МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»  
 (СПБГМТУ)

| ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ  КАФЕДРА КИБЕРФИЗИЧЕСКИХ СИСТЕМ |
| --- |

ЛАБОРАТОРНАЯ РАБОТА №1

«АЛГОРИТМ ОБУЧЕНИЯ ПЕРСЕПТРОНА»  
ВАРИАНТ 8

|  | |
| --- | --- |
|  | Выполнил  студент группы 20221 | | |
|  | Лаптев Иван Александрович | | |
|  | Проверила | | |
|  | Кайнова Татьяна Денисовна | | |
| Санкт-Петербург  2024 год | | |

**СОДЕРЖАНИЕ**

[Цели работы………………………………………………………………………. 2](#_gjdgxs)

[Результат работы………………………………………………………………...... 3](#_30j0zll)

[Заключение…………………………………………………](#_17dp8vu)…………………..10

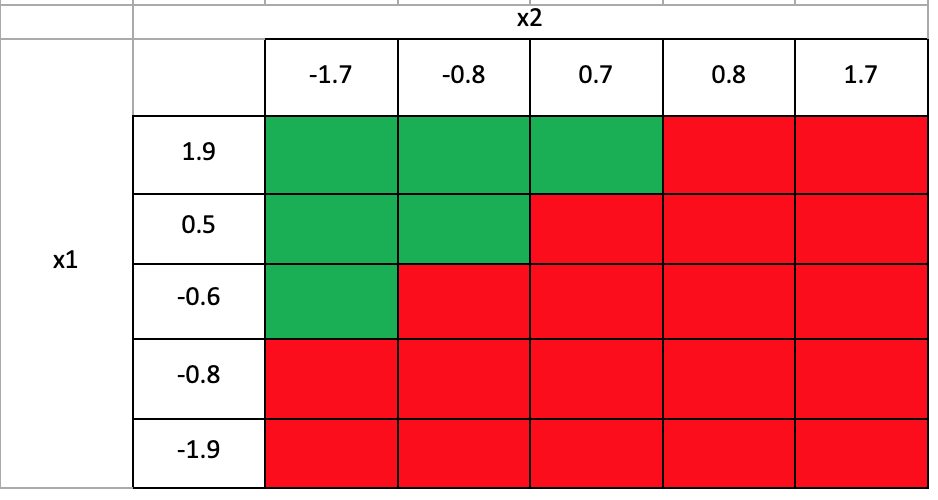
[Листинг кода……………………………………………………………………..](#_26in1rg) 11

