VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Skaitmeninio intelekto dirbtinio neurono kūrimo ataskaita

Darbo ataskaita

Atliko: 3 kurso 3 grupės studentas
Hubertas Vindžigalskis

Vilnius – 2025

Turinys

1. Užduoties tikslas

Užduotyje reikia sukurti dirbtinį neuroną, kuris remdamasis įėjimo parametrais, svoriais ir poslinkiu gali prognozuoti taškų klases pagal jų koordinates. Tikslas yra sugeneruoti įvesties pradines reikšmes, joms rasti svorius, su kuriais galime prognozuoti, kuriai klasei (0 ar 1) priklauso taškas ir rasti šių klasių perskyros tiesę bei jai statmeną vektorių.

2. Sugeneruoti duomenys

X	Y	Klasė
2.882026	2.072022	0.0
2.200079	2.727137	0.0
2.489369	2.380519	0.0
3.120447	2.060838	0.0
2.933779	2.221932	0.0
1.511361	2.166837	0.0
2.475044	2.747040	0.0
1.924321	1.897421	0.0
1.948391	2.156534	0.0
2.205299	1.572952	0.0
-3.276495	-1.922526	1.0
-1.673191	-1.810919	1.0
-1.567782	-2.443893	1.0
-2.371083	-2.990398	1.0
-0.865123	-2.173956	1.0
-2.727183	-1.921826	1.0
-1.977121	-1.384855	1.0
-2.093592	-1.398810	1.0
-1.233610	-2.193663	1.0
-1.265321	-2.151151	1.0

1 lentelė. Sugeneruoti duomenys

3. Programos kodas

https://github.com/HubertasVin/digital-intelligence

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

# Parametrai
center_0 = (2, 2)
center_1 = (-2, -2)
num_points = 10



# 1. Duomenu generavimas
def generate_data(center_0, center_1, num_points):
    np.random.seed(0)
    # Sugeneruojame duomenis abejoms klasems aplinkui du
       centrus: (-2, -2) ir (2, 2)
    class_0_x = np.random.normal(loc=center_0[0], scale=0.5,
       size=num_points)
    class_0_y = np.random.normal(loc=center_0[1], scale=0.5,
       size=num_points)
    class_1_x = np.random.normal(loc=center_1[0], scale=0.5,
       size=num_points)
    class_1_y = np.random.normal(loc=center_1[1], scale=0.5,
       size=num_points)

    # Sujungiame sugeneruotus duomenis i patogesni DataFrame
       tipa
    data = pd.DataFrame({
        'X': np.concatenate((class_0_x, class_1_x)),
        'Y': np.concatenate((class_0_y, class_1_y)),
        'Class': np.concatenate((np.zeros(num_points), np.
           ones(num_points)))
    })
    return data



data = generate_data(center_0, center_1, num_points)
print("Pradiniai duomenys su klasemis:")
print(data)
```

```
# 2. Dirbtinis neuronas
# Aktyvacijos funkcijos
def step_function(a):
    return 1 if a >= 0 else 0


def sigmoid_function(a):
    return 1 / (1 + np.exp(-a))


class Perceptron:
    def __init__(self, w1, w2, b, activation_function):
        self.w1 = w1
        self.w2 = w2
        self.bias = b
        self.activation_function = activation_function

    def __predict_to_class(self, a):
        prediction = self.activation_function(a)
        # Jeigu rezultatas yra float tipo (panaudota sigmoid
           funkcija), apvaliname skaiciu
        if (isinstance(prediction, float)):
            return round(prediction)
        else:
            return prediction

    # Gauti rezultata rementis x1 ir x2 taskais, ju svoriais
       ir poslinkiu.
    # Pritaikoma paskirta aktyvacijos funkcija
    def predict(self, x1, x2):
        a = self.w1 * x1 + self.w2 * x2 + self.bias
        return self.__predict_to_class(a)


# 3. Rasti tinkamus svoriu ir bias rinkinius (be mokymo; step
    funkcija)
def find_weight_sets(data, activation_function, num_sets=3,
   max_iter=100000):
```

```
    found_sets = []
    iterations = 0

    # Ieskome svoriu, kol nesurasime triju rinkiniu
    while len(found_sets) < num_sets and iterations <
        max_iter:
        w1 = np.random.uniform(low=-10, high=10)
        w2 = np.random.uniform(low=-10, high=10)
        b = np.random.uniform(low=-10, high=10)
        neuron = Perceptron(w1, w2, b, activation_function)

        # Patikriname, ar visi duomenu taskai teisingai
            klasifikuojami
        all_correct = True
        for _, row in data.iterrows():
            # Tikriname, ar su sugeneruotais svoriais
                tinkamai prognozuojami visi duomenys
            if neuron.predict(row['X'], row['Y']) != row['
                Class']:
                all_correct = False
                break

