Научно-технологический университет «Сириус»
Научный центр информационных технологий и искусственного интеллекта
Направление «Математическая робототехника»

Отчёт

по заданию №2

Дисциплина: численные методы нелинейной и выпуклой оптимизации

Тема: геометрия

Выполнил
Студент группы M01MP-24 _____А. С. Кондратьев

Преподаватель
Профессор, к.ф.-м.н. _____С. В. Гусев

Сириус
2025

Задача 1

Построим два набора случайных точек на плоскости, используя параметрическую формулу

$$x = x_c + ar\cos(\varphi)\cos(\alpha) - br\sin(\varphi)\sin(\alpha)$$
$$y = y_c + ar\cos(\varphi)\sin(\alpha) + br\sin(\varphi)\cos(\alpha),$$

где a, b, α, x_c, y_c – заданные параметры; φ, r – случайные величины, φ равномерно распределена на интервале $[0, 2\pi]$, r равномерно распределена на $[0, 1]$. В каждом наборе 500 точек. Для первого набора используем параметры

$$a = 3, b = 1, \alpha = 0, x_c = -2, y_c = 1,$$

для второго

$$a = 4, b = 1, \alpha = \frac{\pi}{4}, x_c = 2, y_c = -1$$

Рассчитаем прямую, разделяющую два множества на плоскости. Для этого сформулируем задачу оптимизации. 2 набора точек могут быть разделены гиперплоскостью

$$a_0^T x_i + b_0 > 0, \;\; a_0^T y_i + b_0 < 0$$

Данные выражения эквивалентны по a, b, поэтому

$$a_0^T x_{min} + b \geq \varepsilon > 0 \rightarrow \frac{a_0^T}{\varepsilon} x_{min} + \frac{b_0}{\varepsilon} \geq 1$$
$$a^T x_i + b \geq 1, \;\; a^T y_i + b \leq -1$$

Для построения разделяющей прямой минимизируем сумму отклонений точек множества от прямой (рисунок 1)

$$\min 1^T u + 1^T v \text{ при } a^T x_i + b \geq 1 - u_i, a^T y_i + b \leq -1 + v_i, v \geq 0, u \geq 0$$

Кроме того, учтем норму $\|a\|_2$ для максимизации зазора между множествами и разделяющей прямой, (рисунок 2). Тогда итоговая задача оптимизации выглядит следующим образом

$$\min \|a\|_2 + 1^T u + 1^T v$$

при

$$a^T x_i + b \geq +1 - u_i, \qquad u \geq 0$$
$$a^T y_i + b \leq -1 + v_i, \qquad v \geq 0$$

