KIPS Trans. Comp. and Comm. Sys.
Vol.9, No.11 pp.273~280
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2020.9.11.273

Warning Classification Method Based On Artificial Neural Network Using Topics of Source Code

Jung-Been Lee†

ABSTRACT

Automatic Static Analysis Tools help developers to quickly find potential defects in source code with less effort. However, the tools reports a large number of false positive warnings which do not have to fix. In our study, we proposed an artificial neural network-based warning classification method using topic models of source code blocks. We collect revisions for fixing bugs from software change management (SCM) system and extract code blocks modified by developers. In deep learning stage, topic distribution values of the code blocks and the binary data that present the warning removal in the blocks are used as input and target data in an simple artificial neural network, respectively. In our experimental results, our warning classification model based on neural network shows very high performance to predict label of warnings such as true or false positive.

Keywords : Software Engineering, Static Analysis, Warning Classification, Artificial Neural Network, Topic Modeling

소스코드 주제를 이용한 인공신경망 기반 경고 분류 방법

이 정 빈†

요 약

자동화된 정적분석 도구는 소스 코드상에 잠재된 결함을 개발자들이 적은 노력으로 빠르게 찾을 수 있도록 도와준다. 하지만 이러한 정적분석 도구는 수정할 필요가 없는 오탐지 경고들을 무수하게 발생시킨다. 본 연구에서는 소스코드 블록의 토픽 모델을 이용한 인공신경망 기반의 경고 분류 기법을 제안한다. 소프트웨어 변경 관리 시스템으로부터 버그를 수정한 리비전들을 수집하고, 개발자들로부터 수정된 코드 블록들을 추출한다. 토픽 모델링을 이용하여 수집된 코드 블록의 토픽 분포 값을 구하고, 코드 블록의 리비전 간 경고들의 삭제 여부를 표현하는 이진데이터를 인공신경망의 입력 값과 출력 값으로 사용하여 심층 학습을 수행한다. 그 결과, 인공신경망 기반의 분류 모델이 높은 예측 성능으로 진성 또는 오탐지 경고를 분류하였다.

키워드 : 소프트웨어공학, 정적분석, 경고 분류, 인공신경망, 토픽모델링

1. 서 론

소스 코드상에 잠재된 결함이나 취약점을 개발 또는 테스트 단계에서 미리 점검하고, 예방하기 위해 정적분석 기법을 사용한다. 소프트웨어의 규모가 커지고, 개발 주기가 점점 더 짧아짐에 따라 개발자들이 소스 코드에서 결함을 수동으로 찾아내기란 현실적으로 불가능하므로 보통 자동화된 정적분석 도구를 사용한다. 정적분석 도구는 개발자를 대신하여 코딩 스타일이나 성능 저하, 또는 자원의 비정상적인 할당 등 코드상에서 잠재적으로 발생할 수 있는 문제점들을 규칙으로 정의하여 자동으로 검사해 준다.

자동화된 정적분석 도구로부터 탐지되는 경고는 상용 소프트웨어의 경우 150만 라인당 3,000개 이상으로 매우 많은 경고들을 생성한다[1]. 더욱이 생성된 경고들 중에서 오탐 경고의 비율이 59%에서 최대 86%까지 차지하기도 하며[2]. 개발자가 이러한 오탐 경고를 식별하고 수정하기 위해서 3~9일을 낭비하기도 한다[4]. 오탐 경고는 프로젝트의 환경이나 코드의 성격상 불가피하게 발생할 수 밖에 없는 경고로써 고치지 않아도 되는 경고이다. 따라서 많은 오탐 경고는 오히려 개발자들에게 부담이 되고, 분석된 결과를 신뢰하지 않거나 무시하고 개발을 진행하는 문제점을 야기한다.

본 연구에서는 자동 정적분석 도구가 탐지하는 수많은 경고들 중에서 개발자들이 반드시 수정해야 할 진성 경고와 오탐 경고를 인공신경망을 이용하여 분류해주는 방법을 제시하고자 한다. 심층 학습을 통해 선정된 인공신경망 모델로부터

† 준 회 원 : 고려대학교 시간생물학연구소 박사후연구원
Manuscript Received : September 16, 2020
Accepted : November 4, 2020
* Corresponding Author : Jung-Been Lee(jungbini@korea.ac.kr)

수정된 소스 코드 블록의 토픽 분포 유형에 따라 어떤 경고를 수정해야 하는지 구분하는 것을 목표로 한다.

2. 관련 연구 및 배경 지식

2.1 정적 코드 분석 경고 분류 기법

버그/이슈 트래킹 시스템 또는 버전 관리 시스템으로부터 발생하는 수많은 리포트의 수정 및 관리를 위해서 상당히 많은 시간과 자원을 필요로 한다. 특히 중대한 결함을 수정하기 위해서 프로젝트 개발의 지연을 초래하기도 한다. 소프트웨어 관리 측면에서 버그 선별(Bug triaging) 절차란 이러한 버그들 중에 어떠한 버그들이 중요하고 그렇지 않으며 또 누가 이것들을 수정해야 하는지 분석하는 작업을 말한다[5-7].

