Vilniaus Universitetas

Dirbtinio intelekto pagrindai

2 užduotis. Vieno neurono mokymas sprendžiant klasifikavimo uždavinį

Darbą atliko:

Vilniaus universiteto matematikos ir informatikos fakulteto

duomenų mokslo 3 kurso 2 grupės studentas

Matas Amšiejus

Vilnius, 2021 10 19

Turinys

Užduoties tikslas

Apmokyti vieną dirbtinį neuroną spręsti klasifikavimo uždavinį. Neuronas turės klasifikuoti vilkdalgius (gėlė) į dvi klases keičiant įvairius parametrus bei duomenų struktūrą. Užduotyje reikės gauti geriausius svorius, paklaidas, klasifikavimo tikslumus.

Pirmas punktas. Duomenų tvarkyba

Duomenys

Duomenys yra paimti iš „UCI Machine Learning“ puslapio (https://archive.ics.uci.edu/ml/datasets/iris). Duomenyse yra pateikta informacija apie vilkdalgius (gėlė) – jų taurėlapio ilgis, taurėlapio plotis, žiedlapio ilgis, žiedlapio plotis bei kokiai klasei priklauso ši gėlė (Setosa, Versicolor ir Virginica). Viso duomenyse yra 150 įrašų (gėlių), kiekvienas įrašas turi penkis atributus (stulpelius).

Kodo aprašymas

Pirma nuskaitome duomenis į .dataframe tipo kintamąjį (lentelę) naudojant „Pandas“ biblioteką. Pridedame stulpelių pavadinimus. Užduotuje reikės dviejų failų, tad sukuriame dvi kopijas: *duom1* ir *duom2*. Su *replace* funkcija pakeičiame gėlės klases į klases 0 arba 1. *duom1* bus gėlės, kur *Setosa* yra klasėje 0, o *Versicolor* ir *Virginica* – 1. Gauname sutvarkytus pirmus duomenis. *duom2* išmetame klasę *Setosa*, *Virginica* priskiriame į klasę 1, *Versicolor* į 0. Sukuriame 2 naujus failus, kad ateityje keičiant parametrus nereikėtų vis iš naujo atlikti šios tvarkybos.

```
def duomenu_tvarkyba(duom_pav, duom_pav1, duom_pav2):
    #nuskaitome iris.data duomenis
    duom = pd.read_csv(duom_pav, header = None, sep = ',')
    #pakeisciame stulpeliu vardus i lietuviskus
    duom.columns = ['taurel_ilg','taurel_plot','ziedl_ilg','ziedl_plot','klase']

    #kusime 2 failus, viena, kur Setosa bus klaseje 0 ir kitos - 1, antra, kur Versicolor
    bus 0 klaseje, o virginica - 1.
    #pirma sukuriame nauja lentele kuri yra originalios kopija
    duom1 = duom.copy()
    #pakeiciame setosa i 0, o kitas - i 1
    duom1["klase"].replace({"Iris-setosa": 0, "Iris-virginica": 1, "Iris-versicolor":1},
inplace = True)

    #Sukuriame antra kopija originalios lenteles naujoms klasems, kur Versicolor bus 0
klaseje, o Virginca - 1
    duom2 = duom.copy()
    #Pries tai pasaliname nereikalinga Setosa klase
    duom2.drop(duom2[duom2.klase == "Iris-setosa"].index, inplace=True)
    duom2["klase"] = duom2["klase"].replace({"Iris-virginica": 1, "Iris-versicolor":0})

    #Taigi turime 2 lenteles. Irasykime i faila, kad turetume ateiciai:
    duom1.to_csv(duom_pav1, index=False)
    duom2.to_csv(duom_pav2, index=False)
```

Antras punktas. Neurono kūrimas, mokymas ir testavimas

Neurono funkcija. Kodo aprašymas

Sukūriau elementarią neurono funkciją, kuri priima svorius (mūsų atveju jų yra 4 + 1 (visiems požymiams bei vienas slenksčiui (bias))), įėjimo reikšmes (taurėlapio ilgis, taurėlapio plotis, žiedlapio ilgis, žiedlapio plotis) bei aktyvacijos funkcijos tipą („slen“ arba „sig“). Jei funkcija slenkstinė, grąžinama 0 arba 1, jei sigmoidinė – skaičius intervale (0;1)

```
def neuronas(svoriai, x1, x2, x3, x4, aktyv):
    a = 1*svoriai[0] + x1*svoriai[1] + x2*svoriai[2] + x3*svoriai[3] + x4*svoriai[4]
```

```
if aktyv == "slen":
    if a>=0:    return 1
    else:       return 0
else:
    f = 1/(1+math.e**(-a))
    return f
```

Mokymo principas

Neurono mokymo prasmė – rasti optimalius svorius, su kuriais kuo tiksliau nustatoma klasė. Neurono mokymui pasitelksiu funkciją:

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

kur $w_k(i+1)$ – i+1 mokymo iteracijos[1] (toliau iteracija) k–tojo svorio reikšmė, η – mokymosi greitis, t_i – i iteracijos teisinga (target) klasė, y_i – i iteracijos neurono nustatyta klasė, x_{ik} – i iteracijos k–tasis įmėjimas.

