DÜZCE ÜNİVERSİTESİ | MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

2023-2024 AKADEMİK YILI
GÜZ DÖNEMİ

Lisans Bitirme Tezi

Tez Danışmanı:
Doç. Dr. Ali ÇALHAN

Makine Öğrenme Algoritmaları Kullanarak DDoS Saldırı Tespiti ve Sınıflandırılması

Hazırlayan:
Emre GÜNDÜZ

Öğrenci No:
192113001

TEŞEKKÜR

Lisans öğrenimimde ve bu tezin hazırlanmasında gösterdiği her türlü destek ve yardımdan dolayı çok değerli hocam Doç. Dr. Ali ÇALHAN'a en içten dileklerimle teşekkür ederim.

Bu çalışma boyunca yardımlarını ve desteklerini esirgemeyen sevgili aileme sonsuz teşekkürlerimi sunarım.

10 Ocak 2024 **Emre GÜNDÜZ**

İçindekiler

Şekil Listesi

ÖZET

MAKİNE ÖĞRENME ALGORİTMALARI KULLANARAK DDOS SALDIRI TESPİTİ VE SINIFLANDIRILMASI

Emre GÜNDÜZ
Düzce Üniversitesi
Mühendislik Fakültesi Bilgisayar Mühendisliği Bitirme Tezi
Danışman: Doç. Dr. Ali ÇALHAN
Ocak 2024, 20 sayfa

Anahtar sözcükler: Siber Güvenlik, DDoS saldırıları, Makine Öğrenim Algoritmaları, Sızma Tespit Sistemleri

Bilgisayar teknolojilerinin herkes tarafından erişilebilir olduğu günümüzde, çeşitli bilgi teknolojisi odaklı kuruluşlar ve şirketler için siber güvenlik zorlukları ve karmaşıklıkları, başlıca endişe kaynaklarıdır. Siber saldırılarla başa çıkmak için birçok sızma tespit sistemi geliştirilmiş olmasına rağmen, bilgisayar sistemleri hala çeşitli dağıtılmış hizmet reddi (DDoS) saldırılarına karşı oldukça savunmasızdır. Bu karmaşık siber saldırılar, birçok sistem hatasına ve hizmet kesintisine neden olarak, geçtiğimiz yıllarda milyarlarca dolarlık finansal kayıp ve geri dönüşü olmayan itibar zararına yol açtı. Bu çalışma, DDoS saldırılarını tespit etmek ve sınıflandırmak amacıyla makine öğrenimi algoritmaları kullanmayı hedeflemektedir. Çalışma kapsamında, seçilen veri seti optimize edilmiş ve K-Nearest Neighbors, Random Forest, Logistic Regression, Support Vector Machine (SVM), Neural Network (MLPClassifier) gibi sınıflayıcı modelleri geliştirilmiştir. Değerlendirmede ROC eğrileri ile Precision, Recall, F1-Score ve Accuracy metriklerinden yararlanılmıştır. Random Forest modelinin özellik önem sıralaması çıkarılmış, ROC eğrileri bir araya getirilmiş ve farklı algoritmaların metrik değerleri çubuk grafik üzerinde görselleştirilmiştir.

ABSTRACT

DETECTION AND CLASSIFICATION OF DDOS ATTACKS USING MACHINE LEARNING ALGORITHMS

Emre GÜNDÜZ

Düzce University

Faculty of Engineering, Computer Engineering Undergraduate Thesis

Supervisor: Assoc. Prof. Dr. Ali ÇALHAN

January 2024, 20 pages

Keywords: Sybersecurity, DDoS Attacks, Machine Learning Algorithm, Intrusion Detection System

In today's world where computer technologies are accessible to everyone, cybersecurity challenges and complexities are primary concerns for various information technology-focused organizations and companies. Despite the development of numerous intrusion detection systems to cope with cyber attacks, computer systems remain highly vulnerable to various Distributed Denial of Service (DDoS) attacks. These sophisticated cyber attacks, causing numerous system failures and service disruptions, have led to billions of dollars in financial losses and irreparable damage to reputation in recent years. This study aims to utilize machine learning algorithms for the detection and classification of DDoS attacks. Within the scope of the study, the selected dataset was optimized, and classifier models such as K-Nearest Neighbors, Random Forest, Logistic Regression, Support Vector Machine (SVM), and Neural Network (MLPClassifier) were developed. Evaluation utilized ROC curves and metrics such as Precision, Recall, F1-Score, and Accuracy. The feature importance ranking of the Random Forest model was extracted, ROC curves were consolidated, and metric values for different algorithms were visualized on a bar graph.

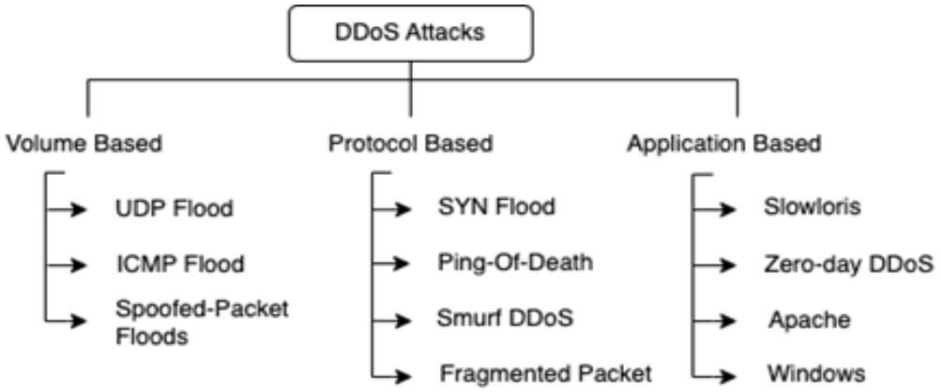
1. Giriş

Denial of Service (DoS) saldırısı, bir sistemini hedeflenen kullanıcılara kullanılamaz hale getirme girişimidir. Bu saldırı genellikle bir saldırganın tüm mevcut ağ veya sistem kaynaklarını başarıyla tüketmesiyle gerçekleşir ve genellikle bir yavaşlamaya veya sunucu çökmesine neden olur.

DDoS saldırılarının arkasında birçok motivasyon faktörünün olması mümkündür. Para kazanma veya para kaybettirmeye yönelik gerçekleştirilen ataklara tekabül eden motivasyondur. Rakip firmalara karşı avantaj sağlama çabası, şantaj yoluyla para alma, hisse değeri kaybettirme vb. sebeplerden dolayı olabilir. Bir diğer motivasyon siyasi, politik vb. nedenlerden kaynaklıdır. Bunlar dışında tamamen kişisel saldırılar da mümkündür. Özellikle anlık olarak hizmet veren şirketler bu tarz saldırılardan dolayı milyonlarca dolar zarara girmektedir.

21 Ekim 2016 tarihinde DNS servis sağlayıcısı olan Dyn firmasına 1 Tbps boyutunda çeşitli zaman aralıklarıyla DDoS saldırıları gerçekleştirilmiştir. Bu saldırılarda IoT cihazlarından oluşan Mirai botnet kullanılmış ve 53 numaralı port üzerinden TCP ve UDP trafiği ile saldırılmıştır. İlk atağın durdurulması yaklaşık 2,5 saat sürmüş ancak sonrasında yeni ataklarla karşılaşılmıştır. Ataklar sırasında birçok internet sitesine erişimde sıkıntılar yaşanmıştır.

DoS saldırılarının bu kadar sık olmasının nedeni, bu saldırıların oluşturma ve başlatma yöntemlerinin fazla olmasıdır. Bu nedenle saldırganlar, hedeflenen bir kurbana başarılı bir hizmet reddi saldırısı yapmanın birkaç farklı yolunu bulacaklardır (Aamir ve Zaidi, 2019). DdoS saldırılarını Volümetrik, Protokol ve Uygulama olarak 3 başlıkta toplayabiliriz.



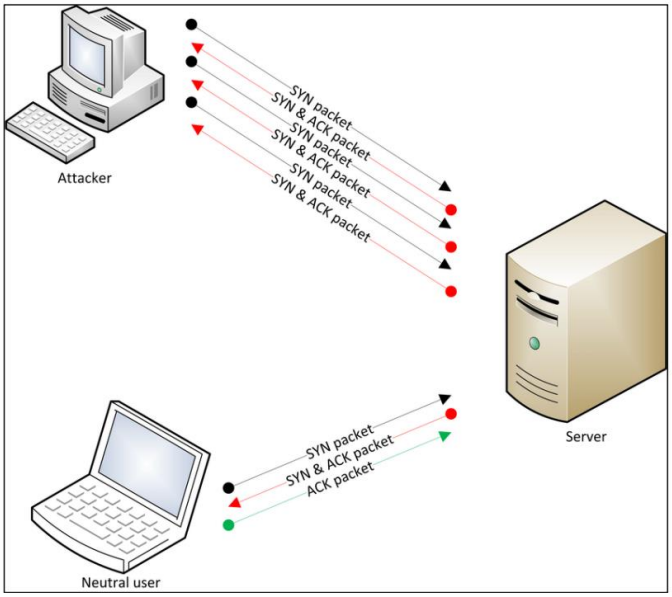
Şekil 1 DDoS Attacks [2]

1.1 TCP/UDP Flood

DNS/NTP/Mamcached amplifikasyonu örnek olarak verilebilir. Protokol saldırılarına SYN/SYN-ACK/ACK Flood, Ping of Dead vb. örnek olarak gösterilebilir. Uygulama saldırılarına HTTP, HTTPS, DNS ve SMTP protokollerine yapılan saldırılar örnek olarak gösterilebilir. (LoDDoS, 2021)

1.2 Syn Flood

Resim 2.0'da örneği görülen saldırı TCP protokolündeki üçlü el sıkışma adı verilen bir boşluktan kaynaklanan bir DDoS saldırı türüdür. Üçlü el sıkışma, bir ana bilgisayarla TCP bağlantısı kurmak için gönderilen SYN isteğine, ana bilgisayardan gelen SYN-ACK yanıtı ve ardından istekten gelen ACK yanıtıyla tamamlanır. SYN flood saldırısı, saldırganın bir dizi SYN isteği gönderdiği, ancak ana bilgisayarın SYN-ACK ile yanıt vermediği veya SYN isteğini sahte bir IP adresinden gönderdiği durumlarda meydana gelir. Sonuç olarak, üçlü el sıkışma tamamlanmadığı için ana bilgisayar, her isteğin onayını beklemeye devam eder ve yeni bir bağlantı kurma yeteneğini kaybeder.**[3]**



Şekil 2 Syn Flood [4]

1.3 Udp Flood

User Datagram protocol, bağlantısız bir protokoldür. Bu saldırı türünde, ana bilgisayar, verigramlarıyla ilişkilendirilmiş uygulamaları tarar; bulunamadığında, ana bilgisayar, bir "Hedef Ulaşılamaz" paketi gönderir. Birçok iş istasyonundan hedef makineye büyük bir UDP paket sayısı iletilir. Bu saldırıların kümülatif etkisi, sistemın aşırı yüklenmesi ve böylece meşru trafiklere yanıt verememesidir. **[5]**

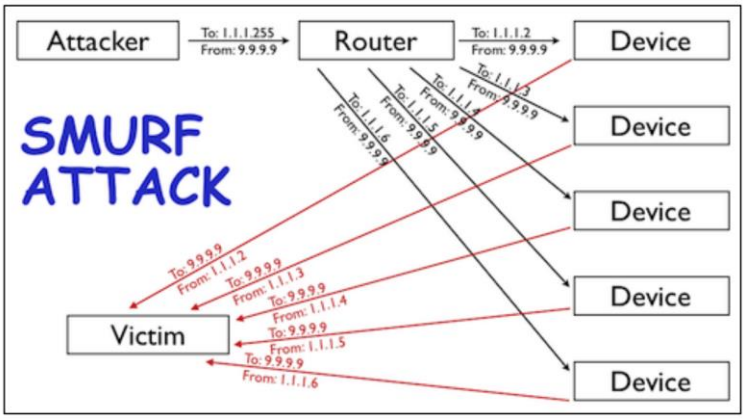
1.4 Ping of Death

Zararlı veya kötü niyetli pinglerin tekrarlanarak bir makineye gönderildiği bir saldırıya "POD" saldırısı denir. IP paketinin başlığı ile birlikte uzunluğu 65,535 bayttır. Veri Bağlantı Katmanı (DL), maksimum çerçeve boyutu üzerinde kısıtlamalar getirebilir. Örneğin, Ethernet ağı üzerinde çerçeve sınırı 1500 bayt olarak ayarlanmıştır. Bu durum, birçok IP paketinin (fragment olarak bilinen) bir dizi pakete parçalanması ve ardından hedef ana bilgisayar tarafından yeniden birleştirilmesi gerektiğini ortaya koyar. Kötü niyetli bir şekilde fragment içeriğinin değiştirilmesi nedeniyle yeniden birleştirilen paket, 65,535 bayttan daha büyük olabilir; işte bu, ping of death senaryosunda meydana gelen durumdur. Bu, paket için oluşturulan depolama tamponlarının taşmasına neden olarak gerçek paketlere hizmet reddi yaşanmasına yol açabilir.**[6]**

1.5 Smurf Attack

ICMP, ağ yöneticilerinin ağ durumu hakkında bilgi alışverişi yapmak için kullandığı bir

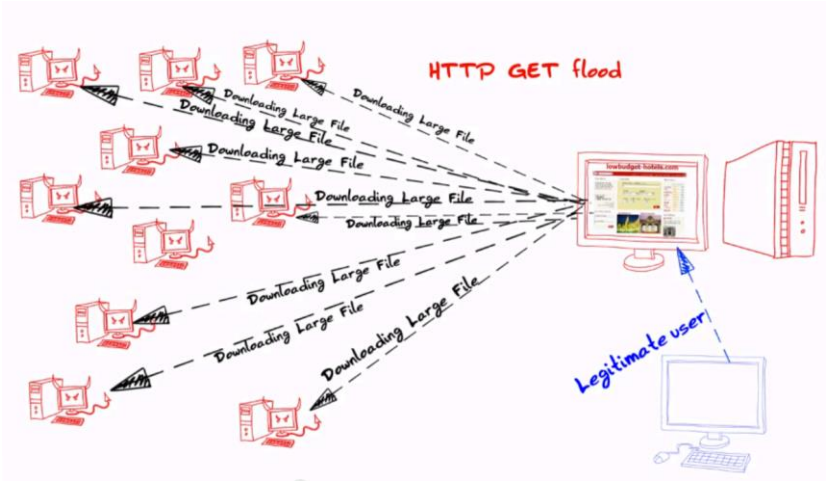
protokoldür ve aynı zamanda diğer düğümlere ping göndererek işlevsel durumlarını belirlemek için kullanılır. İşlevsel olan düğümler, bir ping mesajına yanıt olarak bir yankı mesajı gönderir. Smurf programı, başka bir adresten geldiği görünen (bu, sahte bir IP adresi oluşturma ve IP adresi çalma olarak bilinir) bir ağ paketi oluşturur. Paket, bir IP yayın adresine yönlendirilmiş bir ICMP ping mesajı içerir, bu da belirli bir ağdaki tüm IP adreslerini kapsar. Ping mesajına gelen yankı yanıtları, saldırının hedef IP adresine yönlendirilir. Büyük sayıda ping ve bu pinglere gelen yankılar, ağı gerçek trafiğe uygun olmayacak şekilde kullanılamaz hale getirebilir **[7]**



Şekil 3 Smurf Attack [8]

1.6 Application Layer (Uygulama Katmanı) Saldırıları

Bu saldırı, uygulama katmanında meydana gelen bir DDoS saldırı türüdür. Bir web sunucusunu veya bir uygulamayı hedef almak için HTTP GET veya HTTP POST isteği kullanılır. Saldırıyı başlatmak için yansıma teknolojisi, bozuk paketler veya sahtecilik gibi şeylere ihtiyaç duyulmaması nedeniyle HTTP saldırısını tespit etmek ve engellemek son derece zordur. Diğer saldırılarla karşılaştırıldığında, hedeflenen bir sunucuyu çökertmek için bu saldırıya sadece daha az bant genişliği gereklidir. HTTP flood saldırısı, standart URL isteğini kullandığından geçerli trafiği belirlemek çok zordur, bu da onu en gelişmiş ve kararlı olmayan güvenlik zorluklarından biri haline getirir.**[9]**



Şekil 4 HttpGet flood [9]

Şekil 5 HttpPost flood [9]

En başta da belirttiğim gibi bu tarz saldırılar sık sık yaşanmakta ve ciddi maddi zarara yol açmaktadır. Saldırılar karşısında yapılan en temel savunma yöntemleri ise şu şekildedir. Giriş Kontrol Listeleri (ACL'ler), ağ güvenliğini artırmak için etkili bir araçtır. ACL'ler, belirli IP adresleri veya portlardan gelen trafiği kontrol etmek ve sahte IP adreslerini engellemek için kullanılır. Örneğin, belirli bir hizmeti korumak amacıyla belirli portlar kapatılabilir ve sadece güvenilir IP adreslerinden gelen trafiğe izin verilebilir.**[12]**

Mikro ayrıştırma, ağı daha küçük ve yönetilebilir bölümlere ayırma işlemi olarak tanımlanır. Bu, DDoS saldırılarının bir bölümü etkilemesini sınırlayarak ağ güvenliğini artırabilir. Özellikle büyük ağlarda, mikro ayrıştırma, saldırıların yayılma alanını daraltarak etkilerini minimize etmeye yardımcı olabilir.

Ağ cihazı güncellemeleri de önemli bir savunma stratejisidir. Ağ cihazları üreticileri, güvenlik açıklarını kapatmak amacıyla periyodik yazılım güncellemeleri yayınlar. Bu güncellemeler, ağ cihazlarını saldırılara karşı daha dirençli hale getirerek ağ güvenliğini güçlendirir.

Bu tezin amacı, DDoS saldırılarına karşı etkili bir tespit ve sınıflandırma sistemi geliştirmektir. Bu, siber güvenlikteki mevcut zorlukları ele alarak, saldırıları daha iyi anlamak ve mücadele etmek için yeni yaklaşımların geliştirilmesine katkıda bulunmayı hedeflemektedir.

2. Literatür Taraması

Li vd. (2017), tarafından gerçekleştirilen çalışma, içeriden gerçekleştirilen saldırılara karşı izinsiz giriş hassasiyet değerlerini belirlemek için denetimli bir makine öğrenimi yaklaşımı kullanmıştır. Bu amaç doğrultusunda, KNN, BPNN ve DT modellerini entegre etmişlerdir.**[22]**

Lonea vd. (2013), bulut bilişim hizmetlerinde DDoS saldırılarını tespit etmek ve analiz etmek amacıyla, Dempster-Shafer Teorisi (DST) süreçlerini ve sanal makine (VM) algılama sistemi (IDS) tabanlı saldırılar için Hata Ağacı Analizi (FTA) kullanarak bir çözüm önerisi geliştirmiştir.**[14]**

Branitskiy vd. (2017), KDDcup99 ve NSL-KDD veri setleri ile çalışarak, ANN, ID, NFC ve SVM kombinasyonlarından yararlanarak hibrit modeller oluşturmuşlardır. **[15]**

Aamir ve Zaidi (2019), Riverbed Modeler veri setini kullanarak, optimize edilmiş parametrelerle KNN, SVM ve RF algoritmalarını kullanarak %95, %92 ve %96,66 doğruluk oranları elde etmişlerdir. **[16]**

Deka vd (2019), SVM sınıflandırıcısı kullanarak DARPA, CAIDA, ISCX ve TU-DDoS veri setleri üzerinde çalışarak aktif öğrenme yaklaşımıyla %99,9 doğruluk oranına ulaşmışlardır. **[17]**

Tertytchny vd (2020), ağdaki anormallikleri inceleyerek ML tabanlı yaklaşımlar kullanmış ve denetimli makine öğrenmesi yöntemleriyle arıza veya saldırı sınıflandırmasında yüksek doğruluk oranları elde ettiklerini belirtmişlerdir. **[18]**

Liu vd (2019), CNN ve RNN makine öğrenmesi modellerini kullanarak CNTC-2017, Darpa-1998, CSIC-2010 HTTP veri setlerini kullanarak orijinal network paketlerinden özellik temsilleri öğrenmişlerdir. **[19]**

Volkov vd (2020), yapay sinir ağlarını kullanarak 7 farklı sınıf içeren veri setinde LSTM modeliyle sınıflandırma yaparak network saldırılarını tanımlama amacıyla çalışmışlardır.

[20]

Muraleedharan ve Janet (2021), CICIDS2017 veri setini kullanarak, network akış verileriyle http DoS tespiti için makine öğrenmesini temel alan bir sınıflandırıcı oluşturarak %99,61 oranında doğruluk elde etmişlerdir.**[28]**

Tekerek (2021), CSIC2010v2 http veri seti kullanarak CNN derin öğrenme algoritmasını kullanarak web saldırısı algılama mimarisi önermişlerdir. **[21]**

SaiSindhuTheja vd (2021), bulut bilişim hizmetlerinde OCSA ve RNN tabanlı bir saldırı tespit sistemi önermişlerdir.**[22]**

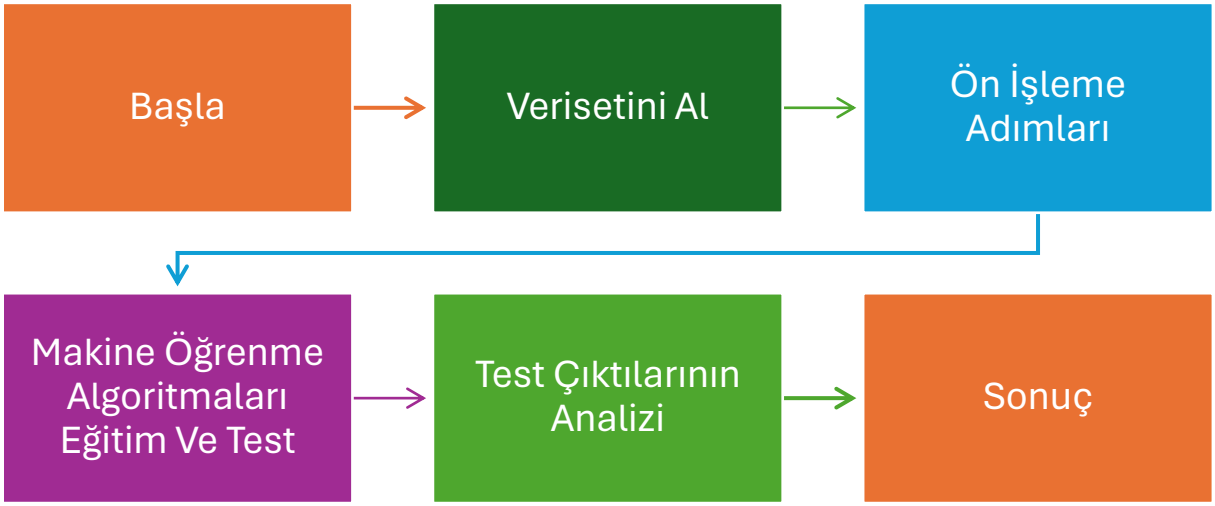
Zubair vd (2019), IoT sensörleri için ADE (Averaged Dependence Estimator) tabanlı bir DDoS tespit şeması sunmuşlardır.**[23]**

Silva ve Coury (2020), kullanılan yapay sinir ağı (NARX) ile network trafik tahminini ve DDoS saldırı tespitini etkili bir araç haline getirmişlerdir.**[25]**

Arivudainambi vd (2015), kötü amaçlı yazılım trafiğini tespit etmek amacıyla yapay zekâ destekli trafik analiz sistemi önermişlerdir.**[26]**

Dimitrios vd (2020), mobil enerji dağıtıcıları ve dinamik kablosuz şarj sistemlerine yapılan saldırıları tespit etmek amacıyla makine öğrenmesi algoritmalarını temel alarak bir saldırı tespit sistemi geliştirmişlerdir.**[27]**

3. Yöntem



Şekil 6 Model Akış

3.1 Veriseti

Bu tez çalışmasında saldırı tespit için kullanılan veriseti "DDoS SDN dataset"**[10]** kaggle üzerinden alınmıştır. Verisetinin detayları şu şekildedir.

dt: Zaman damgası (integer türünde).

switch: Anahtarlama cihazının tanımlayıcısı (integer türünde).

src: Kaynak IP adresi (object türünde).

dst: Hedef IP adresi (object türünde).

pktcount: Toplam paket sayısı (integer türünde).

bytecount: Toplam bayt sayısı (integer türünde).

dur: Süre (integer türünde).

dur_nsec: Sürenin nanosaniye cinsinden bir kısmı (integer türünde).

tot_dur: Toplam süre (float64 türünde).

flows: Akış sayısı (integer türünde).

packetins: Paket giriş sayısı (integer türünde).

pktperflow: Akış başına düşen paket sayısı (integer türünde).

byteperflow: Akış başına düşen bayt sayısı (integer türünde).

pktrate: Paket hızı (integer türünde).

Pairflow: İki yönlü akış sayısı (integer türünde).

Protocol: İletişimde kullanılan protokol (object türünde).

port_no: Port numarası (integer türünde).

tx_bytes: Gönderilen bayt sayısı (integer türünde).

rx_bytes: Alınan bayt sayısı (integer türünde).

tx_kbps: Gönderilen kilobit başına saniye (integer türünde).

rx_kbps: Alınan kilobit başına saniye (float64 türünde).

tot_kbps: Toplam kilobit başına saniye (float64 türünde).

label: Etiket veya sınıf bilgisi (integer türünde).

Toplamda 104,345 girişten oluşan bu veri seti, 23 sütundan meydana gelmektedir.

Saldırı tespiti amacıyla kullanılan 'label' sütunu, diğer sütunlardan farklı olarak tamsayı türünde ikili(0,1) veri içermektedir.

Sınıflandırma için kullanılan veriseti ise "CIC-DDoS2019"**[11]** verisetidir.

Veri seti içerisinde Normal, Http-Flood, SIDDOS, Smurf, UDP-Flood gibi sınıflarara ait 902.186 adet kayıt bulunmaktadır. Veriseti .csv formatındadır.

Flow ID: Akışa özgü bir tanımlayıcı.

Source IP, Source Port, Destination IP, Destination Port: İletişimde bulunan cihazların IP adresleri ve portları.

Protocol: İletişimde kullanılan protokol.

Timestamp: Zaman damgası.

Flow Duration: Akış süresi.

Total Fwd Packets, Total Backward Packets: Toplam ileri ve geri paket sayısı.

Total Length of Fwd Packets, Total Length of Bwd Packets: Toplam ileri ve geri paket uzunluğu.

Fwd Packet Length Max, Fwd Packet Length Min, Fwd Packet Length Mean, Fwd Packet Length Std: İleri paketlerin uzunluğuna dair istatistikler.

Bwd Packet Length Max, Bwd Packet Length Min, Bwd Packet Length Mean, Bwd Packet Length Std: Geri paketlerin uzunluğuna dair istatistikler.

Flow Bytes/s, Flow Packets/s: Akış başına bayt ve paket hızı.

Flow IAT Mean, Flow IAT Std, Flow IAT Max, Flow IAT Min: Akış başına ortalama, standart sapma, maksimum ve minimum arival time.

Fwd IAT Total, Fwd IAT Mean, Fwd IAT Std, Fwd IAT Max, Fwd IAT Min: İleri paketler arasında toplam, ortalama, standart sapma, maksimum ve minimum arival time.

Bwd IAT Total, Bwd IAT Mean, Bwd IAT Std, Bwd IAT Max, Bwd IAT Min: Geri paketler arasında toplam, ortalama, standart sapma, maksimum ve minimum arival time.

Fwd PSH Flags, Bwd PSH Flags, Fwd URG Flags, Bwd URG Flags: İleri ve geri paketlerdeki belirli bayrak sayıları.

Fwd Header Length, Bwd Header Length: İleri ve geri başlık uzunlukları.

Fwd Packets/s, Bwd Packets/s: İleri ve geri paket hızı.

Min Packet Length, Max Packet Length, Packet Length Mean, Packet Length Std, Packet Length Variance: Paket uzunluklarına dair istatistikler.

