Министерство транспорта Российской Федерации
Федеральное агентство железнодорожного транспорта

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Дальневосточный государственный университет путей сообщения»

Кафедра «Вычислительная техника и компьютерная графика»

РЕАЛИЗАЦИЯ КЛАССИФИКАТОРА НА ОСНОВЕ МАШИНЫ ОПОРНЫХ ВЕКТОРОВ

Лабораторная работа №4

ЛР 09.04.01.МРО.08.04.МО921ИВС

Выполнил
студент гр. МО921ИВС _____ А.Ю. Панченко

Проверил
доцент, к.ф.-м.н. _____ Ю.В. Пономарчук

Хабаровск 2025

Цель работы: изучение теоретических основ и экспериментальное исследование метода построения классификатора на основе алгоритма SVM.

1 УСЛОВИЕ ЗАДАЧИ

Выполнить реализацию алгоритма SVM на основе обучающих выборок (по вариантам).

Реализовать классификатор – машину опорных векторов по заданию в файле SVM\R\lab5.pdf

Исходные данные приведены в папке SVM\R\lr5 – по вариантам (5 вариантов)

Инструмент реализации: произволен (можно написать собственную программу, использовать библиотеки Python, R, функции Matlab, SPSS).

2 ХОД РЕШЕНИЯ

Реализация классификатора на основе машины опорных векторов произведена средствами библиотеки Accord.

SVM – это алгоритм классификации, который ищет оптимальную гиперплоскость, разделяющую данные разных классов с максимальным зазором. Опорными векторами называются точки данных, которые находятся ближе всего к разделяющей гиперплоскости и определяют её положение.

Основная идея SVM заключается в поиске разделяющей гиперплоскости, которая:

– корректно классифицирует все точки обучающей выборки (или минимизирует ошибку);

– максимизирует расстояние между ближайшими точками разных классов.

По варианту (3) даны файлы с обучающей и тестовой выборкой. Формат файлов – табличный. Столбцы: признак X1, признак X2, метка (цвет, красный или зеленый, бинарная классификация).

В листинге 1 – код для загрузки данных.

Листинг 1 – Загрузка данных

```
public static List<DataPoint> LoadData(string path)
{
    var result = new List<DataPoint>();

    foreach (var line in File.ReadLines(path))
    {
        var parts = line.Split('\t');

        // Пропускаем заголовок или некорректные строки
        if (parts.Length < 3 || parts[0] == "X1")
        {
            continue;
        }

        // Создаем объект DataPoint из строки
        var dataPoint = new DataPoint
        {
            X1 = double.Parse(parts[1],
CultureInfo.InvariantCulture),
            X2 = double.Parse(parts[2],
CultureInfo.InvariantCulture),
            Color = parts[3].Trim().ToLower()
        };

        result.Add(dataPoint);
    }

    return result;
}
```

Для обучения модели средствами библиотеки метки были закодированы бинарными значениями, а данные – отмасштабированы для использования SVM (листинг 2).

Листинг 2 – Кодирование меток и масштабирование данных

```
// Словарь для преобразования строковых меток в числовые классы
var labelMap = new Dictionary<string, int> { ["red"] = 0,
["green"] = 1 };

// Заполняем обучающие данные: признаки и метки классов
for (var i = 0; i < trainSize; i++)
{
    trainInputs[i, 0] = trainList[i].X1;
```

```
    trainInputs[i, 1] = trainList[i].X2;
    trainOutputs[i] = labelMap[trainList[i].Color];
}

// Заполняем тестовые данные: признаки и метки классов
for (var i = 0; i < testSize; i++)
{
    testInputs[i, 0] = testList[i].X1;
    testInputs[i, 1] = testList[i].X2;
    testOutputs[i] = labelMap[testList[i].Color];
}

// Масштабируем признаки с помощью Z-преобразования
(нормализация по столбцам)
var means = trainInputs.Mean(0);                    // Среднее
по каждому признаку
var stdDevs = trainInputs.StandardDeviation();      //
Стандартное отклонение

trainInputs = trainInputs.ZScores(means, stdDevs);  //
Масштабируем обучающую выборку
testInputs = testInputs.ZScores(means, stdDevs);    //
Масштабируем тестовую выборку теми же параметрами
```

Обучение модели и последующее её использование для классификации элементов тестового набора представлены в листинге 3.

