YAPAY SİNİR AĞLARI

ÖDEV-1 RAPORU

Prof. Dr.Neslihan Serap Şengör

Osman Kaan Kurtça

040160090

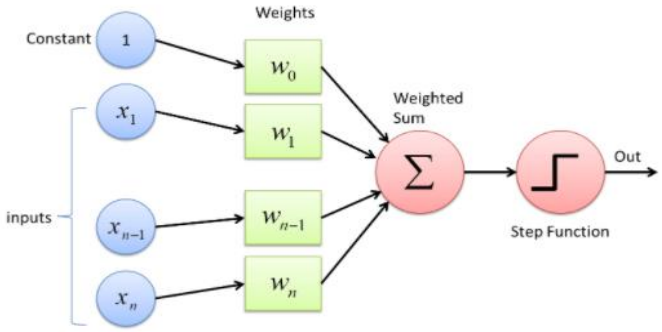
Hasan Göktuğ Kayan

040160072

BURAYA DA OKUL NUMARANIZ GELECEKTİR!

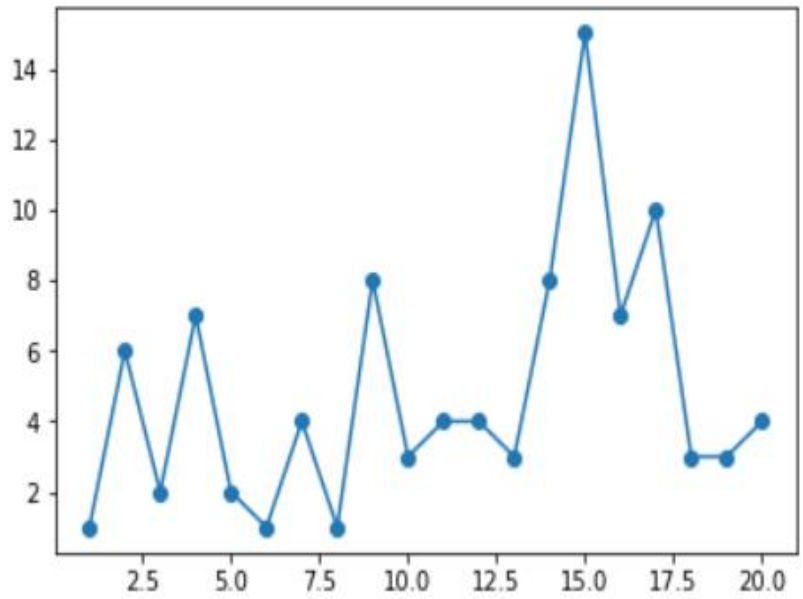
---

1) Genlikte ayrık algılayıcı(Perceptron) ikili sınıflandırmada kullanılan doğrusal, eğiticili bir öğrenme biçimidir. İkili sınıflandırma denmesinin sebebi Perceptron sinir ağı girişindeki verileri yalnızca iki sınıfa ayırabilir. Bunu yaparken girişteki veri kendisine ilişkin ağırlıkla çarpılır, bu işlem veri setindeki tüm veriler için yapıldıktan sonra bir ağırlıklı toplam oluşur bu toplam daha sonra bir aktivasyon fonksiyonundan geçirilerek girişteki verinin hangi sınıfa ait olduğunu belirten bir nevi etiketleme işlemi yapılır.

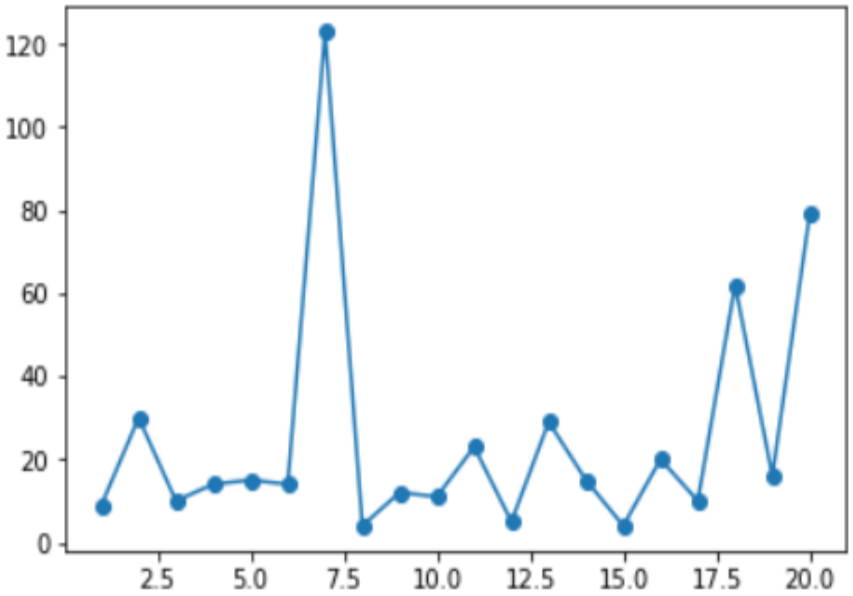


Şekil1.a Perceptron nasıl çalışır?

