OPTICS KÜMELEME VE YAPAY SİNİR AĞI TAHMİNİ

Berkay Aktürk

Günümüzde, veri analizi ve makine öğrenimi teknolojileri, birçok alanda karar verme süreçlerini desteklemekte ve otomatize etmektedir. Özellikle büyük veri setlerinin işlenmesi ve anlamlı bilgilere dönüştürülmesi, modern iş dünyasının ve bilimsel araştırmaların vazgeçilmez bir parçası haline gelmiştir. Bu çalışmada, veri analizi, veri madenciliği, görüntü işleme, örüntü tanıma ve makine öğrenme gibi alanlarında sıkça kullanılan iki yöntem olan optik kümeleme (OPTICS) ve yapay sinir ağı (YSA) sınıflandırması üzerine odaklanacağız. Bu iki algoritma geniş bir uygulama yelpazesine sahiptir ve literatürde güçlü sonuçlar üreten bir yapıya sahiptir [1, 2, 3].

Optik Kümeleme (OPTICS)

OPTICS (Ordering Points To Identify the Clustering Structure), veri madenciliğinde kullanılan bir kümeleme algoritmasıdır, yoğunluk tabanlı kümeleme yöntemlerinden biridir ve genellikle veri setlerinin yoğunluk tabanlı dağılımlarını tespit etmek için kullanılır. DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algoritmasının bir genişletmesi olarak Mihael Ankerst, Markus M. Breunig, Hans-Peter Kriegel ve Jörg Sander tarafından 1999 yılında tanıtılmıştır [4]. OPTICS, genellikle veri setinin yapısını anlamak için bir stand-alone (yoğunluk farklılıklarını tespit etme) aracı olarak kullanılır veya diğer veri işleme tekniklerinin önceden belirlenen kümeler üzerinde işlem yapabilmesi için bir önişleme adımı olarak kümeleme işlemini gerçekleştirir [2].

OPTICS algoritmasının temel avantajlarından biri, kullanıcının küme sayısını önceden belirlemesini gerektirmemesidir. Bunun yerine, algoritma veri noktalarını bir ulaşılabilirlik grafiği üzerinde sıralar ve bu grafik üzerinden farklı yoğunluklardaki kümeleri ayırt etmeyi mümkün kılar. Bu sayede, kullanıcılar algoritmanın çıktısını analiz ederek en uygun küme sayısına ve yapısına karar verebilirler. Ayrıca yüksek boyutlu veri setlerinde çeşitli yoğunluk ve şekillere sahip kümeleri çıkarmak için kullanılabilir. Ancak, OPTICS algoritması büyük veri setleri veya boyut sayısı yüksek durumlar için hala uzun bir uygulama süresi ile mücadele etmektedir [5]. Ayrıca değişken yoğunluklara sahip veri setlerinde ve gürültülü (outlier) verilerin olduğu durumları tespit etmede etkilidir.

OPTICS algoritması, değişken yoğunluklu veri setlerinde etkin bir şekilde kümeleme yapabilen ve gürültüyü etkin bir şekilde tanımlayabilen FOP-OPTICS (Finding of the Ordering Peaks Based on OPTICS) adlı bir algoritmayla daha da iyileştirildi. FOP-OPTICS, OPTICS tarafından oluşturulan Augmented Cluster-Ordering'den ayrım noktası (DP) bulur ve DP'nin ulaşılabilir mesafesini karşılık gelen kümeyi belirlemek için kullanır. Bu sayede, çoğu algoritmanın eşitsiz yoğunluklara sahip veri setlerini kümeleme konusundaki zayıflıklarını aşar [6, 7].

Yapay Sinir Ağı (YSA) Sınıflandırması

Yapay sinir ağları, insan beyninin bilgi işleme mekanizmasından esinlenerek geliştirilmiş ve bu bilgi işlemeye yönelik yaklaşımını taklit etmeye çalışan yapay öğrenme sistemleridir. İnsan beynindeki nöronların karmaşık ağından ilham alan YSA'lar, veri setlerindeki karmaşık ilişkileri modellemek ve tahminler yapmak için kullanılır [3]. YSA sınıflandırması, girdi olarak aldığı verileri işleyerek, bu verilerin hangi kategorilere ait olduğunu

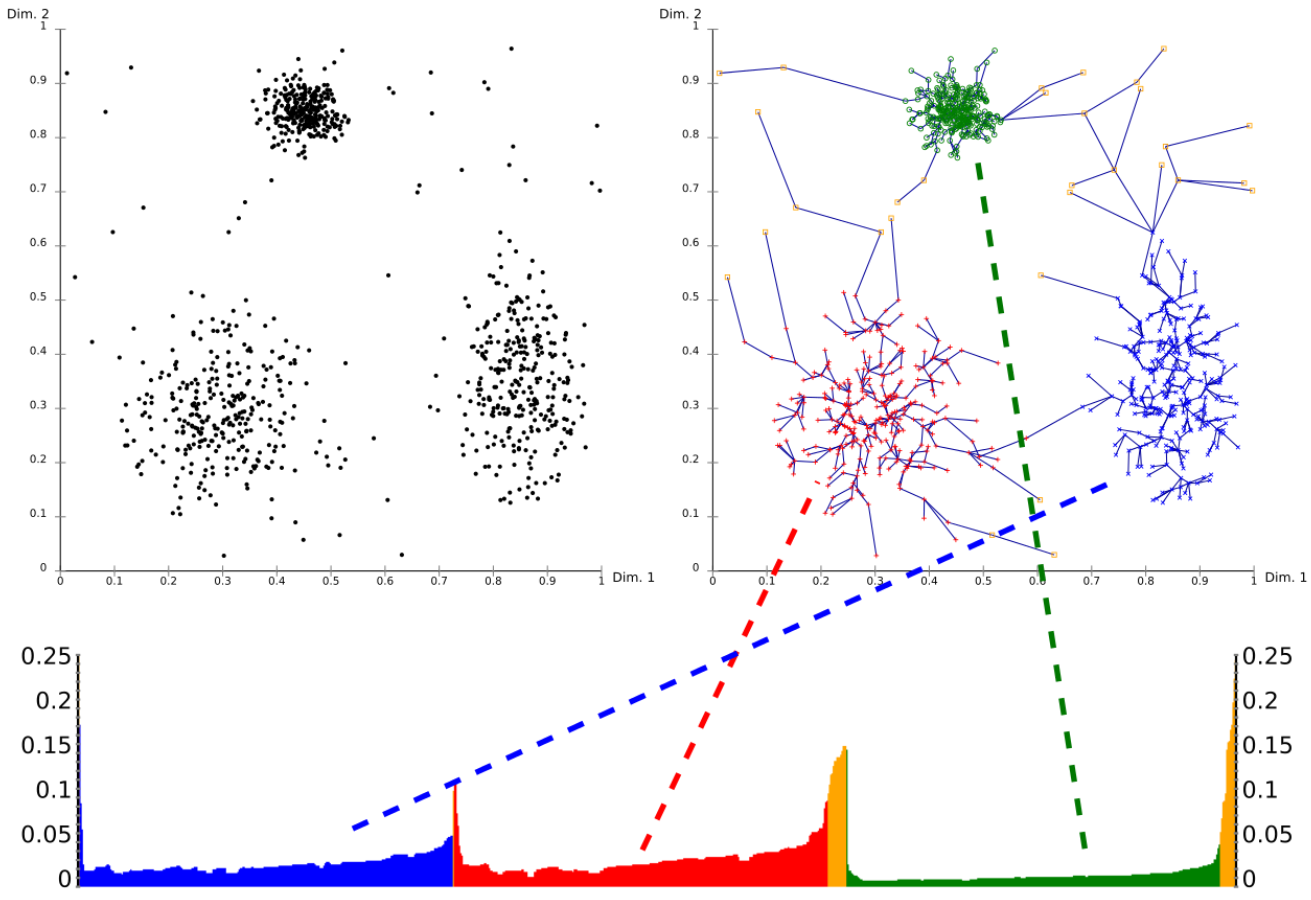
belirlemeye çalışan bir süreçtir. YSA'lar kesin çözümler sunmazlar; bunun yerine, genellikle 'en doğru' çözümü tahmin etmekte ve büyük ölçekli karmaşık verileri işlemede oldukça başırılıdırlar.

YSA'lar genellikle çok katmanlı perceptronlar (MLP) olarak adlandırılan yapıları kullanır. Bu yapılar, giriş katmanı, bir veya daha fazla gizli katman ve çıkış katmanından oluşur. Her katman, birbirine ağırlıklarla bağlı olan nöronlardan meydana gelir. Eğitim sürecinde, ağın ağırlıkları geri yayılım (backpropagation) algoritması kullanılarak güncellenir. Bu sayede YSA, verilen girdilere karşılık doğru çıktıları üretecek şekilde eğitilmiş olur ve başarı olasılığı artmış olur [8]. YSA sınıflandırması, özellikle görüntü işleme, ses tanıma ve metin sınıflandırma gibi alanlarda yaygın olarak kullanılmaktadır. Yapay sinir ağlarının başarısı, genellikle yeterli miktarda etiketlenmiş verinin varlığına ve ağın doğru şekilde eğitilmesine bağlıdır.

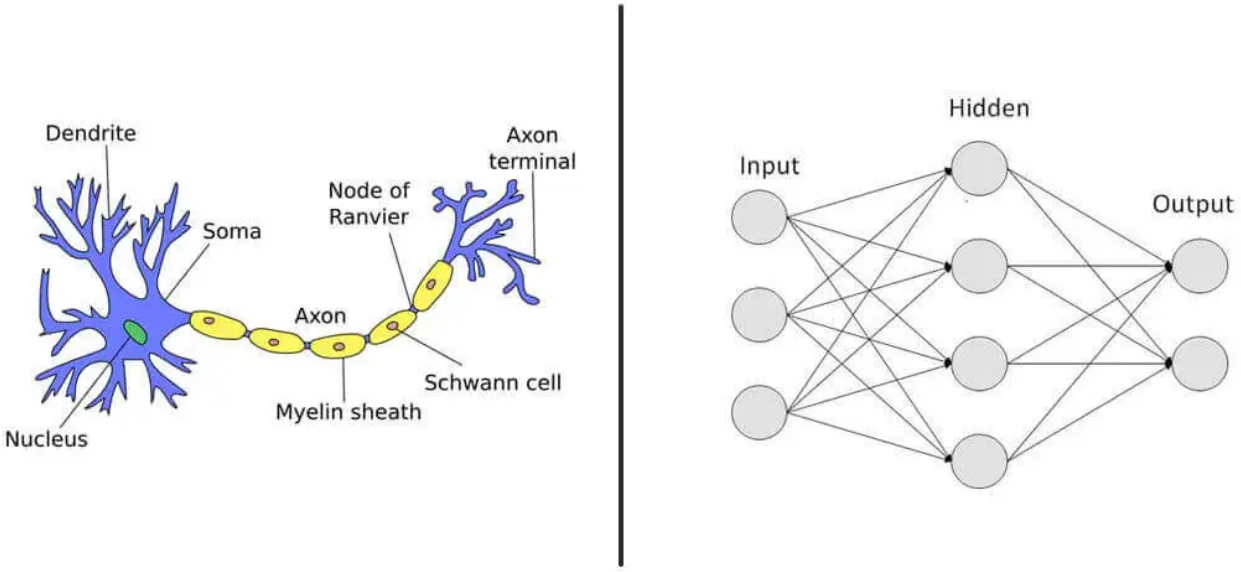
OPTICS ve YSA'nın Birlikte Kullanımı

OPTICS kümeleme algoritması ve YSA'nın beraber kullanılması, veri setleri üzerinden daha kapsamlı analizler gerçekleştirme ve daha verimli tahmin modelleri oluşturma potansiyeline sahiptir. Örneğin, büyük ve kompleks bir veri setinin incelenmesi durumunda, OPTICS algoritması veri setini belirgin gruplara (kümeler) ayırabilir ve daha sonra bu kümelerden alınan örnekler YSA modelinin eğitiminde kullanılabilir. Bu tür bir yaklaşım, hem veri setinin genel yapısının anlaşılmasını kolaylaştırabilir, hem de YSA modelinin sınıflandırma performansını artırabilir [9].

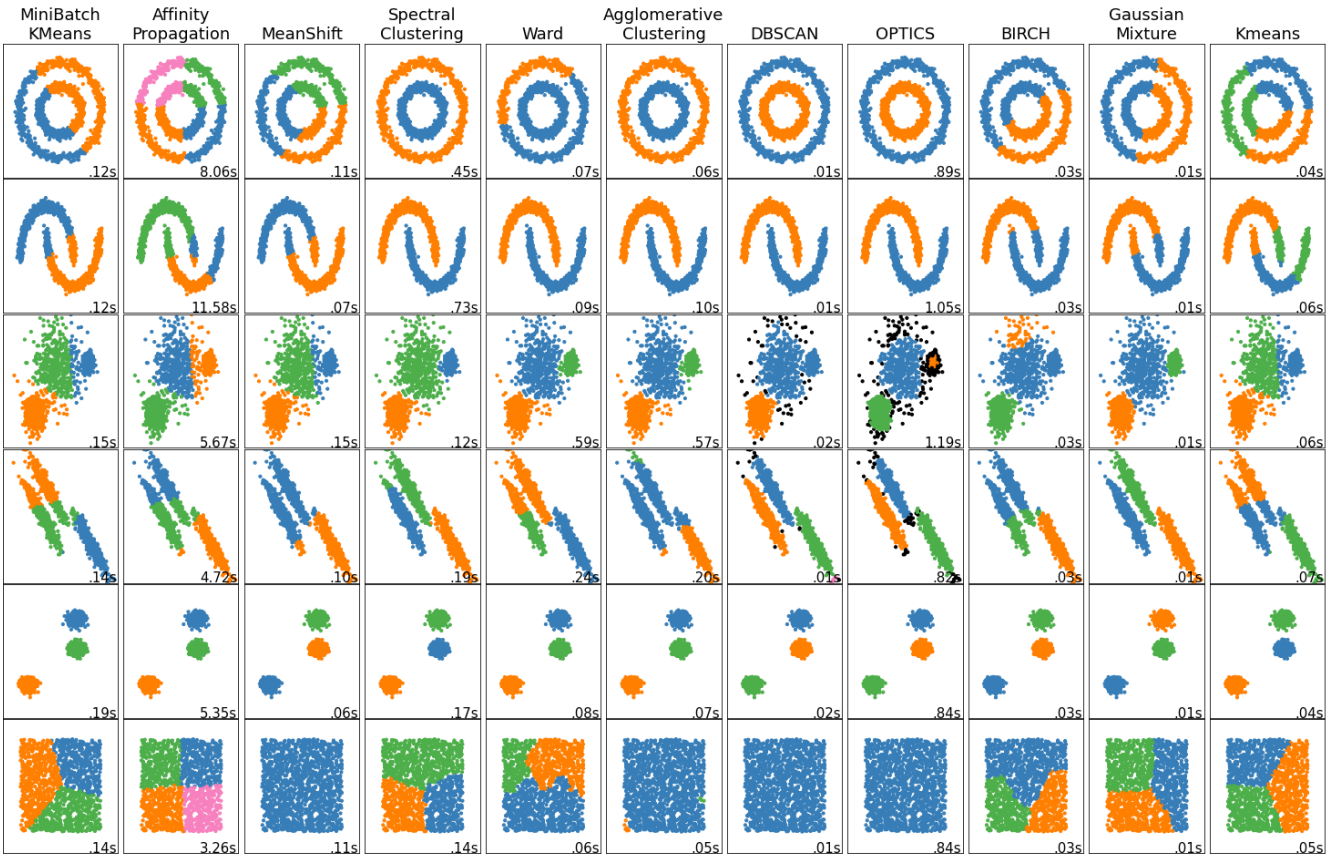
OPTICS kümeleme ve YSA'nın her biri, veri analitiği ve makine öğrenmesi alanlarında güçlü tekniklerdir. Her bir teknik, çeşitli problem tiplerinin çözümünde kendi başına etkileyici sonuçlar doğurabilirken; bir araya getirilerek kullanıldıklarında, daha kompleks veri yapılarının çözümüne yardımcı olabilir, ve ayrıca daha doğru tahminlerin elde edilmesine olanak sağlayabilir [10]. Ayrıca yapay sinir ağı modelinin kümeleme için kullanılması, işletmeler ve araştırmacılar tarafından karşılaşılan çeşitli problemlerin üstesinden gelebilmek adına önemli stratejik adımlar olarak gösterilebilir [11].



> **Functioning of Optics Clustering**



> **Artificial Neural Network Reference**



> **Clustering Algorithms**

Buraya bakın Görsel ve algoritmalar için daha fazla bilgi.

Görselin oluşturulması

Kütüphaneler Ve Veri Seti

In [1]:
```
# Veri işleme ve analizi için kütüphaneler
import pandas as pd
import numpy as np

# Görselleştirme kütüphaneleri
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px

# Makine öğrenimi ve veri önişleme kütüphaneleri
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from sklearn.cluster import OPTICS
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_score, f1_score, s
from sklearn.metrics.cluster import contingency_matrix
from sklearn.model_selection import train_test_split, KFold

# Yapay sinir ağları ve Keras kütüphaneleri
import tensorflow as tf
from keras.models import Sequential, Model
from keras.layers import Dense, Conv1D, Flatten, Input
from keras.utils.vis_utils import plot_model

# İstatistik ve diğer yardımcı kütüphaneler
from scipy.spatial.distance import euclidean
from scipy import stats
import networkx as nx
import random as python_random

# IPython display için
from IPython.display import Image
```

Optics ve yapay sinir ağı algoritmalarının matematiksel hesaplamaları ile uygulamalarının karşılaştırılması, aşağıdaki veri seti üzerinden gerçekleştirilecektir. Optics algoritması ile kümeleme yapılacak ve elde edilen yeni değişkeninin tahmini için yapay sinir ağı kullanılacaktır.

In [2]:
```
data = np.array([
    [10, 4, 21],
    [13, 5, 20],
    [11, 8, 20],
    [14, 7, 19],
    [11, 4, 15],
    [26, 7, 23],
    [40, 10, 23],
    [32, 14, 31],
    [23, 13, 25],
    [19, 10, 24]
])
data2 = pd.DataFrame(data, columns=['Feature1', 'Feature2', 'Feature3'])
data2
```

Out[2]:

	Feature1	Feature2	Feature3
0	10	4	21
1	13	5	20
2	11	8	20
3	14	7	19
4	11	4	15
5	26	7	23
6	40	10	23

7	32	14	31
8	23	13	25
9	19	10	24

In [3]:
```
fig = px.scatter_3d(data2, x='Feature1', y='Feature2', z='Feature3', title="3D Scatter P
fig.update_layout(margin=dict(l=0, r=0, b=0, t=0))
fig.show()
```

OPTICS Kümeleme Yöntemi

Algoritmanın Temel Adımları

OPTICS algoritması, yoğunluk tabanlı bir kümeleme yöntemidir ve veri noktalarını belirli parametrelere bağlı olarak bir sıralamaya koyarak kümeleri belirler. Yani, her veri noktası için komşu noktaların yoğunluğunu analiz ederek ve bu yoğunluğa göre bir ulaşılabilirlik mesafesi hesaplayarak çalışır. Her bir veri noktasının çekirdek mesafesi ve ulaşılabilirlik mesafesi hesaplanır ve bu bilgiler veri setinin yoğunluk tabanlı bir sıralamasını oluşturmak için kullanılır. Sonrasında bu sıralama, kümelerin tanımlanması için kullanılır. Algoritmanın temel adımları şu şekildedir:

1. Her veri noktası için bir çekirdek mesafesi (ϵ) ve minimum nokta sayısı (MinPts) tanımlanır.
2. Her noktanın ulaşılabilirlik mesafesi hesaplanır.
3. Noktalar ulaşılabilirlik mesafesine göre sıralanır.

4. Bu sıralama daha sonra kümelerin çıkarılması için kullanılır.

Matematiksel Tanımlar

- **Çekirdek Mesafe (ϵ):** Bir noktanın çekirdek nokta olarak kabul edilmesi için çevresindeki belirli bir yarıçap içinde bulunması gereken minimum komşu sayısıdır. Matematiksel olarak aşağıdaki gibi ifade edilir:

$$\text{Core-Dist}_\epsilon(p) = \begin{cases} \text{UNDEFINED} & \text{if } |N_\epsilon(p)| < \text{MinPts} \\ \text{MinPts-th smallest distance} & \text{if } |N_\epsilon(p)| \geq \text{MinPts} \end{cases}$$

  Burada, $N_\epsilon(p)$, p noktasının ϵ mesafesi içindeki komşu noktaların sayısını ifade eder.

- **Ulaşılabilirlik Mesafesi:** Bir noktanın başka bir noktaya olan ulaşılabilirlik mesafesidir. Yoğunluk-tabanlı bir ölçüttür ve aşağıdaki şekilde tanımlanır:

$$\text{Reachability-Dist}_\epsilon(p, q) = \max\{\text{Core-Dist}_\epsilon(p), d(p, q)\}$$

  Burada, $d(p, q)$, p ve q arasındaki öklid mesafesini temsil eder.

Algoritmanın Özellikleri

- Yoğunluk tabanlı bir kümeleme yöntemidir.
- Farklı yoğunluklara sahip kümeleri belirleyebilir.
- Gürültüyü (outlier) ayırt edebilir ve kümelenmeye dahil etmez.

Matematiksel Hesaplama

Veri seti, 3 boyutlu (x, y, z) ve 10 veri noktasından oluşmaktadır. Bu veri seti üzerinden OPTICS algoritmasının uygulanabilmesi için öncelikle `\epsilon` (çekirdek mesafe) ve `MinPts` (minimum nokta sayısı) parametrelerinin belirlenmesi gerekmektedir.

1. **Çekirdek Mesafe (ϵ) Hesaplama:** Bu değer genellikle kullanıcı tarafından belirlenir. Diyelim ki `epsilon = 7` olarak belirlendi.

2. **Minimum Nokta Sayısı (MinPts) Hesaplama:** Bu değer de kullanıcı tarafından belirlenir. Diyelim ki `MinPts = 2` olarak belirlendi.

