**Н И Т У « М И С и С »**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ОТЧЁТ**

по

**ЛАБОРАТОРНОЙ РАБОТЕ**

**«Однослойный персептрон»**

Учебная дисциплина «Нейронные сети»

**Группа:** БПМ-16-2

**Студент:** Новицкий Дмитрий

**Преподаватель:** доц., к.т.н. Курочкин И.И.

**Отметка:**

**Дата защиты:**

**2019 г.**

Оглавление

[Постановка задачи 3](#_Toc24406785)

[Основное условие 3](#_Toc24406786)

[Результаты и визуализация 3](#_Toc24406787)

[Демонстрация работы 3](#_Toc24406788)

[Входные/выходные данные 4](#_Toc24406789)

[Описание работы программы 5](#_Toc24406790)

[Считывание датасета из файла 5](#_Toc24406791)

[Проверка данных на корректность 5](#_Toc24406792)

[Конвертация 5](#_Toc24406793)

[Определение вспомогательных данных 6](#_Toc24406794)

[Генерация случайных значений для весов 6](#_Toc24406795)

[Скорость обучения и количество эпох 6](#_Toc24406796)

[Разделение датасета на множества 7](#_Toc24406797)

[Нормализация 7](#_Toc24406798)

[Единичный вход для нейронов 7](#_Toc24406799)

[Обучение персептрона 8](#_Toc24406800)

[Отрисовка графиков 10](#_Toc24406801)

[Проверка на валидационной выборке 11](#_Toc24406802)

[Демонстрация работы программы 12](#_Toc24406803)

[Двумерные данные 12](#_Toc24406804)

[Мой датасет 14](#_Toc24406805)

[Сторонний датасет (Роман Рачеев) 15](#_Toc24406806)

# Постановка задачи

## Основное условие

1. Самостоятельно реализовать однослойный перцептрон для решения задач классификации.
2. Количество слоев (полноценных нейронов): 1.
3. Количество нейронов в слое от 1 до 200.
4. Предусмотреть единичный вход для нейронов.
5. Функции активации по вариантам (вариант 12 – softsign activation function) + сигмоидальная.
6. Разделение множества на 3 части (обучающее, валидационное и тестовое) должно происходить случайным образом.

## **Результаты и визуализация**

1. Реализовать визуализацию результатов для 2-хмерных данных с визуализацией границ разделения различных классов и истинной принадлежности точек классам. (К примеру, при разделении точек трех линейно неразделимых классов однослойным перцептроном, для точек использовать маркеры разной формы и/или разного цвета + залить области принадлежности разных классов с точки зрения перцептрона).
2. Реализовать визуализацию динамики ошибки для обучающего и валидационного множеств в процессе обучения.
3. Качество классификации показать с помощью: Accuracy, Precision, Recall и СКО (среднеквадартичной ошибки).

## **Демонстрация работы**

Продемонстрировать работу нескольких сценариев:

1. На разных датасетах, в том числе с количеством признаков 2 (для визуализации).
2. По разделимости классов: линейно разделимое множество, линейно неразделимое множество (средняя площадь пересечения классов 10-20%), линейно неразделимое множество (средняя площадь пересечения классов 50-70%).

## **Входные/выходные данные**

1. Входные данные (датасеты) в виде текстового файла. (к примеру, экспорт таблицы из Excel в формате TXT или CSV).
2. Выходные данные (результаты) сохраняются в виде, необходимом для формирования отчета PDF или DOCX.

# Описание работы программы

## Считывание датасета из файла

Для начала задаётся путь к файлу и инициализируется список, который в дальнейшем будет содержать данные из датасета. После чего вызывается функция считывания данных из файла.

my\_dataset\_path = "dataset/htru\_2.csv"

file\_matrix = []

file\_matrix = functions.read\_from\_file(my\_dataset\_path)

Функция чтения данных из файла выглядит следующим образом:

def read\_from\_file(path):

file\_reader = open(path)

file\_mas = []

all\_file = file\_reader.read()

file\_mas = []

file\_mas = all\_file.split("\n")

file\_matrix = []

for i in file\_mas:

file\_matrix.append(i.split(","))

return file\_matrix

## Проверка данных на корректность

Проверяем датасет (матрицу с исходными данными) на корректность с помощью функции «check\_dataset».

functions.check\_dataset(file\_matrix)

Функция выглядит следующим образом:

def check\_dataset(file\_matrix):

for i in range(len(file\_matrix)):

if(len(file\_matrix[0]) != len(file\_matrix[i])):

print("Датасет испорчен. Количество столбцов различное")

exit(0)

## Конвертация

Так как данные хранятся в строковом типе данных, то, для работы с ними, необходимо конвертировать их в тип данных float:

functions.converting(file\_matrix)

Функция выглядит следующим образом:

def converting(file\_matrix):

try:

for i in range(len(file\_matrix)):

for j in range(len(file\_matrix[0]) - 1):

file\_matrix[i][j] = float(file\_matrix[i][j])

except:

print("Плохие значения в датасете!")

