情報処理学会論文誌 Vol.58 No.4 861–870 (Apr. 2017)

機械学習を利用した構文情報に基づく自動生成ファイルの特定

下仲 健斗[1,a)] 鷲見 創一[1,b)] 肥後 芳樹[1,c)] 楠本 真二[1,d)]

受付日 2016年8月5日, 採録日 2017年1月10日

概要：近年，ソースコード解析に関する研究がさかんに行われている．解析対象のソースファイルの中にはしばしば自動生成ファイルが含まれており，多くの場合自動生成ファイルは解析の対象にはならず除外される．自動生成ファイルを除外する方法として，自動生成ファイル内に存在する特有のコメント文を文字列検索することにより特定するという方法がある．しかしこの方法では，自動生成ファイル特有のコメント文が消された場合に，自動的に自動生成ファイルを特定することができない．また，ソースファイルが自動生成ファイルであるかどうか，1つずつ目視で特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ファイルを自動的に特定する手法を提案する．提案手法では，ソースファイルの構文情報を学習することで自動生成ファイルであるかどうかを判定する．また，提案手法を評価するために，4つの自動生成プログラムから生成された自動生成ファイル群を対象に実験を行った．その結果，90%以上の高い精度で自動生成ファイルを特定できることを確認した．

キーワード：自動生成ファイル，機械学習，ソースコード解析，ソフトウェア保守

Identifying Auto-Generated Files by Using Machine Learning Techniques Based on Syntactic Information

Kento Shimonaka[1,a)] Soichi Sumi[1,b)] Yoshiki Higo[1,c)] Shinji Kusumoto[1,d)]

Received: August 5, 2016, Accepted: January 10, 2017

Abstract: These days, source code analysis is keenly studied because it came into use in practice and research such as mining source code repositories. We often see auto-generated files in target repositories, and remove them prior to source code analysis because they can be noise for source code analysis. We can remove auto-generated files by searching particular comments which exist in auto-generated files. However, we cannot identify auto-generated files automatically with such a way if comments have been deleted. Moreover, manually identifying auto-generated files makes us spend too much time. Therefore, in this study we propose a method to identify auto-generated files automatically by using machine learning techniques. In our method, we learn syntactic information of source code. Then, we can identify whether source files are auto-generated files or not. In this study, in order to evaluate the proposed method, we conducted experiments with source files generated by four kinds of code generators. As a result, we confirmed that the proposed method was able to identify auto-generated files with high accuracy.

Keywords: auto-generated file, machine learning techniques, source code analysis, software maintenance

1. はじめに

近年，ソースコード解析に関する研究がさかんに行われている．たとえば，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

[1] 大阪大学大学院情報科学研究科
Graduate School of Information Science and Technology, Suita, Osaka 565–0871, Japan
[a)] s-kento@ist.osaka-u.ac.jp
[b)] s-sumi@ist.osaka-u.ac.jp
[c)] higo@ist.osaka-u.ac.jp
[d)] kusumoto@ist.osaka-u.ac.jp

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．解析対象のソースファイル群の中にはしばしば自動生成ファイルが含まれており，ソースコード解析を行う際に自動生成ファイルが弊害となることがある [1], [2]．たとえば，コードクローン検出において，自動生成ファイル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の発見が難しくなってしまう [3], [4]．また，リポジトリマイニングにおいて，ソースファイルを分析する際に自動生成ファイルの存在によって解析時間が増加してしまうケースがある [5], [6]．したがって，ソースコード解析において自動生成ファイルは除外すべき存在である．

通常，自動生成ファイルにはそれ自身が自動生成ファイル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，そのようなソースファイルに対しては grep コマンドなどを用いれば特定および除外することができる．しかしソースファイルを変更および修正していく過程でそのコメント文が消されてしまい，上記の方法が使用できない場合がある．コメント文が消された自動生成ファイルを 1 つずつ目視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ファイル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ファイルを自動的に特定するためには，自動生成ファイルにおける何らかの特徴を発見する必要がある．たとえば，自動生成プログラムが生成するファイル名にはいくつかの規則が定められている．したがって，その規則に基づき，正規表現などを用いて検索を行えば自動生成ファイルを特定できると考えられる．しかしコメント文の場合と同様に，ファイル名が変更されれば特定できなくなる．また，コメント文が残っていた場合でも grep コマンドの引数として適切な文字列を与えなければ特定することはできない．そこで本研究では，機械学習を用いて，与えられたソースファイルが自動生成ファイル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，既知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，まず自動生成ファイルだと判明しているソースファイルを収集する．収集したソースファイル群から，それらの構文情報を取得する．本研究では，ソースファイルにおける各プログラム要素の出現回数を構文情報とする．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ファイルの特徴を学習することができる．取得した構文情報から，与えられたソースファイルが自動生成ファイルか否かを判定する学習モデルを構築する．

提案手法の評価を行うため，4 つの自動生成プログラムによって生成された自動生成ファイル群を対象とする評価実験を行った．実験の結果，90%以上の高い精度で自動生成ファイルを特定できることを確認した．

