KIPS Trans. Softw. and Data Eng.
Vol.10, No.9 pp.339~348
ISSN: 2287-5905 (Print), ISSN: 2734-0503 (Online)

https://doi.org/10.3745/KTSDE.2021.10.9.339

Bayesian Optimization Framework for Improved Cross-Version Defect Prediction

Jeongwhan Choi† · Duksan Ryu††

ABSTRACT

In recent software defect prediction research, defect prediction between cross projects and cross-version projects are actively studied. Cross-version defect prediction studies assume WP(Within-Project) so far. However, in the CV(Cross-Version) environment, the previous work does not consider the distribution difference between project versions is important. In this study, we propose an automated Bayesian optimization framework that considers distribution differences between different versions. Through this, it automatically selects whether to perform transfer learning according to the difference in distribution. This framework is a technique that optimizes the distribution difference between versions, transfer learning, and hyper-parameters of the classifier. We confirmed that the method of automatically selecting whether to perform transfer learning based on the distribution difference is effective through experiments. Moreover, we can see that using our optimization framework is effective in improving performance and, as a result, can reduce software inspection effort. This is expected to support practical quality assurance activities for new version projects in a cross-version project environment.

Keywords : Software Defect Prediction, Bayesian Optimization, Transfer Learning, Cross-Version Defect Prediction

향상된 교차 버전 결함 예측을 위한 베이지안 최적화 프레임워크

최 정 환† · 류 덕 산††

요 약

최근 소프트웨어 결함 예측 연구는 교차 프로젝트 간의 결함 예측뿐만 아니라 교차 버전 프로젝트 간의 결함 예측 또한 이루어지고 있다. 종래의 교차 버전 결함 예측 연구들은 WP(Within-Project)로 가정한다. 하지만, CV(Cross-Version) 환경에서는 프로젝트 버전 간의 분포 차이의 중요성을 고려한 연구들이 없다. 본 연구에서는 다른 버전 간의 분포 차이까지 고려하는 자동화된 베이지안 최적화 프레임워크를 제안한다. 이를 통해 분포 차이에 따라 전이 학습(Transfer Learning) 수행 여부를 자동으로 선택하여 준다. 해당 프레임워크는 버전 간의 분포 차이, 전이 학습과 분류기(Classifier)의 하이퍼파라미터를 최적화하는 기법이다. 실험을 통해 전이 학습 수행 여부를 분포차 기준으로 자동으로 선택하는 방법이 효과적이라는 것을 알 수 있다. 그리고 최적화를 이용하는 것이 성능 향상에 효과가 있으며 이러한 결과 소프트웨어 인스펙션 노력을 감소할 수 있다는 것을 확인할 수 있다. 이를 통해 교차 버전 프로젝트 환경에서 신규 버전 프로젝트에 대하여 효과적인 품질 보증 활동 수행을 지원할 것으로 기대된다.

키워드 : 소프트웨어 결함 예측, 베이지안 최적화, 전이학습, 교차 버전 결함 예측

1. 서 론

소프트웨어 결함 예측 연구는 최근 활발하게 이루어지고 있는 소프트웨어 공학 연구 분야 중 하나이다. 소프트웨어 결함 예측은 소프트웨어 데이터를 기계학습 기법에 적용하여 결함이 발생하기 쉬운 소프트웨어 모듈들을 효과적으로 찾는다. 이를 통해 소프트웨어의 품질 보장과 코드 리뷰 혹은 테스팅에 대한 리소스 할당을 효과적으로 할 수 있다.

이미 배포된 소프트웨어 프로젝트의 경우 WP(Within-Project) 환경에서 다수의 버전을 포함하고 있고 충분한 데이터를 가지고 있을 수 있다. 그렇지만 프로젝트 초기 단계에서는 데이터를 충분히 가지고 있지 않다. 다른 프로젝트의 충분한 데이터를 사용하기 위해 CP(Cross-Project) 환경에서

※ 본 연구는 원자력안전위원회의 재원으로 한국원자력안전재단의 지원을 받아 수행한 원자력안전연구사업(No. 2105030)과 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2019R1G1A1005047)을 받아 수행된 연구사업의 결과임.
※ 이 논문은 한국정보처리학회 소프트웨어공학연구회가 주관한 2021 한국소프트웨어공학 학술대회(KCSE 2021)의 우수논문으로 "교차 버전 결함 예측을 위한 베이지안 최적화 프레임워크"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 연세대학교 인공지능학과 석 · 박사통합과정
†† 종신회원 : 전북대학교 소프트웨어공학과 조교수
Manuscript Received : May 15, 2021
First Revision : June 23, 2021
Accepted : July 8, 2021
* Corresponding Author : Duksan Ryu(duksan.ryu@jbnu.ac.kr)

는 프로젝트 간의 분포 차이를 줄여주는 전이 학습(Transfer Learning)이 해결책이 된다.

최근에는 WP 환경에서 같은 프로젝트의 이전 버전들을 사용하여 신규 버전의 결함을 예측하는 연구가 대두되었다. 이를 CV(Cross-Version) 환경이라 하며 종래의 결함 예측 연구들은 버전 간의 분포 차이를 고려하지 않는다. CV 환경 또한 CP와 마찬가지로 다른 버전 간의 분포가 결함 예측 성능에 영향을 미치는지 규명이 되어있지 않기에 본 연구에서는 분포차이가 중요한지 규명하고자 한다.

본 연구에서는 이전 연구의 확장 연구이며, 다른 버전 간의 분포 차이까지 고려하는 자동화된 CVDP(Cross-Version Defect Prediction)용 베이지안 최적화 프레임워크를 제안한다[15]. 이를 통해 분포 차이에 따라 전이 학습 수행 여부를 자동으로 선택하여 준다. 해당 프레임워크는 버전 간의 분포 차이, 전이 학습과 분류기(Classifier)들을 모두 베이지안 최적화(Bayesian optimization)를 통해 성능을 높인다.

Tantithamthavorn et al.의 연구에 따르면 CPDP(Cross-Project Defect Prediction) 기법들 중 85%가 최소 하나 이상의 파라미터가 전이학습 단계에서 필요하다[1,2]. 그리고 결함 예측에서의 분류기들 중 87%가 최소 하나 이상의 파라미터가 사용된다. 본 연구에서는 전이학습, 분류기, 분포 차이 분석 알고리즘에 대한 모든 하이퍼파라미터를 고려한다.

본 연구에서 제안하는 CVDP를 위한 베이지안 최적화 프레임워크는 실험을 통해서 모든 데이터셋에 대해 성능 향상 효과가 있다. 이 프레임워크를 통해 보여주는 결과는 소프트웨어 인스펙션 노력 비용을 감소할 수 있음을 보여주며 실험 결과 분포차이를 CV 환경에서도 고려하는 것이 성능 향상에 도움이 되는 것을 보여준다. CV 환경에서 신규 버전 프로젝트에 대한 결함 예측을 통해 소프트웨어 품질 보증 활동 수행을 효과적으로 지원할 것으로 기대된다.