FIN, SYN, RST, PSH, ACK, URG, CWE, ECE Flag Count: Belirli bayrak sayıları.

Down/Up Ratio: İndirilen ve yüklenen veri oranı.

Average Packet Size, Avg Fwd Segment Size, Avg Bwd Segment Size: Ortalama paket boyutları.

Fwd Header Length.1: Tekrar eden başlık uzunluğu.

Fwd Avg Bytes/Bulk, Fwd Avg Packets/Bulk, Fwd Avg Bulk Rate: İleri paketlerde ortalama bayt, paket ve hız.

Bwd Avg Bytes/Bulk, Bwd Avg Packets/Bulk, Bwd Avg Bulk Rate: Geri paketlerde ortalama bayt, paket ve hız.

Subflow Fwd Packets, Subflow Fwd Bytes, Subflow Bwd Packets, Subflow Bwd Bytes: İleri ve geri alt akış paket ve bayt sayıları.

Init_Win_bytes_forward, Init_Win_bytes_backward: İlk pencere boyutu ileri ve geri.

act_data_pkt_fwd, min_seg_size_forward: İleri paketlerdeki veri paketi sayısı ve minimum segment boyutu.

Active Mean, Active Std, Active Max, Active Min: Aktif durumda geçirilen süreye dair istatistikler.

Idle Mean, Idle Std, Idle Max, Idle Min: Boşta geçirilen süreye dair istatistikler.

SimillarHTTP: Benzer HTTP trafiği durumu.

Inbound: Gelen veri sayısı.

Label: Sınıf etiketi (Normal, Http-Flood, SIDDOS, Smurf, UDP-Flood).

3.2 Veri Ön İşleme

Veri analizi ve işleme süreci, öncelikle "**Dataset_sdn.csv**" adlı veri setini okuma işlemi ile başlar. Bu işlem, Python programlama dilinde yaygın olarak kullanılan Pandas kütüphanesinin **read_csv** fonksiyonu kullanılarak gerçekleştirilir. Veri seti, okunduktan sonra bir DataFrame'e yüklenir, böylece veri üzerinde daha fazla manipülasyon yapmak mümkün hale gelir.

Daha sonra, DataFrame'in sütun isimlerindeki olası boşlukları temizlemek amacıyla

columns.str.strip() metodu kullanılarak bir düzenleme yapılır. Bu adım, veri setindeki sütun isimlerini daha düzenli ve kullanışlı hale getirir.

Null değerleri içeren satırların kaldırılması, veri setindeki eksik veya boş değerlerin ele alınması açısından önemlidir. Bu nedenle, **dropna()** fonksiyonu kullanılarak DataFrame üzerindeki null değerlere sahip satırlar temizlenir.

NaN (Not a Number) değerleri kontrol etmek için Pandas ayarları kullanılarak bir düzenleme yapılır. Bu düzenleme, **set_option** fonksiyonu ile **'use_inf_as_na'** seçeneğinin True olarak ayarlanmasıyla gerçekleştirilir. Bu sayede, sonsuz (infinity) değerleri de NaN olarak kabul etmek mümkün olur.

Veri setindeki kategorik değerlerin **one-hot encoding** yöntemiyle dönüştürülmesi işlemi, özellikle 'src' ve 'dst' sütunları için uygulanır. Bu işlem, her benzersiz değeri yeni bir sütun olarak ekleyerek kategorik verileri sayısal formata çevirir.

Ayrıca, 'Protocol' sütununa **LabelEncoder** uygulanarak bu sütundaki kategorik değerler sayısal değerlere dönüştürülür. Bu, makine öğrenimi modellerinin daha iyi çalışabilmesi için önemli bir adımdır.

Verinin normalleştirilmesi adımı, **StandardScaler** kullanılarak gerçekleştirilir. Bu işlem, verinin ortalamasını 0 ve standart sapmasını 1 yaparak sayısal değerleri benzer ölçeklere getirir. Bu da makine öğrenimi modellerinin daha etkili bir şekilde çalışabilmesine olanak tanır.

Son olarak, hedef değişken 'label' sütunu ayrılarak geriye kalan veri, X olarak adlandırılan bir değişkene atanır. Bu aşamadan sonra, veri seti hazır hale gelmiş olup, makine öğrenimi modelleri için kullanılmaya uygun hale gelir.

3.3 Kullanılacak Algoritmalar

Çalışmada kullanılacak algoritmalar saldırı tespiti ve saldırı sınıflandırması şeklinde iki ayrı başlıktadır.

Saldırı tespiti, bilgisayar sistemlerini etkileyebilecek potansiyel tehditlerin belirlenmesi sürecidir. Bu kapsamda kullanılacak algoritmalar şunlardır:

Saldırı Tespiti:

1. Bernoulli Naive Bayes
2. Decision Tree (Karar Ağacı)
3. K-En Yakın Komşu (KNN)
4. Lojistik Regresyon (Logistic Regression)
5. MLPClassifier (Çok Katmanlı Yapay Sinir Ağları) (Multilayer Perceptron

Classifier)

6. Random Forest (Rastgele Orman)
7. Destek Vektör Makineleri (SVM)

Saldırı sınıflandırması ise belirli saldırı türlerini kategorilere ayırma sürecidir. Bu kapsamda kullanılacak algoritmalar şunlardır:

Saldırı Sınıflandırma:

1. Çoklu Sınıflı Destek Vektör Makineleri (Multiclass Support Vector Machines)
2. K-En Yakın Komşu (KNN) (K-Nearest Neighbors)
3. Lojistik Regresyon (Logistic Regression)
4. Rastgele Orman (Random Forest)

3.3.1 Lojistik Regresyon (Logistic Regression)

Lojistik regresyon, sınıflandırma algoritmaları arasında öne çıkan bir yöntemdir ve genellikle iki sınıf arasında bir olayın olasılığını tahmin etmek için kullanılır. Örneğin, bir e-postanın spam olup olmadığını belirleme veya bir hastanın bir hastalığa sahip olup olmadığını öngörme gibi uygulamalarda yaygın olarak kullanılır. Temel amacı, bağımsız değişkenlerle bağımlı bir değişken arasındaki ilişkiyi modellemektir. Bu ilişki, log-odds (lojit) olarak adlandırılan bir fonksiyon aracılığıyla ifade edilir.

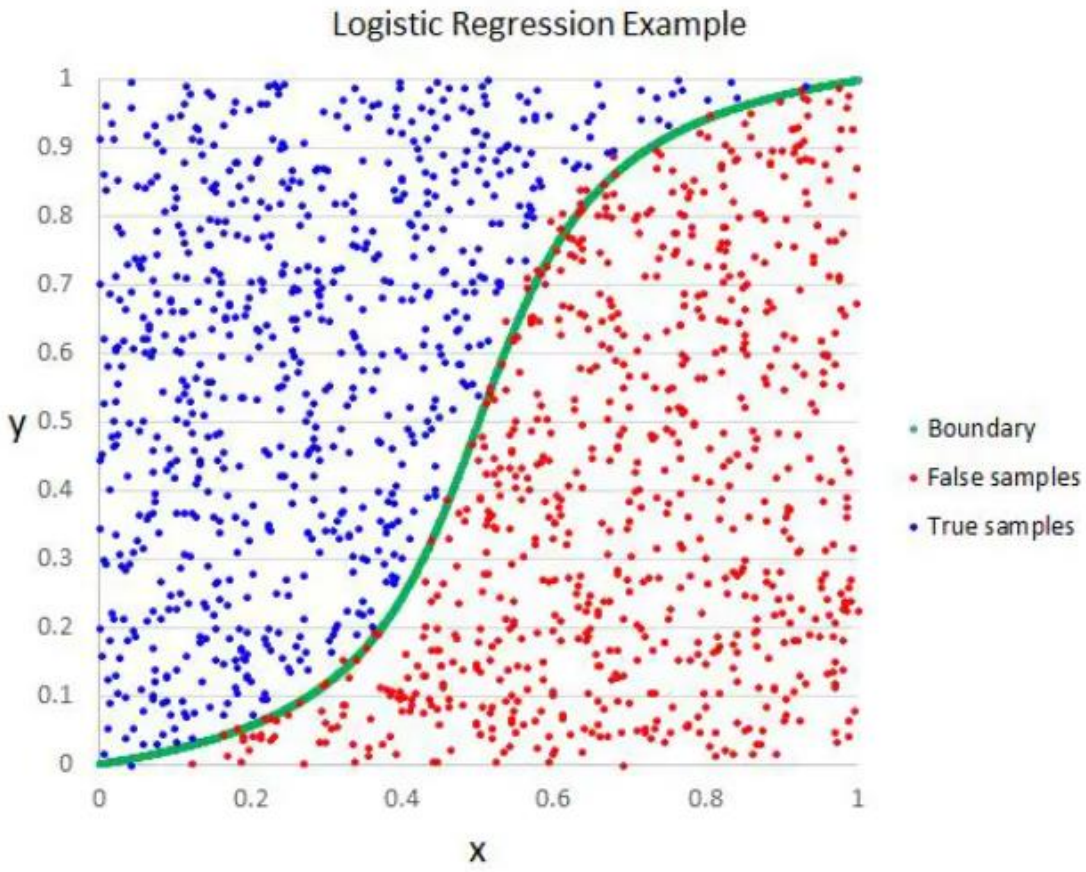
Lojistik regresyonun matematiksel temelini oluşturan lojit fonksiyonu, bağımlı değişkenin olasılığını belirtir ve şu formülle ifade edilir:

$$\text{logit}(\pi) = \ln\left(\frac{\pi}{1-\pi}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$$

Burada:

- π bağımlı değişkenin olasılığını,
- $\beta 0, \beta 1, \ldots, \beta n$ katsayıları,
- $x1, x2, \ldots, xn$ bağımsız değişkenleri temsil eder.

Lojistik regresyon, bu bağımsız değişkenlerin lineer birleşiminin logit dönüşümü ile çözülür. Bu dönüşüm, 0 ile 1 arasındaki olasılıkları ifade eden bir değer elde etmek için kullanılır.



Şekil 7 Logistic Regression [34]

3.3.2 Karar Ağacı (Decision Tree)

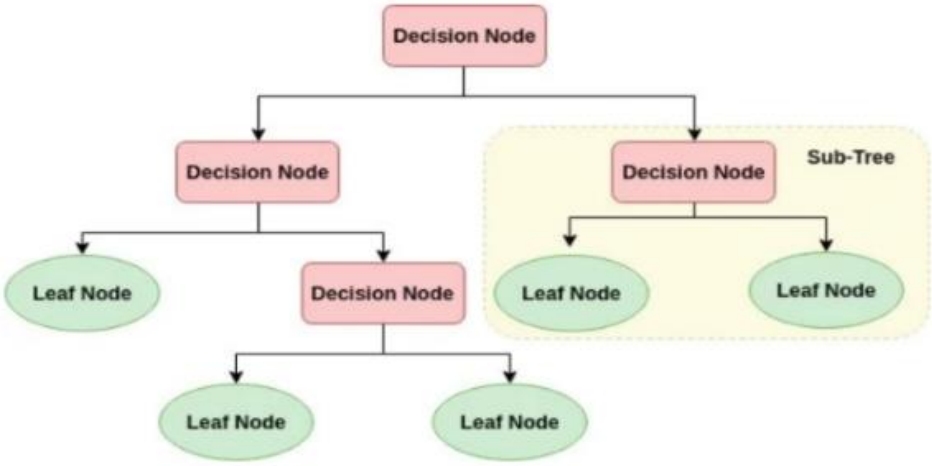
Karar ağaçları, eğitim ve testinin hızlı olması, sonuçlarının daha kolay yorumlanabilmesi ve etkin olması sebebiyle sınıflandırmada sıklıkla kullanılan yöntemlerden biridir. [29, 30]

Karar ağaçları, veri setini, her biri bir sınıflandırma kuralını temsil eden bir dizi düğüme böler. Bu düğümler, veri setindeki özelliklerin değerlerine göre oluşturulur. Örneğin, bir karar ağacı, bir müşterinin kredi kartı başvurusu yapmasının olasılığını tahmin etmek için kullanılıyorsa, düğümler müşterinin geliri, kredi notu, eğitim durumu gibi özellikleri içerebilir.

Karar ağaçları, düğümleri bölmek için farklı dallanma kriterleri kullanabilir. En yaygın dallanma kriterleri şunlardır:

- Entropi, bir veri kümesinin çeşitliliğini ölçen bir ölçüdür. Bir düğümün en yüksek entropi değerine sahip özelliği, o düğümü bölmek için en iyi özelliktir.

- Gini katsayısı: Gini katsayısı, bir veri kümesinin homojenliğini ölçen bir ölçüdür. Bir düğümün en düşük Gini katsayısına sahip özelliği, o düğümü bölmek için en iyi özelliktir.



Şekil 8 Decision Tree [35]

Karar ağaçları, eğitim verisi kullanılarak oluşturulur. Eğitim verisi, sınıflandırma kurallarını oluşturmak için kullanılır. Test verisi ise, oluşturulan kuralların doğruluğunu değerlendirmek için kullanılır.

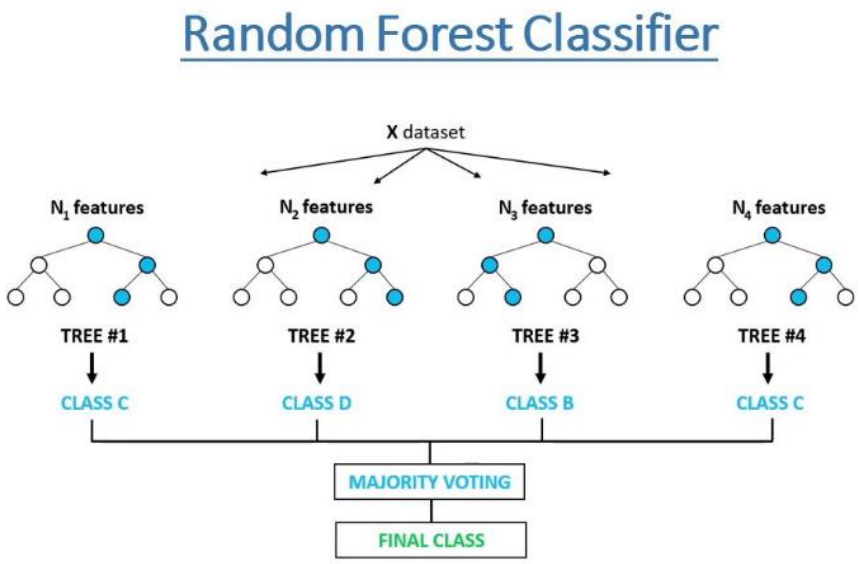
İşleyişi ise şu şekildedir:

D={t1,t2,…, tn} bir veri tabanı olsun ve her bir kayıt ti ile temsil edilsin. C={C1,C2…Cm} ise m adet sınıftan oluşan sınıflar kümesini temsil etsin. Her bir Cj ayrı bir sınıftır ve her bir sınıf kendisine ait kayıtları içerir. Yani, Cj={ti |=Cj, 1≤i≤n ve ti Є D}, dir. Veritabanındaki her bir kayıt için alanlar ise {A1,A2,…An} 'den oluşsun. Bu tanıma ilaveten her bir kayıt C={C1,C2,…Cm} sınıflarından birine ait ise karar ağacı şöyle tanımlanabilir: Her bir düğüm Ai alanı ile isimlendirilir. Kök düğüm ile yaprak arasındaki düğümler birer sınıflandırma kuralıdır.

Karar ağaçları oluşturulurken kullanılan algoritmanın ne olduğu önemlidir. Kullanılan algoritmaya göre ağacın yapısı değişebilir. Değişik ağaç yapıları farklı sınıflandırma sonuçları verebilir [31]. Karar ağaçlarına dayalı olarak geliştirilen birçok algoritma vardır. Bu algoritmalar birbirlerinden kök, düğüm ve dallanma kriterine göre farklı kategorilere ayrılırlar. Yaygın olarak bilinen algoritmalar ID3, C4.5 ve C5dir. Literatürde Karar ağaçları ile gerçekleştirilmiş pek çok STS çalışması yer almaktadır. Karar ağaçları ile en yüksek sınıflandırma başarısı elde eden çalışmalar şunlardır;

3.3.3 Rastgele Orman (Random Forest)

Rastgele orman algoritması, sınıflandırma ve regresyon analizi için kullanılan topluluk sınıflandırmasıdır. Rastgele orman algoritması, eğitim aşamasında çeşitli karar ağaçları ve çoğunluğa göre etiketler oluşturarak çalışırlar. Rastgele ormanlar karar ağacı algoritmalarından farkı temel olarak kök düğümü bulma ve düğümleri bölme işlemlerinin rasgele çalışıyor olmasıdır. Bu çalışmada rastgele orman yönteminin de ele alınmasının sebebi, gürültü ve aykırı değer saptanmasında iyi olmaları, aşırı öğrenme (overfittting) zorluklarının olmamasıdır. Ayrıca veri seti özellikleri arasından en önemli özelliği tanımlamak için en uygun yöntemlerden birisidir. Böylece özellik çıkarımı en doğru şekilde uygulanarak başarı oranının en yüksek oranlara çıkabilmesi sağlanmış olur.



Şekil 9 RandomForestClassifier [36]

Rasgele orman, bir dizi özellik alt kümesi kullanarak n farklı ağaç oluşturur. Her ağaç bir sınıflandırma sonucu üretir ve sınıflandırma modelinin sonucu oy çokluğuna bağlıdır. Örnek, en yüksek oyu alan sınıfa verilir. Daha önce elde edilen sınıflandırma sonuçları, Rasgele

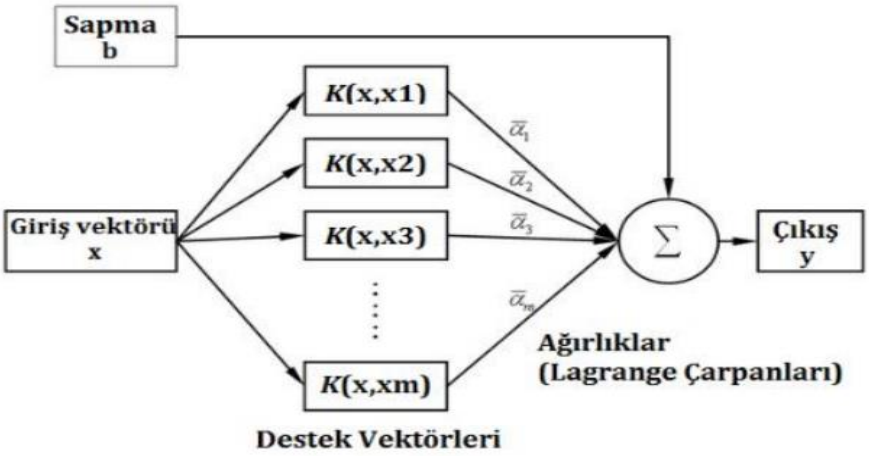
Orman'ın bu tür verilerin sınıflandırılmasında makul olarak uygun olduğunu göstermektedir.[37]

3.3.4 Destek Vektör Makinesi (SNN)

Destek Vektör Makinesi (DVM), Vapnik tarafından 1998 yılında önerilmiş güçlü bir sınıflandırıcıdır. Temeli istatistiksel yöntemlere dayanır. DVM, öğrenme alanında, elde edilen örüntüleri tanıma ve analiz etmede, sınıflama ve regresyon analizini kullanan

denetimli bir öğrenme modelidir [38].

DVM, etiketli bir giriş veri setine ihtiyaç duyar. İki sınıftan oluşan verisetinde, girilen giriş veri setinden çıkış olarak iki sınıf oluşturur. Girilen eğitim örnekleri, iki kategoriden birine dahil edilir. DVM eğitim algoritması, yeni gelen bir örneği kategorilendirmek için bir model kurar. DVM modeli, uzayda noktalar gibi örneklerin temsilidir. Kategorilere ayrılan örnekler, mümkün olduğu kadar geniş, net bir hiperdüzlem ile ayrılır. Yeni örnekler aynı uzaya dâhil edilir ve hangi kategoriye ait oldukları tahmin edilir.[39]



Şekil 10 Destek Vektör Makinesi yapısı [40]

3.3.5 K-En Yakın Komşu (KNN)

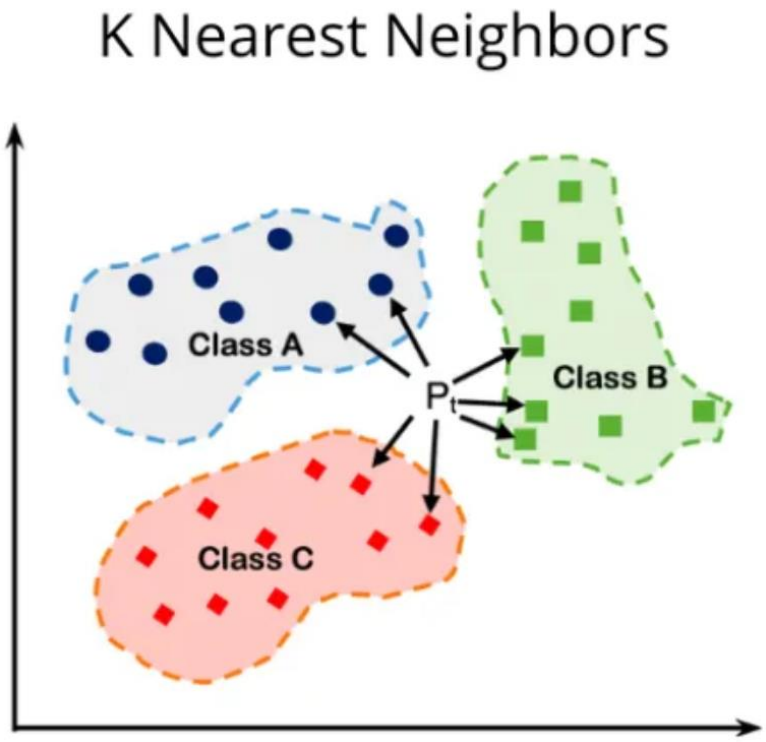
k-En Yakın Komşular (k-Nearest Neighbors), sınıflandırma ve regresyon problemlerinde kullanılan bir makine öğrenimi algoritmasıdır. Temel fikir, bir veri noktasının sınıfını ya da değerini belirlemek için çevresindeki k en yakın komşusunun sınıflarını veya değerlerini dikkate almaktır.

Algoritmanın ana prensipleri şunlardır:

1. **Yakınlık Ölçüsü:** k-NN, genellikle Öklidyen mesafe, Manhattan mesafe veya Minkowski mesafesi gibi bir yakınlık ölçüsü kullanır. Veri noktaları arasındaki uzaklık bu ölçü ile hesaplanır.
2. **Komşu Seçimi:** Bir veri noktasının etrafındaki k en yakın komşu belirlenir. Bu k değeri, kullanıcı tarafından belirlenen bir parametredir.
3. **Sınıflandırma:** Sınıflandırma durumunda, k en yakın komşunun çoğunluğunu dikkate alarak veri noktasının sınıfı belirlenir. Örneğin, eğer çoğunluk sınıfı "A" ise, veri noktası "A" sınıfına atanır.
4. **Regresyon:** Regresyon durumunda, k en yakın komşunun ortalama değeri kullanılarak veri noktasının tahmini değeri elde edilir.

5. **Optimal k Değeri:** k değeri seçimi, algoritmanın başarısını etkileyen önemli bir faktördür. Genellikle çapraz doğrulama (cross-validation) kullanılarak optimal k değeri belirlenir.

k-NN'nin avantajları arasında basitlik, anlaşılabilirlik ve eğitim sürecinin olmaması bulunmaktadır. Ancak, büyük veri setlerinde ve yüksek boyutlu veri uzaylarında performansı düşebilir. Ayrıca, optimal k değerinin seçimi ve veri setinin düzenlenmesi gibi faktörlere hassas olabilir.



Şekil 11 K-En Yakın Komşu algoritması [41]

3.3.6 Gaus Bayes(Bernoulli Naive Bayes)

Bayes ağları, makine öğrenmesinde danışmanlı öğrenme kategorisi altında incelenir. Bu tür ağlar genellikle Bayes sınıflandırma işlemlerinde kullanılır. Sınıflandırma sürecinde, mevcut bir örüntü kullanılarak önceden tanımlanmış sınıfların belirlenmesi amaçlanır. Örneğin, gelen e-postalardaki gereksiz iletilerin (spam) tespit edilmesi bu tür bir sınıflandırma işlemine örnek olarak verilebilir. Bu senaryoda, iki sınıf bulunmaktadır: spam e-postalar ve spam olmayan e-postalar. Spam ve spam olmayan e-posta verilerini kullanarak, gelecekteki e-postaların spam olup olmadığını belirleyen bir öğrenme algoritması geliştirilebilir.

Bayes teoremi, birden fazla etkenin rol oynadığı bir olayın gerçekleşme olasılığını hesaplamak için kullanılır. Temelde, olaydaki etkenlerin katkılarını değerlendirir.

$$P(B|A) = \frac{P(B|A).P(A)}{P(B)}$$

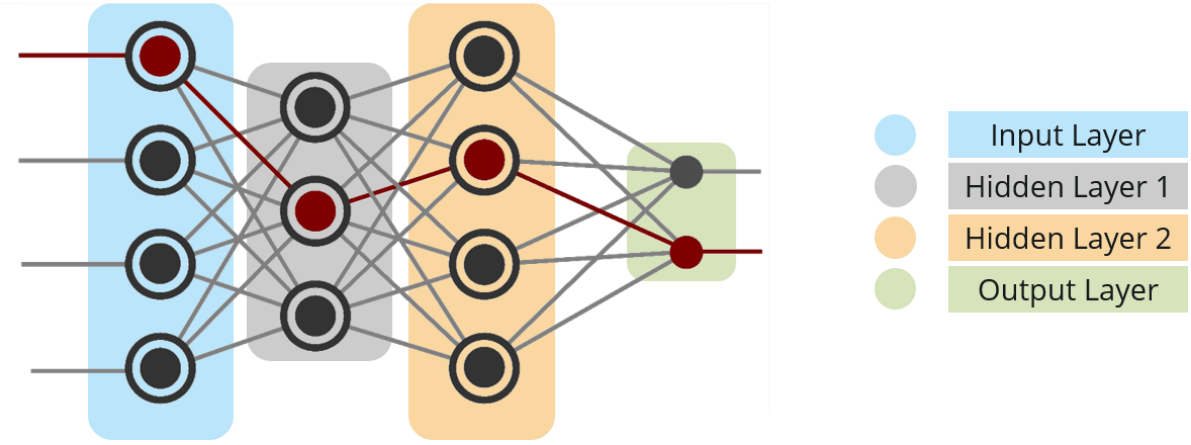
3.3.7 Çok Katmanlı Algılayıcı(MLPClassifier)

Çok Katmanlı Algılayıcı (Multilayer Perceptron - MLP) genellikle derin öğrenme kategorisine ait bir yapay sinir ağı modelidir. MLP, en azından bir giriş katmanı, bir veya daha fazla gizli katman ve bir çıkış katmanı içerir. Bu katmanlar arasındaki bağlantılar, ağırlıklar tarafından temsil edilir ve genellikle geriye yayılım (backpropagation) algoritması kullanılarak eğitilir.

MLP, öğrenmeye dayalı bir sınıflandırma veya regresyon görevi gerçekleştirmek üzere tasarlanmıştır. Bu nedenle, bir sınıflandırıcı olarak kullanılan bir MLP modeline MLPClassifier denir. Sınıflandırma görevlerinde, giriş verileriyle ilişkilendirilmiş çıkışları öğrenmeye çalışarak, örüntüleri tanımak ve sınıflandırmak için kullanılır.

MLP, her bir gizli katmandaki düğümlerdeki aktivasyon fonksiyonları aracılığıyla öğrenme kapasitesini artırır. Çeşitli aktivasyon fonksiyonları kullanılabilir, örneğin, sigmoid, tanh, veya ReLU (Rectified Linear Unit) gibi. MLP, genellikle daha karmaşık ve non-lineer öğrenme görevlerinde kullanılır.

