修士学位論文

題目

ソースコードの特徴量を用いた機械学習による
メソッド抽出リファクタリング推薦手法

指導教員
井上 克郎 教授

報告者
後藤 祥

平成 26 年 2 月 5 日

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 ソフトウェア工学講座

平成 25 年度 修士学位論文

ソースコードの特徴量を用いた機械学習によるメソッド抽出リファクタリング推薦手法

後藤 祥

内容梗概

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．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．

これまで，リファクタリング作業の支援を目的とした研究が数多く行われており，その中でリファクタリング対象を特定して開発者に推薦することで支援を行う研究も行われている．リファクタリング対象の特定においては，リファクタリングを行うべきソースコードを探すための基準が必要である．既存研究では，Fowler が定義したコードの不吉な臭い (Bad Smell) を基準にしている場合が多いが，この不吉な匂いは定量的な定義ではないことに加え，実際に開発者がどのようなコードを対象にリファクタリングを行っているかは調査されていない．そのため，既存研究では開発者にとって有用なリファクタリング対象を特定できない可能性がある．

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対象の特徴を調査し，それに基づいた推薦を行うことによって，開発者にとって有効な支援を行うことができる．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の特徴を調査することで，開発者の考えにあったメソッドをリファクタリング対象として推薦する手法を提案する．メソッドの特徴として 21 種類のメトリクスを使用し，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．

実際に，オープンソースのソフトウェアに対して，提案手法の適用と評価を行った結果，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．

主な用語

リファクタリング (Refactoring)
機械学習 (Machine Learning)
リポジトリマイニング (Repository Mining)

目 次

1 まえがき **5**

2 背景 **7**

2.1 リファクタリング 7

2.1.1 メソッド抽出リファクタリング 7

2.1.2 リファクタリング動向の調査 8

2.2 リファクタリングの支援手法 10

2.2.1 リファクタリング対象の推薦手法とその問題点 10

2.2.2 リファクタリング支援ツール 11

2.3 リファクタリング検出ツール 12

2.3.1 版管理システム 12

2.3.2 既存のリファクタリング検出ツール 12

2.4 機械学習 14

2.4.1 予測モデル 15

2.4.2 変数選択 17

2.4.3 モデルの評価方法 20

2.4.4 機械学習を用いたバグ予測 20

3 提案手法 **22**

3.1 Step 1：メソッド抽出事例の収集 22

3.2 Step 2：メソッドの特徴量の計測 23

3.2.1 準備 23

3.2.2 特徴量の定義 25

3.3 Step 3：メソッド抽出対象の特定のためのモデル構築 30

4 適用実験 **32**

4.1 実験方法 32

4.1.1 実験手順 32

4.1.2 評価尺度 33

4.2 実験結果と考察 34

4.2.1 実験結果 34

4.2.2 考察 35

4.3 妥当性への脅威 41

5　まとめと今後の課題 **42**

謝辞 **43**

参考文献 **45**

1 まえがき

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 [6]．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．

リファクタリング作業は，リファクタリング対象のソースコードの特定，ソースコードの編集，動作が変わってないことを保証するためのテストなどの手順が必要となり，開発者にとってはコストの大きい作業である．そのため，数多くのリファクタリング支援の研究が行われている [9, 12, 22]．リファクタリング支援の 1 つであるリファクタリング対象の特定については，Fowler が定義したコードの不吉な臭い (Bad Smell) に基づいて，対象の推薦を行っているものが多い．しかし，コードの不吉な臭いは定量的な定義が存在しないことに加え，実際に開発者がコードの不吉な臭いに従ってリファクタリングを行っているのか調査されていない．開発者が実際に対象にしているソースコードと，一般にリファクタリングすべきといわれているコードの不吉な臭いが一致していなければ，開発者にとって有用なリファクタリング対象を特定できない可能性がある．

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対象の特徴を調査し，それに基づいた推薦を行うことによって，開発者にとって有効な支援を行うことができると考えられる．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の特徴を調査することで，開発者の考えにあったメソッドをリファクタリング対象として推薦する手法を提案する．メソッドの特徴として 21 種類のメトリクスを使用し，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から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．本研究では，ソフトウェアの履歴からリファクタリングが適用された箇所を特定するリファクタリング検出ツールを用いてメソッド抽出が行われたかどうか判別した．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 章では背景として，本研究に関連する用語，関連研究，既存研究におけるリファクタリング対象の推薦手法とその問題点について述べる．3 章では，提案手法にとして，メソッド抽出事例の収集から，メソッドの特徴の計測，それらの用いたメソッド抽出対象の推薦方法について述べる．4 章では，適用実験として，実験の方法や結果を述べ，結果に基づいた考察を行う．最後に 5 章で本研究のまとめと今後の課題について述べる．

2 背景

本章では，背景として，本研究に関連する用語，関連研究，既存研究におけるリファクタリング対象の推薦手法とその問題点について述べる．まず，2.1 節で本研究における支援の対象であるリファクタリングについて説明すえる．2.2 節では，リファクタリング支援に関する既存研究について述べ，その中でリファクタリング対象の推薦手法とその問題点について詳細に述べる．2.3 節では，本研究でリファクタリング事例を収集するために用いるリファクタリング検出ツールについて説明する．最後に，2.4 節で，本研究でリファクタリング対象の推薦に用いる機械学習について述べる．

2.1 リファクタリング

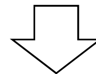
リファクタリングとは，“外部から見たときの振る舞いを保ちつつ，理解や修正が簡単になるように，ソフトウェアの内部構造を整理すること” である [6]．Fowler は文献 [6] の中で，はソフトウェア開発における設計の不備を，リファクタリングによって解消できると述べている．文献 [6] には多くのリファクタリングパターンがまとめられており，それぞれのリファクタリングパターンについて，詳細な手順などが記述されている．Fowler は，どのようなソースコードをリファクタリングすべきかについて，コードの不吉な臭い (Bad Smell) を定義し，この定義に当てはまるソースコードを対象として，リファクタリングを検討すべきとしている．コードの不吉な臭いの例としては，同じようなコードが複数個所に存在する状態である “Duplicated Code”や，あるメソッドの引数が多すぎることを意味する “Long Parameters List”などがある．Fowler はこれらの不吉な臭いについて，どのようなリファクタリングを適用して解消できるか述べているが，定量的な定義は行っておらず，開発者の経験や感覚に基づいてリファクタリングを行う方が良いと述べている．

以降では，本研究で支援の対象としているメソッド抽出リファクタリングについて説明する．また，関連研究として，リファクタリングの調査を行っている既存研究について述べる．

2.1.1 メソッド抽出リファクタリング

メソッド抽出はリファクタリングパターンの 1 つであり，既存のメソッドの一部を新たなメソッドとして抽出することによって，メソッドの分割を行う手法である．メソッド抽出の主な目的は，長すぎるメソッドの分割や，複数の機能が実装されたメソッドの分割である．メソッド抽出を行うことの利点としては，メソッドを適切なサイズに分割することで可読性を向上させることができる点や，機能とメソッドを 1 対 1 に対応させることで，機能追加やバグ修正を行う個所を容易に特定できるようになる点などが挙げられる．メソッド抽出は，既存の調査研究 [15] において開発者が頻繁に行うリファクタリングの 1 つであることがわ

```
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: " + outstanding);
9  }
```

図 1: メソッド抽出の例 [6]

