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 ЛИНЕЙНОГО КЛАССИФИКАТОРА

Лабораторная работа №3

ЛР 09.04.01.МРО.08.03.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 линейного классификатора для распознавания образов.

1 УСЛОВИЕ ЗАДАЧИ

Выполнить реализацию алгоритма линейного классификатора на основе обучающих выборок, сгенерированных в работе 2.

1. Разделить каждый из наборов данных, сгенерированных в работе 2, на 2 части: **обучающую выборку** – 150 случайных двумерных векторов из нормального распределения и **контрольную** – оставшиеся 50 значений.

2. На основании полученных **обучающих выборок** построить линейные классификаторы, максимизирующие критерий Фишера, для классов Ω_1 и Ω_2, Ω_1 и Ω_3, Ω_2 и Ω_3 двумерных нормально распределенных векторов признаков соответственно. Сравнить качество полученного классификатора с байесовским классификатором. Использовать формулы (14, 15 – 16) раздела 1.3 [Коломиец, ч.3].

3. На основании **контрольных выборок** оценить эффективность классификации, как отношение правильно классифицированных векторов к общему количеству N=150. Оценить ошибку классификации.

4. Отобразить на графике исходные наборы данных и дискриминантные функции (по аналогии с рисунком 4 [Коломиец, ч.2]).

2 ХОД РЕШЕНИЯ

Задача классификации выполнялась с использованием алгоритма линейной классификации Фишера.

В качестве критерия для определения оптимального разделения классов Ω_1 и Ω_0 выбирается функция $f(m_0, m_1, \sigma_0^2, \sigma_1^2)$ вида (Коломиец 1.3, 14):

$$f = \frac{(m_1 - m_0)^2}{\sigma_1^2 + \sigma_0^2}$$

Критерий (14) называется критерием Фишера и представляет собой меру отличия значений линейной дискриминантной функции в классах Ω_1 и Ω_0. Для

наилучшего разделения классов необходимо определить $\overline{W}$ и w_N, которые доставляли бы этому критерию максимум. Получаемый при этом линейный классификатор называется классификатором Фишера.

Выражение для вектора весовых коэффициентов дискриминантной функции (Коломиец 1.3, 15):

$$\overline{W} = \left(\frac{1}{2}(B_1 + B_2)\right)^{-1} (\overline{M}_1 - \overline{M}_0).$$

Выражение для порогового значения дискриминантной функции w_N: (Коломиец 1.3, 16):

$$w_N = -\frac{(\overline{M}_1 - \overline{M}_0)^T \left(\frac{1}{2}(B_1 + B_0)\right)^{-1} (\sigma_1^2 \overline{M}_0 + \sigma_0^2 \overline{M}_1)}{\sigma_1^2 + \sigma_0^2}.$$

Разделение сгенерированного при помощи ранее сгенерированного набора данных на тренировочный и тестовый наборы производится при помощи функции, приведенной в листинге 1.

Листинг 1 – Разделение набора данных на обучающий и тестовый

```
internal static class SamplesGenerator{
    private static readonly Random random = new(2356);

    public static Matrix<double> GenerateSamples(Vector<double>
mean, Matrix<double> cov, int count){
        var samples = DenseMatrix.Create(count, mean.Count, (i,
j) => 0.0);

        for (var i = 0; i < count; i++){
            var sample = SampleMultivariateNormal(mean, cov);
            samples.SetRow(i, sample);
        }

        return samples;
    }

    private static Vector<double>
SampleMultivariateNormal(Vector<double> mean, Matrix<double>
cov)
    {
        var chol = cov.Cholesky().Factor;
```

```
        var stdNormal = Vector<double>.Build.Dense(mean.Count, _
=> Normal.Sample(random, 0, 1));
        return mean + chol * stdNormal;
    }
}
…
var train1 = new ClassData(SamplesGenerator.GenerateSamples(M1,
B1, 150));
var train2 = new ClassData(SamplesGenerator.GenerateSamples(M2,
B2, 150));
var train3 = new ClassData(SamplesGenerator.GenerateSamples(M3,
B3, 150));

var test1 = new ClassData(SamplesGenerator.GenerateSamples(M1,
B1, 50));
var test2 = new ClassData(SamplesGenerator.GenerateSamples(M2,
B2, 50));
var test3 = new ClassData(SamplesGenerator.GenerateSamples(M3,
B3, 50));
```

Структура данных для каждого класса и классификатора определены в виде пользовательских типов данных (листинг 2).

