Journal of KIIT. Vol. 18, No. 3, pp. 79-88, Mar. 31, 2020. pISSN 1598-8619, eISSN 2093-7571
http://dx.doi.org/10.14801/jkiit.2020.18.3.79

TBCNN 기반 코드 클론 유형 분류

조영빈*, 유철중**, 이지현***

Clone Type Classification Based on Tree-Based Convolution Neural Network

Young-Bin Jo*, Cheol-Jung Yoo**, and Jihyun Lee***

요 약

소프트웨어 개발에서 코드 클론은 비용 절감과 개발 속도 증가를 가져오는 요소로 작용한다. 그러나 코드의 무분별한 복사는 코드의 품질을 낮추고, 버그 수정에 많은 비용과 시간을 필요로 한다. 코드 클론을 검출하기 위한 기존 기술들은 온전히 복사된 코드 혹은 식별자만 수정한 코드 이외의 클론은 검출하기 어렵고, 클론의 유형 분류는 사람이 직접 수행해야 한다. 본 논문은 다양한 형태의 클론 검출과 검출된 클론의 유형을 자동으로 분류하는 것을 목표로 TBCNN(Tree-Based Convolution Neural Network) 기반 클론 분류 방법을 제안한다. 제안한 방법을 코드 클론 검출 방법 검증에 많이 사용하는 BigCloneBench 데이터 셋을 이용하여 실험하였다. 그 결과, 제안한 방법은 78%의 재현율과 정밀도로 4가지 유형의 코드 클론을 분류할 수 있었다.

Abstract

In software development, code clone can significantly reduce costs and speed up the development process. However, indiscreet use of the clone not only lowers the quality of the code but also adds extra cost and time to fix the bugs. Due to this problem, many researchers have been trying to detect clones. However, the existing clone detection techniques are limited to detect clones that are fully identical or only with modified identifiers and they don't give information about clone type. This paper proposes a TBCNN(Tree-Based Convolution Neural Network)-based clone classification technique for detecting variety types of clones as well as their automatic classification. The experimental is performed using BigCloneBench, a well known and wildly adopted data set for clone detection. As a result, with 78% recall and precision, the proposed technique was able to classify four types of code clones.

Keywords

clone detection, clone classification, CNN, machine learning

* 전북대학교 소프트웨어공학과 석사과정
- ORCID: https://orcid.org/0000-0002-5968-1520
** 전북대학교 소프트웨어공학과 교수(교신저자)
- ORCID: https://orcid.org/0000-0002-7903-2309
*** 전북대학교 소프트웨어공학과 부교수
- ORCID: https://orcid.org/0000-0003-4512-806X

· Received: Nov. 04, 2019, Revised: Dec. 12, 2019, Accepted: Dec. 15, 2019
· Corresponding Author: Cheol-Jung Yoo
Dept. of Software Eng, Chonbuk National University, 567 Baekje-daero, Deokjin-gu, Jeonju-si, Jeollabuk-do, Korea,
Tel.: +82-63-270-3383, Email: cjyoo@jbnu.ac.kr

Ⅰ. 서 론

개발 및 유지보수 단계에 있는 소프트웨어 프로그램에는 종종 중복된 코드가 존재한다. 많은 연구자들은 소스코드의 20~59% 이상이 중복임을 보고한 바 있다[1]-[3]. 이처럼 이미 존재하는 코드의 일부를 복사하여 중복 코드를 만드는 일은 소프트웨어 개발 과정에서 빈번하게 일어난다. 이와 같이 복사된 코드를 '코드 클론(Code clone)' 혹은 간단히 '클론(Clone)'이라 부른다[4][5].

소프트웨어 개발에서 유사한 혹은 같은 기능을 수행하는 코드를 복사하여 사용하는 것은 빠른 소프트웨어 개발을 가능하게 하는 방법 중 하나이다. 이는 쉽고 저렴한 비용으로 기능을 구현할 수 있기 때문이다. 하지만 이러한 이점에도 불구하고 코드 클론은 해로운 코드로 간주된다. 예를 들어 코드 클론에서 버그가 발견되어 수정해야 하는 경우 해당 코드를 사용한 모든 곳을 확인하고 동일한 수정을 반복해야 한다. 특히, 대규모 소프트웨어 개발과 소프트웨어 유지보수의 경우 관련된 모든 코드를 확인해야하기 때문에 이 작업은 많은 비용과 시간을 필요로 한다. 또한 수정 도중에 누락이 발생한 경우 결함으로 이어지기도 한다.

이와 같은 문제를 해결하고자 많은 연구자들이 클론을 찾는 방법을 연구하였다. 일반적으로 클론을 검출하는 방법은 텍스트 기반, 토큰 기반, 트리 기반, 프로그램 의존성 그래프(PDG, Program Dependency Graph) 기반 그리고 메트릭 기반 방법으로 분류할 수 있다[5][6]. 클론 검출을 위한 많은 연구가 수행되었지만 PDG 기반 방법을 제외한 대부분의 클론 검출 방법은 코드 내 문장의 순서 재배열, 새로운 문장 삽입, 기존의 문장 제거 그리고 구현 방식은 다르지만 의미적으로 동일한 코드를 가리키는 클론(클론 유형 3과 4)을 잘 검출하지 못한다는 문제가 있다. PDG 기반 방법의 경우 클론을 검출하기 위한 시간이 오래 걸린다는 문제가 있다[6].

