**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 использованием

машины опорных векторов**»**

Выполнил: студент гр. ИТП-31

Расшивалов Н.И.

Принял: преподаватель-стажёр

Дашкевич Д.А.

Гомель 2021

**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.

Распознаваемые классы: стоящий и идущий человек.

**Ход работы**

На рисунке 1 представлен один из человечков выборки.

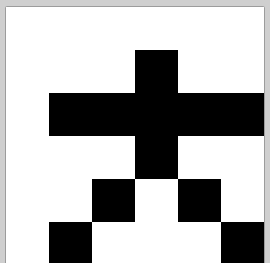


Рисунок 1 – Пример человека выборки

На рисунке 2 представлен пример конфигурационного файла.

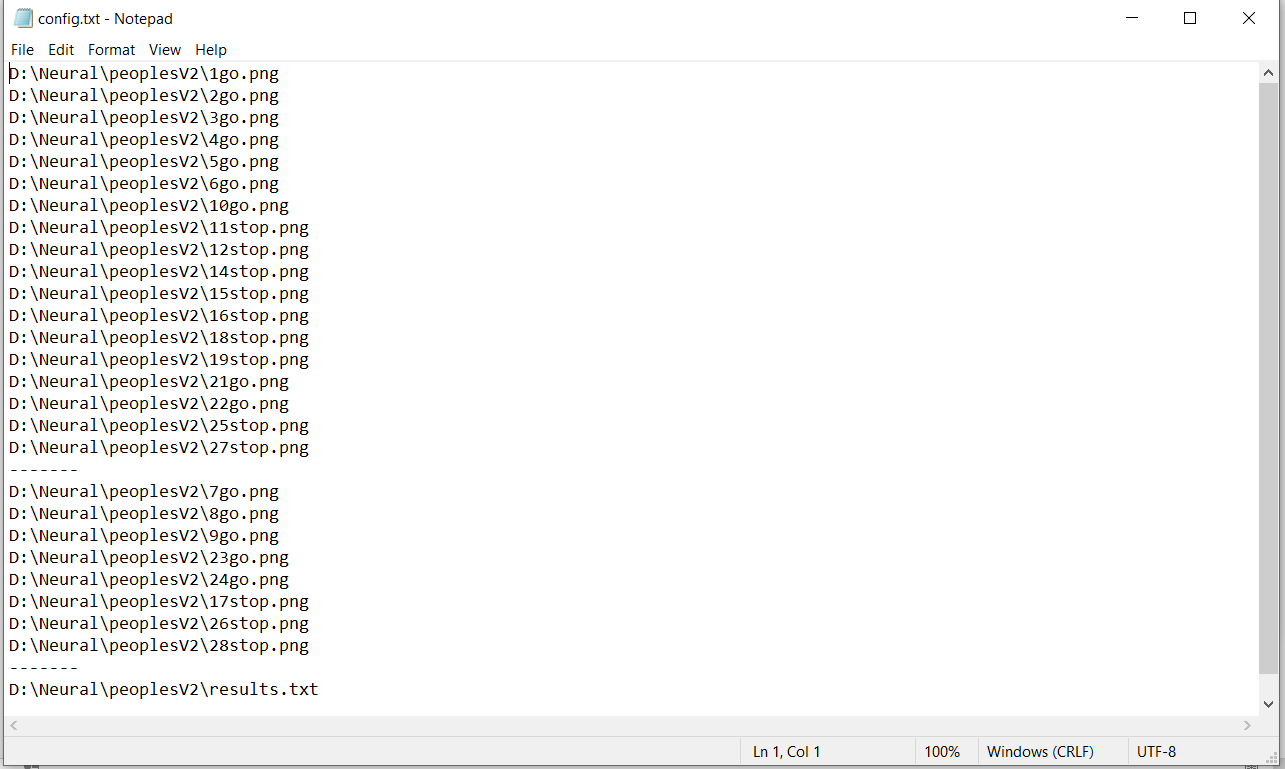


Рисунок 2 – Пример конфигурационного файла

На рисунке 3 представлена визуализация разделения классов полиномиальным ядром.