かっていることや，他のリファクタリングパターンの一部として行われる場合があることから，適用される頻度が高い重要なリファクタリングである．

図 1 にメソッド抽出の例を示す．図の例では，printOwing メソッドの 5 行目と 6 行目が抽出され，新たに printDetails メソッドとして定義されている．元のメソッドである printOwing メソッドには，printDetails メソッドの呼び出し文が追加されている．図 1 のようにメソッド抽出を行うことによって，主に 2 つの利点がある．まず 1 つめは，抽出が行われた printOwing メソッドの処理内容が容易に把握できるようになるという点である．メソッドにその処理内容を表す名前をつけることによって，printOwing メソッド内では，バナーの表示と詳細の表示の 2 つの処理を行っていることが容易にわかる．2 つめは，メソッドを機能単位に分割したことで，修正などの保守作業のコストを低下させることができる点である．もし，このソースコードを実行した際に，詳細が正しく表示されていなければ，詳細を表示する部分のソースコードに修正を加える必要がある．この時，メソッド抽出によってメソッドが機能単位に分割されていれば，詳細を表示する機能が printDetails メソッドに実装されているということが容易に特定でき，修正箇所を探すコストを減らすことができる．

2.1.2 リファクタリング動向の調査

リファクタリングの支援を行うためには，開発者がどのようにリファクタリングを行っているかを知ることが重要である．そのため，これまで開発者のリファクタリング動向の調査

が多く行われている．

Murphy-Hill らは，8 つのデータセットを対象にしたリファクタリング動向の調査を行っている [15]．この調査の結果，リファクタリングの適用頻度やリファクタリング支援ツールの利用状況などに関する知見が多く得られている．適用頻度については，リファクタリングは短期間に集中して行われることが多いことや，ローカル変数の名前変更など，変更する箇所が局所的であるようなリファクタリングが多く行われていることが示されている．また，支援ツールについては，多くの開発者がツールの設定を変更せずに利用していることや，ツールを利用せず手動でリファクタリングする場合が多いことが示されている．Murphy-Hill らはこのような調査の結果に基づいてリファクタリング支援ツールを開発することで，よい良いツールを開発することができるとしている．

Vakilian らは，統合開発環境 Eclipse に実装されている自動リファクタリング機能について，その使用状況を，正しく使用された例，使用されなかった例，間違って使用された例に分けて分析している [23]．調査の結果，主に機能が使用されなかった例，間違って使用された例について議論を行い，今後のリファクタリング支援ツールが満たすべき要件について述べている．自動リファクタリング機能が使用されない主な原因は，機能名がわかりにくく何が起こるか直感的に理解できないことや，機能の学習コストが高い点が挙げられている．また，間違って使用された例については，振る舞いが保存されないリファクタリングが行われた場合などで，開発者のツールへの過度な信頼や，ツールが警告をださないことが原因として挙げられている．Vakilian らは，調査結果から，リファクタリング機能はより開発者が使いやすいように設計するべきであると述べている．

Rachatasumrit らは，リファクタリングによるソースコードの編集が，テストにどのような影響を与えるかを調査している [20]．リファクタリングを実施するための障害として，リファクタリングによってバグを混入してしまう恐れがある点や，追加のテストが必要になる点などが挙げられる．Rachatasumrit らはその点に着目し，リファクタリングによる変更をテストするようなテストケースが存在するかどうかや，不具合を混入させた変更のうちどの程度がリファクタリングに関連したものかを調査している．調査の結果として，リファクタリングによる変更のテストは不十分であることや，リファクタリングによって不具合が発生した事例が確認されている．

このように，開発履歴を調査することによって，開発者のリファクタリングに関する動向を知るための研究が多く行われている．これらの調査結果をもとに，リファクタリング支援の手法を提案することで，実際のソフトウェア開発において有効な手法を提案できると考えれる．

2.2 リファクタリングの支援手法

2.2.1 リファクタリング対象の推薦手法とその問題点

これまで，リファクタリング対象を特定して開発者に推薦する手法がいくつか提案されている．これらの既存研究とその問題点について本節で説明する．

Emden らは，コードの不吉な臭い (論文中では Code Smells) を検出，可視化するツールを提案している [5]．Emden らは，コードの不吉な臭いを，対象のソースコードから直接検出できる “Primitive Smell Aspects” と，他の機能から推定することができる “Derived Smell Aspects” の 2 種類に分類し，検出を行っている．また，Emden らのツールは検出したコードの不吉な臭いを，グラフを用いて可視化することできる．このグラフは，頂点がクラスなどのコードの要素を表し，辺が継承やメソッド呼び出しなどの要素間の関係を表している．グラフを用いた可視化を行うことによって，システム全体で検出されたコードの不吉な臭いの概要を容易に把握することができる．

Tsantalis らは，JDeodorant というリファクタリング支援ツールを統合開発環境 Eclipse のプラグインとして，開発している [1]．JDeodorant は，対象のソースコードから不吉な臭いを検出して開発者へと通知するツールであり，一つのクラスが巨大である “God Class” や，メソッドが長く複雑である “Long Method” などの複数の不吉な臭いに対応している．例えば，“Long Method” は，メソッド内のデータや制御の依存関係をもとに処理のまとまりを求めることによって，メソッド抽出リファクタリングの必要があるメソッドを特定している [22]．JDeodorant は，Eclipse と連動することで，リファクタリング対象として検出された位置への移動や，ソースコードのハイライトなどの機能を実装しており，ツールの利用者は検出結果を即時に確認することができる．

Hotta らは，ソフトウェア中からコードクローンを検出して，Template Method の形成というリファクタリングパターンを適用できる対象を推薦する手法を提案している [9]．コードクローンとは，互いに一致または類似したコード片のことであり [27]，Fowler の不吉な匂いの定義では “Duplicated Code” にあたる．Hotta らの手法では，コードクローンの検出結果を用いて，Template Method の形成が適用可能である条件を満たすメソッドの組を探索し，それらのメソッドをリファクタリング対象として開発者へと通知する．Hotta らの手法は，大規模のソフトウェアから高速かつ容易にリファクタリング対象が特定可能である．

ここで説明した，既存研究で行われているリファクタリング対象の推薦手法は，コードの不吉な臭いに基づいてリファクタリング対象を特定し推薦するものである．しかし，前述したとおり，コードの不吉な臭いには定量的な定義は存在しないため，推薦のための基準はツールの開発者が経験的に決定するか，ツールの使用者が個別に設定する必要がある．開発者が推薦の基準を決定する場合，その基準がツールを利用する開発者の考えるリファクタリ

ング対象の基準と異なると，有用なリファクタリング対象の推薦を行うことができない．そのため，推薦のための基準は開発者が個別に設定できるようにする場合が多い．前述した3つの既存研究においても，Emden らのツールと Tsantalis らのツールは開発者が推薦のための基準を設定することができ，Hotta らツールでは推薦結果の並び替えやフィルタリングすることができる．設定を行うことによって，開発者の考えに合った推薦結果を得ることができるが，どのような設定を行えば開発者が望む推薦結果を得られるか知ることは困難である．そのため，開発者は設定の変更と推薦結果の確認を繰り返し行い，より良い設定を探索する必要がある．