2. 관련 연구

Turhan et al.[16]이 제안한 NNFilter는 유클리드 거리를 기준으로 타겟 프로젝트 데이터의 각 인스턴스에 대한 교차 프로젝트 데이터에서 근접 이웃들을 선택한다. Nam et al.[17]은 TCA[18] 방법을 CPDP에 성공적으로 적용했다. TCA는 소스 프로젝트를 선택하는 대신, 타겟 프로젝트와의 분포가 가까운 공유 잠재 공간(Shared Latent Space)에 직접 매핑한다. CP 환경에서는 분포 차이를 극복하고자 하는 연구들이 많은 반면에 CV 환경에서는 분포 차이가 중요한 요소로 작용되는지 고려한 연구가 존재하지 않는다.

CVDP에 대한 최근 연구로는 S.Amasaki의 연구가 있다[3,4]. 이 연구는 전이 학습이 CP 환경이 아닌 CV 환경에서도 적용되는지를 연구하였다. 하지만, 우리의 연구와는 다르게 이 연구는 버전 간의 분포 차이에 대한 분석이 없으며 이를 고려하지 않았다. 다른 CVDP 연구들에 대해서도 버전 간의 분포 차이에 대한 고려가 이루어지지 않는다. 특히 X.Yang et al. 의 연구에서는 CV 환경에서 회귀 문제를 다루는데 이 연구 또한 버전 간의 분포를 확인하지 않은 채, CV 환경이 WP 환경과 유사하다고 가정한다[5].

소프트웨어 결함 예측 연구에서 하이퍼파라미터 최적화를 진행하는 연구들은 CP 환경에서 진행이 되어왔다[6,7]. 우리의 연구는 CV 환경에서 전이학습과 분류기와 분포 차이를 확인하는 알고리즘 또한 파라미터화하였다.

3. 연구 방법

아래 Fig. 1은 본 연구의 CVDP를 위한 베이지안 최적화 프레임워크이다. 베이지안 최적화 프레임워크는 CVDP 단계가 포함되어 있다. CVDP 단계에서 최신 버전을 타겟 버전

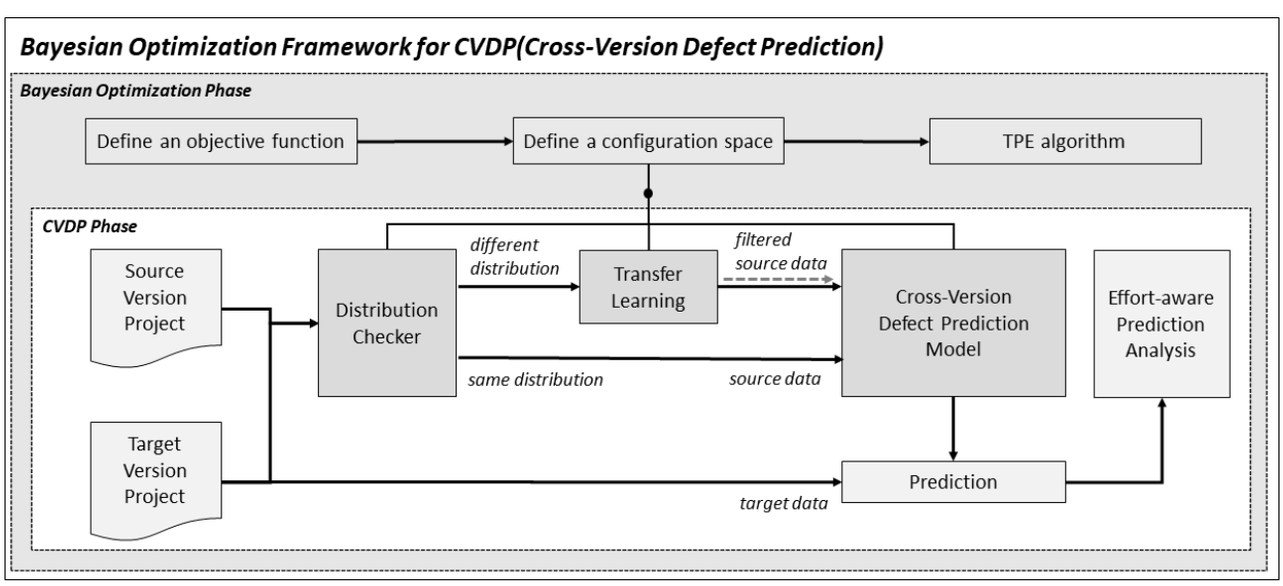


Fig. 1. Bayesian Optimization Framework for Improved Cross-Version Defect Prediction

프로젝트, 이전 버전을 소스 버전 프로젝트로 설정한다. 타겟과 소스 버전 프로젝트는 일대일 관계를 가진다. 두 버전에 대한 분포 차이를 KS 테스트 (Kolmogorov-Smirnov Test)를 사용하여 확인한다. KS 테스트는 각 데이터셋에서 각각의 칼럼 혹은 피쳐에 대한 분포를 통계적으로 분석한다. KS 테스트 결과 분포가 다른 칼럼이 우리가 설정한 임계값 보다 크면 전이 학습을 진행하도록 한다. 반면에 임계값 보다 작으면 전이 학습 없이 진행한다.

선택적 전이 학습을 통해 타겟 버전 프로젝트의 분포와 유사한 인스턴스들로 필터링된 데이터들이 CVDP 모델의 트레이닝셋으로 사용된다. 그리고 기존 타겟 버전 프로젝트의 데이터는 테스트셋이다. 타겟 버전에 대한 예측 성능 평가를 AUC(Area Under Curve)로 진행한다. 그리고 소프트웨어 인스펙션 감소 비율에 대해서 분석한다.

베이지안 최적화 프레임워크의 최적화 단계에서는 CVDP 단계의 세 가지 요소에 대한 하이퍼파라미터들의 탐색 공간(search space)를 정의한다. 그리고 정의된 목적 함수(obejective function)에 의해 가장 최적의 하이퍼파라미터들을 찾는다. 최적화 알고리즘은 hyperopt 라이브러리에서 제공하는 TPE (Tree-of-Parzen-Estimators)를 사용한다.

3.1 Distribution Checker

두 데이터 간의 분포를 확인하기 위해서 KS 테스트 (Kolmogorov-Smirnov Test)를 사용한다. KS 테스트는 데이터셋의 각 칼럼 간 분포를 비교할 수 있다. 귀무가설은 “두 데이터의 분포는 같다”로 설정되며 p-value가 0.05 보다 작을 경우 이 귀무가설을 기각할 수 있어 대립가설인 “두 데이터의 분포는 다르다”를 수용할 수 있다. 본 연구에서 분포 차이를 확인하기 위해 기각된 칼럼의 개수를 이용한다.

