Vol.2013-SE-182 No.21
2013/10/25

ソフトウェア開発プロジェクトをまたがるJust-In-Timeバグ予測の実験的評価

福島 崇文[1]　亀井 靖高[1]　鵜林 尚靖[1]

概要：本稿では，ソフトウェアの変更に対するバグ予測（Just-In-Time バグ予測）において，他のプロジェクトのデータを利用してモデル構築・判別を行う上で有用と考えられる手法を明らかにする．一般に，モデルの構築には同一プロジェクトのデータから計測されたメトリクスを用いることが多いが，データの蓄積ができていない企業や，新規開発のためデータの蓄積が少ない場合などでは利用できない．そこで本稿では，他のプロジェクトのデータを利用した有用な手法を明らかにするため，4 つのリサーチクエスチョンを実験的に検証した．6 つのオープンソースプロジェクトを用いて実験を行った結果，1）Just-In-Time バグ予測モデル構築において，ランダムフォレストはロジスティック回帰分析に比べて有用である，2）データに前処理（正規化）を適用しても，予測精度の向上に効果はない，3）複数のプロジェクトデータの中からメトリクス間の分布が類似しているデータを選ぶことで予測精度の向上が期待できる，4）複数のプロジェクトデータによる集団学習は安定した予測精度を得られる事が期待できる．といった知見が得られた．

キーワード：バグ予測モデル，クロスプロジェクト，ソフトウェア変更，ランダムフォレスト

1. はじめに

ソフトウェア試験や保守において，欠陥を含む確率の高いモジュール（以降，バグモジュールと呼ぶ）を特定することは，テスト工数割り当てやリファクタリングの優先順位の決定に役立つ [1], [2]．これまでに，バグモジュールを予測する手法として，モジュールのメトリクス（コード行数など計測可能な指標）を説明変数とし，バグの有無を目的変数とした予測モデル（以降，バグ予測モデルと呼ぶ）が多数提案されてきた [1], [2], [3].

バグ予測モデルは，テスト割り当ての優先付け等の試験・保守活動において一定の有用性が認められているものの，いくつかの課題を有することが指摘されている [4]．例えば，従来手法の予測対象は，主にパッケージやファイルといった粗い粒度である．予測モデルによってバグがあると予測されたとしても，開発者はどの部分を重点的にテストしてよいかの判断が困難である．また，従来手法は実装工程完了後にモジュールがテストされることを前提として提案されている．一般に，実装工程開始から完了までは 6 ヶ月以上かかることも多く，実装工程完了後に予測結果を受け取ったとして，開発者はそのモジュールをどう実装したのかを思い出すのに時間を費やす必要がある．

これらの課題を解決するため，ソフトウェアの変更に対してバグが含まれるか否かを予測する試みが行われている [5], [6], [7]．ソフトウェア変更レベルのバグ予測は，ファイルに対する予測に比べて予測対象の粒度が小さく，また，変更時に予測結果を開発者にフィードバックできるという利点があり，開発現場での実用性・有効性が期待できる．本稿では，変更毎にバグ予測を行いソフトウェア品質に役立てることを，Just-In-Time バグ予測（以降，JIT バグ予測）と呼ぶ．

しかしながら，JIT バグ予測にも従来手法と同様の課題が存在する．バグ予測モデルを構築するためのデータ（メトリクスとバグの有無の関係）の蓄積がないとモデルの構築ができず，予測モデルを利用できないという点である．そのため，データの蓄積ができていない企業や，新規開発のためデータの蓄積が少ない場合などは JIT バグ予測が利用できない．

そこで本稿では，JIT バグ予測において，オープンソースソフトウェアなどの公開されているデータや，他プロジェクトのデータを利用してモデル構築・判別を行うことを試みる．その評価として本稿では，オープンソースプロジェクトから著者らが収集したデータセット（6 プロジェ

[1] 九州大学大学院システム情報科学研究院，福岡市
Graduate School and Faculty of Information Science and Electrical Engineering, Kyushu University, 744 Motooka, Nishi-ku, Fukuoka-shi, 819-0395 Japan

クト）を用いた．

以降，2 章で関連研究を紹介する．3 章では，本稿で取り組む 4 つのリサーチクエスチョンを述べる．4 章で実験に用いるデータと実験における評価指標を示し，5 章では実験の結果と考察を述べる．最後に 6 章でまとめを述べる．

2. 背景と関連研究

2.1 バグモジュール判別モデルの取り組み

これまでにも，バグの有無を目的変数とし，メトリクスを説明変数としたバグ予測モデルが用いられてきた．モデルにはいくつかの構築法があり，代表的なものとしては（1）ロジスティック回帰分析，（2）ランダムフォレスト がある．

