한국컴퓨터정보학회 동계학술대회 논문집 제26권 제1호 (2018. 1)

악성코드 탐지를 위한 기계학습 알고리즘의 성능 비교

이현종[O], 허재혁[*], 황두성[*]
[O]단국대학교 소프트웨어학과
[*]단국대학교 컴퓨터공학과
e-mail: 72170146@dankook.ac.kr[O], {72170150, dshwang}@dankook.ac.kr[*]

Performance Comparison of Machine Learning Algorithms for Malware Detection

Hyun-Jong Lee[O], Heo-Jae Hyeok[*], Doosung Hwang[*]
[O]Dept. of Software, Dankook University
[*]Dept. of Computer Engineering, Dankook University

● 요 약 ●

서명기반 악성코드 탐지는 악성 파일의 고유 해싱 값을 사용하거나 패턴화된 공격 규칙을 이용하므로, 변형된 악성코드 탐지에 취약한 단점이 있다. 기계 학습을 적용한 악성코드 탐지는 이러한 취약점을 극복할 수 있는 방안으로 인식되고 있다. 본 논문은 정적 분석으로 *n*-gram과 API 특징점을 추출해 특징 벡터로 구성하여 XGBoost, *k*-최근접 이웃 알고리즘, 지지 벡터 기기, 신경망 알고리즘, 심층 학습 알고리즘의 일반화 성능을 비교한다. 실험 결과로 XGBoost가 일반화 성능이 99%로 가장 우수했으며 *k*-최근접 이웃 알고리즘이 학습 시간이 가장 적게 소요됐다. 일반화 성능과 시간 복잡도 측면에서 XGBoost가 비교 대상 알고리즘에 비해 우수한 성능을 보였다.

키워드: 의사 결정 트리(decision tree), 기계 학습(machine learning), 악성 코드(malware)

I. Introduction

악성코드(malware)란 사용자 컴퓨터를 침해하고 정상적인 기능을 수행하지 못하게 하는 프로그램으로 트로이목마, 웜 바이러스, 키로거, 스파이 웨어, 애드 웨어, 랜섬웨어(ransom-ware) 등이 대표적이다. 이러한 악성코드 종류의 수는 많으며 새로운 형태의 악성코드로 발전하고 있다. 과거의 서명 기반 악성코드 탐지 방법은 발현되었던 악성코드의 고유 해싱 값을 사용하거나 패턴화된 공격 규칙을 이용해 탐지하기 때문에 새로운 형태의 악성코드 탐지에 취약하다[1]. 최근에는 기계 학습을 적용한 악성코드 탐지 방법이 이러한 취약점을 극복할 수 있는 방안으로 인식되고 있으며 관련 연구가 진행되고 있다[2,3,4].

본 논문에서는 정적 분석으로 *n*-gram과 API 특징을 추출해 특징 벡터를 구성한다. 추출한 특징 벡터와 두 특징을 통합한 특징 벡터를 이용해 XGBoost 알고리즘, *k*-최근접 이웃 알고리즘, 지지 벡터 기기, CNN(Convolutional Neural Network) 모델의 성능을 비교하여 어떤 학습 알고리즘이 악성 코드 분류 문제에 적합한지 평가한다.

본 논문의 구성은 다음과 같다. II장에서는 기 연구된 악성코드 탐지 방법을 소개하고 III장에서는 악성코드 탐지를 위한 특징점 추출과 의사 결정 트리(decision tree) 기반 부스팅 알고리즘인 XGBoost를 소개한다. IV장에서는 기계 학습 알고리즘과 성능을 비교하고 V장에서는 실험 결과를 토대로 어떤 학습 알고리즘이 적합한지 토의한다.

II. Related works

Youngjoon Ki 외 2명은 서명 기반 악성코드 탐지를 개선시키기 위해 동적 분석(dynamic analysis)을 통한 API 실행 순서 정보를 추출하고 DNA 시퀀스 정렬 알고리즘을 사용해 악성코드 종을 분류했다. 23,080개의 악성코드 데이터와 66개의 정상코드 데이터를 사용해 99%의 정확도를 보였다[5].

Igor Santos 외 3명은 *n*-gram 특징점을 추출하고 *k*-최근접 이웃 알고리즘을 사용해 149,882 개의 악성코드와 4,934개의 정상 코드 데이터에 대해서 N=4, k=2 파라미터를 설정하여 91.25%의 성능을 보였다[2].

