논문 2019-1-5 http://dx.doi.org/10.29056/jsav.2019.06.05

트리 기반 컨볼루션 신경망을 이용한 BigCloneBench 개선

박건우*, 홍성문*, 김현하**, 도경구***†

Improvement of BigCloneBench Using Tree-Based Convolutional Neural Network

Gunwoo Park*, Sung-Moon Hong*, Hyunha Kim**, Kyung-Goo Doh***†

요 약

기계 학습을 이용하여 의미가 유사한 코드 클론을 탐지하는 도구의 성능 평가에 빅클론벤치를 많이 활용한다. 하지만 빅클론벤치는 기계 학습에 최적화된 벤치마크가 아니기 때문에 그대로 기계 학습에 사용하면 잘못된 학습 데이터가 만들어질 수 있다. 본 연구에서는 빅클론벤치에서 제공하고 있는 코드 클론 데이터에서 누락된 타입-4 클론을 기계 학습을 이용하여 추가로 찾아 보완함으로써 빅클론벤치를 개선할 수 있음을 실험적으로 밝힌다. 트리 기반 컨볼루션 신경망을 이용한 기계 학습 모델을 사용해서 개선된 데이터를 학습했을 때, 기존의 데이터를 학습했을 때에 비해 기계 학습의 정확도 및 성능이 향상되었음을 확인하였다.

Abstract

BigCloneBench has recently been used for performance evaluation of code clone detection tool using machine learning. However, since BigCloneBench is not a benchmark that is optimized for machine learning, incorrect learning data can be created. In this paper, we have shown through experiments using machine learning that the set of Type-4 clone methods provided by BigCloneBench can additionally be found. Experimental results using Tree-Based Convolutional Neural Network show that our proposed method is effective in improving BigCloneBench's dataset.

한글키워드 : 코드 클론, 클론 검사, 기계 학습, 벤치마크, 합성곱 신경망

keywords : code clone, clone checking, machine learning, benchmark, Convolutional Neural Network

1. 서 론

프로그램에서 구문 구조 또는 실행 기능이 유사한 코드 조각을 코드 클론(code clone)이라고 한다. 일반적으로 코드 클론은 개발자가 소스 코드를 복사-붙여넣기 하거나, 다른 개발자가 같은 기능을 갖는 코드를 구현할 때 발생한다. 결점이 있는 프로그램을 지속해서 생산하는 주요 원인

* 한양대학교 대학원 컴퓨터공학과
** 소프토피아(주)
*** 한양대학교 ERICA 소프트웨어학부
† 교신저자: 도경구(email: doh@hanyang.ac.kr)
접수일자: 2019.06.01. 심사완료: 2019.06.15.
게재확정: 2019.06.20.

중 하나로 잘못된 소스 코드의 재사용을 꼽는다[1]. 또, 소스 코드 불법 복제와 같은 저작권 위배로 법적 분쟁이 발생한 경우, 코드 클론을 탐지하는 기술은 소스 코드의 복제 여부 판단의 신뢰를 좌우하는 중요한 요소이다.

코드 클론을 탐지하는 프로그램은 다양한 방법으로 클론을 찾는다. 가장 먼저 선보인 클론 탐지 방법은 토큰 기반 분석과 트리 기반 분석이다. 토큰 기반 분석 방법은 탐지 시간이 빠르지만, 단순한 토큰의 나열을 비교하기 때문에 구문 구조의 변경에 잘 대응하지 못하는 단점이 있다. 반면 트리 기반 분석은 구문 구조의 정보를 포함하는 트리를 기본 단위로 프로그램을 비교하기 때문에 토큰 기반 분석 방법보다는 정확한 클론을 찾지만, 탐지 시간이 비효율적이다.

소스 코드의 클론을 탐지하는 다양한 연구 방법과 함께 각 연구 결과를 평가하기 위한 성능 측정 방법에 대한 연구도 발표되었는데, 대표적으로 벨론 레퍼런스 코퍼스(Bellon Reference Corpus)[2]와 빅클론벤치(BigCloneBench)[3]가 있다. 벨론 레퍼런스 코퍼스는 2002년에 Stefan Bellon이 주도한 연구그룹이 제시하였으며, 토큰 및 트리 기반 분석의 성능을 가리는 데 사용했다. 벨론 레퍼런스 코퍼스는 기존의 여러 가지 코드 클론 탐지 도구를 사용하여 탐지한 클론의 일부를 표본으로 수집한 클론 모음이다.

