The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 22, No. 4, pp.73-79, Aug. 31, 2022. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2022.22.4.73
JIIBC 2022-4-11

Python 기반 AI 프로젝트에서 예외 제안을 위한 자동화 접근 방식

An Automated Approach for Exception Suggestion in Python-based AI Projects

강민구*, 김순태, 류덕산****

Mingu Kang*, Suntae Kim**, Duksan Ryu**

요 약 Artificial intelligence (AI) 프로젝트에 널리 사용되는 Python 언어는 Interpreter 언어로 Runtime 시에 오류가 발생한다. 오류로 인한 프로젝트의 실패를 방지하기 위해서는 사전에 예외적인 상황이 발생할 수 있는 코드에 대한 예외 처리가 필요하다. 특히, 많은 리소스를 필요로 하는 AI 프로젝트에서, 오랜 실행 후 발생하는 예외는 큰 리소스 낭비를 초래한다. 하지만, 예외 처리는 개발자의 경험에 의존하기 때문에 개발자들은 잡아야 할 적절한 예외를 결정하는데 어려움을 가진다. 이러한 필요성을 해결하기 위해 기존 예외 처리문을 학습하여 개발 중에 개발자에게 잡아야 할 예외를 제안해주는 접근 방법을 제안한다. 제안 방법은 try 블록의 소스 코드를 입력으로 받아 except 블록에서 처리되어야 할 예외들을 제안해준다. 우리는 2개의 프레임워크로 구성된 대규모 프로젝트에 대해 접근 방법을 평가한다. 우리의 평가 결과에 따르면, 예외 제안을 수행할 때 평균 AUPRC는 0.92 이상을 나타낸다. 연구 결과는 제안된 방법이 비교 모델들을 능가하는 예외 제안 성능으로 개발자의 예외 처리를 지원할 수 있음을 보여준다.

Abstract The Python language widely used in artificial intelligence (AI) projects is an interpreter language, and errors occur at runtime. In order to prevent project failure due to errors, it is necessary to handle exceptions in code that can cause exceptional situations in advance. In particular, in AI projects that require a lot of resources, exceptions that occur after long execution lead to a large waste of resources. However, since exception handling depends on the developer's experience, developers have difficulty determining the appropriate exception to catch. To solve this need, we propose an approach that recommends exceptions to catch to developers during development by learning the existing exception handling statements. The proposed method receives the source code of the try block as input and recommends exceptions to be handled in the except block. We evaluate our approach for a large project consisting of two frameworks. According to our evaluation results, the average AUPRC is 0.92 or higher when performing exception recommendation. The study results show that the proposed method can support the developer's exception handling with exception recommendation performance that outperforms the comparative models.

Key Words : AI Project, Exception Suggesting, Handling Exception

*준회원, 전북대학교 소프트웨어공학과
**정회원, 전북대학교 소프트웨어공학과
접수일자 2022년 7월 11일, 수정완료 2022년 7월 30일
게재확정일자 2022년 8월 5일

Received: 11 July, 2022 / Revised: 30 July, 2022 /
Accepted: 5 August, 2022
*Corresponding Author: stkim@jbnu.ac.kr
Department of Software Engineering, CAIIT, Jeonbuk National University, Korea

Ⅰ. 서 론

Artificial intelligence (AI) 기술은 컴퓨팅 성능과 데이터의 성장으로 다양한 분야에 널리 적용되고 있다[1][2]. AI 개발자들은 데이터 분석과 시각화 도구를 통한 데이터의 통찰을 얻기 위해 다양한 라이브러리 및 프레임 워크를 지원하는 Python 언어를 선택하여 사용한다[3]. 이러한 Python 언어는 컴파일 과정 없이 interpreter를 직접 거쳐 시스템을 한 줄 단위로 실행시키는 언어이기 때문에 Runtime 시에 시스템의 오류가 발생한다. Runtime 시에 발생하는 오류는 오직 소스 코드의 실행을 통해 확인할 수 있어 프로그램의 비정상적인 종료를 야기한다. 특히, Python 기반의 AI 서비스를 지원하는 프로젝트에서 오랜 실행 후 발생한 오류는 많은 양의 GPU 및 네트워크 I/O와 같은 리소스 낭비를 초래한다. 따라서, 이러한 Runtime 시의 시스템 오류를 방지하기 위해 사전에 예외에 대한 적절한 처리 작업을 수행해 주어야 한다.

