# 데이터 마이닝을 이용한 소프트웨어 결함의 예측 방법

부산대학교 ▌김태연 · 채흥석*

## 1. 서 론

소프트웨어를 실행하여 기대와 상이한 결과가 관찰되면 결함이 존재함을 유추하는 방법이 테스팅이라고 볼 수 있다. 뿐만 아니라 소프트웨어를 실행시키지 않고 소프트웨어 데이터에 마이닝 기법을 적용하여 결함의 존재에 대한 예측 모형을 구축한 후, 이를 이용하여 결함의 존재 여부를 추측할 수 있다. 특히 지난 20년간 소스 컨트롤 시스템과 버그 추적 시스템의 데이터를 기반으로 결함 예측 연구가 활발히 진행되고 있다[1-9].

데이터 마이닝 기법을 통한 예측 모형을 통해 결함의 존재를 유추하는 것은 실제 테스팅을 통해서 결함을 검출하는 방법에 비하여 다음과 같은 장점이 있다. 첫째, 구현 단계에서 개발자에게 자신이 개발한 많은 수의 모듈에서 결함 발생 가능성이 높은 모듈을 알려줄 수 있다. 개발자는 단위 테스트를 수행할 때 결함 발생 가능성이 높은 모듈에 대해서 더 높은 강도의 커버리지를 적용할 수 있다. 둘째, 전문 테스터는 시스템 테스트를 수행할 때 결함 발생률이 높은 모듈과 연관된 기능에 대해서는 더 많은 테스트 노력을 투입하는 전략을 적용할 수가 있다. 마지막으로, 결함 예측 모형은 소프트웨어에 대한 설계 정보를 바탕으로도 구축될 수가 있다. 그러므로 설계 단계에서 소스 코드가 작성되기에 앞서서 결함 발생 가능성이 높은 모듈을 추정할 수 있다.

그림 1은 데이터 마이닝을 통한 소프트웨어를 구성하는 각 모듈에 대한 결함을 예측하는 과정을 보여준다. 요약하면 결함 예측은 결함의 발생 연관된 소프트웨어의 구조적 특성을 측정하고, 기계학습을 통해서 이 구조적 특성과 실제 발생한 결함 정보와의 관련성을 파악하여 결함 예측 모형을 구축한다. 그리고 결함을 예측하고자 하는 모듈의 구조적 특성에 결함 예측 모형을 적용함으로써 결함 존재를 추정한다.

**단계 1)** 소프트웨어의 구조적 특성 측정

결함 예측은 심리학적으로 소프트웨어의 크기, 복잡도 등의 구조적 특성이 개발자의 정신적 부담에 영향을 미치고 결국은 결함 발생 확률에도 영향을 준다는 점에 이론적 근거를 두고 있다[10]. 예를 들어 크기, 복잡도, 결합도 등의 구조적 특성 값이 큰 모듈일수록 결함이 발생할 가능성이 클 수 있다. 소프트웨어 메트릭은 소프트웨어의 구조적 특성을 정량화하는 척도를 뜻한다. 예를 들어, Line of Code(LOC) 메트릭은 소스 코드를 바탕으로 모듈의 크기를 측정하며, Cyclomatic Complexity(CC)는 소스 코드 및 설계를 바탕으로 모듈의 복잡도를 측정한다[11].

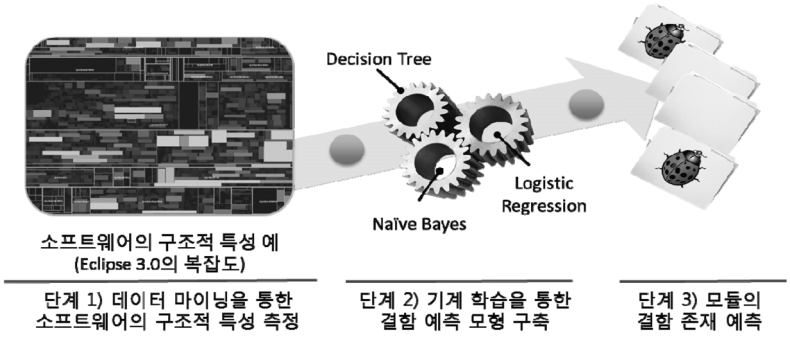


그림 1 소프트웨어의 결함 예측 방법

* 종신회원

† 이 논문은 2013년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2012R1A2A2A01015660).

**단계 2)** 결함 예측 모형 구축

소프트웨어 메트릭 측정치와 이미 알려진 결함 정보를 바탕으로 결함 예측 모형은 구축된다. 그리고 예측 모형 구축에 사용되는 알고리즘은 결함 존재의 유무를 예측하는 분류(classification) 방법과 결함 수를 예측하는 회귀(regression) 방법이 주로 사용된다[12]. 예를 들어, 분류 방법으로는 Naive Bayes, Logistic Regression, Support Vector Machine 등이 사용되며, 회귀 방법으로는 선형 회귀 방법(linear regression) 등이 대표적이다[13].

**단계 3)** 결함 예측

결함을 예측하고자 하는 소프트웨어에 대해서 단계 1)에서 선정한 소프트웨어 메트릭을 측정한다. 그리고 단계 2)에서 구축한 결함 예측 모형에 이 메트릭을 적용함으로써 결함의 발생 가능성을 예측한다.

