VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Praktinė užduotis Nr. 1

Dirbtinis neuronas
Artificial neuron

Jokūbas Zaveckis

Darbo vadovas : Doc., Dr., Viktor Medvedev

Vilnius
2025

Iliustracijų sąrašas

Lentelių sąrašas

Turinys

Užduoties tikslas

Šios praktinės užduoties tikslas – išanalizuoti dirbtinio neurono modelio veikimo principus ir realizuoti jo modelį sprendžiant binarinės klasifikacijos uždavinį. Neuronas turės priimti dviejų požymių duomenis, atlikti svorių sumavimą, pritaikyti aktyvacijos funkciją ir grąžinti klasifikacijos rezultatą. Bus įgyvendintos dvi aktyvacijos funkcijos: slenkstinė ir sigmoidinė.

Sugeneruoti duomenys

Sugeneruoti 20 dvimačių taškų (x_1, x_2), kurie buvo suskirstyti į dvi klases (*Klasė 0; Klasė 1*). Kiekvienoje klasėje yra 10 taškų. Tam, kad atskirti atitinkamus taškus koordinačių plokštumoje, *Klasei 0* priskirta kairioji apatinė koordinačių plokštumos dalis, *Klasei 1* – dešinioji viršutinė koordinačių plokštumos dalis.

x_1, x_2 Reikšmės atsitiktinai sugeneruotos nurodytame diapozone:

- Klasei 0: $x_1, x_2 \in [0; 2]$.
- Klasei 1: $x_1, x_2 \in [3; 5]$.

Sugeneruoti duomenys buvo atvaizduoti koordinačių plokštumoje, kur:

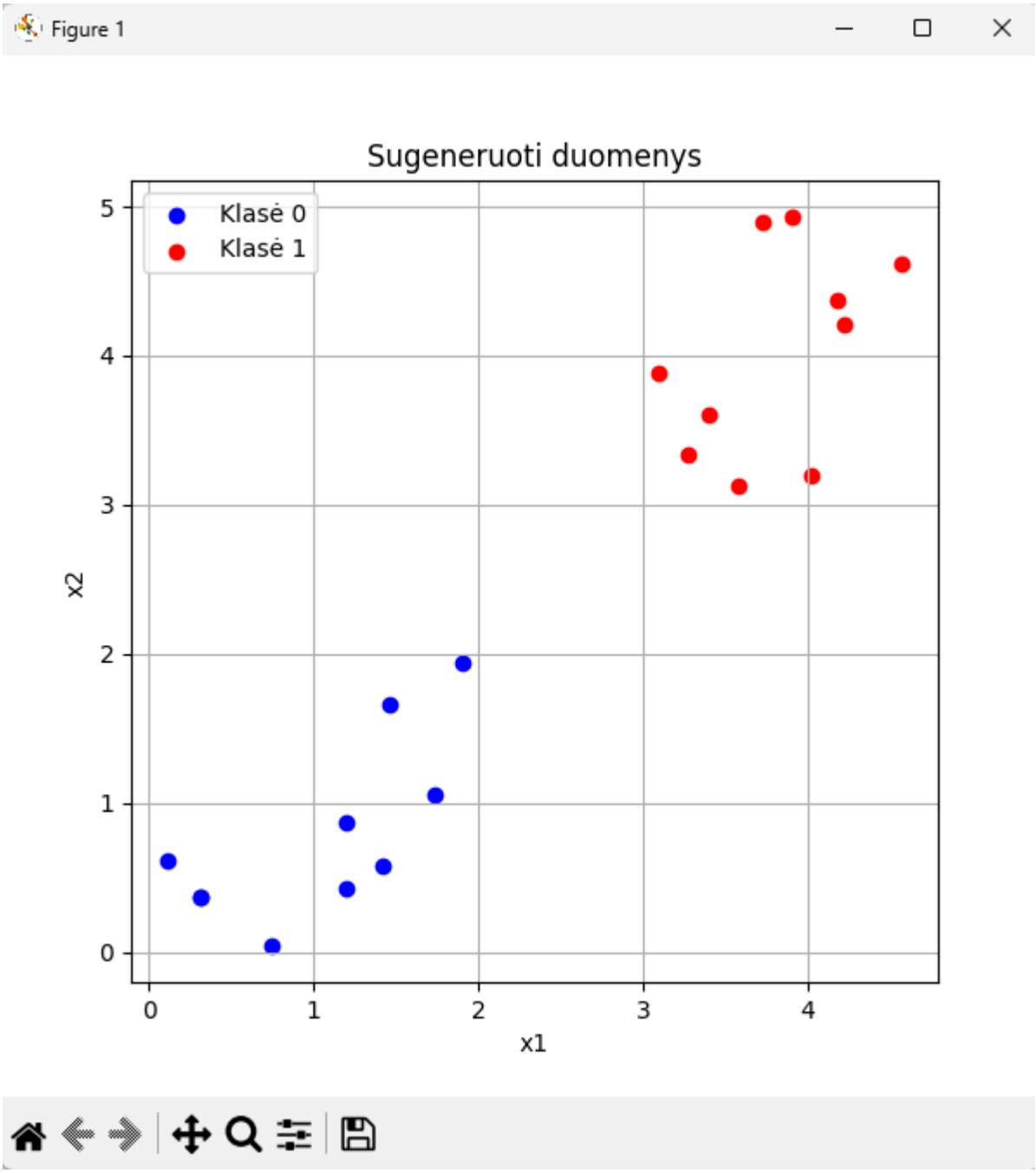
- Klasės 0 taškai buvo vizualizuoti mėlyna spalva.
- Klasės 1 taškai buvo vizualizuoti raudona spalva.

