МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра САПР

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Автоматизация схемотехнического проектирования»
Тема: **«Генерация модельных наборов данных»**

Студент гр. 2301 _____ Комиссаров П.Е.

Преподаватель: _____ Боброва Ю.О.

Санкт-Петербург
2026

Цель работы:

Получение навыков работы с numpy-массивами и написание функций на языке Python на примере генерации массивов произвольно распределенных данных.

Основные теоретические положения

Нормальное распределение. Когда на величину действует куча случайных факторов, значения часто кучкуются около какого-то среднего (μ), а разброс задаётся стандартным отклонением σ.

На графике это выглядит как колокол. В NumPy такие числа получают через np.random.normal(mu, sigma, size).

Объект, признак, класс, метка.

Объект — это одна строка в таблице (одно наблюдение).

Признак — столбец, одно свойство.

Класс — группа объектов (у нас два: 0 и 1).

Метка Y говорит, к какому классу объект относится. Важно: порядок меток в Y должен совпадать с порядком строк в X, иначе объект и его метка разъедутся.

Обучающая и тестовая выборки. Данные делим на две части. На одной (обучающей) алгоритм «учится», на другой (тестовой) мы смотрим, как он работает на новых данных.

В работе взято соотношение 70% на обучение и 30% на тест. Перед разбиением данные обязательно перемешиваем, чтобы в обе части попали объекты обоих классов.

Визуализация. Гистограммы — по горизонтали интервалы значений признака, по вертикали сколько объектов попало. По тому, как перекрываются столбики двух классов, видно, насколько они смешаны по этому признаку.

Скатерограммы — это просто точки на плоскости по двум признакам; по ним видно, как классы расположены друг относительно друга и можно ли их разделить прямой (или нет).

Линейная и нелинейная разделимость. Если два класса можно разделить одной прямой (в 2D) или плоскостью (в 3D и выше) — данные линейно разделимы.

Если один класс сидит внутри другого (кольцо, оболочка и т.д.), одной прямой не обойтись — нужны нелинейные методы или ядра.

Ход выполнения работы

Этапы 1–2. Импорты и параметры

В main подключаем numpy и matplotlib, создаём папку для графиков. Потом задаём параметры: сколько объектов в классе (N), средние mu0, mu1 и сигмы sigma0, sigma1 для каждого признака. В методичке пример на трёх признаках — так и сделано, всё собрано в списки mu и sigma, чтобы передать в функцию.

```
# --- Параметры по методичке (3 признака) ---
N = 1000
mu0 = [0, 2, 3]
mu1 = [3, 5, 1]
sigma0 = [2, 1, 2]
sigma1 = [1, 2, 1]
mu = [mu0, mu1]
sigma = [sigma0, sigma1]
```

Этапы 3–5 и 9. Функция norm_dataset (DataGenerator.py)

Вся генерация нормальных данных вынесена в отдельный файл DataGenerator.py.

Функция norm_dataset(mu, sigma, N) принимает средние и сигмы для двух классов и N.

По каждому признаку для каждого класса вызывается np.random.normal, столбцы добавляются через np.hstack.

Метки — просто нули и единицы (bool), склеивание классов через np.vstack, для Y ещё делается .ravel(), чтобы получить одномерный массив.

Внутри функции перемешиваются индексы и этим же порядком переставляются строки в X и Y — так метки не отстают от объектов. Возвращаются X, Y и отдельно class0, class1 (они без перемешивания, для графиков).

```
def norm_dataset(mu, sigma, N):
    """
    Генерирует данные двух классов с нормальным распределением по каждому
признаку
    """
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]
    col = len(mu0)

    class0 = np.random.normal(mu0[0], sigma0[0], (N, 1))
    class1 = np.random.normal(mu1[0], sigma1[0], (N, 1))
    for i in range(1, col):
        v0 = np.random.normal(mu0[i], sigma0[i], (N, 1))
        class0 = np.hstack((class0, v0))
        v1 = np.random.normal(mu1[i], sigma1[i], (N, 1))
        class1 = np.hstack((class1, v1))

    Y0 = np.zeros((N, 1), dtype=bool)
    Y1 = np.ones((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()

    rng = np.random.default_rng()
    arr = np.arange(2 * N)
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

Этап 6. Разбиение на train и test

Перемешивание уже сделано в norm_dataset, поэтому в main просто считается размер обучающей выборки:

train_count = round(0.7 * 2 * N) и режу X и Y на две части — первые 70% в train, остальное в test.

