ZARARLI YAZILIMLARIN STATİK ANALİZ İLE TESPİTİNDE MAKİNE ÖĞRENMESİ VE DERİN ÖĞRENME YÖNTEMLERİNİN KULLANIMI

YÜKSEK LİSANS TEZİ

NİSA VURAN SARI

MERSİN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

MERSİN
AĞUSTOS - 2023

Zararlı Yazılımların Statik Analiz ile Tespitinde Makine Öğrenmesi ve Derin Öğrenme Yöntemlerinin Kullanımı

YÜKSEK LİSANS TEZİ

Nisa VURAN SARI
ORCID ID: 0000-0001-7042-3031

MERSİN ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

DANIŞMAN
DR. ÖĞR. ÜYESİ MEHMET ACI
ORCID ID: 0000-0002-7245-8673

MERSİN
Ağustos-2023

ÖZET

Zararlı Yazılımların Statik Analiz ile Tespitinde Makine Öğrenmesi ve Derin Öğrenme Yöntemlerinin Kullanımı

Zararlı yazılım tespiti, bilgisayar sistem güvenliğinin önemli bir yönüdür. İnternet çağında, zararlı yazılımlar (virüsler, truva atları, fidye yazılımları ve botlar gibi) internet kullanıcıları için önemli derecede güvenlik tehditleri oluşturmuştur. Zararlı yazılımların çeşitliliği hızla ve katlanarak artmaktadır. Bu durum zararlı yazılımların doğru tanımlanması ve tespiti için yeni tekniklerin kullanılması gerekliliğini beraberinde getirmektedir. Derin Öğrenme (DÖ) ve Makine Öğrenimi (MÖ) modelleri, siber güvenlik araştırmalarının etkinliğini arttıran güçlü teknolojilerdir.
Çalışmada modelin eğitimi için kullanılan veriler, C-Prot Siber Güvenlik Teknolojileri San. ve Tic. A.Ş. firmasından temin edilen zararlı ve zararsız yazılımların özelliklerinden oluşmaktadır. Statik yazılım analizi kapsamında yazılım çalıştırılmadan 1000 adet zararsız yazılım 1000 adet zararlı yazılım analiz edilerek yazılımların kullandığı fonksiyonlar, kütüphaneler, dijital imzalar ve diğer özellikler ile yazılımın işlevsel yapısı çözümlenmiş ve özellikler oluşturulmuştur. Makine Öğrenmesi ve Derin Öğrenme yöntemleri kullanılmış ve elde edilen bulgular değerlendirilmiştir. Sonuçlar doğruluk, F1-skoru, kesinlik ve duyarlılık metrikleri açısından karşılaştırılarak değerlendirilmiştir. Kullanılan yöntemler arasında en başarılı sonucu %99,75 doğruluk oranıyla derin öğrenme yöntemlerinden biri olan Uzun Kısa Vadeli Hafıza Ağları algoritmasının verdiği gözlemlenmiştir.

Anahtar Kelimeler: Zararlı Yazılım Algılama, Zararlı Yazılım Analizi, Makine Öğrenmesi, Derin Öğrenme, Statik Analiz.

Danışman: Dr. Öğretim Üyesi Mehmet ACI, Mersin Üniversitesi, Bilgisayar Mühendisliği Anabilim Dalı, Mersin.

ABSTRACT

Detection of Malware by Static Analysis Using Machine Learning and Deep Learning Methods

Malware detection is an important aspect of computer system security. In the internet era, malware (such as viruses, trojans, ransomware, and bots) has posed significant security threats to internet users. The variety of malware is growing rapidly and exponentially. This situation brings with it the necessity of using new techniques for the correct identification and detection of malicious software. Deep Learning (DL) and Machine Learning (ML) models are powerful technologies that are transforming the effectiveness of cybersecurity research.

The data used for the training of the model in the study consists of harmful and harmless software obtained from C-Prot company. Within the scope of static software analysis, 1000 harmless software and 1000 malicious software were examined without running the software, and the functions, libraries, digital signatures and other features used by the software were analyzed and the functional structure of the software was extracted. Machine Learning and Deep Learning methods were used and the obtained results were evaluated. Calculated accuracy, f1-score, precision, and recall metrics are compared to determine success of the methods. It has been observed that the Long Short Term Memory Networks algorithm, which is one of the deep learning methods, gives the most successful result with an accuracy rate of 99.75% among the methods used.

Keywords: Malware Detection, Malware Analysis, Deep Learning, Machine Learning, Static Analysis.

Advisor: Asst. Prof. Dr. Mehmet ACI, University of Mersin, Department of Computer Engineering, Mersin.

TEŞEKKÜR

Yüksek lisans eğitimim boyunca, çalışmalarımda değerli görüşleri ile bana yol gösteren, bilgi, birikim ve tecrübeleri ile bana destek olan danışman hocam sayın Dr.Öğr.Üyesi Mehmet ACI'ya sonsuz teşekkür ve saygılarımı sunarım. Yüksek lisans tez jüri üyeleri sayın Doç.Dr. Erdinç AVAROĞLU ve Dr.Öğr. Üyesi Esra SARAÇ EŞSİZ'e değerli görüşleri için teşekkürlerimi sunarım.

Eğitim hayatım boyunca maddi ve manevi hiçbir konuda desteğini esirgememiş olan ve bugünlere gelmemde en büyük emeğe sahip olan annem Azime VURAN ve babam Selahattin VURAN'a güven duygusuyla hep yanımda oldukları için sonsuz teşekkür ederim.

İÇİNDEKİLER

TABLOLAR DİZİNİ

ŞEKİLLER DİZİNİ

SİMGELER VE KISALTMALAR

Kısaltma/Simge	Tanım
BiLSTM	Bidirctional Long Short-term Memory (Çift Yönlü Uzun Kısa Vadeli Hafıza Ağları)
CART	Classification and Regression Tree (Sınıflandırma ve Regresyon Ağacı)
CB	Catboost
CNN	Convolutional Neural Network (Evrişimli Sinir Ağları)
DAE	Deep Auto Encoder (Derin Oto Kodlayıcı)
DBN	Deep Belief Network (Derin İnanç Ağları)
DLL	Dynamic Link Library (Dinamik Bağlantı Kütüphanesi)
DT	Decision Tree (Karar Ağacı)
ET	Extra Trees (Ekstra Ağaçlar)
GB	Gradient Boosting (Gradyan Artırma)
GRU	Gated Recurrent Unit (Kapılı Tekrarlayan Birim)
IF	Isolation Forest (İzolasyon Ormanı)
KNN	K-Nearest Neighbor (K En Yakın Komşu)
LDA	Linear Discriminant Analysis (Lineer Diskriminant Analizi)
LGBM	Light Gradient Boosting Machine (Hafif Gradyan Artırma Makinesi)
LR	Logistic Regression (Lojistik Regresyon)
LSTM	Long Short-term Memory (Uzun Kısa Vadeli Hafıza Ağları)
MLP	Multi-Layer Perceptron (Çok Katmanlı Algılayıcı)
NB	Naive Bayes
PAC	Passive Aggresive Classifier (Pasif Agresif Sınıflandırıcı)
PE	Portable Executable (Taşınabilir Yürütülebilir)
PReLU	Parametric Recified Linear Unit (Parametrik Doğrultulmuş Lineer Birim)
QDA	Quadratic Discriminant Analysis (İkinci Dereceden Diskriminant Analizi)
RBF	Radial Basis Function (Radyal Temel Fonksiyon)
ReLU	Recified Linear Unit (Doğrultulmuş Lineer Birim)
RF	Ranfom Forest (Rastgele Orman)
RNN	Recurrent Neural Network (Tekrarlayan Sinir Ağı)
ROC	Receiver Operating Characteristic (Alıcı Çalışma Karakteristiği)
SGD	Stochastic Gradient Descent (Stokastik Gradyan İnişi)
SMO	Sequantial Minimal Optimization (Sıralı Minimum Optimizasyon)
SVM	Support Vector Machine (Destek Vektör Makinesi)
XGB	eXtreme Gradient Boosting

1. GİRİŞ

İnternet hızla günlük yaşamın ayrılmaz bir parçası haline gelmektedir. Bununla birlikte, hızlı benimsenmesi, onu kötüye kullanıma açık hale getirmiştir (Islam, 2013). Bilişim teknolojilerinin yaygınlaşması ile birlikte bilişim sistemlerini tehdit eden zararlı yazılımların çeşitlendiği ve etkilerinin arttığı görülmektedir. Zararlı yazılım, kullanıcı tarafından yetkilendirilmemiş işlemleri gerçekleştiren zararlı programlardır. Bu yazılımlar kullanıcıların programlanabilir cihazlarına, web sitelerine veya ağlarına zarar verebilir veya yetkisiz erişim sağlayabilirler. Zararlı yazılımların Virüs, Solucan, Truva Atı, Rootkit, Backdoor, Botnet, Spyware, Adware gibi farklı türleri vardır (Patil, 2020). Cisecurity.org tarafından yayımlanan verilere göre 2022 yılının en popüler 10 zararlı yazılımı Shlayer, CoinMiner, NanoCare, AgentTesla, Zeus, Arechclient2, Delf, Mirai, CryptoWall ve RedLine'dır. Bu yazılımlar genellikle tarayıcı hareketlerini gözlemlemek, finansal verilere ulaşmak veya verileri kalıcı olarak şifreleyip fidye talep etmek üzere programlanmaktadır. Saldırganlar, sistem güvenlik açıklarından faydalanmayı ve kullanıcılar veya sistem verileri üzerinde avantaj elde etmeyi amaçlamaktadırlar.

Birçok yeniliği beraberinde getiren ve hayatımızı her anlamda kolaylaştıran teknoloji çağı siber risk ve tehditleri de hayatımıza dahil etmiştir. Zararlı yazılımların çeşitliliği artmaya devam ederken, koruma tarayıcıları virüslere karşı yeterli koruma sağlayamaz hale gelmektedir ve bu durum milyonlarca bilgisayarın saldırıya uğramasına sebep olmaktadır. Kaspersky Labs'e (2016) göre, 2015 yılında 6.563.145 farklı bilgisayar saldırıya uğramış ve 4.000.000 benzersiz (unique) zararlı yazılım tespit edilmiştir. Buna karşılık Juniper Research (2016), veri ihlallerinin maliyetinin 2019 yılında küresel olarak 2,1 trilyon ABD Dolarına çıkacağını belirlemiştir (Chumachenko, 2017).

Zararlı yazılım analizi, zararlı yazılım örneklerinin bir kullanıcı sistemi üzerindeki davranışını, işlevselliğini ve etkisini analiz etme sürecidir (Harshalatha, 2020). Zararlı yazılım analizinde şüpheli dosyaları araştırmak için statik ve dinamik olmak üzere iki farklı yaklaşım vardır. (Şahin, 2021). Statik yaklaşımlarda analistler dosyaları yürütmeden araştırırlar. Statik analiz, zararlı yazılımları tespit etmek için diziler, n-gramlar, fonksiyonlar, işlem kodları, bayt dizileri, kütüphaneler ve çağrı grafikleri gibi desenlerden bilgi ve özellikleri çıkarmak için kullanılır (Patil, 2020). Ayrıca yazılımın bir imza veri tabanında bulunup bulunmadığının kontrolü de sağlanabilir. Yazılımın içerdiği dizgiler (stringler) analiz edilerek hangi kütüphaneleri ve/veya Dinamik Bağlantı Kitaplıkları (Dynamic Link Library, DLL) kullandığı incelenebilir. Genel olarak, statik analiz, bir programın kaynak kodunu girdi olarak alan, bu girişi kodu çalıştırmadan inceleyen ve ayrıca kod yapısını ve ifade dizilerini kontrol eden ve sonucu çıktı yapan otomatik bir analizdir.

Analiz için statik ve dinamik yaklaşımlar kullanılabilse de yakın insan analizi gerektiren örneklerin sayısını en aza indirebilmek ve zararlı yazılım analizi ve sınıflandırma adımlarını otomatikleştirmek için makine öğrenmesi ve derin öğrenme algoritmaları kullanılabilmektedir. Statik ve/veya dinamik analizden elde edilen kalıpları incelemek ve bilinmeyen zararlı yazılımları ilgili ailelere

sınıflandırmak için makine öğrenmesi teknikleri (kümeleme, sınıflandırma gibi) ve derin öğrenme kullanılmaktadır.

Bu çalışmada, zararlı yazılım analiz yöntemleri ile makine öğrenmesi ve derin öğrenme yöntemleri kullanılarak zararlı yazılımların tespit edilmesi üzerine çalışılmıştır. Çalışmada modelin eğitimi için kullanılan veriler, C-Prot Siber Güvenlik Teknolojileri San. ve Tic. A.Ş. tarafından temin edilen zararlı ve zararsız yazılımlara ait özelliklerden oluşmaktadır. Statik yazılım analizi kapsamında yazılım çalıştırılmadan 1000 adet zararsız yazılım 1000 adet zararlı yazılım analiz edilerek yazılımların kullandığı fonksiyonlar, kütüphaneler, dijital imzalar ve diğer özellikler ile yazılımın işlevsel yapısı çözümlenmiş ve özellikler oluşturulmuştur. Daha sonra veriler ön işlemden geçirilmiştir. Modelin makine öğreniminde ve derin öğrenmede eğitilebilmesi için kategorik girdilerin sayısal terimlere dönüştürülmesi gerekmektedir. Veri kümesindeki tüm özellikler "0" veya "1" olarak ikili bir değere normalleştirilmiştir.

Bir dosyanın zararlı yazılım olup olmadığını tespit etmek bir sınıflandırma problemidir (Zararlı/Zararsız). Bu yaklaşımda, genellikle veri kümesi etiketleri, dosyanın zararlı veya zararsız olduğunu göstermektedir. Veri kümesi, eğitim kümesi ve test kümesi olarak ikiye ayrılır. Eğitim kümesi, belirli bir modeli eğitmek için kullanılır. Ayrıca modelin daha iyi performans göstermesi için çapraz doğrulama tekniği kullanılabilir. Eğitilen modele test verileri girdi olarak verilerek çıktı verisini tahmin etmesi sağlanır. Model eğitildikten sonra, bu model test veri kümesine uygulanır. Bu test aşamasında model etiketlerden habersizdir. Model her dosya için etiketi zararlı veya zararsız olarak tahmin eder. Daha sonra modelin doğruluğu, kaç dosyanın doğru şekilde sınıflandırıldığına bağlı olarak hesaplanır.

Bu çalışmada sınıflandırma işlemleri için makine öğrenmesi yöntemlerinden rastgele Orman (Random Forest, RF), Karar Ağaçları (Decision Tree, DT), K-En Yakın Komşu (K-Nearest Neigbour, KNN), Gaussian Naive Bayes (NB), Catboost (CB), LightGBM (LGBM), Gradient Boosting (GB) ve eXtreme Gradient Boosting (XGB) kullanılmıştır. Derin öğrenme yöntemlerinden ise Evrişimli Sinir Ağları (Convolutional Neural Network, CNN), Tekrarlayan Sinir Ağları (Recurrent Neural Network, RNN), Uzun ve Kısa Vadeli Hafıza Ağları (Long Short Term Memory, LSTM), Çift Yönlü Uzun ve Kısa Vadeli Hafıza Ağları (Bidirectional Long Short Term Memory, BiLSTM) ve Geçitli Tekrarlayan Birimler (Gated Recurrnt Units, GRU) kullanılmıştır.

Makine Öğrenmesi ve Derin Öğrenme yöntemleri kullanılarak elde edilen bulgular doğruluk, f1-skoru, kesinlik ve duyarlılık metrikleri açısından karşılaştırılarak değerlendirilmiştir. Kullanılan yöntemler arasında en başarılı sonucu %99,75 doğruluk oranıyla LSTM algoritmasının verdiği gözlemlenmiştir.

2. KAYNAK ARAŞTIRMALARI

Zararlı yazılımlar geçmişten günümüze bilgisayar sistemleri için büyük bir tehdit oluşturmuştur. Geçmişten beri yapılan zararlı yazılım tespiti çalışmalarına her geçen gün yenisi eklenerek yeni teknik ve yöntemler test edilmiştir. Mevcut operasyon yöntemlerinin, yeni tehditlere karşı verimliliğinin düşmesiyle birlikte makine öğrenimi ve derin öğrenme yöntemleri gibi yapay zeka destekli yöntemlerin, zararlı yazılım algılamalarını iyileştirdiği ve mevcut yöntemlerden daha iyi performans gösterdiği gözlemlenmiştir. Bu bölümde, literatürde bulunan zararlı yazılım algılama ile ilgili çalışmaların bir özeti sunulmuştur.

2.1. Literatürde Zararlı Yazılım Algılama Üzerine Yapılış Çalışmalar

Patil ve diğ. (2020) sistem çağrıları, kodlar, bölümler ve bayt kodları dahil olmak üzere zararlı yazılım dosyalarından çok sayıda özellik kümesi çıkaran bir sisteme dikkat çekmiştir. Microsoft'a ait, Kaggle web sitesinde bulunan zararlı yazılım veri kümesi üzerinde çalışmışlardır. Veri kümesi 9 farklı zararlı yazılım ailesine ait 10868 zararlı yazılım dosyası içermektedir. Çalışmada, bu özelliklerin her biri üzerinde makine öğrenimi ve derin öğrenmeye dayalı yöntemlerin etkinliği karşılaştırılmıştır ve sonuçlara göre, sistem çağrıları için özellik vektörünün en yüksek doğruluğa ulaştığı görülmüştür.

Algahtani ve diğ. (2019) Android cihazlarda zararlı yazılımları belirlemek için makine öğrenimi tabanlı sınıflandırıcılara odaklanmıştır. Destek Vektör Makinesi (Support Vector Machine, SVM), NB, Perceptron, J48, OneRand (OneR), Derin Ağ (Deep Network, DN) algoritmaları dahil olmak üzere farklı makine öğrenmesi algoritmaları, tespit ve algılama çerçevelerinin geliştirilmesinin temelini oluşturmuştur. Kullanılan veri kümesi 2081 iyi huylu uygulama ve 91 kötü huylu uygulama içermektedir. OneR ve J48 algoritmaları, %0,00 yanlış pozitif oranıyla %100 doğruluk elde etmiştir.

Zararlı yazılımı tanımak için Schultz ve diğ. (2001), Windows tabanlı taşınabilir yürütülebilir (Portable Executables, PE) dosyalarına üç farklı yaklaşım uygulaımşlardır. Bu teknikler, bilinen zararlı yazılımları bulmak için kullanılmıştır. İlk yaklaşımda DLL, işlev çağrıları ve bu işlevlerin çağrılma sıklığı gibi öznitelikleri kullanan bir yöntem kullanmışlardır. İkinci yaklaşım için string yazılımını kullanarak PE dosyalarından çıkarılan ikili verileri kullanmışlardır. Son olarak, hex dump yazılımını kullanarak program tarafından sağlanan 2 baytlık dizileri kullanmışlardır. Bu çalışmada tespit, için RIPPER, NB ve çeşitli sınıflandırıcı teknikleri kullanılmıştır. NB kullanan sistem, veri başına en yüksek algılama oranını sağlamıştır.

2.1.1. Makine Öğrenmesi Yöntemleri ile Yapılmış Çalışmalar

Harshalatha ve diğ. (2020), makine öğrenimi algoritmalarını kullanarak daha önce var olan zararlı yazılım algılama sınıflandırma çalışmalarının literatür çalışmasını sunmaktadırlar. Bu

yaklaşımda kullanılan sınıflandırıcı modelleri Rastgele Orman (Random Forest, RF), BayesNet, Çok Katmanlı Algılayıcı (Multi-Layer Perceptron, MLP) ve SVM'dir. İlk olarak, tüm zararlı yazılım örnekleri 10 kat çapraz doğrulama ile test edilmiş ve sınıflandırıcı sonuçları RF'nin %98 doğruluk oranı ile en iyi performansa sahip olduğunu göstermiştir. Diğer yandan RF sınıflandırıcı farklı veri kümeleri için uygulandığında aynı RF doğruluğunun %12 azaldığı gözlemlenmiştir.

Kötü niyetli yürütülebilir dosyaları bulmak için Kolter ve diğ. (2004) makine öğrenme yöntemlerini kullanmıştır. Çalışmada karakteristik veri özelliği olarak n-gram bayt kodları kullanılmıştır. Eğitim örnekleri olarak, topladıkları 1651 zararlı yürütülebilir dosyanın ve 1971 iyi huylu yürütülebilir dosyanın her biri kodlanmıştır. Çalışmada Naive Bayes (NB), Karar Ağacı (Decision Tree, DT), SVM ve Boosting algoritmaları kullanılmış ve tahmin için en uygun n-gramlar seçilip test edilmiştir. Güçlendirilmiş DT için Alıcı Çalışma Karakteristiği (Receiver Operating Characteristic**,** ROC) eğrisinin altındaki alan 0,996'dır ve diğer tüm yaklaşımlardan daha iyi performans göstermiştir.

Azeez ve diğ. (2021), zararlı yazılım tespiti için toplu öğrenmeye dayalı bir yöntem önermektedir. Temel aşama sınıflandırması, tam bağlantılı ve tek boyutlu evrişimli sinir ağlarının (Convolutional Neural Network, CNN) yığınlanmış bir topluluğu tarafından yapılırken, son aşama sınıflandırması bir makine öğrenmesi algoritması tarafından yapılır. Bir meta-öğrenici için, DT, doğrusal ve radyal temel fonksiyon (Radial Basis Function, RBF) çekirdekleri ile SVM, RF, k-en yakın komşu (K-Nearest Neighbor, KNN), MLP, AdaBoost sınıflandırıcı, Ekstra Ağaçlar (ExtraTrees, ET) sınıflandırıcısı, izolasyon ormanı (Isolation Forest, IF), Gauss Naive Bayes (NB), doğrusal diskriminant analizi (Linear Discriminant Analysis, LDA), ikinci dereceden diskriminant analizi (Quadratic Discriminant Analysis, QDA), lojistik regresyon (LR), pasif agresif sınıflandırıcı (Passive Aggressive Classifier, PAC), ridge sınıflandırıcı (Ridge Classifier, RC) ve stokastik gradyan iniş (Stochastic Gradient Descent, SGD) sınıflandırıcıları kullanılmıştır. Karşılaştırma için beş makine öğrenimi algoritması kullanılmıştır: NB, DT, RF, gradyan artırma ve adaboost. Windows PE zararlı yazılım veri kümesi üzerinde yapılan deneylerin sonuçları sunulmaktadır. En iyi sonuçlar, yedi sinir ağı topluluğu ve son aşama sınıflandırıcısı olarak ET sınıflandırıcısı ile elde edilmiştir.

Tian ve diğ. (2009), sınıflandırma algoritmaları kullanan, dizi bilgilerine dayanan bir zararlı yazılım sınıflandırma çalışması sunmaktadır. Yapılan testlerde, paketlenmemiş truva atları, virüsler ve temiz dosyalar olmak üzere 1367 örnekten diziler elde edilmiştir. Her örnekte bulunan dizileri tanımlayan bilgiler ağaç tabanlı sınıflandırıcılar, KNN, istatistiksel algoritmalar ve AdaBoost dahil olmak üzere çeşitli sınıflandırma algoritmaları ile değerlendirilmiştir. Paketlenmemiş zararlı yazılımlar ve temiz dosyalar üzerinde k-kat çapraz doğrulama kullanarak RF sınıflandırıcısı ile %97'lik bir doğruluk oranı elde edilmiştir.

Muhammad ve diğ. (2019) tarafından yapılan çalışmada dinamik ve statik analiz yötemleri ile elde edilen veri üzerinde ve makine öğrenmesi yöntemleri kullanılarak zararlı yazılımların tespiti yapılmıştır. Statik özellikler, 39.000 zararlı ikili dosyadan ve 10.000 zararsız dosyadan elde edilmiştir. Dinamik olarak 800 iyi huylu dosya ve 2.200 zararlı yazılım dosyası Cuckoo Sandbox'ta analiz edilmiş

ve 2300 özellik elde edilmiştir. Veriler Lojistik regresyon, DT, RF, Adaboost, Bagging sınıflandırıcı, ağaç sınıflandırıcı ve gradyan sınıflandırıcı ile test edilmiştir. Dinamik zararlı yazılım analizinin doğruluğu %94,64 iken statik analizin doğruluğu %99,36'dır.

