KIPS Trans. Softw. and Data Eng.
Vol.7, No.12 pp.451~460 pISSN: 2287-5905

https://doi.org/10.3745/KTSDE.2018.7.12.451

A Study on the Identification of Open Source License Compatibility Violations

Dong-Gun Lee† · Yeong-Seok Seo††

ABSTRACT

Open source software is used in various ways when developing new softwares all around the world. It requires rights and responsibilities as a form of an open source software license. Because the license is a contract between original software developers of the open source software and users, we must follow it and extremely cautious to avoid copyright infringement. In particular, we must verify license compatibility when we develop new software using the existing open source softwares. However, license violation issues always occur and lead to lawsuits so that they are having an adverse effect on the open source software ecosystem. Thus, in this paper, we propose a method, OSLC-Vid, to identify license violations whether compatibility issues exist between open source softwares. The proposed method is verified by the experiments to detect actual license violation cases.

Keywords : Open Source Software, License, License Violation, Compatibility, Association Rule Analysis

오픈 소스 라이선스 양립성 위반 식별 기법 연구

이 동 건† · 서 영 석††

요 약

전 세계적으로 각종 산업 분야를 불문하고 소프트웨어 개발 시 오픈 소스 소프트웨어가 다양하게 활용되고 있다. 이러한 오픈 소스 소프트웨어는 자유로운 사용에 대한 권리뿐만 아니라 그에 따른 책임을 라이선스(license) 형태로 요구한다. 오픈 소스 소프트웨어 라이선스는 오픈 소스 소프트웨어 개발자와 이용자 간의 조건 범위를 명시한 계약이기 때문에 개발자가 규정한 라이선스를 지켜야 하며 이를 위반할 경우에는 저작권 침해가 발생하고, 이에 대한 책임을 지게 된다. 특히, 새로운 소프트웨어 개발 시 기존에 개발된 오픈 소스 소프트웨어를 활용하는 경우, 각 코드의 라이선스가 양립성(compatibility)문제를 발생시키지 않고 서로 호환되는지 확인해야만 한다. 그러나 이러한 철학에 반하여 양립성 문제 관련 사건들이 다수 발생하고 소송으로 이어지기도 하면서 원활한 오픈 소스 소프트웨어 생태계에 악영향을 미치고 있다. 따라서 본 논문에서는 사용하고자 하는 오픈 소스 소프트웨어들 간에 오픈 소스 규칙을 준수하고 라이선스 양립성 문제가 발생하지 않는지 식별할 수 있는 새로운 기법인 OSLC-Vid를 제안한다. 이렇게 제안된 기법은 실제 오픈 소스 소프트웨어를 활용하여 위반사례 식별 성능을 검증하였다.

키워드 : 오픈 소스 소프트웨어, 라이선스, 라이선스 위반, 양립성, 연관 분석

1. 서 론

현대의 소프트웨어 개발 환경은 오픈 소스 소프트웨어(Open Source Software: OSS) 중심으로 활발히 이루어지고 있다. 오픈 소스 소프트웨어란 누구나 해당 소프트웨어를 자유롭게 사용할 수 있는 소프트웨어로서 라이선스 방식을 통해 배포되며, 소스코드가 공개되어 있고, 자유롭게 복제, 수정, 사용, 재배포가 가능한 소프트웨어를 의미한다[1]. 오픈 소스 소프트웨어는 무엇보다 개발한 내용이 대중들에게 공개되는 것뿐만 아니라 수정 및 업그레이드도 자유롭기 때문에 소비자가 원하는 방향으로 나아갈 가능성이 높고, 이에 따라 오픈 소스 소프트웨어는 경제적인 완성도가 높아 소프트웨어 개발자들이 개발 시 가장 먼저 생각하는 분야가 되었다. 실제로 오픈 소스 사용 건수는 2007년 20만 건에서 2015년 140만 건으로 약 7배 정도 급상승하였다[2].

오픈 소스 소프트웨어는 저작권자들이 코드 공개, 자유로운 수정 및 배포 등의 권리를 허가하지만 라이선스(License) 형태로서 그에 따른 책임을 부과한다. 오픈 소스 소프트웨어의 라이선스란 오픈 소스 소프트웨어 개발자와 이용자 간에 이용 방법 및 조건의 범위를 명시한 계약이다. 따라서 오픈

※ 이 성과는 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2017R1C1B5018295).
† 비 회 원 : 영남대학교 컴퓨터공학과 학사과정
†† 종신회원 : 영남대학교 컴퓨터공학과 교수
Manuscript Received : May 24, 2018
First Revision : July 13, 2018
Accepted : August 1, 2018
* Corresponding Author : Yeong-Seok Seo(ysseo@yu.ac.kr)

Table 1. Feature and Duty of Various Licenses [7-9]

	BSD	Apache 2.0	GPL 2.0	GPL 3.0	LGPL 2.0	MPL 1.0	CDDL 1.0	CPL 1.0 /EPL 1.0
Incompatible licenses	GNU GPL	GPL 2.0, LGPL 2.0 etc.	BSD, Apache 2.0, MPL 1.0, CDDL 1.0, CPL 1.0, EPL 1.0	BSD, MPL 1.0, CDDL 1.0, CPL 1.0, EPL 1.0	Apache 2.0	GNU GPL	GNU GPL	GNU GPL
Permission to copy, modify, and release	O	O	O	O	O	O	O	O
Attach copy of license at SW distribution	-	O	O	O	O	O	O	O
Maintain copyright notices or Attribution notices	O	O	O	O	O	O	O	O
Scope of source code distribution	-	-	whole source code	whole source code	derivative works	file	file	module
Allow combined work and other licenses to be deployed	Conditional	O	Conditional	-	O	O	O	O
Notify modifications on amendment	-	-	-	O	O	O	O	O
Allow explicit patent licenses	-	O	-	O	-	O	O	O
Licenses terminated when patent litigation filed	-	O	-	O	-	O	O	O
Restrictions on use of names, trademarks, and trade names	O	O	-	-	-	O	O	-
Disclaimer of warranty	O	O	O	O	O	O	O	O
Limitation of responsibility	O	O	O	O	O	O	O	O