하지만 예외 처리는 개발자의 경험에 의존하기 때문에 사전에 발생할 모든 예외를 인지하고 처리하기 어렵다. 이러한 예외 처리의 어려움으로 인해 개발자들은 예외 처리의 중요성에도 불구하고 예외 처리를 무시하거나 일반적인 상위 예외를 잡는다[4]. 예를 들어, Zhang의 연구는 AI 프로젝트에서 방대한 양의 리소스 손실이 발생할 수 있음에도 불구하고 예외를 처리하지 않아 많은 Runtime 오류를 발생시키고 있음을 보여준다[5]. 또한, 적절하지 않은 예외를 잡는 것은 프로그램의 충돌로 이어질 수 있으며 결함을 가질 가능성을 더 높인다[6][7].

이러한 문제를 다루기 위해, 예외 처리를 지원하기 위한 기존의 연구들은 잘못된 예외 처리 정책을 위반한 코드를 탐지하거나[8] 예외의 처리 코드를 제안하려고 시도한다[9][10][11][12]. Montenegro et al.[8]는 예외 처리에 대한 룰을 정의하고 작성중인 코드를 분석해 예외 처리 룰에 위반하는 코드를 탐지한다. Nguyen et al.[10]은 룰 기반의 퍼지 논리를 통해 처리되어야 할 예외 유형을 제안하고 n-gram 기반의 접근 방법으로 예외 처리 코드를 제안한다. Zhang et al.[12]는 신경망 기반의 접근 방법으로 예외 처리가 필요한 try 블록의 위치를 지정하고 예외 처리 코드를 생성한다. 하지만, 기존 연구들은 두 가지 한계가 존재한다. 첫 번째, 룰 기반의 접근 방식은 try 블록에 등장하는 함수와 처리할 예외가 맵핑된다고 가정하지만, Python에서는 함수를 변수로 명명할 수 있기 때문에 등장하는 모든 변수에 대한 룰을 정의할 수 없다. 두 번째, 예외 처리 코드의 제안 및 생성은 개발자가 잡아야 할 예외를 인지하고 있다고 가정하지만 실제로 항상 그렇지는 않다[4][5]. 따라서, 예외 처리에서 잡아야 할 다수의 예외를 인지하는 것이 선행되어야 한다.

이 논문에서는, 예외 처리에 대한 지원을 제공하는 예외 제안 접근 기법을 제시한다. 먼저 우리는 방대한 양의 AI 프로젝트를 수집하고 코드 저장소로부터 예외 처리와 관련된 소스 코드를 추출한다. 다음으로, 예외 처리문에 대해 문맥 정보를 학습하기 위한 사전 훈련 단계를 수행한다. 마지막으로, 훈련된 모델들을 통해 예외 유형을 학습하여 try 블록에 따라 처리되어야 할 예외 유형을 제안한다.

접근 방법의 평가를 위해 2개의 Framework로 구성된 프로젝트를 수집했으며, 평가 결과에 따르면 Area under the average Precision-Recall curve(AUPRC)가 각각 0.95, 0.93로 비교 모델들을 능가한다. 이는 우리의 접근 방식이 예외를 지원하는데 효과적임을 보여주고, 개발자들은 실행 없이 예외를 파악하고 처리할 수 있다.

Ⅱ. 관련 연구

이 절에서는 예외 처리를 지원하는 관련 연구에 대해 소개한다.

barbosa et al.[9]은 예외의 글로벌 콘텍스트를 인식하고 위반된 예외 처리를 해결할 수 있는 코드를 생성하는 휴리스틱 기반의 접근 방법을 제안했다. Rahman et al.[10]은 오픈 소스 코드 저장소에서 검색 API를 활용해 예외 처리 코드를 검색하여 예외를 제안하는 접근 방법을 제안했다. Nguyen et al.[11]은 예외 유형을 제안하고 예외 처리 코드를 호출하는 fuzzy 및 n-gram 기반의 접근 방법을 제안했다. Zhang et al.[12]은 예외 처리를 지원하기 위해 try 블록의 위치를 지정해주고 처리 코드를 생성한다. 이러한 접근 방법들은 예외 처리를 위해 개발자에게 도움이 될 수 있지만, 적절한 예외 유형을 제공하지 않으면 개발자는 처리 시나리오가 없는 예외를 포착하여 오류를 발생시킬 수 있다. 또한, Java 언어에 특화된 지원은 함수를 변수에 할당 가능한 Python에서 유용하지 않을 수 있다. 기존 연구들과 비교해 우리의 접근 방식은 언어 의존적이지 않으면서 처리되어야 할 예외 유형을 제안하는 접근 방법을 제안한다.

