

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

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

**2021–2022 FALL SEMESTER**

**PROJECT-1 REPORT**

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

**DELIVERY DATE**

06/12/2021

**PREPARED BY**

Student\_number, Name Surname

05190000028, Altuğ Şahin

05200000030, Osman Bora Yılmazmış

İçindekiler

[1) POINTS IN A 2D PLANE 2](#_Toc87438605)

[1.a Rastgele Nokta Üretimi 2](#_Toc87438606)

[1.a.1 Kodlar 2](#_Toc87438607)

[1.a.2 Ekran görüntüleri 2](#_Toc87438608)

[1.a.3 Açıklama 2](#_Toc87438609)

[1.b Uzaklık Matrisi 3](#_Toc87438610)

[1.b.1 Kodlar 3](#_Toc87438611)

[1.b.2 Ekran görüntüleri 3](#_Toc87438612)

[1.b.3 Açıklama 3](#_Toc87438613)

[1.c En yakın komşu yöntemi ile dolaşma 4](#_Toc87438614)

[1.c.1 Kodlar 4,5](#_Toc87438615)

[1.c.2 Ekran görüntüleri 6](#_Toc87438616)

[1.c.3 Açıklama 7](#_Toc87438617)

[2) DEVELOPING A SIMPLE ARTIFICIAL NEURON and CLASSIFICATION 2](#_Toc87438618)

[2.a Neuron (Sinir Hücresi) sınıfı kaynak kodu 2](#_Toc87438619)

[2.b Eğitim 2](#_Toc87438620)

[2.b.1 Kaynak Kod 2](#_Toc87438621)

[2.b.2 Sonuçlar/Ekran görüntüleri 3](#_Toc87438622)

[2.b.3 Açıklama 3](#_Toc87438623)

[2.c Test 3](#_Toc87438624)

[2.c.1 Test Verisi 3](#_Toc87438625)

[2.c.2 Kaynak Kod 3](#_Toc87438626)

[2.c.2 Sonuçlar/Ekran görüntüleri 3](#_Toc87438627)

[2.c.3 Açıklama 3](#_Toc87438628)

[Öz değerlendirme Tablosu 3](#_Toc87438629)

# 1) POINTS IN A 2D PLANE

//The platform, version, and programming language used

## 1.a Rastgele Nokta Üretimi

### 1.a.1 Kodlar

static double[,] matris\_olustur\_rastgele\_deger\_ver(int n, int width, int height)

{

double[,] f = new double[n, 2];

Random rnd = new Random();

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

{

double rastgelesayi = rnd.NextDouble() \* width;

double rastgelesayi2 = rnd.NextDouble() \* height;

f[i, 0] = rastgelesayi;

f[i, 1] = rastgelesayi2;

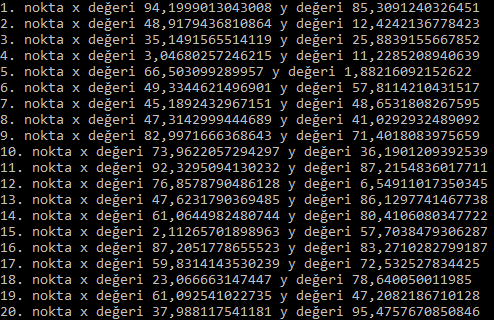
Console.WriteLine("{0}. nokta x değeri {1} y değeri {2} ", i + 1, f[i, 0], f[i, 1]);

}

return f;

}

### 1.a.2 Ekran görüntüleri



### 1.a.3 Açıklama

matris\_olustur\_rastgele\_deger\_ver metodunda bizden istenen gerekli parametreleri alarak nx2 lik double tipinde bir matris oluşturuyoruz .Daha sonra her bir noktanın x ve y değerini iki tane for döngüsünün içinde parametremizden gelen değerlere göre rastgele noktalar üreterek matrise ekliyoruz. Bu noktaların tek tek x ve y değerlerini ekrana yazdırıyoruz.