본 원고에서는 먼저 소프트웨어 공학 데이터에 마이닝 기법을 적용한 결함 예측 방법을 소개한다. 다음으로 결함 예측에 필요한 데이터의 수집 방법을 소개하고, 분류 방식을 이용한 결함 예측 모형을 구축하는 실제 과정을 소개한다. 마지막으로 결함 예측 모형의 성능(performance), 검증(validation), 범용성(generalizability)에 관해 최신 연구 결과를 제시한다.

## 2. 결함 예측 모형의 개발

### 2.1 결함 예측 요인 결정

결함 예측 연구들에서 사용되는 예측 요인들은 소프트웨어의 구조적 특성이다. 즉 크기, 복잡도, 응집도, 결합도 등이 소프트웨어의 결함 발생과 연관된 요인으로서 결함 예측 모형을 구축할 때 활용된다[11,14,15]. 또한, 단일 요인 즉 하나의 메트릭만을 이용해서 결함을 예측하기 보다는 여러 개의 요인 즉 여러 개의 메트릭을 이용하여 결함을 예측하는 것이 일반적이다[7,12].

수집된 메트릭 데이터 모두가 결함에 영향을 미치는 것은 아니다. 그렇다고 결함에 영향을 미치는 요인들에 대한 고정된 메트릭 집합이 결정되어 있지는 않다. 또한 각 메트릭이 결함의 발생에 미치는 영향도 또한 도메인 또는 소프트웨어의 유형에 따라서 상이할 수가 있다[8,9].

그러므로 소프트웨어로부터 추출할 수 있는 수많은 구조적 특성 즉 메트릭 유형 중에서 결함 발생의 요인으로서의 역할을 하는 메트릭들을 결정할 필요가 있으며 통계적 방법과 기계 학습 방법이 주로 이용된다.

**통계적 방법[16]:** 결함에 영향을 미치는 메트릭을 통계적 수치로 평가하여 결정하는 방법이다. Logistic Regression의 경우 AIC(Akaike information criterion)나 우도비 검정(likelihood ratio test)을 이용하여 메트릭을 선택한다. 이때 메트릭을 하나씩 포함시켜가는 전진선택법과 모든 메트릭을 모형에 포함시킨 후 하나씩 제거하는 후진 제거법을 사용할 수 있다.

**기계 학습 방법[17]:** feature ranking 방법과 feature subset selection 방법이 있다. 전자는 특정 방법에 따라 개별 메트릭을 평가하여 해당 데이터에 알맞은 메트릭을 선별하며, 정보 이득(information gain), 이득 비율(gain ratio) 등의 방법이 있다. 후자는 더 좋은 예측 성능을 보이는 메트릭의 하위 집합을 찾는 방법으로 완결 탐색(ex-

표 1 결함 예측 메트릭

| 분류 | 특성 | 메트릭 | 설명 |
|---|---|---|---|
| 크기 | 코드 | LOC(Line of Code) | 소스 코드의 라인 수. 주석은 포함할 수도 제외할 수도 있다. |
| Halstead | 복잡도 | length(N) | num_operators + num_operands |
| | | volume(V) | N × log2(num_unique_operators + num_unique_operands) |
| | | level(L) | V' / V, V' = (2 + num_unique_operands) log2(2 + num_unique_operands) |
| | | difficulty(D) | 1 / L |
| 객체 지향 메트릭 | 복잡도 | WMC(Weighted Methods per Class) | 클래스의 메서드별 복잡도를 합한 값이다. 복잡도는 Cyclomatic Complexity (CC)로 계산할 수 있고, 1로 계산할 수도 있다. |
| | 상속 | DIT(Depth of the Inheritance Tree) | 개별 클래스의 상속 깊이를 측정하는데 사용된다. 상속 계층도에서 자신으로부터 최상단의 노드까지의 level로 계산한다. |
| | | NOC(Number of Children) | 클래스의 자식 클래스의 수. 이때 직계 자식 클래스만을 포함한다. |
| | 응집도 | LCOM(Lack of COhesion in Methods) | 클래스의 응집도를 측정한다. 값이 작을수록 응집도가 높다. 클래스의 멤버 변수를 공유하는 메소드 집합과 그렇지 않은 집합의 크기의 차이로 정의된다. |
| | 결합도 | CBO(Coupling Between Object classes) | 의존적 관계가 있는 다른 클래스의 수이다. 의존 관계가 있는 다른 클래스의 메서드나 멤버 변수의 수로서 정의된다. |
| | | RFC(Response For a Class) | 클래스가 호출할 수 있는 메서드의 수 |

haustive search), 휴리스틱 탐색(heuristic search) 등이 있다.

표 1은 결함과 높은 상관 관계를 보이는 것으로 알려진 대표적인 메트릭이다. LOC 메트릭은 소스의 크기를 측정한다. Hastead 메트릭은 복잡도를 측정한다. 대표적인 객체지향 메트릭인 CK 메트릭은 6개의 메트릭으로 구성되어 있고, 소스의 복잡도(WMC), 상속(DIT, NOC), 응집도(LCOM), 결합도(CBO, RFC)를 측정한다.

