Yapay Sinir Ağları ve Destek Vektör Makineleri ile İkili Sınıflandırma

Alican Tunç
Veri Bilimi ve Büyük Veri Yüksek Lisans Programı
Yıldız Teknik Üniversitesi
Istanbul, TÜRKİYE:
can.tunc1@std.yildiz.edu.tr

***Abstract*—Bu çalışmada, iki sınıflı bir veri kümesi üzerinde yapay sinir ağları (YSA) ve destek vektör makineleri (DVM) yöntemlerinin sınıflandırma başarıları karşılaştırılmıştır. Veri kümesi, %60 eğitim, %20 doğrulama ve %20 test oranlarında bölünerek her iki yöntem için aynı koşullar altında analiz edilmiştir. DVM modellerinde lineer, polinomsal ve Gaussian RBF çekirdek fonksiyonları kullanılmış; YSA'da ise bir, iki ve üç gizli katmanlı ağlar değerlendirilmiştir. Modeller, doğruluk, precision, recall ve F1-score metrikleriyle karşılaştırılmış, en başarılı sonuçlar elde edilerek analiz edilmiştir. Sonuçlar, DVM'nin küçük veri setlerinde daha istikrarlı olduğunu, ancak YSA'nın doğru hiperparametre seçimiyle daha yüksek performans gösterebildiğini ortaya koymuştur.**

Keywords—Yapay sinir ağları, destek vektör makineleri, sınıflandırma, doğruluk analizi,

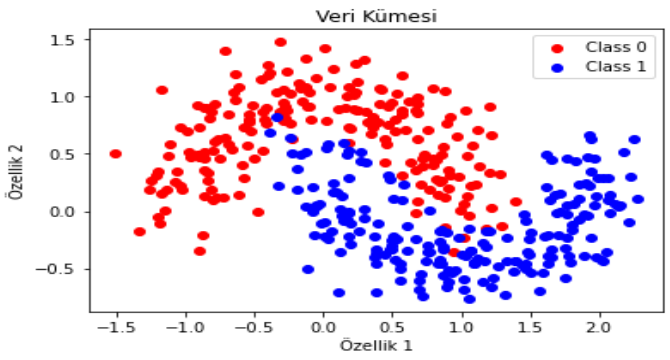
I. GIRIŞ

Makine öğrenmesi alanında, sınıflandırma problemleri farklı yöntemlerle çözülebilir. Bu çalışmada, popüler iki yöntem olan YSA ve DVM'nin başarımı incelenmiştir. İkili sınıflandırma için scikit-learn kütüphanesindeki make_moons fonksiyonu ile 400 örnekten oluşan bir veri kümesi oluşturulmuş ve belirli bir gürültü seviyesi eklenmiştir. Bu veri kümesi, YSA ve DVM'nin performansını adil bir şekilde değerlendirmek amacıyla eğitim, doğrulama ve test setlerine ayrılmıştır. Çalışmada kullanılan metrikler doğruluk (accuracy), precision, recall ve F1-score olarak seçilmiştir.

II. DENEYSEL ANALİZ

A. Veri Seti Dağılımı

400 örnekten oluşan veri kümesi, scikit-learn kütüphanesindeki make_moons fonksiyonu kullanılarak oluşturulmuş ve 0.2 gürültü eklenmiştir. Veri kümesi %60 eğitim, %20 doğrulama ve %20 test oranında ayrılmıştır. Veri setinin sınıf dağılımı ve özellikleri Şekil 1'de görselleştirilmiştir. Örneklerin iki sınıfa dağılımını görmek için iki sınıfa ait örnekleri iki farklı renk ile göstererek örnekler çizilmiştir



Şekil 1: Veri kümesinin görselleştirilmesi

B. *Yöntem*

Bu çalışmada, hem DVM hem de YSA modelleri aşağıdaki şekilde eğitilmiştir:

C. *3.1 Destek Vektör Makineleri (DVM):*

DVM yönteminde üç farklı çekirdek fonksiyonu (lineer, polinomsal ve Gaussian RBF) kullanılmıştır. Çekirdek fonksiyonlarının başarımı, hiper parametre optimizasyonu ile artırılmıştır. En uygun parametreler, doğrulama seti doğruluklarına göre seçilmiş ve karar sınırları görselleştirilmiştir.

D. *3.2 Yapay Sinir Ağları (YSA):*

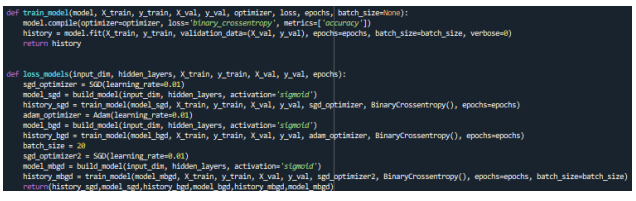
YSA, sırasıyla bir, iki ve üç gizli katmanlı olarak yapılandırılmıştır. Aktivasyon fonksiyonu olarak sigmoid kullanılmış, kayıp fonksiyonu olarak binary cross-entropy seçilmiştir. Öğrenme sürecinde stokastik gradyan inişi (SGD), batch gradyan inişi ve mini-batch gradyan inişi yöntemleri uygulanmıştır. Eğitim ve doğrulama setlerindeki kayıpların epoch bazında değişimi incelenmiştir. Ayrıca en uygun parametreler, doğrulama seti doğruluklarına göre seçilmiş ve karar sınırları görselleştirilmiştir.

