社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

現実的な設定に基づいたバグ予測モデルの構築及び精度評価

荻野　翔† 肥後　芳樹† 楠本　真二†

† 大阪大学大学院情報科学研究科
〒 565-0871 大阪府吹田市山田丘 1-5
E-mail: †{s-ogino,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　バグ予測は品質保証に必要なコストを低減できると期待されている．バグを予測するモデルを構築する場合，以下の 3 つの条件を全て満たす現実的な設定が用いられるべきである．1 つ目に，データセットの構築方法について，モデルの予測能力を正しく評価できる方法が用いられている．2 つ目に，予測対象の粒度について，品質保証のコストを最小化する最適な粒度が採用されている．3 つ目に，目的変数について，予測対象となるソフトウェアモジュールのバグの有無を正しく表す指標が用いられている．しかし，これらの条件を全て満たす設定で構築されたバグ予測モデルについての精度評価実験は未だ行われていない．そこで本研究では，これらの条件を満たす現実的な設定のもとでバグ予測モデルを構築し，その予測精度を計測した．実験の結果，構築されたバグ予測モデルの F 値は 0.18 と低く，現実的な設定のもとで高精度のバグ予測モデルを構築するには大きな課題が残されていると判明した．

キーワード　品質保証，バグ予測，機械学習

1. まえがき

近年，ソフトウェア開発の規模は増大の一途を辿っている [1]．このような状況で，開発コストを低減する技術は重要である．開発コストの中でもレビュー等の品質保証に必要なコストが大きいため，それを低減する技術は特に重要である [2]．

品質保証に必要なコストを低減する技術としてバグ予測が存在する．バグ予測とは，プロジェクトを構成するソフトウェアモジュール（例：ソースファイル）に対象に，バグの有無を予測することである．バグ予測によりバグが存在するソフトウェアモジュールを特定し，それらを重点的にレビューすることで，品質保証に必要なコストを低減できる．

バグを予測するモデル（バグ予測モデル）の予測能力を正しく評価するためには，データセットを適切に構築すべきである．従来，プロジェクトのある時点に存在するソフトウェアモジュールについて目的変数を算出し，プロジェクト開始時からその時点までの開発履歴を用いて説明変数を算出することでデータセットが構築されていた [3] [4]．しかし，上記のように構築されたデータセットでは，モデルの予測能力が正しく評価されないと Pascarella らは批判した [5]．Pascarella らはその問題を回避できるデータセットの構築方法としてリリースバイリリースという手法を提案し，その手法を用いてバグ予測モデルの予測精度を調査した．その結果，低い予測精度が計測され，高精度のバグ予測モデルを構築するには大きな課題が残されていると彼らは結論づけた．

しかし，Pascarella らがリリースバイリリースの調査を目的として構築したバグ予測モデルの設定には議論の余地がある．具体的には，目的変数として，「その時点でソフトウェアモジュールにバグが存在するかどうか（isBuggy）」という指標ではなく，「過去の一定期間内で一度でもバグが修正されたかどうか（hasBeenFixed）」という指標が設定されている点が問題である．なぜなら，ソフトウェアモジュールには，「hasBeenFixed について偽だが isBuggy について真である」という状況があり得るため，hasBeenFixed を目的変数とするモデルはバグを正しく予測できていない可能性があるからである．

Pascarella らが下した結論について真偽を確かめるために，hasBeenFixed がバグ予測モデルの目的変数として適切かを調査すべきであり，もし適切でないならば，現実的な設定に基づいて構築されたバグ予測モデルについて予測精度を調査すべきである．しかし，そのような調査は未だ行われていない．そこで，著者らはまずバグ予測モデルの目的変数として hasBeenFixed が適切かどうかを調査した．次に，目的変数・データセットの構築方法・予測対象の粒度について現実的な設定を用いて構築されたバグ予測モデルについて精度評価実験を行った．

その結果，hasBeenFixed はバグが存在するメソッドの 25 % しか検出できず，バグ予測モデルの目的変数としては不適切だと判明した．また，現実的な設定のもとで構築されたバグ予測モデルの F 値は 0.18 であり，現実的な設定のもとで高精度のバグ予測モデルを構築するためには大きな課題が残されていることが明らかになった．

