YAPAY SİNİR AĞLARI

ÖDEV-3 RAPORU

Prof. Dr.Neslihan Serap Şengör

Osman Kaan Kurtça

040160090

Hasan Göktuğ Kayan

040160072

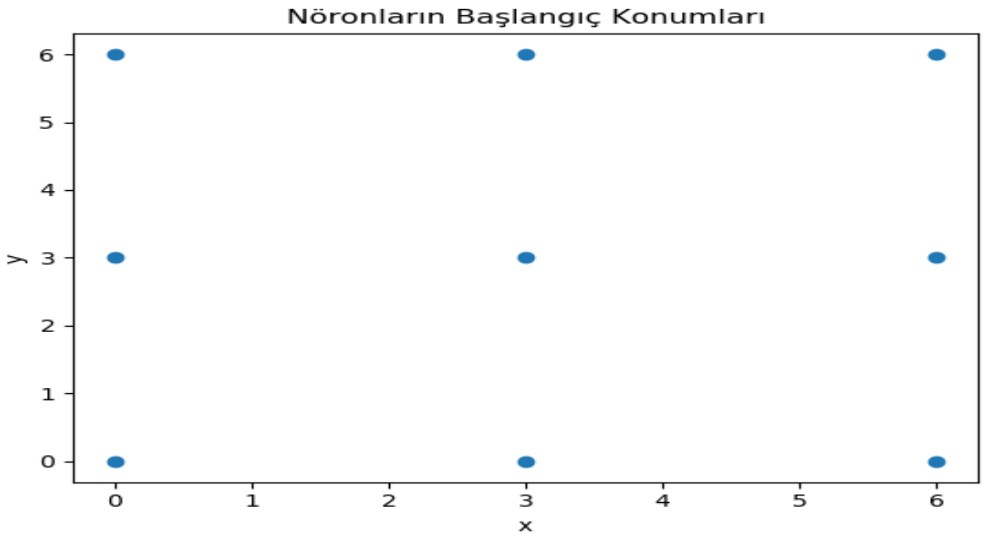
SORU-1)

Kohonen'in öz düzenlemeli ağı eğiticisiz öğrenmenin bir biçimi olan yarışmalı öğrenme temeline dayanan bir ağ yapısıdır. Bu ağ yapısında nöronlar konum ve ağırlık olmak üzere iki parametreyle temsil edilmektedir. Konum iki boyutlu bir düzlem üzerinde olabileceği gibi bir boyutlu bir doğru üzerinde de olabilir. Öbekleme problemlerinde bu ağı kullanmaktaki amaç: her bir öbeği en iyi temsil eden nöronun belirlenmesidir. Bunu yaparken ilk önce ağa verilen her bir örüntü için bu örüntüye en çok benzeyen ağırlık değeri iç çarpım veya örüntü ve ağırlık arasındaki mesafeye bakılarak bulunur, daha sonra öğrenme kuralına göre en çok benzeyen ağırlık en fazla olmak üzere uzaklığa göre azalan bir biçimde bu ağırlığın düzlemdeki veya doğrudaki komşularının ağırlıkları güncellenir. Bu işlem ağ istenilen hata oranına yakınsayıncaya kadar tekrarlanır ve her öbeği en iyi temsil eden nöronlar bulunur.

Bizim modelimizde eğitim iki aşamadan oluşmakta:

1- Öz-düzenleme: Kazanan nöronun bütün komşularının ağırlığı değiştiriliyor
2- Yakınsama: Kazanan nöronun en yakın komşularının ağırlığı değiştiriliyor.

Her biri 3 boyutlu 200 noktadan oluşan 3 öbeğimizi oluştururken Gauss dağılımından yararlandık ve dağılımın parametrelerini değiştirerek ağımızın performansını test ettik.



Şekil1.Nöronların Başlangıç Konumları

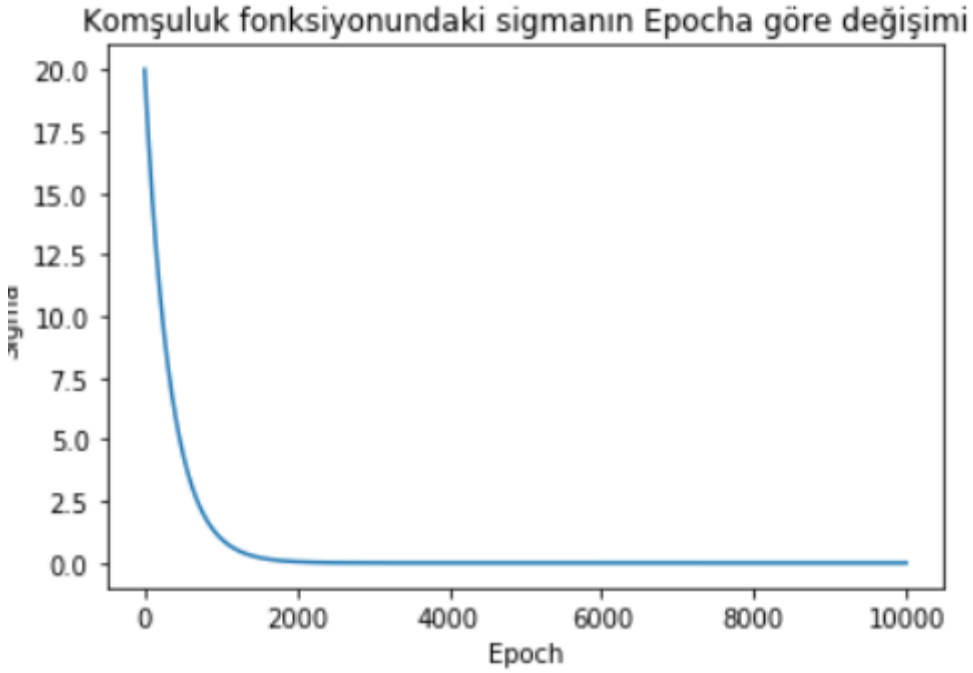
Şekil1.'de görüldüğü gibi nöronlarımızı(farklı nöron sayılarıyla denedik) iki boyutlu düzlemde kare üzerine yerleştirdik.

$$h_{j,i\,(\mathbf{x})}(n) = \exp\left(-\frac{d_{j,i}^2}{2\sigma^2(n)}\right), \qquad n = 0, 1, 2, \ldots,$$

