석사학위논문

소프트웨어 결함 예측에서 트리 기반 앙상블 학습의 하이퍼파라미터 최적화 기법 비교

Comparison of hyperparameter optimization techniques of tree-based ensemble learning in software defect prediction

이 유 진

한양대학교 대학원

2024년 12월

석사학위논문

소프트웨어 결함 예측에서 트리 기반 앙상블 학습의 하이퍼파라미터 최적화 기법 비교

Comparison of hyperparameter optimization techniques of tree-based ensemble learning in software defect prediction

지도교수 Scott Uk-Jin Lee

이 논문을 공학 석사학위논문으로 제출합니다.

2025년 2월

한양대학교 대학원

지능정보융합공학과

이 유 진

이 논문을 이유진의 석사학위 논문으로 인준함

2025년 2월

심 사 위 원 장 : 조 성 현 (인)

심 사 위 원 : 이 석 복 (인)

심 사 위 원 : Scott Uk-Jin Lee (인)

한양대학교 대학원

목차

요약 ····· 5
제 1 장 서 론 ····· 6
1.1 연구 배경 및 목적 ····· 6
1.2 연구 질문 및 목표 ····· 8
1.3 논문의 구성 ····· 9

제 2 장 관련 연구 ····· 9
2.1 소프트웨어 결함 예측에 사용되는 앙상블 모델 ····· 9
2.2 하이퍼파라미터 최적화 기법을 적용한 앙상블 및 단일 모델 ····· 11

제 3 장 이론적 배경 ····· 12
3.1 트리 기반 앙상블 모델 ····· 12
3.1.1 랜덤 포레스트 ····· 13
3.1.2 엑스트라 트리 ····· 14
3.1.4 XGBoost ····· 16
3.1.5 LightGBM ····· 17
3.1.6 CatBoost ····· 18
3.1.7 AdaBoost ····· 19
3.2 하이퍼파라미터 최적화 기법 ····· 20
3.2.1 그리드서치 ····· 21
3.2.2 랜덤서치 ····· 22
3.2.3 베이지안 최적화 ····· 23
3.2.4 유전 알고리즘 ····· 24

제 4 장 하이퍼파라미터 최적화 기법 비교 실험 ····· 25
4.1 데이터셋 설명 ····· 25
4.1.1 사용된 나사 데이터셋의 특징 ····· 25
4.1.2 데이터 전처리 과정 ····· 27
4.2 실험 설계 및 평가 방법 ····· 28
4.2.1 실험 설계 방식 ····· 28
4.2.3 성능 평가 지표 ····· 34

제 5 장 ····· 36
5.1 모델별 최적화 성능 분석 ····· 36
5.2 최적화 알고리즘별 성능 비교 ····· 42
5.2.1 최적화 알고리즘에 따른 모델 성능 비교 ····· 42
5.2.2 최적화 알고리즘별 모델 간 일관성 평가 ····· 44
5.3 모델과 최적화 알고리즘 간 상호작용 분석 ····· 46
5.3.1 모델과 최적화 알고리즘 조합이 성능에 미치는 상호작용 효과 분석 ····· 46
5.3.2 최적의 모델-알고리즘 조합 도출 ····· 48

제 6 장 결론 ····· 51
참 고 문 헌 ····· 53

표 목차

표 1 소프트웨어 결함 데이터 셋 설명 ····· 26
표 2 트리 기반 앙상블 모델별 최적화 기법 적용에 따른 성능 비교 ····· 37
표 3 트리 기반 앙상블 모델에 대한 최적화 기법별 성능 비교 ····· 42
표 4 최적의 모델-알고리즘 조합 도출 결과 ····· 48

그림 목차

그림 1 훈련 데이터와 테스트 데이터의 분할의 예시 코드 ····· 29
그림 2 랜덤 포레스트 모델 초기화 예시 코드 ····· 32
그림 3 정확도 계산식 ····· 34
그림 4 정밀도 계산식 ····· 35
그림 5 재현율 계산식 ····· 35
그림 6 F1 스코어 계산식 ····· 35
그림 7 최적화 방법별 모델 성능(Accuracy) ····· 40
그림 8 최적화 방법별 모델 성능(F1 Score) ····· 41
그림 9 최적화 방법에 의한 모델 간 평균 정확도 ····· 44
그림 10 모델-최적화 알고리즘 상호 작용 그래프 ····· 46
그림 11 최적화 방법에 따른 모델별 정확도 비교 ····· 47

요약

소프트웨어 결함 예측은 소프트웨어 개발 과정에서 잠재적인 오류를 조기에 탐지하고 품질을 개선하며 유지보수 비용을 절감하는 데 중요한 역할을 한다. 특히, 정확하고 효율적인 결함 예측은 개발 리소스를 최적화하고 프로젝트의 성공 가능성을 높이는 데 기여한다. 본 연구는 소프트웨어 결함 예측 문제를 해결하기 위해 트리 기반 앙상블 모델(Random Forest, ExtraTrees, Gradient Boosting, XGBoost, LightGBM, CatBoost, AdaBoost)을 대상으로 네 가지 하이퍼파라미터 최적화 기법(그리드 서치, 랜덤 서치, 베이지안 최적화, 유전 알고리즘)이 모델 성능에 미치는 영향을 체계적으로 비교하였다. 각 최적화 기법은 다양한 소프트웨어 결함 데이터셋을 활용하여 평가되었으며, 성능 비교를 위해 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 스코어(F1 Score), AUC(Area Under the Curve)와 같은 다섯 가지 지표를 사용하였다.

연구 결과, 최적화 기법은 모델 성능 향상에 중요한 영향을 미치며, 특히 Genetic Algorithm은 Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 구조의 모델에서 가장 우수한 성능을 나타냈다. 반면, Grid Search와 Random Search는 ExtraTrees, LightGBM과 같은 상대적으로 단순한 모델에서 안정적이고 일관된 성능을 보였다. 이러한 결과는 모델의 구조와 하이퍼파라미터 탐색 기법 간의 상호작용 효과가 존재함을 시사한다.

또한, 모델과 최적화 기법 간 상호작용을 분석한 결과, 특정 모델에서 특정 최적화 기법이 유의미한 성능 차이를 나타냄을 확인하였다. 예를 들어, Bayesian Optimization은 CatBoost와 Gradient Boosting에서 일관된 성능을 보였지만, RandomForest나 Adaboost에서는 상대적으로 낮은 성능을 기록하였다. 이러한 분석은 각 모델의 특성에 따라 최적화 기법을 선택하는 것이 성능 극대화에 필수적임을 보여준다.

본 연구는 소프트웨어 결함 예측에서 하이퍼파라미터 최적화 기법의 중요성을 실증적으로 평가하며, 모델 특성과 최적화 기법의 조합이 성능에 미치는 영향을 심층적으로 분석하였다. 이를 통해 소프트웨어 품질 보증 시스템에서 최적의 예측 성능을 달성하기 위한 구체적인 방향성을 제시하였다.

제 1 장 서 론

1.1 연구 배경 및 목적

소프트웨어 개발의 복잡성과 규모가 지속적으로 증가하면서, 소프트웨어 시스템의 결함을 조기에 예측하고 해결하는 작업은 더욱 중요해지고 있다. 소프트웨어 결함이 발생하면 운영 비용이 증가하고 사용자 만족도가 저하될 뿐만 아니라, 심각한 경우 시스템의 안전성과 신뢰성까지 크게 저해할 수 있다. 가장 효율적인 품질 보증 기술을 사용하여 소프트웨어 결함을 식별하고 수정하는 것은 어렵고 비용이 많이 드는 프로세스이다[1]. 소프트웨어 결함 예측(Software Defect Prediction, SDP)은 소프트웨어 품질 보증과 유지보수 과정에서 핵심적인 역할을 한다. 복잡한 소프트웨어 개발 프로젝트에서 발생하는 결함은 단순히 기능적인 오류뿐만 아니라, 치명적인 시스템 중단, 데이터 손실, 보안 위협 등을 초래할 수 있다. 특히, 결함이 개발 후반 단계에서 발견될 경우 수정 비용은 기하급수적으로 증가하며, 이는 개발 일정의 지연과 추가적인 유지보수 비용으로 이어질 수 있다. 따라서, 소프트웨어 개발 초기 단계에서 결함을 조기에 탐지하고 이를 효과적으로 예측하는 것은 개발 효율성을 높이고 품질을 유지하기 위해 필수적이다. 특히, 결함 예측의 정확도를 높이기 위한 다양한 기법이 개발되고 있으며, 이들 기법의 성능을 최적화하는 방법이 중요한 연구 주제로 떠오르고 있다.

구체적으로, SDP는 소스 코드 메트릭(코드 복잡도, 결합도, 라인 수 등)과 결함 여부 간의 관계를 학습하여 결함이 포함될 가능성이 높은 소프트웨어 모듈을 예측하는 데 중점을 둔다. 이러한 접근법은 개발 팀이 한정된 자원을 가장 취약한 영역에 우선적으로 투입할 수 있도록 하며, 이를 통해 전체 프로젝트의 품질을 개선하고 유지보수의 효율성을 극대화할 수 있다.

최근 연구에서 트리 기반 앙상블 모델이 소프트웨어 결함 예측에서 뛰어난 성능을 보이며 널리 사용되고 있다[4]. 트리 기반 앙상블 모델은 여러 개의 결정 트리(Decision Tree)를 결합하여 예측 정확도를 높이는 기법으로, 각 트리가 개별 예측을 수행하고 이를 종합하여 최종 예측을 산출한다. 이 모델은 특히 불균형 데이터나 결측치가 포함된 데이터에 대해 높은 내성을 가지며, 데이터의 노이즈를 효과적으로 처리할 수 있다는 강점을 지닌다. 대표적인 트리 기반 앙상블 모델로는 랜덤 포레스트

(Random Forest), 엑스트라 트리(Extra Trees), 그라디언트 부스팅 머신(Gradient Boosting Machine, GBM), XGBoost, LightGBM, CatBoost, AdaBoost 등이 있으며, 이들 모델은 각기 다른 방식으로 트리를 결합하여 성능을 극대화하고 있다. 이러한 모델은 소프트웨어 결함 예측뿐 아니라, 다양한 분야에서 높은 성능을 보이는 유망한 예측 기법으로 주목받고 있다.

트리 기반 앙상블 모델이 우수한 성능을 발휘하기 위해서는 각 모델의 하이퍼파라미터를 최적화하는 과정이 필수적이다. 하이퍼파라미터는 모델이 데이터를 학습하는 방식과 일반화 성능에 직접적인 영향을 미치는 중요한 변수로, 적절한 하이퍼파라미터 설정이 이루어져야만 모델이 데이터에 적합하게 학습할 수 있다[2]. 그러나 각 모델이 가지는 다양한 하이퍼파라미터의 조합을 모두 시도하는 것은 현실적으로 불가능하며, 최적의 성능을 달성하기 위해 효율적인 하이퍼파라미터 최적화 방법이 필요하다. 최근 연구에서는 그리드 서치(Grid Search), 랜덤 서치(Random Search), 베이지안 최적화(Bayesian Optimization), 유전 알고리즘(Genetic Algorithm) 등 여러 최적화 기법이 모델의 성능을 높이는 효과적인 방법으로 주목받고 있다.

본 연구는 SDP의 중요성을 바탕으로, 각 최적화 기법이 트리 기반 모델에서 예측 성능을 어떻게 향상시키는지 평가하는 것이다. 예를 들어, 그리드 서치는 가능한 모든 조합을 테스트하여 최적의 하이퍼파라미터를 찾지만, 연산 시간이 매우 길어질 수 있는 단점이 있다. 반면, 랜덤 서치는 일부 조합만을 무작위로 탐색하여 시간 효율성을 높일 수 있지만, 최적의 조합을 놓칠 가능성도 존재한다. 베이지안 최적화는 모델 성능에 대한 사전 정보를 기반으로 탐색을 수행하여, 보다 효율적으로 최적의 하이퍼파라미터를 찾고, 유전 알고리즘은 진화적 접근을 통해 탐색 공간의 다양한 영역을 빠르게 탐색하는 장점을 지닌다.

본 논문은 이러한 최적화 기법들이 소프트웨어 결함 예측 상황에서 트리 기반 앙상블 모델의 성능에 미치는 영향을 체계적으로 평가하는 것을 목적으로 한다. 다양한 최적화 기법을 트리 기반 앙상블 모델에 적용하고, 성능을 비교 분석함으로써, 소프트웨어 결함 예측의 정확도를 높이는 데 유용한 정보를 제공하고자 한다. 이를 통해 소프트웨어 결함 예측에서 트리 기반 앙상블 모델의 최적화에 대한 지침을 제시하고, 궁극적으로는 결함 예측의 정확성과 신뢰성을 강화하는 데 기여하고자 한다.

1.2 연구 질문 및 목표

본 연구는 트리 기반 앙상블 모델의 하이퍼파라미터 최적화에 대한 종합적인 이해를 제공하기 위해 다음과 같은 주요 연구 질문을 다룬다.

1. 특정 최적화 기법이 특정 모델에서 더 높은 성능을 나타내는가?
본 연구에서는 랜덤 포레스트(Random Forest), 엑스트라 트리(Extra Trees), 그라디언트 부스팅 머신(GBM), XGBoost, LightGBM, CatBoost, AdaBoost 등의 트리 기반 앙상블 모델을 대상으로 각 최적화 기법이 성능에 미치는 영향을 비교한다. 이를 통해 특정 최적화 기법이 특정 모델에서 특히 높은 성능을 나타내는지 여부를 확인하여, 모델과 최적화 기법 간의 최적 조합을 탐색한다.

2. 하이퍼파라미터 최적화 기법과 모델 간 상호작용이 존재하는가?
최적화 기법과 모델 간에 성능에 영향을 미치는 상호작용 효과가 존재하는지 분석한다. 특정 모델이 특정 최적화 기법과 결합될 때 성능이 유의미하게 향상되는 경우, 이를 통해 최적화 기법과 모델 간의 시너지 효과를 확인할 수 있다. 이러한 상호작용 분석을 통해 모델-최적화 기법 조합이 성능에 미치는 종합적인 효과를 파악하고, 소프트웨어 결함 예측에 가장 적합한 조합을 식별하고자 한다.

본 논문은 이러한 연구 질문에 대한 답을 통해 최적화 기법과 모델의 조합이 성능에 미치는 영향을 체계적으로 분석하는 것을 목표로 한다. 최적화 기법들이 모델의 성능에 미치는 구체적인 영향과, 소프트웨어 결함 예측에서 가장 효과적인 최적화 기법을 도출함으로써, 소프트웨어 결함 예측 모델 개발에 유용한 정보를 제공하고자 한다.

여기까지 보면 “앙상블 모델의 하이퍼파람 최적화” 문제 같고 버그 예측과 전혀 관계 없어 보이는데... 이부분이 잘 들어나도록 수정하면 좋겠어요

1.3 논문의 구성

본 논문은 다음과 같이 구성된다. 2장에서는 트리 기반 앙상블 모델과 하이퍼파라미터 최적화 기법에 대한 기존 연구를 소개하고, 본 연구의 차별성을 논의한다. 3장에서는 본 연구에 사용된 트리 기반 앙상블 모델과 하이퍼파라미터 최적화 기법의 이론적 배경을 다룬다. 4장에서는 연구에 사용된 데이터셋과 실험 설계를 설명하며, 실험

평가 지표와 방식을 제시한다. 5장에서는 실험 결과를 분석하고 모델과 최적화 기법 간의 상호작용을 논의한다. 마지막으로, 6장에서는 연구의 결론과 향후 연구 방향을 제시한다.

제 2 장 관련 연구

본 장에서는 소프트웨어 결함 예측에 사용되는 앙상블 모델 연구를 2.1장에서 다루고, 2.2장에서는 하이퍼파라미터 최적화 기법을 적용한 앙상블 및 단일 모델 연구를 소개한다.

2.1 소프트웨어 결함 예측에 사용되는 앙상블 모델

소프트웨어 결함 예측을 위한 지능형 앙상블 모델 연구[3]에서는 소프트웨어 결함 예측의 정확도를 높이기 위해 다중 분류 모델을 결합한 지능형 앙상블 접근법을 제안하였다. 연구는 Random Forest, Support Vector Machine, Naive Bayes, Artificial Neural Network의 네 가지 알고리즘을 개별적으로 훈련한 후, 각 모델의 예측 결과를 투표 기반 앙상블 방식으로 결합하여 최종 예측 성능을 향상시키는 구조를 채택하였다. 실험에서는 NASA MDP 저장소의 다양한 결함 데이터셋을 활용하여 제안한 모델의 성능을 평가하였고, 기존의 개별 분류 모델 및 20여 가지 결함 예측 기법 대비 우수한 성능을 확인했다. 본 연구는 앙상블 접근법을 통해 소프트웨어 결함 예측의 예측력을 강화하는 방안을 제시함으로써, 소프트웨어 품질 개선과 유지보수 비용 절감에 기여할 수 있는 가능성을 보여주었다.

트리 기반 앙상블을 활용한 소프트웨어 결함 예측 연구[4]에서는 트리 기반 앙상블 모델을 소프트웨어 결함 예측에 적용하여 예측 성능을 개선하고자 했다. 연구는 소프트웨어 결함 예측의 정확도를 높이기 위해 Random Forest, Gradient Boosting Machine (GBM), XGBoost와 같은 트리 기반 앙상블 알고리즘을 사용했으며, 이 모델들의 강력한 분류 성능이 결함 있는 모듈을 식별하는 데 적합하다는 점을 강조했다. 실험에서는 다양한 소프트웨어 결함 데이터셋을 활용하여 트리 기반 앙상블 모델의 성능을 평가하였고, 연구는 트리 기반 앙상블 모델이 소프트웨어 결함 예측의 강력한 대안이 될 수 있음을 보여주며, 특히 결함 데이터의 불균형 및 복잡한 특성에 대응할 수 있는 가능성을 제시했다. 그러나 모델 간의 성능 차이에 대한 상세한 분석이 부족

하며, 하이퍼파라미터 최적화 없이 기본 설정으로 실험이 수행되었다.

