

TÜRKİYE CUMHURİYETİ
YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK - ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ



**ANDROID ZARARLI YAZILIMLARIN WORD
EMBEDDINGS VE MAKİNE ÖĞRENMESİ İLE
AİLELERİNE GÖRE KATEGORİLENMESİ**

14011903 — Alibek ERKABAYEV

12011015 — Fatih ÇOMAK

BİLGİSAYAR PROJESİ

Danışman
Arş.Gör. Alper EĞİTMEN

Haziran, 2019

TEŞEKKÜR

Projenin geliştirme sürecinde bilgi ve tecrübeleriyle bize yol gösteren ve her türlü imkanı sağlayan bölümümüzün değerli hocalarından, aynı zamanda proje danışmanımız sayın Prof. Dr. Oya KALIPSIZ hocamıza sonsuz teşekkürlerimizi sunuyoruz.

Çalışmalarımızda bize moral ve destek veren, aynı zamanda anlayış gösteren arkadaşlarımıza ve ailemize tüm kalbimizle teşekkür ederiz.

Alibek ERKABAYEV

Fatih ÇOMAK

İÇİNDEKİLER

SİMGE LİSTESİ	v
KISALTMA LİSTESİ	vi
ŞEKİL LİSTESİ	vii
TABLO LİSTESİ	viii
1 Giriş	1
1.1 Proje Hedefleri	1
1.2 Proje Kapsamı/Yöntemler	1
1.3 Proje Çıktıları	2
2 Ön İnceleme	3
3 Fizibilite	5
3.1 Teknik Fizibilite	5
3.1.1 Yazılım Fizibilitesi	5
3.1.2 Donanım Fizibilite	6
3.2 İletişim Fizibilitesi	6
3.3 Yasal Fizibilite	6
3.4 Ekonomik Fizibilite	6
3.5 İşgücü ve Zaman Fizibilitesi	7
4 Sistem Analizi	8
5 Sistem Tasarımı	10
5.1 Uygulama Tasarımı	10
5.1.1 Ön İşleme	10
5.1.2 CBOW Uygulanması	11
5.1.3 GloVe Uygulanması	12
5.2 Veri Seti Tasarımı	13
5.3 Girdi - Çıktı Tasarımı	13

6	Uygulama	15
7	Deneysel Sonular	18
8	Sonu	19
	Referanslar	20
	Özgemiř	21

SİMGE LİSTESİ

₺	Türk Lirası
---	-------------

KISALTMA LİSTESİ

API	Application Programming Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CBOW	Continuous Bag of Words
Colab	Colaboratory
GB	GigaByte
GloVe	Global Vectors for Word Representation
GPU	Graphics Processing Unit
RAM	Random Access Memory
Weka	Waikato Environment for Knowledge Analysis

ŞEKİL LİSTESİ

Şekil 3.1	Gantt Diyagramı Çizelgesi	7
Şekil 3.2	Gantt Diyagramı Zaman Grafiği	7
Şekil 4.1	Sistemin Akış Diyagramı	9
Şekil 5.1	Hash Tablosu	10
Şekil 5.2	Opcode to Hashing Opcode Dönüşümü Örneği	11
Şekil 5.3	CBOW Word2Vec Modeli Örneği	11
Şekil 5.4	CBOW modelin Neural Network'te gösterimi [5]	12
Şekil 5.5	Random Forest Algoritmasına Göre CBOW Sonuçları	13
Şekil 5.6	Random Forest Algoritmasına Göre GloVe Sonuçları	13
Şekil 6.1	Preprocessing	15
Şekil 6.2	Word2Vec ile Vektör Elde Etme	16
Şekil 6.3	Weka Loading Train File	16
Şekil 6.4	Weka Classifying From Test File	17
Şekil 6.5	Step of Using Codes from GitHub	17

TABLO LİSTESİ

Tablo 3.1	Yazılım Fizibilite Tablosu	5
Tablo 3.2	Donanım Fizibilite Tablosu	6
Tablo 3.3	Maliyet Analizi	6
Tablo 5.1	Veri Seti Yapısı	14
Tablo 7.1	Sınıflandırılmış Sonuçlar	18

ANDROID ZARARLI YAZILIMLARIN WORD EMBEDDINGS VE MAKİNE ÖĞRENMESİ İLE AİLELERİNE GÖRE KATEGORİLENMESİ

Alibek ERKABAYEV

Fatih ÇOMAK

Bilgisayar Mühendisliği Bölümü

Bilgisayar Projesi

Danışman: Arş.Gör. Alper EĞİTMEN

Mobil cihazların kişisel bilgisayarlara kıyasla yaygın olarak kullanılması, bilgi alışverişinde yeni bir döneme yol açtı. Taşınabilir cihazların taşınabilirlik özellikleri ile birlikte artan gücü, kullanıcıların dikkatini çekmiş ve piyasanın gözünü üzerine çevirmiştir. Bu tür cihazlar sadece kullanıcılar arasında popüler olmakla kalmayıp aynı zamanda saldırganların favori hedefleri haline gelmiştir. Kullanıcı verilerinin çalınması, özel mesajların gönderilmesi ve kullanıcıların bilmediği özel numaralara telefon görüşmesi yapılması gibi kötü amaçlı etkinliklerle mobil kötü amaçlı yazılımların sayısı hızla artmıştır. Bu yüzden Android kötü amaçlı yazılımlarının tespiti ve sınıflandırılması için daha etkili teknikler gerekmiştir. Bu projede Android kötü amaçlı yazılımları sınıflandırmak ve kategorize etmek için makine öğrenmesini kullanan opcode analizine dayalı bir yaklaşım sunmaktayız.

Anahtar Kelimeler: makine öğrenmesi, word embedding, CBOW, GloVe, Word2Vec, android zararlı yazılım tespit, baksmalı opcode

YILDIZ TEKNİK ÜNİVERSİTESİ
ELEKTRİK - ELEKTRONİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

ABSTRACT

CLASSIFICATION OF ANDROID MALWARE BY FAMILIES WITH WORD EMBEDDING AND MACHINE LEARNING

Alibek ERKABAYEV

Fatih ÇOMAK

Department of Computer Engineering
Computer Project

Advisor: Assist. Alper EĞİTMEN

The widespread use of mobile devices in comparison to personal computers has led to a new era of information exchange. The increasing power of mobile devices along with portability characteristics has attracted the attention of users. Not only are such devices popular among users, but they are favorite targets of attackers. The number of mobile malware is rapidly on the rise with malicious activities, such as stealing users data, sending premium messages and making phone call to premium numbers that users have no knowledge. This calls for more effective techniques for detection and classification of Android malware. Hence, in this paper we present an opcode analysis based approach that utilizes machine learning to classify and categorize Android malware.