본 논문은 클론 유형 3과 4의 클론을 잘 검출할 뿐만 아니라 클론 유형의 분류까지도 할 수 있는 클론 분류 방법을 제안한다. 제안한 방법은 AST (Abstract Syntax Tree)로부터 코드의 구조적인 특징을 추출하기 위하여 트리 기반 TBCNN(Tree-Based Convolution Neural Network)[7]을 사용한다.

본 논문의 구성은 다음과 같다. 2장은 관련연구로 클론 검출과 관련된 기존 연구에 대하여 논한다. 3장에서는 본 논문이 제안하는 분류 방법을 설명한다. 4장에서는 클론 검출 분야에서 널리 사용되는 BCB(Big Clone Bench) 데이터 셋을 사용하여 제안한 클론 분류 방법을 평가한다. 마지막으로 5장에서는 결론과 향후 연구를 기술한다.

Ⅱ. 관련 연구

클론 검출 기법은 일반적으로 텍스트 기반, 토큰 기반, 트리 기반, PDG 기반, 메트릭 기반 기법으로 5가지 유형으로 구분된다.

텍스트 기반 클론 검출 기법은 실제 코드의 텍스트의 직접적인 비교를 통하여 코드의 클론을 검출하는 기법이다[1][8][9]. 텍스트 기반 기법은 단순하고 빠르게 클론을 찾는 장점을 갖는다. 하지만 텍스트를 직접 비교하는 방법적 한계로 인하여 완전히 동일한 클론 이외의 다른 클론의 검출에는 효과적이지 못하다는 한계가 있다.

토큰 기반 클론 검출 기법은 텍스트의 직접적인 비교 대신 코드의 텍스트를 토큰으로 분류한 후, 토큰들을 순차적으로 비교하여 클론을 검출한다[10][11]. 문장의 순서가 수정되거나 새로운 코드가 삽입되는 것과 같은 유형의 클론은 검출 효과가 떨어진다는 문제가 있다.

트리 기반 클론 검출 기법은 코드를 AST로 변환 후, 트리 매칭 알고리즘을 이용하여 클론을 검출하는 방법으로 텍스트 기반 기법의 한계를 해결하고자 하였다[12]-[14]. 이 방법은 토큰 기반 클론 검출 방법과 마찬가지로 코드의 텍스트 순서가 재배열된 경우 클론 검출이 어렵다는 한계가 있다.

이전의 기법들의 한계로 지적된 문장의 순서 변경, 새로운 문장의 삽입 또는 기존의 문장의 제거가 포함된 클론의 검출 문제를 해결하기 위한 기법이 PDG 기반 클론 검출 기법이다[15][16]. PDG 기반 클론 검출 기법은 데이터 흐름과 제어 흐름의 정보가 포함된 PDG를 이용하여 클론을 검출하는 방법

이다. 그렇지만 이 방법은 대규모 시스템에 적용이 어렵고 클론 검출에 많은 시간이 걸린다는 단점이 존재한다.

메트릭 기반 클론 검출 기법은 코드로부터 다양한 메트릭 값을 계산하여 코드의 직접적인 비교 대신에 수집한 메트릭 값의 비교를 통해 클론을 검출하는 기법을 말한다[2][17]. 메트릭 기반의 방법은 코드의 문장이나 구조와 같은 코드의 구체적인 정보를 활용하지 않기 때문에 유사한 메트릭 값을 갖는 두 코드가 실제로 동일한 코드가 아닌 경우가 클론으로 검출되는 한계가 있다.

최근에는 머신러닝 기술을 적용하여 코드의 특징을 추출하고 클론을 검출하고자 하는 연구가 이루어지고 있다[18][19]. 머신러닝 기술을 이용한 클론 검출 도구인 CCLearner는 토큰을 이용한 분석방법과 딥러닝 기법을 이용하여 클론 검출을 수행한다[18]. 머신러닝 기술을 이용한 다른 방법은 DeepSim 기법이다[19]. DeepSim은 제어 흐름과 데이터 흐름 정보를 이용하여 코드의 특징을 추출하고 클론 검출을 수행한 방법이다. CCLearner와 DeepSim 모두 BCB 데이터 셋[20]을 활용한 실험에서 전체적으로 준수한 결과를 보이지만, MT3와 WT3/4 유형의 클론 검출이 다른 유형에 비해 현저히 낮게 나오는 특징이 있다. 이에 본 논문에서는 AST와 머신러닝 기술을 이용하여 4가지 유형의 클론 검출 효과를 높이고 클론의 유형 분류까지 할 수 있는 TBCNN 기법을 이용한 클론 분류 방법을 제안한다.

III. TBCNN 기반 클론 분류

본 장에서는 본 논문에서 제안하는 TBCNN 기반 클론 분류 방법에 대해 설명한다. 그림 1은 본 논문에서 제안하는 TBCNN 기반 클론 분류 방법의 개요를 보여주고 있다.

제안하는 TBCNN 기반 코드 클론 분류 방법은 한 코드 조각이 다른 한 코드 조각의 클론인 경우, 해당 클론이 어떤 유형의 클론인지 판단하기 위하여 다음과 같은 두 번의 분류를 수행한다.

- 1차 분류(1st classification): 주어진 코드 조각이 다른 코드 조각의 클론인지 아닌지 두 가지로 분류한다(그림 1의 Binary classification).
- 2차 분류(2nd Classification): 클론으로 분류한 코드 조각의 클론 유형을 분류한다(그림 1의 Multi-Class Classification).

