G-29

アンサンブル学習ソフトウェア欠陥予測に対するオンライン最適化におけるテスト戦略の影響分析の試み

Analyzing Influence of Test Strategies to Online Optimization on Ensemble-Learning Software Defect Prediction

高木 亮多† 浜元 健成† 角田 雅照† 戸田 航史‡ 中才 恵太朗*
Ryota Takaki Kensei Hamamoto Masateru Tsunoda Koji Toda Keitaro Nakasai

1. はじめに

ソフトウェアの品質を保証するために，テストは欠くことのできない活動である．ただし，人的資源や開発期間の制約のため，すべてのソフトウェアモジュールを重点的にテストすることは難しい．ソフトウェア欠陥予測はこの問題を解決するためのアプローチであり，機械学習モデルを用いて欠陥が含まれるモジュールを特定し，それらを重点的にテストする[19].

近年，ソフトウェア欠陥予測の精度を高めるために，アンサンブル学習の研究が多く行われている[12]．バギング，ブースティング，スタッキングなど，これまで多くのアンサンブル学習の方法が提案されているが，それらの予測精度は学習データに依存する[4]．例えば，あるソフトウェアプロジェクトで高い予測精度となったアンサンブル学習方法が，他のプロジェクトでも高い精度となるとは限らない．これはソフトウェア欠陥予測の外的妥当性問題[3]とみなされ，最も高い精度となるアンサンブル学習方法を選択することは容易ではないといえる．

ソフトウェア欠陥予測における，アンサンブル学習方法の選択を支援するため，本稿ではバンディットアルゴリズムに基づくオンライン最適化方法[7]を適用することを提案する．バンディットアルゴリズムの説明にはスロットマシンの比喩表現がしばしば用いられる．プレイヤーが 100 コイン所持しており，スロットマシンが複数存在するとき，プレイヤーの報酬を最大限にすることを想定する．バンディットアルゴリズムでは，プレイヤーは 1 コインずつ，報酬の期待値が最も高いと思われるマシンに賭けることを繰り返すことにより，報酬を最大限にする．本稿ではアンサンブル学習方法をスロットマシン（アームとも呼ばれる），スロットマシンに賭けることをモジュールのテストとみなす．テスト結果とアンサンブル学習による予測結果が一致している場合，コイン（報酬）が得られるとみなす．

バンディットアルゴリズムの適用を前提とする場合，「規模の小さいモジュールから順次テストする」などのテスト戦略が，バンディットアルゴリズムの性能に影響する可能性がある．例えば規模の小さいモジュールでは各アンサンブル学習方法の予測精度が同じで，規模の大きなモジュールでは予測精度が異なるとする．この場合，最も精度の高いアンサンブル学習方法の特定できる時期が，テストの終盤になり，バンディットアルゴリズムの効果が小さなくなる可能性がある．バンディットアルゴリズムによりソフトウェア欠陥予測方法を最適化していた従来研究[1][7][16]では，テスト順序の影響は考慮されていなかった．

† 近畿大学，Kindai University
‡ 福岡工業大学，Fukuoka Institute of Technology
* 大阪公立大学工業高等専門学校，Osaka Metropolitan University College of Technology.

2. バンディットアルゴリズム

2.1 概要

図 1 に示すように，予測モデルはアンサンブル学習により構築され，テスト対象の各モジュールの欠陥はバンディットアルゴリズム適用前に，あらかじめ予測されているとする．各モデルの予測結果をアームとみなす．モジュールは順次テストされ，バンディットアルゴリズムはテスト結果に基づいてアームを選択する．図 1[16]はこの手順を示したものである．手順において各アームの予測精度（AUC）を平均報酬とみなす．以下に詳細な手順を述べる．

1. 平均報酬（AUC）に基づきアームを選択する
2. 手順 1 で選択されたアームの予測結果に基づき，モジュールをテストする
3. 手順 2 でのテスト結果と予測結果を比較し，各アームの平均報酬を再計算する
4. 手順 1 に戻る

ソフトウェアテスト中に上記手順を繰り返し行う．初期状態ではすべてのアームの平均報酬は 0 であるため，最初のみアームがランダムで選択される．手順 2 において予測結果が「欠陥あり」の場合，開発者は多数のテストケースを作成し，「欠陥」なしの場合，少数のテストケースを作成する．これによりテストのためのコストを削減する[11][19].

