ソフトウェアエンジニアリングシンポジウム 2019
IPSJ/SIGSE Software Engineering Symposium (SES2019)

深層学習による
プロジェクトを跨いだソフトウェア不具合混入予測

川田秀司[†1]　安田康二[†1]　山下剛[†1]　山元和子[†1]

概要：ソフトウェア構成管理システムのコミット履歴を用い，深層学習により不具合を予測する高精度の手法が提案されており，レビューやテスト，デバッグ等への利用が期待されている．しかしこの手法では予測対象プロジェクト自身のコミット履歴で学習するため，一定数のコミット履歴を持つプロジェクト以外では適用が難しい．今回，我々は複数のオープンソース・ソフトウェア（OSS）のコミット履歴を用いて学習を行い，この学習モデルを社内開発したプロプライエタリ・ソフトウェアのプロジェクトに適用して不具合混入予測を行った．この結果，OSS のみのコミット履歴で学習した場合でも一定の予測精度を得ることができた．これは新規プロジェクトや開始後間もないプロジェクトについても不具合混入予測ができることを示唆している．さらに，複数の OSS のコミット履歴に社内プロジェクトのコミット履歴を一部混ぜて学習させたモデルでは，混ぜないものよりも予測精度が向上することがわかった．
　また本研究では，深層学習の畳込みニューラルネットワークがどの領域に着目しているかを可視化する手法である Grad-CAM を応用することで，予測モデルがソースコードのどの部分を着目して不具合混入と判定したかを可視化し，着目点について考察した．この結果，着目点は複数の単語の組み合わせで決まること，着目される単語は特定のライブラリや API，キーワードであり，それぞれ不具合を起こしやすいものと，不具合を起こしやすいものではないが，不具合箇所周辺に出現しやすいものに分類できることがわかった．また，社内プロジェクトで学習したモデルによる予測では，プロジェクト固有の着目点が含まれていることが分かった．　前述した複数の OSS のコミット履歴に社内プロジェクトのコミット履歴を一部混ぜて学習させたモデルによる予測精度の向上は，この着目点の追加が理由と考えられる．

キーワード：深層学習，不具合混入予測，構成管理システム，オープンソース・ソフトウェア

Cross-Project Software Defect Prediction Using Deep Learning

HIDEJI KAWATA[†1]　KOJI YASUDA[†1]
TSUYOSHI YAMASHITA[†1]　KAZUKO YAMAMOTO[†1]

1. はじめに

ソフトウェアの不具合混入予測は，不具合が含まれる可能性が高い箇所を予測するものである．高い予測精度で不具合が予測できれば，予測に基づいてコードレビューやテストを効率よく行うことが出来る[1]．また，不具合混入予測の多くは各種ソフトウェア・メトリクスを組み合わせた機械学習を行うが，その機械学習モデルによって，不具合を引き起こすソフトウェア・メトリクスの傾向が明らかになれば，その知識を元に今後作成するソースコードでの不具合の予防が期待できる[2][3]．このため非常に多くの不具合混入予測に関する研究が行われているが，従来研究では，データの入手のしやすさから圧倒的にオープンソース・ソフトウェア（OSS）を用いた研究が多く，企業内のプロプライエタリ・ソフトウェアに適用した事例は少ない．

他方で，近年の深層学習技術の発展に伴い，不具合混入予測に深層学習を適用する方法が提案されている[4][5][6]．特に近藤らの研究[6]は，構成管理システムに登録（コミット）されたソースコード片のみを利用して不具合を予測する手法であり，高い予測精度が報告されている．

そこで我々はプロプライエタリ・ソフトウェアに，近藤らの深層学習による不具合混入予測手法を適用し，その有効性を確認したいと考えた．しかし，この方法の評価実験では予測対象のコミット履歴で学習したモデルを利用しており，これが前提条件であるならば新規や開始間もないプロジェクトの様な学習データに利用できるコミット履歴が十分ではないものには利用できない．

そこで今回，不具合混入予測の対象となる社内のプロプライエタリ・ソフトウェアと同一言語の複数の OSS プロジェクトのデータを用いて学習を行い，そのようにして作成した学習モデルを対象ソフトウェアに適用する方法で不具合混入予測を行い，予測精度を評価した．さらに深層学習の畳込みニューラルネットワークがどの領域に着目しているかを可視化する手法である Grad-CAM[7]を本予測手法に応用することで，予測モデルがソースコードのどの部分に着目し判定したかを可視化し，着目点やモデル間の違いを考察した．

以後，まず 2 章では関連する研究について述べ，3 章では不具合混入予測の評価実験で確認したい仮説と，利用するネットワークやデータについて，4 章では評価実験の詳細とその結果について述べ，仮説の成否について考察する．また，5 章では実験の結果に対する可視化の観点からの考察を，6 章ではまとめと今後の課題を述べる．

†1 (株)東芝　ソフトウェア技術センター
Toshiba Corporation

© 2019 Information Processing Society of Japan

2. 関連研究

2.1 深層学習による不具合混入予測

不具合混入予測は 40 年以上にわたり，様々な方法で取り組まれている[8]. それらの取り組みの中でも，構成管理システムへのコミットのタイミングで，コミットするソースコードの中に不具合が含まれるか否かを予測する手法は，"Just-in-Time" の不具合混入予測と呼ばれている．この手法によれば，ソースコードを作成した直後のタイミングで不具合混入の可能性を指摘できるため，開発者は実装したソースコードの記憶が薄れないタイミングで不具合のリスクが高い箇所を確認することができ，効率が良い[9].

