ZARARLI ANDROİD YAZILIMLARININ MAKİNE ÖĞRENMESİ İLE AİLELERİNE GÖRE SINIFLANDIRILMASI

ANDROID MALWARE FAMILY CLASSIFICATION WITH MACHINE LEARNING

SERCAN TÜRKER

DOÇ. DR AHMET BURAK CAN

Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS TEZİ

olarak hazırlanmıştır.

2019

SERCAN TÜRKER'in hazırladığı "**Zararlı Android Yazılımlarının Makine Öğrenmesi İle Ailelerine Göre Sınıflandırılması**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Doç. Dr. Süleyman TOSUN

Başkan .....

Doç. Dr. Ahmet Burak CAN

Danışman .....

Dr.Öğr. Üyesi Murat AYDOS

Üye .....

Dr.Öğr. Üyesi Adnan ÖZSOY

Üye .....

Dr.Öğr. Üyesi Mehmet DEMİRCİ

Üye .....

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak ... / ... / ..... tarihinde onaylanmıştır.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU

Fen Bilimleri Enstitüsü Müdürü

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında,

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

14 / 06 / 2019

SERCAN TÜRKER

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma ama iznini Hacettepe Üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alınarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan "**Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge**" kapsamında tezim aşağıda belirtilen koşullar haricinde YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

- Enstitü / Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.

- Enstitü / Fakülte yönetim kurulunun gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren .... ay ertelenmiştir.

- Tezimle ilgili gizlilik kararı verilmiştir

14 / 06 / 2019

SERCAN TÜRKER

ÖZET

ZARARLI ANDROİD YAZILIMLARININ MAKİNE ÖĞRENMESİ İLE AİLELERİNE GÖRE SINIFLANDIRILMASI

Sercan TÜRKER

Yüksek Lisans, Bilgisayar Mühendisliği Bölümü

Tez Danışmanı : Doç. Dr. Ahmet Burak CAN

Haziran 2019, 56 sayfa

Android mobil işletim sisteminin oldukça popüler olması bu sistemin kullanıcılarına zarar vermek amacıyla geliştirilen yazılımların sayısında artışa sebep olmaktadır. Bu nedenle zararlı Android yazılımlarının tespit edilmesi amacıyla birçok çalışma yapılmıştır. Android yazılımlarının zararlı ya da zararsız olarak sınıflandırılması dışında, zararlı olduğu bilinen yazılımların ait oldukları ailelere göre sınıflandırılması da Android işletim sisteminin güvenliği kapsamında oldukça önemlidir. Bu çalışmada zararlı Android yazılımlarını analiz ederek bu yazılımların ait olduğu aileyi tahmin eden makine öğrenmesi tabanlı bir sistem geliştirilmiştir. Geliştirilen bu sistem, zararlı Android yazılımlarının talep ettiği izinleri ve yaptığı API çağrılarını tespit ederek bunları makine öğrenmesi algoritmalarında öznitelik olarak kullanmakta ve farklı sınıflandırma algoritmaları ile zararlı yazılımların sınıflandırılmasına imkan tanımaktadır. Sistemin performansı çeşitli veri kümelerinde yapılan deneylerle incelenmiş ve deney sonuçlarında tüm sınıflandırma algoritmalarının zararlı yazılımları yüksek doğruluk değerleriyle sınıflandırdığı görülmüştür. Bu çalışmaya ek olarak, daha önce karşılaşılmamış bir zararlı yazılım ailesine ait zararlı yazılımın bilinmeyen olarak tespit edilebilmesi için de çalışma yapılmış ve bu yazılımlar yüksek oranda başarıyla tespit edilmiştir.

Anahtar Kelimeler: Makine öğrenmesi, Zararlı Android Yazılımları, Statik Analiz, Zararlı Yazılım Ailelerine Göre Sınıflandırma

ABSTRACT

ANDROID MALWARE FAMILY CLASSIFICATION WITH MACHINE LEARNING

Sercan TÜRKER

Master of Science, Department of Computer Engineering

Supervisor: Doç. Dr. Ahmet Burak CAN

June 2019, 56 Pages

As the popularity of Android mobile operating system grows, the number of software developed to harm the users of this system increases. Therefore, many studies have been done to detect malicious Android software. Apart from the classification of Android software as malicious or benign, classification of the malicious software into their families is also very important in terms of the security of the Android operating system. In this study, a machine learning based classification system is developed that analyzes malicious Android software and estimates the family of them. The developed system detects the requested permissions and API calls of the malicious Android software and uses them as features in machine learning algorithms to classify malwares. The performance of the system is investigated using various data sets and the evaluation results show that all classification algorithms classified the malware with a high accuracy. In addition to this work, a study of detecting an unknown malware which belongs to a family that had never seen before is made and these unknown malwares are classified with a high success rate.

Keywords: Machine Learning, Android Malware, Static Analysis, Android Malware Family Classification

TEŞEKKÜR

Lisansüstü eğitimim boyunca bilgi ve tecrübelerinden yararlandığım, katkılarıyla çalışmama yön veren değerli hocam Sayın Doç. Dr. Ahmet Burak CAN'a,

Hayatım boyunca her koşulda bana destek veren ve sabır gösteren, yaptıklarının karşılığını hiçbir zaman ödeyemeyeceğim aileme,

Sonsuz Teşekkürler...

Sercan TÜRKER
Haziran 2019, Ankara

İÇİNDEKİLER

ŞEKİLLER

ÇİZELGELER

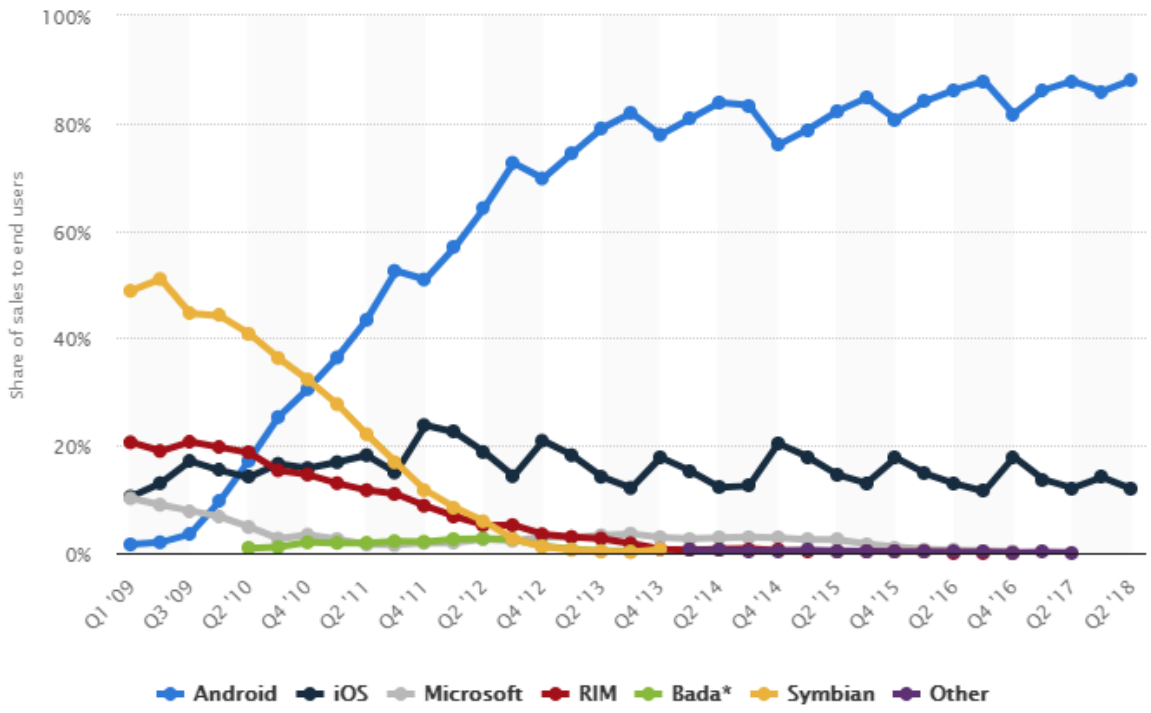
SİMGELER VE KISALTMALAR

Kısaltmalar

API	Application Programming Interface
DT	Decision Tree
KNN	K-Nearest Neighbors
LR	Logistic Regression
MLP	Multi Layer Perceptron
RF	Random Forest
SVM	Support Vector Machine

1. GİRİŞ

Akıllı telefonlar son yıllarda hayatımızın önemli bir parçası haline gelmiştir. Kullanıcılarına İnternet erişimi, görüntülü konuşma, sosyal medya kullanma, mesajlaşma gibi birçok kolaylık sağlayan bu cihazların işletim sistemleri arasında Android işletim sistemi en popüler olanıdır. Yapılan araştırmalara göre 2009 yılından 2018'in ikinci çeyreğine kadar satışı yapılan akıllı telefonların yüzde 88'i Android işletim sistemine sahiptir [1]. Android işletim sisteminin ve diğer mobil işletim sistemlerinin kullanım oranının yıllara göre değişimini gösteren grafik [1] Şekil 1.1.'de gösterilmektedir. Grafikte Android işletim sisteminin diğer sistemlere oranla çok daha fazla tercih edildiği görünmektedir. 2019 yılı Ocak ayı itibariyle resmi Android yazılım mağazası olan Google Play, yaklaşık 2,5 milyon Android yazılımı bulundurmaktadır [2]. Android yazılımlarının Google Play mağazası dışında ApkMirror ve ApkPure gibi alternatif mağazalardan da edinilebildiği düşünüldüğünde toplam Android yazılımı sayısının çok daha fazla olduğu düşünülmektedir.



Şekil 1.1. Mobil işletim sistemlerinin yıllara göre kullanım oranları

Android işletim sisteminin böylesine büyük bir pazar oranına sahip olması ve Android yazılımlarının dağıtımının kolay olması, zararlı yazılım geliştiricilerin Android işletim sistemini IOS ve Windows gibi diğer mobil işletim sistemlerine oranla daha çok hedef almasına sebep olmuştur [3]. Ayrıca Android işletim sisteminin açık kaynak kodlu olması ve Çizelge 1.1.'de görüldüğü üzere Android kullanıcılarının büyük bir kısmının halen Google tarafından destek verilmeyen eski Android sürümlerini kullanıyor olması da zararlı yazılım geliştiricilerin dikkatini çekmektedir [4].

Çizelge 1.1. 2018 yılı Ekim ayı itibariyle Android sürümleri ve kullanım oranları [55]

SÜRÜM	İSİM	YAYIN TARİHİ	API DÜZEYİ	DAĞILIM
9.0	Pie	06.08.2018	28 (Güncel)	< %0.1
8.1	Oreo	05.12.2017	27 (Destek Veriliyor)	%7.5
8.0		21.08.2017	26 (Destek Veriliyor)	%14
7.1	Nougat	04.10.2016	25 (Destek Veriliyor)	%10.1
7.0		22.08.2016	24 (Destek Veriliyor)	%18.1
6.0	Marshmallow	05.10.2015	23 (Destek Verilmiyor)	%21.3
5.1	Lollipop	09.03.2015	22 (Destek Verilmiyor)	%14.4
5.0		03.11.2014	21 (Destek Verilmiyor)	%3.5
4.4	KitKat	31.10.2013	19 (Destek Verilmiyor)	%7.6
4.3	Jelly Bean	24.07.2013	18 (Destek Verilmiyor)	%0.4
4.2		13.11.2012	17 (Destek Verilmiyor)	%1.5
4.1		09.07.2012	16 (Destek Verilmiyor)	%1.1
4.0	Ice Cream Sandwich	19.10.2011	15 (Destek Verilmiyor)	%0.3
2.3	Gingerbread	09.02.2011	10 (Destek Verilmiyor)	%0.2

Android işletim sisteminin saldırganlar tarafından sıklıkla hedef alınması, zararlı Android yazılımı analizinde çalışmalar yapılmasına sebep olmuştur. Android platformu için geliştirilen bir yazılımın zararlı ya da zararsız olduğunu tahmin eden sistemler üzerine birçok çalışma yapılmıştır. Bir yazılımın zararlı ya da zararsız olduğu bilgisinin yanı sıra, zararlı olduğu bilinen bir yazılımın hangi zararlı yazılım ailesine ait olduğunu bilebilmek de önemlidir. Belirli bir zararlı yazılım ailesine ait zararlı yazılımlar benzer davranışlar

sergileyeceğinden, yeni karşılaşılan bir zararlı yazılımın bilinen bir aileye ait olup olmadığı bilgisi zararlı yazılım analistlerinin işlerini kolaylaştıracaktır. Analistler, bilinen zararlı yazılımlarla vakit harcamak yerine hiçbir aileye dahil edilemeyen yazılımlara odaklanabileceklerdir. Ayrıca zararlı yazılımlara karşı koruma sağlayan programlar zararlı bir yazılımla karşılaştıklarında, kullanıcıya yazılımın zararlı olduğu bilgisi dışında yazılımla ilgili detaylı bilgiler sunabilmesi açısından da zararlı yazılımların ailelerinin tespit edilebilmesi oldukça önemli bir araştırma konusudur.

Bu tez çalışması kapsamında, Android platformu için geliştirilmiş zararlı yazılımların ait oldukları zararlı yazılım ailelerini tahmin eden bir sistem geliştirilmiştir. AMD [5] veri kümesindeki zararlı Android yazılımlarının statik olarak analiz edilmesiyle elde edilen izin talepleri ve API çağrılarından en belirleyici olanlar tespit edilmiş ve bunlar çeşitli makine öğrenmesi sınıflandırma algoritmalarında öznitelik olarak kullanılmıştır. Sınıflandırma aşamasında K-En yakın komşuluk algoritması [6] , Karar Ağacı [7] , Lojistik Regresyon [8], Destek Vektör Makineleri [9] ve Çok Katmanlı Algılayıcı [10] gibi algoritmaların yanında Rastgele Orman [11] ve AdaBoost [12] gibi topluluk algoritmaları da kullanılmıştır. Ayrıca Çoğunluk Oylama (Majority Voting) yöntemini kullanan yeni bir topluluk algoritması oluşturulmuştur. Sınıflandırma algoritmalarının en iyi sonucu verebilmesi için hiper-parametreleri belirlenmiş ve deneyler bu parametrelerle yapılmıştır. Deneylerde sınıflandırma algoritmalarının başarısı öncelikle 10 parçalı çapraz doğrulama ile gözlemlenmiştir. Ardından veri kümesi, eğitim kümesi ve test kümesi olacak şekilde ikiye bölünmüş ve sınıflandırma algoritmalarının eğitim kümesinde yer almayan veriler üzerindeki başarısı da incelenmiştir. Yapılan deneyler sonrası alınan sonuçlarda bütün sınıflandırma algoritmalarının zararlı yazılımları yüksek doğruluk yüzdesiyle sınıflandırdığı görülmüştür. Doğruluk yüzdelerinin dışında duyarlılık, hassasiyet ve F1-Skoru değerleri de hesaplanmış ve sonuçlar yorumlanmıştır. Deney sonuçlarının tutarlı olduğunu görebilmek ve sonuçları literatürdeki benzer çalışmalarla karşılaştırabilmek amacıyla sınıflandırma işlemleri Drebin [28] ve UpDroid [65] veri kümeleri üzerinde tekrarlanmış ve alınan sonuçlar raporlanmıştır. Çalışmada sınıflandırmanın yanı sıra zararlı yazılım ailelerinin özniteliklerle ilişkileri incelenmiş ve zararlı yazılımların zararlı yazılım tipine ya da ait olduğu zararlı yazılım ailesine göre sergiledikleri davranışlar raporlanmıştır. Ayrıca, önceden karşılaşılmamış bir

zararlı yazılım ailesine ait zararlı yazılımın bilinmeyen olarak tespit edilebilmesi konusunda çalışma yapılmış ve alınan sonuçlar incelenmiştir.

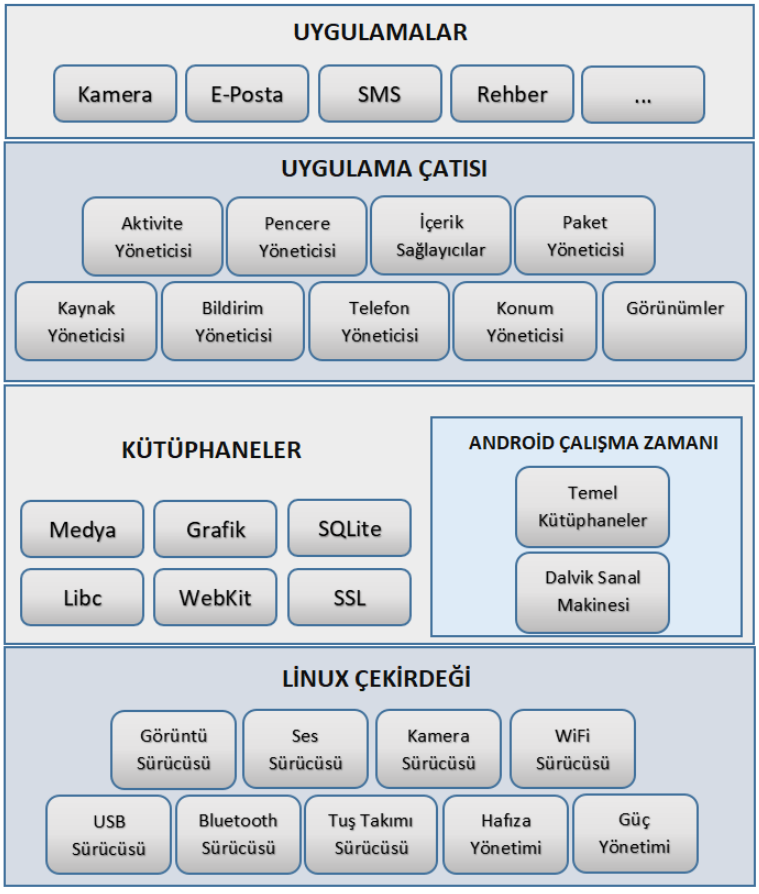
Tez 6 bölümden oluşmaktadır. 2. bölümde Android platformu, Android yazılımları, tersine mühendislik araçları, zararlı mobil yazılımlar, Android yazılımlarının analiz edilmesi ve makine öğrenmesiyle ilgili temel bilgiler verilmiştir. 3. bölümde zararlı Android yazılımlarının tespiti ve ailelerine göre sınıflandırılmasıyla ilgili yapılan çalışmalar anlatılmaktadır. 4. bölümde geliştirilen model detaylı olarak anlatılmıştır. Bir sonraki bölümde deneysel çalışmalara yer verilmiş ve geliştirilen modelin performansı detaylı olarak incelenmiştir. Kullanılan özniteliklerin değerlendirilmesi ve bilinmeyen zararlı yazılımların sınıflandırılmasına ilişkin çalışma da bu bölümde yer almaktadır. Son olarak 6. bölümde yapılan çalışmalar özetlenerek sonuçlar yorumlanmıştır.

2. GENEL BİLGİLER

Bu bölümde çalışmanın detaylarına girilmeden önce Android işletim sisteminin mimarisi, Android yazılımlarının yapısı, zararlı mobil yazılımlar, tersine mühendislik araçları, Android yazılımlarının analizi ve çalışmada kullanılan makine öğrenmesi algoritmalarıyla ilgili genel bilgi verilmiştir.

2.1. Android Platformu

Android, mobil cihazlar için geliştirilen Linux tabanlı, açık kaynak kodlu ve ücretsiz bir işletim sistemidir. Google ve Open Handset Alliance tarafından geliştirilmektedir. Temel olarak dört katmandan oluşan Android işletim sisteminin mimari görünümü Şekil 2.1.'de verilmiştir.



Şekil 2.1. Android Mimari Yapısı [56]

Katmanların en altında Linux çekirdeği katmanı bulunmaktadır. Bu katmanda dokunmatik ekran, kamera, hoparlör, bluetooth gibi donanım bileşenleri ile üst katmanlar arasındaki haberleşmeyi sağlayan cihaz sürücüleri bulunmaktadır. Aynı zamanda bellek, süreç ve güç yönetimi gibi hizmetleri de sunmaktadır.

İkinci katmanda kütüphaneler ve Android çalışma zamanı bileşenleri bulunmaktadır. Kütüphaneler yazılımların kullandığı C ve C++ dilleri ile yazılmış harici kütüphaneleri ifade etmektedir. İnternet tarayıcısının çalışabilmesi için gerekli olan Webkit, grafik işlemleri için OpenGL, veri tabanı işlemleri için SQLite, ses ve video işlemleri için Media Framework gibi yapılar bu bileşenin içinde bulunmaktadır. Bu katmanın diğer bir bileşeni olan Android çalışma zamanı bileşeni de Android yazılımlarının üzerinde çalışacağı Dalvik sanal makinesini ve Java kütüphanelerini bulundurur. Dalvik sanal makinesi Android platformu için özelleştirilmiş Java tabanlı bir sanal makinedir ve Android yazılımlarının birbirlerinden soyutlanabilmesi için her Android yazılımı için farklı bir sanal makine oluşturulur.

Bir üst katman, yazılımın kullanabileceği üst düzey servislerin sunulduğu Uygulama Çatısı katmanıdır. Servislere örnek olarak yazılımın yaşam döngüsünün yönetildiği Aktivite Yöneticisi, diğer yazılımlarla veri paylaşmasına olanak sağlayan İçerik Sağlayıcılar, kullanıcı ayarlı dış kaynaklara erişimi sağlayan Kaynak Yöneticisi, kullanıcı arayüzlerinin yönetilmesini sağlayan Görüntü Sistemi verilebilir.

En üst katmanda ise Android yazılımları (uygulamaları) bulunmaktadır. Android yazılımlarının ayrıntılı yapısı bir sonraki bölümde anlatılmıştır.

2.2. Android Yazılımları

Android yazılımları *apk* uzantılı dosyalardır. *Apk* kısaltması, uygulama paketi anlamına gelen *application package* söz öbeğinden elde edilmiştir. Bu dosyalar, yazılımın Java programlama dili ile yazılan kaynak kodlarını, manifesto dosyasını, kullandığı kaynakları ve üçüncü parti kütüphaneleri içeren sıkıştırılmış dosyalardır. Android yazılımlarının bileşenleri ve nasıl geliştirildiği alt başlıklarda anlatılmaktadır.

2.2.1. Android Yazılımlarının Bileşenleri

Android yazılımları aktiviteler, servisler, yayın alıcılar ve içerik sağlayıcılar olmak üzere 4 ana bileşenden oluşmaktadır. Bunlar dışında fragmanlar, görünümler, layoutlar, intentler, kaynaklar ve manifestolar gibi ek bileşenler de bulunmaktadır.

