USBTU
INTERNATIONAL JOURNAL OF SIVAS UNIVERSITY OF SCIENCE AND TECHNOLOGY
ULUSLARARASI SİVAS BİLİM VE TEKNOLOJİ ÜNİVERSİTESİ DERGİSİ
ISSN:
dergipark.org.tr
DOI:
Araştırma Makalesi / Research Article

Hiperparametreleri Ayarlanmış Makine Öğrenimi Algoritmalarını Kullanarak Android Sistemlerde Kötü Amaçlı Yazılım Tespiti

Abdulmuttalip DURAN[1] (0000-0003-1139-6314)

Halit BAKIR[2]* (0000-0003-3327-2822)

[1]Sivas Bilim ve Teknoloji Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, Bilgisayar Mühendisliği Bölümü, 58000, Sivas, Türkiye

[2]Sivas Bilim ve Teknoloji Üniversitesi, Lisansüstü Eğitim Enstitüsü, Savunma Teknolojileri Bölümü, 58000, Sivas,Türkiye

*Sorumlu yazar (Corresponding author): halit.bakir@sivas.edu.tr

Geliş Tarihi (Received): 19.05.2023 **Kabul Tarihi (Accepted):** 20.06.2023

Özet

Gelişen teknoloji ile birlikte günümüzde telefon, tablet gibi mobil cihazlar oldukça yaygınlaşmıştır. Bu cihazların yaygınlaşması ile birlikte cihazlar üzerine yapılan siber saldırılar da artmıştır. Mobil cihazlarla en yaygın kullanılan ve en çok siber saldırı yapılan işletim sistemi Android işletim sistemidir. Bu çalışmada makine öğrenme algoritmaları kullanılarak Android işletim sistemi için statik analiz tabanlı bir kötücül uygulama tespiti çalışması gerçekleştirilmiştir. Kullanılan makine öğrenme algoritmaları içerisinde en başarılı sonuç %99.36 doğruluk değeriyle SVM yönteminde elde edilmiştir. Veri setindeki sınıfların dengesiz dağılımı SMOTE algoritması kullanılarak suni veriler üretilerek dengelenmiştir. Makine öğrenme algoritmalarının doğruluk değerlerini artırmak için hiper parametre optimizasyonu uygulanmıştır. Optimizasyon algoritmalarından Grid Search metodu ile en iyi hiper parametreler tespit edilmiştir.

Anahtar Kelimeler: Android, kötücül uygulama, makine öğrenmesi, hiper parametre optimizasyonu

Malware Detection on Android Systems Using Machine Learning Algorithms with Hyperparameters Tuning

Abstract

With the developing technology, mobile devices such as phones and tablets have become very common nowadays. With the widespread use of these devices, cyber-attacks on devices have also increased. The most widely used and most cyber-attacked operating system with mobile devices is the Android operating system. In this study, a static analysis-based malicious application detection study has been conducted for the Android operating system using machine learning algorithms. The imbalance distribution of the classes in the dataset has been tackled using generating data samples using the SMOTE algorithm. Afterward, multiple machine learning algorithms have been trained and tested using the balanced dataset.

The obtained results show that the most successful results have been obtained using the SVM algorithm with an accuracy value of 99.36%. Finally, hyperyperparameter optimization has been applied to increase the performance of the used machine learning algorithms using three different algorithms namely Grid Search method, Random search, and Bayesian optimization algorithm. The best-optimized results have been obtained by using the Grid Search algorithm for tuning the hyperparameters of the machine learning algorithms.
Keywords: Android, malicious application, machine learning, hyper parameter tuning

---

GİRİŞ

Teknolojinin gelişmesiyle birlikte günümüzde telefon, tablet, bilgisayar gibi akıllı cihazlar hayatımızın bir parçası haline gelmiştir. Günümüzde alışverişten bankacılık işlemlerine kadar birçok işlem akıllı cihazlar sayesinde yapılabilmektedir. Akıllı cihazların bu kadar yaygınlaşması bu cihazlara yapılan siber saldırıların da yaygınlaşmasına sebep olmuştur. Günümüzde en yaygın kullanılan mobil işletim sistemleri Android, IOS ve Nokia Symbian dur (Arslan ve ark., 2017). Bu sistemler arasında en yaygın olanı Android işletim sistemidir. Dünya genelinde 3 milyardan fazla Android cihaz bulunmaktadır (Volkan, 2021) . Android işletim sistemi linux tabanlı açık kaynaklı bir işletim sistemidir (Kabakuş ve ark., 2014). Ayrıca Android işletim sistemi iOS vb. sistemlerde olan sadece kendi marketlerinden uygulama yükleme kısıtlaması bulundurmamaktadır. Kullanıcılar isterlerse 3. taraf mağazalardan veya doğrudan cihazlarına indirdikleri APK dosyaları üzerinden uygulama yüklemesi yapabilmektedir (Utku ve Doğru, 2017). Bu durum Android işletim sistemini kötücül yazılımlar için hedef haline getirmektedir (Bakour ve ark., 2019). Android sistemler buna karşılık güvenlik önlemi olarak yüklenen uygulamalara izin onay mekanizması eklemiştir (Dağlıoğlu ve Doğru, 2020). Bu mekanizma ile kullanıcılar uygulamayı yüklerken uygulamanın çalışmak için ihtiyaç duyduğu izinleri görebilmektedir. Ayrıca bu izinleri onaylayıp onaylamamak ta yine kullanıcıya bırakılmaktadır. Kötücül yazılımın cihaza zarar verme, cihazı ele geçirme, cihazdaki kullanıcı verilerini ele geçirmek gibi pek çok amacı olabilmektedir (Dinçer ve ark., 2021) . Kaspersky raporlarına göre 2019 ve 2020 yıllarında aylık 6 milyonun üzerinde mobil saldırı gerçekleşmiştir (Chebyshev, 2020). Bu kötücül yazılımlar hedeflerini gerçekleştirmek için cihaz izinlerine ihtiyaç duymaktadır. Bu projede uygulamaların istediği izinler, API'lar, Inetnt'ler ve bazı diğer önemli özellikleri makine öğrenme algoritmaları ile analiz edilerek test edilecek uygulamanın kötücül olup olmadığı tespit edilecektir.

Literatür özeti

Peynirci ve Eminağaoğlu (2020) yaptıkları çalışmalarda makine öğrenme algoritması ile kötücül yazılım tespiti için seçilen veri setlerindeki kötücül ve iyi huylu uygulamaların sayılarının birbirine yaklaştıkça doğruluk değerinin arttığını tespit etmişlerdir (Peynirci ve Eminağaoğlu, 2020).