3. **Çekirdek Mesafe Ulaşılabilirlik Mesafesi Hesaplama:** Her bir veri noktası için, diğer tüm noktalarla olan mesafeler hesaplanır ve `$\epsilon$` içinde kalan komşu sayısı tespit edilir. Eğer bir noktanın komşu sayısı `MinPts`'den fazla ise, bu nokta bir çekirdek nokta olarak kabul edilir.

4. **Sıralama Oluşturma:** Çekirdek mesafe ve ulaşılabilirlik mesafesi bilgileri kullanılarak bir sıralama oluşturulur. Bu sıralama, veri setindeki yoğunluk tabanlı yapıyı yansıtır.

5. **Kümelerin Çıkarılması:** Sıralama üzerinden gidilerek, yoğunluk tabanlı kümeler çıkarılır.

Bu adımları veri seti üzerinden uygulamak için öncelikle `$\epsilon$` ve `MinPts` değerlerinin belirlenmesi ve ardından uzaklık hesaplamalarının yapılması gerekir. Uzaklık hesaplamaları için genellikle Öklid uzaklığı kullanılır. Öklid uzaklığı, iki nokta arasındaki en kısa doğrusal mesafeyi ifade eder ve iki boyutlu bir uzayda

bu mesafe, iki nokta arasındaki uzaklığın hesaplanmasıyla bulunur. İki nokta arasındaki Öklid mesafesi `(x1, y1)` ve `(x2, y2)` ise bu mesafe şu formülle hesaplanır:

$$d(p, q) = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Veri setimiz için aşağıdaki öklid uzaklık mesafesi formülünü kullanabiliriz:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}$$

Burada `p` ve `q`, üç boyutlu uzaydaki iki farklı noktayı temsil eder ve `p1, p2, p3` ile `q1, q2, q3`, bu noktaların koordinatlarıdır.

Örneğin, MinPts değerimiz 2 olsun ve ilk veri noktamız `[10, 4, 21]` için diğer tüm noktalarla olan mesafeleri aşağıdaki gibi hesaplayabiliriz.

$$d([10, 4, 21], [13, 5, 20]) = \sqrt{(10 - 13)^2 + (4 - 5)^2 + (21 - 20)^2} = \sqrt{9 + 1 + 1} = \sqrt{11} \approx 3.32$$

$$d([10, 4, 21], [11, 8, 20]) = \sqrt{(10 - 11)^2 + (4 - 8)^2 + (21 - 20)^2} = \sqrt{1 + 16 + 1} = \sqrt{18} \approx 4.24$$

$$d([10, 4, 21], [14, 7, 19]) = \sqrt{(10 - 14)^2 + (4 - 7)^2 + (21 - 19)^2} = \sqrt{(-4)^2 + (-3)^2 + (2)^2}$$
$$= \sqrt{16 + 9 + 4} = \sqrt{29} \approx 5.39$$

$$d([10, 4, 21], [11, 4, 15]) = \sqrt{(10 - 11)^2 + (4 - 4)^2 + (21 - 15)^2} = \sqrt{(-1)^2 + (0)^2 + (6)^2}$$
$$= \sqrt{1 + 0 + 36} = \sqrt{37} \approx 6.08$$

...

Ve bu işlem veri setindeki tüm noktalar için yapmaya devam edilir. Toplam 45 adet öklid uzaklığı hesaplanır. Her bir nokta için hesaplanan mesafeler sıralandığında, en yakın MinPts sayıda komşunun en büyük mesafesi alınır ve bu nokta çekirdek mesafesi olur.

In [4]:
```python
def calculate_distances(data):
    distances = {}
    for i in range(len(data)):
        for j in range(i+1, len(data)):
            distance = np.linalg.norm(data[i] - data[j])
            distances[f'd({i+1},{j+1})'] = distance
    return distances

distances = calculate_distances(data)
distances_sorted = dict(sorted(distances.items()))
distances_sorted
```

Out[4]:
```
{'d(1,10)': 11.224972160321824,
 'd(1,2)': 3.3166247903554,
 'd(1,3)': 4.242640687119285,
 'd(1,4)': 5.385164807134504,
 'd(1,5)': 6.082762530298219,
 'd(1,6)': 16.401219466856727,
 'd(1,7)': 30.659419433511783,
 'd(1,8)': 26.153393661244042,
 'd(1,9)': 16.30950643030009,
 'd(2,10)': 8.774964387392123,
 'd(2,3)': 3.605551275463989,
 'd(2,4)': 2.449489742783178,
 'd(2,5)': 5.477225575051661,
 'd(2,6)': 13.490737563232042,
```

```
'd(2,7)': 27.622454633866266,
'd(2,8)': 23.727621035409346,
'd(2,9)': 13.74772708486752,
'd(3,10)': 9.16515138991168,
'd(3,4)': 3.3166247903554,
'd(3,5)': 6.4031242374328485,
'd(3,6)': 15.329709716755891,
'd(3,7)': 29.223278392404914,
'd(3,8)': 24.454038521274967,
'd(3,9)': 13.92838827718412,
'd(4,10)': 7.681145747868608,
'd(4,5)': 5.830951894845301,
'd(4,6)': 12.649110640673518,
'd(4,7)': 26.476404589747453,
'd(4,8)': 22.737634001804146,
'd(4,9)': 12.36931687685298,
'd(5,10)': 13.45362404707371,
'd(5,6)': 17.26267650163207,
'd(5,7)': 30.675723300355934,
'd(5,8)': 28.231188426986208,
'd(5,9)': 18.027756377319946,
'd(6,10)': 7.681145747868608,
'd(6,7)': 14.317821063276353,
'd(6,8)': 12.206555615733702,
'd(6,9)': 7.0,
'd(7,10)': 21.02379604162864,
'd(7,8)': 12.0,
'd(7,9)': 17.378147196982766,
'd(8,10)': 15.297058540778355,
'd(8,9)': 10.862780491200215,
'd(9,10)': 5.0990195135927845}
```

Yukarıdaki noktaların minimum değerleri çekilir ve nokta çekirdek mesafesi elde edilir.

In [5]:
```
distances = np.array([[euclidean(p1, p2) for p2 in data] for p1 in data])
core_distances = np.partition(distances, 1)[:,1]
core_distances
```

Out[5]:
```
array([ 3.31662479,  2.44948974,  3.31662479,  2.44948974,  5.47722558,
        7.        , 12.        , 10.86278049,  5.09901951,  5.09901951])
```

Mesafe değerlerini kullanarak her bir noktanın en yakın komşusunun mesafesini bulmak için, her bir nokta için diğer tüm noktalarla olan mesafeleri karşılaştırıp en küçük olanı seçmemiz gerekiyor. Eğer belirli bir gözlemin eşit iki farklı noktası olursa önce listelenen eklenir. Her bir nokta için en yakın komşuların mesafeleri aşağıdaki gibidir:

- Nokta 1 için en yakın komşu mesafesi: `d(1,2)` = 3.32
- Nokta 2 için en yakın komşu mesafesi: `d(2,4)` = 2.45
- Nokta 3 için en yakın komşu mesafesi: `d(3,4)` = 3.32
- Nokta 4 için en yakın komşu mesafesi: `d(4,2)` = 2.45
- Nokta 5 için en yakın komşu mesafesi: `d(5,3)` = 5.47
- Nokta 6 için en yakın komşu mesafesi: `d(6,9)` = 7
- Nokta 7 için en yakın komşu mesafesi: `d(7,6)` = 12
- Nokta 8 için en yakın komşu mesafesi: `d(8,6)` = 10.86
- Nokta 9 için en yakın komşu mesafesi: `d(9,10)` = 5.10
- Nokta 10 için en yakın komşu mesafesi: `d(10,9)` = 5.10

Bu hesaplamalar sırasında her bir nokta için diğer tüm noktalarla olan Öklid mesafelerini karşılaştırarak en küçük değeri buluyoruz ve bu da her bir noktanın diğer tüm noktalara olan en yakın komşu mesafesini verir.

Bu değerler her bir noktanın MinPts=2 için olan ulaşılabilirlik mesafesini de belirler. Ayrıca `MinPts = 2` olarak belirlendiği için, çekirdek mesafe, en yakın komşu mesafesi ile aynı olacaktır.

Ulaşılabilirlik Mesafesi Hesaplama

Bir noktanın ulaşılabilirlik mesafesi, o noktanın çekirdek mesafesi ile o noktaya olan Öklid mesafesi arasındaki maksimum değer olarak tanımlanır. Eğer bir nokta çekirdek nokta değilse (yani `ε` içinde yeterli sayıda komşusu yoksa), ulaşılabilirlik mesafesi sonsuz olarak kabul edilir.

Örnekteki her bir nokta için ulaşılabilirlik mesafesini hesaplayalım:

- Nokta 1 için en yakın komşu mesafesi `d(1,2)` = 3.32, `ε = 7` değerinden küçük olduğu için Nokta 1 bir çekirdek noktadır ve ulaşılabilirlik mesafesi en yakın komşu mesafesi ile aynıdır.
- Nokta 2 için en yakın komşu mesafesi `d(2,4)` = 2.45, bu da `ε = 7` değerinden küçük olduğu için Nokta 2 bir çekirdek noktadır ve ulaşılabilirlik mesafesi en yakın komşu mesafesi ile aynıdır.
- ... ve benzeri şekilde diğer noktalar için de bu hesaplama yapılır.

Bu bilgilerle, sıralama oluşturabiliriz. Sıralama genellikle ulaşılabilirlik mesafelerine göre yapılır ve düşükten yükseğe doğru sıralanır. Ancak, burada sadece en yakın komşu mesafeleri verildiği için ve her noktanın `ε` içinde yeterli sayıda komşusu olduğunu varsayarsak, tüm noktalar çekirdek nokta olarak kabul edilebilir ve sıralamayı doğrudan bu mesafelere göre yapabiliriz.

Bu durumda sıralama şu şekilde olur (en yakın komşu mesafesine göre artan sırada):

1. Nokta 2 (`d(2,4)` ≈ 2.45)
2. Nokta 4 (`d(4,2)` ≈ 2.45)
3. Nokta 1 (`d(1,2)` ≈ 3.32)
4. Nokta 3 (`d(3,4)` ≈ 3.32)
5. Nokta 9 (`d(9,10)` ≈ 5.10)
6. Nokta 10 (`d(10,9)` ≈ 5.10)
7. Nokta 5 (`d(5,3)` ≈ 5.47)

Nokta 6 için:

En yakın komşu mesafesi d(6,9) = 7. Bu değer ε = 7 ile sınırda olduğu için, Nokta 6 bir çekirdek nokta olarak kabul edilebilir. Eğer ε sınır değerler çekirdek nokta olarak kabul etmiyorsa, Nokta 6 çekirdek nokta olarak kabul edilmez ve ulaşılabilirlik mesafesi sonsuz olur.

Nokta 7 için:

En yakın komşu mesafesi d(7,6) = 12. Bu değer ε = 7 değerinden büyük olduğu için Nokta 7 bir çekirdek nokta değildir ve ulaşılabilirlik mesafesi sonsuz olarak kabul edilir.

Nokta 8 için:

En yakın komşu mesafesi d(8,6) = 10.86. Bu değer de ε = 7 değerinden büyük olduğu için Nokta 8 de bir çekirdek nokta değildir ve ulaşılabilirlik mesafesi sonsuz olarak kabul edilir.

Bu hesaplamalar sonucunda, eğer sınır değerler çekirdek nokta olarak kabul ediliyorsa, Noktalar 2, 4, 1, 3 ve (muhtemelen) 5, 9 ve 10 aynı veya benzer kümelerde yer alabilirler.

Bu sıralama algoritmanın bir sonraki adımında kullanılacak olan yoğunluk tabanlı kümelerin oluşturulması için temel oluşturur.

Küme Sınırlarının Belirlenmesi

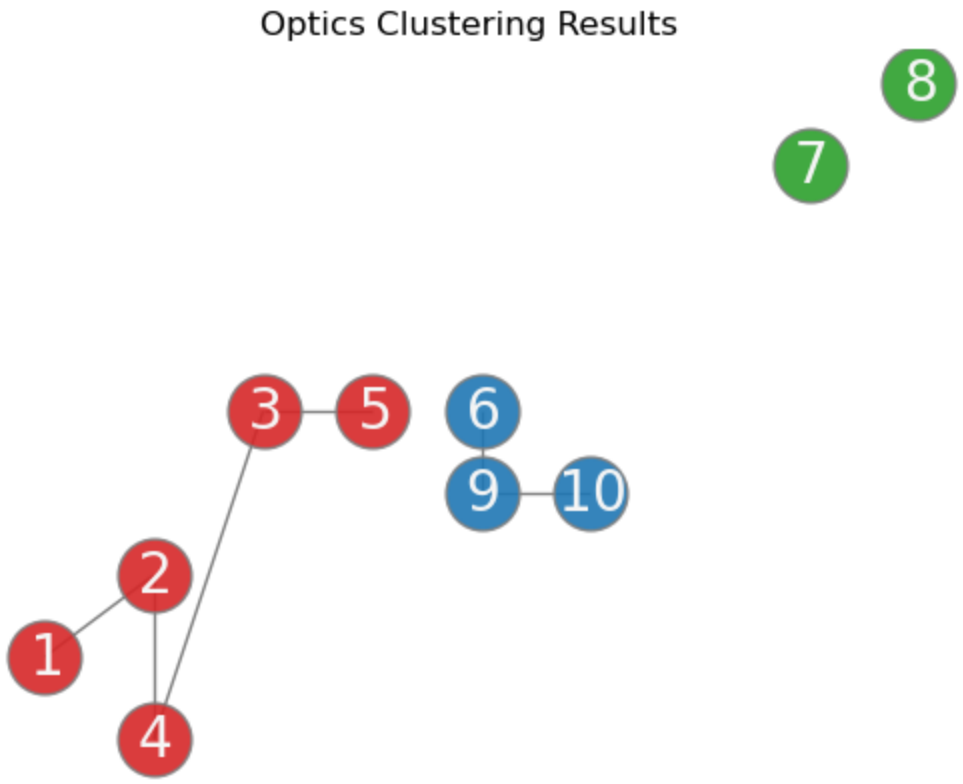
Küme sınırlarını belirleme işlemi, ulaşılabilirlik grafiğini analiz etmekle ilgilidir. Grafikte, yüksek dikey çizgiler genellikle farklı kümeler arasındaki sınırları temsil eder. Bu çizgiler, bir kümeden diğerine geçişteki yoğunluk farklılıklarını gösterir.

- Ulaşılabilirlik grafiğinde, yüksek dikey çizgilerin konumları belirlenir. Bu çizgiler, potansiyel küme sınırlarını gösterir.
- Yüksek çizgiler arasında kalan ve daha düşük ulaşılabilirlik değerlerine sahip noktalar genellikle aynı küme içinde yer alır.
- Küme içi noktaları ve küme geçişlerini temsil eden yüksek çizgileri ayırt etmek için ulaşılabilirlik değerlerindeki değişimleri incelenir.
- Küme içi noktalar ve küme sınırları belirlendikten sonra, bu bilgiyi kullanarak veri setindeki kümeler tanımlanır ve etiketlenir.

Bu süreç, veri setindeki yoğunluk tabanlı kümelerin belirlenmesine yardımcı olur ve görsel bir analizle desteklenir.

In [6]:
```
points = {
    '1': (1, 2),
    '2': (2, 3),
    '3': (3, 5),
    '4': (2, 1),
    '5': (4, 5),
    '6': (5, 5),
    '9': (5, 4),
    '10': (6, 4),
    '7': (8, 8),
    '8': (9, 9),
}

cluster_edges = [
    ('1', '2'),
    ('2', '4'),
    ('3', '4'),
    ('3', '5'),
    ('6', '9'),
    ('9', '10'),
]

G = nx.Graph()
G.add_nodes_from(points.keys())
G.add_edges_from(cluster_edges)

core_nodes = ['1', '2', '3', '4', '5', '6', '9', '10']
border_nodes = []
outlier_nodes = ['7', '8']
pos = points
options = {"edgecolors": "tab:gray", "node_size": 700, "alpha": 0.9}

nx.draw_networkx_nodes(G, pos, nodelist=['1', '2', '3', '4', '5'], node_color="tab:red",
nx.draw_networkx_nodes(G, pos, nodelist=['6', '9', '10'], node_color="tab:blue", **optio
nx.draw_networkx_nodes(G, pos, nodelist=['7', '8'], node_color="tab:green", **options)
nx.draw_networkx_edges(G, pos, width=1.0, alpha=0.5)

labels = {str(i): r"$" + str(i) + r"$" for i in G.nodes()}
nx.draw_networkx_labels(G, pos, labels, font_size=20, font_color="whitesmoke")
plt.title("Optics Clustering Results")
```

```
plt.axis('off')
plt.show()
```



Model Kurulumu

In [7]:
```
optics_cluster = OPTICS(min_samples=2, max_eps=7, metric='euclidean')
clusters = optics_cluster.fit_predict(data)
clusters
```

Out[7]:
```
array([ 0,  0,  0,  0,  0,  1, -1, -1,  1,  1])
```

Model Parametreleri

min_samples: int > 1 veya 0 ile 1 arasında float, default=5

Bir noktanın çekirdek nokta olarak kabul edilmesi için komşuluktaki örnek sayısı. Ayrıca, yukarı ve aşağı dik bölgeler min_samples ardışık dik olmayan noktalardan daha fazlasına sahip olamaz. Mutlak bir sayı veya örnek sayısının bir kesri olarak ifade edilir (en az 2 olacak şekilde yuvarlanır).

max_eps: float, default=np.inf

Birinin diğerinin komşuluğunda kabul edilmesi için iki örnek arasındaki maksimum mesafe. Varsayılan np.inf değeri tüm ölçeklerdeki kümeleri tanımlayacaktır; max_eps değerinin azaltılması daha kısa çalışma sürelerine neden olacaktır.

metric: str veya callable, default='minkowski'

Mesafe hesaplaması için kullanılacak metrik. scikit-learn veya scipy.spatial.distance'dan herhangi bir metrik kullanılabilir.

Metrik çağrılabilir bir işlevse, her bir örnek çifti (satır) üzerinde çağrılır ve elde edilen değer kaydedilir. Çağrılabilir fonksiyon girdi olarak iki dizi almalı ve aralarındaki mesafeyi gösteren bir değer döndürmelidir. Bu, Scipy'nin metrikleri için çalışır ancak metrik adını bir dize olarak geçirmekten daha az verimlidir. Metrik "önceden hesaplanmış" ise, X'in bir mesafe matrisi olduğu ve kare olması gerektiği varsayılır.

Metrik için geçerli değerler şunlardır:

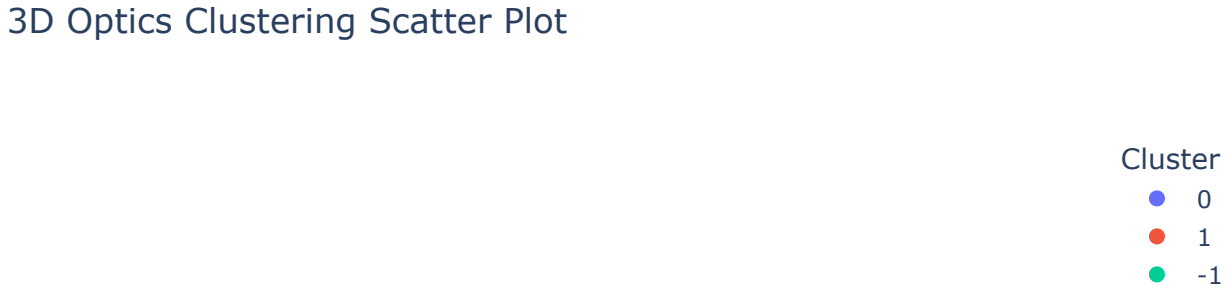
scikit-learn'den: ['cityblock', 'cosine', 'euclidean', 'l1', 'l2', 'manhattan']

from scipy.spatial.distance: ['braycurtis', 'canberra', 'chebyshev', 'correlation', 'dice', 'hamming', 'jaccard', 'kulsinski', 'mahalanobis', 'minkowski', 'rogerstanimoto', 'russellrao', 'seuclidean', 'sokalmichener', 'sokalsneath', 'sqeuclidean', 'yule']

In [8]:
```
combined_data = np.hstack((data, clusters.reshape(-1,1))) # Küme bilgisini veriye ekleme
df = pd.DataFrame(combined_data, columns=['Feature1', 'Feature2', 'Feature3', 'Cluster'])
print(df)
```

```
   Feature1  Feature2  Feature3  Cluster
0        10         4        21        0
1        13         5        20        0
2        11         8        20        0
3        14         7        19        0
4        11         4        15        0
5        26         7        23        1
6        40        10        23       -1
7        32        14        31       -1
8        23        13        25        1
9        19        10        24        1
```

In [9]:
```
df['Cluster'] = df['Cluster'].astype('category')
fig = px.scatter_3d(df, x='Feature1', y='Feature2', z='Feature3',
                    color='Cluster', labels={'Cluster': 'Cluster'},
                    title='3D Optics Clustering Scatter Plot')
fig.show()
```



Grafik incelendiğinde, -1 değerine sahip aykırı gözlemlerin bir kümeye ayrılması gerektiği düşünülmemelidir, veri kümesinin dağılımına göre her konumda yer alabilir. Bu kümeleme algoritması, veri kümesinde aykırı değerler olduğunda bu değerlere karşı önlem alınması gerektiğini göstermektedir. Veri setinden çıkartma, dönüşüm veya robust yöntemler uygulama gibi istatistiksel teknikler kullanılabilir. Ayrıca bu aykırı değerler, 1 kümesine yakın bir konumda bulunabilir ve epsilon değerine bağlı olarak 1 kümesine atanabileceği unutulmamalıdır.

