VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Vieno neurono mokymas sprendžiant klasifikavimo uždavinį

Praktinio darbo ataskaita

Atliko: Justinas Bliujus

VU el. p.: justinas.bliujus@mif.stud.vu.lt

Vertino:

Vilnius

2025

1. SKYRIUS

Šiame darbe aprašomas dirbtinio neurono apmokymas sprendžiant klasifikavimo uždavinį bei atlikti tyrimai.

1.1. Tikslas

Apmokyti vieną neuroną spręsti dviejų klasių uždavinį ir atlikti tyrimą.

1.2. Naudojami duomenys

Užduočiai atlikti naudojami duoti duomenys. Duomenys apibūdina klientų tyrimų rezultatus ieškant krūties vėžio. Pradiniai duomenys susideda iš vienuolikos kintamųjų. Jeigu yra tuščių reikšmių, tokia duomenų eilutė pašalinama. Duomenys paruošiami, pirmąjį kintamąjį – identifikacinį numerį, ištrinant. Paskutinis kintamasis – klasė, pakeičiamas, kai reikšmė 2 į reikšmę 1, kai vėžio nėra. Kai reikšmė 4 pakeičiama į 1, kai vėžys yra. Taigi sutvarkyti duomenys susideda iš devynių apibūdinamųjų kintamųjų ir vienos klasės.

1.3. Duomenų padalijimas į tris dalis

Duomenų aibė sumaišoma ir padalijama į tris dalis. 80% duomenų skirti apmokyti neurono modelį. 10% duomenų skirti validavimui, tai yra tikrinti, kaip apmokytas modelis sugeba išspręsti klasių uždavinį, tinkinti parametrams, kad būtų gautas optimalus modelis. Paskutiniai 10% duomenų skirti testavimui, tai yra modelio panaudojimui. Šiuo atveju apmokymo aibė susideda iš 546 duomenų eilučių, kur kiekvienoje yra 9 požymiai ir 1 klasė. Validavimo aibė susideda iš 68 eilučių, testavimo iš 69 eilučių.

1.4. Pradinių reikšmių parinkimas

Pradinės svorių reikšmės parenkamos atsitiktinai iš intervalo (-1;1). Pradinis parinktas epochų skaičius yra 2000, mokymosi greitis 0,001, mokymosi slenkstis 0,0001. Epochų skaičius ir slenkstis reikalingi mokymosi sustabdymui. Mokymasis stabdomas po 2000 epochų arba pasiekus norimą rezultatą. Mokymosi greitis yra žingsnio dydis antigradiento kryptimi, tai yra kaip greitai mokosi modelis.

1.5. Paketinis ir stochastinis gradientinis nusileidimas

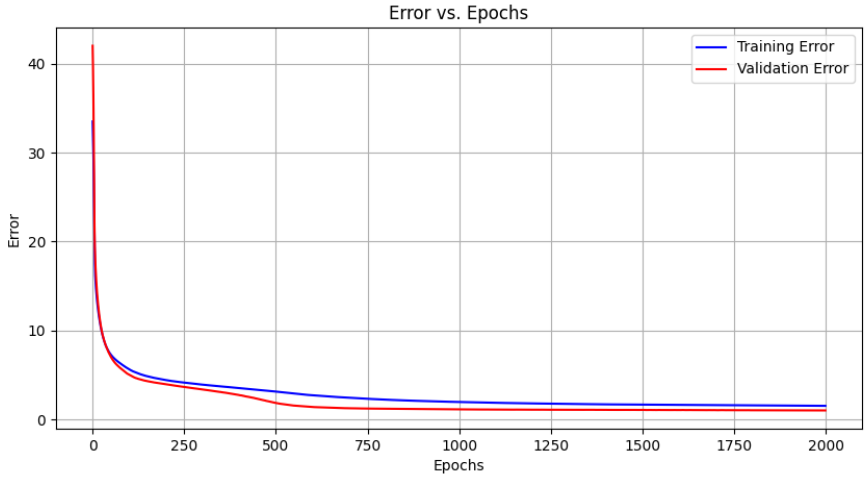
- Stochastiniame gradientiniame nusileidime kiekvienos epochos, tai yra iteracijos metu skaičiuojamas gradientas kiekvienai duomenų eilutei. Mokymo metu minimizuojama funkcija $E(W) = \sum_{i=0}^{m}(t_i - y_i)^2$, kur t_i – tikroji klasė, y_i – modelio numatyta klasė. Tai suma paklaidų kiekvienam įėjimų vektoriui. Minimizuojame $E(W)$ išvestinę pagal svorius w_k. Svoriai atnaujinami pagal formulę $w_k = w_k - \eta(y_i - t_i)y_i(1 - y_i)x_{ik}$, kur η – mokymosi greitis.
- Paketiniame gradientiniame nusileidime gradientas skaičiuojamas apdorojus visą duomenų rinkinį. Taigi kiekvienos epochos metu atnaujinami svoriai vieną kartą sudėjus visų duomenų gradientus $w_k = w_k + \eta(gradientSum_k/m)$, kur m – duomenų eilučių skaičius.