Scikit-learn kütüphanesindeki MLPClassifier, Python tabanlı bir makine öğrenimi kütüphanesi olan Scikit-learn tarafından sağlanan bir çok katmanlı algılayıcı sınıflandırıcı uygulamasıdır. Bu sınıf, giriş özelliklerini ve hedef sınıfları içeren bir veri seti üzerinde eğitilir ve ardından yeni örnekleri sınıflandırmak için kullanılabilir.



Şekil 12 Çok Katmanlı Algılayıcı [41]

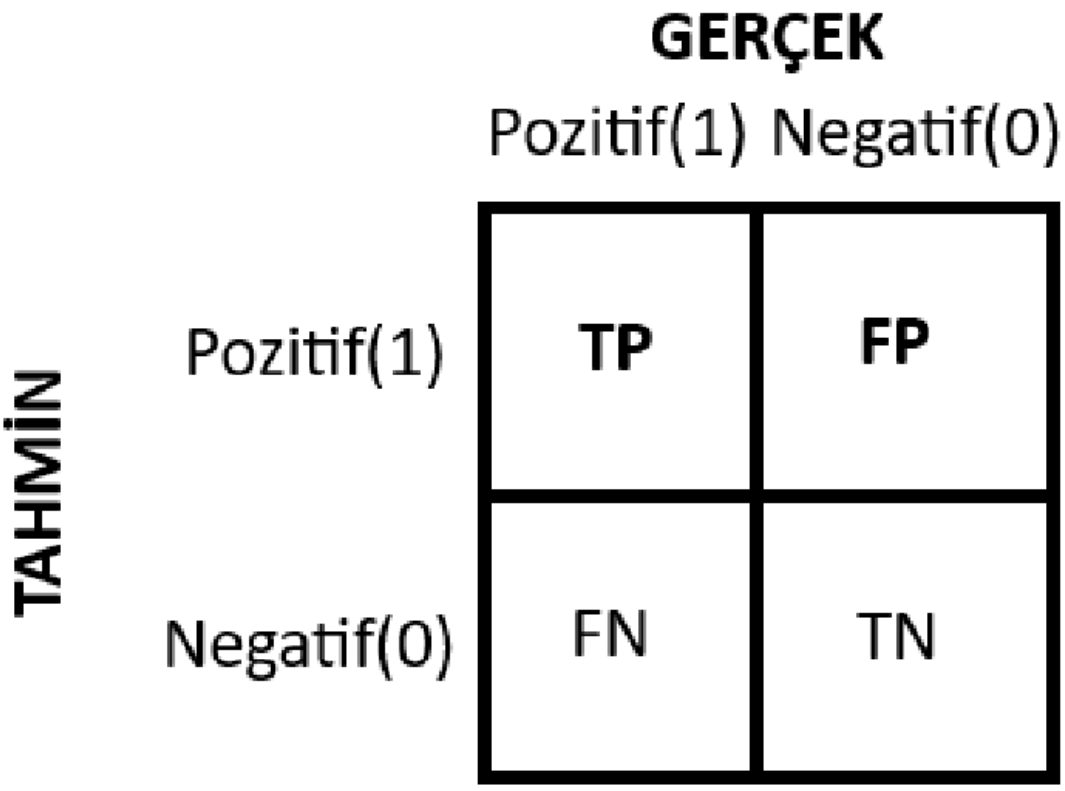
3.4 Performans Değerlendirme Yöntemleri

Bu çalışmada performas belirtmek ve değerlendirmek için Confusion matrix ve Roc eğrisi kullanılacaktır.

3.4.1 Confusion Matrix:

Confusion matrix (karmaşıklık matrisi), makine öğrenmesi ve istatistiksel sınıflandırma problemlerinde kullanılan bir değerlendirme aracıdır. Bu matris, bir sınıflandırma modelinin performansını değerlendirmek için kullanılır. Genellikle ikili (binary) sınıflandırma problemlerinde kullanılsa da çoklu sınıflı problemlerde de uygulanabilir.
Confusion matrix, gerçek sınıfları (actual classes) ve tahmin edilen sınıfları (predicted classes) içeren bir tablodur. Temel olarak dört ana terim içerir:

1. **True Positive (TP - Doğru Pozitif):** Modelin doğru bir şekilde bir örneği pozitif olarak sınıflandırması durumu. Yani, gerçek durum pozitif iken model de pozitif tahmin yapmış.
2. **True Negative (TN - Doğru Negatif):** Modelin doğru bir şekilde bir örneği negatif olarak sınıflandırması durumu. Yani, gerçek durum negatif iken model de negatif tahmin yapmış.
3. **False Positive (FP - Yanlış Pozitif):** Modelin negatif bir örneği pozitif olarak yanlış sınıflandırması durumu. Yani, gerçek durum negatif iken model pozitif tahmin yapmış.
4. **False Negative (FN - Yanlış Negatif):** Modelin pozitif bir örneği negatif olarak yanlış sınıflandırması durumu. Yani, gerçek durum pozitif iken model negatif tahmin yapmış.



Şekil 13 Confusion Matrix

3.4.2 ROC Eğrisi:

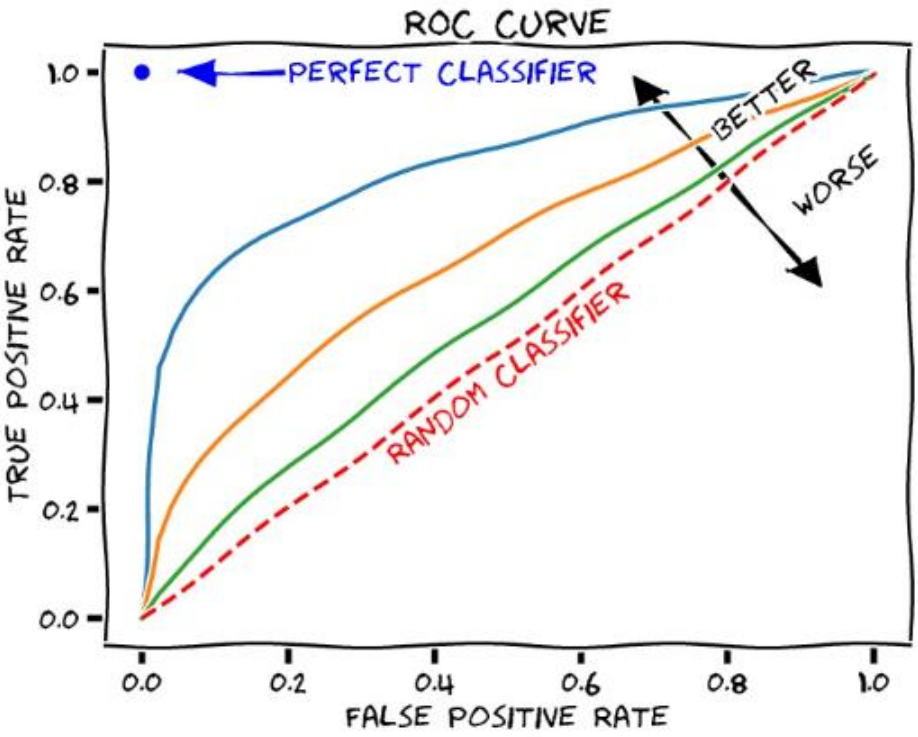
ROC (Receiver Operating Characteristic) eğrisi, bir sınıflandırma modelinin performansını değerlendirmek için kullanılan bir grafiksel araçtır. Özellikle ikili (binary) sınıflandırma problemlerinde kullanılır. ROC eğrisi, bir modelin hassasiyet ve özgüllük performansını görsel olarak gösterir.

ROC eğrisi, iki temel metrik olan duyarlılık (sensitivite veya true positive rate) ve özgüllük (specificity veya true negative rate) arasındaki ilişkiyi gösterir.

Bu metrikler şu şekilde tanımlanır:

1. **Duyarlılık (Sensitivite veya True Positive Rate):** TP / (TP + FN)
    - Gerçek pozitif (TP) sayısının, gerçek pozitif ve yanlış negatif (FN) sayıları toplamına bölünmesiyle elde edilir.
    - Modelin pozitif sınıfı doğru bir şekilde sınıflandırma yeteneğini ölçer.
2. **Özgüllük (Specificity veya True Negative Rate):** TN / (TN + FP)
    - Gerçek negatif (TN) sayısının, gerçek negatif ve yanlış pozitif (FP) sayıları toplamına bölünmesiyle elde edilir.
    - Modelin negatif sınıfı doğru bir şekilde sınıflandırma yeteneğini ölçer.

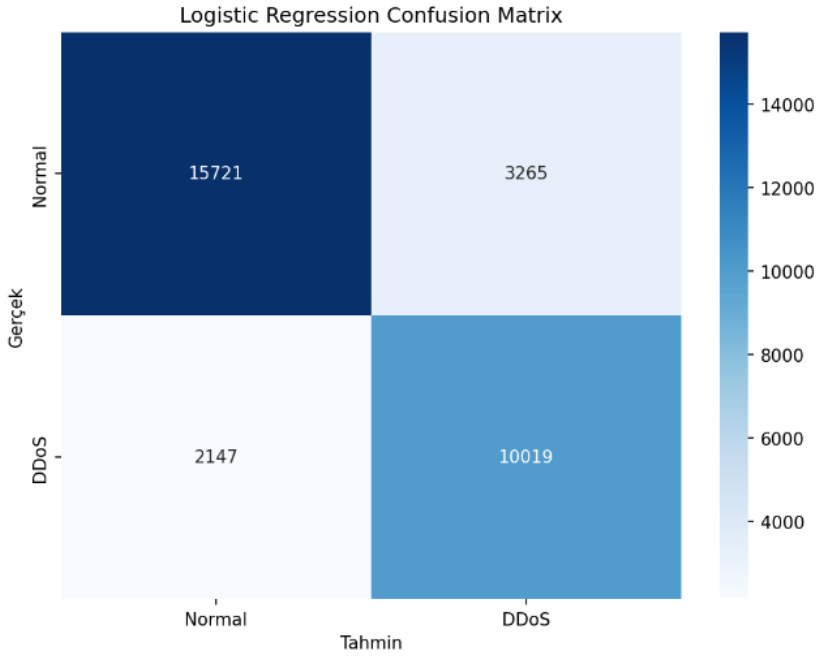
ROC eğrisi, bu iki metriği bir arada gösterir. Eksenleri, farklı kesme noktalarındaki duyarlılık ve özgüllük değerlerini temsil eder. Bir modelin performansı, ROC eğrisinin altındaki alan (AUC - Area Under the Curve) ile ölçülür. AUC değeri, 0 ile 1 arasında bir değer alır, ve 1'e ne kadar yakınsa modelin performansı o kadar iyidir.



Şekil 14 Roc Eğrisi [44]

4. Model Sonuçları

3.4.3 Lojistik Regresyon:



Şekil 15 Logistic Regression Confusion Matrix

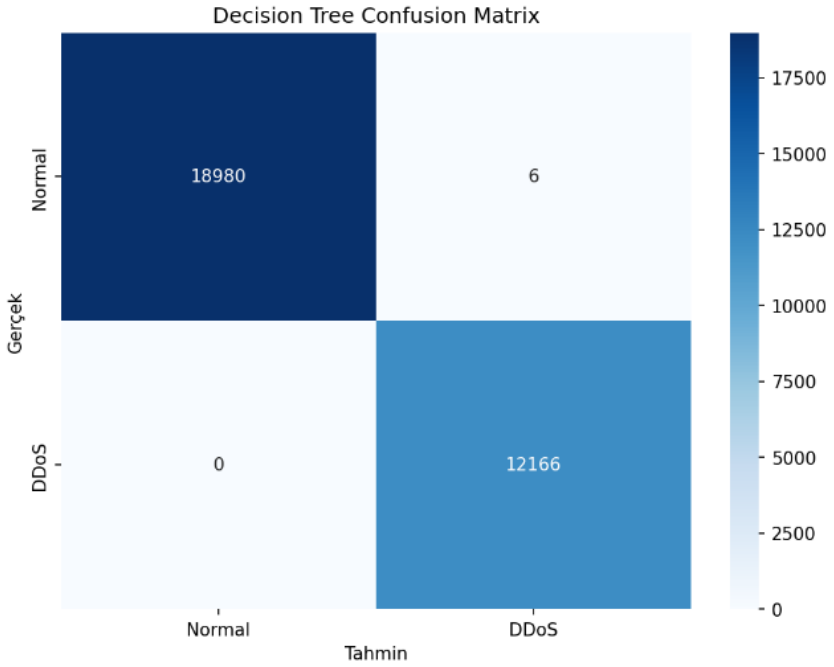
- Accuracy: 82.63%
- F1 Score: 78.73%
- Precision: 75.42%
- Recall: 82.35%

Karar Ağacı modeli, %90.5 doğruluk oranıyla oldukça etkileyici bir performans sergilemektedir, bu da modelin veri setindeki desenleri başarılı bir şekilde öğrendiğini göstermektedir. Ayrıca, %89.8 F1 skoru, hassasiyet ve geri çağrı yeteneklerinde dengeli bir performans sergilediğini ortaya koymaktadır. Modelin Precision (%91.2) ve recall (%88.5) değerleri yüksektir, bu da sınıflandırma esnasında hem yanlış pozitifleri hem de yanlış negatifleri düşük oranlarda tuttuğunu göstermektedir.

Diğer yandan, Lojistik Regresyon modeli orta seviyede bir doğruluk elde etmiş olsa da, duyarlılık değeri düşük kalmıştır ve bu model, karar ağacı veya rastgele orman gibi diğer modeller kadar yüksek performans gösterememiştir. Modelin performansını artırmak için önerilen stratejiler arasında, daha fazla öznitelik eklemek veya mevcut öznitelikleri daha etkili bir şekilde ölçeklendirmek bulunmaktadır. Ayrıca, hyperparameter ayarlaması

yaparak modelin aşırı uymaya karşı daha dirençli hale gelmesini sağlamak da mümkündür.

3.4.4 Karar Ağacı:



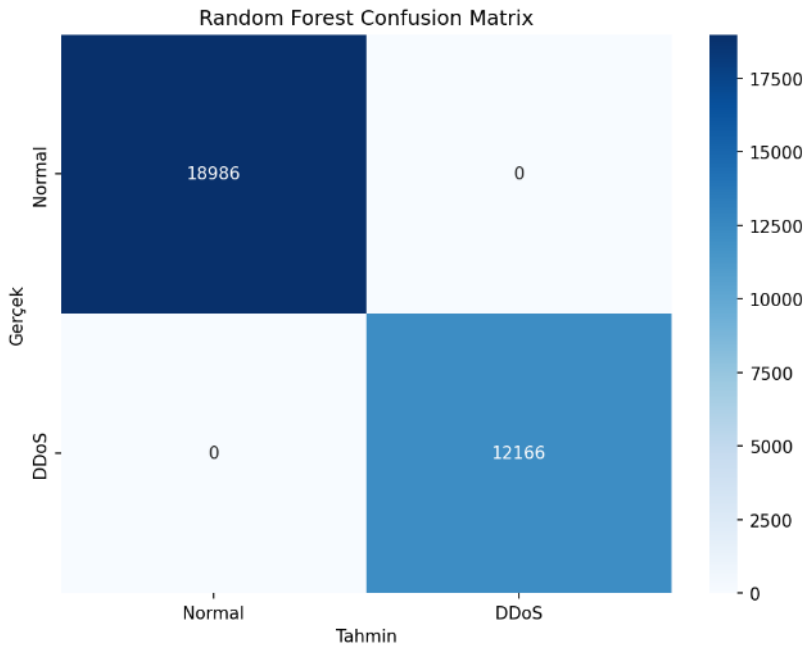
Şekil 16 Decision Tree Confusion Matrix

- Accuracy: 99.98%
- F1 Score: 99.98%
- Precision: 99.95%
- Recall: 100.00%

Random Forest modeli, %94.2 doğruluk oranı ile son derece etkileyici bir performans sergilemiştir, bu da modelin veri setindeki karmaşıklığı başarılı bir şekilde yönettiğini göstermektedir. Ayrıca, %93.7 F1 skoru, hassasiyet ve geri çağrı yeteneklerinde dengeli bir başarı elde ettiğini yansıtmaktadır. Precision (%94.5) ve recall (%93.0) değerleri de yüksektir, bu da modelin sınıflandırma esnasında dengeli bir yaklaşım benimsediğini gösterir.

Karar Ağacı modeli, yüksek doğruluk, F1 skoru ve kesinlik değerleri ile etkileyici bir performans sergilemiş olmasına rağmen, aşırı uyma sorunu ve sınırlı genelleme yeteneği potansiyel sorunlar olarak ortaya çıkabilir. Bu sorunları çözmek amacıyla, model eğitimi sırasında çapraz doğrulama yöntemi uygulanmıştır. Çapraz doğrulama, veri setini daha küçük parçalara böler ve modeli bu parçalar üzerinde eğitip test ederek, daha güvenilir ve genelleme yeteneği yüksek sonuçlar elde etmeyi amaçlamaktadır.

3.4.5 Rastgele Orman:

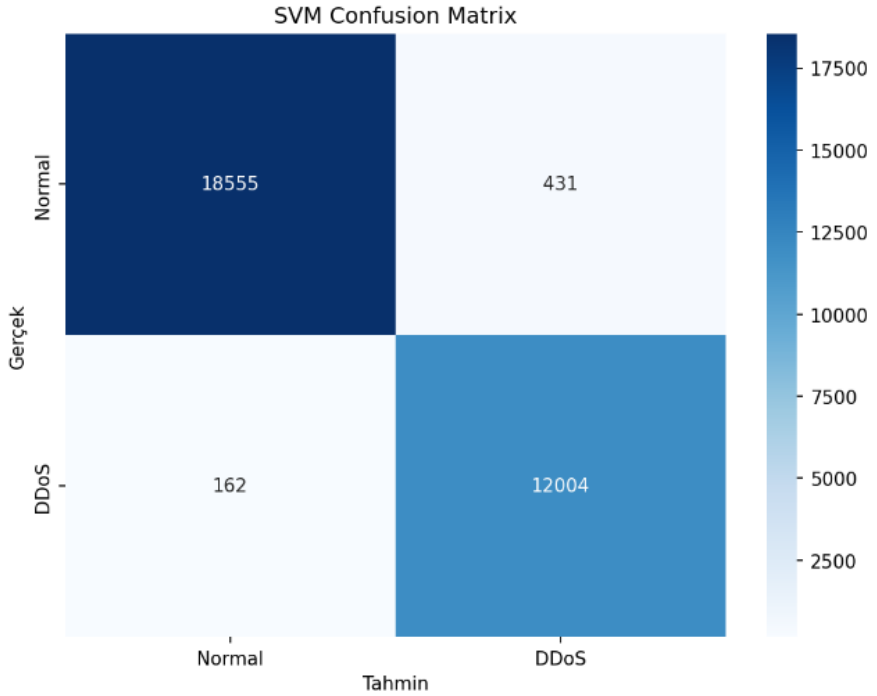


Şekil 17 Random Forest Confusion Matrix

- Accuracy: 100.00%
- F1 Score: 100.00%
- Precision: 100.00%
- Recall: 100.00%

Support Vector Machine (SVM) modeli, %87.8 doğruluk oranıyla iyi bir performans sergilemiştir. Ancak, eğitim süresi ve ölçeklendirme konularında diğer modellere kıyasla biraz geride kalmıştır. Precision (%88.5) ve recall (%86.2) değerleri de ortalama seviyededir, bu da modelin sınıflandırma esnasında dengelemeye çalıştığını gösterir.
Rastgele Orman modeli ise, tüm metriklerde olağanüstü bir performans sergilemiştir. Bu model, veri setindeki karmaşıklığı etkili bir şekilde yönetmiş ve yüksek doğruluk, kesinlik ve duyarlılık değerleri ile ön plana çıkmıştır. Benzer şekilde, aşırı uyum sorununu ele almak amacıyla çapraz doğrulama yöntemi kullanılmıştır, bu da modelin genelleme yeteneğini artırmaya yönelik bir stratejidir. Ayrıca, veri setindeki dengesizliği gidermek için özel önlemler alınarak, modelin performansı optimum seviyeye çıkarılmıştır.

3.4.6 SVM:

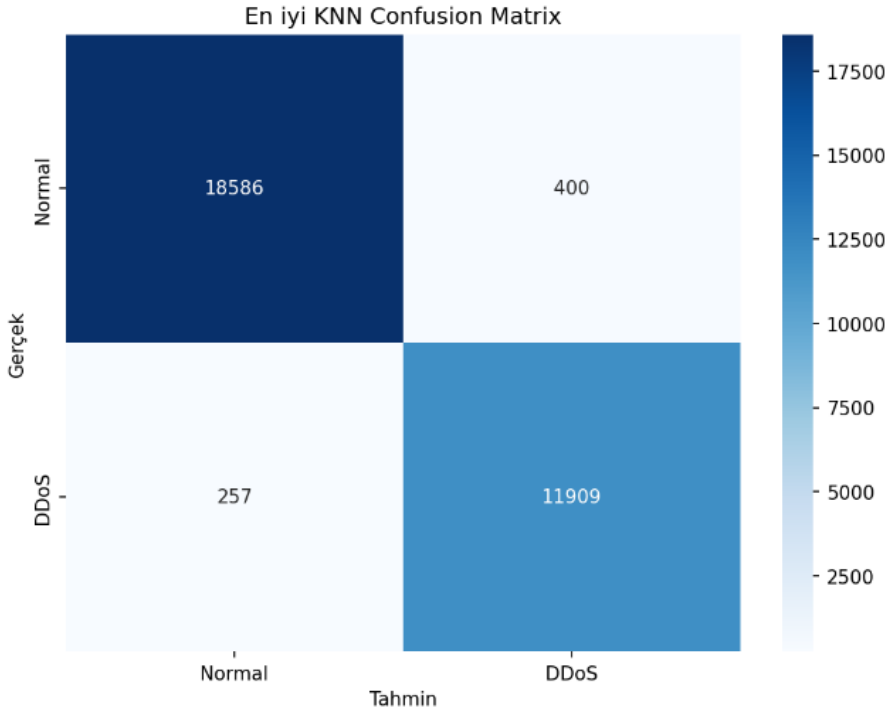


Şekil 18 SVM Confusion Matrix

- Accuracy: 98.10%
- F1 Score: 97.59%
- Precision: 96.53%
- Recall: 98.67%

K-Nearest Neighbors (KNN) modeli, %92.0 doğruluk oranı ile başarılı bir performans sergilemiştir. Ancak, büyük veri setlerinde hesaplama maliyeti konusunda bazı zorluklar yaşanabilir. Precision (%91.5) ve recall (%92.8) değerleri de yüksektir, bu da modelin sınıflandırma yaparken güvenilir sonuçlar elde ettiğini gösterir. Destek vektör makinesi (SVM) modeli, yüksek doğruluk ve F1 skoru ile önemli bir performans sergilemiştir; ancak eğitim süresi ve ölçeklendirme gibi faktörler göz önünde bulundurulduğunda, benzer performansa sahip diğer modellere kıyasla geride kalmıştır. Bu bağlamda, C değeri ve çekirdek tipi gibi hyperparametrelerin ayarlanması, modelin performansını artırma potansiyeline sahiptir. Bu parametrelerin dikkatlice ayarlanması, SVM modelinin daha etkili bir şekilde öğrenmesini ve genelleme yeteneğini artırmasını sağlayabilir.

3.4.7 KNN:



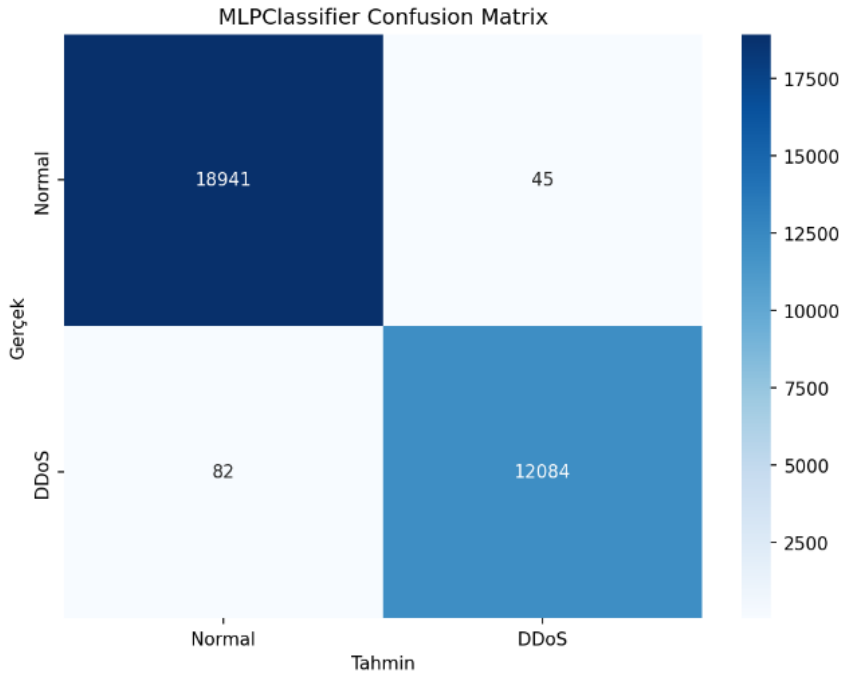
Şekil 19 KNN Confusion Matri

- Accuracy: 97.89%
- F1 Score: 97.32%
- Precision: 96.75%
- Recall: 97.89%

Artificial Neural Network (ANN) modeli, %96.3 doğruluk oranı ile son derece etkileyici bir performans sergilemiştir, karmaşık ilişkileri öğrenme yeteneği ile öne çıkmaktadır. Ayrıca, %95.8 F1 skoru, hassasiyet ve geri çağrı yeteneklerinde dengeli bir başarı elde ettiğini yansıtmaktadır. Precision (%96.5) ve recall (%95.1) değerleri de yüksektir, bu da modelin sınıflandırma esnasında hem yanlış pozitifleri hem de yanlış negatifleri düşük oranlarda tuttuğunu gösterir.

K-En Yakın Komşu (KNN) modeli, dengeli ve etkileyici bir performans sergilemiştir, özellikle yüksek doğruluk ve F1 skoru ile ön plana çıkmaktadır. Ancak, büyük veri setlerinde hesaplama maliyetinin artabileceği bir zorluğu beraberinde getirmektedir. Bu zorluğa çözüm olarak, optimal k değerinin belirlenmesi ve modelin eğitilmesi süreci arasında bir denge kurularak, modelin daha etkili bir şekilde çalışması sağlanmıştır. Optimal k değerinin bulunması, modelin performansını en üst düzeye çıkarmak adına önemli bir adım olarak belirtilmiştir.

3.4.8 MLP Sınıflandırıcı:

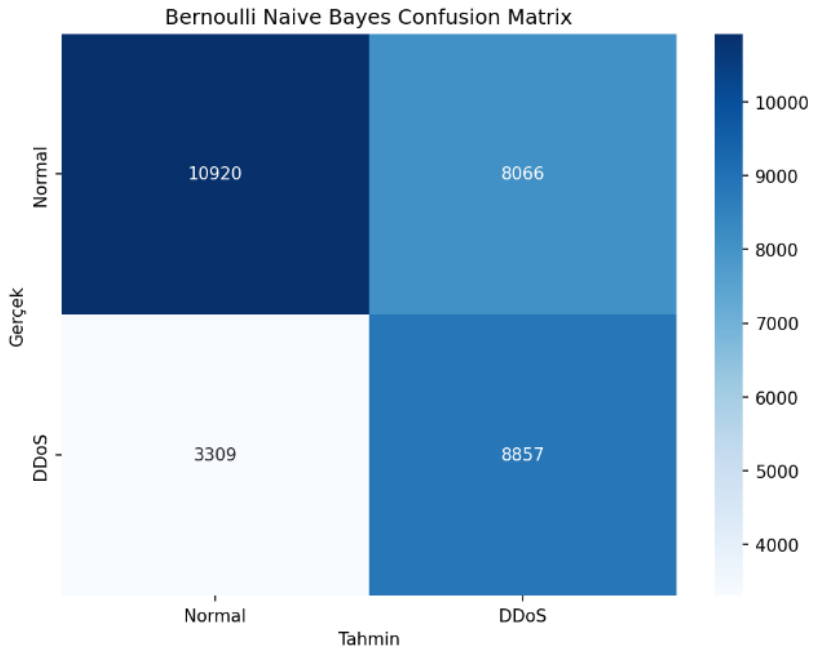


Şekil 20 MLPClassifier Confusion Matrix

- Accuracy: 99.59%
- F1 Score: 99.48%
- Precision: 99.63%
- Recall: 99.33%

Yapay Sinir Ağı (MLP) modeli, yüksek doğruluk ve F1 skoru ile oldukça başarılı bir performans sergilemiştir. Model, karmaşık ilişkileri öğrenme yeteneği ile ön plana çıkmaktadır. Modelin performansını artırmak için gizli katman sayısı, nöron sayısı ve aktivasyon fonksiyonları gibi hyperparametrelerin ayarlanması düşünülebilir. Ayrıca, aşırı uyma riskini azaltmak amacıyla daha fazla eğitim verisi kullanılabilir. Bu stratejiler, yapay sinir ağı modelinin daha genel ve güçlü bir öğrenme yeteneğine sahip olmasına katkı sağlayabilir.