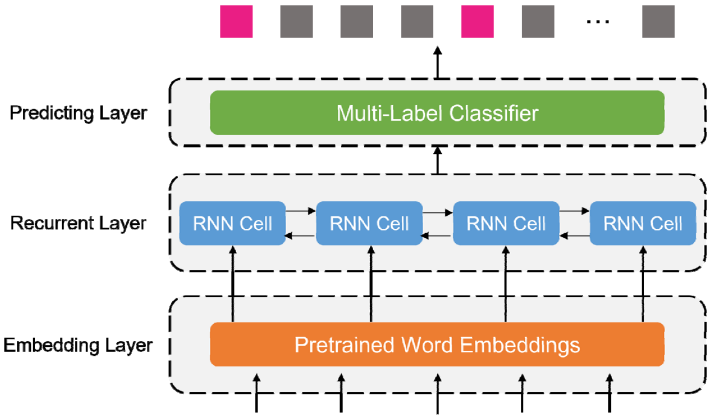


그림 1. 접근 방법의 전체 구조
Fig. 1. The overall structure of the approach

Ⅲ. 접근 방법

이 절에서 우리는 예외를 잡기 위한 우리의 자동화된 접근 방법을 자세히 설명한다. 먼저, 예외 제안을 위한 문제를 정의한다. 그런 다음 예외 제안 모델의 전체 아키텍처와 접근 방법의 각 구성요소를 소개한다. 제안된 접근 방법을 통해 개발자는 작성한 소스 코드로부터 처리되어야 할 예외 유형을 제안받을 수 있다.

1. Formulation

예외 유형 제안 문제에는 일련의 소스 코드 조각이 있으며 각 소스 코드 조각에는 오류가 있는지 확인하기 위한 try 블록과 오류 발생 시 오류를 처리하기 위한 except 블록으로 구성된다. except 블록에는 처리하기 위해 잡을 예외 유형이 포함된다.

우리는 개발자가 예외 처리 중에 잡아야 할 예외 유형을 제안하는 것을 목표로 한다. 우리는 이 문제를 다음과 같이 공식화한다. 일련의 소스 코드 토큰 $S=\{s_1,s_2,\ldots,s_k\}$ 가 주어지면 개발자가 잡아야 할 예외 $E=\{e_1,e_2,\ldots,e_k\}$ 를 잡는다. 여기에서 k는 소스 코드 토큰 길이 및 잡아야 할 예외의 수를 나타낸다. try 블록 안에는 중첩된 try 블록이 포함될 수 있으며, try 블록마다 대응되는 except 블록이 존재할 수 있지만 이 작업에서는 try 내의 중첩된 try-except 블록은 고려하지 않는다. 하지만, 개발자는 동일한 방법으로 제안 방법을 적용하여 중첩된 try-except 문에 대해 잡아야 할 예외를 제안받을 수 있다.

이 작업에서 우리는 소스 코드 조각에서 잡아야 할 예외 유형을 제안하는 문제를 다중 라벨 분류 문제로 처리한다. 소스 코드 토큰 $S=\{s_1,s_2,\ldots,s_k\}$와 잡아야 할 예외 $E=\{e_1,e_2,\ldots,e_k\}$가 주어지면 분류 모델 f는 $f(S)=\{e_1^{'},e_2^{'},\ldots,e_k^{'}\}$와 $y_k\in\{0,1\}$ 사이의 손실을 최소화하도록 학습한다. 학습된 모델은 예외를 제안하는 데 사용될 수 있다. 이 작업은 다음과 같이 공식화할 수 있다. 여기에서 θ는 모델 파라미터를 나타낸다.

$$f(S,\theta)=y \tag{1}$$

2. Model Architecture

이 절에서는 우리가 제안하는 접근 방법의 전체 프레임워크를 소개한다. 그림 1과 같이 제안 방법은 3개의 계층으로, Embedding Layer, Recurrent Layer, Predicting Layer 3개의 계층으로 구성된다. Embedding Layer는 사전 훈련 단계에서 문맥 정보를 학습하고, 예외 제안 단계에서 일련의 소스 코드 시퀀스를 입력으로 받아 상위 계층에서 처리 가능하도록 텍스트 형태의 소스 코드 토큰을 단어 벡터로 맵핑한다. Recurrent Layer는 코드 토큰 벡터 시퀀스를 순방향과 역방향 정보를 모두 고려하여 입력된 소스 코드에 대한 벡터 표현을 생성한다. 마지막으로, Predicting Layer는 이전 계층에서 표현된 벡터를 통해 잡아야 할 예외들의 확률을 계산한다. 다음 하위 절에서 각 계층에 대해 상세히 설명한다.