소스 소프트웨어를 이용하기 위해서는 개발자가 규정한 라이선스를 지켜야 하며, 이를 위반할 경우에는 라이선스 위반이 발생하고, 이에 대한 책임을 지게 된다. 따라서 오픈 소스 소프트웨어를 사용하는 소프트웨어 개발자들에게 가장 민감한 부분 중 하나가 바로 이러한 라이선스 문제라고 해도 과언이 아니다[3, 4]. 즉, 소프트웨어를 개발할 경우 기존 만들어진 코드를 재사용하거나 통합할 시, 결합되는 각 코드의 라이선스가 서로 상충될 수 있다. 오픈 소스 소프트웨어 라이선스 철학 및 필수 준수사항에 따라 이러한 라이선스 양립성(Compatibility)은 반드시 피해야 하는데, 이를 위해서는 결합되는 각 코드의 라이선스가 서로 호환되는지 미리 확인해야 한다. 오픈 소스 소프트웨어 사용자가 라이선스에서 요구하는 준수사항을 이행하지 않으면 해당 소프트웨어의 저작권자로부터 저작권법 위반으로 소송을 제기당할 수 있다. 패소할 경우, 소프트웨어 배포가 불가능하며 손해 배상 등의 책임을 부담할 수 있기 때문에 라이선스의 의무사항을 명확하고 구체적으로 이해하여 이런 상황을 예방해야 한다[5, 6]. Table 1은 오픈 소스 소프트웨어에 적용되고 있는 대표적인 라이선스들의 특징 및 의무사항을 보여준다. 많은 라이선스들 중에서 GPL(General Public License) 라이선스가 대표적으로 엄격한 라이선스 중 하나인데, 특히 GPLv2(version 2)의 경우 그 엄격함이 문제가 되어 다른 라이선스(e.g., Apache License)들과 호환이 되지 않고 양립성 문제가 발생할 수 있어, 개발자가 해당 라이선스를 가지는 코드 사용에 많은 어려움을 겪는다[10-12].

따라서 본 논문에서는 소프트웨어 개발자가 사용하고자 하는 오픈 소스 소프트웨어들 사이에 라이선스 양립성 문제를 자동적으로 식별할 수 있는 새로운 기법인 OSLC-Vid (**O**pen **S**ource **L**icense **C**ompatibility **V**iolation **Id**entification)를 제안한다. OSLC-Vid에서는 연관 규칙 분석(Association rule analysis)을 활용하는 동시에 효율을 높이기 위한 선험적 알고리즘(Apriori algorithm)[13]을 사용하여 오픈 소스 코드의 단어 중심으로 패턴을 추출함으로써 라이선스 위반을 식별할 수 있도록 하였다. 제안된 기법은 자동화 도구로 구현하였고 실제 28개의 오픈 소스 소프트웨어를 활용하여 라이선스 위반 식별 성능을 검증하였다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구에 대해 살펴보고 3장에서는 본 논문에서 제시하는 기법을 이해하는데 필요한 배경지식을 간단히 소개한다. 4장에서는 본 연구에서 제안하는 OSLC-Vid에 대해 상세히 제시한다. 5장에서는 제안한 기법을 검증하는 실험 설계 및 결과를 제공하고 마지막 6장에서는 결론 및 향후 연구를 기술한다.

2. 관련 연구

오픈 소스 라이선스 위반 문제를 해결하기 하여 다양한 연구들이 이루어지고 있다.

Lee의 연구[14]에서는 양립성 문제의 발생원인, 특히 GPL과 다른 라이선스간의 양립성을 다루며 이러한 문제의 해결 방법과 그에 대한 한계를 라이선스 차원과 커뮤니티 차원에서 서술하였다. Duan의 연구[15]에서는 거대 오픈 소스 프로젝트에서 양립성과 오픈 소스의 취약점등을 함께 식별하는 방법을 서술한다. 식별을 위하여 검사하고자 하는 오픈 소스 소프트웨어(A)에 대해 해당 소프트웨어에서 사용된 라이선스와 동일한 라이선스로 핵심 기능을 새롭게 구현한 네이티브 라이브러리(Native library)(A')를 제작한다. A와 A'를 이진 유사도 측정 기술(Binary similarity measurement techniques)로 대조하여 같은 형태의 소스코드가 분석된다면 라이선스 위반으로 식별한다. 이러한 연구는 기본적으로 네이티브 라이브러리를 새로이 구성해야 하기 때문에 분석 대상에 따라 시간적인 측면에서 좋은 성능을 기대하기 어렵고 소스 코드 단위로 비교하는 것보다 정확성이 저하될 수 있다. Gordon의 연구[16]에서는 Carneades 시스템의 지원을 받는 오픈 소스 소프트웨어의 양립성 이슈를 다루었고, Kapitsaki의 연구[17]에서는 소프트웨어 라이선스 표준 포맷인 SPDX(Software Package Data Exchange)를 사용한 오픈 소스 소프트웨어에 대해서 SPDX 파일 내에 오픈 소스 양립성 위반을 탐색하는 방법을 다룬다. Gordon과 Kapitsaki의 연구들은 검증하고자 하는 오픈 소스 소프트웨어에 대해 해당 기법들을 적용할 수 있는 특정 파일 양식 체계로 변환이 필요하다는 한계점이 존재한다. 특히, 대형 오픈 소스 소프트웨어의 경우 해당 파일 양식 체계로 변환하는데 많은 시간과 비용이 필요할 수 있다.

한국저작권위원회의 코드아이(CodeEye) 또한 오픈 소스 소프트웨어 라이선스 양립성을 식별하는 서비스를 제공한다[18]. 코드아이 서비스에서는 사용자들이 자신의 소스 코드 파일이나 폴더를 선택해 검사를 요청하면 저작권위원회에서 저작권위원회의 서버에 저장된 오픈 소스 소프트웨어 데이터베이스와 전체 비교해 오픈 소스 소프트웨어 라이선스 준수 여부를 검사하여 검사 보고서를 제공한다. 최근 Synopsys사에 인수된 Black Duck사에서 개발한 오픈 소스 소프트웨어 보안 및 관리 솔루션인 Black Duck Hub의 경우에도 라이선스 위반 여부 검증을 위해 코드아이와 동일한 방법을 사용한다[19]. 그러나 이러한 서비스들은 소스 코드 전체에 대해 관련 데이터베이스와 비교하므로 시간적인 측면에서 비효율적이며 사용자가 의도적으로 코드를 수정할 경우 정확성이 떨어질 수 있다.

3. 배경지식

3.1 연관 규칙 분석(Association rule analysis)

연관 규칙 분석은 특정 사건이 발생하였을 때 함께 빈번하게 발생하는 또 다른 사건의 규칙을 분석하는 기법이다[20]. 데이터의 패턴 및 규칙을 찾아내는 비지도학습(Unsupervised learning) 기법의 하나로서 어떤 행동이 일어났을 때 결과행동이 일어난다는 현상에 대한 표현으로 $X \rightarrow Y$ 의 형식으로 나타낸다.

연관 규칙은 특정 항목 집합(Item set)이 발생하였을 때 또 다른 항목 집합이 발생하는 규칙을 의미한다. 항목 집합은 전체 Item(I) 중에서 가능한 부분 집합으로서 $X_t,\ t = 1, 2, \ldots, 2^{|I|}$의 형식으로 표현한다. 항목 집합의 집합 (The set of item sets)은 I의 부분집합들로 구성된 집합으로 $X = X_1, \ldots, X_k$의 형식으로 나타낼 수 있다. 이러한 연관 규칙 분석은 유통업계에서 상품 소비 분석, 의료업계에서 DNA 패턴 분석 및 증상/질병 관계를 추출 등 각종 산업분야에서 다방면으로 활용되는 데이터 마이닝 기법 중 하나이다.

