情報処理学会論文誌 Vol.60 No.2 642–650 (Feb. 2019)

言語モデルによるソースコードの「自然さ」を利用した自動生成ファイルの特定

土居 真之[1,a)] 肥後 芳樹[1,b)] 有馬 諒[1,c)] 下仲 健斗[1,d)] 楠本 真二[1,e)]

受付日 2018年4月6日, 採録日 2018年11月7日

概要：ソースコードの解析において，解析対象のソースファイルの中には自動生成ファイルが含まれていることがある．自動生成ファイルの存在が解析に悪影響を及ぼす場合があるため，多くの場合自動生成ファイルは除外して解析する必要がある．自動生成ファイルを除外する方法として，ソースコードが自動生成ファイルであるかを目視で判定するという方法がある．しかしこの方法は時間的コストが大きくなってしまうといった問題がある．他にも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が，この方法に関しても，自動生成ファイル特有のコメント文が消された場合に，自動生成ファイルを自動的に特定できないといった問題がある．そこで本研究では，自動生成コードとしての「自然さ」と人が作成したコードとしての「自然さ」を比較することで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．コードの自然さ，すなわち，自動生成あるいは人が生成したコードとしてもっともらしい度合いは，確率的言語モデルである N-gram 言語モデルによって数値化する．この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，高い精度で自動生成ファイルを特定できた．

キーワード：自動生成コード，N-gram 言語モデル，ソースコード解析

Identification of Auto-generated Files Using Naturalness of Source Code by Language Model

Masayuki Doi[1,a)] Yoshiki Higo[1,b)] Ryo Arima[1,c)] Kento Shimonaka[1,d)] Shinji Kusumoto[1,e)]

Received: April 6, 2018, Accepted: November 7, 2018

Abstract: In source code analysis, target source files include auto-generated files in some cases. However, auto-generated file may adversely affect the source code analysis, and so it is often necessary to exclude the auto-generated files before analyzing. A way of excluding auto-generated files is visually determining whether each source file is an auto-generated file or not, but it takes too much time to see source files manually. Another way is searching special comments which are included in the auto-generated file, An issue of the way is when such special comments have deleted for some reasons, the file cannot be identified automatically. Therefore, in this technique, we propose a way to automatically identify auto-generated files by comparing "naturalness" as an auto-generated file and as a handwritten file. The naturalness of the files, that is, the degree that is likely to be auto-generated or handwritten code, is quantified by a N-gram language model which is a probabilistic language model. In order to evaluate the proposed techinique, experiments were conducted on datasets which are groups of generated files from four auto-generated Java files, handwritten Java files, JavaScript files translated from TypeScript files and JavaScript files. As a result, we were able to identify auto-generated files with very high accuracy.

Keywords: auto-generated code, N-gram language model, source code analysis

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

1. まえがき

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，コードクローンに関する研究[*1]や，リポジトリマイニングに関する研究[*2]などが行われている．ソースコード解析において，解析対象のソフトウェアに含まれるソースコードファイルのなかには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．たとえば，コードクローン検出結果を分析して開発プロセスに対する知見を得ようとする場合，自動生成ファイルの存在が弊害となる．多くの場合，自動生成ファイルから検出されたコードクローンは，分析対象とはしない．これは自動生成ファイルから検出されるコードクローンは自動生成ツールにより生成されたコードであり，コピーアンドペーストによって生成されたコードではないためである．したがって，UCI データセット[*3]のような自動生成ファイルを含んだ複数のソフトウェアを対象としてコードクローン検出を行うと，開発者の書いたソースファイルに存在するコードクローンが目立たなくなってしまうことが知られている [11], [14]．実際に既存研究ではライブラリ化可能なコードクローンを検出する研究で自動生成ファイルを対象から手動で除去して実験している [13]．また，リポジトリマイニングにおいては，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 [10]．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，そのようなソースコードに対しては grep コマンド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しまう場合がある．その場合 grep コマンドによる特定は難しく，またコメント文が消された自動生成ファイル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場合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用いる必要がある．しかし任意の自動生成ファイル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，N-gram 言語モデルを用いて，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自動的に判定する手法を提案する．

[*1] International Workshop on Software Clones（IWSC），https://iwsc2018.github.io/
[*2] Mining Software Repositories（MSR），https://conf.researchr.org/home/msr-2018
[*3] http://www.ics.uci.edu/~lopse/datasets/