Aktiviteler (Activities): Yazılımların kullanıcı ile etkileşiminin sağlandığı bileşendir. Bir yazılım tek bir aktiviteden oluşabileceği gibi birden çok aktivite bulundurabilir. Ancak aynı anda sadece tek bir aktivite aktif olmaktadır. *Activity* sınıfından türetilirler.

Servisler (Services): Yazılımlarda arka planda yapılması gereken işlemler bu bileşen aracılığıyla gerçekleştirilir. Kullanıcı arayüzü içermeyen bu bileşenler *Service* sınıfından türetilerek oluşturulur.

Yayın Alıcılar (Broadcast Receivers): Sistemden ya da diğer yazılımlardan gelen mesajları dinleyen ve bu mesajlara yanıt veren bileşenlerdir. Örneğin bir SMS'in gelmesini veya bir arka plan işleminin bitmesini yazılımlar, bu bileşenleri kullanarak anlarlar. *BroadcastReceiver* sınıfından türetilirler.

İçerik Sağlayıcılar (Content Providers): Bir yazılımdan diğerlerine istek üzerine veri sağlanması için kullanılan bileşenlerdir. Örneğin *Whatsapp* yazılımının kişi listesine erişimi içerik sağlayıcılar ile sağlanmaktadır. *ContentProvider* sınıfından türetilirler.

Fragmanlar (Fragments): Bir aktivitedeki kullanıcı arayüzünün bir bölümünü temsil eden fragmanlar, aktiviteleri modüler şekilde kullanabilmeye yardımcı olan bileşenlerdir. *Fragment* sınıfından türetilirler.

Görünümler (Views): Metin kutusu, düğme, liste gibi kullanıcı arayüz bileşenlerini temsil eden bileşendir.

Düzenler (Layouts): Ekran formatının ve arayüz bileşenlerinin görünümünün kontrol edildiği bileşendir.

Niyetler (Intents): Bileşenler arası iletişim ve veri alış verişi bu bileşen ile sağlanmaktadır.

Kaynaklar (Resources): Yazılımın kullanacağı görsel, tema, animasyon gibi harici elemanların oluşturduğu bileşendir.

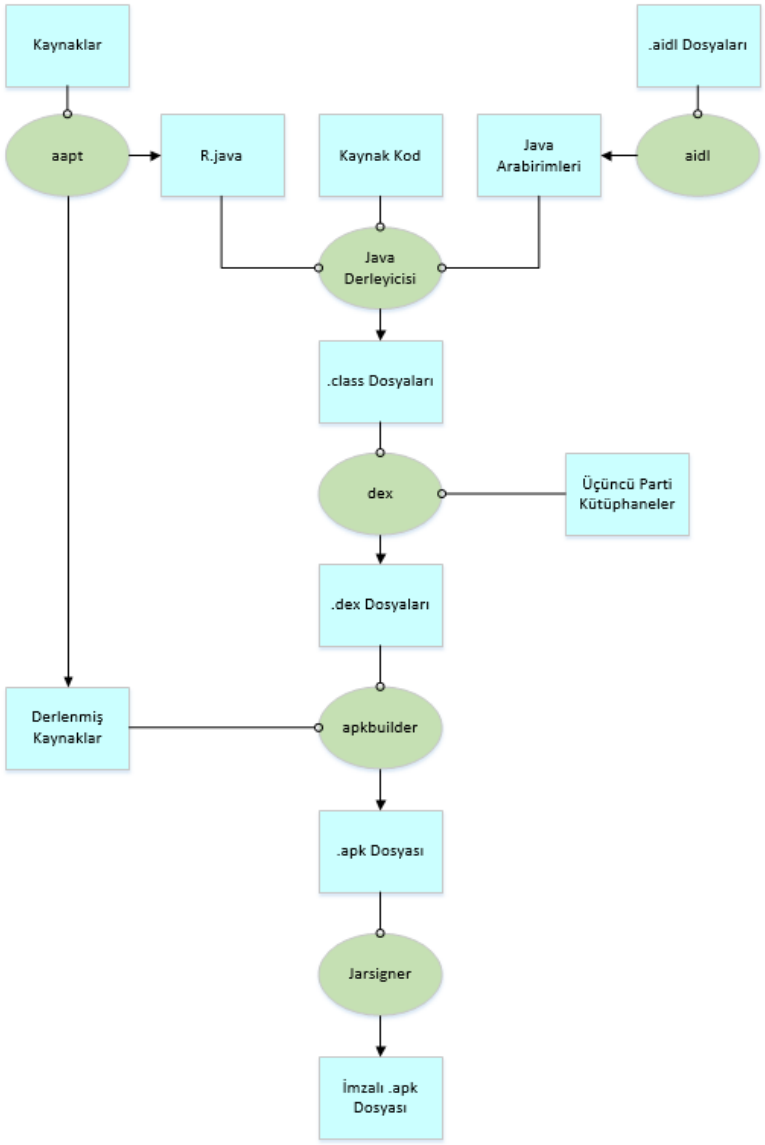
Manifestolar (Manifests): Yazılımın konfigürasyon dosyasıdır. Her Android yazılımının yazılım içerisinde kullanılan aktivitelere, servislere, yayın alıcılarına ve intent filtrelerine ait bilgi edinebileceği bir manifesto dosyasına ihtiyacı vardır. Ayrıca zararlı Android yazılımı analizinde sıklıkla kullanılan izin talepleri de bu dosyadan elde edilmektedir. Android yazılımları belirli işlemleri yapabilmek için kullanıcının iznini talep ederler. Bu izinler Internet'e erişme izni, ağ durumunu öğrenme izni gibi normal izin talepleri olabileceği gibi rehbere erişme izni, mesaj gönderme izni gibi tehlikeli izin talepleri de olabilir. Bu sebeple herhangi bir Android yazılımının kötü niyetli olup olmadığı, manifesto dosyasındaki bazı izin taleplerinden anlaşılabilir.

2.2.2. Android Yazılımlarının Geliştirilmesi

Android yazılımları Java, C/C++, Kotlin, C#, Python gibi dillerle geliştirilebilmektedir. Fakat bunlar arasında en çok tercih edilen ve Google tarafından resmi olarak desteklenen programlama dili Java'dır. Java ile geliştirilen bir Android yazılımında yazılımın Java kaynak kodları ve AIDL (Android Interface Definition Language) tarafından uygun formata çevrilen Java arabirimleri, Java derleyicisi tarafından *class* uzantılı sınıf dosyalarına dönüştürülür. Android SDK'sında bulunan *dx* aracı, Java sınıf dosyalarını ve üçüncü parti kütüphaneleri dex (Dalvik Executable) dosyalarına dönüştürür. Bu dosyayla birlikte *aapt* aracı tarafından bir araya getirilen diğer yazılım dosyaları *apkbuilder* aracı ile sıkıştırılarak apk dosyası oluşturulur ve son olarak oluşturulan apk dosyasının imzalanması işlemi *jarsigner* aracı ile gerçekleştirilir. Bu adımlar Şekil 2.2.'de gösterilmektedir.

2.3. Zararlı Mobil Yazılımlar

Mobil cihazlara müdahale etmek, kullanıcı bilgilerini ele geçirmek ve cihazları uzaktan kontrol edebilmek gibi amaçlarla geliştirilen zararlı mobil yazılımlar 8 alt grupta incelenebilir [5] [13].



Şekil 2.2. Android Yazılımının Oluşturulma Süreci [57]

Reklam Yazılımı (Adware): Belirli firmalar tarafından sağlanan reklamları programın içine gömerek kullanıcıya sunan yazılımlardır. Genellikle ücretsiz olarak dağıtılan bu yazılımları geliştiren saldırganlar, gösterilen reklamlardan ve bu reklamların kullanıcı tarafından tıklanmasından fayda sağlar.

Casus Yazılım (Spyware): Kullanıcının izni olmadan rehber, mesajlar, konum gibi kişisel verileri toplayarak üçüncü bir kişiye göndermek üzere geliştirilen yazılımlardır. Dışarıya

bilgi aktarmasının dışında, cihazdaki aktiviteleri sürekli dinlediği için cihazın performansını da olumsuz anlamda etkilerler.

Arka Kapı (Backdoor): Yüklendikleri sistemde süper kullanıcı izni almaya çalışarak cihaz üzerinde sınırsız yetki elde etmeyi amaçlayan yazılımlardır. Cihaz ele geçirildiğinde uzaktan kontrol edilebilmekte ve istenilen eylemler kullanıcıya fark ettirilmeden gerçekleştirilebilmektedir. Rootkitler de bu sınıfa dahil edilebilir.

Botnet: Saldırganların zararlı eylemleri birçok farklı cihaz üzerinden gerçekleştirebilmesi için geliştirdiği zararlı yazılımlardır. Botnet saldırılarında ele geçirilen cihazlar uzak bir sunucu tarafından kontrol edilir ve sunucudan gelen komutlarla saldırganın istediği eylem gerçekleştirilir. Botnet sözcüğü *robot* ve *network* sözcüklerinden türetilmiştir.

Solucanlar: Kendi kendilerine çoğalarak farklı cihazlara ağ üzerinden yayılmayı amaçlayan yazılımlardır. Virüslerin alt kümesi olarak kabul edilir.

Truva Atı (Trojan): Zararsızmış gibi görünen ancak zararlı eylemler gerçekleştiren yazılımlardır. Virüs ve solucanların aksine kendi kendine yayılamayan bu yazılımlar genellikle kullanıcının bilgisi olmadan gizli bir şekilde yüklenir. Bankacılık sistemlerini hedef alarak kullanıcıların hesap bilgilerini çalmak (Trojan-Banker), Internet tarayıcısına komutlar göndererek ve bazı sistem dosyalarını değiştirerek belirli web sitelerine bağlanılması için girişimlerde bulunmak (Trojan-Clicker), kötü amaçlı yazılımların algılanmasını engellemek (Trojan-Dropper) ve kullanıcının haberi olmadan belirli numaralara SMS göndermek gibi eylemler gerçekleştirirler.

Fidye Yazılımı (Ransomware): Saldırganın mobil cihazdaki verileri yalnızca kendisinin çözebileceği şekilde şifreleyerek ya da cihazın tamamen kilitlenmesini sağlayarak kullanıcıdan cihazın tekrar sorunsuz şekilde çalışabilmesi için fidye talep ettiği yazılımlardır.

Korsanlık Aracı (HackerTool): Korsanlar tarafından cihazlara saldırı yapabilmek için kullanılan şifre kırıcı, port tarayıcı gibi yazılımlardır. Yazılımlar herhangi bir zararlı eylemde bulunmasa da, korsanlar tarafından kullanıldığında tehlikeli olabilir.

Zararlı mobil yazılımlar genel olarak 3 farklı yöntemle üretilmekte ve cihazlara yüklenmektedir [14]. Yeniden paketleme (Repackaging) yönteminde yaygın olarak kullanılan yazılımlara zararlı eylemde bulunacak kod parçacıkları eklenerek aynı paket adıyla sunulmaktadır. Diğer bir yöntem olan güncelleme (Update Attack) yöntemiyle zararlı yazılımlar zararlı kod parçalarını baştan yüklemek yerine sonradan yaptıkları güncellemelerle yükler. Son yöntem olan indirme (Drive-by Download) yönteminde de, kullanıcının zararlı yazılımı sahte bir yazılım indirme sayfası aracılığıyla indirmesi sağlanmaktadır.

2.4. Tersine Mühendislik

Tersine mühendislik, bir sistemin yapısının, içeriğinin veya çalışma biçiminin birtakım analizler yapılarak keşfedilmesidir. Zararlı bir yazılıma karşı önlem alabilmek için saldırıyı tersine mühendislik yöntemlerini kullanarak incelemek gerekebilir. Bu nedenle tersine mühendislik yöntemi zararlı yazılım analizlerinde sıklıkla kullanılmaktadır. Android yazılımlarının bileşenleri de tersine mühendislik yöntemleriyle görünür hale getirilebilir.

Android yazılımlarının kaynak kodlarının okunabilir formatta görünebilmesi için kaynak koda dönüştürme (decompiling) yapan araçlar kullanılabilir. Yazılımları apk formatından jar formatına dönüştüren dex2jar [15] gibi araçlar apk içerisindeki Dalvik kodlarını Java sınıf dosyalarına çevirir. Java sınıf dosyalarından da Procyon [16], jd-cmd [17], jadx [18] ve CFR [19] gibi araçlarla kaynak kodlar elde edilebilir. Bazı araçlar da kaynak kodları çıkarmak yerine dex kodlarından insanlar tarafından okunabilir olan smali [20] kodlarını üretir. Bu işleme tersine çevirme (disassamble) ya da *baksmali* denir. Bu işlem için kullanılan en popüler araç Apktool [21] aracıdır. Bir Android yazılımı Apktool aracı ile tersine mühendislik işlemine sokulduğunda *smali* kodları, kaynak dosyalar, varlıklar (assets) ve manifesto dosyası elde edilir.

2.5. Android Yazılımlarının Analizi

Android işletim sisteminin güvenliği kapsamında zararlı Android yazılımlarının tespit edilebilmesi ve doğru sınıflandırılabilmesi için Android yazılımları analiz edilerek incelenmektedir. Analizler statik ve dinamik olmak üzere iki farklı şekilde yapılabilir.

Statik analizde yazılımlar herhangi bir cihazda çalıştırılmadan, yalnızca apk dosyası incelenerek analiz edilir. Yazılımların kaynak koduna ve diğer bileşenlerine tersine mühendislik araçları yardımıyla ulaşılarak yazılımların zararlı eylemlerde bulunan kod parçaları içerip içermediği analiz edilir. Bu yöntemin avantajı herhangi bir cihaza zarar verilmeden zararlı yazılımın analiz edilebilmesidir. Ancak saldırganlar bu yazılımları geliştirirken birtakım kod karıştırma ve şifreleme teknikleri uygulayabilmektedir. Bu durumda tersine mühendislik yöntemiyle elde edilen kaynak kodu incelemek anlamsız olacaktır.

Dinamik analizde ise Android yazılımları sanal ya da gerçek bir cihaz üzerinde çalıştırılarak sistem çağrıları, ağ hareketleri, kaynak kullanımı gibi bazı davranışları incelenir ve yazılımım zararlı bir eylem gerçekleştirip gerçekleştirmediğine karar verilir. Bu yöntemle kaynak koda uygulanacak kod karıştırma ve şifreleme işlemlerinden etkilenilmeyeceği için statik analize göre daha sağlıklı sonuçlar üretilebilir. Fakat statik analize göre daha uzun sürebilmekte ve zararlı yazılımların üzerinde çalıştığı cihaza vereceği zararlar düşünüldüğünde daha uğraştırıcı olabilmektedir. Ayrıca saldırganlar dinamik analiz yöntemlerinden kaçınmak için de bazı yöntemler geliştirmişlerdir. Örneğin Android'in sunduğu bazı metotlar yardımıyla sanal bir cihazda çalıştığını anlayan zararlı yazılım, zararlı eylemini gizleyerek normal bir yazılımmış izlenimi verebilmektedir. Böyle durumlarda gerçek bir cihazın analiz için kullanılması düşünülebilir fakat bu da gerçek cihazların zarar görmesi istenmeyeceği için iyi bir çözüm olarak değerlendirilmeyebilir.

Her iki yöntemin dezavantajlarından kaçıp avantajlarını kullanmak amacıyla hibrit analiz gerçekleştirilebilir. Hibrit analiz ile yazılımlar hem statik hem de dinamik olarak analiz edilir ve çıkan sonuçlar yorumlanır.

2.6. Makine Öğrenmesi Algoritmaları

Makine öğrenmesi, bilgisayarların geçmiş bilgilerden matematiksel ve istatistiksel işlemler ile çıkarımlar yaparak gelecekteki olaylar hakkında tahmin yürütmesine ve modelleme yapmasına imkan veren bir yapay zeka alanıdır. Makine öğrenmesi, öğrenme yöntemine göre gözetimli öğrenme (supervised learning), gözetimsiz öğrenme (unsupervised learning) ve takviyeli öğrenme (reinforcement learning) olmak üzere üç grupta incelenmektedir.

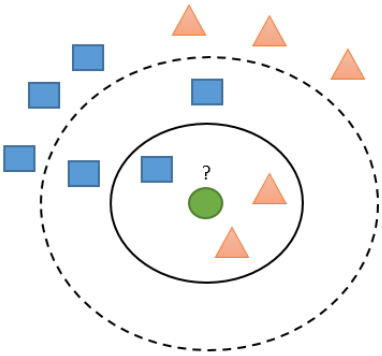
Gözetimli öğrenmede işaretlenmiş verilerle çalışılır ve her giriş değerinin bir çıkış değeri bulunmaktadır. Sınıflandırma ve regresyon algoritmaları bu öğrenme tekniğini temsil etmektedir. Veri setindeki çıkışlar kategorik ise sınıflandırma, nümerik ise regresyon algoritmaları kullanılır. Gözetimsiz öğrenmede veriler işaretlenmemiştir ve işaretlenmemiş veri üzerinden bilinmeyen bir yapı tahmin edilir. Kümeleme algoritmaları gözetimsiz öğrenme yaparlar. Takviyeli öğrenme ise amaca yönelik ne yapılması gerektiğinin öğrenildiği bir makine öğrenmesi yaklaşımıdır.

Makine öğrenmesi ile yapılan sınıflandırma işlemi, geçmiş verilerin hangi sınıflara ait olduğu bilindiğinde yeni gelen verinin hangi sınıfa dahil olduğunun bulunması olarak tanımlanabilir. Sınıflandırma algoritmaları, eğitim verilerini kullanarak öğrenme işlemini gerçekleştirir ve bir model oluşturur. Bu model, sınıfı belli olmayan test verilerinin sınıflarının tahmin edilmesinde kullanılır.

Bu bölümde tez kapsamında kullanılan K-En Yakın Komşu, Karar Ağaçları, Lojistik Regresyon, Destek Vektör Makineleri, Rastgele Orman, Adaboost ve Çok Katmanlı Algılayıcı algoritmalarıyla ilgili genel bilgi verilmektedir.

2.6.1. K-En Yakın Komşu Algoritması

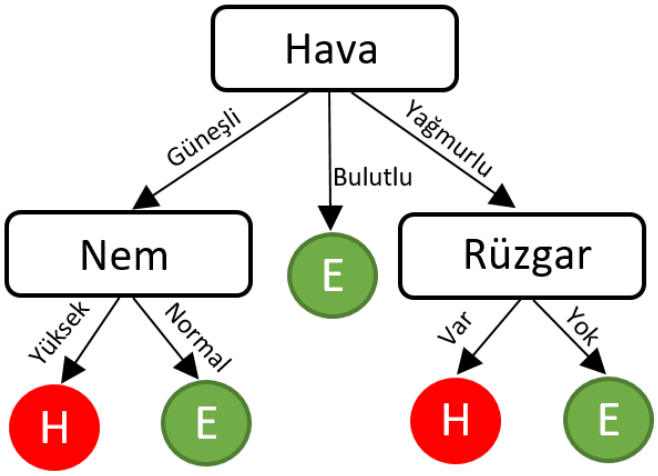
K-En Yakın Komşu algoritması, öznitelik uzayındaki en yakın eğitim örneklerinden faydalanılarak nesnelerin sınıflandırıldığı denetimli öğrenme algoritmasıdır. Sınıflandırma, verilen K değeri adedince yakın komşunun sınıflarına göre yapılır. Test edilen örneğin sınıfının belirlenmesi için eğitim kümesinde o örneğe en yakın K adet örnek seçilir ve hangi sınıfın örneği bu örnek kümesinde daha çok bulunuyorsa, bu sınıf test edilen örneğin sınıfı olarak belirlenir. Örnekler arasındaki mesafe Öklid mesafesi ya da Manhattan mesafesi ile hesaplanır. Şekil 2.3.'te gösterilen örnekte K=3 olması durumunda test edilen örnek üçgen sınıfına, K=5 olması durumunda da kare sınıfına atanır.



Şekil 2.3. K-En Yakın Komşu Algoritması [58]

2.6.2. Karar Ağaçları

Sınıfları bilinen örnek verilerin basit karar verme adımları ile küçük gruplara bölündüğü, her bölme işlemi ile birbirine benzeyen verilerin gruplandığı ve tümevarım yöntemi ile sınıflandırmanın yapıldığı algoritmadır. Şekil 2.4.'te bir hava verisi üzerinde sınıflandırma için oluşturulmuş karar ağacının görünümü verilmiştir.



Şekil 2.4. Örnek Karar Ağacı Görünümü [59]

Karar ağaçlarında karar düğümlerinin oluşturulması için Shannon tarafından ortaya çıkarılan *Bilgi Kazancı* teorisi [54] kullanılmaktadır. Karar ağacının dengeli bir şekilde dallanması ve sınıflandırmanın doğru yapılabilmesi için en iyi öznitelikler düğüm olarak seçilmelidir. Bunun için her özniteliğin bilgi kazancı hesaplanır ve en iyi bilgi kazancı değerini veren

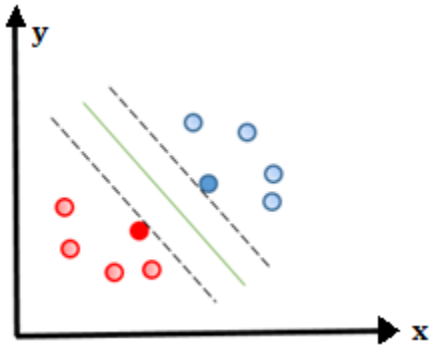
öznitelik karar düğümü olarak atanır. Bu karar düğümüne göre alt gruplar üretilir ve bu işleme tüm gruplar sınıflandırılıncaya dek devam edilir.

2.6.3. Lojistik Regresyon

Bağımlı değişkenin veya çıktı değişkeninin kategorik bir değişken olduğu durumlarda kullanılan lojistik regresyon yönteminde sınıflandırma işlemi, bağımsız değişkenlerle kurulan neden sonuç ilişkisine göre yapılmaktadır. Bağımsız değişkenlerin bağımlı değişken üzerindeki etkilerine göre bağımlı değişkenin alabileceği değerlerin olasılıkları hesaplanmaktadır. Bir durumun gerçekleşme ya da gerçekleşmeme olasılığının hesaplanmasında lojistik fonksiyonu (sigmoid fonksiyonu) kullanıldığından yönteme lojistik regresyon adı verilmiştir.