exit()

# Конвертируем классы датасета

try:

for i in range(len(file\_matrix)):

file\_matrix[i][len(file\_matrix[0]) - 1] = float(file\_matrix[i][len(file\_matrix[0]) - 1])

except ValueError:

dict = {}

counter = 0

for i in range(len(file\_matrix)):

try:

file\_matrix[i][len(file\_matrix[0]) - 1] = dict[file\_matrix[i][len(file\_matrix[0]) - 1]]

except KeyError:

dict[file\_matrix[i][len(file\_matrix[0]) - 1]] = counter

file\_matrix[i][len(file\_matrix[0]) - 1] = counter

counter = counter + 1

## Определение вспомогательных данных

Определение количества классов, количества нейронов и количества весов осуществляется с помощью нижеприведённого кода. Полагается, что количество нейронов равно количеству классов в датасете.

signs = []

signs.append(file\_matrix[0][len(file\_matrix[0]) - 1])

for i in range(len(file\_matrix)):

find = 0

for j in range(len(signs)):

if(signs[j] == file\_matrix[i][len(file\_matrix[i]) - 1]):

find = 1

if(find == 0):

signs.append(file\_matrix[i][len(file\_matrix[i]) - 1])

count\_of\_classes = len(signs)

count\_of\_neurons = count\_of\_classes

lines\_count\_of\_weights = len(file\_matrix[0])

columns\_count\_of\_weights = count\_of\_neurons

## Генерация случайных значений для весов

Случайные значения для весов генерируются случайным образом по следующему алгоритму:

synaptic\_weights = []

for i in range(lines\_count\_of\_weights):

helper\_mas = []

for j in range(columns\_count\_of\_weights):

helper\_mas.append(2 \* np.random.random() - 1)

synaptic\_weights.append(helper\_mas)

## Скорость обучения и количество эпох

Ввод с клавиатуры скорость обучения персептрона и количество эпох для обучения персептрона.

print("Введите скорость обучения персептрона:")

speed = float(input())

print("Введите количество эпох обучения:")

count\_of\_eras = int(input())

## Разделение датасета на множества

Разделение исходного датасета на обучающее, тестовое и валидационное множества происходит следующим образом:

learn\_training\_inputs = []

learn\_training\_outputs = []

validation\_training\_inputs = []

validation\_training\_outputs = []

test\_training\_inputs = []

test\_training\_outputs = []

# Количество % из всего датасета для тестового множества

test\_probability = 20

# Количество % из всего датасета для валидационного множества

validation\_probability = 10

# Разделение исходного датасета на множества случайным способом

for i in range(len(file\_matrix)):

random\_number = random.randint(1, 100)

if(random\_number > test\_probability + validation\_probability):

learn\_training\_inputs.append(file\_matrix[i])

learn\_training\_outputs.append(file\_matrix[i][len(file\_matrix[i]) - 1])

learn\_training\_inputs[len(learn\_training\_inputs) - 1].pop()

elif(random\_number <= validation\_probability):

validation\_training\_inputs.append(file\_matrix[i])

validation\_training\_outputs.append(file\_matrix[i][len(file\_matrix[i]) - 1])

validation\_training\_inputs[len(validation\_training\_inputs) - 1].pop()

else:

test\_training\_inputs.append(file\_matrix[i])

test\_training\_outputs.append(file\_matrix[i][len(file\_matrix[i]) - 1])

test\_training\_inputs[len(test\_training\_inputs) - 1].pop()

## Нормализация

Нелинейная нормализация входных данных происходит следующим образом:

learn\_training\_inputs = functions.not\_linear\_matrix\_normalization(learn\_training\_inputs)

validation\_training\_inputs = functions.not\_linear\_matrix\_normalization(validation\_training\_inputs)

test\_training\_inputs = functions.not\_linear\_matrix\_normalization(test\_training\_inputs)

Функция нормализации выглядит следующим образом:

def not\_linear\_matrix\_normalization(file\_matrix):

a = 0.5 # Коэффициент нормализации

for i in range(len(file\_matrix[0])):

average = 0

for j in range(len(file\_matrix)):

average = average + file\_matrix[j][i]

average = average / len(file\_matrix)

for j in range(len(file\_matrix)):

file\_matrix[j][i] = 1 / (np.exp((-1) \*a\* (file\_matrix[j][i] - average)) + 1)

return file\_matrix

## Единичный вход для нейронов

Единичный вход для нейронов осуществляется следующим образом:

for i in range(len(learn\_training\_inputs)):

learn\_training\_inputs[i].append(1)

for i in range(len(test\_training\_inputs)):

test\_training\_inputs[i].append(1)

for i in range(len(validation\_training\_inputs)):

validation\_training\_inputs[i].append(1)

## Обучение персептрона

Обучение персептрона происходит следующим образом:

activation\_function = "sigmoid"

#activation\_function = "softsign"

MSE\_mas\_learning = []

