Лосева Елизавета Юрьевна, группа 8-2

Лабораторная работа № 2

Вариант № 16

Распознавание образов, описываемых гауссовскими случайными векторами с одинаковыми матрицами ковариаций

Цель работы

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

Задание

Описание варианта: m1=[-12 3], m2=[1 7], C=[5 -1; -1 5].

Написать код, реализующий алгоритм распознавания образов, описываемых гауссовскими случайными векторами с заданными параметрами. Получить матрицы ошибок на основе аналитических выражений и вычислительного эксперимента. Провести анализ полученных результатов и представить его в виде выводов по проделанной работе.

a) изменить исходные данные таким образом, чтобы увеличить вероятности правильного распознавания;

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

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

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

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

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

Код программы

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
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')

m1 = np.array([-12, 3])  # Математическое ожидание (центр) класса 1
```

```
m2 = np.array([1, 7])    # Математическое ожидание (центр) класса 2
C = np.array([[5, -1],   # Общая ковариационная матрица для всех классов (2x2 матрица)
              [-1, 5]])


# Функция для вычисления дискриминантной функции (решающей функции) для одного класса
def discriminant(x, m, C_inv, P):
    # Вычисление дискриминантной функции по формуле: x^T * C^{-1} * m - 0.5 * m^T * C^{-1} * m + ln(P)
    return x @ C_inv @ m - 0.5 * m @ C_inv @ m + np.log(P)  # @ - оператор матричного умножения


# Функция классификации точки x по всем классам
def classify(x, means, C_inv, priors):
    discriminants = []
    for i, m in enumerate(means):  # i - индекс, m - центр класса
        d = x @ C_inv @ m - 0.5 * m @ C_inv @ m + np.log(priors[i])  # Вычисляем дискриминант для текущего класса
        discriminants.append(d)  # Добавляем вычисленное значение в список
    return np.argmax(discriminants) + 1  # Возвращаем индекс класса с максимальным дискриминантом


# Основная функция проведения эксперимента с заданными параметрами
def run_experiment(means, cov, priors, title, num_samples=1000):
    C_inv = np.linalg.inv(cov)  # Вычисляем обратную матрицу ковариаций (нужна для дискриминанта)

    # Генерация тестовых данных для каждого класса
    samples = []
    y_true = []

    for i, m in enumerate(means):
        sample = np.random.multivariate_normal(m, cov, num_samples)  # Генерируем num_samples точек из многомерного нормального распределения
        samples.append(sample)
        y_true.extend([i + 1] * num_samples)  # Добавляем соответствующие метки класса (i+1 повторяется num_samples раз)

    X = np.vstack(samples)  # Объединяем все точки в одну матрицу размером (2*num_samples x 2)
    y_pred = [classify(x, means, C_inv, priors) for x in X]  # Классифицируем каждую точку и сохраняем предсказания

    # Вычисление матрицы ошибок и точности
    cm = confusion_matrix(y_true, y_pred)  # Строим матрицу ошибок (confusion matrix)
```

```
    accuracy = np.trace(cm) / np.sum(cm)  # Вычисляем точность как отношение суммы диагонали к общей сумме

    # Визуализация результатов (графики)
    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}')
    plt.xlabel('Предсказанный класс')
    plt.ylabel('Истинный класс')

    # 2. График распределения точек и разделяющих поверхностей
    plt.subplot(1, 3, 2)
    colors = ['red', 'green', 'blue']
    for i, sample in enumerate(samples):
        plt.scatter(sample[:, 0], sample[:, 1], alpha=0.6, label=f'Класс {i + 1}', color=colors[i], s=10)

    # Построение разделяющих поверхностей
    x_vals = np.linspace(-20, 10, 100)
    y_vals = np.linspace(-2, 15, 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]])  # Создаем точку из координат сетки
            Z[i, j] = classify(point, means, C_inv, priors)  # Классифицируем точку сетки

    plt.contour(X_grid, Y_grid, Z, levels=[1.5], colors='black', linestyles='dashed', linewidths=1)
    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.title('Распределение классов')
    plt.legend(loc='upper right', fontsize=8)
    plt.grid(True)