2.6.4. Destek Vektör Makineleri

Temelleri istatistiksel öğrenme teorisine dayanan bu algoritma, farklı sınıflara ait verileri birbirinden en uygun şekilde ayıracak bir hiper düzlem belirleyerek sınıflandırma işlemi yapar. Veri kümesinin doğrusal olarak ayrılabildiği durumlarda veri kümesindeki örnekler karar doğrusu adı verilen doğruyla birbirlerinden ayrılır. Karar doğrusunun yeni katılacak olan veriye karşı dayanıklı olabilmesi için Şekil 2.5.'te görüldüğü gibi sınıfların sınır çizgilerine en yakın uzaklıkta konumlandırılması gerekmektedir.

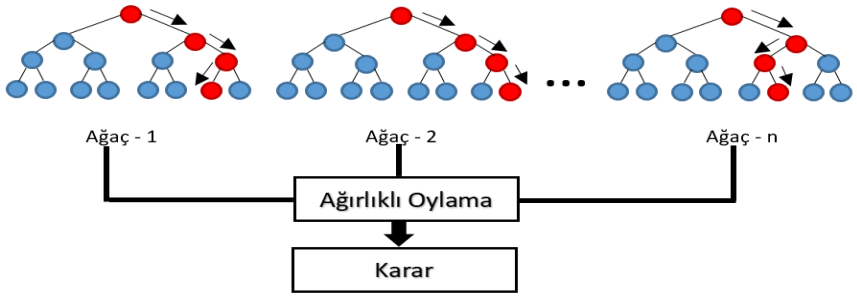


Şekil 2.5. Destek Vektör Makineleri Algoritması [60]

2.6.5. Rastgele Orman

Topluluk öğrenme algoritmalarından olan rastgele orman algoritması, birden fazla karar ağacının kullanılmasına dayalı bir sınıflandırma algoritmasıdır. Bu yöntemde veri

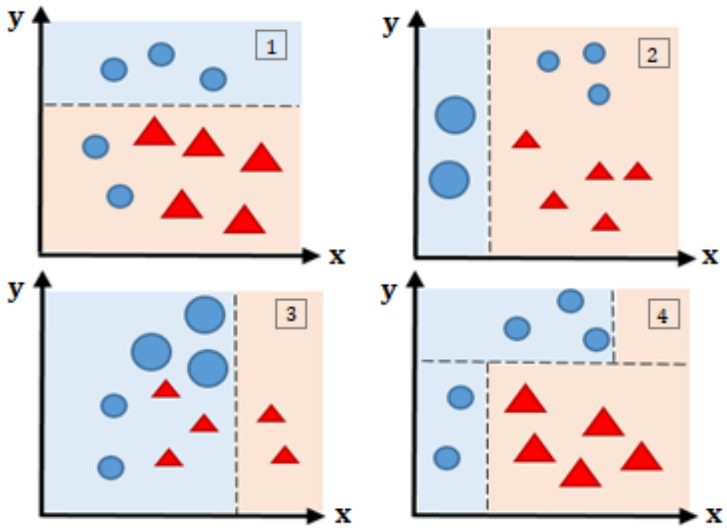
kümesinden rastgele seçilen örneklerle N adet karar ağacı oluşturulur ve her bir karar ağacından elde edilen tahminlerin ortalaması ya da çoğunluğu göz önüne alınarak daha doğru bir tahmin üretmek amaçlanır (Şekil 2.6.). Karar ağaçlarının sayısı ve ağaçların maksimum derinlikleri kullanıcı tarafından belirlenirken hangi özniteliklerin karar düğümlerini oluşturacağı rastgele seçilmektedir.



Şekil 2.6. Rastgele Orman Görünümü [61]

2.6.6. Adaboost

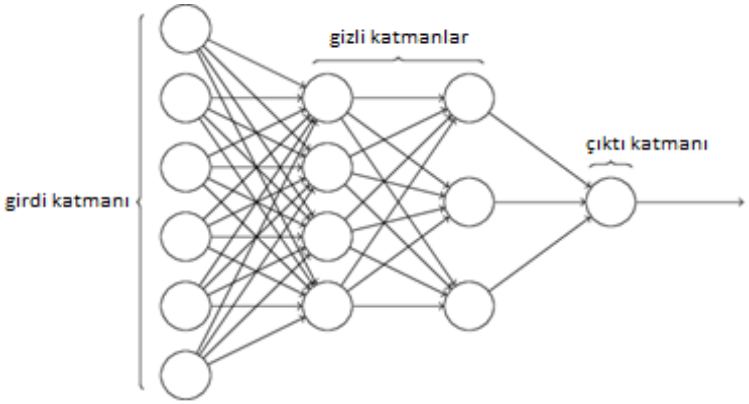
Adaboost, zayıf sınıflandırıcıların bir araya getirilmesiyle oluşturulan güçlü bir topluluk sınıflandırıcıdır. Bu yöntemde sınıflandırma iterasyonlar halinde yapılır ve her bir iterasyonda zayıf sınıflandırıcılar ile sınıflandırma işlemi gerçekleştirilir. Diğer güçlendirme (boosting) yöntemlerinde algoritmaların uygulandığı veri kümesi, önceki iterasyonlarda yanlış sınıflandırılmış verilere öncelik verilecek şekilde rastgele seçimlerle oluşturulurken, AdaBoost algoritmasında her bir iterasyonda eğitim kümesinin tamamı kullanılmaktadır. İterasyonlarda yanlış sınıflandırılan örneklere diğer iterasyonlarda daha yüksek ağırlık verilerek algoritmanın o örneklere odaklanması sağlanır (Şekil 2.7.). İterasyonlar belirlenen parametre sayısı kadar tekrarlanır ve sınıflandırma sonucu her bir iterasyonda kullanılan sınıflandırıcıların ağırlıklı oylamasına göre belirlenir. Sınıflandırıcıların ağırlığı, örnekleri ne kadar düzgün sınıflandırdığıyla ilgilidir. İyi sınıflandırma yapan sınıflandırıcının ağırlığı fazla olacağından son kararı vermede de etkisi fazla olacaktır.



Şekil 2.7. Adaboost Algoritması [62]

2.6.7. Çok Katmanlı Algılayıcı

Yapay sinir ağları, insan sinir sistemini taklit ederek öğrenmeyi amaçlayan makine öğrenmesi yöntemlerinden biridir. Nöronların birbirlerine bağlanması ve haberleşmesiyle oluşan yapay sinir ağı modellerinin en çok kullanılan yöntemlerinden birisi Çok Katmanlı Algılayıcı ağlarıdır. Çok katmanlı algılayıcılar Şekil 2.8.'de görüldüğü gibi girdi katmanı, çıktı katmanı ve gizli katmanlar olmak üzere üç farklı katmandan oluşurlar. Verilerin okunduğu girdi katmanında bilgi işlenmez ve öznitelik sayısı kadar nöron tanımlanır. Çıktı katmanı, verilerin sınıf bilgisinin çıkış olarak hesaplandığı katmandır ve sınıf sayısı kadar nörondan oluşur. Gizli katmanlardan gelen bilgiler bu katmanda işlenerek girdi katmanından sunulan girdi için ağın ürettiği çıktı bulunur. Gizli katmanlar ise girdi katmanından gelen bilgilerin işlendiği katmanlardır ve katman sayısı problemden probleme değişebilmektedir. Karmaşık problemlerde katman sayısı ve katmanlardaki nöron sayısı fazla olmaktadır. Bu katmanlarda ileri yönde hesaplamalar ve geri yönde hata yayılımı yapılmaktadır. İleri yönde hesaplama yapılırken girdilerin çıktılar üzerindeki etkisini ayarlayabilmek için ağırlık değerleri kullanılır. Başlangıçta rastgele şekilde belirlenen bu değerler geri yönde yapılan hata yayılımlarıyla güncellenmekte ve optimal seviyeye getirilmektedir. Modelin örnek görünümü şekildeki gibidir.



Şekil 2.8. Çok Katmanlı Algılayıcı [63]

3. LİTERATÜR ÖZETİ

Zararlı Android yazılımlarının analizi kapsamında zararlı Android yazılımlarının tespit edilmesi ve zararlı olduğu bilinen Android yazılımlarının ait oldukları ailelerin tahmin edilmesi üzerine literatürde çok sayıda çalışma bulunmaktadır. Bu çalışmalarda Android yazılımları statik ve/veya dinamik olarak analiz edilerek sahip oldukları özellikleri öğrenilmiş ve bu özellikleri kullanılarak sınıflandırma işlemi gerçekleştirilmiştir. Çalışmaların birçoğunda Drebin [28] ve Genome [29] veri kümelerinden yararlanılmıştır. Bu çalışmalar yaptıkları sınıflandırma çeşidine göre alt başlıklarda incelenmektedir.

3.1. Zararlı Android Yazılımının Tespiti Konusunda Yapılan Çalışmalar

Statik analizle elde ettiği öznitelikleri kullanarak Android yazılımlarının zararlı olup olmadığını tespit eden çalışmalara örnek olarak Drebin [35], DroidAPIMiner [36] ve Anastasia [37] verilebilir. Arp ve arkadaşları tarafından geliştirilen Drebin, Android yazılımlarından statik analizle elde ettikleri izin taleplerini, Android sistem çağrılarını, bağlantı kurulan ağ adreslerini ve kullanılan donanım bileşenlerini öznitelik olarak kullanmış ve Destek Vektör Makineleri algoritmasıyla %94 oranında sınıflandırma başarısı elde etmiştir. DroidAPIMiner statik analiz aracı, Aafer ve arkadaşları tarafından yapılan çalışma ile geliştirilmiştir. Çalışmada Androguard [22] isimli tersine mühendislik aracıyla kaynak kod elde edilmiş ve kaynak kod içerisinden alınan sistem çağrıları ve Dalvik komutları öznitelik olarak kullanılmıştır. Sınıflandırma aşamasında K-En Yakın Komşuluk algoritması kullanılmış ve %99'luk bir başarı oranı yakalanmıştır. Anastasia isimli çalışmada Fereidooni ve arkadaşları Python ile geliştirdikleri uniPDroid adlı programla öznitelikleri çıkarmış ve çeşitli makine öğrenmesi algoritmalarını kullanarak modellerini test etmişlerdir. Farklı veri kümelerinden elde edilen Android yazılımlarının kullanıldığı sınıflandırma aşamasında en iyi sonucu %97 ile XGBoost algoritmasının verdiği görülmüştür.

Bazı çalışmalarda da zararlı Android yazılımlarının tespit edilebilmesi için dinamik analizden faydalanılmış ve Android yazılımları sanal makinelere yüklenip çalıştırılarak davranışları incelenmiştir. Amos ve arkadaşları [38] yaptıkları çalışmada Android yazılımlarını sanal makinede 5 dakika boyunca Android Monkey [23] aracının ürettiği

girdilerle çalıştırarak davranışlarını gözlemleyen STREAM aracını geliştirmişlerdir. Yazılımın ağda oluşturduğu hareketlerle, pil ve bellek kullanımı gibi özellikler sınıflandırmada kullanılacak öznitelik vektörünün tanımlanmasında kullanılmıştır. Çeşitli makine öğrenmesi modellerinden en iyi sonucu Rastgele Orman algoritması vermiş ve %94.5'lik bir başarı oranı elde edilmiştir. Madam [39] adlı çalışmada Saracino ve arkadaşları sistem çağrıları, bellek ve işlemci kullanımı gibi çekirdek seviyesindeki işlemlerin gözlenmesi dışında SMS gönderimi, Bluetooth kullanımı, Wi-Fi kullanımı gibi kullanıcı seviyesindeki işlemleri de gözlemleyen bir sistem geliştirmişlerdir. Sınıflandırma aşamasında K-En Yakın Komşuluk algoritması kullanılmış ve yapılan deneysel çalışmalarda yazılımların %96'sının doğru sınıflandırıldığı görülmüştür. DroidDolphin [40] adlı çalışmada Wu ve arkadaşları 64000 adet Android yazılımını DroidBox [26] aracını kullanarak çalıştırmış ve yazılımların çağırdıkları API metotlarını Destek Vektör Makineleri algoritmasında öznitelik olarak kullanmışlardır. %86.1 oranında doğru sınıflandırma yapılan çalışmada veri kümesindeki örnek sayısı arttıkça doğruluk oranının da iyileştiği vurgulanmıştır. Bir diğer çalışmada Nancy ve arkadaşları [41] ağ trafiğini gözlemleyerek öznitelik vektörü oluşturmış ve Karar Ağacı yöntemiyle sınıflandırma yapmıştır. Çalışmada %90.32 oranında başarı yakalanmıştır.

Hem statik hem dinamik analizden yararlanarak zararlı yazılım tespiti yapan çalışmalara örnek olarak Lindorfer ve arkadaşlarının Marvin [42] çalışması gösterilebilir. Bu çalışmada statik analizle elde edilen özniteliklerin yanında Andrubis [43] adlı araç kullanılarak dinamik analiz gerçekleştirilmiş ve bu analizle elde edilen öznitelikler de sınıflandırmaya dahil edilmiştir. Sınıflandırmada Lineer Regresyon ve Destek Vektör Makineleri algoritmaları kullanılmış ve başarı oranı olarak %98.24 değerine ulaşılmıştır. F. Yang ve arkadaşları [44] da hibrit analiz uygulayarak öznitelikleri belirledikten sonra çeşitli makine öğrenmesi algoritmaları kullanarak modellerini test etmiş ve Rastgele Orman algoritmasıyla %95.9'luk bir başarı sağlamışlardır.

Son yılların popüler konularından derin öğrenme de zararlı Android yazılımlarının tespiti için yapılan çalışmalarda kullanılmıştır. Xu ve arkadaşlarının geliştirdiği HADM [45], Yuan ve arkadaşlarının geliştirdiği DroidSec [46] ve Su ve arkadaşlarının geliştirdiği DroidDeep

[47] araçları derin öğrenme ile sınıflandırma yaparak sırasıyla %94.7, %96 ve %99.4 oranlarında başarı elde etmişlerdir. DroidDeep çalışmasında statik analiz yapılırken diğer çalışmalarda hibrit analiz yapılmıştır.

3.2. Zararlı Android Yazılımının Ailesinin Tespiti Konusunda Yapılan Çalışmalar

Her ne kadar zararlı Android yazılımı analizi alanında yapılan çalışmaların çoğunluğu Android yazılımlarının zararlı olup olmadığını tespit etmeye yönelik olsa da, zararlı olduğu bilinen Android yazılımlarının hangi zararlı yazılım ailesine ait olduğunu tespit edecek sistemleri geliştirmeye yönelik çalışmalar da bulunmaktadır.

Deshotels ve arkadaşlarının geliştirdikleri DroidLegacy [48] sistemi bu çalışmalara örnek olarak gösterilebilir. Bu çalışmada aynı zararlı yazılım ailesine ait Android yazılımlarının birbirlerine olan benzerlikleri API çağrılarından elde edilen bağımlılık çizgelerine göre hesaplanmış ve bir aile içerisindeki en çok ortak olan özellik o ailenin imzası olarak belirlenmiştir. Tüm ailelerin imzası bu şekilde belirlendikten sonra yazılımların bu ailelere olan benzerlikleri hesaplanarak sınıflandırma yapılmıştır. 11 zararlı yazılım ailesine ait 1052 örnek veri içeren veri kümesi kullanılarak yapılan çalışmada %98 oranında başarı yüzdesi elde edilmiştir.

Garcia ve arkadaşları, geliştirdikleri RevealDroid [49] adlı sistemde hem zararlı yazılım tespiti hem de aile sınıflandırması yapmışlardır. Statik analizle çıkardıkları karıştırmaya karşı dayanıklı öznitelikleri Destek Vektör Makineleri ve Karar Ağacı algoritmalarında kullanarak zararlı yazılımların ailelerini %95 oranında doğru tahmin etmişlerdir.

Dash ve arkadaşları [50] DroidScribe ismini verdikleri çalışmada Android yazılımlarını dinamik olarak analiz etmiş ve elde ettikleri öznitelikleri Destek Vektör Makineleri algoritmasında kullanarak %84'lük bir başarı oranı yakalamışlardır. Yalnızca dinamik analiz yapılan başka bir çalışmada Massarelli ve arkadaşları [51] kaynak kullanımlarından elde ettikleri öznitelikleri Destek Vektör Makineleri algoritmasında kullanarak sınıflandırma yapmış ve %82 oranında başarı elde etmişlerdir.

Tangil ve arkadaşları [52] DroidSieve ismini verdikleri sistemi geliştirmişlerdir. Statik analizle elde ettikleri özniteliklerin karıştırma tekniklerine karşı duyarlı olmasına özen gösterilen çalışmada Aşırı Rassal Ağaç (Extremely Randomized Tree) algoritmasıyla çalışılmış ve %99.26 oranında başarı yakalanmıştır.

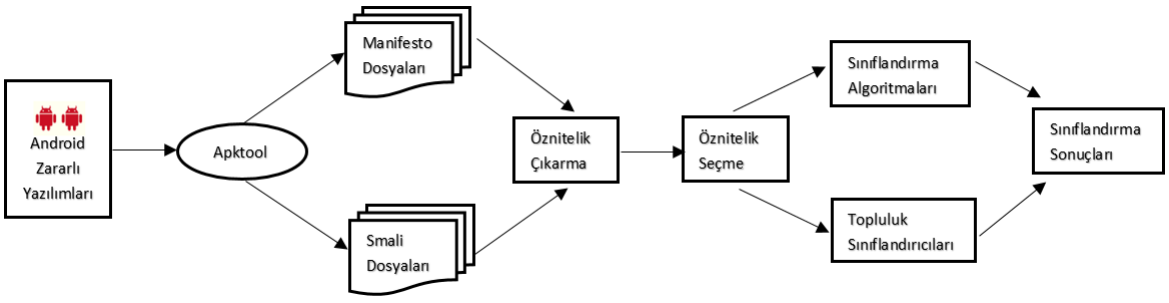
Chakraborty ve arkadaşları [53] yaptıkları çalışmada hem statik hem dinamik öznitelikler kullanmış ve çeşitli makine öğrenmesi algoritmaları kullanarak sınıflandırma yapmışlardır. Deney sonuçlarında yüksek doğruluk yüzdelerinin elde edildiği çalışmada az örnek barındıran yazılım ailelerinin de doğru sınıflandırılabildiği vurgulanmıştır.

Aktaş ve arkadaşları da, UpDroid ismini verdikleri çalışmada [64] hibrit analiz yaparak statik ve dinamik öznitelikler elde etmişlerdir. Android yazılımlarının aktivite sayısı, servis sayısı, talep edilen izin sayısı ve dosya boyutu gibi özellikleri statik analizle elde edilmiş, dinamik öznitelikler içinse DroidBox aracı kullanılmıştır. Kendi veri kümeleri olan UpDroid veri kümesinde ve Drebin veri kümesinde ayrı ayrı yapılan sınıflandırma çalışmasında KNN algoritması ile sırasıyla 96.37 ve 96.85 oranında başarı elde edilmiştir.

4. MODEL VE YÖNTEM

Bu bölümde zararlı Android yazılımlarının ailelerine göre sınıflandırılması için yapılan çalışmalar detaylı olarak ele alınmaktadır. Sınıflandırma için öncelikle veri kümesindeki zararlı yazılımlar statik olarak analiz edilmiş ve öznitelikler çıkarılmıştır. Çıkarılan öznitelikler arasından en belirleyici olanları belirlenmiş ve çeşitli makine öğrenmesi algoritmalarına girdi olarak verilmiştir. Çalışmada Destek Vektör Makineleri, Karar Ağaçları, K-En Yakın Komşu, Lojistik Regresyon, Rastgele Orman, AdaBoost ve Çok Katmanlı Algılayıcı algoritmaları kullanılmıştır. Bunlar dışında Karar Ağaçları, K-En Yakın Komşu ve Lojistik Regresyon algoritmalarının bir araya getirilmesiyle yeni bir topluluk algoritması oluşturulmuştur. Sınıflandırma algoritmalarının tasarlanması, eğitilmesi ve sınıflandırmanın yapılabilmesi için Python programlama diliyle geliştirilmiş Scikit-Learn [32] adlı makine öğrenmesi kütüphanesi kullanılmıştır.

Sınıflandırma algoritmalarının kullandığı öznitelikler, zararlı yazılımların *Apktool* aracı ile statik olarak analiz edilmesiyle elde edilen izin talepleri ve API çağrılarından oluşmaktadır. Bu özniteliklerin en belirleyici olanlarının seçilmesinin ardından hiper-parametreleri belirlenen sınıflandırma algoritmaları ile sınıflandırma işlemi gerçekleştirilmekte ve sınıflandırma algoritmalarının performansları çeşitli performans metrikleriyle raporlanmaktadır. Sistemin genel görünümü Şekil 4.1.'de gösterilmektedir.



Şekil 4.1. Sistemin Genel Görünümü

Bölüm içerisinde çalışmada kullanılan veri kümeleri, özniteliklerin çıkarılması, en etkili özniteliklerin seçilmesi ve makine öğrenmesi algoritmalarının tasarlanması anlatılmaktadır.

4.1. Veri Kümeleri

Bu tez kapsamında sınıflandırma algoritmalarının başarısını analiz edebilmek için AMD [5], Drebin [28] ve UpDroid [65] veri kümeleri kullanılmıştır. AMD veri kümesi, özniteliklerin çıkarılması, seçilmesi ve sınıflandırma algoritmalarının hiper-parametrelerinin belirlenmesinde kullanılan ana veri kümesiyken diğer veri kümeleri, AMD veri kümesinde yapılan sınıflandırma sonuçlarının tutarlı olduğunu göstermek ve sonuçları literatürdeki benzer çalışmalarla karşılaştırabilmek amacıyla kullanılmıştır.

İçerisinde 71 zararlı yazılım ailesine ait 24553 zararlı yazılım bulunduran AMD veri kümesi, 2010 ve 2016 yılları arasında geliştirilen reklam yazılımları, arka kapı yazılımları, fidye yazılımları, korsanlık araçları ve farklı tipte truva atlarından oluşmaktadır. Veri kümesinin Web sayfasında, veri kümesinde bulunan zararlı yazılım ailelerinin karakteristik özellikleri ile ilgili detaylı bilgiler yer almaktadır. Veri kümesindeki 86 zararlı yazılım, analiz sırasında oluşan hatalar dolayısıyla çalışmaya dahil edilmemiştir.

2010 ve 2012 yılları arasında geliştirilen zararlı Android yazılımlarının toplanmasıyla oluşturulan Drebin veri kümesinde ise 179 zararlı yazılım ailesi ve 5560 zararlı yazılım bulunmaktadır.

2015 yılından sonra geliştirilmiş zararlı Android yazılımlarını bulunduran UpDroid veri kümesinde ise 20 zararlı yazılım ailesine ait 2408 zararlı yazılım bulunmaktadır.

