**Yapay Sinir Ağları**

**1.Ödev**

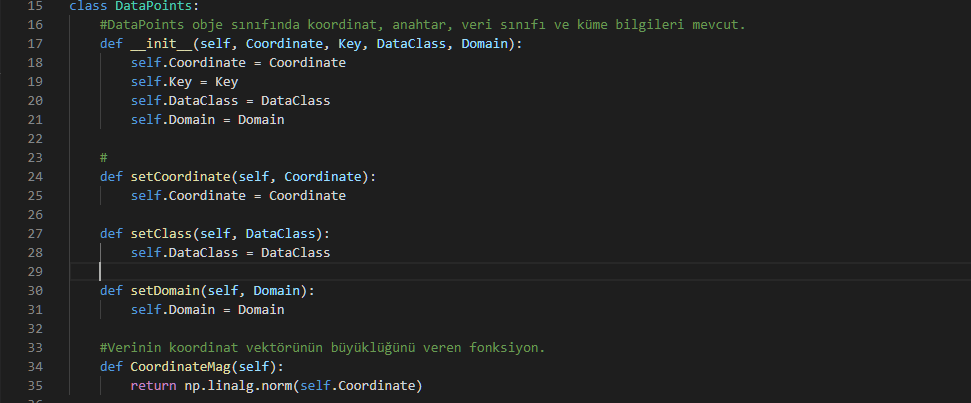
**22.11.2020**

**Mehmet Şerbetçioğlu -- 040160056**

**Ahmet Hulusi Tarhan -- 040170738**

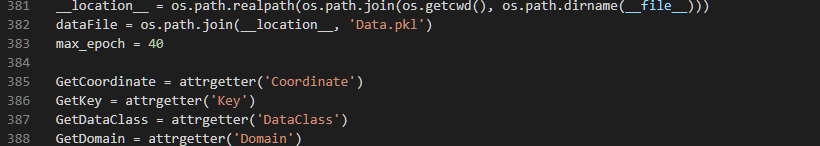
**Soru 1:**

Öncelikle bir “DataPoints” obje sınıfı oluşturuldu. Bu sınıfın içinde koordinat, anahtar, veri sınıfı ve verinin kullanılacağı küme bilgileri mevcut. Ek olarak bu bilgilerin değişmesini sağlayacak ve koordinatın vektörel uzunluğunu verecek fonksiyonlar var.



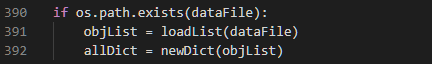
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

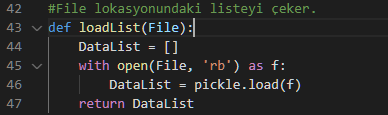
İlk olarak kullanılacak verilerin saklanacağı yeri, veri sayısını ve verilerden bilgi çekebilecek basit fonksiyonlar tanımlandı.

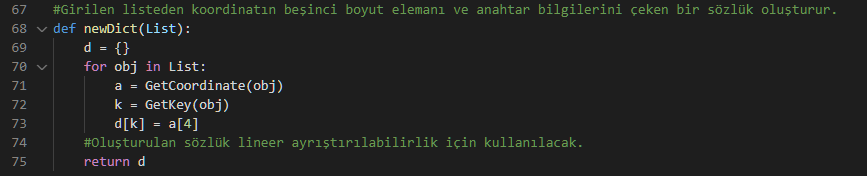


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

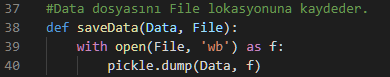
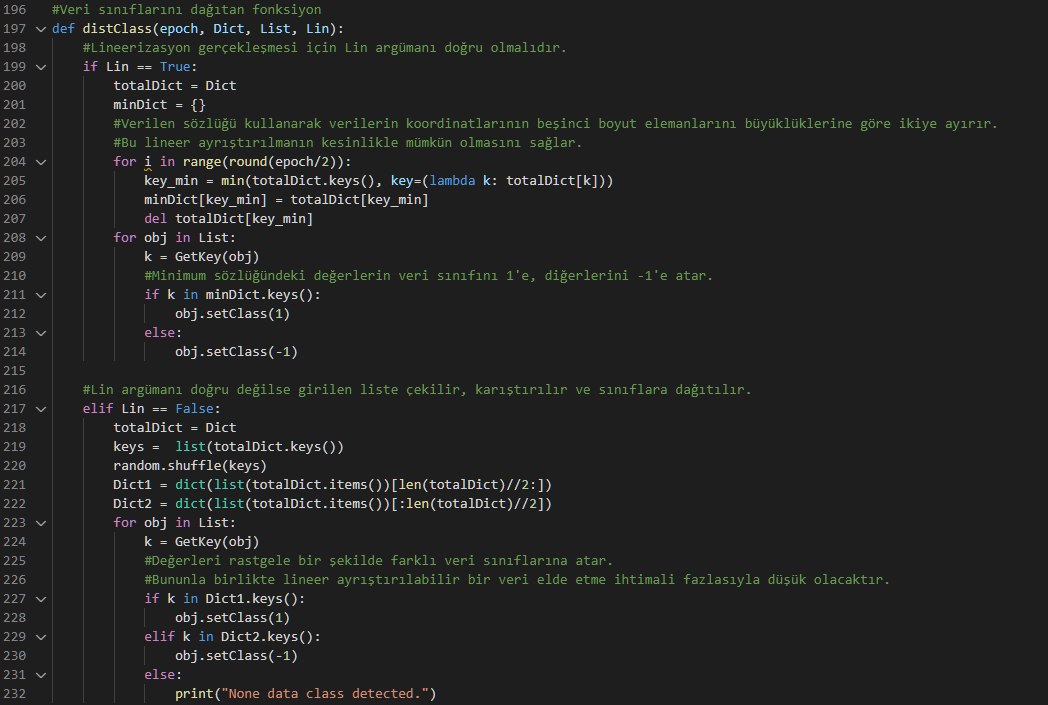
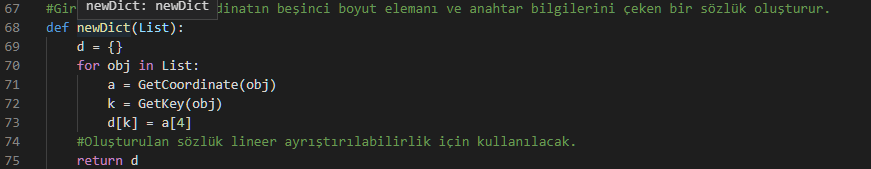
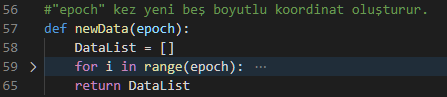
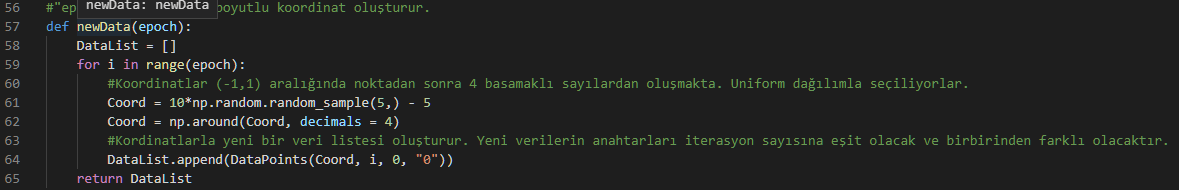
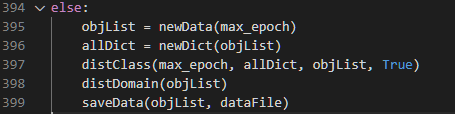
Ardından belirtilen lokasyonda bir veri dosyası var mı diye kontrol edilir. Eğer belirtilen dosya bulunmaktaysa dosyadaki veriler “objList” listesine yüklenir. Ardından bu listedeki verilerin beşinci boyut koordinatları ve anahtarlarıyla yeni bir sözlük meydana getirilir.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



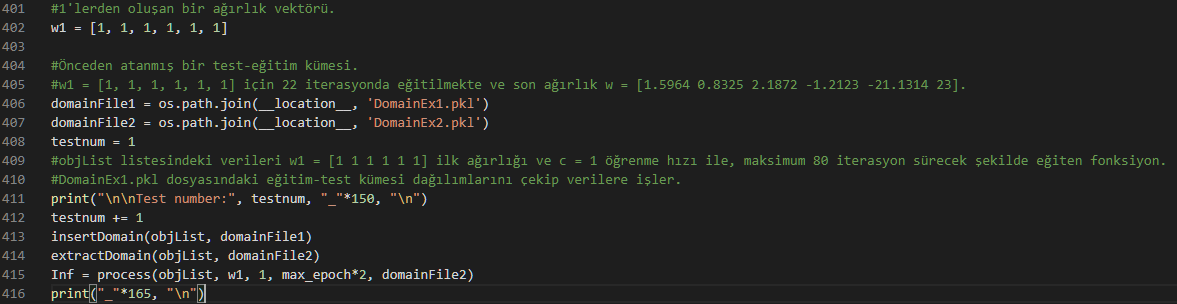
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

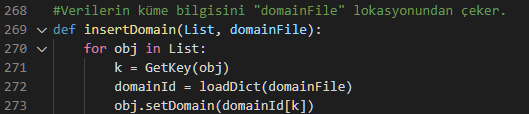
Eğer böyle bir veri yoksa, belirtilen veri sayısında yeni veri oluşturulur. Bu verilerin koordinatları noktadan sonra dört basamaklı, (-5,5) aralığında rastgele seçilir. Oluşturulan veriler “objList” listesine kaydedilir. Bu listeyle yeni bir sözlük oluşturulur. Daha sonra “distClass” komutuyla listedeki veriler yarı yarıya beşinci boyut elemanlarının büyüklüğüne göre iki sınıfa ayırılır. Ardından eğitim ve test kümeleri dağıtılır ve son olarak veri kaydedilir. “distClass” fonksiyonunda dördüncü argümanda False kullanılırsa veriler rastgele sınıflara dağıtılıp lineer ayrıştırılabilir olması kesinleşmeyecektir.

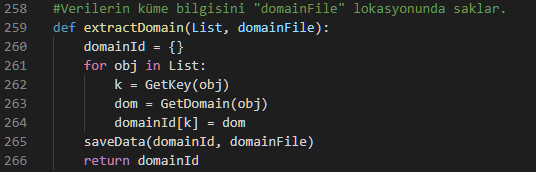


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

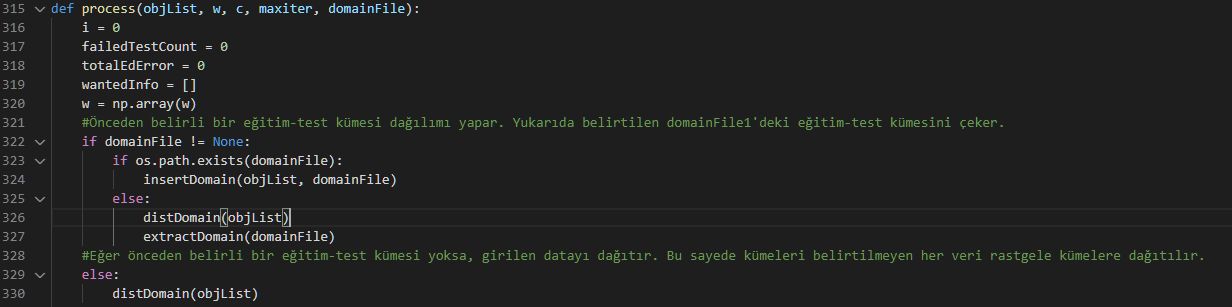
Bu şekilde veriler oluşturulup sınıflara ve kümelere dağıtılmış olur. Eğitimi başlatmak için w1 = [1, 1, 1, 1, 1, 1] ağırlığı seçelim. Öncelikle daha önce eğitilmiş ve test kümelerini başarıyla sınıflandırmış DomainEx1 kümesiyle test yapalım.