2. 研究背景

バグ予測モデルの構築において，様々な設定が混在する．本章では，本研究と関わりが深い項目として，予測対象の粒度と

データセットの構築方法について紹介する．

2.1 予測粒度

これまでバグ予測モデルの研究に採用されてきた予測対象の粒度としては，ソースファイル・メソッド・コミットの 3 種類が存在する．本研究ではメソッドを予測粒度としたバグ予測モデルについて調査する．なぜなら，本研究では Pascarella らの結論の真偽を確かめることを目標の 1 つとしており，Pascarella らは現実的な設定として予測粒度にメソッドを採用しているからである．

2.2 データセットの構築方法

バグ予測モデルの予測精度を正しく評価するためには，適切なデータセットの構築方法を採用すべきである．従来，プロジェクトのある時点に存在するソフトウェアモジュールについて目的変数を算出し，プロジェクト開始時からその時点までの開発履歴を用いて説明変数を算出することでデータセットが構築されていた [3] [4]．そのようにデータセットを構築する場合，大半のメソッドについてバグの有無が判明している状態で残りのメソッドのバグを予測するという非現実的な状況下でのバグ予測精度が評価されてしまうと Pascarella らは批判した．

そして Pascarella らは，そのような問題を回避できる適切なデータセットの構築方法としてリリースバイリリースを提案した．リリースバイリリースは，プロジェクトのあるリリース時に存在するソフトウェアモジュールについて，目的変数と説明変数を算出する．このとき，説明変数は 1 つ前のリリースからそのリリースまでの開発履歴を用いて算出する．リリースバイリリースに基づいてデータセットを構築する具体的な手順については，詳細を 6.1.4 項で述べる．

3. Research Questions

本研究では，以下の Reasearch Question (RQ) を調査する．

> RQ1: hasBeenFixed はバグ予測モデルの目的変数として適切か．

RQ1 の目的は，先行研究のモデルの目的変数が適切かを定量的に判定することである．

例えば，hasBeenFixed が真であるメソッドに isBuggy が真であるメソッドの大半が含まれない場合，hasBeenFixed を目的変数として採用したモデルはバグの所在を正しく予測できず，hasBeenFixed はバグ予測モデルの目的変数として不適切であると言える．

> RQ2: 現実的な設定のもとで構築されたバグ予測モデルの予測精度はどの程度か．

RQ2 の目的は，以下に示す現実的な設定を用いてバグ予測モデルを構築した場合の予測精度を明らかにすることである．

目的変数：isBuggy
予測粒度：メソッド
データセットの構築方法：リリースバイリリース

予測精度が低ければ，バグ予測モデルについての研究にはまだ課題が残されており，Pascarella らの結論は正しいと言える．

> RQ3: 現実的な設定のもとで構築されたバグ予測モデルの予測精度は，開発履歴の量が変化するとどのように変化するか．

RQ3 の目的は，開発履歴の量によって，利用するモデルアルゴリズムを変化させるべきか等の知見を得ることである．

4. 各 RQ に共通する実験設定

本章では，各 RQ で共通する実験設定について述べる．

4.1 調査対象

本実験が調査対象とするプロジェクトの概要が表 1 に記されている．以下の 6 つの理由により，これらのプロジェクトを選択した．

- GitHub でリポジトリが公開されている．
- Java で開発されている．
- Jira で 1,000 以上のバグレポートが取得可能である．
- コミット数が 10,000 以上である．
- 4 回以上のメジャーリリースが行われている．
- セマンティックバージョニングが採用されている．

4.2 リリースの特定方法

本研究では，2.2 節で説明されたリリースバイリリースという手法でデータセットを構築する．そのためには，対象プロジェクトの各リリースのタイミングを特定する必要がある．本研究では，Pascarella らが提案した方法に従い，セマンティックバージョニングを採用したプロジェクトについて，リリースを以下のように特定する．セマンティックバージョニングは，X.Y.Z という形式（例: 1.0.0）でバージョンを表すバージョン表記である．セマンティックバージョニングでは，X がメジャーバージョンを表す．メジャーバージョンのリリース，つまり Y と Z がともに 0 であるようなバージョンのリリースをプロジェクトのリリースとして特定する．