Ivan Firdausi 외 2명은 220개의 악성코드와 250개의 정상코드를 수집하고 동적 분석기 Anubis[6]를 활용한 동적 분석의 결과물을 특징 벡터로써 사용해 분류 문제를 해결했다. *k*-최근접 이웃 알고리즘, Naive Bayes,지지 벡터 기기, J48 의사 결정 트리, MLP 학습 모델의 성능을 비교했으며 대부분 모델에서 정확도 90% 이상의 성능을 보였다[3].

Rafiqul Islam 외 3명은 2,398 개의 악성코드와 541 개의 정상코드로 구성된 데이터 셋에서 정적 분석으로 문자열 정보의 빈도수를 추출하고 동적 분석으로 코드 실행 과정에서 호출된 API 정보를 추출하여 특징 벡터가 구성됐다. 추출된 특징 벡터를 사용해 지지

벡터 기기, *k*-최근접 이웃 알고리즘, 의사 결정 트리, 랜덤 포레스트(random forest)의 성능을 비교하여 랜덤 포레스트가 정확도 97%로 성능이 가장 우수했으며 동적 분석 특징점을 사용했을 때보다 정적 분석과 동적 분석 특징점을 통합해서 사용했을 때가 성능이 더 우수한 것으로 평가했다[4].

III. The Proposed Approach

실험에 사용된 데이터는 Kaggle의 Microsoft Malware Classification Challenge(BIG 2015) 대회에 사용된 데이터를 사용한다[7]. 데이터 수는 총 10,869개이며 9 종류의 악성코드로 구성된 데이터 셋이다. 각 데이터는 악성코드 실행 파일의 16진수 바이트 정보와 IDA(Interactive Disassembler)로 분석한 결과를 포함한다.

Fig 1은 학습 데이터 준비부터 모델 평가까지의 과정을 보여준다. *n*-gram 분석은 순서 있는 데이터 시퀀스(sequence)에서 *n*개의 데이터 요소로 구성된 부분 시퀀스를 연속적으로 생성하는 분석 방법으로 각 부분 시퀀스의 빈도수로 데이터별 연관 관계를 파악할 수 있다. 악성코드 바이트 정보에서 2-gram 특징점을 추출하여 '0000' 바이트부터 'FFFF '바이트 까지 두 바이트 조합의 빈도수로 구성된 65,536 차원의 특징 벡터를 얻을 수 있다.

IDA 분석은 악성코드 실행 파일의 PE(Portable Executable) 섹션(sections) 분석 정보를 담고 있다. .idata 섹션에는 일반적으로 IAT(Import Address Table) 정보를 포함한다[1]. IAT는 악성코드에서 사용되는 DLL과 DLL 내 API 사용 정보를 담고 있으며 윈도우에서 제공하는 DLL과 사용자 정의 DLL 파일이 IAT에 기록이 될 수 있다. 사용된 API를 분석하기 위해 IAT 정보를 추출해 JSON 형태의 파일로 저장했다(Fig 2). 주요한 API만 뽑기 위해 사용자 정의 API를 제외하고 악성코드에서 흔히 사용되는 DLL 11개를 선정하여 API 범위를 제한했다(Table 1).

Table 1. DLLs List Often Used in Malwares

Common DLLs			
User32.dll	Kernel32.dll	Advapi32.dll	Ntdll.dll
Ws2_32.dll	Wininet.dll	wsock32.dll	Shell32.dll
Msvcrt.dll	Ole32.dll	Oleaut32.dll	

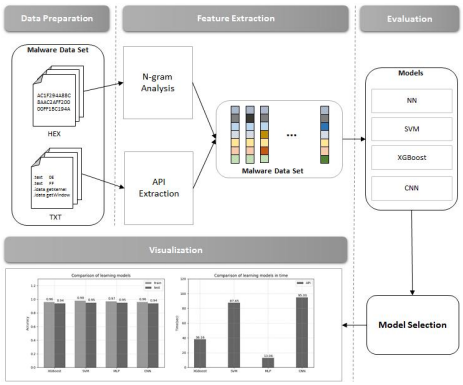

Fig. 1. The Process of Evaluation

```
"filename":"a528001d0074f0c140f6e9f
"imports":{
    "kernel32.dll":[
        "GetCurrentProcessId",
        "VirtualAlloc",
        "GetCommandLineW",
        "GlobalUnlock",
        ],
    "user32.dll":[
        "GetDialogBaseUnits"
        ],
    "mscoree.dll":[
        "_CorExeMain"
        ],
}
```

Fig. 2. The Example of JSON File

XGBoost 알고리즘은 Tianqi Chen 외 1명에 의해 제안됐으며 gradient tree boosting 기법을 사용해 다양한 주제의 challenge에서 높은 성능을 보여 다양한 문제에 적용가능하다는 것을 입증했다[7].

