T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇALIŞTIRILABİLİR DOSYALARIN NÜMERİK VE METİNSEL ÖZELLİKLERİ KULLANILARAK MAKİNE ÖĞRENMESİ İLE ZARARLI YAZILIM TESPİTİ

YÜKSEK LİSANS TEZİ

Sefu MOHAMED

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Doç. Dr. İbrahim ÖZÇELİK**

Eylül 2019

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÇALIŞTIRILABİLİR DOSYALARIN NÜMERİK VE METİNSEL ÖZELLİKLERİ KULLANILARAK MAKİNE ÖĞRENMESİ İLE ZARARLI YAZILIM TESPİTİ

YÜKSEK LİSANS TEZİ

Sefu MOHAMED

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 09.09.2019 tarihinde aşağıdaki jüri tarafından oybirliği / ~~oyçokluğu~~ ile kabul edilmiştir.

Prof. Dr.	**Prof. Dr.**	**Doç. Dr.**
Cemil ÖZ	**Resul KARA**	**İbrahim ÖÇELİK**
Jüri Başkanı	**Üye**	**Üye**

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim.

Sefu MOHAMED

06.09.2019

TEŞEKKÜR

Öncelikle, ALLAH'a beni sağlıklı tuttuğu ve tezimi tamamlamamda yardımcı olduğu için teşekkür ederim. Onun yardımı olmadan, şu an bulunduğum yerde olmazdım.

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Doç. Dr. İbrahim ÖZÇELİK'e teşekkürlerimi sunarım.

Ayrıca bu çalışma, 117E100 proje numarası ile TÜBİTAK tarafından desteklenmiştir. TÜBİTAK kurumuna desteklerinden dolayı teşekkür ederim. Aile üyelerime, Sakarya'daki arkadaşlarıma ve Siber Güvenlik alanında ortak çalışma yaptığım arkadaşlarıma her şey için teşekkür ederim.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR LİSTESİ

ANN	: Artificiel Neural Network
API	: Application Programming Interface
AUC	: Area Under The Curve
AUROC	: Area Under the Receiver Operating Characteristics
AV	: Anti-virus
COFF	: Common Object File Format
CSV	: Comma-Separated Values
CTI	: Cyber Threat Intelligence
DD	: Data Directories
DLL	: Dynamic Link Library
DOS	: Disc Operating System
DT	: Decision Tree
FNR	: False Negative Rate
FPR	: False Positive Rate
GHz	: Gigahertz
HWT	: Haar Wavelet Transform
IAT	: Import Address Table
IDF	: Inverse Document Frequency
IR	: Information Retrieval
kNN	: k-Nearest Neighbors
L1	: L1 Based feature selection
Light GBM	: Light Gradient Boosting Machine
MD5	: Message-Digest algorithm 5
NB	: Naive Bayes
NLP	: Natural Language Processing
NN	: Neural Network

OS	: Operating System
OEP	: Original Entry Point
PCA	: Principal Component Analysis
PE	: Portable Execuble file format
PUA	: Potentially Unwanted Applications
RAID	: Redundant Array of Independent Disks
RF	: Random Forest
RFE	: Recursive Feature Elimination
RFR	: Redundant Feature Removal
RIPPER	: Repeated Incremental Pruning to Produce Error Reduction
RVA	: Relative Virtual Address
ROC	: Receiver Operating Characteristics
SHA-1	: Secure Hash Algorithm 1
SMO	: Sequential minimal optimization
SSD	: Solid-State Drive
SVM	: Support Vector Machine
TB	: Tree-based feature selection
TF	: Term Frequency
TF-IDF	: Term Frequency-Inverse Document Frequency
VM	: Virtual machine
VSM	: Vector Space Model

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Taşınabilir Yürütülebilir (PE) dosya format, statik zararlı yazılım analizi, makine öğrenmesi, zararlı yazılım tespiti, metin sınıflandırma, siber güvenlik

Son zamanlarda zararlı yazılımların hızlı bir şekilde gelişmesi ve birçok platform aracılığıyla dağıtılabilir olmasından dolayı, siber saldırıların sayısı da gün geçtikçe artmaktadır. Bu nedenle, zararlı yazılım tespiti ile ilgili araştırmalar bilgi güvenliği topluluğunda ciddi bir sorun haline gelmiştir. Zararlı yazılım tespiti için önerilen ticari çözümlerin çoğu imza tabanlı yöntemlerdir. İmza tabanlı yöntemler temel olarak önceden zararlı yazılım varyantı bilgisini gerektirir. İncelenen akademik ve ticari çalışmalar neticesinde bu yöntemlerin, sıfırıncı gün zararlı yazılım adı verilen "yakalanmayan" zararlı yazılımlar üzerinde çalışmayacakları tespit edilmiştir. Bu sebepten dolayı bu tez çalışmasında, imza tabanlı zararlı yazılım algılamadaki eksikliklerin giderilmesi için Windows Çalıştırılabilir (Portable Executable-PE) dosya biçiminin özelliklerini çıkaran statik analiz kapsamı içerisinde makine öğrenmesi tabanlı bir yöntem geliştirilmiştir. Bu yöntemde PE başlık bilgisi olan Windows çalıştırılabilir dosyaların ham özellikleri, derleyici bilgileri, dinamik bağlantı kütüphaneleri (DLL) ve her DLL'deki API fonksiyonları özellik olarak çıkarılmıştır. Daha sonra, PE başlığının özelliklerinde ilgisiz özellikleri kaldırmak ve özellik kümesinin boyutunu azaltmak için Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) ve Tree-based (ExtraTree) isimli dört farklı özellik seçim tekniği kullanılmıştır. Her bir tekniğin özelliği seçip seçmediğine bakılarak, oylama yapılmış ve sonrasında en yüksek oy puanına sahip özellikler seçilmiştir. Derleyici bilgilerileri, dinamik bağlantı kütüphaneleri ve API fonksiyonları için de metin sınıflandırma teknikleri içerisinde bulunan TF-IDF aracı kullanılarak özellik seçimi yapılmıştır.

Çalışmada kNN, Decision Tree, Random Forest ve Light GBM gibi sınıfladırıcılar kullanılmış ve test sonucunda kullanılan sınıflamdırıcıların performanslarını ölçmek ve zararlı yazılımları tespit etme yeteneklerini değerlendirmek için üç farklı test yapılmıştır: Bu testlerden ilkinde sadece PE başlıklarına (nümerik) ait özellikler kullanılmış, Random Forest sınıflandırıcısı ile %99.49 doğruluk elde edilmiştir. İkincisinde Derleyici bilgileriler, kütüphaneler ve API fonksiyonlarına ait metinsel özellikler kullanılmış, yine Random Forest sınıflandırıcısı ile %99.06'lık tespit doğruluğu elde edilmiştir. Son olarak ise nümerik ve metinsel özellikler birleştirilerek test yapılmış ve bu test sonucunda da genel olarak yine %99.44 duyarlılık, %1.2 yanlış pozitif, %99.25 kesinlik ve %99.13 doğruluk oranları ile en iyi performans Random Forest ile elde edilmiştir.

MALWARE DETECTION WITH MACHINE LEARNING USING EXECUTABLE FILES NUMERIC AND TEXTUAL FEATURES

SUMMARY

Keywords: Portable Executable (PE) file format, Static malware analysis, Machine learning, Malware detection, Text Classification, Cyber Security.

Since malicious software or malware has become too agile, evolves quickly and can distribute itself, the numbers of cyber-attacks continue to grow as well. Therefore malware detection and relevant research became a serious issue in the information security community. Most of the commercial proposed solutions for malware detection are signature-based methods. Signature-based method fundamentally requires prior malware variant knowledge which will not work on malwares that have never been "captured" for analysis, called zero-day malware. This work explores an approach to overcome the deficiencies in signature-based malware detection, namely usage of the static analysis method to extract valuable features of Windows Portable Executable (PE) file format. We extract raw features of Windows executables which are PE header information, used Compilers, DLLs, and API functions inside each DLL of Windows PE file. Thereafter, for PE header's information, four different feature selection techniques (Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) and Tree-based feature selection (ExtraTree)) are used to remove irrelevant features and so reduce dimension of feature set, where each technique votes whether they have selected the feature. Finally, the vote is counted and the features with higher votes score are used with Compilers, DLLs and API functions to train our classifiers. For Compilers Information, DLLs and API functions Text Classification techniques tools such as TF-IDF were used for feature selection.

To evaluate the performances of our detection models (kNN, Decision Tree, Random Forest and Light GBM) and their ability to detect unseen and new malware we conducted three experiments: First by using PE headers features (Numerical) only where we achieve 99.49 % of accuracy with Random Forest, second by using only Compilers Information, Dlls and API functions features (Textual) where 99.06% of detection accuracy was achieved again with Random Forest, and finally we combined Numerical and Textual features where an overall best performance was achieved by Random Forest with a recall of 99.44%, a false positive rate of 1.2%, a precision of 99.25%, and an accuracy of 99.13%.

BÖLÜM 1. GİRİŞ

Hükümetler, askeri kurumlar, kamu ve özel sektörler gibi farklı sektörleri hedef alan siber saldırıların sayısı gün geçtikçe katlanarak artmaktadır. Bu siber saldırılar, değerli ve kritik bilgiler elde etme çabasıyla bireyleri veya kuruluşları hedeflemeye odaklanır. Bazen, bu siber saldırıların siber suç veya devlet destekli gruplarla bağlantılı olduğu düşünülse de, sadece hedeflerine ulaşmak için bireysel gruplar tarafından veya merakını tatmin etmek için tek bir kişi tarafından da yapılabilir. Siber saldırıların çoğu, hedeflerini etkilemek/zarar vermek için zararlı yazılım kullanmaktadırlar. Zararlı yazılım kullanılarak artan siber saldırılarla mücadele edebilmek ve etkili/verimli bir şekilde zararlı yazılım tespiti için, gerçek ve büyük günlük örneklem koleksiyonundan yararlanılarak akıllı yöntemler geliştirilmesi gerekmektedir.

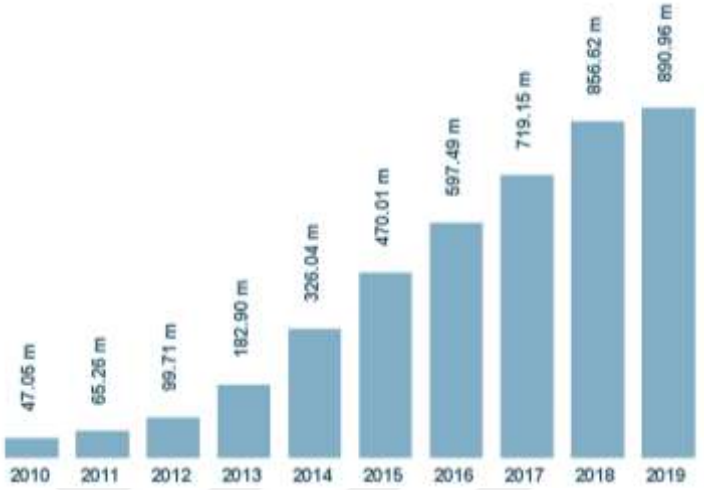
1.1. Problem Tanımı

Zararlı yazılım (Malware), sahibinin bilgilendirilmiş onayı olmadan bir bilgisayara zarar vermek veya sızmak için tasarlanmış zararlı yazılımların kısaltmasıdır. Zararlı yazılım, herhangi bir hesaplama işlemi üzerinde olumsuz etkiye neden olabilecek tüm tehlikeli yazılım uygulamalarını içeren genel bir terimdir.

Bu tanımlara göre, zararlı yazılımı virüsler, solucanlar, arka kapılar, Truva atları, anahtar kaydediciler, şifre hırsızları, script virüsleri, rootkitler, makro virüsleri, casus yazılım ve hatta reklam yazılımı gibi çok çeşitli zararlı (veya potansiyel olarak zararlı) yazılım uygulamalarını kapsayan bir şemsiye terim olarak da kabul edebiliriz.

AV-TEST Enstitüsü [1] tarafından Şekil 1.1.'de sunulan istatistiklere göre, her gün 350.000'in üzerinde yeni zararlı program (zararlı yazılımın) ve potansiyel olarak istenmeyen uygulamalar (PUA) özelliklerine göre incelenip sınıflandırılarak kayıt

altına alınırlar. Örneğin 2017'de, zararlı yazılımların sayısı 597.49 milyondan (2016) 719.15 milyona, 2018'de 719.15'ten 856.62 milyona yükselmiştir. Şekil 1.1.'de görebileceğimiz gibi, sadece 2019'un ilk üç ayında, 34.34 milyon yeni zararlı yazılım artışı söz konusudur.



Şekil 1.1. AV-TEST Zararlı yazılımların yıllara göre dağılımı - Nisan 2019 [1].

Bu durum, bilgisayar sistemlerini korumak için etkili zararlı yazılım analizi ve algılama tekniklerine olan ihtiyacı arttırmaktadır. Ayrıca, günlük olarak yayınlanan sayılamayacak kadar büyüklükteki sıfırıncı gün (zero day) zararlı yazılım miktarı, otomatik sınıflandırıcılar / dedektörler oluşturmasını da zorunlu kılmıştır.

1.2. Literatür Taraması

Akademik incelemeler neticesinde son yıllarda, birçok bilgisayar ve siber güvenlik araştırmacısının, zararlı yazılımları sınıflandırmak için bir hesaplamalı teknik olan makine öğrenmesini kullandığı görülmüştür.

Bilinmeyen bir yazılımın, zararlı olup olmadığının belirlenmesi genellikle zor bir süreç olsa da [2], zararlı yazılım araştırmacıları için zararlı yazılımların kabul edilebilir bir algılama oranına sahip olarak algılanması araştırmanın en önemli önceliğinden biri olmaya devam etmektedir. Zararlı yazılımları tespit etmek için iki ana yaklaşım kullanılır: statik yaklaşım ve dinamik yaklaşım.

Statik yaklaşım, herhangi bir kod çalıştırılmadan tamamen çalıştırılabilir ikili dosyaları veya montaj kodunu kontrol eder. Bilinmeyen zararlı yazılımları tespit etmek için statik özellikleri kullanarak makine öğrenmesi sınıflandırıcılarının uygulanması üzerine son yıllarda yapılan bir çalışmada [3], Makine Öğrenimi destekli statik zararlı yazılım analizinin, PE32 Windows dosyalarının özellikleriyle statikten ödün verme belirtilerinin algılanmasını otomatikleştirmek için Siber Tehdit Zekası (Cyber Threat Intelligence - CTI) faaliyetlerinin bir parçası olarak kullanılabileceğini düşünülmüştür. Statik yaklaşımın dinamik karşılığı üzerinde en büyük avantajı, yürütme süresinin ek yükünden bağımsız olmasıdır.

Statik tespitin genel olarak kararsız olduğu bilinmesine rağmen [4], yürütmeden önce zararlı dosyaların tespit edilmesine izin verdiğinden dolayı güvenlik protokol yığınında önemli bir koruma katmanı olarak kabul edilir.

Önerilen çalışma, çalıştırılabilir dosyaların (PE) zararlı yazılım tespiti için göz önünde bulundurulması amacıyla statik analize dayanmaktadır, bu nedenle bu ilgili çalışma bölümü, PE dosyalarının statik analizine dayanan, imza dışı zararlı yazılım tespiti konusundaki kayda değer önceki çalışmalardan bazılarını açıklamaktadır.

[5] 'de Schultz ve arkadaşları sıfırıncı gün (daha önce görülmeyen) zararlı yazılımları tespit etmek için zararlı ve zararsız yazılımların statik özelliklerinin kullanılmasını araştırarak Windows platformunda bu amaç için üç farklı yöntem önermişlerdir. İlk teknik, DLL'lerin (dinamik bağlantı kütüphanesi) listesine dayanmaktadır, ikili kod tarafından kullanılan her bir DLL içindeki farklı fonksiyon çağrılarının sayısı belirlenmektedir. İkinci teknikte ise dizgiyi ikili özellik olarak kullanmışlardır. Son olarak üçüncü teknikte ise, ikili özellik olarak iki bayt n-gram (bir dize yerine) yöntemini kullanmışlardır. Çalışmalarında farklı veri madenciliği sınıflandırıcıları (RIPPER, Naive Bayes and Multi-Naive Bayes) kullanarak, 3,265'inin zararlı ve 1,001 zararsız yazılım olduğu 4,266 dosyadan oluşan standart bir veri setini eğitmişlerdir. Yapılan performans denemeleri neticesinde, çoklu naive bayes sınıflandırıcılar için %97.76 başarı elde edilmiştir. Sonuçları geleneksel imza bazlı yöntemlerle karşılaştırdıktan sonra, bilinmeyen zararlı yazılımın veri madenciliği tabanlı tespit

oranının imza tabanlı tespit algoritmasında iki kat daha fazla başarı gösterdiği iddia edilmiştir. Bu çalışmada, yazarların yalnızca 4,266 dosyadan oluşan bir veri seti kullandıklarını, bununla birlikte, eğer daha fazla dosya ile deneyler yapılabilirse ortaya çıkan sonuçlar hipotezlenebilir kanısına varılmıştır.

Bir başka çalışmada, J.Z. Kolter ve arkadaşları [6], üst üste binmeyen sekanslar yerine bayt n-gramları uygulayarak; her dört baytlık sekans tek bir terim halinde kullanmak kaydıyla Schultz ve arkadaşlarının önermiş olduğu üçüncü tekniği geliştirmişlerdir. Yazarların birincil veri seti 1,971 zararsız ve 1,651 zararlı programdan oluşmaktadır. 4 gram özellik olarak kullanılmış ve ilk 500 n gram bilgi edinme ölçüsü ile seçilmiştir. Seçilen özellikler tüm sınıflandırıcılara girdi olarak sağlanmıştır. Naive Bayes, örnek tabanlı öğrenmeler, Karar ağaçları (Decision Tree), TF-IDF ve Destek vektör makineleri (Support Vector Machine) kullanılmış ve son üç sınıflandırma algoritmasını da güçlendirilmiştir. En iyi sonuç "eğri altındaki alan" (AUC) kullanılarak sağlanmış, 0.996'da artırılmış J48 ve %97 doğruluk elde edilmiştir. J.Z. Kolter ve arkadaşlarının çalışmasının [6] eksik yönlerini göstermek için, Dai [7] deneylerini yeniden üretmiş, filtrelenmiş özellikleri analiz etmiş ve özelliklerin çoğunun PE dosyasının biçim özellikleri olduğunu ve çok azının OpCode dizileri olduğunu belirtmişlerdir, bu nedenle bu tez çalışmasında OpCode dizisi özellikleri çok fazla dikkate alınmamıştır.

Shafiq ve arkadaşları [8] 189 özelliği PE dosya başlıklarından çıkardıktan sonra, özellik setinin boyutunu azaltmak için üç farklı özellik seçim yöntemi ((Redundant Feature Removal (RFR), Principal Component Analysis (PCA) ve Haar Wavelet Transform (HWT)) kullanmışlardır. 1,444'ü zararlı yazılım ve 1,330'u zararsız yazılım olan 2,774 dosyadan oluşan bir veri seti üzerinde beş veri madenciliği sınıflandırıcısını (IBk, J48, NB, RIPPER & SMO) çalıştırmışlardır. Yapılan denemeler neticesinde %0.5 Yanlış Pozitif (FP) ve %99 doğruluk oranları elde edilmiştir. Bu çalışmadan farklı olarak bu çalışmada her DLL dosyasından çıkartılan API fonksiyonlarına, farklı özellik seçim yöntemlerinden (Filter, wrapped and Embedded methods) dört farklı özellik seçim tekniği uygulanmış ve bilinmeyen ve yeni zararlı yazılımı tespit etmek için de dört farklı sınıflandırıcının (Random Forest, Decision Tree, k-Nearest

Neighbor and Light GBM) yeteneğini değerlendirmek için ayrı ayrı üç önemli deney (Nümerik özellikler olarak adlandırdığımız sadece PE başlıkları özelliklerini kullanarak, bazı Derleyicilerin bilgilerini, metinsel özellikler olarak adlandırdığımız DLL ve her DLL içindeki API fonksiyonlarını kullanarak ve son olarak her ikisinin bir kombinasyonunu kullanarak zararlı yazılım tespiti) yapılmıştır.

[8] ile aynı yılda, Wang ve ark. [9] görünmeyen PE zararlı yazılımlarını tespit etmek için Support Vector Machine (SVM) tabanlı bir algılama yöntemi önermişlerdir. Statik analiz kullanılarak, PE başlık girişleri çıkarılan bu çalışmada seçilen özellikler için SVM sınıflandırıcısı kullanılarak eğitilmiştir. Kullandıkları özellikler ikili PE başlık özellikleridir. [9] nolu çalışmada ise, Veri Dizinleri'nde (Data Directories) (DLL ve API çağrıları) depolanan bilgiler hakkında yeterince araştırma yapılmadığı ve sadece PE başlık özelliklerini kullanıldığı ve sonuçların bu tez çalışmasındaki çalışmayla (% 0.26 iken pozitif oran Tablo 2.6.'ya bakılabilir) karşılaştırıldığında bazı metriklerin yanlış pozitif oranına sahip oldukları tespit edilmiştir.

[10] 'nolu çalışmada ise Liao, özellik olarak sadece beş PE başlık alanı seçerek sınıflandırma için el yapımı kural tabanlı bir algoritma uygulamıştır. Liao'nun algoritması bilinmeyen zararlı yazılımlar için %99 doğruluk ve %0.2 yanlış pozitif başarı oranına sahiptir. Bu çalışmada ne özellik seçimi için ne de sınıflandırma için herhangi bir makine öğrenme algoritmasının kullanılmadığı tespit edilmiştir.

Bai ve arkadaşarı [11] yaptıkları çalışmada, farklı PE başlık alanlarından çıkarılan 197 özellikten oluşan bir ilk özellik grubu oluşturmuşlardır. Özellikleri sırasıyla 19 ve 20'ye düşürmek için Filter ve Wrapped özellik seçim yöntemini kullanan Bai ve arkadaşları, seçilen özellikler için Decision Tree (DT), Random forest (RF), Bagged and Boosted decision tree gibi ağaç temelli sınıflandırıcıları kullanmışlardır. Bu yaklaşımlardan (denemelerden) en iyi sonuç, %1.4'ün altında yanlış-pozitif oranlarla %99.10 tespit oranlarına ulaşmışlardır. Çalışmamızla karşılaştırıldığında yaklaşımımızın bilinmeyen zararlı yazılımları, sadece başlık özellikleri (Nümerik) kullanıldığında %0.26; derleyici bilgilerileri, dinamik kütüphaneler (DLL) ve API Fonksiyonları özellikleri (sadece Metinsel) kullanıldığında %2.91; nümerik ve

metinsel özellikleri birleştirerek kullandığımızda ise %1.2 yanlış-pozitif oranı ile tespit edebildiği ortaya çıkmıştır. Bai ve ark. ise başlık, Dlls ve API Fonksiyonları özelliklerini kullanarak bilinmeyen zararlı yazılımları %1.3 yanlış-pozitif oranı ile tespit edebilmiştir. Yanlış-pozitif oranları nümerik özellikler için %99.49, metinsel özellikler için %99.06 ve nümerik ve metinsel özellikler birlikte için %99.13 doğruluk tespit oranlarında elde edilmiştir. Bizim çalışmamızdan bilinmeyen zararlı yazımları sadece nümerik özellikler kullanıldığında Bai ve ark.'dan daha düşük yanlış-pozitif oranı ile tespit edilmiştir. Hem nümerik hem de metinsel özellikler kullanıldığında Bai ve ark.'dan yine daha düşük yanlış-pozitif oranı elde edilmiştir. Sadece metinsel özellikler kullanıldığında yanlış-pozitif oranında bir artış görülmüştür.

Ayrıca, tez çalışmasında farklı özellik seçim yöntemlerinden 4 farklı özellik seçim tekniği (Filter, Wrapped ve Embedded methods) uygulanırken, Bai ve arkadaşları [11] 2 özellik seçim yönteminden (Filter ve Wrapped) sadece 2 özellik seçim (CfsSubsetEval (Filter yöntemi) ve WrapperSubsetEval (Wrapper yöntemi)) tekniğini kullanmışlardır.

Son zamanlarda, Kozachok ve ark. [12], PE başlık içeriğinden hesaplanan ikili değerler dikkatlice hazırlanmış özellik temelli ikili değerler ile yeni seviyelere ulaşmışlardır. Decision Tree ve Artificial Neural Network (ANN) kullanarak tespit oranları sırasıyla %99.2 ve %99.1'e yükseldiği gözlemlenmiştir. Bu çalışmada [12], yaptığımız çalışmayla karşılaştırıldığında hala bir boşluk bırakan Veri Dizinleri'nde (DLL ve API çağrıları) depolanan bilgileri incelemedikleri görülmüştür.

Diğer taraftan yakın tarihli makalelerde bile, veri setinin numuneleri için (genellikle yaklaşık 19,000 numune) ortak bir sınırlama fark edebiliriz. Bu çalışmada diğer çalışmalardan farklı olarak 50,932 numune (11,122 zararsız yazılım ve 39,810 zararlı yazılım (yeni ve eski numunelerin karışımı)) kullanılmıştır. Dahası, her biri farklı bir veri seti kullanan çalışmaları karşılaştırmanın zor olmasının yanı sıra, her biri farklı bir veri seti kullandığında ve bunları kamuya açık hale getirmediğinde, bunların çoğaltılmasını imkânsız kılmaktadır. Ayrıca, bu sorun için standart ölçümler olmadığı da açıktır.

