Мигачев Павел Игоревич, группа 8-1

Лабораторная работа № 3

**Вариант № 1**

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= [2 1], m2= [-1 1], C1= [3 -1; -1 3], C2= [5 2; 2 6].**

Построить график зависимости экспериментальной ошибки первого рода (для первого класса) от числа испытаний (объема выборки). Сравнить с теоретическим значением.

**Код программы (внесённые изменения в шаблон кода выделены)**

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  
  
m1 = np.array([ 2.0, 1.0]) *# центр класса 1*m2 = np.array([-1.0, 1.0]) *# центр класса 2*C1 = np.array([[3.0, -1.0],  
 [-1.0, 3.0]]) *# ковариация класса 1*C2 = np.array([[5.0, 2.0],  
 [2.0, 6.0]]) *# ковариация класса 2*p1 = 0.5 *# априорная вероятность класса 1*p2 = 0.5 *# априорная вероятность класса 2*np.random.seed(42)  
  
*# Объекты многомерных нормальных распределений*rv1 = multivariate\_normal(mean=m1, cov=C1)  
rv2 = multivariate\_normal(mean=m2, cov=C2)  
  
  
*# Байесовский классификатор (QDA)*def bayes\_classifier(x, rv1, rv2, p1, p2):  
 *# Сравниваем P(w1|x) ~ p(x|w1)P(w1) и P(w2|x) ~ p(x|w2)P(w2)  
 # Можно работать в pdf (вероятностях), SciPy сам стабилен для 2D* return 1 if (rv1.pdf(x) \* p1) >= (rv2.pdf(x) \* p2) else 2  
  
*# Теоретическая (MC) и экспериментальная СУММАРНЫЕ ошибки 1-го рода*def theoretical\_total\_error\_type1\_class1(rv1, rv2, p1, p2, sample\_size, mc\_n=100000):  
 *"""  
 MC-оценка теоретической вероятности ошибки 1-го рода:  
 P(решить класс 2 | истинный класс 1).  
 Возвращаем ожидаемое КОЛИЧЕСТВО ошибок для заданного sample\_size.  
 """* samples = rv1.rvs(size=mc\_n)  
 *# Векторизованно классифицируем* prob1 = rv1.pdf(samples) \* p1  
 prob2 = rv2.pdf(samples) \* p2  
 mis = (prob2 > prob1).sum()  
 prob = mis / mc\_n  
 return prob \* sample\_size *# ожидаемое число ошибок*def experimental\_total\_error\_type1\_class1(rv1, rv2, p1, p2, sample\_size, num\_trials=50):  
 *"""  
 Эксперимент: генерируем выборки из класса 1, считаем КОЛИЧЕСТВО ошибок (1-го рода) в каждой,  
 возвращаем среднее и СКО по повторам.  
 """* totals = []  
 for \_ in range(num\_trials):  
 samples = rv1.rvs(size=sample\_size)  
 *# Векторизованная классификация* prob1 = rv1.pdf(samples) \* p1  
 prob2 = rv2.pdf(samples) \* p2  
 err\_cnt = (prob2 > prob1).sum()  
 totals.append(err\_cnt)  
 totals = np.array(totals, dtype=float)  
 return totals.mean(), totals.std(ddof=1)  
  
def visualize\_original\_data(rv1, rv2, p1, p2, num\_samples=1000):  
 *# 1) Генерация тестовых данных (по 1000 из каждого класса)* samples1 = rv1.rvs(size=num\_samples)  
 samples2 = rv2.rvs(size=num\_samples)  
 X = np.vstack([samples1, samples2])  
 y\_true = np.array([1]\*num\_samples + [2]\*num\_samples)  
  
 *# Классификация (векторизованно)* prob1 = rv1.pdf(X) \* p1  
 prob2 = rv2.pdf(X) \* p2  
 y\_pred = np.where(prob1 >= prob2, 1, 2)  
  