각 분류는 모두 코드 조각의 구조를 기술한 AST를 입력으로 한다. 1차 분류는 입력된 AST를 합성곱 신경망(그림 1의 Convolution step)을 거쳐 벡터(그림 1의 Latent representation)로 변환한 후, 합성곱 신경망과 연결된 인공신경망을 통하여 코드 클론에 대한 클론 분류를 수행한다(그림 1의 Binary classification). 2차 분류는 2진 분류 결과 클론으로 분류된 코드 조각을 대상으로 하며, 1차 분류와 같이 코드 조각의 AST를 합성곱 신경망을 통하여 벡터로 변환하고 합성곱 신경망과 연결된 인공신경망을 통해 클론 유형 분류를 수행한다(그림 1의 Multi-Class Classification).

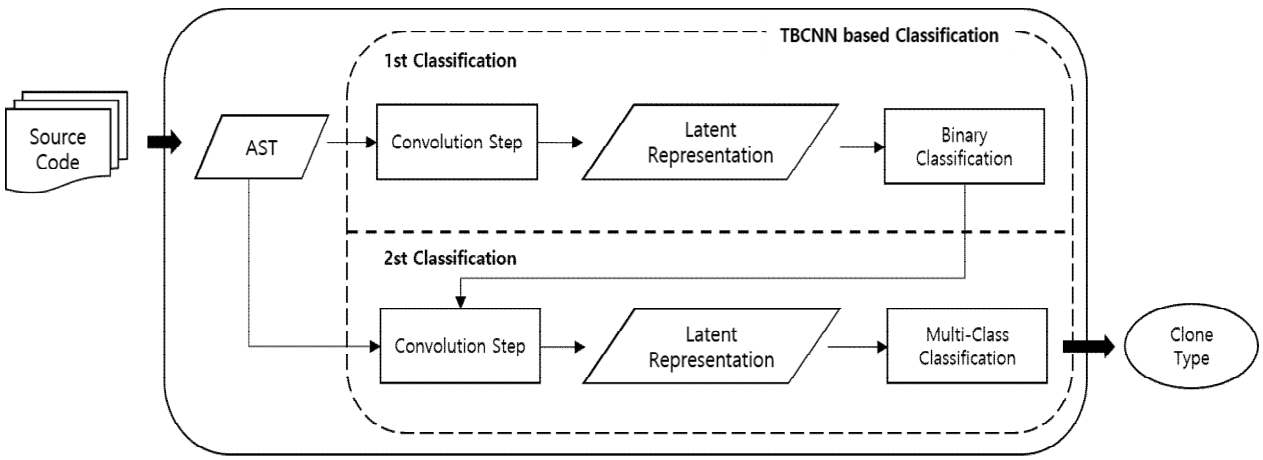

그림 1. TBCNN 기반 클론 분류 절차
Fig. 1. TBCNN-based clone classification procedure

3.1절부터 3.3절에서는 제안하는 TBCNN 기반 클론 분류 방법의 주요 단계인 합성곱, 2진 분류, 클론 유형 분류 단계를 상세히 설명한다.

3.1 트리 기반 합성곱(Tree-based convolution)

합성곱은 입력 데이터로부터 특징을 추출하는 단계이다. 합성곱은 입력 데이터의 형상을 유지하며 수행되는 특성을 갖고 있다. 일반적인 인공신경망의 경우, 이미지와 같은 다차원 데이터를 입력으로 사용할 때 입력 데이터를 1차원 데이터로 변환하는 과정을 수행한다. 이 과정에서 입력 데이터가 갖는 공간적 정보의 손실을 발생시킨다. 합성곱을 통한 특징 추출은 데이터의 형상을 유지는 특징으로 인해 공간적 정보의 손실을 최소화하는 방법이다.

본 논문의 경우, AST로부터 정보 손실을 최소화하며 특징을 추출하기 위해 CNN 기반 알고리즘 중 AST를 입력으로 하여 데이터의 구조적인 특징을 추출하는 TBCNN을 이용하였다. TBCNN은 CNN이 입력 데이터로부터 데이터의 형상을 유지하며 특징을 추출하는 과정을 트리 구조에서 수행할 수 있도록 설계된 인공신경망이다[7].

TBCNN은 일반적인 CNN의 합성곱 레이어와 유사하게, 고정된 깊이의 트리를 탐색하는 특징 추출기를 이용하여 주어진 트리를 탐색하는 트리 기반 합성곱 레이어를 갖는다. 특징 추출기는 하위 트리에서 동작하며, 그림 2[7]에서 보여주는 것과 같이 특징 추출기에 주어진 트리의 노드 벡터로부터 새로운 벡터를 생성한다.

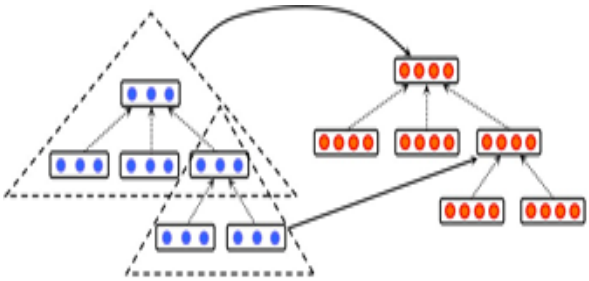

그림 2. 트리 기반 합성곱
Fig. 2. Tree-based convolution(left. original AST, right. new AST created by feature extractor)

특징 추출기에 주어진 하위 트리의 개의 노드는 다음 수식을 통해 하나의 벡터로 계산된다.

$$y = \tanh\left(\sum_{i=1}^{n} W_{conv,i} \cdot \quad x_i + b_{conv}\right) \tag{1}$$

식 (1)에서 $x_1,\cdots,x_i$는 트리의 노드를 벡터로 표현한 것이다. $W_{conv,i}$는 특징 추출기의 가중치를, b_{conv}는 바이어스를 의미한다. AST는 부모 노드가 가지는 자식의 수가 일정하지 않기 때문에 특징 추출기의 가중치인 $W_{conv,i}$의 크기를 설정하는데 어려움이 있다.