Kısacası, önceki çalışmalarda kullanılan zararlı yazılım numuneleri çeşitli nedenlerden dolayı kullanılamamakta ya da veri setleri kurumlara özel oldukları için paylaşılamamaktadır. Yalnızca kullanılmış numuneleri kullanılabilir hale getirebilselerdi, yinelenen kaldırma, dosya türü tanımlama ve çok zaman alan etiketleme gibi ön işleme görevinden kaçınılmış olunabilinirdi. Bu nedenle, önerilen çalışmada bu konuyu dikkate alarak konuyla ilgili daha fazla araştırma için standart etiketli bir veri seti sağlanmıştır. Ayrıca, daha önce sunulan çalışmaların çoğu ya sadece yeni zararlı yazılım numunelerini ya da sadece eski zararlı yazılım numunelerini kullanmaktadır. Ayrıca, bu çalışmaların çoğunda zararsız yazılımların numuneleri için sadece .exe dosyası kullanılmakta iken bu tez çalışmasında önerilen yöntemde .dlls dosyalarını da dikkate alınmıştır.

Yukarıda belirtilen akademik çalışmalardaki yöntemler dikkate alınarak, pefile [13] kütüphanesini kullanan ve bir özelliği oluşturan bir kod (script) yazılmıştır. Yont'un çalışmasına dayanarak [14], dört ana başlıktan bir seti PE başlık (ham ve türetilmiş özellikler) oluşturulmuş, sonrasında ise Veri Dizinleri'nden (Derleyici bilgileri, DLL ve API Fonksiyonları) bir setiyle birleştirilmiştir.

İncelenen araştırmalara göre, bu tez çalışmasında, farklı özellik seçim yöntemlerinden özellik seçimi için 5 farklı teknik (Bölüm 3.4.3.'de belirtilmiştir.), PE başlıkları için 4 teknik (Nümerik) ve Dlls - API Fonksiyonları (Metinsel) için 1 teknik kullanmanın zararlı yazılım tespitinde hızlı ve etkin bir çözüm olabileceğine inanılarak çalışma yapılmıştır.

1.3. Tez Motivasiyonu

[15] 'e göre pazarın yüzde 81.8'i Windows İşletim Sistemini kullanmaktadır, bu da dünya çapında zararlı dosyaların büyük çoğunluğunun Windows kullanıcılarına yönelik olduğu anlamına gelmektedir. Bu nedenle, bu çalışmada Windows Portable Executable (PE) dosyalarıyla çalışılması seçilmiştir. PE dosyasının başlığının zararlı yazılımlar için çok güçlü bir gösterge olabileceğini göz önünde bulundurarak [3], derleyici (Compiler) tarafından Microsoft özelliklerine göre otomatik olarak

oluşturulduğundan, bu özelliklere uymayan herhangi bir özelliğin kasıtlı olarak manipüle edilmiş olması gerektiği anlamına gelir. Ayrıca, DLL ve API'lerin çağrı fonksiyonlarının kullanımını analiz ederek çalıştırılabilir bir programın davranışını ve işlevselliğini tahmin etmenin uygulanabilirliği göz önüne alındığında, her çalıştırılabilir dosyanın PE formatındaki içe aktarma tablosu çıkarılmış ve yürütmeden önce zararlı yazılım algılama gerçekleştirilmiştir. İmza tabanlı yöntem için imzaların ayarlanan boyutu ve eşleşen imzaların süresi katlanarak artmaktadır. Son olarak, yeni (görünmeyen) bir zararlı yazılımın ortaya çıkması ile müşterinin antivirüs grubunun imzalarının güncellenmesi arasındaki süreçte milyonlarca bilgisayar yeni zararlı yazılıma maruz kalmaktadır. Bu sebepten yeni bir algılama yöntemine ihtiyaç duyulmaktadır.

Statik tespitin genel olarak kararsız olduğu bilinmesine rağmen [4], güvenlik paketinde önemli bir koruma katmanıdır, çünkü başarılı olduğunda, yürütmeden önce zararlı dosyaların tespit edilmesine izin vermektedir. Mevcut araştırmalar, antivirüs üreticilerinin zararlı yazılımları tespit etmek için yalnızca imzaları kullanmaya devam etmelerinin net nedenlerini ortaya koymamıştır; bu çalışmada önerilen yöntem, makine öğrenmesini kullanarak zararlı yazılım saptamadaki mevcut yöntemlere ek destek sağlamaktadır.

1.4. Tezin Literatüre Katkısı

Bu çalışmada yalnızca Windows ikili (binary) dosyalarını değil, aynı zamanda diğer uygulamaların Windows için yürütülebilir dosyalarını içeren özel bir veri setimizi (araştırma amacıyla paylaşılabilir) toplanmıştır. Farklı çalışmalardaki birçok veri seti sadece Window ikilileri içerir. Zararsız dosyalar Windows XP, Windows 7, Windows 8 ve Windows 10 ikili dosyalarından (EXE ve DLL) toplanmış, zararlı dosyalar da [16] ve [17] 'den EXE dosyaları indirilip toplanmıştır. Yeni toplanan dosyalar, veri setinin numune sayısını 50,932'e (11,122'si zararsız ve 39,810'u zararlı dosya) kadar çıkartmıştır. İyi bir analiz modelinin oluşturulması için zararsız ve zararlı yazılım koleksiyonumuz, [18] ve [19] 'dan yeni ve eski zararlı dosyalar da indirerek çeşitlendirilmiştir.

Bu çalışmada öncelikle ayrı ayrı üç farklı tespit sistemi gerçekleştirilmiş ve sonrasında bu sistemler tek bir çatı altında birleştirilmiştir. İlk önce, Nümerik özellikler dediğimiz PE dosya başlıkları kullanarak bir çalışma yapılmıştır. İkinci olarak ise sadece Derleyici bilgilerileri, DLL ve API Fonksiyonları (Metinsel) kullanılmıştır. Üçüncü ve son olarak da birinci ve ikinci sistem birleştirilmiştir. PE dosya başlıklarını kullanarak, birçok çalışmanın özellik seçimi için neredeyse tek bir teknik kullandığını hatırlamamız gerekmektedir. Bu tez çalışmasında, veri setindeki en alakalı özellikleri bulmak için, farklı özellik seçim yöntemlerinden (Filter, wrapped ve Embedded methods) özellikleri seçmek için Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) ve Tree-based feature selection (ExtraTree) gibi farklı teknikler de kullanılmıştır. Her bir tekniğin özelliği seçip seçmediğine bakılarak, oylama yapılmış ve sonrasında en yüksek oy puanına sahip özellikler, sınıflandırıcıları eğitmek için kullanılmıştır.

1.5. Sınırlar

Microsoft Windows işletim sistemleri bugün dünyada en yaygın kullanılan işletim sistemleridir. Windows'un yaygınlığı, virüs yazarlarının zararlı yazılım yazması için çekici bir ortam sağlamaktadır. Bu çalışmada, yalnızca Windows işletim sistemleri için PE formatında yazılmış zararlı yazılımlara odaklanılmaktadır.

Ayrıca, gerçek dünyadaki senaryonun öngörülemez olduğunu ve deneysel ortamdan farklı olabileceği dikkate alındığında; hassas bir sistemi zararlı yazılımlardan korumak için, yalnızca statik analiz tabanlı zararlı yazılım tespit sınıflandırıcısının kullanılması tavsiye edilmez. Bu nedenle, önerilen çalışma opcode-n-gram tabanlı sınıflandırıcılarla rekabet etmesi beklenmemektedir ve zararlı yazılımı çalıştıran talimat tabanlı analiz tekniklerini veya analiz tekniklerini uygulamamaktadır.

1.6. Tez Organizasyonu

Bu tez çalışması, beş bölümden oluşmaktadır. İkinci bölüm, bu çalışmanın teorik temelini belirlerken, Windows PE dosya biçimi, zararlı yazılım türleri, virüslerin

dosya bulaştırma stratejileri, zararlı yazılım analizi ve algılama teknikleri, bilgi çıkarma ve Makine Öğrenimi konularını sunar. Üçüncü bölümde bu çalışmada kullanılan metodoloji açıklanmaktadır. Dördüncü bölümde araştırmamızın uygulanması ve deneysel sonuçları analiz edilecektir. Son olarak, beşinci bölümde ise bu çalışmanın sonuçları ve gelecek çalışmalarından bahsedilecektir.

BÖLÜM 2. TEORİK ARKA PLAN

Bu bölüm zararlı yazılım türleri, virüslerin dosya enfeksiyonu stratejileri, windows PE dosya biçimi, zararlı yazılım analizi ve algılama teknikleri, bilgi çıkarma ve makine öğrenmesi konuları hakkında teorik bir arka plan sunar. Makine öğrenmesi bölümünde, zararlı yazılım tespitinde makine öğrenmesi gerekliliğini ve ardından bu çalışma ilgili yöntemlerin nasıl kullanıldıkları ile ilgili bilgiler verilecektir. Bu yöntemler arasında k-Nearest Neighbors, Decision Trees, Light Gradient Boosting Machine ve Random Forests bulunmaktadır.

2.1. Zararlı Yazılım Tipleri

Bu çalışmanın amacı zararlı yazılım tespiti olduğundan, önce zararlı yazılımların tipik davranışları, bir sistemi nasıl etkilediği, nasıl gizlediği ve nasıl zarar verdiği hakkında bazı bilgiler verilecektir.

1) Reklam yazılımı (Adware): Kullanıcıya istenmeyen reklamlar (ads) sunan zararlı yazılım tipidir. Genellikle ücretsiz indirmeler şeklinde olup ve hedef sisteme zorla yazılım yükleyebilirler [20].

2) Arka kapı (Backdoor): Saldırganın erişmesine izin vermek için kendisini bir bilgisayara yükleyen zararlı kodlardır. Arka kapılar genellikle saldırganın bilgisayara kimlik doğrulaması olmadan bağlanmasına izin vererek yerel sistemde komutları uygularlar [21].

3) Spam gönderen zararlı yazılım (Spammer): Bir kullanıcının makinesine bulaşan ve daha sonra spam göndermek için bu makineyi kullanan zararlı yazılımlara verilen isimlerdir. Spam gönderenler mesajlarını çevrimiçi topluluklardaki e-posta,

SMS veya postalamalar ve yorumlar şeklinde gönderebilirler. Bu zararlı yazılımlar, saldırganlar için spam gönderme hizmetleri satmalarına izin vererek gelir sağlar.

4) Virüs (Virus): Bir virüs, hedef programlara sızarak ve onların başlık yapılarını değiştirerek veya bunun yerine virüs kodunu işaret etmek için bu programlara yapılan referansları değiştirerek kendisini tekrar edebilir. Bir virüs muhtemelen yeni nesiller ile kendini mutasyona uğratır. [22]. Kullanıcı, virüslü bir dosyayı çalıştırırsa virüs hedef sistemde aktif olur. Böyle bir virüslü dosyaya biyolojik parazitler için kullanılan terminolojiye atıfta bulunularak "ana bilgisayar" dosyası da denir. Virüsler, geleneksel olarak disket, USB flash sürücü, CD veya DVD gibi aktarılan ortamlar aracılığıyla diğer bilgisayarlara yayılırlar.

   Herhangi bir enfeksiyondan önce, orijinal halindeyse, virüs olarak adlandırılır. Mikrop kodunun ilk kurulumu bir damlalık tarafından yapılır, daha sonra virüs "kendi başına kopyalanabilir" [22].

   Amaçlanan bir virüs, bir hata veya uyumsuz bir ortam nedeniyle çoğalamayabilir [23]. Uyuyan virüsler pasif bir durumdadırlar, hedef makine üzerinde hiçbir işlem yapmadan, bir tetikleyicinin hareketini veya uyumlu bir sistemin yayılmasını beklerler [23]. Uyuyan virüslerin hedef sistemde aktif olanlarına ise aktif virüsler denir.

5) Solucan (Worm): Bir Truva atı (Truva atı), hedef sistem için faydalı fonksiyonlar sağlayarak veya kullanıcının bu yazılımın yararlı ve iyi niyetli olduğuna inanmasını sağlayarak onu çalıştırmasını sağlayan zararlı bir yazılımdır [22]. Truva atları genellikle indirici, damlalık, arka kapı, bilgi hırsızı özellikleri ile birleştirilmiştirler.

6) Indirici (Downloader): Yalnızca diğer zararlı kodları indirmek için mevcut olan zararlı yazılımlardır. İndiriciler genellikle bir sisteme ilk eriştiğinde saldırganlar tarafından yüklenirler. İndirme programı ek zararlı kod indirir ve yükler [21].

7) Fidye yazılımı (Ransomware): Fidye yazılımı, kullanıcıların sistem ekranını kilitleyerek veya fidye ödenmediği sürece kullanıcıların dosyalarını kilitleyerek kullanıcıların sistemlerine erişmelerini engelleyen veya sınırlayan bir zararlı yazılım türüdür. Toplu olarak kripto-fidye yazılımı olarak sınıflandırılan daha modern fidye yazılımı aileleri, virüslü sistemlerde belirli dosya türlerini şifreler ve şifre çözme anahtarı almak için kullanıcıları belirli çevrimiçi ödeme yöntemleriyle fidye ödemeye zorlarlar [24].

8) Dropper: Bir dropper, diğer zararlı yazılımları dosya sistemine yazan bir programdır. Dropper, kurulum rutinleri uygulayabilir ve [25] ve [21] zararlı yazılımlarını çalıştırabilir. Bir indiricinin dropper'den farkı, dropperın zaten kötü niyetli kodu kendi içinde içermesidir. [26].

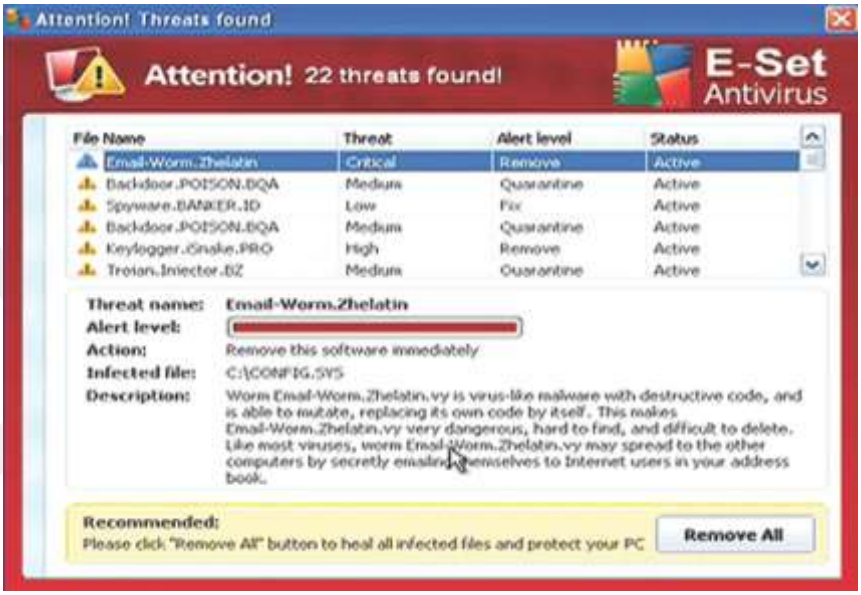
9) Rootkit: Diğer kodun varlığını gizlemek için tasarlanmış zararlı yazılımlardır. Rootkit'ler genellikle saldırgana uzaktan erişim sağlamak ve kurbanın tespit etmesini zorlaştırmak için arka kapı gibi diğer zararlı yazılımlarla eşleştirilir [21].

10) Bilgi çalan zararlı yazılım (veya kısa hırsızlık): Bir kurbanın bilgisayarından bilgi toplayan ve genellikle saldırgana gönderen bir zararlı yazılımdır. Örnekler sniffer, şifre karması gaspçıları ve keyloggerları içermektedir. Bu zararlı yazılımlar genellikle e-posta veya çevrimiçi bankacılık gibi çevrimiçi hesaplara erişmek için kullanılır.

11) Scareware: Scareware Malware, virüslü bir kullanıcıyı bir şey satın almaya korkutmak için tasarlanmıştır. Genellikle antivirüs veya diğer güvenlik programları gibi görünmesini sağlayan bir kullanıcı arayüzü vardır. Kullanıcılarına, sistemlerinde kötü niyetli kodlar olduğunu ve ondan kurtulmanın tek yolunun "yazılımlarını" satın almaları olduğunu, gerçekte, sattığı yazılımın korsanların kaldırılmasından başka bir şey yapmadığını bildirir.

Tipik bir korsan yazılım örneği, bir virüsten koruma tarayıcısına benzeyen ve kullanıcının zararlı kod hakkında sahte uyarıları gösteren, kullanıcının bir virüsü olduğunu ve kurtulmak için tek yapması gereken, reklamı yapılan virüsten koruma yazılımını satın alması olduğunu söyleyen bir programdır.

Kullanıcıya zararlı kodu kaldırmak için yazılım satın alınması söylenir. Şekil 2.1., virüsten koruma yazılımı gibi görünen korsan yazılımları göstermektedir.



Şekil 2.1. Korsan yazılım - Scareware [27]

12) Zararlı yazılım geliştirme kiti: Bir zararlı yazılım yapım kiti, kullanıcı tarafından tanımlanan ayarlara dayalı zararlı yazılım veya zararlı yazılım kaynakları üreten bir programdır [22]. Zararlı yazılım geliştirme kitleri, herhangi bir programlama bilgisi olmayan kişilerin, özel zararlı yazılım oluşturmalarını sağlar. Basit kitler, e-posta adresleri veya kurbanın bilgisayarında bulunan zararlı yazılımın bulduğuna ilişkin bilgi için alıcı olan FTP hesapları gibi küçük ayarlamalar yapılır. Şekil 2.2.'de, fidye tutarı özelliğine sahip bir zararlı yazılım inşaatı kiti örneği gösterilmektedir.

Şekil 2.2. Zararlı yazılım yapı kiti örneği [28]

2.2. Dosya Enfeksiyon Stratejileri

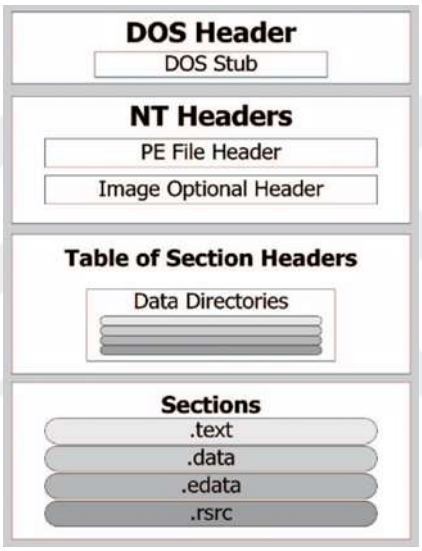
PE dosyası formatı, virüs bulaşan bir ana bilgisayar formatıdır. Dosya enfeksiyonları, yanlış formatlara neden olabilir çünkü çoğu enfeksiyon tipi, ana bilgisayar dosyasının modifikasyonunu gerektirir. Bu nedenle, bazı kötü biçimlerin nasıl oluştuğunu anlamak için virüslerin dosya enfeksiyon stratejilerini bilmek, aynı zamanda analiz sırasında enfeksiyon tiplerini belirlemek için de faydalıdır. Mevcut bir PE yürütülebilir dosyasına yeni virüs kodu eklemek için pek çok strateji vardır.

Üç yaygın strateji verelim:

- Mevcut kod bölümünün kullanılmayan dolgusunda visus kodunu tek parçaya ekleme.
- Virüs kodunun parçalara ayrılması ve bu parçaların mevcut bir kod bölümünde bulunan kullanılmayan bir alana konulması.
- Virüs kodunu tamamen yeni bir kod bölümü olarak ekleme.

2.3. Windows PE Dosya Biçimi

Taşınabilir yürütülebilir (PE) dosya, EXE dosyaları, DLL'ler, ekran koruyucular ve sürücüler dahil olmak üzere bir dizi farklı nesne türünü tanımlayabilir. Yükleyici, işlem yüklenirken bellekteki verileri nasıl eşlemesi gerektiğini tanımlar. Yürütülebilir dosyalar ve Dinamik Bağlantı Kitaplığı dosyaları (DLL) aynı PE biçimini kullanır; Bu bağlamda çalıştırılabilir ve DLL arasındaki fark şudur: bir DLL başlatılamaz, ancak yalnızca başka bir ikili tarafından kullanılabilir ve bir ikili çalıştırılabilir dosyaya bağlanamaz.



Şekil 2.3. PE dosya biçiminin basit yapısı

Bir PE dosyasının başında bir ‘MS-DOS çalıştırılabilir’ ("stub") bulunur; Bu, herhangi bir PE dosyasını geçerli bir MS-DOS çalıştırılabilir hale getirir.

’DOS-stub’tan sonra, sihirli sayı 0x00004550 olan 32 bitlik bir imza vardır (IMAGE_NT_SIGNATURE). İmza, dosyayı PE görüntüsü olarak tanımlayan 4 baytlık bir değerdir (PE\0\0). Daha sonra, ikili makinenin hangi makinede çalışması gerektiğini, içinde kaç bölüm olduğunu, bağlanma süresini, çalıştırılabilir veya DLL olup olmadığını söyleyen bir Dosya başlığı (COFF formatında) vardır.

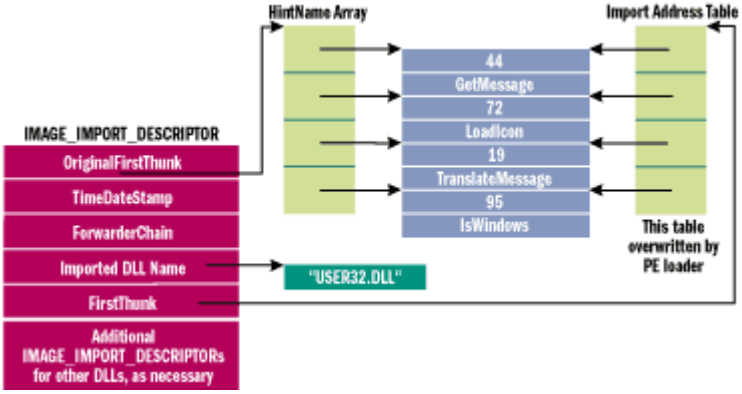
Dosya başlığı, aşağıdaki gibi işletim sistemi için kritik bilgiler içerir:

a) Machine: Bu alan, dosyayı çalıştırabilecek makineleri tanımlamak için kullanılır.
b) NumberOfSections: Bölüm tablosunun boyutunu tanımlamaktadır.
c) TimeDateStamp: Dosyanın ne zaman oluşturulduğunu belirtir.
d) PointerToSymbolTable: Dosyanın başlangıcından COFF sembol tablosunun bulunduğu bayt ofsetini belirtir.
e) NumberOfSymbols: Dosyanın sembol tablosundaki sembollerin sayısını belirtir.
f) SizeOfOptionalHeader: Bu özellik, isteğe bağlı üstbilgideki toplam bayt sayısını belirtir.
g) Characteristics: Bireysel bitleri dosyanın belirli özellikler gösterip göstermediğini temsil eden iki bayt, örneğin, bu karakteristik bitlerden biri, dosyanın bir sistem dosyası olup olmadığını ve bir kullanıcı programı olmadığını gösterir; sistem dosyaları belirli enfeksiyon türlerine karşı daha duyarlı olabilir.

Ondan sonra, Optional header (her zaman var ancak yine de "Optional" olarak adlandırılıyor) vardır. (COFF kütüphaneler için "Optional header" kullanır, ancak nesneler için kullanmaz, bu yüzden "Optional" olarak adlandırılır). Bu bizlere ikilin nasıl yükleneceği hakkında daha fazla bilgi verir: Başlangıç adresi, ayrılacak yığın miktarı, veri bölümünün boyutu vb. İsteğe bağlı başlığın bir parçası "Data Directories" (DD)'nin son dizisidir; bu dizinler bölümlerindeki verilere yönelik işaretçiler içerir. Örneğin, ikili dosyada bir dışa aktarma dizini varsa, dizi üyesinde bu dizine bir işaretçi bulanacaktır, IMAGE_DIRECTORY_ENTRY_EXPORT, ve bölümlerden birine işaret edecektir.

DLL'den içe aktarılan fonksiyonla ilgili bilgiler aranmak istendiğinde, önce Veri Dizinleri'ndeki IMAGE_DIRECTORY_ENTRY_IMPORT girişi bulunur, RVA alınır ve bu adres ham bölüm verilerinde aranmak suretiyle IMAGE_IMPORT_DESCRIPTOR'ların bir dizisi bulunur (Şekil 2.4.). Adın gösterdiği karakter dizileri incelenerek ilgili DLL'e ait IMAGE_IMPORT_DESCRIPTOR veri kaydı alınıp içindeki "OriginalFirstThunk"

işaretçi takip edilerek IMAGE_THUNK_DATA işaretçiler elde edilir ve fonksiyon, RVA'lar incelenerek bulunur.



Şekil 2.4. USER32.DLL'in bazı API fonksiyonlarını kendi içine aktaran bir yürütülebilir dosya (EXE) [29]

İsteğe bağlı üstbilginin kendisi üç ana bölümden oluşur: Standart alanlar, Windows'a özel alanları, Veri dizinleri.

Burada bazı önemli alanlar bulunmaktadır:

a) Magic: Sihirli sayı, PE dosyasının tam türünü tanımlar: PE32 veya PE32 + (64 bit) çalıştırılabilir.
b) MajorLinkerVersion: Tek bir görüntü dosyası oluşturmak için nesne dosyasını ya da dosyaları birleştiren bağlayıcının ana sürüm numarasını belirtir.
c) MinorLinkerVersion: Görüntü dosyasını oluşturmak için nesne dosyasını veya dosyaları birleştiren bağlayıcının küçük sürüm numarasını belirtir.
d) SizeOfCode: Bu özellik, kod bölüm (ler) deki toplam bayt sayısını belirtir.
e) SizeOfInitializedData: Bu özellik, başlatılan veri bölümündeki bölümlerin toplam bayt sayısını belirtir.
f) SizeOfUninitializedData: Bu özellik, başlatılmamış veri bölümündeki bölümlerin toplam bayt sayısını belirtir.
g) AddressOfEntryPoint: "Yürütülebilir dosya belleğe yüklendiğinde, giriş noktasının görüntü tabanına göre adresi" (Microsoft PE ve COFF Şartname, 2013). Bu özellik, yüklü görüntüdeki işletim sisteminin dosyayı ilk

çalıştırdığında kontrolü aktarması gereken konumu gösterir. Bu programın giriş noktasıdır ve genellikle virüs yazarları tarafından manipüle edilir.

h) BaseOfCode: "Belleğe yüklendiğinde, kod başlangıcı bölümünün resim tabanına göre olan adres." (Microsoft PE ve COFF Şartname, 2013). Özellik, yüklenen görüntüdeki kod bölümünün başladığı konumu gösterir.
i) BaseOfData: "Veri başlangıcı bölümünün görüntü tabanına göre, belleğe yüklendiği zamanki adres." (Microsoft PE ve COFF Şartname, 2013). Bu özellik, yüklenen görüntüdeki veri bölümünün başladığı ve yalnızca PE32 dosyalarında bulunduğu konumu gösterir.