1.6. Tyrimai

1.6.1. Paklaidos reikšmių priklausomybė nuo epochų skaičiaus

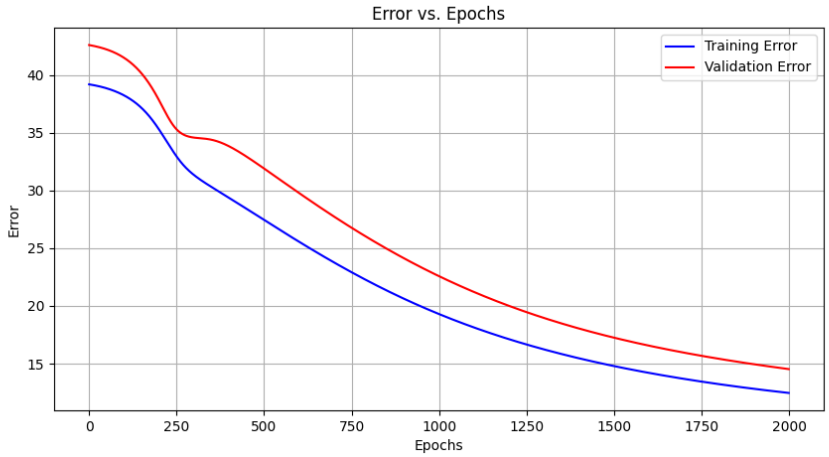
Paklaidos skaičiuojamos po kiekvienos epochos vis pridedant naują paklaidą, tada sumą padalinant iš bendro skaičiaus.

Stochastinio gradientinio nusileidimo atveju, kaip matoma *1 pav.*, kur mėlyna kreivė – apmokymo paklaidos, raudona – validavimo paklaidos, visų epochų metu gautos paklaidos mokymo metu ir validavimo metu sutampa gana glaudžiai, kas reiškia, kad gali veiksmingai klasifikuoti uždavinius. Paklaidos krenta viso mokymo metu, todėl modelis nėra permokytas.



1 pav. Stochastinio nusileidimo paklaidos

Paketinio gradientinio nusileidimo atveju, kaip matoma 2 pav. *Paketinio nusileidimo paklaidos* paklaidos tiek mokymo tiek validavimo metu taip pat koreliuoja. Paklaidos krenta viso apmokymo metu. Priešingai nei stochastinio modelio metu, paklaidos krenta tolydžiau, reikia daugiau epochų pasiekti žemesnę paklaidą.

