VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

# Skaitmeninio intelekto dirbtinio neurono mokymo ataskaita

Darbo ataskaita

Atliko: 3 kurso 3 grupės studentas
Hubertas Vindžigalskis

Vilnius – 2025

# Turinys

# 1. Užduoties tikslas

Užduotyje reikia ištreniruoti dirbtinį neuroną remiantis krūties vėžio pacientų duomenimis. Tikslas yra, naudojant pradines įvesties reikšmes, joms rasti svorius, su kuriais galime prognozuoti, ar pacientas serga krūties vėžiu, ar ne (4 ar 2). Taip pat, gale atliksime testus, kuriais nustatysime geriausią būdą treniruoti tokios paskirties modelį.

# 2. Pradiniai duomenys

Duomenys gauti iš Mašininio mokymosi duomenų repozitorijos (https://archive.ics.uci.edu/dataset/15/breast+cancer+wisconsin+original). Pirmas duomenų stulpelis atitinka paciento ID, todėl treniravime jo nenaudojame, sekantys 9 stulpeliai yra požymiai, kuriuos nuodosime kaip įvesties reikšmes, ir paskutinis stulpelis, kuris apibūdina ar paciento turėtas vėžys buvo gerybinis (2) ar piktybinis (4). Iš viso duomenų įrašų: 699.

## 2.1. Duomenų padalinimas į mokymui, validavimo ir testavimo

Duomenys iš pateikto šaltinio buvo padalinti į duomenis skirtiems mokymui, validavimui ir testavimui į atitinkamas dalis: 80%, 10%, 10%.

### 2.1.1. Mokymo duomenys

Šie duomenys skirti atrasti galutiniams svoriams (naudojami modelio svorių optimizavimui).

### 2.1.2. Validavimo duomenys

Naudojami stebėti, kaip modelis veikia mokymo metu, su naujais duomenimis. Jie neįtraukiami į mokymo procesą, taip galutiniai parametrai išlieka neapmokyti šiems duomenims.

### 2.1.3. Testavimo duomenys

Naudojami galutinio modelio našumo įvertinimui, atspindint realias modelio galimybes dirbant su naujais, nematytų duomenų rinkiniais.

## 2.2. Pradiniai svoriai

Pradžioje visi svoriai yra nostatomi 0. Tai suteikia neutralią poziciją modelio mokymui.

