ZARARLI YAZILIMLARIN TESPİTİ İÇİN HİBRİT SİSTEM TASARIMI

Kerim Can KALIPCIOĞLU

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ZARARLI YAZILIMLARIN TESPİTİ İÇİN HİBRİT SİSTEM TASARIMI

Kerim Can KALIPCIOĞLU
0000-0003-4885-346X

Dr. Öğr. Üyesi Cengiz TOĞAY
(Danışman)

Dr. Öğr. Üyesi Esra N. YOLAÇAN
(İkinci Danışman)
Eskişehir Osmangazi Üniversitesi

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BURSA – 2020

TEZ ONAYI

Kerim Can KALIPCIOĞLU tarafından hazırlanan "ZARARLI YAZILIMLARIN TESPİTİ İÇİN HİBRİT SİSTEM TASARIMI" adlı tez çalışması aşağıdaki jüri tarafından oy birliği ile Bursa Uludağ Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman : Dr. Öğr. Üyesi Cengiz TOĞAY
İkinci Danışman : Dr. Öğr. Üyesi Esra N. YOLAÇAN

Başkan : Dr. Öğr. Üyesi Cengiz TOĞAY
0000-0001-5739-1784
Bursa Uludağ Üniversitesi,
Mühendislik Fakültesi,
Bilgisayar Mühendisliği Anabilim Dalı

İmza

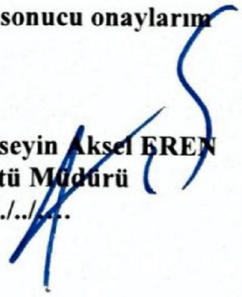
Üye : Doç. Dr. Cemal HANILÇI
0000-0002-9174-0367
Bursa Teknik Üniversitesi,
Mühendislik ve Doğa Bilimleri Fakültesi,
Elektrik Elektronik Mühendisliği Anabilim Dalı

İmza

Üye : Doç. Dr. Gıyasettin ÖZCAN
0000-0002-1166-5919
Bursa Uludağ Üniversitesi,
Mühendislik Fakültesi,
Bilgisayar Mühendisliği Anabilim Dalı

İmza

Yukarıdaki sonucu onaylarım



Prof. Dr. Hüseyin Aksel EREN
Enstitü Müdürü
../../....

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 tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

…/…/……

Kerim Can KALIPCIOĞLU

ÖZET

Yüksek Lisans Tezi

ZARARLI YAZILIMLARIN TESPİTİ İÇİN HİBRİT SİSTEM TASARIMI

Kerim Can KALIPCIOĞLU

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Cengiz TOĞAY
İkinci Danışman: Dr. Öğr. Üyesi Esra N. YOLAÇAN

Zararlı yazılımlar uzun süredir bilgisayar güvenliği için tehlike oluşturmaktadır. Bunun yanında zararlı yazılımlar devlet kurumları ve ticari kuruluşlara saldırılardan geniş çaplı kripto-fidye saldırılarına kadar birçok amaçla kullanılmaya başlanmıştır. Günümüzde yaygın bir şekilde kullanılmakta olan imza tabanlı yaklaşımlar, özellikle sıfır gün saldırıları gibi henüz tespit edilmemiş saldırı vektörlerine karşı başarısız olmaktadırlar. Kritik noktalardaki bilgisayar sistemlerinin gerek güncelleme ve gerekse yeni uygulamaların kurulmasının ardından sıfır gün saldırıları ile karşılaşma riski bulunmaktadır. Bu tip saldırılar genellikle en az bir sisteme zarar verdikten sonra tespit edilmektedir. Dolayısı ile bu süre zarfında kullanıcılar bu tip saldırılara karşı savunmasız kalırlar. Gerek statik ve gerekse dinamik analiz zararlı yazılım analiz sürecini kısaltması ve sıfır gün saldırılarına karşı umut vermesi nedeni ile makine öğrenmesi yaklaşımlarından yaygın olarak yararlanılmaktadır. Makine öğrenmesi modellerinden beklenen, uygulamada kullanılan ticari ürünler kadar kararlı ve hızlı olması, aynı zamanda da zararlı örüntüleri insanlar kadar iyi tanımasıdır. Bu alanda yapılan akademik çalışmalar örnek veriler üzerinde başarılı ölçümler gösterse de ticari ürünler halen imza tanıma temelli yaklaşımları tercih etmektedir. Bu tez çalışmasında taşınabilir, ölçeklenebilir ve yorumlanabilir bir makine öğrenme modeli oluşturması amaçlanmıştır. Tez kapsamında yapılan çalışmada çalıştırılabilir dosyalara ait basit özellikler kullanılarak modelin yorumlanabilir ve taşınabilir olması sağlanmıştır. Yapılan deneylerde oluşturulan topluluk modeli, bu statik özellikler ile temsil edilen çok sınıflı zararlı yazılım kümesini tekil modellere göre daha başarılı şekilde sınıflandırdığı gözlemlenmiştir. Bu şekilde iki farklı özellik kümesinden ve beş farklı makine öğrenmesi yönteminden oluşturulan hibrit topluluk modelinin zararlı yazılımların sınıflandırılması için kullanılabileceği gösterilmiştir. Çalışmada kullanılan dosya tipinden bağımsız yöntem ile bir hibrit zararlı yazılım tespit yöntemi geliştirilmiştir. Bu şekilde farklı özellik kümelerinden ve makine öğrenmesi yöntemlerinden oluşturulan hibrit topluluk modelinin zararlı yazılımları %98 üzerinde doğrulukla sınıflandırabildiği gösterilmiştir.

Anahtar Kelimeler: Zararlı yazılım, statik analiz, makine öğrenmesi, topluluk öğrenmesi, gözetimli öğrenme, yapay zekâ

2020, vi + 63 sayfa.

ABSTRACT

MSc Thesis

HYBRID SYSTEM DESIGN FOR MALICIOUS SOFTWARE DETECTION

Kerim Can KALIPCIOĞLU

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Asst. Prof. Dr. Cengiz TOĞAY
Second Supervisor: Asst. Prof. Dr. Esra N. YOLAÇAN

Malicious software is a known threat in computer security for a long time. However, in recent years, malicious software was started to use with different purposes, e.g., attacks for high-class governmental & commercial organizations and large scale crypto-ransom attacks. Widely used signature-based methods are mostly ineffective against attack vectors like zero-day attacks. Updated or newly installed computer systems, including critical infrastructure, also face zero-day attacks. Most of the time, this type of attack spotted after at least one incident. Computer systems will be vulnerable until the incident detected. Both static and dynamic analyses benefit machine learning methods for shorten analysis processes and prevent zero-day attacks. Machine learning is both expected to be robust and fast as commercial security products, also recognize malicious patterns like humans. Although most research done on this topic shows promising results, most commercial products still use signature-based methods. The purpose of this thesis is to develop a portable, scalable, and interpretable machine learning model. For ensuring an interpretable and portable model, basic features extracted from executable files were used as inputs. An ensemble model developed according to experimental results. The developed model was observed to be more successful than individual models on multi-class malware dataset represented as static feature vectors. In this way, a model which made using two different feature sets and five different classifiers presented as a hybrid ensemble model. Also, this hybrid model can be used for different executable file types without any change. As a result of this work, the developed hybrid machine learning model showed higher than %98 accuracy on the multi-class malware dataset.

Key words: Malicious software, static analysis, machine learning, ensemble learning, supervised learning, artificial learning

2020, vii + 63 pages.

TEŞEKKÜR

Başta annem olmak üzere aileme, tez hocalarım Cengiz TOĞAY ve Esra YOLAÇAN'a çalışmalarımda yardımcı olmuş ve kolaylık sağlamış olan akademisyen ve öğrenci arkadaşlarıma, amirlerime, çalışmalarımda yararlandığım bilimsel çalışmalarla bana yol gösteren bilim insanlarına teşekkürü borç bilirim. Bu tez çalışmamı canını hiçe sayarak vatan için gazi ve şehit olmuş kahramanlara armağan ederim.

Kerim Can KALIPCIOĞLU
…/…/…….

İÇİNDEKİLER

Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

Kısaltmalar	Açıklama
API	Application programming interface
APT	Advanced persistent threat
ASCII	American standard code for information interchange
AV	Anti-virüs
BoW	Bag of words
CART	Classification and regression tree
CFG	Code flow graph
CNN	Convolutional neural network
CPU	Central processing unit
ELF	Executable and linkable format
PCA	Principal component analysis
PE	Portable executable
IDS	Intrusion detection system
IPS	Intrusion prevention system
INRIA	Institut national de recherche en informatique et en automatique
KNN	K-nearest neighbours
LSTM	Long short-term memory
ReLU	Rectified linear unit

Çeviriler	Açıklama
Accuracy	Doğruluk
Bias	Önyargı
Boot sector	Önyükleme sektörü
Covert channel	Gizli kanal
Cross-validation	Çapraz doğrulama
Debugger	Hata ayıklayıcı
Decision tree	Karar ağacı
Ensemble learning	Topluluk öğrenmesi
Firmware	Aygıt yazılımı
F-score	F-skoru
Gini impurity	Gini kirlilik değeri
Gradient boosting	Gradyan artırma
Halting problem	Durma problemi
Information theory	Bilgi teorisi
Nearest neighbours	En yakın komşuluk
Overfitting	Ezberlemek
Precision	Hassasiyet
Random forest	Rastgele orman
Ransonware	Fidye yazılımı
Recall	Geri çağırma
Sandbox	Kum havuzu
Section	Bölüm
Seed	Tohum

Self-replicating	Kendini kopyalayan
Side channel	Yan kanal
Spyware	Casus yazılımı
Symbolic execution	Sembolik çalıştırma
Trojan	Truva atı
Worm	Solucan

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

1. GİRİŞ

1.1. Genel Bakış

Bilgisayarlarda işlevselliği sağlayan yazılımlarla birlikte, güvenlik yazılımlarına olan ihtiyaç da artmaktadır. Saldırganlar ile güvenlik yazılımları arasında galibi sürekli değişen ancak saldırganların avantajlı olduğu bir oyun oynanmaktadır. Özellikle korsan yazılımın yaygın olarak kullanıldığı ortamlarda yayılma şansı yakalayan zararlı yazılımlar, bulaştığı bilgisayara ve onun vasıtasıyla başka hedeflere saldırılar düzenlenmesinde kullanılabilmektedir. Symantec firmasının yayınladığı rapora göre, 2018 yılında 246 milyon yeni zararlı yazılım varyantı belirlenmiştir (Symantec 2019). Belirlenen zararlı yazılım varyantı sayısı bir önceki yılan göre %63,3 oranında azalmış olmasına rağmen, saldırı grupları arasında hedefe yönelik yıkıcı zararlı yazılım kullanımı artmıştır (Symantec 2019). Bu gelişmeler bilinen zararlı yazılımların AV (anti-virüs) yazılımları tarafından tespit edilebilmesi dolayısıyla belirli hedeflere yönelik zararlı yazılımların geliştirilmeye başlandığı şeklinde yorumlanabilir. Zararlı yazılımların kullanım amacındaki bu paradigma değişimi tek başına kurumların zararlı tehditlere karşı alması gereken önlemlerin önemini göstermektedir.

Zararlı yazılım, genel bir tanıma göre bilgisayar sistemlerini kötüye kullanan yazılımlar olarak tanımlanır (Nash 2005). Ancak kötüye kullanım kesin bir tanım olmadığından, zararlı yazılımlar türlerine göre sınıflandırılarak incelenmelidir. Zararlı yazılım türleri aşağıdaki şekilde sınıflandırılabilir:

- Bilgisayar virüsleri: kendini kopyalayan otomatlarla modellenebilen yazılımlardır. Bu tip zararlılar çoğunlukla kendilerini başka dosyaların içine kopyalayarak çoğalırlar.
- Solucanlar: bilgisayar virüslerine benzer şekilde kendini kopyalayan yazılımlardır. Ancak, genelde tek bir dosyada bulunurlar ve farklı tipte zafiyetleri kullanarak diğer bilgisayarlara bulaşırlar. Bilgisayar virüsleri ile kıyaslandığında kullanıcı etkileşimine ihtiyaç duymamaları önemli bir özellikleridir.
- Rootkitler: yüksek yetkilerle (çekirdek modülü, aygıt sürücüsü veya önyükleme sektörü yazılımı olarak) çalışacak şekilde tasarlanırlar. Rootkitlerin kelime kökeni Unix-benzeri sistemlerdeki yetkili kullanıcı olan “root” sözcüğünden

gelmektedir. Programların işletim sistemi, aygıt yazılımı veya kullanıcı programlarıyla etkileşiminde araya girerek yanlış bilgi verirler. Bu şekilde kendilerini AV yazılımlarından korurlar.

- Truva atları: kullanıcıya zararsız bir yazılım olarak görünerek yayılan, sistemde arka kapı açılması yolu ile sistemi kullanıcı yetkileri ile saldırgana açan yazılımlardır.
- Casus yazılımlar: kullanıcı ile ilgili bilgiler toplayan yazılımlardır.
- Fidye yazılımları: son zamanlarda yaygınlığı artmış olan bir zararlı yazılım tipidir. Bulaştığı bilgisayarlardaki dosyaları şifreleyerek karşılığında kullanıcılardan fidye talep edilmesinde kullanılan yazılımlardır.
- Botnet yazılımları: ağa bağlı bilgisayarları birçok diğer cihazın olduğu botnet ağına katarak spam gönderimi ve servis engelleme saldırısı gibi amaçlar doğrultusunda kullanır.

Bunun yanında bir sınıf olarak ele alınmayan, günümüzde özellikle devlet kurumların ve ticari şirketlerin bilgisayar sistemlerine karşı en büyük tehdit olan APT (advanced persistent threat – gelişmiş sürekli tehdit) yazılımları AV geliştiricilerinin üzerine en çok çalışma yaptığı konulardandır. Aslında APT, bir ya da daha fazla sayıda zararlı yazılımı saldırı vektörü olarak kullanarak hedeflediği amacı gerçekleştirmeye çalışan organize bir saldırıdır. APT çoğunlukla devletler ve ticari organizasyonlar tarafından desteklenmektedir. Bunun yanında sıfır gün olarak adlandırılan ve yazılımın geliştiricisi tarafından henüz haberdar olunmayan zafiyetler gibi kaynaklara erişim sağlamakta ve hızlı bir şekilde bilgisayar sistemlerine zarar verme veya istihbarat toplama gibi faaliyetleri gerçekleştirmektedir. Bu zararlı yazılımlarda sıfır gün zafiyetlerinin kullanılması hem zararlının bulaşıcılığını artırmakta hem de fark edilmesini zorlaştırmaktadır. Çoğu APT kullandığı zararlı yazılımlarda paketleme gibi yöntemleri kullandığından bu zararlıları yakalayabilmek için akıllı AV yazılımlarına ihtiyaç duyulmaktadır. Bunun yanında yapılan araştırmalarda APT saldırılarında güvenlik ürünlerine özel atlatma teknikleri kullanıldığı saptanmıştır (2019). Bu nedenle AV ürünlerinin sadece imza-tabanlı yöntemleri kullanmasının yeterli olmadığı düşünülmektedir. Bir diğer yandan yazılımlarının büyük çoğunluğu halen kendini değiştiren kodlar gibi gelişmiş yöntemleri kullanmamışlardır (Raiu 2018). Bu

yöntemlerin kullanılmadığı saldırıların dahi toplumda ve ticari firmalarda bu kadar çekinceye neden olduğu düşünüldüğünde, APT yazılımlarının bu yöntemleri benimsemesi durumunda bilgisayar sistemleri üzerindeki zararının çok daha fazla olacağı tahmin edilebilir.