3.2 Transfer Learning

전이학습(Transfer Learning)은 TCA(Transfer Component Analysis)와 NNFilter(Nearest Neighbor Filter), DSNF (Data Selection and Nearest Neighbor Filter) 사용된다. TCA는 소스 데이터를 선택하는 대신 소스 데이터와 타겟 데이터 프로젝트의 분포가 가까운 공유된 잠재 공간에 직접 매핑하는 방법이다[8]. NNFilter는 타겟 데이터의 각 인스턴스에 대한 유클리드 거리를 기준으로 소스 데이터에서 근접 이웃 인스턴스 k개를 선택한다[9]. DSNF는 유클리드 거리를 사용하여 이상치를 제거하여 NNFilter를 수행하는 방법이다[10].

전이 학습을 통해 타겟 데이터 분포에 유사한 소스 데이터가 모델의 입력으로 들어간다. 전이 학습시에 라벨 정보는 사용되지 않으며 타겟 데이터는 테스트셋으로 사용된다.

3.3 Cross Version Defect Prediction Model

CVDP 모델의 분류기로는 BRF(Balanced Random Forest)를 사용한다. 이 BRF는 각 bootstrap 마다 샘플들을 임의로 언더샘플링(under-sampling) 한다. 이를 통해 클래스 불균형한 문제를 다룬다.

3.4 Bayesian Hyperparameter Optimization

베이지안 하이퍼파라미터 최적화를 위해 먼저 목적함수를 정의한다. 목적 함수는 지정된 loss 값을 최소화 하는 것이 목표이다. 본 연구에서 사용하는 평가 척도인 AUC를 목적함수의 loss로 사용한다. AUC는 높을수록 좋은 성능 지표를 나타내기에 목적 함수에서는 음수로 설정하여 사용한다. 탐색 공간에 대해서는 분포 확인 알고리즘, 전이 학습, 분류기들의 하이퍼파라미터를 사용한다. 그리고 최적화 알고리즘은 TPE(Tree-of-Parzen-Estimators)를 사용한다.

4. 실험 설계

4.1 연구 질문

본 연구에서는 실험 설계를 하기 위해 네 가지 연구질문을 가설과 함께 아래와 같이 설정하였다. RQ1을 제외한 나머지 연구질문들은 귀무가설과 대립가설을 포함하고 있다.

1) RQ1: CV 환경에서 버전 간 프로젝트들의 분포가 다른가?

CV 환경에서 두 버전에 대한 데이터의 다른 칼럼 분포를 KS 테스트를 통해서 확인한다. 각 칼럼에 대해 p-value를 통해 통계적으로 검정한다. Equation (1)과 같이 기각된 칼럼의 개수가 임계값(threshold) 보다 클 경우 분포가 다르다고 가정하고 전이 학습을 진행한다.

$$f(X_1, X_2) = \begin{cases} same, & n(rejected) < thres \\ diff, & n(rejected) \geq thres \end{cases} \quad (1)$$

2) RQ2: CV 환경에서 분포 차이에 따른 선택적 전이 학습을 적용할 경우 성능 향상이 있는가?

- H_0: CV 환경에서 분포 차이에 따른 선택적 전이 학습을 적용한 모델과 적용하지 않은 모델의 AUC 성능이 유사하다.
- H_A: CV 환경에서 분포 차이에 따른 선택적 전이 학습을 적용한 모델이 적용하지 않은 모델보다 AUC 성능이 높다.

분포 차이에 따른 선택적 전이 학습에 대한 성능 향상 효과를 확인한다. 전이 학습을 모두 사용한 모델과 분포 차이를 고려한 선택적 전이 학습 모델과 비교한다. AUC 성능 평가 척도를 사용하여 두 모델의 성능 차이를 실험한다. 해당 실험에서의 선택적 전이학습의 임계값은 1을 사용한다.

Table 1. Hyper-parameters Setting for Classifier

Classifier	Hyper-parameters	
	Name	Range
BRF	n_estimators (default=10)	[10, 100] [N]
	criterion (default='gini')	{'gini', 'entropy'} [C]
	max_features (default='auto')	{'auto','sqrt','log2'} [C]
	min_samples_split (default='2)	[2, #sources/10} [N]

Table 2. Hyper-parameters Setting for Transfer Learning

Classifier	Hyper-parameters	
	Name	Range
TCA	kernel (default='linear')	{'primal', 'linear', 'rbf', 'sam'} [C]
	dimension (default=5)	[5, max(#source, #target)] [N]
	lamb (default=1)	[0.000001, 100] [R]
	gamma (default=1)	[0.00001, 100] [R]
NNFilter	k (default=10)	[1,100] [N]
DSNF	Toprank (default=1)	[1,10] [N]
	k (default=25)	[1, 100] [N]

3) RQ3: 베이지안 하이퍼파라미터 최적화를 적용한 프레임워크의 성능 향상 효과가 얼마나 있는가?

- H_0: 베이지안 하이퍼파라미터 최적화를 적용한 모델과 최적화를 적용하지 않은 모델의 AUC 성능이 유사하다.
- H_A: 베이지안 하이퍼파라미터 최적화를 적용한 모델은 최적화를 적용하지 않은 모델보다 AUC 성능이 높다.

베이지안 최적화를 적용한 프레임워크가 최적화를 하지 않은 모델보다 성능 효과가 있는지 확인한다. 최적화를 적용하지 않는 모델은 기본 값을 하이퍼파라미터들에 적용한다. 그리고 최적화를 진행하는 모델은 휴리스틱 방법으로 일정 범위를 지정한다. 본 연구에서 사용되는 분류기와 전이학습의 하이퍼파라미터들은 Table 1, Table 2와 같다. 범위 항목에서 [N]는 범위 내에서 정수 값을 가지는 것을 의미한다. [R]은 범위 내에서 실수 값을 가지며 [C]는 카테고리 목록에서 하나를 선택하는 것을 의미한다.

선택적 전이학습을 위한 분포 확인 알고리즘의 하이퍼파라미터는 분포 차이 여부를 결정하는 임계값(threshold)이며 범위는 [1,5]으로 설정하며 임계값은 정수값을 가진다.

4) RQ4: CVDP 모델에서 우리의 접근 방법이 최적화를 하지 않은 기본 모델과 대비하여 소프트웨어 인스펙션 노력을 감소시키는가?