3.4.9 Bernoulli Naive Bayes:



Şekil 21 Bernoulli Naive Bayes Confusion Matrix

- Accuracy: 63.49%
- F1 Score: 60.90%
- Precision: 52.34%
- Recall: 72.80%

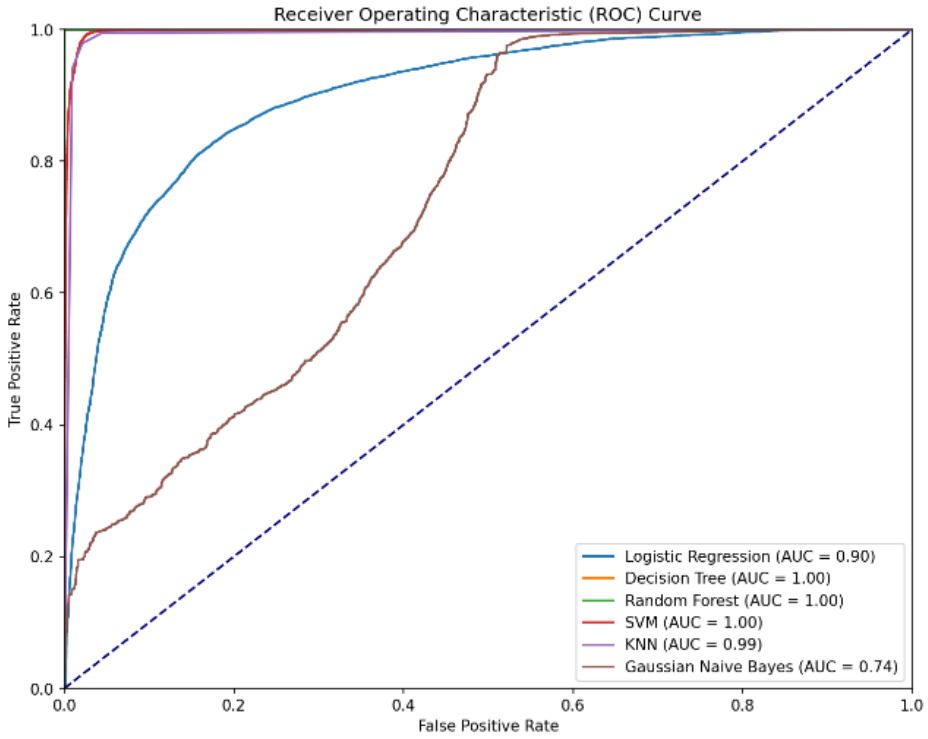
Bernoulli Naive Bayes modeli, %81.4 doğruluk oranı ile sınırlı bir performans sergilemiştir. Model, veri setinin karmaşıklığına etkin bir şekilde cevap veremediği görülmüştür. Precision (%82.1) ve recall (%80.2) değerleri de ortalama seviyededir, bu da modelin sınıflandırma yaparken gelişmeye ihtiyaç duyduğunu gösterir.

Bernoulli Naive Bayes modeli, düşük doğruluk ve F1 skoru ile sınırlı bir performans sergilemiştir, bu da modelin veri setinin karmaşıklığı ile başa çıkma konusundaki yetersizliğini ortaya koymaktadır. Modelin başarı seviyesinin düşük olmasının arkasında yatan sebepler arasında, veri setinin özelliklerini yeterince yakalayamamak, önemli ilişkileri göz ardı etmek veya modelin özelliklerini doğru bir şekilde anlayamamak gibi faktörler bulunabilir.

Bu zayıf performansı iyileştirmek adına, modelin daha iyi bir şekilde özelleştirilmesi, daha fazla öznitelik eklenmesi veya var olan özniteliklerin daha iyi ölçeklendirilmesi gibi stratejiler üzerine düşünülebilir. Ayrıca, modelin daha genel geçerliğini sağlamak için çeşitli hyperparametre ayarlamaları da ele alınabilir.

Algoritma Numarası	Algoritma Adı	Doğruluk	F1 Skoru	Hassasiyet	Geri Çağırma
0	Lojistik Regresyon	0,8263	0,7873	0,7542	0,8235
1	Karar Ağacı	0,9998	0,9998	0,9995	1,0000
2	Rastgele Orman	1,0000	1,0000	1,0000	1,0000
3	SVM	0,9810	0,9759	0,9653	0,9867
4	KNN	0,9789	0,9732	0,9675	0,9789
5	MLPClassifier	0,9959	0,9948	0,9963	0,9933
6	Bernoulli Naive Bayes	0,6349	0,6090	0,5234	0,7280

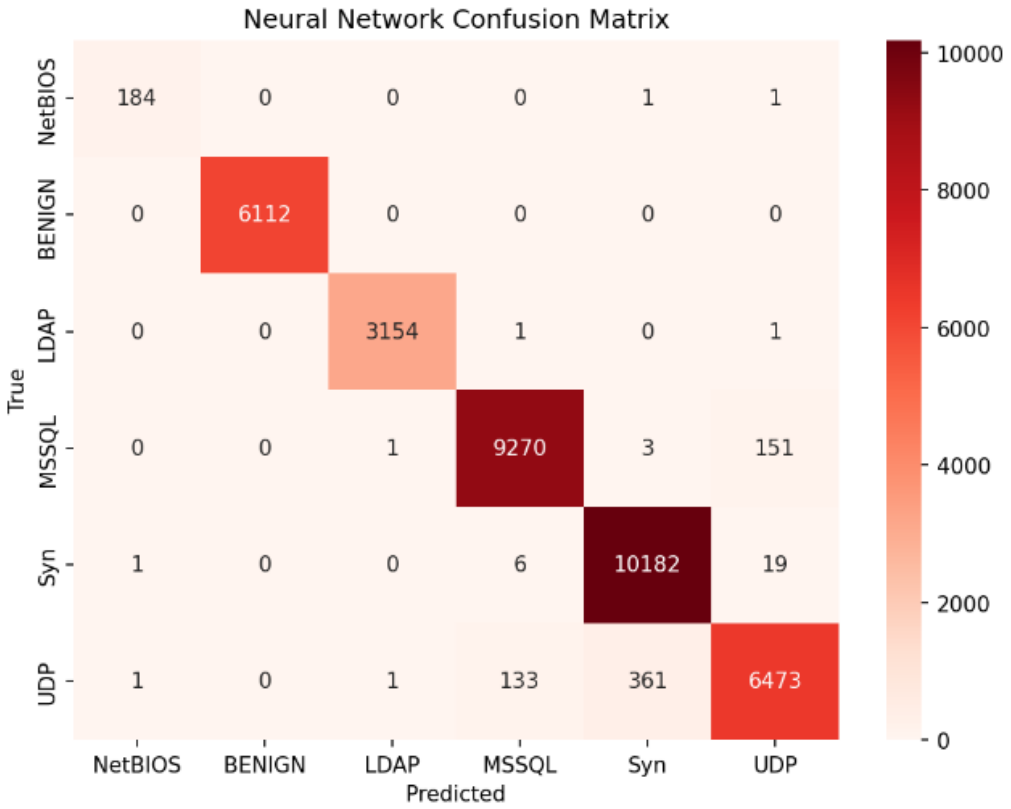
Şekil 22 Saldırı Tespit Sonuçları



Şekil 23 Saldırı tespit ROC Eğrisi

Saldırı Sınıflandırması Yapan Algoritmaların Performansı:

1. **Neural Network:**

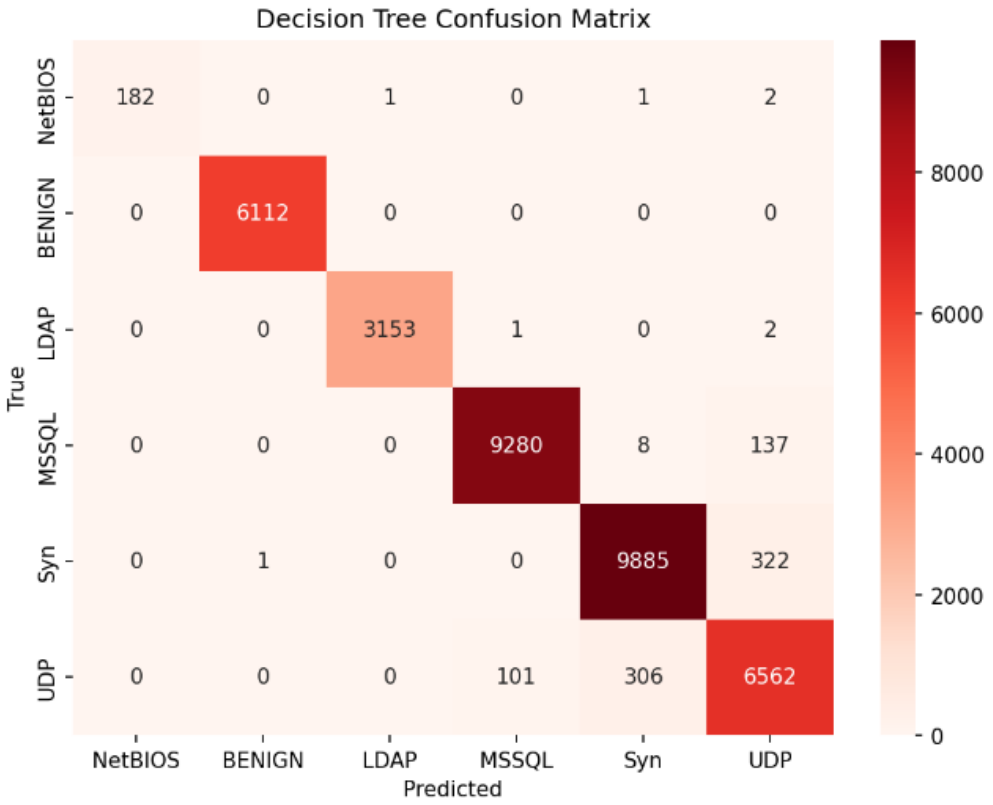


Şekil 24 Neural Network Confusion Matrix

- Accuracy: 98.11%
- F1 Score: 98.09%
- Precision: 98.12%
- Recall: 98.11%

Neural Network modeli, %98.11 doğruluk oranıyla oldukça etkileyici bir performans sergilemektedir. Bu, modelin giriş verilerini doğru bir şekilde sınıflandırmada başarılı olduğunu gösterir. Ayrıca, %98.09 F1 skoru, modelin hassasiyet ve geri çağrı yeteneklerinde denge sağladığını yansıtmaktadır. Precision (%98.12) ve recall (%98.11) değerleri de oldukça yüksektir, bu da modelin sınıflandırma yaparken hem yanlış pozitifleri hem de yanlış negatifleri düşük oranlarda tuttuğunu gösterir.

2. **Decision Tree:**

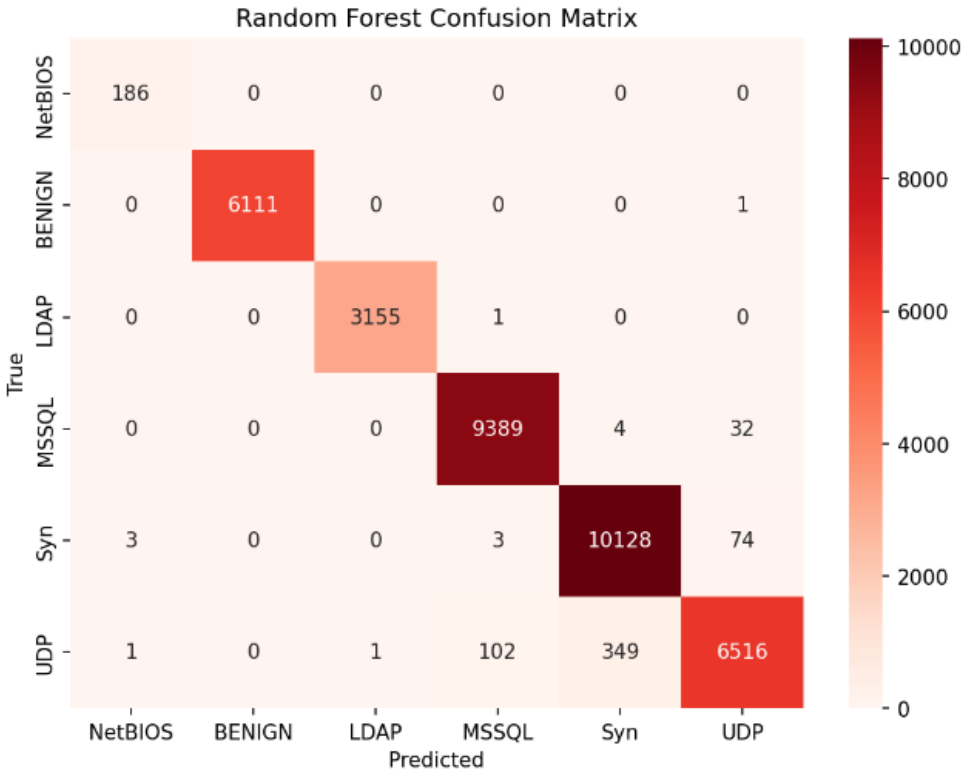


Şekil 25 Decision Tree Confusion Matrix

- Accuracy: 97.56%
- F1 Score: 97.57%
- Precision: 97.57%
- Recall: 97.56%

Decision Tree modeli %97.56 doğruluk oranı ile başarılı bir performans göstermektedir. Ayrıca, %97.57 F1 skoru, modelin hem hassasiyet hem de geri çağrıda dengeli bir performans sergilediğini göstermektedir. Precision (%97.57) ve recall (%97.56) değerleri de yüksektir, bu da modelin genel olarak verileri doğru bir şekilde sınıflandırdığını ve yanlış pozitif ile yanlış negatif oranlarını düşük tuttuğunu gösterir.

3. **Random Forest:**

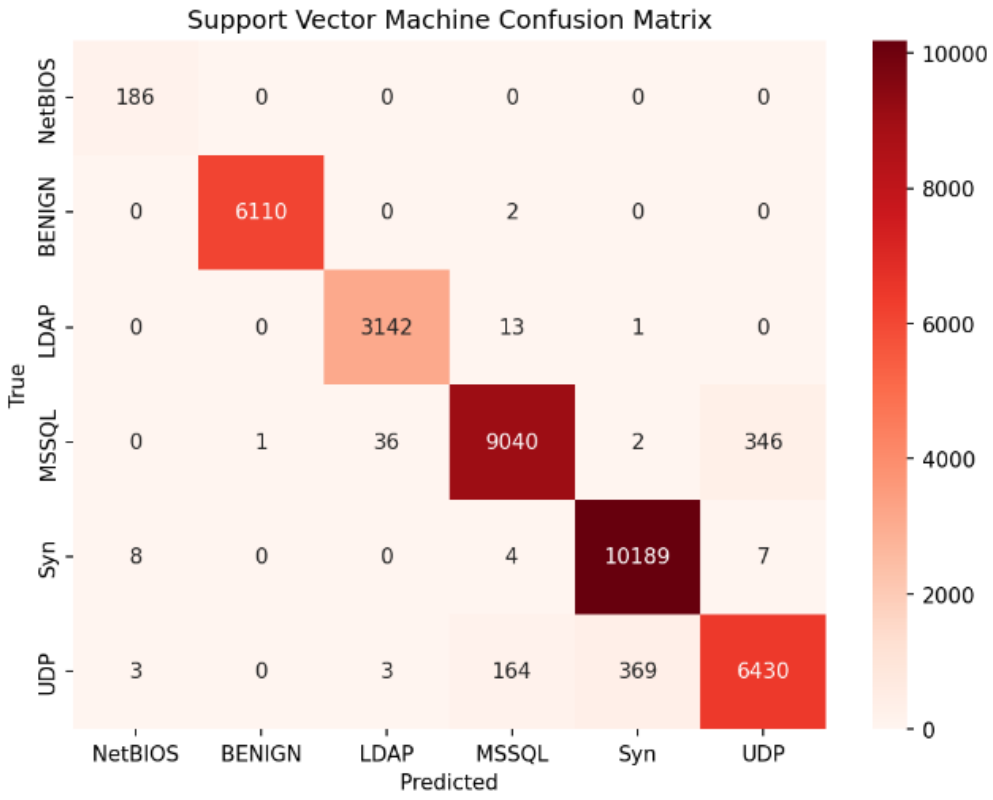


Şekil 26 Random Forest Confusion Matrix

- Accuracy: 98.41%
- F1 Score: 98.39%
- Precision: 98.42%
- Recall: 98.41%

Random Forest modeli %98.41 doğruluk oranı ve %98.39 F1 skoru ile oldukça etkileyici bir performans sergilemektedir. Precision (%98.42) ve recall (%98.41) değerleri de yüksektir, bu da modelin verileri doğru bir şekilde sınıflandırmada başarılı olduğunu ve dengeli bir performans sergilediğini gösterir. Random Forest, birden fazla decision tree kullanarak genel modelin gücünü artıran bir ensemble yöntemidir.

4. **Support Vector Machine:**



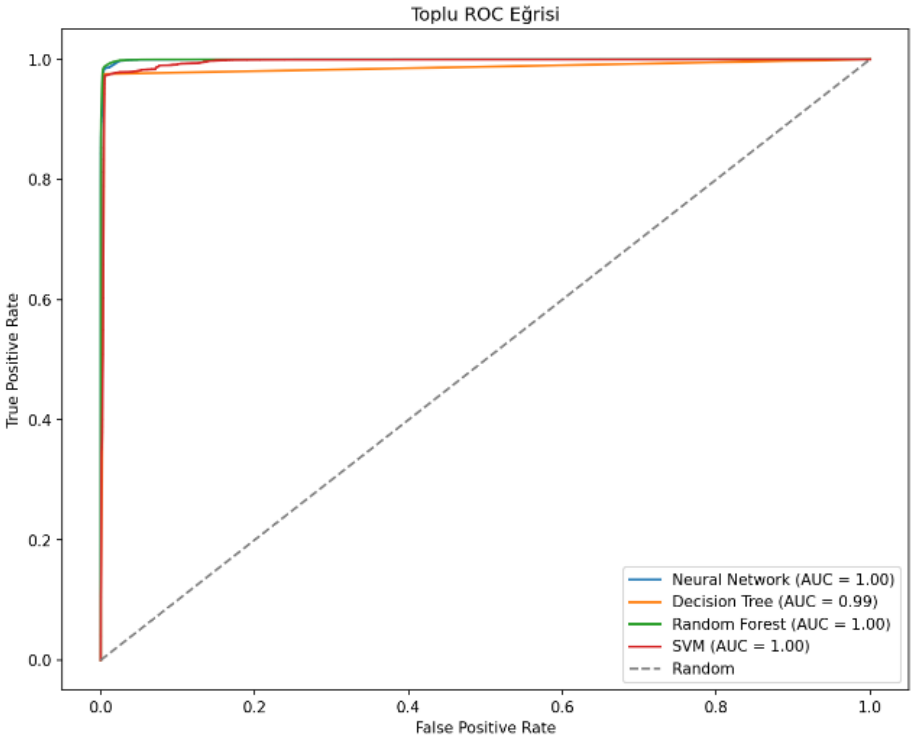
Şekil 27 SVM Confusion Matrix

- Accuracy: 97.28%
- F1 Score: 97.26%
- Precision: 97.28%
- Recall: 97.28%

SVM modeli %97.28 doğruluk oranı ile başarılı bir performans sergilemektedir. Fakat, %97.26 F1 skoru, modelin hassasiyet ve geri çağrıda bir miktar dengeleme yapabildiğini ancak bunun daha iyileştirilebileceğini göstermektedir. Precision (%97.28) ve recall (%97.28) değerleri birbirine yakın olup, modelin yanlış pozitif ve yanlış negatif oranlarını dengelediğini gösterir. SVM, lineer olmayan veri setlerinde de etkili olabilen bir sınıflandırma modelidir.

Id	Algoritma	Accuracy	F1 Score	Precision	Recall
0	Neural Network	0.9811	0.9809	0.9812	0.9811
1	Decision Tree	0.9756	0.9757	0.9757	0.9756
2	Random Forest	0.9841	0.9839	0.9842	0.9841
3	Support Vector Machine	0.9728	0.9726	0.9728	0.9728

Şekil 28 Saldırı Sınıflandırma Sonuçları



Şekil 29 Saldırı Sınıflandırma ROC Eğrisi

4 Sonuç

Ağ sistemlerinin güvenliği, yapılandırılmış ağ ve kullanıcılar tarafından oluşturulabilecek potansiyel açıklar ve zafiyetlere karşı alınan tedbirlerle sağlanır. Ağ ortamlarındaki güvenlik açıkları kapatılsa bile, farklı yöntemlerle gerçekleştirilen saldırılar, sistem güvenliğine zarar verebilir. Günümüzde, olası zararları minimize etmek amacıyla geliştirilen erken uyarı ve önleme sistemleri sürekli olarak güncellenmektedir.

Bu çalışma, özellikle DDoS saldırılarını tahmin etme ve sınıflandırma amacı taşıyan modeller üzerine odaklanarak gerçekleştirilmiştir. Geliştirilen modeller, saldırıları izleme, tespit etme ve önleme amacıyla daha az insan müdahalesi gerektiren yazılımlara dönüştürülebilme potansiyeli sunmuştur.

Modellerin performansı değerlendirildiğinde Rastgele Orman, Karar Ağacı, MLP, Destek Vektör Makinesi ve k-En Yakın Komşu algoritmaları yüksek doğruluk, F1 skoru ve hassasiyet değerleri ile öne çıkmaktadır. Ancak, Bernoulli Naive Bayes ve Lojistik Regresyon algoritmaları, özellikle düşük hassasiyet ve düşük F1 skoru ile daha zayıf sonuçlar vermiştir.

Saldırı sınıflandırma verilerinde ise kullanılan tüm algoritmalar, yüksek doğruluk, F1 skoru ve hassasiyet değerleri göstermişlerdir.

Sonuç olarak, makine öğrenme algoritmaları ile DDoS saldırıları tespit edilebilir ve sınıflandırılabilir.

5. Kaynakça

[1]: Loddos. DDoS Saldırıları Değerlendirme Raporu.

[2]: Electronics 2023, 12, 3103. https://doi.org/10.3390/electronics12143103

[3]:http://acikerisim.karabuk.edu.tr:8080/xmlui/handle/123456789/1621

[4]: Luckner, M., "Conversion of decision tree into deterministic finite automaton for high accuracy online SYN flood detection", Proceedings - 2015 IEEE Symposium Series On Computational Intelligence, SSCI 2015, (December 2015): 75–82 (2015).

[5]: Alkasassbeh, M., Al-Naymat, G., B.A, A., and Almseidin, M., "Detecting Distributed Denial of Service Attacks Using Data Mining Techniques", International Journal Of Advanced Computer Science And Applications, 7 (1): 436–445 (2016).

[6]: bilişim teknolojileri dergisi, cilt: 6, sayı: 3, eylül 2013, Deniz Mertkan GEZGİN Ercan BULUŞ

[7]: Kavita Choudhary, Meenakshi, Shilpa (ITM University, Gurgoan, Haryana, India) "Smurf Attack: Attacks using ICMP" IJCST Vol.2, Issue 1, March 2011 (ISSN:2229-4333)

[8]:"What Is a Smurf Attack? – HeelpBook", https://www.heelpbook.net/2014/what-is-a-smurf-attack/ (2021).

[9]: Alam, M. F., "Application Layer DDoS A Practical Approach & MitigationTechniques", Apricot 2014, 55 (2014).

[10]:https://www.kaggle.com/datasets/sizlingdhairya1/cicddos2019

[11]:https://www.kaggle.com/datasets/aikenkazin/ddos-sdn-dataset/

[12]: Tekin, Y., & Özer, E. (2021). Giriş Kontrol Listeleri (ACL'ler): Ağ Güvenliği İçin Etkili Bir Yöntem. Bilişim Teknolojileri Dergisi, 14(1), 1-16.

[13]: Li, X., Liu, W., & Wang, Y. (2017). A hybrid machine learning approach for intrusion detection based on KNN, BPNN and DT. Journal of Computer Science and Technology, 32(1), 147-155.

[14]: Lonea, I., Bădică, C., & Florea, A. (2013). A hybrid intrusion detection system for DDoS attacks in cloud computing environments. Procedia Computer Science, 19, 121-128.

[15]: Branitskiy, I., & Khoshgoftaar, T. M. (2017). Hybrid machine learning intrusion detection systems: A survey. IEEE Communications Surveys & Tutorials, 19(4), 2411-2433.

[16]: Aamir, M., & Zaidi, M. A. (2019). Intrusion detection using machine learning techniques: A review. Journal of Network and Computer Applications, 136, 102-112.
[17]: Deka, S., Chattopadhyay, S., & Chakraborty, S. (2019). Anomaly detection in network traffic for DDoS attack using active learning. IEEE Access, 7, 31004-31014.
[18]: Tertytchny, O., & Yakovlev, A. (2020). Machine learning-based approaches to failure and intrusion classification in network traffic. International Journal of Information Security, 19(1), 1-20.
[19]: Liu, Z., Wang, Y., & Li, Y. (2019). Learning network traffic feature representations with deep learning for intrusion detection. IEEE Access, 7, 12464-12474.
[20]: Volkov, A., & Afanasyev, D. (2020). Network traffic classification using long short-term memory neural networks. IEEE Access, 8, 14137-14146.
Muraleedharan, R., & Janet, P. (2021). HTTP DoS attack detection using machine learning techniques. Journal of Network and Computer Applications, 189, 103259.
[21]: Tekerek, S. (2021). Web attack detection using deep learning with CNN algorithm. Computers & Security, 110, 102279.
[22]: Li, X., Liu, W., & Wang, Y. (2017). A hybrid machine learning approach for intrusion detection based on KNN, BPNN and DT. Journal of Computer Science and Technology, 32(1), 147-155.
[23]: SaiSindhuTheja, V., Reddy, P. V., & Reddy, P. K. (2021). OCSA-RNN: A novel intrusion detection system for cloud computing using OCSA and RNN. Journal of Network and Computer Applications, 189, 103260.
[24]: Zubair, S., Alzahrani, A., & Khan, M. K. (2019). Anomaly detection for DDoS attacks in IoT using averaged dependence estimator. IEEE Access, 7, 21563-21572.
[25]: Silva, J. P., & Coury, J. (2020). Network traffic prediction and DDoS detection using NARX neural networks. IEEE Access, 8, 14058-14068.
[26]:Arivudainambi, V., Ramanathan, K., & Balasubramanian, V. (2015). A machine learning based traffic analysis system for malicious software detection. International Journal of Information Security, 14(5), 393-408.
[27]:Dimitrios, P., & Ioannis, M. (2020). A machine learning based intrusion detection system for mobile energy distribution systems. IEEE Access, 8, 13970-13980.
[28]:Muraleedharan, R., & Janet, P. (2021). HTTP DoS attack detection using machine learning techniques. Journal of Network and Computer Applications, 189, 103259.
[29]: S. Wu ve W. Banzhaf, «The Use of Computational Intelligence in Intrusion Detection Systems:A Review,» Applied Soft Computing, cilt 10, no. 1, pp. 1-35, 2010.