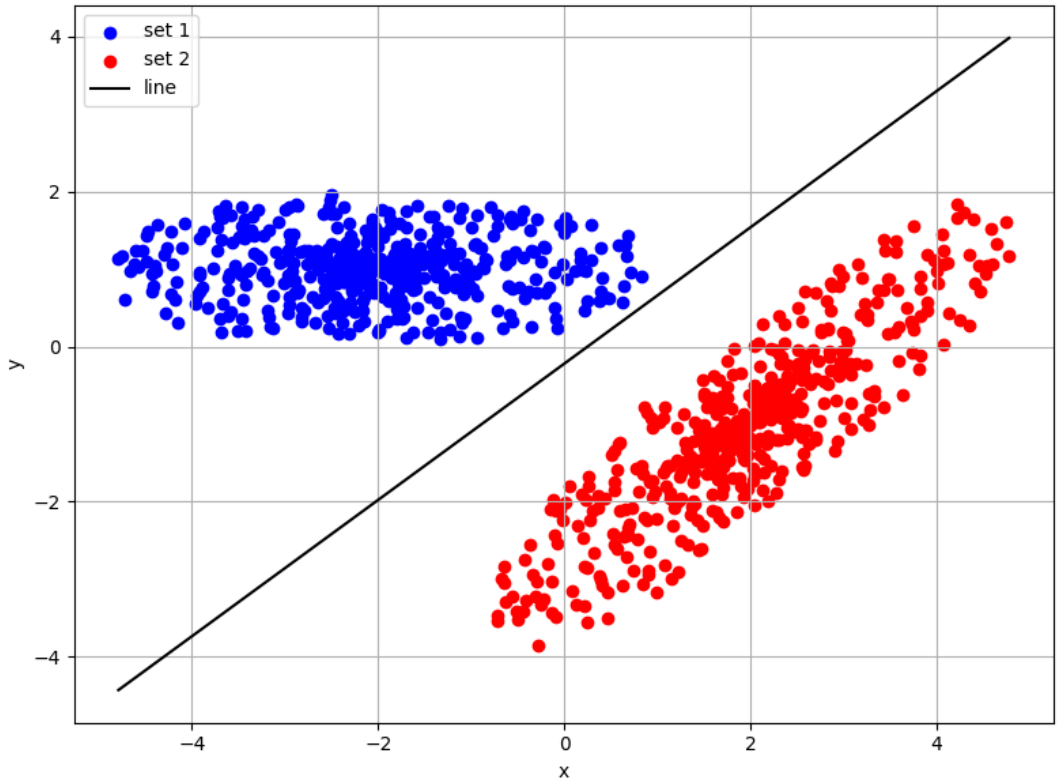

Рисунок 1 – Разделяющая прямая

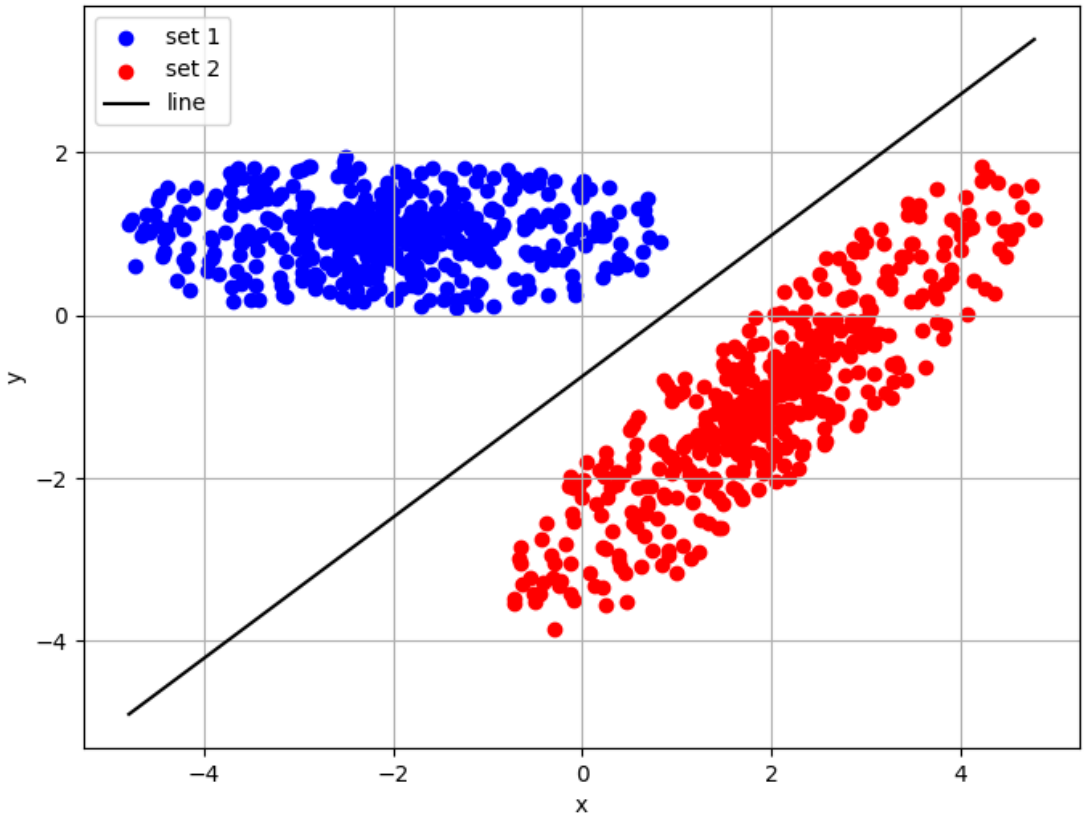

Рисунок 2 – Разделяющая прямая (с учетом $\|a\|_2$)

Задача 2

Изменим значения параметров генерации множеств. Параметры для первого множества

$$a = 3, b = 2.5, \alpha = 0, x_c = -2, y_c = 1$$

для второго множества

$$a = 4, b = 2, \alpha = 0, x_c = -2, y_c = 1$$

Результаты разделения множеств гиперплоскостью с учетом и без учета $\|a\|_2$ представлены на рисунках 3, 4 соответственно.

