機械学習を用いたメソッド抽出リファクタリングの推薦手法

後藤 祥[1]　吉田 則裕[2]　藤原 賢二[3]　崔 恩瀞[1]　井上 克郎[1]

概要：リファクタリングとは “外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること” であり，ソフトウェア開発における重要な活動の 1 つである．これまで，複数のリファクタリング対象の推薦手法が提案されているが，推薦の基準に使用されているソースコードの Bad Smell に定量的な定義が存在しないことなどから，リファクタリング対象の推薦基準を決定することは難しい．本研究では，メソッド抽出というリファクタリングパターンについて，実際にリファクタリングが行われたメソッドを収集し，それらの特徴量を用いた機械学習によって，メソッド抽出リファクタリングの対象を推薦する手法を提案する．実験として，5 つのオープンソースソフトウェアに提案手法を適用した結果，メソッド抽出の対象となるメソッドのうち，6 割から 9 割程度を特定することができていることがわかった．また，実験の結果から，メソッド抽出が行われるか否かに，メソッドの文の数と凝集度が大きく関与していることがわかった．

1. はじめに

リファクタリングとは “外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること” である [3]．リファクタリングは，主にソフトウェアの保守性や可読性を向上させることを目的としており，ソフトウェア開発における重要な活動の 1 つである．

大規模なソフトウェアから，開発者が手作業でリファクタリング対象を特定することは困難であるため，リファクタリング対象を自動で特定し開発者へ推薦する手法が提案されており，それらの手法に基づく支援ツールが開発されている [2], [6], [13]．Fowler は，どのようなソースコードに対してリファクタリングを検討すべきかを，ソースコードの Bad Smell として定義しており [3]，リファクタリング対象の推薦はこの Bad Smell を基準にして行われている．しかし，Bad Smell には定量的な定義が存在しないことや，プロジェクトごとにリファクタリングを行う基準が異なる場合があることから，リファクタリング対象の推薦基準を決定することは難しい．そのため，既存の支援ツールでは推薦結果のフィルタリングや，実行時のパラメータ設定の変更が可能になっている．しかし，パラメータ設定をどのように変更すればどのように結果が変化するかを開発者が予測することは困難である．

このような問題に対して，実際にリファクタリングが行われた事例を収集し，リファクタリング対象となったソースコードの特徴を用いた機械学習を行うことで，プロジェクトごとのリファクタリングを行う基準を推薦に反映させることができると考えられる．バグ予測などの分野では機械学習を用いた手法が多く提案されており [8], [11], [12]，ソフトウェア開発履歴から抽出した情報を用いたバグ予測に機械学習が有効な手法であることが示されている．しかし，リファクタリング対象の推薦に機械学習を用いた手法は，筆者らの知る限り存在しない．

本研究では，メソッド抽出というリファクタリングパターンについて，実際にリファクタリングが行われたメソッドを収集し，それらの特徴量を用いた機械学習によって，メソッド抽出リファクタリングの対象を推薦する手法を提案する．メソッド抽出リファクタリングは，実行される回数の多いことが既存研究 [10] によって明らかにされている点や，メソッドはソフトウェア中に大量に存在し，その中から開発者がリファクタリング対象を選択するのは困難である点から，リファクタリング対象の推薦手法による支援の必要がある．リファクタリング対象の推薦に機械学習を用いる利点としては以下の点が挙げられる．

- 実際にリファクタリングが行われた対象の特徴を学習することで，パラメータ設定をせずに開発者の考えに合った対象を推薦することができる
- プロジェクトごとに学習を行うことで，プロジェクト

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[3] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology

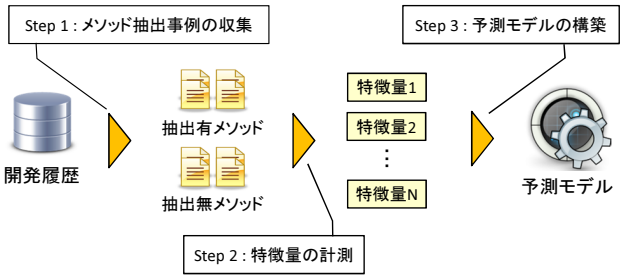

図 1　提案手法の概要図

ごとにリファクタリングを行う基準が異なる場合でも対応可能である．

- 学習に使用するリファクタリング事例を収集することで，他のリファクタリングパターンへ拡張可能である．

本研究では，機械学習に用いるメソッドの特徴として，メソッドの文の数や凝集度など，21 種類のメトリクスを使用した．これらの特徴をもとに機械学習を用いてメソッド抽出の対象となるかどうかを判別するモデルを作成した．

