VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

Vieno neurono mokymas sprendžiant klasifikavimo uždavinį

2-oji skaitmeninio intelekto ir sprendimų priėmimų dalyko užduotis

Atliko: 4 kurso 5 grupės studentė
Gabrielė Žielytė (parašas)

Darbo vadovas: Prof., Dr. Olga Kurasova (parašas)

Vilnius – 2020

TURINYS

Tikslas

Šio darbo tikslas yra:

1. Sukurti programą, kuri apmokytų vieną neuroną (perceptroną) spręsti nesudėtingą klasifikavimo uždavinį.

2. Atlikti tyrimą naudojant du duomenų rinkinius (aprašytus skyrelyje „Duomenys”), kiekvieno rezultatus pateikiant lentelėse arba grafikuose su atitinkamais komentarais (rezultatai vertinami klasifikavimo tikslumo mato prasme).

1. Duomenys

Naudojami irisų duomenys (`https://archive.ics.uci.edu/ml/datasets/iris`), tačiau jie yra trijų klasių: Setosa, Versicolor ir Virginica, todėl reikia pasidaryti du duomenų rinkinius:

1. vieną klasę sudaro Setosa rūšis (50 duomenų įrašų), kitą klasę – Versicolor ir Virginica rūšys (100 duomenų įrašų).

2. vieną klasę sudaro Versicolor rūšis (50 duomenų įrašų), kitą klasę – Virginica rūšys (50 duomenų įrašų).

Klasių žymės (label) turi būti 0 arba 1.

Pradiniai svoriai pasirinkti nuliniai, o nuliniai įėjimai– 1.

1-ąjam duomenų rinkiniui iš viso skirta 150 duomenų įrašų: Setosa ir Versicolor, bei Virginica. 0 klasė priklauso Setosa duomenims, 1 – Versicolor ir Virginica duomenims.

2-ąjam duomenų rinkiniui iš viso skirta 100 duomenų įrašų: Versicolor ir Virginica. 0 klasė priklauso Versicolor duomenims, 1 – Virginica duomenims.

Kiekvienas duomenų vektorius turi 5 stulpelius(požymius): nulinį įėjimą ir 4 faile įrašytus požymius.

Pirmojo duomenų rinkinio mokymui skirta 120 duomenų įrašų (40 Setosa ir 80 Versicolor/Virginica), testavimui - 30 įrašų (10 Setosa, 20 Versicolor/Virginica).

Antrojo duomenų rinkinio mokymui skirta 80 duomenų įrašų (po tiek pat iš abiejų klasių) ir 20 įrašų testavimui.

1.1. Apibrėžimai

Epocha – neuronų mokymo proceso dalis, kurios metu apdorojamas **visas** įėjimų vektorių rinkinys vieną kartą.

Iteracija – tai neuronų mokymo proceso dalis, kurios metu apdorojamas **vienas** įėjimų vektorius.

Vienos mokymo epochos metu įvyksta tiek iteracijų, kiek yra įėjimo vektorių

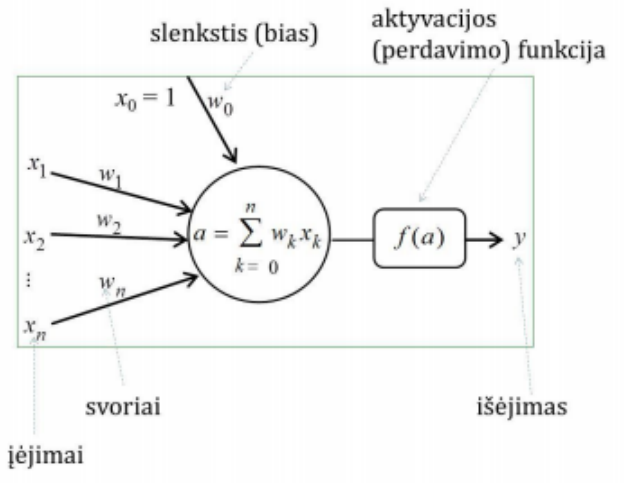
2. Dirbtinis neuronas

Dirbtinio neurono modelyje (1 pav.) pažymėtos įėjimo reikšmės, slenkstis, pradiniai svoriai, bei norimos išėjimo reikšmės įvedamos programos pagrindinėje dalyje. Vartotojas įveda norimą aktyvacijos funkciją: 0 – slenkstinei funkcijai, 1 – sigmoidinei.

