Vol.2021-SE-207 No.18
2021/3/1

メソッド抽出リファクタリング推薦手法に対するメソッド名予測を用いた精度改善の試み

山中 仁斗[1,a)]　早瀬 康裕[1,b)]　天笠 俊之[1,c)]

概要：ソフトウェア開発においてリファクタリングは欠かせない活動であり，その中でもメソッド抽出リファクタリングは特に使用頻度が高いことが知られている．メソッド抽出リファクタリングの支援を目的としてリファクタリングを行うべきコードを推薦する複数の先行研究では，抽出元メソッドと抽出されるコード片からそれぞれメトリクスを得ることで，推薦候補を順位付けしている．本研究では，先行研究が用いたメトリクスに追加して，抽出されるコード片が明確なメソッド名を付けられるような意味的なまとまりであるかどうかを考慮することで，メソッド抽出リファクタリングの推薦精度向上を図る．明確なメソッド名の付けやすさの基準としては，メソッド名予測手法 code2seq が出力する，コード片に対する予測メソッド名の確信度を用いる．メソッド抽出リファクタリング推薦手法 GEMS に対して予測メソッド名の確信度を加える変更を行なったところ，推薦精度が改善されることを確認できた．

Extract Method refactoring recommendation using method name prediction

Jinto Yamanaka[1,a)]　Yasuhiro Hayase[1,b)]　Toshiyuki Amagasa[1,c)]

Abstract: Refactoring is an important activity in software development, and Extract Method refactoring is known to be one of the most commonly applied refactoring. In some previous studies about recommendation code to be refactored for the purpose of supporting Extract Method refactoring, the recommended candidates are ranked by obtaining metrics from the original method and the code fragment to be extracted, respectively. In this study, we try to improve the recommendation accuracy of Extract Method refactoring by considering not only the metrics used in the previous studies, but also whether the code fragments to be extracted have semantic clusters that can be given clear method names. As a criterion for the ease of giving a clear method name, we use the confidence of the predicted method name for the code fragment output by the method name prediction approach, "code2seq". We made a change to the Extract Method refactoring recommendation method "GEMS" by adding the confidence of the predicted method name, and confirmed that the recommendation accuracy was improved.

1. はじめに

ソフトウェアの開発において，リファクタリングは欠かせない活動である．リファクタリングとは，プログラムの外部的な振る舞いを変更することなくソースコードの設計を改善する活動であり，ソフトウェアの可読性や理解性を向上させる目的で行われる [1], [2].

リファクタリングの中でも，既存のメソッドの一部のコードを新たなメソッドとして抽出する，メソッド抽出というリファクタリングが，使用場面の多い手法であるとされている．Murphy ら [3] は，統合開発環境 Eclipse のリファクタリング機能によって実行される主要な 11 種類のリファクタリングのうち，メソッド抽出リファクタリングが，調査対象となった開発者の 50%以上に実行されていることを明らかにした．同様に，Eclipse のリファクタリン

[1] 筑波大学
University of Tsukuba
[a)] yamanaka@kde.cs.tsukuba.ac.jp
[b)] hayase@cs.tsukuba.ac.jp
[c)] amagasa@cs.tsukuba.ac.jp

グプラグインである JDeodorant の使用統計[*1]においても，実行されたリファクタリング全体の中で約 50%という最大の割合をメソッド抽出リファクタリングが占めている．また，Chatzigeorgiou ら [4] は，オープンソースソフトウェアのソースコードの中に存在する，Long Method，Feature Envy，State Checking という 3 種類の Bad Smell の数を比較した．Bad Smell とは，リファクタリングを必要とする何らかの兆候を，匂いで比喩して表したものである [1]．3 種類の Bad Smell の数を比較した結果，メソッド抽出リファクタリングが解決策とされる Long Method の数が最も多く，メソッド抽出リファクタリングの使用機会の多さが示された．

リファクタリングすべきコードを手作業で特定することが困難になるような大規模なソフトウェアの開発を支援するために，使用場面が多いとされるメソッド抽出リファクタリングを行うべきコードを自動で特定し，開発者へ推薦する手法が，いくつか提案されてきた [5], [6], [7], [8], [9]．既存手法では，抽出元のメソッドや抽出候補のコード片から得られる様々なメトリクスを利用することで，リファクタリングすべき箇所の推薦が行われている．

メソッド抽出リファクタリングは，抽出する箇所を決定した後，新たに作成するメソッドに，その役割を理解できるような明確な名前を付けることで完成する．そのため，抽出するコード片がメソッド名を付けやすい内容であるかどうかは，推薦を行う際に意味を持つ情報であると考えられるが，コード片に対するメソッド名の付けやすさを考慮した推薦手法は我々の知る限り存在していない．

