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.

機械学習を用いたコードクローンの危険予測手法

今里　文香† 　堀田　圭佑† 　肥後　芳樹† 　楠本　真二†

† 大阪大学大学院情報科学研究科

あらまし　一般的に，コードクローンは，ソフトウェアの保守性を低下させる原因になるとされている．一方で，すべてのコードクローンがソフトウェアの保守性に悪影響を与えるとは限らない．そこで本研究では，機械学習を用いてソースコード中に存在するコードクローンの危険予測を自動的に行う手法を提案する．提案手法では，過去に存在したコードクローンの特徴を，そのコードクローンが危険かどうかという情報とともに学習する．そして，そのデータをもとに，現在ソースコード中に存在するコードクローンの危険予測を行う．また，本研究では，提案手法よるコードクローンの危険予測の精度を評価するために，2 つのオープンソースプロジェクトを対象とした評価実験を行った．その結果，1 つのプロジェクトについては，高い精度で危険なコードクローンを予測できていることを確認した．

キーワード　コードクローン，ソフトウェア保守，機械学習

Risk Prediction for Code Clones Based on Machine Learning

Ayaka IMAZATO†, Keisuke HOTTA†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University

Abstract　Code clones often cause the deterioration of software maintainability. Meanwhile not all code clones have a bad influence on software. Because there are massive code clones in source code, it is not practical to cope with all of them. Hence it is necessary to detect only risky code clones which reduce software maintainability. In this study, we propose a method to predict risks of code clones in source code automatically using machine learning techniques. In our method, we learn features and risks of code clones which existed in the past. Then, we predict risks of code clones in target source code based on these training data. In this study, in order to evaluate accuracy of prediction for risky code clones by using our proposed method, we conducted a case study with 2 open source projects. As a result, we confirmed that the proposed method can predict risky code clones with high accuracy for 1 project.

Key words　Code Clone, Software Maintenance, Machine Learning

1. はじめに

一般的に，コードクローンはソフトウェアの保守性に悪影響を及ぼすと言われている．例えば，ソフトウェアの開発過程では，あるコード片が修正された場合に，そのコード片と類似した他のコード片に対しても同様の修正が加えられることがしばしば起こる．既存研究では，36〜38 %のコードクローンが，開発過程で一貫した修正を要することが報告されている [1]．類似した複数のコード片に一貫した修正を加える際，本来修正を加えるべきであったコード片に対して修正漏れが生じることも少なくない．修正漏れが生じると，その箇所には新たなバグが発生する可能性がある．そのため，コードクローンはソフトウェアの保守性を低下させる原因になりうる [2], [3]．一方で，すべてのコードクローンが危険とは限らない．生成されてから 1 度も修正が加えられないコードクローンなどは，ソフトウェアの保守にとって無害といえる．実際に，48 %のコードクローンは，開発過程で 1 度も修正が加えられないことを示す既存研究が存在する [4]．また，例えば，ある時点でコードクローンとなったコード片が，その後すぐに修正が加えられることによってコードクローンではなくなる場合もある．こういった存在期間が短いコードクローンは，開発過程で一貫した修正を繰り返し要するコードクローンに比べて，ソフトウェアに悪影響を及ぼす危険性が低い．さらに，ソースコード中に含まれるコードクローンの数は膨大であり，開発者がすべてのコードクローンを管理することは現実的ではない．

著者らは，効率的にコードクローンを管理するために，ソースコード中に含まれるコードクローンのうち，ソフトウェアに悪影響を及ぼすような危険なコードクローンのみを開発者に提示する必要があるのではないかと考えた．そこで本研究では，機械学習を用いて，ソースコード中に含まれるコードクローン

