Vol.2022-SE-210 No.9
2022/3/11

複数の学習済みモデルを用いた 業務ソフトウェアのバグ予測精度の比較

内藤大貴 [1,a)]　上田賀一 [1,b)]

概要：バグ予測の精度向上はテストの効率化にとって重要であり，バグ予測の研究は盛んに行われる．多くの研究の対象はデータ収集しやすいオープンソースソフトウェア（OSS）を対象にしたものである．本研究は，VB.NET で開発された業務ソフトウェアに対し，実用的な学習モデルの構築を行いデバッグ作業の効率化を目的とする．本研究では，バグ予測の学習モデルに畳み込みニューラルネットワークを用い，メインシステムと複数のサブシステムから構成される医療業務用システムを対象とする．各システムのバグデータ数は少ないため，各システムで浅い学習を行った学習済みモデルを用意し，多数決を採ることで，過学習を抑え，精度の高い分類を行うことが可能か検討した．別々のシステムのデータを一括にまとめて学習を行った場合の精度と比較した．単体のシステムで突出して上手く学習できたモデルが無い場合は，多数決を採る方法が高精度であることが捉えられた．

キーワード：バグ予測，業務ソフトウェア，畳み込みニューラルネットワーク，機械学習

Comparison of bug prediction accuracy of business software using multiple trained models

Hiroki Naitoh[†1]　Yoshikazu Ueda[†1]

Abstract: Improving the accuracy of bug prediction is important for improving the efficiency of testing, and research on bug prediction is actively conducted. Most of the researches target open source software (OSS), which is easy to collect data. The purpose of this study is to construct a practical learning model for business software developed in VB.NET to improve the efficiency of debugging. In this study, we use convolutional neural networks as the learning model for bug prediction, and target a medical business system consisting of a main system and several subsystems. Since the number of bug data in each system is small, we prepared a trained model with shallow training in each system, and examined whether it is possible to suppress overtraining and classify with high accuracy by adopting majority voting. We compared the accuracy with that obtained by combining the data of different systems into a single training model. The results showed that the majority voting method was more accurate when there was no model that could be trained outstandingly well in a single system.

Keywords: Bug prediction, Business software, Convolutional Neural Network, Machine Learning

1. はじめに

1.1 研究背景

近年の情報社会では，多くの企業によってソフトウェアが構築されている．構築されるソフトウェアの規模は様々だが，規模が大きくなる程ソフトウェアに掛かるコストは膨大になる．例えば，デバッグ作業はテストを行い不具合の起きた個所をソースコード中から探し出し修正を行う．ソフトウェアの規模が小さく複雑でなければ，容易に修正を行うことができる．しかし，大規模なもので複雑な構造を持つソフトウェアのテストやデバッグ作業には多大なコストと時間が掛かっている．よって，この一連の作業を効率化することは，ソフトウェア品質の保証や保守・運用にとって重要である．そこで，テストやデバッグ作業の効率化を図るため，近年ではバグ予測の研究が盛んに行われ，本来，人の手によって探されるバグを，機械学習等を用いて予測する手法が採られている．その中でも，畳み込みニューラルネットワーク（CNN）を用いた学習モデルでの分類は高い精度で報告されている．しかし，殆どの研究はオープンソースソフトウェア（OSS）を対象に行われており，ソースの公開が行われないためデータの収集の難しさから，業務ソフトウェアを対象としたものは少ない．さらに，対象としている言語も Java 等のものが多く，本研究で取り上げた VB.NET を対象とした研究は見当たらなかった．

1.2 関連研究

バグ予測精度の向上は，ソフトウェアテストやソフトウェア保守の効率化に大きな恩恵をもたらす．そこで，最近の研究では機械学習や深層学習を用いたバグ予測モデルの研究が行われている．近藤らの研究「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」[1] は，ソフトウェアの変更が行われた時にその変更によって不具合が起きるかどうかを予測する手法である．全体のソースコードからメトリクスを計算し機械学習に適用するのではなく，変更があったソースコード片のみに対して深層学習を適用するというもので高い精度を出している．しかし，この研究の対象ソフトウェアはオープンソースソフトウェアであり，企業の開発した業務ソフトウェアではない．

1 茨城大学大学院
Ibaraki University Graduate School
a) 20nm724f@vc.ibaraki.ac.jp
b) yoshikazu ueda.se@vc.ibaraki.ac.jp

本研究と同じ業務ソフトウェアを対象とした中庭らの研究「ソースコード片を用いた深層学習による業務ソフトウェアの不具合予測」[2]は，近藤らの研究[1]と同じように学習データにソースコード片を用いている．仕様変更によって変更が加えられた箇所と，不具合修正によって変更が加えられた箇所で区別を付け，なにも修正が無いものを含めた 3 値分類をしている．学習には畳み込みニューラルネットワークを用いており，分類精度は高い成果を出している．

