**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине: **«Введение в нейронные сети»**

на тему: **Бинарная классификация с использование машины опорных векторов**

Выполнил: студент группы ИТП-31

Пронуза М.Ю.

Проверил:

Башаримов Ю.С.

Гомель 2024

**Цель работы**: разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием метода опорных векторов.

**Задание:**

В качестве параметров программа должна принимать: файл, содержащий: ссылки на файлы с обучающей выборкой;

ссылки на файлы с тестируемыми образцами;

ссылку на файл с результатами проверки образцов (принадлежит / не

принадлежит).

Если происходит запуск без параметров, то запускается программа с графическим интерфейсом, ссылки на обучающую выборку берутся из конфигурационного файла аналогичной структуры, как и файл параметров командной строки.

Решить задачу из лабораторной работы №2. В качестве бинарного классификатора использовать метод опорных векторов. Определить вид ядра SVM, обеспечивающий наилучшее разделение классов. Сравнить полученные результаты с результатами лабораторной работы №2. Построить графики. Сделать выводы.

**Ход выполнения и результаты работы:**

На рисунке 1 представлен результат эпох и точность методом для разных ядер *SVM*.

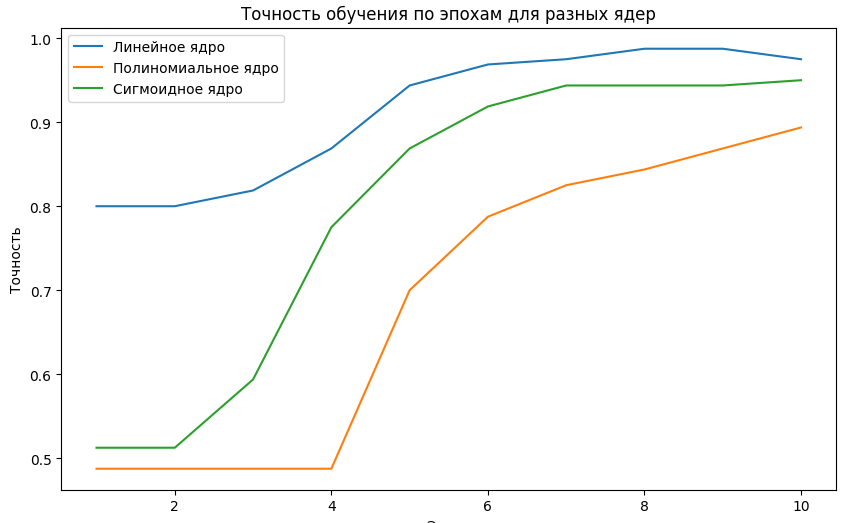


Рисунок 1 – точность методом для разных ядер *SVM*

На рисунке 2 представлен результат обучения и примеры предсказаний для разных ядер.

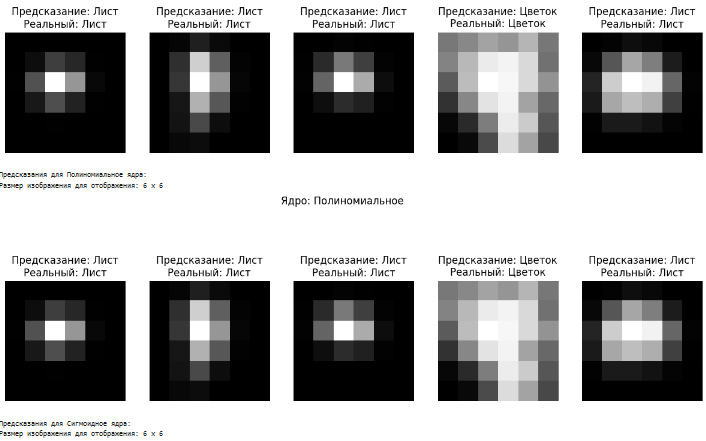


Рисунок 2 – результат обучения и примеры предсказаний для разных ядер

Текст обработки входных данных представлен в приложении А.

Текст программы методом *SVM* представлен в приложении Б.

Текст графиков сравнения представлен в приложении В.

