

İçindekiler

1) KARAYOLLARI UZAKLIK HESAPLAMALARI.....	3
1.a Uzaklık Matrisi Oluşturma	3
1.a.1 Kodlar.....	3
1.a.2 Ekran görüntüleri.....	4
1.b Verilen İlden Belli Bir Uzaklığa Kadar Olan İllerin ve Uzaklıklarının Listelenmesi	4
1.b.1 Kodlar.....	4
1.b.2 Ekran görüntüleri.....	4
1.c Türkiye'deki Birbirine En Yakın İki İlin ve En Uzak İki İlin Bulunması	4
1.c.1 Kodlar	4
1.c.2 Ekran görüntüleri	5
1.d Verilen İlden Verilen Mesafe Kullanılarak En fazla Kaç İl Dolaşılabilirdiğinin Bulunması	5
1.d.1 Kodlar.....	5
1.d.2 Algoritma ve Açıklama	6
1.d.3 Ekran görüntüleri.....	6
1.e Matris Şeklinde İllerin Adlarıyla Birlikte Ekrana Listelenmesi	7
1.e.1 Kodlar.....	7
1.e.2 Ekran görüntüleri.....	8
2) DEVELOPING A PERCEPTRON MODEL and IMPLEMENTATION of a REGRESSION EXAMPLE	8
2.a Neuron (Sinir Hücresi) Sınıfı	8
2.a.1 Kaynak Kod.....	8
2.a.2 Açıklama.....	11
2.b Eğitim.....	13
2.b.1 Kaynak Kod.....	13
2.b.2 Açıklama.....	13
2.b.3 Ekran Görüntüleri	13
2.c Modelin Görmediği Veriden Sınav Sonucu Tahminleme	14
2.c.1 Kaynak Kod	14
2.c.2 Sonuçlar/Ekran görüntüleri	14
2.d Deneyler	15
2.d.1 Kaynak Kod.....	15
2.d.2 Sonuçlar	15
Öz değerlendirme Tablosu	16

1) KARAYOLLARI UZAKLIK HESAPLAMALARI

//The platform, version, and programming language used

1.a Uzaklık Matrisi Oluşturma

1.a.1 Kodlar

```
class Cities
{
    string[] cities;
    int[][] distances;
    ArrayList dictionary;
    Dictionary<string, Dictionary<int, List<string>>>> cityDistances;
    Dictionary<string, Dictionary<string, int>>> cityToCityDistances;

    public Cities(string[] cities_, int[][] distances_, ArrayList dictionary_, Dictionary<string, Dictionary<int,
List<string>>>> cityDistances_, Dictionary<string, Dictionary<string, int>>> cityToCityDistances_)
    {
        cities = cities_;
        distances = distances_;
        dictionary = dictionary_;
        cityDistances = cityDistances_;
        cityToCityDistances = cityToCityDistances_;
    }

    public void ProcessMatrix(string[][] matrix) {
        for (int i = 2; i < 83; i++)
        {
            cities[i - 2] = matrix[i][1];
            distances[i - 2] = new int[81];
            for (int j = 2; j < 83; j++)
            {
                int num = 0;
                if(!string.IsNullOrEmpty(matrix[i][j])) {
                    num = int.Parse(matrix[i][j]);
                }

                distances[i - 2][j - 2] = num;
                dictionary.Add(new object[] { matrix[i][1], matrix[1][j], num }); if (!
cityDistances.ContainsKey(matrix[i][1]))
                {
                    cityDistances[matrix[i][1]] = new Dictionary<int, List<string>>>();
                    cityToCityDistances[matrix[i][1]] = new Dictionary<string, int>>>();
                }
                if(!cityDistances[matrix[i][1]].ContainsKey(num))
                {
                    cityDistances[matrix[i][1]][num] = new List<string>();
                }
                if(!cityToCityDistances[matrix[i][1]].ContainsKey(matrix[1][j]))
                {
                    cityToCityDistances[matrix[i][1]][matrix[1][j]] = num;
                }
                cityDistances[matrix[i][1]][num].Add(matrix[1][j]);
            }
        }
    }
}
```

```
}  
}
```

1.a.2 Ekran görüntüleri

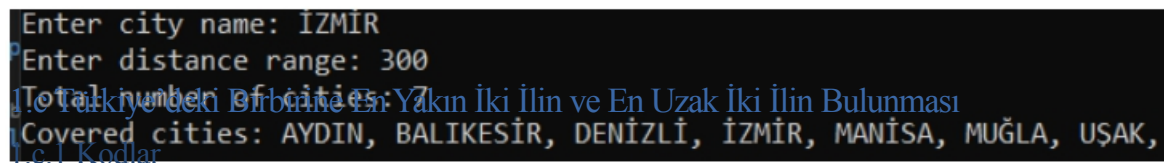
//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