Başlıkların ardından, "Section headers" tarafından sunulan Sections bulunmaktadır. Temel olarak, bölümlerin içeriği bir programı yürütmek için gerçekten ihtiyaç olan şeydir ve tüm başlık ve dizin öğeleri onu bulmaya yardımcı olmak için oradadır.

İşletim sistemi bir PE dosyası yüklediğinde, bölüm tablosunda belirtilen adres, boyut ve bayraklarla birlikte her bölüm için sanal bellek blokları oluşturur ve verileri dosyadan bu bellek bölümlerine kopyalar. İki özel bölüm var: kod ve veri. Kod ve veri bölümleri, PE başlığındaki BaseOfCode ve BaseOfData alanları tarafından tanımlanır ve sırasıyla CS ve DS segment kayıtlarına atanır. EIP kaydı daha sonra PE başlığındaki AddressOfEntryPoint alanında belirtilen değere (ya da 64-bit olarak kopyala) ayarlanır ve program başlatılır.

Her bölüm, ne tür veriler içerdiği ("initialized data" vb.), paylaşılıp paylaşılamayacağı gibi ve verilerin kendisiyle ilgili bazı bayrakları içerir. Çoğu bölüm, dışa aktarılan fonksiyonların dizini veya temel yer değiştirme dizini gibi, optional header'in "Data Directory" dizisinin girişlerinde başvurulan bir veya daha fazla dizin içerir. Yönetimsiz içerik türleri, örnek olarak "executable code" veya "initialized data" dır. Dosya formatı hakkında daha fazla ayrıntı [30] ve [31] 'de bulunabilir.

Ayrıca, dosyanın tüm bölümlerinin belleğe eşlendiğini bilmemiz gerekir; bazı parçalar imaj yüklemeyi kolaylaştırır ancak bellekten yürütülmesi için gerekli değildir. Eşlenmemiş kısımlar dosyanın sonuna yerleştirilir.

2.4. Zararlı Yazılım Analizi ve Tespiti

Zararlı yazılım analizi, virüs, solucan veya Truva atı gibi belirli bir zararlı yazılım örneğinin amacını ve işlevselliğini belirleme işlemidir. Bu işlem, etkileyici zararlı kod algılama teknikleri geliştirebilmek için gerekli bir adımdır. Ek olarak, zararlı yazılımı virüslü bir makineden tamamen silebilen temizleme araçlarının geliştirilmesi için önemli bir ön koşuldur. Geleneksel olarak, zararlı yazılım analizi, sıkıcı ve zaman alıcı bir süreçtir.

Statik analiz çalıştırılabilir dosyayı çalıştırmadan incelemeyi içerir. Bunlar Taşınabilir Yürütülebilir (PE) dosyasının öznitelikleri (veya özellikleri) (Microsoft'un DLL ve çalıştırılabilir biçimleri) veya çalıştırılabilir dosyanın içe aktardığı Fonksiyonların listesi gibi yürütülebilir dosyalardan elde edilen statik bilgiler olabilir. Bu bilgiyi toplayarak, bir zararlı yazılım uzmanı, orijinal dosyanın zararlı olup olmadığına karar verebilir. Yapıdan toplanan bilgilerin yeterli olmaması durumunda, yürütülebilir dosyayı tersine mühendislik yapılması, küçük parçalara ayrıştırılması ve bir kez daha analiz edilmesi gerekir.

Statik analiz çeşitli teknikleri içerebilir [32]:

1) Dosya Biçimi Kontrolü: dosya meta verileri faydalı bilgiler sağlayabilir. Örneğin, Windows PE (taşınabilir yürütülebilir dosyalar), derleme süresi, içe aktarılan ve verilen fonksiyonlar vb. Hakkında çok fazla bilgi sağlayabilir.

2) Dize Çıkarma: bu, yazılım çıktısının incelenmesini (örneğin durum veya hata mesajlarını) ve zararlı yazılımın çalışmasıyla ilgili bilgileri çıkarmayı ifade eder.

3) Parmak İzi Kontrolu: Bu, şifreleme karma hesaplamasını ve kodlanmış kullanıcı adı, dosya adı, kayıt dizeleri gibi çevresel yapay doku bulunmasını içerir.

4) Antivirüs Taraması: Eğer kontrol edilen dosya tanınmış bir zararlı yazılım ise, büyük olasılıkla tüm antivirüs tarayıcıları tespit edebilir. Her ne kadar alakasız

gibi görünse de, bu tespit yöntemi genellikle antivirüs satıcıları veya sanal alanlar tarafından sonuçlarını "doğrulamak" için kullanılır.

5) Dağılım: bu, makine kodunun derleme diline çevrilmesini ve yazılım mantığını ve niyetlerini çıkarmayı ifade eder. Bu, statik analizin en yaygın ve güvenilir yöntemidir.

Statik analiz için yürütülebilir dosyayı çalıştırmamız gerekmediğinden, bilgilerin çıkarılması dinamik analizle karşılaştırıldığında hızlıdır. Ancak, statik analizi kullandığında, bazı bilgiler ve analizler orta çıkmamaktadır.

Dinamik analiz dosyayı sözde sanal alanda (sandbox) çalıştırmayı içerir. Sandbox proses ön işlemlerinin kaydını tutan bir sanal ortamdır. Bir sandbox kullanmanın nedenleri, analiz üzerinde daha fazla kontrol sahibi olurken ana makineyi temiz tutmaktır. Analistin temel görevi, çalıştırılabilir dosya tarafından gerçekleştirilen eylemlere ilişkin gözlemlerine dayanarak, mevcut tehdit ve kötülükleri değerlendirmektir. Ancak, bu yöntemin bazı sakıncaları vardır. İlk olarak, zararlı yazılımların tümü, başlatıldıktan hemen sonra zararlı etkinliklerini yürütmemektedir. Bu nedenle, bu yöntemler genel olarak zaman alıcıdır. Bunun dışında, sanallaştırılmış ortamların keşfi için yöntemler vardır. Zararlı yazılımlar, bir sanal alanda çalıştırıldıklarını algıladıklarında, davranışlarını tamamen değiştirebilir ve iyi tehlikesiz gibi davranabilirler. Bu nedenle, dinamik analiz ile tespitten kaçınabilirler.

Statik ve dinamik zararlı yazılım analizi yöntemlerinin her ikisinin de önemli avantaj ve dezavantajları vardır. Dezavantajları ortadan kaldırmak için hibrit analiz yöntemleri önerilmiştir. Statik ve dinamik yöntemlerin herhangi bir kombinasyonu, hibrit zararlı yazılım analizi olarak adlandırılabilir.

Bu çalışmada, zararlı yazılım analizini, yazılımların zararlı olup olmadığını belirlenmesine yardımcı olmak için yürütülebilir dosya analiz etme işlemi olarak tanımlanabilir.

Yazılımın kötü niyetli olma konusunda karar vermek için, çalıştırılabilir dosyanın ne olduğu hakkında iyi bir görüş sahibi olmamız gerekir. Bu görüşü kazanmak için, zararlı yazılım analizi yöntemleri kullanılır. Bu yöntemler, analiz süreçlerinde izlenen ve değerlendirilen izlere dayanarak üç kategoriye ayrılır. Kategorilemek genellikle bir kaynaktan diğerine farklılık gösterir. Temel olarak kabul görmüş iki analiz tekniği vardır: Statik analiz ve dinamik analiz [21] ve ayrıca bu iki teknikten oluşan bir Hibrit analiz olarak bir kombinasyon da söz konusudur [33].

Zararlı yazılım analiz yöntemlerinin tüm fikirleri ve kavramları ile özdeşleştirildikten sonra, Yürütülebilir dosyanın kötü niyetli olup olmadığına, karar verme aşamasına geçilebilir. Bu aşamaya zararlı yazılım algılama adı verilir ve genel olarak kararsız sayılır. [21] 'te, Chess and White bunu teorik olarak ispatlamışlardır. Zararlı yazılım algılama, sağlam sonuçlar elde edebilir ve bilgisayar sistemlerinin korunmasına yardımcı olabilir. Ayrıca, zararlı yazılım açıkça tanımlanmamıştır. [34] 'de, zararlı yazılımın bilgisayar virolojisindeki açık sorunlardan biri olduğu belirtilmiştir.

İmza tabanlı yaklaşımı antivirüs satıcıların çoğu tarafından kullanılan en popüler yaklaşımdır [35]. Yeni bir zararlı yazılımın büyük ölçüde yayılmasından sonra, antivirüs şirketlerinin uzmanları, numunelerini araştırmak için numune alır ve etkin bir şekilde her zararlı yazılım örneği için benzersiz olan talimatlarının bir sırası olan bir imza ortaya çıkartırır ve ardından veritabanına imza eklenir. İmzayı içeren bir program göründüğünde, antivirüs yazılımı onu zararlı yazılım olarak tanımlar. İmza Tabanlı Metodoloji bilinen zararlı yazılımları düşük yanlış pozitif oranı (tehlikesiz bir dosyayı yanlışla tehlikeli olarak göstermek, 2.6.2.'de detaylandı) ile tespit etmek için çok etkilidir.

Yine de bu yöntemin zayıflıkları vardır: sınırlı imza veri tabanı, imzaların boyutu ve imzaların eşleştirme süresi katlanarak arttığından dolayı yeni zararlı yazılım veya bilinmeyen zararlı yazılımların tespit edilmesi zorlaşmaktadır. Son olarak, yeni (görünmeyen) bir zararlı yazılımın ortaya çıkması ile virüsten koruma istemcilerinin imzalarının güncellenmesi arasındaki dönemde, milyonlarca bilgisayar yeni zararlı yazılımlara maruz kalmaktadır.

Zararlı yazılım tespiti için makine öğrenimi kullanmak, imza temelli yöntemin dezavantajları göz önüne alındığında doğaldır. Görünmeyen numunelerin özelliklerini tahmin etme yeteneğinin güçlü olmasından dolayı, bilgisayar güvenliği araştırmacıları da bu güçlü araçtan yararlanmış ve zararlı yazılım tespiti için makine öğrenme algoritmaları kullanmaya başlamışlardır. 2001 yılında Schultz ve arkadaşlarının çalışması, [5], genel olarak zararlı yazılım algılamasında makine öğreniminin literatürdeki ilk kullanımı olarak kabul edilir. Bu yaklaşımda, algılama modeli, eğitim verilerindeki makine öğrenme algoritmaları tarafından eğitilir ve ardından algılama için kullanılır. İmza tabanlı bir yaklaşıma kıyasla, makine öğrenme yöntemleri daha yüksek bir yanlış pozitif orana sahip olma eğilimindedir. Ancak, tahmine dayalı modeller sayesinde, bu yöntemlerin, model tarafından daha önce görülmeyen zararlı yazılımları tanıyabildiğini ve her örneği manuel olarak analiz edilmesine gerek olmadığını kanıtlamıştır. Bu nedenle, şüpheli yürütülebilir dosyalar düzgün bir şekilde analiz edilip imza veri tabanına eklenene kadar, makine öğrenimi modelleri iyi bir ilk savunma seviyesi sağlayabilir. Schultz ve arkadaşlarının çalışmasından günümüze kadar, API çağrı dizileri, makine kodu talimatları üzerindeki bayt seviyesi n-gramlar ve zararlı yazılım tespiti için PE başlıkları verileri gibi farklı araştırmacılar tarafından birçok özellik çıkarma tekniği önerilmiştir. Dosyaları zararlı veya iyi niyetli olarak sınıflandırmak için makine öğrenimi kullanırken, başta eğitim için etiketli bir veri seti oluşturulmalıdır. Dosyanın özelliklerini elde etmek için, dosyanın belirli özellikleri kullanılır, sonra da her dosya iyi veya kötü niyetli olarak atanır. Makine öğrenmeye dayalı zararlı yazılım algılama algoritmaları genellikle üç ana işlemden oluşur, i.e. özellik çıkarma, özellik seçimi ve sınıflandırma. Özellik çıkarma işlemi, orijinal programları özellik setine dönüştürür, özellik seçimi, en iyi özelliklerle birlikte gelecek şekilde ayarlanan özelliklerin boyutunu azaltmak için yapılır ve daha sonra sınıflandırma işlemi, eğitim setinde indirgenmiş özellik üzerinde bir ikili sınıflandırma algoritması geliştirerek, dosya özellikleri ve etiketlerin ilişkisini eşler. Öğrenme algoritması, yeni dosyanın (görünmeyen) iyi niyetli olup olmadığını tahmin etmek için kullanılır.

2.5. Bilgi Çıkarma (Information Retrieval)

Belirli bir ihtiyacı karşılamak için bir kaynaktan faydalı bilgi veya kaynak toplama genel etkinliği Bilgi Edinme olarak bilinir. Bilgi, metin, belgeler, görüntüler, ses veya video gibi herhangi bir biçimde görünebilir. Bilgi edinme, Doğal Dil İşleme (NLP) 'nin en çok kullanılan uygulamalarından biridir.

[36] 'ya göre, bilgi edinme, büyük koleksiyonlardan bilgi ihtiyacını karşılayan, yapılandırılmamış nitelikte bir materyal bulmak demektir. Kısaca, bizim durumumuzda, yapılandırılmamış verileri (açık, anlamsal olarak açık olmayan, bilgisayar için kolay olmayan bir yapıya sahip olan verileri) yapılandırılmış, metinleri bir sınıflandırıcı girişi olarak kullanılabilecek özelliklere dönüştürmenin yollarını bulur.

Aşağıdaki Alt bölümler, metinlere uygulanan bilgi erişimini içeren adımları sunmayı amaçlamaktadır.

a) Vaka Katlama ve Normalleştirme (Case Folding and Normalization): Ön işleme metinlerinde ilk adım, karşılaştırma için bir standardı korumak adına tüm metinleri küçük harf olarak tutmayı hedefleyen harf katlama işlemidir. Bundan sonra, normalleştirme özel karakterleri, aksanları, işaretleri ve sayıları kaldırmayı amaçlar. Bu son adım, sembollerin ve sayıların adlarıyla değiştirilmesi gibi çeşitli kuralları izleyebilir.

b) Engellenecek Kelimeleri Kaldırma (Stopwords Removal): Metni normalleştirdikten sonra, durma sözcüklerini, bir metinde temel anlamları olmayan kelimeleri, genellikle bir dilde en yaygın kullanılan kelimeleri silmek gerekir. Bununla, onlar birbirinden ayrılmaya katkıda bulunmadıklarından tamamen kelime haznesi dışında tutulurlar.

c) Kelimelerin Torbası (Bag of Words): Kelimelerin torbası, doğal dil işlemede kullanılan ve genellikle belgelerin sınıflandırılmasında kullanılan, her bir terimin oluşum sayısını sayan bir bilgidir [36]. Yapılandırılmış bir veri olmasına rağmen,

kelime çantası modeli sınıflandırma görevleri için metinleri ayırmak için yeterli değildir. Bu gösterim, bu durumda, yalnızca sonraki bölümde sunulan Vector Space Modeli'nde kullanılan her iki terimi de hesaplamak için kullanılır.

d) Vektör Uzay Modelleri (Vector Space Model): Bir vektör uzayı, her vektörün doğrusal olarak bağımsız olduğu V boyutları için bir V temel vektör kümesidir. Bilgi Edinme'de, herhangi bir metin, genel olarak, varlığını açıklayan vektörlerde temsil edilebilir. Vektör Uzay Modelleri [37] olarak da bilinen Vektör Modelleri terimi, her terimin kelime haznesinde kendi boyutu olarak görünmesini sağlayarak belgeleri vektörler kullanarak temsil eden cebirsel modellerdir. Bu, metnin bir sorgu anahtar sözcükleri ve bir belgedeki terimlerin terim vektörlerine dönüştürülmesini sağlar.

Teoriyi bu bölümün anlaşılmasını kolaylaştırmak için dört belgeden oluşan aşağıdaki örnek korpus eşlik eder (Tablo 2.1.):

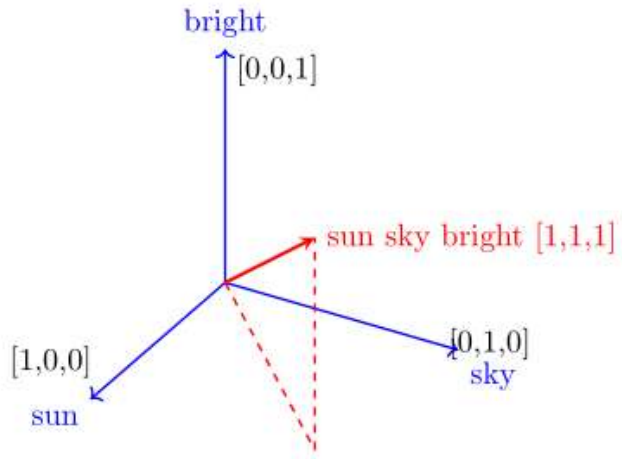
Tablo 2.1. Benzer terimlere sahip birden fazla belge örneği

Docs	Terms	Terms (w/o stopwords)
d_1	The sky is blue	sky blue
d_2	The sun is bright today	sun bright
d_3	The sun in the sky is bright	sun sky bright
d_4	We can see the shining sun, the bright sun	shining sun, bright sun

Tablo 2.1.'de bulunan değerleri göz önünde bulundurduğunuzda her değer için her terimi sayarak, bir terim vektörü oluşturulabilir. Her deger farklı olduğundan, karşılık gelen vektörler sonuç olarak farklı yönlere işaret edecektir. Benzer şekilde, iki belge eşit olsaydı, tam olarak aynı yönde bir nokta oluşurdu. Bu, her temel vektörün, belirtilen terimin belgelerde varlığına bağlı olmasının bir sonucudur.

Yukarıdaki terimlerin çoğunun herhangi bir bilgi vermediği kabul edilmektedir. *"the", "is", "in"* vs. gibi terimler, İngilizce'deki en yaygın kelimelerden bazılarıdır ve durma kelimeler olarak bilinir. Durdurma sözcükleri bir süredir çevrededir ve içeriğe bağlı olarak, birçok arama motorunun performansını iyileştirmek için doğal dil metni [38] işlemeden önce filtrelenecek kelimeler olarak

tanımlanmıştır. Bazı durumlarda, bu kelimeler önemlidir ve örneğin bir kullanıcı *"The Who"* için bir arama yaptığında filtrelenmemelidir. Tablo 2.1.'deki belgelerde yer alan önemli terimlerin vurgulanması amacıyla, kelime kaldırmayı durdurma işlemi gerçekleştirir. Bir kelimenin muhtemel varyasyonlarının sayısını azaltmak için uygulanabilir bir başka prosedür stemming'i içerir. Bu, özellikle kullanıcı sorgusunu doğal metin olarak yazdığınızda ortaya çıkabilecek ayrıntılı sorgular için kullanışlıdır [39]. Şekil 2.5.'te Tablo 2.1.'deki d3 belgesi için bir terim vektörünün nasıl görüntüleneceği gösterilmiştir.



Şekil 2.5. "sun sky bright" için üç boyutlu terim vektör

Bu vektörler, belgeler arasındaki ve belge ile sorgu arasındaki benzerlik için ölçülebilir. Kosinüs benzerliği [39], her iki terimin varlığına dayanarak iki terimli vektörlerin ne kadar benzer olduğunu gösteren bir ölçüdür. Bir kullanıcı *"bright sky"* üzerinde bir anahtar kelime araması yaparsa, sorguda görünen terimlere dayalı olarak benzer bir vektör oluşturulabilir. Daha sonra, bu iki vektör aşağıdaki fonksiyon kullanılarak karşılaştırılabilir:

$$cos(\theta_{q,d}) = \frac{\sum_{j=1}^{n} term_{q,j} \cdot term_{d,j}}{\sqrt{\sum_{j=1}^{n} term_{q,j}^2} \cdot \sqrt{\sum_{j=1}^{n} term_{d,j}^2}} \tag{2.1}$$

$\theta_{q,d}$, q sorgusu ile d belgesi arasındaki açıyı ve q, j ve d terimlerini temsil eden açıyı belirtir, j, sırasıyla q ve d için vektörlerin bileşenlerini temsil eder. Bu ölçüm, 0 ile 1 arasında bir değer verir; burada daha yüksek bir değer daha yüksek benzerliği gösterir, 1 ile iki vektör aynıdır.

Yukarıdaki örnekte, birkaç belirgin terim olduğu için durum önemsizdir ve sadece kosinüs benzerliği kullanarak sorgular üzerinde basit bir karşılaştırma yapılabilir. Ancak, belge koleksiyonu daha da büyük bir kelime dağarcığı olan gerçekçi bir boyutta olduğunda bu düşük performans gösterir.

Topikal alaka düzeyi, bir belgede hangi terimlerin sık olduğunu ve bu belgede hangi terimlerin diğer belgelere göre daha fazla göründüğünü belirleyerek tahmin edilebilir. Bu, Terim Frekansı (TF) ve Ters Belge Frekansı (IDF) [40] kullanılarak dokümanları ağırlıklandırarak yapılabilir. Frekans terimi şu şekilde yazılabilir:

$$TF(t,d) = \frac{f_{t,d}}{|d|} \tag{2.2}$$

$f_{t,d}$, t teriminin d ve | d | belgelerinde görünme sayısıdır. d cinsinden terimlerin sayısını belirtir. Ters belge sıklığı şöyle tanımlanır:

$$idf_{t,D} = log(\frac{|D|}{|d \in D : t \in d|}) \tag{2.3}$$

D belge koleksiyonu, ve $|d \in D : t \in d|$, t teriminin tüm belge koleksiyonunda görünme sayısını belirtir. Paydayı 1 artırmak, sıfıra bölmeyi önleyecektir.

TF-IDF, en popüler ağırlıklandırma şemalarından biri olarak yaygın şekilde kullanılan bir ölçüdür. Bu önlem, dijital kütüphanelerin metin bazlı öneri sistemlerinin yaklaşık %83'ünde kullanılmaktadır [40].

Her bir terimi bir belgede saymak ve terim vektöründe satır sayısını kullanmak yerine, TF-IDF ağırlıklandırması, belgeye özgü terimleri içeren belgeleri, ortak terimlere sahip belgelerden ayırmak için yapılır. TF için daha yüksek değer, terimin daha sık olduğunu, IDF için daha yüksek değer ise belge koleksiyonunda terimin

nadir olduğunu gösterir. Birleştirilmiş TF-IDF için yüksek bir değer, yüksek vadeli özgüllüğü belirtir.

Tablo 2.1.'deki belgelerde her bir terim için Terim Frekansının Hesaplanması:

Tablo 2.2. Terim Frekanslarının Hesaplanması Örneği

	Term Frequency			
Terms	d_1	d_2	d_3	d_4
Sky	0.5	0	0.33	0
Blue	0.5	0	0	0
bright	0	0.5	0.33	0.25
Sun	0	0.5	0.33	0.5
Shining	0	0	0	0.25

Ayrıca, her terimin IDF'sini hesaplayabiliriz:

Tablo 2.3. Ters Belge Frekanslarının Hesaplanması Örneği

Terms	Sky	Blue	Bright	sun	shining
IDF	0.30	0.60	0.125	0.125	0.60

Tablo 2.2.'deki frekans terimlerinin birleştirilmesi ve Tablo 2.3.'teki ters belge frekansları ile çarpılması:

Tablo 2.4. TF-IDF'nin hesaplanmasına örnek

	Term Frequency			
Terms	d_1	d_2	d_3	d_4
Sky	0.150	0	0.100	0
Blue	0.300	0	0	0
Bright	0	0.063	0.043	0.031
Sun	0	0.063	0.043	0.063
Shining	0	0	0	0.150

Ağırlık şeması uygulandığında, d3 vektörü terimi (0.043, 0.1, 0.043) olur. Özet olarak, her bir metin/belge, sözcükteki her bir kelime için TF-IDF ölçümlerini içeren seyrek bir vektörle temsil edilir. Kelime, verilen belgelerde veya metinlerde en fazla bulunan bir dizi V kelimesine indirgenebilir.

e) Özelliklerin Normalizasyonu: Özellikler ölçeğinde farklılıklar olduğu için, onları normalleştirme ihtiyacı vardır. Bu çalışmanın çıkarılan özelliklerine uygulanan normalizasyon tekniği Min-Max'tır. Min-Max tekniği Z-puan normalleşmesine alternatif bir yaklaşımdır. Bu teknik, her özelliği (karakteristik) Denklem 2.4'teki formülü kullanarak sıfır ile bir arasındaki bir aralıkta ölçeklendirir; bu sınırlandırılmış aralığa sahip olmanın maliyeti - standardizasyonun aksine - daha küçük standart sapmalarla sonuçlanacaktır. Aykırı değerlerin etkisi şu şekildedir [41].

$$z_i = \frac{x_i - min(x)}{max(x) - min(x)} \tag{2.4}$$

2.6. Makine Öğrenmesi

Bilgi almak veya görevleri gerçekleştirmek için açıkça programlama algoritmalarının bazı uygulamalarda sınırlamaları vardır. Makine öğrenimi, “bilgisayarların açıkça programlanmadan öğrenmelerini sağlayan” bir bilgisayar bilimi alt alanıdır [42]. Makine öğrenmesi, yapay zekâ ile örtüşen özelliklere sahiptir, burada makinelere akıllı davranış ilkelerini verilir.