연관 규칙은 항목 집합들 간에 발생하는 규칙 상황을 얼마만큼 잘 뒷받침해주는가를 평가하기 위해 지지도(Support)와 신뢰도(Confidence)라는 2가지 척도를 사용한다. Fig. 1에서 설명하고 있는 것과 같이 $X \rightarrow Y$ 의 연관 규칙이 있을 때 지지도는 전체 transaction 개수 중 X와 Y가 모두 포함된 개수의 비율이고, 신뢰도는 발생한 전체 X의 개수 중에 Y가 발생한 개수의 비율을 의미한다.

이러한 연관 규칙 기법을 활용하여 본 연구에서는 기존 오픈 소스 소프트웨어들을 트레이닝 셋으로 구성한 후, 전처리 과정을 통해 나온 단어들을 중심으로 연관분석을 수행하여 관련 단어들 사이에 직접적인 경향을 분석하도록 하였다. 이러한 결과를 기반으로 유사한 성격을 가지고 있는 오픈 소스 소프트웨어에도 이러한 경향이 나타나는지 확인하여 라이선스 위반 여부를 식별할 수 있도록 하였다.

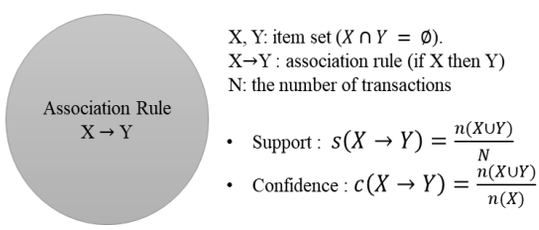

Fig. 1. Measures for Association Rules

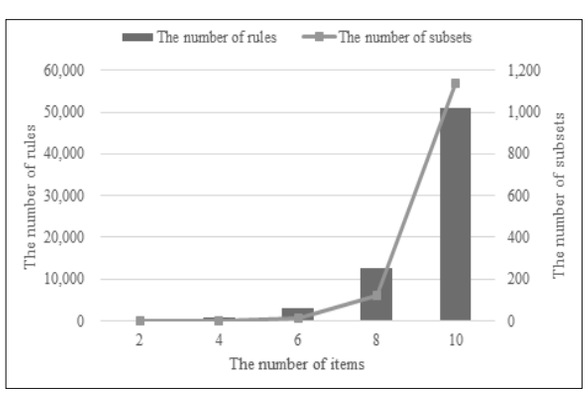

Fig. 2. The Rapid Increase of the Number of Association Rules

3.2 선험적 알고리즘(Apriori algorithm)

선험적 알고리즘은 연관 분석에서 규칙에 증가에 따라 기하급수적으로 증가하는 처리량을 완화시키기 위한 알고리즘

이다. 즉, 선험적 알고리즘은 모든 가능한 항목 집합의 개수를 줄이는 방식으로 처리량을 완화시킨다[21].

오픈 소스 소프트웨어 상에 나타나는 모든 단어들의 집합 $I = i_1, i_2, \ldots, i_k$에서 모든 가능한 부분집합의 개수는 공집합을 제외하고 $2^k - 1$개 이며 가능한 모든 연관규칙의 개수는 $3^k - 2^{k+1} + 1$에 달한다[21]. Fig. 2에서는 이러한 현상을 그래프로 표현하였다. 그래프의 x축은 단어의 개수를 나타내며 왼쪽 y축은 규칙의 개수, 오른쪽 y축은 부분 집합의 개수를 의미한다. 가능한 부분집합의 개수나 연관규칙의 개수가 item이 증가할 때 마다 지수적으로 급증함을 확인할 수 있다.

따라서 모든 항목집합에 대한 지지도를 계산하는 대신에 최소 지지도, 신뢰도 이상의 빈발항목집합(Frequent item set)만을 찾아내서 연관규칙을 계산하는 것이 선험적 알고리즘의 핵심이다. 빈발항목집합 추출의 원칙은 다음과 같다.

(1) 한 항목집합이 빈발(Frequent)하다면 이 항목집합의 모든 부분집합은 역시 빈발항목집합이다. 이 경우 다음 단계로 진행한다(Frequent item sets -> next step).
(2) 한 항목집합이 비빈발(Infrequent)하다면 이 항목집합을 포함하는 모든 집합은 비빈발항목집합이다. 이 경우 가지치기를 통해 잘라낸다(Infrequent item sets-> pruning).

OSLC-Vid에서 많은 수의 오픈 소스 소프트웨어를 분석하면 단어 수와 연관 규칙 수가 기하급수적으로 증가하기 때문에 보다 효율적인 방법으로 성능을 높이기 위하여 연관 규칙 분석 기법 사용시 선험적 알고리즘을 적용하였다. 이를 통해 OSLC-Vid에서는 전처리 과정 후 연관 규칙 분석을 수행할 때 모든 규칙에 대해 계산하지 않아도 되도록 구성하였다.

4. 라이선스 위반 식별 기법

본 논문에서 제안한 라이선스 위반 식별 기법인 OSLC-Vid의 전체적인 접근법은 Fig. 3과 같다. 다음 절부터는 각각의 단계에 대해 보다 구체적으로 소개한다.

4.1 Step 1: Collect open source softwares

Step 1은 OSLC-Vid에서 연관 규칙 분석을 통해 특정 경향을 추출하기 위한 기존 오픈 소스 소프트웨어들을 수집하는 단계이다. 보다 명확한 경향을 추출할 수 있도록 하기 위해, 라이선스 위반 여부를 검사하기 위한 소프트웨어와 유사한 역할과 기능을 수행하는 오픈 소스 소프트웨어들을 우선적으로 수집하여 이들을 트레이닝 셋으로 구성한다. 이렇게 구성된 트레이닝 셋은 다음 단계들을 통해 라이선스 위반 가능성을 파악할 수 있는 특정 경향들을 추출하기 위해 활용되고, 추출된 경향들은 현재 분석하고자 하는 소프트웨어에 적용하여 라이선스 위반 여부를 검출하게 된다.

4.2 Step 2: Conduct preprocessing

Step 2는 Step 1에서 수집한 오픈 소스 소프트웨어들을 효율적으로 분석하기 위해 전처리과정을 수행하는 단계이다. 수집한 오픈 소스 소프트웨어들에 대해 연관 규칙 분석을 적용하기 위해서는 분석 단위를 결정해야 하는데, 빈도수 측면에서 여러 소스코드에 동시에 나타날 확률이 높고 의미적 측면에서 소스코드 개발 시 가장 함축적인 의미를 내포하고 있는 단어 단위로 분석을 수행하였다.