E. *3.2.1 Yapay Sinir Ağları için İşlemler*

YSA, için sırasıyla bir, iki ve üç gizli katmanlı olarak ve aktivasyon fonksiyonu olarak sigmoid kullanarak bir yapay nöron ağı elde ediyoruz. Bunun için Tensorflow kütüphanesi kullanıyoruz. Build_model fonksiyonu ile input_dim, hidden_layers ve activation belirterek modelimizi oluşturuyoruz.

```python
def build_model(input_dim, hidden_layers, activation='sigmoid'):
    """Modeli oluşturma fonksiyonu"""
    model = tf.keras.Sequential()

    # İlk katman olarak Input kullanmak
    model.add(Input(shape=(input_dim,)))

    # Gizli katmanları ekleme
    for units in hidden_layers:
        model.add(layers.Dense(units, activation=activation))

    # Çıktı katmanı
    model.add(layers.Dense(1, activation='sigmoid'))

    return model
```

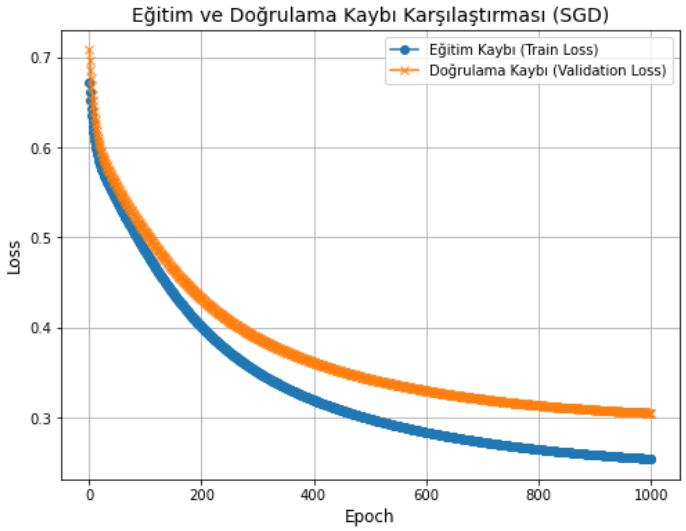
Şekil 2: Nöron ağ modelinin oluşturulması



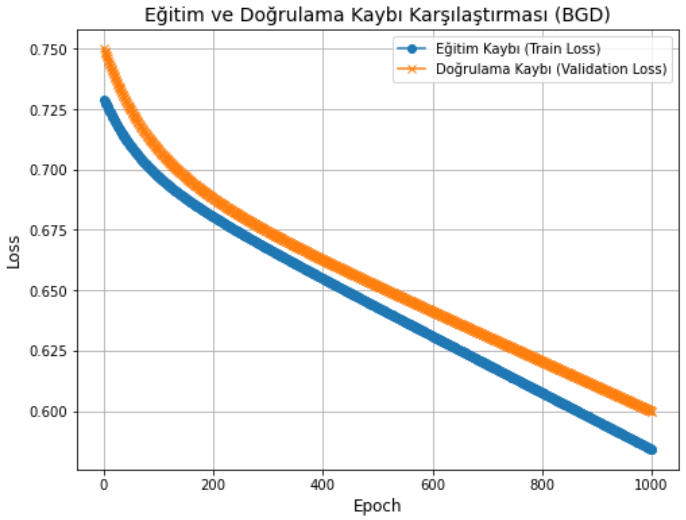
Şekil 3: Nöron ağ modelinin eğitimi ve train-valid loss hesabı

Sonrasında modelimizi eğitmek için train fonksiyonu ve eğitimdeki çıktıları görmek için kayıp değerlerini görmek için ayrıca bir fonksiyon kullanacağız. Bunun için yine tensorflow kütüphanesi kullanıyoruz.

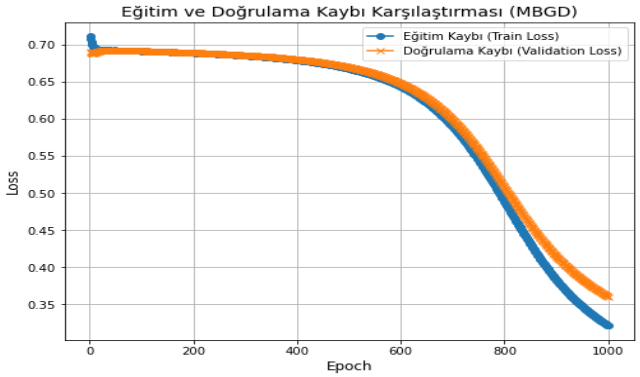
İlk olarak başlangıçtaki parametreleri bilmediğimiz random bir şekilde vererek eğitime başlıyoruz. Birkaç denemeden sonra optimal bir sonuç için ilk katman için 10 adet nöron iki katmanlı için 20 nörön 10-10 şekilde olacak şekilde ve 3 katmalı 30 nörondan 10-10-10 şekilde üç model eğitip train-valid loss değerlerini 1000 epoch için bakacak olursak aşağıdaki sonuçları elde edeceğiz.