N-gram 言語モデルとは，既存のコードの字句の並びからモデルを生成することにより，未知のファイルのモデルに対する自然さ，すなわちモデルらしさを数値として出力する統計的言語モデルである．提案手法では自動生成ファイルだと判明しているソースファイル，および自動生成でないファイルと判明しているソースファイルをそれぞれ収集する．収集したソースファイルから，自動生成ファイルの言語モデルと自動生成でないファイルの言語モデルを生成する．それらを用いて未知のソースコードの自動生成ファイルとしての自然さと自動生成でないファイルとしての自然さをそれぞれ求め，これらを比較することによって自動生成ファイルかどうかを判定する．

提案手法を評価するために，4 種類の自動生成 Java ファイルと JavaScript に変換した TypeScript ファイルによる評価実験を行った．実験の結果，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．

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する．そのなかでも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 Java ファイルと，JavaScript に変換された TypeScript を対象とする．

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．

- 変数宣言文が連続して出現することが多い．
- 似た条件式が連続して出現することが多い．

自動生成ファイルのソースコードの一部を**図 1** 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

```
/* Generated By:JavaCC: Do not edit this line. Func2Parser.java */

      . . .

  if ((active0 & 0x400L) != 0L)
         return jjStartNfaWithStates_0(2, 10, 10);
  else if ((active0 & 0x4000L) != 0L)
         return jjStartNfaWithStates_0(2, 14, 10);
  else if ((active0 & 0x40000L) != 0L)
         return jjStartNfaWithStates_0(2, 18, 10);
  else if ((active0 & 0x400000L) != 0L)
         return jjStartNfaWithStates_0(2, 22, 10);
  else if ((active0 & 0x800000L) != 0L)
         return jjStartNfaWithStates_0(2, 23, 10);
  else if ((active0 & 0x10000000L) != 0L)
         return jjStartNfaWithStates_0(2, 28, 10);
  else if ...
```

図 1 自動生成ファイルの例
Fig. 1 An example of auto-generated code.

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．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コメント文が削除されてしまう場合があるため，この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が必要である．そこで本研究では，著者らが過去の研究で作成した Java ファイルのデータセット [12] と，今回新しく作成した JavaScript のデータセットを用いた．

まず Java ファイルのデータセット [12] について説明する．Java ファイルのデータセットは 4 種類のパーサジェネレータ（ANTLR，JavaCC，JFlex，SableCC）により生成された 4 つの自動生成ファイル群と自動生成でないファイル群からなる．

自動生成ファイルの収集は GitHub [5] からコメント文検索し，jsoup [9] を用いたウェブスクレイピングにより自動で行った．自動生成でないファイルの収集は大規模なソースファイルの集合である Apache リポジトリ[*4]からソースファイルをランダムに収集した．

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見なしているもの，もしくは誤って自動生成でないファイルと見なしているもの（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．

[*4] Apache Source code repository, http://svn.apache.org/repos/asf/

表 1 データセット作成時におけるノイズデータ含有率
Table 1 Ratio of noise data in making data set.

ファイル群	チェックしたファイル数	ノイズデータ含有率
ANTLR 生成ファイル群	1,000	0.0%
JavaCC 生成ファイル群	1,006	0.6%
JFlex 生成ファイル群	1,010	1.0%
SableCC 生成ファイル群	1,004	0.4%
自動生成でないファイル群	1,032	3.0%

この 1,000 ファイルを収集する過程でファイル群に含まれていたノイズデータの割合を**表 1** に示す．コメント文検索により人が書いたのに自動生成ファイルと判定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．

次に JavaScript ファイルのデータセットについて説明する．JavaScript ファイルのデータセットは，JavaScript に変換された TypeScript のファイル群と JavaScript のファイル群からなる．

TypeScript ファイルの収集は Github から拡張子検索により行った．その後収集したファイルを tsc コマンドで JavaScript へ変換することで，JavaScript に変換された TypeScript ファイルを生成した．JavaScript ファイルも同様に Github から拡張子検索により収集した．収集した JavaSript ファイルには TypeScript が JavaScript に変換されたファイルが存在する可能性がある．そのためノイズデータの除去を行った．

これにより JavaScript に変換された TypeScript ファイルと JavaScript 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．

2.2 ファイル名検索による自動生成 Java 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 [3]．

- 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，ファイル名検索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 2** 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

表 2 自動生成プログラムのファイル名生成規則
Table 2 Filename generation rule of auto-generated file.