2.2 欠陥見逃し

タイプ 1：あるモジュールの予測結果が陰性（欠陥なし）の場合，コストを削減するため[19]，開発者は少数のテストケースを作成する[11]．その結果，実際には欠陥が含まれるモジュールにも関わらず欠陥見逃しが発生し，「欠陥

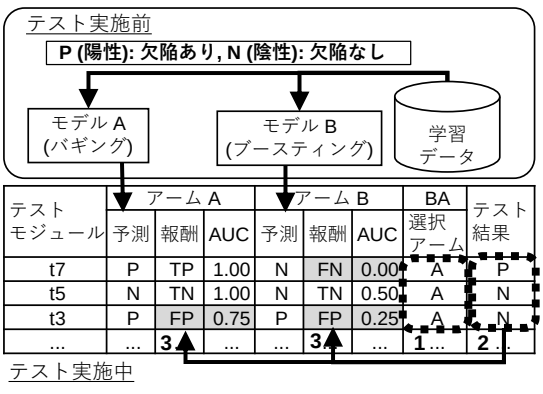

図 1　バンディットアルゴリズムによる欠陥予測

テスト モジュール	アーム A			アーム B			BA	テスト 結果	後工程 結果
	予測	報酬	AUC	予測	報酬	AUC	選択 アーム		
t11	P	**FP**	0.00	N	**TN**	1.00	B	**N**	P
t19	P	**FP**	0.00	N	**TN**	1.00	B	**N**	P
t15	N	TN	0.33	N	TN	1.00	B	N	N
t13	P	**FP**	0.20	P	**FP**	0.80	B	**N**	P
…	…	…	…	…	…	…	…	…	…

タイプ 1: 多くの場合，発生　　**タイプ 2**: 約20%の確率で発生

図 2　タイプ 1，タイプ 2 の欠陥見逃し

テスト モジュール	LOC	テスト 工数	テスト 工数 比率	アーム B		テスト 結果	後工程 結果	タイプ 1 発生確率
				予測	報酬			
t56	1000	10.00	100%	P	TP	P	P	0%
t55	1000	2.50	**25%**	N	**TN**	**N**	P	**75%**

図 3　テスト工数比率とタイプ 1 発生確率の関係

なし」とみなされる可能性がある[16]．この種類の見逃しをタイプ 1 と呼ぶ[5].

図 2 において「テスト結果」の列はテスト工程における結果を示し，「後工程結果」はテスト後（ソフトウェアリリース後など）の，実際の結果を示す．ここでは結果が陰性の場合，テストケースが少数のため，高い確率で欠陥が見逃されると想定している．

図 2 では，最初にアーム B がランダムに選ばれ．モジュール t11 と t19 におけるアーム B の報酬は真陰性とみなされる．ただし「後工程結果」に基づくと，正しい報酬は偽陰性となる．同様にテスト結果に基づくと，アーム A の報酬は誤って偽陽性とされる．その結果，AUC が不正確となり，予測精度の低いアーム B が誤って選択される[16].

タイプ 2：予測結果が陽性（欠陥あり）の場合でも，テスト工程において欠陥が見逃される可能性がある[1][16]．この種類の見逃しタイプ 2 と呼ぶ[5]．これは，予測結果に基づきテストケースが多数生成されていても，起こり得る欠陥見逃しである．図 2 においてモジュール t13 がタイプ 2 の例である．企業横断的に収集された大規模データによると，約 17%の欠陥が結合テストにおいて見逃されている[8].

2.3　テスト工数

定義：一般に，テスト工数はモジュールの規模にしたがって増加する[10][13]．また前節で説明したように，テスト工数（作成するテストケース数）は予測結果に従って変化する．本稿では，モジュールの予測結果が陰性の場合と陽性の場合における，テスト工数比率 *ratio* を用いて，テスト工数を以下のように定義した．

$$effort = \begin{cases} size \cdot c & \text{if } pred = 1, \\ size \cdot c \cdot ratio & \text{if } pred = 0 \end{cases} \quad (1)$$

上記の式において，size は規模などのコード行数，*c* は定数，*pred* は予測結果を示す（1: 陽性，0: 陰性）．例えば図 3 では $c = 0.01$，$ratio = 0.1$ としている．

タイプ 1 発生確率: テスト工数比率が 1.0 の場合，陰性予測となったモジュールと陽性予測となったモジュールの工数は同じとなり，タイプ 1 の発生確率も 0%となる． テスト工数比率が小さくなると，タイプ 1 の発生確率は逆に増加する．本稿ではテスト工数比率と発生確率に線形の関係があると想定し，前者が *n*%の場合，後者は 1 - *n*%となるとした．例えば図 3 に示すように，モジュール t55 において，テスト工数比率が 25%の場合，タイプ 1 の発生確率は 75%とした．

2.4　テスト戦略

テスト戦略として主要なもののひとつとして，リスクベースドテスト**エラー! 参照元が見つかりません。**がある．これはテスト時に，リスクの高いモジュール，例えば複雑な（規模の大きい）モジュールや，欠陥を含む可能性の高いモジュールなどから，優先的にテストする方法である．以降では，リスクベースドテストに基づき，規模の大きいモジュール順にテストする戦略を LF と呼ぶ.