本研究では，既存手法で使用されているメトリクスに加え，抽出候補のコード片が明確なメソッド名を付けやすいような意味的なまとまりであるかどうかを考慮することで，既存手法を拡張し，推薦精度の改善を試る．拡張対象として，既存手法の中で最も良い推薦精度を示している GEMS[9] を用いる．あるメソッドの抽出候補となっているコード片に対して，メソッド名予測手法の code2seq[10] によって抽出後のメソッド名を予測させ，予測時にモデルが算出する，メソッド名を正しく予測できている確率を，抽出候補に対するメソッド名の付けやすさ表す指標として利用する．評価実験を行い，メソッド名の付けやすさを新たなメトリクスとして加えることが，推薦精度の向上に貢献することを示した．

本稿の構成は以下の通りである．まず，2 章で本研究における前提知識として，メソッド抽出リファクタリングと code2seq，および既存の推薦手法について説明する．3 章では提案手法について説明する．4 章で評価実験について説明する．最後に 5 章で本研究についてまとめる．

[*1] https://users.encs.concordia.ca/∼nikolaos/stats.html

```
1  void printOwing(){
2    printBanner();
3
4    //print details
5    System.out.println("name: " + _name);
6    System.out.println("amount: " + getOutstanding());
7  }
```

```
1  void printOwing(){
2    printBanner();
3    printDetails(getOutstanding());
4  }
5
6  void printDetails(double outstanding){
7    System.out.println("name: " + _name);
8    System.out.println("amount: " + getOutstanding());
9  }
```

図 1　メソッド抽出リファクタリングの例
Fig. 1　Example of Extract Method refactoring

2. 前提知識

2.1 メソッド抽出リファクタリング

メソッド抽出リファクタリングとは，リファクタリングの種類の 1 つであり，既存のメソッドの一部のコードを新たなメソッドとして抽出することでメソッドの分割を行う手法である [1]．長すぎるメソッドや複数の機能が実装されたメソッドを，機能ごとに分割することで，プログラムの理解度を向上させる目的で行われる．本リファクタリングは，大きく以下の手順に分けることができる．

(1) 既存のメソッドから抽出するコードを決定する．
(2) 新たなメソッドを作成し，コードの役割が分かるような明確な名前を付ける．
(3) 抽出元メソッドにおいて，抽出したコード片を新たなメソッドの呼び出し文に置き換える．

図 1 にメソッド抽出リファクタリングの例 [1] を示す．図 1 では，printOwing メソッドにおける 5 行目と 6 行目を抽出することで，printDetails メソッドを新しく作成している．元のメソッドである printOwing メソッドには，抽出したコード片の代わりに printDetails メソッドを呼び出すステートメントが追加されている．

2.2 関連研究

2.2.1 code2seq

Alon らが提案した code2seq[10] は，メソッドボディのソースコードから作られる抽象構文木を用いて，メソッドの分散表現を生成し，この分散表現を利用してメソッド名を予測する手法である．code2seq は，Alon らが提案した code2vec[11] を拡張した手法となっている．この手法のモデルは入力として名前を予測させたいメソッドのソースコードを受け取り，予測したメソッド名の候補を，正しく

予測できている確率の高いものから順に返す．本研究では，この正しく予測できている確率を「確信度」と呼ぶこととする．予測されたメソッド名は，単語のシーケンスとして出力される．code2seq は，我々の知る限り，現在最もメソッド名の予測精度が高い手法となっている．

2.2.2 既存の推薦手法

メソッド抽出リファクタリングの対象を自動で特定し，開発者へ推薦する手法が様々提案されている．

Tsantalis ら [5] が提案した JDeodorant は，複数種類のリファクタリング支援に対応した手法となっている．メソッド抽出リファクタリングにおいては，プログラム内の任意の変数に影響を与えるソースコードのみを元のプログラムから抽出する技術である，プログラムスライシングの考え方を用いて，リファクタリングの対象を判別している．彼らの手法では，メソッドのボディ内で与えられた，ある変数の値やあるオブジェクトの状態を変更する全てのステートメントを含んだソースコードを，抽出対象として推薦する．

Silva ら [6], [7] は，メソッド内で使用されている変数，型，パッケージの集合から集合類似度係数を求めることで抽出対象の判別を行う，JExtract を提案した．彼らの手法では，まず，抽出候補として，メソッドから抽出可能なコード片を全て生成する．メソッドのソースコードをブロックと呼ばれる入れ子構造ごとの単位に分割し，同一ブロック内で連続しているステートメントの組み合わせを全て取得した後，実際に抽出した際にコンパイルエラーが発生する組み合わせを排除し，残ったものを全て抽出可能なコード片として扱う．次に，抽出元のメソッドと抽出候補の各コード片からそれぞれ変数，型，パッケージの要素集合を作成する．そして，抽出元のメソッドと各抽出候補から作成した要素集合を用いて，Kulczynski 係数と呼ばれる集合類似度係数を求め，抽出元のメソッドとの類似度が小さい候補は機能が独立していると考え，抽出対象として推薦する．