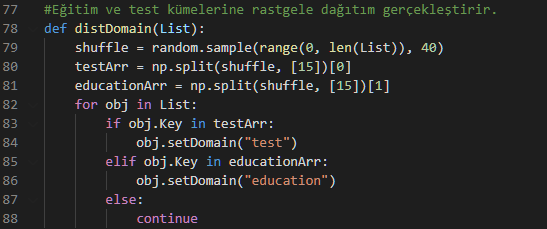
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

İlk olarak w1, DomainEx1 dosya lokasyonu ve DomainEx2 dosya lokasyonu tanımlanır. Burada DomainEx2 dosyasının amacı DomainEx1in değişmemesidir. DomainEx2, DomainEx1i kopyalar ve eğitim DomainEx2 ile başlar. Kümeler anahtarlarıyla birlikte kaydedilip tekrar yüklenirken anahtarları kullanılarak verilerle eşlenirler.



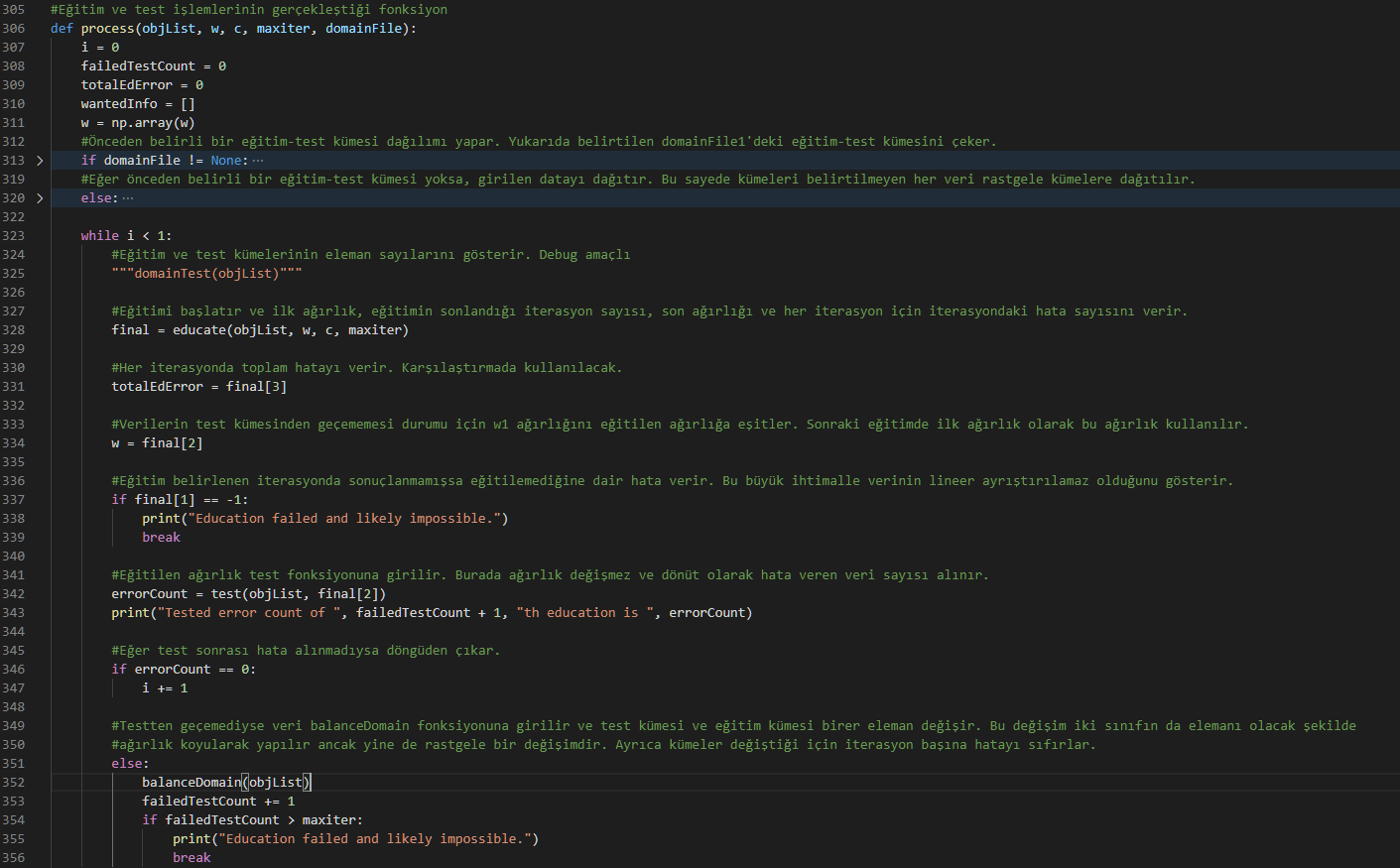
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**  
Daha sonra işlemlerimiz başlar. Öncelikle bazı değişkenler tanımlanır. Fonksiyonun argümanında bir veri kümesi olup olmadığına bakılır. Eğer varsa ve veri dosyası bulunuyorsa dosyadaki kümeler kullanılır. Argümanda belirtilip dosyası bulunmuyorsa kümeler dağıtılıp yeni dosya oluşturulur. Eğer argüman “None” olarak girilmişse sadece kümeleri dağıtır, dosya yüklemez ve oluşturmaz. Kümeler ilk seferde rastgele bir şekilde dağıtılır.



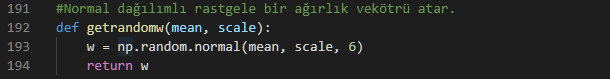
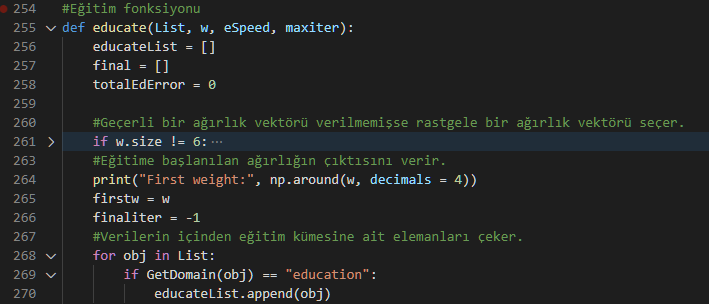


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Daha sonra eğitim başlatılır. Eğitim sonucu ilk ağırlık, iterasyon sayısı, son ağırlık ve iterasyondaki değişim sayısının bulunduğu bir liste alınır. Eğitimdeki son ağırlık, bir sonraki eğitimde ilk ağırlık olmak üzere güncellenir. Eğer iterasyon sayısı -1’se, yani iterasyon sayısı verilen maksimum iterasyon sayısını geçmişse eğitim başarısız sayılır. Daha sonra eğitimle elde edilen ağırlık test aşamasına girer. Buradan testte gerçekleşen hata miktarı alınır. Eğer hiç hata alınmamışsa döngüden çıkılır ve eğitim tamamlanır. Fakat test aşamasında hata görüldüyse test ve eğitim kümeleri dengelenir, başarısız test sayısı 1 artırılır. Eğer bu işlem “maxiter” sayısından fazla kez gerçekleşirse eğitim durdurulup başarısız sayılır.