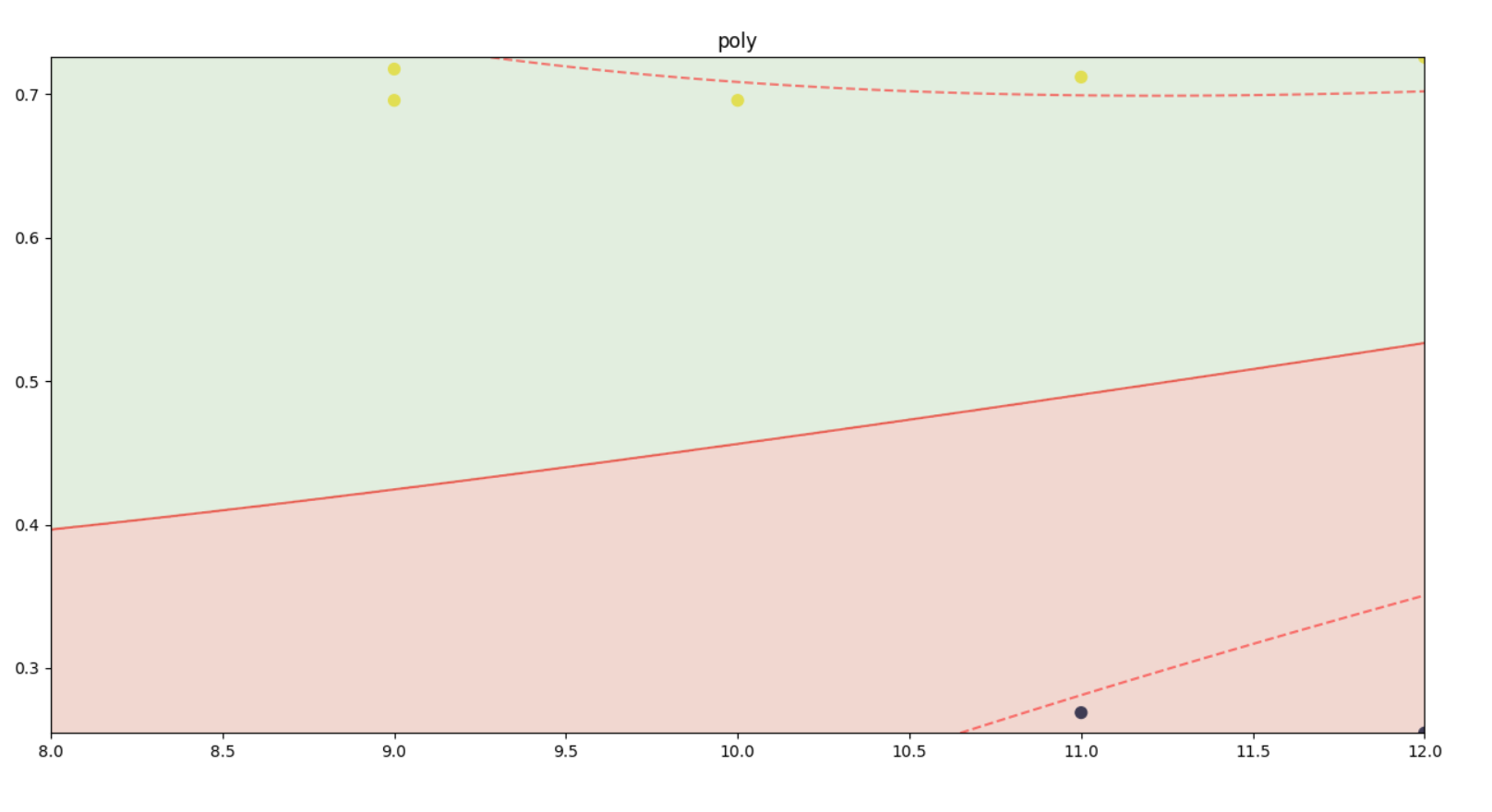


Рисунок 3 – Визуализация разделения классов полиномиальным ядром

На рисунке 4 представлена визуализация разделения классов линейным ядром.