Keywords: machine learning, word embedding, CBOW, GloVe, Word2Vec, android malware detection, baksmali opcode

YILDIZ TECHNICAL UNIVERSITY
FACULTY OF ELECTRICAL AND ELECTRONICS
DEPARTMENT OF COMPUTER ENGINEERING

1.1 Proje Hedefleri

Android İşletim Sistemi, açık kaynak kodlu olduğu için mevcut işlevsellik konusunda daha fazla hareket serbestliği tanımaktadır. Android İşletim Sisteminin yerleşik güvenlik modeli, sağlam koruma mekanizmaları içermektedir; ancak yine de kötü amaçlı yazılımlarla ilgili bazı dezavantajları vardır. Modern internetin popülaritesi ve gelişimi bu tür programları yüksek hızda birçok mobil cihaza yaymayı mümkün kılmaktadır[1]. "Human-centric Computing and Information Sciences" dergisinin 2018'de yayınladığı araştırmaya göre mobil cihazlara yapılan tüm saldırıların %40'ının Android İşletim Sistemli cihazlara yapıldığı bilinmektedir[2].

Yukarıda bahsedilen güvenlik açıklarına ilişkin tespit yazılımları yapılmaya çalışılmaktadır. Bu sayede zararlı yazılımlar tespit edilip engellenmeye çalışılmaktadır. Bu projede bazı modeller büyük bir veri seti aracılığıyla kullanılarak makine öğrenmesi yaptırılacaktır. Bu zararlı yazılımlar tespit edilerek ailelerine göre kategorilendirilmesi hedeflenmektedir. Bu işlem CBOW ve GloVe gibi bazı doğal dil işleme yöntemleriyle yapılacaktır. Fakat asıl önemli nokta şudur: Programlama dilinin kaynak kodları yani fonksiyonları, baksmalı [3] isimli yöntemle opcode yani metin haline getirilip bu yaklaşımla analiz yapılacaktır.

1.2 Proje Kapsamı/Yöntemler

Android İşletim Sisteminde zararlı yazılımları tespit etmek için yazılımların kaynak kodları word embeddings ve makine öğrenmesi ile kategorilize edilecek, ailelerine ayrıştırılacak.

Word Embedding olarak Word2Vec modeli ve ayrıca Stanford Üniversitesi'nin geliştirdiği GloVe(Global Vectors for Word Representation) modeli kullanılacak[4]. Word2Vec, kelimeleri vektör uzayında ifade etmeye çalışan unsupervised (no labels) ve tahmin temelli(prediction-based) bir modeldir. İki çeşit alt yöntemi vardır: CBOW(Continuous Bag of Words) ve Skip-Gram. Bu projede ise CBOW yöntemi

uygulanacak. CBOW yönteminde pencere boyutının merkezinde olmayan kelimeler input olarak alınıp , merkezinde olan kelimeler output olarak tahmin edilmeye çalışılıyor.

Kullanılacak teknolojiler:

1. Python ve C++ programlama dili
2. Google Colab
3. Google Cloud
4. Tesla K80 GPU
5. Tensorflow library
6. Weka 3.8.3

1.3 Proje Çıktıları

Android İşletim Sistemindeki zararlı yazılımlar tespit edilip kategorilendirilecek ve API'ye entegrasyonu sağlanacaktır.

Proje kapsamında mobil zararlı yazılımların ailelerine göre kategorize edilmesi ve zararlı yazılım tespiti konusunda CBOW ve GloVe yöntemlerinin başarımı ölçülecektir.

2 Ön İnceleme

Önceki çalışmalar birçok kategoriye ayrılabilir. Örneğin çalışmaların bazıları Android kötü amaçlı yazılımlarının statik analizine, bazıları ise Android kötü amaçlı yazılımlarının dinamik analizine odaklanır. Çok çeşitli gelişmiş algılama teknikleri yalnızca statik analize, yani yazılımı çalıştırmaksızın kaynak kod üzerinden elde edilebilecek özelliklere dayanır. Ayrıca, kötü amaçlı yazılım saptama sorununa dinamik analiz başarıyla uygulanır. Son zamanlarda, hem statik hem de dinamik özelliklerin kullanıldığı melez yaklaşımlar analiz edilmiştir.

Her şeyden önce dinamik ve statik uygulamaların farklılığı anlaşılmalıdır. Statik uygulamalar bir uygulama çalıştırılmadan zararlı kod tespitini yapar. Bu yöntem iki açıdan iyidir:

Birincisi, kod hiç çalıştırılmadığından dolayı güvenilirlik sağlar. İkinci olarak ise bu yöntem fazla sistem kaynağı kullanmaz ve çalışma zamanı üzerinde bir etkisi olmaz. Ama şifreli ve gizli kodları tespit etmekte çok zorlanır.

Dinamik yöntemde ise bu sorunlardan kurtulunur; fakat bir uygulamayı doğru analiz edebilmek için yeterince süreyle çalıştırılmış olması gerekir ve bu süreden sonra kod zarar vermiş hale gelebilir. Bu yüzden de dinamik yöntem sanal bir ortamda kullanılmalıdır. Sanal ortam gerekliliği sebebiyle de bu yöntem kullanıcının herhangi bir kaynaktan uygulama yükleyebileceği gerçek dünya senaryolarında çok pratik değildir.

Biz, projemizde statik analiz yöntemini uygulamaktayız. Android uygulamalarının analizini gerçekleştirme işlemini kaynak kodu çıkartılmış apk'ların analizini gerçekleştirerek yapacağız. Bunu yaparken de kaynak kodunu word embedding yönteminin CBOW ve GloVe modellerini kullanıp bu kaynak kodlarının vektör sayısal değerlerini çıkartarak zararlı ve zararsız yazılımlar sınıflandırılıp zararlı yazılımlar ailelerine göre kategorize edilecektir.

Tespit etme aşamasında WEKA isimli açık kaynak kodlu makine öğrenme yazılımının

3.8.3 sürümü, default parametreler ile kullanılacaktır. Sınıflandırılmış ve ailelerine göre kategorize edilmiş zararlı ve zararsız yazılımlarımızın %70'ini eğitim için, kalan %30'unu da test için kullandık. Böylece bu yöntemlerin doğruluk payını ede ettik. Ayrıca bu projede MALWARE ve BENIGN olarak iki sınıf uygulama türü incelendi. MALWARE uygulamalar zararlı yazılımlar anlamına gelmekte ve BENIGN uygulamalar ise iyi huylu yani zararsız uygulamalar anlamına gelmektedir. İlerleyen sayfalarda zararlı yazılımlardan Malware, zararsız yazılımlardan ise Benign diye söz edeceğiz.