4.3 hasBeenFixed の算出方法

前提として，プロジェクトのリポジトリを GitHub 等から取得できているとする．また，そのプロジェクトのバグレポートを Jira 等から取得できているとする．

各メソッドについて，hasBeenFixed は以下のように算出さ

表 1 対象プロジェクト一覧

プロジェクト名	開発期間	コミット回数	リリース回数	バグレポート数	バグが存在するメソッドの割合（リリース時）
lucene-solr	18 年 348 日	34,037	9	6,198	1.9%
wicket	15 年 337 日	20,925	7	2,868	1.2%
cassandra	11 年 145 日	23,537	4	5,351	5.2%

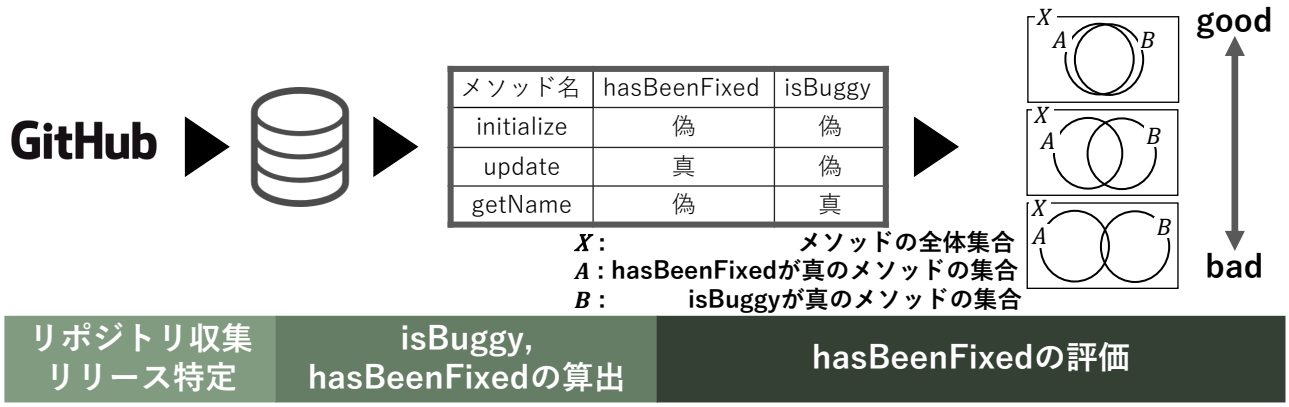

図 1　RQ1 について行う実験の流れ

れる．

step1　過去に発見されたバグについてはバグレポートが発行されており，バグレポートには ID が付与されている．その ID がコミットメッセージに記されているコミットが，そのバグについて修正を行うコミット（バグ修正コミット）である．バグ修正コミットを取得するために，コミットメッセージにバグレポート ID が記されているコミットをリポジトリから検索する．

step2　検索により得られたバグ修正コミットでは，あるメソッド A のある行について修正が行われているとする．そのコミットが $n-1$ 番目のリリース（R_{n-1}）から n 番目のリリース（R_n）までの区間に含まれる場合，メソッド A は R_n の時点で hasBeenFixed が真である．

step3　すべてのバグレポートについて上記の操作が完了した時に，hasBeenFixed が真でないメソッドは，hasBeenFixed が偽である．

4.4　isBuggy の算出方法

前提として，プロジェクトのリポジトリを GitHub 等から取得できているとする．また，そのプロジェクトのバグレポートを Jira 等から取得できているとする．

isBuggy の算出には SZZ アルゴリズム [6] を利用し，実装としては [7] を用いた．各メソッドについて，isBuggy は以下のように算出される．

