Лосева Елизавета Юрьевна, группа 8-2

Лабораторная работа № 3

Вариант № 16

Распознавание образов, описываемых гауссовскими случайными векторами с разными матрицами ковариаций

Цель работы

Синтезировать алгоритмы распознавания образов, описываемых гауссовскими случайными векторами с разными матрицами ковариаций. Исследовать синтезированные алгоритмы распознавания с точки зрения ожидаемых потерь и ошибок.

Задание

Описание варианта: m1=[0 -1], m2=[4 -2], C1=[3 1; 1 3], C2=[4 -2; -2 4].

Построить график зависимости экспериментальной ошибки первого рода (для второго класса) от величины дисперсии классов. Сравнить с теоретическим значением.

Код программы

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import multivariate_normal
from sklearn.metrics import confusion_matrix
from matplotlib.patches import Ellipse
import sys
import io

sys.stdout = io.TextIOWrapper(sys.stdout.buffer, encoding='utf-8')

# Параметры классов
m1 = np.array([0, -1])
m2 = np.array([4, -2])
C1_base = np.array([[3, 1], [1, 3]])
C2_base = np.array([[4, -2], [-2, 4]])

# Априорные вероятности (предположим равные)
p1 = 0.5
p2 = 0.5

# Функция байесовского классификатора
def bayes_classifier(x, rv1, rv2, p1, p2):
    prob1 = rv1.pdf(x) * p1
    prob2 = rv2.pdf(x) * p2
    return 1 if prob1 > prob2 else 2

# Функция для проведения эксперимента с заданным коэффициентом дисперсии
def run_variance_experiment(factor, title, num_samples=1000):
    # Масштабируем ковариационные матрицы
    C1_scaled = C1_base * factor
```

```
    C2_scaled = C2_base * factor

    # Создаем распределения
    rv1 = multivariate_normal(mean=m1, cov=C1_scaled)
    rv2 = multivariate_normal(mean=m2, cov=C2_scaled)

    # Генерация тестовых данных
    samples1 = rv1.rvs(size=num_samples)
    samples2 = rv2.rvs(size=num_samples)

    y_true = [1] * num_samples + [2] * num_samples
    X = np.vstack([samples1, samples2])

    # Классификация
    y_pred = []
    for x in X:
        y_pred.append(bayes_classifier(x, rv1, rv2, p1, p2))

    # Вычисление матрицы ошибок
    cm = confusion_matrix(y_true, y_pred)
    accuracy = np.trace(cm) / np.sum(cm)

    # Ошибка первого рода для второго класса (класс 2 ошибочно отнесен к классу
1)
    error_type1_class2 = cm[1, 0] / np.sum(cm[1, :]) if np.sum(cm[1, :]) > 0 else
0


    # Теоретическая ошибка
    theoretical_error = theoretical_error_type1_class2(rv1, rv2, p1, p2)

    # Визуализация результатов
    plt.figure(figsize=(18, 5))

    # 1. График матрицы ошибок
    plt.subplot(1, 3, 1)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Матрица ошибок\nТочность: {accuracy:.3f}\nОшибка 1р/2кл:
{error_type1_class2:.3f}')
    plt.xlabel('Предсказанный класс')
    plt.ylabel('Истинный класс')

    # 2. График распределения точек и разделяющих поверхностей
    plt.subplot(1, 3, 2)
    colors = ['red', 'blue']

    # Отображаем сгенерированные точки
    plt.scatter(samples1[:, 0], samples1[:, 1], alpha=0.6, label='Класс 1',
color=colors[0], s=10)
    plt.scatter(samples2[:, 0], samples2[:, 1], alpha=0.6, label='Класс 2',
color=colors[1], s=10)
```

```
    # Построение разделяющей поверхности
    x_vals = np.linspace(-5, 10, 100)
    y_vals = np.linspace(-8, 4, 100)
    X_grid, Y_grid = np.meshgrid(x_vals, y_vals)
    Z = np.zeros_like(X_grid)

    for i in range(X_grid.shape[0]):
        for j in range(X_grid.shape[1]):
            point = np.array([X_grid[i, j], Y_grid[i, j]])
            Z[i, j] = bayes_classifier(point, rv1, rv2, p1, p2)

    plt.contour(X_grid, Y_grid, Z, levels=[1.5], colors='black',
linestyles='dashed', linewidths=2)
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.title('Распределение классов и решающая граница')
    plt.legend()
    plt.grid(True, alpha=0.3)

    # 3. График центров и эллипсов ковариаций
    plt.subplot(1, 3, 3)

    # Отображаем центры классов
    plt.scatter(m1[0], m1[1], color='darkred', s=150, marker='x', linewidth=3,
label='Центр класса 1')
    plt.scatter(m2[0], m2[1], color='darkblue', s=150, marker='x', linewidth=3,
label='Центр класса 2')