MSE\_mas\_test = []

accuracy\_mas\_learning = []

accuracy\_mas\_test = []

counter = 0

min\_MSE = -1

min\_MSE\_era = 0

best\_synaptic\_weights = []

while(counter < count\_of\_eras):

print("Эпоха обучения №", counter + 1)

print("Сейчас идёт обучающая выборка")

functions.learning\_function(learn\_training\_inputs, learn\_training\_outputs, synaptic\_weights, speed, activation\_function, MSE\_mas\_learning, accuracy\_mas\_learning, count\_of\_neurons)

print("Сейчас идёт тестовая выборка")

functions.test\_function(test\_training\_inputs, test\_training\_outputs, synaptic\_weights, activation\_function, MSE\_mas\_test, accuracy\_mas\_test, count\_of\_neurons, 0)

counter = counter + 1

if((min\_MSE == -1) or (MSE\_mas\_test[len(MSE\_mas\_test) - 1] < min\_MSE)):

min\_MSE = MSE\_mas\_test[len(MSE\_mas\_test) - 1]

min\_MSE\_era = counter

best\_synaptic\_weights = synaptic\_weights

Функция «learning\_function» выглядит следующим образом:

def learning\_function(inputs, outputs, weights, speed, activation\_function, MSE\_mas, accuracy\_mas, count\_of\_neurons):

# MSE - Mean Squared Error (среднеквадратичная ошибка)

MSE = 0

TP = 0 # True Positive (Правильно определена 1)

FP = 0 # False Positive (Неправильно определена 1)

FN = 0 # False Negative (Неправильно определён 0)

TN = 0 # True Negative (Правильно определён 0)

for j in range(len(inputs)):

output = []

err = []

for i in range(count\_of\_neurons):

value\_synaptic\_weights = []

for k in range(len(weights)):

value\_synaptic\_weights.append(weights[k][i])

sum = scalar(inputs[j], value\_synaptic\_weights)

if(activation\_function == "softsign"):

output.append(softsign\_activation\_function(sum))

if(activation\_function == "sigmoid"):

output.append(sigmoid\_activation\_function(sum))

if(i == outputs[j]):

err.append(1 - output[i])

else:

err.append(-output[i])

MSE = MSE + (err[i] \* err[i])

for i in range(count\_of\_neurons):

if((round(output[i]) == 0) and (i != outputs[j])):

TN = TN + 1

if((round(output[i]) == 0) and (i == outputs[j])):

FN = FN + 1

if((round(output[i]) == 1) and (i == outputs[j])):

TP = TP + 1

if((round(output[i]) == 1) and (i != outputs[j])):

FP = FP + 1

if(activation\_function == "softsign"):

for i in range(len(weights)):

for k in range(len(weights[0])):

weights[i][k] = weights[i][k] + speed \* err[k] \* inputs[j][i]

if(activation\_function == "sigmoid"):

for i in range(len(weights)):

for k in range(len(weights[0])):

weights[i][k] = weights[i][k] + speed \* err[k] \* inputs[j][i]

MSE = MSE / len(inputs)

MSE\_mas.append(MSE)

print("Среднеквадратичная ошибка в данной эпохе составила", MSE)

print("TP = ", TP)

print("FP = ", FP)

print("FN = ", FN)

print("TN = ", TN)

accuracy\_mas.append((TP + TN) / (TP + TN + FP + FN))

print("accurancy = ", (TP + TN) / (TP + TN + FP + FN))

print("precision = ", TP / (TP + FP))

print("recall = ", TP / (TP + FN))

Функция «test\_function» выглядит следующим образом:

def test\_function(inputs, outputs, weights, activation\_function, MSE\_mas, accuracy\_mas, count\_of\_neurons, marker):

# MSE - Mean Squared Error (среднеквадратичная ошибка)

MSE = 0

TP = 0 # True Positive (Правильно определена 1)

FP = 0 # False Positive (Неправильно определена 1)

FN = 0 # False Negative (Неправильно определён 0)

TN = 0 # True Negative (Правильно определён 0)

for j in range(len(inputs)):

output = []

err = []

for i in range(count\_of\_neurons): # len(synaptic\_weights = кол-во столбцов входных данных)

#count\_of\_neurons = count\_of\_signs

value\_synaptic\_weights = []

for k in range(len(weights)):

value\_synaptic\_weights.append(weights[k][i])

sum = scalar(inputs[j], value\_synaptic\_weights)

if(activation\_function == "softsign"):

output.append(softsign\_activation\_function(sum))

if(activation\_function == "sigmoid"):

output.append(sigmoid\_activation\_function(sum))

if(i == outputs[j]):

err.append(1 - output[i])

else:

err.append(-output[i])

MSE = MSE + (err[i] \* err[i])

count\_of\_repeats = 0

for i in range(count\_of\_neurons):

if(round(output[i]) == 1):

count\_of\_repeats = count\_of\_repeats + 1

if(count\_of\_repeats == 1):

for i in range(count\_of\_neurons):

if((round(output[i]) == 1) and (i == outputs[j])):

TP = TP + 1

if((round(output[i]) == 1) and (i != outputs[j])):

