The Korean Journal of Applied Statistics (2017) DOI: https://doi.org/10.5351/KJAS.2017.30.6.851
30(6), 851–865

Malware classification using statistical techniques

Sungmin Won[a] · Hyunjoo Kim[a] · Jongwoo Song[a,1]

[a]Department of Statistics, Ewha Womans University

(Received August 14, 2017; Revised September 29, 2017; Accepted October 12, 2017)

Abstract

Ransomware such as WannaCry is a global issue and methods to defend against malware attacks are important. We have to be able to classify the malware types efficiently in order to minimize the damage from malwares. This study makes models to classify malware properly with various statistical techniques. Several classification techniques such as logistic regression, random forest, gradient boosting, and support vector machine are used to construct models. This study also helps us understand key variables to classify the type of malicious software.

Keywords: malicious software, multi-class classification, random forest, gradient boosting, support vector machine, important variables

1. 서론

최근 워너크라이라는 이름의 랜섬웨어가 전 세계적으로 큰 이슈가 되었다. 랜섬웨어는 사용자의 컴퓨터 시스템에 대한 접근을 제한한 뒤에 금전적인 요구를 하는 악성 소프트웨어의 일종이다. 개인 뿐 아니라 기업 또한 피해 대상에 포함되었기 때문에 이로 인한 피해 금액이 상당한 것으로 알려져 있다. 한 인터넷 매체에 따르면 (데이터넷, http://www.datanet.co.kr/news/articleView.html?idxno=111694) 2016년 전 세계 랜섬웨어 피해 금액은 1조 200억 원이며, 랜섬웨어 공격 건수는 2015년 380만 건에서 2016년 6억 3,800만 건으로 167배 증가했다. 또한 국내 랜섬웨어 피해자는 2016년 13만 명에 이르며 총 3,255건, 3,000억 원의 피해가 발생했다. 오늘날 컴퓨터와 각종 IT 기기들의 발달로 상당히 많은 사람들이 이에 의존하게 되면서, 이를 악용하여 이득을 취하려고 하는 무리가 점점 늘어나는 추세에 있다. 다시 말해 정보가 곧 돈으로 직결되는 경우가 많기 때문에, 현대 사회는 결국 이를 보호하려는 무리와 공격하려는 무리로 양극화 되어가고 있다. 문제는 우리 사회의 많은 시스템이 컴퓨터 및 다른 기기에 의존하는 경향이 있어 이 장비들이 손상을 입을 경우 피해가 너무 막심하다는 점이다.

이러한 피해를 최소화하기 위해서는 무엇보다도 공격 형태를 파악하고 방어 체계를 만들어내는 것이 가장 중요하다. 또한 신속하게 백신을 만들어 보급하기 위해서는 공격하는 주체의 유형을 빠르게 판단하는 과정이 필수적이다. 그러나 악성 소프트웨어의 경우 쉽게 탐지되지 않도록 패킹과 다형화를 통하여 코드 해독을 어렵게 만들어낸 경우가 많다 (Kim 등, 2010). 또한 이렇게 난독화된 코드 파

This work was supported by the Ministry of Education of the Republic of Korea and the National Research Foundation of Korea (NRF-2017R1D1A1B03036078).
[1]Corresponding author: Department of Statistics, Ewha Womans University, 52, Ewhayeodae-gil, Seodaemun-gu, Seoul 03760, Korea. E-mail: josong@ewha.ac.kr

일은 대부분 매우 큰 용량을 가지며, 매일 발생하는 새로운 악성코드의 개체수 역시 상당하다. AV test(https://www.av-test.org/en/statis tics/malware)에 따르면 2013년도부터 새로운 악성 소프트웨어 발생이 본격적으로 활성화되었으며 2016년도에는 약 120,000,000개의 새로운 악성 소프트웨어가 발생하였으므로 해당 분야의 전문가가 모든 코드를 일일이 분류하는 것이 현실적으로 불가능한 상황이다.

본 논문에서는 다양한 데이터마이닝 기법을 이용하여 컴퓨터를 대상으로 하는 악성 소프트웨어들의 유형을 파악하고자 한다. 기존에는 은닉 신경망을 이용하거나 (Dahl 등, 2013), 서포트 벡터 기계 (Chen과 Aritsugi, 2006) 또는 로지스틱을 이용한 모형들을 (Konrad, 2008) 통해 악성 소프트웨어 분류에 대한 통계적 모형의 가능성들을 보여왔다. 또한 현재까지도 정확도를 높이기 위한 여러 방법들을 모색 중이다. 우리는 주어진 자료를 이용하여 분류에 가장 효과적인 모형을 비교해보고 최적의 모형을 제안하고자 한다.

본 연구에서 진행되는 모든 분석은 R(R project ver. 3.3.4)을 통해 이루어졌으며 모형 구축을 위해서는 R 패키지를 이용하였다. 분석에 사용된 기법들은 랜덤 포레스트 (Breiman, 2001), 그래디언트 부스팅 (Friedman, 2002; Ridgeway, 2007), 서포트 벡터 기계 (Cortes와 Vapnik, 1995), 의사결정나무 (Brieman 등, 1984)이며 예측력을 판단하기 위하여 오차 행렬(confusion matrix)을 이용하여 최종 모형을 비교하였다. 또한 각 모형에서 사용되는 적절한 모수를 정하기 위해서는 교차 타당화 오류(cross validation error) 값을 이용하여 가장 적은 오차를 발생시키는 모수를 지정하였다.

본 논문은 다음과 같은 순서로 구성된다. 2장에서는 자료에 대한 설명과 데이터 분석을 시작하기에 앞서 진행했던 전처리 과정에 대해서 설명하며, 3장에서는 상기된 방법들을 이용하여 구축한 모형에 대한 실질적인 비교 과정이 진행되고 마지막 4장에서 결론과 본 모형의 한계점에 대해서 짚어보고 개선될 점을 제안할 것이다.

2. 분석자료 설명

2.1. 자료수집 과정

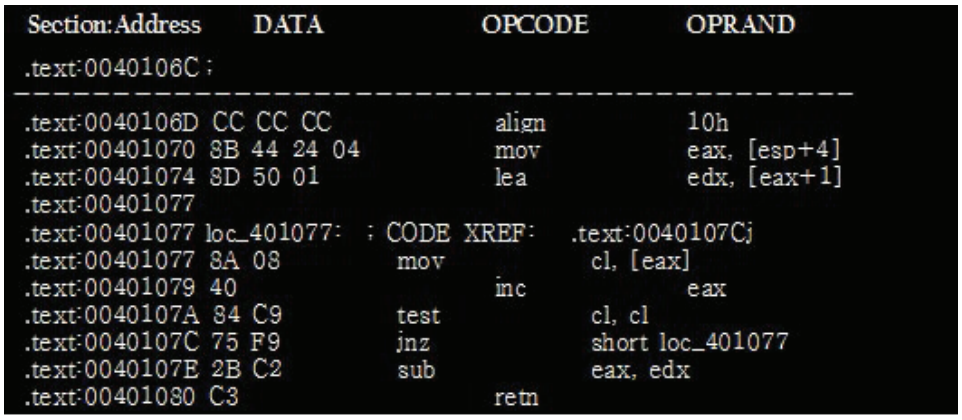
본 논문은 컴파일 된 악성 소프트웨어 여덟 종류의 파일 10,761개를 대상으로 한다. 이 파일은 악성 소프트웨어에 대한 파악 시간을 줄여서 백신을 비롯한 방어 기제를 좀 더 빠르게 구축하기 위해, 2015년도에 마이크로소프트사에서 캐글(www.kaggle.com)에 제공했던 데이터이다. 처음에는 총 10,868개의 아홉 가지 종류의 소프트웨어 파일을 수집했으며, 이 중에서 전체가 결측치로 이루어지거나 관측치가 적어 모형을 구축하기 어려운 한 유형의 반응 변수를 제거한 뒤에 10,761개의 여덟 가지 악성 소프트웨어 파일을 대상으로 연구를 진행하였다. 결측치 제거와 반응 변수 선별 등에 대한 자료의 가공 과정은 2.4절 데이터 전처리 과정에 기재되어있다.

