ISSN 2383-630X(Print) / ISSN 2383-6296(Online)
Journal of KIISE, Vol. 49, No. 1, pp. 52-59, 2022. 1
https://doi.org/10.5626/JOK.2022.49.1.52

소프트웨어 결함 예측에 적합한 생성적 적대 신경망 모델 식별 연구

(Identification of Generative Adversarial Network Models Suitable for Software Defect Prediction)

최 지 원 † (Jiwon Choi)　이 재 욱 † (Jaewook Lee)　류 덕 산 †† (Duksan Ryu)　김 순 태 ††† (Suntae Kim)

요 약 소프트웨어 결함 예측은 결함이 야기될 모듈을 식별해 한정된 품질 보증 자원을 효과적으로 배분하는데 도움을 준다. 소프트웨어 결함 데이터는 비결함 인스턴스의 수가 결함 인스턴스의 수보다 많은 클래스 불균형 문제를 겪는다. 대부분의 기계 학습에서 특정 클래스의 인스턴스 비율이 한쪽으로 치우치게 되면 결함 예측 성능에 부정적인 영향을 끼친다. 따라서 본 연구에서는 생성적 적대 신경망 모델(Generative Adversarial Network, GAN)을 사용해 클래스 불균형 문제를 해결하고, 결함 예측 성능 향상을 목표로 한다. 이를 위해, 본 연구에서는 여러 종류의 GAN 모델 중 소프트웨어 결함 예측에 적합한 모델은 무엇인지 비교하고, 관련 연구에서 적용하지 않았던 GAN 모델들의 적용성 여부를 확인한다. 본 연구에서는 이미지 생성에 최적화되어 있는 Vanilla-GAN(GAN)과 Conditional GAN(cGAN), Wasserstein GAN(WGAN) 모델을 소프트웨어 결함 예측 데이터에 적합하게 개조한 후, 개조한 GAN과 cGAN, WGAN, Tabular GAN(TGAN), Modeling Tabular data using Conditional GAN(CTGAN)의 성능을 비교 실험한다. 실험 결과, CTGAN 모델이 소프트웨어 결함 예측 데이터에 적합함을 보인다. 또한 CTGAN의 하이퍼파라미터 중 결함 발견율(Recall)을 높이고, 결함 오보율(Probability of False Alarm, PF)를 낮추는 하이퍼파라미터 값은 무엇인지 민감도 분석을 수행한다. 실험 결과, 데이터셋에 따라 하이퍼파라미터를 조정해야 함을 보였다. 우리의 제안한 기법이 소프트웨어 결함 예측의 성능을 향상시켜 한정된 자원을 효과적으로 할당하는데 도움이 될 것이라고 기대한다.

키워드: 소프트웨어 결함 예측, 생성모델, 클래스 불균형 문제, 하이퍼파라미터 최적화

Abstract Software Defect Prediction(SDP) helps effectively allocate quality assurance resources which are limited by identifying modules that are likely to cause defects. Software defect data suffer from class imbalance problems in which there are more non-defective instances than defective instances. In most machine learning methods, the defect prediction performance is degraded when there is a disproportionate number of instances belonging to a particular class. Therefore, this research

· 본 연구는 원자력안전위원회의 재원으로 한국원자력안전재단의 지원을 받아 수행한 원자력안전연구사업(No. 2105030)과 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(NRF-2019R1G1A1005047)을 받아 수행된 연구사업의 결과이다.
· 이 논문은 2020 한국소프트웨어종합학술대회에서 '소프트웨어 결함 예측을 위한 생성적 적대 신경망 모델 비교 연구'의 제목으로 발표된 논문을 확장한 것임

논문접수 : 2021년 4월 12일
(Received 12 April 2021)
논문수정 : 2021년 11월 16일
(Revised 16 November 2021)
심사완료 : 2021년 11월 17일
(Accepted 17 November 2021)

† 학생회원 : 전북대학교 소프트웨어공학과 학생
wanny_94@naver.com
aewexpress01@jbnu.ac.kr
†† 종신회원 : 전북대학교 소프트웨어공학과 교수(Jeonbuk Nat'l Univ.)
duksan.ryu@jbnu.ac.kr
(Corresponding author임)
††† 정 회 원 : 전북대학교 소프트웨어공학과 교수
stkim@jbnu.ac.kr

정보과학회논문지 제49권 제1호(2022. 1)

aimed to solve the class imbalance problem and improve defect prediction performance by using a Generative Adversarial Network(GAN) model. To this end, we compared different kinds of GAN models for their suitability for SDP and checked the applicability of GAN models that were not applied in the related work. In our study, Vanilla-GAN(GAN), Conditional GAN (cGAN), and Wasserstein GAN (WGAN) models which were initially proposed for image generation were adapted for software defect prediction. Then those modified models were compared with Tabular GAN(TGAN) and Modeling Tabular data using Conditional GAN(CTGAN). Our experimental results showed that the CTGAN model is suitable for SDP data. We also conducted a sensitivity analysis examining which hyper-parameter values of CTGAN increase the recall rate and lower the probability of false alarm (PF). Our experimental results indicated that the hyper-parameters should be adjusted according to the dataset. We expect that our proposed approach can help effectively allocate limited resources by improving the performance of SDP.

Keywords: software defect prediction, Generative Adversarial Network, class imbalance problem, hyper-parameter optimization

1. 서 론

소프트웨어 결함 예측(Software Defect Prediction)은 테스팅 단계 전에 결함이 야기될 소프트웨어 모듈을 보다 많이 식별해 한정된 자원(인적 자원, 물적 자원 등)을 효율적으로 배분하는데 기여한다[1,2]. 소프트웨어 결함 예측은 소프트웨어 결함 추정(Estimation)과 다르며, 소프트웨어 결함 추정은 주로 테스팅 단계에서 수집한 소프트웨어 고장(Failure) 데이터에 의거해 향후 소프트웨어 고장 수를 추정하는 방법이다[3].

