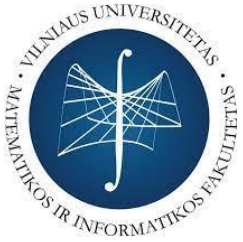


VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

DUOMENŲ MOKSLO BAKALAURO STUDIJOS

Vieno neurono (perceptrono) mokymas sprendžiant klasifikavimo uždavinį
Single Neuron (Perceptron) Classification Training

tiriamasis darbas

Atliko: Vainius Gataveckas

VU el.p.: vainius.gataveckas@mif.stud.vu.lt

Vertintojas: Dr. Viktor Medvedev

Vilnius

2021

Darbo tikslas – apmokyti vieną neuroną spręsti dviejų klasių uždavinį, atlikti tyrimą su dviem

duomenų aibėm.

Neurono mokymo procesas yra optimizavimo uždavinys. Pasirenkama paklaidos funkcija $E(w)$ tokia, kad ją minimizuojant būtų randami tokie svoriai w tokie, kad neuronas teisingai suklasifikuotų duomenis x. Ši funkcija dar vadinama tikslo funkcija. Minimizuojama tikslo funkcija būtinai turi turėti išvestinę. Vienas iš būdų mokyti neuroną yra „ADELINE“ algoritmas. Pasirenkama tikslo funkcija $E(w) = \frac{1}{2}\sum_{i=1}^{m}(t_i - y_i)^2$. Šios funkcijos išvestinė $(t_i - y_i)(-x_{ik})$, kai $y_i = f(\sum_{k=0}^{n} w_k x_{ik})$. Bendra mokymo taisyklė:

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

Kur η yra mokymo greičio hiperparametras.

Tikslo įgyvendinimui naudojamos dvi duomenų aibės. Irisų[1] ir krūties vėžio pacientų diagnostikos[2] duomenų aibė.

Darbo uždaviniai:

1. Paruošti duomenų rinkinius modelio taikymui
2. Programiškai realizuoti „ADELINE“ neurono modelį.
3. Pritaikyti modelį duomenų rinkinių klasifikavimo uždaviniui.
4. Atlikti hiperparametrų, paklaidos funkcijos ir klasifikavimo tikslumo analizę.

Darbo tikslų įgyvendinimui bus pasitelkiama programavimo kalba „Python“.

1. Duomenys.

Irisų duomenų aibė sumažinima iki dviejų klasių: „Versicolor" ir „Virginica". Suteikiamos žymės „1" ir „0" atitinkamai. Vėžio duomenų aibėje paciento būklė koduojamos „2" ir „4", kur „2" reiškia nepiktybinį naviką, o „4" reiškia piktybinį. Šios klasės žymės taip pakeičiamos „0/1" atitinkamai. Duomenyse taip pat pašalinami įrašai, kurie savo požymių aibėje turi pažymėjimą nežinoma reikšmė: „?".

Irisų duomenų aibėje naudojami 4 požymiai, o krūties vežio duomenyse 9 požymiai. Atitinkamai suformuojami neurono modeliai su 5 ir 10 įėjimų (papildomas įėjimas skiriamas poslinkiui).

Kode duomenų rinkiniai paverčiami „Numpy Array" duomenų struktūra ir priskiriami kintamiesiems: „d_iris" ir „d_cancer"(1 kodo fragmentas). Naudojama mokymo ir testavimo aibių „80:20" paskirstymo strategija, t.y. 80% duomenų skiriama mokymo 20% testavimo aibei. Sukuriami abiejų duomenų aibių mokymo ir testavimo rinkiniai su kintamojo prierašu „_training" ir „_test".

1 kodo fragmentas. Duomenų nuskaitymas ir pertvarkymas.

```
d_iris = pd.read_csv("iris.data.csv", header=None).to_numpy()
d_cancer = pd.read_csv("breast-cancer-
           wisconsin.data.csv",header=None).to_numpy()
d_cancer = np.delete(d_cancer, 0, 1)
d_cancer[:, -1] = (d_cancer[:, -1] > 3).astype(int)
d_cancer = d_cancer[np.all(d_cancer != "?", axis=1), :]

d_iris = d_iris[d_iris[:,-1]!='Iris-setosa']
d_iris[:, -1] = (d_iris[:, -1] != 'Iris-versicolor').astype(int)

np.random.shuffle(d_iris)
d_iris_training, d_iris_test = d_iris[:80,:], d_iris[80:,:]
np.random.shuffle(d_cancer)
d_cancer_training, d_cancer_test = d_cancer[:80,:], d_cancer[80:,:]
```

2. Neurono modelio realizacija.

Modelyje apibrėžtos dvi „Python“ klasės. Įėjimų (svorių) klasė ir pačio neurono klasė. Įėjimų klasė atlieka svorio ir įėjimo duomenų sandaugą, svorių atnaujinimo funkcijas(1 kodo fragmentas). Jos iniciavimo metu sukuriama atsitiktinė pradinė neurono svorio reikšmė iš intervalo [0; 1].