2.2. 자료의 구성

데이터는 한 관측치 당 두 개의 파일로 구성되어 있다. 첫 번째 파일은 프로그램의 소스 코드를 기계어로 변환하여 16진수로 구성된 HEX 파일로 출력한 bytes 파일이며 (Figure 2.1), 두 번째 파일은 어셈블리어로 구성된 확장자 .asm을 갖는 파일이다 (Figure 2.2). 이 두 가지 파일은 컴퓨터에서 프로그램이 시작된 주소와 무엇이 실행됐는지가 나열된 기록으로, 해당 악성 코드의 행동반경을 파악하는 데에 매우 유용하다. 이처럼 실행파일에 포함된 문자열을 추출하고 분석하는 것을 정적 분석이라 부르며, 악성코드를 실행하지 않은 상태에서 분석 가능하다는 안전성 때문에 악성코드의 구조와 특성을 파악하는 데에 자주 이용된다 (Choi 등, 2014).

Address	DATA
00401000	56 8D 44 24 08 50 8B F1 E8 1C 1B 00 00 C7 06 08
00401010	BB 42 00 8B C6 5E C2 04 00 CC CC CC CC CC CC CC
00401020	C7 01 08 BB 42 00 E9 26 1C 00 00 CC CC CC CC CC
00401030	56 8B F1 C7 06 08 BB 42 00 E8 13 1C 00 00 F6 44
00401040	24 08 01 74 09 56 E8 6C 1E 00 00 83 C4 04 8B C6
00401050	5E C2 04 00 CC CC CC CC CC CC CC CC CC CC CC CC
00401060	8B 44 24 08 8A 08 8B 54 24 04 88 0A C3 CC CC CC
00401070	8B 44 24 04 8D 50 01 8A 08 40 84 C9 75 F9 2B C2
00401080	C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
00401090	8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010A0	08 50 51 52 56 E8 18 1E 00 00 83 C4 10 8B C6 5E
004010B0	C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010C0	8B 44 24 10 8B 4C 24 0C 8B 54 24 08 56 8B 74 24
004010D0	08 50 51 52 56 E8 65 1E 00 00 83 C4 10 8B C6 5E
004010E0	C3 CC CC CC CC CC CC CC CC CC CC CC CC CC CC CC
004010F0	33 C0 C2 10 00 CC CC CC CC CC CC CC CC CC CC CC

Figure 2.1. A sample of bytes file.



Section:Address	DATA	OPCODE	OPRAND
.text:0040106C ;			
.text:0040106D	CC CC CC	align	10h
.text:00401070	8B 44 24 04	mov	eax, [esp+4]
.text:00401074	8D 50 01	lea	edx, [eax+1]
.text:00401077			
.text:00401077 loc_401077:		; CODE XREF:	.text:0040107Cj
.text:00401077	8A 08	mov	cl, [eax]
.text:00401079	40	inc	eax
.text:0040107A	84 C9	test	cl, cl
.text:0040107C	75 F9	jnz	short loc_401077
.text:0040107E	2B C2	sub	eax, edx
.text:00401080	C3	retn	

Figure 2.2. A sample of asm file.

Bytes 파일 먼저 살펴보면 크게 Address 부분과 DATA 부분으로 구성된 것을 확인할 수 있다. 이들이 각각 의미하는 바는 다음과 같다.

- Address: 파일이 실행(run)되는 주소를 의미한다.
- DATA: 16진수로 표현되는 실행 명령어로 각 address 별 16바이트의 열로 구성된다.

Asm 파일은 실행 파일(Portable Executable; PE)로부터 지원되는 파일 형식을 Interactive Disassembler (IDA) 툴을 이용하여 어셈블리어 코드를 형성한 파일로, 크게 다음과 같은 구성으로 이루어진 파일이다.

- Section: Table 2.1 참고.
- Address: 파일이 실행되는 주소를 의미한다.

Table 2.1. Structure of section

Section	.text:	프로그램 실행을 위한 코드
	.rdata:	읽기 전용 변수 (문자열, C++ 가상함수)
	.data:	초기화된 전역 변수 (읽기, 쓰기 가능)
	.edata:	Export할 AIP 정보 (DLL에서 주로 사용)
	.idata:	Import할 DLL과 그 API 정보
	.rsrc:	리소스 관련 데이터
	.reloc:	기본 재배치 정도 (DLL에서 주로 사용)

Table 2.2. Statistics of the number of rows of files

	Min	Q1	Median	Mean	Q3	Max
bytes file	2,560	15,104	75,840	81,002	146,432	963,584
asm file	294	5,287	56,769	307,596	200,804	3,850,497

- DATA: Byte 형식의 데이터로 16진수로 표현되어 있다.
- OPCODE: 지시를 내리는 함수로 소프트웨어의 행동반경을 결정하는 부분이다.
- OPRAND: 함수에서 지시한 내용에 대한 세부 사항이라고 볼 수 있다.

두 파일 모두 데이터 부분은 00에서 FF까지 16진수의 숫자로 구성된 바이트 형식을 갖추고 있는 것을 볼 수 있다. 또한 두 파일 모두에 실행이 시작되는 주소가 모두 기재되어 있으며 asm 파일에는 어떤 지시 함수를 통하여 어떤 행위를 시행하는지 OPCODE를 통해 살펴볼 수 있다. 즉 두 파일로부터 악성 소프트웨어가 반복적으로 시행하거나 다른 코드와는 다르게 행동하는 코드를 포착할 수 있다면 이들을 구분하는 것이 가능할 것이다. 상기된 두 파일을 통해 모형에 유효한 변수를 선정하는 방법은 2.5절에서 이어지는 변수 선정 과정에 기술되어 있다.

2.3. 데이터 특징

"악성 소프트웨어를 처방하기 위해 오늘날 마주하고 있는 가장 큰 문제는 방대한 양의 데이터와 파일입니다. (중략) 이는 유사한 공격 형태를 가지고 행동하는 악성 소프트웨어 군에 속하는 파일들이 마치 다른 군에 속하는 파일인 것처럼 지속적으로 변형되거나, 다양한 전략으로 분석가를 혼란시키기 때문입니다 (Kaggle, https://www.kaggle.com/c/malwar e-classification)." 이 인용문은 캐글에 실린 데이터 기술의 일부 발췌 내용으로 오늘날 악성 프로그램을 방어하기 위해 마주한 가장 어려운 점들에 대하여 설명되어 있다. 서론에서 언급했듯이 자료의 양이 상당하다는 점과, 난독화하기 위한 다양한 속임수들을 쓴다는 점이 오분류를 발생시키는 원인임을 예상할 수 있다.

