Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение

высшего образования «НИТУ МИСиС»

Институт ИТАСУ

Кафедра ИК

Отчет по домашнему заданию №1

“Генератор датасетов”

по дисциплине

«Нейронные Сети»

Выполнил:

Студент группы

ММ-15-1

Замыслов Степан

Москва 2019

Оглавление

[Структурная модель программы 3](#_Toc2801752)

[PointClass 3](#_Toc2801753)

[Model 4](#_Toc2801754)

[Описание функционала и особенностей использования 4](#_Toc2801755)

[Примеры Визуализации 4](#_Toc2801756)

[Линейно разделимые множества 4](#_Toc2801757)

[Линейно неразделимые множества 8](#_Toc2801758)

# Описание функционала

Программа позволяет генерировать датасеты в виде многомерных сфер. Первое множество задается случайным образом, последующие генерируются с учетом количества оставшихся пересечений и необходимой площади пересечений. Когда количество признаков равно двум выводится график. В программе используются следующие функции:

* *generateClass(pointCount, varCount, centerPoint, radius)* – генерация класса (сферы) с определенным кол-вом точек, признаков, центром и радиусом.
* *checkIntersectionArea(d, r)* – проверка площади пересечения двух множеств
* *countPointDistance(point1,point2)* – рассчет расстояния между центрами классов
* *checkIntersectionCount(centers, point, radius)* – проверка количества пересечений между уже существующими множествами и точкой (потенциальным центром нового класса)
* *generateCenter(centers, varCount, radius, intersectionsLeft)* – метод для генерации центра нового множества с учетом количества недостающих пересечений
* *generatePoints(pointCount, classCount, varCount, radius, intersectionCount)* – метод для генерации необходимого количества классов с заданным кол-вом точек, признаков, радиусом сфер и количеством пересечений.