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 3 ファイル名検索による自動生成ファイル特定の結果
Table 3 result of identification auto-generated file by filename search.

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

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見なす．

ファイル名検索による自動生成ファイル特定の結果を**表 3** 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 機械学習を利用した自動生成ファイルの特定

コメント文検索とファイル名による検索以外の自動生成ファイル特定手法として，機械学習を用いて特定する手法がある [12]．これは自動生成ファイルと自動生成でないファイルの構文情報の出現回数を学習データを用いてモデルを構築し，このモデルによって自動生成ファイルか否かを判定する手法である．この手法を 2.1 節で述べたデータセットに対して 4 種類の機械学習アルゴリズム（Decision Tree，Naive Bayes，Random Forest，SVM）を用いた場合の適合率の最大値と最小値は 99.9%，82.7%，再現率の最大値と最小値は 99.9%，78.4%である．

またファイルサイズが 10 kB 以下の小さいファイルに対して適用した場合，精度が悪くなるといった課題がある．したがってファイルサイズに依存せず高精度で自動生成ファイルを特定する手法が必要であり，これが本論文の研究動機である．

2.4 N-gram 言語モデルと自然さ

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字句の並びに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，字句の並びにより自然さを求める N-gram 言語モデル [2] 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．N-gram 言語モデルとは，大規模なテキストデータを統計的に解析し，直前の $\mathrm{N}-1$ 個の単語列から次の単語への遷移確率を与えるモデルである．この N-gram 言語モデルによって求まる値を自然さと呼ぶ．自然さは，式 (1)，(2) で計算される [8]．

なお式 (1)，(2) における W は入力となるテキストデータであり，出力として自然さ $P(W)$ を得る．また W を単語に分解した場合の i 番目の単語が w_i であり，$c(w)$ は単語 w の出現回数である．

$$P(W) = \prod_{i=1}^{|W|+1} p(w_i|w_0 \dots w_{i-1}) \tag{1}$$

$$p(w_i|w_0 \dots w_{i-1}) = \frac{c(w_{i-n+1} \dots w_i)}{c(w_{i-n+1} \dots w_{i-1})} \tag{2}$$

この自然さを用いることで自動生成ファイルの特定を行う．

3. 提案手法

本研究では，N-gram 言語モデルを利用し，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，すなわち自動生成ファイル群および自動生成でないファイル群と，テストデータ，すなわち自動生成ファイルか判定したいファイルである．また出力はテストデータが自動生成ファイルか否かの判定結果である．

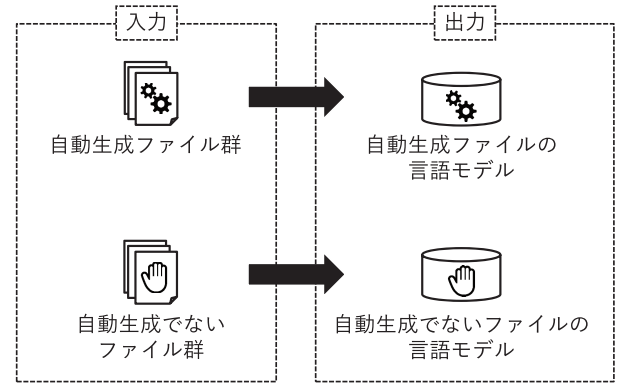
提案手法は次の 2 ステップから構成される．Step1 では，学習データから自動生成ファイルと自動生成でないファイルそれぞれの言語モデルを構築する．Step2 では，Step1 で構築した各言語モデルに対して自動生成ファイルかどうかを判定したいテストデータを適用し，得られた自然さの比較を行う．以降，各ステップについて詳細に説明する．

