|  |  |
| --- | --- |
| Picture 1 | **МИНОБРНАУКИ РОССИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  «Санкт-Петербургский государственный морской технический университет» (СПбГМТУ) |

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Факультет цифровых промышленных технологий

Направление подготовки 09.03.01

" **Интеллектуальные технологии киберфизических систем**"

Лабораторная работа №1

Вариант 14

Студент 2 курса группы 20221

Очного отделения

Руденко Вячеслав Сергеевич

Проверил:

Преподаватель CПбГМТУ

Кайнова Татьяна Денисовна

Санкт-Петербург

2024

Оглавление

[1. ОБЩАЯ ЧАСТЬ 3](#_Toc181008747)

[1.2 Цель работы 3](#_Toc181008748)

[1.2 Формулировка задачи 3](#_Toc181008749)

[1.3 Исходные данные 3](#_Toc181008750)

[2. ХОД РАБОТЫ 4](#_Toc181008751)

[ЗАКЛЮЧЕНИЕ 10](#_Toc181008752)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 11](#_Toc181008753)

[ПРИЛОЖЕНИЕ А 12](#_Toc181008754)

# 1. ОБЩАЯ ЧАСТЬ

## 1.2 Цель работы

Написать правильно работающую программу, печатающую все значения весов, получаемые в процесс обучения и две диаграммы.

## 1.2 Формулировка задачи

Доработать программу из примера таким образом, чтобы персептрон обучался реализации зависимости y=f(x1,x2), соответствующей варианту.

## 1.3 Исходные данные

Исходные данные 14 варианта представлены на рисунке 1.

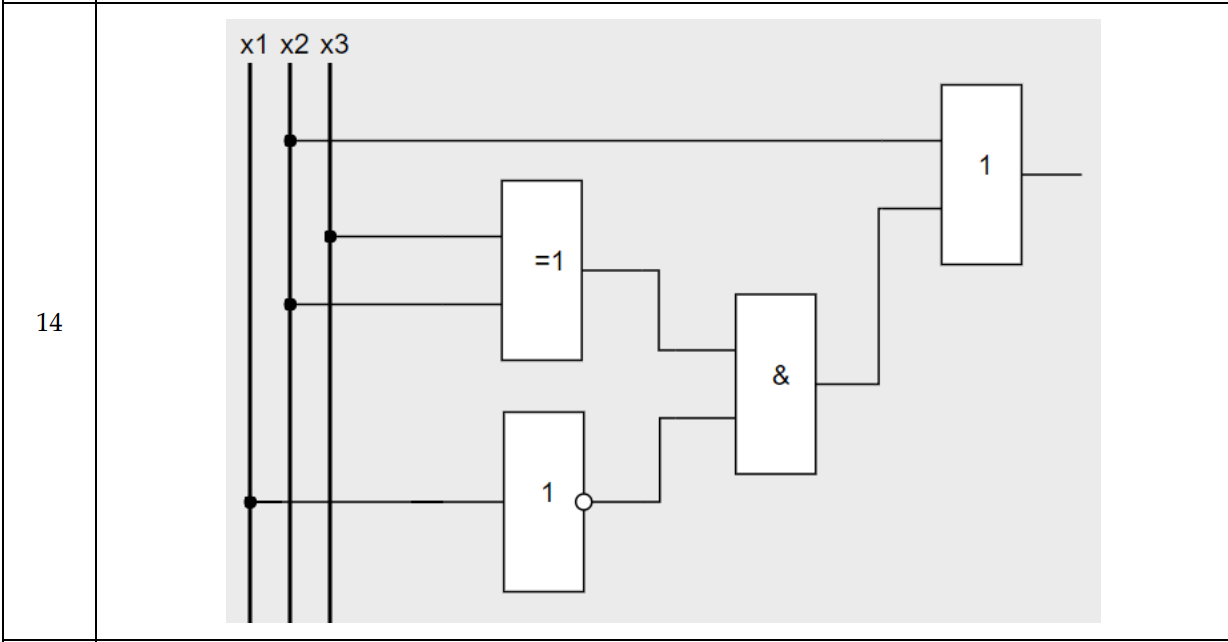


Рисунок 1 – Исходные данные

# 2. ХОД РАБОТЫ

В данном разделе содержится подробная информация о процессе выполнения работы, импортирование библиотек и так далее.