최적화된 트리 기반 앙상블의 스태킹 일반화를 활용한 소프트웨어 결함 예측 연구[5] 논문은 트리 기반 앙상블 모델의 성능을 향상시키기 위해 최적화된 여러 모델을 스태킹 방식으로 결합하는 접근법을 제안하였다. 연구는 Random Forest, Gradient Boosting Machine, XGBoost 등 트리 기반 모델의 하이퍼파라미터를 최적화한 후, 이들 모델을 메타 학습기로 스태킹하여 예측 성능을 극대화하였다.
해당 연구는 다양한 소프트웨어 결함 데이터셋을 이용해 실험을 수행했으며, 단일 모델이나 단순 앙상블 방법보다 높은 예측 성능을 기록했다. 특히, 최적화된 모델을 스태킹함으로써 데이터의 복잡한 패턴을 효과적으로 학습하고, 소프트웨어 결함 예측의 정확도와 안정성을 개선할 수 있음을 증명했다. 그러나 최적화된 모델의 조합 방법론과 메타 학습기의 영향에 대한 체계적 분석이 부족하다는 문제가 있다.

2.2 하이퍼파라미터 최적화 기법을 적용한 앙상블 및 단일 모델

하이퍼파라미터 튜닝을 통한 최적화된 앙상블 학습 기반 소프트웨어 결함 예측 연구[6]에서는 소프트웨어 결함 예측 성능을 높이기 위해 하이퍼파라미터 최적화가 적용된 앙상블 학습 접근법을 제안하였다. 연구는 Random Forest, Gradient Boosting Machine, XGBoost와 같은 트리 기반 모델에 최적화된 하이퍼파라미터를 적용하고, 이들 모델을 앙상블하여 결함 예측의 정확성을 극대화하는 방법을 탐구했다.
실험 결과, 최적화된 앙상블 모델은 기존의 개별 모델 또는 단순한 앙상블 방식보다 높은 예측 성능을 보였으며, 특히 하이퍼파라미터 튜닝이 예측 성능 향상에 미치는 긍정적인 영향을 확인했다. 해당 연구는 하이퍼파라미터 최적화와 앙상블 학습을 결합함으로써 소프트웨어 결함 예측의 정확도와 신뢰성을 크게 향상시킬 수 있음을 보여주며, 하이퍼파라미터 튜닝이 소프트웨어 결함 예측 모델 성능 개선에 있어 중요한 요소임을 강조했다. 그러나 하이퍼파라미터 튜닝이 모델 성능에 미치는 긍정적인 영향을 확인했지만, 여러 최적화 방법 간의 비교는 부족하며, 특정 모델에만 적용되었다.

소프트웨어 결함 예측에서 랜덤 포레스트 분류의 다양한 하이퍼파라미터 튜닝 기법 비교 연구[8]는 소프트웨어 결함 예측의 성능을 향상시키기 위해 Random Forest 모델에 다양한 하이퍼파라미터 최적화 기법을 적용하고, 데이터 불균형 문제와 특징 선택을 해결하는 방법을 비교한 연구로, SMOTE(Synthetic Minority Over-sampling Technique)[7]를 활용하여 데이터 불균형을 조정하고, 유전 알고리즘(Genetic Algorithm)을 통해 중요한 특징을 선택하여 모델의 예측 성능을 최적화했다. 이 연구는 Grid Search, Random Search, Bayesian Optimization 등 다양한 하이퍼파라미터 최적화 방법을 사용하여 랜덤 포레스트 모델을 튜닝한 결과를 비교하였고, 최적화된 모델이 데이터 불균형 문제와 특징 선택을 고려할 때 결함 예측의 정확도가 향상됨을 입증했다. 본 연구는 하이퍼파라미터 튜닝, 데이터 불균형 처리, 그리고 최적의 특징 선택이 결합되어 소프트웨어 결함 예측에서 Random Forest 모델의 성능을 극대화할 수 있음을 보여주었다.

최적화된 하이퍼파라미터를 적용한 XGBoost 기반 소프트웨어 결함 예측 연구[9]는 소프트웨어 결함 예측에서 XGBoost 모델의 예측 성능을 극대화하기 위해 하이퍼파라미터 최적화 기법을 적용한 연구다. XGBoost는 강력한 성능을 가진 그라디언트 부스팅 알고리즘으로, 연구에서는 Bayesian Optimization과 같은 최적화 기법을 통해 주요 하이퍼파라미터를 조정하여 모델의 정확도를 개선하고자 했다.
실험에서는 여러 소프트웨어 결함 데이터셋을 사용하여 최적화된 XGBoost 모델의 성능을 평가하였으며, 최적화된 하이퍼파라미터가 모델의 예측력을 크게 향상시키는 결과를 확인했습니다. 본 연구는 하이퍼파라미터 최적화가 XGBoost와 같은 강력한 모델에서도 성능 개선에 중요한 역할을 할 수 있음을 보여주며, 소프트웨어 결함 예측에서 XGBoost와 최적화 기법의 결합이 유효한 접근법임을 입증했다. 그러나 SMOTE와 유전 알고리즘을 활용해 데이터 불균형과 특징 선택 문제를 해결하려 했지만, 이러한 전처리 과정이 다른 모델에서 어떻게 적용될 수 있는지에 대한 분석이 부족하다. 또한 연구 모델이 랜덤 포레스트에 국한되어 있어, 다른 트리 기반 모델들과의 비교가 이루어지지 않았다.

이와 같이, 다양한 앙상블 기법과 최적화 방법을 활용한 연구들은 소프트웨어 결함 예측의 정확성과 신뢰성을 크게 향상시킬 수 있는 가능성을 제시하며, 하이퍼파라미터 최적화는 트리 기반 앙상블 모델의 예측 성능을 극대화하고 소프트웨어 결함 예측의 정확성과 신뢰성을 향상시키는 데 핵심적인 역할을 한다. 그러나 이러한 연구들은

여러 최적화 방법간의 비교가 부족하거나 특정 모델에만 적용되었다는 한계가 있다.

이에 따라 본 연구는 다양한 앙상블 모델과 최적화 기법을 통합적으로 분석하고, 데이터 전처리 및 최적화 과정의 상호작용을 나타내며, 실용성과 일반화 가능성을 고려한 접근 방식을 통해 기존 연구의 한계를 극복한다. 이는 소프트웨어 결함 예측의 정확성과 신뢰성을 높이는 데 기여하며, 효율적인 소프트웨어 품질 관리와 유지보수 비용 절감을 위한 실질적인 방법를 제시한다.

관련 연구가 7개 밖에 없나요? 관련 연구 조사가 부족해 보입니다. 그리고 관련연구의 내용 요약 후 본인의 연구와의 차이점 (보통은 관련 연구의 안 좋은 점을 본인 연구에서 개선하였다거나 하는 부분을 강조하며)을 드러내는데.. 이러한 기술 보다는 오히려 관련연구가 본인 연구인 것처럼 서술이 되어 있어요. 이부분은 수정이 필요해 보입니다. 이 커멘트는 2장 전체 관련 연구에 대한 것이예요.

제 3 장 이론적 배경

3.1 트리 기반 앙상블 모델

트리 기반 앙상블 모델은 여러 개의 결정 트리(Decision Tree)를 결합하여 예측의 정확성과 견고성을 향상시키는 대표적인 머신러닝 기법으로, 다양한 예측 및 분류 문제에서 널리 활용되고 있다. 이러한 모델은 개별 결정 트리들이 가지는 한계를 보완하며, 모델의 예측력을 높이기 위해 트리들의 결합 방식을 활용한다[10]. 기본적으로 트리 기반 앙상블은 배깅(Bagging)과 부스팅(Boosting)이라는 두 가지 주요 접근 방식을 통해 동작한다.

배깅 기법은 무작위로 선택된 샘플을 사용하여 다수의 트리를 독립적으로 학습시키고, 예측 단계에서 다수결 투표(분류 문제의 경우)나 평균값(회귀 문제의 경우)을 통해 최종 결과를 도출한다[11]. 이 과정은 모델의 분산을 줄이고 과적합을 방지하여 예측의 견고성을 높이는 효과를 제공한다. 배깅의 대표적인 예로는 랜덤 포레스트(Random Forest)가 있다. 랜덤 포레스트는 개별 트리 학습 시 일부 특징을 무작위로 선택하여 모델을 구성함으로써, 다수의 트리가 서로 다른 예측을 수행할 수 있게 한다[12]. 이는 모델의 일반화 성능을 향상시키고 데이터의 노이즈에 강인한 특성을 갖게 만든다.

반면, 부스팅은 연속적으로 학습하는 트리들이 이전 단계에서 발생한 예측 오류를 보완하는 방식으로 작동한다[13]. 부스팅은 각 단계에서 학습한 트리의 오류에 가중치

를 부여해 모델의 예측력을 점진적으로 향상시킨다. 이 접근 방식의 대표적인 예로는 그라디언트 부스팅(Gradient Boosting), XGBoost, LightGBM등이 있다. XGBoost는 빠른 연산 속도와 정교한 가지치기 기법을 통해 성능을 대폭 향상시킨 모델로, 다양한 데이터 과학 경진대회에서 주목받고 있다[14]. LightGBM은 리프 중심의 트리 성장 방식과 메모리 최적화를 통해 대규모 데이터셋에서 효율적으로 작동하는 특성을 갖추고 있다[15]. 이와 더불어, CatBoost는 범주형 데이터를 효과적으로 처리할 수 있도록 설계된 부스팅 모델로, 범주형 특징의 자동 변환 및 순서 편향을 줄이는 기능을 통해 높은 성능을 보인다[16].

트리 기반 앙상블 모델의 주요 장점은 불균형 데이터, 결측치, 노이즈가 포함된 데이터에 대해 강인한 성능을 보이며, 복잡한 데이터 관계를 효과적으로 학습할 수 있다는 점이다[17]. 그러나 모델의 복잡성과 많은 하이퍼파라미터로 인해 최적의 성능을 발휘하기 위해서는 하이퍼파라미터 최적화가 필수적이다. 최적화된 하이퍼파라미터 설정은 모델의 일반화 성능과 예측 정확도를 결정짓는 중요한 요소로 작용하며, 다양한 최적화 기법이 이에 대한 연구로 지속적으로 발전하고 있다.

3.1.1 랜덤 포레스트

랜덤 포레스트(Random Forest)는 다수의 결정 트리(Decision Trees)를 결합하여 예측 성능을 향상시키는 앙상블 학습 기법으로, 분류 및 회귀 문제에서 높은 예측 정확도와 안정성을 제공하는 것으로 알려져 있다[11]. 이 모델은 배깅(Bagging) 기법을 기반으로 하며, 개별 트리들이 서로 다른 데이터 샘플과 특징 집합을 사용하여 독립적으로 학습됨으로써 과적합(overfitting)을 방지하고 모델의 일반화 성능을 높인다.

이 모델은 기본적으로 두 가지 주요 단계로 구성된다. 첫째, 부트스트랩 샘플링(Bootstrap Sampling)을 통해 원본 데이터셋에서 중복을 허용한 무작위 샘플을 생성하여 각 트리를 학습시킨다. 이로 인해 각 트리는 서로 다른 데이터 샘플에 기반하여 학습되며, 모델의 예측 다양성을 증대시킨다[12]. 둘째, 특징의 무작위 선택을 통해 각 노드의 분할 시 전체 특징 집합에서 무작위로 선택된 일부 특징만을 사용한다. 이는 트리 간 상관관계를 낮추고, 특정 특징에 지나치게 의존하는 현상을 방지하여 모델의 성능을 더욱 향상시킨다[18].

랜덤 포레스트는 개별 트리의 예측을 집계하여 최종 예측을 산출한다. 분류 문제의 경우, 다수결 투표를 통해 최종 클래스를 결정하며, 회귀 문제에서는 각 트리의 예측값을 평균내어 결과를 도출한다. 이러한 예측 집계 방식은 모델의 안정성을 높이고, 예측 오류를 줄이는 데 기여한다.

또한, 랜덤 포레스트는 다음과 같은 강점을 지닌다. 다양한 데이터 샘플링과 특징 무작위화를 통해 모델의 과적합을 효과적으로 방지한다. 또한, 각 특징의 중요도를 평가할 수 있어 모델의 해석성을 제공하며, 이는 변수 선택 및 데이터 이해에 유용하게 활용될 수 있다[17]. 마지막으로, 노이즈가 포함된 데이터나 결측치가 있는 데이터에 대해서도 강인한 성능을 보여주며, 다양한 예측 문제에 적합하다. 다만, 모델의 복잡성과 트리 수가 증가할수록 계산 비용이 커질 수 있으며, 다수의 트리로 구성된 모델의 해석이 어려울 수 있다는 한계도 존재한다.

3.1.2 엑스트라 트리

엑스트라 트리(Extremely Randomized Trees, Extra Trees)는 랜덤 포레스트(Random Forest)와 유사한 트리 기반 앙상블 모델로, 다수의 결정 트리를 결합하여 예측 성능을 향상시키는 기법이다. Geurts et al.[19]에 의해 제안된 엑스트라 트리는 트리 생성 과정에서 보다 높은 무작위성을 도입하여 모델의 편향과 분산을 조정한다. 이러한 무작위성은 모델의 다양성을 극대화하며, 예측 정확도와 훈련 속도의 향상을 목적으로 한다.

엑스트라 트리는 트리 구성 시 기존 랜덤 포레스트와 차별화된 방식을 채택한다. 랜덤 포레스트가 특정 노드에서 최적의 분할 기준을 찾기 위해 무작위로 선택된 특징 집합에서 최적의 기준을 결정하는 반면, 엑스트라 트리는 분할 기준 자체를 무작위로 설정한다. 구체적으로, 엑스트라 트리는 각 노드에서 선택된 특징에 대해 무작위 분할 지점을 결정하여 노드를 나눈다. 이러한 방식은 모델의 편향을 다소 증가시킬 수 있으나, 전체적으로 분산을 감소시켜 과적합(overfitting)을 방지하고 모델의 일반화 성능을 높인다[19]. 이로 인해, 엑스트라 트리는 연산적으로 효율적이며 대규모 데이터셋에서도 빠르게 학습할 수 있다.

엑스트라 트리는 랜덤 포레스트와 마찬가지로 부트스트랩 샘플링을 사용할 수 있으나, 필수적으로 적용하지는 않는다. 경우에 따라 전체 데이터셋을 이용하여 트리를 학습할 수 있으며, 이는 특정 데이터 패턴을 보다 다양한 방식으로 반영할 수 있도록 한다. 이러한 특성은 데이터의 다변성을 반영하고, 트리 간 상관관계를 낮춰 예측의 안정성을 높이는 데 기여한다.

엑스트라 트리의 주요 장점은 높은 무작위성 도입으로 인한 트리 간 상관관계의 감소와 모델의 빠른 학습 속도이다. 이러한 무작위성은 모델이 특정 특징에 과도하게 의존하는 것을 방지하고, 다양한 데이터 특성에 대한 예측 능력을 제공한다. 또한, 랜

덤 포레스트와 유사하게 변수 중요도 평가 기능을 제공하여 모델의 해석 가능성을 높인다[20]. 그러나 무작위성이 증가함에 따라 모델의 편향이 상대적으로 증가할 수 있으며, 이로 인해 예측 성능이 데이터 특성에 따라 다르게 나타날 수 있는 점은 엑스트라 트리의 한계로 지적된다.

결론적으로, 엑스트라 트리는 랜덤 포레스트와 유사한 구조를 기반으로 하면서도 높은 무작위성을 통해 모델의 예측 성능을 향상시키고, 효율적인 연산을 가능하게 한다. 이러한 특성은 소프트웨어 결함 예측과 같은 다양한 예측 문제에서 안정적이고 강력한 성능을 제공하며, 데이터의 복잡성과 잡음에 대응할 수 있는 유용한 모델로 평가된다.

3.1.3 그라디언트 부스팅 머신

그라디언트 부스팅 머신(GBM)은 앙상블 학습 기법으로, 다수의 약한 학습기(보통 결정 트리)를 결합하여 강력한 예측 모델을 생성하는 방식으로 작동한다. GBM은 각 단계에서 이전 모델이 만든 예측 오류를 보완하는 방식으로 트리를 추가하며, 이러한 과정은 순차적으로 진행된다[21]. 부스팅(Boosting) 기법을 활용하여 예측 성능을 점진적으로 개선하는 GBM은 회귀 및 분류 문제에서 뛰어난 예측력을 제공한다.

GBM의 핵심 아이디어는 각 단계에서 모델이 예측한 오류를 줄이는 방향으로 새로운 학습기를 추가하는 것이다. 초기 단계에서 기본 예측 모델을 구성한 후, 각 후속 단계는 이전 단계에서 예측한 값과 실제 값 간의 잔차(오류)를 예측하는 학습기를 추가로 훈련한다. 이를 통해 모델은 잔차를 점진적으로 보완하며, 최종적으로 높은 예측 성능을 달성한다. 이러한 과정은 경사 하강법(Gradient Descent)을 기반으로 잔차를 최소화하는 방향으로 학습을 진행하므로, 모델의 이름이 "그라디언트 부스팅 머신"이다[22].

GBM은 다양한 하이퍼파라미터를 통해 모델의 성능을 제어할 수 있다. 주요 하이퍼파라미터로는 학습률(Learning Rate), 트리의 최대 깊이(Max Depth), 최소 샘플 분할 수(Minimum Samples Split) 등이 있으며, 이러한 매개변수를 적절히 조정하면 모델의 예측 성능과 일반화 능력을 크게 향상시킬 수 있다. 그러나 과도한 복잡성은 과적합(overfitting)을 초래할 수 있으므로, 적절한 하이퍼파라미터 최적화가 필수적이다

[23].

이 모델은 다른 앙상블 모델과 비교했을 때 몇 가지 장점을 지닌다. 첫째, 각 단계에서 오류를 보완하는 방식으로 학습하므로, 예측 정확도가 매우 높다. 둘째, 다양한 손실 함수를 사용하여 사용자 정의 목적 함수를 최적화할 수 있어 유연성이 뛰어나다. 셋째, 회귀 및 분류 문제에 모두 적용 가능하며, 다양한 응용 분야에서 높은 성능을 보여준다. 다만, GBM은 순차적으로 모델을 학습시키므로 병렬 처리가 어렵고, 학습 속도가 상대적으로 느리다는 단점이 있다[14]. 또한, 하이퍼파라미터의 조정이 민감하게 작용할 수 있어 최적의 성능을 위해 신중한 설정이 필요하다.

결론적으로, 그라디언트 부스팅 머신은 예측 성능을 극대화하기 위해 각 단계에서 이전 오류를 보완하는 방식으로 동작하며, 다양한 문제에서 강력한 성능을 제공한다.