빅클론벤치는 2015년에 Jeffrey Svajlenko와 Chanchal Roy가 만든 코드 클론 성능 탐지를 위한 벤치마크이다. 기존의 연구에서는 소스 코드 구문 구조의 유사성에 집중한 데 비해, 소스 코드의 구문 구조가 상이하더라도 유사한 기능을 가진 함수 단위 코드 클론의 집합을 제시하였다. 또, 기존의 벤치마크는 여러 가지 도구를 수행한 결과에서 코드 클론을 구성한 데 비해 눈으로 직접 검증하여 만들었다는 장점을 내세우고 있다.

한편 빅클론벤치는 기계 학습에 그대로 적용하기에 적합하지 않다는 단점을 갖고 있다. 빅클론벤치의 구성상 놓친 클론 쌍이 존재하며, 그에 대한 정답이 제시되어 있지 않기 때문에 기계 학습 모델의 학습 과정 및 성능에 영향을 끼칠 수 있다. 따라서 기계 학습에 앞서 빅클론벤치의 데이터를 개선할 필요가 있다.

본 논문에서는 빅클론벤치의 데이터셋에 놓친 클론 쌍이 존재함을 보이고, 발견한 클론 쌍을 추가해 빅클론벤치를 개선하는 방법을 제안한다.

2. 빅클론벤치

빅클론벤치는 범 프로젝트 자바 리퍼지터리인 IJaDataset-2.0[4](25,000 Java system)을 눈으로 직접 검증하여 구축한 리얼월드 벤치마크이다. 키워드와 소스 코드 패턴 휴리스틱을 이용해 모든 메소드를 기능별로 구분하여 모으고, 각 기능 군내의 메소드를 눈으로 직접 비교해 클론인지 아닌지 판단했다. 클론 검사 도구를 사용하지 않고 만들었기 때문에 기존의 벤치마크와 특정 도구에 편향성을 갖지 않는다는 장점을 갖고 있다. 또, 소스 코드의 구문만으로는 유사성이 낮지만 같은 기능을 수행하는 코드 클론도 많이 포함하고 있어서 기계 학습을 이용하여 의미가 유사한 코드 클론을 탐지하는 도구의 성능 평가에 활용하고 있다[5][6].

2.1 구성

빅클론벤치는 총 73,319개의 메소드를 기능별로 43개의 클론 군으로 모아놓았으며, 클론 쌍의 개수는 총 8,375,313개이다. 각 클론 쌍은 구문 구조뿐만 아니라 기능(실행의미)까지 고려하여 결정하였다.

빅클론벤치는 코드 클론을 크게 네 가지로 표

표 1. 빅클론벤치의 클론 요약
Table 1. BigCloneBench Clone Summary

클론 타입	Type 1	Type 2	Type 3			Type 4
	T1	T2	VST3	ST3	MT3	WT3 / T4
특징	공백 레이아웃 주석	식별자 상숫값	명령문의 추가, 수정, 삭제			기능 유사
유사성	~100%	~100%	90~100%	70~90%	50~70%	0~50%
개수	47,146	4,609	4,191	9,516	38,894	5,818,503
	8,375,313					

1과 같이 분류한다[7].

- Type-1 (T1) : 공백, 레이아웃, 주석이 다르지만 구문 구조가 같은 코드 조각
- Type-2 (T2) : 식별자, 상숫값, 공백, 레이아웃, 주석이 다르지만 구문 구조가 같은 코드 조각
- Type-3 (T3): 구문 구조가 유사하지만, 명령문 수준에서 추가, 수정, 삭제된 명령문이 있는 코드 조각
- Type-4 (T4): 구문 구조가 다르지만 같은 기능을 구현한 코드 조각

구분이 모호한 T3와 T4의 분류를 위해 추가적인 세부 분류기준을 사용한다. 표 1에서 각 클론 타입은 Very-Strongly Type-3 (VST3), Strongly Type-3 (ST3), Moderately Type-3 (MT3), Weakly Type-3 (WT3)를 의미한다. 그리고 각 클론 타입은 구문 구조의 유사도를 기준으로 분류한다.