FN = FN + 1

else:

FP = FP + 1

if(marker == 1):

if(i < 100):

print("outputs = ", output)

print("test\_training\_outputs[", i, "] = ", outputs[i])

print()

MSE = MSE / len(inputs)

MSE\_mas.append(MSE)

print("Среднеквадратичная ошибка в данной эпохе составила", MSE)

print("TP = ", TP)

print("FP = ", FP)

print("FN = ", FN)

print("TN = ", TN)

accuracy\_mas.append((TP + TN) / (TP + TN + FP + FN))

print("accurancy = ", (TP + TN) / (TP + TN + FP + FN))

print("precision = ", TP / (TP + FP))

print("recall = ", TP / (TP + FN))

## Отрисовка графиков

Отрисовка графиков для значений среднеквадратичной ошибки происходит следующим образом:

functions.graph\_drawing\_function(MSE\_mas\_learning, MSE\_mas\_test, accuracy\_mas\_learning, accuracy\_mas\_test)

Функция «graph\_drawing\_function» выглядит следующим образом:

def graph\_drawing\_function(MSE\_mas\_learning, MSE\_mas\_test, accuracy\_mas\_learning, accuracy\_mas\_test):

# Массив для значений x

helper\_mas = []

for i in range(len(MSE\_mas\_learning)):

helper\_mas.append(i)

fig = plt.figure()

MSE\_graph = fig.add\_subplot(1, 2, 1)

MSE\_graph.set\_title("green - MSE for learning, blue - MSE for validation")

MSE\_graph.scatter(helper\_mas, MSE\_mas\_learning, color = 'green', marker = '\*')

MSE\_graph.scatter(helper\_mas, MSE\_mas\_test, color = 'blue', marker = '\*')

MSE\_graph.plot(helper\_mas, MSE\_mas\_learning, color = 'green')

MSE\_graph.plot(helper\_mas, MSE\_mas\_test, color = 'blue')

accuracy\_graph = fig.add\_subplot(1, 2, 2)

accuracy\_graph.set\_title("green - accuracy for learning, blue - accuracy for validation")

accuracy\_graph.scatter(helper\_mas, accuracy\_mas\_learning, color = 'green', marker = '\*')

accuracy\_graph.scatter(helper\_mas, accuracy\_mas\_test, color = 'blue', marker = '\*')

accuracy\_graph.plot(helper\_mas, accuracy\_mas\_learning, color = 'green')

accuracy\_graph.plot(helper\_mas, accuracy\_mas\_test, color = 'blue')

plt.show()

## Проверка на валидационной выборке

Проверка работы программы на валидационной выборке осуществляется следующим образом:

MSE\_mas\_validation = []

accuracy\_mas\_validation = []

functions.test\_function(validation\_training\_inputs, validation\_training\_outputs, best\_synaptic\_weights, activation\_function, MSE\_mas\_validation, accuracy\_mas\_validation, count\_of\_neurons, 1)

# Демонстрация работы программы

## Двумерные данные

Напишем функцию, которая будет генерировать двумерные данные. Для удобства генерации данных будем использовать sklearn. Функция генерации данных выглядит следующим образом:

from sklearn.datasets.samples\_generator import make\_blobs

import matplotlib.pyplot as plt

X, y = make\_blobs(n\_samples = 1000,n\_features=2, centers=2,cluster\_std = 1,center\_box=(-8.0,8.0),shuffle=False)

plt.figure()

plt.scatter(X[:, 0], X[:, 1], c=y)

plt.show()

a = open('dataset/new\_generated\_dataset.txt', 'w')

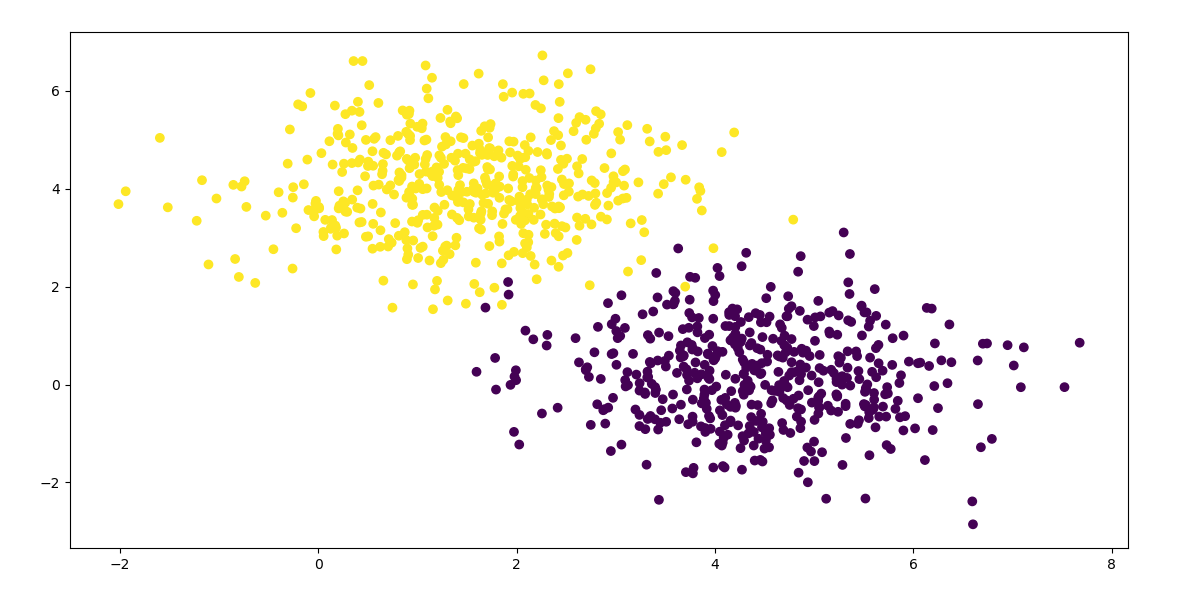
for i in range(X.shape[0]):