Step1：言語モデルの構築

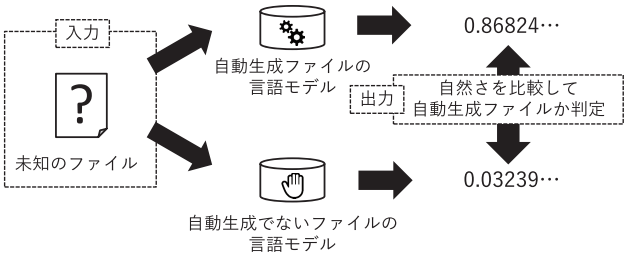
Step1 では，学習データから字句の並びを取得し，言語モデルを構築する．よって Step1 における入力は学習データであり，出力は学習データの各ファイル群より生成された自動生成ファイルの言語モデルと自動生成でないファイルの言語モデルである．まず学習データの各ファイル群に対して字句解析を行い字句の並びを取得する．この字句の並びから N-gram 言語モデルを構築することによって自動生成ファイルの言語モデルと自動生成でないファイルの言語モデルを得る．

Step2：言語モデルの適用

Step2 では，自動生成ファイルであるか判定したいソースファイルに対して Step1 で作成した 2 つの言語モデルを適用する．よって Step2 における入力はテストデータであり，出力はテストデータが自動生成ファイルか否かの判定結果である．まずテストデータを Step1 と同様に字句解析を行い，その結果を言語モデルに適用し，得られた自然さを比較することで自動生成ファイルか否か判定する．すなわち，自動生成ファイルとしての自然さの方が高ければ自動生成ファイル，そうでないならば自動生成でないと判定する．



(a) Step 1



(b) Step 2

図 2 提案手法の概要

Fig. 2 Overview of the proposed technique.

4. 実装

本章では提案手法の実装方法について述べる．

4.1 字句解析

Java ファイルの字句解析には Eclipse JDT 4.5.2 [*5] を用いて AST 解析を行った結果を用いる．Eclipse JDT 4.5.2 では 92 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 89 種類のノードからなる AST を構成する字句列を抽出して，その字句の並びから自然さの計測を行う．

JavaScript ファイルの字句解析にはパーサジェネレータである ANTLR を用いて AST 解析を行った結果を用いる．ANTLR の JavaScript 文法ファイルには 110 種類の AST ノードが定義されているが，そのうち 4 つはコメントに関するトークンであるため，コメントに関するトークンを除いた 106 種類のノードからなる AST を構成する字句列を抽出して，その字句の並びから自然さの計測を行う．

字句解析および自然さの計測の例を図 3 に示す．

4.2 N-gram 言語モデルの構築と適用

N-gram 言語モデルの構築と適用には KenLM [6] を用いる．なお N-gram 言語モデルは学習データ中に存在しない N-gram がテストデータで出現した場合，自然さが 0 になってしまうという問題がある．これを避けるためにスムージングを行う．スムージングの方法としていくつか種類があるが，KenLM では実験的に良いとされている modified Kneser-Ney smoothing を用いられている [7]．

5. 評価実験

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

提案手法の評価を行うために，実験を行った．その概要を以下に示す．

実験 1 Java を対象に提案手法の精度を交差検証により評価した．

[*5] Eclipse Java development tools (JDT), http://www.eclipse.org/jdt/

表 4 交差検証による精度評価の結果
Table 4 Precision and recall of our technique and Shimonaka's one with leave-one-out method.

自動生成プログラム		ANTLR		JavaCC		JFlex		SableCC		MIX	
		適合率	再現率	適合率	再現率	適合率	再現率	適合率	再現率	適合率	再現率
提案手法		100.0%	100.0%	99.6%	99.3%	99.8%	99.7%	100.0%	99.9%	98.7%	99.7%
下仲らの手法 [12]	Decision Tree	99.9%	99.9%	97.0%	97.0%	99.4%	99.4%	96.3%	96.2%	95.3%	95.3%
	Naive 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%

