Министерство образования Республики Беларусь

Учреждение образования

«Брестский государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Модели решения задач в интеллектуальных системах»

Тема: «Бинарная классификация»

Выполнил:

Студент 3 курса

Группы ИИ-26

Кушнеревич Е.А

Проверила:

Андренко К.В.

Брест 2026

Цель работы: Изучить принципы бинарной классификации и реализовать однослойную нейронную сеть (персептрон) для решения задачи классификации с использованием пороговой функции активации, а также исследовать процесс обучения модели с применением среднеквадратичной ошибки (MSE).

Ход работы:

1. Реализовать алгоритм обучения однослойной нейронной сети с использованием **MSE** в качестве функции ошибки.

2. Провести обучение сети с **разными значениями шага обучения** и построить **график зависимости** MSE от номера эпохи.

3. Выполнить визуализацию результатов классификации:

- исходные точки обучающей выборки,
- разделяющую линию (границу между двумя классами).

4. Реализовать режим функционирования сети:

- пользователь задаёт произвольный входной вектор,
- сеть вычисляет выходной класс,
- соответствующая точка отображается на графике,
- для корректной визуализации рекомендуется выбирать значения из диапазона ВСТАВИТЬ СВОЙ ДИАПАЗОН, например $-0.5 \leq x_1, x_2 \leq 1.5$

8.

x_1	x_2	e
5	6	0
-5	6	1
5	-6	1
-5	-6	1

x_1, x_2 - входные данные сети, e - эталонные значения

Код программы:

```
import random
import numpy as np
import matplotlib.pyplot as plt

dataset = [
    {'x1':  5,  'x2': 6, 'label':0},
    {'x1': -5,  'x2': 6, 'label': 1},
    {'x1':  5,  'x2': -6, 'label':1},
    {'x1': -5,  'x2': -6, 'label': 1}
]

class Perceptron:
    def __init__(self, learning_rate=0.1):
```

```
        self.w1 = random.random() - 0.5
        self.w2 = random.random() - 0.5
        self.bias = random.random() - 0.5
        self.lr = learning_rate

    def activate(self, sum_val):
        return 1 if sum_val >= 0 else 0

    def train(self, dataset):
        error_sum = 0
        for vector in dataset:
            sum_val = (vector['x1'] * self.w1) + (vector['x2'] * self.w2) +
self.bias

            prediction = self.activate(sum_val)
            error = vector['label'] - prediction

            if error != 0:
                self.w1 += self.lr * error * vector['x1']
                self.w2 += self.lr * error * vector['x2']
                self.bias += self.lr * error

            error_sum += error ** 2

        return error_sum / len(dataset)

class Visualizer:
    def __init__(self):
        self.fig, (self.ax_mse, self.ax_decision) = plt.subplots(1, 2,
figsize=(12, 5))

    def draw_results(self, mse_history, brain, dataset, user_point=None):
        self.ax_mse.clear()
        self.ax_mse.plot(mse_history, color="#E2E20E", linewidth=2)
        self.ax_mse.set_title("График среднеквадратичной ошибки")
        self.ax_mse.set_xlabel("Epochs")
        self.ax_mse.set_ylabel("MSE")
        self.ax_mse.grid(True)

        self.ax_decision.clear()
        self.ax_decision.set_title("График с разделяющей поверхностью")

        self.ax_decision.axhline(0, color='#eee', linewidth=1)
        self.ax_decision.axvline(0, color='#eee', linewidth=1)

        x_vals = np.linspace(-5, 5, 100)
        if brain.w2 != 0:
            y_vals = (-brain.w1 * x_vals - brain.bias) / brain.w2
            self.ax_decision.plot(x_vals, y_vals, 'r-', label='Decision Line')

        for p in dataset:
```

```
            color = 'blue' if p['label'] == 1 else 'orange'
            self.ax_decision.scatter(p['x1'], p['x2'], color=color, s=100,
edgecolors='black', zorder=3)

        if user_point:
            ux, uy, u_label = user_point['x1'], user_point['x2'],
user_point['label']
            self.ax_decision.scatter(ux, uy, c='black', marker='s', s=120,
label=f"User: Class {u_label}", zorder=4)
            self.ax_decision.text(ux + 0.5, uy, f"Class {u_label}", fontsize=9)

        self.ax_decision.set_xlim(-8, 8)
        self.ax_decision.set_ylim(-8, 8)
        self.ax_decision.legend()
        self.ax_decision.grid(True, linestyle='--')
        plt.tight_layout()
        plt.show()

brain = Perceptron(learning_rate=0.01)
viz = Visualizer()
mse_history = []

print("Начинается процесс обучения")
for epoch in range(200):
    mse = brain.train(dataset)
    mse_history.append(mse)
    if epoch % 10 == 0:
        print(f"Epoch {epoch}: MSE = {mse}")
    if mse == 0:
        print(f"Процесс обучения завершено досрочно на эпохе: {epoch}")
        break

def predict_user_point(x1, x2):
    sum_val = x1 * brain.w1 + x2 * brain.w2 + brain.bias
    label = brain.activate(sum_val)
    print(f"User Point ({x1}, {x2}) -> Class: {label}")
    return {'x1': x1, 'x2': x2, 'label': label}

user_pt = predict_user_point(0, -3)

viz.draw_results(mse_history, brain, dataset, user_pt)
```

Результат тестирования:

Начинается процесс обучения

Epoch 0: MSE = 0.25

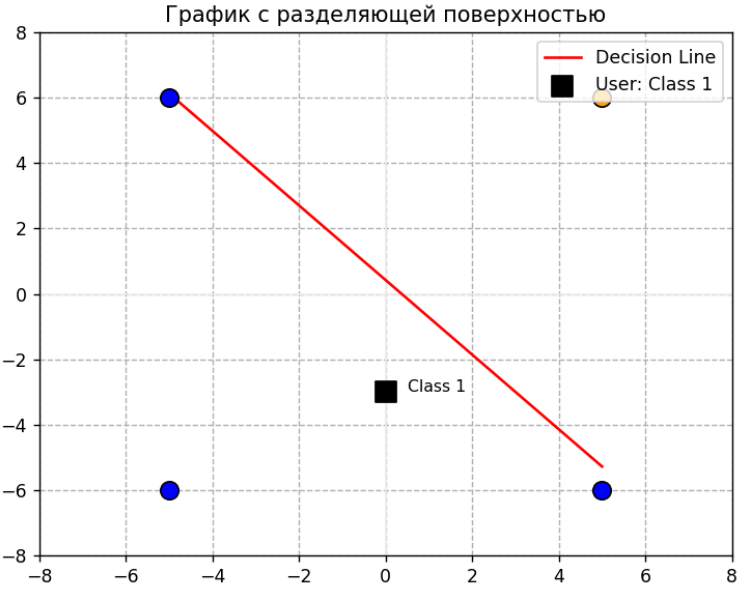
Epoch 10: MSE = 0.5

Процесс обучения завершено досрочно на эпохе: 11

User Point (0, -3) -> Class: 1

График среднеквадратичной ошибки:

График с разделяющей поверхностью:



Вывод: Проделав данную лабораторную работу я изучил принципы бинарной классификации и реализовал однослойную нейронную сеть (персептрон) для решения задачи классификации с использованием пороговой функции активации, а также исследовал процесс обучения модели с применением среднеквадратичной ошибки (MSE).