```
# Разбиение на обучающую и тестовую подвыборки 70/30
train_count = round(0.7 * 2 * N)
Xtrain = X[0:train_count]
Xtest = X[train_count : 2 * N]
Ytrain = Y[0:train_count]
Ytest = Y[train_count : 2 * N]
```

Этапы 7–8. Визуализация

Сделаны гистограммы по каждому из трёх признаков (на каждом графике оба класса, подписи осей и заголовок) и одна скатерограмма по признакам 1 и 3.

Всё сохраняется в папку figures/. Ниже сами картинки и что на них видно.

Гистограммы по признакам 1–3 (hist_1, hist_2, hist_3): форма похожа на нормальное распределение, по перекрытию столбиков видно, насколько классы пересекаются по этому признаку.

```
# Гистограммы по каждому признаку
for i in range(col):
    plt.figure()
    plt.hist(class0[:, i], bins="auto", alpha=0.7, label="Класс 0")
    plt.hist(class1[:, i], bins="auto", alpha=0.7, label="Класс 1")
    plt.xlabel("Значение признака")
    plt.ylabel("Частота")
    plt.title(f"Гистограмма признака {i + 1}")
    plt.legend()
    plt.savefig(FIGURES_DIR / f"hist_{i + 1}.png")
    plt.show()