前述のように業務ソフトウェアはデータの収集が難しくバグのデータ量も少ないことが多い．本研究の対象ソフトウェアにおいてもバグのデータ数が 1 つのシステムでは最大 3000 個程と少ない．よって，構築する学習モデルもすぐに過学習を起こしてしまうため，深い学習モデルの構築が難しいという問題がある．中庭らの研究[2]では，最もバグのデータが多いメインシステムを対象としていたが，バグ数は 1000 個程と少なくすぐに過学習を起こしてしまっていた．そのため，データ数が少なくても過学習を抑えつつ精度を向上させることが課題の 1 つであった．宮本らの研究にバグ予測精度向上に向けた個人化予測モデルの組み合わせ手法[3]がある．これは，コミット数の多い開発者複数人でそれぞれの個人化バグ予測モデルを構築し，そのモデルでコミット数の少ない開発者のバグ予測を行い多数決を採るといったような手法である．個人化予測モデルは，単体では別の開発者のバグ予測への代用は難しいが複数の個人化予測モデルの結果を統合した場合，バグ判定の閾値によってはまとめて学習をした際と同等以上の精度が報告されている．

本研究では，開発者ではなくシステムごとにバグ予測モデルを構築し分類結果で多数決を採ることによって，単体のモデルで分類をするよりも過学習を抑えながら高い精度が出せるのではないかと考えた．そこで，本研究ではメインシステムのみで学習して分類するのではなく，複数あるサブシステムのバグデータそれぞれで浅い学習をしたモデルも用意し，これら複数のモデルを用いた分類結果で多数決を採る手法を検討した．

1.3 研究の目的

本研究では，実際に企業で VB.NET で開発され，現在も現場で使われている大規模ソースコードから機械学習を用いてバグ予測を試みる．機械学習には高い分類精度を期待できる畳み込みニューラルネットワークを用いる．業務ソフトウェアはバグのデータの収集が難しいためデータ数が少ないことが多い．そのため，データ数が少なくても過学習を抑え高精度な分類結果を出せるのが望ましい．そこで，本研究では同一ソフトウェアのサブシステム 14 個を用いてそれぞれで浅い学習をする．その学習済みモデルとメインシステムの学習済みモデルを用いて多数決を採ることにより，単体で分類をした際と精度にどのような差が出るのか比較する．また，複数のシステムのデータをまとめて学習したモデルとの精度の差も比較することで，対象ソフトウェアに対し実用的なバグ予測モデルであるかどうかを検討する．

本報告では，以降に，まず機械学習に利用するための教師データの内容と，それを利用して用意する学習済みモデルについて述べる．次に，用意した複数の学習済みモデルを組み合わせたものを用いて実験を試みる．さらに，その分類結果と精度を比較する．最後に実験結果の考察と今後の課題について述べ，全体のまとめと結論を述べる．

2. 教師データと学習済みモデル

業務ソフトウェアで長期間稼働しているものには，リリース当時の開発者などが退職していてシステム全体を把握している者が社内に残っていない場合がある．本研究の対象ソフトウェアも長期間稼働しており，加えて開発時の設計書などが存在しない．ソースコードからバグ予測が可能であれば，そのようなソフトウェアでもソースコードは存在するため，より広いソフトウェアに適用可能である．さらに，ソースコード全体を学習に用いるのではなく，必要な部分を抽出しソースコード片として学習に用いることで学習に掛かる時間を削減することができる．今回対象としているソフトウェアの規模は総行数約 250 万行でクラス数は約 2600 個規模であるため，ソースコード全てを学習に用いるのは難しい．そこで，本研究のバグ予測モデルの学習にはソースコード片を用いる．ここでは，学習に用いる教師データと学習モデルの構築から学習までについて述べる．過学習を抑えつつ更なる精度の向上を目的に，複数の学習済みモデルを組み合わせた分類をするため学習済みモデルを複数個用意する．

2.1 ソースコード片

ソースコード片とは，ソースコード全体から任意の場所を切り取り 1 つのデータとしたものである．本研究で利用するソースコード片は，対象としたソースコード全体からバグの出現した箇所の前後 3 行を抜き出したものと，バグの存在しない箇所の最大 30 行となるように抜き出したものである．バグを含むソースコード片は，バグの修正履歴からバグ ID を抽出し，バグ ID をもとに修正が行われた箇所と前後 3 行をまとめたものを抜き出した．バグ ID とはバグの修正を行ったときに与えられるバグの識別番号のことである．本研究で利用するバグを含むソースコード片をバグのある箇所の前後 3 行含めたものと設定したのは，近藤らの研究[1]により有効性が示されているためである．この研究では，0 行から 10 行の幅でコメント部分を除いた文脈行数の不具合予測への影響を測定している．この結果は，対象となるソフトウェアは異なっているが，本研究では今回はこの研究結果を採用し前後 3 行とした．