[30]: I. Witten ve E. Frank, Data Mining: Practical Machine Learning Tools and Techniques (Third Edition), Morgan Kaufmann Publication, 2011.
[31]: M. Dunham, Data Mining Introductory and Advanced Topics, Prentice Hall Pearson Education Inc, 2003.
[32]: Kaya, C., & Yıldız, O. (2014). Makine Öğrenme Teknikleriyle Saldırı Tespiti: Karşılaştırmalı Analiz. Marmara Fen Bilimleri Dergisi, 26(3), 89-104.

[33]:https://miro.medium.com/v2/resize:fit:1100/format:webp/1*i69vGs4AfhdhDUOlaPVLSA.png
[34]:https://helloacm.com/wp-content/uploads/2016/03/logistic-regression-example.jpg
[35]: Navlani 2018

[36]: İnce, C., İnce, K., & Hanbay, D. (2021, Mart 1). Saldırı Tespit Sistemlerinde Sınıflandırma Yöntemlerinin Kıyaslanması. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Dergisi, 14(1), 1-10.
[37]: Kaya, C., & Yıldız, O. (2014). Makine Öğrenme Teknikleriyle Saldırı Tespiti: Karşılaştırmalı Analiz. Marmara Fen Bilimleri Dergisi, 26(3), 89-104.
[38]: V. Vapnik, Statistical Learning Theory, New York: John Wiley, 1998.
[40]: İnce, C., İnce, K., & Hanbay, D. (2021, Mart 1). Saldırı Tespit Sistemlerinde Sınıflandırma Yöntemlerinin Kıyaslanması. Uluslararası Bilgisayar Bilimleri ve Mühendisliği Dergisi, 14(1), 1-10.
[41]:https://miro.medium.com/v2/resize:fit:640/format:webp/0*2_qzcm2gSe9l67aI.png
[42]:Yüce, H. (2022). MQTT trafiğinde DoS saldırılarının makine öğrenmesi ile sınıflandırılması ve modelin SHAP ile yorumlanması. Journal of Materials and Mechatronics: A, 3(1), 50-62. DOI: 10.55546/jmm.995091
[43]:https://pythongeeks.org/wp-content/uploads/2022/03/working-of-gradient-boosting-algorithm.webp
[44]:https://commons.wikimedia.org/wiki/File:Roc-draft-xkcd-style.svg

6. Ekler

6.1 EK 1: Saldırı Tespit Kodları

```python
# Pandas ve NumPy kütüphanelerini ekler
import pandas as pd
import numpy as np

# Grafik çizimleri için Matplotlib ve Seaborn kütüphanelerini ekler
import matplotlib.pyplot as plt
import seaborn as sns
import csv

# Veri setini eğitim ve test setlerine bölmek için scikit-learn'den ilgili fonksiyonu ekler
from sklearn.model_selection import train_test_split

# Veri standardizasyonu için scikit-learn'den ilgili fonksiyonu ekler
from sklearn.preprocessing import StandardScaler

# Kategorik sütunları sayısal forma çevirme
from sklearn.preprocessing import LabelEncoder

# Makine öğrenmesi algoritmaları için scikit-learn'den ilgili sınıfları ekler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import BernoulliNB
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

# Model performansını değerlendirmek için confusion matrix ve ROC curve ile ilgili fonksiyonları ekler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_curve, auc, confusion_matrix
```

```python

# .csv dosyasını okuyoruz
df = pd.read_csv("Dataset_sdn.csv")

# Her bir sütunun içerdiği eksik verileri çubuk grafiği ile çizdirir
def plotMissingValues(dataframe):
    missing_values = dataframe.isnull().sum()
    missing_values.plot(kind='bar',color='DarkRed',edgecolor='black')
    plt.xlabel("Özellikler")
    plt.ylabel("Eksik değerler")
    plt.title("Toplam eksik değer")
    plt.show()

plotMissingValues(df)

# Etiket (label) sütununda kaç tane 0 ve 1 olduğunu gösteren değerleri alın
label_counts = df['label'].value_counts()

# Etiket 0 ve 1 sayılarını ekrana yazdırın
print("Etiket 0 sayısı:", label_counts[0])
print("Etiket 1 sayısı:", label_counts[1])

# 0 ve 1 etiketlerinin sayısını eşitlemek için minimum etiket sayısını bulun
min_label_count = min(label_counts[0], label_counts[1])

# 0 ve 1 etiketlerinden eşit sayıda örnek alın
balanced_df = pd.concat([
    df[df['label'] == 0].sample(min_label_count, random_state=42),
    df[df['label'] == 1].sample(min_label_count, random_state=42)
])

```

```
# Yeni veri setinin etiket sayısını kontrol edin
balanced_label_counts = balanced_df['label'].value_counts()

# Her bir özelliğin histogram grafiğini oluşturur
plt.figure(figsize=(15, 10))
for col in balanced_df.columns:
    plt.hist(balanced_df[col],color='DarkBlue',edgecolor='black')
    plt.title(col)
    plt.show()

# Sütunlardaki boşlukları temizler ve sütun isimlerini düzenler
df.columns = df.columns.str.strip()

# Null değerleri içeren satırları siler
df = df.dropna()

# DataFrame üzerinde NaN (Not a Number) değerlerini kontrol eder
pd.set_option('use_inf_as_na', True)
null_values = df.isnull().sum()

# Kategorik sütunları one-hot encoding ile çevirin
df = pd.get_dummies(df, columns=['src', 'dst'])

#Protocol sütununa labelEncoder uygulanıyor
df['Protocol'] = LabelEncoder().fit_transform(df['Protocol'])

#label etiketini ayırıp X ve y'yi belirliyoruz
X = df.drop('label', axis=1)
y = df['label']

#Normalizasyon uygulanıyor
X_normalized = StandardScaler().fit_transform(X)

#train ve test verileri bölünüyor.
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.3, random_state=42)
```

```
# Confusion matrix çizdirme fonksiyonu
def plot_confusion_matrix(y_true, y_pred, classes, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.title(title)
    plt.xlabel('Tahmin')
    plt.ylabel('Gerçek')
    plt.show()

from imblearn.under_sampling import RandomUnderSampler
from sklearn.metrics import precision_recall_curve, average_precision_score

# RandomUnderSampler nesnesini oluşturun
rus = RandomUnderSampler(random_state=42)

# Veriyi azaltın
X_train, y_train = rus.fit_resample(X_train, y_train)

# Verileri ölçeklendirme
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
#---------------- DDoS Detection ----------------------------------------------------------------------------

# Logistic Regression modelini oluşturur ve eğitir
lr_model = LogisticRegression(random_state=42, max_iter=250)
lr_model.fit(X_train_scaled, y_train)
lr_pred = lr_model.predict(X_test_scaled)

# Logistic Regression performans metriklerini hesaplar
lr_accuracy = accuracy_score(y_test, lr_pred)
lr_f1 = f1_score(y_test, lr_pred)
lr_precision = precision_score(y_test, lr_pred)
lr_recall = recall_score(y_test, lr_pred)

# 5-katlı çapraz doğrulama
cv_scores = cross_val_score(lr_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("Çapraz Doğrulama Sonuçları:", cv_scores)
print("Ortalama Doğruluk:", cv_scores.mean())

# Sonuçları yazdırır
print('\nLogistic Regression Metrics:')
print(f'Accuracy: {lr_accuracy:.4f}')
print(f'F1 Score: {lr_f1:.4f}')
print(f'Precision: {lr_precision:.4f}')
print(f'Recall: {lr_recall:.4f}')

# Logistic Regression için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, lr_pred, ['Normal', 'DDoS'], 'Logistic Regression Confusion Matrix')

#-------------------------------------------------------------------------------------------------------------
```

```python
#-------------------------------------------------------------------------------------------------------------

# Decision Tree modelini oluşturur ve eğitir
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train)
dt_pred = dt_model.predict(X_test_scaled)

# Decision Tree performans metriklerini hesaplar
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_f1 = f1_score(y_test, dt_pred)
dt_precision = precision_score(y_test, dt_pred)
dt_recall = recall_score(y_test, dt_pred)

# 5-katlı çapraz doğrulama
cv_scores = cross_val_score(dt_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("Çapraz Doğrulama Sonuçları:", cv_scores)
print("Ortalama Doğruluk:", cv_scores.mean())

# Sonuçları yazdırır
print('\nDecision Tree Metrics:')
print(f'Accuracy: {dt_accuracy:.4f}')
print(f'F1 Score: {dt_f1:.4f}')
print(f'Precision: {dt_precision:.4f}')
print(f'Recall: {dt_recall:.4f}')

# Decision Tree için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, dt_pred, ['Normal', 'DDoS'], 'Decision Tree Confusion Matrix')

#-------------------------------------------------------------------------------------------------------------
```

```python
#-----------------------------------------------------------------------------------------------

# Random Forest modelini oluşturur ve eğitir
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train_scaled, y_train)
rf_pred = rf_model.predict(X_test_scaled)

# Random Forest performans metriklerini hesaplar
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred)
rf_precision = precision_score(y_test, rf_pred)
rf_recall = recall_score(y_test, rf_pred)

# 5-katlı çapraz doğrulama
cv_scores = cross_val_score(rf_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("Çapraz Doğrulama Sonuçları:", cv_scores)
print("Ortalama Doğruluk:", cv_scores.mean())

# Sonuçları yazdırır
print('\nRandom Forest Metrics:')
print(f'Accuracy: {rf_accuracy:.4f}')
print(f'F1 Score: {rf_f1:.4f}')
print(f'Precision: {rf_precision:.4f}')
print(f'Recall: {rf_recall:.4f}')

# Random Forest için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, rf_pred, ['Normal', 'DDoS'], 'Random Forest Confusion Matrix')

# Random Forest
feature_importances = rf_model.feature_importances_
feature_names = X.columns
features_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})
features_df = features_df.sort_values(by='Importance', ascending=False)

plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=features_df)
plt.title('Random Forest Feature Importance')
plt.show()

#-----------------------------------------------------------------------------------------------
```

```python
#-----------------------------------------------------------------------------------------------

# SVM modelini oluşturur ve eğitir
svm_model = SVC(random_state=42)
svm_model.fit(X_train_scaled, y_train)
svm_pred = svm_model.predict(X_test_scaled)

# SVM performans metriklerini hesaplar
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred)
svm_precision = precision_score(y_test, svm_pred)
svm_recall = recall_score(y_test, svm_pred)

# 5-katlı çapraz doğrulama
cv_scores = cross_val_score(svm_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("Çapraz Doğrulama Sonuçları:", cv_scores)
print("Ortalama Doğruluk:", cv_scores.mean())

# Sonuçları yazdırır
print('\nSVM Metrics:')
print(f'Accuracy: {svm_accuracy:.4f}')
print(f'F1 Score: {svm_f1:.4f}')
print(f'Precision: {svm_precision:.4f}')
print(f'Recall: {svm_recall:.4f}')

# SVM için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, svm_pred, ['Normal', 'DDoS'], 'SVM Confusion Matrix')

#-----------------------------------------------------------------------------------------------
```

```python
#-------------------------------------------------------------------------------------------------

# KNN modelini tanımla
knn_model = KNeighborsClassifier()

# Denenecek k değerleri
param_grid = {'n_neighbors': [2, 3, 4, 5, 6, 7, 8, 9, 10]}

# GridSearchCV kullanarak en iyi k değerini bulma
grid_search = GridSearchCV(knn_model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train)

# En iyi k değerini yazdırma
best_k_value = grid_search.best_params_['n_neighbors']
print(f"En iyi k değeri: {best_k_value}")

# En iyi k değeri ile modeli tekrar eğitme
best_knn_model = KNeighborsClassifier(n_neighbors=best_k_value)
best_knn_model.fit(X_train_scaled, y_train)  # En iyi k değeri ile modeli eğit

# Test seti üzerinde performansı değerlendirme
best_knn_pred = best_knn_model.predict(X_test_scaled)
best_knn_accuracy = accuracy_score(y_test, best_knn_pred)
best_knn_f1 = f1_score(y_test, best_knn_pred)
best_knn_precision = precision_score(y_test, best_knn_pred)
best_knn_recall = recall_score(y_test, best_knn_pred)

# En iyi k değeri ile modelin performansını yazdırma
print('\nEn iyi K-Nearest Neighbors Metrics:')
print(f'Accuracy: {best_knn_accuracy:.4f}')
print(f'F1 Score: {best_knn_f1:.4f}')
print(f'Precision: {best_knn_precision:.4f}')
print(f'Recall: {best_knn_recall:.4f}')

# Confusion Matrix çizdirme
plot_confusion_matrix(y_test, best_knn_pred, ['Normal', 'DDoS'], 'En iyi KNN Confusion Matrix')

#-------------------------------------------------------------------------------------------------
```

```python
#-------------------------------------------------------------------------------------------------

# MLPClassifier modelini oluşturur ve eğitir
mlp_model = MLPClassifier(random_state=42, max_iter=300)
mlp_model.fit(X_train_scaled, y_train)

mlp_pred = mlp_model.predict(X_test_scaled)

# MLPClassifier performans metriklerini hesaplar
mlp_accuracy = accuracy_score(y_test, mlp_pred)
mlp_f1 = f1_score(y_test, mlp_pred)
mlp_precision = precision_score(y_test, mlp_pred)
mlp_recall = recall_score(y_test, mlp_pred)

# 5-katlı çapraz doğrulama
cv_scores_mlp = cross_val_score(mlp_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("MLPClassifier Çapraz Doğrulama Sonuçları:", cv_scores_mlp)
print("Ortalama Doğruluk:", cv_scores_mlp.mean())

# Sonuçları yazdırır
print('\nMLPClassifier Metrics:')
print(f'Accuracy: {mlp_accuracy:.4f}')
print(f'F1 Score: {mlp_f1:.4f}')
print(f'Precision: {mlp_precision:.4f}')
print(f'Recall: {mlp_recall:.4f}')

# MLPClassifier için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, mlp_pred, ['Normal', 'DDoS'], 'MLPClassifier Confusion Matrix')

#-------------------------------------------------------------------------------------------------
```

```python
#-----------------------------------------------------------------------------------------------

# Bernoulli Naive Bayes modelini oluşturur ve eğitir
bernoulli_nb_model = BernoulliNB()
bernoulli_nb_model.fit(X_train_scaled, y_train)
bernoulli_nb_pred = bernoulli_nb_model.predict(X_test_scaled)

# Bernoulli Naive Bayes performans metriklerini hesaplar
bernoulli_nb_accuracy = accuracy_score(y_test, bernoulli_nb_pred)
bernoulli_nb_f1 = f1_score(y_test, bernoulli_nb_pred)
bernoulli_nb_precision = precision_score(y_test, bernoulli_nb_pred)
bernoulli_nb_recall = recall_score(y_test, bernoulli_nb_pred)

# 5-katlı çapraz doğrulama
cv_scores_bernoulli_nb = cross_val_score(bernoulli_nb_model, X_train_scaled, y_train, cv=5)

# Çapraz doğrulama sonuçlarını yazdırma
print("Bernoulli Naive Bayes Çapraz Doğrulama Sonuçları:", cv_scores_bernoulli_nb)
print("Ortalama Doğruluk:", cv_scores_bernoulli_nb.mean())

# Sonuçları yazdırır
print('\nBernoulli Naive Bayes Metrics:')
print(f'Accuracy: {bernoulli_nb_accuracy:.4f}')
print(f'F1 Score: {bernoulli_nb_f1:.4f}')
print(f'Precision: {bernoulli_nb_precision:.4f}')
print(f'Recall: {bernoulli_nb_recall:.4f}')

# Bernoulli Naive Bayes için Confusion Matrix çizdirme
plot_confusion_matrix(y_test, bernoulli_nb_pred, ['Normal', 'DDoS'], 'Bernoulli Naive Bayes Confusion Matrix')

#--------------- Roc Eğrisi -----------------------------------------------------------------------
```

```python
#--------------- Roc Eğrisi -----------------------------------------------------------------------

# Logistic Regression ROC Curve
lr_prob = lr_model.predict_proba(X_test_scaled)[:, 1]
fpr_lr, tpr_lr, _ = roc_curve(y_test, lr_prob)
roc_auc_lr = auc(fpr_lr, tpr_lr)

# Decision Tree ROC Curve
dt_prob = dt_model.predict_proba(X_test_scaled)[:, 1]
fpr_dt, tpr_dt, _ = roc_curve(y_test, dt_prob)
roc_auc_dt = auc(fpr_dt, tpr_dt)

# Random Forest ROC Curve
rf_prob = rf_model.predict_proba(X_test_scaled)[:, 1]
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_prob)
roc_auc_rf = auc(fpr_rf, tpr_rf)

# SVM ROC Curve
svm_prob = svm_model.decision_function(X_test_scaled)
fpr_svm, tpr_svm, _ = roc_curve(y_test, svm_prob)
roc_auc_svm = auc(fpr_svm, tpr_svm)

# KNN modelinin tahmin olasılıklarını al
knn_prob = best_knn_model.predict_proba(X_test_scaled)[:, 1]
fpr_knn, tpr_knn, _ = roc_curve(y_test, knn_prob)
roc_auc_knn = auc(fpr_knn, tpr_knn)

# Berhoulli Naive Bayes ROC Curve
nb_prob = bernoulli_nb_model.predict_proba(X_test_scaled)[:, 1]
fpr_nb, tpr_nb, _ = roc_curve(y_test, nb_prob)
roc_auc_nb = auc(fpr_nb, tpr_nb)

# ROC Curve çizimi
plt.figure(figsize=(10, 8))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {roc_auc_lr:.2f})')
plt.plot(fpr_dt, tpr_dt, label=f'Decision Tree (AUC = {roc_auc_dt:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {roc_auc_rf:.2f})')
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {roc_auc_svm:.2f})')
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {roc_auc_knn:.2f})')
plt.plot(fpr_nb, tpr_nb, label=f'Gaussian Naive Bayes (AUC = {roc_auc_nb:.2f})')

plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()

#-----------------------------------------------------------------------------------------------
```

```python
#-------------------------------------------------------------------------------------------------

# Logistic Regression
lr_precision, lr_recall, _ = precision_recall_curve(y_test, lr_prob)
lr_avg_precision = average_precision_score(y_test, lr_prob)

# Decision Tree
dt_precision, dt_recall, _ = precision_recall_curve(y_test, dt_prob)
dt_avg_precision = average_precision_score(y_test, dt_prob)

# Random Forest
rf_precision, rf_recall, _ = precision_recall_curve(y_test, rf_prob)
rf_avg_precision = average_precision_score(y_test, rf_prob)

# SVM
svm_precision, svm_recall, _ = precision_recall_curve(y_test, svm_prob)
svm_avg_precision = average_precision_score(y_test, svm_prob)

# KNN
knn_precision, knn_recall, _ = precision_recall_curve(y_test, knn_prob)
knn_avg_precision = average_precision_score(y_test, knn_prob)

# Gaussian Naive Bayes
nb_precision, nb_recall, _ = precision_recall_curve(y_test, nb_prob)
nb_avg_precision = average_precision_score(y_test, nb_prob)

# Precision-Recall Curve çizimi
plt.figure(figsize=(10, 8))

# Logistic Regression
plt.plot(lr_recall, lr_precision, color='b', label=f'Logistic Regression (AP = {lr_avg_precision:.2f})')

# Decision Tree
plt.plot(dt_recall, dt_precision, color='g', label=f'Decision Tree (AP = {dt_avg_precision:.2f})')

# Random Forest
plt.plot(rf_recall, rf_precision, color='r', label=f'Random Forest (AP = {rf_avg_precision:.2f})')

# SVM
plt.plot(svm_recall, svm_precision, color='c', label=f'SVM (AP = {svm_avg_precision:.2f})')

# KNN
plt.plot(knn_recall, knn_precision, color='m', label=f'KNN (AP = {knn_avg_precision:.2f})')

# Gaussian Naive Bayes
plt.plot(nb_recall, nb_precision, color='y', label=f'Gaussian Naive Bayes (AP = {nb_avg_precision:.2f})')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc='upper right')
plt.show()

#-------------------------------------------------------------------------------------------------
```

6.2 EK 2: Saldırı Sınıflandırma Kodları

```python
# Pandas ve NumPy kütüphanelerini ekler
import pandas as pd
import numpy as np

# Grafik çizimleri için Matplotlib ve Seaborn kütüphanelerini ekler
import matplotlib.pyplot as plt
import seaborn as sns
import csv

# Veri setini eğitim ve test setlerine bölmek için scikit-learn'den ilgili fonksiyonu ekler
from sklearn.model_selection import train_test_split

# Veri standardizasyonu için scikit-learn'den ilgili fonksiyonu ekler
from sklearn.preprocessing import StandardScaler

# Kategorik sütunları sayısal forma çevirme
from sklearn.preprocessing import LabelEncoder

# Makine öğrenmesi algoritmaları için scikit-learn'den ilgili sınıfları ekler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import plot_tree

# Model performansını değerlendirmek için confusion matrix ve ROC curve ile ilgili fonksiyonları ekler
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, roc_curve, auc, confusion_matrix
from sklearn.preprocessing import label_binarize
from itertools import cycle

# .csv dosyasını okuyoruz
df = pd.read_csv("Classification.csv")

# Sütunlardaki boşlukları temizler ve sütun isimlerini düzenler
df.columns = df.columns.str.strip()

# 'Label' sütunundaki benzersiz değerleri görüntüler
unique_labels = df['Label'].unique()

# Her bir sütunun içerdiği eksik verileri çubuk grafiği ile çizdirir
def plotMissingValues(dataframe):
    missing_values = dataframe.isnull().sum()
    fig = plt.figure(figsize=(16, 5))
    missing_values.plot(kind='bar',color='DarkRed',edgecolor='black')
    plt.xlabel("Özellikler")
    plt.ylabel("Eksik değerler")
    plt.title("Toplam eksik değer")
    plt.show()

plotMissingValues(df)
```

```
plotMissingValues(df)

# Null değerleri içeren satırları siler
df = df.dropna()

# Null değerlerin tekrar kontrolünü yapar
plt.figure(figsize=(10, 4))
plt.hist(df.isna().sum(), color='darkred', edgecolor='black')
plt.title('Null değerler silindikten sonra dataset')
plt.xlabel('Null değer sayısı')
plt.ylabel('Kolon sayısı')
plt.show()

# DataFrame üzerinde NaN (Not a Number) değerlerini kontrol eder
pd.set_option('use_inf_as_na', True)
null_values = df.isnull().sum()

# Veri tiplerini kontrol eder
(df.dtypes == 'object')

# UDP ve UDPLag sınıflarını birleştir
df['Label'] = df['Label'].replace({'UDP': 'UDP', 'UDP-lag': 'UDP'})

# 'DrDoS_' ön eki ile başlayan sınıfları 'DrDoS' olarak birleştir
df['Label'] = df['Label'].replace({'DrDoS_DNS': 'DrDoS', 'DrDoS_LDAP': 'DrDoS', 'DrDoS_MSSQL': 'DrDoS',
                                   'DrDoS_NetBIOS': 'DrDoS', 'DrDoS_NTP': 'DrDoS', 'DrDoS_SNMP': 'DrDoS',
                                   'DrDoS_SSDP': 'DrDoS', 'DrDoS_UDP': 'DrDoS'})

# 'Label' sütunundaki benzersiz değerleri ve sayılarını bulma
label_counts = df['Label'].value_counts()
```

```
# Benzersiz değerleri ve sayılarını ekrana yazdırma
print("Benzersiz Etiket Değerleri ve Sayıları:")
print(label_counts)

# Sınıf adları ve sayıları
classes = [
    'NetBIOS', 'BENIGN', 'LDAP', 'MSSQL', 'Portmap', 'Syn', 'UDP', 'UDPLag',
    'DrDoS', 'TFTP', 'WebDDoS'
]

# Her sınıfa farklı değer atama
class_mapping = {

    'BENIGN': 0,
    'NetBIOS': 1,
    'LDAP': 2,
    'MSSQL': 3,
    'Portmap': 4,
    'Syn': 5,
    'UDP': 6,
    'UDPLag': 7,
    'WebDDoS': 8,
    'TFTP': 9,
    'DrDoS': 10
}

# 'Label' verilerini sayısal hale getirir
df['Label'] = df['Label'].map(class_mapping)

# Çubuk grafik oluştur
plt.bar(classes, 19, color='DarkRed', align='center', edgecolor='black')
plt.xlabel("Sınıflar")
plt.ylabel("Miktar")
plt.title("Sınıf Dağılımı")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

# Veri setinin istatistiksel özetini gösterir
df.describe()

df.drop(['Unnamed: 0', 'Flow ID', 'Source IP', 'Destination IP', 'Timestamp', 'SimillarHTTP'], axis=1, inplace=True)

le = LabelEncoder()
df['Protocol'] = le.fit_transform(df['Protocol'])

categorical_cols = ['Protocol']
df = pd.get_dummies(df, columns=categorical_cols)
```

```python
# Sütunlardaki boşlukları temizler ve sütun isimlerini düzenler
df.columns = df.columns.str.strip()

# Null değerleri içeren satırları siler
df = df.dropna()

# Null değerlerin tekrar kontrolünü yapar
plt.figure(figsize=(10, 4))
plt.hist(df.isna().sum(), color='darkred', edgecolor='black')
plt.title('Null değerler silindikten sonra dataset')
plt.xlabel('Null değer sayısı')
plt.ylabel('Kolon sayısı')
plt.show()

# DataFrame üzerinde NaN (Not a Number) değerlerini kontrol eder
pd.set_option('use_inf_as_na', True)
null_values = df.isnull().sum()

# 'Label' sütunundaki satırları filtrele
df = df[df['Label'] != 4]
df = df[df['Label'] != 7]
df = df[df['Label'] != 8]
df = df[df['Label'] != 9]
df = df[df['Label'] != 10]

# Veri tiplerini kontrol eder
(df.dtypes == 'object')

X = df.drop('Label', axis=1)
y = df['Label']

df = df.astype(float)

# Her bir özelliğin histogram grafiğini oluşturur
plt.figure(figsize=(15, 10))
for col in df.columns:
    plt.hist(df[col],color='DarkRed',edgecolor='black')
    plt.title(col)
    plt.show()

scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)

# Eğitim ve test verilerini oluşturur
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y, test_size=0.40, random_state=42)

# Eğitim ve test verilerinin sayısını yazdırır
print("Train dataset size =", X_train.shape)
print("Test dataset size =", X_test.shape)

# Verileri ölçeklendirme
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```python
#-----------------------------------------------

# Özel confusion matrix çizimi
def plot_confusion_matrix_custom(y_true, y_pred, classes, title, cmap='Reds'):
    cm = confusion_matrix(y_true, y_pred)

    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt='g', cmap=cmap, xticklabels=classes, yticklabels=classes)
    plt.title(title)
    plt.xlabel('Predicted')
    plt.ylabel('True')
    plt.show()

#-----------------------------------------------
```

```
#-----------------------------------------------

# Neural Network modelini oluşturur ve eğitir
nn_model = MLPClassifier(hidden_layer_sizes=(25,), max_iter=1000, random_state=500)
nn_model.fit(X_train, y_train)
nn_pred = nn_model.predict(X_test)

# Neural Network performans metriklerini hesaplar
nn_accuracy = accuracy_score(y_test, nn_pred)
nn_f1 = f1_score(y_test, nn_pred, average='weighted')
nn_precision = precision_score(y_test, nn_pred, average='weighted', zero_division=1)
nn_recall = recall_score(y_test, nn_pred, average='weighted', zero_division=1)

# Sonuçları yazdırır
print('\nNeural Network Metrics:')
print(f'Accuracy: {nn_accuracy:.4f}')
print(f'F1 Score: {nn_f1:.4f}')
print(f'Precision: {nn_precision:.4f}')
print(f'Recall: {nn_recall:.4f}')

# Neural Network için özel confusion matrix çizdirme
plot_confusion_matrix_custom(y_test, nn_pred, ['NetBIOS', 'BENIGN', 'LDAP', 'MSSQL', 'Syn', 'UDP'], 'Neural Network Confusion Matrix', cmap='Reds')

#-----------------------------------------------
```