近年は深層学習の発展に伴い，"Just-in-Time" の不具合混入予測に深層学習を適用する研究が報告されており，ソフトウエアメトリクスから予測する手法[4]，抽象構文木から予測する方法[5]，ソースコード片そのものから予測する手法[6]等が提案されている.

ここで，ソースコード片を利用する近藤らの手法 [6]は，利用しやすくかつ予測精度が高い．この手法では，コミットコメントを手掛かりにコミットが不具合修正かどうかを判定し，不具合修正だった場合には，修正されたソースコードを挿入したコミットを不具合混入コミットとラベル付け，それ以外のコミットを不具合の無いコミットとラベル付けすることでコミットを分類する．学習では，コミットのソースコード差分にその周囲のソースコードを加えたものを特徴量として，不具合混入コミットかどうかを学習する．この手法には，

- 学習においては，構成管理システムのコミット履歴から人手を介すことなく，学習データを自動的に作成できる
- 予測においては，関連研究で述べたように，ソースコード作成直後のタイミングで不具合混入の可能性を指摘できる
- 高い予測精度が期待できる

等のメリットがある.

一方で，"Just-in-Time"の不具合混入予測は，基本的にコミットの履歴を用いて不具合を予測する手法であるため，コミット履歴が少ない場合には，精度の高い予測を行うことができない．この問題を解決するため，異なるプロジェクトのコミット履歴を用いて学習を行い，履歴の少ない対象プロジェクトの不具合混入予測を行うという手法が研究されている [10][11]．これらの報告によれば，多くのプロジェクトを使って学習させることにより，履歴の少ない対象プロジェクトの予測が一定の精度で行える可能性がある.

3. 新規プロジェクトの不具合混入予測

不具合混入予測を，社内で開発されているプロプライエタリ・ソフトウェアに適用し，予測に基づいてコードレビューやテストを行うことが出来れば，開発期間の短縮や品質の向上を図ることができる．前章で述べた近藤らの提案手法[6]は，適用し易く，かつ予測精度が高い等のメリットがあるが，報告された評価実験は，評価対象プロジェクト自身のコミット履歴で学習したモデルが利用されたものである.

これがこの手法導入の前提条件とするならば，この手法が適応可能なプロジェクトは，

- 構成管理システムが導入された，学習可能な量のコミットが蓄積されたプロジェクト

となるため，構成管理システムが導入されていない場合，もしくは導入後間もない場合，もしくは導入されていた場合でもコミットコメントから不具合修正かどうかが識別できない等の問題がある場合には,利用できないことになる.

しかし，[10][11]の様に，異なるプロジェクトのコミット履歴を用いて学習を行い，履歴の少ない対象プロジェクトの不具合混入予測を行うという手法が，近藤らの提案手法でも利用可能ならば，モデルの学習に予測対象プロジェクトのコミット履歴を使う必要がないため，構成管理システムの運用を開始すれば,開始直後より本手法が利用できる.

そこで，我々は不具合混入予測対象のプロジェクトとは別のプロジェクトで学習を行った場合の予測結果について，以下の様な仮説をたて予測精度の評価を行った.

仮説1. 不具合混入予測対象のプロジェクトとは別のプロジェクトのデータで学習し，予測した場合でも一定の予測精度は確保可能

仮説2. 学習に用いるプロジェクトを増やすほど予測精度は向上

仮説3. 評価対象プロジェクトのデータを学習データに追加することでさらに予測精度は向上

ここで，企業で行われる特定顧客に対するソフトウェア開発では，多くの場合顧客情報の保護のため，プロジェクト間，特に部門を跨るプロジェクト間での開発履歴の利用は難しい．よって．学習に利用する多数のプロジェクトを社内から集めることが困難であるため，学習用のプロジェクトとしては OSS が望ましい．この様な理由により，以後の評価実験では学習には OSS を利用し，社内で開発されたプロプライエタリ・ソフトウェアを評価対象とする.

3.1 ニューラルネットワークと学習データ

近藤らの手法[6]の評価実験では， Git による構成管理を行なっているプロジェクトを対象とする．コミットコメントを手掛かりに Commit Guru [12]によって各コミットに不具合有無のラベルを付与し，各コミットのソースコード差分(diff)にその周囲のソースコードを追加したものを特徴量として学習を行う．不具合予測のフェーズでは，学習の結果得られたモデルに，各コミットにおける特徴量を入力することで不具合混入の有無を予測する.