Импортируем необходимые библиотеки

1. NumPy.

NumPy — это библиотека для работы с многомерными массивами и матрицами, а также для выполнения различных математических вычислений.

2. random.

random — это встроенный модуль Python, который предоставляет функции для генерации случайных чисел и выбора случайных элементов.

3. Matplotlib.

Matplotlib — это библиотека для создания статичных, анимационных и интерактивных визуализаций на Python. pyplot — это модуль внутри Matplotlib, который предоставляет удобный интерфейс для построения графиков

На рисунке 2 представлена функция show\_learning, входы и инициализации

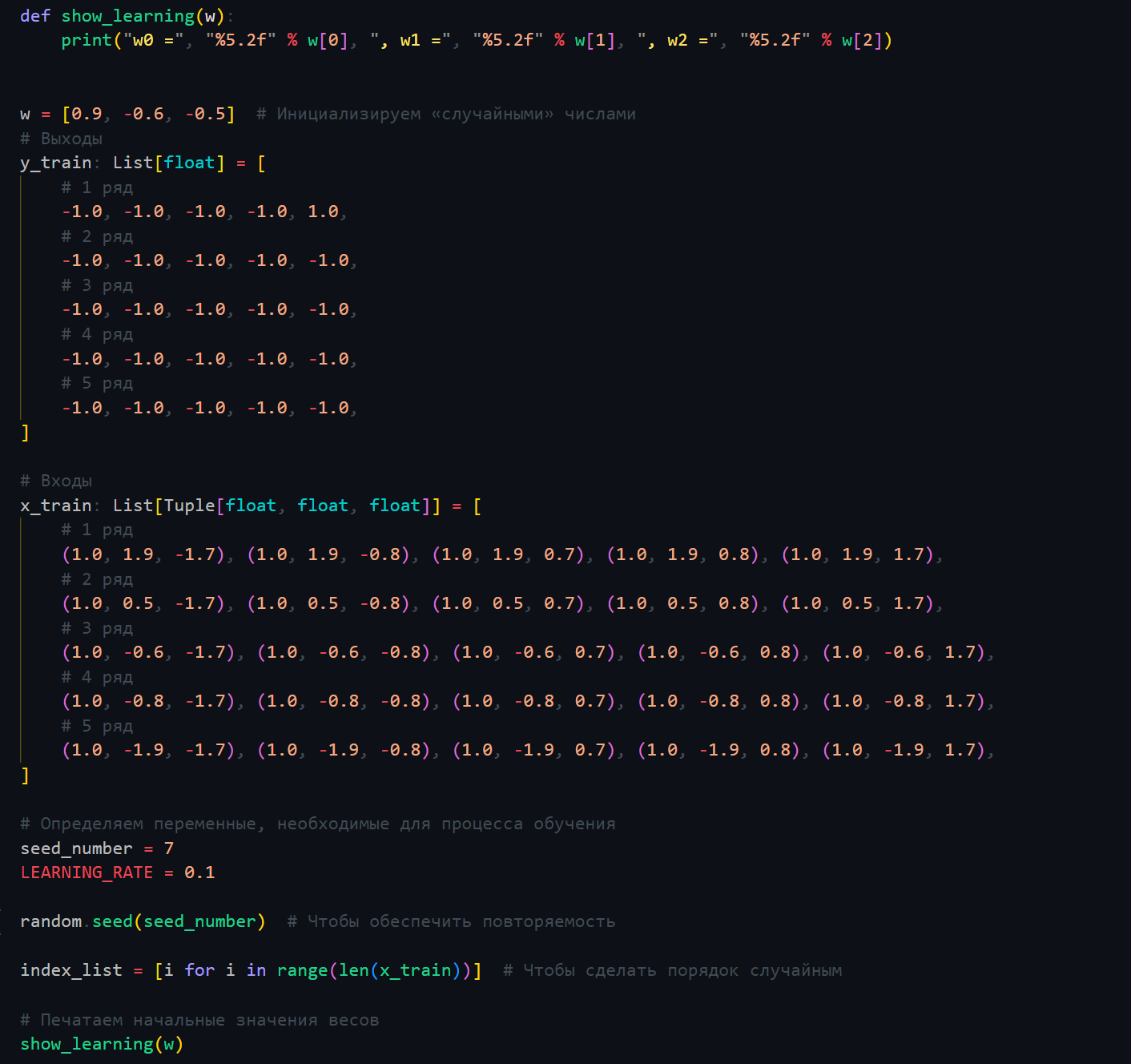


Рисунок 2 - Листинг функции show\_learning, входов и инициализации

Функция show\_learning(w) предназначена для отображения значений весов в формате, удобном для понимания результатов обучения модели.