Table 2.3은 해독해야 하는 파일 행수의 분포를 살펴본 자료이다. 분포의 중위수를 살펴보았을 때 bytes 파일은 75,840 줄을, asm 파일은 56,769 줄의 값을 갖는 것을 볼 수 있다. 다시 말해 약 1만 개 중에서 절반 이상의 자료들이 각각 75,000 줄과 56,000 줄이 넘는 코드를 가지고 있다. 또한 최댓값은 bytes 파일이 약 90만 줄을, asm 파일이 3백 8십만 줄을 이루는 것을 볼 수 있다.

2.4. 자료 전처리 과정

초기에 자료를 수집했을 때는 Table 2.4와 같은 아홉 가지의 악성 소프트웨어가 포함되어 있었다. Name은 각 악성 소프트웨어의 이름이며 Type은 해당 소프트웨어의 유형이다.

Table 2.3. Class of response variables

Name	Class	Type	Description
Ramnit	1	Worm	윈도우 사용자들을 겨냥한 컴퓨터 웜 프로그램으로 개인 정보 유출에 주로 이용
Lollipop	2	Trojan	컴퓨터에 광고성 창을 띄우거나 개인 컴퓨터에서 실행 중인 목록을 추격, 프로그램 다운로드 경로를 통해 감염
Kelihos_ver3	3	Botnet	명령 및 제어 서버에 의해 통제 당하는 대량 시스템으로 구성, 광고성 스팸이나 비트 코인 절도 목적
Vundo	4	Trojan	트로이 목마나 웜 형태. 광고성 팝업, 네트워크 수행 저하, 타 악성 소프트웨어 전송 수행
Simda	5	Backdoor	시스템 보안이 제거된 상태로 해커가 접속하기 쉽게 만든 비밀 통로, 개인 정보 및 컴퓨터를 통제
Tracur	6	Trojan	위장 프로그램을 이용하여 사용자가 설치하도록 유도. 개인 정보 유출, PC 성능 저하, 디도스 공격으로 활용
Kelihos_ver1	7	Botnet	Class 3과 유사
Obfuscator.ACY	8	Trojan	은닉 목적, 컴퓨터에서 감지 방지용 소프트웨어
Gatak	9	Trojan	개인 컴퓨터의 정보를 해커들에게 전달, 다른 악성 소프트웨어를 함께 다운로드. key generator나 software crack을 다운로드 할 때 주로 감염

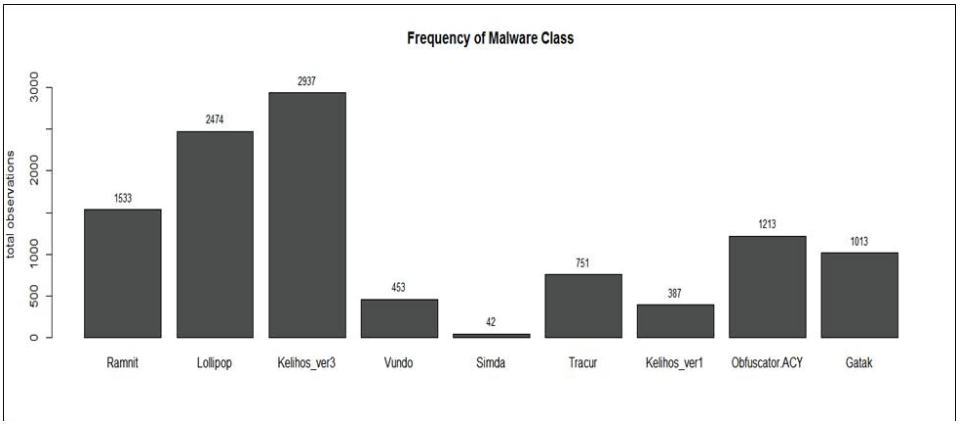


Figure 2.3. Barplot of malware class.

Figure 2.3은 처음에 수집했던 아홉 가지 반응 변수의 빈도수를 나타낸 그래프다. 다섯 번째 그룹에 해당하는 'Simda'라는 이름의 악성 소프트웨어가 42개의 자료를 가진 것을 볼 수 있다. 모형을 구현하기에는 관측치 수가 너무 빈약하기 때문에 'Simda'는 제외한 상태로 분석을 진행하였다. 이 외에도 파일 내에 모든 값이 '??'인 경우와 파일 형식이 다른 경우를 제거하였고 변수 변환 과정에서 오류가 발생하는 세 파일을 제거하였다.

2.5. 변수 선정 과정

2.5.1. N-gram 기법 N-gram이란 확률적 언어 모형의 한 방법으로 문자열을 특정 길이 n으로 나누는 기법이다. 예를 들어 'DATA'라는 단어를 2-gram으로 표현한다면 각각 DA, AT, TA, 총 세 부분으로 나누어진다. 즉 출현 가능한 문자열의 빈도수를 이용하여 다음에 등장할 문자열을 예측하는 확

Table 2.4. Pre-treatment of data sets

	제외한 자료 및 이유	♯ of obs
Simda (Class 5)	데이터 수가 작아 모형을 적합하기 어려운 경우	42
bytes file	파일 내 모든 값이 '??'인 경우	8
asm file	Format이 PE(Portable Executable)가 아닌 경우 (예: Format이 binary file 또는 알 수 없는 경우)	54
other	데이터 변환 작업 시 오류가 나는 경우	3
Total		107

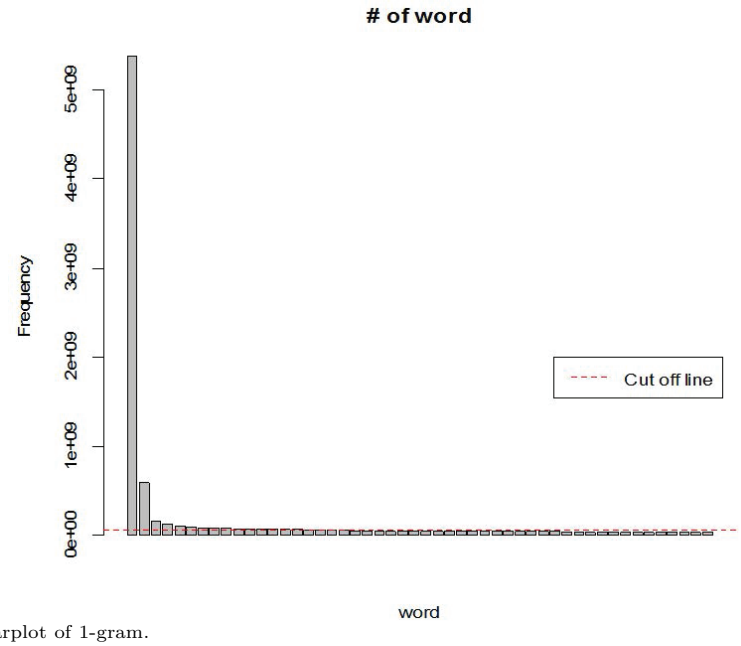


Figure 2.4. Barplot of 1-gram.