일반적으로 소스코드는 해당 코드에서 사용할 변수를 정

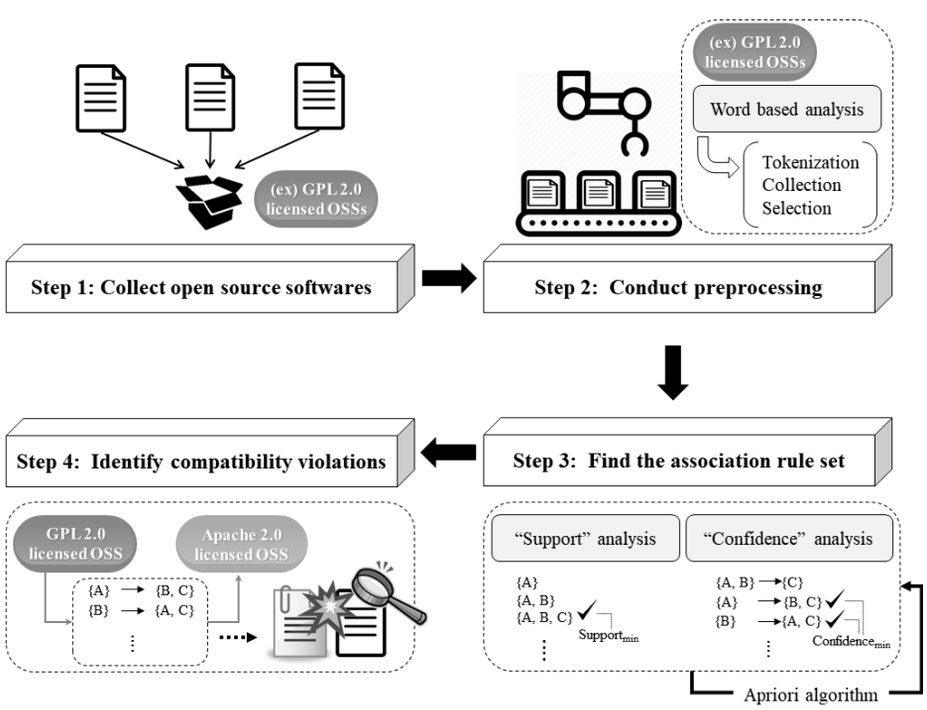

Fig. 3. Overall Approach

의하는 선언부(Declaration part), 선언부의 변수를 이용해 실질적인 기능을 수행하는 실행부(Execution part), 그리고 코드를 설명하는 주석(Comments)으로 이루어져 있다. 이 들 중 실행부의 경우 소스코드의 특징보다는 사용 언어의 특징이 뚜렷하게 드러나고, 주석의 경우 자연어로 이루어져 있어 공통된 단어로부터 연관성을 판별하기 위한 특징을 찾아내기 힘들다. 그에 비해 선언부의 경우 일반적으로 변수나 함수명을 결정할 때 변수나 함수가 실제 어떤 기능을 하는지 의미적으로 내포하는 경우가 많기 때문에 소스코드의 특징을 보다 성공적으로 식별해 낼 수 있다. 따라서 기계적으로 소스코드 전문(Full text)의 선언부에서 단어 중심의 변수들을 인지하고 식별해내기 위해 변수 선언 스타일 중 하나인 스네이크 케이스(Snake case)를 활용하였다. 스네이크 케이스란 변수 선언에 쓰이는 여러 단어들을 연결할 때 언더바(Under bar) '_'를 사용하는 스타일이다. 스네이크 케이스 형태로 작성된 변수는 실제 무슨 역할을 하는지 보다 명확하게 설명해준다. 즉, 한 단어만으로 표현된 'value' 보다 하나 이상의 단어로 표현된 'calc_value'가 보다 직관적인 이해를 도울 수 있다.

이러한 과정을 바탕으로 OSLC-Vid의 전처리 절차는 Fig. 4에 표현한 것과 같이, 토큰화(Tokenization), 수집(Collection), 선별(Selection)의 3단계로 이루어진다. 토큰화 단계에서는 공백, 개행과 같은 구분자(Separator)를 기준으로 문자열을 하나의 토큰(Token)으로 구분한다. 수집 단계에서는 토큰화에 의해 만들어진 토큰 리스트에서 스네이크 케이스에 해당하는 '_' 문자가 포함된 토큰을 수집한다. 마지막으로 선별 단계에서는 이전 단계에서 수집한 스네이크 케이스에 해당하는 토큰들 중 '__init__'같은 소스코드의 특성이 아닌 언어 특성만을 보여주는 토큰들을 제거한다.

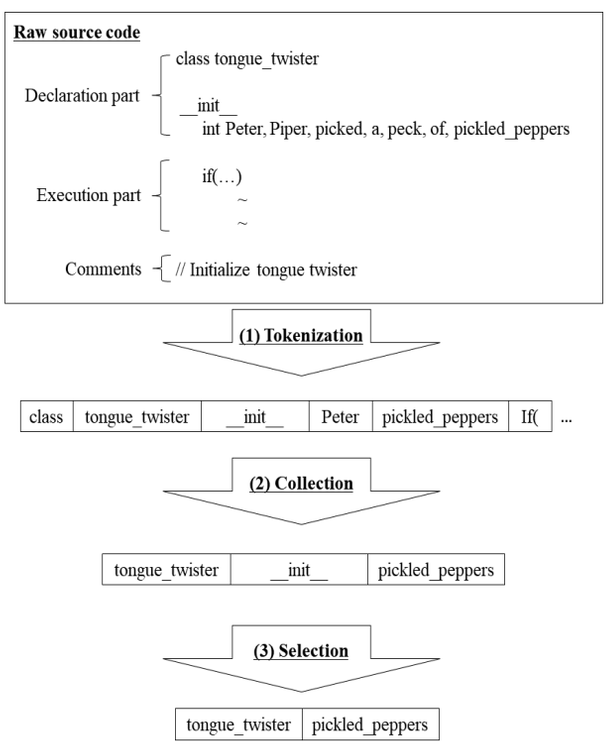

Fig. 4. Example of Preprocessing Stages

4.3 Step 3: Find the association rule set

Step 3은 Step 2에서 전처리된 단어 정보를 이용하여 연관 규칙 분석을 시행하는 단계이다. 같은 라이선스를 사용하고 있는 오픈 소스 소프트웨어들을 충분히 수집하여 트레이닝 셋을 구성하고 이들을 전처리한 다음 연관 규칙 분석을 적용하면, 해당 오픈 소스 소프트웨어들이 함축하고 있는 단어 활용 경향을 추출해 낼 수 있다. 이렇게 추출된 경향(연관 규칙)들은 라이선스 위반이 의심되는 신규 오픈 소스 소프트웨어 검증 시 해당 경향 발생 여부 확인을 위해 활용함으로써 라이선스 위반 여부를 즉각 파악할 수 있도록 해주는 기반이 된다.

구체적으로 Step 3에서는 이전 단계에서 확보한 전처리된 단어 정보에 연관 규칙 분석(선험적 알고리즘)을 적용하고, 설정된 최소 지지도 및 신뢰도 이상의 유효한 연관 규칙들을 추출한다. 전처리된 단어 정보들의 경향을 정확하게 반영해 내기 위하여 1차적으로 최소 지지도 이상의 유효한 항목(단어)집합들을 얻어내고 이러한 결과를 바탕으로 연관 규칙을 구성하여 최소 신뢰도 이상의 연관 규칙들을 추출해낸다.