Sugeneruoti taškai atvaizuoti 1 lentėlėje.

1 lentelė *Sugeneruoti taškai*

Klasė 0	*Klasė 1*
(0,75; 0,04)	(4,22; 4,22)
(1,90; 1,94)	(3,28; 3,34)
(1,46; 1,66)	(3,58; 3,13)
(1,20; 0,42)	(3,73; 4,90)
(0,31; 0,36)	(3,91; 4,93)
(0,31; 0,37)	(4,57; 4,62)
(0,12; 0,61)	(3,40; 3,61)
(1,73; 1,05)	(4,03; 3,20)
(1,20; 0,86)	(4,18; 4,37)
(1,42; 0,58)	(3,09; 3,88)

Sugeneruotų duomenų vizualizaciją galima pamatyti 1 paveikslėlyje.



1 pav. *Sugeneruoti duomenys atvaizduoti koordinačių plokštumoje*

Sugeneruoti duomenys buvo išsaugoti kitų užduoties punktų panaudojimui.

Programos kodas

Programa susideda iš 5 failų (*generate_data.py, neuron.py, sigmoid_activation.py, treshhold_activation.py, visualization.py*). Kiekvienas failas atlieka tam tikrą užduoties dalį. Programos tikslas – implementuoti dirbtini neuroną, kuris klasifikuoja duomenis naudodamas 2 skirtingas aktyvacijos funkcijas (slenkstinę is sigmoidinę). Programa taip pat randa geriausius svorius klasifikacijai ir vizualizuoja sprendimo ribas.

1. ***generate_data.py*** – šis failas sugeneruoja 20 taškų, kurie priskiriami dviejoms klasėms.
2. ***neuron.*****py** – dirbtinis neurono modelis.

Neuronas turi:

- 2 įėjimo taškus (x_1, x_2).
- 2 svorius (w_1, w_2).
- Poslinkį (*bias*), kuris padeda reguliuoti sprendimo ribą.
- Aktyvacijos funkciją, kuri nustato neurono išvestį.

Neuronas apskaičiuoja išvestį pagal formulę:

$$a = (w_1 \times x_1) + (w_2 \times x_2) + b.$$

Tada rezultatas perduodamas per aktyvacijos funkciją, kuri gali būti:

1. Slenkstinė funkcija – jei $a \geq 0$, išvestis yra 1, kitu atveju – 0.
2. Sigmoidinė funkcija – išvestis apskaičiuojama pagal formulę:

$$f(a) = \frac{1}{1+e^{-a}}.$$

   Jei rezultatas didesnis nei 0,5, jis suapvalinamas į 1, kitu atveju – į 0.

Šis failas leidžia patikrinti neuroną, pateikiant jam pavyzdinius duomenis.

3. **threshold_activation.py**

   Šis failas ieško geriausių svorių (w_1, w_2) ir poslinkio b, kad neuronui būtų įmanoma teisingai klasifikuoti visus duomenų taškus, naudojant slenkstinę funkciją.

   - Programa atsitiktinai parenka w_1, w_2 ir b reikšmes.
   - Patikrina, ar neuronui pavyksta teisingai klasifikuoti visus 20 taškų.
   - Jei pasirinkti svoriai yra tinkami ir atitinka 100 %, jie išsaugomi.
   - Procesas kartojamas, kol randami trys teisingų svorių rinkiniai.