 *# Матрица ошибок и точность* cm = confusion\_matrix(y\_true, y\_pred, labels=[1,2])  
 acc = np.trace(cm) / np.sum(cm)  
  
 *# Общая фигура* plt.figure(figsize=(15, 5))  
  
 *# (1) Матрица ошибок* plt.subplot(1, 3, 1)  
 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',  
 xticklabels=['Класс 1','Класс 2'],  
 yticklabels=['Класс 1','Класс 2'])  
 plt.title(f'Матрица ошибок\nТочность: {acc:.3f}')  
 plt.xlabel('Предсказанный класс')  
 plt.ylabel('Истинный класс')  
  
 *# Диапазоны для осей — по данным* all\_x = X[:,0]; all\_y = X[:,1]  
 x\_min, x\_max = all\_x.min()-2, all\_x.max()+2  
 y\_min, y\_max = all\_y.min()-2, all\_y.max()+2  
  
 *# (2) Облака точек + эллипсы* plt.subplot(1, 3, 2)  
 plt.scatter(samples1[:,0], samples1[:,1], s=10, alpha=0.6, label='Класс 1', color='red')  
 plt.scatter(samples2[:,0], samples2[:,1], s=10, alpha=0.6, label='Класс 2', color='blue')  
 *# центры* plt.scatter(m1[0], m1[1], color='darkred', s=100, marker='x', linewidth=3, label='Центр 1')  
 plt.scatter(m2[0], m2[1], color='darkblue', s=100, marker='x', linewidth=3, label='Центр 2')  
 *# эллипсы 95%* for mean, cov, color in zip([m1, m2], [C1, C2], ['red', 'blue']):  
 eigenvals, eigenvecs = np.linalg.eigh(cov) *# eigenvalues asc  
 # χ²(0.95; df=2) ≈ 5.991* width = 2\*np.sqrt(5.991\*eigenvals[1])  
 height = 2\*np.sqrt(5.991\*eigenvals[0])  
 angle = np.degrees(np.arctan2(eigenvecs[1,1], eigenvecs[0,1]))  
 plt.gca().add\_patch(Ellipse(xy=mean, width=width, height=height,  
 angle=angle, alpha=0.3, color=color, label=f'Эллипс 95%'))  
 plt.xlim(x\_min, x\_max); plt.ylim(y\_min, y\_max)  
 plt.xlabel('Признак 1 (X)')  
 plt.ylabel('Признак 2 (Y)')  
 plt.title('Распределения и эллипсы ковариации')  
 plt.legend(fontsize=8)  
 plt.grid(True, alpha=0.3)  
  
 *# (3) Разделяющая граница (QDA)* plt.subplot(1, 3, 3)  
 xs = np.linspace(x\_min, x\_max, 300)  
 ys = np.linspace(y\_min, y\_max, 300)  
 XX, YY = np.meshgrid(xs, ys)  
 grid = np.stack([XX.ravel(), YY.ravel()], axis=1)  
  
 *# Решаю по байесовскому правилу* P1\_grid = rv1.pdf(grid) \* p1  
 P2\_grid = rv2.pdf(grid) \* p2  
 Z = np.where(P1\_grid >= P2\_grid, 1, 2).reshape(XX.shape)  
  
 *# Граница по уровню 1.5* plt.contour(XX, YY, Z, levels=[1.5], colors='black', linestyles='dashed', linewidths=2)  
 *# Поля* plt.contourf(XX, YY, Z, levels=[0.5, 1.5, 2.5], colors=['red', 'blue'], alpha=0.15)  
  
