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.

機械学習を用いた自動生成コードの特定

下仲　健斗† 　鷲見　創一† 　肥後　芳樹† 　楠本　真二†

† 大阪大学大学院情報科学研究科，吹田市
E-mail: †{s-kento,s-sumi,higo,kusumoto}@ist.osaka-u.ac.jp

あらまし　近年，ソースコード解析に関する研究が盛んに行われている．解析対象のソフトウェアに含まれるソースコードの中にはしばしば自動生成コードが含まれており，多くの場合自動生成コードは解析の対象にはならず除外される．除外する方法としては，自動生成コード内に存在する特有のコメント文を文字列検索により特定する方法がある．しかしこの方法では，自動生成コード特有のコメント文が消された場合に，文字列検索などにより機械的に自動生成コードを特定することができない．また，ソースコードが自動生成コードであるかどうか，目視で判定するのは時間的コストが大きい．そこで本研究では，機械学習を用いて任意の自動生成コードを自動的に特定する手法を提案する．提案手法では，ソースコードの構文情報を学習することで自動生成コードであるかどうかを特定する．また，提案手法を評価するために，4 種類の自動生成プログラムから生成された自動生成コードを対象に実験を行った．その結果，高い精度で自動生成コードを特定できることを確認した．

キーワード　自動生成コード，機械学習，ソースコード解析

Identifying Generated Code by Using Machine Learning Techniques

Kento SHIMONAKA†, Soichi SUMI†, Yoshiki HIGO†, and Shinji KUSUMOTO†

† Graduate School of Information Science and Technology, Osaka University, Suita-shi, 565–0871, Japan
E-mail: †{s-kento,s-sumi,higo,kusumoto}@ist.osaka-u.ac.jp

1. はじめに

近年，ソースコード解析に関する研究が盛んに行われている．例えば，ソースコード中に存在する互いに一致または類似したコード片であるコードクローンに関する研究や，ソフトウェア開発履歴データなどを対象に分析を行うリポジトリマイニングに関する研究などが行われている．解析対象のソフトウェアに含まれるソースコードの中にはしばしば自動生成コードが含まれており，ソースコード解析を行う際に自動生成コードが弊害となることがある．例えば，コードクローン検出において，自動生成コードから多くのコードクローンが検出されることにより他のコードクローンが目立たなくなってしまう [1]．また，リポジトリマイニングにおいて，ソースコードを追跡する際に自動生成部分の追跡によって解析時間が増加してしまうケースがある [2]．したがって，ソースコード解析において自動生成コードは除外すべき存在である．

通常，自動生成コードにはそれ自身が自動生成コードであると明示するためのコメント文が残されている．ゆえに，そのようなソースコードに対しては grep コマンドなどを用いれば特定および除去することができる．しかしソースコードを修正していく過程でそのコメント文が消されている場合がある．コメント文が消された自動生成コードを目視などで特定するのは時間的コストが大きい．したがって，そのようなコメント文が消されたものも含めて自動生成コードを自動的に特定することが必要となる．

コメント文が消された自動生成コードを自動的に特定するためには，自動生成コードにおける何らかの特徴を発見する必要がある．しかし任意の自動生成コードに共通する特徴を目視などで発見するのは困難である．そこで本研究では，機械学習を用いて，コメント文の有無にかかわらず自動生成コードか否かを自動的に判定する手法を提案する．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．提案手法では，まず自動生成コードだと判明しているソースコードを収集する．収集したソースコードから，それらの構文情報を取得する．構文情報を取得することで，コメント文の有無にかかわらず自動生成コードの特徴を学習することが出来る． それらを学習データとして，あるソースコードが自動生成コードか否かを判定する学習モデルを構築する．