        # Jeigu visu tasku aktyvacijos funkcijos rezultatai
            atitinka ju klases,
        # pridedame rinkini prie rezultatu
        if all_correct:
            found_sets.append((w1, w2, b))
        iterations += 1

    return found_sets


weight_sets = find_weight_sets(data, step_function)
print("\nRasti tinkami svoriu (w1, w2) ir bias (b) rinkiniai
    (step funkcija):")
for idx, (w1, w2, b) in enumerate(weight_sets):
    print(f"Rinkinys {idx + 1}: w1 = {w1:.4f}, w2 = {w2:.4f},
        bias = {b:.4f}")
```

```
# 4. Rasti tinkamus svoriu ir bias rinkinius (be mokymo;
   sigmoid funkcija)
weight_sets2 = find_weight_sets(data, sigmoid_function)
print("\nRasti tinkami svoriu (w1, w2) ir bias (b) rinkiniai
   (sigmoidine funkcija):")
for idx, (w1, w2, b) in enumerate(weight_sets2, start=1):
    print(f"Rinkinys {idx}: w1 = {w1:.4f}, w2 = {w2:.4f},
       bias = {b:.4f}")


# 5. Klases skiriancioji tiese
def plot_data_and_boundaries(data, weight_sets):
    plt.figure(figsize=(8, 8))
    # Piesiame abieju klasiu taskus atskirai
    class_0 = data[data['Class'] == 0]
    class_1 = data[data['Class'] == 1]
    plt.scatter(class_0['X'], class_0['Y'], color='blue',
       label='Class 0')
    plt.scatter(class_1['X'], class_1['Y'], color='red',
       label='Class 1')

    colors = ['green', 'orange', 'purple']

    # Braizome kiekviena sprendimo riba
    for idx, (w1, w2, b) in enumerate(weight_sets):
        color = colors[idx % len(colors)]
        # Vaizduojame klases skiriancias tieses
        if abs(w2) < 1e-3:
            # Jeigu w2 yra arti 0, breziame vertikalia tiese
            x_val = -b / w1
            plt.axvline(x=x_val, color=color,
                        label=f'Boundary {idx+1}: w1={w1:.2f
                           }, w2={w2:.2f}, b={b:.2f}')
        else:
            # Apskaiciuojame tieses galu kordinates taskuose
               -3 ir 3
            ys = [-(w1 * x + b) / w2 for x in [-3.0, 3.0]]
            plt.plot([-3, 3], ys, color=color,
                     label=f'Boundary {idx+1}: w1={w1:.2f},
                        w2={w2:.2f}, b={b:.2f}')
```

```
            # 6. Vektoriu atvaizdavimas
            norm = np.sqrt(w1**2 + w2**2)
            if norm < 1e-6:
                continue
            # Atrandame taska esanti ant tieses
            x0 = -b * w1 / (norm**2)
            y0 = -b * w2 / (norm**2)

            # Normalizuojame svoriu vektoriu
            dx = (w1 / norm) * 2
            dy = (w2 / norm) * 2
            # Braizome rodykle, kuri prasideda nuo tasko ant
               sprendimo linijos
            plt.arrow(x0, y0, dx, dy, head_width=0.15,
               head_length=0.15, fc=color, ec=color)
            # Pazymime taska, kuriame prasideda rodykle
            plt.plot(x0, y0, 'o', color=color)

        plt.xlabel("X")
        plt.ylabel("Y")
        plt.title("Duomenys ir sprendimo ribos")
        plt.legend()
        # Nustatome ribas, kad grafiko X ir Y krastiniu santykis
           butu 1:1.
        # Reikalinga, kad vektoriai matytusi, jog jie yra
           statmeni ju tiesems
        plt.gca().set_xlim([-4, 4])
        plt.gca().set_ylim([-4, 4])
        plt.show()


    plot_data_and_boundaries(data, weight_sets)
```

3.1. Svorių paieška

Programa atranda tinkamus svorius generuodama 3 atsitiktines reikšmes (w1, w2 ir b) intervale [-10; 10] ir tikrindama jas su kiekvienu pradiniu įvesties tašku pritaikant formulę:

```
        activation_function(w1 * x + w2 * y + b)
```

activation_function – gali būti arba slenkstinė, arba sigmoidinė funkcija. Ji grąžina klasę, jeigu naudojama slenkstinė funkcija, ir skaičių su kableliu, kai naudojama sigmoidinė funkcija.

Gavus šią reikšmę tikriname su „tikra“ klase, kurią priskyrėme dar duomenų generavimo metu. Jeigu naudota sigmoidinė funkcija, prieš tai dar suapvaliname skaičių iki artimiausio sveikojo skaičiaus.

3.2. Gauti svorių rinkiniai

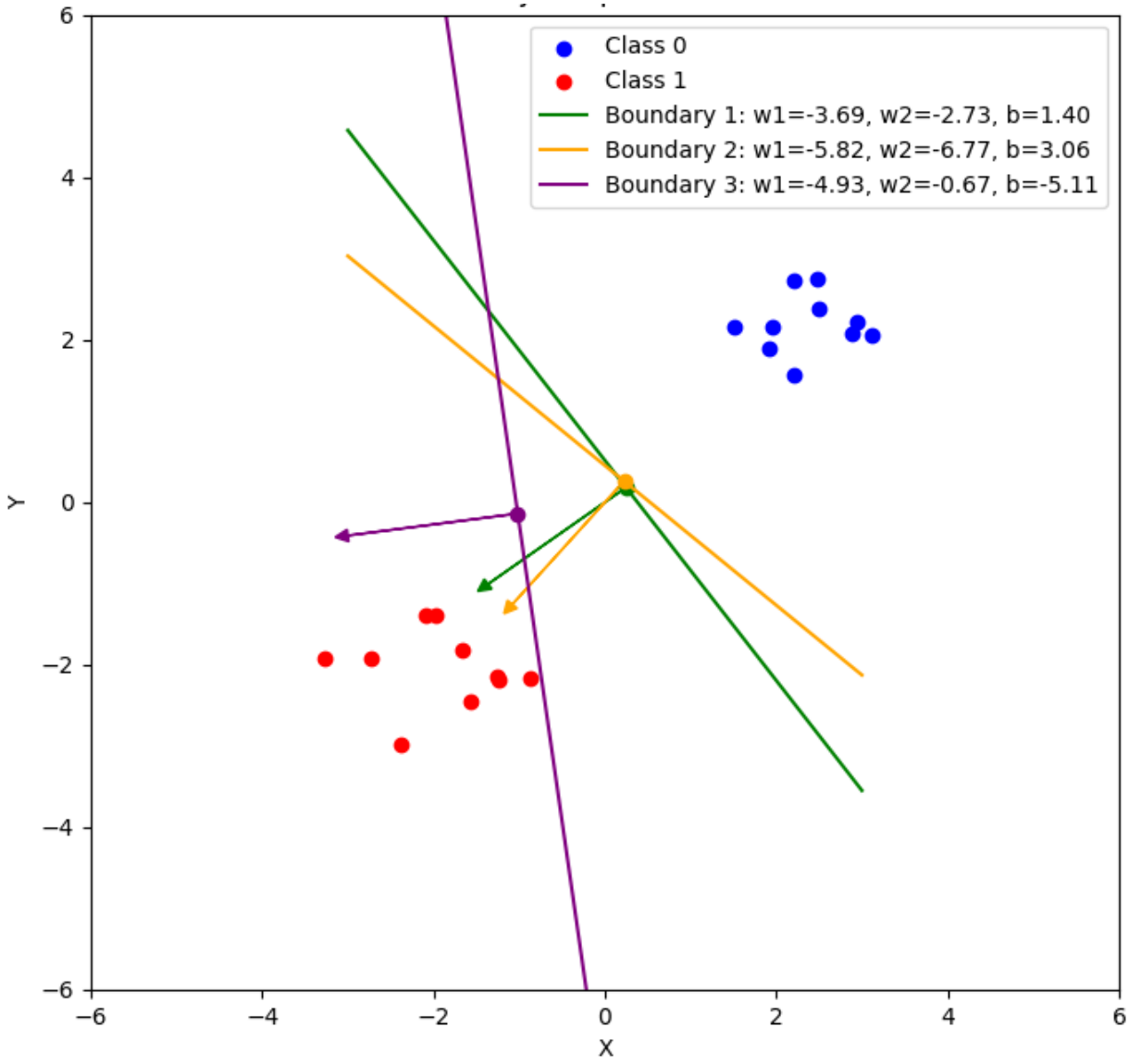
Rinkinio nr.	w_1	w_2	bias
1	-3.6914	-2.7258	1.4039
2	-5.8225	-6.7738	3.0622
3	-4.9342	-0.6738	-5.1115

2 lentelė. Svorių (w_1, w_2) ir bias (b) rinkiniai (slenkstinė funkcija)

Rinkinio nr.	w_1	w_2	bias
1	-6.8206	-7.7925	3.1266
2	-7.2363	-6.0684	-2.6255
3	-9.2162	-4.3439	-7.5961

3 lentelė. Svorių (w_1, w_2) ir bias (b) rinkiniai (sigmoidinė funkcija)

3.3. Gauti duomenys grafe



1 pav. Duomenų grafa

Grafe pateikti mėlyni ir raudoni taškai atitinka pradinius duomenis, kurio klasės yra atitinkamai 0 ir 1. Linijos yra klases skiriančios tiesės, gautos naudojant svorius, kurie buvo rasti su slenkstine funkcija. O rodyklės yra joms tiesėms vektoriai.

3.4. Dirbtinio intelekto sugeneruotos kodo dalys

Su ChatGPT įrankiu aiškinausi kaip naudotis matplotlib biblioteka ir atradau vektorio formulę.

4. Išvada

Šiame dokumente pristatytas paprasto dirbtinio neurono kūrimo procesas, kuriuo siekiama prognozuoti taško klasę (0 ar 1) pagal jo koordinates. Įgyvendintos dvi aktyvacijos funkcijos: slenkstinė ir sigmoidinė. Programa naudodama jas atranda svorius, kuriais galime prognozuoti, ar pateiktas taškas priklauso 0-tai ar 1-ai klasei. Dėl rezultatų aiškumo, grafe greta pradinių taškų nupieštos klases perskiriančios tiesės ir joms statmeni vektoriai.