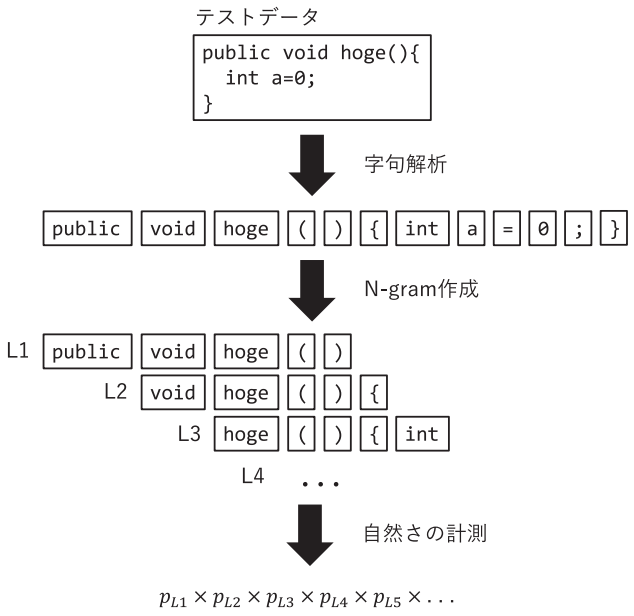


図 3 字句解析と自然さ計測の例
Fig. 3 Calculating naturalness from unknown file.

実験 2 Java を対象に提案手法の精度をブートストラップ法により評価した．

実験 3 JavaScript に変換された TypeScript を対象に提案手法の精度を交差検証により評価した．

実験 4 JavaScript に変換された TypeScript を対象に提案手法の精度をブートストラップ法により評価した．

5.1 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 Java 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．

5.2 評価尺度

実験の評価尺度として，適合率と再現率を用いる．以下，それぞれの尺度の定義について説明する．

適合率 自然さを比較して自動生成ファイルと判定されたファイルのうち，実際に自動生成ファイルであるものの割合

再現率 実際に自動生成ファイルであるもののうち，自然さを比較して自動生成ファイルと判定されたものの割合

5.3 実験 1

まず収集した各自動生成ファイル群と MIX ファイル群から言語モデルを構築した．同様に自動生成でないファイル群からも言語モデルを構築し，計 6 種類の言語モデルを得た．なお本実験では N-gram 言語モデルのオーダーは 5 として構築した．構築された計 6 種類の言語モデルの性能を評価するために交差検証による検証を行った．

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，すべてのブロックが 1 度テストデータとなるよう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 $N = 10$ として言語モデルの評価を行った．またその結果を下仲らによる機械学習の 4 種類のアルゴリズム（Decision Tree，Naive Bayes，Random Forest，SVM）を用いた手法と比較した [12]．その結果を**表 4** に示す．

提案手法ではすべての場合において適合率，再現率ともに 98%を超える精度になっていることが分かる．また機械学習による判定結果と比較すると，すべての場合において精度が同じもしくは提案手法の方が精度が高くなっていることが分かる．

5.4 実験 2

実験 1 では，言語モデルの性能を交差検証により評価した．一方，交差検証は実際の精度に近い精度をだすとされているが，データセットからブロックを作成する際に，データの偏りが生じて実験結果の分散が大きくなるといった問題がある [1]．その問題を回避するため，実験 2 ではブートストラップ法による検証を行った．ブートストラップ法では，まずデータセットからデータセットと同じ数だけのファイルをランダムに抽出し，抽出したファイルで新しいデータセットを作成することを繰り返して複数のデータセットを得る．この複数のデータセットにおける精度の

表 5 ブートストラップ法による精度評価の結果
Table 5 Precision and recall of our technique derived from bootstrapping method.

自動生成プログラム	ANTLR		JavaCC		JFlex		SableCC		MIX	
	適合率	再現率	適合率	再現率	適合率	再現率	適合率	再現率	適合率	再現率
提案手法	99.99%	99.98%	99.86%	99.75%	99.97%	99.86%	100.0%	99.92%	99.42%	99.84%

表 6 JavaScript と TypeScript の交差検証による精度評価の結果
Table 6 Precision and recall of our technique to JavaScript with leave-one-out method.

	適合率	再現率
提案手法	98.1%	98.1%

表 7 JavaScript と TypeScript のブートストラップ法による精度評価の結果
Table 7 Precision and recall of our technique to JavaScript with bootstrap method.