Shatnawi ve diğ. (2022) çalışmalarında, Android izinlerine ve API çağrılarına dayalı olarak zararlı yazılım tespiti için statik bir temel sınıflandırma yaklaşımı sunmuşlardır. Bu yaklaşım, kapsamlı bir yeni Android zararlı yazılım veri kümesine (CIC InvesAndMal2019) karşı SVM, KNN ve NB gibi iyi bilinen üç Makine Öğrenimi algoritmasına dayanmaktadır.

Santos ve diğ. (2013) operasyonel kodların (statik olarak elde edilen) oluşma sıklığını bir yürütülebilir dosyanın (dinamik olarak elde edilen) yürütme izinin bilgisiyle birleştiren bilinmeyen hibrit bir zararlı yazılım dedektörü olan OPEM'i önermektedir. Bu hibrit yaklaşımın, ayrı ayrı çalıştırılması durumunda her iki yaklaşımın da performansı artırdığı gözlemlenmiştir. Çalışmada KNN, DT, RF, SVM ve NB algoritmaları kullanılmış ve statik, dinamik ve hibrit yaklaşımlar ayrı ayrı test edilmiştir. SVM: Normalleştirilmiş Polinom Çekirdek, her yaklaşım için en iyi doğruluk sonucunu elde etmiştir (Statik: %95,90, Dinamik: %77,26, Hibrit: %96,60).

Aydoğan ve diğ. (2014) çalışmarında, makine öğrenimi tekniklerini kullanarak daha önce hiç görülmemiş yeni zararlı yazılımları tespit etmeyi amaçlamıştır. Yazılımın yapısal bilgileri kullanılarak zararlı yazılımları iyi huylu yazılımlardan ayıran özellikler çıkarılmış ve zararlı yazılımları tespit etmek için bu özelliklere makine öğrenimi teknikleri uygulanmıştır. Örnek tabanlı öğrenme (Instance-Based Learning, IBL), Karar ağacı tabanlı bir algoritma olan J48, NB, Sıralı Minimal Optimizasyon (Sequential minimal optimization, SMO) kullanan SVM, MLP, Bagging J48 (BJ48), Karar Tablosu (KT), RF algoritmaları eğitim ve test için kullanılmıştır. En yüksek tespit oranı %96.2 ile BJ48 tarafından elde edilmiştir. En düşük yanlış pozitif (False Positive, FP) oranı ise RF tarafından elde edilmiştir.

Tahtacı ve diğ. (2020), Android tabanlı dosyaların n-gram özellikleri ile makine öğrenmesi modeli üzerine çalışılmıştır. Modeller varyans eşik değeri ve bilgi kazancı öznitelik çıkarma yöntemleri ile birleştirilmiştir. 3000 adet Android Paket (APK) dosyası apktool aracı ile kaynak koda dönüştürülerek uygulamaların opcode'ları elde edilmiştir. N-gram özellikleri kullanılarak DT, KNN, NB, LR, RF ve SVM makine öğrenmesi yöntemleri ile eğitilen modeller test edilmiştir. En yüksek test skoruna 3-gram için özellik sayısı 2'ye düşürüldükten sonra ulaşılmıştır (KNN ve SVM modelleri %100'lük bir test skoru elde etmiştir).

2.1.2. Derin Öğrenme Yöntemleri ile Yapılmış Çalışmalar

Rafique ve diğ. (2019) tarafından gerçekleştirilen çalışmada, farklı zararlı yazılım ailelerini sınıflandırmak için statik yöntemlere dayalı bir derin öğrenme tabanlı zararlı yazılım algılama (DLMD) tekniğini sunmaktadır. Önerilen DLMD tekniği, özellik mühendisliği için hem bayt hem de ASM dosyalarını kullanmaktadır, böylece zararlı yazılım ailelerini sınıflandırmaktadır. İlk olarak, iki farklı

Derin CNN kullanılarak bayt dosyalarından özellikler çıkarılmıştır. Bundan sonra, SVM sınıflandırıcısı kullanılarak temel ve ayırt edici opcode özellikler seçilmiştir. Son olarak, hibrit özellik alanı, dokuz farklı zararlı yazılımın tümünü sınıflandıran MLP eğitim için kullanılır. Eğitim sırasında optimize edilmiş parametrelerle birlikte SVM, MLP ve CNN için kayıp puan (log loss) hesaplanır. Doğrusal SVM en yüksek kayıp puanı değerini (0,8948), CNN ise en düşük kayıp puanı değerini vermiştir (0,1514).

Bulut ve diğ. (2017) tarafından yapılan araştırmada, daha önce bilinmeyen zararlı yazılımları yürütmeden tespit etmek için derin öğrenme tekniklerini kullanılmıştır. Mobil yazılımda kullanılan niteliklerden öznitelikler çıkarılmış, otomatik kodlayıcı ile elde edilen özniteliklerin ağırlıklarının optimizasyonu gerçekleştirilmiş ve beş katmanlı yapay sinir ağı kullanılarak sınıflandırma işlemi gerçekleştirilmiştir. Test sonucunda %93,67 bir oran ile zararlı yazılım doğru tespit edilmiştir. 3229 iyi huylu Android uygulaması ve 1668 zararlı Android uygulaması için işletim sistemi tarafından istenen izinler GGOK ile önceden işlenmiş ve çok katmanlı Yapay Sinir Ağı (YSA) ile kategorilere ayrılmıştır. Önerilen sistem bilinmeyen zararlı yazılımları %93,67 doğruluk oranı ile tespit etmiştir.

Yuxin ve diğ. (2019) çalışmalarında, zararlı yazılımların işlem kodu dizileri ile temsil ederek, derin inanç ağı (Deep Belief Network, DBN) ile tespit edilmesini amaçlamıştır. DBN'lerin performansı KNN, DT ve SVM gibi üç temel sınıflandırma algoritması ile karşılaştırılmıştır. DBN ile, etiketlenmemiş veriler kullanılarak eğitilip, verimli katman katman öğrenme tekniği ile, verilerdeki karmaşık yapıları ve bağımlılıkları modellemek için çok sayıda gizli katman kullanılması mümkün kılınmıştır. Sonuçlar, DBN modelinin temel modellerden daha doğru algılama ve tespit yaptığını göstermektedir.

Vinayakumar ve diğ. (2019) bu çalışmada, genel ve özel veri kümeleri kullanarak zararlı yazılım algılama ve sınıflandırma için klasik makine öğrenmesi algoritmaları ve derin öğrenme mimarilerini değerlendirmiştir. Çalışmada statik, dinamik ve görüntü işleme tabanlı hibrit yaklaşım kullanılmıştır. Çalışmada LR, NB, DT, SVM, KNN gibi makine öğrenmesi algoritmaları ve CNN, GRU, MLP ve LSTM gibi derin öğrenme algoritmaları ayrı ayrı ve hibritler halinde 10 kat çapraz doğrulama ile test edilmiştir. En yüksek doğruluk oranını %98.8 ile 2 katmanlı CNN- LSTM hibrit çalışması vermiştir.

Alzaylaee ve diğ. (2020) dinamik analiz yoluyla zararlı Android uygulamalarını tespit etmek için bir derin öğrenme sistemi olan DL-Droid'i (Deep Learning Droid) önermişlerdir. Girdi olarak durum bilgisi kullanmışlardır. Çalışmada 30.000'den fazla uygulama (zararlı ve zararsız yazılım) ile gerçekleştirilen bir deney sunulmaktadır. DL-Droid'in makine öğrenimi tekniklerinden daha iyi performans göstermiştir. %97,8'e (yalnızca dinamik özelliklerle) ve %99,6'ya (dinamik + statik özelliklerle) kadar algılama oranı sağlayabildiği ortaya koyulmuştur. SVM, RF, NB, J48, Basit Lojistik (Simple Logistic, SL), Kısmi Karar Ağacı (Partial Decision Tree ,PART) ve DL-Droid algoritmaları kullanılmıştır. DL-Droid en yüksek doğruluk oranını vermiştir.

Wang ve diğ. (2018) derin otomatik kodlayıcı (Deep Auto Encoder, DAE) ve CNN'e dayalı hibrit bir modelle android zararlı yazılım algılama yöntemini önermiştir. 10.000 iyi huylu uygulama ve 13.000 zararlı uygulama üzerinde deneyler yapılmıştır. Seri CNN (CNN-S) mimarisinde, seyrekliği

artırmak için aktivasyon işlevi olarak doğrusal olmayan bir işlev olan Relu'yu ve aşırı öğrenmeyi önlemek için "dropout" kullanılmıştır. Eğitim süresini azaltmak için, CNN'nin ön eğitimi için derin otomatik kodlayıcı kullanılmıştır. Kombinasyon ile DAE ve CNN modelinin (DAE-CNN) kısa sürede daha esnek kalıpları öğrenebildiği gözlemlenmiştir. SVM ile karşılaştırıldığında, CNN-S modeli ile doğruluk %5 artarken, DAE-CNN modeli kullanılarak yapılan eğitim süresi CNN-S modeli ile karşılaştırıldığında %83 oranında azaltılmıştır.

Haq ve diğ. (2021) tarafından tasarlanan yaklaşımda, zararlı yazılımları verimli bir şekilde tanımlamak için CNN ve ten BiLSTM'den yararlanmışlardır. Önerilen teknik, kamuya açık veri kümeleri, standart performans ölçümleri ve hibrit DL tabanlı mimariler ile kapsamlı bir şekilde değerlendirilmiştir. Daha doğru tahmin için Classifier öznitelik değerlendirici, CFS alt küme değerlendirici, Rölyef öznitelik değerlendirici, Temel bileşenler analizi ve InfoGai gibi beş öznitelik çıkarma algoritması kullanılmıştır. CNN-BiLSTM, CNN-LSTM ve CNN-GRU hibritleri 10 kat çapraz doğrulama ile test edilmiştir. CNN-BiLSTM %99,47 ile en yüksek doğruluk oranını vermiştir.

Cho (2019) tarafından önerilen modelde zararlı yazılım verileri büyük olduğunda çok sayıda görüntü oluşturur ve küçük veriler için küçük bir görüntü oluşturur. Oluşturulan görüntü, dinamik RNN tarafından zaman serisi verileri olarak öğrenilmiştir. RNN'nin çıktı değeri, yalnızca en yüksek ağırlıklı çıktı kullanılarak ve CNN ile RNN çıktı değeri tekrar öğrenilerek zararlı yazılım olarak sınıflandırılır. Önerilen model üzerindeki deneyler, doğrulama veri kümesinde %92'lik bir Mikro ortalama F1 skoru ve %96'lık bir doğruluk oranı göstermiştir.

Bayazit ve diğ. (2022) tarafından önerilen sistemde RNN tabanlı LSTM, BiLSTM ve GRU algoritmaları, zararlı yazılım tespiti için 8115 statik özellik içeren CICInvesAndMal2019 veri kümesi üzerinde değerlendirilmektedir. Deneysel sonuçlar, BiLSTM modelinin, önerilen diğer RNN tabanlı derin öğrenme yöntemlerinden %98,85 doğruluk oranıyla daha iyi performans gösterdiğini göstermektedir.

Tablo 2.1 Literatür Özeti.

Sıra No.	Yıl ve Yazarlar	Kullanılan Algoritma	Sonuçlar
1	Schultz ve diğ. (2001)	RIPPER, NB ve sınıfladırma algoritmaları	DLL'ler, işlev çağrıları ve bu işlevlerin çağrılma sıklığı gibi öznitelikleri kullanılmıştır. NB veri başına en yüksek algılama oranını sağlamıştır.
2	Kolter ve diğ. (2004)	RF, BayesNet, MLP ve SVM	Veri özelliği olarak n-gram bayt kodları kullanılmıştır. 1651 zararlı ve 1971 iyi huylu yürütülebilir dosyanın her biri kodlanmıştır. NB, DT, SVM ve Boosting algoritmaları kullanılmıştır. Güçlendirilmiş DT için ROC eğrisinin altındaki alan 0,996 'dır ve diğer tüm yaklaşımlardan daha iyi sonuç vermiştir.
3	Tian ve diğ. (2009)	RF, NB, DT, IB1 ve SVM	1367 örnekten diziler (stringler) elde edilmiştir. KNN, istatistiksel algoritmalar ve AdaBoost dahil olmak üzere çeşitli sınıflandırma algoritmaları ile değerlendirilmiştir. Çapraz doğrulama kullanarak RF

			sınıflandırıcısı ile %97'lik bir doğruluk oranı elde edilmiştir.
4	Santos ve diğ. (2013)	KNN, DT, RF, SVM ve NB	KNN, DT, RF, SVM ve NB algoritmaları kullanılmıştır. Statik, Dinamik ve hibrit yaklaşımlar test edilmiştir. SVM: Normalleştirilmiş Polinom Çekirdek, her yaklaşım için en iyi doğruluk sonucunu elde etmiştir. (Statik: %95,90, Dinamik: %77,26, Hibrit: %96,60).
5	Aydoğan ve diğ. (2014)	IBL, J48, NB, SMO, MLP, BJ48, KT, RF	IBL, J48, NB, SMO, MLP, BJ48, KT, RF algoritmaları eğitim ve test için kullanılmıştır. En yüksek tespit oranı %96,2 ile Bj48 tarafından elde edilmiştir. En düşük FP oranı ise RF tarafından elde edilmiştir.
6	Bulut ve diğ. (2017)	YSA	Beş katmanlı yapay sinir ağı kullanılarak sınıflandırma işlemi gerçekleştirilmiştir. Test sonucunda %93,67 bir oran ile zararlı yazılım doğru tespit edilmiştir.
7	Wang ve diğ. (2018)	DAE, CNN ve SVM	DAE ve CNN modelinin (DAE-CNN) kısa sürede daha esnek kalıpları öğrenebildiği gözlemlenmiştir. SVM ile karşılaştırıldığında, CNN-S modeli ile doğruluk %5 artarken, DAE-CNN modeli kullanılarak yapılan eğitim süresi CNN-S modeli ile karşılaştırıldığında %83 oranında azaltılmıştır.
8	Algahtani ve diğ. (2019)	SVM, NB, Perceptron, J48, OneR, Deep Network	Kullanılan veri kümesi 2081 iyi huylu uygulama ve 91 kötü huylu uygulama içermektedir. OneR ve J48 algoritmaları, %0 yanlış pozitif oranıyla %100 doğruluk elde etmiştir.
9	Muhammad ve diğ. (2019)	LR, DT, RF, Adaboost, Bagging ve Gradyan sınıflandırıcı	Statik özellikler ve dinamik özellikler kullanılarak, veriler LR, DT, RF, Adaboost, Bagging sınıflandırıcı, ağaç sınıflandırıcı ve gradyan sınıflandırıcı ile test edilmiştir. Dinamik zararlı yazılım analizinin doğruluğu %94,64 iken statik analiz doğruluğu %99,36'dır.
10	Rafique ve diğ. (2019)	CNN, SVM ve MLP	CNN kullanılarak bayt dosyalarından özellikler çıkarılmıştır. SVM sınıflandırıcısı kullanılarak temel ve ayırt edici opcode özellikler seçilmiştir. Son olarak, MLP eğitim için kullanılmıştır. Doğrusal SVM en yüksek kayıp puanı değerini (0,8948), CNN ise en düşük kayıp puanı değerini vermiştir (0,1514).
11	Vinayakumar ve diğ. (2019)	LR, NB, DT, SVM KNN ile ve CNN, GRU, MLP ve LSTM	LR, NB, DT, SVM KNN ile ve CNN, GRU, MLP ve LSTM algoritmaları ayrı ayrı ve hibritler halinde 10 kat çapraz doğrulama ile test edilmiştir. En yüksek doğruluk oranını %98,8 ile 2 katmanlı CNN- LSTM hibrit çalışması vermiştir.
12	Yuxin ve diğ. (2019)	DBN, KNN, DT ve SVM	İşlem kodu dizileri ile temsil ederek, DBN ile tespit edilmesini amaçlanmıştır. DBN'lerin performansı KNN, DT ve SVM gibi üç temel sınıflandırma

			algoritması ile karşılaştırılmıştır. Sonuçlar, DBN modelinin temel modellerden daha doğru algılama ve tespit yaptığını göstermektedir.
13	Cho. (2019)	CNN, RNN	Oluşturulan görüntü, RNN tarafından zaman serisi verileri olarak öğrenilmiştir. CNN ile RNN çıktı değeri tekrar öğrenilmiştir. Önerilen model üzerindeki deneyler, doğrulama veri kümesinde %92'lik bir Mikro ortalama F1 skoru ve %96'lık bir doğruluk oranı göstermiştir.
14	Patil ve diğ. (2020)	DT, SGD, RF, SVC, LR, DNN, NB, KNN	Sistem çağrıları, opcodelar, bölümler ve bayt kodları ile özellik kümesi çıkarılmış, en yüksek doğruluk oranı DNN ile elde edilmiştir.
15	Tahtacı ve diğ. (2020)	DT, KNN, NB, LR, RF ve SVM	3000 adet Android Paket (APK) dosyasından opcode'lar elde edilmiştir. N-gram özellikleri kullanılarak DT, KNN, NB, LR, RF ve SVM makine öğrenmesi yöntemleri ile en yüksek test skoruna 3-gram için özellik sayısı 2'ye düşürüldükten sonra ulaşılmıştır (KNN ve SVM modelleri %100'lük bir test skoru elde etmiştir.).
16	Alzaylaee ve diğ. (2020)	SVM, RF, NB, J48, SL, PART ve DL-Droid	Android uygulamalarını tespit etmek için bir derin öğrenme sistemi olan DL-Droid ve makine öğrenmesi algoritmaları test edilmiştir. DL-Droid en yüksek doğruluk oranını vermiştir.
17	Harshalatha ve diğ. (2020)	RF, BayesNet, MLP ve SVM	10 kat çapraz doğrulama altında test edilmiş ve sınıflandırıcı sonuçları, RF'nin %98 doğruluk oranı ile en iyi performansa sahip olduğunu göstermiştir.
18	Azeez ve diğ. (2021)	RF, NB, DT, GB ve Adaboost	Toplu öğrenmeye dayalı bir yöntem önermektedir. CNN sınıflandırıcısı uygulandıktan sonra son aşama için çeşitli makine öğrenmesi algoritmaları uygulanmıştır. Karşılaştırma için ise RF, NB, DT, GB ve Adaboost algoritmaları kullanılmıştır.
19	Haq ve diğ., (2021)	CNN, BiLSTM, LSTM ve GRU	Öznitelik çıkarma algoritmaları ile birlikte CNN-BiLSTM, CNN-LSTM ve CNN-GRU hibritleri 10 kat çapraz doğrulama ile test edilmiştir. CNN-BiLSTM %99,47 ile en yüksek doğruluk oranını vermiştir.
20	Shatnawi ve diğ. (2022)	SVM, KNN ve NB	Android izinleri ve API çağrıları kullanılmıştır. SVM, KNN ve NB gibi iyi bilinen üç Makine Öğrenimi algoritması ile test edilmiştir.
21	Bayazit ve diğ. (2022)	LSTM, BiLSTM ve GRU	Deneysel sonuçlar, BiLSTM modelinin, önerilen diğer RNN tabanlı derin öğrenme yöntemlerinden %98,85 doğruluk oranıyla daha iyi performans gösterdiğini göstermektedir.

3. MATERYAL VE YÖNTEM

Bu bölümde zararlı yazılım veri kümesinin ayrıntıları ve zararlı yazılım tespiti için kullanılan algoritmaları sunulmaktadır.

3.1. Materyaller

3.1.1. Veri Kümesi

Bu çalışmada C-Prot Siber Güvenlik Teknolojileri San. ve Tic. A.Ş. tarafından temin edilen zararlı ve zararsız yazılımların özelliklerinden oluşan veri kümesi kullanılmaktadır. Veri kümesi tablo 3.1'de gösterildiği gibi, 1000'i zararlı ve 1000'i zararsız yazılım olmak üzere zararlı veya zararsız olarak etiketlenmiş 2000 yazılım örneğinden oluşmaktadır.

Tablo 3.1. Veri kümesinde bulunan veri sayısı.

Sınıf	Adet
Zararlı Yazılım	1000
Zararsız Yazılım	1000
Zararlı + Zararsız Yazılım	2000

Bir yazılımın zararlı yazılım olduğu bilgisi “Zararlı yazılım”, zararsız yazılım olduğu bilgisi “Zararsız yazılım” olarak belirlenmiştir ve Tablo 3.2’de bulunan sayısal değerler ile eşleştirilmiştir.

Tablo 3.2. Zararlı/zararsız yazılım bilgisinin veri kümesinde sayısal karşılığı.

Sınıf	Veri Kümesi Sayısal Karşılığı
Zararlı Yazılım	1
Zararsız Yazılım	0

PE, bir Windows yürütülebilir dosya biçimidir. PE dosya formatı, Windows yürütülebilir dosyaları ve DLL dosyaları tarafından kullanılabilmektedir. PE dosyaları başlık bilgisi ile başlamaktadır ve bu başlık bilgisi programın gerektirdiği kütüphane fonksiyonlarının çeşitleri ve uygulama tipleri gibi bilgileri içermektedir. İletilen dosyaların zararlı/temiz tasnifi C-Prot Antimalware Engine kullanılarak elde edilmiştir. İletilen dosyaların tamamının PE dosya formatında olmasına önem gösterilmiştir bu sayede yapılacak işlemin daha tutarlı olması amaçlanmıştır. Dosyalar tasnif edildikten sonra yazılan harici bir uygulama ile tüm dosyaların kullandığı kütüphaneler ve bu kütüphanelerde kullanılan fonksiyonlar tespit edilmiştir. Bu tespit yapılırken PE dosya formatı incelenmiş dosyada bulunan bölümler ayrıştırılıp ilgili işlemler yapılmıştır. Sadece PE

formatına sahip olan dosyalar incelendiği için yapı Windows platformlarına etki eden zararlı yazılımların tespit için kullanılacaktır.

Tablo 3.3'de veri kümesinin özellikleri ve açıklamaları verilmiştir. Dosya boyutu, yazılımın dijital imza durumu, yazılımda kullanılan kütüphaneler ve fonksiyonların tümü veri kümesi özelliği olarak kullanılmıştır.

Tablo 3.3. Veri kümesinde bulanan özellikler ve açıklamaları

Öznitelik	Açıklama
Boyut	Dosyanın boyutu (Tamsayı)
Dijital İmza	Yazılımın dijital imza bilgisi (0-1)
Fonksiyonlar	Yazılım içerisinde kullanılan fonksiyonlar
Kütüphaneler	Yazılımda bulunan kütüphane ve DLL'ler
Sınıf	Yazılımın ait olduğu sınıf (0-1)

Veri kümesindeki her bir satır için modelin makine öğreniminde ve derin öğrenmede daha verimli çalışabilmesi için kategorik girdilerin sayısal terimlere dönüşümü sağlanmıştır. Veri kümesindeki tüm özellikler "0" veya "1" olarak ikili bir sayısal değere dönüştürülmüştür. Veri kümesinin normalizasyon işleminden önceki bir örneği Tablo 3.4'de verilmiştir. Sınıf etiketi içerisinde bulunan "1" etiketi yazılımın zararlı yazılım olduğunu "0" ise yazılımın zararsız yazılım olduğunu belirtmektedir.

Tablo 3.4. Veri işleme işlemi öncesi veri kümesi örneği

Boyut	Dijital İmza	Kütüphaneler	Fonksiyonlar	Sınıf
323800	1	KERNEL32.dll,USER32.d...	GetSystemTime,PostMessage,Free…	1
6595084	0	MSVCR100.dll,IMM32.dl...	MD5,SHA,SHA1,HMAC,GetSe...	1
139084	0	lib1.dll,bcrypt.dll,KERNEL3...	BCryptOpenAlgorithmProvider,BC...	0
119160	0	kernel32.dll,user32.dll	ExitProcess,VirtualAlloc,VirtualFr…	1
560906	0	uxtheme.dll,gdi32.dll,kern...	CreateWindow,CreateWindowEx,…	1
162733	0	zlib1.dll,bcrypt.dll,KERN…	BCryptOpenAlgorithmProvider,BC…	0
511552	1	VCRUNTIME140.dll,KER…	SHA,InitializeConditionVariable…	0

Veri kümesindeki özellikler arasında bulunan fonksiyonlar ve kütüphaneler (özellikler) sütunundaki fonksiyon ve kütüphane adları sütunlar halinde düzenlenmiş, böylece her bir fonksiyon ve kütüphane adı aynı zamanda bir veri kümesi özelliği olarak kabul edilmiştir. Bu işlem sonucunda 195 benzersiz kütüphane ve 4.327 benzersiz fonksiyon elde edilmiştir. Dosya boyutu, dijital imza, sınıflandırma etiketi, fonksiyonlar ve kütüphaneler de dahil olmak üzere oluşturulan veri kümesinde toplam 4.545 özellik (sütun) kullanılmıştır. Bu işlemlerin ardından veri kümesinin son hali tablo 3.5'te

gösterilmiştir. Her bir yazılım için, içerisinde özellikler (sütunlarda belirtilen fonksiyon, kütüphane gibi özellikler) aranarak var ise "1" yok ise "0" değeri atanmıştır. Örneğin bir kütüphane özelliği için, o kütüphane yazılım içerisinde bulunuyorsa "1" bulunmuyorsa "0" değeri atanmıştır. Sayısal değerlere dönüştürülmüş bir veri kümesi örneği tablo 3.4'de verilmiştir.

