

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 НАИВНОГО БАЙЕСОВСКОГО КЛАССИФИКАТОРА ОБРАЗОВ

Лабораторная работа №2

ЛР 09.04.01.МРО.08.02.МО921ИВС

Выполнил		
студент гр. МО921ИВС	_____	А.Ю. Панченко
Проверил		
доцент, к.ф.-м.н.	_____	Ю.В. Пономарчук

Цель работы: изучение теоретических основ и экспериментальное исследование метода построения байесовского классификатора для распознавания образов.

1 УСЛОВИЕ ЗАДАЧИ

Выполнить реализацию алгоритма наивного байесовского классификатора на основе обучающих выборок, сгенерированных в работе 1.

2 ХОД РЕШЕНИЯ

Разделим 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 (листинг 1).

Листинг 1 – Метод для разбиения на обучающую и тестовую выборки

```
(double[] xTrain, double[] yTrain, double[] xTest, double[] yTest) SplitData(double[] x, double[] y) =>
    (x.Take(150).ToArray(), y.Take(150).ToArray(),
    x.Skip(150).ToArray(), y.Skip(150).ToArray());
```

На основании полученных обучающих выборок найти точечные оценки параметров нормального закона для каждого из распределений: оценки математических ожиданий M_1, M_2, M_3 , размерности 2×1 , и оценки ковариационных матриц B_1, B_2, B_3 (листинг 2).

Оценка максимального правдоподобия математического ожидания и ковариационной матрицы производятся по формулам:

$$\widehat{M} = \frac{1}{N} \sum_{i=1}^N \bar{x}_i, \quad \widehat{B} = \frac{1}{N} \sum_{i=1}^N (\bar{x}_i - \widehat{M})(\bar{x}_i - \widehat{M})^T = \frac{1}{N} \sum_{i=1}^N \bar{x}_i \bar{x}_i^T - \widehat{M} \widehat{M}^T.$$

Листинг 2 – Получение параметров

```
public static (double[][] ClassMeans, double[][,]
ClassCovariances, double[] ClassPriors)
    Train(double[] trainDataX, double[] trainDataY, int[]
labels, int numClasses)
{
    var totalSamples = trainDataX.Length;

    // Количество точек в каждом классе
```

```

        var classSampleCounts = GetClassSampleCounts(labels,
numClasses, totalSamples);

        // Оценки математических ожиданий
        var classMeans = GetClassMeans(labels, numClasses,
totalSamples, classSampleCounts, trainDataX, trainDataY);

        // Оценки ковариационных матриц
        var classCovariances = GetClassCovariances(labels,
numClasses, totalSamples, classSampleCounts, trainDataX,
trainDataY, classMeans);

        // Априорные вероятности
        var classPriors = GetClassPriors(numClasses,
classSampleCounts, totalSamples);

        return (classMeans, classCovariances, classPriors);
    }

private static int[] GetClassSampleCounts(int[] labels, int
numClasses, int totalSamples)
{
    var classSampleCounts = new int[numClasses];

    for (var i = 0; i < totalSamples; i++)
    {
        var classLabel = labels[i];

        classSampleCounts[classLabel]++;
    }

    return classSampleCounts;
}

private static double[][] GetClassMeans(int[] labels, int
numClasses, int totalSamples, int[] classSampleCounts,
double[] trainDataX, double[] trainDataY)
{
    var classMeans = new double[numClasses][];

    // Оценка математических ожиданий для каждого класса
    for (var c = 0; c < numClasses; c++)
    {
        classMeans[c] = new double[2]; // для x и y
        classMeans[c][0] = 0; // x
        classMeans[c][1] = 0; // y
    }

    // Суммируются значения координат для каждого класса
    for (var i = 0; i < totalSamples; i++)
    {