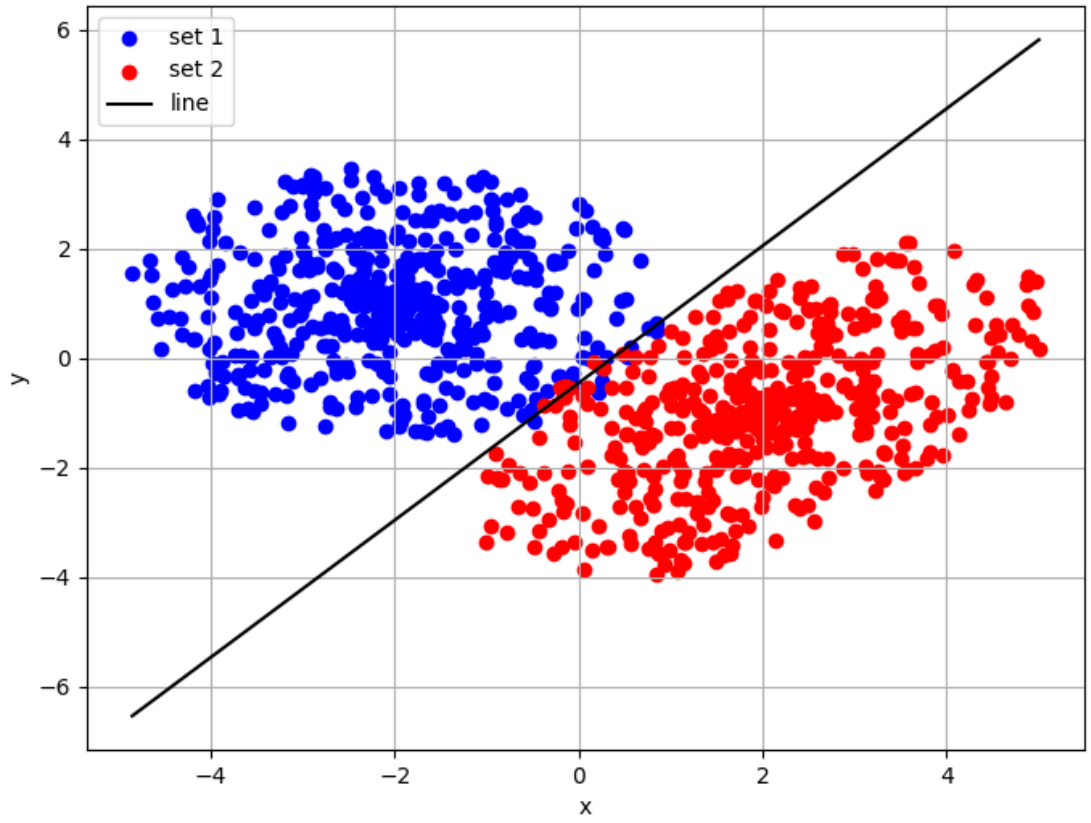

Рисунок 3 – Разделяющая прямая

Задача 3

Необходимо построить гиперплоскость, разделяющую два множества изображений с цифрами (0 и 1), с целью распознавания образов на тестовых изображениях. Пример цифр в тренировочном датасете представлен на рисунке 5.