Tablo 3.5. Veri işleme işlemi sonrası sayısal verilere dönüştürülmüş veri kümesi örneği

Boyut	Dijital İmza	Sınıf	.ctor	...	KERNEL32.dll	ExitProcess
323800	1	1	0		1	0
6595084	0	1	1		0	0
139084	0	0	0		0	0
119160	0	1	0		0	1
560906	0	1	0		0	0
162733	0	0	0		1	0
511552	1	0	0		1	0

Minimum-maksimum normalleştirme olarak da bilinen min-max normalizasyon bir özelliğin değerlerini belirli bir aralığa yeniden ölçeklendirmek için kullanılan bir veri normalleştirme tekniğidir. Veri ön işleme dahil olmak üzere çeşitli makine öğrenimi ve derin öğrenmede yaygın olarak kullanılmaktadır. min-max normalizasyonun amacı özellik değerlerini yeni bir aralığa tipik olarak 0 ile 1 arasında eşlenecek şekilde dönüştürmektir. Model 0 ile 1 arasında çıktılar veriyorsa, bu değerleri 0 veya 1 gibi belirli sınıf etiketleriyle eşlemek için min-max normalizasyon kullanmak yaygın bir uygulamadır. Bu genellikle ikili sınıflandırma gerçekleştirirken yapılabilmektedir. Çalışmada, derin öğrenme modellerinden elde edilen ilk çıktıların 0 ve 1 arasında olması sebebiyle min-max normalizasyon kullanılarak veriler 0 veya 1 olarak dönüştürülmüştür. Min-max normalizasyon formülü (1) aşağıdaki gibidir. $\boldsymbol{x}$ girdi değerini, $\boldsymbol{x_{max}}$ girdideki en büyük sayı değerini ve $\boldsymbol{x_{min}}$ girdideki en küçük sayı değerini temsil etmektedir.

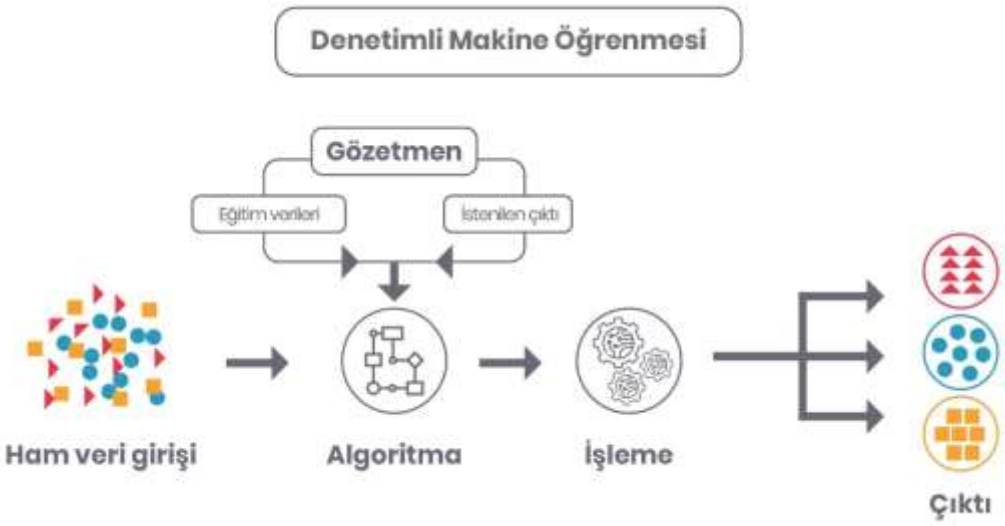
$$\boldsymbol{x_{yeni}} = \frac{\boldsymbol{x - x_{min}}}{\boldsymbol{x_{max} - x_{min}}} \quad (1)$$

Elde edilen veri kümesi makinenin eğitilmesi esnasında veri kümesi %80 eğitim, %20 test verisi olarak ayrılmış ve 10 kat çapraz doğrulama ile test edilmiştir. K-katlı çapraz doğrulama, genellikle sınıflandırma algoritmalarının performansını değerlendirmek için kullanılır. İlk olarak, bir veri seti eşit sayıda örnekle K ayrık kıvrıma bölünür. Diğer K-1 katlarından türetilen model daha sonra sırayla her kat tarafından test edilir (Wong, 2020). Katlamalardan elde edilen on sonucun daha sonra tek bir tahmin oluşturmak için ortalaması alınabilir.

3.2. Yöntemler

3.2.1. Makine Öğrenmesi

Yapay zekâ, insan zekâsına benzetilerek çeşitli görevleri yerine getiren ve sürekli olarak kendini geliştiren sistemler veya makinelerdir. 1950'li yıllarda ortaya çıkan yapay zekâ sistemleri, hatalardan öğrenebilen sistemler olduğu için sürekli iyileştirme göstermektedir. Makine öğrenimi, bilgisayarların açıkça programlanmadan öğrenmesine ve tahminler veya kararlar vermesine olanak tanıyan algoritmaların ve modellerin geliştirilmesine odaklanan yapay zekanın bir alt alanıdır. Bunun yerine, makine öğrenimi algoritmaları kalıpları tanımlamak, anlamlı içgörüler çıkarmak ve doğru tahminler veya sınıflandırmalar yapmak için büyük miktarda veri üzerinde eğitilir. Makine öğrenmesi algoritmalarında iki tür öğrenme biçimi vardır: denetimli ve denetimsiz öğrenme. Denetimsiz öğrenme, algoritmanın etiketlenmemiş veriler üzerinde eğitildiği bir makine öğrenimi türüdür. Denetimsiz öğrenmede, algoritma önceden tanımlanmış herhangi bir etiket veya hedef olmaksızın verilerdeki kalıpları, yapıları veya ilişkileri bulmaya çalışır. Amaç, verilerdeki doğal kalıpları veya gruplamaları keşfetmektir. Denetimli öğrenme ise algoritmanın etiketli veriler üzerinde eğitildiği bir makine öğrenimi türüdür. Etiketli veriler, her girdi veri noktasının karşılık gelen bir hedef veya çıktı değeri ile ilişkilendirildiği anlamına gelir. Denetimli öğrenmenin amacı, girdi verileri ile karşılık gelen çıktı etiketleri arasında bir eşleme öğrenmek ve algoritmanın yeni, görünmeyen veriler üzerinde tahminler veya sınıflandırmalar yapmasına izin vermektir. Denetimli öğrenme yapısı Şekil 3.1'de gösterilmiştir.



Şekil 3.1. Denetimli Öğrenme yapısı

Makine öğrenimi, doğrulukta artan iyileştirmelerle, insanların nasıl öğrendiğini taklit etmek için veri ve algoritmaları kullanmaya odaklanan bir yapay zekâ ve bilgisayar bilimi dalıdır. Derin öğrenme

ise, sonuçları tahmin etmek için belirli bir veri kümesini kullanan çok katmanlı bir makine öğrenimi tekniğidir.

Zararlı yazılımların analizi ve tespiti için çalışmada kullanılan makine öğrenmesi ve derin öğrenme yöntemlerinin detaylı açıklaması aşağıda sunulmuştur.

3.2.1.1. Gaussian Naive Bayes Algoritması

Naive Bayes algoritması Bayes Teoremi'e dayalı olasılıksal sınıflandırma mekanizmasıdır. Bu sınıflandırma perspektifinde temel amaç, belirli bir problem alanı içindeki bir dizi yeni veri ile bir dizi sınıflandırma arasındaki en iyi eşlemeyi bulmaktır (Yang, 2018). Bir Naive Bayes sınıflandırıcısı, bir sınıftaki belirli bir özelliğin varlığının başka herhangi bir özelliğin varlığıyla ilgisi olmadığını varsayar (Mahesh, 2020). Bayes teoremi $\boldsymbol{f_1}$ ve $\boldsymbol{f_n}$ sınıf değişkenine ve bağımlı özellik vektörüne göre (2)'deki ilişkiyi belirtir. *C* verilen hedef ve *f* özellikleri temsil eder.

$$p(C|\boldsymbol{f_1}, \dots \boldsymbol{f_n}) = \frac{P(C)p(\boldsymbol{f_1}, \dots \boldsymbol{f_n}|C)}{p(\boldsymbol{f_1}, \dots \boldsymbol{f_n})} \tag{2}$$

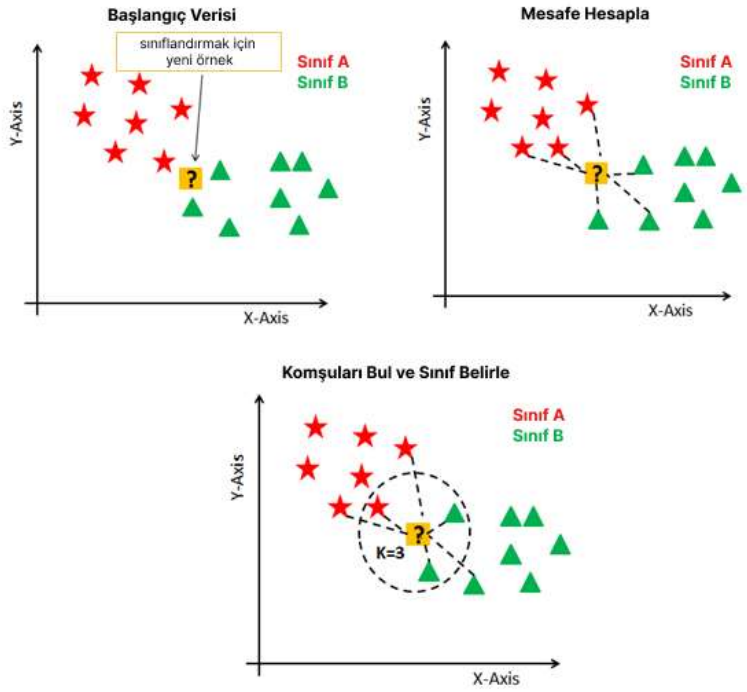
Gauss dağılımı, basit Bayes sınıflandırmasında sürekli özellikleri kullanma yöntemidir. Özelliğin sürekli değerleri varsa, bu değerlerin bir Gauss dağılımından veya normal dağılımdan geldiği varsayılmaktadır.

$$\boldsymbol{p(x_i \mid y)} = \frac{\mathbf{1}}{\sqrt{\mathbf{2}\pi\boldsymbol{\sigma_y^2}}}\ \boldsymbol{exp}\left(-\frac{(x_i - \boldsymbol{\mu_y})^2}{\mathbf{2}\boldsymbol{\sigma_y^2}}\right) \tag{3}$$

(3)'te gösterilen $\boldsymbol{\sigma_y}$ ve $\boldsymbol{\mu_y}$ parametreleri maksimum olasılık kullanılarak tahmin edilir.

3.2.1.2. K- En Yakın Komşu Algoritması (K-Nearest Neighbour, KNN)

KNN algoritması denetimli öğrenme tekniğine dayalı hem sınıflandırma hem de regresyon için kullanılan Makine Öğrenimi algoritmalarından biridir. Yeni durum (veriler) ile mevcut durumlar arasındaki benzerliği gözetir ve yeni durumu mevcut sınıflara en çok benzeyen sınıfa koyar. Test verileri ile tüm eğitim noktaları arasındaki mesafeyi hesaplayarak test verileri için doğru sınıfı tahmin etmeye çalışır. Ardından test verilerine yakın olan K nokta sayısını seçer. K adet eğitim verisinin sınıflarına ait test verilerinin olasılığını hesaplar ve en yüksek olasılığa sahip sınıf seçilir. Regresyon durumunda, seçilen K değeri eğitim noktalarının ortalamasıdır. Örnek bir KNN yapısı Şekil 3.2'de gösterilmiştir.



Şekil 3.2. KNN yapısı (Zhu, 2021).

En yakın komşuları bulmak için farklı mesafe ölçüm yöntemleri kullanılmaktadır (Chumachenko, 2017). Popüler olanlar arasında Hamming, Manhattan, Minkowski ve Euclidean mesafeleri bulunur.

Hamming Mesafesi Formülü (4)'te belirtilmiştir.

$$\boldsymbol{d_{ij} = \sum_{k=1}^{p} |x_{ik} - x_{jk}|} \tag{4}$$

Manhattan Mesafesi Formülü (5)'te belirtilmiştir.

$$\boldsymbol{d_1(p,q) = ||p - q||_1 = \sum_{i=1}^{n} |p_i - q_i|} \tag{5}$$

Minkowski Mesafesi Formülü (6)'te belirtilmiştir.

$$\boldsymbol{(\sum_{i=1}^{n} |x_i - y_i|^p)^{1/p}} \tag{6}$$

Euclidean Mesafesi Formülü (7)'da belirtilmiştir.

$$\boldsymbol{d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}} \tag{7}$$

KNN algoritması sözde kodu Algoritma-1'de verilmiştir.

Algoritma 1: KNN Algoritması Sözde Kodu (Lopez-Bernal, 2021)

Giriş: *x, l, s // x: eğitim veri kümesi, l: sınıf, s:test veri kümesi*

for (*i* ← *0* **to** *eğitim veri boyutu* **do**)

Mesafeyi hesapla d(xi, s)

end for

En yakın komşuların istenen k sayısını seç

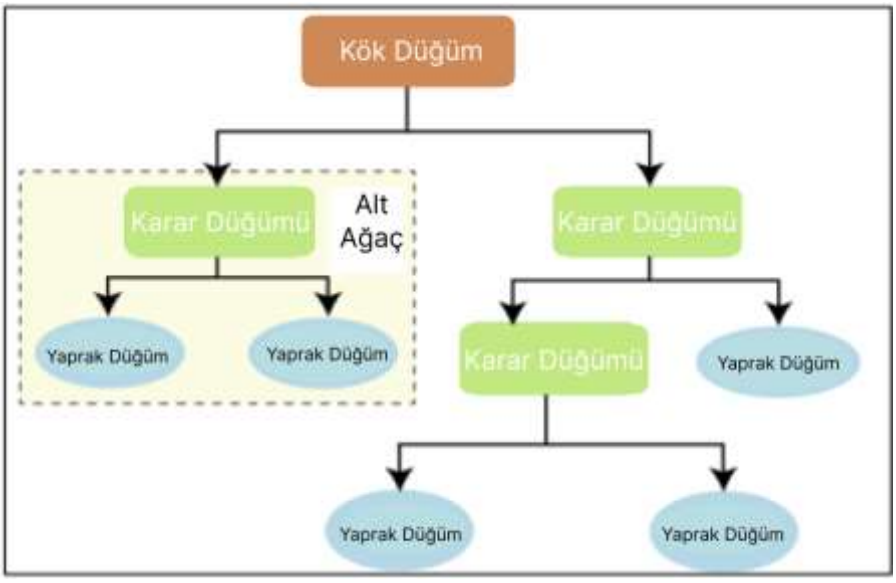
Mesafeleri artan düzene göre sırala

Her bir sınıfın en üstteki k komşudaki oluşum sayısını say

Çıktı: *s'ye en sık kullanılan l sınıfını ata*

3.2.1.3. Karar Ağaçları Algoritması (Decision Tree, DT)

DT algoritması hem sınıflandırma hem de regresyon görevleri için kullanılan popüler bir denetimli öğrenme algoritmasıdır. Girdi verilerinin özelliklerine dayalı olarak ağaç benzeri bir karar modeli ve olası sonuçları oluşturur. Kök düğüm, dallar, iç düğümler ve yaprak düğümlerden oluşan hiyerarşik bir ağaç yapısına sahiptir. Karar ağacı yinelemeli bir yapıdadır ve adından da anlaşılacağı üzere bir ağaç yapısı kullanır. Tek bir düğüm ile başlar ve yeni sonuçlara ulaştıkça dallara ayrılarak bir ağaç yapısı oluştur. Algoritma çalıştığında girilen değer düğümlerde ilerletilerek sonuç elde edilir. Örnek bir karar ağacı Şekil 3.4'de gösterilmiştir.



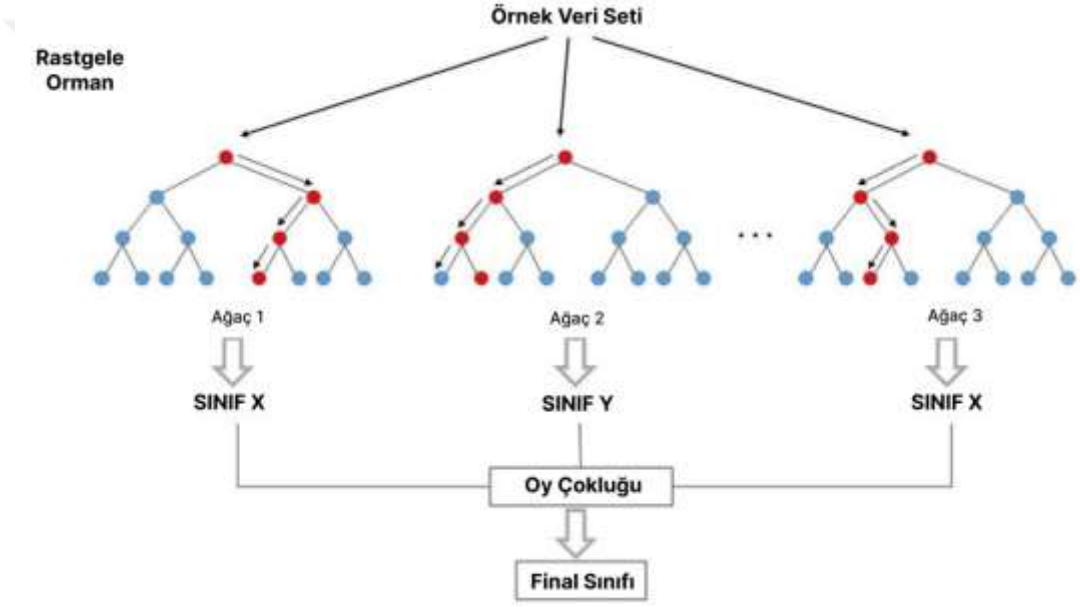
Şekil 3.3. DT yapısı (Abdulazeez, 2021).

Karar Ağaçları eğitilirken farklı CART (Sınıflandırma ve Regresyon Ağacı), ID3 (Tekrarlı İkililikçi Ağacı), C4.5 ve C5.0 (güçlü bir yaklaşımın farklı sürümleri), CHAID (Ki-kare Otomatik Etkileşim Tespiti) gibi algoritmalar kullanılır.

3.2.1.4. Rastgele Orman Algoritması (Random Forest, RF)

RF algoritması denetimli öğrenme tekniğine dayalı olukça popüler bir makine öğrenmesi algoritmasıdır. Hem sınıflandırma hem de regresyon problemleri için kullanılabilir. RF, verilen veri kümesinin çeşitli alt kümelerinde bir dizi karar ağacı içeren ve bu veri kümesinin tahmin doğruluğunu iyileştirmek için bu ağaçların ortalamasını alan bir sınıflandırıcıdır. Örnek yapı Şekil 3.4'te gösterilmektedir.

RF tek bir karar ağacına güvenmek yerine, her ağaçtan tahmini alır ve tahminlerin çoğunluk oylarına dayanarak nihai sonucu tahmin eder. Ormandaki daha fazla ağaç sayısı, daha yüksek doğruluk sağlar ve fazla uyum sorununu (overfitting) önler.



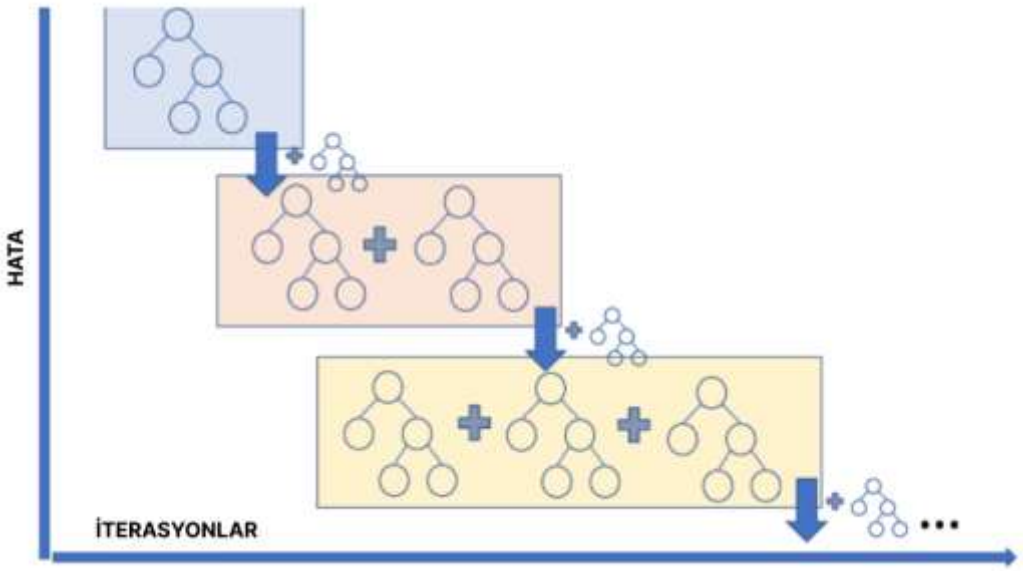
Şekil 3.4. RF yapısı (Dimitriadis, 2018).

Rastgele orman algoritması ile karar ağacı algoritması arasındaki fark, kök düğümü bulma ve rastgele ormandaki düğümleri ayırma işleminin rastgele yapılmasıdır. Tüm veri kümesini bir karar ağacı algoritmasıyla beslemek yerine parçalara bölerek daha fazla sayıda karar ağacı algoritmasıyla besleyerek rastgele orman oluşturulabilir.

3.2.1.5. Gradyan Arttırma Algoritması (Gradient Boosting, GB)

Güçlendirme (Boosting) algoritması zayıf öğrenenleri güçlü öğrenenlere dönüştürme yöntemidir. Bu dönüştürme işlemi yinelemelerle aşamalı olarak gerçekleştirilir. Güçlendirme algoritmaları genellikle zayıf öğrencilerin eksikliklerini nasıl algıladıklarına göre farklılık gösterir.

Bu yöntem gradyan descent ve boosting kelimelerinin birleştirilmesiyle oluşturulmuştur ve karar ağacı sonuçlarını iyileştirmek için gradyan descent algoritmasını kullanır. Veri kümesini rastgele ormandaki gibi çoklu kısmi veri setlerine bölmek yerine veri kümesini olduğu gibi kullanarak ve hatalara dayalı olarak yeni bir karar ağacı oluşturur. Bu şekilde yüzlerce ardışık karar ağacı elde edilir. Örnek yapı Şekil 3.5'te gösterilmiştir.



Şekil 3.5. Gradient Boosting yapısı (Baturnyska, 2021).