## 1.b Uzaklık Matrisi

### 1.b.1 Kodlar

static double[,] uzaklık\_matrisi\_olustur(double[,] m)

{

double[,] uzaklıkmatrisi = new double[m.GetLength(0), m.GetLength(0)];

for (int i = 0; i < m.GetLength(0); i++)

{

for (int j = 0; j < m.GetLength(0); j++)

{

double x1 = m[i, 0];

double y1 = m[i, 1];

double x2 = m[j, 0];

double y2 = m[j, 1];

uzaklıkmatrisi[i, j] =Math.Sqrt((Math.Pow((x2-x1),2)+ Math.Pow((y2 - y1), 2)));

Console.Write(uzaklıkmatrisi[i, j]+" ");

}

Console.WriteLine(" ");

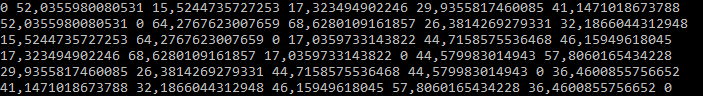
}

return uzaklıkmatrisi;

}

### 1.b.2 Ekran görüntüleri

Ekrana yazdırdığımız değerin daha rahat görünebilmesi için 6 tane rastgele nokta içeren uzaklık matrisinin ekran görüntüsü alınmıştır.



### 1.b.3 Açıklama

uzaklık\_matrisi\_olustur metoduna daha önce daha önce oluşturduğumumz noktaların x ve y değerlerini tutan matrisimizi parametre olarak veriyoruz. İlk aşamada nxn lik uzaklık matrisimizi tanımlıyoruz. Daha sonra iki for döngüsü içinde tek tek x1,y1,x2,y2 değerlerini çekip o değerlerin öklid uzaklık formülünü hesaplayım uzaklık matrisimize ekliyoruz.

## 

## 1.c En yakın komşu yöntemi ile dolaşma

### 1.c.1 Kodlar

static void en\_yakin\_komsu\_gez(Double[,] m)

{

ArrayList rastgelebaslangicnoktaları = new ArrayList();

for (int a = 0; a<10; a++)

{

Console.WriteLine("Tur " + (a+1));

Random rnd = new Random();

int randomnokta = rnd.Next(m.GetLength(0) - 1);

while (rastgelebaslangicnoktaları.Contains(randomnokta) == true)

{

randomnokta = rnd.Next(m.GetLength(0) - 1);

}

rastgelebaslangicnoktaları.Add(randomnokta);

int yeninokta = -1;

double toplam\_uzaklık = 0;

double[,] uzaklıkmatrisikopya = new double[m.GetLength(0), m.GetLength(0)];

for (int i = 0; i < m.GetLength(0); i++)

{

for (int j = 0; j < m.GetLength(0); j++)

{

uzaklıkmatrisikopya[i, j] = m[i, j];

}

}

int[] dolanan\_noktalar = new int[m.GetLength(0)];

for (int i = 0; i < m.GetLength(0); i++)

{

double min\_uzaklık = 99999999999;

for (int j = 0; j < m.GetLength(0); j++)

{

if (uzaklıkmatrisikopya[randomnokta, j] == 0)

{

uzaklıkmatrisikopya[randomnokta, j] = 89999999999;

uzaklıkmatrisikopya[j, randomnokta] = 89999999999;

}

else if (uzaklıkmatrisikopya[randomnokta, j] < min\_uzaklık)

min\_uzaklık = uzaklıkmatrisikopya[randomnokta, j];

if (uzaklıkmatrisikopya[randomnokta, j] == min\_uzaklık)

{

yeninokta = j;

for (int l = 0; l < m.GetLength(0); l++)

{

uzaklıkmatrisikopya[l, randomnokta] = 99999999999;

}

uzaklıkmatrisikopya[randomnokta, j] = 99999999999;

uzaklıkmatrisikopya[j, randomnokta] = 99999999999;

}

}

dolanan\_noktalar[i] = randomnokta;

randomnokta = yeninokta;

}

for (int t = 0; t < dolanan\_noktalar.Length; t++)

{

Console.Write(dolanan\_noktalar[t] + "-");

}

Console.WriteLine(" ");

int k = 1;

for (int l = 0; l < dolanan\_noktalar.Length; l++)