Zararlı yazılımların tespitinde kullanılan yaklaşımların zararlı örneğini mümkün olduğunca hatasız bir şekilde ayırt edebiliyor olması kullanılabilirlik açısından önemlidir. Aksi taktirde kullanıcılar hatalı bildirimlerden dolayı uyarıları göz ardı etmeye başlamaktadırlar. Dolayısı ile zararlı yazılımların iyi amaçlı yazılımlardan ayırt edilmesi ve AV yazılımları tarafından tespit edilebilmesi için mümkün olduğunca kesin çizgiler ile tanımlanmasına ihtiyaç vardır. Zararlı yazılım tanımının belirsizliği zararlı yazılımları sınıflandırmak için genel kurallar üretilmesini zorlaştırmıştır. Halihazırdaki güvenlik ürünlerinde yaygın olarak imza-tabanlı yöntemler kullanılmaktadır. Bu yöntemle, zararlı yazılımlar insanlar tarafından analizi gerçekleştirilerek ayırt edici şekilsel kurallar oluşturulmaktadır. Bilgisayar güvenliğinin sağlanmasına yönelik olarak imza ve anomali tabanlı yaklaşımlar kullanılmaktadır. Ancak ileriki bölümlerde incelenen yöntemlerin her zaman bu iki yaklaşımdan birine dahil edilemeyeceği görülecektir. Tez kapsamında sunulan toplu sınıflandırıcı, yazılım örneklerinin istatistiksel özelliklerinin kullanılması açısından anomali-tabanlı sistemlere benzemekle birlikte yapılan çalışmada sadece istatistiksel özellikler kullanılmamıştır. Ayrıca çoklu sınıfa ait dokuz farklı zararlı yazılım örnekleri üzerinde sınıflandırma çalışması yapılmıştır. Anomali tabanlı sistemlerde genelde iki sınıfa ait örnekler (anomali var/yok) bulunmaktadır. Bu nedenle imza-tabanlı yöntemlerle benzeşen çalışma; sabit karakter ve bayt dizileri gibi imzalar yerine doğrusal olmayan sınıflandırma fonksiyonunun kullanıldığı bir imza-tabanlı sınıflandırıcı gibi değerlendirilebilir. Sunulan yaklaşımda, sınıflandırma fonksiyonu makine öğrenmesi modelleri tarafından örnekler üzerinden otomatik olarak çıkarılmaktadır

1.2. Problem Tanımı

Zararlıların diğer yazılımlardan ayırt edilmesi, bir başka deyişle zararlı yazılım olarak sınıflandırılması, AV gibi güvenlik yazılımlarının en temel problemidir. Bu kapsamda yazılımın zararlı olduğunun tespit edilmesi ve hangi zararlı yazılım tipine ait olduğunun

belirlenmesi gerekmektedir. Zararlı yazılımın tespiti sonrasında, belirlenen zararlı yazılım tipine göre zararın engellenmesi ya da tersine çevrilmesi konusunda çalışmalar gerçekleştirilir. İncelenecek yazılımın zararlı olup olmadığının belirlenmesi için statik dosya analizleri ya da çalışma zamanı davranışının incelenmesi yoluna gidilmektedir. Tüm zararlı yazılım belirleme ve sınıflandırma yöntemleri bu iki yaklaşımdan biri veya ikisini beraber kullanarak analiz işlemlerini gerçekleştirmektedir.

Davranış tabanlı zararlı tespit yönteminde oluşturulan kum havuzu gibi sanallaştırılmış ortamlarda yazılımın sistemde eriştiği kaynaklar, programın dinamik özellikleri ve kullanıcı uzayı uygulama etkileşimlerinden yola çıkılarak nasıl bir davranış sergilediği tespit edilmektedir. Ardından bu davranışlardan yola çıkarak yazılımın zararlı olup olmadığı tespit edilmektedir. Örnek olarak eğer yazılım bir süre sonra diskteki tüm dosyaları şifreliyorsa, bulunduğu ağda tarama yapmaya başlıyorsa, sistemdeki bazı dosyaları olağandışı şekilde değiştiriyorsa yazılım zararlı olarak tanımlanabilir. Zararlı yazılım geliştiricileri yazılımların kum havuzu gibi ortamlarda analiz edilememesi için önlemler almaktadırlar.

Statik analiz yönteminde uygulamanın hangi kütüphaneleri kullandığı ve önceki örnekler kullanılarak hazırlanmış imzalarla uyuşup uyuşmadığı gibi bilgiler üzerinden testler yapılmaktadır. Tez kapsamında yapılan çalışmada çalıştırılabilir dosyalardan statik analiz ile çıkartılan özellikler kullanılarak veri kümesi üzerinde farklı zararlı yazılım tiplerinin belirlenmesi amaçlanmıştır. Bu şekilde sadece makine öğrenmesi yöntemleri ile yorumlanabilir bir AV yazılımının gerçek uygulamalarda kullanılabileceği gösterilmek istenmiştir. Bu bağlamda statik özellikler kullanılarak yapılan sınıflandırmanın sınırları ve kullanılabilirliği incelenmiştir.

1.3. Tezin Hedef ve Katkıları

Tez kapsamında yapılan çalışmanın hedef ve katkıları aşağıda listelenmiştir:

1. Sunulan yöntem, dosyanın formatından bağımsız çalışacak şekilde tasarlanmıştır. Dolayısı ile deneylerde kullandığımız Windows PE (portable executable – taşınabilir ve çalıştırılabilir dosya) tipinde dosyalar yerine Unix-benzeri

sistemlerde kullanılan ELF (executable and linkable format – çalıştırılabilir ve bağlanabilir format) ve Mach-O dosya tiplerinde dosyalarda da kullanılabilmektedir. Bunun için yazılımda herhangi bir değişiklik yapılmasına ihtiyaç bulunmamaktadır. Burada önemli nokta eğitim ve test kümesinin benzer dosya formatlarına sahip olması gerektiğidir. Çünkü her ne kadar model dosya formatından veya yazılımların çalıştığı CPU (central processing unit – merkezi işlemci birimi) komut setinden bağımsız olsa da öğreneceği özellikler üzerinde bu nitelikler önemlidir.

2. Günümüzde yapılan çalışmalarda gelişmiş özelliklere sahip kum havuzu var olmakla beraber, pratikte zararlı yazılımlar bu sanal sistemler üzerinde çalıştığını fark edebilmekte ve normalde sergilemesi gereken davranışları sergilememektedirler. Dolayısı ile sunulan yöntem ile bu tip atlatma yöntemlerine karşı bağışıklık sağlamaktadır.
3. Zararlı yazılımların statik analizi için örnek oluşturabilecek, hesaplama maliyeti düşük bir makine öğrenmesi sistemi oluşturulmasıdır. Dolayısı ile tez kapsamında yapılan çalışmanın AV ve IDS/IPS benzeri güvenlik ürünlerinde kullanılabileceği değerlendirilmektedir. Statik analiz yönteminin tek başına kullanılması durumunda zararlı yazılım üreticileri karşı önlem olarak ilgili kodların içerisine rastgele içerik koymak ya da zararlı kodların şifrelenerek konması gibi karşı adımlar atabilmektedirler. Buna karşı kodun analiz edilerek bu tip adımların atılıp atılmadığına yönelik analizlere yer verilebilmektedir.
4. Teoride her zaman zararlı bir yazılımın kum havuzundan veya sanal makinelerden kaçması ve test ortamına zarar vermesi mümkündür. Şu ana kadar sanallaştırma sistemlerinde ve işlemcilerde ortaya çıkan zafiyetler göstermiştir ki bunlar çok da olanaksız değildir. Sanallaştırma güvenliği alanında yapılan çalışmalarda, kullanılan yazılımlarında bulunan bazı zafiyetler sanal sistemde çalışan bir kullanıcı yazılımının konak sisteme erişmesini mümkün kılmaktadır (Ferrie 2007). Ayrıca donanımların fiziksel özelliklerinden kaynaklanan bazı yan kanallar da sanal sistemin gerçek donanımdan tamamen izole edilmesini imkânsız kılmaktadır (Wang ve ark. 2006). Szefer (2019) yaptığı çalışmada işlemcilerin mikromimari tasarımı nedeniyle oluşan gizli ve yan kanalları göstermiştir. Bunun yanında RAM gibi bellek birimlerinde de yetki yükseltmeyi sağlayan yan kanal

saldırıları gözlemlenmiştir (Kwong ve ark. 2020). Statik ve dinamik analiz yöntemlerinin birbirlerine olan üstünlükleri 2.1.2 bölümünde tartışılmıştır.

1.4. Tezin Anahatları

Yapılan tez çalışması beş ana başlıkta toplanmıştır. Bunlardan ilki olan "Giriş" bölümünde zararlı yazılım araştırması ve zararlı yazılım tanımadaki temel kavramlar tanıtılmıştır. Bunun yanında günümüzde zararlı yazılımların geldiği nokta, zararlı yazılım tanımadaki zorluklar ve güvenlik ürünlerinin durumu üzerinde durulmuştur. Ardından zararlı yazılımların sınıflandırılması problemi, şu andaki çalışmaların durumu ve amaçlardan bahsedilmiştir. Ardından tez kapsamındaki çalışmada yapılan katkılar ve tez çalışmasının hangi sorunlara çözüm oluşturabileceği tartışılmıştır. İkinci bölüm olan "Kuramsal Temeller ve Kaynak Araştırması" bölümünde bilgisayar virüsleri ile ilgili kuramsal bulgular ve zararlı yazılım belirleme konusundaki yaklaşımlar incelenmiştir. Bu bölümde ayrıca tez çalışması boyunca kullanılacak çalıştırılabilir dosya özellikleri, makine öğrenmesi yöntemleri ve bu yöntemlerin başarımı ölçmekte kullanılan hesaplar ile alakalı inceleme yapılmıştır. Bu şekilde okuyucu bu yöntemlerin zararlı yazılım sınıflandırmayı hangi matematiksel temellere dayandırdığı hakkında bilgi sahibi olacak ve deneylerde alınan sonuçları anlamlandırabilecektir. "Materyal ve Yöntem" olarak adlandırılan üçüncü bölümde bahsedilen kuramsal temellerin gerçeklendiği yazılım kütüphaneleri gibi alt parçalardan bahsedilmiş ve bu vasıtayla yapılan tasarım tercihleri okuyucuya açıklanmıştır. Bunun yanında veri kümesi gibi materyallerin özellikleri ve yazılım mimarisinin açıklaması yapılmıştır. "Bulgular ve Tartışma" bölümünde ilk olarak topluluk modelinde kullanılacak makine öğrenmesi modelleri aynı veri kümesi üzerinde tek tek denenmiş ve bu şekilde bu modellerin tek başına elde ettiği başarımlar belirlenmiştir. Bu modeller dosya bayt frekansından elde edilen bilgiler için ayrı, entropi ve dosya boyutu bilgisi için ayrı ayrı eğitilerek bu modellerin başarımları gözlenmiştir. Farklı özellikleri ve farklı makine öğrenmesi yöntemlerini kullanan bu modellerin bir topluluk modelinde birleştirilmesi gösterilmiştir. Yapılan bu deneyler sonucunda görülmüştür ki oluşturulan hibrit topluluk modeli tek bilgi kaynağından ve tek makine öğrenmesi yönteminden daha başarılı sonuçlar vermiştir. Son bölümde ise çalışmanın

zararlı yazılım araştırmasına katkıları, zararlı yazılım araştırmasında hibrit modellerin kullanımının gelecekteki durumu ve yapılabilecek çalışmalar tartışılmıştır.

2. KURAMSAL TEMELLER ve KAYNAK ARAŞTIRMASI

2.1. Kuramsal Temeller

Bu bölümde, sunulan çalışmada kullanılan yönteme ilişkin zararlı yazılımlar ile alakalı kuramsal bulgular, zararlı yazılım belirlemenin temelleri ve makine öğrenmesi modellerinin kuramsal altyapı araştırması ile ilgili sonuçlara yer verilmiştir.

2.1.1. Bilgisayar virüsleri ve karar verilememezlik

Zararlı yazılımların belirlenmesi problemi, diğer bir ifadeyle bir programın zararlı olup olmadığının sınıflandırılması ile ilgili birçok çalışma yapılmış olup bunlardan bazıları problemin doğasına ışık tutmuştur. Bunlardan en çarpıcı olanı Fred Cohen tarafından bilgisayar virüsleri üzerine yapılan çalışmadır. Cohen (1987) bilgisayar virüslerini tanıyan evrensel bir tanımlayıcı yapılamayacağını, Turing'in durma probleminin karar verilemezliğini kanıtladığına benzer şekilde kanıtlamıştır (Turing 1937).

Bu kapsamda bilgisayar virüsleri diğer programlara kendi kodunu ekleyerek bulaşma işlemini gerçekleştirebilecek programlar olarak tanımlanmaktadır (Cohen 1987). Bilgisayar virüslerini belirleyen bir A algoritması için, A(p) bir p programının virüs olup olmadığını göstermektedir. A algoritması için aşağıdaki p programı verilmiştir. Bulaş_ve_zarar_ver() fonksiyonu programın diğer programlara bulaşma ve zarar verme gibi virüs işlevlerini gerçekleştirdiği kısımdır.

Eğer A(p) 'Doğru' ise
    Programı sonlandır
Eğer A(p) 'Yanlış' ise
    Bulaş_ve_zarar_ver()

Yukarıda gösterilen p programı kendi içerisinde bir çelişki içermektedir. Eğer p programı virüs ise A(p) 'Doğru' dönecektir. Ancak bu durumda program sonlanacaktır ve bu nedenle p programı bilgisayar virüsü değildir. Ancak p virüs değil ise bu sefer A(p) yanlış dönecektir. Bu durumda ise Bulaş_ve_zarar_ver() fonksiyonu çalışacak ve bilgisayar

virüsünün gerçekleştirmesi gereken işlemleri gerçekleşecektir (Chess ve White 2000). Bu nedenle böyle bir A algoritması olamaz. Genel olarak durum aşağıdaki kod örneğinde görülmektedir.

Virüs_ise_sonlandır(p)
Bulaş_ve_zarar_ver()

Buradaki tespit sonucu Virus_ise_sonlandır() fonksiyonunun sonlanıp sonlanmamasıyla ilişkilidir. Bu nedenle bu problem durma problemine indirgenmiştir ve karar verilemezdir. Bu tespit sonucunda görülmektedir ki bilgisayar virüsleri için evrensel bir sınıflandırıcı yapılamaz.

2.1.2. Statik özellikler ve sınıflandırma

Zararlı yazılım analizi, yazılımın durumuna göre iki yaklaşımla yapılabilir. Dinamik analizde, analist zararlı yazılımın ortamla etkileşimini inceler. API (application programming interface – uygulama programlama arayüzü) çağrıları, ağ trafiği, çalıştırılan yazılım kodları ve diğer kullanıcı programlarıyla etkileşim yazılımın davranışını analiz etmek için kullanılır. Statik analizde ise zararlı yazılım analistleri statik inceleme araçlarıyla dosya bilgilerini, çalıştırılabilir kodları ve diğer dosya bölümlerini incelerler. Sembolik çalıştırma gibi bazı ileri düzey yöntemler de statik bilgiyi kullanarak dinamik davranışı belirlemeye çalışır (King 1976). Tez kapsamındaki çalışmada çıkarılan statik özellikler ise bir zararlı yazılım analisti tarafından kullanılmamaktadır. Sunulan tezde ise bu özellikler örneklerin bir yapay zekâ yazılımı ile farklı zararlı yazılım ailelerine sınıflandırılmasında kullanılmaktadır.

Statik ve dinamik zararlı yazılım analizinin birbirine göre farklı yararlı kullanımları vardır. Güçlü bir şekilde şifrelenmiş ve karıştırılmış kodlar statik analiz yöntemiyle incelenemeyebilir, diğer yandan zararlı yazılım geliştiricileri çalışma zamanında yapılan incelemeden kaçmak için birçok teknik uygulamaktadır. Sanal ortamları belirlemek ve çalışma zamanı ile ilgili yanlış bilgi vermek için kullanılan bu yöntemler genelde anti-analiz teknikleri olarak adlandırılır (Chen ve ark. 2008, You ve ark. 2010). Zararlı yazılım

geliştiricileri hata ayıklayıcılar, sanal makineler ve kum havuzları gibi farklı analiz araçlarında inceleme yapılmasını zorlaştırmak için genellikle araca özel anti-analiz yöntemlerini kullanır. Bunun yanında doğal olarak zararlı yazılımın çalıştırılması üzerinde koşulan sisteme zarar verilebilir.

2.1.3. Shannon entropisi

Shannon entropisi veya bilgi entropisi, kısaca entropi olarak adlandırılır. Entropi bir rastgele değişkenin olası çıktılarındaki belirsizliği, bu nedenle de bilgiyi ölçer (Shannon 1948). Bilgi entropisine ait hesaplama Denklem 2.1'de görülebilir.

$$H(X) = -\sum_{i} P_X(x_i) log_{256} P_X(x_i) \tag{2.1}$$

Denklemde görülen X rastgele değişkeni, $P_x(x_i)$ ise x_i değeri için görülme olasılığını göstermektedir. Çalışma kapsamında kullanılan olasılık dağılımı bayt dizisine ait olduğundan logaritma fonksiyonunda kullanılan taban değerinin 256 seçilmiştir. Bunun nedeni kullanılan bayt değerlerinin 8 bit değerleri ifade etmesidir.

Entropi bilgi teorisinde önemli bir yer tutmaktadır. Bunun yanında Shannon'un bilgi tanımı o kadar geniş kapsamlıdır ki aynı bilgi tanımı fizik gibi bilim dallarının yanında kriptoloji ve zararlı yazılım araştırmasında da kendine önemli bir yer edinmiştir. Entropi sıkıştırma ve şifreleme gibi yöntemlerin uygulanıp uygulanmadığının belirlenmesi için kullanılmaktadır. Bu şekilde paketlenmiş ve şifrelenmiş yazılımların dosya bölümleri incelenerek bu yazılımların asılları elde edilmeye çalışılır. Tez kapsamında da entropi özellikleri dosya boyutuyla beraber kullanılarak modellerin bu tespitleri yapabilmesi sağlanmaya çalışılmıştır.

