VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFOMATIKOS FAKULTETAS

DUOMENŲ MOKSLO BAKALAURAS

PERCEPTRONO MOKYMAS

Ataskaita

Atliko: Simona Gelžinytė 3 k. 2 gr.

Vadovas: dr. Viktor Medvedev

Vilnius, 2022

Turinys

ĮVADAS

Tyrimo tikslas

Apmokyti vieną neuroną spręsti dviejų klasių uždavinį, atlikti tyrimą su dviem duomenų aibėm.

Tyrimo uždaviniai

- Apskaičiuoti tikslumą mokymo ir testavimo duomenims.
- Nustatyti kaip klasifikavimo tikslumas priklauso nuo epochų skaičiaus.
- Nustatyti kaip paklaidos reikšmės priklauso nuo epochų skaičiaus
- Nustatyti kaip rezultatai priklauso nuo skirtingų mokymosi greičio reikšmių
- Nustatyti kaip rezultatai priklauso nuo to, kuri aktyvacijos funkcija yra naudojama?

Duomenys ir naudota programinė įranga

Šiame tyrime buvo naudoti duomenys iš internetinės svetainės „UCI Machine Learning Repository“. Buvo pasirinktos dvi duomenų aibės:

- Duomenys apie irisus (https://archive.ics.uci.edu/ml/datasets/iris). Buvo naudojami įrašai tik apie dvi irisų rūšis: Versicolor ir Virginica. Šios dvi rūšys vėliau buvo prilygintos klasėms, Versicolor – 0, Virginica – 1 klasei. Šią duomenų aibę sudaro 100 eilučių.
- Duomenys apie krūties vėžio atvejus Viskonsino valstijoje (https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)). Įrašai pateikti tiek apie piktybinius, tiek nepiktybinius auglius. Auglio tipas buvo priskirtas vienai iš klasių: nepiktybinis – 0, piktybinis – 1 klasei. Šią duomenų aibę sudaro 683 eilutės.

Padalinti duomenims į mokymo ir testavimo klases buvo naudojama funkcija train_test_split, ši funkcija, atsitiktinai padalina duomenų aibe į mokymo ir testavimo klases, taip pat, sukurdama atskirą poaibį klasėms saugoti. Santykis tarp mokymo ir testavimo aibių buvo pasirinktas 80:20.

Programos kodas parašytas „Python“ programavimo kalba. Buvo naudota „openpyxl“, „pandas“, „numpy“, „math“, „sklearn.model_selection“, „matplotlib“ bibliotekos.

Sąvokos

- Epocha – kai algoritmas pamato visus duomenis vieną kartą.
- Iteracija – kai algoritmas pamato vieną duomenų eilutę.

PERCEPTRONO MOKYMAS

Naudotos formulės ir funkcijos

1. Įėjimo reikšmių ir svorių sandaugų sumos formulė

$$a = w_0 \cdot x_0 + w_1 \cdot x_1 + \cdots + w_2 \cdot x_n = \sum_{k=1}^{n} w_k \cdot x_k$$

2. Slenkstinė funkcija

$$f(a) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

3. Sigmoidinė funkcija

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

4. Paklaidos minimizavimo formulė

$$E(W) = \frac{1}{2}\sum_{i=1}^{m}(y_i - t_i)^2$$

5. Svorio reikšmių keitimo formulė

$$w_k(t+1) = w_k(t) + \eta(t_i - y_i)x_{ik}$$

Svorio reikšmių keitimo formulės realizavimas programavimo kalba:

```
def nauji_svoriai(svoriai, eilute, isejimo_reiksme, norima_reiksme,
mokymo_greitis):
    # Svorių koregavimas
    nauji_svoriai = []
    for i in range(len(svoriai)):
        nauji_svoriai.append(svoriai[i] + mokymo_greitis*(norima_reiksme -
isejimo_reiksme)*eilute[i])
    return nauji_svoriai
```

Slenkstinės funkcijos metodai

```
def slenkstine_fja(eilute, svoriai):
    a = np.dot(eilute, svoriai)
    if a >= 0:
        return 1
    return 0

def mokymo_slenkstine(mokymo_greitis, epochos, x_mok, y_mok):
    # Mokymo funkcija
    svoriai = [] # sukuriama vieta svoriams saugoti
    svoriai.extend([0.5 for i in range(len(x_mok[0]))]) # paprastumo delei
pradiniai svoriai yra pasirenkami 0.5
    paklaidos = []  # paklaidu skaiciavimas
    for e in range(epochos):
        paklaidu_suma = 0 # paklaidu skaiciavimas
        for i in range(len(x_mok)):
            y = slenkstine_fja(x_mok[i], svoriai) # suzinome kuriai klasei
priklausys
            eilute = x_mok[i] # paimame visa reiksmiu eilute
            rezultatas = y_mok[i] # paimame klase
            svoriai = nauji_svoriai(svoriai, eilute, y, rezultatas,
mokymo_greitis) # senuosius svorius keiciame naujausiais, naudodami svorio
reiksmiu keitimo formule
            paklaidu_suma += (rezultatas - y) ** 2  # paklaidu skaiciavimas

        paklaidos.append([e, (0.5 * paklaidu_suma)[0]])  # paklaidu
skaiciavimas epochoms

    #braizome grafika
    plt.plot(list(zip(*paklaidos))[0], list(zip(*paklaidos))[1])
    plt.xlabel('Epochos')
    plt.ylabel('Paklaidos')
    plt.title("Slenkstinė funkcija")
    plt.show()
    return (svoriai, paklaidos[-1][1])

def testavimo_slenkstine(svoriai, x_test, y_test):
    sekme = 0 # skaiciuosime kiek teisingai duomenu buvo klasifikuota
    for i in range(len(x_test)):
        if slenkstine_fja(x_test[i], svoriai) == y_test[i][0]:
            sekme += 1
    return sekme/len(x_test)
```

