**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

по дисциплине «Введение в нейронные сети»

**Бинарная классификация с использованием машины опорных векторов**

Выполнил: студент гр. ИТИ-21

Шеметков К.Д.

Принял: преподаватель

Карабчикова Е.А.

Гомель 2020

**Цель работы:** изучить задачу бинарной классификации с учителем с использованием машины опорных векторов.

**Задание**

Разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием метода опорных векторов.

В качестве параметров программа должна принимать файл, содержащий:

* Ссылки на файлы с обучающей выборкой;
* Ссылки на файлы с тестируемыми образцами;
* Ссылку на файл с результатами проверки образцов (принадлежит/не принадлежит)

Если происходит запуск без параметров, то запускается программа с графическим интерфейсом, ссылки на обучающую выборку берутся из конфигурационного файла аналогичной структуры, как и файл параметров командной строки.

Решить задачу из лабораторной работы №2. В качестве бинарного классификатора использовать метод опорных векторов. Определить вид ядра SVM, обеспечивающий наилучшее разделение классов.

Сравнить полученные результаты с результатами лабораторной работы №2. Построить графики. Сделать выводы.

**Вариант 14/2**

Вычисление значения функции XOR.

**Ход работы**

В ходе выполнения лабораторной работы была разработана машина опорных векторов с полиномиальным ядром, которое обеспечивает лучший результат работы машины опорных векторов. Также была подготовлена обучающая выборка, которая используется для обучения машины опорных векторов. Обучающая выборка изображена на рисунке 1.



Рисунок 1 – Обучающая выборка.

Для реализации машины опорных векторов была создана библиотека классов SVMLib. В библиотеке классов был создан интерфейс IKernel, описывающий базовый функционал ядра без реализации функций. Также были реализованы наследуемые классы Linear, описывающий базовое ядро машины и Polynomial, описывающий полиномиальное ядро. Затем был создан класс SupportVectorMachine и наследуемый от него KernelSupportVectorMachine, описывающие функционал машины опорных векторов. Был создан класс DataSet, имеющий массив сигналов, для подачи на входной слой.

Также был разработан графический интерфейс, позволяющий загружать обучающую выборку и преобразовывать его в массив входных сигналов, создавать машину опорных векторов, производить обучение машины с отслеживанием процесса обучения. Отслеживание процесса осуществляется благодаря графику. Интерфейс приложения представлен на рисунке 3. График обучения SVM изображен на рисунке 4. Результат тестирования нейронной сети на бинаризированном изображение представлен на рисунке 5.

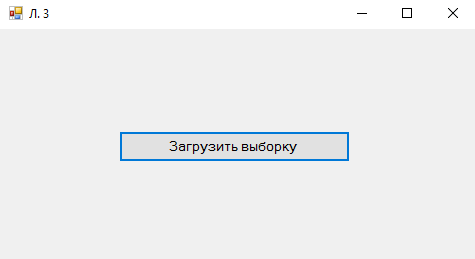


Рисунок 2 – Интерфейс приложения.

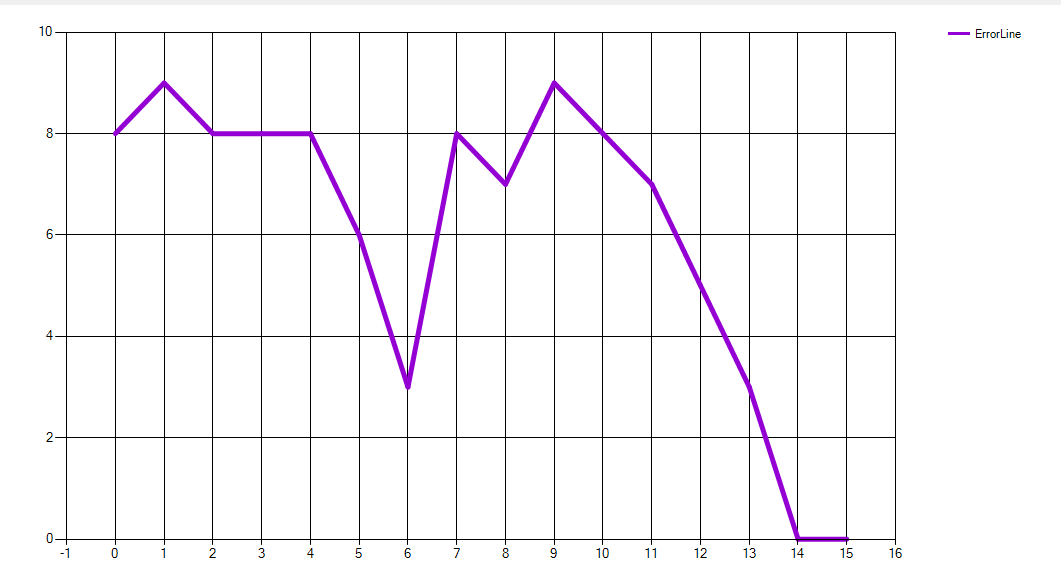


Рисунок 3– График обучения машины опорных векторов.