手法の評価を行うために，オープンソースのソフトウェアに対して適用実験を行った．まず，実験の準備として，実験対象ソフトウェアの履歴にリファクタリング検出ツールを適用し，メソッド抽出が行われたメソッドと行われなかったメソッドを収集した．次に，それぞれのメソッドについて特徴の計測を行い，実験用のデータセットを準備した．適用実験では，これらのデータセットを学習用セットと評価用セットの 2 つに分割して，学習用セットから機械学習で構築した予測モデルを用いることで，評価用セットのメソッドについて，メソッド抽出が行われたかどうかを判別できるかを評価した．実験の結果，提案手法を用いることで，メソッド抽出の対象となるメソッドのうち，6 割から 9 割程度を特定することができていることがわかった．また，実験の結果から，メソッド抽出が行われるか否かに，メソッドの文の数と凝集度が大きく関与していることがわかった．

以降，2 章では，提案手法について説明し，3 章では，適用実験について述べる．最後に，4 章でまとめと今後の課題について述べる．

2. 提案手法

提案手法の概要を表した図を図 1 に示す．提案手法は，ソフトウェアの開発履歴を入力として，メソッド抽出事例の収集，メソッドの特徴量の計測，機械学習を用いたモデル構築の 3 つの処理を行い，あるメソッドがメソッド抽出対象であるかを予測するためのモデルを出力する．以降，それぞれの手順について詳細に説明する．

2.1 Step 1 : メソッド抽出事例の収集

機械学習を用いて，メソッド抽出対象を判別するモデルを作るためには，メソッド抽出が行われたメソッドの特徴と，メソッド抽出が行われなかったメソッドの特徴を調べる必要がある．本節では，それぞれのメソッドをどのようにして収集したかを述べる．

まずメソッド抽出が行われたメソッドについて説明する．メソッド抽出事例の収集には，藤原らが提案しているリファクタリング検出ツール [15] を用いる．本研究では，藤原らのツールから出力される類似度の閾値を 0.3 とし，閾値以上の類似度のメソッド抽出事例を使用する．これは，藤原の既存研究において，類似度の閾値が 0.3 の時に最も精度よくメソッド抽出事例の検出が行えることが示されているためである．閾値でフィルタリングを行った後，メソッド抽出の対象となったメソッドについて，抽出が行われる直前のリビジョンのものを取得する．これらのメソッドをメソッド抽出が行われたメソッドとして，以降の特徴量の計測や機械学習に用いる．

次に，メソッド抽出が行われなかったメソッドの収集方法について説明する．本研究では，メソッド抽出が行われなかったメソッドは，リポジトリからランダムに収集する．具体的な手順としては，まず，ランダムでリビジョンを選択し，そのリビジョンに存在する全てのメソッドの中から，次のコミットでメソッド抽出が行われていないメソッドを 1 つをランダムで選択するという方法である．これらの手順を，学習に必要なメソッドが集まるまで繰り返すことで，メソッド抽出が行われなかったメソッドを収集する．

2.2 Step 2 : 特徴量の計測

Step 1 で収集したメソッドに対して，特徴量の計測を行う．表 1 に本研究で使用する 21 種類の特徴量を示す．

リファクタリングは可読性や保守性の向上を目的としており，それらが低いソースコードに対して行われていると考えられる．そのため，本研究では，可読性や保守性に関連する特徴量を選択した．また，最終的なモデル構築の際には，変数選択アルゴリズムによって有用でない特徴量を除外するため，計測する特徴量はメソッド単位のものからクラス単位のものまで，なるべく多くのものを選択した．そのため，本研究で使用する特徴量は，メソッド抽出リファクタリングを特定するために十分な量であると考えられる．

特徴量のうち，クローンの有無についてはコードクローン検出ツールである CCFinder[7] を用いて検出する．その他の特徴量については，ソースコード解析ツール MASU[5] や Eclipse jdt[*1] を用いて計測する．