Sigmoidinės funkcijos metodai

```
def sigmoidine_fja(eilute, svoriai):
    # Sigmoidine funkcija
    a = np.dot(eilute, svoriai)
    return 1/(1 + math.exp(-a))

def mokymo_sigmoidine(mokymo_greitis, epochos, x_mok, y_mok):
    # Testavimo funkcija
    svoriai = [] # sukuriama vieta svoriams saugoti
    svoriai.extend([0.5 for i in range(len(x_mok[0]))]) # paprastumo delei
pradiniai svoriai yra pasirenkami 0.5
```

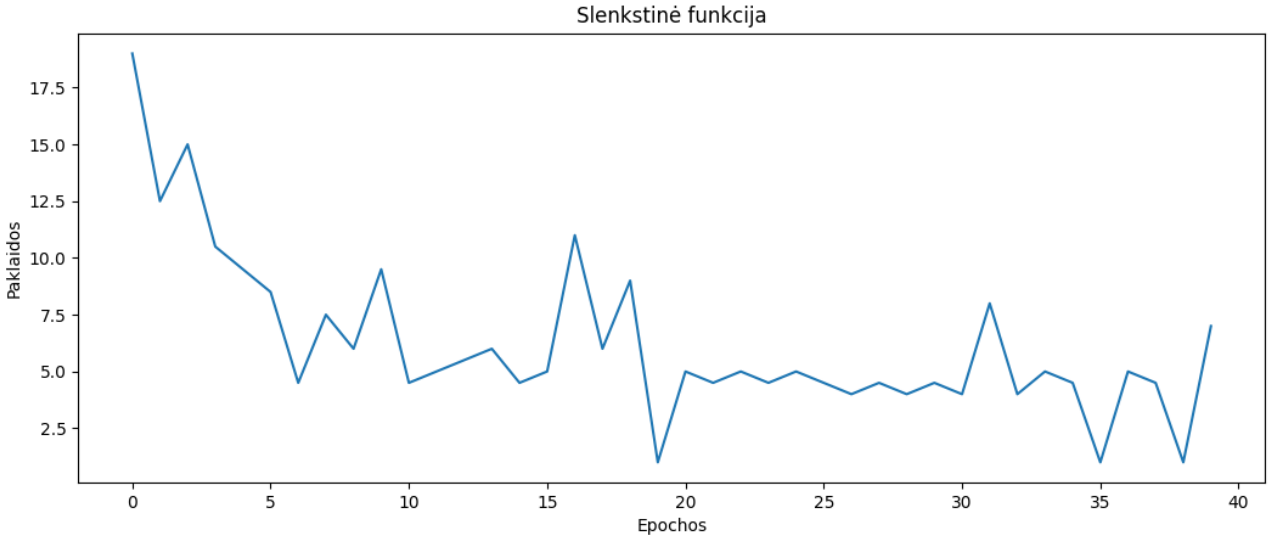
```
    paklaidos = []  # paklaidu skaiciavimas

    for e in range(epochos):
        paklaidu_suma = 0  # paklaidu skaiciavimas
        for i in range(len(x_mok)):
            y = sigmoidine_fja(x_mok[i], svoriai) # suzinome kuriai klasei
priklausys
            eilute = x_mok[i] # paimame visa reiksmiu eilute
            rezultatas = y_mok[i] # paimame klase
            svoriai = nauji_svoriai(svoriai, eilute, y, rezultatas,
mokymo_greitis) # senuosius svorius keiciame naujausiais, naudodami svorio
reiksmiu keitimo formule

            paklaidu_suma += (rezultatas - y) ** 2  # paklaidu skaiciavimas

        paklaidos.append([e, (0.5 * paklaidu_suma)[0]])  # paklaidu
skaiciavimas epochoms

    #braizome grafika
    plt.plot(list(zip(*paklaidos))[0], list(zip(*paklaidos))[1])
    plt.xlabel('Epochos')
    plt.ylabel('Paklaida')
    plt.title("Sigmoidinė funkcija")
    plt.show()
    return (svoriai, paklaidos[-1][1])


def testavimo_sigmoidine(svoriai, x_test, y_test):
    # Matuoja sigmoidinio modelio tiksluma
    sekme = 0 # skaiciuosime kiek teisingai duomenu buvo klasifikuota
    for i in range(len(x_test)):
        if round(sigmoidine_fja(x_test[i], svoriai),0) == y_test[i][0]:
            sekme += 1
    return sekme/len(x_test)
```

REZULTATAI

Irisų duomenys, slenkstinė funkcija

Mokymo greitis – 1

Epochų skaičius - 40



1 pav. Slenkstinė funkcija, naudojant irisų duomenis

Gauti svoriai: [-52.7, -39.4, 63.6, 77.8, -30.5]