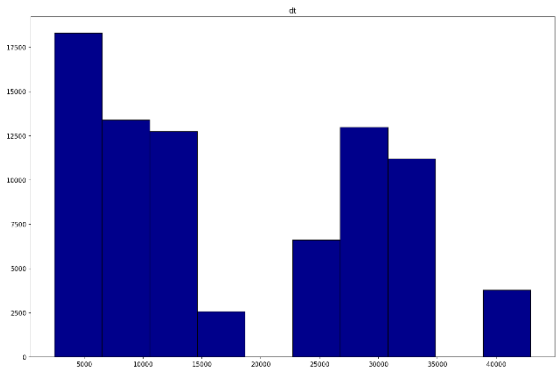
```
#-----------------------------------------------

# Decision Tree modelini oluşturur ve eğitir
dt_model = DecisionTreeClassifier(random_state=500)
dt_model.fit(X_train, y_train)
dt_pred = dt_model.predict(X_test)

# Decision Tree performans metriklerini hesaplar
dt_accuracy = accuracy_score(y_test, dt_pred)
dt_f1 = f1_score(y_test, dt_pred, average='weighted')
dt_precision = precision_score(y_test, dt_pred, average='weighted', zero_division=1)
dt_recall = recall_score(y_test, dt_pred, average='weighted', zero_division=1)

# Sonuçları yazdırır
print('\nDecision Tree Metrics:')
print(f'Accuracy: {dt_accuracy:.4f}')
print(f'F1 Score: {dt_f1:.4f}')
print(f'Precision: {dt_precision:.4f}')
print(f'Recall: {dt_recall:.4f}')

# Decision Tree için özel confusion matrix çizdirme
plot_confusion_matrix_custom(y_test, dt_pred, ['NetBIOS', 'BENIGN', 'LDAP', 'MSSQL', 'Syn', 'UDP'], 'Decision Tree Confusion Matrix', cmap='Reds')

#-----------------------------------------------
```

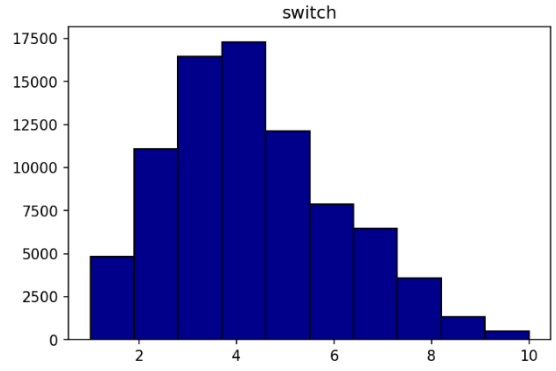
```
#-----------------------------------------------

# Random Forest modelini oluşturur ve eğitir
rf_model = RandomForestClassifier(random_state=500)
rf_model.fit(X_train, y_train)
rf_pred = rf_model.predict(X_test)

# Random Forest performans metriklerini hesaplar
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_f1 = f1_score(y_test, rf_pred, average='weighted')
rf_precision = precision_score(y_test, rf_pred, average='weighted', zero_division=1)
rf_recall = recall_score(y_test, rf_pred, average='weighted', zero_division=1)

# Sonuçları yazdırır
print('\nRandom Forest Metrics:')
print(f'Accuracy: {rf_accuracy:.4f}')
print(f'F1 Score: {rf_f1:.4f}')
print(f'Precision: {rf_precision:.4f}')
print(f'Recall: {rf_recall:.4f}')

# Random Forest için özel confusion matrix çizdirme
plot_confusion_matrix_custom(y_test, rf_pred, ['NetBIOS', 'BENIGN', 'LDAP', 'MSSQL', 'Syn', 'UDP'], 'Random Forest Confusion Matrix', cmap='Reds')

#-----------------------------------------------
```

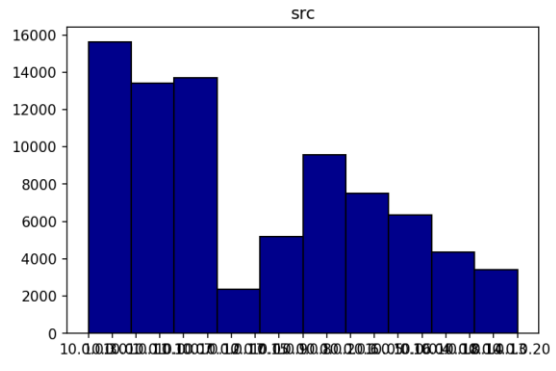
```
#-----------------------------------------------

# Support Vector Machine modelini oluşturur ve eğitir
svm_model = SVC(random_state=500)
svm_model.fit(X_train, y_train)
svm_pred = svm_model.predict(X_test)

# Support Vector Machine performans metriklerini hesaplar
svm_accuracy = accuracy_score(y_test, svm_pred)
svm_f1 = f1_score(y_test, svm_pred, average='weighted')
svm_precision = precision_score(y_test, svm_pred, average='weighted', zero_division=1)
svm_recall = recall_score(y_test, svm_pred, average='weighted', zero_division=1)

# Sonuçları yazdırır
print('\nSupport Vector Machine Metrics:')
print(f'Accuracy: {svm_accuracy:.4f}')
print(f'F1 Score: {svm_f1:.4f}')
print(f'Precision: {svm_precision:.4f}')
print(f'Recall: {svm_recall:.4f}')

# Support Vector Machine için özel confusion matrix çizdirme
plot_confusion_matrix_custom(y_test, svm_pred, ['NetBIOS', 'BENIGN', 'LDAP', 'MSSQL', 'Syn', 'UDP'], 'Support Vector Machine Confusion Matrix', cmap='Reds')

#-----------------------------------------------
```

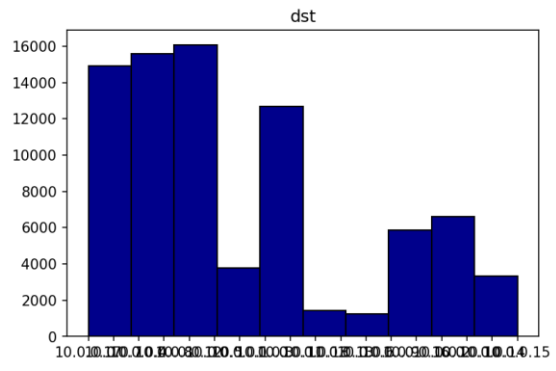
```
#-------------------------------------------------

# Her bir model için toplu ROC eğrisi çizimi
plt.figure(figsize=(10, 8))

# Neural Network ROC
nn_prob = nn_model.predict_proba(X_test)
nn_prob_bin = label_binarize(y_test, classes=[0, 1, 2, 3, 5, 6])
nn_fpr, nn_tpr, _ = roc_curve(nn_prob_bin.ravel(), nn_prob.ravel())
nn_auc = auc(nn_fpr, nn_tpr)
plt.plot(nn_fpr, nn_tpr, label=f'Neural Network (AUC = {nn_auc:.2f})')

# Decision Tree ROC
dt_prob = dt_model.predict_proba(X_test)
dt_prob_bin = label_binarize(y_test, classes=[0, 1, 2, 3, 5, 6])
dt_fpr, dt_tpr, _ = roc_curve(dt_prob_bin.ravel(), dt_prob.ravel())
dt_auc = auc(dt_fpr, dt_tpr)
plt.plot(dt_fpr, dt_tpr, label=f'Decision Tree (AUC = {dt_auc:.2f})')

# Random Forest ROC
rf_prob = rf_model.predict_proba(X_test)
rf_prob_bin = label_binarize(y_test, classes=[0, 1, 2, 3, 5, 6])
rf_fpr, rf_tpr, _ = roc_curve(rf_prob_bin.ravel(), rf_prob.ravel())
rf_auc = auc(rf_fpr, rf_tpr)
plt.plot(rf_fpr, rf_tpr, label=f'Random Forest (AUC = {rf_auc:.2f})')

# Support Vector Machine ROC
svm_prob = svm_model.decision_function(X_test)
svm_prob_bin = label_binarize(y_test, classes=[0, 1, 2, 3, 5, 6])
svm_fpr, svm_tpr, _ = roc_curve(svm_prob_bin.ravel(), svm_prob.ravel())
svm_auc = auc(svm_fpr, svm_tpr)
plt.plot(svm_fpr, svm_tpr, label=f'SVM (AUC = {svm_auc:.2f})')

# Diğer ayarlamalar
plt.plot([0, 1], [0, 1], linestyle='--', color='grey', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Toplu ROC Eğrisi')
plt.legend()
plt.show()
```

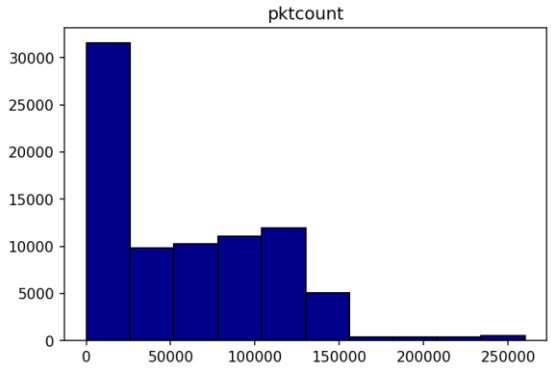
4.1 EK 3: Veri Tabloları(Saldırı Tespit)

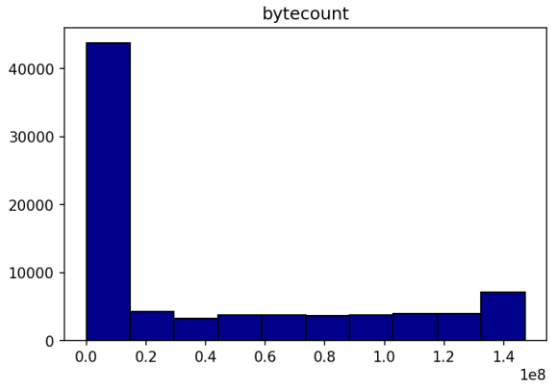


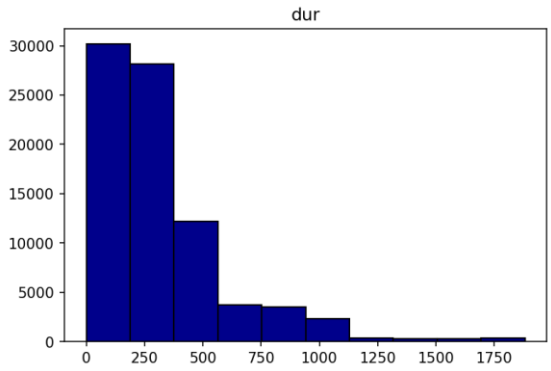


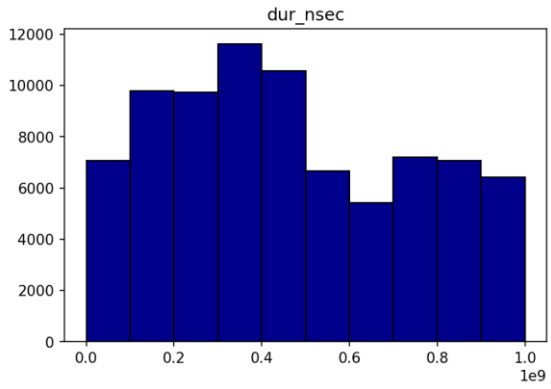


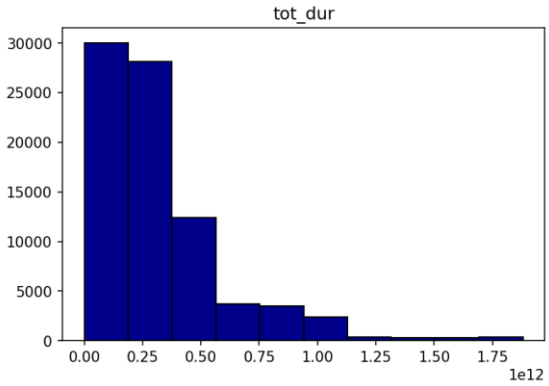


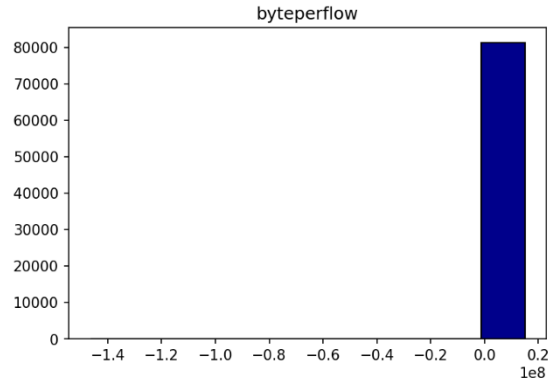


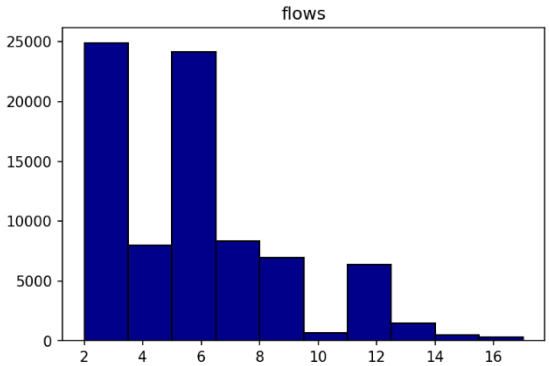


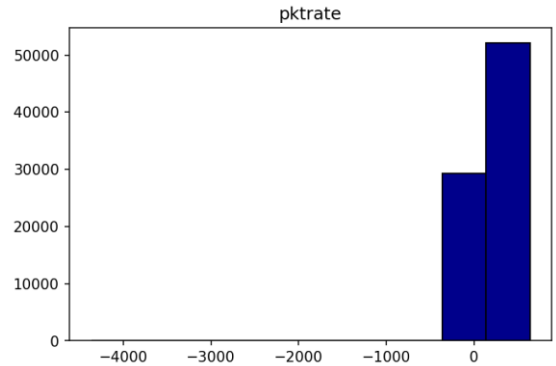


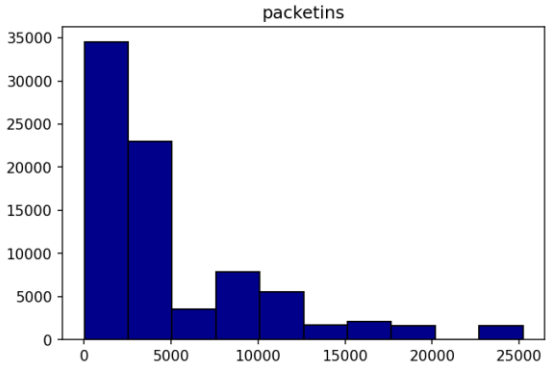


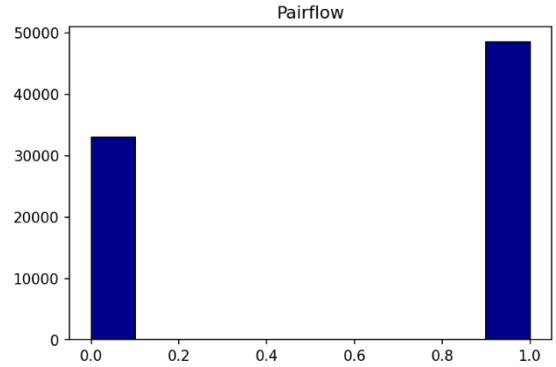

tot_dur
30000
25000
20000
15000
10000
5000
0
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e12

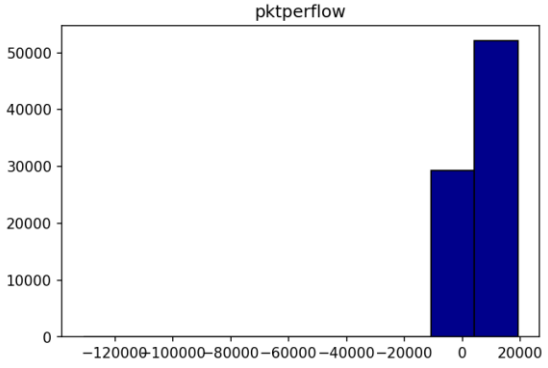

byteperflow
80000
70000
60000
50000
40000
30000
20000
10000
0
−1.4
−1.2
−1.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
1e8

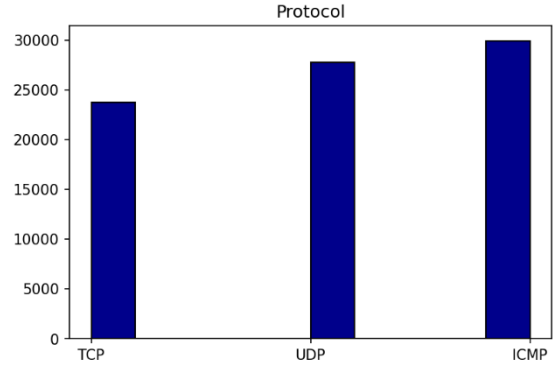

flows
25000
20000
15000
10000
5000
0
2
4
6
8
10
12
14
16

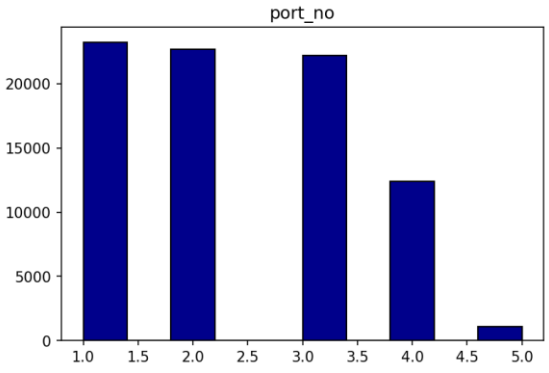

pktrate
50000
40000
30000
20000
10000
0
−4000
−3000
−2000
−1000
0

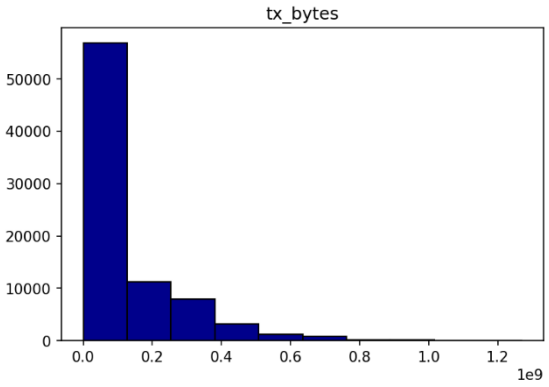

packetins
35000
30000
25000
20000
15000
10000
5000
0
0
5000
10000
15000
20000
25000

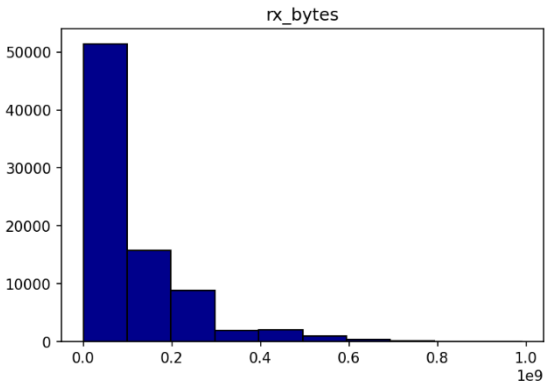

Pairflow
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0

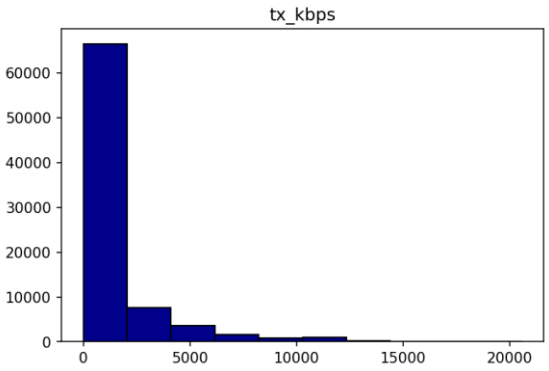

pktperflow
50000
40000
30000
20000
10000
0
−120000
−100000
−80000
−60000
−40000
−20000
0
20000

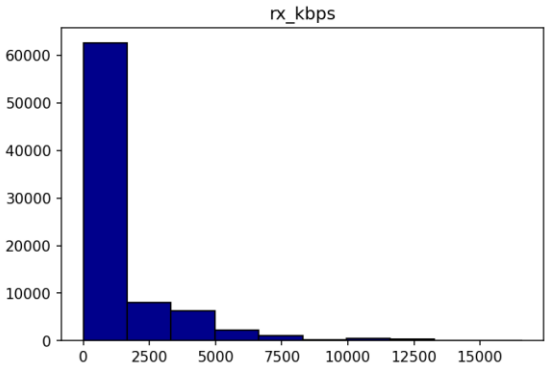

Protocol
30000
25000
20000
15000
10000
5000
0
TCP
UDP
ICMP

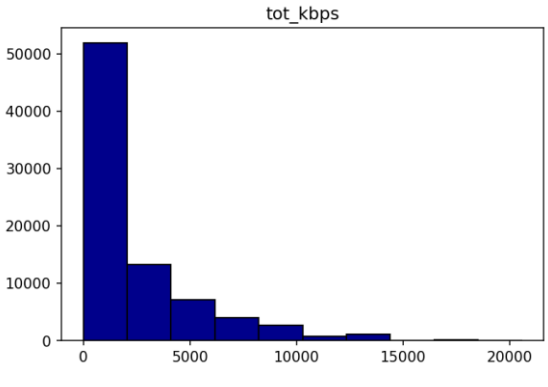

port_no
20000
15000
10000
5000
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0

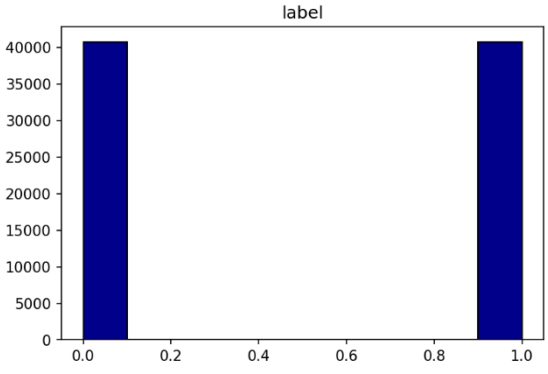

tx_bytes
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e9

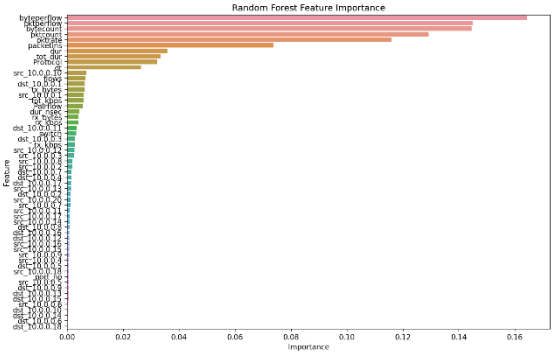

rx_bytes
50000
40000
30000
20000
10000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e9


tx_kbps
60000
50000
40000
30000
20000
10000
0
0
5000
10000
15000
20000

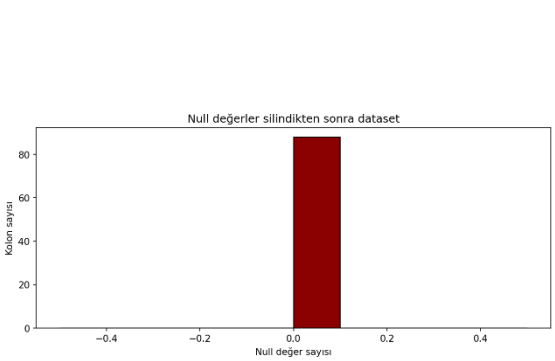

rx_kbps
60000
50000
40000
30000
20000
10000
0
0
2500
5000
7500
10000
12500
15000

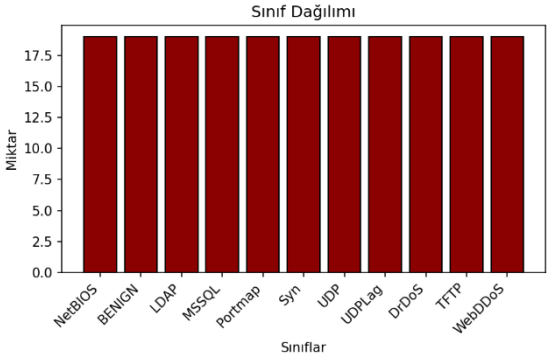

tot_kbps
50000
40000
30000
20000
10000
0
0
5000
10000
15000
20000

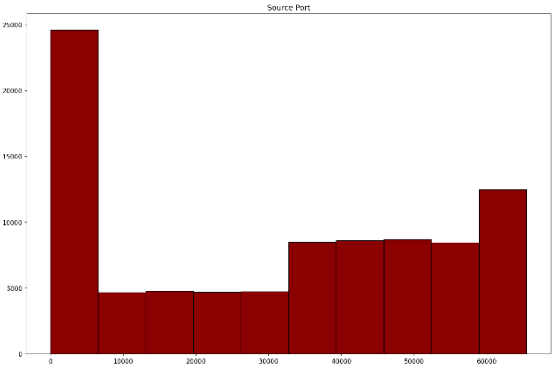

label
40000
35000
30000
25000
20000
15000
10000
5000
0
0.0
0.2
0.4
0.6
0.8
1.0

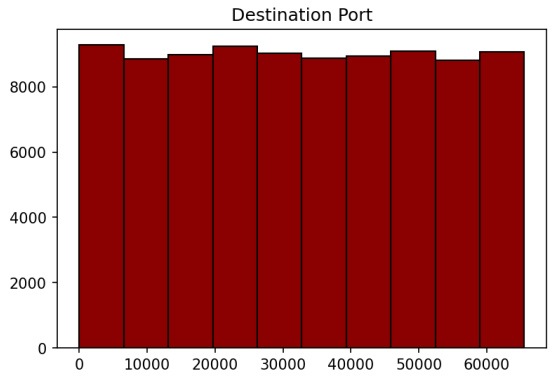

Random Forest Feature Importance
Feature
Importance
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0.16

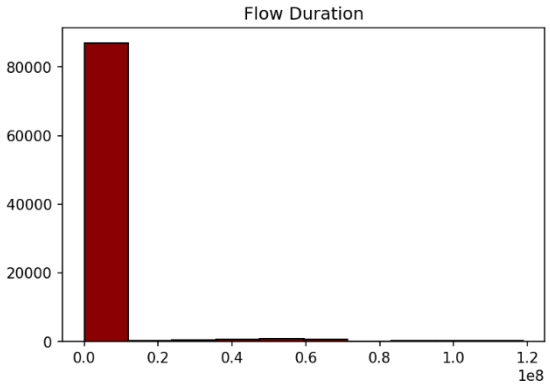
4.2 EK 3: Veri Tabloları(Saldırı Sınıflandırma)

