VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Vieno neurono mokymas sprendžiant klasifikavimo uždavinį

Laboratorinis darbas

Atliko: 4 kurso 1 grupės studentas
Paulius Minajevas (SN: 2110599)

Vilnius – 2024

Turinys

Įvadas

Užduoties tikslas – apmokyti vieną neuroną spręsti dviejų klasių klasifikavimo uždavinį ir atlikti tyrimą su dviem duomenų aibėm. Užduoties variantas (0) pasirinktas pagal studento numerio paskutinio skaitmens liekaną iš 3 (mod(9,3) = 0), todėl bus sukurtas sigmoidinis neuronas bei jo apmokymui panaudotas paketinis gradientinis nusileidimas.

1. Klasifikavimo duomenys ir metodika

1.1. Klasifikavimo duomenys

Dirbtiniam neuronui apmokyti naudojamos dvi skirtingos duomenų aibės – irisų bei krūties vėžio. Irisų duomenų rinkinys skirtas atpažinti gėlės rūšį pagal gėlės atributus, o krūties vėžio duomenų rinkinys skirtas nustatyti ar auglys yra piktybinis, pagal skirtingas auglio savybes. Pagal gerąsias praktikas abu duomenų rinkiniai padalinti į dvi dalis pagal proporciją 80/20, kur 80% atitinka mokymosi duomenų kiekį, o 20% testavimo duomenų kiekį.

1.1.1. Irisų duomenų rinkinys

Irisų duomenų rinkinys susidaro iš atributų:

1. ”sepal length” [taurėlapio ilgis]
2. ”sepal width” [taurėlapio plotis]
3. ”petal length” [žiedlapio ilgis]
4. ”petal width” [žiedlapio plotis]
5. ”class” [klasė]

, iš kurių, šiame duomenų rinkinyje, galima atpažinti kelias gėlių rūšis:

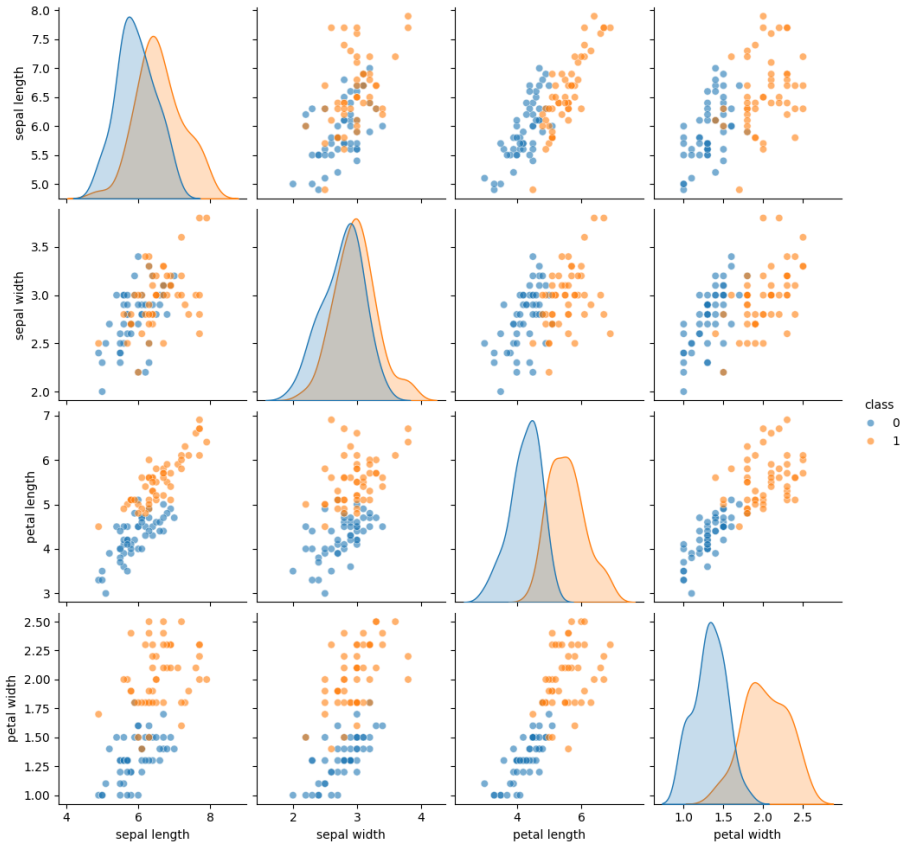
1. Iris Setosa
2. Iris Versicolour
3. Iris Virginica

Kiekvienas pateiktas atributas bus naudojamas apmokant ir tikrinant dirbtinį neuroną, tačiau bus atliekamas binarinis klasifikavimas, todėl visi Iris Setosa duomenys bus pašalinami ir neuronas klasifikuos tik dvi gėlių rūšis – Iris Versicolour, kaip 0 klasę, bei Iris Virginica, kaip 1 klasę.