빅클론벤치 클론 데이터의 전체적인 구성은 그림 1과 같다. 2개 이상의 기능 군에 중복으로 속해있는 229개 메소드를 제외한 나머지 메소드는 모두 하나의 기능 군에만 유일하게 속해있다. 각 기능 군은 IJaDataset-2.0를 대상으로 43개 기능을 대표하는 소스 코드 패턴을 만들고, 이와 일치하는 패턴이 존재하는 메소드를 모아 43개의 기능 군으로 구축하였다. 그리고 각 클론을 눈으로 확인하여, 기능이 동일한 클론으로 판단되는 메소드 쌍은 TP(True Positive)로 구분하고, 기능이 같다고 할 수 없는 클론은 FP(False Positive)로 구분하였다. 이 연구의 목적은 서로 다른 기능 군에 속한 메소드를 비교해 클론으로 인정할 수 있는 쌍이 있는지 밝히고, 빅클론벤치를 개선하는 데 있다.

그림 1. 빅클론벤치 구성
Fig 1. BigCloneBench Structure

2.2 클론 샘플

그림 2는 TP에 속한 클론 쌍의 예시이다. 앞 메소드의 라인 10과 뒤 메소드의 라인 4~8이 구문 구조는 좀 다르지만, 기능적으로 유사하다고 보고 TP로 구분하였음을 알 수 있다. 반면 그림 3은 키워드와 소스 코드 패턴 휴리스틱을 이용해 검출되었지만, 실제로는 기능이 같다고 할 수 없어 FP로 구분한 클론 쌍의 예시이다.

라인	소스코드
1	private static double[][] makeAutoCovarianceMatrice(double[][] vec) {
2	int dim = vec[0].length;
3	double[][] out = new double[dim][dim];
4	double _n = 1. / vec.length;
5	for (int k = 0; k < vec.length; k++) {
6	double[] x = vec[k];
7	for (int i = 0; i < dim; i++) for (int j = i; j < dim; j++) out[i][j] += x[i] * x[j];
8	}
9	for (int i = 0; i < dim; i++) for (int j = i; j < dim; j++) out[i][j] *= _n;
10	for (int i = 0; i < dim; i++) for (int j = i; j < dim; j++) out[j][i] = out[i][j];
11	return out;
12	}

라인	소스코드
1	public Matrix transpose() {
2	Matrix X = new Matrix(n, m);
3	double[][] C = X.getArray();
4	for (int i = 0; i < m; i++) {
5	for (int j = 0; j < n; j++) {
6	C[j][i] = A[i][j];
7	}
8	}
9	return X;
10	}

그림 2. TP 메소드 쌍의 샘플
Fig 2. Sample of TP Method Pair

라인	소스코드
1	protected static void copy(InputStream in, String name, ZipOutputStream out) throws Exception {
2	out.putNextEntry(new ZipEntry(name));
3	int i;
4	while ((i = in.read()) != -1) out.write(i);
5	}

라인	소스코드
1	public static void copyFile2(File srcFile, File destFile) throws IOException {
2	FileUtils.copyFile(srcFile, destFile);
3	}

그림 3. FP 메소드 쌍의 샘플
Fig 3. Sample of FP Method Pair

3. Tree-Based Convolutional Neural Network (TBCNN)

서로 다른 기능 군에 해당하는 메소드를 비교했을 때 클론 쌍이 하나도 없는지 알아보기 위해 2억개가 넘는 모든 메소드 쌍을 눈으로 직접 검증하는 것은 시간상으로 무리가 있다. 그래서 본 연구에서는 기계 학습을 사용하여 유사성이 낮은 메소드 쌍을 배제하는 방법으로, 직접 눈으로 검증할 메소드 쌍을 최소화하였다. 우선 표 1을 보면 클론 쌍의 대부분을 T4가 차지하는 것을 볼 수 있다. T4는 구문 구조가 다르지만 같은 기능을 구현한 코드 조각, 즉 실행의미가 같은 코드를 의미한다. 문자열이나 토큰 기반의 탐지 기법으로는 코드 조각의 실행의미까지 파악하기 어렵기 때문에, 추상 구문 트리 (AST: Abstract Syntax Tree)를 이용해 신경망을 학습시킬 필요가 있다.