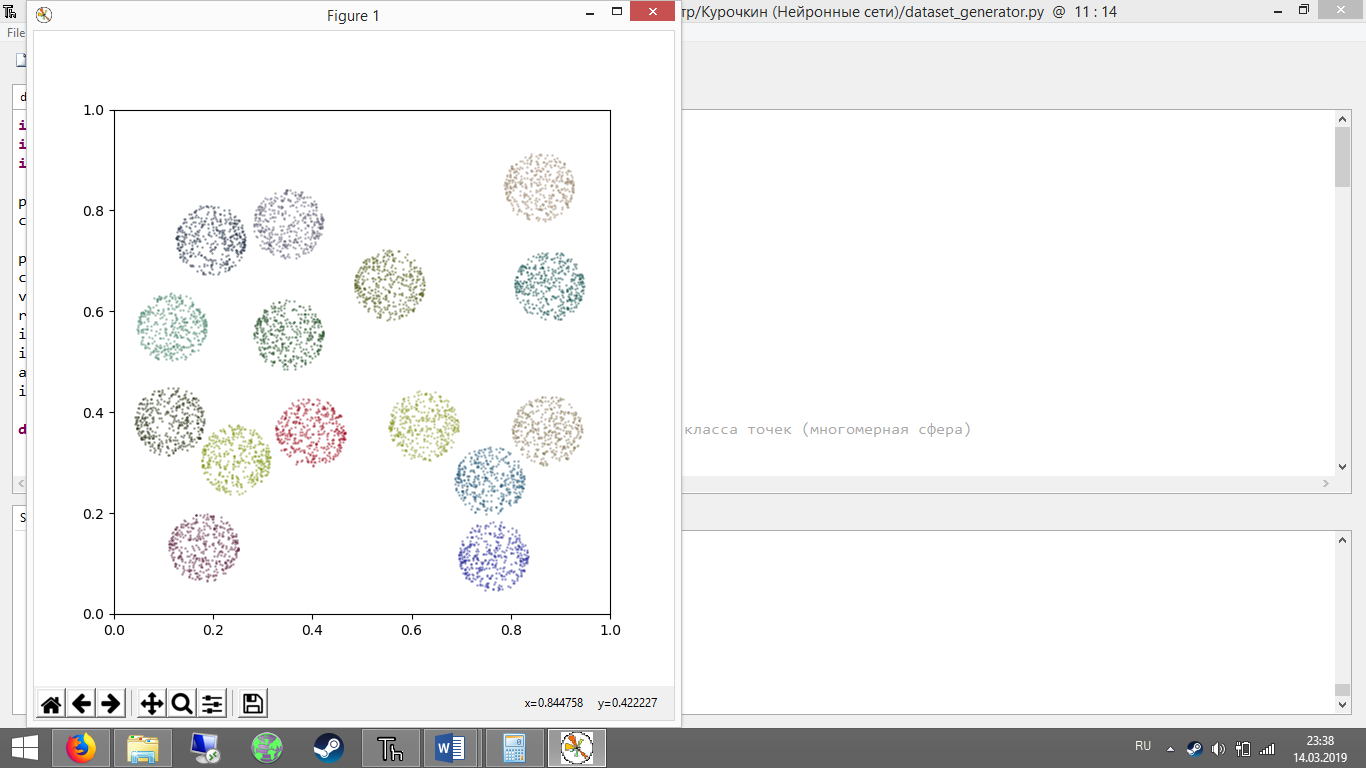
# Примеры Визуализации

## Линейно разделимые множества

pointCount = 400 # Количество точек в классе

classCount = 15 # Количество классов

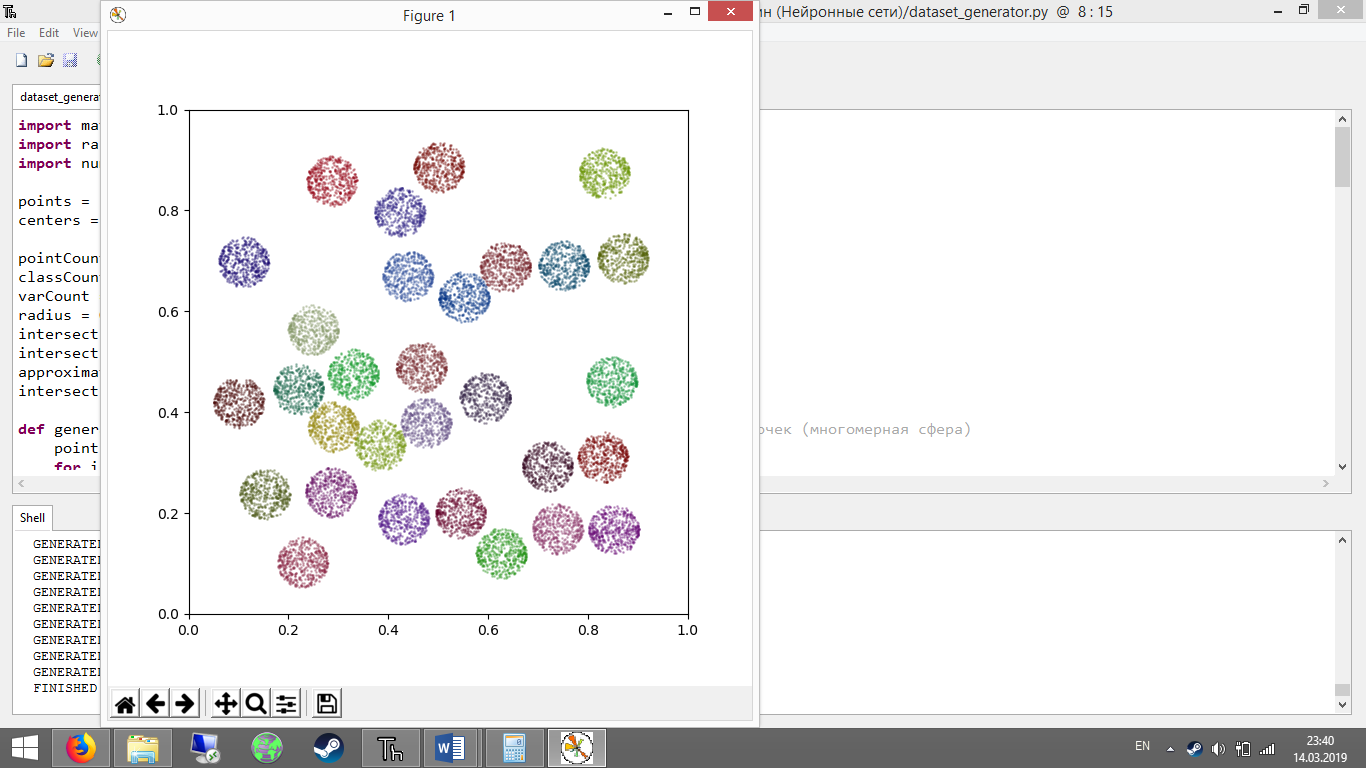