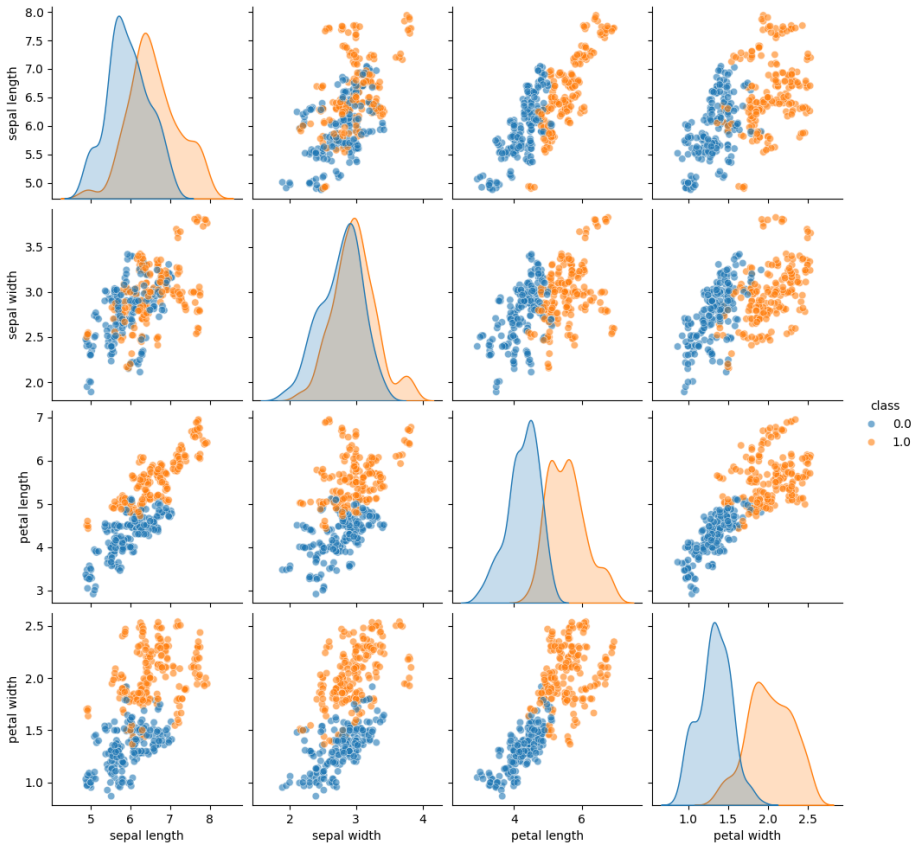
1.1.1.1. Duomenų augmentacija

Šis duomenų rinkinys yra pakankamai mažas, kadangi po Iris Setosa duomenų pašalinimo lieka bendrai 100 duomenų eilučių, iš kurių 50 – Iris Versicolour (0 klasė), bei 50 – Iris Virginica (1 klasė). Dėl šios priežasties duomenų rinkiniui bus pritaikoma duomenų augmentacija ir kiekvienos rūšies duomenys praplėčiami iki 200 eilučių naudojant Gauso triukšmo funkciją.

Žemiau nuotraukose pateikiami duomenų rinkinio atributų porų pasiskirstymo grafikai prieš ir po duomenų augmentacijos:



1 pav. Iris duomenys prieš augmentaciją



2 pav. Iris duomenys po augmentacijos

1.1.2. Krūties vėžio duomenų rinkinys

Krūties vėžio duomenų rinkinys susidaro iš atributų

1. ”Sample code number” [atpažinimo numeris]
2. ”Clump thickness” [gumulėlio storis]
3. ”Uniformity of cell size” [ląstelių dydžio vienodumas]
4. ”Uniformity of cell shape” [ląstelių formos vienodumas]
5. ”Marginal adhesion” [ribinis sukibimas]
6. ”Single epithelial cell size” [vieno epitelio ląstelės dydis]
7. ”Bare nuclei” [pliki branduoliai]
8. ”Bland chromatin” [švelnus chromatinas]
9. ”Normal nucleoli” [normalūs branduoliai]
10. ”Mitoses” [mitozės]
11. ”Class” [klasė]

, iš kurių, šiame duomenų rinkinyje, galima nustatyti ar krūties vėžys piktybinis (1 klasė) ar ne (0 klasė).

Iš šio duomenų rinkinio nebus naudojamas ”Sample code number” atributas, kadangi jis naudojamas duomenų rinkinio eilutės atpažinimui, o ne krūties vėžio tipo nustatymui, todėl šis atributas nėra prasmingas turimam klasifikavimo uždaviniui.

Po kitų nekorektiškų duomenų pašalinimo, liko 683 duomenų eilutės, taigi jokios papildomos duomenų augmentacijos duomenų praplėtimui nebus naudojamos.

1.2. Metodika

Kadangi pasirinktas 0 laboratorinio variantas, todėl sukurtas sigmoidinis neuronas, kurio apmokymui panaudotas paketinis gradientinis nusileidimas. Pradiniai svoriai, abiem duomenų rinkiniams, inicializuoti su nulinėmis reikšmėmis, jų kiekis pasirinktas pagal naudojamų atributų skaičių + 1 (reikalingas papildomas svoris ”bias” nustatymui).

