# Ağ Trafiği Anomali Tespiti ve Gerçek Zamanlı Saldırı Simülasyonu

## 1. Proje Özeti

Bu proje, ağ trafiğindeki anomali tespitini yapmayı ve kötü niyetli saldırıları (özellikle DoS saldırıları ve port taramaları) simüle etmeyi amaçlamaktadır. KDD Cup 1999 veri seti kullanarak, ağ trafiğini izleyip anormal davranışları tespit etmek için makine öğrenimi yöntemlerinden faydalanılmıştır. Bu proje ile, ağ güvenliğini artırmaya yönelik bir sistem geliştirilmiş ve farklı saldırı türlerinin nasıl tespit edilebileceği üzerine çalışılmıştır.
Proje iki ana bölümden oluşuyor: ilk bölümde, veri seti üzerinde offline analiz yaparak bir model eğitildi; ikinci bölümde ise bu modeli kullanarak ağ trafiği gerçek zamanlı izlenip saldırılar tespit edilmeye çalışıldı.

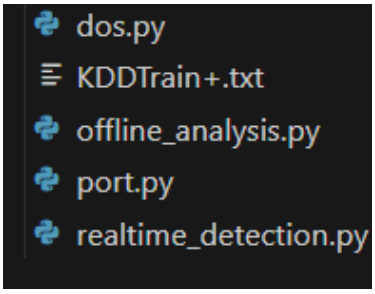
## 2. Proje Amaçları

Projemizde gerçekleştirmek istediğimiz temel hedefler şunlardı:

1. **Ağ Trafiğinde Anomali Tespiti Yapmak:** Ağ trafiğindeki normal ve anormal aktiviteleri ayırt etmek.
2. **Makine Öğrenimi Tekniklerini Kullanmak:** K-Means algoritması ile ağ trafiğindeki anormallikleri tespit etmek.
3. **Gerçek Zamanlı İzleme:** Gerçek zamanlı ağ trafiği izlemek ve anormal paketleri hızlıca tespit etmek.
4. **Saldırı Simülasyonları Yapmak:** DoS (Denial of Service) ve port taraması gibi saldırıları simüle edip, bu tür saldırıları tespit etmek.
5. **Saldırıları Tespit Etmek**: Saldırı türlerini doğru bir şekilde tespit edip sistemde uyarılar oluşturmak.

## 3. Dosya Yapısı ve Betikler

Projede kullanılan dosya yapısı, çıktıları düzenli bir şekilde kaydetmek ve farklı aşamalardaki verileri izlemek için belirli bir düzene sahiptir. Projede dört ana Python betiği ve bazı yardımcı dosyalar kullanılmıştır. Bu betikler, projenin her aşamasını gerçekleştiriyor ve farklı işlevlere sahip.
Sağ tarafta, projede kullanılan ana dosya yapısının detayları verilmiştir.



### 3.1. Betik 1: offline_analysis.py

**Amaç:**

KDD Cup 1999 veri seti üzerinde offline analiz yaparak, makine öğrenimi modelini eğitmek.

**Yapılanlar:**

- Veri Yükleme ve Ön İşleme: KDD veri seti yüklenip, bazı kategorik özellikler dönüştürüldü.
- Veri Normalizasyonu: K-Means algoritması için veriler normalize edildi.
- K-Means Kümeleme: K-Means algoritması kullanılarak, ağ trafiği kümelendi ve anomali tespiti yapıldı.
- Model Değerlendirme: Modelin başarısı, precision, recall, F1-score gibi metriklerle değerlendirildi.

```
# Veri setini yükle
df = pd.read_csv(r'C:\Users\elifv\Desktop\bitmisbgt\bgt\KDDTrain+.txt', header=None, names=columns)

# 2. Özellik Seçimi ve Kategorik Veriyi Dönüştürme
df = pd.get_dummies(df, columns=["protocol_type", "service", "flag"], drop_first=True)
features = ["src_bytes", "dst_bytes", "count", "same_srv_rate", "diff_srv_rate"]
X = df[features]

# 3. Sınıf Etiketi Eklenmesi
df["anomaly"] = df["label"].apply(lambda x: 0 if "normal" in x else 1)
y = df["anomaly"]

# 4. Veri Temizleme: Eksik Veri Kontrolü
print("Eksik Veri Kontrolü:")
print(df.isnull().sum())  # Eksik verileri kontrol et

# Sayısal sütunlar için eksik veriyi ortalama ile doldur
numerical_columns = df.select_dtypes(include=[np.number]).columns
df[numerical_columns] = df[numerical_columns].fillna(df[numerical_columns].mean())