Функция compute\_output показана на рисунке 4. Она предназначена для вычисления выходных данных персептрона на основе входных данных и весов. Она реализует простую линейную модель, которая имеет два возможных выхода: -1 или 1.

На рисунке 3 показан листинг цикла обучения персептрона.

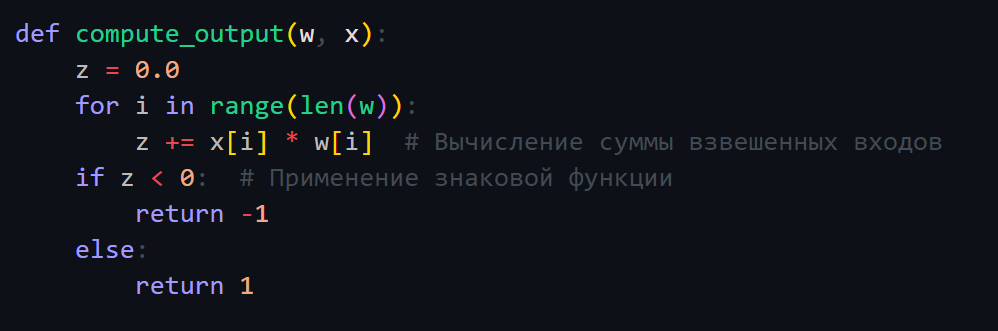


Рисунок 3 - Листинг compute\_output

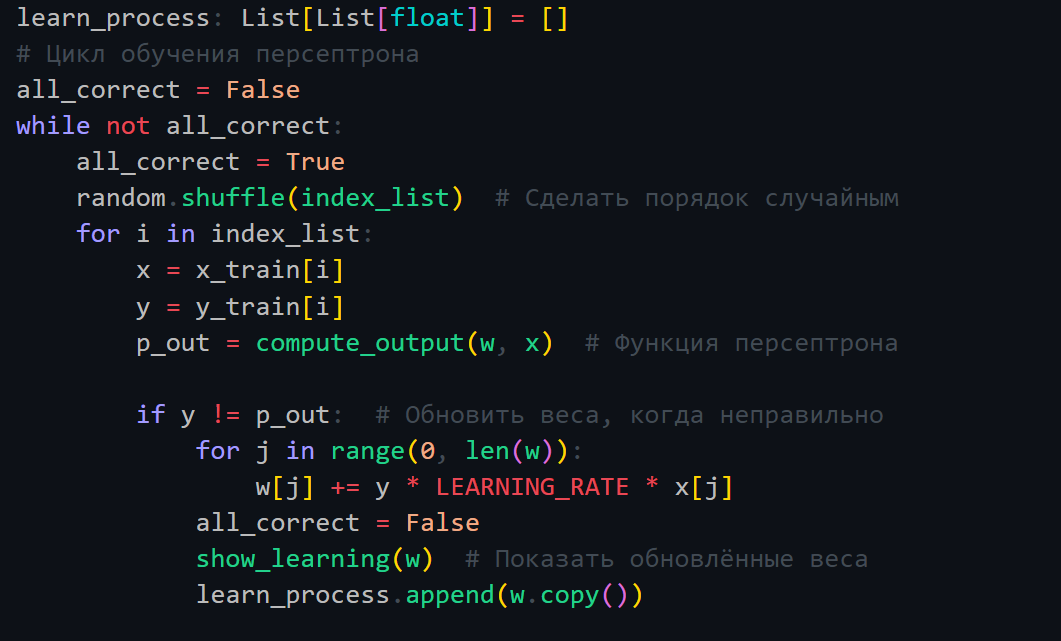


Рисунок 4 - Листинг цикла обучения персептрона

Описание кода: это вложенный цикл, в котором внутренний цикл проходит через все четыре обучающих примера в случайном порядке. Для каждого примера он вычисляет выходные данные, корректирует и печатает веса, если выходные данные неверны. Внешний цикл проверяет, выдал ли персептрон правильный вывод для всех четырех примеров, и если да, то завершает программу.