string = str(X[i][0]) + ',' + str(X[i][1]) + ',' + str(y[i]) + '\n'

a.write(string)

a.close()

Сгенерируем 1000 данных для датасета с двумя классами. Пусть значения классов на плоскости будут распределяться следующим образом:



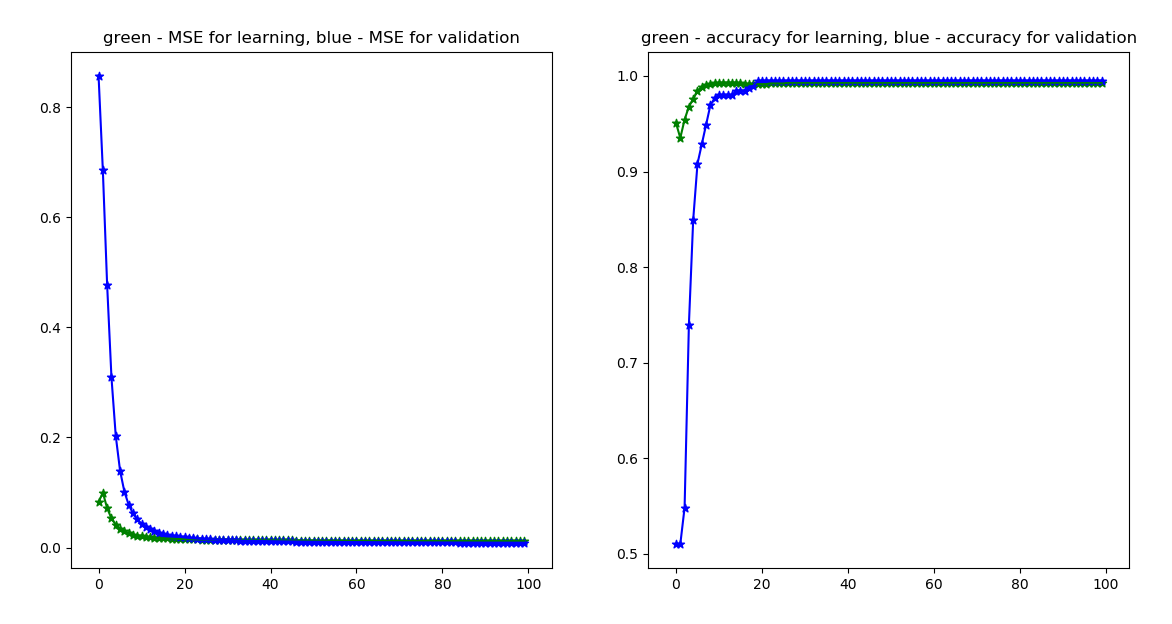
*Рис. 1. Двумерные данные на плоскости*

В таком случае графики для ошибок будут выглядеть следующим образом:

Количество эпох обучения – 100.

Коэффициент обучения – 0,1.

Функция активации – сигмоида.