1 kodo fragmentas. Įėjimo klasės struktūra.

```
class Iejimas:
  def __init__(self):
    self.svoris = random.uniform(0,1)
  def pakeisti_svori(self, naujas_svoris):
    self.svoris = naujas_svoris
    return self
  def w_x(self, x):
    return self.svoris*x
```

Neuronas realizuojamas kaip klasinis komponentas, kuris savyje turi mokymo funkcijas ir klasifikavimo rezultato funkcijas(2 kodo fragmentas). Jo iniciavimo metu suteikiamas įėjimų skaičius bei mokymo greitis. Reikšmės skaičiavimui yra 3 funkcijos „grazinti_a“, „slenkstine_aktivacija“ ir „sigmoidine_aktivacija“. Pastarosios dvi yra aktyvacijos funkcijos (be apvalinimo sigmoidinėje funkcijoje). Pirmoji kviečia Įėjimo klasės funkciją „w_x“, kad suskaičiuotų aktyvacijos funkcijų argumentą a.

2 kodo fragmentas. Neurono klasė.

```
class Neuronas:
  def __init__(self, iejimai, mokymo_greitis):
    self.iejimai = [Iejimas() for x in range(iejimai)]
    self.mokymo_greitis = mokymo_greitis
  def grazinti_a(self, duomenys):
    a = 0 + self.iejimai[0].svoris
    for i in range(len(duomenys)):
      a += self.iejimai[i+1].w_x(duomenys[i])
    return a

  def slenkstine_aktivacija(self,a):
    if(a<0):
      return 0
    elif(a>=0):
      return 1

  def sigmoidine_aktivacija(self,a):
    return 1/(1+math.exp(-a))

  def mokymo_epocha(self, duomenys,klases, funkcija):
    duomenu_skaicius = duomenys.shape[0]
```

```
    teisingai_klasifikuota = 0
    paklaida = 0
    for i in range(duomenu_skaicius):
      klase = klases[i][0]
      a = self.grazinti_a(duomenys[i,])
      if(funkcija == "sig"):
        y = self.sigmoidine_aktivacija(a)
        paklaida += (klase-y)**2
        if(round(y,0) == klase):
          teisingai_klasifikuota+=1
      elif(funkcija == "slen"):
        y = self.slenkstine_aktivacija(a)
        paklaida += (klase-y)**2
        if(y == klase):
          teisingai_klasifikuota+=1
      for ind,w in enumerate(self.iejimai):
        if(ind==0):
          self.iejimai[ind] = w.pakeisti_svori(w.svoris + self.mokymo_grei
tis*(klase-y)*1)
        else:
          self.iejimai[ind] = w.pakeisti_svori(w.svoris + self.mokymo_grei
tis*(klase-y)*duomenys[i,ind-1])
    return teisingai_klasifikuota/duomenu_skaicius, paklaida

  def klasifikavimas_diag(self, duomenys, klases, funkcija):
    duomenu_skaicius = duomenys.shape[0]
    teisingai_klasifikuota = 0
    paklaida = 0
    for i in range(duomenu_skaicius):
      klase = klases[i][0]
      a = self.grazinti_a(duomenys[i,])
      if(funkcija == "sig"):
        y = self.sigmoidine_aktivacija(a)
        paklaida += (klase-y)**2
        if(round(y,0) == klase):
          teisingai_klasifikuota+=1
      elif(funkcija == "slen"):
        y = self.slenkstine_aktivacija(a)
        paklaida += (klase-y)**2
        if(y == klase):
          teisingai_klasifikuota+=1
    return teisingai_klasifikuota/duomenu_skaicius, paklaida

  def klasifikavimas(self, duomenys, funkcija):
    a = self.grazinti_a(duomenys)
```

```
if(funkcija == "sig"):
  y = round(self.sigmoidine_aktivacija(a),0)
elif(funkcija == "slen"):
  y = self.slenkstine_aktivacija(a)
return y
```