3. Embedding Layer

이 계층에서 사용되는 Pretrained Word Embedding은 소스 코드 토큰을 임베딩하여 단어 벡터로 표현하는 역할을 수행한다. 단어 임베딩을 수행하는 BERT와 ELMO 같은 사전 훈련된 모델들은 여러 자연어 처리 작업에서 성능을 향상시켰다. 특히, Feng et al.[13]은 BERT 기반의 모델을 통해 소스 코드를 사전 훈련하여 코드 검색 및 코드 생성에 높은 성능을 달성할 수 있음을 보였다. 따라서, 우리는 RoBERTa [14]를 활용하여 소스 코드를 사전 학습하고 임베딩한다. 이 작업에서는 Masked language modeling이라는 비지도 학습을 이용해 방대한 양의 try-except 코드 토큰에 대해 사전 훈련하고 임베딩을 수행한다. 사전 훈련된 단어 임베딩은 입력된 각 소스 코드 토큰에 문맥 정보를 반영하여 단어 벡터에 맵핑한다.

4. Recurrent Layer

Recurrent Layer는 Embedding Layer에서 출력된 단어 벡터 시퀀스를 사용해 순서 정보를 고려한 새로운 벡터를 생성한다. Word Embedding은 토큰간의 문법적인 특징과 문맥 정보를 포착할 수 있지만, try-except 소스 코드 토큰에 대한 의미적인 정보를 반영하지 못할 수 있다. 따라서, 우리는 이전 단계에서 출력된 단어 벡터에 대해 RNN(Recurrent Neural Network)을 활용하여 순서 정보를 고려한 소스 코드 토큰 벡터를 생성한다. 이 작업에서, 우리는 RNN의 변형인 Bi-LSTM[14]을 사용한다. 순방향의 순서 정보만 고려하는 RNN과는 달리, Bi-LSTM은 순방향 및 역방향의 순서 정보를 모두 고려하여 긴 소스 코드 토큰에 대해 정보량 손실을 최소화한 벡터 표현을 생성한다.

5. Predicting Layer

이전의 계층들은 일련의 소스 코드 토큰을 입력받아 예외를 표현하는 단어 벡터인 $H_k \in R^{d_n}$로 변환한다. Predicting Layer에서는 H를 입력으로 사용한 이진 분류기를 활용해 예외 유형 중 잡아야 할 예외 유형을 예측한다. 이 작업에서 신경망 기반 분류 작업에 널리 사용되는 시그모이드를 적용한다. 이 계층에서는 입력 행렬 H가 주어지면 다음과 같이 확률 $y^{'}$를 계산한다.

$$y^{'} = sigmoid(S\theta + b) \in [0,1] \quad (2)$$

여기에서 $\theta \in R^{d_n}$와 b는 학습에 필요한 모델의 파라미터와 bias를 나타낸다. $y_k^{'}$는 잡아야 할 예외 e_k의 확률을 나타낸다. 모델 학습을 위해, 우리는 binary cross-entropy loss를 다음과 같이 정의하고 사용한다.

$$L(\theta, y, y^{'}) = -\frac{1}{N}\sum_{k=1}^{K}\left(y_k \log\left(y_k^{'}\right) + \left(1-y_k^{'}\right)\log\left(1-y_k^{'}\right)\right) \quad (3)$$

여기에서 θ는 학습되는 모델의 파라미터를 나타내고, N은 전체 훈련 데이터 셋의 개수를 나타낸다. 출력 결과가 $y^{'} > \delta$이면 1, 그렇지 않으면 0으로 예측한다. 여기에서 δ는 $threshold$를 나타낸다. 우리는 분류 작업에서 일반적으로 사용되는 $\delta = 0.5$로 설정했다.

Ⅳ. 실험 설정

이 절에서는 실험을 위한 데이터 셋을 수집하고 실험을 구현하기 위한 상세 설정을 소개한다.