   Pasibaigus šiai funkcijai, programa išveda tris svorių rinkinius, kurie leidžia atskirti Klasę 0 ir Klasę 1.

4. **sigmoid_activation.py**

   Šis failas veikia taip pat, kaip threshold_activation.py, tačiau vietoj slenkstinės funkcijos naudoja sigmoidinę funkciją.

   - Programa atsitiktinai parenka w_1, w_2 ir b reikšmes.
   - Neuronoas naudoja sigmoidinę funkciją, kad klasifikuotų taškus.
   - Išvesties reikšmės suapvalinamos į 0 arba 1.
   - Jei visos klasifikacijos teisingos, svoriai išsaugomi.
   - Procesas kartojamas, kol randami trys geriausi svorių rinkiniai.

   Pasibaigus programos vykdymui, programa išveda tris tinkamus svorių rinkinius, kurie leidžia atskirti duomenis naudojant sigmoidinę funkciją.

5. **visualization.py**

   Šis failas nupiešia rezultatus grafiškai, nubrėždamas:

   - Sugeneruotus duomenų taškus (mėlyni – Klasė 0, raudoni – Klasė 1).
   - Tris sprendimo ribas (kiekvienai geriausių svorių rinkinių kombinacijai).
   - Svorio vektorius, kurie rodo klasifikavimo kryptį.

   Sprendimo ribos yra braižomos pagal formulę:

$$y = -\frac{w_1 \times x + b}{w_2}.$$

   Svorio vektoriai nubraižomi taip, kad jie būtų statmeni sprendimo ribai.

Programos kodą galima rasti prieduose.

Svoriai ir poslinkis

Kad dirbtinis neuronas teisingai klasifikuotų sugeneruotus duomenis, reikėjo rasti tinkamus svorio koeficientus (w_1, w_2) ir poslinkį (b). Šios reikšmės lemia, kaip neuronas atskiria Klasę 0 ir Klasę 1.

Kadangi naudojamos dvi aktyvacijos funkcijas (slenkstinė ir sigmoidinė), kiekvienai jų reikia atskirai surasti tinkamus svorius.

1. Slenkstinės aktyvacijos funkcijos svoriai

Slenkstinė aktyvacijos funkcija veikia taip:

- Jei neurono išvestis didesnė arba lyg 0, ji priskiria Klasę 1.
- Jei neurono išvestis mažesnė nei 0, ji priskiria Klasę 0.

Kad rasti tinkamus svorius, reikėjo atlikti šiuos žingsnius:

- Atsitiktinai sugeneruoti w_1, w_2 ir b reikšmes intervale $[-5, 5]$.
- Patikrinti, ar neuronas teisingai klasifikuoja visus 20 taškų.
- Jei visi taškai buvo klasifikuoti teisingai, išsaugoti svorius.
- Kartoti procesą tol, kol radami trys tinkamus svorių rinkiniai.

Gaunami šie slenkstinės funkcijos svorių rinkiniai:

2 lentelė *Slenkstinės funkcjos svorių ir poslinkio rezultatai*

Rinkinys	w_1	w_2	Poslinkis (b)
1	1,34	0,36	-4,10
2	1,32	-0,52	-2,07
3	-1,58	3,21	-3,89

2. Sigmoidinės aktyvacijos funkcijos svoriai

Sigmoidinė aktyvacijos funkcija skiriasi tuo, kad ji grąžina reikšmę tarp 0 ir 1:

$$f(a) = \frac{1}{1+e^{-a}}.$$

Kad klasifikacija veiktų, sigmoidinės funkcijos išvestis suapvalinama:

- Jei reikšmė ≥ 0,5, laikome, kad tai Klasė 1.
- Jei reikšmė < 0,5, laikome, kad tai Klasė 0.

Atliekami tokie patys žingsniai, kaip ir su slenkstine funkcija ir gaunami rezultatai (3 lentelė).

3 lentelė *Sigmoidinės funkcijos svorių ir poslinkio rezultatai*

Rinkinys	w_1	w_2	Poslinkis (b)
1	1,34	0,36	-4,10
2	1,32	-0,52	-2,07
3	-1,58	3,21	-3,89

Kadangi duomenys yra aiškiai atskiriami, tiek slenkstinė, tiek sigmoidinė funkcijos rado tuos pačius svorius. Jei duomenys būtų sudėtingesni, sigmoidinės funkcijos svoriai galėtų būti kitokie. Sigmoidinė funkcija yra lankstesnė esant neaiškiems arba persidengiantiems duomenims.

