VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMATIKOS KATEDRA

Skaitmeninis intelektas ir sprendimų priėmimas

II užduotis: Vieno neurono (perceptrono) mokymas sprendžiant klasifikavimo uždavinį

Tomas Giedraitis
VU MIF Informatika
3 kursas 3 grupė

Vilnius 2021

1. Darbo tikslas

Apmokyti vieną neuroną spręsti nesudėtingą dviejų klasių uždavinį, ir atlikti tyrimą, kaip rezultatai priklauso nuo mokymo greičio parametro, aktyvacijos funkcijos tipo (slenkstinė ir sigmoidinė) bei iteracijų (tuo pačiu – ir epochų) skaičiaus. Naudojami irisų duomenys[1], kurių klasės yra žinomos (Setosa, Versicolor ir Virginica).

2. Darbo eiga

2.1. Duomenys

Kadangi vienas neuronas geba spręsti tik dviejų klasių klasifikavimo uždavinį, tai trijų klasių uždavinys suskaidytas į du dviejų klasių uždavinius:

(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 yra 0 arba 1. Kiekvienas duomenų objektas, be nurodytos jo klasės, turi dar keturis požymius, nusakytus realiais skaičiais:

1. Taurėlapio ilgis, cm
2. Taurėlapio plotis, cm
3. Žiedlapio ilgis, cm
4. Žiedlapio plotis, cm

Abiem uždaviniams kiekvienos iš dviejų klasių duomenys padalinti į mokymo (90 %) ir testavimo (10 % duomenų) aibes. Pasiskirstymas atrodo taip:

(1) Setosa: 45 objektai mokymui, 5 – testavimui, Versicolor ir Virginica: 90 – mokymui, 10 – testavimui.
(2) Versicolor: 45 objektai mokymui, 5 – testavimui. Virginica: 45 – mokymui, 5 – testavimui.

2.2. Apmokymas

Buvo sukurta Python programa (žr. A.1 priedą), kuri apmoko vieną neuroną (perceptroną) spręsti klasifikavimo uždavinį.

Pradinių svorių W_0 reikšmės buvo nustatytos atsitiktinai iš intervalo [-1;1), panaudojus *numpy.random.random()* funkciją, generuojančią atsitiktinius skaičius intervale [0;1) ir panaudojant gautą skaičių kaip argumentą funkcijai $2x - 1$, praplečiančiai intervalą iki [-1;1).

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

Pirmajam įėjimo vektoriui

$X_1 =< x_{10}, x_{11}, x_{12}, x_{13}, x_{14} >$

buvo apskaičiuota paklaida

$E_1(W_1) = t_1 - y_1$,

kur W_1 yra vektorius iš 5 svorių, įskaitant slenkstį w_{10}, t.y.

$W_1 =< w_{10}, w_{11}, w_{12}, w_{13}, w_{14} >$,

$y_1 = f_{akt.}(\sum_{k=0}^{4} w_{1k}x_{1k})$ – gauta aktyvacijos funkcijos reikšmė, priklausomai nuo svorių ir įėjimų, o t_1 – norima klasės reikšmė.

Tuomet visi svoriai, pasinaudojus gauta paklaida, keičiami pagal šią taisyklę:

$w_{(i+1)k} = w_{ik} + \eta E\ (W_i)x_{ik} = w_{ik} + \eta(t_i - y_i)x_{ik}$,

kur η– iš anksto pasirinktas mokymosi greitis.

Toliau, svorių reikšmėms jau pakitus, apdorojamas antrasis įėjimo vektorius, ir taip tęsiamas iteracinis procesas, kol visi įėjimo vektoriai yra apdorojami, ir galutinės svorių reikšmės yra gaunamos.

Duomenys programoje nuskaitomi iš *.csv* failo, prieš paleidžiant programą galima nustatyti mokymo greitį, aktyvacijos funkciją (slenkstinė arba sigmoidinė), ir mokymo iteracijų (epochų) skaičių (iteracija – neurono mokymo proceso dalis, kurios metu apdorojamas vienas įėjimų vektorius, tuo tarpu vienos epochos metu apdorojamas visas įėjimų vektorių rinkinys vieną kartą).

Taip pat, pradiniai duomenys prieš apmokymą buvo perdėlioti taip, kad kas antras duomuo būtų iš kitos klasės. Tokiu būdu jau net ir su labai mažu iteracijų skaičiumi galima apžvelgti neurono mokymąsi klasifikuoti tarp dviejų klasių.

2.3. Tyrimo eiga

Abiejų klasifikavimo uždavinių atveju tyrimui buvo pasirinkti skirtingi mokymosi greičiai (η) iš aibės $\{0,01;\ 0,1;\ 0,3;\ 0,5;\ 0,7;\ 1,0\}$.

Su kiekvienu mokymo greičiu buvo nustatomas klasifikavimo tikslumas (*k.t.*) tiek su mokymo duomenimis, tiek su testavimo duomenimis.

Sigmoidinės aktyvacijos funkcijos atveju, prieš skaičiuojant klasifikavimo tikslumą, buvo pasirinkti šie klasifikavimo rėžiai:

0,0–0,1: klasė 0
0,1–0,9: klasė nenustatyta
0,9–1,0: klasė 1

1–osios užduoties atveju buvo fiksuojamas klasifikavimo tikslumas su šiais iteracijų skaičiais:

{2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
110, 120, 130, 140, 150, 160, 170, 180, 200,
210, 220, 230, 240, 250, 300, 350, 400, 450, 500,
550, 600, 750, 800, 850, 900, 950, 1000,
1200, 1400, 1600, 1800, 2000,
2200, 2400, 2600, 2800, 3000,
3200, 3400, 3600, 3800, 4000}

2–osios užduoties atveju:

{2, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,
110, 120, 130, 140, 150, 160, 170, 180, 200,
210, 220, 230, 240, 250, 300, 350, 400, 450, 500,
550, 600, 750, 800, 850, 900, 950, 1000,
1200, 1400, 1600, 1800, 2000,
2200, 2400, 2600, 2800, 3000,
3200, 3400, 3600, 3800, 4000,
5000, 6000, 7000, 8000, 9000, 10000,
20000, 30000, 50000, 100000,
200000, 300000, 500000,
1000000}

2–ajame uždavinyje prireikė žymiai daugiau iteracijų, kad klasifikavimo tikslumas pasiektų didesnę reikšmę (daugiau nei 0,9). Tai susiję su tuo, kad uždavinys nėra tiesiškai atskiriamas.