소프트웨어 결함 데이터는 클래스 불균형 문제를 가지고 있으며, 이는 비결함 인스턴스의 수가 결함 인스턴스 수보다 많은 것을 의미한다. 데이터의 클래스가 불균형한 상태에서 분류 모델을 학습할 경우, 분류 성능이 저하돼 이를 해결하는 것은 매우 중요하다[4]. 클래스 불균형 문제를 해결하기 위해 많은 연구가 진행되어 왔다[5,6]. 특히 데이터 수준에서 클래스 불균형 문제를 해결하는 방법은 관련 연구에서 보편적으로 사용하는 방법이다[6]. 대표적으로 소수 클래스의 인스턴스를 생성하는 오버샘플링(Oversampling)과 다수 클래스의 인스턴스를 제거하는 언더샘플링(Undersampling)방법이 포함된다. 소프트웨어 결함 예측에서는 인스턴스에 결함과 관련된 중요한 정보가 포함되어 있을 수 있기 때문에 언더샘플링 기법보다 오버샘플링 기법이 연구에서 보다 많이 적용된다[6].

최근 딥러닝에서는 생성적 적대 신경망(Generative Adversarial Networks, GAN)[7] 모델을 사용해 이미지와 같은 비정형 데이터를 증강하는 연구[8]가 활발히 진행되고 있다. Choi et al.[9] 연구는 소프트웨어 결함 예측 데이터에서 소수 클래스인 결함 데이터를 GAN과 cGAN[10], CTGAN[11] 모델을 통해 생성하고 이들 간의 성능을 비교 실험하였다. 실험 결과 정형 데이터를 모델링하는 CTGAN 모델이 소프트웨어 결함 예측 데이터 생성에 적합함을 보였다.

본 연구에서는 기존 연구[9]를 확장시켜 두 개의 GAN 모델(WGAN[12]와 TGAN[13])을 추가하고, 7개의 오픈 소스 프로젝트를 추가 실험한다. 본 연구에서는 기존 연구[9]에서 사용한 GAN과 cGAN, CTGAN 모델을 동일하게 사용한다. 본 연구의 목표는 다음과 같다.

- 기존 연구[9]에서 다루지 않았던 GAN 모델들(WGAN, TGAN)의 적용성 여부를 확인한다.
- 새로운 데이터셋을 추가시켜 기존 연구[9]에서 사용한 모델들의 성능을 검증한다. 실험 결과를 통계적으로 검증하기 위해 Cohen's d[14]를 사용한다. Cohen's d는 비교 방법 간의 성능 차이를 수치적으로 보인다.
- 본 연구에서 사용한 5개의 GAN 모델 중 소프트웨어 결함 예측 데이터 생성에 적합한 모델은 무엇인지 실험을 통해 확인한다.

본 연구의 실험 결과, WGAN과 TGAN이 소프트웨어 결함 예측에 적용 가능함을 보였다. 하지만, CTGAN 모델이 다른 모델 대비 우수한 Balance 성능을 보여 소프트웨어 결함 예측에 가장 적합함을 보였다. 또한 사용하는 데이터셋에 따라 CTGAN 모델의 하이퍼파라미터 조정이 필요함을 보였다.

2. 관련 연구

클래스 불균형 문제를 해결하는 방법은 크게 데이터 수준과 알고리즘 수준, 앙상블 기법으로 나눠진다[2].

데이터 수준에서 클래스 불균형 문제를 해결하는 방법은 모델을 구축하기 이전에 결함과 비결함의 데이터 분포를 조정한다. 특히 해당 방법은 소프트웨어 결함 예측 연구에서 널리 사용되는 방법이다[6]. 알고리즘 수준에서 클래스 불균형 문제를 해결하는 방법은 실험에 사용할 모델 내부에서 결함을 잘 식별할 수 있도록 내부 훈련 메커니즘을 수정하는 방법이다[15]. 앙상블 기법은 개별 학습 모델을 결합시켜 전반적인 성능을 향상시킨다[5].

소프트웨어 결함 예측 연구에서도 생성적 적대 신경망을 적용하려는 접근법이 제안되어왔다. 교차 프로젝트 결함 예측(CPDP)과 이기종 결함 예측(HDP)에서는 각 프로젝트의 데이터 분포 차이를 줄이기 위해 GAN 모델을 사용했다. Go-ng et al.[16]는 전이 학습(Transfer learning) 방법 대신에 GAN 모델을 사용해 서로 다른 프로젝트 간의 공통 메트릭을 만들었다. Lee et al.[17] 연구는 프로젝트 내 결함 예측(WPDP)에 GAN 모델을 적용시켰고, 예측 성능의 개선이 필요함을 보였다. 본 연구에서는 데이터 수준에서 클래스 불균형 문제를 해결한다.

3. 연구 방법

그림 1은 본 연구의 전체적인 접근 방법을 나타낸다. 프로젝트 내 결함을 예측하기 위해 10-fold 교차 검증을 진행한다. 실험에 가용한 데이터셋 중 Audi 데이터셋은 소프트웨어 출시된 시간 정보(release historical data)를 포함하고 있다. 따라서 해당 데이터셋의 세 가지 프로젝트(Project A, K, L)은 교차 검증을 진행하지 않고, 프로젝트 A와 L은 50:50, 프로젝트 K는 40:60의 비율로 학습 데이터와 테스트 데이터를 나눈다. 이처럼 나누는 기준은 관련 연구[18]를 참고해 동일하게 설정하였다. 이후, 최소-최대화 정규화를 적용한다.

