**Artificial\_Neural\_Networks**

Perceptron and Delta learning in Single Layer

**Kodun çalıştırılması**

*1-Perceptron Learnin Rule :*

1. Kod çalıştırıldıktan sonra açılan form ekranında PictureBox içerisindeki koordinat düzleminde istenilen yerlere yandaki class radiobuttonlarından biri seçilerek noktalar eklenir.
2. Daha sonra Menu’den Process->İnitialize->Binary itemlerine tıklanır. (Burada tek katman için işlem yaptığımız için tek boyutlu çözümleme yapılacaktır. Noktalar karışık olursa perceptron çözümlemek için yeterli olmayacaktır.)

*2-Delta Learnin Rule :*

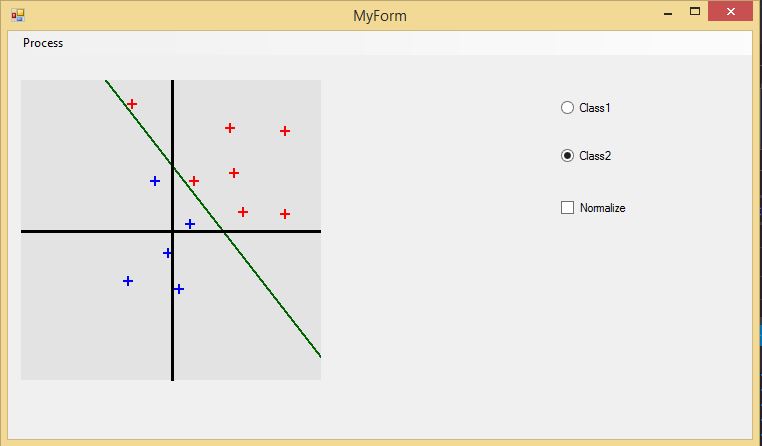
1. Kod çalıştırıldıktan sonra açılan form ekranında PictureBox içerisindeki koordinat düzleminde istenilen yerlere yandaki class radiobuttonlarından biri seçilerek noktalar eklenir.
2. Daha sonra sağ taraftaki normalize checkbox’ı işaretlenir.
3. Son olarak Menudeki Process->Training-> Continous itemlerine tıklanarak çalıştırılır. (Burada da perceptrondaki gibi tek katmanlı çözümleme yapılmaktadır. )

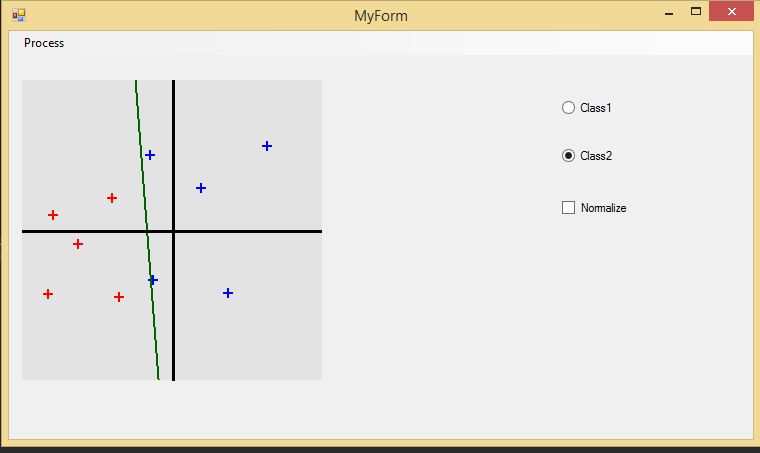
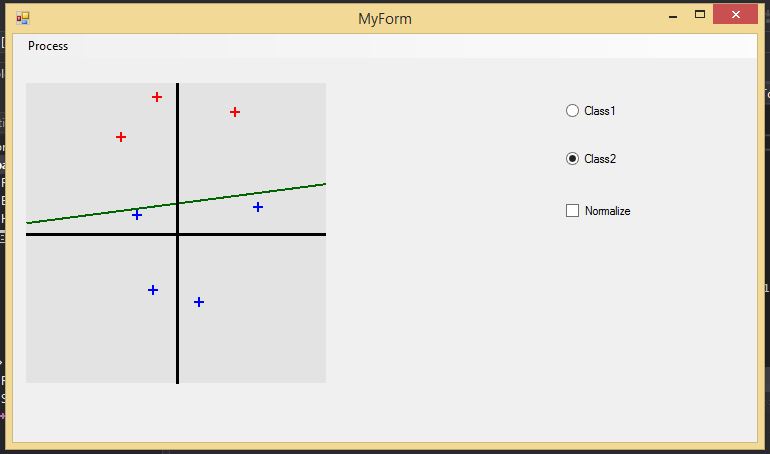
***Perceptron Learning Rule:***