3.1.4 XGBoost

XGBoost(eXtreme Gradient Boosting)는 그라디언트 부스팅(Gradient Boosting) 알고리즘을 확장하여 예측 성능과 연산 효율성을 크게 향상시키기 위해 개발된 트리 기반 앙상블 모델이다. 이 모델은 Chen과 Guestrin[14]에 의해 개발되었으며, XGBoost는 회귀 및 분류 문제에서 뛰어난 예측 성능을 제공하며, 여러 분야에서 중요한 모델로 자리 잡고 있다.

XGBoost의 기본 개념은 기존의 그라디언트 부스팅과 유사하게 약한 학습기를 순차적으로 결합하여 모델의 예측 능력을 점진적으로 개선하는 것이다. 초기 단계에서는 간단한 예측 모델이 구성되고, 이후 각 단계에서는 이전 모델의 예측 오류를 보완하기 위해 새로운 학습기를 추가한다. 이러한 방식은 예측의 정확도를 점진적으로 높이며, 최종적으로 강력한 예측 모델을 구축할 수 있도록 한다.

이 모델은 기존의 그라디언트 부스팅과 비교할 때 몇 가지 중요한 개선 사항을 포함하고 있다. 우선, 정규화(Regularization) 기능이 추가되어 모델의 복잡성을 제어하고 과적합(overfitting)을 방지할 수 있다. 구체적으로, XGBoost는 L1 및 L2 정규화를 통해 트리 가중치를 규제하고, 모델이 데이터에 과도하게 적합되지 않도록 한다. 이러한 기능은 예측 성능의 안정성과 일반화 능력을 높이는 데 기여한다[21].

또한, 병렬 처리를 통해 연산 속도를 크게 향상시킨 점도 중요한 특징이다. 기존의 부스팅 알고리즘은 트리를 순차적으로 학습시키는 반면, XGBoost는 트리 분할을 병렬

로 수행함으로써 큰 데이터셋에 대해서도 빠르게 학습할 수 있다. 이 외에도 메모리 효율성을 높여 대규모 데이터셋을 처리할 때 더 적은 메모리를 사용하며, 분산 처리 기능을 통해 클러스터 환경에서도 학습이 가능하다.

또한, XGBoost는 결측치 처리(Missing Value Handling) 기능을 제공하여 데이터에 결측치가 있을 때도 안정적인 성능을 유지할 수 있다. 이는 데이터 전처리 단계에서 복잡성을 줄이고, 모델이 결측치를 자동으로 처리하여 예측의 일관성을 보장한다. 또한, 정교한 가지치기(Pruning) 전략을 채택하여 트리의 불필요한 노드를 미리 제거함으로써 학습 효율성을 높이고 계산 부담을 줄인다[24].

XGBoost는 사용자 정의 목적 함수(Custom Objective Function)를 지원하여 손실 함수를 유연하게 정의할 수 있어, 다양한 문제에 맞춤형 모델을 설계할 수 있다. 또한, 교차 검증(Cross Validation)을 통해 훈련 중 최적의 반복 횟수를 자동으로 결정하여 과적합을 방지하고, 모델의 성능을 최대화할 수 있도록 돕는다. 이러한 다양한 기능과 최적화 기법을 통해 데이터 분석 및 예측 문제에서 높은 성능을 발휘한다.

이와 같은 특징 덕분에 XGBoost는 대규모 데이터셋을 처리할 때 특히 강력한 성능을 발휘한다. 다만, 다양한 하이퍼파라미터를 적절히 설정하는 것이 모델의 성능에 중요한 영향을 미치므로, 최적의 성능을 위해 신중한 튜닝이 필요하다.

3.1.5 LightGBM

LightGBM (Light Gradient Boosting Machine)은 마이크로소프트에서 개발한 그라디언트 부스팅 프레임워크로, 대규모 데이터와 높은 차원의 특징을 효율적으로 처리할 수 있도록 설계된 트리 기반 앙상블 학습 알고리즘이다[15]. 기존의 그라디언트 부스팅 머신(GBM)과 비교하여 학습 속도와 메모리 사용 측면에서 크게 개선된 LightGBM은 데이터 처리 성능을 극대화하여 대규모 데이터셋에서 탁월한 성능을 발휘한다.

LightGBM은 리프 중심의 트리 성장 방식(Leaf-wise Tree Growth)을 채택하여 모델의 성능과 학습 효율성을 향상시킨다. 기존의 레벨 중심 트리 성장 방식(Level-wise Tree Growth)은 모든 레벨을 균등하게 확장하지만, LightGBM은 가장 손실을 많이 줄이는 리프(leaf)를 우선적으로 확장한다. 이로 인해 같은 깊이의 트리를 구성할 때 더 적은 수의 노드로 복잡한 패턴을 효과적으로 학습할 수 있다. 이는 학습 속도를 가속화하고 메모리 사용량을 줄이는 데 기여한다[25].

또한, LightGBM은 그레디언트 기반의 원샷(histogram-based) 방법을 통해 특징을 이산화하고, 이를 통해 트리 분할 시 연산 효율성을 극대화한다. 이 방식은 기존의 연속적인 분할 방식에 비해 계산 속도를 개선하며, 메모리 효율성을 높이는 데 중요한 역할을 한다. 특히, 대규모 데이터셋에서 트리 분할 시 불필요한 연산을 줄여주므로, 빠르고 효율적인 학습이 가능하다[26].

LightGBM은 균형 트리 분할과 데이터 정렬 최적화 기능을 통해 데이터의 불균형 문제와 정렬 비용을 최소화하여 예측 성능을 향상시킨다. 균형 트리 분할은 학습 데이터를 균등하게 나누어 노드가 불균형하게 성장하는 문제를 방지하고, 데이터 정렬 최적화는 트리 생성 시 데이터 정렬 비용을 줄여 학습 속도를 높인다.

또한, LightGBM은 다중 스레드 및 GPU 지원을 통해 병렬 처리를 효율적으로 활용할 수 있으며, 대규모 데이터셋과 복잡한 모델 학습에서 시간과 리소스를 절약할 수 있다. 이를 통해 LightGBM은 대규모 데이터를 다루는 실제 응용 사례에서 경쟁력 있는 성능을 보장하며, 높은 차원의 특징이 포함된 데이터에서도 우수한 성능을 발휘한다.

LightGBM은 다양한 하이퍼파라미터 조정을 통해 모델의 성능을 최적화할 수 있다. 주요 하이퍼파라미터로는 학습률(Learning Rate), 최대 깊이(Max Depth), 리프 노드의 최소 데이터 수(Minimum Data in Leaf) 등이 있으며, 적절한 설정은 과적합(overfitting)을 방지하고 모델의 일반화 능력을 높일 수 있다. 다만, 리프 중심의 성장 방식은 과적합을 유발할 수 있으므로 신중한 하이퍼파라미터 조정이 필요하다.

3.1.6 CatBoost

CatBoost는 Yandex에서 개발한 고성능 그라디언트 부스팅 라이브러리로, 특히 범주형 데이터(categorical data)를 효과적으로 처리할 수 있도록 설계된 트리 기반 앙상블 학습 알고리즘이다[27]. CatBoost는 범주형 변수를 자동으로 인코딩하고, 기존 그라디언트 부스팅 알고리즘에서 발생할 수 있는 과적합 문제와 데이터 편향 문제를 완화하기 위해 여러 혁신적인 기법을 도입하였다. 이를 통해 다양한 예측 및 분류 문제에서 탁월한 성능을 보인다.

CatBoost는 범주형 데이터 처리를 위해 순서적 인코딩(Order Encoding) 방식을 사용한다. 기존의 원-핫 인코딩(One-Hot Encoding)이나 레이블 인코딩(Label Encoding)과 달리, CatBoost는 데이터의 순서와 모델 학습 중의 의존성을 고려하여 각 범주형 변수를 인코딩한다. 이 방식은 데이터 순서에 의해 발생할 수 있는 편향을 줄이고, 모델의 일반화 성능을 향상시키는 데 기여한다. 이러한 특성은 특히 범주형

데이터가 많거나 복잡한 경우에 유리하다[13].

또한, CatBoost는 그라디언트 부스팅 과정에서 과적합 방지를 위한 특별한 규제 기법을 적용한다. 학습 과정에서 CatBoost는 각 트리의 학습 순서를 무작위로 섞어 과적합을 방지하고, 예측 잔차(residuals)를 보정하여 노이즈를 줄이는 방식으로 작동한다. 이를 통해 모델은 과적합의 가능성을 줄이며, 다양한 데이터셋에 대해 우수한 일반화 성능을 보장한다. CatBoost는 이러한 접근을 통해 비교적 깊은 트리를 학습할 수 있으며, 예측 성능이 유지된다.

CatBoost의 또 다른 주요 특징은 CPU 및 GPU 병렬 처리 지원이다. 병렬 처리를 통해 학습 속도를 크게 향상시킬 수 있으며, 대규모 데이터셋에 대해 빠르고 효율적인 학습이 가능하다. 이러한 성능 향상은 대규모 데이터 환경에서도 CatBoost가 효율적으로 동작할 수 있도록 지원한다. 또한, CatBoost는 다양한 하이퍼파라미터 최적화를 통해 모델 성능을 미세 조정할 수 있다. 주요 하이퍼파라미터로는 학습률(Learning Rate), 트리 깊이(Depth), 리프 노드의 최소 데이터 수(Minimum Data in Leaf) 등이 있으며, 이들 매개변수의 적절한 설정은 예측 정확도와 모델의 일반화 능력을 최적화하는 데 중요한 역할을 한다.

3.1.7 AdaBoost

AdaBoost(AdaBoosting)는 Yoav Freund와 Robert Schapire[13]에 의해 제안된 앙상블 학습 알고리즘으로, 다수의 약한 학습기(보통 결정 트리)를 결합하여 강력한 예측 성능을 가지는 모델을 구축하는 데 중점을 둔다. 이 알고리즘은 각 단계에서 약한 학습기를 순차적으로 추가하고, 잘못 예측된 데이터에 대해 가중치를 증가시킴으로써 예측 성능을 점진적으로 향상시킨다. AdaBoost는 약한 학습기의 결합을 통해 예측 정확도를 높이며, 분류 및 회귀 문제에 효과적으로 활용될 수 있다.

AdaBoost의 주요 원리는 각 약한 학습기의 예측 성능에 따라 데이터 가중치를 조정하는 방식으로, 잘못 예측된 데이터에 더 큰 가중치를 부여하여 이후 학습 단계에서 이를 보완한다. 초기 단계에서는 모든 데이터에 동일한 가중치가 부여되며, 약한 학습기를 훈련시킨 후 잘못 예측된 샘플의 가중치를 증가시켜 다음 단계에서 더욱 집중적으로 학습할 수 있도록 한다. 이러한 과정을 반복하여 최종적으로 모든 약한 학습기의 예측을 가중 합산하여 최종 결과를 산출한다[28].

AdaBoost는 특정 약한 학습기의 성능이 낮더라도 전체 모델의 성능을 향상시킬 수 있도록 각 학습기의 가중치를 조정하는 점에서 뛰어난 적응성을 보인다. 이 과정은

모델이 예측 오류를 최소화하는 방향으로 학습을 진행하도록 유도하며, 기존의 부스팅 기법과 비교했을 때 높은 예측 성능을 제공할 수 있다.

AdaBoost는 데이터 가중치를 조정하여 잘못 예측된 데이터에 집중적으로 학습하는 특성을 통해 예측 성능을 지속적으로 개선할 수 있다. 비교적 단순한 구조로 다양한 약한 학습기를 결합하여 유연하게 적용할 수 있으며, 과적합을 방지하면서도 높은 예측 성능을 유지한다. 다만, 데이터에 노이즈가 많을 경우 노이즈에 민감하게 반응할 수 있는 한계를 가진다[29].

그러나, AdaBoost는 잘못된 데이터나 노이즈가 많은 데이터에 대해 민감할 수 있으며, 이는 잘못된 예측에 대한 가중치 증가로 인해 과적합(overfitting)을 초래할 가능성이 있다. 또한, 반복적인 학습 과정에서 연산 비용이 증가할 수 있어 대규모 데이터셋에 대해 높은 연산 비용이 요구될 수 있다.

결론적으로, AdaBoost는 약한 학습기를 결합하여 강력한 예측 성능을 제공하는 부스팅 기법으로, 각 단계에서 예측 오류를 보완하고 가중치를 조정하는 방식으로 작동한다. 데이터의 가중치를 조정하여 점진적으로 성능을 개선하는 이 접근 방식은 다양한 예측 문제에서 높은 성능을 발휘하는 데 중요한 역할을 한다.

3.2 하이퍼파라미터 최적화 기법

하이퍼파라미터 최적화는 머신러닝 모델의 성능을 극대화하기 위해 반드시 필요한 과정이다. 하이퍼파라미터는 모델이 데이터를 학습하는 방식을 결정짓는 중요한 요소로, 적절한 설정이 이루어지지 않으면 모델의 성능이 저하될 수 있다[30]. 트리 기반 앙상블 모델과 같은 복잡한 모델의 경우, 하이퍼파라미터 최적화는 모델의 일반화 성능을 향상시키고 예측 오류를 줄이는 데 중요한 역할을 한다.

가장 기본적인 최적화 방법 중 하나로 그리드 서치(Grid Search)가 있다. 그리드 서치는 미리 정의된 하이퍼파라미터 값의 조합을 체계적으로 탐색하여 최적의 설정을 찾는 방식이다. 모든 조합을 시도하므로 탐색의 완전성이 보장되지만, 연산 비용이 매우 크다는 단점이 있다[31]. 이러한 점에서 그리드 서치는 작은 탐색 공간에서 주로 사용되며, 대규모 하이퍼파라미터 탐색에는 적합하지 않을 수 있다.

이에 비해 랜덤 서치(Random Search)는 설정된 하이퍼파라미터 공간에서 일부 조합을 무작위로 선택해 탐색하는 방식이다[30]. 그리드 서치와 비교하여 연산 효율성이 뛰어나며, 전체 탐색 공간에서 고르게 샘플링하기 때문에 높은 차원의 탐색 공간에서 유리하다. 그러나 최적의 설정을 반드시 찾는 것은 아니므로 일정한 한계가 존재한다.

최근 들어 베이지안 최적화(Bayesian Optimization)와 같은 보다 정교한 최적화 기법이 주목받고 있다. 베이지안 최적화는 이전 평가 결과를 바탕으로 후속 탐색 위치를 결정하는 확률 모델을 생성하여, 효율적인 탐색을 가능하게 한다. 이 과정에서 가우시안 프로세스가 자주 사용되며, 최적의 하이퍼파라미터를 찾기 위한 탐색 공간을 점진적으로 좁혀나가는 방식으로 연산 비용을 줄이면서도 높은 성능을 기대할 수 있다[32].

또한, 유전 알고리즘(Genetic Algorithm)은 생물학적 진화 원리를 기반으로 한 최적화 기법으로, 하이퍼파라미터 설정을 최적화하기 위해 개체군을 생성하고 교배 및 돌연변이 연산을 통해 최적의 조합을 찾는다. 이 방법은 탐색 공간을 폭넓게 탐색할 수 있으며, 탐색 공간 내에서 다양한 조합을 고려해 전역 최적화에 유리한 특성을 지닌다[33].

이와 같은 최적화 기법은 각각의 장단점이 있으며, 특정 문제에 따라 적절한 방법을 선택하는 것이 중요하다. 하이퍼파라미터 최적화는 트리 기반 앙상블 모델의 성능을 극대화하기 위해 필수적인 과정으로, 연구자들은 모델 성능 향상을 위해 다양한 최적화 기법을 결합하거나 실험을 통해 최적의 설정을 찾는 노력을 지속하고 있다[34].

3.2.1 그리드서치

그리드 서치(Grid Search)는 모델의 성능을 최적화하기 위해 미리 정의된 하이퍼파라미터 공간에서 가능한 모든 조합을 체계적으로 탐색하여 최적의 하이퍼파라미터 값을 찾는 기법이다. 하이퍼파라미터는 모델 학습에 중요한 영향을 미치는 요소로, 적절하게 설정되지 않으면 모델의 성능이 저하될 수 있다. 따라서 하이퍼파라미터 최적화는 모델의 예측 성능을 극대화하고 일반화 능력을 높이기 위해 필수적인 과정이다[35].

그리드 서치는 사용자가 정의한 여러 하이퍼파라미터의 가능한 값을 조합하여 모든 경우의 수를 탐색하는 방식으로 동작한다. 예를 들어, 결정 트리 모델의 경우, 최대 깊이(max depth)와 분할에 필요한 최소 샘플 수(minimum samples split)를 조정할 때, 각각의 하이퍼파라미터에 대해 다양한 값을 설정한 후, 모든 조합을 실험적으로 적용하여 최적의 성능을 달성할 수 있는 조합을 찾는다. 이러한 방식은 비교적 간단한 개념을 기반으로 하지만, 포괄적인 탐색을 통해 모델 성능을 개선할 수 있는 가능성을 제공한다[30].

그리드 서치는 모든 가능한 하이퍼파라미터 조합을 평가하기 때문에 탐색의 완전성이 보장된다. 이는 최적의 하이퍼파라미터 조합을 놓치지 않고 발견할 수 있도록 돕

는다. 또한, 그리드 서치는 특정 하이퍼파라미터가 모델 성능에 미치는 영향을 명확히 분석할 수 있어, 사용자가 모델 설정에 대한 직관적 이해를 높이는 데 유용하다.

그러나, 그리드 서치는 탐색 공간이 클수록 연산 비용과 시간이 급격히 증가할 수 있다는 한계를 가진다. 특히, 많은 하이퍼파라미터가 존재하거나 각 하이퍼파라미터의 값이 넓은 범위에 걸쳐 있을 경우, 가능한 조합의 수가 기하급수적으로 증가하여 연산 속도가 느려질 수 있다[36]. 이러한 문제는 고차원 데이터와 복잡한 모델에 대해 그리드 서치의 효율성을 떨어뜨릴 수 있으며, 제한된 연산 자원을 가진 환경에서는 현실적으로 모든 조합을 탐색하는 것이 어렵다.

그리드 서치는 상대적으로 작은 탐색 공간에서 최적의 하이퍼파라미터를 찾는 데 유용하며, 다양한 머신러닝 모델에서 널리 활용된다. 예를 들어, 지원 벡터 머신(SVM), 결정 트리, 랜덤 포레스트 등 다양한 모델에서 최적의 하이퍼파라미터 조합을 찾기 위해 사용된다. 그러나 대규모 데이터셋이나 복잡한 모델에서는 랜덤 서치(Random Search)나 베이지안 최적화(Bayesian Optimization)와 같은 다른 최적화 기법과 함께 활용되기도 한다.