また，中庭らの研究[2]ではメインシステムのソースコー

ド片のみ抽出したが，本研究では複数のサブシステムも対象とする．よって，サブシステムでもバグを含むソースコード片の抽出を行った．なお，バグを含まないソースコード片に関してはメインシステムから抽出したものを利用している．表 1 に抽出したバグを含むソースコード片のシステム毎の数を示す．バグのデータ数は最大でもメインシステムである ER の 3838 個と少ない．さらに，サブシステムも同じ業務ソフトウェアのシステムでありメインシステムと似ているものが多い．例として，ERJR は ER と規模もほぼ同じであり構造も似ているため抽出されたソースコード片も ER のものと同じものが多く存在する．しかし，名称 Eiyo で始まるサブシステムは比較的に他のシステムとは性質が異なっている．これらのソースコード片を用いて学習を試みる．

表 1　各ソリューションのバグを含むソースコード片の数

solution	bug files
ER	3838
ERJR	3719
ERJB	3667
EiyoER	2883
EiyoMap	2846
ERJ	2792
ERJY	2782
EiyoJEMaster	2271
ER3CL	2106
EiyoJE	1922
ERMaster	1692
ERKTRef	1534
ERKDAI	465
ERMAILKB	103
ERLblPrt	38

2.2 データの前処理

ここでは，得られたソースコード片を機械学習に与えるために行ったデータの前処理について述べる．初めに，作成したソースコード片から不要な情報を削除し正規化を行った．ここでの不要な情報はコメント部分や「=」などの演算子や int 型の数値,「:」といった記号を指す．近藤ら[1]は，コメント行をソフトウェアの動作に影響を及ぼさないため不要な情報として扱っている．本研究でも，コメントに着目したモデルを構築していないので除去を行った．演算子や記号も同様に学習には不要な情報であると考え除去した．ソースコード片の抽出と正規化には Visual Studio に搭載されている .NET コンパイラープラットフォームである Roslyn[4]の CodeAnalysis を用いた．CodeAnalysis は C# と visualbasic で構文解析を行うことができる．この機能を利用しソースコード片の抽出と正規化を行った．

機械学習にデータを与えるためには String 型のままでは扱うことができないため，単語毎に Int 型の値を与えた辞書を作成した．この辞書は，全てのシステムのソースコード片に出現する単語全てに値を与え作成した．さらに，機械学習に与えるデータは固定長である必要がある．そのため，正規化を行ったデータの長さを 512 に揃えるためにパディングを行った．パディングは，全ての配列の長さを指定した値に揃えて足りない分だけ 0 を補いデータ長を揃えることである．長さを 512 と設定したのは，計算時間を短縮することと，本研究が扱うデータの 90%以上のデータ長が 512 以下であることに依る．

2.3 学習済みモデルの準備

ここでは機械学習のモデルの構築を行う．本研究では畳み込みニューラルネットワーク（CNN）を使用する．CNN は，畳み込み層とプーリング層を持ったニューラルネットワークである．CNN の構築には，Google が開発した機械学習のライブラリである Tensorflow[5]をバックエンドに利用したニューラルネットワークライブラリの keras[6]を用いた．更に，今回利用した畳み込みニューラルネットワークは 1 次元のものを利用した．構築した畳み込みニューラルネットワークのモデルについて述べる．畳み込みニューラルネットワークは，他のニューラルネットワークとは違い全結合層だけでなく，畳み込み層とプーリング層から構成されるネットワークである．畳み込み層とプーリング層は次のような役割を持つ層である

本研究では，embedding 層，畳み込み層,プーリング層，2 つの隠れ層からなるネットワークモデルを構築した．embedding というのは，データの前処理で String 型から Int 型に変換したデータをさらに固定次元の密ベクトルに変換する層である．畳み込み層の作成には，keras の Conv1D レイヤーを用いた．このレイヤーは，入力値を与えるだけで 1 次元の畳み込みを自動で行い，プーリング層に渡すための値を出力するレイヤーである.プーリング層の作成には，keras の MaxPooling1D レイヤーを用いた．このレイヤは，1 次元データのためのレイヤであり，Pooling 層を自動で生成するレイヤである．今回使用した教師データと学習モデルのパラメータを表 2 に示す．ここで設定したパラメータは，筆者が手動で出来るだけ精度が高くなるように設定したものである．よって，パラメータ設定においては，より適した設定が存在する可能性がある．