XGBoost 알고리즘은 트리(tree)의 터미널(terminal) 노드에 실수 값 가중치가 부여된다. XGBoost 알고리즘은 터미널 노드의 가중치를 이용해 입력 데이터를 분류한다. 주어진 데이터 셋 $D = (x_i, y_i)$ $(|D| = n, x_i \in \mathbb{R}^m, y_i \in \{1, 2, \cdots, c\})$에 서 n은 데이터 크기이며 x_i는 m 차원 벡터, y_i는 클래스 레이블, c는 클래스 수이다. 데이터 셋에 대해 XGBoost의 분류는 수식 (1)로 정의된다.

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

여기서 $F = f(x) = w_{q(x)}$ $(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$ K는 모델을 구성하는 트리의 수를 나타내며 F 는 트리의 공간(space)을 정의하고 q는 입력 변수 x를 해당 터미널 노드로 사상(mapping)시키는 함수이다. T는 해당 트리를 이루는 터미널 노드의 인덱스를 나타내고 f_k는 k번째 독립된 트리 구조 q와 해당 트리의 가중치 벡터 w를 나타낸다. 입력 데이터를 q개의 각 트리에서 의사 결정 트리 규칙을 이용해 터미널 노드의 가중치 $w_{q(x)}$를 합산하여 분류한다.

IV. Experiments

클래스 별 데이터 구성은 Table 2와 같다. Malware challenge 데이터 셋은 10,868개의 데이터를 갖고 9개의 클래스로 구성된 다중 분류 문제이다.

Table 2. Data Size on Each Classes

Seq.	Type	Size
1	Ramnit	1,541
2	Lollipop	2,478
3	Kelihos_ver3	2,942
4	Vundo	475
5	Simda	42
6	Tracur	751
7	Kelihos_ver1	398
8	Obfuscator.ACY	1,228
9	Gatak	1,013
Total		10,868

Table 3은 데이터 셋에서 제안하는 특징 벡터를 추출했을 때 차원 비교를 보여준다. 2-gram 특징 벡터의 각 특징은 65,536가지의 두 바이트의 조합이 있으며 API 특징 벡터의 차원은 7,094로 데이터 셋에서 추출된 API 수와 동일하다. 2-gram+API 특징 벡터는 각각의 특징벡터를 같이 사용한 것이며 차원 수는 72,630이다.

Table 3. Dimension of Features

Features	No. Dimension
2-gram	65,536
API	7,094
2-gram + API	72,630

실험 시스템 환경으로 CPU는 Intel(R) Xeon(R) CPU E5-2623 v4 @ 2.60GHz, 메모리는 128GB, 운영체제는 Linux mint를 사용했다.

준비된 악성코드 데이터 셋에 대해서 k-최근접 이웃 알고리즘, 지지 벡터 기기, XGBoost, CNN 알고리즘의 성능 평가를 수행했다. k-최근접 이웃 알고리즘의 경우 k는 1로 설정했다. 지지 벡터 기기의 경우 선형(linear) 커널을 사용했고 $C = 1000$와 같이 파라미터를 설정했다. XGBoost의 경우 트리 수는 50개, 트리 최대 깊이는 3으로 설정했다. CNN의 네트워크 구조는 Table 4와 같으며 학습 파라미터로 batch 크기는 200, epochs는 10으로 설정했고 SGD(Stochastic Gradient Descent) 최적화를 사용했다. 지역 최소점(local minumum point)을 피하기 위해서 학습률(learning rate)은 매 epoch의 가중치 업데이트마다 1e-6만큼 감소된다.

Table 4. The CNN Network

Layer	Description	Activation Function
Conv2D	32 x (3 x 3)	Rectifier
MaxPooling	(2 x 2)	
Conv2D	64 x (3 x 3)	Rectifier
MaxPooling	(2 x 2)	
Hidden	500	Rectifier
Dropout	50%	
Hidden	500	Rectifier
Dropout	30%	
Output	9	Softmax

성능 평가로 5 식 교차 검증(5-way cross-validation)을 수행했으며 성능 비교의 척도로써 정확도와 학습 시간을 사용했다. Table 5는 각 학습 모델의 정확도와 시간 평균값을 보여준다. 학습 시간 측면에서 k-최근접 이웃 알고리즘이 775.97초로 가장 빨랐다. 정확도는 XGBoost가 99%로 가장 우수했으며 학습 데이터에 대한 정확도와 테스트 데이터에 대한 정확도의 차이가 가장 적었다. 대부분 학습 모델에서 2-gram과 API 특징 벡터를 각각 사용했을 때 보다 같이 사용했을 때 성능이 개선되었으며 XGBoost 모델의 경우 API 특징 벡터를 사용했을 때보다 4% 만큼 분류 성능이 개선됐다.