{

toplam\_uzaklık += m[dolanan\_noktalar[l], dolanan\_noktalar[k]];

k += 1;

if (k == dolanan\_noktalar.Length)

break;

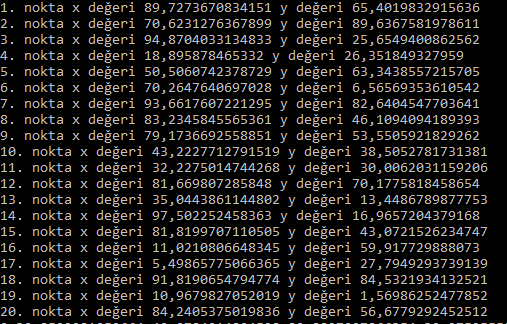
}

Console.WriteLine("{0}. turda dolanan toplam uzaklık {1} ",(a+1),toplam\_uzaklık);

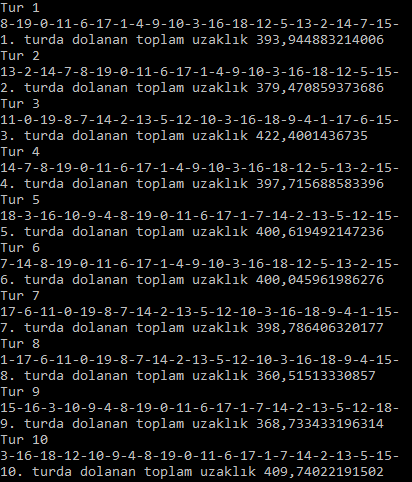
}

}

### 1.c.2 Ekran görüntüleri



Yukarıda verilen noktaların X ve Y değerlerine göre aşağıdaki çıktı ortaya çıkmıştır.



### 1.c.3 Açıklama

Öncelikle en\_yakin\_komsu\_gez metodumuza uzaklık matrisi gelecek şekilde parametremizi veriyoruz. Bu metodumuzu 10 kere ve her biri farklı noktadan başlayacak şekilde döndüreceğimiz için öncelikle rastgele başlangıç noktalarını tutan arraylistimizi oluşturuyoruz. For döngüsüne girdiğimizde hangi turdaysak o turu yazdırıyoruz ve hangi noktadan başladığımızı while döngüsü içinde arraylistimize ekliyoruz eğer ki yeni oluşturulan rastgele başlangıç noktamız arraylistin içinde varsa while döngüsünde farklı bir başlangıç noktası bulana kadar tekrardan döndürülüyor. Daha sonra noktaları dolaşırken tekrardan aynı noktaya gitmemek için uzaklık matrisinde gidilen en kısa noktanın değerini 99999999 olarak değiştiriyoruz. Böylece min uzaklık olarak seçim yapan algoritmada daha önce gidilen minimum uzaklığa tekrardan gidilmiyor. Uzaklık matrisimizin içini değiştirdiğimizden tutulan değerlerin bozulmaması adına uzaklık matrisimizin kopyasını oluşturuyoruz ve bu kopya içinde değişiklikler yaparak en kısa mesafeyi buluyoruz. Her bir başlangıç noktası için toplam uzaklığı hesaplatıp ekrana yazdırıyoruz.

# 2) DEVELOPING A SIMPLE ARTIFICIAL NEURON and CLASSIFICATION

//The platform, version, and programming language used

## 2.a Neuron (Sinir Hücresi) sınıfı kaynak kodu

//İlgili kaynak kodu buraya ekleyiniz.

class SinirHücresi

{

double LambdaSabiti = 0.05;

double Agirlikx1;

double Agirlikx2;

Random RandomSayi = new Random();

public SinirHücresi()

{

Agirlikx1 = RandomSayi.NextDouble() \*2 -1;

Agirlikx2 = RandomSayi.NextDouble() \* 2 - 1;

}

public int egit(int[][] egitimVerisi)

{

int dogruSayisi = 0;

foreach(int[] i in egitimVerisi)

{

int cikti = Cikti(i[0] \* 0.1,i[1] \* 0.1);

if (cikti != i[2])