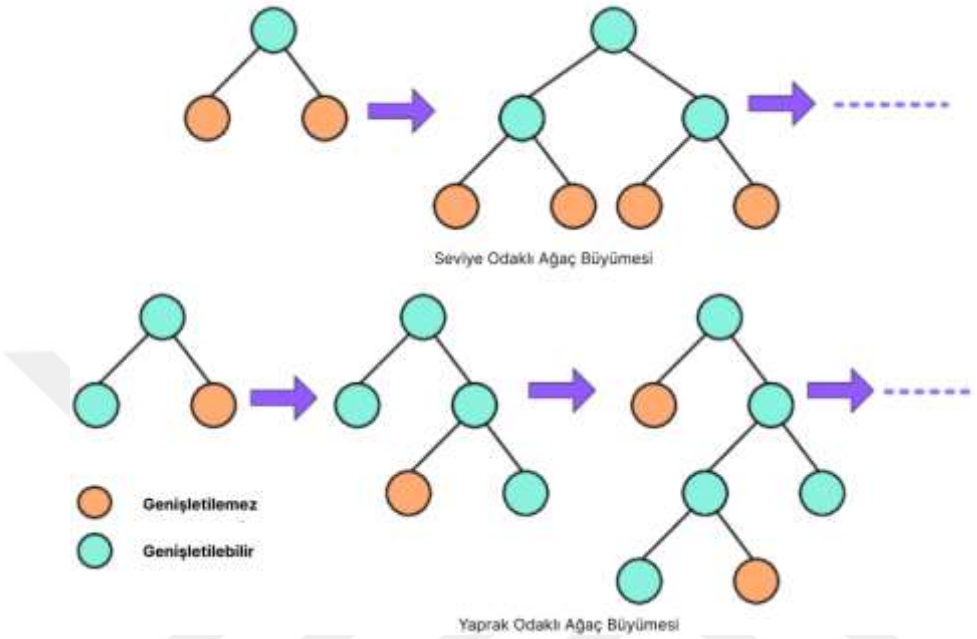
Her aşama hatayı en aza indirmek amacıyla eğitilir. Böylece her yinelemede gerçek değer ve tahmin değerinin hatası hesaplanır. Ancak aynı anda, aşırı öğrenme sorununun da kontrol edilmesi gerekir. Her yinelemede tahmin hatasını azaltmak için bir adım ilerlenir. Bu algoritmaya tahmin için hataların gradyanının alınması nedeniyle gradyan artırma (gradient boosting) adı verilir. İterasyon sayısı arttıkça hata oranı en aza indirilir.

3.2.1.6. LightGBM Algoritması

LightGBM algoritması histogram tabanlı bir algoritmadır. Sürekli değerli değişkenleri ayrıklaştırarak hesaplama yükünü azaltır. Karar ağacı eğitim süresi bölme sayısı ile doğru orantılıdır. Bu yöntem sayesinde eğitim süresi kısaltılır ve kaynak tüketimi azaltılır.

Karar ağaçlarında öğrenme için iki strateji kullanılabilir: seviye odaklı (depth-wise or level-wise) veya yaprak odaklı (leaf-wise). Seviye odaklı bir strateji ağacı büyürken dengede tutar. Öte yandan, yaprak odaklı bir strateji, kaybı az olan yapraklar ile yaprakları bölmeye ve kayıpları azaltmaya devam edecektir. Bu özellik LightGBM'yi diğer boosting algoritmalarından farklı kılar. Yaprak odaklı bir stratej, daha düşük bir model hata oranı ve daha hızlı öğrenme ile sonuçlanır. Bununla birlikte, yaprak odaklı büyüme stratejisi veri sayısı az olduğunda modeli aşırı öğrenmeye meyilli hale getirir. Bu nedenle

bu algoritma büyük veri ile kullanım için daha uygundur. Ek olarak ağaç derinliği ve yaprak sayısı gibi parametreler aşırı öğrenmeyi önlemek için optimize edilebilir. LightGBM yaprak odaklı büyüme stratejisini kullanırken XGboost algoritması seviye odaklı büyüme stratejisini kullanır. Şekil 3.6'da yaprak odaklı ağaç büyümesi yapısını görülmektedir.



Şekil 3.6. LightGBM Yaprak Odaklı Ağaç Büyümesi yapısı (Dong, 2022).

LightGBM diğer algoritmalardan farklı olarak iki teknik kullanır. Bu teknikler Gradyan Tabanlı Tek Yönlü Örnekleme (Gradient based One Side Sampling, GOSS) ve Özel Değişken Paketi'dir (Exclusive Feature Bundling, EFB). GOSS, karar ağaçlarının doğruluğunu korurken veri miktarını azaltmayı amaçlar. Geleneksel gradyan artırma tüm veri örneklerini tararken ve her özellik için bilgi kazancını hesaplarken, GOSS yalnızca önemli verileri kullanır. Bu nedenle veri dağılımını önemli ölçüde etkilemeden veri miktarı azaltılır.

EFB'ler doğruluktan ödün vermeden değişken sayısını azaltmayı ve buna göre model eğitiminin verimliliğini artırmayı amaçlar. EFB'nin iki işlem adımı vardır. Bu adımlar paketler oluşturur ve değişkenleri aynı pakette birleştirir. EFB daha yoğun özellikler oluşturmak için seyrek özellikleri birleştirir. Bu durum karmaşıklığı azaltır ve daha az bellek tüketimi ile eğitim sürecini hızlandırır.

3.2.1.7. XGBoost Algoritması (eXtreme Gradient Boosting, XGB)

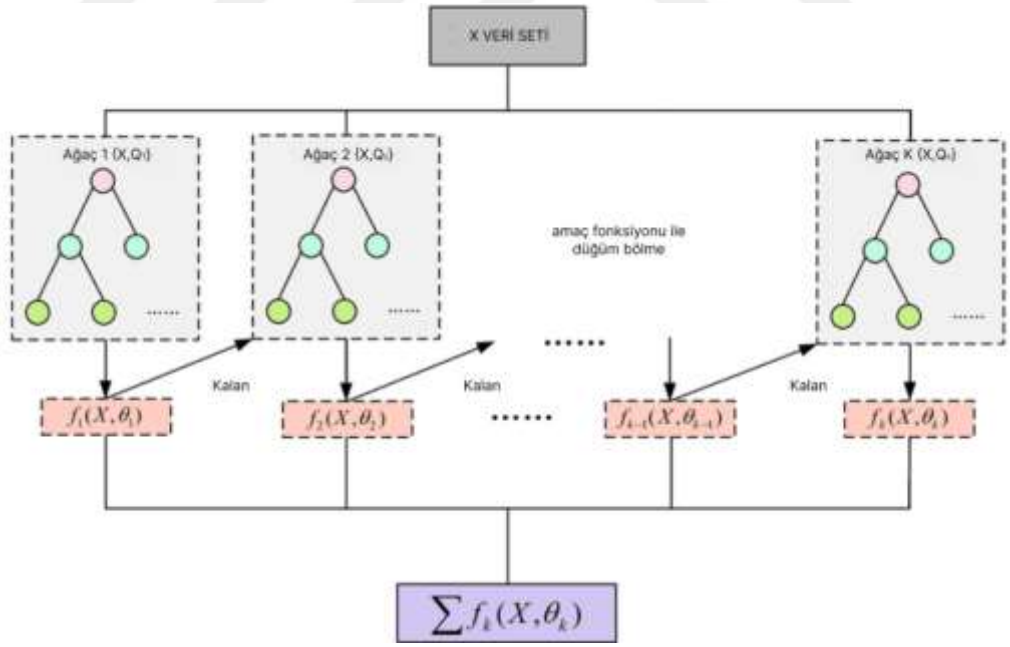
XGB algoritması gradyan destekli bir karar ağacı uygulaması olarak önerilmiştir. Algoritma bir karar ağacı oluşturur. XGB'de önemli bir rol oynayan ağırlıklar tüm bağımsız değişkenlere atanır ve sonuçları tahmin eden karar ağaçlarına verilir. Ağacın yanlış tahmin ettiği değişkenler daha fazla

ağırlıklandırılır ve bu değişkenler ikinci karar ağacına verilir. Bu bireysel sınıflandırıcılar/tahminciler sonrasında daha güçlü ve doğru bir model üretmek için birleştirilir.

Gradient Boosting ve XGB aynı mantık ile çalışmaktadır ancak aralarında bazı farklar bulunmaktadır. XGB farklı yöntemler kullanarak daha yüksek tahmin başarısı elde eder ve büyük veri setleri için optimize edilmiştir. Gradient Boosting'ten farklılaşmasını sağlayan başlıca konular şu şekildedir;

- Eğitim boyunca tüm CPU çekirdeklerini kullanarak ağaç yapısını paralelleştirir.
- Bir makine kümesi kullanarak çok büyük modelleri eğitmek için dağıtılmış hesaplama kullanılır.
- Belleğe sığmayan çok büyük veri kümeleri için çekirdek dışı bilgi işlem hesaplama yapar.
- Donanımdan daha fazla yararlanmak için veri yapılarının ve algoritmanın önbellek optimizasyonundan yararlanır.

Şekil 3.7'de gösterildiği gibi her XGB yinelemesi sırasında önceki tahmin ediciler kalanlarla (residuals) düzeltilir. Algoritma model değerlendirmesi için kullanılan kayıp fonksiyonlarının türlerini bağımsız olarak belirleyebilir. Aşırı uyum riskini azaltmak için modele ek bir düzenleme terimi eklenir.

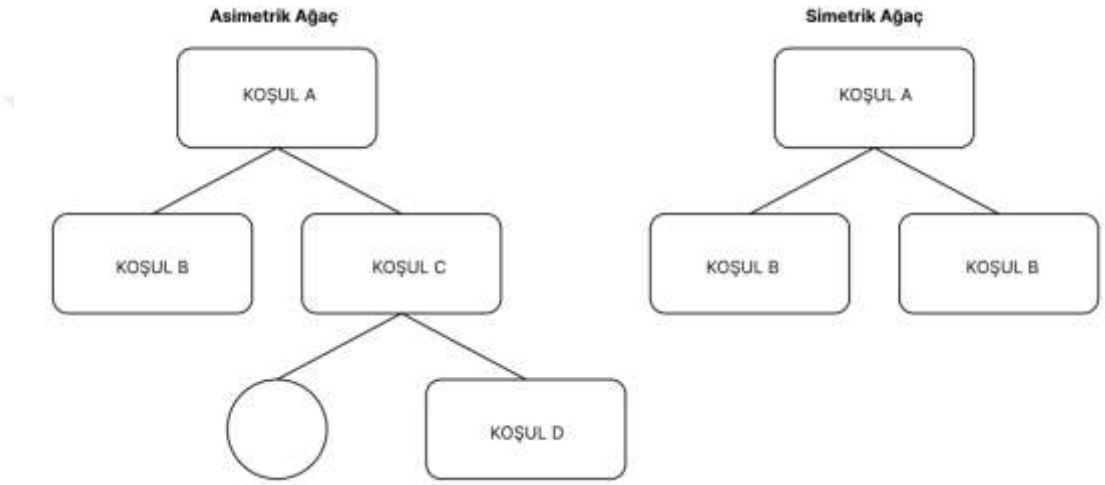


Şekil 3.7. XGB yapısı (Guo, 2020).

3.2.1.8. Kategorik Yükseltme Algoritması (Categorical Boosting, Catboost)

Catboost algoritması Yandex tarafından geliştirilen sıralama, öneri sistemleri, tahmin ve kişisel asistanlarda kullanılabilen açık kaynak kodlu bir algoritmadır. CatBoost, özellikle "kategorik

değişkenler" içeren verilerle iyi sonuç verir. Yüksek öğrenme hızına sahip olması sayısal, kategorik ve metin verilerinin işlenebilmesi, GPU desteği ve görselleştirme seçenekleri onu diğer algoritmalardan ayıran başlıca özelliklerdir. CatBoost, XGBoost ve LightGBM'den farklı olarak simetrik (dengeli) ağaçlar oluşturur. Şekil 3.8'de Catboost algoritmasının ağaç yapısı gösterilmiştir. Her adımda önceki ağaçtan gelen yapraklar aynı koşul kullanılarak bölünür. En düşük kaybı açıklayan özellik bölme çifti seçilir ve tüm seviye düğümleri için kullanılır. Bu dengeli ağaç mimarisi verimli CPU uygulamasına yardımcı olur, tahmin süresini azaltır, hızlı model uygulayıcıları yapar ve yapı düzenlileştirme işlevi görürken fazla uydurmayı kontrol eder.

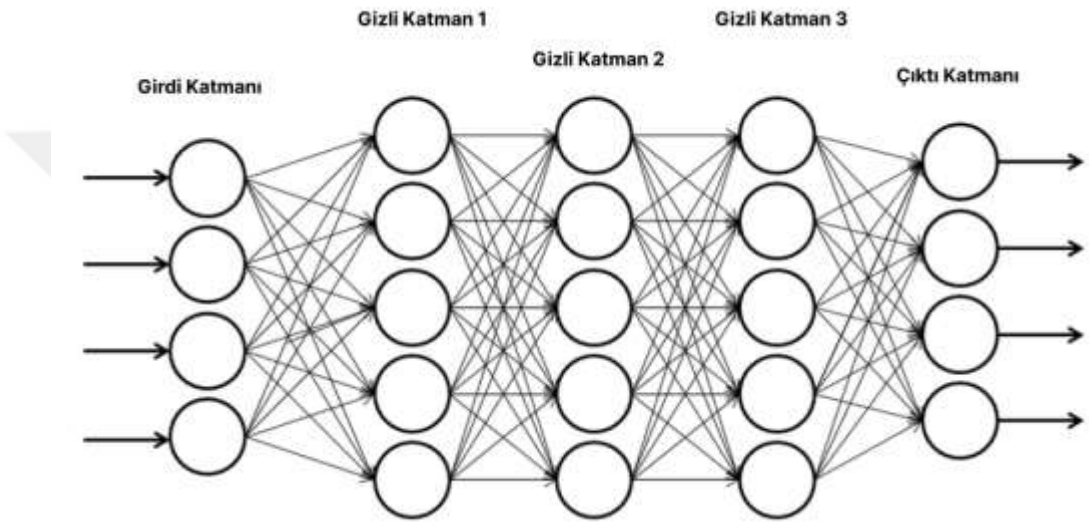


Şekil 3.8. Catboost Simetrik Ağaç yapısı

3.2.2. Derin Öğrenme

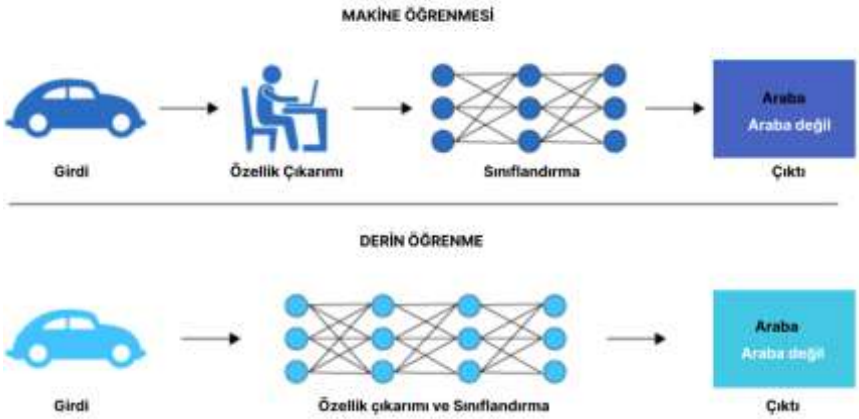
Derin öğrenme insan beyninin problemler için analiz etme, gözlemleme, karar verme ve öğrenme gibi yeteneklerini taklit edebilen ve denetimli veya denetimsiz olarak sınıflandırma, özellik çıkarma, dönüştürme gibi işlemleri verilerden yararlanarak yapabilen bir tekniktir. 2010 yılı itibariyle yaygın bir şekilde kullanılmaya başlanan derin öğrenme büyük veriler ile tek bir katman yerine birçok katmanda makine öğreniminde kullanılan hesaplamaları tek bir seferde yapabilen, tanımlanması gereken parametreleri keşfedebilen, hatta daha iyi parametreler ile değerlendirmelerde bulunabilen bir sistemdir. Derin öğrenme özellik çıkarma ve dönüştürme için birçok doğrusal olmayan işlem birimi katmanını kullanır. Sonraki her katman bir önceki katmanın çıktısını girdi olarak alır. Geleneksel makine öğrenimi algoritmaları doğrusaldır, ancak derin öğrenme algoritmaları uygulama alanının karmaşıklığına bağlı olarak farklı hiyerarşik modellere sahiptir. Elde edilen başarı oranı belli bir seviyeye ulaşana kadar derin öğrenme süreci tekrarlanır.

Derin öğrenmede özellikleri en düşük seviyeden en üst seviyeye yükseltmek için çoklu katmanlar eklenir. Her seviye belirli bir özellik tipini tanımlar ve onu bir sonraki seviyeye iletir. Bir sonraki düzeyde bu alt düzey işlevler daha üst düzey işlevler oluşturmak için birleştirilir. Bu özellikler, iletilirken bir tür kümeleme yapılır ve sonunda modeldeki son katman sınıflandırma işlemini gerçekleştirilir (Patil, 2020). Derin öğrenmede giriş ve çıkış katmanları arasında birçok katman vardır. Bir katman yüzlerce veya binlerce sinir birimi içerebilir. Girdi ve çıktı katmanları arasında bulunan yapı gizli katmanlar ve düğümler olarak adlandırılır. Derin öğrenme katman yapısı Şekil 3.9'da gösterilmiştir.



Şekil 3.9. Derin öğrenme katman yapısı (Miralles-Pechuan, 2017).

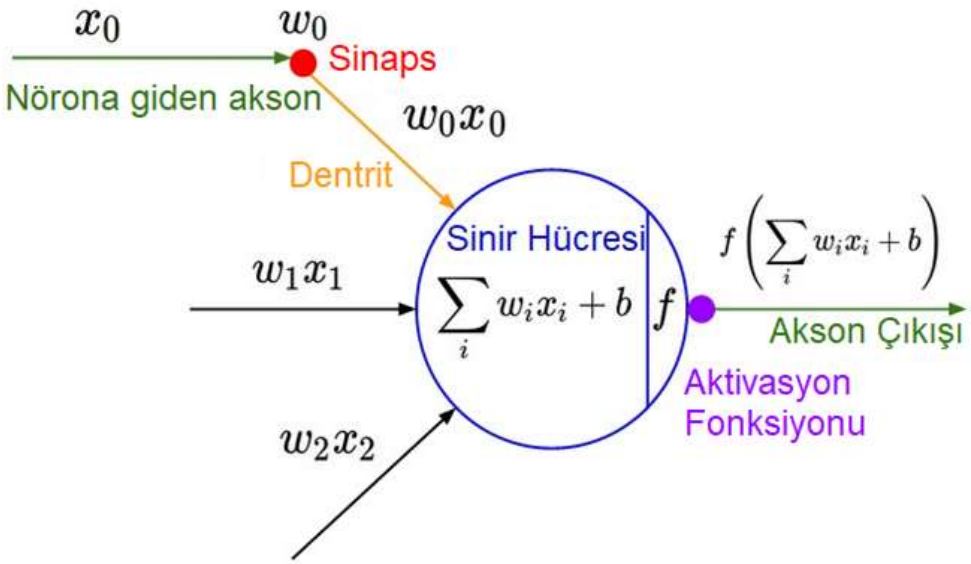
Derin öğrenme ağa bir özellik seti eklemeyi gerektiren makine öğreniminden farklı olarak, özelliklerin modelin kendisi tarafından çıkarılmasına izin verir. Şekil 3.10 derin öğrenme ile makine öğrenmesi arasında yapı farkını anlatmaktadır.



Şekil 3.10. Derin Öğrenme ve Makine Öğrenmesi arasındaki yapı farkı (Odi, 2018).

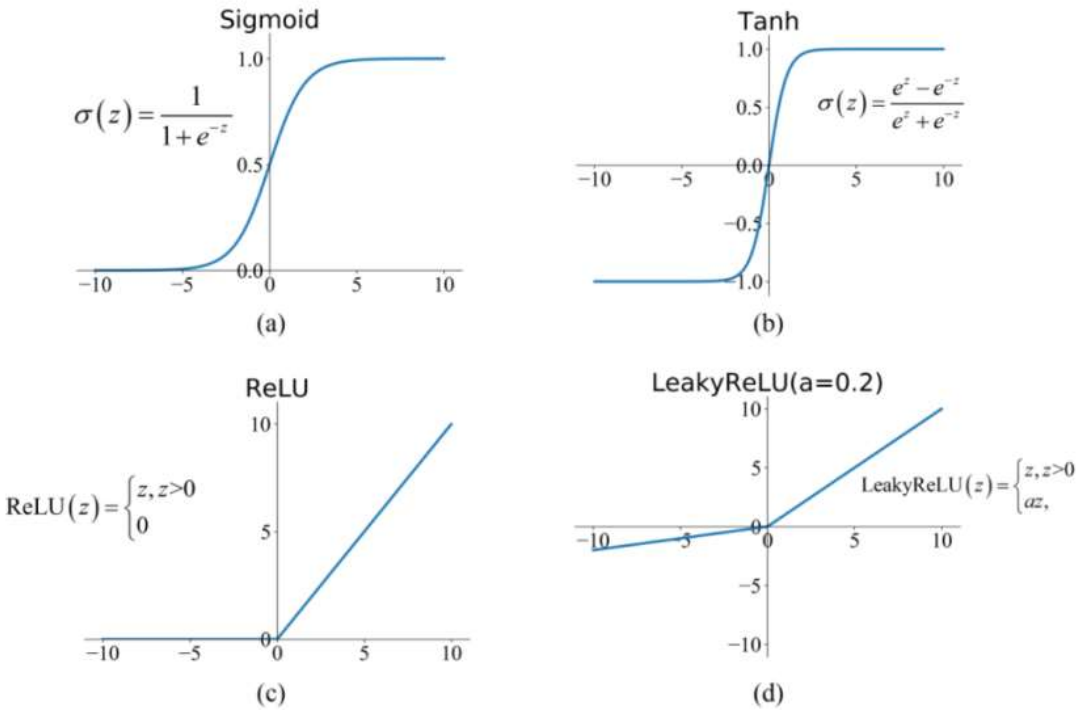
Sinir ağlarında katmanlar arasında her bir nöronu diğer katmandaki nörona bağlayan bağlantılar ve bu bağlantıların her birinin sayısal değerleri vardır. Bu sayısal değerlere ağırlık denir. Bu ağırlık değerleri sayesinde eğitim sonucunda her bir nöronun çıktı değeri için ne kadar önemli olduğu anlaşılır (Tüfekçi, 2019). Her bir nöronda hesaplanan çıkış değeri ağırlığı ile çarpılarak diğer katmandaki nöronun girdi değeri oluşturulur.

Başlangıçta ağırlık değerleri rastgele ayarlanır. Tüm nöronların bir aktivasyon fonksiyonu vardır. Temel olarak basit bir sinir ağında x girdi olarak, w ağırlık olarak tanımlanır ve ağın çıkışına aktarılan değerlere f(x) aktivasyon işlemi uygulanır. Şekil 3.11'de bir nöronun yapısı österilmiştir.



Şekil 3.11.Nöron yapısı ve aktivasyon fonksiyonu

Matris çarpımından elde edilen değerlerin doğrusal olmayan değerlere dönüştürülmesi bir aktivasyon fonksiyonu kullanılarak gerçekleştirilir. Çok katmanlı yapay sinir ağlarında doğrusal olmayan dönüşüm işlemleri için aktivasyon fonksiyonları kullanılmaktadır. Bu aktivasyon fonksiyonları arasında Step, Sigmoid, TanH, Rectified Linear Unit (ReLu) ve Parametrik RELU (PReLU) vb. bulunur. Şekil 3.13'de yayın aktivasyon fonksiyonlarının farklı girdilere göre fonksiyon çıktıları gösterilmiştir. Sonraki adımda bu nihai fonksiyon çıktıları çıkış ya da bir başka katmanın girişi olacaktır.



Şekil 3.12. Yaygın olarak kullanılan aktivasyon fonksiyonları: Sigmoid (a), Tanh (b), ReLU (c) ve LeakyReLU (d) (Feng, 2019).