    # Рисуем эллипсы ковариаций для каждого класса
    covs = [C1_scaled, C2_scaled]
    means = [m1, m2]
    ellipse_colors = ['red', 'blue']

    for i, (cov, mean) in enumerate(zip(covs, means)):
        eigenvals, eigenvecs = np.linalg.eig(cov)
        angle = np.degrees(np.arctan2(eigenvecs[1, 0], eigenvecs[0, 0]))
        width = 2 * np.sqrt(5.991 * eigenvals[0])  # 95% доверительный интервал
        height = 2 * np.sqrt(5.991 * eigenvals[1])

        ellipse = Ellipse(xy=mean, width=width, height=height, angle=angle,
                          alpha=0.3, color=ellipse_colors[i], linewidth=2,
fill=True)
        plt.gca().add_patch(ellipse)

        # Контур эллипса
        ellipse_contour = Ellipse(xy=mean, width=width, height=height,
angle=angle,
                                  alpha=1.0, color=ellipse_colors[i], linewidth=2,
fill=False)
        plt.gca().add_patch(ellipse_contour)
```

```
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.title('Центры и эллипсы ковариаций')
    plt.legend()
    plt.grid(True, alpha=0.3)

    plt.suptitle(f'{title}\nКоэффициент дисперсии: {factor}, Теор. ошибка:
{theoretical_error:.3f}',
                 fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    return cm, accuracy, error_type1_class2, theoretical_error

# Теоретическая ошибка первого рода для второго класса
def theoretical_error_type1_class2(rv1, rv2, p1, p2, num_points=10000):
    samples = rv2.rvs(size=num_points)
    errors = 0
    for sample in samples:
        if bayes_classifier(sample, rv1, rv2, p1, p2) == 1:
            errors += 1
    return errors / num_points

# Экспериментальная ошибка первого рода для второго класса
def experimental_error_type1_class2(rv1, rv2, p1, p2, sample_size,
num_trials=100):
    errors = []
    for _ in range(num_trials):
        samples = rv2.rvs(size=sample_size)
        error_count = 0
        for sample in samples:
            if bayes_classifier(sample, rv1, rv2, p1, p2) == 1:
                error_count += 1
        errors.append(error_count / sample_size)
    return np.mean(errors), np.std(errors)

# Основной эксперимент
if __name__ == "__main__":
    # Диапазон коэффициентов масштабирования дисперсии
    variance_factors = [0.1, 0.2, 0.5, 1.0, 2.0, 5.0]

    experimental_errors = []
    experimental_stds = []
    theoretical_errors = []

    print("Зависимость ошибки первого рода для второго класса от дисперсии:")
    print("Коэф. дисперсии | Эксп. ошибка | Теор. ошибка")
    print("-" * 50)

    # Создаем график зависимости ошибки от дисперсии
    plt.figure(figsize=(10, 6))
```

```
    for i, factor in enumerate(variance_factors):
        # Вычисляем ошибки
        C1_scaled = C1_base * factor
        C2_scaled = C2_base * factor
        rv1 = multivariate_normal(mean=m1, cov=C1_scaled)
        rv2 = multivariate_normal(mean=m2, cov=C2_scaled)

        exp_error, exp_std = experimental_error_type1_class2(rv1, rv2, p1, p2,
sample_size=1000)
        theor_error = theoretical_error_type1_class2(rv1, rv2, p1, p2)

        experimental_errors.append(exp_error)
        experimental_stds.append(exp_std)
        theoretical_errors.append(theor_error)

        print(f"{factor:15.1f} | {exp_error:12.4f} | {theor_error:12.4f}")

        # Запускаем эксперимент с визуализацией для ключевых коэффициентов
        if factor in [0.2, 1.0, 5.0]:
            cm, accuracy, exp_error, theor_error = run_variance_experiment(
                factor, f"Эксперимент с коэффициентом дисперсии: {factor}")

    # График зависимости ошибки от дисперсии
    plt.errorbar(variance_factors, experimental_errors, yerr=experimental_stds,
                 fmt='o-', linewidth=2, markersize=8, capsize=5, capthick=2,
                 label='Экспериментальная ошибка ± СКО', color='blue', alpha=0.7)

    plt.plot(variance_factors, theoretical_errors, 's--', linewidth=2,
markersize=6,
             label='Теоретическая ошибка', color='red', alpha=0.8)

    plt.xlabel('Коэффициент масштабирования дисперсии', fontsize=12)
    plt.ylabel('Ошибка первого рода для второго класса', fontsize=12)
    plt.title('Зависимость ошибки первого рода от величины дисперсии',
fontsize=14)
    plt.grid(True, alpha=0.3)
    plt.legend(fontsize=10)
    plt.tight_layout()
    plt.show()

    # Анализ матриц ковариации
    print("\n" + "=" * 60)
    print("АНАЛИЗ КОВАРИАЦИОННЫХ МАТРИЦ")
    print("=" * 60)
    print(f"Исходная C1 = \n{C1_base}")
    print(f"Исходная C2 = \n{C2_base}")