### 2.2 결함 예측 모형 구축

통계나 기계 학습 분야에서 분류는 특정 카테고리라고 알고 있는 데이터에 기반해 새로 관찰한 데이터를 특정 카테고리로 식별하는 문제이다[13]. 결함 예측에서 분류 알고리즘(이하 분류기)은 소프트웨어 메트릭을 이용하여 해당 모듈의 결함 유무를 판별한다. 표 2는 결함 예측 연구에서 이용된 대표적 분류 알고리즘이다[6,7]. 통계적 방법, 신경망, 커널 함수, 의사 결정 나무 등으로 다양한 알고리즘이 사용되고 있다[13,18]. 오픈 소스 소프트웨어인 Weka[19], R[20] 등을 이용하면 다양한 알고리즘으로 모형을 구축할 수 있다.

결함 예측 분야에서의 성능 즉 결함 예측의 정확도는 분류 알고리즘 마다 상이하다. 연구 초기에는 LR이 주로 사용되었다[1-3]. NB 역시 log filter와 함께 사용 시 좋은 성능을 보였다[8,21]. Lessman은 NASA 데이터 중 10개 시스템을 이용하여 22개의 분류 알고리즘을 비교

표 2 다양한 분류 알고리즘들

| Category | Classification model | Description |
|---|---|---|
| Statistical classifier | Bayesian Networks(BN) | 확률 변수들과 그들간의 조건부 확률이 비순환 그래프 형태로 표현된 확률 그래프 모델이다. |
| | Logistic Regression(LR) | 범주화된 종속 변수의 결과를 예측하기 위해 사용되는 회귀 분석의 한 종류이다. |
| | Naïve Bayes(NB) | 확률 변수들이 서로 독립이라는 가정하에 베이즈 이론에 기반한 간단한 확률 분류기이다. |
| Neural Networks | Multi-Layer Perceptron(MLP) | 입력 데이터의 집합을 적절한 출력들로 매핑하는 전진형 인공 신경망이다. |
| Support vector machine-based classifier | Least Square Support Vector Machine (LS-SVM) | SVM의 최소 제곱 버전으로 전통적인 SVM의 이차 계획법 대신에 선형 계획법으로 해를 찾는 방법이다. |
| Ensemble methods | Random Forest(RF) | 입력 데이터로 여러 개의 의사 결정 나무를 생성하고 개별 트리들의 예측 결과를 다수결로 출력하는 앙상블 학습이다. |
| Decision tree approaches | C 4.5 Decision Tree(C4.5) | 정보 이득 개념을 이용하여 입력 데이터로부터 의사 결정 나무를 생성한다. |

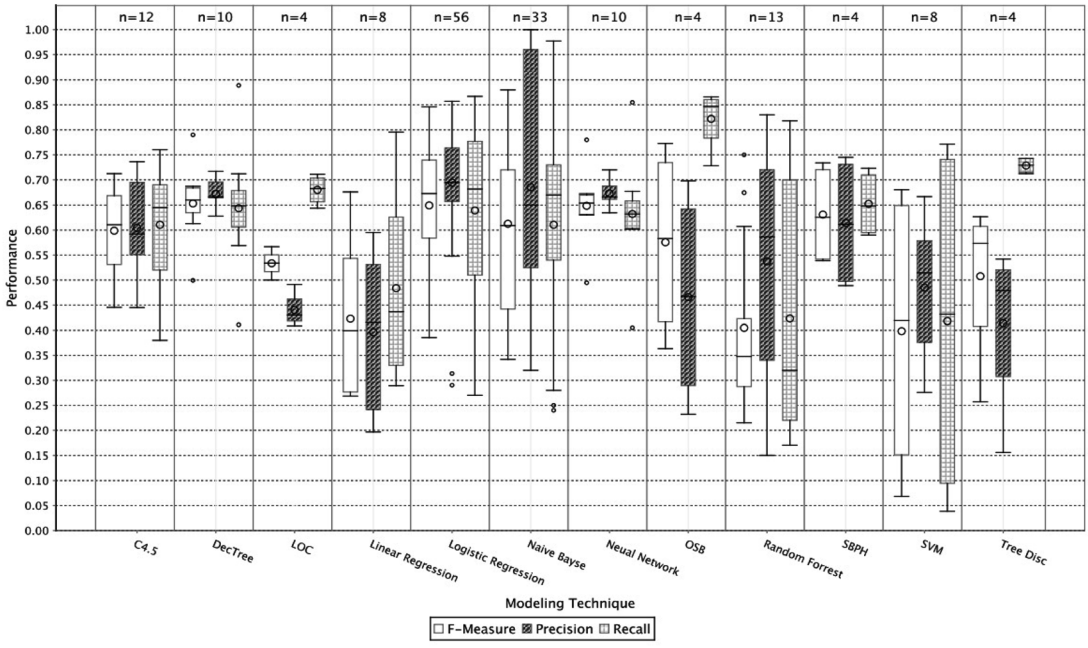


그림 2 분류 알고리즘의 성능[7]

하는 벤치마킹 실험을 실시하였다[6]. 실험 결과에 따르면 RF, LS-SVM, MLP 등이 좋은 성능을 보였다.