Листинг 3 – Обучение модели и получение классов на тестовом наборе

```
// Настройка обучающего алгоритма SVM с гауссовским (RBF) ядром
var teacher = new SequentialMinimalOptimization<Gaussian>
{
    Kernel = new Gaussian(1.0),     // Параметр ширины ядра (σ)
    Complexity = 1.0                // Параметр регуляризации (C)
};

// Accord требует jagged-массивы, преобразуем из 2D
var jaggedTrainInputs = trainInputs.ToJagged();

// Обучаем модель SVM
var svm = teacher.Learn(jaggedTrainInputs, trainOutputs);

// Предсказываем классы на тестовой выборке
var jaggedTestInputs = testInputs.ToJagged();
var boolPredictions = svm.Decide(jaggedTestInputs);          //
true/false
var predictions = boolPredictions.Select(b => b ? 1 :
0).ToArray(); // Преобразуем в 0/1
```

```
// Вычисляем точность классификации
var accuracy = predictions.Zip(testOutputs, (p, t) => p == t ?
1.0 : 0.0).Average();
Console.WriteLine($"Accuracy: {accuracy:P2}"); // Выводим
точность в формате процентов
```

Радиальная базисная функция

Когда данные линейно неразделимы в исходном пространстве, SVM использует «ядерный трюк» – неявное отображение данных в пространство более высокой размерности, где разделение становится возможным.

Радиально-базисная функция (гауссово ядро) имеет вид:

$$K(x, y) = e^{\left(-\gamma||x-y||^2\right)}$$

Где x, y – векторы данных, γ – параметр, определяющий влияние отдельных тренировочных примеров.

Радиально-базисное ядро эффективно при нелинейных зависимостях, случаях, когда классы не разделяются линейно, неизвестной структуре данных (хороший выбор по умолчанию).

Параметр регуляризации C контролирует компромисс между максимизацией ширины разделяющей полосы и минимизацией ошибки классификации на обучающих данных

Маленькие значения C отдают приоритет широкой разделяющей полосе, даже если это приведёт к ошибкам классификации (более простая модель, меньше переобучения).

Большие значения C показывают стремление классифицировать все точки правильно, жертвуя шириной полосы (более сложная модель, риск переобучения).

Параметр γ определяет влияние каждого тренировочного примера.

Маленькие значения γ: большой радиус влияния, гладкая граница решения

Большие значения γ: малый радиус влияния, более сложная граница решения

Опция $scale$ означает, что γ вычисляется автоматически по формуле:

$$\gamma = \frac{1}{n_f \times X_{var}}$$

Где n_f – количество признаков, X_{var} — дисперсия входных данных

Это делает выбор γ адаптивным к масштабу и распределению данных.

При вызове метода fit происходит:

– решение задачи квадратичной оптимизации для поиска оптимальных коэффициентов

– определение опорных векторов — точек, ближайших к разделяющей гиперплоскости

– построение модели принятия решений на основе найденных коэффициентов

В листинге 4 – построение графика с результатами классификации и матрицы ошибок.

Листинг 4 – Визуализация классификации данных тестового набора

```
public static void ShowConfusionMatrix(int[] actual, int[]
predicted, string filename)
{
    var plt = new Plot();

    // Заполняем матрицу ошибок 2x2: [истинный класс,
предсказанный класс]
    var confusion = new int[2, 2];

    for (int i = 0; i < actual.Length; i++)
    {
        confusion[actual[i], predicted[i]]++;
    }

    var data = new double[2, 2];

    // Переворачиваем матрицу по вертикали для правильного
визуального порядка
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
```

```
        {
            data[1 - i, j] = confusion[i, j];
        }
    }

    // Добавляем тепловую карту (Heatmap)
    var hm = plt.Add.Heatmap(data);
    hm.Colormap = new Blues();

    // Подписываем значения в ячейках
    for (int i = 0; i < 2; i++)
    {
        for (int j = 0; j < 2; j++)
        {
            var value = data[i, j];
            var label = plt.Add.Text(value.ToString(), j, i);
            label.Alignment = Alignment.MiddleCenter;
            label.LabelFontSize = 20;
            label.LabelBold = true;
            label.LabelFontColor = value < 30 ? Colors.Black :
Colors.White;
        }
    }

    // Добавляем цветовую шкалу
    var cb = plt.Add.ColorBar(hm);
    cb.Label = "Количество";

    // Подписи осей
    plt.Title("Матрица ошибок", size: 24);
    plt.XLabel("Предсказанный класс", size: 18);
    plt.YLabel("Истинный класс", size: 18);

    // Настраиваем метки осей вручную (0 — red, 1 — green)
    plt.Axes.Bottom.TickGenerator = new NumericManual(
        new[] { 0, 1d }, new[] { "red", "green" });
    plt.Axes.Left.TickGenerator = new NumericManual(
        new[] { 0, 1d }, new[] { "green", "red" }); //
перевернуто для визуального соответствия

    // Устанавливаем квадратную сетку
    plt.Axes.SetLimits(-0.5, 1.5, -0.5, 1.5);

    // Сохраняем график
    SavePlotToPng(plt, filename);
}
```

На рисунке 1 представлен результат классификации данных из тестового набора.