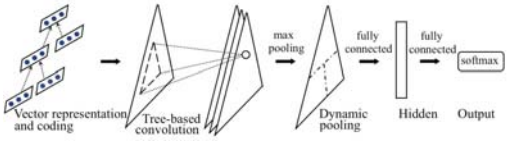

그림 4. TBCNN의 신경망 구성
Fig 4. Organization of TBCNN

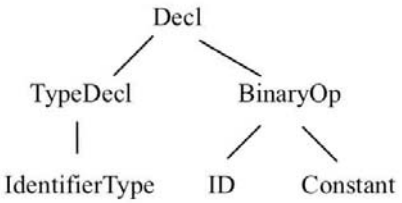

그림 5. 입력 AST 예시
Fig 5. Example of Input AST

베이징 대학 연구그룹이 제시한 트리 기반 컨볼루션 신경망(Tree-Based Convolutional Neural Network, TBCNN)[8][9]은 트리 모양의 데이터를 학습하고 분류하는데 좋은 성능을 보였다. TBCNN은 그림 4와 같이 구성된다. 신경망은 그림 5와 같이 모든 정보를 담지 않고 일부를 생략한 AST를 입력으로 받는다.

그림 6은 실험에 사용하기 위해 TBCNN을 응용해 만든 신경망이다. 메소드를 javalang[10]을 사용해 AST로 만든다. AST의 각 노드를 word2vec 임베딩 모델을 이용해 학습한다[11][12]. 학습한 임베딩 모델을 바탕으로 추상 구문 트리를 벡터화하고 트리 기반 컨볼루션, 풀링 등을 거친다. 두 메소드를 비교하기 위해 두 개의 신경망을 동시에 사용하고, 두 신경망은 매개 변수를 공유한다. Fully Connected Layer의 결과인 두 개의 피쳐 벡터(feature vector)의 차이를 계산한 후 그 값을 기준으로 클론 여부를 판단한다.

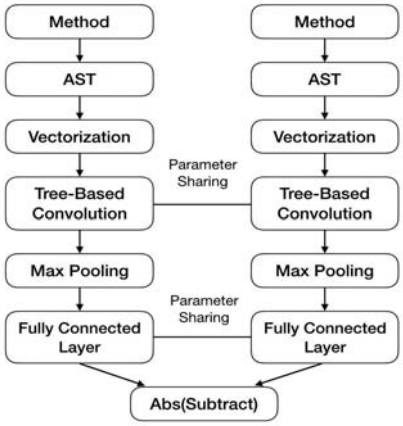

그림 6. 실험에 사용한 신경망
Fig 6. Neural Network for Experiment

4. 실험

빅클론벤치의 데이터 중 가장 적절한 규모라고 판단한 2번 기능 군을 가지고 실험한다.

4.1 데이터셋

2번 기능 군의 기능 명은 'Download from Web'이다. 표 2를 보면, 2번 기능 군에 속한 메소드 개수를 확인할 수 있다. 클론에 해당하는 메소드 906개 중 샘플 메소드 3개가 있다. 이 3개의 샘플 메소드는 빅클론벤치에서 이 기능 군을 구별하는 데 사용한 휴리스틱의 유형이 모두 포함되어 있다. 3개의 샘플 메소드중에서 대표적인 유형은 그림 7에서 확인할 수 있다. 그림 8에서는 FP의 대표적인 2가지 유형을 가진 메소드를 확인할 수 있다.

표 2. 2번 기능 군의 데이터셋
Table 2. Dataset of Functionality 2

메소드 개수	샘플 메소드 개수
906	3

라인	소스코드
1	public static String downloadWebpage1(String address) throws MalformedURLException, IOException {
2	URL url = new URL(address);
3	BufferedReader br = new BufferedReader(new InputStreamReader(url.openStream()));
4	String line;
5	String page = "";
6	while((line = br.readLine()) != null) {
7	page += line + "\n";
8	}
9	br.close();
10	return page;
11	}