3

Fizibilite

Fizibilite çalışmasında proje uygulaması başlamadan önce gerekli hazırlıklar yapılmıştır.

Bu proje için gerekli Teknik, İşgücü ve Zaman, Yasal, Ekonomik Fizibiliteler çıkarılmıştır.

3.1 Teknik Fizibilite

Teknik fizibilite kısmında projenin çalıştırılabilmesi için gerekli yazılım ve donanım fizibiliteleri belirtilmiştir.

3.1.1 Yazılım Fizibilitesi

Yazılım fizibilite kısmında ortamın çalışıldığı işletim sistemi, gerekli framework ve simulasyon ortamı için gerekli programlar belirtilmiştir.

İşletim sistemi olarak Windows işletim sistemi kullanılıyor; fakat diğer işletim sistemleri de kullanılabilir. Çünkü işletim sisteminden bağımsız olarak projemizde Google Colab teknolojisinin Tesla K80 GPU makinesi ortamı kullanılıyor. Ayrıca programlama dili olarak Python dili tercih edilmiştir. Daha sonra makine öğrenmesi için Weka isimli açık kaynak kodlu yazılım kullanılıyor. Tablo 3.1’de görebilirsiniz.

Tablo 3.1 Yazılım Fizibilite Tablosu

Özellik	Ortam
İşletim Sistemi	Bağımsız
Teknoloji Ortamı	Tesla K80 GPU (Google Colab)
Programlama Dilleri	Python ve C++
Makine Öğrenmesi Editörü	Weka 3.8.3

3.1.2 Donanım Fizibilite

Projenin geliştirileceği bilgisayarın üzerindeki donanım gereksinimleri Tablo 3.2’de verilmiştir.

Tablo 3.2 Donanım Fizibilite Tablosu

Özellik	Donanım Gereksinimleri
Disk Boyutu	64GB Hard Disk
RAM	minimum 4GB, tavsiye edilen 8GB
CPU	minimum Intel i5 veya eşdeğer
GPU	CUDA teknolojisini destekleyen Nvidia ekran kartı

3.2 İletişim Fizibilitesi

Ekibin proje üzerinde eşzamanlı olarak çalışmasına olanak sağlamak için, Google Drive bulut teknolojisinin dosya paylaşım özelliği kullanılması öngörülmektedir.

3.3 Yasal Fizibilite

TensorFlow, Apache open-source License 2.0 lisans sürümüne sahiptir.

WEKA, GNU General Public license (Weka 3.6 için GPL 2.0) ve (Weka > 3.7.5 için GPL 3.0) lisansına sahiptir. Bu proje mevcut kanun ve yönetmeliklere uygun olup herhangi bir patent vb. korunmuş hakkı ihlal etmemektedir.

3.4 Ekonomik Fizibilite

Açık kaynak kodlu lisanslamanın olduğu yazılımlar, geliştirme sürecinde laboratuvar bilgisayarları ve ücretsiz kullanıma sunulan Google Colab teknolojisi kullanılmıştır. Gereken maliyet fizibilitesi ise Tablo 3.3’de verilmiştir.

Tablo 3.3 Maliyet Analizi

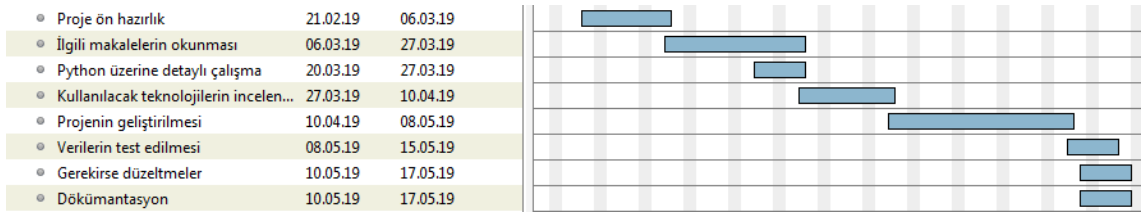
Özellik	Gereksinim Maliyeti(₺)
Donanım (CPU, RAM, Disk...)	3.000,00 ₺
Proje Ekibi	18.000,00 ₺
Toplam	21.000,00 ₺

3.5 İşgücü ve Zaman Fizibilitesi

Proje 2 kişilik ekibin bir dönemlik çalışması ve işgücü sonucunda üretilmiştir.

GANTT project		
Taslak Adı	Başlangıç	Bitiş
• Proje ön hazırlık	21.02.19	06.03.19
• İlgili makalelerin okunması	06.03.19	27.03.19
• Python üzerine detaylı çalışma	20.03.19	27.03.19
• Kullanılacak teknolojilerin incelenmesi	27.03.19	10.04.19
• Projenin geliştirilmesi	10.04.19	08.05.19
• Verilerin test edilmesi	08.05.19	15.05.19
• Gerekirse düzeltmeler	10.05.19	19.05.19
• Dökümantasyon	10.05.19	19.05.19