이 문제를 해결하기 위하여 TBCNN에서는 '연속 2진 트리(Continuous binary tree)' 개념을 사용한다. 연속 2진 트리는 노드의 자식 수에 관계없이 트리를 '2진' 트리로 보는 것이다. AST를 2진 트리로 간주함으로써 세 메트릭스의 선형결합으로 $W_{conv,i}$를 정의할 수 있다. 이들 세 메트릭스는 부모 노드에 대한 메트릭스인 W_{conv}^{t}, 자식 노드에 대한 메트릭스인 W_{conv}^{r} 그리고 W_{conv}^{l}이며, $W_{conv,i}$는 각 메트릭스에 실제 노드의 위치 정보를 포함한 계수를 곱하여 다음과 같이 정의된다.

$$W_{conv,i} = \eta_i^t W_{conv}^t + \eta_i^r W_{conv}^r + \eta_i^l W_{conv}^l \tag{2}$$

계수는 각각 다음과 같이 정의된다.

$$\eta_i^t = \frac{d_i - 1}{d - 1} \tag{3}$$

(단, d는 특징 추출기의 깊이, d_i는 특징 추출기에서 노드 i의 깊이)

$$\eta_i^r = (1 - \eta_i^t)\frac{p_i - 1}{n - 1} \tag{4}$$

(단, p_i는 형제 노드 중 노드 i의 인덱스, n은 형제 노드의 총 수)

$$\eta_i^l = (1 - \eta_i^t)(1 - \eta_i^r) \tag{5}$$

특징 추출기는 이와 같은 계산을 통해 AST를 탐색하며, 잎 노드에 도달하게 되면 잎 노드에 가상의

자식 노드를 생성하여 0의 값을 할당하여 계산을 수행한다. 이러한 가상 자식 노드의 생성은 트리 기반 합성곱 레이어의 입력과 출력의 트리의 형태를 동일하게 만드는 역할을 한다.

합성곱 레이어 이후, 데이터는 풀링 레이어(Pooling layer)를 통해 다운샘플링이 된다. 본 논문에서 사용되는 풀링 기법은 최대 풀링(Max pooling)이다.

그림 3[7]은 TBCNN의 각 레이어가 연결된 전체 동작 과정을 보여준다. 처음 코드 조각의 AST는 트리 기반 합성곱 레이어의 입력으로 사용하기 위하여 각 노드를 벡터로 변환한다. 각 노드가 벡터로 변환된 AST는 트리 기반 합성곱 레이어의 입력으로 주어져 특징이 추출된다. 이후, 풀링 레이어의 최대 풀링을 통해 특징 강화 단계를 거친다. 최종적으로 추출된 특징은 전결합 레이어를 통해 1차원 벡터로 변환되어 출력된다.

3.2 2진 분류

2진 분류는 입력으로 주어진 코드 조각이 다른 코드 조각의 클론인지 아닌지 분류하는 1차적으로 분류하는 단계이다. 클론 유형 분류를 수행하기에 앞서 2진 분류를 수행하는 이유는 클론 유형을 분류하는 과정에서 클론이 아닌 코드 조각들이 클론으로 판단되거나, 클론에 해당되는 코드 조각들이 클론이 아닌 것으로 분류되는 경우가 있기 때문이다. 여기에서 2진 분류를 수행한 경우와 수행하지 않은 경우의 클론 유형 분류의 정확도가 현저히 다른데, 이에 대한 내용은 4장에서 자세히 다룬다.

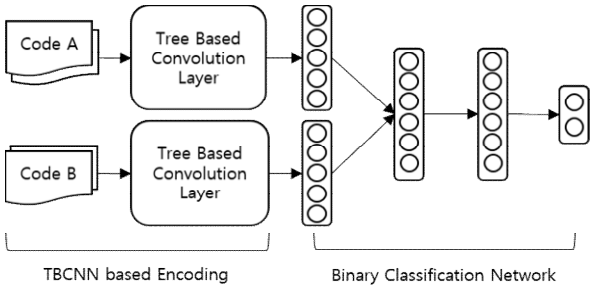

그림 4. TBCNN 기반 2진 분류기 구성
Fig. 4. Architecture of TBCNN-based binary classifier

본 논문은 피드-포워드 신경망을 이용하여 두 개의 서로 다른 코드 조각의 클론 여부에 대한 2진 분류를 수행한다. 2진 분류를 위한 피드-포워드 신경망은 합성곱 결과로 얻어진 각 코드 조각의 벡터 값을 입력으로 한다. 출력 노드 값은 0~1 사이의 값을 가지며, 이 값은 입력으로 주어진 코드 조각이 다른 코드 조각의 클론일 확률을 의미한다. 그림 4는 3.1절에서 설명한 합성곱 레이어와 피드-포워드 신경망이 결합된 코드 클론의 2진 분류를 위한 인공신경망의 전체적인 구조를 보여준다.

3.3 클론 유형 분류

클론 유형 분류 단계는 2진 분류 단계에서 클론으로 분류된 코드를 대상으로 세부 클론 유형을 분류한다. 코드 클론은 프로그램의 구문상의 유사도와 프로그램의 기능상 유사도를 기반으로 세부 유형으로 나눈다. 프로그램 구문이 유사한 코드 클론은 수정 없이 그대로 복사한 type 1, 구문이 동일한 type 2 그리고 구문이 유사한 type 3의 세 유형으로 분류하며, 프로그램의 의미가 유사한 클론을 type 4로 분류한다.