Yukarıda verilen komşuluk fonksiyonundaki sigma(n) fonksiyonunu seçerken S.Haykin "Neural Networks: A Comprehensive Foundation" kitabının sf.474-475 referans aldık.

$$\sigma(n) = \sigma_0 \exp\left(-\frac{n}{\tau_1}\right) \qquad n = 0, 1, 2, \ldots,$$

Fonksiyonunda sigma_0 değeri için kitapta nöronların yerleştiği kafesin yarıçapı önerilmiş, biz bu değer de içinde olmak üzere birçok sigma_0 değeri için ağımızın performansını denedik. Eğitim iterasyon sayımızı 10000, sigma_0 değerini 20, T1 zaman sabitini 1000 olarak seçtiğimizde sigma(n) fonksiyonun iterasyon'a göre değişimi aşağıdaki gibidir:

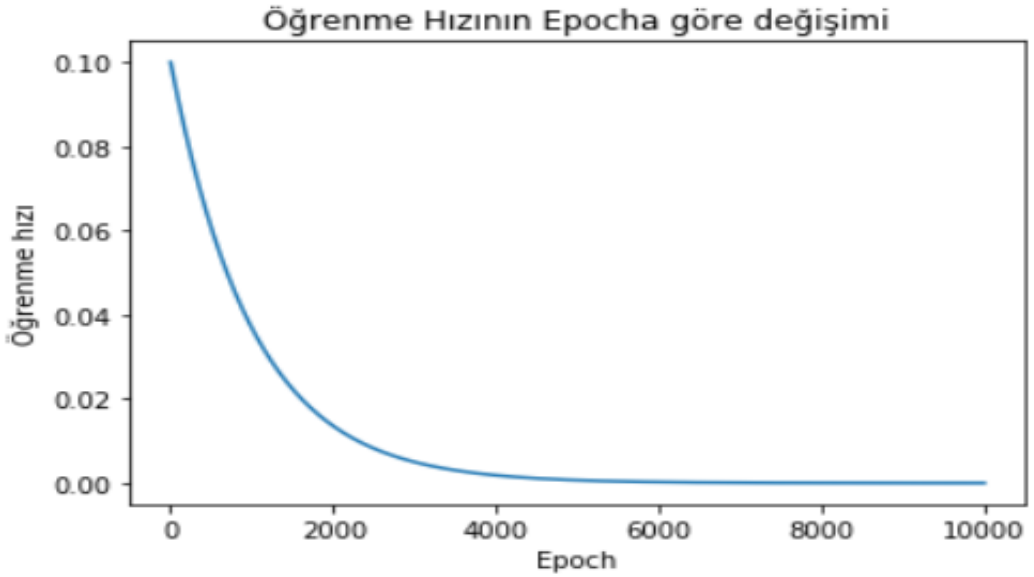


Şekil2. Sigma(n)'nin iterasyon'a göre değişimi

Benzer şekilde öğrenme hızı fonksiyonu için n0 ve T2 değerlerinin belirlenmesinde kitabı referans alarak n0 başlangıç değerini 0.1 ve T2 zaman sabiti değerini 1000 aldık . Bu parametrelerin farklı değerleri için testler yaptık.

$$\eta(n) = \eta_0 \exp\left(-\frac{n}{\tau_2}\right), \qquad n = 0, 1, 2, \ldots,$$

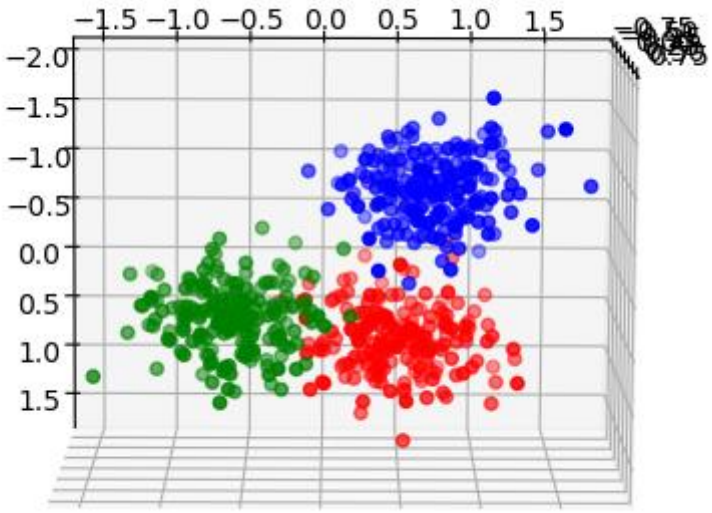
Bu fonksiyonun da n iterasyon sayısını temsil etmek üzere değişimi aşağıdaki gibidir:



Şekil3.Öğrenme hızı fonksiyonunun Epocha göre değişimi

Test kümemizi her bir sınıftan rastgele seçilen 40 nokta olmak üzere toplam 120 nokta olarak seçtik. Geri kalan 480 nokta ile de ağımızı eğittik.

a) Farklı sigma_0 değerleri için ağın performansının test edilmesi



Şekil4. Farklı sigma_0 değerlerini test ettiğimiz veri kümesi

Sırasıyla kırmızı,mavi ve yeşil renklerle gösterilen kümeleri 0,1 ve 2 olmak üzere etiketledik ve test kümemizi de ilk 40 nokta 0 sınıfına ikinci 40 nokta 1 sınıfına son 40 nokta 2 sınıfına ait olacak şekilde düzenledik. Nöron sayımız 9, eğitim için iterasyon sayımız 10000 ve öğrenme hızı fonksiyonu için başlangıç değeri 0.1 ve T2 zaman sabiti 1000 olarak seçtik.