の中から，ソフトウェアに危険を及ぼすもののみを特定する手法を提案する．提案手法では，開発履歴情報を分析することで，過去に存在したコードクローンの特徴や，それらがバグを発生させたか否かなどの情報を取得する．そして，それらの情報をもとに，機械学習の技術を用いて，現在のソースコード中に含まれるコードクローンの中から危険なコードクローンを特定する．また，提案手法よるコードクローンの危険予測の精度を評価するために，2 つのオープンソースプロジェクトを対象とした評価実験を行った．その結果，1 つのプロジェクトについては，高い精度で危険なコードクローンを予測できていることを確認した．

2. 先 行 研 究

先行研究には，機械学習を用いてコードクローンの性質を予測する研究がいくつか存在する．この章では，それらの一部を紹介する．

楊らは，有用なコードクローンは開発者ごとに異なるとし，それぞれの開発者にとって有用なコードクローンを，機械学習によって自動的に特定する手法を提案している [5]．楊らの提案手法では，まず，コードクローン検出ツールによって検出されたコードクローンの一部を，利用者が有用か有用でないかのどちらかに分類する．そして，機械学習を用いてそれらの情報を学習することによって，残りのコードクローンの中からその利用者にとって有用なコードクローンを自動的に特定する．楊らは，8 人の被験者に対して被験者実験を行い，提案手法を評価した．その結果，楊らは，開発者ごとに有用なコードクローンが異なることを明らかにするとともに，提案手法によって，およそ 70 %以上の精度で，開発者にとって有用なコードクローンを予測できることを確認した．

Wang らは，コード片の複製が行われる際に，その操作によって生成されるコードクローンが危険かどうかを判定する手法を提案している [6]．Wang らは，コード片の複製によって生成されたコードクローンについて，その後一貫した修正が少なくとも 1 度以上加えられるものを危険，1 度も一貫した修正が加えられないものを危険でないと定義している．また，コードクローンの危険予測を行うシナリオとして，Wang らは 2 つのシナリオを用いている．1 つは，生成されるコードクローンが危険なものではない場合にのみ複製を許容する“安全にコード片の複製を行うシナリオ”である．もう 1 つは，生成されるコードクローンが危険なものである場合に複製を禁止する“危険な複製を回避するシナリオ”である．Wang らは，2 つのソフトウェアを対象として，それぞれのシナリオについて提案手法の評価を行った．その結果，安全に複製を行うシナリオについては，提案手法によって危険ではないと判定された複製のうち，95 %以上が実際に危険ではないということが確認された．また，危険な複製を回避するシナリオについては，提案手法を用いることにより，複製によって生成された危険なコードクローンのうち，34 %を検出することに成功した．

Mondal らは，互いに類似するコード片の集合（クローンセット）について，その構成要素のうち一貫した修正を要する要素のグループを予測する手法を提案している [7]．類似したコード片は，しばしば一貫した修正を必要とする．しかし，クローンセット中のすべての要素が一貫した修正を要するとは限らない．Mondel らは，クローンセットの進化履歴から，似たような進化を遂げる類似コードのグループを取得し，その情報をもとに予測を行った．その結果，Mondal らは，クローンセット中のある要素について，同一クローンセットの中から，その要素と連動して進化を遂げる別の要素を予測することに成功した．

3. 提 案 手 法

本研究では，機械学習を利用し，コードクローンの危険予測を行う手法を提案する．機械学習とは，既存のデータを学習することにより，未知のデータの性質を予測・特定する技術のことである．機械学習では，学習用のデータのことを**訓練データ**，データの学習によって得られる，未知のデータの性質を予測するためのモデルを**学習モデル**と呼ぶ．本研究では，互いにコードクローンとなっているコード片の集合であるクローンセットを訓練データとして，クローンセットの危険予測を行うための学習モデルを構築する．また，本研究では，将来的にバグの原因となるクローンセットを予測することを目的とする．そのため，将来的にバグの原因となるクローンセットを危険，そうではないクローンセットを危険ではないと定義する．学習モデルを構築するための手法について，この章で説明する．