Makine öğreniminin alt alanları denetimli öğrenmeyi, denetimsiz öğrenmeyi ve güçlendirme öğrenmesini içerir. Denetimli öğrenme, nihai hedefe uyan kuralları öğrenmek için örnek girdi (öngörücüler) ve istenen çıktı çiftleriyle bir makine sağlama olarak tanımlanabilir [43]. Bu, önceden tanımlanmış bir amaç ile sistemle etkileşime giren bir insana dayanır. Öte yandan, denetimsiz öğrenme bir insan kullanıcısı gerektirmez ve açıkça görülmeyen modelleri keşfetme potansiyeli yüksek olan veri noktaları arasında ilişkisel kurallar oluşturmaktan sorumludur [44]. Son olarak,

pekiştirici öğrenme, genellikle deneme yanılma ile bir sorunun ara adımlarına odaklanan bir öğrenme yöntemidir [45]. Belirli bir hedefe ulaşma olasılığını artıran eylemler ödüllendirilir ve aynı olasılığı düşüren muhalif eylemler ceza alır. Ödüller ve cezalar nihayetinde, en iyi sonuçlar için en üst seviyeye çıkarılması gereken kümülatif bir puan (yani, uygunluk ölçüsü) oluşturur. Bu tanımlara göre tez çalışmasında denetimli öğrenmeye odaklanılmıştır.

Modern zararlı yazılımlarda kullanılan antivirüs önleme tekniklerine potansiyel olarak ayak uydurabilmenin bir yolu, mevcut algılama sistemlerini makine öğrenme sınıflandırıcılarıyla güçlendirmektir. Makine öğrenme sınıflandırma algoritmaları, örnek verilerden öğrenerek, zararlı yazılım ve iyi niyetli dosyalar gibi her bir sınıfın özelliklerini otomatik olarak öğrenen sınıflayıcıların oluşturulmasını kolaylaştırır. Bu tür bir yaklaşım, zararlı yazılım tespitinde hala geliştirici olmakla birlikte, yeni uyarlanmış tehditler karşısında hafifçe değiştirilmiş, ancak geçmiş zararlı yazılımın özelliklerini koruyacak şekilde artan sağlamlık vaadi sunar.

Makine öğrenmesini PE dosyalarını zararlı veya zararsız olarak sınıflandırmak amacıyla kullanmak için, önce eğitim için etiketli veri setleri oluşturulmalıdır. Her zararsız ve zararlı dosya için bir kaydı olan bir veri seti içerir. Her kayıtta bir dizi özellik ve bir etiket (ayrıca sınıf olarak da adlandırılır) bulunur. Her dosyanın özellikleri, dosyanın boyutu veya dosyadaki belirli bir kod parçasının sıklığı gibi, dosyanın belirli özelliklerinden kaynaklanır; etiket, dosyanın zararlı olup olmadığını gösteren ikili bir değerdir. Öğrenme algoritmaları, dosya özellikleri ve etiketler arasındaki ilişkiyi haritalayan matematiksel bir model oluşturmak için eğitim için belirlenmiş kayıtları analiz eder. Sınıflandırıcı adı verilen bu model, test verilerindeki her bir kaydın sınıfını veya test için belirlenen kayıtları tahmin etmek için kullanılır. Sınıflandırıcı, öngörülerde bulunurken etiketleri okuyamaz; Test verileri etiketleri yalnızca sonraki performans analizinde tahminler gerçek etiketlerle karşılaştırıldığında kullanılır.

Veriler önceden işlendikten sonra, bir sonraki adım, sınıflandırma için uygun olan numunelerden bu özellikleri (veya özellikleri) seçmektir. Başka bir deyişle, en iyi

karar verme özelliğine sahip özelliklerin bir alt kümesini seçme işlemidir. Bu adım oldukça zordur, [46] 'de belirtildiği gibi, *n* büyüklüğünden seçilen *n* büyüklüğündeki olası altkümelerin sayısı *m*'dir.

$$\frac{m!}{(m-n)!\,n!} \tag{2.5}$$

Belirli bir alt kümenin kalitesini değerlendirme metriği de önemsiz olamayacağından, bu sayı oldukça düşük n ve m için bile tatmin edici bir şekilde hesaplanabileceğinden fazladır. Bu nedenle, bu sorunu çözmek için çeşitli sezgisel yaklaşımlar verilmiştir.

1) Univariate Feature Selection: Tek değişkenli özellik seçimi, tek değişkenli istatistiksel testlere dayanarak en iyi özellikleri seçerek çalışır. Bir tahmincinin ön işleme adımı olarak görülebilir [47]. Scikit-learn, dönüştürme yöntemini uygulayan nesneler olarak özellik seçim yordamlarını sunar:

   - SelectKBest en yüksek puanlama özelliklerinden başka tümünü kaldırır.
   - SelectPercentile, yalnızca kullanıcı tarafından belirtilen en yüksek puan yüzdesi özelliklerinin hepsini kaldırır.
   - Her özellik için ortak tek değişkenli istatistiksel testler kullanılması işlemi: SelectFpr; yanlış pozitif oran, SelectFdr; yanlış keşif oranı veya SelectFwe; aile bilge hatası.
   - GenericUnivariateSelect, yapılandırılabilir bir strateji ile tek değişkenli özellik seçimi yapmanıza izin verir. Bu, hiper-parametre arama tahmincisi ile en iyi tek değişkenli seçim stratejisinin seçilmesine izin verir.

   Bu tez çalışmasında tek değişkenli özelliklerden SelectKBest kullanılmıştır.

2) Recursive Feature Elimination (RFE): En iyi performans gösteren özellik alt kümesini bulmayı amaçlayan açgözlü bir optimizasyon algoritmasıdır.

   Sürekli olarak modeller oluşturur ve her bir yinelemede en iyi veya en kötü performans özelliğini bir yana bırakır. Tüm özellikleri tükenene kadar bir

sonraki modeli son özelliklerle inşa eder. Daha sonra özellikleri eleme sırasına göre sıralar [47].

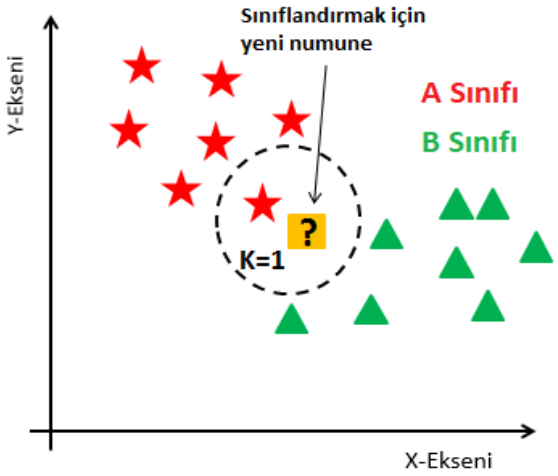
3) L1-based Feature Selection (L1): L1 cezalı tahmin yöntemleri, regresyon katsayılarının tahminlerini, maksimum olabilirlik tahminlerine göre sıfıra doğru küçültür. Bu büzülmenin amacı, değişkenlerin eşdüzeyliği veya yüksek boyutluluk nedeniyle ortaya çıkan fazla çabayı önlemektir. Bir L1 cezası uygulanması, birçok regresyon katsayısının tam olarak sıfıra küçülmesine ve nispeten daha az büzüşen birkaç diğer regresyon katsayısına neden olma eğilimindedir.

   L1 normu ile cezalandırılan lineer modeller seyrek çözümlere sahiptir: tahmini katsayılarının çoğu sıfırdır. Amaç, başka bir sınıflandırıcı ile kullanılacak verilerin boyutsallığını azaltmak olduğunda, sıfır olmayan katsayıları seçmek için *feature_selection.SelectFromModel* ile birlikte kullanılabilir [47].

4) Tree-based Feature Selection (TB): Tree-based tahmin ediciler, önemsiz özellikleri atmak için kullanılabilecek özellik önemini hesaplamak için kullanılabilir (sklearn.feature_selection.SelectFromModel meta-transformatörü ile birleştiğinde) [47]. Bu tez çalışmasında, özellik seçimi için ExtraTree algoritmasını kullanılmıştır.

2.6.1. Sınıflandırıma algoritmaları

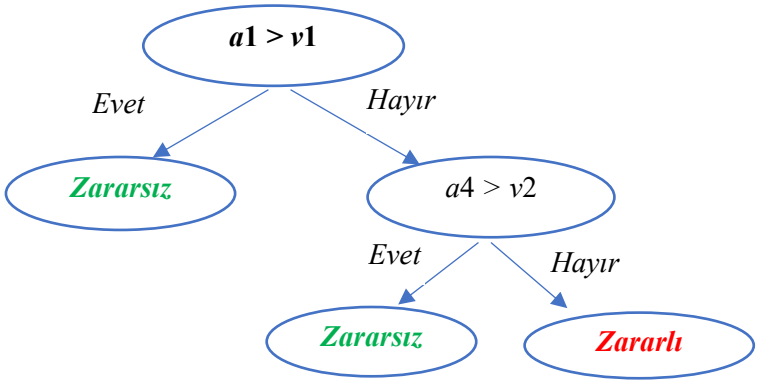
a) K-En Yakın Komşular (K-Nearest Neighbours): K'ye en yakın komşular (K-NN) daha basit sınıflandırıcılardan biridir. Eğitim aşaması oldukça basittir ve eğitim veri setinin depolanmasından oluşur. K-NN sınıflandırıcısı, en yakın komşuları bulmak için hesaplanan ve sınıflandırmada ele alınan komşuların sayısını tanımlayan K parametresi ile hesaplanan eğitim seti ve mesafe ölçüsü ile tanımlanır. Sınıflandırma işlemi, sınıflandırılmış numunenin, eğitim aşamasında saklanan diğer tüm numunelere olan mesafesinin hesaplanmasından ibarettir [48].



Şekil 2.6. Sınıf, en yakın komşulardan hesaplanır.

b) Karar Ağacı (Decision Tree): Karar ağacı en basit sınıflandırıcılardan biri olarak tanımlanmaktadır [49]. Quinlan tarafından tanıtılan [50], oldukça uzun bir süredir kullanılmaktadır. İç düğümlerdeki koşulları ve yaprak düğümlerinde kararları içeren hiyerarşik bir ağaç yapısıdır.

Bir örnek bir ağaç tarafından sınıflandırıldığında, ulaştığı düğümün durumu değerlendirilir ve sonuçlara göre karşılık gelen kenar izlenir. Bir yaprak düğümüne ne zaman ulaşılırsa, sınıflandırma işlemi sona erer ve numuneye verilen yaprağa ait bir sınıfa atanır. Tek bir karar ağacının örneği Şekil 2.7.'de verilmiştir. Ağaç yapısı oluşturmak için kullanılan algoritmaları ve bunların sınıflandırma için nasıl kullanılacağını açıklanacaktır.

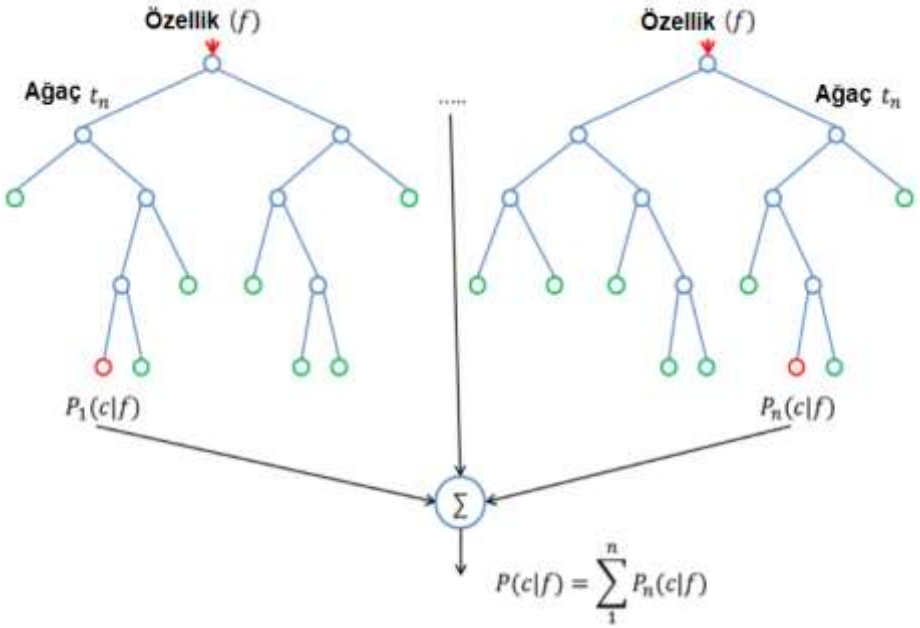


Şekil 2.7. Tespit için kullanılan eğitimli karar ağacının basit bir örneği

c) Random Forest (Rastgele Orman): Rastgele Orman algoritması denetimli bir sınıflandırma algoritmasıdır. Onu bir şekilde orman yaratan ve rastgele yapan adından görebiliyoruz. Ormandaki ağaç sayısı ile elde edebileceği sonuçlar arasında doğrudan bir ilişki vardır: ağaç sayısı büyüdükçe, sonuç daha doğru olur. Ancak dikkat edilmesi gereken, orman oluşturmanın kararın bilgi kazancı veya kazanım endeksi yaklaşımıyla yapılandırılması ile aynı olmadığıdır.

Rastgele Orman algoritması ile karar ağacı algoritması arasındaki fark, Rastgele Orman'da kök düğümü bulma ve özellik düğümlerini bölme işlemlerinin rastgele çalışacağıdır [51].

Basit kelimelerle söylemek gerekirse: Rastgele orman birden fazla karar ağacı oluşturur ve daha doğru ve istikrarlı bir tahmin elde etmek için onları birleştirir. Aşağıda rastgele ormanın iki ağaçla nasıl görüneceğini görebilirsiniz:



Şekil 2.8. İki ağaçlı rastgele orman örneği [51]

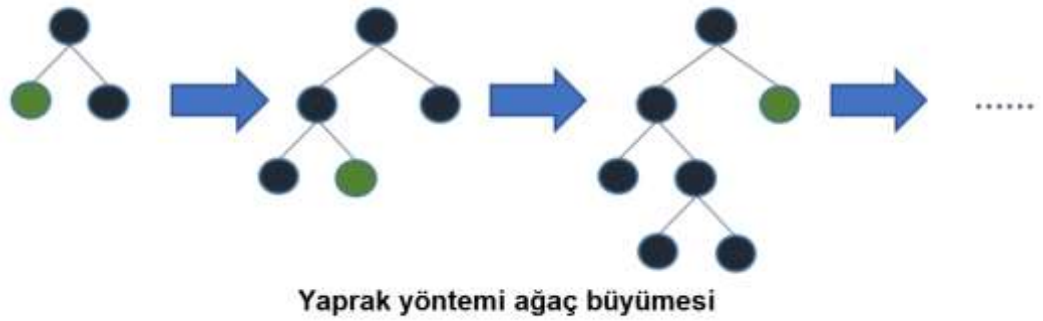
d) Light GBM (Hafif Gradyan Artırma Makinası): Light GBM nispeten yeni bir algoritmadır ve internette dokümantasyonu dışında pek çok okuma kaynağı yoktur [52]. LightGBM, ağaç tabanlı öğrenme algoritmaları kullanan bir gradyanı artıran

bir çerçevedir. Aşağıdaki avantajlarla dağıtılacak ve verimli olacak şekilde tasarlanmıştır:

- Daha hızlı eğitim hızı ve daha yüksek verimlilik
- Düşük bellek kullanımı
- Daha iyi doğruluk
- Paralel ve GPU öğrenmesi desteklenir
- Büyük ölçekli veri işleme yeteneği

Light GBM'in diğer ağaç tabanlı algoritmalardan farkı , Light GBM dikey olarak ağaç gibi büyürken, diğer algoritmalar yatay ağaç şeklinde büyümektedir. Ayrıca, Light GBM'nin ağaçın ilgili yollardaki tüm düğümlerine (leaf-wise) erilebilmekteyken, diğer algoritmalar alt seviyelere doğru (level-wise) büyümektedir. Büyümesi için maksimum delta kaybı olan yaprağı seçecektir. Aynı yaprağını büyütürken, Leaf-Wise algoritması seviye-seviyeli bir algoritmaya göre daha fazla kaybı azaltabilir [52].

Aşağıdaki diyagramlar Light GBM ve diğer yükseltme algoritmalarının uygulanmasını açıklamaktadır.



Şekil 2.9. LightGBM'nin çalışması [53]

2.6.2. Sınıflandırma performans ölçütleri

Model eğitildikten sonra, test veri setinde test edilir ve sonuçlar kaydedilir. Sınıflandırıcının verilen bir test veri setinde ne kadar iyi performans gösterdiğini

değerlendirmek için birkaç ölçüm kullanılır. Bu bölümde, performans değerlendirmemizde kullandıklarımızı açıklayacağız.

Tespit işleminin dört olası sonucu vardır:

- Gerçek pozitif (True Pozitive — TP): Zararlı yazılım olarak doğru bir şekilde sınıflandırılmış Zararlı PE dosyası
- Gerçek negatif (True Negative — TN): Zararsız PE dosyası doğru şekilde benign olarak sınıflandırıldı
- Yanlış pozitif (False Positive — FP): Zararlı yazılım olarak yanlış sınıflandırılmış zararsız PE dosyası
- Yanlış negatif (False Negative — FN): Zararlı PE dosyası yanlış bir şekilde zararsız olarak sınıflandırıldı

Tablo 2.5. Karışıklık matrisi

		Olarak Sınıflandırılmış	
		Zararlı yazılım	Zararsız yazılım
Gerçekte	Zararlı yazılım	TP	FP
	Zararsız	FN	TN

Bu değerlerden yanlış pozitif (FPR) ve yanlış negatif oranı (FNR) hesaplanabilir. Yanlış pozitif oran, zararlı yazılım olarak yanlış bir şekilde etiketlenmiş zararsız dosyaların test setindeki tüm zararsız dosyalara oranı, yanlış negatif oran ise doğru şekilde test setindeki tüm zararlı yazılım dosyalarına sınıflandırılmış zararlı yazılım dosyalarının oranıdır. Bu sırasıyla (2.6) ve (2.7) denkleminde gösterilmiştir.

$$FNR = \frac{FN}{TP + FN} \quad (2.6)$$

$$FPR = \frac{FP}{FP + TN} \quad (2.7)$$

Doğruluk (Accuracy)

Başka bir değerlendirme ölçütü doğruluk (ACC) olarak adlandırılır. Doğruluk, sınıflandırıcının genel güvenilirliğini sağlar. Önceki metriklerde olduğu gibi, sadece

belirli bir sınıftakileri değil tüm dosyaları kapsar. Doğruluk, doğru sınıflandırılmış dosyaların tüm dosyalara oranıdır.

$$Acc = \frac{TP + TN}{TP + FN + TN + FP} \tag{2.8}$$

Kesinlik (Precision)

Sınıflandırıcı tarafından tahmin edilen pozitif sonuç sayısına bölünen doğru pozitif sonuçların sayısıdır.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{2.9}$$

Duyarlılık (Recall)

İlgili tüm numunelerin sayısına (pozitif olarak tanımlanması gereken tüm numunelerin) bölünmesiyle elde edilen doğru pozitif sonuçların sayısıdır.

$$Precision = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{2.10}$$

F1-Skoru (F1-score)

F1 skoru, bir F1 skorunun en iyi değerine 1 ve en kötü skoru 0'da ulaştığı kesinlik ve hatırlamanın ağırlıklı ortalaması olarak yorumlanabilir. Kesinlik ve hatırlamanın F1 skoruna göreceli katkısı eşittir. F1 skorunun formülü (2.11) 'de:

$$F1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \tag{2.11}$$

Makine öğrenimi uzmanları tarafından yaygın olarak kullanılan bir diğer önlem ROC Eğrisi Altındaki Alan'dır. ROC, Alıcı Çalışma Karakteristiği (ROC) anlamına gelir.

Gerçek pozitif TP ile yanlış pozitif FP Oranları arasındaki dengeyi ölçer [54]. ROC Eğrisinin Altındaki Alan (AUC), 1'den daha büyük olan eğri altındaki toplam alandır. AUC ne kadar yüksek olursa, sınıflandırıcı daha doğru pozitifleri tahmin etme olasılığı o kadar yüksektir [54].

ROC Eğrisi / AUC Skoru, bir modeli kendisine değerlendirirken en faydalıdır yöntemdir. Temel gerçeklerimiz ve öngörülerimiz 1'ler ve 0'lar. Ancak, tahminimiz hiçbir zaman 1 veya 0 değildir. Bunun yerine, bir olasılık tahmin edilmektedir ve sonra 1 veya 0 olup olmadığı değerlendirilmiştir. Tipik olarak, sınıflandırma eşiği .5 dir (ortada). Ancak daha iyi performans gösteren sınıflandırma eşiğine sahip olunmaktadır.

ROC Eğrisi'nin ölçütleri TPR (Gerçek Pozitif Oran) ve FPR (Yanlış Pozitif Oran) dır.

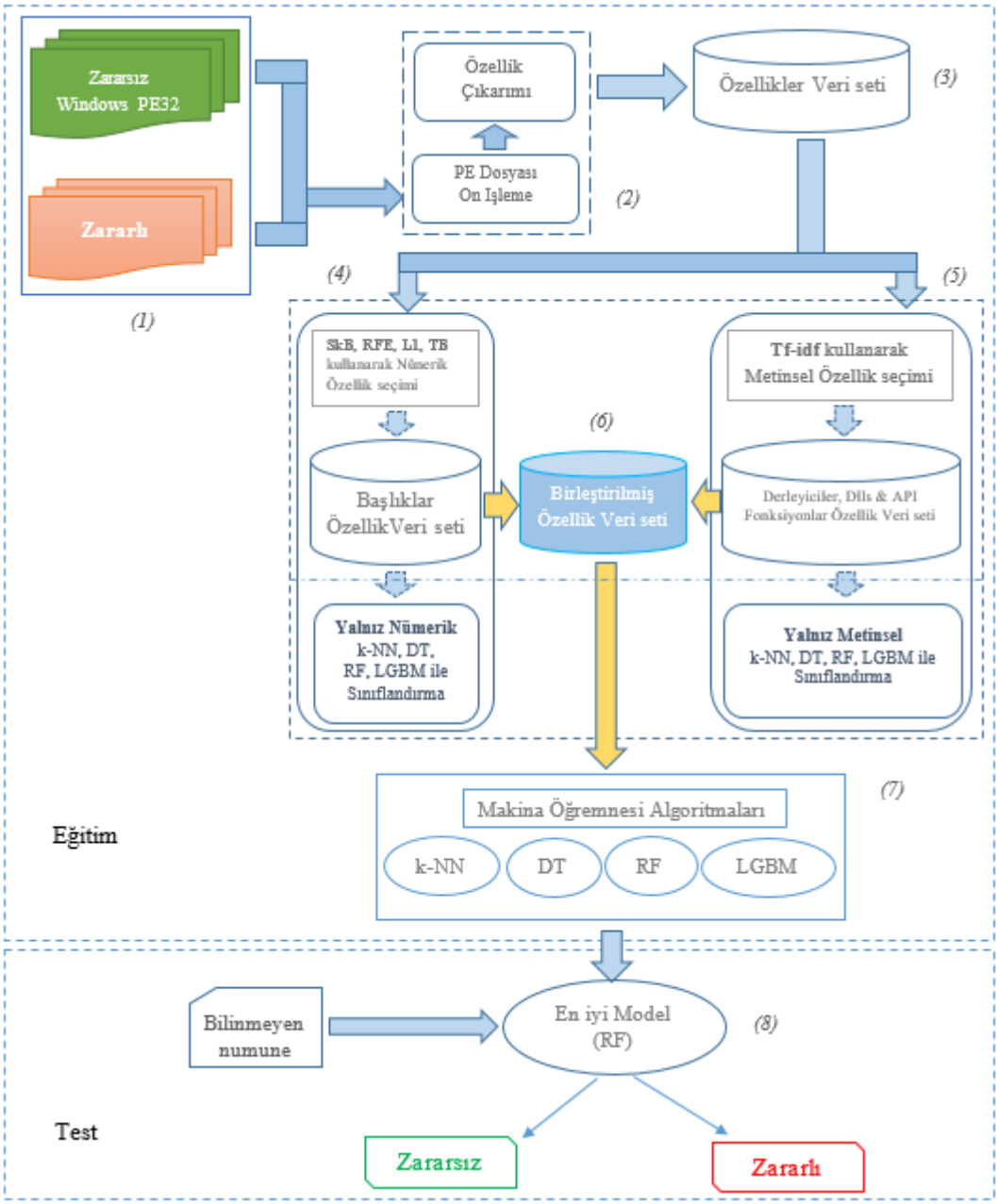
- Gerçek Pozitif Oranı: TP / (TP + FN)
- Yanlış Pozitif Oranı: FP / (FP + TN)

BÖLÜM 3. ÇALIŞTIRILABİLİR DOSYALARIN NÜMERİK VE METİNSEL ÖZELLİKLERİ KULLANILARAK MAKİNE ÖĞRENMESİ İLE ZARARLI YAZILIM TESPİTİ

Bu çalışmada çalıştırılabilir dosyaların zararlı bir yazılım özelliği barındırıp barındırmadığını tespit etmek amacıyla üç farklı özellik kümesi kullanan makine öğrenmesi tabanlı tespit sistemi geliştirilmiştir (Şekil 3.1.): Birinci özellik kümesinde sadece PE başlıklarına (Nümerik) ait özellikler kullanılmıştır. İkincisinde derleyici bilgilerine, kütüphanelere ve API fonksiyonlarına (Metinsel) ait özellikler kullanılmıştır. Son olarak da nümerik ve metinsel özellikler birleştirilerek bir test yapılmıştır. Özellik seçimi için PE dosyası başlıklarını kullanan birçok çalışma tek bir teknik kullanmaktadır. Bu çalışmada önerilen yöntemde PE dosyası başlıklarından özellikleri seçmek için üç farklı özellik seçim yönteminden (Filter, wrapped ve Embedded methods) Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) ve Tree-based feature selection (ExtraTree) teknikleri kullanılmıştır. Özellikler bu dört teknik ile seçildiğinde veri setindeki PE başlıklarına ait ilgisiz (önemsiz) özellikler kaldırılarak özellik kümesinin boyutu azaltılmıştır. Her bir tekniğin özellik seçip seçmediğine bakılarak oylama yapılmış ve sonrasında en yüksek oy puanına sahip özellikler sınıflandırıcıları eğitmek için kullanılmıştır. Derleyici bilgileri, kütüphane dosyaları ve API fonksiyonlarından özellik seçimi için Terim Frekansı-Ters Metin Frekansı tekniği (Term Frequency - Inverse Document Frequency, TF-IDF) kullanılmıştır. Bu çalışmada zararlı yazılımları tespit etmek için kNN, Decision Tree, Random Forest ve Light GBM sınıfladırıcılar kullanılmıştır.

