한국컴퓨터정보학회 동계학술대회 논문집 제25권 제1호 (2017. 1)

베이지안 확률을 적용한 기계학습 기반 다중 결함 위치 식별 기법

송지현[O], 김정호*, 이은석*
*[O]성균관대학교 소프트웨어대학

e-mail: sjh0362@skku.edu[O], {jeonghodot, leees}@skku.edu*

Machine Learning-based Multiple Fault Localization with Bayesian Probability

Jihyoun Song[O], Jeongho Kim*, Eunseok Lee*
*[O]College of Software, Sungkyunkwan University

● 요 약 ●

소프트웨어의 개발과정 중 결함을 제거하는 작업인 디버깅을 위해서는 가장 먼저 그 결함의 정확한 위치를 찾아야한다. 이 작업은 많은 시간이 소요되며, 이 시간을 단축시키기 위한 결함 위치 식별 기법들이 소개되었다. 많은 기법들 중 프로그램 커버리지 정보를 학습하여 규칙을 분석하는 인공신경망 기반 선행 연구가 있다. 이를 기반으로 본 논문에서는 문장들 간의 관계를 추가적으로 파악하여 학습 데이터로 사용하는 기법을 제안한다. 특정 문장이 항상 지나는 테스트케이스들 중 나머지 다른 문장들이 지나는 테스트케이스의 비율을 통해 문장들 간의 관계를 나타낸다. 해당 비율을 계산하기 위해 조건부 확률인 베이지안 확률을 사용한다. 베이지안 확률을 통해 얻은 문장들의 관계에 따라 인공신경망 내에서 의심도를 결정하는 웨이트(weight)가 기존 기법과는 다르게 학습된다. 이 차이는 문장들의 의심도를 조정하며, 결과적으로 다중 결함 위치 식별의 정확도를 향상시킨다. 본 논문에서 제안한 기법을 이용하여 실험한 결과, Tarantula 대비 평균 39.8%, 기존 역전파 인공신경망(BPNN) 기반 기법 대비 평균 60.5%의 정확도 향상이 있었음을 확인할 수 있다.

키워드: 다중 결함(multiple fault), 베이지안 확률(bayesian probability), 기계학습(machine learning)

I. Introduction

소프트웨어의 개발과정 중에는 결함의 존재유무를 확인하고 그 위치를 찾아 제거하는 테스트, 결함 위치 식별, 디버깅 작업이 필수적이다. 테스트를 위한 연구는 이미 과거에 많이 존재한다. 하지만 기술적 한계로 인해 결함 위치 식별 연구는 매우 부족한 실정이다. 또한 디버깅 작업은 결함 위치를 정확하게 식별하는 것이 선행되어야 하므로 매우 중요한 작업이다[1]. 결함 위치 식별은 일반적으로 개발자가 프로그램 실행을 추적해서 찾아야하기 때문에 많은 시간이 소요된다. 개발자들의 수고를 덜어주고, 디버깅 시간을 단축시키기 위한 결함 위치 식별 기법들이 연구되어 왔다.

W.E.Wong et al.이 2016년에 발표한 논문에서는 소프트웨어 결함 위치 식별 기법들을 slice-based, spectra-based, program state-based, machine learning-based, model-based, data mining-based method로 분류하여 정리한다[2].

본 논문에서 새롭게 제안하는 기법은 기계학습 기반 기법으로 분류되며, 역전파(Back-Propagation, BP) 신경망과 베이지안 확률을 적용한다.

인공신경망 기반의 모델은 다른 모델과 비교하여 여러 장점을 가진다. 복잡한 데이터를 학습하여 규칙을 분석하는 능력을 가진다.

또한, 학습된 정보는 분산 형태로 저장되기 때문에 신경망의 일부분이 제거되는 오류에 영향을 적게 받는다. 이러한 특성들로 소프트웨어 위험 분석, 소프트웨어 비용 추정 등의 소프트웨어 공학 분야에 성공적으로 적용되어 왔고, 소프트웨어 결함 위치 식별 연구에도 적용이 되고 있다[3].

본 논문에서는 인공신경망과 관련된 선행연구에서 사용한 기법을 기반으로 문장들 간의 관계를 추가적으로 고려하는 기법을 제안한다. 특정 문장이 항상 지나는 테스트케이스들 중 다른 문장들이 지나는 테스트케이스의 비율을 통해 문장들 간의 관계를 나타낸다. 이 비율을 계산하기 위해 조건부 확률인 베이지안 확률을 사용한다. 문장들 간의 확률적 관계를 파악하여 의심도를 조정함으로써 다중 결함 위치 식별의 정확도를 향상시킨다.

앞으로 2장에서는 본 논문을 위해 참고한 관련 연구들 중 가장 연관성이 높은 논문을 소개하며, 3장에서는 제안기법에 대해 설명한다. 마지막 4장에서는 제안 기법에 대한 실험 결과에 대해 결론을 내린다.