그림 7. 2번 기능 군의 샘플 메소드
Fig 7. Sample methods of Functionality 2

라인	소스코드
1	protected synchronized Class<?> loadClass(String name, boolean resolve) throws ClassNotFoundException {
2-8	...
9	String resource = name.replace('.', '/') + ".class";
10	try {
11	URL url = super.getResource(resource);
12	if (url == null) {
13	throw new ClassNotFoundException(name);
14	}
15	File f = new File("build/bin/" + resource);
16	System.out.println("FileLen:" + f.length() + " " + f.getName());
17	InputStream is = url.openStream();
18	try {
19	ByteArrayOutputStream os = new ByteArrayOutputStream();
20	byte[] b = new byte[2048];
21	int count;
22	while ((count = is.read(b, 0, 2048)) != -1) {
23	os.write(b, 0, count);
24-37	...
38	}

라인	소스코드
1	public static String stringOfUrl(String addr) throws IOException {
2	ByteArrayOutputStream output = new ByteArrayOutputStream();
3	URL url = new URL(addr);
4	IOUtils.copy(url.openStream(), output);
5	return output.toString();
6	}

그림 8. 2번 기능 군의 FP 메소드
Fig 8. FP Methods of Functionality 2

4.2 학습

선정한 샘플 메소드 3개가 2번 기능 군을 충분히 대표한다고 보고, 빅클론벤치에 정답이 존재하는 샘플 메소드와 나머지 메소드의 쌍을 가지고 학습한다. 그 데이터는 표 3과 같다. 데이터셋이 작아 학습이 잘 안 될 경우를 고려하여 학습 과정에는 5-Fold 교차 검증 방법을 사용했다.

표 3. 학습 데이터셋
Table 3. Training Dataset

TP 개수	FP 개수
2715	357

학습 결과는 표 4와 같다. 학습된 모델의 성능 지표는 Precision과 Recall의 조화 평균인 F1-score를 이용했다. TP 개수와 FP 개수의 차이가 크기 때문에 가중치를 준 Weighted Avg를 사용하고, 평균적으로 0.94의 F1-score를 갖는 것을 볼 수 있다. F1-score가 1에 근접할수록 높은 정확도를 갖고 있음을 의미한다.

표 4. 학습 결과
Table 4. Training Result

	Precision	Recall	F1-score
True Clone	0.95	0.98	0.97
False Clone	0.85	0.59	0.69
Weighted Avg	0.94	0.94	0.94

4.3 패턴 매칭

학습한 모델을 가지고 다른 기능 군내에 2번 기능 군에 해당하는 코드가 있는지 알아보기 전에 2번 기능 군을 제외한 나머지 메소드를 대상으로 빅클론벤치에서 사용했던 휴리스틱을 간단히 적용한다. 그 결과로 발견한 총 86개의 유사 패턴을 가진 메소드를 대상으로 샘플 메소드 3개와 비교하여 그중에서 클론이라고 할 수 있는 28개의 메소드를 찾았다. 판단 기준은 그림 7의 메소드를 보면, URL에 openStream을 하고 나서 반복문을 통해 데이터를 읽어오는 것을 볼 수 있다. 이러한 유형을 가진 메소드를 클론으로 판단했다. 또한 그림 8에서 볼 수 있는 메소드는 빅클론벤치에 있는 2번 기능 군의 FP 예시이다. URL에서 데이터를 읽어오지만 실제로는 웹에 연결하는 것이 아니거나, IOUtils.copy를 사용하는 경우에는 FP로 판단된다고 생각하고 판단 기준에 참고했다. 얻은 데이터를 가지고 앞서 학습된 모델에 실험했을 때의 결과는 표 5와 같다.

표 5. 86개 메소드 대상 실험 결과
Table 5. Test Result for 86 Methods

	Precision	Recall	F1-score
True Clone	0.53	1.00	0.69
False Clone	1.00	0.57	0.73
Weighted Avg	0.85	0.71	0.71

		모델의 판정	
	쌍 개수	True	False
True Clone	84	84	0
False Clone	174	75	99

4.4 보강 후 학습

4.3에서 얻은 258개의 메소드 쌍을 빅클론벤치에 더해 보강하고, 보강했을 때 얼마나 성능이 개선되는지 확인한다. 보강 후의 데이터는 표 6과 같다. 보강 후의 데이터셋 또한 학습이 잘 안 될 경우를 고려하여 5-Fold 교차 검증 방법을 사용했다.