一度の実行で何回繰り返し学習を行うのかを示す epoch 数は，1 から 15 までの範囲で変化させて精度を比較した．その結果を表 3 に示す．1 列目は epoch を表し，それ以外はシステム毎のテストデータを分類した accuracy である．作成したデータの 9 割を訓練データ，残りの 1 割をテストデータとして学習を行い分類している．本研究では，過学

習を抑えつつ高い精度を出すことを目的の 1 つに設定している．表 3 を見ると，epoch が 3 から 5 程度でも十分な精度が出ており accuracy の上り幅も少なくなっているため，今回の実験では epoch を一律 5 と設定して学習した．
これらのパラメータでモデルを構築し，メインシステムである ER と他 14 個のサブシステムでそれぞれ学習を行い，学習済みモデルを 15 個用意した. 学習に利用したデータは前処理で得られたデータである．また，バグを含むデータと含まないデータで数に偏りが生じないように，バグを含むデータ数に合わせ，含まないデータ利用した．例えば，メインシステムのバグデータ数は 3838 個なのでバグを含まないデータも 3838 個を用いた.

表 2　教師データと学習モデルのパラメータ

パラメータ名	パラメータの値
単語数	21190
要素数	512
Embedding 層の次元数	8
フィルタサイズ	30
フィルタ数	128
隠れ層 1 の次元数	64
隠れ層 2 の次元数	16
epoch	5

表 3　epoch 数での精度(accuracy)の比較

epoch	ER	ERJ	EiyoER	EiyoJE
1	0.927	0.875	0.880	0.816
2	0.924	0.889	0.924	0.888
3	0.938	0.912	0.946	0.904
4	0.936	0.909	0.934	0.909
5	0.934	0.903	0.936	0.917
6	0.932	0.914	0.938	0.912
7	0.927	0.903	0.941	0.914
8	0.941	0.921	0.945	0.922
9	0.936	0.911	0.943	0.909
10	0.926	0.909	0.946	0.909
11	0.931	0.911	0.946	0.917
12	0.938	0.916	0.950	0.914
13	0.936	0.921	0.946	0.914
14	0.941	0.914	0.945	0.912
15	0.936	0.914	0.945	0.917

3. バグ予測精度の比較実験

ここでは，前節で作成したデータと複数の学習済みモデルを使用して比較実験を行う．対象とするソフトウェアは VB.NET で開発されたものであり，現在も実際に医療の現場で稼働しているソフトウェアである．総行数は約 250 万行であり，機能としてはソフトウェア 1 つで医療施設のデータ管理を行うものである．来客数や投薬データ，空き病室の参照など病院内で利用するデータをまとめ管理する機能を持つ．よって，プログラムの基本的な機能は読み込みや書き込みといったデータベースに格納されているデータの操作を行うものである．学習済みモデルは，実際にドクターが利用するメインシステム（ER）と薬局専用のシステム（ERY）や，院内でのメールのやり取りを扱うシステム（ERMAIL）を含む 14 個のサブシステムそれぞれで学習を行ったモデルを合わせた計 15 個を扱う. これらのシステム単体で学習を行ったモデルを ER 単体モデル，ERY 単体モデルと呼ぶことにする．行う実験は以下の 3 つである．

(1) 単体モデルを利用して学習に利用したシステム以外のシステムで分類した場合どのような精度になるのか.
(2) 分類を行うシステム以外の単体モデルで分類し，その結果で多数決を採った場合どのような精度になるのか.
(3) 分類を行うシステム以外のソースコード片を利用して一括で学習を行い，そのモデルで分類すると精度はどのようになるのか.

本研究では，(2) で作成したモデルを多数決モデルと呼び，(3) で作成したモデルを一括モデルと呼ぶ. 実験で分類するシステムは，メインシステムである ER, ER と規模が近い ERJR を扱う．ここで得られた単体モデル，多数決モデル，一括モデルの精度を以降で比較し考察する.