Sigma_0=20 için:

Eğitimden sonra nöronların son ağırlıkları:

[0.0048242 0.07662069 -0.10136659]
[-0.71203992 0.62709095 0.02978045]
[-0.8611515 0.7208962 0.0908459]
[0.36572832 -0.46596547 -0.19817499]
[0.05645243 0.20250485 -0.14402475]
[-0.17180268 0.59863994 -0.13686664]
[0.35832583 -0.62543306 -0.11036098]
[0.49368629 -0.13101419 -0.25601226]
[0.71250296 0.45396024 -0.30023237]

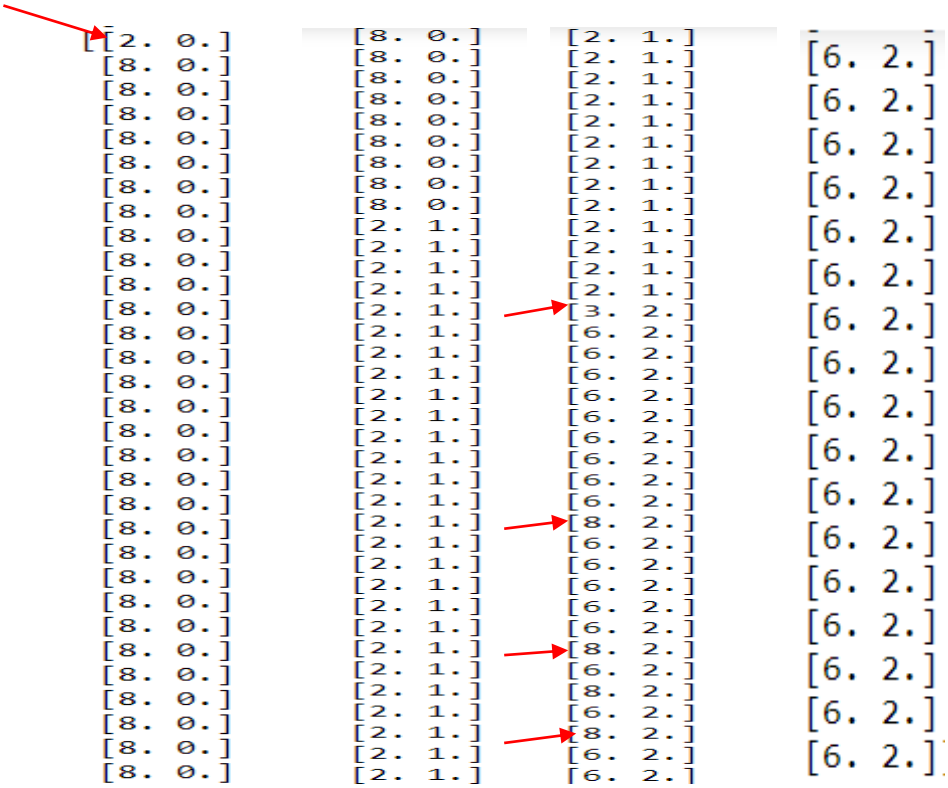
Nöronların ağırlıklarının ortalama değerleri:

-0.006640567235386266
-0.018389506407218772
-0.01646979879740358
-0.09947071371759197
0.03831084167958196
0.09665687523658577
-0.1258227354717648
0.03555327898056413
0.28874361172183943

Şekil4.'teki verilerimiz için ortalama değerleri:

0 ile etiketlenen sınıf için 0.2859170888697425
1 ile etiketlenen sınıf için -0.015287018410418528
2 ile etiketlenen sınıf için -0.12323275630403174
Not: İndislerden dolayı nöron sayısı 0'dan başlıyor ve 8'de bitiyor

Ortalama değerleri karşılaştırdığımızda 2 numaralı sınıfının 8. nöron ile 1 numaralı sınıfın 2. Nöron ile 2 numaralı sınıfın ise 6. nöron ile en iyi şekilde temsil edilebileceği sonucunu çıkarabiliriz. Ağımızın tahmin ettiği nöronları ve test kümesindeki noktaların gerçekte hangi sınıfa ait olduğunu bastırdığımızda:



Kırmızı oklarla gösterilen veriler yukarıda belirlenen nöronlarla temsil edilmemiş dolayısıyla yanlış tahmin edilmiştir.

Diğer hiçbir parametreyi değiştirmeden Sigma_0=5 için yanlış tahmin edilen veri sayısı 9'a çıkıyor . Sigma_0=0.5 için 120 veri içinden 11'i yanlış tahmin ediliyor. Sigma_0=40 için yanlış tahmin edilen veri sayısı 3 olarak çıktı. Buradan çıkaracağımız sonuç sigma_0 düştükçe hata oranımız artar bunun sebebi: komşuluk fonksiyonumuz Gauss dağılımıyla ifade edildiği için kazanan nöron ile herhangi bir nöron arasındaki mesafe değişmeksizin; küçük sigma_0 değerinde komşuluk fonksiyonu daha düşük bir değer alır , bu da o komşuya ait ağırlık değerinin daha az değiştirilmesi demektir. Komşuların ağırlıkları daha az değiştiği için nöronların istenen ağırlık değerlerine ulaşması için daha fazla iterasyon gerekir, bundan dolayı hata oranı artar.

b) Öğrenme hızı fonksiyonunun farklı başlangıç değerleri için ağın performansının test edilmesi

$$\mathbf{w}_j(n+1) = \mathbf{w}_j(n) + \eta(n)h_{j,i(\mathbf{x})}(n)(\mathbf{x} - \mathbf{w}_j(n))$$

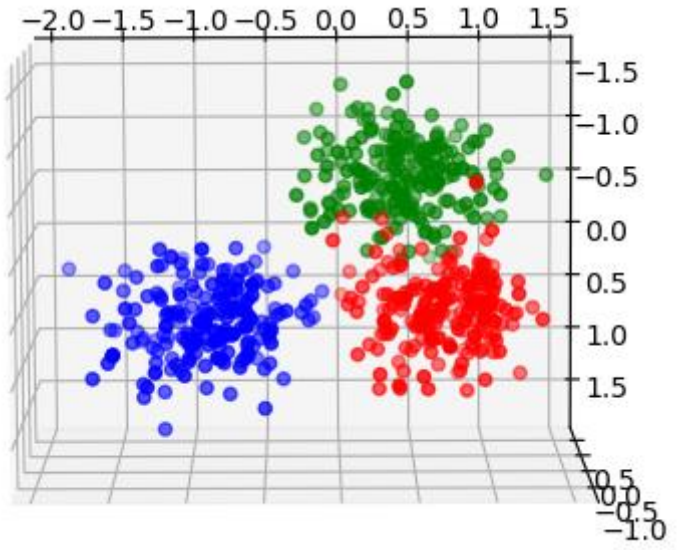
Bu ağ yapısı için öğrenme kuralı yukarıdaki gibidir, n_0'ın büyük değerleri için nöronların ağırlıklarındaki değişim daha fazla olacaktır. Bu durum da ağırlıkların belirlenen iterasyon sayısında öbekleri en iyi temsil edecek nöronların ağırlıklarına yakınsamaması problem ine neden olabilir.