step1　hasBeenFixed の算出方法の step1 と同様である．過去に発見されたバグについてはバグレポートが発行されており，バグレポートには ID が付与されている．その ID がコミットメッセージに記されているコミットが，そのバグについて修正を行うコミット（バグ修正コミット）である．バグ修正コミットを取得するために，コミットメッセージにバグレポート ID が記されているコミットをリポジトリから検索する．

step2　検索により得られたバグ修正コミットで，あるメソッド A のある行について修正が行われていると仮定する．ここで，修正された箇所はバグであったとみなせる．修正された箇所を挿入したコミットを，"git diff"等の操作によって特定し，そのコミットをメソッドにバグを混入させたコミット（バグ混入コミット）と定義する．

step3　メソッド A についてのバグ混入コミットが R_n 以前に存在し，R_n 以降にそのバグについてのバグ修正コミットが存在する場合，R_n の時点でメソッド A は isBuggy が真である．

step4　すべてのバグレポートについて上記の操作が完了した時に，isBuggy が真でないメソッドは，isBuggy が偽である．

5.　RQ1

RQ1 では，hasBeenFixed という指標はバグが存在するメソッドを検出できているのか，すなわち hasBeenFixed という指標はバグ予測モデルの目的変数として適切かを調査する．

5.1　実 験 設 定

5.1.1　実験の流れ

RQ1 について調査するために，実験を行う．その概要を図 1 と以下に示す．

step1　リポジトリ・バグレポートを収集する．

step2　各リリース時に存在するメソッドについて，isBuggy・hasBeenFixed を算出する．

step3　hasBeenFixed の適切さを評価する．

5.1.2　評 価 指 標

hasBeenFixed が適切かを評価する指標としては，以下に示す再現率を用いる．

$$\text{再現率} = \frac{|X_{hasBeenFixed} \cap X_{isBuggy}|}{|X_{isBuggy}|}$$

ここで，$X_{hasBeenFixed}$ は hasBeenFixed が真と評価されるメソッドの集合，$X_{isBuggy}$ は isBuggy が真と評価されるメソッドの集合である．算出された再現率の値が 0.5 以下であれば，目的変数として hasBeenFixed を採用したバグ予測モデルは半数以上のバグを予測できないため，hasBeenFixed はバグ予測モデルの目的変数として不適切であるとする．

5.2　実 験 結 果

実験の結果を表 2 に示す．表 2 は，各プロジェクトにおける再現率の値，すなわち hasBeenFixed という指標が実際のバグを検出した割合を示している．プロジェクトごとの再現率の値，そして平均値はいずれも 0.50 を大幅に下回っている．よって，hasBeenFixed はメソッドのバグの有無を表せておらず，バグ予測モデルの目的変数として不適切であると言える．

