Vilniaus Universitetas

Perceptrono mokymas

Dirbtinio intelekto pagrindai

2 užduotis

Darbą atliko:

Dovydas Martinkus

Duomenų Mokslas 4 kursas 2 gr.

Vilnius, 2022

Turinys

1 Tikslas ir uždaviniai

Tikslas: apmokyti vieną neuroną spręsti nesudėtingą dviejų klasių uždavinį, atlikti tyrimą.

Uždaviniai:

Naudojamų duomenų aibių paruošimas.

Duomenų aibių padalijimas į mokymo ir testavimo aibes, neurono mokymo ir testavimo įgyvendinimas.

Klasifikavimo rezultatų priklausomybės nuo epochų skaičiaus, mokymo greičio parametro reikšmės, naudojamos aktyvacijos funkcijos tyrimas.

Visų tyrimų gautus rezultatus pateikimas lentelėse arba grafikuose.

2 Duomenys

Užduotyje naudoti 2 duomenų rinkiniai:

Irisų duomenų aibė sudaro sudaro 150 stebėjimų, iš kurių kiekvienas turi po 4 skaitinius požymius ir klasę, kuriai priklauso. Šiuose duomenyse yra trys klasės Setosa, Versicolor ir Virginica. Analizei naudotos tik dvi: Versicolor ir Virginica. Prieiga per internetą: https://archive.ics.uci.edu/ml/datasets/iris.

Krūties vėžio duomenų aibę sudaro 6873 stebėjimai, turintys po 10 skaitinių požymių ir klasės kintamąjį, kurį sudaro dvi klasės: 2 – nepiktybinis navikas , 4 – piktybinis navikas. Prieiga per internetą: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic).

Abi duomenų aibes pasirinkta dalinti į mokymo ir testavimo aibes naudojant santykį 80-20.

Kai kurie ataskaitoje naudojami terminai:

Epocha – algoritmo pilnas perėjimas pro visus turimus mokymo duomenis.

Iteracija – tai mokymo proceso dalis, kai į tinklą pateikiamas vienas mokymo aibės duomuo.

3 Užduoties ataskaita

3.1 Programinis kodas

Užduotis atlikta naudojant programavimo kalbą „Python“. Žemiau pateiktas programinis kodas su tarpiniais paaiškinimais:

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



# pirmojo duomenų rinkinio nuskaitymas, pertvarkymas ir padalijimas į mokymo ir
testavimo aibes
x1 = pd.read_csv('iris.data', sep=",", header=None).values

x1 = x1[x1[:,4] != "Iris-setosa"]
x1 = np.column_stack([np.ones(x1.shape[0]),x1])
x1[:,-1] = np.where(x1[:,-1]=='Iris-virginica',0,1)

X1_train, X1_test, y1_train, y1_test = train_test_split(x1[:,0:-1],x1[:,-1],

train_size=0.8,random_state=123,stratify=x1[:,-1])


# antrojo duomenų rinkinio nuskaitymas, pertvarkymas ir padalijimas į mokymo ir
testavimo aibes
x2 = pd.read_csv('breast-cancer-wisconsin.data', sep=",", header=None).values

x2 = x2[x2[:,6] != "?"]
x2[:,-1] = np.where(x2[:,-1]==2,0,1)
x2 = x2[:,1:]
x2 = np.column_stack([np.ones(x2.shape[0]),x2])
x2 = x2.astype(float)

X2_train, X2_test, y2_train, y2_test = train_test_split(x2[:,0:-1],x2[:,-1],

train_size=0.8,random_state=123,stratify=x2[:,-1])
```

Užduotyje naudotos slenkstinė ir sigmoidinė aktyvacijos funkcijos:

```
def sigmoid(row,weights):
    # Sigmoidine aktyvacijos f-ja
    a = np.dot(row, weights)
    return 1/(1+math.exp(-a))
```

```
def threshold(row,weights):
    # Slenkstinė aktyvacijos f-ja
    a = np.dot(row, weights)
    return int(a >= 0)

# abi aktyvacijos funkcijos skirtos paduoti kaip tolimesnių funkcijų argumentas
```

Svoriai atnaujinti naudojantis formule:

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