Ağırlıkların ilk koşullarının eğitim sürecine etkisi eğitilen sinir ağının yakınsaması için gereken iterasyon sayısını değiştirmesiyle görülebilir. İlk ağırlıkların çok küçük seçilmesi durumunda Kaybolan Gradyan Problemi(Vanishing Gradient Descent Problem) ortaya çıkar bu durumda ağırlıklardaki güncelleme miktarı çok az olur ve sinir ağının yakınsaması için çok fazla iterasyon gerekebilir. Tam tersi durumda ağırlıklar çok büyük seçildiğinde Exploding Gradient Problem'i ortaya çıkar optimize edilecek hata fonksiyonunda çok büyük türevler elde edilmesine sebep olur bu durumda model çok dengesiz olur ve öğrenme yeteneğine sahip olmaz. Biz kodumuzda ilk ağırlık koşullarını eğitim fonksiyonumuza bir parametre olarak koyduk , buna göre 3 farklı senaryo oluşturduk, ilk durumda ilk ağırlıklar 0 , ikinci durumda ilk ağırlıklar 1 ve son senaryoda ilk ağırlıklar rastgele olacak şekilde düşündük.
Ağın 20 defa eğitilmesi sonucu ağırlıkların ilk koşullarına bağlı olarak ağın kaçıncı iterasyonda yakınsadığına ilişkin karşılaştırmalı grafik:

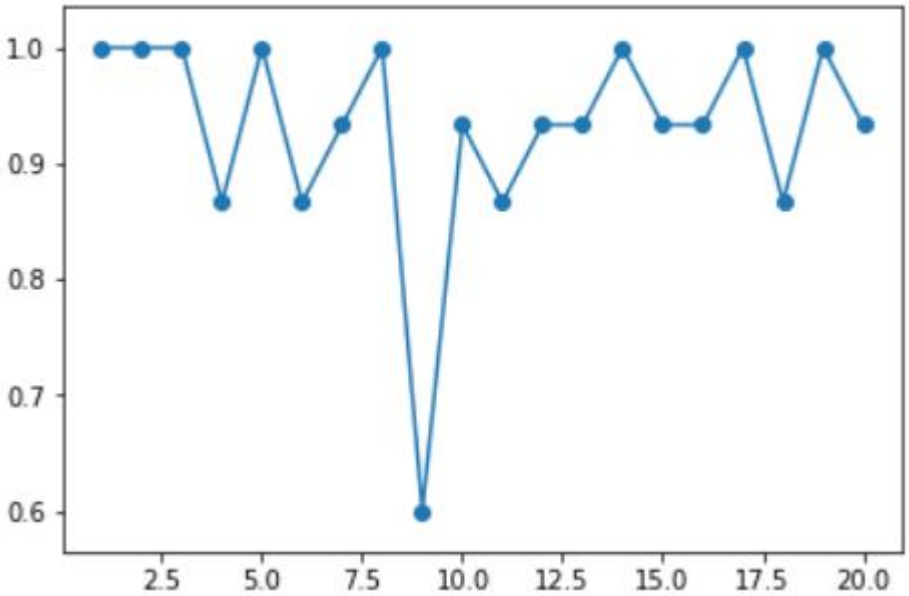


Şekil1.b İlk ağırlıkların 0 olduğu durumda eğitim sayısına göre ağın kaçıncı iterasyonda yakınsadığını gösteren grafik

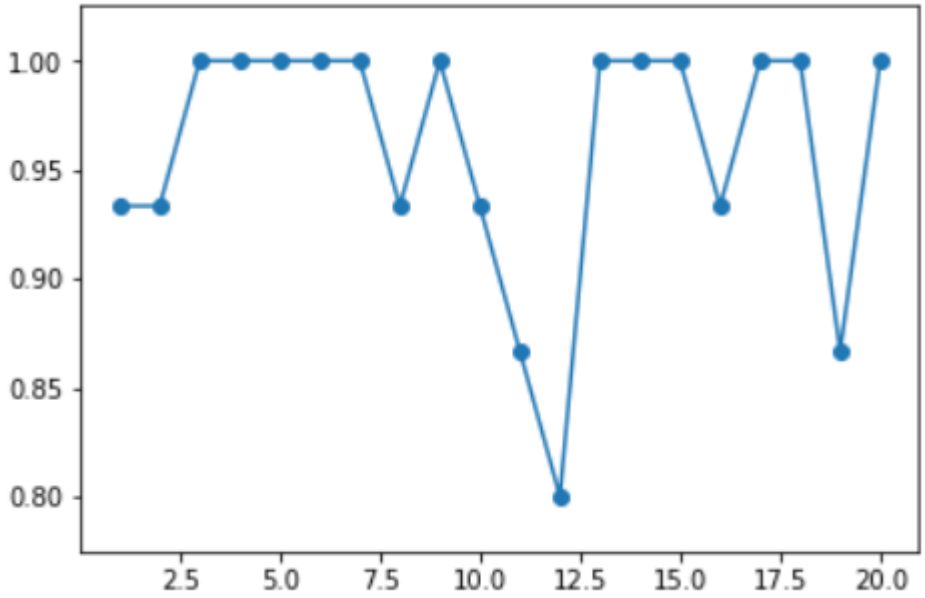


Şekil1.c İlk ağırlıkların 1 olduğu durumda eğitim sayısına göre ağın kaçıncı iterasyonda yakınsadığını gösteren grafik

Şekil1.b ve Şekil1.c'den görülebileceği gibi aynı ağın ilk ağırlıkların 0 seçilmesi durumunda ilk ağırlıkların 1 seçildiği duruma göre aynı eğitim sayısında daha az iterasyonda yakınsadığını görmek mümkündür. Doğruluk açısından pek bir fark göremedik.