경고 분류 기법은 버그 선별 절차와 비슷하다. 버그 선별을 위한 대부분의 기법들은 기계학습이나 정보 검색, 군집화, 수학적이고 통계적인 모델을 사용하여 버그/이슈 트래킹, 버전 관리 시스템으로부터 보고된 버그들이 중요한 버그인지 아닌지 분류한다. 이러한 기법들은 자동화된 정적 코드 분석 도구로부터 생성되는 무분별한 오탐 경고들을 줄이고 진성 결함들을 식별하도록 도와준다[8-9].

특히 기계학습을 이용한 기법들은 과거에 개발자들이 소스 코드로에서 어떤 경고들을 주로 수정했는지[10], 경고가 보고된 이후로 얼마나 빨리 해당 경고를 고치는지[11], 그리고 경고들이 수정된 코드들의 패턴 즉, 파일, 클래스, 함수 등 식별자의 이름이나 타입이 어떻게 되는지[9] 학습하는 방법으로 고쳐야 하는 경고, 고치지 않아도 되는 경고들을 식별한다. 그러나 이러한 기계학습 기반 기법들은 코드의 성격을 분리하지 않고 일괄적인 경고 분류 규칙을 적용함으로써 코드별로 여전히 많은 오탐 경고들을 발생시키고 있다.

2.2 인공신경망의 개요

인공신경망을 통한 심층 학습은 그 훈련과정에서 입력층(Input Layer)에서 주어지는 값이 은닉층(Hidden Layer)을 거쳐 출력층(Output Layer)에서 계산되는 출력값과의 오차를 최소화되도록 각 층에서 입력되는 값에 곱해지는 가중치와 편향을 계산하고 지속해서 갱신하는 오류역전파(Backpropagation) 알고리즘[12]을 사용한다. Fig. 1은 인공신경망의 각 요소를 나타내고 있다.

입력층의 각 노드에 입력되는 입력값($x_1 \sim x_n$)이 가중치($w_1 \sim w_n$)의 곱과 편향 값이 더해져 다음 층인 은닉층에 모든 노드로 전달된다. 은닉층은 1개 이상이 존재할 수 있으며, 모든 은닉층의 노드로부터 동일하게 가중치와 편향 값

이 계산된다. 은닉층의 각 노드는 바로 이전 층의 모든 노드와 가중치들의 곱을 합한 가중 합을 취함으로써 새로운

입력값이 되고 활성화 함수(Activation function)를 통해 다음 층으로 전달된다. 은닉층을 통해 최종적으로 계산된 값은 다시 출력층의 모든 노드로 전달된다. 활성화 함수 중에서 선형함수(Linear function)는 미분하는 과정에서 항상 상숫값

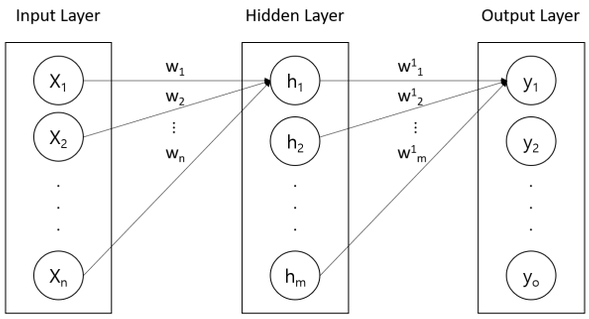

Fig. 1. Artificial Neural Network Structure

이 나오기 때문에 역전파 사용이 불가능하기 때문에 사용하지 않으며 주로 시그모이드(Sigmoid), ReLU 함수를 사용한다.

인공신경망은 출력층을 통해 분류와 회귀분석에 모두 이용할 수 있다. 또한, 출력층의 노드 수는 풀고자 하는 문제에 맞게 적절하게 정할 수 있으며 분류를 위해서는 분류하고 싶은 클래스의 수로 설정한다. 회귀분석을 위해서는 주로 항등함수(Identity function)를 사용하고 분류를 위해서는 소프트맥스 함수(Softmax function)를 사용한다[13].

2.3 토픽 모델링의 개요

토픽 모델링은 정보 검색(Information Retrieval) 방법의 한 종류로써 체계적이지 않은 텍스트 정보를 분석하기 위해 사용되는 기법을 말한다. 토픽 모델링의 입력 값으로 비정형적인 텍스트 문서들이 주어지면 문서에서 주제별로 어떤 단어들이 분포하는지, 문서별로 주제들이 어떤 확률로 분포하고 있는지를 추정해 낼 수 있다.

토픽 모델링을 위해서 LSI(Latent Semantic Indexing)[14, 15]와 LDA(Latent Dirichlet Allocation)[16]가 주로 사용되며 본 연구에서는 새로운 텍스트 문서에 대한 확률값을 계산하기 위해서 LDA 알고리즘을 사용한다. LDA는 텍스트 문서가 주어졌을 경우, 해당 문서들의 중심 토픽의 분포를 찾아 주제별로 클러스터링해주는 비지도 학습(Unsupervised Learning) 알고리즘이다. Fig. 2는 베이지언 네트워크 형태로 그려진 LDA 모형을 나타내고 있다.