Tüm aykırı değerlerin en yakın kümeye atanması

In [10]:
```
# tüm aykırı değerlerin en yakın kümeye atanması
optics_cluster = OPTICS(min_samples=2)
clusters = optics_cluster.fit_predict(data)
# aykırı değerleri (-1 olarak işaretlenmiş) yakın oldukları kümeye atama
def assign_outliers_to_nearest_cluster(clusters, data):
    changed = False
    for index, cluster in enumerate(clusters):
        if cluster == -1:
            # aykırı değerin diğer noktalara olan mesafelerini hesaplama
            distances = np.sqrt(np.sum((data - data[index])**2, axis=1))
            # Kendi mesafesini sıfırlama (sonsuz yapma)
            distances[index] = np.inf
            # en yakın noktanın kümelenme indeksini bulma
            nearest_cluster = clusters[np.argmin(distances)]
            clusters[index] = nearest_cluster
            changed = True
    return changed
while assign_outliers_to_nearest_cluster(clusters, data):
    continue
clusters
```

Out[10]:
```
array([0, 0, 0, 0, 0, 1, 1, 1, 1, 1])
```

In [11]:
```
combined_data = np.hstack((data, clusters.reshape(-1,1)))
df = pd.DataFrame(combined_data, columns=['Feature1', 'Feature2', 'Feature3', 'Cluster'])
print(df)
```

```
   Feature1  Feature2  Feature3  Cluster
0        10         4        21        0
1        13         5        20        0
2        11         8        20        0
3        14         7        19        0
4        11         4        15        0
5        26         7        23        1
6        40        10        23        1
7        32        14        31        1
8        23        13        25        1
9        19        10        24        1
```

In [12]:
```
df = pd.DataFrame(df)
```

In [13]:
```
df.describe()
```

Out[13]:

	Feature1	Feature2	Feature3	Cluster
count	10.000000	10.000000	10.000000	10.000000
mean	19.900000	8.200000	22.100000	0.500000
std	10.202941	3.521363	4.254409	0.527046
min	10.000000	4.000000	15.000000	0.000000

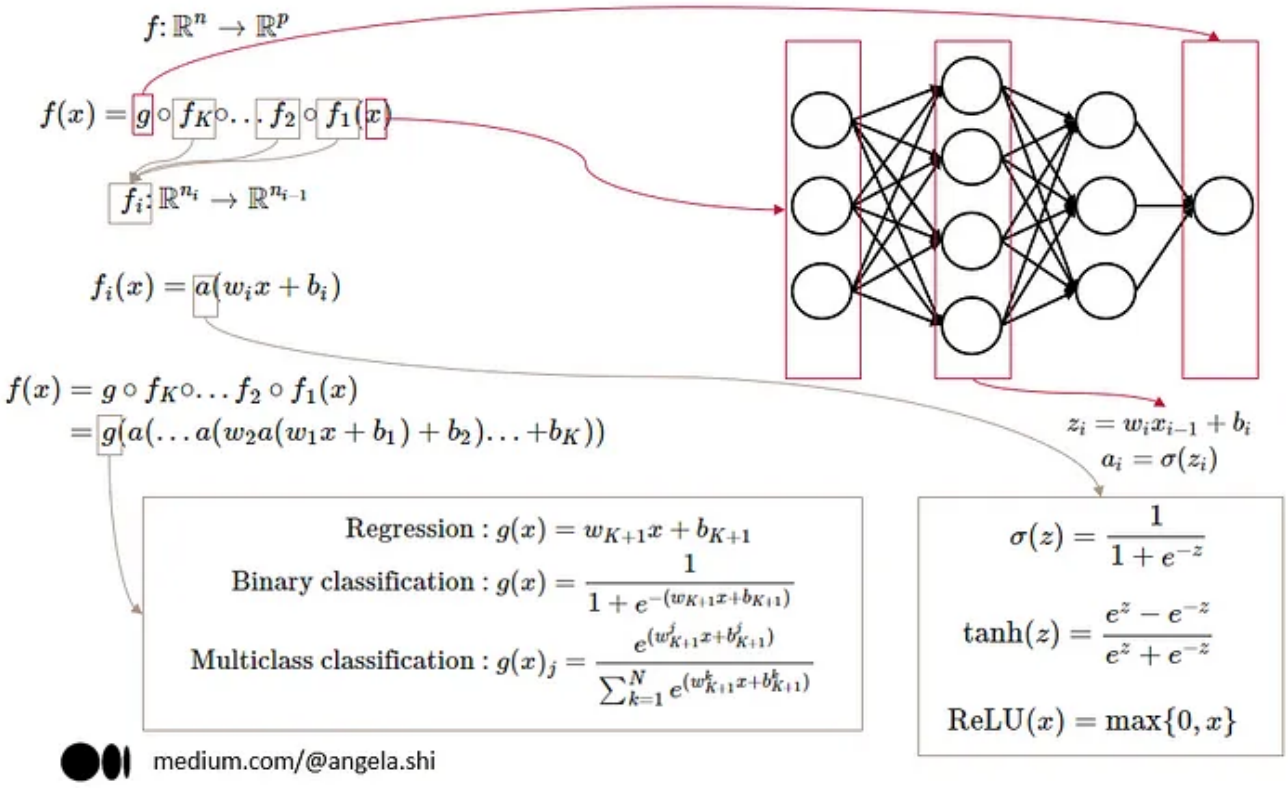
25%	11.500000	5.500000	20.000000	0.000000
50%	16.500000	7.500000	22.000000	0.500000
75%	25.250000	10.000000	23.750000	1.000000
max	40.000000	14.000000	31.000000	1.000000

Yapay Sinir Ağı Modeli ve Matematiksel Çözümü

Bir YSA modelinin matematiksel çözümü, genellikle ağırlık ve bias değerlerinin ayarlanmasıyla gerçekleşir. Bu, öğrenme süreci olarak adlandırılır ve genellikle geri yayılım algoritması kullanılarak yapılır. Model, eğitim veri seti üzerinde iteratif olarak çalıştırılır ve her iterasyonda ağırlıklar, hata oranını azaltacak şekilde güncellenir.

Sınıflandırma işlemi, YSA'nın çıktı katmanında gerçekleşir. Her nöron, belirli bir sınıfı temsil eder ve girdi verilerine göre bir aktivasyon değeri üretir. En yüksek aktivasyon değerine sahip nöron, girdinin ait olduğu sınıf olarak kabul edilir.

Bu süreçte kullanılan başlıca matematiksel fonksiyonlar aktivasyon fonksiyonları, kayıp fonksiyonları ve optimizasyon algoritmalarıdır. Aktivasyon fonksiyonları, nöronların çıktılarını belirlerken; kayıp fonksiyonları, modelin performansını ölçer ve optimizasyon algoritmaları ise modelin daha iyi sonuçlar vermesini sağlamak için ağırlıkları düzenler.



> **Formulation of Artificial Neural Network**

Model Süreci

1. **Verileri Ölçeklendirme**

RobustScaler kullanılarak veriler ölçeklendirilir. Bu, verilerin medyanını çıkarıp IQR (çeyrekler arası aralık) ile bölünmesi anlamına gelir. Bu işlem için her bir özelliğin medyanını ve IQR değerini hesaplayabilir ve bu değerleri kullanarak ölçeklendirme işlemini gerçekleştirebiliriz.

1. **Yapay Sinir Ağı Modelinin Oluşturulması**

Model, giriş katmanı, bir gizli katman ve bir çıkış katmanından oluşmaktadır. Her bir katmandaki ağırlıklar ve biaslar rastgele başlatılır. Ağırlık ve bias değerlerini ya kendimiz atayabiliriz ya da optimizasyon teknikleri ile rastgele değerlerle başlatarak, kaydetme işlemini gerçekleştirebiliriz.

1. **İleri Besleme (Forward Propagation)**

Modelin tahmin yapabilmesi için ileri besleme işlemi gerçekleştirilir. Bu, giriş verilerinin ağırlıklarla çarpılması, biasların eklenmesi ve aktivasyon fonksiyonlarının uygulanması sürecidir. Girdi verisi, yapay sinir ağının giriş katmanına yerleştirilir. Bu veri genellikle özellik vektörleri şeklindedir. Her nöron, kendisine bağlı olan her bir girdi için bir ağırlık değeri içerir. İleri besleme sırasında, her girdi değeri ilgili ağırlıkla çarpılır. Ayrıca, her nöronun bir bias değeri vardır ve bu değer, ağırlıkla çarpılmış girdilerin toplamına eklenir. Ağırlıklarla çarpılmış ve bias eklenmiş toplam, bir aktivasyon fonksiyonundan geçirilir. Aktivasyon fonksiyonu, nöronun çıktısını belirler. Bu fonksiyonlar genellikle ReLU, Sigmoid veya Tanh gibi non-lineer fonksiyonlardır. Aktivasyon fonksiyonu, ağın karmaşık desenleri öğrenebilmesini ve doğrusal olmayan problemleri çözebilmesini sağlar. Aktivasyon fonksiyonundan çıkan değerler, sonraki katmana girdi olarak geçer. Bu süreç, ağın tüm katmanlarında tekrarlanır. Son katmanda, nöronların çıktıları son aktivasyon fonksiyonundan geçirilir. Bu katman, genellikle problemin türüne göre düzenlenmiş (sınıflandırma, regresyon vb.) çıktıları üretir.

1. **Geri Yayılım ve Optimizasyon (Backpropagation and Optimization)**

Modelin eğitimi sırasında, geri yayılım algoritması ve bir optimizasyon algoritması (bu durumda 'adam') kullanılır. Bu, modelin ağırlıklarını güncellemek için kullanılır. El ile bu işlemi yapmak, türevleri hesaplamayı ve ağırlıkları uygun şekilde güncellemeyi gerektirir.

Üçüncü ve dördüncü süreçte kullanılan ağırlıklar, yapay sinir ağlarında bir nöronun girdileri ile çıktıları arasındaki ilişkiyi ifade eden parametrelerdir. Ağırlıklar, girdi verisinin nöron tarafından ne kadar dikkate alınacağını belirler. Yüksek bir ağırlık, ilgili girdinin nöronun aktivasyonu üzerinde daha fazla etkiye sahip olacağını gösterir. Ağın eğitimi sırasında geri yayılım (backpropagation) ve optimizasyon algoritmaları ile güncellenir. Bu güncellemeler, ağın veriler üzerindeki tahminlerinin doğruluğunu artırmak için yapılır.

Bias ise, bir nöronun aktivasyon fonksiyonuna eklenen ekstra bir sabit değerdir. Her nöron için genellikle bir bias değeri bulunur. Bias, nöronun aktivasyon eşiğini ayarlar. Yani, nöronun aktif hale gelmesi için gerekli olan girdi sinyalinin toplam seviyesini belirler. Bu, ağın sadece belirli bir eşik değeri aşıldığında aktive olmasını sağlar ve böylece modelin daha esnek ve etkili bir şekilde öğrenmesine yardımcı olur. Biaslar da ağırlıklar gibi eğitim sırasında güncellenir. Modelin daha karmaşık desenleri öğrenmesini ve doğrusal olmayan problemleri çözebilmesini sağlar.

Kısaca ağırlıklar, girdi ve çıktı arasındaki ilişkinin gücünü belirlerken, bias ise nöronun ne zaman aktive olacağını (yani bir sinyali ne zaman önemli kabul edeceğini) belirler. Her ikisi de yapay sinir ağının öğrenme yeteneğinin temel taşlarıdır ve ağın eğitimi sırasında sürekli olarak ayarlanır. Bu süreç, ağın veri setindeki yapıyı ve ilişkileri öğrenmesini sağlar.

1. **Yineleme**

Bu süreç, belirlenen sayıda epoch boyunca veya belirli bir durdurma kriteri karşılanana kadar tekrarlanır. Bu süreç, modelin verilerdeki karmaşık ilişkileri öğrenmesini ve tahminlerini iyileştirmesini sağlar. Her epoch sonunda, model genellikle daha düşük bir kayıp değeri ile sonuçlanır, bu da daha iyi bir performans anlamına gelir.

1. **Tahminlerin Yapılması ve Değerlendirilmesi**

Son olarak, modelin tahminleri yapılır ve bu tahminler gerçek değerlerle karşılaştırılır. Bu, ileri besleme işleminin tekrarlanması ve çıktıların belirlenen eşik değere göre sınıflandırılması anlamına gelir.

El ile bu süreci gerçekleştirmek, özellikle geri yayılım ve optimizasyon aşamalarında, oldukça karmaşık matematiksel işlemleri içerir. Bu nedenle, genellikle bu tür işlemler için özel olarak tasarlanmış kütüphaneler (bu durumda Keras ve TensorFlow) kullanılır. Bu kütüphaneler, yüksek düzeyde optimizasyon ve verimlilik sağlayarak bu işlemleri çok daha hızlı ve hatasız bir şekilde gerçekleştirebilir.

Ölçeklendirme Yöntemleri

1. **StandardScaler (Standartlaştırma)**:
   - $z = \frac{x - \mu}{\sigma}$
   - Bu yöntem, verileri ortalama (μ) değerinden çıkarıp standart sapmaya (σ) böler. Sonuç olarak, veriler ortalaması 0 ve standart sapması 1 olacak şekilde standartlaştırılır.
   - Bu yöntem, veri dağılımını merkezlemek ve veriler arasındaki ölçek farkını gidermek için kullanılır.
2. **MinMaxScaler (Min-Max Normalizasyonu)**:
   - $X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}}$
   - Bu yöntem, verileri belirli bir aralığa (genellikle 0-1) dönüştürmek için kullanılır. Veriler, minimum (X_{min}) ve maksimum (X_{max}) değerlerine göre normalize edilir.
   - Bu yöntem, verilerin orijinal aralığını korumak istediğinizde veya bazı algoritmaların verilerin belirli bir aralıkta olmasını gerektirdiği durumlarda kullanılır.
   - Manuel olarak MinMaxScaler:

     min_val = X.values.min()

     max_val = X.values.max()

     X_scaled = (X - min_val) / (max_val - min_val)

     X_scaled = X_scaled.values
3. **MaxAbsScaler**:
   - $X_{\text{scaled}} = \frac{X}{|X_{\text{max}}|}$
   - Bu yöntem, verileri mutlak değerleriyle maksimum değere böler. Böylece, tüm veriler -1 ile 1 arasında olacak şekilde ölçeklenir.
   - Bu yöntem, verilerin orijinal işaretini korumak istediğinizde kullanışlıdır.
4. **RobustScaler**:
   - $X_{\text{scaled}} = \frac{X - \text{Median}}{\text{IQR}}$
   - Bu yöntem, verileri medyan (Median) değerinden çıkarıp IQR (çeyrekler arası aralık) değerine böler. IQR, verilerin yüzde 25 ile yüzde 75 arasındaki değerlerin aralığıdır.

- Bu yöntem, verilerdeki aykırı değerlere karşı daha dirençli olmasını sağlar.

5. **Normalizer**:

- L_2 normu için: $X_{\text{norm}} = \frac{X}{\sqrt{\sum X_i^2}}$
- Bu yöntem, her bir veri noktasının L_2 normuna (Öklidyen normu) göre normalize edilmesini sağlar. L_2 normu, bir vektörün bileşenlerinin karelerinin toplamının karekökü'dür.
- Bu yöntem, her bir veri noktasının uzunluğunu sabitlemek veya vektörlerin benzerliklerini hesaplarken kullanılmak istendiğinde kullanılır.

RobustScaler Ölçeklendirmesi

Adım 1: Medyan ve IQR Hesaplama

Feature1 için:

1. Önce verileri sıralayın: `[10, 11, 11, 13, 14, 19, 23, 26, 32, 40]`
2. Medyan (Q2): Bu durumda, 5. ve 6. değerlerin ortalaması alınır (çünkü 10 değer var ve bu çift sayıdır). Medyan = `(14 + 19) / 2 = 16.5`
3. Q1 (ilk çeyrek): İlk 5 değerin medyanı. Q1 = `11.5`
4. Q3 (üçüncü çeyrek): Son 5 değerin medyanı. Q3 = `25.25`
5. IQR = Q3 - Q1 = `25.25 - 11.5 = 13.75`

Adım 2: RobustScaler Ölçeklendirme

RobustScaler formülü:

$$\text{Yeni Değer} = \frac{\text{Orijinal Değer} - \text{Medyan}}{\text{IQR}}$$

Feature1 için Ölçeklendirme:

Örneğin, ilk değer için (10):

$$\frac{10 - 16.5}{13.75} = \frac{-6.5}{13.75} \approx -0.4727$$

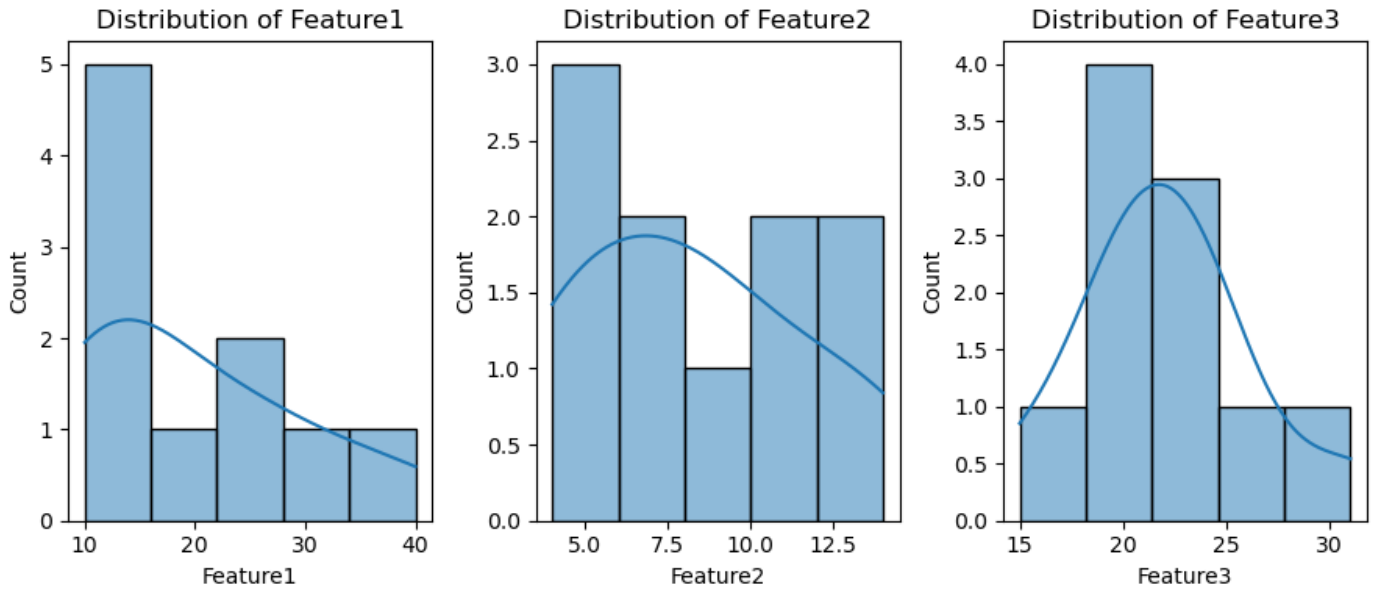
Feature2 ve Feature3 için de benzer şekilde ölçeklendirme yapılır.