2.1.4. Karar ağacı

Karar ağacı parametrik olmayan bir gözetimli öğrenme yöntemidir. Karar ağacı özellik ve kararlar arasındaki ilişkiyi ağaç benzeri graflar kullanarak modeller. Hem sınıflandırma hem de regresyon için kullanılabilen karar ağaçlarının birçok alanda

uygulaması vardır. Karar ağacı bilgi kazancı en yüksek olan kök düğümden başlayarak örnek uzayını öz yinelemeli bir biçimde bölümler. Bu şekilde düğümlerdeki basit karşılaştırmalar ile alt karar ağaçlar oluşturur. Girdi uzayı x_j için Denklem 2.2'de görülen koşul bir karar düğümünde w_{m0} ile yapılan dallanmayı göstermektedir. Koşul olası x_j değerlerinin her durumu için yeni bir dal oluşturur.

$$f_m(x) : x_j > w_{m0} \tag{2.2}$$

Bu şekilde karar uzayı ikiye ayrılır. $L_m = x|x_j > w_{m0}$ ve $R_m = x|x_j < w_{m0}$ şeklindeki iki karar uzayı yine aynı şekilde alt uzaylara ayrılarak sadece aynı sınıfta örneklerin bulunduğu yapraklar oluşana kadar devam eder (Özdemir 2014).

Tez kapsamında yapılan çalışmada kullanılan karar ağaçları CART (classification and regression tree – sınıflandırma ve regresyon ağacı) algoritması ile oluşturulmuştur (Rokach ve ark. 2005). CART algoritması C5.0 ile beraber ID3 ve C4.5 algoritmalarının ardılıdır ve ikili ağaçlar oluşturulmasını sağlar (Rokach ve ark. 2005). CART kategorik özellikleri de kullanabilmekle beraber deneylerde kullanılan kütüphane uyarlaması sadece sayısal değerleri desteklemektedir. CART algoritması karar düğümleri koşullarının seçilmesinde Gini kirlilik ölçümünden yararlanır (Rokach ve ark. 2005).

$$I_G(p) = 1 - \sum_{i=1}^{c} p_i^2 \tag{2.3}$$

Denklem 2.3'de *C* sınıflı bir veri kümesinde p_i, *i* sınıfından bir örneğin rastgele seçilme olasılığıdır. Bu şekilde hesaplanan Gini kirlilik değeri kararın veri uzayını ne kadar saf alt uzaylara böldüğünü gösterir. Bu şekilde Gini kazancı yüksek olan koşullar seçilerek yinelemeli şekilde sınıflandırma elde edilir.

2.1.5. Rastgele Orman Algoritması

Rastgele orman algoritması rastgele üretilen karar ağaçlardan oluşan bir topluluk öğrenme modelidir (Breiman 2001). Rastgele orman modelleri karar ağaçları gibi

sınıflandırma ve regresyon için kullanılmaktadır. Rastgele orman yöntemi eğitim sırasında birden çok karar ağacı üreterek sonuçta seçilen ağaçların oluşturduğu bir topluluk sınıflandırıcısı üretir. Kullanılan bu topluluk öğrenmesi modeli bootstrap aggregation veya kısaca begging olarak adlandırılmaktadır. Bu şekilde sınıflandırıcı fonksiyonun varyansının düşürülmesi amaçlanmıştır. Özellikle karar ağaçları gibi yüksek varyanslı ve düşük önyargılı sistemlerin başarımının artırılması için kullanılmaktadır.

Rastgele orman algoritması aşağıdaki şekildedir.

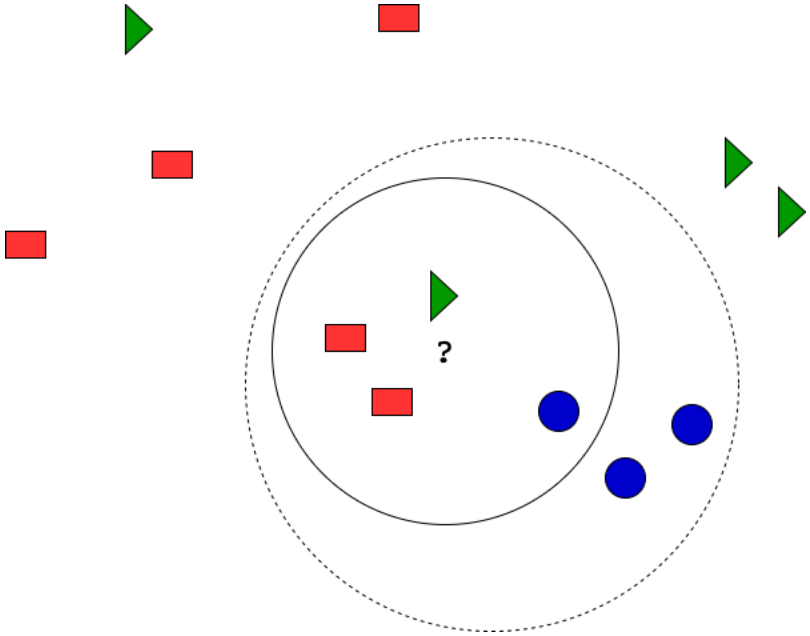
1. *b = 1'den B'ye kadar:*
    a. *Eğitim kümesinden N uzunlukta bootstrap örneği seçilir.*
    b. *Karar ağacı bu örneklerle eğitilir.*
2. Üretilen bu B tane karar ağacının çıktıları birleştirilerek bir tane sınıflandırma sonucu üretilir (Hastie 2017).

Kullanılan begging yönteminin karar ağacı yönteminin başarımını çoğu uygulamada artırdığı görülmüştür. Bu nedenle rastgele orman algoritması birçok uygulamada kullanılmaktadır.

2.1.6. KNN

KNN (k-nearest neighbours) veya k-en yakın komşuluk algoritması (Altman 1992), sınıflandırma ve regresyonda kullanılan parametrik olmayan bir yöntemdir. Örnek temelli bir yöntem olan KNN yeni gelen örneklerin eğitim sırasında görülen örneklerle karşılaştırılması sonucu çıktı olarak sınıf veya sayısal değer elde edilir.

Şekil 2.1'de görülen örnekte kırmızı dikdörtgenler birinci sınıfı, mavi daireler ikinci sınıfı ve yeşil üçgenler üçüncü sınıfı temsil etmektedir. Şekilde görülen veri eğitim sonrası iki boyutlu özellik uzayında temsil edilmektedir. Yeni gelen ve soru işareti "?" ile temsil edilen örneğin model tarafından sınıflandırılması gerekmektedir.



Şekil 2.1. Eğitim sonrası KNN modeli ve yeni gelen örneğin sınıflandırılması

Şekilde k komşuluk değerinin farklı durumlardaki gösterimleri kesikli ve sürekli çizgi ile gösterilen iki örnek kümesinde belirtilmiştir. Buna göre KNN komşuluk değerinin dört olduğu durum için sürekli çizgi ile gösterilen örnek kümesini değerlendirecek ve kırmızı dikdörtgenlerin diğer sınıftaki örneklere göre çok sayıda olması nedeniyle yeni gelen örneği birinci sınıfa dahil edecektir. Ancak, komşuluk değerinin altı seçilmesi durumunda mavi dairelerin çok sayıda olması nedeni ile yeni örnek farklı şekilde sınıflandırılacaktır. Göründüğü gibi KNN anlaşılması basit bir algoritmadır. Ancak uygulamada bazı parametrelerin belirlenmesi ve en yakın noktalar gibi bazı ifadelerin ölçülebilir şekilde ifadesi gereklidir.

En yakın noktaların bulunabilmesi için KNN ile beraber farklı algoritmalar kullanılmıştır. Bu algoritmalar zaman karmaşıklığı açısından farklılık göstermekle beraber sınıflandırmaya en etkili ölçüt örnekler arasındaki uzaklık ölçümüdür. Tezde yapılan çalışmada Öklid uzayında tanımlı bir fonksiyon olan Minkowski uzaklık ölçümünün özel hali olan Öklid uzaklığı kullanılmıştır.

$X = (x_1, x_2 \dots x_n)$ ve $Y = (y_1, y_2 \dots y_n) \in \mathbb{R}$ şeklinde iki özellik vektörü için Minkowski uzaklık formülü Denklem 2.4'de görülmektedir.

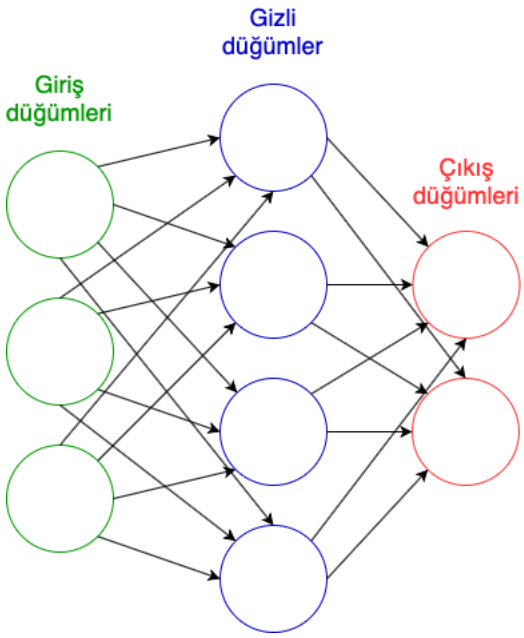
$$D_m(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{2.4}$$

Minkowski uzaklığının *p=1* için çözümü Manhattan uzaklığı, *p=2* için çözümü ise Öklid uzaklığıdır. Öklid uzayında en yaygın kullanılan mesafe ölçüm biçimi olan Öklid uzaklığı Denklem 2.5'de görülmektedir.

$$D_e(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.5}$$

2.1.7. Yapay sinir ağları

Yapay sinir ağları günümüzde en yaygın kullanılan makine öğrenme modellerinden biridir ve birçok diğer makine öğrenmesi yönteminin temelini oluşturur. LSTM (long-short term memory – uzun kısa süreli bellek) ve CNN (convolutional neural network – konvolüsyonel sinir ağı) gibi popüler derin öğrenme modelleri aslında yapay sinir ağlarının farklı şekillerde bir araya gelmesinden oluşmaktadır. Yapay sinir ağları biyolojik sinir sistemlerinden esinlenerek geliştirilmiş bir modeldir. Bu tip modellerin temel yapı taşı içerdiği nöronlardır ve bu nöronlar yan yana ve art arda gelerek yapay sinir ağı modellerini oluştururlar. Örnek yapay sinir ağı diyagramı Şekil 2.2'de görülmektedir.



Şekil 2.2. Örnek yapay sinir ağı diyagramı

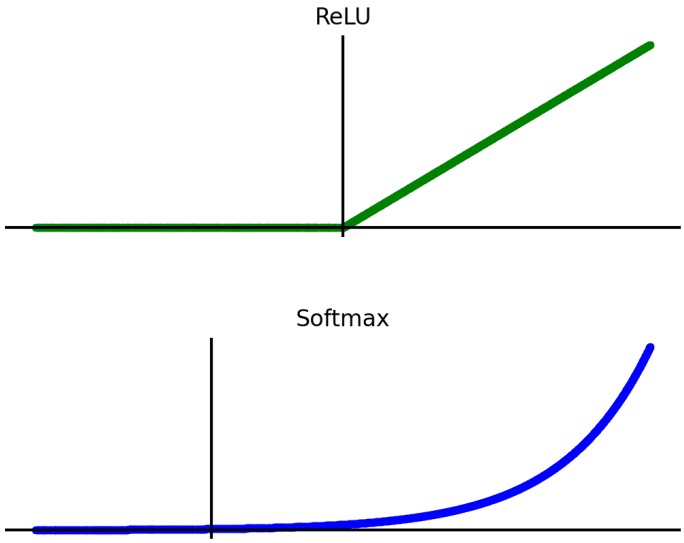
Şekil 2.2'de önceki katmandaki çıktıları girdi olarak alan ileri beslemeli genel bir yapay sinir ağı modeli gösterilmiştir. Burada bahsedilmesi gereken birkaç işlem vardır. Bunlardan ilki nöronların çıktıları nasıl ürettiğidir. Nöronlar kendilerine gelen girdileri ağırlıkları ile beraber Denklem 2.6'daki şekilde hesaplarlar. Burada $p_j(t)$ j düğümüne gelen girdilerin ağırlıklandırılmış halini, $o_i(t)$ önceki katmandaki i. düğümün çıktısını, w_{ij} i. düğümün ağırlığını göstermektedir. Eğer önceki katmanda önyargıyı ifade eden sabit bir düğüm var ise o da aynı şekilde ağırlıklandırılarak toplama işlemine dahil edilir.

$$p_j(t) = \sum_i o_i(t)\, w_{ij} \tag{2.6}$$

Bu aşamadan sonra Denklem 2.7'de görülen şekilde j. düğümün çıktısı a_j aktivasyon fonksiyonundan geçirilerek hesaplanır.

$$o_j(t) = a_j(p_j(t)) \tag{2.7}$$

Aktivasyon fonksiyonları doğrusal olmayan fonksiyonları modelleyebilecek şekilde seçilir. Tez kapsamında yapılan çalışmada kullanılan aktivasyon fonksiyonları aşağıda açıklanmıştır. Şekil 2.3'de "Rectified linear" parçalı fonksiyonu ve bu aktivasyon fonksiyonu kullanan ReLU (rectified linear unit – düzeltilmiş doğrusal birim) nöronuna ait çıktı, ayrıca softmax fonksiyonunun çıktısı görülmektedir.



Şekil 2.3. Yapay sinir ağı modelinde kullanılan aktivasyon fonksiyonu grafikleri

Yapay sinir ağı yöntemlerinde yaygın kullanılan kullanılan fonksiyonlar basit matematiksel yapılar kullanılarak açıklanabilir.

$$R(x) = enbuyuk(0, x) \tag{2.8}$$

$enbuyuk: \mathcal{R}^2 \rightarrow \mathcal{R}$ için Denklem 2.8 ReLU aktivasyon fonksiyonunu, aynı şekilde Denklem 2.9 parçalı fonksiyon şeklinde aynı fonksiyonu göstermektedir.

$$R(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{2.9}$$

ReLU aktivasyon fonksiyonu özellikle kaybolan gradyan problemine kısmen çözüm olması açısından çoğu çok katmanlı sinir ağı uygulamasında tercih edilmektedir. Bu

problem öğrenmeyi gerçekleştirmek için geri yayılım yöntemi kullanan ve özellikle çok katmanlı olan sinir ağları açısından önemli bir sorundur (Hochreiter 2001).

$$S(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{2.10}$$

Denklem 2.10'da görülen softmax fonksiyonu ise çıktı olarak olasılık değeri vermektedir. Bu şekilde örneği uygun/uygun değil (var/yok) şeklinde veya en olası gruba göre sınıflandırmak mümkün olmaktadır. Bu nedenle softmax fonksiyonu genelde çıktı olarak sınıf değeri elde edilmek istenilen uygulamalarda kullanılmaktadır.

2.1.8. XGBoost

XGBoost ağaç yapılarını kullanan bir gradyan artırma sistemi ve açık kaynaklı yazılım kütüphanesidir (Friedman 2001). Aşağıda XGBoost'un topluluk ağaç modelini nasıl öğrendiği anlatılmıştır.

m özellikten oluşan n örneğin oluşturduğu veri kümesi $\mathcal{D} = \{(x_i, y_i)\}$ çiftlerinden oluşmaktadır. Veri kümesi özellikleri Denklem 2.11'deki şekilde tanımlanabilir.

$$|\mathcal{D}| = n, x_i \in \mathcal{R}^m, y_i \in \mathcal{R} \tag{2.11}$$

Bu veri kümesini sınıflandıracak ağaç topluluk modeli, $\mathcal{F}$ kümesindeki K tane sınıflandırma fonksiyonu kullanılarak tahminlerini yapmaktadır. Topluluk modeli tahmin fonksiyonu Denklem 2.12'de gösterilmiştir.

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i), f_k \in \mathcal{F} \tag{2.12}$$

Topluluk modelinde kullanılacak fonksiyonların öğrenilmesi için tahmin ve gerçek sınıf arasındaki hata hesaplanmalıdır. Bunun yanında ezberlemenin önüne geçilmesi için

fonksiyon karmaşıklığı da cezalandırılmaktadır. Bu iki değişkeni ifade eden optimizasyonu gereken fonksiyon Denklem 2.13'de gösterilmiştir.