構築した学習モデルの評価を行うために，4 種類の自動生成

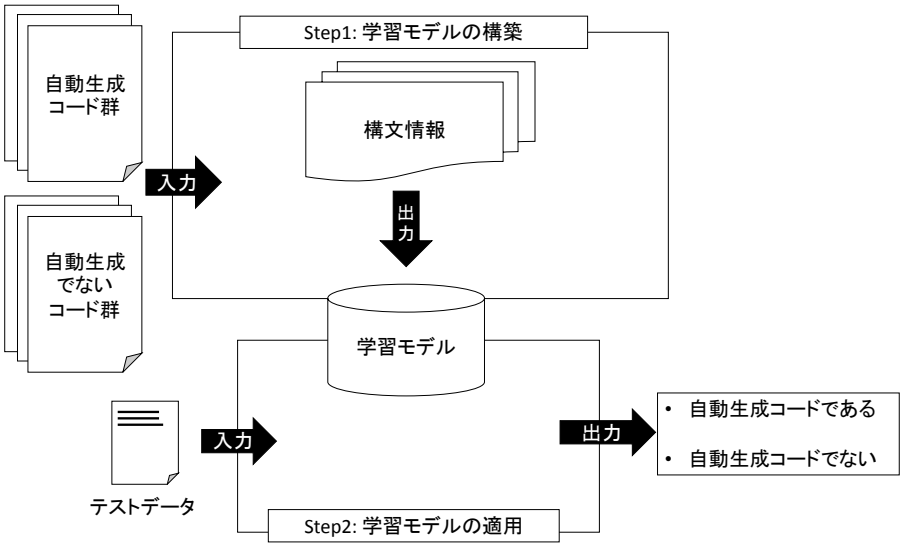

図 1　提案手法の概要

コードを対象とする評価実験を行った．実験の結果，高い精度で自動生成コードを特定できることを確認した．また，大規模なソースコードの集合に対して提案手法を適用した結果，コメント文が消された多くの自動生成コードを特定することができた．

以降，2 章では提案手法について説明し，3 章で評価実験について述べる．4 章で実験に対する考察，5 章で妥当性への脅威について述べ，最後に 6 章で本研究のまとめと今後の課題について述べる．

2. 提 案 手 法

本研究では，機械学習を利用し，自動生成コードを自動的に特定する手法を提案する．提案手法の概要を図 1 に示す．本研究における提案手法の入力は，学習データとテストデータである．出力として，テストデータが自動生成コードか自動生成でないコードかを判定する．

機械学習では，学習データの特徴を学習し，未知のデータであるテストデータの性質を予測する学習モデルを構築する．構築した学習モデルに，テストデータであるソースコードの構文情報を与えることで，そのソースコードが自動生成コードであるかどうかを判定することが可能になる．学習データの特徴のことを説明変数と呼び，予測したいテストデータの性質のことを目的変数と呼ぶ．

提案手法は次の 2 ステップから構成される．Step1 では，学習データから構文情報を取得し，学習モデルを構築する．Step2 では，自動生成コードかどうかを判定したいテストデータに対して学習モデルを適用する．以降，各ステップについて詳細に説明する．

Step1: 学習モデルの構築

Step1 では，学習データから構文情報を取得し，学習モデルを構築する．構文情報として，ソースコードの AST(Abstract Syntax Tree: 抽象構文木) ノードを用いる．説明変数がとる値は，各 AST ノードの出現回数であり，整数値である．また，目的変数は自動生成コードであるかそうでないか，である．構文情報を取得した後，学習モデルを構築する．学習モデルを構築する際，説明変数が多いと過学習という状態に陥り，精度が低下することが知られている．そこで，全ての説明変数を用いるのではなく，有用な説明変数のみを選択するため，変数選択という処理を行う．変数選択を行うアルゴリズムとしてフィルター法とラッパー法が知られており，多くの場合ラッパー法を用いた方が予測精度が向上することが示されているため，本研究ではラッパー法を用いる [4]．学習モデルを構築するアルゴリズムとして，決定木 [5]，Random Forest [6]，Naive Bayes [7]，SVM [8] を用いる．これらは機械学習を行う上で代表的なものである．各アルゴリズムの特徴を表 1 に示す．