Şekil1.d İlk ağırlıkların 0 seçildiği durumda eğitim sayısına bağlı olarak doğruluk (accuracy) ortalama değeri:0.93

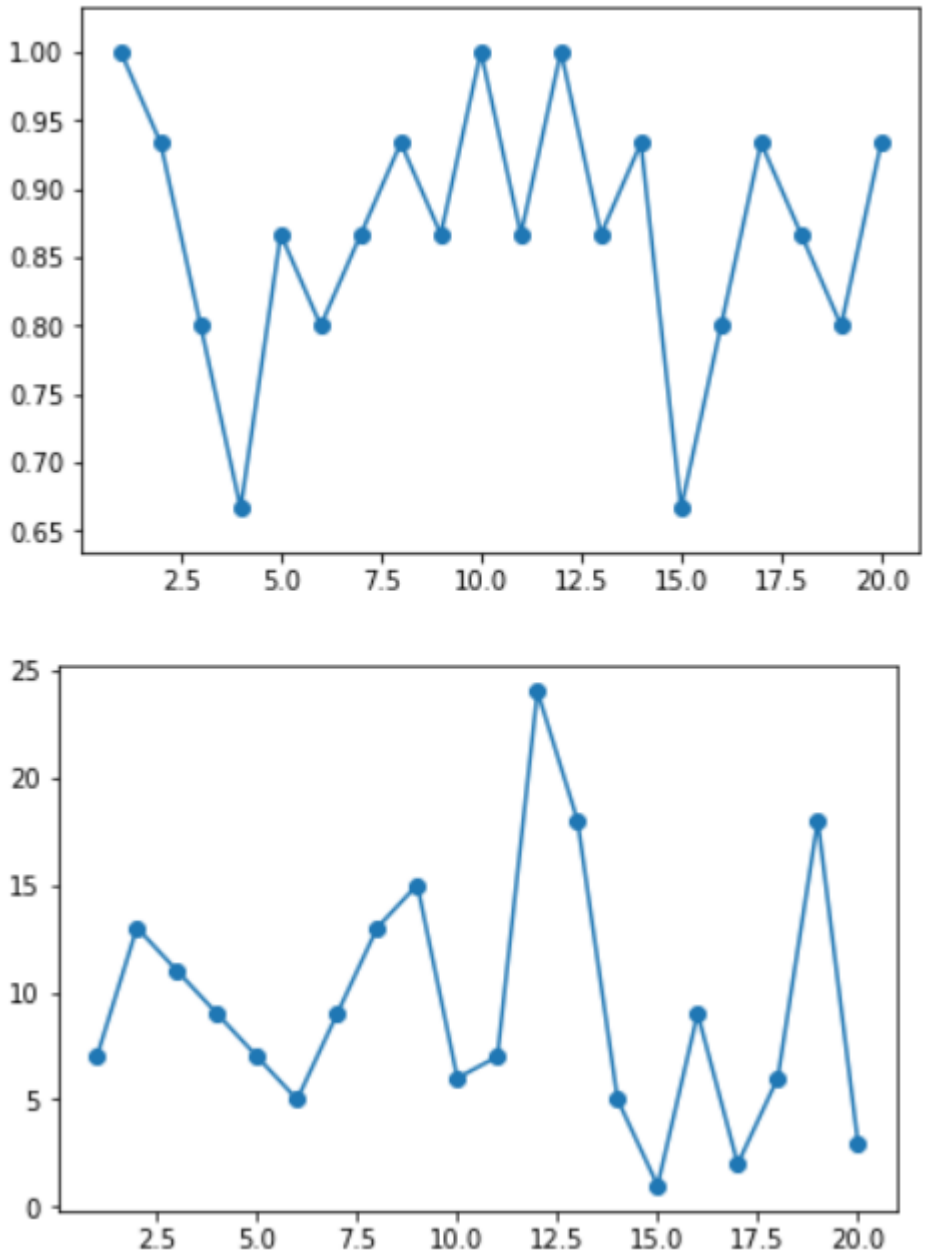


Şekil1.e İlk ağırlıkların 1 seçildiği durumda eğitim sayısına bağlı olarak doğruluk (accuracy) ortalama değeri:0.96

Öğrenme hızının öğrenme sürecine etkisi: öğrenme hızını küçük seçersek iterasyon sayımız artar verimiz daha uzun sürede eğitilir ama küçük adımlarla gittiğimiz için cost'un minimum olduğu ağırlık değerlerine daha iyi yaklaşırız ve test verisinde daha iyi sonuçlar

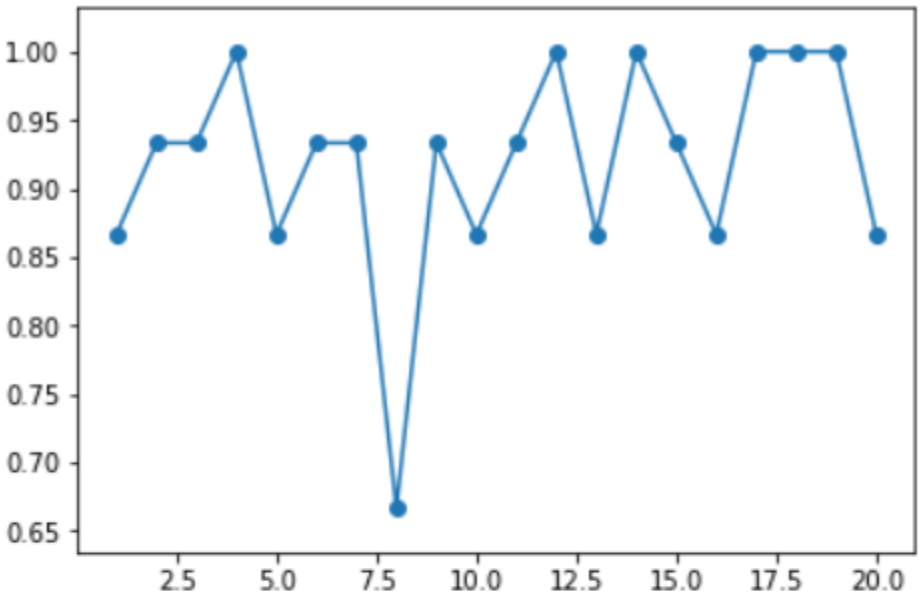
alırız. Öğrenme hızını 1'e yakın seçersek eğitimimiz daha az iterasyonda biter fakat büyük adımlarla gittiğimiz için hatayı minimize eden ağırlık değerinden uzaklaşabiliriz ve bu da test verisindeki sonuçlarımızı etkileyebilir.