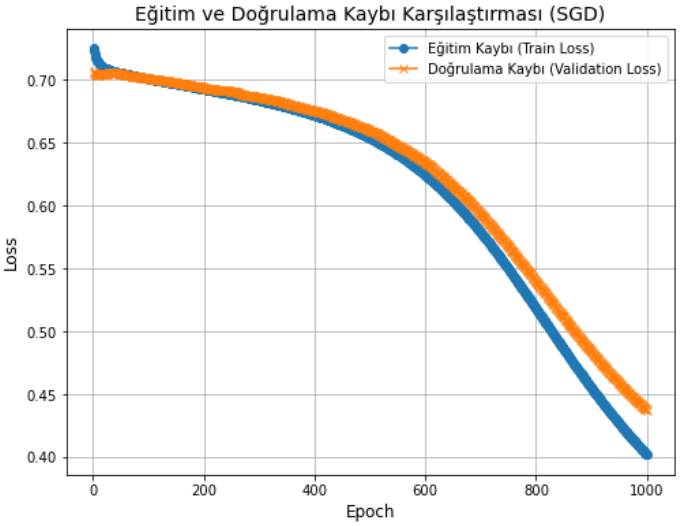
Şekil 4: Tek katmanlı 10 nörona sahip ağın 1000 epochs ve sgd optimizer için loss grafiği



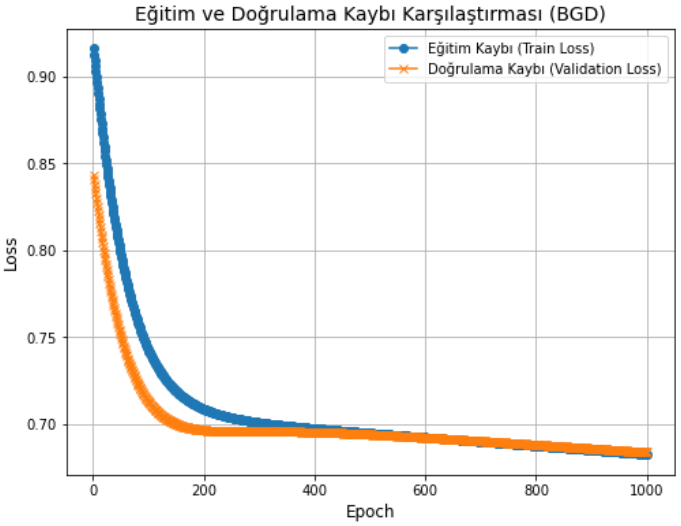
Şekil 5: Tek katmanlı 10 nörona sahip ağın 1000 epochs ve bgd optimizer için loss grafiği



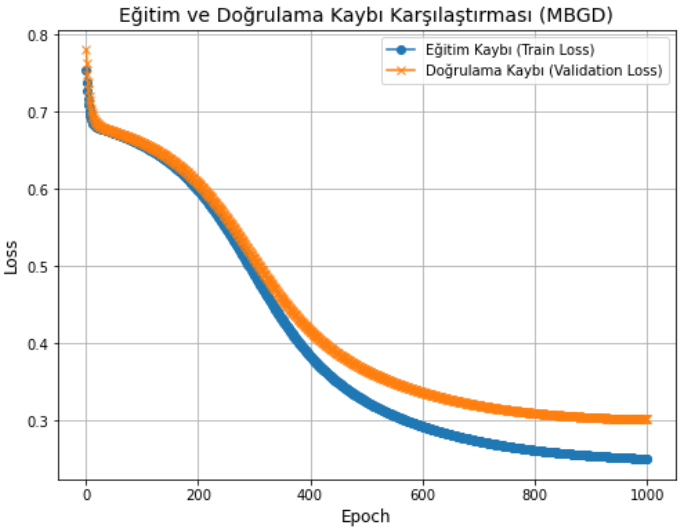
Şekil 6: Tek katmanlı 10 nörona sahip ağın 1000 epochs ve mbgd optimizer için loss grafiği



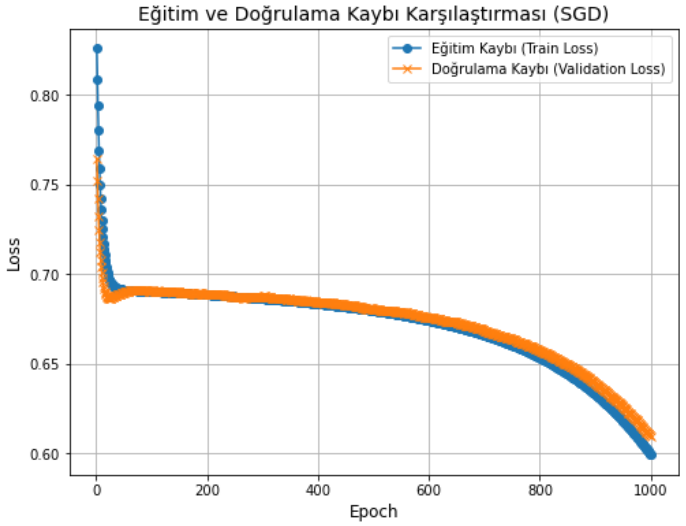
Şekil 7: İki katmanlı 10-10 nörona sahip ağın 1000 epochs ve sgd optimizer için loss grafiği



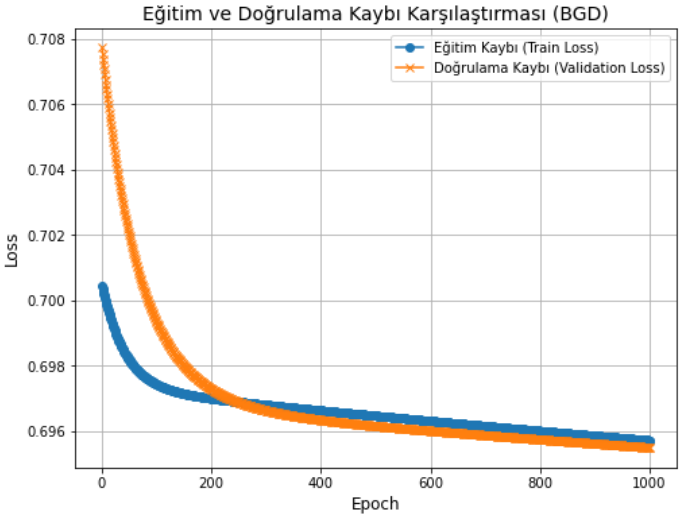
Şekil 8: İki katmanlı 10-10 nörona sahip ağın 1000 epochs ve bgd optimizer için loss grafiği