Step2: 学習モデルの適用

Step2 では，自動生成コードであるか判明していないソースコードに対して学習モデルを適用する．そのために，自動生成コードか否かを判定したいソースコードをテストデータとして

表 1　学習モデル構築アルゴリズム

アルゴリズム	説明
決定木	説明変数の値に従って，条件分岐の木を生成するアルゴリズム．欠損値や外れ値が多い学習データに対して有用．
Random Forest	学習データをランダムにサンプリングし，複数の決定木を生成する．それらを結合し，より精度の高い決定木を生成するアルゴリズム．決定木に比べて時間的コストが大きい．
Naive Bayes	条件付き確率を用いたアルゴリズム．機械学習アルゴリズムの中で最も基本的であり，計算量も少ない．
SVM	2 クラスのパターン識別器を生成するアルゴリズム．機械学習アルゴリズムの中で最も識別性能が優れていることで知られている [9]．

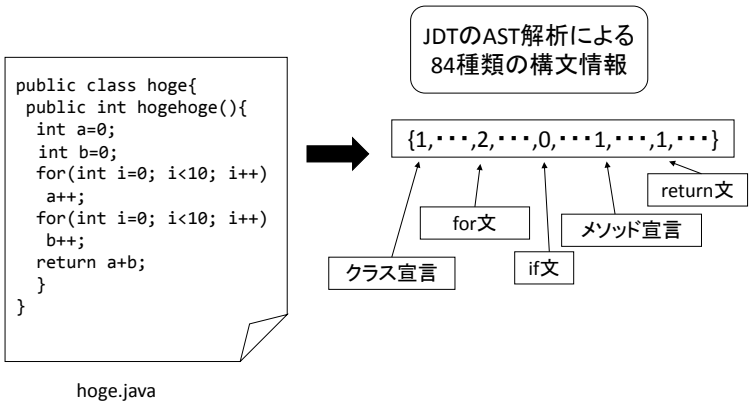

図 2　AST 解析の例

用意する．用意したテストデータの構文情報を Step1 と同様に取得し，変数選択を行う．選択する変数は，Step1 で，学習データにおいて選択されたものと同じものを選択する．以上の処理を行った後，テストデータに対して学習モデルを適用することで，自動生成コードか否かを判定する．

3. 評 価 実 験

本章では，提案手法を評価するために行った 4 つの実験と，その実験結果について述べる．

3.1 実 験 概 要

提案手法の評価を行うために，4 つの実験を行った．その概要を以下に示す．

実験 1　提案手法における Step1 に従って構築した学習モデルの評価を行った．

実験 2　収集した自動生成コードおよび自動生成でないコードのうち，ファイルサイズが 10KB 以上のものに限定して，実験 1 と同様に学習モデルの評価を行った．

実験 3　ある自動生成プログラムから構築した学習モデルを別種類の自動生成コードへ適用した．

実験 4　テストデータに対して構築した学習モデルを適用し，自動生成コードが特定できるかどうかの実験を行った．

3.2 実 験 対 象

実験には，4 つの自動生成プログラムから生成された自動生成コードを用いた．4 つの自動生成プログラム名と，各自動生成プログラムによって生成された自動生成コードのファイル数を表 2 に示す．これらの自動生成プログラムは Java で開発されたパーサジェネレータである．

収集したソースコードは全て Java で記述されている．また，テストデータには，大規模データセットである *UCI Source Code Data Sets* [13](以降，*UCI datasets* と呼ぶ) を用いた．

構文情報の取得には，Eclipse JDT 3.10 [3] を用いて AST 解析を行った．AST 解析の例を図 2 に示す．Eclipse JDT 3.10 では 84 種類の AST ノードが定義されており，本研究ではそれらを説明変数とする．つまり本研究では 84 個の説明変数が存在することになる．