Bu bölümde Şekil 3.1.'de verilen tespit mimarisi içerisindeki veri setini oluşturacak PE dosyalarının toplanması, toplanan PE dosyalarının ön işlenmesi ve veri setinin

oluşturulması, veri setinin analiz edilmesi ve özellik seçimi gibi konular ele alınacaktır.



Şekil 3.1. Zararlı yazılım tespit sisteminin mimarisi

3.1. Veri Seti Oluşturma

Bu çalışmada kullanılan veri setini oluşturmak için Şekil 3.1.'in 1.aşamasında belirtildiği gibi zararsız ve zararlı yazılım olmak üzere iki farklı yazılım sınıfı kullanılmıştır.

Zararsız Yazılımların Toplanması

Zararsız yazılım numuneleri .exe ve .dll uzantılı dosyaların yeni yüklenen Windows XP, Windows 7, Windows 8 ve Windows 10 (32 bit) işletim sistemlerinden ayıklanmasından elde edilmiştir. Daha sonra çok kullanılan uygulamalara ait 1,037 adet .exe uzantılı dosya ve web siteleri tarayıp hedef dosyaları bulan uygulama (Crawler) ile Sourceforge [16] ve CNET Download [17]'den farklı dosyalar indirilmiştir. Toplamda 11,122 zararsız yazılım numunesi elde edilmiştir.

Zararlı Yazılımların Toplanması

Zararlı çalıştırılabilir numune dosyaları VirusShare web sitesinden [18] ve daha eski zararlı yazılımları barındıran VX Heaven web sitesinden [19] indirilmiştir. Bu koleksiyon veri setini çeşitlendirmek ve hem eski hem de yeni zararlı yazılım türlerini örneklemek için kullanılmıştır. Kullanılan toplam zararlı yazılım numune sayısı 39,810'dur (Tablo 3.1. ve 3.2.'de ayrıntıları bulabilirsiniz).

Tablo 3.1. Veri seti kaynakları

Koleksiyon	**Kullanılan Numuneler**	**Veri seti Yüzdesi**
VirusShare_00321	15,728	30.88
VirusShare_00322	17,102	33.58
VX Heaven Collection	6,980	13.70
Toplam Zararlı yazılım	**39,810**	78.16
Zararsız Windows PE dosyaları	10,085	19.80
Diğer zararsız	1,037	2.04
Toplam Zararsız yazılım	**11,122**	21.84

Tablo 3.2. PE formatındaki zararlı yazılımların soyadlarına göre dağılımı.

Zararlı yazılım Tipi	Adware	Backdoor	Exploit	Trojan	Virus	Worm	Toplam
Miktar	7,849	8,138	1,209	7,240	7,672	7,702	39,810

Yukarıda belirtilen ilgili çalışmalarda (Tablo 1.1.), çoğu yazarın, önerilen algılama veya sınıflandırma yöntemini test etme konusunda 11,000 veya daha az sayıda zararlı numune veri setine bağlı kaldığı görülmüştür. Bu tez çalışmasında 39,810 zararlı yazılım numunesi kullanılmıştır.

3.2. Ön İşleme ve Özellik Çıkarımı

PE (Çalıştırılabilir) dosyaları yeni kurulan Windows XP, Windows 7, Windows 8 ve Windows 10'dan alındıktan sonra bu dosyaların SHA1 özetleri (SHA1 hash) hesaplanmıştır. Tekrarlanan PE dosyalarını bulmak için “ssdeep” aracı kullanılmıştır. Bu araç, hashleri karşılaştırmak suretiyle dosyaların benzerlik oranlarını üretir. Belli benzerlik oranına sahip dosyalar tekrarlanan dosya olarak kabul edilip veri setinden kaldırılmıştır. VirusShare tarafından sağlanan dosyalar PE dosyaları olmayanlarla karıştırıldığı için, PE dosyaları olmayan zararlı dosyalar veri setinden kaldırılmıştır. Bu işlemler, Şekil 3.1.'deki 2. aşamasınnda yapılmıştır.

Daha sonra, özellik çıkarımı için, tüm PE formatındaki (.exe ve .dll) dosyaların başlık bilgilerini, derleyici bilgilerini, DLL'leri ve her bir DLL içerisinde API fonksiyonlarını ayıklayan, etiketleyen ve bir CSV dosyasına depolayan bir PE dosya ayrıştırıcı komut dosyası (Şekil 3.2.) geliştirilmiştir.

PE dosyalarında birçok özellik vardır [11], ancak bu özelliklerin çoğu zararlı yazılım ve zararsız yazılımları ayırt etmede yardımcı değildir. Yonts ve ark. [14] ve yaptığımız PE dosyalarının özelliklerinin derinlemesine analizinde, daha detaylı bilgi için Tablo 3.3. 'de gösterildiği gibi 58,724 özellik çıkarılmıştır.

```
def getSizeOfOptionalHeader(self):
    return self.pe.FILE_HEADER.SizeOfOptionalHeader
def getCharacteristics(self):
    return self.pe.FILE_HEADER.Characteristics

# OPTIONAL HEADER
def getOPTIONAL_HEADER(self):
    return self.pe.OPTIONAL_HEADER
def getMagic(self):
    return self.pe.OPTIONAL_HEADER.Magic
def getMajorLinkerVersion(self):
    return self.pe.OPTIONAL_HEADER.MajorLinkerVersion
def getMinorLinkerVersion(self):
    return self.pe.OPTIONAL_HEADER.MinorLinkerVersion
def getSizeOfCode(self):
    return self.pe.OPTIONAL_HEADER.SizeOfCode
def getSizeOfInitializedData(self):
    return self.pe.OPTIONAL_HEADER.SizeOfInitializedData
def getSizeOfUninitializedData(self):
    return self.pe.OPTIONAL_HEADER.SizeOfUninitializedData
def getAddressOfEntryPoint(self):
    return self.pe.OPTIONAL_HEADER.AddressOfEntryPoint
```

Şekil 3.2. PE dosya çıkarımı için betiğin bölümü

Özelliklerimiz aşağıda sunulan iki ana kategoriye ayrılmıştır:

- Nümerik özellikler: Aynı formatı korumak için normalleştirilmesi gereken tamsayı veya kayan noktalı sayıları temsil ederler. Nümerik özelliklerde 77 adet ham özellik ve 8 adet türetilmiş özellik olmak üzere toplamda 85 adet özellik çıkarılmıştır. Türetilmiş özelliklerin değerleri, [55]'deki bazı başlık alanları ve bölüm alanları kullanılarak hesaplanır.
- Metinsel özellikler: Daha önce işlenmesi ve kullanmadan önce normalleştirilmesi gereken bir dizi kelimeden oluşurlar. Sunulan çalışmada, üç kelime kümesi vardır: Derleyici bilgileri, İçe Aktarılan DLL (ImportedDlls) ve İçe Aktarılan Fonksiyonlar (ImportedFunctions).

Taşınabilir çalıştırılabilir dosya yapısından çıkarılan özelliklere (Dosya başlığı, isteğe bağlı başlık ve diğerleri) ek olarak, dosya adı, boyutu, MD5, SHA1 ve bulanık (fuzzy) özetleri gibi genel özellikler de çıkarıldı.

MD5 ve SHA1 özetleri ve dosyanın adı gibi özellikler her bir çalıştırılabilir dosya için benzersizdir ve sınıflar için değil yalnızca dosyalar için ayrımcı oldukları farz edilirse Makine Öğrenmesi algoritmaları için kullanışlı değildir. Bununla birlikte, bir ön işleme adımında kullanışlı olabilirler. Ayrıca, özetleri aracılığıyla zararsız veya zararlı yazılımların kara listelerinde veya beyaz listelerinde kontrol etmek ve bilinen güvenli

sistem işlemi veya uygulamaların adını kullanarak kurbanları aldatmaya çalışıp çalışmadıklarını keşfetmek de mümkündür. Yukarıda verilen tüm bu bilgilere bağlı olarak çıkarılan özellikler Tablo 3.3.'te verilmiştir.

Tablo 3.3. PE Dosyasından Çıkarılan Özellikler.

Özellik Tanımı	**Özellik Sayısı**
Nümerik Özellikler	
DOS HEADER	2
FILE HEADER	9
OPTIONAL HEADER	29
Data Directory	30
SECTIONS	6
Türetilmiş Özellikler	
.text section — EntropySectionText	1
.data section — EntropySectionData	1
CountNonSuspiciousSections	1
CountSuspiciousSections	1
EntropyFile	1
DLLs referred — NumberOfImportedDlls	1
APIs referred — NumberOfImportedFunctions	1
SizeOfCode & SizeOfInitializedData Ration — CodeDataRation	1
Metinsel Özellikler	
Dlls	1,846
API Fonksiyonları	56,575
Derleyici Bilgileri	218
	58,724

Metinsel özellikler ile ön işleme sırasında benzer metinlerin (belgeler) kümelenmesi amaçlanmıştır. Bu şekilde, aynı kütüphaneleri, fonksiyonları ve derleyicileri kullanan yazılımlar, özelliklerin alanında yakın olma eğilimindedir. Bunu mümkün kılmak için, her bir kelime grubu, her kelimenin bir boşlukla ayrıldığı bir belgeye (Bilgi çıkarma bağlamında) dönüştürülür. Bu nedenle, her bir metinsel özellik (Derleyiciler, ImportedDlls and ImportedFunctions) için özellikler dosyasında bir "belge" alanı yaratılır. Ardından, tüm metinler büyük / küçük harf olarak katlanır ve normalleştirilir. Metin içeriği küçük harf olarak tutulur ve özel karakterler, aksan, işaretler ve sayılar

kaldırılır. Bu nedenle, her belge hakkında sözlerine dayanarak istatistiksel önlemler alınmıştır. Bu çalışmada, bölümlerinin 2.5.'te gösterilen, kelimelerinin göreceli önemi ile bir dokümanı temsil eden vektör uzay modeli (Vector Space Model —VSM) kullanılmıştır [55]. Bu, zararsız ve zararlı yazılımları, programlama dili, derleyici, paketleyici ve içe aktarılan kütüphaneleri ve programın çağırdığı fonksiyonları ayırt etmeye çalışmak için yapılmıştır.

3.3. Veri Analizi

Bu bölümün içerisinde çıkarılan ya da tercih edilen özellikler üzerinde analiz yapılmış ve özellik seçimi için gerekli ön bilgilerin elde edilmesi için çalışmalar yapılmıştır.

3.3.1. PE başlıkları özellik analizi

Bu bölümde PE özelliklerinin ortalama değerleri hesaplanmıştır. Tablo 3.5.'te verilen "Characteristics" özellik değerinin ortalaması zararsız (Benign) ve zararlı yazılım (Malware) arasındakin farkın yüksek bir değer olduğu görülmektedir. Örnek olarak, PE dosya başlığındaki "Characteristics" değer, "Characteristics" alan değerlerinin ortlama değerini tanımlar. Zararsız yürütülebilir dosyaların ortalama değeri iken 6,947.779, zararlı için bu değer 1,663.935'dir. Aynı durum diğer özellikler için de "ImageBase", "RelocationSize", "SectionsMeanRawsize" ve "SizeOfStackReserve" gözlemlenmiştir.

Tablo 3.4. Bazı PE özellikleri ortalama değerleri

Özellikler	Ortalama değerler	
	Zararsız	Zararlı
Characteristics	6,947.779	1,663.935
CheckSum	382,527.100	142,360,800.000
DebugSize	41.942	12.316
DllCharacteristics	11,609.110	16,033.630
ExportDirectorySymbols	179,583.400	407,243.000
ImageBase	550,268,700.000	53,726,480.000
MajorImageVersion	243.329	20.846
MinorImageVersion	228.093	18.825
RelocationSize	11,333.580	111,867.500
SectionsMeanRawsize	69,846.040	386,347.700
SizeOfStackReserve	308,887.100	1,327,880.000

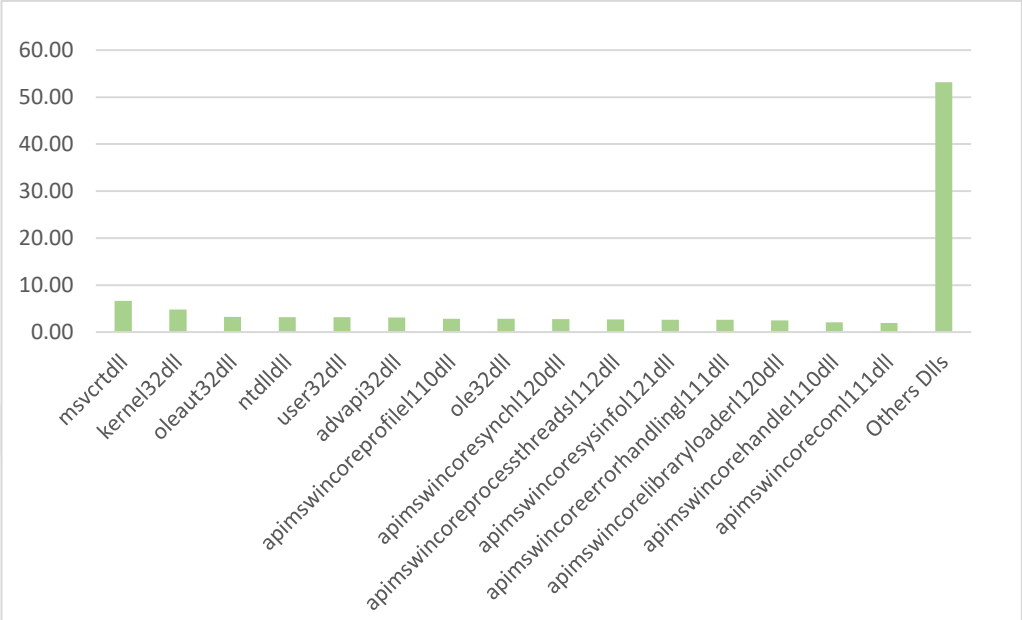
3.3.2. Dinamik bağlantı kütüphaneleri (DLL) dağılımları

Bir programın kullandığı DLL'ler, fonksiyonları hakkında birçok bilgi taşıdığı için her DLL'i bir özellik olarak kabul etmekte fayda görünmektedir.

3.3.2.1. DLL zararsız yazılım dağılımları

İlk önce tüm veri seti analiz edildikten sonra analiz için sadece 2000 numune alınmıştır. Bu numuneler içerisinde en çok kullanılan kütüphanelerin zararsız yazılım (benign) veri seti için ne olduğu kontrol edilmiştir. Toplanan numuneler tarafında toplamda 628 kütüphane kullanılmıştır. Şekil 3.3.'te görüldüğü üzere sadece 15 kütüphanenin toplamda kullanılma yüzdesi %46.80'dir. Şekil 3.3.'de gösterildiği gibi, numunelerin % 6.62'sinde yaygın olan msvcrt.dll, en çok kullanılan kütüphanedir, ardından diğer kütüphaneler ise şu şekildedir: kernel32.dll (%4.79), oleaut32.dll (%3.23), ntdll.dll (%3.17)), user32.dll (%3.14), advapi32.dll (%3.12), apimswincoreprofilel110.dll (2.84%), ole32.dll (%2.82), apimswincoresynchl120.dll (%2.77), apimswincoreprocessthreadsl112.dll (%2.66), apimswincoresysfol1121 .dll (%2.61), apimswincoreerrorhandlingl111.dll (%2.60), apimswincorelibraryloaderl120.dll (%2.45), apimswincorehandlel110.dll (%2.06) ve apimswincorecoml111.dll (%1.92). Kalan 613 dinamik kütüphanenin toplam kullanım

oranı %53.20' olmakta ve bu kütüphanelerin kullanım oranları %1,92'nin altında kalmaktadır. Bu oranlar, 628'den DLL'den 15 adedinin daha fazla kullanıldığını belli etmektedir.

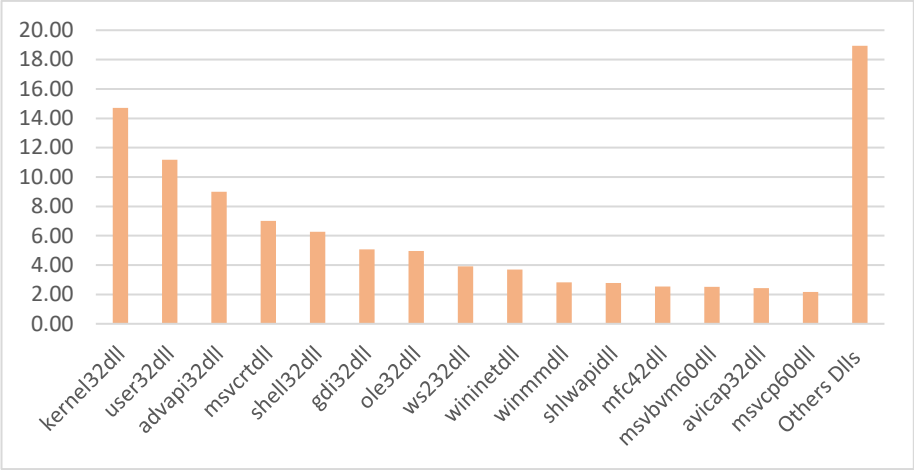


Şekil 3.3. Zararsız Dlls dağılımları

3.3.2.2. DLL zararlı yazılım dağılımları

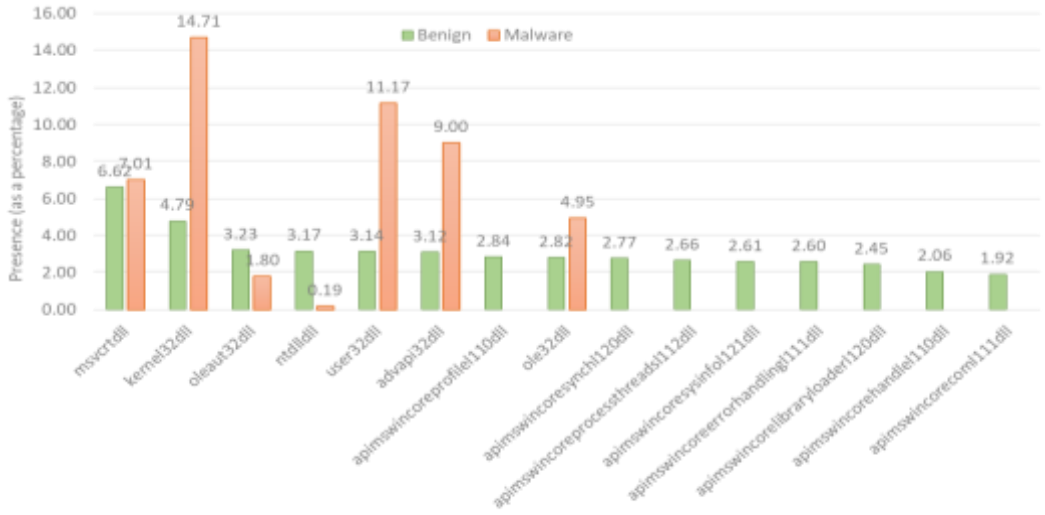
Öncelikle tüm veri seti incelendikten sonra analiz veri setini oluşturmak için sadece 2000 numune alınmıştır. Analiz veri seti için en çok kullanılan kütüphanelerin ne olduğu kontrol edilmiştir. Toplanan numuneler tarafından toplamda 87 kütüphane kullanılmıştır. Tüm kütüphaneler arasında 15 kütüphanenin kullanımı toplam kütüphane kullanımının %81.07'sini oluşturmaktadır.

Şekil 3.4.'te gösterildiği gibi, kernel32.dll %14.71 oranı ile en çok kullanılan kütüphane fark edilmiştir. user32.dll (%13.78), advapi32.dll (%9.00), msvcrt.dll (%7.01)), shell32.dll (%6.27), gdi32.dll (%5.07), ole32.dll (%4.95), ws232.dll (%3.92), wininet.dll (%3.69), winmm.dll (% 2.82), shlwapi.dll (%2.78), mfc42.dll (%2.55), msvbvm60.dll (%2.51), avicap32.dll (%2.42), msvcp60.dll (%2.17) ve diğer kütüphaneler (%18.93) geriye kalan %85.29 oranını paylaşmaktadır. Zararsız yazılımlarda olduğu gibi bu DLL'lerin kullanım dağılımı (oranları), yukarıda belitilen DLL'lerin daha fazla kullanıldığını göstermektedir.

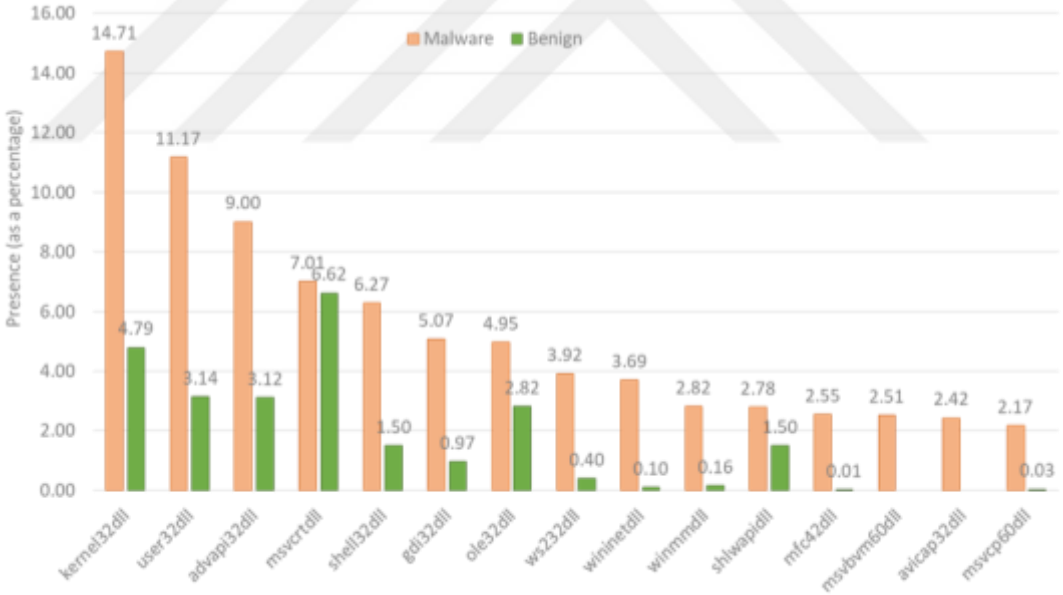


Şekil 3.4. Zararlı Dll dağılımıları

Her iki sınıf da yan yana incelenerek, kendileri tarafından en çok kullanılan kütüphaneler göz önünde bulundurularak, farklılıklarını vurgulamak mümkündür. Ayrıca, zararlı ve zararsız yazılım tarafından kullanılan kütüphanelerin en yaygın olanları arasında ortak kütüphaneler de vardır. Bunlar: kernel32.dll, user32.dll, advapi32.dll, oleaut32.dll ve ole32.dll olarak belirlenmiştir. Bu kütüphaneler, programın amacından bağımsız olarak her program tarafından yaygın olarak kullanılan standart Windows kütüphaneleri olarak tanımlanabilirken, aynı zamanda bir program tarafından kullanılan dinamik kütüphanelerin zararlı yazılımları zararsız yazılımlarından ayırmak için kullanılabileceğini de göstermektedir. Şekil 3.5. ve 3.6., zararsız ve zararlı yazılım dağıtımlarından en çok kullanılan 15 kütüphanenin kullanımı ile alakalı karşılaştırmalarını vermektedir.



Şekil 3.5. Kütüphanelerin (Dll) zararsız ve zararlı yazılımlarda kullanım karşılaştırılması–referans zararsız yazılım



Şekil 3.6. Kütüphanelerin (Dll) zararsız ve zararlı yazılımlarda kullanım karşılaştırılması–referans zararlı yazılım

3.3.3. API fonksiyonları dağılımları

Windows işletim sisteminin çekirdeği, Uygulama Programlama Arabirimi'nden (API) oluşur. Windows API, programların Windows'un gücünden yararlanmasını sağlar ve bu nedenle zararlı yazılım yazarları, API çağrılarını zararlı eylemler gerçekleştirmek için araç olarak kullanır. Windows API fonksiyon çağrısı, sistem hizmetleri, kullanıcı arayüzleri, ağ kaynakları, pencereler ve kütüphaneler vb. gibi ayrıntıları içerir. API çağrıları bir programın fonksiyonel seviyelerini yansıttığından, API çağrılarının analizi dosyanın davranışının anlaşılmasına yol açacaktır.

Zararlı kodlar, görevlerini yerine getirmek için Win32 ortamında sağlanan API fonksiyonlarını kullanarak davranışlarını gizleyebilir. Bu nedenle, ikili statik analizde amaç, zararlı yazılım kümesi için ortak olan bir dizi API çağrısını ve benzer şekilde, çalıştırılabilir zararsız dosyalarda ortak olan bir dizi API çağrısını tanımlamaktır.