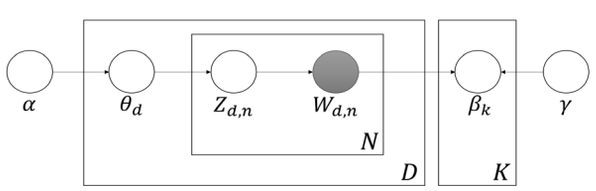

Fig. 2. LDA Model

Fig. 2에서 W는 문서 집합에서 관측된 단어들로써 d 문서의 n번째 단어(=$W_{d,n}$)들로부터 토픽 Z를 추론하고, 문서별 토픽 분포인 θ와 토픽별 단어의 분포인 β를 추정한다. 각 사각형 박스는 단어의 개수 N, 문서의 개수 D, 토픽의 개수 K

만큼 반복된다. α와 η는 LDA 모델 결합확률 분포에서 θ와 β를 우선 제거하고 Z를 추론하기 위해 디레클래분포를 이용하기 위한 초모수 값이다.

LDA는 정의된 토픽의 개수만큼 토픽 군집으로 분류한다. 이 과정을 통해 문서별로 토픽들의 분포를 확률값으로 추론할 수 있고, 토픽별로 단어들의 분포 역시 확률값으로 추론할 수 있다. 하나의 문서에 대한 토픽의 확률값을 토픽 기여도(Topic contribution) 또는 토픽 분포 값이라고 한다. 본 연구에서는 변경된 소스코드 블록이 하나의 문서에 해당하며 이 블록에 대한 k개의 토픽 분포 확률값을 인공신경망 입력층의 각 노드 값으로 사용한다.

토픽의 개수 k는 토픽 모델을 평가하는 방법의 하나인 혼잡도(Perplexity)[16]를 바탕으로 결정한다. 혼잡도는 특정 확률모델이 실제 관측되는 값을 얼마나 잘 예측하는지 평가할 때 사용하는 척도로써 값이 낮을수록 해당 토픽 모델의 학습이 잘되었다고 평가할 수 있다.

3. 데이터 수집 및 입출력 변수 추출

본 장에서는 인공신경망을 구축하여 학습하기 위해 소스코드를 관리하고 이에 대한 변경사항을 추적하기 위한 버전 관리 시스템으로부터 필요한 데이터를 수집하고, 이로부터 인공신경망의 입력층과 출력층에 변수로 활용될 데이터들을 추출하는 과정에 대하여 설명한다.

3.1 데이터 수집

가장 먼저 인공신경망의 입출력 값을 추출하기 위한 기반 데이터로써 소스코드를 수집하는데 Git 저장소[1]로부터 버그 수정과 관련된 커밋 로그 메시지를 검색하여 관련 소스코드를 수집하였다. 커밋 로그를 검색할 때, 단순히 성능 개선이나 기능추가와 같은 목적으로 소스코드를 수정하는 경우는 잠재적 결함을 제거하는 목적과는 관련이 없기 이와 같은 목적의 소스코드는 수집하지 않았다. 각각의 소스코드는 정적 분석을 통해 경고의 변화를 관찰하기 위해서 버그가 수정되기 전의 코드와 수정된 후의 코드를 모두 수집한다.

3.2 변경 코드 블록 식별

코드가 그 역할에 따라 파일별로 모듈화되었어도 해당 파일에서 수정된 코드 블록과 파일의 전체적인 특징은 상이할 수 있다. 예를 들어, GUI와 관련된 파일일지라도 그 안에 수정된 코드 블록은 DB에 접근하는 코드일 수 있다. 따라서, 본 연구에서는 수집된 소스 파일 전체가 아닌 변경된 코드 블록을 추출한다. 이를 위해, 커밋 로그를 참고하여 파일 별로 수정된 코드 블록을 식별한다. Git을 이용한 버전 관리 시스템은, 'Git diff' 명령어를 통해 두 버전 간의 수정된 코드 블록을 식별해 낼 수 있다.

1) RxJava Git 저장소: https://github.com/ReactiveX/RxJava

3.3 입력 변수 추출

인공신경망의 입력 변수로써 2.3절에서 소개한 소스코드 블록의 토픽 분포 값이 사용된다. 토픽 모델링을 위해 3.2절에서 식별한 소스코드 블록 중 버그를 수정하기 전 소스파일의 코드블록을 대상으로 토픽 모델링을 수행한다. 그 이유는 학습된 인공신경망 모델의 입력 값이 버그가 수정되기 이전의 코드 블록에 대한 토픽 분포 값이고 모델은 이에 대한 특정 값을 예측해야 하기 때문이다.

토픽 모델링 수행에 앞서 토픽 모델의 품질 향상을 위해서 간단한 자연어 전처리 과정을 수행하였다. 소스코드 블록 상에서 구두점이나 숫자를 제거하고, 여러 단어로 이루어진 식별자들은 단어별로 분리한다. 또한, 'int', 'double', 'if', 'for', 'assert'와 같은 자바 언어의 예약어들은 모든 자바 소스코드에 공통으로 분포되어 있으므로 토픽별로 코드를 구분하는데 변별력이 없는 단어이다. 따라서, 영어 불용어(Stopword)와 함께 모두 삭제하였다.