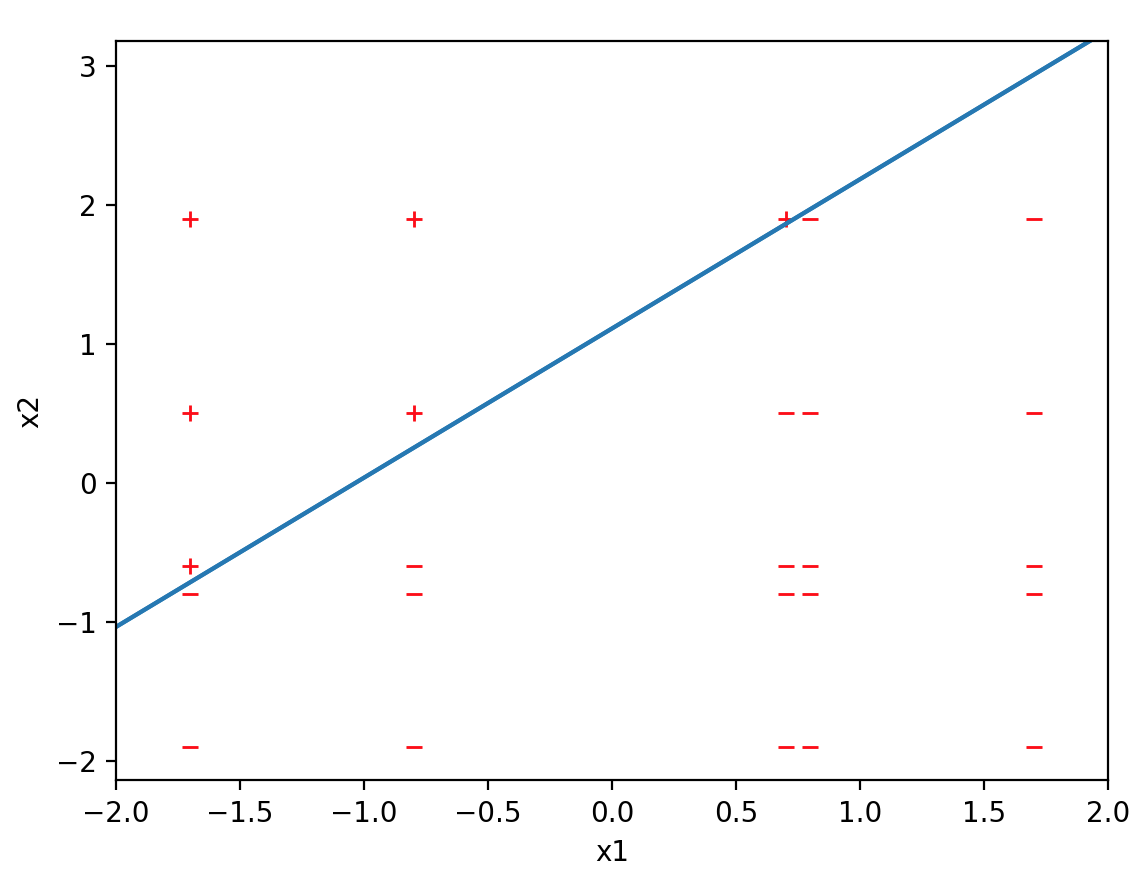
# Цели работы