1.2.1. Paketinis gradientinis nusileidimas

Optimizavimo algoritmas, naudojamas dirbtinių neuronų mokyme. Jis skirtas rasti optimalius modelio svorius, minimizuojant klaidos funkciją, remiantis jos gradientu. Šis gradientas nurodo kryptį, kuria reikia keisti modelio svorius (didinti ar mažinti), kad klaidos dydis sumažėtų. Kadangi naudosime paketinį gradientinį nusileidimą, tai prieš svorių atnaujinimą paskaičiuosime gradientą visiems įeities duomenims - epochai, radę šių gradientų vidurkį atnaujinsime modelio svorius vieną kartą per epochą.

1.2.2. Sigmoidinis neuronas

Tai neuronas, kuris gautam neurono rezultatui pritaiko sigmoidinę aktyvacijos funkciją, dėl to gaunamas rezultatas rėžiuose (0,1), kas leidžia neuronui tiksliau mokytis atliekant binarinį klasifikavimą, kadangi tiksliau paskaičiuojamos gautos paklaidos.

2. Programos realizacija ir rezultatai

2.1. Programos kodas

Užduotis atlikta naudojantis „Python" kalba. Tolesni poskyriai išskirti pagal esmines kodo realizacijos grupes.

2.1.1. Dirbtinio neurono modelio realizacija

Žemiau pateikiama sigmoidinio dirbtinio neurono implementacija, aprašanti neurono inicializaciją, treniravimo eigą, testavimo eigą, bei spėjimo atlikimą.

```
class SigmoidNeuron:
    def __init__(self, input_dim, learning_rate=0.01, epochs
        =1000):
        self.weights = np.zeros(input_dim + 1) # + bias
        self.learning_rate = learning_rate
        self.epochs = epochs

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def train(self, X, y, error_threshold=0.05):
        data = X
        X = np.c_[np.ones(X.shape[0]), X] # + x0
        y = np.c_[y]
        total_error = float('inf')
        epoch = 0
        hist_errors = []
        hist_accuracy = []

        while total_error > error_threshold and epoch < self.
            epochs:
            total_error = 0
            gradientSum = np.zeros(self.weights.shape[0])
            for i in range(X.shape[0]):
                a = np.dot(X[i], self.weights)
                y_pred = self.sigmoid(a)
                t = y[i][0]
                for k in range(self.weights.shape[0]):
                    gradientSum[k] = gradientSum[k] + (y_pred - t) *
                        y_pred * (1 - y_pred) * X[i][k]
```

```
            error = (t - y_pred) ** 2
            total_error = total_error + error

        for k in range(self.weights.shape[0]):
            self.weights[k] = self.weights[k] - self.
                learning_rate * (gradientSum[k] / X.shape[0])

        preds = self.predict(data)
        accurate = 0
        for i in range(X.shape[0]):
            if (preds[i] == y[i][0]):
                accurate += 1
        accuracy = accurate/X.shape[0]

        total_error = total_error / X.shape[0]
        hist_errors += [total_error]
        hist_accuracy += [accuracy]
        epoch += 1
        print(f'Epoch: {epoch}, Accuracy: {accuracy} Total
            error: {total_error}')

    return (hist_errors, hist_accuracy)

def predict(self, X):
    X = np.c_[np.ones(X.shape[0]), X]
    result = []

    for i in range(X.shape[0]):
        y_pred = np.dot(X[i], self.weights)
        pred = self.sigmoid(y_pred)
        result += [round(pred)]

    return result

def evaluate(self, X, y):
    data = X
    X = np.c_[np.ones(X.shape[0]), X]
    y = np.c_[y]

    totalError = 0
```

```
        for i in range(X.shape[0]):
          a = 0
          for k in range(self.weights.shape[0]):
            a = a + self.weights[k] * X[i][k]
          y_pred = self.sigmoid(a)
          error = (y[i][0] - y_pred) ** 2
          totalError = totalError + error

        preds = self.predict(data)
        accurate = 0
        for i in range(X.shape[0]):
          if (preds[i] == y[i][0]):
            accurate += 1
        accuracy = accurate/X.shape[0]

        return (accuracy, totalError/X.shape[0])
```

Klasės „SigmoidNeuron" metodas „__init__" atlieka inicializacijos darbą iš parametrų „input_dim" (reikalingų svorių kiekis), „learning_rate" (gradiento žingsnio dydis), bei "epochs" (kiekis, kiek iteracijų bus atlikta per visą duomenų aibę).

Klasės „Neuron" metodas „sigmoid" pritaiko sigmoidinę funkciją. Sigmoidinės funkcijos matematinė išraiška:

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

Klasės „Neuron" metodas „train" atlieka neurono apmokymo žingsnius, svoriai yra naujinami tol kol pasiekiama numatyta epocha arba klaida įgyja minimalią leistiną ribą. Per vieną epochą yra paskaičiuojami gradientai su kiekviena duomenų eilute, kiekvienam svoriui atskirai, epochos pabaigoje gaunamas svorio gradiento vidurkis ir svoris atnaujinamas pagal gradientą ir gradiento žingsnį.