Utku ve ark. (2017) yaptıkları çalışmada Android platformlardaki kötücül yazılım tespiti için izin tabanlı analiz yöntemi kullanmışlardır. Makine öğrenmesi tekniklerinden Naive Bayes ile %97,29 KNN ile de %97,74 oranında doğru sınıflandırma elde etmişlerdir (Utku ve Doğru, 2017).

Arslan ve ark. (2017) Android uygulamaları tersine mühendislik ile kaynak dosyası ve izin dosyası şeklinde parçalamışlardır. Uygulamanın çalışmak için istediği izinlerin ne kadarının kaynak dosya içerisinde kullanıldığını analiz eden bir java uygulaması geliştirmişlerdir. Bu uygulamada gereksiz izinlere bakılarak kötücül uygulama tespiti yapılmıştır. Çalışma sonucunda %97,62 oranında doğruluk elde edilmiştir (Arslan ve ark., 2017).

Karakuş ve ark. (2014) yaptıkları çalışmada mevcut Android kötücül yazılım tespit sistemlerinin Android uygulama sisteme yüklendikten sonra uygulamanın davranışlarına göre kötücül faaliyetlerini tespit ettiğini bunun ise güvenlik açısından yetersiz kaldığını ve daha güvenli bir koruma için uygulama sisteme yüklenmeden önce izin istekleri kullanılarak bir kötücül uygulama tespit sistemi geliştirilmesi gerektiğini savunmuşlardır (Kabakuş ve ark., 2014).

Utku ve Doğru (2017) 18 farklı kötücül yazılım tespit aracının kapsamlı araştırmasını yapmışlardır ve karşılaştırmaları sonucu en kapsamlı yazılım tespit aracını DREBIN olduğunu belirlemiştir (Utku ve Doğru, 2017).

Dinçer ve Doğru (2017) yaptıkları Android kötücül yazılım tespit sistemleri araştırmasında kötücül yazılım tespitindeki analiz yöntemlerini araştırmıştır. Araştırma sonuçlarına göre dinamik analiz yöntemi sadece yürütme esnasındaki uygulamaları izleyebildiği için potansiyel zafiyetleri tespit edememektedir. Statik analizin ise uygulama yüklenmeden önce tespit yaptığı için potansiyel zafiyeti tespit edebilmektedir fakat çalışma anındaki davranışı tespit edememektedir. İkisinin birleşiminden oluşan hibrit model ise ikisini de tespit edebilmektedir ama uygulaması daha maliyetlidir (Dinçer ve Doğru, 2017).

Dağlıoğlu ve Doğru (2020) yapmış oldukları araştırmada statik analiz, dinamik analiz ve hibrit analiz yöntemlerinin kullanımını ve başarı oranlarını incelemişler ve incelemelerinin sonucuna göre en iyi başarı %97,87 ile statik analiz olduğunu tespit etmişlerdir (Dağlıoğlu ve Doğru, 2020).

Aydın ve ark. (2018) yaptıkları makine öğrenmesi ile kötücül yazılım tespiti uygulamasında DVM, Random forest, naive bayes ve K en yakın komşu algoritmalarını kullanmışlardır. En başarılı sonucu random forest algoritması kullanılarak elde etmiş olup %95,65 ulaşmıştır (Aydın ve ark., 2018).

Han ve ark. (2016) izin isteklerini ve sistemsel olayları analiz eden bir çalışma yapmışlardır ve IPBD metoduyla true positive ratio da %93,07 oranında başarı elde etmişlerdir (Han ve ark., 2016).

Rani ve Dhindsa (2020) yaptığı çalışmada play storede yükülü olan uygulamaların %12,9 unu, üçüncü parti mağazalarda yüklü uygulamaların %28 inin kötücül yazılım olduğunu tespit etmişlerdir (Rani ve Dhindsa, 2020).

Singh ve ark. (2016) statik ve dinamik analiz yöntemlerini kullanarak kötücül yazılım tespit sistemi geliştirmişlerdir. Yaptıkları sistemde Android APK dosyasını, ".class" dosyasına dönüştürüp kodlarını analiz etmişlerdir. Daha sonra içerisine takip için gerekli yazılımı entegre edip tekrar APK ya çevirip izlemesini yapmışlardır (Singh ve ark., 2016).

Cho ve ark. (2020) statik analiz yöntemini kullanarak kötücül yazılım tespitini araştırmışlardır. Yaptıkları araştırmada her sistemin kendine göre avantajı ve sınırları olduğunu tespit etmişlerdir ve veri tabanlarını sürekli güncelleyerek önemli bir başarı elde edilebileceğini savunmuşlardır (Cho ve ark., 2020).

Li ve ark. (2018) izin tabanlı tespit sisteminde tüm izinleri denetlemek yerine sadece 22 tane izini denetlemenin daha avantajlı olduğunu savunmuşlardır. Yaptıkları çalışmada destek vektör makinesini kullanmışlardır %93.62 başarı elde etmişlerdir (Li ve ark., 2018).

Akcayok (2018a) yaptıkları çalışmada; geliştirdikleri birçok katmanlı algılayıcı ve izin tabanlı kötücül yazılım tespit çalışması yapmışlardır. Kötücül yazılımları %95,66 oranında doğru tespit etmişlerdir (Akcayol, 2018a).

Hasan ve ark. (2021) yaptıkları çalışmada MEGDroid olarak adlandırılan bir kötücül yazılım tespiti gerçekleştirmişlerdir. Bu yöntemde APK dosyasını kaynak kodlarına ayırmışlardır. Kötücül yazılım tespitini kaynak dosyaları üzerinden yapmışlardır (Hasan ve ark., 2021).

Mehtab ve ark. (2020) Yaptıkları çalışmada kötücül yazılım tespitini kural tabanlı analiz etmişlerdir. %99,11 oranında doğru tahminleme yapmışlardır (Mehtab ve ark., 2020).

Gao ve ark. (2021) android sistemlerde kötücül yazılım tespiti için GDroid adında bir prototip sistem geliştirmişlerdir. Geliştirdikleri sistem de Grafik sinir ağı yöntemi kullanmışlardır. Kötücül uygulamaların %98,99 unu düşük yanlış pozitif oranında doğru tespit etmişlerdir (Gao ve ark., 2021).

Raghuraman ve ark. (2020) yaptıkları çalışmada dinamik ve statik analiz yöntemlerini kullanmışlardır. Dinamik kötücül yazılım analiz yönteminde 94,64 statik kötücül yazılım analiz yönteminde ise %99,36 oranında başarı elde etmişlerdir (Raghuraman ve ark., 2020).

Şahin ve ark. (2021) yaptıkları çalışmada kötücül uygulama tespiti yapmak için uygulamanın istediği izinleri analiz etmişlerdir. Çalışmalarında temel bileşen analiz yöntemi ve doğrusal ayırıcı yöntemi kullanarak boyut küçültme yapmışlardır. Doğrusal ayrıcı yöntemi ve NB makine öğrenmesi tekniği kullandıkları VirusShare data setinde %99 oranında başarı elde etmişlerdir (Şahin ve ark., 2021).