```
def update_weights(weights, row, y_true, y_pred, lrate):
    # Svorius atnaujinanti funkcija
    updated_weights = []
    for i in range(len(weights)):
        updated_weights.append(weights[i] + lrate*(y_true - y_pred)*row[i])
    return updated_weights
```

```
from matplotlib.ticker import MaxNLocator

def lineplot(y, xlabel,ylabel, title,c="b"):
    # funkcija, skirta nubraižyti linijinį grafiką
    fig, ax = plt.subplots(figsize=(8, 4))
    ax.plot(range(1,len(y)+1),y, alpha = 0.7,c=c)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
    ax.xaxis.set_major_locator(MaxNLocator(integer=True))
```

Kaip pradines parametrų reikšmes pasirinkta naudoti 20 epochų ir mokymosi greitį (angl. learning rate) lygų 0.5.

Klasifikavimo tikslumas naudojant mokymo ir testavimo aibes skaičiuotas padalijant teisingai priskirtų klasių skaičių iš bendro stebėjimų atitinkamoje aibėje skaičiaus.

```
def train_perceptron(X, y, activation, lrate, epochs, plot = True):
    weights = np.full(X.shape[1], 1) # pradiniai svoriai
    errors = []  # paklaidų sąrašas
    accuracies = [] # klasifikavimo tikslumų sąrašas

    for e in range(epochs):
        error = 0
        accuracy = 0
```

```
        for i in range(len(X)):
            y_true = y[i]
            row = X[i]
            y_pred = activation(row, weights)
            weights = update_weights(weights, row, y_true, y_pred , lrate)

            error += (y_true - y_pred)**2
            accuracy += round(y_pred,0) == y_true

        errors.append(round(error,2)) # sąrašas papildomas paklaida, gauta po
kiekvienos epochos
        accuracies.append(round(accuracy/len(X),2))

    if plot:
        names = {"sigmoid": "sigmoidinę", "threshold": "slenkstinę"}
        lineplot(errors, "Epocha", "Paklaida", "Epochų paklaida, naudojant " +
                 names[activation.__name__] +
                     " aktyvacijos f-ją","b")
        lineplot(accuracies, "Epocha", "Paklaida", "Epochų tikslumas mokymo
duomenims, naudojant " +
                 names[activation.__name__] +
                     " aktyvacijos f-ją","orange")
    return errors, accuracies, weights
```

```
def test_accuracy(X_test, y_test, weights, activation):
    # Apskaičiuoja tikslumą testavimo duomenims, naudojant pasirinktą aktyvacijos
funkciją
    correct = []
    error = 0
    for i in range(len(X_test)):
        y_pred = round(activation(X_test[i],weights),0)
        correct.append(y_pred == y_test[i])
        error += (y_test[i] - y_pred)**2
    return round(np.mean(correct),2), round(error,2)
```

3.2 Tyrimo rezultatai

Pirmiausia naudotas pirmas (irisų) duomenų rinkinys.

```
sns.set_context("notebook")
errors, _, weights = train_perceptron(X1_train, y1_train, threshold, 0.5, 20)
# perceptrono mokymas, naudojantis mokymo duomenimis ir slenstine aktyvacijos
funkcija
```

