3p

소프트웨어 개발의 복잡성과 규모가 지속적으로 증가하면서, 소프트웨어 시스템의 결함을 조기에 예측하고 해결하는 작업은 점점 더 중요해지고 있습니다.  
소프트웨어 결함은 단순한 오류를 넘어 치명적인 시스템 중단, 데이터 손실, 그리고 보안 위협까지 초래할 수 있습니다.

특히, 개발 후반 단계에서 발견된 결함은 수정 비용이 기하급수적으로 증가하여 개발 일정 지연과 추가 유지보수 비용으로 이어지게 됩니다.

이러한 문제를 해결하기 위해 소프트웨어 결함 예측, 즉 SDP는 소프트웨어 품질 보증과 유지보수 과정에서 핵심적인 역할을 합니다.  
SDP는 소스 코드 메트릭, 예를 들어 코드 복잡도, 결합도, 라인 수와 같은 데이터를 기반으로 결함 여부를 학습합니다.

이를 통해 결함이 포함될 가능성이 높은 소프트웨어 모듈을 예측할 수 있습니다.  
이를 통해 개발팀은 한정된 자원을 취약한 영역에 우선적으로 투입할 수 있으며, 전체 프로젝트의 품질을 개선하고 유지보수의 효율성을 극대화할 수 있습니다.

4p

최근 연구에서는 트리 기반 앙상블 모델이 소프트웨어 결함 예측에서 매우 뛰어난 성능을 보이고 있어 주목받고 있습니다.  
트리 기반 앙상블 모델은 여러 개의 결정 트리를 결합하여 최종 예측을 산출하는 기법으로, 특히 불균형 데이터나 결측치가 포함된 데이터에서 높은 내성을 보입니다.  
대표적인 모델로는 **랜덤 포레스트(Random Forest)**, **엑스트라 트리(Extra Trees)**, **그라디언트 부스팅 머신(GBM)**, **XGBoost**, **LightGBM**, **CatBoost**, **AdaBoost** 등이 있습니다.

이들 모델은 다양한 방식으로 트리를 결합해 성능을 극대화하며, 소프트웨어 결함 예측뿐 아니라 다양한 분야에서 높은 성능을 보여줍니다.

그러나 트리 기반 앙상블 모델의 뛰어난 성능을 온전히 발휘하기 위해서는 하이퍼파라미터 최적화가 필수적입니다.  
하이퍼파라미터는 모델이 데이터를 학습하는 방식과 일반화 성능에 직접적인 영향을 미칩니다. 하지만, 하이퍼파라미터 조합의 가능성이 매우 많기 때문에 모든 조합을 시도하는 것은 현실적으로 불가능합니다.

효율적으로 최적의 성능을 달성하기 위해 다양한 최적화 기법이 제안되었습니다.  
가장 대표적으로 **그리드 서치(Grid Search)**, **랜덤 서치(Random Search)**, **베이지안 최적화(Bayesian Optimization)**, **유전 알고리즘(Genetic Algorithm)** 등이 있습니다.  
이들 기법은 각각의 특성과 장단점을 가지며, 특정 상황에서 모델 성능을 효과적으로 향상시킬 수 있습니다.

결론적으로, 소프트웨어 결함 예측의 중요성은 점점 커지고 있으며, 트리 기반 앙상블 모델과 하이퍼파라미터 최적화 기법은 이를 해결하기 위한 강력한 도구입니다.  
오늘 발표를 통해 이러한 방법론들이 어떤 방식으로 적용되고, 최적의 결과를 도출하는지 더 깊이 알아보겠습니다.

5p

결론적으로, 이 질문은 **모델 특성과 최적화 기법 간의 관계를 규명**하고,

이를 통해 **최적화 전략의 선택을 체계화**하며, **실질적 응용과 학문적 기여를 동시에 도모**하기 위해 제기됩니다.

6p

결론적으로, 이 질문은 **모델 특성과 최적화 기법의 관계를 분석**하여,

1. 최적화 효과를 극대화하고,
2. 최적화 기법 선택 전략을 제안하기 위해 제안됩니다.