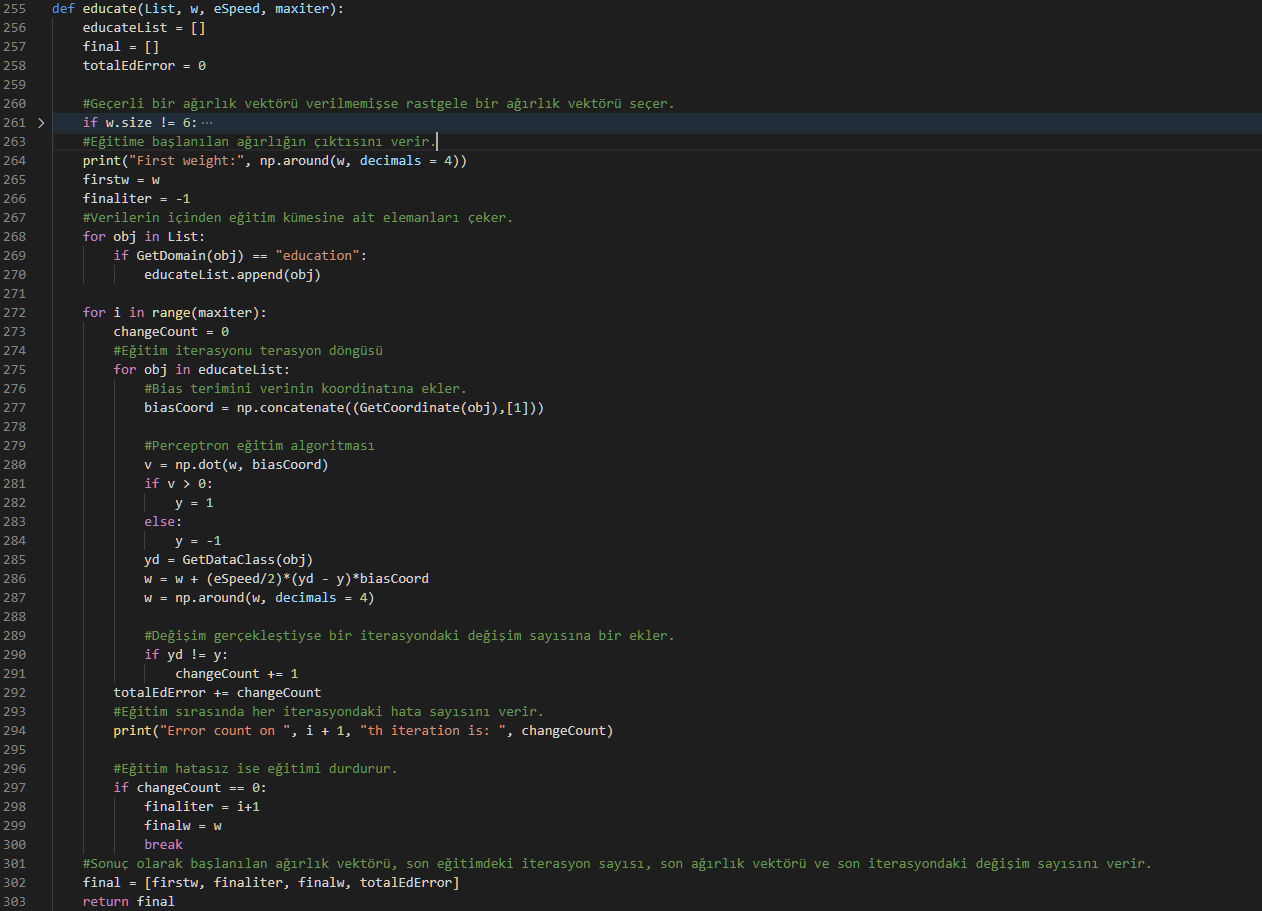


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

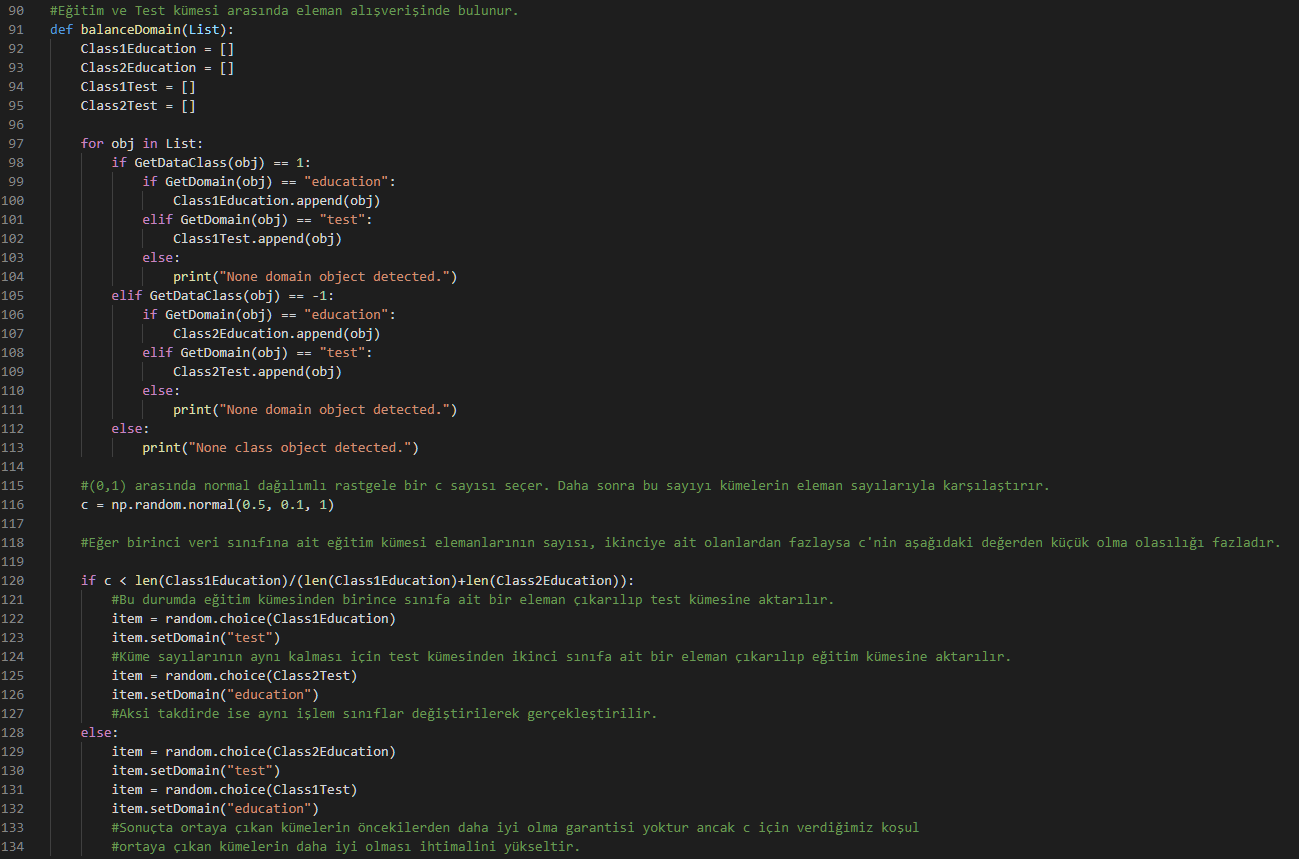
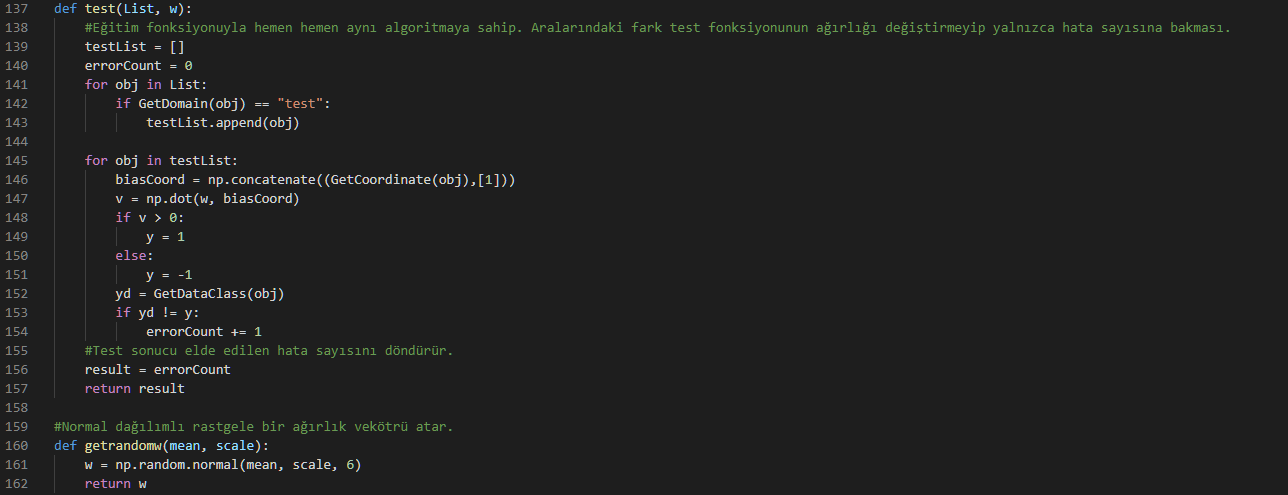
Eğitimi başlatırken veriler, ilk ağırlık, eğitim hızı ve maksimum iterasyon sayısı alınır. Eğer ağırlık geçerli değilse rastgele bir ağırlık atanır. Bunun için getrandomw fonksiyonuyla ortalama değeri 0 ve standart sapması 1 olan normal dağılımlı altı boyutlu bir w vektörü üretilir. İlk ağırlık belirlenir, finaliter değişkeni, iterasyon sayısının maksimumu aşması ihtimaliyle -1’e atanır. Eğitim kümesi elemanları listesi oluşturulur ve eğitim başlar. 

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Eğitim kümesi “maxiter” kez eğitilir. Bu sayı aşılırsa “process” fonksiyonuna son iterasyon sayısı “-1” olarak döner ve eğitim başarısız sayılır. Öncelikle iterasyondaki değişim sayısı 0’a eşitlenir. Eğitim kümesindeki her veri için bias koordinatı eklenir. Daha sonra w ağırlık vektörü ile v = wT\*x işlemi yapılır. Elde edilen v 0’dan büyükse y = 1, küçükse y = -1 aktivasyon fonksiyonu tanımlanır. yd değeri verinin sınıfıyla eşitlenir. Daha sonra w = w + (c/2)\*(yd – y)\*x perceptron algoritması kullanılarak her veri için ağırlık vektörü güncellenir. y değerinin yd’den farklı çıkması durumunda değişim sayısı 1 artırılır. Bu şekilde tüm eğitim kümesi üzerinden geçildiğinde bir eğitim iterasyonu tamamlanmış olur. Eğer iterasyon sonucunda değişim sayısı 0 bulunursa son iterasyon ve son ağırlık çekilip eğitim durdurulur. “process” fonksiyonuna ilk ağırlık, iterasyon sayısı, son ağırlık ve son iterasyondaki toplam hata bilgileri döner.

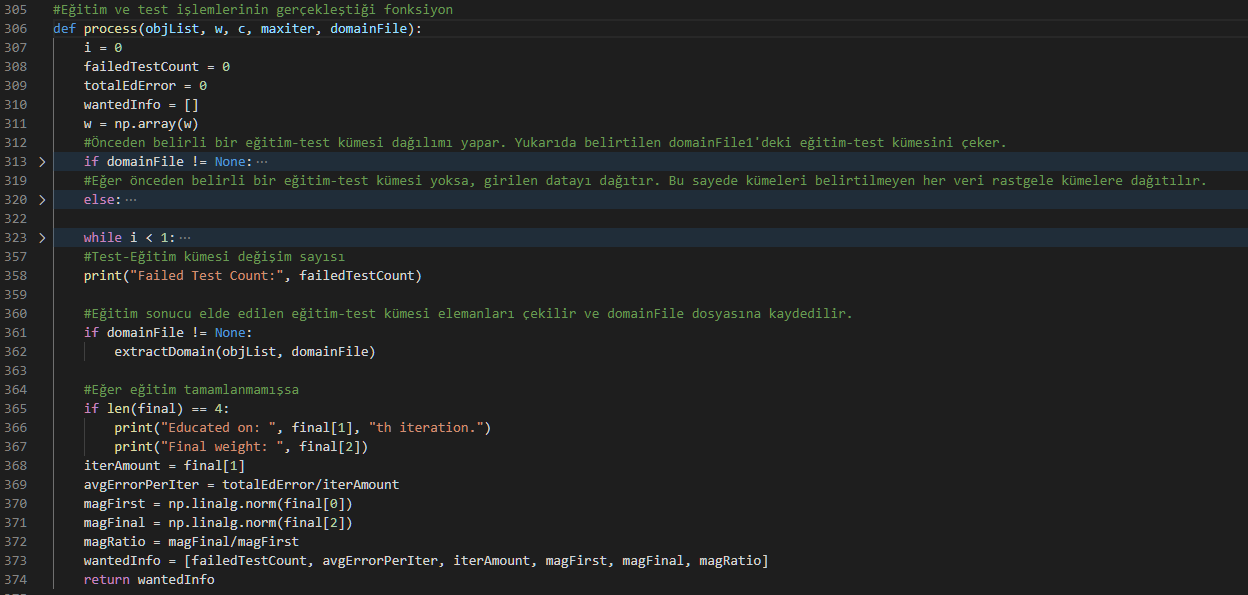


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

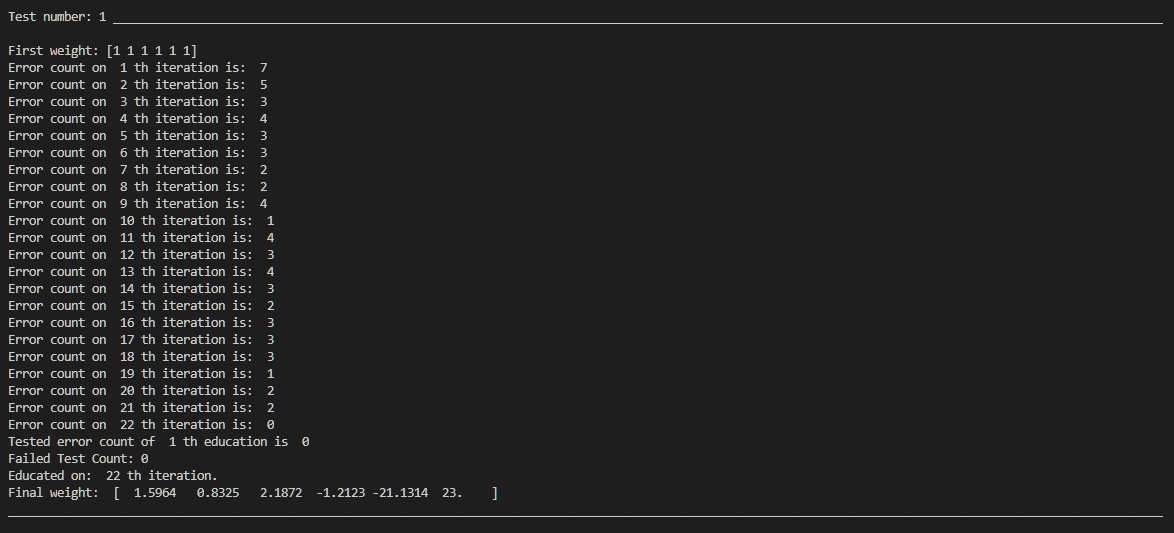
Daha sonra güncellenen ağırlık test aşamasına girer. Bu, eğitim fonksiyonuyla aynı karşılaştırmayı yapar fakat ağırlıkları güncellemez. Yalnızca yd = y karşılaştırması yapıp farklı çıkınca errorCount değerini 1 artırır. Test sırasında hata bulunmuşsa balanceDomain fonksiyonu ile eğitim kümesi ve test kümesinden birer eleman yer değiştirir. Bu değişim, rastgele olmasına rağmen iki veri sınıfının test ve eğitim kümesinde bulunan elemanlarının sayılarını yakınlaştıracak şekilde ağırlıklandırılmıştır. process fonksiyonunda errorCount = 0 olmadıkça veya hatalı test ya da eğitim sayısı maxiter değerini aşmadıkça eğitim-test döngüsü devam eder. 