표 1. 연구 데이터 셋
Table 1. Study dataset

Framework	# Paper	# Project	# Leaf Exception
Pytorch	40,955	19,791	58,067
Tensorflow	24,341	12,980	30,385

1. Study Subject

접근 방법 평가에 사용할 데이터 셋을 수집하기 위해, 먼저 Paper With Code(https://paperswithcode.com)에서 코드와 함께 포함된 AI 관련 논문을 크롤링했다. Paper With Code는 학회 또는 저널에 등재된 AI 관련 논문, 구현 코드 URL, 데이터를 포함하는 오픈 리소스이다. Paper With Code에서는 고품질 데이터를 보장하기 위해 커뮤니티 참여자가 데이터를 검토하고 모니터링한다. 우리는 논문 구현 코드를 포함하고 있는 AI 논문에서 Github repository URL을 추출하고, Github repository로부터 프로젝트를 수집했다. 표 1은 연구에 사용된 프로젝트에 대한 통계를 보여준다. 표의 각 열은 첫 번째 열부터 프레임워크 종류, 논문 수, 프로젝트 수, 추출된 하

표 2. 예외 제안의 성능 비교
Table 2. Performance Comparison of Exception Recommendations

(a) Pytorch

Models	Metrics				
	Precision	Recall	F1-score	AUPRC	Precision@1
BiLSTM	0.8089±0.017	0.7723±0.017	0.7902±0.017	0.8454±0.011	0.8089±0.017
Transformer	0.8406±0.011	0.8126±0.011	0.8264±0.011	0.8989±0.011	0.8411±0.011
CodeBERT	0.7948±0.013	0.7654±0.013	0.7798±0.018	0.8140±0.334	0.7948±0.013
PT CodeBERT	0.8601±0.019	0.8288±0.019	0.8441±0.019	0.9054±0.019	0.8611±0.019
Ours	0.8840±0.013	0.8524±0.019	0.8779±0.013	09295±0.013	0.8949±0.012

(b) Tensorflow

Models	Metrics				
	Precision	Recall	F1-score	AUPRC	Precision@1
BiLSTM	0.5372±0.009	0.5201±0.009	0.5285±0.011	0.5806±0.011	0.5372±0.011
Transformer	0.8352±0.004	0.8210±0.003	0.8285±0.004	0.8950±0.002	0.8357±0.003
CodeBERT	0.4701±0.021	0.4551±0.021	0.4625±0.022	0.4947±0.019	0.4701±0.022
PT CodeBERT	0.8563±0.014	0.8291±0.015	0.8357±0.016	0.9001±0.016	0.8505±0.019
Ours	**0.8689±0.009**	**0.8694±0.006**	**0.8641±0.006**	**0.9298±0.014**	**0.8632±0.008**

위 예외 수를 나타낸다. 우리는 7개의 프레임워크로 실험된 67,576개의 논문을 수집하였으나 두(Pytorch, Tensorflow) 프레임워크로 작성된 논문이 전체의 96.6%를 차지하기 때문에 해당 프로젝트에 집중한다. 실험에 사용된 샘플은 일반적인 상위 예외의 제안을 방지하기 위해, Python에서 지원하는 예외(https://docs.python.org/3/library/exceptions.html) 중 코드와 예외를 맵핑시킬 수 있는 하위 예외(Leaf Exception)를 사용한다.

2. Implementation and Training

우리는 접근 방법을 구현하기 위해 Pytorch 프레임워크를 사용했다. 워드 임베딩을 위해 사전 학습에 사용된 모델은 RoBERTa[14] 아키텍처가 활용되었다. RNN Layer에서 사용된 Bi-LSTM은 1개의 계층으로 구성되었으며, hidden states는 256으로 설정했다. 이 작업에서, 입력 시퀀스의 최대 길이를 256으로, 차원은 200으로 설정했다. 출력을 위한 Predicting layer는 fully connected layer로 구성되며 hidden states는 256으로 설정했다. 출력된 값의 손실을 계산하기 위한 손실 함수로 우리는 Binary Cross Entropy를 활용한다.

모델 학습을 위해, 우리는 Adam[16] optimizer를 사용하고 batch size는 64로 설정했다. 최대 40 Epoch 동안 모델을 훈련한 후, 가장 높은 Area Under the Precision-Recall Curve (AUPRC) 값을 갖는 모델을 선택하여 평가를 진행했다. 접근 방법의 유효성을 검증하기 위해, 모든 학습 및 테스트 데이터 셋에 대해 stratified 10-fold cross validation을 수행했다. 평가 항목으로는 분류 작업에서 널리 사용되는 Precision, Recall, F1-score과 다중 분류 작업에서 널리 사용되는 Precision@1, AUPRC[17]을 사용했다.