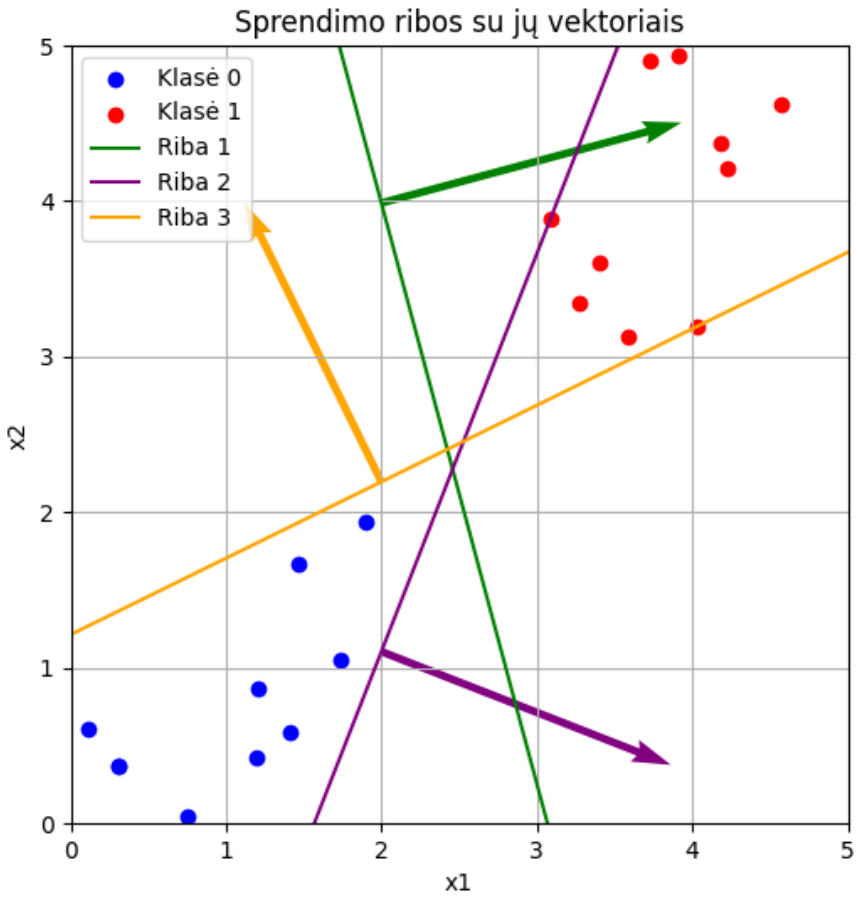
Rezultatų analizė

2 paveikslėlyje yra grafikas, kuris parodo, kaip dirbtinis neuronas klasifikuoja duomenis ir kur yra sprendimo ribos, atskiriančios dvi klases.

Sugeneruoti duomenys sudaro dvi aiškiai atskirtas grupes, kurios gali būti atskirtos tiesėmis. Kiekviena sprendimo riba atitinka skirtingą svorių ir poslinkio reikšmių rinkinį, kuris buvo rastas bandymų būdu. Tai reiškia, kad egzistuoja keletas būdų, kaip galima atskirti šias dvi klases naudojant tiesinį modelį.

Be pačių sprendimo ribų, grafike taip pat pavaizduoti svorio vektoriai, kurie nurodo klasifikavimo kryptį. Jie rodo, kuria kryptimi kinta neurono aktyvacijos funkcija, ir padeda suprasti, kaip neuronas priima sprendimus.

Apskritai, ši vizualizacija leidžia suprasti, kaip skirtingi svorių rinkiniai keičia sprendimo ribų padėtį, tačiau visais atvejais modelis sėkmingai atskiria dvi klases. Tai patvirtina, kad neuronui pavyko išmokti tinkamą duomenų atskyrimą.



2 pav. *Gauti rezultatai*

Išvados

- Dirbtinis neuronas gali būti naudojamas paprastiems klasifikavimo uždaviniams spręsti.
- Slenkstinė ir sigmoidinė aktyvacijos funkcijos gali duoti tuos pačius rezultatus, jei duomenys yra aiškiai atskiriami.
- Svorio koeficientai ir poslinkis lemia, kaip neuronas priima sprendimus.