토픽 모델링을 위해 파이썬의 gensim 라이브러리에 포함된 LDA 알고리즘을 사용하였으며, α와 γ 초모수 값은 기본값을 사용하였다. 또한, 토픽의 개수 K는 혼잡도 측정을 통해 결정하였다. Fig. 3은 토픽의 개수 k 값의 변화에 따른 혼잡도 그래프를 보여주고 있다.

Fig. 3에서 보는 바와 같이 15개에서 20개 사이의 혼잡도 값이 급격하게 낮아지는 것을 볼 수 있으며 그 이후로도 지속적으로 낮아지지만, 너무 낮은 혼잡도는 사람이 점점 더 사람이 해석하기 어려운 결과를 보인다[17]. 따라서 본 연구에서는 토픽의 개수를 20개로 정의하였다. 앞서 수집된 변경 코드 블록들이 토픽 모델링을 위한 문서 집합으로 사용되고, 최종적으로 각 코드 블록들로부터 20개의 토픽에 대한 각각의 토픽 분포 확률값이 추론되어 인공신경망의 입력 변수로 활용된다.

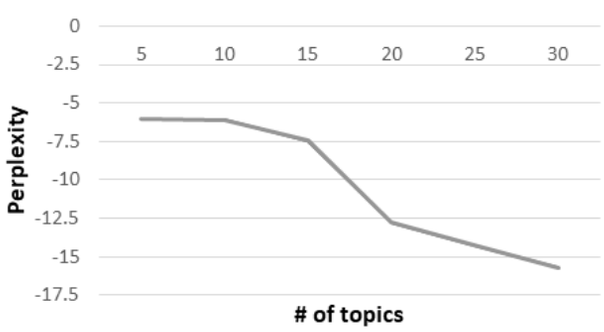

Fig. 3 Perplexity of LDA Model with Each Topic k

3.4 출력 변수 추출

인공신경망의 출력 변수 즉, 예측하고자 하는 변수는 오탐 경고와 진성 경고를 구분하는 이항 변수(0 또는 1)다. 본 연구에서 오탐 경고와 진성 경고를 구분하는 기준은 버그 수정 전후에 발견되는 정적분석 경고의 변화 유무이다. 즉, 버그 수정 전의 코드 블록에서 발견된 특정 경고가 버그를 수정한

후의 코드 블록에서 사라졌다면 진성 결함으로 분류된다. 반대로 버그 수정 전 코드 블록에서 존재했던 결함이 버그 수정 후에도 여전히 존재한다면 개발자들이 중요하지 않아 수정하지 않은 것으로 판단하여 오탐 경고로 분류한다. Fig. 4는 버그 수정 전과 후의 코드 블록에서 경고 변화의 예를 보여주고 있다.

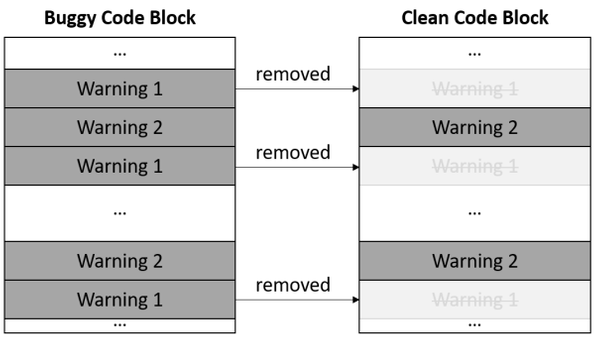

Fig. 4. Warning Changes between Code Blocks before and after Revision for Fixing Bug

Fig. 4의 왼쪽 박스는 버그를 수정하기 전 코드 블록(Buggy Code Block)으로써 경고 1과 2가 각각 3개, 2개 존재하고 있다. 오른쪽 박스는 버그를 수정한 후의 코드 블록(Clean Code Block)으로써 경고 2는 그대로 남아 있지만, 경고 1이 발생한 부분의 코드가 수정되어 삭제된 것을 볼 수 있다. 이 경우 경고 1을 진성 결함(1로 표현)으로 판단하고, 경고 2는 그대로 남아있기 때문에 오탐 결함(0으로 표현)으로 판단한다.

출력 변수의 개수는 정적 분석 도구에서 탐지할 수 있는 경고의 종류에 비례한다. 즉, n개의 종류에 대한 경고를 탐지할 수 있는 정적분석 도구라면 인공신경망의 출력 변수도 n개가 된다. 여러 입력값을 바탕으로 단일 출력값만을 예측하는 기존 기계학습기반 분류 기법들과는 달리 인공신경망의 심층학습을 통해 출력 노드에 n개의 결과를 한꺼번에 도출할 수 있는 특징을 가지고 있다.

4. 결과 및 평가

본 연구에서 제안한 인공신경망 기반 경고 분류 모델의 심층학습을 위해 오픈 소스 프로젝트인 RxJava[2]로부터 관련 데이터를 수집하였다. RxJava는 자바 언어로 리액티브 프로그래밍을 할 수 있는 라이브러리 중에 하나로, 2012년 3월부터 현재까지 263여명의 개발자들에 의해서 활발하게 관리되는 대규모의 오픈소스 프로젝트 중 하나이기 때문에 풍부한 커밋 데이터를 가지고 있어 본 연구의 대상으로 선정하였다. Table 1은 RxJava 프로젝트로부터 수집된 데이터들을 요약하여 보여주고 있다.