表 2　自動生成コード

自動生成プログラム	ANTLR	JavaCC	JFlex	SableCC
ファイル数	8,778	21,219	3,789	16,066

また，変数選択，学習モデルの構築およびテストデータの予測には，Java で開発された機械学習ライブラリである Weka [10] を使用した．

3.3 データの収集

自動生成コードか否かを判定する学習モデルを構築するため，自動生成コードと自動生成でないコードを収集した．その収集方法について述べる．

まず自動生成コードの収集方法について説明する．上述したとおり，自動生成コードにはそれ自身が自動生成コードであると明示するためのコメント文が残されているため，そのコメント文の一部を検索キーワードとして収集した．自動生成コードのコメント文の例を図 3 に示す．具体的には，GitHub からコメント文検索により収集した．収集の際は，JSoup [11] を用いたウェブスクレイピングを行い，自動で収集した．なお，本研究では自動生成コード特有のコメント文が存在すれば，自動生成コードに人の手が加わったものも自動生成コードとみなす．

次に，自動生成でないコードの収集方法について説明する．本研究では，大規模なソースコードの集合である Apache リポジトリ [12] から自動生成でないコードをランダムに収集した．収集したファイル数は，自動生成コードと同数である．また，Apache リポジトリの中に自動生成コードが含まれている可能性があるため，それらをコメント文の検索により特定し，除外した．

3.4 評 価 尺 度

実験の評価尺度として，*Precision* と *Recall* を用いる．以下，それぞれの尺度の定義について説明する．

Precision　学習モデルによって自動生成コードと判定されたソースコードのうち，実際に自動生成コードであるものの割合

Recall　実際に自動生成コードであるもののうち，学習モデルによって自動生成コードと判定されたものの割合

実験 1 および実験 2 の評価尺度は *Precision* と *Recall* の両方を算出した．実験 3 では，自動生成コードにのみ学習モデルを適用したため，*Recall* のみ算出した．実験 4 では，テス

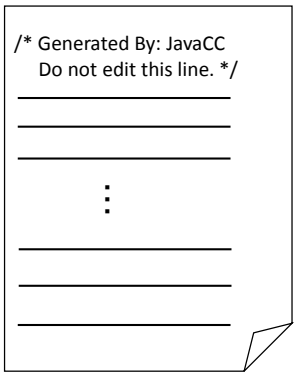

図 3　自動生成コードのコメント文の例

トデータとして用いる *UCI datasets* のうち，実際に自動生成コードであるファイルの数が未知であるため，*Precision* のみ算出した．

3.5 実　験　1

収集した各自動生成コードを用いて 4 種類の学習モデルを構築した．さらに，収集した全ての自動生成コードを用いた学習モデルも構築した．構築した計 5 種類の学習モデルの性能を評価するために交差検証を行った．交差検証では，まずデータセットを N 個のブロックにランダムに分割する．分割したブロックのうち，$N-1$ 個のブロックを学習データとし，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．この処理を，ブロックを変化させながら N 回行い，それらの精度の平均をとる．本実験では $N=10$ として学習モデルの評価を行った．

交差検証の結果を表 3 に示す．ただし，表中の P と R はそれぞれ *Precision* と *Recall* を表す．また，All Files は，収集した全ての自動生成コードを用いて構築した学習モデルを表す．多くの場合で *Precision*，*Recall* ともに 90%を超える精度になっていることが分かる．特に，ANTLR と JFlex ではほとんどの場合で 99%と，非常に高い精度である．*Precision* と *Recall* を比較すると，全体的に *Precision* の方が高いことが分かる．自動生成プログラムごとに比較すると，ANTLR と JFlex に比べて，JavaCC と SableCC の精度が低くなっていることが分かる．アルゴリズムごとに比較すると，Random Forest の精度が最も高く，Naive Bayes の精度が最も低いことが分かる．