本研究では，凝集度を表す特徴量として，メソッドレベルの凝集度メトリクスであるスライスベースの凝集度メトリクスを用いる [14]．スライスベースのメトリクスの計算

[*1] http://www.eclipse.org/jdt/

に必要となるメソッドの出力変数は，Tsantalis らの定義に従い，メソッドの返り値と，メソッドの本体とスコープが一致する変数とした [13]．スライスベースのメトリクスは，メソッドの出力変数に基づいて凝集度を計算するものであり，プログラムスライシングの基準となる変数が存在しないメソッドに対しては，凝集度を計算することができない．

2.3 Step 3：予測モデルの構築

計測した特徴量を基に，機械学習アルゴリズムを用いて，メソッド抽出の対象を判別するモデルを構築する．機械学習には，フリーのデータマイニングツールである Weka[*2] を用いる．Weka は，機械学習アルゴリズムだけでなく，変数選択アルゴリズムや予測モデルの評価機能など，機械学習に関連した多くの機能を実装したツールである．

予測モデルの構築する前に，データセット中の欠損値の補完と，変数選択を行う．データセット中の欠損値の補完は，出力変数が存在しないメソッドに対して，値を計算することができないスライベースのメトリクス Tightness, Coverage，Overlap に対する処理である．欠損値が存在すると正しく予測することができないため，欠損値が存在する要素を除外するか，定数値やその特徴の平均値などで欠損値を補完する必要がある．本手法では欠損値の補完法の 1 つである，その特徴量の平均値を用いた補完を行う．欠損値の補完のあと，有用な特徴量のみをモデルの構築に用いるために，変数選択を行う．変数選択アルゴリズムは複数提案されており，対象データセットや使用する予測モデルによってどのアルゴリズムが優れているかは異なる．Hall らが行った変数選択アルゴリズムの比較 [4] では，ラッパー法を用いて変数選択を行うと，多くの場合予測性能が最も良いという結果が示されているため，本研究ではラッパー法を用いて変数選択を行う．

提案手法では，収集したメソッド抽出事例とその特徴量を学習データとして，決定木とロジスティック回帰，ベイジアンネットの 3 種類の予測モデルを構築する．構築した予測モデルでは，あるメソッドとその特徴量が与えられた時に，そのメソッドに対してメソッド抽出を行うべきであるか判断することができる．そのため，メソッド抽出対象の推薦にあたっては，まず対象のメソッドの特徴量の計測を行い，次にそれらの特徴量を予測モデルに与えてメソッド抽出の対象であるか判別し，開発者に通知するという手順になる．

3. 適用実験

提案手法の有効性を評価するために，オープンソースのソフトウェアに対して提案手法を適用した．本研究では，実験対象として 5 個のオープンソースソフトウェアを選択した．実験対象の一覧を表 2 に示す．これらのソフトウェアは全て Java 言語を用いて記述されている．また，表 2 中のファイル数とメソッド数は，最新リビジョンにおける Java のファイル数とメソッド数である．本節では，実験の手順と評価尺度について述べる．

3.1 実験手順

まず，表 2 に示したソフトウェアに対して，藤原らのリファクタリング検出ツールを適用して，メソッド抽出が行われた事例の収集を行う．そして，メソッド抽出が行われなかったメソッドをメソッド抽出が行われたメソッドと同じ数だけランダムで選択し，これらを合わせて実験用データセットとする．

次に，実験用データセットのメソッドの特徴量の計測，モデルの構築と評価を行う．モデルの構築と評価には，10 分割交差検定を用いる．10 分割交差検定は，データセットを 10 個のブロックに分割し，そのうち 1 個を評価セット，残り 9 個を学習セットとしてモデルの構築と評価を行うという処理を，評価セットに用いるブロックを変化させながら 10 回繰り返す手法である．本実験では，モデルの構築に用いる学習セットはメソッド抽出が行われたメソッドと行われなかったメソッドの割合は 50%ずつのものを用いて，評価セットには，メソッド抽出が行われたメソッドの割合を 50%から 10%まで 10%で刻みで変化させて評価を行う．これは，評価セット中のメソッド抽出が行われたメソッドの割合を変化させた時に，どのように結果が変化するかを確認するためである．