    # Собственные значения и векторы
    eigvals1, eigvecs1 = np.linalg.eig(C1_base)
    eigvals2, eigvecs2 = np.linalg.eig(C2_base)
```

```python
print(f"\nСобственные значения C1: {eigvals1}")
print(f"Собственные векторы C1:\n{eigvecs1}")
print(f"\nСобственные значения C2: {eigvals2}")
print(f"Собственные векторы C2:\n{eigvecs2}")

# Расстояние Махаланобиса между классами (для исходных матриц)
delta = m1 - m2
C_avg = (C1_base + C2_base) / 2
mahalanobis_dist = np.sqrt(delta.T @ np.linalg.inv(C_avg) @ delta)
print(f"\nРасстояние Махаланобиса между классами: {mahalanobis_dist:.4f}")
```

Результаты выполнения задания

Зависимость экспериментальной ошибки первого рода (для второго класса) от величины дисперсии классов.

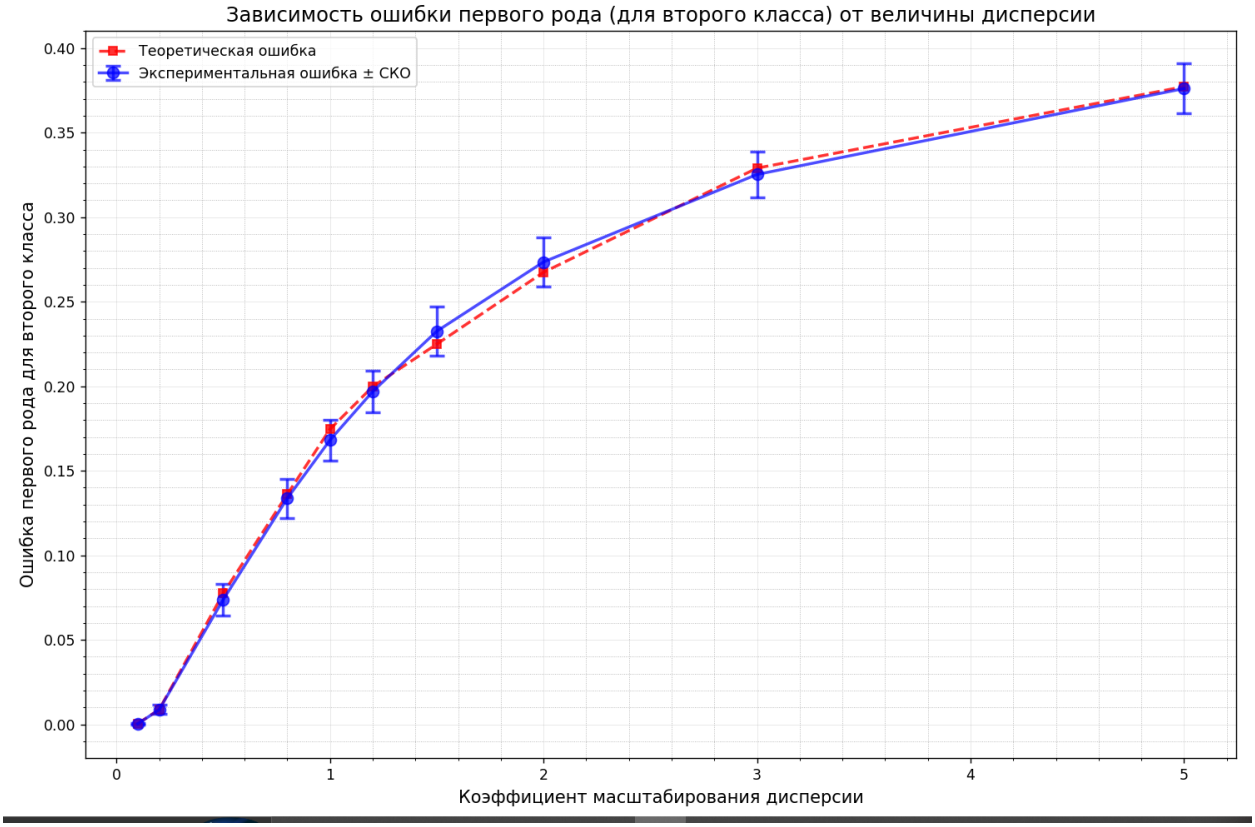

Рисунок 1.

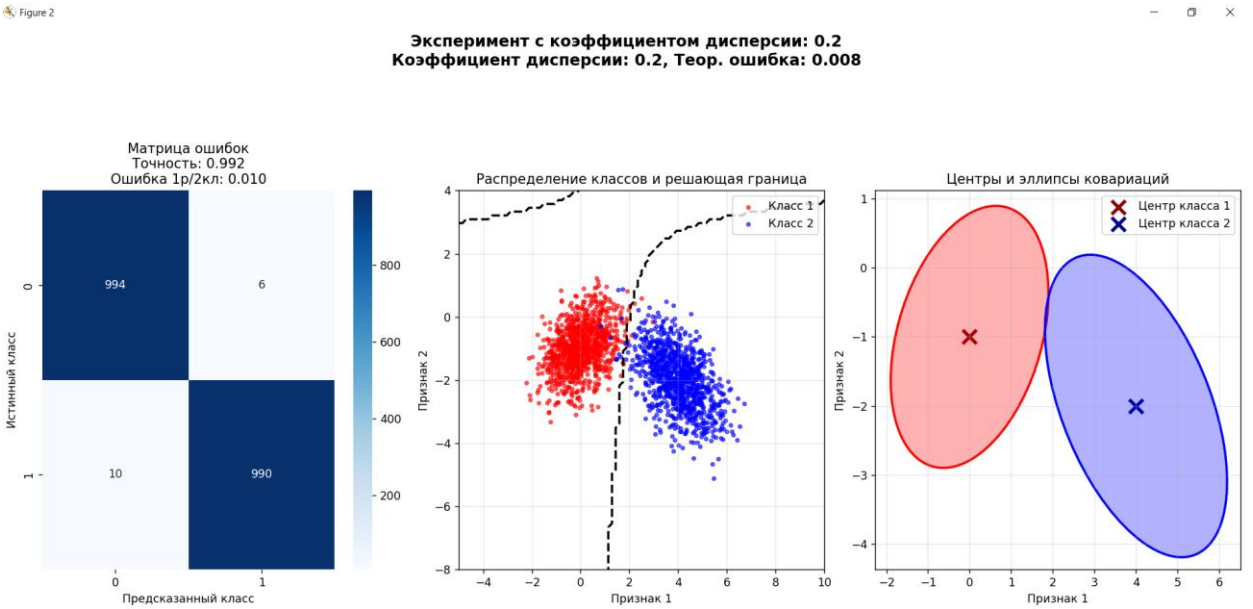

Рисунок 2.

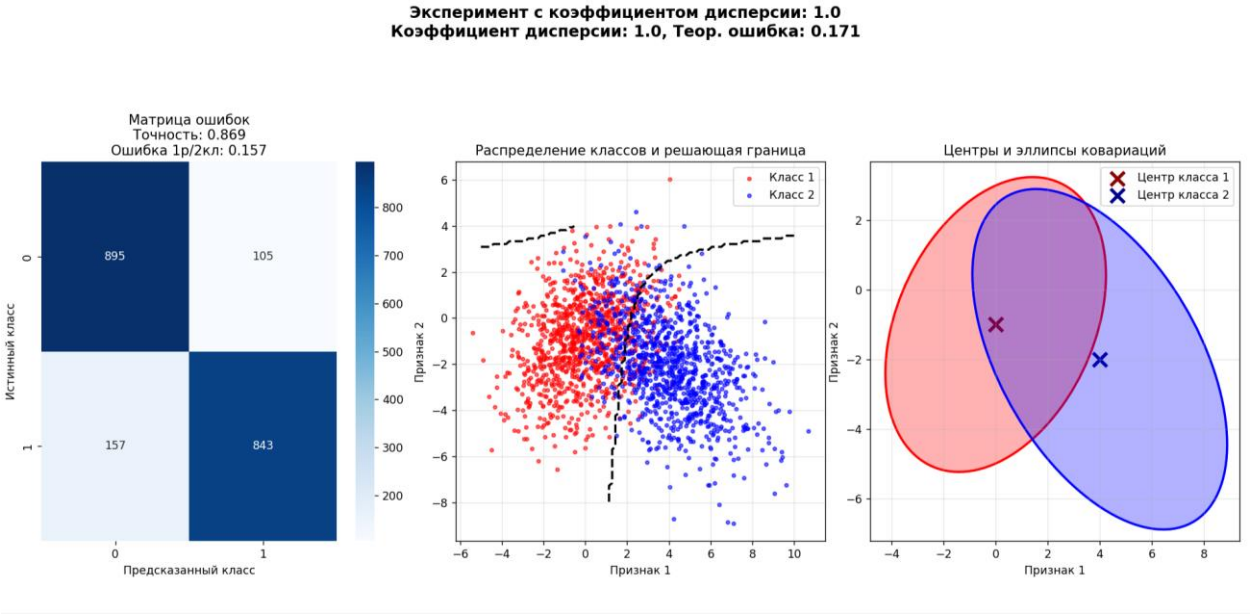

Рисунок 3.

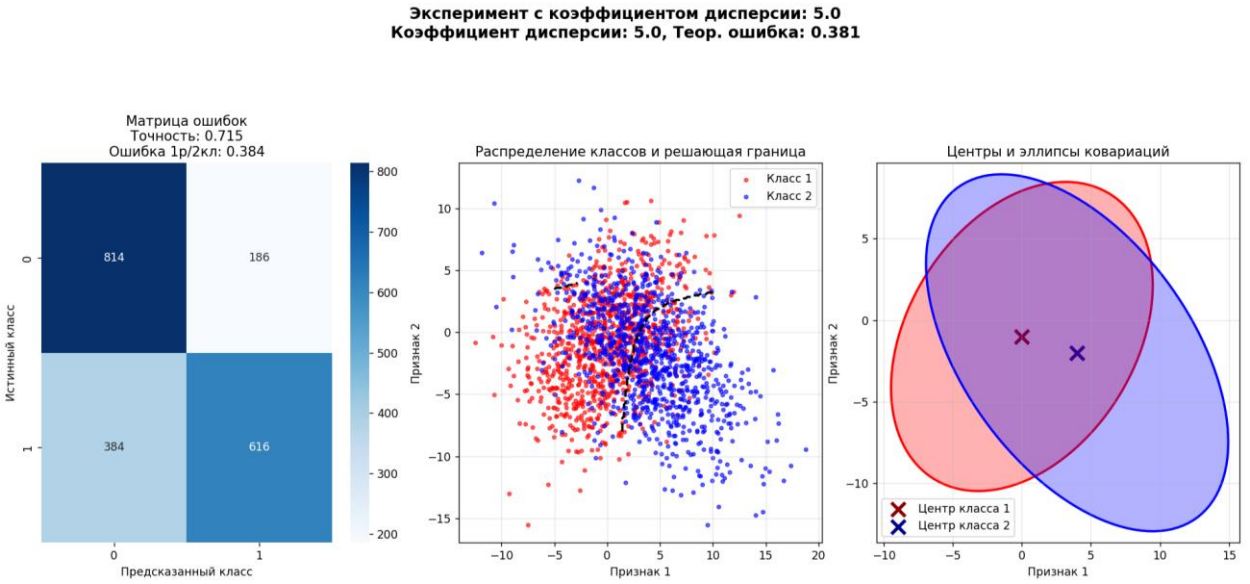

Рисунок 4.

Результаты и анализ ошибок