Bakour ve Ünver yaptıkları çalışmada görüntü tabanlı özelliklerin derin öğrenme teknikleriyle hibridleştirilmesine dayalı olarak android kötü amaçlı yazılım örneklerini tespit etmek için DeepVisDroid adlı yeni bir hibrit derin öğrenme modeli önermişlerdir. Yaptıkları çalışmada %98 den yüksek doğruluk değerinde başarı elde etmişlerdir (Bakour ve Ünver, 2021).

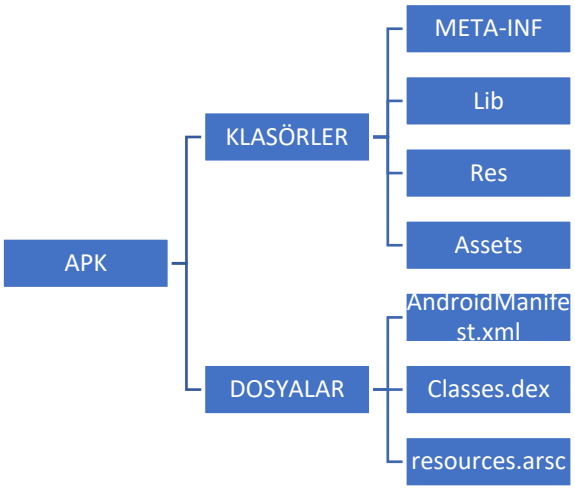
Ünver ve Bakour yaptıkları çalışmada Android uygulamaların kaynağındaki dosyaların gri tonlamalı görüntülere dönüştürülmesine dayanan kötücül uygulama tespit sistemi geliştirmişlerdir. Yaptıkları çalışmada %98,75 doğruluk değerine ulaşmışlardır (Ünver ve Bakour, 2020).

Ünver ve Bakour (2020) yaptıkları çalışmada Android kötücül uygulamaları tespit etmek için yeni bir genel görüntü tabanlı sınıflandırma yöntemi olan VisDroid önermiş ve geliştirmişlerdir. Yaptıkları çalışmada %98,14 doğruluk değerine ulaşmışlardır (Ünver ve Bakour, 2020).

MATERYAL VE METOT

Android mimarisi

Android işletim sistemi açık kaynaklı ve linux tabanlı bir mobil işletim sistemidir. Andorid işletim sisteminde çalışan mobil uygulamalar .apk veya .aab formatında sıkıştırılmış dosyalardır. ".aab" formatı sadece Play store ye uygulama yüklerken kullanılmaktayken APK formatı harici olarak Android cihazlara uygulamayı yüklemek için kullanılmaktadır. Sıkıştırılmış Android dosyaları şekil 1 gösterdiği gibi META-INF, lib, res, assets klasörlerinden ve AndroidManifest.xml, Classes.dex, resources.arsc dosyalarından oluşmaktadır.



Şekil 1. Apk dosya bileşenleri

Bu dosya bileşenlerinden Android Manifest dosyasının içerisinde Android uygulamanın çalışmak için istediği izinler bulunmaktadır. Bu projede makine öğrenmesi yöntemleri kullanılarak uygulamanın istediği izinler, kullanılan Intent'ler, kod içinde kullanılan API'lar

gibi özellikler doğrultusunda kötücül yazılım tespiti yapılmıştır.

Veri setini hazırlama

Projede KAGGLE' den alınan 15036 kayıt ve 215 öznitelikten oluşan veri seti kullanılmıştır. Veri setinin orijinali Süleyman Yerima tarafından, Drebin veri tabanından (Arp ve ark., 2014) alınan kötücül uygulamalara, iyi huylu uygulamalar eklenilerek ve ardından statik öznitelikler çıkarılarak hazırlanmıştır (Yerima ve Sezer, 2018). Data setindeki öz nitelikler, uygulamanın istediği izinler, API Çağrı İmzaları, niyetler (Intentler), ve Komutlar İmzalar gibi uygulamanın davranışı açıklayan statik özellikleri içermektedir. Uygulamada kullanılan özellikler için 1, Uygulamada kullanılmayan özellikler için ise 0 değeri kullanılmıştır. Uygulama tarafından kullanılabilecek izinlerin bazıları, Google tarafından tehlikeli izin olarak tanımlanmıştır (Dinçer ve ark., 2021). Google tarafından tanımlanan tehlikeli izinler Çizelge 1 de gösterilmiştir. Veri seti içerisinde, Drebin verilerinden alınan 5560 kötü amaçlı yazılım ve Google play'dan indirilen 9476 iyi huylu yazılım bulunmaktadır.

Çizelge 1. Öznitelik açıklamaları

READ_CALENDAR	Cihaz takvimine erişim sağlayan izin
WRITE_CALENDAR	
CAMERA	Cihaz kamerasına erişim sağlayan izin
READ_CONTACTS	Cihaz kişi listesine erişim sağlayıp mevcut kişileri görmeye, düzenlemeye ve yeni kişi eklemeye erişim sağlayan izin
WRITE_CONTACTS	
GET_ACCOUNTS	
ACCESS_FINE_LOCATION	Cihazın yaklaşık ve kesin konumunu görmeye erişim sağlayan izin
ACCESS_COARSE_LOCATION	
RECORD_AUDIO	Cihaz mikrofonuna erişim sağlayan izin
READ_PHONE_STATE	Cihazın arama kayıtlarınaerişim sağlayan izinler
CALL_PHONE	
READ_CALL_LOG	
WRITE_CALL_LOG	
ADD_VOICEMAIL	
USE_SIP	
PROCESS_OUTGOING_CALLS	
BODY_SENSORS	Cihazın vücut sensörlerine erişim sağlayan izin
SEND_SMS	Mesaj alma, okuma, kaydetme vb. SMS ve MMS işlemlerine erişim sağlayan izin
RECEIVE_SMS	
READ_SMS	
RECEIVE_WAP_PUSH	
RECEIVE_MMS	
READ_EXTERNAL_STORAGE	Cihaz depolamasına erişim sağlayan izinler
WRITE_EXTERNAL_STORAGE	