7p

트리 기반 앙상블 모델은 여러 개의 결정 트리를 결합하여 예측의 정확성과 견고성을 높이는 머신러닝 기법으로, 다양한 예측 및 분류 문제에서 널리 활용됩니다.  
이 모델은 개별 결정 트리가 가지는 한계를 보완하며, 배깅(Bagging)과 부스팅(Boosting)이라는 두 가지 주요 방식을 통해 작동합니다.

8p

트리기반 앙상블 모델에 대한 설명입니다.

랜덤 포레스트는 배깅(Bagging) 기법을 기반으로 다수의 결정 트리를 결합하여 예측 성능과 안정성을 향상시키는 앙상블 모델입니다.

엑스트라 트리는 랜덤 포레스트와 유사하지만, 더 높은 무작위성을 도입하여 모델의 편향과 분산을 조정한 앙상블 모델입니다.

**랜덤 포레스트**는 안정성과 정확성을 중시하며, 다양한 문제에서 널리 사용됩니다.**엑스트라 트리**는 빠른 학습과 더 강력한 다양성을 제공하며, 대규모 데이터와 복잡한 데이터에 적합합니다.

9p,10p 피피티 읽기

11p

**그리드 서치**는 완전한 탐색을 보장하지만, 계산 비용이 높아 제한된 탐색 공간에서 유리합니다.

**랜덤 서치**는 효율적이고 빠르지만, 최적의 조합을 놓칠 가능성이 있어 보완적으로 사용됩니다.

두 기법은 문제의 복잡도와 자원 제약에 따라 적절히 선택하거나 조합하여 활용할 수 있습니다.

12p

**장점**

* **효율성**: 기존 그리드 서치나 랜덤 서치보다 적은 평가로 최적의 조합 탐색 가능.
* **연산 비용 절감**: 평가 비용이 높은 목표 함수에서도 효과적.
* **균형 탐색**: 중요한 영역을 집중 탐구하면서도 미지의 영역을 적절히 탐색.
* **응용 가능성**: 대규모 데이터셋과 복잡한 모델에 적합.

**단점**

* **제약된 탐색 공간**: 상대적으로 작은 하이퍼파라미터 공간에서 효과적.
* **초기 평가 의존성**: 초기 데이터 부족 시 모델의 부정확성 발생 가능.
* **연산 비용**: 고차원 공간에서 가우시안 프로세스 기반 최적화의 연산 비용 증가

13p

**장점**

* **전역 탐색 능력**: 탐색 공간이 크거나 비선형성이 강한 문제에서도 전역 최적해를 탐색.
* **유연성**: 초기 해의 품질에 크게 의존하지 않으며, 병렬 처리 가능.
* **적용 범위**: 다양한 해를 생성하고 지역 최적해에 빠질 가능성 낮음.

**단점**

* **계산 비용 증가**: 적합도 함수의 평가가 복잡하거나 자원이 많이 소모될 수 있음.
* **수렴 속도 느림**: 탐색 시간이 길어질 수 있음.
* **목표와 무관한 해 생성 가능성**: 탐색 공간이 지나치게 넓을 경우 비효율적 탐색 발생.

여기까지가 하이퍼파라미터 최적화 기법에 대한 설명이었습니다.

14p

이번 슬라이드에서는 사용한 nasa 데이터셋에 대한 설명을 하겠습니다.

본 연구에서는 소프트웨어 결함 예측을 위해 **NASA MDP 데이터셋**을 활용하였습니다. 이 데이터셋은 다양한 소프트웨어 프로젝트에서 수집된 결함 정보와 소스 코드 메트릭을 포함하고 있으며, 결함 예측 및 탐지 연구에서 널리 사용됩니다.

15p

이제 NASA 소프트웨어 결함 예측 데이터셋에서 수행한 **데이터 전처리 과정**에 대해 설명드리겠습니다.

먼저, **NASA 데이터셋**은 결함이 있는 모듈과 결함이 없는 모듈 간의 **데이터 불균형**이 존재합니다.  
이러한 불균형은 학습된 모델이 소수 클래스인 **결함 모듈**을 제대로 예측하지 못하는 문제를 유발할 수 있습니다.  
따라서, 본 연구에서는 데이터 불균형 문제를 완화하기 위해 **SMOTE(Synthetic Minority Over-sampling Technique)** 기법을 적용하였습니다.