# Скатерограмма
plt.figure()
plt.scatter(class0[:, 0], class0[:, 2], marker=".", alpha=0.7, label="Класс 0")
plt.scatter(class1[:, 0], class1[:, 2], marker=".", alpha=0.7, label="Класс 1")
plt.xlabel("Признак 1")
plt.ylabel("Признак 3")
plt.title("Скатерограмма: признаки 1 и 3 (нормальный датасет)")
plt.legend()
plt.savefig(FIGURES_DIR / "scatter_norm.png")
plt.show()
```

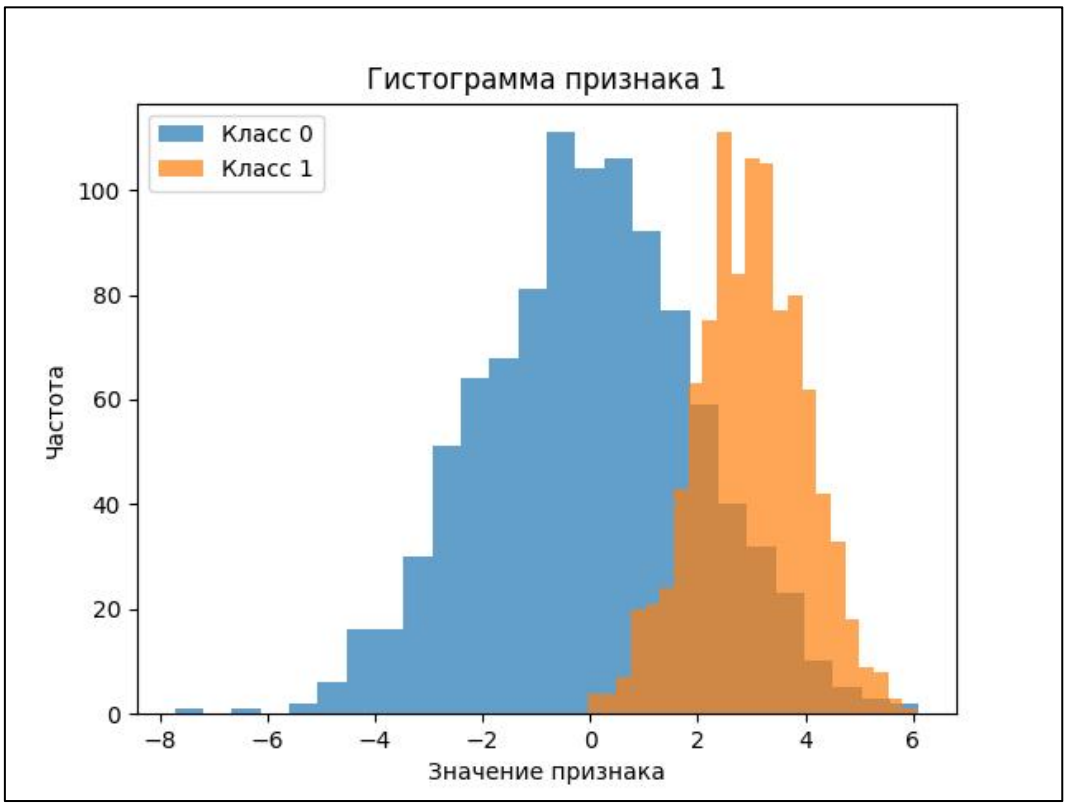

Рисунок 1 - Гистограмма признака 1

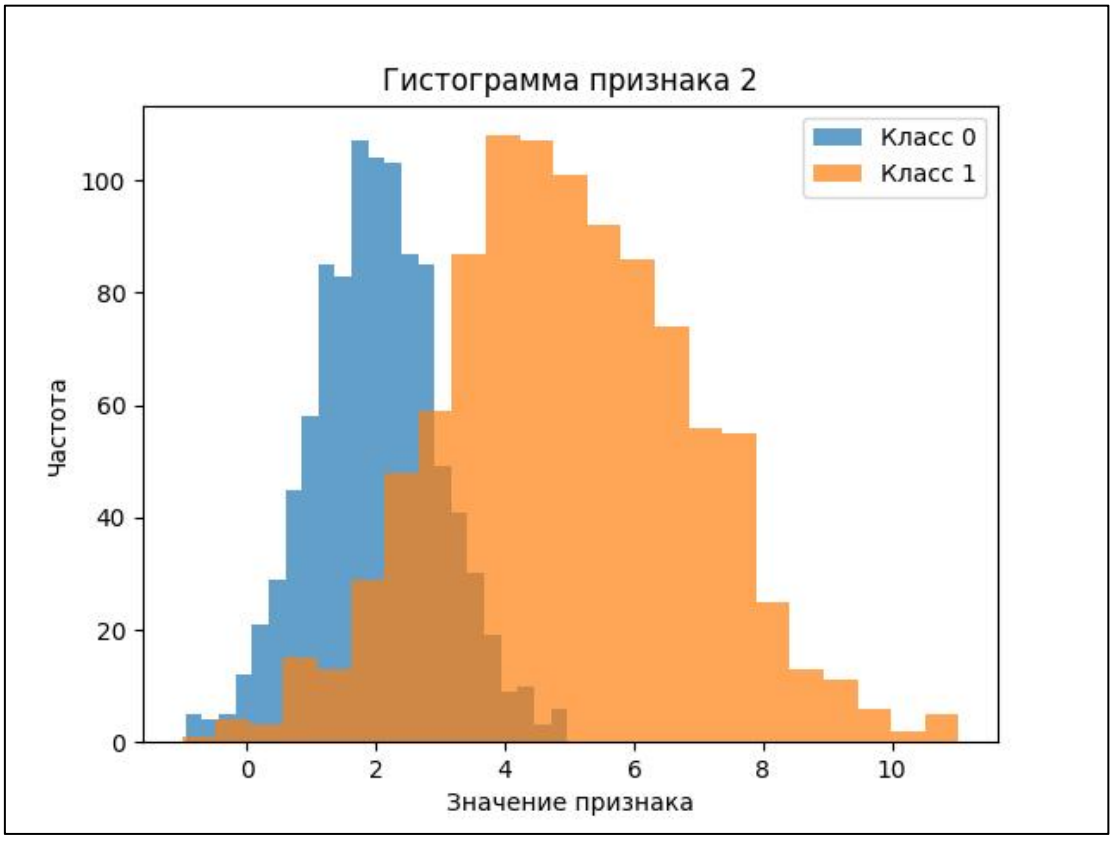

Рисунок 2 - Гистограмма признака 2

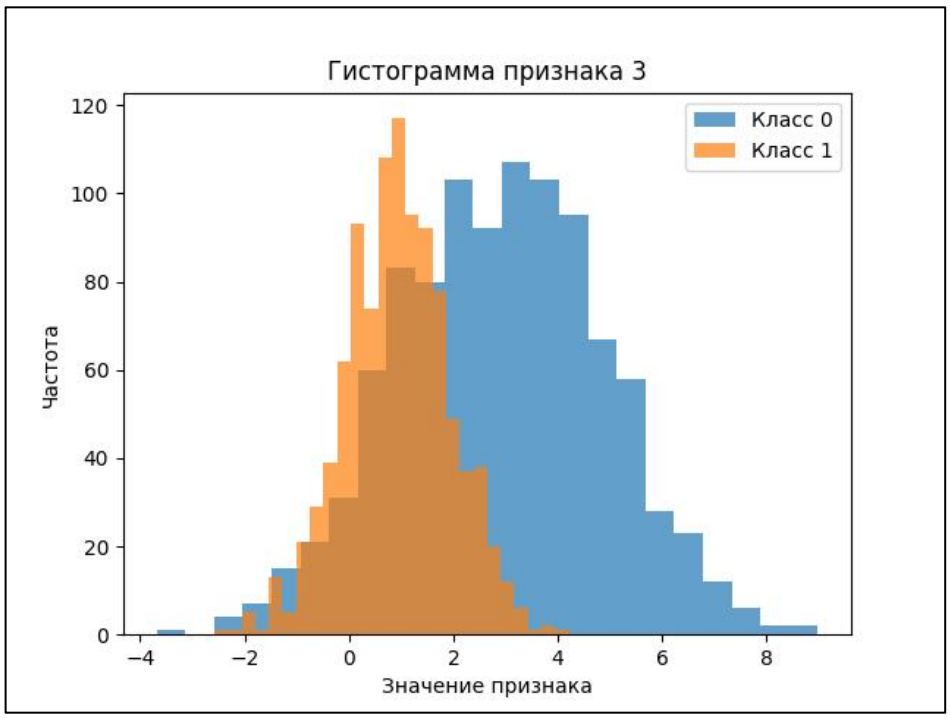

Рисунок 3 - Гистограмма признака 3

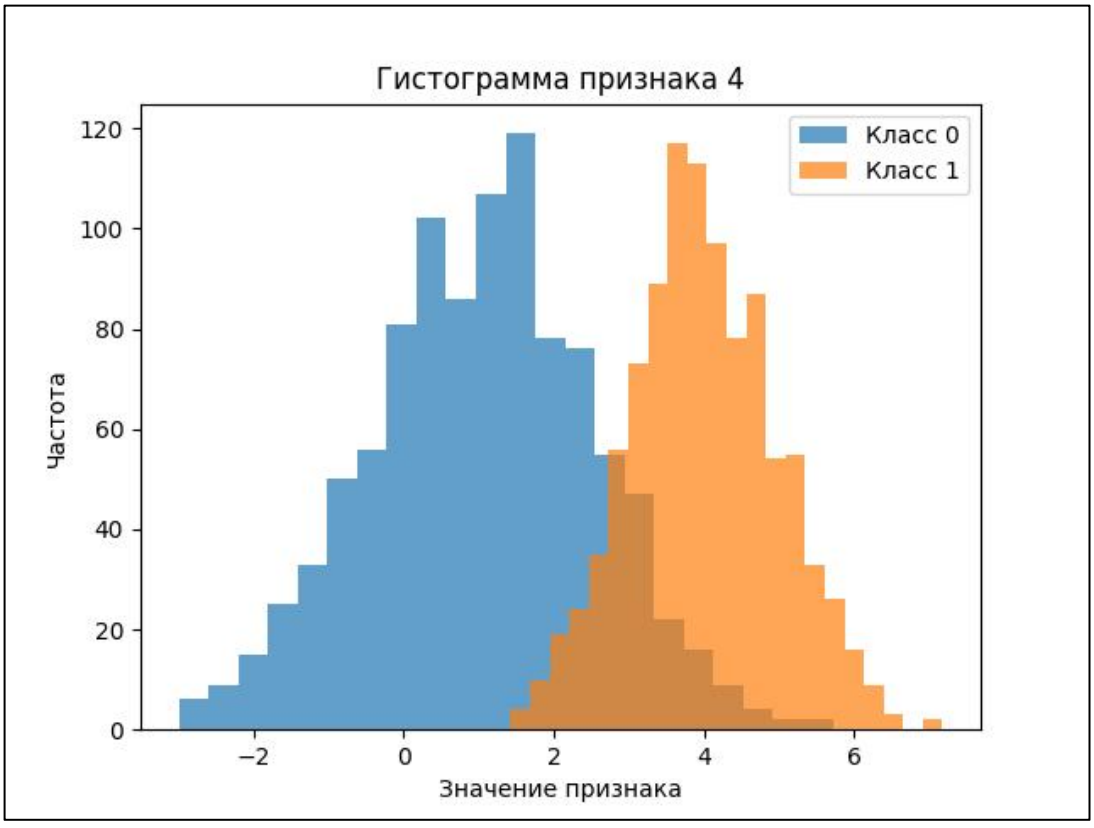

Рисунок 4 - Гистограмма признака 4

Скатерограмма нормального датасета (scatter_norm): по осям признаки 1 и 3. Два облака точек — классы частично перекрываются, но в целом их можно было бы разделить прямой (при таких mu и sigma).

