API 정보와 기계학습을 통한 윈도우 실행파일 분류

Classifying Windows Executables using API-based Information and Machine Learning

저자
(Authors)

조대희, 임경환, 조성제, 한상철, 황영섭
DaeHee Cho, Kyeonghwan Lim, Seong-je Cho, Sangchul Han, Young-sup Hwang

출처
(Source)

정보과학회논문지 43(12), 2016.12, 1325-1333 (9 pages)
Journal of KIISE 43(12), 2016.12, 1325-1333 (9 pages)

발행처
(Publisher)

한국정보과학회
KOREA INFORMATION SCIENCE SOCIETY

URL

http://www.dbpia.co.kr/Article/NODE07079310

APA Style

조대희, 임경환, 조성제, 한상철, 황영섭 (2016). API 정보와 기계학습을 통한 윈도우 실행파일 분류. 정보과학회논문지, 43(12), 1325-1333.

ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 43, No. 12, pp. 1325-1333, 2016. 12
http://dx.doi.org/10.5626/JOK.2016.43.12.1325

API 정보와 기계학습을 통한 윈도우 실행파일 분류

(Classifying Windows Executables using API-based Information and Machine Learning)

조 대 희 † 임 경 환 † 조 성 제 †† 한 상 철 ††† 황 영 섭 ††††
(DaeHee Cho) (Kyeonghwan Lim) (Seong-je Cho) (Sangchul Han) (Young-sup Hwang)

요 약 소프트웨어 분류 기법은 저작권 침해 탐지, 악성코드의 분류, 소프트웨어 보관소의 소프트웨어 자동분류 등에 활용할 수 있으며, 불법 소프트웨어의 전송을 차단하기 위한 소프트웨어 필터링 시스템에도 활용할 수 있다. 소프트웨어 필터링 시스템에서 유사도 측정을 통해 불법 소프트웨어를 식별할 경우, 소프트웨어 분류를 활용하여 탐색 범위를 축소하면 평균 비교 횟수를 줄일 수 있다. 본 논문은 API 호출 정보와 기계학습을 통한 윈도우즈 실행파일 분류를 연구한다. 다양한 API 호출 정보 정제 방식과 기계학습 알고리즘을 적용하여 실행파일 분류 성능을 평가한다. 실험 결과, PolyKernel을 사용한 SVM (Support Vector Machine)이 가장 높은 성공률을 보였다. API 호출 정보는 바이너리 실행파일에서 추출할 수 있는 정보이며, 기계학습을 적용하여 변조 프로그램을 식별하고 실행파일의 빠른 분류가 가능하다. 그러므로 API 호출 정보와 기계학습에 기반한 소프트웨어 분류는 소프트웨어 필터링 시스템에 활용하기에 적당하다.

키워드: 소프트웨어 분류, 소프트웨어 필터링, API 정보, 기계학습

Abstract Software classification has several applications such as copyright infringement detection, malware classification, and software automatic categorization in software repositories. It can be also employed by software filtering systems to prevent the transmission of illegal software. If illegal software is identified by measuring software similarity in software filtering systems, the average number of comparisons can be reduced by shrinking the search space. In this study, we focused on the classification of Windows executables using API call information and machine learning. We evaluated the classification performance of machine learning-based classifier according to the refinement method for API information and machine learning algorithm. The results showed that the classification success rate of SVM (Support Vector Machine) with PolyKernel was higher than other algorithms. Since the API call information can be extracted from binary executables and machine learning-based classifier can identify tampered executables, API call information and machine learning-based software classifiers are suitable for software filtering systems.

Keywords: software classification, software filtering, API information, machine learning

· 이 논문은 (1) 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(no. NRF-2015R1D1A1A02061946), 또한 (2) "2016년도 정보보호 석사과정 지원사업"으로 수행되었음
· 이 논문은 2016 한국컴퓨터종합학술대회에서 'MS 윈도우 바이너리 분류를 위한 특징 정보와 기계학습 알고리즘에 대한 연구'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 단국대학교 컴퓨터학과
forever811@dankook.ac.kr
limkh120@dankook.ac.kr
†† 종신회원 : 단국대학교 소프트웨어학과 교수
sjcho@dankook.ac.kr
††† 종신회원 : 건국대학교 컴퓨터공학과 교수
schan@kku.ac.kr
†††† 종신회원 : 선문대학교 컴퓨터공학과 교수(Sun Moon Univ.)
young@sunmoon.ac.kr
(Corresponding author임)

논문접수 : 2016년 8월 22일
(Received 22 August 2016)
논문수정 : 2016년 10월 8일
(Revised 8 October 2016)
심사완료 : 2016년 10월 12일
(Accepted 12 October 2016)

정보과학회논문지 제43권 제12호(2016. 12)

1. 서 론

소프트웨어 분류란 서로 다른 프로그램 사이의 유사성을 찾아 유사한 프로그램들을 동일한 카테고리로 지정하는 작업이다. 이 때 유사함의 기준은 명령(instruction) 시퀀스, 실행파일 헤더의 데이터, 프로그램의 기능 등 여러 가지가 있을 수 있다. 소프트웨어 분류의 활용 분야는 취약점 탐지, 저작권 침해 탐지, 악성코드의 분류, 대규모 소프트웨어 보관소(software repository)의 소프트웨어 자동분류 등이 있다. 또한, 소프트웨어 분류는 소프트웨어 필터링 시스템에서 불법 소프트웨어의 원본 소프트웨어 탐색 오버헤드를 줄이기 위해 사용될 수 있다[1].