16p

먼저, 실험에서 사용된 데이터셋은 훈련 데이터(80%)와 테스트 데이터(20%)로 분할하였습니다.

**데이터 분할** "데이터 분할은 Python의 train\_test\_split 함수를 사용해 구현되었습니다.

전체 데이터셋을 훈련 데이터와 테스트 데이터로 나누어, 훈련 데이터는 모델 학습과 하이퍼파라미터 최적화에, 테스트 데이터는 최적화된 모델의 일반화 성능 평가에 활용되었습니다."  
이러한 분할은 **모델의 일반화 성능**을 평가하고 **과적합을 방지**하기 위해 필수적입니다.  
또한, 데이터 분할 과정에서 **random\_state=42**를 고정하여 실험의 재현성을 확보하였습니다.  
이를 통해 동일한 데이터셋을 기반으로 반복 실험을 수행할 때도 일관된 결과를 얻을 수 있도록 설정하였습니다.

훈련 데이터와 테스트 데이터를 분리하는 이유는 **과적합을 방지**하고 모델의 **일반화 성능**을 독립적으로 평가하기 위함입니다.  
특히, 소프트웨어 결함 데이터셋은 클래스 불균형 문제가 존재하기 때문에, 훈련 데이터에 **SMOTE 기법**을 적용하여 소수 클래스를 증강하였습니다.  
이를 통해 모델이 특정 클래스에 치우치지 않고, 양쪽 클래스 모두에서 균형 잡힌 성능을 발휘하도록 했습니다.

**3. 교차 검증** "또한, 교차 검증(cross-validation)을 병행하여 모델 성능의 변동성을 줄이고 안정성을 검증하였습니다. 교차 검증은 여러 번의 데이터 분할을 통해 모델의 성능을 평가하는 방식으로, 신뢰성 있는 결과를 보장합니다."

**4. 결과 분석** "교차 검증 결과를 종합적으로 분석하여 각 최적화 기법과 모델의 성능을 비교하였습니다. 이 과정을 통해 최적화된 모델이 새로운 데이터에서도 얼마나 잘 작동하는지 확인할 수 있었습니다."

**5. 결론** "결론적으로, 데이터 분할과 교차 검증은 모델의 성능을 객관적으로 평가하고, 최적화 기법과 모델의 안정성을 검증하는 데 핵심적인 역할을 했습니다."

**17p**

먼저, 본 연구에서는 모든 모델이 동일한 초기 상태에서 시작하도록 설정하였습니다.  
이렇게 초기화를 통일한 이유는 **최적화 기법의 성능을 공정하게 비교할 수 있는 기준**을 제공하기 위함입니다.  
이를 통해, 각 기법이 모델 성능을 얼마나 효과적으로 개선할 수 있는지를 평가할 수 있습니다.

또한, 초기 상태에서의 모델 성능을 기준으로 하여, 최적화 기법이 성능을 얼마나 향상시켰는지 정량적으로 분석하였습니다.  
최적화 과정에서는 특정 하이퍼파라미터가 모델 성능에 미치는 영향을 분석하였고,  
이를 통해 각 모델별로 최적의 하이퍼파라미터 조합을 도출하고, 예측 성능을 극대화할 수 있었습니다.

마지막으로, 초기화 단계에서는 과적합을 방지하기 위해 **트리 깊이 제한**이나 **최소 샘플 분할 크기**와 같은 규제 관련 하이퍼파라미터를 조정하였습니다.  
이를 통해 모델의 **일반화 성능**을 확보하고, 안정적인 최적화를 위한 환경을 마련하였습니다.

결론적으로, 모델 초기화와 최적화 과정은 성능 비교와 개선 효과 분석, 그리고 과적합 방지의 중요한 역할을 담당합니다.

18p

본 연구에서는 트리 기반 앙상블 모델의 하이퍼파라미터 최적화를 위해 **탐색 범위를 사전에 정의**하였습니다.  
이는 최적화 기법의 효율성을 극대화하고, 불필요한 계산 자원의 낭비를 방지하기 위한 전략적 접근이었습니다.