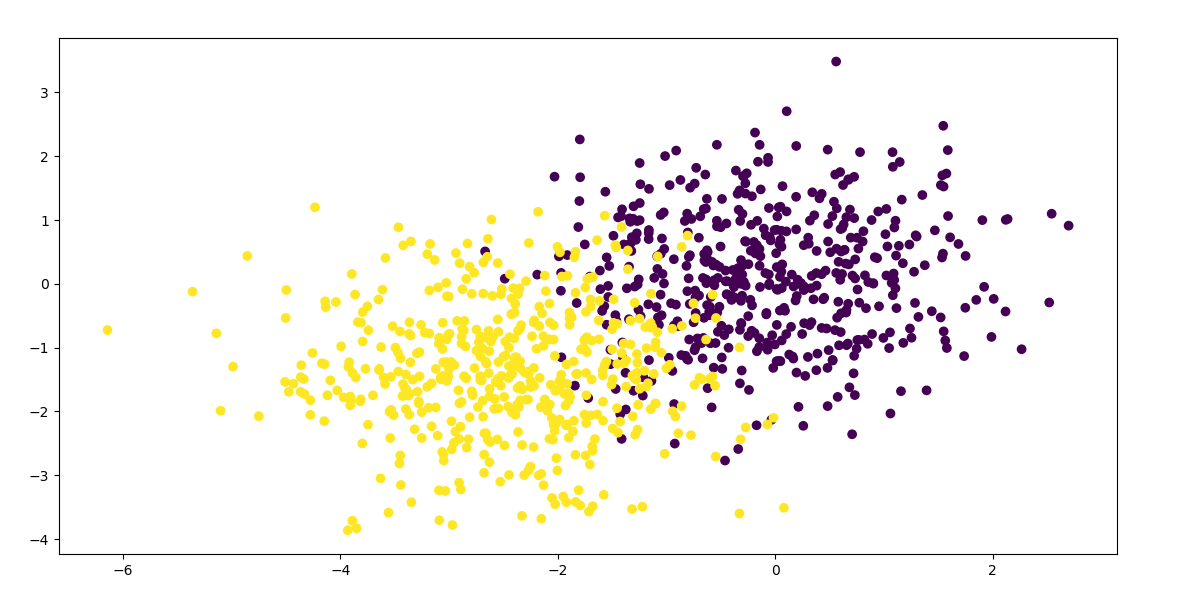


*Рис. 2. Значения СКО и accuracy для определённой эпохи.*



*Рис. 3. Значения ошибок на валидационной выборке*

Сгенерируем датасет с большей площадью пересечения классов:



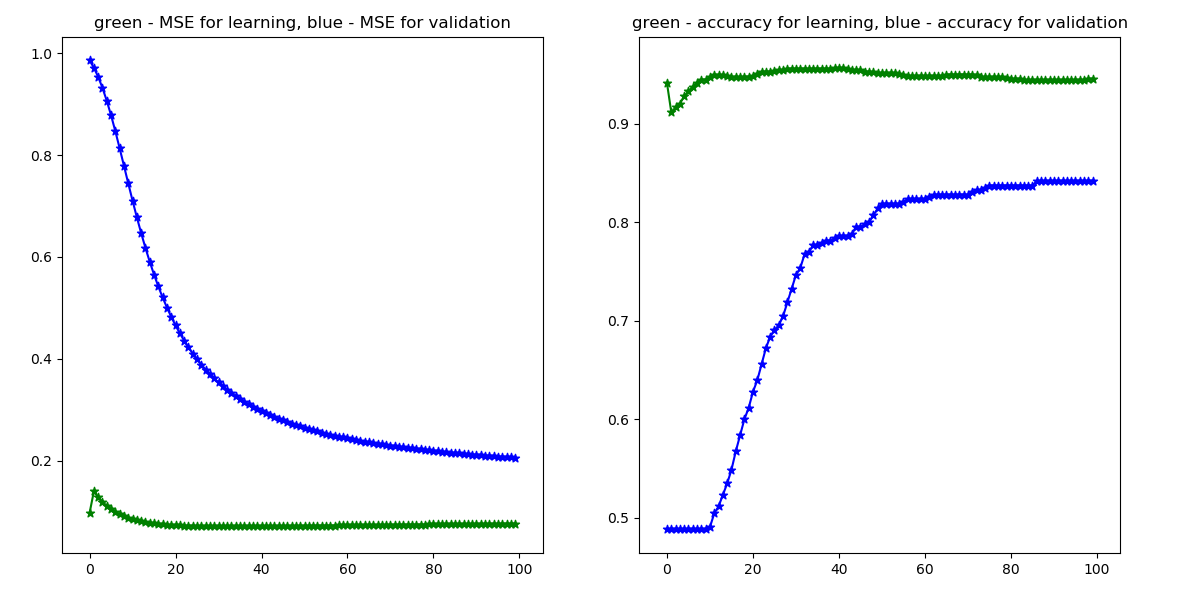
*Рис. 4. Двумерные данные на плоскости*

В таком случае графики для ошибок будут выглядеть следующим образом:

Количество эпох обучения – 100.

Коэффициент обучения – 0,1.

Функция активации – сигмоида.