# 3. Programos kodas

https://github.com/HubertasVin/digital-intelligence

```
import os
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


# 1. DUOMENU PARUOSIMAS
# 1.1 Nuskaitomi duomenys
def prepare_data(file_path):
    prefix_line = "id,clump_thickness,uniform_cell_size,
       uniform_cell_shape,margin_adhesion,single_cell_size,
       bare_nuclei,bland_chromatin,normal_nucleoli,mitoses,class"
    with open(file_path, "r+") as source:
        content = source.read()
        with open("temp.data", "w") as out:
            out.write(prefix_line.rstrip("\r\n") + "\n" + content
               )
    data = pd.read_csv("temp.data")
    os.remove("temp.data")
    # Pasaliname "id" stulpeli ir eilutes su trukstamomis
       reiksmes
    data.drop("id", axis=1, inplace=True)
    data = data[data["bare_nuclei"] != "?"]
    # Konvertuojame visus stulpelius i skaiciu
    for col in data.columns:
        data[col] = pd.to_numeric(data[col])
    # Sumaisome duomenis
    data = data.sample(frac=1).reset_index(drop=True)
    return data


# 1.2 Padalinti duomenis i mokymo (80%), validavimo (10%) ir
   testavimo (10%) aibes.
def split_data(data, train_ratio=0.8, val_ratio=0.1, test_ratio
   =0.1):
    total = len(data)
    train_end = int(total * train_ratio)
```

```
    val_end = train_end + int(total * val_ratio)
    train = data.iloc[:train_end].reset_index(drop=True)
    val = data.iloc[train_end:val_end].reset_index(drop=True)
    test = data.iloc[val_end:].reset_index(drop=True)
    return train, val, test


# 2. MODELIO TRENIRAVIMAS, VALIDAVIMAS IR TESTAVIMAS
# 2.1 Sigmoidine aktyvacijos funkcija
def sigmoid_function(a):
    return 1 / (1 + np.exp(-a))


# 2.2 Ivertinam modeli
def evaluate_model(data, w):
    features = data.columns.drop("class")
    predictions = []
    true_labels = []
    error_sum = 0
    for _, row in data.iterrows():
        # Pridedame poslinkio reiksme (1) i žpoymiu vektoriu
        x = np.array([1] + [row[f] for f in features])
        a = np.dot(w, x)
        y = sigmoid_function(a)
        prediction = round(y)
        predictions.append(prediction)
        # Konvertuojame originalia klase: 2 -> 0 ir 4 -> 1
        true_label = row["class"] / 2 - 1
        true_labels.append(true_label)
        error_sum += (true_label - y) ** 2
    accuracy = np.mean(np.array(predictions) == np.array(
       true_labels))
    return error_sum, accuracy, predictions, true_labels


# 2.3 Treniruojam neurona fiksuojant mokymo ir validavimo klaidas
    bei tiksluma kiekvienoje epochoje.
def trainBatchWithMetrics(
    train_data, val_data, errorsMin, learningRate=0.05, epochs
       =2000
```

```
):
    features = train_data.columns.drop("class")
    w = np.zeros(
        len(features) + 1
    )  # Inicijuojame svorius ir poslinki (w[0] yra poslinkis)
    train_errors = []
    train_accuracies = []
    val_errors = []
    val_accuracies = []

    for epoch in range(epochs):
        data = train_data.sample(frac=1)
        totalError = 0
        gradientSum = np.zeros(len(features) + 1)
        # Sukaupti gradientai per visus mokymo pavyzdzius
        for _, row in data.iterrows():
            x = np.array([1] + [row[f] for f in features])
            a = np.dot(w, x)
            y = sigmoid_function(a)
            t = (
                row["class"] / 2 - 1
            )  # ėTikslin reiksme: 0 (nepiktybinis) arba 1 (
               piktybinis)
            gradientSum += (y - t) * y * (1 - y) * x
            totalError += (t - y) ** 2
        # Atnaujiname svorius naudodami vidutini gradienta
        w = w - learningRate * (gradientSum / len(data))
        train_errors.append(totalError)
        _, train_acc, _, _ = evaluate_model(train_data, w)
        train_accuracies.append(train_acc)
        val_error, val_acc, _, _ = evaluate_model(val_data, w)
        val_errors.append(val_error)
        val_accuracies.append(val_acc)
        # print(
        #     f"Paketinio GD epocha {epoch}: mokymo klaida = {
           totalError:.4f}, tikslumas = {train_acc:.4f},
           validavimo klaida = {val_error:.4f}, tikslumas = {
           val_acc:.4f}"
        # )
```

```
        # Jei mokymo klaida maziau uz nustatyta riba, stabdome
            mokyma
        if totalError < errorsMin:
            print(
                "Paketinio gradientinio nusileidimo: stabdomas