II. Preliminaries

본 논문에서 제안하는 기법은 인공신경망과 관련된 선행연구에서 사용한 기법을 기반으로 디자인한다.

W.E.Wong et al.의 Back-Propagation(BP, 역전파) neural network-based method[3]을 시작으로 관련 연구가 진행되어 왔다. 프로그램 커버리지 정보를 학습한 인공신경망을 통해 가상의 각 문장만을 지나는 프로그램 커버리지 정보의 결과 값을 예측하고, 이를 의심도로 활용하는 것이 인공신경망을 활용한 결함 위치 식별 기법의 기본이다.

그 다음으로 Radial Basis Function(RBF, 방사 기저 함수) neural network-based method[1]가 소개되었다. 이 두 인공신경망 기반 기법은 본 논문에서 제안하는 기법의 기본적인 토대가 되고 있다.

위의 두 연구 모두 다중 결함에 대해 다루고 있지만, 본문 내용에서 그 비중은 크지 않다. 또한, 프로그램 커버리지 정보에서 각 문장이 갖는 규칙만을 독립적으로 사용하여 각 문장의 의심도를 계산한다. 본 논문에서는 문장들의 관계를 추가적으로 참고한다. 인공신경망 내에서 문장들의 관계에 따라 의심도를 결정하는 웨이트를 조정하여 정확도를 향상시킨다.

III. The Proposed Scheme

1. 제안 기법

1.1 개요

기존 인공신경망 기반의 결함 위치 식별 기법을 보완하기 위해 프로그램 커버리지 정보와 문장들 간의 관계 정보를 학습한다. 그 다음으로 테스트 데이터를 사용하여 프로그램을 이루는 각 문장들의 의심도를 구한다.

특정 문장이 항상 지나는 테스트케이스들 중 나머지 다른 문장들이 지나는 테스트케이스의 비율을 계산하기 위해 베이지안 확률을 사용한다. control flow graph나 date flow graph와 같은 부가적인 정보 없이 단순한 계산을 통해 얻을 수 있는 문장들 간의 관계를 의미한다. 이를 학습 데이터로 사용하면, 문장들의 관계에 따라 인공신경망 내에서 의심도를 결정하는 웨이트가 기존 기법과는 다르게 학습된다. 결과적으로 의심도의 값이 변하고, 이 차이가 다중 결함 위치 식별의 정확도를 향상시킨다.

1.2 학습 데이터

프로그램 내 문장이 10개 존재한다고 가정하고, 이를 Sn(S1~S10)으로 나타낸다. 각 문장이 테스트케이스를 지나는 경우에는 1로, 그렇지 않은 경우에는 0으로 표시하여 커버리지 정보로 활용한다. 해당 테스트케이스의 실행 결과가 실패한 경우에는 1로, 성공한 경우에는 0으로 나타낸다(Table 1).

Table 1. Coverage information and execution results

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	r
T1	1	1	1	1	0	1	0	0	1	1	0
T2	1	0	0	1	1	0	1	0	0	1	0
T3	1	1	1	0	0	1	0	0	1	1	0
T4	1	0	1	0	0	1	1	0	1	1	0
T5	1	1	1	0	1	0	0	1	1	0	1
T6	1	1	1	1	0	0	0	1	1	1	1
T7	1	0	1	1	1	1	1	1	0	1	1

Table 1에서 실패한 테스트케이스 정보만을 가지고 각 문장들 간의 관계를 구한다. 베이지안 확률을 이용하여 Sn이 항상 지나는 테스트케이스 내에서 다른 문장들이 지나는 확률을 계산한다. P(a)를 a에 대한 확률, N(b)를 b의 개수라고 하면, Sn과 다른 문장 Sm의 관계를 계산하는 식은 다음과 같다. Sn=1은 Table 1에서와 같이 Sn이 테스트케이스를 지나간다는 것을 의미한다.

$$P(Sm=1|Sn=1)=\frac{N(Sm=1,Sn=1)}{N(Sn=1)} \qquad (1)$$

Formula 1을 사용하여 실패한 테스트케이스 커버리지 정보에서 문장들 간의 관계를 계산한다(Table 2).