```
print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
threshold)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
threshold)[1])
Galutiniai svoriai:  [21.0, 21.6, 38.1, -49.2, -49.7]
Paklaida:  12
Tikslumas testavimo duomenims:  0.55
Paklaida testavimo duomenims:  9
```

Paklaidos ir tikslumo priklausomybė nuo epochos pavaizduota grafiškai naudojant linijinę diagramą (atitinkamai 1 ir 2 pav.). Pastebimas paklaidos mažėjimas ir tikslumo didėjimas didėjant atliktų iteracijų skaičiui, tačiau su 0,5 mo kymosi greičiu jis yra stipriai nepastovus: neretai sekančios epochos metu gauta paklaida daug didesnė už praėjusi os. Tikėtina, kad ši priežastis atsirado dėl per didelio parinkto mokymosi greičio.

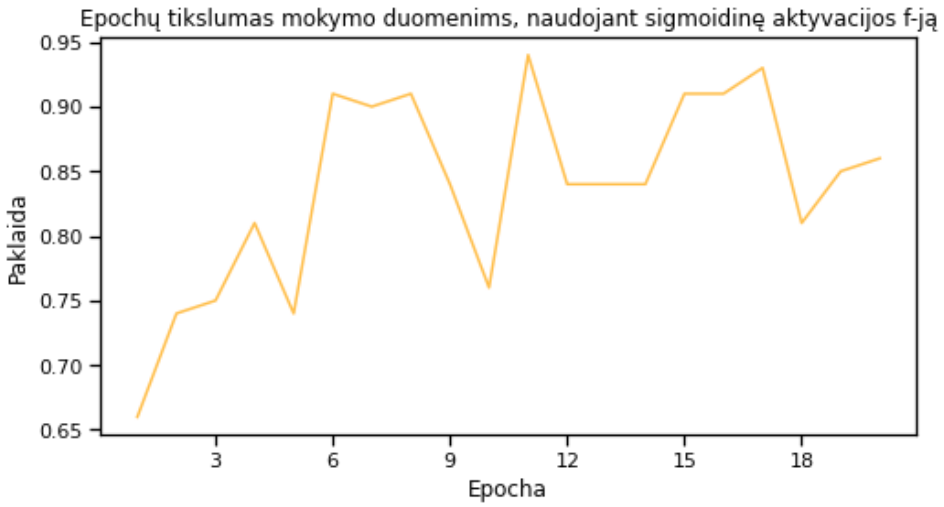
1 pav. Paklaida pagal epochą, naudojant slenkstinę aktyvacijos funkciją (pirmas duomenų rinkinys)

2 pav. Tikslumas pagal epochą, naudojant slenkstinę aktyvacijos funkciją (pirmas duomenų rinkinys)

```
serrors, _, weights = train_perceptron(X1_train, y1_train, sigmoid, 0.5, 20)
# sigmoidinė aktyvacijos fuhnkcija

print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
sigmoid)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
sigmoid)[1])
Galutiniai svoriai:  [21.72, 20.92, 46.34, -44.19, -43.66]
Paklaida:  10.95
Tikslumas testavimo duomenims:  0.9
Paklaida testavimo duomenims:  2.0
```

Tokie pat grafikai nubraižyti vietoje slenkstinės aktyvacijos funkcijos naudojant sigmoidinę (atitinkamai 3 ir 4 pav.) . Tiek paklaidos, tiek tikslumo tendencijos išlieka labai panašios.

3 pav. Paklaida pagal epochą, naudojant sigmoidinę aktyvacijos funkciją (pirmas duomenų rinkinys)



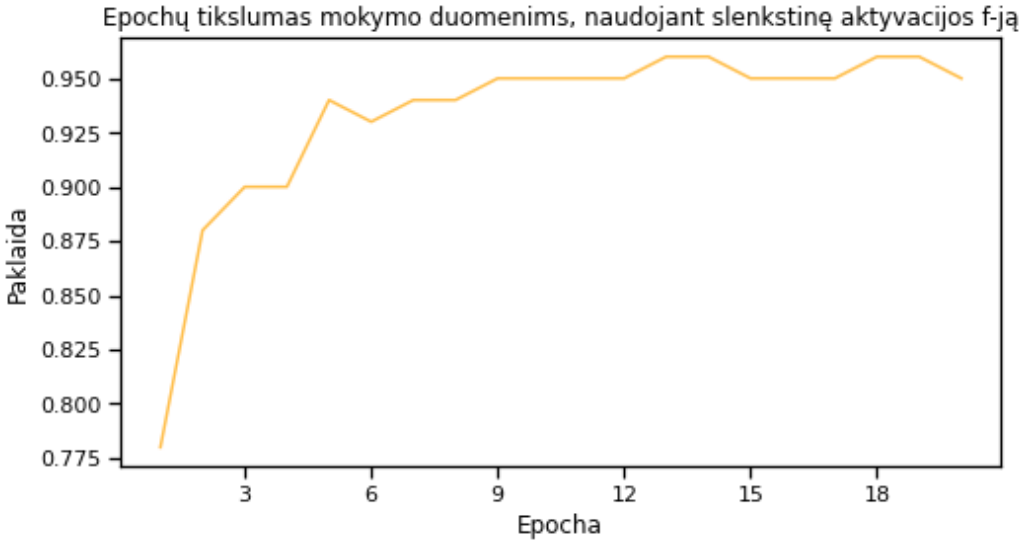
4 pav. Tikslumas pagal epochą, naudojant sigmoidinę aktyvacijos funkciją (pirmas duomenų rinkinys)

Analogiška procedūra kartota ir naudojant antrąjį duomenų rinkinį:

```
errors, _, weights = train_perceptron(X2_train, y2_train, threshold, 0.5, 20)

print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
threshold)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
threshold)[1])
Galutiniai svoriai:  [-202.0, 5.0, 12.0, 9.0, 12.0, 8.0, 7.0, 5.0, 5.0, 17.0]
Paklaida:  25.0
Tikslumas testavimo duomenims:  0.98
Paklaida testavimo duomenims:  3.0
```

Naudojant slenkstinę aktyvacijos funkciją paklaidos mažėjimas ir tikslumo didėjimas daug pastovesnis didėjant ep ochoms, lyginant su pirmu duomenų rinkiniu (5 ir 6 pav.). Taip pat galima matyti, kad paklaida stipriai mažėja iki 5-6 epochos, bet tolimesnėse epochose beveik išvis nustoja keistis.

5 pav. Paklaida pagal epochą, naudojant slenkstinę aktyvacijos funkciją (antras duomenų rinkinys)

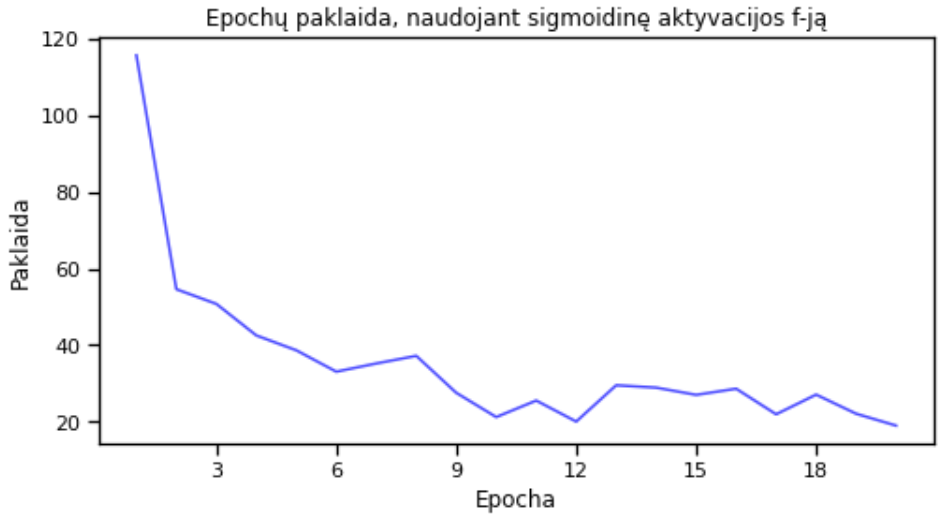


6 pav. Tikslumas pagal epochą, naudojant slenkstinę aktyvacijos funkciją (antras duomenų rinkinys)

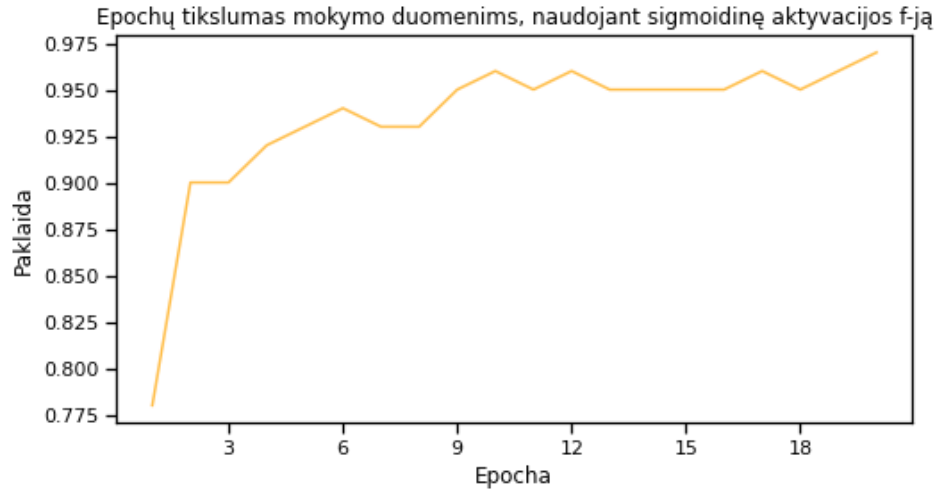
Lyginant slenkstinę ir sigmoidinę aktyvacijos funkcijas vėl gauname labai panašius rezultatus (7 ir 8 pav.).