                    mokymas epochoje", epoch
            )
            epoch += 1  # Skaiciuojame si epocha kaip baigta
            break
    return w, epoch, train_errors, train_accuracies, val_errors,
        val_accuracies


# 2.4 Funkcija "trainStochasticWithMetrics": treniruoja neurona
    naudojant stochastini gradientini nusileidima,
#     fiksuojant mokymo ir validavimo klaidas bei tiksluma
    kiekvienoje epochoje.
def trainStochasticWithMetrics(
    train_data, val_data, errorsMin, learningRate=0.05, epochs
        =10000
):
    features = train_data.columns.drop("class")
    w = np.zeros(len(features) + 1)
    train_errors = []
    train_accuracies = []
    val_errors = []
    val_accuracies = []

    for epoch in range(epochs):
        data = train_data.sample(frac=1)
        totalError = 0
        # Atnaujiname svorius kiekvienam pavyzdziui atskirai
        for _, row in data.iterrows():
            x = np.array([1] + [row[f] for f in features])
            a = np.dot(w, x)
            y = sigmoid_function(a)
            t = row["class"] / 2 - 1
            for j in range(len(w)):
                w[j] = w[j] - learningRate * (y - t) * y * (1 - y
                    ) * x[j]
```

```
            totalError += (t - y) ** 2
        train_errors.append(totalError)
        _, train_acc, _, _ = evaluate_model(train_data, w)
        train_accuracies.append(train_acc)
        val_error, val_acc, _, _ = evaluate_model(val_data, w)
        val_errors.append(val_error)
        val_accuracies.append(val_acc)
        # print(
        #     f"Stochastinio GD epocha {epoch}: mokymo klaida = {
            totalError:.4f}, tikslumas = {train_acc:.4f},
            validavimo klaida = {val_error:.4f}, tikslumas = {
            val_acc:.4f}"
        # )
        if totalError < errorsMin:
            print(
                "Stochastinio gradientinio nusileidimo: stabdomas
                    mokymas epochoje",
                epoch,
            )
            epoch += 1
            break
    return w, epoch, train_errors, train_accuracies, val_errors,
        val_accuracies


# 3. TYRIMU ATLIKIMAS IR REZULTATU VIZUALIZACIJA
if __name__ == "__main__":
    # 3.1 Ikeliame ir paruosime duomenis
    data = prepare_data("breast-cancer.data")
    print("Duomenu irasu skaicius po valymo:", len(data))

    # 3.2 Padaliname duomenis i mokymo (80%), validavimo (10%) ir
        testavimo (10%) aibes
    train_set, val_set, test_set = split_data(data, 0.8, 0.1,
        0.1)
    print(
        "Mokymo aibe:",
        len(train_set),
        "Validavimo aibe:",
        len(val_set),
```

```
    "Testavimo aibe:",
    len(test_set),
)

# 3.3 Eksperimentas 1: Mokymas naudojant paketini gradientini
    nusileidima
print("\nMokymas naudojant paketinio gradientinio nusileidimo
    metoda...")
start_time = time.time()
(
    w_batch,
    epochs_batch,
    train_err_batch,
    train_acc_batch,
    val_err_batch,
    val_acc_batch,
) = trainBatchWithMetrics(
    train_set, val_set, errorsMin=12, learningRate=0.05,
       epochs=500
)
batch_time = time.time() - start_time
test_err_batch, test_acc_batch, test_pred_batch,
   test_true_batch = evaluate_model(
    test_set, w_batch
)

print("\nRezultatai naudojant paketini gradientini
   nusileidima:")
print("Galutiniai svoriai ir poslinkis:", w_batch)
print("Epochos skaicius:", epochs_batch)
print(
    "Paskutines epochos mokymo klaida:",
    train_err_batch[-1],
    "Mokymo tikslumas:",
    train_acc_batch[-1],
)
print(
    "Paskutines epochos validavimo klaida:",
    val_err_batch[-1],
    "Validavimo tikslumas:",
```

```
    val_acc_batch[-1],
)
print("Testavimo klaida:", test_err_batch, "Testavimo
    tikslumas:", test_acc_batch)
print("Mokymo laikas (s):", batch_time)

# Isspausdiname prognozes kiekvienam testavimo pavyzdziui
print("\nPaketinio GD: Testavimo pavyzdziu prognozes:")
for i, (pred, true_val) in enumerate(zip(test_pred_batch,
    test_true_batch)):
    print(f"Pavyzdys {i+1}: Prognoze = {pred}, Tiesa = {int(
        true_val)}")

# Pavaizduojame mokymo ir validavimo klaidu priklausomybe nuo
    epochu (paketinis metodas)
plt.figure()
plt.plot(range(len(train_err_batch)), train_err_batch, label
    ="Mokymo klaida")
plt.plot(range(len(val_err_batch)), val_err_batch, label="
    Validavimo klaida")
plt.xlabel("Epochos")
plt.ylabel("Klaida (kvadratiniu klaidu suma)")