Dirbtinio intelekto indėlis

Darbe buvo pasitelkta OpenAI ChatGPT įrankiu, kuriant slenkstinės ir sigmoidinės aktivacinių funkcijų realizacijai. Taip pat, pasinaudota sąvokų aiškinimui, dirbtinio neurono analizei.

1 Priedas *generate_data.py*

```
import numpy as np
import matplotlib.pyplot as plt
from generate_data import X, y
from threshold_activation import find_threshold_weights

# Gauna tinkamus svorių rinkinius
valid_weights = find_threshold_weights()

# Nubraizo sugeneruotus duomenu taskus
plt.figure(figsize=(6,6))
plt.scatter(X[:10, 0], X[:10, 1], color='blue', label='Klasė 0')
plt.scatter(X[10:, 0], X[10:, 1], color='red', label='Klasė 1')

# Funkcija sprendimo ribos braizymui
def plot_decision_boundary(w1, w2, b, color, label):
    x_vals = np.linspace(0, 5, 100)
    y_vals = -(w1 * x_vals + b) / w2  # Perskaiciuoja sprendimo riba
    plt.plot(x_vals, y_vals, color=color, label=label)

# Funkcija nubrezia svorio vektoriu, kuris yra statmentas sprendimo ribai
def plot_weight_vector(w1, w2, b, color):
    x_ref = 2  # Parenka taska ant sprendimo ribos
    y_ref = -(w1 * x_ref + b) / w2
    # Normalizuoja vektoriu
    scale_factor = 2 / np.sqrt(w1**2 + w2**2) # Pataiso vektoriaus ilgi
    w1_scaled, w2_scaled = w1 * scale_factor, w2 * scale_factor

    plt.quiver(x_ref, y_ref, w1_scaled, w2_scaled, color=color, angles='xy', scale_units='xy', scale=1, width=0.01)

# Braizo tris sprendimo ribas ir ju svorio vektorius
colors = ['green', 'purple', 'orange']
for i, (w1, w2, b) in enumerate(valid_weights):
    plot_decision_boundary(w1, w2, b, colors[i], f"Riba {i+1}")
    plot_weight_vector(w1, w2, b, colors[i])

plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Sprendimo ribos su jų vektoriais")
plt.legend()
plt.grid(True)
plt.show()
```

2 Priedas *neuron.py*

```
import numpy as np

class ArtificialNeuron:
    def __init__(self, w1, w2, bias, activation_function="threshold"): # Inicijuoja neurono svorius, poslinki ir
aktyvacijos funkcija
        self.w1 = w1
        self.w2 = w2
        self.bias = bias
        self.activation_function = activation_function

    def activate(self, a): # Pritaiko pasirinkitos aktyvacijos funkcija
        if self.activation_function == "threshold":
            return 1 if a >= 0 else 0
        elif self.activation_function == "sigmoid":
            return 1 / (1 + np.exp(-a))  # Sigmoidine funkcija

    def compute_output(self, x1, x2): # Apskaiciuoja neurono isvesti pagal svorius ir iejimus
        a = (self.w1 * x1) + (self.w2 * x2) + self.bias
        return self.activate(a)

# Testavimas su pavyzdiniais iejimais
if __name__ == "__main__":
    neuron = ArtificialNeuron(w1=1.0, w2=1.0, bias=-2.0, activation_function="threshold")

    # Test the neuron with sample inputs
    test_inputs = [(0, 0), (1, 1), (2, 2), (3, 3)]

    print("Neuron Output:")
    for x1, x2 in test_inputs:
        output = neuron.compute_output(x1, x2)
        print(f"Input: (x1={x1}, x2={x2}) -> Output: {output}")
```