以降，2 章では準備として自動生成ファイルについて詳細に述べる．3 章では提案手法について説明し，4 章では評価実験について述べる．5 章では評価実験の結果について考察を行い，6 章では妥当性への脅威について述べる．最後に 7 章で本研究のまとめと今後の課題について述べる．

2. 準備

本章では，自動生成ファイルの定義，収集および特定方法について述べる．自動生成ファイルとは，プログラムによって自動的に生成されたソースファイルのことである．ソースファイルを自動で生成するプログラムは多数存在し，たとえば GUI プログラムを生成する GUI ビルダー，あるプログラミング言語を別の言語に変換するトランスレータ，構文解析器を生成するパーサジェネレータなどがある．本研究では比較的収集が容易であるパーサジェネレータによって生成されたファイル（**パーサジェネレータ生成ファイル**と呼ぶ）を対象とする．

また，研究の準備段階において，著者らがパーサジェネレータ生成ファイルに対して目視による調査を行ったところ，ソースコードにおいて以下のような特徴があることが分かった．

- switch-case 文が多い
- 16 進数の数値リテラルなど，人が書かないようなプログラム要素が多く存在する

自動生成ファイルのソースコードの一部を**図 1** の下部に示す．このように，人が書いたソースコードとプログラムによって自動的に生成されたソースコードには視覚的な差異が存在するため，目視によって自動生成ファイルを特定することは可能である．しかし，目視によって自動生成ファイルを特定すると時間的コストが大きくなってしまう．

通常，自動生成ファイルにはそれ自身が自動生成ファイルであると明示するためのコメント文が記述されている．

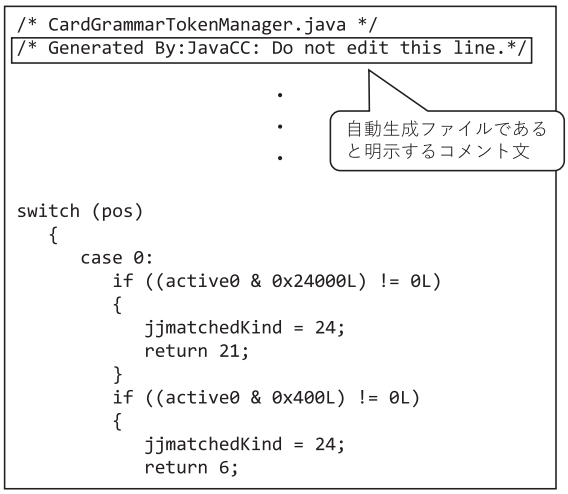


図 1 自動生成ファイルのコメント文の例
Fig. 1 Example of code comments in auto-generated files.

表 1 自動生成プログラムの種類とファイル数
Table 1 Auto-generated source files.

自動生成プログラム	ANTLR	JavaCC	JFlex	SableCC
ファイル数	9,218	15,033	3,737	16,603

自動生成ファイルのコメント文の例を図 1 の上部に示す．このようなコメント文を文字列検索することにより，自動生成ファイルを自動的に特定することができる．しかし，機能追加やバグ修正などの過程においてこのコメント文が消されてしまう場合がある．そのため，コメント文検索だけでは特定できない自動生成ファイルが存在する．また，コメント文が残っていた場合でも grep コマンドの引数として適切な文字列を与えなければ特定することはできない．

2.1 ソースファイルの収集

自動生成ファイルを特定するためのデータセットとして，自動生成ファイル群と自動生成でないファイル群を用いる．本研究では，それらをオープンソースソフトウェアから収集した．その収集方法について述べる．

まず自動生成ファイルの収集方法について説明する．上述したとおり，自動生成ファイルにはそれ自身が自動生成ファイルであると明示するためのコメント文が記述されているため，そのコメント文の一部を検索キーワードとして用いて自動生成ファイルを収集した．本研究では，GitHub [7] からコメント文検索により収集した．しかし GitHub 上の検索では 1,001 件以上の検索結果を取得することができない．この問題に対して高澤らは，検索結果数が 1,000 件を超えないようなファイルサイズを指定し，ファイルサイズごとに分割して収集することで対応している [8]．本研究においても同様の方法を用いた．また，jsoup [9] を用いたウェブスクレイピングを行い，自動で収集を行った．なお，本研究では自動生成ファイルに人の手が加わったものも自動生成ファイルとみなす．収集した自動生成ファイルは，4 つの自動生成プログラムから生成された自動生成ファイルである．4 つの自動生成プログラム名と，収集した自動生成ファイルのうち各自動生成プログラムによって生成された自動生成ファイルの数を**表 1** に示す．

これらの自動生成プログラムはパーサジェネレータであり，Java で記述されたソースファイルを生成する．以降，ANTLR によって生成されたファイル，JavaCC によって生成されたファイル，JFlex によって生成されたファイル，SableCC によって生成されたファイルをそれぞれ**ANTLR 生成ファイル**，**JavaCC 生成ファイル**，**JFlex 生成ファイル**，**SableCC 生成ファイル**と呼ぶ．

次に，自動生成でないファイルの収集方法について説明する．本研究では，大規模なソースファイルの集合である Apache リポジトリ [10] からソースファイルをランダムに収集した．また，Apache リポジトリの中に自動生成ファイルが含まれている可能性があるため，それらをコメント文検索により特定し，除外した．

表 2 データセット作成時におけるノイズデータ含有率
Table 2 Ratio of noise data in making data set.