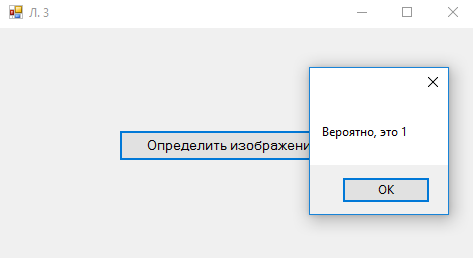


Рисунок 4 – Результаты тестирования на изображении.

Сопоставив график с графиками из лабораторной работы №2 видно, что машина опорных векторов с правильно подобранными параметрами обучается качественнее и быстрее. Количество изменений алгоритма оптимизации довольно быстро стремится к 0. То есть машина опорных векторов хорошо подходит для бинарной классификации данных.

**Вывод:** в ходе выполнения лабораторной работы была изучена машина опорных векторов и методы ее оптимизации. А также были получены практические знания создания SVM для распознавания бинарных изображений.

**Листинг Program.cs:**

using FormForNeuronSystem;

using NeuroLib;

using SVMLib;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Windows.Forms;

namespace SVMForm

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main(string[] args)

{

if (args.Length == 0)

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new SVMApp());

}

else

{

string datasetPath = args[0];

string testPath = args[1];

string outputPath = args[2];

List<Data> dataSets = new List<Data>();

FillDataset(dataSets, datasetPath);

var machine = new KernelSupportVectorMachine(new Polynomial(2), 25);

var learn = new SequentialMinimalOptimization(machine, dataSets.ToArray());

\_ = learn.Run();

dataSets.Clear();

List<string> paths = FillDataset(dataSets, testPath);

double[] output = machine.Compute(dataSets.ToArray());

using (StreamWriter writer = new StreamWriter(outputPath))

{

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

{

writer.WriteLine(paths[i] + ':' + output[i]);

}

}

}

}

private static List<string> FillDataset(List<Data> dataSet, string datasetPath)

{

List<string> paths = new List<string>();

using (StreamReader reader = new StreamReader(datasetPath))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

double expected = double.Parse(line.Substring(0, 1));

string path = line.Substring(2);

using (Bitmap bitmap = new Bitmap(path))

{

dataSet.Add(bitmap.ToDataSet(expected == 0 ? -1 : 1));

paths.Add(path);

}

}

}

return paths;

}

}

}

**Листинг BitmapConverter.cs:**

using NeuroLib;

using System.Drawing;

namespace FormForNeuronSystem

{

public static class BitmapProcessor

{

public static Data ToDataSet(this Bitmap bitmap, double expected)

{

double[] inputs = new double[25];

for (int i = 0; i < bitmap.Height; i++)

{

for (int j = 0; j < bitmap.Width; j++)

{

Color color = bitmap.GetPixel(j, i);

if (color.R == 0 && color.G == 0 && color.B == 0)

inputs[i \* bitmap.Height + j] = 1;

else

inputs[i \* bitmap.Height + j] = 0;

}

}

return new Data(expected, inputs);

}

}

}

**Листинг SVMFormApp.cs:**

using System;

using System.Collections.Generic;

using System.Drawing;

using System.IO;

using System.Windows.Forms;

using NeuralLibrary;

using FormForNeuronSystem;

using SVMLib;

namespace SVMForm

{

public partial class SVMFormApp : Form

{

private bool isTrained = false;

private List<DataSet> dataSets;

private KernelSupportVectorMachine machine;

private SVMGraphic graphicsForm;

public SVMFormApp()

{

InitializeComponent();

dataSets = new List<DataSet>();

}

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

{

OpenFileDialog openFile = new OpenFileDialog();

if (openFile.ShowDialog() == DialogResult.OK)

{

dataSets.Clear();

string file = openFile.FileName;

using (StreamReader reader = new StreamReader(file))