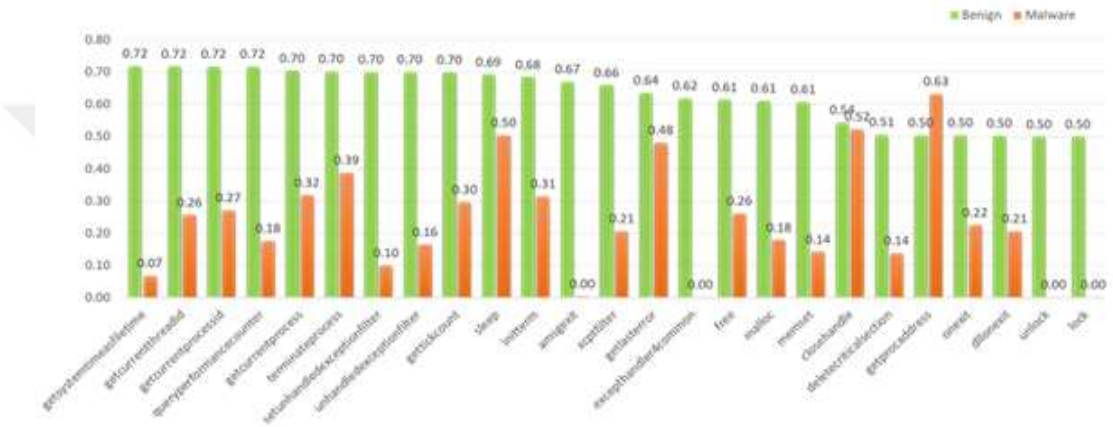
3.3.3.1. Zararsız ve zararlı yazılımlarda API fonksiyonlarının karşılaştırılması

Tüm zararsız yazılım veri seti analiz edildikten sonra 2000 numuneden oluşan bir analiz veri seti hazırlanmış ve o kümedeki yazılımlardan 10,319 farklı API fonksiyonu çıkarılmıştır. Analiz veri setindeki uygulamaların bu fonksiyonları çağırma sıklığı incelendiğinde fonksiyonların çağrılma sıklığı eşit olmadığı ortaya çıkmıştır. 10,319 fonksiyonun çağrılma sıklığı 275,396 olarak hesaplanmıştır.

Elde edilen genel zararlı yazılım veri setinden hazırlanan 2000 numunelik bir analiz veri setinden farklı sıklıklarda 3,602 farklı API fonksiyonunun kullanıldığı gözlemlenmiştir. Bu farklı fonksiyonların kullanılma sıklıkları 214,732 olarak hesaplanmıştır.

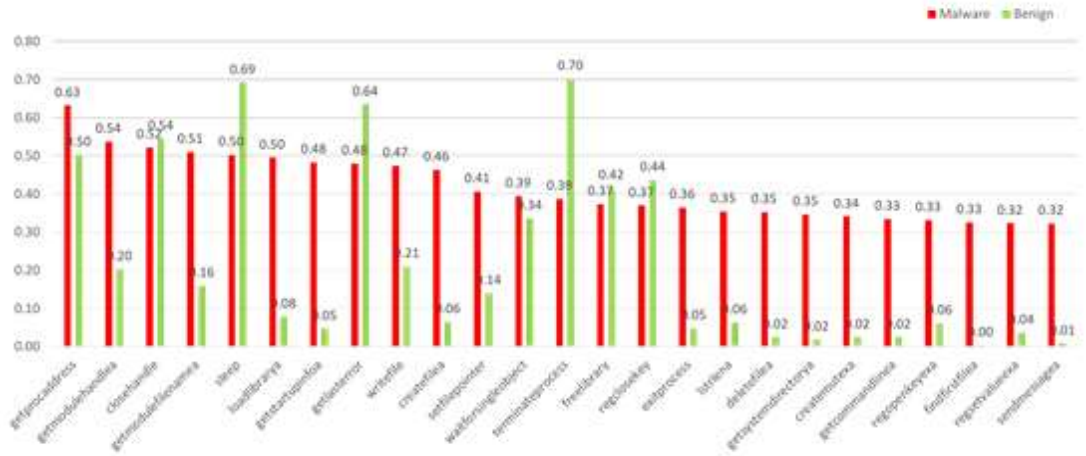
Şekil 3.7., Zararsız (Benign) yazılımları referans alarak zararlı ve zararsız yazılım numunelerinde en çok kullanılan fonksiyonları karşılaştırmaktadır. Fonksiyon, aynı yüzdelere sahip her iki yazılım sınıfında da kullanılır. Bu iki kategorinin karşılaştırılması aynı zamanda zararlı ve zararsız yazılımlar tarafından en çok

kullanılan fonksiyonların farklı olduğunu göstermektedir. Örneğin, fonksiyon, getsystemtimeasfiletime yalnızca zararlı yazılımların %0.07'sinde, zararsız yazılımların %0.72'sinde yaygındır. Ayrıca, handhandler4common dışındaki fonksiyonlar zararlı yazılımda sadece %0.00, zararsız yazılımda %0.67 prevalans göstermiştir. Getprocaddress fonksiyonu selimenin % 0.50'inde, zararlı yazılımlarda % 0.63'ünde yaygındır. Bu durum, bir program tarafından kullanılan fonksiyon türlerinin zararlı yazılımları, zararsız yazılımdan ayırmak için kullanılabileceğinin kanıtıdır.



Şekil 3.7. Zararsız vs zararlı yazılım fonksiyon dağılımlar – zararsız referans alnıarak (en iyi 25)

Şekil 3.8., zararlı yazılımları referans alarak, zararlı ve zararsız yazılım numunelerinde en çok kullanılan fonksiyonları karşılaştırmaktadır. Getprocaddress fonksiyonu Malware sınıfı tarafından en çok kullanılanıdır. Bu iki kategorinin karşılaştırılması aynı zamanda zararlı ve zararsız yazılımlar tarafından en çok kullanılan fonksiyonların farklı olduğunu göstermektedir. Örneğin, exitprocess fonksiyonu, zararsız sınıfın sadece %0.05'inde, zararlı yazılım sınıfının %0.36'sında yaygındır. Ayrıca, sendmessagea fonksiyonu zararsız sınıfında sadece %0.01, zararlı yazılım sınıfında %0.32 oranında görülmüştür. Fonksiyon terminateprocess, zararsız sınıfın %0.70'inde yaygın, ancak zararlı yazılımlarda yalnızca %0.39'dur. Bu durum, bir program tarafından kullanılan fonksiyon türlerinin zararlı yazılımları zararsız yazılımlardan ayırmak için kullanılabileceğinin bir başka kanıtıdır.

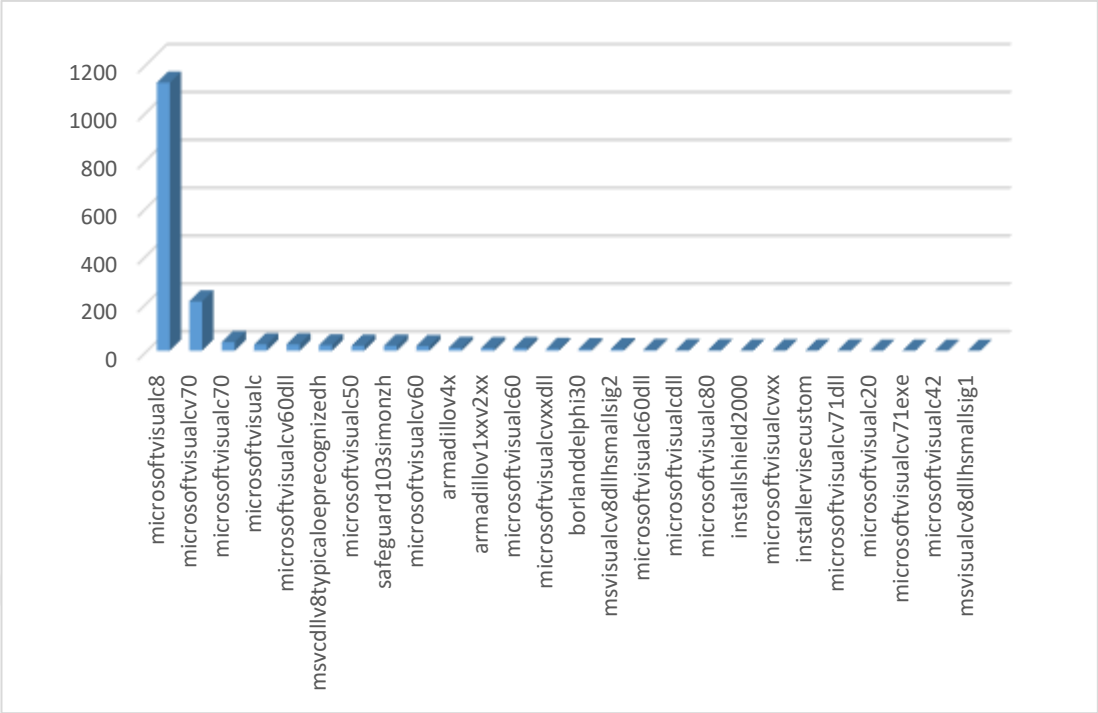


Şekil 3.8. Zararsız vs zararlı yazılım fonksiyon dağılımlar – zararlı referans alnıarak (en iyi 25)

3.3.4. Derleyici bilgileri dağılımları

3.3.4.1. Zararsız derleyici bilgileri dağılımları

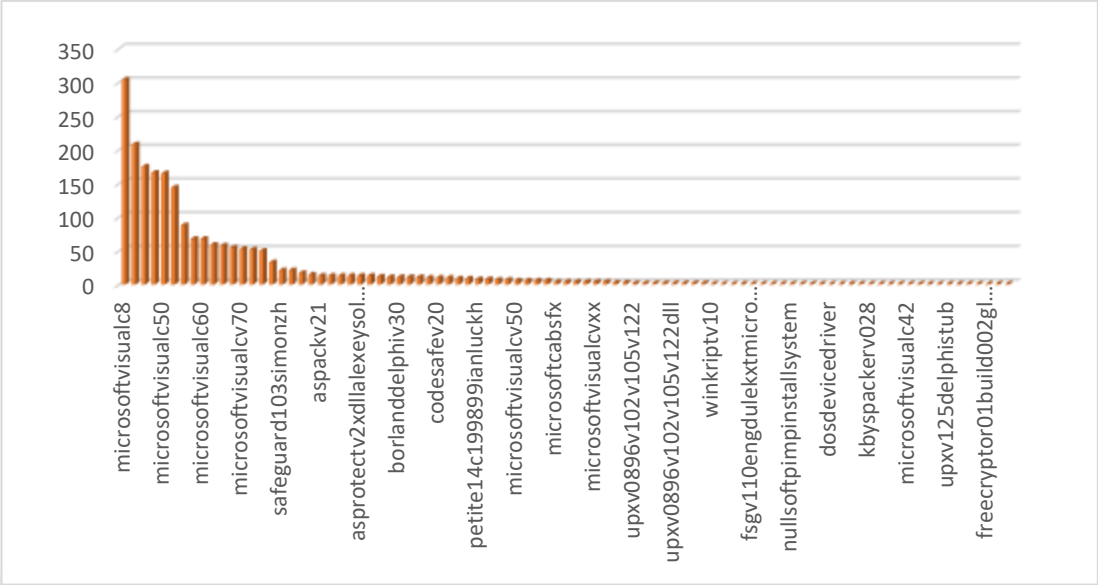
DLL ve API fonksiyonlarında yapıldığı gibi genel zararsız yazılım veri seti incelenip analiz için 2000 numunelik bir zararsız yazılım analiz veri seti oluşturulmuştur. Toplanan numunelerden derleyici bilgileri çıkarılmıştır. Bu çıkarımlar, şekil 3.9.'te görüldüğü üzere örnek olarak microsoftvisualc8 (1,121 yazılım), microsoftvisualcv70 (206 yazılım), microsoftvisualc70 (36 yazılım), microsoftvisualc (27 yazılım), microsoftvisualcv60dll (27 yazılım), msvcdllv8typicaloeprecognizedh (23 yazılım) ve microsoftvisualc50 (21 yazılım) 26 derleyicinin kullanıldığını göstermektedir. Dört derleyicinin (microsoftvisualc20, microsoftvisualcv71exe, microsoftvisualc42 ve msvisualcv8dllhsmallsig1) daha az kullanıldığı görülmektedir.



Şekil 3.9. Zararsız derleyici bilgileri dağılımları

3.3.4.2. Zararlı derleyici bilgileri dağılımları

Genel zararlı yazılım veri seti incelenip analiz için 2000 numunelik bir zararsız yazılım analiz veri seti oluşturulmuştur. Numuneler 90 farklı derleyici tarafından derlendiği fark edilmiştir. Şekil 3.10. derleyicilerin oluşturduğu yazılım istatistik bilgilerini göstermektedir. Şekilden microsoftvisualc8 (304 yazılım) derleyicisinin en fazla kullanıldığı görülmektedir. microsoftvisualbasicv50v60 208, microsoftvisualbasicv50 175, microsoftvisualc 166, microsoftvisualc50 165, microsoftvisualcv60 144, microsoftvisualcv60dll 88, armadillov1xxv2xx 68, microsoftvisualc60 68, nullsoftpimpstubsfx 59, microsoftvisualc70 36, microsoftvisualc 27, microsoftvisualcv60dll 27, msvcdllv8typicaloeprecognizedh 23, safeguard103simonzh 21 ve microsoftvisualc50 21 yazılım derlemek için kullanılmıştır. pecryptv100v101, pebundlev02v20x, winkriptv10mrcrimson, freecryptor01build002gloff ve freecryptor01build002gloff gibi diğer derleyiciler sadece bir (1) yazılım derlemek için kullanılmıştır.



Şekil 3.10. Zararlı derleyici bilgileri dağılımları

3.4. Özellik Seçimi

Özellik Seçimi, makine öğrenmesinde modelin performansını büyük ölçüde etkileyen temel kavramlardan biridir. Alakasız veya kısmen ilgili özellikler, özellik seçiminin neden yapılması gerektiğini belirten sınıflandırıcı performansını olumsuz etkileyebilir [56]. Başka bir deyişle, özellik seçiminin özellik seçimi algoritmaları kullanarak otomatik olarak belirlenen veya ilgilendiğimiz tahmin değişkenine veya çıktısına en çok katkıda bulunacak özellikleri manuel olarak seçtiği süreç olduğunu söyleyebiliriz.

Özellik seçim yöntemlerinin üç genel sınıfı vardır: filtreleme (filtering) yöntemleri, sarıcı (wrapper) yöntemler ve gömülü (embedded) yöntemler [57] ve [58].

a) Filtre yöntemleri (Filter methods): Filtre yöntemleri, istatistiki olarak özellikleri puanlar. Daha yüksek puan alan özellikler veri setinde tutulurken, düşük puan alan özellikler kaldırılır. Örneğin: bilgi kazancı, ki-kare testi, balıkçı skoru, korelasyon katsayısı ve varyans eşiği.

   Çalışmada, en yüksek puanlama özelliklerinden bazı özellikleri kaldırmak için SelectKBest yöntemi uygulanmıştır. Chi2, SelectKBest'in "*score_func*"'u olarak

seçilmiştir. Bu puan, ki kare testi için en yüksek değerlere sahip olan n_özelliği özelliklerini seçmek için kullanılabilir [59].

b) Sarıcı yöntemleri (Wrapper methods): Burada, farklı özellik kombinasyonları bir tahmin modeli ile denenmekte ve en yüksek doğruluğa yol açan kombinasyon seçilmektedir. Örneğin özyinelemeli özelliğin ortadan kaldırılması, sıralı özellik seçim algoritmaları ve genetik algoritmalar.

   Bu çalışmada Wrapper yöntemlerinden özyinelemeli özelliklerin kaldırılması (RFE) kullanılmıştır. RFE, en yüksek algılama doğruluğunu sağlayan özellik alt kümesini seçtiği için seçilmiştir [60], [61]. Ayrıca, özellik ağırlıkları sağlayan algılama algoritmalarıyla çalışmaktadır. RFE, özellikleri tekrar tekrar ortadan kaldırır, dedektörü kalan özellik alt kümesiyle yeniden eğitir. Ortadan kaldırılan özellikler, egzersiz sırasında hesaplanan en düşük ağırlıklara sahip olanlardır.

c) Gömülü yöntemler (embedded methods): Bu yöntemler, model oluşturulurken kullanılan özellikleri değerlendirir. Burada L1 (LASSO) normalizasyonu, karar ağacı ve Ağaç tabanlı özellik seçimi gibi yöntemler bulunmaktadır. Bu çalışmada gömülü yöntemlerden L1 ve ExtraTree yöntemleri kullanılmıştır.

   LASSO (Least Absolute Shrinkage and Selection Operator: En Az Mutlak Çekme ve Seçim Operatörü) yöntemi, model parametrelerinin mutlak değerlerinin toplamına bir sınırlama getirir; toplamın sabit bir değerden (üst sınır) daha az olması gerekir. Bunu yapmak için yöntem, bazılarını sıfıra çeken regresyon değişkenlerinin katsayılarını cezalandırdığı bir daraltma (düzenlileştirme) işlemi uygular. Özellik seçim sürecinde, küçültme işleminden sonra hala sıfır olmayan bir katsayısı olan değişkenler modelin bir parçası olarak seçilir. Bu sürecin amacı, tahmin hatasını en aza indirmektir [62].

   ExtraTree (Extremely Randomized Trees: Son Derece Randomize Ağaçlar)'de [63], ağaç yapısını sayısal giriş özellikleri bağlamında daha fazla randomize etmenin ana amacı ile önerilmiştir; buradaki optimum kesme noktası seçimi,

varyansın büyük bir farkının büyük bir kısmından sorumludur. İndüklenmiş ağaç Rastgele ormanlarla ilgili olarak, yöntem öğrenme örneğinin önyükleme kopyalarını kullanma fikrini bırakır ve her düğümde rastgele seçilen özelliklerin her biri için en uygun kesme noktasını bulmaya çalışmak yerine, bir kesme noktası rastgele seçer.

Bu fikir daha çok veya daha az sürekli değişen, çok sayıda sayısal özellik ile karakterize edilen birçok problem bağlamında oldukça üretkendir: yumuşatma sayesinde sık sık artan doğruluk sağlanır ve aynı zamanda standart ağaçlarda ve rastgele ormanlarda kesme noktalarının optimal değerlerinin belirlenmesine bağlı hesaplama yüklerini önemli ölçüde azaltır.

Verileri yeni bir özellik alanına yansıttığımız için Temel Bileşen Analizi (PCA) gibi dönüştürme ve projeksiyon teknikleri, bir özellik çıkarma yaklaşımı olarak sayılabilmektedir. Bir veri setinin boyutsallığını açıkça azaltmasına rağmen, orijinal değişkenleri de sentetik olanlarla değiştirir. Bu nedenle, bu çalışmada PCA özellik seçimi algoritmaları sınıfına dahil edilmemiştir.

3.4.1. Nümerik özellikler

Nümerik özellikler, aynı format biçimini korumak için normalleştirilmesi gereken tam sayı veya kayan nokta sayılarını temsil eder. Bu işlem için 85 özellik çıkarılmıştır. Ancak bu özelliklerin tümü çalışmada önerilen modelin eğitilmesi için kullanılmamıştır, önemsiz ya da alakasız özellikleri kaldırmak için özellik seçimi yapılması gerekmektedir. Özelliklerin kaldırılmasında nümerik özellik seçimi için dört farklı özellik seçimi tekniği kullanılmıştır (Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) and Tree-based feature selection (ExtraTree). Bu teknikler sayesinde özellik kümesinin boyutu azaltılmıştır. Bunun için her teknik ilgili özelliği seçip seçmeme noktasında oy verir. Son olarak, oylar sayılır ve daha yüksek oy puanına sahip özellikler sınıflandırıcıları eğitmek için kullanılır. Bu çalışmada, Tablo 3.4.'te de görüldüğü üzere 2 ve üzeri oy alan özellikler önemli özellik olarak kabul edilmiş ve seçilmiştir.

Bu işlem, Şekil 3.1.'in 4. aşamasında gerçekleştirilmiştir. Bu algoritmalar nümerik veriler üzerinde iyi sonuç vermesi sebebi ile seçilmiştir (Bölüm 2.1.6.). Tablo 3.4., Bölüm 3.3.1.'de yapılan veri analiz çalışmalarını doğrular bir sonuç göstermektedir.

Tablo 3.5. Özellik Seçimi Sonrası Nümerik Özellikler

No	Özellikler	SkB	TB	RFE	L1	Final-Score
1	MajorOperatingSystemVersion	1	1	1	1	4
2	MajorImageVersion	0	1	1	1	3
3	ImageBase	1	1	0	1	3
4	Characteristics	1	1	0	1	3
5	ExportDirectorySymbols	1	1	0	1	3
6	DebugSize	0	1	0	1	2
7	CheckSum	1	0	0	1	2
8	EntropyFile	1	0	1	0	2
9	DllCharacteristics	0	1	0	1	2
10	SectionsMeanRawsize	1	0	1	0	2
11	SizeOfStackReserve	0	1	0	1	2
12	RelocationSize	0	1	0	1	2
13	MinorImageVersion	1	1	0	0	2

3.4.2. Metinsel özellikler

Bölüm 3.3.2., 3.3.3., ve 3.3.4.'te yapılan veri analiz sonuçları dikkate alınarak etkili metinsel özellikleri oluşturulmuştur. Bu alt bölümde sabit sayıda en çok kullanılan V kelimesini kullanmak yerine, her metinsel özelliğin sözlüğünü belge sıklığını kullanmanın yeni bir yolu araştırılmıştır.

Bölüm 2.5.'te açıklandığı gibi, bir sözcük alma probleminde, durma sözcüklerini, bir metinde anlamı olmayan sözcükleri kaldırmak gerekir. Bununla birlikte, bu çalışmada, sözlerin bir anlamı yoktur, yani, bağlamımız için önemli olup olmadıklarını ölçülmemektedir (yani bir durma kelimesi yoktur). Sistem çağrıları da düşünülürse, hemen hemen tüm programlar tarafından kullanılan, örneğin sadece kullanıcı için bir şeyler yazmak amacıyla kullanılan ve kötü niyetli davranışı olmayan sistem çağrıları da mevcuttur.

Bu nedenle, bu tür bilgiler zararlı yazılımları zararsız yazılımlardan ayırmak için kullanışlı değildir ve aynı zamanda hangi kelimeleri görmezden gelmek gerektiğine karar verilmesi gerektiğinden, engellenecek kelimeler (stopwords) olarak kabul edilen bir DLL listesi, API Fonksiyonları veya derleyicileri oluşturmak da mümkün değildir.

Bu sorunu çözmek amacıyla çıktıyı etkilemek için değiştirilebilecek birkaç dahili ayara sahip *TfidfVectorizer* kullanarak, belge frekansı aralığının dışında kalan terimleri yok saymak için bir alt ve üst limit kullanılmıştır. Her iki parametre de belirtilen eşik değerden kesinlikle daha düşük bir belge sıklığına sahip terimleri yok sayan "*min_df*" ve verilen eşik değerden kesinlikle daha yüksek bir belge frekansına sahip terimleri yok sayan "*max_df*"'dir [64]. Bununla, tanımlanan değerlere bağlı olarak en yaygın ve en nadir bulunan DLL kütüphaneleri, API fonksiyonları ve derleyicileri görmezden gelinebilir ve bu tekniğin bir makine öğrenme modelindeki etkisi incelenebilir. Çalışmanın bu aşaması Şekil 3.1.'in 5. aşamasında gerçekleştirilmiştir.

Tablo 3.6. Özellik Seçimi Sonrası Metinsel Özellikler

Özellik Tanımı	Özellik Sayısı
Dlls	100
API Fonksiyonları	100
Derleyici Bilgileri	100

3.4.3. Özelliklerin birleştirilmesi

Tüm metinsel özellikler, ön işlemden geçtikten sonra, VSM vektörlerinin her biri dosyanın diğer özellikleri ile birleştirilir, 13 + 3 × V ile yeni bir vektör elde edilir, burada 13 nümerik özellik sayısı ve V'nin boyutu metinsel özellikler üzerinde kullanılan kelimeler mevcuttur (Tablo 3.4. ve Tablo 3.5.). Birleştirme, Şekil 3.1.'in 6. aşamasında gerçekleştirilmiştir.

Yukarıda tanımlanan birleştirme işleminden sonra çıkarılan özelliklerin normalize edilmesi gerekmektedir. Şekil 3.6.'de verilen özelliklere normalizasyon tekniği olarak Bölüm 2.5.'te sunulan "Min-Max" denklemi kullanılmıştır. Seçimi yapılan özellikler Min-Max denkleminde (Denklem 2.4) "0" ile "1" arasında bir aralık değeri kullanılarak ölçeklendirilmiştir.

BÖLÜM 4. UYGULAMA VE PERFORMANS SONUÇLARI

Bu bölümde tez süresince önerilen yaklaşımlara ait çözümlerin performans sonuçları ile alakalı bilgiler verilecektir. Bilinmeyen ve yeni bir zararlı yazılımı tespit etmek için Random Forest, Decision Tree, k-Nearest Neighbor and Light GBM sınıflandırıcılarının performansını değerlendirmek için farklı özellik kümeleri ile ayrı ayrı üç önemli test yapılmıştır:

a) Sadece Nümerik özellikleri (PE başlıkları) kullanarak,
b) Sadece Metinsel özellikleri (Derleyicilerin bilgileri, DLL ve her DLL içindeki API fonksiyonları) kullanarak.
c) Her ikisinin bir kombinasyonunu kullanarak.