Dar vienas iš uždavinių parametrų – aktyvacijos funkcija – taip pat buvo keičiamas, tad neuronas abiejuose užduotyse buvo apmokomas nurodytą iteracijų skaičių tiek su slenkstine, tiek su sigmoidine aktyvacijos funkcija:

$$f_{slenkstinė}(x) = \begin{cases} 1, & jei\ x > 0 \\ 0, & jei\ x \leq 0 \end{cases}$$

$$f_{sigmoidinė}(x) = \frac{1}{1+e^{-x}}$$

Rezultatai

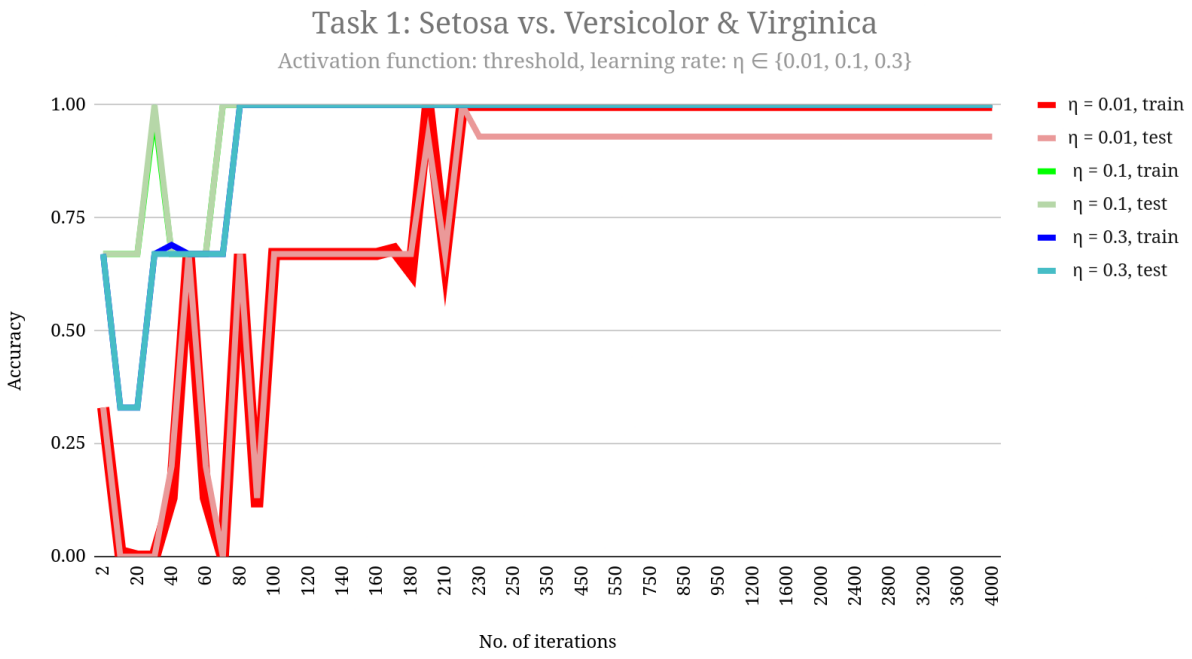
3.1. 1–oji užduotis (1–a klasė: Setosa, 2–a klasė: Versicolor ir Virginica)

3.1.1. Aktyvacijos funkcija – slenkstinė

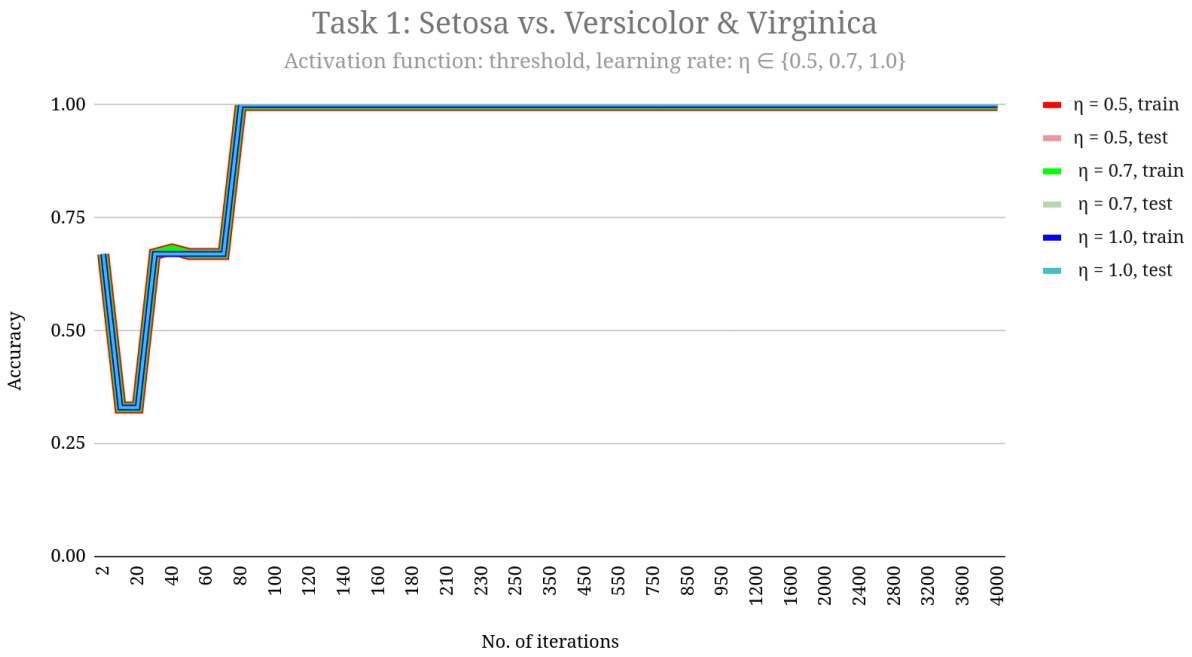
Grafike vaizduojama, kaip klasifikavimo tikslumas priklauso nuo iteracijų skaičiaus. 1–oje užduotyje, kadangi mokymo duomenų aibės dydis yra 135, tai ties šiuo skaičiumi pasibaigia pirmoji epocha, ir toliau epochų skaičius didėja vienetu kas 135 iteracijas. Didžiausia reikšmė *x* ašyje, 4000, reiškia jau beveik 30 epochų.

Dėl patogumo pateikiami du grafikai (1 pav., 2 pav.), kur pirmame matomi rezultatai naudojant pirmuosius tris mokymo greičius (0,01, 0,1, 0,3), antrame – (0,5, 0,7, 1,0).

Ties kiekvienu mokymosi greičiu grafiko reikšmės pateikiamos skirtinga spalva (iš eilės – raudona, žalia, mėlyna). Šios spalvos nurodo mokymo aibės rezultatus, o grafiko legendoje pažymėti šių spalvų atspalviai – testavimo duomenų rezultatus.



