**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

Гошко В.В.

Принял: преподаватель

Гуменников Е.Д.

Гомель 2021

**Цель:** разработать программное обеспечение для решения задачи бинарной классификации с учителем с использованием машины 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.

**Вариант 6:** Легковой и грузовой автомобиль.

**Ход работы:**

Создан класс, реализующий метод опорных векторов *SVM*.

Созданы тренировочные картинки, изображающие автомобили в количестве минимум 15 штук на каждый тип. Созданы тестовые картинки, для верификации сети.

Сеть получает исходные данные из конфигурационного файла, показанного на рисунке 1.

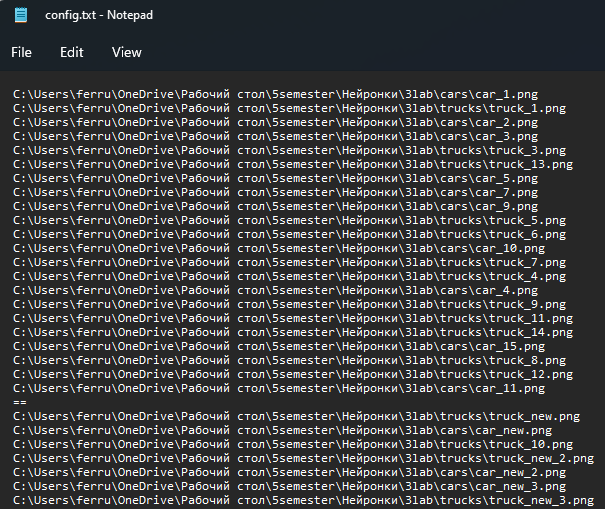


Рисунок 1 – Конфигурационный файл со ссылками на картинки

Процесс обучения и проверки тестированием нейронной сети по методу опорных векторов на рисунке 2.

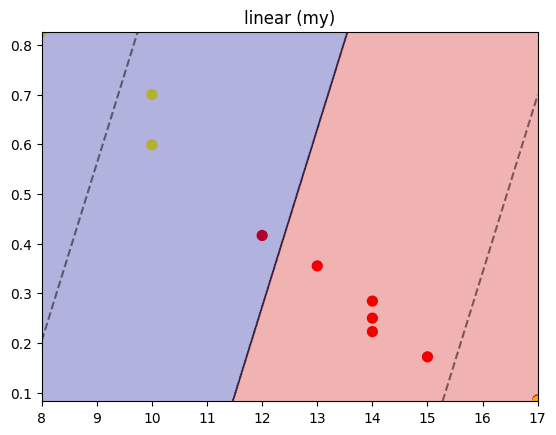


Рисунок 2 – График результатов нейронной сети по методу опорных векторов с линейным ядром

На рисунке 3 представлено тестирование сети, обученной методом обратного распространения ошибки.

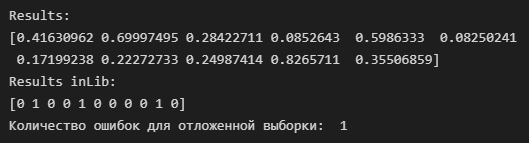


Рисунок 3 – Тестирование сети, обученной методом обратного распространения ошибки

Нейронная сеть была протестирована на непохожих на тренировочную выборку рисунках и показала неплохую точность.

**Вывод**: разработано программное обеспечение для решения задачи бинарной классификации с учителем с использованием машины опорных векторов.