表 4　ファイルサイズが 10KB 以上の自動生成コード

自動生成プログラム	ANTLR	JavaCC	JFlex	SableCC
ファイル数	8,686	6,661	3,786	860

3.6 実　験　2

学習データのファイルサイズの違いによって学習モデルの性能に変化があるかを確認するため，学習データのファイルサイズを 10KB 以上に限定して実験を行った．学習モデルの評価は実験 1 と同様に，交差検証を行った．なお，ファイルサイズが 10KB の場合，ソースコードの行数は 400 行程度である．ファイルサイズを 10KB 以上に限定した場合の，各自動生成コードのファイル数を表 4 に示す．ANTLR と JFlex に関しては，ほとんどのファイルが 10KB 以上であるのに対して，JavaCC と SabelCC は 10KB 以上のファイル数が比較的低い割合になっている．

ファイルサイズを 10KB 以上に限定して交差検証を行った結果を表 5 に示す．なお，表中の下線は，表 3 に示す結果と比べて，数値が向上しているものを表す．実験 1 では比較的精度が低かった JavaCC と SableCC が，ANTLR と JFlex と同様に，ほとんどの場合で 90%を超える精度に向上している．アルゴリズムを比較すると，依然として Naive Bayes の精度が最も低い．しかし数値だけを見ると，ほとんどの場合で 90%を超えており，実験 1 で見られるようなアルゴリズム間の格差は軽減している．

3.7 実　験　3

実験 1 および実験 2 では同一種類の自動生成コードに対して交差検証を行っているが，実験 3 では，ある自動生成コードを用いて構築した学習モデルで，別種類の自動生成コードを特定できるかどうかを確認した．そのため，学習モデルを適用したのは自動生成コードのみであり，自動生成でないコードには適用していない．また，学習モデルの構築に用いた学習データは，実験 2 と同様にファイルサイズが 10KB 以上のソースコードに限定している．

ある自動生成プログラムから構築した学習モデルを別種類の

表 3　交差検証の精度

アルゴリズム	ANTLR		JavaCC		JFlex		SableCC		All Files	
	P	R	P	R	P	R	P	R	P	R
決定木	99%	99%	91%	91%	99%	99%	97%	97%	96%	96%
Naive Bayes	98%	98%	79%	67%	99%	99%	86%	81%	81%	57%
Random Forest	99%	99%	91%	91%	99%	99%	97%	97%	97%	97%
SVM	98%	98%	83%	82%	99%	99%	92%	91%	78%	76%

表 5　ファイルサイズが 10KB 以上の場合における交差検証の精度

アルゴリズム	ANTLR		JavaCC		JFlex		SableCC		All Files	
	P	R	P	R	P	R	P	R	P	R
決定木	98%	98%	99%	99%	99%	99%	99%	99%	98%	98%
Naive Bayes	97%	97%	96%	96%	99%	99%	94%	93%	92%	86%
Random Forest	98%	98%	99%	99%	99%	99%	99%	99%	98%	98%
SVM	97%	97%	97%	97%	99%	99%	97%	97%	93%	93%

自動生成コードへ適用した結果を表 6 に示す．表中の数値は *Recall* を表している．表 6 から，ほとんどの場合において，実験 1 と実験 2 における交差検証の結果よりも精度が大幅に低下していることが分かる．ANTLR，JavaCC，JFlex の 3 種類の自動生成コード間における精度は 60〜90%であるのに対し，SableCC における精度は 0〜40%と，極端に低いことが分かる．このことから，SableCC によって生成された自動生成コードの特徴と，他の 3 つの自動生成プログラムから生成された自動生成コードの特徴はあまり似通っていないと考えられる．