Şekil 3.1 Gantt Diyagramı Çizelgesi



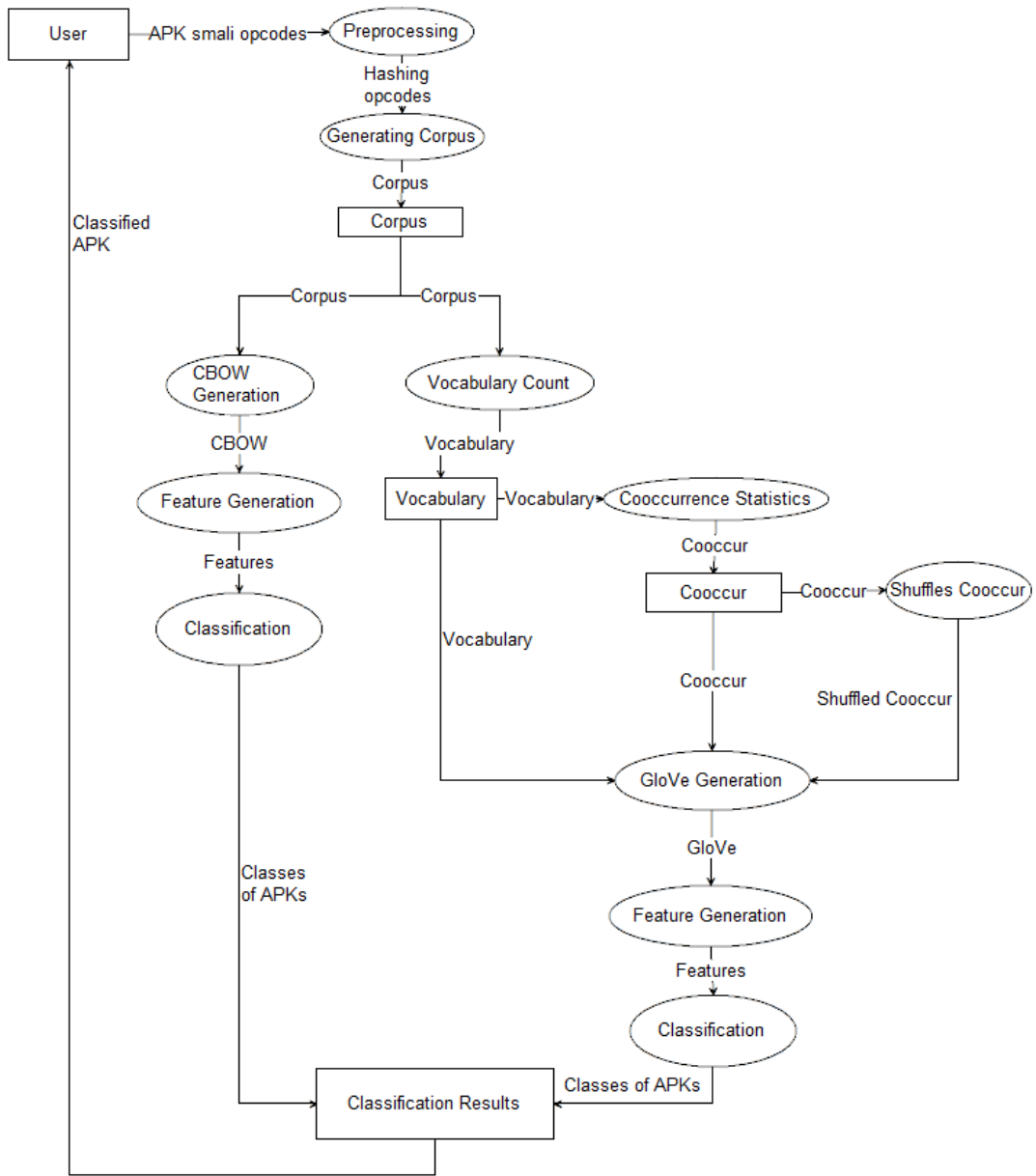
Şekil 3.2 Gantt Diyagramı Zaman Grafiği

4 Sistem Analizi

Verimli, sağlam ve ölçeklenebilir bir kötü amaçlı yazılım tanıma modülü, her siber güvenlik ürününün temel bileşenidir. Kötü amaçlı yazılım tanıma modülleri, bir nesnenin üzerinde topladıkları verilere dayanarak bir tehdit olup olmadığına karar verir. Bu veriler farklı aşamalarda toplanabilir. Kötü amaçlı yazılım, bunları gerçekleştirerek veya yürütme işlemi olmadan yani çalıştırılmadan da algılanabilir. Bu yöntemler Dinamik Analiz ve Statik Analiz olarak adlandırılır. Dinamik analiz, fiziksel veya sanal bir makinenin çalıştırılabilir yürütme davranışına odaklanmıştır. Statik analiz ise herhangi bir ortamda zararlı yazılımlar çalıştırmadan kaynak kodu analizine dayanır. Yürütme öncesi aşama verileri, yürütmeden dosya hakkında söyleyebileceğiniz herhangi bir şeydir. Bu, çalıştırılabilir dosya formatı açıklamalarını, kod açıklamalarını, ikili veri istatistiklerini, metin dizelerini ve kod emülasyonu ve benzeri diğer verilerle elde edilen bilgileri içerebilir.

Modern dünyada tehditler ve kötü niyetli olaylar önemli ölçüde arttı. Bu yüzden siber güvenlik çözümleri, yeni nesil algoritmalar kullanmalı ve kötü amaçlı yazılım algılama etki alanına yaklaşmalıdır. Ayrıca bu tespit edilen kötü amaçlı yazılımları saptama sorunu bir sınıflandırma sorunu olarak da adlandırılabilir. Bu yüzden makine öğrenmesi üzerine sınıflandırma işlemleri, veri seti yapısına göre iki ana bölüme ayrılabilir. Bizim veri setimiz Malware veya Benign olan sınıf etiketlerine sahiptir, ayrıca Malware sınıf etiketli veriler ailelerine göre de dosya isimleri, aile isimleriyle başlayacak şekilde sınıflandırılmıştır. Bu nedenle denetimli sınıflandırma bu soruna uygulanabilir diyebiliriz.

Sonuç olarak sistem analizinde tanımlanmış olan akış diyagramı aşağıdaki Şekil 4.1’de görüldüğü gibidir.



Şekil 4.1 Sistemin Akış Diyagramı

5

Sistem Tasarımı

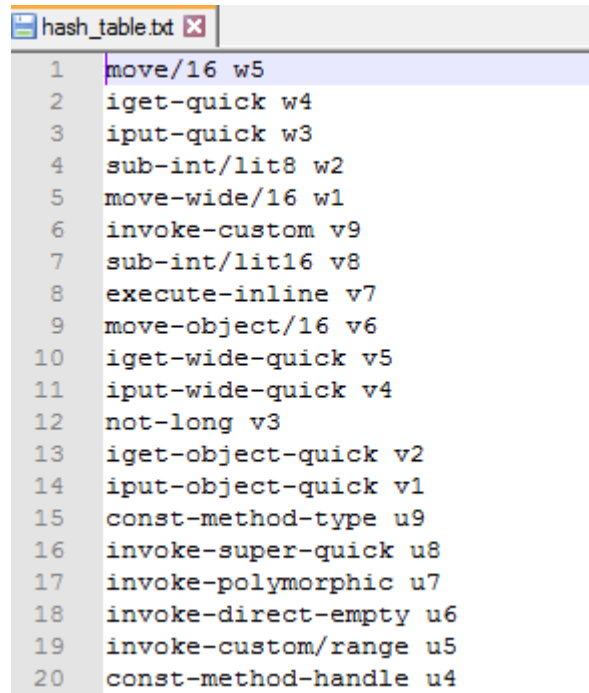
Bu bölümde zararlı yazılımların tespitinin sistem tasarımı açıklanmıştır.

5.1 Uygulama Tasarımı

Bu bölümde, projenin yazılım tasarımı incelenmiştir.