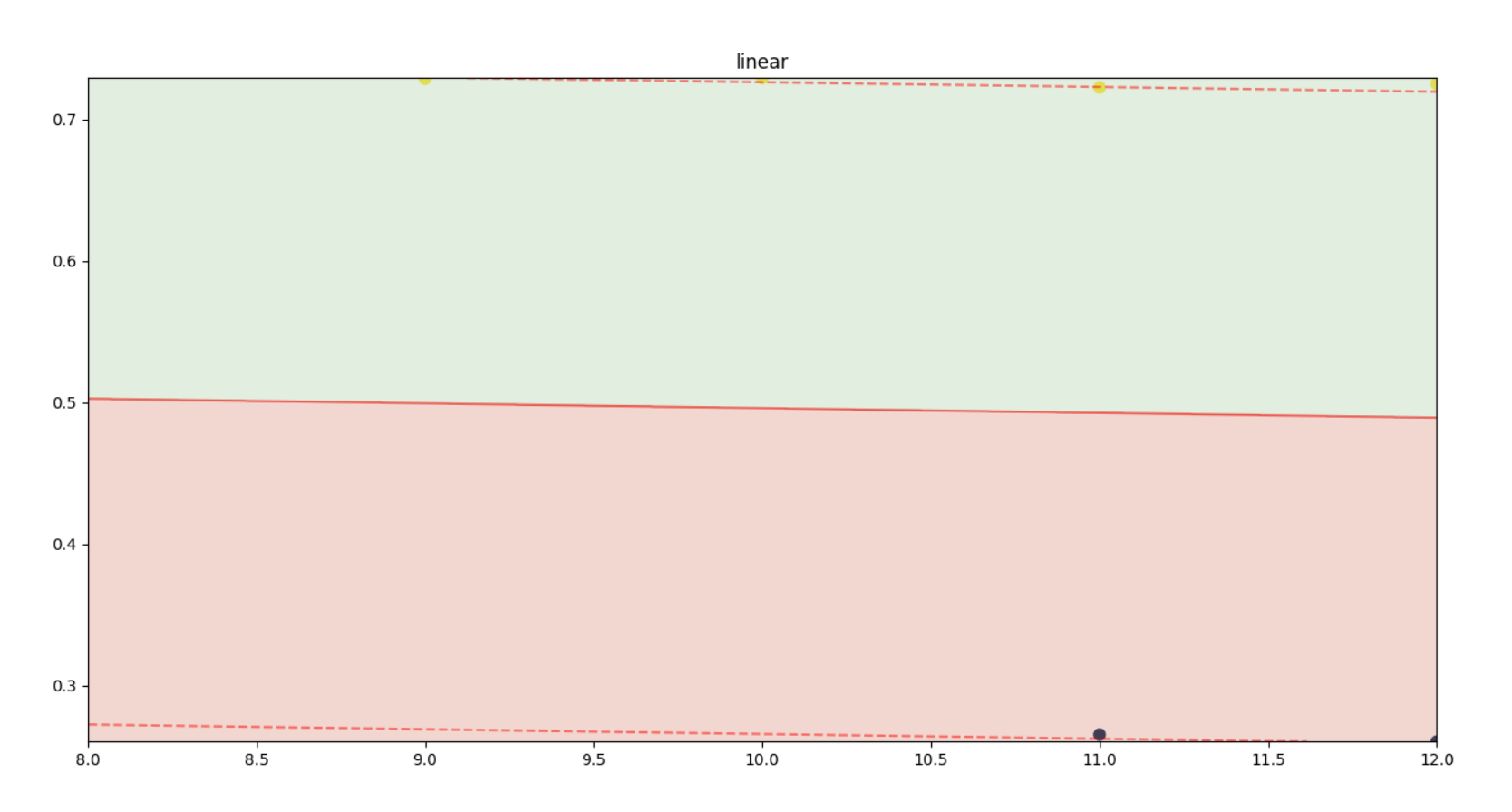


Рисунок 4 – Визуализация разделения классов линейным ядром

На рисунке 5 представлена визуализация разделения классов ядром радиальной базисной функции.