Sofia ら [8] が提案した SEMI という手法では，クラスの凝集度の欠如値を測る $LCOM_2$ というメトリクスをメソッドレベルで利用することで，抽出対象の判別を行っている．凝集度とはソースコードの機能がどれほど集約されているかを表す指標であり，凝集度が高いほど，機能が独立しておりソースコードが理解しやすいとされる．SEMI では，著者らが提案したアルゴリズムに沿って抽出対象の候補を作成した後，抽出元のメソッドと各抽出候補から求められる $LCOM_2$ を用いて，Benefit と呼ばれる値を計算し，Benefit の大きさによって抽出対象の判別，推薦を行う．

Xu ら [9] は GEMS という手法を提案した．彼らは，既存手法で使用されているメトリクスでは特定のプログラム要素のみしか考慮されていないという課題や，実際のリファクタリングは特定のメトリクスの改善のみを目的にする訳ではないため，特定のメトリクスに依存するアプローチは実用性に欠けるという課題を挙げ，メソッド抽出リファクタリングの対象を特定するために 48 種類のメトリクスを特徴量とした機械学習モデルを作成した．GEMS では，JExtract と同様の抽出候補生成アルゴリズムに従って抽出候補のコード片を作成した後，各抽出候補から特徴量を作成する．そして，上述の機械学習モデルに基づいて各候補が抽出されるべきであるかどうかを分類し，抽出対象と分類されたものを予測確率の高いものから順に推薦を行う．GEMS は，我々の知る限り既存手法の中で最も良い推薦結果を示している．

3. 提案手法

本研究では，既存のメソッド抽出リファクタリングの推薦手法が用いたメトリクスに追加して，あるメソッドの抽出候補となったコード片が，明確なメソッド名を付けやすいような意味的なまとまりであるかどうかを考慮することで，既存手法を拡張し推薦精度の改善を試る．メソッド抽出リファクタリングは，抽出する箇所を決定した後，新たに作成するメソッドに，その役割を理解できるような明確な名前を付けることで完成する．そのため，抽出するコード片がメソッド名を付けやすい内容であるかどうかという要素は，推薦に役立つと考えられる．

3.1 提案手法の概要

提案手法では，メソッド名予測手法の code2seq を用いて，既存手法の中で最も良い推薦精度を示している GEMS を拡張し，推薦精度の改善を目指す．抽出候補のコード片に明確なメソッド名を付けやすいかどうかという指標に，code2seq のモデルがメソッド名予測時に算出する確信度を採用する．この値を GEMS の分類モデルが用いていた特徴量に加えることで，GEMS の拡張を行う．GEMS と code2seq は，それぞれオープンソースソフトウェアとして GitHub で公開されており [13], [14]，本研究ではこれらのソースコードを用いる．提案手法の全体像を図 2 に示す．本手法は大きく，抽出候補の分類モデルを事前に構築するための学習段階と，モデルを用いて抽出対象を推薦する推薦段階に分けられる．

3.1.1 学習段階

学習段階では，メソッド抽出リファクタリングの実例データから特徴量を作成し，機械学習によって，推薦段階に用いる分類モデルを事前に構築する．メソッド抽出リファクタリングの実例データとは，オープンソースソフトウェアにおいて本リファクタリングが実際に行われたメソッドと，抽出されたコード片のペアを指す．