Листинг 2 – Определение пользовательских типов данных

```
internal class ClassData
{
    public ClassData(Matrix<double> samples)
    {
        Samples = samples;
        Mean = samples.ColumnSums() / samples.RowCount;
        Covariance = CovarianceMatrix(samples);
    }

    public Matrix<double> Samples { get; }
    public Vector<double> Mean { get; }
    public Matrix<double> Covariance { get; }

    private Matrix<double> CovarianceMatrix(Matrix<double> data)
    {
        var mean = data.ColumnSums() / data.RowCount;
        var centered = data - DenseMatrix.Create(data.RowCount,
data.ColumnCount, (i, j) => mean[j]);
        return centered.TransposeThisAndMultiply(centered) /
(data.RowCount - 1);
    }
}
```

```
internal class ClassifierParameters
{
    public ClassifierParameters(Vector<double> weights, double
threshold, (int, int) indices)
    {
        Weights = weights;
        Threshold = threshold;
        ClassIndices = indices;
    }

    public Vector<double> Weights { get; }
    public double Threshold { get; }
    public (int, int) ClassIndices { get; }
}
```

Класс ClassData содержит сами выборки данных в виде массива матрицы, а также вычисляет и хранит среднее значение признаков и ковариационную матрицу для этого класса.

Класс ClassifierParameters используется для хранения параметров, необходимых для работы классификатора, таких как вектор весов, пороговое значение и индексы классов, которые классификатор различает. Эти параметры определяют, как классификатор будет принимать решения о принадлежности объектов к различным классам.

Полная реализация обучения линейного классификатора (классификатора Фишера) представлено в листинге 3.

Листинг 3 – Реализация обучения классификатора Фишера

```
public static ClassifierParameters Train(ClassData class1,
ClassData class2, int idx1, int idx2)
{
    // Мат ожидания
    var M0 = class1.Mean;
    var M1 = class2.Mean;

    // Ковариационные матрицы
    var B0 = class1.Covariance;
    var B1 = class2.Covariance;

    // Весовой вектор по формуле 15
    var BCombined = (B0 + B1) * 0.5;
    var BInv = BCombined.Inverse();
```

```
    var weights = BInv * (M1 - M0);

    // Применение весовых коэф и нахождение дисперсии
    var proj0 = class1.Samples * weights;
    var proj1 = class2.Samples * weights;

    var sigma0Sq = Variance(proj0);
    var sigma1Sq = Variance(proj1);

    // Пороговое значение по формуле 16
    var weightedMeans = sigma1Sq * M0 + sigma0Sq * M1;
    var numerator = (M1 - M0).ToRowMatrix() * BInv *
weightedMeans;
    var denominator = sigma1Sq + sigma0Sq;

    var threshold = -numerator[0] / denominator;

    return new ClassifierParameters(weights, threshold, (idx1,
idx2));
}
```

Сначала вычисляются основные характеристики каждого класса: матожидание и ковариационная матрица (описывающая взаимосвязь между признаками внутри класса).

Затем на их основе определяется вектор весов. Он определяет направление в пространстве признаков, в котором различие между классами становится наиболее выраженным.

После этого, данные каждого класса проецируются на найденный вектор весов. Эта проекция позволяет увидеть, как данные распределяются вдоль направления, которое максимально разделяет классы.

Далее вычисляются дисперсии проекций для каждого класса, показывающие разброс данных вдоль этого направления.