반대로 Tracy는 지난 10년간의 결함 예측 연구들을 조사한 결과 분류 알고리즘에 따른 성능은 NB, LR이 좋은 결과를 보였다[7]. 그림 2는 해당 논문에서 발췌한 실험 결과이다. 앞서 Lessman의 벤치마킹 결과와는 상이하게 RF의 성능이 LR, NB 등에 비해 낮다. 이것은 예측 모형 구축 방법의 성능에 대해서는 아직까지 일관된 결과가 발견되지 않고 있음을 보여 준다.

### 2.3 결함 예측 모형의 성능 척도 결정

구축된 결함 예측 모형의 성능에 대한 평가가 필요하다. 즉 결함 모형을 적용한 결과와 실제 결함 유무와의 일관성이 요구된다. 통계 추론이나 기계 학습 분야에서는 오차행렬(Confusion matrix)을 기본으로 하는 모형 성능 척도를 사용하고 있다. 표 3은 오차행렬을 나타낸다. 오차행렬은 분류기의 결과를 임계값(threshold value)을 기준으로 4가지 범주로 구분하고 그것을 2×2 행렬로 나타낸 것이다.

분류기는 시스템의 모듈이 fault-prone(fp) 또는 non fault-prone(nfp) 인지를 판별한다. 모형의 분류 결과는 TP(true positive), FN(false negative), TN(true negative), FP(false positive)의 4가지로 나눌 수 있다. TP는 fp로 예측하고 실제로도 fp인 경우이고, TN은 nfp로 예측하고 실제로도 nfp인 경우이다. 반면에 FP는 fp로 예측했지만 실제로는 nfp인 경우이고, FN은 nfp로 예측했지만 실제로는 fp인 경우이다.

오차 행렬을 이용해 계산할 수 있는 성능 척도는 precision, recall 등이 있다. precision은 fp 모듈로 예측한 것 중 실제로 fp인 비율이며, recall은 실제 fp 모듈이 정확하게 예측된 비율을 나타낸다. 두 척도의 값이 높을 수록 예측 모형의 예측력이 높다는 것을 의미한다.

결함 예측 연구에서는 이들 성능 척도를 혼합한 f-measure, AUC(Area Under the roc Curve) 등이 널리 사용된다. 그림 3은 f-measure를 계산하는 식으로, precision과 recall의 조화평균으로 계산할 수 있다.

표 3 오차 행렬

| 예측 / 실제 | Fp(p 〉 임계값) | nfp(p ≤ 임계값) |
|---|---|---|
| Fp | TP | FN |
| Nfp | FP | TN |

표 4 예제 데이터에 기반한 분류기의 성능 평가 예제

| #inst. | prob. | Label | TP | FN | FP | TN | pf | pd |
|---|---|---|---|---|---|---|---|---|
| 1 | 0.9 | T | 1 | 4 | 0 | 5 | 0 | 0.2 |
| 2 | 0.8 | T | 2 | 3 | 0 | 5 | 0 | 0.4 |
| 3 | 0.7 | F | 2 | 3 | 1 | 4 | 0.2 | 0.4 |
| 4 | 0.6 | T | 3 | 2 | 1 | 4 | 0.2 | 0.6 |
| 5 | 0.55 | T | 4 | 1 | 1 | 4 | 0.2 | 0.8 |
| 6 | 0.5 | F | 4 | 1 | 2 | 3 | 0.4 | 0.8 |
| 7 | 0.4 | F | 4 | 1 | 3 | 2 | 0.6 | 0.8 |
| 8 | 0.3 | F | 4 | 1 | 4 | 1 | 0.8 | 0.8 |
| 9 | 0.2 | F | 4 | 1 | 5 | 0 | 1 | 0.8 |
| 10 | 0.1 | T | 5 | 0 | 5 | 0 | 1 | 1 |

AUC는 우선 ROC 곡선을 바탕으로 결정된다. 예를 들어 표 4는 임의의 분류기를 이용한 예제 데이터의 예측 결과이다. 먼저 분류기가 예측한 결함 발생 확률을 내림차순으로 정렬한다. 다음으로 해당 확률을 임계값로 하여 오차행렬을 구한다. 마지막으로 오차행렬로부터 ROC를 그리기 위해 필요한 성능척도인 pd(recall, $\frac{TP}{TP+FN}$)과 pf(false alarm, $\frac{FP}{FP+TN}$)를 계산한다.

그림 4는 예제의 pd와 pf의 계산 결과를 이용하여 ROC 곡선으로 나타낸 것이다. 그림의 X축은 pf, Y축은 pd이다. A 곡선은 예제. 데이터의 예측 결과로부터 그린 ROC 곡선으로 AUC는 0.72이다. B 곡선은 사분

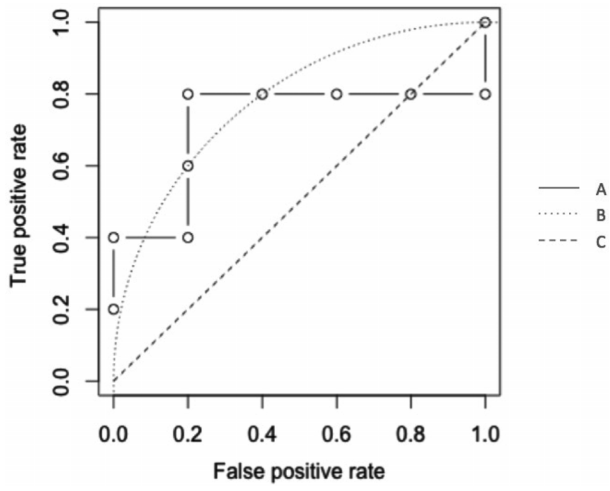


그림 4 ROC 곡선(A: 예제, B: 사분원, C: 대각선)

$$Precision = \frac{TP}{(TP + FP)}, Rccall = \frac{TP}{(TP + FN)}, F-measure = \frac{2 \times Recall \times Precision}{Recall + Precision}$$

