바이트 코드와 SBERT를 활용한 파이썬 코드 유사도 분석

문성수, 박태준*
전남대학교(학부생), *전남대학교(교수)
dalcw@jnu.ac.kr, *taejune.park@jnu.ac.kr

The method for Python code analysis using bytecode and SBERT

Seongsu Moon, Taejune Park*
Chonnam National Univ., *Chonnam National Univ.

요 약

소스 코드의 동작상 유사도 분석은 악성 코드 탐지 및 취약점 분석 등에 널리 활용되는 기법으로, 일반적으로 소스 코드로부터 생성된 실행 바이너리나 소스 코드 원문에 대한 의미론(Semantic)을 비교하며 이루어진다. 그런데 이를 조금 더 높은 정확도로 분석하기 위해서는 실행 시간 및 단계에 따른 동작을 이해하기 위한 흐름이 명확히 반영되는 것이 좋으나, 바이너리 기반 분석의 경우 소스 코드를 구성하는 비트의 수가 너무 많아 간단한 소스 코드의 변경에 대해서는 민감하게 반응하지 못한다는 문제점이 있으며, 코드의 의미론적 분석의 경우에는 개발자가 임의로 정의한 키워드에 의해 분석이 방해될 수 있기에 분석에 있어 흐름적 특성을 온전히 고려하는 것이 쉽지가 않다. 따라서 본 논문에서는 이러한 문제를 해결하기 위해 바이트 코드 수준에서 소스 코드의 동작상 유사도를 분석할 수 있는 기법을 제시하고자 한다. 본 기법은 소스 코드를 바이너리보다는 추상적이며 소스 코드보다는 더 명확한 형태의 바이트 코드로 변환시키고, 이를 자연어 처리에서 문장 유사도를 비교하는 데 사용되는 SBERT(Sentence BERT)을 통해 분석하도록 한다. 이때, 자연어 처리에 필요한 학습 가중치와 코드 분석에서 필요한 가중치의 차이를 극복하기 위해, 미세 조정(Fine tuning) 방법론을 통해 SBERT가 바이트 코드 간의 유사도 분석에 활용될 수 있도록 하였다. 본 기법을 실제 "*월간 데이콘 코드 유사성 판단 AI 경진대회*"에서 공개된 3,317개의 파이썬 소스 코드를 통해 평가해 본 결과 99.5%의 높은 분류 정확도를 달성할 수 있음을 확인하였다.

Ⅰ. 서 론

소스 코드의 동작상 유사도 분석은 악성 코드 탐지, 시스템의 취약점 발견과 같은 보안적으로 밀접한 분야에서 주로 사용된다 [1]. 이처럼 중요한 도구로 사용되는 만큼 다양한 방법론들이 제시되고 있는데, 주로 실행 파일에 해당하는 바이너리 분석이나, 소스 코드 자체에서의 의미론적 분석 기법들이 제시되고 있다 [2]. 그런데 더욱 효과적인 동작상 유사도 분석을 위해서는 소스 코드의 실행 시간 또는 단계에 따른 흐름이 명확하게 표현되는 것이 좋지만, 바이너리 분석은 많은 비트열을 포함하고 있어 간단한 소스 코드의 변경에 대해서는 민감하게 반응하지 못한다는 문제점이 있으며, 소스 코드를 통한 의미론적 분석은 개발자가 정의한 변수명, 함수명 등의 키워드로 인해 분석이 방해를 받는 등의 문제로, 흐름적 특성을 온전히 보존할 수 없다 [3,4].

따라서, 해당 문제를 해결하기 위해 바이너리와 소스 코드의 중간 형태의 언어인 바이트 코드(Bytecode)를 사용하고자 한다. 바이트 코드는 파이썬과 자바와 같이 코드 실행의 주체가 가상 머신(Virtual Machine)인 언어들에서 중간 단계로 존재하는 형태로 프로그램의 동작 흐름을 일련의 정해진 집합의 명령어들의 나열로 표현하기 때문에 코드의 실행 과정에 따른 흐름적 표현은 잘 유지하면서 코드 의미에 따른 의미 확장은 최소화할 수 있다.