Öncelikle Visual Studio’da bir form ekranı oluşturdum ve bir PictureBox ekledim. Bu PictureBox’ta koordinat düzlemi sol üst köşe noktası (0,0) noktası olduğu için mouseClick ile aldığım noktaları bu koordinat düzleminde hesaplasam da Kartezyen koordinat düzlemi ekleyip ekranda doğrumu ona göre çizdirmem gerekti. İlk başta Kartezyen koordinat düzlemini pictureBox1\_Paint’te DrawLine() fonksiyonu ile çizmiştim. Fakat adımlar ilerledikçe öğrenme sırasında her doğru çiziminde refresh yaptığımda koordinat düzlemini de tekrar tekrar çizdirmem gerekecekti. Üzerinde işlem yapmadığım için label olarak pictureBox’a eklemek daha mantıklı geldi ve başta yazdığım kodları yorum satırı haline getirerek bu şekilde devam ettim. Mouse\_Click ile noktaları aldım ve data[] dizisinde tuttum. Bu data dizisinde her nokta için data[i]= x, data[i+1]= y, data[i+2]= sbt(bias)=1, data[i+3] ise class1 için -1 class2 için 1 dir.

Form ekranı tamamlandıktan sonra binaryToolStripMenuItem\_Click’te Perceptron Öğrenme kuralını gerçekleştirdim. Her nokta için datada 4 eleman olduğu için döngülerim i+=4 artarak ilerliyor. Öğrenme hata minimum olana kadar (error<0) olana kadar devam edeceği için while(Error > 0) döngüsünü kullanıyorum. Kodda yorum satırları ile tek tek işlemleri açıklamış bulunmaktayım. Ama genel olarak işlemlerden bahsetmem gerekirse; öncelikle net ile sgnNet hesaplandı. Daha sonra bunlara göre ağırlıklar güncellendi ve adım adım tek tek doğru çizilerek classları ayıran doğru hesaplanıp çizilmiş oldu.

Örnek çıktılar;



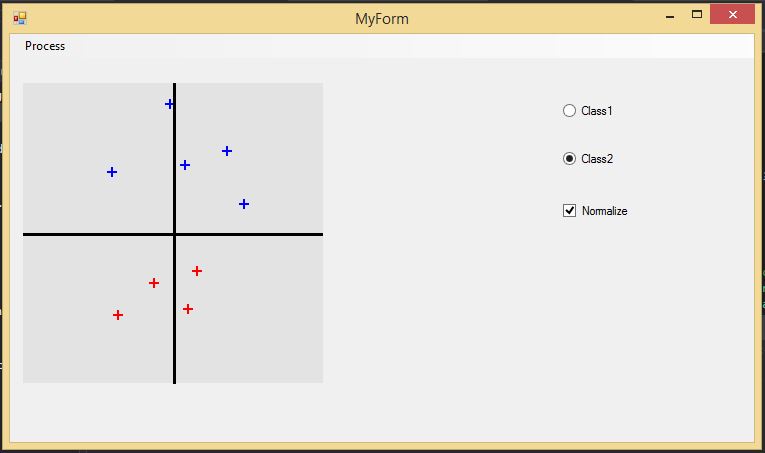


***Delta Learning Rule:***

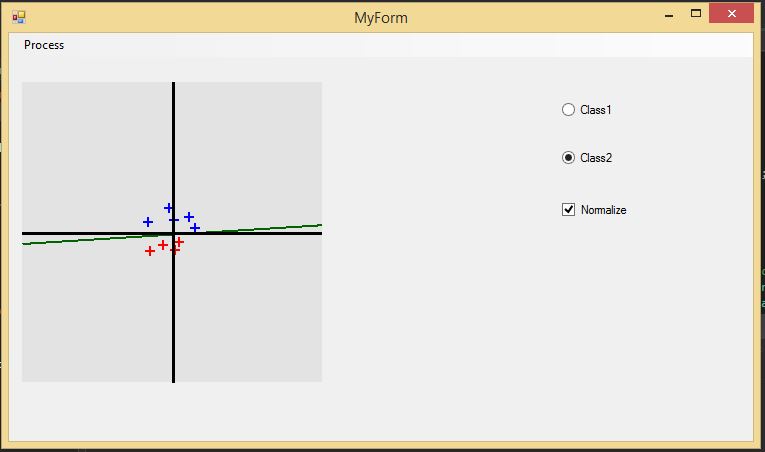
Perceptron yöntemine ana hatları ile benzese de Delta daha hassas bir değerlendirme yapar. Delta’da kullandığımız fnet() fonksiyonunda 1/(1+ e^(-lamda\*net) )-1 hesabında noktalarımızın kendi değerlerini kullandığımızda e üzeri çok büyük bir değer geleceği için öncelikle noktalarımızı normalize etmemiz lazımdır. Bunun için iki yöntem vardır. Birincisi noktanın boyuna bölündüğü zaman normunu vermesidir. Diğeri ise benim tercih ettiğim standart sapma hesabına dayanan x-m/varyans hesaplamasıdır. İşlem adımlarını yine kodda açıklamış bulunmaktayım. Yeni noktalarımı normalize[] dizisinde tuttum ve delta hesabını bunun üzerinden gerçekleştirdi. (data[i+1] ve data[i+2] elamanları değişmediği için onları normalize dizisine direk aktardım.)