На основе полученных проекций и их дисперсий вычисляется критерий Фишера, который служит мерой качества разделения классов. Он учитывает как разницу между средними значениями проекций (чем больше, тем лучше), так и их дисперсии (чем меньше, тем лучше).

Последний этап – определить пороговое значение, которое будет

использоваться для принятия решения о принадлежности нового объекта к одному из классов.

Метод для проверки принадлежности к одному из классов представлен в листинге 4.

Листинг 4 – Метод классификатора

```
public static int Predict(Vector<double> sample,
List<ClassifierParameters> classifiers)
{
    var votes = new int[3];

    foreach (var clf in classifiers)
    {
        var decision = sample.DotProduct(clf.Weights) +
clf.Threshold;
        if (decision > 0)
            votes[clf.ClassIndices.Item2]++;
        else
            votes[clf.ClassIndices.Item1]++;
    }

    return Array.IndexOf(votes, votes.Max());
}
```

Функция Predict определяет класс объекта, используя ансамбль классификаторов Фишера: для каждого классификатора вычисляется значение дискриминантной функции, и в зависимости от знака этого значения происходит «голосование» за один из двух классов, которые классификатор различает; после обработки всех классификаторов, выбирается класс, набравший наибольшее количество баллов, и возвращается как предсказанный класс объекта.

Программный код для проверки точности на тестовом наборе представлен в листинге 5.

Листинг 5 – Оценка точности классификатора

```
public static (double accuracy, double error)
Evaluate(List<ClassifierParameters> classifiers,
    List<ClassData> testClasses){
    var correct = 0;
    var total = 0;

    for (var i = 0; i < testClasses.Count; i++){
```

```
        var samples = testClasses[i].Samples;
        for (var j = 0; j < samples.RowCount; j++){
            var sample = samples.Row(j);
            var predicted = Predict(sample, classifiers);
            if (predicted == i) correct++;
            total++;
        }
    }

    var accuracy = (double)correct / total;
    var error = 1.0 - accuracy;
    return (accuracy, error);
}
```

Итоговый программный код для построения визуализации дискриминаторных функций и точек тестового набора показан в листинге 6.

Листинг 6 – Точка входа программы

```
var M1 = Vector.Build.DenseOfArray(new[] { xM1, yM1 });
var B1 = DenseMatrix.OfArray(new[,] { { B1varX, B1covXY }, {
B1covYX, B1varY } });
var M2 = Vector.Build.DenseOfArray(new[] { xM2, yM2 });
var B2 = DenseMatrix.OfArray(new[,] { { B2varX, B2covXY }, {
B2covYX, B2varY } });
var M3 = Vector.Build.DenseOfArray(new[] { xM3, yM3 });
var B3 = DenseMatrix.OfArray(new[,] { { B3varX, B3covXY }, {
B3covYX, B3varY } });

var train1 = new ClassData(SamplesGenerator.GenerateSamples(M1,
B1, 150));
var train2 = new ClassData(SamplesGenerator.GenerateSamples(M2,
B2, 150));
var train3 = new ClassData(SamplesGenerator.GenerateSamples(M3,
B3, 150));

var test1 = new ClassData(SamplesGenerator.GenerateSamples(M1,
B1, 50));
var test2 = new ClassData(SamplesGenerator.GenerateSamples(M2,
B2, 50));
var test3 = new ClassData(SamplesGenerator.GenerateSamples(M3,
B3, 50));

var clf12 = FisherClassifier.Train(train1, train2, 0, 1);
var clf13 = FisherClassifier.Train(train1, train3, 0, 2);
var clf23 = FisherClassifier.Train(train2, train3, 1, 2);

var classifiers = new List<ClassifierParameters> { clf12, clf13,
```

```
clf23 };
var testSet = new List<ClassData> { test1, test2, test3 };
var (fisherAccuracy, fisherError) =
FisherClassifier.Evaluate(classifiers, testSet);
        foreach (ClassifierParameters clf in classifiers){
            int idx1 = clf.ClassIndices.Item1;
            int idx2 = clf.ClassIndices.Item2;
            var w = clf.Weights;
            double threshold = clf.Threshold;
            Console.WriteLine($"Классификатор между классами
{idx1 + 1} и {idx2 + 1}:");
            Console.WriteLine($"  Вектор весов: [{w[0]:F4},
{w[1]:F4}]");
            Console.WriteLine($"  Норма вектора весов:
{w.L2Norm():F4}");
            Console.WriteLine($"  Порог (threshold):
{threshold:F4}");
            Console.WriteLine();}
Console.WriteLine($"Fisher effectiveness: {fisherAccuracy *
100:0.00}%");
Console.WriteLine($"Fisher error: {fisherError * 100:0.00}%");
PlotHelper.VisualizeFisher(testSet, classifiers);
```