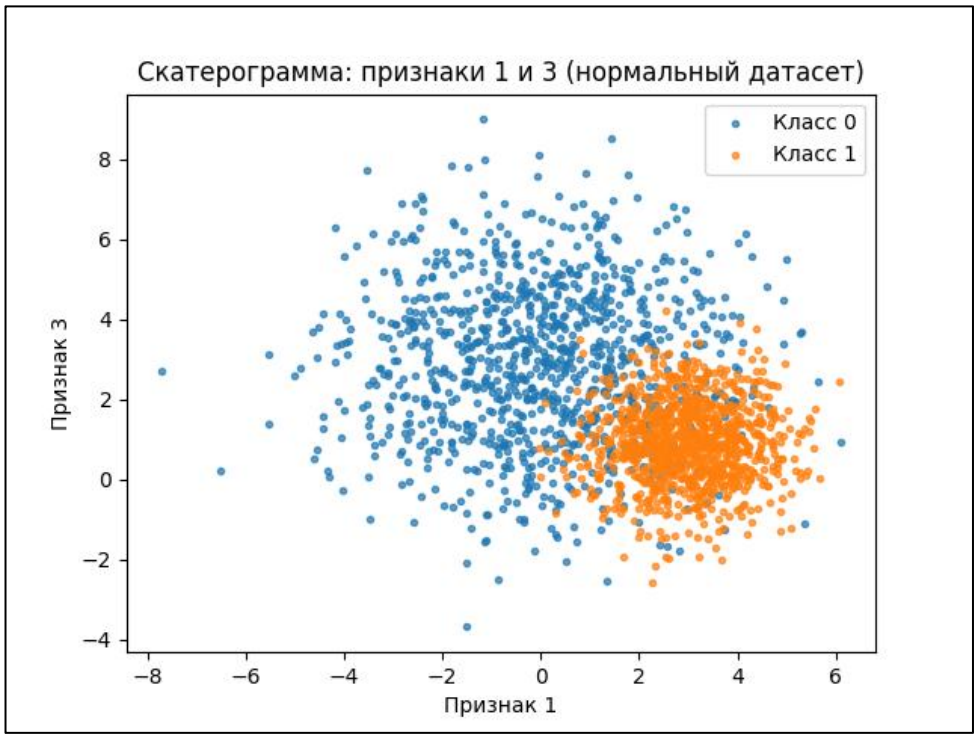

Рисунок 5 - Скатерограмма: признаки 1 и 3

Самостоятельное задание. nonlinear_dataset_5

По варианту 5 из приложения А нужно было сделать двумерные данные заданной формы.

У меня класс 1 — овал внутри, класс 0 — как бы U-образная оболочка вокруг (эллипс с вырезом).

Генерировал методом отбора: кидаю случайные точки в прямоугольник, проверяю — попала ли точка в нужную область.

Для класса 1 — просто внутри эллипса.

Для класса 0 — внутри большого эллипса, но снаружи внутренней границы и не в «вырезе». К каждой точке добавил немного шума.

Формат такой же, как у norm_dataset: X, Y, class0, class1, всё перемешано.

В main вызывается dg.nonlinear_dataset_5(N), строю скатерограмму и сохраняю в scatter_nonlinear_5.png.