2.2.2 リファクタリング支援ツール

リファクタリングを手動をで行うと，振る舞いを保存できずにバグを混入させてしまう場合がある．そのため，自動でリファクタリングを行うツールが多く提案されている．

Murphy-Hill らは，メソッド抽出リファクタリングの作業を支援するツールを開発している [14]．このツールには，ソースコードの選択補助 (Selection Assist) とソースコードのネスト構造の可視化 (Box View)，メソッド抽出リファクタリングに関連したデータフローと制御フローの可視化 (Refactoring Annotation) という 3 つの支援機能が実装されている．ツールの評価として，ツールの使用の有無によってソースコードの選択に失敗した回数や作業時間に差がでるかを調査している．実験の結果，ソースコードの選択に失敗した回数は 30 回程度減少し，作業時間も約 25%から 50%減少させることに成功している．

Lee らは，ドラッグドロップによってリファクタリング作業を実現するツールを提案している [12]．これまでのリファクタリングツールの多くは，メニューを選択することによってツールを起動し，新たに開かれるウィンドウで設定を行ったのち実行するものが多く，リファクタリングを実行するまでに複数の手順を踏む必要があった．Lee らのツールは，そのような問題点を解決するために開発され，Eclipse のエディタやパッケージエクスプローラ上でドラッグドロップを行うだけで，リファクタリングを実行することができるようになっている．評価実験では，被験者を用いて作業時間や作業効率がどのように変化するか調査しており，Lee らのツールを用いてより短時間で効率良くリファクタリングを実行できることが示されている．

これらのリファクタリング支援ツールは，ソースコードを編集する作業を支援するものであり，リファクタリングを行う対象は開発者が探す必要がある．そのため，上記のような支援ツールとは別に，2.2.1 節で説明したような，リファクタリング対象を推薦する手法が提案されている．リファクタリング対象を推薦する手法と，リファクタリング作業の自動化ツールを組み合わせて使うことで，開発者は効率良くリファクタリングを行うことができる．

2.3 リファクタリング検出ツール

リファクタリング検出ツールとは，版管理システムなどを用いて作成されたソフトウェアの開発履歴から，リファクタリングが行われた事例を検出するツールである．リファクタリング検出ツールを用いて，リファクタリングが行われた事例を収集することによって，リファクタリングが行われたソースコードの調査や，リファクタリングを行った開発者の動向の調査が可能になる．本節では，リファクタリング検出ツールの関連知識である版管理システムと，既存研究で提案されているリファクタリング検出ツールについて述べる．

2.3.1 版管理システム

版管理システムとは，ファイルの変更履歴を管理するためのシステムであり，ソフトウェア開発においてはソースコードやドキュメントなどの変更管理に用いられる．版管理システムの例としては，CVS(Concurrent Versions System)[1]，Subversion[2]，Git[3]などが挙げられる．版管理システムにおいて，ファイルの変更を保存しているデータベースをリポジトリと呼び，編集を加えたファイルをリポジトリにアップロードすることをコミットという．また，それぞれのファイルには変更を管理するためのリビジョン番号が付加されており，そのファイルをコミットするたびに番号が 1 づつ加算される．例えば，リビジョン番号が r のファイルに変更を加えてコミットを行うと，そのファイルのリビジョン番号は $r+1$ になる．

版管理システムを使用する主な目的は，ファイルの集中管理や不具合が発生した際の手戻りであるが，それ以外にも，版管理システムの履歴を調査して知見を得ることによって，開発作業を改善するという目的も存在する．版管理システムの履歴を解析して，開発者の動向を調査することで，どのような支援を行えば開発者の作業を効果的に支援できるか知ることができる．実際に，リファクタリング支援の研究においても，版管理システムの履歴を解析して，開発者がどのようにリファクタリングを行っているか調査が行われている [15]．

2.3.2 既存のリファクタリング検出ツール

本節では，既存研究で提案されているリファクタリング検出ツールである，UMLDiff[25]，Ref-Finder[19]，藤原らのツール [26] について説明する．

UMLDiff は 2 つのバージョンから，設計レベルの差分を抽出し，それらの差分をもとに適用されたリファクタリングを検出するツールである．UMLDiff では，まず，入力として与えられた 2 つのバージョンのソースコードに対して，パッケージやクラス，フィールドなどの設計レベルの情報を抽出する．そして，それらの情報をもとに 2 つのバージョン間での

[1] http://www.nongnu.org/cvs/
[2] http://subversion.apache.org/
[3] http://git-scm.com/

設計レベルの差分 (各要素の追加，削除，名前変更など) を求める．これらの設計レベルの差分から，名前の変更や要素の移動といった簡単なリファクタリングを検出することができる．その他の複雑なリファクタリングについても，設計レベルの差分をいくつか組み合わせることによって検出を行っている．UMLDiff は，32 個のリファクタリングパターンを検出することが可能である．

Ref-Finder は，リファクタリングパターンを論理規則で表現し，それらの規則と 2 つのバージョンから抽出したソースコードレベルの差分を用いて，リファクタリングを検出している．Ref-Finder も UMLDiff と同様に，2 バージョン間の差分を抽出して使用するが，Ref-Finder が抽出する情報は，条件分岐やゲッターメソッドとフィールドの関係なども含まれており，UMLDiff より詳細である．また，2 つのメソッドの本体がどの程度類似しているかなどの情報も計算している．そして，あらかじめ定めておいたリファクタリングパターンを表す論理規則 [18] を用いて，その規則を満たすソースコードを推論することによってリファクタリングの検出を行っている．Ref-Finder は，63 個のリファクタリングパターンは検出することができ，現在最も多くのパターンに対応しているリファクタリング検出ツールである．

上記の 2 つのリファクタリング検出ツールは，多くのリファクタリングパターンに対応しており，実験によって高精度でリファクタリングが検出できることが示されている．その一方で，これらのツールは 2 つのバージョン間でのリファクタリング検出を目的としたものであり，リポジトリの複数のリビジョンから一度に検出を行うには適していない．

本研究では，メソッド抽出事例を収集するために，藤原らが提案しているリファクタリング検出ツールを用いる [26]．藤原らのツールは，メソッド抽出リファクタリングのみしか検出できないが，複数のリビジョンからリファクタリングを高速に検出を行うことができる．本研究では，機械学習のために多くのメソッド抽出事例が必要であったため，藤原らのツールを使用した．藤原のツールの詳細な検出手順は以下の 4 つの処理から構成される．また，検出方法の概要を図 2 に示す．