Veri setine makine öğrenmesi işlemi uygulanmadan önce veri setinde bazı düzenlemeler yapılmıştır. Bu düzenlemeler Python programlama dili kullanılarak gerçekleştirilmiştir. İlk olarak data set .csv dosyası Pandas Python kütüphanesi kullanılarak açılmıştır. Daha sonra veri setindeki eksik değerlerin kontrolü yapılmıştır, herhangi bir eksik veriye rastlanılmamıştır. Daha sonrasında veri setindeki özniteliklerin anomalilerine bakılmıştır ve tüm öznitelikler 0 veya 1'den oluştuğu için herhangi bir anomali değeri bulunmamaktadır. Kalan verinin sınıf değişkeninde şekil 2'de gösterildiği gibi ortalama %63 oranında iyi huylu %36 oranında ise kötü huylu uygulama verisi olduğu tespit edilmiş olup bu dengesizlik SMOTE metodu ile sentetik veriler üretilerek dengelenmiştir. SMOTE (Synthetic Minority Over-Sampling Technique) metodu, sentetik veriler üretmek için kullanılan bir yöntemdir. Bu yöntemin ana işlevi, örnek sayısı az olan sınıfa örnekler arasında K-en yakın komşu algoritması kullanarak ve belirli işlemler gerçekleştirerek yeni örnekler oluşturmaktır (Yavaş ve ark., 2020). Veri seti dengeleme işlemi yaptıktan sonra, veriler %80 eğitim %20 test verisi olarak ayrılmıştır.

Şekil 2. Veri seti dağılımı

Makine öğrenmesi kullanılarak verilerin sınıflandırılması

Veri seti hazırlandıktan sonra verileri analiz etmek için makine öğrenmesi algoritmaları çalıştırılır. Bu algoritmalar ile uygulamaların kötücül veya iyi huylu olma durumları tespit edilir. Eğitim veri seti kullanılarak makine öğrenmesi algoritmaları eğitilir daha sonra test veri seti üzerinden algoritmaların tespit doğruluğu test edilir.

Projede kullanılan makine öğrenmesi algoritmaları şunlardır:

Logistic regression: Logistic regression, sınıflandırma problemlerini çözmek için kullanılan bir algoritmadır. Bu algoritma kategorik değişkenler ile çalışır. Örneğin 0 veya 1, evet veya hayır, doğru veya yanlış vb. ayrık küme kategorilerini sınıflandırır. Olasılık kavramı üzerinde çalışan bir tahmine dayalı analiz algoritmasıdır.

Algoritmanın matematiksel ifadesi:

$$f\,(x)= \frac{1}{1+e^{-x}} \qquad (1)$$

Decision trees: Tıpkı insanlar gibi karar vermek için bir dizi kurallar kullanan denetimli bir makine öğrenmesi algoritmasıdır. Karar ağaçları, sınıflandırma ve tahminleme de kullanılan en popüler algoritmadır. Karar ağaçları akış şeması benzeri bir ağaç yapısındadır. Burada her dahili düğüm bir öznitelik üzerindeki testi, her dal testin sonucunu, yapraklar ise sınıfı belirtir.

Random forest: Popüler bir makine öğrenimi algoritmasıdır. Hem sınıflandırma hem de tahminleme yapmak için kullanılmaktadır. Karmaşık sorunları çözmek için kullanılan birden çok sınıflandırıcıyı birleştiren topluluk öğrenme kavramına dayalı bir algoritmadır. Random Forest, rastgele olarak birçok Decision Trees algoritması kullanır, bu algoritmaların sonuçlarını toplayarak problemin çözümüne karar verir.

Support vector machines (SVM): Bir başka popüler sınıflandırma ve tahminleme algoritması destek vektör makinasıdır. SVM algoritmasının amacı, gelecekteki yeni veri noktasını doğru sınıfa yerleştirmek için n boyutlu uzayı sınıflara ayırabilecek en iyi çizgiyi veya karar sınırını oluşturmaktır.

K-Nearest neighbors (KNN): Hem sınıflandırmada hem de tahminlemede kullanılan bir denetimli makine öğrenme algoritmasıdır. KNN, test verileri ile tüm eğim noktaları arasındaki mesafeyi hesaplayarak test verileri için doğru sınıfı tahmin etmeye çalışır. İlk olarak test verilerine en yakın olan K tane noktayı seçer. Sonrasında eğitim verisi sınıflarına ait test verilerinin olasılıklarını hesaplar ve olasılığı en yüksek tutan sınıfı seçer.

Naive bayes: Sınıflandırma algoritmaları arasında en bağımsızlık varsayımı olan algoritmadır. Naive Bayes algoritması, bir sınıfta belli bir özelliğin varlığının başka herhangi bir özelliğin varlığıyla ilişkisiz olduğunu varsayar. Yani her bir özellik birbirinden bağımsız kabul edilirken hepsi sonuç ile bağlantılı olarak kabul edilir.

Algoritmanın matematiksel ifadesi:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \qquad (2)$$

Gradient boosting (GBM): Hem tahminlemede hem de sınıflandırmada kullanılan bir topluluk makine öğrenme algoritmasıdır. GBM, güçlü bir makine öğrenme algoritması oluşturmak için bazı zayıf öğrenme algoritmalarını birleştirerek, yükseltme tekniğini kullanır. Her biri seri halinde olan bir ağaç önceki ağaçlar tarafından hesaplanan hatalar üzerine kurulur.

Xgboost: Hız ve performans için tasarlanmış olan Xgboost, Gradient Boosting karar ağaçlarının bir uygulamasıdır.

Bootstrap aggregation (Bagging): Sınıflandırma veya tahminleme problemleri için

Overfitting'i çözmeye çalışan bir birleştirme yöntemidir. Bagging, makine öğrenimi algoritmalarının doğruluğunu ve performansını iyileştirmeyi amaçlar.

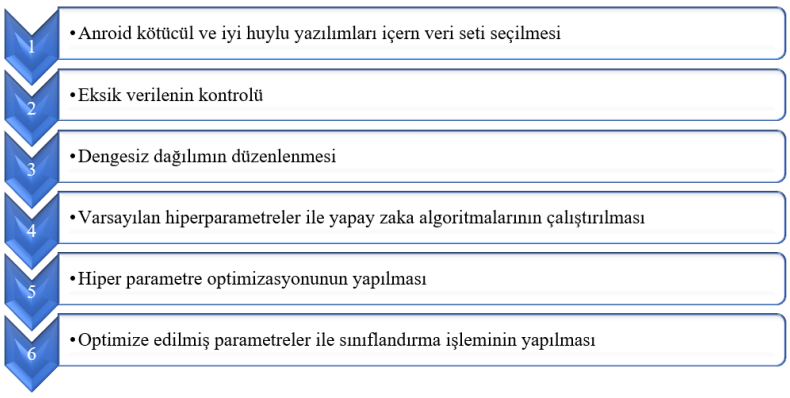
Hiperparametre optimizasyonu

Yapılan çalışmada, makine öğrenmesi algoritmalarının doğruluk değerini artırmak için algoritmaların değişkenleri değiştirilmiştir. En verimli değişken değerini tespit etmek için hiperparametre optimizasyonu yapılmıştır. hiperparametre optimizasyonu gerçekleştirmek için "Grid Search", "Random Search" ve "Bayesian" gibi optimizasyon algoritmaları kullanılabilmektedir.