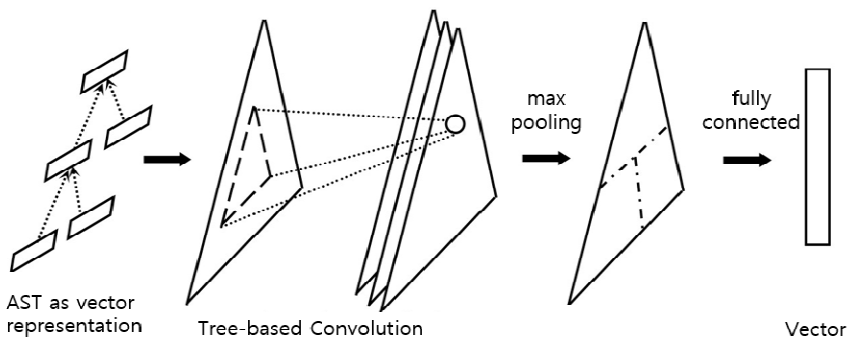

그림 3. TBCNN 구성
Fig. 3. Architecture of the TBCNN

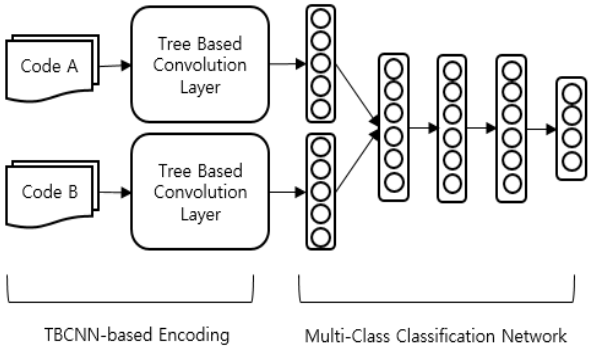

그림 5. TBCNN 기반 클론 유형 분류기의 구성
Fig. 5. Architecture of TBCNN-based clone type classifier

클론 유형 분류는 2진 분류와 같이 트리 기반 합성곱 레이어를 통해 벡터로 변환된 AST를 입력으로 하는 피드-포워드 신경망을 통하여 수행된다. 클론 유형 분류 모델의 구성은 2진 분류 모델과 유사한 구성을 갖는다. 그렇지만, 2진 분류 모델의 신경망이 클론과 클론이 아닌 것에 대응되는 두 개의 출력 노드로 구성된 것과 달리, 클론 유형 분류의 신경망은 네 개의 클론 유형에 대응되는 네 개의 출력 노드로 구성된다. 출력 노드는 각각 0~1사이의 값을 가지며, 각 노드는 클론 유형에 대응되어 코드가 해당 클론 유형일 확률을 의미한다. 그림 5는 클론 유형 분류에서 사용된 피드-포워드 신경망의 구조를 보여준다.

IV. 실 험

4.1 데이터 셋

본 논문은 분류 모델의 훈련 및 테스트를 위한 데이터로 BCB 데이터 셋을 사용하였다. BCB 데이터 셋은 J Svajlenko 등[19]이 클론 검출 기법간의 비교를 위하여 자바 언어로 구현된 특정 기능들의 클론을 수집하고 클론과 클론이 아닌 코드를 수동으로 분류해 놓은 데이터 셋이다. BCB 데이터 셋은 클론의 유형을 4가지로 분류하는 전통적인 유형 분류와는 다르게 5가지 유형(type 1, type 2, Strong type 3, Moderately type 3 그리고 Weakly type 3+4)으로 나눈 것이 특징이다.

BCB 데이터셋에서 클론 유형 type 1(T1)과 type 2(T2)는 각각 정규화 후 완전히 동일한 코드로 식별이 되는 코드들을 의미한다. T1의 정규화는 주석 제거와 Prettyprint의 수행을 의미하며 T2의 정규화는 T1의 정규화를 확장하여 식별자 이름과 리터럴 값의 일괄적인 수정이 포함된다. 이와 같은 방법으로 분류된 클론은 기존 클론 분류의 type 1과 type 2의 정의와 일치한다.

BCB 데이터 셋의 클론 유형 Strongly type 3(ST3), Moderately type 3(T3) 그리고 Weakly type 3+4(T3/4)는 동일한 기능을 수행하는 코드임에도 정규화 이후 완전히 동일한 코드로 식별되지 않은 클론의 분류이며 기존의 코드 유형 분류의 type 3과 type 4에 해당하는 클론이다. BCB 데이터 셋은 이와 같은 클론 유형 분류는 기존의 type 3와 type 4의 분류 기준이 모호하기 때문이다. 이 문제에 대하여 J Svajlenko는 라인 기반 메트릭스를 기준으로 구문 유사성을 측정하여 기존의 type 3와 type 4를 ST3, MT3 그리고 WT3/4로 분류하였다.

ST3는 구문 유사성이 [0.7, 1.0)에 해당되는 클론을 의미하며 이 기준은 대부분의 구문 클론 검출 도구들이 클론을 검출하는 기준과 동일하다. MT3은 구문 유사성이 [0.5, 0.7)인 클론을 의미하며 기존의 구문 클론 검출 도구를 통해서는 클론으로 판별되지는 않지만 50%이상 70%미만의 구문적 유사성으로 기존의 type 3, type 4 중 어느 것으로 특정하기 애매한 클론을 분류하기 위한 범주이다. 마지막 WT3/4는 구문 유사성 [0.0, 0.5)에 해당하는 클론을 말하며 이는 구문적 유사성이 없거나, 그 유사성이 미미한 클론의 분류를 위한 범주이다.