Išėjimo reikšmės paskaičiuojamos ”calculateOutput” dalyje: suskaičiuojama a = įėjimo reikšmių ir svorių sandaugų suma, o tuomet reikšmė a įstatoma į pasirinktą aktyvacijos funkciją.

Tinkamų svorių radimui (perceptrono mokymui) vartotojas parenka mokymo greitį (angl. *learning rate*) ir iteracijų skaičių.

Naujasis svoris apskaičiuojamas prie senojo pridedant mokymo greičio, paklaidos, bei įėjimo reikšmės sandaugą.



1 pav. Dirbtinio neurono modelis

Kodas reikalingas dirbtinio neurono modelio sukūrimui:

```
import numpy as np


class Neuronas:
    def __init__(self, data, weights, data_outputs):
        self.data = data
        self.data_outputs = data_outputs
        self.weights = weights

# slenkstinė funkcija
    def slenkstine(self, a):
        return 1 if a > 0 else 0

# sigmoidinė funkcija
    def sigmoidine(self, a):
        return 1 / (1 + np.exp(-a))

# įėjimo reikšmių ir svorių sandaugų suma
```

```
    def suma(self, data):
        a = np.dot(data, self.weights)
        return a
# skaiciuojama isejimo reiksme
    def calculateOutput(self, data, funkcija):
        a = self.suma(data)
        if funkcija == 0:
            return self.slenkstine(a)
        else:
             return self.sigmoidine(a)[0]
```

Kodo dalis dirbtinio neurono apmokymui:

```
# mokymo paklaida ir naujų svorių gavimas
    def paklaida(self, y, i, l_rate):
        error = self.data_outputs[i] - y
        if y != self.data_outputs[i]:
            for j in range(len(self.weights)):
                self.weights[j] = self.weights[j] + \
                    (l_rate * error * self.data[i][j])

# neurono apmokymo funkcija
# funkcija = 0, jei pasirinkta slenkstinė aktyvacijos funkcija
# funkcija = 1, jei sigmoidinė
    def train(self, funkcija, iterations, l_rate):
        for _ in range(iterations):
            for i in range(len(self.data)):
                a = self.suma(self.data[i])
                if funkcija == 1:
                    y = self.sigmoidine(a)
                else:
                    y = self.slenkstine(a)

                # apskaičiuojama paklaida
                self.paklaida(y, i, l_rate)
```