Farklı senaryolarda ağımızın performansını değerlendirmek için hata matrisi kullandık. Hata Matrisi sınıflandırma problemlerinde hangi sınıf kaç kere doğru kaç kere yanlış tahmin edildiğini gösteren matristir. Bu matrisi kullanmak için **scikit-learn** kütüphanesinde tanımlı olan **confusion _matrix** fonksiyonunu çağırdık. Öncelikle ağımızı eğittik ve test kümesiyle test ettik , test sonucunda ağımızın tahmin ettiği nöronlara karşılık düşen sınıflar ile daha önceden verimize verdiğimiz sınıf bilgilerini karşılaştırdık.

```
from sklearn import metrics
confusionMatrix=metrics.confusion_matrix(comparison[:,1], comparison[:,0],labels=[0,1,2])
```

Şekil5. Hata matrisinin hesaplanması

Nöron sayımız 9, sigma_0 değeri 20 ve T1 zaman sabiti 1000, eğitim iterasyon sayısı 1000 ve öğrenme hızı fonksiyonunun zaman sabiti T2=100 ve ayırmaya çalıştığımız veri kümesi Şekil6.'daki gibi olmak üzere;



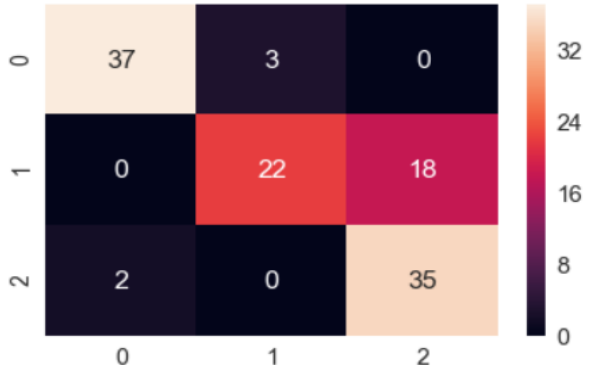
Şekil6.Öbeklenmek istenen veri kümesi

Öğrenme hızımızın başlangıç değeri n_0'ı 0.9 seçtiğimizde:

```
from sklearn import metrics
from scipy import stats
first = stats.mode(karsilastirma[0:40,0])
karsilastirma[:40,0]=np.where(karsilastirma[:40,0]==first[0][0],0,karsilastirma[:40,0])
second = stats.mode(karsilastirma[40:80,0])
karsilastirma[40:80,0]=np.where(karsilastirma[40:80,0]==second[0][0],1,karsilastirma[40:80,0])
third = stats.mode(karsilastirma[80:,0])
karsilastirma[80:,0]=np.where(karsilastirma[80:,0]==third[0][0],2,karsilastirma[80:,0])

confusionMatrix=metrics.confusion_matrix(karsilastirma[:,1], karsilastirma[:,0],labels=[0,1,2])

df = pd.DataFrame(confusionMatrix, range(3), range(3))
fig4 = plt.figure()
sn.set(font_scale=1.4)
sn.heatmap(df, annot=True, annot_kws={"size": 16})
plt.show()
```

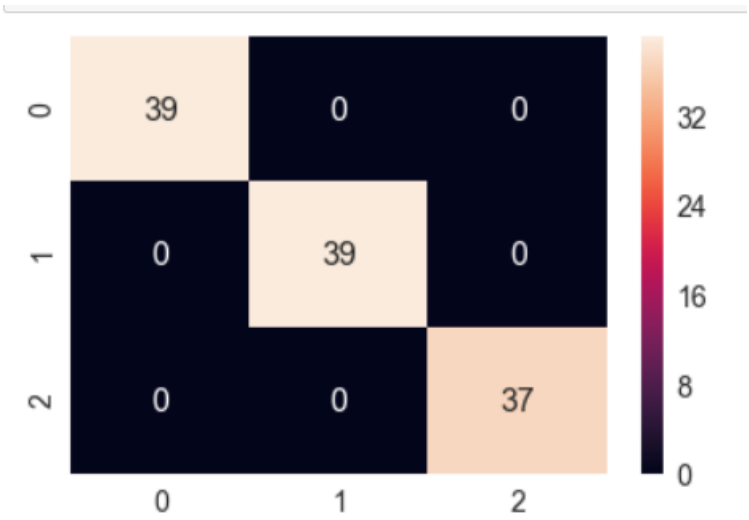


Şekil7. n_0=0.9 için hata matrisi

İlk satırdaki 37 değeri 1. öbekteki verilerin kaç tanesinin doğru olarak tahmin edildiğini gösterir. Aynı şekilde 2. Satırdaki 22 değeri 2. öbekteki kaç verinin doğru olarak tahmin edildiğini ifade eder 18 değeri de ağımızın kaç tane veriyi 2.öbek yerine 3. öbekte olarak tahmin ettiğini gösterir , son olarak da 3. öbek için 35 veri doğru tahmin edilmiştir ve 2 verinin de

sınıfını 1.öbek olarak yanlış tahmin etmiştir.

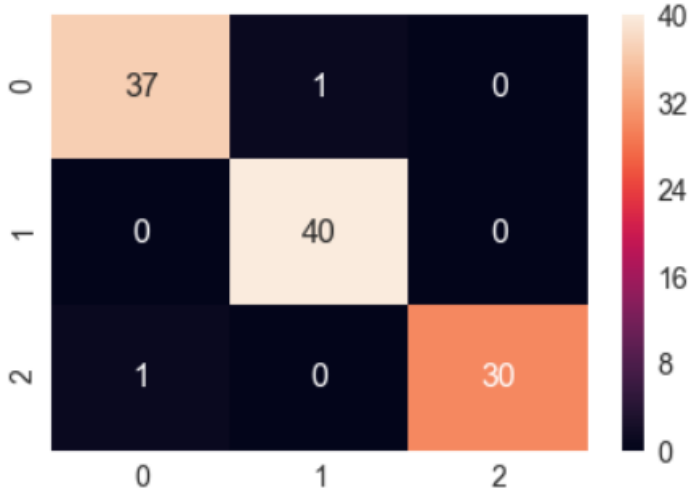
Öğrenme hızımızın başlangıç değeri n_0'ı 0.1 seçtiğimizde:



Şekil8.n_0=0.1 için hata matrisi

1. Sınıftaki verilerden 39'u 2. sınıftaki verilerin de 39'u ve 3. sınıftaki verilerden 37'si doğru tahmin edildi. Öğrenme hızı fonksiyonunun başlangıç değerini 0.9 olarak seçtiğimiz duruma göre performans farkı açık bir şekilde görülüyor. Ağımızın büyük n_0 değeri için yakınsama problemine dair hipotezimiz doğrulanmıştır.