本研究における提案手法の入力はソフトウェアの開発履歴情報であり，出力はクローンセットの危険予測を行う学習モデルである．提案手法の流れを図 1 に示す．提案手法は 4 つの段階に分かれている．

（1） まず，ソフトウェアの開発履歴情報から，開発の中で生成されたすべてのクローンセットを特定する．そして，特定した各クローンセットについて，そのクローンセットが生成されてからの進化過程を追跡したデータを取得する．クローンセットの進化データは，一般的に**クローンセットの系譜**（以下，**系譜**）と呼ばれる．

（2） 次に，取得した各クローンセットの系譜について，危険判定を行う．本研究では，バグ修正が行われた系譜を危険，それ以外の系譜を危険でないとする．

（3） 系譜の危険判定が完了したら，各系譜から 1 つずつ，訓練データとするクローンセットを取り出す．その後，取り出した各クローンセットについて，危険判定を行う．危険判定は，そのクローンセットが属する系譜が危険かどうかで行う．すなわち，危険な系譜に属するクローンセットは危険とし，危険でない系譜に属するクローンセットは危険でないとする．また同時に，訓練データとするクローンセットについて，そのクローンセットの状態を調査する．クローンセットの状態とは，例えば，クローンセットを構成する要素の数や，各要素同士の類似度などである．以下，本論文では，このようなクローンセットの状態を示す数値や指数のことを，**特徴量**と呼ぶ．

（4） 最後に，訓練データとして取り出した各クローンセットについて，その特徴量と，危険か危険でないかという情報を合わせて学習し，学習モデルを構築する．この学習モデルに，

