

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**204 DATA STRUCTURES (3+1)**

**2023–2024 FALL SEMESTER**

**PROJECT-1 REPORT**

**(Arrays, Matrices, Methods, Classes, Random Numbers)**

**DELIVERY DATE**

19/11/2023

**PREPARED BY**

05220000380, Yaren KILIÇ

05220000316, Furkan İYEM

05220000295, Altuğ Emre TOSUN

İçindekiler

[1) KARAYOLLARI UZAKLIK HESAPLAMALARI 3](#_Toc149748651)

[1.a Uzaklık Matrisi Oluşturma 3](#_Toc149748652)

[1.a.1 Kodlar 3](#_Toc149748653)

[1.a.2 Ekran görüntüleri 3](#_Toc149748654)

[1.b Verilen İlden Belli Bir Uzaklığa Kadar Olan İllerin ve Uzaklıklarının Listelenmesi 4](#_Toc149748655)

[1.b.1 Kodlar 4](#_Toc149748656)

[1.b.2 Ekran görüntüleri 4](#_Toc149748657)

[1.c Türkiye’deki Birbirine En Yakın İki İlin ve En Uzak İki İlin Bulunması 5](#_Toc149748658)

[1.c.1 Kodlar 5](#_Toc149748659)

[1.c.2 Ekran görüntüleri 5](#_Toc149748660)

[1.d Verilen İlden Verilen Mesafe Kullanılarak En fazla Kaç İl Dolaşılabildiğinin Bulunması 6](#_Toc149748661)

[1.d.1 Kodlar 6](#_Toc149748662)

[1.d.2 Algoritma ve Açıklama 6](#_Toc149748663)

[1.d.3 Ekran görüntüleri 7](#_Toc149748664)

[1.e Matris Şeklinde İllerin Adlarıyla Birlikte Ekrana Listelenmesi 9](#_Toc149748665)

[1.e.1 Kodlar 9](#_Toc149748666)

[1.e.2 Ekran görüntüleri 9](#_Toc149748667)

[2) DEVELOPING A PERCEPTRON MODEL and IMPLEMENTATION of a REGRESSION EXAMPLE 10](#_Toc149748668)

[2.a Neuron (Sinir Hücresi) Sınıfı 10](#_Toc149748669)

[2.a.1 Kaynak Kod 10](#_Toc149748670)

[2.a.2 Açıklama 10](#_Toc149748671)

[2.b Eğitim 14](#_Toc149748672)

[2.b.1 Kaynak Kod 14](#_Toc149748673)

[2.b.2 Açıklama 14](#_Toc149748674)

[2.b.3 Ekran Görüntüleri 15](#_Toc149748675)

[2.c Modelin Görmediği Veriden Sınav Sonucu Tahminleme 16](#_Toc149748676)

[2.c.1 Kaynak Kod 16](#_Toc149748677)

[2.c.2 Sonuçlar/Ekran görüntüleri 16](#_Toc149748678)

[2.d Deneyler 16](#_Toc149748679)

[2.d.1 Kaynak Kod 16](#_Toc149748680)

[2.d.2 Sonuçlar 16](#_Toc149748681)

[Öz değerlendirme Tablosu 18](#_Toc149748682)

# 1) KARAYOLLARI UZAKLIK HESAPLAMALARI

Microsoft Visual Studio, Version 17.8.0, C#

## 1.a Uzaklık Matrisi Oluşturma

### 1.a.1 Kodlar

static int[][] CreateMatrix()

{

string excelFilePath = @"C:\Users\lenovo\Downloads\ilmesafe.xlsx";

FileInfo fileInfo = new FileInfo(excelFilePath);

ExcelPackage.LicenseContext = LicenseContext.NonCommercial;

using (ExcelPackage package = new ExcelPackage(fileInfo))

{

var worksheet = package.Workbook.Worksheets[0];

int startRow = 4; // row index for the start cell

int startColumn = 3; // column index for the start cell

int endRow = 83; // row index for the end cell

int endColumn = 82; // column index for the end cell

// gets the number of rows and columns in the diagonal

int diagonalRowCount = endRow - startRow + 1;

int diagonalColumnCount = endColumn - startColumn + 1;

int[][] diagonalData = new int[diagonalRowCount][];

for (int i = 0; i < diagonalRowCount; i++)