In [14]:
```
x1 = np.array([10, 11, 11, 13, 14, 19, 23, 26, 32, 40])
q75, q25 = np.percentile(x1, [75, 25])
iqr = q75 - q25

x1_med = np.median(x1)

x1_scaled = (x1 - x1_med) / iqr
x1_scaled
```

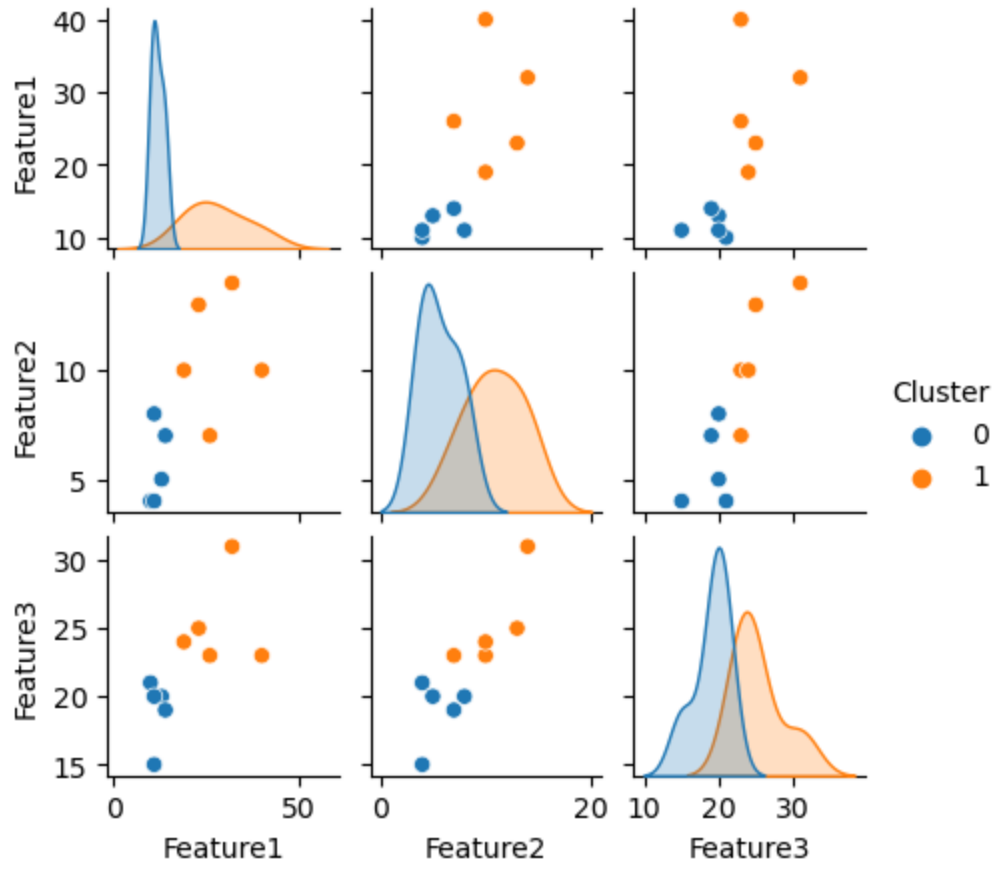
Out[14]:
```
array([-0.47272727, -0.4       , -0.4       , -0.25454545, -0.18181818,
        0.18181818,  0.47272727,  0.69090909,  1.12727273,  1.70909091])
```

In [15]:
```
plt.figure(figsize=(9, 4))
for i, feature in enumerate(['Feature1', 'Feature2', 'Feature3']):
    plt.subplot(1, 3, i+1)
    sns.histplot(df[feature], kde=True)
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()
```

```
sns.pairplot(df, hue='Cluster', height=1.5)
plt.show()
```

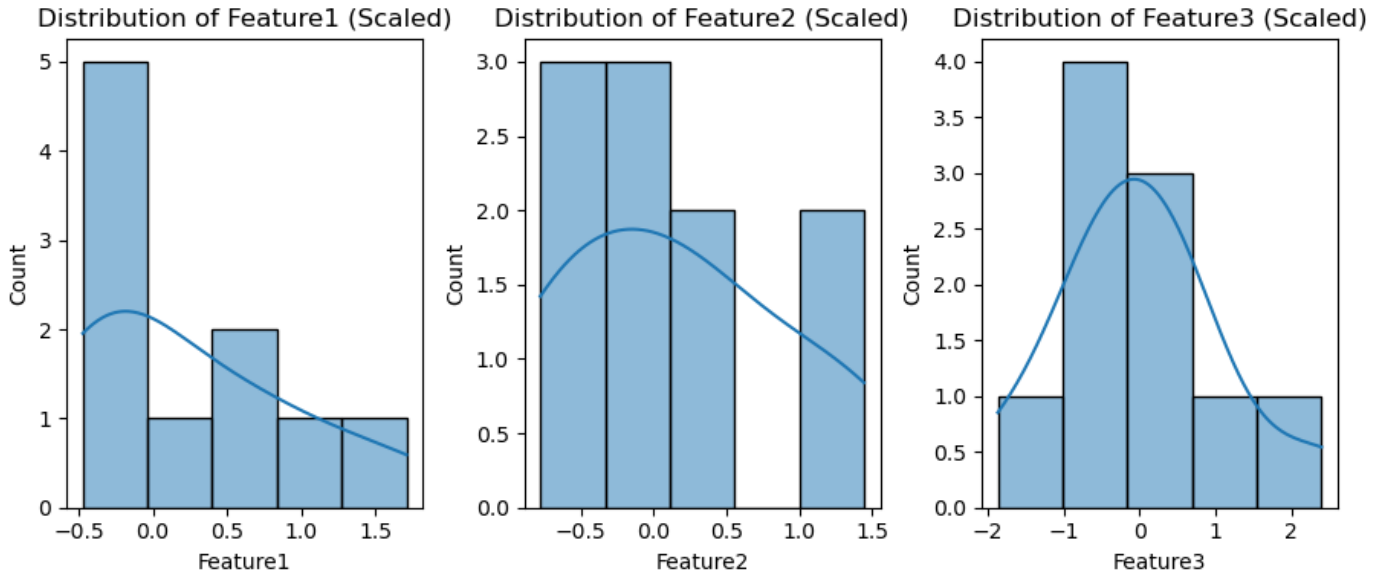



```
C:\Users\berka\.conda\envs\tensorflowkeras\lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight
```

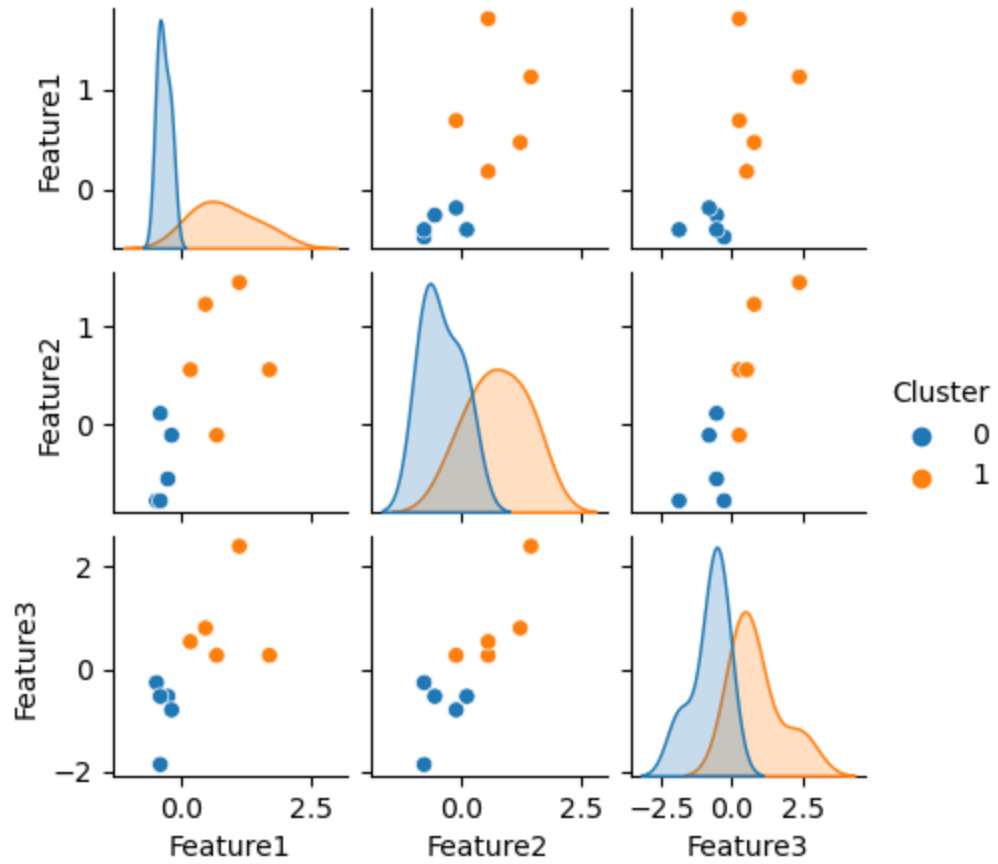


In [16]:
```
scaler = RobustScaler()
features = df[['Feature1', 'Feature2', 'Feature3']]
scaled_features = scaler.fit_transform(features)

scaled_df = pd.DataFrame(scaled_features, columns=['Feature1', 'Feature2', 'Feature3'])
scaled_df['Cluster'] = df['Cluster']
plt.figure(figsize=(9, 4))
for i, feature in enumerate(['Feature1', 'Feature2', 'Feature3']):
```

```
    plt.subplot(1, 3, i+1)
    sns.histplot(scaled_df[feature], kde=True)
    plt.title(f'Distribution of {feature} (Scaled)')
plt.tight_layout()
plt.show()

sns.pairplot(scaled_df, hue='Cluster', height=1.5)
plt.show()
```




```
C:\Users\berka\.conda\envs\tensorflowkeras\lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight
```



In [17]:
```
X = df[['Feature1', 'Feature2', 'Feature3']]
y = df['Cluster']
scaler = RobustScaler()
```

```
X_scaled = scaler.fit_transform(X)

X_scaled
```

Out[17]:
```
array([[-0.47272727, -0.77777778, -0.26666667],
       [-0.25454545, -0.55555556, -0.53333333],
       [-0.4       ,  0.11111111, -0.53333333],
       [-0.18181818, -0.11111111, -0.8       ],
       [-0.4       , -0.77777778, -1.86666667],
       [ 0.69090909, -0.11111111,  0.26666667],
       [ 1.70909091,  0.55555556,  0.26666667],
       [ 1.12727273,  1.44444444,  2.4       ],
       [ 0.47272727,  1.22222222,  0.8       ],
       [ 0.18181818,  0.55555556,  0.53333333]])
```

Yapay Sinir Ağı Modeli

Yapay Sinir Ağı Modelinin Matematiksel Kurulumu

Modelimiz üç katmandan oluşacak: bir giriş katmanı, bir gizli katman ve bir çıkış katmanı.

Modelin Yapısı

- **Giriş Katmanı:** Bu katman, özelliklerin (features) modelimize giriş noktasıdır. Örneğimizde 3 özellik (Feature1, Feature2, Feature3) bulunmaktadır. Dolayısıyla giriş katmanında 3 düğüm (nöron) olacaktır.
- **Gizli Katman:** Bu katman, giriş ve çıkış katmanları arasında yer alır ve karmaşık ilişkileri öğrenmek için kullanılır. Örneğimizde bu katmanda 10 düğüm bulunmaktadır.
- **Çıkış Katmanı:** Bu katman, modelin sonucunu verir. İkili sınıflandırma problemi için genellikle tek bir düğüm kullanılır ve bu düğüm, sigmoid aktivasyon fonksiyonu ile sonuç üretir.

Ağırlıklar ve Biaslar

- **Ağırlıklar:** Her bağlantı noktasındaki gücü temsil eder.
- **Biaslar:** Her nöronun aktivasyon eşiğini ayarlar.

Aktivasyon Fonksiyonları

- **Gizli Katman:** Genellikle ReLU (Rectified Linear Unit) veya benzeri bir aktivasyon fonksiyonu kullanılır.
- **Çıkış Katmanı:** İkili sınıflandırma için genellikle sigmoid aktivasyon fonksiyonu kullanılır.

Matematiksel Gösterim

- **Giriş Katmanı:**

$$X = [x_1, x_2, x_3]$$

- **Gizli Katman:**

$$Z = XW^{(1)} + B^{(1)}$$

ve

$$A = \text{ReLU}(Z)$$

  - $W^{(1)}$: Gizli katmana ait ağırlıklar
  - $B^{(1)}$: Gizli katmana ait biaslar

- A: Aktivasyon sonucu

- **Çıkış Katmanı:**

$$Z^{(2)} = AW^{(2)} + B^{(2)}$$

ve

$$\hat{y} = \sigma(Z^{(2)})$$

- $W^{(2)}$: Çıkış katmanına ait ağırlıklar
- $B^{(2)}$: Çıkış katmanına ait biaslar
- $\hat{y}$: Tahmin edilen değer
- σ: Sigmoid aktivasyon fonksiyonu

Eğitim Süreci

Eğitim sürecinde, gerçek değerlerle tahmin edilen değerler arasındaki farkı azaltacak şekilde ağırlıklar ve biaslar güncellenir. Bu süreç genellikle bir kayıp fonksiyonu (loss function) ve bir optimizasyon algoritması (örneğin, gradient descent) kullanılarak yapılır. Yapay sinir ağlarının eğitiminde kullanılan optimizasyon algoritmaları, modelin ağırlıklarını ve biaslarını ayarlayarak kayıp fonksiyonunu (loss function) minimize etmeye çalışır.

Stochastic Gradient Descent (SGD):

SGD, her bir güncellemede yalnızca bir örnek veya bir mini-batch üzerinden hesaplama yapar. Bu, hesaplama açısından daha verimli olup, genellikle daha hızlı yakınsama sağlar. SGD'nin varyasyonları arasında momentumun eklenmesi (SGD with Momentum) yer alır. Momentum, algoritmanın yerel minimumlara takılıp kalmamasını sağlayarak daha hızlı ve etkili bir şekilde global minimuma ulaşmasına yardımcı olur.

Ayrıca Adagrad veya RMSprop gibi optimizasyon teknikleride bulunmaktadır.

Modelde Kullanılan Optimizasyon Tekniği: Adam (Adaptive Moment Estimation):

Adam, hem momentum hem de özelleştirilmiş öğrenme hızları kavramlarını birleştirir. Bu algoritma, hem ilk an (ortalama gradyan) hem de ikinci an (gradyanların karesinin ortalaması) hesaplamalarını kullanır. Genellikle, farklı türdeki problemlerde iyi sonuçlar verdiği ve hiperparametre ayarlamasının nispeten daha kolay olduğu için popülerdir.

Adam algoritması, aşağıdaki gibi bir sürece sahiptir:

Momentum, önceki gradyanların hareketli ortalamasını kullanarak ağırlıkların güncellenmesini sağlar.

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$

Burada m_t momentumun t zaman adımındaki tahminidir, β_1 momentumun bozulma oranı, ve g_t ise t zaman adımındaki gradyandır. dam ayrıca gradyanların karelerinin hareketli ortalamasını da hesaplar.

$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$

Burada v_t ikinci anın t zaman adımındaki tahminidir, β_2 ikinci anın bozulma oranı. dam, başlangıçtaki düşük m_t ve v_t değerlerini düzeltmek için bir bias düzeltme mekanizması kullanır.

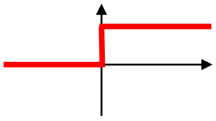
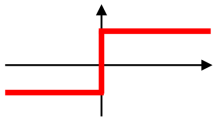
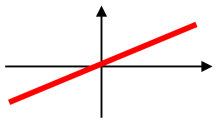
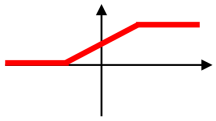
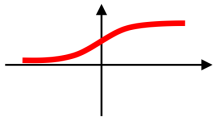
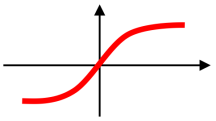
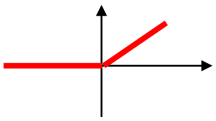
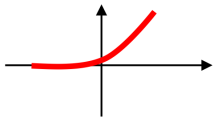
$\hat{m}_t = m_t / (1 - \beta_1^t)$

$\hat{v}_t = v_t / (1 - \beta_2^t)$

Ağırlıklar aşağıdaki formül kullanılarak güncellenir.

$$\theta_{t+1} = \theta_t - \eta/(\sqrt{\hat{v}_t} + \varepsilon)\hat{m}_t$$

Burada θ_{t+1} yeni parametre değeri, η öğrenme oranı, ve ε çok küçük bir sabit (sayısal kararlılık için).

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

> **Activation Functions for Artificial Neural Networks**

Matematiksel Hesaplama

Verilen bilgilere göre, modelin ilk katmanındaki ilk nöron için aktivasyon çıktısını hesaplayalım. Bu aktivasyon, ilk katmandaki her bir nöronun çıktılarıdır ve ikinci katmanın girdisi olarak kullanılacak.

İlk Örnek için Giriş Değerleri

İlk örnek: `[-0.47272727, -0.77777778, -0.26666667]`

İlk Katman Ağırlık ve Bias Değerleri (Model Üzerinden otomatik olarak çekildi)

Ağırlıklar (3x10 matris):

[[0.28296012, -0.05687276, 0.04556922, 0.3255316, -0.44156694, -0.0733676, 0.43049505, -0.06258934, 0.31083593, -0.35943946], [0.26263136, -0.42746422, 0.13826287, -0.34895825, 0.18902431, -0.04993188, 0.54785335, 0.548638, -0.49485925, 0.03004686], [-0.03421543, -0.73417526, -0.09849247, 0.27465415, 0.71510494, 0.60688436, -0.5720368, 0.2352761, -0.3921339, -0.0706273]]

Biaslar (10 elemanlı vektör):

[0.05250309, 0.06799143, -0.03498707, -0.05418956, 0.0515959, -0.03407031, 0.01480952, -0.0583354, -0.03062888, 0.06837697]

İlk Örnek Üzerinden Hesaplama

İlk nöron için (z) değerini hesaplayalım ve ReLU aktivasyon fonksiyonunu uygulayalım.

$$z_1^{[1]} = W_1^{[1]} \cdot X_1 + b_1^{[1]}$$

$= [0.28296012 \times (-0.47272727) + 0.26263136 \times (-0.77777778) - 0.03421543 \times (-0.26666667)]$
$+ 0.05250309$
$= -0.1337003 - 0.2042664 + 0.0091226 + 0.05250309 = -0.27634101$

ReLU aktivasyon fonksiyonunu uygulayalım:

$$a_1^{[1]} = \text{ReLU}(z_1^{[1]})$$

$$= \max(0, -0.27634101)$$

$$= 0$$

Bu, ilk nöronun ilk örnek için aktivasyon çıktısıdır ve bu değer 0 olarak hesaplanmıştır. Bu, nöronun ReLU aktivasyon fonksiyonu nedeniyle negatif veya sıfır olan bir girdiye sahip olduğunu gösterir.

Verilen aktivasyon çıktıları ve ikinci katmanın ağırlık ve bias değerlerini kullanarak, ikinci katman (çıkış katmanı) için hesaplama yapalım. Bu aktivasyon değerleri, ilk katmandaki her bir nöronun çıktılarıdır ve ikinci katmanın girdisi olarak kullanılacak.

Verilen Aktivasyon Çıktıları (İlk Örnek için)

[0.0, 0.623129, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.31189007, 0.23375799]

İkinci Katman Ağırlık ve Bias Değerleri

Ağırlıklar (10 elemanlı vektör):

[0.7697679, -0.7555279, -0.34198648, -0.3977732, 0.5127915, 0.01032777, 0.6190769, -0.57430345, 0.3178662, -0.6981783]

Bias (tek bir değer):

0.00639721

İkinci Katman için Hesaplama

İkinci katman, bu aktivasyon çıktılarını alır, her biri için bir ağırlıkla çarpar, toplar ve bir bias ekler. Sonra, bu toplam üzerinde sigmoid aktivasyon fonksiyonunu uygular.

Z Değerinin Hesaplanması

Çıkış nöronu için (z) değerini hesaplayalım:

$$z^{[2]} = W^{[2]} \cdot a^{[1]} + b^{[2]}$$

$$= [0.7697679 \times 0.0 - 0.7555279 \times 0.623129 - 0.34198648 \times 0.0 - 0.3977732 \times 0.0 + 0.5127915 \times 0.0 + 0.01032777 \times 0.0 + 0.6190769 \times 0.0 - 0.57430345 \times 0.0 + 0.3178662 \times 0.31189007 - 0.6981783 \times 0.23375799] + 0.00639721$$

$$= 0 - 0.470802 - 0 - 0 + 0 + 0 + 0 - 0 + 0.099177 - 0.163428 + 0.00639721 = -0.52865579$$

Sigmoid Aktivasyon Fonksiyonu

Sigmoid fonksiyonu $\sigma(x) = \frac{1}{1+e^{-x}}$ olarak tanımlanır. Dolayısıyla:

$$\hat{y} = \sigma(z^{[2]})$$

$$= \frac{1}{1 + e^{-(-0.52865579)}}$$

$$= \frac{1}{1 + e^{0.52865579}}$$

$$\approx \frac{1}{1 + 1.697}$$

$$\approx 0.37087628$$

Bu, modelin ilk örnek için tahmin ettiği çıktıdır ve bu hesaplama, modelinizin eğitim sürecinde öğrenilen gerçek ağırlık ve bias değerleriyle yapılan gerçek bir tahmindir. Bu tahmin değerinde 0.5 değerinden küçük olan değerler 0, büyük olan değerler 1 olarak atanır ve sınıflandırma işlemi gerçekleştirilir.