3.2 評価尺度

モデルの評価には，Precision と Recall，F 値という 3 つの評価尺度を用いる．以下それぞれの尺度の定義について説明する．

Precision は，モデルによってメソッド抽出の対象であるとされたメソッドのうち，実際にメソッド抽出が行われたメソッドの割合である．Recall は，データセット中に存在する，全てのメソッド抽出が行われたメソッドのうち，モデルによってメソッド抽出の対象であるとされたメソッドの割合である．そして，トレードオフの関係にある Precision と Recall を総合的に評価するため，これらの値の調和平均である F 値を使用する．

表 2 実験対象ソフトウェア

ソフトウェア名	リビジョン数	ファイル数	メソッド数
Ant	12783	1222	12369
ArgoUML	17748	1904	14284
jEdit	5787	576	6632
jFreeChart	916	1060	10495
Mylyn	8414	1028	8936

[*2] http://www.cs.waikato.ac.nz/ml/weka/

表 **1** 機械学習に使用する特徴量

カテゴリ	特徴量 (略称)
メソッドのサイズ	メソッド中の文の数 (NOS)
メソッドのシグネチャ	メソッドの引数の数 (ARG)，返り値の有無 (RET)，アクセスレベル (ACCESS)
複雑度 [9]	サイクロマチック数 (CYCLO)
凝集度 [14]	Tightness，Coverage，Overlap
メソッドの構文	ループ数 (LOOP)，if 文数 (IF)，case 文数 (CASE)，ブロック数 (BLOCK)，ネストの深さ (NEST)
メソッド内の変数	ローカル変数の数 (VAR)
CK メトリクス [1]	WMC，DIT，CBO，NOC，RFC，LCOM
コードクローン	コードクローンの有無 (CLONE)

以下に，Precision と Recall，F 値の定義式を示す．式において，E をメソッド抽出が行われたメソッドの集合，C_E をモデルによってメソッド抽出対象であるとされたメソッドの集合とする．

$$Precision = \frac{|E \cap C_E|}{|C_E|} \tag{1}$$

$$Recall = \frac{|E \cap C_E|}{|E|} \tag{2}$$

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3}$$

3.3 実験結果

表 3 に，藤原らのリファクタリング検出ツールによって，検出されたメソッド抽出事例の数を示す．jFreeChart のみ，他のソフトウェアに比べリビジョン数が少なかったため，検出されたメソッド抽出の事例数も少なくなっている．jFreeChart 以外のソフトウェアでは，490 個から 766 個のメソッド抽出事例数が検出された．次に各ソフトウェアごとに，メソッド抽出が行われたメソッドを，メソッド抽出が行われたメソッド同じ数だけランダムで選択した．表 3 に，作成したデータセット中のメソッド数を示す．

次に，データセットを用いて予測モデルの構築と評価を行った結果を示す．Precision と Recall，F 値について，結果を表したグラフを図 2 に示す．グラフの縦軸は各評価値，横軸は評価セット中のメソッド抽出が行われたメソッドの割合である．これらの図には，各ソフトウェアに対する結果と比較のためのベースラインの結果 (図中の Base) を示している．ベースラインの結果とは，あるメソッドが与えられた際に，メソッド抽出の対象であるかどうかをランダム (50%ずつの確率) で返すモデルに対する結果である．このようなランダムなモデルを用いた結果よりも良い結果であれば，使用した特徴量がメソッド抽出が行われるかどうかに関係したものであり，学習の効果があったことを意味する．

表 **3** メソッド抽出リファクタリングの検出結果

	メソッド抽出数	データセットのメソッド数
Ant	766	1532
ArgoUML	740	1480
jEdit	502	1004
jFreeChart	90	180
Mylyn	490	980