3. Klasifikavimas

Kodo dalis, kurioje duomenys yra nuskaitomi iš failo, suskirstomi į klases pagal vartotojo pasirinktą duomenų rinkinį ir klasifikuoti.

```
if __name__ == '__main__':

    def klasifikuoti(klase1, klase2, learningRate, iterations, funkcija):
        dataMokymui = []
        dataTestavimui = []
        count = 0
        count2 = 0

        for item in klase1:
            a = [float(i) for i in item.split(',')]
            a.insert(0, 1.0)
            if count < 40:
                dataMokymui.append(a)
            if count >= 40:
                dataTestavimui.append(a)
            count += 1

        for item2 in klase2:
            b = [float(j) for j in item2.split(',')]
            b.insert(0, 1.0)
            if (inputDuomenys == 0 and count2 < 80) or (inputDuomenys == 1 and
     count2 < 40):
                dataMokymui.append(b)
            if (inputDuomenys == 0 and count2 >= 80) or (inputDuomenys == 1
    and count2 >= 40):
                dataTestavimui.append(b)
            count2 += 1

        if inputDuomenys == 0:
            output1 = np.array([[0] * 40]).T
            output2 = np.array([[1] * 80]).T
            data_outputsMokymui = np.concatenate((output1, output2))
            output3 = np.array([[0] * 10]).T
            output4 = np.array([[1] * 20]).T
            data_outputsTestavimui = np.concatenate((output3, output4))
        if inputDuomenys == 1:
            output1 = np.array([[0] * 40]).T
            output2 = np.array([[1] * 40]).T
            data_outputsMokymui = np.concatenate((output1, output2))
            output3 = np.array([[0] * 10]).T
            output4 = np.array([[1] * 10]).T
            data_outputsTestavimui = np.concatenate((output3, output4))

        weights = [0, 0, 0, 0, 0]
```

```
43
44          neuronas = Neuronas(dataMokymui, weights, data_outputsMokymui)
45          neuronas.train(funkcija, iterations, learningRate)
46          print("svoriai", neuronas.weights)
47          print("paklaida:  ", neuronas.paklaidaEW(dataTestavimui, funkcija,
    data_outputsTestavimui))
48          print("tikslumas:  ", neuronas.tikslumas(dataTestavimui, funkcija,
    data_outputsTestavimui))
49
50  #nuskaitomi duomenys is failo
51      setosa = []
52      versicolor = []
53      virginica = []
54      versicolorVirginica = []
55
56      with open("kelias iki iris.data failo", 'r') as read_obj:
57          for line in read_obj:
58              if "setosa" in line:
59                  setosa.append(line.replace(',Iris-setosa\n', ''))
60              if "versicolor" in line:
61                  versicolor.append(line.replace(',Iris-versicolor\n', ''))
62              if "virginica" in line:
63                  virginica.append(line.replace(',Iris-virginica\n', ''))
64
65  #sudaroma versicolor-virginica klase
66      versicolorVirginica.append(versicolor)
67      versicolorVirginica.append(virginica)
68      versicolorVirginica = [j for i in versicolorVirginica for j in i]
69
70  #gaunami duomenys is vartotojo
71      inputDuomenys = int(input("Pasirinkite duomenu rinkini [0] [1]: "))
72      inputGreitis = int(input("Mokymo greitis: "))
73      inputIteracijos = int(input("Iteracijos: "))
74      inputFunkcija = int(input("Slenkstine[0] ar sigmoidine[1] aktyvacijos
    funkcija?: "))
75
76      if inputDuomenys == 0:
77          klasifikuoti(setosa, versicolorVirginica, inputGreitis,
    inputIteracijos, inputFunkcija)
78      elif inputDuomenys == 1:
79          klasifikuoti(versicolor, virginica, inputGreitis, inputIteracijos,
    inputFunkcija)
80      else: print("neteisingi duomenys")
```

Taip pat yra galimybė apskaičiuoti paklaidą E(W), pagal formulę

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

, kuri gali būti apibrėžiama, kaip skirtumų tarp neurono išėjime gautų reikšmių ir norimų reikšmių sumos funkcija. Ši funkcija atvaizduojama kode:

```
    def paklaidaEW(self, data, funkcija, data_outputs):
        paklaidaE = 0
        for i in range(len(data)):
            paklaidaE += self.calculateOutput(data[i], funkcija) -
    data_outputs[i][0]
        return paklaidaE
```

Atlikus mokymo procesą, gaunamas klasifikavimo tikslumas, kuris yra santykis tarp teisingai klasifikuotų ir visų duomenų. Kiekvienam duomenų įrašui paskaičiuojama klasė pagal gautus neuronų svorius ir suskaičiuojamas santykis tarp teisingų išeičių bei visų išeičių. Funkcija tikslumui sužinoti:

```
    def tikslumas(self, data, funkcija, data_outputs):
        teisingiOutputs = 0
        for i in range(len(data)):
            if int(round(self.calculateOutput(data[i], funkcija))) ==
    data_outputs[i][0]:
                teisingiOutputs += 1
        return teisingiOutputs / len(data)
```

4. Tyrimas

Atliekamas tyrimas naudojant du duomenų rinkinius, nustatoma kaip rezultatai (tikslumo mato prasme) priklauso nuo mokymo greičio parametro, nuo to, kuri aktyvacijos funkcija yra naudojama, bei nuo epochų/iteracijų skaičiaus.