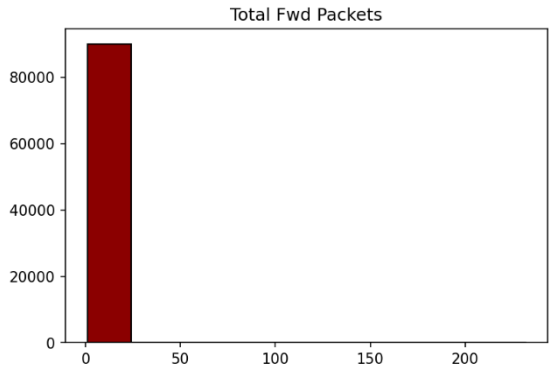


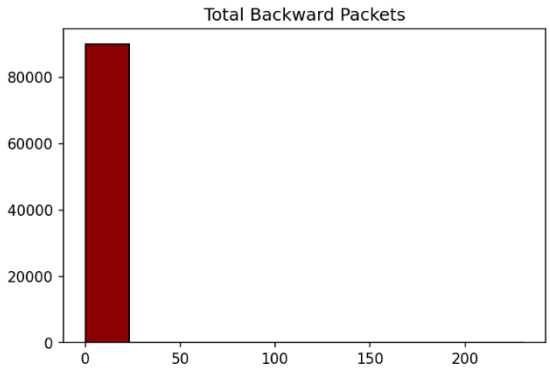


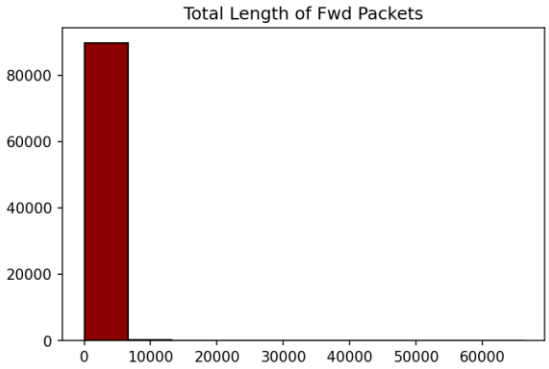


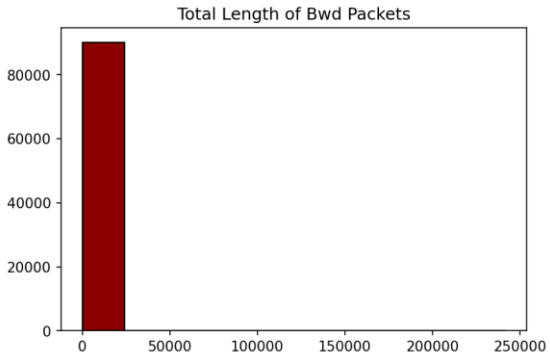


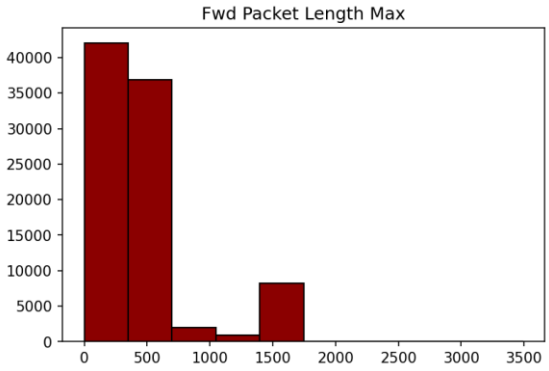


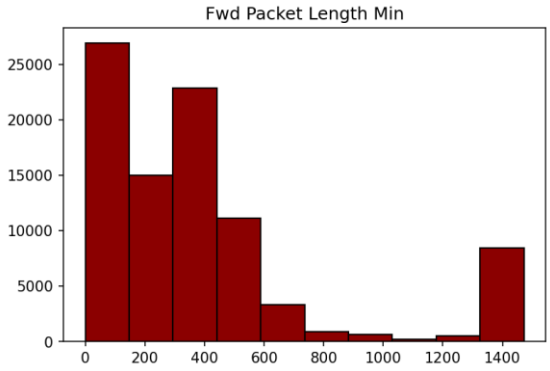


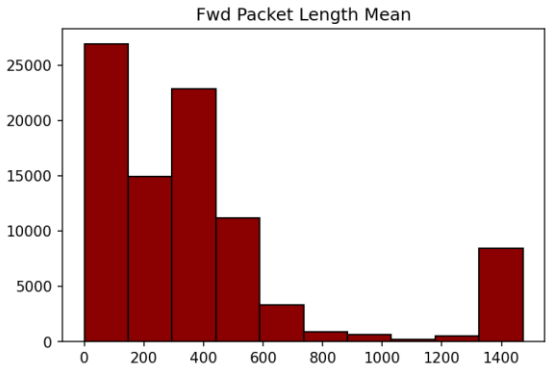


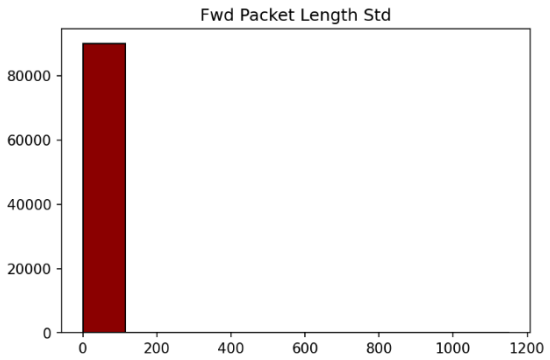

Total Length of Fwd Packets
0
20000
40000
60000
80000
0
10000
20000
30000
40000
50000
60000

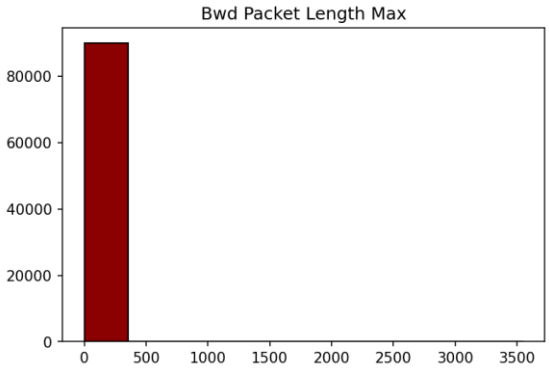

Total Length of Bwd Packets
0
20000
40000
60000
80000
0
50000
100000
150000
200000
250000

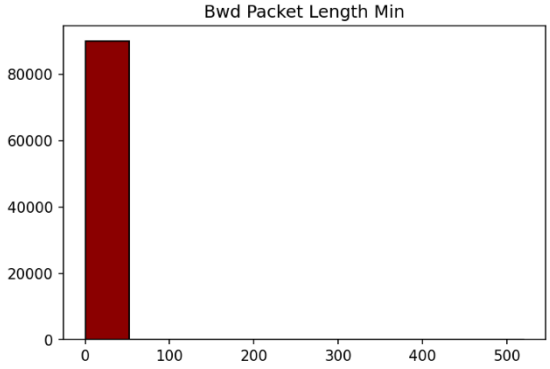

Fwd Packet Length Max
0
5000
10000
15000
20000
25000
30000
35000
40000
0
500
1000
1500
2000
2500
3000
3500

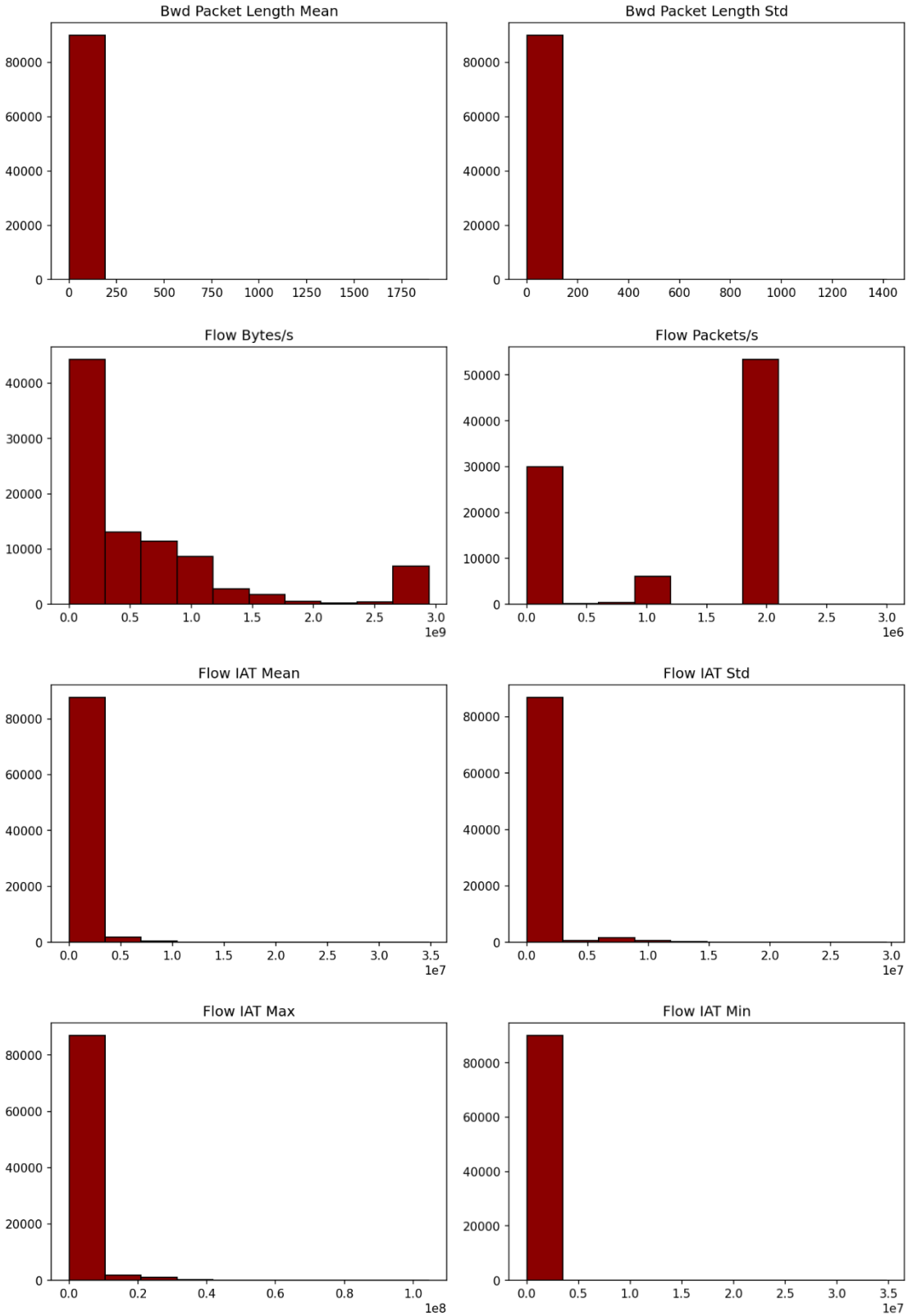

Fwd Packet Length Min
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400

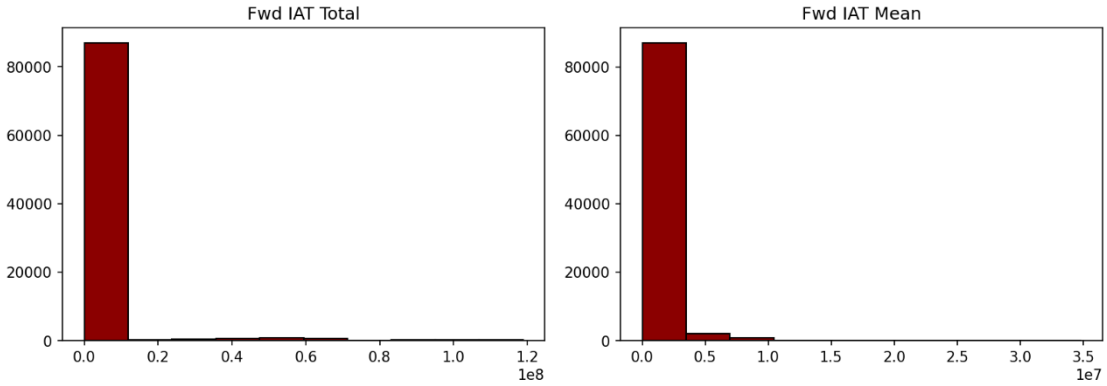

Fwd Packet Length Mean
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400


Fwd Packet Length Std
0
20000
40000
60000
80000
0
200
400
600
800
1000
1200

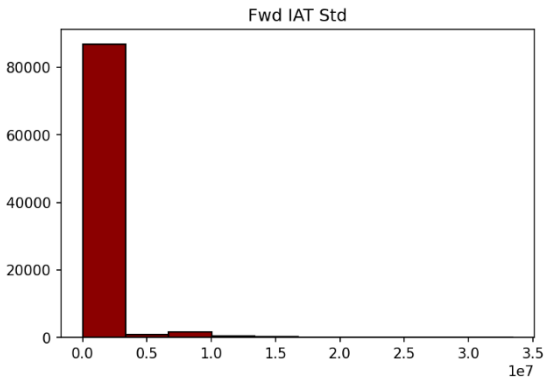

Bwd Packet Length Max
0
20000
40000
60000
80000
0
500
1000
1500
2000
2500
3000
3500

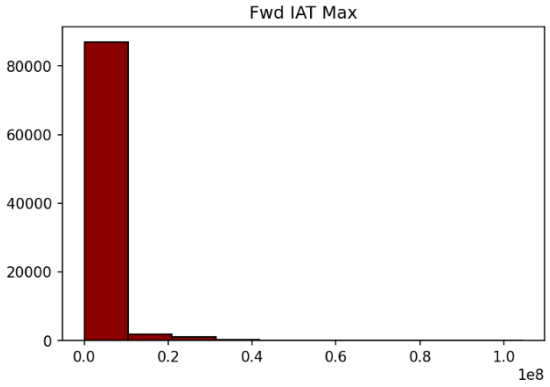

Bwd Packet Length Min
0
20000
40000
60000
80000
0
100
200
300
400
500

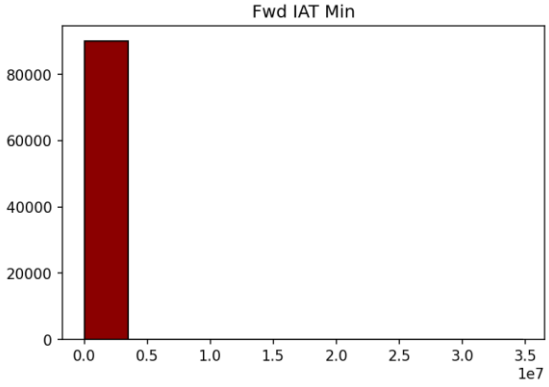

Bwd Packet Length Mean
Bwd Packet Length Std
Flow Bytes/s
1e9
Flow Packets/s
1e6
Flow IAT Mean
1e7
Flow IAT Std
1e7
Flow IAT Max
1e8
Flow IAT Min
1e7

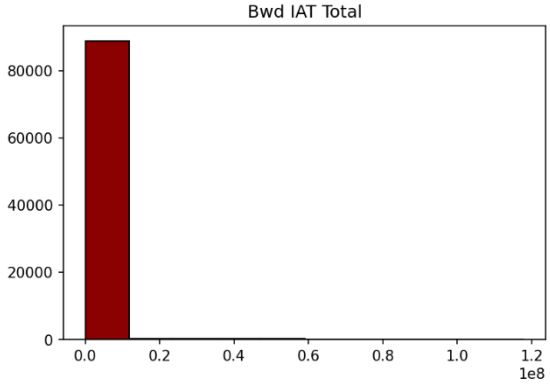

Fwd IAT Total
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e8

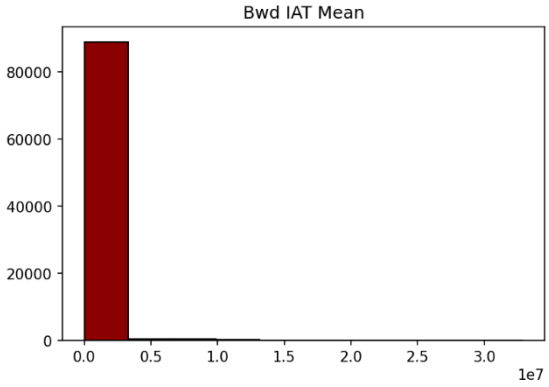
Fwd IAT Mean
80000
60000
40000
20000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e7

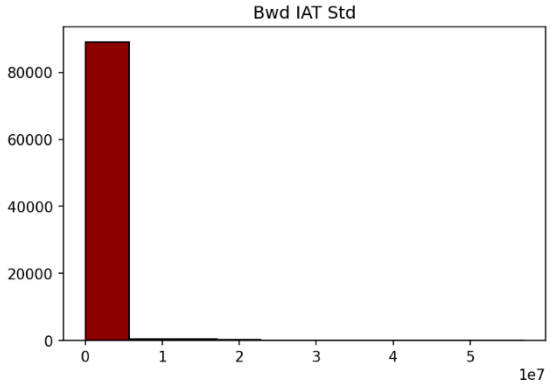

Fwd IAT Std
80000
60000
40000
20000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e7

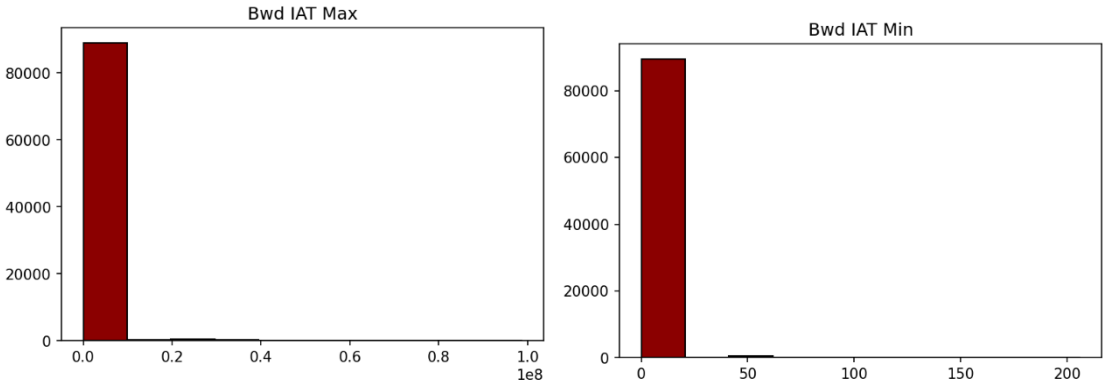

Fwd IAT Max
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e8


Fwd IAT Min
80000
60000
40000
20000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
1e7

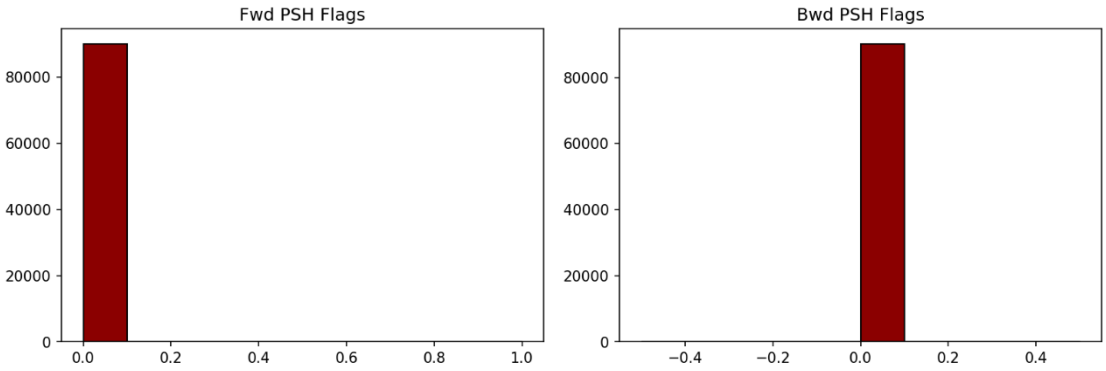

Bwd IAT Total
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1e8


Bwd IAT Mean
80000
60000
40000
20000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
1e7

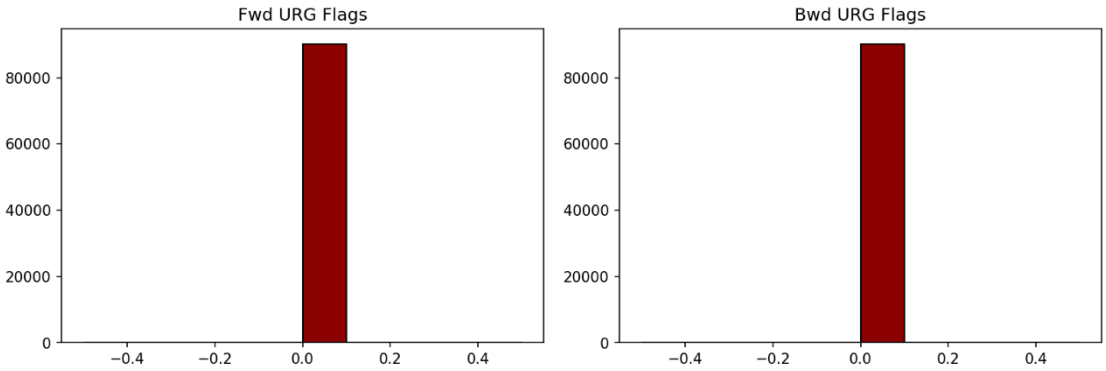

Bwd IAT Std
80000
60000
40000
20000
0
0
1
2
3
4
5
1e7


Bwd IAT Max
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0
1e8

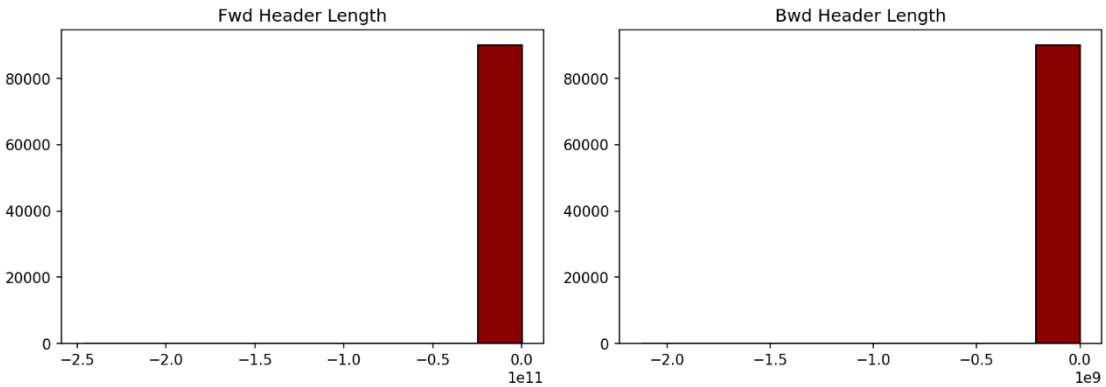
Bwd IAT Min
80000
60000
40000
20000
0
0
50
100
150
200


Fwd PSH Flags
80000
60000
40000
20000
0
0.0
0.2
0.4
0.6
0.8
1.0

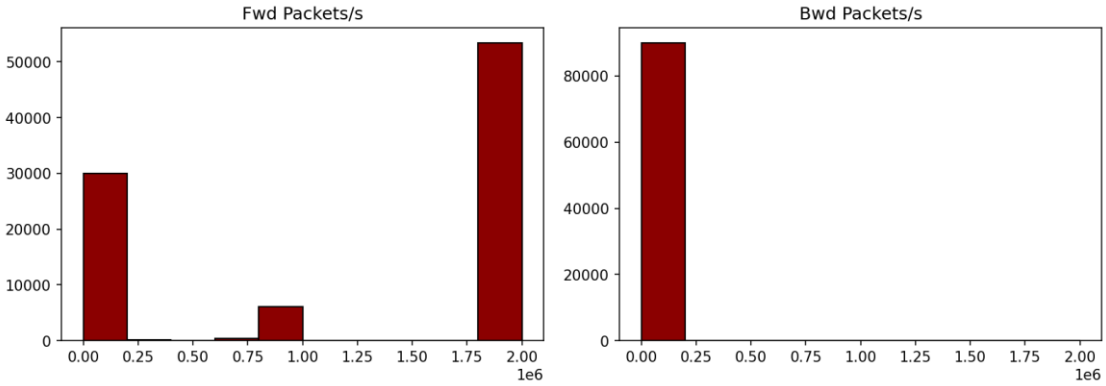
Bwd PSH Flags
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4


Fwd URG Flags
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4

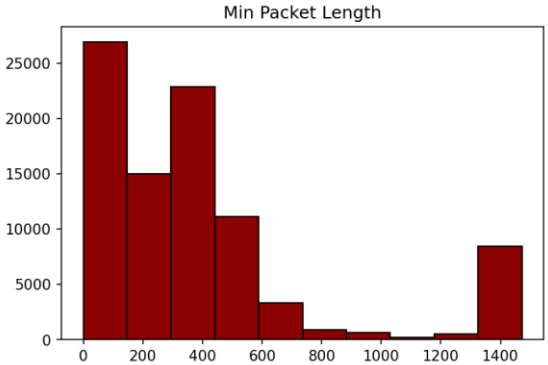
Bwd URG Flags
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4

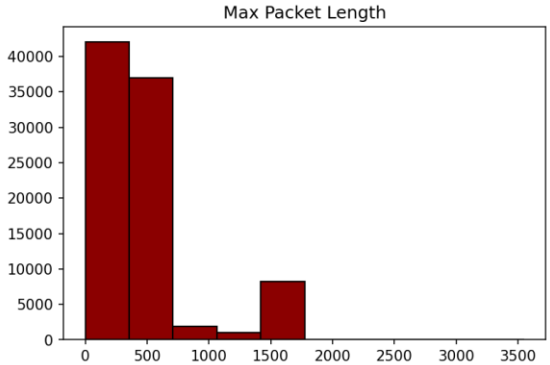

Fwd Header Length
80000
60000
40000
20000
0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
1e11

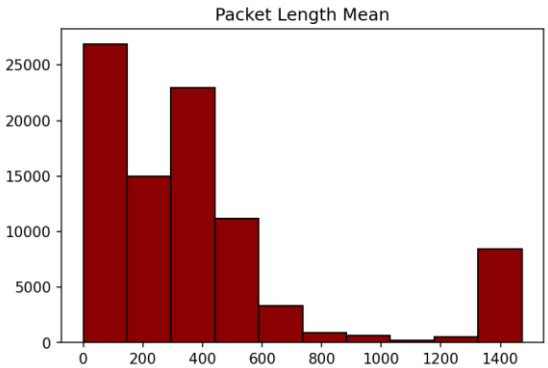
Bwd Header Length
80000
60000
40000
20000
0
−2.0
−1.5
−1.0
−0.5
0.0
1e9

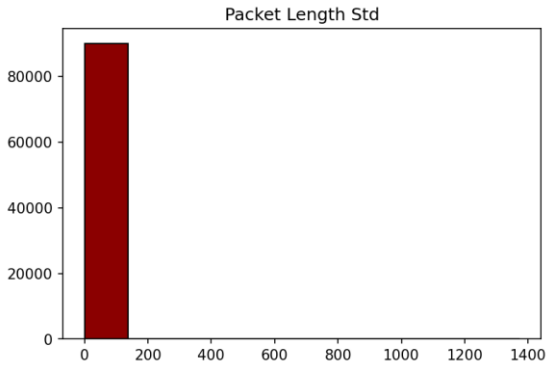

Fwd Packets/s
0
10000
20000
30000
40000
50000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6

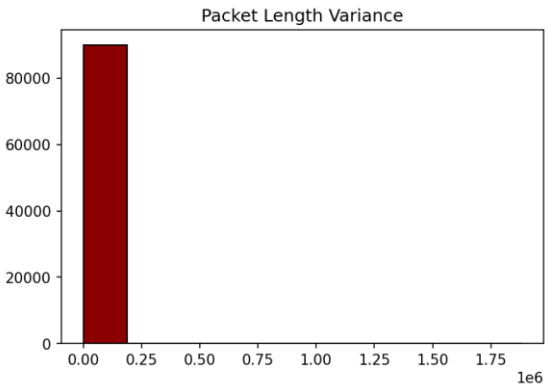
Bwd Packets/s
0
20000
40000
60000
80000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
1e6

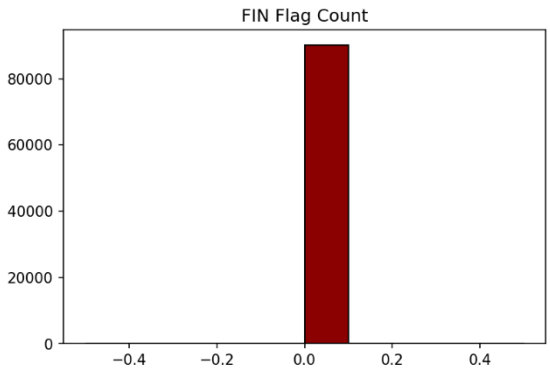

Min Packet Length
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400

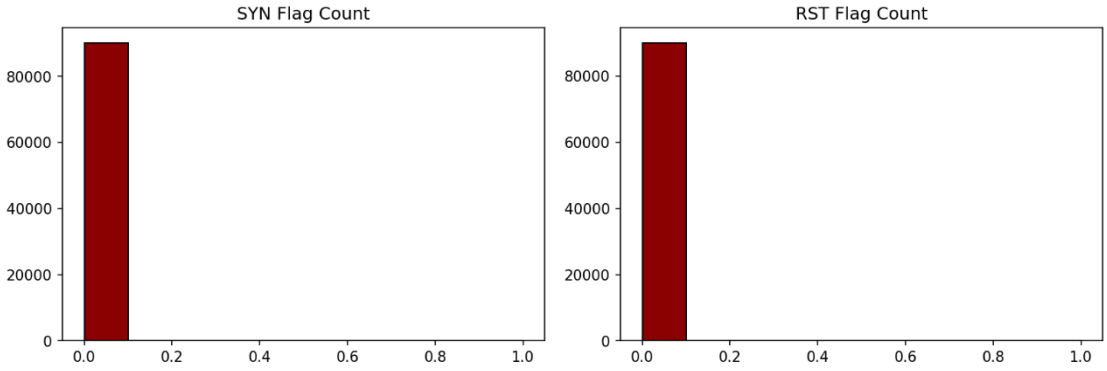

Max Packet Length
0
5000
10000
15000
20000
25000
30000
35000
40000
0
500
1000
1500
2000
2500
3000
3500