 *# Центры и эллипсы добавим для наглядности* plt.scatter(m1[0], m1[1], color='darkred', s=100, marker='x', linewidth=3, label='Центр 1')  
 plt.scatter(m2[0], m2[1], color='darkblue', s=100, marker='x', linewidth=3, label='Центр 2')  
 for mean, cov, color in zip([m1, m2], [C1, C2], ['red', 'blue']):  
 eigenvals, eigenvecs = np.linalg.eigh(cov)  
 width = 2\*np.sqrt(5.991\*eigenvals[1])  
 height = 2\*np.sqrt(5.991\*eigenvals[0])  
 angle = np.degrees(np.arctan2(eigenvecs[1,1], eigenvecs[0,1]))  
 plt.gca().add\_patch(Ellipse(xy=mean, width=width, height=height,  
 angle=angle, alpha=0.25, color=color))  
 plt.xlim(x\_min, x\_max); plt.ylim(y\_min, y\_max)  
 plt.xlabel('Признак 1 (X)')  
 plt.ylabel('Признак 2 (Y)')  
 plt.title('Разделяющая граница (QDA)')  
 plt.grid(True, alpha=0.3)  
  
 plt.suptitle('Визуализация исходных данных и классификации', fontsize=14, fontweight='bold')  
 plt.tight\_layout()  
 plt.show()  
  
 return cm  
  
*# график "увеличить вероятность правильного распознавания"*def shift\_means\_apart(m1, m2, k=0.75):  
 *"""  
 Раздвигаем центры вдоль прямой между ними:  
 m1' = m1 + k\*(m1 - m2), m2' = m2 + k\*(m2 - m1)  
 k>0 — чем больше, тем дальше центры.  
 """* v = m1 - m2  
 m1\_new = m1 + k \* v  
 m2\_new = m2 - k \* v  
 return m1\_new, m2\_new  
  
def confusion\_and\_acc(samples1, samples2, rv1, rv2, p1, p2):  
 *"""Векторно считаем матрицу ошибок и точность для пары облаков."""* X = np.vstack([samples1, samples2])  
 y\_true = np.array([1]\*len(samples1) + [2]\*len(samples2))  
 prob1 = rv1.pdf(X) \* p1  
 prob2 = rv2.pdf(X) \* p2  
 y\_pred = np.where(prob1 >= prob2, 1, 2)  
 cm = confusion\_matrix(y\_true, y\_pred, labels=[1,2])  
 acc = np.trace(cm) / np.sum(cm)  
 return cm, acc, X  
  
def add\_cov\_ellipse(ax, mean, cov, color, alpha=0.25):  
 *"""Рисуем 95%-эллипс (χ²\_2(0.95)≈5.991)."""* vals, vecs = np.linalg.eigh(cov)  
 width = 2\*np.sqrt(5.991\*vals[1])  
 height = 2\*np.sqrt(5.991\*vals[0])  
 angle = np.degrees(np.arctan2(vecs[1,1], vecs[0,1]))  
 ax.add\_patch(Ellipse(mean, width, height, angle=angle,  
 facecolor=color, edgecolor=color, alpha=alpha))  
  
def plot\_decision(ax, rv1, rv2, p1, p2, xlim, ylim):  
 *"""Контур QDA-границы (g1=g2)."""* xs = np.linspace(xlim[0], xlim[1], 300)  
 ys = np.linspace(ylim[0], ylim[1], 300)  
 XX, YY = np.meshgrid(xs, ys)  
 grid = np.stack([XX.ravel(), YY.ravel()], axis=1)  
 Z = np.where(rv1.pdf(grid)\*p1 >= rv2.pdf(grid)\*p2, 1, 2).reshape(XX.shape)  
 ax.contour(XX, YY, Z, levels=[1.5], colors='k', linestyles='--', linewidths=2)  
  
def plot\_improvement(m1, m2, C1, C2, p1, p2, num\_samples=1000, k\_shift=0.75):  
 *"""  
 Две панели: слева — исходные параметры, справа — улучшенные (центры раздвинуты).  
 Печатаем точность и рисуем всё в стиле ЛР-2.  
 """  
 # Исходные распределения* rv1\_orig = multivariate\_normal(mean=m1, cov=C1)  
 rv2\_orig = multivariate\_normal(mean=m2, cov=C2)  
 samp1\_o = rv1\_orig.rvs(size=num\_samples)  
 samp2\_o = rv2\_orig.rvs(size=num\_samples)  
 cm\_o, acc\_o, Xo = confusion\_and\_acc(samp1\_o, samp2\_o, rv1\_orig, rv2\_orig, p1, p2)  
  