まず，実例データの各抽出元メソッドに GEMS を適用することで，メソッドから抽出可能な全てのコード片を，抽出候補として生成する．次に，生成された抽出候補のう

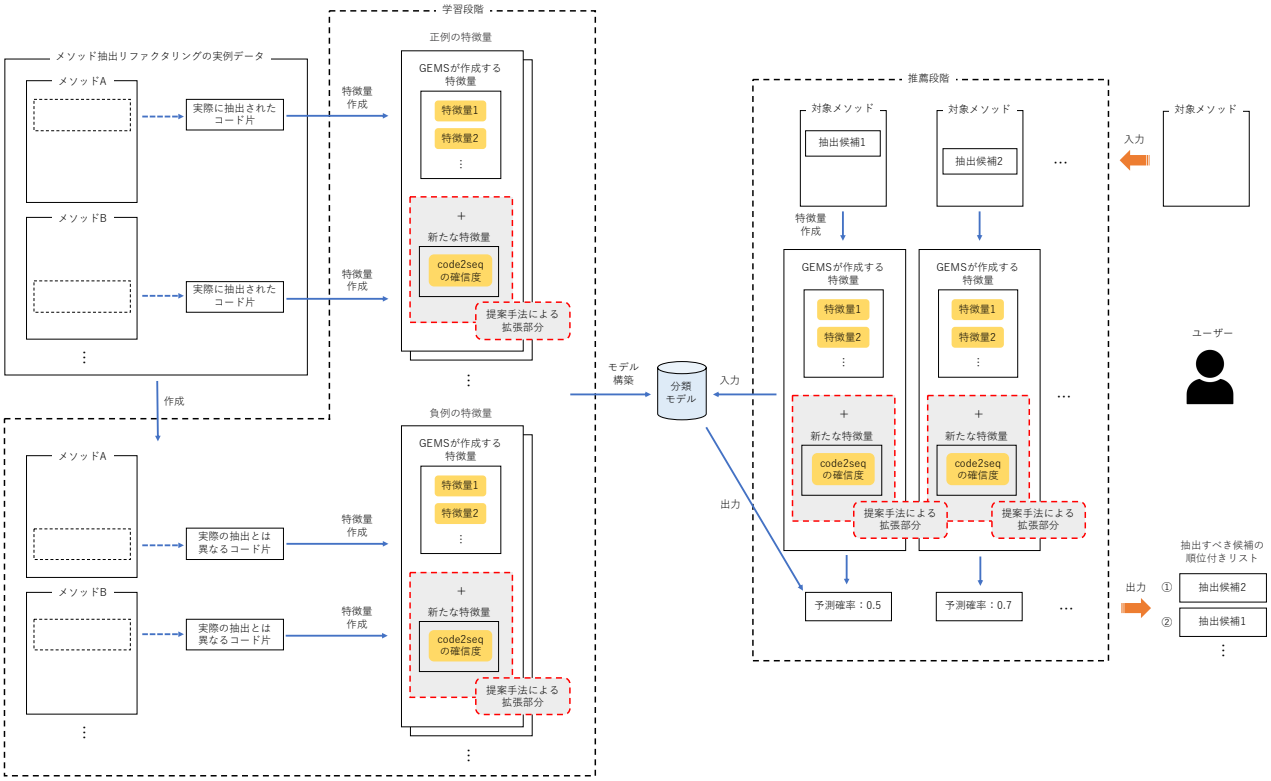

図 2　提案手法の全体像
Fig. 2　Overview of the proposed method

ち，実例データで実際に抽出された範囲とは異なるコード片を，ランダムに一つ選択する．

そして，実例データのメソッドと実際に抽出されたコード片のペア，同メソッドと実際の抽出とは異なるコード片のペアそれぞれから，GEMS と code2seq を用いて特徴量を作成し，機械学習による分類モデルの構築を行う．前者のペアから作られた特徴量を正例，後者のペアから作られた特徴量を負例として，学習データを作成する．GEMS の処理を継承することで各ペアに対して作成した特徴量に，code2seq を用いて新たな特徴量を加えることによって，GEMS を拡張する．

3.1.2　推薦段階

推薦段階では，ユーザーがメソッド抽出リファクタリングの必要性を調べたいメソッドを入力すると，抽出すべきコード片の順位付きリストが出力される．提案手法による拡張部分を除いて，推薦段階は GEMS の処理を継承している．まず，入力されたメソッドから抽出候補のコード片を生成し，そのメソッドと各抽出候補のペアから，GEMS で使用される特徴量を作成する．この特徴量に，学習段階と同様にして，code2seq を用いた新たな特徴量を加え，事前に構築した分類モデルへ入力するように拡張する．そして，モデルが抽出すべきであると分類した抽出候補を，その予測確率の高いものから順に出力する．

3.1.3　提案手法による拡張部分

本研究では，抽出候補のコード片に明確なメソッド名を付けやすいかどうかという指標に，code2seq のモデルがコード片のメソッド名予測時に算出する確信度を使用し，この値を GEMS が用いる特徴量に加えることで GEMS の拡張を行う．GEMS の拡張方法の概要を図 3 に示す．あるメソッドに対して GEMS を適用すると，まず抽出候補として，抽出可能なコード片が全て生成された後，抽出元のメソッドと各コード片のペアから 48 種類のメトリクスが特徴量として作成される．ここで，拡張を行うために，各コード片をそれぞれ，Eclipse JDT（Java Development Tools）のメソッド抽出機能によってメソッドの形に変換し，code2seq に入力する．この変換作業を行うのは，code2seq がメソッド形式のソースコードのみを受取可能とするためである．そして，それぞれのコード片に対して，code2seq が最上位に予測したメソッド名の確信度を取得し，GEMS が作成した特徴量に加える．

GEMS によって生成される特徴量の内容を表 1，表 2 に示す．特徴量は大きく構造的特徴量と機能的特徴量の 2 つのカテゴリに分けられており，構造的特徴量には 28 種類，機能的特徴量には 20 種類の特徴量が含まれる．表 1 に示した構造的特徴量は主に，抽出候補のコード片と，抽出元のメソッドから抽出候補を取り除いたコードのそれぞれから特徴量が作られる．表 1 の「抽出候補のコード片」の列