2) RxJava 홈페이지: http://reactivex.io/

Table 1. Experimental Results

Data Type	Value
Data collection period	Jan 12, 2013~Aug. 06, 2020
# of commits	941
# of source files (pair)	7,049
# of change blocks	26,990

2013년부터 2020년까지 941개의 커밋이 버그 수정과 관련된 커밋들이었으며 해당 커밋에서 수정된 자바 소스코드 파일이 7,049개이다. 이 파일에서 총 27,990개의 수정된 코드 블록을 식별하였고, 이 블록으로부터 입력 변수(=토픽 분포 값) 및 출력 변수(=진성 및 오탐 경고에 대한 이진값) 추출하여 인공지능망에서 심층학습에 사용하였다. 이중 최근 20%의 파일은 인공신경망 모델의 테스트 데이터로 사용하고, 나머지 과거 80%의 데이터를 모델 평가를 통한 초모수 탐색을 위해 훈련 데이터로 사용하였다.

또한, 본 연구에서 구축한 인공신경망의 입출력 변수 및 값의 형태는 다음과 같다(Table 2).

Table 2. Input and Output Variables of ANN

Layer	Variables	Value Type
Input	Contribution for Topic1	Probability (%)
	Contribution for Topic2	
	...	
	Contribution for Topic19	
	Contribution for Topic20	
Output	Warning1	Binary Number (1: TP Warning 0: FP Warning)
	Warning2	
	...	
	Warning307	
	Warning308	

입력층(Input Layer)의 노드 개수는 총 20개로써 3.3절에서 정의한 토픽의 개수에 비례한다. 각 입력 노드의 변수값 형태는 각 코드 블록에 대한 토픽 분포 확률값이며, 토픽 모델링의 특성상 모든 분포 확률값의 합은 1이므로 일반적으로 인공신경망의 입력층에서 수행되는 정규화 과정이 토픽 모델링에 의해 기본적으로 수행된다. 또한, 출력층(Output layer)의 노드 개수는 정적분석 도구에서 분석 가능한 경고의 종류에 비례한다.

본 연구에서 사용한 정적분석 도구는 PMD 6.27.0 버전으로써 사용되지 않는 변수, 비어있는 catch 블록, 불필요한 객체의 생성 등 개발자들이 흔하게 위반할 수 있는 308개의 결함들[3]을 Rule로 정의를 하여 분석한다. PMD 뿐만 아니라 다른 오픈소스 또는 상용 정적분석도구도 이러한 Rule을 정

3) PMD 6.27.0 버전 기준 Java 코드 룰셋: https://pmd.github.io/latest/pmd_rules_java.html

의하여 소스코드를 분석하기 때문에 다른 정적분석 도구에서 지원하는 Rule을 출력 변수로 사용하여도 된다. 출력층의 변수값 형태는 이진 값으로써 진성 경고(TP: True Positive)를 표현하는 1과 오탐(FP: False Positive) 경고를 표현하는 0이다. 인공신경망 심층학습을 위해 Python 3.7 환경에서 Pytorch 1.5.1 패키지[4]를 사용하였다.

4.1 초모수 탐색 및 모델 평가 척도

인공신경망 훈련의 초모수의 결정을 위한 은닉층의 노드의 개수는 입력 데이터가 n개 일 경우, n/2, n, 2n, 2n+1로 정의[18]하여 탐색한다. 또한, 최적화 함수로 확률적 경사하강법(SGD), 학습을 진행하면서 학습률을 줄여나가는 Adam[19]를 사용하고, 이에 따른 학습률은 0.1, 0.05로 정의한다. 인공신경망 훈련을 위한 손실 함수는 확률적 분류 모델의 학습에 적합한 BCE (Binary Cross-Entropy) 손실 함수를 사용한다. BCE 손실 함수는 인공신경망 출력층의 Sigmoid 활성화 함수 뒤에 Cross-Entropy 함수를 붙인 형태로 정적분석을 통해 보고되는 경고가 진성 경고(1)인지 오탐 경고(0)인지 판단하기 위해 Sigmoid 함수로부터 출력되는 0과 1사이의 실수값을 0 또는 1로 출력하여 평가한다. BCE 손실 함수의 목표는 인공신경망의 입력값 즉, 어떤 코드 블록의 토픽의 확률분포 값들에 대해서 해당 코드 블록에서 어떤 경고가 수정해야 하는 진성 경고이고 수정하지 않아도 되는 오탐 경고인지 확률적으로 예측하는 것이다. 이 예측을 통해 실제 개발자들이 수정한 ground truth 확률값에 최대한 유사하게 학습함으로써 인공신경망의 초모수 값을 조정할 수 있다.