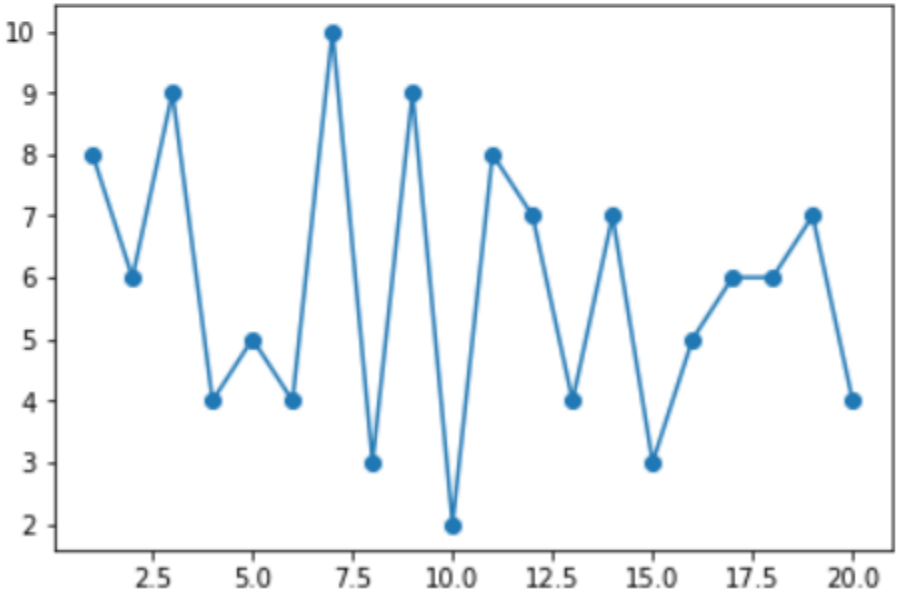
Öğrenme hızını 1 seçtiğimizde ağımızın doğruluk değeri ve yakınsaması için gereken iterasyon sayısı grafikleri:



Şekil1.f Öğrenme hızının 1 seçildiği durumda ağın 20 kere eğitilmesi sonucu ortalama doğruluk Değeri =0.87

Öğrenme hızını 0.01 seçtiğimizde ağımızın doğruluk değeri ve yakınsaması için gereken iterasyo
n sayısı grafikleri :





Şekil1.g Öğrenme hızının 0.01 seçildiği durumda ağın 20 kere eğitilmesi sonucu ortalama
Doğruluk değeri=0.92

Eğitim kümesinin farklı sıralanmasını sağlamak için numpy kütüphanesinden random.choice fonksiyonunu kullanarak tüm veri setinden her eğitimde 25 rastgele indis seçtik. Kodun diğer hiçbir şey değişmeden birçok kez çalıştırılması sonucu doğruluk değeri , ağın yakınsaması için gereken iterasyon sayısı gibi değerlerin farklı çıkmasının sebebini eğitim kümesinin her seferinde

farklı sınıflandırılmasına bağladık. Çünkü eğitim kümemiz dengesiz dağılmış olursa yani hedef değişkenlerinden birinin sayısı diğerinden fazla olursa, modelimiz test verisini tahmin ederken, eğitim kümesinde sayıca fazla olan değişkeni tahmin etmeye eğimli olur ve test verisindeki skorumuz düşük olur.

1)b) Lineer ayrıştırılamaz diziyi oluştururken lineer ayrıştırılabilir dizinin bias terimleriyle birlikte ilk 4 sütununu aldık , ayrıştırılmaz yapmak için labelları rastgele dağıttık. Lineer ayrıştırılamaz olduğunu görselleştiremediğimiz için aynı koşullarda lineer ayrıştırılabilir dizi ile elde ettiğimiz ortalama doğruluk değerlerini karşılaştırdık.