Список learn\_process необходим для сохранения весов на каждом шаге для визуализации обучения персептрона.

На рисунке 4 показан фрагмент кода, отвечающий за визуализацию изменения разделяющих прямых.

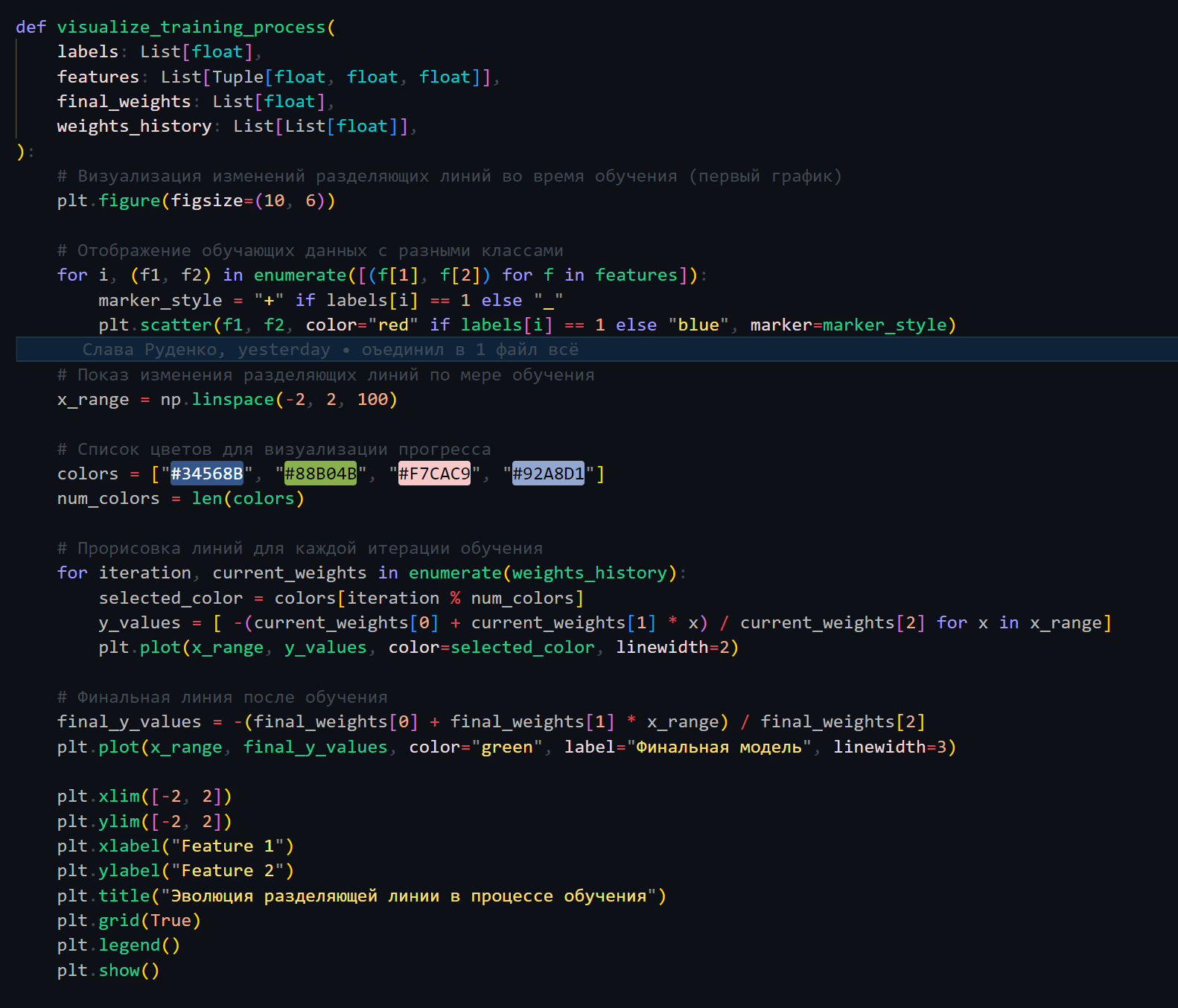


Рисунок 5 – Листинг реализации визуализации разделяющих прямых

Первый график в функции визуализации (график на рис. 7) отображает процесс изменения разделяющей линии во время обучения персептрона. На графике отображаются все обучающие точки, каждая из которых соответствует одному из двух классов. Точки одного класса помечены красными маркерами `+`, а точки другого — синими маркерами `\_`. Эти точки представляют собой двумерные данные, которые персептрон пытается разделить с помощью линейной модели.