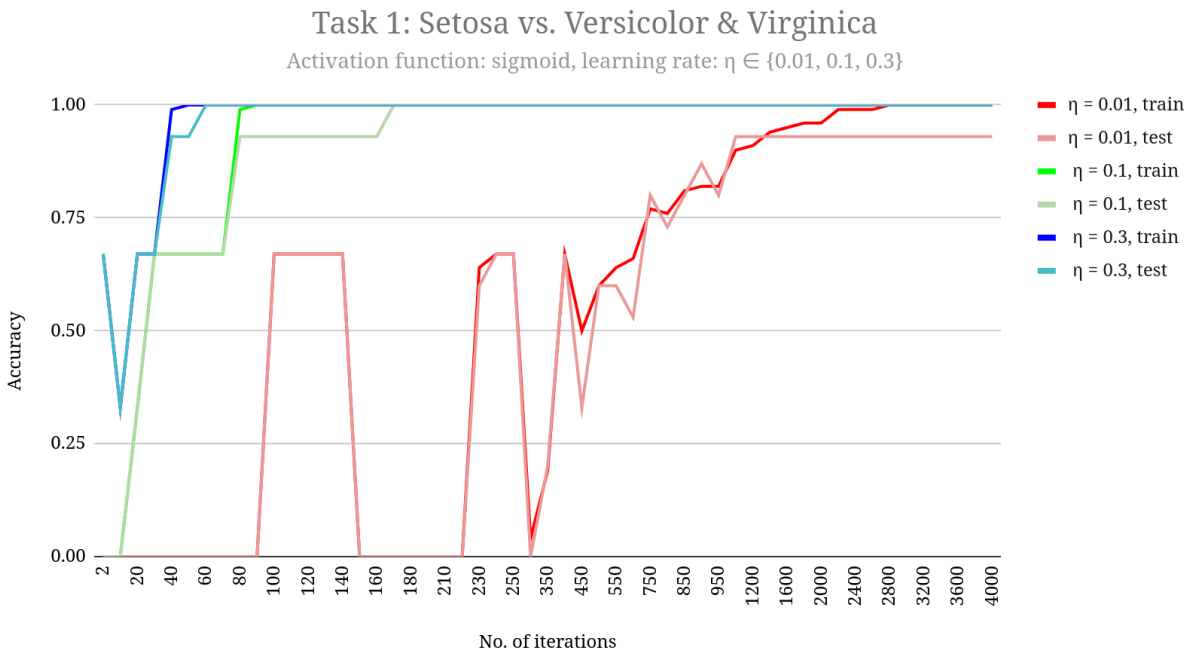
1 pav. 1–os užduoties grafikas nr. 1



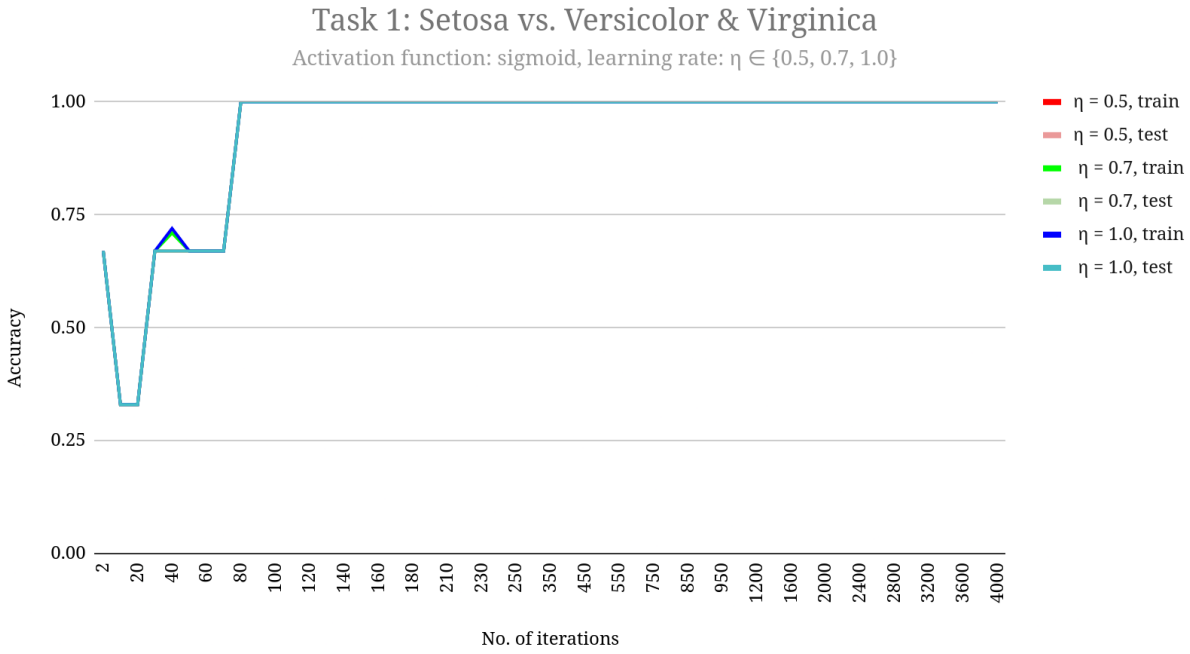
2 pav. 1–os užduoties grafikas nr. 2

Iš grafikų matome, kad, didėjant iteracijų skaičiui, didėja ir klasifikavimo tikslumas. Su beveik visais mokymo greičiais tiek mokymo, tiek testavimo aibės klasifikavimo tikslumas pasiekia stabilią reikšmę 1,00, ir toliau jau nekinta didėjant iteracijų skaičiui. Ši riba skirtinga pirmiems trims mokymosi greičiams (1 pav.), o paskutinių trijų (2 pav.) – sutampa (80 iteracijų). Taip pat, 1 pav. matome, kad su mokymo greičiu 0,01 testavimo aibės klasifikavimo tikslumas nepasiekia 1,00, o laikosi ties 0,93. Taigi, 0,01 mokymosi greitis davė prasčiausią rezultatą, o labiausiai tiko greitis 0,1, kuomet stabilumas pasiektas su 70 iteracijų.

3.1.2. Aktyvacijos funkcija – sigmoidinė



3 pav. 1–os užduoties grafikas nr. 3



4 pav. 1–os užduoties grafikas nr. 4

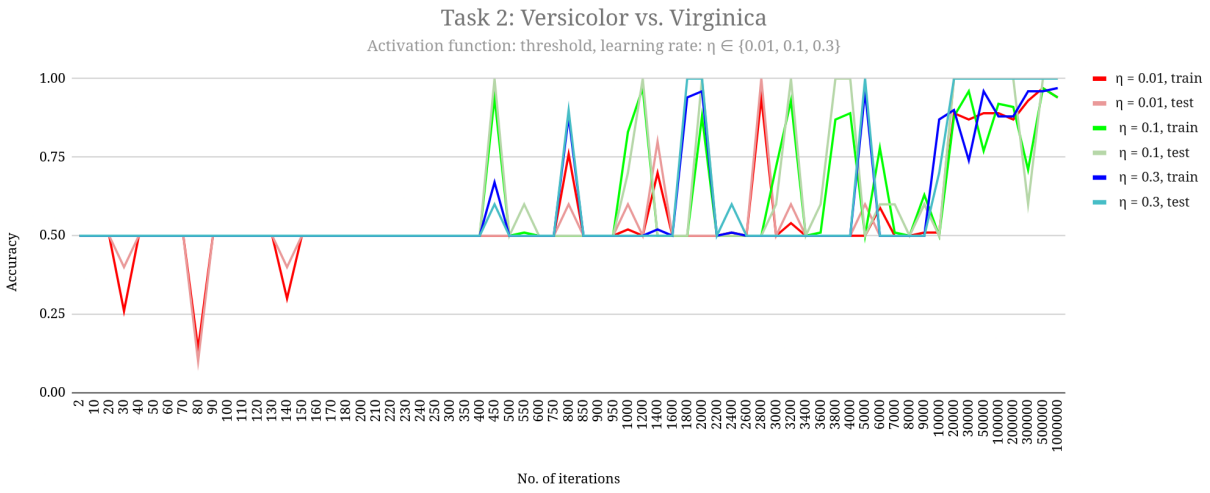
Su sigmoidine aktyvacijos funkcija rezultatai yra gana panašūs, ypač su mokymosi greičiais 0,5, 0,7 ir 1,00. Geriausias rezultatas pasiektas šįkart ne su 0,1, o su 0,3 mokymosi greičiu. Tuo tarpu su $\eta = 0{,}01$ mokymo aibės klasifikavimo tikslumas pasiekia 1,00 tik ties 2800 iteracijų, o

testavimo – nepasiekia 1,00, ir lieka ties 0,93. Taigi, abiem atvejais prasčiausią rezultatą davė η = 0,01.