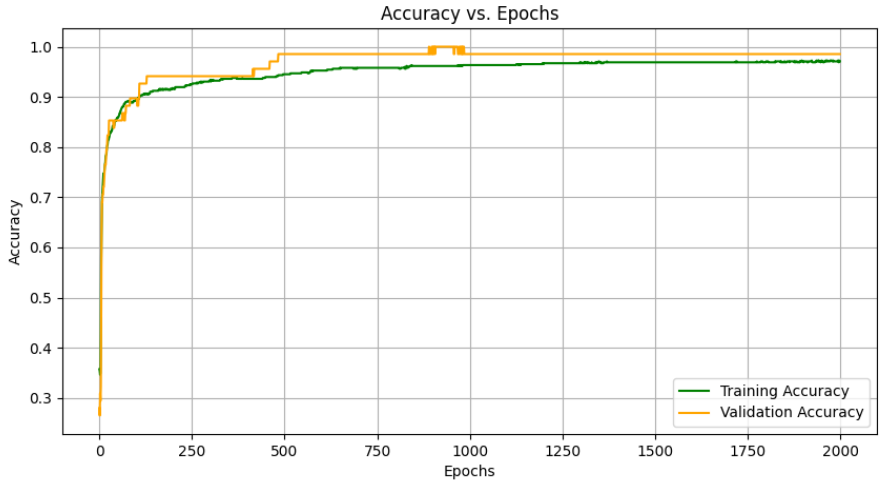


2 pav. Paketinio nusileidimo paklaidos

1.6.2. Klasifikavimo tikslumo priklausomybė nuo epochų skaičiaus

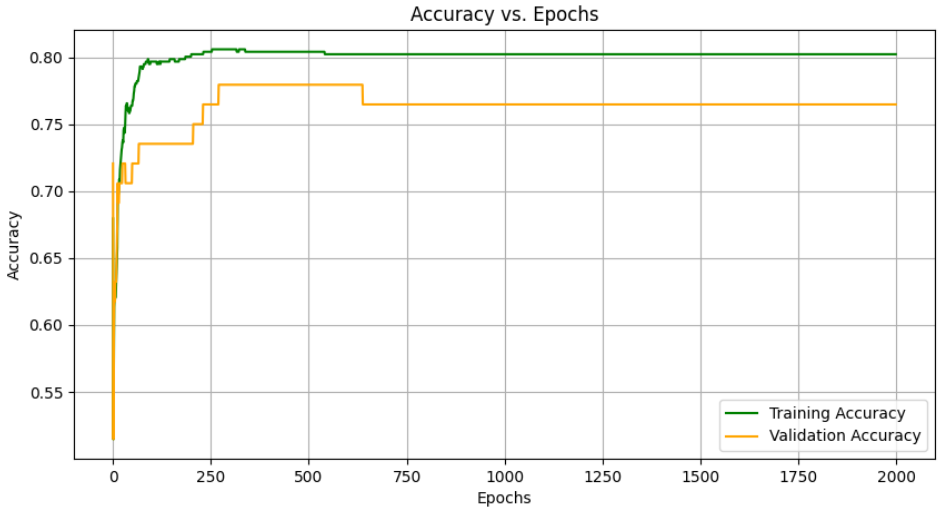
Klasifikavimo tikslumas skaičiuojamas naudojant konkrečioje epochoje gautus svorius, lyginant modelio nustatomas klases su tikrosiomis klasėmis.

Stochastinio gradientinio nusileidimo atveju, kaip matoma *3 pav. Stochastinio nusileidimo tikslumas* kur žalia kreivė atspindi tikslumą mokymosi metu, o geltona – validavimo metu per visas epochas, klasifikavimo tikslumas išauga stipriai jau per pirmą šimtą epochų, tačiau stabiliai kyla ir toliau. Kreivė mokymo metu turi daugiau pakitimų, o validavimo daugiau pastebimų lūžių, nes validavimo duomenų yra mažiau.



3 pav. Stochastinio nusileidimo tikslumas

Paketinio gradientinio nusileidimo atveju, kaip matoma *4* pav. Paketinio nusileidimo tikslumas rezultatas panašus, tačiau yra didesnis skirtumas tarp tikslumo mokymo ir validavimo metu, be to tikslumas mokymo metu nebekito jau nuo maždaug 500 epochų. Taigi galutinis tikslumas buvo pasiektas kur kas greičiau nei stochastinio modelio metu, bet jis yra mažesnis.