- H_0: 본 연구에서 제안하는 모델은 베이스라인 모델과 대비하여 코드 인스펙션 노력 감소 비율이 유사하다.
- H_A: 본 연구에서 제안하는 모델은 베이스라인 모델과 대비하여 코드 인스펙션 노력 감소 비율이 크다.

Effort-aware 예측 분석은 소프트웨어 인스펙션 노력 비용을 얼마나 감소할 수 있는지에 대한 관점으로 모델을 평가한다. 파일 관점과 LOC(Line of code) 관점에서 코드 인스펙션 노력 비용 감소 비율을 확인할 수 있는 척도인 FIR과 LIR을 사용한다. 목차 4.3에서 두 평가 척도를 설명한다.

4.2 실험 환경

데이터셋은 Jureczko와 Madeyski가 에서 제안된 MORPH 데이터셋을 사용한다[11]. 이 데이터셋은 총 10개의 프로젝트로부터 소프트웨어 정적 메트릭들에 대한 피쳐들로 구성되어 있으며, 이 피쳐로 구성되어 있다. Table 3에서 결함 비율과 버전 개수를 알 수 있다. 실험은 Python 3.8 환경에서 진행되며 베이지안 하이퍼파라미터 최적화를 위해 Hyperopt 라이브러리를 사용한다[12].

본 연구의 CVDP 환경은 가장 최신의 버전이 타겟 버전 프로젝트로 고정된다. 그리고 Amasaki가 분류한 시나리오에서 하나의 초기 버전을 이용하는 SFV(Single Farthest Version)와 하나의 최근 버전을 이용하는 SPV(Single Prior Version) 뿐만 아니라 중간 버전의 프로젝트들에 대해서도 소스 데이터로써 사용된다[3]. 즉, 이전 버전들을 하나씩 소스 데이터로 사용하여 총 27가지의 조합으로 진행한다. 모든 실험 결과는 10번의 실험에 대한 평균값을 사용한다.

4.3 평가 척도

본 연구의 RQ2와 RQ3의 실험에서 성능 차이를 확인하기 위해 AUC(Area Under the Curve)를 평가척도로 사용한다. AUC는 ROC(Receiver Operating Characteristic) 곡선의 아래 면적을 의미한다. 모델이 옳은 예측을 할 수록 이 면적이 1에 가까워진다. AUC를 사용하는 이유는 불균형한 클래스 데이터셋에 민감하지 않기 때문이다[13].

Table 3. MORPH Dataset Description

Project	# of releases	Ave. % of defects
ant	5	19
camel	4	19
ivy	3	25
jedit	5	20
log4j	3	50
lucene	3	55
poi	4	50
synapse	3	24
velocity	3	58
xerces	4	38

RQ4에서는 FIR과 LIR 척도를 이용하여 소프트웨어 인스펙션 감소 비율을 분석한다. FIR(File Inspection Reduction)은 랜덤 선택과 비교하여 예측 모델을 사용하여 인스펙션 할 때, 동일한 PD(Probability of Detection)를 달성하기 위해 줄어든 인스펙션 할 파일 수의 비율이다[14]. FI(File Inspection)는 전체 파일에 대해 인스펙션 할 파일 수의 비율이고 FIR은 이와 같이 정의한다.

$$FIR = (PD - FI)/PD. \quad (2)$$

LIR(LOC Inspection Reduction)은 랜덤 선택과 비교하여 예측 모델을 사용하여 인스펙션 할 때, 동일한 PrD(Predictive Defectiveness)를 달성하기 위해 줄어든 LOC(Line of Code)의 비율이다[14]. LI(LOC Inspection)는 전체 LOC에 대해서 인스펙션 할 LOC의 비율이며 LIR은 다음과 같이 정의한다.

$$LIR = (PrD - LI)/PrD. \quad (3)$$

5. 실험 결과

5.1 RQ1: 프로젝트의 버전 간 데이터 분포 차이

Table 4는 프로젝트 버전 간의 분포 차이를 보여준다. 타겟 버전 프로젝트를 기준으로 이전 버전의 소스 프로젝트들에 대한 KS 테스트에 대한 결과이다. 각 칼럼들에 대한 p-value 값의 평균과 분포가 다른 칼럼의 개수를 보여준다. camel 프로젝트와 같은 경우 모든 버전 간의 분포 차이가 없다는 것을 Fig. 2에서도 확인할 수 있다. 반면에 Fig. 3에서도 poi 프로젝트 같은 경우 최대 9개의 칼럼에 대한 분포 차이가 있다는 것을 확인할 수 있다. 이처럼 CV 환경에서 프로젝트마다 분포 차이가 다르다는 것을 확인할 수 있다. 그리고 기각된 칼럼의 개수는 [0, 9]의 분포를 가진다. 해당 분포에 대한 중간값인 5를 RQ3에서 임계값 범위의 최대값으로 설정하여 사용한다.

5.2 RQ2: 분포 차이에 따른 선택적 전이 학습의 효과

분포 차이에 따른 선택적 전이 학습의 성능 향상에 대해서 실험한 결과이며 Table 5에서 모든 프로젝트에 대한 평균과 표준편차를 확인할 수 있다. 분포 차이를 고려하지 않고 전이 학습을 고려한 모델들을 베이스라인으로 설정하고 분포 차이를 고려한 선택적 전이 학습 모델을 비교 실험하여 이에 대한 효과를 확인한다. 이 실험에서의 전이 학습은 TCA, NNFilter와 DSNF를 사용하여 비교한다. Table 5에서 Default가 베이스라인 모델이며 Selective는 선택적 전이 학습을 의미한다.

이에 대한 실험 결과 대부분의 AUC 성능이 향상되었으며 이는 버전 간의 분포 차이에 따른 선택적 전이 학습을 반영한 것이 좋은 효과가 있다는 것을 알 수 있다. TCA의 경우 베이스

Table 4. Distribution Differences between Versions

Project	Target Version	Source Version	KS Test (p-value)	Different Columns
ant	1.7	1.3	0.56	3
		1.4	0.56	3
		1.5	0.48	2
		1.6	0.63	2
camel	1.6	1.0	0.45	0
		1.2	0.74	0
		1.4	0.97	0
ivy	2.0	1.1	0.64	0
		1.4	0.74	0
jedit	4.3	3.2	0.42	3
		4.0	0.45	2
		4.1	0.5	2
		4.2	0.54	0
log4j	1.2	1.0	0.71	1
		1.1	0.77	1
lucene	2.4	2.0	0.64	1
		2.2	0.68	2
poi	3.0	1.5	0.29	5
		2.0	0.29	9
		2.5	0.42	8
synapse	1.2	1.0	0.39	8
		1.1	0.92	0
velocity	1.6	1.4	0.6	1
		1.5	0.96	0
xerces	1.4	1.0	0.32	8
		1.2	0.78	2
		1.3	0.76	1