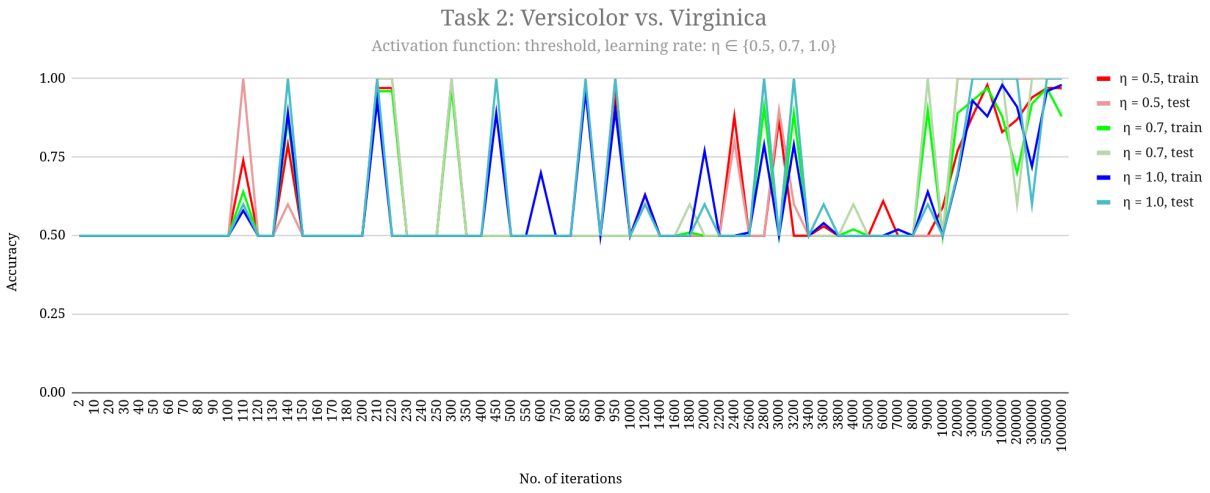
Palyginus slenkstinės ir sigmoidinės funkcijos rezultatus 1-oje užduotyje, matome, kad su η = 0,5, 0,7 ir 1,00 rezultatai pilnai sutapo, tuo tarpu su mažesniais η = 0,01, 0,1 slenkstinė akt. f-ja davė geresnius rezultatus (stabilumas ties 230 ir ties 70 iteracijų prieš stabilumą ties 2800 ir 170 iteracijų). 0,1 atveju, augimas link stabilumo buvo labai panašus, o 0,01 atveju sigmoidinė f-ja pasirodė žymiai prasčiau. Tačiau pats geriausias rezultatas buvo pasiektas sigmoidinės funkcijos atveju su η = 0,3 (stabilumas ties 60 iteracijų).

3.2. 2-oji užduotis (1-a klasė: Versicolor, 2-a klasė: Virginica)

3.2.1. Aktyvacijos funkcija – slenkstinė



5 pav. 2-os užduoties grafikas nr. 1

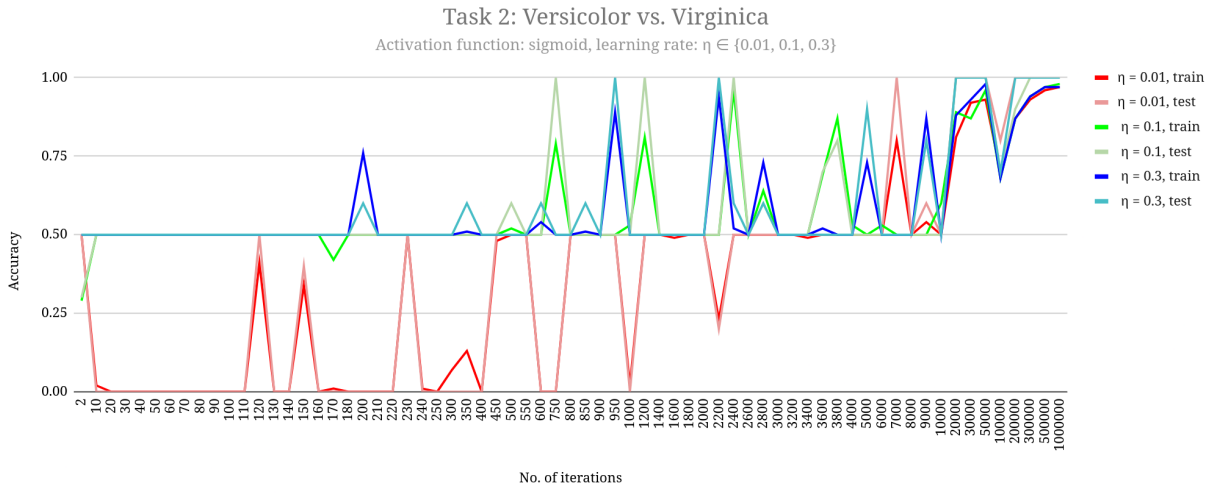


6 pav. 2-os užduoties grafikas nr. 2

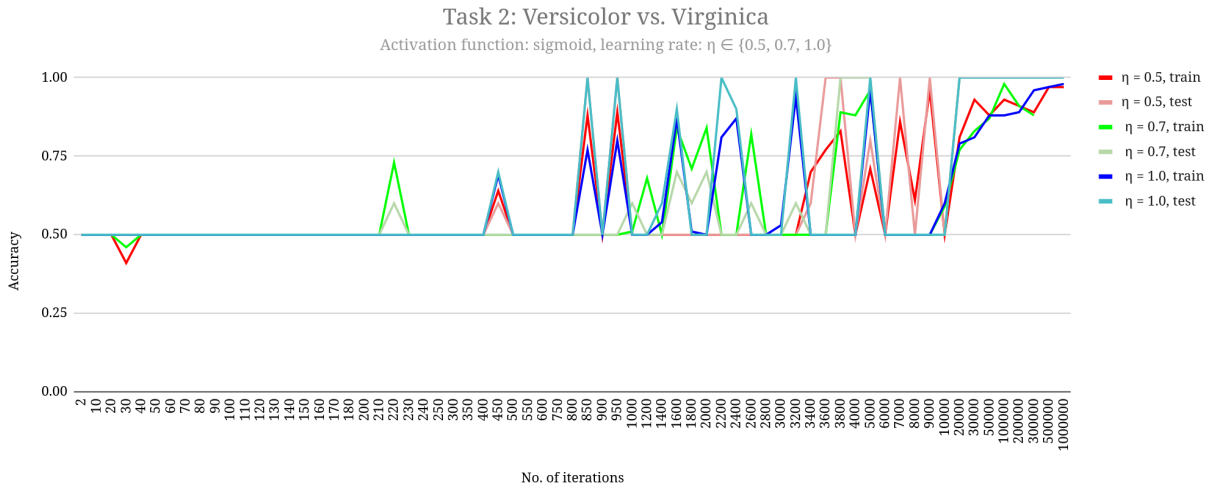
2-oje užduotyje grafikai atrodo kiek kitaip (5 pav., 6 pav.). Stabilus klasifikavimo tikslumo dydis su mokymo greičiais nebuvo pasiektas, išskyrus testavimo aibėms su dalimi mokymo greičių, ir tai tik esant gana dideliam iteracijų skaičiui (20 000).