Neurono mokymas. Kodo aprašymas

Kreipiamės į funkciją, duodami jai mokymosi duomenis (jau atrinktus nuo testavimo) (žr. *main* funkciją), aktyvacijos funkciją, norimų iteracijų skaičių, norimų mokymo epochų[2] (toliau epochų) skaičių bei mokymosi greitį. Taip pat galima įrašyti *True* reikšmes tam, kad funkcija spausdintų svorius, paklaidą ir klasifikavimo tikslumą (pirma *True* reikšmė) arba paklaidos grafiką (antra *True* reikšmė)). Atsitiktinai (intervale (0;1)) parenkame pradinius svorius. Kadangi programoje galima rinktis mokymosi epochų / iteracijų skaičių, nustatome, su kuria paduota reikšme gaunasi daugiau iteracijų (tą skaičių ir naudosime toliau mokant neuroną). Pirma sukame *while* ciklą, kuris iteruoja per visas duotas epochas (jei buvo paduotos iteracijos, tai jos buvo konvertuotos į epochas, o jų liekana toliau bus apdorojama *for* cikle). Iš duomenų paimama viena eilutė (*iejimai*). Tada kreipiamasi į funkciją *neuronas*, kuris grąžina išėjimą *y*. Lyginame, ar gautas išėjimas sutampa su tikslu ($t_i - isejimai[4]$). Jei taip, svorių nekeičiame, jei ne – keičiame svorius naudojantis anksčiau pateikta formule. Nulintas, slenksčio svoris, turi atskirą formulę, nes nulinė įėjimo reikšmė visada lygi 1. Kitus 4 svorius keičiame cikle. Taip pat suskaičiuojame mokymosi paklaidą bei klasifikavimo tikslumą. Formulė:

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

Kai praeiname visas epochas, tęsiame mokymąsi *for* cikle (tik tuo atveju, jei paduotas iteracijų skaičius nesidalino iš mokymosi failo dydžio). Mokymasis jame toks pats kaip ir *while* cikle, tačiau neskaičiuojama nei paklaida, nei klasifikavimo tikslumas (lyginti būtų nekorektiška). Ši *mokymas* funkcija grąžina svorių masyvą bei klasifikavimo tikslumą ir paklaidą. Jei kreipiantis į funkciją buvo pasirinktas bent vienas True, tai atitinkamai bus spausdinama informacija arba grafikas (žr. žemiau)

[1] Mokymo iteracija - neurono mokymo proceso dalis, kai apdorojamas vienas įėjimų vektorius (mūsų atveju viena gėlė).
[2] Mokymo epocha – tai neurono mokymo proceso dalis, kai apdorojamas visas įėjimų rinkinys (mūsų atveju visas mokymo failas).

```
def mokymas(mokDuom, aktyv, iterac_sk, epochu_sk, mok_greit, spausdint = False, grafikas =
False):
    np.random.seed(419)#nustatome, kad atsitiktinumas butu vienodas visoms versijoms
    svoriai = np.random.uniform(size=5)#sugeneruojame 5 svorius pradziai

    iter_temp = len(mokDuom)*epochu_sk # tikrinsime, su kuria riba bus mokomasi ilgiau
    if iterac_sk < iter_temp: iterac_sk = iter_temp # jei epochose bus daugiau iteraciju nei
duota, tai mokysim tie, kiek yra epochu

    epochu_sk = math.floor(iterac_sk/len(mokDuom))#gauname kiek minimum bus epochu
    iter_liek = iterac_sk % len(mokDuom) #gauname kiek dar reiks atlikti iteraciju praejus
visas epochas

    paklaida = 0
    klas_tiksl = 0
    ep_nr = 1

    paklaidos = np.empty([1,2])
    paklaidos = np.delete(paklaidos, 0, axis = 0)
    #ciklas, skirtas eiti tol, kol pasiekeme uzsibrezta epochu skaiciu
    while ep_nr <= epochu_sk:
        #mokymosi klasifikavimo tikslumas
        klas_tiksl = 0
        #mokymosi paklaida
        paklaida = 0
        for iterac in range(len(mokDuom)):
            iejimai = mokDuom.iloc[iterac]#nuskaitome lenteles eilute
            y = neuronas(svoriai, iejimai[0], iejimai[1], iejimai[2], iejimai[3], aktyv)
            #naudojame round jei sigmoidine funkcija, tai is intervalo (0,1) taps klase 0
arba 1
            if round(y,0) != iejimai[4]:
                svoriai[0]=svoriai[0]+mok_greit*(iejimai[4]-y)*1
                for i in range(4):
                    svoriai[i+1]=svoriai[i+1] + mok_greit * (iejimai[4]-y)*iejimai[i]
            else: klas_tiksl += 1
            paklaida += (iejimai[4] - y)**2
        paklaidos = np.append(paklaidos, [[ep_nr, 0.5*paklaida]], axis = 0)
        ep_nr+=1


        #ciklas, skirtas eiti tol, kol pasiekeme norima iteraciju skaiciu (tuo atveju, jei
norimu iteraciju sk. nesidalina
    #is duomenu rinkinio dydzio)
    for iterac in range(iter_liek):
        iejimai = mokDuom.iloc[iterac]#sukuriame iejimu vektoriu + klase (nuskaitome eilute
lenteles)
        y = neuronas(svoriai, iejimai[0], iejimai[1], iejimai[2], iejimai[3], aktyv)
        if round(y,0) != iejimai[4]:
            svoriai[0] = svoriai[0] + mok_greit *(iejimai[4]-y)
            for i in range(4):
                svoriai[i+1]=svoriai[i+1]+mok_greit*(iejimai[4]-y)*iejimai[i]

    #tam, kad paklaidu grazinimas veiktu su iteracijomis, su kuriuomis nesigauna pilna
epocha
    if(iter_liek == 0): gal_pakl = paklaidos[-1,1]
    else:
        #kazkoks didelis skaicius, kad mums aiskiai parodyti, kad nekorektiska lyginti
        gal_pakl = 1000000
        klas_tiksl = 0

    klas_tiksl = klas_tiksl/len(mokDuom)
    svoriai = np.append(svoriai, [[klas_tiksl, gal_pakl]])
```

```
    if grafikas:
        pakl_graf(paklaidos, aktyv)

    if spausdint:
        mok_info(svoriai)

    return svoriai
```