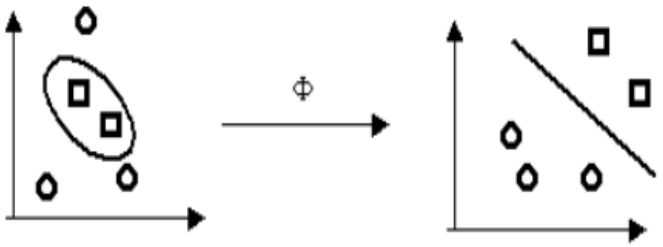
Ayrıştırılabilir dizide, 20 eğitim sonucu test verimizdeki ortalama doğruluk skorumuz: [0.92]
Ayrıştırılamayan dizide, 20 eğitim sonucu test verimizdeki ortalama doğruluk skorumuz: [0.42]

```
In [77]: lineerAyristirilamazDizi

Out[77]: array([[ 2.,  0.,  2.,  2.,  1., -1.],
                [ 0.,  1.,  2.,  4.,  1., -1.],
                [ 0.,  4.,  1.,  3.,  1., -1.],
                [ 2.,  2.,  2.,  2.,  1., -1.],
                [ 0.,  1.,  1.,  4.,  1.,  1.],
                [ 2.,  2.,  1.,  4.,  1., -1.],
                [ 3.,  0.,  3.,  0.,  1.,  1.],
                [ 2.,  0.,  3.,  4.,  1.,  1.],
                [ 0.,  1.,  3.,  1.,  1.,  1.],
                [ 3.,  3.,  0.,  1.,  1.,  1.],
                [ 0.,  4.,  0.,  0.,  1.,  1.],
                [ 0.,  4.,  0.,  2.,  1., -1.],
                [ 2.,  2.,  1.,  3.,  1.,  1.],
                [ 0.,  3.,  0.,  1.,  1.,  1.],
                [ 2.,  2.,  1.,  4.,  1., -1.],
                [ 1.,  0.,  2.,  2.,  1.,  1.],
                [ 4.,  0.,  4.,  4.,  1.,  1.],
                [ 1.,  1.,  3.,  0.,  1.,  1.],
                [ 4.,  4.,  2.,  1.,  1.,  1.],
                [ 2.,  2.,  3.,  4.,  1.,  1.],
                [ 3.,  2.,  4.,  2.,  1.,  1.],
                [ 2.,  4.,  0.,  2.,  1.,  1.],
                [ 1.,  4.,  4.,  2.,  1., -1.],
                [ 2.,  3.,  0.,  3.,  1.,  1.],
                [ 4.,  3.,  4.,  3.,  1.,  1.],
                [ 4.,  2.,  0.,  0.,  1.,  1.],
                [ 3.,  3.,  2.,  2.,  1., -1.],
                [ 4.,  2.,  3.,  0.,  1.,  1.],
                [ 4.,  1.,  0.,  4.,  1., -1.],
                [ 4.,  1.,  1.,  1.,  1.,  1.],
                [ 4.,  0.,  1.,  2.,  1.,  1.],
                [ 4.,  4.,  0.,  4.,  1.,  1.],
                [ 2.,  1.,  2.,  2.,  1.,  1.],
                [ 2.,  0.,  4.,  1.,  1., -1.],
                [ 2.,  1.,  3.,  3.,  1., -1.],
                [ 2.,  4.,  0.,  3.,  1., -1.],
                [ 4.,  0.,  0.,  1.,  1.,  1.],
                [ 2.,  0.,  4.,  4.,  1., -1.],
                [ 3.,  3.,  2.,  2.,  1.,  1.],
                [ 1.,  4.,  1.,  4.,  1.,  1.]])
```

2) N boyutlu lineer ayrıştırılabilir veri kümesi için hiper düzlem: veri setini iki ayrı sete ayıracak ve her set farklı sınıflara ait noktaları içerecek (N-1) boyutlu alt kümedir. Örneğin 2 boyutlu uzayda veri setindeki noktaları iki ayrı sınıfa ayıracak hiper düzlem 1 boyutlu bir doğrudur. Bu noktada Destek Vektör Makineleri(Support Vector Machines) tanımlamak gerekirse SVM kısaca

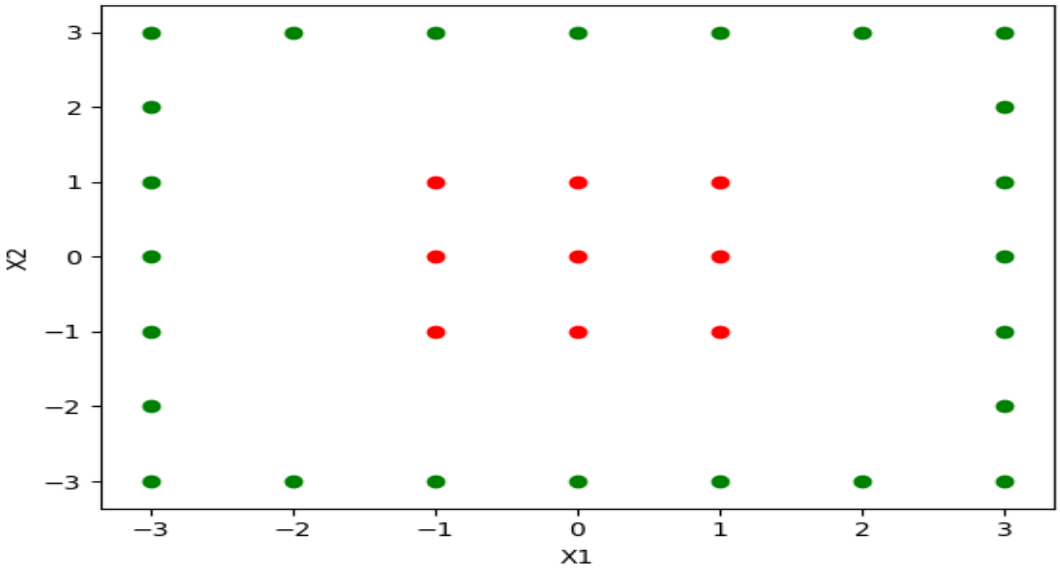
veriyi ayıracak hiper düzlemi öğrenerek sınıflandırma yapan bir lineer ikili sınıflandırıcıdır. Bize verilen sorudaki gibi veri seti lineer bir sınırla ayrıştırılamadığında aşağıdaki şekilde gösterildiği gibi veri setine bir Φ dönüşüm fonksiyonu uygulayıp veriyi lineer ayrıştırılabilir olacağı daha üst boyutlara taşıyabiliriz.



Şekil 2.a

Destek Vektör Makineleri bu bahsedilen dönüşümü Kernel olarak isimlendiren fonksiyonlar ile yapar. Bu kernel fonksiyonu ile Şekil2.a'da gösterilen Φ dönüşüm fonksiyonu arasındaki ilişki aşağıdaki denklem ile verilmiştir. Eşitliğin sol tarafındaki ifade xi ve xj verileri arasındaki benzerlik ilişkisini gösteren bir sabittir.