Slenkstinės f-jos atveju minėtą stabilumą pasiekė testavimo aibės su greičiais 0,01, 0,3 ir 0,5. Taip pat, apskritai pasiekti didesnį klasifikavimo tikslumą prireikė ženkliai daugiau iteracijų nei pirmosios užduoties atveju, ir taip pat *k.t.* reikšmės toliau didėjant iteracijų skaičiui kito, tai sumažėdamos, tai padidėdamos. Stabiliausiai po 20 000 iteracijų elgėsi perceptronas su $\eta = 0{,}01$, nes mažas η daugiklis reiškia mažą mokymosi žingsnį. Tačiau galutinis rezultatas po 1 mln. iteracijų su šiuo η nebuvo geriausias – 0,94 mokymo aibei, aplenkęs tik $\eta = 0{,}7$ (0,88). Tuo tarpu geresnius rezultatus davė $\eta = 0{,}3$ ir $\eta = 0{,}5$, o patį geriausią – $\eta = 1{,}0$, davęs 0,98 tikslumą mokymo aibei po 1 mln. iteracijų, kuris atitinka ir teorinį geriausią šio uždavinio tikslumą, nes mokymo aibėje pasitaikė du duomenys, kurie negalėjo būti tiesiškai atskirti. Testavimo aibėje nepasitaikė tiesiškai neatskiriamų duomenų, ir dėl to su dalimi η jos tikslumas pasiekė 1,0. Taigi geriausias mokymo greitis slenkstinės f-jos atveju galėtų būti $\eta = 1{,}0$ (geriausias mokymo aibės rezultatas), tačiau $\eta = 0{,}3$ nuo 20 000 iteracijų davė stabilesnes *k.t.* reikšmes su mokymo aibe ir stabilios *k.t.* su testavimo aibe, o mokymo aibės tikslumas nusileido tik per 0,01. Taip pat, jei žiūrėti, kurie mokymosi greičiai buvo geresni esant nedideliam iteracijų skaičiui (pvz. iki 3 epochų), tai $\eta = 0{,}5$, 0,7 ir 1,0 yra labiau tinkamesni (2 pav.), tačiau jų *k.t.* išlieka nepastovus.

3.2.1. Aktyvacijos funkcija – sigmoidinė



7 pav. 2-os užduoties grafikas nr. 3



8 pav. 2-os užduoties grafikas nr. 4

Sigmoidinės akt. f-jos atveju rezultatai (7 pav., 8pav.) buvo ganėtinai panašūs. Tačiau su η = 0,01 fluktuacijų buvo daugiausia, ir jau esant dideliam iteracijų skaičiui tos fluktuacijos išliko, priešingai nei slenkstinės akt. f-jos atveju. Geriausią rezultatą davė η = 1,0 (pasiekta 0,98 *k.t.* su mokymo aibe) ir sąlyginai stabilesnės reikšmės. Pirmieji trys mokymo grečiai buvo davė mažiau stabilų *k.t.* nei likusieji trys. Taigi matome, kad sigmoidinės funkcijos atveju mokymosi greičiai gali būti didesni, t.y. ne taip pavojinga žengti didesnį mokymosį žingsnį, nei slenkstinės f-jos atveju.

Palyginant slenkstinę ir sigmoidinę f-jas 2-oje užduotyje, matome, kad su keliais mokymosi greičiais slenkstinė f-ja gali duoti geresnį rezultatą esant mažesniam iteracijų skaičiui (iki 400), tačiau su dideliu iteracijų skaičiumi (20 000 ir daugiau) sigmoidinė funkcija duoda aiškesnę tendenciją – su didesniais mokymo greičiais (η = 0,5, 0,7 ir 1,0) pasiekiama stabilesnė klasifikavimo tikslumo kaita, tokiu būdu leidžiant žengti didesnį mokymo žingsnį ir nesibaiminant dėl padidėjusių fluktuacijų.

Geriausiu 2-os užduoties variantu būtų sigmoidinės funkcijos variantas su mokymo greičiu η = 1,0.

3.3. Geriausi rezultatai

svorius, paklaidas, epochų/iteracijų skaičius, klasifikavimo tikslumo įverčius. Taip pat pateikti duomenų klasifikavimo rezultatus (kokias klases nustatė neuronas);

1-oji užduotis

Geriausi rezultatai buvo nustatyti su šiais parametrais:

mokymosi greitis η = 0,3
aktyvacijos funkcija – sigmoidinė
iteracijų skaičius – 60 ir daugiau.

Gauti svoriai (su 60 iteracijų) = $W_{60} = \ < w_0, w_1, w_2, w_3, w_4 > =$
< -1,11785471, -3,47109311, 4,93795664, 1,95386069, -1,2883658 >

Klasifikavimo tikslumas:
Su mokymo aibe *k.t.* = 1,00
Su testavimo aibe *k.t.* = 1,00

Visos neurono nustatytos klasių reikšmės atitinka mokymo ir testavimo duomenų klasių realias reikšmes (žr. priedą A.2).

2-oji užduotis

Geriausi rezultatai buvo nustatyti su šiais parametrais:
mokymosi greitis η = 1,0
aktyvacijos funkcija – sigmoidinė
iteracijų skaičius – 1 000 000.

Gauti svoriai (su 1 000 000 iteracijų): = $< w_0, w_1, w_2, w_3, w_4 > =$
< -10378,01870443, -9092,62938597, 15088,12011384, 15667,75799848, -9572,44843345 >

Klasifikavimo tikslumas:
Su mokymo aibe *k.t.* = 0,98
Su testavimo aibe *k.t.* = 1,00

Vienam duomeniui iš mokymo aibės klasė buvo nustatyta neteisingai (turėjo būti 1, nustatyta – 0), taip pat vienam duomeniui – turėjo būti 0, nustatyta – 1 (žr. priedą A.3). Testavimo duomenys buvo suklasifikuoti teisingai.

3. Išvados

Neuronas buvo apmokytas spręsti nesudėtingą dviejų klasių klasifikavimo uždavinį, ir atliktas tyrimas, kaip jo klasifikavimo tikslumas priklauso nuo parametrų.

Mokymosi greičio ir klasifikavimo tikslumo priklausomybė nevienareikšmė. Slenkstinės funkcijos atveju didesnis mokymo greitis sąlygojo didesnį mokymo žingsnį, kas sukėlė nestabilius rezultatus 2-ajame uždavinyje kintant iteracijų skaičiui (tiesiškai neatskiriamos klasės). Tuo tarpu sigmoidinės funkcijos atveju galima buvo rinktis didesnį mokymo žingsnį ir negauti didesnių fluktuacijų dėl to.

Mažas mokymo greitis 0,01 labiau tiko slenkstinei funkcijai (daugiau stabilumo), o sigmoidinės funkcijos atveju mažas mokymo greitis stabilumui įtakos neturėjo, ir tik lėtino neurono mokymosi konvergavimą.

Šio tyrimo atveju sigmoidinė funkcija davė geresnį rezultatą, tačiau didele dalimi abi funkcijos davė panašius klasifikavimo rezultatus.

1-oje užduotyje pasiekti aukštą klasifikavimo tikslumą neprireikė daug iteracijų, tačiau 2-oje užduotyje jau jų prireikė žymiai daugiau, ir tai tikslumas vis keitėsi.

Matome, kad kiekvienam uždaviniui visų šių parametrų parinkimas – tai atskiras uždavinys, ir dažnai nustatyti optimalius parametrus galima eksperimentavimo būdu.