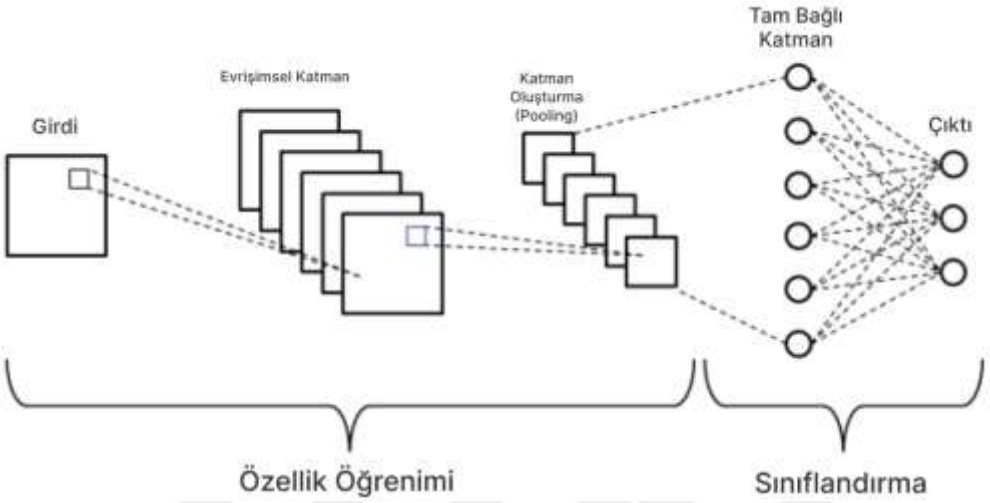
Oluşturulacak olan modelin başarımının iyileştirilebilmesi için farklı yöntemler kullanılabilmektedir. Performans iyileştirme yöntemi modelden bağımsızdır ve modelin performansını iyileştiren dışsal bir parametre olarak düşünülebilir. Problemin türüne göre çok farklı türde derin öğrenme mimarileri bulunmaktadır. Çalışmada kullanılan mimarilere aşağıda değinilmiştir.

3.2.2.1. Evrişimli Sinir Ağı (Convolutional Neural Network, CNN)

Evrişimli sinir ağları (ya da evrişimli sinir ağları) çok katmanlı sinir ağları olarak bilinmektedir. Evrişimli sinir ağları sinyal işleme, video analizi, görüntü analizi, sınıflandırma gibi birçok alanda kullanılan popüler bir derin öğrenme yöntemidir (Doğan, 2019). Standart çok katmanlı bir sinir ağı gibi evrişimli sinir ağı da bir veya daha fazla evrişim katmanından, alt örnekleme katmanından ve bir veya daha fazla bağlı katmandan oluşur. CNN algoritmasının örnek mimarisi Şekil 3.13'te gösterilmektedir. CNN'nin avantajı aynı miktarda gizli birime sahip tam bağlı bir ağdan daha az eğitim ve parametre gerektirmesidir. CNN'ler verileri çeşitli katmanlarla işlerler. Bu katmanlar şu şekildedir;

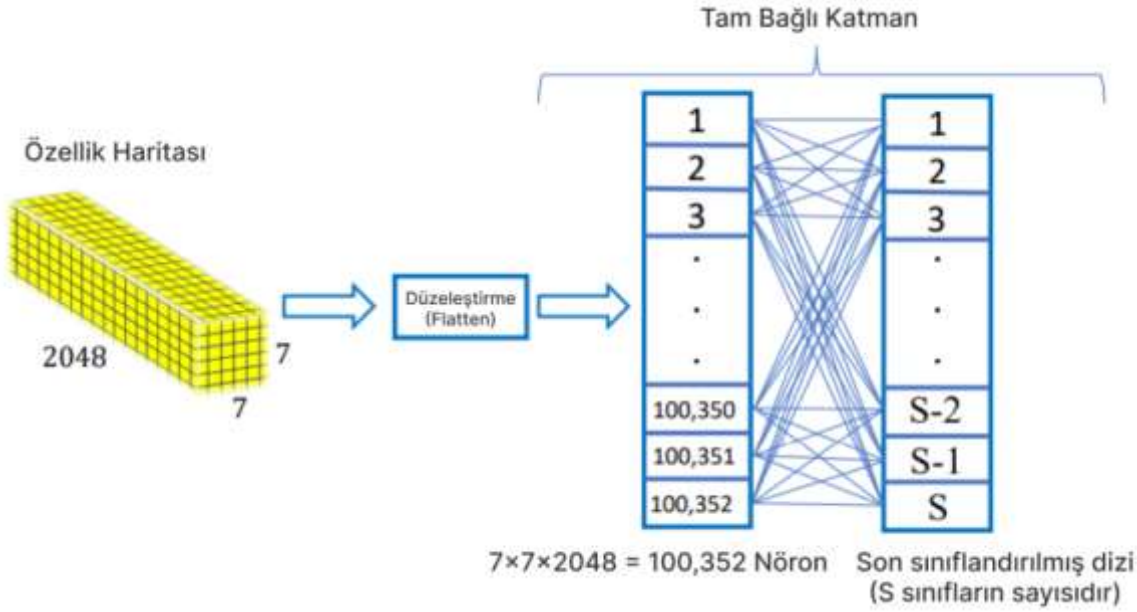
- **Evrişimsel Katmanlar (Convolutional Layer):** Özellik tespiti için kullanılır.
- **Doğrusal Olmayan Katman (Non-Linearity Layer):** Sisteme doğrusal olmayanlığı benimsetir.

- **Katman oluşturma katmanları (Pooling Layer):** Ağırlıkların sayısını azaltır ve uygunluğu doğrular.
- **Düzleştirme Katmanları (Flattening Layer):** Klasik sinir ağları için veriler hazırlar.
- **Tam Bağlantılı Katmanlar (Fully-Connected Layer):** Standart sinir ağları için veri hazırlayarak sınıflandırma için kullanılır.



Şekil 3.13.CNN mimarisi (Phung, 2019).

Evrişimsel katman bir girdi görüntüsünden özellikler çıkaran ilk katmandır. Bu katman girdi verilerinin piksellerini kullanarak görüntü özelliklerini öğrenir ve pikseller arasındaki ilişkiyi korur. Görüntü matrisi ve bir filtre olmak üzere iki girdi alan matematiksel bir işlemdir. Genelde tüm evrişimli katmanlardan sonra doğrusal olamayan katman gelir. Aktivasyon fonksiyonlarından bir tanesi bu katmanda kullanıldığı için bu katman aktivasyon katmanı (activation layer) olarak da adlandırılır. Katman oluşturulurken genellikle bir CNN'deki ardışık evrişimli katmanlar arasına eklenir. Aktivasyon katmanın görevi ağdaki temsil kaydırma boyutu ile parametre ve hesaplama sayısını azaltmaktır. Bu şekilde ağdaki uyumsuzluklar kontrol edilir. Düzleştirme katmanı ise tam bağlı katmanın girişindeki verileri hazırlar. Sinir ağları genellikle giriş verilerini tek boyutlu bir diziden alır. Bu sinir ağındaki veriler ise evirşimli ve katman oluşturma katmanından gelen matrisleri, tek boyutlu diziye çevrilmiş halidir. Düzleştirme katmanına ait bir örnek Şekil 3.14'te gösterilmiştir.

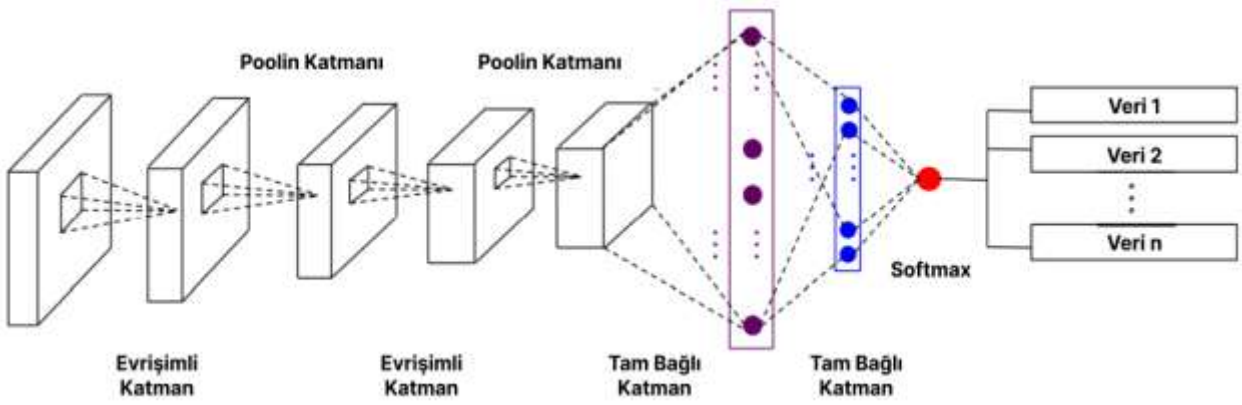


Şekil 3.14.Düzleştirme Katmanı (Flatten) ve Tam Bağlı (Full-Connected) katmanlarla sınıflandırma (Rahimzadeh, 2021).

Verilerin düzleştirme katmanından alınıp sinir ağıyla öğrenmesini sağlayan son katman tam bağlantılı katmandır. Bu katmanda verilen her bir girdinin bir sınıfa ait olma olasılığını gösteren [0,1] arası çıktılar üreten softmax fonksiyonu kullanılır (Şekil 3.15). Formülü (8)'de gösterilen softmax fonksiyonu çoklu sınıflandırma problemleri için kullanılan bir fonksiyondur. Softmax fonksiyonunun formülü (8)'de gösterilmiştir.

$$\boldsymbol{\sigma}(\underset{\boldsymbol{z}}{\rightarrow})_{\boldsymbol{i}} = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{8}$$

K çok sınıflı sınıflandırıcıdaki sınıf sayısı ve $\underset{\boldsymbol{z}}{\rightarrow}$ (z0, ... zK)'den oluşan softmax işlevine giriş vektörüdür.

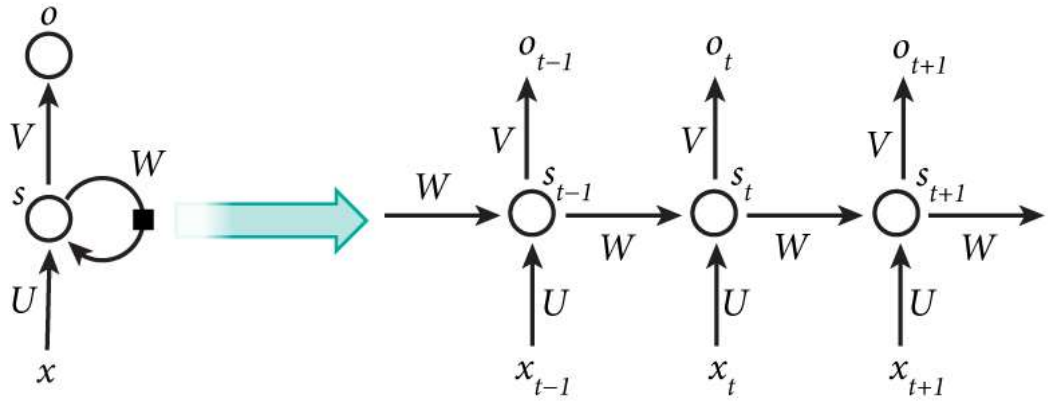


Şekil 3.15. Tam Bağlantılı Katman (Fully-Connected Layer) mimarisi (Bi, 2019).

Şekil 3.15'teki tam bağlantlı katman örneği iki evrişim katmanı, ardından maksimum havuzlama, iki tam bağlı katman ve bir softmax çıktı katmanından oluşur.

3.2.2.2. Tekrarlayan Sinir Ağı (Recurrent Neural Network, RNN)

Tekrarlayan sinir ağları, zaman adımı indeksi t ile sıralı verileri modellemek ve bağlam vektörleştirme tekniğini dahil etmek için kullanılır. Bağlam vektörü, o ana kadar hesaplananlarla ilgili bilgileri depolar ve RNN'lerin uzun ve değişen dizilerinin bilgiyi tutabildiği, geçmişten gelen bilgileri hatırlamasını sağlayan bir "bellek" görevi görür. Bu nedenle RNN'ler bir veya daha fazla girdi vektörü alabilir ve bir veya daha fazla çıktı vektörü üretebilir. RNN'ler yalnızca ağa gelen girdi örneklerini değil aynı şekilde zaman serilerinde daha önce meydana gelen girdi örneklerini de alır. Bu sinir ağının amacı ardışık olarak gelen verileri kullanmaktır (Doğan, 2019). Geleneksel sinir ağlarında girdiler ağa birbirinden bağımsız olarak girerken tekrarlayan sinir ağlarında her tablodaki verilerin sonucu önceki hesaplamalara bağlıdır. Şekil 3.15'te bir RNN'in sol tarafında açılmamış hali ve sağ tarafında nasıl açıldığı gösterilmiştir. Tasarım gereği RNN'ler derin sinir ağlarına benzer. Giriş vektörleri, ağırlık vektörleri, gizli durumlar ve çıkış vektörleri vardır.



Şekil 3.16.RNN mimarisi (Lecun, 2015).

Dahili bellekleri sayesinde RNN'ler aldıkları girdiyle ilgili önemli bilgileri hatırlayabilir ve bir sonraki adımı tahmin etmede onları çok doğru hale getirebilirler. Bu nedenle zaman serisi, finansal veriler, video, konuşma, metin, ses, hava durumu ve benzeri gibi sıralı veriler için iyi sonuçlar üretebilen bir algoritmadır. Tekrarlayan sinir ağları bir dizi ve o dizinin bağlamı hakkında diğer algoritmalara kıyasla çok daha derin ve başarılı bir anlayış yaratabilir.

RNN'ler, içinde döngüler olan ağları oluşturur ve bu durum bilginin kalıcı olmasını sağlar. Bu döngü yapısı sinir ağının girdi sırasını almasını sağlar. RNN tüm katmanlara aynı ağırlıkları ve önyargıları sağlayarak bağımsız aktivasyonları bağımlı aktivasyonlara dönüştürür, böylece artan

parametrelerin karmaşıklığını azaltır ve her çıktıyı bir sonraki gizli katmana girdi olarak vererek önceki her çıktıyı ezberler. Bu nedenle bu üç katman tüm gizli katmanların ağırlıkları ve önyargıları aynı olacak şekilde tek bir tekrarlayan katmanda birleştirilebilir.

Mevcut durum ($\boldsymbol{h_t}$), önceki durum ($\boldsymbol{h_{t-1}}$) ve giriş durumu ($\boldsymbol{x_t}$) olmak üzere mevcut durumu hesaplama formülü (9)'de gösterilmiştir.

$$\boldsymbol{h_t = f(h_{t-1}, x_t)} \tag{9}$$

Tekrarlayan nörondaki ağırlık ($\boldsymbol{w_{hh}}$), giriş nöronundaki ağırlık ($\boldsymbol{w_{xh}}$) olmak üzere tanh aktivasyon fonksiyonu uygulama formülü (10)'da gösterilmiştir.

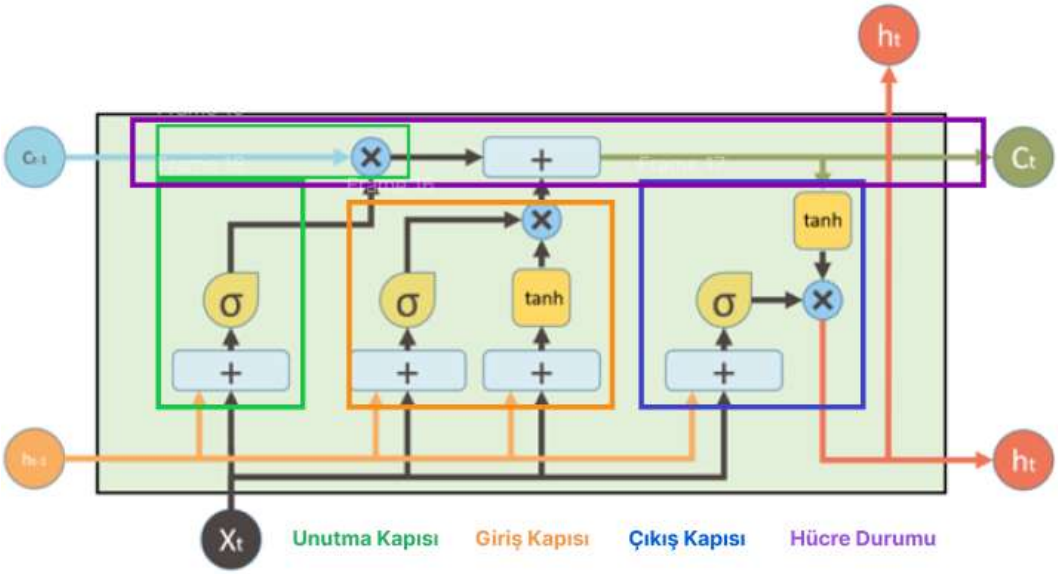
$$\boldsymbol{h_t = tanh\,(w_{hh}h_{t-1} + w_{xh}x_t)} \tag{10}$$

Çıktı ($\boldsymbol{y_t}$) ve çıktı katmanındaki ağırlık ($\boldsymbol{w_{hy}}$) olmak üzere çıktı hesaplama formülü (11)'da gösterilmiştir.

$$\boldsymbol{y_t = w_{hy}h_t} \tag{11}$$

3.2.2.3.Uzun Kısa Vadeli Hafıza Ağları (Long Short Term Memory, LSTM)

Bilinen ve geliştirilmiş bir tekrarlayan sinir ağı olan uzun kısa süreli bellek (LSTM) algoritması zaman serisi problemlerini işlemek ve tahmin elde etmek için uygun bir algoritmadır. LSTM modeli bilgi iletimini kontrol etmek için bir giriş kapısı, bir unutma kapısı ve bir çıkış kapısı olmak üzere üç ana unsurdan oluşan bellek hücresi adı verilen yeni bir yapı sunmaktadır. (Lu, 2019).



Şekil 3.17. LSTM mimarisi (Guo, 2020).

LSTM'nin temel mimarisi bir hücrenin durumu (cell state) ve kullandığı çeşitli kapılardır. Cell state olarak adlandırılan bir hücrenin durumu bir iletişim hattı ve tahminlerde bulunmak için hücreler arasında ilgili bilgileri taşıyan ağ belleği olarak açıklanabilir. Bu şekilde kısa bellek sorunu çözülür ve eski veriler ağ zincirine aktarılabilir. Hücre durumunun yolculuk sırasında taşıması gereken bilgiler kapılar aracılığıyla tanımlanır. Bu bağlantı noktaları (kapılar), hangi verilerin gerekli olup olmadığını belirleyebilir. Kapılar verileri 0 ve 1 arasında sıkıştıran bir sigmoid aktivasyon fonksiyonu kullanır. Sigmoid aktivasyonu sonucunda 0 olan bilgi unutulur ve 1 olan bilgi cell state ile birlikte yayılıp ilerlemeye devam eder.

Unutma kapısı hangi verilerin unutulacağına veya tutulacağına karar verir. Bir önceki hücreden ($\boldsymbol{h_t}$) gelen bilgiler ve mevcut bilgi ($\boldsymbol{x_t}$) sigmoid aktivasyon fonksiyonuna eklenir ve sonuca göre karar verilir. 0'lı veriler unutulur ve 1'li veriler hücre durumu boyunca devam eder.

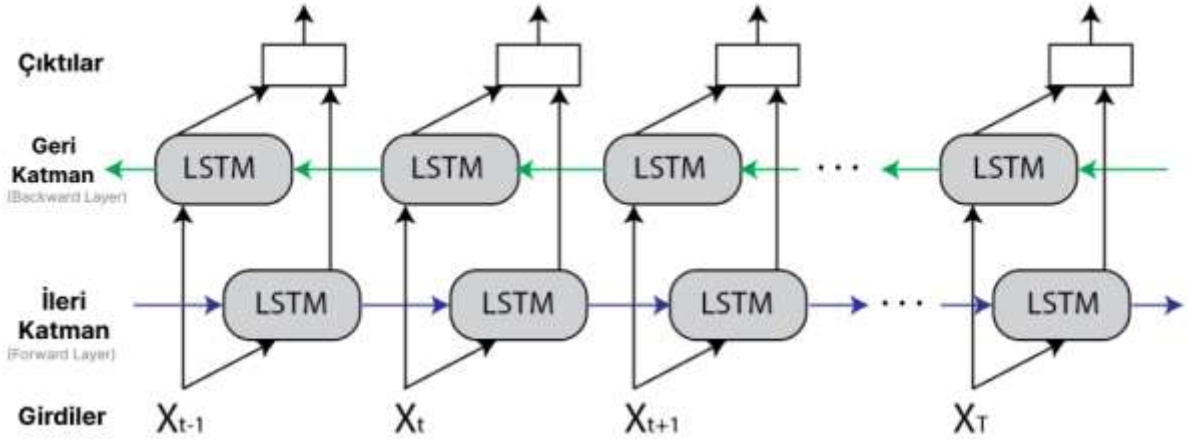
Giriş kapısı hücrenin durumunu günceller. Önceki ve güncel verilerin güncellenmesine sigmoid işleminin sonucuna göre karar verilir. 0 ile verilen bilgiler önemsiz, 1 ile verilen bilgiler önemli olarak kabul görür. Ayarlama işlemi (ağı düzenleme) ise verileri -1 ile 1 arasında sıkıştıran tanh aktivasyon fonksiyonunu kullanır. Ardından sigmoid ve tanh fonksiyonlarının çıktıları çarpılır ve bu sonuca göre hangi verilerin güncellenip hangi verilerin güncellenmeyeceğine karar verilir.

Çıkış kapısı bir sonraki hücrenin ($\boldsymbol{h_t}$+1) girişini belirler ve tahmin için kullanılır. İlk olarak, önceki veriler ve mevcut giriş verileri bir sigmoid fonksiyonuna tabi tutulur. Daha sonra hücrenin durumu ile ilgili mevcut bilgi tanh fonksiyonundan geçirilir. Son olarak bu iki sonuç çarpılarak bir sonraki hücreye hangi verinin ($\boldsymbol{h_t}$+1) girileceği kararı verilir. Mevcut hücrenin kapı fonksiyonları yapıldıktan sonra bir sonraki hücreye giden hücrenin durumuna karar verilir ve hücrenin giriş bilgisi olarak tanımlanan gizli durum ($\boldsymbol{h_t}$) bilgisine karar verilmiş olur.

3.2.2.4.Çift Yönlü Uzun Kısa Vadeli Hafıza Ağları (Bidirectional Long Short Term Memory, BiLSTM)

Çift Yönlü LSTM veya BiLSTM birisi girişi ileri yönde, diğeri ise geri yönde olmak üzere iki LSTM'den oluşan bir dizi işleme modelidir. BiLSTM'ler, ağ için mevcut olan bilgi miktarını etkili bir şekilde arttırır ve algoritma için mevcut olan bağlamı iyileştirir (örneğin, hangi kelimelerin hemen ardından geldiğini ve bir cümlede bir kelimeden önce geldiğini bilmek gibi). LSTM modelinin iki kez uygulanması uzun vadeli bağımlılıkların öğrenilmesine neden olur ve bu durum modelin doğruluğunu artıracaktır. (Siami-Namini, 2019).

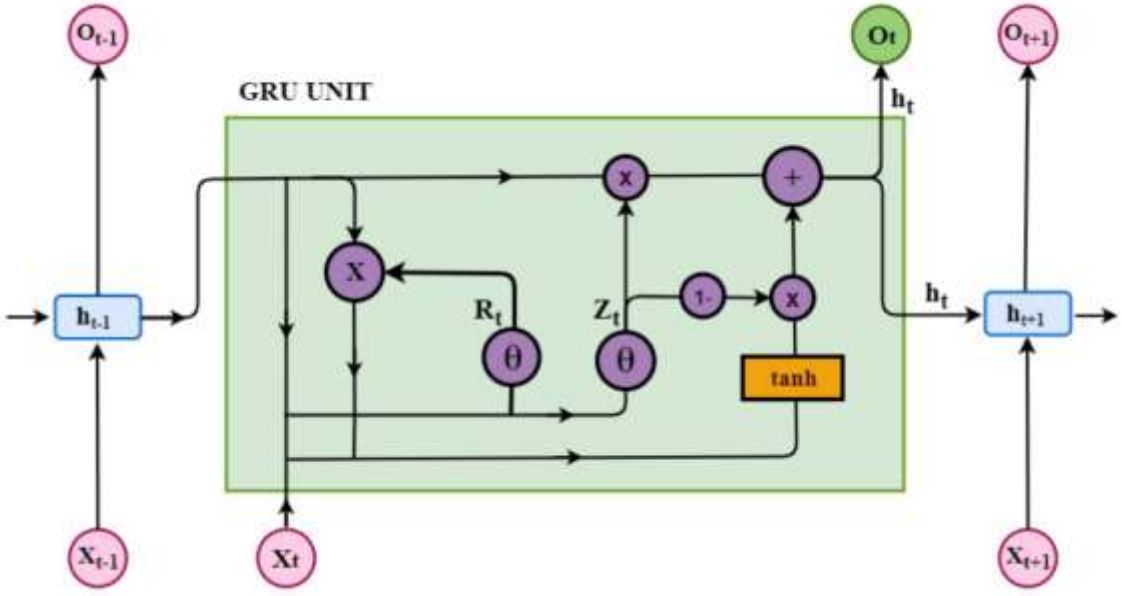
Şekil 3.18'de, ileri ve geri katmanlardan bilgi akışı gösterilmektedir. BiLSTM genellikle sıralamaların gerekli olduğu durumlarda kullanılmaktadır. Bu tür bir ağ metin sınıflandırma, konuşma tanıma ve tahmin modellerinde kullanılabilmektedir.