3.2.2 랜덤서치

랜덤 서치(Random Search)는 하이퍼파라미터 최적화를 위해 미리 정의된 하이퍼파라미터 공간에서 무작위로 하이퍼파라미터 조합을 선택하여 성능을 평가하는 방식의 탐색 기법이다. 기존의 그리드 서치(Grid Search)가 모든 가능한 조합을 체계적으로 탐색하는 것과 달리, 랜덤 서치는 무작위로 조합을 선택함으로써 탐색 속도를 개선하고 연산 효율성을 높인다. 이 방법은 Bergstra와 Bengio[31]에 의해 하이퍼파라미터 최적화에서의 효율성이 입증되었으며, 특히 고차원 공간에서 유용하다.

랜덤 서치의 작동 방식은 탐색 공간에서 하이퍼파라미터 값을 무작위로 선택하고 각 조합에 대해 모델 성능을 평가하는 것이다. 이를 통해 전체 탐색 공간의 일부만을 탐색함에도 불구하고, 중요한 하이퍼파라미터 조합을 발견할 가능성을 제공한다. 예를 들어, 특정 하이퍼파라미터가 모델 성능에 중요한 영향을 미칠 경우, 무작위 탐색은 다양한 영역을 커버할 수 있으므로 효율적으로 최적의 조합에 근접할 수 있다[36].

랜덤 서치는 고차원 하이퍼파라미터 공간에서 효율적인 탐색이 가능하다는 점이 주요 장점이다. 그리드 서치와 달리, 탐색 공간이 커질수록 랜덤 서치의 효율성이 더 두드러진다. 탐색 시간이 제한된 경우에도 일정한 탐색 결과를 제공할 수 있어 실용적이다. 또한, 무작위로 선택된 조합을 평가하기 때문에 특정 하이퍼파라미터에 치우치지 않는 전반적인 탐색을 기대할 수 있다.

그러나 랜덤 서치는 탐색 공간의 일부만을 평가하므로 최적의 조합을 발견하지 못

할 가능성이 존재한다. 특히 탐색 공간이 매우 크고 중요한 하이퍼파라미터가 특정 영역에 집중된 경우, 무작위 선택 방식의 한계가 나타날 수 있다. 이러한 단점을 보완하기 위해 랜덤 서치를 다른 최적화 기법과 병행하여 활용할 수도 있다.

결론적으로, 랜덤 서치는 무작위 선택을 통해 하이퍼파라미터를 탐색하는 방식으로, 연산 자원과 시간의 제한이 있는 상황에서도 효과적인 최적화를 가능하게 한다. 전반적인 탐색 효율성을 높이면서도 특정 조합을 놓칠 가능성을 고려하여 활용하면 강력한 하이퍼파라미터 최적화 도구로 작동할 수 있다.

3.2.3 베이지안 최적화

베이지안 최적화(Bayesian Optimization)는 하이퍼파라미터 최적화를 효율적으로 수행하기 위한 방법으로, 비용이 많이 드는 목표 함수의 평가 횟수를 최소화하는 데 중점을 둔다. 이 기법은 하이퍼파라미터 공간을 탐색할 때, 이전에 평가된 결과를 바탕으로 다음 평가할 지점을 결정하는 방식으로 작동하며, 매번 새롭게 평가된 정보를 바탕으로 모델을 업데이트한다[32]. 이를 통해 적은 수의 평가로도 최적의 하이퍼파라미터 조합을 효과적으로 찾아낼 수 있다.

베이지안 최적화는 초기에 몇 가지 하이퍼파라미터 조합을 무작위로 선택하고, 이를 평가하여 초기 확률 모델을 구성한다. 이후 획득 함수를 통해 가장 유망한 새로운 지점을 탐색하고, 평가 결과를 기존 모델에 반영하여 업데이트한다. 이러한 과정이 반복되면서, 최적의 하이퍼파라미터 조합을 점진적으로 찾아낸다.

베이지안 최적화는 기존의 그리드 서치나 랜덤 서치보다 효율적인 탐색을 제공하며, 연산 비용이 높은 목표 함수에 적합하다. 탐색 공간 내에서 중요한 영역을 집중적으로 탐구하면서도 미지의 영역을 균형 있게 탐색하여, 적은 수의 평가로 최적의 결과를 얻을 수 있다[37]. 이로 인해 하이퍼파라미터 최적화 문제에서 높은 성능을 보이며, 대규모 데이터셋이나 복잡한 모델에서도 유용하다.

하지만, 베이지안 최적화는 상대적으로 작은 하이퍼파라미터 공간에서 효과적이며, 매우 고차원 공간에서는 모델링이 복잡해질 수 있다. 초기 평가가 부족할 경우, 확률 모델이 부정확해질 수 있어 최적화 성능에 영향을 미칠 수 있다. 또한, 복잡한 모델에서 가우시안 프로세스 기반의 최적화가 연산 비용을 증가시킬 수 있다는 점도 고려해야 한다.

베이지안 최적화는 탐색과 활용을 균형 있게 유지하여 적은 평가로도 최적의 하이퍼파라미터 조합을 찾아낼 수 있는 강력한 최적화 기법이다. 이는 다양한 예측 및 최적화 문제에서 실질적인 성능 향상을 가능하게 한다.

3.2.4 유전 알고리즘

유전 알고리즘(Genetic Algorithm, GA)은 생물학적 진화 원리에서 영감을 얻어 최적화 문제를 해결하는 탐색 알고리즘으로, 복잡한 최적화 문제에서 전역 최적해를 찾는 데 효과적인 도구로 사용된다. 유전 알고리즘은 후보 해(해결책)의 집합인 개체군(population)을 유지하고, 이를 반복적으로 진화시키면서 최적의 해를 찾는다. 이 과정에서 선택(selection), 교차(crossover), 돌연변이(mutation) 등의 연산을 통해 새로운 세대를 생성하며, 점진적으로 더 나은 해를 탐색한다[32].

유전 알고리즘의 작동 원리는 초기 단계에서 무작위로 생성된 개체군을 기반으로 한다. 각 개체는 문제의 해를 나타내며, 적합도 함수(fitness function)를 통해 해당 개체의 성능을 평가한다. 적합도가 높은 개체일수록 다음 세대에서 더 큰 비중을 차지하게 되며, 선택 연산을 통해 우수한 개체들이 교배에 참여할 기회를 얻게 된다. 교차 연산은 부모 개체의 유전자를 결합하여 새로운 자손을 생성하는 과정으로, 다양한 해를 탐색하는 데 기여한다. 또한, 돌연변이 연산을 통해 일부 개체의 유전자를 무작위로 변경하여 탐색 공간에서 새로운 해를 탐구하는 능력을 제공한다[38].

유전 알고리즘은 탐색 공간이 매우 크거나 비선형성이 강한 문제에서도 전역 최적해를 찾을 가능성이 높다. 초기 해의 품질이나 탐색 공간의 형태에 크게 의존하지 않고 다양한 해를 탐색할 수 있어, 지역 최적해에 빠지기 쉬운 전통적인 탐색 알고리즘에 비해 유연성을 제공한다. 또한, 병렬 처리가 가능하여 대규모 최적화 문제에서도 효율적인 수행이 가능하다.

그러나 유전 알고리즘은 최적화 과정에서 계산 비용이 증가할 수 있으며, 적합도 함수의 평가가 연산적으로 복잡하거나 많은 자원을 필요로 하는 경우 적용이 어렵다. 또한, 수렴 속도가 느려질 수 있고, 지나치게 탐색 공간을 넓게 탐구할 경우 목표와 무관한 해를 생성할 가능성도 존재한다[39]. 이러한 한계를 극복하기 위해 다양한 하이브리드 기법이 연구되고 있으며, 문제의 특성에 따라 유전 알고리즘과 다른 최적화 기법을 병행하여 사용하기도 한다.

제 4 장 하이퍼파라미터 최적화 기법 비교 실험

4.1 데이터셋 설명

4.1.1 사용된 나사 데이터셋의 특징

본 연구에서는 소프트웨어 결함 예측을 위한 대표적인 데이터셋으로 NASA의 MDP(Metrics Data Program)에서 제공하는 결함 데이터셋을 활용하였다[40]. NASA MDP 데이터셋은 다양한 소프트웨어 프로젝트에서 수집된 결함 정보와 소스 코드 메트릭을 포함하고 있으며, 소프트웨어 품질 예측 및 결함 탐지 연구에서 널리 사용된다[1]. 실세계 소프트웨어 프로젝트에서 수집된 데이터를 기반으로 하여, 다양한 예측 문제를 해결하는 데 있어 현실적인 평가 지표를 제공한다.

NASA 데이터셋의 주요 특징은 다음과 같다. 첫째, 이 데이터셋은 코드 복잡도, 코드 길이, 결합도 등과 같은 다양한 소프트웨어 메트릭을 포함하고 있다. 주요 메트릭으로는 소스 코드 라인 수(LOC), 사이클로매틱 복잡도(Cyclomatic Complexity), 클래스 수 등이 있으며, 이러한 메트릭은 소프트웨어 결함 발생 가능성을 예측하는 데 중요한 정보로 사용된다. 둘째, 데이터셋은 각 소프트웨어 모듈이 결함을 포함하고 있는지 여부에 대한 이진 레이블을 제공한다. 결함이 있는 모듈과 결함이 없는 모듈로 구분된 이 레이블은 결함 예측 모델을 학습시키고 성능을 평가하는 데 있어 필수적인 요소이다[41]. 이 데이터셋에서 제공하는 코드 복잡도와 결합도, 라인 수 등의 특성은 결함 탐지의 중요한 지표로 활용된다. 본 연구는 이러한 메트릭을 바탕으로 트리 기반 앙상블 모델이 결함 예측에서 가지는 성능적 장점을 강조하고, 최적화 기법을 통해 예측 정확도를 향상시키는 방법론을 제시한다.

NASA 데이터셋은 다양한 프로젝트와 도메인에서 수집된 데이터로 구성되어 있어, 결함 예측 문제에 대한 모델의 일반화 가능성을 평가할 수 있는 중요한 기회를 제공한다. 예를 들어, PROMISE 리포지토리에서 제공하는 NASA 프로젝트 데이터셋은 각각의 프로젝트가 서로 다른 크기와 특성을 가지며, 다양한 소프트웨어 결함 탐지 문제를 다루는 데 적합하다[42]. 이러한 특성은 모델이 다양한 환경에서 일관된 성능을 보이는지 확인하는 데 기여한다.

Dataset	No. of Ind. variables	Defective	Non-defective	Total
CM1	37	42	285	327
JM1	21	1672	6110	7782
KC1	21	314	869	1183
KC3	39	36	158	194
MC1	37	38	1942	1988
MC2	38	44	81	125
MW1	37	27	226	253
PC1	37	61	644	705
PC2	36	16	729	745
PC3	37	134	943	1077
PC4	37	177	1110	1287

표 1 소프트웨어 결함 데이터 셋 설명

한편, NASA 데이터셋은 데이터의 불균형성 문제를 특징으로 한다. 결함이 있는 모듈에 비해 결함이 없는 모듈이 더 많이 포함되어 있어, 모델이 특정 클래스에 편향될 가능성이 존재한다. 이러한 불균형성은 모델의 성능 저하를 초래할 수 있으므로, 데이터 전처리 과정에서 SMOTE(Synthetic Minority Over-sampling Technique)와 같은 불균형 데이터 처리 기법을 적용할 필요가 있다. 또한, 실제 소프트웨어 프로젝트에서 수집된 데이터이므로 노이즈와 결측치가 포함될 수 있으며, 이로 인해 모델의 예측 성능이 영향을 받을 수 있다. 노이즈 제거 및 결측치 대체와 같은 정제 작업은 보다 신뢰할 수 있는 모델 성능 평가를 위해 중요하다.

표 1을 통해 각 데이터셋의 규모와 결함이 있는 데이터와 없는 데이터 간의 비율을 확인할 수 있다. 예를 들어, CM1 데이터셋에는 총 327개의 인스턴스가 있으며, 그중 42개가 결함이 있고 285개는 결함이 없는 데이터로 구성되어 있다. JM1 데이터셋은 7782개의 인스턴스를 포함하고 있으며, 이 중 1672개는 결함이 있고, 나머지 6110개는 결함이 없는 데이터이다.

4.1.2 데이터 전처리 과정

NASA 소프트웨어 결함 예측 데이터셋은 결함이 있는 소프트웨어 모듈과 결함이 없는 모듈 간에 불균형한 분포를 보이는 경향이 있다. 이러한 데이터 불균형은 학습된 모델이 소수 클래스인 결함 모듈을 적절히 예측하지 못하는 문제를 유발할 수 있기 때문에, 따라서 본 연구에서는 데이터 균형을 맞추기 위해 SMOTE(Synthetic Minority Over-sampling Technique) 기법을 적용하였다.

본 연구에서는 NASA의 11개의 데이터셋을 대상으로 하여 데이터 전처리 과정에서

불균형 문제를 완화하기 위해 SMOTE를 적용하였다. 해당 데이터셋은 결함이 있는 모듈과 없는 모듈 간의 데이터 불균형이 존재하며, 이러한 불균형은 모델의 학습 및 예측 성능에 영향을 미칠 수 있다. 따라서, 데이터의 균형을 맞추기 위한 전처리 과정을 다음과 같이 수행하였다.

먼저, 데이터셋을 특성(feature)과 레이블(label)로 분리하고, 이를 학습 데이터와 테스트 데이터로 나누었다. 학습 데이터는 모델의 학습을 위한 주요 데이터로 사용되며, 테스트 데이터는 모델 성능 평가를 위해 별도로 보관되었다.

이후, 학습 데이터에 대해 SMOTE 기법을 적용하였다. SMOTE는 소수 클래스에 대한 데이터를 증강하여 데이터의 불균형을 완화하는 방법으로, imblearn 라이브러리의 SMOTE 클래스를 사용하여 구현하였다. 이 과정에서 SMOTE의 k_neighbors 파라미터는 5로 설정되었으며, 무작위 시드(random seed)는 42로 고정하여 실험의 재현성을 보장하였다. k_neighbors는 새로운 샘플을 생성할 때 가장 가까운 이웃으로 고려할 데이터 수를 의미하며, 설정된 값은 소수 클래스의 증강에 영향을 미친다.

SMOTE 적용 결과, 학습 데이터 내 소수 클래스(결함 모듈)의 수가 증가하였고, 데이터셋의 불균형이 완화되었다. 이는 모델이 학습 과정에서 소수 클래스를 보다 균등하게 학습할 수 있도록 하여, 예측 성능의 향상에 기여하였다. SMOTE 적용 전과 후의 데이터 분포 차이를 확인한 결과, 소수 클래스에 대한 정확도 및 F1-score와 같은 성능 지표가 개선되었음을 확인할 수 있었다. 이와 같은 전처리 과정을 통해 데이터 불균형 문제를 해소함으로써, 보다 일반화된 모델을 학습하는 데 기여하였다.

4.2 실험 설계 및 평가 방법

4.2.1 실험 설계 방식

본 연구에서는 소프트웨어 결함 예측을 위해 트리 기반 앙상블 모델(Random Forest, Extra Trees, Gradient Boosting Machine (GBM), XGBoost, LightGBM, CatBoost, AdaBoost)의 하이퍼파라미터 최적화 기법(그리드서치, 랜덤서치, 베이지안 최적화, 유전 알고리즘)을 비교하고 최적의 모델을 찾기 위한 실험을 설계하였다. 실험 설계 방식은 다음과 같은 절차로 진행되었다.

1. 데이터셋 분할 (Data Splitting)

본 연구에서는 소프트웨어 결함 예측 모델의 학습과 성능 평가를 위해 각 데이터셋을 훈련 데이터(Training Data)와 테스트 데이터(Test Data)로 분할하였다. 이 분할은

모델의 일반화 성능을 독립적으로 평가하고, 과적합(overfitting)을 방지하기 위한 필수적인 단계로 수행되었다.

2.1 훈련 데이터와 테스트 데이터의 비율 설정

전체 데이터셋의 80%를 훈련 데이터로 할당하였으며, 나머지 20%는 테스트 데이터로 할당하였다. 훈련 데이터는 모델 학습과 하이퍼파라미터 최적화에 사용되었으며, 테스트 데이터는 모델의 최종 성능 평가에 사용되었다. 이러한 비율 설정은 소프트웨어 결함 예측에서 모델의 일반화 능력을 충분히 평가하기 위한 표준적인 분할 방식에 기반한다.

2.2 무작위 데이터 분할과 난수 시드 고정

데이터 분할은 무작위로 수행되었으며, 이를 통해 특정 순서나 패턴에 의한 데이터 편향을 방지하였다. 또한, 실험의 재현성을 확보하기 위해 난수 시드(random seed)를 고정하였다. 본 연구에서는 `random_state=42`를 사용하여 분할의 일관성을 보장하였다. 이를 통해 동일한 데이터셋에 대해 반복 실험을 수행할 경우, 훈련 데이터와 테스트 데이터의 분할 결과가 동일하게 유지되도록 하였다.

2.3 데이터 분할의 중요성

모델의 일반화 능력을 평가하고 훈련 데이터에 대한 과적합을 방지하기 위해 독립적인 테스트 데이터에서 성능을 평가하는 과정이 필수적이다. 본 연구에서는 각 최적화 기법을 사용하여 최적의 하이퍼파라미터 조합을 탐색한 후, 해당 최적화된 모델을 테스트 데이터에서 평가하여 최적화 성능을 검증하였다. 이 과정을 통해 최적화된 모델이 새로운 데이터에 대해 얼마나 잘 일반화되는지를 확인할 수 있었다.

모델 최적화와 검증 단계에서, 훈련 데이터는 모델 학습과 최적화를 위한 주된 데이터로 활용되었으며, 다양한 하이퍼파라미터 최적화 기법을 적용하여 모델의 예측 성능을 극대화하고자 하였다. 테스트 데이터는 학습 및 최적화 과정에 사용되지 않으며, 최적화된 모델이 새로운 데이터에서 예측 성능을 유지하는지를 평가하기 위해 사용되었다. 이를 통해, 각 최적화 기법의 성능이 실제 응용 환경에서 모델의 일반화 능력을 개선하는 데 얼마나 효과적인지를 평가하고 비교하였다.