 *# Улучшенные (раздвинули центры)* m1\_new, m2\_new = shift\_means\_apart(m1, m2, k=k\_shift)  
 rv1\_new = multivariate\_normal(mean=m1\_new, cov=C1)  
 rv2\_new = multivariate\_normal(mean=m2\_new, cov=C2)  
 samp1\_n = rv1\_new.rvs(size=num\_samples)  
 samp2\_n = rv2\_new.rvs(size=num\_samples)  
 cm\_n, acc\_n, Xn = confusion\_and\_acc(samp1\_n, samp2\_n, rv1\_new, rv2\_new, p1, p2)  
  
 *# Общие пределы осей* allX = np.vstack([Xo, Xn])  
 xlim = (allX[:,0].min()-2, allX[:,0].max()+2)  
 ylim = (allX[:,1].min()-2, allX[:,1].max()+2)  
  
 *# Фигура: 1x2* fig, axes = plt.subplots(1, 2, figsize=(14, 6))  
  
 *# ---- ЛЕВО: исходные* ax = axes[0]  
 ax.scatter(samp1\_o[:,0], samp1\_o[:,1], s=10, alpha=0.6, label='Класс 1', color='red')  
 ax.scatter(samp2\_o[:,0], samp2\_o[:,1], s=10, alpha=0.6, label='Класс 2', color='blue')  
 ax.scatter(m1[0], m1[1], s=100, marker='x', linewidth=3, color='darkred', label='Центр 1')  
 ax.scatter(m2[0], m2[1], s=100, marker='x', linewidth=3, color='darkblue', label='Центр 2')  
 add\_cov\_ellipse(ax, m1, C1, 'red', 0.22)  
 add\_cov\_ellipse(ax, m2, C2, 'blue', 0.22)  
 plot\_decision(ax, rv1\_orig, rv2\_orig, p1, p2, xlim, ylim)  
 sns.heatmap(cm\_o, annot=True, fmt='d', cmap='Blues',  
 cbar=False, xticklabels=['Кл.1','Кл.2'], yticklabels=['Кл.1','Кл.2'],  
 ax=ax.inset\_axes([0.02, 0.62, 0.33, 0.33]))  
 ax.set\_xlim(\*xlim); ax.set\_ylim(\*ylim)  
 ax.set\_title(f'Исходные параметры\nТочность: {acc\_o:.3f}')  
 ax.set\_xlabel('Признак 1'); ax.set\_ylabel('Признак 2')  
 ax.grid(True, alpha=0.3)  
 ax.legend(loc='upper right', fontsize=8)  
  
 *# ---- ПРАВО: улучшенные* ax = axes[1]  
 ax.scatter(samp1\_n[:,0], samp1\_n[:,1], s=10, alpha=0.6, label='Класс 1', color='red')  
 ax.scatter(samp2\_n[:,0], samp2\_n[:,1], s=10, alpha=0.6, label='Класс 2', color='blue')  
 ax.scatter(m1\_new[0], m1\_new[1], s=100, marker='x', linewidth=3, color='darkred', label='Центр 1 (нов.)')  
 ax.scatter(m2\_new[0], m2\_new[1], s=100, marker='x', linewidth=3, color='darkblue', label='Центр 2 (нов.)')  
 add\_cov\_ellipse(ax, m1\_new, C1, 'red', 0.22)  
 add\_cov\_ellipse(ax, m2\_new, C2, 'blue', 0.22)  
 plot\_decision(ax, rv1\_new, rv2\_new, p1, p2, xlim, ylim)  
 sns.heatmap(cm\_n, annot=True, fmt='d', cmap='Greens',  
 cbar=False, xticklabels=['Кл.1','Кл.2'], yticklabels=['Кл.1','Кл.2'],  
 ax=ax.inset\_axes([0.02, 0.62, 0.33, 0.33]))  
 ax.set\_xlim(\*xlim); ax.set\_ylim(\*ylim)  
 ax.set\_title(f'Раздвинули центры (k={k\_shift})\nТочность: {acc\_n:.3f}')  
 ax.set\_xlabel('Признак 1'); ax.set\_ylabel('Признак 2')  
 ax.grid(True, alpha=0.3)  
 ax.legend(loc='upper right', fontsize=8)  
  