Fig. 5에서는 최소 지지도 이상의 유효한 항목집합들을 도출하기 위한 과정을 단계적으로 예를 들어 상세하게 표현하고 있다. 처음 Transaction DB 테이블에서 첫 번째 열에는 수집한 오픈 소스 소프트웨어를 ID화 하여 A, B, C, D로 표현하였고, 두 번째 열에는 각 소프트웨어를 전처리 후 확보한 항목들을 나타낸다. 우선, 모든 항목이 존재하는 전체 항목집합 U로부터 1개의 항목만으로 이루어진 항목집합 C_1을 구성하고, 항목집합별 지지도를 계산 한다(예제에서는 간단한 표현을 위해 지지도를

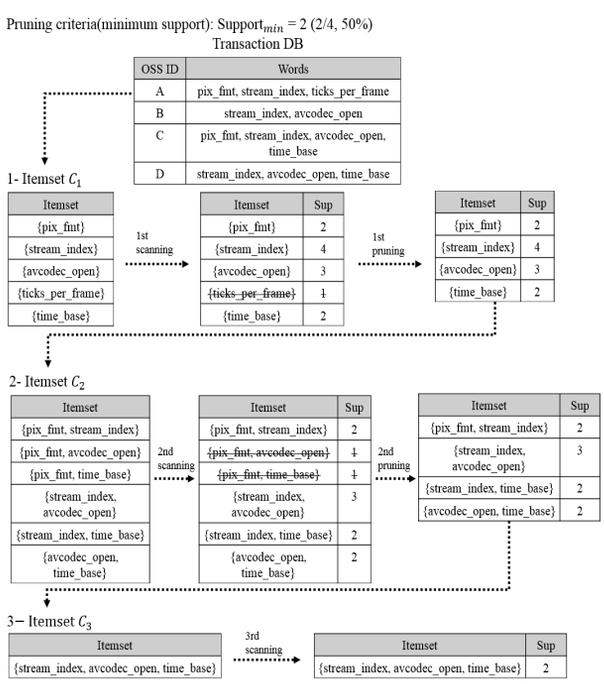

Fig. 5. Example of a Support Based Pruning Process

비율 형태 대신 개수만으로 계산하여 표시). 이렇게 구한 항목 집합별 지지도는 최소 지지도($Support_{min}$=2)와 비교해 이에 미치지 못하는 원소를 제외시킨다. 예제에서는 {ticks_per_frame}이 제외되는데 이렇게 결정된 {ticks_per_frame}는 비빈발항목집합(Infrequent item set)을 발생시키는 요인으로 정해져 U에서 제거된다. 다음으로, 집합의 크기를 1 증가시켜 2개의 빈발항목(Frequent item)으로만 이루어진 항목집합 C_2를 구성하고, 마찬가지 방식으로 항목집합별 지지도 계산 후 최소 지지도에 미치지 못하는 원소들을 제외시킨다. 예제에서는 {pix_fmt, avcodec_open}, {pix_fmt, time_base}가 제외된다. 여기서 'pix_fmt'는 중복되는 단어이므로 비빈발항목집합을 발생시키는 요인이 되어 U에서 제거된다. 마지막으로, 집합의 크기를 1 증가시켜 3개의 빈발항목으로만 이루어진 항목집합 C_3를 구성하고, 최소 지지도에 미치지 못하는 원소들을 제외시킨다. 그 결과 남은 항목집합이 최종 결과가 된다. 예제에서는 {stream_index, avcodec_open, time_base}가 최종적인 항목집합 결과가 되었다.

이러한 결과를 바탕으로 실제 연관 규칙을 생성하여 최소 신뢰도 이상의 유효한 연관 규칙들을 추출한다. Fig. 6에서는 최소 신뢰도 이상의 연관 규칙들을 얻어내기 위한 과정을 예를 들어 구체적으로 표현하고 있다. 이전 단계에서 최소 지지도 탐색 결과 최종적으로 확보한 항목집합 C_3의 원소들에 대해 규칙을 생성한다. 선험적 알고리즘에 따라 후건의 집합의 크기는 1부터 시작하고 전건의 집합은 C_3에서 후건의 원소를 뺀 모든 원소들을 가진다. Fig. 6에 표현되어 있는 것처럼, C_3가 {stream_index, avcodec_open, time_base}이고 후건의 집합이 {stream_index}이라고 할 때 전건의 집합은 {stream_index, avcodec_open, time_base} - {stream_index} = {avcodec_open, time_base}이 된다. 이렇게 확보한 각각의 연관 규칙에 대해서 신뢰도 수치를 구하고 최소 신뢰도($Confidence_{min}$=70%)를 만족하는지 계산한다. Fig. 6의 연관 규칙들 하단에 나타나있는 각각의 신뢰도 수치는 Fig. 5 상황에서 계산한 결과, 어떤 규칙이 최소 신뢰도를 만족하지 못한다면 그 규칙의 전건 집합은 비빈발규칙(Infrequent rule)발생 요인이 되어 앞으로 생성될 규칙의 전건에 들어가지 못하게 된다. 예제에서는 {stream_index, avcodec_open} -> {time_base}가 최소신뢰도를 만족하지 못하여 {stream_index, avcodec_open}가 비빈발 규칙발생 요인이 되고 이를 포함한 {stream_index} -> {avcodec_open, time_base}, {avcodec_open} -> {stream_index, time_base} 규칙들 역시 제외 된다. 이와 같은 방식을 통해 단계수를 높여가면서 더 이상 검사할 규칙이 없을 때까지 진행하게 되고, 최종 남은 모든 연관 규칙들이 수집한 오픈 소스 소프트웨어들의 경향을 보여주는 규칙으로서 활용된다.

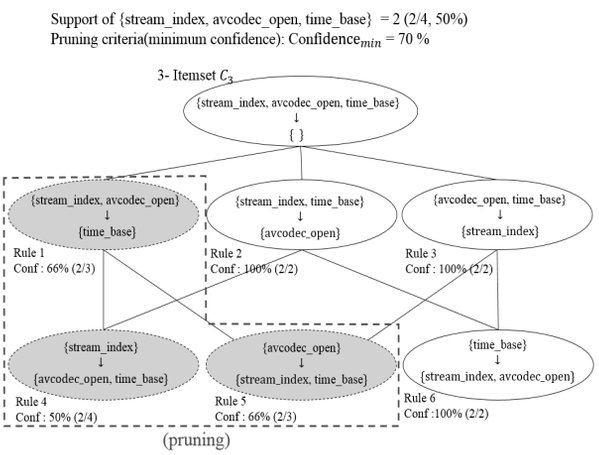

Fig. 6. Example of a Confidence Based Pruning Process

4.4 Step 4: Identify compatibility violations

Step 4에서는 Step 3를 통해 도출된 연관 규칙 집합(R)을 이용하여 라이선스 양립성 위반이 의심되는 신규 오픈 소스 소프트웨어를 검증한다.

