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

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:

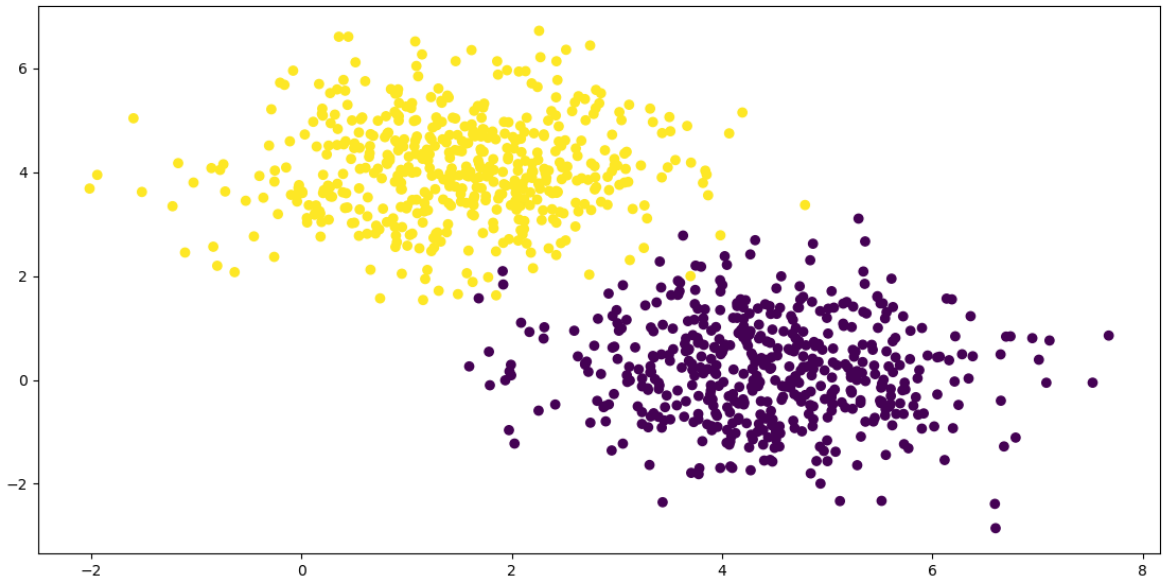


Рис. 1. Двумерные данные на плоскости

В таком случае графики для ошибок будут выглядеть следующим образом:

Количество эпох обучения – 100.

Коэффициент обучения – 0,1.

Функция активации – сигмоида.