```
errors, _, weights = train_perceptron(X2_train, y2_train, sigmoid, 0.5, 20)

print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
sigmoid)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
sigmoid)[1])
```

```
Galutiniai svoriai:  [-200.57, 3.72, 10.81, 15.86, 15.1, 4.6, 10.1, 3.15, -0.76, 18.6
]
Paklaida:  19.0
Tikslumas testavimo duomenims:  0.96
Paklaida testavimo duomenims:  5.0
```



7 pav. Paklaida pagal epochą, naudojant sigmoidinę aktyvacijos funkciją (antras duomenų rinkinys)



8 pav. Tikslumas pagal epochą, naudojant sigmoidinę aktyvacijos funkciją (antras duomenų rinkinys)

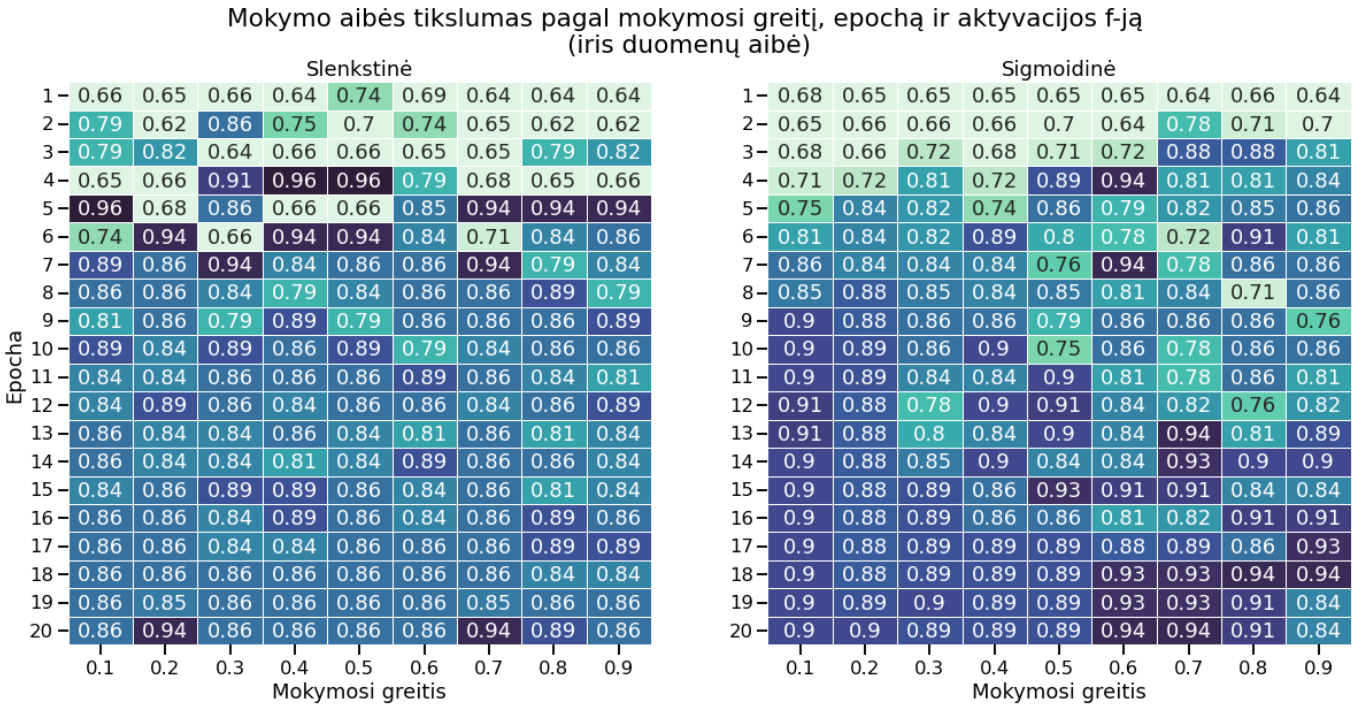
```
# grafikas, parodantys klasifikavimo tikslumo priklausomybę nuo mokymo epochos ir
mokymosi greičio parametro reikšmių
def plot_heatmap(data, ax, title, left=True):
    cmap = sns.cm.mako_r
    plot=sns.heatmap(data, vmax=1,vmin=0.7,center=0.85,cbar=False,cmap=cmap,
            linewidth=1,annot=True, ax=ax)
    plot.set_title(title)
    plot.set_xlabel("Mokymosi greitis")
```