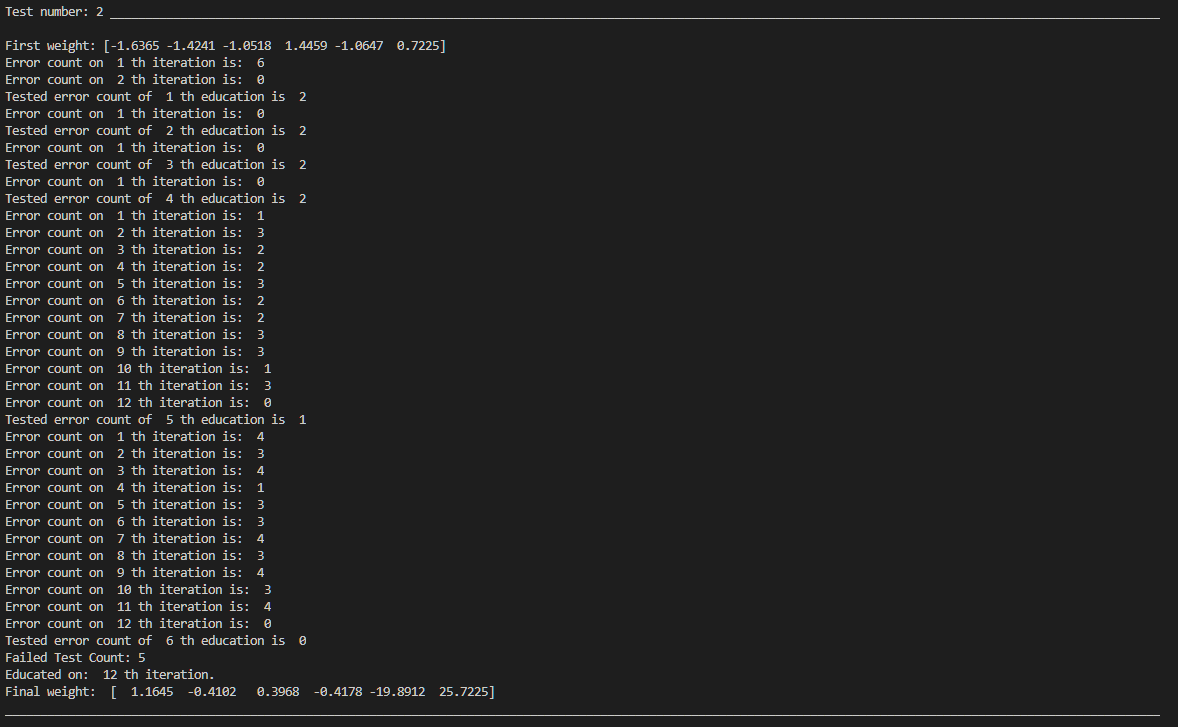
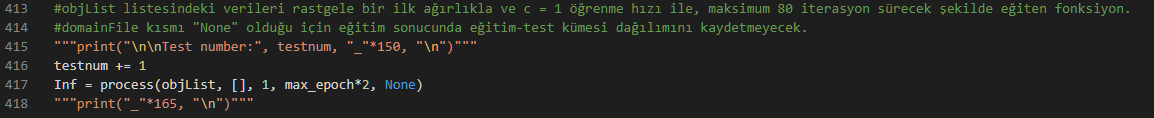
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

process fonksiyonu sonuncunda kaç testin başarısız olduğu, hangi iterasyonda eğitimin gerçekleştiği ve son ağırlık vektörü görülür. Başarısız test sayısı, iterasyon başına ortalama hata, iterasyon sayısı, ilk ağırlık, son ağırlık ve ilk ve son ağırlıkların oranı bilgileri elde edilir.

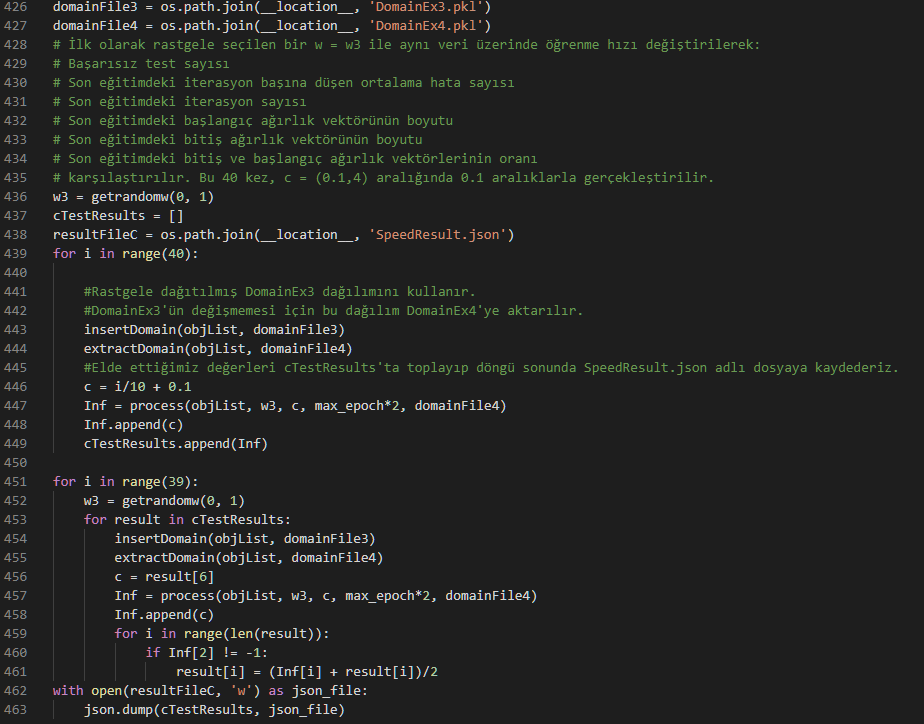


**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

w = [1, 1, 1, 1, 1, 1] için c = 1 eğitim hızıyla daha önce belirtilen DomainEx1 küme dağılımıyla eğitim yapıldığında şu sonuç elde edilir: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Bu işlem rastgele bir w ve küme dağılım argümanlarıyla gerçekleştirilince örnek olarak böyle bir sonuç elde edilebilir. balanceDomain fonksiyonu rastgele çalıştığı için, doğru bir test-eğitim kümesi dağılımı kullanmadan aynı argümanlarla aynı sonuçları tekrarlama ihtimali çok düşüktür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Eğer rastgele seçilen bir ağırlık vektörü ve önceden belirlenmiş bir dağılım kümesi için sadece eğitim hızı değiştirilerek ölçüm alınır. Bu 40 kez gerçekleştirilip elde edeceğimiz sonuçların ortalaması SpeedResult.json dosyasında saklanır. **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Sonuçlara bakılırsa, hata sayısı her zaman 0’da kalmıştır. Bu seçilen kümelerin iyi dağılmasından kaynaklanır. Bu şekilde kümeleri dengelemeye gerek kalmaz.

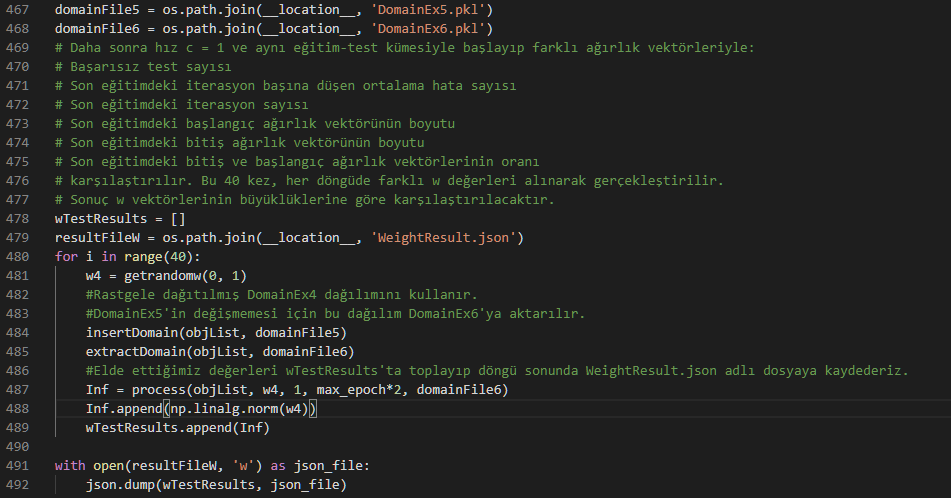
Her eğitim iterasyonunda meydana gelen ortalama hataya bakıldığında, 3 ile 4 arasında devam ettiği görülmektedir. Çok küçük eğitim hızı değerlerinde yüksektir, 0.3 ve 1 aralığında optimal değerlere sahiptir ve sonrasında tekrar yükselir. Buna rağmen büyük farklılık göstermez.

Son eğitimdeki iterasyon sayılarına bakıldığında, grafiğin oldukça dalgalı olmasına rağmen c = 2.8 ve üzeri eğitim hızlarının 13 civarı iterasyonda eğitimi tamamladığı ve önceki değerlerden oldukça hızlı olduğu gözlemlenir. Eğitim hızının yükselmesiyle eğitimin hızlanması beklendiktir.

Son eğitimdeki ilk ağırlık vektörünün büyüklüğüne bakılırsa sabit olduğu gözlenir. eğitim esnasında test kümesi hiç başarısız olmadığı için aynı ağırlık vektörü kullanılmıştır. Bu sebeple ilk ağırlık vektörü her zaman testin başında seçilen w3 vektörü olacaktır. Bu vektörün büyüklüğü 2.0188 değerindedir.

Son ağırlık vektörünün büyüklüğüne bakılırsa lineer olarak arttığı gözlenir. Eğitim sırasında iterasyon sayısının hıza göre çok da fazla değişmemesi ve ağırlık vektörü güncellenirken eğitim hızıyla doğru orantılı bir şekilde büyümesi bunu sağlar. İlk ağırlık vektörü tüm hızlarda sabit olduğu için aynı grafik son-ilk ağırlık vektörü oranlarında da görülür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Her seferinde rastgele bir ağırlık vektörü seçip önceden belirlenmiş bir dağılım kümesi ve sabit eğitim hızı kullanır ve sonuçlar ilk seçilen ağırlık vektörünün büyüklüğüyle karşılaştırılır. Bu 40 kez gerçekleştirilip elde edeceğimiz sonuçların ortalaması WeightResult.json dosyasında saklanır. 

Bu deneyde test kümesinden birkaç kez hata alınmıştır. Hatalı olan kümelerin ortalama 20 seferde dengelendiği görülür. Eğitim ve test kümeleri iyi dağılmıştır.

İterasyon başına düşen ortalama hata ilk ağırlık vektörünün büyüklüğünden ilişkisiz durmaktadır. Test kümesinin hatalı olduğu deneylerde düşük olduğu gözlemlenmektedir. Bunun sebebi test kümesindeki bir veya iki elemanın hatalı vermesi, bu elemanlar eğitim kümesine geçene kadar da ortalama hatanın düşmesidir.

