**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.**

Выполнила: студент гр. ИТИ-21

Тесёлкин Б.А.

Принял: преподаватель

Карабчикова E.A.

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 24

Цифры отличия 0 от 9.

**Ход работы**

В ходе выполнения лабораторной работы была разработана машина опорных векторов с полиномиальным ядром, которое обеспечивает наилучший результат работы машины опорных векторов на подготовленных данных. Также была подготовлена обучающая выборка, которая используется для обучения машины опорных векторов.

Для реализации машины опорных векторов была создана библиотека классов SVMLibrary. В библиотеке классов был создан интерфейс IFunction, описывающий базовый функционал ядра без реализации функций. Также были реализованы наследуемые классы LinearFunctuin, описывающий базовое ядро машины и PolynomialFunction, описывающий полиномиальное ядро. Затем был создан класс SupportVectorMachine и наследуемый от него CoreSVM, описывающие функционал машины опорных векторов. Для обучающей выборки был создан класс Data, имеющий массив сигналов для подачи на входной слой и ожидаемая реакция машины. Затем был создан класс SequentialMinimalOptimization, описывающий алгоритм последовательной минимальной оптимизации.

После 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 1. График обучения SVM изображен на рисунке 2. Результат тестирования нейронной сети на бинаризированном изображение представлен на рисунке 3 и 4.

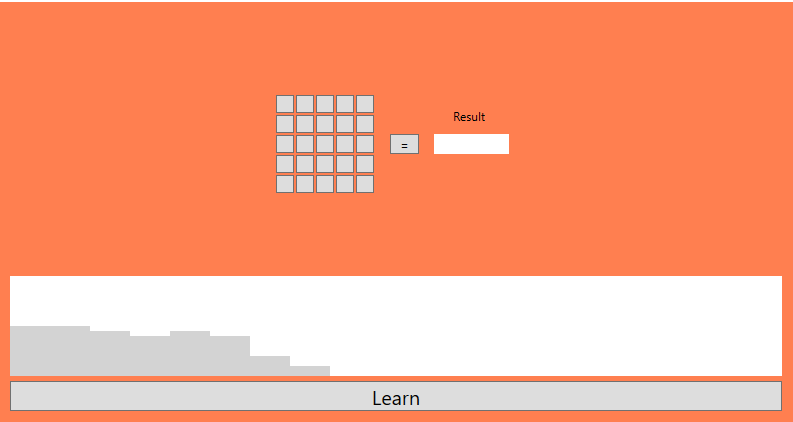


Рисунок 1 – Интерфейс приложения.



Рисунок 2 – График обучения машины опорных векторов.

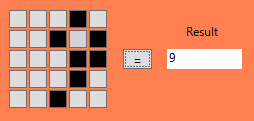


Рисунок 3 – Результаты тестирования на изображении.

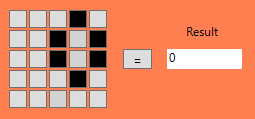


Рисунок 4 – Результаты тестирования на изображении.

Сопоставив график с графиками из лабораторной работы №2 видно, что машина опорных векторов с правильно подобранными параметрами обучается качественнее и быстрее. Из графика видно, что количество изменений алгоритма оптимизации довольно быстро стремится к 0. Это говорит о том, что 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 бинаризированных изображений.

**Приложение:**

**Листинг MainWindow.cs:**

using SVMLibrary;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

namespace NNInterface

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

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

Pen pen = new Pen(Brushes.Black, 2);

CoreSVM machine;

double[] statistic;

public MainWindow()

{

InitializeComponent();

AddData();

Learn();

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

Learn();

}

private void Learn()

{

machine = new CoreSVM(new PolynomicalFunction(2), 25);

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

statistic = learn.Run();

DisplayGraphic();

}

private void DisplayGraphic()

{

int i = 0;

graphic.Children.Clear();

foreach (var item in statistic)

{

Rectangle rectangle = new Rectangle();

rectangle.Fill = new SolidColorBrush(Colors.LightGray);

rectangle.Height = Math.Abs(item) \* 5;

rectangle.Width = 40;

rectangle.Margin = new Thickness(rectangle.Width \* i, graphic.Height - rectangle.Height, 0, 0);

graphic.Children.Add(rectangle);

i++;

