Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №1
По дисциплине : «Модели решения задач в ИС»
По теме : «Бинарная классификация»

Выполнил:
студент 3 курса
группы ИИ-26
Згера Е. А.
Проверил:
Адренко 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).

Вариант 5:

x_1	x_2	e
2	6	0
-2	6	1
2	-6	0
-2	-6	0

Ход работы :

Код программы:

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

def load_custom_dataset():
    data = [
        [2, 6, 0],
        [-2, 6, 1],
        [2, -6, 0],
        [-2, -6, 0]
    ]

    X = np.array([[point[0], point[1]] for point in data])
    y = np.array([point[2] for point in data])

    return X, y

def step_function(x):
    return 1 if x >= 0 else 0

class Perceptron:
    def __init__(self, input_size, learning_rate=0.1):
        self.weights = np.random.randn(input_size) * 0.1
        self.bias = np.random.randn() * 0.1
        self.learning_rate = learning_rate
        self.mse_history = []

    def predict(self, x):
        z = np.dot(x, self.weights) + self.bias
        return step_function(z)

    def fit(self, X, y, epochs=20):
        n_samples = X.shape[0]

        print("Начальные параметры:")
        print(f"  Веса: {self.weights}")
        print(f"  Смещение: {self.bias:.4f}")
        print("-" * 40)

        for epoch in range(epochs):
            total_error = 0

            print(f"Эпоха {epoch + 1}:")
            print(f"  Веса до обновления: {self.weights}, Bias: {self.bias:.4f}")

            for i in range(n_samples):
                z = np.dot(X[i], self.weights) + self.bias
                prediction = step_function(z)

                error = y[i] - prediction
                total_error += error ** 2

                weight_update = self.learning_rate * error * X[i]
                bias_update = self.learning_rate * error

                self.weights += weight_update
                self.bias += bias_update

                print(f"  Точка {i+1}: ({X[i,0]}, {X[i,1]}) -> z={z:.2f}, предсказание={prediction}, истинное={y[i]}, ошибка={error}")
                print(f"    Обновление весов: {weight_update}")
                print(f"    Обновление bias: {bias_update:.4f}")

            mse = total_error / n_samples
            self.mse_history.append(mse)

            print(f"  MSE эпохи: {mse:.4f}")
            print(f"  Веса после обновления: {self.weights}, Bias: {self.bias:.4f}")
            print("-" * 40)

    def decision_boundary(self, x):
        if abs(self.weights[1]) < 1e-10:
            return np.full_like(x, np.nan)
        return (-self.weights[0] * x - self.bias) / self.weights[1]

def visualize_results(X, y, perceptron):
    plt.figure(figsize=(10, 4))
```

```
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

    plt.subplot(1, 2, 1)
    epochs = range(1, len(perceptron.mse_history) + 1)
    plt.plot(epochs, perceptron.mse_history, 'b-o', linewidth=2, markersize=6)
    plt.xlabel('Номер эпохи')
    plt.ylabel('MSE')
    plt.title('Снижение ошибки по эпохам')
    plt.grid(True)
    plt.xticks(epochs)

    plt.subplot(1, 2, 2)
    colors = ['red' if label == 0 else 'blue' for label in y]
    plt.scatter(X[:, 0], X[:, 1], c=colors, edgecolors='k', s=150)

    predictions = [perceptron.predict(x) for x in X]
    for i, (x, y_val) in enumerate(X):
        plt.text(x, y_val, f'Предск.: {predictions[i]}\nИстин.: {y[i]}',
                 ha='center', va='bottom', fontsize=9,
                 bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.8))

    x_vals = np.linspace(x_min, x_max, 100)
    y_vals = perceptron.decision_boundary(x_vals)
    if not np.isnan(y_vals[0]):
        plt.plot(x_vals, y_vals, 'g--', linewidth=2, label='Разделяющая линия')

    xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                         np.linspace(y_min, y_max, 100))
    grid_points = np.c_[xx.ravel(), yy.ravel()]
    grid_predictions = np.array([perceptron.predict(p) for p in grid_points])
    grid_predictions = grid_predictions.reshape(xx.shape)

    plt.contourf(xx, yy, grid_predictions, alpha=0.2, cmap=plt.cm.RdYlBu)
    plt.xlabel('x1')
    plt.ylabel('x2')
    plt.title('Классификация и разделяющая линия')
    plt.xlim(x_min, x_max)
    plt.ylim(y_min, y_max)
    plt.grid(True)
    plt.legend()

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    X, y = load_custom_dataset()

    print("ПЕРСЕПТРОН С ПРАВИЛОМ ВИДРОУ-ХОФФА")

    print("ОБУЧАЮЩАЯ ВЫБОРКА:")
    for i in range(len(X)):
        print(f"  Точка {i+1}: ({X[i,0]}, {X[i,1]}) -> Класс {y[i]}")

    print("ПАРАМЕТРЫ ОБУЧЕНИЯ:")
    print(f"  Количество эпох: 20")
    print(f"  Learning rate: 0.1")

    perceptron = Perceptron(input_size=2, learning_rate=0.1)
    perceptron.fit(X, y, epochs=20)

    print("РЕЗУЛЬТАТЫ ОБУЧЕНИЯ")

    print(f"Финальные параметры модели:")
    print(f"  Веса: w1 = {perceptron.weights[0]:.6f}, w2 = {perceptron.weights[1]:.6f}")
    print(f"  Смещение (bias): {perceptron.bias:.6f}")

    print(f"Уравнение разделяющей линии:")
    print(f"  {perceptron.weights[0]:.4f}*x1 + {perceptron.weights[1]:.4f}*x2 + {perceptron.bias:.4f} = 0")