 plt.suptitle('Увеличение вероятности правильного распознавания: ДО vs ПОСЛЕ', fontsize=14, fontweight='bold')  
 plt.tight\_layout()  
 plt.show()  
  
 *# Напечатаем сводку в консоль* print('=== Сводка улучшения ===')  
 print('Исходные: точность = {:.4f}, матрица ошибок:\n{}'.format(acc\_o, cm\_o))  
 print('Улучшенные: точность = {:.4f}, матрица ошибок:\n{}'.format(acc\_n, cm\_n))  
  
*# Запуск визуализации + исследовательская часть*if \_\_name\_\_ == "\_\_main\_\_":  
 cm = visualize\_original\_data(rv1, rv2, p1, p2)  
 plot\_improvement(m1, m2, C1, C2, p1, p2, num\_samples=1000, k\_shift=0.75)  
  
 *# Размеры выборок (как у друга)* sample\_sizes = [50, 100, 200, 500, 1000, 2000, 5000]  
 differences = []  
 difference\_stds = []  
  
 print("Объем выборки | Разность (эксп - теор) | СКО эксперимента")  
 print("-" \* 60)  
  
 for size in sample\_sizes:  
 exp\_total\_error, exp\_std = experimental\_total\_error\_type1\_class1(rv1, rv2, p1, p2, size, num\_trials=50)  
 theor\_total\_error = theoretical\_total\_error\_type1\_class1(rv1, rv2, p1, p2, size, mc\_n=100000)  
  
 diff = exp\_total\_error - theor\_total\_error  
 differences.append(diff)  
 difference\_stds.append(exp\_std)  
  
 print(f"{size:13d} | {diff:+8.2f} | {exp\_std:6.2f}")  
  
 *# График разности суммарных ошибок (эксп - теор) c СКО* plt.figure(figsize=(12, 8))  
 plt.errorbar(sample\_sizes, differences, yerr=difference\_stds,  
 fmt='o-', linewidth=2, markersize=6, capsize=6, capthick=2,  
 label='Разность суммарных ошибок (эксп − теор) ± СКО', color='green', alpha=0.8)  
 plt.axhline(0, color='red', linestyle='--', linewidth=2, label='Идеальное совпадение')  
 plt.xscale('log')  
 plt.xlabel('Объем выборки (число испытаний)', fontsize=12)  
 plt.ylabel('Разность суммарных ошибок 1-го рода', fontsize=12)  
 plt.title('Разность суммарной экспериментальной и теоретической\nошибок первого рода (класс 1) от объема выборки', fontsize=14)  
 plt.grid(True, which='both', alpha=0.3)  
 plt.legend(fontsize=10)  
 plt.tight\_layout()  
 plt.show()

**Результаты выполнения задания**



Рисунок 1.