plt.title("Paketinis GD: Klaida nuo epochu")
plt.legend()
plt.show()

# Pavaizduojame mokymo ir validavimo tikslumo priklausomybe
    nuo epochu (paketinis metodas)
plt.figure()
plt.plot(range(len(train_acc_batch)), train_acc_batch, label
    ="Mokymo tikslumas")
plt.plot(range(len(val_acc_batch)), val_acc_batch, label="
    Validavimo tikslumas")
plt.xlabel("Epochos")
plt.ylabel("Tikslumas")
plt.title("Paketinis GD: Tikslumas nuo epochu")
plt.legend()
plt.show()
```

```
# 3.4 Eksperimentas 2: Mokymas naudojant stochastini
    gradientini nusileidima
print("\nMokymas naudojant stochastini gradientini
    nusileidima...")
start_time = time.time()
(
    w_stoch,
    epochs_stoch,
    train_err_stoch,
    train_acc_stoch,
    val_err_stoch,
    val_acc_stoch,
) = trainStochasticWithMetrics(
    train_set,
    val_set,
    errorsMin=12,
    learningRate=0.05,
    epochs=10000,
)
stoch_time = time.time() - start_time
test_err_stoch, test_acc_stoch, test_pred_stoch,
    test_true_stoch = evaluate_model(
    test_set, w_stoch
)

print("\nRezultatai naudojant stochastini gradientini
    nusileidima:")
print("Galutiniai svoriai ir poslinkis:", w_stoch)
print("Epochos skaicius:", epochs_stoch)
print(
    "Paskutines epochos mokymo klaida:",
    train_err_stoch[-1],
    "Mokymo tikslumas:",
    train_acc_stoch[-1],
)
print(
    "Paskutines epochos validavimo klaida:",
    val_err_stoch[-1],
    "Validavimo tikslumas:",
    val_acc_stoch[-1],
```

```
)
print("Testavimo klaida:", test_err_stoch, "Testavimo
    tikslumas:", test_acc_stoch)
print("Mokymo laikas (s):", stoch_time)

# Isspausdiname prognozes kiekvienam testavimo pavyzdziui (
    prognozuota vs. tikra klase)
print("\nStochastinio GD: Testavimo pavyzdziu prognozes:")
for i, (pred, true_val) in enumerate(zip(test_pred_stoch,
    test_true_stoch)):
    print(f"Pavyzdys {i+1}: Prognoze = {pred}, Tiesa = {int(
        true_val)}")

# Pavaizduojame mokymo ir validavimo klaidu priklausomybe nuo
    epochu (stochastinis metodas)
plt.figure()
plt.plot(range(epochs_stoch), train_err_stoch, label="Mokymo
    klaida")
plt.plot(range(epochs_stoch), val_err_stoch, label="
    Validavimo klaida")
plt.xlabel("Epochos")
plt.ylabel("Klaida (kvadratiniu klaidu suma)")
plt.title("Stochastinis GD: Klaida nuo epochu")
plt.legend()
plt.show()

# Pavaizduojame mokymo ir validavimo tikslumo priklausomybe
    nuo epochu (stochastinis metodas)
plt.figure()
plt.plot(range(epochs_stoch), train_acc_stoch, label="Mokymo
    tikslumas")
plt.plot(range(epochs_stoch), val_acc_stoch, label="
    Validavimo tikslumas")
plt.xlabel("Epochos")
plt.ylabel("Tikslumas")
plt.title("Stochastinis GD: Tikslumas nuo epochu")
plt.legend()
plt.show()
```

```
# 3.5 Eksperimentas 3: Itaka mokymosi greiciui (paketinis
   metodas)
learning_rates = [0.01, 0.05, 0.1]
batch_results = {}

print("\nIvertiname skirtingus mokymosi greicius (paketinis
   metodas)...")
# Paleidziame eksperimenta su skirtingomis mokymosi greicio
   reiksmes
for lr in learning_rates:
    start = time.time()
    w_tmp, epochs_tmp, train_err_tmp, train_acc_tmp,
       val_err_tmp, val_acc_tmp = (
        trainBatchWithMetrics(
            train_set,
            val_set,
            errorsMin=12,
            learningRate=lr,
            epochs=500,
        )
    )
    elapsed = time.time() - start
    test_err_tmp, test_acc_tmp, _, _ = evaluate_model(
       test_set, w_tmp)
    batch_results[lr] = {
        "epochs": epochs_tmp,
        "train_error": train_err_tmp[-1],
        "train_accuracy": train_acc_tmp[-1],
        "val_error": val_err_tmp[-1],
        "val_accuracy": val_acc_tmp[-1],
        "test_error": test_err_tmp,
        "test_accuracy": test_acc_tmp,
        "time": elapsed,
    }
    print(
        f"Mokymosi greitis {lr}: Testavimo tikslumas = {
           test_acc_tmp:.4f}, Mokymo laikas = {elapsed:.4f} s
           "
    )
```

```
    # Stulpeline diagrama, palyginanti testavimo tiksluma
       skirtingiems mokymosi greiciams (paketinis metodas)
    plt.figure()
    plt.bar(
        [str(lr) for lr in learning_rates],
        [batch_results[lr]["test_accuracy"] for lr in
           learning_rates],
    )
    plt.xlabel("Mokymosi greitis")
    plt.ylabel("Testavimo tikslumas")
    plt.title("Paketinis GD: Testavimo tikslumas nuo mokymosi
       greicio")
    plt.show()
```