4.1. Kullanılan Araçlar

Tüm çalışmalar, Ubuntu 16.04'te özel bir sanal makine üzerinde yapılmıştır. Windows Host, 4 çekirdekli bir Intel (R) Çekirdek (TM) i7-5500U CPU @ 2.40GHz özelliklerine sahiptir. Disk alanı, SSD RAID deposuna ayrılmıştır. Ön işletim dosyaları Linux işletim sistemindeki yerel destek nedeniyle bash scriptleri kullanılarak yapılmıştır.

Python pefile Modülü

Python pefile modülü, [13], PE formatındaki dosyaları ayrıştırmak için kullanılan bir python modülüdür. Modül bir PE dosyasının tüm niteliklerini bir nesneye yükler. Özellik çıkarımı için python dilinde özel olarak üretilen aracımız, pefile modülünün özelliklerine dayanmaktadır.

Scikit-learn Library

Scikit-learn, [65], NumPy, SciPy ve matplotlib üzerine inşa edilmiş makine öğrenmesi için olan python çerçevesidir. Özellik seçme yöntemleri, sınıflandırma algoritmaları, doğrulama ve performans değerlendirme işlevselliğini kapsar.

4.2. Sınıflandırma Algoritmalarının Performans Sonuçları

4.2.1. Nümerik özellikler (sadece PE başlığı) kullanılarak elde edilen test sonuçları

Nümerik özellikler üzerinde yapılan işlemleri aşağıdaki gibi gerçekleştildi:

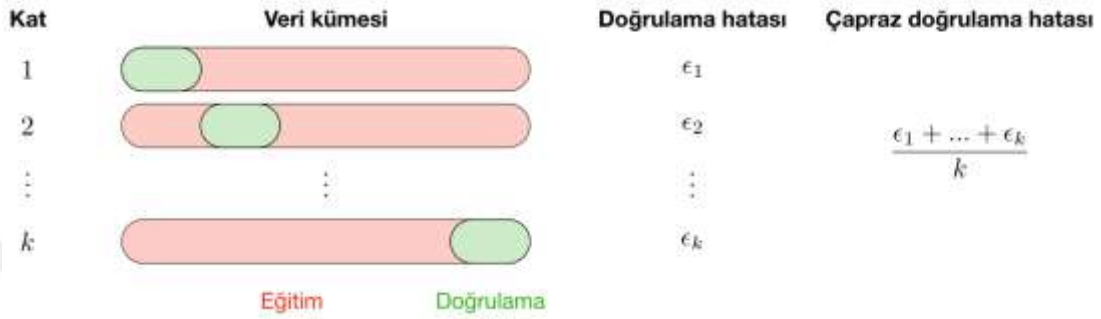
- PE dosyalarından PE başlık özellik çıkartımı yapılmıştır (Şekil 3.1.'deki 1.aşama). Bu çalışma sonucunda ilk olarak 85 adet özellik çıkartılmıştır.

- Özellik kümesinin boyutunu azaltmak için, üzerlerinde dört farklı özellik seçim tekniği kullanılmıştır. Filter, wrapped and Embedded seçim metotlarında Univariate feature selection (SelectKBest), Recursive Feature Elimination (RFE), L1-based feature selection (L1) ve Tree-based feature selection (ExtraTree) algortimaları kullanılmıştır. Burada her kullanılan tekniğin özelliği seçip seçmediğine bakılmıştır. Son olarak, oylar sayılmış ve 2 ve üzeri oy alan özellikler sınıflandırıcıları eğitmek için kullanılmıştır (Tablo 3.1.). Bu işlemin sonucunda 13 adet özelliğe düşürülmüştür.

Yukarıdaki çalışmalar sonucunda elde edilen özellikler, zararlı yazılım tespiti için kNN, Decision Tree, Light GBM and Random Forest sınıflandırma algoritmaları kullanılarak eğitilmiştir (Şekil 3.1.'in 7. ve 8. aşamaları). Eğitilen dört modelin tümü, 10 kat çapraz doğrulama yöntemi kullanılarak çeşitli sınıflandırma metrikleri üzerinde test edilmiştir.

Eğitimli modelin geçerliliği, kullanılan makine öğrenmesi sınıflandırıcılarını eğitmek ve test etmek için denetimli öğrenimde kullanılan bir işlem olan K-Katlama (K-Fold)

doğrulaması kullanılarak gerçekleştirilmiştir. Burada, tüm veri seti, k-1 katlarının bir eğitim seti olarak kullanıldığı ve geri kalan katın, eğitilmiş modeli doğrulamak için bir test seti olarak kullanıldığı, k daha küçük kümelere bölünmüştür. Bu validasyonun performansı, aynı işlemin ortalaması, k kez tekrarlandığından, her seferinde ayarlanan test olarak farklı bir katlama veya veri bölümü kullanıldığında alınmaktadır [43].



Şekil 4.1. Çapraz doğrulama [66]

Bu çalışmada Şekil 4.1.'de gösterildiği üzere 10 kat çapraz doğrulama kullanılmıştır ve veri seti, test için tutulan verilerin %10'unu ve sınıflandırma modelini eğitmek ve oluşturmak için kullanılan %90'ına ayrılarak, 10 eşit alt gruba veya katlamaya bölünmüştür. Bu işlem, 10 tekrardan seçilen en iyi sonuçla tekrarlanır.

Bu metodoloji, önceden belirlenmiş bir bilgi içermeyen zararlı yazılım içeren zararlı Win32 dosyalarını tespit etmek için verilen bir yaklaşımın sağlamlığının değerlendirilmesine yardımcı olur. Tablo 4.1. ve 4.2., her modelin performansını göstermektedir.

Tablo 4.1. Nümerik özelliklere göre yapılan sınıflandırmaya ait performans sonuçları

Model	**Doğruluk**	**Kesinlik**	**Duyarlılık**	**F1-skoru**
Random Forest	99.49	99.20	99.14	99.49
Light gbm	99.37	99.00	99.16	99.37
Decision Tree	99.21	98.76	98.95	99.22
K-Nearest Neighbors	99.03	98.40	98.77	99.03

Farklı modeller kullanılarak elde edilen sonuçları karşılaştırmak için Doğruluk, Kesinlik, Geri Çağırma ve F ölçümü performans ölçütleri kullanılmıştır. Diğer ana ölçümler ile bu performans ölçütleri, Bölüm 2.6.2.'deki formüllerle açıklanmaktadır.

KNN'in %99.03 doğruluk ve %0.78 yanlış hata oranı (FPR) ile diğer sınıflandırıcılara göre en düşük performansa sahip sınıflandırıcı olduğu gözlemlenmiştir. Decision Tree ile %99.22 doğruluk, %0.39 FPR sonuçları elde edilirken, Light gbm ile de %99.37 doğruluk %0.40 FPR'ne sahip sonuçlar elde edilmiştir. Random Forest ise %99.49 doğruluk ve %99.49 FPR oranları ile, kullanılan sınıflandırıcılar arasında en iyi sonuçları vermiştir.

Çeşitli sınıflandırıcıların, farklı değerlerde çapraz onaylı 10 katlama performansının sonucu da Tablo 4.1. ve 4.2.'de özetlenmiştir.

Tablo 4.2. Başlıklardaki nümerik özelliklere göre performans ölçümleri

Model	False Positive Rate	False Negative Rate
Random Forest	0.26	1.18
Lgbm	0.40	1.15
Decision Tree	0.39	1.19
K-Nearest Neighbors	0.78	1.77

Tablo 4.3, sadece PE başlığı nümerik özellikleri kullanılarak elde edilen sonucun önceki çalışmayla karşılaştırmasını göstermektedir. Tablo 4.3.'den, önerilen özellik setimize göre yapılan sınıflandırmanın önceki bütün çalışmalardan daha iyi bir sonuç ürettiği görülmektedir. Elde ettiğimiz sonuçlara en yakın sonuç, Kozachok [12] çalışmasında elde edilmiş ve bu çalışma ile karşılaştırma yapıldığında ise Decision Tree'de %0.03 bizim çalısmamızdan bir farkla üstünlük göstermiştir. Kozachok [12] Random Forest kullanmadığı için bir karşılaştırma yapılmamıştır. Ayrıca çalışmamız bilinmeyen zararlı yazılımları %0.26 yanlış pozitif oranı ile tespit etmektedir.

Tablo 4.3. Nümerik sonuçların önceki çalışmalarla karşılaştırılması

Çalışmalar		**Doğruluk**	**Yanlış pozitif Oranı**
Önerilen çalışma	Decision Tree	99.21	0.39
	Random Forest	99.49	0.26
Wang ve ark. [9]	SVM	98.86	-
Liao [10]	if-else-rulse	99.00	0.20
Kozachok ve ark. [12]	Decision Tree	99.24	-

4.2.2. Metinsel özellikler (derleyici bilgileriler, DLLs and API fonksiyonları) kullanılarak elde edilen test sonuçları

Bir önceki bölümde açıklandığı üzere derleyici bilgileri, kütüphaneler ve API fonksiyonları bir bütün halinde metinsel özellikler olarak ele alınmıştır.

Her iki eşik için (*max_df* ve *min_df*) ve metinsel özelliklerin tüm olası kombinasyonları (Dll kütüphaneleri, API fonksiyon çağrıları ve Derleyici bilgileri) önceki bölümde sunulan on kat çapraz doğrulama yöntemi kullanılarak test edilmiştir.

Tüm metinsel özellikler kullanılarak (ilk 100 DLL adet, ilk 100 API adet ve ilk 100 derleyici adet özellik max_df - min_df arasında alınmıştır), max_df %55 ve min_df %10 ayarları uygulanarak Random Forest ile %99.06 en iyi doğruluğu, %99.20 kesinlik, %99.60 duyarlılık ve %99.40 f1 puanı elde edilmiştir. Bu test ile elde edilen tüm sonuçlar (Tablo 4.4.), bu eşiklerin kullanılmasının sınıflandırıcı performansını arttırmaya yardımcı olduğunu kanıtlamaktadır.

Tablo 4.4. Belge sıklığı terimlerine göre Metinsel Özellikler kullanarak elde edilen sonuçlar

Özellik tipi	Sınıflandırcı	Doğruluk	Kesinlik	Duyarlılık	F1-skoru	FPR	FNR
Derleyiciler, DLLs & API fonksiyonları	kNN	96.66	99.63	96.08	97.83	1.11	3.92
	Decision Tree	98.76	99.22	99.20	99.21	2.84	0.80
	Light gbm	98.84	99.33	99.18	99.26	2.37	0.81
	Random Forest	**99.06**	99.12	99.60	99.40	**2.92**	0.40

Tablo 4.9.'da gösterildiği üzere literatürde daha önce yapılan akademik çalışmaların hiçbirisinde hem veri dizinleri hem de derleyici bilgileri (metinsel özellikler) birlikte dikkate alınarak bir sınıflandırma yapılmamıştır. Bu yüzden Tablo 4.5.'te bu alt bölümde sadece veri dizinleri ile alakalı yapılan çalışmaları ile bir karşılaştırma yapılmıştır.

Tablo 4.5. Metinsel sonuçların önceki çalışmalarla karşılaştırılması

Çalışmalar		**Doğruluk**	**Yanlış pozitif Oranı**
Önerilen çalışma	Light gbm	98.84	2.37
	Random Forest	99.06	2.92
Schultz ve ark. [5]	Naive Bayes	97.11	3.80
Kolter ve ark. [6]	if-else-rulse	99.00	0.20

Tablo 4.5.'den, önerilen özellik setimize göre yapılan Random Forest sınıflandırmanın önceki bütün çalışmalardan daha iyi bir sonuç ürettiği görülmektedir. Elde ettiğimiz sonuçlara en yakın sonuç, Kolter ve ark. [6] çalışmasında elde edilmiş ve bu çalışma ile karşılaştırma yapıldığında ise Decision Tree'de %0.06 pozitif bir farkla üstünlük göstermiştir. Ayrıca çalışmamız bilinmeyen zararlı yazılımları Random Forest sınıflandırıcı kullanıldığında %2.92 yanlış pozitif oranı ile tespit etmektedir.

4.2.3. Nümerik ve metinsel özellikler birlikte kullanılarak elde edilen test sonuçları

Bu bölümünün içerisinde hem Tablo 4.1.'de verilen nümerik özellikler hem de Tablo 4.4.'de verilen tüm metinsel özellikler birlikte kullanılarak test edilmiştir.

Bir önceki bölümde sunulan on kat çapraz doğrulama yöntemi kullanılarak tüm eşikler (*max_df* ve *min_df*) ile tüm olası metin kombinasyonları (Dll kütüphaneler, API fonksiyon çağrıları ve Derleyici bilgileri) test edilmiştir. Daha sonra metinsel

özellikleri nümerik özellikler ile birleştirmek için test edilen metinsel özellik kombinasyonları arasından en iyi sonucu veren kombinasyon seçilmiştir.

4.2.3.1. Nümerik ve veri dizinleri (DLL ve API fonksiyonları)

Tablo 4.6. Nümerik ve Veri Dizinleri özelliklere göre elde edilen sonuçlar

Özellik tipi	Sınıflandırcı	Doğruluk	Kesinlik	Duyarlılık	F1-skoru	FPR	FNR
PE Başlıkları, DLLs & API Fonksiyonları	kNN	94.49	95.63	96.08	95.85	8.70	3.92
	Decision Tree	96.97	97.67	97.76	97.71	4.60	2.24
	Light gbm	98.08	98.69	98.41	98.55	2.55	1.59
	Random Forest	**99.20**	99.53	99.25	99.39	**0.91**	0.75

Tablo 4.6, Nümerik ve Veri Dizinleri özelliklerinin birleştirilmesi sonucunda elde edilen sonuçların diğer yöntemlerle karşılaştırmalı sonuçlarını göstermektedir. Tablo 4.7.'den, önerilen kombine özellik seti ile Decision Tree ve Random Forest'in, önerilen modelin sırasıyla %96.97 ve %99.20'lük bir genel doğruluğa sahip olduğu, yine sırasıyla Decision Tree için %4.60 ve Random Forest için %0.91 yanlış hata oranına (FPR) sahip olduğu görülmektedir. Bai ve ark. [11]'nın bulduğu sonuçlar ise sırasıyla %98.70 ve %98.90 olarak verilmiştir.

Tablo 4.7. Önerilen çalışmanın nümerik ve veri dizinlerın sonucunun önceki çalışmayla karşılaştırılması

Çalışmalar		**Doğruluk**	**Yanlış pozitif Oranı**
Önerilen çalışma	Decision Tree	96.97	4.60
	Random Forest	99.20	0.91
Vinod ve ark. [69]	Ibk	96.80	0.13
Bai ve ark. [11]	Decision Tree	98.70	1.4
	Random Forest	98.90	1.4
Shafiq ve ark. [8]	Decision Tree	99	0.5

Bai ve arkadaslarının elde etmiş olduğu sonuçlar train-test split (%80 eğitim seti ve %20 test seti)'e göre verilirken, önerdiğimiz çalışma sonucu elde edilen değerler ise

10 kat çapraz doğrulamaya göre verilmiştir. Önerilen yöntem sonucunda elde edilen sonuçlar, Bai ve ark. [11] ile hem Decision Tree hem de Random Forest kullanım karşılaştırması Tablo 4.7.'de verilmiştir. Verilen sonuçlara göre çözümümüz belirtilen sınıflandırma algoritmalarında sırasıyla - %1.73 ve %0.3 pozitif bir farkla üstünlük göstermiştir. Ayrıca çalışmamız bilinmeyen zararlı yazılımları %0.9 yanlış pozitif oranı ile tespit ederken Bai ve ark. aynı tespiti %1.4 yanlış pozitif oranı ile yapmaktadır.

Önerdiğimiz çözümden elde edilen sonuçlar, Tablo 4.7.'de verilen Shafiq ve ark. ile karşılaştırıldığında sonuçların Decision Tree'de %2.03'lük bir değerle daha düşük bir sonuç elde edilmiştir. Ayrıca Shafiq ve ark.'nın çalışmasında Random Forest sonuçları paylaşılmamıştır. Dolayısıyla bu konuda bir karşılaştırma sunulamamıştır.

4.2.3.2. Nümerik ve metinsel özellikler (derleyiciler, DLL ve API fonksiyonları)

Tablo 4.8. Metinsel ve nümerik özelliklere göre elde edilen sonuçlar.

Özellik Tipi	Sınıflandırıcı	Doğruluk	Kesinlik	Duyarlılık	F1-skoru	FPR	FNR
PE Başlıkları	kNN	99.03	98.40	98.77	99.03	0.78	1.77
	Decision Tree	99.22	98.76	98.95	99.22	0.39	1.20
	Light gbm	99.37	99.00	99.16	99.37	0.40	1.15
	Random Forest	**99.49**	**99.20**	**99.14**	**99.49**	**0.26**	**1.19**
Derleyiciler, DLLs & API Fonksiyonları	kNN	96.66	99.63	96.08	97.83	1.11	3.92
	DT	98.76	99.22	99.20	99.21	2.84	0.80
	Light gbm	98.84	99.33	99.18	99.26	2.37	0.82
	Random Forest	**99.06**	**99.20**	**99.60**	**99.40**	**2.92**	**0.40**
Birleştirlimiş Metinsel ve Nümerik özellikler							
PE Başlıkları, Derleyiciler, DLLs & API Fonksiyonları	kNN	95.80	96.65	97.01	96.83	1.27	4.12
	Decision Tree	98.76	99.07	99.06	99.07	2.78	0.68
	Light gbm	99.00	99.34	99.37	99.36	1.91	0.83
	Random Forest	**99.13**	**99.25**	**99.44**	**99.35**	**1.22**	**0.79**

Tablo 4.8, Metinsel ve Nümerik özelliklerin birleştirilmesi sonucunda elde ettiğimiz sonuçların yine önerdiğimiz diğer yöntemlerle karşılaştırmalı sonuçlarını

özetlemektedir. Tablo 4.8.'ten, önerilen kombine özellik seti ile Decision Tree ve Random Forest'in, önerilen modelin sırasıyla %98.76 ve %99.13'lük bir genel doğruluğa sahip olduğu, yine sırasıyla Decision Tree için %2.78 ve Random Forest için %1.22 yanlış hata oranına (FPR) sahip olduğu görülmektedir.

Literatürde zararlı ve zararsız yazılımları sınıflandırmak için yapılan çalışmalar ve yaptığımız çalışmanın sonuçları aşağıda verilen maddeler kapsamında karşılaştırmalı bir şekilde Tablo 4.9.'da sunulmuştur:

1. Sadece PE başlık bilgileri
2. Sadece veri dizinleri'ndeki (DLL ve API fonksiyonları) bilgiler
3. Sadece veri dizinleri bilgileri (DLL ve API fonksiyonları) ve derleyici bilgileri
4. Hem PE başlık bilgileri hem de Veri Dizinleri'ndeki bilgiler
5. PE başlık bilgileri, derleyici bilgileri ve Veri Dizinleri'ndeki bilgiler.

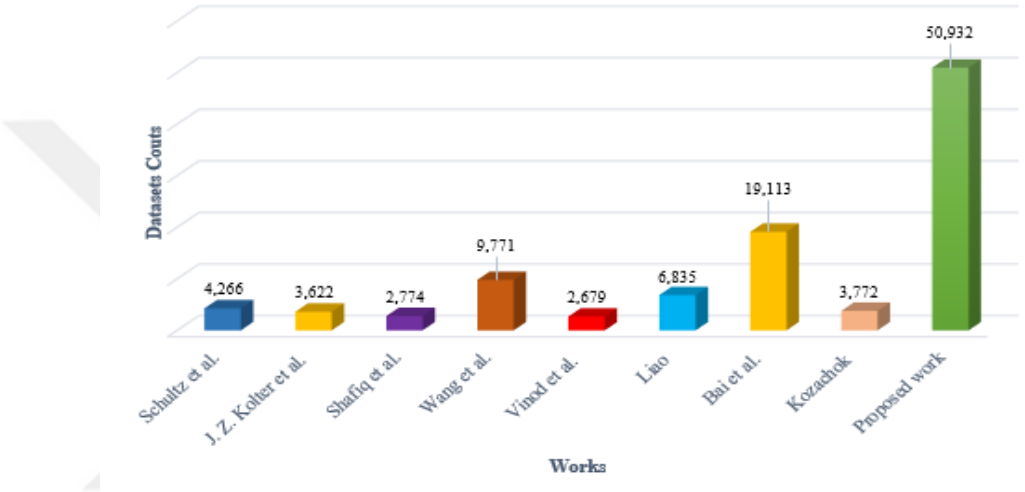
Ayrıca aynı tabloda veri setinde kullanılan zararlı ve zararsız yazılım sayıları, kullanılan özellik seçim teknikleri, tercih edilen sınıflandırma algoritmaları ve elde edilen FPR ve doğruluk sonuçları da verilmiştir.

Tablo 4.9. Veri setlerini karşılaştıran ilgili çalışma tablosu, özellik seçimi ve sınıflandırma algoritmaları ve sonuçları

	Veri Seti			Testler								
Yayın	Zararsız Sayısı	Zararlı Sayısı	Toplam	Sadece Başlık (Nümerik) (1)	Sadece Veri Dizinleri (DLL ve API Fonk.) (2)	Veri Dizinleri ve Derleyici Bilgileri (Metinsel) (3)	Başlık ve Veri Dizinleri (4)	Başlık, Veri Dizinleri ve Derleyici Bilgileri (5)	Özellik Seçim Teknikleri	Sınıflandırma Algoritmaları	FPR (Yanlış Pozitif Oranı)	Genel Sonucu Doğruluk (Accuracy)
Schultz et al. (2001), [5]	1,001	3,265	4,266	Hayır	Evet	Hayır	Hayır	Hayır	-	Ripper and Naïve Bayes	-	%97.11
J. Z. Kolter et al. (2004), [6]	1,971	1,651	3,622	Hayır	Evet	Hayır	Hayır	Hayır	Information gain	IBk, Naïve Bayes, DT, SVM, BoostedJ48	%0.05	%97
Shafiq et al. (2009), [8]	1,330	1,444	2,774	Hayır	Hayır	Hayır	Evet	Hayır	RFR, PCA, HWT	IBk, J48, NB, Ripper & SMO	%0.50	%99.00
Wang et al. (2009), [9]	1,908	7,863	9,771	Evet	Hayır	-	Hayır	Hayır	Information gain	SVM	-	%98.86
Liao (2012), [10]	1,237	5,598	6,835	Evet	Hayır	Hayır	Hayır	Hayır	-	If-else-rules	%0.20	%99.00
Bai et al. (2014), [11]	8,592	10,521	19,113	Hayır	Hayır	Hayır	Evet	Hayır	Filter, wrapper	DT, NB, BoostedTree and BaggedDT	%1.40	%99.01
Kozachok et al. (2017), [12]	1,862	1,910	3,772	Evet	Hayır	Hayır	Hayır	Hayır	-	DT, NN	-	%99.02
Vinod et al. (2011), [69]	1,085	1,594	2,679	Hayır	Hayır	-	Evet	Hayır	Scatter Criterion	SMO, IBk, J48, Adaboost1J48, RF	%0.13	%96.80
Önerilen Çalışma	11,1122	39,810	50,932	Evet	Hayır	Evet	Evet	Evet	SkB, RFE, L1, ExtraTree (Nümerik Özellikler için) ve TF-IDF Metinsel Özellikler için	kNN, DT, RF, Lgbm	(1) %0.26 (3) %2.92 (4) %0.91 (5) %1.22	(1) %99.49 (3) %99.06 (4) %99.20 (5) %99.13

PE başlık bilgilerini ve Veri Dizinleri'ndeki bilgileri kullanarak zararlı yazılım tespiti literatüründen aşağıdaki gözlemler ortaya çıkmaktadır:

1) Tablo 4.9.'daki, Kozachok [12] ve ark. gibi çoğu çalışmada yazarların 20,000 veya daha az örnek içeren veri setini kullandıkları görülmektedir. Yapılan bu çalışmada ise, eski ve yeni tip zararlı yazılımların iyi örneklenmesi için zararlı yazılım numuneleri çeşitlendirilerek 11,122'si zararsız ve 39,810'u zararlı olmak üzere 50,932 numunelik önişlenmiş veri seti kullanılmıştır.



Şekil 4.2. Veri seti karşilartırma

2) Tablo 4.9.'da kullanılan zararlı yazılımların elde edilmesi, şirketlere özel olduklarından veya çeşitli sebeplerden mümkün değildir. Bu yazılımların herkese açık olması tekrarlanan dosyaların çıkarılması, dosya tipinin bulunması ve etiketleme gibi çok zaman gerektiren önişleme süreçlerinin ortadan kaldırılmasını sağlayabilirdi. Bu çalışma bu sorunu dikkate alıp ilerideki zamanlarda yeni araştırmaların yapılması için etiketlenmiş bir standart veri setini sunmaktadır.

3) Zararlı yazılım tespiti için Tablo 4.9.'da geçen çoğu çalışma sadece PE başlığı özelliklerini kullanarak üç özellik seçim tekniğini kullanmaktadır. Bu çalışmada, Bölüm 3.4.'te anlatıldığı gibi, analiz veri setinden yararlı özellikleri bulmak için dört özellik seçim tekniği kullanılmıştır.

4) Zararsız ve zararlı yazılımları ayırt edebilmek için önemli bilgi taşıyan derleyici bilgileri çoğu çalışmada özellik olarak kabul edilmemiştir.

Bu çalışmada, yukarıda belirlenen gözlemler dikkate alınarak PE başlığı özelliklerini, DLL ve API fonksiyonları ve derleyici bilgileri özelliklerini birleştiren bir sistem oluşturulmuştur.

Farklı zararlı ve zararsız yazılım örneklerinden oluşan test veri seti ile doğrulandığında, alt bölüm 4.2.3.'te gösterildiği gibi, bu çalışmada geliştirilen yöntem daha iyi bir doğruluk göstermektedir.