표 1. BCB 데이터 셋 구성
Table 1. Statistics of BCB data set

Total num of code fragments	59,618
Total num of code pairs	97,535
Num of none clone code pairs	20,000
Num of type-1	15,555
Num of type-2	3,663
Num of strongly type-3	18,317
Num of moderately type-3	20,000
Num of weakly type-3+4	20,000

BCB 데이터 셋의 구성은 표 1과 같다. BCB는 59,618개의 개별적인 코드 조각으로 구성되어 있으며 이 코드 조각들이 두 개씩 쌍을 이뤄 97,535개의

코드 조각 쌍을 만든다. 코드 조각의 쌍은 77,535개의 클론 조각 쌍과 20,000개의 클론이 아닌 조각 쌍으로 구분할 수 있으며 코드 클론 조각 쌍은 또한 각 유형별로 T1은 15,555개, T2는 3,663개, ST3는 18,317개, MT3는 18,317개, 마지막으로 WT3/4는 20,000개의 코드 쌍으로 구성되어 있다.

4.2 실험 환경

본 논문이 제안한 클론 분류 방법의 입력 형태인 AST를 자바 코드로부터 얻기 위하여 본 논문에서는 javalang 라이브러리를 사용하였다. javalang은 파이썬 환경에서 자바 코드의 분석과 같은 작업을 수행하기 위한 목적으로 만들어진 순수 파이썬 라이브러리이다.

또한 본 논문은 javalang을 통해 얻어진 AST의 각 노드를 벡터로 변환하기 위하여 AST2vec[21] 방법을 사용한다. AST2vec을 통해 얻어지는 노드의 벡터를 입력으로 하는 인공신경망은 텐서플로우(TensorFlow)를 기반으로 구현하였다. 인공신경망의 최적화된 성능을 위하여 파리미터 튜닝을 수행하였으며, 본 논문에서 설정한 인공신경망의 파라미터는 표 2와 같다.

4.3 실험 결과

표 3과 4는 각각 제안한 클론 분류 방법의 1차 분류와 2차 분류의 테스트 데이터에서 수행 결과이다. 각 분류 결과를 정확도 측면에서 살펴보면 1차 분류는 정확도가 0.95이고 2차 분류는 정확도가 0.76이었다. 재현율과 정밀도 측면에서 살펴보면 1차 분류는 클론인 코드 조각 쌍에 대해 재현율은 0.94, 정밀도는 0.96이고 클론이 아닌 코드 조각 쌍에 대해서 재현율이 0.97, 정밀도가 0.95이었다. 2차 분류의 경우 T1의 정밀도가 0.94로 가장 높았으며 WT3/4가 0.67로 가장 낮은 정밀도를 보였다. 그리고 재현율에는 T2가 0.93로 가장 높았으며 MT3이 0.58로 가장 낮게 나타났다.

표 3. 1차 분류(2진 분류) 결과
Table 3. Results of binary classification

Type	Recall	Precision	F1 score
Clone	0.94	0.96	0.95
Not clone	0.97	0.95	0.96

표 4. 2차 분류(클론 유형 분류) 결과
Table 4. Results of clone type classification

Type	Recall	Precision	F1 score
T1	0.75	0.94	0.84
T2	0.93	0.77	0.84
ST3	0.71	0.69	0.70
MT3	0.58	0.67	0.62
WT3/4	0.81	0.74	0.77

표 2. TBCNN 기반 클론 유형 분류 모델의 파라미터 설정
Table 2. Parameter setting for experiment

Step		Parameter configuration	
1st classification	Tree based convolution layer	Convolution: 1, Pooling layer: 1, Output size: 50	Learning rate: 0.001, Epoch: 100, Batch size : 5
	Fully-connected network	Input size: 100, Output size: 2, Hidden layer: 2, First hidden layer size: 100, Second hidden layer size: 150, Activation function: LRelu	
2nd classification	Tree based convolution layer	Convolution: 4, Pooling layer: 1, Output size : 200	Learning rate: 0.001, Epoch: 200, Batch size : 5
	Fully-connected network	Input size: 400, Output size: 5, Hidden layer: 3, First hidden layer size: 400, Second hidden layer size: 400, Third hidden layer size: 300, Activation function: LRelu	

제안한 클론 유형 분류 방법에서 1차 분류의 유무에 따른 분류 성능을 평가하기 위해 1차 분류를 수행하지 않고 클론 유형을 분류하는 인공신경망을 구축하여 제안한 방법과 비교하였다.

표 5는 본 논문에서 제안한 클론 분류 방법과 1차 분류를 수행하지 않은 방법의 결과 비교이다. 재현율에서 본 논문이 제안한 방법은 1차 분류를 수행하지 않는 방법보다 T1을 제외한 나머지 클론 유형에서 최소 0.05이상의 향상된 성능을 보여주었으며 특히 클론이 아닌 코드 조각 쌍의 분류에서 0.24의 향상된 성능을 보였다. 정밀도에서는 본 논문에서 제안한 방법이 클론 유형 T1, ST3, MT3 그리고 WT3/4에서 향상된 성능을 보여주었으며, 나머지 두 유형(Not clone, T2)에서 유사한 성능을 보였다. 이러한 결과를 통해 클론 유형 분류에 앞서 1차 분류를 수행하는 것이 클론 유형 분류에 있어 향상된 결과를 얻을 수 있다는 것을 알 수 있다.