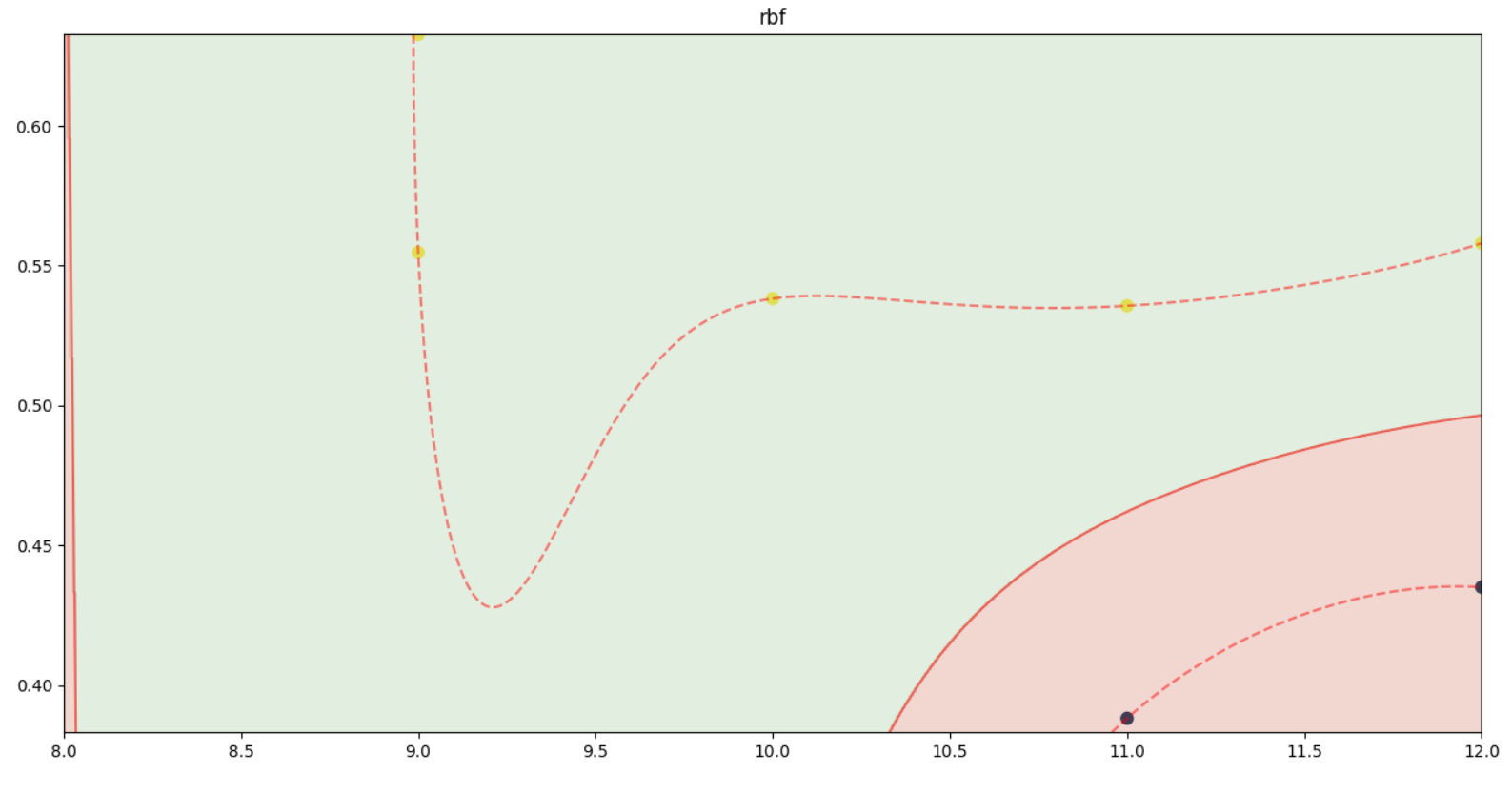


Рисунок 5 – Визуализация разделения классов ядром радиальной базисной функции

На рисунке 6 представлено сравнение правильности предсказываемых классов.