radius = 0.07 # Радиус разброса



pointCount = 500 # Количество точек в классе

classCount = 30 # Количество классов

radius = 0.05 # Радиус разброса



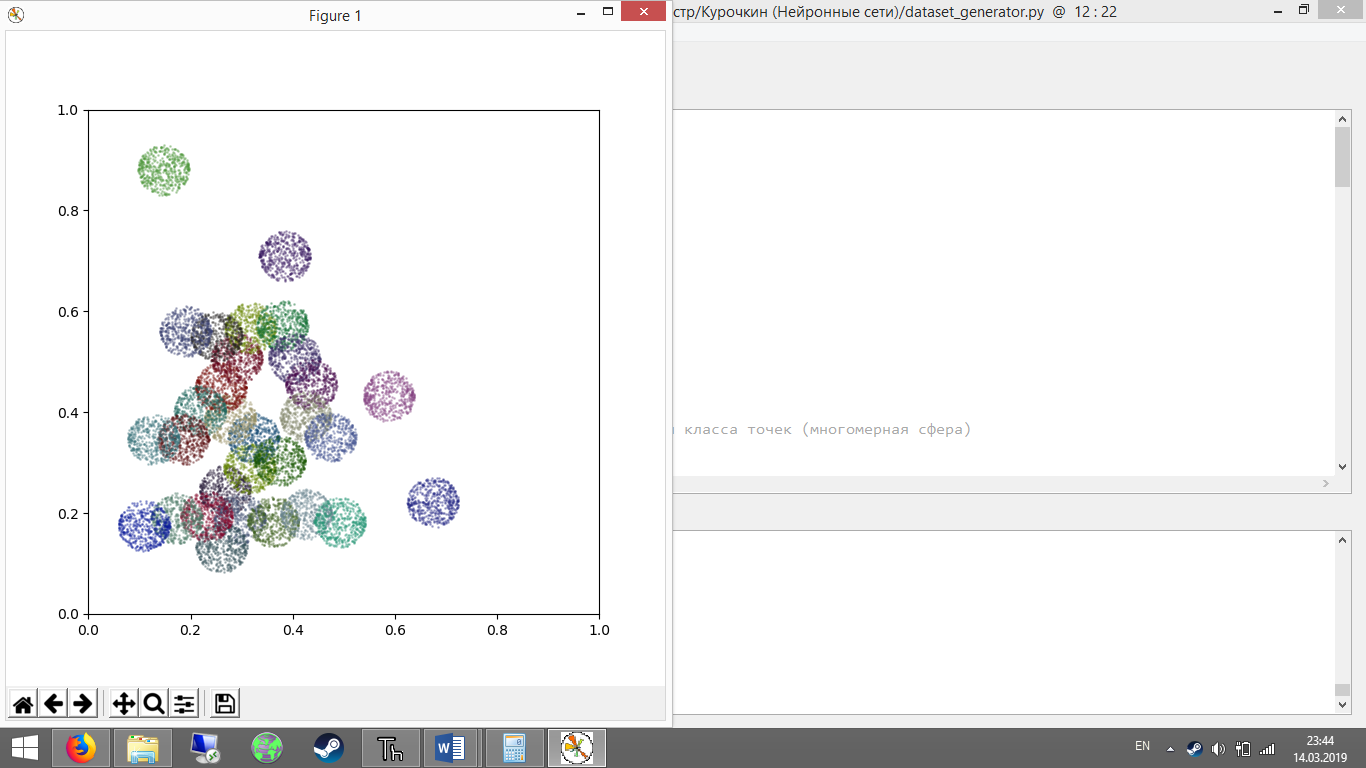
pointCount = 500 # Количество точек в классе