Klasės „Neuron" metodas „predict" atlieka spėjimą pateiktiems duomenims, paskaičiuodamas neurono išeitį, pritaikydamas sigmoidinę aktyvaciją ir suapvalindamas rezultatą į binarinę reikšmę.

Klasės „Neuron" metodas „evaluate" paskaičiuoja klaidos reikšmę testavimo duomenų aibei, tačiau svorių korekcijos neatlieka.

2.1.2. Iris duomenų augmentacija

Skyriuje 1.1.1.1. aprašytos duomenų augmentacijos realizacija, naudojant Gauso triukšmą.

```
import numpy as np
import pandas as pd

def add_gaussian_noise(data, num_samples, mean=0, std_dev=0.05):
    augmented_data = []
```

```
    for _ in range(num_samples):
        noise = np.random.normal(mean, std_dev, (50,4))
        noise = np.column_stack((noise, [0]*50))
        noisy_data = data + noise
        augmented_data.append(noisy_data)
    return pd.DataFrame(np.vstack(augmented_data), columns=data.
        columns)

iris_0 = iris_data.data.original[iris_data.data.original['class']
    == 0]
iris_1 = iris_data.data.original[iris_data.data.original['class']
    == 1]

iris_0_aug = add_gaussian_noise(iris_0, num_samples=3)
iris_1_aug = add_gaussian_noise(iris_1, num_samples=3)

iris_data_aug = pd.concat([iris_0, iris_0_aug, iris_1, iris_1_aug
    ])
iris_data_aug = iris_data_aug.sample(frac=1).reset_index(drop=
    True)

iris_X = iris_data_aug.iloc[:,0:4]
iris_y = iris_data_aug.iloc[:,4:5]
```

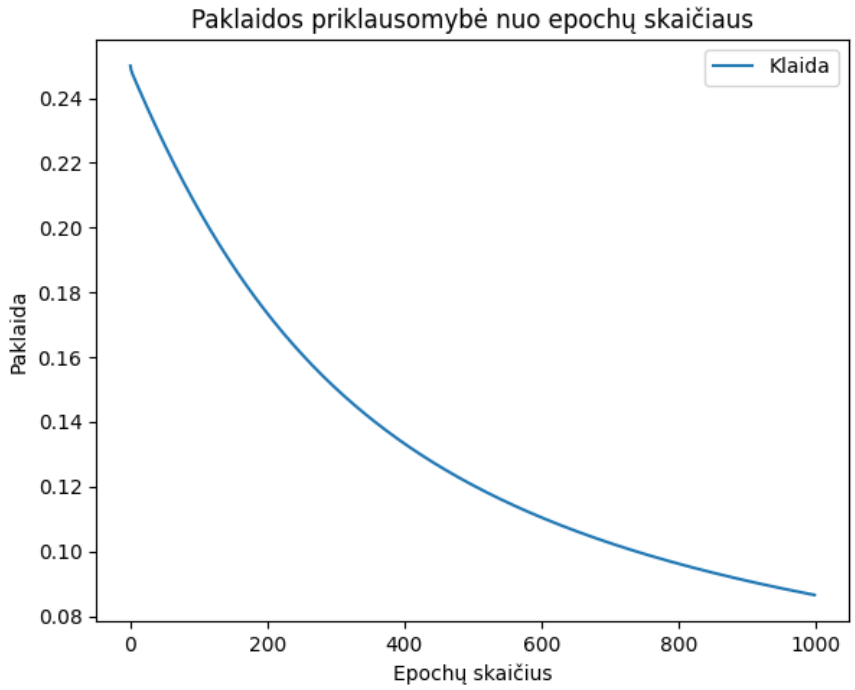
Metodas „add_gaussian_noise" pagal įeities duomenis „data" (duomenys augmentacijai), „num_samples" (kiekis, kiek kartų atlikti augmentaciją perduotam duomenų kiekiui), „mean" (Gauso funckijai reikalingas vidurkis), „std_dev" (Gauso funkcijai reikalingas standartinis nuokrypis), generuoja papildomus duomenis pritaikydamas jiems Gauso triukšmą ir grąžina sąjungą gautų duomenų aibių.