```
Зависимость ошибки первого рода для второго класса от дисперсии:
Коэф. дисперсии | Эксп. ошибка | Теор. ошибка
-------------------------------------------------
            0.1 |       0.0004 |       0.0005
            0.2 |       0.0087 |       0.0088
            0.5 |       0.0738 |       0.0775
            0.8 |       0.1337 |       0.1361
            1.0 |       0.1682 |       0.1746
            1.2 |       0.1969 |       0.1998
            1.5 |       0.2325 |       0.2249
            2.0 |       0.2735 |       0.2674
            3.0 |       0.3253 |       0.3290
            5.0 |       0.3761 |       0.3772
```

Рисунок 5.

```
АНАЛИЗ КОВАРИАЦИОННЫХ МАТРИЦ
============================================================
Исходная C1 =
[[3 1]
 [1 3]]
Исходная C2 =
[[ 4 -2]
 [-2  4]]

Собственные значения C1: [4. 2.]
Собственные векторы C1:
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]

Собственные значения C2: [6. 2.]
Собственные векторы C2:
[[ 0.70710678  0.70710678]
 [-0.70710678  0.70710678]]

Расстояние Махаланобиса между классами: 2.1506
```

Рисунок 6.

Результаты выполнения доп. заданий:
а) изменить исходные данные таким образом, чтобы увеличить вероятности правильного распознавания;

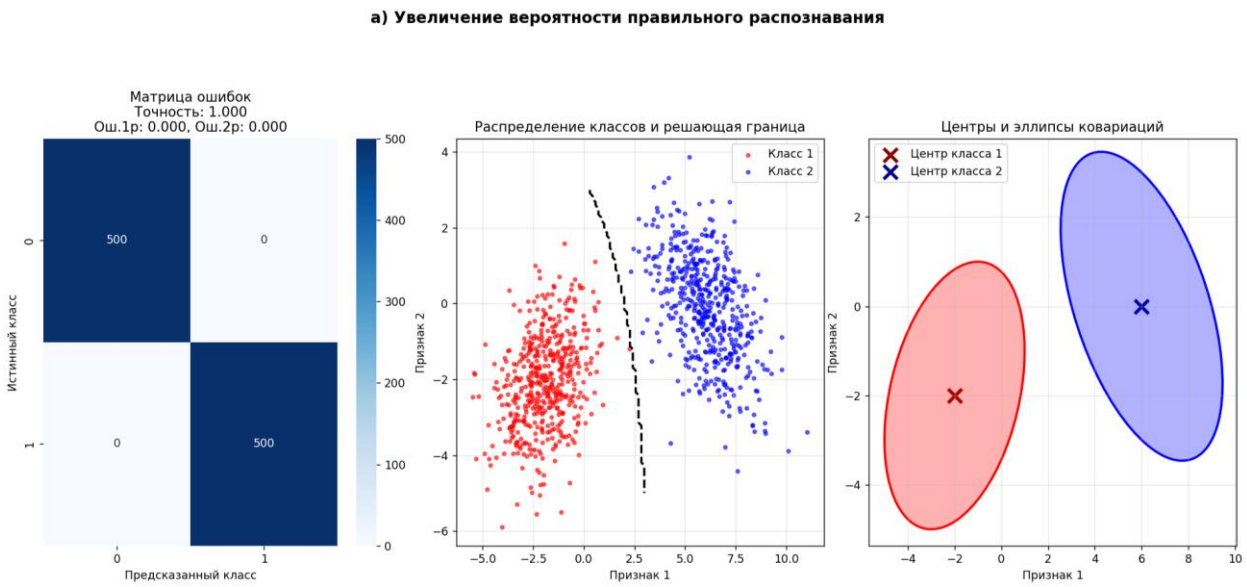

Рисунок 7.

b) изменить исходные данные таким образом, чтобы увеличить суммарную ошибку;

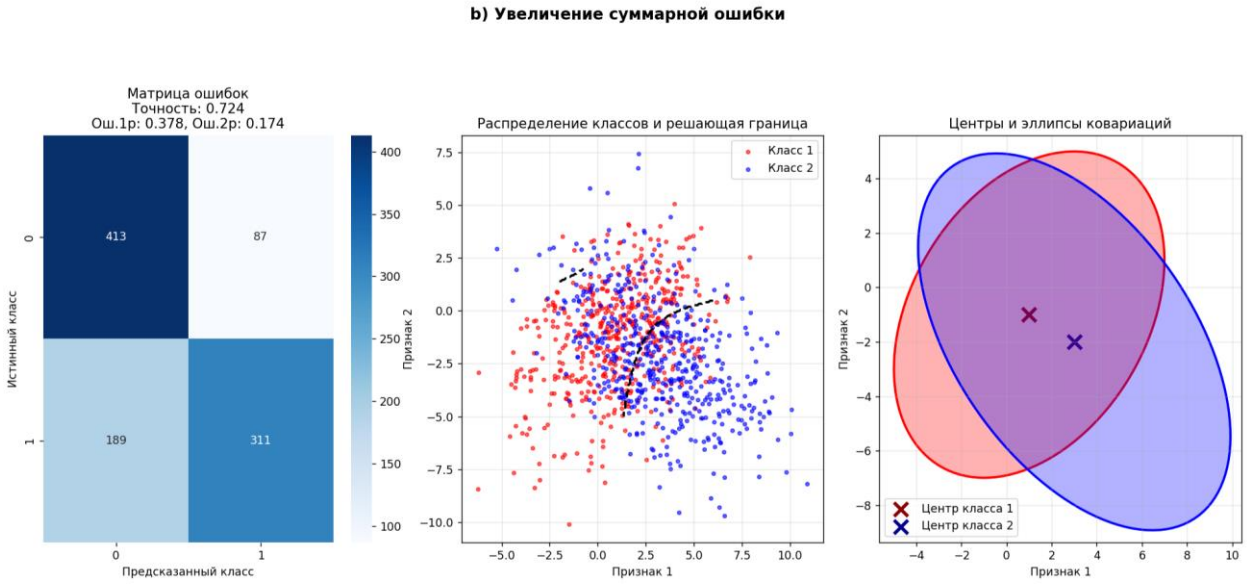

Рисунок 8.

c) изменить исходные данные таким образом, чтобы в теоретической матрице ошибок увеличилась ошибка первого рода, а ошибка второго рода уменьшилась;

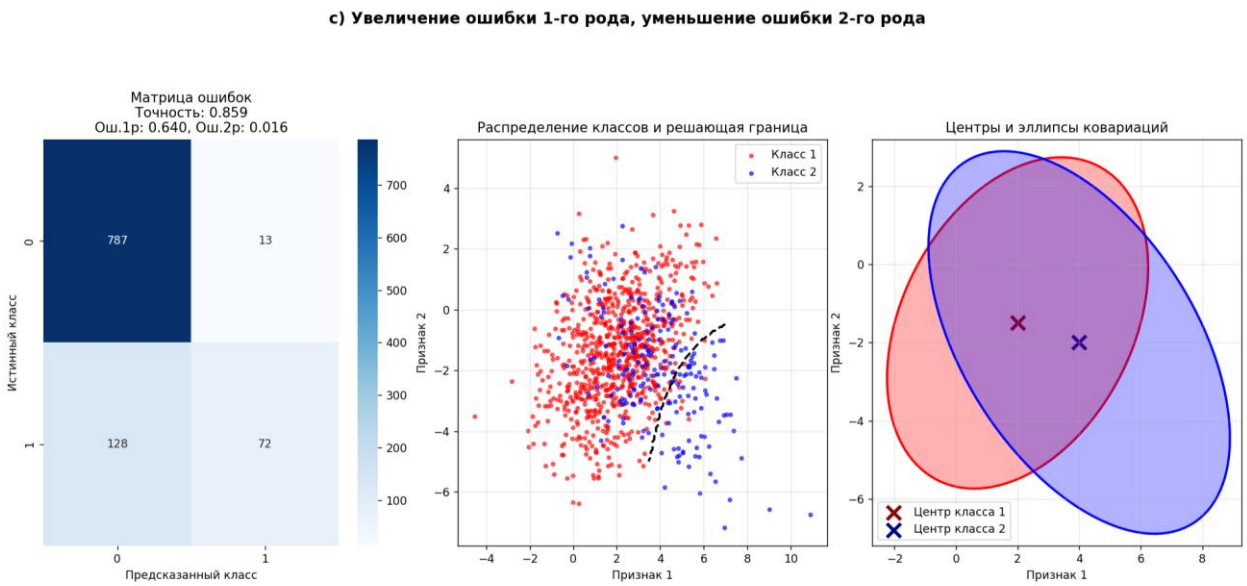

Рисунок 9.

d) изменить исходные данные таким образом, чтобы в теоретической матрице ошибок увеличилась ошибка второго рода, а ошибка первого рода уменьшилась;

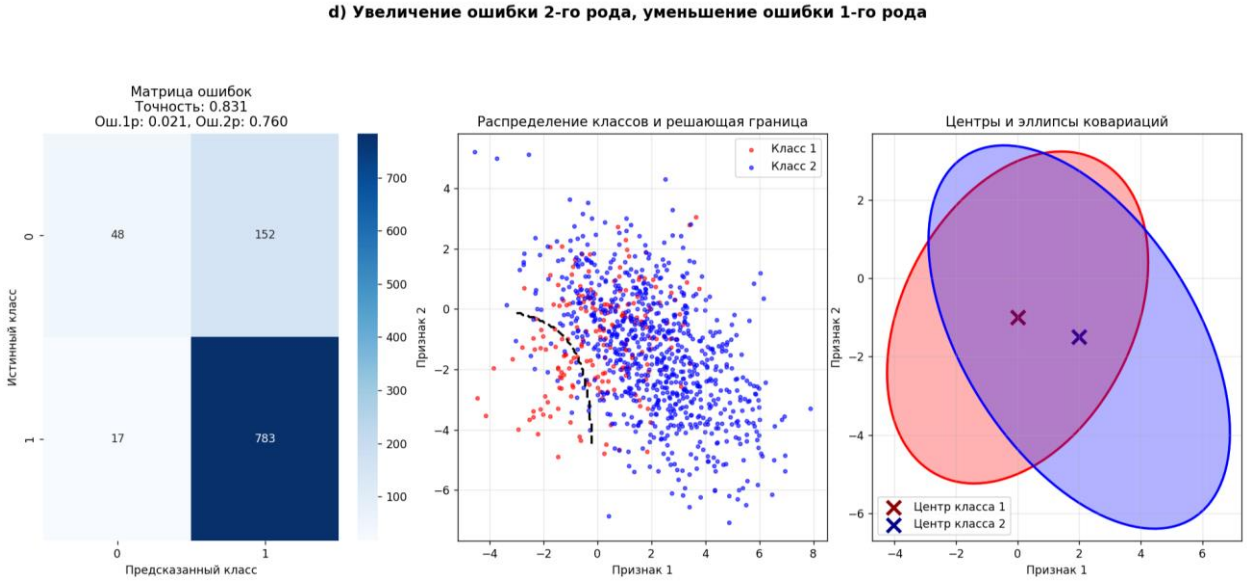

Рисунок 10.