또한 유사도 분석을 위한 방법론으로는 SBERT를 도입하고자 한다. SBERT는 자연어 처리 분야에서 두 문장의 유사도를 비교하기 위해 사용되는 모델로, 유사한 문장들끼리는 높은 유사도를, 다른 문장들끼리는 낮은 유사도를 갖도록 임베딩할 수 있다 [5]. SBERT는 사전 학습된 가중치를 제공하고 있지만, 이는 자연어를 기반으로 학습되었기 때문에 코드 도메인에 곧바로 적용하기에는 어려움이 있다. 이러한 문제를 미세 조정(Fine tuning)기법을 적용하여 해결하였으며, 이로써 코드 도메인에서도 유사도에 따른 임베딩을 가능케 하였다.

본 논문에서는 이와 같이 소스 코드를 바이트 코드로 변환하는 전처리와 유사도 분석에 SBERT를 사용하는 것과 같은 새로운 접근법을 도입함으로써 소스 코드의 동작상 유사도 분석에서 극적인 성능을 달성할 수 있었으며, 이는 소스 코드의 동작상 유사도 분석 분야에서 새로운 방향을 제시할 수 있을 것으로 기대된다.

Ⅱ. 본론

2.1 Background

본 장에서는 제안하는 방법론을 이해하기 위해 필요한 배경지식인 파이썬 바이트 코드(Bytecode)와 SBERT의 개념에 대해 소개한다.

2.1.1 Python Bytecode

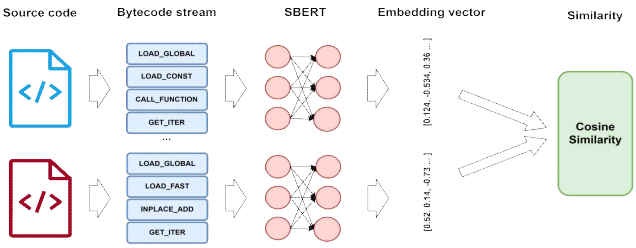
파이썬 언어로 작성된 소스 코드는 곧바로 CPU가 해석 가능한 기계어로 변환되는 것이 아닌, 파이썬 가상 머신(Python Virtual Machine: PVM)을 경유하여 실행된다. 이 때 소스 코드가 가상 머신에 전달되어 실행되기까지의 일련의 과정에서 전처리 작업이 수행되는데, 그중 한 작업이 파이썬 소스 코드를 바이트 코드로 변환하는 작업이다.

이와 같은 바이트 코드는 실행을 위한 저수준의 표현으로, 소스 코드와 동작상 일치하는 120여 가지의 간결한 명령어 나열로 치환된다 [6]. 최종적으로 이처럼 변환된 명령어 나열을 가상 머신이 순차적으로 처리함으로써 프로그램은 동작하게 된다. 이러한 변환된 바이트 코드는 프로그램의 흐름에 따른 동작 정보가 잘 나타나 있기 때문에 본 논문에서는 소스 코드의 동작상 유사도 분석을 위한 전처리 기법으로 해당 변환을 적용한다.

2.1.2 SBERT(Sentence BERT)

BERT는 12개 또는 24개의 트랜스포머(Transformer)의 인코더로 구성된 모델로, 위키피디아와 BooksCorpus와 같은 대규모 데이터 셋으로 사전 훈련되었다 [7]. BERT는 감정 분석, 질의응답, 텍스트 생성 등의 자연어처리 분야에서 높은 성능을 보여주며, 그뿐만 아니라 사전 훈련된 가중치를 다른 과업에 전이할 수 있다는 개념을 도입함으로써, 자연어 처리 분야에서도 미세 조정 및 전이 학습을 가능하게 하였다.