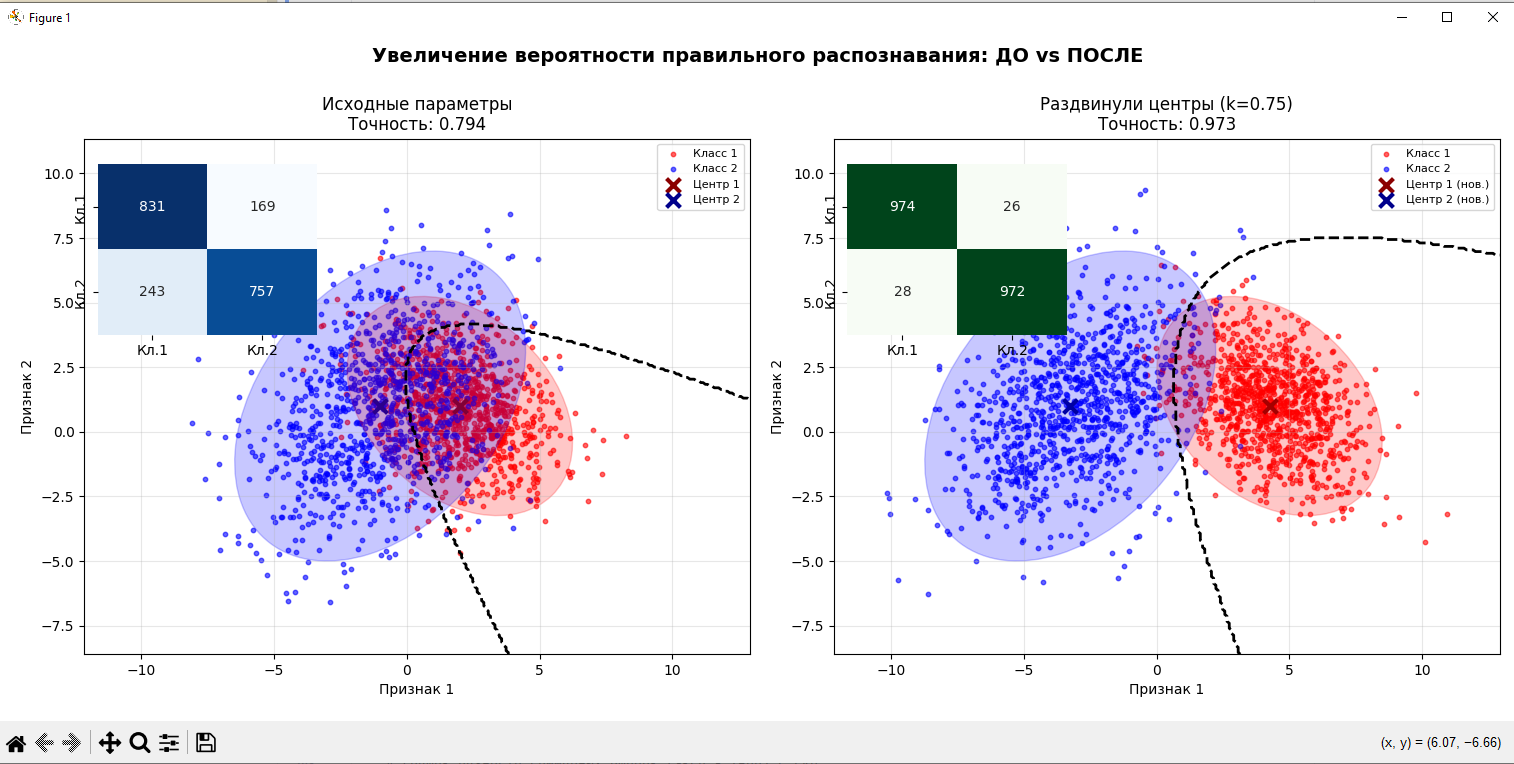


Рисунок 2.

График разности суммарной экспериментальной и теоретической ошибок первого рода (для первого класса) от числа испытаний (объема выборки).