**ПРИЛОЖЕНИЕ А**

(обязательное)

**Текст программы**

**3lab.py:**

**import numpy as np**

from PIL import Image

import random as rnd

import matplotlib.pyplot as plt

import matplotlib.lines as mlines

from sklearn.datasets import make\_blobs, make\_circles

from matplotlib.colors import ListedColormap

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, 0, 0], [0, 0, 150]])/255.0

  svm\_model.fit(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='autumn')

  plt.contour(xx, yy, z\_model, colors='k', 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)

  plt.show()

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))

def sigmoid(x):

  # функция активации: f(x) = 1 / (1 + e^(-x))

  return 1 / (1 + np.exp(-x))

def add\_bias\_feature(a):

    a\_extended = np.zeros((a.shape[0],a.shape[1]+1))

    a\_extended[:,:-1] = a

    a\_extended[:,-1] = int(1)

    return a\_extended

def newline(p1, p2, color=None): # функция отрисовки линии

    ax = plt.gca()

    xmin, xmax = ax.get\_xbound()

    if(p2[0] == p1[0]):

        xmin = xmax = p1[0]

        ymin, ymax = ax.get\_ybound()

    else:

        ymax = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])\*(xmax-p1[0])

        ymin = p1[1]+(p2[1]-p1[1])/(p2[0]-p1[0])\*(xmin-p1[0])

    l = mlines.Line2D([xmin,xmax], [ymin,ymax], color=color)

    ax.add\_line(l)

    return l

f = open("C:\\Users\\ferru\\OneDrive\\Рабочий стол\\5semester\\Нейронки\\3lab\\config.txt", mode="r", encoding="utf-8")

y\_train = []

y\_test = []

train = []

test = []

flag = True

for line in f:

    if(line == ""):

        break

    if(line == "==\n"):

        flag = False

        continue

    if(flag):

        if(line.find("car") == -1):

            y\_train.append(1)

        else:

            y\_train.append(0)

        tr = Image.open(line[:-1], 'r').convert('L')

        arr = np.array(tr)

        train.append(arr.ravel())

    else:

        if(line.find("car") == -1):

            y\_test.append(1)

        else:

            y\_test.append(0)

        tr = Image.open(line[:-1], 'r').convert('L')

        arr = np.array(tr)

        test.append(arr.ravel())

for line in train:

    for i in range(len(line)):

        if(line[i] == 0):

            line[i] = 1

        else:

            line[i] = 0

for line in test:

    for i in range(len(line)):

        if(line[i] == 0):

            line[i] = 1

        else:

            line[i] = 0

train = np.array(train)

test = np.array(test)

y\_train = np.array(y\_train)

y\_test = np.array(y\_test)

for i in range(len(y\_train)):

        if(y\_train[i] == 0):

            y\_train[i] = 1

        else:

            y\_train[i] = 0

for i in range(len(y\_test)):

        if(y\_test[i] == 0):

            y\_test[i] = 1

        else:

            y\_test[i] = 0

class SVM:

  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

  # ограничение параметра t, чтобы новые лямбды не покидали границ квадрата

  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 fit(self, X, y):

    self.X = X.copy()

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

    self.y = y \* 2 - 1

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

    # формула (3)

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

    # выполняем self.max\_iter итераций

    for \_ in range(self.max\_iter):

      # проходим по всем лямбда

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

        # idxL выбираем случайно

        idxL = np.random.randint(0, len(self.lambdas))

        # формула (4с)

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

        # формула (4a)

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

        # формула (4b)

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

        # формула (4d)

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

        # регуляризированная формула (5), регуляризация только для idxM = idxL

        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))

svm =  SVM(kernel='linear', max\_iter=100)

svm.fit(train, y\_train)

from sklearn.svm import SVC

print("kernel Linear")

y\_pred = svm.predict(test)

X = []

for i in range(len(y\_pred)):

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

X = np.array(X)

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

test\_plot(X, y\_test, SVM(kernel='linear', C=10, max\_iter=60), axs, 'linear (my)')

print("Results:")

print(y\_pred)

graftest = []

for i in test:

    graftest.append(sum(i))

print("Results inLib:")

svm = SVC(kernel='linear', C=10)

svm.fit(train, y\_train)

print(svm.predict(test))

y\_pred = svm.predict(test)

y\_pred[y\_pred != y\_test] = -100 # find and mark classification error

linearSVC = (y\_pred == -100).astype(int).sum()

print('Количество ошибок для отложенной выборки: ', linearSVC)