Son eğitimin iterasyon sayısı da ilk ağırlık vektörünün büyüklüğünden ilişkisiz durmaktadır fakat test kümesinin hatalı olduğu deneylerde düşüktür. Bunun sebebi test ve eğitim kümelerinin, eğitim tamamlanana kadar optimize olmasıdır. Diğer deneylerde de eğitim tamamlanmasına rağmen, verimsiz kümeler kullanılmıştır.

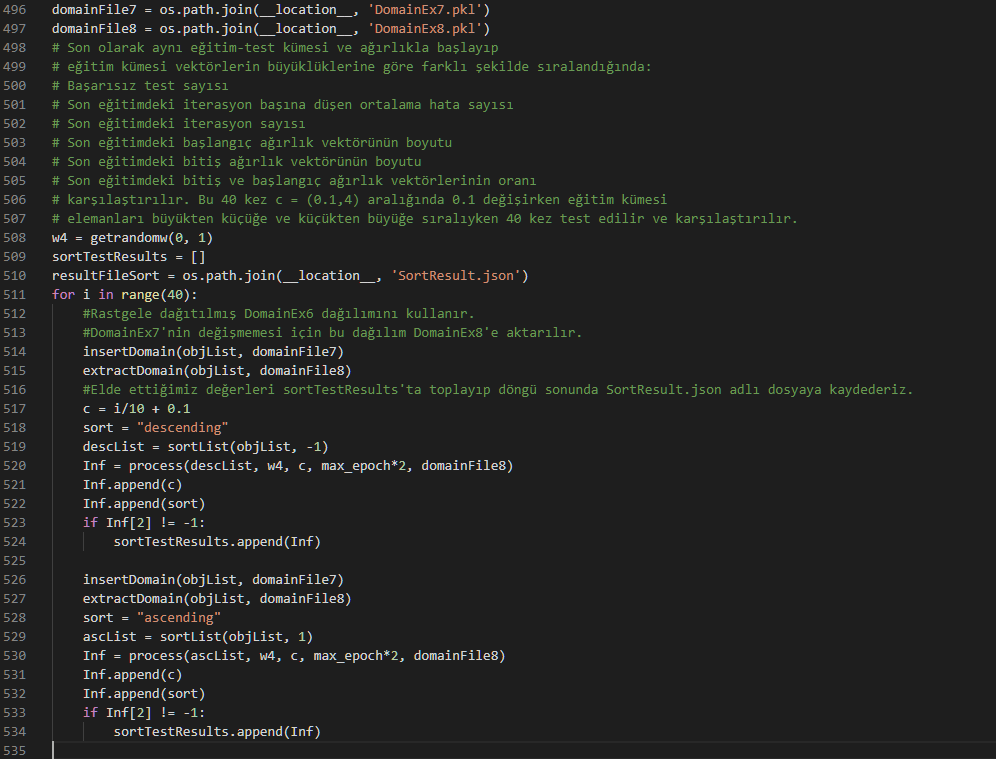
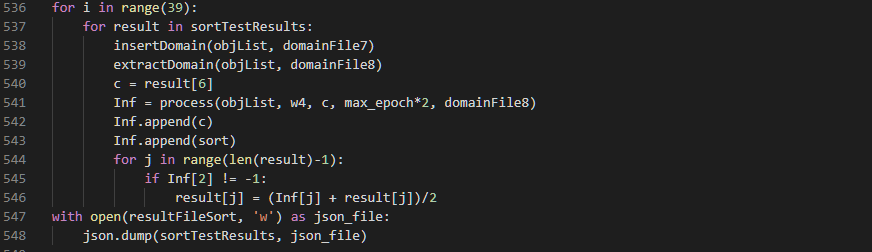
Son eğitimin ilk ağırlık vektörü, hata olmadığı sürece ilk girilen ağırlık vektörünün büyüklüğüne eşittir. Hata yaşanan deneylerde ilk ağırlık gitgide büyümektedir.

İlginç bir şekilde son ağırlık vektörü büyüklüğü, hatalı testlerden ve ilk vektörün ağırlığından ilişkisiz bir sonuç vermektedir. Bu vektörün büyüklüğünü yalnızca veri setinin belirlediği gözlemlenir.

Son ağırlık vektörünün büyüklüğünün diğer parametrelerden ilişkisiz olması ve hatalı deneylerde ilk ağırlık vektörünün büyümesi sebebiyle, son eğitimin son-ilk ağırlık vektörleri büyüklükleri oranı çok düşük olacaktır. Bunun deneylerin dışında, yine son ağırlık vektörü nispeten sabit hareket ettiği için, ilk ağırlık vektörü büyüklüğü artarken son-ilk ağırlık vektörü büyüklükleri oranı düşmektedir.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Farklı eğitim kümesi sıralamalarına göre, eğitim hızı değişirken gözlem yapılır. Rastgele seçilmiş w4 ağırlık vektörü ve DomainEx7 küme dağılımı sabit tutulur.

Sonuçlara bakıldığında, hatalı test sayısının yükselen sıralamada daha düşük olmasıyla birlikte iniş ve çıkışların sıralamadan fazla etkilenmediği görülür. Eğitim hızı 1.5-2.5 aralığında hatalı test sayısında bir artış vardır ve aralığın dışında genel olarak düşüktür.

İterasyon başına düşen hata da sıralamadan bağımsız olmakla birlikte, eğitim hızı 0.8-1.9 aralığında düşüktür.

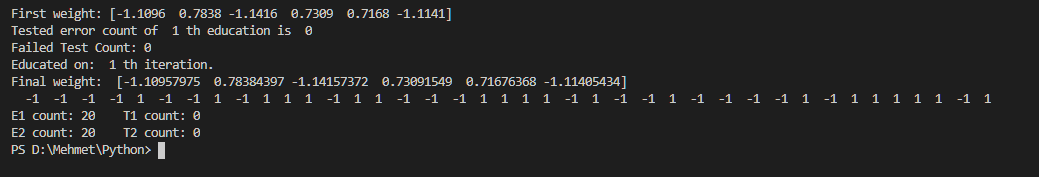
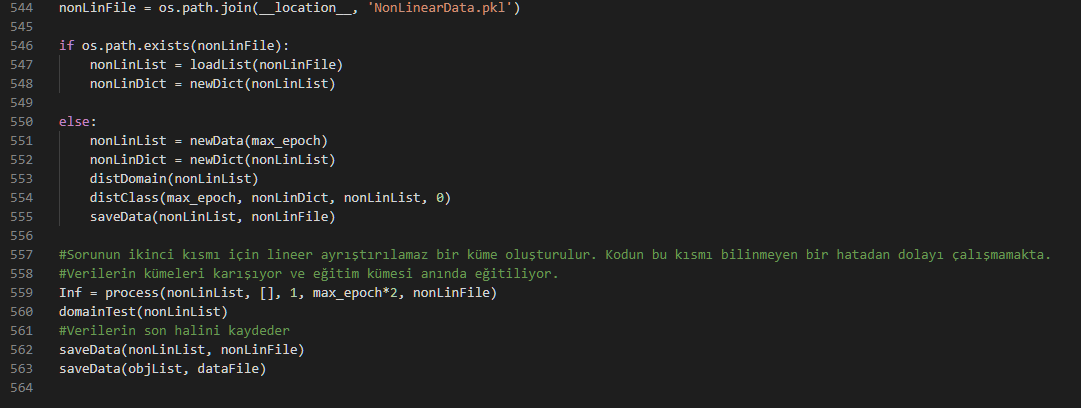
Son eğitimde gerçekleşen iterasyon sayısı da verilerin sıralamasıdan etkilenmemektedir.

Son eğitimdeki ilk ağırlık vektörü de sıralamadan etkilenmemektedir. Büyüklüğü, yüksek ve düşük eğitim hızlarında, 1-3 aralığındakinden daha düşüktür.

Son eğitimin son ağırlık vektörü; ilk ağırlık vektörü, iterasyon sayısı ve eğitim hızıyla doğru orantılı olduğundan sıralamadan bağımsız bir şekilde, eğitim hızı arttıkça artmaktadır.

Son olarak, son ağırlık vektörlerinin ilk ağırlık vektörlerine oranı karşılaştırıldığında bunların da sıralamadan bağımsız olduğu görünür.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

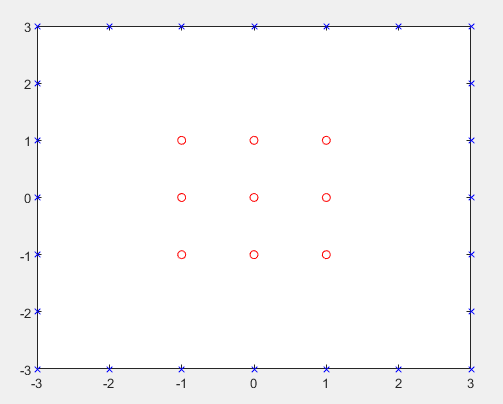
Sorunun diğer kısmı için malesef lineer ayrıştırılamaz bir veri seti elde edilemedi. 

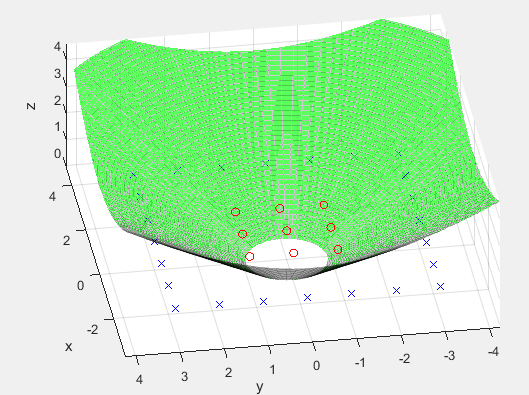
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

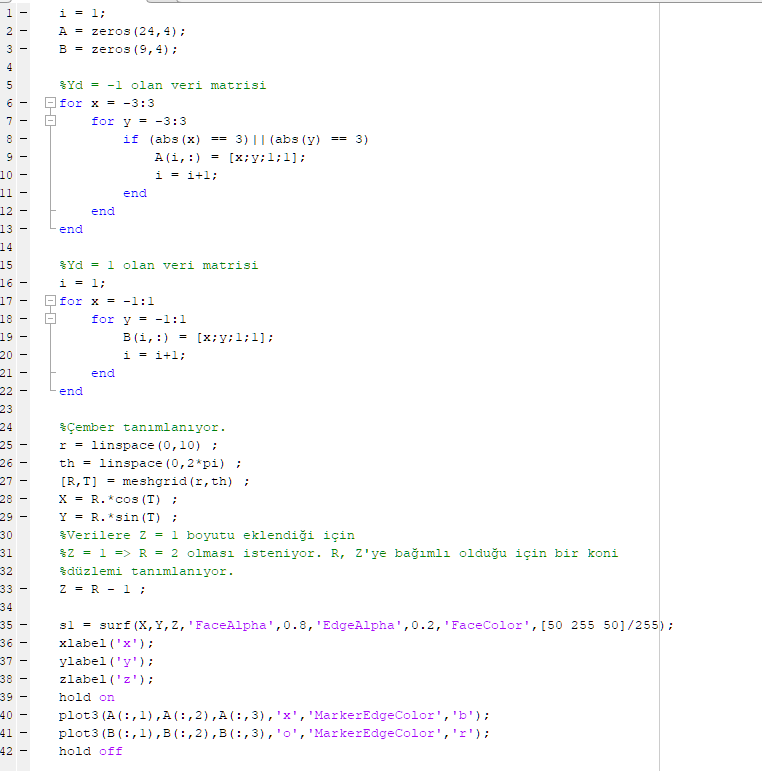
**Soru 2:**

Bu soru diğerlerinden farklı olarak MATLAB üzerinden çözülmüştür.