© 2019 Information Processing Society of Japan

採用するネットワークは Word-Convolutional Neural Network（W-CNN）[6]と呼ばれるネットワークであり，文字列を文字列ベクトルに変換する Embedding 層（埋め込み層）と画像処理で利用される CNN 層を持つことを特徴とする（図 1）.

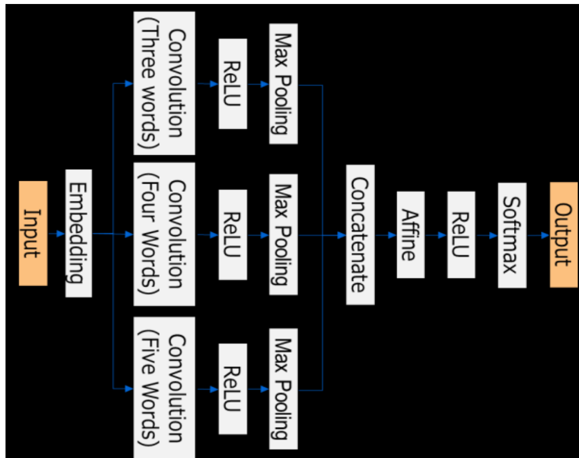

図 1 近藤らのネットワーク構造

本稿では，この近藤らの提案するネットワークに対し若干の変更を加えている.加えた変更は，Batch Normalization 層[13]と Dropout 層[14]の追加，Affine 層の追加である（図 2）. Batch Normalization 層では精度の向上と過学習の抑制，Dropout 層では過学習の抑制の効果が得られた.

更に分割した単語に対し数値をマッピングする方式では，近藤らの手法の「頻出の単語から小さな数値を割り付ける方式」と，「無作為に出現順に数値を割り付ける方式」を試したところ不具合混入予測の精度に差異は認められなかったため，扱いやすい「無作為に出現順に数値を割り付ける方式」を採用した.

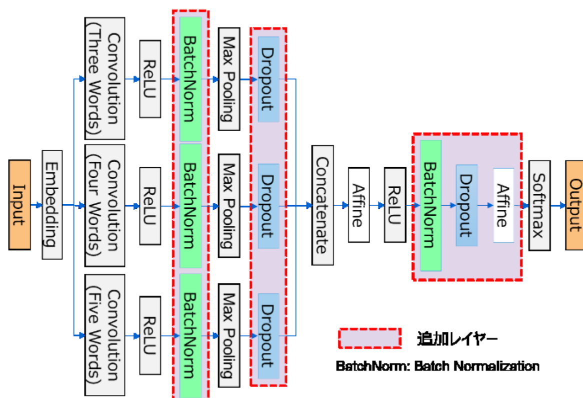

図 2 採用するネットワーク構造

学習データは，近藤らの手法と同様に Commit Guru によりコミットの不具合挿入有無を特定したものを利用し作成した． ただし，プロプライエタリ・ソフトウェアには，日本語のコミットコメントが含まれる場合があるので，Commit Guru の日本語対応等を行った.

これらの変更を適用し，複数の OSS に対し学習・予測を行った結果得られた AUC†を表 1 に示す．C++，Python，Java など言語に関係なく一定の予測精度が確保されている.

† AUC: Area Under the Curve

表 1 本論文で用いる手法での評価結果

プロジェクト名	言語	AUC
Bitcoin	C++	0.72
Flask	Python	0.77
elasticsearch-hadoop (hadoop)	Java	0.75

4. 評価実験

評価に利用するプロプライエタリ・ソフトウェアは，社内で開発している「統合開発環境上でソフトウェアの構造を解析，可視化を行なう」ツールである．このプロジェクトは構成管理システム Git で管理されており，運用についても定められた運用ルールに従い，コミット作成・ブランチ作成・マージや，各コミットに対するコミットコメントの記述等が適切に行われている．また，言語は Java であり，表 2 に示した様に，プロジェクト単体でも深層学習による不具合混入予測が可能な量のコミットがある.

表 2 本論文で評価対象とするプロジェクト PT

プロパティ	値
コミット数	2880
バグ混入コミット数	631
バグ混入コミットの割合	21.9%

4.1 学習に使用するデータ

評価に利用するプロプライエタリ・ソフトウェアとは別のプロジェクトで学習を行った場合に，どの程度の予測精度が出るかを確認するため，GitHub 上で公開されている Java のプロジェクトから 10 個のプロジェクトを利用する．このプロジェクトの選定方法として，月間トレンドに表示されるプロジェクト，つまり活発な活動が行われているプロジェクトから JavaDoc などのドキュメント関連のプロジェクトを除外し，各プロジェクト内でも不具合推定を行えるだけの履歴を持つプロジェクトから選定した.

これらのプロジェクト一覧と，取得時のコミット数を表 3 に示す.