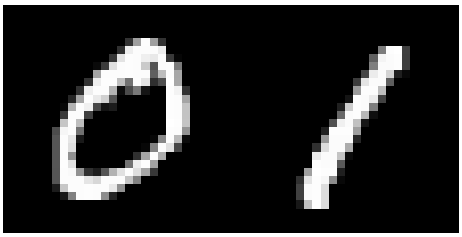

Рисунок 5 – Примеры изображений из тренировочного датасета

Тестовый датасет состоит из двух наборов изображений: в одном наборе содержится 980 изображений нулей, в другом – столько же изображений единиц. Для оценки точности распознавания образов использовалась доля неправильно распознанных образов. При использовании тренировочного датасета, состоящего из 100 изображений, доля неправильно распознанных образов составила 0.3% (3 изображения из 980). На рисунке 6 представлены изображения некорректно распознанных образов.

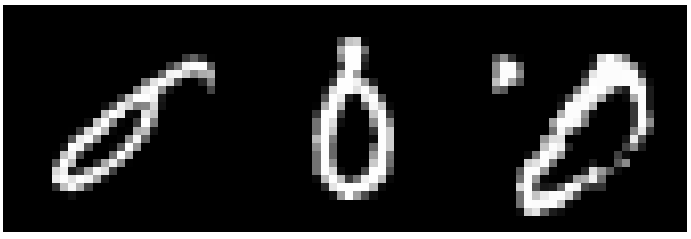

Рисунок 6 – Некорректно распознанные образы

Заключение

Разделение множества случайных точек с помощью гиперплоскости возможно регулировать за счет изменения целевой функции. При выборе большого коэффициента перед нормой $\|a\|_2$ зазор между гиперплоскостью и множествами, который обратно пропорционален этой норме, будет больше. В обратном случае будет минимизироваться сумма ошибок u и v, при этом зазор будет меньше.

Решение задачи распознавания образов с помощью разделения множеств гиперплоскостью на примере распознавания цифр на изображениях работает с погрешностью 0.3% при размере тестового датасета 980 изображений. Таким образом, решение задачи распознавания образов с помощью данного метода является успешным.

Приложение №1

dataloader.py

```
import numpy as np
import struct
from array import array
import os

class MnistDataloader(object):
    def __init__(self, training_images_filepath,training_labels_filepath,
                 test_images_filepath, test_labels_filepath):
        self.training_images_filepath = training_images_filepath
        self.training_labels_filepath = training_labels_filepath
        self.test_images_filepath = test_images_filepath
        self.test_labels_filepath = test_labels_filepath

    def read_images_labels(self, images_filepath, labels_filepath):
        labels = []
        with open(labels_filepath, 'rb') as file:
            magic, size = struct.unpack(">II", file.read(8))
            if magic != 2049:
                   raise ValueError('Magic number mismatch, expected 2049, got
{}'.format(magic))
            labels = array("B", file.read())

        with open(images_filepath, 'rb') as file:
            magic, size, rows, cols = struct.unpack(">IIII", file.read(16))
            if magic != 2051:
                   raise ValueError('Magic number mismatch, expected 2051, got
{}'.format(magic))
            image_data = array("B", file.read())
        images = []
        for i in range(size):
            images.append([0] * rows * cols)
        for i in range(size):
            img = np.array(image_data[i * rows * cols:(i + 1) * rows * cols])
            img = img.reshape(28, 28)
            images[i][:] = img

        return images, labels

    def load_data(self):
```

```
        x_train, y_train = self.read_images_labels(self.training_images_filepath,
self.training_labels_filepath)
            x_test,  y_test  =  self.read_images_labels(self.test_images_filepath,
self.test_labels_filepath)
               return  np.array(x_train),  np.array(y_train),  np.array(x_test),
np.array(y_test)

def load_dataset(path):
    training_images_filepath = os.path.join(path, 'train-images.idx3-ubyte')
    training_labels_filepath = os.path.join(path, 'train-labels.idx1-ubyte')
    test_images_filepath = os.path.join(path, 't10k-images.idx3-ubyte')
    test_labels_filepath = os.path.join(path, 't10k-labels.idx1-ubyte')

            mnist_dataloader     =     MnistDataloader(training_images_filepath,
training_labels_filepath, test_images_filepath, test_labels_filepath)
    return mnist_dataloader.load_data()
```