表 2　isBuggy に対する hasBeenFixed の再現率

	lucene-solr	wicket	cassandra	平均値
再現率	0.195	0.254	0.353	0.25

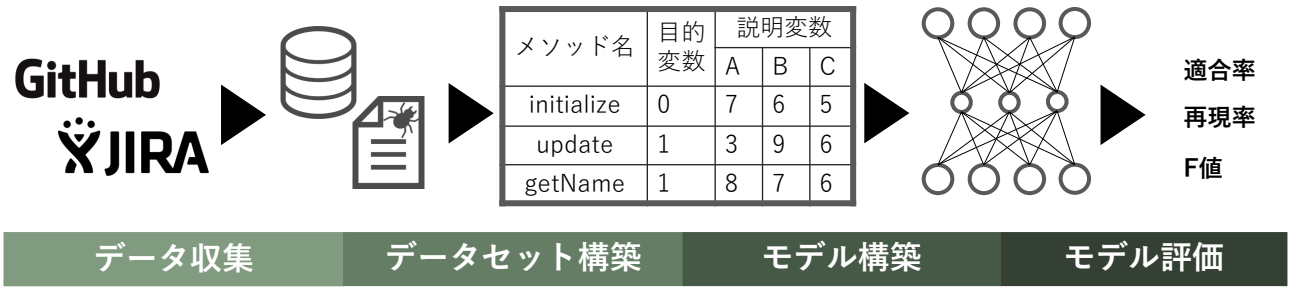

図 2　RQ2 について行う実験の流れ

6. RQ2

RQ2 では，現実的な設定のもとで構築されたバグ予測モデルの予測精度を調査する．

6.1 実験設定

6.1.1 実験の流れ

RQ2 について調査するために，実験を行う．その概要を図 2 と以下に示す．

step1　リポジトリ・バグレポートを収集する．

step2　収集されたデータを元にデータセットを構築する．

step3　データセットを元にバグ予測モデルを構築する．

step4　バグ予測モデルの予測精度を評価する．

6.1.2 目的変数

目的変数としては isBuggy を設定した．定義は 4.4 節で述べられた．

6.1.3 説明変数

説明変数としては，Pascarella らが用いたプロダクトメトリクス・プロセスメトリクスを採用した．その概要を表 3，表 4 に示す．プロダクトメトリクスとは，ソースコード単体から得られるメトリクス（例：ソースコードの行数）である．プロセスメトリクスとは，開発履歴から得られるメトリクス（例：コミットの回数）である．

6.1.4 学習データセット・テストデータセットの構築

本調査においては，データセットをリリースバイリリースで構築する．つまり，プロジェクトのリリースに着目して，リリース時に存在するソフトウェアモジュールについて目的変数を算出し，説明変数を 1 つ前のリリースからそのリリースまでの開発履歴を用いて算出し，その組合せをデータセットに追加していくことでデータセットを構築する．

表 3　説明変数（プロダクトメトリクス）

メトリクス名	概要
FanIn	そのメソッドを参照するメソッドの数
FanOut	そのメソッドが参照するメソッドの数
LocalVar	ローカル変数の数
Parameters	引数の数
CommentToCodeRatio	ソースコードに対するコメントの割合（行単位）
CountPath	実行可能経路の数
Complexity	サイクロマティック数
execStmt	実行可能ステートメントの数
maxNesting	ネストの最大値

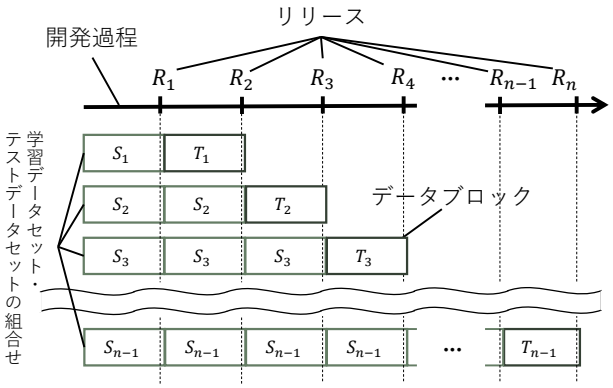

図 3　データセットの概要

このとき，図 3 に示すように，プロジェクト開始から k 番目のリリース（R_k）までの開発履歴を用いて構築されるデータセットを学習データセット S_k とし，R_k から R_{k+1} までの開発履歴を用いて構築されるデータセットをテストデータセット T_k とすることで，リリース総数が n のプロジェクトからは，$n-1$ 個の学習データセット・テストデータセットの組合せが得られる．

あるプロジェクトについて S_k，T_k を構築する手順を以下に示す．まず，各リリース番号 m について，以下のようにデータブロック B_m を構築する．

step1　R_m 時点で存在するメソッドについて以下の操作を行う．

step1-1　R_m におけるソースコード，そして R_{m-1} から R_m ま

表 4　説明変数（プロセスメトリクス）

メトリクス名	概要
MethodHistories	コミット回数
Authors	そのメソッドを編集した人数
StmtAdded	追加されたステートメントの総数
MaxStmtAdded	追加されたステートメントの最大値
AvgStmtAdded	追加されたステートメントの平均値
StmtDeleted	取り除かれたステートメントの総数
MaxStmtDeleted	取り除かれたステートメントの最大値
AvgStmtDeleted	取り除かれたステートメントの平均値
Churn	StmtAdded - StmtDeleted
MaxChurn	Churn の最大値
AvgChurn	Churn の平均値
Decl	メソッド宣言の変更回数
Cond	条件文の変更回数
ElseAdded	else 文の追加回数
ElseDeleted	else 文の削除回数