4 pav. Paketinio nusileidimo tikslumas

1.6.3. Mokymosi greičių testavimas

Kaip matoma *1 Lentelė. Modelių greičio priklausomybė nuo mokymosi greičio reikšmės*. mokymo greičio reikšmės kitimas neturėjo didelės įtakos modelio apmokymo greičiui. Paketinio

gradientinio nusileidimo atveju modelis kiekvienu atveju buvo greitesnis už stochastinį gradientinį nusileidimą.

1 Lentelė. Modelių greičio priklausomybė nuo mokymosi greičio reikšmės.

Mokymosi greičio reikšmė	***Stochastinis (sekundėmis)***	***Paketinis (sekundėmis)***
0,1	2,79	2,33
0,01	2,74	2,35
0,001	2,76	2,35
0,0001	2,74	2,30
0,00001	2,68	2,28

1.6.4. Skirtingų metodų rezultatai

- Rezultatai validavimo duomenims naudojant stochastinį gradientinį nusileidimą matomi *2 Lentelė. Stochastinio nusileidimo paklaidos ir tikslumas* Panaudojus modelį penkis kartus, mokymo paklaida buvo intervale (2,7;7,1) o pasiektas tikslumas tarp 88,24 % ir 97,06 %.

2 Lentelė. Stochastinio nusileidimo paklaidos ir tikslumas

Pasiekta paklaida mokymo gale	***Pasiektas tikslumas***
7,1	88,24%
4,5	90,01%
4,2	91,18%
2,5	92,65%
2,7	97,06%

- Rezultatai apmokant naudojant paketinį gradientinį nusileidimą buvo mažiau tenkinami. Kaip matoma *3 Lentelė. Paketinio nusileidimo paklaidos ir tikslumas* Pasiekta paklaida svyravo nuo 8,1 iki 11,9, tuo tarpu pasiektas tikslumas nuo 58,82 % iki 92,65 %.

3 Lentelė. Paketinio nusileidimo paklaidos ir tikslumas

Pasiekta paklaida mokymo gale	***Pasiektas tikslumas***
11,9	72,06%
12,7	83,82%
17,4	58,82%
8,1	92,65%

9,1	86,76%

1.6.5. Skirtingų metodų mokymo laikas

Kaip matoma *4 Lentelė. Epochų skaičiaus įtaka apmokymo laikams* paketinis gradientinis nusileidimas visada buvo greitesnis už stochastinį, nepriklausomai nuo epochų skaičiaus.

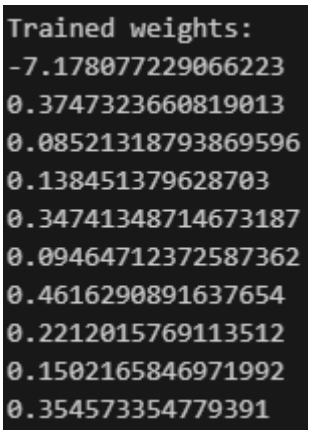
4 Lentelė. Epochų skaičiaus įtaka apmokymo laikams

Epochų skaičius	***Stochastinio metodo laikas (sekundėmis)***	***Paketinio metodo laikas (sekundėmis)***
10000	13,51	11,47
5000	6,79	5,62
2000	2,76	2,39
1000	1,37	1,16
500	0,70	0,59

1.7. Optimaliausias atvejis

Stochastinis gradientinis nusileidimas šiuo atveju daug stabilesnis, beveik visada siekia tikslumą virš 90 procentų, todėl tinkinamas būtent šis modelis. Gaunami geriausi rezultatai kai mokymosi greitis 0,001 arba 0,0001. Pasirinktas greitis – 0,001. Epochų skaičius stipriai daro įtaką paklaidai. Kuo didesnis epochų skaičius, tuo labiau krenta paklaida, tačiau po maždaug 7000 epochų, ji krenta labai lėtai. Dėl to pasirinktas epochų skaičius yra 7000.

- *5* pav. Gauti svoriai ir poslinkis matomi gauti rezultatai. Pirmasis svoris – maždaug -7,19 yra poslinkis. Kiti svoriai buvo gauti tarp maždaug 0,09 ir 0,46 reikšmių.