{

while (!reader.EndOfStream)

{

string line = reader.ReadLine();

double expected = double.Parse(line.Substring(0, 1));

string path = line.Substring(2);

using (Bitmap bitmap = new Bitmap(path))

{

dataSets.Add(bitmap.ToDataSet(expected == 0 ? -1 : 1));

}

}

}

machine = new KernelSupportVectorMachine(new Polynomial(2), 25);

var learn = new SequentialMinimalOptimization(machine, dataSets.ToArray());

double[] changes = learn.Run();

graphicsForm = new SVMGraphic();

graphicsForm.Show();

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

{

graphicsForm.SVMData.Add(i, changes[i]);

graphicsForm.DrawFucntion();

}

isTrained = true;

}

}

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

{

if (!isTrained)

{

MessageBox.Show("Машина не обучена.");

return;

}

OpenFileDialog openFile = new OpenFileDialog();

openFile.RestoreDirectory = true;

if (openFile.ShowDialog() == DialogResult.OK)

{

using (Bitmap bitmap = new Bitmap(openFile.FileName))

{

DataSet dataset = bitmap.ToDataSet(0);

double output = machine.Compute(dataset.InputsSignals);

int res = output > 0 ? 1 : 0;

MessageBox.Show("Результат: " + output.ToString()

+ " = " + res.ToString());

}

}

}

}

}

**Листинг SVMGraphic.cs:**

using System.Collections.Generic;

using System.Linq;

using System.Windows.Forms;

namespace SVMForm

{

public partial class SVMGraphic : Form

{

public GraphicsData SVMData { get; set; }

public SVMGraphic()

{

InitializeComponent();

SVMData = new GraphicsData();

}

public void DrawFucntion()

{

var data = SVMData.Data.Last();

Graphic.Series["ErrorLine"].Points.AddXY(data.Key, data.Value);

}

}

public class GraphicsData

{

public Dictionary<int, double> Data { get; private set; }

public GraphicsData()

{

Data = new Dictionary<int, double>();

}

public void Add(int epoch, double changes)

{

Data.Add(epoch, changes);

}

}

}

**Листинг LinearClassif.cs:**

namespace SVMLib

{

public class LinearClassif : IKernel

{

public double Constant { get; set; }

public LinearClassif(double constant = 0)

{

Constant = constant;

}

public double Function(double[] x, double[] y)

{

double sum = Constant;

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

sum += x[i] \* y[i];

return sum;

}

}

}

**Листинг Polynomial.cs:**

using System;

namespace SVMLib

{

public class Polynomial : IKernel

{

private int degree;

public int Degree

{

get { return degree; }

set

{

if (value <= 0)

throw new ArgumentOutOfRangeException("value", "Degree must be positive.");

degree = value;

}

}

public double Constant { get; set; }

public Polynomial(int degree, double constant = 1.0)

{

this.degree = degree;

Constant = constant;

}

public double Function(double[] x, double[] y)

{

double sum = Constant;

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

sum += x[i] \* y[i];

return Math.Pow(sum, degree);

}

}

}

**Листинг SequentialOptimization.cs:**

using NeuroLib;

using System;

using System.Collections.Generic;

namespace SVMLib

{

public class SequentialOptimization : IVectorLearning

{

private static Random random = new Random();

private double[][] inputs;

private int[] outputs;

private double c = 1.0;

private double tolerance = 1e-3;

private double epsilon = 1e-3;

private bool useComplexityHeuristic;

private SupportVectorMachine machine;

private IKernel kernel;

private double[] alpha;

private double bias;

private double[] errors;

/// <summary>

/// Инициализация алгоритма SMO.

/// </summary>

/// <param name="machine">SVM.</param>

/// <param name="dataset">Набор данных для обучения.</param>

public SequentialOptimization(SupportVectorMachine machine, Data[] dataset)

{

if (machine == null)

throw new ArgumentNullException("machine");

if (machine.Inputs > 0)

{

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

if (dataset[i].InputsSignals.Length != machine.Inputs)

throw new ArgumentException("The size of the input vectors does not match the expected number of inputs of the machine");

}

this.machine = machine;

VectorMachine ksvm = machine as VectorMachine;

this.kernel = (ksvm != null) ? ksvm.Kernel : new LinearClassif();

inputs = new double[dataset.Length][];

outputs = new int[dataset.Length];

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

{

inputs[i] = new double[dataset[i].InputsSignals.Length];

for (int j = 0; j < dataset[i].InputsSignals.Length; j++)

inputs[i][j] = dataset[i].InputsSignals[j];

outputs[i] = (int)dataset[i].Expected;

}

}

/// <summary>

/// Параметр сложности.