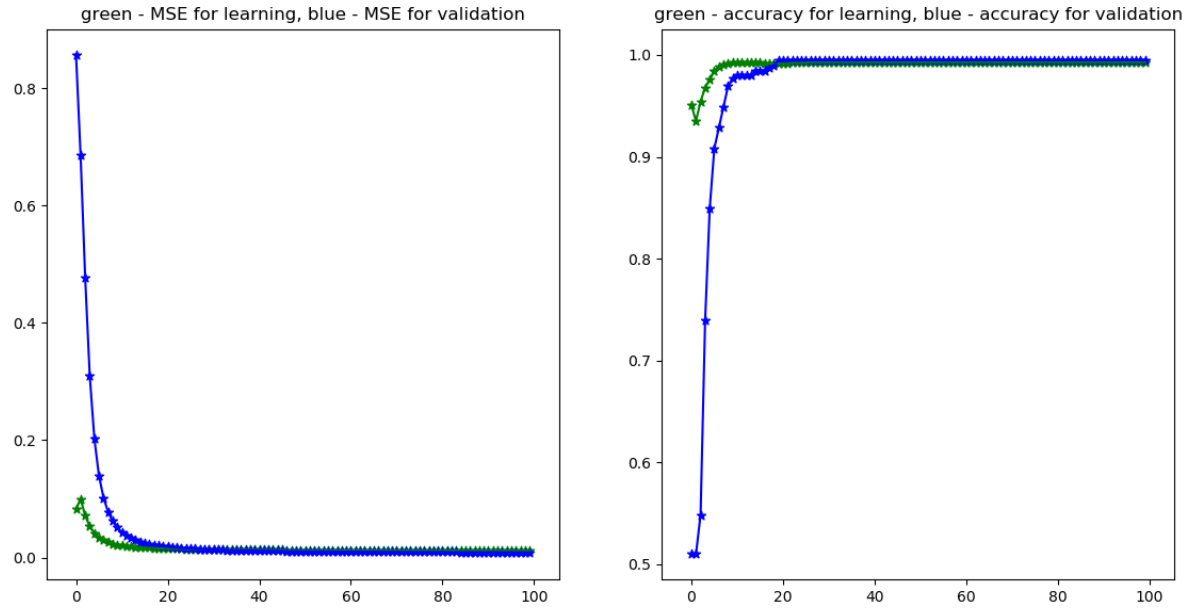


Рис. 2. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.970873786407767
precision =  0.970873786407767
recall =  0.970873786407767
```

Рис. 3. Значения ошибок на валидационной выборке

Сгенерируем датасет с большей площадью пересечения классов:

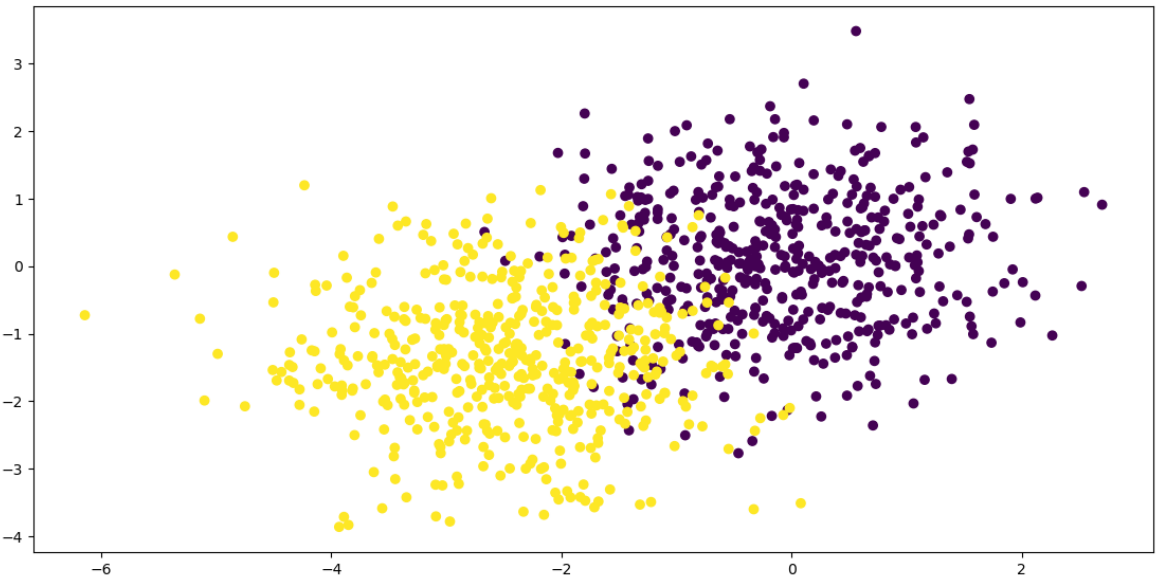


Рис. 4. Двумерные данные на плоскости