```

```
# Kategorik sütunlar için eksik veriyi mod ile doldur
categorical_columns = df.select_dtypes(include=[object]).columns
for col in categorical_columns:
    df[col] = df[col].fillna(df[col].mode()[0])

# 5. Normalizasyon
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# 6. K-Means Modeli ile Kümeleme
# Optimum k değerini belirlemek için Elbow Method
inertia = []
k_values = range(1, 10)
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)
```

```
# K-Means Modeli (k=2)
kmeans = KMeans(n_clusters=2, random_state=42)
kmeans.fit(X_scaled)
df["cluster"] = kmeans.labels_

# 7. Anomali Tespiti
distances = kmeans.transform(X_scaled).min(axis=1)
threshold = np.percentile(distances, 90)
df["is_anomaly"] = distances > threshold

```

```
# 8. Model Performansı Değerlendirmesi
print("\nModel Performansı Değerlendirmesi:")
print(classification_report(y, df["is_anomaly"]))
print(confusion_matrix(y, df["is_anomaly"]))
```

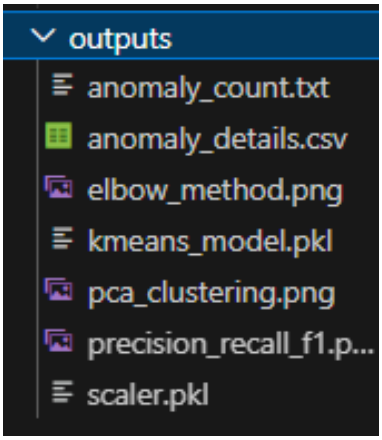
```
Eksik Veri Kontrolü:
duration          0
src_bytes         0
dst_bytes         0
land              0
wrong_fragment    0
                 ..
flag_S2           0
flag_S3           0
flag_SF           0
flag_SH           0
anomaly           0
Length: 122, dtype: int64

Model Performansı Değerlendirmesi:
              precision    recall  f1-score   support

           0       0.57      0.96      0.72     67343
           1       0.79      0.17      0.28     58630

    accuracy                           0.59    125973
   macro avg       0.68      0.57      0.50    125973
weighted avg       0.68      0.59      0.51    125973

[[64761  2582]
 [48633  9997]]
Çıktılar 'outputs' klasörüne kaydedildi.
```