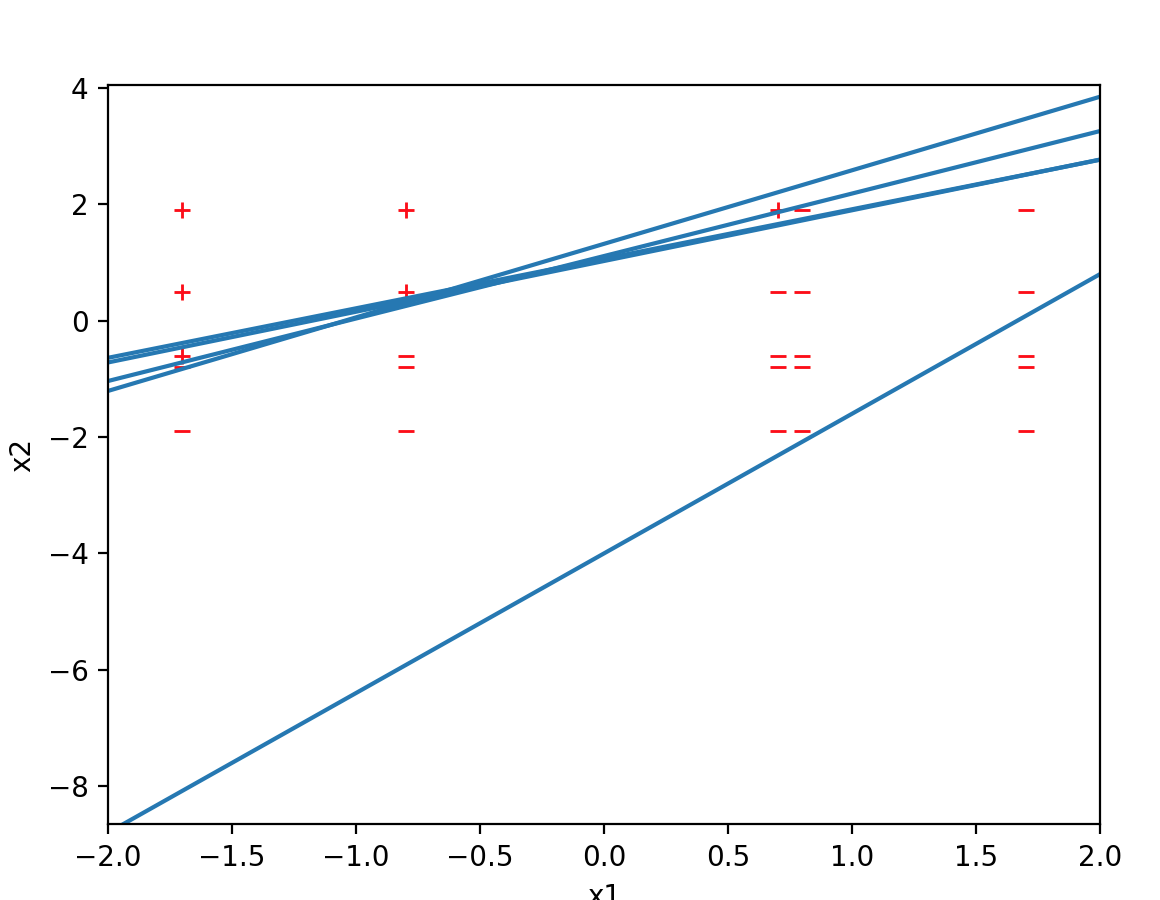
Цель:создание алгоритма обучения персептрона