Veri kümelerindeki apk dosyalarının isimleri anlamlı olmadığı için çalışmada kolaylık sağlamak amacıyla dosya isimlerinin başına numara getirilmiş ve tüm dosyalar 1'den başlanarak numaralandırılmıştır.

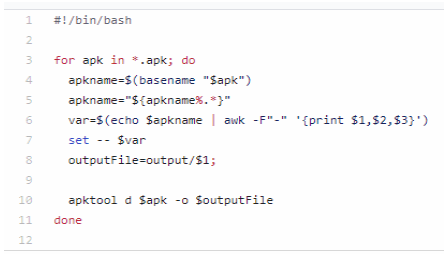
4.2. Özniteliklerin Çıkarılması

Android yazılımlarının AndroidManifest.xml dosyasında belirttikleri izin talepleri, Android platformunda önemli bir güvenlik mekanizması olarak kullanılmaktadır. Yazılımların rehber, sms, galeri gibi kişisel verilere erişimleri ve kamera, bluetooth, mikrofon gibi sistem bileşenlerini kullanabilmeleri, kullanıcıdan izin almalarıyla mümkün olmaktadır ve bu durum, izin taleplerinin zararlı Android yazılımı analizinde en çok kullanılan özniteliklerden

biri olmasına sebep olmuştur [30]. İzin talepleri, Android yazılımlarının zararlı olup olmadığını belirlemede yardımcı olabileceği gibi yazılımların davranışları hakkında bilgi verdiği için zararlı bir yazılımın hangi zararlı yazılım ailesine ait olduğunu belirlemede de yardımcı olabilir. Örnek olarak AMD veri kümesindeki Svpeng isimli zararlı yazılım ailesi verilebilir. Mobil cihazları kilitlemeye çalışan Svpeng ailesi yazılımları, bu eylemi gerçekleştirebilmek için *SYSTEM_ALERT_WINDOW* iznini talep ederler [31]. Bu durum göz önüne alındığında, bu iznin öznitelik olarak kullanılması, bu izni talep eden zararlı bir Android yazılımının Svpeng ailesine ait olup olmadığını belirlemede işe yarar bir bilgi olacaktır. Bu sebeple izin talepleri çalışmada öznitelik olarak kullanılmıştır.

Çalışmada izin talepleri dışında API çağrıları öznitelik olarak kullanılmıştır. Android API'si yazılımların Android işletim sistemini kullanabilmeleri için birtakım metotlar tanımlamıştır. Yazılımların kullandıkları metotlar ve bu metotları hangi sıklıkla kullandıkları, yazılımların davranışlarını belirlemede önemlidir. Örneğin AMD veri kümesindeki BankBot ve Triada ailelerine ait yazılımlar dinamik analizden kaçınmak için yazılımın gerçek bir cihazda açılıp açılmadığını kontrol ederek, sanal makinede açıldığını tespit etmeleri halinde zararlı davranışlarını gizlemektedir. Cihazın gerçek olup olmadığı kontrolü *TelephonyManager* sınıfında bulunan *GetDeviceId* metodundan dönen sonuç ile anlaşılabilmektedir. Bu sebeple herhangi bir yazılımın bu metodu kullanıyor olması, o yazılımın BankBot ya da Triada ailelerine ait olup olmadığı hakkında ipucu verebilir. Bu nedenle API çağrıları da sınıflandırma algoritmaları için öznitelik olarak kullanılmıştır.

Öznitelikleri elde edebilmek için bir tersine mühendislik aracı olan *Apktool* kullanılmıştır. Ubuntu işletim sisteminde yazılan ve zararlı yazılımların her birini *Apktool* aracı ile statik olarak analiz edilen Betik programı (bash script) Şekil 4.2.'de gösterilmektedir.



```
#!/bin/bash

for apk in *.apk; do
  apkname=$(basename "$apk")
  apkname="${apkname%.*}"
  var=$(echo $apkname | awk -F"-" '{print $1,$2,$3}')
  set -- $var
  outputFile=output/$1;

  apktool d $apk -o $outputFile
done
```

Şekil 4.2. Apktool ile Android yazılımlarının analiz edilmesini sağlayan betik programı

Programda klasör içindeki apk dosyaları dolaşılmakta ve apk dosyalarının başında bulunan numara *outputFile* değişkenine atıldıktan sonra *apktool* komutu çalıştırılarak sonuç çıktısı oluşturulmaktadır. Program AMD veri kümesindeki dosyalar için koşturulduğunda 1'den 24467'ye kadar numaralandırılmış çıktı klasörleri elde edilmiştir. Çıktı klasörlerinde zararlı yazılımların kaynak kodlarını içeren smali dosyalarının bulunduğu *smali* klasörü, yazılımın kullandığı kaynak dosyalarının bulunduğu *res* klasörü, yazılımın kullandığı diğer varlıkları bulunduran *assets* klasörü, apk dosyasıyla ilgili meta bilgilerin bulunduğu *original* klasörü ve AndroidManifest.xml dosyası bulunmaktadır. Şekil 4.3.'te rastgele seçilen bir çıktı klasörünün görünümü verilmiştir.

Şekil 4.3. Analiz sonrası her bir Android yazılımı için oluşan çıktı klasörleri

Apktool aracının her bir Android yazılımı için ürettiği bu çıktı klasöründe izin talepleri için AndroidManifest.xml dosyası, API çağrıları için de smali dosyaları analiz edilmiştir. AndroidManifest.xml dosyasındaki izin taleplerinin ve smali dosyasındaki API çağrılarının örnek görünümleri Şekil 4.4.'te verilmiştir.

```
2227     move-result-object v1
2228
2229     sget-object v2, Lcom/android/advsdk/e/b;->a:Ljava/lang/String;
2230
2231     invoke-virtual {v0}, Landroid/telephony/TelephonyManager;->getDeviceId()Ljava/lang/String;
2232
2233     move-result-object v3
2234
2235     new-instance v2, Ljava/lang/StringBuilder;
```

```
<?xml version="1.0" encoding="utf-8" standalone="no"?><manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.GET_ACCOUNTS"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.WAKE_LOCK"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
    <uses-permission android:name="android.permission.READ_SMS"/>
    <uses-permission android:name="android.permission.WRITE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.MODIFY_AUDIO_SETTINGS"/>
```

Şekil 4.4. İzin talebi ve API Çağrısı

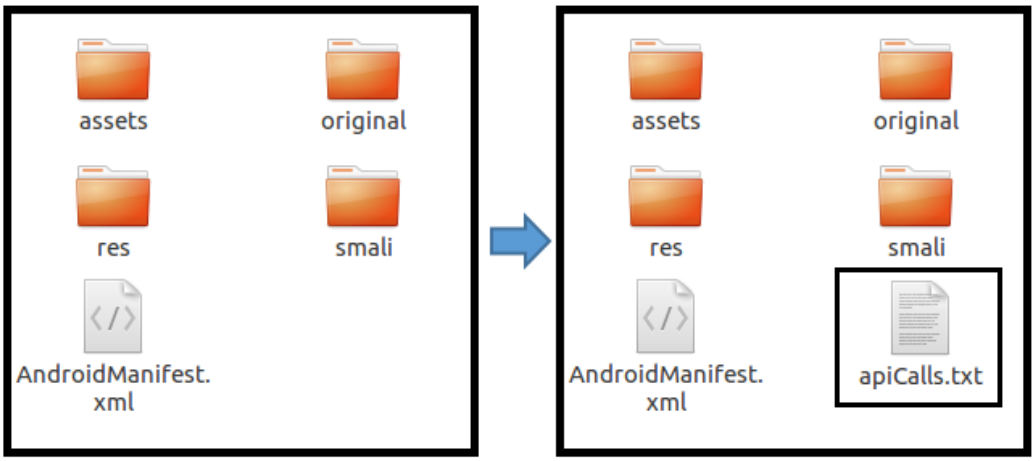
Android resmi sayfasından [27] elde edilen izin türleriyle birlikte kaldırılan izin türleri de düşünülerek toplam 278 izin öznitelik olarak belirlenmiştir. API çağrıları da AMD veri kümesindeki bütün yazılımların smali kodlarının taranmasıyla elde edilmiştir. Smali kodları Ubuntu işletim sisteminde yazılan ve Şekil-4.5.'te görülen bir betik programıyla taranarak, *android*, *java* ve *javax* paketleri altındaki sınıfların metotlarına yapılan çağrılar tespit edilmektedir.

```
1   #!/bin/bash
2
3   for D in *; do
4           cd "$D"
5
6           grep -r 'Landroid.*;->.*(' . | grep -o 'Landroid.*(' | sed 's/.$//' > temp.txt
7           grep -r 'Ljava.*;->.*(' . | grep -o 'Ljava.*(' | sed 's/.$//' >> temp.txt
8           grep -r 'Ljavax.*;->.*(' . | grep -o 'Ljava.*(' | sed 's/.$//' >> temp.txt
9           awk '!x[$0]++' temp.txt > apiCalls.txt
10          cd ..
11  done
```

Şekil 4.5. Smali kodlarını tarayarak API çağrılarını bulan betik programı

Programda çıktı klasörünlerinde 'Landroid', 'Ljava' ve 'Ljavax' deseniyle başlayan ve ';->' desenini içeren tüm satırlar belirlenmekte ve geçici bir dosyaya yazılmaktadır. Ardından bu

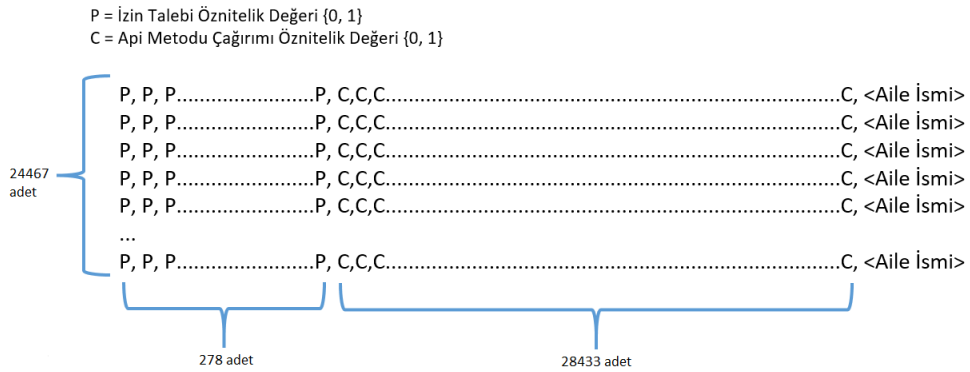
geçici dosya, içindeki tekrar eden satırlar kaldırılarak *apiCalls.txt* isminde kaydedilmektedir. Betik programın çalışması sonucu her bir çıktı klasöründe oluşan görünüm Şekil 4.6.'da verilmektedir.



Şekil 4.6. Her bir Android yazılımı için API metodu çağrılarının kaydedilmesi

AMD veri kümesindeki tüm zararlı Android yazılımlarının API çağrıları belirlendikten sonra 24677 adet *apiCalls.txt* dosyasında bulunan metot çağrıları tekrar edenler yalnızca bir kez alınacak şekilde birleştirilmiş ve bu şekilde toplam 28433 farklı API metodu öznitelik olarak belirlenmiştir.

İzin talebi özniteliklerinin değerleri, ilgili izin talep ediliyorsa; API çağrısı özniteliklerinin değerleri de ilgili API metodu çağrılıyorsa 1, aksi takdirde 0 olarak atanmıştır. Bu işlemi yapan program C# programlama dili ile Windows ortamında geliştirilmiştir. EK 1'de verilen programda Apktool aracının ürettiği çıktı klasörleri altında bulunan AndroidManifest.xml ve apiCalls.txt dosyaları okunur. Manifesto dosyasında *<uses-permission* deseni ile arama yapılarak, öznitelik olarak belirlenen izin taleplerinin dosyada bulunup bulunmadığı kontrol edilir. İzin talebi dosyada bulunuyorsa ilgili öznitelik için çıktı dosyasına 1, bulunmuyorsa 0 yazılır. Ardından API metodu öznitelikleri sırasıyla dolaşılarak API metodunun ilgili zararlı yazılıma ait apiCalls.txt dosyasında bulunup bulunmadığı kontrol edilir. Eğer bulunuyorsa çıktı dosyasına 1, bulunmuyorsa 0 yazılır. Son olarak zararlı yazılımın bağlı olduğu aile adı da dosyaya yazılır ve öznitelik dosyası Şekil 4.7.'de verilen formatta oluşturulur.



Şekil 4.7. Öznitelik dosyasının formatı

Veri kümesinde bulunan zararlı yazılım ailelerinin bazıları tersine mühendislik araçlarının işini zorlaştırmak amacıyla kaynak koddaki sınıfların, metotların ve değişkenlerin isimlerini rastgele isimlerle değiştirmektedir. Yeniden adlandırma (renaming) adı verilen bu yöntem uygulandığında Apktool aracının ürettiği smali kodlarında anlamsız isimler yer almakta ve yazılımın analiz edilmesi zorlaşmaktadır. Ancak smali kodunda bulunan API çağrıları, yeniden adlandırma yöntemi uygulansa bile bozulmamakta ve okunabilir durumda olmaktadır [31]. Bu nedenle çalışmada kullanılan özniteliklerin yeniden adlandırma yöntemine karşı dayanıklı olduğu söylenebilir.

Bazı zararlı yazılım ailelerine ait yazılımlar da statik analizden kaçınmak için zararlı eylemi gerçekleştirecek yazılımı sonradan yüklemektedir. Dinamik yükleme [31] adı verilen bu yöntemde zararlı yazılımı yüklemenin yollarından birisi zararlı yazılımı *Assets* klasöründe bulundurup buradan yüklemektir. Bu yönteme karşı dayanıklılık sağlamak için de Apktool aracının ürettiği çıktı klasöründeki *Assets* klasörünün herhangi bir apk dosyası bulundurup bulundurmadığı kontrol edilmiş ve apk dosyası bulunması durumunda bu dosyanın da Apktool aracılığıyla analiz edilmesi sağlanmıştır. Herhangi bir özniteliğin değeri ilk analiz edilen yazılım için 0 olsa bile *Assets* klasörü altındaki yazılım o özniteliğe sahip olduğunda öznitelik değeri 1 olarak değiştirilmektedir. Böylece sonradan yüklenecek zararlı yazılım göz ardı edilmemiş olduğundan, bu yöntem karşı dayanıklılık sağlanmıştır. Ancak dinamik yükleme işlemi farklı şekillerle de yapılabildiğinden, bu kaçınma yöntemine karşı tümüyle dayanıklılık sağlandığı söylenemez.

4.3. Özniteliklerin Seçilmesi

Makine öğrenmesi yöntemlerinde sınıflandırma algoritmaları kadar, veri kümesinden elde edilen özniteliklerin kalitesi de sınıflandırma başarısına etki eden önemli bir unsurdur. Sınıflandırma performansını olumsuz yönde etkileyen gereksiz özniteliklerin elenmesi, sınıflandırma algoritmalarının daha doğru sonuçlar üretmesini sağlayacaktır. Bununla birlikte öznitelik sayısı azaldığı için algoritmaların sonuç üretme hızı artacaktır. Bu durum göz önüne alınarak çalışma kapsamında belirlenen 28711 özniteliğin sınıflandırma başarısına olan etkileri, Rastgele Orman algoritmasının ürettiği öznitelik önem değerlerinden yararlanılarak incelenmiştir. Scikit-Learn kütüphanesindeki RandomForestClassifier algoritması Şekil 4.8.'de görüldüğü gibi varsayılan parametreleriyle oluşturulmuş ve AMD veri kümesindeki eğitilme işleminin ardından *feature_importances_* niteliğiyle özniteliklerin önem dereceleri belirlenmiştir.

```
forest=RandomForestClassifier()
forest.fit(X, y)
importances = forest.feature_importances_
```

Şekil 4.8. En etkili özniteliklerin belirlenmesi

Deneylerde önem değerlerine göre sıralanan özniteliklerden kaç tanesinin kullanılacağını belirleyebilmek için Destek Vektör Makineleri, Rastgele Orman ve Lojistik Regresyon algoritmalarıyla AMD veri kümesi üzerinde ayrı ayrı sınıflandırma işlemi yapılmıştır. Bu algoritmalar için Scikit-Learn kütüphanesinden yararlanılmış ve sınıflandırmada en etkili özniteliklerin sırasıyla 250, 500, 750, 1000, 1500, 2500 ve 5000 adedi kullanılmıştır. Her öznitelik sayısı için algoritmaların ürettikleri doğruluk değerleri ve bu değerlerin aritmetik ortalaması Çizelge 4.1.'de verilmektedir.

Çizelgedeki değerler incelendiğinde öznitelik sayısının her algoritmayı farklı şekilde etkilediği görülmektedir. En iyi sınıflandırma performansını Destek Vektör Makineleri algoritması 1000, Rastgele Orman algoritması 500, Lojistik Regresyon algoritması ise 5000 öznitelik kullanıldığında göstermiştir. Ancak ortalama başarı değerleri incelendiğinde 1000 öznitelik kullanılarak yapılan sınıflandırmalar sonucu elde edilen ortalama başarı oranının diğerlerine göre fazla olduğu görülmektedir. Bu nedenle deneysel çalışmalarda en etkili 1000

öznitelik kullanılarak sınıflandırma yapılacaktır. Bu özniteliklerin 42'si izin talebi, geri kalanı API çağrısı özniteliğidir.

Çizelge 4.1. Farklı öznitelik sayılarında algoritmaların sınıflandırma başarısı

ÖZNİTELİK SAYISI	DESTEK VEKTÖR MAKİNELERİ(%)	RASTGELE ORMAN (%)	LOJİSTİK REGRESYON (%)	ORTALAMA BAŞARI (%)
250	97.87	97.81	97.81	97.83
500	98.05	**97.97**	98.13	98.05
750	98.15	97.92	98.15	98.07
1000	**98.43**	97.82	98.29	**98.18**
1500	97.95	97.78	98.21	97.98
2500	96.88	97.93	98.25	97.69
5000	95.42	97.11	**98.33**	96.95

4.4. Sınıflandırma Algoritmalarının Tasarlanması

Sınıflandırma işlemi için kullanılacak makine öğrenmesi algoritmaları Scikit-Learn kütüphanesinden elde edilmiştir. K-En Yakın Komşu, Karar Ağaçları, Lojistik Regresyon, Destek Vektör Makineleri, Rastgele Orman, AdaBoost ve Çok Katmanlı Algılayıcı algoritmaları için kütüphanede bulunan KneighborsClassifier, DecisionTreeClassifier, LogisticRegression, SVC, RandomForestClassifier, AdaBoostClassifier ve MLPClassifier sınıfları kullanılmıştır. Bunlar dışında Scikit-Learn kütüphanesinin sağladığı metotlar yardımıyla K-En Yakın Komşu, Karar Ağaçları ve Lojistik Regresyon algoritmaları birleştirilerek MajorityVoteClassifier adında bir topluluk sınıflandırma algoritması oluşturulmuştur. Sınıfı oluşturan Python kodu EK 2'de verilmiştir.

Makine öğrenmesi modelleri tasarlanırken, model mimarisine tasarımcının belirleyeceği birtakım parametrelerle karar verilmektedir. Bu parametrelere hiper-parametre adı verilir ve hiper-parametreler makine öğrenmesi algoritmalarının başarısı üzerinde önemli bir etkiye sahiptir. Çoğu zaman, belirli bir model için en uygun mimarinin ne olması gerektiği, bir başka deyişle hiper-parametrelerin alacağı değerler başlangıçta bilinmemektedir. Modelin en iyi sonucu vermesi için bu parametrelerin hangi kombinasyonda olması gerektiği belirlenmelidir. Grid Search algoritması [33] en iyi hiper-parametre değerlerinin seçilebilmesi için kullanılan algoritmalardan biridir. Bu algoritmada modeller hiper-parametrelerin alabileceği tüm değerlerle test edilir ve en iyi sonucu veren kombinasyon belirlenir. Tez kapsamında Scikit-Learn paketinde bulunan GridSearchCV metodu

yardımıyla AMD veri kümesi kullanılarak hiper-parametreler belirlenmiştir. Metodun genel kullanımı Şekil 4.9.'da gösterilmektedir.

```
grid_search = GridSearchCV(estimator=model, param_grid=grid, cv=5)
grid_result = grid_search.fit(X, y)
print grid_result.best_params_
```

Şekil 4.9. Hiper-parametrelerin belirlenmesi

GridSearchCV metodu, kullanılacak sınıflandırma algoritmasını *estimator* parametresiyle, algoritmanın parametrelerinin alabileceği değerleri *param_grid* parametresiyle ve kaç katlı çapraz doğrulama yapılacağı bilgisini de *cv* parametresiyle girdi olarak almaktadır. Sınıflandırma algoritması, *param_grid* parametresiyle verilen parametre tablosundaki tüm değerlerle *cv* parametresiyle verilen sayı kadar eğitilir ve en iyi sınıflandırma başarısını sağlayan parametre kombinasyonu *best_params_* niteliğiyle belirlenir. Tez kapsamında kullanılan sınıflandırma algoritmaları için metoda verilen parametre tabloları ve metodun çıktı olarak verdiği hiper-parametreler Çizelge 4.2.'de gösterilmektedir. Topluluk sınıflandırma algoritmasının hiper-parametreleri, içerdiği algoritmaların hiper-parametreleriyle aynı olduğundan çizelgede gösterilmemiştir.

Çizelge 4.2. Parametre tablosu ve Hiper-parametreler

SINIFLANDIRMA ALGORİTMASI	PARAMETRE TABLOSU	HİPER-PARAMETRELER
KneighborsClassifier	n_neighbors : [3,4,5,6,7,8] p : [1,2,3]	n_neighbors=5 p=2
DecisionTreeClassifier	criterion : ['gini', 'entropy'] max_depth : [2,4,6,8,10,12,14,16,18,20,None]	criterion='entropy', max_depth=18
LogisticRegression	C: [0.01, 0.1, 1, 10, 100] penalty:['l1','l2']	C=0.1 penalty='l2'
SVC	kernel: ['linear', 'poly', 'rbf'] gamma: [0.001,0.01,0.1] C: [1, 10, 100]	kernel='poly' gamma=0.1 C=1
RandomForestClassifier	criterion : ['gini','entropy'] n_estimators : [100,1000,10000]	criterion='entropy' n_estimators=10000
AdaBoostClassifier	n_estimators : [100,250,500,750,1000] learning_rate : [0.1,0.5,1]	n_estimators=500 learning_rate=1
MLPClassifier	max_iter : [50, 100, 150, 200] batch_size : [100, 200, 300] activation: ['tanh', 'relu', 'logistic']	max_iter=100, batch_size=200, activation='relu'

5. DENEY ÇALIŞMALARI

Bu bölümde, tez çalışmasında geliştirilen sınıflandırma sisteminin performansının değerlendirilebilmesi için deneysel çalışmalar yapılmış ve alınan sonuçlar raporlanmıştır. AMD veri kümesinde yapılan çalışmada elde edilen sonuçların tutarlı sonuçlar olduğunu gösterebilmek ve sonuçları benzer çalışmalarla karşılaştırabilmek için Drebin ve UpDroid veri kümelerinde de sınıflandırma işlemi gerçekleştirilmiştir. Bunların yanı sıra, zararlı yazılımların öznitelikleri incelenmiş ve zararlı yazılım kategorilerinin özniteliklerle olan ilişkileri raporlanmıştır. Son olarak önceden karşılaşılmamış bir zararlı yazılımın bilinmeyen bir yazılım olduğunun belirlenebilmesi için çalışma yapılmıştır.