## 3.1. Paketinis ir stochastinis gradientiniai nusileidimai

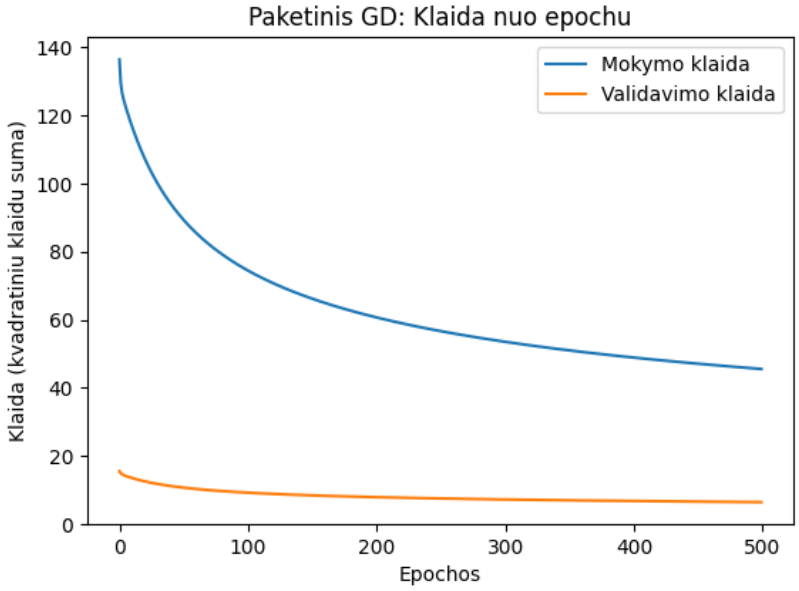
**Paketinis gradientinis nusileidimas** – atnaujina svorius su visais duomenimis vienu metu. Epocha reiškia vieną ciklą, kurio metu peržiūrimi visi mokymo duomenys, o svoriai atnaujinami vienu bendru žingsniu pagal vidutinį gradientą.

**Stochastinis gradientinis nusileidimas** – atnaujina svorius prie kiekvieno duomenų įrašo atskirai. Epochoje panaudojami visi duomenų įrašai, tačiau svoriai atnaujinami po kiekvienos iteracijos, todėl epochoje atliekama tiek atnaujinimų, kiek yra mokymo duomenų.

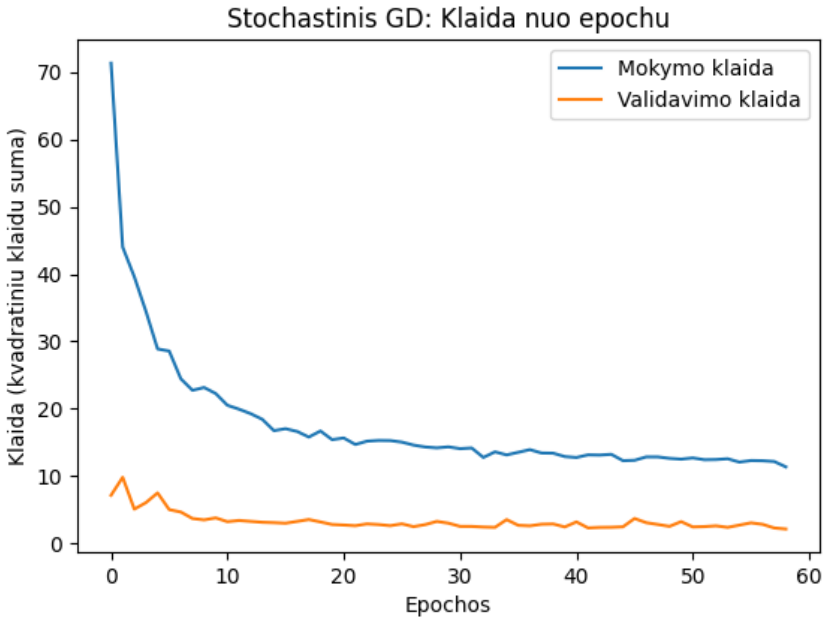
# 4. Tyrimas

## 4.1. Paklaidos reikšmės priklausomybė nuo epochų skaičiaus

Mokymo ir validavimo metu buvo vykdomas tyrimas, kurio tikslas atvaizduoti, kaip paklaida priklauso nuo epochos. 1 pav. vaizduoja paklaidos priklausomybę nuo epochos taikant paketinio gradientinio nusileidimo, o 2 pav. naudojant stochastinio gradientinio nusileidimo algoritmus.