본 연구에서는 다섯 가지의 GAN 모델들(Vanilla GAN, cGAN, WGAN, TGAN, CTGAN)을 사용해 성능을 비교한다. 라벨 정보를 사용하는 cGAN과 CTGAN 모델을 제외한 세 가지 GAN 모델에 소수 클래스 인스턴스를 학습시키기 위해 학습 데이터에서 결함 인스턴스와 비결함 인스턴스를 나눈다. 소프트웨어 결함 예측에서 소수 클래스는 결함 클래스를 의미한다. 이후 GAN 모델들(Vanilla GAN, WGAN, TGAN)을 사용해 결함 인스턴스를 비결함 인스턴스와 1:1 비율이 되도록 데이터를 생성한다. cGAN과 CTGAN 모델의 경우, 결함과 비결함 클래스를 나누지 않은 학습 데이터를 사용해 GAN 모델을 학습시키고, 결함 인스턴스만 생성한다. 생성된 데이터와 기존의 학습 데이터를 합쳐 새로운 학습 데이터를 만든다. 이후 새로운 학습 데이터로 Logistic Regression 예측 모델을 학습하고, 테스트 데이터로 모델의 성능을 확인한다.

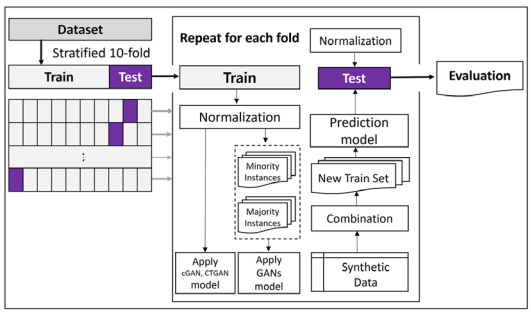

그림 1 전체 프레임워크
Fig. 1 Overall Framework

4. 실험 설정

4.1 연구 질문

본 연구에서는 소프트웨어 결함 예측에 적합한 GAN 모델을 확인하기 위해, 두 가지의 연구 질문을 설정한다.

RQ1. 소프트웨어 결함 예측 데이터에 가장 좋은 성능을 보이는 GAN 모델은 무엇인가?

H_0: GAN 모델들의 성능이 유사하다.

H_A: GAN 모델들의 성능 차이가 있다.

RQ2. CTGAN 모델 성능에 영향을 미치는 하이퍼파라미터는 무엇인가?

4.2 실험 환경

본 연구에서 실험 환경은 3.40GHz CPU, 8GB RAM의 Window 10에서 python 3.7을 이용하였다. 실험에 사용한 데이터는 Altinger et al.[19]이 제시한 자동차 분야 데이터셋과 오픈 소스 데이터셋인 AEEEM와 JIRA을 사용한다. 자동차 분야 데이터셋은 모델기반 개발 기법으로 생성된 코드에서 정적 메트릭을 추출한다. 데이터에는 소스코드 정적 소프트웨어 메트릭 정보(LOC, McCabe, Halstead)들이 포함되어 있다. 각 데이터셋에 대한 전체 크기 및 결함 수는 표 1과 같다.

표 1 실험에 가용한 15개의 프로젝트 정보
Table 1 An Overview of 15 studied Projects

Dataset	Project	#of Metric	# of Instances	
			All	Defective
AEEEM	EQ	61	324	129
	JDT	61	997	206
	LC	61	691	64
	ML	61	1,862	245
	PDE	61	1,497	209
Audi	Project A	12	999	80
	Project K	12	2,452	362
	Project L	12	1,671	75
JIRA	activemq-5.0.0	65	1,884	293
	derby-10.5.1.1	65	2,705	383
	groovy-1_6_BETA_1	65	821	70
	hbase-0.94.0	65	1,059	218
	hive-0.9.0	65	1,416	283
	jruby-1.1	65	731	87
	wicket-1.3.0-beta2	65	1,763	130

4.3 평가 척도

이진 분류(Binary classification)에 대한 예측 성능은 일반적으로 표 2의 혼동 행렬(Confusion matrix)을 사용해 평가된다. 특히 소프트웨어 결함 예측에서는 결함 발견율(Recall)을 높이고, 결함 오보율(Probability of False Alarm, PF)은 낮추는 것을 목표로 한다. 본 연구에서는 총 네 가지의 평가 척도를 사용해 성능을 확인한다. 또한, Cohen's d 효과 크기 시험[14]을 사용해 모델들 간의 성능 차이를 통계적으로 검증한다.

표 2 혼동 행렬
Fig. 2 Confusion Matrix

		Predicted Class	
		Defective	Clean
Actual Class	Defective	TP(True Positive)	FN(False Negative)
	Clean	FP(False Positive)	TN(True Negative)

4.3.1 성능 평가 지표

• 결함 발견율(Recall)

Recall은 실제 결함 중에서 모델이 결함이라고 예측한 지표이다. 공식은 식 (1)과 같다.

$$Recall = \frac{TP}{TP+FN} \tag{1}$$

• 결함 오보율(Probability of False Alarm, PF)

결함 오보율은 결함으로 잘못 분류된 비결함의 수와 전체 비결함 수의 비율을 나타낸다. 공식은 식 (2)와 같다.

$$pf = \frac{FP}{FP+TN} \tag{2}$$

• Balance

Balance는 Recall과 PF를 모두 고려하는 지표로 소프트웨어 결함 예측에 많이 사용되는 평가 지표 중 하나이다[20]. 공식은 식 (3)과 같다.