（1） ロジスティック回帰分析

ロジスティック回帰分析では，ある現象の発生する確率 p を，その現象の生起を説明するための n 個の変数群 $x = (x_1, x_2, ..., x_n)$ で説明しようと試みる．本稿では，バグが混入する確率を説明することを目的とする．説明変数群 x のもとでバグが混入するという条件付き確率 $p(x)$ を，$p(x) = Pr\{$ バグ混入 $|x_1, x_2, ..., x_n\}$ という関数でモデル化する．a を定数と置いた一般線形式 $Z = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n$ を $p(x)$ に与え，ロジスティック回帰分析のモデル式は以下の形で表される．

$$p(x) = \frac{e^Z}{1 + e^Z} = \frac{1}{1 + e^{-Z}}$$

（2） ランダムフォレスト

ランダムフォレストは，2001 年に Breiman によって提案された，分類木や回帰木を用いて集団学習を行う手法である [8]．まず，データセットに対し，リサンプリング法により複数のデータをサンプリングする．各データからランダムサンプリングによる回帰木を作成し，全ての木による結果を多数決により組み合わせて判別結果を得る．ランダムフォレストは，変数をランダムランプリングしたものを用いるため，高次元データの解析に向いている．

従来の分類木は，説明変数と目的変数の間の非線形関係を表現できるものの，適切な木の深さや分岐数を定めなければ判別精度が高くなかった [9]．ランダムフォレストは，分類木の利点を有しつつ，判別精度が高いという特徴を持つ．また，ランダムサンプリングにより，モデル構築に用いるデータに過度な適合をしにくいという特徴を持つ．

（3） その他

（1），（2）の他にも，線形判別分析 [10]，分類木 [11]，ニューラルネットワーク [12]，Support Vector Machine[13] など，多くのバグモジュール判別モデルへの取り組みがなされている．（3）によるバグ予測の取り組みが行われているが，本研究では代表的なモデル構築法である（1）ロジスティック回帰分析，（2）ランダムフォレストを扱う．

2.2 ソフトウェア変更に対するバグ予測の先行研究

ソフトウェア変更を粒度としたバグ予測（JIT バグ予測）には，変更をした際に予測結果を開発者にフィードバックできる利点がある．1 章で述べたように，変更を行った際のフィードバックは開発現場での実用性，有効性が期待できる．ソフトウェア変更を粒度としたバグ予測はこれまでにも行われてきた．Mockus ら [7] は修正要求を単位とした，大規模なテレコミュニケーションシステムを対象とバグ予測を行った．また，亀井ら [5] は 14 種類のメトリクスを用いて予測モデルを構築し，その精度の評価を行った．亀井らはファイル数やコード行数の変更が多い場合，バグ修正を目的としたソフトウェア変更はバグを混入させる確率が高いことを示している．

本稿では，バグ予測モデルを構築するためのデータの蓄積がないとモデル構築ができないという課題を解決するために，他のプロジェクトのデータを用いることを試みる．

2.3 プロジェクトをまたがるバグ予測の先行研究

Sato ら [14] は，2 つの異なるプロジェクトの双方で，バグの有無に対して相関の強いメトリクスを抽出した．ここで抽出されたメトリクスはソースコード行数とネストレベルであった．Sato らは，これらのメトリクスを用いてモデル学習をおこなったときに予測がうまくいったことを示している．

Turhan ら [15] は，10 個の異なるプロジェクトデータを用いてバグ予測を行った．予測対象となるプロジェクト以外のデータを 1 つのデータとみなし，ベイズ識別器による予測モデルを構築した．同様に，木浦ら [16] は異なるプロジェクトデータを用いてバグ予測を行った．木浦らはプロジェクトごとにそれぞれ予測モデルを構築し，各モデルの出力値をもとに判別する方法を検証した．結果として，ロジスティック回帰分析において，両者の方法ともランダムに予測するよりも効果があることが示された．

蔵本ら [17] は予測モデルの構築に有用と考えられる 4 つの方法（Research Question）を考察し，各方法の予測精度を実験的に検証した．結果として，ランダムフォレストが有望であることや，学習データの選択がバグ予測精度を向上させることを示した．

本稿では，これらの従来研究で得られた結果が JIT バグ予測においても確認できるか否かを実験的に検証する．

3. Research Question

我々は，蔵本らの研究を参考に 4 つの Research Question

（RQ）を設定した．RQ1 はロジスティック回帰分析とランダムフォレストのどちらが JIT バグ予測で有効かを検証する．RQ2〜4 では，各検証が他のプロジェクトのデータを用いた場合の JIT バグ予測の予測精度を向上させるか否かを検証する．RQ2 は学習データを事前に正規化すること，RQ3 は複数プロジェクトから学習データを選択すること，RQ4 は複数プロジェクトによる集団学習の効果を検証する．