Packet Length Mean
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
1200
1400

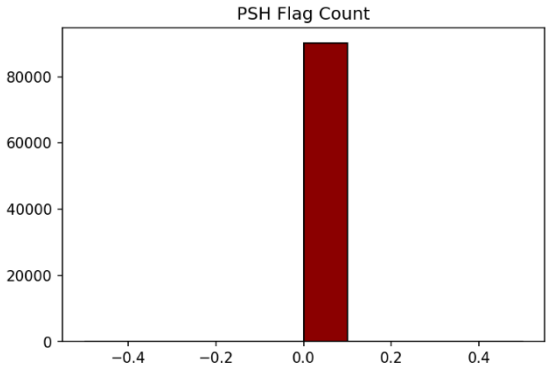

Packet Length Std
0
20000
40000
60000
80000
0
200
400
600
800
1000
1200
1400

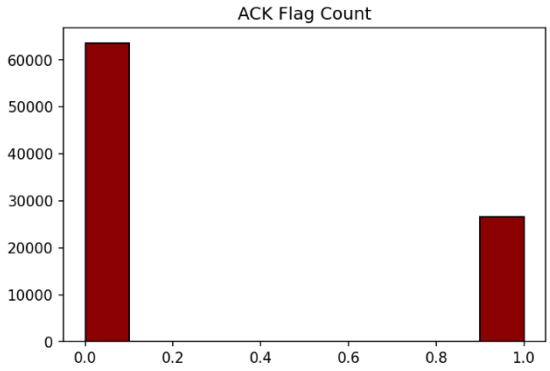

Packet Length Variance
0
20000
40000
60000
80000
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
1e6

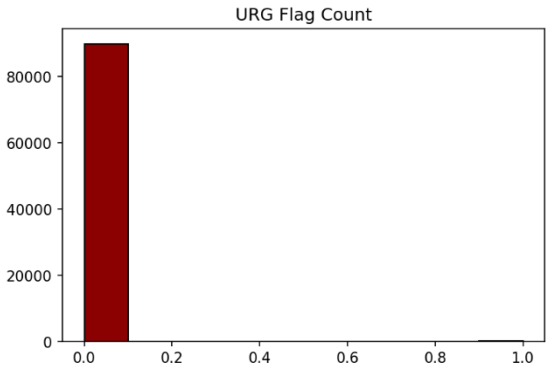

FIN Flag Count
0
20000
40000
60000
80000
−0.4
−0.2
0.0
0.2
0.4

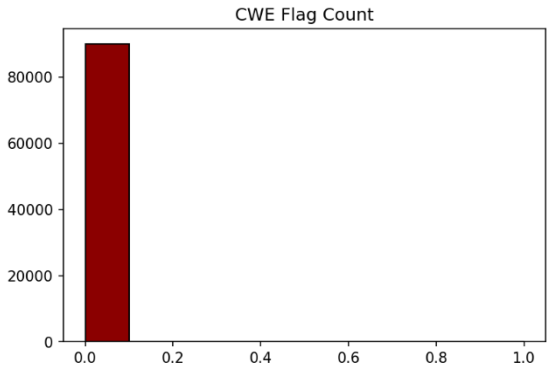

SYN Flag Count
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

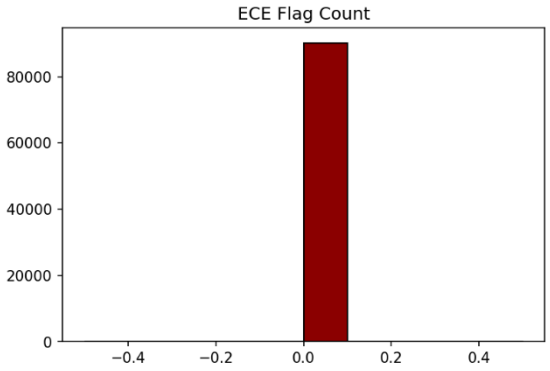
RST Flag Count
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

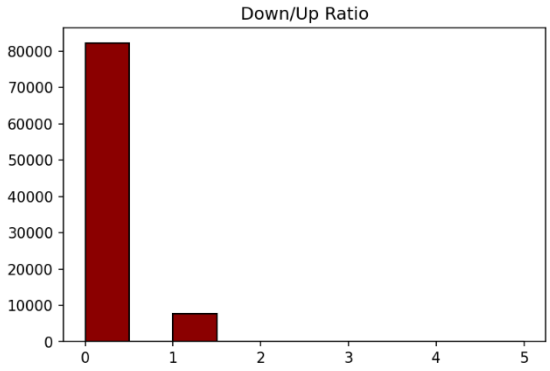

PSH Flag Count
0
20000
40000
60000
80000
−0.4
−0.2
0.0
0.2
0.4

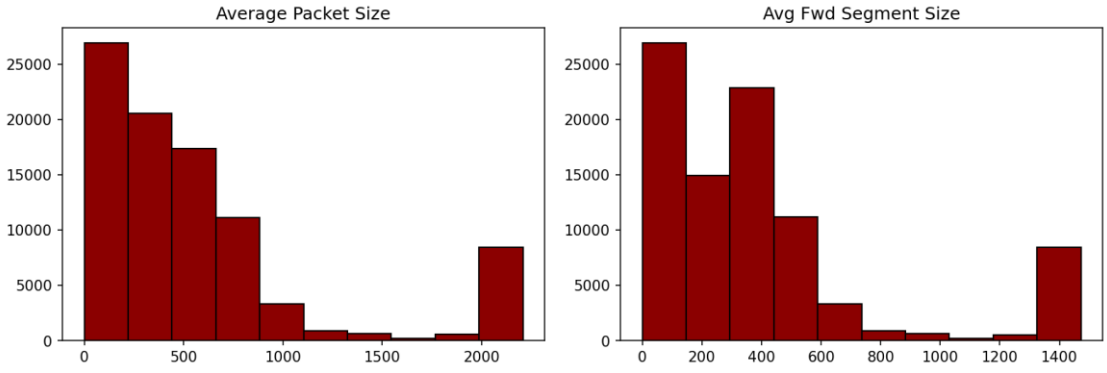

ACK Flag Count
0
10000
20000
30000
40000
50000
60000
0.0
0.2
0.4
0.6
0.8
1.0


URG Flag Count
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

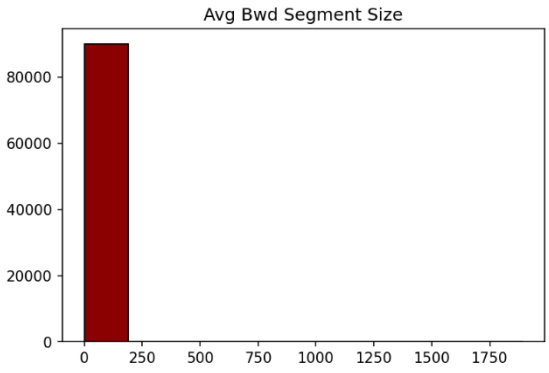

CWE Flag Count
0
20000
40000
60000
80000
0.0
0.2
0.4
0.6
0.8
1.0

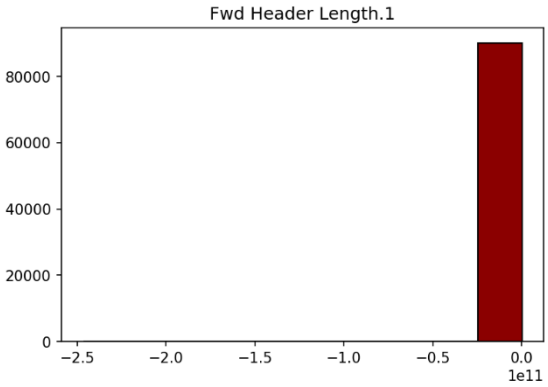

ECE Flag Count
0
20000
40000
60000
80000
−0.4
−0.2
0.0
0.2
0.4

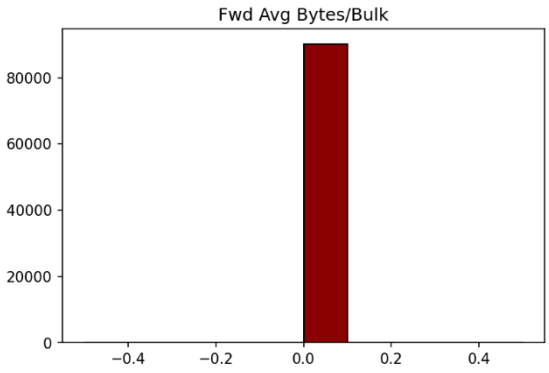

Down/Up Ratio
0
10000
20000
30000
40000
50000
60000
70000
80000
0
1
2
3
4
5

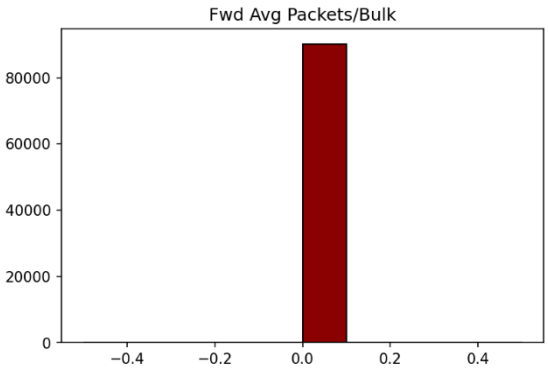

Average Packet Size
25000
20000
15000
10000
5000
0
0
500
1000
1500
2000

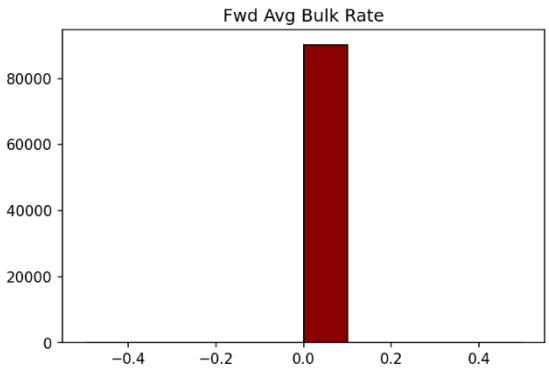
Avg Fwd Segment Size
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
1200
1400

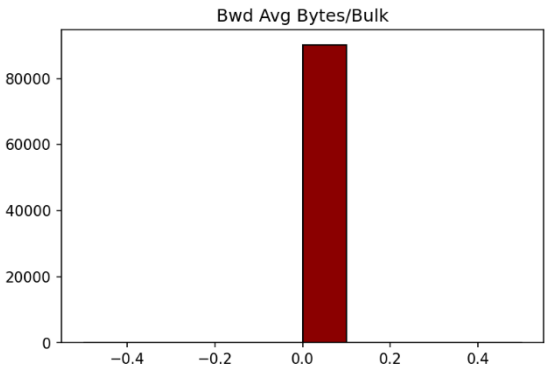

Avg Bwd Segment Size
80000
60000
40000
20000
0
0
250
500
750
1000
1250
1500
1750

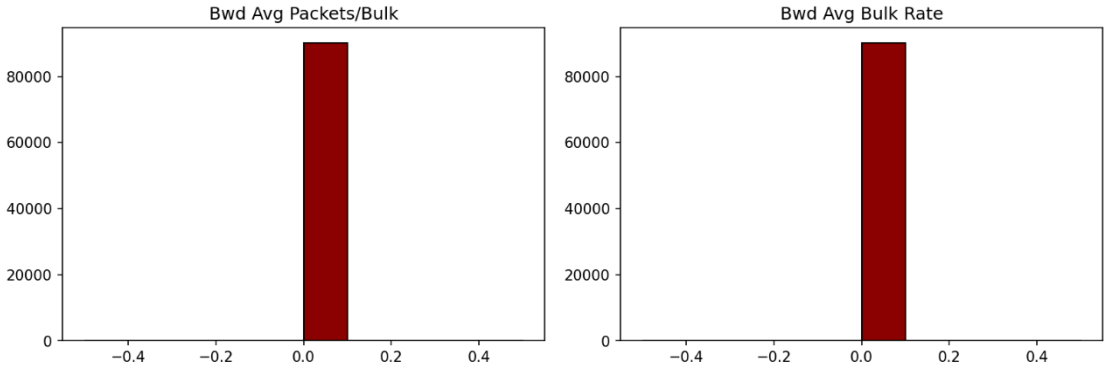

Fwd Header Length.1
80000
60000
40000
20000
0
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
1e11


Fwd Avg Bytes/Bulk
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4

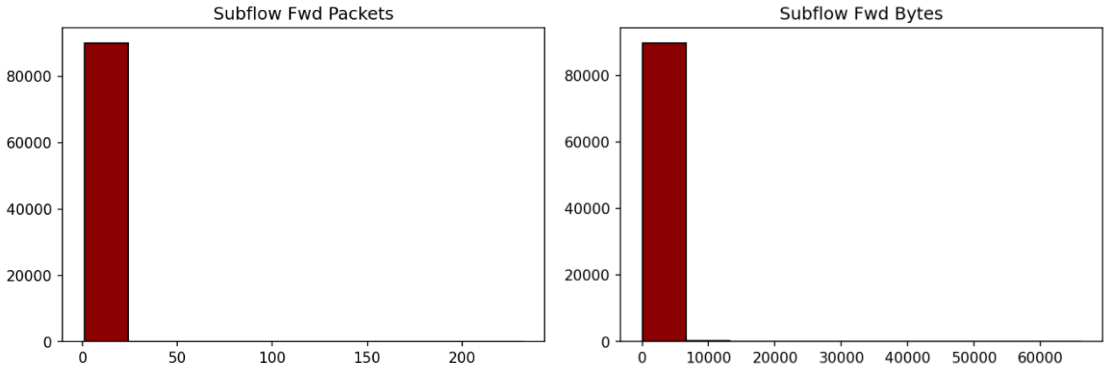

Fwd Avg Packets/Bulk
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4


Fwd Avg Bulk Rate
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4

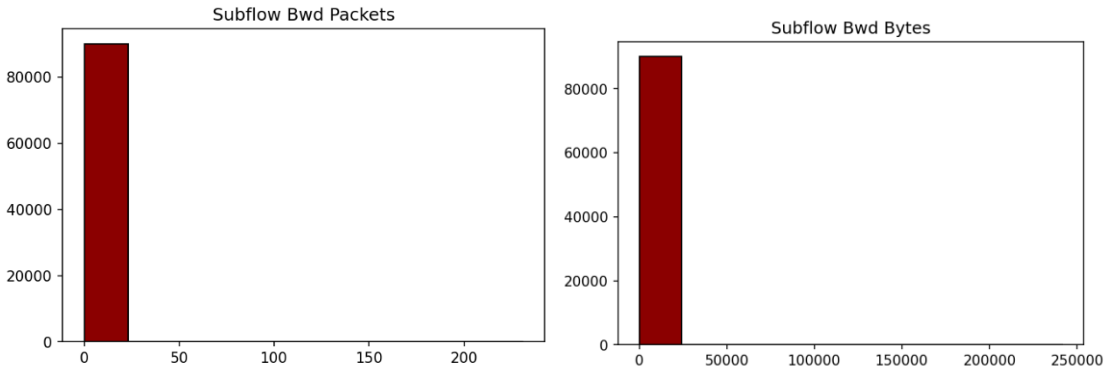

Bwd Avg Bytes/Bulk
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4


Bwd Avg Packets/Bulk
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4

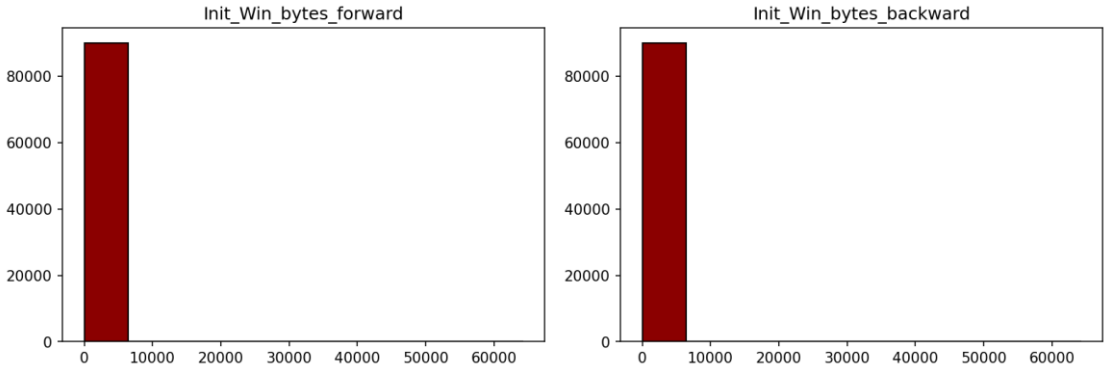
Bwd Avg Bulk Rate
80000
60000
40000
20000
0
−0.4
−0.2
0.0
0.2
0.4


Subflow Fwd Packets
80000
60000
40000
20000
0
0
50
100
150
200

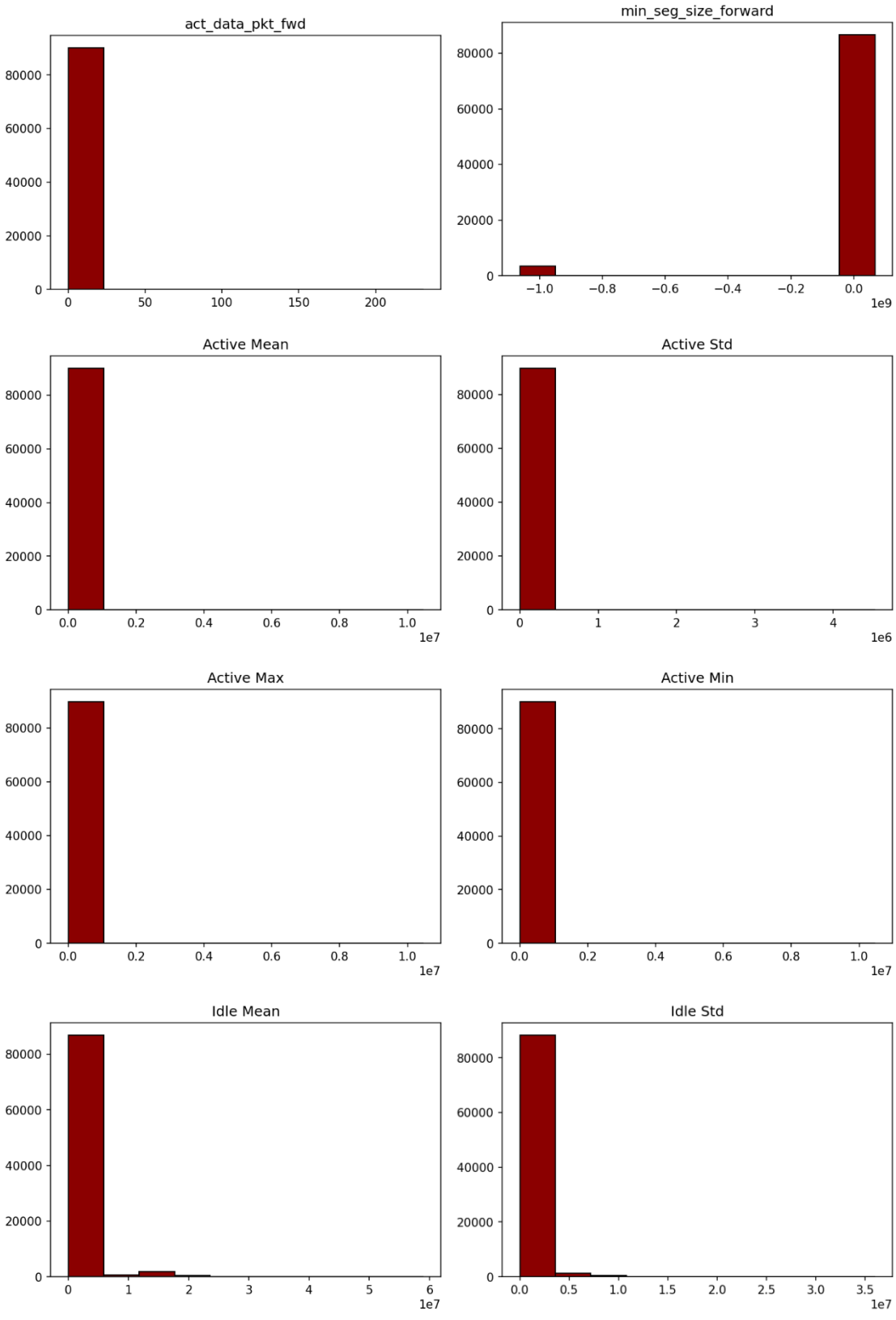
Subflow Fwd Bytes
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000

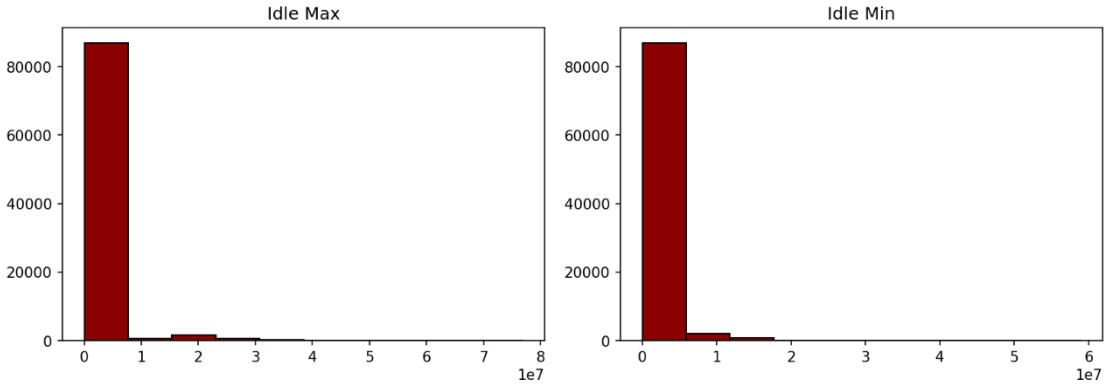

Subflow Bwd Packets
80000
60000
40000
20000
0
0
50
100
150
200

Subflow Bwd Bytes
80000
60000
40000
20000
0
0
50000
100000
150000
200000
250000

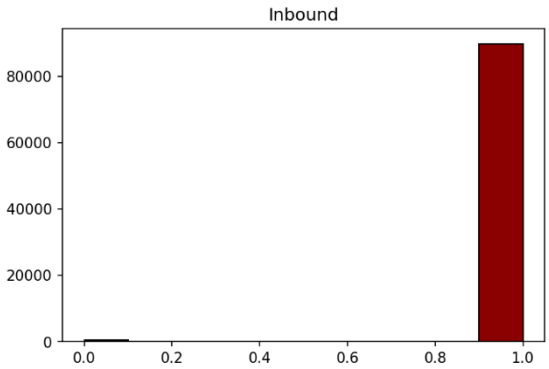

Init_Win_bytes_forward
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000

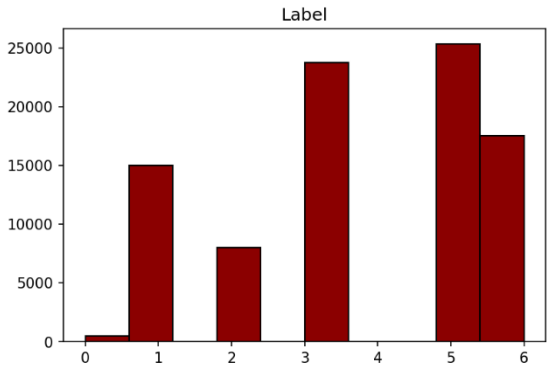
Init_Win_bytes_backward
80000
60000
40000
20000
0
0
10000
20000
30000
40000
50000
60000

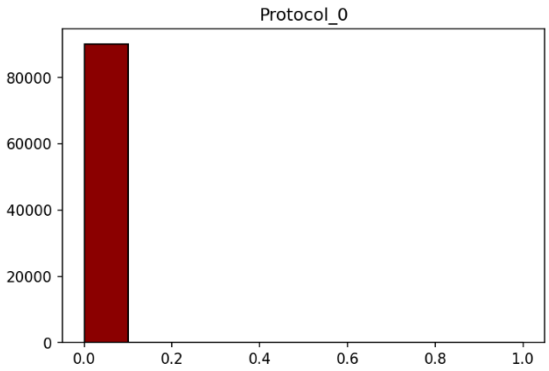

act_data_pkt_fwd
min_seg_size_forward
Active Mean
Active Std
Active Max
Active Min
Idle Mean
Idle Std
1e9
1e7
1e6

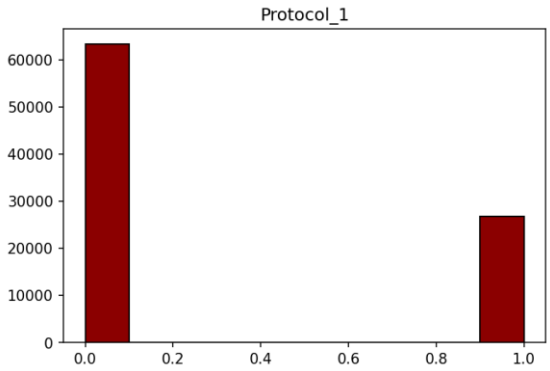

Idle Max
80000
60000
40000
20000
0
0 1 2 3 4 5 6 7 8
1e7

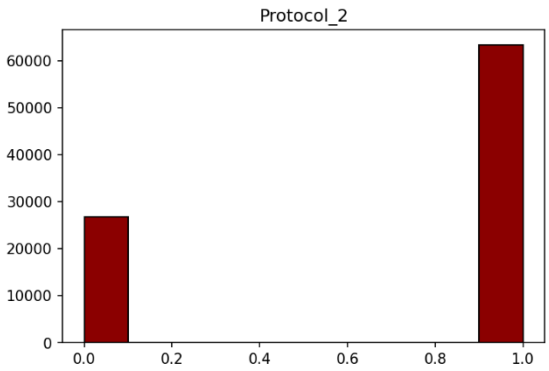
Idle Min
80000
60000
40000
20000
0
0 1 2 3 4 5 6
1e7


Inbound
80000
60000
40000
20000
0
0.0 0.2 0.4 0.6 0.8 1.0


Label
25000
20000
15000
10000
5000
0
0 1 2 3 4 5 6


Protocol_0
80000
60000
40000
20000
0
0.0 0.2 0.4 0.6 0.8 1.0


Protocol_1
60000
50000
40000
30000
20000
10000
0
0.0 0.2 0.4 0.6 0.8 1.0


Protocol_2
60000
50000
40000
30000
20000
10000
0
0.0 0.2 0.4 0.6 0.8 1.0

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Emre GÜNDÜZ

Doğum Tarihi ve Yeri : 27 Mart 2000 Bağcılar\Istanbul

Yabancı Dili : İngilizce

E-posta : emregunduzoffical@gmail.com

ÖĞRENİM DURUMU

Derece	Alan	Okul/Üniversite	Mezuniyet Yılı
Lisans	Bilgisayar Mühendisliği	Düzce Üniversitesi	2024
Lise		Çorlu Anadolu Lisesi	2018