2.2. Tyrimo rezultatai

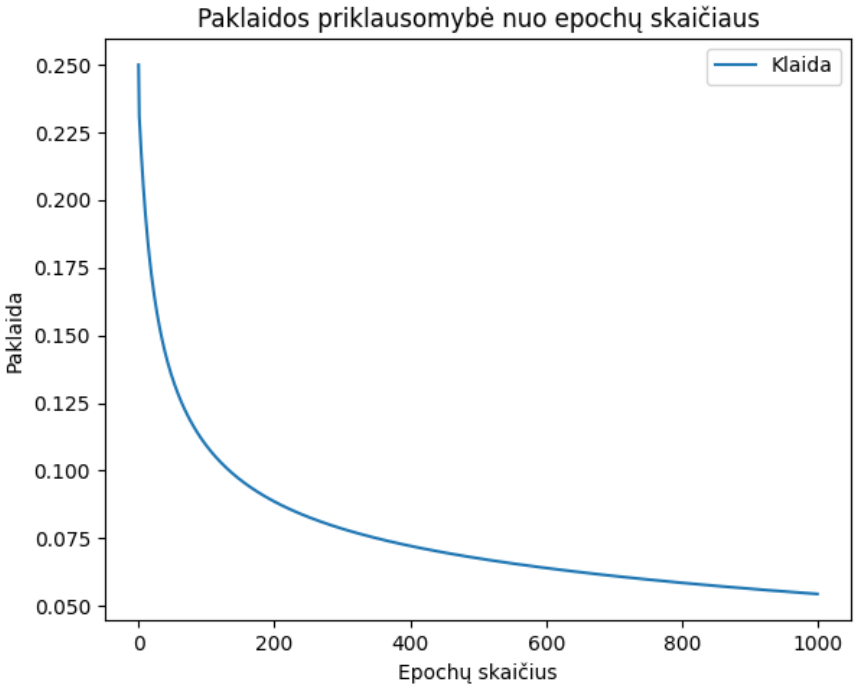
Šiame skyriuje pateikiami tyrimo rezultatai po neurono apmokymo naudojant du skirtingus duomenų rinkinius.

2.2.1. Paklaidos reikšmės priklausomybė nuo epochų skaičiaus

Šio tyrimo metu analizuota kaip kinta paklaida nuo epochų skaičiaus, kurį modelis yra apmokomas. Hiperparametrai naudoti apmokymui: mokymosi žingsnis = 0,01; duomenų padalinimas = 80/20. Žemiau grafikuose pateikiami gauti rezultatai abiems duomenų rinkiniams.



3 pav. Iris: paklaidos nuo epochos skaičiaus tyrimo rezultatai

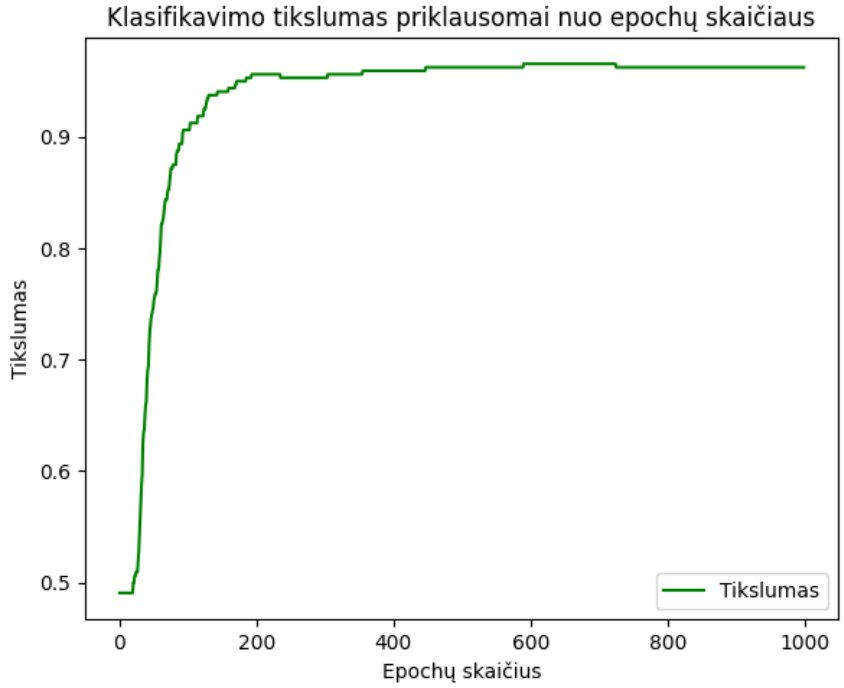


4 pav. Krūties vėžio: paklaidos nuo epochos skaičiaus tyrimo rezultatai

Iš pateiktų grafikų galime daryti išvadą, kad modelio gebėjimas sumažinti paklaidos reikšmę priklauso nuo epochų skaičiaus. Paklaidos reikšmė pradžioje mažėja greičiau, o veliau matomi smulkus patobulėjimai priklausomai nuo epochos reikšmės.

2.2.2. Klasifikavimo tikslumo priklausomybė nuo epochų skaičiaus

Šio tyrimo metu analizuota kaip kinta klasifikavimo tikslumas nuo epochų skaičiaus, kurį modelis yra apmokomas. Hiperparametrai naudoti apmokymui: mokymosi žingsnis = 0,01; duomenų padalinimas = 80/20. Žemiau grafikuose pateikiami gauti rezultatai abiems duomenų rinkiniams.