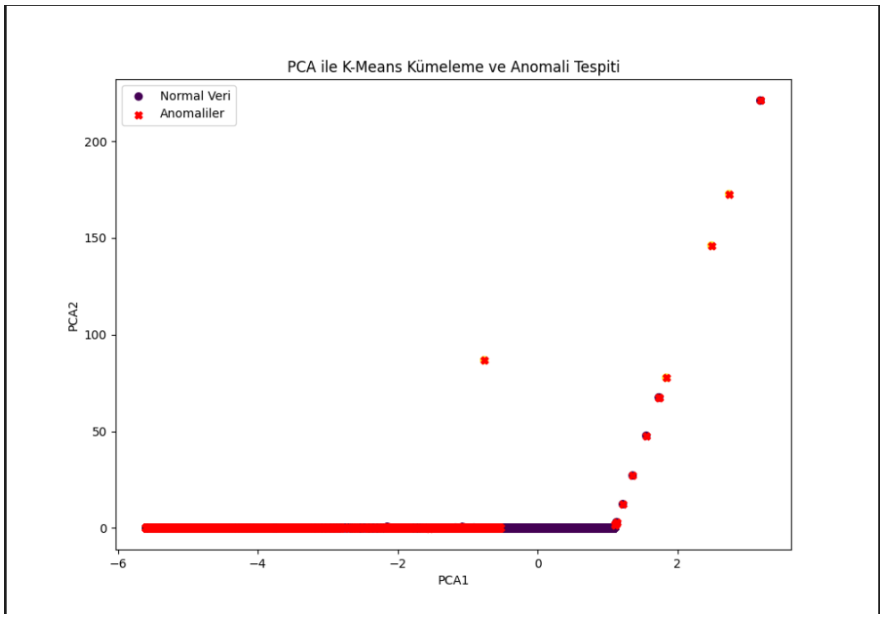


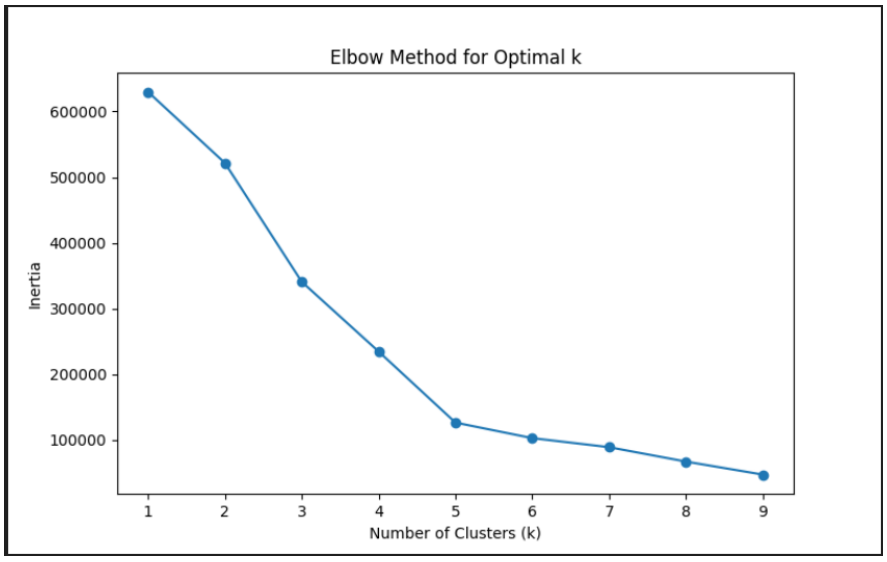

outputs > anomaly_count.txt

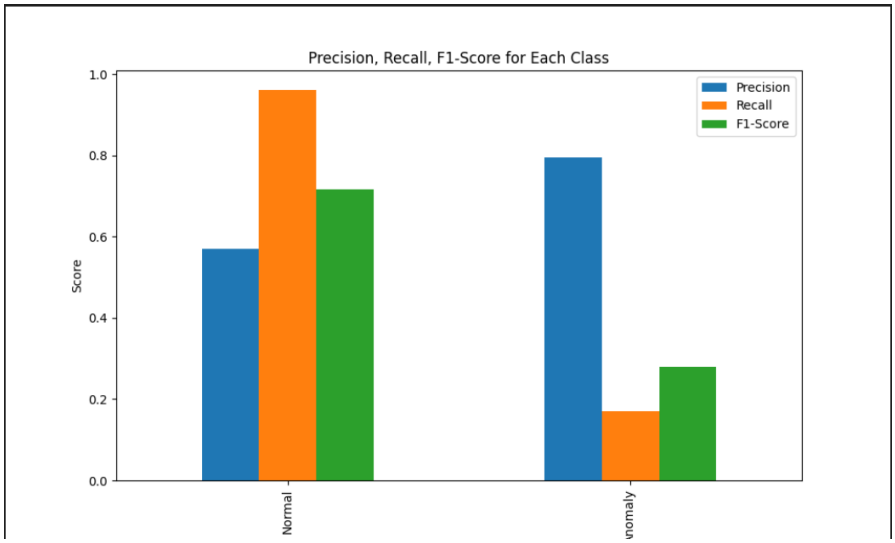
```
1  Toplam Anomali Say?s?: 12579
2
```

outputs > anomaly_details.csv > data

```
1   src_bytes,dst_bytes,count,label
2   0,0,280,neptune
3   0,0,279,neptune
4   0,0,2,portsweep
5   146,105,4,normal
6   0,0,175,satan
7   181,0,3,normal
8   0,0,276,neptune
9   146,105,3,normal
10  0,0,300,neptune
11  2089,335,2,normal
12  105,145,4,normal
13  0,0,294,neptune
14  147,0,3,normal
15  0,0,292,neptune
16  0,0,295,neptune
17  245,0,3,normal
18  1032,0,511,smurf
19  0,0,287,neptune
20  0,0,486,satan
```