При работе над данным лабораторным проектом будут затронуты следующие вопросы:

1. Обработка входных данных.
2. Разработка алгоритма обучения персептрона на языке Python.
3. Анализ результатов обучения модели.
4. Создание диаграмм при помощи библиотеки matplotlib.

  
Рисунок 1 - Исходные данные

Рисунок 2 - Процесс обучения

Рисунок 3 - Конечное значение весов после обучения

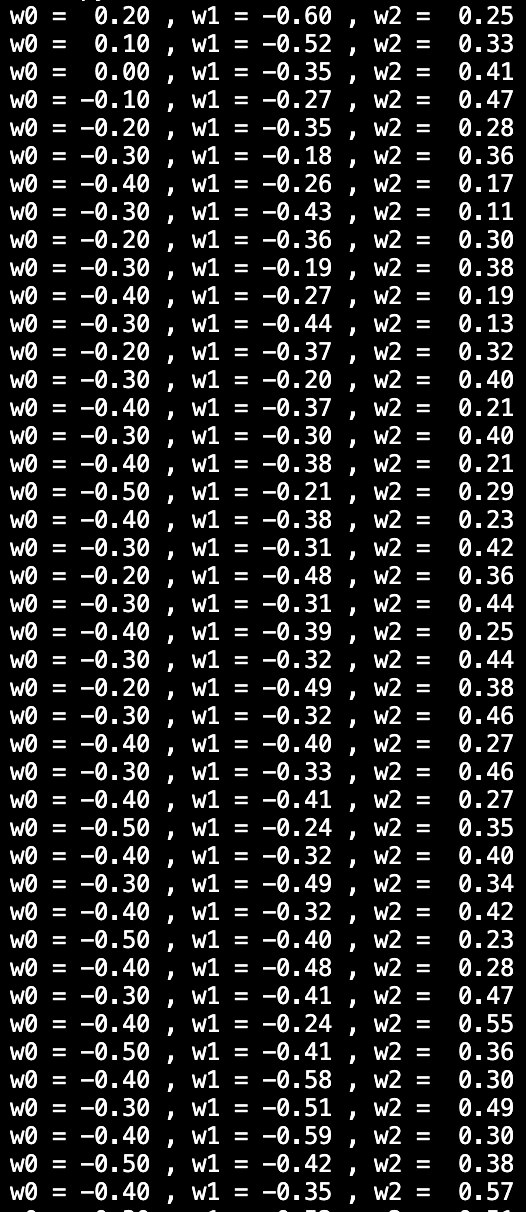


Рисунок 4 - Веса, выводимые в консоль

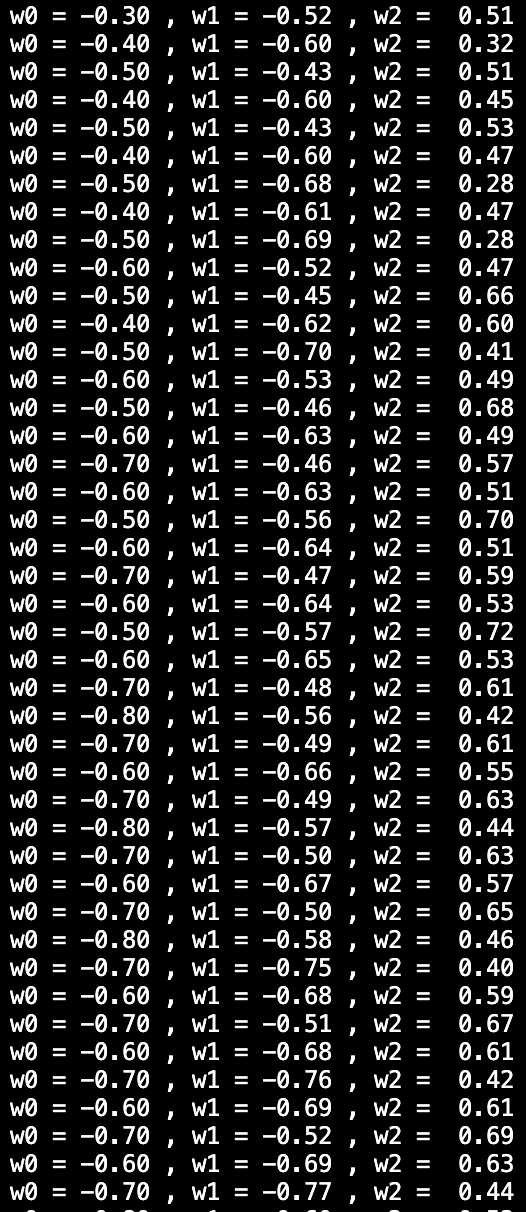


Рисунок 5 - Веса, выводимые в консоль

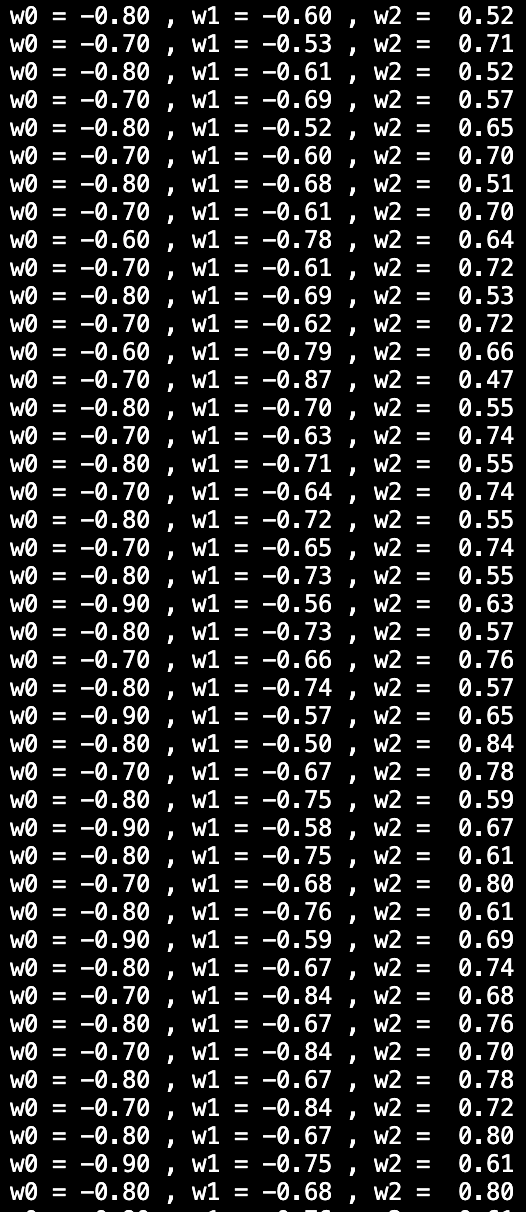
1. 