A. Priedai

A.1 Python programos kodas

```
#!/usr/bin/env python3

import numpy as np

def debug(text):
    global debug
    if DEBUG_FLAG:
        print(text)


class NeuralNetwork():

    def __init__(self):
        np.random.seed(1)

        # 2x - 1, where x is random number in interval [0.0,1.0),
        # so the values will be in range [-1,1) with equal probability
        self.synaptic_weights = 2 * np.random.random((5, 1)) - 1

    def threshold_fn(self, x):
        return np.heaviside(x, 0)

    def sigmoid_fn(self, x):
        sig = 1 / (1 + np.exp(-x))     # Define sigmoid function
        sig = np.minimum(sig, 0.9999)  # Set upper bound
        sig = np.maximum(sig, 0.0001)  # Set lower bound
        return sig

    def train(self,
              train_inputs,
              train_outputs,
              learning_rate,
              activation_function,
              train_iterations):

        train_inputs_size = train_inputs.shape[0]
        train_outputs_size = train_outputs.shape[0]

        tr_inputs = np.empty((0,5))
        tr_outputs = np.empty((0,1))

        epochs = train_iterations // train_inputs_size
        iterations_left = train_iterations % train_inputs_size
```

```
        for i in range(epochs):
            tr_inputs = np.concatenate((tr_inputs, train_inputs))
            tr_outputs = np.concatenate((tr_outputs, train_outputs))

                                    tr_inputs    =    np.concatenate((tr_inputs,
train_inputs[:iterations_left]))
                                  tr_outputs    =    np.concatenate((tr_outputs,
train_outputs[:iterations_left]))

        size = int(tr_inputs.size / 5)

        for i in range(size):
            tr_input = tr_inputs[:(i+1)]
            tr_inputs = tr_inputs[(i+1):]

            tr_output = tr_outputs[:(i+1)]
            tr_outputs = tr_outputs[(i+1):]

            output = self.think(tr_input, activation_function)
            # error is a signed number meaning the needed change
            error = tr_output - output

            adjustments = np.dot(tr_input.T, error * learning_rate)
            self.synaptic_weights = self.synaptic_weights + adjustments

    def think(self, inputs, activation_function):
        inputs = inputs.astype(float)
        if (activation_function == 'threshold'):
            output = self.threshold_fn(np.dot(inputs, self.synaptic_weights))
        elif (activation_function == 'sigmoid'):
            output = self.sigmoid_fn(np.dot(inputs, self.synaptic_weights))
        else:
            raise ValueError('Incorrect activation function specified')
            exit(1)

        return output


def get_data(task):
    # ================================================================
    # Reading from a file
    # ================================================================

    data = np.genfromtxt('dataset/iris.data', delimiter=',')
    data[:, 4] = 1

    data_setosa = data[:50]
```

```
data_versicolor = data[50:100]
data_virginica = data[100:150]
data_versicolor_and_virginica = data[50:150]

data_train_setosa = data_setosa[:45]
data_test_setosa = data_setosa[45:50]

data_train_versicolor = data_versicolor[:45]
data_test_versicolor = data_versicolor[45:50]

data_train_virginica = data_virginica[:45]
data_test_virginica = data_virginica[45:50]

data_train_versicolor_and_virginica = np.concatenate((
    data_train_versicolor, data_train_virginica))

data_test_versicolor_and_virginica = np.concatenate((
    data_test_versicolor, data_test_virginica))

# ============================================================
# Setting training & test data
# ============================================================

if (task == 1):

    # ------------------------------------------------------------
    # Classification task no.1 (Setosa vs. Versicolor_&_Virginica)
    # ------------------------------------------------------------

    # -------------------- Training data -------------------------

    given_train_inputs = []

    # mix the data (every second element from another class)
    for element in zip(
        data_train_setosa, data_train_versicolor_and_virginica[:45]):

        given_train_inputs.extend(element)

    given_train_inputs = np.concatenate((
        np.array(given_train_inputs),
        data_train_versicolor_and_virginica[45:]))

    # classes should correspond to the input elements
    given_train_outputs = np.zeros(90)

    # every second element is of class 1
    given_train_outputs[1::2] = 1
```

```
    given_train_outputs = np.concatenate((
        given_train_outputs, np.ones(45))).reshape(1, -1).T

    # -------------------- Testing data --------------------------

    given_test_inputs = np.concatenate((
        data_test_setosa,
        data_test_versicolor_and_virginica))

    given_test_outputs = np.concatenate((
        np.zeros(5), np.ones(10))).reshape(1, -1).T

elif (task == 2):

    # ------------------------------------------------------------
    # Classification task no.2 (Versicolor vs. Virginica)
    # ------------------------------------------------------------

    # -------------------- Training data --------------------------

    given_train_inputs = []

    for element in zip(data_train_versicolor, data_train_virginica):
        given_train_inputs.extend(element)

    given_train_inputs = np.array(given_train_inputs)

    given_train_outputs = np.zeros(90)

    # every second element is of class 1
    given_train_outputs[1::2] = 1
    given_train_outputs = given_train_outputs.reshape(1, -1).T

    # -------------------- Testing data --------------------------

    given_test_inputs = np.concatenate((
        data_test_versicolor, data_test_virginica))

    given_test_outputs = np.concatenate((
        np.zeros(5), np.ones(5))).reshape(1, -1).T

else:
    raise ValueError('Incorrect task specified')

return [given_train_inputs,
        given_train_outputs,
        given_test_inputs,
```

```
            given_test_outputs]


def run(init_weights):
    neural_network.synaptic_weights = init_weights

    neural_network.train(given_train_inputs,
                         given_train_outputs,
                         learning_rate,
                         activation_function,
                         iterations)

    train_outputs = neural_network.think(
        given_train_inputs, activation_function)

    test_outputs = neural_network.think(
        given_test_inputs, activation_function)

    # let NaN mean that the item was not assigned to any class
    if activation_function == 'sigmoid':
        train_outputs = np.array([
                 1 if xi > 0.9
             else 0 if xi < 0.1
             else np.nan for xi in train_outputs])

        test_outputs = np.array([
                 1 if xi > 0.9
             else 0 if xi < 0.1
             else np.nan for xi in test_outputs])

    train_results = np.array([sum(x) for x in zip(
        given_train_outputs.flatten(), train_outputs)])

    test_results = np.array([sum(x) for x in zip(
        given_test_outputs.flatten(), test_outputs)])

    # if the sum of the given class and resulting class is 1, that means
    # misclassification has occured.
    # if, however, the sum is nan, the item was not assigned to any class
    # else, the sum values of 0 and 2 means correct classification.
    #
    unique, counts = np.unique(
        train_results[~np.isnan(train_results)], return_counts=True)

    train_results = dict(zip(unique, counts))

    unique, counts = np.unique(
        test_results[~np.isnan(test_results)], return_counts=True)
```

```
    test_results = dict(zip(unique, counts))

    true_assignments_train = \
        train_results.get(0.0, 0) + train_results.get(2.0, 0)

    true_assignments_test = \
        test_results.get(0.0, 0) + test_results.get(2.0, 0)

    train_accuracy = true_assignments_train / len(train_outputs)
    test_accuracy = true_assignments_test / len(test_outputs)

     train_accuracy = ( train_results.get(0.0, 0) + train_results.get(2.0, 0)
) / len(train_outputs)
     test_accuracy = ( test_results.get(0.0, 0) + test_results.get(2.0, 0) ) /
len(test_outputs)

    print(f'{iterations},'
          f'{train_accuracy:.2f},'
          f'{test_accuracy:.2f}')


if __name__ == '__main__':
    # classification task is one of {1,2}
    task = 2

    [given_train_inputs,
     given_train_outputs,
     given_test_inputs,
     given_test_outputs] = get_data(task)

    # ================================================================
    # changeable parameters
    # ================================================================
    # activation function is one of {'threshold', 'sigmoid'}
    activation_function = 'threshold'
    learning_rate = 1.0
    iterations_list = [2,10,20,30,40,50,60,70,80,90,100]
    # ================================================================

    neural_network = NeuralNetwork()
    init_weights = neural_network.synaptic_weights

    iterations = 0
    for i in iterations_list:
        iterations = i
        run(init_weights)
```