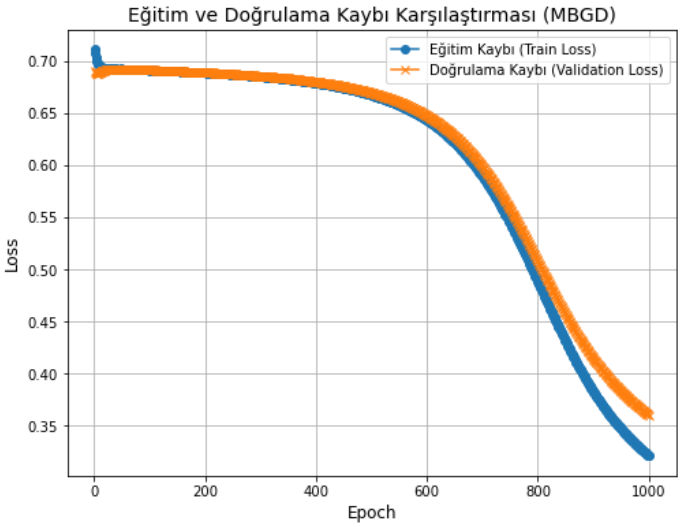
Şekil 9: İki katmanlı 10-10-10 nörona sahip ağın 1000 epochs ve mbgd optimizer için loss grafiği



Şekil 9: Üç katmanlı 10-10-10 nörona sahip ağın 1000 epochs ve sgd optimizer için loss grafiği

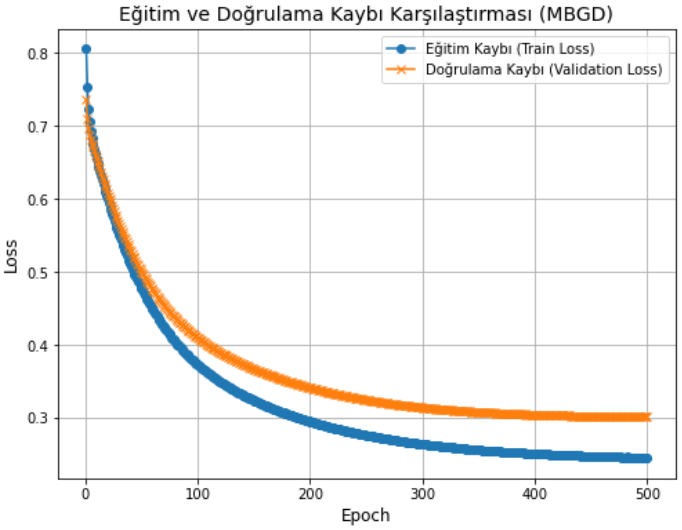


Şekil 10: Üç katmanlı 10-10-10 nörona sahip ağın 1000 epochs ve bgd optimizer için loss grafiği



Şekil 11: Üç katmanlı 10-10-10 nörona sahip ağın 1000 epochs ve bgd optimizer için loss grafiği

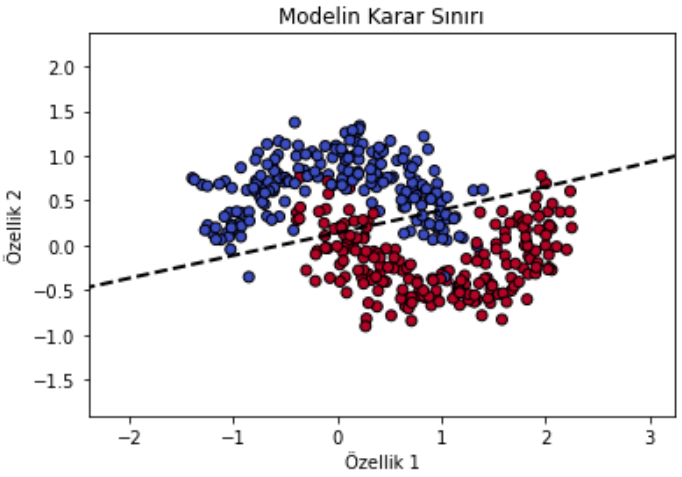
Görüldüğü üzere her üç katmanlı ve her üç yöntem içinde mini-batch gradient descent ile sgd bgd'ye göre daha iyi ve birbirine yakın sonuçlar veriyor. Grafiklere göre başarı oranını arttırmak için nöron sayısını artırmamız gerekiyor. Tek bir katmanda 20 adet nöronla 500 epoch sayısı için valid-loss grafiği ve başarı oranları şekil 12'deki gibidir.