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i) + \sum_k \Omega(f_k) \tag{2.13}$$

Burada l sınıflandırma sonucu $\hat{y}_i$ ve gerçek sınıf y_i arasındaki farklı ölçen kayıp fonksiyonu, Ω ise modelin karmaşıklığını cezalandıran fonksiyondur. İkinci fonksiyon modelin örnekleri ezberlememesi için kullanılmaktadır. Chen ve Guestrin (2016) çalışmalarında Denklem 2.13'de fonksiyonların parametre olarak alındığını ve bu denklemin Öklid uzayındaki olağan yöntemlerle optimize edilemeyeceğini ifade etmişlerdir. XGBoost yönteminin iteratif yöntemi için amaç fonksiyonu ise Denklem 2.14'deki şekilde ifade edilebilir.

$$\mathcal{L}^{(t)} = \sum_i l(y_i, \hat{y}_i{}^{(t-1)} + f_t(x_i)) + \Omega(f_k) \tag{2.14}$$

Bu denklemim optimize edilebilir bir biçime dönüşmesi için araştırmacılar Denklem 2.14'e ikinci dereceden Taylor serisine açmışlardır. $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i{}^{(t-1)})$ ve $h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i{}^{(t-1)})$ birinci ve ikinci seviyeden gradyan istatistikleri olmak üzere Denklem 2.14'ün Taylor serisine açıldıktan sonra sabit elemanların çıkarılarak sadeleştirilmiş şekli Denklem 2.15'de görülmektedir.

$$\tilde{\mathcal{L}}^{(t)} = \sum_i [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_k) \tag{2.15}$$

Denklem 2.15, *t* adımı için basitleştirilmiş amaç fonksiyonunu göstermektedir. Bu şekilde denklem optimize edilebilir forma dönüştürülmüştür. Bu denklem üzerinde Ω ceza fonksiyonu açılarak sabit $q(x)$ ağacı için *t* adımında Denklem 2.16'daki amaç fonksiyonu elde edilir. Bu denklem *t* adımındaki en küçük kayıp değerini vermektedir.

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T \tag{2.16}$$

Denklem 2.16'da q ağacının kalitesini ölçmek için gereken amaç fonksiyonu elde edilmiş olur. λ, γ ceza fonksiyonu için kullanılan parametreler, T yaprak sayısı ve I_j j yaprağı durumlarıdır.

Bu amaç fonksiyonu kullanılarak karar ağaçlarının eğitimine benzer şekilde yinelemeli bir süreç işletilir. Seçilen kökten başlayarak aynı yaprakta bir tipte örnekler kalacak şekilde örnek uzayı bölümlenir. Bu şekilde nihai XGBoost modeli elde edilir (Chen ve ark. 2016). Yazılım kütüphanesi ile ilgili bilgi bölüm 3.1.4'de verilmiştir.

2.1.9. Topluluk öğrenmesi

Topluluk öğrenmesi birden çok sınıflandırıcı tarafından oluşturulan topluluk modellerinin makine öğrenmesinde kullanılmasını temel alan yaklaşımdır. Makine öğrenmesi ile alakalı matematiksel modeller elektronik bilgisayarların yapay zekâ çalışmalarında kullanılmasında çok daha önce ortaya çıkmışlardır. Ancak topluluk öğrenmesi yaklaşımı özellikle günümüz hesaplama kabiliyetlerinin ve veri kaynaklarının kullanılması sonucunda ortaya çıkmış bir yaklaşımdır. Bu yaklaşımın popüler olmasının nedeni şüphesiz yarışmalarda (veya bilimsel yayınlarda yapılan kıyaslamalarda) bilindik basit yöntemlerden oluşturulan topluluk modellerinin birçok karmaşık modele göre üstün başarı göstermesidir.

2.1.10. Başarım ölçümünde kullanılan göstergeler

Özellikle makine öğrenmesinde ve bilgi erişim sistemlerinde yaygın olarak kullanılan, örneklerin asıl ait olduğu sınıfa ve sınıflandırma sonucuna dayanan kümelerin kesirli ifadelerinden oluşan göstergelerdir. İlk olarak örneklerin hangi kümelere ayrıldığı Çizelge 2.1'de gösterilmiştir.

Çizelge 2.1. Örneklerin ait olduğu sınıfa ve sınıflandırma sonucuna göre kümeler

Test / Durum	Pozitif	Negatif
Pozitif	Doğru pozitif	Yanlış pozitif
Negatif	Yanlış negatif	Doğru negatif

Çizelgede gösterilen yatay sütunda örneğin gerçek sınıfı, düşey sütunda ise test sonucu iki sınıflı durum için gösterilmektedir.

Test sonucunda ortaya çıkan bu kümeler sınıflandırıcı ile ilgili farklı özellikleri betimleyen ölçümlerin oluşturulması için kullanılmıştır.

Çizelge 2.2. Tez kapsamında kullanılan istatistiksel ölçümler ve denklemleri

Ölçüm	İfade	Denklem
Doğruluk	$\frac{\text{Doğru sınıflandırma}}{\text{Tüm örnekler}}$	$\frac{\lvert DP\rvert + \lvert DN\rvert}{\lvert DP\rvert + \lvert DN\rvert + \lvert YP\rvert + \lvert YN\rvert}$
Hassasiyet	$\frac{\text{Doğru pozitifler}}{\text{Tüm pozitifler}}$	$\frac{\lvert DP\rvert}{\lvert DP\rvert + \lvert YP\rvert}$
Geri çağırma	$\frac{\text{Doğru pozitifler}}{\text{Doğru sınıflandırılanlar}}$	$\frac{\lvert DP\rvert}{\lvert DP\rvert + \lvert YN\rvert}$
F-skoru	$2 * \frac{\text{Hassasiyet} * \text{Geri çağırma}}{\text{Hassasiyet} + \text{Geri çağırma}}$	$\frac{2 * \lvert DP\rvert}{2 * \lvert DP\rvert + \lvert YN\rvert + \lvert YP\rvert}$

Çizelge 2.2'de doğru pozitif (DP), yanlış pozitif (YP), doğru negatif (DN) ve yanlış negatif (YN) örnek sayısına göre istatistiksel ölçümler görünmektedir. Burada kullanılan ölçümlerin açıklamaları Çizelge 2.3'de verilmiştir.

Çizelge 2.3. Tez kapsamında kullanılan istatistiksel ölçümler ve açıklamaları

Ölçüm	Açıklama
Doğruluk	Tüm örnekler arasından doğru sınıflandırılan örneklerin oranını göstermektedir.
Hassasiyet	Sınıflandırmanın ne kadar hassas yapıldığını gösterir. Bu ölçüm pozitif sınıflandırmanın güvenilirliğini göstermektedir.
Geri çağırma	Doğru sınıflandırılanların gerçekte pozitif olarak sınıflandırılması gereken örneklere oranıdır.
F-skoru	Hassasiyet ve geri çağırmanın harmonik ortalamasını kullanan genel bir başarım ölçümüdür. F_1 skoru olarak da adlandırılır.

2.2. Kaynak Araştırması

Erken dönemde Kephart ve Arnold (1994) istatistiksel yöntemleri kullanarak zararlı yazılım bulaştırdıkları yazılımlardan otomatik imza üretmişlerdir. Zararlı yazılım araştırmasında makine öğrenmesi yöntemlerinin kullanımı başlamasıyla araştırmacılar çalıştırılabilir dosyalardan elde edilen n-gramlar ve frekans özelliklerini kullanmaktadırlar. Kephart ve ark. (1995) önyükleme sektörü virüslerini n-gramları ve yapay sinir ağlarını kullanarak sınıflandırmışlardır. Tesauro ve ark. (1996) önyükleme zararlı yazılımlarının belirlenmesi için kullandıkları sistemi IBM firmasının ticari AV ürününe entegre etmişlerdir. Schultz ve ark. (2001) PE dosyalarının DLL çağrılarını, karakter dizilerini ve bayt dizilerini kullanarak daha önceden görülmemiş zararlıları belirlemiştir. Araştırmacılar bu özellikleri kullanan farklı veri madenciliği yöntemlerini uygulamışlardır. Özellikle bayt ve karakter dizilerini kullanan makine öğrenmesi modellerinin imza-tabanlı tespit yönteminden başarılı olduğu görülmüştür. Kolter ve Maloof (2004, 2006) n-gramları kullanarak farklı makine öğrenmesi ve veri madenciliği yöntemlerini kıyaslamışlardır. Birçok diğer çalışmada da bayt özelliklerine ait n-gramlar farklı şekillerde kullanılmıştır (Abou-Assaleh ve ark. 2004, Raff ve ark. 2018, Tabish ve ark. 2009). Masud ve ark. (2008) ise n-gramları benzer şekilde assembly kodları üzerinde kullanmışlardır.

Nataraj ve ark. (2011) statik analiz ve zararlı yazılım sınıflandırma için ilk kez bilgisayarlı görü tekniklerini kullanmışlardır. Bir boyutlu binary dosyaları iki boyutlu gri ölçek görüntülere dönüştüren araştırmacılar, bu görüntüler üzerinde obje tanımada kullanılan yöntemleri kullanarak sınıflandırma yapmışlardır. Kancherla ve Mukkamala (2013) yaptıkları çalışmada asıl görüntülere ek olarak Wavelet dönüşümü ve Gabor filtresi ile özellik çıkarımı yapmışlardır. Sonuçlar göstermektedir ki zararlı yazılım görüntülerinden çıkarılan sınırlı sayıda özellikle dahi başarılı sınıflandırma yapılabildiği gözlemlenmiştir. Han ve ark. (2015), PE dosyalarından çizgi entropi grafları üretmişlerdir. Üretilen farklı entropi grafları arasındaki benzerlikten yola çıkarak sınıflandırma yapmışlardır. Narayanan ve ark. (2016), binary görüntülerinden boyut indirgeme ile PCA (principal component analysis – temel bileşen analizi) özellikleri çıkarmışlar ve bu özelliklere birden fazla makine öğrenmesi algoritması uygulamışlardır. Yaptıkları deneyler, Microsoft Malware Classification Challenge verileri üzerinde kNN ve PCA özellikleri ile düşük hesaplama maliyetiyle başarılı sınıflandırma yapılabileceğini göstermiştir (Ronen ve ark. 2018). Zhang ve ark. (2016) benzer şekilde PCA ve kNN kullanarak zararlı yazılım görüntülerini sınıflandırmıştır. Önceki çalışmalardan farklı olarak binary dosyasında bulunan görsel örüntüler yerine opcode özelliklerinden oluşturdukları görüntüler ile sınıflandırma yapmışlardır. Bu çalışma göstermektedir ki assembly opcode özellikleri kullanılarak zararlı yazılım sınıflandırma için belirleyici örüntüler oluşturulabilmektedir. Liu ve ark. (2017), binary dosyaları bir boyutlu gri ölçek vektöre haritalamışlardır. Gri ölçek vektör, opcode n-gram, CFG (code-flow graph – kod akış çizgesi) bilgisi ve kütüphane fonksiyonlarını kullanarak topluluk modeli geliştirmişlerdir. Yapılan deneylerde topluluk modelinin sadece gri ölçek vektörü kullanan modele göre daha başarı olduğu görülmüştür. Le ve ark. (2018), binary dosyalarını bir boyutlu görüntüye çevirerek farklı CNN-LSTM mimarilerinde kullanmışlardır. Çalışma sonucunda CNN-LSTM mimarilerinin sadece CNN kullanan yöntemlere göre daha başarılı olduğunu gözlemlemişlerdir. Fu ve ark. (2018), PE dosyaları bölümlerindeki bayt değerleri, entropi ve göreli boyut gibi değerleri ifade edebilmek için üç kanallı görüntüleri kullanmışlardır. Bunun yanında ASCII (American standard code for information interchange – bilgi paylaşımı için Amerikan standart kodu) karakter dizilerinden oluşan görüntüler de kullanmışlardır. Bu çalışmayla yaptıkları katkı asıl olarak kullandıkları üç kanallı görüntülerle bir dosyayı bir görüntüde birden fazla şekilde temsil etmiş

olmalarıdır. Bu teknik kullanılarak bir binary dosyaya ait birden fazla özellik bir görüntüde temsil edilebilmektedir.

Bunun yanında zararlı yazılım sınıflandırmada topluluk modellerinin başarımı artırmasıyla ilgili çalışmalar yapılmıştır. Menahem ve ark. (2009) yaptığı çalışmada karar ağacı, saf Bayes ve kNN gibi yaygın kullanılan modelleri kullanarak topluluk sınıflandırıcısı modelleri oluşturmuşlardır. Çalışmada bazı topluluk modelleri en iyi sınıflandırıcıdan daha yüksek başarım gösterirken bazıları ise daha düşük başarım göstermiştir. Smutz ve Stavrou (2016) yaptıkları çalışmada zararlı yazılımların belirlenmesi için kullanılan topluluk modellerindeki sınıflandırma sonuçlarının güvenirliğini derecelendirmişlerdir. Yaptıkları çalışmada topluluk oylama modelinin yüksek oranda birliktelik sağlamadığı koşullarda örnekleri sınıflandırılamaz olarak kabul etmişlerdir. Araştırmacılar bu yöntemi kullanarak yaptıkları denemelerde doğru pozitif ve yanlış negatif oranlarının düştüğünü gözlemlemişlerdir.

Rathore ve ark. (2019) yaptığı çalışmada opcode özelliklerini kullanarak makine öğrenmesi yöntemleri aracılığıyla sınıflandırma yapmışlardır. Araştırmacılar yaptıkları deneylerde birden fazla sayıda özellik çıkarım yöntemi ve sınıflandırıcı kullanmışlardır. Tez kapsamındaki hipotezler açısından düşünüldüğünde çalışma sınıflandırıcı karmaşıklığının başarıma etkisinin görülmesi açısından önemlidir. Çalışmada elde edilen sonuçlar göstermektedir ki rastgele orman temelli yöntem farklı özellik mühendisliği yöntemleriyle çıkarılan özelliklerin hepsinde derin yapay sinir ağı mimarilerine göre başarılı olmuştur.

Halihazırda AV yazılımları yanlış pozitif oranını göz önünde bulundurarak ağırlıklı olarak imza-tabanlı virüs tarama yöntemlerini kullanmaktadırlar. Ancak Trapmine firmasının ThreatScore adlı ürünü, sadece makine öğrenmesi motorunu kullanarak Google'ın VirusTotal web sitesi üzerinde taranan dosya veya adreslerin zararlı olup olmadığını değerlendirmektedir. Trapmine tarafından yayınlanan dökümanlarda görülen deneylerde bölüm, entropi, karakter dizileri ve kullanılan kütüphaneler gibi statik özellikler kullanılmıştır. Bu özellikler kullanılarak derin öğrenme, rastgele orman, gradyan artırma ve lojistik regresyon gibi yöntemleri kıyaslayan araştırmacılar; gradyan

artırma yönteminin derin öğrenmeye göre sınıflandırma başarısının yüksek olduğunu gözlemlemişlerdir. Bunun yanında gradyan artırma temelli ağaç algoritmasının diskte daha az yer kaplaması ve dört kata yakın yüksek hızda tarama yaptığı görülmüştür (2020a). Bu da her durumda karmaşık ve katmanlı yöntemlerin basit görünen yöntemlere göre daha başarılı olmadığını göstermektedir. Bu başka makine öğrenmesi uygulamalarında bu tip modellerin başarılı olacağını göstermez. Ama zararlı yazılım sınıflandırılmasında kullanılan veri tipinin bu tip modellerle ifade edilebileceği Microsoft Malware Classification Challange adıyla düzenlenen yarışmada birincilik elde eden Saynotooverfitting ekibinin yayınladığı çalışmada (https://github.com/xiaozhouwang/kaggle_Microsoft_Malware/blob/master/Saynotooverfitting.pdf) da görülmektedir. Yayınlanan dökümanda benzer şekilde komut sayısı, frekans ve n-gram sayıları gibi özellikler; bunların yanında ise gradyan artırma gibi modeller kullanılmıştır.

Bir diğer konu verilerin nasıl ele alınacağıdır. Bu kapsamda tez çalışmasında sıralama bilgisi ihmal edilerek özellik çıkarımı yapılmıştır. Dizi yaklaşımı ve BoW (bag of words – kelime çantası modeli) yaklaşımı yöntemlerinin birbirlerine göre yararlı yönleri bulunmaktadır. Dizi yaklaşımında word2vec ve LSTM gibi bir yöntemle veriler dizi şeklinde işlenebilir (Mikolov ve ark.). Ancak bu yöntemde çalıştırılabilir dosyanın büyük bölümünü oluşturan program kodu ve diğer birçok verinin tam olarak ardışıl bağlılığının olmaması önemli bir sorundur. Program çalışma zamanında fonksiyon çağrılarıyla kodun farklı kısımlarını çalıştırmaktadır. Bu nedenle çalıştırılabilir dosyadan elde edilen dizi ile CPU'nun program kodunu çalıştırma sırası aynı olmayacaktır ki bazı komutlar arasında hiçbir şekilde ardışıl bağlılık yoktur. Bunun yanında sıralama bilgisini önemsemeden alt diziler çıkararak frekans ve olasılık vektörlerini kullanan yaklaşımların elde ettiği özellikler aynı işlevli yazılımlar için çok farklı değerler içerebilir. Bununla ilgili yapılan çalışmalar göstermektedir ki aynı program işlevi komut setini farklı şekilde kullanarak gerçekleştirilebilir (You ve Yim 2010, Dolan 2013).