BÖLÜM 5. SONUÇ VE GELECEK ÇALIŞMALAR

Bu tez çalışmasında, bir makine öğrenme yaklaşımı kullanılarak bir zararlı yazılım tespit sistemi sunulmuştur. Zararsız veya zararlı yazılımları sınıflandırmak için nümerik özellikler kapsamında PE başlığı bilgisi, metinsel özellikler kapsamında ise derleyici bilgileri, kütüphane (DLL) ve API fonksiyonları kullanılmıştır.

Önerilen çalışmada, ilgili özellikleri çıkartabilmek için dinamik analizden daha az zaman ve kaynak gereksinimi olan statik analiz tekniği tercih edilmiştir.

Ayrıca Univariate feature selection (*SelectKBest*), Recursive Feature Elimination (RFE), L1-based feature selection (L1) ve Tree-based feature selection (ExtraTree) özellik seçim teknikleri kullanılmış ve seçilen özelliklerin performansı karşılaştırılmıştır.

Zararlı yazılımları tespit etmek için önceki araştırmalarda olduğu gibi, makine öğrenme yaklaşımları kapsamında kullanılan kNN, DT, RF, Ligth GBM sınıflandırıcıları ile başarılı sonuçlar alınabileceği görülmüştür. Ancak ticari alanda elde edilen yanlış pozitif oranı, makine öğrenme modellerinden elde edilen oranlardan daha düşük olmalıdır. Farklı makine öğrenme algoritmaları sınıflarını eğiterek ve test ederek sınıflandırıcıların performansının Bölüm 1.2.'de verilen önceki çalışmalara göre önemli ölçüde arttığı tespit edilmiştir. Sınıflandırıcılardan sağlam ve kararlı bir performans ölçümü yapmak için onaylama 10 kat çapraz doğrulama ile gerçekleştirilmiştir.

Deneylerden ilkinde sadece PE başlıklarına (nümerik) ait özellikler kullanılmış, Random Forest sınıflandırıcısı ile %99.49 doğruluk elde edilmiştir. İkincisinde Derleyici bilgileri, kütüphaneler ve API fonksiyonlarına ait metinsel özellikler kullanılmış, yine Random Forest sınıflandırıcısı ile %99.06’lık tespit doğruluğu elde

edilmiştir. Son olarak ise nümerik ve metinsel özellikler birleştirilerek test yapılmış ve bu test sonucunda da genel olarak yine %99.44 doğruluk, %1.2 yanlış pozitif, %99.25 kesinlik ve %99.13 doğruluk oranları ile en iyi performans Random Forest ile elde edilmiştir.

Eski ve yeni zararlı yazılım türlerinin iyi örneklenmesi için zararlı yazılım veri seti çeşitlendirilmiştir. Önerilen bu çalışma önceden işlenmiş ve 50,932 numuneden (11,122 zararsız ve 39,810 zararlı yazılım) oluşan bir veri setini sağlamaktadır.

Kullanılan veri seti, [67]'da önerildiği gibi zararlı yazılım analizi için gerekli metrikler kullanılarak önişlemeye alınmıştır. Analiz için bir veri setinin elde edilip ön-işlenmesi çok vakit alan bir süreç olduğu için hazır bu veri seti, bu çalışmada geçen deneyleri yeniden yapıp aynı sonuçları elde etmeleri veya farklı yöntemleri kullanıp yeni araştırma yapmaları için zararlı yazılım analizi alanındaki araştırmacılara önerilmektedir.

Statik analiz, doğru özelliklerle kullanıldığı takdirde bir yazılım çalıştırılmadan o yazılımın zararlı olup olmadığına karar vermektedir. Bu oldukça önemli bir özelliktir. Bilgisayar sistemlerindeki etkileşimler sistemi sürekli değiştirebildiği için yazılımlar normal davranışların dışına çıkabilir. Böylelikle zararsız bilinen bir yazılım çalışırken zararlı bir davranış sergileyebilir. Bu durumda çözüm olarak dinamik analiz kullanılabilir. Dinamik analiz, bir yazılım sistemde proses olarak çalışırken o yazılımın zararlı olup olmadığına karar verebilir. Bu özellik, gerçek zamanlı zararlı yazılım tespit sisteminin gerçekleştirilmesini sağlayabilir. Statik analizin zayıf kaldığı noktalarda dinamik analiz kullanılarak zararlı yazılım tespit sistemi daha iyi sonuç verebilir. Bu sebepten dolayı statik ve dinamik analiz yöntemlerini birlikte kullanan ve gerçek zamanlı zararlı yazılım tespit sisteminin gerçekleştirilmesi sonraki çalışma olarak ele alınabilir bir konudur.

KAYNAKLAR

[1] AV-TEST Institute https://www.av-test.org/en/statistics/malware/, Erişim Tarihi: 20.04.2019.

[2] Chess, D. M., & White, S. R. (2000, September). An undetectable computer virus. In Proceedings of Virus Bulletin Conference (Vol. 5, pp. 1-4).

[3] Shalaginov, A., Banin, S., Dehghantanha, A., & Franke, K. (2018). Machine learning aided static malware analysis: A survey and tutorial. In Cyber Threat Intelligence (pp. 7-45). Springer, Cham.

[4] Cohen, F. (1987). Computer viruses: theory and experiments. Computers & security, 6(1), 22-35.

[5] Schultz, M. G., Eskin, E., Zadok, F., & Stolfo, S. J. (2000, May). Data mining methods for detection of new malicious executables. In Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001 (pp. 38-49). IEEE.

[6] Kolter, J. Z., & Maloof, M. A. (2004, August). Learning to detect malicious executables in the wild. In Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 470-478). ACM.

[7] Dai, J. (2009). Detecting Malicious Software By Dynamicexecution.

[8] Shafiq, M. Z., Tabish, S. M., Mirza, F., & Farooq, M. (2009, September). Pe-miner: Mining structural information to detect malicious executables in realtime. In International Workshop on Recent Advances in Intrusion Detection (pp. 121-141). Springer, Berlin, Heidelberg.

[9] Wang, T. Y., Wu, C. H., & Hsieh, C. C. (2009, August). Detecting unknown malicious executables using portable executable headers. In 2009 Fifth International Joint Conference on INC, IMS and IDC (pp. 278-284). IEEE.

[10] Liao, Y. (2012). Pe-header-based malware study and detection. Retrieved from the University of Georgia: http://www. cs. uga. edu/~ liao/PE_Final_Report. pdf.

[11] Bai, J., Wang, J., & Zou, G. (2014). A malware detection scheme based on mining format information. The Scientific World Journal, 2014.

[12] Kozachok, A. V., & Kozachok, V. I. (2018). Construction and evaluation of the new heuristic malware detection mechanism based on executable files static analysis. Journal of Computer Virology and Hacking Techniques, 14(3), 225-231.

[13] Carrera, E. (2006). PEFILE–a Python module to read and work with PE (Portable Executable) files.

[14] Yonts, J. (2012). Attributes of malicious files. SANS Institute InfoSec Reading Room.

[15] Desktop Operating System Market Share Worldwide - May 2018, http://gs.statcounter.com/os-market-share/desktop/worldwide, Erişim Tarihi: 20.06.2018.

[16] Browse Open Source Software, https://sourceforge.net/directory/os:windows/, Erişim Tarihi: 15.05.2018.

[17] CNET Download, https://download.cnet.com/windows/, Erişim Tarihi: 15.05.2018.

[18] Virusshare.com, http://virusshare.com/, Erişim Tarihi: 15.05.2018.

[19] VX heaven, https://archive.org/download/vxheaven-windows-virus-collection Erişim Tarihi: 16.05.2018.

[20] Monnappa, K. A. (2018). Learning Malware Analysis: Explore the concepts, tools, and techniques to analyze and investigate Windows malware.

[21] Sikorski, M., & Honig, A. (2012). Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press.

[22] Szor, P. (2005). Advanced code evolution techniques and computer virus generator kits. The Art of Computer Virus Research and Defense.

[23] Aycock, J. (2006). Computer Viruses and Malware'Springer Advances in Information Security, Vol. 22, 2006.

[24] Trendmicro, Ransomware, https://www.trendmicro.com/vinfo/us/security/definition/ransomware, Erişim Tarihi: 22.08.2019.

[25] Trojan.Dropper, https://www.symantec.com/security-center/writeup/2002-082718-3007-99, Erişim Tarihi: 22.08.2019.

[26] Microsoft Corporation. Naming malware. https://docs.microsoft.com/en-gb/windows/security/threat-protection/intelligence/malware-naming, Erişim Tarihi: 28.08.2019.

[27] 'Scareware' Distributors Targeted. https://www.fbi.gov/news/stories/scareware-distributors-targeted, Erişim Tarihi: 28.08.2019.

[28] Mcafree. Meet 'Tox': Ransomware for the Rest of Us. https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/meet-tox-ransomware-for-the-rest-of-us/, Erişim Tarihi: 28.08.2019.

[29] Bontchev, V. (1993). Possible virus attacks against integrity programs and how to prevent them (pp. 118-130). Technical report, Virus Test Center, University of Hamburg.

[30] The PE file format, http://www.pelib.com/resources/luevel.txt, Erişim Tarihi: 18.03.2019.

[31] Pietrek, M. Peering Inside the PE: A Tour of the Win32 Portable Executable File Format,bytepointer.com/resources/pietrek_peering_inside_pe.htm, Erişim Tarihi:18.03.2019.

[32] Prasad, B. J., Annangi, H., & Pendyala, K. S. (2016). Basic static malware analysis using open source tools.

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

[34] Kramer, S., & Bradfield, J. C. (2010). A general definition of malware. Journal in computer virology, 6(2), 105-114.

[35] Mujumdar, A., Masiwal, G., & Meshram, D. B. (2013). Analysis of signature-based and behavior-based anti-malware approaches. International Journal of Advanced Research in Computer Engineering and Technology (IJARCET), 2(6).

[36] Christopher, D. M., Prabhakar, R., & Hinrich, S. (2008). Introduction to information retrieval. An Introduction To Information Retrieval, 151(177), 5.

[37] Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. Communications of the ACM, 18(11), 613-620.

[38] Anand Rajaraman, A., & Ullman, J. D. (2011). Mining of massive datasets. Cambridge University Press.

[39] Gupta, M., & Bendersky, M. (2015). Information retrieval with verbose queries. Foundations and Trends® in Information Retrieval, 9(3-4), 209-354.

[40] Salton, G., & Buckley, C. (1987). Term weighting approaches in automatic text retrieval. Cornell University.

[41] Raschka, S. (2014). About Feature Scaling and Normalization (and the effect of standardization for Machine Learning algorithms). Polar Political & Legal Anthropology Review, 30(1), 67-89.

[42] Simon P. (2012). "The Elements of Persuasion: Big Data Techniques". In: Too Big to Ignore. John Wiley Sons, Inc., pp. 77–109.

[43] Hastie, T., Tibshirani, R., Friedman, J., & Franklin, J. (2005). The elements of statistical learning: data mining, inference and prediction. The Mathematical Intelligencer, 27(2), 83-85.

[44] Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning (Vol. 1, No. 10). New York: Springer series in statistics.

[45] Massoud Farahmand, A., & Szepesvári, C. Model Selection in Reinforcement Learning.

[46] Webb, A. R. (2003). Statistical pattern recognition. John Wiley & Sons.

[47] Scikit-Learn, http://scikit-learn.org/stable/modules/feature_selection.html, Erişim Tarihi: 29.08.2019.

[48] Liao, Y., & Vemuri, V. R. (2002). Use of k-nearest neighbor classifier for intrusion detection. Computers & security, 21(5), 439-448.

[49] Alpaydin, E. (2009). Introduction to machine learning. MIT press.

[50] Quinlan, J. R. (1986). Induction of decision trees. Machine learning, 1(1), 81-106.

[51] Polamuri, S. (2017). How the random forest algorithm works in machine learning. Retrieved December, 21.

[52] Light GBM, https://lightgbm.readthedocs.io/en/latest/, Erişim Tarihi: 22.08.2019.

[53] Mandot, P.(2017). "What is LightGBM, How to implement it? How to fine tune the parameters?" Retrieved August 14 (2017).

[54] Machine Learning Crash Course, https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc, Erişim Tarihi: 25.08.2019.

[55] Pietrek, M. (1994). Peering inside the PE: a tour of the win32 (R) portable executable file format. Microsoft Systems Journal-US Edition, 9(3), 15-38.

[56] Turney, P. D., & Pantel, P. (2010). From frequency to meaning: Vector space models of semantics. Journal of artificial intelligence research, 37, 141-188.

[57] Feature Selection Methods in Machine Learning, https://medium.com/@sagar.rawale3/feature-selection-methods-in-machine-learning-eaeef12019cc, Erişim Tarihi: 27.08.2019.

[58] Guyon, I., Gunn, S., Nikravesh, M., & Zadeh, L. A. (Eds.). (2008). Feature extraction: foundations and applications (Vol. 207). Springer.

[59] Raschka, S., & Mirjalili, V. (2017). Python machine learning. Packt Publishing Ltd.

[60] Belaoued, M., & Mazouzi, S. (2016). A Chi-Square-Based Decision for Real-Time Malware Detection Using PE-File Features. Journal of Information Processing Systems, 12(4).

[61] Qiao, M., Sung, A. H., & Liu, Q. (2016, July). Merging permission and API features for Android malware detection. In 2016 5th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI) (pp. 566-571). IEEE.

[62] Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. Machine learning, 46(1-3), 389-422.

[63] Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. Machine learning, 63(1), 3-42.

[64] Matthew J. Lavin, "Analyzing Documents with TF-IDF," The Programming Historian 8 (2019), https://programminghistorian.org/en/lessons/analyzing-documents-with-tfidf, Erişim Tarihi: 28.08.2019.

[65] Pedregosa, Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. Journal of machine learning research, 12(Oct), 2825-2830.

[66] https://stanford.edu/~shervine/l/tr/teaching/cs-229/cheatsheet-machine-learning-tips-and-tricks#model-selection. Erişim Tarihi: 30.08.2019.

[67] Danesh, A., Moshiri, B., & Fatemi, O. (2007, July). Improve text classification accuracy based on classifier fusion methods. In 2007 10th International Conference on Information Fusion (pp. 1-6). IEEE.

[68] Firdausi, I., Erwin, A., & Nugroho, A. S. (2010, December). Analysis of machine learning techniques used in behavior-based malware detection. In 2010 second international conference on advances in computing, control, and telecommunication technologies (pp. 201-203). IEEE.

[69] Vinod, P., Laxmi, V., & Gaur, M. S. (2011, July). Scattered feature space for malware analysis. In International Conference on Advances in Computing and Communications (pp. 562-571). Springer, Berlin, Heidelberg.

EKLER

Çıkarılan Özelliller

EK A: Nümerik Özellikler

1	AddressOfEntryPoint	31	ExportTableRva	61	ResourceTableRVA
2	ArchitectureRVA	32	ExportTableSize	62	ResourceTableSize
3	ArchitectureSize	33	FileAlignment	63	SectionAlignment
4	BaseOfCode	34	GlobalPtrRVA	64	SectionsMaxRawsize
5	BaseOfData	35	GlobalPtrSize	65	SectionsMeanRawsize
6	BaseRelocationTableRVA	36	IATRVA	66	SectionsMinRawsize
7	BaseRelocationTableSize	37	IATSize	67	Size
8	BoundImportRVA	38	ImageBase	68	SizeOfCode
9	BoundImportSize	39	ImportTableRVA	69	SizeOfHeaders
10	CLRRuntimeHeaderRVA	40	ImportTableSize	70	SizeOfHeapCommit
11	CLRRuntimeHeaderSize	41	LoadConfigTableRVA	71	SizeOfHeapReserve
12	CertificateSecTableRVA	42	LoadConfigTableSize	72	SizeOfImage
13	CertificateSecTableSize	43	LoaderFlags	73	SizeOfInitializedData
14	Characteristics	44	Machine	74	SizeOfOptionalHeader
15	CheckSum	45	Magic	75	SizeOfStackCommit
16	CodeDataRation	46	MajorImageVersion	76	SizeOfStackReserve
17	CountNonSuspiciousSections	47	MajorLinkerVersion	77	SizeOfUninitializedData
18	CountSuspiciousSections	48	MajorOperatingSystemVersion	78	Subsystem
19	DebugRVA	49	MajorSubsystemVersion	79	TLSTableRVA
20	DebugSize	50	MinorImageVersion	80	TLSTableSize
21	DelayImportDescriptorRVA	51	MinorLinkerVersion	81	TimeDateStamp
22	DelayImportDescriptorSize	52	MinorOperatingSystemVersion	82	VirtualAddress
23	DllCharacteristics	53	MinorSubsystemVersion	83	VirtualSize
24	E_lfanew	54	NumberOfImportedDLLs		
25	E_magic	55	NumberOfImportedFunctions		
26	Entropy	56	NumberOfRvaAndSizes		
27	EntropySectionData	57	NumberOfSections		
28	EntropySectionText	58	NumberOfSymbols		
29	ExceptionTableRVA	59	PE_Type		
30	ExceptionTableSize	60	PointerToSymbolTable		

EK B: Nümerik Olmayan Özellilker

1. EntropySections
2. FileType
3. FormatedTimeDateStamp
4. Fuzzy
5. Compilers Information
6. ImportedDlls
7. ImportedFunctions
8. MD5
9. Filename
10. SHA1
11. SectionName

EK C: Metinsel Özellikler

1. Compilers Information
2. ImportedDlls
3. ImportedFunctions

C.1. ImportedDlls

1	advapi32	41	setupapi	81	cc3260mt
2	apimswincorecoml111	42	shell32	82	cygregex
3	apimswincoredebugl111	43	shlwapi	83	svrapi
4	apimswincoredelayloadl111	44	urlmon	84	pstorec
5	apimswincoreerrorhandlingl111	45	user32	85	mfc70
6	apimswincorefilel121	46	version	86	bcrypt
7	apimswincorehandlel110	47	winhttp	87	uxtheme
8	apimswincoreheapl120	48	wininet	88	propsys
9	apimswincoreheapl210	49	winmm	89	slc
10	apimswincorelibraryloaderl120	50	winspooldrv	90	msacm32
11	apimswincorelocalizationl121	51	ws232	91	msvbvm50
12	apimswincoreprocessenvironmentl120	52	wsock32	92	iphlpapi
13	apimswincoreprocessthreadsl112	53	rasapi32	93	msvfw32
14	apimswincoreprofilel110	54	hal	94	invalid
15	apimswincoreregistryl110	55	imagehlp	95	crt
16	apimswincorestringl110	56	atl	96	dnsapi
17	apimswincoresynchl120	57	oledlg	97	wintrust
18	apimswincoresysinfol121	58	olepro32	98	cfgmgr32
19	apimswineventingproviderl110	59	odbc32	99	wincorlib
20	apimswinsecuritybasel120	60	msvcirt	100	mpclient
21	comctl32	61	cygcrypt0		
22	comdlg32	62	wtsapi32		
23	crypt32	63	mswsock		
24	gdi32	64	msvcr80		
25	gdiplus	65	sfc		
26	imm32	66	oleacc		
27	kernel32	67	userenv		
28	mfc42	68	dbghelp		
29	mpr	69	msimg32		
30	mscoree	70	cyggnutlsopenssl11		
31	msvbvm60	71	rtl60bpl		
32	msvcp60	72	libcurl		
33	msvcrt	73	cyggeoip1		
34	netapi32	74	sqlite		
35	ntdll	75	libxml2		
36	ole32	76	vfw32		
37	oleaut32	77	ndissys		
38	psapi	78	opengl32		
39	rpcrt4	79	borlndmm		
40	dui70	80	combase		

C.2. ImportedFunctions

1	acmdln	41	enddialog	81	getfilesize
2	adjstfdiv	42	endpaint	82	getfiletype
3	amsgexit	43	entercriticalsection	83	getlocaleinfoa
4	beginpaint	44	excepthandler3	84	getlocaleinfow
5	cexit	45	excepthandler4common	85	getlocaltime
6	closehandle	46	exit	86	getmainargs
7	cocreateinstance	47	exitprocess	87	getmodlefilenamea
8	coinitialize	48	expandenvironmentstringsw	88	getmodlefilenamew
9	coinitializeex	49	filetimetosystemtime	89	getmodlehandlea
10	comparestringw	50	fillrect	90	getmodlehandleexw
11	controlfp	51	findclose	91	getmodlehandlew
12	cotaskmemalloc	52	findfirstfilea	92	getoemcp
13	cotaskmemfree	53	findfirstfilew	93	getparent
14	coninitialize	54	findnextfilea	94	getprocaddress
15	createcompatibledc	55	findnextfilew	95	getprocessheap
16	createdirectorya	56	findresorcew	96	getstartpinfoa
17	createdirectoryw	57	flshfilebffers	97	getstartpinfow
18	createeventw	58	formatmessagew	98	iswindow
19	createfilea	59	free	99	killtimer
20	createfilemappingw	60	freeenvironmentstringsa	100	typeinfoaexz
21	createfilew	61	freeenvironmentstringsw		
22	createprocessa	62	freelibrary		
23	createthread	63	getacp		
24	createwindowexa	64	getclientrect		
25	createwindowexw	65	getcommandlinea		
26	cxxframehandler3	66	getcommandlinew		
27	cxxthrowexception	67	getconsolecp		
28	decodepointer	68	getconsolemode		
29	defwindowproca	69	getcpinfo		
30	deletecriticalsection	70	getcrrentprocessid		
31	deletedc	71	getcrrentthread		
32	deletefilea	72	getcrrentthreadid		
33	deletefilew	73	getdc		
34	deleteobject	74	getdevicecaps		
35	destroywindow	75	getdlgitem		
36	dispatchmessagea	76	getenvironmentstrings		
37	dispatchmessagew	77	getenvironmentstringsw		
38	dllonexit	78	getexitcodeprocess		
39	enablewindow	79	getfileattribtesa		
40	encodepointer	80	getfileattribtesw		

C.3. Derleyici Belgeleri (Compiler Informations)

1	microsoftvisualc8	41	petitev22	81	execryptor22xsoftcompletedevelopement
2	microsoftvisualbasicv50v60	42	installshieldcustom	82	ahteamepprotector03fakepcguard403415feuerrader
3	microsoftvisualbasicv50	43	acprotectv132	83	innosetupmoduleheuristicmode
4	Microsoftvisualc	44	Microsoftcabsfx	84	fsgv110engdulekxtmicrosoftvisualc60
5	microsoftvisualc50	45	upxv125delphistub	85	xcexc4xbcxfexc0xa6xb0xf3xc6xf7v10xd0xedxd4xc6
6	microsoftvisualcv60	46	bobSoftminidelphi	86	execryptorv21xsoftcompletecom
7	Asprotectvxx	47	aspackv10804	87	armadillov430440siliconrealmstoolworks
8	armadillov1xxv2xx	48	Microsoftvisualcvxx	88	xdcxa5xb1xb1xd1xb9xcbxf522bantixiaohui
9	microsoftvisualc60	49	purebasic4xneilhodgson	89	devc4992bloodshedsoftware
10	nullsoftpimpstubsfx	50	freebasic014	90	asprotectv2xdllalexeysolodovnikov
11	microsoftvisualc60dll	51	aspack102bor10803	91	bobsoftminidelphibobbobsoft
12	upxv0896v102v105v122	52	pecompactv20	92	pecrypt32consolev10v101v102
13	microsoftvisualcv70	53	asdpack20asd	93	pellesc300400450exex86crtlib
14	borlanddelphi30	54	nspack3x	94	acprotect13x14xdllriscosoftwareinc
15	armadillov4x	55	microsoftvisualc20	95	ASProtectv12xNewStrain
16	microsoftvisualc70	56	upxv0896v102v105v122dll	96	armadillo500siliconrealmstoolworks
17	safeguard103simonzh	57	borlanddelphiv50kolmck	97	starforceproactive11starforcetechnology
18	borlanddelphi40	58	crunchpe	98	armadillo500siliconrealmstoolworks
19	installshield2000	59	microsoftvisualcv60dll	99	armadillo3x5xsiliconrealmstoolworks
20	installervisecustom	60	winkriptv10	100	ThinstallEmbeddedV2501
21	aspackv21	61	nsisinstallernullsoft		
22	pklite32v11	62	packmanV10		
23	microsoftvisualc80	63	alloy4xpgwarellc		
24	aspackv212	64	thinstall24x25x		
25	installshieldcustom	65	peprotector093cryptocrack		
26	lccwin32v1x	66	asprotectv2xdll		
27	devcv5	67	wiseinstallerstub		
28	Borlandcdll	68	nullsoftpimpinstallsystem		
29	borlanddelphiv30	69	rarsfx		
30	upxv080v084	70	innoinstallerv512		
31	ezipv10	71	microsoftvisualcv71dlldebug		
32	dosdevicedriver	72	armadillov2xxcopymemll		
33	codesafev20	73	borlanddelphi		
34	zealpack10zeal	74	pecompact253dll		
35	mingwin32gcc3x	75	pecryptv100v101		
36	mingwgcc3x	76	kbyspackerv028		
37	petite14c199899ianluckh	77	aspackv211		
38	microsoftvisualbasic40	78	vmprotectv125polytech		
39	borlanddelphiv60v70	79	upxv30exelzma		
40	gleamv100	80	microsoftvisualc42		

ÖZGEÇMİŞ

Sefu MOHAMED, Bujumbura'da doğdu. İlk, orta ve lise eğitimini Bujumbura'da tamamladı. 2008 yılında İslam Kültür Merkezi (Lycée du Centre Culturel İslamique) Lisesi'nden mezun oldu. 2008 yılında başladığı Lac Tanganyika Üniversitesi Bilgisayar Mühendisliği Bölümü'nü 2013 yılında bitirdi. 17 Eylül 2013-10 Temmuz 2015 Lisede (Lycée du Centre Culturel İslamique) Bilgi Teknolojileri Öğretmeni olarak çalıştı. TÜBİTAK Bursunu (2235 numaralı) kazandıktan sonra, 2015-2016, Sakarya Üniversitesi Türk Dili Araştırma Merkezi'nde Türkçe Eğitimi (Tömer) tamamladı ve 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nde yüksek lisans eğitimine başladı.