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

1.b. Kodlar

```
public List<string> GetCoveredCities()  
{  
    List<string> coveredCities = new List<string>();  
    Console.Write("Enter city name: ");  
    string cityName = Console.ReadLine();  
    Console.Write("Enter distance range: ");  
    int distanceRange = int.Parse(Console.ReadLine());  
    if (distanceRange > 0 && cityDistances.ContainsKey(cityName)) {  
        foreach (int distance in cityDistances[cityName].Keys)  
        {  
            if (distance < distanceRange) {  
                foreach (string city in cityDistances[cityName][distance]) {  
                    coveredCities.Add(city);  
                }  
            }  
        }  
    }  
    return coveredCities;  
}
```

1.b.2 Ekran görüntüleri



```
Enter city name: İZMİR  
Enter distance range: 300  
Total number of cities: 7  
Covered cities: AYDIN, BALIKESİR, DENİZLİ, İZMİR, MANİSA, MUĞLA, UŞAK,
```

```
public void GetNearestAndFarthestCities()  
{  
    int nearest = Int32.MaxValue;  
    string[] nearestCities = new string[2];  
    int farthest = Int32.MinValue;  
    string[] farthestCities = new string[2];  
    foreach (object[] s in dictionary)  
    {  
        int distance = (int)s[2];  
        if (distance == 0) {  
            continue;  
        }  
        if (distance > farthest)  
        {  
            farthest = distance;
```

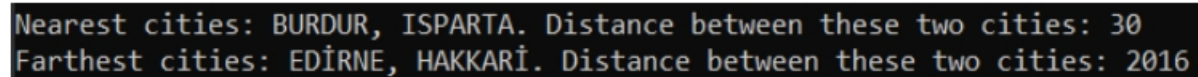
```

        farthestCities = new string[] { (string)s[0], (string)s[1] };
    }
    if (distance < nearest) {
        nearest = distance;
        nearestCities = new string[] { (string)s[0], (string)s[1] };
    }
}

Console.WriteLine("Nearest cities: " + nearestCities[0] + ", " + nearestCities[1] + ". Distance between these two cities: " + nearest);
Console.WriteLine("Farthest cities: " + farthestCities[0] + ", " + farthestCities[1] + ". Distance between these two cities: " + farthest);
}

```

1.c.2 Ekran görüntüleri



```

Nearest cities: BURDUR, ISPARTA. Distance between these two cities: 30
Farthest cities: EDİRNE, HAKKARİ. Distance between these two cities: 2016

```

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

1.d.1 Kodlar

```

public ArrayList MostVisitedCities(string startCity, int maxDistance)
{
    List<ArrayList> nextCities = new List<ArrayList>();

    nextCities.Add(new ArrayList { startCity, 0, new List<string>(), new HashSet<string> { startCity } });

    ArrayList answer = new ArrayList { int.MaxValue, new List<string>(), 0 };

    while (nextCities.Count > 0) {
        ArrayList currentCity = nextCities[0];
        nextCities.RemoveAt(0);

        string city = (string)currentCity[0];
        int distance = (int)currentCity[1];
        List<string> path = (List<string>)currentCity[2]; HashSet<string>
        visitedSet = (HashSet<string>)currentCity[3];

        if (path.Count > ((List<string>)answer[1]).Count) {
            answer = new ArrayList { distance, new List<string>(path), path.Count };
        }

        foreach (string targetCity in cities) {
            HashSet<string> visited = new HashSet<string>(visitedSet);

            if (visited.Contains(targetCity)) {
                continue;
            }

            int newDistance = distance + cityToCityDistances[city][targetCity];

            if (newDistance > maxDistance) {
                continue;
            }
        }
    }
}

```

```

        visited.Add(targetCity);
        List<string> newPath = new List<string>(path) { targetCity };

        nextCities.Add(new ArrayList { targetCity, newDistance, newPath, visited });
    }
}

return new ArrayList { answer[0], new List<string>(((List<string>)answer[1])), answer[2] };

```

1.d.2 Algoritma ve Açıklama

Bu C# kodu, şehirler ve aralarındaki mesafelerle ilgili coğrafi verilerle uğraşan bir programı tanımlar. Cities sınıfı, şehir isimlerini, mesafeleri ve bu verilerle ilgili bilgileri depolamak için çeşitli veri yapıları kullanır.

Cities sınıfının alanları arasında şehir isimlerini ve mesafeleri depolayan diziler, mesafeleri daha yapılandırılmış bir şekilde depolamak için ArrayList, şehirler arasındaki mesafeleri farklı mesafelere göre gruplandırmak için Dictionary türündeki cityDistances, ve iki şehir arasındaki doğrudan mesafeleri depolamak için citytoCityDistances yer alır.