task2.py

```
import cv2 as cv
import cvxpy as cp
import matplotlib.pyplot as plt
import numpy as np
import os
import torch
import torch.nn as nn
import torch.optim as optim

from sklearn import svm

from dataloader import load_dataset


alpha = (0, np.pi / 4)
xc = (-2, 2)
yc = (1, -1)

size = 500

r = np.random.uniform(0, 1, size=size).reshape((size, 1))
phi = np.random.uniform(0, 2 * np.pi, size=size).reshape((size, 1))
```

```
use_task1 = False
use_task2 = False


def gen_x(k1, k2, alpha):
    def gen_x_(xc, yc, k1, k2, alpha):
        x = xc + k1 * r * np.cos(phi) * np.cos(alpha) - k2 * r * np.sin(phi) *
np.sin(alpha)
        y = yc + k1 * r * np.cos(phi) * np.sin(alpha) + k2 * r * np.sin(phi) *
np.cos(alpha)
        return np.hstack([x, y])

    x1 = gen_x_(xc[0], yc[0], k1[0], k2[0], alpha[0])
    x2 = gen_x_(xc[1], yc[1], k1[1], k2[1], alpha[1])
    return x1, x2, np.min([x1[:, 0], x2[:, 0]]), np.max([x1[:, 0], x2[:, 0]])


def draw_sets(x1, x2, line, sets_only=False):
    _, ax1 = plt.subplots(figsize=(8, 6))
    ax1.set_xlabel('x')
    ax1.set_ylabel('y')
    ax1.grid(True)

    ax1.scatter(x1[:, 0], x1[:, 1], color='blue', label='set 1')
    ax1.scatter(x2[:, 0], x2[:, 1], color='red', label='set 2')
    if not sets_only:
        ax1.plot(line[0], line[1], color='black', label='line')

    ax1.legend(loc='upper left')


def calc_line_coeffs(x1, x2, use_norm):
    print(x1.shape)
    u = cp.Variable((size, 1))
    v = cp.Variable((size, 1))
    a = cp.Variable((2, 1))
    b = cp.Variable((1, 1))
    constraints = [
        a.T @ x1.T + b >= 1 - u,
        a.T @ x2.T + b <= -1 + v,
        u >= 0,
        v >= 0,
```

```
    ]
    if use_norm:
        objective = cp.Minimize(0.0001 * cp.norm2(a) + 1 * (cp.sum(u) +
cp.sum(v)))
    else:
        objective = cp.Minimize(cp.sum(u) + cp.sum(v))
    problem = cp.Problem(objective, constraints)
    problem.solve(verbose=True, solver=cp.ECOS)
    return a.value, b.value


def get_line(x_min, x_max, a, b):
    def line(x):
        return 1 / a[1][0] * (-a[0][0] * x - b[0])
    x_line = [x_min, x_max]
    y_line = [line(x_min), line(x_max)]
    return (x_line, y_line)


def task1():
    global use_task1
    use_task1 = True

    k1 = (3, 4)
    k2 = (1, 1)

    x1, x2, x_min, x_max = gen_x(k1, k2, alpha)

    a, b = calc_line_coeffs(x1, x2, True)
    line1 = get_line(x_min, x_max, a, b)

    a, b = calc_line_coeffs(x1, x2, False)
    line2 = get_line(x_min, x_max, a, b)

    draw_sets(x1, x2, line1)
    draw_sets(x1, x2, line2)

    print(x1.shape)


def task2():
    global use_task2
    use_task2 = True
```

```
    k1 = (3, 4)
    k2 = (2.5, 2)

    x1, x2, x_min, x_max = gen_x(k1, k2, alpha)

    # scikit-learn

    clf = svm.SVC(kernel='linear')

    X = np.vstack([x1, x2])
    y = np.array([1] * x1.shape[0] + [2] * x2.shape[0])

    clf.fit(X, y)

    a = clf.coef_.T
    b = clf.intercept_

    print(a, b)

    line1 = get_line(x_min, x_max, a, b)

    draw_sets(x1, x2, line1)

    # torch

    X = torch.tensor(np.vstack([x1, x2]), dtype=torch.float32)
    y = torch.tensor(np.array([1] * x1.shape[0] + [-1] * x2.shape[0]),
dtype=torch.float32).view(-1, 1)

    class SVM_Network(nn.Module):
        def __init__(self):
            super(SVM_Network, self).__init__()
            self.fc1 = nn.Linear(2, 1)

        def forward(self, x):
            return self.fc1(x)

    model = SVM_Network()
    optimizer = optim.SGD(model.parameters(), lr=0.01)