{

diagonalData[i] = new int[i + 1];

for (int j = 0; j <= i; j++)

{

var cellValue = worksheet.Cells[startRow + i, startColumn + j].Value;

if (int.TryParse(cellValue?.ToString(), out int distance))

{

diagonalData[i][j] = distance;

}

else

{

// if the cell value is not an integer, it sets it to 0

diagonalData[i][j] = 0;

}

}

}

return diagonalData;

}

### }}1.a.2 Ekran görüntüleri

## 1.b Verilen İlden Belli Bir Uzaklığa Kadar Olan İllerin ve Uzaklıklarının Listelenmesi

### 1.b.1 Kodlar

public static (ArrayList, ArrayList) ListCitiesInGivenDistance(int[][] jaggedArray, string givenCity, int givenDistance, ArrayList cities)

{

ArrayList validCities = new ArrayList();

ArrayList validCitiesDistance = new ArrayList();

for (int i = 0; i < jaggedArray.Length; i++)

{

if (i + 2 == cities.IndexOf(givenCity)) //search the given city

{

for (int j = 0; j < jaggedArray[i].Length; j++) // iterates through horizontally

{

if (jaggedArray[i][j] <= givenDistance)

{

validCities.Add(cities[j + 1]);

validCitiesDistance.Add(jaggedArray[i][j]);

}

}

}

// iterates through diagonally

if (i + 2 > cities.IndexOf(givenCity) && jaggedArray[i][cities.IndexOf(givenCity) - 1] <= givenDistance)

{

validCities.Add(cities[i + 2]);

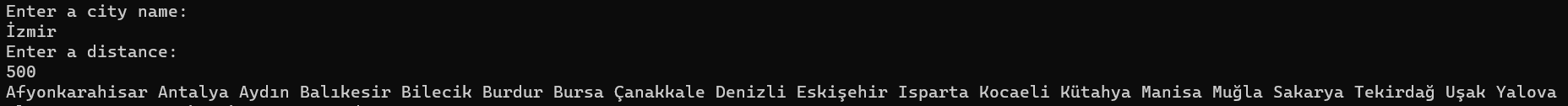
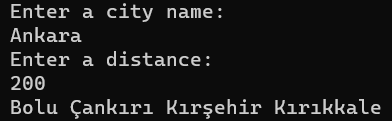
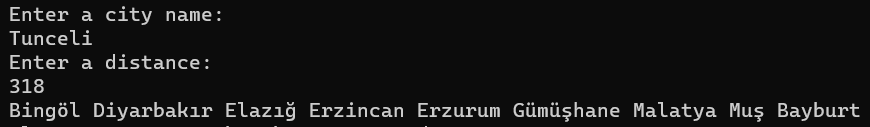
validCitiesDistance.Add(jaggedArray[i][cities.IndexOf(givenCity) - 1]);

}

}

return (validCities, validCitiesDistance);}

### 1.b.2 Ekran görüntüleri

1. 
2. 
3. 

## 1.c Türkiye’deki Birbirine En Yakın İki İlin ve En Uzak İki İlin Bulunması

### 1.c.1 Kodlar

public static void MaxMinDistance(int[][] jaggedArray, ArrayList cities)

{

//creating the city objects

object city1 = null;

object city2 = null;

object city3 = null;

object city4 = null;

int maxDistance = 0;

int minDistance = 3000;

for (int i = 0; i < jaggedArray.Length; i++) //iterates through the jaggedArray to check the distances between cities

{

for (int j = 0; j < i + 1; j++)

{

int distance = jaggedArray[i][j];

if (distance > maxDistance) // updates the max distance

{

maxDistance = distance;

city1 = (string)cities[i + 2];

city2 = (string)cities[j + 1];

}

else if (distance < minDistance) // updates the min distance

{

minDistance = distance;

city3 = (string)cities[i + 2];

city4 = (string)cities[j + 1];

}

}

}

//prints out the results

Console.WriteLine($"Max distance: {city1} - {city2} => {maxDistance}km");

Console.WriteLine($"Min distance: {city3} - {city4} => {minDistance}km");}

### 1.c.2 Ekran görüntüleri

## 

## 1.d Verilen İlden Verilen Mesafe Kullanılarak En fazla Kaç İl Dolaşılabildiğinin Bulunması

### 1.d.1 Kodlar

public static void FindMaxCityCount(int[][] sqArray, ArrayList cities, string startCity, int maxDistance)