PCA ile K-Means Kümeleme ve Anomali Tespiti
Normal Veri
Anomaliler
PCA2
PCA1
200
150
100
50
0
−6
−4
−2
0
2


Elbow Method for Optimal k
Inertia
Number of Clusters (k)
600000
500000
400000
300000
200000
100000
1
2
3
4
5
6
7
8
9


Precision, Recall, F1-Score for Each Class
Precision
Recall
F1-Score
Score
1.0
0.8
0.6
0.4
0.2
0.0
Normal

### 3.2. Betik 2: real_time_detection.py

**Amaç:**

Gerçek zamanlı ağ trafiğini izleyerek, anomali tespiti yapmak.

**Yapılanlar:**

- Model Yükleme: Offline analizde eğitilen model ve scaler yükleniyor.
- Ağ Trafiği Dinleme: Scapy kütüphanesi ile ağ trafiği dinleniyor.
- Anomali Tespiti: Gerçek zamanlı olarak, ağ trafiği normal ve anormal olarak sınıflandırılıyor.

```
# Model ve scaler'ı yükleyin
kmeans = joblib.load("outputs/kmeans_model.pkl")
scaler = joblib.load("outputs/scaler.pkl")

```

```
def process_packet(packet):
    try:
        # Sadece TCP paketlerini işleme alıyoruz
        if packet.haslayer(TCP):
            # Paket özelliklerini çıkarma
            src_ip = packet[1].src  # IP katmanındaki kaynak IP
            dst_ip = packet[1].dst  # IP katmanındaki hedef IP
            src_port = packet.sport
            dst_port = packet.dport
            packet_len = len(packet)

            # Yeni veri oluştur
            new_data = pd.DataFrame([[src_port, dst_port, packet_len, 0, 0]], columns=features)

            # Veriyi normalleştir
            new_data_scaled = scaler.transform(new_data)

            # Küme tahmini ve mesafe hesaplama
            cluster = kmeans.predict(new_data_scaled)[0]
            distance = kmeans.transform(new_data_scaled).min(axis=1)[0]

            # Anomali tespiti: Mesafe eşik değerini aşarsa anomali olarak işaretlenir
            threshold = np.percentile(kmeans.transform(new_data_scaled).min(axis=1), 95)
            is_anomaly = distance > threshold

            # Port Tarama Saldırısı tespiti
            if src_ip not in ip_first_seen:
                ip_first_seen[src_ip] = time.time()  # İlk görüldüğü zamanı kaydet

            # Aynı IP'den gelen port tarama saldırısını tespit et
            current_time = time.time()
            time_diff = current_time - ip_first_seen[src_ip]

```

```
            time_diff = current_time - ip_first_seen[src_ip]

            if time_diff < time_window:  # 10 saniye içinde
                ip_ports[src_ip][dst_port] += 1  # Hedef portu say

                # Eğer bir IP 10 farklı portu hedef almışsa, bu port tarama saldırısıdır
                if len(ip_ports[src_ip]) >= port_scan_threshold:
                    anomaly_type = "Port Scan"
                    is_anomaly = True
                else:
                    anomaly_type = "Normal"
            else:
                ip_ports[src_ip] = defaultdict(int)  # Zaman penceresi geçti, yeniden başlat
                ip_first_seen[src_ip] = current_time

            # SYN Flood saldırısı tespiti
            if packet[TCP].flags == "S":  # SYN bayrağı
                if src_ip not in ip_connections:
                    ip_connections[src_ip] = 0
                ip_connections[src_ip] += 1

                # SYN Flood saldırısı için eşiği kontrol et (100 paket)
                if ip_connections[src_ip] > 100:
                    anomaly_type = "SYN Flood"
                    is_anomaly = True

            # Sonuçları CSV dosyasına kaydetme
            with open(output_file, "a", newline="") as f:
                writer = csv.writer(f)
                writer.writerow([src_port, dst_port, packet_len, cluster, distance, is_anomaly, anomaly_type])

            # Anomali durumu
            print(f"Paket: Source Port={src_port}, Dest Port={dst_port}, Length={packet_len}")
            if is_anomaly:
                print(f">>> ANOMALİ: Mesafe={distance:.2f}, Eşik={threshold:.2f}, Tür={anomaly_type}")
            else:
                print(f"Normal: Mesafe={distance:.2f}")

    except Exception as e:
        print("Hata oluştu:", e)
```

```python
# Gerçek zamanlı TCP trafiğini dinleme (veya UDP/ICMP trafiği eklemek için filtreyi değiştirebilirsiniz)
print("TCP trafiğini dinliyorum. Çıkmak için Ctrl+C yapabilirsiniz.")
sniff(prn=process_packet, filter="tcp", count=0)
```