5 pav. Gauti svoriai ir poslinkis

- Gauta paklaida paskutinėje epochoje mokymo duomenims: 1.14.
- Gauta paklaida paskutinėje epochoje validavimo duomenims: 2,06.
- Klasifikavimo tikslumas validavimo duomenims: 97,06%.
- Klasifikavimo tikslumas testavimo duomenimis: 97,10%.
- Testavimo įrašų palyginimas su gautais rezultatais matomas pirmame priede.

1.8. Išvados

Sprendžiant klasifikavimo uždavinį buvo apmokytas neuronas stochastinio bei paketinio gradientinio nusileidimo metodais. Stochastinio gradientinio nusileidimo metodo atveju modelis buvo mokomas ilgiau, tačiau pasiekia geresnius rezultatus ir žemesnes paklaidas. Paketinio gradientinio nusileidimo metodo atveju modelis pasiekia galutinį tikslumą kur kas greičiau, bet rezultatas būna žemesnis nei stochastinio metodo atveju, be to šio metodo modelis netoks stabilus, dažnai pasiekia prastus rezultatus. Atlikus tyrimus buvo rastas optimaliausias atvejis, tai yra naudojamas stochastinis modelis, kai mokymosi greitis lygus 0,001 o epochų skaičius 7000. Tuomet modelis pasiekia daugiau nei 97 procentų tikslumą.

1 priedas. Testavimo rezultatai.

Tikėtasi	Gauta
1.0	1
1.0	1
0.0	0
1.0	1
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
1.0	1
0.0	0
0.0	0
1.0	1
0.0	0
0.0	0
1.0	1
0.0	0
1.0	1
0.0	1
1.0	1
0.0	0
0.0	0
0.0	0
1.0	1
0.0	0
0.0	0
1.0	1
0.0	0
0.0	0
0.0	0

0.0	0
0.0	0
0.0	0
0.0	0
0.0	1
0.0	0
1.0	1
1.0	1
1.0	1
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
0.0	0
1.0	1
1.0	1
1.0	1
0.0	0
1.0	1
0.0	0
0.0	0
1.0	1
1.0	1
0.0	0
0.0	0
0.0	0
1.0	1
0.0	0
1.0	1
0.0	0
1.0	1
1.0	1
0.0	0

2 priedas. Programinis kodas.

- Pagrindinis paketinio modelio failas

```
import random
```

```
import math
import time
from data_processing import load_and_preprocess_data, split_data
from error_analysis import compute_error, save_error_to_file
from predict_and_evaluate import evaluate_accuracy
from graphs import plot_error_vs_epochs,plot_accuracy_vs_epochs

features, labels = load_and_preprocess_data('breast-cancer-wisconsin.data')
features_train, class_train, features_val, class_val, features_test, class_test =
split_data(features, labels)

def sigmoid(a):
    return 1 / (1 + math.exp(-a))

def train_model(features_train, class_train, features_val,
class_val,learning_rate=0.001, epochs=2000, Emin=1e-4):

    w = [random.uniform(-1, 1) for _ in range(len(features_train[0]) + 1)]
    error_history_train = []
    error_history_val = []
    accuracy_history_train = []
    accuracy_history_val = []
    totalError = float('inf')
    epoch = 0

    start_time = time.time()

    while totalError > Emin and epoch < epochs:
        train_data = list(zip(features_train, class_train))
        random.shuffle(train_data)
        xi, ti = zip(*train_data)

        totalError = 0.0
        gradientSum = [0 for _ in range(len(features_train[0]) + 1)]

        for features, label in zip(xi, ti):
            ai = w[0]
            for k in range(len(features)):
                ai += w[k + 1] * features[k]

            yi = sigmoid(ai)
            error = yi - label
            totalError += error ** 2

            for k in range(len(features)):
                gradientSum[k + 1] += error * features[k]

        w[0] -= learning_rate * (gradientSum[0] / len(features_train))
        for k in range(len(features_train[0])):
            w[k + 1] -= learning_rate * (gradientSum[k + 1] / len(features_train))

        epoch += 1
        error_history_train.append(totalError / len(features))
        error_history_val.append(compute_error(features_val, class_val, w,
sigmoid))
        train_accuracy = evaluate_accuracy(features_train, class_train, w)
        val_accuracy = evaluate_accuracy(features_val, class_val, w)
        accuracy_history_train.append(train_accuracy)
        accuracy_history_val.append(val_accuracy)

        end_time = time.time()
        training_time = end_time - start_time

    save_error_to_file("train_error_history.txt", error_history_train)
    save_error_to_file("val_error_history.txt", error_history_val)
    plot_error_vs_epochs(error_history_train, error_history_val)
```

```
        plot_accuracy_vs_epochs(accuracy_history_train, accuracy_history_val)

        return w, training_time


    trained_weights, training_time  = train_model(features_train, class_train,
    features_val, class_val)
    print(f"Training completed in {training_time:.2f} seconds.")
    print("Trained weights:")
    for weight in trained_weights:
        print(f"{weight}")

    val_accuracy = evaluate_accuracy(features_val, class_val, trained_weights)
    test_accuracy = evaluate_accuracy(features_test, class_test, trained_weights)

    print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
    print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

- Pagrindinis stochastinio modelio failas