Papildomos neurono mokymo funkcijos. Kodo aprašymas

Buvo sukurtos papildomos dvi funkcijos: funkcija *mok_info*, skirta atspausdinti neurono mokymo informaciją (svorius, klasifikavimo tikslumą ir paklaidą) ir *pakl_grafikas*, skirta nupiešti mokymosi paklaidų grafiką nuo epochų skaičiaus.

```
#Pagalbine mokymo funkcija, jei norime spausdinti gautas reiksmes (svorius, paklaida,
rusiavimo tiksluma)
def pakl_graf(paklaidos, aktyv):
    plt.plot(paklaidos[:,0], paklaidos[:,1])
    if aktyv == 'slen':
        plt.title('Slenkstinės funkcijos paklaida nuo epochų skaičiaus')
    else: plt.title('Sigmoidinės funkcijos paklaida nuo epochų skaičiaus')
    plt.xlabel('Epochų skaičius')
    plt.ylabel('Paklaida')
    plt.show()

#Pagalbine mokymo funkcija, jei norime spausdinti paklaidos grafika
def mok_info(svoriai):
    print("Gauti svoriai: w_0 = {0}, w_1 = {1}, w_2 = {2}, w_3 = {3}, w_4 =
{4}\n".format(svoriai[0],
          svoriai[1], svoriai[2], svoriai[3], svoriai[4]))
    print("Paklaida = ", svoriai[5], "\n")
    print("Klasifikavimo tikslumas (mokymo) = ", svoriai[6])
```

Neurono testavimas. Kodo aprašymas

Kreipiamės į funkciją paduodami jau mokymas funkcijoje gautus svorius, testavimo duomenis (20 % visų duomenų (atrinkta main funkcijoje)), aktyvacijos funkciją. Šiuo atveju kodas geriausiai pakomentuotas komentaruose.

```
#Dirbtinio neurono testavimas

def testavimas(svoriai, testDuom, aktyv):
    #ivedame klasifikavimo tiksluma
    klas_tiksl = 0
    #einame per testDuom lentele (20 % visu duomenu)
    for iter in range(len(testDuom)):
        #nuskaitome salyga (eilute is lenteles)
        salyga = testDuom.iloc[iter]
        #liepiame neuronui nustatyti klase su konkreciais svoriais
        y=neuronas(svoriai, salyga[0], salyga[1], salyga[2], salyga[3], aktyv)
        #tikriname, ar neuronas parinko teisinga klase (apvaliname del sigmoidines
funkcijos)
        if salyga[4] == round(y, 0):
            #jei teisinga, sumuojame visas teisingai parinktas klases
            klas_tiksl+=1
    #uzbaigiame klasifikavimo tikslumo formule
    klas_tiksl = klas_tiksl / len(testDuom)
    #graziname paklaida ir klasivfikavimo tiksluma
    return klas_tiksl
```

Neurono mokymas ir testavimas

Šis kodas skirtas atlikti elementarų neurono mokymą ir testavimą bei spausdanti pastarojo rezultatus. Funkcijai paduodami mokymo, testavimo duomenys, aktyvacijos funkcija, norimų iteracijų, epochų skaičius, mokymosi greitis. Funkcija spausdina testinių duomenų eilutę i, neurono nustatytą klasę y_i (jei sigmoidinė funkcija, apvalinama iki 5 skaičių po kablelio, kad matytume skirtumus) ir trokštamą reikšmę t_i.

```
def mokTest(mokDuom, testDuom, aktyv, iter_sk, epochu_sk, mok_greit):
    #gauname neurono svorius
    svoriai = mokymas(mokDuom, aktyv, iter_sk, epochu_sk, mok_greit)
    #einame per testDuom lentele (20 % visu duomenu)
    print(" i\ty_i\tt_i")
    for iter in range(len(testDuom)):
        #nuskaitome salyga (eilute is lenteles)
        salyga = testDuom.iloc[iter]
        #liepiame neuronui nustatyti klase su konkreciais svoriais
        y=neuronas(svoriai, salyga[0], salyga[1], salyga[2], salyga[3], aktyv)
        #lyginame klases. Apvalinu iki 5, nes sigmoidine funkcija gali tureti labai maza
skirtuma nuo 0 ar 1
        #print(iter+1,"%7d"%",round(y,5),"\t\t",salyga[4])
        print('{: >2}  {: >7}\t{:0}'.format(iter+1, round(y,5), salyga[4]))
    return
```