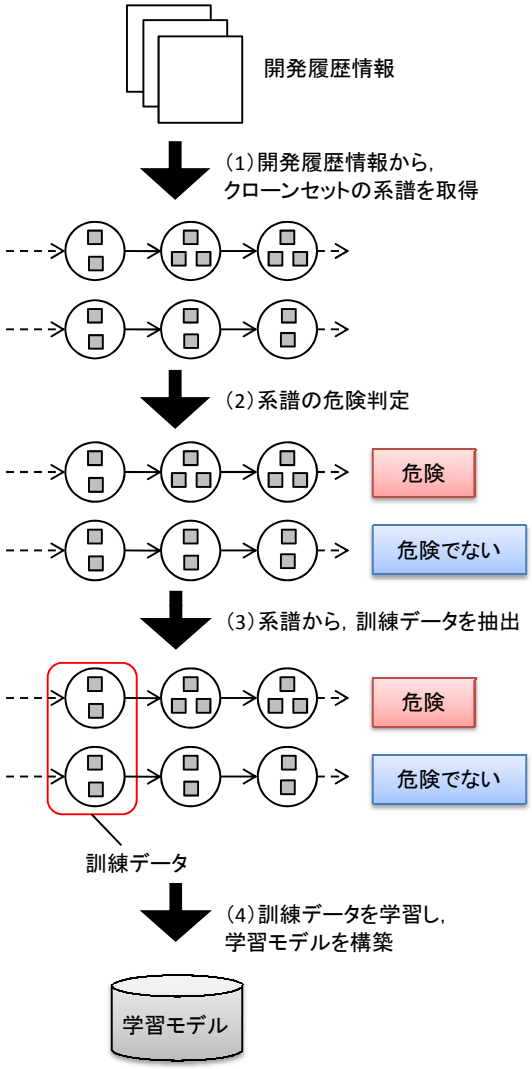

図 1　手法の流れ

危険予測を行いたいクローンセットの特徴量を与えることで，そのクローンセットが危険かどうか予測する．

以上が，本研究における提案手法の概要である．以下では，提案手法のそれぞれの段階について，詳細を説明する．

3.1　クローンセットの系譜の取得

クローンセットの系譜の取得方法について説明する．本研究では，既存のコードクローン追跡手法である CRD（Clone Region Descriptors）[8] に基づいて，クローンセットの系譜を取得する．CRD とは，コード片の位置情報を用いて，コードクローンの追跡を行う手法である．CRD はコード片の位置情報に基づきコードクローンの追跡を行うため，コードクローンに対して大きな修正が加えられた場合でも，追跡が可能である．しかし一方で，CRD には，あるメソッドの一部分を別のメソッドとして抽出するなどの，コード片の位置が変わるような修正が加えられた場合に，コードクローンの追跡ができなくなるといった問題点がある．そこで本研究では，堀田らが提案する，従来の CRD の問題点を解決した手法 [9] を用いて，クローンセットを追跡し，系譜を取得した．取得した系譜の例を図 2 に示す．図中の，矢印で繋がれ横方向に連なっているものが，1

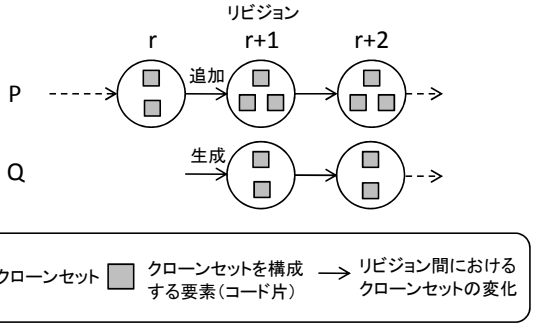

図 2　クローンセットの系譜

つの系譜である．また，丸いオブジェクトはある時点におけるクローンセットを表し，クローンセット中の四角いオブジェクトはそのクローンセットを構成する要素を表している．矢印は，リビジョン間におけるそのクローンセットの変化を表している．例えば，系譜 P は，リビジョン r においてクローンセットの要素数が 2 であり，その後修正により新たに要素が追加され，リビジョン $r+1$ で要素数が 3 になっている．このように，取得した系譜からは，そのクローンセットの要素数の変化や，修正によるコード内容の変化といった，開発過程における変遷を知ることができる．

3.2　各系譜の危険判定

各系譜について，危険か危険でないかを判定する方法を説明する．提案手法では，開発過程においてバグ修正が行われた系譜を危険とみなす．まず，系譜を構成するクローンセットに対して修正が行われたリビジョンをすべて抽出する．その後，抽出したそれぞれのリビジョンにおけるコミットメッセージを参照し，バグ修正に関する記述が含まれていれば，そのリビジョンにおいてクローンセットにもバグ修正が行われたものとし，その系譜を危険と判定する．なお，バグ修正に関する記述の判定については，既存研究 [10] で紹介されているパターンを利用した．

3.3　訓練データの抽出

本研究では，系譜の開始リビジョンにおけるクローンセットを訓練データとして取り出す．次に，取り出した訓練データについて，その特徴量を取得する．本研究では，全部で 30 種類の特徴量を用いた．その一部を以下に記述する．

- クローンセットの要素数
- クローンセットを構成する各要素のトークン数の平均
- クローンセットを構成する要素を含むファイルの数

また，特徴量はすべて，訓練データとして取り出した時点におけるクローンセットの情報から取得できるものを採用している．すなわち，いずれの特徴量も，クローンセットの過去の履歴情報を必要としない．

例として，図 3 の系譜から訓練データとするクローンセットを取り出す場合を考える．まず，系譜の開始リビジョン x におけるクローンセットを，訓練データとして取り出す．次に，特徴量を取得する．訓練データとして取り出したクローンセットは，2 つの要素から構成されるため，要素数は 2 となる．また，要素 α，β のトークン数はともに 20 であるため，各要素