검증하고자 하는 신규 오픈 소스 소프트웨어에 대해 전처리 과정을 거쳐 단어 중심의 항목 집합을 구해낸다. 다음으로 Step 3의 결과로 얻은 연관 규칙 집합에 등장하는 모든 규칙들에 대한 전건의 집합이 검증하고자 하는 오픈 소스 소프트웨어에 나타나는지 확인하고 그 중 후건의 집합 역시 등장하는지 체크한다. 즉, 기존 특정 라이선스를 활용하고 있는 오픈 소스 소프트웨어들의 내부 경향을 연관 규칙 분석을 통해 얻어낸 이후, 경향이 반영된 연관 규칙들이 라이선스 양립성 충돌이 의심되는 오픈 소스 소프트웨어에서 탐지되는지(양립성 문제로 함께 사용할 수 없는 특정 라이선스의 코드들이 숨어 있지 않은지 또는 몰래 반영이 되어 있지 않은지) 파악한다.

예를 들어, GPL 라이선스를 기반으로 하는 오픈 소스들을 트레이닝 한 결과가 Fig. 6과 같다면 연관 규칙 집합 R은 pruning한 결과를 제외하고 Fig. 7과 같이 나타낼 수 있다.

R = {(stream_index, time_base) -> (avcodec_open),
(avcodec_open, time_base) -> (stream_index),
(time_base) -> (avcodec_open, stream_index)}

Fig. 7. Example of a Association Rule Set

사용자는 GPL의 라이선스 양립성 요구사항을 위반한 것으로 의심되는 오픈 소스 소프트웨어(소스 코드를 공개하지 않거나 수정 후 이를 공지하지 않음)나 GPL과 충돌이 일어나는 라이선스를 기반으로 하는 오픈 소스 소프트웨어에서 R의 규칙들이 탐지되는지 확인한다. R에는 트레이닝을 수행한 GPL 오픈 소스 소프트웨어들의 경향이 모두 반영되어 있기 때문에, 검증 대상인 오픈 소스 소프트웨어에서 R의 규칙들이 하나 이상 탐지된다면, 이는 해당 오픈 소스 소프트웨어와 라이선스 양립성 충돌로 사용하지 말아야 하는 GPL 라이선스의 코드가 포함되어 있음을 나타낸다. 이러한 결과는 오픈 소스 소프트웨어의 라이선스 양립성 위반 사항에 대한 판단 근거와 함께 오픈 소스 소프트웨어 배포 전 수정 여부를 확인할 수 있는 기회를 제공해준다.

5. 실험 설계 및 결과

5.1 실험 설계

1) OSS 수집

라이선스 위반 여부 검증을 위해, 검증하고자 하는 오픈 소스 소프트웨어와 유사한 기능 및 특성을 가지는 기존 오픈 소스 소프트웨어들을 수집한다. 오픈 소스 소프트웨어의 특성상 다수의 개발자들이 협업을 통해 작업을 진행하므로, 분산 버전 관리를 위한 형상 관리 도구인 깃(Git)과 깃허브(GitHub)에서 오픈 소스 소프트웨어의 소스코드들을 수집하였다. 본 연구의 실험에서는 GPL 2.0 라이선스를 가진 총 18개의 인코딩 소프트웨어들을 모두 검색하여 Table 2와 같은 트레이닝 셋으로 구성하였다. 각 프로젝트별 파일의 개수와 프로젝트 크기가 상당하기 때문에 이를 기반으로 트레이닝을 충분히 수행할 수 있도록 하였다. 테스팅 셋은 GPL 2.0 라이선스와 양립성 충돌을 일으키는 Apache 2.0 라이선스를 가진 오픈 소스 소프트웨어로 구성하였다. 실험을 위해 Apache 2.0 라이선스 양립성을 위반하지 않는(해당 소스 코드에 GPL 2.0 라이선스 코드가 존재하지 않는) 인코딩 소프트웨어 5개와 Apache 2.0 라이선스 양립성을 위반한(해당 소스 코드에 라이선스 충돌을 일으키는 GPL 2.0 라이선스 코드가 존재하는) 인코딩 소프트웨어 5개를 합하여 총 10개로 Table 3과 같이 구성하였다. 인코딩 소프트웨어는 여러 분야들 중에서 법적 이슈가 많고 현재까지도 공개 또는 상업 소프트웨어에 활발히 사용되고 있는 분야인 동시에, 해당하는 오픈 소스의 개수가 많기 때문에 본 연구의 실험에서 활용하였다.

Table 2. Training Set Used in the Experiments

No.	Project	# of files	Project size(MB)
1	ffmpeg	6,722	59.5
2	FFmpegAdapter	332	45.9
3	mpc-qt	123	1.25
4	orion	132	1.35
5	ffmpegthumbnailer	68	1.72
6	Yet Another Process Monitor	202	2.24
7	Gplus	33,588	529
8	Gstreamer	139	1.05
9	JavaAV	69	29.9
10	FFmpegCatapult	43	0.84
11	php7-ffmpeg	55	0.51
12	sgs-server	774	6.09
13	mpv	671	9.37
14	ijkplayer	729	2.92
15	mplayer	1,679	29.8
16	Video-container-switcher	2,965	30.3
17	xvidcore	226	3.53
18	HandBrake	1,515	28.9

Table 3. Testing Set Used in the Experiments

No.	Project	# of files	Project size(MB)	Is the compatibility violated?
A	GSYVideoPlayer	463	79.5	No
B	jitsi-meet	1,042	6.07	No
C	logstash	1,670	19.1	No
D	red5	323	17.1	No
E	hls.js	364	141	No
F	FFmpeg-Compile-For-Android	27	22.8	Yes
G	FFmpeg-Android	149	51.3	Yes
H	FFmpegJni	582	127	Yes
I	ffplayer	6,809	81.5	Yes
J	ExoPlayer-release-v2	1,169	7.66	Yes

2) OSLC-Vid 적용

본 논문에서 제안한 기법을 검증하기 위해 Java기반의 오픈 소스 라이선스 위반 탐색 도구를 구현하여 활용하였다.

경향을 알고자하는 소프트웨어를 한 폴더 안에 정리한 후 해당 폴더에 속한 모든 파일 및 폴더를 재귀적으로(Recursive) 순회했다. 만약 순회하다 파일을 만났을 경우 파일의 이름이 “.c”, “.java”, “.py” 같이 소스 코드로 판단되는 문자열을 포함한다면 그 파일을 분석하도록 하였다. 해당 파일은 4.2의 전처리 과정을 수행하고 결과로서 도출된 단어들은 리스트의 형태로 메모리에 저장해 놓는다. Fig. 8은 본 연구의 실험 수행을 위해 수집한 대표적인 인코딩 오픈 소스 소프트웨어 중 하나인 ‘FFmpeg’를 전처리하여 단어 길이가 긴 순서대로 내림차순 정렬한 예를 보여준다.