グラフをみると，全ての場合においてベースラインを上回る結果となっていることがわかる．それぞれの評価値ごとにみると，Precision は評価セットの割合によって値が大きく変化しているが，Recall は，ほぼ横ばいで値に変化がみられなかった．Recall については，ほとんどの場合においてが 0.6 から 0.9 程度の値となっており，メソッド抽出の対象であるメソッドのうち，6 割から 9 割程度が提案手法によって特定できていることがわかった．モデルごとの結果をみると，使用するモデルによって大きな差がないが，それぞれのモデルごとに結果の平均値を調べた結果，Precision についてはロジスティック回帰が優れており，Recall については決定木が優れていることがわかった．対象ソフトウェアごとに違いをみると，jFreeChart に対する結果が良く，jEdit に対する結果が悪くなっている．その他の 3 つのソフトウェアについては，評価値に大きな違いはみられなかった．

次に，変数選択の結果から，どの特徴量が予測に有用であったかを調べた結果を示す．表 4 は，変数選択によって各特徴量が選択された回数を示している．表 4 では，各ソフトウェアごとの特徴量が選択された回数と，合計回数を示しており，合計回数で降順に並べている．本手法で変数選択に用いているラッパー法は，各モデルごとに変数選択を行う手法である．そのため，各ソフトウェアについて 3 回変数選択を行っており，各ソフトウェアにおける選択回数の最大値は 3，合計の選択回数の最大値は 15 となる．

表 4 の結果から，本実験においてはメソッドの文の数である NOS と，スライスベースの凝集度メトリクスである Coverage が最も選択された特徴量であることがわかった．各ソフトウェアにおける結果を比較すると，対象ソフトウェアによって有用な特徴量が異なることがわかる．例えば，ブロック数を表す BLOCK は，Ant と ArgoUML，jFreeChart では全てのモデルにおいて選択されているが，

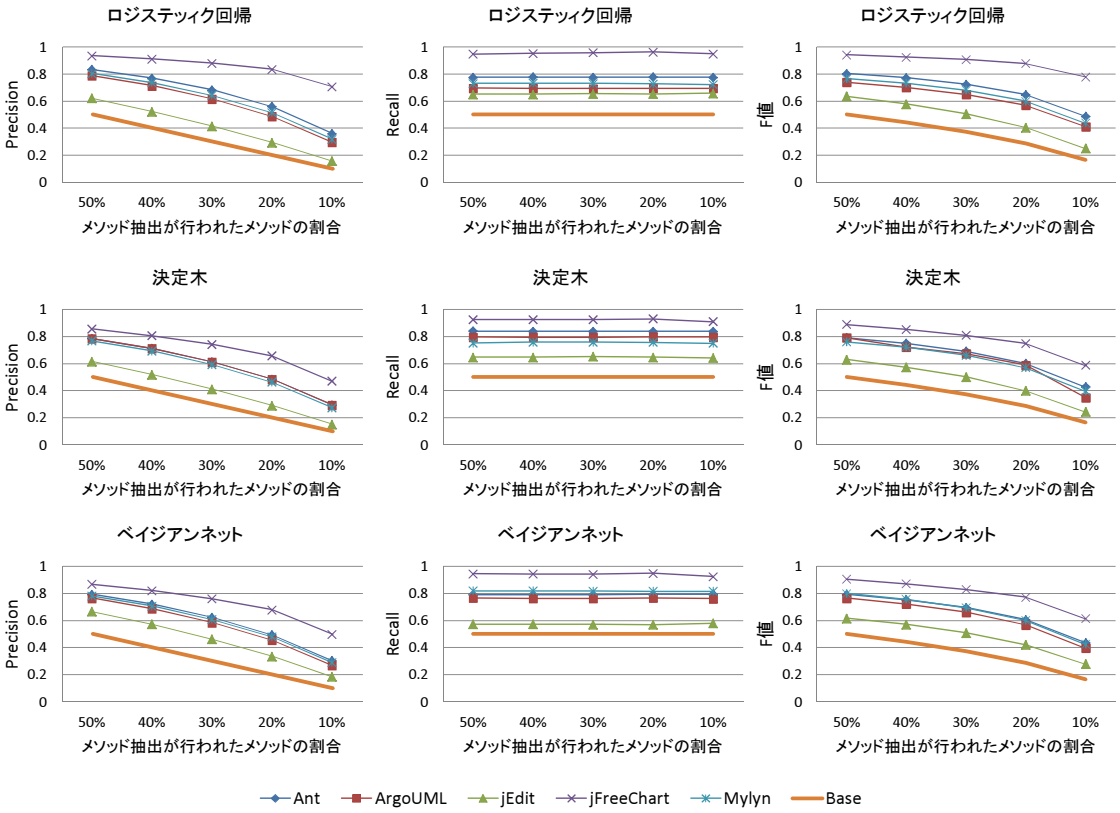

図 **2**　評価結果 (Precision, Recall, F 値)