© 2019 Information Processing Society of Japan

表 3 選定した OSS プロジェクト

プロジェクト名	コミット数
elasticsearch-hadoop	1849
Elasticsearch	39074
hibernate-ogm	3597
hibernate-orm	9319
hibernate-search	7396
incubator-dubbo	2603
Jmeter	15892
Jspwiki	7959
Maven	10405
spring-framework	16976

4.2 実験

本評価実験では，以下の実験を行なった．但し，評価用社内プロジェクトを P_T，他の OSS のプロジェクトを P_1～P_{10}（①の結果の F 値の昇順でソートされているとする）とする．

① **評価データ：プロジェクト P（P= P_T or P_n（0≦n≦10））の一部（10%）**
学習データ：P の残り(90%)
評価法：10 分割交差検証
コミット履歴を $c_1, c_2, \dots, c_{10}$ の 10 に等分し，$c_2, c_3, \dots, c_{10}$ を合わせたもので学習し c_1 で検証，$c_1, c_3, c_4, \dots, c_{10}$ を合わせたもので学習し c_2 で検証，$c_1, c_2, c_4, \dots, c_{10}$ を合わせたもので学習し c_3 で検証．．．． という様に 10 回の学習を行い平均を出す検証(10 分割交差検証)を，各プロジェクト毎に行い，予測精度を測定する．

② **評価データ：$P_T, P_1, P_2, \dots, P_{10}$ を混合したものの一部(10%)**
学習データ：$P_T, P_1, P_2, \dots, P_{10}$ を混合したものの残り(90%)
評価法：10 分割交差検証
各プロジェクト P_i のコミット履歴をそれぞれ 10 分割した $c_{i1}, c_{i2}, \dots, c_{i10}$ とし，C_j= “$c_{1j}, c_{2j}, \dots, c_{10j}$ を結合したもの” とする．このとき，$C_1, C_2, \dots, C_{10}$ で Leave-One-Out 交差検証を用い予測精度を測定する．

③ **評価データ：あるプロジェクト P_n（n=1,3,5,7,9）**
学習データ：P_1, P_3, P_5, P_7, P_9 から P_n を除いた 4 つのプロジェクト
評価法：プロジェクト単位の Leave-One-Out 交差検証
①で用いた OSS について，F 値でソートし奇数順位のみを用いプロジェクト単位の Leave-One-Out 交差検証を行い，同一のプロジェクト内（以降，自プロジェクト）のデータが学習データに含まれない状態での予測精度を測定する．

④ **評価データ：あるプロジェクト P_n（1≦n≦10）**
学習データ：P_1~P_{10} から P_n を除いた 9 つのプロジェクト
評価法：プロジェクト単位の Lave-One-Out 交差検証
学習データが増えることによる効果を検証するため，取得した OSS をすべて用いた Leave-One-Out 交差検証を行い③と比較する．

⑤ **評価データ：社内プロジェクト P_T**
学習データ：10 個のプロジェクト P_1～P_{10}
評価法：１度実行
OSS のコミット履歴のみで学習を行ったモデルを用い社内プロジェクトの不具合をどの程度予測できるかを測定する．

⑥ **評価データ：社内プロジェクト P_T の一部(10%, 33%, 67%, 90%)**
学習データ：P_T の残り＋10 個のプロジェクト P_1～P_{10}
評価法：10%，90%は 10 パターン，33%，67%では 3 パターンで交差検証
OSS のコミット履歴に社内プロジェクトのコミット履歴を混ぜたもので学習する．混合率を増やすことで予測精度が向上するかを確認する．パーセンテージは社内プロジェクトの全コミット履歴のうち，学習に利用する割合を示す（10%：10 分割中の 1 つ，90%： 10 分割中の 9 つ，33%：3 分割中の 1 つ，67%：3 分割中の 2 つを学習データに利用したもの）．

①は評価対象プロジェクト自身のコミット履歴で学習したモデルでの推論実験を，各プロジェクト毎に行ったもので，②以降の実験の精度と比較するためのものである．

②は複数のプロジェクトを混合したデータによる学習がうまく行えるかを確かめるための実験である．

③④は，**仮説 1**，**仮説 2** を確かめるための実験で，混合するプロジェクトが 4 の場合と 9 の場合の予測精度を求める．これらの実験で一定の予測精度が確認できれば，始まって日が浅いプロジェクトに対しても深層学習を用いた不具合混入予測を適用可能と考えられる．また，④の予測精度が③の予測精度よりも高ければ混合するプロジェクトを増やすことで予測精度を向上できると考えられる．

⑤⑥は**仮説 3** を確かめる実験であり，⑤では対象プロジェクトのデータを混ぜない場合，⑥では対象プロジェクトのデータを混合した場合（混合率は 10%，33%，67%，90% の 4 パターン）の実験を行い予測精度を比較する．但し，学習に長い時間がかかるため対象プロジェクトは P_T のみに限定して実験する．これらの実験で，混合率を増やすことで予測精度が向上するならば，一定期間毎に予測対象プロジェクトのコミット履歴を加えて，学習モデルを更新してくことで予測精度が向上する可能性があり，社内のプロジェクトに対してもそういった適用が可能と考えられる．