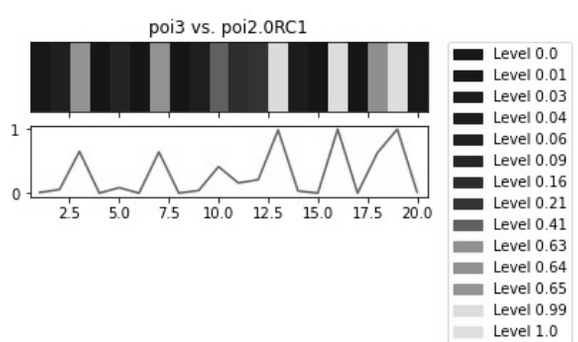


Fig. 2. Distribution Differences of the Poi Project

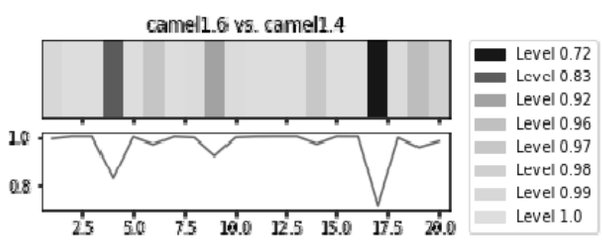


Fig. 3. Distribution Differences of the Camel Project

Table 5. AUC and Standard Deviation on the Selective Transfer Learning

	Default	Selective
TCA	0.5834±0.0911	0.6464±0.0865
NNFilter	0.5822±0.0876	0.6511±0.0948
DSNF	0.5799±0.0971	0.6673±0.0689

라인 모델의 AUC 평균과 표준편차는 0.5834와 0.0911이고 선택적 전이학습 모델을 적용하는 경우 0.6464와 0.0865이다. 선택적 전이 학습을 적용한 NNFilter의 경우 AUC 평균과 표준편차는 0.6511, 0.0948이다. DSNF는 베이스라인 모델의 AUC 평균과 표준편차는 0.5799와 0.0971이다. 선택적 전이학습을 적용할 경우 각 각 0.6673, 0.0689이다. 이에 대한 모든 프로젝트들에 대한 실험 결과는 Appendix의 Table 9에서 확인할 수 있다.

이 실험 결과에 따른 Cohen's D 효과 크기는 각 전이학습에 따라 0.71, 0.75, 1.05로 매우 크다는 것을 알 수 있다. 따라서 RQ1의 귀무가설을 기각할 수 있고 대립가설을 채택할 수 있다. CV 환경에서 선택적 전이 학습을 적용한 모델의 효과가 크다는 것을 확인할 수 있다. 전이 학습에 대한 분석에서는 TCA와 NNFilter의 경우 성능 향상에 대한 효과가 유사하다는 것을 확인할 수 있다. 반면에 DSNF의 경우 이상치에 대한 인스턴스들을 고려하고 NNFilter를 수행하기 때문에 두 전이학습들보다 성능 향상 효과가 크다.

또한 프로젝트마다 버전 간 분포 차이가 모두 다르다는 것을 RQ1에서 확인했으며 이를 반영한 것이 RQ2의 실험 결과이다. 즉 CV 환경이 WP와 CP 환경에 대해서 각각 선택적으로 반영할 수 있다. 이를 통해 선택적으로 전이 학습을 수행하는 것이 CV 환경에 적합하다는 것을 확인하였다.

5.3 RQ3: 베이지안 최적화 프레임워크의 효과

Table 6은 베이지안 최적화를 적용한 프레임워크가 최적화를 하지 않은 모델보다 성능 효과가 있는지 실험한 결과이며 평균과 표준편차를 보인다. Default 모델은 모든 하이퍼파라미터들에 대해서 기본 값을 할당해주고 Optimized 모델은 일정 범위 혹은 항목들을 할당해준다.

모든 실험에 대한 결과는 Appendix의 Table 10에서 확인할 수 있다. TCA를 이용한 실험 결과 최적화 프레임워크를 적용한 전후의 AUC 평균 값은 0.59와 0.67이다. NNFilter를 이용한 실험 결과 최적화 프레임워크를 적용한 전후의 AUC 평균 값은 0.61와 0.67이다. DSNF의 실험 결과는 0.54과 0.66이다. 대부분의 프로젝트들은 0.7에 가까운 AUC 성능을 보여주지만 xerces는 초기 버전이 소스 버전 프로젝트일 때 성능이 0.4 이하로 감소하는 것을 확인할 수 있다. DSNF를 이용할 경우에는 log4j 프로젝트에서 전이학습이 잘 이루어지지 않는 문제가 있다. DSNF의 경우 log4j 프로젝트만을 제외하면 평균 0.7이 넘는 성능을 보여준다.

Table 6. AUC and Standard Deviation on Optimization Framework

	Default	Optimized
TCA	0.5934±0.1026	0.6732±0.0925
NNFilter	0.6084±0.1046	0.6771±0.0963
DSNF	0.5453±0.2040	0.6590±0.1429

세 가지 전이 학습들의 실험 결과에 대한 Cohen's D 효과 크기는 각각 0.80, 1.17, 0.64으로 매우 크기에 RQ3의 귀무가설을 기각할 수 있다. 본 연구의 최적화 프레임워크가 효과적이라는 것을 확인할 수 있다. 세 가지 전이 학습들의 실험 결과에 대한 Cohen's D 효과크기는 각각 0.80, 1.17, 0.64으로 매우 크기에 RQ3의 귀무가설을 기각할 수 있다. 본 연구의 최적화 프레임워크가 효과적이라는 것을 확인할 수 있다. 그리고 전이 학습과 분류기, 분포 차이에 대한 하이퍼파라미터들이 성능 향상에 있어서 중요한 요소로 작용된다는 것을 확인할 수 있다.

5.4 RQ4: Effort-aware 예측 분석

Effort-aware 예측 분석 관점에서 본 연구에서 제안하는 프레임워크를 평가한다. Table 7과 Table 8은 FIR과 LIR에 대한 실험 결과(평균과 표준편차)를 나타낸다. 모든 프로젝트들에 대한 실험 결과는 Appendix의 Table11과 12에서 확인할 수 있다. TCA를 이용한 경우 최적화 이전의 평균 FIR은 0.23이며 베이지안 최적화 프레임워크를 적용하게 되면 평균 FIR은 0.3226으로 상승된다. 다른 전이 학습들에 대한 실험 결과 또한 상승되는 결과를 볼 수 있다. NNFilter의 경우 0.2793에서 0.3388으로 증가하고 DSNF는 0.2073에서 0.3693로 결과가 나온다. 이는 모듈 파일에 대한 인스펙션 노력을 30%에서 50% 까지 비용을 감소할 수 있다는 것이다.