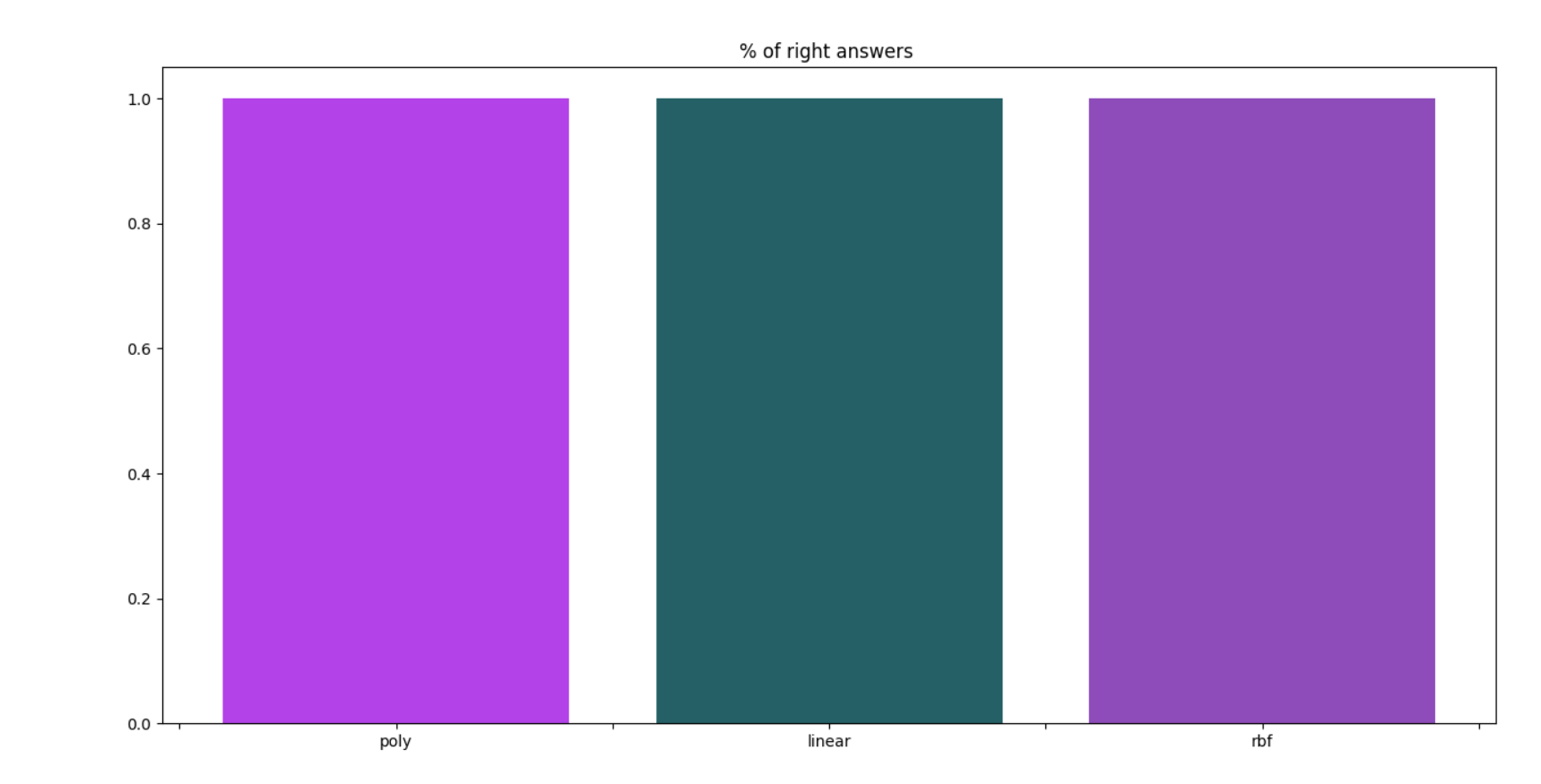


Рисунок 6 – Сравнение правильности предсказываемых классов

На рисунке 7 представлен файл с результатами.