Öğrenme hızımızın başlangıç değeri n_0'ı 0.5 seçtiğimizde:



Şekil9.n_0=0.5 için hata matrisi

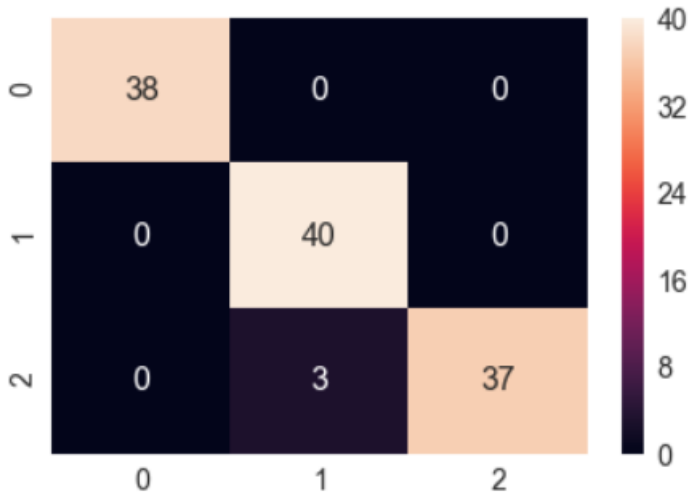
Sınıftan 37'si 2. Sınıftan verilerin hepsi 3. Sınıftan ise yalnız 30 tanesi doğru tahmin edilmiştir.

Not: öbeklerimizden birisi 2 farklı nörona yakınsıyorsa; hata matrisi sadece en çok verinin yakınsadığını kabul ediyor bu yüzden matriste performans düşük görünüyor.

c) Nöron sayısına göre ağın performansının test edilmesi

Şekil6.'daki veri kümeleri değişmeden , eğitim iterasyon sayısı 1000, komşuluk fonksiyonunun başlangıç değeri 20 ve zaman sabiti 1000 , öğrenme hızı fonksiyonunun başlangıç değeri 0.1 ve zaman sabiti 1000 olmak üzere

Nöron sayısı=9 için;



Şekil10.Hata matrisi

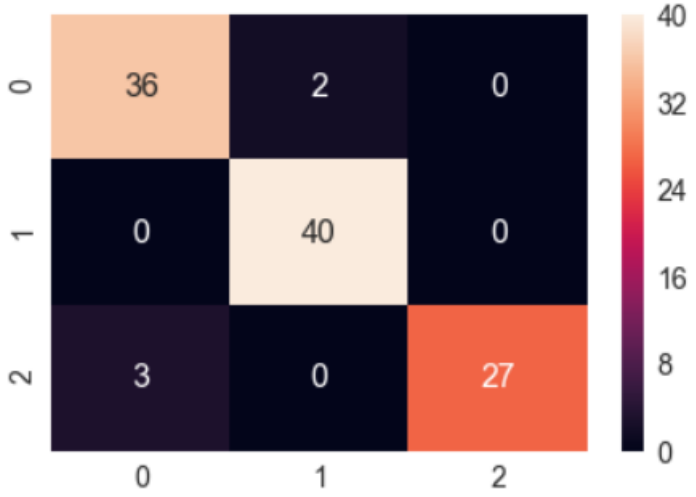
İlk sınıftaki verilerden 38'i ikinci sınıftaki verilerden 40'ı son sınıftaki verilerden 37'si doğru tahmin edilmiştir, 3 veri de ikinci sınıftaki verilerle karıştırılmıştır.

Nöron sayısı=16 için(nöronları karenin üzerine yerleştirdiğimiz için tam kare değerler aldık);

```
[[ 0.  0.] [ 0.  0.]  [ 7.  1.]  [13.  2.]
 [ 0.  0.] [ 0.  0.]  [ 7.  1.]
 [ 0.  0.] [ 0.  0.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 0.  0.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 0.  0.]  [ 7.  1.]  [12.  2.]
 [ 1.  0.] [ 0.  0.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]  [12.  2.]
[13.  0.] [ 7.  1.]  [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  1.]  [13.  2.]
 [ 0.  0.] [ 7.  1.]  [13.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]  [13.  2.]
 [ 0.  0.] [ 7.  1.]  [13.  2.]  [12.  2.]
[12.  0.] [ 7.  1.]  [12.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 0.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [13.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]  [12.  2.]
 [ 1.  0.] [ 7.  1.]  [ 0.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [13.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 0.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [ 7.  2.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [12.  2.]
 [ 0.  0.] [ 7.  1.]  [13.  2.]  [12.  2.]]
 [ 0.  0.] [ 7.  1.]  [12.  2.]
```

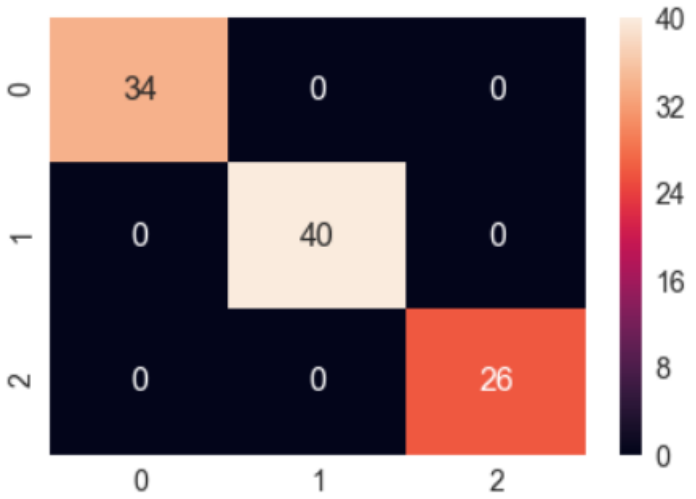
Şekil11. Sınıflar için ağımızın tahmin ettiği nöronlar (0,1 ve 2 sınıfları belirtir.)

