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 Adaline:
    def __init__(self, input_size, learning_rate=0.001, momentum=0.9):
        self.weights = np.random.randn(input_size) * 0.01
        self.bias = np.random.randn() * 0.01
        self.learning_rate = learning_rate
        self.momentum = momentum
        self.prev_weight_update = np.zeros(input_size)
        self.prev_bias_update = 0
        self.mse_history = []
        self.weight_history = []
        self.bias_history = []
        self.best_epoch = 0

    def linear_output(self, x):
        return np.dot(x, self.weights) + self.bias

    def predict(self, x):
        z = self.linear_output(x)
        return step_function(z)

    def fit(self, X, y, epochs=50, patience=5):
        n_samples = X.shape[0]

        best_mse = float('inf')
        best_weights = None
        best_bias = None
        epochs_without_improvement = 0

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

        for epoch in range(epochs):
            total_error = 0
            epoch_weight_updates = []
            epoch_bias_updates = []

            for i in range(n_samples):
                linear_z = self.linear_output(X[i])

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

                weight_update = (self.learning_rate * error * X[i] +
                                 self.momentum * self.prev_weight_update)
                bias_update = (self.learning_rate * error +
                               self.momentum * self.prev_bias_update)

                self.prev_weight_update = weight_update.copy()
                self.prev_bias_update = bias_update

                epoch_weight_updates.append(weight_update.copy())
                epoch_bias_updates.append(bias_update)
```

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

            self.weight_history.append(self.weights.copy())
            self.bias_history.append(self.bias)

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

            if mse < best_mse - 0.0001:
                best_mse = mse
                best_weights = self.weights.copy()
                best_bias = self.bias
                self.best_epoch = epoch + 1
                epochs_without_improvement = 0
            else:
                epochs_without_improvement += 1

            if epochs_without_improvement >= patience:
                print(f"Ранняя остановка на эпохе {epoch + 1}")
                print(f"Лучшая эпоха: {self.best_epoch} с MSE = {best_mse:.6f}")
                self.weights = best_weights
                self.bias = best_bias
                break

            if epoch < 5 or epoch >= epochs-5:
                print(f"Эпоха {epoch + 1}:")
                print(f"  Momentum обновления:")
                for i, (w_update, b_update) in enumerate(zip(epoch_weight_updates, epoch_bias_updates)):
                    print(f"    Точка {i+1}: Δw = {w_update}, Δb = {b_update:.6f}")
                print(f"  Новые веса: {self.weights}")
                print(f"  Новое смещение: {self.bias:.4f}")
                print(f"  MSE: {mse:.6f}")
                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, adaline):
    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(adaline.mse_history) + 1)
    plt.plot(epochs, adaline.mse_history, 'b-o', linewidth=2, markersize=2)
    plt.xlabel('Номер эпохи')
    plt.ylabel('MSE')
    plt.title('Снижение ошибки по эпохам')
    plt.grid(True)

    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 = [adaline.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=8,
                 bbox=dict(boxstyle='round,pad=0.3', facecolor='white', alpha=0.8))

    x_vals = np.linspace(x_min, x_max, 100)
    y_vals = adaline.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([adaline.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("ОБУЧАЮЩАЯ ВЫБОРКА:")
    for i in range(len(X)):
        print(f"  Точка {i+1}: ({X[i,0]}, {X[i,1]}) -> Класс {y[i]}")

    print("\nПАРАМЕТРЫ ОБУЧЕНИЯ:")
    print(f"  Количество эпох: 50")
    print(f"  Learning rate: 0.001")
    print(f"  Momentum: 0.9")
    print(f"  Patience: 5")

    adaline = Adaline(input_size=2, learning_rate=0.001, momentum=0.9)
    adaline.fit(X, y, epochs=50, patience=5)

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

    print(f"\nУравнение разделяющей линии:")
    print(f"  {adaline.weights[0]:.4f}*x1 + {adaline.weights[1]:.4f}*x2 + {adaline.bias:.4f} = 0")
```

```
print(f"\nMSE по эпохам (первые/последние 5):")
for epoch, mse in enumerate(adaline.mse_history[:5], 1):
    print(f"  Эпоха {epoch:2d}: {mse:.6f}")
print("  ...")
for epoch, mse in enumerate(adaline.mse_history[-5:], len(adaline.mse_history)-4):
    print(f"  Эпоха {epoch:2d}: {mse:.6f}")

print(f"\nКлассификация обучающих данных:")
correct_count = 0
for i in range(len(X)):
    prediction = adaline.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, adaline)
```