ファイル群	ノイズデータ含有率
ANTLR 生成ファイル群	0%
JavaCC 生成ファイル群	0.6%
JFlex 生成ファイル群	1%
SableCC 生成ファイル群	0.4%
自動生成でないファイル群	3%

上記の方法で収集した自動生成ファイル群および自動生成でないファイル群の中には，誤って自動生成ファイルとみなしているもの，もしくは誤って自動生成でないファイルとみなしているもの（これらを**ノイズデータ**と呼ぶ）が存在している可能性がある．そこで，ノイズデータを除去するために目視確認を行った．しかし，すべてのソースファイルを目視確認するのは時間的コストが膨大であるため，4 種類の自動生成ファイル群，および自動生成でないファイル群に対し，以下の処理を行った．

1. 各 1,000 ファイルずつランダムに抽出する
2. 目視確認によりノイズデータを除去する
3. 除去したソースファイルの数だけ，再度ランダムに抽出する
4. **3.** で抽出したソースファイルに対して目視確認によりノイズデータを除去する

上記の **3.** および **4.** を繰り返し行い，ANTLR 生成ファイル，JavaCC 生成ファイル，JFlex 生成ファイル，SableCC 生成ファイル，自動生成でないファイルがそれぞれ 1,000 ファイルずつ存在するデータセットを作成した．処理前における，各自動生成ファイル群および自動生成でないファイル群に含まれていたノイズデータの割合を**表 2** に示す．コメント文検索により誤って自動生成ファイルとみなされていた要因としては，キーワードとして用いたコメント文が文字列リテラルとしてソースコード中に存在していたことがあげられる．

2.2 ファイル名検索による自動生成ファイルの特定

コメント文検索以外の自動生成ファイル特定手法として，ファイル名による検索がある．通常，自動生成ファイルのファイル名には，自動生成プログラムごとに定められている生成規則がある．たとえば SableCC では，以下のようなものが定められている [11]．

- AXxx.java
- TXxx.java

ただし，X は大文字の任意のアルファベット，x は小文字の任意のアルファベットを表す．2.1 節で作成したデータセットを用いて，ファイル名による自動生成ファイルの特

定を行った．4つの自動生成プログラムごとのファイル名生成規則を**表 3** に示す．表中の *ClassName* は Java ファイルのクラス名を表す．正規表現を用いて，これらの生成規則にマッチするものを自動生成ファイルとして特定する．また，これらの生成規則にマッチしないものを自動生成でないファイルとみなす．**表 4** にファイル名検索による自動生成ファイル特定の結果を示す．ANTLR 生成ファイル群，JavaCC 生成ファイル群，SableCC 生成ファイル群においては 1,000 ファイル中約 800 ファイルの自動生成ファイルをそれぞれ特定できているのに対し，JFlex 生成ファイル群は 79 ファイルと極端に少なくなっていることが分かる．これは，JFlex のファイル名生成規則の数が少ないことが要因と考えられる．このことから，ファイル名生成規則だけでは自動生成ファイルを特定するのに十分ではないことが分かる．

2.3 自動生成ファイルの特定手法

自動生成ファイルを自動的に特定する手法として，著者らが知る限りでは，コメント文検索とファイル名検索があげられる．コメント文検索では，コメント文が削除されている場合うまく機能せず，ファイル名検索では，十分な量のファイルを特定できないといった問題が存在する．一方で，人が書いたソースコードとプログラムによって自動的に生成されたソースコードには視覚的な差異が存在するため，目視では容易に特定することが可能である．ソースコードの視覚的な差異は，そこに存在している構文の違いによるものであるため，構文情報を利用することにより，高い精度で自動生成されたソースコードを特定できると著者らは考えた．

3. 提案手法

本研究では，機械学習を利用し，自動生成ファイルの構文情報を学習することで自動生成ファイル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 2** に示す．本研究における提案手法の入力は，学習データ（自動生成ファイル群と自動生成でないファイル群）とテストデータ（未知のソースファイル）である．学習データを用いて構築した学習モデルに，テストデータの構文情報を与えることで，テストデータが自動生成ファイルか自動生成でないファイルかを判定し，結果を出力する．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．Step 1 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

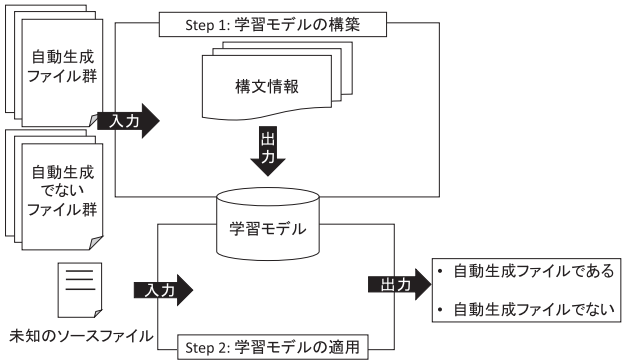


図 2 提案手法の概要
Fig. 2 Overview of the proposed technique.