{

// creates route lists

List<List<string>> allRoutes = new List<List<string>>();

List<string> currentRoute = new List<string>();

// finds all possible routes

FindAllRoutes(sqArray, cities, startCity, maxDistance, currentRoute, allRoutes);

List<string> mostCitiesRoute = null;

int maxCityCount = 0;

// finds the route with the maximum number of cities visited

foreach (var route in allRoutes)

{

if (route.Count > maxCityCount)

{

maxCityCount = route.Count;

mostCitiesRoute = route;

}

}

// displays the information

if (mostCitiesRoute != null)

{

Console.WriteLine($"The route with the most cities visited: {string.Join(" -> ", mostCitiesRoute)}");

Console.WriteLine($"Total city count: {mostCitiesRoute.Count}");

Console.WriteLine("Distances:");

int totalDistance = 0;

for (int i = 0; i < mostCitiesRoute.Count - 1; i++)

{

int cityIndex1 = cities.IndexOf(mostCitiesRoute[i]) - 1;

int cityIndex2 = cities.IndexOf(mostCitiesRoute[i + 1]) - 1;

int distance = sqArray[cityIndex1][cityIndex2];

totalDistance += distance;

Console.WriteLine($"{mostCitiesRoute[i]} - {mostCitiesRoute[i + 1]}: {distance}km");

}

Console.WriteLine($"Total Distance: {totalDistance}km");

}

else

{

Console.WriteLine($"The city visited within the specified distance was not found.");

}

}

private static void FindAllRoutes(int[][] sqArray, ArrayList cities, string currentCity, int remainingDistance,

List<string> currentRoute, List<List<string>> allRoutes)

{

if (remainingDistance < 0)

{

return;

}

currentRoute.Add(currentCity);

int currentIndex = cities.IndexOf(currentCity) - 1;

// searches thruogh the neighbour cities to find possible routes within distance

for (int i = 0; i < sqArray[currentIndex].Length; i++)

{

if (sqArray[currentIndex][i] > 0)

{

string nextCity = (string)cities[i + 1];

int nextDistance = sqArray[currentIndex][i];

if (!currentRoute.Contains(nextCity))

{

// uses recursion to complete the route

FindAllRoutes(sqArray, cities, nextCity, remainingDistance - nextDistance, currentRoute, allRoutes);

}

}

}

// adds the routes to list when there is no city left to visit

if (currentRoute.Count > 1)

{

allRoutes.Add(new List<string>(currentRoute));

}

currentRoute.RemoveAt(currentRoute.Count - 1);

}

### 1.d.2 Algoritma ve Açıklama

**Kullanılan veri yapıları:**

int[][] sqArray: Şehirlerin arasındaki mesafeleri tutan kare şeklindeki iki boyutlu dizi

ArrayList cities: Şehirlerin listesi

List<string> currentRoute: O anki rota

List<List<string>> allRoutes: Bütün rotalar

**Algoritmanın İşleyişi:**

1. FindMaxCityCount() metodu, tüm olası rotaları bulmak için FindAllRoutes() metodunu çağırır.

### FindAllRoutes() metodunda, belirli bir başlangıç şehriyle başlayarak, verilen maksimum mesafe içinde gidilebilecek tüm rotalar bulunur.

* Şu anki şehir currentCity olarak adlandırılır ve currentRoute listesine eklenir.
* remainingDistance kontrol edilir:
* Eğer remainingDistance negatifse, bu yolculuğun bu rotası geçersizdir ve işlem sonlandırılır. Bu, geriye dönüş anlamına gelir.
* Aksi halde, şehirler arasındaki mesafeler matrisi (sqArray) kullanılarak, mevcut şehirden gidilebilecek diğer şehirlere doğru bir döngü başlatılır.

