特別研究報告

題目

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
ーパーサジェネレータ生成ファイルへの適用ー

指導教員
楠本　真二　教授

報告者
下仲　健斗

平成 28 年 2 月 16 日

大阪大学 基礎工学部 情報科学科

平成 27 年度 特別研究報告

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
―パーサジェネレータ生成ファイルへの適用―

下仲　健斗

内容梗概

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，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．また，自動生成ファイル特有のコメント文が消されていても自動生成ファイルを特定できることを確認した．

主な用語

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

目 次

図 目 次

表 目 次

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．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]．例えば，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．例えば，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．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．また，コメント文の除去やファイル名の変更が行われた自動生成ファイルについても，提案手法による特定が可能であることを確認した．

以降，2 章では準備として自動生成ファイルおよび機械学習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．

```
private static final int jjStopStringLiteralDfa_0(int pos, long active0)
{
   switch (pos)
   {
      case 0:
         if ((active0 & 0x40040L) != 0L)
            return 0;
         if ((active0 & 0x600L) != 0L)
         {
            jjmatchedKind = 13;
            return 6;
         }
         return -1;
      case 1:
         if ((active0 & 0x600L) != 0L)
         {
            jjmatchedKind = 13;
            jjmatchedPos = 1;
            return 6;
         }
         return -1;
  }
}
```

図 1: 自動生成コードの例 (一部抜粋)

2 準備

2.1 自動生成ファイル

本節では，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が，本研究では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．パーサジェネレータ生成ファイルにおけるソースコードの一部を図 1 に示す．パーサジェネレータ生成ファイルにおけるソースコードの特徴として，以下のようなものが挙げられる．

- 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

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．

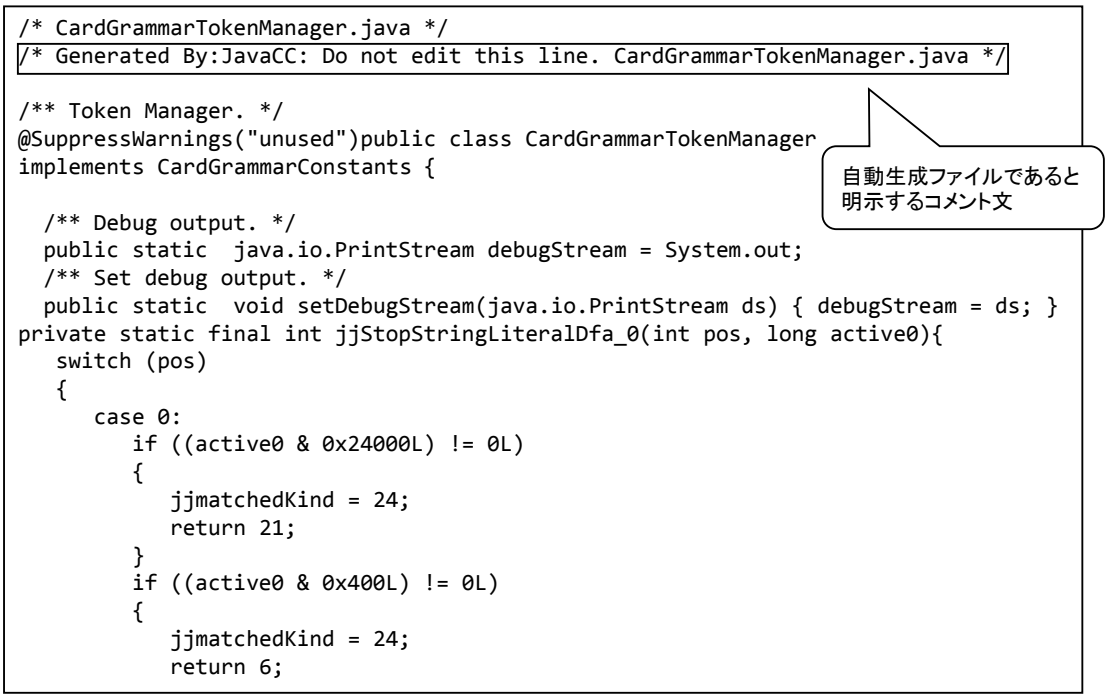

図 2: 自動生成ファイルのコメント文の例

しかし，ソースコード解析において，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 2 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，ソースファイルの変更や修正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．