A.2 1-osios užduoties įvesties duomenys

Mokymo aibė:

```
[parametras_1 parametras_2 parametras_3 parametras_4] -> reali klasė

[5,1 3,5 1,4 0,2] -> 0
[7,  3,2 4,7 1,4] -> 1
[4,9 3,  1,4 0,2] -> 0
[6,4 3,2 4,5 1,5] -> 1
[4,7 3,2 1,3 0,2] -> 0
[6,9 3,1 4,9 1,5] -> 1
[4,6 3,1 1,5 0,2] -> 0
[5,5 2,3 4,  1,3] -> 1
[5,  3,6 1,4 0,2] -> 0
[6,5 2,8 4,6 1,5] -> 1
[5,4 3,9 1,7 0,4] -> 0
[5,7 2,8 4,5 1,3] -> 1
[4,6 3,4 1,4 0,3] -> 0
[6,3 3,3 4,7 1,6] -> 1
[5,  3,4 1,5 0,2] -> 0
[4,9 2,4 3,3 1, ] -> 1
[4,4 2,9 1,4 0,2] -> 0
[6,6 2,9 4,6 1,3] -> 1
[4,9 3,1 1,5 0,1] -> 0
[5,2 2,7 3,9 1,4] -> 1
[5,4 3,7 1,5 0,2] -> 0
[5,  2,  3,5 1, ] -> 1
[4,8 3,4 1,6 0,2] -> 0
[5,9 3,  4,2 1,5] -> 1
[4,8 3,  1,4 0,1] -> 0
[6,  2,2 4,  1, ] -> 1
[4,3 3,  1,1 0,1] -> 0
[6,1 2,9 4,7 1,4] -> 1
[5,8 4,  1,2 0,2] -> 0
[5,6 2,9 3,6 1,3] -> 1
[5,7 4,4 1,5 0,4] -> 0
[6,7 3,1 4,4 1,4] -> 1
[5,4 3,9 1,3 0,4] -> 0
[5,6 3,  4,5 1,5] -> 1
[5,1 3,5 1,4 0,3] -> 0
[5,8 2,7 4,1 1, ] -> 1
[5,7 3,8 1,7 0,3] -> 0
[6,2 2,2 4,5 1,5] -> 1
[5,1 3,8 1,5 0,3] -> 0
[5,6 2,5 3,9 1,1] -> 1
[5,4 3,4 1,7 0,2] -> 0
```

```
[5,9 3,2 4,8 1,8] -> 1
[5,1 3,7 1,5 0,4] -> 0
[6,1 2,8 4,  1,3] -> 1
[4,6 3,6 1,  0,2] -> 0
[6,3 2,5 4,9 1,5] -> 1
[5,1 3,3 1,7 0,5] -> 0
[6,1 2,8 4,7 1,2] -> 1
[4,8 3,4 1,9 0,2] -> 0
[6,4 2,9 4,3 1,3] -> 1
[5,  3,  1,6 0,2] -> 0
[6,6 3,  4,4 1,4] -> 1
[5,  3,4 1,6 0,4] -> 0
[6,8 2,8 4,8 1,4] -> 1
[5,2 3,5 1,5 0,2] -> 0
[6,7 3,  5,  1,7] -> 1
[5,2 3,4 1,4 0,2] -> 0
[6,  2,9 4,5 1,5] -> 1
[4,7 3,2 1,6 0,2] -> 0
[5,7 2,6 3,5 1, ] -> 1
[4,8 3,1 1,6 0,2] -> 0
[5,5 2,4 3,8 1,1] -> 1
[5,4 3,4 1,5 0,4] -> 0
[5,5 2,4 3,7 1, ] -> 1
[5,2 4,1 1,5 0,1] -> 0
[5,8 2,7 3,9 1,2] -> 1
[5,5 4,2 1,4 0,2] -> 0
[6,  2,7 5,1 1,6] -> 1
[4,9 3,1 1,5 0,1] -> 0
[5,4 3,  4,5 1,5] -> 1
[5,  3,2 1,2 0,2] -> 0
[6,  3,4 4,5 1,6] -> 1
[5,5 3,5 1,3 0,2] -> 0
[6,7 3,1 4,7 1,5] -> 1
[4,9 3,1 1,5 0,1] -> 0
[6,3 2,3 4,4 1,3] -> 1
[4,4 3,  1,3 0,2] -> 0
[5,6 3,  4,1 1,3] -> 1
[5,1 3,4 1,5 0,2] -> 0
[5,5 2,5 4,  1,3] -> 1
[5,  3,5 1,3 0,3] -> 0
[5,5 2,6 4,4 1,2] -> 1
[4,5 2,3 1,3 0,3] -> 0
[6,1 3,  4,6 1,4] -> 1
[4,4 3,2 1,3 0,2] -> 0
[5,8 2,6 4,  1,2] -> 1
[5,  3,5 1,6 0,6] -> 0
[5,  2,3 3,3 1, ] -> 1
[5,1 3,8 1,9 0,4] -> 0
```

```
[5,6 2,7 4,2 1,3] -> 1
[6,3 3,3 6,  2,5] -> 1
[5,8 2,7 5,1 1,9] -> 1
[7,1 3,  5,9 2,1] -> 1
[6,3 2,9 5,6 1,8] -> 1
[6,5 3,  5,8 2,2] -> 1
[7,6 3,  6,6 2,1] -> 1
[4,9 2,5 4,5 1,7] -> 1
[7,3 2,9 6,3 1,8] -> 1
[6,7 2,5 5,8 1,8] -> 1
[7,2 3,6 6,1 2,5] -> 1
[6,5 3,2 5,1 2, ] -> 1
[6,4 2,7 5,3 1,9] -> 1
[6,8 3,  5,5 2,1] -> 1
[5,7 2,5 5,  2, ] -> 1
[5,8 2,8 5,1 2,4] -> 1
[6,4 3,2 5,3 2,3] -> 1
[6,5 3,  5,5 1,8] -> 1
[7,7 3,8 6,7 2,2] -> 1
[7,7 2,6 6,9 2,3] -> 1
[6,  2,2 5,  1,5] -> 1
[6,9 3,2 5,7 2,3] -> 1
[5,6 2,8 4,9 2, ] -> 1
[7,7 2,8 6,7 2, ] -> 1
[6,3 2,7 4,9 1,8] -> 1
[6,7 3,3 5,7 2,1] -> 1
[7,2 3,2 6,  1,8] -> 1
[6,2 2,8 4,8 1,8] -> 1
[6,1 3,  4,9 1,8] -> 1
[6,4 2,8 5,6 2,1] -> 1
[7,2 3,  5,8 1,6] -> 1
[7,4 2,8 6,1 1,9] -> 1
[7,9 3,8 6,4 2, ] -> 1
[6,4 2,8 5,6 2,2] -> 1
[6,3 2,8 5,1 1,5] -> 1
[6,1 2,6 5,6 1,4] -> 1
[7,7 3,  6,1 2,3] -> 1
[6,3 3,4 5,6 2,4] -> 1
[6,4 3,1 5,5 1,8] -> 1
[6,  3,  4,8 1,8] -> 1
[6,9 3,1 5,4 2,1] -> 1
[6,7 3,1 5,6 2,4] -> 1
[6,9 3,1 5,1 2,3] -> 1
[5,8 2,7 5,1 1,9] -> 1
[6,8 3,2 5,9 2,3] -> 1
[6,7 3,3 5,7 2,5] -> 1
```