In [18]:
```
seed_value= 123
python_random.seed(seed_value)
np.random.seed(seed_value)
tf.random.set_seed(seed_value)

model = Sequential()
model.add(Dense(10, input_dim=3, activation='relu', name='relu_layer'))  # Giriş katmanı
model.add(Dense(1, activation='sigmoid'))  # Çıkış katmanı


model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X_scaled, y, epochs=10, batch_size=1)
predictions = model.predict(X_scaled)

layer_name = 'relu_layer'
intermediate_layer_model = Model(inputs=model.input,
                                 outputs=model.get_layer(layer_name).output)

intermediate_output = intermediate_layer_model.predict(X_scaled)

print(intermediate_output[0])
predictions
```

```
Epoch 1/10
10/10 [==============================] - 1s 2ms/step - loss: 0.6794 - accuracy: 0.6000
Epoch 2/10
10/10 [==============================] - 0s 1ms/step - loss: 0.6513 - accuracy: 0.6000
Epoch 3/10
10/10 [==============================] - 0s 1ms/step - loss: 0.6283 - accuracy: 0.8000
Epoch 4/10
```

```
10/10 [==============================] - 0s 1ms/step - loss: 0.6056 - accuracy: 0.9000
Epoch 5/10
10/10 [==============================] - 0s 1ms/step - loss: 0.5837 - accuracy: 0.9000
Epoch 6/10
10/10 [==============================] - 0s 1ms/step - loss: 0.5626 - accuracy: 1.0000
Epoch 7/10
10/10 [==============================] - 0s 2ms/step - loss: 0.5441 - accuracy: 1.0000
Epoch 8/10
10/10 [==============================] - 0s 1ms/step - loss: 0.5251 - accuracy: 1.0000
Epoch 9/10
10/10 [==============================] - 0s 2ms/step - loss: 0.5072 - accuracy: 1.0000
Epoch 10/10
10/10 [==============================] - 0s 1ms/step - loss: 0.4911 - accuracy: 1.0000
1/1 [==============================] - 0s 99ms/step
1/1 [==============================] - 0s 51ms/step
[0.         0.623129   0.         0.         0.         0.
 0.         0.         0.31189007 0.23375799]
```

Out[18]:

```
array([[0.37087628],
       [0.3685483 ],
       [0.40739173],
       [0.4075819 ],
       [0.27389655],
       [0.5399527 ],
       [0.6945853 ],
       [0.6068378 ],
       [0.62437236],
       [0.56759953]], dtype=float32)
```

In [19]:

```
# Tahminleri ikili formata çevirme (0 veya 1)
predictions = [1 if p > 0.5 else 0 for p in predictions]
for i in range(len(predictions)):
    print(f"Gerçek: {y[i]}, Tahmin: {predictions[i]}")
```

```
Gerçek: 0, Tahmin: 0
Gerçek: 0, Tahmin: 0
Gerçek: 0, Tahmin: 0
Gerçek: 0, Tahmin: 0
Gerçek: 0, Tahmin: 0
Gerçek: 1, Tahmin: 1
Gerçek: 1, Tahmin: 1
Gerçek: 1, Tahmin: 1
Gerçek: 1, Tahmin: 1
Gerçek: 1, Tahmin: 1
```

In [20]:

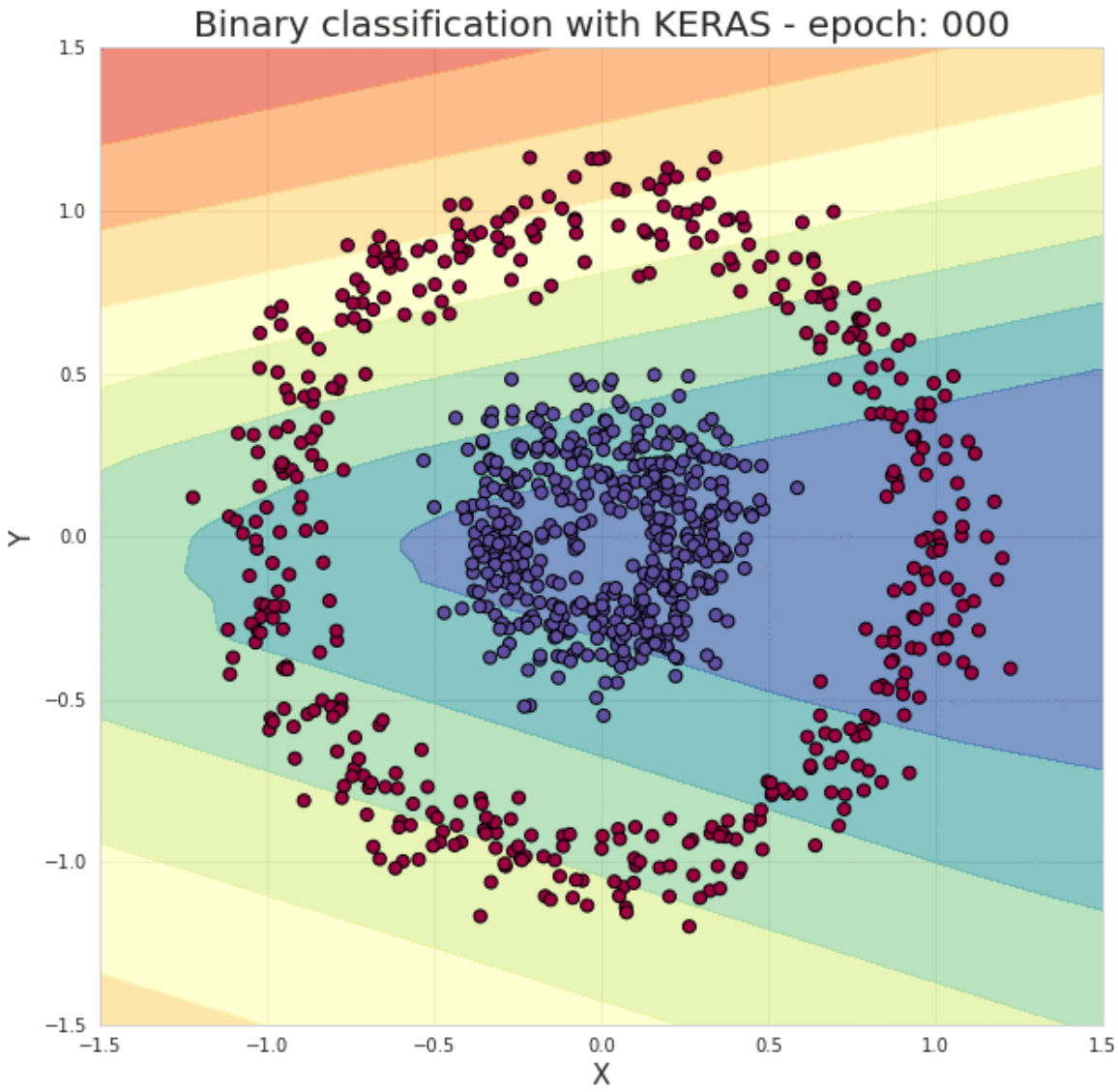
```
for layer in model.layers:
    weights, biases = layer.get_weights()
    print(f"Katman: {layer.name}")
    print("Ağırlıklar:\n", weights)
    print("Biaslar:\n", biases)
    print("\n")
```

```
Katman: relu_layer
Ağırlıklar:
 [[ 0.28296012 -0.05687276  0.04556922  0.3255316  -0.44156694 -0.0733676
   0.43049505 -0.06258934  0.31083593 -0.35943946]
 [ 0.26263136 -0.42746422  0.13826287 -0.34895825  0.18902431 -0.04993188
   0.54785335  0.548638   -0.49485925  0.03004686]
 [-0.03421543 -0.73417526 -0.09849247  0.27465415  0.71510494  0.60688436
  -0.5720368   0.2352761  -0.3921339  -0.0706273 ]]
Biaslar:
 [ 0.05250309  0.06799143 -0.03498707 -0.05418956  0.0515959  -0.03407031
  0.01480952 -0.0583354  -0.03062888  0.06837697]


Katman: dense
Ağırlıklar:
```

```
 [[ 0.7697679 ]
 [-0.7555279 ]
 [-0.34198648]
 [-0.3977732 ]
 [ 0.5127915 ]
 [ 0.01032777]
 [ 0.6190769 ]
 [-0.57430345]
 [ 0.3178662 ]
 [-0.6981783 ]]
Biaslar:
 [0.00639721]
```

epochs parametresi (Dönem Sayısı), tüm eğitim veri setinin yapay sinir ağı tarafından kaç kez tamamen işleneceğini belirtir. Bir epoch, veri setinin başından sonuna kadar olan tüm verilerin ağ üzerinden bir kez geçirilmesi sürecidir. Daha yüksek epoch sayısı, genellikle modelin eğitim verilerini daha iyi öğrenmesine yardımcı olur. Ancak, çok fazla epoch, modelin eğitim verilerine aşırı uyum sağlamasına (overfitting) ve genellemeyi kaybetmesine yol açabilir.

batch_size (Yığın Boyutu) parametresi, her bir eğitim iterasyonunda ağa beslenen örnek sayısını belirtir. Yani, veri seti bu belirtilen boyuttaki yığınlar (batches) halinde modele verilir. Örneğin, batch_size=1 olduğunda, model her seferinde tek bir veri noktası üzerinden eğitilir. Bu, genellikle daha yavaş bir eğitim süreci anlamına gelir ve her örnek için ayrı ayrı ağırlık güncellemeleri yapılır. Daha büyük bir batch size, genellikle eğitim sürecini hızlandırır, çünkü her iterasyonda daha fazla veri noktası işlenir. Ancak, çok büyük batch boyutları bellek sorunlarına yol açabilir ve modelin genelleme yeteneğini etkileyebilir.



Epoch Training

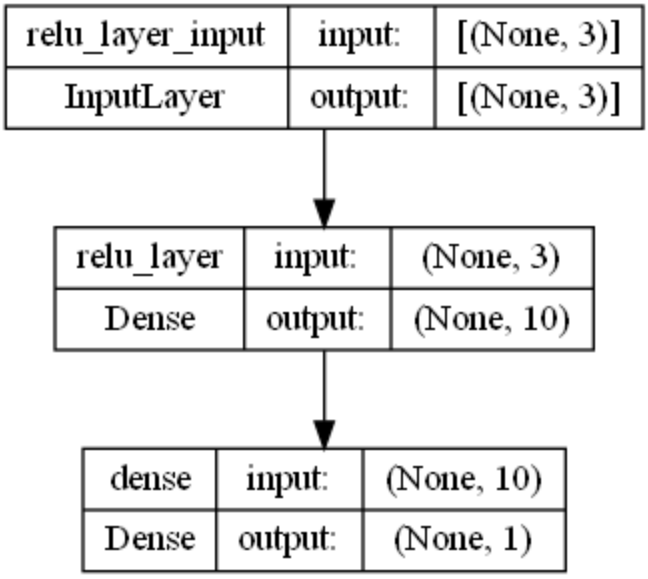
In [21]:
```
# Modelin Özeti
model.summary()
for layer in model.layers:
    print(f"Katman Adı: {layer.name}, Katman Tipi: {type(layer).__name__}, Çıktı Şekli:
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 relu_layer (Dense)          (None, 10)                40

 dense (Dense)               (None, 1)                 11

=================================================================
Total params: 51
Trainable params: 51
Non-trainable params: 0
_________________________________________________________________
Katman Adı: relu_layer, Katman Tipi: Dense, Çıktı Şekli: (None, 10), Parametre Sayısı: 4
```

```
0
Katman Adı: dense, Katman Tipi: Dense, Çıktı Şekli: (None, 1), Parametre Sayısı: 11
```

In [22]:
```
plot_model(model, to_file='model.png', show_shapes=True, show_layer_names=True)
Image(filename='model.png')
```

Out[22]:



Yapay Sinir Ağı Modelleri

1. **Zaman Serileri Analizi**

Uzun Kısa Süreli Bellek (LSTM) Ağları, Gate Recurrent Unit (GRU) Ağları, Evrişimsel Sinir Ağları (CNN)

1. **Doğal Dil İşleme (NLP)**

Transformer Modeller (Generative Pretrained Transformer), Tekrarlayan Sinir Ağları (RNN) ve Türevleri, Evrişimsel Sinir Ağları (CNN)

1. **Görüntü İşleme**

Evrişimsel Sinir Ağları (CNN), Yapay Sinir Ağları (ANN), SegNet

1. **Regresyon Tahmini**

Çok Katmanlı Algılayıcılar (MLP - Multi-Layer Perceptrons), Radial Basis Function (RBF) Ağları, Genelleştirilmiş Regresyon Sinir Ağları (GRNN - Generalized Regression Neural Networks)

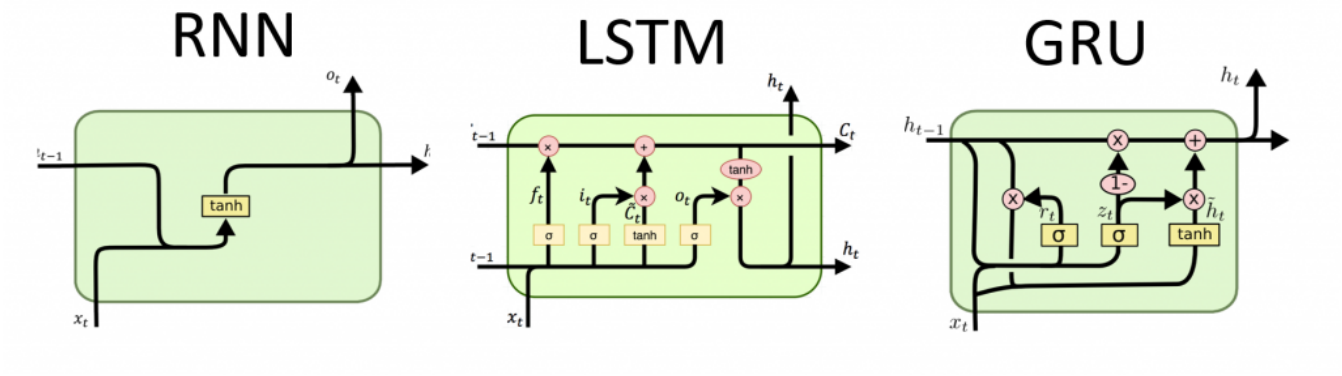
1. **Sınıflandırma**

Konvolüsyonel Sinir Ağları (CNN - Convolutional Neural Networks), Tekrarlayan Sinir Ağları (RNN - Recurrent Neural Networks), Derin Gelişimsel Ağlar (DBN - Deep Belief Networks)

1. **Kümeleme**

Kendi Kendini Organize Eden Haritalar (SOM - Self-Organizing Maps), Uyarlamalı Rezonans Teorisi (ART - Adaptive Resonance Theory), Otoenkoderler (Autoencoders)

Resim referansı: Buraya Bakın



Daha fazla bilgi edinmek için aşağıdaki referans bağlantılarını incelenebilir:

- GRU hakkında kapsamlı rehber
- LSTM hakkında kapsamlı bir rehber

Gerçek Veri Üzerinden Optics Kümeleme Ve Yapay Sinir Ağı Sınıflandırması

Açıklama Yazar: Kun Zhang, Wei Fan, XiaoJing Yuan

Kaynak: UCI

Lütfen alıntı yapın:

Çarpık önyargılı stokastik ozon günlerinin tahmini: analizler, çözümler ve ötesi, Bilgi ve Bilgi Sistemleri, Cilt 14, No. 3, 2008.

1 . Özet: Bu koleksiyonda iki yer ozon seviyesi veri seti bulunmaktadır. Bunlardan biri sekiz saatlik pik seti (eighthr.data), diğeri ise bir saatlik pik setidir (onehr.data). Bu veriler 1998-2004 yılları arasında Houston, Galveston ve Brazoria bölgelerinden toplanmıştır.

Kaynak: Kun Zhang, zhang.kun05 '@' gmail.com, Bilgisayar Bilimleri Bölümü, Xavier Lousiana Üniversitesi Wei Fan, wei.fan '@' gmail.com, IBM T.J.Watson Research XiaoJing Yuan, xyuan '@' uh.edu, Mühendislik Teknolojisi Bölümü, Teknoloji Koleji, Houston Üniversitesi

Veri Seti Bilgileri: T ile başlayan tüm nitelikler gün boyunca farklı zamanlarda ölçülen sıcaklık anlamına gelir; WS ile başlayanlar ise çeşitli zamanlarda rüzgar hızını gösterir.

WSR_PK: sürekli. peek rüzgar hızı -- sonuç (rüzgar vektörünün ortalaması anlamına gelir) WSR_AV: sürekli. ortalama rüzgar hızı T_PK: sürekli. Tepe T T_AV: sürekli. Ortalama T T85: sürekli. 850 hpa seviyesinde (veya yaklaşık 1500 m yükseklikte) T RH85: sürekli. 850 hpa'daki Bağıl Nem U85: sürekli. (U rüzgarı - 850 hpa'da doğu-batı yönlü rüzgar) V85: sürekli. V rüzgarı - 850 HT85'te N-S yönündeki rüzgar: sürekli. 850 hpa'daki jeopotansiyel yükseklik, alçak irtifadaki yükseklikle yaklaşık olarak aynıdır T70: sürekli. 700 hpa seviyesinde T (kabaca 3100 m yükseklik)

RH70: sürekli. U70: sürekli. V70: sürekli. HT70: sürekli.

T50: sürekli. 500 hpa seviyesinde T (kabaca 5500 m yükseklikte)

RH50: sürekli. U50: sürekli. V50: sürekli. HT50: sürekli.

KI: sürekli. K-Index [Web Bağlantısı] TT: sürekli. T-Totals [Web Bağlantısı] SLP: sürekli. Deniz seviyesi basıncı SLP_: sürekli. Önceki güne göre SLP değişimi

Precp: sürekli. -- yağış

Özellik Bilgileri: Aşağıda, Teksas Çevre Kalitesi Komisyonu (TCEQ) tarafından oldukça değer verilen en önemli birkaç özelliğe ilişkin spesifikasyonlar yer almaktadır. Daha fazla ayrıntı ilgili iki belgede bulunabilir.

O 3 - Yerel ozon pik tahmini Upwind - Upwind ozon arka plan seviyesi EmFactor - Öncü emisyonlarla ilgili faktör Tmax - F derece cinsinden maksimum sıcaklık Tb - Net ozon üretiminin başladığı temel sıcaklık (50 F) SRd - Gün için toplam güneş radyasyonu WSa - Gün doğumuna yakın rüzgar hızı (09-12 UTC tahmin modu kullanılarak) WSp - Gün ortasında rüzgar hızı (15-21 UTC tahmin modu kullanılarak)

İlgili Makaleler: Forecasting skewed biased stochastic ozone days: analyses, solutions and beyond, Knowledge and Information Systems, Vol. 14, No. 3, 2008.

Veri kümesi, kullanımı ve birçok son teknoloji yöntem kullanılarak yapılan çeşitli deneyler (hem çapraz doğrulama hem de akış) hakkındaki ayrıntıları tartışmaktadır. Makalenin daha kısa bir versiyonu (yukarıdaki dergi makalesi gibi bazı ayrıntılı deneyler içermemektedir) şurada yer almaktadır: Çarpık Yanlı Stokastik Ozon Günlerinin Tahmini: Analizler ve Çözümler. ICDM 2006: 753-764

https://www.openml.org/search?type=data&sort=runs&status=active&qualities.NumberOfClasses=%3D_2&qualities.NumberOfFeatures=betwee

In [23]:
```
dosya_yolu = 'ozone.csv'
sutun_isimleri = ['v' + str(i) for i in range(1, 74)]
df = pd.read_csv(dosya_yolu, header=None, names=sutun_isimleri)
print(df.head())
```

```
    v1   v2   v3   v4   v5   v6   v7   v8   v9  v10  ...       v64        v65  \
0  0.8  1.8  2.4  2.1  2.0  2.1  1.5  1.7  1.9  2.3  ...  0.150000  10.670000
1  2.8  3.2  3.3  2.7  3.3  3.2  2.9  2.8  3.1  3.4  ...  0.480000   8.390000
2  2.9  2.8  2.6  2.1  2.2  2.5  2.5  2.7  2.2  2.5  ...  0.600000   6.940000
3  4.7  3.8  3.7  3.8  2.9  3.1  2.8  2.5  2.4  3.1  ...  0.490000   8.730000
4  2.6  2.1  1.6  1.4  0.9  1.5  1.2  1.4  1.3  1.4  ...  0.304716   9.872418

         v66          v67        v68        v69           v70        v71  \
0  -1.560000  5795.000000 -12.100000  17.900000  10330.000000 -55.000000
1   3.840000  5805.000000  14.050000  29.000000  10275.000000 -55.000000
2   9.800000  5790.000000  17.900000  41.300000  10235.000000 -40.000000
3  10.540000  5775.000000  31.150000  51.700000  10195.000000 -40.000000
4   0.830116  5818.821222  10.511051  37.388335  10164.198442  -0.119949

    v72  v73
0  0.00    1
1  0.00    1
2  0.00    1
3  2.08    1
4  0.58    1

[5 rows x 73 columns]
```

In [24]:
```
df_v73 = df[['v73']]
df_v73['v73'] = df_v73['v73'].replace(2, 0)

df = df.drop(['v73'], axis=1)
for col in df.columns:
```

```
    if df[col].dtype == 'object':
        df[col] = df[col].replace('[\$,]', '', regex=True).astype(float, errors='ignore'
df.head()
```

```
C:\Users\berka\AppData\Local\Temp\ipykernel_3496\3270934868.py:2: SettingWithCopyWarnin
g:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
```

Out[24]:

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	...	v63	v64	v65	v66	v67	
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	-15.50000	0.150000	10.670000	-1.560000	5795.000000	-12.
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	-14.50000	0.480000	8.390000	3.840000	5805.000000	14.
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	-15.90000	0.600000	6.940000	9.800000	5790.000000	17.
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	-16.80000	0.490000	8.730000	10.540000	5775.000000	31.
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	-10.51141	0.304716	9.872418	0.830116	5818.821222	10.