Рисунок 5 - Веса, выводимые в консоль

# Процесс обучения

Программа обучает персептрон классифицировать данные на два класса. В процессе работы она прогоняет примеры из обучающего набора через сеть, вычисляет предсказания и сравнивает их с истинными значениями. Если сеть ошибается, её веса корректируются, чтобы в будущем она правильно обрабатывала аналогичные примеры. Алгоритм повторяется до тех пор, пока сеть не научится классифицировать все примеры правильно. Во время работы программа выводит текущие веса на каждом шаге, что позволяет отслеживать процесс обучения.

# Результат работы

После завершения обучения программа строит графики, визуализирующие процесс адаптации персептрона и финальное разделение данных. На рисунке 1 показаны исходные данные, разделенные на две группы. Графики с рисунков 2 и 3 отображают изменения границы разделения по мере обучения. Вывод значений весов на каждом этапе представлен в приложениях с рисунков 4, 5 и 6.

# Заключение

Программа успешно обучила персептрон разделять данные на два класса. Построенные графики и консольный вывод подтвердили успешность каждого шага обучения. Итог показывает, что базовый алгоритм персептрона подходит для решения задач бинарной классификации.

# Приложение

# Листинг кода

Листинг 1 - алгоритм персептрона

import random # Импортируем модуль для генерации случайных чисел

import matplotlib.pyplot as plt # Импортируем библиотеку для построения графиков

weights = [] # Список для хранения значений весов после каждой итерации обучения

def show\_learning(w):

"""

Функция для вывода текущих значений весов на экран

"""

weights.append(w) # Добавляем текущие значения весов в список для отслеживания изменений

# Печатаем значения весов в удобном формате (с точностью до 2 знаков после запятой)

print('w0 =', '%5.2f' % w[0], ', w1 =', '%5.2f' % w[1], ', w2 =', '%5.2f' % w[2])

# Инициализация случайного генератора и параметров обучения

random.seed(7) # Устанавливаем начальное значение генератора случайных чисел для воспроизводимости результатов

LEARNING\_RATE = 0.1 # Задаём скорость обучения, которая контролирует величину корректировок весов на каждой итерации

# Обучающие данные (входные данные) - список примеров для тренировки сети

x\_train = [

(1.0, -1.7, 1.9), (1.0, -0.8, 1.9), (1.0, 0.7, 1.9), (1.0, 0.8, 1.9), (1.0, 1.7, 1.9),

(1.0, -1.7, 0.5), (1.0, -0.8, 0.5), (1.0, 0.7, 0.5), (1.0, 0.8, 0.5), (1.0, 1.7, 0.5),

(1.0, -1.7, -0.6), (1.0, -0.8, -0.6), (1.0, 0.7, -0.6), (1.0, 0.8, -0.6), (1.0, 1.7, -0.6),

(1.0, -1.7, -0.8), (1.0, -0.8, -0.8), (1.0, 0.7, -0.8), (1.0, 0.8, -0.8), (1.0, 1.7, -0.8),

(1.0, -1.7, -1.9), (1.0, -0.8, -1.9), (1.0, 0.7, -1.9), (1.0, 0.8, -1.9), (1.0, 1.7, -1.9)

] # Входные данные (x1, x2 и дополнительный компонент, равный 1)

# Истинные выходные значения для каждого обучающего примера (y\_train)

y\_train = [

1.0, 1.0, 1.0, -1.0, -1.0,

1.0, 1.0, -1.0, -1.0, -1.0,

1.0, -1.0, -1.0, -1.0, -1.0,

-1.0, -1.0, -1.0, -1.0, -1.0,

-1.0, -1.0, -1.0, -1.0, -1.0

] # Истинные значения для каждого входного примера

# Список индексов для случайного порядка обучения

index\_list = [i for i in range(len(x\_train))] # Список чисел от 0 до длины x\_train, для случайного перемешивания