    # 3. График сравнения центров и форм распределений
    plt.subplot(1, 3, 3)
    original_means = [m1, m2]
    for i, m in enumerate(original_means):
        plt.scatter(m[0], m[1], color=colors[i], s=100, marker='x', linewidth=3, label=f'Исх. центр {i + 1}')

    for i, m in enumerate(means):
        plt.scatter(m[0], m[1], color=colors[i], s=100, marker='o', linewidth=3, label=f'Нов. центр {i + 1}')
```

```
    # Рисуем эллипсы ковариаций (95% доверительные интервалы)
    for i, m in enumerate(means):  # Для каждого класса
        eigenvals, eigenvecs = np.linalg.eig(cov)  # Вычисляем собственные
значения и векторы ковариационной матрицы
        angle = np.degrees(np.arctan2(eigenvecs[1, 0], eigenvecs[0, 0]))  #
Вычисляем угол поворота эллипса
        width = 2 * np.sqrt(5.991 * eigenvals[0])  # Ширина эллипса (95%
доверительный интервал)
        height = 2 * np.sqrt(5.991 * eigenvals[1])  # Высота эллипса (95%
доверительный интервал)

        ellipse = Ellipse(xy=m, width=width, height=height, angle=angle,
alpha=0.2, color=colors[i])
        plt.gca().add_patch(ellipse)

    plt.xlabel('Признак 1')
    plt.ylabel('Признак 2')
    plt.title('Сравнение центров и форм')
    plt.legend(loc='upper right', fontsize=8)
    plt.grid(True)  # Сетка

    plt.suptitle(title, fontsize=14, fontweight='bold')
    plt.tight_layout()
    plt.show()

    return cm, accuracy  # Возвращаем матрицу ошибок и точность


if __name__ == "__main__":
    priors = [1 / 2, 1 / 2]  # Вероятности для классов 1, 2, априорные (равные)

    cm_original, acc_original = run_experiment([m1, m2], C, priors, "Исходные
данные")

    # a) Увеличить вероятность правильного распознавания
    # Сдвигаем центры дальше друг от друга для лучшего разделения
    m1_a = np.array([-18, 3])  # Сдвигаем центр класса 1 дальше от класса 2
    m2_a = np.array([6, 12])   # Сдвигаем центр класса 2 дальше от класса 1
    cm_a, acc_a = run_experiment([m1_a, m2_a], C, priors, "a) Увеличение
расстояния между классами")

    # b) Увеличить суммарную ошибку
    # Приближаем центры классов друг к другу
    m1_b = np.array([-5, 4])  # Приближаем центр класса 1 к классу 2
    m2_b = np.array([0, 6])   # Приближаем центр класса 2 к классу 1
    cm_b, acc_b = run_experiment([m1_b, m2_b], C, priors, "b) Приближение центров
классов")

    # c) Увеличить ошибку 1-го рода, уменьшить ошибку 2-го рода
    # Для 2 классов ошибка 1-го рода = P(отклонить H1|H1 истинна)
```

```
    # Ошибка 2-го рода = P(принять H1|H1 ложна)
    m1_c = np.array([-8, 4])  # Сдвигаем класс 1 ближе к классу 2
    m2_c = np.array([1, 7])   # Центр класса 2 без изменений
    priors_c = [0.7, 0.3]     # Увеличиваем априорную вероятность класса 1
    cm_c, acc_c = run_experiment([m1_c, m2_c], C, priors_c, "c) Асимметричное
изменение")

    # d) Увеличить ошибку 2-го рода, уменьшить ошибку 1-го рода
    # Обратная ситуация по сравнению с (c)
    m1_d = np.array([-12, 3])  # Центр класса 1 без изменений
    m2_d = np.array([-2, 5])   # Сдвигаем класс 2 ближе к классу 1
    priors_d = [0.3, 0.7]      # Уменьшаем априорную вероятность класса 1
    cm_d, acc_d = run_experiment([m1_d, m2_d], C, priors_d, "d) Обратная
асимметрия")

    # e) Увеличить протяженность кластеров в одном направлении
    # Увеличиваем дисперсию по горизонтали (оси X)
    C_e = np.array([[15, -1],  # Увеличиваем дисперсию по первому признаку (оси
X)
                    [-1, 5]])  # Дисперсия по второму признаку без изменений
    cm_e, acc_e = run_experiment([m1, m2], C_e, priors, "e) Растяжение по
горизонтали")