5 rows × 72 columns

In [25]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2534 entries, 0 to 2533
Data columns (total 72 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   v1      2534 non-null   float64
 1   v2      2534 non-null   float64
 2   v3      2534 non-null   float64
 3   v4      2534 non-null   float64
 4   v5      2534 non-null   float64
 5   v6      2534 non-null   float64
 6   v7      2534 non-null   float64
 7   v8      2534 non-null   float64
 8   v9      2534 non-null   float64
 9   v10     2534 non-null   float64
 10  v11     2534 non-null   float64
 11  v12     2534 non-null   float64
 12  v13     2534 non-null   float64
 13  v14     2534 non-null   float64
 14  v15     2534 non-null   float64
 15  v16     2534 non-null   float64
 16  v17     2534 non-null   float64
 17  v18     2534 non-null   float64
 18  v19     2534 non-null   float64
 19  v20     2534 non-null   float64
 20  v21     2534 non-null   float64
 21  v22     2534 non-null   float64
 22  v23     2534 non-null   float64
 23  v24     2534 non-null   float64
 24  v25     2534 non-null   float64
 25  v26     2534 non-null   float64
 26  v27     2534 non-null   float64
 27  v28     2534 non-null   float64
 28  v29     2534 non-null   float64
 29  v30     2534 non-null   float64
```

```
 30  v31     2534 non-null   float64
 31  v32     2534 non-null   float64
 32  v33     2534 non-null   float64
 33  v34     2534 non-null   float64
 34  v35     2534 non-null   float64
 35  v36     2534 non-null   float64
 36  v37     2534 non-null   float64
 37  v38     2534 non-null   float64
 38  v39     2534 non-null   float64
 39  v40     2534 non-null   float64
 40  v41     2534 non-null   float64
 41  v42     2534 non-null   float64
 42  v43     2534 non-null   float64
 43  v44     2534 non-null   float64
 44  v45     2534 non-null   float64
 45  v46     2534 non-null   float64
 46  v47     2534 non-null   float64
 47  v48     2534 non-null   float64
 48  v49     2534 non-null   float64
 49  v50     2534 non-null   float64
 50  v51     2534 non-null   float64
 51  v52     2534 non-null   float64
 52  v53     2534 non-null   float64
 53  v54     2534 non-null   float64
 54  v55     2534 non-null   float64
 55  v56     2534 non-null   float64
 56  v57     2534 non-null   float64
 57  v58     2534 non-null   float64
 58  v59     2534 non-null   float64
 59  v60     2534 non-null   float64
 60  v61     2534 non-null   float64
 61  v62     2534 non-null   float64
 62  v63     2534 non-null   float64
 63  v64     2534 non-null   float64
 64  v65     2534 non-null   float64
 65  v66     2534 non-null   float64
 66  v67     2534 non-null   float64
 67  v68     2534 non-null   float64
 68  v69     2534 non-null   float64
 69  v70     2534 non-null   float64
 70  v71     2534 non-null   float64
 71  v72     2534 non-null   float64
dtypes: float64(72)
memory usage: 1.4 MB
```

In [26]:
```
missing_values = df.isnull().sum()
missing_values
```

Out[26]:
```
v1     0
v2     0
v3     0
v4     0
v5     0
      ..
v68    0
v69    0
v70    0
v71    0
v72    0
Length: 72, dtype: int64
```

In [27]:
```
df
```

Out[27]:

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	...	v63	v64	v65	v66	v67
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	-15.50000	0.150000	10.670000	-1.560000	5795.000000

1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	-14.50000	0.480000	8.390000	3.840000	5805.000000
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	-15.90000	0.600000	6.940000	9.800000	5790.000000
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	-16.80000	0.490000	8.730000	10.540000	5775.000000
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	-10.51141	0.304716	9.872418	0.830116	5818.821222
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2529	0.3	0.4	0.5	0.5	0.2	0.3	0.4	0.4	1.3	2.2	...	-12.40000	0.070000	7.930000	-4.410000	5800.000000
2530	1.0	1.4	1.1	1.7	1.5	1.7	1.8	1.5	2.1	2.4	...	-12.00000	0.040000	5.950000	-1.140000	5845.000000
2531	0.8	0.8	1.2	0.9	0.4	0.6	0.8	1.1	1.5	1.5	...	-11.80000	0.060000	7.800000	-0.640000	5845.000000
2532	1.3	0.9	1.5	1.2	1.6	1.8	1.1	1.0	1.9	2.0	...	-10.80000	0.250000	7.720000	-0.890000	5845.000000
2533	1.5	1.3	1.8	1.4	1.2	1.7	1.6	1.4	1.6	3.0	...	-11.90000	0.540000	13.070000	9.150000	5820.000000

2534 rows × 72 columns

In [28]:
```
# Çok sayıda karakter içeren değişkenler için Frequency Encoding yapma
#freq_encoding_DRG = df['DRG Definition'].value_counts().to_dict()
#freq_encoding_ProviderName = df['Provider Name'].value_counts().to_dict()
#freq_encoding_ProviderCity = df['Provider City'].value_counts().to_dict()

#df['DRG Definition'] = df['DRG Definition'].map(freq_encoding_DRG)
#df['Provider Name'] = df['Provider Name'].map(freq_encoding_ProviderName)
#df['Provider City'] = df['Provider City'].map(freq_encoding_ProviderCity)
```

In [29]:
```
df.describe()
```

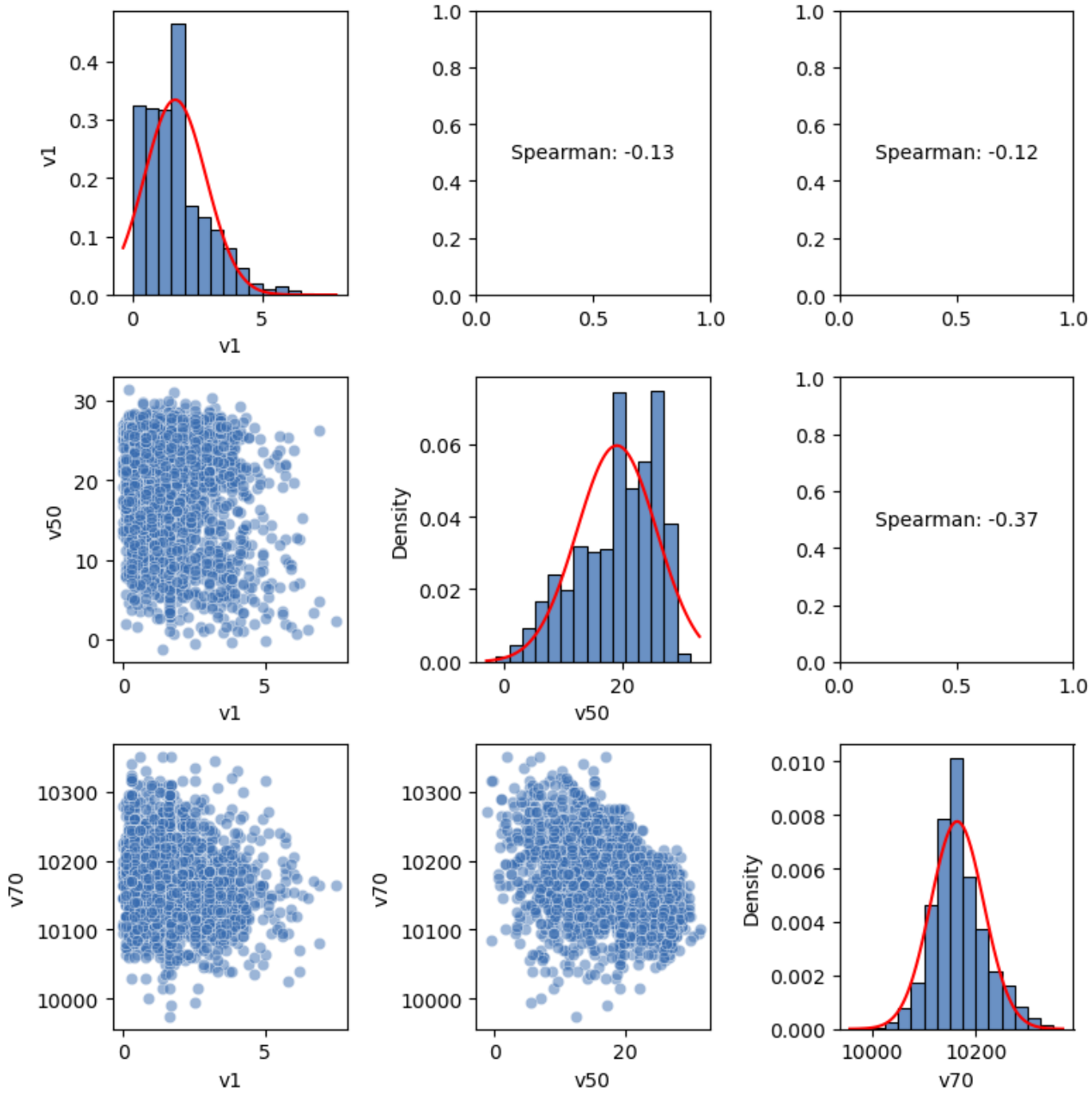
Out[29]:

	v1	v2	v3	v4	v5	v6	v7	v8
count	2534.000000	2534.000000	2534.000000	2534.000000	2534.000000	2534.000000	2534.000000	2534.000000
mean	1.640179	1.586351	1.545580	1.526405	1.522624	1.542417	1.637896	2.047127
std	1.194568	1.191320	1.165651	1.133908	1.127523	1.102557	1.092811	1.092315
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.800000	0.700000	0.700000	0.700000	0.700000	0.700000	0.800000	1.300000
50%	1.600000	1.500000	1.400000	1.400000	1.400000	1.400000	1.600000	2.047127
75%	2.200000	2.100000	2.100000	2.075000	2.000000	2.000000	2.100000	2.600000
max	7.500000	7.700000	7.100000	7.300000	7.200000	7.400000	7.400000	7.500000

8 rows × 72 columns

In [30]:
```
selected_columns = ['v1', 'v50', 'v70']
subplot_size = len(selected_columns)

fig, axes = plt.subplots(subplot_size, subplot_size, figsize=(8, 8), tight_layout=True)

for i in range(subplot_size):
    for j in range(subplot_size):
        if i > j:
            sns.scatterplot(x=selected_columns[j], y=selected_columns[i], data=df, ax=ax
        elif i == j:
            sns.histplot(df[selected_columns[j]], ax=axes[i, j], bins=15, color="#386cb0
            mean = df[selected_columns[j]].mean()
            std = df[selected_columns[j]].std()
```

```
            xmin, xmax = axes[i, j].get_xlim()
            x = np.linspace(xmin, xmax, 100)
            y = stats.norm.pdf(x, mean, std)
            axes[i, j].plot(x, y, color="red")
        elif i < j:
            spearman_corr = df[selected_columns[i]].corr(df[selected_columns[j]], method
            axes[i, j].annotate(f"Spearman: {spearman_corr:.2f}", xy=(0.5, 0.5), xycoord

        if i == subplot_size - 1:
            axes[i, j].set_xlabel(selected_columns[j])
        if j == 0:
            axes[i, j].set_ylabel(selected_columns[i])

plt.show()
```



min_samples: Bu değer ne kadar yüksek olursa, algoritma o kadar yüksek yoğunluklu kümeleri tespit eder. Düşük bir değer, daha fazla sayıda küme bulunmasına yol açabilir.

max_eps: Bu değer çok küçükse, algoritma daha fazla sayıda küme bulabilir. Çok büyük bir değer, kümelerin birleşmesine yol açabilir.

Kümeleme Karşılaştırma Metrikleri

Silhouette Skoru

Bu metrik, kümeleme sonuçlarının kalitesini ölçmek için kullanılır. Her bir veri noktasının kendi kümesi içindeki benzerliği ve diğer kümelere olan farklılığı dikkate alarak bir skor hesaplar.

$s(i) = \frac{b(i)-a(i)}{\max\{a(i),b(i)\}}$

- Burada (a(i)) i'nin ait olduğu kümeye ait diğer noktalarla olan ortalama uzaklığı, $b(i)$ ise i'nin en yakın kümeye ait noktalarla olan ortalama uzaklığını ifade eder.

Rand İndeksi (ARI-Adjusted Rand Index)

Bu metrik, kümeleme sonuçlarının bir altın standartla karşılaştırılması için kullanılır. Kümeleme sonuçları ile gerçek sınıflandırma etiketleri arasındaki uyumu ölçer.

$RI = \frac{TP+TN}{TP+FP+FN+TN}$

- Burada TP doğru pozitif, TN doğru negatif, FP yanlış pozitif, FN yanlış negatif sayılarını ifade eder.

Normalleştirilmiş Karşılıklı Bilgi (NMI-Normalized Mutual Information)

Bu metrik, kümeleme sonuçlarının gerçek sınıflandırma etiketleriyle ne kadar uyumlu olduğunu ölçer. İki kümeleme sonucu arasındaki bilgi paylaşımını hesaplar.

$NMI(U,V) = \frac{2 \cdot I(U;V)}{H(U)+H(V)}$

- Burada (I(U; V)) iki kümeleme arasındaki karşılıklı bilgi miktarını, $H(U)$ ve $H(V)$ ise sırasıyla U ve V kümelerinin entropilerini ifade eder.

Purity

Bu metrik, küme merkezlerinin gerçek sınıflandırma etiketleriyle ne kadar uyumlu olduğunu ölçer. Her bir kümenin en yaygın sınıf etiketiyle ne kadar uyumlu olduğunu hesaplar.

$Purity(\Omega, C) = \frac{1}{N} \sum_k \max_j |\omega_k \cap c_j|$

- Burada Ω kümeleme sonucunu, C gerçek sınıflandırmaları, ω_k kümeleme sonucundaki k. kümeyi, c_j gerçek sınıflandırmadaki j. sınıfı, ve N toplam veri noktası sayısını ifade eder.

In [31]:
```
df.head()
```

Out[31]:

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	...	v63	v64	v65	v66	v67	
0	0.8	1.8	2.4	2.1	2.0	2.1	1.5	1.7	1.9	2.3	...	-15.50000	0.150000	10.670000	-1.560000	5795.000000	-12.
1	2.8	3.2	3.3	2.7	3.3	3.2	2.9	2.8	3.1	3.4	...	-14.50000	0.480000	8.390000	3.840000	5805.000000	14.
2	2.9	2.8	2.6	2.1	2.2	2.5	2.5	2.7	2.2	2.5	...	-15.90000	0.600000	6.940000	9.800000	5790.000000	17.
3	4.7	3.8	3.7	3.8	2.9	3.1	2.8	2.5	2.4	3.1	...	-16.80000	0.490000	8.730000	10.540000	5775.000000	31.
4	2.6	2.1	1.6	1.4	0.9	1.5	1.2	1.4	1.3	1.4	...	-10.51141	0.304716	9.872418	0.830116	5818.821222	10.

5 rows × 72 columns

In [32]:
```
scaler = RobustScaler()
robust_scaled_df = scaler.fit_transform(df)
robust_scaled_df = pd.DataFrame(robust_scaled_df, columns=df.columns)
df = robust_scaled_df
```

In [33]:
```
df.head()
```

Out[33]:

	v1	v2	v3	v4	v5	v6	v7	v8	v9	v10	...	
0	-0.571429	0.214286	0.714286	0.509091	0.461538	0.538462	-0.076923	-0.267021	-0.456455	-0.391220	...	-0.8
1	0.857143	1.214286	1.357143	0.945455	1.461538	1.384615	1.000000	0.579133	0.400688	0.394494	...	-0.7
2	0.928571	0.928571	0.857143	0.509091	0.615385	0.846154	0.692308	0.502210	-0.242169	-0.248363	...	-0.9
3	2.214286	1.642857	1.642857	1.745455	1.153846	1.307692	0.923077	0.348364	-0.099312	0.180209	...	-1.1
4	0.714286	0.428571	0.142857	0.000000	-0.384615	0.076923	-0.307692	-0.497790	-0.885026	-1.034077	...	-0.0

5 rows × 72 columns

In [34]:
```
df_train, df_test = train_test_split(df, test_size=0.3, random_state=42)

param_combinations = [(min_samples, max_eps) for min_samples in [50, 100, 200, 300, 400]
best_score = -1
best_params = None

for min_samples, max_eps in param_combinations:
    model = OPTICS(min_samples=min_samples, max_eps=max_eps, metric='euclidean')
    df_train['target'] = model.fit_predict(df_train)
    score = silhouette_score(df_train, df_train['target'])
    print(f"Min_samples: {min_samples}, Max_eps: {max_eps}, Silhouette Score: {score}")

    if score > best_score:
        best_score = score
        best_params = (min_samples, max_eps)


best_model = OPTICS(min_samples=best_params[0], max_eps=best_params[1], metric='euclidea
df_test['target'] = best_model.fit_predict(df_test)
silhouette_test = silhouette_score(df_test, df_test['target'])

print(f"En iyi parametreler: {best_params}, En yüksek Silhouette Skoru: {best_score}")
print(f"Test Set - Silhouette Score: {silhouette_test}")
```

```
Min_samples: 50, Max_eps: 5, Silhouette Score: 0.4413528571165199
Min_samples: 50, Max_eps: 10, Silhouette Score: 0.8217728581082214
Min_samples: 50, Max_eps: 60, Silhouette Score: 0.8217728581082214
Min_samples: 100, Max_eps: 5, Silhouette Score: 0.3685144900506701
Min_samples: 100, Max_eps: 10, Silhouette Score: 0.7804688717482187
Min_samples: 100, Max_eps: 60, Silhouette Score: 0.910206647790529
Min_samples: 200, Max_eps: 5, Silhouette Score: 0.250345335639424
Min_samples: 200, Max_eps: 10, Silhouette Score: 0.7406495585261705
Min_samples: 200, Max_eps: 60, Silhouette Score: 0.90128271628075
Min_samples: 300, Max_eps: 5, Silhouette Score: 0.1833067002314214
Min_samples: 300, Max_eps: 10, Silhouette Score: 0.7377190382077174
Min_samples: 300, Max_eps: 60, Silhouette Score: 0.8968294304865038
Min_samples: 400, Max_eps: 5, Silhouette Score: 0.07106865365511149
Min_samples: 400, Max_eps: 10, Silhouette Score: 0.7337426262449301
Min_samples: 400, Max_eps: 60, Silhouette Score: 0.8968294304865038
En iyi parametreler: (100, 60), En yüksek Silhouette Skoru: 0.910206647790529
Test Set - Silhouette Score: 0.9129750159338755
```

In [35]:
```
optics_model = OPTICS(min_samples=300, max_eps=5, metric='euclidean')
df['target'] = optics_model.fit_predict(df)
df['target'] = df['target'].replace(0, 1).replace(-1, 0)
print(df['target'].value_counts())
```

```
target
1    1440
0    1094
Name: count, dtype: int64
```

In [36]:
```
# Silhouette Skoru
silhouette = silhouette_score(df, df['target'])

# Rand İndeksi ve Normalleştirilmiş Karşılıklı Bilgi (gerçek değerler gerekli)
adjusted_rand = adjusted_rand_score(df_v73['v73'], df['target'])
nmi = normalized_mutual_info_score(df_v73['v73'], df['target'])

# Purity hesaplama fonksiyonu
def purity_score(y_true, y_pred):
    matrix = contingency_matrix(y_true, y_pred)
    return np.sum(np.amax(matrix, axis=0)) / np.sum(matrix)

# Purity skoru
purity = purity_score(df_v73['v73'], df['target'])

print(f"Silhouette Score: {silhouette}")
print(f"Adjusted Rand Index: {adjusted_rand}")
print(f"Normalized Mutual Info Score: {nmi}")
print(f"Purity Score: {purity}")
```

```
Silhouette Score: 0.23612773030553386
Adjusted Rand Index: -0.012332359638663406
Normalized Mutual Info Score: 0.02806355485085088
Purity Score: 0.9368587213891081
```

Silhouette Skoru, kümelerin makul düzeyde tanımlanmış olduğunu ancak kümeler arasında net ayrımların eksik olduğunu gösterir. 1'e yakın bir değer ideal olurken, 0.24 değeri kümelerin birbirinden tamamen ayrılmadığını ve birbirine yakın olabileceğini işaret eder. Kümeler daha iyi tanımlanabilir ve aralarındaki mesafeler artırılabilir.

ARI'nin 1'e yakın olması idealdir ve mükemmel bir kümelemeyi gösterir. -0.01 değeri, modelin neredeyse rastgele bir kümeleme performansı sergilediğini gösterir. Bu değer, modelin kümeleme kalitesinin oldukça düşük olduğuna işaret eder.

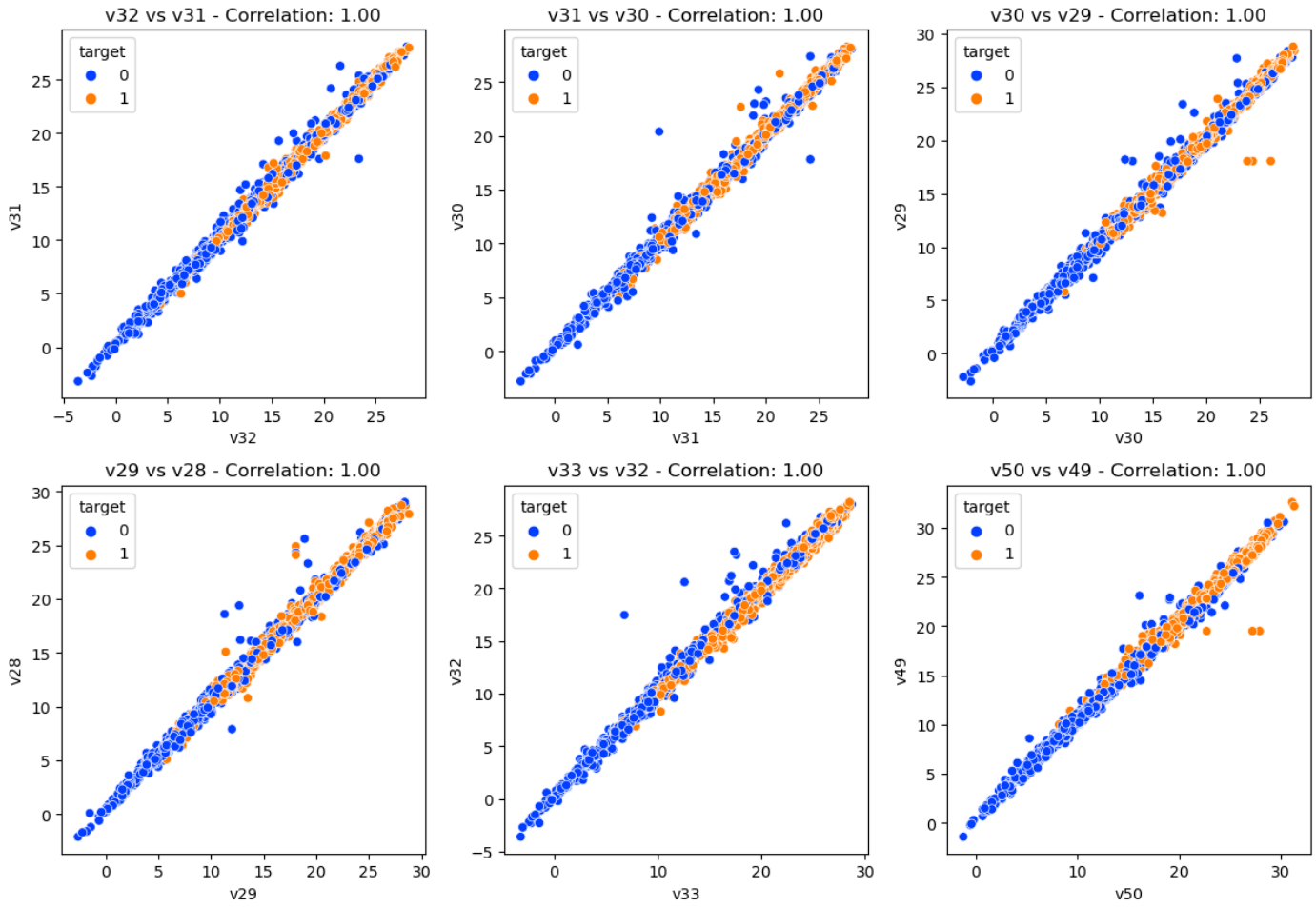
NMI skoru, modelin gerçek sınıflandırmalarla ne kadar bilgi paylaştığını gösterir. 1'e yakın bir değer mükemmel uyumu temsil ederken, 0.028 gibi düşük bir değer, modelin gerçek sınıflandırmalarla çok az ortak bilgi paylaştığını gösterir. Bu, modelin kümeleri tanımlama konusunda zayıf olduğunu işaret eder.

Purity skoru, her bir kümenin büyük oranda tek bir sınıfa ait veri noktalarından oluştuğunu gösterir. 0.9369 oldukça yüksek bir değerdir ve bu, her bir kümeyi oluşturan veri noktalarının büyük ölçüde homojen olduğunu gösterir. Ancak, diğer metrikler düşük olduğunda, bu durum kümelerin sayısının fazla olduğu veya çok spesifik kümelerin oluşturulduğu anlamına gelebilir. Bu, modelin belirli veri noktalarını aşırı derecede spesifik kümelerde topladığını ancak genel olarak kümeler arası farklılık ve doğru sınıflandırma açısından yetersiz kaldığını gösterir.

In [37]:
```
target_column = df['target']
df_without_target = df.drop('target', axis=1)
df_inverse_scaled = scaler.inverse_transform(df_without_target)
df_inverse_scaled = pd.DataFrame(df_inverse_scaled, columns=df_without_target.columns)
```