На рисунке 1 представлена визуализация с использованием записной библиотеки ScottPlot.

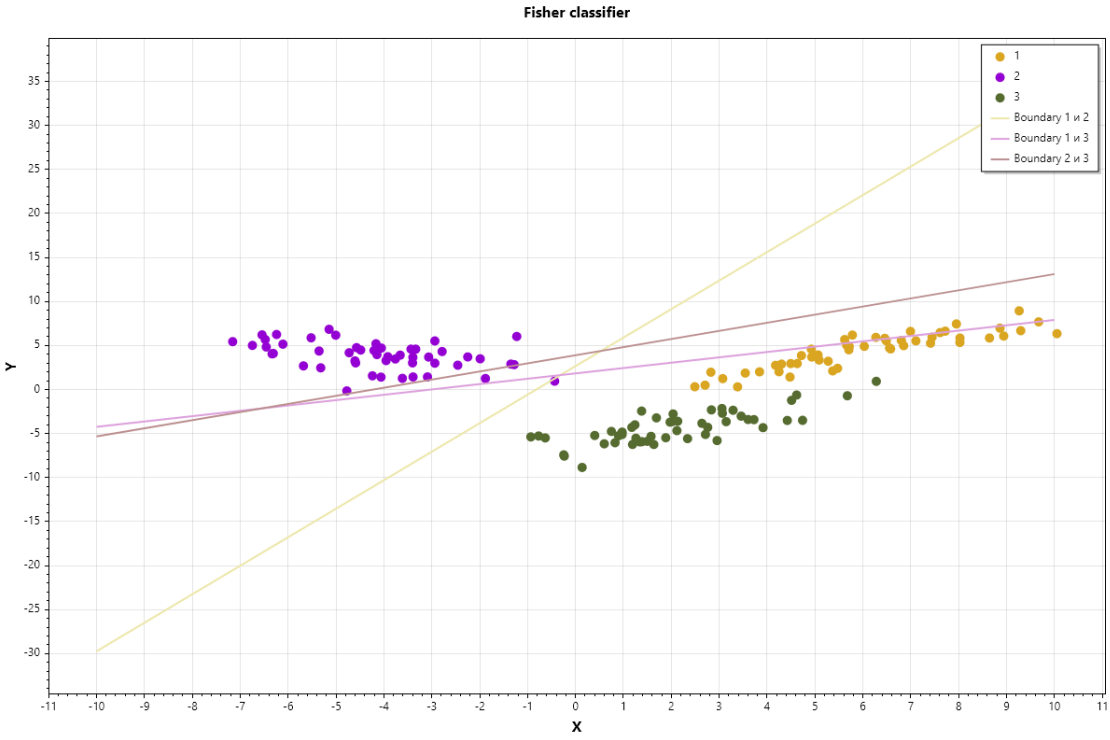

Рисунок 1 – Дискриминантые функции и тестовые наборы

В сравнении с наивным байесовским классификатором точность

Ниже представлена оценка точности классификации на фиксированном генеративном наборе:

Эффективность: 0.9933 (149/150)