3.1 単体モデル

他システムの単体モデルで ER のバグ予測を行った．その結果を表 4 に示す．また，ERJR でも同様の実験を行った．その結果を表 5 に示す．表は accuracy の高い順に並べてある．6 列目は，バグを含むソースコード片のデータ数である．2 列目は分類結果のが正確さを示す accuracy である．3 列目の適合率（precision）「バグと判定したファイルの中に，どの程度本物のバグありが含まれていたか」を示す値である．4 列目の再現率（recall）は「全体のバグの中からどの程度見つけることができたか」を示す値である．accuracy，precision，recall はいずれも最大値が 1 であり，1 に近いほど精度が良いことを表す.
予測結果を見てみると, バグのデータ数が 1500 個以上のシステムで学習を行った単体モデルの全てが，今回対象とした 2 つのシステムにおいて 79%以上の精度で分類できている．逆に，バグのデータ数が極端に少ないシステムの単体モデルは上手く分類できていない. 特に ERLblPrt の単体モデルはどちらの分類も 50%という精度であり，precision と recall が 0 であることから全てのデータに対してバグなしと判定していることが分かる．たとえ同じ業務用ソフトウェアのシステムであっても，データ数が極端に少ない単体モデルは他のシステムへの代用とすることが困難であることが分かる．また，precision と recall を比較してみると

accuracy が下がるのに比例して precision も下がっていることが分かる. それに比べると recall は殆ど下がっていない. これは実際にバグを予測してはいるが，バグではないものをバグと判定してしまっていることを示している.

表 4　他システム単体モデルによる ER の予測結果

solution	accuracy	precision	recall	F 値	bugs
ERJB	0.948	0.937	0.966	0.951	3667
ERJR	0.924	0.890	0.956	0.922	3719
ERJ	0.900	0.878	0.929	0.903	2792
ERJY	0.890	0.836	0.949	0.889	2782
ER3CL	0.849	0.767	0.920	0.836	2106
EiyoMap	0.838	0.716	0.947	0.815	2846
EiyoER	0.832	0.689	0.962	0.803	2883
ERKTRef	0.822	0.754	0.883	0.813	1534
EiyoJEMaster	0.809	0.655	0.940	0.772	2271
ERMaster	0.808	0.671	0.927	0.778	1692
EiyoJE	0.791	0.627	0.947	0.754	1922
ERKDAI	0.708	0.493	0.885	0.633	465
ERMAILKB	0.501	0.002	0.983	0.005	103
ERLblPrt	0.500	0.000	0.000	0.000	38

表 5　他システム単体モデルによる ERJR の予測結果

solution	accuracy	precision	recall	F 値	bugs
ERJB	0.939	0.925	0.961	0.943	3667
ERJY	0.925	0.899	0.953	0.925	2782
ERJ	0.920	0.916	0.926	0.921	2792
ER	0.872	0.780	0.965	0.863	3838
ER3CL	0.872	0.804	0.923	0.859	2106
EiyoER	0.849	0.718	0.965	0.823	2883
EiyoMap	0.839	0.709	0.949	0.812	2846
ERKTRef	0.830	0.758	0.874	0.812	1534
EiyoJEMaster	0.829	0.696	0.955	0.805	2271
ERMaster	0.818	0.693	0.925	0.792	1692
EiyoJE	0.813	0.658	0.945	0.776	1922
ERKDAI	0.653	0.379	0.851	0.525	465
ERMAILKB	0.502	0.003	0.944	0.006	103
ERLblPrt	0.500	0.000	0.000	0.000	38

3.2 多数決モデル

前節での分類結果を用いて多数決を採る．多数決モデルの具体的な分類例を表 6 に示す．例では 6 個の単体モデル A, B, C, D, E, F でそれぞれ分類を実行し多数決を採っている．source のコード片 4 個の各入力データに対し，単体モデルでのバグ判定で閾値 3 個以上のものを最終的な分類においてバグと判定したものである．0 はバグなし，1 はバグありを表す.

表 6　多数決モデルの分類例

単体モデル	source[0]	source[1]	source[2]	source[3]
A	0	1	1	0
B	0	1	0	0
C	0	1	1	0
D	1	1	1	0
E	0	1	0	1
F	1	1	0	0
バグ判定	0	1	1	0

分類を行うシステム以外の単体モデル 14 個を利用した多数決モデルによる分類を行った．ここで，バグと判定するためには賛同するモデル数を検討する必要がある．この賛同数の閾値の設定は，対象とするソフトウェアや利用する単体モデルの数など，種々の要因によって変化することが推測されるため, 実験により精度を比較する必要がある. そこで, 閾値を最小の 1 から最大の 14 の範囲で変化させ精度の変化を計測する．表 7 に多数決モデルによる ER のバグ予測結果を示す．また，表 8 に ERJR で多数決モデルによる分類を行った際の精度を示す．モデルの優劣の判断材料として precision と recall のバランスを示す F 値を用いた. F 値は 1 を最大値として値が大きい程，そのモデルの precision と recall のバランスが良いことを表す.