먼저, 탐색 범위를 설정하는 이유는 각 최적화 기법의 특성 때문입니다.  
**Grid Search**와 **Random Search**는 탐색 공간이 명시적으로 요구되며, 탐색 공간이 과도하게 넓을 경우 계산 비용이 기하급수적으로 증가할 수 있습니다.  
**Bayesian Optimization**은 초기 탐색 공간을 모델링하여 점진적으로 탐색을 진행하므로, 탐색 범위가 너무 좁으면 최적해를 찾지 못할 수 있고,  
반대로 탐색 범위가 지나치게 넓으면 초기 탐색 과정이 비효율적일 수 있습니다.  
**Genetic Algorithm**은 상대적으로 초기 탐색 범위에 덜 민감하지만, 지나치게 큰 탐색 공간에서는 수렴 속도가 느려질 수 있습니다.

다음으로, 탐색 범위는 모델과 기법의 특성을 고려해 설정되었습니다.  
예를 들어, 학습률(learning rate)은 [0.01, 0.2]로 제한하여 학습 속도와 성능 저하를 방지했으며,  
트리 개수(n\_estimators)는 [50, 300] 범위로 설정해 과적합과 학습 시간 증가를 최소화했습니다.  
이러한 설정은 기존 연구에서 제시된 최적 탐색 범위를 참고하여 실용성을 확보하였습니다.

탐색 범위를 사전에 정의하는 방식은 성능 최적화와 계산 효율성을 동시에 달성할 수 있는 효과적인 전략입니다.  
또한, 연구 재현성을 보장하고, 각 최적화 기법의 성능을 균등하게 비교할 수 있도록 했습니다.

19p

**정밀도**는 모델이 양성으로 예측한 사례 중 실제로 양성인 비율을 나타내며, **양성 예측의 정확성**을 평가하는 지표입니다.  
정밀도는 특히 **False Positive**를 줄이는 데 중점을 둡니다.  
예를 들어, 소프트웨어 결함 탐지 상황에서는 **잘못된 경고를 줄이고**, 실제 결함이 있는 경우에만 탐지할 수 있도록 높은 정밀도가 중요합니다.  
이러한 높은 정밀도를 통해 개발 과정에서 불필요한 시간 낭비와 리소스 소모를 줄일 수 있습니다.

따라서, **양성 예측의 정확성**이 중요한 문제에서 정밀도는 성능 평가의 핵심 지표로 활용됩니다.

**재현율**은 **실제 양성 사례 중에서 모델이 정확히 탐지한 비율**을 나타내며,

재현율은 특히 **False Negative**를 줄이는 데 중점을 둡니다.  
즉, 중요한 양성 사례를 놓치지 않고 탐지하는 능력을 평가하는 지표입니다.

예를 들어, **소프트웨어 품질 관리**와 같은 분야에서는 결함을 놓치지 않고 탐지하는 것이 매우 중요합니다.  
재현율이 높은 모델은 실제로 존재하는 결함을 가능한 한 모두 식별할 수 있어, 품질 관리 및 유지보수의 효율성을 높이는 데 기여합니다.

따라서, 중요한 사례를 빠짐없이 탐지해야 하는 상황에서 재현율은 성능 평가의 필수적인 지표로 활용됩니다

20p

정확도는 전체 샘플 중에서 모델이 올바르게 예측한 비율을 나타내는 지표입니다.  
여기서 **TP**는 True Positive, **TN**은 True Negative, **FP**는 False Positive, **FN**은 False Negative를 의미합니다.

정확도는 전반적인 예측 성능을 나타내는 데 유용하며, 특히 클래스 불균형이 심하지 않은 데이터셋에서 효과적입니다.  
그러나, 데이터셋이 불균형한 경우에는 **False Negative**와 **False Positive**가 모델 평가에 중요한 영향을 미칠 수 있습니다.  
예를 들어, 결함 예측과 같은 문제에서는 결함을 놓치는 경우(False Negative)가 큰 비용을 초래할 수 있습니다.

따라서, 정확도는 \*\*다른 성능 지표(F1 Score, Precision, Recall 등)\*\*와 함께 해석하여 모델 성능을 더 종합적으로 평가하는 것이 중요합니다.