Nöron sayısı arttığı için ağımız aynı sınıf için birden fazla nöronu seçmiştir , yukarıda da belirtildiği gibi hata matrisinde en çok seçilen nöron doğru olarak kabul edildiği için hata oranımız artmıştır.



Şekil12. 16 nöronlu ağ için hata matrisi

Nöron sayısı=25 için;



Şekil13.25 nöronlu ağ için hata matrisi

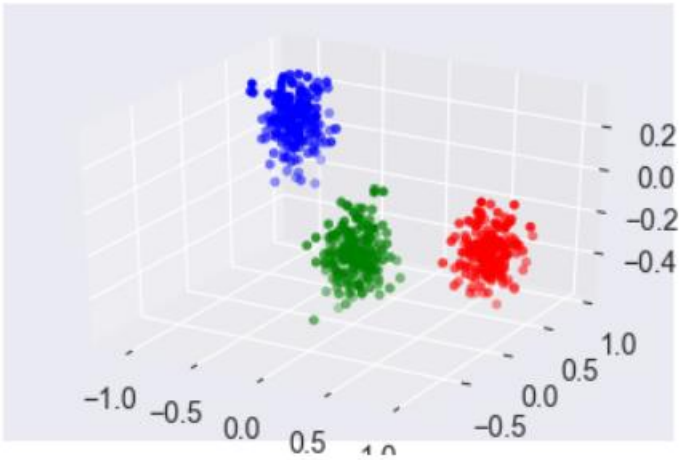
Yüksek sayıdaki nöronda hatanın daha yüksek çıkmasının
sebebi olarak ağımız çok fazla nöron içerisinden hangisinin örüntüleri daha iyi temsil ettiğini ayıramamasıdır. Ağımız bir küme için birden fazla nörona yakınsamaktadır.

d) Veri kümelerinin dağılımına göre ağın performansının test edilmesi

Veriler Gauss dağılımıyla oluşturulduğu için standart sapma ve ortalama değerlerini değiştirerek veri kümelerinin birbirine göre konumunu ayarlayabiliriz.
Bu test aşamasında nöron sayısı 9, eğitim iterasyon sayısı 1000, komşuluk fonksiyonunun başlangıç değeri 20 ve zaman sabiti 1000 , öğrenme hızı fonksiyonunun başlangıç değeri 0.1 ve zaman sabiti 1000 olarak alınmıştır.

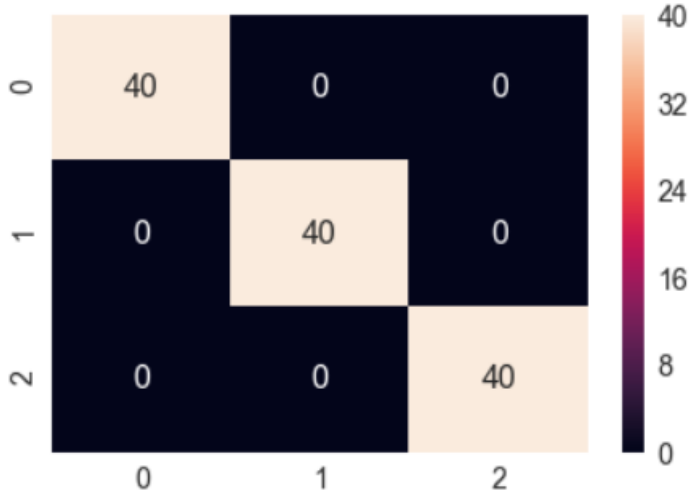
```
a=np.random.normal(0.7,0.1,[200,3]) - np.array([0,0.2,1])
b=np.random.normal(0.1,0.1,[200,3]) - np.array([1,-0.6,0])
c=np.random.normal(0.4,0.1,[200,3]) - np.array([0,1,0.5])
```

Verilerimizi yukarıdaki gibi merkezleri birbirinden uzak ve standart sapmaları küçük Seçtiğimizde birbirinden ayrık 3 öbek elde ediyoruz:



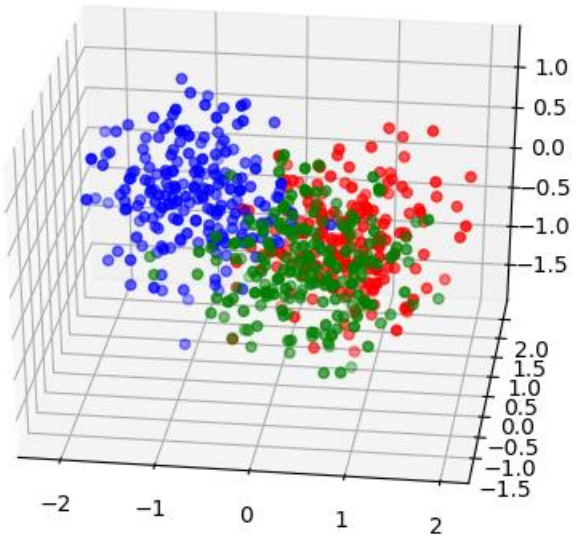
Şekil14. Standart sapma=0.1 için öbekler

Sezgisel olarak Şekil14.'e bakınca tüm öbeklerin rahatça birbirinden ayrılmasını bekliyoruz, hata matrisimiz de bunu doğrular nitelikte çıktı.



Şekil15.Hata matrisi

Öbeklerin merkezlerini değiştirmeden standart sapmaları 0.5 olarak seçersek Şekil16.'daki veri kümesini elde ediyoruz:



Şekil16. Standart sapma=0.5 için veri kümesi

Şekil16.'ya ait hata matrisi ise aşağıdaki gibidir:

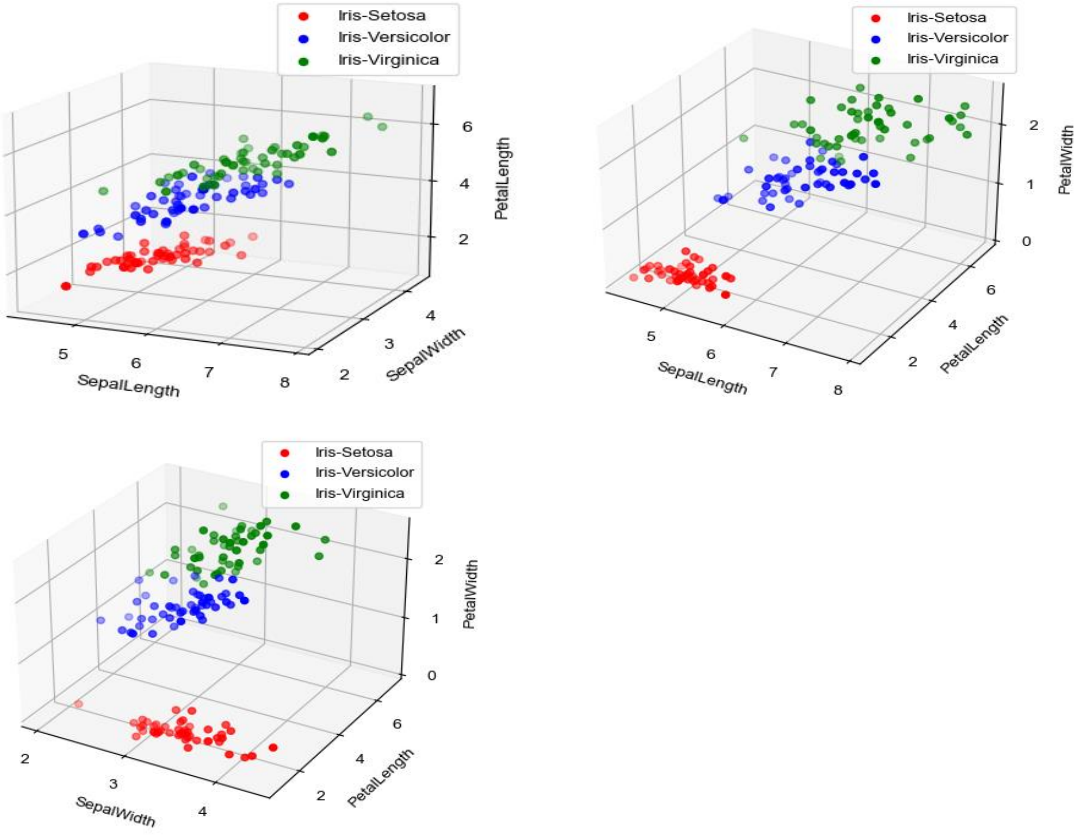
Şekil17. Standart sapma=0.5 için hata matrisi

Beklendiği gibi öbeklerimiz ne kadar iç içe ise hata oranımız o kadar yüksek çıkmaktadır.

SORU-2) İris Verisetinin öbeklenmesi

İlk soruda ağımızın iyi çalışıp çalışmadığını çeşitli parametreler ile test ettik ve kendi oluşturduğumuz verisetleri için iyi sonuçlar verdiğini, verileri iyi bir şekilde öbeklediğini gözlemledik. Şimdi yapmamız gereken bu ağ ile iris verisetindeki 3 farklı sınıfı öbeklemek. İrisi veriseti hakkında 2. Ödevden dolayı bazı önbilgilerimiz mevcut. 2. ve 3. Sınıfın bazı özellikleri birbirine çok benzediğini biliyoruz.

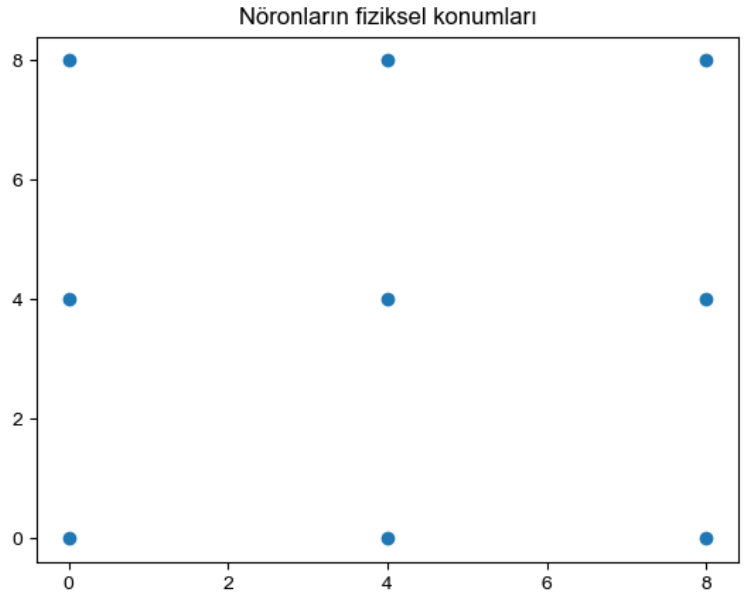
Öncelikle, 4 boyutlu İris verisetini farklı 3 boyutlarına göre çizdirerek gözlemledik.



Şekil17. İris veriseti 3 boyutlu gösterimleri

Grafiklere bakacak olursak, Versicolor ve Virginica türünün “SepalLength” ve “SepalWidth” uzunluğunu birbirine çok benzediğini ve bundan dolayı iki türün birbirine karışabileceğini öngörüyoruz.

Daha sonra, her türden 40’ar tane olacak şekilde 120 verilik eğitim kümemizi oluşturduk ve 30 verimizi de test için ayırdık ve verisetimizi eğitim ve test için hazır hale getirdik.



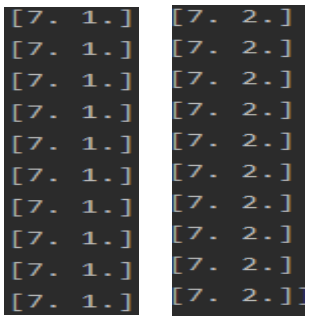
Şekil18. Nöronların Fiziksel Konumları

Bu soruda da verilerimizi 3 öbeğe ayırmak istediğimiz için nöron sayımızı 9 olarak seçtik ve iki boyutlu düzleme fiziksel olarak yerleştirdik.

Iris verisetini öbeklerken eğitimimizin ilk aşamasında (özdüzenleme) 1000 iterasyon yaptık ve öğrenme hızını 0.1'den başlattık. Eğitimin ikinci aşamasında ise (yakınsama) 500*m (m=nöron sayısı) kadar iterasyon yaptık ve öğrenme hızını 0.01'den başlattık. Sigma başlangıç değerimizi ise nöronlarımızın yerleştiği iki boyutlu düzlemin çapı olarak aldık. (5,65). Burada da yine S.Haykin "Neural Networks: A Comprehensive Foundation" kitabının sf.474-475 referans aldık.