1. リビジョン i と $i+1$ の間で新規に作成されたメソッド M_{new} をメソッド抽出によって新たに作成されたメソッドの候補とする

2. リビジョン i と $i+1$ の間で行の追加および削除が行われたメソッド M を抽出が行われたメソッドの候補とする．

3. M に追加された行 L_{add} の中で，M_{new} が呼び出されているか調べる．

4. M の削除された行 L_{del} と M_{new} の本体に対して，トークンベースの 2-shingles[3] 類似度を計算し，類似度が閾値以上であれば，メソッド抽出が行われたとする．

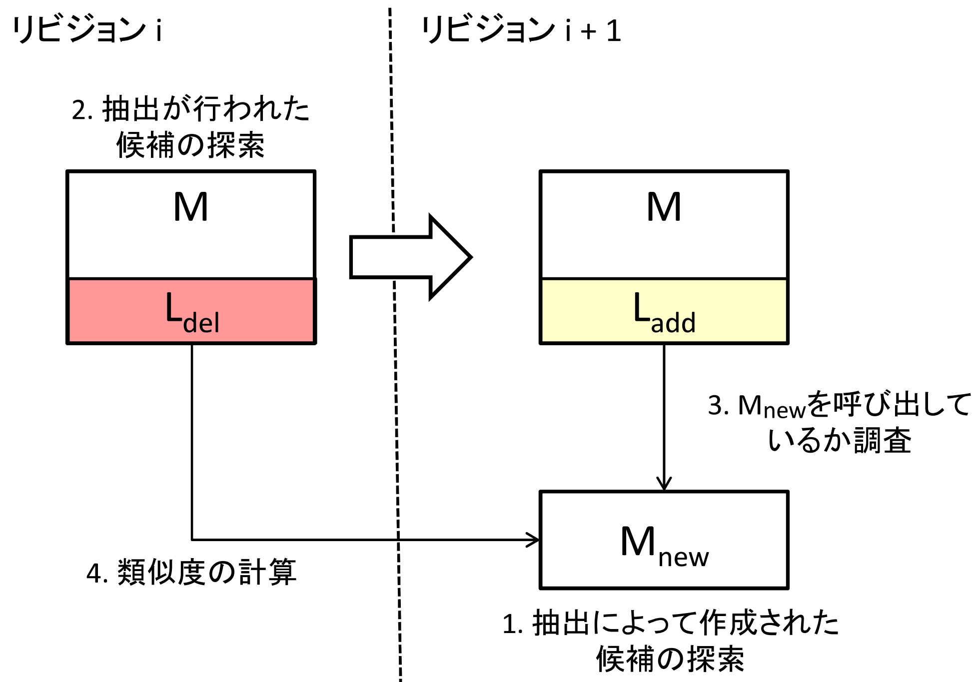


図 2: 藤原らのツールによるメソッド抽出事例の検出

2-shingles は，自然言語処理において，文書間の類似度を計算するために用いられている．2-shingles とは文書 D の先頭から単語を 2 つずつ順番に取り出した組の集合 $SH(D)$ のことである．文書 D_1 と D_2 の 2-shingles 類似度 $sim(D_1, D_2)$ は以下の式で定義される．また，2-shingles 類似度の計算例を図 3 に示す．

$$sim(D_1, D_2) = \frac{|SH(D_1) \cap SH(D_2)|}{|SH(D_1) \cup SH(D_2)|} \tag{1}$$

藤原らのツールでは，コード片からトークン単位の 2-shingles を計算して類似度を算出している．この類似度は 0 から 1 の間の実数値であり，値が高いほどメソッド抽出が行われた可能性が高いことを示している．藤原らが行った実験では，類似度が 0.3 のとき最も検出精度が高いことが示されている．

2.4 機械学習

機械学習とは，説明変数と目的変数の値が既知のデータセットから学習を行い，予測モデルを構築する手法である．予測モデルを用いることによって，説明変数は既知であるが，目的変数の値が未知のデータが与えられた際に，説明変数をもとに目的変数の値を予測することができる．本節では，機械学習について，予測モデルと変数選択，モデルの評価方法について説明する．

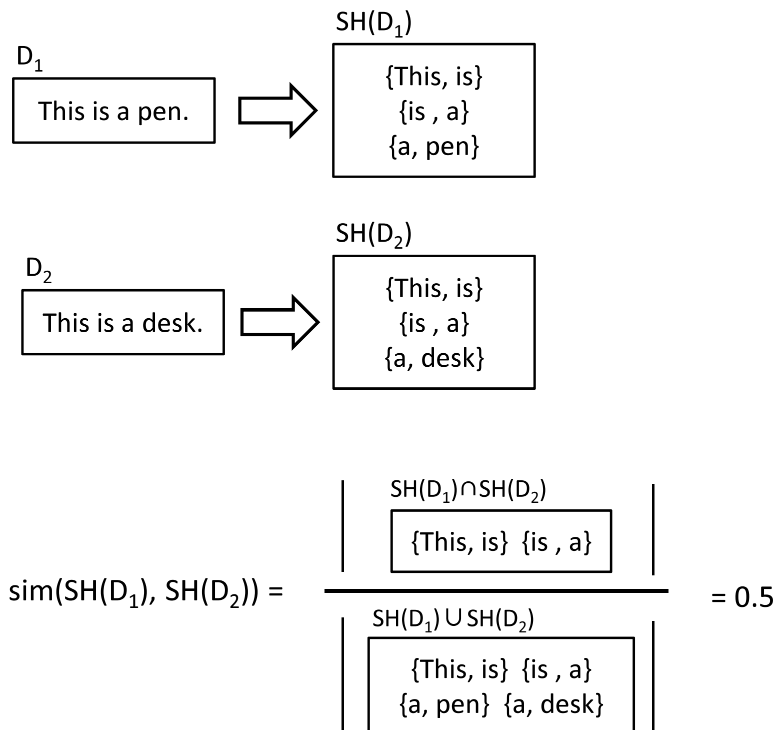


図 3: 2-shingles 類似度の計算例

2.4.1 予測モデル

機械学習で構築する予測モデルには，いくつかの種類がある．ここでは，代表的な予測モデルとして，決定木，ロジスティック回帰，ベイジアンネットについて説明する．

- 決定木

  決定木は木構造の予測モデルである．決定木の各頂点は，説明変数のうち 1 つを表しており，その値によってその頂点からどの子へと遷移するかが決定する．決定木を用いた分類では，根から開始して，説明変数の値によって子へと遷移するという処理を葉に到達するまで繰り返す．各葉は，目的変数の値を表しており，根から分類を行って葉に到達したとき，説明変数はその葉に定められた値に分類される．

  決定木の例を図 4 に示し，この例を用いた予測の方法を説明する．この例では，天気と風という 2 つの説明変数と，外で遊ぶという目的変数がある．図のような決定木を用いて，天気が “晴れ”，風が “弱い” という値をもつデータの目的変数の予測を行うとする．まず，根における分岐では，天気の値が “晴れ” なので，左の子へと遷移する．次の分岐では，風の値が “弱い” なので，左の子へと遷移する．この時点で葉へと到達し，その値が “はい” なので，このデータの目的変数の値は “はい” であると予測される．

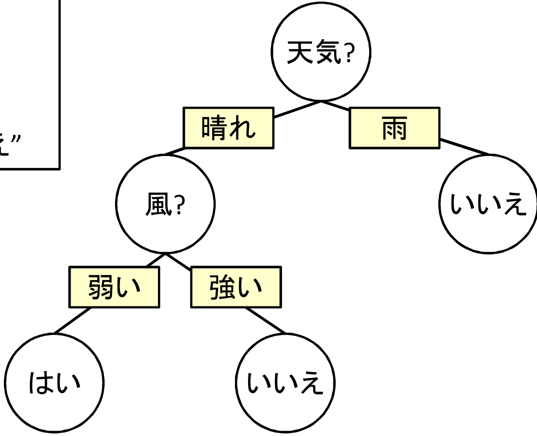


図 4: 決定木の例

- ロジスティック回帰

  ロジスティック回帰は，回帰分析手法の 1 つであり，以下の式で表されるモデルである．ここで，説明変数は n 個とし，X_i は i 番目の説明変数であるとする．

$$P = \frac{1}{1 + e^{-(a+b_1X_1+b_2X_2+\cdots+b_nX_n)}} \tag{2}$$

  式において，右辺の分母中の e の指数部分は，説明変数の線形結合式となっている．左辺 P は 0 からの 1 の値をとり，説明変数の線形結合式の部分を X とおくと，図 5 のようなグラフとなる．目的変数の予測においては，回帰式に説明変数の値を代入して P の値を求め，求めた値が閾値超えているかどうかで目的変数の値を決定する．