Sınıf, şehir matrisi üzerinde işlem yapabilen çeşitli metotlar içerir. Bu metotlar arasında, belirli bir şehrin belirli bir mesafe aralığındaki diğer şehirleri bulan GetCoveredCities, en yakın ve en uzak şehirleri bulan, GetNearestAndFarthestCities belirli bir şehirden başlayarak en çok ziyaret edilen şehirleri ve toplam mesafeyi bulan MostVisitedCities, ve rastgele şehirler arasındaki mesafeleri gösteren RandomCityDistance bulunmaktadır.

Program sınıfı, bu Cities sınıfını kullanarak bir dizi işlemi gerçekleştirir. Önce bir dosyadan (ilmesafe.txt) şehir ve mesafe bilgilerini okur, ardından bu bilgileri işleyerek çeşitli metotları kullanır. Ayrıca, başka bir dosyadan (iller.txt) şehirler arasındaki mesafeleri temsil eden bir jagged array oluşturur ve yazdırır.

Bu programın genel akışı, coğrafi veriler üzerinde çeşitli işlemleri gerçekleştirmek ve sonuçları ekrana yazdırmak şeklindedir.

1.d.3 Ekran görüntüleri

```

Enter city name: İZMİR
Enter distance range: 300
Total number of cities: 7
Covered cities: AYDIN, BALIKESİR, DENİZLİ, İZMİR, MANİSA, MUĞLA, UŞAK,
Nearest cities: BURDUR, ISPARTA. Distance between these two cities: 30
Farthest cities: EDİRNE, HAKKARİ. Distance between these two cities: 2016

Enter a city name: BURSA
Enter the maximum distance you can travel: 500
Most visited cities: BİLECİK, YALOVA, KOCAELİ (İZMİR), SAKARYA (ADAPAZARI), BOLU, DÜZCE

```

1.e Matris Şeklinde İllerin Adlarıyla Birlikte Ekrana Listelenmesi

1.e.1 Kodlar

```
string desktop = Environment.GetFolderPath(Environment.SpecialFolder.DesktopDirectory);
StreamReader dosya = new StreamReader(desktop + "\\iller.txt");//iller.txt dosyası masaüstünde kalsın
string[] iller = {"",
"Adana", "Adıyaman", "Afyonkarahisar", "Ağrı", "Amasya",
"Ankara", "Antalya", "Artvin", "Aydın", "Balıkesir", "Bilecik",
"Bingöl", "Bitlis", "Bolu", "Burdur",
"Bursa", "Çanakkale", "Çankırı", "Çorum", "Denizli", "Diyarbakır",
"Düzce", "Edirne", "Elazığ", "Erzincan", "Erzurum", "Eskişehir",
"Gaziantep", "Giresun", "Gümüşhane", "Hakkari", "Hatay", "İğdır",
"Isparta", "İstanbul",
"İzmir", "Kahramanmaraş", "Karabük", "Karaman", "Kars",
"Kastamonu", "Kayseri", "Kırıkkale", "Kırklareli", "Kırşehir",
"Kilis", "Kocaeli", "Konya", "Kütahya", "Malatya",
"Manisa", "Mardin", "Mersin", "Muğla", "Muş",
"Nevşehir", "Niğde", "Ordu", "Osmaniye", "Rize",
"Sakarya", "Samsun", "Siirt", "Sinop", "Sivas",
"Şanlıurfa", "Şırnak", "Tekirdağ", "Tokat", "Trabzon",
"Tunceli", "Uşak", "Van", "Yalova", "Yozgat", "Zonguldak", ""};

// Jagged Array oluşturmak için örnek
int[][] jaggedArray = null;

// İlk boyutu belirleme
jaggedArray = new int[81][];

for (int i = 0; i < iller.Length - 2; i++) {
    jaggedArray[i] = new int[i + 1];
    string satir = dosya.ReadLine();
    string[] inputs = satir.Split(' '); //DOSYADAKİ VERİLERİ AYIRMAK İÇİN
    for (int j = 0; j < jaggedArray[i].Length; j++)
    {
        jaggedArray[i][j] = Convert.ToInt32(inputs[j]);
    }
}
// Jagged Array'i yazdırma
for (int i = 0; i < jaggedArray.Length; i++) {

    Console.WriteLine(iller[i + 1] + " ");
    for (int j = 0; j < jaggedArray[i].Length - 1; j++) {
        Console.WriteLine(jaggedArray[i][j] + " ");
    }
    Console.WriteLine();
}
}
```