5.1.1 Ön İşleme

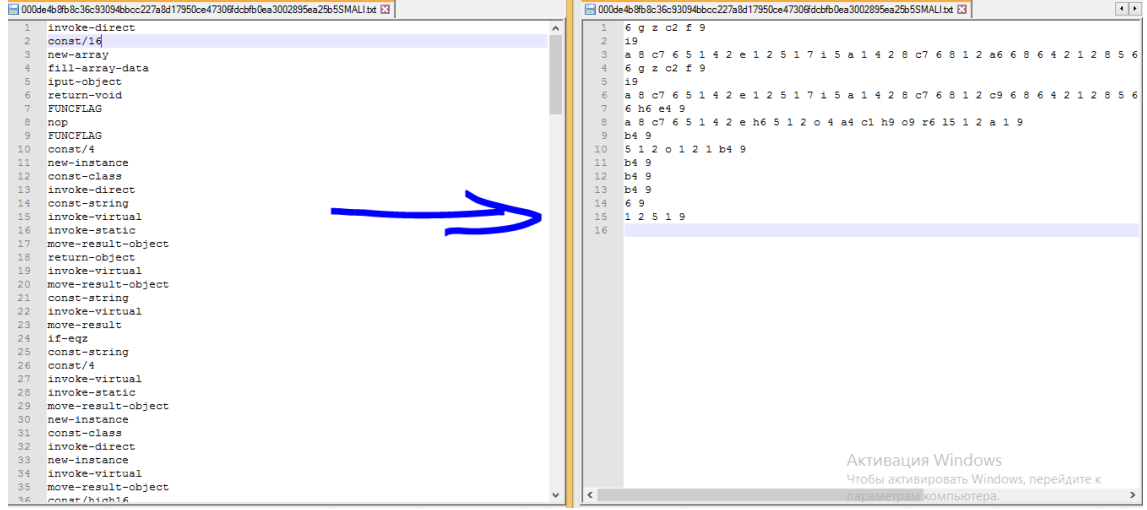
Bu projede ön işleme tasarımı projenin gerçekleşmesinde ilk adımdır. Bu adımda, veri setindeki bütün yazılımların içindeki benzersiz opcode'lar(238 adet) kullanılarak bir hash tablosuna atılır. Bu hash tablosunun yazıldığı dosya Şekil 5.1'deki gibidir. Bu dosyada tüm benzersiz opcode'lar ve yanlarında da hash değerleri yazılmıştır.



1	move/16 w5
2	iget-quick w4
3	iput-quick w3
4	sub-int/lit8 w2
5	move-wide/16 w1
6	invoke-custom v9
7	sub-int/lit16 v8
8	execute-inline v7
9	move-object/16 v6
10	iget-wide-quick v5
11	iput-wide-quick v4
12	not-long v3
13	iget-object-quick v2
14	iput-object-quick v1
15	const-method-type u9
16	invoke-super-quick u8
17	invoke-polymorphic u7
18	invoke-direct-empty u6
19	invoke-custom/range u5
20	const-method-handle u4

Şekil 5.1 Hash Tablosu

Daha sonra opcode dosyalarının içerisindeki opcode'lar, bu hash tablosundaki karşılıkları kullanılarak yeniden üretilir. Böylece ön işleme adımında dosya boyutlarında büyük miktarda azalma sağlanmış olup verinin işlenmesi hızlandırılmış olur. Örnek bir opcode dosyasının dönüşümünü Şekil 5.2'de görebilirsiniz. Daha sonra bütün hashlenmiş opcode dosyalarının içerikleri tek bir corpus dosyası oluşturularak içine atılır. Bu projedeki tüm opcode dosyaları .txt formatındadır.



Şekil 5.2 Opcode to Hashing Opcode Dönüşümü Örneği

5.1.2 CBOW Uygulanması

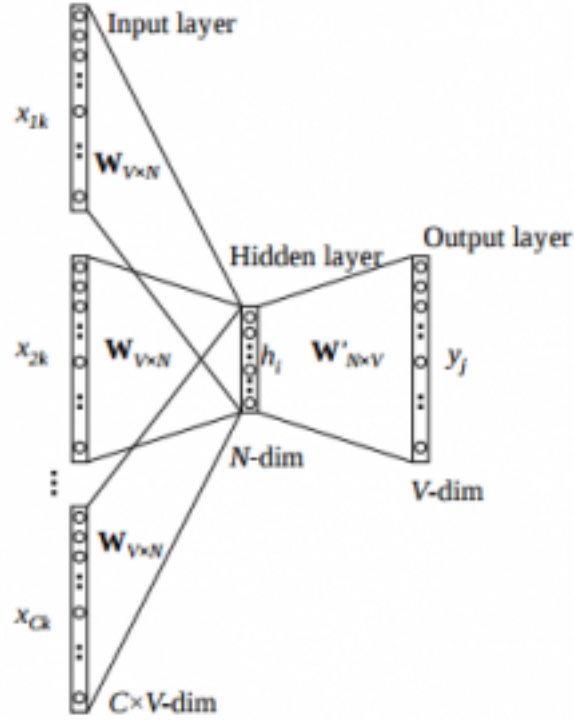
Word2vec, ham metinden kelimelerin vektörlerini elde etmek için bir yöntemdir. Word2vec yöntemi iki modelden oluşur: Skip-Gram ve CBOW (Sürekli Sözcük Torbası). Bu projede, CBOW modeli kullanılmıştır.

CBOW modelinde window size'ın merkezinde olmayan kelimeler input olarak alınıp , merkezinde olan kelimeler output olarak tahmin edilmeye çalışılır[5]. Pencere boyutu 1 olan CBOW örneği Şekil 5.3'te gösterilmiştir. Şekle göre ortadaki kelime output'u, kenardaki kelime(ler) input'u temsil ediyor.

CBOW WORD2VEC WITH WINDOW_SIZE=1							Training Samples	
							Input	Output
1)	bir	yandan	da	hiç	konuşmak	istemiyor	yandan	bir
2)	bir	yandan	da	hiç	konuşmak	istemiyor	bir	yandan
							da	yandan
3)	bir	yandan	da	hiç	konuşmak	istemiyor	yandan	da
							hiç	da

Şekil 5.3 CBOW Word2Vec Modeli Örneği

Şekil 5.3'teki örnek kullanılarak bir sinir ağı eğitilebilir. Bu sinir ağı mimarisi Şekil 5.4'te gösterildiği gibidir.