$$balance = 1-\sqrt{\frac{(0-pf)^2+(1-Recall)^2}{2}} \tag{3}$$

• 코드 검사 노력도(File Inspection Reduction, FIR)

FIR은 실무자의 입장에서 코드 검사 작업에 대한 노력을 어느 정도 감소시킬 수 있는지 평가하는 지표이다[21]. FIR은 동일한 PD(Probability of Detection, Recall)를 달성하기 위해 검사할 파일 수가 감소한 비율이다. FIR 성능이 높으면 파일 크기가 작을수록 결함을 잘 찾을 수 있음을 의미한다. FI(File Inspection)은 전체 파일에 대한 검사할 파일 수의 비율이고(5), FIR은 식 (4)와 같이 정의된다.

$$FIR = \frac{PD-FI}{PD} \tag{4}$$

$$FI = (TP+FP)/(TP+TN+FP+FN) \tag{5}$$

4.3.2 통계적 유의성 평가 지표

본 연구에서는 실험 결과가 통계적으로 유의하다는 것을 보이기 위해 Cohen's d를 사용한다. Cohen's d는 제안한 방법과 다른 비교 방법들의 성능 차이를 수치적으로 확인할 수 있는 효과 크기(Effect-size) 측정 방법이다[14]. 공식은 식 (6)과 같다.

$$cohen'sd = \frac{M_1-M_2}{\sqrt{\frac{\delta_1^2+\delta_2^2}{2}}} \tag{6}$$

Cohen's d의 값에 따라 네 가지 레벨로 나뉘고, 표 3에 정리되어 있다. 이 레벨은 성능 차이의 정도를 나타내며, Medium-level의 이상의 효과 크기는 제안한 방법이 다른 방법 대비 성능이 우수하다는 것을 의미한다.

표 3 Cohen's d를 기반으로 한 효과 수준
Fig. 3 Effectiveness Levels Based on Cohen's d

Cohen's d	level
$0 \le \lvert d \rvert < 0.2$	Negligible(N)
$0.2 \le \lvert d \rvert < 0.5$	Small(S)
$0.5 \le \lvert d \rvert < 0.8$	Medium(M)
$\lvert d \rvert \ge 0.8$	Large(L)

5. 실험 결과

5.1 RQ1: 소프트웨어 결함 예측 데이터에 가장 좋은 성능을 보이는 GAN 모델은 무엇인가?

본 연구 질문에서는 5가지 GAN 모델의 성능을 비교한다. 표 4는 비교 실험 결과이며, 표의 굵은 글씨는 각 평가 지표에서 가장 성능이 우수함을 나타낸다.

먼저, 클래스 불균형 문제를 가지고 있는 경우(Default)와 이를 해결했을 때의 성능을 비교한다. 표 4에서 확인할 수 있듯이, 클래스 불균형 문제(Default)를 가지고 있는 경우, PF 성능은 우수하지만, Recall과 Balance는 낮은 성능을 보인다. 또한, FIR 성능은 데이터 불균형 문제를 해결하지 않았을 때가 성능이 더 우수함을 보인다. 하지만, FIR 성능은 우수하지만, Recall과 Balance 성능이 낮다면 적은 리소스로 보다 많은 소프트웨어 결함을 찾으려는 소프트웨어 결함 예측의 목표에 부적합하다. 따라서 GAN 모델을 사용해 데이터 불균형 문제를 해결하고, Recall과 Balance 성능 향상을 이뤄야 할 필요성이 보인다.

소프트웨어 결함 예측 데이터에 가장 적합한 GAN 모델을 확인하기 위해, 표 4에서 각 GAN 모델들의 성능을 비교한다. 그 결과, CTGAN 모델이 15개의 프로젝트 중 11개의 프로젝트에서 Recall과 Balance 성능이 우수함을 보인다. 특히 오픈 소스 데이터셋(AEEEM, JIRA)에서 Recall과 Balance 성능이 우수하다. TGAN 모델은 15개