2.1.1 自動生成ファイルと自動生成でないファイルの収集

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. そのため．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，特定の検索キーワードに対して 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 [1] 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．

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，全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．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

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

[1] ウェブサイトから情報を抽出する技術

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挙げられる．

2.1.2 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．例えば 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．2.1.1 項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 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 3 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．

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	Ylex.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

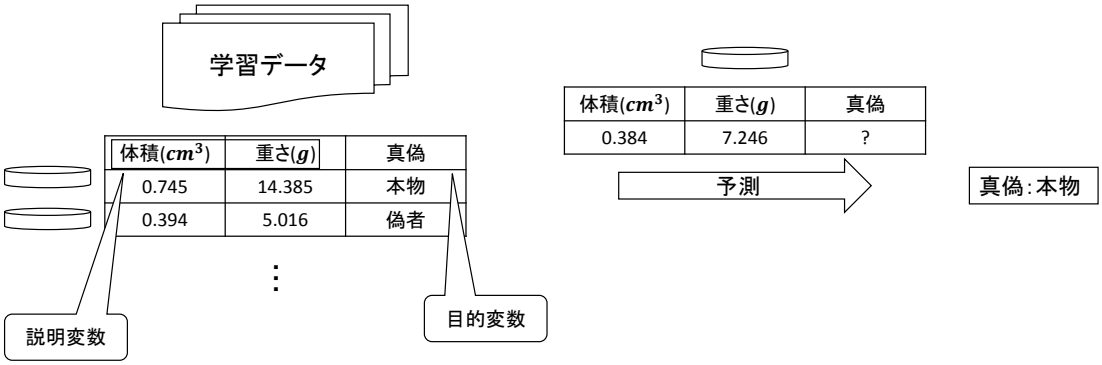

図 3: 教師あり機械学習の例

2.2 機械学習

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．機械学習では，既知のデータのことを**学習データ**と呼び，学習データの特徴から未知のデータであるテストデータの性質を予測するモデルを構築する．このモデルのことを**学習モデル**と呼ぶ．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．

機械学習は，教師あり機械学習と教師なし機械学習に大きく分類できる．教師あり機械学習では，学習データの説明変数と目的変数の値から，テストデータの目的変数の値を予測する．教師なし機械学習では，学習データの目的変数が定義されておらず，学習データに対して何らかの構造や法則を予測する．