본 실험의 트레이닝 셋에 대한 전처리 과정의 결과, 총 1,091개의 단어가 생성되었다. 이러한 단어들을 이용해 연관 규칙을 조합하여 분석 시 선험적 알고리즘을 활용하여 결과

Fig. 8. Partial Results of Preprocessing for FFmpeg

도출의 효율성을 극대화하였다. 연관 규칙 분석 결과 Fig. 9와 같은 일부 규칙들이 도출되었고, Fig. 10과 같이 이를 라이선스 위반이 의심되는 오픈 소스 소프트웨어에 적용하여 이러한 경향의 규칙들이 포함되어있는지 분석한다. 즉, 본 연구의 실험에서는 GPL 2.0 라이선스 기반의 오픈 소스 소프트웨어를 트레이닝하여 얻은 연관규칙들을 GPL 2.0과 라이선스 양립성 위반이 의심되는 Apache 2.0 라이선스 기반의 오픈 소스 소프트웨어를 대상으로 적용해보고 이들이 탐지되는지 확인한다. Fig. 10과 같이 만일 탐지된다면 GPL 2.0 라이선스 기반의 코드가 포함되어 있고 이는 라이선스 양립성을 위반함을 나타낸다.

본 연구의 실험에서는 의미 있는 연관 규칙 도출을 위해 수집한 트레이닝 셋의 오픈 소스 소프트웨어에서 가장 주요한 연관 규칙들이 도출되기 시작하는 최소지지도 60% 기준

$$\{codec\_id, pix\_fmt, size\_t\} \rightarrow \{time\_base\}$$
$$\{codec\_id, pix\_fmt, time\_base\} \rightarrow \{size\_t\}$$
$$\{codec\_id, pix\_fmt\} \rightarrow \{size\_t, time\_base\}$$
$$\{codec\_id, size\_t\} \rightarrow \{pix\_fmt, time\_base\}$$
$$\{pix\_fmt, size\_t\} \rightarrow \{codec\_id, time\_base\}$$
$$\{size\_t, time\_base\} \rightarrow \{codec\_id, pix\_fmt\}$$
$$\{pix\_fmt\} \rightarrow \{codec\_id, size\_t, time\_base\}$$
$$\{codec\_id\} \rightarrow \{pix\_fmt, size\_t, time\_base\}$$
$$\vdots$$

Fig. 9. Partial Results of Association Rules

```
void *opaque;

char codec_name[32];
enum AVMediaType codec_type; /* see AVMEDIA_TYPE_xxx */
enum CodecID codec_id /* see CODEC_ID_xxx */

        ⋮

/**
 * Supported pixel format.
 *
 * Only hardware accelerated formats are supported here.
 */
enum PixelFormat pix_fmt

/**
 * Hardware accelerated codec capabilities.
 * see FF_HWACCEL_CODEC_CAP_*
 */
int capabilities;

        ⋮

size_t av_get_codec_tag_string(char *buf, size_t buf_size, unsigned int codec_tag);

#define FF_LOSS_RESOLUTION  0x0001 /**< loss due to resolution change */
#define FF_LOSS_DEPTH       0x0002 /**< loss due to color depth change */
#define FF_LOSS_COLORSPACE  0x0004 /**< loss due to color space conversion */
#define FF_LOSS_ALPHA       0x0008 /**< loss of alpha bits */
#define FF_LOSS_COLORQUANT  0x0010 /**< loss due to color quantization */
#define FF_LOSS_CHROMA      0x0020 /**< loss of chroma (e.g. RGB to gray conversion) */

        ⋮

AVRational time_base

/* video only */
/**
 * picture width / height.
 * - encoding: MUST be set by user.
 * - decoding: Set by libavcodec.
 * Note: For compatibility it is possible to set this instead of
 * coded_width/height before decoding.
 */
int width, height;
```

Fig. 10. Example of the Rule Application

으로 실험을 수행하였다. 또한 라이선스 위반 여부 검증 대상이 되는 오픈 소스 소프트웨어에 연관 규칙을 적용할 시 전건이 존재하면 후건이 반드시 존재해야 하는 것이 합리적이므로 정확한 판단을 위해 최소 신뢰도는 100%로 설정하여 진행하였다.

5.2 실험 결과 분석

Table 4와 Table 5는 OSLC-Vid를 10개의 실제 오픈 소스 소프트웨어로 구성된 테스팅 셋에 적용한 결과이다. Table 4는 Apache 라이선스 양립성을 위반하지 않은 5개의 오픈 소스 소프트웨어 A, B, C, D, E에 대해 실험한 결과를 보여준다. OSLC-Vid 적용 결과, 라이선스 양립성을 위반하지 않은 모든 소프트웨어에 대해 오진 없이 라이선스 양립성을 위반하지 않음을 식별해 주었다.

Table 5는 Apache 라이선스 양립성을 위반한 5개의 오픈 소스 소프트웨어 F, G, H, I, J에 대해 실험한 결과를 나타낸다. OSLC-Vid 적용 결과, 라이선스 양립성을 위반한 소프트웨어인 G, H, I, J에 대해 실제 라이선스 양립성을 위반했다고 정확히 식별해 주었다. F의 경우, OSLC-Vid를 통해 라이선스 양립성 위반을 식별해낼 수 없었는데, 이를 분석해 본 결과 F는 일반적인 인코딩 오픈 소스 소프트웨어에서 일반적으로 잘 이용하지 않는 부분의 코드(단순 스트리밍 시간이나 누적 카운트 등)를 사용하고 있었고, 또한 그러한 부분들조차도 매우 일부분만 활용하고 있었다. 이러한 특성으로 인해 OSLC-Vid의 연관 규칙 분석이 라이선스를 위반한 일반적인 인코딩 오픈 소스 소프트웨어의 경향과는 다른 것으로 평가하였고, 그 결과 F는 라이선스 양립성을 위반하지 않은 것으로 진단하였다.

Table 4. Verification of OSS that does not Violate Apache License Compatibility

OSS ID	Verification result	Identification of compatibility violations
A	Verification success (not violated)	No license compatibility violation (Not detecting GPL-based OSS in Apache-based OSS)
B		
C		
D		
E		

Table 5. Verification of OSS that Violates Apache License Compatibility

OSS ID	Verification result	Identification of compatibility violations
F	Verification failure (not violated)	No license compatibility violation (Not detecting GPL-based OSS in Apache-based OSS)
G	Verification success (violated)	License compatibility violation (Detecing GPL-based OSS in Apache-based OSS)
H		
I		
J		

본 실험 결과를 기반으로 Table 6과 같은 혼돈행렬(Confusion matrix)을 도출하였고, 분류 및 식별 기법들의 정량적인 성능 평가를 위해 일반적으로 활용되는 정밀도(Precision), 재현율(Recall), 정확도(Accuracy), 그리고 정밀도와 재현율의 조화평균인 F1 Score 값을 산출하였다. 대부분의 경우에서 True인 경우를 True로 식별하고 False인 경우를 False로 정확히 평가하였다. Equation (1), (2), (3), (4)를 통한 OSLC-Vid 성능 평가 결과, 정밀도는 100%, 재현율은 80%, 정확도는 90%, 그리고 F1 Score는 89%를 보여주었다.