**Her bir komşu şehir için:**

* Eğer matristeki mesafe pozitifse (0'dan büyükse):
  + nextCity olarak adlandırılan bu komşu şehir seçilir.
  + nextDistance, mevcut şehirden bu komşu şehire olan mesafeyi temsil eder.
  + Eğer currentRoute listesi, nextCity'yi içermiyorsa (döngüye tekrar girmemek için):
    - FindAllRoutes metodu remainingDistance - nextDistance verilerek recursive olarak çağrılır.
    - Bu adım, yeni bir rotanın oluşturulmasını sağlar

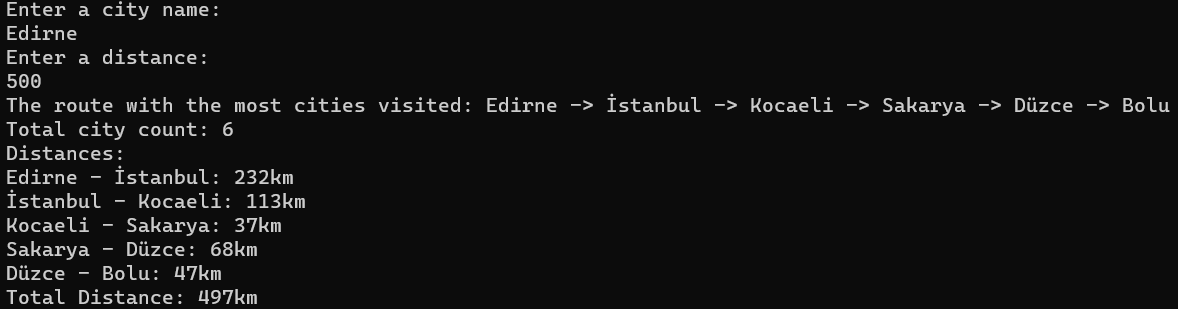
### Bulunan tüm rotalar, allRoutes listesine eklenir.

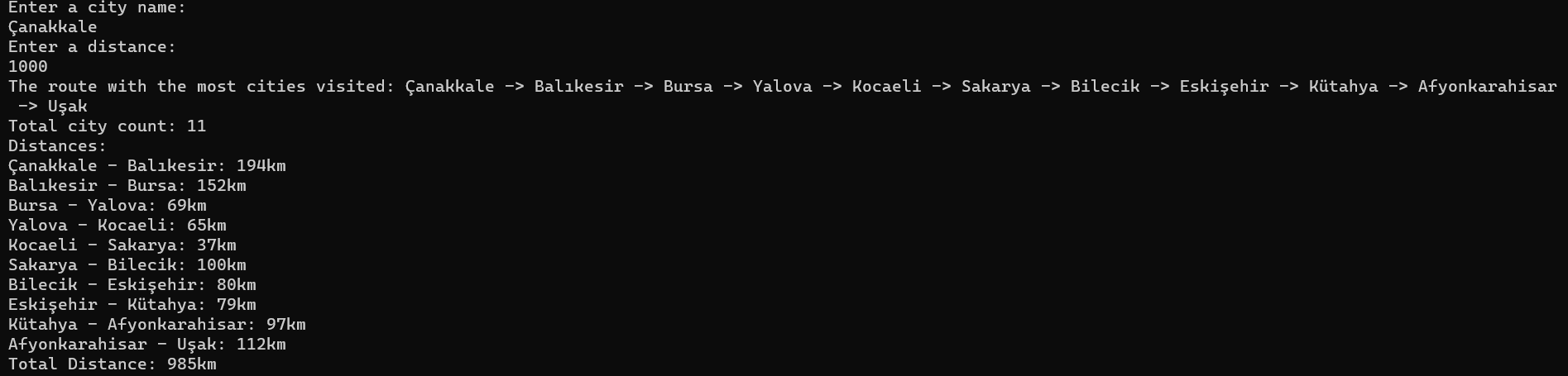
### 4) FindMaxCityCount() daha sonra tüm rotalar arasından en fazla şehri ziyaret eden rotayı seçer ve bu rotayı ve ziyaret edilen şehir sayısını ekrana yazdırır.

### 1.d.3 Ekran görüntüleri

1) A computer screen with white text

Description automatically generated

2) 

3) 

## 1.e Matris Şeklinde İllerin Adlarıyla Birlikte Ekrana Listelenmesi

### 1.e.1 Kodlar

public static void RandomCities(int[][] sqArray, ArrayList cities)

{

int[] BubbleSort(int[] arr)

{

int n = arr.Length;

for (int i = 0; i < n - 1; i++)

{

for (int j = 0; j < n - i - 1; j++)

{

if (arr[j] > arr[j + 1])

{

// swap arr[j] and arr[j+1]

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

return arr;

}

Random rnd = new Random();

int[] citiesSelected = new int[5] { 0, 0, 0, 0, 0 };

int counter = 0;

while (citiesSelected[4] == 0)