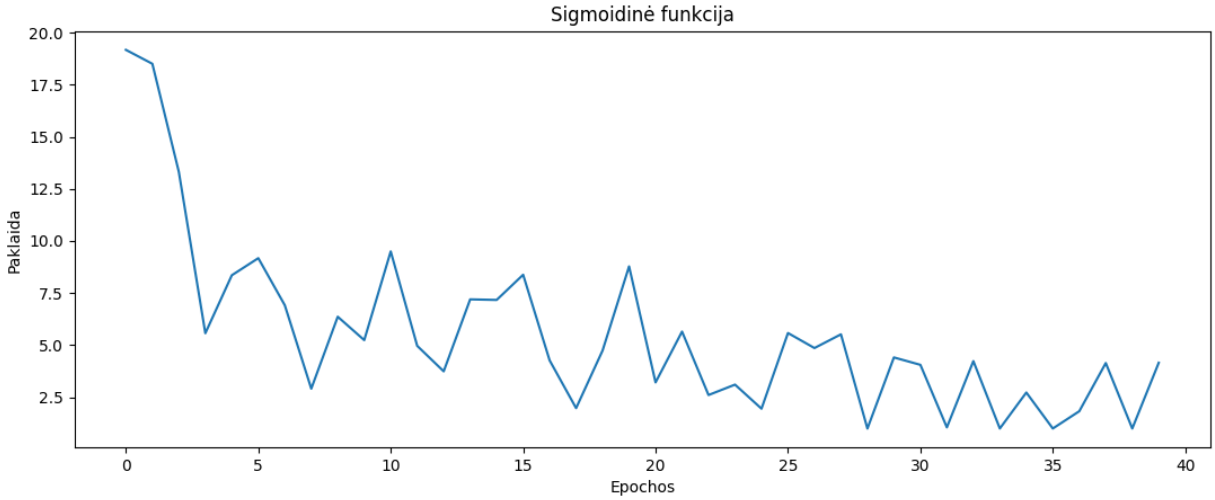
Paklaida: 7.0

Tikslumas: 0.85

Irisų duomenys, sigmoidinė funkcija

Mokymo greitis – 1

Epochų skaičius - 40



2 pav Sigmoidinė funkcija, naudojant irisų duomenis

Gauti svoriai: [-52.0662, -34.9356, 60.9741, 77.7823, -29.4631]

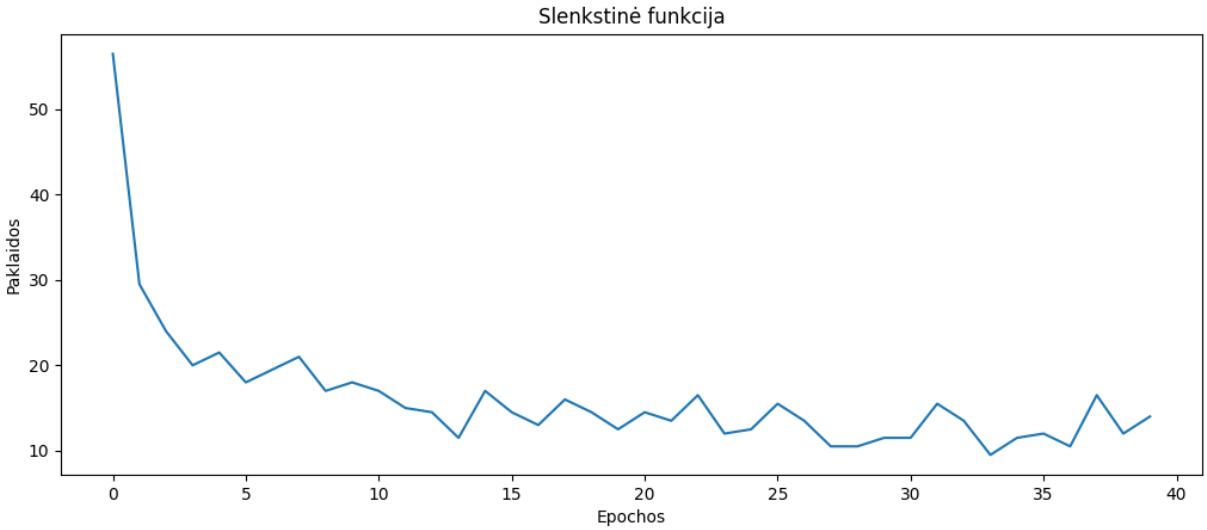
Paklaida: 4.1587814972448625

Tikslumas: 0.9

Vėžio krūties duomenys, slenkstinė funkcija

Mokymo greitis – 1

Epochų skaičius – 40



3 pav. Slenkstinė funkcija, naudojant krūties vėžio duomenis

Gauti svoriai: [11.5, -4.5, 14.5, 11.5, -2.5, 16.5, 10.5, -1.5, 13.5, -262.5]