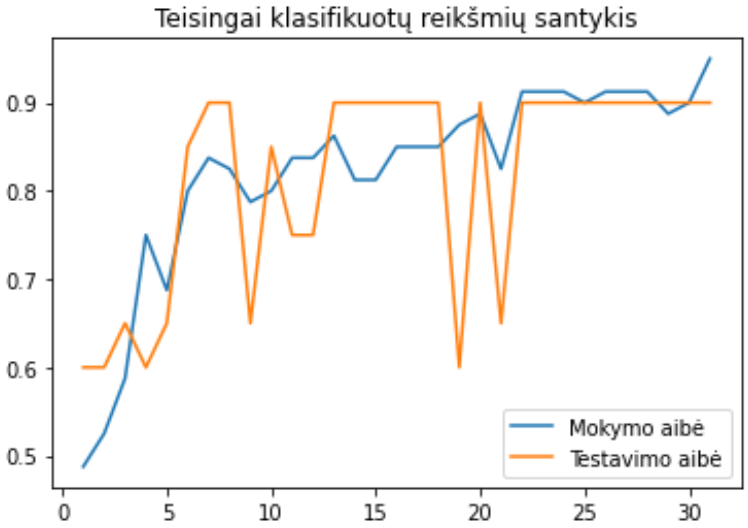
Funkcija „mokymo_epocha” yra pagrindinė neurono mokymosi funkcija, kuri realizuoja vieną mokymosi epochą, t.y. iteruojami visi mokymosi duomenys. Kintamasis „teisingai_klasifikuota“ seka, kiek duomenų buvo klasifikuoti teisingai, vėliau tai naudojama teisingų ir visų duomenų santykiui gauti. „paklaida“ sumuoja kvadratinį aktyvacijos funkcijos reikšmės ir tikrosios klasės skirtumą. Mokymo metu duodamas parametras „funkcija“, kurios reikšmės gali būti „sig“ – naudoti sigmoidinę arba „slen“ – naudoti slenkstinę aktyvacijos funkciją. Šio metodo pabaigoje atnaujinami įėjimo objektų svoriai pagal „ADELINE“ algoritmo paklaidos funkciją. Pati mokymo funkcija gražina klasifikavimo tikslumo santykį ir suminę paklaidą.

Funkcija „klasifikavimas_diag“ naudojama gauti minėtus modelio tinkamumo parametrus, tačiau neturi mokymosi funkcijos. Ji priima duomenis ir jų tikrąsias žymias bei aktyvacijos funkciją. „klasifikavimas“ yra papildoma funkcija kuri leidžia gauti vieno įrašo žymę. Ši funkcija analizėje nenaudojama.

3. Realių duomenų klasifikavimas.

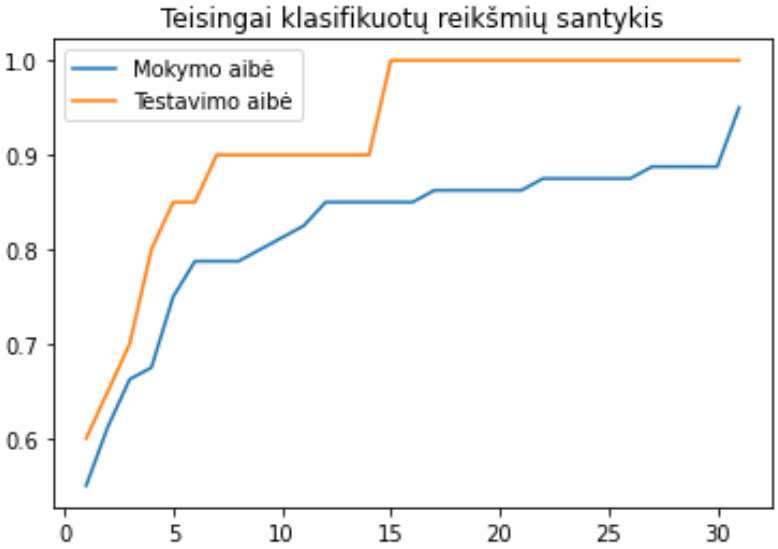
3.1 Irisų duomenų rinkinys.