Для каждого шага обучения визуализируются разделяющие линии, которые соответствуют весам, вычисленным на разных итерациях. Линии постепенно изменяются по мере обновления весов, что позволяет наглядно увидеть, как персептрон учится лучше классифицировать данные. Линии рисуются разными цветами из заданного набора, что помогает выделить каждую итерацию обучения и проследить за их прогрессом.

Для расчета линий используется уравнение прямой, которое строится на основе текущих значений весов. Сначала задаются значения для оси X, а затем для каждой итерации вычисляются соответствующие значения на оси Y. Эти значения формируют разделяющую линию, которая обновляется в процессе обучения до тех пор, пока персептрон не обучится корректно классифицировать все точки.

На рисунке 5 показан фрагмент кода, отвечающий за визуализацию разделяющей прямой.

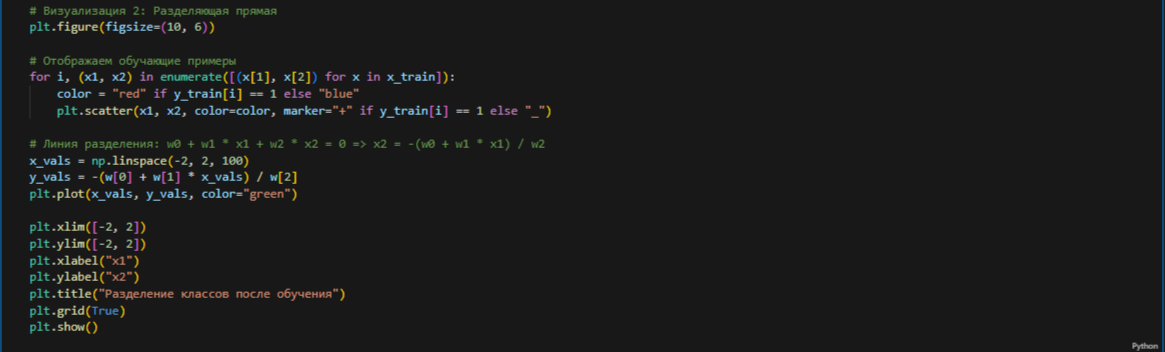


Рисунок 6 – Листинг реализации визуализации разделяющей прямой

Описание кода: второй график, показанный на рисунке 8, визуализирует финальную разделяющую линию, которую персептрон построил после завершения обучения. Как и в первом графике, точки данных отображаются с разными цветами и маркерами, где красные «+» представляют один класс, а синие «\_» — другой. Этот график отличается тем, что показывает только одну разделяющую линию — итоговую, которую персептрон вычислил на основе финальных весов.

Финальная разделяющая линия вычисляется на основе итоговых значений весов модели, которые были получены после всех итераций обучения. Используя эти веса, строится линейное уравнение, которое задает границу между двумя классами данных. Это позволяет визуально оценить, насколько хорошо обученная модель способна разделять точки разных классов.

Этот график показывает, что обучение завершено, и как итоговая модель классифицирует входные данные. Финальная линия, отображенная зелёным цветом, демонстрирует итоговое решение персептрона для задачи классификации, при этом она фиксирована и больше не изменяется.

На рисунке 7 показан график обучения нейрона.