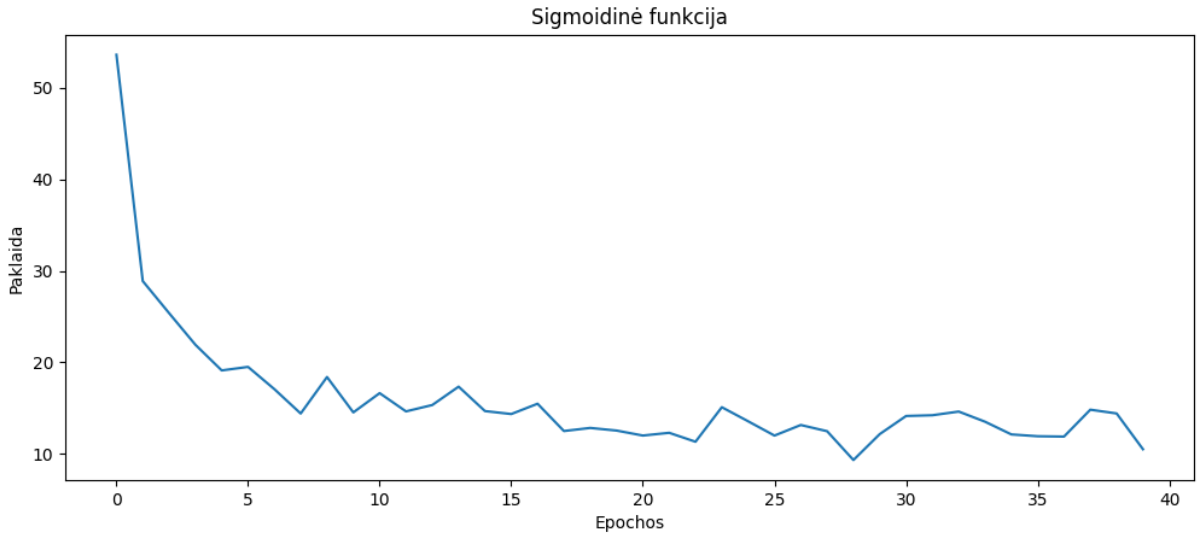
Paklaida: 14.0

Tikslumas: 0.971

Vėžio krūties duomenys, sigmoidinė funkcija

Mokymo greitis – 1

Epochų skaičius – 40



4 pav Sigmoidinė funkcija, naudojant vėžio krūties duomenis.

Gauti svoriai: [14.192, 2.5018, 17.7512, 5.7124, -3.3625, 20.7064, 12.5348, 3.7713, 12.539, -257.773]
Paklaida: 10.501693627794605
Tikslumas: 0.985

Irisų duomenys, slenkstinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

```
# Mokymosi greitis nuo 0,1 iki 1 su zingsniu 0,1
# Epochu skaicius nuo 1 iki 40 su zingsniu 1
```

funkcija	mokymosi_greitis	epochos	tikslumas
Slenkstine	0.2	19	1.0
Slenkstine	0.3	19	1.0
Slenkstine	0.4	9	1.0
Slenkstine	0.5	1	1.0
Slenkstine	1.0	19	1.0

5 pav. slenkstinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

Išfiltravus duomenis, matome, jog geriausias epochų skaičius yra 19 su mokymosi greičiu 0,2 – 0,3. (Čia tikslumas 0,95 yra lygus 1,0, kadangi suprantame, jog tikslumą gauti 1,0 yra beveik neįmanoma)

Irisų duomenys, sigmoidinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

```
# Mokymosi greitis nuo 0,1 iki 1 su zingsniu 0,1
# Epochu skaicius nuo 1 iki 40 su zingsniu 1
```

funkcija	mokymosi_greitis	epochos	tikslumas
Sigmoidine	0.2	22	1.0
Sigmoidine	0.2	28	1.0
Sigmoidine	0.2	30	1.0
Sigmoidine	0.2	31	1.0
Sigmoidine	0.3	15	1.0
Sigmoidine	0.3	22	1.0
Sigmoidine	0.5	17	1.0
Sigmoidine	0.6	5	1.0

6 pav. sigmoidinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

Išfiltravus duomenis, matome, jog geriausias epochų skaičius yra 22, 28, 30, 31 su mokymosi greičiu 0,2 – 0,3. (Čia tikslumas 0,95 yra lygus 1,0, kadangi suprantame, jog tikslumą gauti 1,0 yra beveik neįmanoma)

Vėžio krūties duomenys, slenkstinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

```
# Mokymosi greitis nuo 0,1 iki 1 su zingsniu 0,1
# Epochu skaicius nuo 1 iki 40 su zingsniu 1
```

funkcija	mokymosi_greitis	epochos	tikslumas
Slenkstine	0.1	24	1.0
Slenkstine	0.1	30	1.0
Slenkstine	0.4	27	1.0
Slenkstine	0.5	18	1.0
Slenkstine	1.0	21	1.0

7 pav. slenkstinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

Išfiltravus duomenis, matome, jog geriausias epochų skaičius yra 24 ir 30 su mokymosi greičiu 0,1. (Čia tikslumas 0,95 yra lygus 1,0, kadangi suprantame, jog tikslumą gauti 1,0 yra beveik neįmanoma)

Vėžio krūties duomenys, sigmoidinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

```
# Mokymosi greitis nuo 0,1 iki 1 su zingsniu 0,1
# Epochu skaicius nuo 1 iki 40 su zingsniu 1
```

funkcija	mokymosi_greitis	epochos	tikslumas
Sigmoidine	0.6	28	1.0
Sigmoidine	0.70	27	1.0
Sigmoidine	1.0	38	1.0

8 pav. sigmoidinės funkcijos mokymosi greitis ir gautų reikšmių tikslumas

Išfiltravus duomenis, matome, jog geriausias epochų skaičius yra 27, 28 ir 38 su mokymosi greičiais 0,6; 0,7 bei 1. (Čia tikslumas 0,95 yra lygus 1,0, kadangi suprantame, jog tikslumą gauti 1,0 yra beveik neįmanoma)