Naudojant sigmoidinę aktyvacijos funkciją su mokymosi greičiu 0,5 gauta testavimo aibės klasifikavimo tikslumo tendencija stipriai skirtis tarp epochų (1 paveikslas). Tai gali sukelti per didelis mokymosi greitis.



1 pav. Tikslumo santykis $\eta = 0,5$.

Pakartojus eksperimentą su mažesniu $\eta = 0{,}1$ pasiekiamas tobulas klasifikavimo tikslumo santykis, tačiau vėlesnėje 15-toje epochoje (2 paveikslas). Iš to darome išvadą, kad tam tikru atveju modelio tikslumas priklauso nuo mokymo trukmės. Optimalaus mokymo greičio ir epochų skaičiaus santykis nustatomas eksperimentiškai.



2 pav. Tikslumo santykis $\eta = 0,1$.

Svoriai:

w_0 = 0.05
w_1 = -1.27
w_2 = -1.29
w_3 = 1.60
w_4 = 2.15

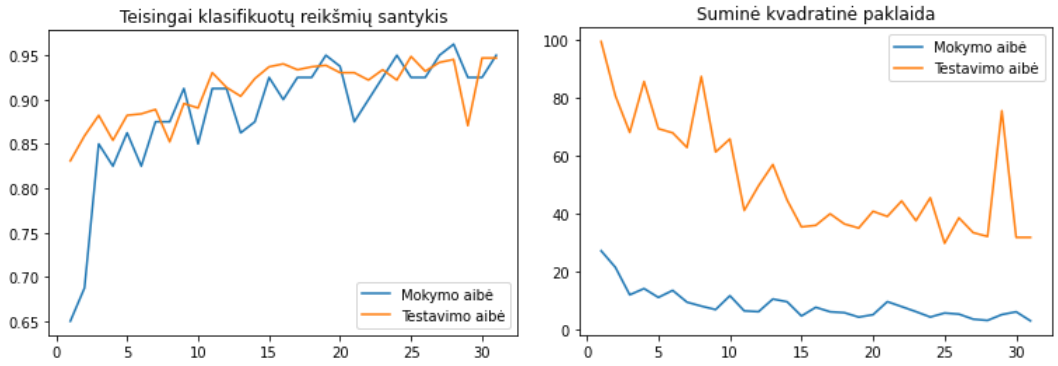
1 Lentelė. Mokymo greičio ir epochų skaičiaus priklausomybė testavimo aibės klasifikavimo tikslumui. Irisų duomenys.

Epochų skaičius	**Mokymo greitis**			
	1	**0,5**	**0,1**	**0,01**
10	**0,85**	**0,8**	**0,8**	**0,75**
20	**0,6**	**0,8**	**0,85**	**0,8**
30	**0,85**	**0,8**	**0,9**	**0,8**

Mokymo greičio ir epochų skaičius daro įtaką klasifikavimo tikslumui (1 lentelė). Mažam mokymo greičiui reikalinga daugiau epochų. Apsiribojant 30 epochų geriausią klasifikavimo tikslumą duoda $\eta = 0{,}1$, kas ir buvo pastebėta antrajame eksperimente.