```
import random
import math
import time
from data_processing import load_and_preprocess_data, split_data
from error_analysis import compute_error, save_error_to_file
from predict_and_evaluate import evaluate_accuracy,evaluate_and_print_accuracy
from graphs import plot_error_vs_epochs,plot_accuracy_vs_epochs

features, labels = load_and_preprocess_data('breast-cancer-wisconsin.data')
features_train, class_train, features_val, class_val, features_test, class_test =
split_data(features, labels)

def sigmoid(a):
    return 1 / (1 + math.exp(-a))

def train_model(features_train, class_train, features_val, class_val,
learning_rate=0.001, epochs=7000, Emin=1e-4):

    w = [random.uniform(-1, 1) for _ in range(len(features_train[0]) + 1)]
    error_history_train = []
    error_history_val = []
    accuracy_history_train = []
    accuracy_history_val = []
    totalError = float('inf')
    epoch = 0

    start_time = time.time()

    while totalError > Emin and epoch < epochs:
        train_data = list(zip(features_train, class_train))
        random.shuffle(train_data)
        xi, ti = zip(*train_data)

        totalError = 0.0

        for features, label in zip(xi, ti):
            ai = w[0]
            for k in range(len(features)):
                ai += w[k + 1] * features[k]

            yi = sigmoid(ai)
            error = yi - label

            w[0] -= learning_rate * error * yi * (1 - yi)
            for k in range(len(features)):
                w[k + 1] -= learning_rate * error * yi * (1 - yi) * features[k]
```

```
            totalError += error ** 2

        epoch += 1
        error_history_train.append(totalError/ len(features))
        error_history_val.append(compute_error(features_val, class_val, w, sigmoid))
        train_accuracy = evaluate_accuracy(features_train, class_train, w)
        val_accuracy = evaluate_accuracy(features_val, class_val, w)
        accuracy_history_train.append(train_accuracy)
        accuracy_history_val.append(val_accuracy)

        end_time = time.time()
        training_time = end_time - start_time

    save_error_to_file("train_error_history.txt", error_history_train)
    save_error_to_file("val_error_history.txt", error_history_val)
    plot_error_vs_epochs(error_history_train, error_history_val)
    plot_accuracy_vs_epochs(accuracy_history_train, accuracy_history_val)
    return w,training_time

trained_weights, training_time  = train_model(features_train, class_train,
features_val, class_val)
print(f"Training completed in {training_time:.2f} seconds.")
print("Trained weights:")
for weight in trained_weights:
    print(f"{weight}")

val_accuracy = evaluate_and_print_accuracy(features_val, class_val, trained_weights)
test_accuracy = evaluate_accuracy(features_test, class_test, trained_weights)

print(f"Validation Accuracy: {val_accuracy * 100:.2f}%")
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