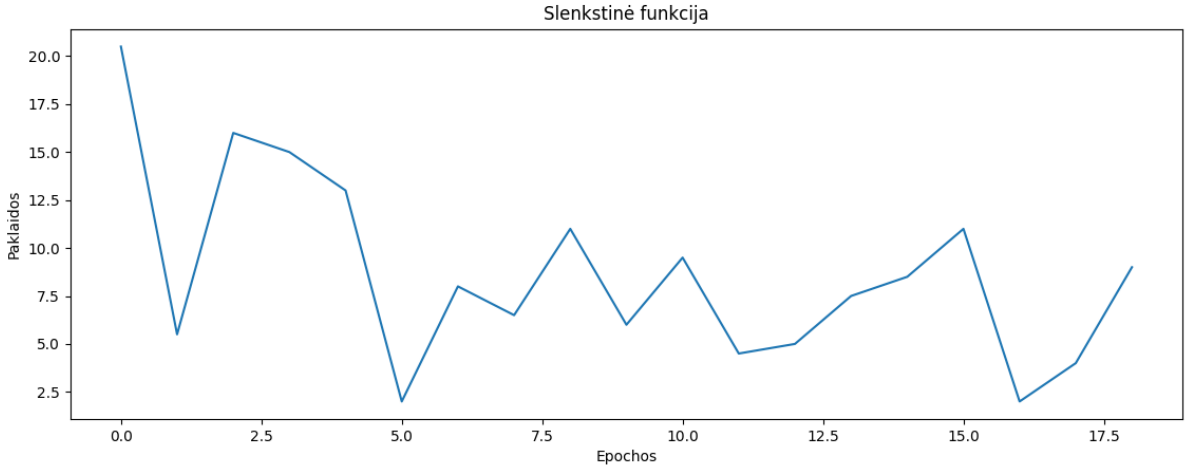
Geriausia kombinacija

Sprendžiant iš irisų duomenų, geriausia kombinacija yra:

Funkcija – slenkstinė
Mokymosi greitis – 0,3
Epochų skaičius - 19

```
def testavimo_slenks_rez(svoriai, xtest, ytest):
    # Matuoja slenkstinio modelio taikluma
    sekme = 0 #  skaiciuosime klasifikavimo tiksluma
    spejimai = [] # irasysime spejama klase
    for i in range(len(xtest)):
        spejimas = slenkstine_fja(xtest[i],svoriai) # spejame kokia klase
turetu buti
        spejimai.append((xtest[i], spejimas, ytest[i][0])) # irasome gauta
rezultata
        if  spejimas == ytest[i][0]:
            sekme +=1 # jeigu spejama klase sutampa sutampa su tikra klase,
padidiname sekmiu skaiciu vienetu
    return (sekme/len(xtest), spejimai) # graziname klasifikavimo tiksluma
ir spejimu rezultatus

geriausia = mokymo_slenkstine(0.3, 19, irisaix_mok, irisaiy_mok) #
```

```
kvieciames funkcija su geriausia kombinacija
geriausias_testas = testavimo_slenks_rez(geriausia[0], irisaix_mok,
irisaiy_mok)  # gauname geriausia testa
print(" Gauti svoriai:", [round(i[0],4) for i in geriausia[0]], "\n",
      "Paklaida:", geriausia[1], "\n"
      " Tikslumas:", geriausias_testas[0])

for i in geriausias_testas[1]:
    print("Įejimas:",i[0], " Spėjimas:", i[1], "Tikra klasė:", i[2])
```



Gauti svoriai: [-11.56, -9.67, 15.17, 17.75, -5.8]

Paklaida: 9.0

Tikslumas: 0.95

Pirmosios 10 eilučių:

```
Įejimas: [6.7 3.3 5.7 2.5 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [4.9 2.5 4.5 1.7 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.3 2.5 4.9 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.5 2.5 4.  1.3 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.4 2.9 4.3 1.3 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.1 2.9 4.7 1.4 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5.7 2.5 5.  2.  1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [5.6 2.8 4.9 2.  1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [7.6 3.  6.6 2.1 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.4 3.1 5.5 1.8 1. ]  Spėjimas: 1 Tikra klasė: 1
```

9 pav. Geriausia kombinacija irisų duomenims, pirmosios 10 eilučių

Paskutinės 10 eilučių:

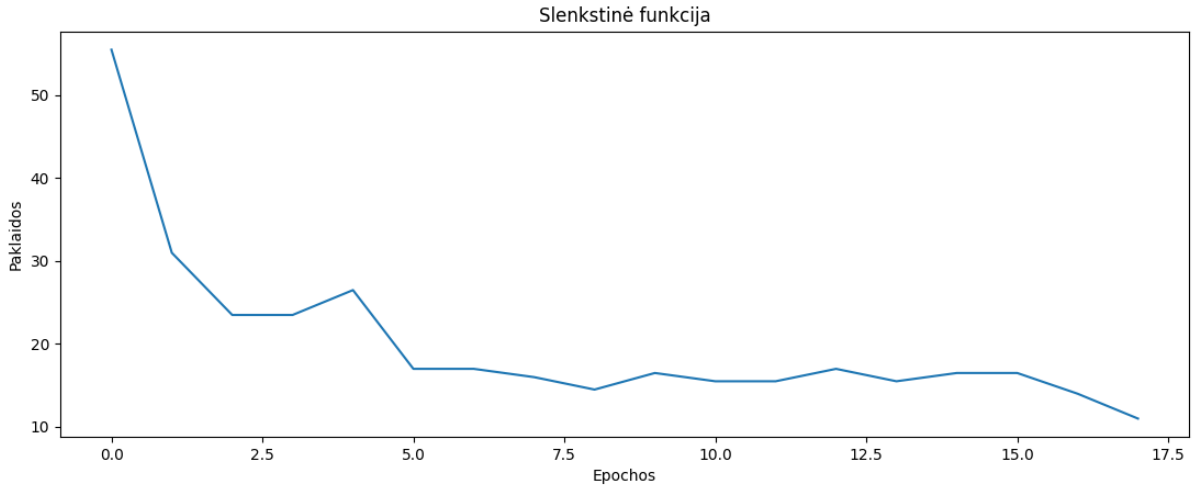
```
Įejimas: [7.4 2.8 6.1 1.9 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [5.5 2.4 3.8 1.1 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.7 3.  5.  1.7 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.8 3.  5.5 2.1 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.6 2.9 4.6 1.3 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.3 2.7 4.9 1.8 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [5.6 3.  4.5 1.5 1. ]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [6.4 2.7 5.3 1.9 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [6.4 2.8 5.6 2.1 1. ]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [5.9 3.  5.1 1.8 1. ]  Spėjimas: 1 Tikra klasė: 1
```

10 pav. Geriausia kombinacija irisų duomenims, paskutinės 10 eilučių

Sprendžiant iš vėžio krūties duomenų, geriausia kombinacija yra:

Funkcija – slenkstinė
Mokymosi greitis – 0,5
Epochų skaičius - 18

```
geriausia = mokymo_slenkstine(0.5, 18, vezysx_mok, vezysy_mok) # kvieciames
funkcija su geriausia kombinacija
geriausias_testas = testavimo_slenks_rez(geriausia[0], vezysx_mok,
vezysy_mok) # gauname geriausia testa
print(" Gauti svoriai:", [round(i[0],4) for i in geriausia[0]], "\n",
      "Paklaida:", geriausia[1], "\n"
      " Tikslumas:", geriausias_testas[0])

for i in geriausias_testas[1]:
    print("Įejimas:",i[0], " Spėjimas:", i[1], "Tikra klasė:", i[2])
```



Gauti svoriai: [1.5, 2.5, 5.5, 5.5, -3.5, 6.5, 6.5, 1.5, 7.0, -100.0]

Paklaida: 11.0

Tikslumas: 0.961

Pirmosios 10 eilučių:

```
Įejimas: [4 4 2 1 2 5 2 1 2 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [3 3 2 1 3 1 3 6 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [3 1 1 1 2 1 2 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [10  7  7  4  5 10  5  7  2  1]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [ 6  6  7 10  3 10  8 10  2  1]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [1 1 1 1 1 1 3 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [10  4  7  2  2  8  6  1  1  1]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [10  3  6  2  3  5  4 10  2  1]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [4 1 1 1 2 1 1 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5 1 1 1 1 1 3 1 1 1]  Spėjimas: 0 Tikra klasė: 0
```

11 pav Geriausia kombinacija vėžio krūties duomenims, pirmosios 10 eilučių.

Paskutinės 10 eilučių:

```
Įejimas: [ 8 10 10 10  5 10  8 10  6  1]  Spėjimas: 1 Tikra klasė: 1
Įejimas: [2 1 1 1 2 1 2 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [1 1 1 1 2 1 1 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [4 1 1 3 2 1 1 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [5 3 6 1 2 1 1 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [2 1 1 1 2 1 1 1 5 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [1 1 2 1 2 2 4 2 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [3 3 2 6 3 3 3 5 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [1 1 2 2 2 1 3 1 1 1]  Spėjimas: 0 Tikra klasė: 0
Įejimas: [1 1 1 1 2 1 3 1 1 1]  Spėjimas: 0 Tikra klasė: 0
```

12 pav . Geriausia kombinacija vėžio krūties duomenims, paskutinės 10 eilučių

IŠVADOS

Supratau, jog kuo daugiau epochų nereiškia, kad modelis bus tikslesnis, logiškiau yra mokyti modelį tol, kol jo paklaida sumažėja iki norimo lygio.

Taip pat per didelis mokymosi greitis gali reikšti, kad svoriai šokinėja per dideliais tarpais ir nebus surandami optimalūs svoriai, todėl geriau rinktis mažesnį mokymosi greitį, jeigu tam yra resursų.

Didžiausią įtaką šios užduoties rezultatams dare duomenų rinkinio dydis, nes kaip galima buvo pastebėti, su vėžio krūties duomenis tikslumas beveik nesikeičia, o su irisų labiau šokinėja, tad vienas svarbiausių kriterijų kuriant neuroninius tinklus – turėti kuo įmanoma daugiau duomenų.