- ベイジアンネット

  ベイジアンネットは，有向グラフ形式の予測モデルで，説明変数や目的変数を確率変数とみなして，それらの関係を条件付き確率で表したモデルである．目的変数を Y，説明変数は n 個とし，X_i は i 番目の説明変数であるとすると，目的変数 Y に関する条件付き確率は以下の式になる．

$$P(Y|X_1 \cap X_2 \cap \cdots \cap X_n) \tag{3}$$

  全ての説明変数の値が既知であれば，学習セットから求めた条件付き確率の式をもとに，目的変数がとりうる値の確率を求めることができる．その確率が最も高い値を，

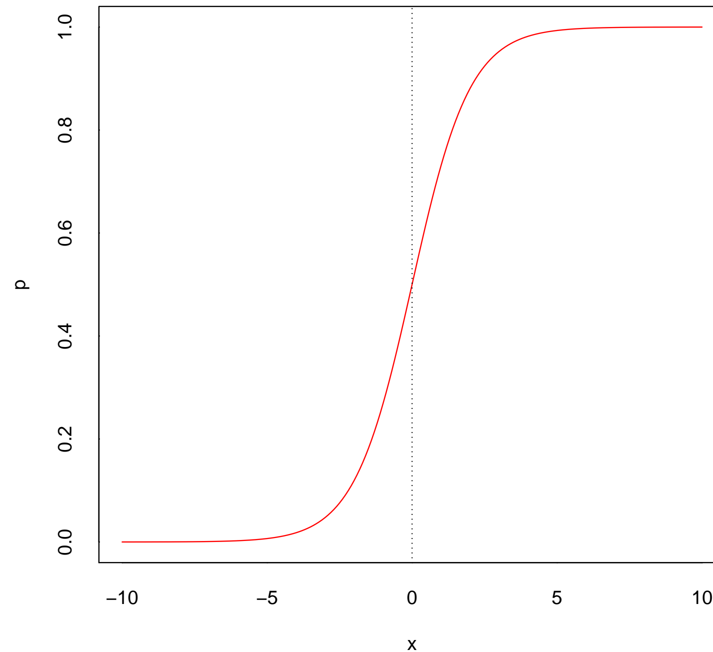


図 5: ロジスティック回帰のグラフ

目的変数の値とするのが，ベイジアンネットの基本的な処理である．

ベイジアンネットの簡単な例を図 6 に示す．図の例では，学習セットとして，物理，英語，数学のテストの合否が与えられている．このようなデータを用いて，物理と英語の結果から数学の結果を予測したいとする．まず，学習セットから物理と数学の間の条件付き確率を，英語と数学の間の条件付き確率を計算する．そして，各変数を表す頂点を辺でつなぐと図のようなグラフになる．条件付き確率をみると，物理のテストが合格の時は必ず数学のテストが合格であることがわかる．このことから，もし物理のテストが合格であれば，高い確率で数学のテストも合格であることがわかる．

2.4.2　変数選択

一般的に，機械学習においては，複数の説明変数を使用して，目的変数の予測モデルを構築する．しかし，説明変数が多いほど予測性能が高いとは必ずしも言えず，説明変数が予測性能に悪影響を与える場合もある [7]．そのため，予測モデルの構築前に変数選択を行い，予測に有用だと思われる説明変数のみを選別する必要がある．

変数選択は，全ての説明変数を含む集合に対して，予測性能が高くなるような説明変数の

学習データ

	物理	英語	数学
A	合格	合格	合格
B	合格	不合格	合格
C	不合格	合格	不合格
D	不合格	不合格	不合格

物理と数学の条件付き確率

P(数学 = 合格 | 物理 = 合格) = 1.0
P(数学 = 合格 | 物理 = 不合格) = 0.0
P(数学 = 不合格| 物理 = 合格) = 0.0
P(数学 = 不合格| 物理 = 不合格) = 1.0

英語と数学の条件付き確率

P(数学 = 合格 | 英語 = 合格) = 0.5
P(数学 = 合格 |英語= 不合格) = 0.5
P(数学 = 不合格|英語= 合格) = 0.5
P(数学 = 不合格|英語= 不合格) = 0.5

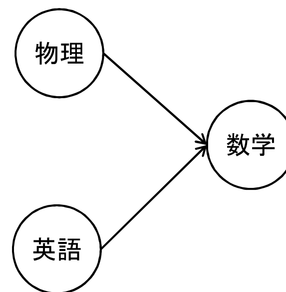


図 6: ベイジアンネットの例

部分集合を求める問題である．全ての部分集合に対してモデルの構築を行うことによって，最も予測性能が高くなる説明変数の集合を求めることができるが，説明変数の数が多い場合，部分集合の数は膨大となるため現実的ではない．そのため，経験則を用いた変数選択アルゴリズムがいくつか提案されている．変数選択アルゴリズムでは，説明変数に順位付けすることによって変数を選択する手法や，説明変数を 1 つずつ追加 (または削除) して部分集合を作成し，各部分集合の中で最も評価値が高い集合を選択する手法などがある．本節では，変数選択アルゴリズムの例として，Correlation-based Feature Selection，RelieF，Wrapper Subset Evaluation の 3 つについて説明する．

Correlation-based Feature Selection

Correlation-based Feature Selection は，説明変数の部分集合を，変数間の相関に基づいて評価するアルゴリズムである．この手法では，説明変数の部分集合について，集合内の各説明変数が目的変数との強い相関を持ち，集合内の説明変数同士の相関が弱いほど予測性能が向上するという経験則に従い，変数選択を行う．具体的には，説明変数の部分集合に対して，以下の $Merit$ という数値を計算し，この値が高い部分集合を結果とする．式において，対象の部分集合を S，部分集合の要素数を K，$\overline{r_{cf}}$ を目的変数と説明変数の相関の平均値，$\overline{r_{ff}}$ を目的変数同士の相関の平均値とする．

$$Merit_s = \frac{K\overline{r_{cf}}}{\sqrt{K + K(K-1)\overline{r_{ff}}}} \tag{4}$$

RelieF

RelieF は，ランダムで選択した要素とその最近傍の要素との間で，説明変数の値を比較することによって，予測に有効な説明変数を特定する手法である．このアルゴリズムは，目的変数が同じ場合は近い値を持ち，目的変数が異なる場合は遠い値を持つような説明変数が予測に有用であるという考えに基づいたアルゴリズムである．具体的な手順としては，まずデータセット中からランダムで 1 つの要素を選択する．そして，その要素と目的変数の値が同じである，かつ，説明変数の距離が最も近い要素と，目的変数の値が異なり，かつ，説明変数の距離が最も近い要素を取得する．そして，それらの要素の間で，説明変数の値がどのように違うかを比較し，有用な説明変数を特定する．この手順を指定された回数行い，最終的にどの変数が予測に有用であるかを求める．RelieF は，予測に悪影響を及ぼすような説明変数 (ノイズ) が多数あったとしても，有効に働くという特徴がある．

Wrapper Subset Evaluation

Wrapper Subset Evaluation は，説明変数の部分集合に対して，実際に予測モデルの構築と評価を行うことによって変数の選択を行う手法である．この手法では，実際に予測モデルの構築を行うため，どの予測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，予測モデルの構築と評価を行うことによって，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良い結果を出力することが多いが，計算コストが高く，他のアルゴリズムに比べて実行に時間がかかる．