Pirmasis duomenų rinkinys:

Funkcija	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.
Greitis	1	1	0.01	0.01	0.001	0.001
Tikslumas:	100%	100%	100%	76%	100%	66%

1 lentelė. 1-ojo duomenų rinkinio rezultatų priklausomybė nuo mokymo greičio

Funkcija	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.
Iteracijos	1	1	2	2	3	3	4	4
Tikslumas:	66%	66%	66%	66%	100%	66%	100%	100%

2 lentelė. 1-ojo duomenų rinkinio rezultatų priklausomybė nuo iteracijų skaičiaus

1-oje lentelėje visiems bandymams naudotos 5 iteracijos. Galima pastebėti, jog keičiasi tik sigmoidinės funkcijos rezultatai, o mažėjant mokymo greičiui tikslumas taip pat mažėja, nes mokymo greičiu reguliuojamas gradientinio optimizavimo žingsnio ilgis, kuris, šiuo atveju mažėja, todėl reikia daugiau iteracijų, norint gauti tinkamą rezultatą.

2-oje lentelėje visiems bandymams naudotas mokymo greitis lygus 1. Matoma, jog, naudojant slenkstinę funkciją, galima pasiekti norimą rezultatą per mažesnį kiekį iteracijų, nei naudojant sigmoidinę funkciją. Šiuo atveju pasiekti 100% tikslumą, naudojant slenkstinę funkciją, užtenka 3 iteracijų, kai naudodant sigmoidinę - prireikia 4-ių.

Antrasis duomenų rinkinys:

Funkcija	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.
Greitis	1	1	0.5	0.5	0.01	0.01
Tikslumas:	60%	70%	60%	55%	60%	50%

3 lentelė. 2-ojo duomenų rinkinio rezultatų priklausomybė nuo mokymo greičio

Funkcija	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.	Slenkst.	Sigmoid.
Iteracijos	10	10	40	40	50	50	60	60	70	70
Tikslumas:	50%	50%	70%	50%	70%	70%	100%	60%	100%	100%

4 lentelė. 2-ojo duomenų rinkinio rezultatų priklausomybė nuo iteracijų skaičiaus

3-oje lentelėje visiems bandymams naudota 30 iteracijų. Galima pastebėti, jog, taip pat kaip ir pirmojo duomenų rinkinio atveju, keičiasi tik sigmoidinės funkcijos rezultatai.

4-oje lentelėje visiems bandymams naudotas mokymo greitis lygus 1. Naudojant slenkstinę aktyvacijos funkciją tinkamam rezultatui pasiekti reikia mažiau iteracijų, bei didinant iteracijų skaičių nėra tikslumo nukrypimų, kokie yra matomi taikant sigmoidinę funkciją, pvz.: kai iteracijų skaičius lygus 60, tikslumas staiga nukrenta 10% nuo praeito bandymo su 50 iteracijų.

5. Programos rezultatai

Paleidus programą, norint gauti 100% klasifikavimo tikslumą, nustatomi tokie geriausi rezultatai:

1. Rezultatai pirmajam rinkiniui:

   (a) Svoriai, taikant slenkstinę funkciją: w_0 = -1, w_1 = -1.1, w_2 = -3.6 w_3 = 5.2 w_4 = 2.2
   w_0 - slenkstis
   Paklaida: 0,
   Iteracijų skaičius: 4,
   Klasifikavimo tikslumas: 100%

   (b) Svoriai, taikant sigmoidinę funkciją: w_0 = -1.09, w_1 = -0.99, w_2 = -4.57 w_3 = 6.47 w_4 = 2.7
   Paklaida: 0.0012,
   Iteracijų skaičius: 4,
   Klasifikavimo tikslumas: 100%

2. Rezultatai antrajam duomenų rinkiniui:

   (a) Svoriai, taikant slenkstinę funkciją: w_0 = -2, w_1 = -45.7, w_2 = -16.3 w_3 = 54.5 w_4 = 46.1
   Paklaida: 0,
   Iteracijų skaičius: 70,
   Klasifikavimo tikslumas: 100%