Şekil 5.4 CBOW modelin Neural Network'te gösterimi [5]

5.1.3 GloVe Uygulanması

Global Vektörlerden elde edilen GloVe , dağıtık kelime sunumu için bir modeldir. Kelimeler için vektör gösterimleri elde etmek için denetimsiz bir öğrenme algoritması olan bir modeldir. Bu, kelimeler arasındaki mesafenin anlamsal benzerlik ile ilişkili olduğu anlamlı bir alana kelimeler eşleştirilerek elde edilir[6]. Eğitim, bir korpustan toplanmış global kelime-kelime eş-oluşum istatistikleri üzerinde gerçekleştirilir ve elde edilen sunumlar, kelime vektör uzayının ilgili doğrusal alt yapılarını gösterir. Stanford'da açık kaynaklı bir proje olarak geliştirilmiştir. Sözcük temsillerinin denetimsiz bir şekilde öğrenilmesi için log-bilinear regresyon modeli olarak, iki model ailenin özelliklerini, yani global matris çarpanlarını gizli anlamsal analiz (LSA)[7] ve Skipgram modeli gibi yerel bağlam penceresi yöntemleri Mikolov ve diğ.[8] yöntemlerini birleştirir.

Window size 2 ve embedding size 300 verilen Random Forest sınıflandırma algoritmasına göre örnek CBOW ve GloVe başarımları aşağıdaki Şekil 5.5 ve Şekil 5.6'da verilmiştir.

```

=== Summary ===

Correctly Classified Instances      11726      94.1923 %
Incorrectly Classified Instances    723        5.8077 %
Kappa statistic                    0.8805
Mean absolute error                0.1203
Root mean squared error            0.2187
Relative absolute error             24.7905 %
Root relative squared error        44.3966 %
Total Number of Instances         12449

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          0,936   0,054   0,925     0,936   0,930     0,881   0,986   0,979   Benign
          0,946   0,064   0,954     0,946   0,950     0,881   0,986   0,991   Malware
Weighted Avg.   0,942   0,060   0,942     0,942   0,942     0,881   0,986   0,986

=== Confusion Matrix ===

      a    b  <-- classified as
4825  332 |    a = Benign
391  6901 |    b = Malware

```

Şekil 5.5 Random Forest Algoritmasına Göre CBOW Sonuçları

```

=== Summary ===

Correctly Classified Instances      11792      94.7225 %
Incorrectly Classified Instances    657        5.2775 %
Kappa statistic                    0.8916
Mean absolute error                0.1206
Root mean squared error            0.214
Relative absolute error             24.8592 %
Root relative squared error        43.4535 %
Total Number of Instances         12449

=== Detailed Accuracy By Class ===

          TP Rate  FP Rate  Precision  Recall  F-Measure  MCC      ROC Area  PRC Area  Class
          0,947   0,053   0,927     0,947   0,937     0,892   0,988   0,982   Benign
          0,947   0,053   0,962     0,947   0,955     0,892   0,988   0,992   Malware
Weighted Avg.   0,947   0,053   0,948     0,947   0,947     0,892   0,988   0,988

=== Confusion Matrix ===

      a    b  <-- classified as
4885  272 |    a = Benign
385  6907 |    b = Malware

```

Şekil 5.6 Random Forest Algoritmasına Göre GloVe Sonuçları

5.2 Veri Seti Tasarımı

Bu projede 41484 tane android uygulamasına ait smali opcode kullanılmakta olup verisetimizin %70'i eğitim için, %30'u ise test için ayrılmıştır. Benign uygulamalar Comodo güvenlik firması tarafından sağlanmıştır. Malware için ise AMD MALWARE Veri Seti kullanılmıştır. AMD MALWARE Veri Seti, 2010 ile 2016 arasında 71 Malware ailesinden oluşan 24304 örnek içermektedir. İlgili veri seti aşağıdaki Tablo 5.1'de gösterilmiştir.

5.3 Girdi - Çıktı Tasarımı

Sisteme bir smali opcode dosyası girdi olarak verilebilir. Malware algılama sistemi ilk önce verilen girişi ön işleme tabi tutar ve CBOW ve GloVe üretir. Ardından her bir apk

Tablo 5.1 Veri Seti Yapısı

Sınıf	Train(70%)	Test(30%)	Toplam(100%)
BENIGN Comodo Veri Seti	12.026	5.154	17.180
AMD MALWARE Veri Seti (71 Malware ailesi içerir.)	17.012	7.292	24.304
Genel Toplam Adeti	29.038	12.446	41.484

için özellik vektörleri üretilir ve .arff dosyasına yazılır. Daha sonra Weka programı ile .arff dosyası işlenerek sınıflandırılmış sonuçlar üretilir.

6 Uygulama

Bu bir araştırma projesi olduğundan, uygulama Grafikselle Kullanıcı Arayüzüne sahip tek bir çalıştırılabilir dosya yerine birkaç tane Python ve C++ komut dosyasından oluşur. İlk olarak tüm yazılım dosyalarının içeriklerindeki opcode'ların benzersiz olanlarının toplandığı bir dosya içerisine hem bu benzersiz opcode'lar hem de hash tablosundaki değerleri yazılarak tutulur. Daha sonra oluşturulan tüm yazılım dosyalarının opcode'ları hash tablosu dosyasındaki karşılıklarıyla değiştirilerek yeniden yazılır. Böylece veri küçültme sağlanarak veri işlemede performans da artırılmış olur. Şekil 6.1'da görüldüğü üzere preprocessing adımı verilmiştir.