規模の大きなモジュールでは各アームの予測精度に差がなく，小さなモジュールでは精度に差がある場合，規模の小さいモジュール順にテストするほうが，バンディットアルゴリズムがより有効に働く可能性がある．この戦略を以降 SF と呼ぶ.

予測結果が陽性のモジュールからテストすれば，タイプ 1 が発生する可能性がないため，バンディットアルゴリズムによる最適なアーム選択がより有効に働く可能性がある．また，規模の大きなモジュールは実際に欠陥が含まれている可能性が高い．欠陥あり，かつ規模の大きいモジュール順にテストする戦略は，過去の研究[1][17]でも類似の方法が示唆されている．以降ではこの戦略を PF と呼ぶ.

3. 実験

3.1　概要

NASA[6]，PROMISE[2]リポジトリから得た 6 つのプロジェクトを実験用データセットとして用いた．これらはアンサンブル学習による欠陥予測の研究で広く用いられている[14][15].

予測精度の評価指標として，AUC を用いた．AUC はソフトウェア欠陥予測の研究において広く用いられている[14][15]．また，テスト工数についても評価指標として用いた．なお，工数の比較時は式(1)中の定数 *c* は無視できる．さらに評価指標を比較するために，以下の *RDIFF* を定義した.

$$RDIFF = 1 - \frac{target}{baseline} \quad (2)$$

例えば LF と SF のテスト工数を比較する場合，LF を *baseline*，LF を *target* として計算した.

ソフトウェア欠陥予測モデルを構築するための，アンサンブル学習の方法として，バギング，XGBoost，ランダムフォレスト，スタッキングを用いた．バンディットアルゴリズムとして，ε グリーディ法と UCB を用いて．ε の値は，広く用いられている[18]，0，0.1，0.2，0.3 を設定した.

NASA データセットを用いる場合，ラーニングデータとテストデータとして 3:1 の比率に分割した．PROMISE でたーセットを用いる場合，クロスバージョン予測を適用した．ε グリーディ法にはランダムな要因が含まれるため，バンディットアルゴリズムについては 20 回繰り返して実施し，評価指標の平均値を採用した.

テスト工数比率は 0.1 とし，タイプ 1 の発生確率は 90%とした．タイプ 2 の発生確率は文献[8]に基づき 20%とした．複数のアンサンブル学習の候補から，ランダムにひとつの方法を選択する場合，予測精度の期待値は，候補の方法の平均値となる．この平均値 0.641 を基準値として，バンディットアルゴリズムが基準値を上回っている場合，有効であるとみなした.

表 1　テスト戦略とバンディットアルゴリズムによる予測の AUC との関係

戦略	SF	LF	PF
ε = 0	0.638	0.651	0.653
ε = 0.1	0.640	0.651	0.656
ε = 0.2	0.640	0.652	0.655
ε = 0.3	0.642	0.650	0.656

3.1　結果

LF，SF，PF による各テスト戦略と，バンディットアルゴリズムの AUC との関係を表 1 に示す．表において，AUC は 6 つのデータセットの平均値である．表に示すように，SF の精度が最も低く，PF の精度が最も高かった．ほとんどの場合，SF の AUC は前節で述べた基準値 0.641 よりも低いことから，SF はバンディットアルゴリズムにおいて用いるべきでないといえる．一方で，LF と PF の AUC は基準値よりも大きく，また PF は LF よりも高い AUC となっていた．このことから，バンディットアルゴリズムの適用時には，PF をテスト戦略とすることが最適であると言える．

LF を *baseline* としたとき，LF のテスト工数の *RFIFF* は 4.8%，PF のテスト工数の *RFIFF* は 12.2%となった．詳細は省略するが，一般のアンサンブル学習においても，予測精度が高い場合，同様にテスト工数も高まる傾向があった．このため，PF によりバンディットアルゴリズムの予測精度を考慮する場合，テスト工数が増加する傾向にあることは大きな欠点とはみなされない．