Veriler çizildiğinde, lineer ayrıştırılamadığı görülebiliyor. Lineer ayrıştırılabilir hale getirmek için verilere yeni bir boyut eklenir, bu şekilde üç boyutlu bir uzayda veriler bir düzlem ile ayrıştırılabilir.



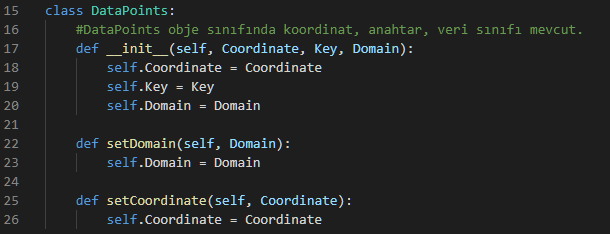
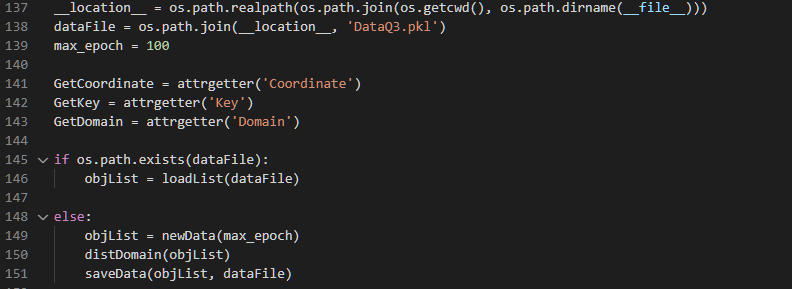
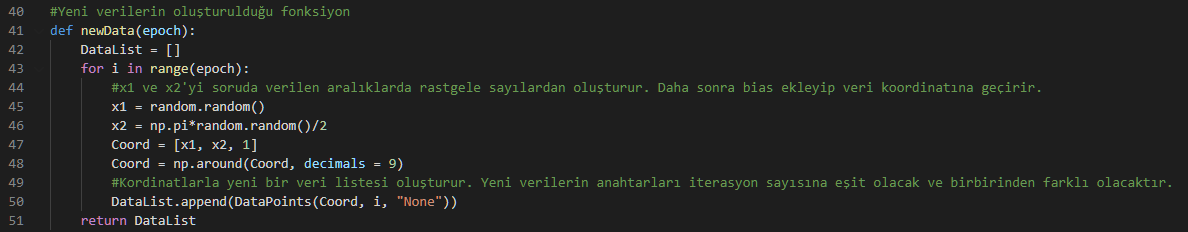
Bunun için birçok düzlem seçilebilir. Bizim seçtiğimiz düzlem, verilerle aynı z koordinatındayken r = 2 çemberi kesitine sahip koni düzlemi oldu. 

Bu işlemlerin yapıldığı MATLAB kodu aşağıdaki gibidir. 

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

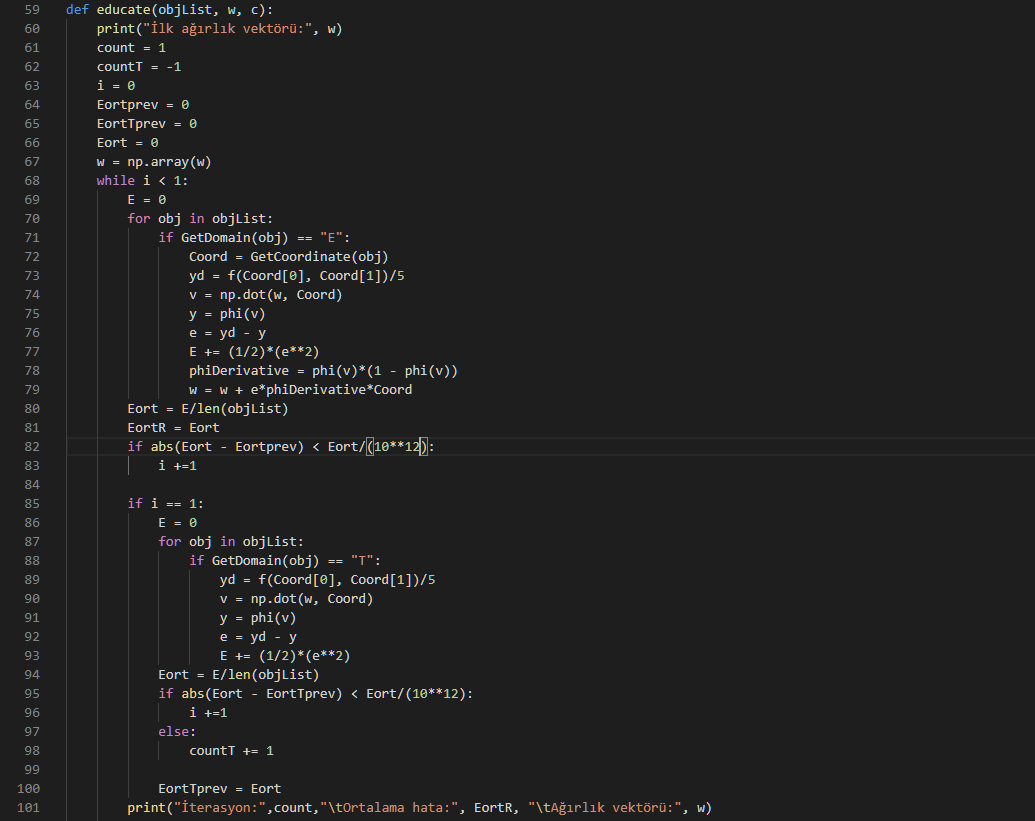
**Soru 3:**

f(x1, x2) = 3\*x1 + 2\*cos(x2) fonksiyonu x1 = [0,1] ve x = [0, π/2] aralığındayken yaklaşık olarak ifade edilmeli. Fonksiyonun minimumu f(0, π/2) = 0, maksimumu f(1, 0) = 5. g(x1, x2) = f(x1, x2)/5 fonksiyonu tanımlanıp kullanılırsa [0,1] aralığını kapsayacak logit Adaline fonksiyonu kullanarak yaklaşılabilir. (x1\*w1 + x2\*w2 + 1\*w3 = v)

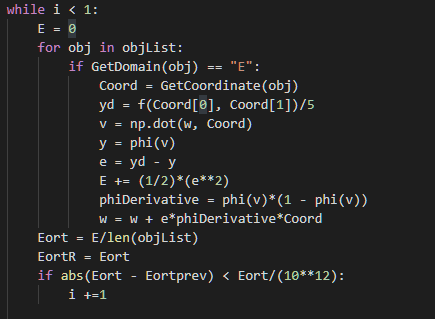
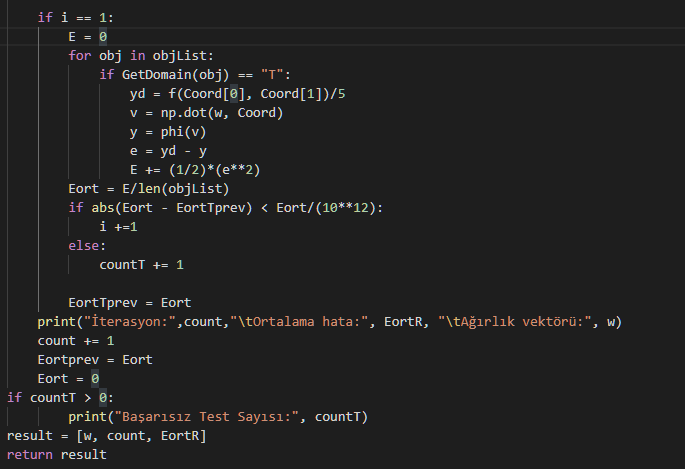
Veri setinin oluşumu için rastgele 100 farklı x1-x2 seti seçilir, rastgele bir şekilde test ve eğitim kümelerine dağıtılır ve veriler oluşturulur.   

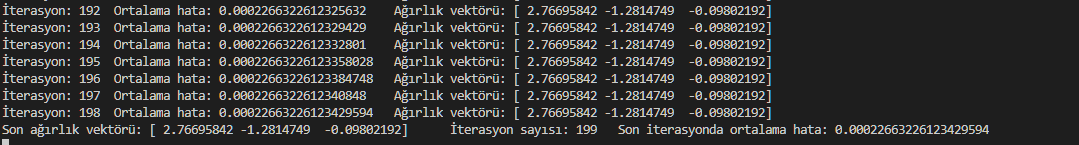
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

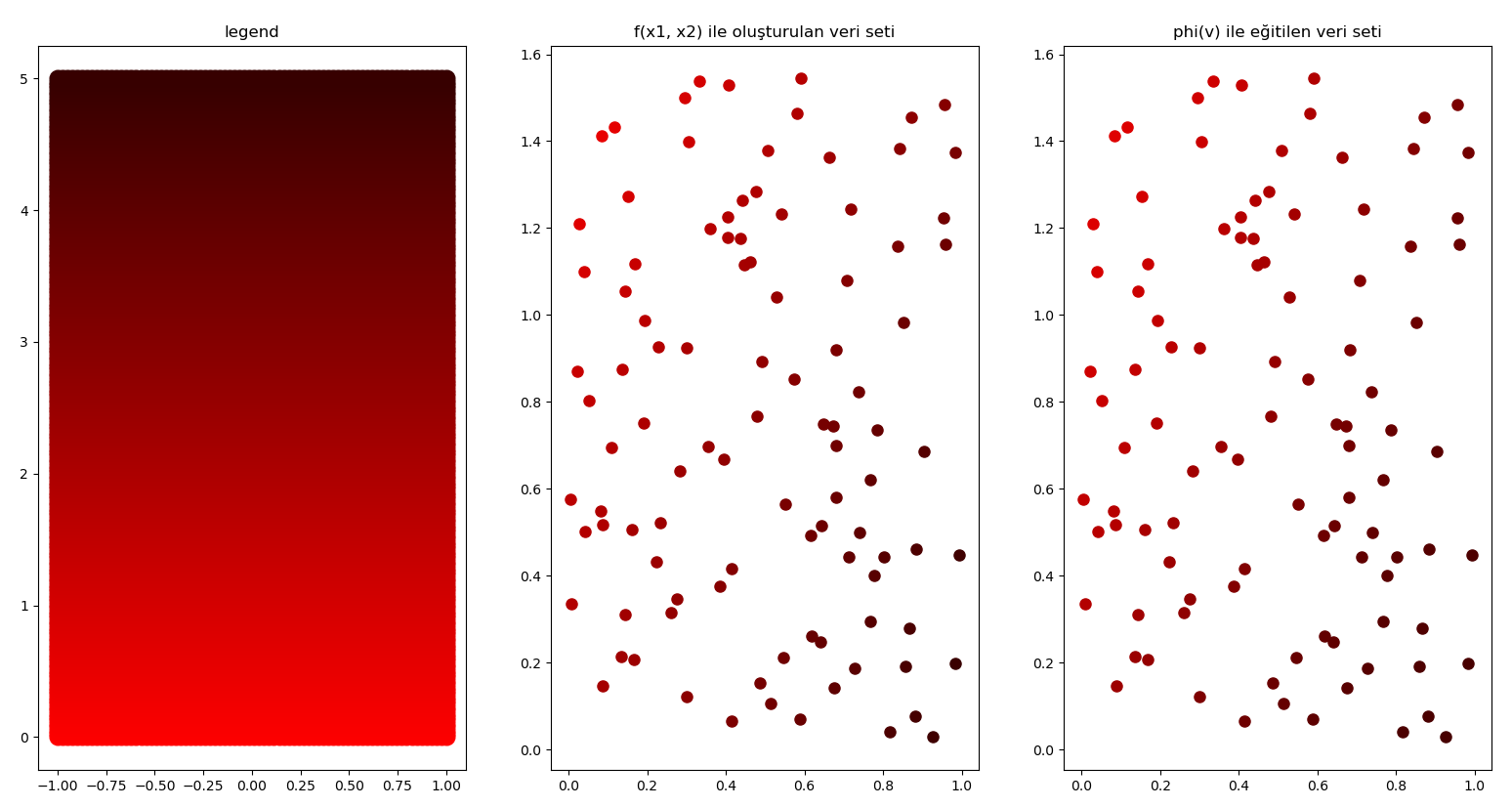
Ardından veri setiyle eğitim başlatılır. Eğitim için w1 = [1 1 1] ağırlık vektörü ve c = 0.5 eğitim hızı kullanılır. Eğitim, ortalama kare hatasının önceki iterasyonla farkı, kendisinin 10^-12 katından aşağıya düşmedikçe devam edecektir. Ayrıca bu koşul sağlandıktan sonra test döngü tekrar gerçekleşir ve test kümesinde aynı koşulun sağlanması gerekir. Bununla birlikte eğitim tamamlanır.