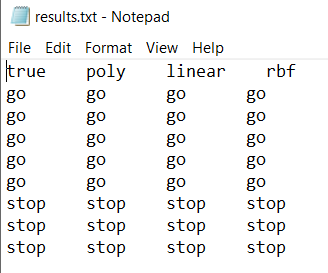


Рисунок 6 – Файл с результатами

Листинг программы представлен в приложении А.

**Вывод:** в ходе выполнения лабораторной работы обучена нейронная сеть с помощью трех видов метода опорных векторов выбрана. Полиноминальное и линейное ядра показали качественное разделение классов, в то время как РБФ ядро в некоторых случаях допускало ошибки в предсказании классов, по графикам можно также отменить, что опорные ядра классов у РБФ метода имеют наименьшее расстояние между друг другом.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Листинг программы**

import numpy as np

import cv2

import os

from tkinter import \*

from tkinter import filedialog

import matplotlib.pyplot as plt

import matplotlib.lines as mlines

from matplotlib.colors import ListedColormap

def main():

    openFile(True)

def openFile(isDefault):

    if isDefault==True:

        path=os.path.dirname(os.path.abspath(\_\_file\_\_))+'\\config.txt'

    else:

        path = filedialog.askopenfilename(filetypes=(("Текстовый файл", "\*.txt"),("Текстовый файл", "\*.txt")))

    if path!='':

        images = []

        lines = open(path).readlines()

        files = []

        j = 0

        for i in range(3):

            files.append([])

            while  j < len(lines) and lines[j].\_\_contains\_\_("-----") == False:

                files[i].append(lines[j].replace("\n",""))

                j += 1

            j += 1

        train\_y\_trues = []

        for i in range(len(files[0])):

            if files[0][i].\_\_contains\_\_(".png"):

                images.append(binarize(cv2.imread(files[0][i],cv2.IMREAD\_GRAYSCALE)))

                if files[0][i].\_\_contains\_\_("go"):

                    train\_y\_trues.append(1)

                else:

                    train\_y\_trues.append(0)

        train\_y\_trues = np.array(train\_y\_trues)

        print(train\_y\_trues)

        trainData = np.array(images)

        test\_y\_trues = []

        images = []

        for i in range(len(files[1])):

            if files[1][i].\_\_contains\_\_(".png"):

                images.append(binarize(cv2.imread(files[1][i],cv2.IMREAD\_GRAYSCALE)))

                if files[1][i].\_\_contains\_\_("go"):

                    test\_y\_trues.append(1)

                else:

                    test\_y\_trues.append(0)

        test\_y\_trues = np.array(test\_y\_trues)

        testData = np.array(images)

        linearKernelNetwork = SvmNeuralNetwork(kernel='linear', C=1000, max\_iter=60, degree=3)

        linearKernelNetwork.train(trainData, train\_y\_trues)

        polyKernelNetwork = SvmNeuralNetwork(kernel='poly', C=1000, max\_iter=60, degree=3)

        polyKernelNetwork.train(trainData, train\_y\_trues)

        rbfKernelNetwork = SvmNeuralNetwork(kernel='rbf', C=1000, max\_iter=60, degree=3)

        rbfKernelNetwork.train(trainData, train\_y\_trues)

        y\_pred = polyKernelNetwork.predict(testData)

        x\_start = 8

        x\_end = 12

        sizeX = 5

        sizeY = 5

        print("poly results:")

        print(test\_y\_trues)

        print(y\_pred)

        poly\_pred = y\_pred

        losses = []

        loss = 0

        s = 0

        for i in range(len(y\_pred)):

            loss += abs(y\_pred[i] - test\_y\_trues[i])

            if y\_pred[i] > 0.5 and test\_y\_trues[i] == 1:

                s += 1

            if y\_pred[i] <= 0.5 and test\_y\_trues[i] == 0:

                s += 1

        losses.append(s/len(y\_pred))

        print("poly loss:" + str(loss))