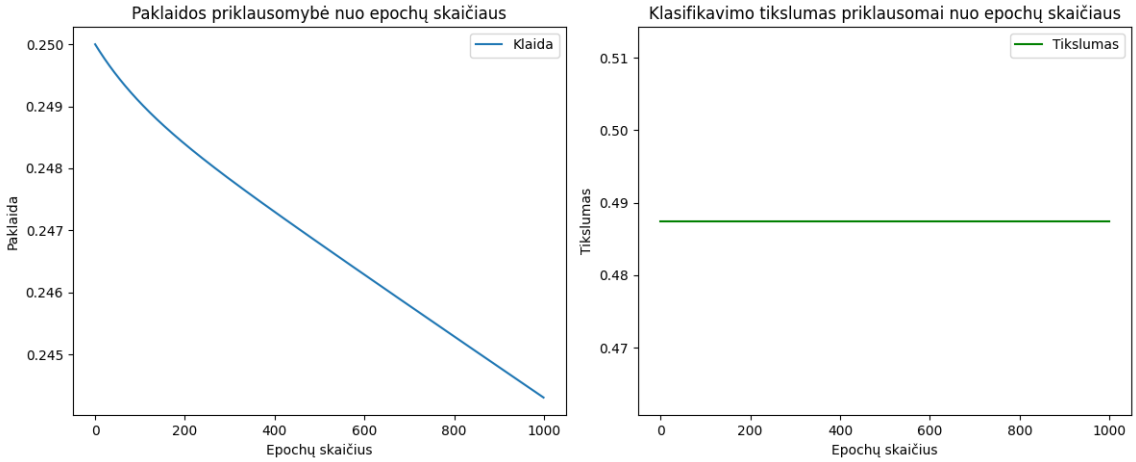
5 pav. Iris: klasifikavimo tikslumo nuo epochos skaičiaus tyrimo rezultatai

6 pav. Krūties vėžio: klasifikavimo tikslumo nuo epochos skaičiaus tyrimo rezultatai

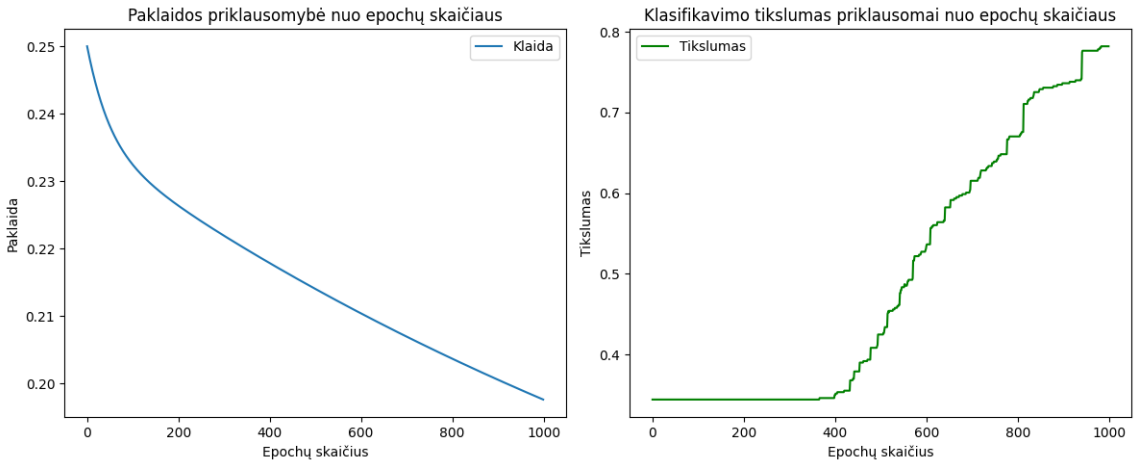
Iš pateiktų grafikų galime daryti išvadą, kad modelio gebėjimas padidinti klasifikavimo tikslumą priklauso nuo epochų skaičiaus. Paklaidos reikšmė pradžioje mažėja dideliais žingsniais, bet Irisų duomenų atveju pasiekia artima maksimaliam tikslumą jau po ∼200 epochų, o krūties vėžio atveju po ∼50 epochų. Taigi tam, kad būtų išnaudota mažiau resursų, mokymosi epochų skaičių būtų galima mažinti daugiau nei per pusę.

2.2.3. Mokymosi greičio įtaką modelio klasifikavimo rezultatams

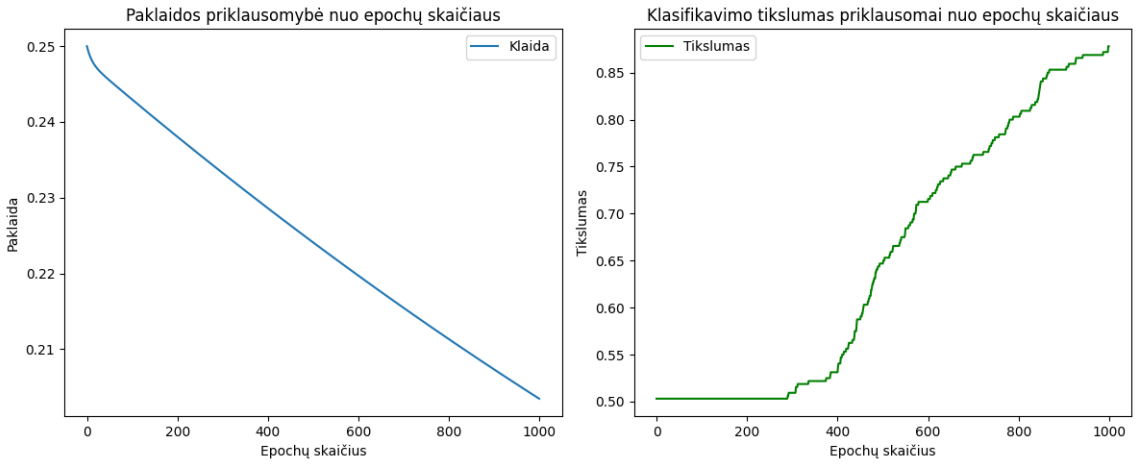
Šio tyrimo metu analizuota kokią įtaką daro mokymosi greitis modelio gebėjimui išmokti teisingai klasifikuoti duomenis. Bus nagrinėjami trys mokymosi greičio dydžiai: 0,001; 0,01; 0,1. Visuomet maksimalus epochų skaičius bus 1000 arba kol modelis pasieks minimalią priimtiną paklaidą - 0,05. Žemiau pateikiami grafikai kiekvienam mokymosi greičiui skirtingiems duomenų rinkiniams, kurių kairėje pusėje - paklaidos kitimo grafikas, o dešinėje - tikslumo kitimo grafikas.