4.3 結果

前節で提示した実験項目に対する結果を下に示す．

表 4 実験①，③，④の結果

プロジェクト名	①の AUC	①の F 値	③の AUC	③の F 値	④の AUC	④の F 値
社内プロジェクト（P_T）	0.74	0.65	-	-	-	-
Jmeter（P_1）	0.75	0.50	0.69	0.40	0.71	0.43
spring-framework（P_2）	0.71	0.54	-	-	0.69	0.49
incubator-dubbo（P_3）	0.75	0.54	0.67	0.41	0.70	0.48
hibernate-search（P_4）	0.74	0.58	-	-	0.69	0.49
hibernate-ogm（P_5）	0.75	0.63	0.68	0.55	0.69	0.56
Jspwiki（P_6）	0.74	0.64	-	-	0.66	0.51
Maven（P_7）	0.76	0.65	0.70	0.57	0.71	0.59
elasticsearch（P_8）	0.73	0.66	-	-	0.67	0.60
elasticsearch-hadoop（P_9）	0.75	0.72	0.70	0.68	0.71	0.68
hibernate-orm（P_{10}）	0.77	0.72	-	-	0.71	0.64
平均	**0.73**	**0.60**	**0.68**	**0.52**	**0.69**	**0.55**

①の結果（表 4，図 3，図 4）では，全体の平均 AUC は 0.73，F 値‡は 0.60 であり，各プロジェクトに対し，一定の予測精度が得られていることが分かる．

②結果（表 5）では，複数の OSS プロジェクトを混合したデータで学習したモデルを用い，複数の OSS プロジェクトを混合したデータに対する予測精度は，AUC は 0.74，F 値は 0.63 となり，①と同様に一定の予測精度が得られている．このことから，複数のモデルを混合しても単一プロジェクトでのモデル（①のモデル）と同様に，学習および予測が機能することを示している．

表 5 実験②の結果

AUC	F 値
0.74	0.63

また，③④の結果（表 4，図 3，図 4）では，複数の OSS プロジェクトによる学習モデルを用いて，学習データに含まれないプロジェクトのデータを予測させた場合には①②の場合よりは予測精度は落ちるが，それでも AUC はそれぞれ 0.68，0.69，F 値は 0.52，0.55 と一定の予測精度は得られる．この際，③④を比較すると elasticsearch-hadoop を除き AUC，F 値とも向上している（elasticsearch-hadoop では AUC は向上，F 値は同じ）ことから，学習データに混合するプロジェクト数が多いほど予測精度が向上すると予想できる．

‡ F 値 ＝2 / (1 /適合率 ＋1/再現率)

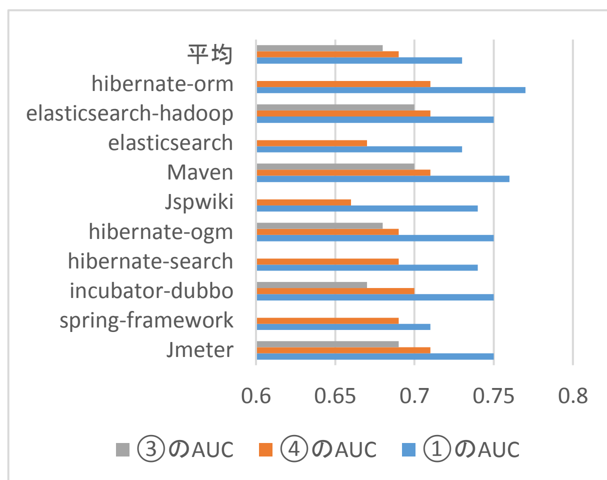

図 3 ①，③，④の AUC

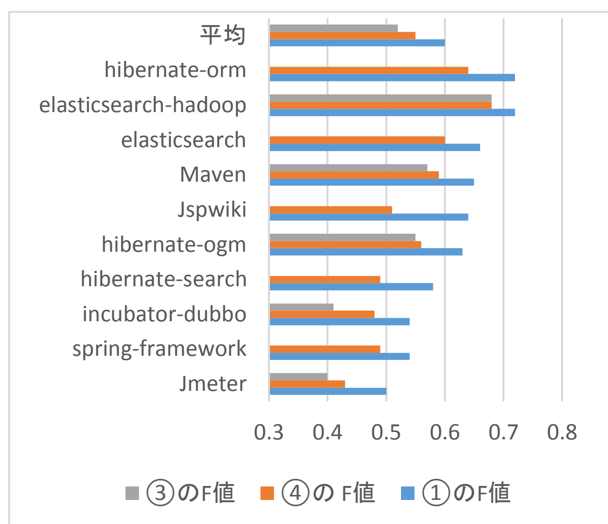

図 4①，③，④の F 値

⑤⑥の結果（表 6，図 5）ついては，

i. OSS で学習したモデルを用いて，社内プロジェクトに対して不具合混入予測を行なっても OSS に対して予測した場合と同等の予測精度が得られる．
ii. OSS のデータに社内プロジェクトを混合させて学習させた場合，混合率が高いほど予測精度がよくなる．

ということがわかる．よって，i の結果より，新規開発のプロジェクトでも，プロジェクト開始直後から他のプロジェクトで学習したモデルを用いて不具合混入予測を運用開始することができ，かつ，ii の結果より，運用が進むに従い得られる自プロジェクトのデータを混合させることで予測精度の向上が可能であると考えられる．