Sonuçların Karşılaştırılması

Veriyi olduğu şekilde ağa verdiğimizde 2. ve 3. Sınıfın her seferinde birbirine karıştığını gözlemledik.



Şekil.19 – 1 ve 2 ler gerçek sınıfları temsil ederken 7 ağın o sınıfa en uygun bulduğu nöronu temsil etmekte.

Veriyi standardize ettik ve verilerin dağılımını bozmadan değerleri yaklaşık -1 ile 1 arasında dağılacak hale getirdik.

```
scaler = StandardScaler()
dataset = scaler.fit_transform(dataset)
```

Şekil.20 StandartScaler

Verileri ağımıza bu şekilde sunduğumuzda daha iyi sonuç aldığımızı gözlemledik.

```
[[6. 0.]  [0. 1.]  [5. 2.]
 [6. 0.]  [0. 1.]  [5. 2.]
 [6. 0.]  [5. 1.]  [5. 2.]
 [6. 0.]  [2. 1.]  [5. 2.]
 [6. 0.]  [0. 1.]  [2. 2.]
 [6. 0.]  [2. 1.]  [5. 2.]
 [6. 0.]  [0. 1.]  [5. 2.]
 [6. 0.]  [0. 1.]  [5. 2.]
 [6. 0.]  [0. 1.]  [2. 2.]
 [6. 0.]  [0. 1.]  [5. 2.]
```

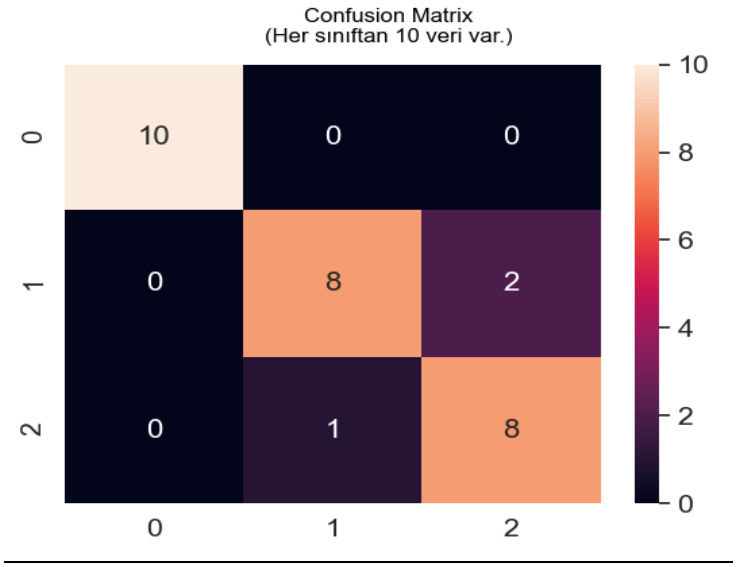
Şekil.21 Tahmin edilen öbekler ve gerçek sınıflar

Burada da yine bazı verilerde 2. Sınıfla 3. Sınıfı karıştırdığını gözlemliyoruz fakat yine de ilk duruma göre çok daha iyi bir sonuç aldığımızı görüyoruz.

```
Nöronların ilk ağırlıkları:  [[ 0.4018516   0.14234457 -0.36352763 -0.23992504]
 [ 0.10032387  0.36470448  0.34520459  0.20502034]
 [ 0.05146897 -0.25463912  0.24990525 -0.37260317]
 [-0.12293163 -0.26592718 -0.34623988  0.49983385]
 [ 0.05697176 -0.17656544 -0.09136709  0.12002728]
 [ 0.20663538  0.00961075 -0.01831214 -0.09753596]
 [ 0.19603282  0.48220385 -0.08297178  0.14920309]
 [ 0.0173671  -0.47448488 -0.43495996  0.15835223]
 [-0.05776193  0.20038633  0.17705316 -0.13656448]]
```

```
Nöronların son ağırlıkları:  [[ 0.54961821 -0.22533462  0.60095503  0.55288353]
 [ 0.49625001 -0.49940185  0.75825251  0.71015002]
 [ 1.00701299  0.03092083  0.89957823  0.86059971]
 [-0.90886921  0.80116507 -1.14670345 -1.10359782]
 [ 0.42772581 -0.22396581  0.48859779  0.44533999]
 [ 0.62984886 -0.24859706  0.70738517  0.64697805]
 [-0.99486898  0.93265471 -1.30117996 -1.2559176 ]
 [-0.8250518   0.27057946 -0.84915338 -0.84728838]
 [-0.39597605 -1.14434504  0.11310306 -0.00295845]]
```

Şekil.22 Eğitim sonrası nöronların ağırlık değişiminin gösterilmesi.



Şekil.23 Iris Veriseti Hata Matrisi

Yukarıdaki sonuçlara benzer olarak hata matrisinde de 1. Sınıfı çok iyi öbeklediğimizi, 2. ve 3. Sınıfta ise bazı verilerin birbirine karıştığını gözlemliyoruz.

Fakat hata matrisinin iris veriseti için kodu her çalıştırdığımızda bu kadar iyi sonuç vermediğini söylememiz gerekir. Hata matrisimizin çalışma mantığı şu şekildedir:

Örneğin ilk sınıf için tahmin edilen öbeklere bakıyor ve en çok tahmin edilen öbeğin numarasını 0 olarak değiştiriyor, aynısını ikinci sınıf için 1 ve üçüncü sınıf için 2 olacak şekilde yapıyor ve böylece gerçek sınıf değerleri olan 0, 1 ve 2 ile kıyaslıyor. Ağımız bir sınıfı birden fazla öbeğe ayırdığında hata matrisimizin gösterdiği sonuç yanıltıcı olabiliyor.

Bu yüzden, performansı test ederken yukarıda da gösterdiğimiz kodumuzda çıktı olarak yazdırdığımız gerçek sınıflar ve öbekler matrisine bakmak daha anlamlı olabilir.

Sonuç olarak, hem çok katmanlı algılayıcıda hem de özdüzenlemeli ağda Iris verisetini sınıflandırdığımızı söyleyebiliriz. Fakat sonuçlarımızı karşılaştıracak olursak; çok katmanlı algılayıcı da daha iyi bir doğruluk oranı aldığımızı gözlemledik.