Grid search: Bu algoritma en temel ve en yavaş çalışan hiper parametre tespit algoritmasıdır. Bu teknikte sağlanan tüm hiper parametre değerleri teker teker denenerek en iyi sonucu veren hiper parametre değerleri seçilir. Tüm parametreleri denendiği için çok yavaş çalışır ana en doğru tespiti yapar.

Random search: Bu algoritma da Grid Search algoritmasına benzerdir fakat tüm parametreleri sırayla denemek yerine rasgele olarak parametreler seçerek dener. Parametrelerin tamamını denemediği için oldukça hızlı çalışır fakat en iyi parametreleri kaçırabileceği için Grid Search kadar doğru tespit yapamaz.

Bayesian optimizasyon algoritması: Doğruluk ve zaman tasarrufu açısından en doğru algoritmadır. Rastgele veya sırayla parametre aramak yerine hiper parametreleri geçmiş değerleri kullanarak oluşturduğu olasılık fonksiyonunu kullanarak tespit eder. Bu sayede bir sonraki parametre setini rastgele seçmek yerine, algoritma seçimini optimize eder ve en iyi parametre setini en kısa sürede tespit eder. Bu çalışmada kullanılan makine öğrenme algoritmaları çok fazla hiper parametrelere sahip olmadığından dolayı ve en iyi sonuçlara ulaşabilmek için Grid search algoritması tercih edilmiştir. Çalışmada data sete yapılan işlemlerin diyagramı Şekil 3 de gösterilmiştir.



Şekil 3. Yapılan işlemlerin diyagramı

BULGULAR VE TARTIŞMA

Bu çalışma i5 işlemcili ve 8GB Ram'e sahip bir bilgisayarda Python programlama dili kullanılarak gerçekleştirilmiştir. Projede Pandas, Seaborn, Numpy, Sklearn, İmblearn, Xgboost, Matplotlib ve Optuna kütüphaneleri kullanılmıştır. Data seti üzerinde ön işleme işlemleri yapılmıştır. Bu ön işlemler ile data setindeki eksik verilerin olduğu satırlar temizlenmiştir dengesiz sınıf dağılımı SMOTE algoritması kullanılarak yapay verileri oluşturularak dengelenmiştir. Ön işleme sonucunda 9476 adet kötücül ve 9476 adet iyi huylu sınıftan oluşan bir data seti oluşturulmuştur. Bu şekilde ön işleme aşaması tamamlanmış olup veri seti, makine öğrenmesi algoritmaları uygulamak için hazırlanmıştır. Analiz sonuçlarını değerlendirmek için doğruluk, kesinlik, duyarlılık ve F1-skor değerlerine bakılmıştır. Bu değerlendirme metrikleri şu şekilde açıklanmaktadır.

	Tahminlenen (Predicted)	
Gerçekleşen (Actual)	True Pozitives (TP)	False Negatives (FN)
	False Pozitives (FP)	True Negatives (TN)

Accuracy (Doğruluk): Bir modelin başarısını ölçmek için çok kullanılan bir metriktir. Doğruluk değeri modelde doğru tahmin edilen alanların toplam veri kümesine oranı ile hesaplanmaktır.

$$\frac{TP+TN}{TP+FP+TN+FN} \text{ (Doğruluk)} \qquad (3)$$

Precision (Kesinlik): Pozitif olarak tahminlenen değerlerin gerçekten kaç tanesinin pozitif olduğunu göstermektedir. Kesinlik değeri özellikle yanlış pozitif tahminlemenin maliyeti yüksek olduğu durumlarda çok önemlidir.

$$\frac{TP}{TP+FP} \text{ (Keskinlik)} \qquad (4)$$

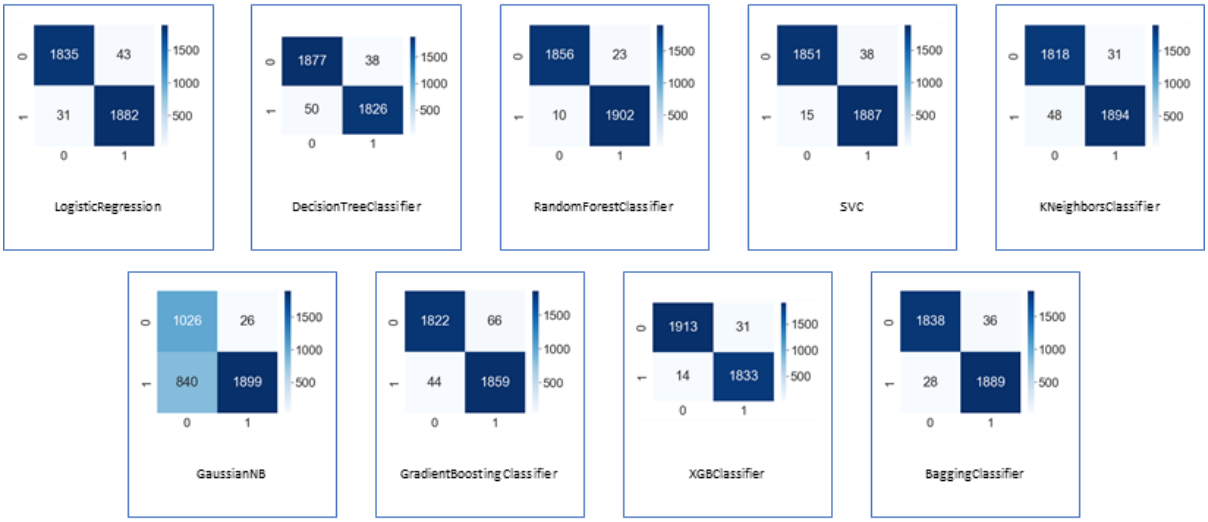
Recall (Duyarlılık): Pozitif olarak tahmin etmemiz gereken verilerin ne kadarını pozitif olarak tahmin ettiğimizi gösteren bir metriktir. Duyarlılık değeri de yanlış negatif olarak tahminlemenin maliyetinin yüksek olduğu durumlarda bize yardımcı olacak bir metriktir. Mümkün olduğunca yüksek olması gereklidir.

$$\frac{TP}{TP+FN} \text{ (Duyarlılık)} \qquad (5)$$

F1 Score (F1 Skor): Kesinlik (Precision) ve Duyarlılık (Recall) değerlerinin harmonik ortalamasını göstermektedir.

$$2x \frac{kesinlik\ x\ doğruluk}{kesinlik+doğruluk} \text{ (F1 skor)} \qquad (6)$$