*Рис. 5. Значения СКО и accuracy для определённой эпохи.*



*Рис. 6. Значения СКО и accuracy для определённой эпохи.*

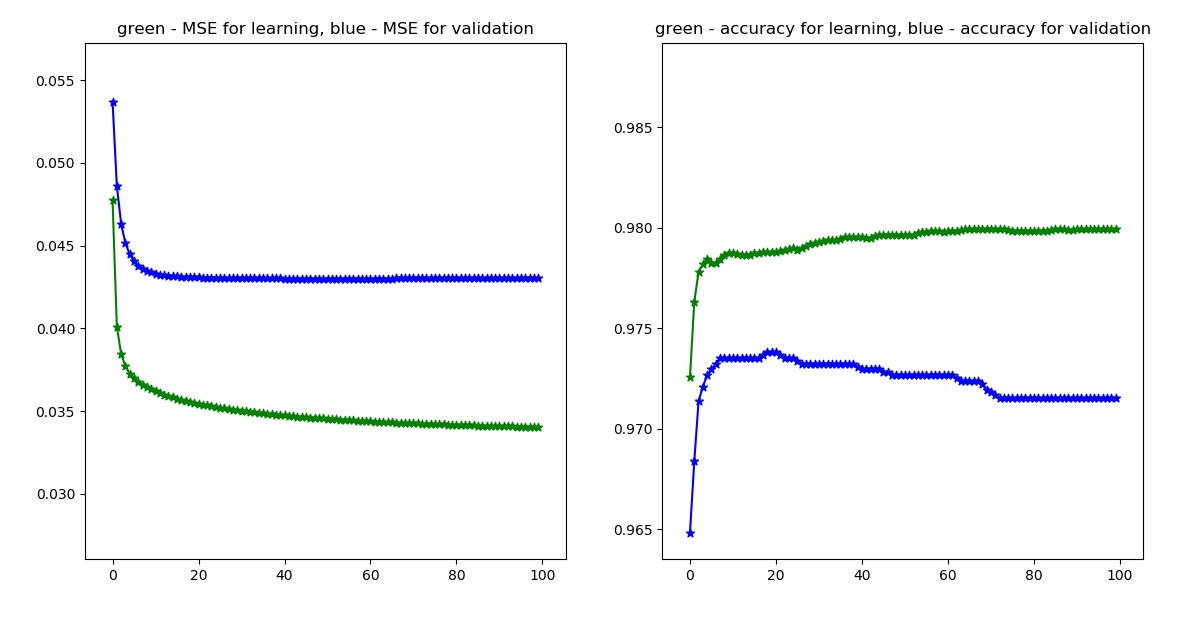
## Мой датасет

На моём датасете (predictiong a pulsar star) графики для ошибок будут выглядеть следующим образом:

Количество эпох обучения – 100.

Коэффициент обучения – 0,1.

Функция активации – сигмоида.



*Рис. 7. Значения СКО и accuracy для определённой эпохи.*



*Рис. 8. Значения СКО и accuracy для определённой эпохи.*

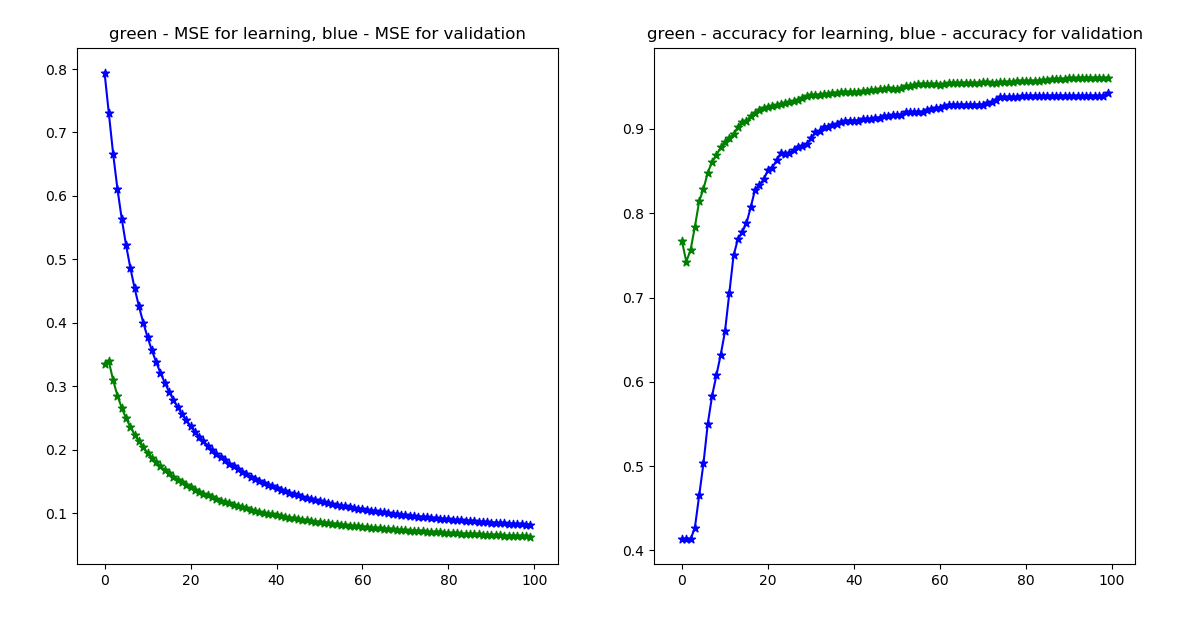
## Сторонний датасет (Роман Рачеев)

Попробуем изменять коэффициент обучения и смотреть, как меняются графики ошибок и итоговая точность.

Пусть:

Коэффициент обучения – 0,01.