2.4 데이터 분할의 기술적 구현 예시

데이터 분할은 Python의 `scikit-learn` 라이브러리의 `train_test_split` 함수를 사용하여 수행되었다. 예시는 다음과 같다.

```python
from sklearn.model_selection import train_test_split

# 특성과 라벨 분리
X = data.drop('Defective', axis=1)
y = data['Defective']

# 훈련 데이터와 테스트 데이터 분할
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rand
om_state=42)
```

그림 1 훈련 데이터와 테스트 데이터의 분할의 예시 코드

2.5 데이터 분할 과정에서의 추가 고려사항

소프트웨어 결함 예측 데이터셋은 결함 클래스와 비결함 클래스 간에 심각한 불균형이 존재할 수 있으며, 이러한 불균형은 모델의 학습 및 평가 과정에 부정적인 영향을 미칠 수 있다. 이를 해결하기 위해, 본 연구에서는 훈련 데이터와 테스트 데이터에서 두 클래스가 균형 있게 포함되도록 고려하였다. 이로써 모델이 특정 클래스에 치우치지 않고, 양쪽 클래스 모두에 대해 일관된 성능을 보일 수 있도록 하였다.

또한, 데이터 분할로 인한 성능 변동성을 줄이고 보다 신뢰성 있는 성능 평가를 수행하기 위해 교차 검증(cross-validation)을 병행하여 다수의 데이터 분할 실험을 수행하였다. 이를 통해 각 최적화 기법과 모델의 일반화 성능을 보다 정확하게 평가하였으며, 다양한 데이터 분할 환경에서 모델의 성능을 검증할 수 있었다. 이러한 접근은 최적화된 모델이 실제 응용 환경에서도 안정적이고 신뢰성 있는 예측 성능을 제공할 수 있도록 보장한다.

3. 트리 기반 앙상블 모델 설정 (Tree-Based Ensemble Model Configuration)

본 연구에서는 다양한 트리 기반 앙상블 모델을 사용하여 소프트웨어 결함 예측 모델을 개발하였다. 각 모델은 기본적으로 제공되는 하이퍼파라미터로 초기화되었으며, 이후 최적화 기법을 통해 모델 성능을 개선하였다. 본 연구에서 사용된 트리 기반 앙

상블 모델은 다음과 같다: 랜덤 포레스트(Random Forest), 엑스트라 트리(Extra Trees), 그라디언트 부스팅 머신(Gradient Boosting Machine, GBM), XGBoost, LightGBM, CatBoost, 그리고 AdaBoost이다.

3.1 모델 초기화

각 모델은 `scikit-learn`, `XGBoost`, `LightGBM`, `CatBoost` 등 관련 라이브러리의 기본 설정을 사용하여 초기화되었다. 초기화 시 각 모델은 최적화 기법 적용 전 기본 하이퍼파라미터 설정을 따르며, 이는 특정 기법의 영향력을 독립적으로 평가하고 비교하기 위한 기준으로 작용하였다. 예를 들어, 랜덤 포레스트 모델은 기본적으로 100개의 결정 트리(n_estimators=100)와 최대 깊이 제한이 없는 트리(max_depth=None)로 초기화되었으며, 이후 최적화 과정을 통해 이 값들이 조정되었다.

본 연구에서는 각 트리 기반 앙상블 모델에 대해 초기 기본 하이퍼파라미터 설정을 다음과 같이 구성하였다.

모델	하이퍼파라미터	기본 값
Random Forest	n_estimators	100
	max_depth	None
	min_samples_split	2
	min_samples_leaf	1
Extra Trees	n_estimators	100

	max_depth	None
	min_samples_split	2
	min_samples_leaf	1
Gradient Boosting (GBM)	learning_rate	0.1
	n_estimators	100
	max_depth	3
XGBoost	learning_rate	0.3
	n_estimators	100
	max_depth	6
LightGBM	learning_rate	0.1
	num_leaves	31
	n_estimators	100
CatBoost	iterations	1000
	learning_rate	0.03
	depth	6
AdaBoost	n_estimators	50

표 2 트리 기반 앙상블 모델에 대한 기본 하이퍼파라미터 설정

이와 같은 기본 설정은 각 모델의 초기 성능 평가를 위한 기준점으로 활용되었으며, 이후 하이퍼파라미터 최적화 기법을 통해 최적의 조합을 탐색하였다.

기본 설정으로 초기화된 모델은 이후 하이퍼파라미터 최적화 기법(그리드 서치, 랜덤 서치, 베이지안 최적화, 유전 알고리즘)을 적용하여 성능을 향상시켰다. 모델 초기화와 하이퍼파라미터 최적화 과정은 중요한 의미를 가진다.

우선, 모든 모델이 동일한 초기 상태에서 시작함으로써 최적화 기법의 성능을 공정하게 비교할 수 있는 기준을 제공한다. 이를 통해 각 최적화 기법이 모델 성능을 얼마나 효과적으로 개선하는지를 평가할 수 있다. 또한, 기본 설정에서의 모델 성능을 기준으로 최적화 기법이 성능을 얼마나 향상시킬 수 있는지를 확인할 수 있어, 최적화 기법의 개선 효과를 정량적으로 분석하는 데 기여한다.

최적화 과정에서는 특정 하이퍼파라미터가 모델 성능에 미치는 영향을 분석하여 각 모델별로 최적의 하이퍼파라미터 조합을 도출할 수 있다. 이를 통해 모델 성능을 최대한으로 향상시키고, 하이퍼파라미터의 중요도를 평가함으로써 보다 전략적인 모델 튜닝이 가능해진다. 이러한 접근은 모델의 예측 성능을 최적화하고, 실질적인 개선을 이끌어내는 데 핵심적인 역할을 한다.

3.4 모델 설정 과정에서 고려사항

기본 설정된 모델이 과적합(overfitting) 문제를 야기하지 않도록 초기화된 상태에서

규제(regularization) 관련 하이퍼파라미터를 적절히 조정하였다. 구체적으로, 트리 기반 모델의 경우 트리 깊이를 제한하거나 최소 샘플 분할 크기를 조정하여 모델이 훈련 데이터에 과도하게 적합되지 않도록 설정하였다. 이러한 조치는 모델의 일반화 성능을 확보하고, 과적합을 방지하기 위한 필수적인 단계로 수행되었다.

또한, 각 모델은 하이퍼파라미터 최적화 기법을 적용하기에 적합한 상태로 구성되었으며, 모델 초기화 후 훈련 데이터에 대한 초기 학습을 통해 기본 성능을 평가하였다. 이를 통해 각 모델이 최적화 과정에서의 성능 향상을 위해 초기 상태에서의 기준 성능을 설정하고, 이후 최적화된 하이퍼파라미터를 통한 성능 개선을 명확히 확인할 수 있도록 하였다.

3.5 기술적 구현 예시

아래는 랜덤 포레스트 모델의 초기화 예시 코드이다:

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# 하이퍼파라미터 탐색 범위 정의
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Random Forest 모델 초기화
rf_model = RandomForestClassifier(random_state=42)