表 3 自動生成プログラムのファイル名生成規則
Table 3 Rules of generating file names on each parser generator.

自動生成プログラム	生成規則
ANTLR	[*ClassName*]Lexer.java，[*ClassName*]Parser.java，[*ClassName*]Listener.java，[*ClassName*]BaseListener.java
JavaCC	JJT[*ClassName*]State.java，[*ClassName*]Constants.java，Node.java，ParseException.java，SimpleCharStream.java，SimpleNode.java，Token.java，TokenMgrError.java，ASTPerl.java，ASTPython.java，[*ClassName*]TreeConstants.java，[*ClassName*]Visitor.java
JFlex	Yylex.java，[*ClassName*].java
SableCC	Lexer.java, LexerException.java, Parser.java, ParserException.java, DepthFirstAdapter.java, Analysis.java, Switch.java, Switchable.java, TXxx.java, Token.java，AXxx.java，Start.java，Node.java，State.java

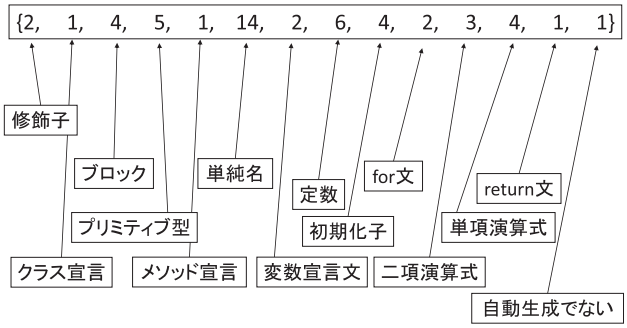
表 4 ファイル名検索による自動生成ファイル特定の結果
Table 4 Results of identifying auto-generated files by searching file names.

自動生成プログラム	特定した自動生成ファイル数	特定した自動生成でないファイル数
ANTLR	894	986
JavaCC	767	989
JFlex	79	1,000
SableCC	867	965

```
public class Hoge{
 public int hogehoge(){
  int a=0;
  int b=0;
  for(int i=0; i<10; i++){
   a++;
  }
  for(int i=0; i<10; i++){
   b++;
  }
  return a+b;
 }
}
```

Hoge.java

(a) ソースコード



(b) 構文情報

図 3 構文情報の取得
Fig. 3 Obtaining syntax information.

する．Step 2 では，自動生成ファイル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．

3.1 Step 1：学習モデルの構築

Step 1 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．各説明変数と目的変数をベクトルとして表したものを構文情報とする．構文情報の取得には，Murakami らの既存研究と同様の方法を用いる [19]．具体的には，Eclipse JDT 3.10 [20] を用いて AST 解析を行う．Eclipse JDT 3.10 では 84 種類の AST ノードが定義されているが，そのうち 3 つはコメント文に関するノードであるので，本研究ではコメント文に関するノードを除いたものを説明変数とする．つまり本研究では 81 個の説明変数が存在することになる．構文情報の例を**図 3** に示す．構文情報を取得した後，学習モデルを構築するアルゴリズムとして，Decision Tree [12]，Random Forest [13]，Naive Bayes [14]，SVM [15] 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．また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 [16]．そこで，すべ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法とラッパ法が知られており，多くの場合ラッパ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，本研究ではラッパ法を用いる [17]．

3.2 Step 2：学習モデルの適用

Step 2 では，テストデータに対して学習モデルを適用する．そのために，自動生成ファイルか否かを判定したいソースファイルをテストデータとして用意する．用意したテストデータの構文情報を Step 1 と同様に取得する．テストデータにおける説明変数は，Step 1 で，学習データにおいて選択されたものと同じである．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，自動生成ファイルか否かを判定する．

4. 評価実験

本章では，提案手法を評価するために行った 3 つの実験と，その実験結果について述べる．

4.1 実験概要

提案手法の評価を行うために，3 つの実験を行った．その概要を以下に示す．

実験 1 提案手法の Step 1 において構築する学習モデルの評価をするため，交差検証を行う．

実験 2 提案手法の Step 2 において，どの程度テストデータの中から自動生成ファイルを特定できるか確認するため，大規模なソースファイルの集合を学習モデルに適用する．

実験 3 ファイル名検索による自動生成ファイル特定手法が提案手法における誤検出を防ぐことができるか確認するため，2 つの手法を組み合わせた上で交差検証を行う．

4.2 実験対象

学習データとして，2.1 節で述べたデータセットを用いた．加えて，4 種類の自動生成ファイル計 4,000 ファイルの中から 1,000 ファイルをランダムで抽出した．このソースファイル群を **MIX ファイル群**と定義する．また，実験 2 におけるテストデータには，大規模なソースファイルの集合である *UCI Source Code Data Sets* [18]（以降，***UCI datasets*** と呼ぶ）を用いた．

表 5 交差検証の精度
Table 5 Results of cross validation.