{

int plateNumber = rnd.Next(1, 82);

if (!citiesSelected.Contains(plateNumber)) { citiesSelected[counter] = plateNumber; counter++; }

}

citiesSelected = BubbleSort(citiesSelected); //sorts out the cities

//prints out the city names and distances in a matrix form

Console.WriteLine("+------------------------------------------------------------------------------+");

Console.Write("| ");

for (int i = 0; i < 5; i++)

{

Console.Write($"| {cities[citiesSelected[i]],-11}");

}

Console.WriteLine("|");

Console.WriteLine("+------------------------------------------------------------------------------+");

for (int i = 0; i < 5; i++)

{

Console.Write($"| {cities[citiesSelected[i]],-12}");

for (int j = 0; j < 5; j++)

{

if (citiesSelected[i] != citiesSelected[j] && citiesSelected[i] <= sqArray.Length && citiesSelected[j] <= sqArray.Length)

{

Console.Write($"| {sqArray[citiesSelected[i] - 1][citiesSelected[j] - 1],-10} ");

}

else

{

Console.Write("| 0 ");

}

}

Console.WriteLine("|");

Console.WriteLine("+------------------------------------------------------------------------------+");

}

}

### 1.e.2 Ekran görüntüleri

# 

# 2) DEVELOPING A PERCEPTRON MODEL and IMPLEMENTATION of a REGRESSION EXAMPLE

Microsoft Visual Studio, Version 17.8.0, C#

## 2.a Neuron (Sinir Hücresi) Sınıfı

### 2.a.1 Kaynak Kod

class Neuron

{

public double[,] inputs = new double[,]

{

{ 7.6, 11, 77 }, { 8, 10, 70 }, { 6.6, 8, 55 }, { 8.4, 10, 78 }, { 8.8, 12, 95 }, { 7.2, 10, 67 },

{ 8.1, 11, 80 }, { 9.5, 9, 87 }, { 7.3, 9, 60 }, { 8.9, 11, 88 }, { 7.5, 11, 72 }, { 7.6, 9, 58 },

{ 7.9, 10, 70 }, { 8, 10, 76 }, { 7.2, 9, 58 }, { 8.8, 10, 81 }, { 7.6, 11, 74 }, { 7.5, 10, 67 },

{ 9, 10, 82 }, { 7.7, 9, 62 }, { 8.1, 11, 82 }

};

public static double[] RandomizeWeight(int inputCount) //creates two random weights for two inputs

{

Random random = new Random();

double[] weights = new double[inputCount];

for (int i = 0; i < inputCount; i++)

{

weights[i] = Math.Round(random.NextDouble(), 2);

}

return weights;

}

public static double[,] UpdateInputs(double[,] inputs, int dataCount) //updates the inputs according to the given factors

{

double StudyTimeFactor = 0.1;

double AttendanceFactor = 0.066;

double ExamResultFactor = 0.01;

for (int j = 0; j < dataCount; j++)

{

inputs[j, 0] = inputs[j, 0] \* StudyTimeFactor;

inputs[j, 1] = inputs[j, 1] \* AttendanceFactor;

if (dataCount == 21) // OutOfSamplePrediction() and Train() are both using this method

{ inputs[j, 2] = inputs[j, 2] \* ExamResultFactor; }

}

return inputs;

}

public static double[] CalculateOutput(int dataCount, double[] weights, double[,] inputs)

{

double[] outputs = new double[dataCount];

for (int i = 0; i < dataCount; i++)

{

outputs[i] += weights[0] \* inputs[i, 0] + weights[1] \* inputs[i, 1]; // the sum formula for neuron model

}

return outputs;

}

public static (double[], double[]) Train(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch)

{

double[] outputs = CalculateOutput(dataCount, weights, inputs);

for (int i = 0; i < epoch; i++)

{

for (int j = 0; j < dataCount; j++) //wi = wi + λ\*(t-o)\*xi

{

weights[0] = weights[0] + LearningRate \* (inputs[j, 2] - outputs[j]) \* inputs[j, 0];

weights[1] = weights[1] + LearningRate \* (inputs[j, 2] - outputs[j]) \* inputs[j, 1];

}

outputs = CalculateOutput(dataCount, weights, inputs);

}

return (outputs.ToArray(), weights);

}

public static double CalculateMSE(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch )

{

(double[] outputs, double[] updatedWeights) = Train(dataCount, weights, inputs, LearningRate, epoch);

double sum = 0;

double MSE = 0;

//MSE formula = 1/n ( Σ(target − output)^2)

for (int i = 0; i < dataCount; i++) { sum += Math.Pow((inputs[i, 2] - outputs[i]), 2); }

MSE = sum / dataCount;

return MSE;

}

public static void OutOfSamplePrediction(double[] updatedWeights) //uses a different input sample

{

double[,] inputSample = new double[,] { { 7.8, 10 }, { 8.5, 12 }, { 7.0, 8 }, { 9.2, 11 }, { 7.8, 9 } };

inputSample = UpdateInputs(inputSample, 5);

double[] outputs = CalculateOutput(5, updatedWeights, inputSample);

//printing out the table

Console.WriteLine("Input Values | Predicted Values");

Console.WriteLine("-------------------------------");

for (int i = 0; i < 5; i++)

{

Console.WriteLine($"{inputSample[i, 0],-4:F2} | {inputSample[i, 1],-6:F2}| {outputs[i],-15:F2}");

}

}

public static void TrainAndCalculateMSEWithDifferentValues(int dataCount, double[] weights, double[,] inputs)

{

double[] learningRateValues = { 0.01, 0.025, 0.05 };

int[] epochValues = { 10, 50, 100 };

double[,] resultsTable = new double[learningRateValues.Length, epochValues.Length];

for (int i = 0; i < learningRateValues.Length; i++)

{

for (int j = 0; j < epochValues.Length; j++)

{

resultsTable[i, j] = CalculateMSE(dataCount, weights, inputs, learningRateValues[j], epochValues[i]);

}

}

// prints the headline

Console.Write("Learning Rate\\Epoch |");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write($"{epochValues[j],-23}|");

}

Console.WriteLine();

// prints the separator line

Console.Write("-----------------------|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write("-----------------------|");

}

Console.WriteLine();

// prints the results

for (int i = 0; i < learningRateValues.Length; i++)

{

Console.Write($"{learningRateValues[i],-23}|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write($"{resultsTable[i, j],-23:F14}|");

}

Console.WriteLine();

// prints the separator line

Console.Write("-----------------------|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write("-----------------------|");

}

Console.WriteLine();

}

}

}

### 2.a.2 Açıklama

The **Neuron** class represents a simple model of a artificial neuron. This neuron takes three input values representing factors such as study time, attendance, and exam results. The class includes methods for initializing random weights, updating inputs based on certain factors, calculating the neuron's output, training the neuron using an algorithm, calculating Mean Squared Error (MSE), and making out-of-sample predictions.

Methods of the **Neuron** class:

1. **RandomizeWeight(int inputCount):** Generates random initial weights for the neuron's inputs.
2. **UpdateInputs(double[,] inputs, int dataCount):** Updates the input values by multiplying them with given factors in the project description. (Normalisation)
3. **CalculateOutput(int dataCount, double[] weights, double[,] inputs):** Calculates the output of the neuron using the given weights and inputs.
4. **Train(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch):** Implements a training method that adjusts the weights using an algorithm to minimize the difference between predicted and target outputs.
5. **CalculateMSE(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch):** Calculates the Mean Squared Error (MSE) between the predicted and target outputs for a given set of weights, inputs, learning rate, and epochs.
6. **OutOfSamplePrediction(double[] updatedWeights):** Makes predictions on a different input sample using the trained weights and displays the results.
7. **TrainAndCalculateMSEWithDifferentValues(int dataCount, double[] weights, double[,] inputs):** Conducts training with different combinations of learning rates and epochs, displaying the MSE results in a matrix form.

## 2.b Eğitim

### 2.b.1 Kaynak Kod

public static (double[], double[]) Train(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch)

{

double[] outputs = CalculateOutput(dataCount, weights, inputs);

for (int i = 0; i < epoch; i++)

{

for (int j = 0; j < dataCount; j++) //wi = wi + λ\*(t-o)\*xi

{

weights[0] = weights[0] + LearningRate \* (inputs[j, 2] - outputs[j]) \* inputs[j, 0];

weights[1] = weights[1] + LearningRate \* (inputs[j, 2] - outputs[j]) \* inputs[j, 1];

}

outputs = CalculateOutput(dataCount, weights, inputs);