Trečias punktas. Tyrimas

Main funkcija. Kodas

Pirma paleidžiame funkciją *duomenu_tvarkyba* (aprašyta anksčiau). Gauname 2 naujus .csv failus. Tada juos nuskaitome. Kiekvienus duomenis padaliname į dvi dalis – mokymo ir testavimo. Daliname santykiu 80 : 20 atitinkamai. Duomenys padalinami naudojant paprastąją negražintinę atsitiktinę imtį nustačius *seed*, t. y. nebus jokio atsitiktinumo leidžiant programą iš naujo (reikalinga atliekant tyrimą). Sukuriame vektorius, kuriuose bus saugomi mūsų norimi testuoti parametrų intervalai (epochų, iteracijų, mokymosi greičių). Kode paleisime funkciją *mokymas* (žr. aukščiau) bei funkcijas, kurios reikalingos norint atlikti tyrimą (geriausias aprašymas kodo komentaruose, funkcijos aprašytos žemiau (išskyrus *mokymas*)).

```
def main(args=None):
    #pirma nuskaitome duomenis ir sukuriame naujus failus:
    duomenu_tvarkyba('iris.data', 'duom1.csv', 'duom2.csv')

    #jei leisime ateityje, galesime iskart skaityti susikurtus failus
    duom1 = pd.read_csv('duom1.csv', sep = ',')
    duom2 = pd.read_csv('duom2.csv', sep = ',')

    #mokymo ir testavimo aibes imsime santykiu 80:20 atitinkamai
    #renkame paprastaja atsitiktine negrazintine imti
    mokDuom1 = duom1.sample(frac=0.8, random_state=419)#
    testDuom1 = duom1.drop(mokDuom1.index)
    mokDuom2 = duom2.sample(frac=0.8, random_state=419)#
    testDuom2 = duom2.drop(mokDuom2.index)

    #Paleiskime paprasta neurono mokyma (jei norime paklaidos grafiko ar spausdinimo
parametru, iveskime True, True)
    mokymas(mokDuom1, "slen", 0, 36, 0.1, True, True)
    mokymas(mokDuom1, "sig", 0, 36, 0.1, True, True)
    mokymas(mokDuom2, "slen", 0, 36, 0.1, True, True)
    mokymas(mokDuom2, "sig", 0, 36, 0.1, True, True)
```

```
    #leidziame paprasta neurono mokyma ir testavima (su testiniais duomenimis), grazina
svorius ir klas. tiksl.
    #print(mokTest(mokDuom1, testDuom1, "slen", 100, 0, 0.1))

    #nustatome testavimo iteraciju ir epochu intervalus (6 uzduotis)
    iter_sk = np.arange(start = 10, stop = 510, step = 10)
    epochu_sk = np.arange(start = 1, stop = 36)

    #nustatome testavimo mokymosi greicio intervalus (6 uzduotis)
    mok_greit = np.arange(start = 0.1, stop = 1, step = 0.1)
    #mok_greit = np.arange(start = 0.01, stop = 0.11, step = 0.01)

    ###############################################################
    #leidziame tyrima, 6 punktas (atkomentuoti tik po viena eilute per programos paleidima)
    #Pirmi duomenys, slenkstine funkcija
    #   Keiciant iteracijas
    test_iter(mokDuom1, testDuom1, "slen", iter_sk, mok_greit)
    #   Keiciant epochas
    test_epoch(mokDuom1, testDuom1, "slen", epochu_sk, mok_greit)

    #Pirmi duomenys, sigmoidine funkcija
    #   Keiciant iteracijas
    test_iter(mokDuom1, testDuom1, "sig", iter_sk, mok_greit)
    #   Keiciant epochas
    test_epoch(mokDuom1, testDuom1, "sig", epochu_sk, mok_greit)


    #Antri duomenys, slenkstine funkcija
    #   Keiciant iteracijas
    test_iter(mokDuom2, testDuom2, "slen", iter_sk, mok_greit)
    #   Keiciant epochas
    test_epoch(mokDuom2, testDuom2, "slen", epochu_sk, mok_greit)

    #Antri duomenys, sigmoidine funkcija
    #   Keiciant iteracijas
    test_iter(mokDuom2, testDuom2, "sig", iter_sk, mok_greit)
    #   Keiciant epochas
    test_epoch(mokDuom2, testDuom2, "sig", epochu_sk, mok_greit)
    return
```

Pastaba: norint leisti kodą, patartina užkomentuoti tuos testus, kurių tuo metu nereikia (t. y. tirti po vieną testą), nes programa užtrunka daug laiko, o grafikas parodomas tik vienas (paskutinis išsaugotas kompiuterio atmintyje).

Tyrimo funkcija. Kodas

Tyrime, kuriame keisime parametrus, naudosime funkciją *test_iter* arba *test_epoch*. Čia aprašysiu tik pastarąją, nes *test_iter* yra analogiška (tik be geriausio svorio ieškojimo, nes tada skaičiai išsiskaipytų). Pirma susikuriame laikiną geriausią svorių rinkinį. Tada iteruojame pro mokymosi greičių masyvą, kiekvienam greičiui bus brėžiama atskira linija. Sukame for ciklą, kuriame vis kreipiamės į mokymosi funkciją su nauju epocha parametru (vis didėjančiu). Paleidžiame testavimą. Tikriname, ar nauji svoriai turi geresnį klasifikavimo tikslumą. Jei taip, perrašome naujus geriausius svorius. Jei klasifikavimo tikslumas sutampa, tai tikriname, ar naujų svorių mokymosi paklaida yra mažesnė. Jei taip, perrašome svorius. Prieš pereinant prie kito greičio, brėžiame liniją, kurias po visų ciklų sujungiame į vieną grafiką. Taip pat atspausdiname geriausių svorių informaciją (aprašyta kode).