**F1 스코어**는 정밀도와 재현율의 **조화 평균**으로, 두 지표 간의 균형을 고려하여 모델 성능을 평가하는 지표입니다.  
F1 스코어는 특히 **정밀도와 재현율이 모두 중요할 때** 유용합니다.  
예를 들어, 결함 예측이나 의료 진단과 같이 **정확히 찾아내는 것**과 **놓치지 않는 것**이 동시에 중요한 상황에서 F1 스코어는 성능 평가의 핵심 지표로 사용됩니다.  
또한, **정밀도와 재현율 중 하나라도 낮아지면 F1 스코어도 낮아지기 때문에**, 두 지표 간의 균형이 얼마나 잘 유지되는지를 평가할 수 있습니다.

특히, F1 스코어는 **불균형한 데이터셋**에서 모델의 전반적인 예측 성능을 평가하는 데 적합합니다.

21p

지금부터 본 연구에서 수행한 **트리 기반 앙상블 모델의 하이퍼파라미터 최적화 결과**에 대해 설명드리겠습니다.

본 연구에서는 **4가지 주요 하이퍼파라미터 최적화 기법**을 적용하여, 각 모델별 최적화 성능을 비교 분석하였습니다.  
사용된 최적화 기법은 다음과 같습니다:

1. **그리드 서치(Grid Search)**
2. **랜덤 서치(Random Search)**
3. **베이지안 최적화(Bayesian Optimization)**
4. **유전 알고리즘(Genetic Algorithm)**

분석은 **NASA의 11개 소프트웨어 결함 데이터셋**을 대상으로 수행되었으며, CM1, JM1, KC1, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4의 결과를 기반으로 하였습니다.  
이를 통해, 각 모델과 최적화 기법이 다양한 데이터셋에서 **일관된 성능을 보이는지 평가**할 수 있었습니다.

**모델의 복잡도**와 **하이퍼파라미터 공간 크기**에 따라 최적화 기법의 성능이 다르게 나타났습니다

.예를 들어, 하이퍼파라미터 공간이 넓고 복잡한 경우 랜덤 서치와 유전 알고리즘이 보다 유리한 성능을 보였습니다.

반면, 그리드 서치는 탐색 비용이 높아지는 단점이 있었지만, 탐색 공간이 작은 경우에는 효과적으로 작동하였습니다.

22p

결과적으로, Genetic Algorithm은 대부분의 모델에서 가장 높은 성능을 달성하며, 특히 Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서 강력한 효과를 발휘했습니다. LightGBM의 경우 Grid Search와 Random Search가 더 적합한 성능을 보였습니다. 이러한 결과는 최적화 기법 선택이 모델 특성에 따라 달라져야 함을 보여줍니다.

23p

먼저, **성능 지표별 분석**입니다.  
**Accuracy**는 전반적인 분류 정확도를 나타내지만, 클래스 불균형 상황에서는 성능을 과대평가할 가능성이 있습니다.  
반면, **F1 Score**는 정밀도와 재현율을 동시에 고려하여, 클래스 불균형 문제에 민감하게 반응하는 지표입니다.

다음으로, **전반적인 결과**를 살펴보겠습니다.  
**Genetic Algorithm**은 대부분의 모델에서 **Accuracy**와 **F1 Score** 모두에서 최고 성능을 기록하였으며,  
특히 Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서 두드러진 성능 향상을 보여주었습니다.  
반면, **Grid Search**와 **Random Search**는 Adaboost, ExtraTree, RandomForest와 같은 단순한 모델에서 안정적인 성능을 보였습니다.  
**Bayesian Search**는 효율성과 성능의 균형을 제공하였으나, 복잡한 모델에서는 Genetic Algorithm에 약간 미치지 못하는 결과를 보였습니다.

**모델별 분석**을 살펴보면,  
Adaboost 모델은 Genetic Algorithm이 두 지표 모두에서 최고 성능을 보였습니다.  
RandomForest와 ExtraTree에서는 Grid Search와 Random Search가 유사한 Accuracy를 보였으나,  
F1 Score에서는 약간의 차이를 보여 클래스 불균형 문제 대응에 F1 Score가 더 적합함을 확인할 수 있었습니다.  
Gradient Boosting, CatBoost, XGBoost와 같은 복잡한 모델에서는 Genetic Algorithm이 두 지표 모두에서 최고 성능을 기록하였고,  
Bayesian Search도 우수한 성능을 보였으나 탐색 공간 복잡성에서는 Genetic Algorithm이 더 뛰어났습니다.  
LightGBM 모델에서는 Random Search와 Genetic Algorithm이 높은 성능을 기록하였으나, Grid Search는 상대적으로 낮은 성능을 보였습니다.