表 7　多数決モデルによる ER の予測結果

多数決(閾値)	accuracy	precision	recall	F 値
1	0.869	0.797	0.991	0.884
2	0.917	0.876	0.971	0.921
3	0.925	0.911	0.942	0.926
4	0.927	0.936	0.917	0.927
5	0.917	0.954	0.876	0.914
6	0.898	0.964	0.826	0.890
7	0.869	0.971	0.761	0.853
8	0.844	0.979	0.702	0.818
9	0.815	0.986	0.639	0.775
10	0.786	0.991	0.577	0.729
11	0.738	0.992	0.480	0.647
12	0.607	0.992	0.217	0.356
13	0.501	1.000	0.002	0.003
14	0.500	0.000	0.000	0.000

実験結果を見ると，いずれのシステムでも多数決モデルの予測精度の最大値は 92%を上回っている．これはバグ予測において高精度に分類できている．また，F 値はいずれのシステムでも最大値が 0.92 以上となっている．これは，ある閾値ではバランスの取れた高精度のバグ予測モデルで

あることを示している．さらに，閾値を増加させていくと accuracy は上がっていくが最大値に達した後にまた下がり始めていることが分かる．precision は殆どの場合で閾値が増加すれば値が増加している．しかし，recall は逆に減少していることが表から分かる．これは，閾値を上げればバグと判定したものの精度は増加するが，網羅率は減少していくことを表している．

表 8　多数決モデルによる ERJR の予測結果

多数決(閾値)	accuracy	precision	recall	F 値
1	0.870	0.799	0.990	0.884
2	0.923	0.880	0.980	0.927
3	0.933	0.915	0.955	0.935
4	0.935	0.940	0.929	0.934
5	0.924	0.954	0.891	0.921
6	0.903	0.966	0.836	0.896
7	0.880	0.974	0.780	0.866
8	0.855	0.980	0.725	0.834
9	0.831	0.987	0.670	0.798
10	0.793	0.991	0.590	0.740
11	0.725	0.994	0.453	0.622
12	0.588	0.991	0.177	0.300
13	0.502	1.000	0.003	0.006
14	0.500	0.000	0.000	0.000

3.3 一括モデル

多数決モデルの実験では，それぞれのシステム単体で学習し多数決を採る方法を示したが，比較対象として全てのデータをまとめて学習を行ったモデルが挙げられる．そこで，バグ予測を行うシステム以外のシステムの全てのソースコード片を学習データとした一括モデルにより実験を行う．しかし，今回のバグを含まないソースコード片は 6000 個程しか無いのに対して，バグのソースコード片は全てのシステムで合計 30000 個程と約 5 倍存在する．そこで，バグを含まないソースコード片の 6000 個を 5 回繰り返し利用しデータ数を揃えた．バグデータは同じソフトウェアのシステム毎に抽出したものであるが，システムの構造が似ているため同じようなソースコード片が多く存在する．そこで，似たようなバグデータを含むバグデータと，同じものを複数回利用したバグを含まないデータで学習を行った．バグ予測の対象は前述の実験と同じシステムを対象とした．一括モデルによる各システムの予測結果を表 9 に示す．1 列目は分類を行ったシステムである．表から分かるように，いずれのシステムでも精度は 93%を超えている．また，precision も 1.0 に近く，高い精度を示している．しかし，ER の recall は約 0.96 と高いが ERJR は 0.87 と，ER と比べると低い．これは，バグと判定したものは高確率でバグであったが，ER と比べるとバグを網羅できていないことを表している．

表 9　一括モデルの分類結果

solution	accuracy	precision	Recall	F 値
ER	0.980	0.999	0.962	0.980
ERJR	0.935	0.999	0.870	0.930

4. 実験結果と精度の比較

比較実験で得た実験結果とモデルの精度の比較を行う．ER を対象とした実験では一括モデルが最も accuracy が高く 0.980 となった．次に，ERJB 単体モデルが 0.948 と高くその次に閾値を 4 とした多数決モデルが 0.927 と高い．ERJR を対象とした実験では，ERJB 単体モデルが最も accuracy が高く 0.939 となった．次に，同率で閾値を 4 に設定した多数決モデルと一括モデルが 0.935 と高くなった．しかし，precision は一括モデルの方が高いのに対して，recall は多数決モデルの方が高いという結果になった．

以上の結果から accuracy においては，上手く学習が完了した単体モデルの方が多数決モデルよりも精度が高くなるという結果であった．しかし，多数決モデルにおいては閾値 4 に設定した際に単体モデルで 2 番目に高い accuracy のモデル以上の精度を出している．さらに，閾値を増加させていくと precision が増加するため全てのモデルの中で最も precision の高い値を出すことができる．