アルゴリズム	ANTLR		JavaCC		JFlex		SableCC		MIX	
	P	R	P	R	P	R	P	R	P	R
Decision Tree	99.9%	99.9%	97.0%	97.0%	99.4%	99.4%	96.3%	96.2%	95.3%	95.3%
Naïve Bayes	98.8%	98.8%	88.0%	85.7%	99.5%	99.5%	82.7%	78.4%	85.3%	80.6%
Random Forest	99.9%	99.9%	97.3%	97.3%	99.7%	99.7%	96.1%	96.1%	96.8%	96.8%
SVM	99.8%	99.8%	95.7%	95.7%	99.6%	99.6%	86.8%	84.5%	85.2%	79.1%

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 [21] を使用した．

4.3 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ファイルと判定されたソースファイルのうち，実際に自動生成ファイルであるものの割合

Recall 実際に自動生成ファイルであるもののうち，学習モデルによって自動生成ファイルと判定されたものの割合

実験 1 および実験 3 の評価尺度は *Precision* と *Recall* の両方を算出した．実験 2 では，テストデータとして用いる *UCI datasets* のうち，実際に自動生成であるファイルの数が未知であるため，*Precision* のみ算出した．以下，各実験結果について述べる．

4.4 実験 1

収集した各自動生成ファイル群を用いて 4 種類の学習モデルを構築した．さらに，MIX ファイル群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予測精度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 5** に示す．ただし，表中の P と R はそれぞれ *Precision* と *Recall* を表す．多くの場合で *Precision*，*Recall* ともに 90%を超えている．特に，ANTLR 生成ファイルと JFlex 生成ファイルではほとんどの場合で 99%と，非常に高い．*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 生成ファイルと JFlex 生成ファイルに比べて，JavaCC 生成ファイルと SableCC 生成ファイル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，Naïve Bayes の精度が最も低いことが分かる．

表 6 *UCI datasets* に学習モデルを適用した結果
Table 6 Results of applying learning models to *UCI datasets*.

全ファイル数		146,346
特定された自動生成ファイルの数	MIX	837
	ANTLR	157
目視確認により自動生成ファイルと判断された数	MIX	552
	ANTLR	106
Precision	MIX	65.9%
	ANTLR	67.5%

4.5 実験 2

実験 1 では，構築した学習モデルの予測精度を評価していた．実験 2 では，自動生成ファイルかどうか判明していない未知のソースファイル群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ファイルと判定されたソースファイルが，実際に自動生成ファイル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ソースファイルの集合である *UCI datasets* すべてに学習モデルを適用すると時間的コストが大きくなるため，*UCI datasets* の一部を用いた．ただし，学習データとテストデータは異なるファイル群である．このテストデータを，MIX ファイル群を用いて構築した学習モデルと実験 1 において最も精度が高かった ANTLR 生成ファイル群を用いて構築した学習モデルにそれぞれ適用した．これは，複数種類の自動生成ファイルを特定する際，学習データとして MIX ファイル群を用いるべきか単一種類の自動生成ファイル群を用いるべきか確認するためである．

UCI datasets に対して学習モデルを適用した結果を**表 6** 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 で最も精度が高かった Random Forest を用いている．表中の *Precision* から，MIX ファイル群を用いて構築した学習モデルよりも ANTLR 生成ファイル群を用いて構築した学習モデルの方が精度が高いことが分かる．つまり，複数種類の自動生成ファイルを特定する場合，自動生成ファイルの種類ごとの学習モデルを構築した方がよいと考えられる．また予測結果の中には，実際にコメント文が消された自動生成ファイルが含まれていた．

表 7 パターン 1 の結果
Table 7 Results of pattern 1.

アルゴリズム	ANTLR		JavaCC		JFlex		SableCC	
	P	R	P	R	P	R	P	R
Decision Tree	98.7%	99.9%	99.2%	99.7%	99.6%	99.6%	95.9%	99.4%
Naive Bayes	97.7%	100%	92.8%	99.8%	99.7%	99.7%	90.4%	99.8%
Random Forest	98.9%	99.9%	99.4%	99.9%	99.8%	99.7%	97.3%	99.7%
SVM	98.9%	100%	98.4%	99.8%	99.5%	99.8%	94.1%	98.9%

表 8 パターン 2 の結果
Table 8 Results of pattern 2.