{

AgirliklariDegistir(i[0] \* 0.1 , i[1] \* 0.1, i[2] - cikti);

}

else {

dogruSayisi++;

}

}

return dogruSayisi;

}

private void AgirliklariDegistir(double x1, double x2,int tEksiO)

{

Agirlikx1 += LambdaSabiti \* tEksiO \* x1;

Agirlikx2 += LambdaSabiti \* tEksiO \* x2;

}

public int Cikti(double x1, double x2)

{

double toplam = Agirlikx1 \* x1 + Agirlikx2 \* x2;

if(toplam > 0.5)

{

return 1;

}else

{

return -1;

}

}

}

## 2.b Eğitim

### 2.b.1 Kaynak Kod

//Veri setini geliştirdiğiniz sinir hücresi sınıfına aktararak eğitim işlemini gerçekleştiren ve doğruluk değerlerini üreten kaynak kodu buraya ekleyiniz.

static void HucreEgitimi(SinirHücresi Shücre, int[][] veriSetleri, int epokSayisi) {

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

{

int toplamDogruluk = Shücre.egit(veriSetleri);

Console.WriteLine("Epok:" + i +" " + "Dogruluk:" + ((double)toplamDogruluk / veriSetleri.Length) \* 100);

}

}

public int egit(int[][] egitimVerisi)

{

int dogruSayisi = 0;

foreach(int[] i in egitimVerisi)

{

int cikti = Cikti(i[0] \* 0.1,i[1] \* 0.1);

if (cikti != i[2])

{

AgirliklariDegistir(i[0] \* 0.1 , i[1] \* 0.1, i[2] - cikti);

}

else {

dogruSayisi++;

}

}

return dogruSayisi;

}

private void AgirliklariDegistir(double x1, double x2,int tEksiO)

{

Agirlikx1 += LambdaSabiti \* tEksiO \* x1;

Agirlikx2 += LambdaSabiti \* tEksiO \* x2;

}

public int Cikti(double x1, double x2)

{

double toplam = Agirlikx1 \* x1 + Agirlikx2 \* x2;

if(toplam > 0.5)

{

return 1;

}else

{

return -1;