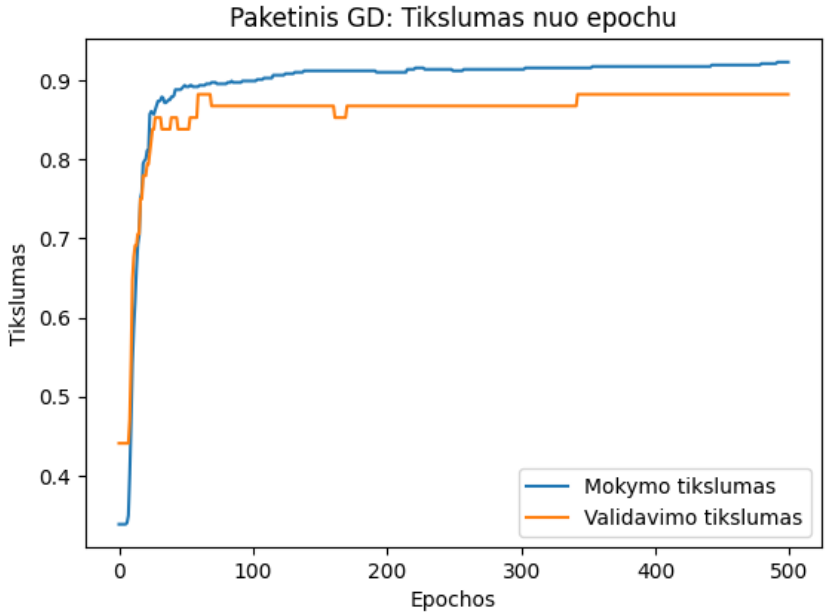

1 pav. Paklaidos priklausomybė nuo epochos naudojant paketinį gradientinį nusileidimą



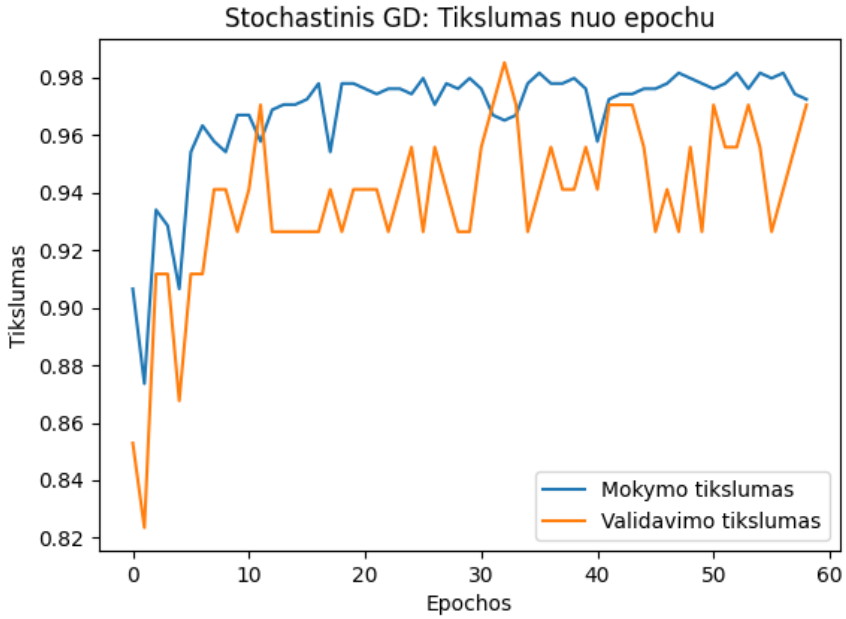

2 pav. Paklaidos priklausomybė nuo epochos naudojant stochastinį gradientinį nusileidimą

## 4.2. Klasifikavimo tikslumo priklausomybė nuo epochų skaičiaus

Mokymo ir validavimo metu taip pat rinkome tikslumą po kiekvienos epochos ir duomenis atvaizdavome grafikuose. 3 pav. vaizduoja tikslumo priklausomybę nuo epochos taikant paketinio gradientinio nusileidimo, o 4 pav. naudojant stochastinio gradientinio nusileidimo algoritmus.