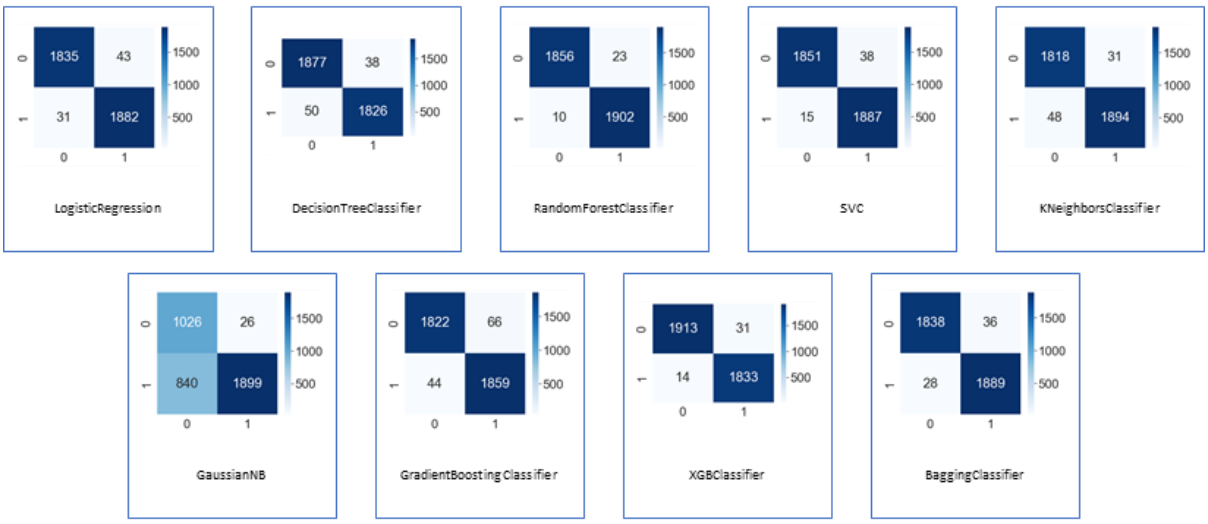
Makine öğrenmesi algoritmaları varsayılan değerlerinde kullanıldıkları zaman analiz sonuçları



de gösterilmiştir. Makine öğrenmesi algoritmaları varsayılan hiper parametreleri ile çalıştırıldığında en başarılı sonuç %99.12 doğruluk değeriyle Random Forest algoritması kullanılarak elde edilmiştir. GaussianNB algoritması haricindeki denenen tüm algoritmalarda ise sonuçlar %97'nin üzerindedir. Şekil 4 te, kullanılan makine öğrenmesi algoritmalarının konfüzyon matrisi gösterilmiştir.

Çizelge 2. Varsayılan hiper parametreler ile sonuçlar

Algoritma	Doğruluk	Kesinlik	Duyarlılık	F1_Skor	Çapraz Geçerlilik
LogisticRegression	98.04%	98.37%	97.76%	98.07%	97.91%
DecisionTreeClassifier	97.67%	97.33%	97.96%	97.64%	97.94%
RandomForestClassifier	**99.12%**	**99.47%**	**98.80%**	**99.13%**	**99.04%**
SVC	98.60%	99.21%	98.02%	98.61%	98.58%
KNeighborsClassifier	97.91%	97.52%	98.38%	97.95%	98.20%
GaussianNB	77.15%	69.33%	98.64%	81.43%	76.86%
GradientBoostingClassifier	97.09%	97.68%	96.57%	97.12%	97.34%
XGBClassifier	**98.81%**	**99.24%**	**98.33%**	**98.78%**	**99.16%**
BaggingClassifier	98.31%	98.53%	98.12%	98.33%	98.32%



Şekil 4. Optimize öncesi konfüzyon matrisi

Bu konfüzyon matrisi incelendiğinde de Random Forest algoritmasının en iyi sonuçları elde etmiş olup, yalnızca 10 tane kötücül yazılımı ve 23 iyi huylu yazılımı yanlış tahmin etiğini ortaya çıkmıştır. %95'in üzerinde başarı gösteren 8 algoritmanın Gridsearch yöntemiyle hiper parametreleri optimize edilmiştir. Optimizasyon sırasında kullanılan hiper parametreler, her hiperparametre için kullanılan değer aralığı ve tespit edilen en iyi hiper parametreler Çizelge 3'te gösterilmiştir. Bu optimize edilmiş hiper parametreler ile analizler yapılmıştır. Analiz sonuçları **Hata! Başvuru kaynağı bulunamadı. Hata! Başvuru kaynağı bulunamadı.** de gösterilmiştir.

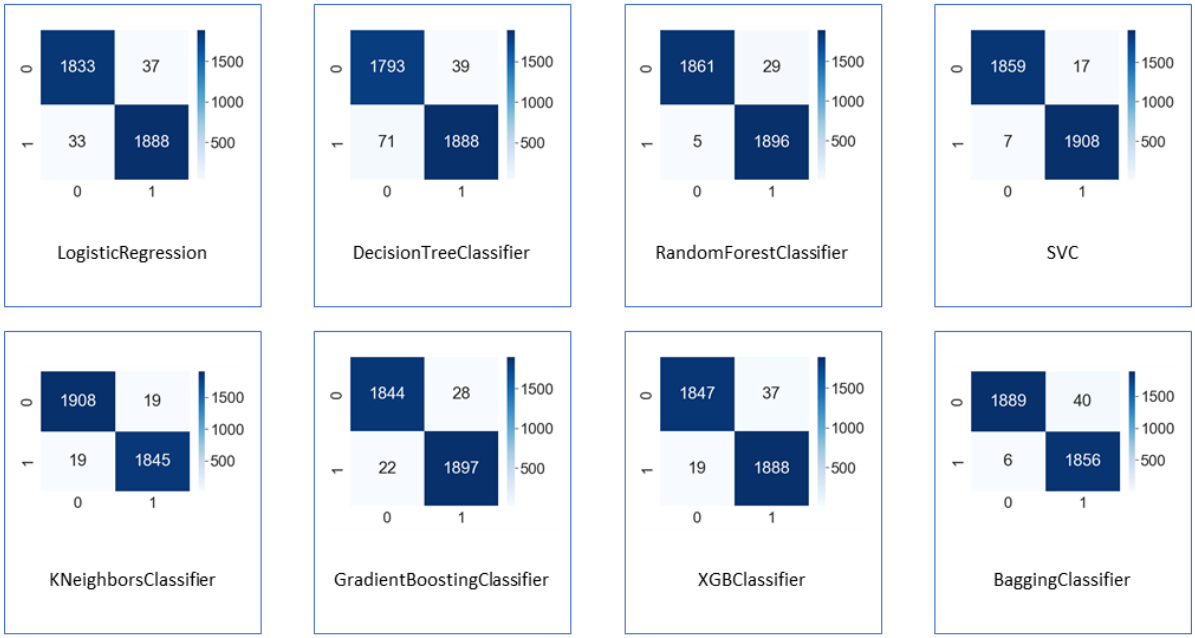
Hiper parametre optimizasyonu yapıldıktan sonra LogisticRegression, SVC, KNeighborsClassifier, GradientBoostingClassifier ve BaggingClassifier algoritma sonuçlarının doğruluk değerlerinde artış gözlemlenmiştir. Çizelge 4'de gösterildiği gibi hiper parametre optimizasyonu sonucunda %99.36 doğruluk değeriyle en başarılı sonuç SVC algoritmasında elde edilmiştir. Şekil 5 da optimizasyon sonrası makine öğrenmesi algoritmalarının konfüzyon matrisi gösterilmiştir. Bu konfüzyon matrisine göre sadece 7 tane kötücül yazılım SVC algoritması tarafından iyi huylu uygulama olarak tespit edilmiştir. Diğer taraftan, SVC tarafından sadece 17 iyi huylu uygulama yanlışlıkla kötücül olarak tahmin edilmiştir. Kötücül uygulama tespit konusunda, Random Forest ve Bagging sınıflandırıcılar da üstün sonuçlar vermişlerdir. konfüzyon matrislere baktığımızda bu iki sınıflandırıcı tarafından sıra ile sadece 5 ve 6 kötücül uygulaması yanlışlıkla iyi huylu olarak sınıflandırılmıştır. Son olarak, bu çalışmada elde edilen sonuçları, literatürde bulunan daha önceki çalışmalar ile karşılaştırılmıştır, bu karşılaştırma çalışması çizelge 5'te gösterilmiştir.