/// </summary>

public double Complexity

{

get { return this.c; }

set { this.c = value; }

}

/// <summary>

/// Автоматическое вычисление сложности.

/// </summary>

public bool UseComplexityHeuristic

{

get { return useComplexityHeuristic; }

set { useComplexityHeuristic = value; }

}

/// <summary>

/// Зона чувствительности.

/// </summary>

public double Epsilon

{

get { return epsilon; }

set { epsilon = value; }

}

/// <summary>

/// Допуск сходимости.

/// </summary>

public double Tolerance

{

get { return this.tolerance; }

set { this.tolerance = value; }

}

/// <summary>

/// Запуск алгоритма SMO

/// </summary>

/// <param name="computeError">Вычислять ли ошибку после завершения.</param>

/// <returns>Ошибка.</returns>

public double[] Run()

{

int N = inputs.Length;

if (useComplexityHeuristic)

c = ComputeComplexity();

this.alpha = new double[N];

this.errors = new double[N];

int numChanged = 0;

int examineAll = 1;

List<double> changes = new List<double>();

while (numChanged > 0 || examineAll > 0)

{

numChanged = 0;

if (examineAll > 0)

{

for (int i = 0; i < N; i++)

numChanged += ExamineExample(i);

}

else

{

for (int i = 0; i < N; i++)

if (alpha[i] != 0 && alpha[i] != c)

numChanged += ExamineExample(i);

}

if (examineAll == 1)

examineAll = 0;

else if (numChanged == 0)

examineAll = 1;

changes.Add(numChanged);

}

List<int> indices = new List<int>();

for (int i = 0; i < N; i++)

{

if (alpha[i] > 0) indices.Add(i);

}

int vectors = indices.Count;

machine.SupportVectors = new double[vectors][];

machine.Weights = new double[vectors];

for (int i = 0; i < vectors; i++)

{

int j = indices[i];

machine.SupportVectors[i] = inputs[j];

machine.Weights[i] = alpha[j] \* outputs[j];

}

machine.Threshold = -bias;

return changes.ToArray();

}

/// <summary>

/// Вычисление ошибки.

/// </summary>

/// <param name="inputs">Входные векторы.</param>

/// <param name="expectedOutputs">Ожидаемые выходные значения.</param>

public double ComputeError(double[][] inputs, int[] expectedOutputs)

{

int count = 0;

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

{

if (Math.Sign(Compute(inputs[i])) != Math.Sign(expectedOutputs[i]))

count++;

}

return (double)count / inputs.Length;

}

/// <summary>

/// Выбор множителей для оптимизации с использованием эвристики.

/// </summary>

private int ExamineExample(int i2)

{

double[] p2 = inputs[i2];

double y2 = outputs[i2];

double alph2 = alpha[i2];

double e2 = (alph2 > 0 && alph2 < c) ? errors[i2] : Compute(p2) - y2;

double r2 = y2 \* e2;

if (!(r2 < -tolerance && alph2 < c) && !(r2 > tolerance && alph2 > 0))

return 0;

int i1 = -1; double max = 0;

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

{

if (alpha[i] > 0 && alpha[i] < c)

{

double error1 = errors[i];

double aux = System.Math.Abs(e2 - error1);

if (aux > max)

{

max = aux;

i1 = i;

}

}

}

if (i1 >= 0 && TakeStep(i1, i2)) return 1;

int start = random.Next(inputs.Length);

for (i1 = start; i1 < inputs.Length; i1++)

{

if (alpha[i1] > 0 && alpha[i1] < c)

if (TakeStep(i1, i2)) return 1;

}

for (i1 = 0; i1 < start; i1++)

{

if (alpha[i1] > 0 && alpha[i1] < c)

if (TakeStep(i1, i2)) return 1;

}

start = random.Next(inputs.Length);

for (i1 = start; i1 < inputs.Length; i1++)

{

if (TakeStep(i1, i2)) return 1;

}

for (i1 = 0; i1 < start; i1++)

{

if (TakeStep(i1, i2)) return 1;

}

return 0;

}

/// <summary>

/// Аналитически решается задача оптимизации двух множителей Лагранжа.