그림 3 예측 모형의 성능 척도들(precision, recall, f−measure)

원으로 AUC는 약 0.78이다. C 곡선은 정사각형의 대각선으로 AUC는 0.5이다. 따라서 AUC의 값에 따라 B, A, C 순서로 예측 결과를 비교할 수 있다.

AUC는 알고리즘의 성능을 평가하는 견고한 척도이다. AUC는 분류기의 성능 외의 다른 특성에 영향을 거의 받지 않는다. 대표적으로 AUC는 임계값에 독립적이다. 기계학습 알고리즘의 예측 값은 확률과 같이 연속적인 수로 표현되는 경우가 많다. 예를 들어, LR은 모듈의 분류 결과를 0에서 1 사이의 확률 값으로 계산하고, NB 역시 마찬가지이다. 이런 경우, 연구자는 결함 모듈의 임계값을 정해야 한다. 일반적으로 0.5를 사용하지만, 임계값 선택 기준에 따라 성능에 큰 영향을 미친다[22].

Menzies에 따르면 결함 예측 모형의 평균적인 예측력은 mean(pd, pf) = (71%, 25%)로 알려졌다[9]. 기계 학습 분야의 평균인 mean(pd, pf) = (81%, 20%)과 비교하여 결함을 발견하는 능력은 10% 낮고, 결함을 잘못 발견할 위험은 5% 더 높다. 예측 성능을 높이기 위한 다양한 시도되고 있으나 이 간격을 줄이는 보편적 방법은 아직 발견되지 않고 있다.

## 3. 오픈 소스 시스템을 이용한 결함 예측 모형 구축 예

기존 연구자들의 헌신적인 노력에 의해 일부 소프트웨어의 결함 정보와 메트릭 정보가 공개되어 있다. 표 5는 대표적으로 PROMISE 저장소에서 제공하는 시스템들이다. 임베디드 소프트웨어와 객체지향 소프트웨어가 제공되며 시스템의 종류 또한 다양한 것이 특징이다. 또한, 이들 시스템은 결함 예측 연구에서 빈번히 이용되고 있는 검증된 데이터 셋이다[12].

여기서는 Ant 시스템을 이용해 결함 예측 모형을 구축하는 과정을 설명한다. 표 6은 PROMISE 저장소에 포함된 Ant 시스템의 버전 별 정보이다. Ant 시스템은 총 4개의 버전이 포함되어 있다. 각각은 100~400개의 클래스로 구성되어 있고, 결함 밀도는 10~30%이다.

우리는 다음과 같은 절차에 따라서 Ant 시스템의 결함 예측 모형을 구축하였다.

**1) 예측 요인 결정:** Ant 시스템을 대상으로 ckjm 도구(http://www.spinellis.gr/sw/ckjm/)를 이용해 20개가 넘는 메트릭을 측정하였다[26]. Ant 시스템이 객체 지향 시스템인 것을 고려하여 이 중에서 CK 메트릭와 LOC 메트릭을 주요 요인으로 선택하였다.

**2) 예측 모형 구축:** 결함 예측 시 모듈의 결함 유무를 판별하는 분류 모형을 NB알고리즘을 이용하여 구축하였다. 알고리즘 구축은 R 도구를 이용하였으며 모형의 성능 향상을 위해 전처리 과정에서 log filter를 이용하였다[8].

**3) 성능 척도 결정:** 구축된 모형의 성능을 pd, pf, AUC 등으로 측정하였다. 표 7은 Ant 시스템의 메트릭 정보와 결함 정보를 이용해 구축한 결함 예측 모형의 성능이다. Pd는 0.73~0.80으로 좋은 성능을 보인 반면, pf는 0.25~0.41로 낮은 성능을 보였다. f-measure는 Ant 1.6을 제외하고 대략 0.4 정도를 나타냈다. AUC의 경우는 Ant 1.4를 제외하고 0.8 이상의 좋은 성능을 보였다.