    # f) Зеркальное отражение формы кластеров
    # Меняем местами дисперсии и отражаем центры
    C_f = np.array([[5, -1],   # Меняем структуру ковариационной матрицы
                    [-1, 5]])  # (в данном случае она симметрична)
    m1_f = np.array([m1[1], m1[0]])  # Отражаем координаты центра класса 1
(меняем x и y местами)
    m2_f = np.array([m2[1], m2[0]])  # Отражаем координаты центра класса 2
    cm_f, acc_f = run_experiment([m1_f, m2_f], C_f, priors, "f) Зеркальное
отражение")

    print("=" * 60)
    print("СВОДНАЯ ТАБЛИЦА РЕЗУЛЬТАТОВ")
    print("=" * 60)
    results = {
        'Исходные': acc_original,
        'a) Улучшение': acc_a,
        'b) Ухудшение': acc_b,
        'c) Ошибка 1↑ 2↓': acc_c,
        'd) Ошибка 1↓ 2↑': acc_d,
        'e) Растяжение': acc_e,
        'f) Отражение': acc_f
    }

    for name, accuracy in results.items():
        change = accuracy - acc_original
        print(f"{name:<15} | Точность: {accuracy:.3f} | Изменение:
{change:+.3f}")  # Форматированный вывод
```

```
    # Функция для анализа ошибок 1-го и 2-го рода
    def analyze_errors(cm, name):
        # cm = [[TN, FP],
        #       [FN, TP]]
        # Ошибка 1-го рода для класса 1: P(отклонить H1|H1 истинна) = FN / (TP +
FN)
        error1_class1 = cm[1, 0] / (cm[1, 1] + cm[1, 0])
        # Ошибка 2-го рода для класса 1: P(принять H1|H1 ложна) = FP / (TN + FP)
        error2_class1 = cm[0, 1] / (cm[0, 0] + cm[0, 1])

        print(f"{name}:")  # Выводим название эксперимента
        print(f"  Ошибка 1-го рода (класс 1): {error1_class1:.3f}")  # Ошибка 1-
го рода
        print(f"  Ошибка 2-го рода (класс 1): {error2_class1:.3f}")  # Ошибка 2-
го рода