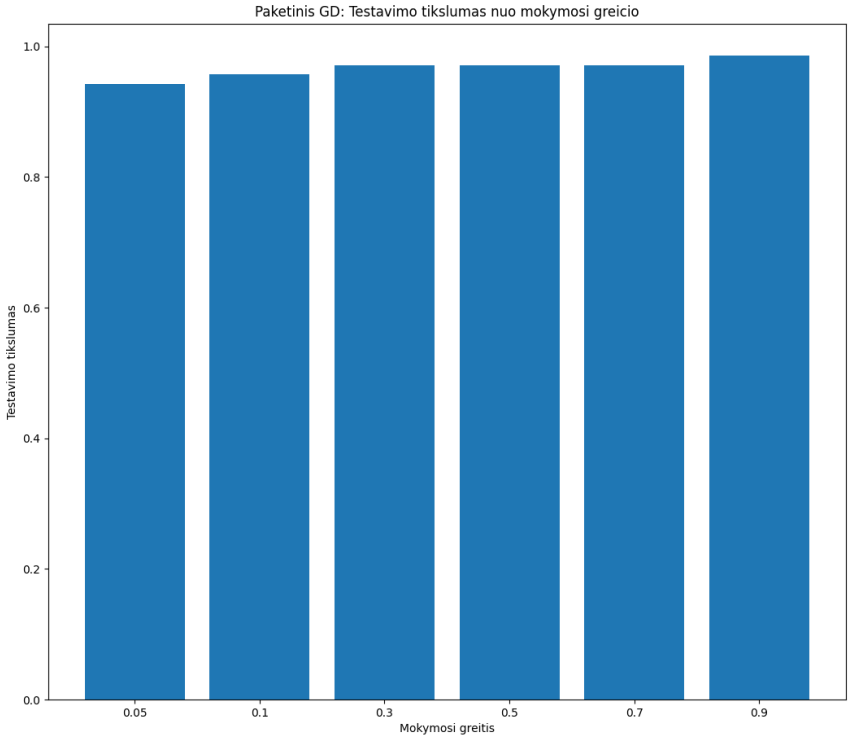

3 pav. Tikslomo priklausomybė nuo epochos naudojant paketinį gradientinį nusileidimą




4 pav. Tikslomo priklausomybė nuo epochos naudojant stochastinį gradientinį nusileidimą

## 4.3. Rezultatų priklausomybė nuo skirtingo mokymosi greičio

Atlikome tyrimą, kurio tikslas atrasti optimaliausią mokymosi greitį mokant modelį paketinio gradientinio nusileidimo algoritmu ir atvaizdavome grafike 5 pav.. Matome, kad geriausias rezultatas gautas su greičiu 0,9.




5 pav. Rezultatų priklausomybė nuo mokymosi greičių: 0,05; 0,1; 0,3; 0,5; 0,7; 0,9

## 4.4. Rezultatų priklausomybė nuo taikomo gradientinio nusileidimo tipo

Paketinis gradientinis nusileidimas pasižymi stabilesne konvergencija.

Stochastinis gradientinis nusileidimas greičiau pasiekia mažesnę paklaidą.

Paketinis metodas yra tinkamesnis, kai norima stabilumo, o stochastinis - kai reikia rezultatų naudojant mažiau epochų.

## 4.5. Mokymo laiko priklausomybė nuo taikomo gradientinio nusileidimo tipo

Paketinis gradientinis nusileidimas yra lėtesnis, nes kiekvienoje epochoje atliekami skaičiavimai su visa mokymo duomenų aibe.

Stochastinis gradientinis nusileidimas gali būti greitesnis, nes kiekviename žingsnyje skaičiuojama tik maža duomenų dalis, o ne visas duomenų rinkinys.

## 4.6. Rezultatai

Geriausi rodikliai pasiekti naudojant stochastinį gradientinį nusileidimą:

**Gauti svoriai**: [-5,83240845; 0,19159356; 0,50333383; 0,17241633; 0,0798375; -0,02853188; 0,38048123; 0,05091436; 0,17209634; 0,17557821] (pirmas elementas yra poslinkis)