4. おわりに

本稿ではアンサンブル学習による欠陥予測の精度に対する，データセットの影響を抑えるため，バンディットアルゴリズムを用いてオンライン最適化することを提案した．実験では，LF（規模の大きなモジュール優先），SF（規模の小さなモジュール優先），PF（欠陥ありと予測されたモジュール優先）による各テスト戦略が，バンディットアルゴリズムベースの欠陥予測とテスト工数にどのように影響するかを分析した．その結果，PF を適用した場合の予測精度が最も高い一方で，かつテスト工数も大きくなる傾向が見られた．

参考文献

[1] Asano T., Tsunoda M., Toda K., Tahir A., Bennin K., Nakasai K., Monden A., Matsumoto K.: Using Bandit Algorithms for Project Selection in Cross-Project Defect Prediction, Proc. of International Conference on Software Maintenance and Evolution (ICSME), pp.649-653 (2021).

[2] Caglayan B., Kocaguneli E., Krall J., Peters F., Turhan B.: The PROMISE repository of empirical software engineering data (2012).

[3] D'Ambros M., Lanza M., Robbes R.: Evaluating defect prediction approaches: a benchmark and an extensive comparison, Empirical Software Engineering, vol.17, no.4-5, pp.531-577 (2012).

[4] Dou J., Yunus A., Bui D. et al.: Improved landslide assessment using support vector machine with bagging, boosting, and stacking ensemble machine learning framework in a mountainous watershed, Japan, Landslides, vol.17, pp.641-658 (2020).

[5] Fedorov N., Yamasaki Y., Tsunoda M., Monden A., Tahir A., Bennin K., Toda K., Nakasai K.: Building Defect Prediction Models by Online Learning Considering Defect Overlooking, arXiv:2404.11033 (2024).

[6] Gray D., Bowes D., Davey N., Sun Y., Christianson B.: The misuse of the NASA metrics data program data sets for automated software defect prediction, Proc. of Annual Conference on Evaluation and Assessment in Software Engineering (EASE), pp.96-103 (2011).

[7] Hayakawa T., Tsunoda M., Toda K., Nakasai K., Tahir A., Bennin K., Monden A., Matsumoto K.: A Novel Approach to Address External Validity Issues in Fault Prediction Using Bandit Algorithms, IEICE Transactions on Information and Systems, vol.E104.D, no.2, pp.327-331 (2021).

[8] 情報処理推進機構(IPA) : ソフトウェア開発データ白書 2018-2019, IPA (2018).

[9] ISO/IEC/IEEE International Standard: Software and systems engineering - Software testing - Part 2: Test processes, ISO/IEC/IEEE 29119-2:2021(E), (2021).

[10] Kamei Y., Shihab E., Adams B., Hassan A., Mockus A., Sinha A., Ubayashi N.: A large-scale empirical study of just-in-time quality assurance, IEEE Transactions on Software Engineering, vol. 39, no. 6, pp. 757-773, (2013).

[11] Mahfuz S.: Software Quality Assurance - Integrating Testing, Security, and Audit, CRC Press, (2016).

[12] Matloob F. et al.: Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review, IEEE Access, vol.9, pp.98754-98771 (2021).

[13] Monden A., Hayashi T., Shinoda S., Shirai K., Yoshida J., Barker M., Matsumoto K.: Assessing the Cost Effectiveness of Fault Prediction in Acceptance Testing, IEEE Transactions on Software Engineering, vol. 39, no. 10, pp. 1345-1357 (2013).

[14] Sun Z., Song Q., Zhu X.: Using Coding-Based Ensemble Learning to Improve Software Defect Prediction, IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol.42, no.6, pp.1806-1817 (2012).

[15] Tong H., Liu B., Wang S.: Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning, Information and Software Technology, vol. 96, pp. 94-111 (2018).

[16] Tsunoda M., Monden A., Toda K., Tahir A., Bennin K., Nakasai K., Nagura M., Matsumoto K.: Using Bandit Algorithms for Selecting Feature Reduction Techniques in Software Defect Prediction, Proc. of Mining Software Repositories Conference (MSR), pp.670-681 (2022).

[17] Turhan B., Menzies T., Bener A., Stefano J.: On the relative value of cross-company and within-company data for defect prediction, Empirical Software Engineering, vol.14, no.5, pp.540-578 (2009).

[18] White J.: Bandit Algorithms for Website Optimization: Developing, Deploying, and Debugging, O'Reilly Media (2012).

[19] Zimmermann T., Nagappan N.: Predicting defects using network analysis on dependency graphs, Proc. of

International Conference on Software Engineering (ICSE), pp.531-540 (2018).