Şekil 12: Tek katmanlı 20 nörona sahip ağın 500 epochs ve mbgd optimizer için loss grafiği

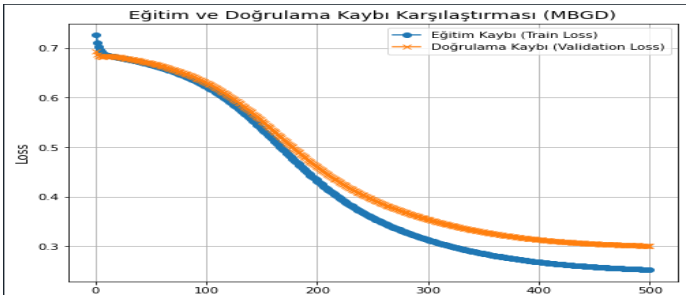
```
   Eğitim Seti - Doğruluk: 0.89, Precision: 0.90, Recall: 0.88, F1-Score: 0.89
   Doğrulama Seti - Doğruluk: 0.88, Precision: 0.86, Recall: 0.86, F1-Score: 0.86
   Test Seti - Doğruluk: 0.85, Precision: 0.83, Recall: 0.91, F1-Score: 0.87
   Confusion Matrix - Eğitim Seti:
   [[108  12]
[ 14 106]]

   Confusion Matrix - Doğrulama Seti:
   [[39  5]
[ 5 31]]

   Confusion Matrix - Test Seti:
   [[28  8]
[ 4 40]]
```

Şekil 13: Tek katmanlı 20 nörona sahip ağın 500 epochs ve mbgd optimizer için çıktıları



Şekil 14: Tek katmanlı 20 nörona sahip ağın 500 epochs ve mbgd optimizer için çıktıları

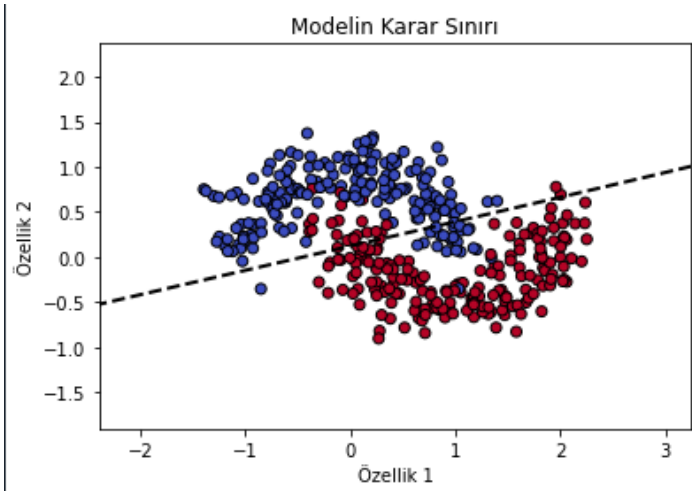
Ve çıktı sonuçları şekil 13 ve 14'deki gibi görünüyor.Bunu yaparken batch size değerini 10 olarak optimize ettik. Benzer bir işlemi iki katmanlı ağ için optimize edersek yine toplam 20 adet nöronla 10-10 olacak şekilde 500 epochs için çalıştırırsak.



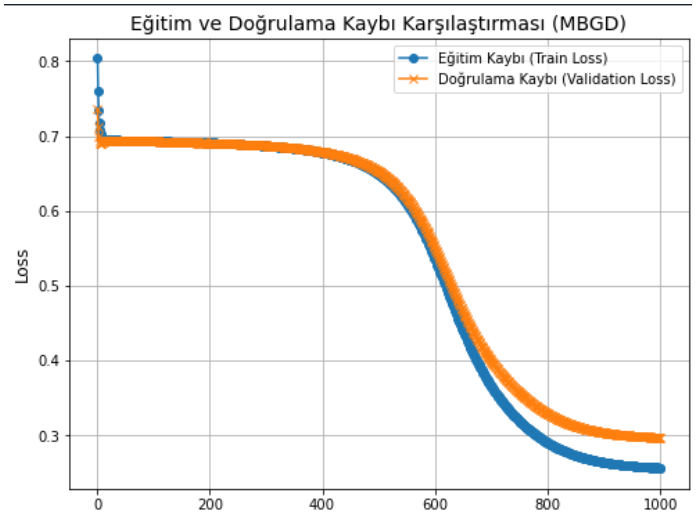
Şekil 15: İki katmanlı 20(10-10) nörona sahip ağın 500 epochs ve mbgd optimizer için loss grafiği

```
   Eğitim Seti - Doğruluk: 0.87, Precision: 0.82, Recall: 0.94, F1-Score: 0.88
   Doğrulama Seti - Doğruluk: 0.85, Precision: 0.77, Recall: 0.94, F1-Score: 0.85
   Test Seti - Doğruluk: 0.85, Precision: 0.81, Recall: 0.95, F1-Score: 0.88
   Confusion Matrix - Eğitim Seti:
   [[ 95  25]
[  7 113]]

   Confusion Matrix - Doğrulama Seti:
   [[34 10]
[ 2 34]]

   Confusion Matrix - Test Seti:
   [[26 10]
[ 2 42]]
```

Şekil 16: İki katmanlı 20(10-10) nörona sahip ağın 500 epochs ve mbgd optimizer optimizer için çıktıları