률 모형으로 줄글 형태의 데이터를 분석할 때 주로 사용된다. 이미 악성 소프트웨어를 탐지하기 위해 n-gram 기법을 이용하는 방법들이 (Santos 등, 2009) 제안되어 왔으며 최근까지도 상당히 많이 이용되고 있는 기법이다. Bytes 파일의 데이터는 16진수$(00, 01, \ldots, 99, \mathrm{AA}, \ldots, \mathrm{FF})$로 표현되는 명령어들을 포함하고 있다. 여기에서 1byte(00∼FF)가 한 단위이기 때문에 1-gram을 1byte로 간주하여 사용한다. 따라서 1-gram을 기준으로 자료를 나눈다면 00부터 FF까지 총 256가지의 명령어가 나타날 수 있다. 또한 bytes 파일을 확인해보았을 때 실행 명령어들을 16진수로 불러오는 과정에서 오류가 발생할 때 등장하는 '??'가 존재했기 때문에 이를 포함하면 총 257가지 1-gram 값들이 가능하다. 이 257개의 값들이 각 파일 내에서 차지하는 비율을 계산한 뒤에, 그 중에서 1차로 유의한 변수를 선택하기 위해 분산 분석(ANOVA)을 실시하였다 ($H_0 : \mu_1 = \cdots = \mu_4 = \mu_6 = \cdots = \mu_9$). 그 결과 총 164개의 단어가 유의수준 0.05에서 그룹별 차이에 영향력을 가진다고 판단할 수 있었다.

우리는 다시 2차적으로 변수를 선택하기 위해서 출현 빈도수를 이용하였다. Bytes 파일에서 자주 등장하는 순으로 변수를 나열했을 때 총 15개의 변수 이후로 등장 횟수가 급격하게 감소하는 것을 볼 수 있었다. 또한 자주 나타나는 상위 15개의 변수 모두 분산 분석에서 매우 유의하다고 나타나는 변수였으므로 우리는 두 과정에서 모두 선택된 변수 15개를 최종 변수로 지정하였다 (cut off: 60,000,000).

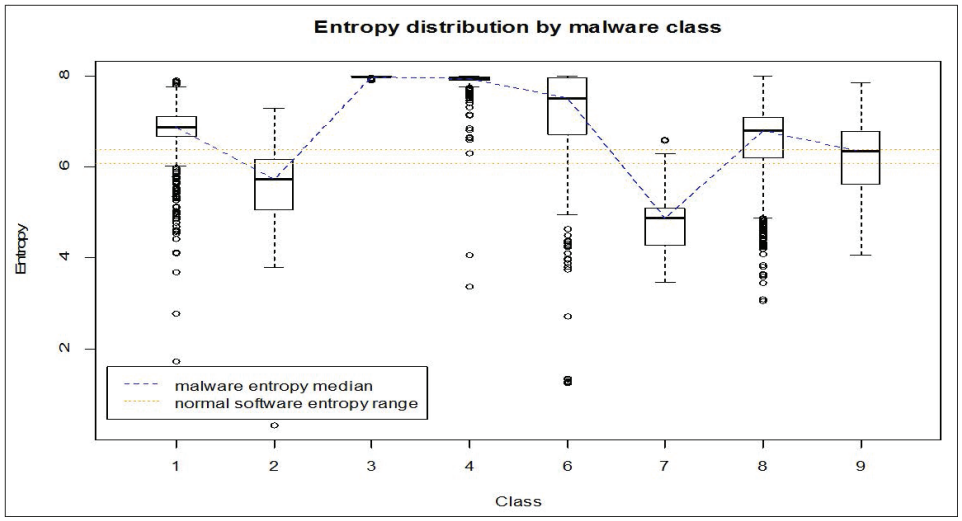


Figure 2.5. Boxplot of entropy value.

2.5.2. 엔트로피(Entropy) 엔트로피란 정보의 양을 나타내는 척도로 사용되는 값이다. 이는 섀넌 엔트로피(Shannon's Entropy)라고도 불리며 아래와 같은 수식에 의해 계산될 수 있다.

$$H(x) = -\sum_{i=1}^{n} P(i) \bullet \log_2 P(i)$$

$P(i)$는 i가 일어날 확률을 의미하고, 엔트로피 값은 $P(i)$와 $\log_2 P(i)$의 곱의 합으로 표현될 수 있다. 높은 엔트로피 값을 가진다는 것은 해당 정보가 나타나는 확률이 작고 모든 섹션에 자료가 균등하게 분포되었다는 것을 의미한다. 엔트로피 값이 높은 소프트웨어들은 패킹(패킹은 원본 코드를 압축하는 과정을 거치는 작업으로 사용자가 코드를 파악하는 것을 방해한다. 따라서 악성 소프트웨어를 은닉할 때 주로 사용하는 방법이다)된 경우가 많아 대체로 악성 소프트웨어는 정상 소프트웨어에 비해 높은 엔트로피 값을 가진다 (Kwon 등, 2012). 이 값들은 암호화된 프로그램 (Han 등, 2009)이나 소프트웨어들의 특징을 반영 (Lyda와 Hamrock, 2007)하는 데에 자주 이용되므로 악성 소프트웨어의 유사도를 비교하는 데에 유용하게 쓰일 수 있다.

주어진 자료에서 클래스별로 엔트로피들을 계산한 결과 Figure 2.5와 같은 분포를 따르는 것을 확인할 수 있었다. 두 집단 외에는 정상 코드에 비해 큰 값을 가지며 클래스별로 분포의 차이가 존재하는 것을 볼 수 있다. 따라서 우리는 bytes 파일들의 1-gram 엔트로피 값을 계산하여 변수로 사용하였다.

2.5.3. 섹션 이름(section name) 우리는 앞서 Table 2.1에서 섹션 이름을 살펴보았다. 섹션 이름은 asm 파일의 가장 앞부분에서 시작된다. 따라서 asm 파일의 첫째 열에서 공백 이전까지의 열들을 추출하면 출현했던 모든 섹션 이름들을 추출할 수 있다. 섹션 이름은 실행 파일의 속성을 나타내기 때문에 어떤 이름들이 많이 등장하는지는 아주 중요하다. 예를 들어 .rsrc의 경우에는 그래픽 정보를 저장하는 섹션이며 .rdata는 읽기 전용의 런타임 에러가 등장했을 때 나타나는 섹션이다. 파일의 실행 과정에 따라서 섹션 이름이 차지하는 비율이 다르고 각각 등장하는 이름도 다양하기 때문에 빈도수를 이용하여 변수를 지정하였다. 또한 빈도수를 세어본 결과 8번째 순위부터는 전체 데이터에서 약 10,000개 이하로 빈도가 떨어지는 것을 확인할 수 있었다. 전체 파일이 약 1만 개였기 때문에 전체 자료의 수보다 작은

Table 2.5. Frequency of section name

Name	HEADER	.text	.data	.idata
Frequency	148,438	2.83e+08	1.28e+09	4,182,869
Name	.rsrc	.reloc	.rdata	
Frequency	17,992	10,296,525	225,991	

빈도가 나온 섹션 이름을 제외하였으며 따라서 우리는 상위 빈도수를 차지한 7개의 섹션 이름을 변수로 지정하였다.