# Grid Search를 사용한 하이퍼파라미터 최적화
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5,
 scoring='f1')
grid_search.fit(X_train, y_train)
```

그림 2 랜덤 포레스트 모델의 초기화 예시 코드

이후 최적화 과정을 통해 위의 기본 설정값들이 각 최적화 기법에 따라 조정되며, 최적의 성능을 발휘하는 하이퍼파라미터 조합을 탐색하게 된다.

4.2.2 하이퍼파라미터 최적화 기법 적용

본 연구에서는 트리 기반 앙상블 모델(Random Forest, Extra Trees, Gradient Boosting Machine, XGBoost, LightGBM, CatBoost, AdaBoost)에 대해 다양한 하이퍼파라미터 최적화 기법(그리드 서치, 랜덤 서치, 베이지안 최적화, 유전 알고리즘)을 적용하여 최적의 성능을 발휘할 수 있는 하이퍼파라미터 조합을 탐색하였다. 각 최적화 기법은 모델의 성능을 개선하고 일반화(generalization) 성능을 극대화하는 데 초점을 두었다.

또한, 본 연구에서는 성능 평가의 일관성을 유지하고 모델의 일반화 능력을 검증하기 위해 5겹 교차 검증(5-fold cross-validation)을 적용하였다. 교차 검증을 통해 훈련 데이터에서 얻은 평균 성능 지표를 산출함으로써, 모델의 과적합(overfitting)을 방지하고, 모델이 새로운 데이터에서도 유사한 성능을 유지할 수 있는지 평가할 수 있었다.

모델별로 최적화 기법을 적용하기 위해 탐색할 하이퍼파라미터의 범위를 사전에 설정하였다. 각 모델과 최적화에 사용된 주요 하이퍼파라미터와 그 탐색 범위는 다음과 같다.

모델	하이퍼파라미터	범위	설명
Random Forest 및 Extra Trees	n_estimators	[50, 100, 200, 300]	결정 트리 개수
	max_depth	[None, 10, 20, 30, 40]	트리의 최대 깊이
	min_samples_split	[2, 5, 10]	노드를 분할하기 위한 최소 샘플 수
	min_samples_leaf	[1, 2, 4]	리프 노드의 최소 샘플 수
	bootstrap	[True, False]	배깅 방식 사용 여부
Gradient Boosting Machine (GBM)	learning_rate	[0.01, 0.05, 0.1, 0.2]	학습률
	n_estimators	[50, 100, 150, 200]	부스팅 단계 수
	max_depth	[3, 4, 5, 6]	트리의 최대 깊이
	min_samples_split	[2, 5, 10]	노드를 분할하기 위한 최소 샘플 수
	min_samples_leaf	[1, 2, 4]	리프 노드의 최소 샘플 수
XGBoost	learning_rate	[0.01, 0.05, 0.1, 0.2]	학습률
	n_estimators	[50, 100, 150, 200]	부스팅 단계 수
	max_depth	[3, 4, 5, 6, 8]	트리의 최대 깊이
	subsample	[0.6, 0.8, 1.0]	데이터 샘플링 비율
	colsample_bytree	[0.6, 0.8, 1.0]	특성 샘플링 비율
LightGBM	learning_rate	[0.01, 0.05, 0.1, 0.2]	학습률
	num_leaves	[20, 31, 40, 50]	트리의 리프 노드 수
	n_estimators	[50, 100, 150, 200]	부스팅 단계 수
	min_child_samples	[5, 10, 20]	리프 노드의 최소 샘플 수
	subsample	[0.6, 0.8, 1.0]	샘플링 비율
CatBoost	iterations	[100, 200, 300, 400]	부스팅 단계 수
	learning_rate	[0.01, 0.05, 0.1, 0.2]	학습률
	depth	[3, 4, 5, 6, 8]	트리 깊이
AdaBoost	n_estimators	[50, 100, 150, 200]	부스팅 단계 수

표 3 트리 기반 앙상블 모델 최적화를 위한 하이퍼파라미터 범위

본 연구에서는 트리 기반 앙상블 모델(Random Forest, Extra Trees, Gradient Boosting Machine, XGBoost, LightGBM, CatBoost, AdaBoost)의 하이퍼파라미터 최적화를 위해 탐색 범위를 사전에 정의하였다. 이는 최적화 기법(Grid Search, Random Search, Bayesian Optimization, Genetic Algorithm)의 탐색 효율성을 극대화하고, 불필요한 계산 자원의 낭비를 방지하기 위한 전략적 결정이었다.

탐색 범위를 사전 정의하는 것은 다음과 같은 이유에서 필요하다. 첫째, Grid Search와 Random Search와 같은 기법은 탐색 공간을 명시적으로 요구한다. 이 기법들은 탐색 공간 내에서 하이퍼파라미터 조합을 평가하므로, 탐색 범위가 과도하게 넓을 경우 최적화 과정이 연산적으로 비효율적이게 된다. 특히, Grid Search는 가능한 모든 조합을 탐색하므로, 탐색 범위가 클수록 계산 비용이 기하급수적으로 증가할 수

있다. 둘째, Bayesian Optimization은 초기 탐색 공간을 모델링하여 탐색을 점진적으로 최적화하는 방식으로 동작한다. 이 경우 탐색 범위가 너무 좁으면 전역 최적해(Global Optimum)를 찾지 못할 가능성이 존재하며, 반대로 탐색 범위가 너무 넓으면 초기 탐색 과정이 비효율적으로 진행될 수 있다. 따라서, Bayesian Optimization에서는 탐색 범위의 설정이 초기 모델링의 품질과 탐색 효율성을 결정짓는 중요한 요인으로 작용한다. 셋째, Genetic Algorithm은 진화적 연산을 기반으로 탐색 공간을 점진적으로 최적화하므로 초기 탐색 범위에 상대적으로 덜 민감하지만, 지나치게 큰 탐색 공간에서는 수렴 속도가 저하될 가능성이 있다.

탐색 범위는 각 기법의 설계 및 모델의 특성을 고려하여 다음과 같이 설정되었다. 예를 들어, 학습률(learning rate)의 경우 [0.01, 0.2]의 범위를 사용하여 과도한 학습 속도나 지나치게 낮은 학습 속도로 인해 모델 성능이 저하되는 경우를 방지하였다. 또한, n_estimators와 같은 하이퍼파라미터는 [50, 300]의 범위로 제한하여 과적합(overfitting) 및 학습 시간의 증가를 최소화하였다. 이러한 설정은 기존 연구에서 제시된 트리 기반 앙상블 모델의 최적 하이퍼파라미터 탐색 범위를 참고하여 결정되었으며, 이를 통해 탐색의 실용성을 확보하였다.

다만, 모든 최적화 기법이 탐색 범위를 요구하는 것은 아니며, 탐색 범위 설정을 최소화하거나 동적으로 설정할 수 있는 대안적 방법이 존재한다. 예를 들어, Adaptive Bayesian Optimization은 초기 탐색 공간 설정에 의존하지 않고 탐색 범위를 점진적으로 축소하여 효율성을 개선할 수 있다. 그러나 본 연구에서는 명시적으로 탐색 범위를 설정함으로써, 연구의 재현성을 보장하고 각 최적화 기법의 성능을 균등하게 비교하는 데 중점을 두었다.

탐색 범위를 사전에 정의하는 접근법은 성능 최적화와 계산 효율성을 동시에 달성할 수 있는 유효한 전략이다. 특히, 소프트웨어 결함 예측(SDP) 데이터셋과 같이 고유한 특성을 가지는 문제 영역에서는 탐색 공간의 설정이 모델의 최적화 결과에 중요한 영향을 미친다. 따라서, 탐색 범위의 설정은 본 연구의 주요 설계 요소 중 하나로, 연구 목적에 부합하는 성능 최적화를 달성하는 데 기여하였다.

4.2.3 성능 평가 지표

최적화된 하이퍼파라미터 조합의 성능은 별도의 테스트 데이터셋을 통해 평가되었으며, 평가 지표로는 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 스코어(F1 Score), ROC AUC 등이 사용되었다. 각 기법의 성능을 비교하기 위해 동일한 실험 조건에서 반복 평가를 수행하였으며, 하이퍼파라미터 최적화의 효과를 정량적으로 분석하였다.

본 연구에서는 소프트웨어 결함 예측을 위한 트리 기반 앙상블 모델의 성능을 정확하고 신뢰성 있게 비교하기 위해 다양한 성능 평가 지표를 설정하였다. 각 지표는 모델의 예측 정확성과 전반적인 성능을 정량적으로 평가하는 데 중요한 역할을 하며, 각 모델 및 하이퍼파라미터 최적화 기법 간의 차이를 효과적으로 비교할 수 있도록 한다. 본 연구에서 사용된 주요 성능 평가 지표는 다음과 같다.

1) 정확도 (Accuracy)

정확도는 전체 샘플 중에서 모델이 올바르게 예측한 비율을 나타낸다.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

그림 3 정확도 계산식

여기서 TP는 True Positive(참 긍정), TN은 True Negative(참 부정), FP는 False Positive(거짓 긍정), FN은 False Negative(거짓 부정)을 나타낸다.

정확도는 전반적인 예측 성능을 나타내는 지표로, 불균형 데이터셋에서도 유용할 수 있으나, False Negative와 False Positive가 중요한 경우 다른 지표와 함께 해석하는 것이 중요하다.

2) 정밀도 (Precision)

정밀도는 모델이 양성으로 예측한 사례 중 실제로 양성인 비율을 나타내는 지표로, 양성 예측의 정확성을 평가한다.

$$\text{Precision} = \frac{TP}{TP + FP}$$

그림 4 정밀도 계산식

정밀도는 False Positive의 발생을 줄이는 데 중점을 두며, 양성 예측이 중요한 경우에 활용도가 높다. 예를 들어, 소프트웨어 결함을 식별할 때, 높은 정밀도는 잘못된 경고를 줄이고, 실제 결함이 있는 경우에만 탐지할 수 있음을 의미한다.

3) 재현율 (Recall, Sensitivity 또는 TPR)

재현율은 실제 양성 사례 중에서 모델이 정확히 탐지한 비율을 의미한다.

$$\text{Recall} = \frac{TP}{TP + FN}$$

그림 5 재현율 계산식

재현율은 False Negative의 감소에 중점을 두며, 결함 탐지에서 중요한 지표로 작용한다. 이는 특히 소프트웨어 품질 관리와 같은 분야에서 결함을 놓치지 않고 탐지하는 것이 중요할 때 유용하다.

4) F1 스코어 (F1 Score)

F1 스코어는 정밀도와 재현율의 조화 평균으로, 두 지표 간의 균형을 고려하여 모델의 성능을 평가하는 지표이다.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

그림 6 F1 스코어 계산식

F1 스코어는 정밀도와 재현율이 모두 중요할 때 유용하며, 두 지표 중 하나라도 낮으면 F1 스코어가 낮아진다. 특히, 불균형한 데이터셋에서 모델의 전반적인 예측 성능을 평가하는 데 적합하다.

5) ROC AUC (Receiver Operating Characteristic - Area Under the Curve)

ROC AUC는 다양한 임계값에서 모델의 True Positive Rate(재현율)과 False Positive Rate 간의 관계를 나타내는 ROC 커브 아래의 면적을 의미한다. ROC AUC 값은 0과 1 사이의 값으로 표현되며, 1에 가까울수록 모델이 양성과 음성 간의 구분을 잘 수행함을 의미한다. 이는 모델의 전반적인 판별 성능을 평가하는 데 유용하다.

본 연구에서는 각 최적화된 모델의 성능을 위의 지표들을 기준으로 비교하였다. 성능 지표는 단일 지표만으로 해석하기보다는, 여러 지표를 종합적으로 분석하여 모델의 강점과 약점을 파악하였다.

제 5 장 실험 결과 및 분석

5.1 모델별 최적화 성능 분석

본 연구에서는 트리 기반 앙상블 모델에 대해 4가지 주요 하이퍼파라미터 최적화 기법(그리드 서치, 랜덤 서치, 베이지안 최적화, 유전 알고리즘)을 적용하여 모델별 최적화 성능을 비교 분석하였다. 성능 분석은 CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4의 11개 나사(NASA) 소프트웨어 결함 데이터셋에 대해 수행된 결과의 평균값을 기반으로 이루어졌다. 이는 각 모델과 최적화 기법이 다양한 데이터셋에서 일관된 성능을 보이는지 평가하기 위한 것이다.

우선, 전반적으로 유전 알고리즘은 대부분의 모델에서 높은 성능 향상을 보여주었으며, 이는 진화적 탐색 접근 방식이 복잡한 하이퍼파라미터 공간에서 전역 최적해를 효과적으로 탐색할 수 있음을 보여준다. 그러나 일부 모델에서는 그리드 서치와 랜덤 서치가 상대적으로 일관된 성능을 보이며, 특정 모델에서의 하이퍼파라미터 조합에 대해 안정적인 최적화를 제공하였다. 이는 탐색 기법이 단순히 탐색 속도와 효율성뿐만 아니라 모델의 구조적 특성과 탐색 공간의 복잡성에 따라 다르게 작용할 수 있음을 의미한다.

또한, 각 모델에 대해 최적화 기법이 제공하는 성능 향상은 모델의 복잡도와 하이퍼파라미터 공간의 크기와도 밀접한 관련이 있다. 예를 들어, 모델의 하이퍼파라미터 공간이 넓고 복잡할수록 랜덤 서치와 유전 알고리즘이 보다 유리한 경향을 보이며, 이는 무작위성 기반 탐색 기법이 다차원 탐색에서 전역 최적화로 수렴할 가능성이 높기 때문이다. 반면, 그리드 서치는 상대적으로 작은 탐색 공간에서 효과적이나, 탐색 비용이 높고, 탐색 공간이 커질수록 효율성이 떨어질 수 있다.

특히, 본 연구에서는 각 최적화 기법을 통해 도출된 최적의 하이퍼파라미터 조합이 모델의 성능을 어떻게 변화시키는지 분석하였으며, 성능 지표(정확도, 정밀도, 재현율, F1 스코어, AUC)를 바탕으로 최적화의 효과를 정량적으로 평가하였다. 최적화 기법이 모델 성능에 미치는 영향은 모델 구조와 데이터셋 특성에 따라 달라질 수 있으므로, 이러한 분석은 모델과 최적화 기법 간의 상호작용을 깊이 있게 이해하는 데 기여할 수 있다

모델	최적화 방법	정확도	정밀도	재현율	F1 Score	AUC
Adaboost	Grid Search	0.8904	0.8874	0.9137	0.8902	0.9361
	Random Search	0.8984	0.8842	0.9180	0.8924	0.9448
	Bayesian Search	0.8887	0.8919	0.9075	0.8929	0.9511
	Genetic Algorithm	**0.9117**	**0.9117**	**0.9267**	**0.9134**	**0.9464**
CatBoost	Grid Search	0.9140	0.9049	0.9547	0.9161	0.9694
	Random Search	0.9129	0.9041	0.9542	0.9147	0.9718
	Bayesian Search	0.9177	0.9034	**0.9588**	0.9183	0.9726
	Genetic Algorithm	**0.9339**	**0.9167**	0.9522	**0.9274**	**0.9763**
ExtraTree	Grid Search	**0.9225**	**0.9067**	0.9382	0.9163	0.9685
	Random Search	0.9221	0.9064	**0.939**	**0.9173**	0.9695
	Bayesian Search	0.9194	0.9048	0.9325	0.9156	0.9693
	Genetic Algorithm	0.9196	0.9041	0.9329	0.9158	**0.9702**
Gradient Boosting	Grid Search	0.9249	0.9068	0.9362	0.9194	0.9757
	Random Search	0.9257	0.9086	0.9388	0.921	0.9761
	Bayesian Search	0.9278	0.9069	0.9387	0.9213	0.9759
	Genetic Algorithm	**0.9372**	**0.9089**	**0.9492**	**0.9281**	**0.9813**
LightGBM	Grid Search	**0.9357**	0.9068	0.9529	0.9263	0.9809
	Random Search	0.9293	0.9082	0.9568	0.9295	0.9812
	Bayesian Search	0.9273	0.9051	0.9568	0.9293	**0.9814**
	Genetic Algorithm	0.9239	**0.9176**	**0.9614**	0.9263	0.9762
Random Forest	Grid Search	0.8908	0.8879	0.8824	0.878	0.9319
	Random Search	0.8931	0.8983	0.8911	0.8935	0.9351
	Bayesian Search	0.8858	0.8797	0.8831	0.8758	0.9371
	Genetic Algorithm	**0.9256**	**0.9223**	**0.9196**	**0.9197**	**0.9407**
XGBoost	Grid Search	0.9151	0.8957	0.9349	0.9104	**0.9654**
	Random Search	0.9143	0.8935	0.9318	0.9091	0.9647
	Bayesian Search	0.9095	0.8828	0.9341	0.9067	0.9649
	Genetic Algorithm	**0.9386**	**0.9283**	**0.9466**	**0.9379**	0.9571

표 4 트리 기반 앙상블 모델별 최적화 기법 적용에 따른 성능 비교

Adaboost 모델은 유전 알고리즘(Genetic Algorithm)을 적용했을 때 가장 높은 성능을 나타냈다(F1 Score: 0.9134, AUC: 0.9464). 이는 다른 최적화 기법과 비교했을 때 전반적으로 향상된 성능을 보여주었으며, 특히 정밀도(Precision)와 재현율(Recall)의 균형이 잘 맞춰진다는 점에서 돋보였다. Random Search와 Bayesian Optimization도 비교적 높은 성능을 보였으나, Genetic Algorithm이 더 뛰어난 최적화 성능을 보여주었다.

CatBoost 모델에서는 Genetic Algorithm을 통해 최고 성능이 달성되었다(F1 Score: 0.9274, AUC: 0.9763). 특히 정밀도와 재현율에서 큰 성능 향상을 보였으며, 이는 CatBoost 모델이 Genetic Algorithm을 통해 효과적으로 최적화될 수 있음을 보여준다. Bayesian Search와 Random Search도 우수한 성능을 기록하였으며, CatBoost는 다양한 최적화 기법에 대해 전반적으로 강건한 성능을 유지하는 경향을 보였다.

ExtraTrees 모델에서는 Grid Search와 Random Search를 통해 유사한 성능이 최적화되었다(F1 Score 약 0.916). 이는 두 기법 모두 기본적인 탐색 범위 내에서 유효한 최적화를 제공한다는 것을 의미한다. 반면, Genetic Algorithm과 Bayesian Optimization은 다소 비슷한 성능을 보였으나, Grid Search와 Random Search와 비교해 큰 성능 차이는 나타나지 않았다.

Gradient Boosting 모델은 Genetic Algorithm을 통해 가장 높은 성능을 달성하였다(F1 Score: 0.9281, AUC: 0.9813). 이는 다른 최적화 기법에 비해 전반적으로 성능 향상 폭이 컸으며, 모델의 학습과 일반화 성능을 효과적으로 최적화하는 데 기여했다. Random Search와 Bayesian Optimization도 비교적 우수한 성능을 보였으나, Genetic Algorithm이 더 큰 성능 향상을 이끌어냈다.

LightGBM 모델의 경우, Grid Search와 Random Search가 높은 성능을 보이며(F1 Score 약 0.926), 다양한 최적화 기법에 대해 비교적 일관된 성능을 유지하는 경향을 보였다. 반면, Genetic Algorithm의 성능은 상대적으로 낮게 나타났으며, 이는 LightGBM의 특정 하이퍼파라미터 탐색 공간에서 Genetic Algorithm이 최적화 효과를 발휘하기 어려웠을 가능성을 시사한다. 이러한 결과는 LightGBM 모델이 최적화 기법에 따라 비교적 일관된 성능을 유지할 수 있음을 보여준다.

Random Forest 모델은 Genetic Algorithm을 적용했을 때 성능이 크게 향상되었다(F1 Score: 0.9197). 이는 기존의 Grid Search나 Bayesian Search에 비해 더 나은 최적화를 달성했음을 나타내며, Random Forest 모델은 다양한 하이퍼파라미터를 포함하고 있어 Genetic Algorithm과 같은 진화 기반 기법이 탐색 범위를 효과적으로 다룰 수 있음을 보여준다.

XGBoost 모델에서는 Genetic Algorithm이 최고 성능을 기록하였으며(F1 Score: 0.9379, AUC: 0.9571), 다른 최적화 기법과 비교해 전반적으로 더 큰 성능 향상을 보

였다. 특히, 정밀도와 재현율에서 뛰어난 성능을 나타내었으며, 최적화 기법이 모델의 예측 성능을 극대화하는 데 기여했음을 확인할 수 있다. Bayesian Optimization과 Random Search 또한 우수한 성능을 보였으나, Genetic Algorithm이 최상의 성능을 보여주었다.

이러한 결과를 통해 각 최적화 기법이 모델 성능에 미치는 영향을 비교 분석할 수 있었으며, 특정 모델에 대해 어떤 최적화 기법이 더 효과적인지를 확인할 수 있었다. 많은 경우에 Genetic Algorithm은 전반적으로 가장 큰 성능 향상을 이끌어냈으며, 이는 진화 기반의 최적화 접근법이 모델의 복잡한 하이퍼파라미터 탐색에서 효과적임을 보여준다.

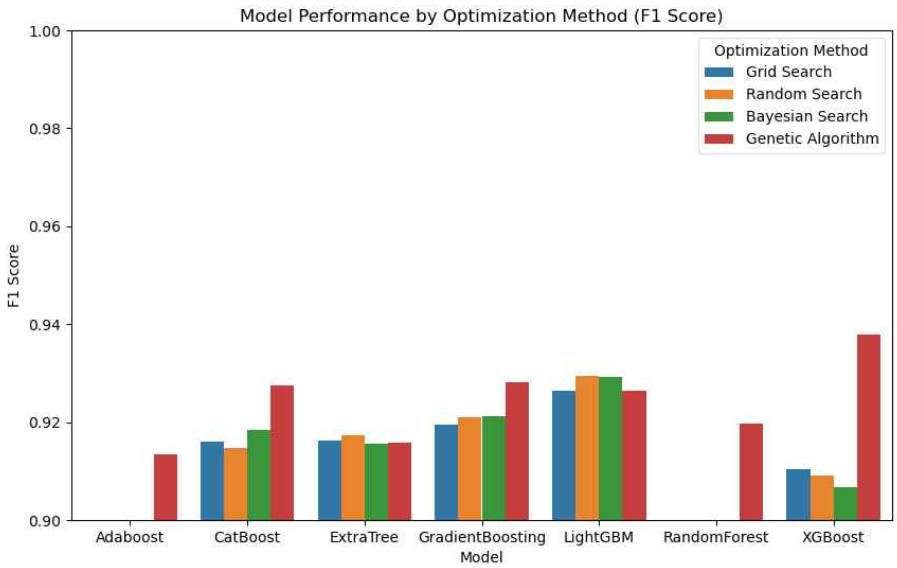

그림 7 최적화 방법별 모델 성능(F1 Score)