}

}

private double[][] ReadFile(string path)

{

List<double[]> res = new List<double[]>();

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

{

string line;

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

{

res.Add(line.Split(',').Select(n => Convert.ToDouble(n)).ToArray());

}

}

return res.ToArray();

}

private void ChangeCollor\_Click(object sender, RoutedEventArgs e)

{

if (((Button)sender).Background == Brushes.Black)

{

((Button)sender).Background = Brushes.LightGray;

}

else

{

((Button)sender).Background = Brushes.Black;

}

}

private void Result\_Button(object sender, RoutedEventArgs e)

{

double[] startData = new double[25];

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

{

if (((Button)userInput.Children[i]).Background == Brushes.Black)

{

startData[i] = 1;

}

else

{

startData[i] = 0;

}

}

Data data = new Data(0, startData);

double output = machine.Compute(data.Inputs);

string res = output > 0 ? "9" : "0";

result.Text = res;

}

private void AddData()

{

double[][] trainData = ReadFile(@"C:\Users\saymo\source\repos\NN\_train.txt");

double[] result = ReadFile(@"C:\Users\saymo\source\repos\NN\_results.txt")[0];

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

{

if(result[i] == 1)

{

data.Add(new Data(1, trainData[i]));

}

else

{

data.Add(new Data(-1, trainData[i]));

}

}

}

}

}

**Листинг IFunction.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SVMLibrary

{

public interface IFunction

{

double Function(double[] v, double[] inputs);

}

}

**Листинг LinearFunction.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SVMLibrary

{

public class LinearFunction : IFunction

{

public double Constant { get; set; }

public LinearFunction(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 PolynomialFunction.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SVMLibrary

{

public class PolynomicalFunction : IFunction

{

private int degree;

public int Degree

{

get { return degree; }

set

{

if(value <= 0)

{

throw new ArgumentOutOfRangeException("value", "Degree must be greater than 0");

}

degree = value;

}

}

public double Constant { get; set; }

public PolynomicalFunction(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 SupportVectorMachine.cs:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SVMLibrary

{

public class SupportVectorMachine

{

public int Inputs { get; private set; }

public double[][] SupportVectors { get; set; }

public double[] Weights { get; set; }

public double Threshold { get; set; }

public SupportVectorMachine(int inputsNum)

{

Inputs = inputsNum;

}

public virtual double Compute(double[] input)

{

double sum = 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];

}

sum += Weights[i] \* p;

}

return sum;

}

public double[] Compute(Data[] data)

{

double[] output = new double[data.Length];

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

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

{

inputs[i] = new double[data[i].Inputs.Length];

for (int j = 0; j < data[i].Inputs.Length; j++)

{

inputs[i][j] = data[i].Inputs[j];

}

}

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

{

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

}

return output;

}

}

}

**Листинг SequentialMinimalOptimization.cs:**

using NeuralLibrary;

using System;

using System.Collections.Generic;

namespace SVMLib

{

public class SequentialMinimalOptimization : ISupportVectorMachineLearning

{

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 SequentialMinimalOptimization(SupportVectorMachine machine, DataSet[] 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;

KernelSupportVectorMachine ksvm = machine as KernelSupportVectorMachine;

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

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 CoreSVM.cs:**

namespace SVMLib

{

public class KernelSupportVectorMachine : SupportVectorMachine

{

/// <summary>

/// Создание новой SVM.

/// </summary>

/// <param name="kernel">Ядро.</param>

/// <param name="inputs">Количество входов.</param>

public KernelSupportVectorMachine(IKernel kernel, int inputs) : base(inputs)

{

Kernel = kernel;

}

/// <summary>

/// Получение или задание ядра.

/// </summary>

public IKernel Kernel { get; set; }

/// <summary>

/// Вычисляет значение для данного входного вектора.

/// </summary>

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

/// <returns>Результат для данного вектора.</returns>

public override double Compute(double[] inputs)

{

double s = Threshold;

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

s += Weights[i] \* Kernel.Function(SupportVectors[i], inputs);

return s;

}

}

}

**Листинг Data.cs:**

namespace NeuralLibrary

{

public class DataSet

{

public double Expected { get; private set; }

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

public DataSet(double expected, double[] inputsSignals)

{

Expected = expected;

InputsSignals = inputsSignals;

}

}

}