불법적인 소프트웨어 사용이란 원작자의 허가 없이 무단으로 복사, 배포, 사용하는 행위를 말한다. 소프트웨어의 불법 사용은 저작권자들에게 금전적 또는 정신적인 피해를 주게 된다. 세계 소프트웨어연합(the Software Alliance, BSA)의 자료에 따르면 전 세계적으로 불법 소프트웨어 사용률이 43%이다[2]. 불법 소프트웨어 사용에 의한 피해는 단순히 원작자들에게만 가는 것이 아니다. 소프트웨어를 사용하는 사용자들 또한 불법 소프트웨어에 삽입된 악성코드에 노출될 수 있다. 불법 소프트웨어에 삽입된 악성코드는 사용자 PC를 감염시켜 중요 데이터 삭제, 사용자 암호 탈취, 사용자 PC 좀비화 등에 사용된다.

불법/악성 소프트웨어의 유통을 차단하기 위해 웹사이트나 P2P 등의 온라인 서비스 제공자는 소프트웨어 필터링을 사용할 수 있다. 소프트웨어 필터링은 소프트웨어가 유통되기 전에 식별하여 전송을 차단하는 것을 의미한다. 우리는 프로그램의 해시정보를 이용하는 단순한 소프트웨어 필터링을 구현할 수 있다. 전송 대상 프로그램의 해시정보를 데이터베이스에 등록되어 있는 각 원본 프로그램의 해시정보와 비교하여, 해시정보가 일치하면 대상 프로그램을 불법 소프트웨어로 판단하고 전송을 차단한다. 대상 프로그램이 원본 프로그램을 변조하지 않고 복제한 경우에는, 어렵지 않게 대상 프로그램을 식별할 수 있다. 하지만, 대상 프로그램이 원본 프로그램을 변조했을 경우(예를 들어, 크랙, 악성코드 삽입 등), 대상 프로그램의 해시정보는 원본 프로그램과 동일하지 않기 때문에 필터링 시스템이 대상 프로그램을 식별하지 못할 수 있다. 따라서 불법/악성 소프트웨어를 식별하기 위해서는 프로그램들 사이의 유사도를 측정하여 비교할 필요가 있다. 또한, 대상 프로그램을 식별하기 위해 유사도 측정을 하는 경우에, 대상 프로그램을 데이터베이스에 있는 모든 원본 프로그램과 비교하면 성능 오버헤드가 매우 크다. 따라서 비교 범위를 축소하여 유사도 측정 횟수를 줄이는 방법이 필요하다.

대상 프로그램과 원본 프로그램들 사이의 비교횟수를 줄이기 위해 소프트웨어 분류 기법을 활용할 수 있다. 데이터베이스에는 프로그램들이 카테고리 별로 분류되어 있다. 어떤 프로그램의 전송이 요청되면, 소프트웨어 분류기가 그 프로그램(대상 프로그램)을 분석하여 한 카테고리로 분류한다. 대상 프로그램과 해당 카테고리에 속한 각 프로그램 사이의 유사도를 측정하여 유사한 원본 프로그램을 찾는다. 만약 유사한 원본 프로그램이 발견되면, 대상 프로그램을 식별한 것으로 간주하고 그 전송을 차단한다. 유사한 원본 프로그램이 발견되지 않으면, 다른 카테고리에 속한 각 프로그램과 유사도를 측정하여 유사한 프로그램을 찾는다. 물론 최악의 경우에는 데이터베이스에 속한 모든 프로그램과 비교해야 하지만, 평균 비교횟수는 상당히 감소한다.

소프트웨어 필터링을 위한 소프트웨어 분류 기법은 다음 요구조건을 만족해야 한다. 첫째, 대부분의 불법 소프트웨어는 소스코드 없이 실행파일 형태로 유통되므로 바이너리 실행파일을 분류할 수 있어야 한다. 둘째, 통산 공격자들은 실행파일을 역공학 및 변조하여 배포하므로 변조된 프로그램을 처리할 수 있어야 한다. 셋째, 파일 전송 요청이 매우 빈번하므로 시간 오버헤드가 작아야 한다.

본 논문은 API 정보와 기계학습을 통한 윈도우 소프트웨어의 분류를 연구한다. 본 논문은 윈도우 실행파일 형식인 PE(Portable Executable) 형식의 프로그램에서 API 호출 정보를 추출하여 기계학습에 적합한 형태로 정제한다. 그리고 다양한 기계학습 알고리즘을 적용하여, API 정제 방식과 기계학습 알고리즘에 따른 프로그램 분류 결과를 평가한다. 실험결과, API 호출 빈도보다는 API 호출 여부만을 기계학습 데이터로 사용하는 것이 더 높은 분류 성공률을 보였다. 또한, 각 기계학습 알고리즘은 최적옵션인 경우에 70% 내외의 분류 성공률을 보였으며, 그 중에서 PolyKernel을 사용한 SVM (Support Vector Machine)이 가장 높은 성공률을 보였다.

API 정보와 기계학습을 이용한 소프트웨어 분류는 다음과 같은 이유로 소프트웨어 필터링 시스템에 적합하다. 첫째, 프로그램이 호출하는 API는 소스코드 없이 바이너리 실행파일에서 추출할 수 있는 정보이다. 둘째, 변조 프로그램의 경우, 원본 프로그램과 API 호출 정보가 다소 차이가 날 수 있으나, 전체적으로 큰 차이가 없으므로 기계학습을 통해 식별이 가능하다. 셋째, 기계학습 기반 분류기는 학습에는 다소 많은 시간이 소요될 수 있으나, 분류에는 매우 적은 시간이 소요되므로 분류 시간 오버헤드가 매우 작다.