**Вывод:** в ходе выполнения лабораторной работы были разработано программное обеспечение для решения задачи бинарной классификации с учителем с использованием метода опорных векторов.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст обработки входных данных**

import requests

import zipfile

import io

import os

from skimage import color, transform

import numpy as np

import tensorflow\_datasets as tfds

import matplotlib.pyplot as plt

# Download and extract the leaf dataset

url = 'https://archive.ics.uci.edu/static/public/288/leaf.zip'

response = requests.get(url)

if response.status\_code == 200:

with zipfile.ZipFile(io.BytesIO(response.content)) as z:

z.extractall('leaf\_dataset')

leaf\_image\_dir = 'leaf\_dataset/BW'

processed\_leaf\_images = []

target\_size = (6, 6)

max\_images = 100

image\_count = 0

# Process leaf images

for root, dirs, files in os.walk(leaf\_image\_dir):

for filename in files:

if filename.endswith('.TIFF'):

image\_path = os.path.join(root, filename)

image = plt.imread(image\_path)

if image.shape[2] == 4:

image = image[:, :, :3] # Remove alpha channel if present

if len(image.shape) == 3:

gray\_image = color.rgb2gray(image) # Convert to grayscale

else:

gray\_image = image

resized\_image = transform.resize(gray\_image, target\_size, anti\_aliasing=True)

processed\_leaf\_images.append(resized\_image)

image\_count += 1

if image\_count >= max\_images:

break

if image\_count >= max\_images:

break

processed\_leaf\_images = np.array(processed\_leaf\_images)

# Load the flower dataset from TensorFlow Datasets (TFDS)

flowers\_dataset, flowers\_info = tfds.load('oxford\_flowers102', split=f'train[:{max\_images}]', with\_info=True)

processed\_flower\_images = []

# Process flower images

for example in tfds.as\_numpy(flowers\_dataset):

image = example['image']

gray\_image = color.rgb2gray(image)

resized\_image = transform.resize(gray\_image, target\_size, anti\_aliasing=True)

processed\_flower\_images.append(resized\_image)

processed\_flower\_images = np.array(processed\_flower\_images)

# Show sample images from both datasets

def show\_images(images, n=10):

plt.figure(figsize=(10, 2))

for i in range(min(n, len(images))):

ax = plt.subplot(1, n, i + 1)

plt.imshow(images[i], cmap="gray")

plt.axis("off")

plt.show()

show\_images(processed\_leaf\_images, n=10)

show\_images(processed\_flower\_images, n=10)

print("Количество обработанных изображений листьев:", processed\_leaf\_images.shape)

print("Количество обработанных изображений цветов:", processed\_flower\_images.shape)

**ПРИЛОЖЕНИЕ Б**

(обязательное)

**Текст метода *SVM***

import numpy as np

import matplotlib.pyplot as plt

# Предполагаем, что processed\_leaf\_images и processed\_flower\_images уже загружены и содержат изображения

# Проверка размеров обработанных изображений

print("Размеры обработанных изображений листьев:", processed\_leaf\_images.shape)

print("Размеры обработанных изображений цветов:", processed\_flower\_images.shape)

# Присваиваем метки: 0 для листьев, 1 для цветов

leaf\_labels = np.zeros(len(processed\_leaf\_images))

flower\_labels = np.ones(len(processed\_flower\_images))

# Объединяем изображения и метки

X = np.concatenate((processed\_leaf\_images, processed\_flower\_images), axis=0)

y = np.concatenate((leaf\_labels, flower\_labels), axis=0)

# Разворачиваем изображения в векторы

X\_flat = X.reshape(len(X), -1)

# Нормализуем данные

X\_flat = X\_flat / np.max(X\_flat)

# Проверка формы данных

print("Форма X\_flat:", X\_flat.shape)

# Перемешиваем и разделяем данные на тренировочную и тестовую выборки

indices = np.arange(len(X\_flat))

np.random.shuffle(indices)

X\_flat = X\_flat[indices]

y = y[indices]

# Разделение на тренировочную и тестовую выборки (80% обучение, 20% тест)

split\_index = int(len(X\_flat) \* 0.8)

X\_train, X\_test = X\_flat[:split\_index], X\_flat[split\_index:]

y\_train, y\_test = y[:split\_index], y[split\_index:]

# Определяем функции ядер

def linear\_kernel(x1, x2):

return np.dot(x1, x2)

def polynomial\_kernel(x1, x2, degree=3, coef=1):

return (np.dot(x1, x2) + coef) \*\* degree

def sigmoid\_kernel(x1, x2, gamma=1, coef=-1):

return np.tanh(gamma \* np.dot(x1, x2) + coef)