표 7 Ant 시스템의 결함 예측 모형의 성능

| | pd | pf | f-measure | AUC |
|---|---|---|---|---|
| Ant 1.3 | 0.80 | 0.41 | 0.42 | 0.83 |
| Ant 1.4 | 0.73 | 0.55 | 0.39 | 0.66 |
| Ant 1.5 | 0.79 | 0.28 | 0.39 | 0.80 |
| Ant 1.6 | 0.78 | 0.25 | 0.61 | 0.83 |

표 5 PROMISE 저장소에 포함된 데이터 셋

| Type | Dataset Name | Source |
|---|---|---|
| NASA | CM1, KC1, KC2, KC3, MC2, MW1, PC1, PC2, PC3, PC4 | NASA MDP revised by Shepperd[23] |
| SOFTLAB | AR1, AR3, AR4, AR5, AR6 | SOFTLAB[24] |
| APACHE | Ant, Camel, Forrest, Ivy, Log4j, Lucene, Poi, Synapse, Tomcat, Velocity, Xalan, Xerces | Metrics Repository by Marian Jureczko[25] |

표 6 Ant 시스템의 버전별 정보

| System | # instances | # defects | % defects | Description |
|---|---|---|---|---|
| Ant 1.3 | 124 | 20 | 16.0 | Ant는 자동 빌드 시스템이다. Apache 재단의 지원을 받는 오픈 소스 프로젝트로 Java로 개발되었다.<br>Apache Ant (http://ant.apache.org/) |
| Ant 1.4 | 178 | 40 | 22.5 | |
| Ant 1.5 | 293 | 32 | 10.9 | |
| Ant 1.6 | 351 | 92 | 26.2 | |

## 4. 결함 예측 모형의 확인

결함 예측 모형을 개발하면 앞으로 개발될 시스템의 결함을 예측하는데 사용할 수 있다. 좋은 모형은 새로운 데이터에 대해서도 높은 예측 성능을 보여야 한다. 모형의 성능은 새로운 데이터를 이용해서 평가할 수 있다. 이때, 모형 구축 시 사용한 데이터를 이용할 수도 있고 새로운 시스템의 데이터를 사용할 수도 있다.

내적 타당도(internal validation)는 데이터를 일정한 비율로 나눠서 한쪽은 모형을 구축하는데 사용하고 나머지는 모형의 성능을 평가하는데 사용한다. 모형을 구축할 때 사용한 데이터로 다시 모형의 성능을 평가하게 되면 모형의 예측 성능이 해당 데이터에만 유효할 수 있다. 이때 모형이 데이터에 과잉 맞춤(overfitting)되었다고 한다.

이를 극복하기 위한 대표적인 방법으로 교차 유효성 검사(cross validation)가 있다. 이 방법은 데이터를 k개의 동일한 크기의 folds로 나눈 후 k-1개를 모형을 구축하는데 사용하고, 나머지 하나의 fold를 모형의 성능을 평가하는데 사용한다. 보통 k 값은 10을 사용하고 모든 folds를 이용해 모형 구축 과정을 반복한 후 성능을 평균한다. 표 8은 Ant 시스템의 교차 유효성 검사 결과이다. 모형의 성능이 일정량 감소한 것을 확인할 수 있다.

내적 타당도는 동일한 모집단(population)으로부터 샘플링한 데이터를 이용하여 모형을 검증한다는 한계가 있다. 즉, 구축 시 사용한 데이터와 검증 시 사용한 데이터가 유사하기 때문에 모형의 성능이 높게 나오는 문제를 근본적으로 해결할 수 없다.

이에 반해 외적 타당도(external validation)는 모형 구축 시 사용한 시스템의 데이터가 아닌 다른 시스템의 데이터를 이용해서 모형의 성능을 평가한다. 예를 들어,

표 8 Ant 시스템의 교차 유효성 검사 결과: AUC

| System | Origin | Cross Validation |
|---|---|---|
| Ant 1.3 | 0.83 | 0.79 |
| Ant 1.4 | 0.66 | 0.62 |
| Ant 1.5 | 0.80 | 0.76 |
| Ant 1.6 | 0.83 | 0.81 |

표 10 Ant 시스템의 예측 모형의 외적 타당도 검증 결과: AUC

| | Cross Validation | Camel 1.0 | Camel 1.2 | Camel 1.4 |
|---|---|---|---|---|
| Ant 1.3 | 0.79 | 0.69 | 0.55 | 0.67 |
| Ant 1.4 | 0.62 | 0.61 | 0.58 | 0.66 |
| Ant 1.5 | 0.76 | 0.70 | 0.59 | 0.72 |
| Ant 1.6 | 0.81 | 0.69 | 0.60 | 0.71 |

Ant 시스템으로 구축한 모형을 Eclipse 시스템의 결함 예측에 사용하는 것이다. 결함 예측 분야에서는 이를 교차 프로젝트 결함 예측(cross project defect prediction)이라고 한다.

여기서는 Camel 시스템을 이용해 Ant 시스템의 결함 예측 모형을 검증해 본다. 표 9는 PROMISE 저장소에 포함된 Camel 시스템의 버전별 정보이다. 검증에는 3개 버전의 Camel 시스템을 사용하였다. 시스템의 크기는 Camel이 Ant 시스템에 비해 더 크며, 결함 밀도는 10~30% 사이이다.

표 10은 Ant 시스템을 이용해 구축한 예측 모형의 외적 타당도 검증 결과이다. 전체적으로 동일한 데이터를 이용해 측정된 성능에 비해 낮은 성능을 보이는 것을 알 수 있다. AUC 결과가 낮은 Ant 1.4 모형을 제외하고 나머지 모형은 0.1~0.2 사이의 큰 폭의 성능 차를 보인다.

결함 예측 모형의 범용성이 부족한 것은 널리 알려진 사실이다. 최근의 교차 프로젝트 결함 예측 연구는 훈련 데이터(training data)와 검증 데이터(test data) 사이의 유사도를 기계 학습 방법에 기반하여 계산하는 알고리즘의 개발에 집중하고 있다. 현재까지 Nearest Neighbor 필터, Transfer Naive Bayes 등이 대표적으로 사용되고 있다[21,27].