이러한 BERT의 등장으로 이를 기본 구조로 활용하는 다양한 언어 모델들이 등장하였는데, 본 논문에서 핵심 구조로 삼고 있는 SBERT 역시 BERT를 기본으로 하는 모델 중 하나이다. SBERT는 주로 문장 사이의 유사도를 분석하기 위한 과업에서 주로 사용되며, 유사한 의미를 가는 문장들은 높은 유사도를, 다른 의미의 문장들은 낮은 유사도를 갖도록 문장을 임베딩하는 역할을 한다 [5]. 즉,



〔Fig 1〕 System overview

SBERT는 문장 간의 유사도를 학습한다. 이를 위해 SBERT는 BERT 모델 두 개를 사용하는 샴 네트워크(Siamese Network) 구조를 도입하였으며, 손실 함수에는 코사인 유사도(Cosine similarity)의 개념이 사용되었다. 이와 같은 설계로 인해 SBERT는 문장 사이의 유사도를 효과적으로 학습할 수 있으며, 텍스트 유사성 분석 분야에서 SOTA(State-of-the-art)로 자리매김하고 있다.

2.2 Methodology

본 논문에서는 파이썬 언어로 작성된 소스 코드에서의 유사도를 판별하는 모델을 개발하기 위해 **(1) 소스 코드를 바이트 코드로 변환하는 전처리 기법을 도입하였으며, (2) 자연어 처리 분야에서 두 문장 사이의 유사도를 비교하는 용도로 사용되는 SBERT 모델을 적용**하였다.

2.3 Design

제안하는 시스템은 [Fig 1]와 같이 5가지 단계로 구성된다: 1) 소스 코드 입력, 2) 소스 코드를 바이트 코드로 변환, 3) SBERT 모델 적용, 4) SBERT 모델에서 임베딩 벡터 생성, 5) 다른 소스 코드로부터 생성된 임베딩 벡터 간의 코사인 유사도 비교

2.3.1 Dataset

본 연구에서는 국내 AI 해커톤 플랫폼인 데이콘에서 개최한 "*월간 데이콘 코드 유사성 판단 AI 경진대회*"에 사용된 데이터 셋을 활용하였다 [8].

해당 대회에서는 학습을 위한 데이터로 알고리즘 문제 300개의 풀이에 해당하는 파이썬 소스 코드 중에서 17,970개의 쌍이 샘플링 되어 제공된다. 쌍을 이루는 두 개의 코드가 같은 동작을 하는 경우 1로, 다른 동작을 하는 경우 0으로 어노테이션 되어 있다.

2.3.2 Transformation to Bytecode

소스 코드의 시계열적 성질을 극대화하기 위해 입력으로 주어진 소스 코드를 바이트 코드로의 변환 과정을 수행한다. 이를 위해 파이썬 패키지인 dis의 dis.dis() 메소드를 사용하여 소스 코드를 바이트 코드로 변환하였다. dis.dis() 메소드를 사용하면 프로그램 동작 과정에서 필요로 하는 기본 명령어(e.g. LOAD_CONST, CALL_FUNCTION)와 인자(argument) 정보가 모두 추출된다. 이때, 인자 정보에 해당하는 값은 프로그램의 실행 환경이나, 입력으로 주어지는 값에 따라 가변적으로 바뀌는 경향이 있으므로, 이를 변환 과정에 포함할 시 프로그램의 시계열적 성질을 오히려 해칠 수 있다. 따라서, 변환 시 인자 정보는 제외하고, 오직 명령어의 종류와 순서 정보만을 활용하였다.

2.3.3 SBERT Model

SBERT는 비슷한 의미의 문장끼리는 코사인 유사도가 높게, 다른 의미의 문장끼리는 코사인 유사도가 낮게 임베딩이 가능하다는 성질을 가진다. 이러한 성질 덕택에 자연어 처리 분야에서는 문장 사이의 의미론적 유사도를 판별하기 위한 모델로 SBERT가 주로 사용된다. 본 논문에서는 SBERT의 이러한 성질을 **"2.3.2 Transformation to bytecode"**에서 변환된 바이트 코드 나열에 적용하여 소스 코드의 동작상의 유사도를 측정하고자 한다.