3. MATERYAL ve YÖNTEM

3.1. Materyal

Tez kapsamında tasarlanan yazılımın, yönetilebilir, elden geldiği kadar anlaşılır ve sonradan yorumlanabilir olması düşünülmüştür. Bu şekilde tez sonunda hazırlanan yazılım internete açık şekilde paylaşılarak diğer araştırmacıların da elde edilen bilgilerden ve kod kütüphanelerinden yararlanması sağlanmış olacaktır. Bu nedenle yazılım tasarlanırken birkaç hususa dikkat edilmiştir. Yazılımın modüler olması, elden geldiği kadar Python kütüphanelerinden sağlanan temel araçlardan yararlanılması ve nesneye yönelim prensiplerine bağlı kalınması amaçlanmıştır.

3.1.1. Python programlama dili

Tez kapsamında yapılan uygulamalı çalışmaların büyük bölümü Python programlama dilindeki yazılımının geliştirilmesi şeklinde gerçekleşmiştir. Sistemin geliştirilmesinde Python dilinin kullanım nedeni aşağıda bahsedilecek yazılım kütüphanelerine sahip olması ve bu alanda yapılan çalışmaların büyük çoğunda bu dilin tercih edilmesi sebebiyle makine öğrenmesi alanında Python kaynaklarının zengin olmasıdır. Bu nedenle çalışmalar süresince internet kaynaklarından ücretsiz destek alınabilmesi imkanına sahip olunmaktadır. Python versiyon 2'nin artık desteklenmemesi nedeniyle Python versiyon 3'de kodlama yapılmıştır.

Python programlama dilinin kodlanan yazılımı çalıştırmak için birden fazla yorumlayıcı vardır. Tezde hazırlanan yazılımın çalıştırılması için en yaygın kullanılan Python uyarlaması olan CPython yorumlayıcısı kullanılmıştır. CPython, C ve Python programlama dillerinde programlanmış olup hem derleyici hem yorumlayıcı olarak düşünülmelidir. Çünkü CPython, kaynak kodu önce bayt koda derleyip ardından yorumlayıcıda çalıştırmaktadır (2020b).

3.1.2. NumPy kütüphanesi

NumPy Python programlama dili için geliştirilen ve gelişmiş matematiksel işlemleri gerçekleştirmek için yüksek seviyeli arayüz sağlayan bir kütüphanedir. Bunun yanında yüksek boyutlu matrisler ve diziler için de destek sağlayan NumPy, bu veri yapıları üzerinde hızlı işlem yapma kabiliyetine sahiptir. Makine öğrenmesi uygulamasında verilere ait matrislerin ve vektörlerin saklanması ve işlenmesi için kullanılmıştır (2020c). Tezde yapılan çalışmada NumPy kütüphanesi tarafından sağlanan veri yapılarının kullanımının öncelikli nedeni sağladığı hız avantajıdır.

3.1.3. Scikit-learn kütüphanesi

Sklearn olarak da bilinen scikit-learn yazılım kütüphanesi, özgür yazılım bir makine öğrenmesi kütüphanesidir. Scikit-learn, bir Google Summer of Code projesi olarak geliştirilmeye başlanmasının ardından Fransız araştırma enstitüsü INRIA tarafından destek görerek geliştirilmiştir. Scikit-learn Python programlama dilinde en yaygın kullanılan makine öğrenmesi ve yapay zekâ kütüphanelerinden biridir (2020d).

3.1.4. XGBoost Kütüphanesi

XGBoost gradyan artırma için kullanılan yazılım kütüphanesidir. C, C++, Java, Python, R, Julia, Scala, Ruby ve Swift gibi dillere desteği vardır ve Windows, Linux ve MacOS üzerinde çalışabilmektedir. Tez kapsamında yapılan deneylerde gradyan artırma yöntemi için Kaggle üzerindeki yarışmalarda sıkça yüksek skorlar elde eden XGBoost kütüphanesi tercih edilmiştir (2020e, 2020f).

3.1.5. Pickle kütüphanesi

Pickle kütüphanesi serileştirme ve ters-serileştirme işlemlerinin yapılabilmesi için gereken binary protokollerini Python diline kazandırmıştır. Bu şekilde Java ve C# gibi modern nesne yönelimli programlama dillerine benzer şekilde Python dilinde nesnelerin serileştirilerek saklanması ve aktarılması gibi işlemlere olanak tanımaktadır (2020g). Tez kapsamında yapılan uygulamada veriler disk üzerinden okunup önişleme yapıldıktan sonra oluşan objeler bu işlemlerin tekrarlanmaması için serileştirilerek saklanmıştır. Bu

serileştirilmiş obje dosyalarının bulunduğu durumda veriler buradan ters-serileştirilme işlemi yapılarak Python objelerine dönüştürülerek kullanılmıştır.

3.1.6. Microsoft Malware Classification Challange veri kümesi

Zararlı yazılım örneklerinin asıl haliyle veya küçük değişikliklerle yayınlandığı toplu bir veri kümesi bulmak zor olmakla beraber, az sayıda da olsa bazı veri kaynaklarından zararlı yazılım örnekleri elde etmek mümkündür. Bunlardan ilki zararlı yazılım dosyalarının görüntü şeklinde temsil edildiği Malimg veri kümesidir (Nataraj 2011). Bunun dışında VX Heaven ve VirusShare gibi web sitelerindeki örneklerden yararlanılabilir (2020h). Ancak bu kaynaklardan elde edilen zararlı yazılım örnekleri makine öğrenmesinde kullanılmadan önce toplanarak bir takım ön işleme ve el ile sınıflandırma işlemlerine tabi tutulmalıdır. Bunun yanında bazı AV firmaları tarafından akademik çalışmalarda kullanılmak üzere virüs veritabanları (Comodo Cloud Security Center vb.) paylaşılmıştır. Ancak, bu tip kaynaklar herkese açık değildir. Bunlar dışında bazı zararlı yazılımlardan çıkarılan statik veya dinamik özelliklerden oluşan veri kümeleri bulunmaktadır.

Tez kapsamında yapılan çalışmalarda Microsoft Malware Classification Challange'a (BIG 2015) ait veriler kullanılmıştır. Kullanılan veri kümesi herkese açık olarak yarışmanın Kaggle sayfasında (https://www.kaggle.com/c/malware-classification/data) paylaşılmaktadır. Bu yönüyle zararlı yazılım araştırmasında kullanılabilecek en erişilebilir veri kümelerindendir. Veri kümesi IDA disassembler çıktılarından ve bayt dizilerinden oluşan 20 binden fazla örneği barındırmaktadır. IDA çıktısı assembly komutlarına ek olarak bölüm, adres ve fonksiyon çağrı bilgisini de içermektedir (2020i). Microsoft araştırmacıları bu zararlı yazılım verilerini dokuz adet zararlı yazılım ailesine gruplamışlardır. İlgili tablo Çizelge 3.1'de görülebilir.

Çizelge 3.1. Veri kümesindeki zararlı yazılım aileleri

Zararlı ismi	Örnek sayısı	Tipi
Ramnit	1541	Solucan
Lollipop	2478	Reklam yazılımı
Kelihos_ver1	398	Botnet
Kelihos_ver3	2942	Botnet
Vundo	475	Truva atı
Tracur	751	Truva atı
Obfuscator.ACY	1228	Karıştırılmış / şifrelenmiş
Simda	42	Arka kapı
Gatak	1013	Truva atı / arka kapı

Çizelge 3.1'de zararlı ismi, örnek sayısı ve tipi ile ilgili bilgiler verilmiştir (Ronen ve ark. 2018). 3.2'de ise 3.1'de verilen zararlı yazılım sınıflandırması tez kapsamında yapılan sınıflandırma cinsinden ifade edilmiştir.

Çizelge 3.2. Tez kapsamındaki sınıflandırmaya göre veri kümesindeki zararlı yazılım aileleri

Zararlı ismi	Tipi
Ramnit	Solucan
Lollipop	Truva atı
Kelihos_ver1	Botnet
Kelihos_ver3	Botnet
Vundo	Truva atı
Tracur	Truva atı
Obfuscator.ACY	Zararlı yazılım tipi değil, virüs geliştirme aracı
Simda	Casus yazılım
Gatak	Truva atı

3.2. Yöntem

Python yazılımının mimarisi oluşturulurken kodun yönetilebilir olması için farklı işlevlerdeki sınıflar ve fonksiyonlar farklı dosyalarda (kütüphanelerde) oluşturulmuştur. Bu kütüphaneler config, io, models, preprocessing ve visualization şeklinde isimledirilmiştir. Bu kütüphaneler utils dizininde saklanarak ana Python dosyasından utils.kutuphane_ismi şeklinde erişim sağlanmaktadır. Bunlardan config dosyasında veriler ile ilgili dizin ve dosya yolları ile beraber diğer ayarlar bulunmaktadır. Diğer kütüphaneler olan io kütüphanesinde dosya işlemleri ile ilgili fonksiyonlar, models kütüphanesinde modeller, preprocessing kütüphanesinde önişleme ile ilgili sınıf ve fonksiyonlar, metrics kütüphanesinde ise görselleştirme ve istatistiksel özelliklerin hesaplanması ile alakalı yardımcı fonksiyonlar bulunmaktadır.

Çizelge 3.3. Program dosyaları ve işlevleri

Dosya ismi	Yolu	İşlevi	Kullanımı
__main__.py	tez/	Ana program dosyası	Yardımcı sınıflar ve fonksiyonlar kullanılarak ana program işlevinin yerine getirilmesi
config.py	tez/utils	Ayar dosyası	Program ayarlarının ve değişkenlerinin saklanması
io.py	tez/utils	Yardımcı kütüphane	Dosya ve klasör işlemleri
models.py	tez/utils	Yardımcı kütüphane	Verilen verilerle eğitilen modellere ait objelerin üretilmesi
preprocessing.py	tez/utils	Yardımcı kütüphane	Önişleme ve özellik mühendisliği fonksiyonlarının ve sınıflarının sağlanması
metrics.py	tez/utils	Yardımcı kütüphane	Ölçüm ve görselleştirme işlemleri

Yazılım mimarisinin bu şekilde yapılandırılmasında birden çok amaç göz önünde bulundurulmuştur. Bunlardan ilki kodun yönetilebilirliği ve okunabilirliğidir. Bu özellikle akademik değer taşıyan tez çalışmasında dikkat edilmesi gereken önemli bir konudur. Kütüphanelerin bu şekilde oluşturulması ve ilgili kodların bir kütüphane altında konumlandırılarak çoğunlukla ana yazılım dosyasında kullanılması bu nedendendir. Bu şekilde __main__.py dosyasını inceleyen bir kişi sınıfların ve fonksiyonların iç işleyişinden bağımsız yapılan işlemleri yorumlayabilmektedir.

İkinci amaç kodun tez çalışması dâhilinde veya başka bir çalışmada yeniden kullanılabilmesidir. Hazırlanan genel amaçlı kütüphaneler sayesinde, örneğin tezdeki görselleştirme arayüzünü kullanmak isteyen bir yazılım geliştirici metrics.py dosyasını elde ederek bundan yararlanabilir. Microsoft Malware Classification Challage verisini kullanacak bir araştırmacı preprocessing.py dosyasını isteklerine göre düzenleyerek önişleme işlemlerinde kullanabilir. Bunun yanında bu yaklaşım çalışmanın önemli amaçlarından biri olan ölçeklenebilirliği de artırmaktadır. Bu şekilde yazılım çalışan bir sisteme kolaylıkla uyumlandırılarak hâlihazırda çalışan sistemlerle beraber kullanılabilir. Sonraki paragraflarda bahsedilen yazılım dosyaları detaylı şekilde açıklanacak ve tez kapsamında yapılan çalışmalar anlatılacaktır.

Ana program dosyası diğer yardımcı kütüphanelerde gerçeklenmiş sınıfların kullanıldığı ve tez kapsamında yapılan çalışmaların genel akışının görülebildiği kısımdır. Bu kapsamda ilk olarak tezde edilen sonuçların tekrar edilebilir olması için NumPy ve random kütüphanelerine sabit tohum değeri verilmiştir. Bu şekilde ilgili kütüphane bağlılıklarına sahip makine öğrenmesi kütüphaneleri veri işlemede ve modellerin parametrelerinin başlatılmasında aynı sözde rastgele değerleri üreteceklerdir. İlgili program kodları aşağıda görülebilir:

```
np.random.seed(1337)
random.seed(1337)
```

Gerekli bağlılıklar sağlandıktan sonra verinin kullanıma uygun hale getirilmesi gerekmektedir. Bu kapsamda preprocessing.py kütüphanesinde bulunan *preprocess()* fonksiyonu kullanılmıştır. Bu fonksiyon iki ana işlevi yerine getirmektedir. Bunlardan ilki daha önce işlenmiş verinin saklanmasını kontrol etmektir. İlk olarak daha önce saklanan bir Pickle dosyası olup olmadığını denetleyen fonksiyon eğer dosyalar mevcut ise bu dosyaları geri serileştirerek tüm veri kümesi için olasılık özellikleri matrisi, dosya özellikleri matrisi ve sınıf vektörünü oluşturur. İkinci işlevi ise bu dosyaların mevcut olmadığı durumdur. Eğer daha önce veriler işlenmemiş ise config.py dosyasında belirlenmiş bayt dosyası dizinindeki dosyalara erişir. Preprocess sınıfını kullanarak buradaki bayt dosyalarını bir boyutlu dizi dosyalarına dönüştürür.

Ardından her bir örnek için bir boyutlu dizi dosyalarını kullanarak BagOfWords sınıf objesi oluşturur. Bu sınıf şu işlemleri gerçekleştirmektedir:

1. *extract()* metodu ile bayt frekans vektörünü oluşturur.
2. *prob()* metodu frekans vektörünü kullanarak bayt olasılık vektörünü oluşturur.
3. *entropy()* metodu olasılık vektörünü kullanarak bayt entropi vektörünü oluşturur.

Bu üç vektör de 256 eleman uzunlukta olup, sınıf değişkenlerinde tutulması hesaplama maliyeti açısından yararlıdır. Yukarıda bahsedilen metotlar ve io.py kütüphanesinde bulunan *class_of_sample(sample)* fonksiyonu her örnek için sırayla işletilerek olasılık matrisi, dosya özellik matrisi ve sınıf vektörü oluşturulur. Bu veriler Pickle kütüphanesi kullanılarak serileştirilir ve diske kaydedilir. Bu şekilde deney tekrarlandığında bu işlem yeniden tekrarlanmamış olur. Fonksiyon bu değerleri dönerek çalışmasını sonlandırır.

Elde edilen veriler Scikit-learn kütüphanesinin *train_test_split(*arrays, **options)* fonksiyonu ile deneylerde kullanılmak üzere parçalara ayrılır. Bu parçalar aşağıda görülmektedir:

- Olasılık eğitim matrisi
- Olasılık test matrisi
- Dosya özellikleri eğitim matrisi
- Dosya özellikleri test matrisi
- Sınıf eğitim vektörü
- Sınıf test vektörü

Bu adımdan sonra tekil modellerin eğitimi için gereken veriler hazırlanmış olur. Buradan sonra bayt olasılık özellikleri için kullanılacak model objelerini üretecek fonksiyonlar sözlük tipinde bir değişkene atanır. Aşağıda bayt olasılık modellerini üreten ve models.py dosyasında bulunan fonksiyonları değer olarak alan sözlük görülmektedir.

models_probability = {'dt': DecisionTree, 'rf': RandomForest, 'knn': KNeighbours, 'ann': ArtificialNeuralNetwork, 'xgb': XGBoost}

Models.py dosyasında bulunan ve farklı sınıflandırıcı objelerini üreten fonksiyonlar aynı tip girdi alacak ve aynı tipte çıktı üretecek şekilde oluşturulmuştur. Bu fonksiyonlar argüman olarak X veri kümesi matrisi ve y sınıf vektörünü alırlar ve çıktı olarak eğitilmiş sınıflandırıcıyı dönerler. XGBoost dışındaki modellerde Scikit-learn kütüphanesinin çapraz doğrulama yardımcısı GridSearchCV kullanılmıştır. Bu vasıtayla farklı model parametreleri ile eğitilen modellerden en başarılı parametre kümesinin seçilmesi sağlanmıştır. Tüm bu aşamalardan geçirildikten sonra fonksiyon çıktısı olarak ilgili sınıflandırıcı objesi üretilmektedir.

Bahsedilen şekilde düzenlenen fonksiyonlar sayesinde ana program dosyasında soyutlama sağlanmıştır. Bu şekilde ilgili model fonksiyonundan dönen *clf* objesi ortak sınıf ve olasılık tahmin metotlarına sahip olmaktadır. Ana program dosyasında *models_probability* sözlüğü üzerinde yapılan döngüde sınıflandırıcı objeleri elde edilerek test verilerinde her sınıf için oluşturulan olasılık sonucu ve nihai sınıflandırma sonucu elde edilir. Bu sonuçlar Results sınıfı kullanarak saklanır. Saklanan sonuçlar ayrıca oylama sınıflandırıcısında kullanılmak üzere *results_probability* sözlüğüne eklenir. İlgili sonuç üzerinde metrics.py tarafından sağlanan *show_metrics(results, y_test, truncate)* ve *show_conf_matrix(results, y_test)* çalıştırılarak ölçütler belirlenir ve hata matrisi oluşturulur. Aynı işlem dosya özellikleri ve ilgili modelleri için de gerçekleştirilir.