Table 2. Dependency between statements

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	r
T1	1	0.7	1	0.7	0.7	0.3	0.3	1	0.7	0.7	1
T2	1	1	1	0.5	0.5	0	0	1	1	0.5	1
T3	1	0.7	1	0.7	0.7	0.3	0.3	1	0.7	0.7	1
T4	1	0.5	1	1	0.5	0.5	0.5	1	0.5	1	1
T5	1	0.5	1	0.5	1	0.5	0.5	1	0.5	0.5	1
T6	1	0	1	1	1	1	1	1	0	1	1
T7	1	0	1	1	1	1	1	1	0	1	1
T8	1	0.7	1	0.7	0.7	0.3	0.3	1	0.7	0.7	1
T9	1	1	1	0.5	0.5	0	0	1	1	0.5	1
T10	1	0.5	1	1	0.5	0.5	0.5	1	0.5	1	1

1.3 테스트 데이터

선행 연구에서와 마찬가지로 가상의 테스트 데이터인 직교행렬(orthogonal matrix) 형태의 테스트 데이터를 사용한다(Table 3). 이를 통해 각 문장의 웨이트만을 고려하여 의심도를 계산한다.

Table 3. Virtual test data

	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10
T1	1	0	0	0	0	0	0	0	0	0
T2	0	1	0	0	0	0	0	0	0	0
T3	0	0	1	0	0	0	0	0	0	0
T4	0	0	0	1	0	0	0	0	0	0
T5	0	0	0	0	1	0	0	0	0	0
T6	0	0	0	0	0	1	0	0	0	0
T7	0	0	0	0	0	0	1	0	0	0
T8	0	0	0	0	0	0	0	1	0	0
T9	0	0	0	0	0	0	0	0	1	0
T10	0	0	0	0	0	0	0	0	0	1

2. 실험

2.1 실험 개요

Step 1 Training process : Table 1의 프로그램 커버리지 정보와 Table 2의 코드 간의 관계를 학습 데이터로 사용하여 인공신경망을 학습시킨다.

Step 2 Testing process : 학습된 인공신경망에 Table 3의 테스트 데이터를 넣어 결과 값을 얻는다.

Step 3 : 결과 값을 그 문장의 의심도로 사용한다. 모든 문장들의 의심도 값에 따라 순위를 매겨 결함을 가질 확률이 높은 문장들을 알 수 있다.

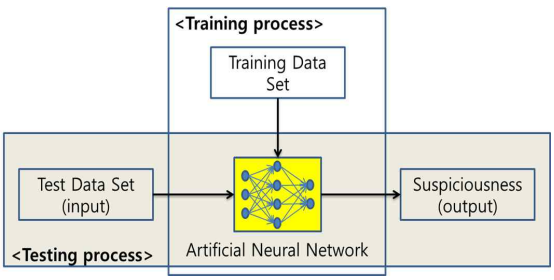

Fig. 1. Overview of the experiment

2.2 실험 내용

printtokens2 프로그램을 사용하고, 결함의 총 개수는 2개로 설정하여 진행한다. 결함이 없는 원본 코드 파일을 기준으로 수정된 문장을 결함이라고 한다. 두 개의 결함 중 가장 빨리 찾은 결함의 순위가 아닌 두 결함 모두를 찾았을 때의 의심도 순위를 비교한다.

실험에서 사용한 printtokens2는 총 10개의 결함 버전을 가진다. 버전 1, 2, 3은 원본 프로그램에서 특정 문장을 삭제해서 결함을 만든 것이므로 본 실험 대상에서 제외한다. 버전 4~10을 사용하여 다중 결함을 가진 결함 버전을 새롭게 만든다. 버전 7과 9는 같은 문장에서 변형이 있기 때문에 이 조합 역시 다중 결함 버전에서 제외한다.

20개의 다중 결함 버전의 프로그램 커버리지 정보를 얻은 결과, 버전 조합 {4, 7}, {4, 9}, {5, 10}은 두 개의 결함 중 하나가 테스트케이스를 전혀 지나지 않는다. 이는 해당 결함을 찾을 수 없다는 것을 의미하기 때문에 실험에서 제외한다. 버전 조합 {6, 10}은 모든 테스트케이스가 실패하며, 커버리지 정보 역시 대부분의 문장이 테스트케이스를 전혀 지나지 않는 무의미한 정보이기 때문에 실험에서 제외한다. 즉, 20가지의 다중 결함 버전 중 총 16개의 다중 결함 버전으로 실험을 진행한다.

2.3 실험 결과

제안 기법의 성능을 확인하기 위하여 일반적으로 사용되고 있는 Tarantula 기법과 비교한다. 또한 제안 기법이 BPNN 기반 기법을 확장한 것이므로 BPNN 기반 기법과도 비교한다.

테스트 데이터의 결과 값을 그 문장의 의심도로 사용하며, 결과 값이 음수인 경우에는 절대값을 사용한다. 의심도가 0과 가까울수록 결함을 가지는 확률(의심도)가 낮다는 것을 의미한다. Table 4에서 각 숫자는 마지막 결함을 찾았을 때의 의심도 순위를 의미한다. 예를 들어, 버전 조합 {4, 5}에서 제안 기법을 사용하면, 전체 문장 중 158번째에서 모든 결함을 찾을 수 있다. Tarantula는 공동 순위가 다수 존재하기 때문에 공동 순위 중 결함을 제일 먼저 찾는 경우를 의미하는 Best-case와 제일 마지막에 찾는 경우를 의미하는 Worst-case로 나누어서 비교한다.