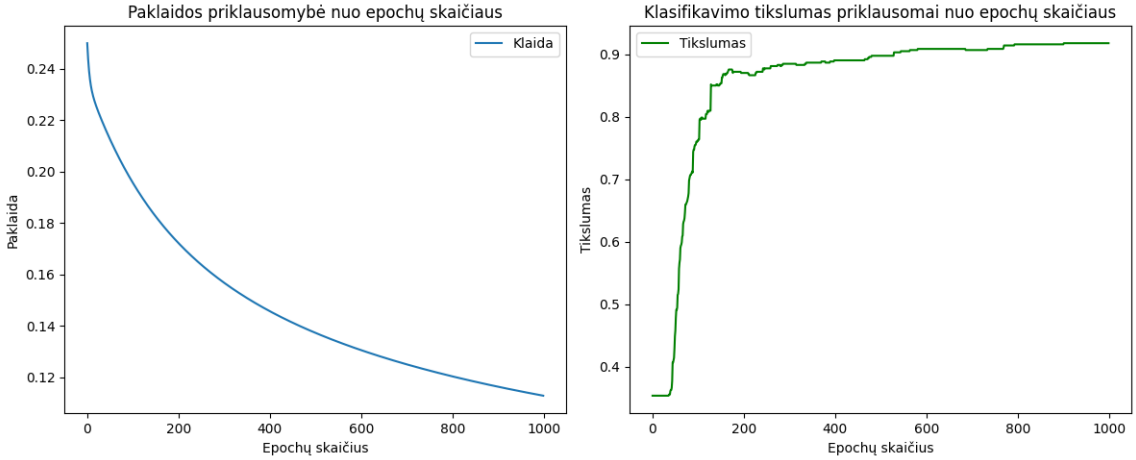
7 pav. Iris: mokymosi greitis 0,001



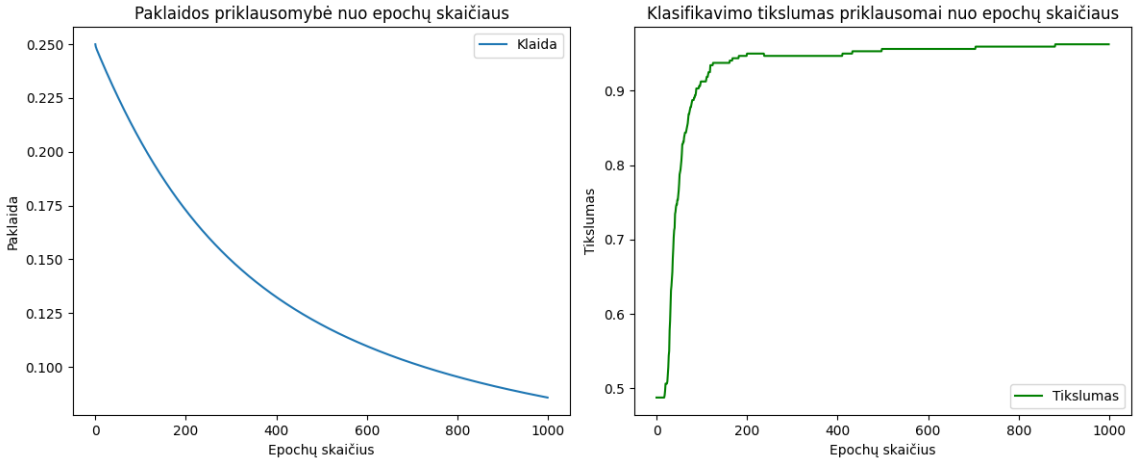
8 pav. Krūties vėžio: mokymosi greitis 0,001