초모수 결정을 위한 예측 모델의 성능을 평가하기 위해 예측값에 대한 RMSE (Root Mean Square Error)를 이용한다. RMSE는 모델이 예측한 값과 실제 값의 차이를 다룰 때 가장 많이 사용하는 척도이며, 모델의 정밀도를 표현한다. RMSE는 모델에서 예측하는 309개의 경고가 가진 확률 값과 실제 값의 차이인 잔차(Residual)들을 하나의 척도로 종합하여 사용한다. 훈련을 통해 가장 작은 RMSE를 갖는 초모수를 선택하였다. Table 3은 은닉층 노드의 수, 최적화 함수, 학습률에 따라 훈련한 RMSE 값을 보여주고 있다. 대체로 Adam 최적화 함수에서, 은닉층의 노드 수가 많아질수록 낮은 RMSE 값을 보였다. 본 연구에서는 최종적으로 인공신경망의 은닉층 노드 개수가 41개, 학습률이 0.05인 Adam 최적화 함수가 가장 낮은 RMSE 값을 보였으므로 이를 최종 예측모델에 적용하였다.

4.2 예측모델 비교 분석 및 고찰

구축된 인공신경망의 성능을 비교하기 위해서 다중회귀모델(Multiple Regression Model)을 구축하고 인공신경망과 동일한 훈련 데이터를 사용하여 학습하였다. 인공신경망의 입력 변수와 마찬가지로 회귀모델의 독립 변수로 20개의 토픽 분포 값을 사용하였고, 종속 변수는 동일한 독립변수에 308개의 출력 변수를 종속변수로 정의하여 학습하였다. 성능 비교를 위한 척도로써 정밀도(Precision)와 재현률(Recall), 그리고 조화평균인 F1-score, 마지막으로 각 모델이 얼마나 정확하게 진성 경고와 오탐 경고를 구분해 냈는지를 평가하는 정확도(Accuracy)를 사용하였다.

Table 3. Comparison of RMSE for ANN Structures

# of Nodes	Optimizer	Learning Rate	RMSE
(20, 10, 309)	SGD	0.1	0.0662
		0.05	0.0661
	Adam	0.1	0.0535
		0.05	0.0521
(20, 20, 309)	SGD	0.1	0.0662
		0.05	0.0661
	Adam	0.1	0.0514
		0.05	0.0503
(20, 40, 309)	SGD	0.1	0.0659
		0.05	0.0662
	Adam	0.1	0.0496
		0.05	0.0481
(20, 41, 309)	SGD	0.1	0.0660
		0.05	0.0661
	Adam	0.1	0.0498
		0.05	0.0478

성능 비교를 위해서 각 모델이 출력하는 모든 예측값과 출력값을 모아 일괄적으로 정밀도, 재현률, 정확도를 측정하였다. 즉, Table 1에서 보는 바와 같이 26,990개의 코드 블록 중, 5,398개(20%)의 토픽 분포 값을 입력으로 각 모델이 예측하는 값을 누적했다. 그 결과, 총 1,662,584(=5,398×308)개의 예측값을 도출했으며, 실제값과의 대조를 통해 성능을 측정하였다. Fig. 5는 본 연구에서 구축한 인공신경망과 다중회귀모델의 성능비교 결과이다.

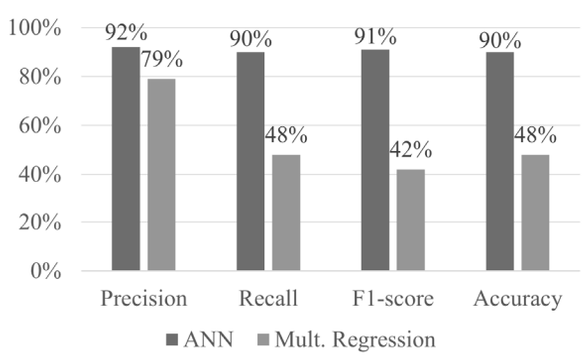

Fig. 5. Comparative Evaluation Result of Full Dataset

Fig. 5와 같이 인공신경망을 이용한 경고 분류 기법이 다중회귀모델에 비해 정밀도와 재현률, 그리고 F1-score에서

4) Pytorch: https://pytorch.org/

각각 13%, 42%, 49% 높은 성능을 보이고 있다. 또한, 진성 경고와 오탐 경고를 구분하는 정확도 면에서도 42% 더 높은 차이를 보인다. 이와 같은 결과가 나타나는 이유는 학습에 사용한 테스트 데이터 상에서 308개의 진성 또는 오탐 경고를 나타내는 출력 변수 중 약 70% (216개)의 값이 모두 0으로 측정되었기 때문이다. 이는 모든 코드 블록에서 해당 경고는 토픽 분포와는 상관없이 수정되지 않았다는 뜻이다. 출력값을 한꺼번에 예측할 수 있는 인공신경망의 경우 이 같은 경우도 학습할 수 있다는 장점이 있지만, 입력 변수의 값에 따라 출력 변수 하나씩을 예측해야 하는 다중회귀모델에서는 학습 자체가 불가능하다. 따라서, 인공신경망이 다중회귀모델보다 더 높은 결과를 보인 것이다. Fig. 6은 0과 1의 값이 모두 존재하여 다중회귀모델도 학습이 가능한 92개의 경고만을 출력변수로 정의하여 다시 두 모델의 성능을 비교하였다.