- **Nearest Neighbor 필터:** 훈련 데이터와 검증 데이터 사이의 유사도를 유클리드 거리로 측정하고, k-Nearest Neighbor(k-NN)방법을 이용하여 검증 데이터에 가장 가까운 학습 데이터만을 학습에 사용한다.

- **Transfer Naive Bayes:** 데이터간 유사도를 계산하기 위해 데이터 중력을 계산한 후 훈련 데이터의 가중

표 9 Camel 시스템의 버전별 정보

| System | # instances | # defects | % defects | Description |
|---|---|---|---|---|
| Camel 1.0 | 339 | 13 | 3.8 | Camel은 엔터프라이즈 통합 패턴에 기반한 오픈 소스 통합 프레임워크이다(http://camel.apache.org/). |
| Camel 1.2 | 608 | 216 | 35.5 | |
| Camel 1.4 | 872 | 145 | 16.6 | |

치로 부여하였다. 그런 후 가중치가 부여된 훈련 데이터를 Naive Bayes 방법을 이용해 모형을 구축한다.

## 5. 결 론

결함 예측은 검증(verification), 테스팅 등과 같이 소프트웨어의 결함을 발견하는 활동 중 하나이다. 결함 발견을 위해 형식 검증이 수학 기반의 정형 기법(formal methods)을 이용하고, 테스팅이 테스트 케이스를 생성하는 것처럼, 결함 예측은 소프트웨어의 구조적 속성과 기계 학습 분야의 학습 알고리즘을 이용한다.

결함 예측은 결함을 찾아내는 이론적 연구뿐만 아니라 실무에서도 적용될 수 있는 가능성이 높은 기술이다. NASA의 Independent Verification and Validation (IV&V) 조직은 최고 수준의 안전성을 보이는 소프트웨어를 개발하는 조직 중 하나이다. 여기서 위성 관련 소프트웨어의 결함 예측 모형 개발 프로젝트를 통해 개발 방법을 획득할 수 있었고, 이 기술에 기반한 Predictive 제품이 상용 도구로 개발되었다[9]. 이 기술을 Turkey의 백색가전 회사에 적용한 결과 40%의 노력만으로 70%의 결함을 발견할 수 있었다고 보고하였다.

연구자의 관점에서도 결함 예측은 큰 전환점에 놓여 있다. 예전의 결함 예측 연구가 새로운 메트릭, 알고리즘, 모듈이나 파일 등의 대상에 초점이 맞춰졌다면, 세밀한 입도(finer granularity), 노이즈 제어(noise handling) 등으로 연구의 중심이 옮겨가고 있다[28].

**세밀한 입도[29]:** 대표적인 정적 결함 발견 도구인 FindBug, JLint, PMD를 이용하면 라인 단위의 결함을 예측할 수 있다. 아직 결함 예측의 정확도는 높지 않지만, 결함이 존재하는 이유를 비교적 잘 설명해 준다는 점에서 높은 관심을 받고 있다.

**노이즈 제어[30]:** 버전 컨트롤 시스템이나 결함 관리 시스템으로부터 결함 데이터를 수집 시 노이즈가 발생한다. 마이닝을 할 때 일반적으로 키워드 기반 검색이나 버그 리포트의 링크 기반으로 결함 유무와 결함 있는 모듈을 찾아낸다. 이 방법은 개발자가 특정 키워드를 누락하거나 링크 정보를 제공하지 않는 경우에 문제가 발생한다. 결함 예측 모형은 노이즈에 일정 부분 내성이 있으나 이를 제거해 줌으로써 모형의 정확성을 높일 수 있다.

이 밖에도 해당 모듈이 결함으로 예측된 이유를 설명하는 모형, 교차 프로젝트 결함 예측, 비용 효과성 척도 등이 중요한 연구 분야로 인식되고 있다. 구체적으로 결함 예측 프레임워크 개발, 새로운 교차 프로젝트 결함 예측 알고리즘 개발, 품질에 영향을 미치는 개발자 메트릭 개발 등이 있다.

## 참고문헌


[ 1 ] V. R. Basili, L. C. Briand, and W. L. Melo, “A validation of object-oriented design metrics as quality indicators,” Software Engineering, IEEE Transactions on, vol. 22, no. 10, pp. 751-761, 1996.

[ 2 ] L. C. Briand, W. L. Melo, and J. Wust, “Assessing the applicability of fault-proneness models across object-oriented software projects,” Software Engineering, IEEE Transactions on, vol. 28, no. 7, pp. 706-720, 2002.

[ 3 ] T. Gyimothy, R. Ferenc, and I. Siket, “Empirical validation of object-oriented metrics on open source software for fault prediction,” Software Engineering, IEEE Transactions on, vol. 31, no. 10, pp. 897-910, 2005.

[ 4 ] Z. Yuming, and L. Hareton, “Empirical Analysis of Object-Oriented Design Metrics for Predicting High and Low Severity Faults,” Software Engineering, IEEE Transactions on, vol. 32, no. 10, pp. 771-789, 2006.