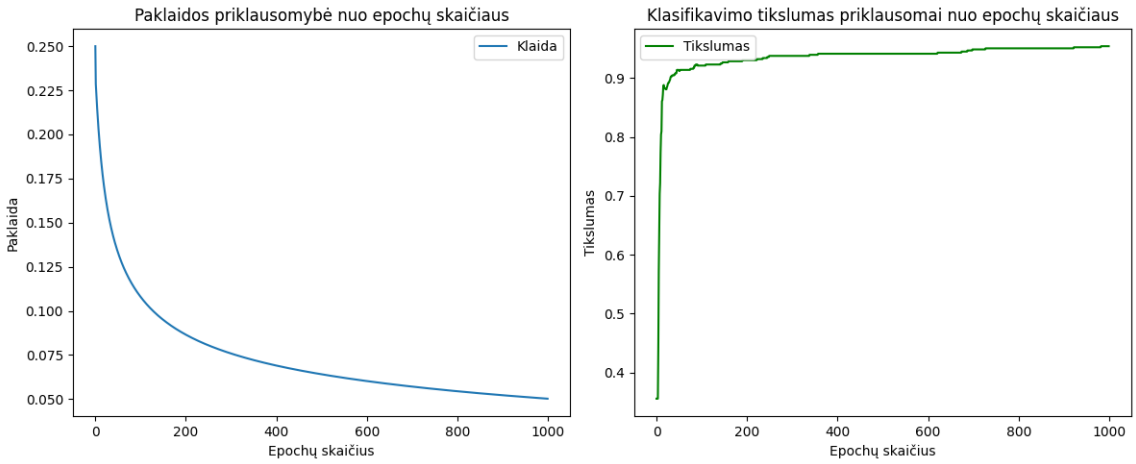
9 pav. Iris: mokymosi greitis 0,01



10 pav. Krūties vėžio: mokymosi greitis 0,01



11 pav. Iris: mokymosi greitis 0,1



12 pav. Krūties vėžio: mokymosi greitis 0,1

Iš pateiktų grafikų galime daryti išvadą, kad priklausomai nuo mokymosi greičio kintą ir modelio paklaidos mažėjimo bei tikslumo augimo greitis. Pavyzdžiui, kai mokymosi greitis buvo 0,001 modelis apie Irisų duomenų rinkinį nepradėjo mokytis, kadangi tikslumas per 1000 epochų nekito, tačiau Krūties vėžio duomenų rinkinys sugebėjo pasiekti net ∼0,8 tikslumą, tam gali turėti įtakos ir duomenų kiekio skirtumas esantis tarp šių dviejų duomenų rinkinių. Kai mokymosi greitis buvo 0,001 arba 0,01 modeliai sugebėjo mažinti savo paklaidą, tačiau jų pradinės ir galutinės paklaidos dydžių deltos buvo mažesnės negu deltos pasiektos su 0,1 mokymosi greičiu abiem duomenų rinkiniams.

2.2.4. Optimalūs hiperparametrai

Atlikus visus tyrimus buvo nustatyti optimalūs hiperparametrai modeliui apmokyti. Žemiau lentelėse pateikiami gauti rezultatai su skirtingais duomenų rinkiniais ir hiperparametrai, kurie buvo panaudoti apmokymui.

1 lentelė. Optimalūs hiperparametrai

Epochų skaičius	Mokymo žingsnis	Duomenų padalijimo proporcija
1000	0,001	80/20

2 lentelė. Iris duomenų rinkiniui gauti svoriai

w_0	w_1	w_2	w_3	w_4
-0,58657047	-1,14447424	-0,89399583	1,65874116	1,37245409

3 lentelė. Krūties vėžio duomenų rinkiniui gauti svoriai

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9
-1,76078028	-0,13550466	0,59973508	0,15256197	0,17523803	-0,39262668	0,46271616	-0,36422973	0,28959368	-0,13059256

4 lentelė. Mokymo ir testavimo rezultatai

Duomenų rinkinys	Paklaida (mokymas)	Tikslumas (mokymas)	Paklaida (testavimas)	Tikslumas (testavimas)
Iris	∼0,084099	0,965625	∼0,096489	0,925
Krūties vėžys	∼0,05177	∼0,950549	∼0,05968	0,941606

Papildomai pridėtame rezultatai.xslx faile yra pateikti modelio spėjamos klasės palyginimai su tiesa iš naudotų testinių duomenų.

Išvados

1. Modelio klasifikavimo tikslumas ir klaidos sumažėjimas tiesiogiai priklauso nuo epochų skaičiaus. Klasifikavimo tikslumas pradžioje didėja reikšmingai, tačiau po tam tikro epochų skaičiaus progresas sulėtėja.
2. Naudojant mažesnį mokymosi greitį, modelio tikslumas didėja lėčiau, tačiau per ilgesnį laiką gali pasiekti panašius arba tikslesnius rezultatus nei naudojant didesnį mokymosi greitį.
3. Optimalus mokymosi greitis buvo nustatytas 0,1, su 1000 epochų, kas leido pasiekti geriausią modelio klasifikavimo tikslumą greičiausiai tiek Iris, tiek Krūties vėžio duomenų rinkiniuose.
4. Iris duomenų rinkinyje klasifikavimo tikslumas siekė apie 92,5%, o Krūties vėžio duomenų rinkinyje – apie 94%, kas rodo modelio efektyvumą sprendžiant šiuos klasifikavimo uždavinius.
5. Paklaidos mažėjimas buvo spartesnis pirmosiomis epochomis, ypač didesnio mokymosi greičio atveju, tačiau vėlesnėse epochose tobulėjimas sulėtėjo.