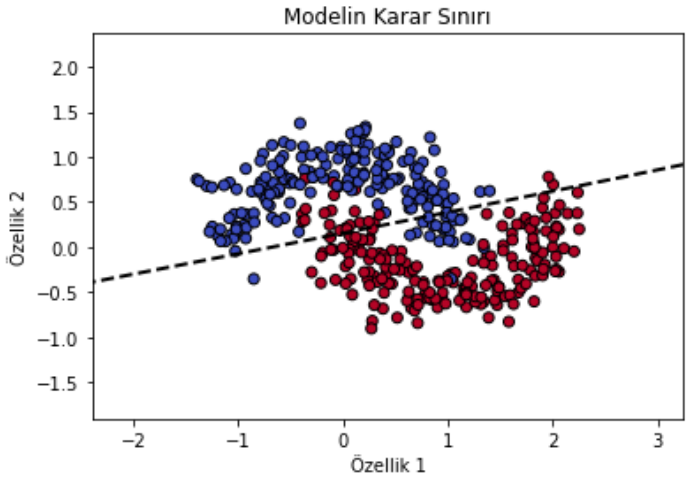
Şekil 17: İki katmanlı 20(10-10) nörona sahip ağın 500 epochs ve mbgd optimizer optimizer için çıktıları



Şekil 18: Üç katmanlı 30(10-10-10) nörona sahip ağın 500 epochs ve mbgd optimizer optimizer için çıktıları

```
   Eğitim Seti - Doğruluk: 0.89, Precision: 0.90, Recall: 0.88, F1-Score: 0.89
   Doğrulama Seti - Doğruluk: 0.88, Precision: 0.86, Recall: 0.86, F1-Score: 0.86
   Test Seti - Doğruluk: 0.86, Precision: 0.84, Recall: 0.93, F1-Score: 0.88
   Confusion Matrix - Eğitim Seti:
   [[108  12]
[ 14 106]]

   Confusion Matrix - Doğrulama Seti:
   [[39  5]
[ 5 31]]

   Confusion Matrix - Test Seti:
   [[28  8]
[ 3 41]]
```

Şekil 19: Üç katmanlı 30(10-10-10) nörona sahip ağın 500 epochs ve mbgd optimizer optimizer için çıktıları



Şekil 20: Üç katmanlı 30(10-10-10) nörona sahip ağın 500 epochs ve mbgd optimizer optimizer için çıktıları

Elimizdeki sonuçlara bakarken tek bir katmanda 20 adet nöron ile yaptığımız işlemin bizim için en optimal sonucu vereceğini söyleyebiliriz. Nöron sayısını arttırmak daha iyi sonuç vermedi. Fakat epoch sayısı artırmak bazı durumlar için bir tık daha iyi sonuçlar verebilir. Loss grafiği çoğunlukla düz bir çizgide ilerliyor bu yüzden devam etmenin çok bir anlamı olmadığını hem görüyor hem denemelerimiz bize bunu söylüyor. Fakat eğer adam optimizer kullanmamıza izin olsaydı hem daha yüksek sonuçları çok daha az nöron ve epoch sayısı ile daha yüksek bir başarıya ulaşabilirdik. Elimizde gpu ve kısıtlı metodlar ile uşabildiğimiz en optimal sonuçlar bunlardır.

F. *3.2.1 Destek Vektör Makineleri (DVM) için işlemler*

DVM yönteminde üç farklı çekirdek fonksiyonu (lineer, polinomsal ve Gaussian RBF) kullanılmıştır. Sklearn kütüphanesi kullanarak dvm modelimizi oluşturuyoruz ve ilgili metrikler ile ölçümlerimizi yapıyoruz. Şekil 21'de ilgili kod satırı belirtilmiştir.

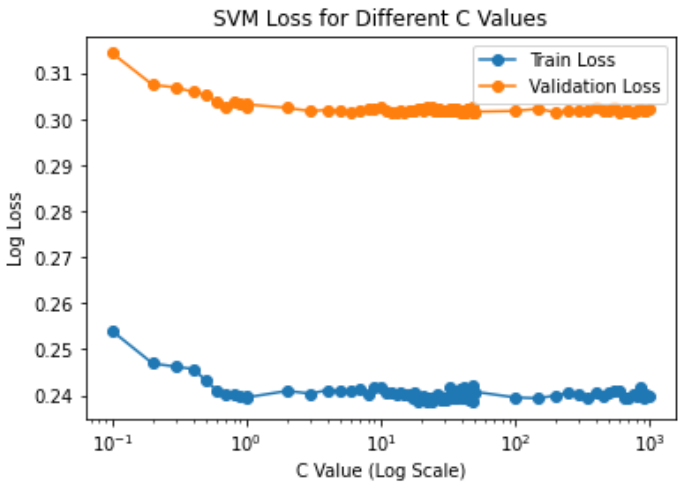
```python
def train_svm(X_train, y_train, kernel, C, **kwargs):
    """SVM modelini eğit."""
    model = SVC(kernel=kernel, C=C,**kwargs)
    model.fit(X_train, y_train)
    return model
```

Şekil 21: DVM oluşturmak ve eğitmek için ilgili kod satırı

Burada bizim ilgileneceğimiz olay C ve kernel parametreleri üzerinden değişiklik yapmak olacak. Bunun için Şekil 22'de 0.1-1 0.1 artışla sonra 1-50 arasında 1 artışla 50-1000 arasında 50 artışla bir c listesi hazırlayıp her kernel için bu değerlerin çıktılarını inceleyeceğiz.

```python
def get_C_values():
    """C değerlerini belirler: 0.1'den 1'e kadar 0.1 artarak, sonra 1'den 50'ye
    C_values_1 = np.arange(0.1, 1.1, 0.1)    # 0.1'den 1'e kadar 0.1 artarak
    C_values_2 = np.arange(1, 51, 1)         # 1'den 50'ye kadar 1 artarak
    C_values_3 = np.arange(50, 1001, 50)     # 50'den 1000'e kadar 50 artarak
    return np.concatenate([C_values_1, C_values_2, C_values_3])
```