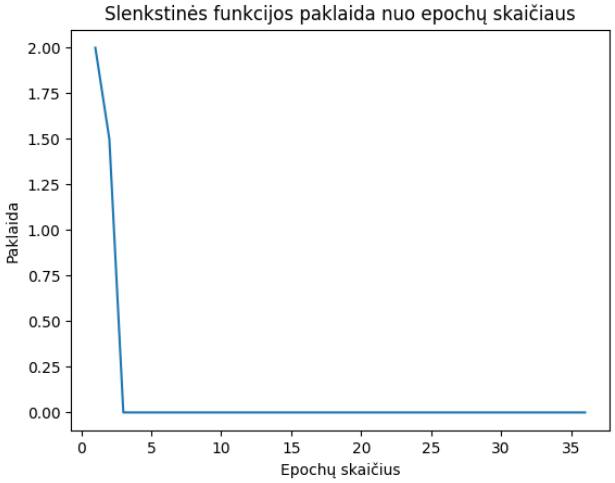
```
def test_epoch(mokDuom, testDuom, aktyv, epochu_sk, mok_greit):
    #sukuriame geriausio masyva: epocha, greitis, svoriai(5), klasifik. tikls. (test),
paklaida (mok)
    geriausias = np.array([0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1000.0])
    for greitis in mok_greit:
        greitis = round(greitis, 1)
        #greitis = round(greitis, 2)
        atsakymai = np.empty([1,3])
        atsakymai = np.delete(atsakymai, 0, axis = 0)
        for epocha in epochu_sk:
            svoriai = mokymas(mokDuom, aktyv, 0, epocha, greitis)
            atsak = testavimas(svoriai, testDuom, aktyv)
            atsakymai = np.append(atsakymai, [[atsak, greitis, epocha]], axis = 0)
            #tikriname, ar tai geriausi gauti svoriai pagal testDuom klasifikavimo tiksluma
            if geriausias[7]<atsak:
                geriausias[0] = epocha
                geriausias[1] = greitis
                for i in range(5):  geriausias[i+2] = svoriai[i]
                geriausias[7] = atsak
                geriausias[8] = svoriai[6]
            #jei klas. tiksl. sutampa, tai tikriname, kurio paklaida (mokDuom) yra mazesne
            elif geriausias[7] == atsak:
                if geriausias[8] > svoriai[6]:
                    geriausias[0] = epocha
                    geriausias[1] = greitis
                    for i in range(5):  geriausias[i+2] = svoriai[i]
                    geriausias[7] = atsak
                    geriausias[8] = svoriai[6]

        plt.plot(atsakymai[:,2], atsakymai[:,0], label = str(greitis))

    plt.legend(title = 'Greičiai')

    if aktyv == 'slen':
        plt.title('Slenkstinės funkcijos klasifikavimo tikslumas nuo mokymosi greičio')
    else: plt.title('Sigmoidinės funkcijos klasifikavimo tikslumas nuo mokymosi greičio')

    plt.xlabel('Epochų skaičius')
    plt.ylabel('Klasifikavimo tikslumas')
    plt.show()

    print('Geriausias rastas variantas: epocha={0},\n mokymosi greitis={1},\n w0={2},
w1={3}, w2={4}, w3={5}, w4={6} \n Klasifikavimo tikslumas (testDuom)={7},\n
Paklaida(mokDuom)={8}'.format(
          round(geriausias[0],0), round(geriausias[1],1), geriausias[2], geriausias[3],
geriausias[4], geriausias[5],
          geriausias[6], round(geriausias[7],2), round(geriausias[8],2)))
    return
```

Tyrimo rezultatai. Pirmi duomenys

Imsime *duom1*, t. y. tuos, kur *Setosa* yra klasėje 0, *Versicolor* ir *Virginica* – 1. Pirma paleisime slenkstinę funkciją kuri pereis per mokymosi failą 30 kartų (30 epochų). Pirma patikrinkime, kaip kito jos mokymosi paklaida didėjant epochoms. *main* funkcijoje kreipsimės sakiniu

```
mokymas(mokDuom1, "slen", 0, 30, 0.1, True, True)
```

Gauname, kad neuronas labai efektyviai atskyrė gėles besimokydamas, ko ir buvo galima tikėtis (*duom1* gėlių klasės turi aiškius skirtumus).



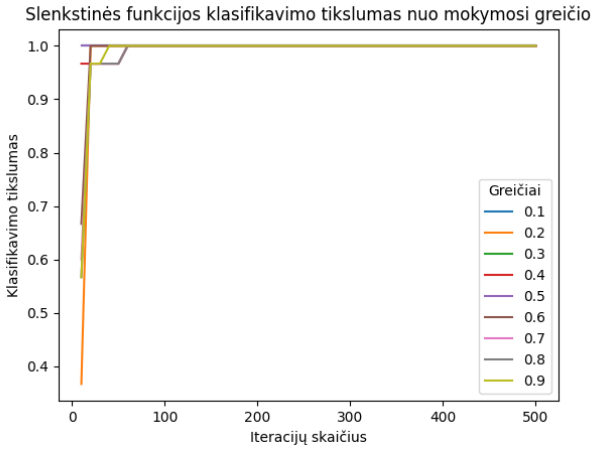
1 pav. Slenkstinės funkcijos paklaida nuo epochų skaičiaus. Pirmi duomenys

Dabar paleiskime pilną testavimą keičiant mokymosi greičius, iteracijas, epochas. Grafikuose bus vertinamas testinio failo klasifikavimo tikslumas. Tam naudosime kodą

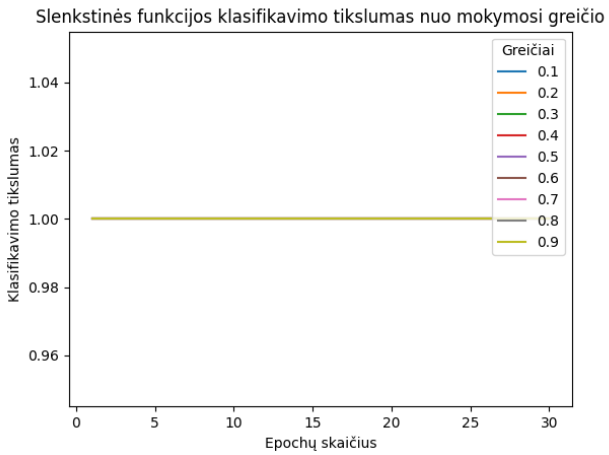
```
test_iter(mokDuom1, testDuom1, "slen", iter_sk, mok_greit)
```