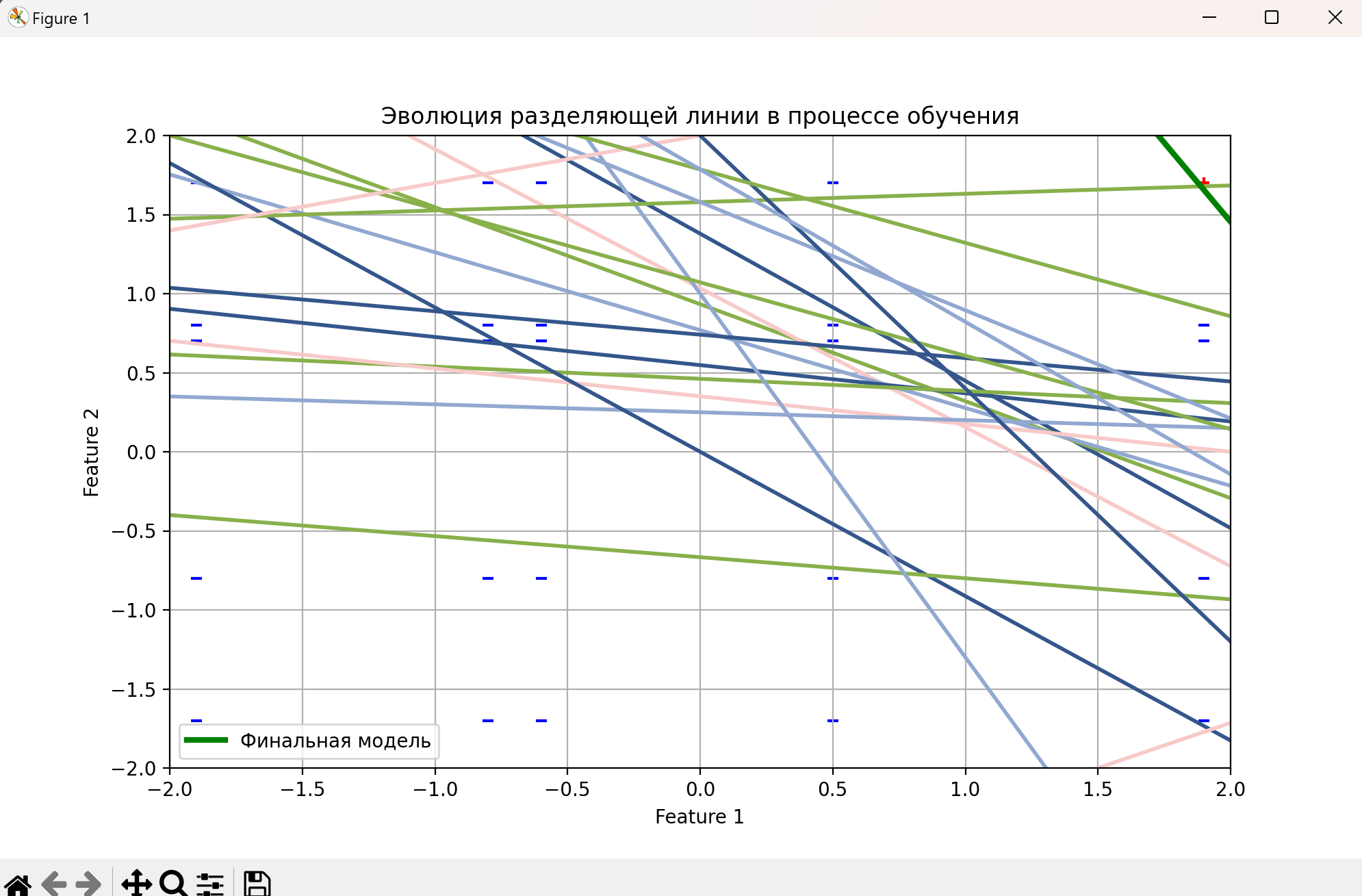


Рисунок 7 - График обучения нейрона

На рисунке 8 представлена график, отражающий разделение классов после обучения.