마지막으로, 결론입니다.  
**Genetic Algorithm**은 특히 복잡한 모델에서 최적의 성능을 제공하며, 전역 최적화를 통해 효과적인 하이퍼파라미터 탐색이 가능함을 확인했습니다.  
Grid Search와 Random Search는 단순한 모델에서 효율적으로 작동하였습니다.

24p

이번 슬라이드에서는 최적화 기법별 성능 비교의 표를 보시겠습니다.

먼저, **Grid Search**입니다.  
Grid Search는 모든 가능한 하이퍼파라미터 조합을 체계적으로 탐색하는 방식으로, **Adaboost, RandomForest, XGBoost**와 같은 모델에서 안정적이고 일관된 성능을 보였습니다.  
특히, CatBoost와 ExtraTree에서 각각 **F1 Score 0.922**와 **0.9243**을 기록하며 신뢰할 수 있는 성능을 제공했습니다.  
하지만 탐색 공간이 클수록 연산 비용이 크게 증가하는 특성 때문에, 대규모 탐색에서는 제약이 발생할 수 있습니다.

다음은 **Random Search**입니다.  
Random Search는 무작위로 하이퍼파라미터 조합을 선택하여 탐색 시간을 줄이는 방식으로,  
CatBoost와 GradientBoosting에서 각각 **F1 Score 0.9223**과 **0.9255**를 기록하며 우수한 성능을 보였습니다.  
그러나 무작위성으로 인해 성능의 일관성을 보장하기 어렵고, 최적 조합을 반드시 찾아내지 못할 수 있습니다.

세 번째로, **Bayesian Search**입니다.  
Bayesian Search는 탐색 효율성과 정확도의 균형을 목표로 하며, 이전 탐색 결과를 바탕으로 최적화를 점진적으로 수행합니다.  
CatBoost에서는 **F1 Score 0.9268**, LightGBM과 GradientBoosting에서도 유사한 높은 성능을 보였습니다.  
그러나 탐색 초기 설정에 따라 성능 편차가 발생할 수 있으며, 초기 탐색 단계의 효율성이 낮아질 가능성이 있습니다.

마지막으로, **Genetic Algorithm**입니다.  
Genetic Algorithm은 모든 모델에서 전반적으로 가장 우수한 성능을 보였습니다.  
Adaboost에서는 **F1 Score 0.9223**, CatBoost에서는 **0.9408**, GradientBoosting에서는 **0.947**을 기록하며 최고 성능을 달성했습니다.  
진화적 연산을 통해 복잡한 탐색 공간에서도 전역 최적화가 가능했으나, 높은 연산 비용과 수렴 속도가 느릴 수 있다는 점은 한계로 지적됩니다.

결론적으로, 최적화 기법은 모델 특성에 따라 성능이 다르게 나타났습니다.  
**Genetic Algorithm**은 복잡한 모델에서 최적의 성능을 제공하였으며, Grid Search와 Random Search는 단순한 모델에서 안정적인 성능을 보였습니다.  
**Bayesian Search**는 효율적이고 빠른 수렴을 제공했지만, 복잡한 모델에서는 Genetic Algorithm에 미치지 못했습니다.

따라서, 모델 특성에 따라 최적화 기법을 전략적으로 선택하는 것이 중요합니다.

25p

이번 슬라이드에서는 **최적화 기법별 평균 정확도 비교** 결과를 설명드리겠습니다.

첫 번째로, 그래프의 주요 목적은 **트리 기반 앙상블 모델에서 최적화 기법이 성능에 미치는 영향을 비교**하는 것입니다.  
이를 통해 각 기법이 **모델 성능을 얼마나 효과적으로 개선할 수 있는지** 확인할 수 있었습니다.

먼저 **Genetic Algorithm**입니다.  
Genetic Algorithm은 평균 정확도가 약 **0.94**로, 네 가지 기법 중 가장 높은 성능을 보였습니다.  
이는 Gradient Boosting, CatBoost, XGBoost와 같은 **복잡한 모델에서 탁월한 성능**을 나타낸 결과입니다.  
특히, **전역 최적화 특성**을 통해 복잡한 탐색 공간에서도 적합한 하이퍼파라미터를 효과적으로 탐색할 수 있었습니다.