본 논문의 구성은 다음과 같다. 2장은 관련 연구를 개략적으로 설명한다. 3장에서는 윈도우 실행파일의 특징

정보(API 호출정보) 추출 및 정제 방법을 설명한다. 4장에서는 기계학습 알고리즘을 적용하는 실험 방법을 기술하고 5장에서 실험 결과를 분석한다. 마지막으로 6장에서 결론을 맺는다.

2. 관련연구

2.1 기계 학습(Machine Learning)

기계 학습은 컴퓨터가 데이터를 통해 학습을 하는 것을 말한다. 위키백과에서 학습이란 "직간접적 경험이나 훈련에 의해 지속적으로 지각하고, 인지하며, 변화시키는 행동 변화"라고 말한다. 사람과 마찬가지로 데이터로 컴퓨터를 훈련시켜 원하는 방식으로 작동하도록 하는 것이 기계 학습이라 할 수 있다. 본 연구는 주어진 윈도우 실행파일이 어느 카테고리에 속하는 지 알아내는 문제이고, 컴퓨터에게 각 카테고리에 속하는 샘플 실행파일을 반복하여 보여주면서 카테고리 분류 능력을 기르도록 하려한다. 이와 같이 해답을 알고 있는 학습 데이터로 훈련하는 기계 학습 알고리즘은 지도학습(supervised learning)이라 불리며 대표적인 방법은 LMT(logistic model tree)[3], 랜덤 포레스트(random forest)[4], SVM (support vector machine)[5], MLP(multi-layer perceptron)과 최근에 각광 받고 있는 딥 러닝(Deep Learning) 등이 있다.

2.2 LMT

LMT는 로지스틱 회귀(logistic regression)와 결정 트리(decision tree) 방법을 결합한 지도 학습 알고리즘이다. 로지스틱 회귀는 선형 회귀 분석과 유사하지만 종속 변수가 범주형 데이터일 때도 처리할 수 있는 점이 다르다. LMT는 트리의 마지막 리프(leaf)에서 로지스틱 회귀를 하도록 하고, 결정 트리(모델 트리)를 구성한다.

2.3 랜덤 포레스트

하나의 분류기보다 여러 개 분류기의 결과를 종합하면 더 나은 분류결과를 얻을 수 있다는 아이디어에서 나온 방법이 분류기 앙상블(ensemble)이다. 랜덤 포레스트는 여러 개의 트리 분류기를 만들고 이를 모아서 최종 분류를 한다. 트리 분류기의 독립성을 위해 학습과정에 임의성(randomness)을 주고, 이런 트리 분류기가 여럿 있으므로 포레스트가 되어 이름이 랜덤 포레스트이다. 트리 분류기는 각 노드에서 특징 하나를 임계값과 비교하여 트리의 어느 가지로 진행할지 결정한다. 학습 알고리즘은 이러한 트리를 어떻게 구성하고, 임계값을 어떻게 바꿀지에 대한 것이다.

2.4 SVM

두 개의 클래스에 속한 데이터를 분류하고자 할 때 두 클래스를 구분하는 결정 초평면을 어떻게 학습할 지에 대한 학습 방법이다. 두 클래스를 구별하는 초평면은 두 클래스를 가장 크게 분류하는 즉 마진(margin)을 최대로 하는 것이 가장 일반화 능력이 좋다는 수학 이론에 따라 주어진 학습 데이터에서 서포트 벡터(support vector)를 구한다. 서포트 벡터는 결정 초평면을 구할 때 사용된다. 학습 데이터가 선형으로 구별이 어려울 때 SVM은 커널함수를 이용하여 이러한 비선형 문제를 해결한다. SVM은 입력 특징이 크고, 학습 데이터가 많지 않은 상황에서도 다른 분류기보다 비교적 잘 분류한다고 알려져 있다.

2.5 기계학습을 이용한 소프트웨어 분류

기존 연구로 API를 이용한 윈도우 소프트웨어 분류에 관한 연구[6]는 대상 프로그램을 9개의 카테고리로 나누고 API 빈도 데이터를 특징정보로 채택하였다. 각 카테고리에서 40% 이상의 프로그램들이 참조(또는 '호출')하는 API들만을 사용하였고, 신경망(neural network)와 랜덤 포레스트 기계학습 알고리즘을 이용하였다. 위 연구는 각 카테고리에서 40% 이상의 프로그램들이 참조하는 API만을 고려하여 API가 겹치지 않는 다수의 프로그램들은 분류가 되지 않는다. 위 연구에서 사용된 테스트 집합이 카테고리마다 5개이므로 의미있는 결과를 입증하기에 불충분하다.

3. 윈도우 실행파일 분류를 위한 특징 추출

실행파일의 주요 특징정보에는 API(Application Programming Interface), 문자열(string), 제어흐름그래프(Control Flow Graph, CFG), 호출그래프(Call Graph, CG) 등이 있다. 본 논문에서는 윈도우 실행파일을 분류하기 위한 특징정보로 API 정보를 사용한다. 본 장에서는 API 기반 특징정보를 사용하는 이유, API 정보를 추출하고 정제하는 방법에 대해 기술한다.