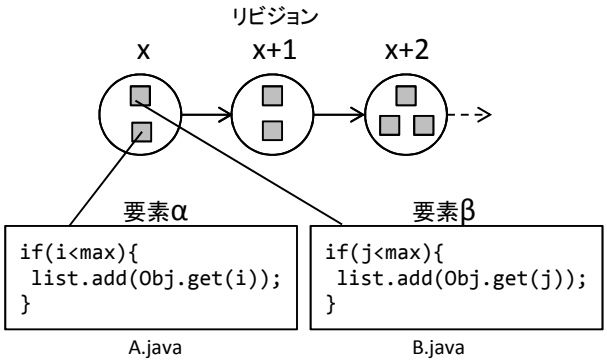

図 3　クローンセットの系譜

のトークン数の平均は 20 となる．さらに，要素 α は A.java，β は B.java 中のコード片であるため，このクローンセットの要素は 2 つのファイルに含まれる．このようにして，系譜から取り出した訓練データについて，30 種類すべての特徴量を得る．

3.4　学習モデルの構築

この時点で，各訓練データについて，特徴量の取得と危険判定が完了している．これらの情報を，機械学習によって学習し，学習モデルを構築する．この学習モデルに，危険判定を行いたいクローンセットの特徴量を与えることで，そのクローンセットが将来的にバグの原因となる危険なクローンセットかどうかを予測することが可能になる．なお，学習モデルの構築には，データマイニングツール weka [11] を使用した．

4. 評 価 実 験

本研究では，2 つのオープンソースプロジェクトを対象とし，提案手法の評価を行った．対象としたプロジェクトを表 1 に示す．本研究で実装したツールは，Java 言語で開発されたプロジェクトのみを解析対象としている．そのため，実験対象プロジェクトはいずれも Java 言語で開発されているものを選択した．また，本研究では 3.1〜3.3 節の操作によって各プロジェクトからクローンセットを取得し，それらをデータセットとして実験を行った．各プロジェクトについて，実験で用いたデータセットの内訳をまとめたものを表 2 に示す．

以下では，評価手法および実験結果について述べる．

4.1　評 価 手 法

本実験では，構築した学習モデルの評価手法として交差検証法を用いた．交差検証法とは，データセットをいくつかのブロックに分割し，評価を行う手法である．この際，各ブロックのデータ数が同程度になるように割り当てを行う．ここで，データセットを N 個に分割したと仮定する．交差検証法では，分割したブロックのうち $N-1$ 個のブロックを訓練データとし，残りの 1 個のブロックをテストデータとして評価を行う．具体的には，まず訓練データを用いて学習モデルを構築する．その後構築した学習モデルを用いてテストデータの予測を行い，精度を計測する．この操作を，各ブロックをテストデータとした場合についてそれぞれ行い，それらの精度の平均を取る．本実験では，$N = 10$ として学習モデルの評価を行った．

また，本実験では評価尺度として *Precision* と *Recall* の 2 つの指標を用いた．これらの説明を表 3 にまとめる．

また，本実験では，学習モデル構築アルゴリズムとして J48，RandomForest，ベイジアンネットワーク，SVM，ロジスティック回帰の 5 つのアルゴリズムを用い，それぞれについて交差検証法を用いて評価を行った．各アルゴリズムの説明を表 4 にまとめる．

4.2　実 験 結 果

評価実験の結果を，表 5 にまとめる．表より，jEdit については，*Precision* は 83〜95 %，*Recall* は 83〜90 %と，高い精度で危険なクローンセットの予測を行うことができている．アルゴリズム別に見ると，決定木ベースである J48 および RandomForest は，*Precision*，*Recall* ともにほぼ 9 割と高い．SVM は，*Precision* が 95 %と 5 つのアルゴリズムの中で最も高いが，*Recall* は 5 つのアルゴリズムの中で最も低い．

表 2　実験データの内訳

プロジェクト名	危険	危険でない
jEdit	1,695	2,563
Ant	2,835	22,818

表 3　評 価 尺 度

評価尺度	説明
Precision	構築した学習モデルによって危険と予測されたクローンセットのうち，実際に危険であったものの割合
Recall	危険なクローンセットのうち，実際に構築した学習モデルによって危険と予測されたクローンセットの割合