    epochs = 5000
    for epoch in range(epochs):
```

```
        model.train()

        optimizer.zero_grad()

        outputs = model(X)

        u = torch.relu(1 - y * outputs)  # Для класса 1: условие 1 - y * output
>= 0
        v = torch.relu(1 + y * outputs)  # Для класса 2: условие 1 + y * output
<= 0

        loss = torch.norm(model.fc1.weight) + torch.sum(u) + torch.sum(v)

        loss.backward()
        optimizer.step()

        if epoch % 100 == 0:
            print(f'Epoch {epoch}, Loss: {loss.item()}')

    model.eval()

    a = model.fc1.weight.detach().numpy().T
    b = model.fc1.bias.detach().numpy()

    print(a, b)

    line = get_line(x_min, x_max, a, b)

    draw_sets(x1, x2, line)


def task3():
    path = os.path.join(os.path.pardir, 'dataset')
    images_train, labels_train, images_test, labels_test = load_dataset(path)

    train_list = []
    test_list = []

    train_size = 100
    test_size = 980

    w = images_train[0].shape[0]
```

```
    label1 = 0
    label2 = 1

    for label in (label1, label2):
        train = images_train[labels_train==label]
        train = np.reshape(train[:train_size], (train_size, w ** 2)).T
        train_list.append(train)

        test_images_i = images_test[labels_test==label]
        test_images_i  =  np.reshape(test_images_i[:test_size],  (test_size,  w  **
2)).T
        test_list.append(test_images_i)

    images_train = np.array(train_list, dtype=object)
    images_test = np.array(test_list, dtype=object)

    u = cp.Variable((images_train[0].shape[1], 1))
    v = cp.Variable((images_train[0].shape[1], 1))
    a = cp.Variable((images_train[0].shape[0], 1))
    b = cp.Variable((1, 1))

    constraints = [
        a.T @ images_train[0] + b >= 1 - u,
        a.T @ images_train[1] + b <= -1 + v,
        u >= 0,
        v >= 0,
    ]

    objective = cp.Minimize(cp.norm2(a) + cp.sum(u) + cp.sum(v))

    problem = cp.Problem(objective, constraints)
    # problem.solve(verbose=True)

    # a = a.value
    # b = b.value

    # a.tofile('a.data')
    # b.tofile('b.data')

    a = np.fromfile('a.data', dtype=np.float64)
    b = np.fromfile('b.data', dtype=np.float64)

    print(images_train.shape)
```

```
    cv.imwrite('0.png', images_train[0].T[0].astype(np.uint8).reshape((w, w)))
    cv.imwrite('1.png', images_train[1].T[0].astype(np.uint8).reshape((w, w)))
    cv.waitKey(0)

    fails_zero = 0
    fails_ones = 0
    fails_zero_list = []
    fails_ones_list = []
    for i in range(test_size):
        if a.T @ images_test[0].T[i] + b < 0:
            fails_zero += 1

fails_zero_list.append(images_test[0].T[i].astype(np.uint8).reshape((w, w)))
        if a.T @ images_test[1].T[i] + b > 0:
            fails_ones += 1

fails_ones_list.append(images_test[1].T[i].astype(np.uint8).reshape((w, w)))

    print(fails_zero, fails_ones)

    print('zeros fails:', fails_zero / test_size)

    i = 0
    for img in fails_zero_list:
        cv.imshow('zeros fails', img)
        cv.imwrite(f'fail{i}.png', img)
        i+= 1
        cv.waitKey(0)

    print('ones fails:', fails_ones / test_size)

    for img in fails_ones_list:
        cv.imshow('ones fails', img)
        cv.waitKey(0)

    # zero = images_test[0][15]
    # one = images_test[1][15]

    # print(a.T @ zero + b > 0, a.T @ one + b < 0)


if __name__ == '__main__':
```

```
# task1()
# task2()
task3()

if use_task1 or use_task2:
    plt.tight_layout()
    plt.show()
```