3.1 API 기반 특징정보 선정 배경

운영체제(OS)는 프로세스 관리, 메모리 관리, 입출력 관리, 그래픽 처리 등과 같은 시스템 서비스를 제공하기 위해 시스템 콜(system call), 또는 OS API 콜을 사용한다. 대부분의 응용 프로그램들은 빈번하게 API를 사용하며, 이들 API는 다른 명령어·문장으로 대체되기 어렵기 때문에, API는 프로그램을 식별하기 위한 중요한 특성이라고 볼 수 있다[7,8]. 정리하면, 커널 모드 프로그램과 콘솔 프로그램들을 제외한 거의 모든 MS 윈도우 프로그램들이 API를 사용하므로, API는 MS 윈도우 애플리케이션 프로그램들의 주요 특성이다. 또한 API는 위·변조 공격에도 강인하다.

MS 윈도우의 실행파일 형식인 PE(Portable Executable) 포맷뿐만 아니라, 리눅스 실행파일 형식인 ELF (Executable and Linkable Format)포맷, 안드로이드 실행파일 형식인 DEX(Dalvik Executable)포맷 등도

API(자바의 경우 method) 정보를 보유하고 있다. 이에 본 논문에서도 API를 실행파일의 특징정보로 선정하였다. 본 논문에서는 실행프로그램을 실행하지 않고 파일로부터 추출 가능한 정적 특징정보만을 고려한다.

3.2 윈도우 API 정보 추출

본 논문에서는 정적으로 분석 가능한 MS 윈도우 실행파일들을 대상으로 API 정보를 추출한다. 따라서 패킹된 실행파일(실행 압축된 실행파일)이나 암호화된 실행파일은 연구 대상이 아니다.

일반적으로 MS 윈도우의 실행파일(확장자가 EXE)에 사용되는 PE 포맷은 IAT(Import Address Table) 및 텍스트(.text) 섹션에 API에 대한 정보를 포함하고 있다. 해당 실행파일의 IAT로부터는 사용되는 API 이름 정보를, 텍스트 섹션으로부터는 API 참조횟수 정보를 획득할 수 있다. API 참조횟수 정보는 텍스트 섹션에서 jump와 call 명령어에 나타난 정보와 IAT에 저장된 주소 정보를 활용하여 확인할 수 있다. API 정보는 IDApro와 IDAPython 스크립트를 사용하여 추출할 수 있다[9].

본 논문에서는 API 이름 정보와 API 참조횟수 정보를 함께 이용한다. 한 프로그램이 어떤 API를 한번만 참조하는지, 여러 번 참조하는지는 의미가 다르다. 예로, 계산 위주의 프로그램들도 파일 관련 API를 참조하지만, 입출력 위주의 프로그램들은 파일 관련 API를 여러 번 참조한다.

3.3 API 기반 특징정보 정제

실행파일로부터 추출한 모든 API를 그대로 사용하는 것은 다음과 같이 2가지 문제가 있다. 첫째는, 추출한 어떤 API는 실행파일 분류에 적합하지 않을 수 있다. 즉, 효과적인 특징정보 여부를 판단하기 위해서 추출된 API들이 어떤 대표성을 나타내는 지를 확인할 필요가 있다. 즉, 어떤 프로그램에서 추출한 API가 그 프로그램이 속한 카테고리를 대표할 수 있어야 한다. 만약, 한 카테고리 내에 50개의 프로그램이 포함되어 있는데, 어떤 API가 그 카테고리에 속한 1~3개의 프로그램에 의해서만 참조된다면 그 API는 해당 카테고리를 대표하는 특성이 아니라고 볼 수 있다.

둘째는, 각 실행파일에서 추출한 API를 모두 사용하면 기계학습(machine learning)의 복잡도가 높아질 수 있다. 효율적인 자동 분류를 위해서는 기계학습에 사용되는 데이터를 줄일 필요가 있다.

이에, API 정보를 카테고리 별로 정제한다. 즉, 각 카테고리 C_i에 대해, C_i 내에서 $C_i * \theta$개보다 많은 프로그램들이 참조하는 API를 특징정보로 선정한다. 여기서 θ는 미리 정의된 임계치(threshold)이다. 예로, 한 카테고리 내 프로그램 55개이고 임계치가 10%일 경우에는, 6개 이상 프로그램이 참조하는 API들을 선정한다.

4. 실험방법

4.1 실험 테스트 데이터의 수집

실험에 필요한 테스트 데이터는 기존 연구[10]에서 사용한 9개 카테고리에 2개의 카테고리를 추가하였다. 추가된 카테고리는 Game과 Security이다. 각 카테고리마다 55개 프로그램이 포함되어 있다. 따라서 본 논문의 프로그램 집합은 605개의 프로그램(11개 카테고리 × 55개 프로그램)으로 구성된다. 해당 프로그램들은 5개의 오픈소스 사이트[11-15]에서 PE 포맷 중 '.exe' 확장자를 갖는 프로그램을 대상으로 수집하였다. 수집된 실행파일들에는 한 프로그램에 대한 메이저버전과 마이너버전을 포함하기도 한다. 수집한 프로그램 집합을 정리한 내용은 아래 표 1과 같다. 표에서 API 수는 서로 다른 API 개수를 나타낸다. 총 개수는 중복되지 않는 프로그램 수/API 수를 나타낸다. 어떤 API가 Audio Player와 Browser 두 카테고리에서 참조되어도, 총 API 수 산정에서는 한 개로 산정하였다.