jEdit では 1 度しか選択されていない．

表 **4**　各特徴量が選択された回数

	Ant	ArgoUML	jEdit	jFreeChart	Mylyn	合計
NOS	2	3	2	3	3	13
Coverage	3	2	3	2	3	13
ARG	3	2	3	3	1	12
BLOCK	3	3	1	3	2	12
CBO	3	3	1	3	2	12
RET	3	3	2	2	1	11
IF	3	1	2	3	2	11
WMC	1	2	2	3	3	11
DIT	2	2	2	3	2	11
ACCESS	3	2	3	1	1	10
LOOP	2	3	2	1	2	10
NEST	1	1	2	3	3	10
RFC	2	3	1	3	1	10
Tightness	1	3	2	1	2	9
VAR	3	2	1	1	1	8
NOC	2	2	1	2	1	8
LCOM	2	1	3	1	1	8
CYCLO	1	1	1	2	2	7
CLONE	0	2	1	1	3	7
Overlap	1	1	1	2	1	6
CASE	1	0	2	1	2	6

3.4　考察

まず，各ソフトウェアごとの結果について考察する．各ソフトウェアごとの結果の差をみると，jFreeChart に対する結果が良く，jEdit に対する結果が他のソフトウェアに比べて悪かった．jFreeChart についてより詳細に開発履歴の調査を行った結果，検出されたメソッド抽出事例のうち多くが同一の開発者によって行われていたことがわかった．このことから，jFreeChart ではメソッド抽出が行われるメソッドの特徴が一貫しており，予測に有効に作用したのではないかと考えられる．jEdit については，データセットの特徴量を調べた結果，メソッド抽出が行われたメソッドと行われなかったメソッドで，特徴量の値に大きな差がないことがわかった．そのため，jEdit についてはモデルによる予測性能が低かったと考えられる．このように，予測性能には使用するデータセットが大きく影響することがわかった．

次に，評価セット中のメソッド抽出が行われたメソッドの割合を変化させた場合の，結果の変化について考察する．図 2 に示したグラフから，メソッド抽出が行われたメソッドの割合を減少させても，本実験においてベースラインとしたランダムで予測を行うモデルより良い結果であることがわかる．これは，今回選択した特徴量がメソッド抽出が行われるかどうかに関連しており，学習によって予測性能が向上したことを示している．評価値ごとの変化をみる

と，Recall に大きな変化はないが，Precision はメソッドの割合の減少にともなって低下していることがわかる．そのため，メソッド抽出が行われたメソッドの割合を減少させても高い予測性能を維持できるように，手法を改善する必要があると考えられる．手法の改善案としては，特徴量の追加，出力結果のフィルタリングやデータセットを開発者ごとに分割するなどの方法が挙げられる．データセットを開発者ごとに分割する方法は，jFreeChart に対する予測性能が良かったことから，有効な手段であると考えられる．

次に，各特徴量の有用性について述べる．表 4 より，メソッドの文の数である NOS と，スライスベースの凝集度メトリクスである Coverage が最も選択された特徴量であった．NOS は，メソッドのサイズを表す特徴量であり，この特徴量が多く選択されていたことから，メソッド抽出を行うかどうかの判断において，メソッドのサイズが重要な要因であることがわかる．Coverage は，メソッドの内の出力変数に関連した文の数とメソッドの文数の比の平均を表している．Coverage が低いことは，出力変数に関連のない文がメソッドに多く存在することを意味している．このことから，メソッドの出力変数に関連した文とそうでない文をメソッド抽出を用いて別々のメソッドに分割しているのではないかと考えられる．

変数選択によって選択された回数が最も少ない特徴量は，スライスベースの凝集度メトリクスである Overlap と，メソッド中の case 文の数を表す CASE であった．これらの 2 つの特徴量について調査したところ，ともに値の分布が偏っていることがわかった．Overlap は，メソッド内に出力変数が 1 つしかない場合は 1 になり，処理が全く重複しない出力変数が存在する場合は 0 になるため，多くのメソッドの Overlap の値は 0 か 1 となる．また CASE についても，今回の対象のメソッドの多くは case 文が使われていなかった．