での開発履歴に基づいて説明変数を算出する．なお，R_0 はプロジェクト開始時とする．

step1-2　isBuggy を目的変数として算出する．

step1-3　算出された説明変数・目的変数の組を B_m に加える．

こうして構築されたデータブロックを以下のように S_k，T_k に分割する．

step1　学習データセット S_k を，k 以下の添字を持つデータブロックに含まれる説明変数・目的変数の組（サンプル）によって構成する．

step2　不均衡データを正しく分類できるモデルを構築するため，S_k について，目的変数 isBuggy が真であるサンプルと偽であるサンプルが同数になるように，目的変数 isBuggy が真であるサンプルをオーバーサンプリング [8] する．

step3　テストデータセット T_k を，添字が $k+1$ のデータブロックに含まれるサンプルによって構成する．

6.1.5　モデルアルゴリズム

モデルアルゴリズムとしてはランダムフォレスト（RF），ディープニューラルネットワーク（DNN）を採用した．上記のアルゴリズムを用いる理由は，バグ予測モデルは予測精度が重要であるため，DNN のように設定可能なパラメータが多くモデルの表現力があるものが望ましいからである．DNN に加えて RF を採用した理由は，パラメータ数が多いからといって，必ずしも DNN が RF より予測精度の観点で優れているわけではない [9] からである．

6.1.6　パラメータチューニング

予測モデルを構築する場合，予測精度の高いモデルを構築するためにはハイパーパラメータのチューニングが必須である．本研究では，パラメータチューニングのアルゴリズムとしてはベイズ最適化 [10] を採用し，その実装としては optuna [11] を利用した．また，パラメータチューニングには CPU として AMD Ryzen 9 3950X，GPU として NVIDIA GeForce RTX 2080Ti を搭載するコンピュータを使用した．それぞれのモデルに対して 24 時間を費やしてパラメータチューニングを行った．

6.1.7　評 価 指 標

バグ予測モデルを評価する指標を以下に示す．

$$\text{再現率} = \frac{|TP|}{|FN+TP|}$$

$$\text{適合率} = \frac{|TP|}{|FP+TP|}$$

$$F\ \text{値} = \frac{2\times\text{再現率}\times\text{適合率}}{\text{再現率}+\text{適合率}}$$

TP はバグが存在すると予想され，実際にバグが存在したメソッドの数である．FN はバグが存在しないと予想され，実際にはバグが存在したメソッドの数である．FP はバグが存在すると予想され，実際にはバグが存在しなかったメソッドの数である．

6.2　実 験 結 果

実験結果を表 5 に示す．表 5 は，現実的な設定のもとでバグ予測モデルを構築した際の予測精度を表す．F 値の観点から見ると，現実的な設定のもとでは，RF の方が DNN よりもバグ予測モデルを構築するアルゴリズムとして優れていることがわかる．しかし，その RF で構築されたモデルの適合率の値は 0.12 であり，これはバグが存在すると予測されたメソッドの 9 個に 1 個しかバグがないことを意味する．また，再現率の値は 0.55 であり，これはおよそ半分ほどの，バグが存在するメソッドを検出できないことを意味する．結論として，現実的な設定のもとで構築されたバグ予測モデルの予測精度はかなり低い．

7.　RQ3

RQ3 では，現実的な設定のもとで構築されたバグ予測モデルの予測精度が，開発履歴の量が変化するにしたがってどのように変化するかを調査する．

7.1　実 験 設 定

データセット・モデル構築時の設定は RQ2 と同様である．

7.2　実 験 結 果

実験結果を図 4，5，6 に示す．これらの図は，前述した現実的な設定のもとで構築されたバグ予測モデルについて，学習データセット・テストデータセットの組合せごとにモデルを構築・評価した結果得られた F 値をグラフの形で示している．図 4，5，6 からわかるように，RF の予測精度が常に DNN を上回っている．この結果は，DNN の表現力の大きさが仇となったためだと考える．つまり，前後のリリースでバグが存在するメソッドの特徴が異なっていて，前のリリースにおいてバグが存在するメソッドの特徴をよく学習してしまったために，後のリリースにおいてメソッドのバグの有無をうまく予測できなかったためだと考える．