3.1 RQ1 ランダムフォレストの効果

ランダムフォレストは同プロジェクトやファイルレベルのプロジェクトをまたいだバグ予測で有効であると述べられてきた [8], [17]．RQ1 では，プロジェクトをまたがる JIT バグ予測において，ランダムフォレストが効果を発揮するかを確かめる．

3.2 RQ2 学習データ正規化の効果

一般的に，バグ予測モデルは，学習データと予測対象のメトリクス分布の傾向がある程度同じであることを前提としている．異なるプロジェクトのデータセットを用いる場合，メトリクス分布の傾向には違いがあると考えられる．メトリクス分布の傾向に違いがある場合，バグ予測モデルの性能が著しく低下する可能性がある．メトリクス分布の傾向の違いを小さくするために，モデル構築の前処理として，データの正規化（RQ2-1．対数変換，RQ2-2．Z-score 変換，RQ2-3．対数変換後に Z-score 変換）を行うことが，予測精度向上に有効であるかを確かめる．

3.3 RQ3 学習データ選択の効果

複数の学習データがある場合，予測対象に合わせて適切な学習データを選択することで，予測精度の向上が期待できる．RQ3 では 2 つの方法で学習データを選択し，それぞれの予測精度を確かめる．

RQ3-1 では，同じプロジェクトのバグ予測精度が高いものほど，学習データとして有用であるという仮定のもとで検証を行う．あるプロジェクトを学習データに用いたとき，そのプロジェクト自体を予測する場合と，異なるプロジェクトを予測する場合の予測精度を比較検証する．

RQ3-2 では，説明変数であるメトリクスの類似度を数値化し，最もメトリクスの傾向が類似しているデータセットを学習データに選択する．類似度は以下の手順で定義する．まず，データセットごとに目的変数であるバグの有無と各説明変数の相関係数を求める．この中から，相関係数の高い 3 つの説明変数を抽出する（q1，q2，q3）．あるデータセットを学習データとして予測モデルを構築する際，予測対象となるデータセットから（q1，q2，q3）と同じ説明変数を抽出する（r1，r2，r3）．次に，q1 と q2 の相関係数を Q1，q2 と q3 の相関係数を Q2，q1 と q3 の相関係数を Q3 とし，抽出した説明変数間の相関係数を求め，3 次元ベクトル（Q1，Q2，Q3）を得る．同様にして予測対象からは（R1，R2，R3）を得る．（Q1，Q2，Q3）と（R1，R2，R3）のユークリッド距離を類似度として計算する．

3.4 RQ4 集団学習の効果

複数のデータセットがある場合，これらを有効に活用することで予測精度の向上が期待できる．RQ4-1 はそれぞれのデータセットを学習データとして予測モデルを構築し，複数の予測モデルから予測精度を求める．RQ4-2 は複数のデータセットのうち，予測対象以外のすべてのデータを結合し，これを学習データとした 1 つの予測モデルから予測精度を求める．

4. 実験準備

4.1 実験対象のプロジェクト

我々は 6 つのオープンソースプロジェクト（Bugzilla，Columba，Eclipse JDT，Eclipse Platform，Mozilla，PostgreSQL）から収集したデータセットを用い，プロジェクトをまたがる JIT バグ予測をおこなった．各プロジェクトデータの概要を**表 1** に示す．

4.2 ソフトウェア変更で計測したメトリクス

従来研究ではバグ予測モデル構築のためにバージョン管理システムから，5 つのカテゴリ，14 種類のメトリクスを測定した．従来研究で測定したメトリクスのうち，ファイル変更に携わった開発者の数などの履歴に関するカテゴリや，開発者がプロジェクトに携わった期間などの経験カテゴリは，本研究では用いないことにする．これらのカテゴリは「データの蓄積ができていない企業や，新規開発のためデータが絶対的に少ない」場合には利用できないためである．我々が JIT バグ予測モデル構築に用いる 3 つのカテゴリ，8 種類のメトリクスを**表 2** に示す．

拡散カテゴリ：拡散カテゴリには，4 種類のメトリクスが含まれる．変更されるファイルが含まれるサブシステム数を NS とし，変更されるファイルが含まれるモジュール数を NM とし，ファイルが変更される数を NF とした．サブシステム，モジュール，及び，ファイルは，ディレクトリ構造によって判断した．例えば，org.eclipse.jdt.core/jdom/-org/eclipse/jdt/core/dom/Node.java が変更された場合は，サブシステムは org.eclipse.jdt.core ，モジュールは org.-eclipse.jdt.core/jdom …/dom，ファイル名は Node.java である．