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

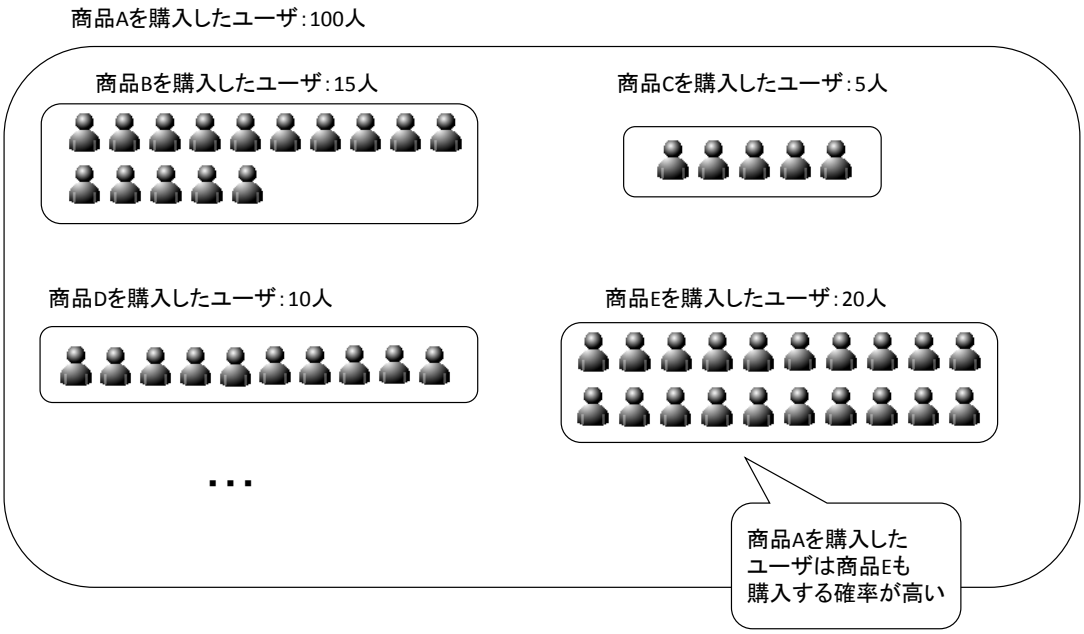

図 4: 教師なし機械学習の例

教師あり機械学習の例を図 3 に示す．この例では，金貨の体積と重さから，その金貨が本物か偽者かを予測する．体積と重さ，および本物か偽者かが既知であるいくつかの金貨を用いて，体積と重さだけが既知である金貨が本物か偽者かを予測する．つまりこの例では，説明変数は体積と重さであり，目的変数は本物か偽者かどうか，である．

次に教師なし機械学習の例を図 4 に示す．この例では，ある商品 A を購入したユーザ 100 人のデータから，一緒に購入する確率が高い商品リストを求める．商品 A を購入したユーザは，商品 E を購入する確率が高いと予測できる．

この例では，説明変数は購入した商品リストであり，目的変数は定義されていない．

本研究では，与えられたソースファイルの特徴を説明変数とし，そのソースファイルが自動生成ファイルかどうかを目的変数とするため，教師あり機械学習を用いる．

2.2.1 学習モデル構築アルゴリズム

教師あり機械学習における学習モデル構築アルゴリズムには，それぞれの場面に応じた適切なアルゴリズムが提案されている．本研究では，中でも代表的な Decision Tree[12]，Random Forest[13]，Naïve Bayes[14]，SVM (Support Vector Machine)[15] を用いる．各アルゴリズムの特徴を表 4 に示す．

2.2.2 変数選択

学習データの説明変数が多いと，学習データに偏った学習モデルが構築され，テストデータに対する汎化能力が不足する．これは過学習という状態で，精度が低下することで知られている [16]．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．変数選択には，Wrapper Subset Evaluation や Correlation-based Feature Selection，RelieF など様々なアルゴリズムが存在する．Hall らは 6 種類の変数選択アルゴリズムについて，どのアルゴリズムが優れているか比較を行っている [17]．変数選択アルゴリズムの比較を行うために，Hall らは，まずすべての変数選択アルゴリズムについてペアを作っている．そして，それらのペアについて同一のデータセットを用意し，それぞれの変数選択アルゴリズムをデータセットに適用する．これにより出された予測結果について，結果が良い方を Win，悪い方を Lose とし，各変数選択アルゴリズムの Win の総数と Lose の総数の差を求める．そして，この総数の差が高い値を示す変数選択アルゴリズムほど精度が良いとみなす．Hall らの調査の結果，良い結果を出す変数選択アルゴリズムは，説明変数の内容や使用する学習モデル構築アルゴリズムにより異なるということが明らかとなっている [17]．例えば，学習モデル構築アルゴリズムとして Naïve Bayes を用いた場合は Wrapper Subset Evaluation を使用したときに予測精度が最も良くなり，Decision Tree を用いた場合は RelieF を使用した時に予測精度が最も良くなるということが示されている．また，上述した Win の総数と Lose の総数の差でみると，Wrapper Subset Evaluation が最も高いことが判明している．この結果から，Hall らは，多くの場合において Wrapper Subset Evaluation は良い結果を出していると結論づけているため，本研究でも Wrapper Subset Evaluation を用いる．以下では，この Wrapper Subset Evaluation について説明を行う．