Pirma tikriname kitimą 500 iteracijų (5 epochos). Matome, kad kaip ir buvo galima tikėtis, maksimalus klasifikavimo tikslumas buvo pasiektas itin greitai (apie 50 iteracijų), nes duomenų skirtumai labai aiškūs. Epochų grafikas tik patvirtina svarstymus. Taip pat gauname, kad geriausi svoriai jau buvo pasiekti 3 epochoje. Jų informacija:

```
Geriausias rastas variantas: epocha=3.0,
mokymosi greitis=0.1,
w0=-0.009830982232829494,          w1=-0.04814652660798974,          w2=-0.9374922259775947,
w3=0.8079479305385608, w4=0.6224444194938454
Klasifikavimo tikslumas (testDuom)=1.0,
Paklaida(mokDuom)=0.0
```

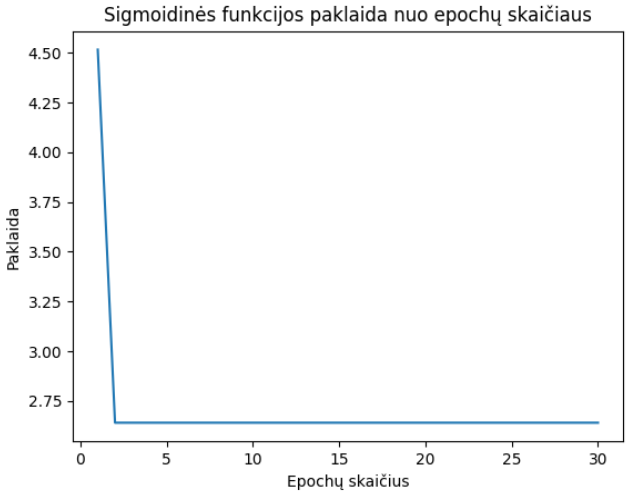


3 pav. Slenkstinės f-jos klasifikacijos tikslumas nuo iteracijų ir mokymosi greičio. Pirmi duomenys



2 pav. Slenkstinės f-jos klasifikacijos tikslumas nuo epochų ir mokymosi greičio. Pirmi duomenys

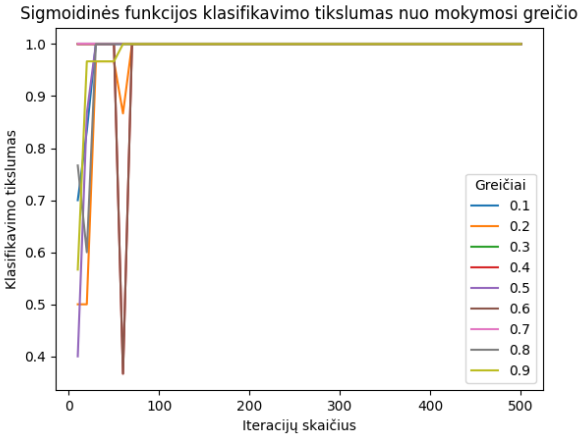
Dabar panaudokime sigmoidinę aktyvacijos funkciją. Vėl tikriname paklaidą:



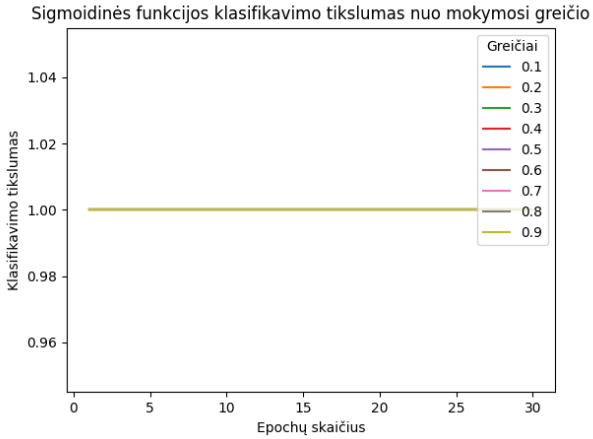
4 pav. Sigmoidinės funkcijos paklaida nuo epochų skaičiaus. Pirmi duomenys

Matome, kad sigmoidinės funkcijos paklaida irgi sumažėjo iki gana mažos (negauname 0, nes mūsų *y* nėra apvalinamas į 0 ar 1, o yra tame intervale). Vėl atlikime pilną testavimą. Gauname, kad maksimalus klasifikavimo tikslumas pasiekiamas per mažiau nei epochą. Taip pat gauname, kad geriausi svoriai jau buvo pasiekti 2 epochoje (paklaida 0 dėl apvalinimo (gaunasi labai maža)). Jų informacija:

```
Geriausias rastas variantas: epocha=2.0,
mokymosi greitis=0.5,
w0=-0.4864092326681872, w1=-1.0035065966546686, w2=-3.544084264127983, w3=5.539355566522306,
w4=2.556188811129071
Klasifikavimo tikslumas (testDuom)=1.0,
Paklaida(mokDuom)=0.0
```



6 pav. Sigmoidinės f-jos klasifikacijos tikslumas nuo iteracijų ir mokymosi greičio. Pirmi duomenys



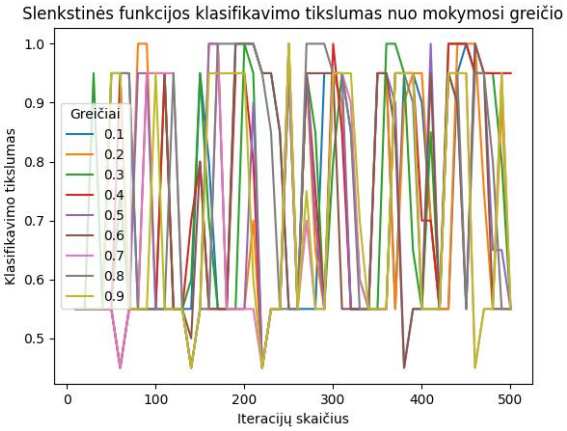
5 pav. Sigmoidinės f-jos klasifikacijos tikslumas nuo epochų ir mokymosi greičio. Pirmi duomenys

Sigmoidinė funkcija geresnius rezultatus pasiekė greičiau (2 epochos) nei slenkstinė (3 epochos), todėl vėliau spausdinsiu sigmoidinės funkcijos reluztatus iš 2 epochų, su mokymosi greičiu 0,5.