© 2019 Information Processing Society of Japan

表 6 実験⑤，⑥の結果

社内プロジェクトの混合割合	AUC	F 値
⑤（0%混合）	0.66	0.52
⑥（10%混合）	0.67	0.53
⑥（33%混合）	0.68	0.56
⑥（67%混合）	0.71	0.60
⑥（90%混合）	0.71	0.61

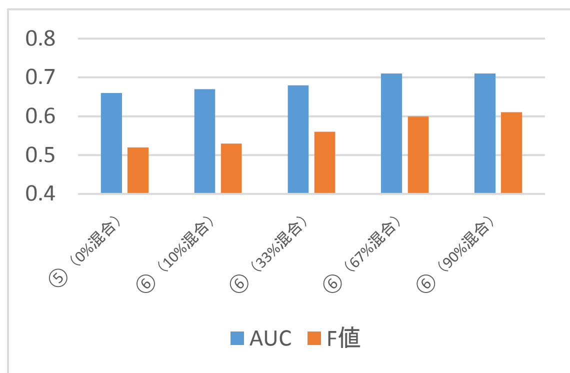

図 5 ⑤，⑥の結果

今回の実験は，以下の仮説の評価のため行った．

仮説1. 不具合混入予測対象のプロジェクトとは別のプロジェクトのデータで学習し，予測した場合でも一定の予測精度は確保可能

仮説2. 学習に用いるプロジェクトを増やすほど予測精度は向上

仮説3. 対象プロジェクトのデータを学習データに追加することでさらに予測精度は向上

実験結果は全て仮説を肯定する結果になっている．但し，実験⑤⑥は評価対象が 1 つの社内プロジェクトのみの結果であるため，特に仮説 3 の信憑性については更に実験による評価をすべきだろう．また，ここでは 10 個の OSS を選択したが，この選択により結果に違いが現れる可能性もある．

実験⑤⑥の結果から，不具合混入予測対象のプロジェクトとは別のプロジェクトのデータで学習したモデルに対し，予測対象のプロジェクトのデータで追加学習を行うことで，予測精度を高めていくことも可能と思われる．これについては，今後の課題としたい．

5. 可視化と考察

前章では，新規プロジェクトや，開始間もないプロジェクトであっても，不具合混入予測手法が適用可能かを調べるため，いくつかの実験を行い，その結果を示した．本章では，これらの実験の中で，自プロジェクトのコミットデータで学習したモデル（以後，自学習モデルと呼ぶ）と，他の 10 プロジェクトのコミットデータで学習したモデル（以後，他学習モデルと呼ぶ）に着目し，２つのモデル間での不具合混入予測の違いについて考察する．

社内プロジェクトの自学習モデルによる予測（①の実験，10 分割交差検証の平均）では AUC 0.74，F 値 0.65，他学習モデルによる予測（⑤の実験）では AUC 0.66，F 値 0.52 であり，自学習モデルほどではないが，他学習モデルでも一定の予測精度が得られている．不具合を正しく予測したコミットに着目すると，真の不具合 518 コミットのうち，両モデル共通で不具合と予測したコミット数が 228，自学習モデルのみで不具合と予測したコミット数が 272，他学習モデルのみで不具合と予測したコミット数が 3 であった．

以後，これら２つのモデルの予測結果について，特に差異の有無を確認し，もし差異があるならその差異について考察を行う．

5.1 Grad-CAM 手法を応用した着目点の可視化手法

考察は，予測結果について，プロジェクト従事者と共に確認を行い，その着目点を分類することにより行う．このため，着目点を明確にする必要であるが，この方法として，Grad-CAM 手法[7]を応用した可視化手法を利用する．

Grad-CAM は，画像判定を行う CNN（Convolutional Neural Network）の最後の Covolution 層の勾配を，画像上でヒートマップとして色付けすることで，モデルが着目する部分を可視化する手法である．

本稿で利用するモデルでも文字列の並びを画像に見立て，3 つの Convolution 層で並列に処理し，結果となる勾配を結合している．よって，各 Convolution 層での勾配を 0〜1 に正規化後，各文字列に対応する各層の値の最大値をその文字列の着目度とする．最後に着目度をヒートマップ化し色付けすることで着目される文字列を可視化できる．

図 6 にこの方法で可視化した例を示す．着目されていない部分は青く，着目される度合いが上がるにしたがって水色，黄色，オレンジ，赤と変化する．例では node や Selector，ast が特に注目されている場所となる．

```
+    IASTTranslationUnit ast = astmanager.getAST(tu, path);
+    IASTNodeSelector  nodeSelector  =  ast.getNodeSelector(null);
+    IASTNode  astNode  =  nodeSelector.findEnclosingNode(offset,  1);
+
+    return  astNode.getFileLocation().getStartingLineNumber();
```