1.e.2 Ekran görüntüleri

```
Adana
Adıyaman 337
Afyonkarahisar 578 915
Ağrı 973 642 1320
Amasya 603 627 596 740
Ankara 492 732 255 1055 331
Antalya 547 884 287 1424 823 542
Artvin 1001 715 1224 364 679 960 1430
Aydın 880 1217 340 1634 935 595 345 1564
Balıkesir 902 1239 324 1579 845 536 505 1454 291
Bilecik 785 1032 207 1361 626 314 472 1236 525 257
Bingöl 631 347 1096 352 641 887 1188 371 1398 1410 1187
Bitlis 741 410 1287 232 832 1078 1288 545 1589 1601 1378 192
Bolu 688 920 417 1146 411 190 682 1021 735 434 215 1075 1266
Burdur 648 985 166 1402 761 421 121 1390 286 393 351 1166 1357 561
Bursa 856 1104 278 1418 683 385 543 1293 441 152 95 1259 1450 272 422
Çanakkale 1096 1383 517 1697 962 665 695 1572 448 194 374 1538 1729 551 588 269
```

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

//The platform, version, and programming language used

2.a Neuron (Sinir Hücresi) Sınıfı

2.a.1 Kaynak Kod

//İlgili kaynak kodu buraya ekleyiniz.

```
using System;
using System.Collections.Generic;

public class Neuron
{
    private double[] weights;
    private double bias;
    private double learningRate;

    // Constructor
    public Neuron(int inputSize, double learningRate) {
        // Başlangıçta ağırlıkları [0, 1] arasında rastgele pozitif double değerlerle başlat
        Random random = new Random();
        weights = new double[inputSize]; for
        (int i = 0; i < inputSize; i++)
        {
            weights[i] = random.NextDouble();
        }

        // Bias'i de [0, 1] arasında rastgele pozitif double değerle başlat
        bias = random.NextDouble();

        // Öğrenme katsayısını ayarla
        this.learningRate = learningRate;
    }
}
```

```

// Çıktıyı hesapla
public double CalculateOutput(double[] inputs) {
    // Girişlerle ağırlıkların iç çarpımını al ve bias ekleyerek topla
    double sum = bias;
    for (int i = 0; i < weights.Length; i++) {
        sum += weights[i] * inputs[i];
    }

    // Sigmoid aktivasyon fonksiyonunu uygula
    return Sigmoid(sum);
}

// Ağırlıkları güncelle
public void UpdateWeights(double[] inputs, double target) {
    double output = CalculateOutput(inputs);
    double error = target - output;

    // Ağırlıkları güncelle
    for (int i = 0; i < weights.Length; i++) {
        weights[i] += learningRate * error * inputs[i];
    }

    // Bias'i güncelle
    bias += learningRate * error;
}

// Sigmoid aktivasyon fonksiyonu
private double Sigmoid(double x) {
    return 1.0 / (1.0 + Math.Exp(-x));
}

// Yeni veriler üzerinde tahmin yap
public void PredictAndPrint(List<double[]> newTestData) {
    Console.WriteLine("\nYeni Veriler ve Tahmin Sonuçları:");
    foreach (var entry in newTestData)
    {
        double[] inputs = new double[] { entry[0], entry[1] };
        double target = entry[2];

        double output = CalculateOutput(inputs);

        Console.WriteLine($"Giriş: {entry[0]}, {entry[1]} | Hedef Çıktı: {target} | Tahmin Çıktı: {output}");
    }
}

}

class Program
{
    static void Main()
    {
        // Eğitim verileri
        List<double[]> trainingData = new List<double[]>
        {
            new double[] { 7.6 / 10, 11.0 / 15, 77.0 / 100 },