В таком случае графики для ошибок будут выглядеть следующим образом:

Количество эпох обучения – 100.

Коэффициент обучения – 0,1.

Функция активации – сигмоида.

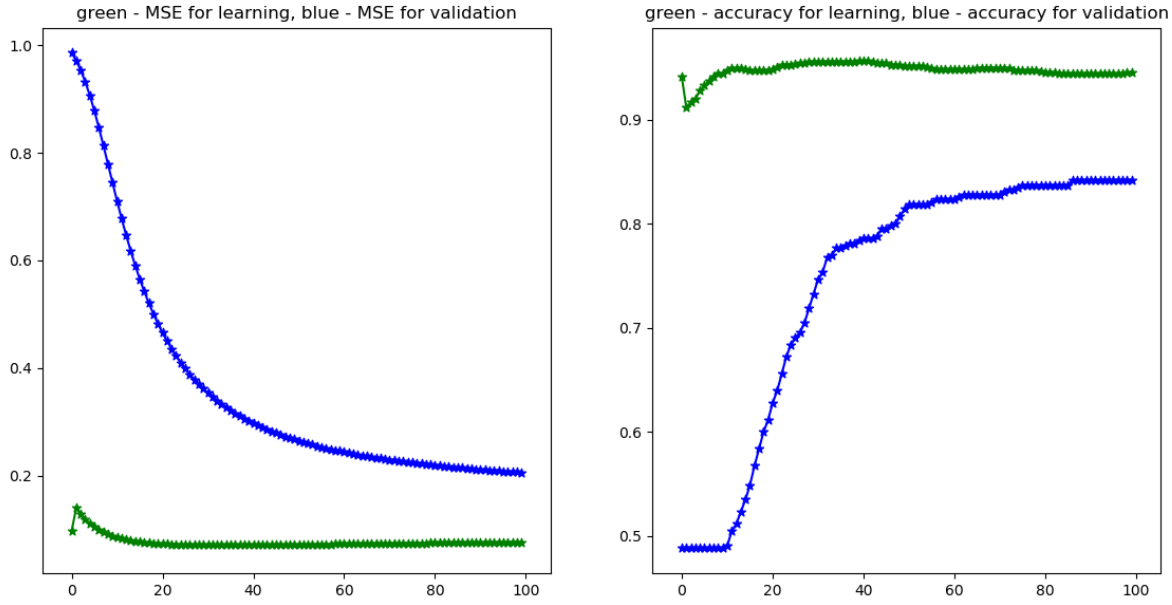


Рис. 5. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.897196261682243
precision =  0.897196261682243
recall =  0.897196261682243
```