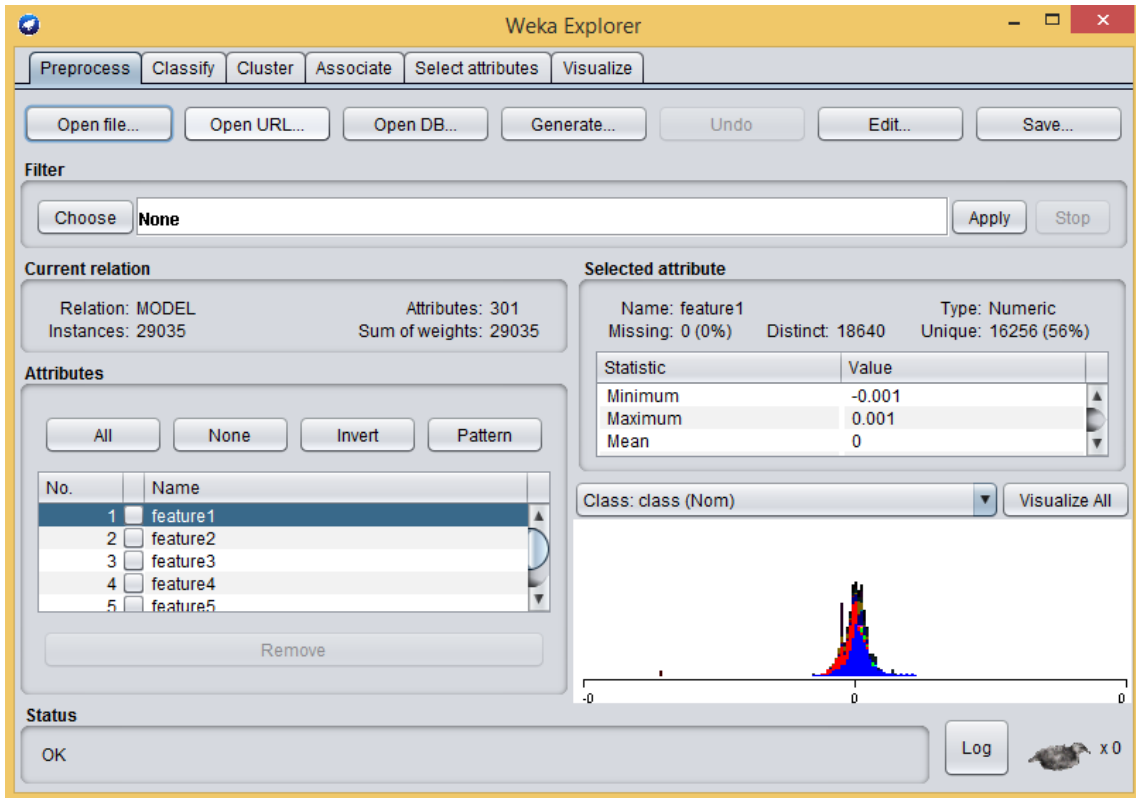
Şekil 6.1 Preprocessing

Sıradaki adımda her bir opcode dosyasının CBOW ve GloVe yöntemleriyle opcode için vektörleri oluşturulur. Word2Vec yöntemine göre vektörleri oluşturma işlemi de Şekil 6.2'e görüldüğü gibidir.

```
demo@demo-pc: ~/Desktop/Word2Vec/Word2Vec/bin
File Edit View Search Terminal Help
demo@demo-pc:~/Desktop/Word2Vec/Word2Vec/bin$ ./word2vec -train corpus.txt -output output.txt -size 500 -window 3 -sample 1e-4 -hs 0 -binary 0 -cbow 1 -iter 3
Starting training using file corpus.txt
Vocab size: 219
Words in train file: 3567871473
Alpha: 0.049990 Progress: 0.02% Words/thread/sec: 540.17k
```

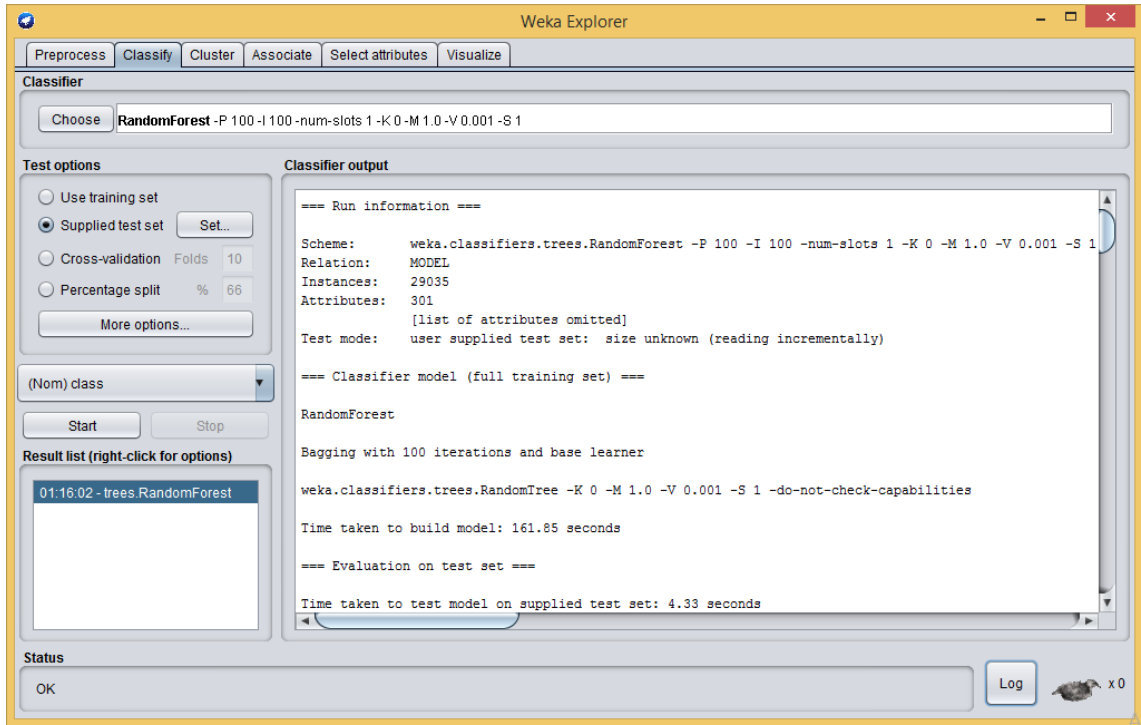
Şekil 6.2 Word2Vec ile Vektör Elde Etme

Daha sonra her apk dosyasının vektör ortalaması alınarak .arff dosyasına yazılır. Sonuçta elimizde %70'i eğitim ve %30'u test için rastgele gruplandırılmış olan her bir Malware ve Benign apk'sinin ortalama vektörlerinin ve ailesinin tutulduğu test.arff ve train.arff dosyaları bulunur. Weka'ya train.arff dosyasının yüklenme adımı Şekil 6.3'deki gibidir.