}

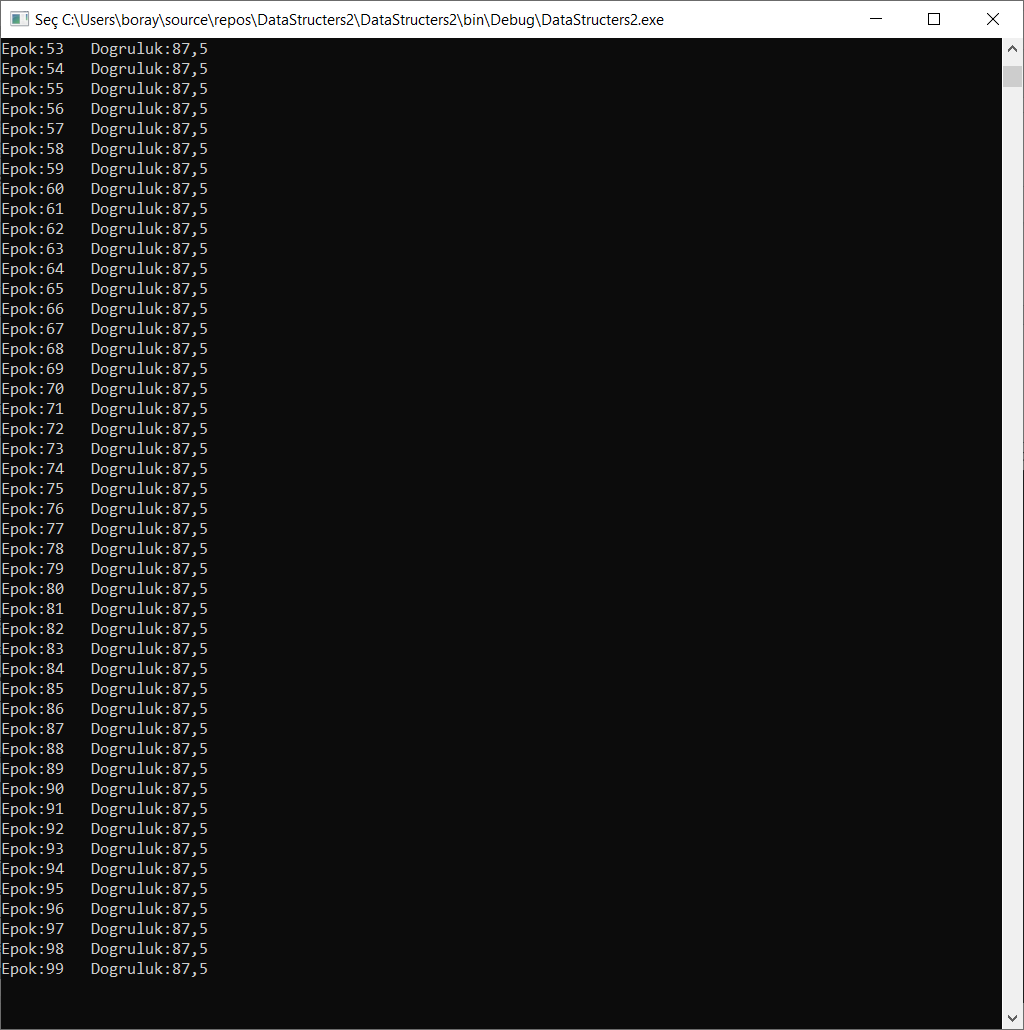
}

### 2.b.2 Sonuçlar/Ekran görüntüleri

//Doğruluk değerlerinin hesaplanmasına ilişkin konsol çıktısına ait ekran görüntülerini buraya ekleyiniz.

metin içeren bir resim

Açıklama otomatik olarak oluşturuldu



Egitimverisinin 10 ve 100 epok için sonuclari

### 2.b.3 Açıklama

//Kullanılan veri yapıları ve algoritmanın kısaca anlatımını burada gerçekleştiriniz

Hücre eğitimi için kullandığım metodlar kısaca şunlardır :

1-) static void HucreEgitimi(SinirHücresi Shücre, int[][] veriSetleri, int epokSayisi)

2-) public int egit(int[][] egitimVerisi)

3-) private void AgirliklariDegistir(double x1, double x2,int tEksiO)

4-) public int Cikti(double x1, double x2)

Yukarıdaki metodların özellikleri sondan başlayacak sekilde sırasıyla;

public int Cikti(double x1, double x2)

--Toplama fonksiyonu ve eşik fonkisyonu burada bulunmaktadır.

--cikti metodu x1’i x1 le alakalı ağırlıkla x2’yi x2’yle alakalı ağırlıkla çarpar ve birbiriyle toplar; daha sonra oluşan bu değer 0.5 ten büyükse, 1 döndürür değilse -1 döndürür.

--kısaca Perceptron Modeli ve İşleyişini gerçekleştirmektedir.

private void AgirliklariDegistir(double x1, double x2,int tEksiO)

--ismindende anlaşılacağı üzere ağın ürettiği çıktı olması gereken değerden farklı ise ağırlıkları

λ\*(t-o)\*x kadar arttırır

public int egit(int[][] egitimVerisi)

--Egitimin gerçekleştiği metottur.

--içinde doğruSayisi değişkeni tanımlanmıştır.

--Egitim verilerini aldıktan sonra girdileri önce 10’a böler daha sonra Cikti metoduna gönderir.

-- Cikti metodundan gelen değerle target yani hedef değerimizi karşılaştırır.

--Eğer farklıysa AgirliklariDegistir metoduna gönderir ve ağırlıklarında değişiklikler gerçekleşir.

--aynıysa doğruSayisi bir arttırılır ve döndürülür.

static void HucreEgitimi(SinirHücresi Shücre, int[][] veriSetleri, int epokSayisi)

--Main metodunun içinde çağırdığımız static metottur.

-- İçinde bir döngü bulunur ve girilen epokSayisi kadar döngü devam eder.