   (b) Svoriai, taikant sigmoidinę funkciją: w_0 = -4.19, w_1 = -41.97, w_2 = -15.49 w_3 = 49.54 w_4 = 44.47
   Paklaida: 0.018,
   Iteracijų skaičius: 70,
   Klasifikavimo tikslumas: 100%

6. Išvados

1. Neuronui mokyti optimaliausia naudoti slenkstinę aktyvacijos funkciją, nes norimas rezultatas įmanomas gauti su mažiau iteracijų, nei taikant sigmoidinę funkciją.

2. Tyrimo metu pastebėta, kad keičiantis mokymo greičiui labiau kinta sigmoidinės aktyvacijos funkcijos rezultatai.

3. Mažėjant mokymo greičiui klasifikavimo tikslumas taip pat mažėja tokiam pat skaičiui iteracijų, nes mokymo greičiu reguliuojamas gradientinio optimizavimo žingsnio ilgis, kuris, jei mažėja, prireikia daugiau iteracijų, norint gauti norimą rezultatą.

4. Šiame tyrime buvo parinktas pakankamas kiekis duomenų mokymui, kad būtų gautas tinkamas klasifikavimo tikslumas.

5. Tyrime matoma, kad kartais galimi tikslumo nukrypimai, keičiant iteracijų skaičių, tai yra, nors tikimasi didesnio tikslumo, jis nebūtinai toks ir bus.

