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.

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† 　肥後　芳樹† 　有馬　　諒† 　下仲　健斗† 　楠本　真二†

† 大阪大学大学院情報科学研究科，吹田市

E-mail: †{m-doi,higo,r-arima,s-kento,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．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．しかし解析結果が目立たない場合や解析時間が増加する場合があるため，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．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．また，機械学習に基づいた既存の手法と比較した結果，精度が向上していることを確認した．

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

1. ま え が き

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．この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中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．具体的には，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 [2]．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．

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，及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．

提案手法の評価を行うために，4 種類の自動生成ファイル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ファイルを特定できることを確認した．

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

                    ...

       int ABSTRACT = 12;
       int ASSERT = 13;
       int BOOLEAN = 14;
       int BREAK = 15;
       int BYTE = 16;

                    ...

       if ((active0 & 0x1f00L) != 0L)
              return 16;
       if ((active0 & 0x6000L) != 0L)
              return 45;
       if ((active0 & 0xff00L) != 0L)
              return 74;
```

図 1　自動生成ファイルの例

2. 準　　備

本章では，自動生成ファイルの定義，収集及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，例として 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．

- 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，下仲らが作成したデータセットを用いた [3]．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 [4] 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 [5]．本研究においても同様の方法を用いた．また，jsoup [6] 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．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ものである．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 [7] 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．

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

表 1　自動生成プログラムの種類とファイル数

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

表 2　自動生成プログラムのファイル名生成規則

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

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

ルずつ存在するデータセットを作成した．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 4 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．

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 [8].

- 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 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　データセット作成時におけるノイズデータ含有率

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%

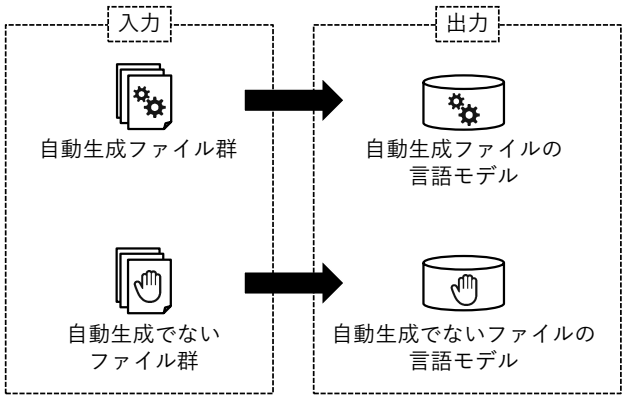
ファイル名検索による自動生成ファイル特定の結果を表 3 示す．ANTLR 生成ファイル群，JavaCC 生成ファイル群，SableCC 生成ファイル群においては 1,000 ファイル中約 800 ファイルの自動生成ファイルをそれぞれ特定できているのに対し，JFlex 生成ファイル群は 79 ファイルと極端に少なくなっていることが分かる．これは，JFlex のファイル名生成規則の数が少ないことが要因と考えられる．このことから，ファイル名生成規則だけでは自動生成ファイルを特定するのに十分ではないことが分かる．

2.3　機械学習を利用した自動生成ファイルの特定

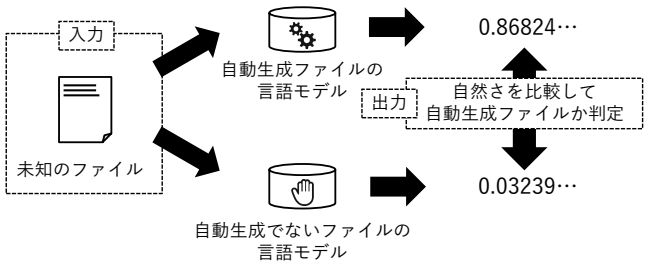
コメント文検索とファイル名による検索以外の自動生成ファイル特定手法として，機械学習を用いて特定する手法がある [3]. これは自動生成ファイルと自動生成でないファイルの構文情報の出現回数を学習データを用いてモデルを構築し，このモデルによって自動生成ファイルか否かを判定する手法である．この手法を 2.1 で述べたデータセットに対して 4 種類の機械学習アルゴリズム (Decision Tree, Naive Bayes, Random Forest, SVM) を用いた場合の適合率の上限と下限は 99.9%，82.7%，再現率の上限と下限は 99.9%，78.4%である．またサイズの小さいファイ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．したがって機械学習を利用した手法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．

2.4　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と機械学習を用いる手法があげられる．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．また機械学習を利用した手法ではサイズの小さいファイルに適用した場合，精度が悪くなるという問題が存在する．一方で，人



(a) Step 1



(b) Step 2

図 2　提案手法の概要

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 [9] 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．

2.5　N-gram 言語モデルと自然さ

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 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り，その値は次式で与えられる．

なお次式における W は入力となるテキストデータであり，W を単語に分解した場合の i 番目の単語が w_i である．また $c(w)$ はその単語列の出現回数である．

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

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

この式で求まる値を**自然さ**と呼ぶ [10]．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及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．

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

ソースファイルの字句解析には Eclipse JDT 4.5.2 [11] 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．字句解析及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 [12] 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 [13]．

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

表 5　交差検証の精度

自動生成プログラム		ANTLR		JavaCC		JFlex		SableCC	
		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%
下仲らの手法 [3]	Decision Tree	99.9%	99.9%	97.0%	97.0%	99.4%	99.4%	96.3%	96.2%
	Naive Bayes	98.8%	98.8%	88.0%	85.7%	99.5%	99.5%	82.7%	78.4%
	Random Forest	99.9%	99.9%	97.3%	97.3%	99.7%	99.7%	96.1%	96.1%
	SVM	99.8%	99.8%	95.7%	95.7%	99.6%	99.6%	86.8%	84.5%

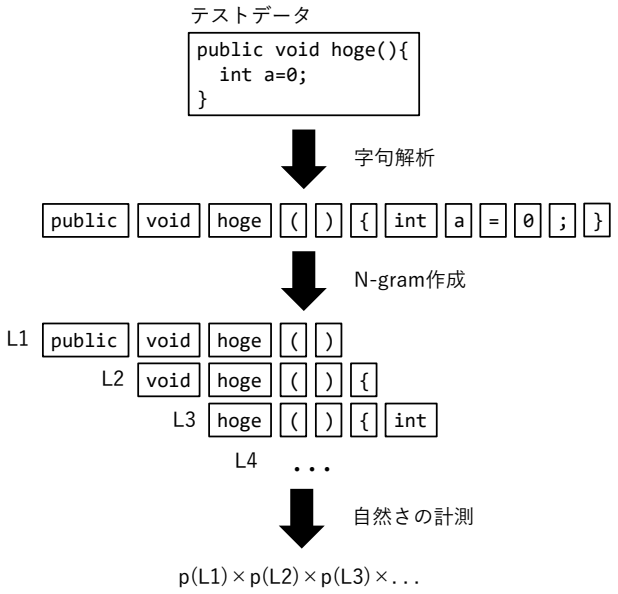


図 3　字句解析と自然さ計測の例

験結果について述べる．

提案手法の評価を行うために，2 つの実験を行った．その概要を以下に示す．

実験 1　ある単一種類の自動生成ファイル群で構築した言語モデルを用いて提案手法の精度を評価した．

実験 2　複数種類の自動生成ファイルを用いた自動生成ファイル群で構築した言語モデルを用いた場合の精度を評価した．

5.1 実験対象

学習データとして，2.1 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．

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

本実験の評価尺度は適合率と再現率の両方を算出した．

5.3 実験 1

まず収集した各自動生成ファイル群から言語モデルを構築する．同様に自動生成でないファイル群から言語モデルを構築し，計 5 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 5 種類の言語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，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 [3]．その結果を表 5 に示す．

提案手法では全ての場合において適合率, 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．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，単一種類の自動生成ファイルで構築した言語モデルの性能を評価した．実験 2 では，MIX ファイル群によって自動生成ファイルを特定できるかを確認するため交差検証を行った．また実験 1 同様，下仲らの手法との比較を行った．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．また機械学習を用いた手法による判定結果と比較すると，すべての場合において提案手法の方が精度が高くなっていることが分かる．

6. 考　　察

本章では，5. 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．

6.1 実験 1 の考察

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

表 6　MIX ファイル群の交差検証結果

自動生成プログラム		MIX	
		適合率	再現率
提案手法		98.7%	99.7%
下仲らの手法 [3]	Decision Tree	95.3%	95.3%
	Naive Bayes	85.3%	80.6%
	Random Forest	96.8%	96.8%
	SVM	85.2%	79.1%