```

```

        var classLabel = labels[i];
        classMeans[classLabel][0] += trainDataX[i]; // x
        classMeans[classLabel][1] += trainDataY[i]; // y
    }

    // Деление на количество точек в каждом классе
    for (var c = 0; c < numClasses; c++)
        if (classSampleCounts[c] > 0)
        {
            classMeans[c][0] /= classSampleCounts[c]; // x
            classMeans[c][1] /= classSampleCounts[c]; // y
        }

    return classMeans;
}

private static double[][, ] GetClassCovariances(int[] labels, int
numClasses, int totalSamples,
    int[] classSampleCounts, double[] trainDataX, double[]
trainDataY, double[][,] classMeans)
{
    var classCovariances = new double[numClasses][, ];

    // Оценка ковариационных матриц для каждого класса
    for (var c = 0; c < numClasses; c++)
    {
        classCovariances[c] = new double[2, 2];
        classCovariances[c][0, 0] = 0; // xx
        classCovariances[c][0, 1] = 0; // xy
        classCovariances[c][1, 0] = 0; // yx
        classCovariances[c][1, 1] = 0; // yy
    }

    // Суммируются произведения отклонений от средних
    for (var i = 0; i < totalSamples; i++)
    {
        var classLabel = labels[i];

        var dx = trainDataX[i] - classMeans[classLabel][0];
        var dy = trainDataY[i] - classMeans[classLabel][1];

        classCovariances[classLabel][0, 0] += dx * dx;
        classCovariances[classLabel][0, 1] += dx * dy;
        classCovariances[classLabel][1, 0] += dx * dy;
        classCovariances[classLabel][1, 1] += dy * dy;
    }

    // Деление на количество точек в каждом классе
    for (var c = 0; c < numClasses; c++)
    {
        classCovariances[c][0, 0] /= classSampleCounts[c]; // xx

```

```

        classCovariances[c][0, 1] /= classSampleCounts[c]; // xy
        classCovariances[c][1, 0] /= classSampleCounts[c]; // yx
        classCovariances[c][1, 1] /= classSampleCounts[c]; // yy
    }

    return classCovariances;
}

private static double[] GetClassPriors(int numClasses, int[]
classSampleCounts, int totalSamples)
{
    var classPriors = new double[numClasses];

    for (var c = 0; c < numClasses; c++) classPriors[c] =
(double)classSampleCounts[c] / totalSamples;

    return classPriors;
}

```

Полученные значения выведены в выход ячейки и представлены на рисунке 1.

```

ClassMeans:
4,955682056116773, 3,603259324689267
-4,029042219086001, 2,0327180555532856
2,3135065107795016, -2,5392712025240236
ClassCovariances:
4,152724338764098, 2,7130375543222245, 2,7130375543222245, 4,174988492351254
2,455619847204429, -0,5995607064555507, -0,5995607064555507, 3,312996059193047
2,8597946881185865, 2,37644049141507, 2,37644049141507, 3,9577183571098327
ClassPriors:
0,3333333333333333
0,3333333333333333
0,3333333333333333

```

Рисунок 1 – Восстановленные параметры

Для обучающей выборки классов запишем формулу оценки плотности вероятности:

$$f(\vec{x}|\Omega_l) = \frac{1}{(2\pi)^{d/2}\sqrt{|B_l|}} \exp\left(-\frac{1}{2}(\vec{x} - \vec{M}_l)^T B_l^{-1}(\vec{x} - \vec{M}_l)\right)$$

Листинг 2 – Оценка плотности вероятности

```

public static double CalculateDensity(double x, double y,
double[] classMeans, double[,] covariance)
{

```

```

var meanX = classMeans[0];
var meanY = classMeans[1];

// Определитель ковариационной матрицы
var determinant = GetDeterminant(covariance);

// Обратная ковариационная матрица
var invCovariance = GetInvertedCovariance(covariance,
determinant);

// Отклонения от среднего
var dx = x - meanX;
var dy = y - meanY;

var quad = GetQuad(dx, invCovariance, dy);

var factor = 1.0 / (2.0 * Math.PI * Math.Sqrt(determinant));

// Плотность вероятности
var density = factor * Math.Exp(-0.5 * quad);

return density;
}

private static double GetDeterminant(double[,] covariance)
{
    return covariance[0, 0] * covariance[1, 1] - covariance[0,
1] * covariance[1, 0];
}

private static double[,] GetInvertedCovariance(double[,]
covariance, double determinant)
{
    var invCovariance = new double[2, 2];

    invCovariance[0, 0] = covariance[1, 1] / determinant;
    invCovariance[0, 1] = -covariance[0, 1] / determinant;
    invCovariance[1, 0] = -covariance[1, 0] / determinant;
    invCovariance[1, 1] = covariance[0, 0] / determinant;
    return invCovariance;
}

private static double GetQuad(double dx, double[,]
invCovariance, double dy)
{
    return dx * (invCovariance[0, 0] * dx + invCovariance[0, 1]
* dy) +
        dy * (invCovariance[1, 0] * dx + invCovariance[1, 1]
* dy);
}

```

Отношение правдоподобия для классов 0 и 1:

$$\Lambda_{0,1}(\vec{x}) = \frac{\hat{f}(\vec{x}|\Omega_0)}{\hat{f}(\vec{x}|\Omega_1)}$$