PROGRAMOS KODAS

```
import openpyxl
import pandas as pd
import numpy as np
import math
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt

#nuskaitome irisu duomenis
irisai = pd.read_excel('/Users/simonagelzinyte/Documents/Duomenų mokslas/5
semestras/AI/iris.xlsx', header = None)
irisai.columns = ["x1", "x2", "x3", "x4", "klase"]

#nuskaitome kruties vezio duomenis
vezys = pd.read_excel('/Users/simonagelzinyte/Documents/Duomenų mokslas/5
semestras/AI/breast-cancer-wisconsin.xlsx', header= None)
vezys.columns =["x1","x2" ,"x3", "x4", "x5", "x6", "x7", "x8", "x9",
"klase"]

irisai["b"] = 1
irisaix_mok, irisaix_test, irisaiy_mok, irisaiy_test = train_test_split(
    np.array(irisai.loc[:, irisai.columns != "klase"]),
    np.array(irisai.loc[:, irisai.columns == "klase"]),
    test_size = 0.2, random_state = 5)

vezys["b"] = 1
vezysx_mok, vezysx_test, vezysy_mok, vezysy_test = train_test_split(
    np.array(vezys.loc[:, vezys.columns != "klase"]),
    np.array(vezys.loc[:, vezys.columns == "klase"]),
    test_size= 0.2, random_state = 5)

def nauji_svoriai(svoriai, eilute, isejimo_reiksme, norima_reiksme,
mokymo_greitis):
    # Svorių koregavimas
    nauji_svoriai = []
    for i in range(len(svoriai)):
        nauji_svoriai.append(svoriai[i] + mokymo_greitis*(norima_reiksme -
isejimo_reiksme)*eilute[i])
    return nauji_svoriai

def slenkstine_fja(eilute, svoriai):
    a = np.dot(eilute, svoriai)
    if a >= 0:
        return 1
    return 0

def mokymo_slenkstine(mokymo_greitis, epochos, x_mok, y_mok):
    # Mokymo funkcija
    svoriai = [] # sukuriama vieta svoriams saugoti
    svoriai.extend([0.5 for i in range(len(x_mok[0]))]) # paprastumo delei
pradiniai svoriai yra pasirenkami 0.5
    paklaidos = []  # paklaidu skaiciavimas
    for e in range(epochos):
        paklaidu_suma = 0 # paklaidu skaiciavimas
        for i in range(len(x_mok)):
            y = slenkstine_fja(x_mok[i], svoriai) # suzinome kuriai klasei
priklausys
            eilute = x_mok[i] # paimame visa reiksmiu eilute
```

```
            rezultatas = y_mok[i] # paimame klase
            svoriai = nauji_svoriai(svoriai, eilute, y, rezultatas,
mokymo_greitis) # senuosius svorius keiciame naujausiais, naudodami svorio
reiksmiu keitimo formule
            paklaidu_suma += (rezultatas - y) ** 2  # paklaidu skaiciavimas

        paklaidos.append([e, (0.5 * paklaidu_suma)[0]])  # paklaidu
skaiciavimas epochoms

    #braizome grafika
    plt.plot(list(zip(*paklaidos))[0], list(zip(*paklaidos))[1])
    plt.xlabel('Epochos')
    plt.ylabel('Paklaidos')
    plt.title("Slenkstinė funkcija")
    plt.show()
    return (svoriai, paklaidos[-1][1])

def testavimo_slenkstine(svoriai, x_test, y_test):
    sekme = 0 # skaiciuosime kiek teisingai duomenu buvo klasifikuota
    for i in range(len(x_test)):
        if slenkstine_fja(x_test[i], svoriai) == y_test[i][0]:
            sekme += 1
    return sekme/len(x_test)

def sigmoidine_fja(eilute, svoriai):
    # Sigmoidine funkcija
    a = np.dot(eilute, svoriai)
    return 1/(1 + math.exp(-a))

def mokymo_sigmoidine(mokymo_greitis, epochos, x_mok, y_mok):
    # Testavimo funkcija
    svoriai = [] # sukuriama vieta svoriams saugoti
    svoriai.extend([0.5 for i in range(len(x_mok[0]))]) # paprastumo delei
pradiniai svoriai yra pasirenkami 0.5
    paklaidos = []  # paklaidu skaiciavimas epochoms

    for e in range(epochos):
        paklaidu_suma = 0  # paklaidu skaiciavimas
        for i in range(len(x_mok)):
            y = sigmoidine_fja(x_mok[i], svoriai) # suzinome kuriai klasei
priklausys
            eilute = x_mok[i] # paimame visa reiksmiu eilute
            rezultatas = y_mok[i] # paimame klase
            svoriai = nauji_svoriai(svoriai, eilute, y, rezultatas,
mokymo_greitis) # senuosius svorius keiciame naujausiais, naudodami svorio
reiksmiu keitimo formule

            paklaidu_suma += (rezultatas - y) ** 2  # paklaidu skaiciavimas

        paklaidos.append([e, (0.5 * paklaidu_suma)[0]])  # paklaidu
skaiciavimas

    #braizome grafika
    """plt.plot(list(zip(*paklaidos))[0], list(zip(*paklaidos))[1])
    plt.xlabel('Epochos')
    plt.ylabel('Paklaida')
    plt.title("Sigmoidinė funkcija")
    plt.show()"""
    return (svoriai, paklaidos[-1][1])

def testavimo_sigmoidine(svoriai, x_test, y_test):
```

```
    # Matuoja sigmoidinio modelio tiksluma