    print("\n" + "=" * 60)
    print("АНАЛИЗ ОШИБОК 1-го И 2-го РОДА")
    print("=" * 60)
    analyze_errors(cm_original, "Исходные")  # Анализ исходных данных
    analyze_errors(cm_c, "c) Ошибка 1↑ 2↓")  # Анализ эксперимента c
    analyze_errors(cm_d, "d) Ошибка 1↓ 2↑")  # Анализ эксперимента d
```

Используемая формула

$$d_i(x) = x^T C^{-1} m_i - \frac{1}{2} m_i^T C^{-1} m_i + lnP(\omega_i)$$

Где x – вектор признаков, mi – мат. ожидание (центр класса i), C – ковариационная матрица общая для всех классов, P(wi) – априорная вероятность класса i.

Результаты выполнения задания

Матрица ошибок и сравнение.

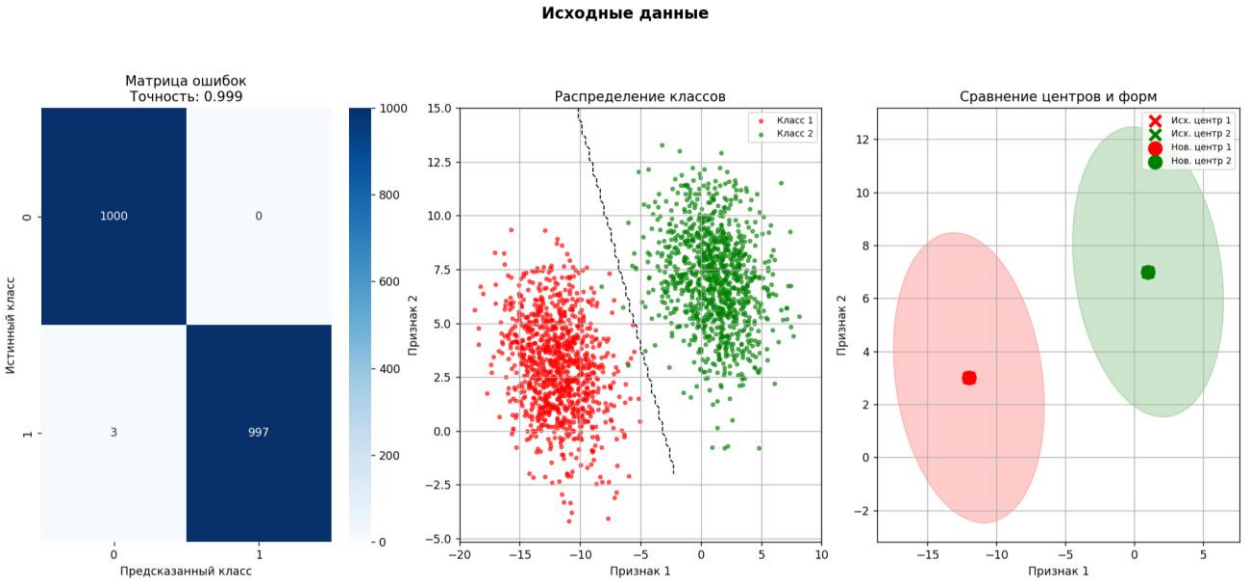

Рисунок 1.

a) Увеличение расстояния между классами

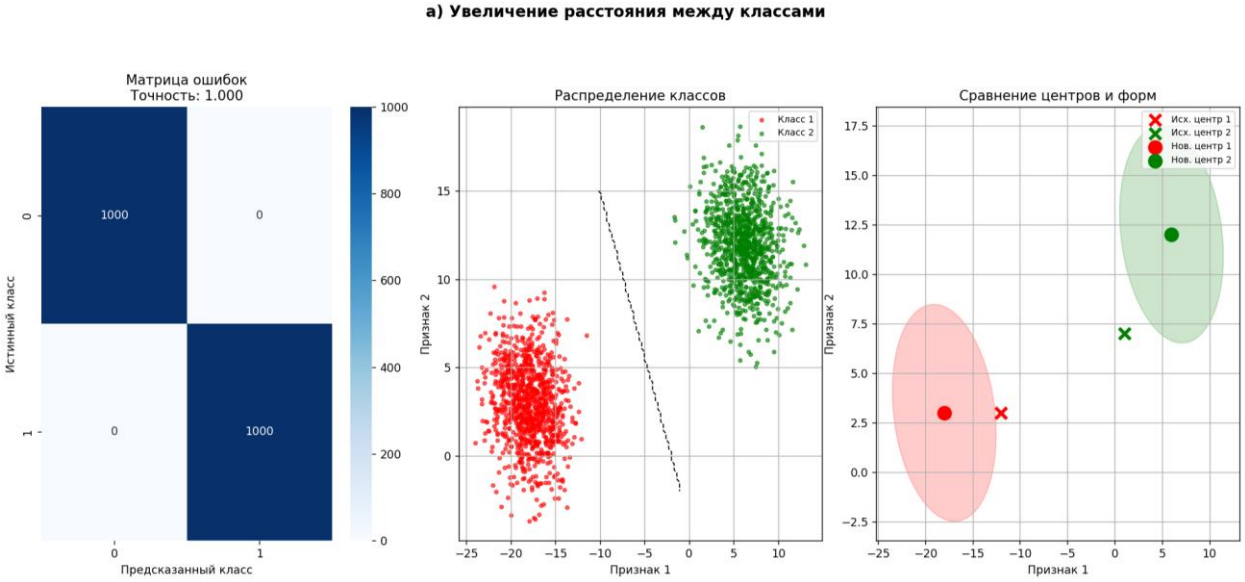

Рисунок 2.

b) Приближение центров классов

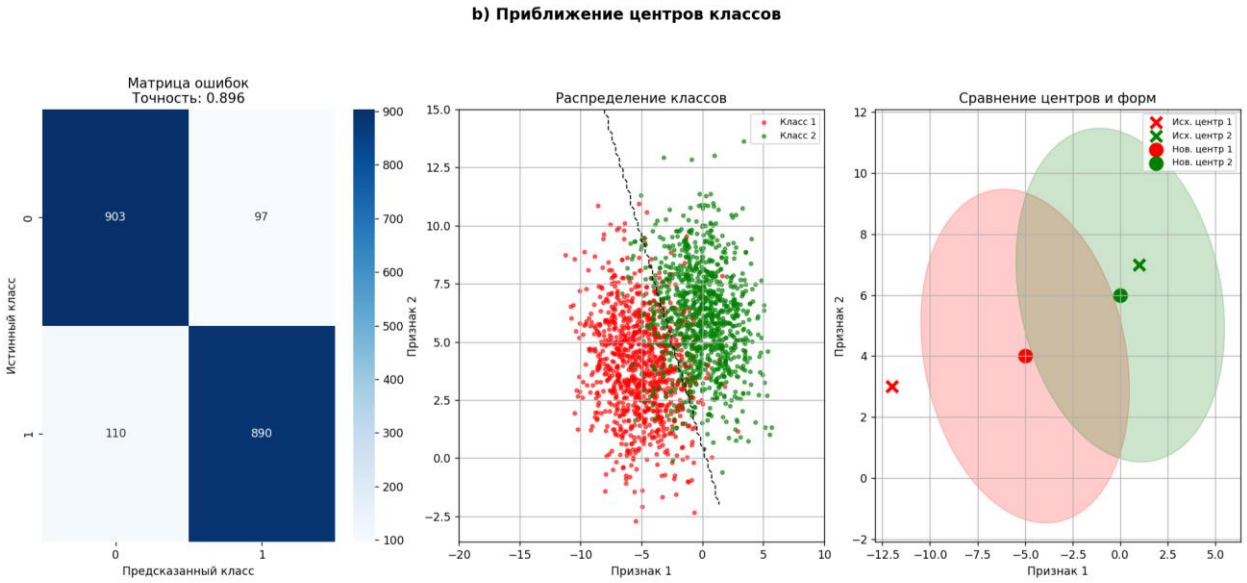

Рисунок 3.

c) Ассиметричное изменение

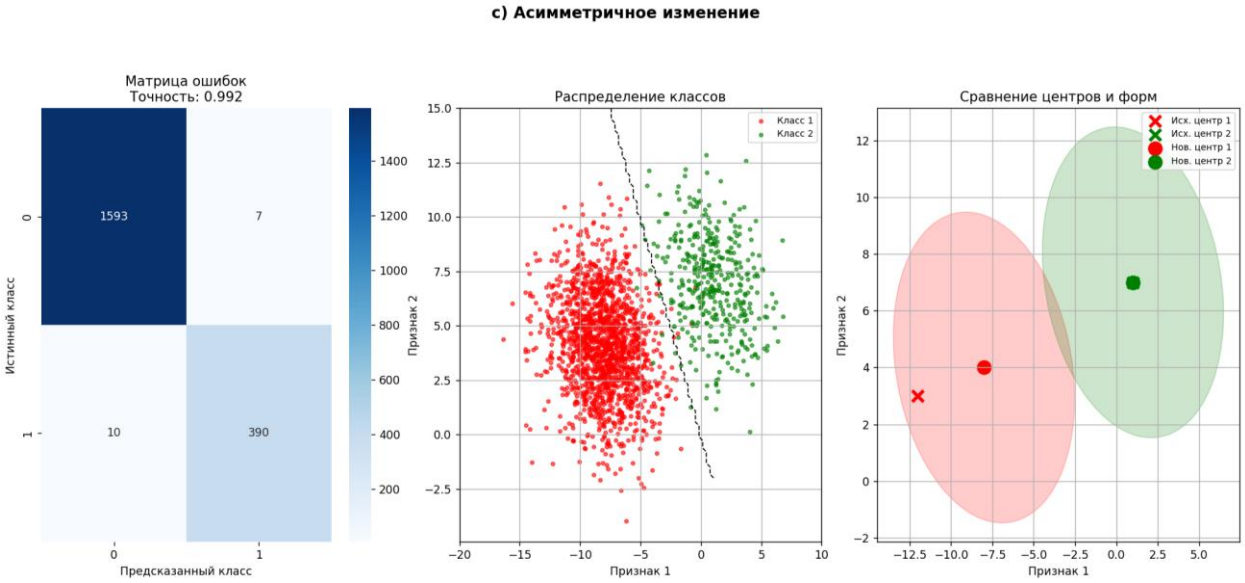

Рисунок 4.