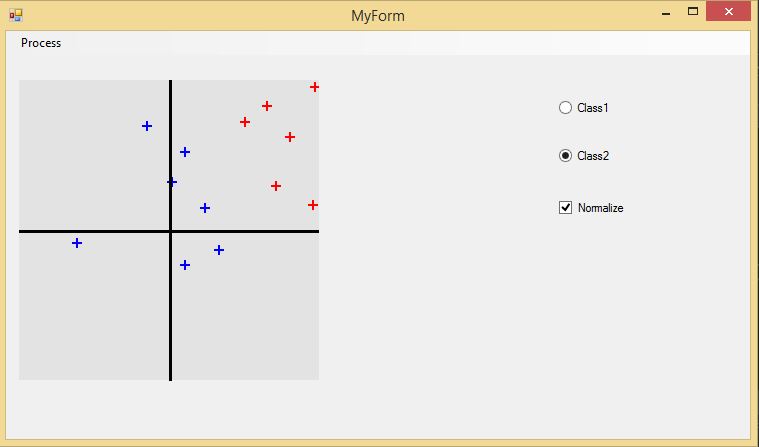
Noktalarımı normalize edip dizime aktardıktan sonra delta hesaplama adımlarına geçtim. Genel olarak döngülerimin mantığı perceptron ile aynı, yine erroru kontrol edecek ve dizinin boyutu counter(nokta sayısı)\*4 olduğu için i+=4 ile ilerleyecektir. Net hesaplamamız da aynı olmakla birlikte artık data[] dizisini değil normalize[] dizisini kullanmaktayım. Net hesaplandıktan sonra fnet() fonksiyonunu[ 1/(1+ e^(-lamda\*net) )-1 ] hesaplamam gerekiyor. Yeni ağırlık denklemim W^k+1 = wk + m (di-fneti) \* fnet\_turev \* x; olduğu için fnet fonksiyonunun türevini hesaplamam lazımdı. fnet\_turev = 1/2\*(1-(fnet^2)); ile onu da hesapladıktan sonra ağırlıklarımı hesaplıyorum. (İşlem adımları kodda yorum satırı olarak ekli.) Yeni error hesaplamamı Error+=1/2((d-o)^2) ile hesapladım. Hesaplama adımları bittikten sonra ekranda sonuçları görmemizdeki en büyük engel noktaların normalize edilmiş olmasıdır. Hesaplama sırasında yanlış sonuçlar ortaya çıkarabileceği için ekranda sonuçları görebilmek için noktaları aynı oranda bir kat sayı ile çarparak büyüttüm(Ben bunun için \*15 oranını yeterli buldum.) Tabi burada yeni bir sorun oluştu noktalar büyütüldüğü zaman doğrumuz doğru da çizilse çıktıda yanlış sonuçlar görebiliyoruz. Doğru denkleminde hesaplama yaparak noktaların büyütüldüğü zaman ax+by+c de sadece c katsayısının değiştiğini gördüm. Kendi denklemim w1x+w2y+w3 olduğu için w3\*15 yaparak bu sorunu da çözdüm. (Noktalarımı en son çizilen doğruda normalize edilmiş şekilde görebilmek için refresh’te normalize[] dizisinin elemanlarına göre çizdim.)

Örnek ekran çıktıları;



İnput noktalarımı bu şekilde aldıktan sonra normalize kutusunu işaretliyorum ve çalıştırmak için önce initialize sonra Process->Training-> Continous dediğim zaman işlem çıktısı :





İnput noktalarımı bu şekilde aldıktan sonra normalize kutusunu işaretliyorum ve çalıştırmak için önce initialize sonra Process->Training-> Continous dediğim zaman işlem çıktısı :