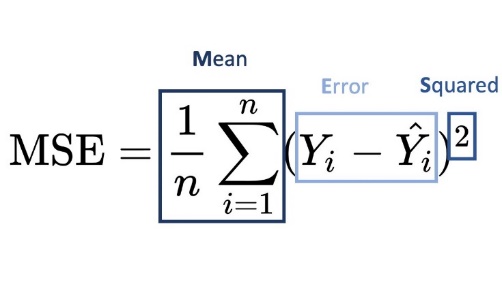
}

return (outputs.ToArray(), weights);}

### 2.b.2 Açıklama

Mean squared error (MSE), the average squared difference between the value observed in a statistical study and the values predicted from a model. When comparing observations with predicted values, it is necessary to square the differences as some data values will be greater than the prediction (and so their differences will be positive) and others will be less (and so their differences will be negative). Given that observations are as likely to be greater than the predicted values as they are to be less, the differences would add to zero. Squaring these differences eliminates this situation.

The formula for the mean squared error is :



, where Yi is the ith observed value, Ŷi is the corresponding predicted value for Yi, and n is the number of observations. The Σ indicates that a summation is performed over all values of i.

If the prediction passes through all data points, the mean squared error is zero. As the distance between the data points and the associated values from the model increase, the mean squared error increases. Thus, a model with a lower mean squared error more accurately predicts dependent values for independent variable values.

*Stewart, Ken. "mean squared error". Encyclopedia Britannica, 30 Mar. 2023, https://www.britannica.com/science/mean-squared-error. Accessed 18 November 2023.*

### The MSE Method

public static double CalculateMSE(int dataCount, double[] weights, double[,] inputs, double LearningRate, int epoch )

{

(double[] outputs, double[] updatedWeights) = Train(dataCount, weights, inputs, LearningRate, epoch);

double sum = 0;

double MSE = 0;

//MSE formula = 1/n ( Σ(target − output)^2)

for (int i = 0; i < dataCount; i++) { sum += Math.Pow((inputs[i, 2] - outputs[i]), 2); }

MSE = sum / dataCount;

return MSE;

}

2.b.3 Ekran Görüntüleri

## A screen shot of a black screen Description automatically generated

## 2.c Modelin Görmediği Veriden Sınav Sonucu Tahminleme

### 2.c.1 Kaynak Kod

public static void OutOfSamplePrediction(double[] updatedWeights) //uses a different input sample

{

double[,] inputSample = new double[,] { { 7.8, 10 }, { 8.5, 12 }, { 7.0, 8 }, { 9.2, 11 }, { 7.8, 9 } };

inputSample = UpdateInputs(inputSample, 5);

double[] outputs = CalculateOutput(5, updatedWeights, inputSample);

//printing out the table

Console.WriteLine("Input Values | Predicted Values");

Console.WriteLine("-------------------------------");

for (int i = 0; i < 5; i++)

{

Console.WriteLine($"{inputSample[i, 0],-4:F2} | {inputSample[i, 1],-6:F2}| {outputs[i],-15:F2}");

}

### }}2.c.2 Sonuçlar/Ekran görüntüleri

A black and white screen with numbers and a black background

Description automatically generated

## 2.d Deneyler

### 2.d.1 Kaynak Kod

public static void TrainAndCalculateMSEWithDifferentValues(int dataCount, double[] weights, double[,] inputs)

{

double[] learningRateValues = { 0.01, 0.025, 0.05 };

int[] epochValues = { 10, 50, 100 };

double[,] resultsTable = new double[learningRateValues.Length, epochValues.Length];

for (int i = 0; i < learningRateValues.Length; i++)

{

for (int j = 0; j < epochValues.Length; j++)

{

resultsTable[i, j] = CalculateMSE(dataCount, weights, inputs, learningRateValues[j], epochValues[i]);

}

}

// prints the headline

Console.Write("Learning Rate\\Epoch |");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write($"{epochValues[j],-23}|");

}

Console.WriteLine();

// prints the separator line

Console.Write("-----------------------|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write("-----------------------|");

}

Console.WriteLine();

// prints the results

for (int i = 0; i < learningRateValues.Length; i++)

{

Console.Write($"{learningRateValues[i],-23}|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write($"{resultsTable[i, j],-23:F14}|");

}

Console.WriteLine();