Şekil 22: C değerler listesi

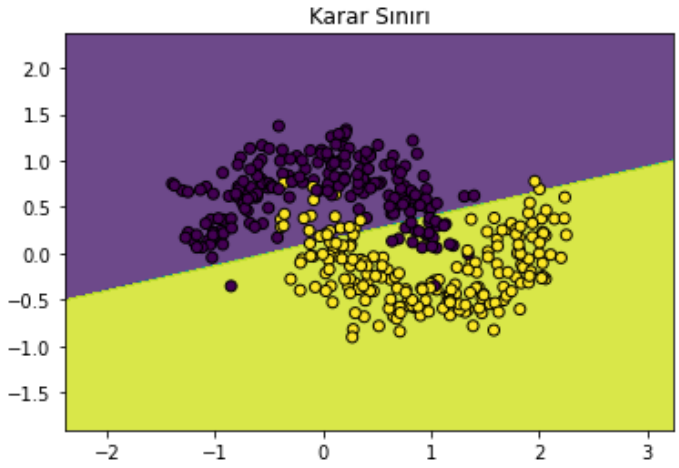


Şekil 23: Linear kernel için C ile loss grafiği

Şekil 23'ü incelediğimiz zaman en iyi C değerinin 10 üzeri 0 civarlarında olduğunu görüyoruz.

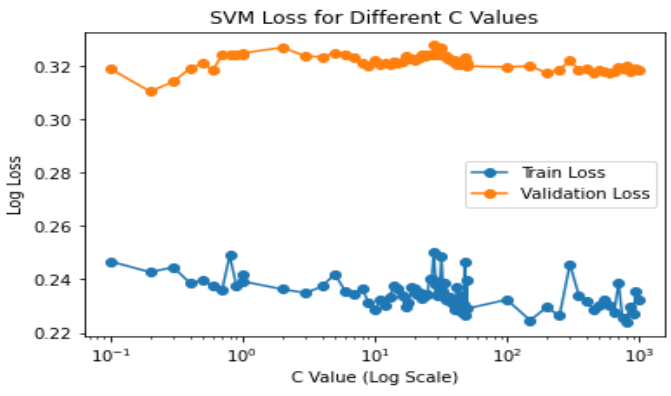
```
   Eğitim Seti - Doğruluk: 0.89, Precision: 0.90, Recall: 0.88, F1-Score: 0.89
   Doğrulama Seti - Doğruluk: 0.88, Precision: 0.86, Recall: 0.86, F1-Score: 0.86
   Test Seti - Doğruluk: 0.85, Precision: 0.83, Recall: 0.91, F1-Score: 0.87

   Confusion Matrix - Eğitim Seti:
   [[108  12]
[ 14 106]]

   Confusion Matrix - Doğrulama Seti:
   [[39  5]
[ 5 31]]

   Confusion Matrix - Test Seti:
   [[28  8]
[ 4 40]]
```

Şekil 24: Linear kernel best C ile çıktıları



Şekil 25: Linear kernel karar sınırı

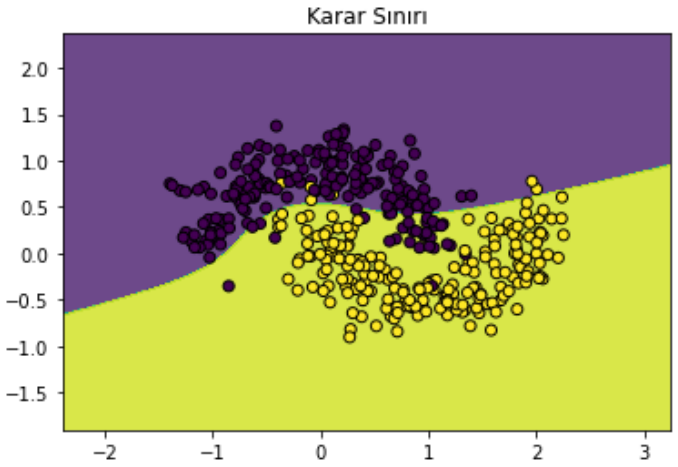
Şekil 24-25'i incelediğimiz zaman lineer kernelin yapay nöron ağlarına benzer bir sonucu daha hızlı bir şekilde benzer doğruluk oranı ve daha az gpu kullanarak verdiğini görüyoruz.



Şekil 26: Poly kernel için C ile loss grafiği

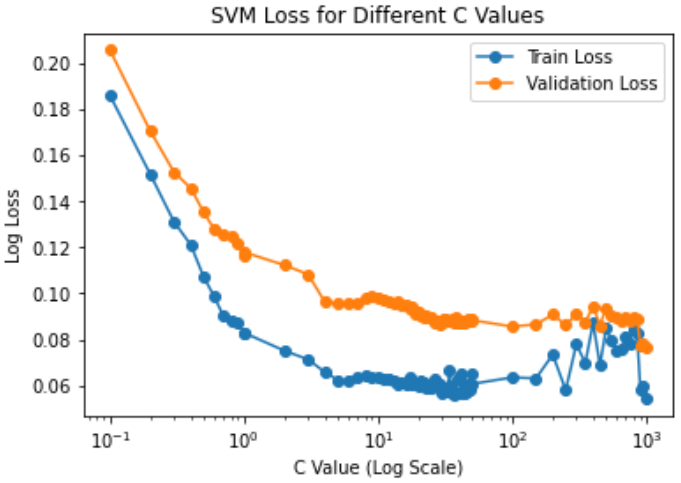
```
   Eğitim Seti - Doğruluk: 0.93, Precision: 0.89, Recall: 0.97, F1-Score: 0.93
   Doğrulama Seti - Doğruluk: 0.90, Precision: 0.85, Recall: 0.94, F1-Score: 0.89
   Test Seti - Doğruluk: 0.89, Precision: 0.84, Recall: 0.98, F1-Score: 0.91

   Confusion Matrix - Eğitim Seti:
   [[105  15]
[  3 117]]

   Confusion Matrix - Doğrulama Seti:
   [[38  6]
[ 2 34]]

   Confusion Matrix - Test Seti:
   [[28  8]
[ 1 43]]
```