3.8 実　験　4

実験 1 から実験 3 では，構築した学習モデルの性能を評価していた．実験 4 では，自動生成コードかどうか判明していないソースコードに対して学習モデルを適用した．この実験では，学習モデルによって自動生成コードと判定されたソースコードが，実際に自動生成コードかどうかは目視によって判断する必要がある．したがって大規模なデータセットである *UCI datasets* 全てに学習モデルを適用すると時間的コストが大きいため，*UCI datasets* の一部に，JavaCC で構築した学習モデルのみを適用した．なお，用いた学習データのファイルサイズは実験 2 と実験 3 同様，10KB 以上である．

UCI datasets に対して学習モデルを適用した結果を表 7 に示す．学習モデルを構築するアルゴリズムとして，実験 1 および実験 2 で最も精度が高かった Random Forest を用いている．表中の *Precision* から，約 70%の精度で自動生成コードが特定可能であることが分かる．また，自動生成でないコードと判定されたものに関しては，ファイル数が膨大であるため約 1%をランダムに抽出し，目視確認を行った．その結果，約 98%の精度で自動生成でないコードが特定されていることを確認した．

表 7　*UCI datasets* に学習モデルを適用した結果

ファイル数	146,346
特定された自動生成コードのファイル数	438
目視確認により自動生成コードと判断されたファイル数	304
Precision	69%

4. 考　　察

本章では，3 章で述べた評価実験についての考察を行う．

まず，実験 1 と実験 2 の結果について考察する．全てのファイルサイズの学習データを用いて構築した学習モデルと，学習データのファイルサイズを 10KB 以上に限定して構築した学習モデルでは，後者の方が精度が高いことから，学習データのファイルサイズが大きい程，学習モデルの性能は高くなると考えられる．また，交差検証の結果において，誤検出されたファイルの中身を調べた．その結果，誤って自動生成コードと判定されたものの特徴として，

- ファイルサイズが小さい．具体的には，説明変数の値が 10 以下のものが多い．
- case 文やリテラルが多い．

などが挙げられる．ファイルサイズが小さいものには，インターフェースや抽象クラスが多く見られた．このことから，構文情報が少ないものは誤検出されやすいと考えられる．しかし，ファイルサイズが小さいものは解析時間の増加などの原因とはなりにくい．したがって，それらの誤検出がコードクローン検出やリポジトリマイニングに与える影響は小さいと考えられる．また，case 文やリテラルが多いものには，構文解析処理を行っているファイルが多く見られた．それらはコメント文が消された自動生成コードの可能性がある．

次に，実験 3 の結果について考察する．ある自動生成コードを用いて構築した学習モデルを別種類の自動生成コードに適用した場合，高い精度で特定できたものもあるが，全く特定することができなかったものもある．したがって，別種類の自動生成コード間における学習モデルの適用は，必ずしも有効ではないことが分かる．

最後に，実験 4 の結果について考察する．実際には自動生成でないコードが，自動生成コードであると判定されたファイル

表 6　別種類の自動生成コード間において学習モデルを適用した場合の精度

自動生成プログラム	アルゴリズム	ANTLR	JavaCC	JFlex	SableCC
ANTLR	決定木	-	62%	71%	38%
	Naive Bayes	-	63%	84%	32%
	Random Forest	-	64%	85%	45%
	SVM	-	52%	62%	30%
JavaCC	決定木	75%	-	86%	22%
	Naive Bayes	85%	-	99%	31%
	Random Forest	75%	-	99%	24%
	SVM	71%	-	99%	54%
JFlex	決定木	71%	75%	-	18%
	Naive Bayes	66%	92%	-	18%
	Random Forest	66%	51%	-	0%
	SVM	71%	95%	-	22%
SableCC	決定木	1%	3%	0%	-
	Naive Bayes	22%	3%	0%	-
	Random Forest	0%	0%	0%	-
	SVM	6%	0%	1%	-