2.5.4. 자료 정의 지시어 Asm 파일에는 변수를 정의하는 데 있어 필요한 기본적인 지시어들이 있다. 일반적으로 db, dw, dd, dq, dt가 존재하며 이는 각각 바이트, 워드, 더블워드, 쿼드워드, 10바이트(변수의 크기 단위)를 가진다. 예를 들어 'stat dw 11' 이라는 코드가 쓰여 있다면 이는 십진수 숫자 11을 stat라는 변수에 워드 크기로 삽입하라는 명령어가 될 것이다. 이들은 변하지 않는 값인 상수가 아닌 변수를 지정할 때 사용되는 지시어들이다. 따라서 이 지시어들이 가지는 크기 단위에 따라 저장될 수 있는 변수 크기의 범위가 다르며 지정될 수 있는 형태 또한 다양하다. 다시 예를 들자면 db는 1바이트 단위의 숫자 또는 문자가 저장 가능하며 dw는 1워드로 1개 또는 2개의 문자, 숫자 또는 오프셋이 저장될 수 있다. 따라서 어떤 파일을 실행할 때 변수를 사용하는 경우라면 무조건 등장하게 되며 만약 실행 파일이 계속해서 어떤 값을 바꾸는 동작을 실행하는 경우라면 (다시 말해서 많은 변수를 필요로 하는 동작이 있다면) 상당히 자주 출현하게 될 것이다. 서론에 언급했듯이 악성 소프트웨어는 사용자가 알아보지 못 하도록 계속해서 변형되는 동작을 취하는 경우가 많기 때문에 이 지시어들은 중요한 변수가 될 수 있다. 실제로 빈도수를 추출해보았을 때 db, dw, dd가 상당히 높은 빈도를 보였다. dq와 dt의 경우 대부분의 파일에서 빈도수가 0에 가까운 값에 수렴하였기 때문에 크게 영향을 미치지 않는다고 여겨져 이는 변수에서 제외하였다. 또한 관측을 통해 섹션별로 지시어의 등장 횟수가 다를 수 있다는 점을 알아차렸다. 따라서 분산 분석을 이용하여 각 섹션 이름별 등장 횟수의 차이를 검정하였으며 특정 섹션(.text, .rdata, .data)에서 유의했기 때문에 해당 섹션별 세 지시어의 출현 빈도수 또한 변수로 선정하였다.

2.5.5. Opcode 악성 소프트웨어 유사도 연구 (Runwal 등, 2012)에서는 실행 파일이 주어졌을 때 opcode의 빈도수를 계산하여 정상 소프트웨어와 악성 소프트웨어를 구별하는 은닉 마코프 모형을 제시했다. 즉 opcode의 빈도수는 소프트웨어의 특징을 반영하는데 이용될 수 있으므로 이를 변수로 만들어 악성 소프트웨어 분류에 이용하고자 한다.

약 1만여 개에 다다르는 모든 asm 파일에 등장하는 opcode를 추출하기 위해서 모든 파일을 열람하는 것은 불가능하다. 그러나 opcode가 asm 파일 내에서 유사한 위치(열)에 존재한다는 것은 육안으로 식별 가능하기 때문에 해당 조건을 통해 training set에 존재하는 모든 opcode를 추출하였다. 그 결과 24,254개의 서로 다른 opcode가 추출되었으며 일차적으로 분산 분석을 통해 악성 소프트웨어 그룹에 따라 각 opcode가 유의한지 확인해 보았다 ($H_0 : \mu_1 = \cdots = \mu_4 = \mu_6 = \cdots = \mu_9$). 분석 결과 총 24,254개의 opcode 중에서 5,241개의 opcode가 유의 수준 0.05를 기준으로 유의하게 나타났으며 이 중에서 유의 확률(p-value)이 0에 수렴하는 상위 198개의 변수를 간추렸다.

우리는 또한 빈도수를 이용하여 2차적으로 변수를 선별하였다. Figure 2.6은 24,254개의 opcode들 중에서 높은 빈도수를 가진 상위 50개의 opcode 빈도 그래프이다. 약 24개의 opcode들이 특히 높은 빈도를 가지고 있으며 특정 시점 이후부터 빈도수가 현저히 떨어지는 것을 볼 수 있다 (cut off: 1,000,000). 또한 상위 24개 opcode 중에서 18개의 opcode들이 분산분석에서 유의했으므로 자주 등장하는 op-

Table 2.6. A sample of opcode selected by the ANOVA

aaa	aad	aam	aas	aBe	adc	aJ	align	aMhhh	and
arg_0	arg_4	arg_8	arg_C	arpl	assume	aVyv	bound	bt	bts
call	cdq	clc	cld	cli	cmc	cmp	cmpsb	cmpsd	cmpsw
Count	cwde	daa	das	dec	Dest	div	Dst	end	ends

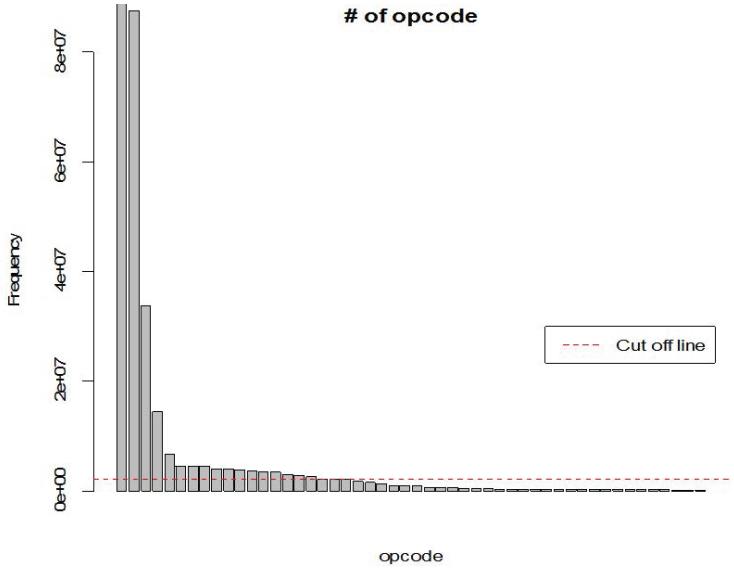


Figure 2.6. Barplot of opcode.

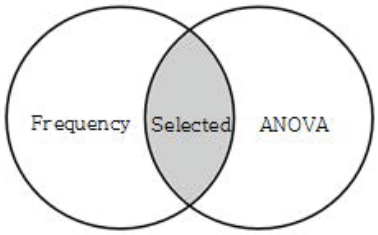


Figure 2.7. Opcode subset selected.

code가 소프트웨어 유형 구분에 영향력 있는 변수라고 볼 수 있을 것이다. 다시 말해서 빈도수가 높은 24개 opcode 중 18개의 opcode가 클래스 별로 유의한 차이를 보였으므로 중복 채택된 18개 opcode 빈도수를 변수로 지정하였다. 분석에 사용된 자료를 정리하면 다음의 Table 2.7과 같다.