/// </summary>

private bool TakeStep(int i1, int i2)

{

if (i1 == i2) return false;

double[] p1 = inputs[i1];

double alph1 = alpha[i1];

double y1 = outputs[i1];

double e1 = (alph1 > 0 && alph1 < c) ? errors[i1] : Compute(p1) - y1;

double[] p2 = inputs[i2];

double alph2 = alpha[i2];

double y2 = outputs[i2];

double e2 = (alph2 > 0 && alph2 < c) ? errors[i2] : Compute(p2) - y2;

double s = y1 \* y2;

double L, H;

if (y1 != y2)

{

L = Math.Max(0, alph2 - alph1);

H = Math.Min(c, c + alph2 - alph1);

}

else

{

L = Math.Max(0, alph2 + alph1 - c);

H = Math.Min(c, alph2 + alph1);

}

if (L == H) return false;

double k11, k22, k12, eta;

k11 = kernel.Function(p1, p1);

k12 = kernel.Function(p1, p2);

k22 = kernel.Function(p2, p2);

eta = k11 + k22 - 2.0 \* k12;

double a1, a2;

if (eta > 0)

{

a2 = alph2 - y2 \* (e2 - e1) / eta;

if (a2 < L) a2 = L;

else if (a2 > H) a2 = H;

}

else

{

double L1 = alph1 + s \* (alph2 - L);

double H1 = alph1 + s \* (alph2 - H);

double f1 = y1 \* (e1 + bias) - alph1 \* k11 - s \* alph2 \* k12;

double f2 = y2 \* (e2 + bias) - alph2 \* k22 - s \* alph1 \* k12;

double Lobj = -0.5 \* L1 \* L1 \* k11 - 0.5 \* L \* L \* k22 - s \* L \* L1 \* k12 - L1 \* f1 - L \* f2;

double Hobj = -0.5 \* H1 \* H1 \* k11 - 0.5 \* H \* H \* k22 - s \* H \* H1 \* k12 - H1 \* f1 - H \* f2;

if (Lobj > Hobj + epsilon) a2 = L;

else if (Lobj < Hobj - epsilon) a2 = H;

else a2 = alph2;

}

if (Math.Abs(a2 - alph2) < epsilon \* (a2 + alph2 + epsilon))

return false;

a1 = alph1 + s \* (alph2 - a2);

if (a1 < 0)

{

a2 += s \* a1;

a1 = 0;

}

else if (a1 > c)

{

double d = a1 - c;

a2 += s \* d;

a1 = c;

}

double b1 = 0, b2 = 0;

double new\_b = 0, delta\_b;

if (a1 > 0 && a1 < c)

{

new\_b = e1 + y1 \* (a1 - alph1) \* k11 + y2 \* (a2 - alph2) \* k12 + bias;

}

else

{

if (a2 > 0 && a2 < c)

{

new\_b = e2 + y1 \* (a1 - alph1) \* k12 + y2 \* (a2 - alph2) \* k22 + bias;

}

else

{

b1 = e1 + y1 \* (a1 - alph1) \* k11 + y2 \* (a2 - alph2) \* k12 + bias;

b2 = e2 + y1 \* (a1 - alph1) \* k12 + y2 \* (a2 - alph2) \* k22 + bias;

new\_b = (b1 + b2) / 2;

}

}

delta\_b = new\_b - bias;

bias = new\_b;

double t1 = y1 \* (a1 - alph1);

double t2 = y2 \* (a2 - alph2);

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

{

if (0 < alpha[i] && alpha[i] < c)

{

double[] point = inputs[i];

errors[i] +=

t1 \* kernel.Function(p1, point) +

t2 \* kernel.Function(p2, point) -

delta\_b;

}

}

errors[i1] = 0f;

errors[i2] = 0f;

alpha[i1] = a1;

alpha[i2] = a2;

return true;

}

/// <summary>

/// Вычисляет вывод SVM для заданной точки.

/// </summary>

private double Compute(double[] point)

{

double sum = -bias;

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

{

if (alpha[i] > 0)

sum += alpha[i] \* outputs[i] \* kernel.Function(inputs[i], point);

}

return sum;

}

private double ComputeComplexity()

{

double sum = 0.0;

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

sum += kernel.Function(inputs[i], inputs[i]);

return inputs.Length / sum;

}

}

}

**Листинг SupportVectorMachine.cs:**

using NeuralLibrary;

namespace SVMLib

{

/// <summary>

/// Класс, описывающий машину опорных векторов.