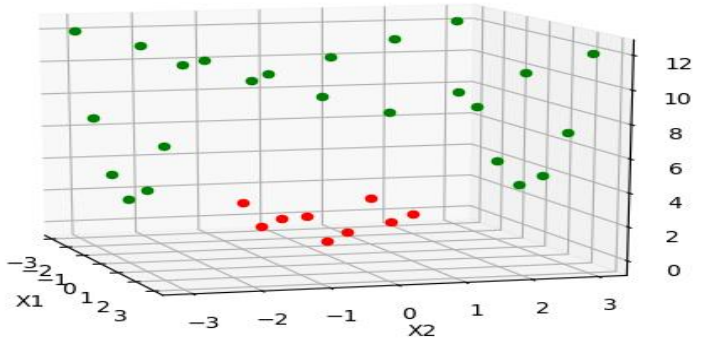
$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j)$$



Şekil 2.b

Şekil 2.b'de soruda verilen noktaları çizdirdiğimizde bu veri setinin 2 boyutlu düzlemde lineer ayrıştıralabilir olmadığını görebiliyoruz. Ayrıştırılabilir olması için noktaları 3 boyuta taşıdık. Bunu yaparken ilk iki birimi sabit tuttuk ve nasıl lineer ayrıştırılabilir yaparız diye düşündük. İlk birim x1 ikinci birim x2 olmak üzere kernel olarak x1**2+|x2| kullandık. Bunu böyle kullanmamızın sebebi x2'nin aldığı -2,-3 gibi negatif değerlerden kurtularak şekil 2.b'de yeşil

noktalar ile gösterilen veri setini yukarı yönlü kaydırmaktı.



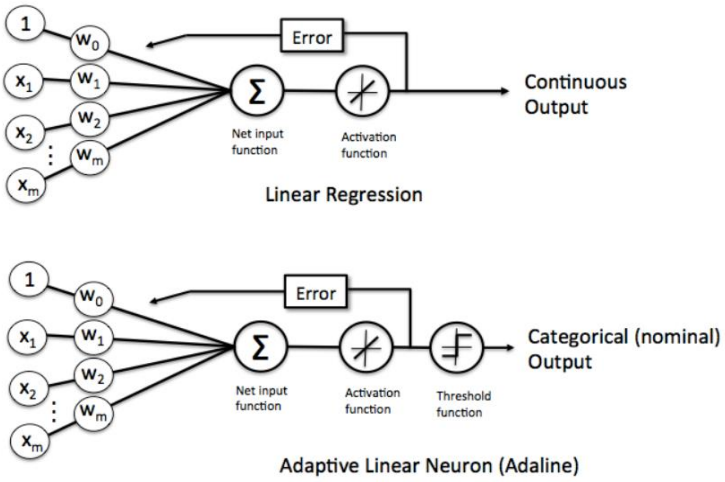
Şekil2.c

Şekil 2.c'de görülebileceği gibi böyle yaptığımızda veri seti açık bir şekilde lineer bir hiper düzlemle ayrılabilir duruma geliyor. Rastgele seçilen eğitim kümeleriyle eğitim 20 defa çalıştırıldığında ortalama doğruluk değerimiz 0.90714286 çıktı.

```
[[1.        ]
 [0.85714286]
 [0.71428571]
 [1.        ]
 [0.85714286]
 [0.85714286]
 [1.        ]
 [0.71428571]
 [0.85714286]
 [1.        ]
 [1.        ]
 [0.85714286]
 [0.85714286]
 [1.        ]
 [0.85714286]
 [0.71428571]
 [1.        ]
 [1.        ]
 [1.        ]
 [1.        ]]
Rastgele eğitim verileri ile yapılan 20 eğitim sonucu test verimizdeki ortalama doğruluk skorumuz:  [0.90714286]
```

Şekil2.d Ağın 20 defa eğitilmesine karşılık gelen doğruluk değerleri ve doğruluk değerlerinin ortalaması

3) Lineer regresyon ile genlikte sürekli algılayıcı aslında birbiriyle aynı fakat ADALİNE'da ağın çıkışındaki sürekli çıktıyı bir eşik değerine göre sınıflandıran bir fonksiyon var.



Şekil3.a ADALİNE ile Lineer Regresyon arasındaki ilişki

Bu soru da aslında bir lineer regresyon sorusu , yaptığımız şey basitçe şekil 3.a'da w ile gösterilen ağırlıkların optimum değerlerini bulmak için minimize edeceğimiz bir hata fonksiyonu belirliyoruz. Bundan sonraki adımları matematiksel notasyonla yazmak istersek:

$$1.\ \phi(z^{(i)})_A = \hat{y}^{(i)}$$
$$2.\ \Delta w_{(t+1),\,j} = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$$
$$3.\ \Delta w_j := \Delta w_j + \Delta w_{(t+1),\,j}$$
$$2.\ \mathbf{w} := \mathbf{w} + \Delta\mathbf{w}$$

1. adımda eğitim setindeki i. X değeri için bir y değeri tahmin ediyoruz.
2. adımda tahmin edilen değerle gerçek çıktı değerini karşılaştırıp bir sonraki iterasyonda ağırlıkta yapılacak güncellemenin büyüklüğünü belirliyoruz.
3. adımda ağırlıkta yapılacak güncellemeyi eski güncelleme değeriyle değiştiriyoruz.
4. ve son adımda ağırlıkları güncelleyip tüm bunları veri setindeki tüm eleman için verilen iterasyon sayısı boyunca tekrar ediyoruz.