Deneylerde K-En Yakın Komşu, Karar Ağaçları, Lojistik Regresyon, Destek Vektör Makineleri, Rastgele Orman ve Çok Katmanlı Algılayıcı algoritmaları, İngilizce kısaltmaları olan KNN, DT, LR, SVM, RF ve MLP ile ifade edilmiştir. Topluluk algoritması ise sınıflandırma kararı ağırlıklı oylama (majority voting) ile verildiği için MV olarak ifade edilmiştir. Algoritmalar sınıflandırma işlemi için Bölüm 4.3.'te belirlenen öznitelikleri ve Bölüm 4.4.'te belirlenen hiper-parametreleri kullanmışlardır.

Bu bölümde öncelikle deneylerde kullanılan metriklere yer verilmiştir. Ardından AMD veri kümesinde sınıflandırma performansı analiz edilmiş ve AMD veri kümesindeki zararlı yazılım kategorilerinin özniteliklerle ilişkisi raporlanmıştır. Bir sonraki bölümde Drebin ve UpDroid veri kümelerinde yapılan sınıflandırma işleminde alınan sonuçlar analiz edilmiş ve benzer çalışmalarla karşılaştırılmıştır. Son bölümde de bilinmeyen zararlı yazılımların sınıflandırılması çalışmasına yer verilmiştir.

5.1. Performans Metrikleri

Sınıflandırma algoritmalarının sınıflandırma performanslarının ölçülebilmesi ve diğer algoritmaların performanslarıyla karşılaştırılabilmesi için başarım ölçüm metriklerine ihtiyaç duyulmaktadır. Geliştirilen modelin yeterince iyi olup olmadığına karar vermek için bu metriklerin hesaplanması ve yorumlanması gerekmektedir. Metriklerin hesaplanabilmesi için sınıflandırma algoritmalarının ürettiği karmaşıklık matrisinden yararlanılır. Karmaşıklık matrisi (confusion matrix), yapılan tahminleri gerçek değerlerle karşılaştırarak tahminlerin

doğruluğu hakkında bilgi veren bir tablodur. Örnek bir karmaşıklık matrisi şekil 5.1.'de gösterilmektedir.

		Gerçek	
Tahmin Edilen	Sonuç	Pozitif	Negatif
	Pozitif	DP	YP
	Negatif	YN	DN

Şekil 5.1. Örnek Karmaşıklık Matrisi

Matriste Doğru Pozitifler (DP), Doğru Negatifler (DN), Yanlış Pozitifler (YP) ve Yanlış Negatifler (YN) bulunmaktadır. Doğru pozitifler gerçek değerleri pozitif olup doğru tahmin edilen örneklerin, doğru negatifler gerçek değeri negatif olup doğru tahmin edilen örneklerin, yanlış pozitifler gerçek değeri negatifken pozitif olarak tahmin edilen örneklerin, yanlış negatifler de negatif olarak tahmin edilip gerçek değeri pozitif olan örneklerin sayısını gösterir. Bu değerler kullanılarak aşağıdaki metrikler hesaplanabilir:

- Doğruluk (Accuracy): Doğru tahmin edilen örnek sayısının toplam örnek sayısına olan oranıdır. Yanlış pozitif ve yanlış negatiflerin sayısı dikkate alınmadığı için dengeli olmayan veri kümesiyle yapılan testlerde yanıltıcı sonuçlarla karşılaşılabilir. Bu nedenle tek başına yeterli bir performans metriği değildir. Doğruluk değeri Eşitlik (1)' de verilen formülle hesaplanır.

$$\text{Doğruluk} = \frac{\text{DP+DN}}{\text{DP+DN+YP+YN}} \tag{1}$$

- Duyarlılık (Precision): Doğru tahmin edilen örnek sayısının, tahmin edilen toplam pozitif örnek sayısına oranıdır. Eşitlik (2)'de verilen formülle hesaplanır.

$$\text{Duyarlılık} = \frac{\text{DP}}{\text{DP+YP}} \tag{2}$$

- Hassasiyet (Recall): Doğru tahmin edilen örnek sayısının, toplam pozitif örnek sayısına oranıdır. Eşitlik (3)'te verilen formülle hesaplanır.

$$\text{Hassasiyet} = \frac{\text{DP}}{\text{DP} + \text{YN}} \tag{3}$$

- F1 skoru: Hassasiyet ve duyarlılık değerlerinin harmonik ortalaması alınarak hesaplanır. Yanlış pozitifleri ve yanlış negatifleri de hesaba kattığından doğruluk değerine göre daha yararlı bir metrik olduğu söylenebilir. Eşitlik (4)'te verilen formülle hesaplanır.

$$\text{F1 skoru} = \frac{2 * \text{Hassasiyet} * \text{Duyarlılık}}{\text{Hassasiyet} + \text{Duyarlılık}} \tag{4}$$

Çok sınıflı sınıflandırmada tanımı verilen bu metrikler ikili sınıflandırmadaki anlamlarına göre farklılık gösterebilirler. Çok sınıflı sınıflandırma sonucu üretilen örnek bir karmaşıklık matrisi Şekil 5.2.'de görülmektedir.

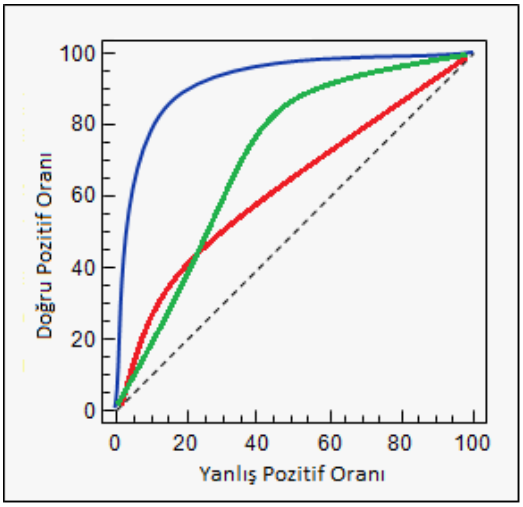
		Gerçek		
Tahmin Edilen	Sınıf	A	B	C
	A	100	20	60
	B	40	80	20
	C	10	40	70

Şekil 5.2. Çok sınıflı sınıflandırma karmaşıklık matrisi

Şekildeki matris incelendiğinde A sınıfı için DP değeri 100, YN değeri 50'dir. Ayrıca normalde B sınıfına ait olan 20 örnek ve normalde C sınıfına ait olan 60 örnek yine A olarak işaretlendiği için A sınıfının YP değerinin 80 olduğu görünmektedir. Burada A sınıfı için duyarlılık 100/180, hassasiyet ise 100/150 olarak hesaplanabilir.

Sınıflandırma başarımını değerlendirmede kullanılan ROC (Receiver Operating Characteristic) eğrisi ve eğri altında kalan alan anlamına gelen AUC (Area Under Curve) değeri de performans analizinde kullanılan metriklerdir. Bir ROC eğrisi, farklı eşik değerleri

için dikey eksen üzerinde doğru pozitiflerin tüm pozitiflere oranlarının, yatay eksen üzerinde ise yanlış pozitiflerin tüm negatiflere oranlarının yer aldığı bir eğridir. Doğru pozitif oranının yüksek, yanlış pozitif oranının düşük olması modelin başarılı olduğunu gösterir. Böyle bir durumda eğri sol üst köşeye doğru yaklaşmaktadır. ROC eğrisinde modelin başarısının ifade edilmesi için AUC değeri kullanılmaktadır. AUC değerinin büyük olması sınıflandırmanın doğru yapıldığını göstermektedir. Örnek bir ROC grafiği Şekil 5.3.'te gösterilmektedir. Çok sınıflı sınıflandırma işlemlerinde ROC eğrisini kullanabilmek için çıktının ikili hale getirilmesi gerekmektedir. Bu şekilde her bir sınıf için bir ROC eğrisi çizilebilir [34].



Şekil 5.3. Örnek ROC Eğrisi

Çok sınıflı sınıflandırma işleminde her sınıf için ayrı ayrı duyarlılık, hassasiyet ve F1 skoru değerlerinin incelenmesi zordur. Bu nedenle bu tez çalışmasında, sınıflandırma algoritmalarını değerlendirme metriği olarak doğruluk, makro duyarlılık (MaD), Makro Hassasiyet (MaH) ve Makro F1 skoru (MaF) kullanılmıştır. Doğruluk değeri sınıflandırma algoritması bazında hesaplanmıştır. Çok sınıflı sınıflandırma işleminde doğruluk değeri, matristeki DP toplamının toplam veri sayısına oranına eşit olmaktadır. Makro duyarlılık ve makro hassasiyet değerleri sınıf bazında hesaplanan duyarlılık ve hassasiyet değerlerinin aritmetik ortalamasıdır. Makro F1 skoru da bu değerlerin harmonik ortalaması alınarak hesaplanmıştır. Her sınıflandırma algoritması için tek bir değer üretebilmek amacıyla makro

ortalama alma işlemi gerçekleştirilmiştir. Makro ortalamalı metriklerin formülleri eşitlik 5, 6 ve 7'de verilmektedir. Formüllerdeki *n* değeri veri kümesindeki sınıf sayısıdır.

$$\mathrm{MaD} = \frac{\sum_{i=1}^{n} \frac{\mathrm{DP_i}}{\mathrm{DP_i + YP_i}}}{\mathrm{n}} \tag{5}$$

$$\mathrm{MaH} = \frac{\sum_{i=1}^{n} \frac{\mathrm{DP_i}}{\mathrm{DP_i + YN_i}}}{\mathrm{n}} \tag{6}$$

$$\mathrm{MaF} = \frac{2 * \mathrm{MaD} * \mathrm{MaH}}{\mathrm{MaD+MaH}} \tag{7}$$

5.2. AMD Veri Kümesi Üzerinde Performans Analizi

Sınıflandırma algoritmalarının performansını ölçebilmek için ilk aşamada 10 parçalı çapraz doğrulama (10-fold cross validation) işlemi yapılmıştır. 10 parçalı çapraz doğrulamada veri kümesi rastgele 10 gruba ayrılır ve gruplardan biri test kümesi olarak ayrılırken geri kalanlar eğitim kümesi olarak kullanılır. Her grup bir defa test kümesi olacak şekilde bu işlem 10 defa tekrarlanır ve her tekrarda üretilen sonuçların ortalaması alınarak tek bir sonuç oluşturulur. Modellerin veri kümesindeki tüm veriler ile eğitilmiş ve test edilmiş olması, yeni karşılaşılan veriler üzerinde ne kadar iyi performans göstereceğine ilişkin daha iyi bir fikir vermektedir. Bu durum göz önüne alınarak 10 parçalı çapraz doğrulama işlemi Scikit-Learn kütüphanesindeki metotlar yardımıyla Şekil 5.4.'te verilen kod satırlarıyla gerçekleştirilmiştir.

```
kfold = StratifiedKFold(y=Y, n_folds=10)
for k, (train_index,test_index) in enumerate(kfold):
    model.fit(X[train_index], Y[train_index])
    score = model.score(X[test_index], Y[test_index])
    print('Fold: %s, Accuracy:%.4f' % (k+1, score))
```

Şekil 5.4. 10 parçalı çapraz doğrulamanın gerçekleştirimi

Verilen koddan da görüleceği üzere model 10 farklı eğitim kümesiyle eğitilmekte ve 10 farklı test kümesiyle test edilmektedir. Tüm sınıflandırma algoritmaları için her bir iterasyonda elde edilen doğruluk değerleri, bu değerlerin ortalaması ve standart sapması Çizelge 5.1.'de gösterilmektedir.

Çizelge 5.1. 10 parçalı çapraz doğrulama ile elde edilen doğruluk değerleri

SINIFLANDIRMA ALGORİTMASI	DOĞRULUK (%)
KNN	96.51 (+-1.44)
DT	97.12 (+-1.71)
LR	98.47 (+-0.96)
SVM	**98.86 (+-0.98)**
RF	98.18 (+-1.24)
ADABOOST	98.68 (+-1.16)
MV	98.31 (+-1.40)
MLP	98.67 (+-0.96)

İkinci aşamada veri kümesi 2 eşit parçaya bölünerek yarısı eğitim verisi diğer yarısı da test verisi yapılmıştır. Bu şekilde sınıflandırma algoritmalarının eğitim aşamasında yer almayan verileri sınıflandırma performansı değerlendirilmek istenmiştir. Veri kümesi dengeli bir veri kümesi olmadığından bölme işlemi yazılım ailesi bazında yapılmış ve her yazılım ailesinde bulunan yazılımların yarısı rastgele seçilerek test verisi olarak işaretlenmiştir. Scikit-Learn kütüphanesinin sağladığı metotlar yardımıyla Şekil 5.5.'te verilen kod satırlarıyla sınıflandırma işlemi gerçekleştirilmiştir.

```
model.fit(X_train, Y_train)
prediction = model.predict(X_test)
accuracy = accuracy_score(Y_test, prediction)
confMat = confusion_matrix(y_true=Y_test, y_pred=prediction)
print('Accuracy:%.4f' % (accuracy))
print(confMat)
```

Şekil 5.5. Eğitim aşamasında yer almayan verilerin sınıflandırılması

Kodda kullanılan X_train ve Y_train eğitim verilerini, X_test ve Y_test ise test verilerini göstermektedir. Her bir sınıflandırma algoritması eğitim verileriyle eğitildikten sonra test verileriyle test edilmiş ve elde edilen doğruluk değerleri ve karmaşıklık matrisleri kaydedilmiştir. Bu deneye ilişkin doğruluk değerleri Çizelge 5.2.'de gösterilmektedir.

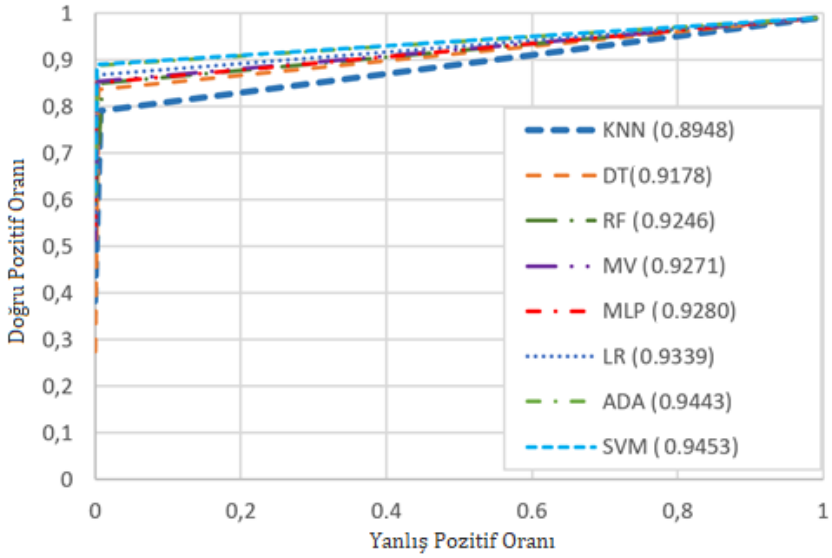
Çizelge 5.2. Test verilerinin sınıflandırılması sonucu elde edilen doğruluk değerleri

SINIFLANDIRMA ALGORİTMASI	DOĞRULUK (%)
KNN	96.98
DT	97.27
LR	98.87
SVM	**99.12**
RF	98.42
ADABOOST	99.08
MV	98.63
MLP	99.00

Çizelgedeki değerler incelendiğinde 10 parçalı çapraz doğrulama ile elde edilen değerlerle yakın değerler üretildiği görülebilir. Kullanılan tüm algoritmalar %96'nın üzerinde doğruluk yüzdesiyle sınıflandırma yapabilmiştir. Bu durum, kullanılan özniteliklerin zararlı yazılım ailelerinin karakterlerini belirlemede oldukça etkili olduğunu göstermektedir. Algoritmalar arasında en iyi sınıflandırma performansını SVM algoritmasının gösterdiği görünmektedir. Topluluk algoritmaları arasında en doğru sınıflandırma AdaBoost tarafından yapılmış ve KNN, DT ve LR algoritmalarının bir araya getirilmesiyle oluşturulan topluluk algoritmasının KNN ve DT algoritmasının performansını yükselttiği görülmüştür.

Algoritmaların sınıflandırma başarısının ROC eğrisinde görünümü Şekil 5.6.'da verilmektedir. ROC eğrisi Scikit-Learn paketinin sağladığı metotlarla çizilmiş ve çizme işlemini gerçekleştiren kod EK 3'te paylaşılmıştır. Veri kümesinde bulunan 71 farklı sonuç değeri için ayrı ayrı eğri çizilmesinin anlaşılmaz bir görüntü oluşturacağı göz önünde bulundurulmuş ve ROC eğrisi her bir sonuç değeri için elde edilen ortalama değerlerle

çizdirilmiştir. Eğrilerin altında kalan alanlar incelendiğinde en büyük alanın SVM algoritmasına ait olduğu görülmektedir.



Şekil 5.6. ROC Eğrisi

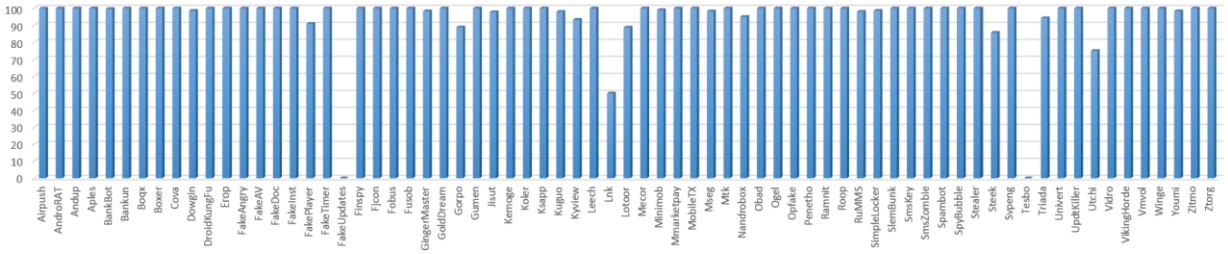
Her bir algoritmanın sınıflandırma işlemi sonucu ürettiği karmaşıklık matrislerinden faydalanılarak Makro Duyarlılık, Makro Hassasiyet ve Makro F1 Skoru değerleri hesaplanmıştır. Bu değerler Çizelge 5.3.'te verilmektedir.

Çizelge 5.3. Makro Duyarlılık, Makro Hassasiyet ve Makro F1 skoru değerleri

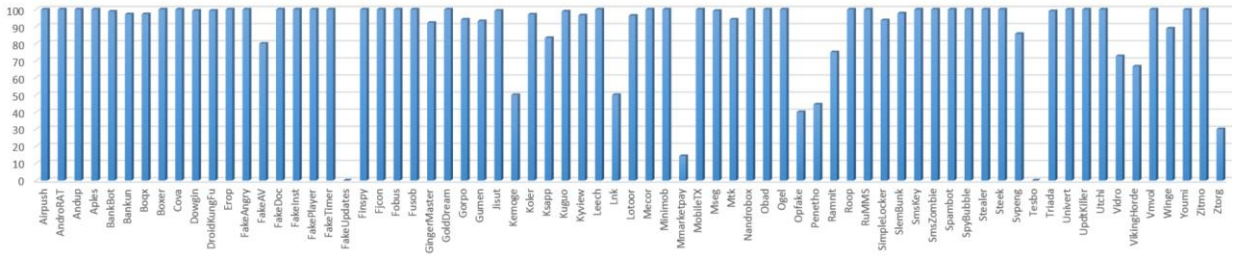
SINIFLANDIRMA ALGORİTMASI	MAD (%)	MAH (%)	MAF
KNN	85.48	79.00	82.11
DT	80.80	84.28	82.50
LR	94.15	87.13	90.50
SVM	**95.02**	**89.06**	**91.94**
RF	93.75	84.93	89.12
AdaBoost	94.69	88.87	91.68
MV	92.61	85.44	88.88
MLP	89.19	86.58	87.86

Tablo incelendiğinde MaD, MaH ve MaF değerlerinin doğruluk değerlerine oranla düşük olduğu görünmektedir. Bu durumun nedeni bazı yazılım ailelerinin duyarlılık ve hassasiyet değerlerinin oldukça düşük olması ve bu düşük değerlerin genel ortalamayı da düşürmesidir. Her bir yazılım ailesi için SVM algoritması tarafından üretilen duyarlılık ve hassasiyet değerlerinin verildiği grafikler Şekil 5.7. ve Şekil 5.8.'de verilmiştir. Bu grafikler incelendiğinde şu sonuçlar çıkarılmaktadır:

- FakeUpdates ve Tesbo ailelerine ait yazılımların hiçbiri doğru sınıflandırılamamıştır.
- Kemoge, Lnk, MmarketPay, Opfake, Penetho, Ramnit, Vidro, VikingHorde ve Ztorg ailelerine ait yazılımlar düşük doğrulukla sınıflandırılmışlardır.
- FakePlayer, Steek ve Utchi ailelerine ait yazılımlar yüksek doğrulukla sınıflandırılmalarına rağmen ailelerin duyarlılık değerleri genel ortalamayı düşürmektedir.



Şekil 5.7. Her bir yazılım ailesi için hesaplanan duyarlılık değerleri grafiği



Şekil 5.8. Her bir yazılım ailesi için hesaplanan hassasiyet değerleri grafiği

Bu sonuçlarla birlikte Çizelge 5.4. incelendiğinde ortalamaları düşüren yazılım ailelerinin ortak özelliğinin az örnek içermesi olduğu görülebilir. SVM algoritması, 40'tan az örnek içeren bazı yazılım ailelerinin örneklerini sınıflandırmada sorun yaşamıştır. Bu yazılım ailelerine ait örnekler, yazılım ailesi yeterince öğrenilemediği için başka ailelere ait örneklere

benzetilmiştir. Buna rağmen örnek sayısı az olan yazılım ailelerinin çoğunluğunda başarı sağlandığı göz önüne alınırsa, algoritmanın başarılı olduğu, ortalamayı düşüren yazılım ailelerinin diğer ailelerden yeterince farklı olmadığı söylenebilir.