Table 4는 구체적인 실험결과를 나타낸다. 음영 처리한 부분은 세 개의 기법 중 가장 정확도가 높은 기법이다. 빗살무늬 부분은 다른 기법과의 정확도가 같은 부분이다.

Table 4. Comparison with other methods

버전 조합	4,5	4,6	4,8	4,10	5,6	5,7	5,8	5,9
Taran_Best	79	84	84	78	4	39	172	161
Taran_Worst	88	93	93	87	5	39	172	161
BPNN	137	180	158	137	179	178	138	24
제안 기법	158	180	12	141	5	17	151	13

버전 조합	6,7	6,8	6,9	7,8	7,10	8,9	8,10	9,10
Taran_Best	74	73	176	78	39	78	171	160
Taran_Worst	74	73	176	78	39	78	171	160
BPNN	20	180	180	180	160	81	162	128
제안 기법	172	48	170	68	163	180	135	180

IV. Conclusions

1. 결론

16개의 버전 조합 중 8개에서 제안 기법의 정확도가 약 39.8% 향상된다. 버전 조합 {5, 6}은 Tarantula Best-case와는 순위 하나 차이, Worst-case와는 차이가 없는 것을 감안하면, 총 9개의 버전

조합에서 제안 기법의 정확도가 Tarantula와 비슷하거나 향상된다.

BPNN 기반 기법과 비교하였을 때, 16개의 버전 조합 중 9개에서 제안 기법의 정확도가 약 60.5% 향상된다. 정확도가 같은 경우를 포함하면, 총 10개의 버전 조합에서 제안 기법의 정확도는 BPNN 기반 기법과 같거나 향상된다.

기존 Tarantula에서는 공동 순위로 인해 구분하지 못했던 문장들의 의심도를 인공신경망을 사용함으로써 세분화할 수 있다. 그리고 두 개의 결함을 모두 찾을 때까지 프로그래머가 확인해야 할 문장의 개수가 Tarantula와 BPNN에 비해 줄어들었다.

2. 향후 연구

본 논문에서는 printtokens2 프로그램만을 사용하여 실험을 진행하였다. 다른 프로그램에서도 유사한 결과를 얻을 수 있는지 확인할 필요가 있다. 또한, 현실에서 개발되고 있는 프로그램의 경우에는 결함의 수가 많으며, 실질적으로 결함의 개수를 알기는 어렵다. 결함의 개수를 늘려가며 실험을 진행해 볼 필요가 있다.

Acknowledgment

본 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임
(No. 2016R1D1A1B03934610).

References

[1] W.E.Wong, V.Debroy, R.Golden, X.Xu, B.Thuraisingham, “Effective software fault localization using an RBF neural network,” IEEE Transactions on Reliability, vol. 61, no. 1, pp. 149-169, 2012.

[2] W.E.Wong, R.Gao, Y.Li, R.Abreu, F.Wotawa, “A survey on software fault localization,” IEEE Transactions on Software Engineering, vol. 42, no. 8, pp. 707-740, 2016.

[3] W.E.Wong, Y.Qi, “BP neural network-based effective fault localization,” International Journal of Software Engineering and Knowledge Engineering, vol. 19, no. 4, pp. 573-597, 2009.

[4] W.Zheng, D.Hu, J.Wang, “Fault localization analysis based on deep neural network,” Mathematical Problems in Engineering, vol. 2016, pp. 1-11, 2016.

[5] L.C.Briand, Y.Labiche, X.Liu, “Using machine learning to support debugging with Tarantula,” The 18th IEEE International Symposium on Software Reliability, 2007.

[6] M.Hall, E.Frank, G.Holmes, B.Pfahringer, P.Reutemann, I.H.Witten, The WEKA Data Mining Software: An Update, SIGKDD Explorations, Volume 11, Issue 1, 2009.

[7] M.Feng, R.Gupta, “Learning Universal Probabilistic Models for Fault Localization,” Proceedings of the 9th ACM SIGPLAN-SIGSOFT Workshop on Program analysis for software tools and engineering(PASTE’ 10), pp. 81-88, Canada, 2010.

[8] D.Gong, X.Su, T.Wang, P.Ma, W.Yu, “State Dependency Probabilistic Model for Fault Localization,” Information and Software Technology, vol. 57, pp. 430-445, 2015.

[9] M.White, C.Vendome, M.L.Vasquez, D.Poshyvanyk, “Toward Deep Learning Software Repositories,” Proceedings of the 12th Working Conference on Mining Software Repositories, pp. 334-345, Italy, 2015.