표 1 실험 대상 프로그램
Table 1 Program set for experiment

Category	Number of Programs	Number of APIs (before reduction)
Audio Player	55	13,912
Browser	55	5,393
CD Writer	55	2,981
FTP	55	5,175
Image Viewer	55	4,413
Messenger	55	9,479
Text Editor	55	6,615
Video Player	55	7,729
Zip	55	5,669
Game	55	6,678
Security	55	9,677
Total	605	45,358 (except duplicate)

4.2 실험 특징 데이터 추출

4.2.1 API 추출 결과

IDApro와 IDApython 스크립트를 사용하여, 각 카테고리의 프로그램들이 사용하는 API 정보를 추출한 결과가 표 2에 나타나 있다. 표에서 각 행은 카테고리를, 각 열(xp)는 x개의 프로그램을 나타낸다. 표의 각 항목(원소)은 해당 카테고리에 속한 x개의 프로그램(들)에 의해 참조된 API 개수를 나타낸다. 예로, 8,142는 Audio player에 속한 하나의 프로그램에 의해서만 참조된 API 수를 나타낸다. 3,741은 두 개의 프로그램에 의해서만 참조된 API 수를 나타낸다. 그리고 543은 11개 이상의 프로그램에 의해서 참조된 API 개수를 나타낸다. Audio player

표 2 카테고리별 프로그램이 참조한 API 개수
Table 2 Number of APIs used in program (s) by category

x in xp : the number of program(s)

Category	1p	2p	3p	4p	5p	6p	7p	8p	9p	10p	11p or more	Total
Audio Player	8,142	3,741	705	238	123	115	99	90	63	53	543	13,912
Browser	1,124	1,684	1,702	127	95	81	67	58	47	87	321	5,393
CD Writer	1,553	308	187	83	64	35	72	38	40	16	585	2,981
FTP	2,148	993	1,001	122	83	73	60	46	39	38	572	5,175
Image Viewer	2,637	483	287	163	77	62	60	54	31	27	532	4,413
Messenger	5,685	1,380	905	500	234	189	90	73	54	50	319	9,479
Text Editor	4,463	651	344	284	87	59	40	44	49	42	552	6,615
Video Player	3,729	1,220	1,117	385	332	200	199	68	59	41	379	7,729
Zip	3,496	696	409	119	106	46	48	39	40	34	636	5,669
Game	4,922	421	243	82	199	92	96	79	72	47	425	6,678
Security	6,913	967	289	208	137	175	76	65	61	50	736	9,677

의 경우, 13,912개의 API 중 8,142개(약 58.5%)는 하나의 프로그램에 의해서만 참조되므로, 이 API들이 Audio player 카테고리를 대표한다고 말할 수 없고 결과적으로 추출한 API들에 대한 정제가 필요함을 보여주고 있다.

4.2.2 추출된 API 정제 결과

3.3절에서 설명한 것처럼, 카테고리의 대표성을 나타내는 API를 선별하고 기계학습의 속도를 개선하기 위해 API를 정제할 필요가 있다. 본 논문에서는 임계치(Θ)를 7.5%와 10%를 사용하여 정제한다. 추출된 총 API 수 45,358개를 대상으로, 7.5% 정제 후에 2,701개로, 10% 정제 후에 2,008개로 줄일 수 있었다. 임계치 기반으로 API 정보를 정제한 후, 카테고리에 포함된 API의 수가 표 3에 나타나 있다.

표 3 임계치 기반 정제 후 API 수
Table 3 Number of APIs after threshold-based reduction

Category	threshold (Θ)	
	7.5%	10%
Audio Player	1,086	963
Browser	756	661
CD Writer	850	786
FTP	911	828
Image Viewer	843	766
Messenger	1,009	775
Text Editor	873	786
Video Player	1,278	946
Zip	949	843
Game	1,010	811
Security	1,300	1,163
Total (except duplicate)	2,701	2,008

4.3 기계 학습 실험

4.3.1 Weka 소개

본 연구에서 기계 학습을 위한 실험도구로 Weka[16]를 사용한다. Weka는 데이터 마이닝 문제를 풀기 위한 기계 학습 알고리즘들을 자바로 구현한 도구이다. 기계학습의 대부분의 알고리즘과 전처리, 시각화 도구까지 포함하고 있다.

4.3.2 실험에 사용한 기계 학습

실험에 사용한 기계학습 알고리즘은 랜덤 포레스트, SVM, LMT이다. MLP와 딥 러닝은 입력 특징이 큰 경우 매우 오랜 학습 시간이 걸려 제외되었다.

랜덤 포레스트 알고리즘은 포레스트를 구현하는 트리를 몇 개로 할 지가 성능에 가장 큰 영향을 미친다. 포레스트가 작으면 학습은 빠르지만 일반화 능력이 떨어지고, 포레스트가 크면 학습시간이 오래 걸린다. Weka에서 랜덤 포레스트의 numFeatures, numIterations를 변경하면서 테스트하였다.

SVM 알고리즘은 비선형 문제를 풀기위한 커널 함수를 선택하는 것에 따라 성능의 영향이 있다. 또한 결정초평면을 학습시킬 때, 오류를 얼마나 허용하는 지에 따라서도 성능이 달라진다. Weka의 SVM에서 Kernel, cost를 변경하며 테스트하였다. SVM의 Kernel이 Poly-Kernel(다항식)일 경우 exponent(다항식의 차원), RBF Kernel(radial basis function)일 경우 gamma(RBF함수의 모양을 결정)를 변경하며 테스트하였다.