Çizelge 5.4. Az örnek içeren yazılım aileleri

ZARARLI YAZILIM AİLELERİ	ÖRNEK SAYISI
Fobus, FakeAV, FakeUpdates, Lnk, Tesbo, Ogel, VikingHorde, Ramnit, Obad, SmsZombie, FakeAngry, Opfake, SpyBubble, Univert	<= 10
FakeTimer, Steek, Utchi, Svpeng, Vmvol, Mmarketpay, Kemoge, Spambot, Fjcon, Cova, MobileTX, Penetho, Winge, Ztorg	>10 ve <= 20
Aples, FakeDoc, FakePlayer, Vidro, UpdtKiller, Zitmo, Stealer, Gorpo, Ksapp	>20 ve <=40

5.3. Öznitelik Analizi

Tez kapsamında zararlı Android yazılımlarını ailelerine göre sınıflandırma işlemi dışında sınıflandırma öncesi AMD veri kümesindeki her bir yazılımdan çıkarılan öznitelikler incelenmiş ve özniteliklerin bulunma sıklıklarına göre hangi zararlı yazılım türünün ya da zararlı yazılım ailesinin hangi öznitelikleri daha sık kullandıkları incelenmiştir. Bu incelemeye göre aşağıdaki sonuçlar elde edilmiştir:

- *Airpush*, *Kuguo*, *Youmi* gibi ailelere ait reklam yazılımı türündeki yazılımların büyük çoğunluğu *NotificationManager* sınıfında bulunan *notify* metodunu kullanarak kullanıcılara bildirim göndermektedir.
- Bankacılık sistemlerini hedef alan Truva atlarının (Trojan-Banker) büyük bir kısmı *SmsManager* sınıfını kullanarak kullanıcılara SMS göndermektedir. *BankBot* ve *SlemBunk* aileleri bu tip Truva atı yazılımları içermektedir.
- *SQLite* veri tabanında işlem yapabilmek için metotlar sunan *SqliteDatabase* sınıfı kötü amaçlı yazılımların algılanmasını engelleyen (Trojan-Dropper) türündeki zararlı yazılımlar tarafından sıklıkla kullanılırken fidye yazılımlarının birkaçı bu sınıfı kullanmaktadır.

- Reklam yazılımları *Location* ve *LocationManager* sınıflarını kullanarak kullanıcının konumunu tespit ederler. Bunun yanında reklam yazılımı türünde zararlı yazılımlar içeren *Airpush* ailesindeki yazılımların birçoğu *Address* sınıfının *getCountryCode, getCountryName* ve *getPostalCode* metotlarını kullanmaktadır.
- Arka kapı ve Truva atı yazılımlarının çoğunluğu *TelephonyManager* sınıfının *getDeviceId, getSubscriberId, getLine1Number, getNetworkType* gibi metotları kullanarak kullanıcının telefonu hakkında bilgi toplarlar.
- Fidye ve korsanlık aracı türündeki yazılımların birçoğu Internet izni talep etmemektedir. Özellikle *Jisut* ailesindeki zararlı yazılımların neredeyse tamamı çevrimdışı çalışmaktadır.
- Fidye yazılımlarının büyük çoğunluğu *read_call_log, read_contacts* ve *write_contacts* izinlerini talep etmektedir.

Belirlenen bu özellikler AMD veri kümesinde bulunan zararlı yazılımlar için belirlenmiş olup, zararlı yazılımların kesin davranışları değildir. Zararlı yazılım ailelerine yeni zararlı yazılımlar dahil oldukça bu özellikler değişebilir. Özniteliklerle zararlı yazılımlar arasında kurulan bu ilişkiler, zararlı yazılımların evrimsel gelişimlerini inceleyebilmek açısından önemlidir.

5.4. Drebin ve UpDroid Veri Kümeleri Üzerinde Performans Analizi

AMD veri kümesindeki zararlı Android yazılımlarının ait oldukları yazılım ailelerine göre sınıflandırılma başarısının doğruluğunu teyit edebilmek amacıyla sınıflandırma işlemi Drebin ve UpDroid veri kümelerinde de gerçekleştirilmiştir. Drebin veri kümesinde yalnızca 1 örnek içeren zararlı yazılım aileleri çıkarılmış ve geriye 132 zararlı yazılım ailesine ait 5507 zararlı yazılım kalmıştır. 2 veri kümesi de, önceki deneyde olduğu gibi yarısı eğitim kümesi yarısı da test kümesi olacak şekilde ayrılmıştır.

Sınıflandırma algoritmalarının eğitilmesinde önceki deneyde kullanılan en etkili 1000 öznitelik kullanılmıştır. Algoritmaların hiper-parametreleri de değiştirilmemiş ve aynı konfigürasyonlarla sınıflandırma işlemi gerçekleştirilmiştir. Önceki deneyde olduğu gibi

doğruluk değerinin yanında MaD, MaH ve MaF değerleri de hesaplanmıştır. Sınıflandırma sonuçları Çizelge 5.5.'te verilmiştir.

Çizelge 5.5. Drebin (D) ve UpDroid (U) veri kümelerinde sınıflandırma sonuçları

SINIFLANDIRMA ALGORİTMASI	DOĞRULUK (%)		MAD (%)		MAH (%)		MAF	
	D	U	D	U	D	U	D	U
KNN	91.59	91.99	79.12	73.32	75.45	72.25	77.24	72.78
DT	92.93	90.83	77.39	72.24	78.61	71.25	77.99	71.74
LR	96.51	93.24	**93.36**	90.45	87.61	85.43	90.39	87.86
SVM	96.66	**94.66**	91.99	93.49	84.64	**89.47**	88.16	91.43
RF	96.37	94.25	91.47	**96.76**	85.89	94.89	88.59	**95.81**
AdaBoost	**96.79**	93.74	92.80	83.90	**88.62**	80.30	**90.66**	82.06
MV	94.84	92,16	88.51	86,86	80.14	83.96	84.11	85.38
MLP	96.10	93.66	92.14	88.30	84.37	84.13	88.08	86.16

Sonuçlar incelendiğinde UpDroid veri kümesinde yapılan deneyde en yüksek doğruluk değerinin AMD veri kümesiyle yapılan deneyde olduğu gibi SVM algoritmasıyla elde edildiği görülmektedir. Drebin veri kümesinde ise en etkili performansı AdaBoost algoritması göstermiştir. Bu algoritmalar dışında LR, RF ve MLP algoritmaları da iyi performans gösteren algoritmalardan olmuşlardır. Elde edilen değerlerin AMD veri kümesinde yapılan sınıflandırma sonucu değerlerinin altında olması, Drebin ve UpDroid veri kümelerinin AMD veri kümesine oranla daha küçük olmasıyla açıklanabilir. Daha çok verinin olması algoritmaların daha iyi eğitilmesini, dolayısıyla daha doğru sonuçlar üretilmesini sağlamaktadır. Ayrıca bu veri kümeleri de AMD veri kümesi gibi dengesiz veri kümeleri olduğundan, düşük Makro ortalamalı değerlerle karşılaşılmıştır.

Android zararlı yazılımlarını ailelerine göre sınıflandıran benzer çalışmalardan DroidSieve [52] ve UpDroid [64], Drebin veri kümesindeki verileri sırasıyla %97,68 ve %96.85 başarıyla sınıflandırmışlardır. Ancak bu deneylerde az örnek içeren zararlı yazılım aileleri veri kümesinden çıkarılmış ve DroidSieve çalışmasında 108, UpDroid çalışmasında da sadece 25 zararlı yazılım ailesinin verileri sınıflandırmaya dahil edilmiştir. Bunlardan farklı olarak bu

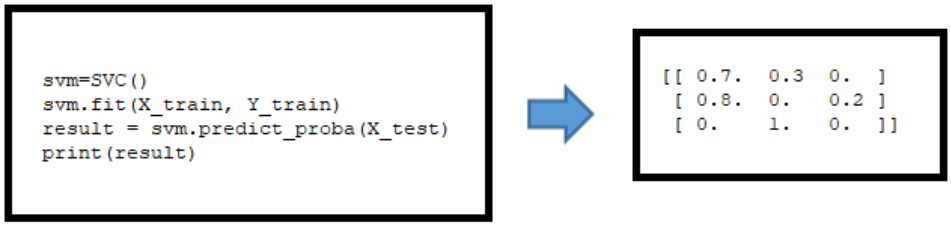
çalışma kapsamında Drebin veri kümesinden sadece tek veri içeren zararlı yazılımlar çıkarılmış ve toplam 132 zararlı yazılım ailesi veri kümesinde kullanılmıştır. Bu nedenle AdaBoost algoritmasının ulaştığı %96.79 doğruluk değerinin bahsedilen çalışmalardakilere göre daha başarılı bir değer olduğu düşünülmektedir.

UpDroid [64] çalışmasında Drebin dışında UpDroid veri kümesinde de sınıflandırma işlemi gerçekleştirilmiş ve %96.37'lik bir başarı oranı elde edilmiştir. Veri kümesinde bu çalışma kapsamında yapılan sınıflandırma sonucunda ise SVM algoritması 94.66 ile sınıflandırma yapmıştır. UpDroid çalışmasının daha yüksek bir başarı oranına ulaşmasında çalışmada kullanılan dinamik özniteliklerin etkili olduğu düşünülmektedir. Yalnızca statik analiz ile UpDroid çalışmasının performansına yakın bir performans gösterilmesi de oldukça önemlidir.

5.5. Bilinmeyen Zararlı Yazılımların Sınıflandırılması

Şimdiye kadar yapılan sınıflandırma çalışmalarında ailesi tahmin edilecek olan zararlı Android yazılımları, önceden bilinen ve sınıflandırıcıların eğitilmesinde kullanılan zararlı yazılım ailelerine ait yazılımlardan seçilmiştir. Bu nedenle zararlı bir yazılımın ailesi tahmin edilirken sınıflandırma algoritmaları, yalnızca önceden karşılaşılan ailelerden birini seçebilecektir. Dolayısıyla önceden karşılaşılmamış, yeni bir zararlı yazılım ailesine ait örnek sınıflandırma algoritmaları tarafından algılanamamaktadır. Her ne kadar bilinen zararlı yazılımları ailelerine göre sınıflandırmak zararlı yazılım analizi kapsamında önemli olsa da, bilinmeyen bir aileye ait zararlı yazılımları tespit edip bilinmeyen olarak işaretleyebilmek de oldukça önemlidir. Bu önem göz önünde bulundurulmuş ve tez kapsamında son olarak bu konu üzerinde çalışılmıştır.

Bilinmeyen ailelere ait zararlı Android yazılımı örneklerini tespit edebilmek için sınıflandırma algoritmalarının ürettikleri tahmin olasılığı değerleri kullanılmıştır. Scikit-learn kütüphanesinde bulunan *predict_proba* metodu her bir test verisinin sınıflara ait olma olasılığını çıktı olarak vermektedir. Metodun A, B ve C adındaki 3 farklı hedef sınıftan birine sınıflandırılabilecek 3 test verisi üzerinde örnek kullanımı ve üretilen çıktı aşağıdaki şekilde gösterilmektedir.



Şekil 5.9. Örnek predict_proba kullanımı ve çıktısı

Verilen örnekte SVM sınıflandırma algoritması eğitim kümesiyle eğitilmiş ve ardından test verileri *predict_proba* metoduna girdi olarak verilmiştir. Toplam 3 sınıf değeri ve 3 test verisi olduğu için 3x3'lük bir matris çıktı olarak üretilmiştir. Bu matriste her test verisi için bir sınıfa ait olma olasılığı verilmektedir. Şekildeki örnekte SVM algoritması ilk test verisininin sınıfını %70 olasılıkla A, %30 olasılıkla B olarak tahmin etmiştir. Aynı şekilde ikinci verinin %80 olasılıkla A, %20 olasılıkla C, son verinin de %100 olasılıkla B sınıfına ait olduğu tahmin edilmiştir. konu üzerinde çalışılmıştır.

Bu çalışmada *predict_proba* metodu tarafından üretilen olasılık değerleri kullanılarak bir model geliştirilmiştir. Modelde olasılık değerlerinin bir tek sınıf için çok yüksek olması durumunda verinin o sınıfa ait olduğu, aksi durumda verinin bilinmeyen bir veri olduğu varsayılmaktadır. Sınıflandırma algoritmasının bilinen test verileri üzerindeki sınıflandırma işlemi sonrasında elde edilen olasılık değerlerine göre bir eşik değeri belirlenmekte ve veriler için üretilen en büyük tahmin olasılığının bu değerden küçük olması durumunda ilgili veri bilinmeyen olarak işaretlenmektedir.

Tasarlanan bu modelin başarılı olup olmadığının görülebilmesi için AMD ve Drebin veri kümelerinde ayrı ayrı test yapılmıştır. İlk adımda AMD ve Drebin veri kümelerinde 25'ten az örnek içeren yazılım ailelerine ait zararlı yazılımlar bilinmeyen zararlı yazılımlar olarak ayrılmıştır. Ardından diğer verilerin %80'i eğitim veri kümesi olarak rastgele seçilmiş, kalanlar da test veri kümesi yapılmıştır. Önceki deneylerde sınıflandırma performansı yüksek olan SVM sınıflandırma algoritması test için seçilmiş ve eğitim kümeleri kullanılarak eğitilmiştir. Ardından test kümeleri kullanılarak sınıflandırma işlemi yapılmış ancak bu defa önceki testlerden farklı olarak *predict_proba* metodu kullanılmış ve her bir test verisi için

tahmin olasılıkları elde edilmiştir. Olasılık değerlerinin ortalamaları (μ_p) ve standart sapmaları (σ_p) kullanılarak farklı eşik değerleri belirlenmiştir. Ardından ilk aşamada bilinmeyen olarak işaretlenen veriler test kümelerine dahil edilmiş ve sınıflandırma işlemi bu test kümeleriyle tekrarlanmıştır. Sınıflandırma sonucunda verilerin hepsi bilinen ailelerden birine sınıflandırılmış olsa da, üretilen olasılık değerleri eşik değerleriyle kıyaslanarak test verisinin bilinmeyen olup olmadığı belirlenmiştir. Veri bilinmeyen olarak işaretlendiğinde pozitif, bilinen olarak işaretlendiğinde negatif olarak kabul edilmiş ve her iki veri kümesinde yapılan testler sonucunda elde edilen Doğru Pozitif (DP), Doğru Negatif (DN), Yanlış Pozitif (YP), Yanlış Negatif (YN) sayıları ve bu sayılardan elde edilen Duyarlılık, Hassasiyet, F1-Skoru ve Doğruluk değerleri Çizelge 5.6.'da ve Çizelge 5.7.'de gösterildiği gibi elde edilmiştir.

Çizelge 5.6. AMD veri kümesinde bilinmeyen verileri sınıflandırma sonuçları

Eşik Değeri	DP	DN	YP	YN	Duyarlılık (%)	Hassasiyet (%)	F1 Skoru	Doğruluk (%)
$\mu_p - 2\sigma_p = 85.68$	365	4635	174	119	**67.71**	75.41	71.35	**94.46**
$\mu_p - \sigma_p = 91.7$	422	4534	275	62	60.54	87.19	**71.46**	93.63
$\mu_p - 2/3\sigma_p = 93.7$	458	4469	340	26	57.39	94.62	71.45	93.08
$\mu_p - 1/2\sigma_p = 94.71$	478	4355	454	6	51.28	98.76	67.51	91.31
$\mu_p - 1/3\sigma_p = 95.71$	480	4238	571	4	45.67	**99.17**	62.54	89.13

Çizelge 5.7. Drebin veri kümesinde bilinmeyen verileri sınıflandırma sonuçları

Eşik Değeri	DP	DN	YP	YN	Duyarlılık (%)	Hassasiyet (%)	F1 Skoru	Doğruluk (%)
$\mu_p - 2\sigma_p = 76.68$	650	903	45	186	**93.52**	77.75	84.91	**87.05**
$\mu_p - \sigma_p = 85.15$	702	839	109	134	86.93	84.02	85.45	86.54
$\mu_p - 2/3\sigma_p = 87.98$	734	800	148	102	83.22	87.79	85.44	85.98
$\mu_p - 1/2\sigma_p = 89.39$	775	749	199	61	79.56	92.70	**85.63**	85.42
$\mu_p - 1/3\sigma_p = 90.8$	805	708	246	25	76.59	**96.99**	85.59	84.81

Çizelgelerde verilen değerler incelendiğinde tanımlanan eşik değerlerinin hesaplanan değerler üzerindeki etkisi görülebilmektedir. Eşik değerinin yüksek tanımlanması,

bilinmeyen verilerin daha doğru tespit edilebilmesini sağlarken, bilinen verilerin de bilinmeyen olarak algılanma riskini arttırmaktadır. F1 skorları ve genel doğruluk oranı değerleri de göz önüne alındığında en uygun eşik değerinin $\mu_p - \sigma_p$ ile elde edilen değer olduğu söylenebilir. Bu eşik değeri kullanılarak geliştirilen modelin önceden karşılaşılmamış zararlı yazılımları tespit edebilme konusunda başarılı olacağı öngörülmektedir.

6. SONUÇLAR VE ÖNERİLER

Kendilerini yenileyerek akıllı telefonlara kolay bir şekilde yüklenip zararlı davranışlarını sergileyebilen Android zararlı yazılımları var olduğu sürece, bu zararlı yazılımları analiz ederek sınıflandırabilen sistemlerin de evrimsel gelişimi, Android işletim sistemi güvenliği kapsamında önem arz etmektedir. Bu durum göz önünde bulundurularak bu tez çalışmasında zararlı Android yazılımlarını ailelerine göre sınıflandıran makine öğrenmesi tabanlı bir sistem geliştirilmiştir. Zararlı Android yazılımlarından statik analiz yöntemiyle elde edilen izin talepleri ve API çağrıları öznitelik olarak kullanılmıştır. Zararlı yazılım geliştiricilerin analizlerde fark edilmemek için kullandığı yeniden adlandırma ve dinamik yükleme (assets klasörü aracılığıyla) yöntemlerine karşı da dayanıklı olan bu özniteliklerin en belirleyici olanları tespit edilmiş ve çeşitli makine öğrenmesi algoritmalarında kullanılmıştır. Doğruluk değerinin yanında duyarlılık, hassasiyet ve F1 skoru gibi metrikler de deneyler sırasında hesaplanmıştır. AMD veri kümesinde yapılan deneylerin doğruluğunun sınanması için deneyler Drebin ve UpDroid veri kümelerindeki verilerle tekrarlanmış ve yakın sonuçlar elde edildiği görülmüştür. Deneylerde tüm makine öğrenmesi algoritmaları iyi performans göstermiş olsalar da Destek Vektör Makineleri ve AdaBoost algoritmalarının diğer algoritmalara oranla daha başarılı olduğu görülmüştür. Ayrıca daha önceden karşılaşılmamış bir zararlı yazılım ailesine ait olan zararlı yazılımların da bilinmeyen olarak tespit edilebilmesi için çalışma yapılmış ve önerilen modelin bu sınıflandırmayı başarılı bir şekilde yapabildiği görülmüştür. Sınıflandırma işlemleri dışında bazı özniteliklerin AMD veri kümesinde bulunan zararlı yazılım türleriyle ilişkileri incelenerek raporlanmıştır.

Deneyler sonucunda makine öğrenmesi algoritmalarının hepsi yüksek doğruluk değerleriyle sınıflandırma yapmasına rağmen makro ortalamalı duyarlılık, hassasiyet ve F1 skoru değerleri doğruluk değerlerine göre daha düşük çıkmıştır. Bunun nedeni, deneylerde kullanılan veri kümelerinde az örnek içeren zararlı yazılım ailelerinin çokluğudur. Az örnek içeren zararlı yazılım ailelerinden bazılarına ait yazılımlar, sınıflandırma algoritmaları tarafından yeterince iyi ayırt edilememiş ve diğer yazılım ailelerinin örneklerine benzetilmiştir. Ancak az örnek içeren yazılım ailelerinin çoğunluğunda bu sıkıntı

yaşanmadığından veri kümelerinde bulunan yazılımların bazılarının gruplandırılmasında yanlışlık olduğu değerlendirilmektedir.

Gelecek çalışmalarda, dengeli olmayan veri kümelerinde sınıflandırma işleminin daha iyi yapılması için çalışma yapılarak az örnek içeren zararlı yazılım ailelerinin de sorunsuz bir şekilde sınıflandırılması sağlanabilir. Kullanılan özniteliklerin çeşidi arttırılarak daha iyi sonuçların alınması hedeflenebilir. Örneğin, API metotlarının çağrılıp çağrılmadığı bilgisinin dışında, kaç defa çağrıldığı veya hangi API metoduyla birlikte kullanıldığı gibi bilgiler analiz sonucu elde edilebilir. Bunun için dinamik analiz yöntemlerinden de yararlanılabilir ve dinamik analiz ile zararlı Android yazılımlarının ağ hareketleri ve kaynak kullanımı gibi bilgileri de öznitelik olarak kullanılabilir. Dinamik analizlerde rastgele girdi üretme amacıyla kullanılan Monkey aracı yerine Droidutan [24] ve DroidBot [25] gibi daha akıllı girdi üretme araçları kullanılarak yazılımların daha doğru ve sistemli bir şekilde analiz edilmesi sağlanabilir. Bilinmeyen zararlı Android yazılımlarının sınıflandırılmasında zararlı yazılımların bilinmeyen olarak işaretlenmesinin yanında var olan zararlı yazılım ailelerine benzerlikleri de tanımlanan birtakım metriklerle gösterilebilir. Son olarak, Android zararlı yazılımlarının sayısının sürekli arttığı düşünüldüğünde, bu zararlı yazılımların özelliklerinin analiz edilmesi ve evrimsel gelişimlerinin izlenmesi de Android işletim sistemi güvenliği kapsamında önemli bir çalışma konusudur.

KAYNAKLAR

[1] Global mobile OS market share in sales to end users from 1st quarter 2009 to 2nd quarter 2018, https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/ (Erişim Tarihi: **25 Mayıs 2019**).

[2] Android and Google Play statistics, https://www.appbrain.com/stats/stats-index/ (Erişim Tarihi: **25 Mayıs 2019**).