```python
def nonlinear_dataset_5(N=1000):

     rng = np.random.default_rng()
     # Шум при генерации: добавляется к каждой точке. Больше — размытее границы.
     noise = 0.04

    # ========== Класс 0 (U-образная оболочка) ==========
    # cx0, cy0 — центр оболочки
    cx0, cy0 = cx + 0.2, cy + 0.12
    # a2, b2 — полуоси внешнего эллипса оболочки
    a2, b2 = 1.2, 0.50
    # a_inner_gap, b_inner_gap — полуоси внутренней границы оболочки.
    a_inner_gap, b_inner_gap = 0.5, 0.21
    # u_cut — граница «выреза» U в локальных координатах оболочки (ось u вдоль
большой оси).
    #   Точки с u < u_cut отбрасываются → получается открытие U. Больше u_cut
(например 0) — вырез шире,
    #   оболочка короче слева. Меньше u_cut (например -0.5) — вырез уже, оболочка
длиннее слева.
    u_cut = -0.25

    class0_list = []
    while len(class0_list) < N:
        x = rng.uniform(cx0 - a2 - 0.15, cx0 + a2 + 0.15)
        y = rng.uniform(cy0 - b2 - 0.15, cy0 + b2 + 0.15)
        u, v = to_local(x, y, cx0, cy0)
        # Внутри внешнего эллипса, снаружи внутренней границы оболочки, и не в
«вырезе» (u >= u_cut)
        in_outer = inside_ellipse(x, y, a2, b2, cx0, cy0)
        in_inner = inside_ellipse(x, y, a_inner_gap, b_inner_gap, cx0, cy0)
        not_in_cut = u >= u_cut
        if in_outer and not in_inner and not_in_cut:
            x += rng.normal(0, noise)
            y += rng.normal(0, noise)
            class0_list.append([x, y])
    class0 = np.array(class0_list)

    Y0 = np.zeros((N,), dtype=bool)
    Y1 = np.ones((N,), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.concatenate((Y0, Y1))

    arr = np.arange(2 * N)
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

Скатерограмма нелинейного датасета: видно, что один класс сидит внутри другого — прямой тут не разделить, нужны нелинейные методы.