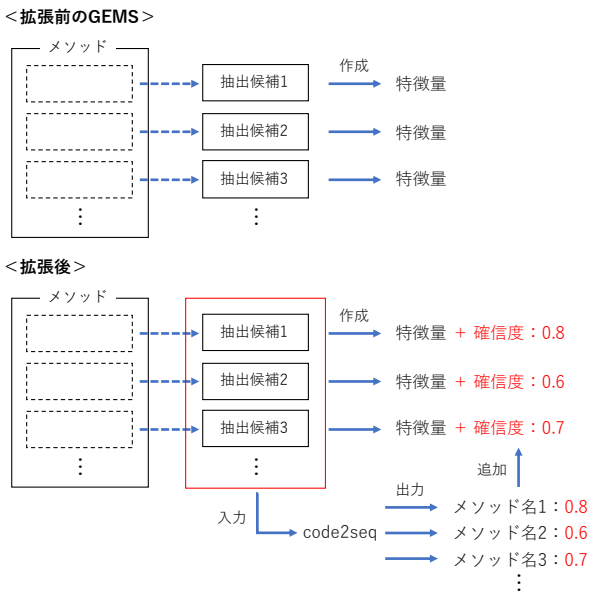

図 3　GEMS の拡張方法
Fig. 3　How to expand GEMS

には，抽出候補のコード片から作成される特徴量の名前を，「抽出候補を除いたコード」の列には，抽出元のメソッドから抽出候補を取り除いたコードから作成される特徴量の名前を記載した．また，表 2 に示した機能的特徴量は，次のようにして作られる．まず，プログラム要素ごとに，抽出元メソッドでの使用回数に対する，抽出候補のコード片での使用回数の割合が，1 位または 1〜2 位になる要素を特定する．この 1 位または 1〜2 位の割合が，表中の「抽出候補中での使用率」に該当する特徴量となる．次に，抽出候補のコード行数に対する，先述の特定された要素が使用されているコード行数の割合を求める．この割合が表中の「使用率の高い要素への専念度」に該当する特徴量となる．

4.　評価実験

特徴量に code2seq による確信度を加えることで，GEMS で用いられている特徴量のみを使用する場合よりも，推薦精度が向上するかどうかを評価するための実験を行った．GEMS で用いられている特徴量のみを使用した予測モデルと，GEMS で用いられている特徴量に確信度を加えた予測モデルを作成し，評価データに対してそれぞれのモデルを用いた推薦結果を比較することで評価を行う．

4.1　予測モデルの構築

学習データを作成し，機械学習を用いて，GEMS で用いられている特徴量のみを使用したモデルと，これらの特徴量に確信度を加えて学習したモデルの 2 つを構築する．

学習データの作成には，Silva ら [12] によって，GitHub[*2] で公開されている Java プロジェクトから作成されたメソッド抽出リファクタリングの実例データを利用する．このデータは，本研究の提案手法で利用する GEMS の研究 [9] でも利用されており，実際にメソッド抽出リファクタリングが行われたメソッドと，そのメソッドから抽出された箇所をデータとして持っている．このデータを用いて，3.1.1 小節に示したようにして，実例データのメソッドと実際に抽出されたコード片のペアから作成した特徴量を正例，同メソッドと実際の抽出とは異なるコード片のペアから作成した特徴量を負例として，学習データを作成する．メソッドから作成される抽出候補が正例に使用される 1 つのみの場合が含まれているため，結果として 244 個のメソッドから，244 個の正例と 235 個の負例から成る，合計 479 個の学習データを作成した．GEMS で用いられている特徴量のみを使用した学習データは 48 次元，確信度を加えた学習データは 49 次元となる．

作成した 2 つの学習データから，機械学習を用いて，メソッド抽出リファクタリングの対象を予測する分類モデルを構築する．本研究では，GEMS のモデルと同様にして，Python の機械学習ライブラリ scikit-learn[15] に実装されている勾配ブースティング回帰木[*3] を機械学習アルゴリズムに用いる．

モデルを構築する際，ハイパーパラメータの値を決定するために，ハイパーパラメータ最適化フレームワークの Optuna[16] を用いてパラーメータチューニングを行った．正例と負例を合計した 479 個の学習データに対して 5 分割交差検証を行い，本研究で推薦精度として用いる F 値の 5 回の平均値ができるだけ高くなるようにハイパーパラメータを探索した．