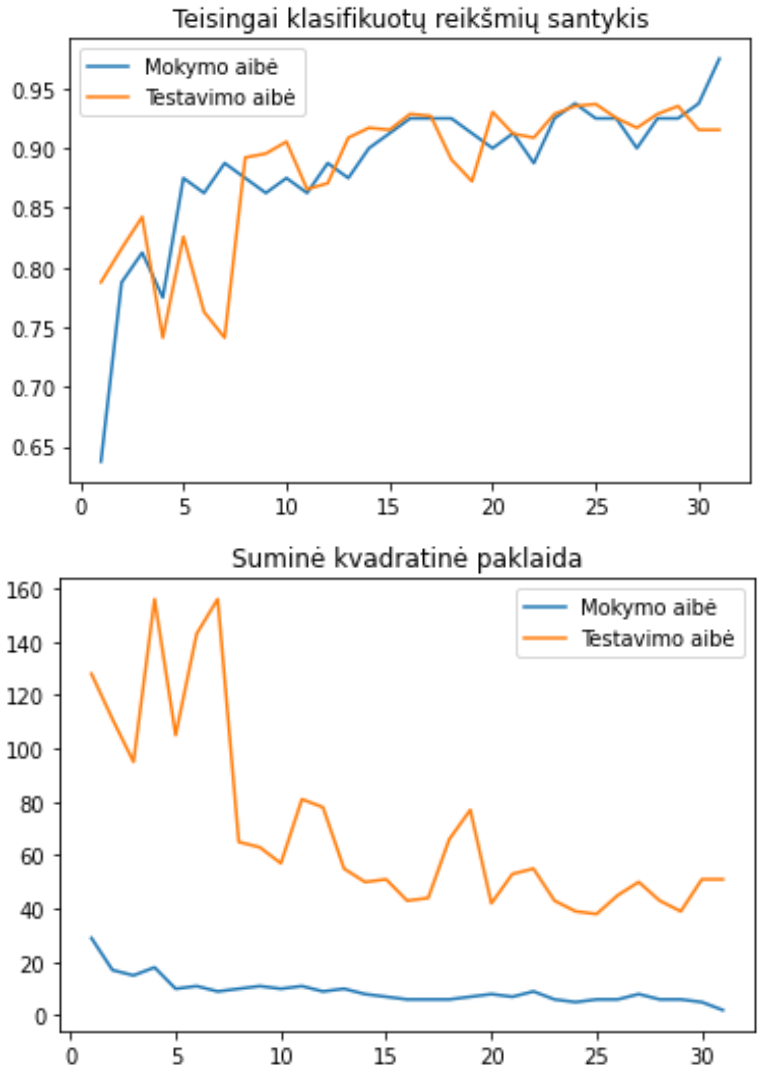
3.1 Vėžio pacientų duomenų rinkinys.

Naudojant sigmoidinę aktyvacijos funkciją su mokymosi greičiu 0,5 pastebima, kad didesnis mokymosi greitis daro mažesnę įtaką staigiems klasifikavimo santykio svyravimams tarp epochų(3 paveikslas).



3 pav. Tikslumo santykis $\eta = 0,5$. Suminės paklaidos grafikas. Sigmoidinė aktyvacija.

Palyginus sigmoidinę ir slenkstinę aktyvaciją testavimo duomenims pastebima, kad suminė kvadratinė paklaida slenkstinės atveju yra nežymiai didesnė(4 paveikslas), nors klasifikavimo tikslumas vėlesnėse epochose yra panašus (pradiniai svoriai atsitiktiniai ir skiriasi skirtingoms aktyvacijos funkcijoms). Taip yra todėl, kad skaičiuojant paklaidą sigmoidinės funkcijos reikšmė nėra apvalinama.



4 pav. Tikslumo santykis $\eta = 0,5$. Suminės paklaidos grafikas. Slenkstinė aktyvacija.

Geriausias klasifikavimo santykis 0,93 naudojant 30 epochų buvo pasiektas su $\eta = 0.01$. Su svoriais:

w_0 = -51.18
w_1 = -4.51
w_2 = 3.28
w_3 = 8.57
w_4 = -4.97
w_5 = -5.95
w_6 = 6.45
w_7 = 4.59
w_8 = 8.75
w_9 = -0.93

Bendruoju atveju modelis geriau prisitaiko prie vėžio pacientų duomenų, nei prie irisų. Nepaisant epochų skaičiaus ir mokymo greičio modelis pasiekia minimo tašką su funkcijos reikšme ~0.92. Su $\eta = 0.01$ – maža mokymo greičio reikšme, taip pat matoma tokia pati tendencija, kaip ir irisų duomenims (2 lentelė).

2 Lentelė. Mokymo greičio ir epochų skaičiaus priklausomybė testavimo aibės klasifikavimo tikslumui. Vežio duomenys.

Epochų skaičius	Mokymo greitis			
	1	**0,5**	**0,1**	**0,01**
10	**0.92**	**0.92**	**0.91**	**0.87**
20	**0.93**	**0.91**	**0.93**	**0.90**
30	**0.92**	**0.91**	**0.93**	**0.93**

Išvados.

Skirtingiems duomenims su skirtingais mokymo greičio ir epochų skaičiaus parametrais gaunami skirtingi rezultatai. Egzistuoja priklausomybė tarp mokymo greičio ir epochų skaičiaus klasifikavimo tikslumo prasme. Kuo didesnis mokymosi greitis, tuo mažiau epochų reikia, kad būtų pasiektas neblogas klasifikavimo tikslumas. Tačiau, esant dideliam mokymosi greičiui yra tikimybė nepasiekti tikslo funkcijos minimumą turinčių svorių reikšmių. Irisų duomenų rinkiniui didelis mokymosi greitis darė didesnę įtaką modelio tikslumo reikšmių skirtumui tarp epochų. Tuo tarpu vėžio pacientų duomenims didesnis mokymosi greitis leido gauti stabilius rezultatus per nedidelį mokymosi epochų skaičių. Slenkstinės aktyvacijos paklaidos matas tam pačiam klasifikavimo tikslumui yra nežymiai didesnis (~10 % skirtumas).