표 4 GAN 모델들과의 성능 비교
Table 4 Performance Comparison with GAN Models

		AEEEM					Audi			JIRA						
		EQ	JDT	LC	ML	PDE	A	K	L	act	derby	groovy	hbase	hive	jruby	wi
Default	Recall	0.8448	0.6057	0.5452	0.2536	0.3788	0.5106	0.5674	0	0.7165	0.5618	0.4714	0.5681	0.5838	0.6527	0.323
	PF	0.3221	**0.1036**	0.1017	**0.0408**	**0.0815**	**0.0505**	**0.027**	0	**0.1162**	**0.1085**	**0.0493**	0.1071	**0.1243**	**0.059**	**0.0391**
	Balance	0.7372	0.7067	0.6676	0.4711	0.5556	**0.6521**	0.6935	0.2929	0.7776	0.6793	0.6234	0.6754	0.6876	0.7481	0.5202
	FIR	0.3652	**0.6558**	**0.7302**	**0.7173**	**0.6657**	**0.8466**	**0.8451**	nan	**0.7083**	**0.6853**	**0.8243**	0.6387	**0.6337**	**0.7917**	**0.7861**
GAN	Recall	0.8224	0.6247	0.5619	0.376	0.5016	**1**	0.7943	0.814	0.7066	0.6641	0.4714	0.5223	0.6293	0.693	0.4461
	PF	0.3439	0.1339	0.1657	0.095	0.1707	0.8525	0.1278	0.1959	0.1175	0.2029	0.0959	0.101	0.1926	0.099	0.0686
	Balance	0.712	0.7085	0.6581	0.5517	0.6168	0.3972	**0.8288**	0.809	0.7684	0.7022	0.6129	0.6491	0.6829	0.7686	0.5986
	FIR	0.3502	0.6258	0.6339	0.6157	0.5792	0.1386	0.7445	0.7107	0.7057	0.6003	0.7107	0.6242	0.5616	0.7518	0.787
cGAN	Recall	0.8461	0.5488	0.4047	0.338	0.3357	0.1064	**0.8369**	0.4186	0.612	0.402	0.3	0.5054	0.5475	0.4125	0.2362
	PF	0.3684	0.1922	0.1915	0.1244	0.111	0.1571	0.6679	0.0334	0.1854	0.115	0.1196	0.1319	0.1924	0.1036	0.0776
	Balance	0.7062	0.647	0.5529	0.5183	0.5199	0.3584	0.5138	0.5882	0.6559	0.5663	0.4712	0.6289	0.6045	0.5603	0.4317
	FIR	0.3378	0.5071	0.4362	0.5527	0.5472	-0.447	0.1792	0.8611	0.606	0.6055	0.5471	0.5876	0.5605	0.583	0.5911
WGAN	Recall	**0.891**	0.5097	0.5952	0.3428	0.4445	0.5106	0.766	0.907	0.7518	0.5379	0.6857	0.6162	0.6848	0.7291	0.523
	PF	0.3352	0.1404	0.2215	0.1094	0.2307	0.0669	0.1134	0.1545	0.1583	0.1184	0.169	0.1959	0.1862	0.1615	0.142
	Balance	**0.7443**	0.6317	0.6553	0.527	0.5668	0.6507	0.8161	0.8725	0.7844	0.6601	0.7175	0.6642	**0.7327**	0.7514	0.639
	FIR	**0.3761**	0.5783	0.5689	0.593	0.4046	0.8164	0.756	0.7765	0.667	0.6722	0.6834	0.5576	0.5739	0.6847	0.6566
TGAN	Recall	0.7814	0.588	0.5619	0.2734	0.4692	0.5106	0.5887	0.2093	0.7303	0.5876	0.4714	0.5383	0.6284	0.7	0.3461
	PF	**0.3179**	0.1075	**0.0973**	0.0557	0.128	0.265	0.0675	0.0255	0.125	0.1257	0.0905	**0.0831**	0.1862	0.0867	0.052
	Balance	0.7074	0.6949	0.6653	0.4838	0.6102	0.6065	0.7053	0.4406	0.785	0.6919	0.6164	0.6664	0.6902	0.7752	0.536
	FIR	0.3557	0.6441	0.6837	0.6955	0.6379	0.4519	0.7856	0.8219	0.7005	0.6733	0.7384	**0.6694**	0.5708	0.7693	0.7361
CTGAN	Recall	0.8609	**0.6704**	**0.719**	**0.4985**	**0.6509**	0.5106	0.7801	0.8605	**0.8152**	**0.7205**	**0.7285**	**0.7441**	**0.7177**	**0.7611**	**0.7615**
	PF	0.3755	0.1769	0.2362	0.1607	0.2584	0.056	0.1341	0.1067	0.1935	0.2359	0.2421	0.2103	0.2858	0.1474	0.2621
	Balance	0.7081	**0.7275**	**0.7243**	**0.6245**	**0.6855**	0.6517	0.8179	0.8758	**0.7993**	**0.7314**	**0.7268**	**0.7482**	0.6965	**0.791**	**0.7206**
	FIR	0.3403	0.5836	0.6068	0.588	0.5174	0.8365	0.7348	0.8198	0.6457	0.5779	0.6153	0.5729	0.4934	0.706	0.6169

의 프로젝트 중 12개의 프로젝트에서 PF 성능이 우수함을 보인다. 또한 9개의 프로젝트에서 우수한 FIR 성능을 보인다. 특히 오픈 소스 데이터셋(AEEEM, JIRA)에서 PF의 성능이 우수함을 확인할 수 있다.

CTGAN과 TGAN은 테이블 형태의 데이터 처리를 위해 제안된 모델로, 테이블 데이터의 특성인 이산형 변수와 연속형 변수를 고려한다. 특히 CTGAN은 TGAN 모델이 가지고 있는 한계점을 개선시킨 모델로, 연속형 변수 처리과정에서 TGAN과 차이가 있다[11]. 소프트웨어 결함 예측 데이터는 이산형 변수와 연속형 변수를 모두 가지고 있다. 따라서 실험 결과를 통해 CTGAN 모델의 특성이 소프트웨어 결함 예측 데이터에서 잘 동작함을 확인할 수 있다.

본 연구의 목표는 Recall 성능을 높이고, PF를 낮추는 GAN 모델을 찾아 소프트웨어 결함 예측력을 높이는 것이다. 따라서 Recall과 PF를 모두 고려하는 Balance 지표의 성능이 우수한 모델이 소프트웨어 결함 예측력을 향상시킬 수 있다고 가정한다.

Balance 성능이 가장 우수한 CTGAN 모델을 기준으로 네 가지 GAN 모델들과 데이터 불균형일 때(Defalut)의 성능을 Cohen's d를 사용해 통계적으로 비교한다. 표 5는 그 결과를 나타낸다. CTGAN을 적용했을 때, Balance

표 5 Coehn's d로 성능을 비교한 결과
Fig. 5 Performance comparison with Cohen's d

	CTGAN vs.				
	Default	GAN	cGAN	WGAN	TGAN
Recall	1.3688(L)	0.5736(M)	1.688(L)	0.662(M)	1.426(L)
PF	1.5868(L)	0.055(N)	0.1751(N)	0.5548(M)	1.099(L)
Balance	1.06(L)	0.7391(M)	2.3668(L)	0.5332(M)	1.124(L)
FIR	-0.2481(N)	0.053(N)	0.6734(M)	-0.061(N)	-0.3765(S)

지표에서 다른 모델 대비 Medium-level 이상의 성능 차이를 보이며, 이는 통계적으로 CTGAN과 다른 모델과의 성능 차이가 큼을 의미한다. 이를 통해 소프트웨어 결함 예측에 CTGAN 모델을 사용하는 것이 Balance 성능을 향상시킬 수 있다는 것을 검증하였다. 따라서 대립 가설을 채택할 수 있다.