classCount = 30 # Количество классов

intersectionCount = 30 # Количество пересечений

intersectionArea = 0.25 # Площадь пересечения

approximation = 0.05 # Разброс точности при подсчете пересечения



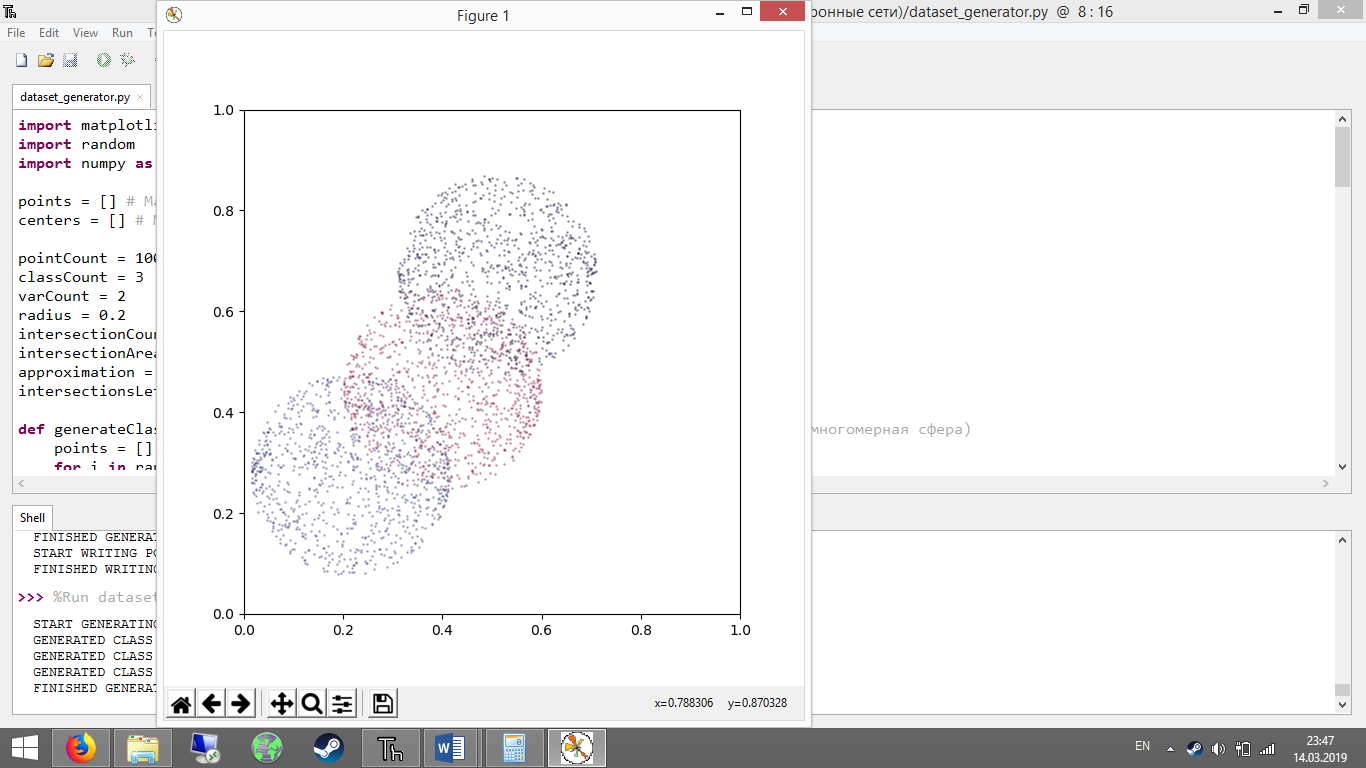
pointCount = 1000 # Количество точек в классе

classCount = 3 # Количество классов

intersectionCount = 2 # Количество пересечений

intersectionArea = 0.3 # Площадь пересечения

approximation = 0.05 # Разброс точности при подсчете пересечения



# Исходный код

import matplotlib.pyplot as pyplot

import random

import numpy as math

points = [] # Массив точек

centers = [] # Массив центров классов

pointCount = 1000 # Количество точек в классе

classCount = 3 # Количество классов

varCount = 2 # Количество признаков

radius = 0.2 # Радиус сферы

intersectionCount = 3 # Количество пересечений

intersectionArea = 0.3 # Площадь пересечения

approximation = 0.05 # Разброс точности при подсчете пересечения

intersectionsLeft = intersectionCount

def generateClass(pointCount, varCount, centerPoint, radius): # Генерация класса точек (многомерная сфера)