    print(f"MSE по эпохам:")
    for epoch, mse in enumerate(perceptron.mse_history, 1):
        print(f"  Эпоха {epoch:2d}: {mse:.6f}")

    print(f"Классификация обучающих данных:")
    correct_count = 0
    for i in range(len(X)):
        prediction = perceptron.predict(X[i])
        is_correct = prediction == y[i]
        if is_correct:
            correct_count += 1

        print(f"  ({X[i,0]}, {X[i,1]}) -> предсказание: {prediction}")

    accuracy = correct_count / len(X) * 100
    print(f"\nТочность на обучающей выборке: {accuracy:.1f}% ({correct_count}/{len(X)})")

    print("\nВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ")

    visualize_results(X, y, perceptron)
```

Результат тестирования:

РЕЗУЛЬТАТЫ ОБУЧЕНИЯ
Финальные параметры модели:
  Веса: w1 = -0.569613, w2 = 0.098768
  Смещение (bias): -0.574901
Уравнение разделяющей линии:
  -0.5696*x1 + 0.0988*x2 + -0.5749 = 0

График с разделяющей поверхностью:

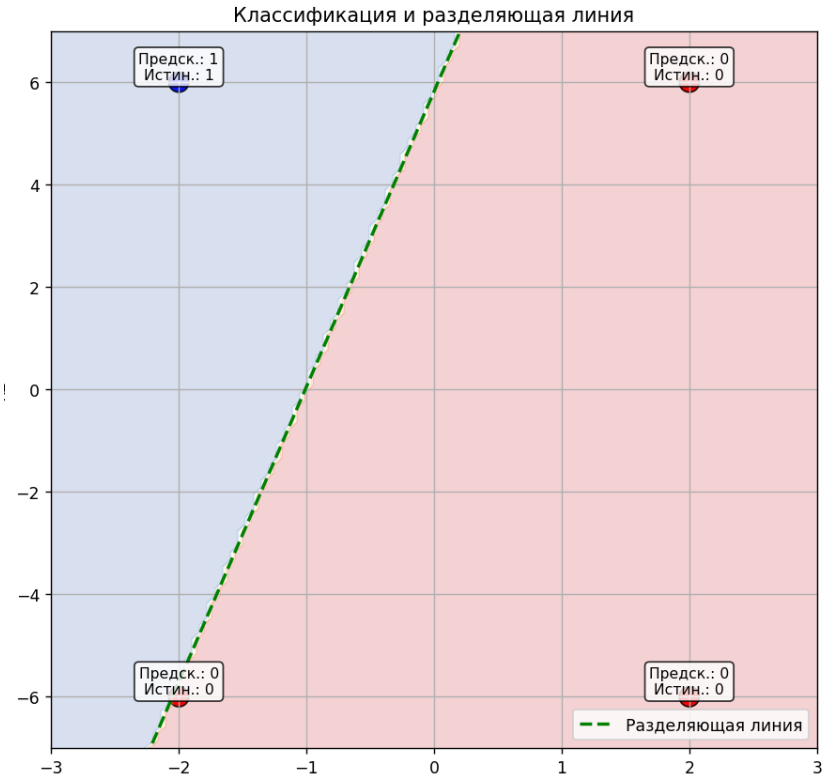
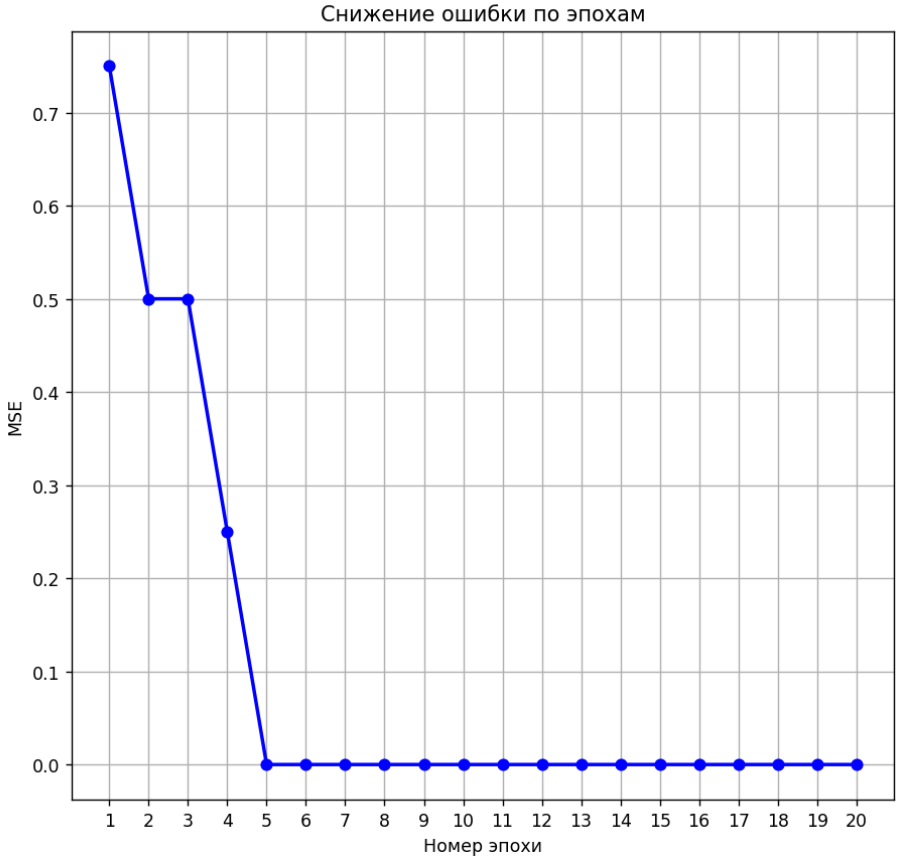


График изменения ошибки:



Вывод: 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).