Entropy は，各ファイル間で修正されるコードの分布を数値化するためのメトリクスである．Entropy を計測するために，Hassan の方法を採用した [20]．Entropy は $H(P) = -\sum_{k=1}^{n}(p_k * \log_2 p_k)$ として定義される $(p_k \geq 0, \forall k \in 1,2,...,n \ and \ (\sum_{k=1}^{n} p_k) = 1)$．例え

表 1　プロジェクトデータの概要

	期間	全変更数	学習データの変更数	テストデータの変更数	開発者の最大値	開発者の平均値
Bugzilla	08/1998 - 12/2006	4,620（36%）*1	2,791（38%）	1,829（35%）	37	8.4
Columba	11/2002 - 07/2006	4,455（31%）	4,076（32%）	379（20%）	10	1.6
Eclipse JDT	05/2001 - 12/2007	35,386（14%）	28,618（15%）	6,768（10%）	19	4.0
Eclipse Platform	05/2001 - 12/2007	64,250（14%）	48,966（15%）	15,284（14%）	28	2.8
Mozilla	01/2000 - 12/2006	98,275（ 5%）	69,651（ 6%）	28,624（ 4%）	155	6.4
PostgreSQL	07/1996 - 05/2010	20,431（25%）	15,048（28%）	5,383 （16%）	20	4.0
中央値		27,909（20%）	21,833（22%）	6,076（15%）	24	4.0

表 2　変更メトリクスの概要

	名称	概要	関連研究
拡散	NS	修正されたサブシステムの数	修正されるサブシステムの数が多いほど，バグの混入確率は高くなる [7]
	NM	修正されたモジュールの数	修正されるモジュール数が多いほど，バグの混入確率は高くなる [7]
	NF	修正されたファイルの数	モジュール内のファイル数は，リリース後に発見されるバグの予測に有用なメトリクスである [18]
	Entropy	エントロピー	変更が多くのファイルに散在するほど，バグの混入確率は高くなる [19], [20]
規模	LA	追加行数	変更行数は，バグモジュールを特定するのに有用である [21], [22]
	LD	削除行数	
	LT	ファイルの総行数	規模の大きいモジュールほど，より多くのバグを含む [23]
目的	FIX	バグ修正のための変更か否か	バグを修正するための変更は，新機能を実装するための変更より，バグを混入している確率が高い [24]

ば，変更によって 3 つのファイル（A，B，C）が修正され，それぞれのファイルにおいて 30 行，20 行，10 行ずつ修正されたとする．その場合，Entropy は，1.46 $(= -\frac{30}{60}\log_2\frac{30}{60} - \frac{20}{60}\log_2\frac{20}{60} - \frac{10}{60}\log_2\frac{10}{60})$ である．

規模カテゴリ：規模カテゴリには，3 種類のメトリクスが含まれる．追加行数（LA）と削除行数（LD）の測定には，変更ログに記されている追加／削除行数の値を用いた．また，ファイルの総行数（LT）には，変更対象のファイルを取得し，LOC の値を用いた．

目的カテゴリ：目的カテゴリでは，バグ修正のための変更か否かを計測する．バグ修正に関するキーワード（“bug”，“fix”，“defect”，“patch”）とバグ ID の組み合わせが変更ログ中に含まれるものを，バグ修正のために行われた変更とした．

4.3　データセットの事前処理

我々は RQ を検証する前に，データセットのメトリクスの相関係数を分析した．バグ予測モデルを構築するにあたり，相関の高いメトリクスを除去する必要がある [25]．分析の結果，NF，NM，LT の間に高い相関 (0.8 以上) があったため，NF と NM を除去し，LT を用いることにした．

4.4　評価指標

バグモジュール判別に関する研究では，適合率（Precision）と再現率（Recall）及びこれらの調和平均（F-measure）が用いられてきた [6], [26]．しかしながら，これらは目的変数となるバグの有無の割合に影響を受ける事が指摘されている [27]．本研究では，異なるプロジェクトをデータセットとするため，それぞれのデータセットごとにバグ有無の割合が大きく異なる．

そこで，本稿では，これらに代わる指標として AUC（Area Under the Curve）を用いる．AUC の値は，バグを混入するか否かとは独立したものであるため，バグ有無の割合の違いによる影響を受けにくい．あるソフトウェア変更にバグが混入するか否かを評価するといった目的に用いられる ROC（Receiver Operating Characteristic）曲線 (**図 1**) を描き，この曲線の下面積が AUC の値となる．ROC 曲線では，横軸にバグを混入していないがバグを含むと判定する確率，縦軸に正しくバグ混入を判別する確率をプロットする．AUC の値域は [0,1] であり，予測精度が高いほど ROC 曲線は左上に凸型となって，AUC の値は 1 に近づく．また，ランダムに抽出した場合，ROC 曲線は 45° の直線に近い形となり，AUC の値はおおよそ 0.5 になる．