比較の結果，単体モデルで最も上手く学習できているモデルの accuracy の方が高いという結果になった．しかし，単体モデル全体を見ると上手く学習できていないモデルも多く存在する．例えば，バグのデータ数の少ない ERKDAI や ERMAILKB，ERLblPrt の単体モデルは殆ど学習できていない．そこで，学習できていない単体モデルを多数決から除外することで，多数決モデルの精度が向上するのか試みた．対象システムは ER で，除外したのは前述の 3 つのシステムである．結果を表 10 に示す．11 個のシステムによる多数決モデルの accuracy の最大は閾値が 4 の時であり，これは 14 個多数決モデルの時と同じである．さらに，acc の値も 0.927 と等しい．これは，上手く学習できていないモデルを除外しても精度の向上は見られないが，上手く学習できていないモデルによる精度の影響はほぼ無いことを示している．

今回対象としたシステムでは上手く学習できた単体モデルが存在するが，選択システムによっては毎回存在するとは限らない．そのため，ある程度の精度で分類できた単体モデルで多数決モデルを構築した場合の精度の比較を行った．対象ソフトウェアは ER である．Eiyo で始まるシステムは ER などと比べ，比較的にシステムが異なる．単体モデルを見ても ER と ERJR においての EiyoER 単体モデルや EiyoMap 単体モデルなどの accuracy の精度は他と比べる

と 0.8 前後と低い. 更に, バグのファイル数の規模も約 2000 から 3000 程度と極端な差はない. よって, 多数決モデルに利用する単体モデルは EiyoER, EiyoMap, EiyoJE, EiyoJEMaster を用いた. 一括モデルを含めた ER を対象にした結果を表 11 に示す. この時の単体モデルの精度を表 12 に示す. 結果は, ER における accuracy は閾値が 1 の時の多数決モデルと一括モデルが最も高くなった. しかし, F 値で比べると多数決モデルの方が高い. 単体モデルでは EiyoMap 単体モデルが最も accuracy が高く 0.841 となった. ERJR でも同じ実験を試みたが, 同様な結果が得られた. 閾値 1 の多数決モデルが最も高く 0.866 となり, 次に EiyoER 単体モデルが 0.850 と高くなった. 一括モデルは 0.812 となった. 突出して上手く学習できた単体モデルの無い場合は, 今回の検証においては多数決モデルが最も精度が高い結果になった.

表 10　11 個多数決モデルのバグ予測結果

多数決(閾値)	accuracy	precision	recall	F 値
1	0.882	0.813	0.991	0.893
2	0.918	0.885	0.961	0.922
3	0.926	0.920	0.933	0.926
4	0.927	0.943	0.908	0.925
5	0.912	0.959	0.862	0.908
6	0.887	0.968	0.800	0.876
7	0.858	0.976	0.735	0.838
8	0.828	0.985	0.667	0.795
9	0.804	0.993	0.612	0.757
10	0.767	0.995	0.537	0.697
11	0.706	0.996	0.413	0.584

表 11　Eiyo システムのみを用いた ER のバグ予測結果

多数決(閾値)	accuracy	precision	recall	F 値
1	0.856	0.903	0.798	0.847
2	0.850	0.952	0.737	0.831
3	0.818	0.983	0.648	0.781
4	0.757	0.996	0.516	0.680
一括	0.856	0.994	0.718	0.833

表 12　Eiyo 単体モデルのみを用いた ER のバグ予測結果

Solution	accuracy	precision	recall
EiyoMap	0.841	0.716	0.951
EiyoER	0.834	0.689	0.969
EiyoJEMaster	0.813	0.655	0.950
EiyoJE	0.794	0.627	0.943

5. 考察と課題

実験の結果より, 対象としているシステム以外の全てのシステムを用いた場合の accuracy は, 最も上手く学習できた単体モデルの方が高い結果になった. これは, 対象としているソフトウェアと実験方法の特徴が出ていると推察できる. 対象としているソフトウェアは, メインシステムと多くのサブシステムから構成されている. システムの中には似たような構成をしているものが存在する. そのため, そういったシステムで学習した単体モデルは高い精度を出しやすい. よって, 上手く単体で学習が完了できた場合は, 単体モデルを利用した方が高い精度で分類できる可能性が高い.

しかし, 必ずしも全てのソフトウェアに単体で学習が上手く完了するシステムが存在するとは限らない. 実際に, 今回の用意した単体モデルでも多数決モデルの精度を上回ったものは少ない. そこで, ある程度学習できた単体モデルを用いて 2 つのシステムを対象に実験を行った. 結果はどちらのシステムでも多数決モデルが 1 番高い結果になった. また, 上手く学習できていない単体モデルを除去した多数決モデルと, 除去しない多数決モデルでは精度に差が見られなかった. これらの結果は, 多数決モデルは 1 つの単体モデルの精度に大きな影響を受けず, ある程度学習できた単体モデルが複数存在していれば, 上手く学習できた単体モデルと同等以上の精度を安定して出せる可能性があることを示している.