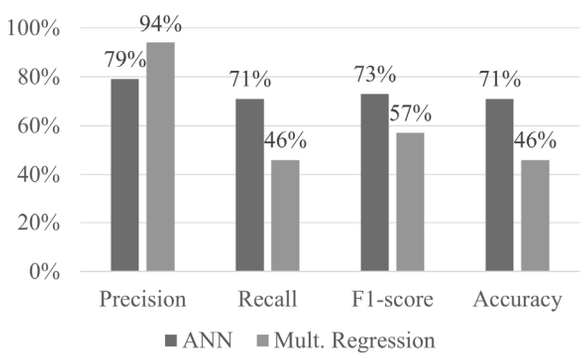

Fig. 6. Comparative Evaluation Result of 101 Dataset

정밀도는 다중회귀모델이 94%로 높지만, 나머지 재현률, F1-score, 정확도에서는 인공신경망 모델이 각각 25%, 16%, 25% 높은 것으로 확인되었다. 인공신경망 모델의 경우, 308개의 출력 변수를 예측할 때(Fig. 5)와 비교하여 F1-score와 정확도가 더 떨어져 다중회귀모델과의 격차가 줄어든 것을 볼 수 있다.

실제로 인공신경망이 비슷한 토픽 분포를 가진 코드 블록에서 유사한 예측을 보여주는지 보여주기 위해서 두 개의 파일을 추출하여 그 결과를 비교해보았다. Fig. 7은 RxJava 프로젝트의 FlowableReduceTest.java의 160-166라인과 FlowableRefCountTest.java의 267-280라인의 토픽 분포값을 보여주고 있다. 그림에서 보다시피 두 코드 블록은 유사한 토픽 분포를 보이고 있는데 특히 Topic 12의 분포 값이 40% 이상인 특징을 가지고 있다. 이는 두 코드 블록은 Topic 12가 가진 단어들을 많이 포함하고 있는 유사한 코드라고 볼 수 있다. 먼저, FlowableReduceTest.java 코드 블록에서 실제로 개발자들이 수정한 경고는 총 92개의 경고들 중, 테스트, 식별자 명명법, 코드 복잡도와 관련된 32가지이다. 즉, 이 파일에서 발생한 버그를 수정하기 위해서 개발자들이 32개의 경고를 수정했다고 볼 수 있다.

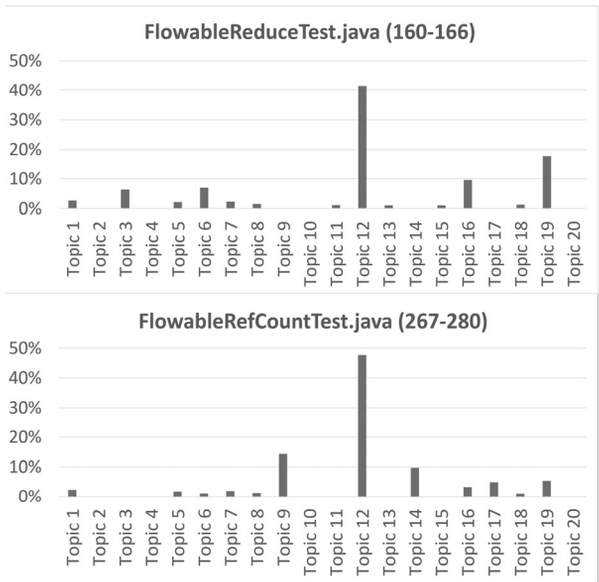

Fig. 7. Topic Distribution between Code Blocks

Table 4는 이 코드 블록의 토픽 분포 값을 인공신경망의 입력 변수로 입력하였을 때, 예측된 진성 및 오탐 경고의 오차 행렬을 보여주고 있다.

Table 4. Confusion Matrix

		Predicted Class	
		Negative (0)	Positive (1)
Actual Class	Negative (0)	58	2
	Positive (1)	1	31

Table 4에서 보는 바와 같이 인공신경망은 실제 32개의 경고 중에서 31개를 진성 경고로 예측하였고, 나머지 60개의 오탐 경고 중에서 58개를 오탐 경고로 예측하였다. 정확도가 약 97% (=89/92)로 매우 높으며, 정밀도와 재현률 역시 매우 높다. 특히, 본 연구에서 구축한 인공신경망은 FlowableRefCountTest.java 코드 블록에서도 정확히 같은 예측 결과를 보여주었다. 이러한 예를 통해, 인공지능망으로 학습된 모델이 다른 코드 블록이라고 할지라도 비슷한 토픽 분포를 갖고 있다면 코드 블록의 진성 및 오탐 경고를 예측하는데 있어서 비슷하거나 동일한 결과를 보임을 확인할 수 있었다.

5. 결 론

본 연구에서는 정적분석 도구를 통해 보고되는 무수히 많은 경고 중에서, 개발자들이 반드시 수정해야 하는 진성 경고와 수정하지 않아도 되는 오탐 경고를 분류하기 위한 인공신경망 기반 경고 분류 기법을 제안하였다. 변경된 코드 블록의 토픽 분포 확률값을 입력 변수로, 버그 수정 전과 후의 코드 블록 안에서 발생된 경고의 변화 여부를 출력 변수로 사용하는 인공신

경망을 구축하여 진성 경고와 오탐 경고를 예측하고, 그 결과를 다중회귀모델과 비교분석하여 모델의 타당성을 검증하였다.