        X = []

        for i in range(len(y\_pred)):

            X.append([sum(testData[i]), y\_pred[i]])

        X = np.array(X)

        fig, axs = plt.subplots(nrows=1)

        fig.set\_size\_inches(sizeX, sizeY)

        plt.xlim([x\_start, x\_end])

        test\_plot(X, test\_y\_trues, SvmNeuralNetwork(kernel='poly', C=1000, max\_iter=60, degree=3), axs, 'poly')

        y\_pred = linearKernelNetwork.predict(testData)

        linear\_pred = y\_pred

        print("Linear results:")

        print(test\_y\_trues)

        print(y\_pred)

        loss = 0

        s = 0

        for i in range(len(y\_pred)):

            loss += abs(y\_pred[i] - test\_y\_trues[i])

            if y\_pred[i] > 0.5 and test\_y\_trues[i] == 1:

                s += 1

            if y\_pred[i] <= 0.5 and test\_y\_trues[i] == 0:

                s += 1

        losses.append(s/len(y\_pred))

        print("linear loss:" + str(loss))

        X = []

        for i in range(len(y\_pred)):

            X.append([sum(testData[i]), y\_pred[i]])

        X = np.array(X)

        fig, axs = plt.subplots(nrows=1)

        fig.set\_size\_inches(sizeX, sizeY)

        plt.xlim([x\_start, x\_end])

        test\_plot(X, test\_y\_trues, SvmNeuralNetwork(kernel='linear', C=1000, max\_iter=60, degree=3), axs, 'linear')

        y\_pred = rbfKernelNetwork.predict(testData)

        print("Rbf results:")

        print(test\_y\_trues)

        print(y\_pred)

        rbf\_pred = y\_pred

        loss = 0

        s = 0

        for i in range(len(y\_pred)):

            loss += abs(y\_pred[i] - test\_y\_trues[i])

            if y\_pred[i] > 0.5 and test\_y\_trues[i] == 1:

                s += 1

            if y\_pred[i] <= 0.5 and test\_y\_trues[i] == 0:

                s += 1

        losses.append(s/len(y\_pred))

        print("rbf loss:" + str(loss))

        X = []

        for i in range(len(y\_pred)):

            X.append([sum(testData[i]), y\_pred[i]])

        X = np.array(X)

        fig, axs = plt.subplots(nrows=1)

        fig.set\_size\_inches(sizeX, sizeY)

        plt.xlim([x\_start, x\_end])

        test\_plot(X, test\_y\_trues, SvmNeuralNetwork(kernel='rbf', C=1000, max\_iter=60, degree=3), axs, 'rbf')

        print(files[2][0])

        f = open(files[2][0], 'w')

        resultStr = "true    poly    linear    rbf\n"

        for i in range(len(test\_y\_trues)):

            if test\_y\_trues[i] > 0.5:

                resultStr += '%-8s' % "go"

            else:

                resultStr += '%-8s' % "stop "

            if poly\_pred[i] > 0.5:

                resultStr += '%-8s' % "go"

            else:

                resultStr += '%-8s' % "stop"

            if linear\_pred[i] > 0.5:

                resultStr += '%-8s' % "go"

            else:

                resultStr += '%-8s' % "stop"

            if rbf\_pred[i] > 0.5:

                resultStr += '%-8s' % "go"

            else:

                resultStr += '%-8s' % "stop"

            resultStr += "\n"

        f.write(resultStr)

        f.close()

        fig, axs = plt.subplots(nrows=1)

        fig.set\_size\_inches(sizeX, sizeY)