5.2 RQ2: CTGAN 모델 성능에 영향을 미치는 하이퍼파라미터는 무엇인가?

본 연구 질문에서는 모델 성능에 영향을 미치는 하이퍼파라미터 값을 확인하기 위해 민감도 분석을 진행한다. Xu et al.[11]의 연구에서는 생성되는 데이터의 품질과 모델의 성능에 영향을 미치는 하이퍼파라미터를 제시한다. 따라서, 본 연구에서는 Xu et al.[11] 연구에서 제시한 하이퍼파라미터 중 두 가지의 하이퍼파라미

터를 선택하여 실험을 진행한다.

Xu et al.[11] 연구에 따르면 생성자(Generator)와 판별기(Discriminator)의 출력 샘플의 크기에 따라 각 네트워크의 층(layer)이 생성된다. 본 연구에서는 실험에 사용한 데이터셋의 피처(feature)와 동일하게 출력 샘플의 크기를 지정하고, 각 네트워크의 층 수를 다르게 설정해 해당 하이퍼파라미터가 결함 예측 성능에 미치는 영향을 확인한다.

표 6의 굵은 글씨로 표현한 수치는 각 평가 지표에서 가장 성능이 우수한 경우를 나타내며, '2-layer'는 층(layer) 수를 2개로 설정했다는 의미이다. 표 6에서 확인할 수 있듯이 층을 2개로 설정했을 때 15개의 프로젝트 중 8개의 프로젝트에서 Balance 성능이 다소 향상됨을 확인할 수 있다. 층 수가 많아질수록 연산량이 많아지며 학습 능력이 좋아진다고 하지만, 적절한 결과를 도출해내려면 학습 데이터의 양이 많아야 한다고 알려져 있다. 실험에 사용한 소프트웨어 결함 예측 데이터의 학습 데이터의 수가 평균적으로 1,000개 미만이라는 점에서 층 수가 많아지게 되면 오히려 성능이 다소 저하됨을 확인할 수 있다.

모델의 훈련에 사용되는 배치 사이즈(batch-size)의 크기가 성능에 미치는 영향을 확인하기 위해 민감도 분석을 진행한다. Xu et al.[11] 연구에서는 배치 사이즈를 10의 배수로 설정할 것을 권고한다. 실험 환경은 각 네트워크의 층 수를 2개로 설정하고, Audi 데이터셋은 네트워크의 층 수를 4개로 설정하고 실험을 진행한다. 표 6에서 확인할 수 있듯이, 배치 사이즈를 50으로 설정했을 때 15개의 프로젝트 중에서 6개의 프로젝트에서 Balance의 성능이 다소 향상됨을 확인할 수 있다. 반면, Audi 데이터셋의 Project A는 층 수나 배치 사이즈의 하이퍼파라미터 값의 설정에 영향을 받지 않는 것을 확인할 수 있다.

실험 결과를 통해, 데이터셋에 따라 모델의 하이퍼파라미터 조정이 필요함을 확인할 수 있다.

6. 위협 요소

외부 유효성에 대한 위협은 15개의 프로젝트만을 사용한 것이다. 현재 자동차 도메인에서 가용한 결함 데이터셋은 본 연구에서 사용한 3개가 전부이며, 추가적인 데이터셋이 확보되면 추가적인 실험을 진행할 예정이다. 내부 유효성에 대한 위협은 관련 연구[17]와 성능 비교가 이루어지지 않았다는 점이다. 관련 연구보다 구현한 모델의 성능이 더 우수하여 이를 비교 지표로 삼았다.