Bu aşamada tekil sınıflandırıcılar eğitilerek test sonuçları elde edilmiştir. Sıradaki işlem models.py tarafından sağlanan *VotingClassifier* sınıfında gerçeklenmiş ağırlıklı oylama sınıflandırıcısı sonucunu bulmaktır. *VotingClassifier* sınıfında *addClassifier(results, weight)* metodu ile eklenen sınıflandırıcı sonuçları ve ağırlık değeri *calculateOverallResult()* metodu tarafından hesaplanarak toplu sınıflandırma sonucu oluşturulmaktadır. Bu şekilde test verisi için sınıflandırma sonuçları elde edilmiş olur. EK 4'de gösterilen ana program kodunda, sonraki bölümde bahsedilen optimizasyon fonksiyonu ile elde edilen deneyler sonucu oluşan oylama ağırlık vektörü kullanarak hesaplamalar yapılmıştır. Eğer deney yeniden tekrarlanmak isteniyorsa fonksiyonun uygun parametrelerle yeniden çalıştırılması gerekmektedir.

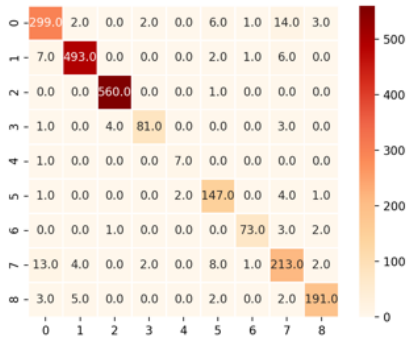
4. BULGULAR ve TARTIŞMA

Yapılan çalışmadaki ilk veri kaynağı, önişleme yapılarak elde edilen 256 elemanlı bayt (00_{16} – FF_{16}) olasılık vektörüdür. Öncelikle dosya içerisindeki bayt frekansının bulunması, ardından da toplam bayt sayısına bölünmesi ile bulunan bayt olasılık vektörü birden çok modelin eğitilmesinde kullanılmıştır. Bayt olasılık vektörlerinin sınıflandırılmasında karar ağacı, rastgele orman, k-nearest neighbour, yapay sinir ağı ve XGBoost modelleri kullanılmıştır. Bu modeller ve elde edilen doğruluk, hassasiyet, geri çağırma, f-skoru aşağıda verilmiştir.

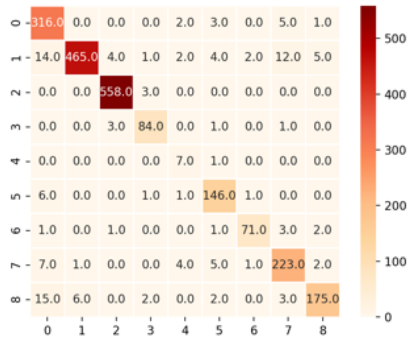
Çizelge 4.1. Bayt olasılık özellikleriyle eğitilen modeller ve sınıflandırma başarımı

Model	Doğruluk	Hassasiyet	Geri çağırma	F-skoru
Karar ağacı	0.95	0.92	0.93	0.92
Rastgele orman	**0.98**	**0.97**	**0.97**	**0.97**
kNN	0.94	0.88	0.92	0.89
YSA	0.93	0.87	0.92	0.89
XGBoost	**0.97**	**0.96**	**0.95**	**0.95**

Çizelge 4.1'de görüldüğü gibi Microsoft Malware Classification Challange veri kümesine ait dokuz sınıftaki (Lollipop, Kelihos_ver1, Kelihos_ver3, Vundo, Tracur, Obfuscator.ACY, Simda, Gatak) zararlı yazılım verilerinde üzerinde eğitilen rastgele orman ve XGBoost modelleri farklı istatistiksel parametrelere göre %95 üzerinde başarım göstererek yüksek doğrulukta sınıflandırma yapabilmişlerdir.

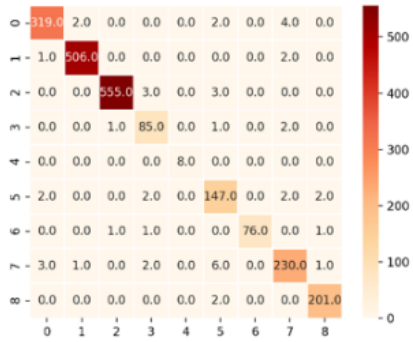


Decision tree

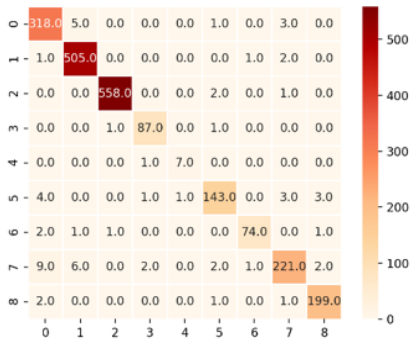


K-en yakın komşu

Yapay sinir ağı



Rastgele orman



XGBoost

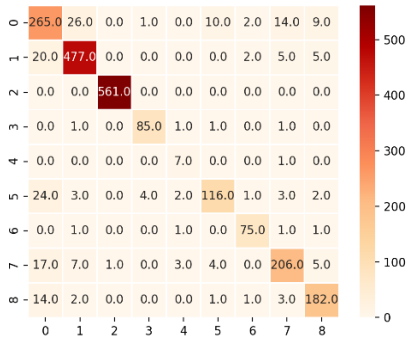
Şekil 4.1. Bayt olasılık özellikleriyle eğitilen modellerin çok sınıflı test verisindeki başarımını gösteren hata matrisleri

Şekil 4.1'de ise farklı makine öğrenmesi yöntemlerine göre oluşturulan modellere ait hata matrisleri görülmektedir. Hata matrislerinde yatayda görülen değerler model tarafından tahmin edilen sınıf değerlerini, dikeyde görülen değerler ise örneğin asıl ait olduğu sınıfı göstermektedir. Köşegen üzerinde olmayan değerlerin daha fazla olması modellerin daha fazla yanlış sınıflandırma yaptığını göstermektedir. Buna göre köşegende bulunan değerlerin doğru sınıflandırma sonucu olduğu görülebilir. Hata matrisi üzerinde de rastgele orman ve XGBoost algoritmalarının yüksek doğrulukla sınıflandırma yaptığı görülebilir.

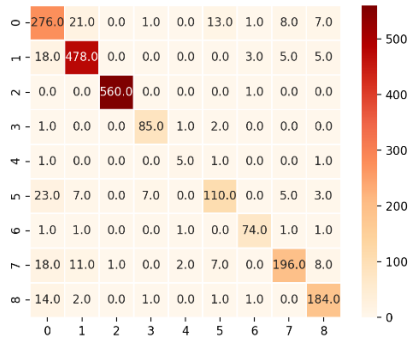
Bir diğer bilgi kaynağı, çalıştırılabilir dosyadaki iki genel özelliği kullanmaktadır. Bunlar dosya boyutu ve entropidir. Bu iki özellik çalıştırılabilir dosyanın barındırdığı bilgiyi göstermektedir. İkinci sınıf sınıflandırıcılar karar ağacı, rastgele orman ve XGBoost bu özellik vektörü kullanılarak eğitilmiştir.

Çizelge 4.2. Entropi ve dosya boyutu özellikleriyle eğitilen modeller ve sınıflandırma başarımı

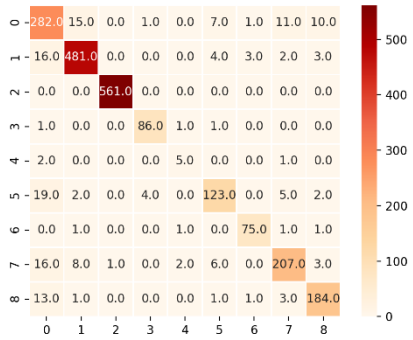
Model	**Doğruluk**	**Hassasiyet**	**Geri çağırma**	**F-skoru**
Karar ağacı	0.91	0.86	0.89	0.87
Rastgele orman	**0.92**	**0.87**	**0.88**	**0.88**
XGBoost	0.91	0.86	0.86	0.86



Decision tree



XGBoost



Rastgele orman

Şekil 4.2. Entropi ve dosya boyutu özellikleriyle eğitilen modellerin çok sınıflı test verisindeki başarımını gösteren hata matrisleri

Yukarıda bahsedilen tekil makine öğrenmesi modellerinin başarımı Çizelge 4.1 ve Çizelge 4.2'de görülmektedir. Tüm modellere ait detaylı başarım değerleri EK 1'de verilmiştir. Görüldüğü gibi sadece iki tane özellik barındırmasına rağmen dosya özellikleri zararlı yazılımların sınıflandırılmasında önemli bir başarı sağlamaktadır.

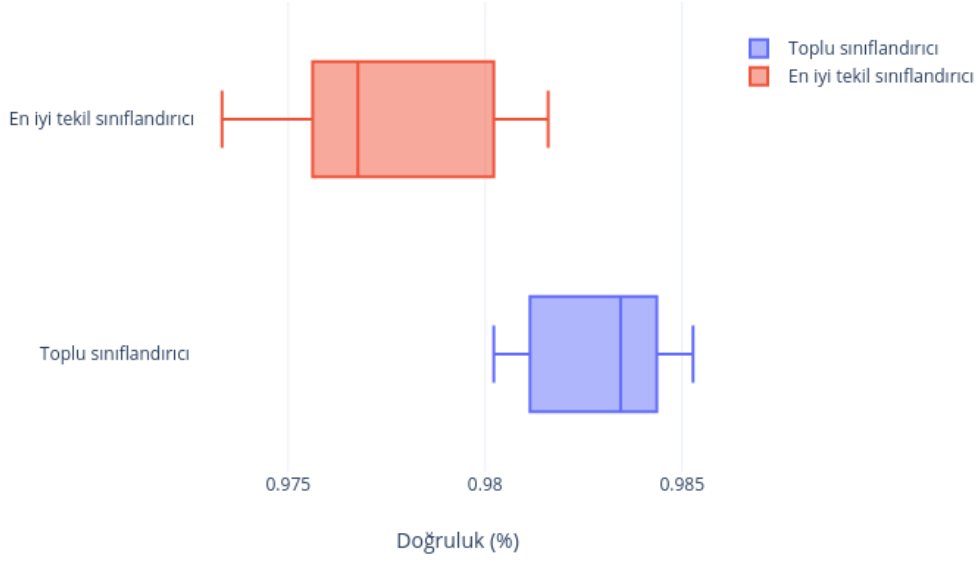
Sonraki adımda ise oluşturulan toplu sınıflandırıcı ile bu tekil modellerin başarımının artırılabileceği gösterilmiştir. Bu toplu sınıflandırıcı oluşturulurken ağırlıklandırılmış bir oylama kullanılmıştır. Bir topluluk sınıflandırma yöntemi olan ağırlıklı oylama yönteminde, farklı modeller tarafından yapılan sınıflandırmalar ağırlıkları göz önünde bulundurarak ilgili sınıf için oy olarak kabul edilmiş ve oy vektöründen en olası sınıf seçilmiştir.

Bu ağırlıkların seçilmesi için iki yol izlenebileceği görülmüştür. Bunlardan ilki modellerin başarım göstergelerine bakılarak el ile ağırlıkların belirlenmesidir. İkincisi ise bu ağırlıkların seçiminin bir optimizasyon problemi olarak değerlendirilmesidir. Tez kapsamında yapılan çalışmada da ikinci yöntem tercih edilmiştir. Bunun en önemli nedeni ilk yöntemle ağırlık seçimi yapıldığında sadece sınıflandırma doğruluğu gibi göstergelere bakılabilir olmasıdır. Sunulan tezde, model.py kütüphanesinin altında *find_optimal_weights(results_probability, results_file_properties, y_test, size)* fonksiyonu gerçeklenmiştir. Bu fonksiyon optimal ağırlıkların bulunması için ağırlık uzayında rastgele arama yapmakta kullanılmıştır. Bu yolla elde edilen ve testlerde kullanılan ağırlıklar Çizelge 4.3'de gösterilmiştir. Yapılan denemeler sonucunda görülmüştür ki gerçekten de dosya özellikleri ile eğitilen XGBoost modelinin oy etkisi rastgele orman modelinden daha yüksektir. Benzer durum bayt olasılık özellikleri ile eğitilen modellerde de görülmüştür.

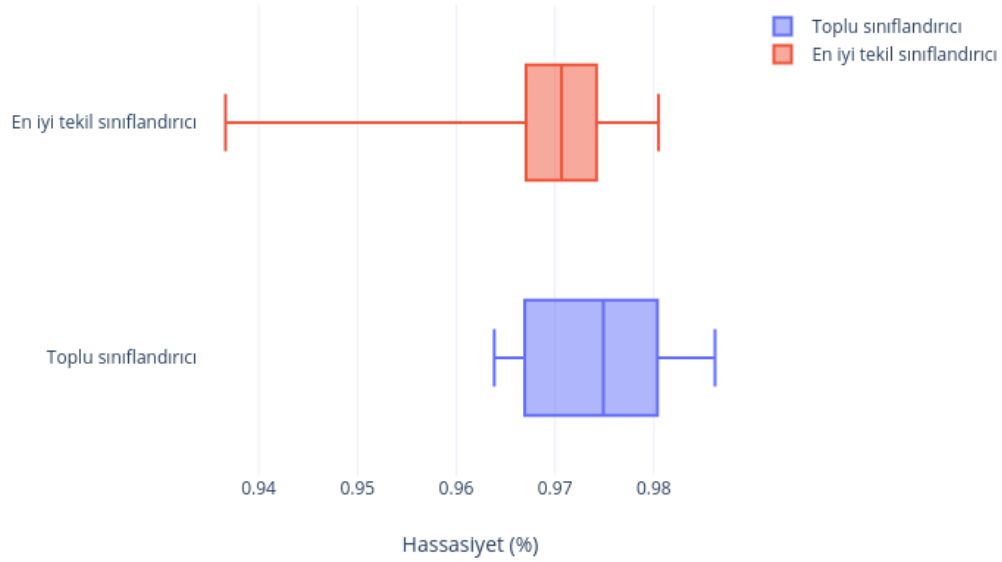
Çizelge 4.3. Farklı sınıflandırıcılar için seçilen oylama ağırlıkları

Bayt olasılık özellikleri				
Karar ağacı	Rastgele orman	kNN	YSA	XGBoost
0.32255784	0.98224021	0.21065342	0.57632579	0.9836862
Dosya özellikleri				
Karar ağacı		Rastgele orman		XGBoost
0.22180763		0.34818698		0.45107591

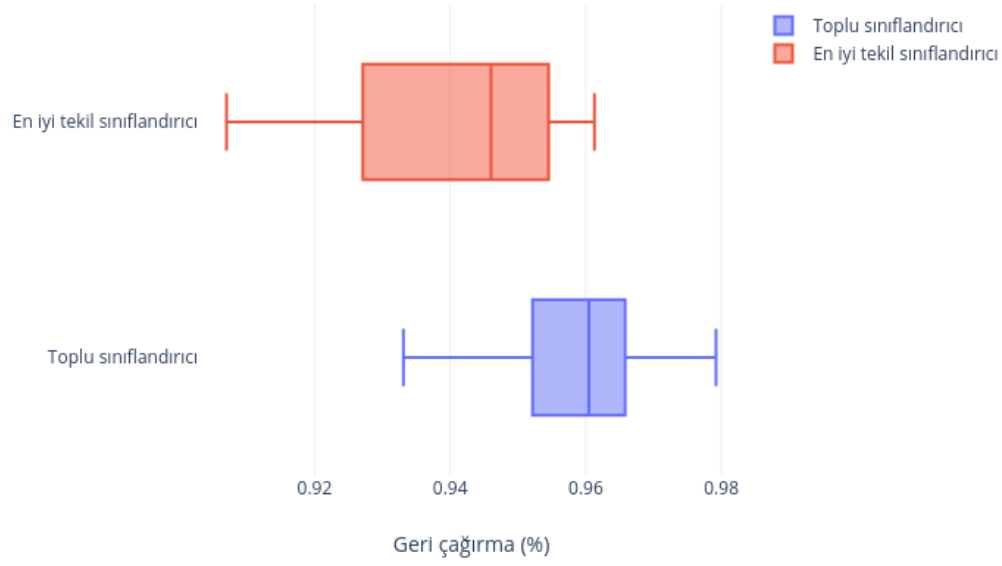
Çizelge 4.3'de görülen ağırlık vektörü, 5000 tane ağırlık vektörü kullanılarak yapılan denemede elde edilmiştir. Bu denemeler farklı modellerin test verisi üzerinde elde ettiği çıktıların rassal olarak seçilen ağırlık vektörleri sonucunda göstereceği toplu başarım hesaplanarak seçilmiştir. Ancak makine öğrenmesinin temel problemlerinden olan eldeki verilerin tüm girdi uzayını temsil yeteneği ölçülmelidir. Bu amaçla da seçilen ağırlık vektörü veri kümesinden farklı alt kümeler seçilerek yeniden oluşturulan eğitim ve test verileri üzerinde denetlenmiştir. Bu denemelerde elde edilen en yüksek doğruluk değerine sahip sınıflandırıcı ve toplu sınıflandırıcıların doğruluk değerleri Şekil 4.3'de, hassasiyetleri Şekil 4.4'de, geri çağırma değerleri Şekil 4.5'de ve f-skoru Şekil 4.6'da görülmektedir.