```



```

new double[] { 8.0 / 10, 10.0 / 15, 70.0 / 100 },
new double[] { 6.6 / 10, 8.0 / 15, 55.0 / 100 }, new
double[] { 8.4 / 10, 10.0 / 15, 78.0 / 100 }, new
double[] { 8.80 / 10, 12.0 / 15, 95.0 / 100 }, new
double[] { 7.2 / 10, 10.0 / 15, 67.0 / 100 }, new
double[] { 8.1 / 10, 11.0 / 15, 80.0 / 100 }, new
double[] { 9.5 / 10, 9.0 / 15, 87.0 / 100 }, new
double[] { 7.3 / 10, 9.0 / 15, 60.0 / 100 }, new
double[] { 8.9 / 10, 11.0 / 15, 88.0 / 100 }, new
double[] { 7.5 / 10, 11.0 / 15, 72.0 / 100 }, new
double[] { 7.6 / 10, 9.0 / 15, 58.0 / 100 }, new
double[] { 7.9 / 10, 10.0 / 15, 70.0 / 100 }, new
double[] { 8.0 / 10, 10.0 / 15, 76.0 / 100 }, new
double[] { 7.2 / 10, 9.0 / 15, 58.0 / 100 }, new
double[] { 8.8 / 10, 10.0 / 15, 81.0 / 100 }, new
double[] { 7.6 / 10, 11.0 / 15, 74.0 / 100 }, new
double[] { 7.5 / 10, 10.0 / 15, 67.0 / 100 }, new
double[] { 9.0 / 10, 10.0 / 15, 82.0 / 100 }, new
double[] { 7.7 / 10, 9.0 / 15, 62.0 / 100 }, new
double[] { 8.1 / 10, 11.0 / 15, 82.0 / 100 },

// Diğer veriler buraya eklenecek...
};

// Yeni veriler
List<double[]> newTestData = new List<double[]>
{
    new double[] { 8.2 / 10, 12.0 / 15, 0 }, // Sınav Sonucu belirtilmemiş, 0 olarak geçici olarak atandı
    new double[] { 6.5 / 10, 9.0 / 15, 0 },
    new double[] { 9.0 / 10, 13.0 / 15, 0 },
    new double[] { 7.0 / 10, 10.0 / 15, 0 },
    new double[] { 8.5 / 10, 11.0 / 15, 0 }
};

// Giriş boyutu
int inputSize = 2; // Çalışma Süresi ve Derse Devam

// Eğitim sonrası tahminler için kullanılacak parametreler
int[] epochsList = { 10, 50, 100 };
double[] learningRatesList = { 0.01, 0.025, 0.05 };

// Eğitim sonrası tahminler ve MSE değerlerini hesapla ve yazdır
CalculateAndPrintMSEValues(trainingData, newTestData, inputSize, epochsList, learningRatesList);
}

// Eğitim sonrası tahminler ve MSE değerlerini hesapla ve yazdır
static void CalculateAndPrintMSEValues(List<double[]> trainingData, List<double[]> newTestData, int inputSize, int[]
epochsList, double[] learningRatesList)
{
    Console.WriteLine("Eğitim Sonrası Tahminler ve MSE Değerleri:");

    foreach (var epochs in epochsList) {
        foreach (var learningRate in learningRatesList) {
            // Yapay sinir hücresi oluştur
            Neuron neuron = new Neuron(inputSize, learningRate); //
            Eğitim işlemi
            for (int epoch = 0; epoch < epochs; epoch++) {

```

```

        foreach (var entry in trainingData)
        {
            double[] inputs = new double[] { entry[0], entry[1] };
            double target = entry[2];

            // Ağı eğit ve ağırlıkları güncelle
            neuron.UpdateWeights(inputs, target);
        }
    }

    // Eğitim sonrası tahminler
    Console.WriteLine($"Epochs: {epochs}, Learning Rate: {learningRate}");
    foreach (var entry in trainingData) {
        double[] inputs = new double[] { entry[0], entry[1] };
        double target = entry[2];

        double output = neuron.CalculateOutput(inputs);

        Console.WriteLine($"Giriş: {entry[0]}, {entry[1]} | Hedef Çıktı: {target} | Tahmin Çıktı: {output}");
    }

    // MSE Hesaplama
    double mse = CalculateMeanSquareError(neuron, trainingData);

    Console.WriteLine($"MSE (Mean Square Error): {mse}");
    // Yeni veriler üzerinde tahmin yap
    neuron.PredictAndPrint(newTestData);
}
}

// MSE Hesaplama
static double CalculateMeanSquareError(Neuron neuron, List<double[]> data) {
    double sumSquaredErrors = 0;

    foreach (var entry in data)
    {
        double[] inputs = new double[] { entry[0], entry[1] };
        double target = entry[2];

        double output = neuron.CalculateOutput(inputs);

        double error = target - output;
        sumSquaredErrors += error * error;
    }

    return sumSquaredErrors / data.Count;
}
}

```

2.a.2 Açıklama

//Kullanılan veri yapıları ve algoritmanın kısaca anlatımını burada gerçekleştiriniz

Bu program, yapay sinir ağı temel alınarak bir sinir hücresi (neuron) sınıfını ve bu sinir

hücrelerini kullanarak eğitim verileri üzerinde öğrenme gerçekleştiren bir uygulamayı içermektedir. İşte kullanılan veri yapıları ve algoritma özeti:

1.	Sinir Hücresi (Neuron)	<ul style="list-style-type: none">? Neuron sınıfı, yapay sinir ağı temsili için kullanılır.? Her sinir hücresi, girişlere ait ağırlıkları, bir bias değeri ve öğrenme katsayısını içerir.? Sigmoid aktivasyon fonksiyonu kullanılarak çıktı hesaplanır.? Ağırlıklar ve bias, eğitim verileri üzerinde güncellenir.
2.	Eğitim Verileri:	<ul style="list-style-type: none">? <code>trainingData</code> listesi, eğitim için kullanılacak giriş-veri çiftlerini içerir.? Her bir giriş, çalışma süresi ve derse devam durumu gibi özellikleri temsil eder.? İlgili çıktı, sınav başarı yüzdesini ifade eder.
3.	Yeni	<ul style="list-style-type: none">? <code>newTestData</code> listesi, eğitim sonrası sinir ağı tarafından tahmin yapılacak yeni veri setini içerir.? Bu veri setinde sınav sonuçları belirtilmemiş ve geçici olarak 0 olarak atandığı belirtilmiştir.
4.	Eğitim ve Tahmin:	<ul style="list-style-type: none">? <code>CalculateAndPrintMSEValues</code> metodu, belirli epoch sayıları ve öğrenme katsayıları için sinir ağını eğitir ve eğitim sonrasında tahminlerde bulunur.? Her bir eğitim seti için ortalama kare hatası (MSE) hesaplanır ve yazdırılır.? <code>PredictAndPrint</code> metodu, yeni veriler üzerinde tahmin yapar ve sonuçları ekrana yazdırır.
5.	MSE	<ul style="list-style-type: none">? <code>CalculateMeanSquareError</code> metodu, belirli bir sinir hücresi ve veri seti için ortalama kare hatasını hesaplar.
6.	Ana	<ul style="list-style-type: none">? <code>Main</code> metodu, eğitim verilerini, yeni verileri ve sinir ağının parametrelerini içerir.? Belirli epoch sayıları ve öğrenme katsayıları için sinir ağını eğitir ve eğitim sonrasında tahminlerde bulunur.

Bu program, basit bir sinir hücresi üzerinden eğitim ve tahmin işlemlerini gerçekleştirmektedir. Eğitim sonrasında ortalama kare hatası (MSE) değerleri değerlendirilerek sinir ağının performansı analiz edilmektedir.

2.b Eğitim

2.b.1 Kaynak Kod

// İlgili kaynak kodu buraya ekleyiniz.

```
// Yapay sinir hücresi oluşturur
Neuron neuron = new Neuron(inputSize, learningRate); //
Eğitim işlemi
for (int epoch = 0; epoch < epochs; epoch++)
{
    foreach (var entry in trainingData)
    {
        double[] inputs = new double[] { entry[0], entry[1] };
        double target = entry[2];

        // Ağı eğit ve ağırlıkları günceller
        neuron.UpdateWeights(inputs, target);
    }
}
```

2.b.2 Açıklama

//Eğitimde kullanılan verilerden modelinizin hatasını 10 devir için MSE (Mean Square Error) cinsinden hesaplayıp yazınız. MSE'nin ne amaçla kullanıldığını ve formülünü araştırıp, kaynağıyla birlikte kısaca anlatımını gerçekleştiriniz.

MSE (Mean Square Error), bir regresyon modelinin tahminlerinin gerçek değerlerden ne kadar uzak olduğunu ölçen bir performans metriğidir. MSE, hata karelerinin ortalamasını hesaplayarak modelin ne kadar iyi veya kötü performans gösterdiğini değerlendirir.

MSE'nin amaçları şunlardır:

1. Modelin tahminlerinin gerçek değerlere ne kadar yakın olduğunu nicel olarak ölçmek.
2. Modelin performansını değerlendirerek farklı modelleri karşılaştırmak.

2.b.3 Ekran Görüntüleri

//Konsol çıktısına ait ekran görüntülerini buraya ekleyiniz

Eğitim Sonrası Tahminler ve MSE Değerleri:

```
Epochs: 10, Learning Rate: 0,01
Giriş: 0,76, 0,7333333333333333 | Hedef Çıktı: 0,77 | Tahmin Çıktı: 0,7704860620896924
Giriş: 0,8, 0,6666666666666666 | Hedef Çıktı: 0,7 | Tahmin Çıktı: 0,7666026214852995
Giriş: 0,6599999999999999, 0,5333333333333333 | Hedef Çıktı: 0,55 | Tahmin Çıktı: 0,7430247192161755
Giriş: 0,8400000000000001, 0,6666666666666666 | Hedef Çıktı: 0,78 | Tahmin Çıktı: 0,7693181035124121
Giriş: 0,8800000000000001, 0,8 | Hedef Çıktı: 0,95 | Tahmin Çıktı: 0,7847972166584509
Giriş: 0,72, 0,6666666666666666 | Hedef Çıktı: 0,67 | Tahmin Çıktı: 0,7611052451746132
Giriş: 0,8099999999999999, 0,7333333333333333 | Hedef Çıktı: 0,8 | Tahmin Çıktı: 0,7738371727682406
Giriş: 0,95, 0,6 | Hedef Çıktı: 0,87 | Tahmin Çıktı: 0,7701752413441953
Giriş: 0,73, 0,6 | Hedef Çıktı: 0,6 | Tahmin Çıktı: 0,7550052184940022
Giriş: 0,89, 0,7333333333333333 | Hedef Çıktı: 0,88 | Tahmin Çıktı: 0,7791265897198946
Giriş: 0,75, 0,7333333333333333 | Hedef Çıktı: 0,72 | Tahmin Çıktı: 0,7698116723525091
Giriş: 0,76, 0,6 | Hedef Çıktı: 0,58 | Tahmin Çıktı: 0,7571131150716055
Giriş: 0,79, 0,6666666666666666 | Hedef Çıktı: 0,7 | Tahmin Çıktı: 0,7659202875936753
Giriş: 0,8, 0,6666666666666666 | Hedef Çıktı: 0,76 | Tahmin Çıktı: 0,7666026214852995
Giriş: 0,72, 0,6 | Hedef Çıktı: 0,58 | Tahmin Çıktı: 0,7542998464401067
Giriş: 0,8800000000000001, 0,6666666666666666 | Hedef Çıktı: 0,81 | Tahmin Çıktı: 0,7720113879796128
Giriş: 0,76, 0,7333333333333333 | Hedef Çıktı: 0,74 | Tahmin Çıktı: 0,7704860620896924
Giriş: 0,75, 0,6666666666666666 | Hedef Çıktı: 0,67 | Tahmin Çıktı: 0,763177120661989
Giriş: 0,9, 0,6666666666666666 | Hedef Çıktı: 0,82 | Tahmin Çıktı: 0,773349694088004
Giriş: 0,77, 0,6 | Hedef Çıktı: 0,62 | Tahmin Çıktı: 0,757813003779139
Giriş: 0,8099999999999999, 0,7333333333333333 | Hedef Çıktı: 0,82 | Tahmin Çıktı: 0,7738371727682406
MSE (Mean Square Error): 0,0107250676323169
```