```
                . . .
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

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. 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，下仲らの手法ではサイズが小さいファイルに対して適用した際に誤検出されてしまう傾向があったが，提案手法においてはその傾向はあまりみられなかった．そのため，提案手法はファイルサイズが小さい場合においても有用であると考えられる．

6.2　実験 2 の考察

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ところ，実験 1 で誤判定されたファイルが多く含まれていた．したがって自動生成ファイルの特徴をもつ自動生成でないファイルは誤検出される傾向があると考えられる．また実験 1 と比較して適合率の値が低下した．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．

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み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．

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，全ての場合で適合率，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．今後は任意の自動生成ファイルにおいて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，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) の助成を得て行われた．

文　献

[1] 大田崇史，井垣宏，堀田圭祐，肥後芳樹，楠本真二他，“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.

[2] A.C. MacLean, L.J. Pratt, J.L. Krein, and C.D. Knutson, “Trends that affect temporal analysis using sourceforge data,” Proceedings of the 5th International Workshop on Public Data about Software Development (WoPDaSD’10), p.6, 2010.

[3] 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，第 115 巻，pp.165–170，2016.

[4] “GitHub”. https://github.com/

[5] 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.

[6] “jsoup: Java html parser”. http://jsoup.org/

[7] “Apache”. http://svn.apache.org/repos/asf/

[8] P. Chakraborty, “Object-oriented compilers: A review,” IUP Journal of Information Technology, vol.13, no.1, p.36, 2017.

[9] P.F. Brown, P.V. Desouza, R.L. Mercer, V.J.D. Pietra, and J.C. Lai, “Class-based n-gram models of natural language,” Computational linguistics, vol.18, no.4, pp.467–479, 1992.

[10] A. Hindle, E.T. Barr, Z. Su, M. Gabel, and P. Devanbu, “On the naturalness of software,” Software Engineering (ICSE), 2012 34th International Conference onIEEE, pp.837–847 2012.

[11] “Eclipse Java development tools (JDT)”. http://www.eclipse.org/jdt/

[12] K. Heafield, “Kenlm: Faster and smaller language model queries,” Proceedings of the Sixth Workshop on Statistical Machine TranslationAssociation for Computational Linguistics, pp.187–197 2011.

[13] K. Heafield, I. Pouzyrevsky, J.H. Clark, and P. Koehn, “Scalable modified kneser-ney language model estimation.,” ACL (2), pp.690–696, 2013.