Şekil 27: Poly kernel ile best C çıktıları



Şekil 28: Poly kernel karar sınırı

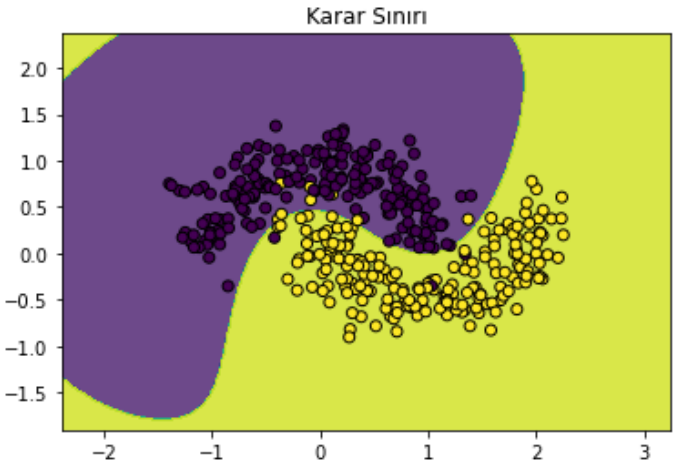
Şekil 26-27-28'i incelediğimiz zaman poly kernelin en iyi c değerinin 10 üzeri 2 civarlarında ve bu değere göre hesaplamalar yapıldığında lineere göre daha iyi sonuç verdiğini görüyoruz. Bunun sebebi ilgili veri kümesinin ayrılması için buna daha uygun olması.



Şekil 29: RBF kernel için C ile loss grafiği

```
   En İyi C Değeri: 5
   Eğitim Seti - Doğruluk: 0.98, Precision: 0.98, Recall: 0.97, F1-Score: 0.98
   Doğrulama Seti - Doğruluk: 0.97, Precision: 0.97, Recall: 0.97, F1-Score: 0.97
   Test Seti - Doğruluk: 0.99, Precision: 1.00, Recall: 0.98, F1-Score: 0.99

   Confusion Matrix - Eğitim Seti:
   [[118   2]
[  3 117]]

   Confusion Matrix - Doğrulama Seti:
   [[43  1]
[ 1 35]]

   Confusion Matrix - Test Seti:
   [[36  0]
[ 1 43]]
```

Şekil 30: RBF kernel için best C ile çıktıları



Şekil 31: Rbf kernel karar sınırı

Şekil 29-30-31'i incelediğimiz zaman rbf kernelinin en iyi C değerinin 10 üzeri 0 civarlarında olduğunu görüyoruz. Ayrıca en iyi sonucu bize bu kernel vermektedir. Grafikte görüldüğü üzere datayı bu dağılıma uyduğu için neredeyse mükemmel bir şekilde ayırmaktadır.

III. Sonuç

Bu çalışmada, ikili sınıflandırma probleminde yapay sinir ağları (YSA) ve destek vektör makineleri (DVM) yöntemlerinin performansları karşılaştırılmıştır. 400 örnekten oluşan bir veri kümesi üzerinde yapılan deneylerde, her iki yöntem de eğitim, doğrulama ve test setleri üzerinde aynı koşullar altında değerlendirilmiştir. DVM modelleri için lineer, polinomsal ve Gaussian RBF çekirdek fonksiyonları kullanılmış; YSA modelleri ise bir, iki ve üç gizli katmanlı yapılar ile test edilmiştir. Sonuçlar, DVM'nin özellikle küçük veri setlerinde daha istikrarlı ve hızlı sonuçlar verdiğini göstermiştir. Lineer çekirdek, basit ve hızlı bir çözüm sunarken, polinomsal ve Gaussian RBF çekirdekleri daha karmaşık veri yapılarında daha yüksek doğruluk oranları elde etmiştir. Özellikle Gaussian RBF çekirdeği, veri kümesini neredeyse mükemmel bir şekilde sınıflandırmıştır.

YSA modelleri ise doğru hiper parametre seçimi ve mimari tasarım ile yüksek performans göstermiştir. Mini-batch gradient inişi (MBGD) yöntemi, stokastik gradyan inişi (SGD) ve batch gradyan inişi (BGD) yöntemlerine kıyasla daha iyi sonuçlar vermiştir. Ancak, YSA'nın eğitim süreci DVM'ye göre daha uzun sürmüş ve daha fazla hesaplama kaynağı gerektirmiştir.

Genel olarak, DVM'nin küçük ve orta ölçekli veri setlerinde daha etkili olduğu, YSA'nın ise doğru hiper parametre optimizasyonu ile daha büyük ve karmaşık veri setlerinde daha yüksek performans gösterebileceği sonucuna varılmıştır. Her iki yöntem de sınıflandırma problemlerinde güçlü araçlar olmakla birlikte, veri setinin boyutu ve karmaşıklığına göre uygun yöntemin seçilmesi önemlidir.