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.

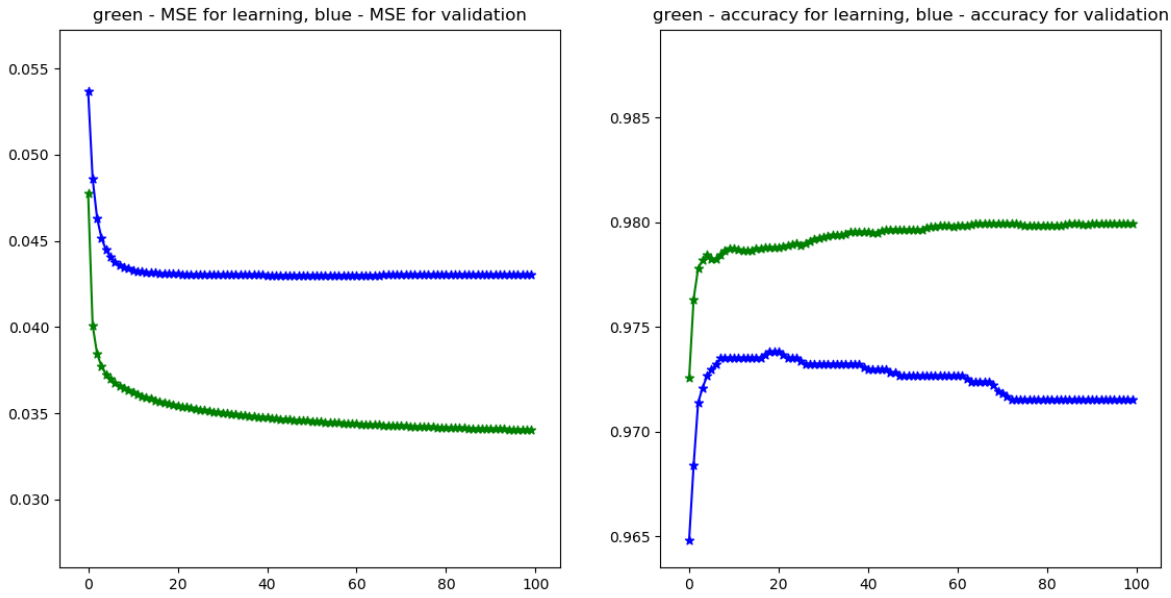


Рис. 7. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.9684329199549042
precision =  0.9684329199549042
recall =  0.9684329199549042
```

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

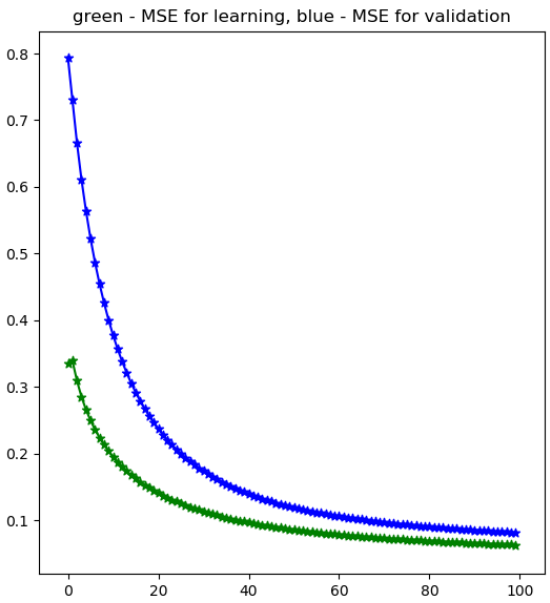
Сторонний датасет (Роман Рачеев)

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

Пусть:

Коэффициент обучения – 0,01.

Функция активации – сигмоида, тогда



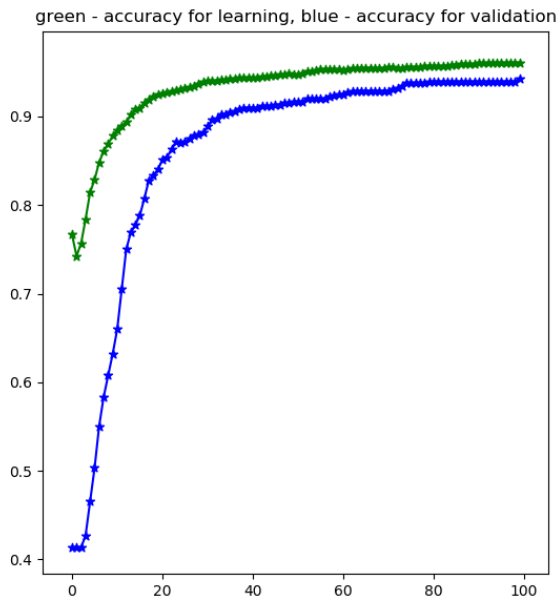


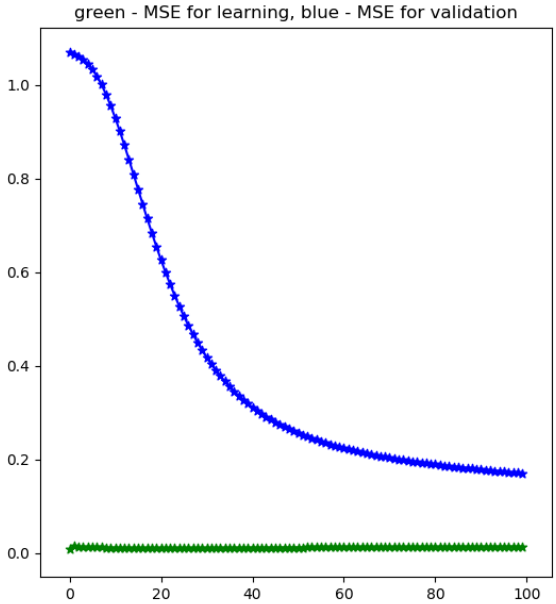
Рис. 9. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.9525316455696202
precision =  0.949685534591195
recall =  0.9556962025316456
```

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

Коэффициент обучения – 1.