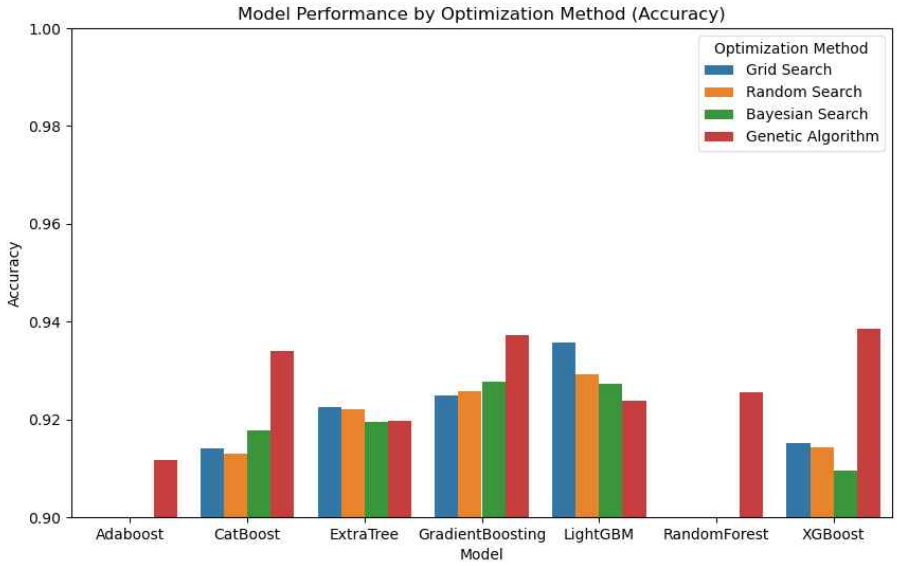

그림 8 최적화 방법별 모델 성능(Accuracy)

위의 그래프에서 Accuracy와 F1 Score 두 성능 지표를 종합적으로 분석한 결과, Genetic Algorithm은 대부분의 모델에서 가장 높은 성능을 기록하였다. 특히 Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서 Genetic Algorithm은 두 지표 모두에서 두드러진 성능 향상을 보여주었다. 반면, Grid Search와 Random Search는 Adaboost, ExtraTree, LightGBM과 같은 비교적 단순한 모델에서 일관된 성능을 나타냈으나, 복잡한 모델에서는 Genetic Algorithm 및 Bayesian Search 대비 낮은 성능을 보였다.

Accuracy는 전반적인 분류 정확도를 평가하는 지표로, 클래스 불균형이 존재하는 경우에는 성능을 과대평가할 가능성이 있다. 반면, F1 Score는 정밀도(Precision)와 재현율(Recall)을 동시에 고려하여 클래스 불균형 문제를 보다 민감하게 반영한다.
예를 들어Adaboost 모델에서 Genetic Algorithm은 Accuracy와 F1 Score 모두에서 가장 높은 성능을 기록하며, 하이퍼파라미터 최적화의 효과를 명확히 입증하였다.
RandomForest와 ExtraTree에서는 Accuracy 지표상 Grid Search와 Random Search가 비슷한 성능을 보였으나, F1 Score에서는 약간의 차이를 나타냈다. 이는 F1 Score가 클래스 불균형 데이터를 처리하는 데 있어 더 적합한 평가 지표임을 보여준다.

Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서는 Genetic

Algorithm이 Accuracy와 F1 Score 두 지표 모두에서 가장 우수한 성능을 기록하였다. 이는 Genetic Algorithm이 전역 최적화를 통해 복잡한 하이퍼파라미터 탐색 공간에서도 우수한 성능을 보일 수 있음을 보여준다. 반면, Bayesian Search는 탐색 효율성과 성능의 균형을 제공하며, Grid Search와 Random Search 대비 우수한 성능을 나타냈다. 그러나 복잡한 모델에서는 Genetic Algorithm만큼의 성능은 기록하지 못했는데, 이는 Bayesian Search가 탐색 효율성을 중시하는 기법으로, 탐색 공간이 매우 크거나 복잡한 경우 Genetic Algorithm보다 한계가 있을 수 있음을 시사한다.

Adaboost, ExtraTree, RandomForest와 같은 단순한 모델에서는 Grid Search와 Random Search도 안정적인 성능을 보였다. 이는 해당 모델들의 하이퍼파라미터 탐색 공간이 비교적 단순하여, 모든 조합을 탐색하는 방식이나 무작위로 탐색하는 방식에서도 유효한 최적화를 달성할 수 있음을 의미한다.

LightGBM 모델에서는 Random Search와 Genetic Algorithm이 Accuracy와 F1 Score에서 모두 우수한 성능을 보였다. 반면, Grid Search는 상대적으로 낮은 성능을 기록했는데, 이는 LightGBM의 빠른 학습 특성이 Grid Search와의 적합성이 낮았음을 암시한다.

Accuracy와 F1 Score를 종합적으로 분석한 결과, Genetic Algorithm은 대부분의 모델에서 가장 우수한 성능을 기록하며, 특히 복잡한 모델에서 최적의 성능을 제공하였다. 두 성능 지표는 모델 성능 평가에서 상호 보완적인 역할을 수행하였으며, 특정 최적화 기법이 모델의 복잡성에 따라 성능 차이를 보일 수 있음을 확인하였다.
Adaboost, ExtraTree, RandomForest와 같은 단순한 모델에서는 Grid Search와 Random Search도 효과적이었으나, Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서는 Genetic Algorithm의 활용이 특히 효과적이었다.
이와 같은 결과는 최적화 기법과 모델 간의 상호작용이 성능에 미치는 영향을 정량적으로 입증하며, 모델 특성에 적합한 최적화 전략의 선택이 성능 향상에 있어 중요함을 시사한다.

5.2 최적화 알고리즘별 성능 비교

5.2.1 최적화 알고리즘에 따른 모델 성능 비교

최적화 방법	모델	정확도	정밀도	재현율	F1 Score	AUC
Grid Search	Adaboost	0.8805	0.8887	0.8798	0.8796	0.9273
	CatBoost	0.9142	0.9019	0.9366	0.922	0.9619
	ExtraTree	0.9119	0.8999	0.936	0.9243	0.962
	Gradient Boosting	0.9165	0.9157	0.9258	0.9239	0.9666
	LightGBM	0.9195	0.9131	0.9254	0.9171	0.966
	Random Forest	0.9091	0.9065	0.9242	0.9185	0.9621
	XGBoost	0.9048	0.8951	0.9128	0.9025	0.9581
Random Search	Adaboost	0.8969	0.8877	0.9024	0.896	0.9506
	CatBoost	0.9178	0.9119	0.9523	0.9223	0.9659
	ExtraTree	0.9176	0.9145	0.9309	0.9178	0.964
	Gradient Boosting	0.9249	0.9176	0.9353	0.9255	0.9714
	LightGBM	0.9121	0.9183	0.9102	0.912	0.9672
	Random Forest	0.8998	0.8946	0.9072	0.8999	0.9566
	XGBoost	0.9047	0.8947	0.9229	0.9078	0.9551
Bayesian Search	Adaboost	0.8887	0.89	0.8873	0.8873	0.9364
	CatBoost	0.9269	0.9156	0.9408	0.9268	0.9697
	ExtraTree	0.9149	0.9019	0.9347	0.9178	0.9682
	Gradient Boosting	0.9185	0.9122	0.9304	0.9204	0.9691
	LightGBM	0.9164	0.9096	0.9286	0.9179	0.9687
	Random Forest	0.9157	0.9009	0.9354	0.9171	0.9606
	XGBoost	0.9127	0.8963	0.9377	0.9156	0.9642
Genetic Algorithm	Adaboost	0.9235	0.9251	0.9207	0.9223	0.9428
	CatBoost	0.9403	0.9278	0.9558	0.9408	0.9733
	ExtraTree	0.9279	0.9157	0.9449	0.9298	0.9602
	Gradient Boosting	0.947	0.9368	0.9585	0.947	0.9698
	LightGBM	0.9402	0.9361	0.9431	0.9392	0.9703
	Random Forest	0.9127	0.8972	0.9322	0.9137	0.9602
	XGBoost	0.9362	0.931	0.9444	0.937	0.9615

표 4 트리 기반 앙상블 모델에 대한 최적화 기법별 성능 비교

Grid Search는 모든 가능한 하이퍼파라미터 조합을 체계적으로 탐색하는 방식으로, 다양한 모델에서 비교적 안정적이고 신뢰할 수 있는 성능을 보였다. 예를 들어, Adaboost, RandomForest, XGBoost와 같은 모델에서 일관성 있는 결과를 기록하며, 특정 조합에서 최적의 성능을 보장할 수 있다는 강점을 지닌다. CatBoost와 ExtraTree 모델에서 각각 F1 스코어가 0.922와 0.9243으로 측정된 결과는 Grid Search가 특정 하이퍼파라미터 탐색에 있어서 신뢰할 수 있는 최적화를 제공할 수 있음을 보여준다. 그러나 탐색 공간이 클수록 연산 비용이 크게 증가하는 특성상, 많은 시간과 자원이 요구되며, 대규모 탐색에서는 현실적으로 제약이 발생할 수 있다. 이러한 효율성 측면에서의 한계는 Grid Search의 주요 단점으로 꼽힌다.

Random Search는 Grid Search에 비해 탐색 효율성을 향상시킨 방식으로, 무작위로 하이퍼파라미터 조합을 선택하여 탐색하는 기법이다. 탐색 시간과 자원 면에서 상대적으로 효율적이지만, 탐색 공간에서 최적의 조합을 반드시 보장하지는 않는다. CatBoost 모델에서는 F1 스코어가 0.9223으로 측정되었으며, GradientBoosting 모델에서도 0.9255의 높은 F1 스코어를 기록하였다. 이러한 결과는 무작위 탐색에도 불구하고 일부 모델에서 우수한 성능을 보일 수 있음을 나타낸다. 그러나 무작위성의 특성상 성능의 일관성을 보장하기 어려우며, 탐색이 전역 최적해에 수렴하는 데에는 한계가 있을 수 있다.

Bayesian Search는 탐색 효율성과 정확도 간의 균형을 목표로 하는 최적화 기법으로, 이전 탐색 결과를 바탕으로 하이퍼파라미터 조합을 점진적으로 최적화하는 방식이다. 이 기법은 대부분의 모델에서 일관된 성능을 나타내며, CatBoost 모델에서는 F1 스코어가 0.9268로 기록되었다. 또한 LightGBM과 GradientBoosting 모델에서도 유사한 성능을 보였으며, 탐색 효율성을 극대화할 수 있는 장점을 보여준다. Bayesian Search는 탐색 초기 단계에서의 설정에 따라 성능이 달라질 수 있으며, 탐색이 지나치게 초기 단계에 영향을 받을 경우 탐색 효율이 저하될 수 있다는 한계가 있지만, 전체적으로 안정적이고 효과적인 탐색 성능을 제공한다.

Genetic Algorithm은 모든 모델에서 전반적으로 가장 우수한 성능을 나타낸 최적화 기법으로 평가되었다. 진화적 연산을 통해 하이퍼파라미터 탐색을 최적화하며, Adaboost 모델에서는 F1 스코어가 0.9223, CatBoost에서는 0.9408, GradientBoosting에서는 0.947로 기록되었고, 전반적인 AUC 값도 모든 모델에서 우수한 수치를 보였다. 이는 Genetic Algorithm이 복잡하고 넓은 하이퍼파라미터 탐색 공간에서 다양한 가능성을 탐구하며, 전역 최적화를 수행할 수 있는 강력한 능력을 보유하고 있음을 의미한다. 이 알고리즘은 선택, 교차, 돌연변이 등 진화적 과정을 통해 적합도를 향상시키며, 탐색이 어려운 문제에서도 뛰어난 성능을 보여준다. 그러나 복잡한 탐색 과정에서 연산 비용이 증가할 수 있으며, 수렴 속도가 느려질 가능성도 존재하기 때문에

사용 시 신중한 설정이 요구된다.

전반적으로, 최적화 기법에 따라 모델의 성능은 상이하게 나타났으며, Genetic Algorithm은 대체로 가장 우수한 성능을 보였다. Grid Search와 Random Search는 상대적으로 단순한 탐색 방식을 통해 안정적인 성능을 보였으나, 탐색 공간의 크기에 따라 한계가 있었다. Bayesian Search는 효율적인 탐색과 빠른 수렴을 통해 우수한 성능을 달성할 수 있었으나, 일부 모델에서는 성능 편차가 존재하였다.

5.2.2 최적화 알고리즘별 모델 간 일관성 평가

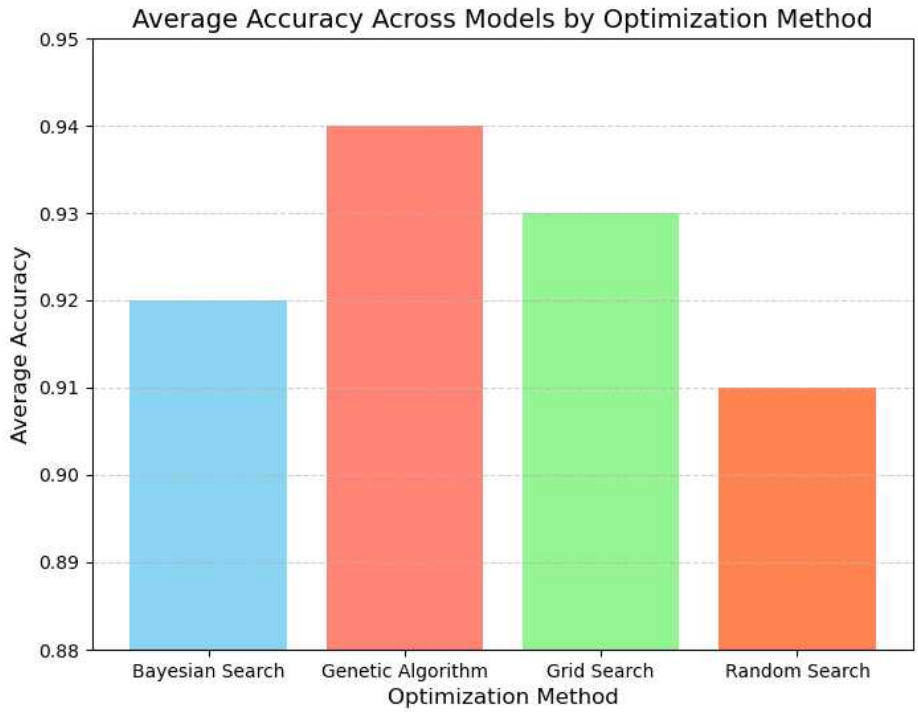

그림 9 최적화 방법에 의한 모델 간 평균 정확도

위 그래프는 다양한 트리 기반 앙상블 모델에서 적용된 최적화 기법별 평균 정확도

(Accuracy)를 시각화한 것으로, 각 최적화 기법이 모델 성능에 미치는 전반적인 영향을 비교하는 데 목적이 있다. 이는 최적화 기법이 모델 성능을 개선하는 데 얼마나 효과적인지를 평가할 수 있는 중요한 지표로 활용된다.

Genetic Algorithm은 평균 정확도가 약 0.94로, 네 가지 최적화 기법 중 가장 높은 성능을 기록하였다. 이는 복잡한 하이퍼파라미터 탐색 공간에서 Genetic Algorithm의 전역 최적화(global optimization) 특성이 발휘된 결과로 해석된다. 특히, Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서 탁월한 성능을 보여주었으며, 이는 이 기법이 복잡한 탐색 공간에서 적합한 하이퍼파라미터 조합을 효과적으로 탐색할 수 있음을 시사한다. Genetic Algorithm은 반복적이고 진화적인 연산을 통해 최적의 조합을 점진적으로 찾아가는 방식으로, 높은 성능과 안정성을 동시에 제공한다.

Grid Search는 평균 정확도가 약 0.93로, Genetic Algorithm에 이어 두 번째로 높은 성능을 보였다. 특히, Adaboost, ExtraTree와 같은 상대적으로 단순한 모델에서 일정한 성능을 유지하며 안정적인 결과를 나타냈다. Grid Search는 모든 가능한 하이퍼파라미터 조합을 체계적으로 탐색하는 방식으로, 탐색 범위 내에서 최적의 조합을 보장한다는 장점이 있다. 그러나 탐색 공간이 크거나 복잡한 경우 연산 비용이 증가하여 효율성이 떨어질 수 있다. 이러한 특성은 복잡한 모델(Gradient Boosting, XGBoost)보다는 단순한 모델에서 안정적인 성능을 보이는 결과로 이어졌다고 해석된다.

Bayesian Search는 평균 정확도가 약 0.92로, Grid Search 및 Genetic Algorithm에 비해 다소 낮은 성능을 기록하였다. 그러나 Bayesian Search는 탐색 효율성에서 강점을 가지며, 기존 탐색 데이터를 기반으로 다음 탐색 지점을 점진적으로 결정하는 방식으로 작동한다. 이는 탐색 범위를 줄이면서도 적절한 성능을 유지할 수 있다는 점에서 효율적인 접근법으로 평가된다. 다만, Gradient Boosting 및 CatBoost와 같은 복잡한 모델에서는 Genetic Algorithm만큼의 성능을 보이지 못하였으며, 이는 Bayesian Search가 전역 최적화 측면에서는 상대적으로 제한적인 탐색을 수행했기 때문으로 추정된다.

Random Search는 평균 정확도가 약 0.91로, 네 가지 기법 중 가장 낮은 성능을 기록하였다. 무작위로 하이퍼파라미터를 선택하여 탐색하는 Random Search의 특성상, 중요한 하이퍼파라미터 조합을 놓칠 가능성이 크며, 이는 특히 복잡한 모델에서 성능 변동성을 높이는 결과를 초래하였다. 예를 들어, Gradient Boosting이나 CatBoost와 같은 모델에서는 다른 기법에 비해 낮은 성능을 보였으나, Adaboost나 Random Forest와 같은 단순한 모델에서는 비교적 안정적인 성능을 유지하였다. 이는 Random Search가 탐색 공간이 단순한 경우에는 유효하게 작동할 수 있음을 보여준다.

이 결과는 각 최적화 기법의 특성과 탐색 방식이 모델의 복잡성 및 하이퍼파라미터 탐색 공간에 따라 성능에 상이한 영향을 미친다는 점을 명확히 보여준다. 특히, Genetic Algorithm은 복잡한 모델에서 최적의 성능을 제공하며, Grid Search는 단순한 모델에서 안정적인 성능을 제공하는 반면, Random Search는 탐색 공간이 복잡할수록 성능 저하의 가능성이 크다는 점이 확인되었다. 이러한 분석은 최적화 기법 선택 시 모델의 특성과 탐색 공간의 복잡성을 고려해야 한다는 점을 강조한다.

전반적으로, 최적화 알고리즘이 모델 간 일관된 성능을 보이는 정도는 기법에 따라 상이하였다. 이러한 결과는 최적화 알고리즘의 선택이 모델의 특성과 요구사항에 따라 달라질 수 있음을 보여주며, 각 기법의 강점과 한계를 고려한 최적화 전략의 중요성을 강조한다.

5.3 모델과 최적화 알고리즘 간 상호작용 분석

5.3.1 모델과 최적화 알고리즘 조합이 성능에 미치는 상호작용 효과 분석

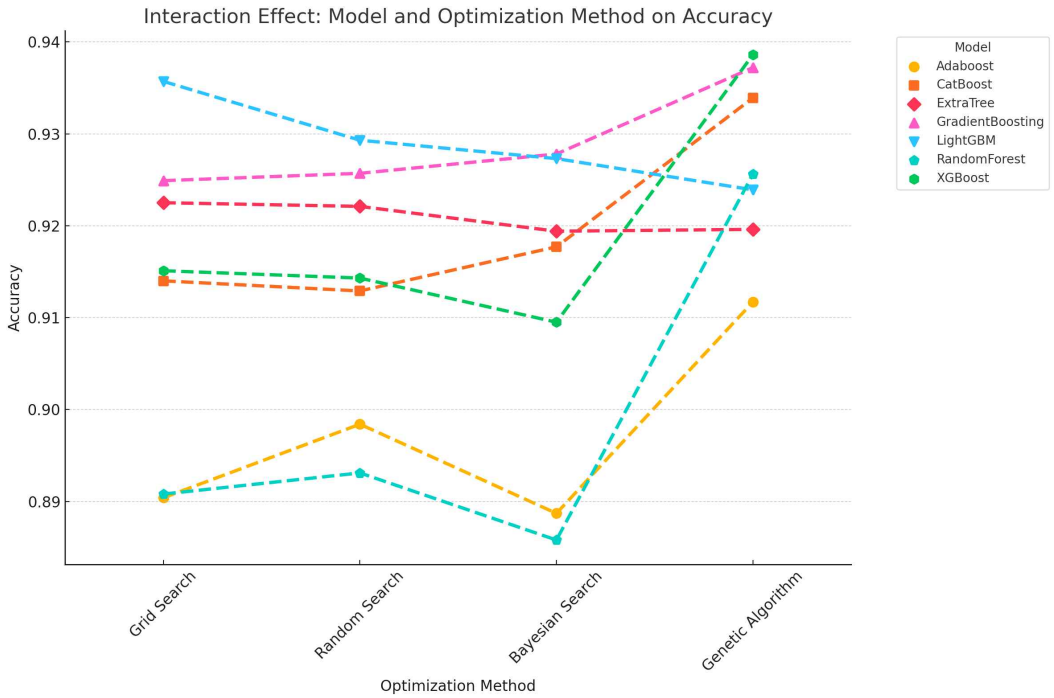

그림 10 모델-최적화 알고리즘 상호 작용 그래프

위의 그래프는 각 모델(Adaboost, CatBoost, ExtraTree, GradientBoosting, LightGBM, RandomForest, XGBoost)과 최적화 방법(Grid Search, Random Search, Bayesian Search, Genetic Algorithm) 간의 정확도(Accuracy) 상호작용을 시각적으로 보여준다.

Genetic Algorithm은 대부분의 모델에서 가장 높은 정확도를 기록하였으며, GradientBoosting, CatBoost, LightGBM과 같은 고성능 모델에서 두드러진 성능 향상을 보였다. GradientBoosting 모델은 Genetic Algorithm을 사용할 때 정확도가 다른 최적화 방법에 비해 현저히 높은 결과를 나타냈으며, 이는 Genetic Algorithm이 복잡한 하이퍼파라미터 공간을 탐색하는 데 매우 효과적임을 시사한다. Genetic Algorithm은 탐색 초기 단계에서 다양한 하이퍼파라미터 조합을 시도하며 점진적으로 최적의 조합에 수렴하는 특성을 가진다. 랜덤성과 적응성을 겸비하여 비선형적이고 복잡한 하이퍼파라미터 공간에서도 높은 효율성을 발휘하는 최적화 방법이다.

Bayesian Search는 Genetic Algorithm 다음으로 높은 성능을 기록한 최적화 기법으로 확인되었다. CatBoost와 LightGBM 모델에서는 Genetic Algorithm과 유사한 성

능을 보이며 안정성과 일관성을 입증하였다. Bayesian Search는 모델 간 성능 편차가 적고 대부분의 모델에서 안정적인 결과를 제공하는 경향을 보였다. 이러한 결과는 Bayesian Search가 복잡한 탐색 공간에서 효율적인 최적화 방법임을 보여준다.

Grid Search와 Random Search는 다른 최적화 방법에 비해 상대적으로 낮은 성능을 기록하였다. Adaboost와 RandomForest 모델에서 성능 차이가 두드러지게 나타났다.Grid Search는 전통적인 하이퍼파라미터 탐색 방식으로 탐색 공간이 제한적이기 때문에 복잡한 모델 구조를 다루는 데 한계를 보였다. 반면, Random Search는 랜덤성에 의존하여 최적의 하이퍼파라미터를 탐색하는 특성상 성능 개선에 제약이 있었다고 해석된다.

GradientBoosting과 CatBoost는 모든 최적화 방법에서 높은 정확도를 기록하며, 최적화 방법에 따른 성능 차이가 상대적으로 크게 나타났다. 이는 두 모델이 하이퍼파라미터 최적화의 영향을 크게 받는 구조적 특성을 가지고 있음을 보여준다.

Adaboost와 RandomForest는 전반적으로 낮은 성능을 기록하였으며 최적화 방법에 따른 성능 변화 폭이 작았다. 이는 두 모델이 하이퍼파라미터 최적화에 의한 성능 개선 여지가 제한적임을 나타내며 기본 구조적 특성만으로도 안정적인 성능을 발휘하는 모델임을 보여준다.일부 모델에서는 최적화 방법에 따른 성능 차이가 크지 않아 상호작용 효과가 미미한 것으로 나타났다. 이는 두 가지 이유로 설명된다. 첫째, Adaboost와 RandomForest는 하이퍼파라미터 최적화의 영향을 덜 받으며 기본 구조만으로도 안정적인 성능을 발휘한다. 둘째, Grid Search와 Random Search와 같은 일부 기법은 탐색 공간의 제약으로 인해 최적화 효과가 제한적으로 나타난다.

5.3.2 최적의 모델-알고리즘 조합 도출

소프트웨어 결함 예측에서 성능을 극대화하기 위해 모델의 선택뿐만 아니라, 적절한 하이퍼파라미터 최적화 기법의 조합이 필수적이다. 본 연구는 7개의 트리 기반 앙상블 모델(Adaboost, CatBoost, ExtraTree, GradientBoosting, LightGBM, RandomForest, XGBoost)과 4개의 하이퍼파라미터 최적화 기법(Grid Search, Random Search, Bayesian Search, Genetic Algorithm)을 결합하여 성능을 비교하였다. 특히, 정확도(Accuracy)를 중심으로 다양한 성능 지표를 평가하고, 이를 바탕으로 최적의 모델-알고리즘 조합을 도출하고자 하였다. 분석 결과는 다음과 같다.

Model	Optimization Method	Accuracy	Precision	Recall	F1 Score	AUC
Gradient Boosting	Genetic Algorithm	0.947	0.9368	0.9585	0.947	0.9698
CatBoost	Genetic Algorithm	0.9403	0.9278	0.9558	0.9408	0.9733
LightGBM	Genetic Algorithm	0.9402	0.9361	0.9431	0.9392	0.9703
XGBoost	Genetic Algorithm	0.9362	0.931	0.9444	0.937	0.9615
ExtraTree	Genetic Algorithm	0.9279	0.9157	0.9449	0.9298	0.9602
CatBoost	Bayesian Search	0.9269	0.9156	0.9408	0.9268	0.9697
Gradient Boosting	Random Search	0.9249	0.9176	0.9353	0.9255	0.9714
Adaboost	Genetic Algorithm	0.9235	0.9251	0.9207	0.9223	0.9428
LightGBM	Grid Search	0.9195	0.9131	0.9254	0.9171	0.966
Gradient Boosting	Bayesian Search	0.9185	0.9122	0.9304	0.9204	0.9691

표 6 최적의 모델-알고리즘 조합 도출 결과