表 4　学習モデル構築アルゴリズム

アルゴリズム	説明
J48	決定木による学習アルゴリズム．C4.5 [12] に基づいた決定木を生成する．
RandomForest	決定木による学習アルゴリズム．訓練データを複数の組に分け，それぞれの組ごとに決定木を生成する．
ベイジアンネットワーク	因果グラフによる学習アルゴリズム．条件付確率を用いて因果グラフを生成する．
SVM	最も優秀な学習アルゴリズムの 1 つ．2 クラスのパターン識別器を生成．
ロジスティック回帰	数学モデルによる学習アルゴリズム．確率的な分類モデルを生成．

表 1　実験対象プロジェクト

プロジェクト名	リビジョン番号		対象リビジョン数	行数		計測時間 (分)
	開始	最終		開始	最終	
jEdit	3,791	21,981	5,292	57,837	183,006	3,691
Ant	267,548	1,233,420	12,621	7,864	255,061	26,730

ベイジアンネットワークについては，*Precision* は 83 %と低いが，*Recall* は 5 つのアルゴリズムの中で最も高い．この結果から，以下のことがわかる．

- いずれのアルゴリズムについても，高精度で危険なクローンセットを予測できている．
- ベイジアンネットワークは，他のアルゴリズムと比べて多くの危険なクローンセットを検出することができている．

一方で，Ant については，*Precision* は 0〜67 %，*Recall* は 0〜54 %と，いずれの評価尺度も低く，さらにアルゴリズムによって大きくばらつきがある．アルゴリズム別に見ると，決定木ベースである J48 および RandomForest は，*Precision* が 60〜67 %，*Recall* が 41〜45 %となっている．他のアルゴリズムと比べると *Precision* は高いが，これは危険と予測したクローンセットのうち実際に危険であったものが 60〜67 %であったということであり，33〜 40 %については予測がはずれている．そのため，予測精度が良いとは言えない．SVM および ロジスティック回帰は，*Precision*，*Recall* ともに非常に低い．ベイジアンネットワーク については，*Recall* が 54 %と 5 つのアルゴリズムの中で最も高いが，これは危険なクローンセットのうち 54 %しか検出できていないということであり，46 %のクローンセットの検出には失敗している．そのため，こちらも予測精度が良いとは言えない．この結果から，以下のことがわかる．

- いずれのアルゴリズムも予測精度が低い．
- アルゴリズムごとに，評価尺度に大きなばらつきがある．

評価実験より，jEdit に対しては，*Precision* は 83〜95 %，*Recall* は 83〜90 %と，高い精度で危険なクローンセットを予測することができた．一方で，Ant に対しては，危険と予測したクローンセットのうち少なくとも 3 割程度が実際には危険ではなかった．さらに，危険なクローンセットのうち多くとも 5 割程度しか検出することができなかった．また，どちらの評価結果からも，ベイジアンネットワーク によって構築した学習モデルは他のアルゴリズムと比較してより多くの危険なクローンセットの検出に成功していることがわかった．

5. 考　　察

5.1 危険なクローンセットの傾向

jEdit を対象とする実験では高い精度でクローンセットの危険予測を行うことに成功した．しかし，Ant を対象とする実験では *Precision*，*Recall* のいずれの評価尺度も低い結果となった．これについて著者らは，jEdit では危険なクローンセットに特有のパターンが存在したが，Ant のデータセットからはそういったパターンを得ることができなかったために，2 つのプロジェクト間で予測精度に大きな差が出たのではないかと考えた．そこで，それぞれのプロジェクトについて，データセットの特徴量から危険なクローンセットの傾向を得ることができるかどうか調査を行った．その結果，いずれのプロジェクトについても，危険なクローンセットほど，要素を含むファイルのディレクトリ上における広がりが大きい傾向があることがわかった．すなわち，クローンセットを構成する要素を含むソースファイルが，ディレクトリ構造の上で離れた位置にあるものほど危険となりやすかった．これは，クローンセットの要素が広範囲に散らばっている場合，ある要素に修正を加えた際に，他の要素への修正を見落としやすくなるためであると考えられる．また，jEdit については，危険なクローンセットほど，そのクローンセットの要素を構成するトークンの種類が多い傾向があることがわかった．しかし，Ant にはそういった傾向は見られなかった．