Šaltiniai.

Dua, D. and Graff, C. (2019). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer:

https://archive.ics.uci.edu/ml/datasets/iris [1]

https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic) [2]

Naudoti „Python“ paketai.

math – matematinių funkcijų skaičiavimo paketas.

random – pseudo atsitiktinių skaičių generavimo paketas.

numpy – matricinių duomenų struktūrų modeliavimo paketas.

matplotlib – duomenų vizualizavimo paketas.

pandas – naudota duomenų nuskaitymui ir išankstiniai duomenų peržiūrai.

1 priedas. Pilnas programos kodas.

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


d_iris = pd.read_csv("/content/sample_data/iris.data.csv", header=None).to
_numpy()
d_cancer = pd.read_csv("/content/sample_data/breast-cancer-
wisconsin.data.csv",header=None).to_numpy()


d_cancer = np.delete(d_cancer, 0, 1)
d_cancer[:, -1] = (d_cancer[:, -1] > 3).astype(int)
d_cancer = d_cancer[np.all(d_cancer != "?", axis=1), :]
d_cancer = d_cancer.astype(float)
d_iris = d_iris[d_iris[:,-1]!='Iris-setosa']
d_iris[:, -1] = (d_iris[:, -1] != 'Iris-versicolor').astype(int)

np.random.shuffle(d_iris)
d_iris_training, d_iris_test = d_iris[:80,:], d_iris[80:,:]
np.random.shuffle(d_cancer)
d_cancer_training, d_cancer_test = d_cancer[:80,:], d_cancer[80:,:]

class Iejimas:
  def __init__(self):
    self.svoris = random.uniform(0,1)
  def pakeisti_svori(self, naujas_svoris):
    self.svoris = naujas_svoris
    return self
  def w_x(self, x):
```

```
    return self.svoris*x

class Neuronas:
  def __init__(self, iejimai, mokymo_greitis):
    self.iejimai = [Iejimas() for x in range(iejimai)]
    self.mokymo_greitis = mokymo_greitis
  def grazinti_a(self, duomenys):
    a = 0 + self.iejimai[0].svoris
    for i in range(len(duomenys)):
      a += self.iejimai[i+1].w_x(duomenys[i])
    return a

  def slenkstine_aktivacija(self,a):
    if(a<0):
      return 0
    elif(a>=0):
      return 1

  def sigmoidine_aktivacija(self,a):
    return 1/(1+math.exp(-a))

  def mokymo_epocha(self, duomenys,klases, funkcija):
    duomenu_skaicius = duomenys.shape[0]
    teisingai_klasifikuota = 0
    paklaida = 0
    for i in range(duomenu_skaicius):
      klase = klases[i][0]
      a = self.grazinti_a(duomenys[i,])
      if(funkcija == "sig"):
        y = self.sigmoidine_aktivacija(a)
        paklaida += (klase-y)**2
        if(round(y,0) == klase):
          teisingai_klasifikuota+=1
      elif(funkcija == "slen"):
        y = self.slenkstine_aktivacija(a)
        paklaida += (klase-y)**2
        if(y == klase):
          teisingai_klasifikuota+=1
      for ind,w in enumerate(self.iejimai):
        if(ind==0):
          self.iejimai[ind] = w.pakeisti_svori(w.svoris + self.mokymo_grei
tis*(klase-y)*1)
        else:
          self.iejimai[ind] = w.pakeisti_svori(w.svoris + self.mokymo_grei
tis*(klase-y)*duomenys[i,ind-1])
```