4.3.3 교차 검증

기계학습은 일반적으로 학습 데이터와 테스트 데이터를 분리하여 사용한다. 학습 데이터로만 분류기를 학습시킨 후에, 학습에 사용하지 않은 테스트 데이터로 분류기의 성능을 평가한다. 학습에 사용한 데이터는 당연히

결과가 잘 나오기 때문이다. 그런데 본 실험에 사용하는 데이터의 수는 카테고리 별로 55개이고, 각 데이터의 특징 차원 수는 2,701과 2,008이다. 즉, 학습에 사용할 데이터가 특징 차원의 수와 비교하여 충분하지 않다. 따라서 학습시킨 분류기의 성능을 검증하기 위한 테스트 데이터를 적은 수의 데이터를 나누어 사용할 수 없다. 이런 경우 사용하는 방법이 교차검증(cross validation)이다. k-겹 교차검증은 데이터를 k개의 부분집합을 나누고, k-1개의 부분집합으로 학습시키고, 나머지 한 개의 부분집합으로 분류기의 성능을 측정한다. 이 과정을 서로 다른 부분 집합으로 k번 수행하여 얻은 성능의 평균값을 분류기의 성능으로 하는 것이다. 본 실험은 10-겹 교차검증(ten fold cross validation)을 수행하였다.

5. 실험결과

5.1 기계학습 알고리즘별 데이터 및 실험결과

기계학습에 사용된 데이터 형식으로

① API 존재 여부(또는 '참조여부') 데이터,

② 정규화(normalization)하지 않은 API 호출빈도(또는 참조 횟수) 데이터,

③ 정규화한 API 호출빈도 데이터

총 3가지를 사용하였다. 정규화란, 기계학습을 진행하기 전, API 호출빈도 데이터의 수치가 0~N(특정수치)로 분포가 되는데 이 분포를 0~1 사이의 값으로 변경하는 과정을 말한다. 3가지 데이터를 10번의 교차검증으로 분류한 결과가 표 4~표 7에 나타나 있다. 표에서 분류율은 실험에 사용한 프로그램을 정확한 카테고리로 분류한 수를 실험에 사용한 전체 프로그램의 수로 나눈 비율이다.

랜덤 포레스트의 경우에 numFeatures 값을 0, 190, 300, 500, 1000으로, nemIterations 값을 100, 199, 299, 399, 499, 599로 설정하여 실험하였다. SVM의 경우에는 filter type을 Normalize(정규화), Standardize(표준화), No Normalization/Standardization)으로 설정하고, PolyKernel (다항식)의 exponent 값은 1, 1.3, 그리고 cost는 1, 2, 100으로 설정하여, 실험하였다. SVM RBF Kernel의 경우에는 gamma를 0.005, 0.01, 0.03으로, 그리고 cost는 1, 2, 100으로 설정하여 실험하였다. LMT 알고리즘은 기본 옵션만을 적용하여 분류하였다.

5.2 실험결과 분석

본 논문에서 수집한 프로그램들을 대상으로 한 소프트웨어 분류 실험 결과를 분석한 내용은 다음과 같다. 먼저, 랜덤 포레스트 알고리즘을 사용할 경우에, API 존재여부, 정규화하지 않은 API 호출빈도, 정규화한 API 호출빈도 등의 학습데이터 형식에 거의 영향을 받지 않았다(표 5 참조). 각 데이터에 따른 가장 높은 분류율은

표 4 LMT 사용시 프로그램 분류율

Table 4 Classification Rate of Programs using LMT

option	API existence		API frequency		Normalized API frequency	
	threhold (Θ)		threshold (Θ)		threshold (Θ)	
	7.5%	10%	7.5%	10%	7.5%	10%
default	66.0	67.9	62.6	63.0	62.3	62.5

표 5 랜덤 포레스트 사용 시 프로그램 분류율

Table 5 Classification Rate of Programs using Random Forest

option		API existence		API frequency		Normalized API frequency	
		threhold (Θ)		threhold (Θ)		threhold (Θ)	
numFeatureS	numIterations	7.5%	10%	7.5%	10%	7.5%	10%
0	100	65.1	64.6	64.1	65.0	64.0	65.5
190	199	67.6	68.4	67.4	67.9	67.1	68.3
	299	68.8	68.9	67.1	67.6	68.1	68.6
300	199	68.4	67.6	66.6	68.4	66.6	67.4
	299	68.3	67.8	67.4	68.4	67.6	67.1
500	199	67.8	67.1	66.9	66.4	67.1	68.4
	299	67.1	67.6	67.6	66.9	67.3	68.1
	399	68.8	67.9	67.4	67.3	68.8	68.4
	499	68.6	67.8	67.3	67.6	68.4	67.9
1000	199	66.9	65.8	66.0	66.1	66.3	66.8
	299	67.8	66.3	65.5	66.8	66.4	66.4
	399	68.3	66.1	66.1	66.8	66.6	66.6
	499	68.3	66.1	66.4	66.0	66.3	66.3
	599	68.4	66.1	66.8	66.6	66.3	66.6