3. Model Comparisons

본 연구에서는 제안 방법의 성능을 기존 모델과 비교하여 확인한다. 이 작업에서, 예외 제안을 위한 학습 기반의 기존 연구가 없었기 때문에 제안 방법의 구성 요소로 사용된 Bi-LSTM[15]과 BERT[13]가 비교 모델로 포함된다. 추가적으로, BERT의 구성 요소를 이루는 Transformer[18]가 비교 모델로 사용된다. 해당 모델들은 소스 코드의 분류 작업에서 널리 사용된다[19][20][21].

V. 실험 결과

이 절에서는 하위 예외 유형 제안을 위한 접근 방법을 평가하는 실험을 수행한다. 제안된 모델이 얼마나 효과적인지 조사하고, 접근 방식이 비교 모델에 비해 얼마나 높은 성능을 달성할 수 있는지 확인한다.

실험 결과는 표 2에 자세히 나타나 있다. 예외 제안을

가장 잘 수행하는 결과는 굵게 표시된다. 각 프레임워크에 대해 우리 접근 방법은 5가지 평가 메트릭에서 모든 비교 모델을 능가하는 것을 보여준다. 이는 예외 유형 제안 모델로 Pretrain Word Embedding 모델과 Bi-LSTM모델을 단독적으로 활용하는 것보다 조합하여 함께 사용하는 것이 성능 향상에 도움을 주는 것을 보여준다. 단독적으로 사용된 Bi-LSTM은 모든 경우에서 Transformer보다 좋지 않은 성능을 보여준다. 이는 예외 작업에서 의미 있는 토큰에 더욱 집중하는 것이 효과적임을 보인다. 특별하게, (a)에서 방대한 양의 Python 코드를 통해 사전 학습 되어있는 CodeBERT의 경우, try-except문에 대해 사전 학습 없이 사용하는 것은 모든 평가 메트릭에서 가장 좋지 않은 성능을 보여준다. 이러한 결과는 일반적인 Python 프로그래밍과 AI를 위한 프로그래밍에 차이가 존재하는 것으로, AI 프로젝트에서 예외 제안을 위해서는 일반적인 프로그래밍 언어 토큰의 제안과 달리 AI 예외 관련 토큰을 학습할 수 있는 기술을 구축할 필요가 있음을 보여준다. 이는 (b)에서도 동일한 경향을 보여준다

VI. 결 론

우리는 개발 중인 개발자에게 처리해야 할 적절한 예외를 제안해주는 것을 목표로 한다. 우리는 이 작업을 다루기 위해 다중 분류 작업으로 예외문제를 처리한다. 접근 방법은 소스 코드를 토큰 단위로 분할하고 사전 학습된 표현을 기반으로 잡아야 할 예외를 예측한다. 우리의 접근 방식은 2개의 프레임워크로 구성된 대규모 프로젝트에 대해 평가한다. 결과는 우리의 접근 방법이 비교 모델들을 큰 차이로 능가한다는 것을 보여준다. 향후 연구로는, 정확한 예외를 예측하기 위해 try 블록 앞의 소스 코드까지 고려할 수 있는 접근 방법을 연구할 계획이다.

References

[1] Do-Young Lee et al., "The effects of middle school mathematical statistics area and python programming steam instruction on problem solving ability and curriculum interest", Journal of the Korea Academia-Industrial cooperation Society, Vol. 20, No. 4, pp. 336-344, 2019.
DOI: https://doi.org/10.5762/KAIS.2019.20.4.336

[2] Kyungmin Park et al., "Exercise healthcare program using artificial intelligenmce", The Journal of KIIT, pp. 607-609, 2022.