Table 5. Performance of Each Algorithms

Models	Features	Train Accuracy	Test Accuracy	Time
1-NN	2-gram	1.00	0.96	705.28
	API	1.00	0.95	142.75
	API + 2-gram	1.00	0.95	755.97
SVM	2-gram	1.00	0.98	2314.38
	API	0.99	0.96	128.31
	API + 2-gram	1.00	0.98	1734.90
XGBoost	2-gram	1.00	0.97	2286.10
	API	0.97	0.95	129.04
	API + 2-gram	0.99	0.99	1396.61
CNN	2-gram	0.88	0.88	15370.64
	API	0.99	0.98	1267.87
	API + 2-gram	0.99	0.98	16783.75

Fig 3은 각 모델에서 특징 벡터에 따른 정확도 비교를 보여주며 Fig 4는 학습 시간 비교를 보여준다.

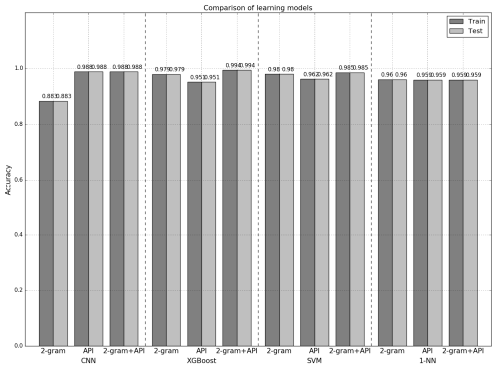

Fig. 3. Comparison of learning models

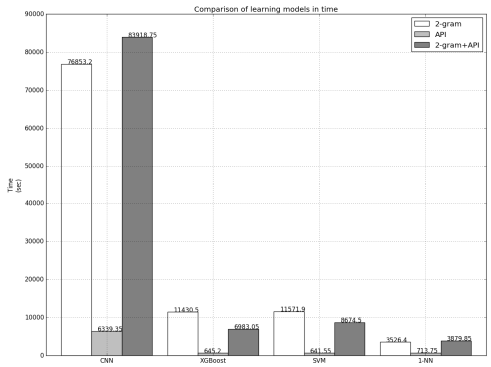

Fig. 4. Comparison of learning models on time

V. Conclusion

본 논문에서는 악성코드 탐지를 위한 기계학습 알고리즘의 성능을 비교했다. 성능 평가를 위한 데이터는 Malware Challenge 데이터를 사용했다. 테스트 데이터에 대한 정확도는 XGBoost, CNN, 지지 벡터 기기, k-최근접 이웃 알고리즘 순으로 성능이 우수했다. 각 학습 모델에서 2-gram 특징 벡터와 API 특징 벡터를 각각 사용하는 것보다 같이 사용하는 것이 성능이 개선되었다.

본 논문에서 제안된 특징 벡터의 차원은 72,630이며 이는 학습

데이터에 비해 큰 차원을 갖으며 모델을 악성코드 탐지 시스템에 적용해 사용할 경우 과적합(overfitting) 현상이 나타날 가능성이 있다. 따라서 차원을 축소하거나 악성코드의 고유한 특징을 나타낼 수 있는 특징을 추출하는 과정이 필요하다.

REFERENCES

[1] Sikorski, Michael, and Andrew Honig. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press, 2012.

[2] Santos, Igor, et al. "n-grams-based File Signatures for Malware Detection." ICEIS (2) 9 (2009): 317-320.

[3] Firdausi, Ivan, Alva Erwin, and Anto Satriyo Nugroho. "Analysis of machine learning techniques used in behavior-based malware detection." Advances in Computing, Control and Telecommunication Technologies (ACT), 2010 Second International Conference on. IEEE, 2010.

[4] Islam, Rafiqul, et al. "Classification of malware based on integrated static and dynamic features." Journal of Network and Computer Applications 36.2 (2013): 646-656.

[5] Ki, Youngjoon, Eunjin Kim, and Huy Kang Kim. "A novel approach to detect malware based on API call sequence analysis." International Journal of Distributed Sensor Networks 11.6 (2015): 659101.

[6] Anubis, http://anubis.iseclab.org/

[7] https://www.kaggle.com/c/malware-classification

[8] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.