*1　表 1 中の（）は，変更数に対するバグ混入割合

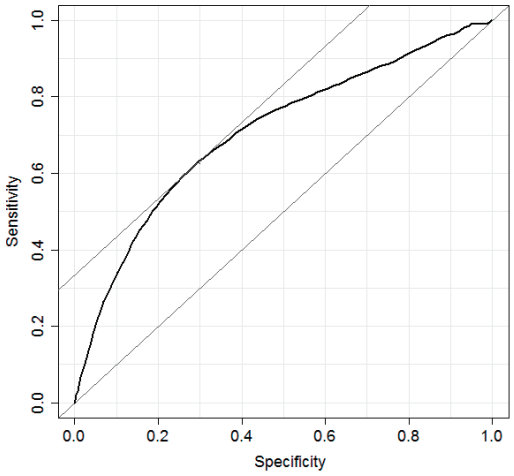

図 1　ROC 曲線と AUC

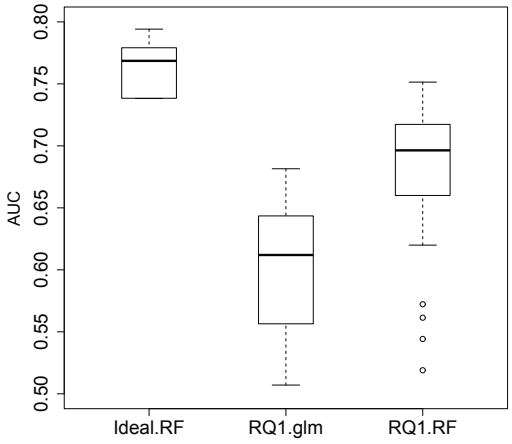

図 2　RQ1　ランダムフォレストの効果

5. 実験

5.1 RQ1 ランダムフォレストの効果

実験方法：6 つのデータセットでの全ての組み合わせについて，一方のデータセットを学習データとし，もう一方を予測対象とする．学習データを用いてバグ予測モデルを構築し，予測対象への予測精度を AUC の値を用いて評価する．モデル構築にロジスティック回帰分析とランダムフォレストを用い，ROC 曲線を描くことで得られた AUC を比較する．

結果：得られた AUC の値を**表 3** に示す．各行に予測対象，各列に学習データを示した．表 3 の値を箱ひげ図に描いたものが**図 2** である．図中の表記は，ロジスティック回帰分析を RQ1.glm，ランダムフォレストを RQ1.RF とした．

データの蓄積ができている場合は，予測を行うプロジェクト自体のデータを学習データとすることができる．異なるプロジェクトのデータを学習データとするよりも予測精度が高いことが考えられる．我々は，予測を行うプロジェクト自体のデータが利用できる場合を理想的なものとし，比較のために 10-fold cross validation を試行した．具体的には，データを 10 個に分割し，9 個分のデータを学習データとし，残った 1 個分のデータを予測対象とする．これを 10 回繰り返し，その平均値を最終的な結果とする．各データセットの 10-fold cross validation の結果を Ideal.RF とした．

結果として，ランダムフォレストの方が大きい AUC の値を示し，ロジスティック回帰分析と比較してランダムフォレストの予測精度が高いことが確かめられた．Ideal.RF は，同じプロジェクトに対する予測を仮想的におこなったものであり，中央値がプロジェクトをまたいだ予測の最大値よりも高いことが読み取れる．異なるプロジェクトデータを学習データに用いるよりも，同じプロジェクトデータを学習データに用いる方が予測精度が高いことが確かめられた．

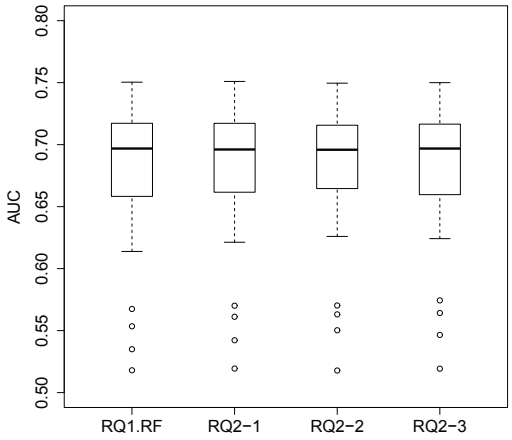

図 3　RQ2　学習データ正規化の効果

RQ1.glm と RQ1.RF を比較した結果，RQ1 においてランダムフォレストの有効性が確かめられたため，RQ2〜4 ではランダムフォレストを用いてバグ予測モデルを構築した．