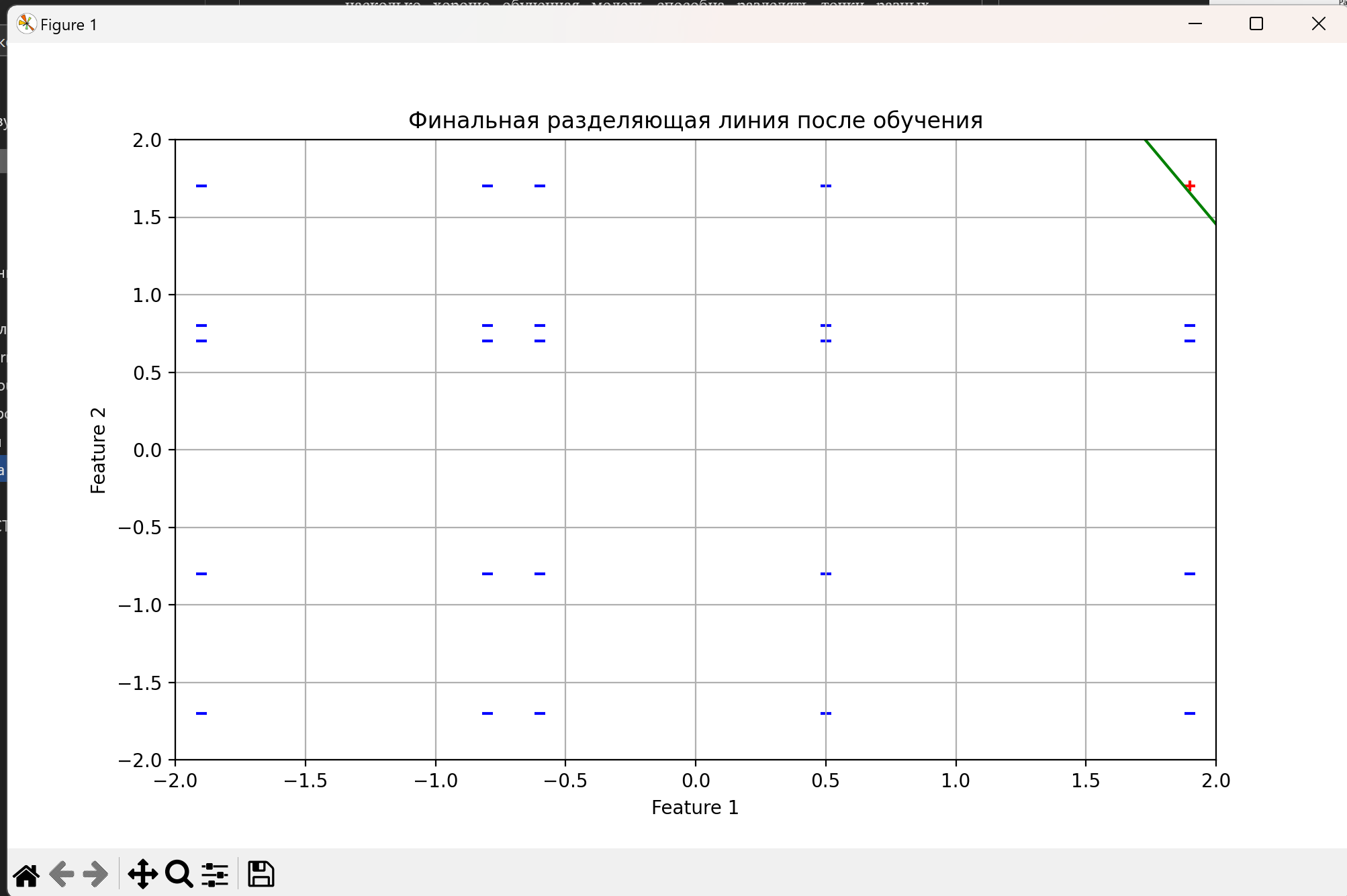


Рисунок 8 - График, отражающий разделение классов после обучения

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы была доработана программа из примера таким образом, чтобы персептрон обучался реализации зависимости y=f(x1,x2). Была создана правильно работающая программа, печатающая все значения весов, получаемые в процессе обучения и две необходимые диаграммы.

Таким образом, цель работы достигнута, и доработанный код может быть использован для визуализации работы персептрона.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. ГОСТ Р 7.0.97-2016. Национальный стандарт Российской Федерации. Система стандартов по информации, библиотечному и издательскому делу. Организационно-распорядительная документация. Требования к оформлению документов: утвержден и введен в действие Приказом Федерального агентства по техническому регулированию и метрологии от 14.05.2018 N 244-ст: Дата введения 2018-07-01. - URL: <https://docs.cntd.ru/document/1200159234> (дата обращения: 02.10.2024). - Текст: электронный.