7. Programos kodas

Pilnas kodas Python kalba:

```
import numpy as np
import os

class Neuronas:
    def __init__(self, data, weights, data_outputs):
        self.data = data
        self.data_outputs = data_outputs
        self.weights = weights

# slenkstine funkcija
    def slenkstine(self, a):
        return 1 if a > 0 else 0

# sigmoidine funkcija
    def sigmoidine(self, a):
        return 1 / (1 + np.exp(-a))

# iejimo reiksmiu ir svoriu sandaugu suma
    def suma(self, data):
        a = np.dot(data, self.weights)
        return a

# nauju svoriu gavimas
    def paklaida(self, y, i, l_rate):
        error = self.data_outputs[i] - y
        if y != self.data_outputs[i]:
            for j in range(len(self.weights)):
                self.weights[j] = self.weights[j] + \
                    (l_rate * error * self.data[i][j])

#perceptrono veikimo paklaida E(W) - skirtumu tarp neurono isejime
#gautu reiksmiu ir norimu reiksmiu sumos funkcija
    def paklaidaEW(self, data, funkcija, data_outputs):
        paklaidaE = 0
        for i in range(len(data)):
            paklaidaE += self.calculateOutput(data[i], funkcija) -
    data_outputs[i][0]
        return paklaidaE


# neurono apmokymo funkcija
# funkcija = 0, jei pasirinkta slenkstine aktyvacijos funkcija
# funkcija = 1, jei sigmoidine
    def train(self, funkcija, iterations, l_rate):
        for _ in range(iterations):
```

```
45              for i in range(len(self.data)):
46                  a = self.suma(self.data[i])
47                  if funkcija == 1:
48                      y = self.sigmoidine(a)
49                  else:
50                      y = self.slenkstine(a)
51
52                  # čapskaiiuojama paklaida
53                  self.paklaida(y, i, l_rate)
54
55 #santykis tarp teisingai klasifikuotu ir visu duomenu
56      def tikslumas(self, data, funkcija, data_outputs):
57          teisingiOutputs = 0
58          for i in range(len(data)):
59              if int(round(self.calculateOutput(data[i], funkcija))) ==
    data_outputs[i][0]:
60                  teisingiOutputs += 1
61          return teisingiOutputs / len(data)
62
63
64 # skaiciuojama isejimo reiksme
65      def calculateOutput(self, data, funkcija):
66          a = self.suma(data)
67          if funkcija == 0:
68              return self.slenkstine(a)
69          else:
70 #Naudojant sigmoidine funkcija, neurono isejimo reiksmes yra intervale (0,1),
    todel
71 #vertinant rezultata, sios reiksmes yra suapvalinamos
72              return self.sigmoidine(a)[0] #int(round(self.sigmoidine(a)[0]))
73
74 if __name__ == '__main__':
75
76      def klasifikuoti(klase1, klase2, learningRate, iterations, funkcija):
77          data = []
78          dataMokymui = []
79          dataTestavimui = []
80          count = 0
81          count2 = 0
82
83          for item in klase1:
84              a = [float(i) for i in item.split(',')]
85              a.insert(0, 1.0)
86              data.append(a)
87              if count < 40:
88                  dataMokymui.append(a)
89              if count >= 40:
90                  dataTestavimui.append(a)
91              count += 1
```

```
92
93          for item2 in klase2:
94              b = [float(j) for j in item2.split(',')]
95              b.insert(0, 1.0)
96              data.append(b)
97              if (inputDuomenys == 0 and count2 < 80) or (inputDuomenys == 1 and
     count2 < 40):
98                  dataMokymui.append(b)
99              if (inputDuomenys == 0 and count2 >= 80) or (inputDuomenys == 1
    and count2 >= 40):
100                 dataTestavimui.append(b)
101             count2 += 1
102
103         if inputDuomenys == 0:
104             output1 = np.array([[0] * 40]).T
105             output2 = np.array([[1] * 80]).T
106             data_outputsMokymui = np.concatenate((output1, output2))
107             output3 = np.array([[0] * 10]).T
108             output4 = np.array([[1] * 20]).T
109             data_outputsTestavimui = np.concatenate((output3, output4))
110         if inputDuomenys == 1:
111             output1 = np.array([[0] * 40]).T
112             output2 = np.array([[1] * 40]).T
113             data_outputsMokymui = np.concatenate((output1, output2))
114             output3 = np.array([[0] * 10]).T
115             output4 = np.array([[1] * 10]).T
116             data_outputsTestavimui = np.concatenate((output3, output4))
117
118         weights = [0, 0, 0, 0, 0]
119
120         neuronas = Neuronas(dataMokymui, weights, data_outputsMokymui)
121         neuronas.train(funkcija, iterations, learningRate)
122         print("svoriai", neuronas.weights)
123         print("paklaida:  ", neuronas.paklaidaEW(dataTestavimui, funkcija,
    data_outputsTestavimui))
124         print("tikslumas:  ", neuronas.tikslumas(dataTestavimui, funkcija,
    data_outputsTestavimui))
125
126 #nuskaitomi duomenys is failo
127     setosa = []
128     versicolor = []
129     virginica = []
130     versicolorVirginica = []
131
132     with open("kelias iki iris.data failo", 'r') as read_obj:
133         for line in read_obj:
134             if "setosa" in line:
135                 setosa.append(line.replace(',Iris-setosa\n', ''))
136             if "versicolor" in line:
```

```
137                 versicolor.append(line.replace(',Iris-versicolor\n', ''))
138             if "virginica" in line:
139                 virginica.append(line.replace(',Iris-virginica\n', ''))
140 
141 #sudaroma versicolor-virginica klase
142     versicolorVirginica.append(versicolor)
143     versicolorVirginica.append(virginica)
144     versicolorVirginica = [j for i in versicolorVirginica for j in i]
145 
146 #gaunami duomenys is vartotojo
147     inputDuomenys = int(input("Pasirinkite duomenu rinkini [0] [1]: "))
148     inputGreitis = float(input("Mokymo greitis: "))
149     inputIteracijos = int(input("Iteracijos: "))
150     inputFunkcija = int(input("Slenkstine[0] ar sigmoidine[1] aktyvacijos
    funkcija?: "))
151 
152     if inputDuomenys == 0:
153         klasifikuoti(setosa, versicolorVirginica, inputGreitis,
    inputIteracijos, inputFunkcija)
154     elif inputDuomenys == 1:
155         klasifikuoti(versicolor, virginica, inputGreitis, inputIteracijos,
    inputFunkcija)
156     else: print("neteisingi duomenys")
```