d) Обратная ассиметрия

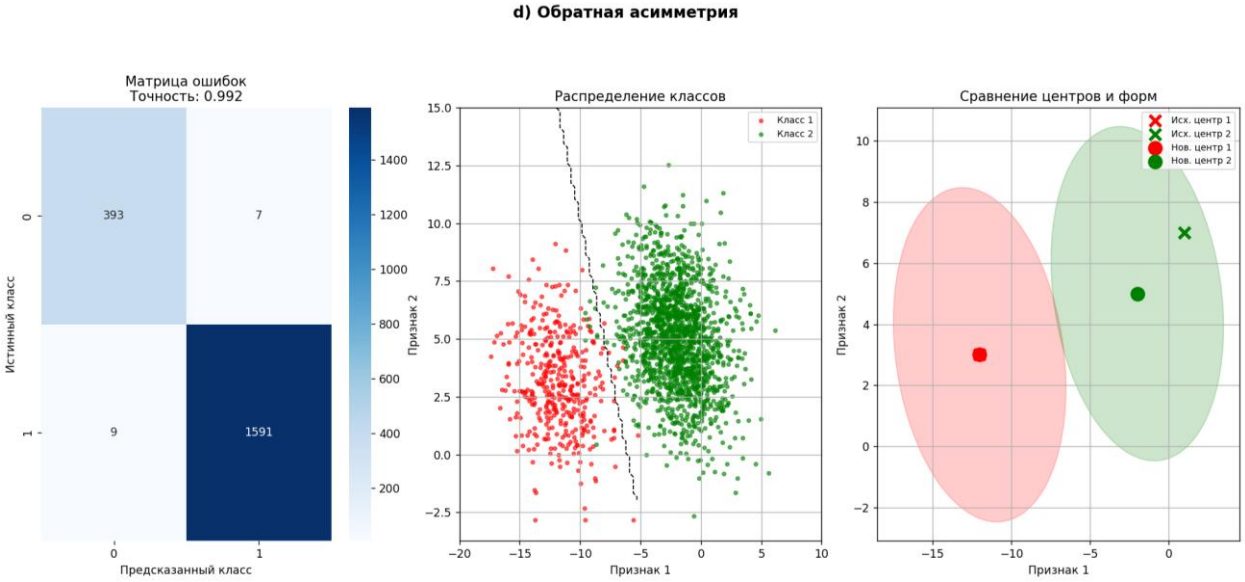

Рисунок 5.

e) Растяжение по горизонтали

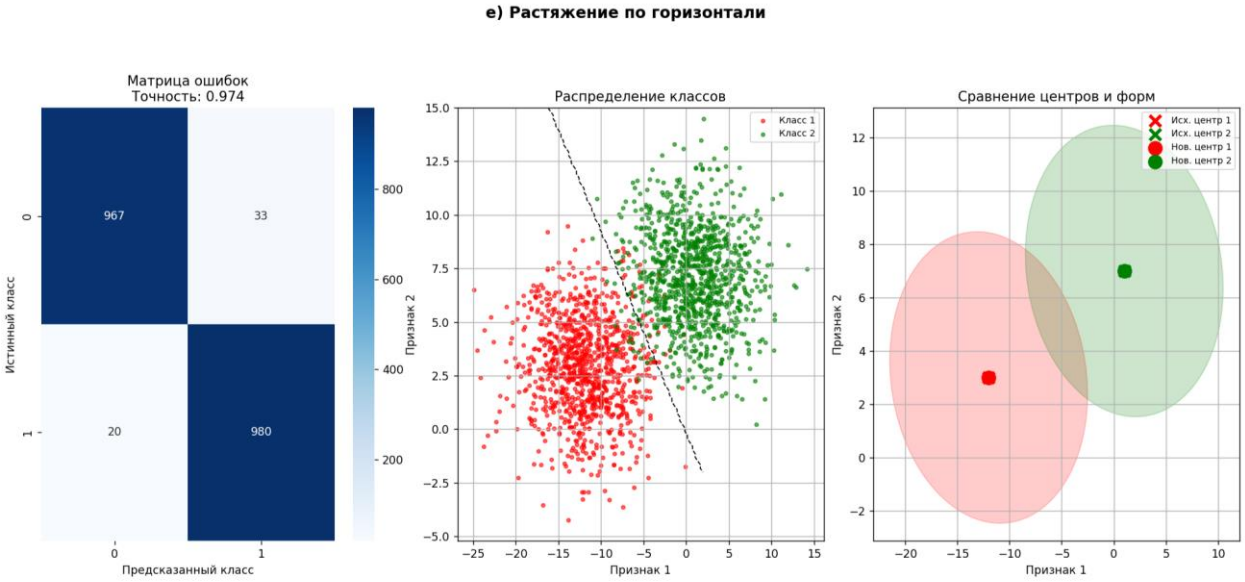

Рисунок 6.

f) Зеркальное отражение

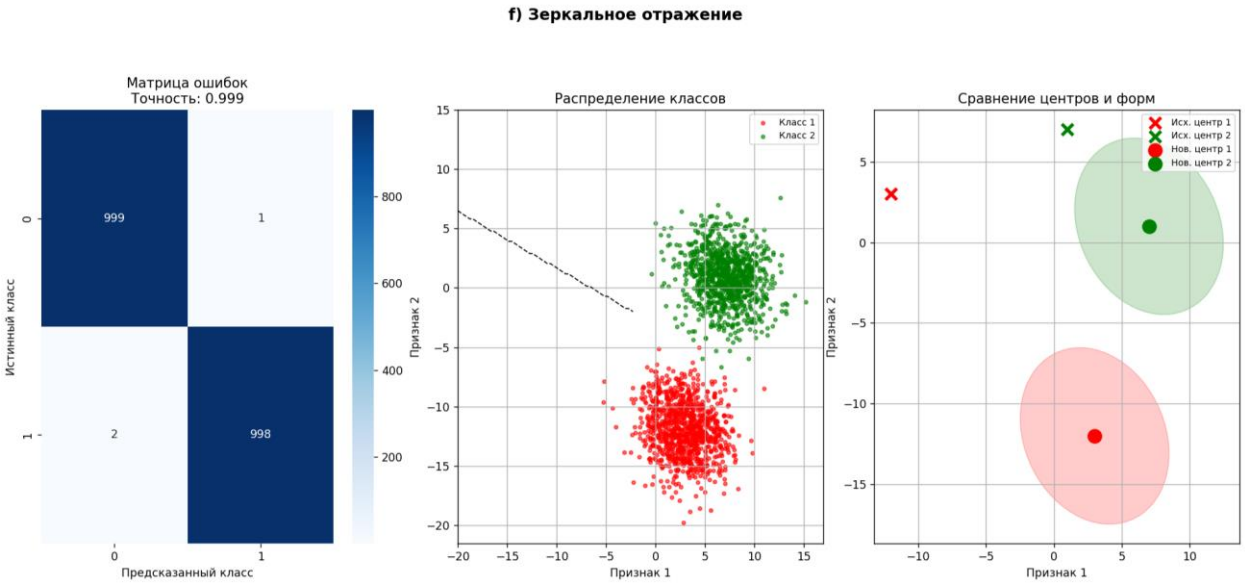

Рисунок 7.

Результаты и анализ ошибок

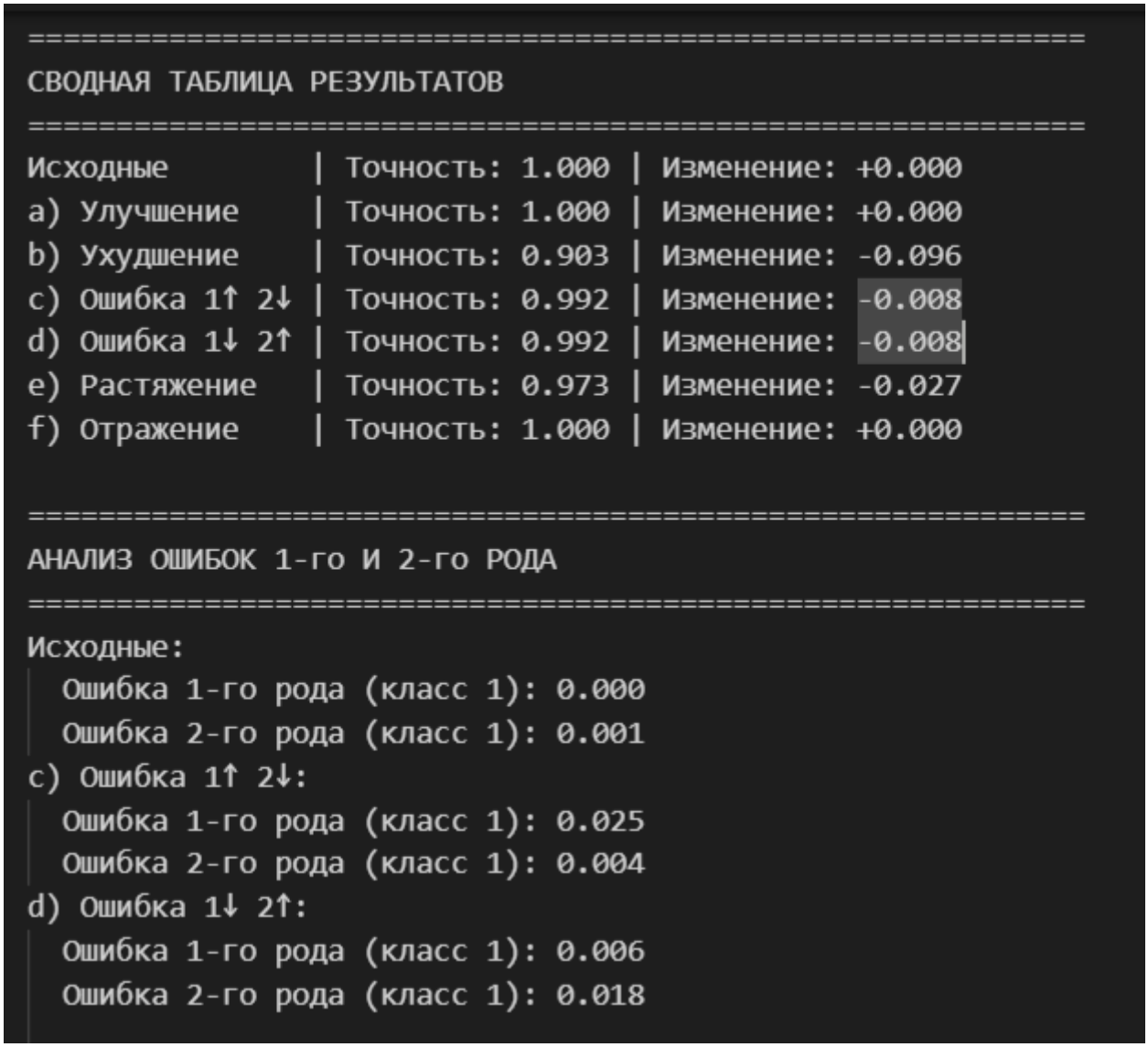

Рисунок 8.

Ответы на контрольные вопросы

1. Элементы главной диагонали матрицы ошибок характеризуют количество правильно классифицированных объектов каждого класса.
2. Элементы побочных диагоналей характеризуют ошибки классификации — количество объектов, которые были неправильно отнесены к другим классам.
3. Форма кластеров объектов определяется ковариационной матрицей распределения признаков внутри каждого класса.

Выводы

В ходе лабораторной работы был синтезирован и реализован на Python байесовский классификатор для гауссовских случайных векторов с одинаковыми ковариационными матрицами для двух классов. Экспериментально подтверждено, что алгоритм обеспечивает эффективное

разделение классов. Модификация параметров распределений (сдвиг центров классов, изменение ковариационной матрицы и априорных вероятностей) позволяет целенаправленно влиять на вероятности ошибок первого и второго рода, что соответствует теоретическим предпосылкам. Наибольшая точность классификации достигается при максимальном расстоянии между центрами классов, а наибольшая ошибка – при их сближении.