4.2　評価データ

評価データには，SelfPlanner，WikiDev，JHotDraw，Junit，MyWebMarket という 5 つの Java プロジェクトから特定された，メソッド抽出リファクタリングを実行すべきメソッドとその抽出対象を利用する．これらのプロジェクトは品質の高いオープンソースソフトウェアとされており，5 つのプロジェクトを合計して 130 個のメソッドから 155 個のリファクタリング対象が特定されている．SelfPlanner と WikiDev のデータは Tsantalis らに [5] よって作成され，JDeodorant の性能評価に使用された．また，JHotDraw, Junit, MyWebMarket のデータは，Silva ら [6] によって作成され，JExtract の性能評価に使用された．Sofia ら [8] と Xu ら [9] もこれら 5 つのデータを用いて SEMI と GEMS を評価しており，現在，メソッド抽出リファクタリングの推薦手法を評価するのに最も適したデータセットとされている．本研究では，これらのメソッドから抽出候補および特徴量を作成することで，評価用のデータセットを構築する．

[*2] https://github.com

[*3] https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html

表 1　GEMS が生成する構造的特徴量
Table 1　Structural features generated by GEMS

特徴量の内容	抽出候補のコード片	抽出候補を除いたコード
コード行数	LOC_EXTRACTED_METHOD	CON_LOC
ローカル変数の定義数	NUM_LOCAL	CON_LOCAL
リテラルが定義されているか否か	NUM_LITERAL	CON_LITERAL
メソッド呼び出し数	NUM_INVOCATION	CON_INVOCATION
if 文の数	NUM_IF	CON_IF
三項演算子の数	NUM_CONDITIONAL	CON_CONDITIONAL
switch 文の数	NUM_SWITCH	CON_SWITCH
変数へのアクセス数	NUM_VAR_AC	CON_VAR_ACC
使用している型の数	NUM_TYPE_AC	CON_TYPE_ACC
フィールドへのアクセス数	NUM_FIELD_AC	CON_FIELD_ACC）
代入文の数	NUM_ASSIGN	CON_ASSIGN
型を持つプログラム要素の数	NUM_TYPED_ELE	CON_TYPED_ELE
参照するパッケージ数	NUM_PACKAGE	CON_PACKAGE
assert 文の数	CON_ASSERT	n/a
抽出元メソッドのコード行数に対する抽出候補のコード行数の割合	RATIO_LOC	

表 2　GEMS が生成する機能的特徴量
Table 2　Functional features generated by GEMS

プログラム要素	抽出候補中での使用率（1 位，2 位）	使用率の高い要素への専念度（1 位，2 位）
ローカル変数	RATIO_VARIABLE_ACCESS RATIO_VARIABLE_ACCESS2	VARAC_COHESION VARAC_COHESION2
フィールド	RATIO_FIELD_ACCESS RATIO_FIELD_ACCESS2	FIELD_COHESION FIELD_COHESION2
メソッド	RATIO_INVOCATION	INVOCATION_COHESION
型	RATIO_TYPE_ACCESS RATIO_TYPE_ACCESS2	TYPEAC_COHESION TYPEAC_COHESION2
型を持つプログラム要素	RATIO_TYPED_ELE	TYPEDELE_COHESION
パッケージ	RATIO_PACKAGE RATIO_PACKAGE2	PACKAGE_COHESION PACKAGE_COHESION2

4.3　評価指標

先行研究 [8], [9] と同様に，4.2 節の 130 個の各メソッドに対する上位 5 個までの推薦結果を使用して，適合率，再現率，F 値の 3 つの指標を用いて性能を評価する．ここで，推薦される抽出候補は全て，リファクタリングすべきであると分類されたものであるということに注意が必要である．適合率は，推薦された抽出候補の総数に対して，推薦結果が正しかった候補の割合を指す．再現率は，155 個のリファクタリング対象に対して，実際に推薦された対象の割合を指す．F 値は適合率と再現率の調和平均であり，手法の推薦精度として用いられる．

推薦対象の正誤を評価するとき，先行研究 [8], [9] に倣って，コード行のばらつきを許容する範囲の指標として許容度と許容行数という値を導入する．使用する許容度は 1%，2%，3%の 3 パターンであり，抽出元のメソッド行数に許容度を掛け，小数点以下を切り上げた値が許容行数となる．例えば 50 行のメソッドが与えられた場合に，許容度が 3%のとき，50*0.03=1.5 となるため，許容行数は 2 行となる．この場合，正しい推薦箇所と ± 2 行の誤差を持つ候補も正しい対象であると考える．

また，確信度を加えた場合と加えなかった場合それぞれのモデルについて，各特徴量が予測にどれほど貢献しているかを測るために，特徴量の重要度を調べる．特徴量の重要度には，scikit-learn の勾配ブースティング回帰木モデルが算出するジニ重要度を用いる．特徴量のジニ重要度が高いほど，その特徴量は重要であるとみなされる．

4.4　実験結果

4.4.1　評価指標の比較

評価指標を比較した結果を表 3 に示す．この表のうち，確信度なしと表記されているモデルは GEMS の特徴量のみから作成されたモデルを表しており，確信度ありと表記されているモデルは GEMS の特徴量に code2seq による確信度を加えたモデルを表している．130 個のメソッドに対する上位 5 個までの推薦結果の合計数は，確信度なしのモデルの場合は 571 個，確信度ありのモデルの場合は 564 個