각 전이 학습들의 FIR 값에 대한 Cohen's D 효과 크기는 0.39, 0.26, 0.72의 결과가 나온다. TCA와 NNFilter의 경우 효과크기가 중간 크기이지만 DSNF를 사용한 경우 효과크기가 크기에 귀무가설을 FIR 값 관점에서 기각할 수 있다.

Table 8에서 또한 평균 LIR 값이 최적화를 수행하면 상승하는 것을 확인할 수 있다. TCA의 경우 -1.04에서 -0.85,

Table 7. FIR and Standard Deviation on Optimization Framework

	Default	Optimized
TCA	0.2280±0.2640	0.2640±0.2026
NNFilter	0.2793±0.2250	0.3388±0.2163
DSNF	0.2073±0.2454	0.3693±0.1971

Table 8. LIR and Standard Deviation on Optimization Framework

	Default	Optimized
TCA	-1.0392±0.7975	-0.8527±0.6893
NNFilter	-1.0237±0.8443	-0.8127±0.7319
DSNF	-0.4031±0.4818	-0.2731±0.5101

NNFilter의 경우 -1.02에서 -0.81, DSNF의 경우 -0.40에서 -0.27로 상승하는 것을 확인할 수 있다. LIR 분석을 통해서는 Cohen's D 효과크기가 각각 0.25, 0.26, 0.26으로 효과크기가 중간 보다 작다. 이러한 효과크기 때문에 LIR 관점에서는 RQ4의 귀무가설을 약하게 기각할 수 있다.

FIR과 LIR을 통한 분석을 본 논문이 제안하는 CVDP를 위한 베이지안 최적화 프레임워크는 소프트웨어 인스펙션 노력 비용을 감소시킬 수 있음을 확인하였다.

6. 위협 요소

불균형한 클래스 데이터는 구성 유효성에 대한 위협이다. 이 위협을 줄이기 위해 불균형한 클래스 데이터에 적합한 평가 척도인 AUC를 사용하였다. 내부 유효성에 대한 위협은 최적화를 적용하지 않은 모델과 적용한 모델을 비교할 때 있을 수 있다. 최적화를 적용하지 않은 모델은 기본 하이퍼파라미터 값들을 정해주는데 이 결정에 의해 향상되는 크기가 달라질 수 있기 때문이다. 이러한 위협을 완화시키기 위해 관련 연구의 하이퍼파라미터 설정을 참고하였다.

7. 결 론

본 연구는 CP 환경에서의 전이 학습이 CV 환경에서 어떻게 적용되어야 할지 통찰력을 제공한다. CVDP의 관련연구들은 분포 차이를 고려하지 않은 채 CV 환경을 WP 환경 혹은 CP 환경으로 간주한다. 우리가 제안하는 프레임워크는 WP와 CP 환경 모두 선택적으로 고려할 수 있으며 분포 차이에 따라 전이 학습 수행 여부를 선택한다.

RQ1을 통해서 프로젝트마다 버전 간의 분포 차이가 다르다는 것을 확인할 수 있으며 이는 CVDP가 WP와 CP 환경이 혼합되어 있다는 것을 확인할 수 있다. 그리고 RQ2를 통해서 CV 환경의 분포 차이에 따른 선택적 전이 학습의 효과를 확인한다. 그리고 선택적 전이 학습이 CV 환경에 적합하다는 것을 통계적으로 검정하였다.

RQ3에서는 선택적 전이 학습이 포함된 CVDP를 위한 베이지안 최적화 프레임워크에 대해서 실험한다. 실험을 통해 베이지안 최적화의 성능 향상 효과를 통계적 검정으로 보인다. 마지막으로 RQ4에서는 Effort-aware 예측 분석을 통해서 소프트웨어 인스펙션에 대한 노력 비용을 감소할 수 있다는 점을 제공한다. 이 때 FIR과 LIR 분석을 통해 효과크기를 이용하여 통계적으로 검정한다. 이러한 결과 본 연구는 CV 환경에서 신규 버전 프로젝트들에 대해서 효과적인 소프트웨어 품질 보증 활동에 기여할 수 있다.

본 연구는 다른 다양한 분류기와 전이 학습 기법들을 베이지안 최적화 프레임워크에 확장하여 적용 가능하다. 향후 연구로 이 프레임워크를 확장할 계획이며 베이지안 최적화 뿐만 아니라 다른 최적화 알고리즘 또한 적용할 것이다.

References

[1] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Transaction on Software Engineering*, Vol.45, No.7, pp.683-711, Jul. 2019.

[2] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the International Conference on Software Engineering*, pp.321-332, May 2016.

[3] S. Amasaki, "Cross-version defect prediction: Use historical data, cross-project data, or both?," *Empirical Software Engineering*, Vol.25, No.2, pp.1573-1595, Mar. 2020.

[4] S. Amasaki, "Cross-version defect prediction using cross-project defect prediction approaches," in *Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering*, pp.32-41, 2018.

[5] X. Yang and W. Wen, "Ridge and lasso regression models for cross-version defect prediction," *IEEE Transaction on Reliability*, Vol.67, No.3, pp.885-896, Sep. 2018.

[6] K. Li, Z. Xiang, T. Chen, S. Wang, and K. C. Tan, "Understanding the automated parameter optimization on transfer learning for CPDP: An empirical study," in *Proceeding of the International Conference on Software Engineering*, 2020.

[7] K. Li, Z. Xiang, T. Chen, and K. C. Tan, "BiLO-CPDP: Bi-Level programming for automated model discovery in cross-project defect prediction," *Automated Software Engineering*, pp.1-24, Aug. 2020.

[8] N. Nagappan, T. Ball, and A. Zeller, "Mining metrics to predict component failures," in *Proceeding of the 28th International Conference on Software Engineering* - ICSE '06, pp.452, 2006.

[9] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, Vol.14, No.5, pp.540-578, 2009.

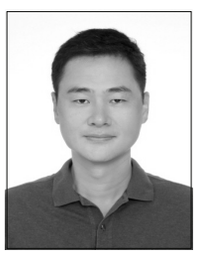
[10] S. Amasaki, K. Kawata, and T. Yokogawa, "Improving cross-project defect prediction methods with data simplification," *Proceeding of the 41st Euromicro Conference on Software Engineering and Advanced Applications*, SEAA 2015, pp.96-103, 2015.

[11] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the International Conference on Predictive Models in Software Engineering* - PROMISE '10, 2010.