表 5　評 価 結 果

アルゴリズム	jEdit		Ant	
	Precision	Recall	Precision	Recall
J48	0.93	0.89	0.67	0.45
RandomForest	0.92	0.89	0.60	0.41
ベイジアンネットワーク	0.83	0.90	0.45	0.54
SVM	0.95	0.83	0	0
ロジスティック回帰	0.92	0.86	0.28	0.03

以上より，危険なクローンセットの傾向は，複数のプロジェクト間で共通している部分もあるが，必ずしも一致しているとは限らないことがわかる．そのため，予測対象とするプロジェクトごとに，適切な特徴量を選択することが必要であろう．また，危険なクローンセットを予測するための特徴量としては，“ 要素を含むファイルのディレクトリ上における広がり ” が有効である可能性がある．

5.2 既存研究との比較

多くの既存研究では，特徴量として，ソフトウェアの開発履歴から得られる情報を用いている．しかし，本研究で使用する特徴量は，利用者が予測を行いたい時点におけるクローンセットの情報からすべて得ることができるため，開発履歴情報を用いる必要がない．そのため，開発履歴情報を用いるのは学習モデルを構築する時のみであり，1 度学習モデルを構築した後は，開発履歴情報を用いることなく危険予測が可能である．したがって，本研究では，既存研究とは異なり，モデル構築後は開発履歴情報そのものを用意する必要がなくなり，また，予測を行うたびに開発履歴情報を遡るといった操作を行わなくてよい．以上の理由より，本研究で提案した手法では，既存研究と比較して容易かつ高速に予測を行うことができるといえる．

6. 結果の妥当性

計 測 対 象

本実験で対象としたソフトウェアは 2 つであり，Java で開発されたオープンソースソフトウェアに限定して調査を行った．このため，より多くのソフトウェアを対象として実験を行った場合や，異なる言語で記述されたソフトウェアや商用ソフトウェアを対象とした場合は，本研究で得られた結果と異なる結果が得られる可能性がある．

バグ修正の判定

本研究では，コミットコメントの内容のみに基づき，系譜に対するバグ修正の有無の判定を行っている．そのため，実際にその系譜に対してバグ修正があったかどうかソースコードを見

て分析を行うに至っていない．したがって，本実験では，危険判定に誤りのあるクローンセットがデータセットに混入している可能性があり，評価実験結果の妥当性を脅かしている恐れがある．

特 徴 量

機械学習では，使用するすべての特徴量が予測に良い影響を与えるとは限らない．すなわち，予測精度を下げるノイズとなるものや，結果に何も影響を与えない不必要なものも存在する．本研究では，クローンセットの特徴を表す指標として 30 種類の特徴量を用いた．しかし，それらが適切なものかどうかの検証は行っていない．したがって，本研究では結果に悪影響を与える特徴量を使用している可能性があるため，予測精度の低下を招いている恐れがある．