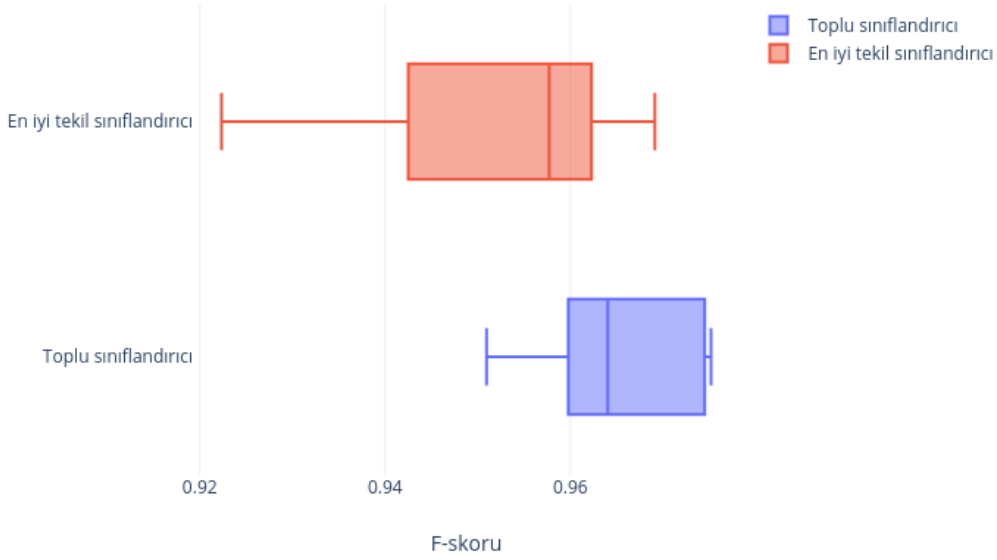
Şekil 4.3. Yapılan deneylerdeki toplu ve en iyi tekil sınıflandırma doğrulukları



Şekil 4.4. Yapılan deneylerdeki toplu ve en iyi tekil sınıflandırma hassasiyetleri



Şekil 4.5. Yapılan deneylerdeki toplu ve en iyi tekil sınıflandırma geri çağırma oranları



Şekil 4.6. Yapılan deneylerdeki toplu ve en iyi tekil sınıflandırma f-skorları

Deneyler için elde edilen başarım ölçümlerin sayısal detayları EK 2'de gösterilmiştir. Sonuçlar incelendiğinde, toplu sınıflandırıcının farklı veri dağılımları için yapılan denemelerde belirgin bir şekilde daha başarılı olduğu görülmektedir.

5. SONUÇ

Makine öğrenmesi alanında yapılan güncel araştırmalarda gelişen hesaplama yeteneği sayesinde daha karmaşık modeller kullanılabilmektedir. Ancak üretilen bu karmaşık modeller her zaman beklenen başarıyı gösterememektedir. Ayrıca karmaşıklaşan modellere eklenen hesaplama yükü her zaman başarımı artırmamakta ve modelin yorumlanmasını zorlaştırmaktadır. Bu nedenle modellerin elden geldiği kadar sade ve işlevsel olmasına özen gösterilmelidir. Bunun yanında öğrenilmek istenen verinin yapısı göz önünde bulundurularak bu kaynaklardan elde edilecek verilerin sınıflandırılmasında kullanılacak makine öğrenmesi modelinin sahip olması gereken özelliklerin ne olması gerektiği değerlendirilmelidir. Bu nedenle popüler modellerin rastgele farklı gerçek yaşam uygulamaları üzerinde uygulanması ne kaliteli bilimsel bilgi üretimini sağlamakta, ne de uygulanabilir mühendislik ürünleri ortaya çıkarmaktadır. Bu alanda ticari ürün sayısı sınırlı olmakla beraber model karmaşıklığının kullanılabilirliğe etkisi üzerine etkilerini görebileceğimiz bir uygulama bulunmaktadır.

Sonuç olarak tez kapsamında yapılan çalışmada farklı özellik kümelerini kullanarak üretilen modellerin beraber kullanılması şeklinde oluşturulan hibrit modelin, hem farklı makine öğrenmesi yöntemleri hem de farklı özellik kümeleri kullanan modelleri kullanarak daha başarılı olduğu görülmüştür. Tekil makine öğrenmesi modelleri bayt olasılık değerlerinden oluşan olasılık özellikleri ve dosya boyutu – entropi değerinden oluşan dosya özellikleri kullanılarak eğitilmiştir. Bu şekilde oluşturulan tekil modeller optimum ağırlık vektörü ile ağırlıklandırılarak toplu oylama modelini oluşturmuşlardır. EK 2'de sonucu verilen deneylerde, kullanılan hibrit modelin %98'in üzerinde doğruluk gösterdiği ve bu sonucun en iyi tekil makine öğrenmesi modelinden iyi olduğu görülmüştür. Bu şekilde tüm deneylerde en iyi tekil sınıflandırıcıdan daha başarılı olmuştur. Bunun yanında seçilen az sayıda kolay çıkarılabilir özelliğin kullanılması bu yaklaşımın önemini göstermektedir. Tezde oluşturulan modele benzer şekilde gerçek sistemlerde kullanılmak üzere bir hibrit model oluşturulabilir. Bunun için büyük veri kümeleri ile burada izlenen yol izlenmeli, özellik mühendisliği kullanılarak örnekleri ifade edecek daha detaylı özellikler seçilmelidir. Seçilen özellik gruplarının birbirini tamamlayıcı ve daha yüksek kavramları ifade eden modeller olması önemlidir. Tez

çalışmasında entropi ve dosya boyutu bilgisine sahip olan model örneğin paketlenmiş olup olmadığını denetlemektedir. Bu şekilde topluluk modeli elde ettiği genel kavramlardan toplam bir sınıflandırma sonucu elde edecektir.

KAYNAKLAR

Abou-Assaleh, T., Cercone, N., Keselj, V., Sweidan, R. 2004. Detection of new malicious code using n-grams signatures. 28th Annual International Computer Software and Applications Conference, 28-30 September, 2004, Hong Kong, China.

Altman, N. S. 1992. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3): 175–185.

Anonim, 2019. Hack at all costs: Putting a price on APT attacks. https://www.ptsecurity.com/upload/corporate/ww-en/analytics/APT-Attacks-eng.pdf (Erişim tarihi: 01.03.2020)

Anonim, 2020a. TRAPMINE Integrates Machine Learning Engine into VirusTotal. https://trapmine.com/blog/trapmine-machine-learning-virustotal (Erişim tarihi: 15.06.2020)

Anonim, 2020b. The Python programming language. https://github.com/python/cpython (Erişim tarihi: 11.04.2020).

Anonim, 2020c. NumPy v1.19.dev0 manual. https://numpy.org/devdocs (Erişim tarihi: 12.04.2020).

Anonim, 2020d. API reference. https://scikit-learn.org/stable/modules/classes.html (Erişim tarihi: 12.04.2020).

Anonim, 2020e. Kaggle: Your Machine Learning and Data Science Community, https://www.kaggle.com (Erişim tarihi: 08.07.2020).

Anonim, 2020f. Python API Reference. https://xgboost.readthedocs.io/en/latest/python/python_api.html (Erişim tarihi: 18.04.2020).

Anonim, 2020g. Python object serialization. https://docs.python.org/3/library/pickle.html (Erişim tarihi: 12.04.2020).

Anonim, 2020h. VirusShare.com - Because Sharing is Caring. https://virusshare.com (Erişim tarihi: 04.07.2020).

Anonim, 2020i. Hex-rays decompiler – user manual. https://www.hex-rays.com/products/decompiler/manual/index.shtml (Erişim tarihi: 08.05.2020).

Breiman, L. 2001. Random forests. *Machine Language,* 45(1): 5-32.

Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J. 2008. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware.

IEEE International Conference on Dependable Systems and Networks with FTCS and DCC (DSN), Alaska, United States.

Chen, T., Guestrin, C. 2016. XGBoost: A scalable tree boosting system. 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, California, United States.

Chess, D., White, S. 2000. An Undetectable Computer Virus. Virus Bulletin Conference.

Cohen, F. 1987. Computer viruses: Theory and experiments. *Computers & Security,* 6: 22-35.

Dolan, S. 2013. Mov is Turing-complete.

Ferrie, P. 2007. Attacks on more virtual machine emulator. *Symantec Technology Exchange,* 55.

Fu, J., Xue, J., Wang, Y., Liu, Z., Shan, C. 2018. Malware visualization for fine-grained classification. *IEEE Access,* 6: 14510-14523

Friedman, J. 2001. Greedy function approximation: a gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232.

Han, K.S., Lim, J.H., Kang, B., Im, E.G. 2015. Malware anaylsis using visualized images and entropy graphs. *International Journal of Information Security,* 14(1): 1-14.

Hastie, T., Tibshirani, R. and Friedman, J. 2017. The elements of statistical learning. Springer, New York, United States.

Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. *A Field Guide to Dynamical Recurrent Networks* (pp.237-243). Wiley-IEEE Press.

Kancherla, K., Mukkamala, S. 2013. Image visualization based malware detection. IEEE Symposium on Computational Intelligence in Cyber Security (CICS).

Kephart, J.O., Arnold, W.C. 1994. Automatic extraction of computer virus signatures. The 4th Virus Bulletin International Conference, Abingdon, England.

Kephart, J.O., Sorkin, G.B., Arnold W.C., Chess, D.M., Tesauro G.J., White, S.R. 1995. Biologically inspired defenses against computer viruses. The 14th International Joint Conference on Artificial Intelligence, San Francisco, California, United States.

King, J.C. 1976. Symbolic execution and program testing. *Communications of the ACM,* 19(7): 385-394.

Kolter, J.Z., Maloof, M.A. 2004. Learning to detect malicious executables in the wild. 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, August, 2004, Seattle, Washington, United States.

Kolter, J.Z., Maloof, M.A. 2006. Learning to detect and classify malicious executables in the wild. *The Journal of Machine Learning Research,* 7: 2721–2744.

Kwong, A., Genkin, D., Gruss, D., Yarom, Y. 2020. RAMBleed: Reading bits in memory without accessing them. 41st IEEE Symposium on Security and Privacy (S & P), 2020.

Le, Q., Boydell O., Namee, B.M., Scanlon, M. 2018. Deep learning at the shallow end: Malware classification for non-domain experts. *Digital Investigation,* 26: 118-126.

Liu, L., Wang, B., Yu, B., Zhong, Q. 2017. Automatic malware classification and new malware detection using machine learning. *Frontiers of Information Technology & Electronic Engineering,* 18(9): 1336-1347.

Masud, M.M., Khan, L., Thuraisingham, B. 2008. A scalable multi-level feature extraction technique to detect malicious executables. *Information Systems Frontiers,* 10(1): 33–45.

Menahem, E., Shabtai, A., Rokach, L., Elovici, Y. 2009. Improving malware detection by applying multi-inducer ensemble. *Computational Statistics & Data Analysis,* 53(6): 1483–1494.

Mikolov, T., Chen, K., Corrado, G., Dean, J. 2013. Efficient estimation of word representations in vector space. *ArXiv e-prints.*

Narayanan, B.N., Djaneye-Boundjou, O., Kebede, T.M. 2016. Performance analysis of machine learning and pattern recognition algorithms for Malware classification. IEEE National Aerospace and Electronics Conference (NAECON) and Ohio Innovation Summit (OIS).

Nash, T. 2005. An undirected attack against critical infrastructure. US-CERT Control Systems Security Center.

Nataraj, L., Karthikeyan, S., Jacob, G., Manjunath, B.S. 2011. Malware images: Visualization and automatic classification. The 8th International Symposium on Visualization for Cyber Security (VizSec '11), United States.

Raff, E., Zak, R., Cox, R., Sylvester, J., Yacci, P., Ward, R., Tracy, A., McLean, M., Nicholas, C. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques,* 14(1): 1-20.

Rathore, H., Agarwal, S., Sahay, S.K., Sewak, M. 2019. Malware detection using machine learning and deep learning. *ArXiv e-prints.*

Özdemir, S. 2014. A decision tree based intrusion detection system with bootstrap aggregating, discretization, and feature selection. *Master Thesis*, BU Institute for Graduate Studies in Science and Engineering, Department of Electrical and Electronics Engineering, İstanbul.

Raiu, Costin. 2018. Where are all the 'A's in APT?. https://www.virusbulletin.com/blog/2018/09/where-are-all-apt (Erişim tarihi: 01.03.2020)

Rokach, L., Maimon, O. 2005. Data Mining and Knowledge Discovery Handbook: Springer US, Ed: Rokach, L., Maimon, O., Boston, MA, United States, pp: 165-192.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E., Ahmadi, M. 2018. Microsoft Malware Classification Challenge. *ArXiv e-prints.*

Schultz, M.G., Eskin, E., Zadok, F., Stolfo, S.J. 2001. Data mining methods for detection of new malicious executables. IEEE Symposium on Security and Privacy, 14-16 May, 2000, Oakland, California, United States.

Shannon, C.E. 1948. A mathematical theory of communication. *The Bell System Technical Journal,* 27(3): 379-423.

Symantec Corporation. 2019. Internet Security Threat Report.

Szefer, J. 2019. Survey of microarchitectural side and covert channels, attacks, and defenses. *Journal of Hardware and Systems Security*, 3(3): 219-234.

Tabish, S.M., Shafiq, M.Z., Farooq, M. 2009. Malware detection using statistical analysis of byte-level file content. The ACM SIGKDD Workshop on CyberSecurity and Intelligence Informatics, June, 2009, Paris, France.

Tesauro, G.J., Kephart, J.O., Sorkin, G.B., 1996. Neural networks for computer virus recognition. *IEEE Expert,* 11(4): 5-6.

Turing, A. 1937. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2): 230-265.

You, I., Yim, K. 2010. Malware obfuscation techniques: A brief survey. International Conference on Broadband, Wireless Computing, Communication and Applications, November, 2010.

Wang, Z., Lee, R.B. 2006. Covert and side channels due to processor architecture. 22nd Annual Computer Security Applications Conference (ACSAC'06), 11-15 December, 2006, Miami Beach, Florida, United States.

Zhang, J., Qin, Z., Yin, H., Ou, L., Xiao, S., Hu, Y. Malware variant detection using opcode image recognition with small training set. 25th International Conference on Computer Communication and Networks (ICCCN).