Функция активации – сигмоида, тогда



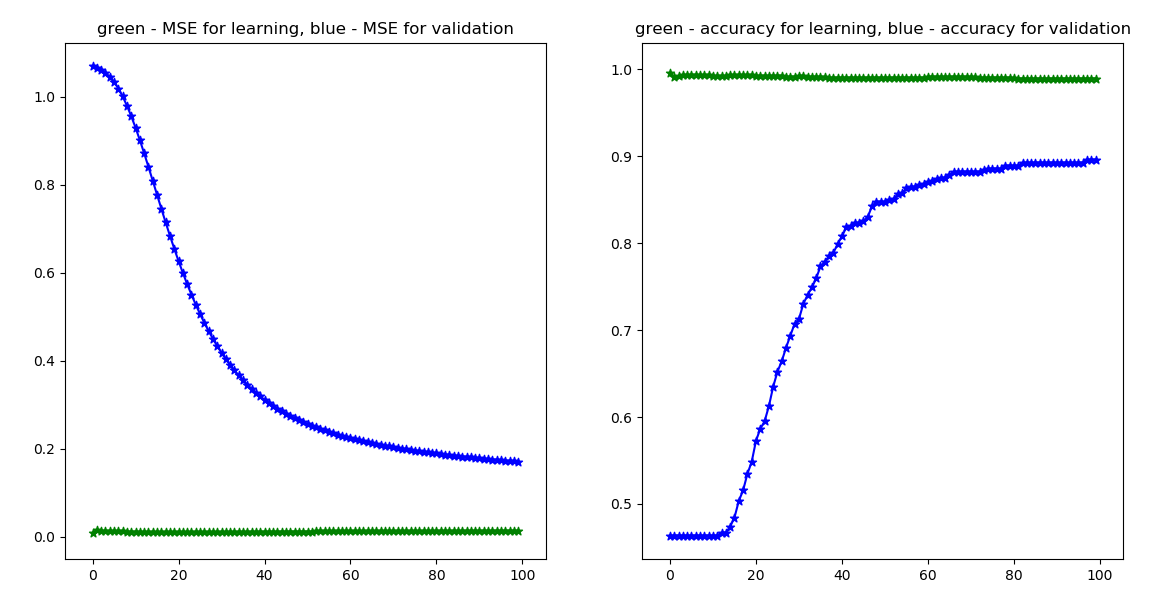
*Рис. 9. Значения СКО и accuracy для определённой эпохи.*



*Рис. 10. Значения СКО и accuracy для определённой эпохи.*

Коэффициент обучения – 1.

Функция активации – сигмоида, тогда



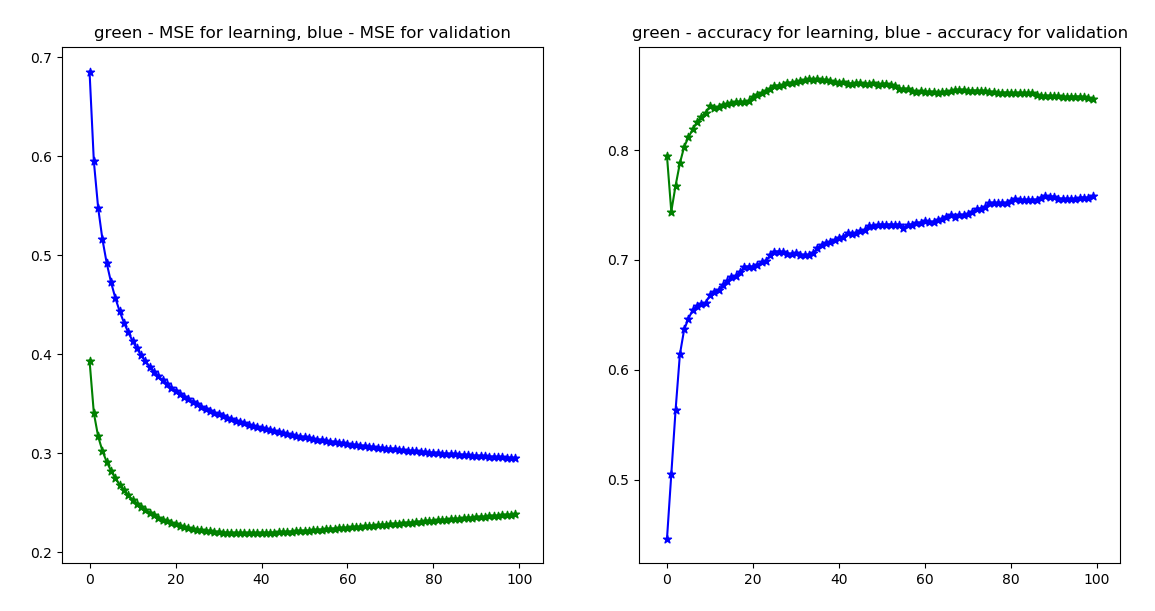
*Рис. 11. Значения СКО и accuracy для определённой эпохи.*



*Рис. 12. Значения СКО и accuracy для определённой эпохи.*

Коэффициент обучения – 0,01.

Функция активации – softsign, тогда



*Рис. 13. Значения СКО и accuracy для определённой эпохи.*



*Рис. 14. Значения СКО и accuracy для определённой эпохи.*