[12] J. Bergstra, D. Yamins, and D. Cox, "Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in Science Conference (SCIPY 2013)*, pp.13-19, 2013.
[13] Z. Li, X. Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, Vol.12, No.3, pp.161-175, 2018.
[14] Y. Shin, A. Meneely, L. Williams, and J. A. Osborne, "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE Transaction on Software Engineering*, Vol.37, No.6, pp.772-787, 2011.
[15] J. Choi, and D. Ryu, "Bayesian optimization framework for cross-version defect prediction," in *Proceedings of the 23rd Korea Conference on Software Engineering*, pp.63-72, 2021.
[16] B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," *Empirical Software Engineering*, 2009.
[17] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *Proceedings - International Conference on Software Engineering*, 2013.
[18] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang, "Domain adaptation via transfer component analysis," *IEEE Transaction on Neural Networks*, Vol.22, No.2, pp.199-210, 2011.

최 정 환

https://orcid.org/0000-0002-6530-2662
e-mail : jeongwhan.choi@yonsei.ac.kr
2020년 전북대학교 소프트웨어공학과(학사)
2020년 ~ 현 재 연세대학교 인공지능학과 석·박사통합과정
관심분야 : Graph Convolutional Networks, Neural Ordinary/Controlled Differential Equation, Software Defect Prediction



류 덕 산

https://orcid.org/0000-0002-9556-0873
e-mail : duksan.ryu@jbnu.ac.kr
1999년 한양대학교 전자계산학과(학사)
2012년 한국과학기술원 및 카네기멜론 대학교 소프트웨어공학 복수학위(석사)
2016년 한국과학기술원 전산학부(박사)
2018년 ~ 현 재 전북대학교 소프트웨어공학과 조교수
관심분야 : 인공지능기반 소프트웨어분석, 소프트웨어 결함예측, 소프트웨어 신뢰성, 소프트웨어 메트릭스, 소프트웨어 품질보증, 자율시스템

Appendix

Table 9. Model Performance of AUC on the Selective Transfer Learning

Model			TCA + BRF		NNFilter + BRF		DSNF + BRF	
Project	Target Version	Source Version	Default (AUC)	Selective (AUC)	Default (AUC)	Selective (AUC)	Default (AUC)	Selective (AUC)
ant	1.7	1.3	0.6199	0.6740	0.6534	0.6806	0.6552	0.7055
		1.4	0.5143	0.6244	0.5389	0.6026	0.5332	0.6453
		1.5	0.5972	0.6491	0.6079	0.6552	0.6191	0.6602
		1.6	0.6955	0.7187	0.6706	0.7231	0.6916	0.7307
camel	1.6	1.0	0.5265	0.5532	0.5210	0.5510	0.5311	0.5504
		1.2	0.5929	0.6218	0.5837	0.6637	0.6090	0.6535
		1.4	0.6343	0.6569	0.6343	0.6616	0.6177	0.6503
ivy	2.0	1.1	0.6917	0.7712	0.7250	0.7686	0.7260	0.7609
		1.4	0.5513	0.6215	0.5465	0.6183	0.5263	0.6138
jedit	4.3	3.2	0.5807	0.6612	0.6199	0.6810	0.5890	0.7500
		4.0	0.7236	0.8246	0.6254	0.7236	0.6264	0.7142
		4.1	0.6594	0.7028	0.6945	0.8649	0.6355	0.8163
		4.2	0.6753	0.7672	0.6795	0.7714	0.6795	0.7704
log4j	1.2	1.0	0.6275	0.6852	0.6222	0.6931	0.6301	0.7037
		1.1	0.5570	0.6407	0.5650	0.6280	0.5804	0.6381
lucene	2.4	2.0	0.6321	0.6613	0.6086	0.6537	0.5956	0.6576
		2.2	0.5658	0.6203	0.6059	0.6375	0.5569	0.6401
poi	3.0	1.5	0.5880	0.6823	0.6232	0.7277	0.6703	0.6964
		2.0	0.5508	0.5757	0.5468	0.6215	0.5437	0.6695
		2.5	0.6433	0.6934	0.5757	0.6868	0.5574	0.6940
synapse	1.2	1.0	0.5347	0.6477	0.5228	0.6334	0.5115	0.6861
		1.1	0.5895	0.6534	0.6039	0.6475	0.6009	0.6361
velocity	1.6	1.4	0.4827	0.5498	0.4606	0.5412	0.4930	0.5938
		1.5	0.6485	0.6876	0.6162	0.6909	0.6487	0.6811
xerces	1.4	1.0	0.2869	0.3995	0.3059	0.3786	0.2407	0.5287
		1.2	0.4109	0.4745	0.3937	0.4696	0.3969	0.5320
		1.3	0.5727	0.6355	0.5693	0.6036	0.5918	0.6373

Table 10. Model Performance of AUC on Optimization Framework using TCA, NNFilter, and DSNF

Model			Selective TCA + BRF		Selective NNFilter + BRF		Selective DSNF + BRF	
Project	Target Version	Source Version	Default (AUC)	Optimized (AUC)	Default (AUC)	Optimized (AUC)	Default (AUC)	Optimized (AUC)
ant	1.7	1.3	0.6928	0.7212	0.7018	0.7562	0.6801	0.7440
		1.4	0.5572	0.6258	0.5389	0.6767	0.5300	0.6968
		1.5	0.6347	0.7477	0.6787	0.7592	0.7133	0.7651
		1.6	0.7074	0.7659	0.7363	0.7682	0.7117	0.7736
camel	1.6	1.0	0.5959	0.6043	0.5896	0.6305	0.5634	0.6528
		1.2	0.6086	0.6398	0.6109	0.6690	0.5300	0.6605
		1.4	0.6395	0.6646	0.6200	0.6662	0.6257	0.6672
ivy	2.0	1.1	0.6679	0.7587	0.6212	0.7702	0.7388	0.7638
		1.4	0.6699	0.7910	0.7292	0.7782	0.7202	0.7788
jedit	4.3	3.2	0.6176	0.7651	0.6085	0.6560	0.6039	0.8159
		4.0	0.6519	0.8077	0.7018	0.7562	0.5471	0.7565
		4.1	0.5825	0.6706	0.6446	0.7737	0.6591	0.8254
		4.2	0.6695	0.7070	0.6664	0.7215	0.6830	0.7049
log4j	1.2	1.0	0.6152	0.7143	0.6465	0.6989	-0.1194	0.2289
		1.1	0.5734	0.6391	0.5835	0.6258	-0.0791	0.2289
lucene	2.4	2.0	0.5991	0.6505	0.6234	0.6615	0.5954	0.6581
		2.2	0.5994	0.6436	0.6030	0.6483	0.5144	0.6499
poi	3.0	1.5	0.6348	0.7303	0.7094	0.7432	0.6054	0.7165
		2.0	0.5369	0.6231	0.5861	0.6515	0.4125	0.7030
		2.5	0.6423	0.7107	0.6254	0.7107	0.6275	0.6744
synapse	1.2	1.0	0.6119	0.6934	0.6354	0.7142	0.7101	0.7315
		1.1	0.6120	0.6963	0.6469	0.6818	0.6036	0.6847
velocity	1.6	1.4	0.4823	0.5998	0.5109	0.5853	0.5412	0.6160
		1.5	0.6210	0.6878	0.6371	0.7008	0.6566	0.6814
xerces	1.4	1.0	0.2347	0.3944	0.2755	0.3630	0.4980	0.5135
		1.2	0.3099	0.4262	0.3001	0.4126	0.4042	0.5340
		1.3	0.6528	0.6985	0.5961	0.7013	0.4458	0.5671