3 Priedas *sigmoid_activation.py*

```
import numpy as np
from neuron import ArtificialNeuron
from generate_data import X, y

np.random.seed(42)

def find_sigmoid_weights(): # Atsitiktinai generuoja tinkamus svorius ir poslinki, kurie teisingai klasifikuoja visus taskus
    valid_weights = []
    max_attempts = 10000
    attempt = 0

    while len(valid_weights) < 3 and attempt < max_attempts:
        attempt += 1
        w1 = np.random.uniform(-5, 5)
        w2 = np.random.uniform(-5, 5)
        b = np.random.uniform(-5, 5)

        neuron = ArtificialNeuron(w1, w2, b, activation_function="sigmoid")

        # Patikrina, ar sis svoriu rinkinys teisingai klasifikuoja visus duomenų taskus
        all_correct = all(round(neuron.compute_output(x1, x2)) == label for (x1, x2), label in zip(X, y))

        if all_correct:
            valid_weights.append((w1, w2, b))
            print(f"Rastas tinkamas rinkinys {len(valid_weights)}: w1={w1:.2f}, w2={w2:.2f}, b={b:.2f}")

    if len(valid_weights) < 3:
        print("Nepavyko rasti 3 tinkamu svoriu rinkiniu")

    return valid_weights

if __name__ == "__main__":
    valid_sets = find_sigmoid_weights()
    print("\nFinal Valid Weight Sets (Sigmoid):")
    for i, (w1, w2, b) in enumerate(valid_sets, start=1):
        print(f"Set {i}: w1={w1:.2f}, w2={w2:.2f}, b={b:.2f}")
```

4 Priedas *threshold_activation.py*

```
import numpy as np
from neuron import ArtificialNeuron
from generate_data import X, y

np.random.seed(42)

def find_threshold_weights(): # Atsitiktinai sugeneruoja tinkamus svorius ir poslinki, kurie teisingai klasifikuoja visus taskus
    valid_weights = []
    max_attempts = 10000
    attempt = 0

    while len(valid_weights) < 3 and attempt < max_attempts:
        attempt += 1
        w1 = np.random.uniform(-5, 5)
        w2 = np.random.uniform(-5, 5)
        b = np.random.uniform(-5, 5)

        neuron = ArtificialNeuron(w1, w2, b, activation_function="threshold")

        # Check if this set correctly classifies all data points
        all_correct = all(neuron.compute_output(x1, x2) == label for (x1, x2), label in zip(X, y))

        if all_correct:
            valid_weights.append((w1, w2, b))
            print(f"Rastas tinkamas rinkinys {len(valid_weights)}: w1={w1:.2f}, w2={w2:.2f}, b={b:.2f}")

    if len(valid_weights) < 3:
        print("Nepavyko rasti 3 tinkamu svoriu rinkiniu")

    return valid_weights
if __name__ == "__main__":
    valid_sets = find_threshold_weights()
    print("\nGalutiniai tinkami svoriu rinkiniai (slenkstine funkcija):")
    for i, (w1, w2, b) in enumerate(valid_sets, start=1):
        print(f"Set {i}: w1={w1:.2f}, w2={w2:.2f}, b={b:.2f}")
```

5 Priedas *visulization.py*

```
import numpy as np
import matplotlib.pyplot as plt
from generate_data import X, y
from threshold_activation import find_threshold_weights

# Gauna tinkamus svorių rinkinius
valid_weights = find_threshold_weights()

# Nubraizo sugeneruotus duomenu taskus
plt.figure(figsize=(6, 6))

# Plot the generated data points
plt.scatter(X[:10, 0], X[:10, 1], color='blue', label='Klasė 0')
plt.scatter(X[10:, 0], X[10:, 1], color='red', label='Klasė 1')

# Funkcija sprendimo ribos braizymui
def plot_decision_boundary(w1, w2, b, color, label):
    x_vals = np.linspace(0, 5, 100)
    y_vals = -(w1 * x_vals + b) / w2
    plt.plot(x_vals, y_vals, color=color, label=label)

# Funkcija nubrezti vektoriu
def plot_weight_vector(w1, w2, b, color):
    # Pick a reference point on the line
    x_ref = 2
    y_ref = -(w1 * x_ref + b) / w2

    magnitude = np.sqrt(w1**2 + w2**2)

    # Sumaiznti vektoriu
    scale_factor = 2.0 / magnitude

    w1_scaled = w1 * scale_factor
    w2_scaled = w2 * scale_factor

    # Nupiesia vektoriu
    plt.quiver(
        x_ref, y_ref, w1_scaled, w2_scaled,
        color=color, angles='xy', scale_units='xy',
        scale=1, width=0.01
    )

# Braizo tris sprendimo ribas ir ju svorio vektorius
colors = ['green', 'purple', 'orange']
for i, (w1, w2, b) in enumerate(valid_weights):
    plot_decision_boundary(w1, w2, b, colors[i], f"Riba {i+1}")
    plot_weight_vector(w1, w2, b, colors[i])
```

```
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Sprendimo ribos su jų vektoriais")
plt.legend()
plt.grid(True)

plt.xlim(0, 5)
plt.ylim(0, 5)

plt.gca().set_aspect('equal', adjustable='box')

plt.show()
```