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 [7]. Hall らは，データ数や説明変数の数が異なるデータセットを 15 個準備し，それらのデータセットに対して異なる変数選択アルゴリズムを適用した結果，予測性能がどのように変化するか調べている．また，変数選択アルゴリズムの優劣を比較するため，全ての変数選択アルゴリズムの組に対して，それぞれの変数選択アルゴリズムを適用した後の予測結果を比較している．この比較では，予測結果に有意差が存在する場合に，予測結果の良い方を Win，悪い方を Lose として，各変数選択アルゴリズムの Win の総数と Lose の総数の差を求めて，それをもとに優劣を決定している．この研究の結果，どのようなデータセットに対しても必ず良い結果を出す手法は存在せず，説明変数の特徴や最終的に使用する予測モデルによって，最も良い結果を出すアルゴリズムが変化するという結果が示さ

れている．例えば，予測モデルがベイジアンネットの場合は，Wrapper Subset Evaluation を使用した時がモデルの予測結果が最も良く，予測モデルが決定木の場合は，ReliF が最も良い予測結果となっている．また，最終的に選択される説明変数の数という観点でみると，Correlation-based Feature Selection が最も優れているなど，目的によって使用するアルゴリズムを選択する必要があると述べられている．前述した，Win の総数と Lose の総数の差でみると，Wrapper Subset Evaluation が 35 と最も高く，その他のアルゴリズムは 14 から-73 となっている．このように，Win の総数と Lose の総数の差では，Wrapper Subset Evaluation と他のアルゴリズムで大きな差があり，多くの場合は Wrapper Subset Evaluation が良い結果となることが示されている．

2.4.3 モデルの評価方法

機械学習で構築したモデルの予測性能の評価には，評価用のデータセットを用いて評価を行うことが多い．評価用データセットとして，目的変数が既知のものを用いることで，予測モデルを用いた予測結果が正しいものであるか確認することができる．モデルの構築と評価を合わせて行うために，データセットを学習セットと評価セットに分割して，学習セットを用いてモデルの構築を行い，評価セットを用いてそのモデルの評価を行うという手法がある．ここでは，代表的な評価手法として，k-fold 交差検定について説明する．

k-fold 交差検定は，用意されたデータセットを k 個のブロックに分割し，そのうち k-1 個を学習セット，残りの 1 個を評価セットとして評価を行う手法である．図 7 は，k-fold 交差検定 (k = 10) の例である．図 7 の例を用いて，k-fold 交差検定の具体的な手順について説明する．分割後のブロックをブロック 1〜ブロック 10 とすると，まず，ブロック 2 からブロック 10 を学習セットとしてモデルの構築を行い，ブロック 1 を評価セットとしてモデルの評価を行う．この時モデルの評価値を e_1 とする．そして，次は，ブロック 2 を評価セット，残りのブロックを学習セットとして，同様にモデルの構築と評価を行う．このような処理を，評価に用いるセットを変更しながら 10 回繰り返す．最終的には，10 回繰り返して算出した評価値 e_1 から e_{10} の平均値をモデルの評価値とする．k-fold 交差検定では，評価セットに用いるブロックを変更しながら処理を繰り返すことで，ブロックの分割方法が評価に与える影響を少なくしている．

2.4.4 機械学習を用いたバグ予測

ソフトウェア中に存在するバグを特定するための研究で，機械学習が用いているものがある．

Nagappan らは，連続したコードの変更 (Change Burst) に関するメトリクスを提案し，そ

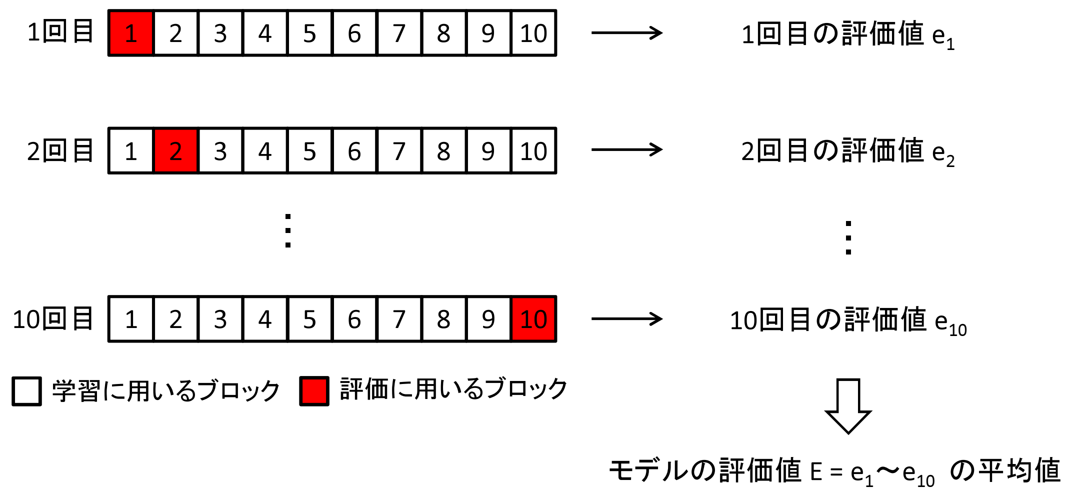


図 7: k-fold 交差検定の例 (k = 10)

れらを用いたバグの予測を行っている [16]．Nagappan らの研究では，バグ予測のために回帰モデルが用いられている．実験では，Windows Vista と Eclipse を対象にバグ予測を行っており，Windows Vista に対する結果では，Precision と Recall がともに 0.9 以上という高い予測性能を示している．

Lee らは，開発者の相互作用を表すメトリクスを用いて，バグが存在するファイルを特定する手法を提案している [11]．この手法では，開発者がタスクにかける時間や開発者があるファイルの作業に費やしているコストなどの 56 種類のメトリクスを提案し，それらを特徴量として機械学習アルゴリズム適用し，予測モデルの構築を行っている．実験の結果，既存のメトリクスを用いて構築したモデルよりも，Lee らが提案したメトリクスを用いた構築したモデルの方が予測性能が高いことが示されている．

このように，機械学習アルゴリズムは，バグ予測の分野では多く使用されており，リファクタリング対象の特定においても有用であると考えられる．本研究では，これらのバグ予測の研究を参考にして，機械学習を用いたメソッド抽出リファクタリング対象の特定を行う．

3　提案手法

2.1 節で述べたように，メソッド抽出は適用される頻度の高い重要なリファクタリングパターンである．しかし，ソフトウェア中には膨大な数のメソッドが存在し，メソッド抽出対象を手動で特定するためには多くのコストを要する．そこで，本手法では，実際にメソッド抽出が行われたメソッドを収集して，その特徴量を用いた機械学習によって，メソッド抽出対象を自動で特定するためのモデルを構築する．

また，2.2 節で述べたように，既存のリファクタリング対象の推薦手法では，開発者が自分の考えに合ったリファクタリング対象を得るためには，設定の変更と推薦結果の確認を繰り返し行わなければならないという問題点がある．提案手法では，機械学習を用いて実際にメソッド抽出が行われた事例をもとに対象の推薦を行うことで，開発者の考えに合ったメソッドを推薦することができる点や，開発者がツールの設定を変更する必要がないという利点がある．

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，メソッド抽出対象の特定のためのモデルを出力する．モデルの構築には，データマイニングツールである Weka[2] に実装されている機械学習アルゴリズムを用いている．以降，それぞれの手順について詳細に説明する．また，提案手法の概要を表した図を図 8 に示す．