表 3　評価指標の比較
Table 3　Comparison of evaluation metrics

モデル	許容度	適合率	再現率	F 値
確信度なし	なし	0.08757	0.32258	0.13774
	1%	0.21191	0.45806	0.28977
	2%	0.21366	0.46452	0.29269
	3%	0.22067	0.48387	0.30310
確信度あり	なし	0.09574	0.34839	0.15021
	1%	0.21986	0.52258	0.30950
	2%	0.22518	0.52903	0.31590
	3%	0.23227	0.54839	0.32632

となった．

表 3 より，確信度を使用しなかった場合と比較して，確信度を使用した場合の適合率，再現率，F 値は，どの許容度においても上昇していることが分かる．F 値は許容度がなしの場合は約 1.25 ポイント，1%の場合は約 1.97 ポイント，2〜3%の場合は約 2.32 ポイント上昇している．また，適合率に比べ再現率の上昇幅が大きく，許容度がなしの場合は約 2.58 ポイント，1〜3%の場合は約 6.45 ポイント上昇している．

ここで，本稿では先行研究に倣って学習データの正例と負例の数がほとんど等しくなるようにしてモデルの構築を行っているが，現実ではメソッドから得られる抽出候補の多くは，抽出すべきではないという結果に分類されると考えられるため，モデル学習時の正例と負例の比率は現実に即していないと考えられる．よって，適合率の数値の信頼性は再現率に比べて低いと考えられる点に注意が必要である．

4.4.2　特徴量の重要度の比較

それぞれのモデルにおける特徴量の重要度の上位 10 個を比較する．確信度を使用していないモデルの重要度を表 4 に，確信度を使用したモデルの重要度を表 5 に示す．表 5 において，CODE2SEQ_CONFIDENCE は提案手法で加えた code2seq による確信度を表す．

表 4 より，モデルに確信度を使用しない場合，最も重要度の高い特徴量は TYPEDELE_COHESION であり，スコアは 0.15175 となっている．また，2 番目に重要度の高い NUM_TYPED_ELE とのスコアの差は，約 2.67 ポイントである．一方，表 5 より，モデルに確信度を使用した場合，確信度が最も重要度の高い特徴量となっている．この重要度のスコアは，表 4 の最上位のスコアを約 14.84 ポイント上回る 0.30011 となっており，2 番目に重要度の高い TYPEDELE_COHESION とのスコアの差は約 14.43 ポイントである．

4.5　考察

4.4.1 小節より，全ての許容度において F 値が上昇していることから，確信度を加えることでメソッド抽出リファクタリングの推薦精度を改善できることが分かった．また，適合率に比べ再現率の上昇幅が大きかったことから，確信度は推薦において，抽出すべきリファクタリング対象の網羅率を高める効果が大きいと考えられる．さらに，4.4.2 小節の結果より，全ての特徴量の中でも確信度が予測において大きく貢献していると言える．

表 4　確信度を使用していないモデルの重要度
Table 4　Feature importance for models without the confidence

順位	特徴量	重要度
1	TYPEDELE_COHESION	0.15175
2	NUM_TYPED_ELE	0.12510
3	RATIO_LOC	0.10388
4	INVOCATION_COHESION	0.09432
5	NUM_PACKAGE	0.09407
6	CON_LOC	0.05041
7	NUM_TYPE_AC	0.04943
8	VARAC_COHESION	0.04165
9	CON_TYPED_ELE	0.03155
10	CON_PACKAGE	0.03067

表 5　確信度を使用したモデルの重要度
Table 5　Feature importance of model with the confidence

順位	特徴量	重要度
1	CODE2SEQ_CONFIDENCE	0.30011
2	TYPEDELE_COHESION	0.15584
3	INVOCATION_COHESION	0.08658
4	NUM_PACKAGE	0.07611
5	RATIO_LOC	0.05153
6	NUM_INVOCATION	0.04172
7	NUM_VAR_AC	0.03058
8	NUM_TYPE_AC	0.02505
9	RATIO_TYPE_ACCESS	0.02409
10	LOC_EXTRACTED_METHOD	0.02188

code2seq は，メソッドの構文構造から，ソースコードの意味的な情報を獲得することでメソッド名を予測するという性質を持っている．code2seq の確信度がメソッド抽出リファクタリングの対象を特定するのに優れていた理由としては，この性質によって，抽出候補のコード片が，リファクタリングに適した，機能を説明しやすいような意味的なまとまりを持っているかどうかを判断できていたためであると考えられる．リファクタリングの実例となったコード片とそれ以外のコード片の間で，確信度の大きさに明確な違いがあるかどうかを調べるために．4.1 節で実例データから作成した正例と負例それぞれの抽出コード片に対する，確信度の最大値，最小値，平均値，中央値を求め，比較を行う．求めた値の比較結果を表 6 に示す．表 6 より，負例の確信度に比べて，正例の確信度は最小値，平均値，中央値が上回っており，リファクタリングされるような意味的なまとまりを持ったコードであるほど，確信度は高くなる傾向があると考えられる．