# Функция обучения SVM

def svm\_train(X\_train, y\_train, kernel\_func, learning\_rate=0.001, epochs=10, lambda\_param=0.01):

num\_samples, num\_features = X\_train.shape

# Инициализация весов и смещения

weights = np.zeros(num\_features)

bias = 0

# Преобразуем метки: заменяем 0 на -1

y\_transformed = np.where(y\_train == 0, -1, 1)

accuracies = []

for epoch in range(epochs):

for idx in range(num\_samples):

x\_i = X\_train[idx]

y\_i = y\_transformed[idx]

# Условие для обновления весов

condition = y\_i \* (kernel\_func(x\_i, weights) - bias) >= 1

if condition:

# Только регуляризация

weights -= learning\_rate \* (2 \* lambda\_param \* weights)

else:

# Регуляризация и обновление весов и смещения

weights -= learning\_rate \* (2 \* lambda\_param \* weights - np.dot(x\_i, y\_i))

bias -= learning\_rate \* y\_i

# Расчет точности на обучающей выборке

y\_pred = svm\_predict(X\_train, weights, bias, kernel\_func)

accuracy = np.mean(y\_train == y\_pred)

accuracies.append(accuracy)

print(f"Эпоха {epoch+1}/{epochs}, Точность на обучении: {accuracy:.4f}")

return weights, bias, accuracies

# Функция предсказания SVM

def svm\_predict(X, weights, bias, kernel\_func):

outputs = np.array([kernel\_func(x\_i, weights) for x\_i in X]) - bias

return np.where(outputs >= 0, 1, 0)

# Обучение SVM с различными ядрами и сохранение результатов

kernels = {

'Линейное': linear\_kernel,

'Полиномиальное': polynomial\_kernel,

'Сигмоидное': sigmoid\_kernel

}

kernel\_accuracies = {}

trained\_weights = {}

trained\_biases = {}

for kernel\_name, kernel\_func in kernels.items():

print(f"\nОбучение SVM с {kernel\_name} ядром...")

weights, bias, accuracies = svm\_train(X\_train, y\_train, kernel\_func, learning\_rate=0.001, epochs=10)

kernel\_accuracies[kernel\_name] = accuracies

trained\_weights[kernel\_name] = weights

trained\_biases[kernel\_name] = bias

# Оценка на тестовых данных

y\_pred\_test = svm\_predict(X\_test, weights, bias, kernel\_func)

test\_accuracy = np.mean(y\_test == y\_pred\_test)

print(f"Точность на тестовых данных с {kernel\_name} ядром: {test\_accuracy:.4f}")

**ПИЛОЖЕНИЕ В**

(обязательное)

**Текст графиков сравнения методов**

# Построение графика точности обучения по эпохам для каждого ядра

plt.figure(figsize=(10, 6))

for kernel\_name, accuracies in kernel\_accuracies.items():

plt.plot(range(1, len(accuracies) + 1), accuracies, label=f'{kernel\_name} ядро')

plt.title('Точность обучения по эпохам для разных ядер')

plt.xlabel('Эпоха')

plt.ylabel('Точность')

plt.legend()

plt.show()

# Отображение изображений с предсказаниями и реальными метками для каждого ядра

num\_images\_to\_show = 5 # Количество изображений для отображения

for kernel\_name, kernel\_func in kernels.items():

print(f"\nПредсказания для {kernel\_name} ядра:")

# Получаем соответствующие веса и смещение для текущего ядра

weights = trained\_weights[kernel\_name]

bias = trained\_biases[kernel\_name]

# Делаем предсказания на тестовых данных

y\_pred\_test = svm\_predict(X\_test, weights, bias, kernel\_func)

# Выбираем несколько изображений для отображения

X\_test\_samples = X\_test[:num\_images\_to\_show]

y\_test\_samples = y\_test[:num\_images\_to\_show]

y\_pred\_samples = y\_pred\_test[:num\_images\_to\_show]

# Определяем размер изображения

image\_size = int(np.sqrt(X\_test\_samples.shape[1]))

print(f"Размер изображения для отображения: {image\_size} x {image\_size}")

# Отображаем изображения с предсказаниями и реальными метками

plt.figure(figsize=(15, 5))

for i in range(num\_images\_to\_show):

plt.subplot(1, num\_images\_to\_show, i + 1)

image = X\_test\_samples[i].reshape((image\_size, image\_size))

plt.imshow(image, cmap='gray')

pred\_label = 'Лист' if y\_pred\_samples[i] == 0 else 'Цветок'

true\_label = 'Лист' if y\_test\_samples[i] == 0 else 'Цветок'

plt.title(f"Предсказание: {pred\_label}\nРеальный: {true\_label}")

plt.axis('off')

plt.suptitle(f"Ядро: {kernel\_name}")

plt.show()