Testavimo aibė:

```
[4,8 3,  1,4 0,3] -> 0
[5,1 3,8 1,6 0,2] -> 0
[4,6 3,2 1,4 0,2] -> 0
[5,3 3,7 1,5 0,2] -> 0
[5,  3,3 1,4 0,2] -> 0
[5,7 3,  4,2 1,2] -> 1
[5,7 2,9 4,2 1,3] -> 1
[6,2 2,9 4,3 1,3] -> 1
[5,1 2,5 3,  1,1] -> 1
[5,7 2,8 4,1 1,3] -> 1
[6,7 3,  5,2 2,3] -> 1
[6,3 2,5 5,  1,9] -> 1
[6,5 3,  5,2 2, ] -> 1
[6,2 3,4 5,4 2,3] -> 1
[5,9 3,  5,1 1,8] -> 1
```

A.2 2-osios užduoties įvesties duomenys

Mokymo aibė:

```
[parametras_1 parametras_2 parametras_3 parametras_4] -> reali klasė

[7,  3,2 4,7 1,4] -> 0
[6,3 3,3 6,  2,5] -> 1
[6,4 3,2 4,5 1,5] -> 0
[5,8 2,7 5,1 1,9] -> 1
[6,9 3,1 4,9 1,5] -> 0
[7,1 3,  5,9 2,1] -> 1
[5,5 2,3 4,  1,3] -> 0
[6,3 2,9 5,6 1,8] -> 1
[6,5 2,8 4,6 1,5] -> 0
[6,5 3,  5,8 2,2] -> 1
[5,7 2,8 4,5 1,3] -> 0
[7,6 3,  6,6 2,1] -> 1
[6,3 3,3 4,7 1,6] -> 0
[4,9 2,5 4,5 1,7] -> 1
[4,9 2,4 3,3 1, ] -> 0
[7,3 2,9 6,3 1,8] -> 1
[6,6 2,9 4,6 1,3] -> 0
[6,7 2,5 5,8 1,8] -> 1
[5,2 2,7 3,9 1,4] -> 0
[7,2 3,6 6,1 2,5] -> 1
[5,  2,  3,5 1, ] -> 0
[6,5 3,2 5,1 2, ] -> 1
[5,9 3,  4,2 1,5] -> 0
[6,4 2,7 5,3 1,9] -> 1
[6,  2,2 4,  1, ] -> 0
[6,8 3,  5,5 2,1] -> 1
[6,1 2,9 4,7 1,4] -> 0
[5,7 2,5 5,  2, ] -> 1
[5,6 2,9 3,6 1,3] -> 0
[5,8 2,8 5,1 2,4] -> 1
[6,7 3,1 4,4 1,4] -> 0
[6,4 3,2 5,3 2,3] -> 1
[5,6 3,  4,5 1,5] -> 0
[6,5 3,  5,5 1,8] -> 1
[5,8 2,7 4,1 1, ] -> 0
[7,7 3,8 6,7 2,2] -> 1
[6,2 2,2 4,5 1,5] -> 0
[7,7 2,6 6,9 2,3] -> 1
[5,6 2,5 3,9 1,1] -> 0
[6,  2,2 5,  1,5] -> 1
[5,9 3,2 4,8 1,8] -> 0
```

```
[6,9 3,2 5,7 2,3] -> 1
[6,1 2,8 4,  1,3] -> 0
[5,6 2,8 4,9 2, ] -> 1
[6,3 2,5 4,9 1,5] -> 0
[7,7 2,8 6,7 2, ] -> 1
[6,1 2,8 4,7 1,2] -> 0
[6,3 2,7 4,9 1,8] -> 1
[6,4 2,9 4,3 1,3] -> 0
[6,7 3,3 5,7 2,1] -> 1
[6,6 3,  4,4 1,4] -> 0
[7,2 3,2 6,  1,8] -> 1
[6,8 2,8 4,8 1,4] -> 0
[6,2 2,8 4,8 1,8] -> 1
[6,7 3,  5,  1,7] -> 0
[6,1 3,  4,9 1,8] -> 1
[6,  2,9 4,5 1,5] -> 0
[6,4 2,8 5,6 2,1] -> 1
[5,7 2,6 3,5 1, ] -> 0
[7,2 3,  5,8 1,6] -> 1
[5,5 2,4 3,8 1,1] -> 0
[7,4 2,8 6,1 1,9] -> 1
[5,5 2,4 3,7 1, ] -> 0
[7,9 3,8 6,4 2, ] -> 1
[5,8 2,7 3,9 1,2] -> 0
[6,4 2,8 5,6 2,2] -> 1
[6,  2,7 5,1 1,6] -> 0
[6,3 2,8 5,1 1,5] -> 1
[5,4 3,  4,5 1,5] -> 0
[6,1 2,6 5,6 1,4] -> 1
[6,  3,4 4,5 1,6] -> 0
[7,7 3,  6,1 2,3] -> 1
[6,7 3,1 4,7 1,5] -> 0
[6,3 3,4 5,6 2,4] -> 1
[6,3 2,3 4,4 1,3] -> 0
[6,4 3,1 5,5 1,8] -> 1
[5,6 3,  4,1 1,3] -> 0
[6,  3,  4,8 1,8] -> 1
[5,5 2,5 4,  1,3] -> 0
[6,9 3,1 5,4 2,1] -> 1
[5,5 2,6 4,4 1,2] -> 0
[6,7 3,1 5,6 2,4] -> 1
[6,1 3,  4,6 1,4] -> 0
[6,9 3,1 5,1 2,3] -> 1
[5,8 2,6 4,  1,2] -> 0
[5,8 2,7 5,1 1,9] -> 1
[5,  2,3 3,3 1, ] -> 0
[6,8 3,2 5,9 2,3] -> 1
[5,6 2,7 4,2 1,3] -> 0
```

```
[6,7 3,3 5,7 2,5] -> 1
```

Testavimo aibė:

```
[5,7 3,  4,2 1,2] -> 0
[5,7 2,9 4,2 1,3] -> 0
[6,2 2,9 4,3 1,3] -> 0
[5,1 2,5 3,  1,1] -> 0
[5,7 2,8 4,1 1,3] -> 0
[6,7 3,  5,2 2,3] -> 1
[6,3 2,5 5,  1,9] -> 1
[6,5 3,  5,2 2, ] -> 1
[6,2 3,4 5,4 2,3] -> 1
[5,9 3,  5,1 1,8] -> 1
```

A.5 Detalūs tyrimo rezultatai

Pridedamas Microsoft Excel failas *Results.xlsx*