Функция активации – сигмоида, тогда



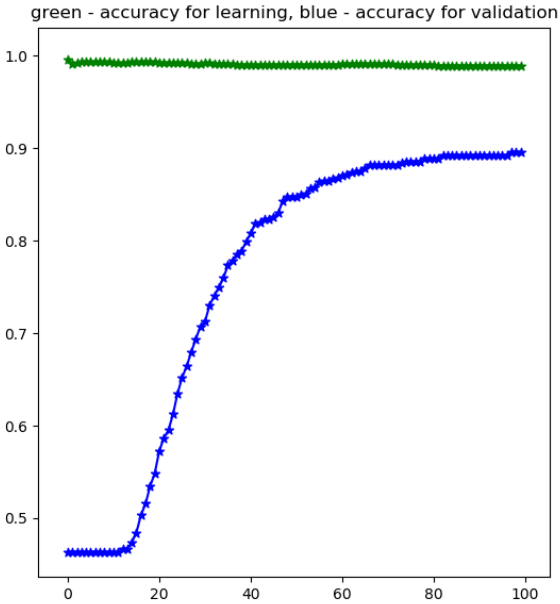


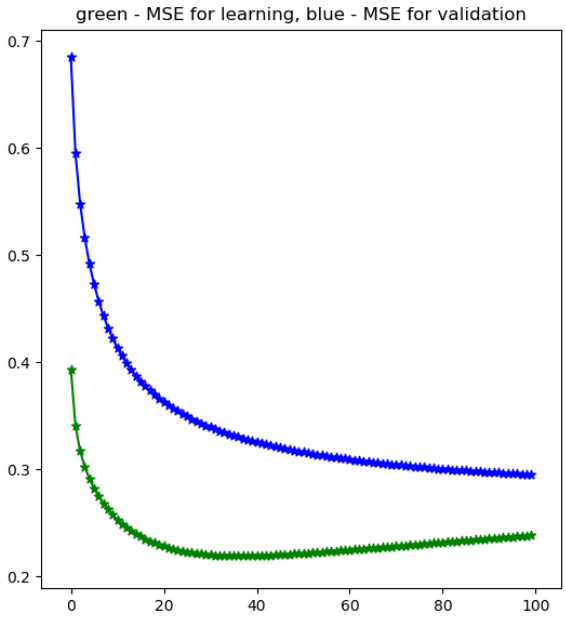
Рис. 11. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.873015873015873
precision =  0.873015873015873
recall =  0.873015873015873
```

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

Коэффициент обучения – 0,01.

Функция активации – softsign, тогда



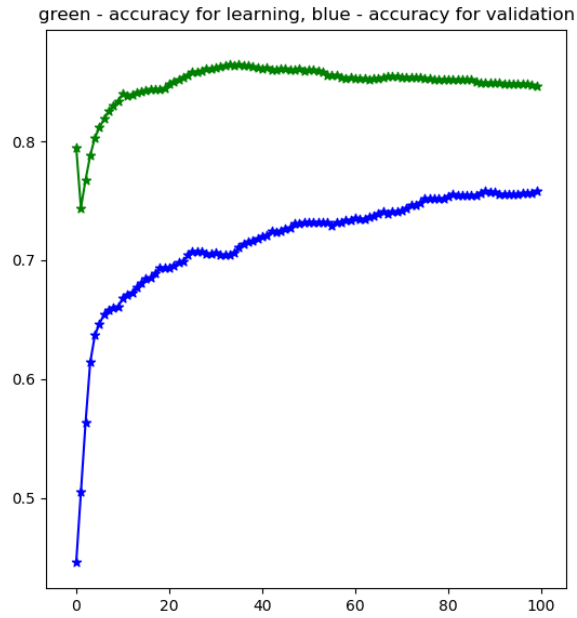


Рис. 13. Значения СКО и accuracy для определённой эпохи.

```
accurancy =  0.7254901960784313
precision =  0.6551724137931034
recall =  1.0
```

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