表 4: 学習モデル構築アルゴリズム

アルゴリズム	説明
Decision Tree	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，複数の条件分岐の木を生成する．それらを結合し，より精度の高い条件分岐の木を生成するアルゴリズム．Decision Tree に比べて計算量が多い．
Naïve 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 [18]．

Wrapper Subset Evaluation とは，説明変数の部分集合に対して，実際に学習モデルの構築と評価を行うことによって変数選択を行うアルゴリズムである．この手法では，実際に学習モデルの構築を行うために，どの学習モデルを使用するかを変数選択の前に与える必要がある．Wrapper Subset Evaluation は，学習モデルの構築と評価を行うことによって，説明変数の部分集合を評価するため，他の変数選択アルゴリズムよりも有用な説明変数を選択することが多いという長所を持つ一方で，計算量が多く，他のアルゴリズムと比べて実行に時間がかかるという欠点もある．

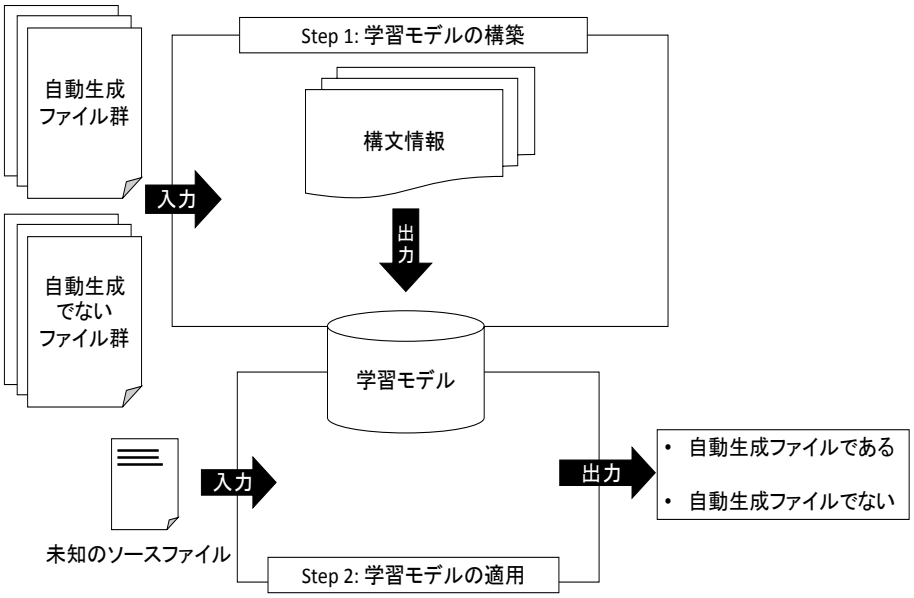

図 5: 提案手法の概要

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 5 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．

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，ソースファイルの 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ファイルであるかそうでないか，である．構文情報を取得した後，学習データに対して変数選択を行い，4 つのアルゴリズムを用いて 4 種類の学習モデルを構築する．

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 と同様に取得し，変数選択を行う．選択する説明変数は，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，自動生成ファイルか否かを判定する．

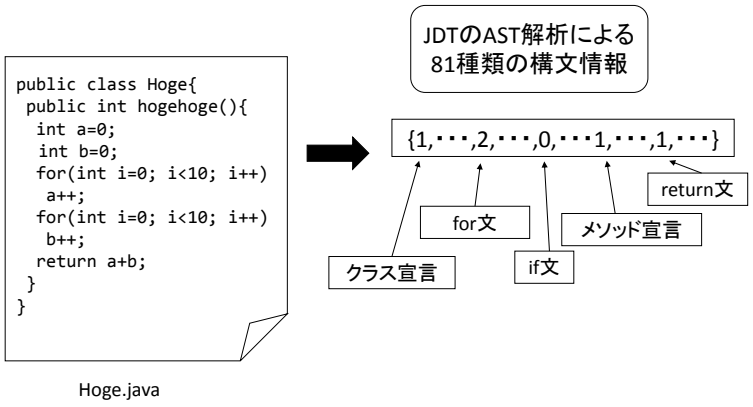

図 6: AST 解析の例

4 評価実験

本章では，提案手法を評価するために行った 5 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，5 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行う．