一括モデルは上手く学習できたものは最も精度が高いものもあるが, 上手く学習できていないものは最も精度が低く, 他の 2 つと比べると recall が低くなる傾向がある. 今回の実験においても, 14 個のシステムを用いて ER を分類した際を除いて, 他の 2 つのモデルよりも低い値になっている. これは, 他の 2 つのモデルよりも網羅率の観点で劣っている可能性を示している. また, 学習時間の観点からみると, 一括モデルは学習を再度行う際に 1 からやり直す必要がある. それに比べて, 多数決モデルは複数の単体モデルからなっているため, 学習のやり直しが必要になったモデルだけ学習を実行すればよい. さらに, 多数決モデルはシステム毎に学習を行っているので, 各システムの特徴を捉えた学習モデルを利用できる. また, 閾値を任意の値に設定できるためある程度用途に合わせて利用できる. ある程度バグを網羅し, バグと判定したものの精度は高いものを希望する際に手軽に利用できる. バグ予測はデバッグ作業などの効率化を目標に行うので, 全体の 5 割程度だけでも確実にバグを発見できる予測モデルは有効であると考える.

構築した多数決モデルの有効性や他のモデルとの比較を検討したが, 課題も多く存在する. 今回構築した多数決モデルは多くのサブシステムを利用している. 規模は様々

であるが全て同じ構造の学習モデルで学習を行っている．そのため，システムによって学習モデルの構造を変えた方が，精度が高くなる可能性がある．今回の実験では，全てのシステムの単位モデルの epoch を一律 5 としたが，システムの規模も様々なためシステム毎に設定することでより良い単体モデルになると推察できる．さらに，今回の実験ではバグを含まないソースコード片は ER のもののみを使用している．それぞれのシステムのソースコード片を抽出し実験を行う必要がある．また，本研究の結果は 1 つの業務ソフトウェアに対してのものであるため，別の業務ソフトウェアに対して，同じ実験を試みても同じような結果になるとは断言できない．よって，別の業務ソフトウェアでの実験を行い，ソフトウェアの特徴がどれほど結果に影響を及ぼすのか検証する必要がある．本研究の最終目標は，デバッグ作業の効率化であるため、どの程度の精度でどれくらい効率化できるのかを調査することも重要である．

6. 結論

単体のシステムで学習を行った単体モデルを複数用意して実験を行った．複数の単体モデルの分類結果で多数決を採る多数決モデルは，単体で上手く学習できたモデルが存在しない状況では最も高い精度を発揮した．多数決モデルは閾値を任意に設定できるため幅広い用途で利用できる可能性がある．また，1 つの単体モデルの影響を大きく受けないため，安定して上手く学習が行えた単体モデルと同等以上の精度が発揮できる．複数のシステムの学習データをまとめて学習した一括モデルは，上手く学習を行えれば最も高い精度を発揮するが，そうでない場合では他の 2 つよりも精度が劣っている．また，precision は高い値を出す傾向にあるが recall は他の 2 つのモデルと比較すると低くなる傾向にある．これは，バグと分類できるものは高い精度で分類できるが，他の 2 つのモデルと比べると分類できるバグの数が少なくなる可能性がある．

以上の結果から，ある程度上手く学習できた単体モデルを複数個用意できるならば多数決モデルを採用した方が，安定した精度を出しつつ幅広い用途で利用できる可能性がある．

謝辞 実験データを提供していただいた株式会社 東日本技術研究所の皆様には感謝いたします．

参考文献

[1] 近藤将成, 森啓太, 水野修, 崔銀惠 : 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, 59(4), 1250-1261 (2018).
[2] 中庭貴洋, 上田賀一 : ソースコード片を用いた深層学習による業務ソフトウェアの不具合予測. 情報処理学会研究報告, Vol.2020-SE-204, No.3,1-8 (2020).
[3] 宮本敦哉, 阿萬裕久, 川原稔:バグ混入予測の精度向上に向けた個人化予測モデルの組合せ手法とその評価. コンピュータソフトウェア, 37(4), 38-49 (2020)
[4] Microsoft. Roslyn アナライザーを使用したコード分析 - VisualStudio(Windows) https://docs.mi-crosoft.com/ja-jp/visualstudio/code-quality/roslyn-analyzers-overview?view=vs-2022, (参照 2022-02-01)
[5] Martín Abadi, et al. : Tensorflow: A system for large-scale machine learning. In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265–283 (2016)
[6] Antonio Gulli and Sujit Pal. : Deep learning with Keras. Packt Publishing Ltd. (2017)