図 6 可視化例

5.2 着目点の分類と考察

本節では着目点の分類と考察について記す．分類と考察は以下の手順で行う．

(1) 両モデルにおいて不具合を正しく予測したコミット 228 個を，それぞれ Grad-CAM を応用した手法により可視化し，着目点を抽出

© 2019 Information Processing Society of Japan

(2) (1)の結果をプロジェクトの従事者と我々で確認・比較
(3) (2)を考慮し着目点の分類の観点を抽出
(4) 観点に従って実際に着目点を分類
(5) 各モデルの予測結果における着目点の差を，分類結果の違いを元に考察

まず，着目点の確認・比較により判ることは，両モデル双方で不具合混入と予測されたものでも，例えば，自学習モデルでの着目点は，自プロジェクト固有のライブラリやAPI・キーワードが着目される箇所だが，他学習モデルでは標準ライブラリのキーワードが着目されている等，着目される箇所は多くの場合異なることである．

また，着目点が必ずしも不具合そのものを示しているものではないことも判る．例えば，自学習モデルでは，TODO や FIXME といったキーワード付近は着目点となる場合が比較的多いが，これらは不具合そのものではなく，プログラマーが仮のコードと共に「後に修正が必要となる」という覚え書きの意味で記述するキーワードである．このキーワード周辺は多くの場合，後に修正されることになるため，着目点となっていると考えられる．

以後，不具合そのものを表していると考えられる箇所を不具合箇所，TODO や FIXME のように不具合そのものを表しているわけではないが不具合に付随して現れる箇所を不具合想起箇所と呼ぶ．

さらに，着目点には共通のキーワードが現れる傾向があるが，逆に，そのキーワードが現れる箇所が着目点になっている確率は必ずしも高くない．例えば dialog という文字列は，全体として 41 箇所で 0.6 以上の着目度であり，着目される文字列の中では 6 番目に多い．しかし，dialog の全出現箇所は 1510 箇所であり，大半の出現箇所（1449 箇所）は着目度が 0.6 未満となっている．このことから着目点は，単にキーワードのみから決まるわけではないことがわかる．例えばキーワードの ast は，図 6 のように Node，Selector が近辺に現れる場合に着目箇所になっている場合が多い．よって，着目箇所は「不具合と同時に現れがちな，複数のキーワード」で同定されると考えられる．

観点１：
- ライブラリ・API 名
- その他キーワード

観点 2：
- 不具合箇所
- 不具合想起箇所

観点 3：
- プロジェクト固有
- 一般的

図 7 分類の観点

これらを考慮しながら，プロジェクト従事者と我々が抽出した着目点の分類観点を図 7 に示す．

この 3 つの観点に従い，着目点に対して意味づけ，分類を行ったものを表 7 に示す．自モデル，他モデルの列では，実際にその単語の組み合わせを着目点として分類したモデルに○を付けた．例えば，「Plugin，Properties」は自モデル，他モデルの両方で注目点になっている．

一部に，着目理由が不明なものもあったが，概ね着目理由は推測することができた．

この観点による分類結果により，

i. 着目点は，複数の単語の組み合わせから決まる
ii. 概ね着目の理由については推測でき，着目部分は，
  a) 不具合修正が起こりやすいライブラリや API
  b) 不具合と同時に現れるライブラリや API
  c) 不具合と絡みやすいキーワード(width, weight 等)
  d) 不具合と同時に現れるキーワード(FIXME, TODO 等)
※ 表 7 を参照
iii. 両モデルの着目点は，必ずしも一致しない．特にプロジェクト固有のライブラリや API，キーワードは自学習モデルのみに含まれる

ということがわかった．

さらに，他学習モデルでの予測でも，一定の予測精度は得られるが，当然のことながら着目点には自プロジェクト固有のライブラリや API，キーワードは含まれない．また，例えば，ある標準ライブラリの利用法に問題があって不具合が多発しているなど，自プロジェクト固有の傾向も反映されない．実験⑤⑥で他のプロジェクトのデータに自プロジェクトのデータを混合させることで予測精度が向上するのは，これらが反映されることが理由と考えられる．

6. おわりに

本稿では，近藤らの手法をベースに，不具合混入予測対象のプロジェクトとは別の複数のプロジェクトのデータで学習を行なったモデルを用い，対象プロジェクトの予測を行った場合でも一定の精度が得られることを示した．これは，新規もしくは，開始して間もないプロジェクトに対しても，不具合の予測を行なえることを示唆している．

また，学習に利用するプロジェクトの数は多いほうが予測精度は向上した．

さらに，不具合混入予測対象のプロジェクトとは別の複数のプロジェクトデータに，対象プロジェクトのデータを混合して学習することで，予測精度を向上させられることを示した．

また，Grad-CAM の技術を W-CNN に応用して予測の着目点を可視化することにより，着目点は複数の単語の組み合わせで決まることを示した．着目される単語は，特定のライブラリや API，キーワードであり，それぞれ不具合を起こしやすいものと，不具合を起こしやすいわけではない