표 6 CTGAN 모델의 성능에 영향을 미치는 하이퍼파라미터

Table 6 Hyper-parameters affecting the Performance of CTGAN model

		AEEEM					Audi			JIRA						
		EQ	JDT	LC	ML	PDE	A	K	L	act	derby	groovy	hbase	hive	jruby	wi
2-layer	Recall	**0.8609**	0.6704	**0.719**	**0.4985**	**0.6509**	0.5106	**0.7801**	0.8605	0.8152	0.7205	0.7285	**0.7441**	**0.7177**	**0.7611**	**0.7615**
	PF	0.3755	0.1769	0.2362	0.1607	0.2584	0.056	0.1341	0.1067	0.1935	**0.2359**	0.2421	**0.2103**	**0.2858**	0.1474	0.2621
	Balance	**0.7081**	0.7275	**0.7243**	**0.6245**	**0.6855**	0.6517	**0.8179**	0.8758	0.7993	0.7314	0.7268	**0.7482**	0.6965	**0.791**	**0.7206**
	FIR	0.3403	0.5836	0.6068	0.588	0.5174	0.8365	**0.7348**	0.8198	0.6457	**0.5779**	0.6153	**0.5729**	0.4934	0.706	0.6169
3-layer	Recall	0.8058	0.675	0.6023	0.3886	0.5604	0.5106	0.7092	1	0.809	0.723	**0.7428**	0.7261	0.7594	0.6972	0.5769
	PF	0.3502	0.153	0.1834	**0.1106**	0.2174	0.056	**0.1278**	0.1146	**0.1835**	0.239	0.1638	0.2402	0.291	0.0914	0.1433
	Balance	0.6889	0.7365	0.6765	0.5596	0.6463	0.6517	0.7754	0.919	0.8049	0.7336	**0.7672**	0.7125	**0.7193**	0.7688	0.673
	FIR	0.3462	0.6181	0.6075	**0.6122**	0.5212	0.8365	0.7274	0.8286	**0.6533**	0.5739	**0.7203**	0.5467	**0.4979**	0.758	0.6892
4-layer	Recall	0.807	0.6814	0.6285	0.4366	0.5347	0.5106	0.7447	**1**	**0.819**	**0.7338**	0.5857	0.701	0.7557	0.7041	0.5923
	PF	**0.3426**	0.1554	**0.1129**	0.1323	**0.1591**	0.056	0.1359	**0.1051**	0.1941	0.2459	**0.1292**	0.2224	0.2939	**0.0806**	**0.1261**
	Balance	0.7068	0.7377	0.7136	0.5896	0.6475	0.6517	0.7955	**0.9257**	**0.8089**	**0.7358**	0.6793	0.7109	0.7186	0.7793	0.6852
	FIR	**0.3486**	0.6167	**0.7318**	0.5982	**0.5902**	0.8365	0.7254	**0.8375**	0.6436	0.5734	0.7065	0.5476	0.4879	**0.7804**	**0.7301**
batch-50	Recall	**0.8609**	0.6704	0.719	0.4985	**0.6509**	0.5106	**0.7801**	0.8605	**0.8152**	0.7205	0.7285	0.7441	0.7177	0.7611	0.7615
	PF	0.3755	0.1769	**0.2362**	0.1607	**0.2584**	0.056	**0.1341**	0.1067	0.1935	**0.2359**	**0.2421**	**0.2103**	**0.2858**	0.1474	0.2621
	Balance	0.7081	0.7275	**0.7243**	0.6245	**0.6855**	0.6517	**0.8179**	0.8758	**0.7993**	0.7314	0.7268	**0.7482**	0.6965	**0.791**	0.7206
	FIR	0.3403	0.5836	**0.6068**	0.588	**0.5174**	0.8365	**0.7348**	0.8198	0.6457	**0.5779**	0.6153	**0.5729**	0.4934	0.706	0.6169
batch-100	Recall	0.8205	**0.6988**	**0.7571**	0.4851	0.6119	0.5106	0.7305	**1**	0.798	0.7283	**0.8142**	0.7099	0.7464	**0.768**	**0.7923**
	PF	0.3381	0.1847	0.258	**0.1391**	0.2663	0.056	0.144	0.1736	**0.1847**	0.242	0.2423	0.2305	0.2903	0.168	0.2769
	Balance	**0.7152**	0.7402	0.709	0.6211	0.6593	0.6517	0.7839	0.8772	0.7955	**0.736**	**0.757**	0.7042	**0.7039**	0.7744	0.7289
	FIR	**0.3499**	0.5822	0.566	**0.6138**	0.4884	0.8365	0.7124	0.7734	**0.6508**	0.5744	**0.6472**	0.5412	**0.4965**	0.6879	0.6095
batch-150	Recall	0.8141	0.698	0.6928	**0.501**	0.6457	0.5106	0.7092	**1**	0.7962	**0.7415**	0.7714	**0.7662**	**0.7485**	0.7555	0.7461
	PF	**0.3368**	**0.1707**	0.2405	0.1534	0.2708	0.056	0.1566	**0.1035**	0.208	0.2614	0.2729	0.2459	0.3133	**0.1289**	**0.2418**
	Balance	0.7098	**0.7439**	0.6981	**0.6272**	0.6744	0.6517	0.7665	**0.9268**	0.7832	0.7328	0.7043	0.7376	0.6932	0.7844	**0.7382**
	FIR	0.3463	**0.5936**	0.5469	0.5811	0.502	0.8365	0.6914	**0.839**	0.6275	0.5562	0.583	0.5479	0.475	**0.7143**	**0.6273**

7. 결론 및 향후 연구

소프트웨어 결함 예측에서 클래스 비율을 맞추는 것은 예측 성능을 높이기 위한 과정 중 하나이다. 소프트웨어 결함 예측에서는 실제 결함 중에서 모델이 결함이라고 예측하는 지표인 Recall을 높이고, 전체 비결함 클래스 수에 대해 결함으로 잘못 분류된 비결함 클래스 수의 비율인 PF는 낮추는 것을 목표로 한다.

본 연구에서는 소프트웨어 결함 예측 데이터에 적합한 GAN 모델은 무엇인지 실험적으로 검증한다. 소프트웨어 결함 예측에 GAN과 cGAN, CTGAN 모델이 적용된 관련 연구는 있지만[9,17] WGAN과 TGAN 모델이 소프트웨어 결함 예측에 적용된 사례는 없다. 따라서 본 연구에서는 WGAN과 TGAN 모델이 소프트웨어 결함 예측에 적용 가능한지 확인하고, 기존 연구에서 사용한 GAN 모델들의 성능을 검증하기 위해 새로운 데이터셋을 추가로 사용한다. 실험 결과, 클래스 불균형 문제를 해결하지 않았을 때 PF 성능은 우수하지만, Recall과 Balance의 성능은 낮았다. 반면 GAN 모델을 사용해 클래스 불균형을 해결한 결과 Balance 성능을 향상시킬 수 있었다. 소프트웨어 결함 예측 데이터에 적합한 GAN 모델을 확인하기 위해 GAN 모델들의 성능을 비교 실험한 결과, CTGAN 모델이 가장 적합함을 보였고, 통계적으로 다른 GAN 모델 대비 유의미한 성능 차이가 있음을 보였다. 또한 WGAN과 TGAN의 적용성 여부를 확인한 결과, 두 모델 의 평균 Balance 성능이 0.6 이상이므로 소프트웨어 결함 예측 데이터셋에 적합함을 보였다. 특히, TGAN은 소프트웨어 결함 예측 데이터셋에서 결함 오보율을 낮추고, 코드 검사 노력도를 낮출 수 있음을 확인할 수 있었다. 더 나아가 본 연구에서는 CTGAN 모델의 성능에 영향을 미치는 하이퍼파라미터 값 조정을 통해 CTGAN의 성능을 다소 향상시킬 수 있음을 실험적으로 보였다.