다중회귀모델과 같은 통계기반, 그리고 기계학습 기반 경고 분류 기법의 경우 수정하는 코드가 가지고 있는 특성을 학습 모델에 반영하지 않는 일반적인 모델을 생성하기 때문에 코드의 특성에 따라 오탐 경고가 빈번하게 발생하여 분류 성능이 떨어진다. 본 연구의 인공신경망 모델은 비슷한 토픽 분포를 가진 코드 블록에서 유사한 분류 결과를 예측하였으며, 해당 코드 블록의 진성 및 오탐 경고를 높은 성능으로 분류하였다. 이를 통해 인공신경망을 통한 경고 분류 예측 모델이 기존 경고 분류 기법이 고려하지 못했던 코드의 성격까지 반영함으로써 성능을 보임을 확인할 수 있었다.

향후 연구로는 인공신경망의 은닉층의 개수 및 최적화 알고리즘 등 인공신경망의 성능을 높일 수 있는 다양한 기법 탐색하고, 더 많은 프로젝트에 본 연구 기법을 적용함으로써 그 효과성을 검증할 예정이다. 또한, 정적분석 도구에서 보고되는 경고를 단순하게 진성 및 오탐 경고로 분류하기보다, 이들의 우선순위까지 예측함으로써 개발자들이 오탐 경고로 인해 낭비되는 시간과 비용을 절약할 것으로 예상된다.

References

[1] U. Yüksel and H. Sözer, "Automated classification of static code analysis alerts: A case study," in *2013 IEEE International Conference on Software Maintenance*, pp.532-535, 2013.

[2] L. M. R. Velicheti, D. C. Feiock, M. Peiris, R. Raje, and J. H. Hill, "Towards modeling the behavior of static code analysis tools," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pp.17-20, 2014.

[3] Z. P. Reynolds, A. B. Jayanth, U. Koc, A. A. Porter, R. Raje, and J. H. Hill, "Identifying and documenting false positive patterns generated by static code analysis tools," in *2017 IEEE/ACM 4th International Workshop on Software Engineering Research and Industrial Practice*, pp.55-61, 2017.

[4] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, pp.470-481, 2016.

[5] S. Mani, A. Sankaran, and R. Aralikatte, "Deeptriage: Exploring the effectiveness of deep learning for bug triaging," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp.171-179, 2019.

[6] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Information and Software Technology*, Vol.112, pp.1-17, 2019.

[7] A. Goyal and N. Sardana, "Analytical Study on Bug Triaging Practices," In *Cognitive Analytics: Concepts, Methodologies, Tools, and Applications*, pp.1698-1725, 2020.

[8] Q. Hanam, L. Tan, R. Holmes, and P. Lam, "Finding patterns in static analysis alerts: improving actionable alert ranking," in *Proc. the 11th Working Conference on Mining Software Repositories, ACM*, pp.152-161, 2014.

[9] J. Wang, S. Wang, and Q. Wang, "Is there a golden feature set for static warning identification?: an experimental evaluation," in *Proc. the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp.1-10, 2018.

[10] S. Arai, K. Sakamoto, H. Washizaki, and Y. Fukazawa, "A gamified tool for motivating developers to remove warnings of bug pattern tools," in *2014 6th International Workshop on Empirical Software Engineering in Practice*. pp.37-42, 2014.

[11] K. Liu, D. Kim, T. F. Bissyandé, S. Yoo, and Y. Le Traon, "Mining fix patterns for findbugs violations," *IEEE Transactions on Software Engineering*, 2018.

[12] Paul J. Werbos, "The Roots of Backpropagation: From Ordered Derivatives to Neural Networks and Political Forecasting," New York: John Wiley & Sons, 1994.

[13] G. Ian, B. Yoshua, and C. Aaron, "6.2.2.3 Softmax Units for Multinoulli Output Distributions," in *Deep Learning*, MIT Press, pp.180-184, 2016.

[14] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the Association for Information Science and Technology*, Vol.41, No.6, pp.391-407, 1990.

[15] T. Hofmann, "Probabilistic latent semantic indexing," in *Proc. the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp.50-57, 1999.

[16] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *Journal of Machine Learning Research*, Vol.3, pp.993-1022, 2003.

[17] J. Chang, S. Gerrish, C. Wang, J. L. Boyd-Graber, and D. M. Blei, "Reading tea leaves: How humans interpret topic models," in *Proc. the 23rd Advances in Neural Information Processing Systems*, pp.288-296, 2009.

[18] N. Singh, S. R. Mohanty, and R. D. Shukla, "Short term electricity price forecast based on environmentally adapted generalized neuron," *Energy*, Vo.125, pp.127-139, 2017.

[19] S. Ruder, "An overview of gradient descent optimization algorithms," arXiv preprint arXiv:1609.04747v2, 2016.

이 정 빈

https://orcid.org/0000-0002-8208-0387
e-mail : jungbini@korea.ac.kr
2009년 조선대학교 컴퓨터공학과(학사)
2011년 고려대학교 컴퓨터전파통신공과 (석사)
2020년 고려대학교 컴퓨터학과(박사)
2020년 ~ 현 재 고려대학교 시간생물학연구소 박사후연구원
관심분야 : Software Engineering, Data Mining, Deep Learning, Bioinformatics, Blockchain