표 6. 보강한 학습 데이터셋
Table 6. Reinforced Training Dataset

TP 개수	FP 개수
2799	531

학습 결과는 표 7과 같다.

표 7. 보강한 학습 결과
Table 7. Reinforced Training Result

	Precision	Recall	F1-score
True Clone	0.95	0.99	0.97
False Clone	0.90	0.70	0.79
Weighted Avg	0.94	0.94	0.94

4.5 실험 결과 분석

실험 결과 분석에 앞서 word2vec 임베딩 모델과 전체 신경망에 사용된 하이퍼 파라미터값은 다음과 같다.

word2vec

- Number of Features: 30
- Batch Size: 256
- Epochs: 100
- Learning Rate: 0.01
- Hidden Nodes: 100

전체 신경망

- Batch Size: 1
- Epochs: 5
- Learning Rate: 0.001
- Optimizer: Adam
- Loss function: Cross Entropy

데이터를 보강하기 전과 보강한 후의 실험 결과인 표 4와 표 7을 살펴보면 모델이 True Clone을 찾는 경우나 가중 평균적으로는 큰 차이를 보이지 않지만, False Clone을 찾을 때의 Precision과 Recall이 각각 0.85에서 0.90, 0.59에서 0.70으로 눈에 띄게 증가하는 것을 볼 수 있다.

작은 데이터셋임에도 불구하고 84개의 놓친 클론 메소드 쌍을 발견할 수 있었고, 총 258개의 메소드 쌍을 추가해 데이터셋을 개선하는 것이 실제로 모델의 정확도에 영향을 끼치는 것을 확인할 수 있다.

패턴 매칭으로 얻은 86개 메소드 중에 클론이라고 판단하고, 모델 또한 클론이라고 판정한 메소드의 예시는 그림 9에서 확인할 수 있다. 그림 7의 샘플 메소드와 같이 URL에서 openStream과 반복문을 통해 데이터를 받아오는 것을 볼 수 있다.

그림 10은 모델이 True라고 판정했으나 실제로는 클론이 아닌 메소드의 예시이다. 겉보기에는 그림 7의 샘플 메소드와 비슷한 유형을 가진 것으로 보이지만, 그림 8과 같이 FP 유형과 같기 때문에 클론이 아닌 것으로 판단했다. 이러한 유형의 코드를 클론으로 판정하는 것을 보고 이 모델의 한계를 확인할 수 있다. 현재의 기계 학습 모델은 그림 5와 같이 클래스명, 변수명 등의 자세한 정보를 제거한 AST를 입력으로 해서 그림 10의 메소드가 샘플 메소드와 비슷한 유형을 가진다고 보고, 클론으로 판단했다고 볼 수 있다.

5. 결론

본 연구에서는 빅클론벤치의 데이터 내에서 서로 다른 기능 군 내의 메소드를 비교했을 때, 클론으로 판단 가능한 메소드를 추가로 발견할 수 있었다.

발견한 데이터를 기존 빅클론벤치 데이터셋에 추가하고, 기존 데이터셋과 개선된 데이터셋을 가지고 동일한 실험을 했을 때 개선된 데이터셋을 사용한 모델이 더 높은 성능을 갖는 것을 보이고, 놓친 메소드 쌍이 빅클론벤치의 데이터를 기계 학습에 적용할 때 잘못된 학습 데이터의 기반이 된다는 것을 확인했다.

이번 실험은 한 개의 기능 군을 대상으로 샘플 코드만 가지고 간단하게 실험했지만, 현재 모델이 가지는 한계를 개선하고 나머지 전체 기능 군에서 놓친 클론 쌍을 보완하여 빅클론벤치를 개선하는 것을 추후 과제로 남기고자 한다.