# ПРИЛОЖЕНИЕ А

Листинг кода

Листинг 1 - perseptron.py

import random

import matplotlib.pyplot as plt

import numpy as np

from typing import Tuple, List

def visualize\_training\_process(

    labels: List[float],

    features: List[Tuple[float, float, float]],

    final\_weights: List[float],

    weights\_history: List[List[float]],

):

    # Визуализация изменений разделяющих линий во время обучения (первый график)

    plt.figure(figsize=(10, 6))

    # Отображение обучающих данных с разными классами

    for i, (f1, f2) in enumerate([(f[1], f[2]) for f in features]):

        marker\_style = "+" if labels[i] == 1 else "\_"

        plt.scatter(f1, f2, color="red" if labels[i] == 1 else "blue", marker=marker\_style)

    # Показ изменения разделяющих линий по мере обучения

    x\_range = np.linspace(-2, 2, 100)

    # Список цветов для визуализации прогресса

    colors = ["#34568B", "#88B04B", "#F7CAC9", "#92A8D1"]

    num\_colors = len(colors)

    # Прорисовка линий для каждой итерации обучения

    for iteration, current\_weights in enumerate(weights\_history):

        selected\_color = colors[iteration % num\_colors]

        y\_values = [ -(current\_weights[0] + current\_weights[1] \* x) / current\_weights[2] for x in x\_range]

        plt.plot(x\_range, y\_values, color=selected\_color, linewidth=2)

    # Финальная линия после обучения

    final\_y\_values = -(final\_weights[0] + final\_weights[1] \* x\_range) / final\_weights[2]

    plt.plot(x\_range, final\_y\_values, color="green", label="Финальная модель", linewidth=3)

    plt.xlim([-2, 2])

    plt.ylim([-2, 2])

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.title("Эволюция разделяющей линии в процессе обучения")

    plt.grid(True)

    plt.legend()

    plt.show()

    # Визуализация финальной разделяющей линии после обучения (второй график)

    plt.figure(figsize=(10, 6))

    # Отображение точек данных с соответствующими метками

    for i, (f1, f2) in enumerate([(f[1], f[2]) for f in features]):

        marker\_style = "+" if labels[i] == 1 else "\_"

        plt.scatter(f1, f2, color="red" if labels[i] == 1 else "blue", marker=marker\_style)

    # Финальная линия разделения: w0 + w1 \* x1 + w2 \* x2 = 0

    final\_y\_values = -(final\_weights[0] + final\_weights[1] \* x\_range) / final\_weights[2]

    plt.plot(x\_range, final\_y\_values, color="green")

    plt.xlim([-2, 2])

    plt.ylim([-2, 2])

    plt.xlabel("Feature 1")

    plt.ylabel("Feature 2")

    plt.title("Финальная разделяющая линия после обучения")

    plt.grid(True)

    plt.show()

def show\_learning(w):

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

w = [0.9, -0.6, -0.5]  # Инициализируем «случайными» числами

# Выходы

y\_train: List[float] = [

    # 1 ряд

    -1.0, -1.0, -1.0, -1.0, 1.0,

    # 2 ряд

    -1.0, -1.0, -1.0, -1.0, -1.0,

    # 3 ряд

    -1.0, -1.0, -1.0, -1.0, -1.0,

    # 4 ряд

    -1.0, -1.0, -1.0, -1.0, -1.0,

    # 5 ряд

    -1.0, -1.0, -1.0, -1.0, -1.0,

]

# Входы

x\_train: List[Tuple[float, float, float]] = [

    # 1 ряд

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

    # 2 ряд

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

    # 3 ряд

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

    # 4 ряд

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

    # 5 ряд

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

]

# Определяем переменные, необходимые для процесса обучения

seed\_number = 7

LEARNING\_RATE = 0.1

random.seed(seed\_number)  # Чтобы обеспечить повторяемость

index\_list = [i for i in range(len(x\_train))]  # Чтобы сделать порядок случайным

# Печатаем начальные значения весов

show\_learning(w)

def compute\_output(w, x):

    z = 0.0

    for i in range(len(w)):

        z += x[i] \* w[i]  # Вычисление суммы взвешенных входов

    if z < 0:  # Применение знаковой функции

        return -1

    else:

        return 1

learn\_process: List[List[float]] = []

# Цикл обучения персептрона

all\_correct = False

while not all\_correct:

    all\_correct = True

    random.shuffle(index\_list)  # Сделать порядок случайным

    for i in index\_list:

        x = x\_train[i]

        y = y\_train[i]

        p\_out = compute\_output(w, x)  # Функция персептрона

        if y != p\_out:  # Обновить веса, когда неправильно

            for j in range(0, len(w)):

                w[j] += y \* LEARNING\_RATE \* x[j]

            all\_correct = False

            show\_learning(w)  # Показать обновлённые веса

            learn\_process.append(w.copy())

visualize\_training\_process(y\_train, x\_train, w, learn\_process)