[3] Nguyen, Giang, et al., "Machine learning and deep learning frameworks and libraries for large-scale data mining: a survey", Artificial Intelligence Review Vol. 52. No. 1, pp. 77-124, 2019.
DOI: doi.org/10.1007/s10462-018-09679-z

[4] ASADUZZAMAN, Muhammad, et al. "How developers use exception handling in Java?", 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories, pp. 516-519, 2016.
DOI: 10.1145/2901739.2903500

[5] ZHANG, Ru, et al. "An empirical study on program failures of deep learning jobs", 2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE), pp. 1159-1170, 2020.
DOI: 10.1145/3377811.3380362

[6] DE SOUZA COELHO, R., et al. "Exception handling bug hazards in Android: Results from a mining study and an exploratory survey", Empirical Software Engineering, pp. 1264-1304, 2017.
DOI:doi.org/10.1007/s10664-016-9443-7

[7] MARINESCU, C, "Should we beware the exceptions? an empirical study on the eclipse project", 2013 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. pp. 250-257, 2013.
DOI: 10.1109/SYNASC.2013.40

[8] MONTENEGRO, Taiza. et al., "Improving developers awareness of the exception handling policy", 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering, pp. 413-422, 2018.
DOI: 10.1109/SANER.2018.8330228

[9] Barbosa, E. A., et al, "Global-aware recommendations for repairing violations in exception handling", IEEE Transactions on Software Engineering, Vol. 44, No. 9, pp. 855-873, 2018. DOI: 10.1109/TSE.2017.2716925

[10] Rahman, M. M., & Roy, C. K., "On the use of context in recommending exception handling code examples", International Working Conference on Source Code Analysis and Manipulation, pp. 285-294, 2014.
DOI: 10.1109/SCAM.2014.15

[11] NGUYEN et al, "Code recommendation for exception handling", Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1027-1038, 2020.
DOI: doi.org/10.6084/m9.figshare.12433667.v1

[12] ZHANG, Jian, et al. "Learning to handle exceptions",International Conference on Automated Software Engineering (ASE). pp. 29-41, 2020.
DOI: dl.acm.org/doi/10.1145/3324884.3416568

[13] Feng et al., "CodeBERT: A Pre-Trained Model for

Programming and Natural Languages", In Findings of the Association for Computational Linguistics: EMNLP, pp. 1536-1547, 2020.
DOI: 10.18653/v1/2020.findings-emnlp.139

[14] LIU, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach". arXiv preprint arXiv:1907.11692, 2019.

[15] SCHUSTER et al, "Bidirectional recurrent neural networks", IEEE transactions on Signal Processing, Vol. 45, No. 11, pp. 2673-2681, 1997.
DOI: 10.1109/78.650093

[16] KINGMA et al., "Adam: A method for stochastic optimization", International Conference on Learning Representations (ICLR), 2015.

[17] DAVIS et al. "The relationship between Precision Recall and ROC curves", In Proceedings of the 23rd international conference on Machine learning, pp. 233-240, 2006.
DOI:doi.org/10.1145/1143844.1143874

[18] VASWANI, Ashish, et al., "Attention is all you need", Advances in neural information processing systems, Vol. 30, pp. 5998-6008, 2017.

[19] UDDIN, Md Nasir, et al., "Software defect prediction employing BiLSTM and BERT-based semantic feature", Soft Computing, pp. 1-15, 2022.
DOI: DOI:10.1007/s00500-022-06830-5

[20] PAN, Cong et al., "An empirical study on software defect prediction using codebert model", Applied Sciences, Vol. 11, No. 11, pp. 4798, 2021.
DOI: https://doi.org/10.3390/app11114793

[21] ZHANG, Qihang et al.,"Software defect prediction via transformer", 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference, pp. 874-879, 2020.
DOI: 10.1109/ITNEC48623.2020.9084745

저 자 소 개

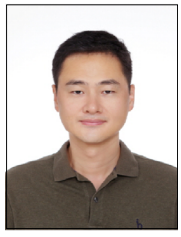
강 민 구(준회원)

- 2020. 2 ~ : 전북대학교 석사 재학
- 2020 : 전북대학교 학사
- Email : 201314090@jbnu.ac.kr

김 순 태(정회원)

- 2014 ~ : 전북대학교 소프트웨어공학과 교수
- 2010 : 서강대학교 공학박사
- 2007 : 서강대학교 공학석사
- Email : stkim@jbnu.ac.kr

류 덕 산(정회원)



- 2018 ~ : 전북대학교 소프트웨어공학과 조교수
- 2016 : KAIST 공학박사
- 2012 : KAIST 및 CMU 공학석사
- Email : duksan.ryu@jbnu.ac.kr

※ 이 성과는 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임. (NO.2020R1F1A1072039)
※ 이 논문은 2022년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(NRF-2022R1I1A3069233)