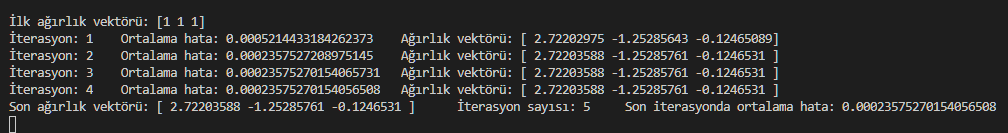
 

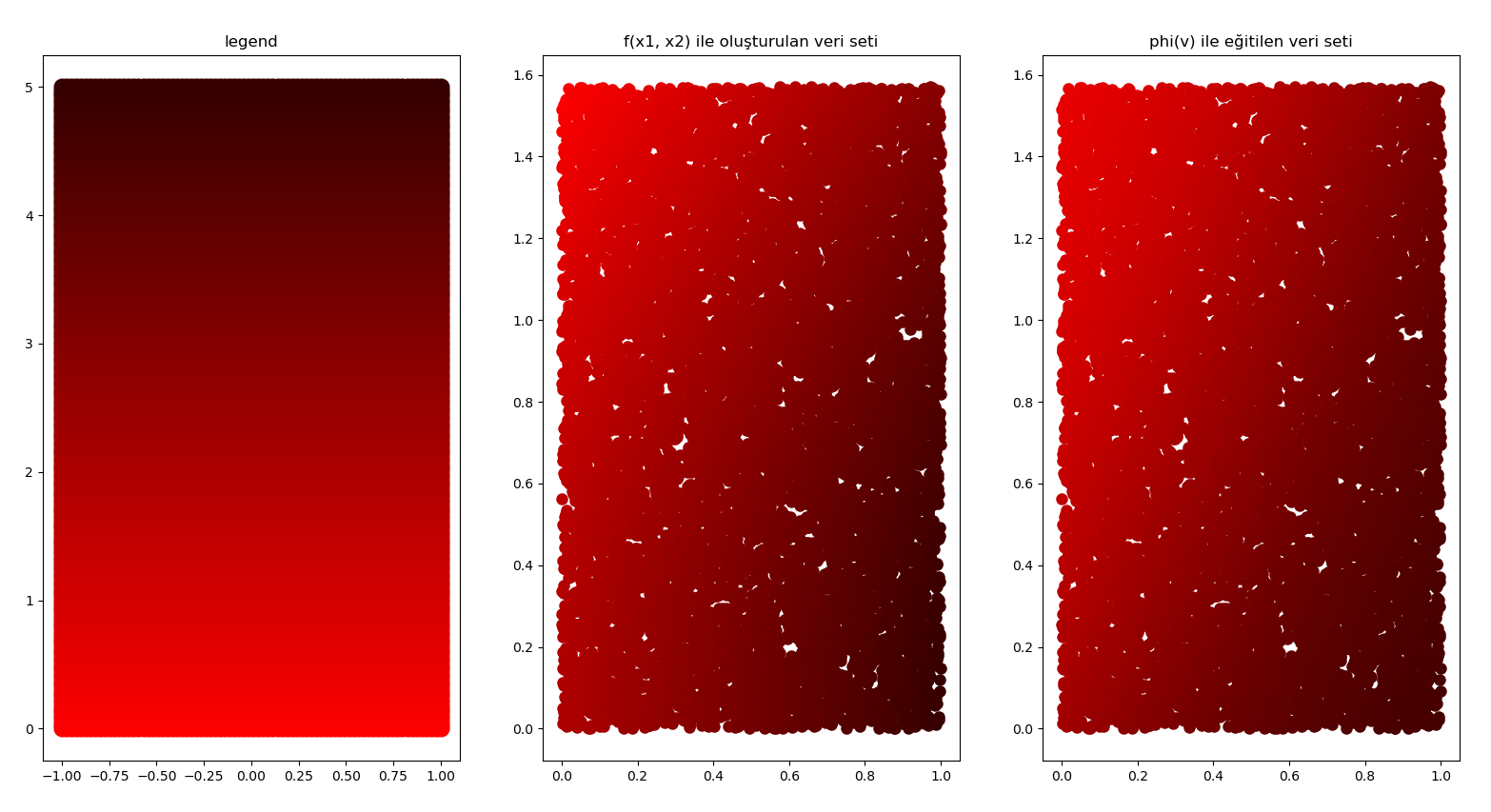
**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Eğitim sırasında her iterasyonda her veri için yd = f(x1, x2)/5 değeri hesaplanır. Daha sonra wT\*x ile v bulunup phi(v) değeri elde edilir. e = yd – y hesaplanıp toplam veri hatası E = (1/2)\*e^2, iterasyonun toplam hatası E’ye eklenir. phi(v) fonksiyonu türevi hesaplanır. Ardından bu değerler ağırlık vektörünün güncellenmesi için w = w + e\*phiDerivative\*Coord şeklinde toplanır. Daha sonra w güncellenmeden, aynı işlem test kümesinde yapılır.



**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Kod çalıştığında 199. Iterasyonda başarılı olup, yaklaşık % 0.02266 ortalama hata ile eğitimin tamamlandığı görülür. Son ağırlık vektörü w = [ 2.76695842 -1.2814749 -0.09802192] elde edilir. 

Ek olarak, sonuçlar çizdirilmek istenirse aşağıdaki şekil elde edilir. 

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**Buradan da eğitilen sistemin istenilen fonksiyona yakınlığı gözlemlenebilir. Daha iyi gözlem yapabilmek için veri sayısını artırırsak, örneğin 10000 veri alırsak eğitimin çok daha hızlı gerçekleştiğini görürüz. Aynı parametrelerle işlem yapıldığında yeni veri setiyle eğitim 5. Iterasyonda gerçekleşir. Elde edilen son ağırlık [ 2.72203588 -1.25285761 -0.1246531 ] ve yaklaşık ortalama hata oranı %0.023575 olarak gözlemlenir.

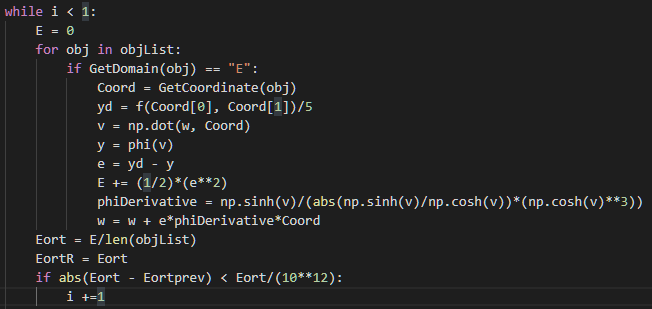
Hata oranının önceki deneyden yüksek olması, fonksiyonun yaklaşmaya çalıştığı çok daha fazla veriye sahip olmasından kaynaklanır. Buna rağmen iki fonksiyon arasındaki yakınlık, sonuçlar çizdirilince gözlemlenebilir. 

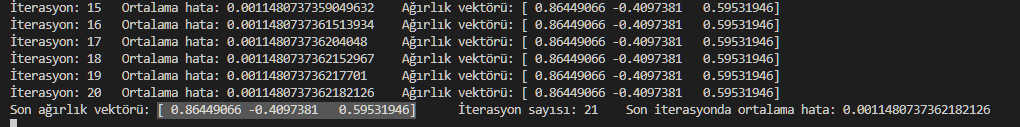
**Soru 4:**

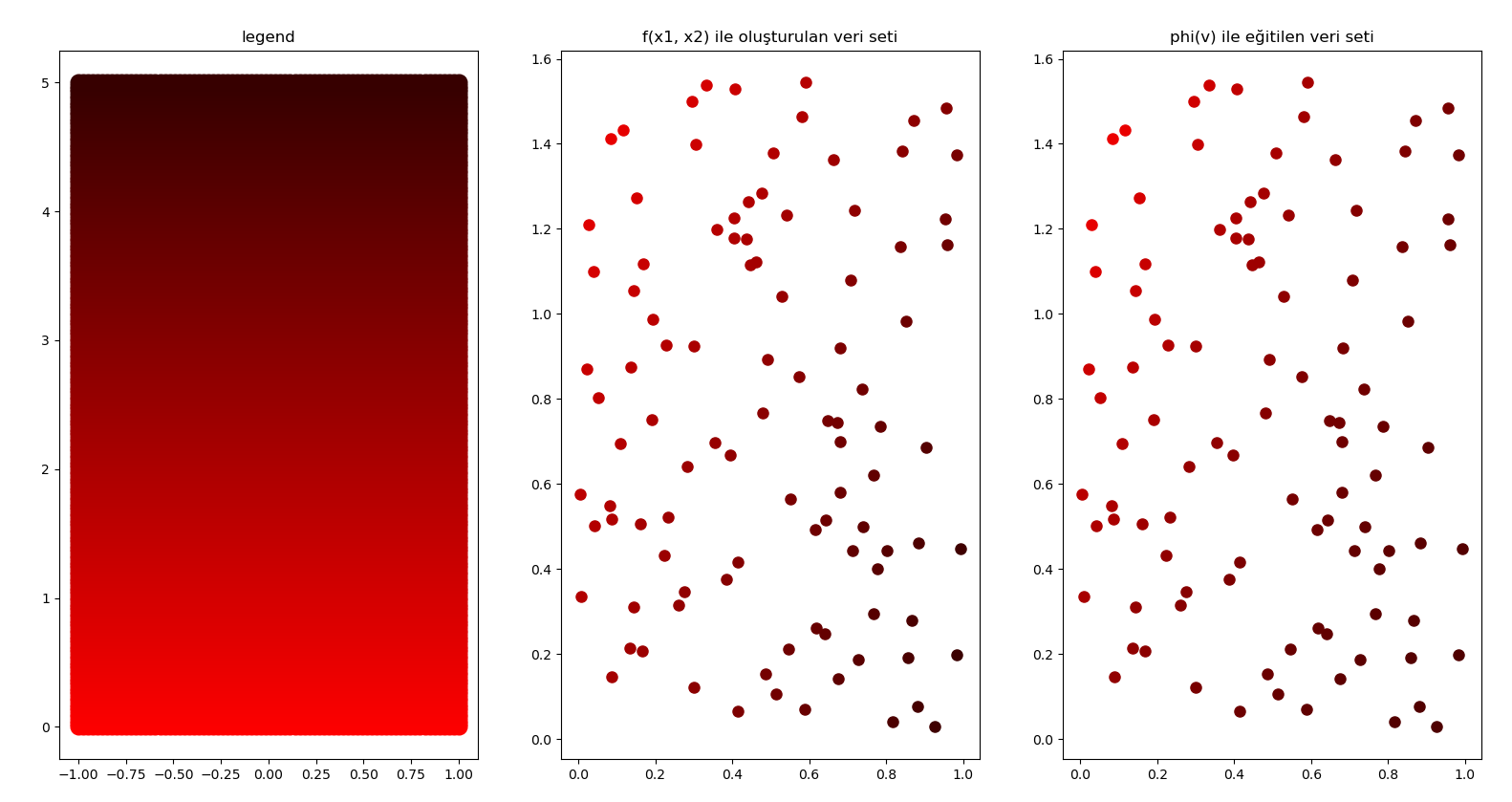
Genlikte sürekli algılayıcı ile böyle bir fonksiyonun elde edilebilmesi için, seçilecek aktivasyon fonksiyonu phi(v) de bu tipte olmalıdır. v = w1x1 + w2x2 + … + wnxn + w(n+1) olarak açılırsa, phi(v) fonksiyonunun