-- Döngüde egit metodundan oluşan toplam doğruluk değerleri tutulur ve önce kaçıncı epok olduğu daha sonra;

((double)toplamDogruluk / veriSetleri.Length) \* 100 şeklinde doğruluk değerleri konsolumuza yazdırılır

Kodumuzun Egitim kısmı burada sonlanır.

## 2.c Test

### 2.c.1 Test Verisi

// Oluşturduğunuz test verisini buraya ekleyiniz.

int[][] TestVerisi1= new int[][] { new int[] { 7, 8, 1 }, new int[] { -5, 4, -1 }, new int[] { -3, -5, -1 }, new int[] { 1, -5, -1 }, new int[] { 1, 4, 1 }, new int[] { -9, 6, -1 }, new int[] { 2, 7, 1 }, new int[] { -4, -5, -1 }, new int[] { -6, -2, -1 } };

int[][] TestVerisi2 = new int[][] { new int[] { 2, 7, 1 }, new int[] { 10, 7, 1}, new int[] { 3, -6, -1 }, new int[] { 7, -5, 1 }, new int[] { -1, 4, 1 }, new int[] { 6, 5, 1 }, new int[] { 3, -7, -1 }, new int[] { 4, -5, -1 }, new int[] { 6, -2, 1 } };

int[][] TestVerisi3 = new int[][] { new int[] { -8, -4, -1 }, new int[] { -9, 7, -1 }, new int[] { -3, 6, 1 }, new int[] { 4, -9, -1 }, new int[] { -1, 2, 1 }, new int[] { -6, -5, -1 }, new int[] { -8, 10, 1 }, new int[] { 2, -6, -1 }, new int[] { 7, 2, 1 } };

int[][] TestVerisi4 = new int[][] { new int[] { -5, -2, -1 }, new int[] { -1, 3, 1}, new int[] { 7, 2, 1 }, new int[] { 6, 3, 1 }, new int[] { -5, -2, -1 }, new int[] { 10, 0, 1 }, new int[] { -4, 10, 1 }, new int[] { -2, -6, -1 }, new int[] { 1, 1, 1 } };

int[][] TestVerisi5 = new int[][] { new int[] { -8, -4, -1 }, new int[] { -9, 7, -1 }, new int[] { -3, 6, 1 }, new int[] { 4, -9, -1 }, new int[] { -1, 2, 1 }, new int[] { -6, -5, -1 }, new int[] { -8, 10, 1 }, new int[] { 2, -6, -1 }, new int[] { 7, 2, 1 } };

### 2.c.2 Kaynak Kod

//Eğittiğiniz modeli test etmek amacıyla test verisini sinir hücresi sınıfına aktaran ve başarı değerini hesaplayan kaynak kodu buraya ekleyiniz.

static void testİslemi(SinirHücresi Shücre,int[][] TestVerisi) {

double dogruOrani = testEt(Shücre, TestVerisi);

Console.WriteLine(String.Format("TestVerisi için doğruluk oranı: {0:00.00}", dogruOrani));

Console.ReadKey();

}

static double testEt(SinirHücresi sHücre,int[][] veriSetleri) {

double dogruSayisi = 0;

foreach (int[] veriseti in veriSetleri)

{

double x1 = veriseti[0] \* 0.1;

double x2 = veriseti[1] \* 0.1;

int hedef = veriseti[2];

if (sHücre.Cikti(x1, x2) == hedef)

{

dogruSayisi++;

}

}

return (dogruSayisi / veriSetleri.Length) \*100;//Accurcy degerini döndürür

}

public int Cikti(double x1, double x2)

{

double toplam = Agirlikx1 \* x1 + Agirlikx2 \* x2;

if(toplam > 0.5)

{

return 1;