の中身を確認した．そのような誤検出と思われるファイルの中身は，case 文やリテラルが多く存在するが，それら以外の要素も多く存在するものであった．すなわち，case 文やリテラルの出現回数は多いが，プログラム要素全体に対する割合が低いことが誤検出の要因と考えられる．自動生成でないコードが自動生成コードであると検出されないために，構文情報以外の特徴量を追加するなど，提案手法の改善が必要となる．

5. 妥当性への脅威

本章では，評価実験に含まれる妥当性への脅威について述べる．

本研究では，自動生成でないコードを Apache リポジトリから収集した．その際，Apache リポジトリに含まれる自動生成コードをコメント文検索により除外しているが，コメント文が消された自動生成コードも含まれている可能性がある．そのため，本来自動生成コードであるものを自動生成でないコードとみなしている可能性がある．

また，本実験で対象とした自動生成コードは，Java で記述された 4 種類の自動生成コードである．そのため，他の種類の自動生成コードを用いた場合や，他の言語で記述された自動生成コードを用いた場合では，本実験とは異なる結果が得られる可能性がある．

実験 4 において，学習モデルによって自動生成コードと判定されたものが，実際に自動生成コードであるかどうかは著者ら 2 人の目視確認によるものである．

6. あ と が き

本研究では，機械学習を用いて自動生成コードを自動的に特定する手法を提案した．提案手法では，自動生成コード特有のコメント文の有無にかかわらず，自動生成コードか否かを判定するために，ソースコードの構文情報を抽出し，それらを学習データとして学習モデルを構築した．

実験として，4 種類の自動生成コードを収集し，それらを対象に評価実験を行った．実験の結果，ほとんどの場合で *Precision*，*Recall* ともに 90%以上と，高い精度で自動生成コードを特定できていることを確認した．しかし，別種類の自動生成コード間において学習モデルを適用した場合は，精度が大幅に減少した．そのため，今後は任意の自動生成コードを高精度で特定できるようにするために，手法を改善していく必要がある．

また，自動生成コードかどうか判明していないソースコードに対して学習モデルを適用する実験を行った．その結果，約 70%の精度で自動生成コードを特定できることを確認した．しかしこの実験では，一部の自動生成コードで構築した学習モデルしか適用していないため，他の自動生成コードで構築した学習モデルを適用した結果も得る必要がある．

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)，および文部科学省科学研究費補助金若手研究 (A)(課題番号：24680002) の助成を得て行われた．

文　献

[1] Takahumi OHTA, Hiroshi IGAKI, Keisuke HOTTA, Yoshiki HIGO, and Shinji KUSUMOTO, ”An Empirical Study on Copy and Paste of Code in Software Development”, 電子情報学会技術研究報告 2014.
[2] Jan Harder and Nils Gode, ”Cloned code: stable code”, Journal of Software: Evolution and Process 2012.
[3] Eclipse Java development tools (JDT). http://www.eclipse.org/jdt/.
[4] Hall, M.A. and Holmes, G.: Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, IEEE Trans. Knowl. Data Eng., Vol. 15, No6, pp. 1437-1447(2003).
[5] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. ”Classification and regression trees.”, CRS press 1984.
[6] Leo Breiman, ”Random Forests.”, Machine Learning 2001.
[7] Domingos, Pedro and Pazzani, ”On the optimality of the simple Bayesian classifier under zero-one loss”, Machine Learning 1997.
[8] V. Vapnik and A. Lerner, ”Pattern recognition using generalized portrait method.”, Automation and Remote Control, 24, 1963.
[9] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2:121167, 1998.
[10] ”Weka 3: Data Mining Software in Java”, http://www.cs.waikato.ac.nz/ml/weka/.
[11] ”jsoup: Java HTML Parser”, http://jsoup.org/.
[12] ”Apache, Source code repository”, http://svn.apache.org/repos/asf/.
[13] C. Lopes, S. Bajracharya, J. Ossher, and P. Baldi, ”Uci source code data sets”. http://www.ics.uci.edu/~lopse/datasets/.