```
if left:
    plot.set_ylabel("Epocha")
plot.set_xticklabels(lrates)
plot.set_yticklabels(range(1,results.shape[0]+1))
plot.tick_params(axis='x', rotation=0)
plot.tick_params(axis='y', rotation=360)
```

```
lrates = np.round(np.arange(0.1, 1, 0.1),1)
results1 = np.zeros((20,len(lrates)))
results2 = np.zeros((20,len(lrates)))
for i,j in enumerate(lrates):
    _, accuracies, __ = train_perceptron(X1_train, y1_train, threshold, j, 20,
plot=False)
    results1[:,i] = accuracies

    _, accuracies, __ = train_perceptron(X1_train, y1_train, sigmoid, j, 20,
plot=False)
    results2[:,i] = accuracies
```

```
sns.set_context("talk")
fig, ax = plt.subplots(1,2,figsize=(22, 9))
fig.suptitle("Tikslumas pagal mokymosi greitį ir aktyvacijos f-ją " + "(iris)")
plot_heatmap(results1,ax[0], "Slenkstinė")
plot_heatmap(results2,ax[1], "Sigmoidinė", left=False)
```

Rezultatų priklausomybė nuo epochos, mokymosi greičio ir naudojamos aktyvacijos funkcijos pavaizduota grafiškai (9 pav.). Kaip galime matyti, didžiausias tikslumas gaunamas, pavyzdžiui, naudojant slenkstinę aktyvacijos funkciją, 0,4 mokymosi greitį ir keturias epochas. Nepaisant to, apskritai matoma tendencija gauti geresnius rezultatus naudojant sigmoidinę aktyvacijos funkciją ir didesnį epochų kiekį.



9 pav. Mokymo aibės tikslumas pagal mokymosi greitį, epochą ir aktyvacijos f-ją (pirma duomenų aibė)

Nustačius geriausią variantą, išvedami galutiniai svoriai, tikslumas ir paklaida naudojant tiek mokymo duomenys, tiek testavimo duomenis:

```
errors, accuracy, weights = train_perceptron(X1_train, y1_train, threshold, 0.4, 4,
plot=False)

print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nTikslumas: ",  accuracy[-1],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
threshold)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X1_test, y1_test, weights,
threshold)[1])
```