アルゴリズム	ANTLR		JavaCC		JFlex		SableCC	
	P	R	P	R	P	R	P	R
Decision Tree	100%	89.1%	100%	38.7%	100%	7.9%	100%	81.5%
Naive Bayes	100%	89.4%	99.8%	38.6%	100%	8.0%	99.6%	82.1%
Random Forest	100%	89.2%	100%	38.7%	100%	8.0%	100%	82.1%
SVM	100%	88.8%	99.9%	34.0%	100%	8.0%	99.9%	79.9%

4.6 実験 3

実験 3 では，ファイル名検索による手法が提案手法における誤検出を防いで予測精度が向上するかどうか確認するため，提案手法と 2.2 節で述べたファイル名検索による手法を組み合わせた．組合せの方法として 2 つのパターンで交差検証を行い，*Precision* と *Recall* を算出した．なお，ファイル名検索による手法は自動生成プログラムごとの生成規則を用いるので，MIX ファイル群に対しては適用していない．以下，2 つのパターンについて説明する．

パターン 1

1. ファイル名検索により自動生成ファイルかどうか判定する．
2. **1.** で自動生成ファイルと判定されなかったものを学習モデルによって判定する．

パターン 2

1. ファイル名検索により自動生成ファイルかどうか判定する．
2. **1.** で自動生成ファイルと判定されたもののうち，学習モデルによっても自動生成ファイルと判定されたものを自動生成ファイルとみなす．

パターン 1 では，ファイル名検索による特定と機械学習による特定の，少なくとも一方で自動生成ファイルと判定されたものを自動生成ファイルとみなす．パターン 2 では，ファイル名検索による特定と機械学習による特定の両方に自動生成ファイルと判定されたものを自動生成ファイルとみなす．パターン 1 による検出結果とパターン 2 による検出結果をそれぞれ**表 7** と**表 8** 示す．

表 7 から，ほとんどの場合で Recall が 100%近い値になっていることが分かる．実験 1 の結果と比較すると，多くの場合で *Precision* が低下していることが分かる．これは，ファイル名特定による誤検出が増えたことが要因と考えられる．

また，表 8 から，ほとんどの場合で *Precision* が 100%近い値になっていることが分かる．また，実験 1 の結果と比較すると，*Recall* が大幅に低下していることが分かる．

以上のことから，*Recall* を 100%に近づけたい場合はパターン 1 を用いればよい．また，*Precision* を 100%に近づけたい場合はパターン 2 を用いればよい．その場合，実験 1 および実験 2 における誤検出を減少させることができると考えられる．

5. 考察

本章では，4 章で述べた評価実験，および提案手法における有用性についての考察を行う．

5.1 実験 1 の考察

まず，実験 1 の結果について考察する．交差検証の結果において，誤検出されたファイルを調べた．その結果，誤って自動生成ファイルと判定されたものの特徴として，以下のことがあげられる．

- 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
- switch-case 文やリテラルが多い

ファイルサイズが小さいものには，インタ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ファイル群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，switch-case 文やリテラルは，2 章で述べたように自動生成ファイルの特徴であるので，それらが誤検出の要因であると考えられる．

次に，学習データの変数選択において，どの説明変数が重要であるかを調べた．選択された説明変数のうち，重要

表 9 変数選択における説明変数のランキング
Table 9 Ranking of explanatory variables in variable selection.

自動生成プログラム	Top 1	Top 2	Top 3
ANTLR	switch-case 文	修飾子	配列型
JavaCC	数リテラル	代入	switch-case 文
JFlex	ラベル付きステートメント	配列型	while 文
SableCC	import 宣言	配列型	プリミティブ型

度の高い上位 3 位を**表 9** に示す．表から switch-case 文および配列型が複数の自動生成プログラムにまたがって登場していることが分かる．また，自動生成プログラムによって説明変数の重要度は異なっているということが分かる．

5.2 実験 2 の考察

実験 2 の結果について考察する．誤って自動生成ファイルであると判定されたファイルの中身を確認した．そのようなファイルの中身は，switch-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，switch-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．自動生成でないファイルが自動生成ファイルであると検出されないために，構文情報以外の説明変数を追加するなど，提案手法の改善が必要となる．

実験 2 の結果は，実験 1 における交差検証の結果と比べると *Precision* が低下している．これは，学習データにおける自動生成ファイルの割合と，テストデータにおける自動生成ファイルの割合が異なることが要因と考えられる．実験 2 の結果から，テストデータにおける自動生成ファイルと自動生成でないファイルでは，自動生成でないファイルの方が数多く存在することが分かる．一方，学習データにおいては自動生成ファイルと自動生成でないファイルが等しい数だけ用いられている．したがって，本実験で用いた学習データには偏りが生じている．これは，不均衡データ問題として知られており，学習モデルの予測精度の低下を招く [22]．改善方法としては以下のようなものが考えられる．

- 学習データに重み付けを行う
- 学習データのファイル数の調整を行う

また，MIX ファイル群を用いて構築した学習モデルと JFlex 生成ファイル群を用いて構築した学習モデルを比較すると，後者の方が精度が高かった．このことから，未知のソースファイル群から自動生成ファイルを特定したい場合，学習データとして複数種類の自動生成ファイル群を用いるのではなく，1 種類の自動生成ファイル群を用いて学習モデルを構築した方が良いと考えられる．

5.3 実験 3 の考察

実験 3 の結果について考察する．2.2 節で述べた，ファイル名による自動生成ファイル特定の結果から，*Precision* および *Recall* を算出すると**表 10** のようになる．表から，*Precision* は提案手法による結果と同程度の精度であるが，*Recall* は提案手法の方が大幅に優れていることが分かる．このことから，ファイル名検索によって特定できない自動生成ファイルも，提案手法では特定することが可能であるといえる．

表 10 ファイル名検索による自動生成ファイル特定の結果
Table 10 Precision and recall of identifying auto-generated files by searching file names.