Table 6. Confusion Matrix Derived from the Experiments

		Verification by OSLC-Vid	
		True	False
Actual license violations	True	4	1
	False	0	5

$$Precision = \frac{4}{4+0} = 1\,(100\%) \tag{1}$$

$$Recall = \frac{4}{4+1} = 0.80\,(80\%) \tag{2}$$

$$Accuracy = \frac{4+5}{4+1+0+5} = 0.90\,(90\%) \tag{3}$$

$$F1\,Score = 2*\frac{1*0.8}{1+0.8} = 0.89\,(89\%) \tag{4}$$

6. 결론 및 향후 과제

본 논문에서는 오픈 소스 소프트웨어 사용 시 라이선스 양립성 문제 해결을 지원하기 위해, 라이선스 위반 여부를 자동적으로 식별할 수 있는 기법인 OSLC-Vid를 제안하였다. OSLC-Vid에서는 기존 오픈 소스 소프트웨어들을 의미있는 단어 중심으로 전처리하여 효율적인 연관 규칙 분석을 통해 해당 소프트웨어가 가지고 있는 경향과 특성을 연관 규칙 형태로 파악하였고, 이를 바탕으로 라이선스 위반이 의심되는 오픈 소스 소프트웨어에도 같은 경향과 특성이 나타나는지 분석함으로써 실제 위반 여부를 식별 및 검증해낼 수 있었다.

본 연구의 실험 및 실제 활용을 위해 기법이 적용된 자동화 도구를 개발하였고, 이를 통해 제안한 기법의 라이선스 식별 성능을 구체적으로 보여줄 수 있었다.

본 연구를 통해 OSLC-Vid가 성공적인 라이선스 위반 탐색 결과를 보여주었지만 향후 다음과 같은 이슈들에 대해 추가 연구를 진행하고자 한다. 보다 다양한 분야의 오픈 소스 소프트웨어를 OSLC-Vid에 적용하여 결과 일반화 및 충분한 트레이닝을 위한 오픈 소스 소프트웨어 개수 범위에 대해 연구하고자 한다. 또한, 오픈 소스 소프트웨어를 특성별로 분류하여 각 특성별 그룹에 대해 최적의 지지도와 신뢰도 범위를 식별하기 위한 경험적인 실험 연구를 진행하고자 한다. 마지막으로, 오픈 소스 소프트웨어에 작성되어 있는 주석 정보를 추가 활용하여 연관 규칙을 확장하여 보다 향상된 라이선스 양립성 위반 기법을 제안하고자 한다.

References

[1] Open Source software Competency Plaza, OSS definition [Internet], https://www.oss.kr/en_oss_definition.

[2] North Bridge & Black Duck, 2015 the future of Open source [Internet], https://www.slideshare.net/blackducksoftware/2015-future-of-open-source-survey-results.

[3] David Perry, The interesting and complex legal issues of 2017 [Internet], https://opensource.com/article/17/12/best-legal.

[4] Mark Radcliffe, GPLv2 goes to court: More decisions from the Versata tarpit [Internet], https://opensource.com/article/17/12/best-legal.

[5] OpensourceSW License Information Systems, License Introduction [Internet], https://olis.or.kr/en/LicenseIntroduction.do.

[6] Byungil Kim, GPL(General Public License) and Legal issues regarding International Private Law, *Korea Private International Law Journal*, No.14, pp. 80-108, 2008.

[7] Open source software License International System, Open source licenses comparison [Internet], https://olis.or.kr/license/compareGuide.do.

[8] GNU Operating System, Various Licenses and Comments about Them [Internet], https://www.gnu.org/licenses/license-list.en.html#GPLIncompatibleLicenses.

[9] The Apache Software Foundation, For the purposes of being included in an Apache product, which licenses are considered to be similar in terms to the Apache license 2.0? [Internet], https://www.apache.org/legal/resolved.html.

[10] Joseph Morris, Which License Should I Use? MIT vs. Apache vs. GPL [Internet], https://exygy.com/which-license-should-i-use-mit-vs-apache-vs-gpl/.

[11] Jim Lynch, Did Remix OS violate the GPL and Apache licenses? [Internet], https://www.infoworld.com/article/3023538/linux/did-remix-os-violate-the-gpl-and-apache-licenses.html

[12] Lisa Fenn, Artifex and Hancom Reach Settlement Over Ghostscript Open Source Dispute [Internet], http://www.prweb.com/releases/2017/12/prweb14991130.htm.

[13] Ashish Shah, "Association rule mining with modified apriori algorithm using top down approach", *in Proceedings of the 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology*, pp.747-752, 2016.

[14] Chul Nam Lee, "The Research on the Compatibility of Open Source Licenses," *Copyright Quarterly*, Vol.30, No.1, pp.131-152, 2017.

[15] Ruian Duan, Ashish Bijlani, Meng Xu, Taesoo Kim, and Wenke Lee, "Identifying Open-source License Violation and 1-day Security Risk at Large Scale," *in Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp.2169-2185, 2017.

[16] Thomas F. Gordon, "Analyzing open source license compatibility issues with Carneades," *in Proceedings of the*

13th International Conference on Artificial Intelligence and Law, pp.51-55, 2011.
[17] Georgia M. Kapitsaki, FrederikKramer, and Nikolaos D. Tselikas, "Automating the license compatibility process in open source software with SPDX," *Journal of Systems and Software*, Vol.131, pp.386-401, 2017.
[18] CodeEye Service, CodeEye Introduction [Internet], https://olis.or.kr/license/compareGuide.do
[19] Black Duck By Synopsys, Manage Open Source Risks with Black Duck Hub [Internet], https://www.blackduck software.com/solutions/open-source-license-compliance.
[20] Charu C. Aggarwal, "Data Mining," 1st ed., Springer Publishing, ch. 4, pp.93-133, 2015.
[21] Jean-Marc Adamo, "Data Mining for Association Rules and Sequential Patterns," 1st ed., Springer Publishing, ch. 3, pp.33-48, 2001.

이 동 건
https://orcid.org/0000-0001-6792-4572
e-mail : dklee77@ynu.ac.kr
2015년～현 재 영남대학교 컴퓨터공학과 학사과정
관심분야 : Data Mining, Open Source, and Software Defect Prediction

서 영 석
https://orcid.org/0000-0002-5319-7674
e-mail : ysseo@yu.ac.kr
2006년 숭실대학교 컴퓨터학부(학사)
2008년 KAIST 전산학과(석사)
2012년 KAIST 전산학과(박사)
2014년～2016년 한국산업기술시험원(KTL) 선임연구원
2016년～현 재 영남대학교 컴퓨터공학과 교수
관심분야 : Data Mining, Software Modularization, Software Cost Estimation, Software Measurement and Analysis, Mining Software Repositories, and Software Process Improvement