Ошибка: 0.0067

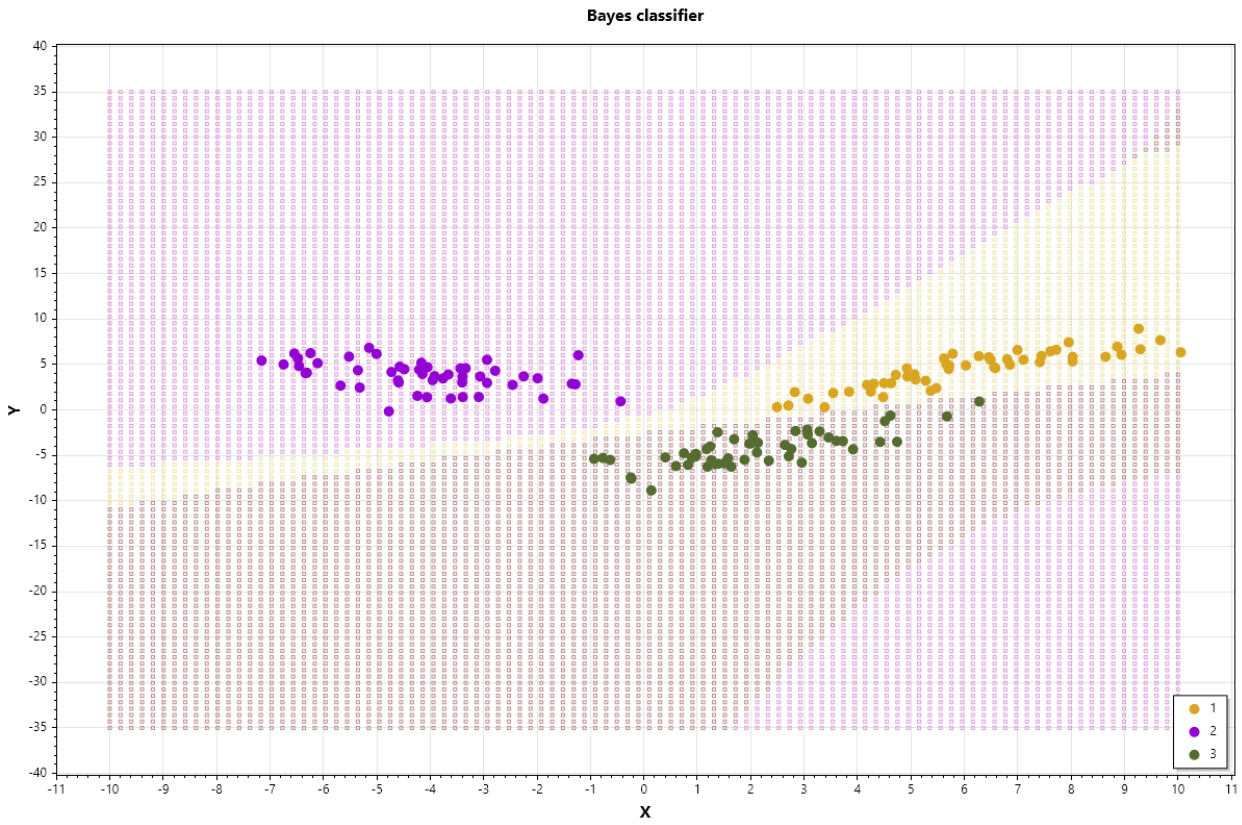

Рисунок 2 – Результат работы Байесовского классификатора

Эффективность классификации: 100,00 %

Ошибка классификации: 0,00 %

Правильно классифицировано: 150 из 150

Неправильно классифицировано: 0 из 150

В сравнении с наивным классификатором Байеса точность незначительно меньше на использованном наборе данных.

Вывод: В работе исследовался линейный классификатор Фишера для распознавания образов на основе сгенерированных выборок, эффективность оценивалась на контрольной выборке.