Отношение правдоподобия для классов 0 и 2:

$$\Lambda_{0,2}(\vec{x}) = \frac{\hat{f}(\vec{x}|\Omega_0)}{\hat{f}(\vec{x}|\Omega_2)}$$

Отношение правдоподобия для классов 1 и 2:

$$\Lambda_{1,2}(\vec{x}) = \frac{\hat{f}(\vec{x}|\Omega_1)}{\hat{f}(\vec{x}|\Omega_2)}$$

В общем виде для любой пары классов:

$$\begin{aligned} \Lambda_{l,j}(\vec{x}) &= \frac{\hat{f}(\vec{x}|\Omega_l)}{\hat{f}(\vec{x}|\Omega_j)} = \\ &= \frac{\sqrt{|\widehat{B}_j|}}{\sqrt{|\widehat{B}_l|}} \exp \left(-\frac{1}{2} (\vec{x} - \widehat{M}_l)^T \widehat{B}_l^{-1} (\vec{x} - \widehat{M}_l) + \frac{1}{2} (\vec{x} - \widehat{M}_j)^T \widehat{B}_j^{-1} (\vec{x} - \widehat{M}_j) \right) \end{aligned}$$

Для каждой пары классов l и j пороговое значение:

$$\lambda_{l,j} = \frac{P(\Omega_j)}{P(\Omega_l)}$$

Объект \vec{x} относится к классу Ω_l , если для всех $j \neq l$ выполняется:

$$\Lambda_{l,j}(\vec{x}) \geq \lambda_{l,j}$$

Для трех классов правила классификации следующие.

Объект относится к классу Ω_0 , если:

$$\Lambda_{0,1}(\vec{x}) \geq \frac{P(\Omega_1)}{P(\Omega_0)} \text{ и } \Lambda_{0,2}(\vec{x}) \geq \frac{P(\Omega_2)}{P(\Omega_0)}$$

Объект относится к классу Ω_1 , если:

$$\Lambda_{1,0}(\vec{x}) \geq \frac{P(\Omega_0)}{P(\Omega_1)} \text{ и } \Lambda_{1,2}(\vec{x}) \geq \frac{P(\Omega_2)}{P(\Omega_1)}$$

Объект относится к классу Ω_2 , если:

$$\Lambda_{2,0}(\vec{x}) \geq \frac{P(\Omega_0)}{P(\Omega_2)} \text{ и } \Lambda_{2,1}(\vec{x}) \geq \frac{P(\Omega_1)}{P(\Omega_2)}$$

Построим отношение правдоподобия на основании формул оценок плотностей вероятностей каждого класса и априорных вероятностей (1/3) появления объектов из каждого класса. Классифицируем элементы контрольной выборки (листинг 4).

Листинг 4 – Классификация тестовой выборки

```
public static int ClassifyNaiveBayes(double x, double y,
double[][] means, double[][,] covariances, double[] priors)
{
    var l1 = DensityCalculator.CalculateDensity(x, y, means[0],
covariances[0]);
    var l2 = DensityCalculator.CalculateDensity(x, y, means[1],
covariances[1]);
    var l3 = DensityCalculator.CalculateDensity(x, y, means[2],
covariances[2]);

    var lr12 = l1 * priors[0] / (l2 * priors[1]); // Отношение
правдоподобия класс 1 к классу 2
    var lr13 = l1 * priors[0] / (l3 * priors[2]); // Отношение
правдоподобия класс 1 к классу 3
    var lr23 = l2 * priors[1] / (l3 * priors[2]); // Отношение
правдоподобия класс 2 к классу 3

    if (lr12 > 1 && lr13 > 1)
    {
        return 0; // Класс 1 выигрывает по отношению
правдоподобия
    }

    if (lr12 < 1 && lr23 > 1)
    {
        return 1; // Класс 2 выигрывает по отношению
правдоподобия
    }

    return 2; // Класс 3 выигрывает по отношению правдоподобия
}
```

Оценка точности классификации приводится в качестве отладочного вывода на рисунке 2.


```
Classification efficiency: 95,33 %
Classification error: 4,67 %
Correctly classified: 143 of 150
Incorrectly classified: 7 of 150
```

Рисунок 2 – Оценка точности классификации

Визуализацию полученных границ и тестовых наборов осуществляем с использованием библиотеки ScottPlot 5. В листинге 5 представлен код визуализации сетки, отображающей границы принятия решений.