향후 데이터셋을 추가하고, 결함 예측력을 향상시킬 수 있는 CTGAN의 하이퍼파라미터를 탐색 및 추가할 계획이다. 본 연구를 통해 소프트웨어 결함 예측 데이터가 가지고 있는 클래스 불균형 문제를 CTGAN 모델을 통해 해결하고 소프트웨어 결함 예측의 성능을 향상시킬 수 있을 것으로 기대한다.

References

[1] Malhotra, Ruchika, and Shine Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing*, 343 (2019): 120-140.

[2] Wang, Shuo and Xin Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability* 62.2 (2013): 434-443.

[3] Yin, Shizhuang, et al., "Summary of software reliability Research," *IOP Conference Series: Materials Science and Engineering*, Vol. 1043, No. 5, IOP Publishing, 2021.

[4] Chawla, Nitesh V., et al., "SMOTE: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, 16 (2002): 321-357.

[5] Bejjanki, Kiran Kumar, Jayadev Gyani, and Narsimha Gugulothu, "Class imbalance reduction (CIR): a novel approach to software defect prediction in the presence of class imbalance," *Symmetry*, 12.3 (2020): 407.

[6] Feng, Shuo, et al., "COSTE: Complexity-based OverSampling TEchnique to alleviate the class imbalance problem in software defect prediction," *Information and Software Technology* (2020): 106432.

[7] Goodfellow, Ian J., et al., "Generative adversarial networks," *arXiv preprint arXiv:1406.2661* (2014).

[8] Choi, Jaehoon, Taekyung Kim, and Changick Kim, "Self-ensembling with gan-based data augmentation for domain adaptation in semantic segmentation," *Proc. of the IEEE/CVF International Conference on Computer Vision*, 2019.

[9] Choi, Lee et al., "Comparative Study of Generative Adversarial Network models for Software Defect Prediction," *KSC 2020*, pp. 171-173.

[10] Mirza, Mehdi and Simon Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784* (2014).

[11] Xu, Lei, et al., "Modeling tabular data using conditional gan," *arXiv preprint arXiv:1907.00503* (2019).

[12] Arjovsky, Martin, Soumith Chintala, and Léon Bottou, "Wasserstein generative adversarial networks," International conference on machine learning, *PMLR*, 2017.

[13] Xu, Lei and Kalyan Veeramachaneni, "Synthesizing tabular data using generative adversarial networks," *arXiv preprint arXiv:1811.11264* (2018).

[14] Sawilowsky, Shlomo S. "Nw effect size rules of thumb," *Journal of Modern Applied Statistical Methods* 8.2 (2009): 26.

[15] Song, Qinbao, Yuchen Guo, and Martin Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering* 45.12 (2018): 1253-1269.

[16] Gong, Lina, Jiang, et al., "Conditional Domain Adversarial Adaptation for Heterogeneous Defect Prediction," *IEEE Access 8* (2020): 150738-150749.

[17] Lee, Jin et al., "Software Defect Prediction using Generative Adversarial Networks," *KCSE* 2019: 116-120.

[18] Altinger, Harald, et al., "Performance tuning for

automotive software fault prediction," *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2017.
[19] Altinger, Harald, et al., "A novel industry grade dataset for fault prediction based on model-driven developed automotive embedded software," *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, IEEE, 2015.
[20] Xu, Zhou, et al., "Cross-version defect prediction via hybrid active learning with kernel principal component analysis," *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, IEEE, 2018.
[21] Shin, Yonghee, et al., "Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities," *IEEE transactions on software engineering* 37.6 (2010): 772-787.

최 지 원
2020년 전북대학교 소프트웨어공학과 학사. 2020년~현재 전북대학교 소프트웨어공학과 석사과정. 관심분야는 소프트웨어 결함 예측, SE4AI, AI4SE, 엣지 컴퓨팅

이 재 욱
2016년~현재 전북대학교 소프트웨어공학과 학사과정. 관심분야는 소프트웨어 신뢰성, 인공지능, 그래프 생성, 분자 최적화, 신약 개발

류 덕 산
2012년 카이스트 및 카네기멜론대학교 소프트웨어공학 복수학위 석사. 2016년 카이스트 전산학부 박사. 2018년 9월~현재 전북대학교 소프트웨어공학과 조교수. 관심분야는 SE4AI, AI4SE, 인공지능기반 소프트웨어분석, 소프트웨어 결함 예측, 소프트웨어 신뢰성, 소프트웨어 메트릭스, 소프트웨어 품질보증, 자율시스템

김 순 태
2003년 중앙대학교 컴퓨터공학과 학사
2007년 서강대학교 컴퓨터공학과 석사
2010년 서강대학교 컴퓨터공학과 박사
2014년~현재 전북대학교 소프트웨어공학과 교수. 관심분야는 소프트웨어공학, 블록체인/스마트컨트랙트, 인공지능