[3] More than 99 percent of all malware designed for mobile devices targets Android devices, explained Olaf Pursche, Head of Communications at AV-TEST, in the F-Secure State of Cyber Security 2017.

[4] Another Reason 99% of Mobile Malware Targets Androids, https://blog.f-secure.com/another-reason-99-percent-of-mobile-malware-targets-androids/ (Erişim Tarihi: **25 Mayıs 2019**).

[5] Android Malware Dataset, http://amd.arguslab.org/ (Erişim Tarihi: **25 Mayıs 2019**).

[6] D. W. Aha, D. Kibler, and M. K. Albert. Instance-Based Learning Algorithms. Machine Learning, 6:3766, 1991

[7] Classification and Regression Trees, Leo Breiman, Jerome Friedman, Charles J. Stone, R.A. Olshen (1984)

[8] Cox, D.R. (1958). The Regression Analysis of Binary Sequences. Journal of the Royal Statistical Society: Series B, 20, 215-242.

[9] V. Vapnik. The Nature of Statistical Learning Theory. Springer-Verlag, NY, 1995

[10] Freund, Yoav & E. Schapire, Robert. (1999). Large Margin Classification Using the Perceptron Algorithm. Machine Learning. 37. 10.1023/A:1007662407062.

[11] Leo Breiman. Random Forests. , University of California, Berkeley, Journal of Machine Learning, Vol. 45 Issue 1, pp 5 - 32, October, 2001

[12] Yoav Freund, and E. Schapire. Experiments with a New Boosting Algorithm. InProc. of the Thirteenth International Conference, 1996.

[13] Ismail, Najiahtul & Saad, H & Robiah, Y & Abdollah, Mohd. (2017). General android malware behaviour taxonomy. Defence S and T Technical Bulletin. 10. 160-168.

[14] Zhou, Y. and Jiang, X. (2012). Dissecting android malware: Characterization and evolution. Security and Privacy (SP), 95-109.

[15] dex2jar, https://code.google.com/p/dex2jar/ (Erişim Tarihi: **25 Mayıs 2019**).

[16] Procyon, https://github.com/ststeiger/procyon/ (Erişim Tarihi: **25 Mayıs 2019**).

[17] Jd-cmd, https://github.com/kwart/jd-cmd/ (Erişim Tarihi: **25 Mayıs 2019**).

[18] Jadx, https://github.com/skylot/jadx/ (Erişim Tarihi: **25 Mayıs 2019**).

[19] CFR, http://www.benf.org/other/cfr/ (Erişim Tarihi: **25 Mayıs 2019**).

[20] Smali, https://github.com/JesusFreke/smali/ (Erişim Tarihi: **25 Mayıs 2019**).

[21] Apktool, https://ibotpeaches.github.io/Apktool/ (Erişim Tarihi: **25 Mayıs 2019**).

[22] Androguard, https://github.com/androguard/androguard (Erişim Tarihi: **25 Mayıs 2019**).

[23] UI Exerciser Monkey, https://developer.android.com/studio/test/monkey.html (Erişim Tarihi: **25 Mayıs 2019**).

[24] Droidutan, https://github.com/aleisalem/Droidutan (Erişim Tarihi: **25 Mayıs 2019**).

[25] Droidbot, https://github.com/honeynet/droidbot (Erişim Tarihi: **25 Mayıs 2019**).

[26] Droidbox, https://github.com/pjlantz/droidbox, (Erişim Tarihi: **25 Mayıs 2019**).

[27] Manifest.Permission, https://developer.android.com/reference/android/Manifest.permission (Erişim Tarihi: **25 Mayıs 2019**).

[28] The Drebin Dataset, https://www.sec.cs.tu-bs.de/~danarp/drebin/ (Erişim Tarihi: **25 Mayıs 2019**).

[29] Android Malware Genome Project, http://www.malgenomeproject.org/ (Erişim Tarihi: **25 Mayıs 2019**).

[30] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, “A review on feature selection in mobile malware detection,” Digital Investigation, vol. 13, no. 0, pp. 22 – 37, 2015.

[31] Wei, Fengguo & Li, Yuping & Roy, Sankardas & Ou, Xinming & Zhou, Wu. (2017). Deep Ground Truth Analysis of Current Android Malware. 252-276. 10.1007/978-3-319-60876-1_12.

[32] scikit-learn, https://scikit-learn.org/ (Erişim Tarihi: **25 Mayıs 2019**)

[33] Tuning the hyper-parameters of an estimator, https://scikit-learn.org/stable/modules/grid_search.html (Erişim Tarihi: **25 Mayıs 2019**)

[34] Receiver Operating Characteristic (ROC), https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html

[35] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS, San Diego,CA,USA, 2014.

[36] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-level features for robust malware detection in android. In SecureComm, pages 86–103, Sydney,AU, 2013. Springer.

[37] Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. Anastasia: Android malware detection using static analysis of applications. In New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on, pages 1–5. IEEE, 2016.

[38] B. Amos, H. A. Turner, and J. White, "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale," in International Conference on Wireless Communications and Mobile Computing (IWCMC), 2013.

[39] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra. MADAM: A multi-level anomaly detector for android malware. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7531 LNCS:240–253, 2012.

[40] W.-C. Wu and S.-H. Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS '14, pages 247–252, New York, NY, USA, 2014. ACM.

[41] Nancy, Dr. Deepak Kumar Sharma, "Android Malware Detection using Decision Trees and Network Traffic", in IJCSIT, Vol. 7 (4), 2016.

[42] M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In IEEE COMPSAC, 2015.

[43] M. Lindorfer, M. Neugschwandtner, L. Weichselbaum, Y. Fratantonio, V. v. d. Veen and C. Platzer, "ANDRUBIS -- 1,000,000 Apps Later: A View on Current Android Malware Behaviors," 2014 Third International Workshop on Building Analysis Datasets and Gathering Experience Returns for Security (BADGERS), Wroclaw, 2014, pp. 3-17.

[44] F. Yang, Y. Zhuang, J. Wang, "Android Malware Detection Using Hybrid Analysis and Machine Learning Technique", from book Cloud Computing and Security: Third International Conference, ICCCS 2017, Nanjing, China, June 16-18, 2017, Revised Selected Papers, Part II (pp.565-575)

[45] Xu, Lifan & Zhang, Dongping & Jayasena, Nuwan & Cavazos, John. (2018). HADM: Hybrid Analysis for Detection of Malware. 702-724. 10.1007/978-3-319-56991-8_51.

[46] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, Droid-sec: Deep learning in Android malware detection, in Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM, poster), 2014, pp. 371–372.

[47] Xin Su et al. 2016. A Deep Learning Approach to Android Malware Feature Learning and Detection. In IEEE TrustCom. 244–251.

[48] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated familial classification of Android malware," in ACM SIGPLAN Program Protection and Reverse Engineering Workshop, PPREW, 2014.

[49] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S.Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of android malware," Department of Computer Science, George Mason University, Tech. Rep., 2015.

[50] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in Security and Privacy Workshops (SPW), 2016 IEEE. IEEE, 2016, pp. 252–261.

[51] Massarelli, Luca & Aniello, Leonardo & Ciccotelli, Claudio & Querzoni, Leonardo & Ucci, Daniele & Baldoni, Roberto. (2017). Android Malware Family Classification Based on Resource Consumption over Time.

[52] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, “Droidsieve: Fast and accurate classification of obfuscated android malware,” in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. ACM, 2017, pp. 309–320.

[53] T. Chakraborty, F. Pierazzi and V. S. Subrahmanian, "EC2: Ensemble Clustering and Classification for Predicting Android Malware Families," in IEEE Transactions on Dependable and Secure Computing.

[54] Chen, Ricky. (2016). A Brief Introduction on Shannon's Information Theory. 10.13140/RG.2.1.2912.3604.

[55] Android, https://en.wikipedia.org/wiki/Android_(operating_system) (Erişim Tarihi: **25 Mayıs 2019**)

[56] Android-Architecture, https://www.tutorialspoint.com/android/android_architecture (Erişim Tarihi: **25 Mayıs 2019**)

[57] Building and Running, https://stuff.mit.edu/afs/sipb/project/android/docs/tools/building (Erişim Tarihi: **25 Mayıs 2019**)

[58] k-nearest neighbors algorithm, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm (Erişim Tarihi: **25 Mayıs 2019**)

[59] Decision Trees Algorithms, https://medium.com/deep-math-machine-learning-ai/chapter-4-decision-trees-algorithms-b93975f7a1f1 (Erişim Tarihi: **25 Mayıs 2019**)

[60] Support Vector Machine, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (Erişim Tarihi: **25 Mayıs 2019**)

[61] Random Forest Simple Explanation, https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d, (Erişim Tarihi: **25 Mayıs 2019**)

[62] Sebastian Raschka, Python Machine Learning, Packt Publishing, 2015

[63] Multilayer perceptron example, https://github.com/rcassani/mlp-example (Erişim Tarihi: **25 Mayıs 2019**)

[64] Aktas, Kursat & Sen, Sevil. (2018). UpDroid: Updated Android Malware and Its Familial Classification. 10.1007/978-3-030-03638-6_22.

[65] Updroid, https://wise.cs.hacettepe.edu.tr/projects/updroid/dataset/, (Erişim Tarihi: **13 Haziran 2019**)

EKLER

EK 1 – Öznitelik dosyasını oluşturan C# programı

```
11  public class FeatureExtractor
12  {
13      public static void Main(string[] args)
14      {
15          StreamReader permissionsReader = new StreamReader(@"D:\Study\permissionFeatures.txt");
16          StreamReader apisReader = new StreamReader(@"D:\Study\apiCallFeatures.txt");
17
18          List<string> permissions = permissionsReader.ReadToEnd().Split(new[] { "\r\n", "\r", "\n" }, StringSplitOptions.None).ToList();
19          List<string> apiCalls = apisReader.ReadToEnd().Split(new[] { "\r\n", "\r", "\n" }, StringSplitOptions.None).ToList();
20
21          StreamWriter writer = new StreamWriter(Path.Combine("D:\\Study", "features.txt"));
22
23          foreach (string familyFolder in Directory.GetDirectories(@"D:\Study\ApktoolOutputs"))
24          {
25              foreach (string malwareFolder in Directory.GetDirectories(familyFolder))
26              {
27                  string apiCallsFile = Path.Combine(malwareFolder, "apiCalls.txt");
28                  string manifestFile = Path.Combine(malwareFolder, "AndroidManifest.xml");
29
30                  StreamReader apiReader = new StreamReader(apiCallsFile);
31                  StreamReader manifestReader = new StreamReader(manifestFile);
32
33                  string manifest = manifestReader.ReadToEnd();
34                  List<string> apisOfMalware = apiReader.ReadToEnd().Split(new[] { "\r\n", "\r", "\n" }, StringSplitOptions.None).ToList();
35
36                  var match = Regex.Matches(manifest, "<uses-permission.*").Cast<Match>().ToList();
37
38                  foreach (string permission in permissions)
39                  {
40                      writer.Write(match.Count(i => i.Value.Contains("\"" + permission + "\"")) > 0 ? "1," : "0,");
41                  }
42
43                  foreach (string apiCall in apiCalls)
44                  {
45                      writer.Write(apisOfMalware.Contains(apiCall) ? "1," : "0,");
46                  }
47
48                  writer.Write(Path.GetFileName(familyFolder));
49                  writer.Write(writer.NewLine);
50
51                  apiReader.Close();
52                  manifestReader.Close();
53              }
54          }
55
56          permissionsReader.Close();
57          apisReader.Close();
58          writer.Close();
59      }
60  }
```

EK 2 – MajorityVotingClassifer adlı topluluk algoritmasını oluşturan Python kodu

```python
import numpy as np
from sklearn.base import BaseEstimator
from sklearn.base import ClassifierMixin
from sklearn.preprocessing import LabelEncoder
from sklearn.externals import six
from sklearn.base import clone
from sklearn.pipeline import _name_estimators

class MajorityVoteClassifier(BaseEstimator, ClassifierMixin):

    def __init__(self,classifiers,vote='classlabel',weights=[1,1,1]):
        self.classifiers = classifiers
        self.named_classifiers = {key: value for key, value in _name_estimators(classifiers)}
        self.vote = vote
        self.weights = weights

    def fit(self, X, y):
        self.lablenc_ = LabelEncoder()
        self.lablenc_.fit(y)
        self.classes_ = self.lablenc_.classes_
        self.classifiers_ = []
        for clf in self.classifiers:
                fitted_clf = clone(clf).fit(X,self.lablenc_.transform(y))
                self.classifiers_.append(fitted_clf)
        return self

    def predict(self, X) :
        if self.vote == 'probability':
                maj_vote = np.argmax(self.predict_proba(X), axis=1)
        else:
                predictions = np.asarray([clf.predict(X) for clf in self.classifiers_]).T
                maj_vote = np.apply_along_axis(lambda x: np.argmax(np.bincount(x,weights=self.weights)),axis=1,arr=predictions)
        maj_vote = self.lablenc_.inverse_transform(maj_vote)
        return maj_vote

    def predict_proba(self, X):
        probas = np.asarray([clf.predict_proba(X) for clf in self.classifiers_])
        avg_proba = np.average(probas,axis=0, weights=self.weights)
        return avg_proba

    def get_params(self, deep=True):

        if not deep:
                return super(MajorityVoteClassifier,self).get_params(deep=False)
        else:
                out = self.named_classifiers.copy()
                for name, step in\
                        six.iteritems(self.named_classifiers):
                                for key, value in six.iteritems(step.get_params(deep=True)):out['%s__%s' % (name, key)] = value

```

EK 3 – ROC eğrilerinin çizilmesini sağlayan Python kodu

```
for i in range(n_classes):
    fpr[i], tpr[i] = roc_curve(Y_test[:, i], prediction[:, i])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)

for i in range(n_classes):
    mean_tpr += interp(all_fpr, fpr[i], tpr[i])

fpr["macro"] = all_fpr
tpr["macro"] = mean_tpr
roc_auc["macro"] = auc(fpr["macro"], tpr["macro"])

plt.figure()

plt.plot(fpr["macro"], tpr["macro"], label='({0:0.2f})'
                                           ''.format(roc_auc["macro"])))

plt.plot([0, 1], [0, 1], 'k--', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Yanlış Pozitif Oranı')
plt.ylabel('Doğru Pozitif Oranı')
plt.legend(loc="lower right")
plt.show()
```

EK 4 – "IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) Workshops - W4: Workshop on Machine Learning for Security and Cryptography" Bildirisi

AndMFC: Android Malware Family Classification Framework

Sercan Türker
Department of Computer Engineering
University of Hacettepe
Ankara, Turkey
n14323115@cs.hacettepe.edu.tr

Ahmet Burak Can
Department of Computer Engineering
University of Hacettepe
Ankara, Turkey
abc@cs.hacettepe.edu.tr

***Abstract*—As the popularity of Android mobile operating system grows, the number of malicious software have increased extensively. Therefore, many research efforts have been done on Android malware analysis. Besides detection of malicious Android applications, recognizing families of malwares is also an important task in malware analysis. In this paper, we propose a machine learning-based classification framework that classifies Android malware samples into their families. The framework extracts requested permissions and API calls from Android malware samples and uses them as features to train a large set of machine learning classifiers. To validate the performance of our proposed approach, we use three different malware datasets. Our experimental results show that all of the tested models classify malwares efficiently. We also make a study of detecting unknown malwares that never seen before and we notice that our framework detects these malwares with a high accuracy.**

Keywords*— *Android Malware, Malware Classification, Family Classification, Static Analysis, Machine Learning

I. Introduction

Smart phones have become an important part of our lives in the last decade. Among the mobile operating systems on smart phones, Android has become the most popular one and has dominated the market with a high sales rate. In the second quarter of 2018, 88 percent of all smart phones sold to end users are working on the Android operating system [1]. Being the most popular mobile operating system, Android has been more targeted than any other mobile operating systems by attackers [2]. Therefore, Android malware analysis has become one of the most interesting topics for researchers. Although most of the studies are done to classify Android applications as either malicious or benign, classifying malwares into their families is also an important research topic. Since malwares of the same family are expected to exhibit the similar behavior, family classification can be useful in deciding whether the malware is a sort of already known malware. This decision eases malware analysts' work by enabling to focus solely on new malwares that does not belong to any known family, rather than wasting time on analyzing known malware types.

Android malware analysis can be grouped into two main sections: static analysis and dynamic analysis. Static analysis is based on analyzing of application's manifest and code files. Thus, the package file of the application and a tool for parsing this file are enough for the analysis. On the other hand, in dynamic analysis, application's behavior is monitored during its execution in a real or emulated environment. Each method has its own advantages and disadvantages. Implementing static analysis methods is easier than the dynamic approach but static analysis can be prevented with obfuscation techniques. Although dynamic analysis is more robust for obfuscation, it can be defeated by some evasion techniques.

In this paper, we propose an Android malware classification framework (AndMFC) that can recognize families of Android malwares. The framework performs static analysis on malwares and extracts requested permissions and API calls as features. After selecting the most distinguishing features, a set of machine learning classification algorithms, including ensemble and neural network algorithms, are used to classify malwares. In the experiments, Support Vector Machine, Decision Tree, Logistic Regression, K-Nearest Neighbors, Random Forest, AdaBoost, and Multi-Layer Perceptron classifiers are used for classification. To extensively evaluate the performance of these classifiers, three different datasets are used. The results of evaluations show that the extracted features from static analysis are useful to identify malware families and most of the tested classification methods can accurately classify Android malware into the families. In addition to classification, we also examine characteristics of malwares by leveraging the feature set and report the specific behaviors of malware categories. As the final work, we propose an approach for detecting the malwares that does not belong to any known family and classified them as unknown. Overall, the main contributions of this paper are:

- We present an Android malware family classification framework that performs static analysis and open the code for future studies of other researchers[1].
- The framework uses static analysis to classify malware families. The proposed method provides resilience to renaming attacks and partial resilience to dynamic loading attacks, which are used to evade static analysis.
- The proposed framework can use a large set of machine learning classification methods including ensemble and neural network algorithms. It can be expendable with future classification methods.
- We measure the performance of each tested classifier in terms of accuracy, precision, recall, and F1 score values. We report the classification results on AMD, Drebin and UpDroid datasets. It is also shown that the feature set obtained in AMD dataset training is helpful to classify malwares on other datasets.
- We show how the framework can be used on detecting unknown malwares and report detection performance in terms of accuracy, precision, recall and F1 score.
- Examining the frequency of use, we have addressed some of the important features that are useful to characterize malware categories.

[1] The code for the AndMFC framework can be found at : https://github.com/srentrkr/AndroidFamilyClassification

The rest of the paper is organized as follows. In Section II, the prior researches that have been made to identify and classify Android malware families are reviewed. Section III explains the proposed model in details. The evaluation results are presented in Section IV and finally, we conclude the work in section V.

II. RELATED WORK

Several machine-learning based approaches have been presented in the literature to detect Android malware automatically with the features extracted from static and/or dynamic analysis. Some works like Drebin [8], DroidAPIMiner [9], AppContext [10] and Anastasia [11] perform static analysis for malware detection, whereas Stream [12], MADAM [13], DroidDolphin [14], Caviglione et al [15] and Nancy et al [16] use dynamic features extracted from dynamic analysis. Marvin [17] and F. Yang et al [18] used both static and dynamic features to detect malware. Deep learning based malware detection is also interested in some works like HADM [19], DroidSec [20] and DroidDeep [21].

In addition to malware detection, there are some studies in malware family classification. DroidLegacy [22] system extracts familial signatures from API calls to classify malware into families. RevealDroid [23] performs static analysis that is resilient against basic obfuscation techniques and classifies malware with C4.5 and 1NN classifiers. Their approach observes information flow and sensitive API flow during static analysis. DroidSieve [24] also considers only static and obfuscation resilient features for classification. DroidScribe [25] uses a dynamic approach for malware family classification and extracts features from runtime behavior such as system calls and network transactions. Masseralli et al [26] also proposes a dynamic approach that relies on resource consumption. Dendroid [27] proposes a text-mining method to automatically classify malware samples into families based on the similarity of their code structures. LEILA [28] analyses Java Bytecode that is produced when the source code is compiled and identifies malicious behavior of each malware family. EC2 [29] proposes an algorithm that effectively classifies a malware sample into both large and small families. The system leverages both static and dynamic features and is designed as an ensemble that combines the best of classification and clustering. UpDroid [32] is another work that performs hybrid analysis and classifies the malwares of their own dataset.

III. DESIGN AND IMPLEMENTATION

In this section, we describe design and implementation of our framework. The goal of our system is classifying Android malware applications into malware families. Our proposed framework performs static analysis of malwares and extracts a huge number of features from package contents, where some of the informative features represent characterization of malwares. Using some of the selected features, classification of malwares is performed with a set of classifiers. The components of the AndMFC framework are illustrated in Figure 1. One of the important points of the AndMFC is that any feature extraction / classification method can be plugged into the system to make further analysis.

A. Feature Extraction

Our framework performs a static analysis and extracts a comprehensive set of features to characterize a malware accurately. To extract these features, a reverse engineering tool called Apktool v2.3.4 [4] is leveraged. Apktool decompiles the malwares and generates an output folder that contains all Java classes in Smali [5] language, the manifest file of the application, resources, libraries and assets. We focus on parsing manifest and Smali files to extract the following features:

- **Permissions**: Permission system is an important security mechanism of Android platform that is used to organize the application's access to sensitive user data (e.g., SMS, Calendar) and system features (e.g., Camera, Microphone). This characteristic makes the requested permissions one of the most used static features in Android malware analysis [6]. Extracting permission features is crucial since they are helpful to address the malware family. One example is that malwares belong to *Svpeng* family lock the mobile device. To achieve this malicious action, they request *SYSTEM_ALERT_WINDOW* permission [7]. Therefore, using this permission as a feature is helpful to determine whether a malware belongs to the Svpeng family or not. With this observation, 278 different permission features are extracted.
- **API Calls:** Android API contains a set of functions used by a software to interact with Android operating system. The functions used by the malware developers can include a clue about the malware's family. For example, some malware families such as *BankBot* and *Triada* include codes to evade dynamic analysis. To achieve this, they check whether the system that runs the application is a real or an emulated system by using some API methods like *GetDeviceId*. Thus, using this API call as a feature can help us identifying a malware's family. A total of 28433 different API call features are extracted regardless of whether they are suspicious or innocuous. Most malware detection methods extract only suspicious calls but AndMFC considers all types of API calls to classify different types of malware families.

Some malware families in the dataset use many techniques to evade static analysis. Renaming is one of them and it is an obfuscation technique that gives a random name to variables, methods and classes. When this technique is applied to an application, the Smali files extracted by the Apktool contains meaningless names instead of real names. However, API calls cannot be renamed and they are always visible in Smali files [7]. Thus, our system is resilient to renaming attacks.