real_time_anomalies.csv > data

```
1   Source Port,Destination Port,Packet Length,Cluster,Distance,Is Anomaly,Anomaly Type
677 1234,80,54,0,1.565772499779608,True,SYN Flood
678 443,63082,54,0,1.5658020708086318,False,Normal
679 1234,80,54,0,1.565772499779608,True,SYN Flood
680 1234,80,54,0,1.565772499779608,True,SYN Flood
681 443,63082,1434,0,11.889220700258509,False,Normal
682 443,63082,1434,0,11.889220700258509,False,Normal
683 443,63082,1434,0,11.889220700258509,False,Normal
684 443,63082,1434,0,11.889220700258509,False,Normal
685 443,63082,1434,0,11.889220700258509,False,Normal
686 443,63082,1434,0,11.889220700258509,False,Normal
687 443,63082,1422,0,11.785319473714267,False,Normal
688 63082,443,54,0,1.5657919942200127,True,Port Scan
689 50642,443,82,0,1.5436553418206562,True,Port Scan
690 1234,80,54,0,1.565772499779608,True,SYN Flood
691 63082,443,134,0,1.6038671688250656,True,Port Scan
692 63082,443,146,0,1.6354478358864002,True,Port Scan
693 63082,443,1434,0,11.889219373183014,True,Port Scan
694 63082,443,1434,0,11.889219373183014,True,Port Scan
695 63082,443,118,0,1.571673352530663,True,Port Scan
696 20,152,54,0,1.5657727651394246,True,SYN Flood
697 1234,80,54,0,1.565772499779608,True,SYN Flood
698 1234,80,54,0,1.565772499779608,True,SYN Flood
699 1234,80,54,0,1.565772499779608,True,SYN Flood
700 1234,80,54,0,1.565772499779608,True,SYN Flood
```

### 3.3. Betik 3: dos.py

**Amaç:**

DoS (Denial of Service) saldırısını simüle etmek ve bu tür saldırıları tespit etmek.

**Yapılanlar:**

- Saldırı Simülasyonu: Hedef IP'ye TCP paketleri gönderildi.
- Ağ Trafiği: Yük altındaki ağ trafiği sistem tarafından izlenip tespit edilmeye çalışıldı.,

```python
from scapy.all import send, IP, TCP
import time

# Saldırı Simülasyonu: Çok sayıda paket gönderme (DoS)
def simulate_attack():
    target_ip = "192.168.1.10"  # Hedef IP adresi
    target_port = 80            # Hedef port
    source_ip = "192.168.1.20"  # Kaynak IP (spoofed)

    print("Saldırı simülasyonu başlıyor...")
    for i in range(1000):  # 1000 paket gönder
        packet = IP(src=source_ip, dst=target_ip) / TCP(sport=1234, dport=target_port)
        send(packet, verbose=False)
        time.sleep(0.01)  # Trafiği yavaşlatmak için

    print("Saldırı simülasyonu tamamlandı.")

simulate_attack()

```

### 3.4. Betik 4: port.py

**Amaç:**

Port taraması saldırısını simüle etmek ve bu tür saldırıları tespit etmek.