5.2 RQ2 学習データ正規化の効果

実験方法：RQ2 ではデータセットに正規化処理をしたうえで，RQ1 と同様の実験を行う．RQ2-1．対数変換，RQ2-2．Z-score 変換，RQ2-3．対数変換後に Z-score 変換の 3 種類の正規化処理による予測精度を比較する．

結果：**図 3** に RQ2 の結果を示す．比較の対象として，RQ1 のランダムフォレストの結果も示す．実験を行った 3 種類の正規化処理では，予測精度に大きな変化は見られなかった．この結果から，JIT バグ予測において事前にデー

表 3　RQ1　ランダムフォレストの AUC

		学習データ					
		bugzilla	columba	jdt	mozilla	platform	postges
予測対象	bugzilla		0.56	0.72	0.70	0.68	0.69
	columba	0.57		0.69	0.72	0.74	0.67
	jdt	0.63	0.63		0.71	0.70	0.69
	mozilla	0.71	0.62	0.74		0.70	0.75
	platform	0.70	0.66	0.75	0.73		0.70
	postges	0.52	0.54	0.71	0.72	0.70	

タセットに正規化処理をしても予測精度の向上は期待できないことが確かめられた．

5.3　RQ3　学習データ選択の効果

実験方法：RQ3-1 では，各データセットごとに，10-fold cross validation の結果と RQ1 の結果を散布図に描き，比較検証する．x 軸に 10-fold cross validation によって予測対象自体のデータを学習データとして用いた場合の AUC の値，y 軸に任意のデータセットを予測対象とした RQ1 の結果の平均値をプロットする．散布図から，10-fold cross vaidation の値が高いものほど，学習データとして有効かどうかを検証する．

RQ3-2 では，3 章に述べた方法でメトリクスの類似度を求める．各予測対象について，最も類似度の高い（ユークリッド距離の小さい）データセットを学習データに選択したときの予測精度を確かめる．

結果：RQ3-1 の結果を**図 4** に示す．各データセットの 10-fold cross validation による AUC の値が，すべて 0.8 付近に集まった．このため，これらの差を仮定した RQ3-1 の学習データ選択を検証することはできなかった．

図 5 に RQ3-2 の結果を示す．ここでも比較の対象として，RQ1 のランダムフォレストの結果を示す．RQ3-2 の値は RQ1 の中から，類似度が高く，バグ予測精度の向上に有用と思われるデータを抜き出したものである．中央値に改善はないが，1 つの外れ値を除いて，最小値は改善している．

5.4　RQ4　集団学習の効果

実験方法：RQ4-1 では，予測対象以外のデータセットを学習データとした予測モデルから得られたバグ混入確率の平均値を求める．Mozilla のデータセットを例に挙げると，まず Mozilla 以外のデータセットを学習データとし，5 つのバグ予測モデルを構築する．各モデルから得られたバグ混入確率の平均値を Mozilla に対する予測結果として AUC の値を求め，箱ひげ図を描く．

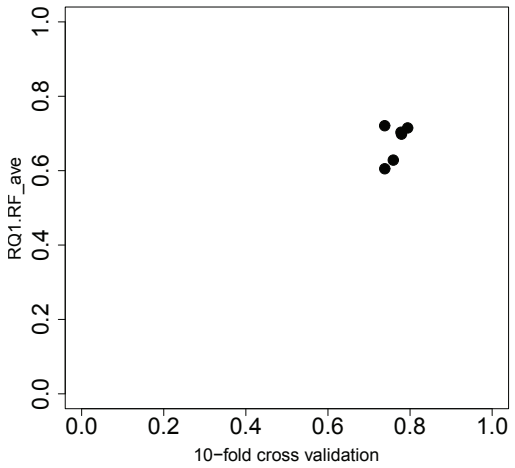

図 4　RQ3-1　10-fold cross validation による学習データ選択の効果

RQ4-2 では，予測対象以外のデータセットを結合し，これを学習データとする．この結合した学習データを用いて，1 つのバグ予測モデルを構築し，予測対象のバグ混入確率を求める．

結果：図 6 に RQ4-1 と RQ4-2 の結果を示す．ここでも比較の対象として，RQ1 のランダムフォレストを同図に含めた．RQ4-1，RQ4-2 の双方とも，RQ1 よりも最大値は小さく，最小値は大きい．しかしながら，RQ4 は RQ1〜3 と比較して，外れ値がなくなっているという特徴が見られる．RQ4-1 と RQ4-2 を比べると，RQ4-1 の方が予測精度向上に有効であることが示された．