```
Galutiniai svoriai:  [3.0, 5.4, 4.44, -7.6, -7.24]
Tikslumas:  0.96
Paklaida:  3
Tikslumas testavimo duomenims:  1.0
Paklaida testavimo duomenims:  0
```

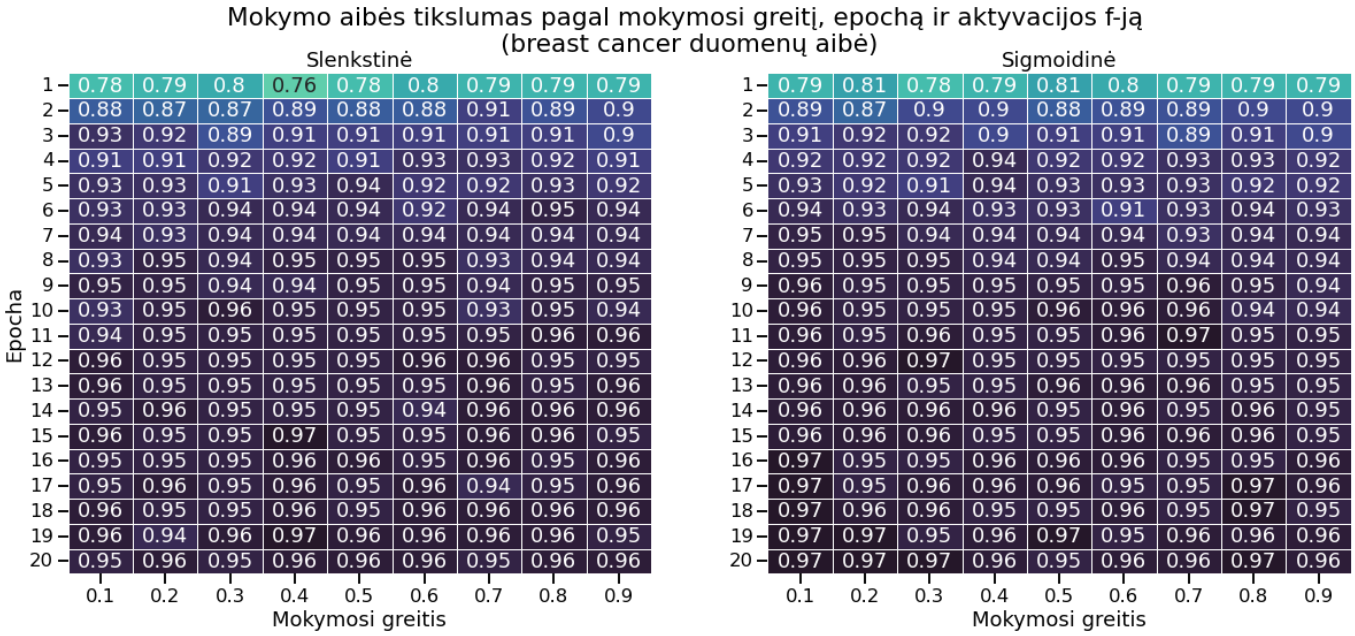
Taip pat pateikiama lentelė kokias klases kiekvienam testavimo aibės įrašui nustatė neuronas ir kokia klasė turėjo būti (1 priedas).

Tokia pati procedūra kartota ir naudojant antrąjį duomenų rinkinį:

```
lrates = np.round(np.arange(0.1, 1, 0.1),1)
results1 = np.zeros((20,len(lrates)))
results2 = np.zeros((20,len(lrates)))
for i,j in enumerate(lrates):
    _, accuracies, __ = train_perceptron(X2_train, y2_train, threshold, j, 20,
plot=False)
    results1[:,i] = accuracies

    _, accuracies, __ = train_perceptron(X2_train, y2_train, sigmoid, j, 20,
plot=False)
    results2[:,i] = accuracies



sns.set_context("talk")
fig, ax = plt.subplots(1,2,figsize=(22, 9))
plot_heatmap(results1,ax[0], "Slenkstinė")
plot_heatmap(results2,ax[1], "Sigmoidinė", left=False)
```

Tokio pat tipo grafikas nubraižytas ir naudojant antrą duomenų rinkinį (10 pav.). Geriausi rezultatai gaunami naudojant 0,7 mokymosi greitį su 11 epochų ir sigmoidine aktyvacijos funkcija. Matoma tendencija mokymo aibės tikslumui didėti didėjant epochų skaičiui, tačiau šiam duomenų rinkinius vos po kelių epochų pasiekiamas labai

didelis tikslumas ir rezultatai nustoja gerėti. Taip pat galime pastebėti, kad naudojant antrąjį duomenų rinkinį įprastai gaunamas geresnis klasifikavimo tikslumas, lyginant su rezultatais pirmajam duomenų rinkiniui.



10 pav. Mokymo aibės tikslumas pagal mokymosi greitį, epochą ir aktyvacijos f-ją (antra duomenų aibė)