표 6 PloyKernel 기반 SVM 사용 시 프로그램 분류율

Table 6 Classification Rate of Programs using PolyKernel based SVM

option			API existence threhold (Θ)		API frequency threhold (Θ)		Normalized API frequency threhold (Θ)	
filter type	exponent	cost	7.5%	10%	7.5%	10%	7.5%	10%
Normalize	1	1	67.9	68.8	63.0	63.1	63.0	63.1
		2	67.6	68.8	62.8	64.0	62.8	64.0
		100	67.6	68.8	64.0	63.6	64.0	63.6
	1.3	1	67.6	68.4	61.5	61.3	61.5	61.3
		2	67.8	68.4	61.7	61.5	61.7	61.5
		100	67.8	68.4	62.5	61.7	62.5	61.8
Standardize	1	1	70.4	72.1	64.3	64.3	64.3	64.3
		2	70.4	72.1	64.3	64.3	64.3	64.3
		100	70.4	72.1	64.3	64.3	64.3	64.3
	1.3	1	15.5	16.0	15.5	14.0	15.2	15.2
		2	15.5	16.0	15.5	14.2	15.2	15.2
		100	15.5	15.7	15.5	14.0	15.2	15.2
No Normal/Standard	1	1	67.9	68.8	53.4	54.2	63.8	63.6
		2	67.6	68.8	53.6	54.2	62.6	64.1
		100	67.6	68.8	53.6	54.2	63.8	64.1
	1.3	1	67.6	68.4	47.8	47.6	61.2	61.2
		2	67.8	68.4	48.4	47.3	60.5	61.8
		100	67.8	68.4	48.4	47.6	60.8	61.7

표 7 RBFKernel 기반 SVM 사용 시 프로그램 분류율

Table 7 Classification Rate of Programs using RBFKernel based SVM

option			API existence threhold (Θ)		API frequency threhold (Θ)		Normalized API frequency threhold (Θ)	
filter type	gamma	cost	7.5%	10%	7.5%	10%	7.5%	10%
Normalize	0.005	1	59.3	58.5	33.9	29.6	33.9	30.7
		2	64.6	64.1	38.0	36.5	38.3	36.2
		100	68.9	69.8	61.5	63.1	61.5	63.1
	0.01	1	60.8	61.3	37.5	34.2	37.9	34.4
		2	63.6	63.8	44.1	41.7	44.3	41.3
		100	65.1	66.3	61.7	63.1	61.7	63.1
	0.03	1	52.1	53.6	42.1	42.3	42.0	42.3
		2	54.7	55.9	51.6	52.1	51.6	52.1
		100	55.0	56.7	60.8	61.8	60.8	61.8
Standardize	0.005	1	45.8	48.6	44.0	47.6	44.3	47.1
		2	46.3	50.1	46.8	49.1	46.9	49.1
		100	46.8	50.6	48.3	50.6	48.3	50.6
	0.01	1	36.7	40.7	41.0	44.1	40.7	44.1
		2	38.2	42.3	42.3	45.1	42.3	45.1
		100	38.2	42.3	43.1	46.3	43.1	46.3
	0.03	1	31.6	33.4	34.0	37.2	45.1	36.7
		2	31.6	33.2	35.5	39.2	54.5	39.2
		100	31.6	33.4	36.0	39.7	62.0	39.7
No Normal/Standard	0.005	1	59.3	58.5	36.7	36.2	33.1	29.3
		2	64.6	64.1	39.2	39.3	40.5	36.7
		100	68.9	69.8	40.5	41.2	63.8	63.3
	0.01	1	60.8	61.3	37.0	37.0	38.5	34.5
		2	63.6	63.8	37.7	37.9	46.0	41.2
		100	65.1	66.3	37.9	38.5	63.6	64.3
	0.03	1	52.1	53.6	34.2	34.4	45.1	44.1
		2	54.7	55.9	35.4	35.7	54.5	53.1
		100	55.0	56.7	35.5	36.4	62.0	64.1

68.9%, 68.4%, 68.8%였다. 주어진 프로그램 집합에 대해, 랜덤 포레스트 알고리즘은 3가지 데이터형식에서 모두 유사한 분류율을 보여준다. 따라서 이 프로그램 집합들의 경우에는, 3가지 학습데이터 형식 중에서 가장 간단한 학습데이터 형식인, 'API 존재여부'를 사용하는 것이 효율적이다.

PolyKernel을 사용하는 SVM에서 API 존재여부 데이터 형식이 API 호출빈도 데이터 형식보다 높은 분류율을 보였다(표 6 참조). 그리고 표준화 필터 유형과 exponent가 1일때 가장 높은 분류율(72.1%)을 보였다. 그러나 exponent가 1.3일 때는 매우 낮은 분류율을 보였다. RBFKernel을 사용하는 SVM에서도 API 존재여부 데이터 형식이 API 호출빈도 데이터 형식보다 높은 분류율을 보였다(표 7 참조). 그러나 "No standardization" 필터 유형과 gamma가 0.005, cost가 100인 경우에 가장 높은 분류율(69.8%)을 보였다.

LMT 알고리즘에서도 API 존재여부 데이터 형식에서 높은 분류율(67.9%)을 보였다(표 4 참조). 이상에서 알 수 있는 것은 API 호출빈도 정보보다는 API 존재여부 정보가, 정보의 양이나 정보의 추출면에서 기계학습 기반의 프로그램 분류에 효과적임을 알 수 있다. API 존재여부 데이터 기반의 기계학습을 통한 분류 시에, SVM 알고리즘의 PolyKernel을 사용하였을 경우에 72.1%의 가장 높은 분류율을 보였다. 이는 본 논문에서 실험한 결과 중에서는 가장 높은 분류율이다. 참고로, API 호출빈도 데이터 기반의 기계학습을 통한 분류 시에, 랜덤 포레스트 알고리즘을 사용하였을 경우에 68.8%의 가장 높은 분류율을 보였다.