[ 5 ] H. M. Olague et al., “Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes,” Software Engineering, IEEE Transactions on, vol. 33, no. 6, pp. 402-419, 2007.

[ 6 ] S. Lessmann et al., “Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings,” Software Engineering, IEEE Transactions on, vol. 34, no. 4, pp. 485-496, 2008.

[ 7 ] T. Hall et al., “A Systematic Literature Review on Fault Prediction Performance in Software Engineering,” Software Engineering, IEEE Transactions on, vol. 38, no. 6, pp. 1276-1304, 2012.

[ 8 ] T. Menzies, J. Greenwald, and A. Frank, “Data Mining Static Code Attributes to Learn Defect Predictors,” Software Engineering, IEEE Transactions on, vol. 33, no. 1, pp. 2-13, 2007.

[ 9 ] T. Menzies et al., “Defect prediction from static code features: current results, limitations, new approaches,” Automated Software Engg., vol. 17, no. 4, pp. 375-407, 2010.

[10] S. N. Cant, D. R. Jeffery, and B. Henderson-Sellers, “A conceptual model of cognitive complexity of elements of the programming process,” Information and Software

Technology, vol. 37, no. 7, pp. 351-362, 9, 1995.
[11] N. E. Fenton, and S. L. Pfleeger, Software metrics: a rigorous and practical approach: PWS Publishing Co., 1998.
[12] C. Catal, and B. Diri, "A systematic review of software fault prediction studies," Expert Systems with Applications, vol. 36, no. 4, pp. 7346-7354, 5, 2009.
[13] I. H. Witten, and E. Frank, Data Mining: Practical machine learning tools and techniques: Morgan Kaufmann, 2011.
[14] S. R. Chidamber, and C. F. Kemerer, "A metrics suite for object oriented design," Software Engineering, IEEE Transactions on, vol. 20, no. 6, pp. 476-493, 1994.
[15] M. Lorenz, and J. Kidd, Object-oriented software metrics: a practical guide: Prentice-Hall, Inc., 1994.
[16] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, Applied logistic regression: Wiley. com, 2013.
[17] K. Gao et al., "Choosing software metrics for defect prediction: an investigation on feature selection techniques," Software: Practice and Experience, vol. 41, no. 5, pp. 579-606, 2011.
[18] S. J. Russell, and P. Norvig, Artificial Intelligence: A Modern Approach: Pearson Education, 2009.
[19] E. Frank et al., "Weka," Data Mining and Knowledge Discovery Handbook, O. Maimon and L. Rokach, eds., pp. 1305-1314: Springer US, 2005.
[20] R. Ihaka, and R. Gentleman, "R: A Language for Data Analysis and Graphics," Journal of Computational and Graphical Statistics, vol. 5, no. 3, pp. 299-314, 1996/09/01, 1996.
[21] B. Turhan et al., "On the relative value of cross-company and within-company data for defect prediction," Empirical Softw. Engg., vol. 14, no. 5, pp. 540-578, 2009.
[22] E. A. Freeman, and G. G. Moisen, "A comparison of the performance of threshold criteria for binary classification in terms of predicted prevalence and kappa," Ecological Modelling, vol. 217, no. 1, pp. 48-58, 2008.
[23] M. Shepperd et al., "Data Quality: Some Comments on the NASA Software Defect Datasets," Software Engineering, IEEE Transactions on, vol. 39, no. 9, pp. 1208-1215, 2013.
[24] T. Menzies et al., "The promise repository of empirical software engineering data," Available: promisedata. googlecode. com, 2012.
[25] M. Jureczko, and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering, Timioara, Romania, 2010, pp. 1-10.
[26] D. Spinellis, "Tool writing: a forgotten art?(software tools)," Software, IEEE, vol. 22, no. 4, pp. 9-11, 2005.
[27] Y. Ma et al., "Transfer learning for cross-company software defect prediction," Inf. Softw. Technol., vol. 54, no. 3, pp. 248-256, 2012.
[28] K. Sung. "Defect, Defect, Defect," http://promisedata.org/2012/.
[29] F. Thung et al., "To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools," in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, Essen, Germany, 2012, pp. 50-59.
[30] S. Kim et al., "Dealing with noise in defect prediction," in Proceedings of the 33rd International Conference on Software Engineering, Waikiki, Honolulu, HI, USA, 2011, pp. 481-490.

## 약 력

**김 태 연**

2007 부산대 정보컴퓨터공학 학사.
2009 부산대 컴퓨터공학 석사.
2010~현재 부산대학교 컴퓨터공학 박사과정.
관심분야 : 소프트웨어 결함 예측, 소프트웨어 테스팅, 소프트웨어 메트릭, 안드로이드 플랫폼
E-mail: tykim@pusan.ac.kr

**채 흥 석**

1994 서울대 원자핵공학 학사
1996 한국과학기술원 전산학 석사
2000 한국과학기술원 전산학 박사
2000~2003 (주)동양시스템즈 기술연구소 선임연구원
2003~2004 한국과학기술원 전산학과 초빙교수
2004~현재 부산대학교 컴퓨터공학과 부교수
관심분야: 객체지향 방법론, 소프트웨어 테스팅, 소프트웨어 메트릭, 소프트웨어 유지보수, 미들웨어 설계, 프로덕트라인 공학
E-mail: hschae@pusan.ac.kr