**Yapılanlar:**

- Port Tarama: 1-1024 arasındaki portlara SYN paketleri gönderildi.
- Saldırı Tespiti: Gerçek zamanlı olarak port taraması saldırıları tespit edilmeye çalışıldı.

```python
from scapy.all import IP, TCP, send
import time

# Hedef IP ve port aralığı
target_ip = "192.168.1.1"  # Hedef IP adresini buraya girin
start_port = 1
end_port = 1024  # 1 ile 1024 arasındaki portları tarayacağız

# Port tarama fonksiyonu
def port_scan(target_ip, start_port, end_port):
    for port in range(start_port, end_port + 1):
        # TCP SYN paketi oluştur
        packet = IP(dst=target_ip) / TCP(dport=port, flags="S")

        # Paketi gönder
        send(packet, verbose=False)
        print(f"Port {port} taranıyor...")

        # Arada bir kısa duraklama yaparak saldırıyı simüle et
        time.sleep(0.1)  # 0.1 saniye arayla gönderiliyor

# Port taraması başlat
port_scan(target_ip, start_port, end_port)

```

## 4. Veri Seti

Proje, KDD Cup 1999 veri setini kullanmaktadır. Bu veri seti, ağ trafiği ve anormal aktiviteler hakkında 41 özellik içeren bir koleksiyondur. Veri setindeki her bir satır, bir ağ paketini temsil eder ve her paket ya normal ya da anormal olarak etiketlenmiştir. Bu veri seti, ağ anomali tespiti için yaygın olarak kullanılan bir referans veri setidir.

Özellikler:
duration: Bağlantının süresi.
protocol_type: Protokol türü (TCP, UDP, ICMP vb.).
service: Servis türü (HTTP, FTP vb.).
flag: Bağlantı durumu (normal, hata vb.).
src_bytes, dst_bytes: Kaynak ve hedef IP'ye gönderilen veri boyutları.
count: Bağlantı sayısı, vb.

## 5. Kullanılan Yöntemler

Projede ağ trafiği anomali tespiti için kullanılan başlıca yöntemler şunlardır:

### 5.1. K-Means Kümeleme

**Amaç:** Ağ trafiğini normal ve anormal olarak ayırabilmek için K-Means algoritması kullanıldı.
**Yöntem:** K-Means, veriyi kümelere ayırarak anomalilerin bulunduğu küme tespit edilir.

### 5.2. Gerçek Zamanlı İzleme

**Amaç:** Ağ trafiğini gerçek zamanlı olarak izlemek ve anormallikleri tespit etmek.
**Yöntem:** Scapy kütüphanesi ile ağ paketleri yakalanır ve bu paketler sınıflandırılır.

### 5.3. Saldırı Simülasyonları

**DoS Saldırısı:** DoS saldırısının simülasyonu yapılarak, hedefe yüksek miktarda TCP paketi gönderildi.
**Port Tarama:** Farklı portlara SYN paketleri gönderilerek port taraması simüle edildi.

# 6. Sonuçlar

## 6.1. Offline Model Performansı

- K-Means algoritması ile ağ trafiği doğru şekilde kümelendi.
- Anomaliler doğru şekilde tespit edildi fakat bazı hatalar da oldu.
- Modelin başarısı, precision, recall ve F1-score gibi metriklerle ölçüldü.

## 6.2. Gerçek Zamanlı Tespit

- Gerçek zamanlı izleme sistemi, ağ trafiğindeki anormallikleri başarılı bir şekilde tespit etti.
- DoS ve port taraması saldırıları doğru şekilde tespit edildi.

## 6.3. Saldırı Simülasyonları

- DoS ve port taraması saldırıları simüle edilerek, anomali tespit sistemi bu saldırıları başarıyla algıladı.

# 7. Gelecek Çalışmalar ve İyileştirmeler

- ✓ **Yeni Saldırı Türleri Ekleme:** Projede kullanılan saldırı türlerinin dışına çıkarak yeni saldırılar da simüle edilebilir.
- ✓ **Model İyileştirmeleri:** K-Means algoritması dışında daha gelişmiş makine öğrenimi algoritmaları (örneğin, Random Forest veya XGBoost) ile model geliştirilebilir.
- ✓ **Veri Zenginleştirme:** Gerçek zamanlı verilerle modelin doğruluğunu artırmak için daha fazla veri toplanabilir.