6. 결 론

본 논문은 API 호출 정보와 기계학습 알고리즘을 이용한 윈도우즈 실행파일의 분류를 연구하였다. 실험에 사용한 기계학습 도구는 WEKA이며, 다양한 API 호출 정보 정제 방식과 여러 가지 기계학습 알고리즘 및 그 옵션의 조합에 대한 분류 결과를 평가하였다. 실행파일에서 추출한 API 호출 정보는 각 API의 호출 빈도이며, 각 카테고리에서 일정 비율 이상의 프로그램이 호출한 API만을 고려하였다. 또한 이 API에 대해 API 호출 여부, API 호출 빈도, 정규화 한 API 호출 빈도를 기계학습 데이터로 사용하였다. 평가한 기계학습 알고리즘은 RF(Random Forest), SVM(Support Vector Machine), LMT(Logistic Model Tree)이며, 다양하게 옵션을 변경하면서 분류 성공률을 측정하였다.

실험 결과 정제된 API 정보가 윈도우 실행파일 분류에 효과적임을 알 수 있었고, 각 기계학습 알고리즘으로 다양한 실험에서 얻은 최고 분류 성공률은 70% 내외이었다. 각 카테고리의 10% 이상의 프로그램이 호출한 API에 대해 호출 여부만을 학습 데이터로 사용하고, PolyKernel을 사용한 SVM의 분류 성공률이 72.1%로 가장 높았다. RF와 LMT의 경우에는 API 호출 정보 정제 방식과 기계학습 옵션에 따른 분류 성공률이 크게 변하지 않았다. 반면에, SVM의 경우에는 API 호출 정보 정제 방식과 옵션에 따라 분류 성공률이 크게 변하였다. 즉, SVM을 사용할 때는 적절한 옵션의 설정이 매우 중요하였다.

References

[1] D. Kim, Y. Kim, S. Cho, M. Park, S. Han, G. Lee, and Y. Hwang, "An effective and intelligent windows application filtering system using software similarity," *Soft Computing: A Fusion of Foundations, Methodologies and Applications*, Vol. 20, No. 5, pp. 1821-1827, May. 2016.

[2] BSA, *The Compliance Gap: BSA Global Software Survey*, Jun. 2014.

[3] N. Landwehr, M. Hall, and F. Eibe, "Logistic Model Trees," *Machine Learning*, Vol. 59, No. 1, pp. 161-205, 2005.

[4] L. Breiman, "Random Forests," *Machine Learning*, Vol. 45, No. 1, pp. 5-32, 2001.

[5] C. Corinna, and V. Vapnik, "Support-vector networks," *Machine learning*, Vol. 20, No. 3, pp. 273-297, 1995.

[6] Y. Kim, J. Park, S. Cho, Y. Nah, S. Han, M. Park, "Machine learning-based software classification scheme for efficient program similarity analysis," *Proc. of the 2015 Conference on Research in Adaptive and Convergent Systems*, pp. 114-118, 2015.

[7] S. Choi, H. Park, H. I. Lim, and T. Han, "A static API birthmark for Windows binary executables," *Journal of Systems and Software*, Vol. 82, No. 5, pp. 862-873, May 2009.

[8] D. Cho, "Classifying MS Windows Executables Using Strings and APIs," Master's thesis, Dept. of Computer Science & Engineering, Dankook University, Jul. 2016. (in Korean)

[9] IDApro. Available: https://www.hex-rays.com/products/ida/, and IDAPython. Available: https://code.google.com/p/idapython/

[10] D. Lee, J. Park, Y. Hwang and S. Cho, "A Study of Features and Machine Learning Algorithms for Classifying MS Windows Executables," *Proc. of the KIISE Korea Computer Congress 2016*, pp. 102-104, 2016. (in Korean)

[11] FileHippo. [Online]. Available: http://filehippo.com/

[12] Freewarefiles. [Online]. Available: http://www.freewarefiles.com/

[13] Uptodown. [Online]. Available: http://www.uptodown.com/

[14] Oldversion. [Online]. Available: http://www.oldversion.com/

[15] Alternativeto. [Online]. Available: http://alternativeto.net/
[16] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, Ian H. Witten, "The WEKA Data Mining Software: An Update," *ACM SIGKDD Explorations newsletter*, Vol. 11, Issue 1, pp. 10-18, 2009.

조 대 희

2014년 단국대학교 멀티미디어학과 졸업(학사). 2016년 단국대학교 컴퓨터학과 졸업(석사). 관심분야는 컴퓨터 보안, 소프트웨어 지적재산권보호, 악성코드, 기계학습 등

임 경 환

2015년 단국대학교 소프트웨어학과 졸업(학사). 2016년 단국대학교 컴퓨터학과 졸업(석사). 2016년~현재 단국대학교 컴퓨터학과 박사과정. 관심분야는 컴퓨터 보안, 스마트폰 보안

조 성 제

정보과학회논문지
제 43 권 제 11 호 참조

한 상 철

정보과학회논문지
제 43 권 제 3 호 참조

황 영 섭

1989년 서울대학교 컴퓨터공학과 학사
1991년 POSTECH 컴퓨터공학과 석사
1997년 POSTECH 컴퓨터공학과 박사
1997년~2002년 한국전자통신연구원 선임연구원. 2002년~현재 선문대학교 컴퓨터공학과 교수. 관심분야는 딥러닝, 패턴인식, 바이오인포매틱스