6.　まとめ

本稿では，異なるプロジェクトのデータから JIT バグ予測を行う場合において，有用なモデル構築手段を確かめることを目的として，4 つの RQ の設定をし，実験的に検証を行った．6 つの異なるプロジェクトデータを用いて，4 つの RQ を実施した結果，下記の知見を得た．

- RQ1 では，プロジェクトをまたがる JIT バグ予測に

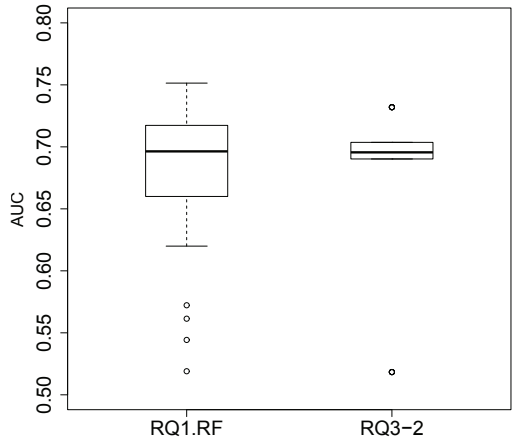

図 5　RQ3-2　類似度による学習データ選択の効果

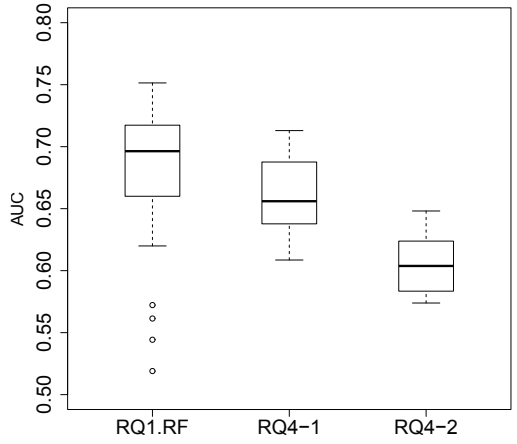

図 6　RQ4　集団学習の効果

対しても，先行研究と同様にランダムフォレストが有効であることが示された．

- RQ2 では，データセットのメトリクス分布の傾向を近づけるために前処理（正規化）をしても，効果がないことを確かめた．
- RQ3 では，プロジェクトをまたがるバグ予測の精度は，メトリクス間の相関が類似しているデータセットを学習データとして選択することに，予測精度の向上が期待できることを示した．しかし，選択する学習データによっては，予測精度の低い場合もあるため，開発現場で活用するには注意が必要である．
- RQ4 では，集団学習が安定した予測精度をもたらすことを示した．予測精度の最大値が多少小さくなってしまうものの，最小値が飛躍的に改善された．RQ4 には，外れ値が除去されるという RQ1〜3 にはなかった特徴が存在する．

これまでの実験から，我々は JIT バグ予測でモデルを構築する場合，ランダムフォレストによる RQ4-1 の方法を推奨する．RQ4-1 を推奨する理由は，予測精度の低い外れ値を除去できる点にある．