Çizelge 3. Hiper parametre Optimizasyon işlemin konfigürasyonu ve sonuçları

Algoritma	Hiper parametre	Varsayılan değer	Optimizasyon aralığı	En iyi değer
Logistic Regression	C	1.0	100, 10, 1.0, 0.1, 0.01	100
	solver	lbfgs	newton-cg, lbfgs, liblinear	newton-cg
Decision Tree Classifier	criterion	gini	gini, entropy	entropy
	max_depth	None	2, 3, 5, 10, 17	17
	max_features	None	log2, sqrt, auto	aqrt
	min_samples_leaf	1	1,5,8,11	1
	min_samples_split	2	2, 3, 5, 7, 9	2
	random_state	None	0,1,2,3,4,5	2
	splitter	best	Best, random	random
Random Forest Classifier	max_depth	None	3,5,10,None	None
	max_features		1,3,5,7	1
	min_samples_leaf	2	1,2,3	1
	min_samples_split	2	1,2,3	3
	n_estimators	100	10,100,200	100
SVC	C	1	0.1, 1, 10, 100, 1000	10
	gamma		1, 0.1, 0.01, 0.001, 0.0001	0.2
KN eighbors Classifier	metric	minkowski	euckidean, manhattan	manhattan
	n_neighbors	5	3,5,11,19	3
	weights	uniform	uniform, distance	distance
Gradient Boosting Classifier	learning_rate	0.1	0.0001, 0.001, 0.01, 0.1, 1.0	0.1
	max_depth	3	3, 7, 9	3
	n_estimators	100	10, 50, 100, 500	500
	subsample	1.0	0.5, 0.7, 1.0	1.0
XGB Classifier	colsample_bytree	0.75	0.75, 1	0.75
	learning_rate	0.3	0.1, 0.01	0.1
	max_depth	6	2, 6	6
	min_child_weight	1	1, 5	1
	subsample	1	0.75, 1	0.75
Bagging Classifier	base_estimator	deprecated	GradientBoostingClassifier, KNeighborsClassifier, SVR, RandomForestClassifier, DecisionTreeClassifier	RandomForestClassifier
	max_features	1	0.5, 2,4,6	0.5
	max_samples	1	0.5, 0.75, 1.0	1.0
	n_estimators	1	20, 50 ,75, 100, 200	20

Çizelge 4. Optimize sonrası sonuçlar

Algoritma	Eski	Doğruluk	Kesinlik	Duyarlılık	F1_Skor	Çapraz
LogisticRegression	98.04%	98.15%	98.28%	98.07%	98.17%	97.79%
DecisionTreeClassifier	97.67%	97.09%	96.37%	97.97%	97.16%	97.73%
RandomForestClassifier	**99.12%**	**99.10%**	**99.84%**	**98.38%**	**99.11%**	**99.03%**
SVC	98.60%	**99.36%**	**99.63%**	**99.11%**	**99.37%**	**99.28%**
KNeighborsClassifier	97.91%	98.99%	98.98%	98.98%	98.98%	98.85%
GradientBoostingClassifier	97.09%	98.68%	98.85%	98.54%	98.69%	98.95%
XGBClassifier	**98.81%**	98.52%	99.00%	98.07%	98.53%	98.73%
BaggingClassifier	98.31%	98.78%	99.67%	97.89%	98.77%	98.90%



Şekil 5. Optimizasyon sonrası konfüzyon matrisi

Çizelge 5. Önceki çalışmalar ile karşılaştırma çizelgesi

Çalışma	**Veri setinin kaynağı**	**Kullanılan yöntem**	**Hiper parametre optimizesi**	**Doğruluk değeri**
Bu çalışma	**Drebin + Google play**	**SVM**	**var**	**%99.36**
(Utku & Doğru, 2017)	Drebin + Google play	KNN	var	%97.74
(Akcayol, 2018)	Drebin + Google play	Çok katmanlı algılayıcı	yok	%95.66
(Li ve ark., 2018)	Google Play + Anzhi Store	SVM	yok	%93.62
(A. Utku & Doğru, 2017)	Drebin + Google play	Naive Bayes	yok	%97,29
(AYDIN ve ark., 2018)	Android Malware Genome + Google play	Random Forest	yok	%95,65

SONUÇLAR VE GELECEKTEKİ ÇALIŞMALAR

Bu projede Kaggle den alınan "Makine Öğrenimi için Android Kötü Amaçlı Yazılım Veri Kümesi" data seti üzerinde makine öğrenmesi algoritmaları kullanılarak Android kötücül uygulama tespiti işlemi yapılmıştır. Ham data seti üzerinde eksik veri kontrolü ve sınıf dengeleme ön işlemleri yapılmıştır. Homojen olmayan sınıf verisi SMOTE algoritması ile suni veriler üretilerek dengelenmiştir. Makine öğrenme algoritmalarının varsayılan hiper parametreleri ile analiz yapıldığında en başarılı yöntem %99.12 doğruluk değeriyle Random Forest sınıflandırıcısı kullanılarak elde edilmiştir. Grid search algoritması kullanılarak hiper parametre optimizasyonu yapıldıktan sonra analiz yapıldığında en başarılı sonuç %99.36 'ya ulaşarak SVM algoritması kullanılarak elde edilmiştir. İleriki çalışmalarda veri setine yeni uygulamalar eklenerek veri seti zenginleştirilecektir. Ayrıca makine öğrenmesi

optimizasyonunda yeni parametreler denenerek sonuçlar iyileştirilecektir. Ayrıca, derin öğrenme teknikleri uygulayarak klasik makine öğrenmesi algoritmalar ile de karşılaştırılacaktır.