```
df_inverse_scaled['target'] = target_column
df = df_inverse_scaled
```

In [38]:
```
corr_matrix = df.corr().abs()
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

top_correlations = upper.unstack().sort_values(ascending=False).dropna()

top_6_correlations = top_correlations.head(6)

plt.figure(figsize=(12, 12))

for i, ((var1, var2), corr_value) in enumerate(top_6_correlations.items(), 1):
    plt.subplot(3, 3, i)
    sns.scatterplot(x=var1, y=var2, hue='target', data=df, palette='bright')
    plt.title(f'{var1} vs {var2} - Correlation: {corr_value:.2f}')

plt.tight_layout()
plt.show()
```



In [39]:
```
df['target']
```

Out[39]:
```
0       0
1       0
2       0
3       0
4       0
       ..
2529    0
2530    0
2531    1
2532    1
2533    1
Name: target, Length: 2534, dtype: int32
```

In [40]:
```
df_v73['v73']
```

Out[40]:
```
0       1
1       1
2       1
3       1
4       1
       ..
2529    1
2530    1
2531    1
2532    1
2533    1
Name: v73, Length: 2534, dtype: int64
```

In [41]:
```
y_true = df_v73['v73']
y_pred = df['target']

accuracy = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
conf_matrix = confusion_matrix(y_true, y_pred)

# Duyarlılık (Sensitivity) ve Özgüllük (Specificity)
sensitivity = conf_matrix[1, 1] / (conf_matrix[1, 1] + conf_matrix[1, 0])
specificity = conf_matrix[0, 0] / (conf_matrix[0, 0] + conf_matrix[0, 1])
auc = roc_auc_score(y_true, y_pred)

print("Accuracy:", accuracy)
print("F1 Score:", f1)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("AUC:", auc)
```

```
Accuracy: 0.5232833464877664
F1 Score: 0.6832721552176193
Sensitivity: 0.5488626790227464
Specificity: 0.14375
AUC: 0.34630633951137324
```

Accuracy (Doğruluk)

Doğruluk, modelin hem pozitif hem de negatif sınıfları ne kadar doğru tahmin ettiğinin bir ölçüsüdür. Yani toplam doğru tahminlerin (hem doğru pozitif hem de doğru negatif) toplam tahminlere oranıdır.

$$\text{Accuracy} = \frac{\text{Doğru Pozitif} + \text{Doğru Negatif}}{\text{Toplam Gözlem}}$$

F1 Score

F1 Skoru, modelin kesinlik (precision) ve duyarlılık (recall) arasındaki dengeli bir ölçüsüdür. Kesinlik, modelin pozitif olarak tahmin ettiği öğelerin gerçekten pozitif olma oranını, duyarlılık ise gerçek pozitiflerin doğru olarak tahmin edilme oranını ifade eder. F1 Skoru, bu iki ölçütün harmonik ortalamasıdır.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Sensitivity (Duyarlılık) / Recall (Geri Çağırma)

Duyarlılık, modelin gerçek pozitifleri ne kadar iyi tespit ettiğinin bir ölçüsüdür. Yani, gerçekten pozitif olan durumların ne kadarının doğru olarak pozitif olarak tahmin edildiğidir.

$$\text{Sensitivity} = \frac{\text{Doğru Pozitif}}{\text{Doğru Pozitif} + \text{Yanlış Negatif}}$$

Specificity (Özgüllük)

Özgüllük, modelin gerçek negatifleri ne kadar iyi tespit ettiğinin bir ölçüsüdür. Yani, gerçekten negatif olan durumların ne kadarının doğru olarak negatif olarak tahmin edildiğidir.

$$\text{Specificity} = \frac{\text{Doğru Negatif}}{\text{Doğru Negatif} + \text{Yanlış Pozitif}}$$

AUC (Area Under the Curve - Eğri Altı Alanı)

AUC, ROC (Receiver Operating Characteristic - Alıcı Çalışma Özellikleri) eğrisinin altında kalan alanın oranını ifade eder. ROC eğrisi, farklı eşik değerlerdeki modelin duyarlılık (sensitivity) ve 1-özgüllük (1-specificity) değerlerini grafik üzerinde gösterir. AUC, modelin sınıflandırma performansının genel bir ölçüsüdür. Değer 1'e ne kadar yakınsa, model o kadar iyi demektir. ROC eğrisi altında kalan alanın oranını ölçer.

In [42]:
```
# RobustScaler
scaler = RobustScaler()
df_scaled = df.drop('target', axis=1)
df_scaled = pd.DataFrame(scaler.fit_transform(df_scaled), columns=df_scaled.columns)
df_scaled['target'] = df['target']
df = df_scaled
print(df.head())
```

```
         v1        v2        v3        v4        v5        v6        v7  \
0 -0.571429  0.214286  0.714286  0.509091  0.461538  0.538462 -0.076923
1  0.857143  1.214286  1.357143  0.945455  1.461538  1.384615  1.000000
2  0.928571  0.928571  0.857143  0.509091  0.615385  0.846154  0.692308
3  2.214286  1.642857  1.642857  1.745455  1.153846  1.307692  0.923077
4  0.714286  0.428571  0.142857  0.000000 -0.384615  0.076923 -0.307692

         v8        v9       v10  ...       v64       v65       v66       v67  \
0 -0.267021 -0.456455 -0.391220  ... -0.277778  0.061506 -0.309601 -0.275862
1  0.579133  0.400688  0.394494  ...  0.638889 -0.114318  0.389881 -0.183908
2  0.502210 -0.242169 -0.248363  ...  0.972222 -0.226136  1.161902 -0.321839
3  0.348364 -0.099312  0.180209  ...  0.666667 -0.088099  1.257757 -0.459770
4 -0.497790 -0.885026 -1.034077  ...  0.151989  0.000000  0.000000 -0.056816

        v68       v69       v70       v71   v72  target
0 -0.831814 -1.923404  2.833333 -1.568001   0.0       0
1  0.034797 -0.978723  1.916667 -1.568001   0.0       0
2  0.162386  0.068085  1.250000 -1.139430   0.0       0
3  0.601491  0.953191  0.583333 -1.139430  41.6       0
4 -0.082484 -0.264823  0.069974  0.000000  11.6       0

[5 rows x 73 columns]
```

Yapay Sinir Ağı Modelleri

In [43]:
```
X = df.drop('target', axis=1)
y = df['target']

# Cross validation kullanılmadığında aşağıdaki ayrım kullanılabilir.
# X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, stratify=y, r
# X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.4, stratif

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, stratify=y, ran
```

```
print("Eğitim seti boyutu:", X_train.shape)
print("Test seti boyutu:", X_test.shape)
```

```
Eğitim seti boyutu: (1520, 72)
Test seti boyutu: (1014, 72)
```

In [44]:
```
print("Eğitim setindeki 'target' dağılımı:\n", y_train.value_counts(normalize=True))
# Cross-validation kullanılmazsa aşağıdaki val setinin dağılımını görmek faydalı olabili
# print("\nDoğrulama setindeki 'target' dağılımı:\n", y_val.value_counts(normalize=True)
print("\nTest setindeki 'target' dağılımı:\n", y_test.value_counts(normalize=True))
```

```
Eğitim setindeki 'target' dağılımı:
 target
1    0.568421
0    0.431579
Name: proportion, dtype: float64

Test setindeki 'target' dağılımı:
 target
1    0.568047
0    0.431953
Name: proportion, dtype: float64
```

In [45]:
```
# Eğitim modeli oluşturma
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='sigmoid', name='sigmoid_laye
model.add(Dense(1, activation='sigmoid'))  # Çıkış katmanı

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=10, verbose=0)

y_pred_train = model.predict(X_train)
y_pred_train = (y_pred_train > 0.5).astype(int).reshape(-1)

accuracy_train = accuracy_score(y_train, y_pred_train)
f1_train = f1_score(y_train, y_pred_train)
auc_train = roc_auc_score(y_train, y_pred_train)
cm_train = confusion_matrix(y_train, y_pred_train)
sensitivity_train = cm_train[1, 1] / (cm_train[1, 1] + cm_train[1, 0])
specificity_train = cm_train[0, 0] / (cm_train[0, 0] + cm_train[0, 1])

print(f"Eğitim Setindeki Accuracy: {accuracy_train:.2f}, F1: {f1_train:.2f}, AUC: {auc_t
```

```
48/48 [==============================] - 0s 830us/step
Eğitim Setindeki Accuracy: 0.94, F1: 0.94, AUC: 0.94, Sensitivity: 0.94, Specificity: 0.
93
```

In [46]:
```
neuronlar = [5, 10, 20]
aktivasyon_fonksiyonları = ['relu', 'tanh', 'sigmoid']
kf = KFold(n_splits=5)
performans_kaydı = {}
fold_metrics = []

for neuron in neuronlar:
    for Activation in aktivasyon_fonksiyonları:
        print(f"\nNöron Sayısı: {neuron}, Aktivasyon: {Activation}")

        fold_num = 1
        for train_index, val_index in kf.split(X_train):

            X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index
            y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index

            model = Sequential()
```

```
            model.add(Dense(neuron, input_dim=X_train_fold.shape[1], activation=Activati
            model.add(Dense(neuron, activation=Activation))
            model.add(Dense(1, activation='sigmoid'))

            model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
            model.fit(X_train_fold, y_train_fold, epochs=100, batch_size=10, verbose=0,

            y_pred_val_fold = model.predict(X_val_fold)
            y_pred_val_fold = (y_pred_val_fold > 0.5).astype(int).reshape(-1)

            accuracy = accuracy_score(y_val_fold, y_pred_val_fold)
            f1 = f1_score(y_val_fold, y_pred_val_fold)
            auc = roc_auc_score(y_val_fold, y_pred_val_fold)
            cm = confusion_matrix(y_val_fold, y_pred_val_fold)
            sensitivity = cm[1, 1] / (cm[1, 1] + cm[1, 0])
            specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])

            print(f"  Katlama {fold_num}: Accuracy: {accuracy:.2f}, F1: {f1:.2f}, AUC: {

            fold_num += 1
            fold_metrics.append([accuracy, f1, auc, sensitivity, specificity])
            performans_kaydı[(neuron, Activation)] = fold_metrics
```

```
Nöron Sayısı: 5, Aktivasyon: relu
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.90, F1: 0.91, AUC: 0.90, Sensitivity: 0.92, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.89, F1: 0.91, AUC: 0.89, Sensitivity: 0.91, Specificity: 0.87
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.90, F1: 0.92, AUC: 0.90, Sensitivity: 0.91, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.92, Specificity: 0.90
10/10 [==============================] - 0s 1000us/step
  Katlama 5: Accuracy: 0.90, F1: 0.91, AUC: 0.90, Sensitivity: 0.93, Specificity: 0.88

Nöron Sayısı: 5, Aktivasyon: tanh
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.90, F1: 0.91, AUC: 0.90, Sensitivity: 0.90, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.93, F1: 0.94, AUC: 0.92, Sensitivity: 0.94, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.89, F1: 0.91, AUC: 0.90, Sensitivity: 0.89, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.91, F1: 0.92, AUC: 0.90, Sensitivity: 0.94, Specificity: 0.87

Nöron Sayısı: 5, Aktivasyon: sigmoid
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.93, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.91, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.94, Specificity: 0.88
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.91, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.96, Specificity: 0.88

Nöron Sayısı: 10, Aktivasyon: relu
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.95, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.93, Sensitivity: 0.92, Specificity: 0.93
10/10 [==============================] - 0s 1000us/step
  Katlama 3: Accuracy: 0.91, F1: 0.93, AUC: 0.91, Sensitivity: 0.95, Specificity: 0.87
```

```
10/10 [==============================] - 0s 1000us/step
  Katlama 4: Accuracy: 0.90, F1: 0.92, AUC: 0.90, Sensitivity: 0.91, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.94, Specificity: 0.91

Nöron Sayısı: 10, Aktivasyon: tanh
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.91, F1: 0.92, AUC: 0.90, Sensitivity: 0.93, Specificity: 0.87
10/10 [==============================] - 0s 1000us/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.92, Specificity: 0.92
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.89, F1: 0.91, AUC: 0.89, Sensitivity: 0.93, Specificity: 0.84
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.91, F1: 0.93, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88
10/10 [==============================] - 0s 1000us/step
  Katlama 5: Accuracy: 0.88, F1: 0.89, AUC: 0.87, Sensitivity: 0.92, Specificity: 0.83

Nöron Sayısı: 10, Aktivasyon: sigmoid
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.93, Sensitivity: 0.92, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.89, F1: 0.91, AUC: 0.88, Sensitivity: 0.93, Specificity: 0.83
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.92, Specificity: 0.93
10/10 [==============================] - 0s 12ms/step
  Katlama 5: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.95, Specificity: 0.88

Nöron Sayısı: 20, Aktivasyon: relu
10/10 [==============================] - 0s 2ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.94, Specificity: 0.90
10/10 [==============================] - 0s 3ms/step
  Katlama 2: Accuracy: 0.93, F1: 0.93, AUC: 0.93, Sensitivity: 0.90, Specificity: 0.96
10/10 [==============================] - 0s 1000us/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.90, Sensitivity: 0.95, Specificity: 0.85
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.91, F1: 0.92, AUC: 0.90, Sensitivity: 0.93, Specificity: 0.88
10/10 [==============================] - 0s 889us/step
  Katlama 5: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88

Nöron Sayısı: 20, Aktivasyon: tanh
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.95, Specificity: 0.87
10/10 [==============================] - 0s 1000us/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.90, Specificity: 0.94
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.88, F1: 0.90, AUC: 0.87, Sensitivity: 0.93, Specificity: 0.82
10/10 [==============================] - 0s 1000us/step
  Katlama 4: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.94, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88

Nöron Sayısı: 20, Aktivasyon: sigmoid
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.94, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.90, Specificity: 0.92
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.90, F1: 0.92, AUC: 0.90, Sensitivity: 0.93, Specificity: 0.88
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.93, Specificity: 0.91
10/10 [==============================] - 0s 1000us/step
  Katlama 5: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88
```

In [47]:
```
# En iyi modeli bulma ve tüm performans metriklerini gösterme
en_iyi_ortalama = 0
en_iyi_kombinasyon = None
en_iyi_metrikler = None

for kombinasyon, metrikler in performans_kaydı.items():
    ortalama_metrikler = np.mean(metrikler, axis=0)
    ortalama_performans = np.mean(ortalama_metrikler)
    if ortalama_performans > en_iyi_ortalama:
        en_iyi_ortalama = ortalama_performans
        en_iyi_kombinasyon = kombinasyon
        en_iyi_metrikler = ortalama_metrikler

en_iyi_metrikler = np.mean(performans_kaydı[en_iyi_kombinasyon], axis=0)
accuracy_val, f1_val, auc_val, sensitivity_val, specificity_val = en_iyi_metrikler

print(f"En iyi kombinasyon: Nöron Sayısı - {en_iyi_kombinasyon[0]}, Aktivasyon Fonksiyon
print(f"Ortalama Performans: {en_iyi_ortalama:.2f}")

accuracy_val, f1_val, auc_val, sensitivity_val, specificity_val = en_iyi_metrikler
print(f"Doğrulama Seti Metrikleri (En İyi Kombinasyon):")
print(f"Accuracy: {accuracy_val:.2f}, F1: {f1_val:.2f}, AUC: {auc_val:.2f}, Sensitivity:
```

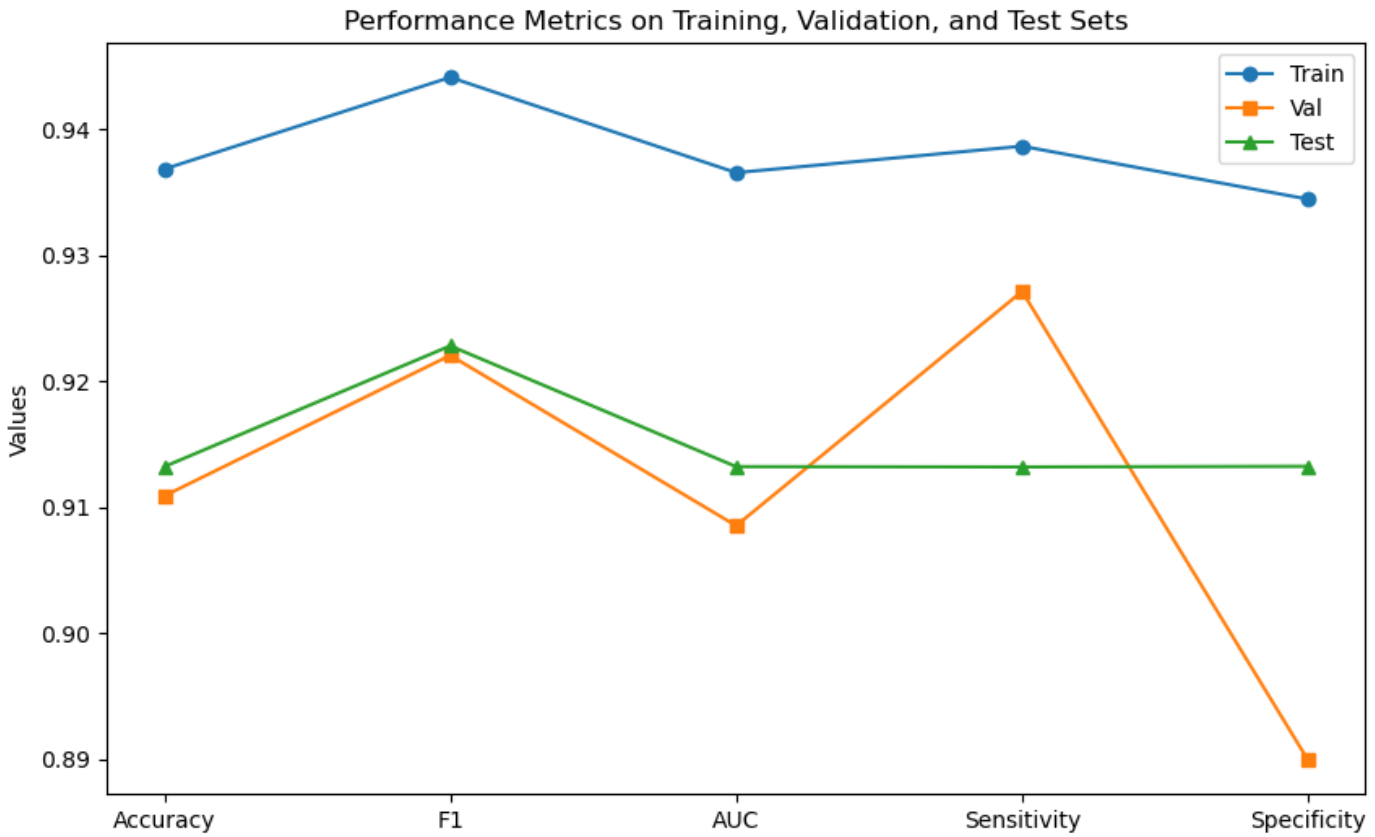
```
En iyi kombinasyon: Nöron Sayısı - 5, Aktivasyon Fonksiyonu - relu
Ortalama Performans: 0.91
Doğrulama Seti Metrikleri (En İyi Kombinasyon):
Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.89
```

In [48]:
```
# En iyi modeli oluşturma
en_iyi_nöron = en_iyi_kombinasyon[0]
en_iyi_aktivasyon = en_iyi_kombinasyon[1]

model = Sequential()
model.add(Dense(en_iyi_nöron, input_dim=X_train.shape[1], activation=en_iyi_aktivasyon))
model.add(Dense(en_iyi_nöron, activation=en_iyi_aktivasyon))  # İkinci gizli katman
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train, y_train, epochs=100, batch_size=10, verbose=0)

y_pred_test = model.predict(X_test)
y_pred_test = (y_pred_test > 0.5).astype(int).reshape(-1)

accuracy_test = accuracy_score(y_test, y_pred_test)
f1_test = f1_score(y_test, y_pred_test)
auc_test = roc_auc_score(y_test, y_pred_test)
cm_test = confusion_matrix(y_test, y_pred_test)
sensitivity_test = cm_test[1, 1] / (cm_test[1, 1] + cm_test[1, 0])
specificity_test = cm_test[0, 0] / (cm_test[0, 0] + cm_test[0, 1])

print(f"Test Setindeki Accuracy: {accuracy_test:.2f}, F1: {f1_test:.2f}, AUC: {auc_test:
```

```
32/32 [==============================] - 0s 871us/step
Test Setindeki Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.91, Specificity: 0.91
```

In [49]:
```
metrikler = ['Accuracy', 'F1', 'AUC', 'Sensitivity', 'Specificity']
eğitim_değerleri = [accuracy_train, f1_train, auc_train, sensitivity_train, specificity_
doğrulama_değerleri = [accuracy_val, f1_val, auc_val, sensitivity_val, specificity_val]
test_değerleri = [accuracy_test, f1_test, auc_test, sensitivity_test, specificity_test]

x = range(len(metrikler))
plt.figure(figsize=(10, 6))
plt.plot(x, eğitim_değerleri, label='Train', marker='o')
plt.plot(x, doğrulama_değerleri, label='Val', marker='s')
plt.plot(x, test_değerleri, label='Test', marker='^')
```

```
plt.xticks(x, metrikler)
plt.ylabel('Values')
plt.title('Performance Metrics on Training, Validation, and Test Sets')
plt.legend()
plt.show()
```



CNN

In [50]:
```
# CNN modelini oluşturma
cnn_model = Sequential()

cnn_model.add(Conv1D(filters=10, kernel_size=2, activation='sigmoid', input_shape=(X_tra
cnn_model.add(Flatten())


cnn_model.add(Dense(10, activation='sigmoid'))
cnn_model.add(Dense(1, activation='sigmoid'))


cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])