3.1　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を用いる．リポジトリに藤原らのツールを適用すると，メソッド抽出の対象となったメソッド，メソッド抽出が行われたコミット，抽出前のコード片と抽出後のメソッドの本体との類似度などが出力される．本研究では，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 [26]．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にリビジョンの状態のものを取得する．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，

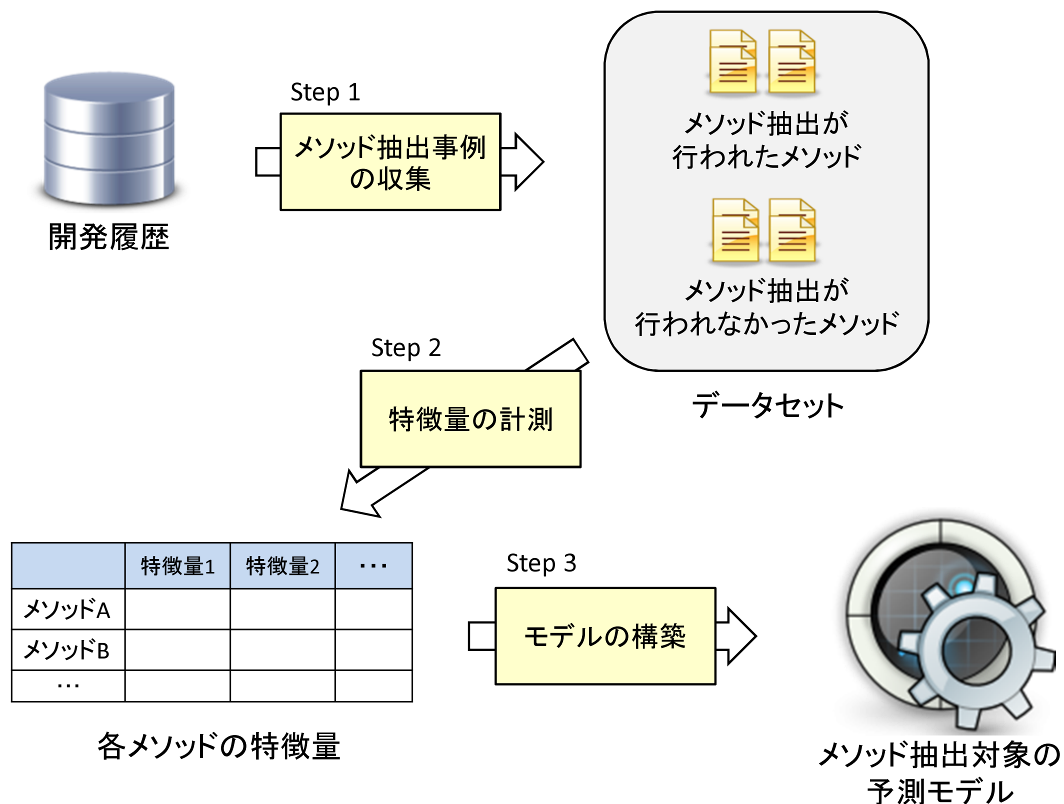


図 8: 提案手法の概要図

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で収集する．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，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ったメソッドを収集する．

3.2　Step 2：メソッドの特徴量の計測

本節では，提案手法で使用する特徴量について述べる．まず，準備として特徴量を計測するために必要な抽象構文木，プログラム依存グラフとプログラムスライスについて説明する．その後，提案手法で使用する特徴量について詳細に説明する．

3.2.1　準備

提案手法にて，特徴量の計測に使用している抽象構文木とプログラム依存グラフ，プログラムスライスについて説明する．

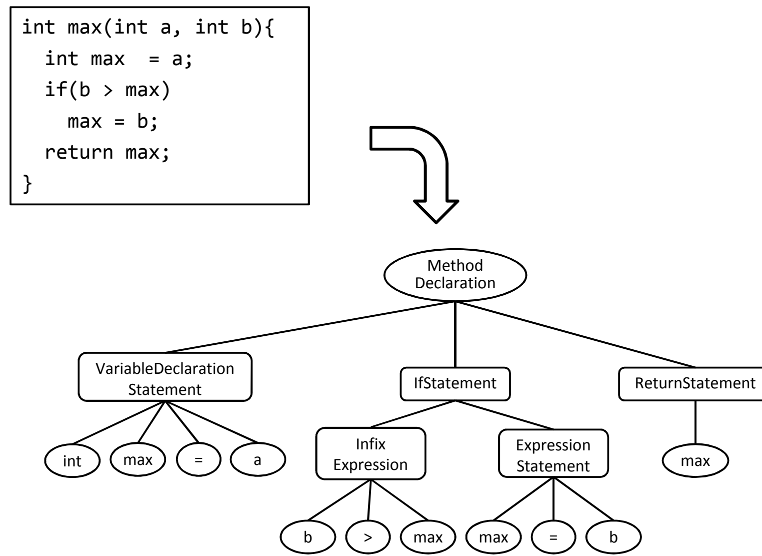


図 9: 抽象構文木の例

まず，抽象構文木について説明する．抽象構文木は，ソースコードの抽象構文 (式，文，識別子など) を木構造で表現したデータ形式であり，コンパイラなどで用いられる．統合開発環境 Eclipse には，抽象構文木を作成する機能があり，本研究ではこの機能を用いて抽象構文木を作成している．図 9 に，抽象構文木の例を示す．図 9 では，簡単のためいくつかの頂点を省略している．抽象構文木を用いることで，メソッド内のループの数やブロックの数など，構文的な情報を得ることができる．

次にプログラム依存グラフについて説明する．プログラム依存グラフとは，ソースコード中の各文を頂点，各文間の依存関係を有向辺で表したグラフである．依存関係にはデータ依存関係と制御依存関係があり，以下のように定義される．

データ依存関係

文 s_1 において変数 v が定義され，その定義が変更されることなく変数 v を参照する文 s_2 到達する実行経路が 1 つ以上存在する場合，$s1$ から $s2$ へデータ依存関係があるという．

制御依存関係

文 s_1 が繰り返し文または分岐文であり，s_1 の結果によって文 $s2$ が実行されるかどうかが直接決まる場合，s_1 から s_2 に制御依存関係があるという．

PDG の例を図 10 に示す．図 10 のプログラム依存グラフでは，通常の文を丸の頂点，制御文とメソッドの入口をひし形の頂点で表しており，中の数字はその頂点が表しているプログラム中の文の行番号である．一般的に，プログラム依存グラフではメソッドの入口を表す