結論として，現実的な設定のもとで構築されたバグ予測モデルの予測精度は，開発履歴の量が増加するにつれ，低下する．また，本研究における設定のもとでは，どのような開発履歴量であろうとも，DNN より RF をモデルアルゴリズムとして用いるべきである．

8.　妥当性の脅威

目的変数　本研究では，修正済みと記されたバグレポートを利用して，メソッドのバグの有無を判定している．よって，正確なデータセットを構築するには，存在する全てのバグが報告され，修正されている必要がある．しかしながら，本研究で対象としたプロジェクトには未報告のバグや，未修正のバグが存在する．この問題を解決することで，予測精度が高く有意義なモデルを構築できる可能性がある．

説明変数　本研究でバグ予測モデルの説明変数として採用したのは，ソースコード・開発履歴から算出されたメトリクスである．メトリクスは元となるソースコード・開発履歴から抽出されたデータであり，バグ予測に重要な情報が抽出の過程で捨てられている可能性がある．よって，他のメトリクスを採用することで，予測精度が高く有意義なバグ予測モデルが構築される

表 5　現実的な設定のもとで構築されたバグ予測モデルの予測精度

モデルアルゴリズム	適合率	再現率	F 値
RF	0.12	0.55	0.18
DNN	0.10	0.63	0.15

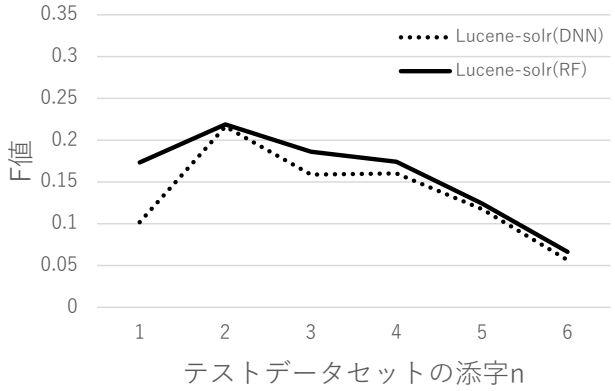

図 4　予測精度の推移（lucene の場合）

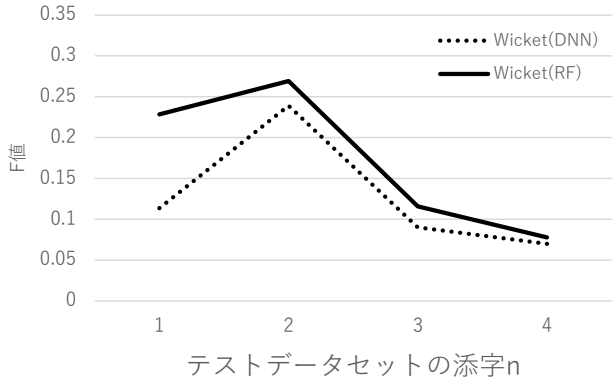

図 5　予測精度の推移（wicket の場合）

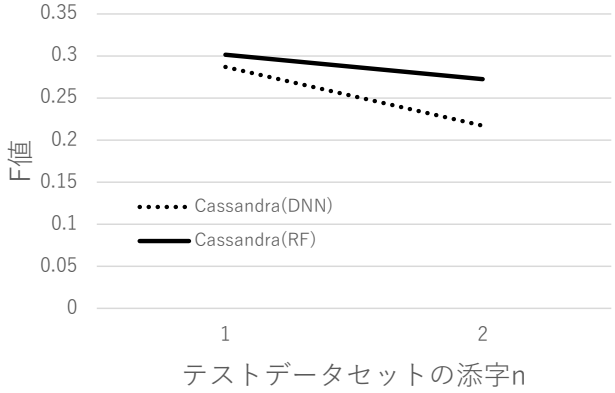

図 6　予測精度の推移（cassandra の場合）

可能性がある．

パラメータチューニング　RF，DNN 等のモデルアルゴリズムを用いてモデルを構築する場合，予測精度の高いモデルを構築するために，パラメータチューニングが不可欠である．パラメータ探索の範囲を広くし，試行回数を増やすことで，より予測精度の高いモデルを構築できる．本研究では，ベイズ最適化のアルゴリズムを用いて 24 時間のパラメータチューニングを行っているが，より長い時間を費やせば，予測精度が高く有意義なモデルを構築できる可能性がある．