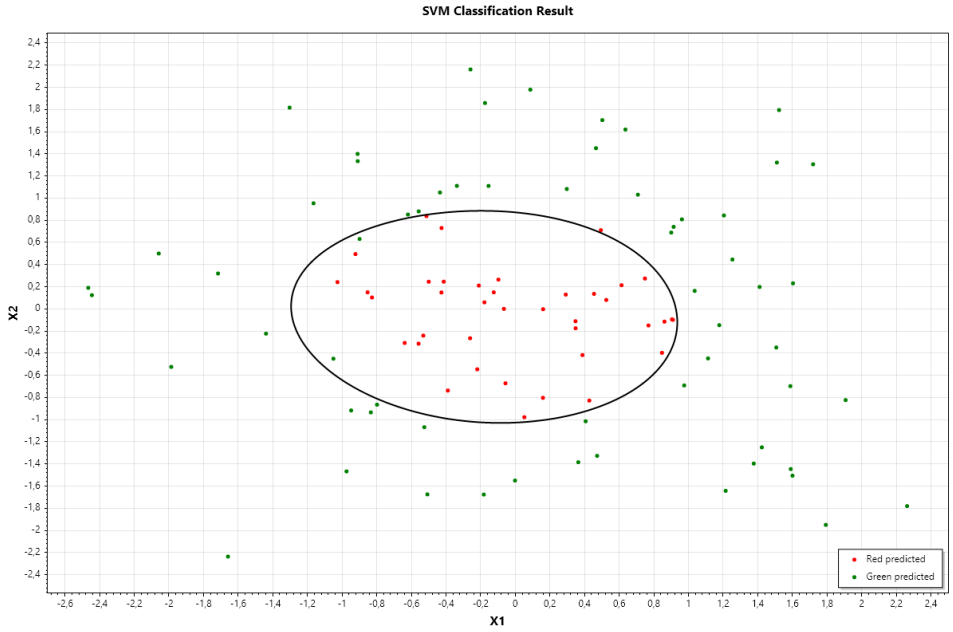

Рисунок 1 – Результат классификации точек тестового набора

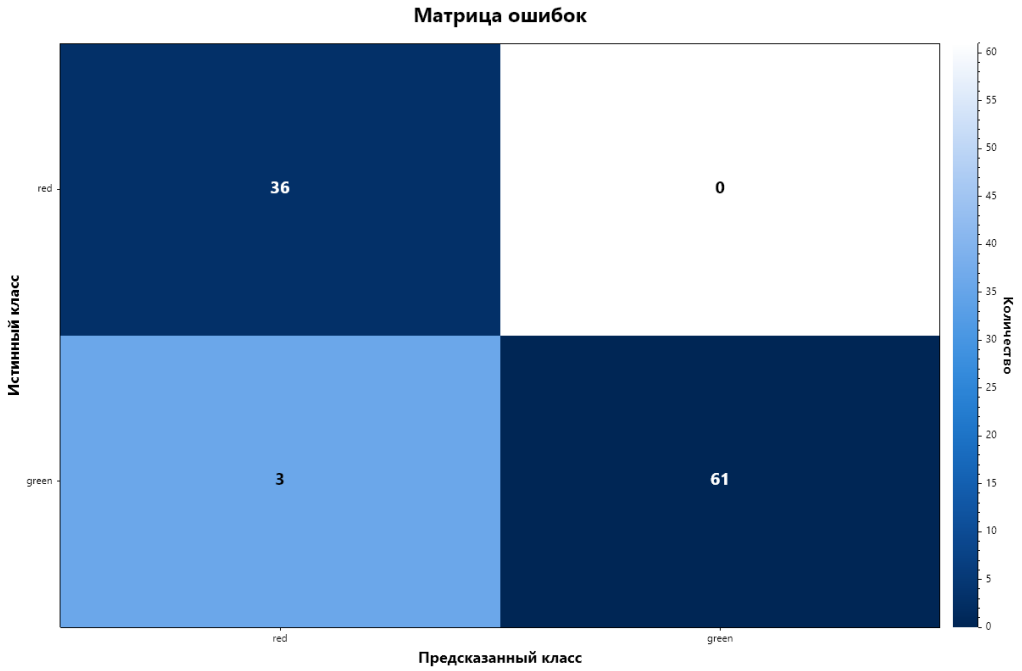

Рисунок 2 – Матрица ошибок

Вывод: в ходе работы была выполнена реализация и экспериментальное исследование алгоритма SVM для построения классификатора на основе предоставленных обучающих выборок.