	適合率	再現率
提案手法	98.12%	98.74%

平均を計算することで，元のデータセットの評価を行う．ブートストラップ法における必要な繰り返し回数は 50 回から 2,000 回で十分とされているため，本実験では 2,000 個のデータセットを作成し評価を行った [4]．その結果を**表 5** に示す．

提案手法では適合率，再現率ともに 99%を超える精度になっていることが分かる．

5.5 実験 3

実験 1，2 では，パーサジェネレータで生成された Java ファイルを対象に実験した．一方で Java 以外にも自動生成ファイルは存在するため，実際には Java 以外の自動生成ファイルに対しても高い精度を出す必要がある．そこで実験 3 では JavaScript と TypeScript を対象に実験を行った．すなわち，人が作成した JavaScript ファイルと TypeScript ファイルを変換して生成された JavaScript ファイルに対して適用できるか交差検証を行った．この結果を**表 6** に示す．

提案手法では適合率，再現率ともに 98%を超える精度になっていることが分かる．

5.6 実験 4

実験 4 では 5.4 節で述べた交差検証の問題を回避するためブートストラップ法により提案手法の精度の評価を行った．データセットの作成は実験 3 で対象とした JavaScript と TypeScript に対して，実験 2 と同様 2,000 回行った．精度の評価結果を**表 7** に示す．

提案手法では適合率，再現率ともに 98%を超える精度になっていることが分かる．

```
                . . .
Object visit(AndImpl node, Object data)
          throws Exception;

Object visit(BindVariableValueImpl node, Object data)
          throws Exception;

Object visit(ChildNodeImpl node, Object data)
          throws Exception;

Object visit(ColumnImpl node, Object data)
          throws Exception;
                . . .
```

(a) シグネチャが類似したメソッドが多数ある例

```
                . . .
case '¥b':
          sb.append("¥¥b");
          break;
case '¥f':
          sb.append("¥¥f");
          break;
case '¥n':
          sb.append("¥¥n");
          break;
case '¥r':
          sb.append("¥¥r");
          break;
                . . .
```

(b) case 文が多い例

図 4 自動生成ファイル誤判定された自動生成でないファイルの一部
Fig. 4 Two examples that our technique misjudged.

6. 考察

本章では，5 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考察を行う．

交差検証の結果において，誤判定されたファイルを調べた．誤判定されたファイルの一部を**図 4** に示す．その結果，自動生成ファイルと誤判定された自動生成でないファイルの特徴として，以下のことがあげられる．

- シグネチャが類似したメソッドが多数ある．
- case 文が多い．

これらは，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．また，下仲らの手法と比較して精度が向上した要因としては，小さいファイルに対しても本手法が適用できるためと考えられる．下仲らの手法ではファイルサイズが 10 kB 以下の小さいファイルに対して適用した際に誤検出してしまう傾向があったが，提案手法においてその傾向はみられなかった．したがって，データセットに存在する 10 kB 以下の小さいファイルに関しても正しく判定ができたため精度が向

上したと考えられる．なお下仲らの手法が小さいファイルで誤検出しやすい要因は要素数が少ないためとしている．一方，本手法で用いている N-gram 言語モデルでは直前の $N-1$ 個の字句の並びに基づいて局所的に解析するため，ファイルサイズが小さくても誤検出されなかったと考えられる．

また単一種類の自動生成ファイル群での実験結果と MIX 自動生成ファイル群での実験結果を比較すると MIX 自動生成ファイル群の適合率が低下していた．これは複数種類の自動生成ファイル群を混ぜたことによって，自動生成ファイルの種類ごとに存在する特有の特徴が目立たなくなったためと考えられる．

また実験 3 と実験 4 によって，Java だけでなく TypeScript でも精度高く検出できたので，特定の言語に依存した手法ではないと思われる．

7. 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，自動生成でないことを目視確認をしているが，この目視確認が間違っている可能性がある．そのため，本来自動生成ファイルであるものを自動生成でないファイルと見なしている可能性がある．

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 種類のパーサジェネレータで生成されたファイルと TypeScript から JavaScript に変換されたファイルである．そのため特定の種類や言語によらず自動生成ファイルを特定できると考えられるが，実際に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．

8. あとがき

本研究では，N-gram 言語モデル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，自動生成ファイルと自動生成でないファイルそれぞれのソースコードの字句の並びを抽出し，それらを学習データとして自動生成ファイルの言語モデルと自動生成でないファイルの言語モデルを構築した．

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．その結果，すべての場合で適合率，再現率ともに 99%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．また 4 種類の自動生成ファイルを混ぜた場合の評価実験も行った．その結果に関しても，適合率，再現率ともに 98%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．加えて別種類の言語の自動生成ファイルを特定できるかの評価実験を行った．その結果，適合率，再現率ともに 98%以上と，高い精度で TypeScript から生成された JavaScript を特定できていることを確認した．