**Epochų skaičius**: 59

**Paklaidos paskutinėje epochoje mokymo ir validavimo duomenyse**:

Mokymo klaida: 11,34923

Mokymo tikslumas: 0,97252

**Tikslumas paskutinėje epochoje mokymo ir validavimo duomenimyse**:

Validavimo klaida: 2,11851

Validavimo tikslumas: 0,97058

**Paklaidą su testavimo duomenimis**: 1,92055

**Tikslumas su testavimo duomenimis** 0,95652

**Kiekvieno testavimo duomenų įrašo nustatytos klasės ir originalios klasės**:

1. Prognozė = 0, Tiesa = 1
2. Prognozė = 0, Tiesa = 0
3. Prognozė = 0, Tiesa = 0
4. Prognozė = 1, Tiesa = 1
5. Prognozė = 0, Tiesa = 0
6. Prognozė = 0, Tiesa = 0
7. Prognozė = 1, Tiesa = 1
8. Prognozė = 0, Tiesa = 1
9. Prognozė = 0, Tiesa = 0
10. Prognozė = 0, Tiesa = 0
11. Prognozė = 0, Tiesa = 0
12. Prognozė = 1, Tiesa = 1
13. Prognozė = 0, Tiesa = 0
14. Prognozė = 0, Tiesa = 0
15. Prognozė = 0, Tiesa = 0
16. Prognozė = 1, Tiesa = 1
17. Prognozė = 0, Tiesa = 0
18. Prognozė = 1, Tiesa = 1
19. Prognozė = 0, Tiesa = 0
20. Prognozė = 0, Tiesa = 0
21. Prognozė = 0, Tiesa = 0
22. Prognozė = 1, Tiesa = 1
23. Prognozė = 0, Tiesa = 0
24. Prognozė = 0, Tiesa = 0
25. Prognozė = 0, Tiesa = 0
26. Prognozė = 1, Tiesa = 1
27. Prognozė = 0, Tiesa = 0
28. Prognozė = 1, Tiesa = 1
29. Prognozė = 0, Tiesa = 0
30. Prognozė = 0, Tiesa = 0
31. Prognozė = 0, Tiesa = 0
32. Prognozė = 1, Tiesa = 1

33. Prognozė = 0, Tiesa = 0
34. Prognozė = 0, Tiesa = 0
35. Prognozė = 0, Tiesa = 0
36. Prognozė = 0, Tiesa = 0
37. Prognozė = 0, Tiesa = 0
38. Prognozė = 0, Tiesa = 0
39. Prognozė = 0, Tiesa = 0
40. Prognozė = 0, Tiesa = 0
41. Prognozė = 0, Tiesa = 0
42. Prognozė = 0, Tiesa = 0
43. Prognozė = 1, Tiesa = 1
44. Prognozė = 1, Tiesa = 1
45. Prognozė = 0, Tiesa = 0
46. Prognozė = 1, Tiesa = 1
47. Prognozė = 1, Tiesa = 1
48. Prognozė = 1, Tiesa = 1
49. Prognozė = 1, Tiesa = 1
50. Prognozė = 1, Tiesa = 1
51. Prognozė = 0, Tiesa = 0
52. Prognozė = 0, Tiesa = 0
53. Prognozė = 0, Tiesa = 0
54. Prognozė = 0, Tiesa = 0
55. Prognozė = 0, Tiesa = 1
56. Prognozė = 0, Tiesa = 0
57. Prognozė = 1, Tiesa = 1
58. Prognozė = 1, Tiesa = 1
59. Prognozė = 0, Tiesa = 0
60. Prognozė = 0, Tiesa = 0
61. Prognozė = 0, Tiesa = 0
62. Prognozė = 0, Tiesa = 0
63. Prognozė = 1, Tiesa = 1
64. Prognozė = 0, Tiesa = 0
65. Prognozė = 1, Tiesa = 1
66. Prognozė = 0, Tiesa = 0
67. Prognozė = 0, Tiesa = 0
68. Prognozė = 0, Tiesa = 0
69. Prognozė = 1, Tiesa = 1

# 5. Išvada

Dokumente aprašytas dirbtinio neurono mokymo procesas naudojant paketinio ir stochastinio gradientinių nusileidimų algoritmus. Šis neuronas skirtas atskirti gerybinį ir piktybinį krūties vėžį. Atliktas tyrimas, kuris parodė, kuris algoritmas yra efektyvesnis mokant neuroną, ir gautas rezultatas, jog naudojant stochastinį gradientinį nusileidimą, svoriai gaunami greičiau. Pastebėjome, kad neurono tikslumas tobulėja labai sparčiai per pirmas 50 epochų, o vėliau labai stipriai sulėtėja. Taip pat, tyrimo būdu sužinojome, jog mokymosi greitis beveik neturi įtakos rezultatams.