2.c Modelin Görmediği Veriden Sınav Sonucu Tahminleme

2.c.1 Kaynak Kod

// İlgili kaynak kodu buraya ekleyiniz.

```
public void PredictAndPrint(List<double[]> newTestData) {
    Console.WriteLine("\nYeni Veriler ve Tahmin Sonuçları:");
    foreach (var entry in newTestData)
    {
        double[] inputs = new double[] { entry[0], entry[1] };
        double target = entry[2];

        double output = CalculateOutput(inputs);

        Console.WriteLine($"Giriş: {entry[0]}, {entry[1]} | Hedef Çıktı: {target} | Tahmin Çıktı: {output}");
    }
}
```

2.c.2 Sonuçlar/Ekran görüntüleri

// Eğitim verileri dışında farklı 5 tane girdi verisi oluşturulması ve modelin bu veriler üzerinde yaptığı tahminleme sonuçlarının ekran görüntüsünü ekleyiniz.

```
Yeni Veriler ve Tahmin Sonuçları:
Giriş: 0,82, 0,8 | Hedef Çıktı: 0 | Tahmin Çıktı: 0,7536516220510191
Giriş: 0,65, 0,6 | Hedef Çıktı: 0 | Tahmin Çıktı: 0,7107622170703596
Giriş: 0,9, 0,8666666666666667 | Hedef Çıktı: 0 | Tahmin Çıktı: 0,7692661334006082
Giriş: 0,7, 0,6666666666666666 | Hedef Çıktı: 0 | Tahmin Çıktı: 0,7248041380914674
Giriş: 0,85, 0,7333333333333333 | Hedef Çıktı: 0 | Tahmin Çıktı: 0,7489984414576654
```

2.d Deneyler

2.d.1 Kaynak Kod

// İlgili kaynak kodu buraya ekleyiniz.

```
foreach (var epochs in epochsList) {
    foreach (var learningRate in learningRatesList)
    {
        // Yapay sinir hücresi oluşturur
        Neuron neuron = new Neuron(inputSize, learningRate);
        // Eğitim işlemi
        for (int epoch = 0; epoch < epochs; epoch++) {
            foreach (var entry in trainingData)
            {
                double[] inputs = new double[] { entry[0], entry[1] };
                double target = entry[2];

                // Ağı eğit ve ağırlıkları günceller
                neuron.UpdateWeights(inputs, target);
            }
        }

        // Eğitim sonrası tahminler
        Console.WriteLine($"Epochs: {epochs}, Learning Rate: {learningRate}");
        foreach (var entry in trainingData)
        {
            double[] inputs = new double[] { entry[0], entry[1] };
            double target = entry[2];

            double output = neuron.CalculateOutput(inputs);

            Console.WriteLine($"Giriş: {entry[0]}, {entry[1]} | Hedef Çıktı: {target} | Tahmin Çıktı: {output}");
        }

        // MSE Hesaplama
        double mse = CalculateMeanSquareError(neuron, trainingData);

        Console.WriteLine($"MSE (Mean Square Error): {mse}");

        // Yeni veriler üzerinde tahmin yapar
        neuron.PredictAndPrint(newTestData);
    }
}
```

