>

{

KDistance[] kDistances = new KDistance[k\_count];

KDistance Kmin = null;

int count = 0;

foreach (var kernel in kernelPoints.Keys)

{

kDistances[count] = new KDistance();

kDistances[count].Kernel = kernel;

kDistances[count].Distance = EvklidDistance(kPoint, NewKernels[count].KPoint);

if (Kmin == null)

{

Kmin = kDistances[count];

}

else if (Kmin.Distance > kDistances[count].Distance)

{

Kmin = kDistances[count];

}

count++;

}

kPoint.Klaster = Kmin.Kernel.Klaster;

lock (lockObject)

{

kernelPoints[Kmin.Kernel].Add(kPoint);

}

});

}

private void Draw()

{

bufferedGraphics.Graphics.Clear(Color.Black);

for (var i = 0; i < kpoints\_count; i++)

{

bufferedGraphics.Graphics.FillRectangle(new SolidBrush(NewKernels[kPoints[i].Klaster].Color),

kPoints[i].X, kPoints[i].Y, 1, 1);

}

for (var i = 0; i < k\_count; i++)

{

bufferedGraphics.Graphics.FillEllipse(new SolidBrush(NewKernels[i].Color),

NewKernels[i].KPoint.X, NewKernels[i].KPoint.Y, 10, 10);

}

bufferedGraphics.Render();

}

private Color NewColor()

{

return Color.FromArgb(rand.Next(256), rand.Next(256), rand.Next(256));

}

private double EvklidDistance(KPoint a, KPoint b)

{

return Math.Sqrt(Math.Pow((a.X - b.X), 2) + Math.Pow((a.Y - b.Y), 2));

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Lab1

{

public partial class Form1 : Form

{

private Graphics graphics;

private BufferedGraphics bufferedGraphics;

private Maximin kMeans;

private bool isSetValue;

public Form1()

{

InitializeComponent();

graphics = displayBox.CreateGraphics();

bufferedGraphics = new BufferedGraphicsContext().Allocate(graphics, new Rectangle(0, 0, displayBox.Width, displayBox.Height));

kMeans = new Maximin(bufferedGraphics);

isSetValue = false;

}

private void set\_Click(object sender, EventArgs e)

{

kMeans.KInit(int.Parse(textBox1.Text), displayBox.Width, displayBox.Height);

isSetValue = true;

}

private void clear\_Click(object sender, EventArgs e)

{

kMeans.ClearBox();

isSetValue = false;

}

private void kmeans\_Click(object sender, EventArgs e)

{

kMeans.MaxMinDo();

}

}

}