# Результат тестирования:

РЕЗУЛЬТАТЫ ОБУЧЕНИЯ
Финальные параметры модели:
  Веса: w1 = -0.124716, w2 = 0.041971
  Смещение (bias): 0.223240

Уравнение разделяющей линии:
  -0.1247*x1 + 0.0420*x2 + 0.2232 = 0

MSE по эпохам (первые/последние 5):
  Эпоха  1: 0.275997
  Эпоха  2: 0.211299
  Эпоха  3: 0.174458
  Эпоха  4: 0.166274
  Эпоха  5: 0.151357
  ...
  Эпоха 46: 0.065189
  Эпоха 47: 0.065089
  Эпоха 48: 0.064998
  Эпоха 49: 0.064914
  Эпоха 50: 0.064837

Классификация обучающих данных:
  (2, 6) -> предсказание: 1
  (-2, 6) -> предсказание: 1
  (2, -6) -> предсказание: 0
  (-2, -6) -> предсказание: 1

Точность на обучающей выборке: 50.0% (2/4)

**График с разделяющей поверхностью:**

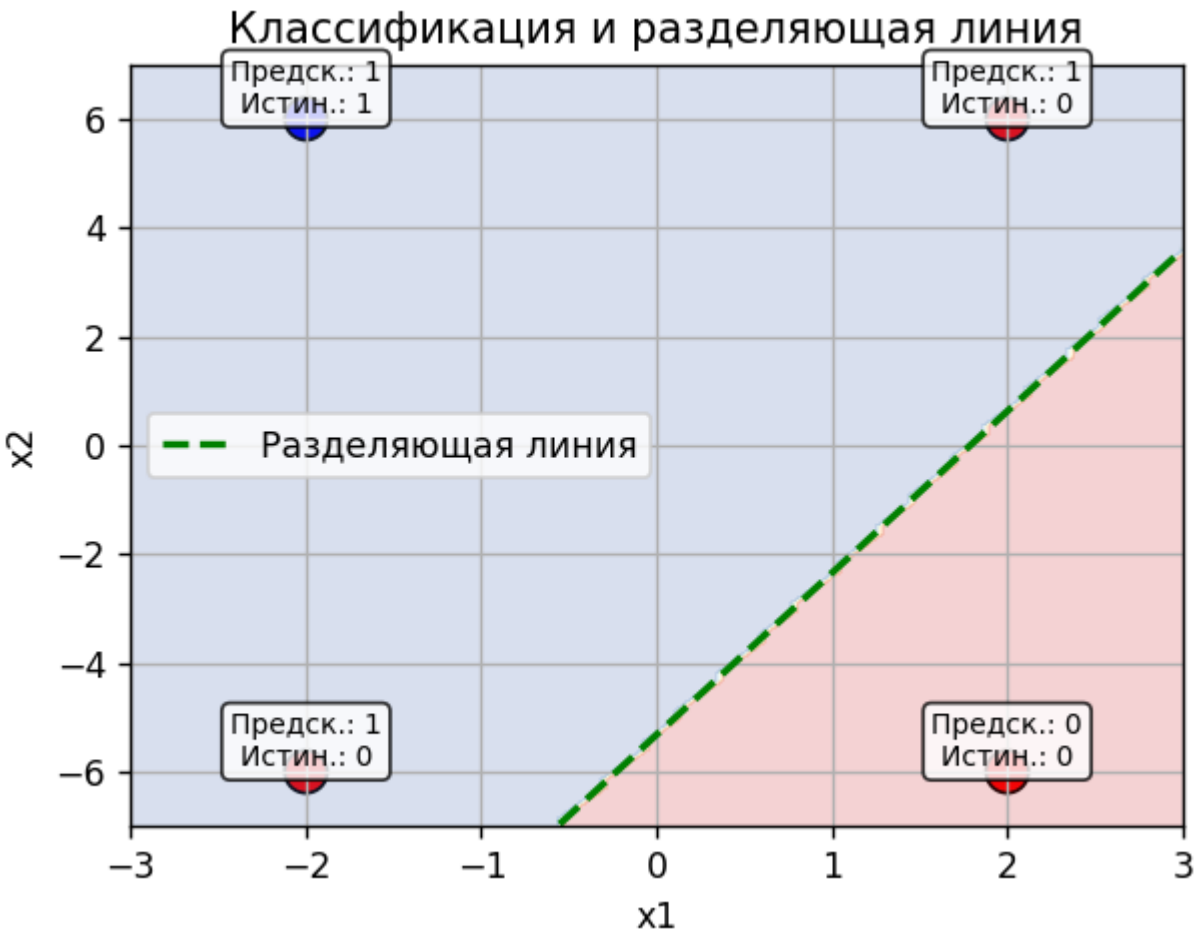
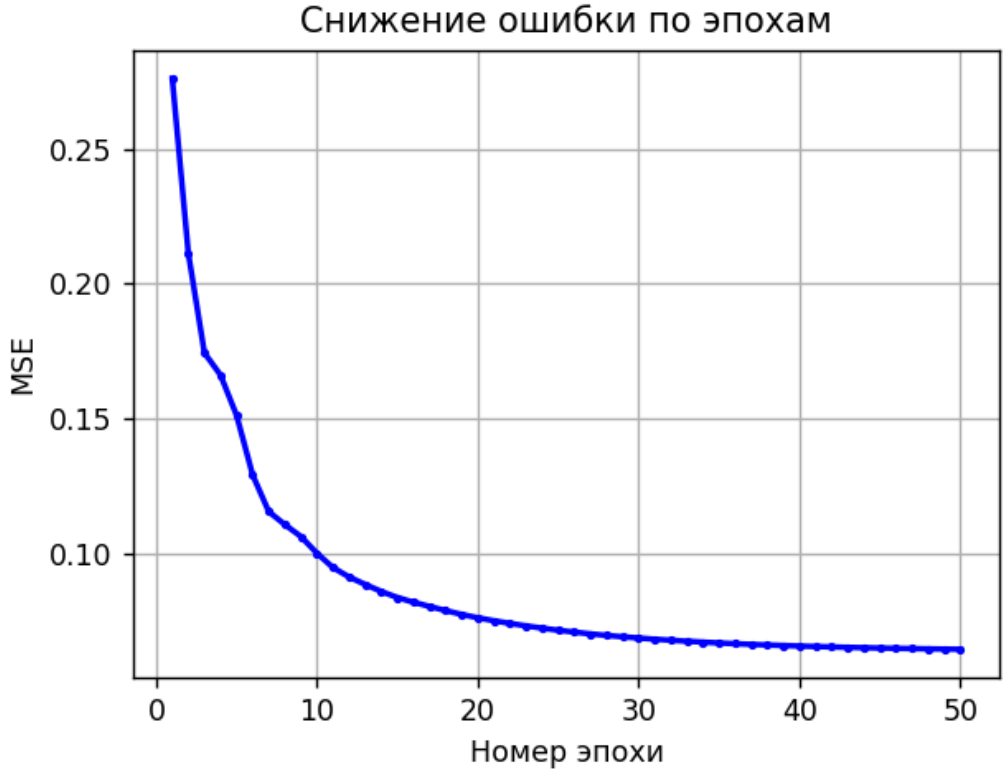



**График изменения ошибки:**



**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).