実験 2 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う．

実験 3 ある自動生成ファイル群から構築した学習モデルを用いて，別種類の自動生成ファイル群を特定できるか確認する．

実験 4 収集した自動生成ファイル群から構築した学習モデルを未知のソースファイル群に対して適用し，自動生成ファイルが特定できるかどうか確認する．

実験 5 ファイル名検索による自動生成ファイル特定手法と，提案手法を組み合わせて，自動生成ファイル特定の予測精度が向上するかどうか確認する．

4.1.1 実験対象

学習データとして，2.1.1 項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 4 における未知のソースファイル群には，大規模なソースファイルの集合である *UCI Source Code Data Sets*[19](以降，***UCI datasets*** と呼ぶ) を用いた．

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た [20]．具体的には，Eclipse JDT 3.10[21] を用いて AST 解析を行った．AST 解析の例を図 6 に示す．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．

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[22] を使用した．

4.2 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，実験 2 および実験 5 の評価尺度は *Precision* と *Recall* の両方を算出した．実験 3 では，自動生成ファイル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．

4.3 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．

4.4 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予測精度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，ソースファイルの行数は 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，各自動生成ファイルのファイル数を表 6 に示す．ANTLR 生成ファイル群に関してはすべてのファイルが 10KB 以上である．JFlex 生成ファイル群に関しては，ほとんどのファイルが 10KB 以上である．しかし，JavaCC 生成ファイル群と SableCC 生成ファイル群は 10KB 以上のファイル数が比較的低い割合になっている．そこで，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のファイルサイズを 10KB 以上に限定した新たなデータセットを作成した．ファイル数は今までと同様に 1,000 ファイルずつである．また，2.1.1 項で述べた処理を同様に行い，ノイズデータを除去した．新たに作成したデータセットを用いて交差検証を行った結果を表 7 に示す. なお，表中の下線は，表 5 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 生成ファイルと SableCC 生成ファイルが，ANTLR 生成ファイルと JFlex 生成ファイルと同様に，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 Naïve 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．

表 5: 交差検証の精度

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%

表 6: ファイルサイズが 10KB 以上の自動生成ファイル

自動生成プログラム	ANTLR	JavaCC	JFlex	SableCC
ファイル数	1,000	288	996	39

4.5 実験 3

実験 1 および実験 2 では同一種類の自動生成ファイル群に対して交差検証を行っているが，実験 3 では，ある自動生成ファイル群を用いて構築した学習モデルで，別種類の自動生成ファイル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ファイル群のみであり，自動生成でないファイル群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 以上のソースファイルに限定している．

ある自動生成ファイル群から構築した学習モデルを別種類の自動生成ファイルへ適用した結果を表 8 に示す．

表 7: 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		MIX	
	P	R	P	R	P	R	P	R	P	R
Decision Tree	99.9%	99.9%	99.3%	99.3%	99.5%	99.5%	97.8%	97.8%	97.9%	97.9%
Naïve Bayes	98.8%	98.8%	93.3%	92.4%	99.7%	99.7%	92.7%	91.8%	89.1%	87.6%
Random Forest	99.9%	99.9%	99.5%	99.5%	99.8%	99.8%	99.4%	99.3%	99.0%	99.0%
SVM	99.8%	99.8%	96.3%	96.3%	99.6%	99.6%	95.0%	95.0%	94.1%	94.0%