3. 분석 결과

우리는 앞서 2.4절에서 각 클래스별로 자료의 수가 상이한 것을 볼 수 있었다. 따라서 데이터를 모형에 적합할 때 모집단의 비율을 유지하기 위해 층화추출을 이용하여 8대 2의 비율로 training set과 test set을 나누었다.

Table 2.7. Variable descriptions to be extracted per one observation

	Category	Type	Description	♯ of var	Source
Response	Class	범주형(1 4, 6 9)	반응 변수 값	1	
Explanatory	1-gram	수치형(0 1, 실수)	파일 내 점유 비율	15	Bytes
	Entropy	수치형(실수)	계산된 값	1	Bytes
	Section Name	수치형(정수)	섹션 이름의 빈도	7	Asm
	Directive	수치형(정수)	지시어 등장 빈도	13	Asm
	Opcode	수치형(정수)	출현 빈도수	18	Asm

Table 3.1. The proportion of training to test set

Data	Class								Total
	1	2	3	4	6	7	8	9	
Train	1,226	1,979	2,349	362	600	309	970	810	8,605
Test	307	495	588	91	151	78	243	203	2,156
Total	1,533	2,474	2,937	453	751	387	1,213	1,013	10,761

Table 3.2. Confusion matrix of polychotomous logistic regression

		Predicted class								error.
		1	2	3	4	6	7	8	9	rate
Actual class	1	196.50	6.98	61.98	2.91	2.80	5.44	15.28	15.11	0.3599
	2	8.20	434.70	20.48	0.39	0.69	1.09	20.43	9.02	0.1218
	3	0.12	0.00	586.06	0.04	0.32	0.23	0.57	0.66	0.0032
	4	0.02	0.00	38.02	51.91	0.01	0.92	0.01	0.11	0.4296
	6	1.46	0.09	37.79	0.68	95.01	0.80	5.44	9.73	0.3707
	7	0.03	1.41	66.17	0.00	0.00	4.85	0.00	5.54	0.9378
	8	10.58	1.11	48.47	2.91	2.99	7.02	162.44	7.48	0.3315
	9	5.60	0.09	42.23	0.75	0.77	0.44	0.84	152.28	0.2499

3.1. 다항 로지스틱 회귀 모형 (Model 1)

Table 3.2는 다항 로지스틱 모형을 100번 반복하여 얻은 평균 오차 행렬이다. 즉 매회 층화추출을 이용하여 다른 training set과 test set을 8대 2로 나누어 100번 모형을 적합시킨 뒤에 평균치를 구한 결과이다. 클래스 1, 4, 6, 8의 경우 1/3이 넘는 자료가 잘못 분류된 것을 볼 수 있으며 클래스 7의 경우 자료의 약 94%가 오분류된 것을 확인할 수 있다. 전체 오분류율은 472.25/2156 ≒ 0.2190이며 클래스 3을 제외하고는 대각 행렬의 값들이 실제 값에 비해 낮은 비율을 차지하고 있으므로 적합시키기에 적절한 모형이 아님을 볼 수 있다.

3.2. 랜덤 포레스트 모형 (Model 2)

Table 3.3은 마찬가지로 랜덤 포레스트 모형을 100회 반복한 뒤 얻은 오차 행렬이다. 표를 살펴보면 랜덤 포레스트 모형의 결과가 앞서 시행된 다항 로지스틱 분석 결과와는 크게 다르다는 것을 볼 수 있다. 전체 test set의 자료 수가 2,156개이며 잘못 분류된 총 자료의 수가 7.34개이므로 100번 적합 시에 평균 오분류율은 7.34/2156 ≒ 0.0034다. 이전 절의 다항 로지스틱 분석에 비해 정확도가 상당히 오른 것을 볼 수 있다. 확인을 위해 의사 결정 나무 모형(decision tree)을 적용했을 때 다항 로지스틱 모형에 비해 정확한 분류를 할 수 있었으며 랜덤 포레스트 모형보다는 분류율이 떨어지는 것을 볼 수 있었다. 이를

Table 3.3. Confusion matrix of random forest

		Predicted class								error.
		1	2	3	4	6	7	8	9	rate
	1	306.14	0.00	0.00	0	0.24	0.00	0.62	0.00	0.0028
	2	0.00	495.00	0.00	0	0.00	0.00	0.00	0.00	0.0000
	3	0.00	0.00	588.00	0	0.00	0.00	0.00	0.00	0.0000
Actual	4	0.02	0.00	0.00	90	0.95	0.00	0.03	0.00	0.0110
class	6	0.00	0.00	0.00	0	150.00	1.00	0.00	0.00	0.0066
	7	0.00	0.00	0.00	0	0.00	78.00	0.00	0.00	0.0000
	8	2.26	0.00	0.00	0	0.22	0.00	240.52	0.00	0.0102
	9	1.98	0.00	0.00	0	0.02	0.00	0.00	201.00	0.0099

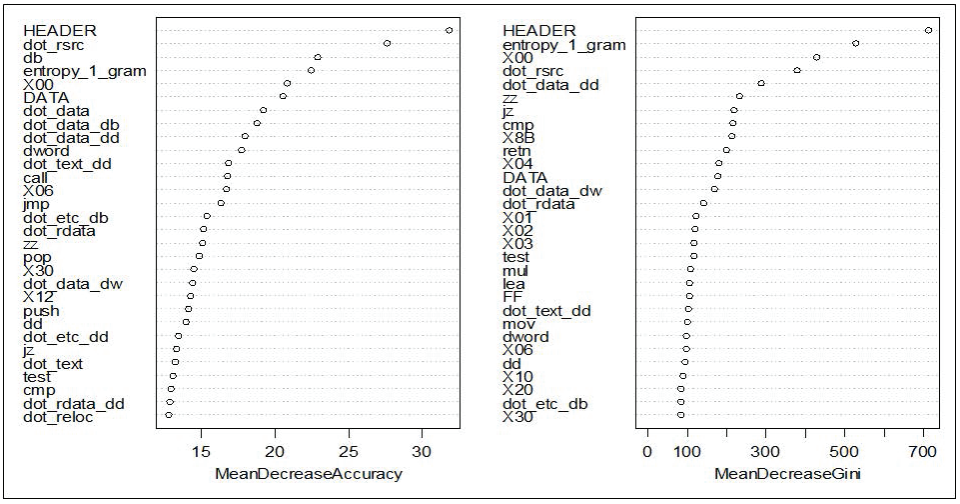


Figure 3.1. Importance variable plot.

통해 자료의 비선형성이 상당히 크다는 점을 볼 수 있으며 해당 데이터는 나무 모형을 기반으로 하기에 적합한 데이터임을 추론해볼 수 있다.

Figure 3.1은 랜덤 포레스트 모형 생성 후에 중요 변수(Importance variable) 그래프를 그린 결과이다. 중요한 변수로는 asm 파일에서의 HEADER:(섹션 이름)의 개수, .rsrc:의 개수, db(데이터 정의 지시어)의 개수, 엔트로피 값, bytes 파일에서 00이 차지하는 비율 등이 꼽힌 것을 볼 수 있다. 특히 HEADER:나 .rsrc:의 경우에는 정확도 개선이나 노드 불순도를 줄이는 두 과정에서 큰 역할을 하는 것을 볼 수 있는데, 이는 섹션 이름이나 특정 명령어의 빈도수가 악성 소프트웨어의 유형을 구분하는 데 유효하다는 것을 의미한다.