    sekme = 0 # skaiciuosime kiek teisingai duomenu buvo klasifikuota
    for i in range(len(x_test)):
        if round(sigmoidine_fja(x_test[i], svoriai),0) == y_test[i][0]:
            sekme += 1
    return sekme/len(x_test)

irisai_slenkstine = mokymo_slenkstine(1, 40, irisaix_mok, irisaiy_mok)
print("Gauti svoriai:", [round(i[0],4) for i in irisai_slenkstine[0]], " ",
      "Paklaida:", irisai_slenkstine[1],
      "Tikslumas:", testavimo_slenkstine(irisai_slenkstine[0],
irisaix_test, irisaiy_test))

irisai_sigmoidine = mokymo_sigmoidine(1, 40, irisaix_mok, irisaiy_mok)
print("Gauti svoriai:", [round(i[0],4) for i in irisai_sigmoidine[0]], " ",
      "Paklaida:", irisai_sigmoidine[1],
      "Tikslumas:", testavimo_sigmoidine(irisai_sigmoidine[0],
irisaix_test, irisaiy_test))

vezio_slenkstine = mokymo_slenkstine(1, 40, vezysx_mok, vezysy_mok)
print("Gauti svoriai:", [round(i[0],4) for i in vezio_slenkstine[0]], " ",
      "Paklaida:", vezio_slenkstine[1],
      "Tikslumas:", testavimo_slenkstine(vezio_slenkstine[0], vezysx_test,
vezysy_test))

vezio_sigmoidine = mokymo_sigmoidine(1, 40, vezysx_mok, vezysy_mok)
print("Gauti svoriai:", [round(i[0],4) for i in vezio_sigmoidine[0]], " ",
      "Paklaida:", vezio_sigmoidine[1],
      "Tikslumas:", testavimo_sigmoidine(vezio_sigmoidine[0], vezysx_test,
vezysy_test))

irisu_testas = []
for mokymo_greitis in np.arange(0.1, 1.1, 0.1):
    for epochos in np.arange(1, 41, 1):
        model1 = mokymo_slenkstine(mokymo_greitis, epochos, irisaix_mok,
irisaiy_mok)
        accuracy1 = testavimo_slenkstine(model1[0], irisaix_test,
irisaiy_test)
        irisu_testas.append(["Slenkstine", mokymo_greitis, epochos,
accuracy1])

        model2 = mokymo_sigmoidine(mokymo_greitis, epochos, irisaix_mok,
irisaiy_mok)
        accuracy2 = testavimo_sigmoidine(model2[0], irisaix_test,
irisaiy_test)
        irisu_testas.append(["Sigmoidine", mokymo_greitis, epochos,
accuracy2])

pd.DataFrame(irisu_testas, columns =
["funkcija","mokymosi_greitis","epochos","tikslumas"]).to_csv("irisai2.csv"
, index = False)

vezys_testas = []
for mokymo_greitis in np.arange(0.1, 1.1, 0.1):
    for epochos in np.arange(1, 41, 1):
        model1 = mokymo_slenkstine(mokymo_greitis, epochos, vezysx_mok,
vezysy_mok)
        accuracy1 = testavimo_slenkstine(model1[0], vezysx_test,
vezysy_test)
        vezys_testas.append(["Slenkstine", mokymo_greitis, epochos,
accuracy1])
```

```
        model2 = mokymo_sigmoidine(mokymo_greitis, epochos, vezysx_mok,
vezysy_mok)
        accuracy2 = testavimo_sigmoidine(model2[0], vezysx_test,
vezysy_test)
        vezys_testas.append(["Sigmoidine", mokymo_greitis, epochos,
accuracy2])

pd.DataFrame(vezys_testas, columns =
["funkcija","mokymosi_greitis","epochos","tikslumas"]).to_csv("vezio.csv",
index = False)

def testavimo_slenks_rez(svoriai, xtest, ytest):
    # Matuoja slenkstinio modelio taikluma
    sekme = 0 #  skaiciuosime klasifikavimo tiksluma
    spejimai = [] # irasysime spejama klase
    for i in range(len(xtest)):
        spejimas = slenkstine_fja(xtest[i],svoriai) # spejame kokia klase
turetu buti
        spejimai.append((xtest[i], spejimas, ytest[i][0])) # irasome gauta
rezultata
        if  spejimas == ytest[i][0]:
            sekme +=1 # jeigu spejama klase sutampa sutampa su tikra klase,
padidiname sekmiu skaiciu vienetu
    return (sekme/len(xtest), spejimai) # graziname klasifikavimo tiksluma
ir spejimu rezultatus

geriausia = mokymo_slenkstine(0.3, 19, irisaix_mok, irisaiy_mok) #
kvieciames funkcija su geriausia kombinacija
geriausias_testas = testavimo_slenks_rez(geriausia[0], irisaix_mok,
irisaiy_mok) # gauname geriausia testa
print(" Gauti svoriai:", [round(i[0],4) for i in geriausia[0]], "\n",
      "Paklaida:", geriausia[1], "\n"
      " Tikslumas:", geriausias_testas[0])

for i in geriausias_testas[1]:
    print("Įejimas:",i[0], " Spėjimas:", i[1], "Tikra klasė:", i[2])


geriausia = mokymo_slenkstine(0.5, 18, vezysx_mok, vezysy_mok) # kvieciames
funkcija su geriausia kombinacija
geriausias_testas = testavimo_slenks_rez(geriausia[0], vezysx_mok,
vezysy_mok) # gauname geriausia testa
print(" Gauti svoriai:", [round(i[0],4) for i in geriausia[0]], "\n",
      "Paklaida:", geriausia[1], "\n"
      " Tikslumas:", geriausias_testas[0])

for i in geriausias_testas[1]:
    print("Įejimas:",i[0], " Spėjimas:", i[1], "Tikra klasė:", i[2])
```