// prints the separator line

Console.Write("-----------------------|");

for (int j = 0; j < epochValues.Length; j++)

{

Console.Write("-----------------------|");

}

Console.WriteLine();

}

### }

### 2.d.2 Sonuçlar

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | Deney 1 | 10 Epok | 50 Epok | 100 Epok | | λ = 0.01 | 0.00311 | 0.00311 | 0.00311 | | λ = 0.025 | 0.00310 | 0.00309 | 0.00308 | | λ = 0.05 | 0.00307 | 0.00306 | 0.00305 | | |  |  |  |  | | --- | --- | --- | --- | | Deney 2 | 10 Epok | 50 Epok | 100 Epok | | λ = 0.01 | 0.00380 | 0.00378 | 0.00373 | | λ = 0.025 | 0.00369 | 0.00360 | 0.00346 | | λ = 0.05 | 0.00341 | 0.00332 | 0.00319 | |

The results from all three experiments consistently show that the best combination for training the neuron model is when using a regularization parameter (λ) set to 0.05 and conducting 100 epochs. These values result in the lowest mean squared error (MSE).

|  |  |  |  |
| --- | --- | --- | --- |
| Deney 3 | 10 Epok | 50 Epok | 100 Epok |
| λ = 0.01 | 0.00310 | 0.00310 | 0.00309 |
| λ = 0.025 | 0.00309 | 0.00308 | 0.00307 |
| λ = 0.05 | 0.00307 | 0.00306 | 0.00305 |

# Öz değerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 1 Maddeleri** | **Not** | **Tahmini Not** | **Açıklama** |
| 1.a | 10 | 10 | Yapıldı / Excel dosyasını okuyarak jagged array oluşturduk. |
| 1.b | 5 | 5 | Yapıldı / Program, önce şehrin plakasını bulana kadar aşağı iniyor. Daha sonra şehrin bulunduğu satırda yatay ilerliyor. Sınıra gelince sonraki sütundan aşağı inmeye devam ediyor. Bu şekilde verilen sayıdan küçük sayıları bulup listeye atıyor. |
| 1.c | 5 | 5 | Yapıldı / 1.b ile aynı mantıkla çalışıyor. Bu işlemleri yaparken en uzun ve en kısa mesafeyi bir değişken aracılığıyla güncel tutuyor. |
| 1.d | 15 | 15 | Yapıldı / Program, önce şehrin plakasını bulana kadar aşağı iniyor. Daha sonra şehrin bulunduğu satırda yatay ilerliyor. Sınıra gelince sonraki sütundan aşağı inmeye devam ediyor. Bu şekilde verilen sayıdan küçük sayıları bulup listeye atıyor. |
| 1.e | 5 | 5 | Yapıldı / Random class’ını kullanarak 5 tane farklı plaka üretiyor. Bu şehirleri ve aralarındaki mesafeleri matris halinde yazdırıyor. |
| 2.a | 10 | 10 | Yapıldı / Programın düzgün çalışması için **Neuron** sınıfına gereken metotları yazdık. Verileri depolamak için tek veya iki boyutlu dizileri kullandık. |
| 2.b | 15 | 15 | Yapıldı / Eğitim için train isimli bir metot kullanıyoruz. Output hesaplama metodundan elde ettiğimiz çıktılara istenilen epok sayısı kadar eğitim formülünü uyguluyoruz. Eğitim sonucunda aldığımız verileri de array halinde tutuyoruz. Tahminlenen sonuçlardaki hata oranını bulabilmek için array içindeki outputlara MSE metodunu uyguluyoruz. |
| 2.c | 5 | 5 | Yapıldı / Önce 5 farklı veri seti oluşturuyoruz. Daha sonra eğitilmiş modelin ürettiği ağırlıkları kullanarak output hesaplıyoruz. |
| 2.d | 10 | 10 | Yapıldı / Train metodunu farklı epok ve eğitim katsayıları ile çağırıyoruz. |
| Rapor | 10 | 10 | Yapıldı |
| Öz değerlendirme Tablosu | 10 | 10 | Yapıldı |
| **Toplam** | **100** | **100** |  |

**Açıklama kısmında yapıldı, yapılmadı bilgisi ve hangi maddelerin nasıl yapıldığı veya neden yapılamadığı kısaca yazılmalıdır.**

**Not: Raporu teslim edilmeyen projeler değerlendirmeye alınmayacaktır.**