X_train_reshaped = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))


cnn_model.fit(X_train_reshaped, y_train, epochs=100, batch_size=10, verbose=0)

y_pred_train_cnn = cnn_model.predict(X_train_reshaped)
y_pred_train_cnn = (y_pred_train_cnn > 0.5).astype(int).reshape(-1)

accuracy_train_cnn = accuracy_score(y_train, y_pred_train_cnn)
f1_train_cnn = f1_score(y_train, y_pred_train_cnn)
auc_train_cnn = roc_auc_score(y_train, y_pred_train_cnn)
cm_train_cnn = confusion_matrix(y_train, y_pred_train_cnn)
sensitivity_train_cnn = cm_train_cnn[1, 1] / (cm_train_cnn[1, 1] + cm_train_cnn[1, 0])
specificity_train_cnn = cm_train_cnn[0, 0] / (cm_train_cnn[0, 0] + cm_train_cnn[0, 1])
```

```
print(f"CNN Eğitim Setindeki Accuracy: {accuracy_train_cnn:.2f}, F1: {f1_train_cnn:.2f},
```

```
48/48 [==============================] - 0s 1ms/step
CNN Eğitim Setindeki Accuracy: 0.96, F1: 0.97, AUC: 0.96, Sensitivity: 0.99, Specificit
y: 0.93
```

In [51]:
```
neuronlar = [5, 10, 20]
aktivasyon_fonksiyonları = ['relu', 'sigmoid', 'tanh']
kf = KFold(n_splits=5)
performans_kaydı_cnn = {}
fold_metrics = []
for neuron in neuronlar:
    for Activation in aktivasyon_fonksiyonları:
        print(f"\nNöron Sayısı: {neuron}, Aktivasyon: {Activation}")

        fold_num = 1
        fold_metrics_cnn = []

        for train_index, val_index in kf.split(X_train):
            X_train_fold, X_val_fold = X_train.iloc[train_index], X_train.iloc[val_index
            y_train_fold, y_val_fold = y_train.iloc[train_index], y_train.iloc[val_index

            X_train_fold_reshaped = X_train_fold.values.reshape((X_train_fold.shape[0],
            X_val_fold_reshaped = X_val_fold.values.reshape((X_val_fold.shape[0], X_val_

            cnn_model = Sequential()
            cnn_model.add(Conv1D(filters=neuron, kernel_size=2, activation=Activation, i
            cnn_model.add(Flatten())
            cnn_model.add(Dense(neuron, activation=Activation))
            cnn_model.add(Dense(1, activation='sigmoid'))

            cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['ac
            cnn_model.fit(X_train_fold_reshaped, y_train_fold, epochs=100, batch_size=10

            y_pred_val_fold = cnn_model.predict(X_val_fold_reshaped)
            y_pred_val_fold = (y_pred_val_fold > 0.5).astype(int).reshape(-1)

            accuracy = accuracy_score(y_val_fold, y_pred_val_fold)
            f1 = f1_score(y_val_fold, y_pred_val_fold)
            auc = roc_auc_score(y_val_fold, y_pred_val_fold)
            cm = confusion_matrix(y_val_fold, y_pred_val_fold)
            sensitivity = cm[1, 1] / (cm[1, 1] + cm[1, 0])
            specificity = cm[0, 0] / (cm[0, 0] + cm[0, 1])
            fold_metrics_cnn.append([accuracy, f1, auc, sensitivity, specificity])

            print(f"  Katlama {fold_num}: Accuracy: {accuracy:.2f}, F1: {f1:.2f}, AUC: {

            fold_num += 1

        fold_metrics.append([accuracy, f1, auc, sensitivity, specificity])
        performans_kaydı_cnn[(neuron, Activation)] = fold_metrics_cnn
```

```
Nöron Sayısı: 5, Aktivasyon: relu
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.96, Specificity: 0.88
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.94, F1: 0.95, AUC: 0.94, Sensitivity: 0.94, Specificity: 0.95
10/10 [==============================] - 0s 1000us/step
  Katlama 3: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.95, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.94, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.97, Specificity: 0.88

Nöron Sayısı: 5, Aktivasyon: sigmoid
```

```
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.95, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.95, F1: 0.96, AUC: 0.95, Sensitivity: 0.97, Specificity: 0.94
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.96, Specificity: 0.90
10/10 [==============================] - 0s 1000us/step
  Katlama 4: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.93, Specificity: 0.93
10/10 [==============================] - 0s 2ms/step
  Katlama 5: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.96, Specificity: 0.91

Nöron Sayısı: 5, Aktivasyon: tanh
10/10 [==============================] - 0s 3ms/step
  Katlama 1: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.95, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.94, F1: 0.94, AUC: 0.94, Sensitivity: 0.92, Specificity: 0.96
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.90, Sensitivity: 0.95, Specificity: 0.86
10/10 [==============================] - 0s 889us/step
  Katlama 4: Accuracy: 0.93, F1: 0.94, AUC: 0.92, Sensitivity: 0.95, Specificity: 0.89
10/10 [==============================] - 0s 1000us/step
  Katlama 5: Accuracy: 0.89, F1: 0.91, AUC: 0.88, Sensitivity: 0.96, Specificity: 0.80

Nöron Sayısı: 10, Aktivasyon: relu
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.95, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.93, F1: 0.93, AUC: 0.93, Sensitivity: 0.90, Specificity: 0.96
10/10 [==============================] - 0s 1000us/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.88
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.93, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.95, F1: 0.95, AUC: 0.95, Sensitivity: 0.97, Specificity: 0.92

Nöron Sayısı: 10, Aktivasyon: sigmoid
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.94, F1: 0.95, AUC: 0.94, Sensitivity: 0.96, Specificity: 0.91
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.93, Sensitivity: 0.89, Specificity: 0.96
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.94, Specificity: 0.92
10/10 [==============================] - 0s 1000us/step
  Katlama 4: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.91, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.94, F1: 0.95, AUC: 0.94, Sensitivity: 0.94, Specificity: 0.94

Nöron Sayısı: 10, Aktivasyon: tanh
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.91, Sensitivity: 0.94, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.90, Specificity: 0.94
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.94, Specificity: 0.87
10/10 [==============================] - 0s 1000us/step
  Katlama 4: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.93, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.98, Specificity: 0.86

Nöron Sayısı: 20, Aktivasyon: relu
10/10 [==============================] - 0s 2ms/step
  Katlama 1: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.95, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.93, F1: 0.94, AUC: 0.94, Sensitivity: 0.92, Specificity: 0.95
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.95, Specificity: 0.88
```

```
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.94, F1: 0.95, AUC: 0.94, Sensitivity: 0.94, Specificity: 0.93
10/10 [==============================] - 0s 3ms/step
  Katlama 5: Accuracy: 0.94, F1: 0.94, AUC: 0.94, Sensitivity: 0.96, Specificity: 0.91

Nöron Sayısı: 20, Aktivasyon: sigmoid
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.94, F1: 0.94, AUC: 0.93, Sensitivity: 0.96, Specificity: 0.90
10/10 [==============================] - 0s 2ms/step
  Katlama 2: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.90, Specificity: 0.94
10/10 [==============================] - 0s 3ms/step
  Katlama 3: Accuracy: 0.92, F1: 0.93, AUC: 0.91, Sensitivity: 0.94, Specificity: 0.89
10/10 [==============================] - 0s 2ms/step
  Katlama 4: Accuracy: 0.93, F1: 0.94, AUC: 0.93, Sensitivity: 0.92, Specificity: 0.93
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.92, F1: 0.93, AUC: 0.93, Sensitivity: 0.92, Specificity: 0.93

Nöron Sayısı: 20, Aktivasyon: tanh
10/10 [==============================] - 0s 1ms/step
  Katlama 1: Accuracy: 0.92, F1: 0.93, AUC: 0.92, Sensitivity: 0.93, Specificity: 0.90
10/10 [==============================] - 0s 1ms/step
  Katlama 2: Accuracy: 0.90, F1: 0.91, AUC: 0.90, Sensitivity: 0.89, Specificity: 0.92
10/10 [==============================] - 0s 1ms/step
  Katlama 3: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.94, Specificity: 0.87
10/10 [==============================] - 0s 1ms/step
  Katlama 4: Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.92, Specificity: 0.89
10/10 [==============================] - 0s 1ms/step
  Katlama 5: Accuracy: 0.90, F1: 0.91, AUC: 0.90, Sensitivity: 0.93, Specificity: 0.86
```

In [52]:
```python
en_iyi_ortalama_cnn = 0
en_iyi_kombinasyon_cnn = None
en_iyi_metrikler_cnn = None

for kombinasyon, metrikler in performans_kaydı.items():
    ortalama_metrikler_cnn = np.mean(metrikler, axis=0)
    ortalama_performans_cnn = np.mean(ortalama_metrikler_cnn)  # Bu, tüm metriklerin ort
    if ortalama_performans_cnn > en_iyi_ortalama_cnn:
        en_iyi_ortalama_cnn = ortalama_performans_cnn
        en_iyi_kombinasyon_cnn = kombinasyon
        en_iyi_metrikler_cnn = ortalama_metrikler_cnn

en_iyi_metrikler_cnn = np.mean(performans_kaydı[en_iyi_kombinasyon_cnn], axis=0)
accuracy_val_cnn, f1_val_cnn, auc_val_cnn, sensitivity_val_cnn, specificity_val_cnn = en

print(f"En iyi CNN kombinasyonu: Nöron Sayısı - {en_iyi_kombinasyon_cnn[0]}, Aktivasyon
print(f"Ortalama Performans: {en_iyi_ortalama_cnn:.2f}")
print(f"Doğrulama Seti Metrikleri (En İyi CNN Kombinasyon):")
print(f"Accuracy: {accuracy_val_cnn:.2f}, F1: {f1_val_cnn:.2f}, AUC: {auc_val_cnn:.2f},
```

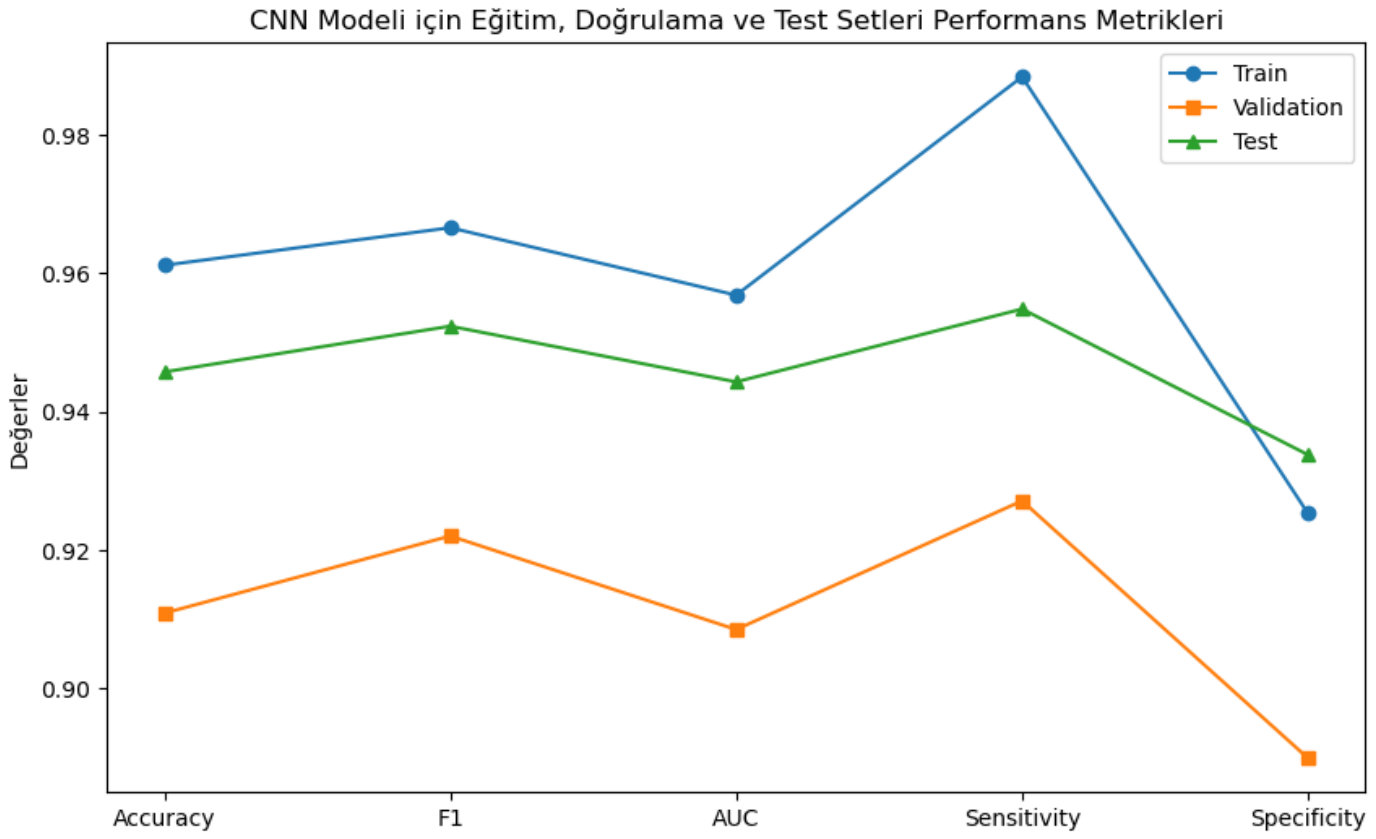
```
En iyi CNN kombinasyonu: Nöron Sayısı - 5, Aktivasyon Fonksiyonu - relu
Ortalama Performans: 0.91
Doğrulama Seti Metrikleri (En İyi CNN Kombinasyon):
Accuracy: 0.91, F1: 0.92, AUC: 0.91, Sensitivity: 0.93, Specificity: 0.89
```

In [53]:
```python
en_iyi_nöron = en_iyi_kombinasyon[0]
en_iyi_aktivasyon = en_iyi_kombinasyon[1]
cnn_model = Sequential()
cnn_model.add(Conv1D(filters=en_iyi_nöron, kernel_size=3, activation=en_iyi_aktivasyon,
cnn_model.add(Flatten())
cnn_model.add(Dense(en_iyi_nöron, activation=en_iyi_aktivasyon))
cnn_model.add(Dense(1, activation='sigmoid'))
cnn_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

X_train_reshaped = X_train.values.reshape((X_train.shape[0], X_train.shape[1], 1))
X_test_reshaped = X_test.values.reshape((X_test.shape[0], X_test.shape[1], 1))
cnn_model.fit(X_train_reshaped, y_train, epochs=100, batch_size=10, verbose=0)
```

```
y_pred_test_cnn = cnn_model.predict(X_test_reshaped)
y_pred_test_cnn = (y_pred_test_cnn > 0.5).astype(int).reshape(-1)

accuracy_test_cnn = accuracy_score(y_test, y_pred_test_cnn)
f1_test_cnn = f1_score(y_test, y_pred_test_cnn)
auc_test_cnn = roc_auc_score(y_test, y_pred_test_cnn)
cm_test_cnn = confusion_matrix(y_test, y_pred_test_cnn)
sensitivity_test_cnn = cm_test_cnn[1, 1] / (cm_test_cnn[1, 1] + cm_test_cnn[1, 0])
specificity_test_cnn = cm_test_cnn[0, 0] / (cm_test_cnn[0, 0] + cm_test_cnn[0, 1])

print(f"CNN Test Setindeki Accuracy: {accuracy_test_cnn:.2f}, F1: {f1_test_cnn:.2f}, AUC
```

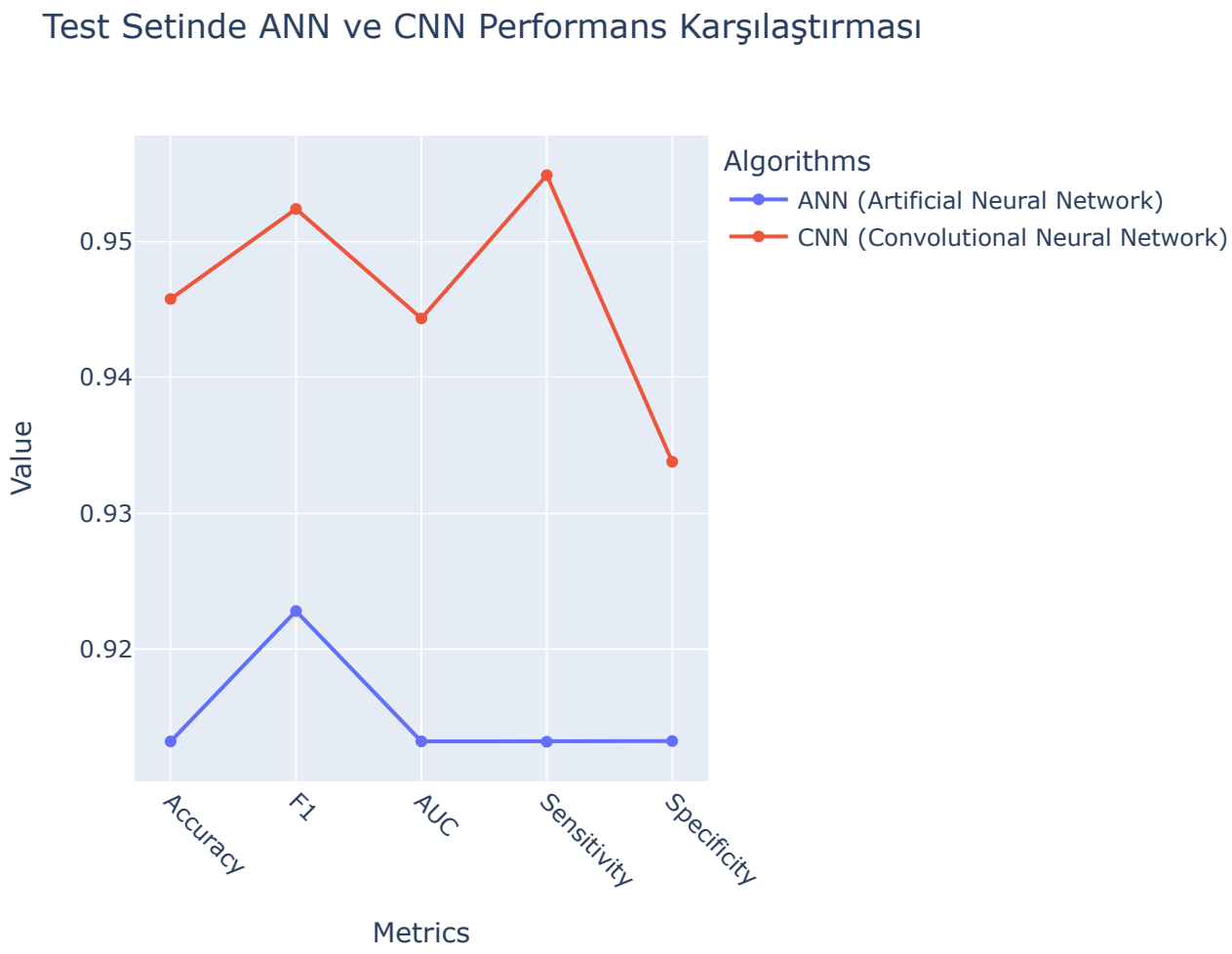
```
32/32 [==============================] - 0s 1ms/step
CNN Test Setindeki Accuracy: 0.95, F1: 0.95, AUC: 0.94, Sensitivity: 0.95, Specificity:
0.93
```

In [54]:
```
metrikler = ['Accuracy', 'F1', 'AUC', 'Sensitivity', 'Specificity']
eğitim_değerleri_cnn = [accuracy_train_cnn, f1_train_cnn, auc_train_cnn, sensitivity_tra
doğrulama_değerleri_cnn = [accuracy_val_cnn, f1_val_cnn, auc_val_cnn, sensitivity_val_cn
test_değerleri_cnn = [accuracy_test_cnn, f1_test_cnn, auc_test_cnn, sensitivity_test_cnn

x = range(len(metrikler))
plt.figure(figsize=(10, 6))
plt.plot(x, eğitim_değerleri_cnn, label='Train', marker='o')
plt.plot(x, doğrulama_değerleri_cnn, label='Validation', marker='s')
plt.plot(x, test_değerleri_cnn, label='Test', marker='^')
plt.xticks(x, metrikler)
plt.ylabel('Değerler')
plt.title('CNN Modeli için Eğitim, Doğrulama ve Test Setleri Performans Metrikleri')
plt.legend()
plt.show()
```



In [55]:
```
Metrics = ['Accuracy', 'F1', 'AUC', 'Sensitivity', 'Specificity']
ann_değerler = [accuracy_test, f1_test, auc_test, sensitivity_test, specificity_test]
cnn_değerler = [accuracy_test_cnn, f1_test_cnn, auc_test_cnn, sensitivity_test_cnn, spec
df_end = pd.DataFrame({'Metrics': Metrics, 'ANN (Artificial Neural Network)': ann_değerl
df_melted = df_end.melt(id_vars='Metrics', var_name='Algorithms', value_name='Value')
```

```
fig = px.line(df_melted, x='Metrics', y='Value', color='Algorithms', markers=True, title
fig.update_xaxes(tickangle=45)
fig.show()
```



Referanslar:

1. Artificial Neural Network: Understanding the Basic Concepts without Mathematics
2. A scalable and fast OPTICS for clustering trajectory big data
3. Artificial neural networks: a practical review of applications involving fractional calculus
4. OPTICS: Ordering Points To Identify the Clustering Structure.
5. BLOCK-OPTICS: An Efficient Density-Based Clustering Based on OPTICS
6. An improved OPTICS clustering algorithm for discovering clusters with uneven densities
7. Clustering algorithms: A comparative approach
8. Learning representations by back-propagating errors
9. Clustering and Neural Networks
10. Combination of artificial neural network and clustering techniques for predicting phytoplankton biomass of Lake Poyang, China
11. Comparison of hierarchical clustering and neural network clustering: an analysis on precision dominance