7. あ と が き

コードクローンは，修正漏れなどを引き起こす可能性があることから，ソフトウェアの保守性に悪影響を与える危険なものとされている．しかし一方で，すべてのコードクローンが危険とは限らない．さらに，ソースコード中には非常に多くのコードクローンが含まれるため，すべてのコードクローンを管理することは現実的ではない．そこで本研究では，機械学習を用いて，ソースコードに含まれるコードクローンの中からソフトウェアに危険を及ぼすもののみを検出する手法を提案した．提案手法では，過去にソースコード中に存在したコードクローンについて，特徴量とそのコードクローンが危険かどうかという情報を学習し，学習モデルを構築した．そして，構築した学習モデルを用いることによって，危険なコードクローンの予測を行った．また，提案手法によって構築した学習モデルの予測精度を評価するために，2 つのプロジェクトを対象とした評価実験を行った．評価実験の結果，一方のプロジェクトについては，*Precision* が 83〜95 %，*Recall* が 83〜90 %と，高い精度で危険なコードクローンを予測できていることを確認した．しかし，もう一方のプロジェクトについては，*Precision* が 0〜67 %，*Recall* が 0〜54 %と，いずれの評価尺度も低い結果となった．そのため，今後は，どのようなプロジェクトについても高精度で危険なコードクローンの予測を行うことを可能にするために，手法を改善していく必要がある．

今後の課題は以下のとおりである．

- より多くのソフトウェアに対して実験を行う．
- 系譜に対するバグ修正の有無の判定が正しいかどうか，実際にソースコードを見て確認する．
- 特徴量の選別を行う．また，必要に応じて新たに特徴量を追加する．

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 (S)(課題番号：25220003)，挑戦的萌芽研究 (課題番号：24650011)，若手研究 (A)(課題番号：24680002)，並びに特別研究員奨励費 (課題番号：25・1382) の助成を得た．

文 献

[1] K. Miryung, S. Vibha, N. David, and M. Gail, “An Empirical Study of Code Clone Genealogies,” SIGSOFT Software Engineering Notes, vol.30, no.5, pp.187–196, Sept. 2005.

[2] L. Zhenmin, L. Shan, M. Suvda, and Z. Yuanyuan, “CP-Miner: finding copy-paste and related bugs in large-scale software code,” IEEE Transactions on Software Engineering, vol.32, no.3, pp.176–192, March 2006.

[3] 肥後芳樹，楠本真二，“コード修正履歴情報を用いた修正漏れの自動検出,” 情報処理学会論文誌，vol.54，no.5，pp.1686–1696，May 2013.

[4] G. Nils and K. Rainer, “Frequency and Risks of Changes to Clones,” Proceedings of the 33rd International Conference on Software Engineering, pp.311–320, May 2011.

[5] 楊嘉晨，堀田圭佑，肥後芳樹，井垣宏，楠本真二，“機械学習を用いた類似度に基づく有用なコードクローンの自動特定手法,” 情報処理学会論文誌，vol.54，no.2，pp.807–819，Feb. 2013.

[6] W. Xiaoyin, D. Yingnong, Z. Lu, Z. Dongmei, L. Erica, and M. Hong, “Can I Clone This Piece of Code Here?,” Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering, pp.170–179, Sept. 2012.

[7] M. Manishankar, R.C. K., and S.K. A., “Prediction and Ranking of Co-change Candidates for Clones,” Proceedings of the 11th Working Conference on Mining Software Repositories, pp.32–41, May 2014.

[8] D.-E. Ekwa and R.M. P., “Clone Region Descriptors: Representing and Tracking Duplication in Source Code,” ACM Transactions on Software Engineering and Methodology, vol.20, no.1, pp.3:1–3:31, July 2010.

[9] 堀田圭佑，肥後芳樹，楠本真二，“CRD の類似度に基づくコードクローン追跡手法,” 電子情報通信学会技術研究報告，第 113 巻，pp.127–132，July 2013.

[10] J. Śliwerski, T. Zimmermann, and A. Zeller, “When do changes induce fixes?,” Proceedings of the 2005 International Workshop on Mining Software Repositories, pp.1–5, July 2005.

[11] “Weka 3: Data Mining Software in Java”. `http://www.cs.waikato.ac.nz/ml/weka/`.

[12] Q.J. Ross, C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers Inc., Jan. 1993.