EKLER

EK 1 Tekil modellerin başarımını gösteren detaylı istatistiksel özellikler

EK 2 Verinin rastgele bölünmesiyle yapılan deneylerde alınan sonuçlar

EK 3 ReLU ve softmax aktivasyon fonksiyonlarının gösterimleri için kullanılan kodlar

EK 4 Tez kapsamında geliştirilen yazılım kodları

EK 1

	Model	Doğruluk	Hassasiyet	Geri çağırma	F-skoru
Olasılık özellikleri	Karar ağacı	0.9494020239190433	0.9216999149379927	0.9284595968261071	0.9245314019575509
	Rastgele orman	0.9783808647654094	0.9708774443038622	0.9734516116969772	0.9720271083213715
	kNN	0.9406623735050598	0.8781096802657523	0.9237581754890232	0.8920871250109182
	YSA	0.9319227230910764	0.8722801647285727	0.9426245290509156	0.8984844715905338
	XGBoost	0.9714811407543699	0.9561277194518121	0.9512054563728173	0.9534901238689655
Dosya özellikleri	Karar ağacı	0.9080036798528058	0.8579911064943265	0.8910697665527282	0.8681737794144564
	Rastgele orman	0.921803127874885	0.873756175722453	0.8777598779102874	0.8751651712495723
	XGBoost	0.9052437902483901	0.8554729140351439	0.8578645773511345	0.8553993832470617

EK 2

Deney No	Model tipi	Doğruluk	Hassasiyet	Geri çağırma	F-skoru
1	Tekil en iyi	0.97884	0.96926	0.95411	0.95984
	Toplu	0.98436	0.97929	0.97155	0.97493
2	Tekil en iyi	0.98022	0.97420	0.95451	0.96304
	Toplu	0.98390	0.97049	0.95991	0.96480
3	Tekil en iyi	0.97562	0.97054	0.93358	0.94770
	Toplu	0.98114	0.97955	0.95214	0.96429
4	Tekil en iyi	0.97608	0.97357	0.95291	0.96228
	Toplu	0.98528	0.98620	0.96580	0.97518
5	Tekil en iyi	0.97332	0.93664	0.91607	0.92296
	Toplu	0.98022	0.96385	0.95636	0.95977
6	Tekil en iyi	0.97470	0.96707	0.90700	0.92234
	Toplu	0.98114	0.98037	0.93309	0.95095
7	Tekil en iyi	0.97746	0.95929	0.95999	0.95952
	Toplu	0.98068	0.96500	0.96278	0.96369
8	Tekil en iyi	0.98022	0.97785	0.96127	0.96912
	Toplu	0.98344	0.97040	0.97921	0.97450
9	Tekil en iyi	0.97562	0.97078	0.92711	0.94250
	Toplu	0.98344	0.98051	0.94378	0.95835
10	Tekil en iyi	0.98160	0.98048	0.93912	0.95591
	Toplu	0.98528	0.96693	0.96099	0.96378

EK 3

draw.py

```
# Rectified linear function
def rectified(x):
    return max(0.0, x)


def rectified_multiple(X):
    return list(map(rectified, X))


# Softmax function
from numpy import exp, sum

def softmax_multiple(X):
    return exp(X) / sum(exp(X), axis=0)


# Example data points
from numpy import random

X1 = random.uniform(low=-2, high=2, size=(10000,))
rectified_y = rectified_multiple(X1)

X2 = random.uniform(low=-2, high=5, size=(10000,))
softmax_y = softmax_multiple(X2)

# Draw functions
import matplotlib.pyplot as plt

fig, (ax1, ax2) = plt.subplots(2)
plt.subplots_adjust(hspace=0.5)

ax1.plot(X1, rectified_y, 'g.')
ax1.axhline(0, color='black')
ax1.axvline(0, color='black')
ax1.set_title('ReLU')
ax1.axis('off')

ax2.plot(X2, softmax_y, 'b.')
ax2.axhline(0, color='black')
ax2.axvline(0, color='black')
ax2.set_title('Softmax')
ax2.axis('off')

plt.show()
```

EK 4

```
__main__.py
-----------

import random

import numpy as np
from sklearn.model_selection import train_test_split

import utils.config as config
from utils.metrics import show_metrics, show_conf_matrix, dtree_viz
from utils.models import DecisionTree, RandomForest, KNeighbours,
ArtificialNeuralNetwork, XGBoost, Results, VotingClassifier
from utils.preprocessing import preprocess

# Eliminate randomness
np.random.seed(1337)
random.seed(1337)

# Take sequence file as a BoW of bytes and make a probability
distribution from that
X1, X2, y = preprocess()

# Cast lists to numpy array
X1 = np.array(X1)
X2 = np.array(X2)
y = np.array(y)

# Split train and test data
X1_train, X1_test, X2_train, X2_test, y_train, y_test =
train_test_split(X1, X2, y,test_size=config.TEST_SIZE)

# Make models and train
models_probability = {'dt': DecisionTree, 'rf': RandomForest, 'knn':
KNeighbours, 'ann': ArtificialNeuralNetwork,'xgb': XGBoost}
results_probability = {}

for name, model in models_probability.items():
    clf = model(X1_train, y_train)
    y_pred = clf.predict(X1_test)
    proba = clf.predict_proba(X1_test)
    results = Results(model.__name__, y_pred, proba)

    # Evaluate results
    show_metrics(results, y_test, truncate=False)
    show_conf_matrix(results, y_test)

    # Save for voting classifier
    results_probability.update({name: results})

    # Visualize decision tree
    if name == 'dt':
        dtree_viz(clf.best_estimator_, 'prob_dt.png')
```

```
models_file_properties = {'dt': DecisionTree, 'rf': RandomForest,
'xgb': XGBoost}
results_file_properties = {}

for name, model in models_file_properties.items():
    clf = model(X2_train, y_train)
    y_pred = clf.predict(X2_test)
    proba = clf.predict_proba(X2_test)
    results = Results(model.__name__, y_pred, proba)

    show_metrics(results, y_test, truncate=False)
    show_conf_matrix(results, y_test)

    results_file_properties.update({name: results})

    if name == 'dt':
        dtree_viz(clf.best_estimator_, 'file_dt.png')

weights = [0.32255784, 0.98224021, 0.21065342, 0.57632579, 0.9836862,
0.22180763, 0.34818698, 0.45107591]

votingClassifier = VotingClassifier()
votingClassifier.addClassifier(results_probability['dt'], weights[0])
votingClassifier.addClassifier(results_probability['rf'], weights[1])
votingClassifier.addClassifier(results_probability['knn'], weights[2])
votingClassifier.addClassifier(results_probability['ann'], weights[3])
votingClassifier.addClassifier(results_probability['xgb'], weights[4])
votingClassifier.addClassifier(results_file_properties['dt'],
weights[5])
votingClassifier.addClassifier(results_file_properties['rf'],
weights[6])
votingClassifier.addClassifier(results_file_properties['xgb'],
weights[7])
clf = votingClassifier.calculateOverallResult()

accuracy = show_metrics(clf, y_test)
```

```
models.py
---------

from dataclasses import dataclass

import numpy as np
from sklearn.model_selection import GridSearchCV

from utils.config import CLASS_COUNT
from utils.metrics import show_metrics


# Model construction functions
def DecisionTree(X, y):
    from sklearn.tree import DecisionTreeClassifier
    clf = DecisionTreeClassifier()
    params_dt = {'max_depth': np.arange(1, 21), 'min_samples_leaf':
[1, 5, 10, 20, 50, 100]}
    clf = GridSearchCV(clf, params_dt, cv=5)
    clf = clf.fit(X, y)
    return clf


def RandomForest(X, y):
    from sklearn.ensemble import RandomForestClassifier
    clf = RandomForestClassifier()
    params_rf = {'n_estimators': [50, 100, 200]}
    clf = GridSearchCV(clf, params_rf, cv=5)
    clf.fit(X, y)
    return clf


def KNeighbours(X, y):
    from sklearn.neighbors import KNeighborsClassifier
    clf = KNeighborsClassifier()
    params_knn = {'n_neighbors': np.arange(1, 26)}
    clf = GridSearchCV(clf, params_knn, cv=5)
    clf.fit(X, y)
    return clf


def ArtificialNeuralNetwork(X, y):
    from sklearn.neural_network import MLPClassifier
    clf = MLPClassifier(solver='adam', learning_rate_init=0.01,
max_iter=500)
    params_ann = {'hidden_layer_sizes': [(50, 25, 25), (50, 25),
(25)]}
    clf = GridSearchCV(clf, params_ann, cv=5)
    clf.fit(X, y)
    return clf


def XGBoost(X, y):
    import xgboost as xgb
    clf = xgb.XGBClassifier(random_state=1, learning_rate=0.01)
    clf.fit(X, y)
    return clf
```

```
class Results:
    def __init__(self, name, results, probabilities=0):
        self.__name__ = name
        self.__results__ = results
        self.__proba__ = probabilities
        self.__weight__ = 0


class VotingClassifier:
    def __init__(self):
        self.testCount = 0
        self.classifiers = []
        self.weights = []

    def addClassifier(self, results, weight=1):
        # Get test sample count from first model
        if self.classifiers.__len__() == 0:
            self.testCount = len(results.__results__)

        results.__weight__ = weight
        self.classifiers.append(results)

    def calculateOverallResult(self):
        voteMatrix = []

        for i in range(self.testCount):
            classVotes = np.zeros(CLASS_COUNT)
            for clf in self.classifiers:
                classVotes += clf.__proba__[i] * clf.__weight__

            voteMatrix.append(classVotes)

        overallResult = np.argmax(voteMatrix, axis=1) + 1
        return Results('VotingClassifier', overallResult)


def find_optimal_weights(results_probability, results_file_properties,
y_test, size=5000):
    weight_matrix = np.random.random(size=(size, 8))

    @dataclass
    class weightedClassifier:
        accuracy: float = 0.0
        weights: np.ndarray = np.array([])

    best = weightedClassifier()
    for weights in weight_matrix:
        votingClassifier = VotingClassifier()
        votingClassifier.addClassifier(results_probability['dt'],
weights[0])
        votingClassifier.addClassifier(results_probability['rf'],
weights[1])
        votingClassifier.addClassifier(results_probability['knn'],
weights[2])
        votingClassifier.addClassifier(results_probability['ann'],
weights[3])
```

```
        votingClassifier.addClassifier(results_probability['xgb'],
weights[4])
        votingClassifier.addClassifier(results_file_properties['dt'],
weights[5])
        votingClassifier.addClassifier(results_file_properties['rf'],
weights[6])
        votingClassifier.addClassifier(results_file_properties['xgb'],
weights[7])
        clf = votingClassifier.calculateOverallResult()

        accuracy = show_metrics(clf, y_test)
        if accuracy > best.accuracy:
            best.accuracy = accuracy
            best.weights = weights

    return best
```

metrics.py

```
from io import StringIO

import matplotlib.pyplot as plt
import pydotplus
import seaborn
import os.path
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_recall_fscore_support
from sklearn.tree import export_graphviz

import utils.config as config


def show_metrics(results, y_test, truncate=False):
    if truncate:
        print("{} accuracy: {:.2f}, precision: {:.2f}, recall: {:.2f},
f-score: {:.2f}".format(results.__name__, accuracy_score(y_test,
results.__results__),
 *precision_recall_fscore_support(y_test, results.__results__,
average='macro')))
    else:
        print("{} accuracy: {}, precision: {}, recall: {}, f-score:
{}".format( results.__name__, accuracy_score(y_test,
results.__results__), *precision_recall_fscore_support(y_test,
results.__results__, average='macro')))
    return accuracy_score(y_test, results.__results__)


# Plot confusion matrix
class modelCount:
    count = 1

def show_conf_matrix(results, y_test):
    if not config.SHOW_CONFUSION_MATRIX:
        return

    plt.figure()
    cm = confusion_matrix(y_test, results.__results__)
    hm = seaborn.heatmap(cm, annot=True, fmt=".1f", linewidths=1.0,
square=1, cmap="OrRd")
    fig = hm.get_figure()
    fig.savefig("{}_{}".format(results.__name__, modelCount.count))
    modelCount.count += 1

# Save decision tree as a PNG
def dtree_viz(model, filename):
    dot_data = StringIO()
    export_graphviz(model, out_file=dot_data, filled=True,
rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    graph.write_png(filename)
```

config.py

```
# Model parameters
TEST_SIZE = 0.2

# Metrics
SHOW_CONFUSION_MATRIX = True

# Files and directories for data
BYTES_DIR = './bytes'
LABEL_FILE = './veri/trainLabels.csv'
PICKLE_X1 = './bytes/data_bow.pickle'
PICKLE_X2 = './bytes/data_file.pickle'
PICKLE_y = './bytes/labels_bow.pickle'

# Data properties
CLASSES = range(1, 10)
CLASS_COUNT = len(CLASSES)

# Filesystem access constants
SEPERATOR = '/'
SEQUENCE_FILE_EXTENSION = 'seq'
BYTE_FILE_EXTENSION = 'bytes'
```

```
preprocessing.py
----------------

"""
    This file preprocesses Microsoft Malware Classification Challange
data for machine learning models and further examination
"""

import csv
import os
import pickle

import numpy as np
from scipy.stats import entropy

import utils.config as config
from utils.io import concat_file_path, get_file_size
from utils.io import is_sequence_file, class_of_sample


class Preprocess:
    def __init__(self, bytes_dir, label_file):
        self.bytes_dir = bytes_dir
        self.label_file = label_file

    def bytes_files_list(self):
        return os.listdir(self.bytes_dir)

    """
        Strips .bytes files from binary addresses and lines to make
byte sequence
        Input file has 8 characters of address in each line, line[9:]
takes remaining characters
    """
    def make_1d_vector(self, file, file_class):
        new_file = '{}{}{}_{}.{}'.format(self.bytes_dir,
config.SEPERATOR, file_class, file.split('.')[0],
config.SEQUENCE_FILE_EXTENSION)

        # Skip if file already exists
        if os.path.isfile(new_file):
            return

        n_file = open(new_file, 'w')
        with open('{}{}{}'.format(self.bytes_dir, config.SEPERATOR,
file)) as o_file:
            for line in o_file:
                n_file.write(line[9:].rstrip())
                n_file.write(' ')

        n_file.close()

    # Makes sequence files using make_1d_vector(*.bytes)
    def make_sequence_files(self, delete_bytes_files):
        with open(self.label_file, 'r') as labels:
            reader = csv.reader(labels)
            class_dict = {rows[0]: rows[1] for rows in reader}
```

```
        for file in self.bytes_files_list():
            name, extension = file.split('.')

            if extension == config.BYTE_FILE_EXTENSION:
                self.make_1d_vector(file, class_dict[name])
                if delete_bytes_files:
                    os.remove(concat_file_path(self.bytes_dir, file))


# BoW feature extraction for byte files
class BagOfWords:
    def __init__(self, bytes_dir, file):
        self.file = concat_file_path(bytes_dir, file)
        self.byte_frequency = [0] * 256
        self.probability = []
        self.extracted = False

    def extract(self):
        with open(self.file, 'r') as f:
            for byte in f.readline().split(' '):
                try:
                    ix = int(byte, 16)
                    self.byte_frequency[ix] += 1
                except:
                    pass

    def prob(self):
        if not self.extracted:
            self.extract()

        byte_count = sum(self.byte_frequency)
        if byte_count == 0:
            self.probability = self.byte_frequency
        else:
            self.probability = [i / byte_count for i in
self.byte_frequency]

        return self.probability

    def entropy(self):
        return np.nan_to_num(entropy(self.probability, base=2 ** 8))


# Preprocessing function
def preprocess():
    if os.path.isfile(config.PICKLE_X1) and
os.path.isfile(config.PICKLE_X2) and os.path.isfile(config.PICKLE_y):
        # If pickle files exist already
        with open(config.PICKLE_X1, 'rb') as f:
            X1 = pickle.load(f)

        with open(config.PICKLE_X2, 'rb') as f:
            X2 = pickle.load(f)

        with open(config.PICKLE_y, 'rb') as f:
            y = pickle.load(f)

    else:
```

```
        # If there are no pickle files
        p = Preprocess(config.BYTES_DIR, config.LABEL_FILE)
        p.make_sequence_files(delete_bytes_files=True)

        X1 = []
        X2 = []
        y = []
        for sample in os.listdir(config.BYTES_DIR):
            if is_sequence_file(sample):
                # Extract byte probability distribution and add to
samples
                bow = BagOfWords(config.BYTES_DIR, sample)
                X1.append(bow.prob())
                # Append class to class vector
                y.append(class_of_sample(sample))

                # Make entropy-file size vector and add to data matrix
                X2.append([bow.entropy(),
get_file_size(concat_file_path(config.BYTES_DIR, sample))])

        # Make pickle files
        with open(config.PICKLE_X1, 'wb') as f:
            pickle.dump(X1, f)

        with open(config.PICKLE_X2, 'wb') as f:
            pickle.dump(X2, f)

        with open(config.PICKLE_y, 'wb') as f:
            pickle.dump(y, f)

    return X1, X2, y
```

io.py

```
import os

import utils.config as config


def is_sequence_file(f):
    return f.endswith(config.SEQUENCE_FILE_EXTENSION)


def class_of_sample(f_without_path):
    return int(f_without_path[0])


def concat_file_path(dir_path, file_name):
    return "{}{}{}".format(dir_path, config.SEPERATOR, file_name)


def get_file_size(f):
    return os.stat(f).st_size
```

ÖZGEÇMİŞ

Adı Soyadı	: Kerim Can KALIPCIOĞLU
Doğum Yeri ve Tarihi	: Osmangazi / Bursa, 1994
Yabancı Dil	: İngilizce
Eğitim Durumu	
Lise	: Bursa Anadolu Lisesi
Lisans	: Yıldız Teknik Üniversitesi
Yüksek Lisans	: Bursa Uludağ Üniversitesi
Çalıştığı Kurum/Kurumlar	: T.C. Eskişehir Osmangazi Üniversitesi TÜBİTAK Bilişim ve Bilgi Güvenliği İleri Teknolojiler Araştırma Merkezi
İletişim (e-posta)	: 501731001@ogr.uludag.edu.tr
Yayınları	:

Kalıpcıoğlu, K.C., Toğay, C., Yolaçan E.N. 2019. Son kullanıcılar için anomali saldırı tespit sistemleri. *Eskişehir Osmangazi Üniversitesi Mühendislik ve Mimarlık Fakültesi Dergisi*, 27(3): 199-212.

Kalıpcıoğlu, K.C. 2020. Zararlı yazılım araştırmasında makine öğrenmesi yöntemleri. *BİLGEM Teknoloji*, 8: 62-65.

Mataracıoğlu, T., Kalıpcıoğlu K.C., Arıkan S.M., Işık G., Demiral, Y. Cincioğlu D., Mantar, H.A. 2020. Küresel salgın sonrasında ulusal bilişim güvenliği. Küresel Salgının Anatomisi: İnsan ve Toplumun Geleceği: Türkiye Bilimler Akademisi, Ed: Şeker, M., Özer, A., Korkut, C., Ankara, Türkiye, pp: 903-937.