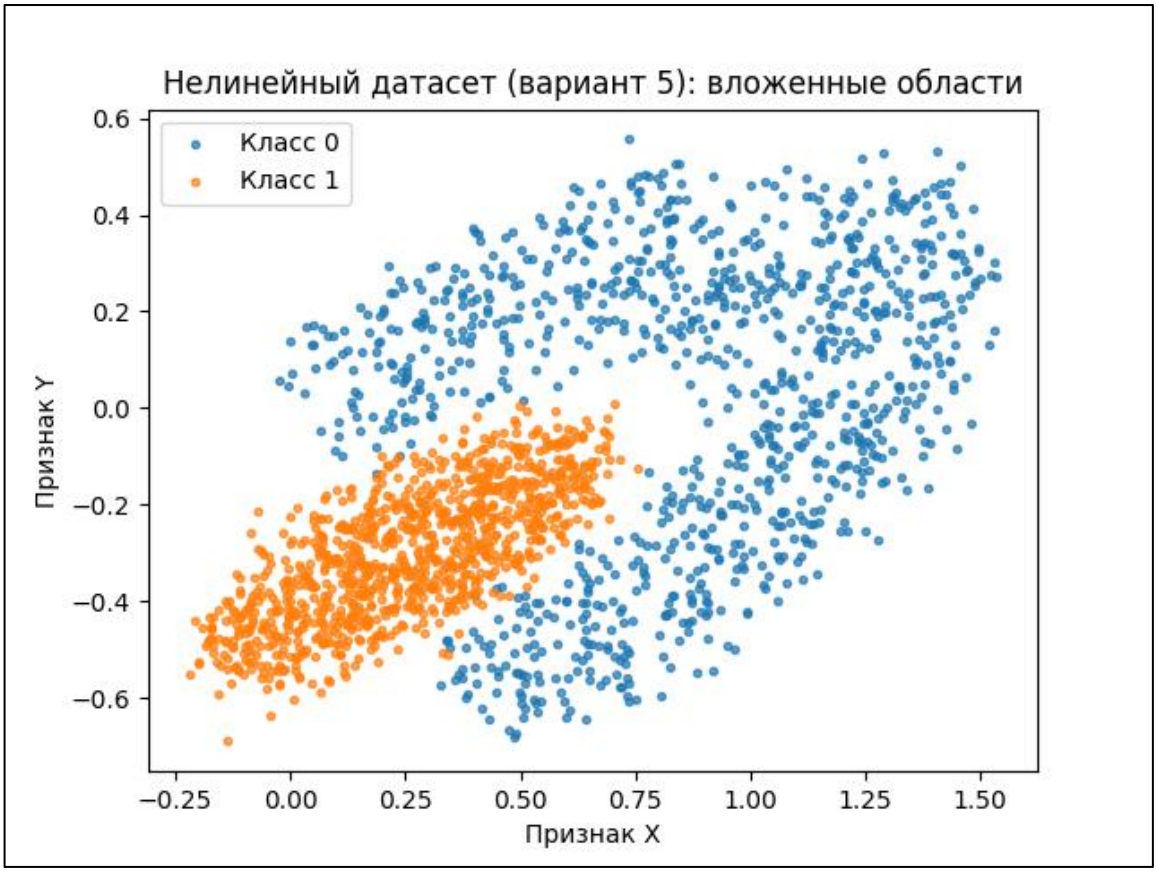

Рисунок 6 - Скатерограмма: Нелинейный датасет

Исходное изображение:

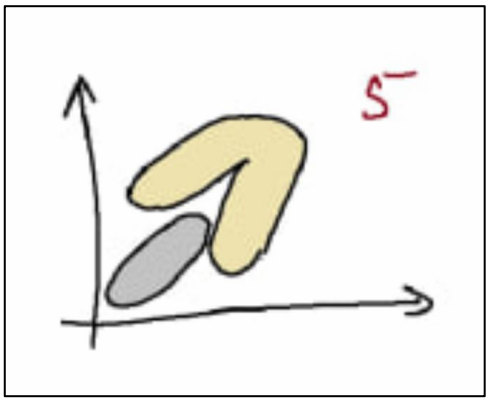

Рисунок 7 - Скатерограмма: Пример

Вывод

В работе получилось сгенерировать два набора данных: один с нормальным распределением по трём признакам (с заданными средними и сигмами для двух классов), второй — двумерный нелинейный по варианту 5 (овал внутри U-образной оболочки).