/// </summary>

public class SupportVectorMachine

{

private int inputCount;

private double[][] supportVectors;

private double[] weights;

private double threshold;

/// <summary>

/// Создание новой SVM.

/// </summary>

public SupportVectorMachine(int inputs)

{

this.inputCount = inputs;

}

/// <summary>

/// Получение количества входов.

/// </summary>

public int Inputs

{

get { return inputCount; }

}

/// <summary>

/// Получение и установка векторов, используемых данной машиной.

/// </summary>

public double[][] SupportVectors

{

get { return supportVectors; }

set { supportVectors = value; }

}

/// <summary>

/// Получение и установка весов, используемых данной машиной.

/// </summary>

public double[] Weights

{

get { return weights; }

set { weights = value; }

}

/// <summary>

/// Получение и установка порога (смещения), используемых данной машиной.

/// </summary>

public double Threshold

{

get { return threshold; }

set { threshold = value; }

}

/// <summary>

/// Вычисляет заданные входы для получения соответствующих выходов.

/// </summary>

/// <param name="input">Входной вектор.</param>

/// <returns>Выход для данного входа.</returns>

public virtual double Compute(double[] input)

{

double s = threshold;

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

{

double p = 0;

for (int j = 0; j < input.Length; j++)

p += supportVectors[i][j] \* input[j];

s += weights[i] \* p;

}

return s;

}

/// <summary>

/// Вычисляет заданные входы для получения соответствующих выходов.

/// </summary>

/// <param name="dataset">Входные вектора.</param>

/// <returns>Выходы для данных входов.</returns>

public double[] Compute(DataSet[] dataset)

{

double[] outputs = new double[dataset.Length];

double[][] inputs = new double[dataset.Length][];

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

{

inputs[i] = new double[dataset[i].InputsSignals.Length];

for (int j = 0; j < dataset[i].InputsSignals.Length; j++)

inputs[i][j] = dataset[i].InputsSignals[j];

}

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

outputs[i] = Compute(inputs[i]);

return outputs;

}

}

}

**Листинг SupportVectorMachine.cs:**

using NeuroLib;

namespace SVMLib

{

/// <summary>

/// Класс, описывающий машину опорных векторов

/// </summary>

public class SupportVectorMachine

{

private int inputCount;

private double[][] supportVectors;

private double[] weights;

private double threshold;

/// <summary>

/// Создание новой SVM.

/// </summary>

public SupportVectorMachine(int inputs)

{

this.inputCount = inputs;

}

/// <summary>

/// Получение количества входов.

/// </summary>

public int Inputs

{

get { return inputCount; }

}

/// <summary>

/// Получение и установка векторов, используемых данной машиной.

/// </summary>

public double[][] SupportVectors

{

get { return supportVectors; }

set { supportVectors = value; }

}

/// <summary>

/// Получение и установка весов, используемых данной машиной.

/// </summary>

public double[] Weights

{

get { return weights; }

set { weights = value; }

}

/// <summary>

/// Получение и установка порога (смещения), используемых данной машиной.

/// </summary>

public double Threshold

{

get { return threshold; }

set { threshold = value; }

}

/// <summary>

/// Вычисляет заданные входы для получения соответствующих выходов.

/// </summary>

/// <param name="input">Входной вектор.</param>

/// <returns>Выход для данного входа.</returns>

public virtual double Compute(double[] input)

{

double s = threshold;

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

{

double p = 0;

for (int j = 0; j < input.Length; j++)

p += supportVectors[i][j] \* input[j];

s += weights[i] \* p;

}

return s;

}

/// <summary>

/// Вычисляет заданные входы для получения соответствующих выходов.

/// </summary>

/// <param name="dataset">Входные вектора.</param>

/// <returns>Выходы для данных входов.</returns>

public double[] Compute(Data[] dataset)

{

double[] outputs = new double[dataset.Length];

double[][] inputs = new double[dataset.Length][];

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

{

inputs[i] = new double[dataset[i].InputsSignals.Length];

for (int j = 0; j < dataset[i].InputsSignals.Length; j++)

inputs[i][j] = dataset[i].InputsSignals[j];

}

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

outputs[i] = Compute(inputs[i]);

return outputs;

}

}

}

**Листинг DataSet.cs:**

namespace NeuroLib

{

public class Data

{

public double Expected { get; private set; }

public double[] InputsSignals { get; private set; }

public Data(double expected, double[] inputsSignals)

{

Expected = expected;

InputsSignals = inputsSignals;

}

}

}