変数選択の結果，多く選択される特徴量とあまり選択されない特徴がわかったが，最も選択されなかった特徴量でも 15 回のうち 6 回選択されている．また，各ソフトウェアにおける変数選択の結果も異なるものであり，どの特徴量が有用であるかは変数選択アルゴリズムを適用するまでわからない．そのため，本提案手法のように，特徴量を計測する段階では多くの特徴量を計測して，モデルの構築を行う際に変数選択によって有用なものを選別する方法が有効であると考えられる．

4. まとめと今後の課題

本研究では，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推薦する手法を提案した．提案手法では，メソッド抽出の対象となったメソッドについて，サイズや複雑度など 21 種類の特徴量を計測し，それらを用いて推薦のための予測モデルを構築した．

実験として，5 つのオープンソースソフトウェアに対して手法を適用し評価を行った．評価の結果，メソッド抽出対象となるメソッドのうち 6 割から 9 割程度を提案手法によって特定できていることがわかった．また，変数選択アルゴリズムの適用結果から，メソッド中の文の数と，スライスベースの凝集度メトリクスである Coverage が有用な特徴量であることがわかった．

今後の課題としては，予測性能の向上と，学習セットと評価セットに別々のソフトウェアから作成したデータセットを用いた実験を行うことが挙げられる．予測性能については，特徴量の追加やデータセットを開発者ごとに分割するなどの方法によって改善することができると考えられる．

参考文献

[1] Chidamber, S. R. and Kemerer, C.: A metrics suite for object oriented design, *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6, pp. 476–493 (1994).
[2] Emden, E. V. and Moonen, L.: Java quality assurance by detecting code smells, *Proc. of WCRE*, pp. 97–106 (2002).
[3] Fowler, M.: *Refactoring: Improving the Design of Existing Code*, Addison Wesley (1999).
[4] Hall, M. A. and Holmes, G.: Benchmarking Attribute Selection Techniques for Discrete Class Data Mining, *IEEE Trans. Knowl. Data Eng.*, Vol. 15, No. 6, pp. 1437–1447 (2003).
[5] Higo, Y., Saitoh, A., Yamada, G., Miyake, T., Kusumoto, S. and Inoue, K.: A Pluggable Tool for Measuring Software Metrics from Source Code, *Proc. of IWSM-MENSURA*, pp. 3–12 (2011).
[6] Hotta, K., Higo, Y. and Kusumoto, S.: Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph, *Proc. of CSMR*, pp. 53–62 (2012).
[7] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670 (2002).
[8] Lee, T., Nam, J., Han, D., Kim, S. and Hoh, I. P.: Micro interaction metrics for defect prediction, *Proc. of ESEC/FSE*, pp. 311–321 (2011).
[9] McCabe, T. J.: A Complexity Measure, *IEEE Trans. Softw. Eng.*, Vol. 2, No. 4, pp. 308–320 (1976).
[10] Murphy-Hill, E., Parnin, C. and Black, A. P.: How We Refactor, and How We Know It, *IEEE Trans. Softw. Eng.*, Vol. 38, No. 1, pp. 5–18 (2011).
[11] Nagappan, N., Zeller, A., Zimmermann, T., Herzig, K. and Murphy, B.: Change Bursts as Defect Predictors, *Proc. of ISSRE*, pp. 309–318 (2010).
[12] Nam, J., Pan, S. J. and Kim, S.: Transfer defect learning, *Proc. of ICSE*, pp. 382–391 (2013).
[13] Tsantalis, N. and Chatzigeorgiou, A.: Identification of extract method refactoring opportunities for the decomposition of methods, *Journal of Systems and Software*, Vol. 84, No. 10, pp. 1757–1782 (2011).
[14] Weiser, M.: Program slicing, *Proc. of ICSE*, pp. 439–449 (1981).
[15] 藤原賢二，吉田則裕，飯田　元：ソフトウェアリポジトリを対象とした細粒度リファクタリング検出，ソフトウェア工学の基礎ワークショップ，pp. 101–106 (2013).