3.3. 그래디언트 부스팅 모형 (Model 3)

Table 3.4는 그래디언트 부스팅을 100회 반복한 결과로 평균 오차 행렬이다. n.trees를 3,000으로 지정하였으며 나머지 변수들은 디폴트값을 이용하여 모형을 구축했다. 변수별 상대 중요도를 살펴보았을 때 랜덤 포레스트와 유사하게 HEADER:의 빈도수, 엔트로피 값, bytes 파일에서 00의 비율 순으로 확인할 수 있다. 전체 오분류율은 62.7/2156 ≒ 0.0291로 랜덤 포레스트 모형보다 약간 정확도가 떨어진다.

Table 3.4. Confusion matrix of gradient boosting

		Predicted class								error.
		1	2	3	4	6	7	8	9	rate
	1	305	0	0	0	2	0	0	0	0.0065
	2	0	495	0	0	0	0	0	0	0.0000
	3	0	0	588	0	0	0	0	0	0.0000
Actual	4	0	0	4	85	0	0	0	2	0.0659
class	6	6	0	0	6.15	131	1	4.85	2	0.1325
	7	0	0	0	0	0	77.9	0	0.1	0.0013
	8	13.8	0	0	0	10.8	1	216.4	1	0.1094
	9	1	0	0	0	1	0	6	195	0.0394

Table 3.5. Confusion matrix of Support vector machine

		Predicted class								error.
		1	2	3	4	6	7	8	9	rate
	1	282.09	6.95	0.16	1.51	0.55	0.52	13.41	1.81	0.0811
	2	56.91	391.34	0.01	35.51	1.14	5.59	4.46	0.04	0.2094
	3	35.06	1.64	545.82	0.06	1.97	2.54	0.91	0.00	0.0717
Actual	4	1.32	4.06	1.63	75.36	1.59	0.58	5.34	1.12	0.1719
class	6	1.56	5.28	0.08	7.45	129.13	0.00	2.25	5.25	0.1448
	7	0.12	2.08	7.98	0.99	1.06	65.76	0.00	0.01	0.1569
	8	9.06	2.83	0.28	1.56	1.15	0.70	225.02	2.40	0.0740
	9	0.42	2.14	0.00	34.56	8.20	0.00	6.33	151.35	0.2544

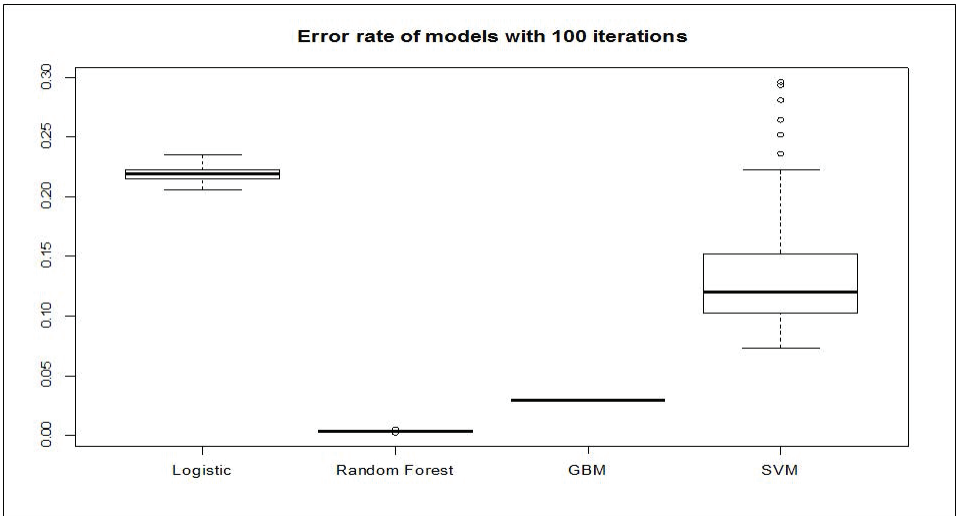


Figure 3.2. Error rate of models with 100 iterations.

3.4. 서포트 벡터 기계 (Model 4)

Table 3.5의 평균 오차 행렬을 통해 서포트 벡터 기계의 전체 오분류율은 290.13/2156 ≒ 0.1346으로 랜덤 포레스트나 그래디언트 부스팅 모형에 비하여 상대적으로 오분류율이 높은 것을 확인할 수 있다. 모

Table 4.1. Experiment environment

Type	Name	Specification
PC	OS	Windows 7 Enterprise K
	CPU	Intel(R)Core(TM) i7-4790 3.60GHz
	RAM	16.0GB
Program	R	64bit, version3.3.2

Table 4.2. Function running time

bytes file		asm file		Mean time
Start of interval	End of interval	Mean rows	Median rows	(sec)
15001	35000	100214	57904	10.937
35001	55000	364987	301067	23.618
55001	75000	521189	452708	37.297
75001	95000	464631	71814	45.687
95001	115000	605968	179972	62.032

형 적합 시 계산 소요시간 역시 그래디언트 부스팅 모형과 함께 가장 긴 시간이 소요되었으므로 주어진 자료에 적절한 모형이 아니라고 여겨졌다.

Figure 3.2는 모형을 100회씩 반복하면서 발생한 오분류율을 상자그림으로 그린 결과이다. 서포트 벡터 기계가 가장 큰 분산을 보였으며 다항 로지스틱 모형이 평균적으로 가장 높은 오차율을 가지는 것을 볼 수 있다. 그래디언트 부스팅 모형의 경우 분산이 가장 작았지만 랜덤 포레스트 모형에 비해 잘못 분류하는 비율이 높다. 따라서 가장 낮은 오분류율을 가지고 분산 또한 크기 않은 랜덤 포레스트 모형을 최종 모형으로 선정하였다.

4. 결론

지금까지 악성 소프트웨어를 분류하기 위한 통계적 기법들을 제안하였다. 본 연구의 의의는 컴퓨터 언어로 나열되어 있는 줄글 형식의 소스 코드로부터 중요한 정보를 파악하고 주어진 자료를 통계적 분류 모형에 적합한 형태로 가공하는 데에 있다. 즉 상당히 비정형화 되어 있던 자료로부터 변수를 추출하고 유효한 변수를 선별하는 과정이 가장 핵심이 되는 부분이다. 우선 데이터 전처리 과정에서는 선행 연구를 바탕으로 bytes 파일의 정보를 반영하기 위해 N-gram 기법을 이용한 16진수 언어 중 일부의 등장 비율과 엔트로피 값을 계산하여 변수로 이용했다. 또한 asm 파일에서 등장한 명령어들을 분산 분석에서 유의한 변수들의 집합을 선정하였다. 그리고 출현 빈도가 높은 변수의 집합 또한 선정한 뒤, 두 집합의 교집합이 되는 변수들을 최종적으로 모형에 사용하기로 결정하였다. 우리는 실험을 통해 초반에 추출했던 모든 변수를 이용했을 때보다 유효한 변수를 추려 모형에 적합시켰을 때 정확도가 상승하는 것을 확인할 수 있었는데, 이는 변수가 아주 많은 경우에 실제로 영향력 있는 변수들을 선택하여 모델링하는 것이 모든 변수들을 이용하는 것보다 좋은 결과를 가질 수 있다는 것을 의미할 것이다. 그러므로 원자료에서 나온 수많은 변수들 중 주요 변수를 뽑는 과정은 자료 분석 및 모형 적합 과정에서 아주 중요하다고 할 수 있다.