```
    return teisingai_klasifikuota/duomenu_skaicius, paklaida

  def klasifikavimas_diag(self, duomenys, klases, funkcija):
    duomenu_skaicius = duomenys.shape[0]
    teisingai_klasifikuota = 0
    paklaida = 0
    for i in range(duomenu_skaicius):
      klase = klases[i][0]
      a = self.grazinti_a(duomenys[i,])
      if(funkcija == "sig"):
        y = self.sigmoidine_aktivacija(a)
        paklaida += (klase-y)**2
        if(round(y,0) == klase):
          teisingai_klasifikuota+=1
      elif(funkcija == "slen"):
        y = self.slenkstine_aktivacija(a)
        paklaida += (klase-y)**2
        if(y == klase):
          teisingai_klasifikuota+=1
    return teisingai_klasifikuota/duomenu_skaicius, paklaida

  def klasifikavimas(self, duomenys, funkcija):
    a = self.grazinti_a(duomenys)
    if(funkcija == "sig"):
      y = round(self.sigmoidine_aktivacija(a),0)
    elif(funkcija == "slen"):
      y = self.slenkstine_aktivacija(a)
    return y

neuronas_iris = Neuronas(5,0.01)

funkcija = "sig"
epochos = 30
mokymo_duomenys = d_iris_training[:,:-1]
mokymo_klases = d_iris_training[:,-1:]

testavimo_duomenys = d_iris_test[:,:-1]
testavimo_klases = d_iris_test[:,-1:]

santykiai_train = []
se_train = []
santykiai_test = []
se_test = []

for epocha in range(epochos):
```

```
  santykis, se = neuronas_iris.mokymo_epocha(mokymo_duomenys, mokymo_klase
s, funkcija)
  santykiai_train.append(santykis)
  se_train.append(se)
  santykis, se = neuronas_iris.klasifikavimas_diag(testavimo_duomenys,test
avimo_klases, funkcija)
  santykiai_test.append(santykis)
  se_test.append(se)

santykis, se = neuronas_iris.klasifikavimas_diag(mokymo_duomenys, mokymo_k
lases, funkcija)
santykiai_train.append(santykis)
se_train.append(se)
santykis, se = neuronas_iris.klasifikavimas_diag(testavimo_duomenys,testav
imo_klases, funkcija)
santykiai_test.append(santykis)
se_test.append(se)


x = [x for x in range(1, epochos+2)]
plt.plot(x, santykiai_train, label = "Mokymo aibė")
plt.plot(x, santykiai_test, label = "Testavimo aibė")
plt.title("Teisingai klasifikuotų reikšmių santykis")
plt.legend()
plt.show()

plt.plot(x, se_train, label = "Mokymo aibė")
plt.plot(x, se_test, label = "Testavimo aibė")
plt.title("Suminė kvadratinė paklaida")
plt.legend()
plt.show()

print("svoriai:")
for idx,w in enumerate(neuronas_iris.iejimai):
  print(f"w_{idx} = {w.svoris}")


neuronas_cancer = Neuronas(10,0.01)

funkcija = "sig"
epochos = 30
mokymo_duomenys = d_cancer_training[:,:-1]
mokymo_klases = d_cancer_training[:,-1:]

testavimo_duomenys = d_cancer_test[:,:-1]
```

```
testavimo_klases = d_cancer_test[:,-1:]

santykiai_train = []
se_train = []
santykiai_test = []
se_test = []

for epocha in range(epochos):
  santykis, se = neuronas_cancer.mokymo_epocha(mokymo_duomenys, mokymo_kla
ses, funkcija)
  santykiai_train.append(santykis)
  se_train.append(se)
  santykis, se = neuronas_cancer.klasifikavimas_diag(testavimo_duomenys,te
stavimo_klases, funkcija)
  santykiai_test.append(santykis)
  se_test.append(se)

santykis, se = neuronas_cancer.klasifikavimas_diag(mokymo_duomenys, mokymo
_klases, funkcija)
santykiai_train.append(santykis)
se_train.append(se)
santykis, se = neuronas_cancer.klasifikavimas_diag(testavimo_duomenys,test
avimo_klases, funkcija)
santykiai_test.append(santykis)
se_test.append(se)


x = [x for x in range(1, epochos+2)]
plt.plot(x, santykiai_train, label = "Mokymo aibė")
plt.plot(x, santykiai_test, label = "Testavimo aibė")
plt.title("Teisingai klasifikuotų reikšmių santykis")
plt.legend()
plt.show()

plt.plot(x, se_train, label = "Mokymo aibė")
plt.plot(x, se_test, label = "Testavimo aibė")
plt.title("Suminė kvadratinė paklaida")
plt.legend()
plt.show()

print("svoriai:")
for idx,w in enumerate(neuronas_iris.iejimai):
  print(f"w_{idx} = {w.svoris}")
```