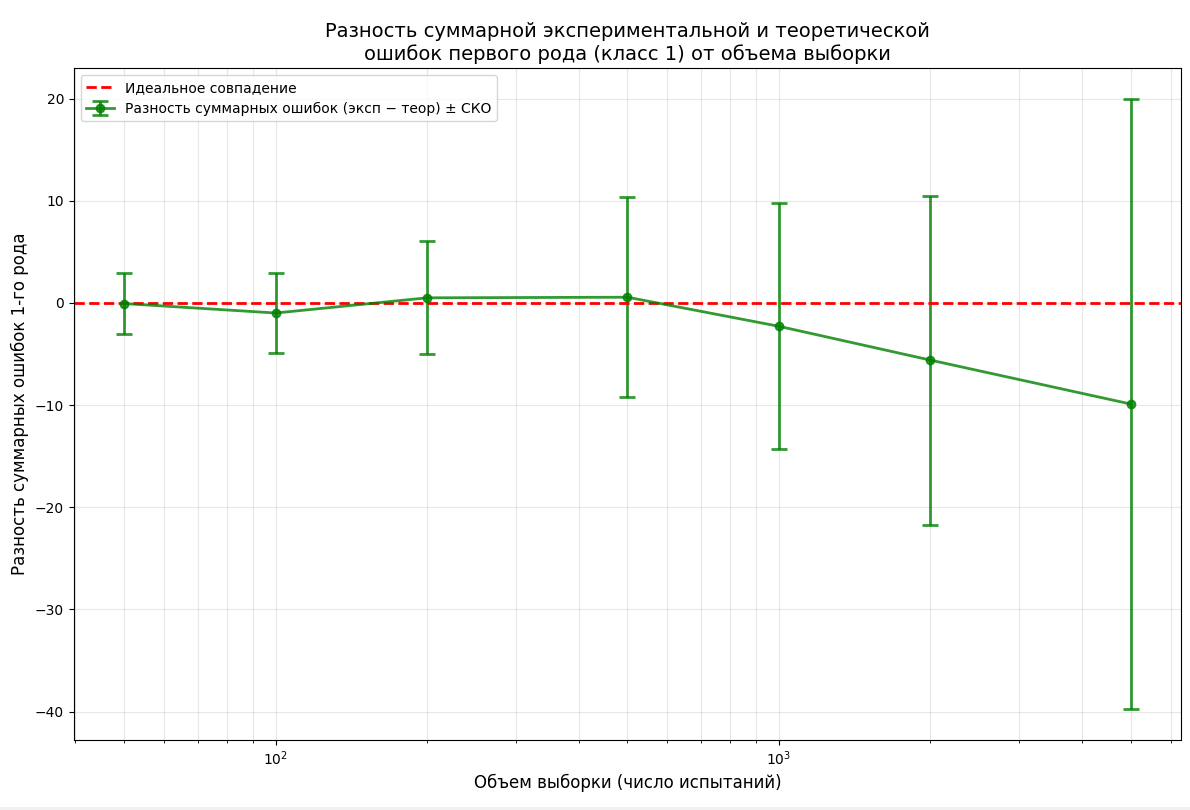


Рисунок 3.

Результаты и анализ ошибок

  
Рисунок 4.

# Ответы на контрольные вопросы

1. Элементы главной диагонали матрицы ошибок характеризуют количество правильно классифицированных объектов каждого класса.
2. Элементы побочных диагоналей (вне главной диагонали) 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 реализован байесовский классификатор для двух гауссовских классов при равных априорных вероятностях. Исследована зависимость экспериментальной ошибки первого рода для первого класса от объёма выборки. В результате эксперимента установлено, что с увеличением числа испытаний значение ошибки постепенно стабилизируется и стремится к теоретическому уровню. Наблюдается уменьшение разброса экспериментальных данных при росте объёма выборки, что подтверждает состоятельность алгоритма. Визуализация распределений, доверительных эллипсов и решающей границы показала влияние различий в ковариационных матрицах на форму областей локализации классов и характер разделения между ними.