表 8: 別種類の自動生成ファイル間において学習モデルを適用した場合の *Recall*

自動生成プログラム	アルゴリズム	ANTLR	JavaCC	JFlex	SableCC
ANTLR	Decision Tree	-	27.4%	8.5%	8.7%
	Naïve Bayes	-	49.0%	69.3%	23.8%
	Random Forest	-	33.5%	25.4%	10.1%
	SVM	-	32.3%	18.5%	7.9%
JavaCC	Decision Tree	98.0%	-	87.1%	0.4%
	Naïve Bayes	98.8%	-	99.6%	26.4%
	Random Forest	91.2%	-	92.8%	21.7%
	SVM	95.3%	-	99.2%	22.8%
JFlex	Decision Tree	58.5%	48.6%	-	3.6%
	Naïve Bayes	52.6%	32.0%	-	6.2%
	Random Forest	73.0%	36.1%	-	0%
	SVM	97.3%	47.1%	-	0.2%
SableCC	Decision Tree	10.7%	0.4%	1.0%	-
	Naïve Bayes	36.0%	6.0%	6.1%	-
	Random Forest	1.4%	0.1%	3.3%	-
	SVM	0.7%	7.7%	16.1%	-

表中の数値は *Recall* を表している．表 8 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 生成ファイル，JavaCC 生成ファイル，JFlex 生成ファイルの 3 種類の自動生成ファイル間における精度は 50〜90%であるのに対し，SableCC 生成ファイルにおける精度は 0〜30%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ファイルの特徴と，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ファイルの特徴はあまり似通っていないと考えられる．

4.6 実験 4

実験 1 から実験 3 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 4 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* 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，MIX ファイル群を用いて構築した学習モデルと JFlex ファイル群を用いて構築した学習モデルをそれぞれ適用した．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 9 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，MIX ファイル群を用いて学習モデルを構築した場合，約 66%の精度で自動生成ファイルが特定可能であり，JFlex ファイル群を用いて学習モデルを構築した場合，約 74%の精度で特定可能であることが分かる．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．コメント文が消された自動生成ファイルのソースコードの一部を図 7 に示す．2.1 節で述べたようなコメント文がソースコード中には存在しない．したがって，コメント文検索では特定できない自動生成ファイルを，提案手法では特定可能であると言える．

表 9: *UCI datasets* に学習モデルを適用した結果

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
	JFlex	66
目視確認により自動生成ファイルと判断された数	MIX	552
	JFlex	49
Precision	MIX	65.9%
	JFlex	74.2%

```
/*
 * Licensed to the Apache Software Foundation (ASF) under one or more
 * contributor license agreements.  See the NOTICE file distributed with
 * this work for additional information regarding copyright ownership.
 * The ASF licenses this file to You under the Apache License, Version 2.0
 * (the "License"); you may not use this file except in compliance with
 * the License.  You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
package org.apache.cocoon.components.xpointer.parser;

static final long[] jjbitVec0 =
        { 0x0L, 0xfffffffffffc000L, 0xfffff0007fffffffL, 0x7fffffL };
    static final long[] jjbitVec2 = { 0x0L, 0x0L, 0x0L, 0xff7fffffff7fffffL };
    static final long[] jjbitVec3 =
        {
            0x7ff3fffffffffffL,
            0x7fffffffffffdfeL,
            0xffffffffffffffffL,
            0xfc31fffffffe00fL };
```

図 7: コメント文が消された自動生成ファイルの例

4.7 実験 5

実験 5 では，実験 2 における交差検証を行う際，提案手法と 2.1.2 項で述べたファイル名検索による手法を組み合わせた．組み合わせの方法として 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 10 と表 11 示す．

表 10 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 2 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 11 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 2 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，*Precision* を 100%に近づけたい場合はパターン 2 を用いれば良いと考えられる．

表 10: パターン 1 の結果

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 11: パターン 2 の結果

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%

5 考察

本章では，4 章で述べた評価実験についての考察を行う．

5.1 実験 1 と実験 2 の比較

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，学習モデルの予測精度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，以下のことが挙げられる．

- 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，switch-case 文やリテラルは，2.1 節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．

5.2 実験 3 の考察

次に，実験 3 の結果について考察する．ある自動生成ファイル群を用いて構築した学習モデルを別種類の自動生成ファイル群に適用した場合，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，別種類の自動生成ファイル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．