が，不具合を起こす箇所周辺に現れる場合が多いものに分類できる．さらに，自プロジェクトのデータによる学習モデルではプロジェクト固有の着目点が加わるため，他プロジェクトのデータに自プロジェクトのデータを混合して学習させることで予測精度の向上が図れると考えられる．

最後に，本稿では，自学習モデル，他学習モデル，他プロジェクトのデータと自プロジェクトのデータを混合したデータによる学習モデルの予測精度を測定し比較したが，社内プロジェクトに対する実際の運用では，自プロジェクトのデータが増える毎に，データを混合しなおして最初から学習を行なうのは運用コストが高い．しかし，他学習モデルに対し，自プロジェクトのデータの増加分で追加学習を繰り返すことで同様の結果が得られるならば運用コストを大幅に低減させることができる．よって，追加学習による予測精度の確認を今後の課題として挙げた．

表 7 着目点の分類

観点 1	観点 2	観点 3	着目点の単語の組み合わせ：単語の由来	自モデル	他モデル
ライブラリ，ライブラリ API	不具合箇所(a)	プロジェクト固有	**Function, Block**: プロジェクト内で作られたライブラリ(不具合修正が多かったライブラリ)	○	
		一般的	**Ast, Node, Selector**: Eclipse で提供されるライブラリ		○
			Xml, Transient,Contents, Chenged: JavaBean の xml 関係の API		○
			Property,Value : JavaBean の xml の値操作の API	○	
			Entity, Info: JavaBean の xml 関係の API		○
			Install, Edit, Policy,Feedback,Role, Figure, Canvas: Eclipse で提供されるライブラリ	○	○
			Plugin，Properties: Eclipse のプラグイン関係のライブラリ等	○	○
	不具合想起箇所(b)	プロジェクト固有	**Exception, Dialog**: 例外時に表示(表示内容の変更は多い)	○	
		一般的	**System.out.println**: デバッグプリントで多用（修正時に削除)	○	
その他キーワード	不具合箇所(c)	プロジェクト固有	—	—	—
		一般的	**Width, Size, Bound**: 窓，グラフィック表示関係で使用する単語		○
			Circle, Line, Size: グラフィック表示関係で使用する単語		○
			Csv, Line, Equal: csv 出力関係で使用する単語	○	
	不具合想起箇所(d)	プロジェクト固有	**FIXME, TODO, LOCK**: 修正箇所が必要な箇所に入れるコメントキーワード（修正時に削除)	○	
		一般的	—	—	—

参考文献

1 Munson, J. C., and Khoshgoftaar, T. M., The Detection of Fault-Prone Programs. IEEE Transactions on Software Engineering. 1992, vol. 18, no. 5, p. 423-433.
2 Cataldo, M., Mockus, A., Roberts, J. A., and Herbsleb, J. D., Software Dependencies, Work Dependencies, and Their Impact on Failures. IEEE Transactions on Software Engineering. 2009, vol. 35, no. 6, p. 864-878.
3 McIntosh, S., Kamei, Y., Adams, B., and Hassan A. E., The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. Proc. Int’l Working Conf. on Mining Software Repositories. 2014, p. 192-201.
4 Wang, S., Liu, T., and Tan. L., Automatically Learning Semantics Features for Defect Prediction. Proc. 38th International Conference on Software Engineering. 2016, p. 1558-1225.
5 Yang, X., Lo, D., Xia, X., Zhang, Y., and Sun, J., Deep Learning for Just-in-Time Defect Prediction. Proc. 2015 IEEE International Conference on Software Quality, Reliability and Security. 2015, p. 17-26.
6 近藤 将成, 森 啓太, 水野 修, 雀 銀恵, 深層学習によるソースコードコミットからの不具合混入予測. 情報処理学会論文誌. 2018, vol. 59, No. 4, p. 1250-1261.
7 Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D., Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. Proc. The IEEE International Conference on Computer Vision. 2017, p. 618-626.
8 Kamei, Y., and Shihab, E., Defect Prediction: Accomplishments and Future Challenges. Proc. 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering. 2016, p. 33-45.
9 Kim, S., Whitehead E. J., and Zhang, Y., Classifying Software Changes: Clean or Buggy?. IEEE Transactions on Software Engineering. 2008, vol. 34, No. 2, p. 181-196.
10 Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., and Ubayashi, N., An Empirical Study of Just-in-Time Defect Prediction Using Cross-Project Models. Proc. Int’l Working Conf. on Mining Software Repositories. 2014, p. 172-181.
11 Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., and Hassan, A. E., Studying Just-in-Time Defect Prediction Using Cross-Project Models. Empirical Software Engineering. 2016, vol.14, no. 5, p. 2072–2106.
12 Rosen, C., Grawi, B., and Shihab, E., Commit Guru : Analytics and Risk Prediction of Software Commits. Proc. 10th Joint Meeting on Foundations of Software Engineering. 2015, p. 966-969.
13 Ioffe, S., and Szegedy, C., Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proc. Int’l Conference on Machine Learning. 2015, p. 448-456.
14 Srivastava, N., Hinton. G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R., Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014, vol. 15, p. 1928-1958.

© 2019 Information Processing Society of Japan