自動生成プログラム	*Precision*	*Recall*
ANTLR	98.4%	89.4%
JavaCC	98.5%	76.7%
JFlex	100%	7.9%
SableCC	96.1%	86.7%

また，実験 3 の結果はパターンごとに見れば実験 1 の結果よりも優れているため，提案手法とファイル名検索による手法を組み合わせるのが最も有用であると考えられる．ただし，ファイル名検索による手法はファイル名規則を調査するコストが必要となる．

5.4 ファイル数と予測精度との関係

評価実験で用いた学習データにおける自動生成ファイルは 1,000 ファイルであるが，このファイル数が学習モデルを構築する上で十分な量であるかは定かではない．そこで，学習データの数を変化させながら実験 2 と同様の評価を行った．自動生成ファイルには実験 2 と同様に ANTLR 生成ファイル群および MIX ファイル群を用いており，200 ファイル刻みで変化させている．また，ANTLR 生成ファイル群，MIX ファイル群および自動生成でないファイル群の平均 LOC は，それぞれ 2,500，1,260，360 であり，ファイス数が 200，400，600，800 のいずれの場合でも均一となるようにしている．その結果を**表 11** に示す．表中の値は *Precision* を表す．表 11 からは，ファイル数と予測精度の間に相関関係が見られないことから，予測精度の向上には学習データの量を増やすだけでなく，5.2 節で述べたような方法が必要であると考えられる．また，ANTLR 生成ファイル群と MIX ファイル群を全体的に比較すると，前者は後者よりも少ない学習データ量で精度を保つことが可能であると考えられる．

表 11 ファイル数と予測精度との関係
Table 11 Relation between the number of files and precision of prediction.

ファイル数	ANTLR	MIX
200	58%	48%
400	60%	28%
600	61%	35%
800	55%	50%

5.5 提案手法が有用な場面

ソースコード解析を行う者自身が解析対象のソフトウェアに精通している場合，どのような自動生成ツールが用いられているのか，どのファイルが自動生成されたものなのかについての知識がある．そのような場合，提案手法を用いて自動的に自動生成ファイルを特定する必要性は少ない．しかし，ソースコード解析を行う者が上記のような知識を持っていない場合，自動生成ファイルを自動的に特定するのは困難である．したがって，そのような場合，提案手法は有用であると考えられる．

また，実験 2 において，特定された自動生成ファイルのうち，約半数がコメントが消された自動生成ファイルであった．これらはコメント文検索では特定できないものである．提案手法を用いることによって，より多くの自動生成ファイルを自動的に特定することが可能となる．

5.6 実行時間について

次に，実行時間に関する考察を行う．実験 2 を行ううえで要した実行時間を以下に示す．

- 学習データの構文情報取得：114 秒
- 学習モデルの構築：1.26 秒
- テストデータの構文情報取得：35 分
- テストデータの分類：1.5 秒

最も時間を要するフェーズは構文情報の取得であることが分かる．実際のソースコード解析における実行時間と比較すると，佐々木らは，約 10 GByte のソースコードからコードクローンを検出する際に約 40 日間を要している [23]．本実験でテストデータとして用いた *UCI datasets* は約 1 GByte である．このことから，機械学習による自動生成ファイルの分類は，現実的な時間で実行可能であると考えられる．

6. 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．

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

実験 2 において，学習モデルによって自動生成ファイルと判定されたものが，実際に自動生成ファイルであるかどうかは目視確認によるものである．そのため，実際には自動生成ファイルではない可能性がある．

7. おわりに

本研究では，機械学習を用いて自動生成ファイルを自動的に特定する手法を提案した．提案手法では，自動生成ファイル特有のコメント文の有無にかかわらず，自動生成ファイルか否かを判定するために，与えられたソースファイル群の構文情報を取得し，それらから学習モデルを構築した．

実験として，4 種類の自動生成ファイルを収集し，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ファイルを特定できていることを確認した．

さらに，自動生成ファイルかどうか判明していないソースファイル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%の精度で自動生成ファイルを特定できることを確認した．しかしこの実験では，一部の自動生成ファイルで構築した学習モデルしか適用していないため，他の自動生成ファイル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：JP25220003）の助成を得て行われた．