2.d.2 Sonuçlar

Deney	10 Epok	50 Epok	100 Epok		
1					Daha fazla epoch kullanmak, modelin eğitim verilerine daha fazla adapte olmasına olanak tanır. Ancak, aşırı eğitim (overfitting) riskini de artırabilir, yani model eğitim verilerine çok özelleşip genelleme yeteneğini kaybedebilir. Örneğin, epoch sayısını artırmak, başlangıçta daha düşük olan MSE'yi (Mean Square Error) azaltabilir, ancak aşırı eğitim olasılığını artırarak yeni veriler üzerinde performansı düşürebilir.
$\lambda = 0.01$	0,7704860620896924	0,7439123654134169		0,7364896402015487	Öğrenme katsayısı, ağırlıkların güncellenme miktarını belirler. Düşük bir öğrenme katsayısı, modelin yavaş öğrenmesine neden olabilir, ancak daha iyi genelleme yapabilir. Yüksek bir öğrenme katsayısı, hızlı öğrenme sağlayabilir ancak aşırı dalgalanmalara neden olabilir.

Deney 3	10 Epok	50 Epok	100 Epok
$\lambda = 0.01$	0,7139443361940734	0,7338408880211733	0,7366380585356934
$\lambda = 0.025$	0,759555083521881	0,7360127472193639	0,7360364299712091
$\lambda = 0.05$	0,7336267624724294	0,7360418588886568	0,7358381611283961
$\lambda = 0.025$	0,7456551006368042	0,7421161405351587	0,7425994789799742
$\lambda = 0.05$	0,7458757083506675	0,7340621733672062	0,7377976076082549

Deney 2	10 Epok	50 Epok	100 Epok
$\lambda = 0.01$	0,7280705962513019	0,7513929265607051	0,7603649913485128
$\lambda = 0.025$	0,7797835418808418	0,7606461414137745	0,771349032269944
$\lambda = 0.05$	0,746197110452771	0,7568149155087079	0,7643851415499773

Öz

değerlendirme Tablosu

Proje 1 Maddeleri	Not	Tahmini Not	Açıklama
1.a	10	10	Yapıldı. Uzaklık matrisi txt dosyasından okutularak jagged array şeklinde oluşturuldu ve iller 81 elemanlı bir dizide tutuldu.
1.b	5	5	Yapıldı. İl ve mesafe kullanıcıdan alınarak verilen ilden belli bir uzaklığa kadar olan iller ve uzaklıkları döngü ile şehirlerin uzaklıkları üzerinde gezip verilen mesafeden yakın olanlar eklendi.
1.c	5	5	Yapıldı. En uzak ve en yakın illerin adları ve aralarındaki mesafe yine döngü ile gezilerek mesafenin en uzak ya da en yakın mesafeyi geçtiği durumda değişken güncellenerek son halini aldı.
1.d	15	15	Yapıldı. Başlangıç şehri ve maksimum mesafe parametrelerini kullanarak, şehirler arasındaki mesafeleri içeren bir veri yapısıyla en uzun yolu bulan bir genişlik öncelikli arama (BFS) algoritması uygular. Algoritma, bir sıra kullanarak şehirleri ziyaret eder, her adımda en uzun yolu günceller ve maksimum mesafeyi

			aşmayan şehirleri keşfeder. Sonuç olarak, en uzun yolun mesafesi, yolun kendisi ve ziyaret edilen şehir sayısı içeren bir ArrayList döndürülür.
1.e	5	5	Yapıldı.Kod, rastgele indislerle şehir seçer, bu şehirler arasındaki mesafeleri bir matriste depolar, ardından seçilen şehirlerin isimlerini ve aralarındaki mesafeleri ekrana yazdırır.
2.a	10	8	Yapıldı. Veri tablosundaki bölme işlemleri manuel olarak yapıldı.
2.b	15	13	Yapıldı.Parametre olarak eğitim verilerini, test verilerini, giriş boyutunu, epoch sayılarını ve öğrenme hızlarını alır. Metodun içinde, her bir epoch ve öğrenme hızı kombinasyonu için ayrı ayrı eğitim yapılır ve sonuçlar ekrana yazdırılır. Epok sayılarının hepsi aynı metotta hesaplanıldı.
2.c	5	5	Yapıldı. Farklı 5 tane girdi verisi listeye atandı. Liste metotlar tarafından kullanılarak hesaplandı.
2.d	10	8	Yapıldı.Parametre olarak eğitim verilerini, test verilerini, giriş boyutunu, epoch sayılarını ve öğrenme hızlarını alır. Metodun içinde, her bir epoch ve öğrenme hızı kombinasyonu için ayrı ayrı eğitim yapılır ve sonuçlar ekrana yazdırılır. Epok sayılarının hepsi aynı metotta hesaplanıldı. MSE değerleri 9 farklı hesaplama için çıktıların altına yazıldı.
Rapor	10	10	Yapıldı. Raporda istenenleri eksiksiz bir şekilde yazmaya çalıştık.
Öz değerlendirme Tablosu	10	10	Yapıldı. Şıkları açıklayıcı şekilde açıklamalar yazarak yapıldı.
Toplam	100		