Листинг 6 – Визуализация сетки границ принятия решений

```
for (int i = 0; i < gridSize; i++)
{
    for (int j = 0; j < gridSize; j++)
    {
        double x = xGrid[i];
        double y = yGrid[j];
        int cls = Classifier.ClassifyNaiveBayes(x, y,
classMeans, classCovariances, classPriors);

        if (cls == 0)
        {
            var marker = plt.Add.Scatter(new[] { x }, new[] { y
}, plotBgColors[0]);
            marker.MarkerShape = MarkerShape.OpenSquare;
            plt.MoveToBack(marker);
        }
        else if (cls == 1)
        {
            var marker = plt.Add.Scatter(new[] { x }, new[] { y
}, plotBgColors[1]);
            marker.MarkerShape = MarkerShape.OpenSquare;
            plt.MoveToBack(marker);
        }
        else
        {
            var marker = plt.Add.Scatter(new[] { x }, new[] { y
}, plotBgColors[2]);
            marker.MarkerShape = MarkerShape.OpenSquare;
            plt.MoveToBack(marker);
        }
    }
}
```

Результат визуализации представлен на рисунке 2.

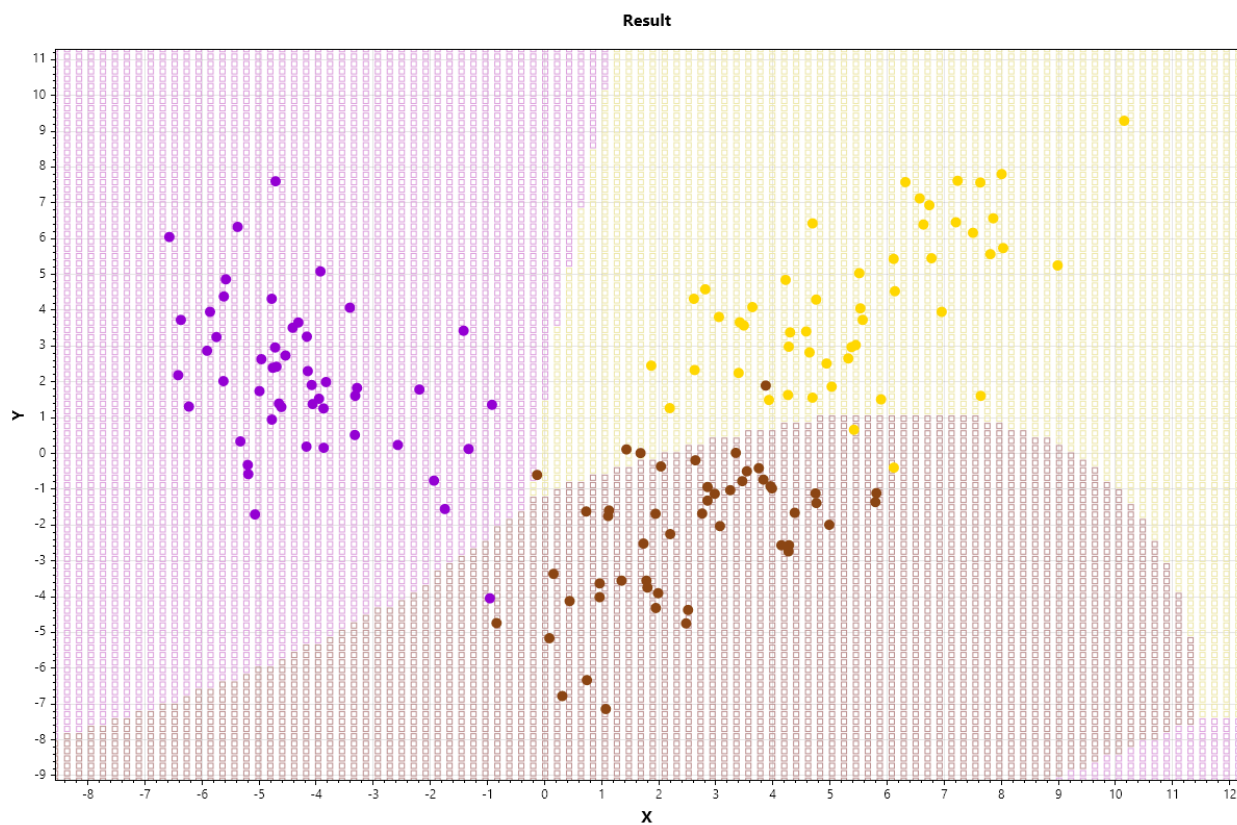


Рисунок 2 - Визуализация

Вывод: в ходе выполнения лабораторной работы были изучены теоретические основы и проведено экспериментальное исследование метода построения байесовского классификатора для распознавания образов.