```
errors, accuracy, weights = train_perceptron(X2_train, y2_train, sigmoid, 0.7, 11,
plot=False)

print("Galutiniai svoriai: ", [round(i,2) for i in weights],
      "\nTikslumas: ",  accuracy[-1],
      "\nPaklaida: ",  errors[-1],
      "\nTikslumas testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
sigmoid)[0],
      "\nPaklaida testavimo duomenims: " ,  test_accuracy(X2_test, y2_test, weights,
sigmoid)[1])
```

```
Galutiniai svoriai:  [-111.46, -0.98, 10.91, 10.45, 11.23, -0.08, 8.59, 0.08, 1.2, 4.
07]
Tikslumas:  0.97
Paklaida:  18.47
Tikslumas testavimo duomenims:  0.96
Paklaida testavimo duomenims:  5.0
```

Antram duomenų rinkiniui taip pat pateiktas tikrų ir perceptrono pateiktų reikšmių palyginimas naudojant testavimo aibę (2 priedas).

4 Išvados

Daugeliu atveju gautas didesnis klasifikavimo tikslumas didėjant epochoms. Nepaisant to, perceptronas per gana mažą epochų skaičių (didžiajai daliai atvejų nedaugiau nei per 6), pasiekia tikslumą, kuris (beveik) nebegerėja tolinesnių epochų metu.

Taip pat pastebėta problema galinti kilti jeigu parenkamas per didelis mokymosi greitis: svoriai atnaujinami per daug greitai ir tolimesnėje epochoje gauti rezultatai yra prastesni negu epochose prieš tai.

Rasta ir naudojamo duomenų rinkinio įtaka.

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

Taip pat per didelis mokymosi greitis gali reikši kad svoriai šokinėja per dideliais tarpais ir nebus surandami optimalus svoriai, todėl geriau rinktis mažesnį mokymosį greiti jeigu yra tam resursų.

Didžiausia itaką šios užduoties rezultatams darė duomenų rinkinio dydis, nes kaip matėme su pirmais duomenim nesikeičia tikslumas, o su antrais duomenim labai šokinėja, tad pagrindinis kriterijus kuriant neuroninius tinklus yra turėti kuo įmanoma daugiau duomenų.

Priedas

Tikra reikšmė	Prognozuota reikšmė
0	0
0	0
0	0
0	0
0	0
1	1
1	1
0	0
0	0
1	1
1	1
1	1
0	0
1	1
1	1
1	1
0	0
1	1
0	0
1	1

1 priedas Tikros ir prognozuotos klasių reikšmės testavimo duomenims (pirma duomenų aibė)

Tikra reikšmė	Prognozuota reikšmė
0	0
0	1
1	1
0	1
0	0
1	1
1	1
0	0
1	1
1	1
0	0
0	0
0	0

1	1
0	0
0	0
0	0
1	1
0	0
0	1
1	1
1	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	1
0	0
0	0
0	0
0	0
1	1
1	1
0	0
0	0
0	0
1	1
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	1
0	0
0	0
0	0
1	1
0	0
0	0
1	1
1	1
1	1
1	1
1	0
0	0
0	0
0	0

1	1
0	0
0	0
0	0
0	0
0	0
1	1
0	0
1	1
1	1
0	0
0	0
1	1
0	0
1	1
1	1
1	1
1	1
0	0
0	0
0	0
0	0
0	0
0	0
1	1
0	0
1	1
0	0
0	0
1	1
0	0
1	1
0	0
0	0
1	1
1	1
1	1
0	0
0	0
0	0
0	0
0	0
0	0
1	1
1	1
1	1
0	0
0	0
1	1
0	0

1	1
0	0
0	0
0	0
0	0
0	1
0	0
0	0
1	1
1	1
1	1
1	1
0	0
0	0
0	0
0	0
1	1
0	0
0	0
1	1
0	0
1	1
0	0
0	0

2 priedas Tikros ir prognozuotos klasių reikšmės testavimo duomenims (antra duomenų aibė)