参考文献

[1] McDonald, P., Strickland, D. and Wildman, C.: Estimating the effective size of autogenerated code in a large software project, *Proc. 17th International Forum on COCOMO and Software Cost Modeling* (2002).
[2] Uchida, S., Monden, A., Ohsugi, N., Kamiya, T., Matsumoto, K. and Kudo, H.: Software analysis by code clones in open source software, *Journal of Computer Information Systems*, Vol.45, No.3, pp.1–11 (2005).
[3] 大田崇史，井垣 宏，堀田圭祐，肥後芳樹，楠本真二：ソフトウェア開発におけるコピーアンドペーストによって生じたコード片に対する調査，情報処理学会研究報告，ソフトウェア工学研究会報告，Vol.2014, No.22, pp.1–6 (2014).
[4] Göde, N. and Koschke, R.: Frequency and risks of changes to clones, *Proc. 33rd International Conference on Software Engineering*, pp.311–320, ACM (2011).
[5] Harder, J. and Göde, N.: Cloned code: Stable code, *Journal of Software: Evolution and Process*, Vol.25, No.10, pp.1063–1088 (2013).
[6] MacLean, A.C., Pratt, L.J., Krein, J.L. and Knutson, C.D.: Trends that affect temporal analysis using sourceforge data, *Proc. 5th International Workshop on Public Data about Software Development* (*WoPDaSD'10*), p.6, Citeseer (2010).
[7] GitHub, available from ⟨http://github.com/⟩.

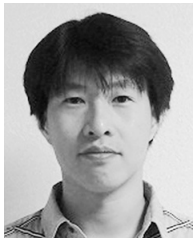
[8] 高澤亮平，坂本一憲，鷲崎弘宜，深澤良彰：Repositoryprobe：リポジトリマイニングのためのデータセット作成支援ツール，コンピュータソフトウェア，Vol.32, No.4, pp.4_103–4_114 (2015).
[9] jsoup: Java HTML Parser, available from ⟨http://jsoup.org/⟩.
[10] Apache Source code repository, available from ⟨http://svn.apache.org/repos/asf/⟩.
[11] Gagnon, E.M. and Hendren, L.J.: Sablecc, an object-oriented compiler framework, *Proc. Technology of Object-Oriented Languages* (*TOOLS 26*), pp.140–154, IEEE (1998).
[12] Breiman, L., Friedman, J., Stone, C.J. and Olshen, R.A.: *Classification and regression trees*, CRC press (1984).
[13] Breiman, L.: Random forests, *Machine learning*, Vol.45, No.1, pp.5–32 (2001).
[14] Domingos, P. and Pazzani, M.: On the optimality of the simple bayesian classifier under zero-one loss, *Machine learning*, Vol.29, No.2-3, pp.103–130 (1997).
[15] Vapnik, V.: Pattern recognition using generalized portrait method, *Automation and Remote Control*, Vol.24, pp.774–780 (1963).
[16] 小川英光，山崎一孝：過学習の理論，電子情報通信学会論文誌 D，Vol.76, No.6, pp.1280–1288 (1993).
[17] Hall, M.A. and Holmes, G.: Benchmarking attribute selection techniques for discrete class data mining, *IEEE Trans. Knowledge and Data Engineering*, Vol.15, No.6, pp.1437–1447 (2003).
[18] Uci Source Code Data Sets, available from ⟨http://www.ics.uci.edu/~lopse/datasets/⟩.
[19] Murakami, H., Hotta, K., Higo, Y. and Kusumoto, S.: Predicting next changes at the fine-grained level, *Software Engineering Conference* (*APSEC*), *2014 21st Asia-Pacific*, Vol.1, pp.119–126, IEEE (2014).
[20] Eclipse Java development tools (JDT), available from ⟨http://www.eclipse.org/jdt/⟩.
[21] Weka 3: Data Mining Software in Java, available from ⟨http://www.cs.waikato.ac.nz/ml/weka/⟩.
[22] Japkowicz, N. et al.: Learning from imbalanced data sets: A comparison of various strategies, *AAAI Workshop on Learning from Imbalanced Data Sets*, Vol.68, pp.10–15, Menlo Park, CA (2000).
[23] 佐々木裕介，山本哲男，早瀬康裕，井上克郎：大規模ソフトウェアシステムを対象としたファイルクローンの検出，電子情報通信学会論文誌 D，Vol.94, No.8, pp.1423–1433 (2011).

下仲 健斗

2016 年大阪大学基礎工学部情報科学科卒業．現在，同大学大学院情報科学研究科博士前期課程在学中．ソースコード分析に関する研究に従事．

鷲見 創一

2015 年岐阜工業高等専門学校専攻科電子システム工学専攻修了．現在，大阪大学大学院情報科学研究科博士前期課程在学中．リポジトリマイニングに関する研究に従事．



肥後 芳樹 （正会員）

2002 年大阪大学基礎工学部情報科学科中退．2006 年同大学大学院博士後期課程修了．2007 年同大学院情報科学研究科コンピュータサイエンス専攻助教．2015 年同准教授．博士（情報科学）．ソースコード分析，特にコードクローン分析やリファクタリング支援に関する研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE 各会員．

楠本 真二 （正会員）

1988 年大阪大学基礎工学部卒業．1991 年同大学大学院博士課程中退．同年同大学基礎工学部助手．1996 年同講師．1999 年同助教授．2002 年同大学大学院情報科学研究科助教授．2005 年同教授．博士（工学）．ソフトウェアの生産性や品質の定量的評価に関する研究に従事．電子情報通信学会，IEEE，IFPUG 各会員．