Teşekkür

Data seti için kötücül uygulama verilerini paylaştığı için Arp vd. (Arp ve ark., 2014) ye teşekkür ederiz. Data setini hazırladığı, statik öznitelikleri çıkarttığı ve data setini ücretsiz olarak paylaştığı için Yerima ve ark. (Yerima & Sezer, 2018)'na teşekkür ederiz.

KAYNAKÇA

Akcayol, M.A. 2018a. Çok katmanlı algılayıcı ile izin tabanlı android kötücül yazılım tespiti permission based android malware detection with multilayer perceptron. 0–3.

Akcayol, M.A. 2018b. Çok katmanlı algılayıcı ile izin tabanlı android kötücül yazılım tespiti permission based android malware detection with multilayer perceptron. 0–3.

Arp, D., Spreitzenbarth, M., Hübner, M., Gascon, H., Rieck, K. 2014. Drebin: effective and explainable detection of android malware in your pocket. Vol. 14 pp.23-26.

Arslan, R.S., Doğru, İ.A., Barışçı, N. 2017. Android mobil uygulamalar için izin karşılaştırma tabanlı kötücül yazılım tespiti. Journal of Polytechnic, 20(1): 175–189.

Aydın, A., Doğru, İ. A., Dörterler, M. 2018. Makine öğrenmesi algoritmalarıyla android kötücül yazılım uygulamalarının tespiti. Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 22(2): 1087.

Bakour, K., Ünver, H.M. 2021. DeepVisDroid: android malware detection by hybridizing image-based features with deep learning techniques. Neural Computing and Applications, 33(18): 11499–11516.

Bakour, K., Ünver, H.M., Ghanem, R. 2019. The Android malware detection systems between hope and reality. SN Applied Sciences, 1(9): 1–42.

Chebyshev, V. 2020. Mobile malware evolution 2020 | Securelist.

Cho, J.Y., Ko, E.G., Yoo, H.B., Cho, M.R., Seo, C.J. 2020. A Study on Malware Detection System Using Static Analysis and Stacking. The Transaction of the Korean Institute of Electrical Engineers P, 69P(3): 187–192.

Dağlıoğlu, A., Doğru, İ.A. 2020. Android işletim sisteminde kötücül yazılım tespit sistemleri. DÜMF Mühendislik Dergisi, 11(2): 499–511.

Dinçer, A.E., Temel, S.C., Öztürk, S.M. 2021. Safranbolu-İncekaya Bölgesi'nde bir mimari stüdyo deneyimi. Düzce Üniversitesi Bilim ve Teknoloji Dergisi, 9(1): 278-292.

Dinçer, C.A., Doğru, İ.A. 2017. Android kötücül yazılım tespiti yaklaşımları. Uluslararası Bilgi Güvenliği Mühendisliği Dergisi, 2: 48–58.

Gao, H., Cheng, S., Zhang, W. 2021. GDroid: Android malware detection and classification with graph convolutional network. Computers and Security, 106: 102264.

Han, H., Li, R., Gu, X. 2016. Identifying malicious Android apps using permissions and system events. International Journal of Embedded Systems, 8(1): 46–58.

Hasan, H., Tork Ladani, B., Zamani, B. 2021. MEGDroid: A model-driven event generation framework for dynamic android malware analysis. Information and Software Technology, 135(July 2020).

Kabakuş, A.T., Doğru, İ.A., Çetin, A. 2014. Android kötücül yazılım tespit ve koruma sistemleri. Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi, 31(1): 9–16.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-An, W., Ye, H. 2018. Significant permission ıdentification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics, 14(7): 3216–3225.

Mehtab, A., Shahid, W. Bin, Yaqoob, T., Amjad, M.F., Abbas, H., Afzal, H., Saqib, M.N. 2020. AdDroid: Rule-Based machine learning framework for android malware analysis. Mobile Networks and Applications, 25(1): 180–192.

Peynirci, G., Eminağaoğlu, M. 2020. Android platformunda kötücül yazılım tespiti: literatür incelemesi. Bilişim Teknolojileri Dergisi, 65–76.

Raghuraman, C., Suresh, S., Shivshankar, S., Chapaneri, R. 2020. Static and dynamic malware analysis using machine learning. Advances in Intelligent Systems and Computing, 1045, 793–806.

Rani, S., Dhindsa, K.S. 2020. Android application security: detecting Android malware and evaluating anti-malware software. International Journal of Internet Technology and Secured Transactions, 10(4): 491.

Singh, P., Tiwari, P., Singh, S. 2016. Analysis of malicious behavior of android apps. Procedia Computer Science, 79: 215–220.

Şahin, D.Ö., Kural, O.E., Akleylek, S., Kılıç, E. 2021. Permission-based Android malware analysis by using dimension reduction with PCA and LDA. Journal of Information Security and Applications, *63*(October): 102995.

Utku, A., Doğru, I.A. 2017. Permission based detection system for android malware. Journal of the Faculty of Engineering and Architecture of Gazi University, 32(4): 1015–1024.

Utku, A.D.İ.A. 2016. Mobil kötücül yazılımlar ve güvenlik çözümleri üzerine bir inceleme. Gazi University Journal of Science, 4(2): 49–64.

Ünver, H.M., Bakour, K. 2020. Android malware detection based on image-based features and machine learning techniques. SN Applied Sciences, 2(7): 1–15.

Volkan M. 2021. Dünya çapında ne kadar android cihazın yer aldığı belli oldu! https://www.teknoburada.net/2021/05/19/dunyada-ne-kadar-android-cihaz-var/

Yavaş, M., Güran, A., Uysal, M. 2020. Covid-19 veri kümesinin SMOTE tabanlı örnekleme yöntemi uygulanarak sınıflandırılması. European Journal of Science and Technology, August, 258–264.

Yerima, S.Y., Sezer, S. 2018. Droidfusion: A novel multilevel classifier fusion approach for android malware detection. IEEE Transactions on Cybernetics, 49(2): 453–466.

Yerima, S.Y., Sezer, S. 2019. DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection. IEEE Transactions on Cybernetics, 49(2): 453–466.