Гистограммы и скатерограммы хорошо показывают, как данные распределены и насколько классы перемешаны. По нормальному набору видно, что облака частично перекрываются — линейный классификатор где-то будет ошибаться. По нелинейному видно, что один класс внутри другого, одной прямой не разделить — тут уже нужны нелинейные методы или ядра. То есть визуализация реально помогает понять, подойдёт ли линейный метод или нет.

Данные перед разбиением перемешивались, потом резались 70/30 на обучение и тест — так в обеих частях есть объекты обоих классов. Генерацию вынес в DataGenerator.py, чтобы можно было подключать эти функции в других лабах без копирования кода.

Numpy и matplotlib хватило для всего: и массивы собирать, и метки, и перемешивание, и графики. Наборы можно дальше использовать для проверки разных классификаторов и смотреть, какие метрики лучше отражают, подходят ли данные под выбранный метод.

Код программы:

Файл DataGenerator.py

```
"""
Генерация модельных наборов данных
"""
import numpy as np


def norm_dataset(mu, sigma, N):
    """
    Генерирует данные двух классов с нормальным распределением по каждому признаку.
    """
    mu0 = mu[0]
    mu1 = mu[1]
    sigma0 = sigma[0]
    sigma1 = sigma[1]
    col = len(mu0)

    class0 = np.random.normal(mu0[0], sigma0[0], (N, 1))
    class1 = np.random.normal(mu1[0], sigma1[0], (N, 1))
    for i in range(1, col):
        v0 = np.random.normal(mu0[i], sigma0[i], (N, 1))
        class0 = np.hstack((class0, v0))
        v1 = np.random.normal(mu1[i], sigma1[i], (N, 1))
        class1 = np.hstack((class1, v1))

    Y0 = np.zeros((N, 1), dtype=bool)
    Y1 = np.ones((N, 1), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.vstack((Y0, Y1)).ravel()

    rng = np.random.default_rng()
    arr = np.arange(2 * N)
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1


def nonlinear_dataset_5(N=1000):
    """
    Генерирует двумерные данные варианта 5"""
    rng = np.random.default_rng()
    # Шум при генерации
    noise = 0.04


    # ========== Класс 1 (внутренний овал) ==========
    # cx, cy — центр овала класса 1 на плоскости (X, Y). Сдвиг меняет положение овала.
    cx, cy = 0.25, -0.3

    # angle_deg — угол поворота овала в градусах
    angle_deg = 28
    angle_rad = np.deg2rad(angle_deg)
    cos_a, sin_a = np.cos(angle_rad), np.sin(angle_rad)

    def to_local(x, y, cx_=None, cy_=None):
        """Перевод (x,y) в локальные координаты эллипса: u вдоль большой оси, v вдоль
```

```
малой."""
        if cx_ is None:
            cx_, cy_ = cx, cy
        u = (x - cx_) * cos_a + (y - cy_) * sin_a
        v = -(x - cx_) * sin_a + (y - cy_) * cos_a
        return u, v

    def inside_ellipse(x, y, a, b, cx_=None, cy_=None):
        if cx_ is None:
            cx_, cy_ = cx, cy
        u, v = to_local(x, y, cx_, cy_)
        return (u**2 / a**2 + v**2 / b**2) < 1

    # a1 — большая полуось, b1 — малая полуось
    a1, b1 = 0.5, 0.14
    class1_list = []
    while len(class1_list) < N:
        x = rng.uniform(cx - a1 - 0.2, cx + a1 + 0.2)
        y = rng.uniform(cy - a1 - 0.2, cy + a1 + 0.2)
        if inside_ellipse(x, y, a1, b1):
            x += rng.normal(0, noise)
            y += rng.normal(0, noise)
            class1_list.append([x, y])
    class1 = np.array(class1_list)


def nonlinear_dataset_5(N=1000):

     rng = np.random.default_rng()
     # Шум при генерации: добавляется к каждой точке. Больше — размытее границы.
     noise = 0.04

    # ========== Класс 0 (U-образная оболочка) ==========
    # cx0, cy0 — центр оболочки
    cx0, cy0 = cx + 0.2, cy + 0.12
    # a2, b2 — полуоси внешнего эллипса оболочки
    a2, b2 = 1.2, 0.50
    # a_inner_gap, b_inner_gap — полуоси внутренней границы оболочки.
    a_inner_gap, b_inner_gap = 0.5, 0.21
    # u_cut — граница «выреза» U в локальных координатах оболочки (ось u вдоль
большой оси).
    #   Точки с u < u_cut отбрасываются → получается открытие U. Больше u_cut
(например 0) — вырез шире,
    #   оболочка короче слева. Меньше u_cut (например -0.5) — вырез уже, оболочка
длиннее слева.
    u_cut = -0.25

    class0_list = []
    while len(class0_list) < N:
        x = rng.uniform(cx0 - a2 - 0.15, cx0 + a2 + 0.15)
        y = rng.uniform(cy0 - b2 - 0.15, cy0 + b2 + 0.15)
        u, v = to_local(x, y, cx0, cy0)
        # Внутри внешнего эллипса, снаружи внутренней границы оболочки, и не в
«вырезе» (u >= u_cut)
        in_outer = inside_ellipse(x, y, a2, b2, cx0, cy0)
        in_inner = inside_ellipse(x, y, a_inner_gap, b_inner_gap, cx0, cy0)
        not_in_cut = u >= u_cut
        if in_outer and not in_inner and not_in_cut:
            x += rng.normal(0, noise)
            y += rng.normal(0, noise)
            class0_list.append([x, y])
    class0 = np.array(class0_list)
```

```
    Y0 = np.zeros((N,), dtype=bool)
    Y1 = np.ones((N,), dtype=bool)

    X = np.vstack((class0, class1))
    Y = np.concatenate((Y0, Y1))

    arr = np.arange(2 * N)
    rng.shuffle(arr)
    X = X[arr]
    Y = Y[arr]

    return X, Y, class0, class1
```