Şekil 3.18. BiLSTM mimarisi (Zvornicanin, 2022).

3.2.2.5. Kapılı Yinelemeli Üniteler (Gated Recurrent Units, GRU)

GRU'lar LSTM'lere benzer şekilde kısa süreli bellek çözümü olarak tek tip bir geçitleme mekanizması kullanan standart tekrarlayan sinir ağlarının (RNN'ler) güçlü çeşitleridir. GRU bilgi akışını düzenleyen ve hatta döngüleyen Gates adı verilen dahili mekanizmalara sahiptir. Gates GRU hücresinin hangi bilgilerin saklanmasının veya silinmesinin önemli olduğunu öğrenmesine yardımcı olur. Bu şekilde tahmini mümkün kılmak için önemli bilgiler ileri doğru iletilir (Bibi ve diğ., 2022). GRU'nun temel mimarisi Şekil 3.19'da verilmiştir.



Şekil 3.19.GRU mimarisi (Bibi, 2022).

Unutma kapısı ve giriş kapıları da bir güncelleme kapısı $\boldsymbol{z_t}$ tasarlamak için birleştirilir. Güncelleme kapısıtutulacak önceki bellek ve yeni bilgilerin miktarını korumaktan sorumludur. $\boldsymbol{x_t}$ bir akım giriş vektörüdür ve $\boldsymbol{h_{t-1}}$ temel olarak önceki bitişik katmandan hesaplanan değerdir. Ancak $\boldsymbol{w_z}$ güncelleme kapısı için öğrenilebilir ağırlık matrisidir.

$$\boldsymbol{z_t = \sigma(w_z\,[h_{t-1}, x_t])} \tag{12}$$

GRU ayrıca sıfırlama geçidi $\boldsymbol{r_t}$'de geçerli girişi önceki bellekle birleştirir. Ayrıca $\boldsymbol{r_t}$, denklemin önceki durumu, yeni çıktıyı tam olarak nasıl birleştirdiğini belirlemekten sorumludur.

$$\boldsymbol{r_t = \sigma(w_r\,[h_{t-1}, x_t])} \tag{13}$$

$\boldsymbol{Tan_h}$ hiperbolik bir tanjant fonksiyonudur. $\boldsymbol{Tan_h}$ için çıkış aralığı (−1,1)'dir. Ayrıca $\boldsymbol{h_t}$ geçerli hücre için hesaplanan değerdir. GRU'nun mimarisi standart RNN'den daha basittir ancak performans ve hız açısından verimli olduğu kanıtlanmıştır.

$$\boldsymbol{h_t = tan_h\,[h_{t-1}, x_t])} \tag{14}$$

$$\boldsymbol{h_t = (1 - z_t)(h_{t-1}) + z_t h_t} \tag{15}$$

3.2.3. Izgara Arama (Grid Search) Yöntemi ile Hiperparametre Optimizasyonu

Izgara arama, ağ yapılandırmasına göre tüm hiperparametre kombinasyonlarını test eden ayrıntılı bir arama veya kaba kuvvet yöntemi olarak tanımlanabilir. Sonlu bir kullanıcı tanımlı değerler kümesinin kartezyen çarpımını değerlendirerek çalışır (Belete, 2022). Grid arama ile hiperparametre seçim sürecinde belirtilen aralıktaki tüm değer kombinasyonları için ağ eğitilir, sonuçlar gözlemlenir ve en iyi kombinasyon hiperparametreler grubu olarak seçilir. Global optimumları belirleyebilmek için aşağıdaki işlem dizisinin manuel olarak yapılması gerekmektedir:

1. Arama ve aşama ölçeği için geniş bir alan belirlenir.

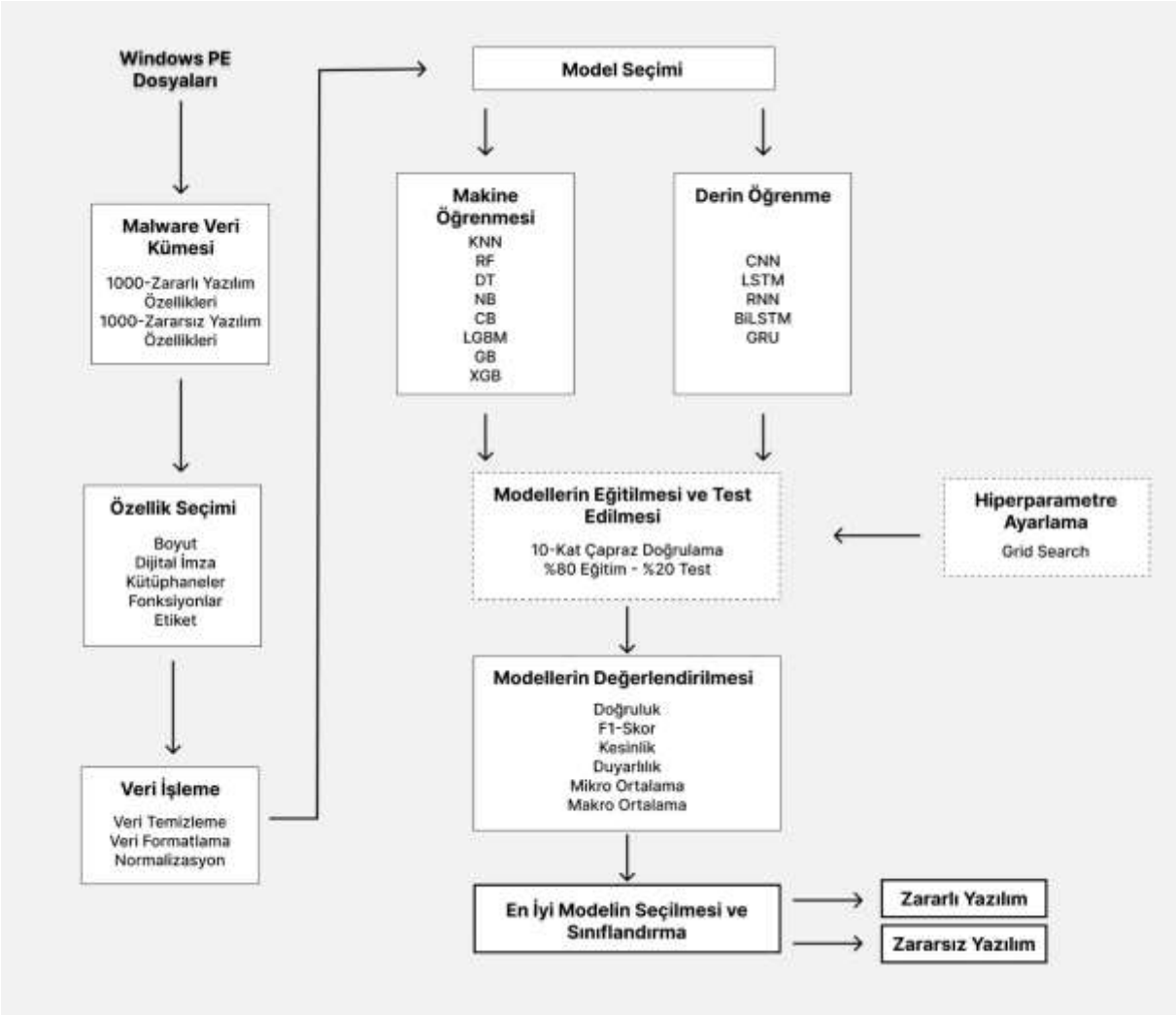
2. İyi performans gösteren hiperparametre değerler aralıklarının önceki bulgularına dayanarak arama alanı ve faz boyutu daraltılır.

3. Optimum değerlere ulaşıncaya kadar 2. adım tekrarlanır.

Bu çalışmada global optimumları belirleyebilmek için ızgara arama yönteminde kullanılan parametreler şu şekildedir.

- **estimator** (Tüm algoritmalar): Bu parametre tanımlanmış makine öğrenimi modelini belirtir.

- **param_grid**: Ayarlamak istenilen hiperparametreye göre bir dizi değer belirlenir. Anahtar olarak hiperparametre ve değerler olarak değer aralığı ile gösterilir. Bu işlem herhangi bir parametre ayarı dizisi üzerinde arama yapılmasını sağlar.
- **n_jobs** (n_jobs=-1): Paralel olarak çalışacak iş miktarını ifade eder. -1 değeri tüm işlemcilerin kullanıldığı anlamına gelmektedir.
- **Cv** (cv=10): Çapraz doğrulama için kullanılacak bölme stratejisini belirlemektedir. Varsayılan 3 katlı çapraz doğrulamayı kullanmak için parametre değeri belirtmeye gerek yoktur.



Şekil 3.20. Zararlı yazılım tespit sisteminin genel tasarımı.

3.2.4. Zararlı Yazılım Analiz Yöntemleri

Zararlı yazılım analizi yöntemleri, genellikle zararlı yazılım olarak bilinen zararlı yazılımları analiz etmek ve anlamak için kullanılan çeşitli teknikleri ve yaklaşımları ifade eder. Bu yöntemler, güvenlik araştırmacılarının ve analistlerin zararlı yazılımın davranışı, işlevselliği ve potansiyel etkisi

hakkında bilgi edinmesine yardımcı olur. Yaygın olarak kullanılan zararlı yazılım analiz yöntemleri statik ve dinamik analizdir.

3.2.4.1. Statik Analiz Yöntemi

Statik analiz genel olarak bir programın kaynak kodunu (ve bazen de nesne kodunu) girdi olarak alır, bu kodu çalıştırmadan inceler ardından kod yapılarını, ifade kümelerini ve nasıl yapıldığını kontrol ederek sonuçlar verir. Değişken olan değerler çeşitli fonksiyon çağrıları boyunca işlenir (Li, 2017). Statik analizde meta veri dizeleri, kodlar ve içe aktarılan kütüphaneler gibi statik özellikler ayıklanır ve makine öğrenimi sınıflandırmasında özellik seçimi veya özellik çıkarma aşamasında kullanılır. Statik zararlı yazılım analizinin giriş dosyası türü bayt kodu, DLL, belgeler, derleme kodu, vb. türünde olmalıdır, bu dosya türlerinden statik özellikler çıktı olarak çıkarılabilir (Harshalatha, 2020). Statik analiz yönteminin temel ve ileri düzey olmak üzere iki kategorisi vardır. Temel statik

Statik analiz ile yapılan analizlerde özellikler dosya içindeki kitaplıklar, dizgiler, fonksiyonlar ve kaynaklar gibi tüm olası statik bilgiler çıkarılarak elde edilir. Bu nedenle analistler zararlı olabilecek dosyaların yürütülmesinden önce bu dosyaların işlevleri ve davranışları hakkında genel ve temel bir bilgi edinir. Bununla birlikte gelişmiş statik analiz ile yapılan analizlerde yani kod analizi ile her bir bileşeni incelemek için ikili dosyalar yine çalıştırmadan incelenir (Şahin, 2021). Gelişmiş statik analiz ayrıca zararlı yazılımın özellikleri hakkında temel analizden daha fazla bilgi çıkarılmasını sağlayabilmektedir.

Statik analiz bazı durumlarda zararlı yazılım yazarları tarafından kullanılan farklı kod gizleme teknikleri nedeniyle zararlı yazılımları keşfetmede başarısız olur. Bu nedenle, dinamik analiz farklı işlevsellikleri incelemek için ve kod gizlemeyi analiz etmek için daha uygun bir yöntem olabilir.

3.2.4.2. Dinamik Analiz Yöntemi

Dinamik analiz, zararlı yazılım örneğini çalıştırmayı önlemek ve bulaşmayı ortadan kaldırmak veya diğer sistemlere yayılmasını durdurmak için sistemdeki davranışını gözlemlemeyi içerir. Sistem kapalı ve yalıtılmış bir sanal ortamda kurulur. Böylece zararlı yazılım örneği sisteme zarar verme riski olmadan kapsamlı bir şekilde çalıştırılıp, incelenebilir. Dinamik analizle elde edilebilecek bilgilere örnek olarak API çağrıları, sistem çağrıları, talimat izleri, kayıt defteri değişiklikleri, bellek yazmaları vb. verilebilir (Damodaran, 2017). Zararlı yazılımların dinamik analizi için kullanılan sistemler (örneğin Cuckoo sandbox) yürütme sırasında çalışma zamanına bakarak davranışları çıkarırlar. Burada amaç, zararlı yazılım çalıştırılmadan gerçek sistemdeki test ortamından izole edilerek istenen bilgileri elde etmek için korumalı alanı kullanmaktır. Raporlar zararlı yazılımın yürütülmesi hakkında özet bir bilgi

sağlamaktadırlar. Örneğin bir Cuckoo Sandbox raporundan çıkarılan özellikler şunlardır: Özet bilgi, dosyalar, yürütme sırasında API çağrısı, kayıt defteri anahtarları, erişim URL'leri, IP adresi ve DNS sorguları (Ijaz, 2019).

Olası enfeksiyonlara karşı korunmak için her zararlı dosyanın korumalı alanlar veya sanal makineler gibi kontrollü bir ortamda çalıştırılması gerekir. Ancak bu durum davranışı izlemek için belirli bir zamana ihtiyaç duyulmasına neden olur. Bazı durumlarda izleme yüksek miktarda tarama süresi gerektirebilir. Ayrıca güvenli ortam gerçek bir çalışma zamanı ortamından farklı olabilir ve zararlı yazılım iki ortamda farklı davranışlar sergileyebilir. Sonuç olarak statik ve dinamik analiz farklı problemler için kullanılabilmektedir ve bu nedenle yazılımlar çıktıları farklı zararlı yazılım tespit yöntemlerinde inceler (Şahin, 2021).

Statik analiz ve dinamik analiz yöntemlerinin avantaj ve dezavantajlarının karşılaştırılmasının yapıldığı Tablo 3.6 aşağıda verilmiştir.

Tablo 3.6. Zararlı yazılım analizi yaklaşımlarının karşılaştırılması

Yöntem	Avantaj	Dezavantaj
Statik Analiz	Hızlı ve güvenlidir. Düşük kaynak tüketimi vardır. Çok yollu zararlı yazılım analizi yapabilir. Dinamik analize göre daha güvenlidir. Yüksek doğruluk oranı elde eder.	Gizlenmiş ve şifrelenmiş zararlı yazılımları analiz edemez. Bilinmeyen zararlı yazılımları analiz etmek zordur.
Dinamik Analiz	Gizlenmiş ve şifrelenmiş zararlı yazılımları analiz edebilir. Statik analizden daha yüksek doğruluk oranı elde eder. Bilinen ve bilinmeyen zararlı yazılımları analiz edebilir.	Yavaş ve güvensizdir. Yüksek kaynak tüketimi vardır. Zaman alıcı ve savunmasızdır. Kod erişilebilirliği sınırlıdır.

4. BULGULAR VE TARTIŞMA

Bu bölümde zararlı yazılımların makine öğrenmesi ve derin öğrenme ile tespiti için önerilen yöntemlerin deneysel sonuç ve grafikleri verilmiştir. Önerilen yöntemlerin başarı oranları doğruluk (16), duyarlılık (17), kesinlik (18), f1-skor (19), mikro ortalama (20), makro ortalama (21) değerleri ile hesaplanmıştır.

Doğruluk: Kaç tane zararlı ve zararsız dosyanın doğru şekilde sınıflandırıldığının yüzdesini verir.

$$\boldsymbol{Doğruluk} = \frac{\boldsymbol{DP+DN}}{\boldsymbol{DP+DN+YP+YN}} \tag{16}$$

Kesinlik: Doğru şekilde sınıflandırılmış zararlı ve zararsız dosyaların yüzdesini verir. Gerçek pozitif örnek sayısının, toplam pozitif örnek sayısına bölünmesiyle hesaplanır.

$$\boldsymbol{Kesinlik} = \frac{\boldsymbol{DP}}{\boldsymbol{DP+YN}} \tag{17}$$

Duyarlılık: Doğru şekilde sınıflandırılmış zararlı ve zararsız dosyaların yüzdesini verir. Gerçek pozitif örnek sayısının toplam tahmin edilen pozitif örnek sayısına bölünmesiyle hesaplanır.

$$\boldsymbol{Duyarlılık} = \frac{\boldsymbol{DP}}{\boldsymbol{DP+YP}} \tag{18}$$

F1-Skor: Kesinlik ve duyarlılık değerlerinin harmonik ortalaması ile hesaplanır. Doğruluk değerinin yanı sıra f1-skor değerinin de kullanılmasının temel nedeni, eşit dağılmayan veri kümeleri üzerinde yanlış model seçiminden kaçınmaktır.

$$\boldsymbol{F1-Skor} = \frac{\textbf{kesinlik} \times \boldsymbol{duyarlılık}}{\textbf{kesinlik} + \boldsymbol{duyarlılık}} \tag{19}$$

F1-Skor Mikro Ortalama: Mikro f-skor normal f1-skor formülüdür, ancak her sınıf için ayrı ayrı yerine DP, YP ve YN toplam sayısı kullanılarak hesaplanır.

$$\boldsymbol{Mikro\ Ortalama} = \frac{\boldsymbol{DP}}{\textbf{DP} + \frac{1}{2}*(\boldsymbol{YP+YN})} \tag{20}$$

F1-Skor Makro Ortalama: Makro f1 skoru, sınıf başına hesaplanan f1 skorlarının ağırlıklandırılmamış ortalamasıdır. F1 skoru için en basit toplamadır.

$$\boldsymbol{Makro\ Ortalama} = \frac{\mathbf{1}}{\boldsymbol{Sınıf\ sayısı}} \sum_{\boldsymbol{i\ \epsilon\ sınıf\ sayısı}} \boldsymbol{F1 - skor(i)} \quad (21)$$

Karışıklık matrisleri (Confusion Matrix, CM) bir sınıflandırma algoritmasının performansını açıklamak için kullanılan tablodur. Bu tablo sınıflandırma algoritmasının performansını görselleştirir ve özetler. Şekil 3.21 zararlı yazılım sınıflandırması için karışıklık matrisini göstermektedir (Singh, 2021).

		TAHMİN EDİLEN DEĞERLER	
		ZARARSIZ YAZILIM	ZARARLI YAZILIM
GERÇEK DEĞERLER	ZARARSIZ YAZILIM	DN	YP
	ZARARLI YAZILIM	YN	DP

Şekil 3.21. Zararlı yazılım sınıflandırmasının karışıklık matrisi, (Čeponis, 2020).

DP (Doğru Pozitif): Zararlı olarak doğru bir şekilde sınıflandırılan dosyaların sayısını tanımlar.
DN (Doğru Negatif): Zararsız olarak doğru bir şekilde sınıflandırılan dosyaların sayısını tanımlar.
YP (Yanlış Pozitif): Yanlış bir şekilde zararlı olarak sınıflandırılan dosyaların sayısını tanımlar.
YN (Yanlış Negatif): Yanlış bir şekilde zararsız olarak sınıflandırılan dosyaların sayısını tanımlar.

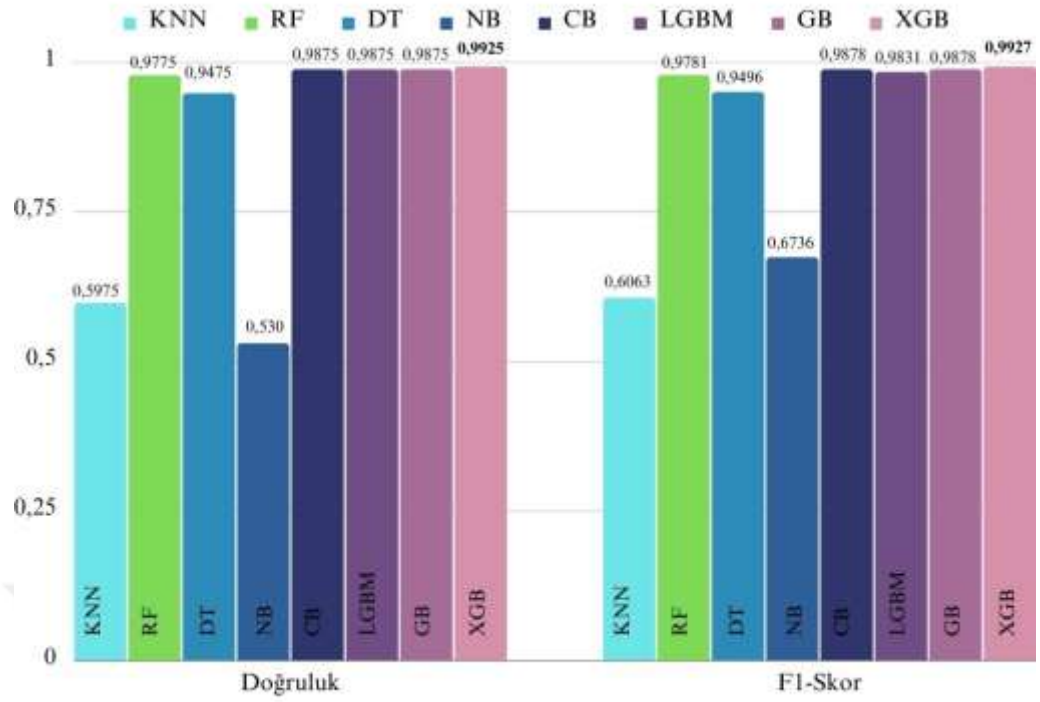
4.1. Makine Öğrenmesi Yöntemlerinin Karşılaştırılması

Veri kümesinde 1000'i zararlı, 1000'i zararsız olmak üzere 2000 adet yazılım etiketlenmiş ve eğitilmiştir. Tablo 4.1'de makine öğrenmesi algoritmalarında kullanılan hiperparametreler gösterilirken makine öğrenmesi yöntemleri ile eğitilen modellerden elde edilen sonuçlar ise Şekil 4.1, 4.2 ve Tablo 4.2, 4.3'te gösterilmiştir.