今後は任意の自動生成ファイルにおいてより高精度で特定できる手法とするため，改善していく予定がある．

謝辞 本研究は，科学研究費補助金基盤研究（B）（課題番号：17H01725）の助成を得て行われた．

参考文献

[1] Bengio, Y. and Grandvalet, Y.: No Unbiased Estimator of the Variance of K-Fold Cross-Validation, *J. Mach. Learn. Res.*, Vol.5, pp.1089–1105 (2004) (online), available from ⟨http://dl.acm.org/citation.cfm?id=1005332.1044695⟩.

[2] Brown, P.F., de Souza, P.V., Mercer, R.L., Pietra, V.J.D. and Lai, J.C.: Class-based N-gram Models of Natural Language, *Comput. Linguist.*, Vol.18, No.4, pp.467–479 (1992) (online), available from ⟨http://dl.acm.org/citation.cfm?id=176313.176316⟩.

[3] Chakraborty, P.: Object-Oriented Compilers: A Review, *IUP Journal of Information Technology*, Vol.13, No.1, p.36 (2017).

[4] Efron, B.: Bootstrap methods: Another look at the jackknife, *Breakthroughs in Statistics*, pp.569–593, Springer (1992).

[5] GitHub, available from ⟨http://github.com/⟩.

[6] Heafield, K.: KenLM: Faster and Smaller Language Model Queries, *Proc. 6th Workshop on Statistical Machine Translation*, *WMT '11*, pp.187–197, Association for Computational Linguistics (2011) (online), available from ⟨http://dl.acm.org/citation.cfm?id=2132960.2132986⟩.

[7] Heafield, K., Pouzyrevsky, I., Clark, J.H. and Koehn, P.: Scalable Modified Kneser-Ney Language Model Estimation, *ACL*, No.2, pp.690–696, The Association for Computer Linguistics (2013).

[8] Hindle, A., Barr, E.T., Gabel, M., Su, Z. and Devanbu, P.: On the Naturalness of Software, Vol.59, No.5, pp.122–131, ACM (online), DOI: 10.1145/2902362 (2016).

[9] jsoup: Java HTML Parser, available from ⟨http://jsoup.org/⟩.

[10] 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 (2010).

[11] Koschke, R. and Weinig, M.: Generated Code in Studies on Clone Rates, *2018 IEEE 11th International Workshop on Software Clones* (*IWSC*), pp.16–22 (2018).

[12] 下仲健斗，鷲見創一，肥後芳樹，楠本真二：機械学習を用いた自動生成コードの特定，電子情報通信学会技術研究報告，Vol.115, No.419, pp.165–170 (2016).

[13] 石原知也，堀田圭佑，肥後芳樹，井垣 宏，楠本真二：大規模なソフトウェア群を対象とするメソッド単位でのコードクローン検出，情報処理学会論文誌，Vol.54, No.2, pp.835–844 (2013).

[14] 大田崇史，井垣 宏，堀田圭祐，肥後芳樹，楠本真二ほか：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，研究報告ソフトウェア工学（SE），Vol.2014, No.22, pp.1–6 (2014).

土居 真之

平成 30 年大阪大学基礎工学部情報科学科卒業．同年より同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程在学中．

肥後 芳樹 （正会員）

平成 14 年大阪大学基礎工学部情報科学科中退．平成 18 年同大学大学院博士後期課程修了．平成 19 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．平成 27 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．

有馬 諒

平成 29 年大阪大学基礎工学部情報科学科卒業．同年より同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程在学中．

下仲 健斗

平成 28 年大阪大学基礎工学部情報科学科卒業．平成 30 年同大学大学院情報科学研究科コンピュータサイエンス専攻博士前期課程修了．在学時ソースコード分析に関する研究に従事．

楠本 真二 （正会員）

昭和 63 年大阪大学基礎工学部情報工学科卒業．平成 3 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．平成 8 年同講師．平成 11 年同助教授．平成 14 年同大学大学院情報科学研究科助教授．平成 17 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，プロジェクト管理に関する研究に従事．IEICE，JSSST，IEEE，JFPUG，PM 学会，SEA 各会員．