表 6　正例と負例における確信度の比較結果
Table 6　Comparison of the confidence between correct and incorrect data

ラベル	最大値	最小値	平均値	中央値
正例	0.67956	5.9918×10^{-15}	6.3695×10^{-2}	3.6578×10^{-3}
負例	0.76879	3.2844×10^{-18}	1.1233×10^{-2}	4.9674×10^{-7}

5. まとめと今後の課題

本研究では，メソッド名予測手法の code2seq によって，抽出候補のコード片が明確なメソッド名を付けられるような意味的なまとまりを持つかどうかを考慮することで，メソッド抽出リファクタリングの推薦精度を改善する方法を提案した．既存手法の GEMS を拡張し，推薦に用いる予測モデルの特徴量に code2seq が算出する確信度を加えることで，精度の改善が見られるかを検証した．

確信度を加えたモデルと加えなかったモデルのそれぞれを利用して，リファクタリング対象の推薦精度を評価し比較する実験を行った．実験の結果，予測モデルの特徴量に確信度を加えることで，推薦精度が向上することを確認できた．

今後の課題としては，学習データを拡張し，より大きなデータセットでモデルの構築を行うことが挙げられる．また，本研究では code2seq が最上位に予測したメソッド名の確信度のみを使用したが，上位複数個の予測メソッド名の確信度や，その確信度同士の差分を新たに特徴量に追加することで，手法を拡張していくことも検討していきたい．

参考文献

[1] M. Fowler. Refactoring: Improving the design of existing code. Addison-Wesley, 1999. In Proceedings of the 2nd XP Universe and First Agile Universe Conference, p. 256, 2002.
[2] T. Mens and T. Tourwe. A survey of software refactoring. IEEE Transactions on Software Enginering (TSE), Vol. 30, No. 2, pp. 126-139, 2004.
[3] G. C. Murphy, M. Kersten, and L. Findlater. How are java software developers using the eclipse ide? IEEE Software, Vol. 23, No. 4, pp. 76-83, 2006.
[4] A. Chatzigeorgiou and A. Manakos. Investigating the Evolution of Bad Smells in Object-Oriented Code. International Conference on the Quality of Information and Communications Technology (QUATIC), pp. 106-115, 2010.
[5] N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. Journal of Systems and Software (JSS), Vol. 84, No. 10, pp. 1757-1782, 2011.
[6] D. Silva, R. Terra, and M. T. Valente. Recommending automated extract method refactorings. In Proceedings of the 22nd International Conference on Program Comprehension (ICPC), pp. 146-156, 2014
[7] D. Silva, R. Terra, and M. T. Valente. Jextract: An eclipse plug-in for recommending automated extract method refactorings. In Brazilian Conference on Software: Theory and Practice (Tool Track), pp. 1-8, 2015.
[8] C. Sofia, A. Apostolos, C. Alexander, G. Antonios, and A. Paris. Identifying extract method refactoring opportunities based on functional relevance. IEEE Transactions on Software Enginering (TSE), Vol. 43, No. 10, pp. 954-974, 2017.
[9] S. Xu, A. Sivaraman, S. Khoo and J. Xu. GEMS: an extract method refactoring recommender. IEEE 28th International Symposium on Software Reliability Engineering (ISSRE), pp. 24-34, 2017.
[10] U. Alon, S. Brody, O. Levy, and E. Yahav. code2seq: Generating sequences from structured representations of code. In International Conference on Learning Representations (ICLR), 2019.
[11] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. code2vec: Learning distributed representations of code. In Proceedings of the ACM on Programming Languages, Vol. 3, No. POPL, pp. 40:1–40:29, 2019.
[12] D. Silva, N. Tsantalis and M. T. Valente. Why we refactor? confessions of GitHub contributors. In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp. 858-870, 2016.
[13] AishwaryaSivaraman/GEM: Code for the model presented in the paper: "GEMS: an extract method refactoring recommender". https://github.com/Aishwarya-Sivaraman/GEM.
[14] tech-srl/code2seq: Code for the model presented in the paper: "code2seq: Generating sequences from structured representations of code". https://github.com/tech-srl/code2seq.
[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, Vol. 12, No. Oct, pp. 2825–2830, 2011.
[16] T. Akiba, S. Sano, T. Yanase, T. Ohta and M. Koyama. Optuna: A next-generationhyperparameter optimization framework. In Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), pp.2623–2631, 2019.