e) изменить исходные данные таким образом, чтобы увеличить протяженность области локализации образов всех классов (растянуть форму кластеров) в одном из направлений;

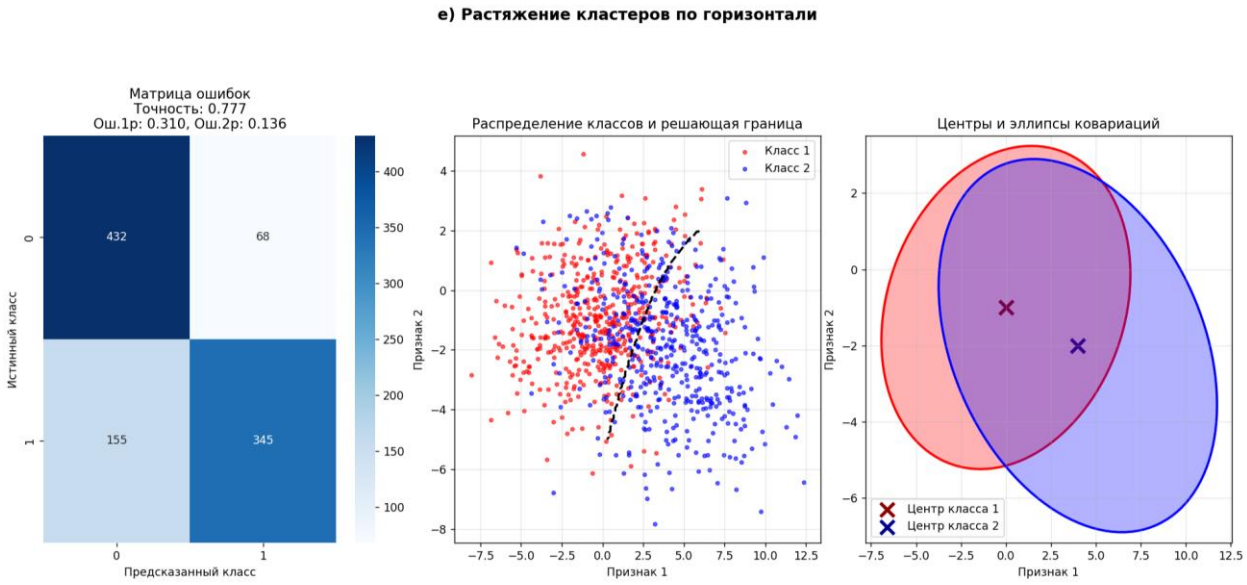

Рисунок 11.

f) изменить исходные данные таким образом, чтобы зеркально отразить форму областей локализации образов всех классов (форму кластеров).

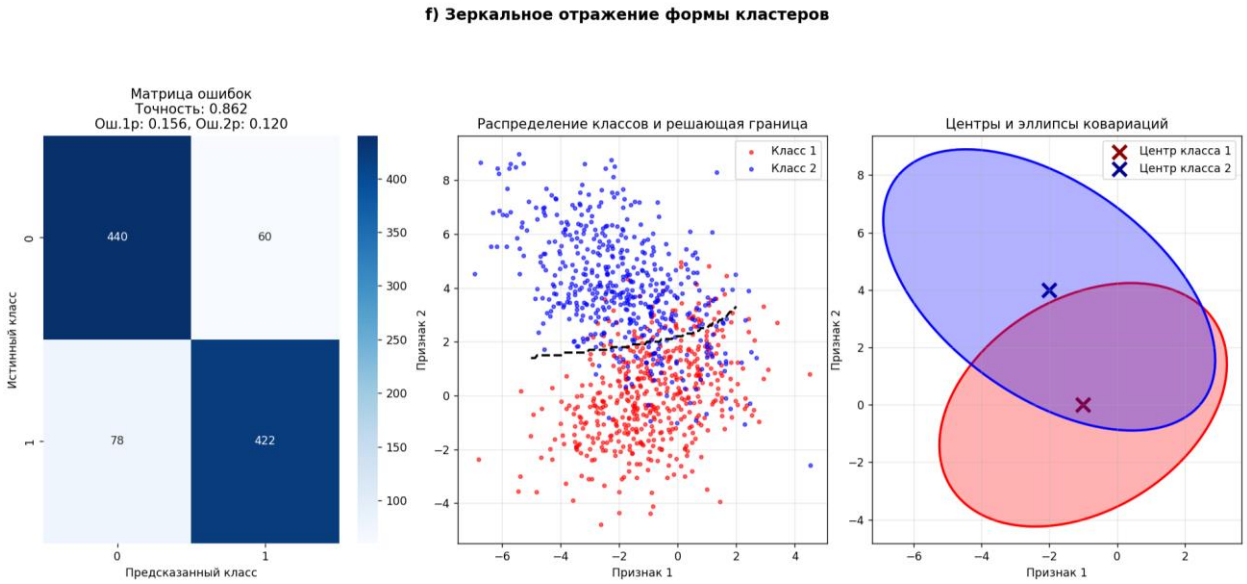

Рисунок 12.

Ответы на контрольные вопросы

1. Элементы главной диагонали матрицы ошибок характеризуют количество правильно классифицированных объектов каждого класса. Каждый элемент a_{ii} показывает, сколько объектов i-го класса было корректно отнесено к своему классу.
2. Элементы побочных диагоналей характеризуют ошибки классификации - количество объектов, которые были неправильно отнесены к другим классам. Элемент a_{ij} (где i != j) показывает, сколько объектов i-го класса было ошибочно классифицировано как j-й класс.

3. Форма кластеров объектов определяется ковариационной матрицей распределения признаков внутри каждого класса. Собственные векторы ковариационной матрицы задают ориентацию кластера, а собственные значения – размеры и форму эллипсоида рассеивания вдоль соответствующих направлений.

Выводы

Реализован байесовский классификатор для двух гауссовских классов с заданными параметрами. Проведено исследование зависимости ошибки первого рода для второго класса от величины дисперсии классов.

Экспериментальные результаты показали, что с увеличением коэффициента масштабирования дисперсии значение ошибки монотонно возрастает. Это объясняется увеличением перекрытия распределений классов при росте дисперсии.

Анализ собственных значений и векторов ковариационных матриц показал различные ориентации эллипсов рассеивания для двух классов, что влияет на форму решающей границы и вероятность ошибки классификации.