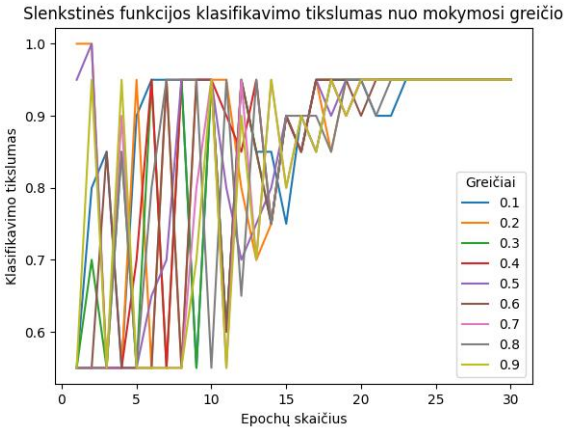
Tyrimo rezultatai. Antri duomenys

7 pav. Slenkstinės funkcijos paklaida nuo epochų skaičiaus. Antri duomenys

Dabar atlikime pilną tyrimą. Gauname, kad pirmų 500 iteracijų tikrai nepakanka norint išmokyti neuroną klasifikuoti *duom2*. Gauti rezultatai gali nustebinti, nes tiksliausia yra 2 epocha. Mums tiesiog pasisekė, nes kaip matome, tikslumas vėliau krenta, kas implikuoja, kad mokymosi paklaida buvo maža ir dėl to neuronas taisėsi, sugadindamas savo tobulą atrinkimą. Permaišius duomenis, šie svoriai nebebūtų tokie tikslūs.



9 pav. Slenkstinės f-jos klasifikacijos tikslumas nuo iteracijų ir mokymosi greičio. Antri duomenys



8 pav. Slenkstinės f-jos klasifikacijos tikslumas nuo epochų ir mokymosi greičio. Antri duomenys

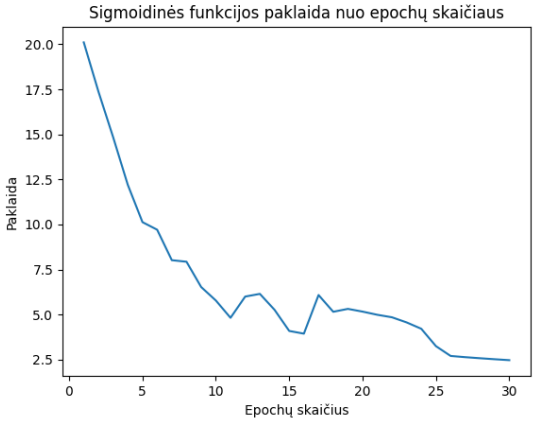
```
Geriausias rastas variantas: epocha=2.0,
mokymosi greitis=0.2,
w0=-0.7098309822328295, w1=-1.9581465266079892, w2=-2.0874922259775937,
w3=2.6579479305385614, w4=3.1724444194938437
Klasifikavimo tikslumas (testDuom)=1.0,
Paklaida(mokDuom)=15.0
```

Pakeitus kode, kad geriausio varianto svorių ieškotų tik nuo 5 epochos, gauname kitokius, universalesnius rezultatus:

```
Geriausias rastas variantas: epocha=28.0,
mokymosi greitis=0.1,
```

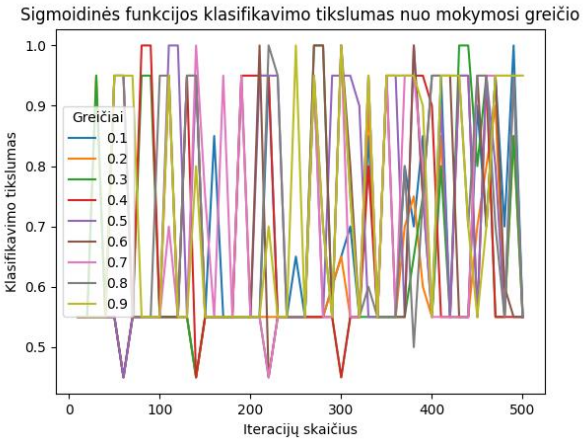
```
w0=-2.809830982232831, w1=-4.348146526607991, w2=-4.787492225977583, w3=6.887947930538544,
w4=7.502444419493836
Klasifikavimo tikslumas (testDuom)=0.95,
Paklaida(mokDuom)=2.0
```

Dabar tikrinkime sigmoidinės funkcijos paklaidų pokyčius. Matome, kad jos paklaidos mažėja pastoviau (mažiau spygliuota) nei slenkstinės funkcijos. Taip pat paklaida nukrenta žemiau. Galime tikėtis, kad sigmoidinė funkcija klasifikuos tiksliau.