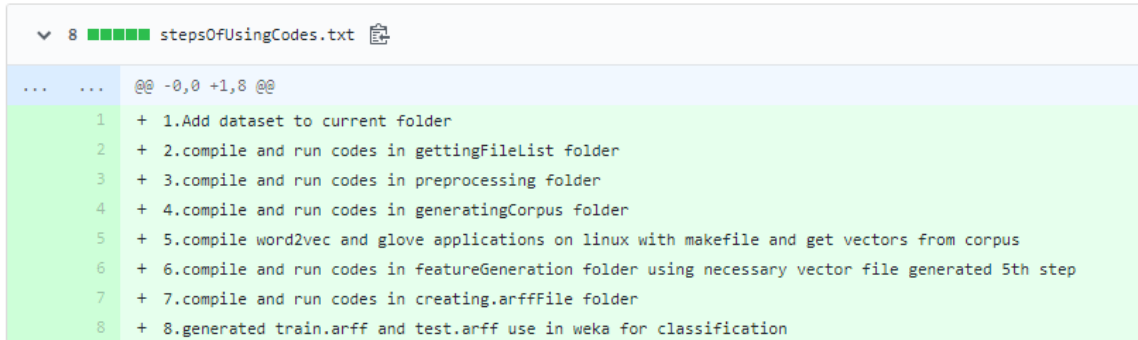
Şekil 6.3 Weka Loading Train File

Daha sonra bu vektörler, uygulamaları kategorilendirmek için sınıflandırıcılara verilir. Bu işlem adımının yapıldığı Weka'da .arff dosyalarındaki ortalama vektörler, farklı sınıflandırma algoritmaları kullanılarak ailelerine göre kategorilendirilir. Weka ile test.arff dosyasının işlenip sınıflandırılması adımı da Şekil 6.4'te gösterilmiştir.



Şekil 6.4 Weka Classifying From Test File

Ayrıca projemizin gerçekleştirildikten sonraki GitHub'a eklenmiş repository'deki how to use adımı da Şekil 6.5'teki gibidir[9].



Şekil 6.5 Step of Using Codes from GitHub

7 Deneysel Sonuçlar

Bu bölümde, önerilen sistemin sonuçları verilmiştir. CBOW ve GloVe yöntemlerinde pencere boyutu 3 ve vektör boyutu 500 veya pencere boyutu 2 ve vektör boyutu 300 olarak alınmıştır. Çeşitli makine öğrenme yöntemlerinin başarı oranları Tablo 7.1’de gösterilmiştir. Sınıflandırıcılar için "Weka" adlı popüler aracın 3.8.3 sürümü default parametreleriyle kullanılmıştır. Sonuçların tümü test seti kullanılarak üretilmiştir.

Tablo 7.1 Sınıflandırılmış Sonuçlar

Method	Window Size	Embedding Size	Classifier	Accuracy
CBOW	3	500	Random Forest	91.32%
GloVe				92.12%
CBOW	2	300		91.37%
GloVe				91.81%
CBOW	3	500	J48	85.66%
GloVe				47.37%
CBOW	2	300		84.72%
GloVe				49.67%
CBOW	3	500	IBk	87.92%
GloVe				89.73%
CBOW	2	300		87.85%
GloVe				89.65%
CBOW	3	500	SMO	70.41%
GloVe				78.39%
CBOW	2	300		67.46%
GloVe				78.39%
CBOW	3	500	Random Sub Space	88.75%
GloVe				89.95%
CBOW	2	300		88.69%
GloVe				89.70%
CBOW	3	500	Average Accuracy	84.81%
GloVe				79.51%
CBOW	2	300		84.02%
GloVe				79.84%

8 Sonuç

Bu arařtırmada Android Dalvik Opcodes[3], "baksmali" adı verilen ters mühendislik teknięi kullanılarak elde edilen 17180 Benign ve 24304 Malware Android uygulaması incelemeřtir. Bu kodları kullanarak uygulamaları sınıflandırmak için iki farklı yöntem uygulanmıřtır: CBOW ve GloVe yöntemleri. GloVe yönteminde her bir opcode için tek pencere boyutuna göre deęil; opcode'ın global kullanımıda kontrol edildięi ve vektörleri onlara göre ürettięi için özellikle Random Forest[10] sınıflandırma algoritmasına göre, daha iyi sonuç vermektedir. CBOW yöntemi tek pencere boyutuna göre vektörleri ürettięi için GloVe yöntemine göre sınıflandırma fonksiyonlarının çoęunda az farkla kötü sonuç üretmektedir. Sonuç olarak hedef uygulamayı gerçeklemek ve sistem istikrarını riske atmak zorunda kalmadan GloVe yöntemi %92.12'lik ve CBOW yöntemi ise %91.32'lik bir başarı elde etmiřtir.

- [1] the SEP Mobile Research Team. (2018). Mobile threat intelligence report 2017: The year in review, [Online]. Available: www.symantec.com/content/dam/symantec/docs/reports/mobile-threat-intelligence-report-2017-en.pdf.
- [2] R. H. Alireza Souri. (2018). Human-centric computing and information sciences, [Online]. Available: link.springer.com/article/10.1186/s13673-018-0125-x.
- [3] B. Gruver. (2009). Android dalvik opcodes, [Online]. Available: github.com/JesusFreke/smali.
- [4] O. Dayıbaşı. (2018). Word embeddings glove, [Online]. Available: medium.com/algorithms-data-structures/word-embeddings-943bf87dcc67.
- [5] M. Buyukkinaci. (2018). Word2vec nedir ? [Online]. Available: <https://medium.com/@mubuyuk51/word2vec-nedir-t%C3%BCrk%C3%A7e-f0cfab20d3ae>.
- [6] VariousUsers. (2019). Glove (machine learning), [Online]. Available: [en.wikipedia.org/wiki/GloVe_\(machine_learning\)](https://en.wikipedia.org/wiki/GloVe_(machine_learning)).
- [7] S. Deerwester, S. T. Dumais, G. Furnas, T. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American Society for Information Science*, vol. 41, pp. 391–407, Sep. 1990. DOI: 10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.
- [8] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, vol. 26, Oct. 2013.
- [9] S. bigalex95. (2019). Detect android malwares from their opcodes with word embedding and machine learning, [Online]. Available: <https://github.com/bigalex95/androidMalwareDetectionSystem>.
- [10] A. Liaw and M. Wiener, "Classification and regression by randomforest," *Forest*, vol. 23, Nov. 2001.

1. ÜYENİN KİŞİSEL BİLGİLERİ

İsim-Soyisim: Alibek ERKABAYEV

Doğum Tarihi ve Yeri: 06.03.1995, Daşoğuz

E-mail: alibek060395@gmail.com

Telefon: 0554 510 54 15

Staj Tecrübeleri: Araştırmacı stajyer - YTU Donanım anabilim dalı

2. ÜYENİN KİŞİSEL BİLGİLERİ

İsim-Soyisim: Fatih ÇOMAK

Doğum Tarihi ve Yeri: 25.09.1993, Antalya

E-mail: fatihchomak@gmail.com

Telefon: 0534 310 78 68

Staj Tecrübeleri: Garanti Teknoloji Genel Staj, etcBase Mesleki Staj

Proje Sistem Bilgileri

Sistem ve Yazılım: Windows veya Linux İşletim Sistemi, Python, C++, Google Colab

Gerekli RAM: 12GB

Gerekli Disk: 100GB