- Failas duomenų nuskaitymui, tvarkymui ir padalijimui

```
import random

def load_and_preprocess_data(filename):
    structured_data = []

    with open(filename, 'r') as file:
        for line in file:
            line = line.strip()

            if '?' in line:
                continue

            data_points = line.split(',')[1:]
            data_points = list(map(float, data_points))

            if data_points[9] == 2.0:
                data_points[9] = 0.0
            elif data_points[9] == 4.0:
                data_points[9] = 1.0

            structured_data.append(data_points)

    random.shuffle(structured_data)

    features = [row[:-1] for row in structured_data]
    labels = [row[-1] for row in structured_data]

    return features, labels

def split_data(features, labels, train_ratio=0.8, val_ratio=0.1):
    train_size = int(train_ratio * len(features))
    val_size = int(val_ratio * len(features))
```

```
    features_train, class_train = features[:train_size], labels[:train_size]
    features_val, class_val = features[train_size:train_size + val_size],
labels[train_size:train_size + val_size]
    features_test, class_test = features[train_size + val_size:], labels[train_size +
val_size:]

    print(f"Training set: {len(features_train)} samples")
    print(f"Validation set: {len(features_val)} samples")
    print(f"Test set: {len(features_test)} samples")

    return features_train, class_train, features_val, class_val, features_test,
class_test
```

- Failas skaičiuoti paklaidoms

```
def compute_error(features, labels, weights, sigmoid):
    total_error = 0.0

    for feature, label in zip(features, labels):
        ai = weights[0]
        for i in range(len(feature)):
            ai += weights[i + 1] * feature[i]

        yi = sigmoid(ai)
        error = yi - label
        total_error += error ** 2

    return total_error

def save_error_to_file(filename, error_history):
    with open(filename, "w") as file:
        for error in error_history:
            file.write(f"{error}\n")
```

- Failas skaičiuoti tikslumui

```
import math

def predict(features, weights):
    ai = weights[0]
    for i in range(len(features)):
        ai += weights[i + 1] * features[i]
    yi = round(1 / (1 + math.exp(-ai)))
    return yi

def evaluate_accuracy(features, labels, weights):
    correct_predictions = sum(1 for feature, label in zip(features, labels) if
predict(feature, weights) == label)
    return correct_predictions / len(features)

def evaluate_and_print_accuracy(features, labels, weights, filename="results.txt"):
    correct_predictions = 0
    with open(filename, "w") as file:
        file.write("Expected\tPredicted\n")
        for feature, label in zip(features, labels):
            predicted = predict(feature, weights)
            file.write(f"{label}\t{predicted}\n")
            if predicted == label:
                correct_predictions += 1

    accuracy = correct_predictions / len(features)
    print(f"Accuracy: {accuracy * 100:.2f}% (Results saved in {filename})")
    return accuracy
```

- Failas grafikams kurti

```
import matplotlib.pyplot as plt
```

```
def plot_error_vs_epochs(error_history_train, error_history_val):
    plt.figure(figsize=(10, 5))
    plt.plot(range(len(error_history_train)), error_history_train, label="Training
Error", color='blue')
    plt.plot(range(len(error_history_val)), error_history_val, label="Validation
Error", color='red')
    plt.xlabel("Epochs")
    plt.ylabel("Error")
    plt.title("Error vs. Epochs")
    plt.legend()
    plt.grid()
    plt.savefig("error_vs_epochs.png")
    plt.show()

def plot_accuracy_vs_epochs(accuracy_history_train, accuracy_history_val):
    plt.figure(figsize=(10, 5))
    plt.plot(range(len(accuracy_history_train)), accuracy_history_train,
label="Training Accuracy", color='green')
    plt.plot(range(len(accuracy_history_val)), accuracy_history_val, label="Validation
Accuracy", color='orange')
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.title("Accuracy vs. Epochs")
    plt.legend()
    plt.grid()
    plt.savefig("accuracy_vs_epochs.png")
    plt.show()
```