Файл Main.py

```
"""
Генерация модельных наборов данных
"""
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path

FIGURES_DIR = Path(__file__).resolve().parent / "figures"
FIGURES_DIR.mkdir(exist_ok=True)

try:
    import DataGenerator as dg
except ImportError:
    import lab1.DataGenerator as dg

N = 1000
mu0 = [0, 2, 3, 1]
mu1 = [3, 5, 1, 4]
sigma0 = [2, 1, 2, 1.5]
sigma1 = [1, 2, 1, 1]
mu = [mu0, mu1]
sigma = [sigma0, sigma1]

X, Y, class0, class1 = dg.norm_dataset(mu, sigma, N)
col = len(mu0)

# Разбиение на обучающую и тестовую подвыборки 70/30
train_count = round(0.7 * 2 * N)
Xtrain = X[0:train_count]
Xtest = X[train_count : 2 * N]
Ytrain = Y[0:train_count]
Ytest = Y[train_count : 2 * N]

# --- Визуализация нормального датасета ---
# Гистограммы по каждому признаку
for i in range(col):
    plt.figure()
    plt.hist(class0[:, i], bins="auto", alpha=0.7, label="Класс 0")
    plt.hist(class1[:, i], bins="auto", alpha=0.7, label="Класс 1")
    plt.xlabel("Значение признака")
    plt.ylabel("Частота")
    plt.title(f"Гистограмма признака {i + 1}")
    plt.legend()
    plt.savefig(FIGURES_DIR / f"hist_{i + 1}.png")
    plt.show()
```

```python
# Скатерограмма (признаки 0 и 2)
plt.figure()
plt.scatter(class0[:, 0], class0[:, 2], marker=".", alpha=0.7, label="Класс 0")
plt.scatter(class1[:, 0], class1[:, 2], marker=".", alpha=0.7, label="Класс 1")
plt.xlabel("Признак 1")
plt.ylabel("Признак 3")
plt.title("Скатерограмма: признаки 1 и 3 (нормальный датасет)")
plt.legend()
plt.savefig(FIGURES_DIR / "scatter_norm.png")
plt.show()

# --- Нелинейный датасет ---
Xn, Yn, class0_nl, class1_nl = dg.nonlinear_dataset_5(N)

plt.figure()
plt.scatter(class0_nl[:, 0], class0_nl[:, 1], marker=".", alpha=0.7, label="Класс 0")
plt.scatter(class1_nl[:, 0], class1_nl[:, 1], marker=".", alpha=0.7, label="Класс 1")
plt.xlabel("Признак X")
plt.ylabel("Признак Y")
plt.title("Нелинейный датасет (вариант 5): вложенные области")
plt.legend()
plt.savefig(FIGURES_DIR / "scatter_nonlinear_5.png")
plt.show()
```