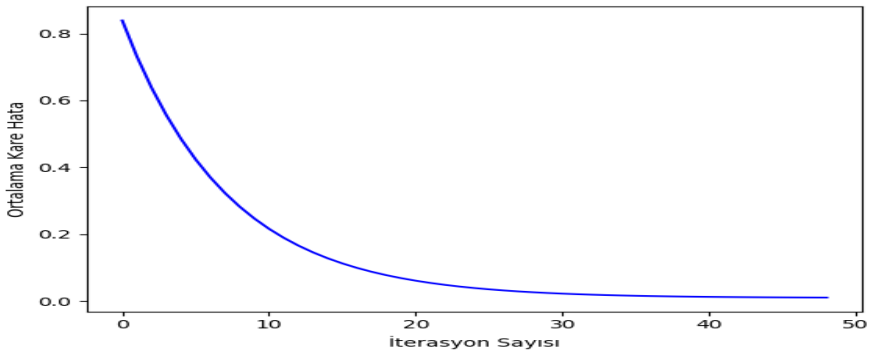
Aktivasyon fonksiyonumuzu y=x olarak seçtik. X1 ve X2 değerlerini verilen aralıkta 50 nokta olacak şekilde rastgele olarak seçtik. Bias eklendikten sonra tüm verilerin %80'ini eğitim için

geri kalan %20'sini test kümesi için ayırdık.

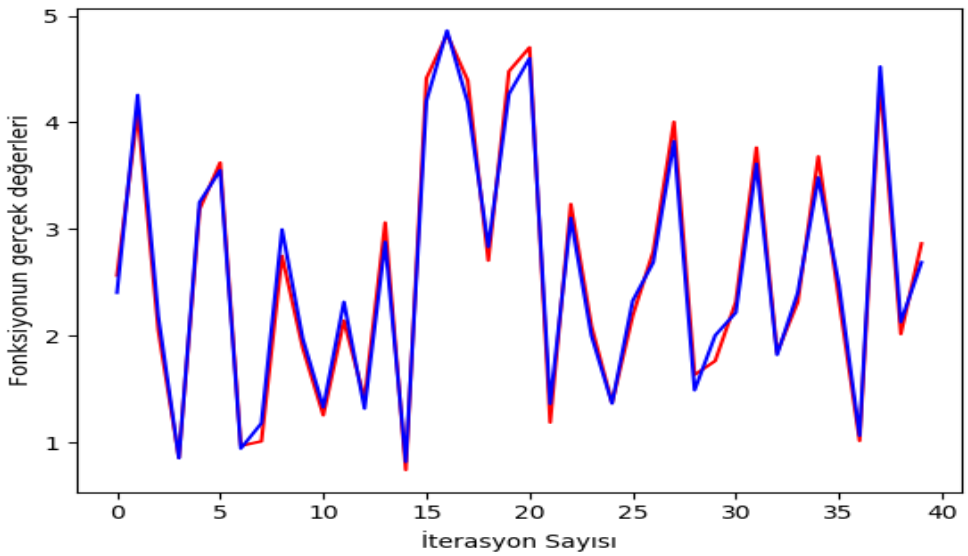
```
            cost[i]=sum(0.5*((y-self.tahminEt(X))**2)/len(y))
            if cost[i]<eps:
                sonİter=i
                break
            else:
                sonİter=i
```

Şekil3.b Durdurma kriterini nasıl belirlediğimiz hakkındaki kod

Durdurma kriterini belirlerken ortalama kare hata değerini esas aldık. Ortalama kare hata fonksiyonun verilen X1 ve X2 değerleri için gerçek çıktısı ile eğittiğimiz modelin çıktısı arasındaki farkın karesinin tüm verilere oranı olarak hesapladık. Bu değeri eğitim fonksiyonumuza parametre olarak girilen **eps** değeri ile karşılaştırarak iterasyonun devam edip etmemesi gerektiğine karar verdik.

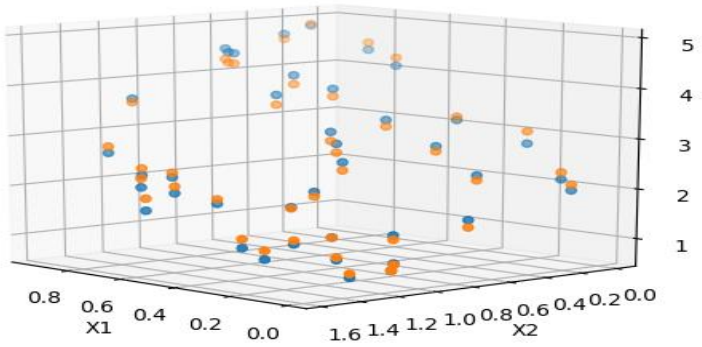


Eğitim fonksiyonumuza eps=0.01 değerini, öğrenme hızı=0.01 vererek eğitimi 1000 iterasyonda tekrarlayıp Şekil3.b'deki kod parçasının çalışıp çalışmadığına baktık. Yukarıdaki grafikte görüldüğü üzere beklendiği gibi ortalama kare hata iterasyon sayısı arttıkça azalmakta ve ortalama kare hatayı çıktı olarak ekrana bastırdığımızda [0.00988896] değeri gözüküyor ve bu değer başta belirlenen eps=0.01 değerinden küçük. Grafikte iterasyon sayısı olarak en fazla 50 değerinin gözükmesinin sebebi modelimiz daha 1000. İterasyona ulaşmadan istediğimiz koşulu sağladığı içindir. Hesapladığımız ortalama kare hata değerinin doğruluğunu ispatlamak için eğitilen model ile fonksiyonun gerçekte aldığı değerlerin noktalar kümesini hem 3 boyutlu hem 2 boyutlu çizdirerek modelin fonksiyona ne kadar yakınsadığını görebildik.