Table 11. Effort-aware Model Performance of FIR on Optimization Framework using TCA, NNFilter, and DSNF

Model			Selective TCA + BRF		Selective NNFilter + BRF		Selective DSNF + BRF	
Project	Target Version	Source Version	Default (FIR)	Optimized (FIR)	Default (FIR)	Optimized (FIR)	Default (FIR)	Optimized (FIR)
ant	1.7	1.3	0.6174	0.4972	0.4239	0.5902	0.3009	0.5697
		1.4	0.1597	0.3079	0.3779	0.4000	0.3087	0.4797
		1.5	0.4680	0.5000	0.6156	0.5815	0.5187	0.5888
		1.6	0.5305	0.5489	0.5461	0.5671	0.5969	0.5614
camel	1.6	1.0	0.3657	0.3181	0.3804	0.4115	-0.0339	0.4546
		1.2	0.3115	0.3448	0.3409	0.3848	0.1800	0.3920
		1.4	0.3970	0.4255	0.4137	0.4376	0.3778	0.4254
ivy	2.0	1.1	0.3999	0.5117	0.5221	0.5178	0.4855	0.5055
		1.4	0.4278	0.6023	0.5530	0.6165	0.5523	0.6591
jedit	4.3	3.2	0.2846	0.4251	0.4027	0.4794	0.3963	0.6192
		4.0	0.3442	0.6037	0.3144	0.5273	0.1407	0.5016
		4.1	0.4474	0.5732	0.4602	0.4730	0.4667	0.6362
		4.2	0.5528	0.6039	0.5752	0.6806	0.5592	0.6295
log4j	1.2	1.0	0.0407	0.0676	0.0302	0.0675	-0.1247	0.2289
		1.1	0.0213	0.0470	0.0417	0.0447	-0.0913	0.2289
lucene	2.4	2.0	0.1664	0.2176	0.1842	0.2339	0.1502	0.2091
		2.2	0.1044	0.2021	0.1316	0.2056	-0.0601	0.1858
poi	3.0	1.5	0.1314	0.2338	0.1600	0.2452	0.1375	0.2305
		2.0	0.1193	0.1606	0.1602	0.2096	-0.1847	0.1907
		2.5	0.0189	0.2283	0.1699	0.2271	0.0944	0.2119
synapse	1.2	1.0	0.4104	0.4259	0.3996	0.4796	0.5056	0.4892
		1.1	0.3735	0.4075	0.3816	0.4072	0.2609	0.4138
velocity	1.6	1.4	-0.0695	0.1485	-0.0033	0.1330	0.0616	0.1883
		1.5	0.2583	0.3081	0.2189	0.3038	0.2815	0.2827
xerces	1.4	1.0	-0.3124	-0.0849	-0.3094	-0.1271	-0.1171	0.0103
		1.2	-0.5979	-0.1270	-0.1679	-0.1486	-0.0330	0.0229
		1.3	0.1842	0.2114	0.2165	0.1981	-0.1326	0.0549

Table 12. Effort-aware Model Performance of LIR on Optimization Framework using TCA, NNFilter, and DSNF

Model			Selective TCA + BRF		Selective NNFilter + BRF		Selective DSNF + BRF	
Project	Target Version	Source Version	Default (LIR)	Optimized (LIR)	Default (LIR)	Optimized (LIR)	Default (LIR)	Optimized (LIR)
ant	1.7	1.3	0.0693	0.0140	0.1304	0.1964	0.2020	0.2764
		1.4	-0.2172	-0.3572	-0.5496	-0.1766	0.1416	0.3358
		1.5	-0.0010	0.0196	-0.0361	0.1793	0.4703	0.4751
		1.6	0.0122	0.1155	0.0388	0.1512	0.3435	0.1400
camel	1.6	1.0	-0.7832	-0.6948	-0.5708	-0.4628	-0.2205	0.2259
		1.2	-0.6473	-0.6284	-0.8126	-0.5291	-0.1736	-0.5113
		1.4	-0.6021	-0.4278	-0.5318	-0.3979	-0.4574	-0.4283
ivy	2.0	1.1	-0.7188	-0.6786	-0.8677	-0.6575	-0.9792	-0.6998
		1.4	-0.8620	-0.3672	-0.7299	-0.3184	-0.7780	-0.1719
jedit	4.3	3.2	-0.4765	-0.3842	-0.3227	-0.2535	-0.3017	0.4372
		4.0	-0.3534	-0.0458	-0.5072	-0.1381	0.1722	0.2317
		4.1	-0.3611	-0.0277	-0.3534	-0.2688	0.1573	0.1240
		4.2	-0.0612	0.0465	0.1157	0.2310	0.1080	0.1080
log4j	1.2	1.0	-1.8554	-1.8518	-1.8853	-1.8522	-0.1487	0.2289
		1.1	-1.9479	-1.9148	-2.0053	-1.9217	-0.1447	0.2289
lucene	2.4	2.0	-1.5561	-1.3925	-1.5561	-1.3429	-0.8210	-0.6147
		2.2	-1.6298	-1.4400	-1.6983	-1.4293	-1.2853	-1.4898
poi	3.0	1.5	-1.4421	-1.1378	-1.1575	-1.1060	-0.5811	-0.7878
		2.0	-1.4034	-1.3421	-1.2667	-1.2054	-1.0447	-0.8802
		2.5	-1.2565	-1.1534	-1.2925	-1.1567	-0.7087	-0.5124
synapse	1.2	1.0	-0.8591	-0.7992	-0.7466	-0.6307	-0.1968	-0.1130
		1.1	-1.3214	-0.8567	-1.0470	-0.8578	-1.2012	-0.8371
velocity	1.6	1.4	-1.5230	-1.1955	-1.3462	-1.2354	-0.6100	-0.3800
		1.5	-0.9684	-0.7838	-0.9229	-0.7951	-0.9263	-0.8495
xerces	1.4	1.0	-2.9912	-2.1603	-3.0433	-2.2832	-0.5780	-0.4808
		1.2	-2.9433	-2.2829	-3.3775	-2.3458	-0.7762	-0.4618
		1.3	-1.3579	-1.2971	-1.2980	-1.3359	-0.5461	-0.9666