points = []

for i in range(0, pointCount):

points.append([])

maxVal = radius\*\*2

for j in range (0, varCount):

p = random.uniform(-math.sqrt(maxVal), math.sqrt(maxVal))

points[i].append(centerPoint[j] + p)

maxVal = maxVal - p\*\*2

return points

def randomColor(): # Задание случайного цвета для отображения множества

colorArr = []

for j in range(30):

color = '#'

for i in range(6):

color += str(random.randint(0, 9))

colorArr.append(color)

return colorArr

def visualize(points): # Метод для визуализации множеств

colorArr = randomColor()

for i in range(points.\_\_len\_\_()):

x = []

y = []

for j in range(points[i].\_\_len\_\_()):

x.append(points[i][j][0])

y.append(points[i][j][1])

pyplot.scatter(x,y,c=colorArr[i],alpha=0.3,s=1)

pyplot.gca().set\_xlim([0, 1])

pyplot.gca().set\_ylim([0, 1])

pyplot.show()

def checkIntersectionArea(d, r): # Метод для подсчета площади пересечения классов

x1 = 2\*math.arcsin(math.sqrt(r\*\*2 - (d\*\*2)/4)/r)

x2 = math.sin(2\*math.arcsin(math.sqrt(r\*\*2 - (d\*\*2)/4)/r))

return (x1 - x2)/math.pi

def countPointDistance(point1,point2): # Метод для расчета расстояния между центрами двух классов

dist = 0

for i in range(len(point1)):

dist += (point2[i]-point1[i])\*\*2

return math.sqrt(dist)

def checkIntersectionCount(centers, point, radius): # Метод для подсчета пересечений нового множества с имеющимися множествами. Возвращает количество пересечений

count = 0

for center in centers:

if (countPointDistance(center,point) < radius\*2):

count += 1

return count

def generateCenter(centers, varCount, radius, intersectionsLeft): # Метод для генерации центра множества с учетом количества оставшихся пересечений

if (len(centers) == 0): # Первый центр ставится случайным образом

newCenter = []

for j in range(0, varCount):

newCenter.append(random.uniform(0.1,0.9))

#print('generate first center: ',newCenter)

return newCenter

while True:

newCenter = []

for j in range(0, varCount):

newCenter.append(random.uniform(0.1,0.9))

intCount = checkIntersectionCount(centers, newCenter, radius)

if (intersectionsLeft > 0):

if (intCount > 0 and intCount <= intersectionsLeft):

flag = True

for center in centers:

area = checkIntersectionArea(countPointDistance(center,newCenter),radius)

if (area > 0):

if not (area > intersectionArea - approximation and area < intersectionArea + approximation):

flag = False

if (flag):

intersectionsLeft -= intCount

#print('generate center: {0} , int={1}, left={2}'.format(newCenter,intCount,intersectionsLeft))

return newCenter

if (intersectionsLeft == 0):

if (intCount == 0):

#print('generate center: {0} , int={1}, left={2}'.format(newCenter,intCount,intersectionsLeft))

return newCenter

def generatePoints(pointCount, classCount, varCount, radius, intersectionCount): # Метод для генерации классов множеств

points = []

centers = []

intersectionsLeft = intersectionCount

for i in range(0,classCount):

newCenter = generateCenter(centers, varCount, radius, intersectionsLeft)

intersectionsLeft -= checkIntersectionCount(centers, newCenter, radius)

centers.append(newCenter)

points.append(generateClass(pointCount, varCount, newCenter, radius))

print('GENERATED CLASS ',i)

return points

print('START GENERATING POINTS')

points = generatePoints(pointCount, classCount, varCount, radius, intersectionCount)

print('FINISHED GENERATING POINTS')

if (varCount == 2):

visualize(points)

print('START WRITING POINTS TO FILE')

text\_file = open('output\_p{0}\_cl{1}\_var{2}\_int{3}.txt'.format(pointCount, classCount, varCount, intersectionCount), "w")

for set in points:

text\_file.write(str(set)+'\n')

print('FINISHED WRITING POINTS TO FILE')

text\_file.close()