Şekil3.c Fonksiyonun gerçek değerleri ile modelin tahmin ettiği değerler arasındaki ilişki

Şekil3.c'de kırmızı ile gösterilen fonksiyonun gerçek çıktılarının eğitim kümesindeki değerlerinin iterasyon sayısı ile olan değişimi , mavi ile gösterilen tahmin edilen çıktı değerlerinin iterasyonsayısı ile olan değişimidir. Aynı ilişkiyi 3 boyutta noktalar kümesi olarak çizdirince:

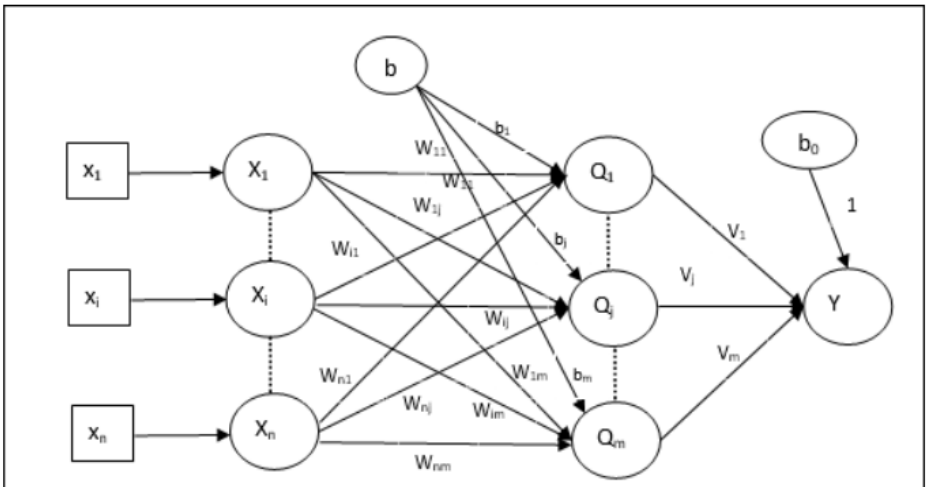


Şekil3.d Şekil3.c'deki ilişkinin 3 boyuta taşınmış hali

Şekil3.c ve 3.d'de görüldüğü üzere tahmin edilen fonksiyonun gerçek fonksiyonu yakın bir ilişkiyle takip ettiği göz önüne alınırsa yukarıda bahsettiğimiz [0.00988896] ortalama kare hata değeri anlam kazanmaktadır.

4) Birden fazla genlikte sürekli algılayıcı yapısıyla bu problemin çözülebileceğini öngördük. Bu yapıya Multiple Adaptive Linear Neuron(Madaline) adı veriliyor. Adından da anlaşılabileceği

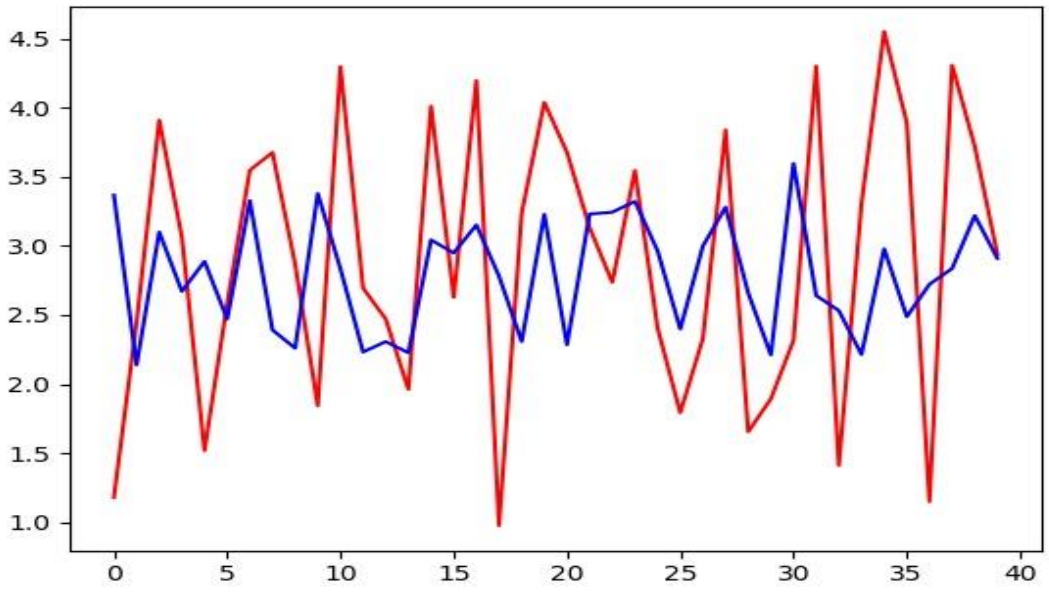
gibi birden fazla Adaline yapısının alt alta sıralanmasıyla oluşuyor.



Şekil4.a Madaline yapısının mimarisi

Şekil4.a'da n adet X değerinin her biri ilk katmandaki ağırlık matrisiyle çarpılarak m adet Adaline yapısına giriyor ve bu katmandan V1,….,Vm olmak üzere m adet çıktı oluşuyor.

Bizim problemimiz için m=2. 3. Soruyu bu yapıyla çözmeyi denediğimizde modelimizin ürettiği fonksiyonun çıktıları gerçek fonksiyon değerlerini takip ediyor fakat aradaki fark biraz fazla çıktı yani 3. Sorudaki tek katmanlı Adaline yapısı daha iyi bir çözüm sunuyor.



Şekil4.b Madaline yapısı ile 3. Soruyu çözünce ortaya çıkan sonuç