다음은 **Grid Search**입니다.  
Grid Search는 평균 정확도가 약 **0.93**로, Genetic Algorithm에 이어 두 번째로 높은 성능을 기록했습니다.  
특히 Adaboost, ExtraTree와 같은 **단순 모델에서 안정적이고 신뢰할 수 있는 성능**을 보였으며, 모든 조합을 체계적으로 탐색하기 때문에 탐색 범위 내에서 **최적의 조합을 보장**합니다.  
그러나 복잡한 탐색 공간에서는 연산 비용이 증가하여 효율성이 떨어지는 한계를 보였습니다.

세 번째로 **Bayesian Search**입니다.  
Bayesian Search는 평균 정확도가 약 **0.92**로, 탐색 효율성과 정확도의 균형을 목표로 한 기법입니다.  
CatBoost, LightGBM, Gradient Boosting 등에서 높은 성능을 보였으나, **전역 최적화에서는 Genetic Algorithm만큼의 성능은 달성하지 못한** 결과를 보였습니다.  
탐색 범위를 점진적으로 줄이는 방식으로 효율적인 접근이 가능하지만, 초기 설정에 따라 성능 편차가 발생할 가능성이 있습니다.

마지막으로 **Random Search**입니다.  
Random Search는 평균 정확도가 약 **0.91**로, 가장 낮은 성능을 기록했습니다.  
이는 탐색 공간이 복잡할수록 중요한 하이퍼파라미터 조합을 놓칠 가능성이 크기 때문입니다.  
그러나 Adaboost나 Random Forest와 같은 **단순 모델에서는 안정적인 성능**을 유지하여, 탐색 공간이 단순한 경우에는 유효한 기법으로 평가되었습니다.

결론적으로, 최적화 기법 선택은 **모델 특성과 탐색 공간의 복잡성을 고려해야** 합니다.  
Genetic Algorithm은 복잡한 모델에서 가장 우수한 성능을 보였으며, Grid Search와 Random Search는 상대적으로 단순한 모델에서 효과적이었습니다.  
이러한 결과는 최적화 기법과 모델 특성 간의 **상호작용**이 성능에 중요한 영향을 미친다는 점을 강조합니다.

26p

다음으로 모델과 최적화 알고리즘 간의 상호작용 효과를 분석한 결과를 말씀드리겠습니다.  
위의 그래프는 각 모델과 최적화 기법별 정확도를 비교하여 최적화 기법이 모델 성능에 미치는 영향을 시각적으로 보여줍니다.

**2. 주요 결과**  
먼저, **Genetic Algorithm**은 대부분의 모델에서 가장 높은 정확도를 기록했습니다. 특히 GradientBoosting, CatBoost, LightGBM과 같은 고성능 모델에서 두드러진 성능 향상을 보여주었는데요.

이는 Genetic Algorithm이 복잡한 하이퍼파라미터 탐색 공간에서 전역 최적화 능력을 발휘하며, 최적의 하이퍼파라미터 조합을 효과적으로 찾아내는 방식 때문입니다.

**3. Bayesian Search**  
다음으로, **Bayesian Search**는 안정성과 효율성을 겸비하여 Genetic Algorithm 다음으로 높은 성능을 기록했습니다. 특히 CatBoost와 LightGBM 모델에서는 Genetic Algorithm과 유사한 결과를 보이며, 복잡한 모델에서도 안정적인 결과를 제공하는 경향이 있었습니다.

**4. Grid Search와 Random Search**  
반면, **Grid Search**와 **Random Search**는 상대적으로 낮은 성능을 보였습니다. Grid Search는 Adaboost와 RandomForest와 같은 단순 모델에서 안정적이었지만, 복잡한 모델에서는 탐색 공간 제약으로 인해 성능 개선이 제한되었습니다.  
Random Search는 랜덤성에 의존하여 탐색 효율성을 높이긴 했으나, 복잡한 모델에서는 최적화를 충분히 달성하지 못하는 한계가 있었습니다.