        axs.set\_title('% of right answers ')

        axs.bar([1,2,3], losses,color = "b")

        axs.set\_xticklabels(['','','poly','','linear','','rbf'])

        plt.show()

def test\_plot(X, y, svm\_model, axes, title):

  plt.axes(axes)

  xlim = [np.min(X[:, 0]), np.max(X[:, 0])]

  ylim = [np.min(X[:, 1]), np.max(X[:, 1])]

  xx, yy = np.meshgrid(np.linspace(\*xlim, num=700), np.linspace(\*ylim, num=700))

  rgb=np.array([[210, 123, 100], [159, 200, 150]])/255.0

  svm\_model.train(X, y)

  z\_model = svm\_model.decision\_function(np.c\_[xx.ravel(), yy.ravel()]).reshape(xx.shape)

  plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='cividis')

  plt.contour(xx, yy, z\_model, colors='r', levels=[-1, 0, 1], alpha=0.5, linestyles=['--', '-', '--'])

  plt.contourf(xx, yy, np.sign(z\_model.reshape(xx.shape)), alpha=0.3, levels=2, cmap=ListedColormap(rgb), zorder=1)

  plt.title(title)

def nonlin(x,deriv=False):

    if(deriv==True):

        return np.exp(-x)/((1 + np.exp(-x))\*(1 + np.exp(-x)))

    return 1/(1+np.exp(-x))

class SvmNeuralNetwork:

  def \_\_init\_\_(self, kernel='linear', C=10, max\_iter=100000, degree=3, gamma=1):

    self.kernel = {'poly'  : lambda x,y: np.dot(x, y.T)\*\*degree,

         'rbf': lambda x,y: np.exp(-gamma\*np.sum((y-x[:,np.newaxis])\*\*2,axis=-1)),

         'linear': lambda x,y: np.dot(x, y.T)}[kernel]

    self.C = C

    self.max\_iter = max\_iter

  def restrict\_to\_square(self, t, v0, u):

    t = (np.clip(v0 + t\*u, 0, self.C) - v0)[1]/u[1]

    return (np.clip(v0 + t\*u, 0, self.C) - v0)[0]/u[0]

  def train(self, X, y):

    self.X = X.copy()

    self.y = y \* 2 - 1

    self.lambdas = np.zeros\_like(self.y, dtype=float)

    self.K = self.kernel(self.X, self.X) \* self.y[:,np.newaxis] \* self.y

    for \_ in range(self.max\_iter):

      for idxM in range(len(self.lambdas)):

        idxL = 1

        Q = self.K[[[idxM, idxM], [idxL, idxL]], [[idxM, idxL], [idxM, idxL]]]

        v0 = self.lambdas[[idxM, idxL]]

        k0 = 1 - np.sum(self.lambdas \* self.K[[idxM, idxL]], axis=1)

        u = np.array([-self.y[idxL], self.y[idxM]])

        t\_max = np.dot(k0, u) / (np.dot(np.dot(Q, u), u) + 1E-15)

        self.lambdas[[idxM, idxL]] = v0 + u \* self.restrict\_to\_square(t\_max, v0, u)

    # найти индексы опорных векторов

    idx, = np.nonzero(self.lambdas > 1E-15)

    # формула (1)

    self.b = np.mean((1.0-np.sum(self.K[idx]\*self.lambdas, axis=1))\*self.y[idx])

  def decision\_function(self, X):

    return np.sum(self.kernel(X, self.X) \* self.y \* self.lambdas, axis=1) + self.b

  def predict(self, X):

    # преобразование классов -1,+1 в 0,1; для лучшей совместимости с sklearn

   # return (np.sign(self.decision\_function(X)) + 1) // 2

   return nonlin(self.decision\_function(X))

def binarize(img):

    vectorImg = []

    for i in range(len(img)):

        for j in range(len(img[0])):

            if img[i][j] > 0:

                vectorImg.append(0)

            else:

                vectorImg.append(1)

    return vectorImg

main()