今後の課題としては，JIT バグ予測の予測精度を上げる新たなモデル構築方法を開発することや，精度向上に寄与するメトリクスを提案することが挙げられる．

謝辞　本研究の一部は，日本学術振興会科学研究費補助金（若手 A：課題番号 24680003）による助成を受けた．

参考文献

[1] Li, P. L., Herbsleb, J., Shaw, M. and Robinson, B.: Experiences and results from initiating field defect prediction and product test prioritization efforts at ABB Inc., Proc. Int'l Conf. on Softw. Eng. (ICSE'06), pp. 413–422 (2006).
[2] Mizuno, O., Ikami, S., Nakaichi, S. and Kikuno, T.: Fault-prone filtering: Detection of fault-prone modules using spam filtering technique, Proc. Int'l Symposium on Empirical Softw. Eng. and Measurement (ESEM'07), pp. 374–383 (2007).
[3] Basili, V. R., Briand, L. C. and Melo, W. L.: A validation of object-oriented design metrics as quality indicators, IEEE Trans. Softw. Eng., Vol. 22, No. 10, pp. 751–761 (1996).
[4] Kamei, Y., Shihab, E., Adams, B., Hassan, A. E., Mockus, A., Sinha, A. and Ubayashi, N.: A large-scale empirical study of Just-in-Time quality assurance, *IEEE Trans. Softw. Eng.*, Vol. 39, No. 6, pp. 757–773 (2013).
[5] 亀井靖高，鵜林尚靖：ソフトウェア変更に対するバグ予測モデルの精度評価，*IEICE*, Vol. 111, No. 481, pp. 91–96 (2012).
[6] Kim, S., Whitehead, E. J. and Zhang, Y.: Classifying software changes: Clean or buggy?, IEEE Trans. Softw. Eng., Vol. 34, No. 2, pp. 181–196 (2008).
[7] Mockus, A. and Weiss, D. M.: Predicting risk of software changes, *Bell Labs Technical Journal*, Vol. 5, No. 2, pp. 169–180 (2000).
[8] Breiman, L.: Random forests, *Machine learning*, Vol. 45, No. 1, pp. 5–32 (2001).
[9] Olshen, L. B. J. F. R. and Stone, C. J.: Classification and regression trees, *Wadsworth International Group* (1984).
[10] Ohlsson, N. and Alberg, H.: Predicting fault-prone software modules in telephone switches, IEEE Trans. Softw. Eng., Vol. 22, No. 12, pp. 886–894 (1996).
[11] Khoshgoftaar, T. M. and Allen, E. B.: Modeling software quality with classification trees, *Recent Advances in Reliability and Quality Engineering*, Vol. 2, pp. 247–270 (2001).
[12] Gray, A. R. and Macdonell, S. G.: Software metrics data analysis—exploring the relative performance of some commonly used modeling techniques, *Empirical Software Engineering*, Vol. 4, No. 4, pp. 297–316 (1999).
[13] Xing, F., Guo, P. and Lyu, M. R.: A novel method for early software quality prediction based on support vector machine, Proc. Int'l Symposium on Software Reliability Engineering (ISSRE'05), pp. 10–pp (2005).
[14] Sato, S., Monden, A. and Matsumoto, K.: Evaluating the applicability of reliability prediction models between

different software, Proc. Int'l Workshop on Principles of Software Evolution (IWPSE'02), pp. 97–102 (2002).
[15] Turhan, B., Menzies, T., Bener, A. B. and Di 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).
[16] 木浦幹雄，まつ本真佑，亀井靖高，門田暁人，松本健一：異なるプロジェクト間における Fault-Prone モジュール判別の精度評価，ソフトウェア工学の基礎 XIV，pp. 131–136 (2007).
[17] 蔵本達也，亀井靖高，門田暁人，松本健一：ソフトウェア開発プロジェクトをまたがる fault-prone モジュール判別の試み，*IEICE*, Vol. J95-D, No. 3, pp. 425–436 (2012).
[18] Nagappan, N., Ball, T. and Zeller, A.: Mining metrics to predict component failures, Proc. Int'l Conf. on Softw. Eng. (ICSE'06), pp. 452–461 (2006).
[19] D'Ambros, M., Lanza, M. and Robbes, R.: An extensive comparison of bug prediction approaches, Proc. Int'l Working Conf. Ming Software Repositories (MSR'10), pp. 31–41 (2010).
[20] Hassan, A. E.: Predicting faults using the complexity of code changes, Proc. Int'l Conf. on Softw. Eng. (ICSE'09), pp. 78–88 (2009).
[21] Moser, R., Pedrycz, W. and Succi, G.: A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction, Proc. Int'l Conf. on Softw. Eng. (ICSE'08), pp. 181–190 (2008).
[22] Nagappan, N. and Ball, T.: Use of relative code churn measures to predict system defect density, Proc. Int'l Conf. on Softw. Eng. (ICSE'05), pp. 284–292 (2005).
[23] Koru, A. G., Zhang, D., El Emam, K. and Liu, H.: An investigation into the functional form of the size-defect relationship for software modules, IEEE Trans. Softw. Eng., Vol. 35, No. 2, pp. 293–304 (2009).
[24] Guo, P. J., Zimmermann, T., Nagappan, N. and Murphy, B.: Characterizing and predicting which bugs get fixed: An empirical study of Microsoft Windows, Proc. Int'l Conf. on Softw. Eng. (ICSE'10), Vol. 1, pp. 495–504 (2010).
[25] Shihab, E., Jiang, Z. M., Ibrahim, W. M., Adams, B. and Hassan, A. E.: Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project, Proc. Int'l Symposium on Empirical Softw. Eng. and Measurement (ESEM'10), p. 4 (2010).
[26] Mizuno, O. and Kikuno, T.: Prediction of fault-prone software modules using a generic text discriminator, *IEICE transactions on information and systems*, Vol. 91, No. 4, pp. 888–896 (2008).
[27] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S.: Benchmarking classification models for software defect prediction: A proposed framework and novel findings, *IEEE Trans. Softw. Eng.*, Vol. 34, No. 4, pp. 485–496 (2008).