サンプル数　各 RQ において得られた考察・結論は 3 つのプロジェクトについて実験を行った結果に基づいている．他のプロジェクトについて実験を行った場合，本研究の結論に当てはまらない結果を得る可能性がある．

9. あとがき

本研究では，まず「過去の一定期間内で一度でもバグが修正されたかどうか（hasBeenFixed）」という指標がバグ予測モデルの目的変数として適切かを調査した．そして，現実的な設定に基づいてバグ予測モデルを構築し，その予測精度を評価した．

結果として，hasBeenFixed は実際のバグの 25%しか検出せず，バグ予測モデルの目的変数としては不適切であると判明した．また，現実的な設定に基づいて構築されたバグ予測モデルの F 値が 0.18 であり，現実的な設定のもとでは，実用的なバグ予測モデルを構築することは難しいと判明した．以上の結果から，現実的な設定のもとで高精度のバグ予測モデルを構築するには大きな課題が残されていると言える．

今後の課題としては，説明変数の追加，より綿密なパラメータチューニング，そして対象プロジェクトの追加が挙げられる．

謝辞　本研究は，日本学術振興会科学研究費補助金基盤研究（B）（課題番号：20H04166）の助成を得て行われた．

文　献

[1] 情報処理推進機構，“ソフトウェア開発データ白書”. https://www.ipa.go.jp/ikc/publish/whitepaper_dl.html

[2] T. Britton, L. Jeng, G. Carver, P. Cheak, and T. KKatzenellenbogen, “Increasing software development productivity with reversible debugging”. https://undo.io/media/uploads/files/Undo_ReversibleDebugging_Whitepaper.pdf

[3] H. Hata, O. Mizuno, and T. Kikuno, “Bug prediction based on fine-grained module histories,” Proceedings - International Conference on Software Engineering, pp.200–210, 06 2012.

[4] E. Giger, M. D’Ambros, M. Pinzger, and H. Gall, “Method-level bug prediction,” International Symposium on Empirical Software Engineering and Measurement, pp.171–180, 09 2012.

[5] L. Pascarella, F. Palomba, and A. Bacchelli, “Re-evaluating method-level bug prediction,” 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp.592–601, 2018.

[6] J. undefinedliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?,” SIGSOFT Softw. Eng. Notes, vol.30, no.4, p.1–5, May 2005. https://doi.org/10.1145/1082983.1083147

[7] M. Borg, O. Svensson, K. Berg, and D. Hansson, “Szz unleashed: An open implementation of the szz algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project,” Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, p.7–12, MaLTeSQuE 2019, Association for Computing Machinery, New York, NY, USA, 2019. https://doi.org/10.1145/3340482.3342742

[8] N.V. Chawla, “Data mining for imbalanced datasets: An overview,” pp.875–886, Springer US, Boston, MA, 2010.

[9] M. Sewak, S.K. Sahay, and H. Rathore, “Comparison of deep learning and the classical machine learning algorithm for the malware detection,” 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), pp.293–296, 2018.

[10] J. Močkus, “On bayesian methods for seeking the extremum,” Optimization Techniques IFIP Technical Conference Novosibirsk, July 1–7, 1974, ed. by G.I. Marchuk, pp.400–404, Springer Berlin Heidelberg, Berlin, Heidelberg, 1975.

[11] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, “Optuna: A next-generation hyperparameter optimization framework,” Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, p.2623–2631, KDD ’19, Association for Computing Machinery, New York, NY, USA, 2019. https://doi.org/10.1145/3292500.3330701