}else

{

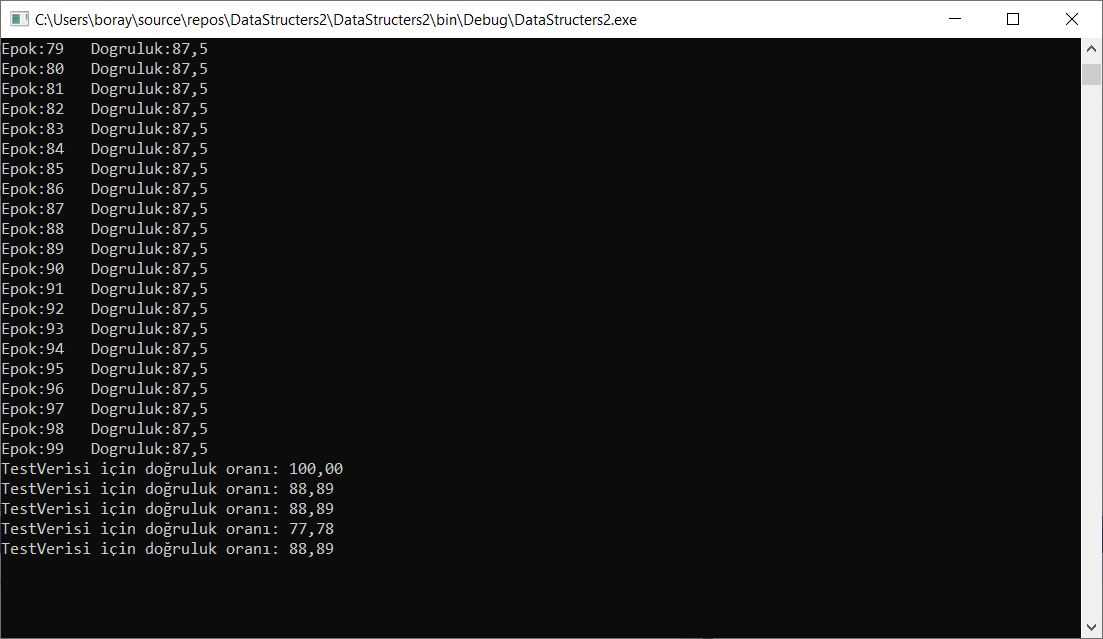
return -1;

}

}

### }2.c.2 Sonuçlar/Ekran görüntüleri

//Test sonucunda elde ettiğiniz başarı değeri sonucunu gösteren konsol çıktısı ekran görüntüsünü buraya ekleyiniz.



TESTVERISI SIRAYLA; TESTVERISI1, TESTVERISI2, TESTVERISI3, TESTVERISI4 , TESTVERISI5 KONTROLU GERCEKLESTIRILMISTIR.

### 2.c.3 Açıklama

Programımızın başarı değerini arttırabilmek için epok sayısını arttırmamız gerekmektedir. Bu düşüncemizin kanıtı olarak 10 epok ve 100 epok boyunca eğittiğimiz nöron örnek verilebilir.

Örnekler dışında yaptığımız deneylerde epok sayısını 300, 700 ve 1000 değerlerinde test ettikten sonra farkına vardık ki 140-200 epok arasında programımızın %100 doğruluk değerine eriştiğini gördük.

Ayrıca ulaşılan doğruluk değeri matematiksel olarak bir grafiğe vurulunca 2 boyutlu x ve y düzleminde bir doğrusal grafik oluşturduğunu saptadık. Girdilerin ağırlıklarını değiştirilmesi bu grafiğin düz bir hatta doğrusal bir şekilde oluşmasını sağlamaktaydı. Oluşan grafik y = -x doğrusuna yaklaşmaktaydı ve bu doğrunun sağında kalan kısmına 1 solunda kalan kısmına -1 tahmin etmesini programımızda oluşturduk.

# Öz değerlendirme Tablosu

|  |  |  |  |
| --- | --- | --- | --- |
| **Proje 1 Maddeleri** | **Not** | **Tahmini Not** | **Açıklama** |
| 1.a | 10 | 10 | Yapıldı |
| 1.b | 10 | 10 | Yapıldı |
| 1.c | 15 | 15 | Yapıldı |
| 2.a | 15 | 15 | Yapıldı |
| 2.b | 10 | 10 | Yapıldı |
| 2.c | 10 | 10 | Yapıldı |
| Rapor | 20 | 20 | İstenen bölümlerle ilgili ekran görüntüleri,kaynak kodu ve algoritmanın kısaca açıklaması yapıldı. |
| Öz değerlendirme Tablosu | 10 | 10 | Yapıldı |

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