제안한 TBCNN 기반 클론 분류 방법의 분류 결과에 대한 분석을 위하여, 먼저 모델이 오분류한 데이터에 대한 분포 조사를 수행하였다.

표 5. 1차 분류를 수행하지 않은 경우와의 비교
Table 5. Comparison of results with clone classification that don't use 1st classification

Clone type	Recall		Precision		F1 score	
	A	B	A	B	A	B
Not clone	0.73	**0.97**	**0.78**	0.77	0.76	**0.86**
T1	**0.76**	0.75	0.90	**0.95**	0.82	**0.84**
T2	0.89	**0.93**	0.77	**0.78**	0.73	**0.85**
ST3	0.68	**0.73**	0.67	**0.72**	0.67	**0.72**
MT3	0.54	**0.57**	0.63	**0.71**	0.58	**0.63**
WT3/4	**0.77**	0.72	0.64	**0.77**	0.70	**0.74**

* A: model without 1st classification, B: our approach

표 6. 오분류된 데이터의 분포
Table 6. Distribution of misclassified data

Clone type	TBCNN based clone classification result					
	Not Clone	T1	T2	ST3	MT3	WT3/4
Not clone	-	2	1	138	165	330
T1	7	-	**2903**	160	13	11
T2	1	**61**	-	**160**	13	11
ST3	922	318	**1283**	-	**2205**	407
MT3	1687	69	68	**3452**	-	**3495**
WT3/4	**2962**	71	15	465	**1841**	-

표 6은 모델에 의해 오분류된 데이터의 분포를 보여준다. 표 6을 통하여 오분류된 데이터의 대다수는 인접한 클론 유형으로 분류되는 경향을 확인할 수 있다.

이러한 경향성의 이유를 파악하기 위하여 오분류된 데이터의 코드 및 AST를 분석을 수행하였다. 그 결과 2가지 일관된 특징을 발견하였다. 첫째, 동일 레벨에서의 일부 노드 순서의 변화는 ST3 수준 이상의 높은 수준의 클론 유형으로 분류하기 위한 특징으로 고려되지 않는다. AST의 노드 순서에 영향을 미치는 경우는 코드 재배열이 이뤄진 경우이다. 이 수준의 코드 수정은 ST3로 분류되는 코드에서 주로 나타난다. 다수의 이러한 코드는 제안한 모델에서 T2로 분류되는 경향을 보였다.

둘째, 두 AST가 동일한 서브트리로 구성되어 있어도 그 서브트리가 나타난 레벨이 다르다면 MT3 수준의 클론으로 분류하는 경향이 있다. AST에서 서브트리의 레벨에 영향을 미치는 코드 수정은 기존 코드에는 없던 제어문(if, for, try-catch 등)을 이용하여 코드를 수정한 경우이다. 이러한 코드 수정은 ST3 혹은 MT3로 분류된 코드에서 발견할 수 있다. 이 중 다른 서브트리는 동일하지만 제어 노드의 추가로 특정 서브트리의 레벨만 변화한 경우, BCB에서는 ST3로 분류하였다. 하지만 이와 같은 특징을 갖는 다수의 코드에 대하여 제안한 클론 분류 방법은 MT3로 분류하는 경향을 보였다.

V. 결론 및 향후 과제

본 논문은 두 코드 조각 사이의 클론 유형을 분류하는 TBCNN 기반 클론 분류 방법을 제안하였다. 이전 머신러닝을 이용하여 클론 분류를 수행한 연구와 비교하였을 때, 다음 세 가지 측면에서 기여하고 있다:

- 제안한 클론 분류 방법은 보다 간단한 전처리 과정만으로 높은 정확도를 갖는 클론 분류 모델을 구축하였다.
- 제안한 클론 분류 방법은 기존 머신러닝을 이용한 방법들과 달리 클론의 유형 분류까지 수행한다.
- 클론의 유형 분류 이외에도 제안한 클론 분류 방법은 사용자의 필요에 따라 선택적으로 사용할

수 있다.

본 논문은 제안한 클론 분류 방법이 간단한 전처리로도 높은 수준의 클론 검출이 가능함을 확인하였고, 더 나아가 78%의 클론 유형 분류의 정확도를 가짐을 검증하였다. 하지만 제안한 클론 분류 방법은 BCB 데이터 셋의 MT3클론 유형에서 재현율이 다른 클론 유형에 비해 낮다. 이러한 문제를 해결하기 위해서는 분류가 잘 수행된 코드 조각의 특징과 잘못 분류된 코드 조각의 특징을 파악하기 위한 데이터 분석이 선행되어야 한다.

향후에는 관련 코드 조각의 데이터 분석을 통하여 얻은 결과를 토대로 전처리 과정의 수정, 보완 또는 TBCNN 이외의 모델 사용 등 제안한 방법을 개선해 나갈 계획이다.

References

[1] B. Baker, "On Finding Duplication and Near-duplication in Large Software System", Proceedings of 2nd IEEE Working Conference on Reverse Engineering, Toronto, Ontario, Canada, pp. 86-95, Jul. 1995.

[2] K. Kontogiannis, R. Demori, E. Merlo, M. Galler, and M. Bernstein, "Pattern Matching for Clone and Concept Detection", Automated Software Engineering, Vol. 3, No. 1-2, pp. 77-108, Jun. 1996.