라인	소스코드
1	@Override
2	public synchronized File download_dictionary(Dictionary dict, String localfpath) {
3	abort = false;
4	try {
5	URL dictionary_location = new URL(dict.getLocation());
6	InputStream in = dictionary_location.openStream();
7	FileOutputStream w = new FileOutputStream(local_cache, false);
8	int b = 0;
9	while ((b = in.read()) != -1) {
10	w.write(b);
11	if (abort) throw new Exception("Download Aborted");
12	}
13-34	...
35	}

그림 9. 4번 기능 군의 메소드
Fig 9. Method of Functionality 4

라인	소스코드
1	private void unzipResource(final String resourceName, final File targetDirectory) throws IOException {
2	assertTrue(resourceName.startsWith("/"));
3	final URL resource = this.getClass().getResource(resourceName);
4-9	...
10	try {
11	in = new ZipInputStream(resource.openStream());
12	ZipEntry e;
13	while ((e = in.getNextEntry()) != null) {
14	if (e.isDirectory()) {
15	continue;
16	}
17	final File dest = new File(targetDirectory, e.getName());
18	assertTrue(dest.isAbsolute());
19	OutputStream out = null;
20	try {
21	out = FileUtils.openOutputStream(dest);
22	IOUtils.copy(in, out);
23-49	...
50	}

그림 10. 모델의 오탐 샘플
Fig 10. False Positive Sample

참 고 문 헌

[1] N. Bettenburg et al., “An empirical study on inconsistent changeto code clones at release level”, Proc. of the 16th Working Conference on Reverse Engineering, https://doi.org/10.1016/j.scico.2010.11.010

[2] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, “Comparison and evaluation of clone detection tools”, IEEE Trans. Software Eng., Vol. 33, No. 9, pp. 577-591, 2007. https://doi.org/10.1109/TSE.2007.70725

[3] J. Svajlenko and C. K. Roy, “Evaluating clone detection tools with BigCloneBench”, ICSME 2015, pp.131-140, 2015. https://doi.org/10.1109/ICSM.2015.7332459

[4] IJaDataSet-2.0, https://sites.google.com/site/asegsecold/projects/seclone
[5] H. Wei and M. Li, “Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code”, IJCAI 2017, pp.3034-3040, 2017. http://lamda.nju.edu.cn/lim/publications/ijcai17-clone.pdf
[6] Hao Yu, Wing Lam, Long Chen, Ge Li, Tao Xie, and Qianxiang Wang, “Neural Detection of Semantic Code Clones via Tree-Based Convolution”, Proceedings of the 27th IEEE/ACM Conference on Program Comprehension(ICPC 2019), Montreal, QC, Canada, May 25-26, 2019. https://doi.org/10.1109/ICPC.2019.00021
[7] C. K. Roy and J. R. Cordy, “A survey on software clone detection research”, Queens University, Tech. Rep., 2007. http://maveric0.uwaterloo.ca/~migod/846/papers/roy-CloningSurveyTechReport.pdf
[8] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, “Convolutional neural networks over tree structures for programming language processing”, in AAAI, 2016, pp.1287-1293. https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/11775/11735
[9] Efficient tree-based convolutional neural networks in Tensorflow, https://github.com/crestonbunch/tbcnn
[10] javalang, https://github.com/c2nes/javalang
[11] Neural network embeddings explained, https://towardsdatascience.com/neural-networkembeddings-explained-4d028e6f0526
[12] Word2Vec embeddings, 2019. https://radimrehurek.com/gensim/models/word2vec.html

저 자 소 개

박건우(Gunwoo Park)

2017.8 한양대학교 ERICA 컴퓨터공학과 학사
2017-현재 한양대학교 대학원 컴퓨터공학과 석사과정
<주관심분야> 프로그래밍언어, 기계 학습

홍성문(Sung-Moon Hong)

2012 위덕대학교 컴퓨터공학과 학사
2014 한양대학교 컴퓨터공학과 석사
2014-현재 한양대학교 대학원 컴퓨터공학과 박사과정
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

저 자 소 개

김현하(Hyunha Kim)

2003 한양대학교 ERICA 전자컴퓨터공학부 학사
2005 한양대학교 컴퓨터공학과 석사
2013 한양대학교 컴퓨터공학과 박사
2013-2016 한양대학교 ERICA 컴퓨터공학과 리서치펠로우
2017-현재 소프토피아(주) CTO, 한양대학교 ERICA 소프트웨어학부 겸임교수
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안

도경구(Kyung-Goo Doh)

1980 한양대학교 산업공학과 학사
1987 아이오와주립대학 컴퓨터과학 석사
1992 캔사스주립대학 컴퓨터과학 박사
1993-1995 일본 아이주대학 교수
1995-현재 한양대학교 ERICA 소프트웨어학부 교수
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안, 소프트웨어 공학