Tablo 4.1. Makine öğrenmesi algoritmaları hiperparametre değerleri

Algoritma	Hiperparametre	Değer Aralıkları
RF	Max_depth	(1,10)
	Max_features	[3, 5, 10, 15]
	N_estimators	[100, 200, 500, 1000, 2000]
XGB	Max_depth	(1,10)

	Learning_rate	[0.1, 0.01, 0.5]
	N_estimators	[100, 200, 500, 1000, 2000]
	Colsample_bytree	[0.4, 0.5, 0.6, 0.9, 1]
LGBM	Max_depth	(1,10)
	Learning_rate	[0.1, 0.01, 0.5]
	N_estimators	[100, 200, 500, 1000, 2000]
	Colsample_bytree	[0.4, 0.5, 0.6, 0.9, 1]
GB	Max_depth	(1,10)
	Learning_rate	[0.1, 0.01, 0.5]
	N_estimators	[50, 150, 200, 500, 1000]
	Subsample	[0.25, 0.5, 0.75]
KNN	N_neighbours	(1,50)
DT	Max_depth	(1,10)
	Min_samples_split	(2,50)
CB	Depth	[4, 5, 6, 7, 8, 9, 10]
	Learning_rate	[0.1, 0.01, 0.5]
	Iterations	[200,500]



Şekil 4.1. Makine öğrenmesi algoritmalarının doğruluk ve f1-skor değerlerinin karşılaştırma grafiği

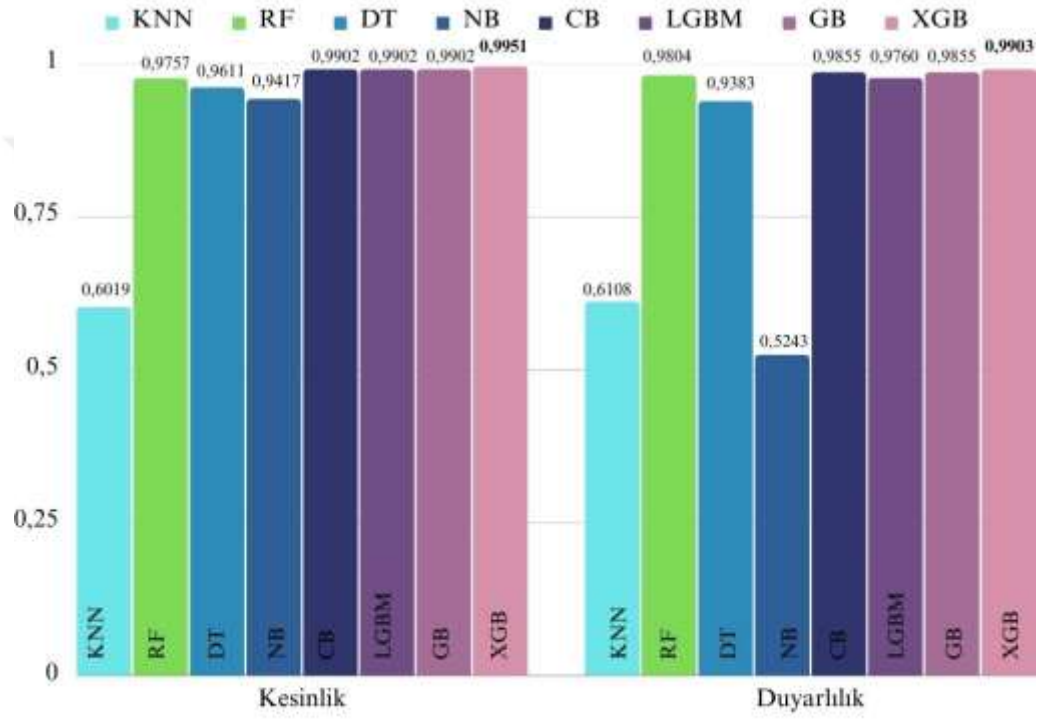
Tablo 4.2'de görüldüğü gibi doğruluk oranı sonuçlarına göre XGB algoritması ile %99,25 değerinde yüksek bir doğru tespit oranı elde edilmiştir. Aynı zamanda Tablo 4.2'yi incelediğimizde duyarlılık ve kesinlik değerlerinin de XGB algoritmasında en yüksek sonuçları verdiği gözlemlenmiştir. F1-skor, kesinlik ve duyarlılık değerleri farklı olmasına rağmen Boosting algoritmaları olan CB, LGBM ve GB algoritmaları da aynı doğruluk değerini vermiştir (%98,25). Bu üç algoritmanın doğruluk oranının aynı çıkması sebebiyle modelde bir hata olup olmadığını tespit edebilmek için farklı bir açık kaynak veri kümesi üzerinde model denenmiş (Pandey, 2020) ve denenen veri kümesi üzerinde algoritmaların doğruluk oranları farklı çıkmıştır. Bu sebeple bu algoritmaların bu veri kümesi üzerinde benzer sonuçlar ürettiği sonucuna ulaşılmıştır.

Tablo 4.2. Makine öğrenmesi algoritmalarının doğruluk ve f1-skor değerleri tablosu

Algoritma	**Doğruluk**	**F1-Skor**
KNN	0,5975	0,6063
RF	0,9775	0,9781
DT	0,9475	0,9496
NB	0,530	0,6736
CB	0,9875	0,9878
LGBM	0,9875	0,9831
GB	0,9875	0,9878
XGB	**0,9925**	**0,9927**

Boosting algoritmalarının ardından RF ve DT algoritmaları sırasıyla %97,75 ve %94,75 doğruluk oranlarını verirken KNN ve NB algoritmaları %59,75 ve %53,0 doğruluk oranlarını vermiştir. F1 skor, değerinin de doğruluk sonuçlarıyla orantılı sonuçlar verdiği gözlemlenmiştir. Şekil 4.2 ve Tablo 4.3 incelendiğinde NB algoritması dışındaki algoritmalarda kesinlik ve duyarlılık değerlerinin doğruluk oranlarına yakın bir şekilde çıktığı görülmüştür.

XGB, GB, LGBM ve CB gibi artırma algoritmaları esasen karar ağacı tabanlı algoritmalar olduklarından çeşitli nedenlerle diğer algoritmalardan daha güçlüdürler. Bunun nedenlerinden biri ürettikleri ağaç yapısının bir önceki ağaçtan kaynaklanan hatayı azaltma eğiliminde olmasıdır.

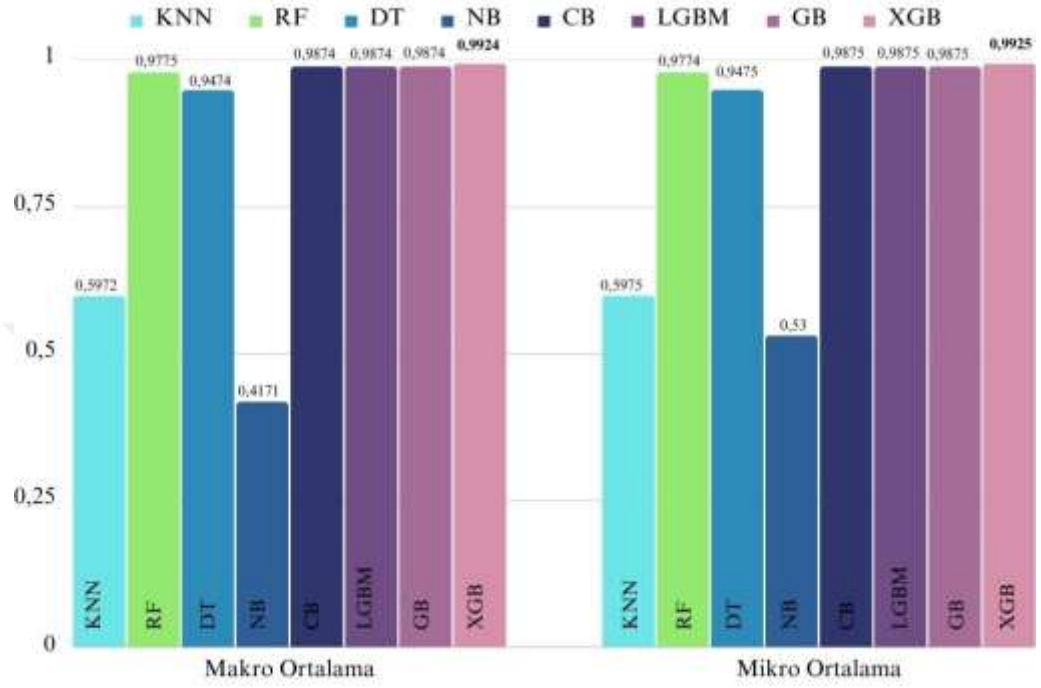


Şekil 4.2. Makine Öğrenmesi algoritmalarının duyarlılık ve kesinlik değerlerinin karşılaştırma grafiği

Tablo 4.3. Makine öğrenmesi algoritmalarının duyarlılık ve kesinlik değerleri tablosu

Algoritma	**Duyarlılık**	**Kesinlik**
KNN	0,6108	0,6019
RF	0,9804	0,9757
DT	0,9383	0,9611
NB	0,5243	0,9417
CB	0,9855	0,9902
LGBM	0,9760	0,9902
GB	0,9855	0,9902
XGB	**0,9903**	**0,9951**

Makro ve mikro ortalama performans ölçütleri arasındaki fark makro ortalamanın her sınıfı eşit, mikro ortalamanın ise her özelliği eşit şekilde ağırlıklandırmasıdır. Şekil 4.3 ve Tablo 4.4 incelendiğinde veri kümesinde bulunan sınıfların örnek sayılarının dengeli olması makro ve mikro ortalamaların aynı veya çok yakın sonuçlanmasına sebep olduğu görülmektedir.



Şekil 4.3. Makine Öğrenmesi algoritmalarının mikro ve makro değerlerinin karşılaştırma grafiği

Tablo 4.4. Makine öğrenmesi algoritmalarının mikro ve makro ortalama değerleri tablosu

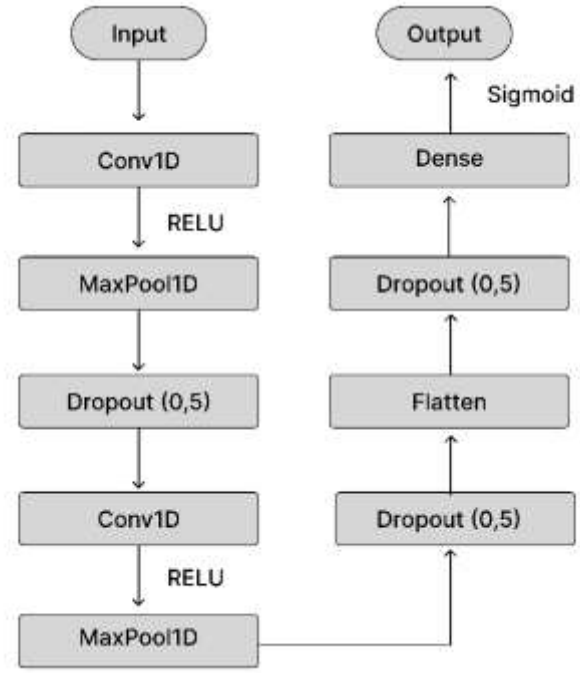
Algoritma	Mikro Ortalama	Makro Ortalama
KNN	0,5975	0,5972
RF	0,9774	0,9775
DT	0,9475	0,9474
NB	0,53	0,4171
CB	0,9875	0,9875
LGBM	0,9875	0,9875
GB	0,9875	0,9875
XGB	**0,9925**	**0,9925**

4.2. Derin Öğrenme Yöntemlerinin Karşılaştırılması

Bir derin öğrenme modeli oluştururken kurulan mimari ve seçilen hiperparametreler modelin performansında önemli bir rol oynamaktadır. İlerleyen satırlarda derin öğrenme modelleri için kurulan

mimariler gösterilmektedir. Tüm derin öğrenme modelleri Python tabanlı bir yazılım kullanılarak uygulanmış ve test edilmiştir. Modeller Keras kütüphanesi ile Tensorflow üzerinden eğitilmiştir. Sınıflandırıcıların uygun parametre değerleri grid search ile belirlenmiştir.

CNN mimarisi konvolüsyon, havuzlama, dropout, aktivasyon, düzleştirme ve dense katmanı gibi bir dizi katman tipi sunmaktadır. Şekil 4.4 ile çalışmada oluşturulan CNN mimarisini sunulmaktadır.



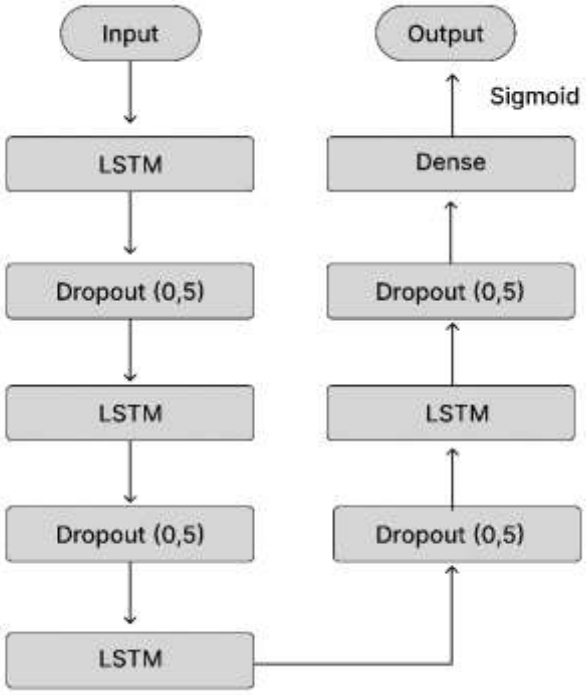
Şekil 4.4. Oluşturulan CNN mimarisi

CNN mimarisinin ana yapı taşı olarak görülen bu katma konvülasyon katmanı özellikleri çıkarmak için filtreleme işlemi yapar.

Havuzlama katmanı (pooling) ise konvolüsyon katmanları arasına sıklıkla eklenen bir katmandır. Ağlar içerisindeki parametre değerlerini ve hesaplama sayılarını azaltma işlemi yapmaktadır.

Dropout katmanı modelin ezberlemesini (overfitting) önlemek için kullanılan bir katmandır. Her adımda belirtilen oranda (modelimizde bu değer 0,5 olarak belirlenmiştir) girdiyi sıfıra eşitler, bu durum modelin veriye aşırı uyum sağlamasını önler.

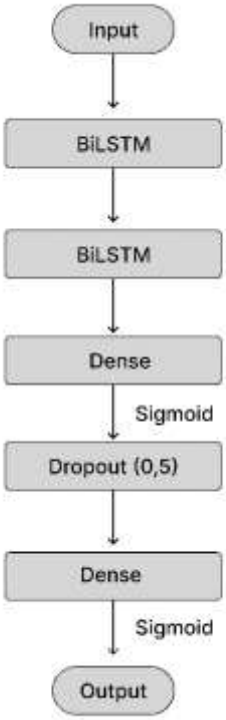
Düzleştirme katmanı (Flatten layer) ise tam bağlı katman için giriş verisi hazırlarken son olarak dense katmanı bir önceki katmandaki düğümleri mevcut katmandaki düğümlere bağlama görevini üstlenir.



Şekil 4.5. Oluşturulan katmanlı LSTM mimarisi

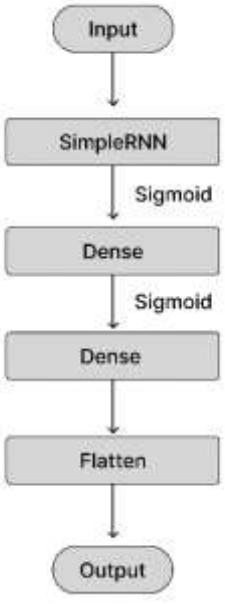
Şekil 4.5'te gösterildiği gibi LSTM modeli için 4 katmanlı bir LSTM mimarisi oluşturulmuştur. İlk LSTM katmanının ardından bir dropout katmanı vardır. Bu katmanda dropout değeri 0,5 olarak tanımlanmıştır. Her bir LSTM katmanı ardından bir dropout işlemi devam etmiştir. Son olarak bir dense katmanı eklenerek ve aktivasyon fonksiyonu olarak sigmoid kullanılarak mimari tamamlanmıştır.

Oluşturulan BiLSTM mimarisi 2 katmandan oluşmaktadır. Dense katmanı ile iletilen veriler dropout katmanı sonrasında tekrar dense katmanına iletilmiş ve nihai çıktıların elde edilmesi sağlanmıştır. Çalışmada oluşturulan BiLSTM modelinin mimarisi Şekil 4.6'da gösterilmiştir.



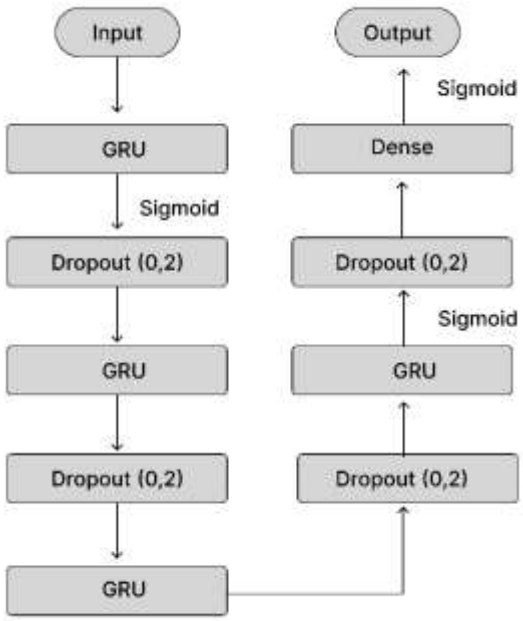
Şekil 4.6. Oluşturulan katmanlı BiLSTM mimarisi

RNN modeli için basit bir RNN yapısı kullanılmıştır. Tek katman RNN ardından dense ve düzleştirme katmanları ile çıktı elde edilmesi sağlanmıştır. Mimari Şekil 4.7'de gösterilmiştir.



Şekil 4.7. Oluşturulan katmanlı RNN mimarisi

GRU mimarisi tıpkı LSTM mimarisi gibi 4 katmandan oluşmaktadır ve her katman arsında aşırı öğrenmeyi önlemek adına dropout katmanları bulunmaktadır. Dropout değeri 0,2 olarak belirlenmiştir. Sonrasında dense katmanı ile veriler çıktı elde etmek için bir sonraki katmana iletilmiştir. Şekil 4.8 GRU mimarisini göstermektedir.



Şekil 4.8. Oluşturulan katmanlı GRU mimarisi

Derin öğrenme ile eğitilen modeller sonucunda elde edilen doğruluk ve f1-skor sonuçları Şekil 4.9 ve tablo 4.6'da gösterilmiştir. Tablo 4.5 derin öğrenme modellerinde kullanılan hiperparametreleri ve aldıkları değerleri göstermektedir.

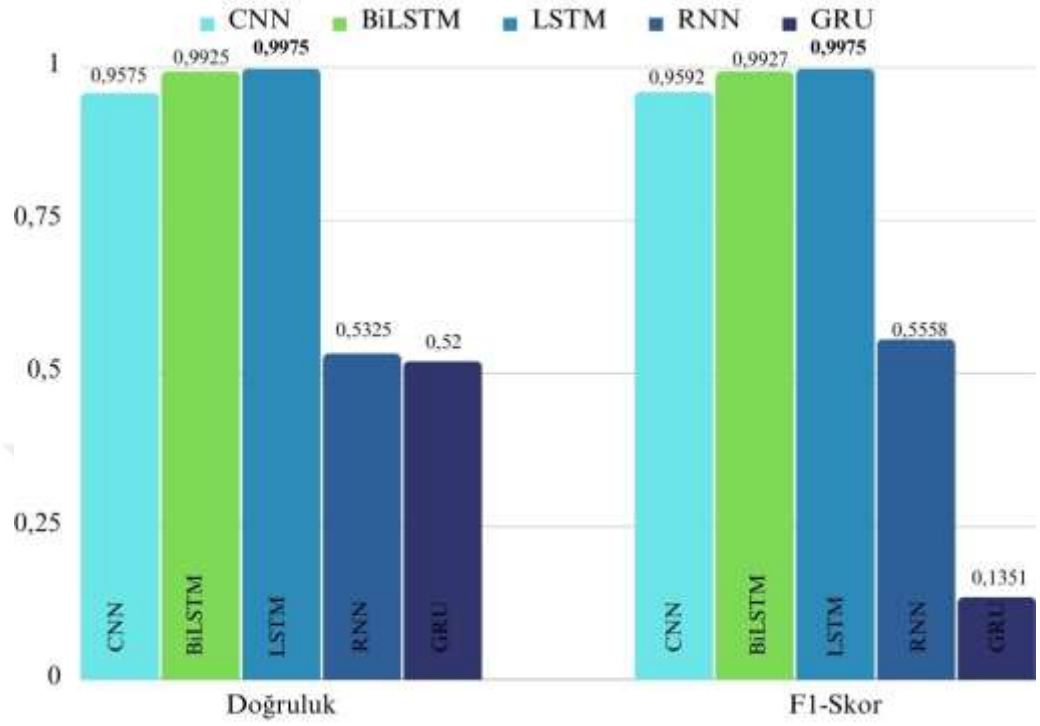
Tablo 4.5. Derin öğrenme modellerinin hiperparametre değerleri

Algoritma	Hiperparametre	Değer
CNN	Dropout	0,5
	Learning Rate	0,001
	Aktivasyon Fonksiyonu	Relu, Sigmoid
	Kernel Size	3
	Optimizer	Adam
	Epochs	200
	Batch Size	16
	Loss Function	Binary Crossentropy
	Pool Method	Max Pooling

RNN	Learning Rate	0,001
	Aktivasyon Fonksiyonu	Sigmoid
	Optimizer	Adam
	Epochs	200
	Batch Size	16
	Loss Function	Binary Crossentropy
GRU	Dropout	0,2
	Learning Rate	0,001
	Aktivasyon Fonksiyonu	Sigmoid
	Units	50
	Optimizer	Adam
	Epochs	150
	Batch Size	32
	Loss Function	Binary Crossentropy
LSTM	Dropout	0,5
	Learning Rate	0,001
	Aktivasyon Fonksiyonu	Sigmoid
	Units	50,1
	Optimizer	Adam
	Epochs	200
	Batch Size	16
	Loss Function	Binary Crossentropy
BiLSTM	Dropout	0,5
	Learning Rate	0,001
	Aktivasyon Fonksiyonu	Sigmoid
	Optimizer	Adam
	Epochs	200
	Batch Size	16
	Loss Function	Binary Crossentropy

Sonuçlar incelendiğinde LSTM algoritmasının %99,75 doğruluk oranı ile en yüksek sonucu verdiği gözlemlenmiştir. BiLSTM algoritması %99,25'lik doğruluk oranı ile LSTM algoritmasından sonra en yüksek doğruluk oranına sahip olan algoritmadır. CNN algoritması %95,75 doğruluk oranı ve %95,92 f1-skor oranı ile LSTM ve BiLSTM algoritmalarına yakın sonuçlar üretmiştir.

RNN algoritması yaklaşık %53 doğruluk oranı verirken en düşük doğruluk ve f1-skor oranını GRU algoritması üretmiştir.

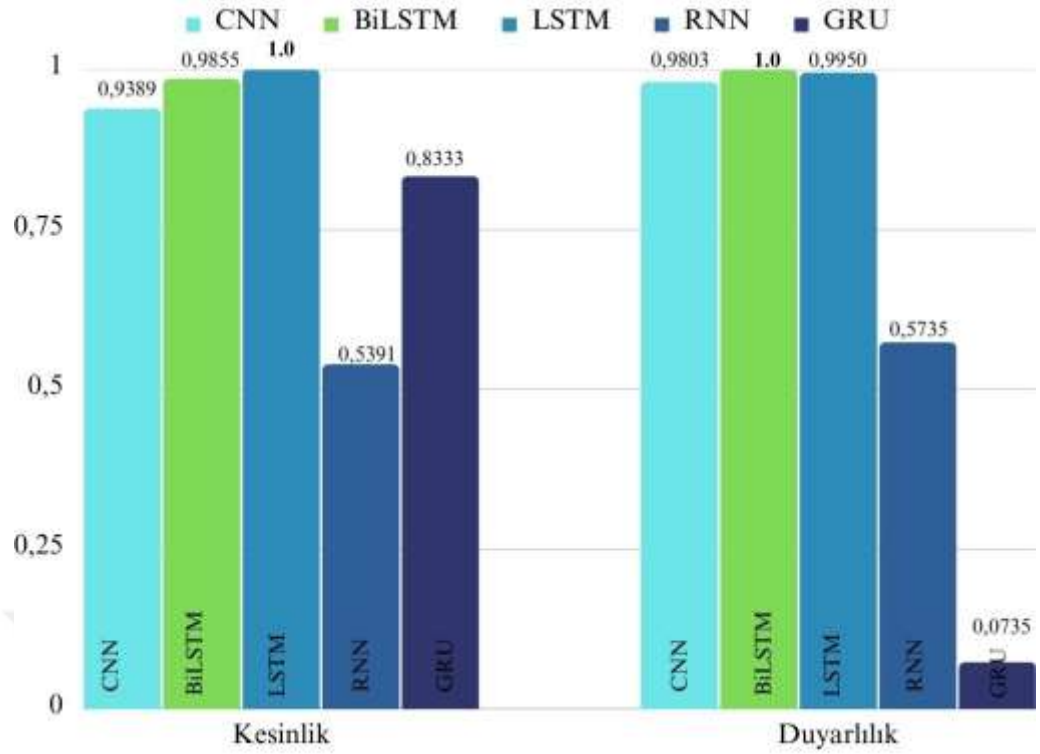


Şekil 4.9. Derin öğrenme modellerinin doğruluk ve f1-skor değerlerinin karşılaştırma grafiği

Tablo 4.6. Derin öğrenme algoritmalarının doğruluk ve f1-skor değerleri tablosu

Algoritma	**Doğruluk**	**F1-Skor**
CNN	0,9575	0,9592
BiLSTM	0,9925	0,9927
LSTM	**0,9975**	**0,9975**
RNN	0,5325	0,5558
GRU	0,52	0,1351

Modellerin kesinlik ve duyarlılık sonuçları Şekil 4.10 ve Tablo 4.7'de gösterilmiştir.