정확도를 기준으로 상위 5개의 모델-알고리즘 조합은 모두 Genetic Algorithm을 사용한 경우에 해당되었다. 이 결과는 Genetic Algorithm이 복잡한 하이퍼파라미터 공간을 탐색하는 데 있어 매우 효과적인 최적화 기법임을 보여준다. 상위 조합의 주요 결과는 아래와 같다.

- GradientBoosting + Genetic Algorithm

GradientBoosting 모델은 모든 성능 지표에서 가장 높은 값을 기록하였다. 특히, 정확도와 F1 Score에서 우수한 결과를 보여 Genetic Algorithm과의 조합이 최적의 성능을 발휘함을 확인할 수 있다.

- CatBoost + Genetic Algorithm

CatBoost는 AUC에서 가장 높은 성능을 기록하며, 다른 성능 지표에서도 GradientBoosting 다음으로 우수한 결과를 보였다. 이는 CatBoost가 Genetic Algorithm과 함께 높은 효율성을 발휘할 수 있음을 보여준다.

- LightGBM + Genetic Algorithm

LightGBM은 정확도와 AUC에서 CatBoost와 유사한 성능을 보였으며, 전반적으로 안

정적이고 우수한 결과를 기록하였다.

• XGBoost + Genetic Algorithm
XGBoost는 높은 정확도와 F1 Score를 기록하며, Genetic Algorithm과의 조합에서 성능이 크게 향상되었다.

• ExtraTree + Genetic Algorithm
ExtraTree는 다른 고성능 모델에 비해 낮은 성능을 기록했으나, Genetic Algorithm과의 조합에서 성능이 안정적으로 개선되었다.

GradientBoosting과 Genetic Algorithm의 조합이 정확도, F1 Score, AUC 등 모든 주요 성능 지표에서 최상의 결과를 보였다. 이러한 결과는 GradientBoosting이 Genetic Algorithm과 결합했을 때 복잡한 하이퍼파라미터 공간을 효과적으로 탐색하며, 소프트웨어 결함 예측 문제에서 최적의 조합임을 나타낸다.

모든 상위 조합에서 Genetic Algorithm이 사용되었으며, 이는 복잡한 하이퍼파라미터 탐색에서 Genetic Algorithm의 탁월한 효율성을 입증한다. Genetic Algorithm은 탐색 초기 단계에서 다양한 조합을 실험하고, 점진적으로 최적의 파라미터 조합에 수렴하는 과정에서 높은 성능을 발휘하였다.

GradientBoosting 외에도 CatBoost와 LightGBM은 안정적이고 높은 성능을 기록하였으며, 특히 CatBoost는 AUC 지표에서 가장 높은 성능을 보였다. 이는 연구 목적에 따라 GradientBoosting 외의 조합도 적합할 수 있음을 보여준다.

Grid Search와 Random Search는 상위 조합에 포함되지 않았으며, 이는 전통적인 탐색 기법이 복잡한 탐색 공간을 다루는 데 한계가 있음을 보여준다. 반면, Bayesian Search는 Genetic Algorithm에 비해 다소 낮은 성능을 보였지만 안정적이고 일관된 결과를 제공하였다.

제 6 장 결론

본 연구는 소프트웨어 결함 예측 문제에서 트리 기반 앙상블 모델의 성능을 극대화하기 위해 하이퍼파라미터 최적화 기법이 성능에 미치는 영향을 체계적으로 분석하였다. 7개의 트리 기반 앙상블 모델과 4개의 하이퍼파라미터 최적화 기법(Grid Search, Random Search, Bayesian Search, Genetic Algorithm)을 대상으로 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 Score, AUC 등의 성능 지표를 비교하고, 모델-최적화 기법 조합 간의 상호작용 효과를 평가하였다.

GradientBoosting과 Genetic Algorithm의 조합이 정확도와 AUC를 포함한 모든 주요 성능 지표에서 최상의 결과를 기록하며, 소프트웨어 결함 예측 문제에 가장 적합한 조합으로 선정되었다. CatBoost와 LightGBM 모델도 높은 성능을 보였으며, 특히 CatBoost는 AUC 지표에서 가장 높은 값을 기록하였다.

모든 상위 조합에서 Genetic Algorithm이 사용되었으며, 이는 복잡한 하이퍼파라미터 공간을 탐색하는 데 있어 Genetic Algorithm이 매우 효과적인 최적화 기법임을 입증한다. 탐색 초기 단계에서 다양한 하이퍼파라미터 조합을 실험하고, 점진적으로 최적의 조합에 수렴하는 Genetic Algorithm의 특성이 우수한 성능을 발휘한 주요 요인으로 분석된다.

GradientBoosting과 CatBoost는 하이퍼파라미터 최적화에 따라 성능이 크게 향상되었으며, 최적화 방법에 따른 성능 변화 폭이 상대적으로 큰 모델로 나타났다. 반면, Adaboost와 RandomForest는 성능 변화 폭이 작았으며, 이는 구조적 특성이 최적화 기법의 영향을 제한적으로 받는 모델임을 시사한다.

Grid Search와 Random Search는 복잡한 하이퍼파라미터 공간에서 제한적인 탐색 성능을 보여, 상위 조합에 포함되지 않았다. Bayesian Search는 안정적이고 일관된 성능을 제공하며, Genetic Algorithm의 대안으로 고려할 수 있는 기법으로 평가되었다.

아래는 서론에서 언급했던 연구 질문에 대한 답변이다.

1. 특정 최적화 기법이 특정 모델에서 더 높은 성능을 나타내는가?

특정 최적화 기법은 특정 모델에서 성능을 더 크게 향상시키는 패턴을 보인다. 예를 들어, Genetic Algorithm은 복잡한 하이퍼파라미터 구조를 가진 CatBoost와 Gradient Boosting과 같은 모델에서 가장 높은 F1 Score와 AUC를 기록하며, Bayesian Optimization도 유사한 경향을 보였다. 반면, Random Search와 Grid Search는 Adaboost와 Random Forest 같은 비교적 간단한 구조를 가진 모델에서 성능이 더 균일하게 나타났다.

이 결과는 모델별 특성과 하이퍼파라미터 최적화 기법 간의 상호작용을 고려하여 특정 모델에 적합한 최적화 전략을 선택하는 것이 성능 향상에 중요하다는 점을 시사한다.

2. 하이퍼파라미터 최적화 기법과 모델 간 상호작용이 존재하는가?

본 연구에서 수행한 이원 분산 분석 결과, 하이퍼파라미터 최적화 기법과 모델 간에는 통계적으로 유의미한 상호작용이 존재함을 확인하였다. 이는 특정 모델에서 특

정 최적화 기법이 다른 기법보다 더 높은 성능을 나타낼 수 있다는 것을 의미한다. 예를 들어, Gradient Boosting과 XGBoost에서는 Genetic Algorithm이 성능 향상에 가장 효과적인 반면, LightGBM에서는 Bayesian Optimization이 더 높은 성능을 기록하였다. ExtraTrees의 경우 Grid Search와 Random Search가 높은 성능을 보이며, 탐색 효율성과 성능의 균형을 유지하였다. 이러한 결과는 모델별로 최적화 기법의 효율성이 다르게 작용할 수 있음을 나타내며, 각 모델에 적합한 최적화 기법을 선택하는 것이 중요함을 보여준다.

본 연구는 하이퍼파라미터 최적화 기법의 선택이 모델 성능에 미치는 영향을 실증적으로 분석하고, 최적의 모델-알고리즘 조합을 제시하였다. 이러한 결과는 소프트웨어 결함 예측뿐만 아니라 다른 머신러닝 응용 분야에서도 하이퍼파라미터 최적화 기법을 선택하는 데 있어 실질적인 지침으로 활용될 수 있다. 특히, GradientBoosting과 Genetic Algorithm의 조합은 복잡한 데이터와 모델 구조를 다루는 데 있어 강력한 선택지임을 입증하였다.

이 연구의 기여는 소프트웨어 결함 예측뿐만 아니라 머신러닝의 다양한 응용 분야에도 확장 가능하다. 소프트웨어 결함 예측은 소프트웨어 품질 보증을 위한 데이터 기반 접근법의 대표적 사례로, 본 연구는 최적화된 결함 탐지 모델이 소프트웨어 유지보수 비용 절감, 개발 시간 단축, 품질 향상 등에 실질적으로 기여할 수 있음을 보여준다. 나아가, 본 연구는 결함 예측 문제를 넘어 의료, 금융, 제조업 등 다양한 데이터 기반 의사결정 문제에서도 하이퍼파라미터 최적화 기법 선택에 대한 실질적 지침을 제공하며, 데이터 기반 모델링의 신뢰성과 효율성을 높이는 데 기여할 것으로 기대된다.

본 연구는 소프트웨어 결함 예측 분야에서 트리 기반 앙상블 모델의 최적화 가능성을 제시함과 동시에, 모델 특성과 최적화 기법 간의 적합성을 고려한 최적화 전략의 필요성을 나타내었다. 이를 통해 소프트웨어 품질 보증을 위한 데이터 기반 접근법의 실질적인 활용 가능성을 높이고, 보다 신뢰성 있는 소프트웨어 개발 환경을 구축하는 데 기여할 것으로 기대된다.

또한, 본 연구는 하이퍼파라미터 최적화 기법과 모델 간 상호작용 효과를 규명하고, 특정 기법이 특정 모델에서 특히 효과적인 이유를 체계적으로 분석하였다. 이를 통해 소프트웨어 품질 보증뿐만 아니라 데이터 중심의 다양한 문제 해결 과정에서 최적화 전략의 선택이 성능 향상에 중요한 요소임을 명확히 제시하며, 향후 소프트웨어 결함 예측(SDP) 분야에서 최적화 전략 수립과 성능 개선에 중요한 실질적 기여를 할 것이다.

데이터 셋만 결함 예측 데이터셋을 썼다고 해서 결함 예측 분야에서 라고 할 수 있나요? 이 부분을 적극 해명할 수 있도록 논문에 정보 및 의견을 더 넣어야 할 것 같아요. 현재 논문은 일반적인 ML 연구에서 hyperparam 최적화 한다는 느낌이 강하고 이러면 벌써 많은 연구가 진행되어 현 연구는 크게 새로울게 없어집니다. 해당 커멘트 반영하여 수정하셔야 합니다.

또, 해당 연구의 기여가 너무 간결하고 일반적인데 이 역시 연구 (실험 + 방법)에 직관적 연관성이 있으며 더욱 구체적인 내용으로 진행해주시면 좋겠습니다.

참 고 문 헌

[1] T. Menzies, Z. Milton, B. Turhan, B. Cukic, Y. Jiang, and A. Bener, "Defect prediction from static code features: Current results, limitations, new approaches," Autom. Softw. Eng., vol. 17, pp. 375 - 407, 2010.

[2] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," IEEE Trans. Softw. Eng.,

vol. 45, no. 7, pp. 683 – 711, Jul. 2019, doi: 10.1109/TSE.2018.2794977.

[3] M. Ali et al., "Software defect prediction using an intelligent ensemble-based model," IEEE Access, vol. 12, pp. 20376 – 20395, 2024, doi: 10.1109/ACCESS.2024.3358201.

[4] H. Aljamaan and A. Alazba, "Software defect prediction using tree-based ensembles," in Proc. 16th ACM Int. Conf. Predictive Models Data Analytics Softw. Eng. (PROMISE), New York, NY, USA, 2020, pp. 1 – 10, doi: 10.1145/3416508.3417114.

[5] A. Alazba and H. Aljamaan, "Software defect prediction using stacking generalization of optimized tree-based ensembles," Appl. Sci., vol. 12, no. 9, p. 4577, 2022.

[6] G. M. Habtemariam, S. K. Mohapatra, and H. W. Seid, "Optimized ensemble learning for software defect prediction with hyperparameter tuning," J. Theor. Appl. Inf. Technol., vol. 102, no. 14, 2024.

[7] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," J. Artif. Intell. Res., vol. 16, pp. 321 – 357, 2002.

[8] M. K. Suryadi et al., "Comparative study of various hyperparameter tuning on random forest classification with SMOTE and feature selection using genetic algorithm in software defect prediction," J. Electron. Electromed. Eng. Med. Informatics, vol. 6, no. 2, pp. 137 – 147, 2024.

[9] T. N. Al-Hadidi and S. O. Hasoon, "Software defect prediction using extreme gradient boosting (XGBoost) with optimization hyperparameter," Al-Rafidain J. Comput. Sci. Math., vol. 18, no. 1, 2024.

[10] J. R. Quinlan, "Improved use of continuous attributes in C4.5," J. Artif. Intell. Res., vol. 4, pp. 77 – 90, 1996.

[11] L. Breiman, "Bagging predictors," Mach. Learn., vol. 24, no. 2, pp. 123 – 140, 1996.

[12] L. Breiman, "Random forests," Mach. Learn., vol. 45, no. 1, pp. 5 – 32, 2001.

[13] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," J. Comput. Syst. Sci., vol. 55, no. 1, pp. 119 – 139, 1997.

[14] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discov. Data Min., 2016, pp. 785 - 794.

[15] G. Ke et al., "LightGBM: A highly efficient gradient boosting decision tree," Adv. Neural Inf. Process. Syst., vol. 30, 2017.

[16] L. Prokhorenkova et al., "CatBoost: Unbiased boosting with categorical features," Adv. Neural Inf. Process. Syst., vol. 31, 2018.

[17] F. Alves Portela, G. Papadakis, and J. C. Vassilicos, "The role of coherent structures and inhomogeneity in near-field interscale turbulent energy transfers," J. Fluid Mech., vol. 896, p. A16, 2020.

[18] A. Liaw and M. Wiener, "Classification and regression by randomForest," R News, vol. 2, no. 3, pp. 18 - 22, 2002.

[19] P. Geurts, D. Ernst, and L. Wehenkel, "Extremely randomized trees," Mach. Learn., vol. 63, pp. 3 - 42, 2006.

[20] G. Biau, "Analysis of a random forests model," J. Mach. Learn. Res., vol. 13, pp. 1063 - 1095, 2012.

[21] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," Ann. Stat., pp. 1189 - 1232, 2001.

[22] L. Mason, J. Baxter, P. Bartlett, and M. Frean, "Boosting algorithms as gradient descent," Adv. Neural Inf. Process. Syst., vol. 12, 1999.

[23] A. Natekin and A. Knoll, "Gradient boosting machines, a tutorial," Front. Neurorobotics, vol. 7, p. 21, 2013.

[24] X. He, H. Fu, A. Liu, and M. Yu, "Practical lessons from predicting clicks on ads at Facebook," in Proc. 8th Int. Workshop Data Min. Online Advertising, 2014.

[25] J. Magdum et al., "A computational evaluation of distributed machine learning algorithms," in Proc. 2019 IEEE 5th Int. Conf. Convergence Technol. (I2CT), 2019.

[26] L. Prokhorenkova et al., "CatBoost: Unbiased boosting with categorical features," Adv.

Neural Inf. Process. Syst., vol. 31, 2018.

[27] A. V. Dorogush, V. Ershov, and A. Gulin, "CatBoost: Gradient boosting with categorical features support," arXiv preprint arXiv:1810.11363, 2018.

[28] R. E. Schapire, "A brief introduction to boosting," in Proc. Int. Joint Conf. Artif. Intell., vol. 99, pp. 1401 – 1406, 1999.

[29] H. Drucker, "Improving regressors using boosting techniques," in Proc. Int. Conf. Mach. Learn. (ICML), vol. 97, pp. 107 – 115, 1997.

[30] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," J. Mach. Learn. Res., vol. 13, no. 2, pp. 281 – 305, 2012.

[31] C.-W. Hsu, "A practical guide to support vector classification," Dept. Comput. Sci., Nat. Taiwan Univ., 2003.

[32] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," Adv. Neural Inf. Process. Syst., vol. 25, 2012.

[33] J. H. Holland, Adaptation in Natural and Artificial Systems. Cambridge, MA: MIT Press, 1992.

[34] M. Feurer and F. Hutter, "Hyperparameter optimization," in Automated Machine Learning: Methods, Systems, Challenges, 2019, pp. 3 – 33.

[35] M. Claesen and B. De Moor, "Hyperparameter search in machine learning," arXiv preprint arXiv:1502.02127, 2015.

[36] L. Li et al., "Hyperband: A novel bandit-based approach to hyperparameter optimization," J. Mach. Learn. Res., vol. 18, no. 185, pp. 1 – 52, 2018.

[37] P. I. Frazier, "A tutorial on Bayesian optimization," arXiv preprint arXiv:1807.02811, 2018.

[38] M. Mitchell, An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press, 1998.

[39] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning.

Reading, MA: Addison-Wesley, 1989.

[40] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," IEEE Trans. Softw. Eng., vol. 33, no. 1, pp. 2 - 13, 2006.

[41] C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Syst. Appl., vol. 36, no. 4, pp. 7346 - 7354, 2009.

[42] G. Boetticher, "The PROMISE repository of empirical software engineering data," [Online]. Available: http://promisedata.org/repository. [Accessed: Nov. 25, 2024].