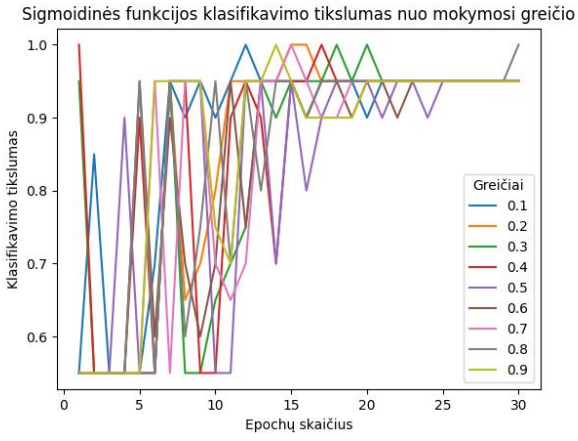


10 pav. Sigmoidinės f-jos paklaida nuo epochų skaičiaus. Antri duomenys

Atlikime pilną tyrimą. Su iteracijomis, kaip ir buvo galima tikėtis, gauname vėl visišką nepastovumą. Tačiau su epochomis mes vėl gauname neblogą klasifikavimo pakilimą daugmaž nuo 13-14 epochos. Tai ir patvirtina geriausios gautos svorių reikšmės iš 14 epochos. Matome, kad sigmoidinė funkcija sugebėjo išgauti tikslesnes reiškmes nei slenkstinė dėl savo detalesnio svorių kitimo.



12 pav. Sigmoidinės f-jos klasifikacijos tikslumas nuo iteracijų ir mokymosi greičio.



11 pav. Sigmoidinės f-jos klasifikacijos tikslumas nuo epochų ir mokymosi greičio.

```
Geriausias rastas variantas: epocha=14.0,
 mokymosi greitis=0.9,
 w0=-16.04114086127243, w1=-31.70097039108246, w2=-32.17159979431841, w3=46.21704216133822,
w4=43.82343336683911
```

```
Klasifikavimo tikslumas (testDuom)=1.0,
Paklaida(mokDuom)=3.337
```

Kadangi su slenkstine funkcija mums iš pradžių pasisekė, mes lyginame slenkstinę po 28 epochų ir sigmoidinę po 14. Pagal klasifikavimo tikslumą galime teigti, kad sigmoidinė funkcija vėl pasirodė geriau.

Tyrimo tiksliausių reikšmių vertės

Duom1

Kaip ir minėjau anksčiau, geriausia buvo sigmoidinė funkcija su mokymosi greičiu 0,5. Duomenys bus po 3 epochų mokymosi. Matome, kad visi skirtumai tarp y_i ir t_i yra labai maži, o apvalinant gautume, kad visos klasės buvo nustatytos teisingai

i	y_i	t_i	i	y_i	t_i	i	y_i	t_i
1	0.00015	0.0	11	0.0006	0.0	21	0.99998	1.0
2	0.00016	0.0	12	0.99998	1.0	22	1.0	1.0
3	0.0	0.0	13	1.0	1.0	23	1.0	1.0
4	8e-05	0.0	14	0.99999	1.0	24	1.0	1.0
5	1e-05	0.0	15	0.99999	1.0	25	1.0	1.0
6	0.0018	0.0	16	0.99991	1.0	26	1.0	1.0
7	0.00116	0.0	17	0.99897	1.0	27	1.0	1.0
8	0.0	0.0	18	0.99991	1.0	28	1.0	1.0
9	0.0004	0.0	19	0.9998	1.0	29	1.0	1.0
10	0.0004	0.0	20	0.99985	1.0	30	1.0	1.0

Duom2

Taigi spausdinsiu duomenis po 14 epochų sigmoidinės funkcijos su mokymosi greičiu 0,9. Skirtumai tarp y_i ir t_i yra vėl labai maži, tai galima teigti, kad neuronas buvo apmokytas ir reikšmes nustatė labai neblogai su komplikuotesniu failu.

i	y_i	t_i	i	y_i	t_i	i	y_i	t_i
1	0.0	0.0	8	0.0	0.0	15	1.0	1.0
2	0.0	0.0	9	0.0	0.0	16	0.99987	1.0
3	0.0	0.0	10	1.0	1.0	17	1.0	1.0
4	0.0	0.0	11	1.0	1.0	18	1.0	1.0
5	0.01821	0.0	12	1.0	1.0	19	1.0	1.0
6	0.0	0.0	13	1.0	1.0	20	1.0	1.0
7	0.0	0.0	14	1.0	1.0			

Tyrimo apibendrinimas

Kaip ir minėta anksčiau, didžiausią įtaką klasifikavimo tikslumui turėjo duomenų failas. Ten, kur duomenų klasės turi aiškią atskirtį, slenkstinė, elementaresnė funkcija, puikiai atlieka savo darbą ir mums net nereikia detalesnės sigmoidinės. Tačiau su painesniais duomenimis sigmoidinė funkcija susitvarkė geriau, ką ir galima matyti iš mūsų tyrimo. Taip pat negalima teigti,

kad geriausi mokymosi greičiai yra mažiausi (bent mūsų atveju). Tačiau jų įtaka ne tokia reikšminga lyginant su duomenimis ir su iteracijų / epochų skaičiumi.

Galutinės išvados

Dirbtinio neurono principas yra pagrįstas mokymusi, tobulėjimu, tad iš šio projekto matome, kad duomenų kiekis, epochų skaičius turi labai didelę įtaką gautoms reikšmėms, neurono tikslumui. Taip pat kuo duomenys painesni, tuo rezultatai labiau svyruoja, todėl kartais verta susimąstyti apie gautų rezultatų tikslumą ir kaip jie priklauso nuo atsitiktinumo (tokiais atvejais būtų naudinga kryžminė patikra). Mūsų atveju duomenyse, kur vilkdalgių klasės turėjo aiškius skirtumus, neuronas susitvarkė puikiai (užtekdavo kelių epochų, net ir keliasdešimt iteracijų), o kur gėlės persimaišė (antri duomenys), reikalingų epochų skaičius išaugo drastiškai. Tad verta žinoti, kokius duomenis tiriame, nes pirma gauta gera klasifikavimo reikšmė nebūtinai reiškia, kad neuronas jau turi baigti mokytis. Mokymasis tol, kol pasieksime norimą mokymosi paklaidą irgi būtų geras variantas.