SBERT는 사전 학습된 가중치가 제공되기는 하지만, 자연어 도메인에서 학습이 이루어졌기 때문에, 이전 과정에서 생성된 바이트 코드 나열에 곧바로 적용하기에는 어려움이 있다. 이는 코드 도메인과 바이트 코드 간의 구조론적 차이에 의해 발생하는 문제로, 해당 문제를 해결하기 위해 사전 학습된 가중치에 대해 미세 조정 기법을 적용하였다. 미세 조정이 진행될수록 SBERT은 점차 바이트 코드에 대해 유사도 기반 임베딩이 가능해지며, 이로써 바이트 코드 도메인에서 유사도 분석을 할 수 있는 SBERT모델이 생성된다.

2.3.4 Cosine similarity

Bytecode 나열은 SBERT를 통해 유사도 기반 임베딩이 된다. 이때 두 코드 간 유사도를 수치상으로 확인하기 위한 지표가 필요한데, 여기서는 코사인 유사도를 사용한다. 코사인 유사도는 비교하고자 하는 두 벡터가 유사할수록 1의 값에 가까워지고, 다를수록 -1에 가까워진다 [9].

2.4 Evaluation

평가를 위한 데이터로는 두 개의 코드가 쌍으로 구성되어 있으며, 그 두 코드의 동작상의 유사도가 어노테이션 되어 있는 형식의 데이터 3,317개를 사용하였다. 방식은 이진 분류에서의 정확도 평가로 진행되었으며, 특정 코사인 유사도 값을 임계값으로 사용하여 임계값보다 이상인 경우 동일한 동작을 하는 소스 코드(label: 1)로, 임계값보다 이하인 경우 다른 동작을 하는 소스 코드(label: 0)로 분류하였다, 이때 임계값 설정은 평가 데이터에서 가장 높은 정확도를 보이는 값으로 설정하였다.

해당 실험에서는 코사인 유사도 0.21을 임계값으로 분류를 수행하였을 때, 분류 정확도 99.5%를 달성하였다.

Ⅲ. 결론

본 논문에서는 파이썬 소스 코드의 동작상의 유사도를 측정하는 시스템을 설계하기 위해 파이썬 소스 코드를 바이트 코드로 변환하는 전처리 기법과 자연어 처리 분야에서 문장 사이의 유사도를 비교하기 위해 사용되는 SBERT 모델을 도입하였다. 이러한 설계를 통해 정확도 측면에서 극적인 성능을 달성하였으며, 이는 기존 코드 유사도 분석 시스템이 사용되는 악성 코드 분석이나 시스템의 취약점을 탐지하는 분야에서 훌륭한 보조 도구로 사용될 수 있을 것이다.

ACKNOWLEDGMENT

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 인공지능융합혁신인재양성사업 연구 결과로 수행되었음(IITP-2023-RS-2023-00256629)

참 고 문 헌

[1] Tian, Donghai, et al. "BinDeep: A deep learning approach to binary code similarity detection." Expert Systems with Applications 168 (2021): 114348.

[2] Haq, Irfan Ul, and Juan Caballero. "A survey of binary code similarity." ACM Computing Surveys (CSUR) 54.3 (2021): 1-38.

[3] Kim, Dongkwan, et al. "Revisiting binary code similarity analysis using interpretable feature engineering and lessons learned." IEEE Transactions on Software Engineering 49.4 (2022): 1661-1682.

[4] Xie, Chunli, et al. "A source code similarity based on Siamese neural network." Applied Sciences 10.21 (2020): 7519.

[5] Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." arXiv preprint arXiv:1908.10084 (2019).

[6] Python Byte code dis assembler page - https://docs.python.org/ko/3.8/library/dis.html

[7] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[8] Dacon Competition-https://dacon.io/competitions/official/235900/overview/description

[9] Xia, Peipei, Li Zhang, and Fanzhang Li. "Learning similarity with cosine similarity ensemble." Information sciences 307 (2015): 39-52.