앞서 주어진 데이터는 랜덤 포레스트 모형과 그래디언트 부스팅 모형 순으로 적절하다는 것을 확인하였다. 다시 말해서 본 자료는 비선형성이 매우 강해 나무 모형을 기반으로 하는 모형에 적합한 자료라는 사실을 생각해볼 수 있을 것이다. 따라서 다항 로지스틱 회귀 모형이나 서포트 벡터 기계보다는 랜덤 포레스트나 그래디언트 부스팅 모형에서 낮은 오분류율을 보였으며, 특히 랜덤 포레스트 모형에서 굉장히

높은 정확도를 보여주었다. 주요 변수는 HEADER:(섹션 이름)의 개수, .rsrc:의 개수, db(데이터 정의 지시어)의 개수, 엔트로피 값, bytes 파일에서 00이 차지하는 비율 순으로 뽑혔다. 또한 이 변수들은 랜덤 포레스트 뿐 아니라 그래디언트 부스팅 등 다른 모형들에서도 중요하다고 꼽히는 것을 볼 수 있었다.

마지막으로 최종 모형을 결정한 후 실용성을 파악하기 위해 소요되는 함수의 계산 시간을 측정해보았다. 최종 함수는 bytes, asm 두 파일을 입력했을 때 어떤 그룹의 악성 소프트웨어인지 도출되도록 짜였으며 실험 환경은 아래 Table 4.1과 같다.

Table 4.2는 bytes 파일 행수 중 사분위수(1사분위, 3사분위)에 속하는 파일들을 5개의 등간격으로 나눈 뒤 해당 범위에 분포하는 bytes 파일과 asm 파일을 10개씩 뽑아 평균 함수 실행 시간을 적은 결과이다. 대체로 소요 시간의 증가 폭이 균일한 것으로 보아 함수의 실행 시간은 asm 파일보다는 bytes 파일의 길이에 비례한다고 생각할 수 있다.

본 연구에서는 악성 소프트웨어 내의 집단들을 구분하는 모형을 제안하였다. 향후 연구 과제로 정상 소프트웨어와 악성 소프트웨어를 구분하는 2단계 분류 모형까지 고려하고 있다. 1차적으로 실제 소프트웨어가 정상 소프트웨어인지 구분하고 2차로 어떤 악성 소프트웨어 유형인지 분류할 수 있다면 활용도가 훨씬 높아질 것으로 예상된다.

References

Brieman, L. (2001). Random forests, *Machine Learning*, **45**, 5–32.

Brieman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*, Chapman and Hall, New York.

Chen, L. and Aritsugi, M. (2006). *An SVM-Based Masquerade Dection Method with Online Update Using Co-occurrence Matrix*, DIMVA 2006, LNCS 4064, 37–53.

Choi, J., Kim, H., Kim, K., Park, H., and Song, J. (2014). A study on extraction of optimized API sequence length and combination for efficient malware classification, *Journal of The Korea Institute of Information Security & Cryptology*, **24**, 897–909.

Cortes, C. and Vapnik, V. (1995). Support-vector networks, *Machine Learning*, **20**, 273–297.

Dahl, G. E., Stokes, J, W., Deng, L., and Yu, D. (2013). LARGE-SCALE MALWARE CLASSIFICATION USING RANDOM PROJECTIONS AND NEURAL NET WORKS, Acoustics, Speech and Processing (ICASSP), IEEE.

Friedman, J. (2002). Stochastic gradient boosting, *Computational Statistics & Data Analysis*, **38**, 367–378.

Han, S., Lee, K., and Lee, S. (2009). Packed PE file detection for Malware forensics, *2nd International Conference on Computer Science and its Applications*, CSA.

Kim, M., Lee, J., Chang, H., Cho, S., and Park, Y. (2010). Design and performance evaluation of binary code packing for protecting embedded software against reverse engineering, *In 13th IEEE International Symposium, (ISORC)*, 80–86.

Konrad, R. (2011). Automatic analysis of malware behavior using machine learning, *Journal of Computer Security*, **19**, 639–668.

Kwon, H., Kim, S., and Im, E. (2012). An Malware classification system using multi N-gram, *Journal of Security Engineering*, **9**, 531–542.

Lyda, R. and Hamrock, J. (2007). Using entropy analysis to find encrypted and packed malware, *IEEE Security & Privacy*, **5**.

Ridgeway, G. (2007). Generalized Boosted Models: A guide to the gbm package, https://cran.r-project.org/web/packages/gbm/

Runwal, N., Low, R. M., and Stamp, M. (2012). Opcode graph similarity and metamorphic detection, *Journal in Computer Virology*, **8**, 37–52.

Santos, I., Penya, Y. K., Devesa, J., and Bringas, P. G. (2009). N-grams-based file signatures for malware detection, *11th International Conference on Enterprise Information Systems (ICEIS), AIDSS*, 317–320.

통계적 기법을 이용한 악성 소프트웨어 분류

원성민[a] · 김현주[a] · 송종우[a,1]

[a]이화여자대학교 통계학과

(2017년 8월 14일 접수, 2017년 09월 29일 수정, 2017년 10월 12일 채택)

요 약

최근 워너크라이라는 이름의 랜섬웨어가 전 세계적으로 큰 화두에 오르면서, 악성 소프트웨어로 인한 피해를 줄이기 위한 방법들이 재조명 되고 있다. 새로운 악성 소프트웨어가 발생했을 때 피해를 최소화하기 위해서는 해당 소프트웨어가 어떤 공격 유형을 가진 악성 소프트웨어인지 빠르게 분류할 필요가 있다. 본 연구 목적은 다양한 통계적 기법을 이용하여 악성 소프트웨어를 효과적으로 분류할 수 있는 모형을 구축하는 데 있다. 모형 적합 시 다항 로지스틱, 랜덤 포레스트, 그래디언트 부스팅, 서포트 벡터 기계 등의 기법들을 이용하였으며, 본 연구를 통해 악성 소프트웨어를 분류하는 데에 있어 중요한 역할을 하는 변수들이 존재한다는 사실을 발견하였다.

주요용어: 악성 소프트웨어, 다중 분류, 랜덤 포레스트, 그래디언트 부스팅, 서포트 벡터 기계, 주요변수

이 논문 또는 저서는 2017년 대한민국 교육부와 한국연구재단의 지원을 받아 수행된 연구임 (NRF-2017R1D1A1B03036078).

[1]교신저자: (03760) 서울특별시 서대문구 이화여대길 52, 이화여자대학교 통계학과.
E-mail: josong@ewha.ac.kr