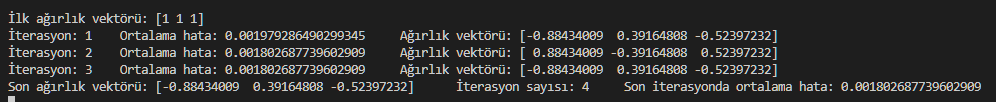
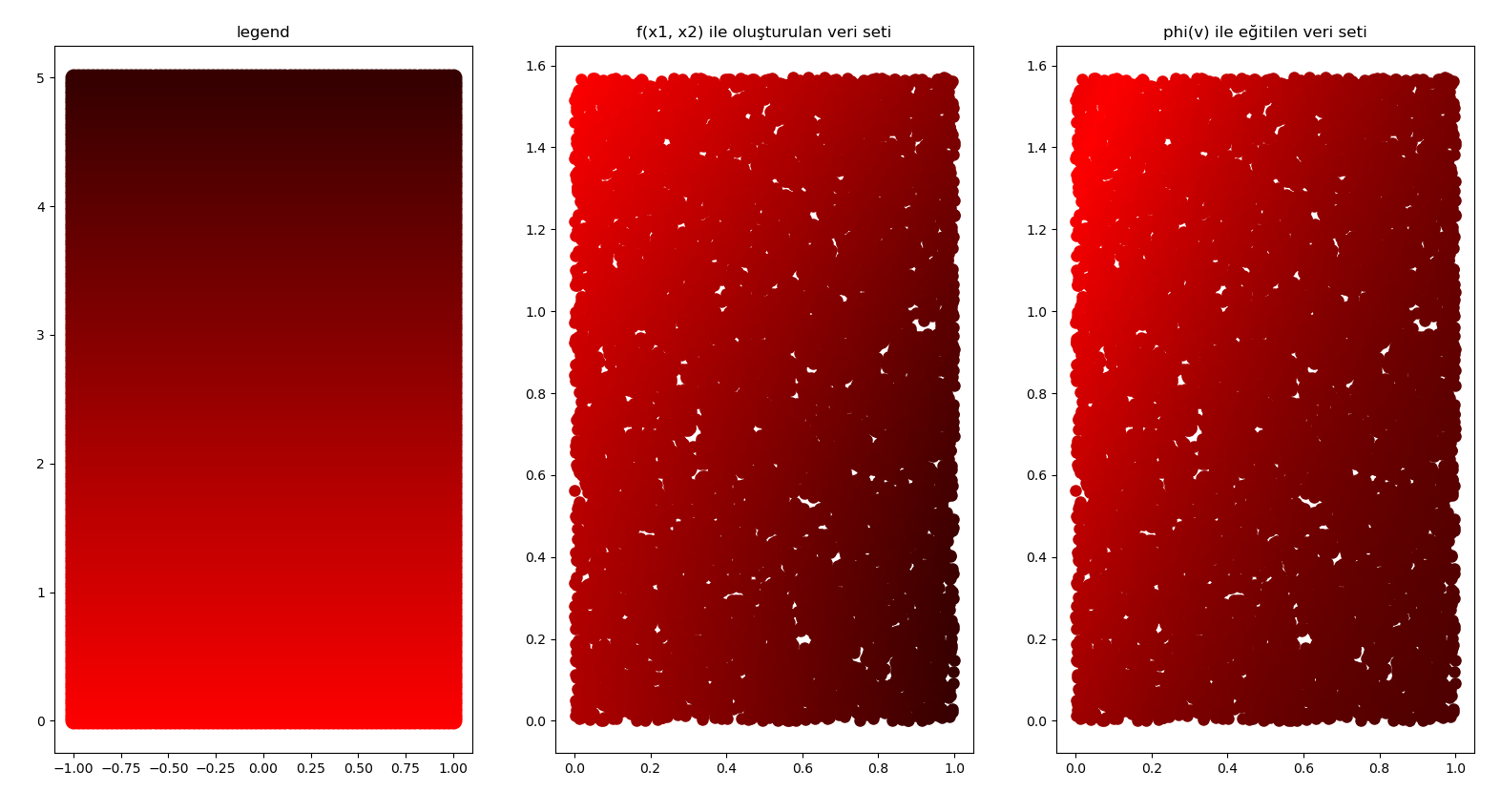
gibi bir fonksiyona benzetilmesi

Için verilen x1 ve x2 aralığı için aktivasyon fonksiyonu kullanılabilir. 



Bu şekilde bir eğitim gerçekleştirildiğinde önceki sorudaki 100 verilik veri seti ile 21 iterasyonda yaklaşık %0.1148 hata ile eğitim tamamlanır ve son ağırlık vektörü [ 0.86449066 -0.4097381 0.59531946] bulunur. Elde edilen veri çizdirildiğinde, birbirine yakın çizimler ortaya çıkar. 



Eğitim 10000 verilik veri seti ile gerçekleştirildiğinde ise 4. Iterasyonda yaklaşık %0.1803 hata oranı ile eğitim tamamlanır ve son ağırlık vektörü [-0.88434009 0.39164808 -0.52397232] bulunur. Bu veri çizdirilince yine birbirine yakın şekiller ortaya çıkar. 

Sonuca bakacak olursak w vektörü yaklaşık olarak ilk veri setiyle

**w = [ 0.8645 -0.4097 0.5953] = [w11 w12 w13]**

ikinci veri setiyle

**w = [-0.8843 0.3917 -0.524] = [w21 w22 w23]**

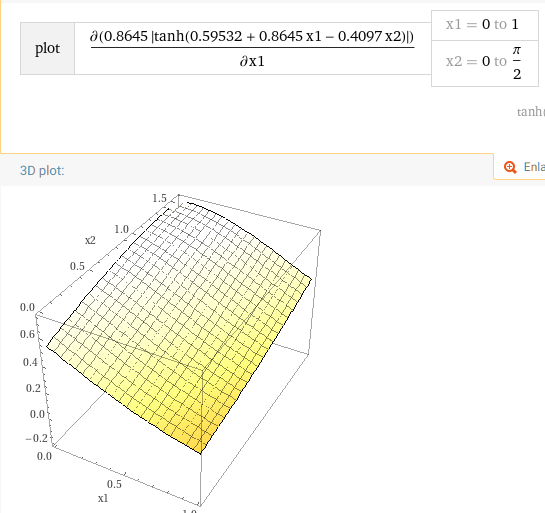
bulunmuştur. Fonksiyona koyulduğunda

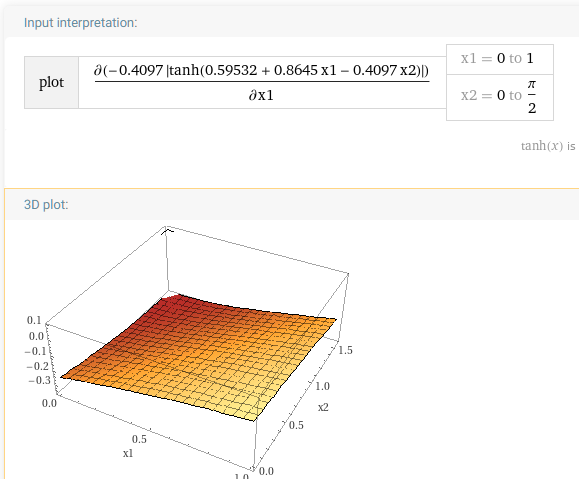
phi(v) = phi(w1\*x1 + w2\*x2 + w3\*1) = c1\*x1 + c2\*x2 + c3\*x1\*x2 = f(x1, x2) elde edilir.

İki tarafında;

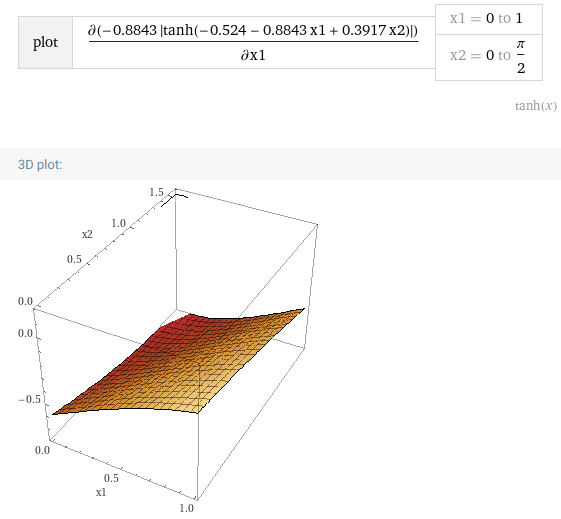
x1’e göre türevi alındığında phi’(v)\*w1 = c1 + c3\*x2

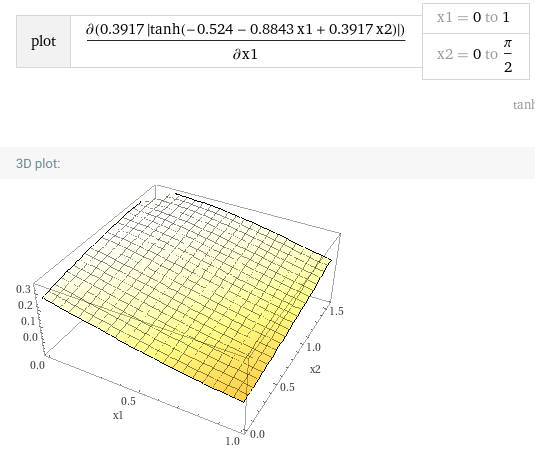
x2’ye göre türevi alındığında phi’(v)\*w2 = c2 + c3\*x1 olacaktır. c1 ve c2 x1 ve x2’ye bağımlılardır.

Wolfram kullanılarak ilk veri seti için çizildiğinde c11 + c13\*x2: 

c12 + c13\*x1: 

İkinci veri seti için c21 + c23\*x1:



c22 + c23\*x2: 

Kullanılan kaynak ve kütüphaneler:

* Python
  + Numpy
  + Matplotlib
  + random
  + math
  + json
  + pickle
  + scipy
* WolframAlpha
* Stackoverflow

Kullanılan kodlara, ödev dosyasına ek olarak [**https://github.com/meserbetcioglu/ehb420hw1.git**](https://github.com/meserbetcioglu/ehb420hw1.git)linkini kullanarak erişebilirsiniz.