5.3 実験 4 の考察

実験 4 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 4 の結果は，実験 1 および実験 2 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 4 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 [23]．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．

5.4 実験 5 の考察

最後に，実験 5 の結果について考察する．2 章で述べた，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 12 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言える．

表 12: ファイル名検索による自動生成ファイル特定の結果における *Precision* と *Recall*

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%

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 4 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 あとがき

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．しかし，別種類の自動生成ファイル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ファイル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．

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．また，実際にコメント文が消された自動生成ファイルを特定することができた．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

本研究を行うにあたり，理解あるご指導を賜り，暖かく励まして頂きました楠本真二教授に心より感謝申し上げます．

本研究の全過程を通し，終始熱心かつ丁寧なご指導を頂きました，肥後芳樹准教授に深く感謝申し上げます．

本研究に関して，有益かつ的確なご助言を頂きました，柗本真佑助教授に深く感謝申し上げます．

本研究を進めるにあたり，適切なご助言および多大なるご助力を頂きました大阪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 1 年の鷲見創一氏に心より感謝申し上げます．

本研究を進めるにあたり，様々な形で励まし，ご助言を頂きましたその他の楠本研究室の皆様のご協力に心より感謝致します．

最後に，本研究に至るまでに，講義，演習，実験等でお世話になりました大阪大学基礎工学部情報科学科の諸先生方に，この場を借りて心から御礼申し上げます．

参考文献

[1] Pam McDonald, Dan Strickland, and Charles Wildman. Estimating the effective size of autogenerated code in a large software project. In *Proceedings of the 17th International Forum on COCOMO and Software Cost Modeling*, 2002.

[2] Shinji Uchida, Akito Monden, Naoki Ohsugi, Toshihiro Kamiya, Ken-Ichi Matsumoto, and Hideo Kudo. 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] Nils Göde and Rainer Koschke. Frequency and risks of changes to clones. In *Proceedings of the 33rd International Conference on Software Engineering*, pp. 311–320. ACM, 2011.

[5] Jan Harder and Nils Göde. Cloned code: stable code. *Journal of Software: Evolution and Process*, Vol. 25, No. 10, pp. 1063–1088, 2013.

[6] Alexander C MacLean, Landon J Pratt, Jonathan L Krein, and Charles 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. Citeseer, 2010.

[7] GitHub. 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. http://jsoup.org/.

[10] Apache Source code repository. http://svn.apache.org/repos/asf/.

[11] Etienne M Gagnon and Laurie J Hendren. Sablecc, an object-oriented compiler framework. In *Technology of Object-Oriented Languages, 1998. TOOLS 26. Proceedings*, pp. 140–154. IEEE, 1998.

[12] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. *Classification and regression trees*. CRC press, 1984.

[13] Leo Breiman. Random forests. *Machine learning*, Vol. 45, No. 1, pp. 5–32, 2001.

[14] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine learning*, Vol. 29, No. 2-3, pp. 103–130, 1997.

[15] Vladimir Vapnik. 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] Mark A Hall and Geoffrey Holmes. Benchmarking attribute selection techniques for discrete class data mining. *Knowledge and Data Engineering, IEEE Transactions on*, Vol. 15, No. 6, pp. 1437–1447, 2003.

[18] Christopher JC Burges. A tutorial on support vector machines for pattern recognition. *Data mining and knowledge discovery*, Vol. 2, No. 2, pp. 121–167, 1998.

[19] Uci Source Code Data Sets. http://www.ics.uci.edu/~lopse/datasets/.

[20] Hiroaki Murakami, Keisuke Hotta, Yoshiki Higo, and Shinji Kusumoto. Predicting next changes at the fine-grained level. In *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, Vol. 1, pp. 119–126. IEEE, 2014.

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

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

[23] Nathalie Japkowicz, et al. Learning from imbalanced data sets: a comparison of various strategies. In *AAAI workshop on learning from imbalanced data sets*, Vol. 68, pp. 10–15. Menlo Park, CA, 2000.