Another technique to evade static analysis is dynamic loading that means a malware load the malicious code from another application. Our framework looks for another application in *assets* folder of Apktool output and analyzes them to extract features as the features of the related malware. In this way, dynamically loaded codes can be taken into account and classification performance can be increased. Because of the fact that loading from the *assets* folder is not the only way of dynamic loading, we can say that we provide a partial resilience to this technique.

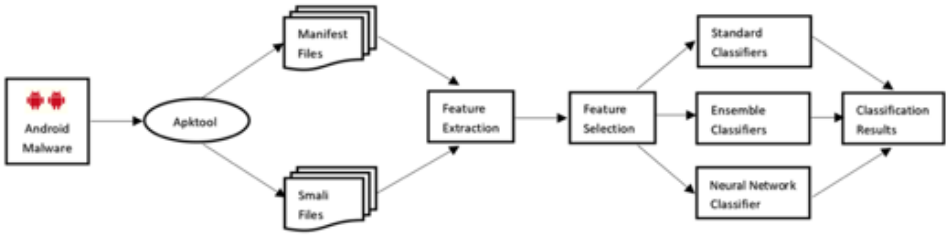


Figure 1: Overview of the AndMFC system

B. Feature Selection

As a preprocessing step, we apply a feature ranking method to determine the features that affect the results most. We use AMD [3] dataset and Random Forest classifier to learn the importance of each feature. After trying various number of features, we observed that using the top 1000 features is the most effective option. (The details are given in [31]). This feature set contains 42 permission features and 958 API call features. Discarding redundant and irrelevant features reduces model complexity and execution time, while increasing the accuracy of classification.

C. Classification Models

We employ several machine learning classifiers to build a promising model for classifying Android malwares into their families. Apart from the standard supervised classifiers such as Support Vector Machine (SVM), Decision Tree (DT), Logistic Regression (LR) and K-Nearest Neighbors (KNN), some ensemble classifiers such as Random Forest (RF) and AdaBoost are used. Since deep learning is a popular approach in machine learning, we also apply Multi-Layer Perceptron (MLP) algorithm in our experiments.

Apart from these classifiers, we combine three standard classifiers (DT, LR and KNN) and create an ensemble Majority Voting (MV) classifier. Each classifier in this ensemble model makes a prediction (vote) and the final prediction is selected as the prediction of the majority of the classifiers.

These classification algorithms have hyper parameters and proper settings of these parameters can affect the classification performance positively. Thus, we perform hyper-parameter optimization on our classifiers through Grid-search and Cross-validation processes implemented in Scikit-learn Machine Learning package [30]. Table I shows the hyper parameters for each classifier.

TABLE I. PARAMETER TUNING VIA GRID-SEARCH AND CROSS-VALIDATION

Classifier	Best Parameters
SVM	kernel='poly', gamma=0.1, C=1.0
DT	max depth=18, criterion='entropy',
LR	penalty='l2', C=0.1,
KNN	n_neighbors=5, p=2, metric='minkowski'
RF	n_estimators=10000
AdaBoost	n_estimators=500, learning_rate=1
MV	Best parameters of DT, LR and KNN
MLP	nEpoch=100, batch_size=100, num_hidden_layers=3, layer_size=[1000,500,250,125,71]

IV. EVALUATION

In this section, we evaluate the overall system and present the classification performance results of each classifier. We use AMD [3] dataset for our experiments and analyze 24467 Android malware samples which belong to 71 different Android malware families and 10 malware categories. We eliminate 86 malware samples because of the execution errors of ApkTool.

After the presentation of the results on AMD dataset, we characterize some malware categories and malware families by examining the frequency of their features. Then, we also validate the performance of our system on Drebin and UpDroid datasets. Finally, we work on classifying *unknown* malware which has not been studied before in our knowledge.

A. Evaluation Metrics

To measure the performance of classifiers, we calculate Accuracy (Acc), Macro Precision (MaP), Macro Recall (MaR) and Macro F1 Score (MaF) values. Accuracy is an overall measure of the classifier and corresponds to the proportion of correct results that the classifier achieved. Precision measures how many of the malware samples associated to a family are indeed belongs to that family, while recall measures how many malware samples are correctly classified. Macro Precision and Macro Recall corresponds to the averages of precision and recall values of families. Finally, Macro F1 Score is the harmonic average of the Macro Precision and Macro Recall values.

B. Classification Results on AMD Dataset

As the first testing method, we perform 10-fold cross validation on the entire dataset to measure performance of the classification models. As the second testing method, we randomly split the whole dataset into equal sized training and test sets (2-fold). Since the dataset is an unbalanced dataset, we pay attention to get samples from each family with equal ratio and we randomly select half of each family as test data. Classification accuracy results obtained from these two experiments are given in Table II.

As shown in Table II, SVM, AdaBoost and MLP gives better classification performance compared to others. 10-fold cross validation accuracies and 2-fold test accuracies of all classifiers are close to each other. All the tested classifiers are able to achieve more than 96% accuracy. This shows that the extracted features are meaningful to describe malware families of the dataset. SVM gives the best performance among the standard classifiers, while AdaBoost is the best

ensemble classifier. MV classifier achieves better performance than individual DT and KNN scores.

TABLE II. CLASSIFICATION ACCURACY SCORES

Classifier	Type	10-fold Accuracy (%)	2-fold Accuracy (%)
SVM	Standard	98.86 (+-0.98)	99.12
DT	Standard	97.12 (+-1.71)	97.27
LR	Standard	98.47 (+-0.96)	98.87
KNN	Standard	96.51 (+-1.44)	96.98
RF	Ensemble	98.18 (+-1.24)	98.42
AdaBoost	Ensemble	98.68 (+-1.16)	99.08
MV	Ensemble	98.31 (+-1.40)	98.63
MLP	Neural Net	98.67 (+-0.96)	99.00

In order to analyze these results further, we make more statistical analysis on the data. Using the confusion matrices produced by the classifiers in the 2-fold evaluation, we calculate MaP, MaR and MaF values and report them in Table III.

TABLE III. MACRO AVERAGES OF PRECISION, RECALL AND F1 SCORES

Classifier	Type	MaP (%)	MaR (%)	MaF
SVM	Standard	95.02	89.06	91.94
DT	Standard	80.80	84.28	82.50
LR	Standard	94.15	87.13	90.50
KNN	Standard	85.48	79.00	82.11
RF	Ensemble	93.75	84.93	89.12
AdaBoost	Ensemble	94.69	88.87	91.68
MV	Ensemble	92.61	85.44	88.88
MLP	Neural Net	89.19	86.58	87.86

As we can observe from the table, averaged precision, recall and F1 Scores are lower than accuracy values. The reason for this situation is that the dataset we used is an unbalanced dataset and there are some malware families that contain just a few malware samples. The tested classifiers generally do not give a good performance on these families and produce low TP rates. This situation decreases the average MaP, MaR, and MaF values. Figure 2 and Figure 3 shows the precision and recall values of each malware family based on the results of SVM classifier. Especially, FakeUpdates, Lnk and Tesbo malware families affect MaP, MaR values negatively with low precision and recall values. For some families, SVM produces high precision values but low recall values, such as Kemoge, MMarketpay, Opfake, Penetho families. This is due to low number of TP and high number of FN values in these classes. This means that SVM cannot identify the samples from these families and assign them to other families. Low number of samples in these families is the main reason of this anomaly. However, there are some malware families with low number of samples but high precision and recall values, such as Univert, Vmvol families. This shows that the extracted features provide enough information to identify these families, which probably have very different structure comparing to other families.

C. Character Identification

After doing classification, we also examine the features which are chosen in the feature selection step and how frequently these features are used by malware families. Based on this study we report on some characteristics of malware families and malware categories:

- Most of the malwares belonging to the families in *Adware* category, such as *Airpush*, *Kuguo* and *Youmi*, call *notify* method of *NotificationManager* class to send notification to the victim.
- The majority of the malwares in *Trojan-Banker* category leverage *SmsManager* class to send SMS to victims. *BankBot* and *SlemBunk* families are examples, which include malwares in this category.
- *SqliteDatabase* class that exposes methods to manage a SQLite database is used by most of the malwares in *Trojan-Dropper* category. However, only few members of *Ransom* and *Trojan-Spy* categories use this class.
- Malwares in *Adware* category generally detect victim's location via using methods of *Location* and *LocationManager* class. Thus, almost all of the samples in *Airpush* family call the methods of *Address* class such as *getCountryCode*, *getCountryName* and *getPostalCode*.
- Most of the malwares belonging to the *Backdoor* and *Trojan* category use the methods of *TelephonyManager* class such as *getDeviceId*, *getSubscriberId*, *getLine1Number*, and *getNetworkType* to access information about the victim's phone.
- There are many malwares in Ransom and HackerTool category whose manifest files do not include Internet permission. Particularly, Jisut family consists of mostly offline applications.
- The majority of the malwares in Ransom category request read_call_log, read_contacts and write_contacts permissions.

Although these observed behaviors are helpful for analyzing malwares, these are not certain behavior of the malware families. As the malwares evolve and new malwares are produced, these kind of behaviors will obviously change.

D. Classification Results On Drebin and UpDroid Datasets

To validate the classification performance of our framework, we also use Drebin and UpDroid datasets. We analyze 5507 Drebin malwares belonging to 132 different families and 2408 UpDroid malwares belonging to 20 different families. Training and test sets are created as in the 2-fold evaluation method of AMD dataset. Additionally, we use the same feature set and same hyper-parameters which are given in Section III. Table IV shows the evaluation results of each classifiers on two datasets. Due to space limit, we just give accuracy results. MaP, MaR and MaF values can be seen in our thesis study [31].

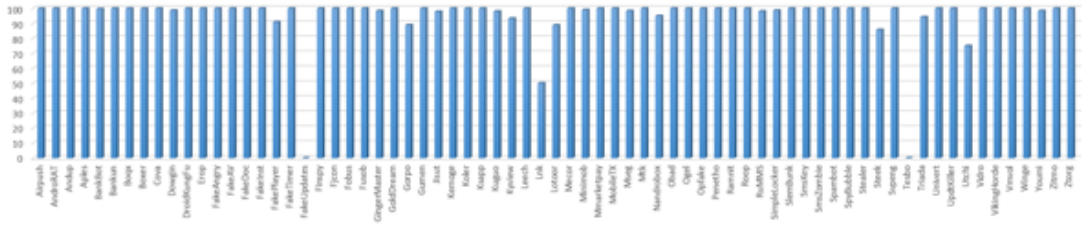


Figure 2: Precision values of SVM classifier per malware family

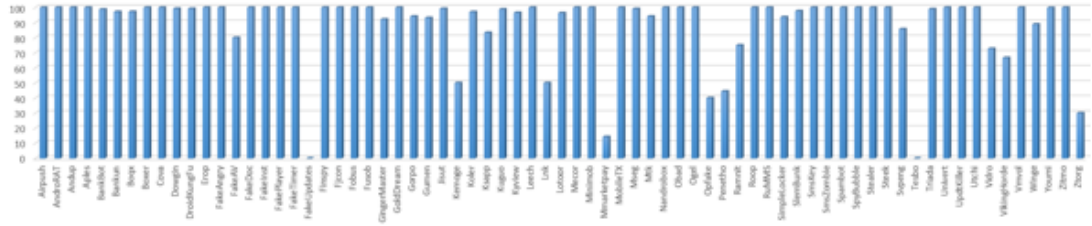


Figure 3: Recall values of SVM classifier per malware family

TABLE IV. CLASSIFICATION RESULTS ON DREBIN AND UPDROID DATASETS

Classifier	Type	Drebin - Acc (%)	UpDroid – Acc (%)
SVM	Standard	96.66	94.66
DT	Standard	92.93	90.83
LR	Standard	96.51	93.24
KNN	Standard	91.59	91.99
RF	Ensemble	96.37	94.25
AdaBoost	Ensemble	96.79	93.74
MV	Ensemble	94.84	92.16
MLP	Neural Net	96.10	93.66

Results show that all classifiers classify malwares with a high accuracy. SVM and AdaBoost classifiers give better accuracy than others as in the previous evaluation. The reason why the accuracy values are lower than the results of previous evaluation is that these datasets are smaller than AMD dataset and low number of samples lead to lower values because of the lack of training data. The results also show that the feature set obtained in AMD dataset training is useful to classify malware in another dataset.

We compare these accuracy results with the results of DroidSieve and UpDroid works. DroidSieve and UpDroid reach 97,68% and 96.85% accuracy values respectively in classification of Drebin malwares into their families. While these studies eliminate the most of the malware families which have low number of samples, we eliminate only the families which have just one sample. Thus, it is acceptable that our method achieve lower accuracy values because of the more unbalanced dataset. UpDroid also classifies the malwares in UpDroid dataset with an accuracy of 96.2%. Our best accuracy result on this dataset is lower than this value. The reason of this difference can be the impact of dynamic analysis performed by UpDroid.

E. Prediction on Unknown Malwares

In previous experiments, our target was classifying the malware that belongs to a known malware family. Although prediction of a malware family is an important task, detecting a new malware that doesn't belong any known family is also important in the context of malware analysis. Thus, we test our model to predict unknown malwares as a final step. We leverage the probability values generated by the classifier during the classification.

As the first step, we identify the families of AMD and Drebin datasets which contain less than 25 samples and separate them as unknown malware samples. Then, we randomly select 80% of the other malwares as training set and the remaining as test set. We use SVM classifier which gives one of the best performances in the previous evaluations. After we train the classifier with training sets and test it with test sets, we note the probability of each prediction. We define a confidence value using these probabilities as shown below:

$$\text{Confidence} = \mu_p - \sigma_p$$

μ_p and σ_p denote the average and standard deviation of probability values generated during classification of all malware samples in the datasets. Using this confidence value, we try to predict an unknown malware as unknown. If the prediction probability of a given sample is lower than this value, the sample is classified as unknown. According to this model, we add the unknown samples to test sets and repeat the classification process with this dataset. Table V shows the confidence values and the classification results for each dataset. The results show that our approach is capable of detecting unknown malwares with a high accuracy. More detailed results and the impact of different confidence values to these results are given in [31]. Lower confidence values results in higher accuracy and precision values but lower recall values. Higher confidence values have reverse impact.

TABLE V. CLASSIFICATION RESULTS ON UNKNOWN MALWARES

Dataset	Confidence Value	Acc (%)	Precision (%)	Recall (%)	F1-Score (%)
AMD	91.7	93.63	60.54	87.19	71.46
Drebin	85.14	86.54	86.93	84.02	85.45

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework to classify Android malwares into their families. The proposed approach

leverages static features and classify malwares with several machine learning classifiers. Overall, the tested machine learning models produce high accuracy in classifying malware families, which shows the saliency of the extracted features. The experimental results show that SVM classifier leads to a better accuracy among all classifiers. Since the framework is developed in a modular structure, feature extraction/selection, and machine learning techniques can be replaced with future techniques. An important conclusion is that the feature set obtained in AMD dataset is useful for classifying malwares in Drebin and UpDroid datasets. This shows the effectiveness of the proposed method in malware family classification. Furthermore, the framework can also be used to identify unknown malware samples by using probabilities generated by the classification algorithm.

As the software advances, new kind of static and dynamic analysis techniques will be needed. In the future work, the set of features can be expanded with new static and dynamic features. A future direction can be investigating unbalanced dataset problems. In most malware datasets, some of the malware families have insufficient number of samples, which makes classification of these families harder. Creating better classification techniques for unbalanced dataset would be a good contribution to the field. Additionally, a historical malware dataset can be created for analyzing evolution of malwares with time. Structures and behaviors of malwares are continuously changing. Thus, analyzing these changes would be helpful to understand future malwares.

REFERENCES

[1] Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018, Available at: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/, (Accessed: Dec 16, 2018)

[2] More than 99 percent of all malware designed for mobile devices targets Android devices, explained Olaf Pursche, Head of Communications at AV-TEST, in the F-Secure State of Cyber Security 2017.

[3] AMD dataset, Available at: http://amd.arguslab.org/, (Accessed: Dec 16, 2018)

[4] Apktool, Available at: https://github.com/iBotPeaches/Apktool/, (Accessed: Dec 16, 2018)

[5] Smali, Available at: https://github.com/JesusFreke/smali, (Accessed: Dec 16, 2018)

[6] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," Digital Investigation, vol. 13, no. 0, pp. 22 – 37, 2015.

[7] Wei, Fengguo & Li, Yuping & Roy, Sankardas & Ou, Xinming & Zhou, Wu. (2017). Deep Ground Truth Analysis of Current Android Malware. 252-276. 10.1007/978-3-319-60876-1_12.

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS, San Diego,CA,USA, 2014.

[9] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-level features for robust malware detection in android. In SecureComm, pages 86–103, Sydney,AU, 2013. Springer.

[10] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. AppContext: Differentiating malicious and benign mobile app behaviors using context. In Proc. ICSE, pages 303–313, 2015.

[11] Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. Anastasia: Android malware detection using static analysis of applications. In New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on, pages 1–5. IEEE, 2016.

[12] B. Amos, H. A. Turner, and J. White, "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale," in International Conference on Wireless Communications and Mobile Computing (IWCMC), 2013.

[13] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra. MADAM: A multi-level anomaly detector for android malware. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7531 LNCS:240–253, 2012.

[14] W.-C. Wu and S.-H. Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS '14, pages 247–252, New York, NY, USA, 2014. ACM.

[15] Caviglione, L., Gaggero, M., Lalande, J., Mazurczyk, W., Urbanski, M.: Seeing the Unseen: Revealing Mobile Malware Hidden Communications Via Energy Consumption and Artificial Intelligence. IEEE Trans. Inform. Forensic Secur. 11, 799-810 (2016)

[16] Nancy, Dr. Deepak Kumar Sharma, "Android Malware Detection using Decision Trees and Network Traffic", in IJCSIT, Vol. 7 (4), 2016.

[17] M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In IEEE COMPSAC, 2015.

[18] F. Yang, Y. Zhuang, J. Wang, "Android Malware Detection Using Hybrid Analysis and Machine Learning Technique", from book Cloud Computing and Security: Third International Conference, ICCCS 2017, Nanjing, China, June 16-18, 2017, Revised Selected Papers, Part II (pp.565-575)

[19] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "Hadm: Hybrid analysis for detection of malware."

[20] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, Droid-sec: Deep learning in Android malware detection, in Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM, poster), 2014, pp. 371–372.

[21] Xin Su et al. 2016. A Deep Learning Approach to Android Malware Feature Learning and Detection. In IEEE TrustCom. 244–251.

[22] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated familial classification of Android malware," in ACM SIGPLAN Program Protection and Reverse Engineering Workshop, PPREW, 2014.

[23] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S. Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of android malware," Department of Computer Science, George Mason University, Tech. Rep., 2015.

[24] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and accurate classification of obfuscated android malware," in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. ACM, 2017, pp. 309–320.

[25] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in Security and Privacy Workshops (SPW), 2016 IEEE. IEEE, 2016, pp. 252–261.

[26] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, R.Baldoni, "Android Malware Family Classification Based on Resource Consumption over Time".

[27] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," Expert Systems with Applications, vol. 41, no. 1, pp. 1104–1117, 2014.

[28] G. Canfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone and C. A. Visaggio, "LEILA: formaL tool for idEntifying mobIle maLicious behAviour," in IEEE Transactions on Software Engineering.

[29] T. Chakraborty, F. Pierazzi and V. S. Subrahmanian, "EC2: Ensemble Clustering and Classification for Predicting Android Malware Families," in IEEE Transactions on Dependable and Secure Computing.

[30] Scikit-learn, Available at: https://github.com/scikit-learn/, (Accessed: Dec 16, 2018

[31] Sercan Türker, Android Malware Family Classification with Machine Learning, MSc thesis, Hacettepe University, 2019

[32] Aktas, Kursat & Sen, Sevil. (2018). UpDroid: Updated Android Malware and Its Familial Classification.

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS/~~DOKTORA~~ TEZ ÇALIŞMASI ORJİNALLİK RAPORU

HACETTEPE ÜNİVERSİTESİ
FEN BİLİMLER ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI BAŞKANLIĞI'NA

Tarih: 26/06/2019

Tez Başlığı / Konusu: Zararlı Android Yazılımlarının Makine Öğrenmesi ile Ailelerine Göre Sınıflandırılması

Yukarıda başlığı/konusu gösterilen tez çalışmamın a) Kapak sayfası, b) Giriş, c) Ana bölümler d) Sonuç kısımlarından oluşan toplam 56 sayfalık kısmına ilişkin, 26/06/2019 tarihinde ~~şahsım~~/tez danışmanım tarafından *Turnitin* adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı % 6 'tür.

Uygulanan filtrelemeler:
1- Kaynakça hariç
2- Alıntılar ~~hariç~~/dâhil
3- 5 kelimeden daha az örtüşme içeren metin kısımları hariç

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü Tez Çalışması Orjinallik Raporu Alınması ve Kullanılması Uygulama Esasları'nı inceledim ve bu Uygulama Esasları'nda belirtilen azami benzerlik oranlarına göre tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Gereğini saygılarımla arz ederim.

Tarih ve İmza
26.06.2019

Adı Soyadı:	Sercan TÜRKER
Öğrenci No:	N14323115
Anabilim Dalı:	Bilgisayar Mühendisliği
Programı:	Bilgisayar Mühendisliği – Yüksek Lisans
Statüsü:	☒ Y.Lisans ☐ Doktora ☐ Bütünleşik Dr.

DANIŞMAN ONAYI

UYGUNDUR.

Doç.Dr. Ahmet Burak CAN
(Unvan, Ad Soyad, İmza)

ÖZGEÇMİŞ

Adı Soyadı	:	Sercan Türker
Doğum yeri	:	Ankara
Doğum tarihi	:	30.06.1990
Medeni hali	:	Bekar
Yazışma adresi	:	Güneşevler Mah. 125. Sokak 9/11 Altındağ/ANKARA
Telefon	:	05378420692
Elektronik posta adresi	:	sercan.turker90@gmail.com
Yabancı dili	:	İngilizce

EĞİTİM DURUMU

Lisans : Hacettepe Üniversitesi, Bilgisayar Mühendisliği

İş Tecrübesi

2013-2015 İnnova Bilişim Çözümleri

2015-Devam Tübitak Bilgem İltaren

Tezden Üretilmiş Yayınlar

1. Turker, S., Can, A.B., AndMFC: Android Malware Family Classification Framework, IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC) Workshops - W4: Workshop on Machine Learning for Security and Cryptography, Istanbul, Turkey, 2019