# Инициализация начальных весов случайными значениями

w = [0.2, -0.6, 0.25] # Начальные веса: w0, w1, w2

show\_learning(w) # Печатаем начальные значения весов

# Функция для вычисления выхода нейрона (персептрона) для данного входа

def compute\_output(w, x):

z = 0.0 # Инициализируем переменную для вычисления взвешенной суммы

for i in range(len(w)): # Проходим по всем весам

z += x[i] \* w[i] # Вычисляем взвешенную сумму входов

if z < 0: # Применяем пороговую функцию активации

return -1 # Если сумма меньше 0, то выход -1

else:

return 1 # Если сумма больше или равна 0, то выход 1

# Цикл обучения персептрона

all\_correct = False # Флаг, который будет проверять, правильно ли классифицированы все примеры

while not all\_correct:

all\_correct = True # Сначала предполагаем, что все правильно

random.shuffle(index\_list) # Перемешиваем порядок примеров, чтобы обучение не было детерминированным

for i in index\_list:

x = x\_train[i] # Получаем текущий обучающий пример

y = y\_train[i] # Получаем истинный выход для текущего примера

p\_out = compute\_output(w, x) # Вычисляем предсказанный выход нейрона

if y != p\_out: # Если предсказание неверное, обновляем веса

for j in range(0, len(w)): # Пробегаем по всем весам

w[j] += (y \* LEARNING\_RATE \* x[j]) # Обновляем вес по правилу: w = w + (learning\_rate \* x \* y)

all\_correct = False # Устанавливаем флаг, что не все правильно

show\_learning(w) # Выводим обновленные веса на каждом шаге

# График 1: Отображаем обучающие данные

fig, ax = plt.subplots() # Создаем новую фигуру для графика

plt.xlim([-2.0, 2.0]) # Устанавливаем пределы по оси X

# Отображаем данные в зависимости от их метки

for i in range(len(x\_train)):

if y\_train[i] == -1:

ax.plot(x\_train[i][1], x\_train[i][2], 'r\_') # Для метки -1 рисуем красные подчеркивания

else:

ax.plot(x\_train[i][1], x\_train[i][2], 'r+') # Для метки 1 рисуем красные крестики

plt.xlabel("x1") # Подпись оси X

plt.ylabel("x2") # Подпись оси Y

# Строим разделяющую гиперплоскость для каждого шага обучения

x1 = [x\_train[0][1], x\_train[-1][1]] # Для построения линии разделения

for i in weights:

x2 = (-(i[1]/i[2]) \* x1[0] - (i[0]/i[2]), -(i[1]/i[2]) \* x1[1] - (i[0]/i[2])) # Вычисляем соответствующие значения y

ax.plot(x1, x2) # Строим линию разделения на графике

# График 2: Тот же график, но только для последних весов

fig, ax = plt.subplots() # Создаем новый график

plt.xlim([-2.0, 2.0]) # Устанавливаем пределы оси X

# Повторно отображаем обучающие данные

for i in range(len(x\_train)):

if y\_train[i] == -1:

ax.plot(x\_train[i][1], x\_train[i][2], 'r\_') # Отображаем красные подчеркивания для класса -1

else:

ax.plot(x\_train[i][1], x\_train[i][2], 'r+') # Отображаем красные крестики для класса 1

plt.xlabel("x1") # Подпись оси X

plt.ylabel("x2") # Подпись оси Y

# Построение линии разделения на основе последних обученных весов

x1\_n = [x\_train[0][1], x\_train[-1][1]] # Для построения линии разделения

for j in x1\_n:

# Вычисляем соответствующие y для последнего шага

x2\_n = (-(weights[-1][1]/weights[-1][2])\* x1[0] - (weights[-1][0]/weights[-1][2]), -(weights[-1][1]/weights[-1][2])\*x1[1] - (weights[-1][0]/weights[-1][2]))

ax.plot(x1\_n, x2\_n) # Строим линию разделения

plt.show() # Показываем графики