**5. 모델별 분석**  
특히 GradientBoosting과 CatBoost 모델은 하이퍼파라미터 최적화에 따라 성능 차이가 크게 나타났습니다. 이는 두 모델이 최적화 기법의 영향을 크게 받는 구조적 특성을 가지고 있음을 보여줍니다.

반면, Adaboost와 RandomForest는 최적화 방법에 의한 성능 변화 폭이 적어, 기본 구조만으로도 안정적인 성능을 발휘하는 모델로 확인되었습니다.

**6. 결론**  
결론적으로, 각 최적화 기법은 모델의 구조적 특성과 탐색 공간의 복잡성에 따라 상이한 성능을 보였습니다. Genetic Algorithm은 복잡한 모델에서 특히 효과적인 결과를 제공했으며,

Grid Search와 Random Search는 단순한 모델에서 유효한 접근법으로 확인되었습니다. 이 분석은 최적화 기법 선택 시 모델의 특성을 고려하는 것이 중요함을 보여줍니다.

27p

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 위와 같습니다.

28p

1. **Genetic Algorithm의 우수성**
   1. 전반적으로, Genetic Algorithm은 복잡한 모델(예: Gradient Boosting, XGBoost, CatBoost)에서 가장 높은 성능을 기록하였습니다.
   2. 예를 들어, Gradient Boosting 모델에서 Genetic Algorithm은 F1 Score와 Accuracy 모두에서 다른 최적화 기법을 압도하며 최상의 결과를 나타냈습니다.
   3. 이는 Genetic Algorithm이 하이퍼파라미터 탐색 공간이 크고 복잡할수록 더욱 효과적으로 작동한다는 점을 보여줍니다.
2. **Grid Search의 안정성**
   1. Grid Search는 Adaboost, ExtraTree, LightGBM과 같은 상대적으로 단순한 모델에서 일관된 성능을 보였습니다.
   2. 특히, 모든 하이퍼파라미터 조합을 체계적으로 탐색하기 때문에, 단순 모델에서는 높은 신뢰성을 제공하였습니다.
   3. 그러나 복잡한 모델에서는 탐색 비용이 증가하며 성능이 다소 제한적이었습니다.
3. **Bayesian Optimization의 효율성**
   1. Bayesian Optimization은 탐색 효율성과 성능 간의 균형을 잘 유지하며, 대부분의 모델에서 안정적인 결과를 보였습니다.
   2. 하지만, 탐색 공간이 매우 크거나 복잡한 경우, Genetic Algorithm만큼의 최적화 성능을 보이지는 못했습니다.
4. **Random Search의 한계**
   1. Random Search는 무작위 탐색 방식의 특성상 성능의 변동성이 크고, 전반적으로 다른 기법에 비해 낮은 성능을 기록하였습니다.
   2. 특히, Gradient Boosting과 같은 복잡한 모델에서 중요한 하이퍼파라미터 조합을 놓치는 경우가 많았습니다.

29p 30p 연구질문 답변 읽기

31p

**연구 기여**  
이번 연구는 하이퍼파라미터 최적화 기법의 선택이 모델 성능에 미치는 영향을 실증적으로 분석하고, 최적의 모델-알고리즘 조합을 제시한 점에서 의의가 있습니다.  
특히, Gradient Boosting과 Genetic Algorithm의 조합은 복잡한 데이터와 모델 구조를 다루는 데 있어 강력한 선택지임을 입증하였습니다.  
소프트웨어 결함 예측(SDP) 분야에서는 최적화된 결함 탐지 모델을 통해 유지보수 비용을 절감하고, 개발 시간을 단축하며, 소프트웨어 품질을 향상시키는 데 기여할 수 있음을 확인하였습니다.

**향후 기대 효과**  
향후, 데이터 기반 접근법을 활용하여 신뢰성 높은 소프트웨어 개발 환경을 구축하는 데 기여할 것으로 기대됩니다.  
뿐만 아니라, SDP 분야를 넘어 다양한 머신러닝 문제에 최적화 기법을 적용하여 성능 개선 및 최적화 전략 수립에 실질적인 기여를 할 것입니다.  
이를 통해 소프트웨어 품질 보증 체계뿐 아니라 데이터 중심 의사결정 문제 해결에서도 유의미한 결과를 도출할 것으로 예상됩니다.