[3] B. Lague, D. Proulx, J. Mayrand, E. Merlo, and J. Hudepohl, "Assessing the Benefits of Incorporating Function Clone DFetection in a Development Process", 1997 Proceedings International Conference on Software Maintenance, Bari, Italy, pp. 314-321, Oct. 1997.

[4] Stefan Bellon, Rainer Koschke, and Jens Krinke, "Comparison and evaluation of clone detection tools", IEEE Transactions on Software Engineering, Vol. 33, No. 9, pp. 577-591, Sept. 2007.

[5] Chanchal K. Roy, James R. Cordy, and Rainer Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach", Science of Computer Programming, Vol. 74, pp. 470-495, May 2009.

[6] Pratiksha Gautam and Hemraj Saini, "Various Code Clone Detection Techniques and Tools: A Comprehensive Survey", International Conference on Smart Trends for Information Technology and Computer Communications, Jaipur, India, pp. 665-667, Aug. 2016.

[7] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin, "Convolution Neural Networks over Tree Structures for Programming Language Processing", Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Arizona USA, pp. 1287-1293, Feb. 2016.

[8] J Howard Johnson, "Navigating the textual redundancy Web in legacy source", Proceedings of the 1996 Conference of the Centre for Advanced Studies on Collaborative Research, Toronto, Ontario, Canada, pp. 7-16, Nov. 1996.

[9] James Cordy, Thomas Dean, and Nikita Synytskyy, "Practical Language-Independent Detection of Near-Miss", Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research, Markham, Ontario, Canada, pp. 1-12, Oct. 2004.

[10] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue, "CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code", Transactions on Software Engineering, Vol. 28, No. 7, pp. 654-670, Aug. 2002.

[11] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou, "CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code", Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI'04), San Francisco, CA, Vol. 6, pp. 289-302, Jan. 2004.

[12] I. D. Baxter, A. Yahin, L. Moura, M. Sant' Anna and L. Bier, "Clone Detection Using Abstract Syntax Trees", Proceedings of the International Conference on Software Maintenance, Bethesda, MD, USA, pp. 368-377, Nov. 1998.

[13] V. Wahler, D. Seipel, Jurgen Wolff von

Gudenberg, and G. Fischer, "Clone detection in source code by frequent itemset techniques", Proceedings of the 4th IEEE International Workshop Source Code Analysis and Manipulation, Chicago, IL, USA, pp. 128-135, Sep. 2004.

[14] Williams Evans and Christopher Fraser, "Clone Detection via Structural Abstraction", Software Quality Journal, Vol. 17, No. 4, pp. 309-330, Dec. 2009.

[15] Raghavan Komondoor and Susan Horwitz, "Using Slicing to Identify Duplication in Source Code", Proceedings of the International Symposium on Static Analysis (SAS'01), Paris, France, pp. 40-56, Jul. 2001.

[16] Jens Krinke, "Identifying Similar Code with Program Dependence Graphs", Proceedings of the 8th Working Conference on Reverse Engineering (WCRE'01), Stuttgart, Germany, pp. 301-309, Oct. 2001.

[17] Jean Mayrand, Claude Leblanc, and Ettore Merlo, "Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics", Proceedings of the International Conference on Software Maintenance, Monterey, CA, USA, pp. 244-253, Nov. 1996.

[18] Liuqing Li, He Feng, Wenjie Zhuang, Na Meng and Barbara Tyder, "CCLearner: A Deep Learning-Based Clone Detection Approach", IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, pp. 249-260, Sep. 2017.

[19] Gang Zhao and Jeff Huang, "DeepSim: Deep Learning Code Functional Similarity", Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Lake Buena Vista FL USA, pp. 141-151, Oct. 2018.

[20] Jeffry Svajlenko and Chanchal K.Roy, "Evaluating Clone Detection Tools with BigCloneBench", IEEE International Conference on Software Maintenance and Evolution, Bremen, Germany, pp. 131-140, Oct. 2015.

[21] Nghi D. Q. Bui, Lingxiao Jiang, and Yijun Yu, "Cross-Language Learning for Program Classification using Bilateral Tree-Based Convolution Neural Networks", Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence, Palo Alto, CA, pp. 758-761, Feb. 2018.

저자소개

조 영 빈 (Young-Bin Jo)

2018년 2월 : 전북대학교 소프트웨어공학과(공학사)
2018년 3월 ~ 현재 : 전북대학교 소프트웨어공학과 석사과정
관심분야 : 빅데이터 분석, 인공지능

유 철 중 (Cheol-Jung Yoo)

1982년 2월 : 전북대학교 전산통계학과(이학사)
1985년 8월 : 전남대학교 대학원 계산통계학과(이학석사)
1994년 8월 : 전북대학교 대학원 전산통계학과(이학박사)
2012년 1월 ~ 2013년 7월 : University of California, Irvine(UCI) 국외연구교수
1997년 ~ 현재 : 전북대학교 소프트웨어공학과 교수
관심분야 : 소프트웨어품질/메트릭스/테스팅, 임베디드 소프트웨어/테스팅, 빅데이터 분석 등

이 지 현 (Jihyun Lee)

2005년 6월 ~ 2012년 2월 : 한국과학기술원 연구조교수
2012년 3월 ~ 2016년 2월 : 대전대학교 리버럴아트칼리지 조교수
2016년 3월 ~ 2019년 9월 : 전북대학교 소프트웨어공학과 조교수
2019년 10월 ~ 현재 : 전북대학교 소프트웨어공학과 부교수
관심분야 : 소프트웨어제품라인, 테스팅, 아키텍처 재구축