Şekil 4.10. Derin öğrenme modellerinin duyarlılık ve kesinlik değerlerinin karşılaştırma grafiği

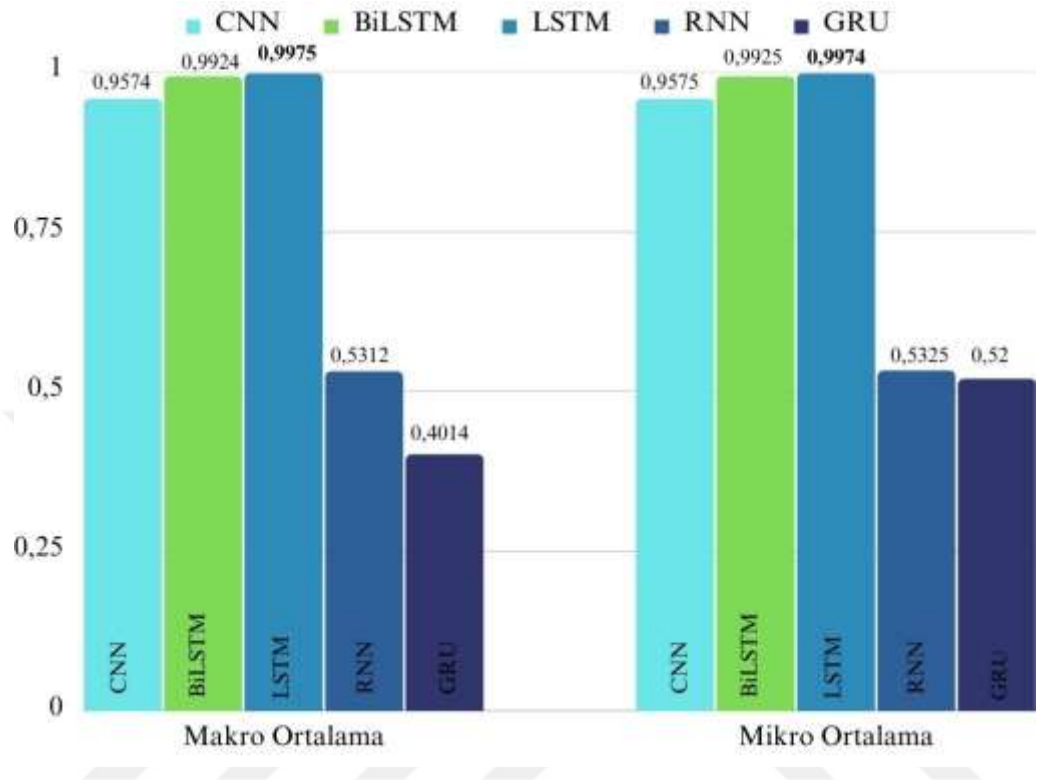
LSTM ve BiLSTM algoritmalarının kesinlik ve duyarlılık sonuçlarının yüksek olması zararlı ve zararsız dosyaların doğru şekilde sınıflandırılmış olduğunu kanıtlar niteliktedir. CNN algoritması incelendiğinde duyarlılık ve kesinlik oranlarının yüksek ve birbirine yakın olduğu gözlemlenmektedir. GRU algoritmasının duyarlılık sonucunun kesinlik sonucuna oranla daha yüksek çıkması yanlış pozitif oranının GRU algoritmasında düşük olduğunu göstermektedir. Yani yanlış bir şekilde zararlı olarak sınıflandırılan dosyaların miktarı düşüktür. Bunun yanı sıra GRU algoritmasının yanlış negatif (YN) yani yanlış bir şekilde zararsız olarak sınıflandırılan dosyaların miktarının çok fazla olduğu çıkarımı yapılabilmektedir. Tablolar genel olarak incelendiğinde en düşük sonucu GRU algoritmasının verdiği görülmektedir.

Tablo 4.7. Derin öğrenme algoritmalarının duyarlılık ve kesinlik değerleri tablosu

Algoritma	Duyarlılık	Kesinlik
CNN	0,9803	0,9389
BiLSTM	0,9855	1.0
LSTM	**0,9950**	**1.0**
RNN	0,5735	0,5391
GRU	0,8333	0,0735

Şekil 4.11 ve Tablo 4.8 incelendiğinde makine öğrenmesi sonuçlarında da değinildiği gibi, veri kümesindeki sınıfların örnek sayılarının dengeli şekilde dağılmış olması sebebiyle makro ve mikro

ortalama sonuçlarının aynı veya çok yakın olarak elde edildiği gözlemlenmiştir. Makro ve Mikro ortalama sonuçları incelendiğinde en yüksek değerlerin LSTM algoritması ile elde edildiği görülmektedir.



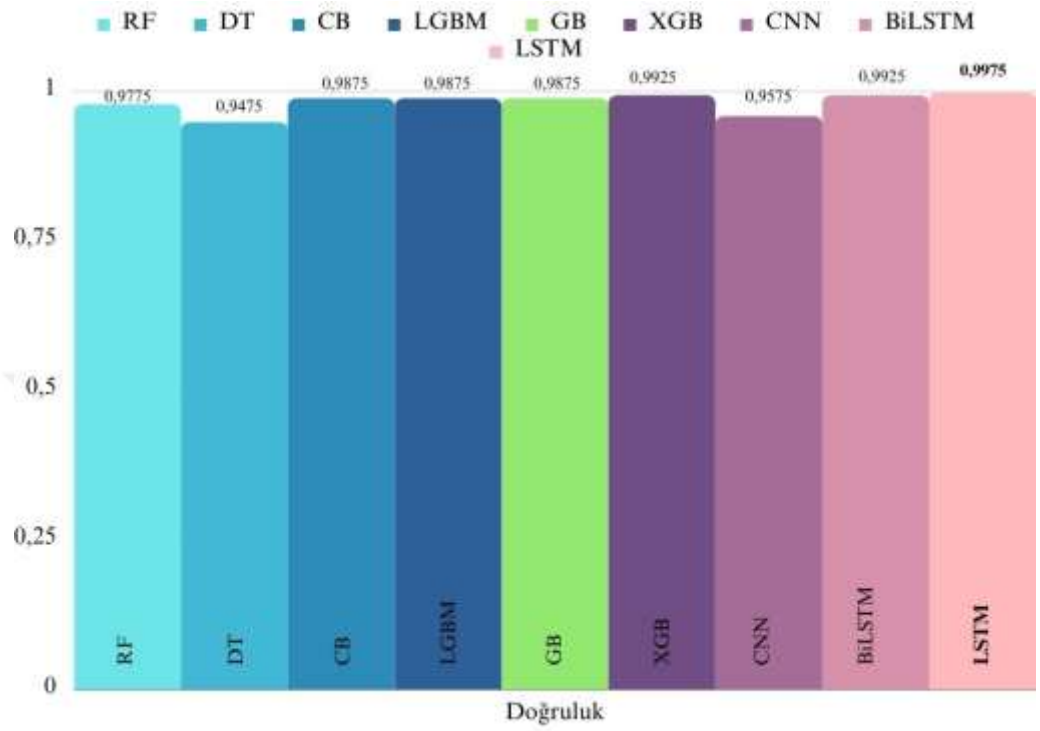
Şekil 4.11. Derin öğrenme modellerinin mikro ve makro ortalama değerlerinin karşılaştırma grafiği

Tablo 4.8. Derin öğrenme algoritmalarının mikro ve makro ortalama değerleri tablosu

Algoritma	Mikro Ortalama	Makro Ortalama
CNN	0,9575	0,9574
BiLSTM	0,9925	0,9924
LSTM	**0,9974**	**0.9975**
RNN	0,5325	0,5312
GRU	0,52	0,4014

Şekil 4.12'de makine öğrenmesi ve derin öğrenme algoritmalarının kıyaslamasını yapmak için doğruluk değerlerinin karşılaştırıldığı grafik verilmiştir. LSTM algoritması %99,75 doğruluk değeri ile en yüksek doğruluk değerine sahip algoritma olmuştur. BiLSTM algoritması arttırma algoritması olan XGB algoritması ile aynı sonucu üretmiştir. CB, LGBM, GB arttırma algoritmalarının %98,75 doğruluk oranı ile CNN, DT ve RF algoritmalarından

daha yüksek sonuç ürettiği gözlemlenmiştir. RF, CNN ve DT algoritmalarının doğruluk oranları sırasıyla %97,75, %95,75 ve %94,75 şeklindedir.



Şekil 4.12. Derin öğrenme ve makine öğrenmesi algoritmalarının doğruluk değerlerinin karşılaştırma grafiği

5. SONUÇLAR VE ÖNERİLER

Günümüzde zararlı yazılımların günden güne artması ve var olan tespit yöntemlerinin yeni zararlı yazılımlar üzerinden yetersiz hale gelmesi sebebiyle makine öğrenmesi ve/veya derin öğrenme yöntemleri ile zararlı yazılımların tespit edilmesi siber güvenlik alanında önemli bir rol oynamaktadır. Bu kapsamda, bu tez çalışmasında statik analiz yöntemleri ile zararlı yazılımların makine öğrenmesi ve derin öğrenme yöntemleriyle tespit edilmesi amaçlanmıştır.

Çalışmada statik yöntemler ile elde edilen veriler işlenerek formatlanmış ve oluşturulan veri kümesi ile analiz gerçekleştirilmiştir. Bu veri kümesi ile eğitilen ve test edilen sınıflandırma algoritmalarının ve derin öğrenme modellerinin deneysel sonuçları karşılaştırılmış ve çıkarımlar yapılmıştır. Veri kümesi 8 makine öğrenmesi algoritması ve 5 derin öğrenme algoritması ile eğitilmiş, eğitilen veriler test edilmiş ve bunun sonucunda makine öğrenmesi algoritmaları arasından XGBoost algoritması %99,25'lik doğruluk oranı ile tespit yaparken, derin öğrenme algoritmaları içerisinden LSTM algoritması %99,75'likdoğruluk oranı ile en yüksek tespit oranını vermiştir. Bunun yanı sıra RNN ve NB algoritmasının bu veri kümesi üzerinde başarılı sonuçlar üretemediği gözlemlenmiştir. Makine öğrenmesi algoritmaları için kullanılan Grid Search hiperparametre optimizasyon tekniğinin modellerin performansında bir iyileştirme sağladığı tespit edilmiştir.

Python programlama dilinin kullanıldığı bu çalışmada makine öğrenmesi ve derin öğrenme kütüphaneleri olarak Keras, Tensorflow ve Scikit-learn kütüphaneleri kullanılmıştır. Yapılan analizler sonucunda doğruluk oranının yanı sıra yanlış negatif ve yanlış pozitif değerlerini de kapsayan duyarlılık ve kesinlik değerleri de incelenmiştir. Makine öğrenmesi algoritmalarından KNN ve NB dışındaki 6 algoritmanın bu veri kümesi üzerinde kayda değer (%94 ve üzeri) seviyede duyarlı ve kesin sonuç ürettiği gözlemlenmiştir. Naive Bayes algoritmasının özelliklerin sınıf değerlerini bağımsız olarak belirleme yaklaşımı nedeniyle diğer sınıflandırma algoritmalarına kıyasla zayıf sınıflandırma gücüne sahip olduğu gözlemlenmiştir. Derin öğrenme algoritmaları için de genel durum benzerdir. Derin öğrenme modellerinin oluşturulması için Keras kütüphanesi kullanılmıştır. Sınıflandırma modelleri ReLu ve sigmoid aktivasyon fonksiyonları kullanılarak ayrı ayrı eğitilmiştir. RNN algoritması dışındaki derin öğrenme algoritmalarının da bu veri kümesi üzerinde kayda değer (%93 ve üzeri) seviyede duyarlı ve kesin sonuç ürettiği gözlemlenmiştir.

Hızla gelişen zararlı yazılım ve siber güvenlik ortamı, zararlı yazılım analizi alanında daha fazla araştırma ve ilerleme için çok sayıda fırsat sunmaktadır. Belirli alanları belirlemek ve zararlı yazılım algılama ve analizinin ilerlemesine katkıda bulunmak için mevcut zorlukları, ortaya çıkan trendleri ve pratik uygulamaları göz önünde bulundurarak gelecek çalışmalarda geliştirmeler yapılabilir.

Özellik Çıkarımının Geliştirilmesi: Statik analizde özellik çıkarma sürecini iyileştirmek için farklı teknikleri denenebilir. Zararlı yazılımın ince ayrıntılarını ve davranış kalıplarını yakalamak için daha gelişmiş statik analiz teknikleri araştırılabilir. Bu, statik analiz çerçevelerini keşfetmeyi, kod analizini, semantik analizi veya gelişmiş statik analiz araçlarını metodolojiye dahil etmeyi içerebilir.

Özellik Seçimi ve Boyut Azaltma: Zararlı yazılım tespiti için en ilgili ve bilgilendirici özellikleri belirlemek üzere özellik seçimi ve boyut azaltma tekniklerini araştırılabilir. Bu, hesaplama karmaşıklığını azaltmaya ve araştırmalarda kullanılan makine öğrenimi ve derin öğrenme modellerinin verimliliğini artırmaya yardımcı olabilir.

Gerçek Zamanlı ve Akış Analizi: Gerçek zamanlı zararlı yazılım analizi senaryolarını işlemek için araştırmalar genişletilebilir. Gerçek zamanlı olarak yeni zararlı yazılım örnekleriyle karşılaşıldığı sürekli ve dinamik bir ortamda kötü amaç zararlı yazılımları analiz etmek ve tespit etmek için metodolojiler geliştirilebilir.

Zararlı Yazılım Ailesi Sınıflandırması: İkili sınıflandırma yerine, zararlı yazılımları farklı ailelere veya kategorilere ayırmaya odaklanılabilir. Zararlı yazılım örnekleri arasındaki ortak özellikleri ve kalıpları belirlemeye yönelik teknikleri araştırarak zararlı yazılımların davranışlarına, kökenlerine veya özelliklerine göre ayrıntılı bir şekilde sınıflandırılmasını ve gruplandırılması üzerine çalışabilir.

KAYNAKLAR

Abdulazeez, A., & Charbuty, B. (2021). Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends*, *2*(01), 20-28.

Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2020). DL-Droid: Deep learning based android malware detection using real devices. *Computers & Security*, *89*, 101663.

Alzheimer's disease: from Alzheimer's disease neuroimaging initiative (ADNI) database. *Neural regeneration research*, *13*(6), 962.

Aydoğan, E., & Şen, S. (2014). Kötücül Yazılımların Tespitinde Makine Öğrenmesi Yöntemlerinin Analizi. In *Signal Processing and Communications Applications Conference (SIU)*.

Azeez, N. A., Odufuwa, O. E., Misra, S., Oluranti, J., & Damaševičius, R. (2021, February). Windows PE malware detection using ensemble learning. In *Informatics* (Vol. 8, No. 1, p. 10). MDPI.

Baturynska, I., & Martinsen, K. (2021). Prediction of geometry deviations in additive manufactured parts: comparison of linear regression with machine learning algorithms. *Journal of Intelligent Manufacturing*, *32*(1), 179-200.

Bayazit, E. C., Sahingoz, O. K., & Dogan, B. (2022, June). A Deep Learning Based Android Malware Detection System with Static Analysis. In *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)* (pp. 1-6). IEEE.

Belete, D. M., & Huchaiah, M. D. (2022). Grid search in hyperparameter optimization of machine learning models for prediction of HIV/AIDS test results. *International Journal of Computers and Applications*, *44*(9), 875-886.

Bi, H., Xu, F., Wei, Z., Xue, Y., & Xu, Z. (2019). An active deep learning approach for minimally supervised PolSAR image classification. *IEEE Transactions on Geoscience and Remote Sensing*, *57*(11), 9378-9395.

Bibi, I., Akhunzada, A., Malik, J., Iqbal, J., Musaddiq, A., & Kim, S. (2020). A dynamic DL-driven architecture to combat sophisticated Android malware. *IEEE Access*, *8*, 129600-129612.

Bulut, I., & Yavuz, A. G. (2017, May). Mobile malware detection using deep neural network. In *2017 25th Signal Processing and Communications Applications Conference (SIU)* (pp. 1-4). IEEE.

Cho, Y. (2019, November). Dynamic RNN-CNN based malware classifier for deep learning algorithm. In *2019 29th International Telecommunication Networks and Applications Conference (ITNAC)* (pp. 1-6). IEEE.

Chumachenko, K. (2017). Machine learning methods for malware detection and classification.

Čeponis, D., & Goranin, N. (2020). Investigation of dual-flow deep learning models LSTM-FCN and GRU-FCN efficiency against single-flow CNN models for the host-based intrusion and malware detection task on univariate times series data. *Applied Sciences*, *10*(7), 2373.

Damodaran, A., Troia, F. D., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. *Journal of Computer Virology and Hacking Techniques*, *13*(1), 1-12.

Dimitriadis, S. I., Liparas, D., & Alzheimer's Disease Neuroimaging Initiative. (2018). How random is the random forest? Random forest algorithm on the service of structural imaging biomarkers for Alzheimer's disease: from Alzheimer's disease neuroimaging initiative (ADNI) database. *Neural regeneration research*, *13*(6), 962.

DOĞAN, F., & TÜRKOĞLU, İ. (2019). Derin öğrenme modelleri ve uygulama alanlarına ilişkin bir derleme. *Dicle Üniversitesi Mühendislik Fakültesi Mühendislik Dergisi*, *10*(2), 409-445.

Dong, S., Khattak, A., Ullah, I., Zhou, J., & Hussain, A. (2022). Predicting and analyzing road traffic injury severity using boosting-based ensemble learning models with SHAPley Additive exPlanations. *International journal of environmental research and public health*, *19*(5), 2925.

Feng, J., He, X., Teng, Q., Ren, C., Chen, H., & Li, Y. (2019). Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, *100*(3), 033308.

Guo, Y., Cao, X., Liu, B., & Peng, K. (2020). El Niño index prediction using deep learning with ensemble empirical mode decomposition. *Symmetry*, *12*(6), 893.

Harshalatha, P., and R. Mohanasundaram. "Classification Of Malware Detection Using Machine Learn-ing Algorithms: A Survey." *International Journal of Scientific & Technology Research* 9.02 (2020).

Haq, I. U., Khan, T. A., & Akhunzada, A. (2021). A dynamic robust DL-based model for android malware detection. *IEEE Access*, *9*, 74510-74521.

Ijaz, M., Durad, M. H., & Ismail, M. (2019, January). Static and dynamic malware analysis using machine learning. In *2019 16th International bhurban conference on applied sciences and technology (IBCAST)* (pp. 687-691). IEEE.

Islam, R., Tian, R., Batten, L. M., & Versteeg, S. (2013). Classification of malware based on integrated static and dynamic features. *Journal of Network and Computer Applications*, *36*(2), 646-656.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.

Li, L., Bissyandé, T. F., Papadakis, M., Rasthofer, S., Bartel, A., Octeau, D., ... & Traon, L. (2017). Static analysis of android apps: A systematic literature review. *Information and Software Technology*, *88*, 67-95.

Lopez-Bernal, D., Balderas, D., Ponce, P., & Molina, A. (2021). Education 4.0: teaching the basics of KNN, LDA and simple perceptron algorithms for binary classification problems. *Future Internet*, *13*(8), 193.

Lu, R. (2019). Malware detection with lstm using opcode language. *arXiv preprint arXiv:1906.04593*.

Mahesh, B. (2020). Machine learning algorithms-a review. *International Journal of Science and Research (IJSR).[Internet]*, *9*, 381-386.

Miralles-Pechuán, L., Rosso, D., Jiménez, F., & García, J. M. (2017). A methodology based on Deep Learning for advert value calculation in CPM, CPC and CPA networks. *Soft Computing*, *21*(3), 651-665.

Odi, U., & Nguyen, T. (2018, July). Geological facies prediction using computed tomography in a machine learning and deep learning environment. In *SPE/AAPG/SEG Unconventional Resources Technology Conference*. OnePetro.

Rafique, M. F., Ali, M., Qureshi, A. S., Khan, A., & Mirza, A. M. (2019). Malware classification using deep learning based feature extraction and wrapper based feature selection technique. *arXiv preprint arXiv:1910.10958.*

Pandey, P. (2020). Kaggle. https://www.kaggle.com/code/parulpandey/penguin-dataset-the-new-iris/data

Patil, R., & Deng, W. (2020, March). Malware Analysis using Machine Learning and Deep Learning techniques. In *2020 SoutheastCon* (Vol. 2, pp. 1-7). IEEE.

Phung, V. H., & Rhee, E. J. (2019). A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets. *Applied Sciences*, *9*(21), 4500.

Rahimzadeh, M., Parvin, S., Safi, E., & Mohammadi, M. R. (2021). Wise-srnet: A novel architecture for enhancing image classification by learning spatial resolution of feature maps. *arXiv preprint arXiv:2104.12294.*

Santos, I., Devesa, J., Brezo, F., Nieves, J., & Bringas, P. G. (2013). Opem: A static-dynamic approach for machine-learning-based malware detection. In *International joint conference CISIS'12-ICEUTE´ 12-SOCO´ 12 special sessions* (pp. 271-280). Springer, Berlin, Heidelberg.

Shatnawi, A. S., Yassen, Q., & Yateem, A. (2022). An Android Malware Detection Approach Based on Static Feature Analysis Using Machine Learning Algorithms. *Procedia Computer Science*, *201*, 653-658.

Siami-Namini, S., Tavakoli, N., & Namin, A. S. (2019). A comparative analysis of forecasting financial time series using arima, lstm, and bilstm. *arXiv preprint arXiv:1911.09512.*

Singh, P., Singh, N., Singh, K. K., & Singh, A. (2021). Diagnosing of disease using machine learning. In *Machine learning and the internet of medical things in healthcare* (pp. 89-111). Academic Press.

Şahin, N. (2021). *Malware Detection Using Transformers-based Model GPT-2* (Master's thesis, Middle East Technical University).

TAHTACI, B., & CANBAY, B. (2020, October). Android malware detection using machine learning. In *2020 Innovations in Intelligent Systems and Applications Conference (ASYU)* (pp. 1-6). IEEE.

Tian, R., Batten, L., Islam, R., & Versteeg, S. (2009, October). An automated classification system based on the strings of trojan and virus families. In *2009 4th International conference on malicious and unwanted software (MALWARE)* (pp. 23-30). IEEE.

Tüfekçi, M., & Karpat, F. (2019). Derin Öğrenme Mimarilerinden Evrişimli Sinir Ağları (CNN) Üzerinde Görüntü İşleme-Sınıflandırma Kabiliyetininin Arttırılmasına Yönelik Yapılan Çalışmaların İncelenmesi. In *International Conference on Human-Computer Interaction, Optimization and Robotic Applications* (pp. 28-31).

Vinayakumar, R., Alazab, M., Soman, K. P., Poornachandran, P., & Venkatraman, S. (2019). Robust intelligent malware detection using deep learning. *IEEE Access*, *7*, 46717-46738.

Yang, F. J. (2018, December). An implementation of naive bayes classifier. In *2018 International conference on computational science and computational intelligence (CSCI)* (pp. 301-306). IEEE.

Yuxin, D., & Siyi, Z. (2019). Malware detection based on deep learning algorithm. *Neural Computing and Applications*, *31*(2), 461-472.

Zhu, L., & Spachos, P. (2021). Support vector machine and YOLO for a mobile food grading system. *Internet of Things*, *13*, 100359.

Zvornicanin, E. (2022). Difference between bidirectional and unidirectional LSTM. 6 Kasım 2022 tarihinde www.baeldung.com/cs/bidirectional-vs-unidirectional-lstm sitesinden erişildi.

Wang, W., Zhao, M., & Wang, J. (2019). Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *Journal of Ambient Intelligence and Humanized Computing*, *10*(8), 3035-3043.

Wong, T. T., & Yeh, P. Y. (2019). Reliable accuracy estimates from k-fold cross validation. *IEEE Transactions on Knowledge and Data Engineering*, *32*(8), 1586-1594.