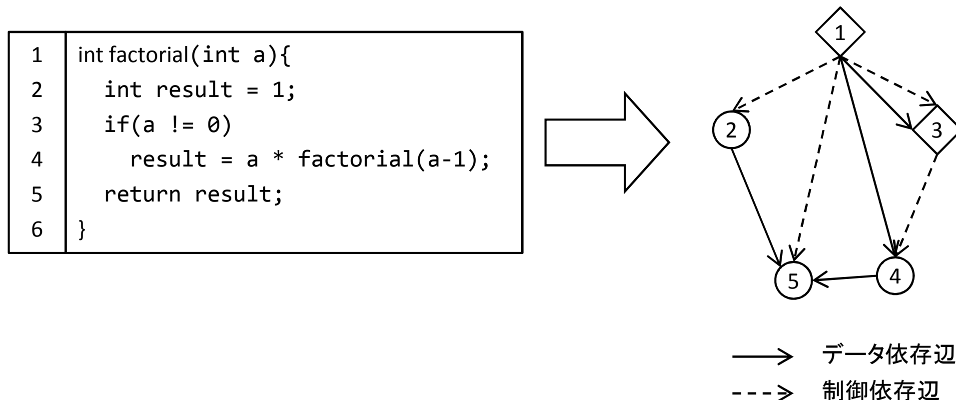


図 10: プログラム依存グラフの例

頂点が存在し，図中では 1 と書かれたひし形の頂点がそれに該当する．また，メソッドの入口を表す頂点からはメソッドの直下にある全ての頂点へ制御依存辺がひかれる．依存関係については，データ依存関係を実線の矢印，制御依存関係を破線の矢印でそれぞれ表している．本手法では，ソースコード解析ツール MASU[8] を用いてプログラム依存グラフを作成する．

プログラムスライスとは，お互いに依存関係にある文の集合のことである [24]．また，プログラム依存グラフを用いてプログラムスライスを求める手法を，プログラムスライシングという．プログラムスライシングでは，まず文と変数の組で表されるスライシング基準を定める．そして，スライシング基準に対応するプログラム依存グラフ上の頂点を開始点として，依存関係をたどることによって到達可能な頂点の集合を求める．このプログラムスライシングの結果の頂点の集合と対応する文の集合がプログラムスライスとなる．プログラムスライシングはプログラム依存グラフ上で開始点から依存関係をたどる方向によって前向きスライシングまたは後ろ向きスライシングと呼ばれる．また，前向きスライシングで得られた結果を前向きスライス，後ろ向きスライシングで得られた結果を後ろ向きスライスと呼ぶ．

3.2.2 特徴量の定義

本節では，本研究で使用する特徴量について，それぞれ定義と詳細な説明述べる．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．

- メソッドのサイズ

  メソッドのサイズを表す特徴量として，メソッド中の文の数 (NOS) を用いる．一般にサイズの大きいメソッドは可読性が低いため，適切なサイズに分割すべきとされている．また，Fowler も “Long Method” というコードの不吉な臭いを定義しており,，メソッドのサイズが大きいはリファクタリングすべきとしている [6]．メソッドのサイズの計測方法については，単純に行数を測るものや，空白行やコメントを除いて測る方法などが存在する．メソッド抽出はメソッド中の文に対して行われるリファクタリングであり，メソッド中のコメントや空白行の数とは大きな関連がないと思われる．そのため，本研究では，メソッドのサイズを，コメントと空白行を除去したメソッドの本体中の文数をとした．

- メソッドのシグネチャ

  メソッドのシグネチャに関連する特徴量は，メソッドの引数の数 (ARG)，返り値の有無 (RET)，メソッドのアクセスレベル (ACCESS) の 3 つである．引数が多いメソッドは，Fowler が定義したコードの不吉な臭いでは “Long Parameters List” とされ，よくない兆候であるといわれている [6]．メソッドの引数を減らすためのリファクタリングパターンは，引数オブジェクトの導入などであり，メソッド抽出は直接の関係はない．しかし，リファクタリングは同時に適用されることが多いと言われており [15]，他のリファクタリングと同時にメソッド抽出が適用される可能性がある．

  メソッドの返り値とアクセスレベルは，そのメソッドを変更した際に，他のソースコー

表 1: メソッドの特徴量

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	サイクロマチック数 (CYC)
凝集度	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	WMC，DIT，CBO，NOC，RFC，LCOM
コードクローン	コードクローンの有無 (CLONE)

ドにどの程度影響を与えるかを表している．リファクタリングの問題点の 1 つとして，リファクタリングによって新たなバグを混入させてしまうという点が挙げられる．このことから，他のソースコードに多くの影響を与える部分は，リファクタリングが行われにくいのではないかと考えられる．例えば，返り値のあるメソッドを変更すれば，その返り値を用いて処理を行っている部分に影響を与える可能性がある．またメソッドのアクセスレベルについても，アクセス可能範囲が広いメソッドほど，広範囲に影響を与える可能性が高いといえる．

- 複雑度

  メソッドの複雑度を表す特徴量として，サイクロマチック数 (CYC) を用いる．リファクタリングの目的は，保守性や可読性を向上させることであり，複雑なメソッドほどリファクタリングの対象になりやすいと考えられる．サイクロマチック数は Macabe によって提案されたメトリクスで，メソッドの制御構造を基にした複雑度である [13]．サイクロマチック数の計算には，メソッド内の逐次実行，分岐，繰り返しを表現した制御フローグラフが必要である．対象のメソッドを M，M の制御フローグラフの頂点数を N，辺数を E とすると，M のサイクロマチック数 $CYC(M)$ は以下の式で表される．

$$CYC(M) = E - N + 2 \tag{5}$$

- 凝集度

  凝集度とは，モジュールが機能的にまとまったものかどうかを表す度合である [21]．凝集度を計測するためのメトリクスは複数提案されているが，本研究では計測対象がメソッドであるため，メソッド単位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 [24] を用いる．スライスベースの凝集度メトリクスは，Weiser によって 5 つ提案されており，そのうち Tightness，Coverage，Overlap の 3 つがメソッドの凝集度を計測するのに有用であることを Ott らが実験によって示している [17]．本研究ではこれら 3 つの凝集度メトリクスを特徴量として使用する．それぞれのメトリクスの定義を以下に示す．式において，M をメソッド，$length(M)$ を M の文の数，V_o を M における出力変数の集合，SL_x を変数 x を基準にした後ろ向きスライス，SL_{int} を V_o 中の全変数に対する後ろ向きスライスの積集合とする．

$$Tightness(M) = \frac{|SL_{int}|}{length(M)}$$

$$Coverage(M) = \frac{1}{|V_o|} \sum_{x \in V_o} \frac{|SL_x|}{length(M)}$$

$$Overlap(M) = \frac{1}{|V_o|} \sum_{x \in V_o} \frac{|SL_{int}|}{|SL_x|}$$

メトリクスの計算に必要となるメソッドの出力変数 V_o 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 [22]．これらのメトリクスは，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．

- メソッドの構文

  メソッドの構文に関係した特徴量はループ数 (LOOP)，if 文数 (IF)，case 文数 (CASE)，ブロック数 (BLOCK)，ネストの深さ (NEST) の 5 つである．繰り返しや分岐は，その中が 1 つのまとまった処理で構成されている場合があり，その部分がメソッド抽出される可能性があると考えられる．そのため，1 つのメソッド内に繰り返しや分岐が多いほど，そのメソッド内に複数の処理が実装されており，メソッド抽出される可能性が高いと考えらえる．また，ブロックについても同様で，ブロック内の処理を 1 つのまとまった処理と考えてメソッド抽出が行われる可能性がある．ネストの深さについては，一般にネストが深いほど構文が複雑で可読性が低いと言われている．このことからネストが深いメソッドは，リファクタリングの対象になるのではないかと考えられる．

- メソッド内の変数

  メソッド内の変数に関する特徴量は，メソッドの内のローカル変数の数 (VAR) である．変数が多いメソッドは，内部が複雑であったり，複数の処理が実装されている可能性がある．そのようなメソッドはリファクタリング対象になりやすいと考えられる．

- CK メトリクス

  CK メトリクスは，Chidamber と Kemerer によって提案された，オブジェクト指向言語を対象にした複雑度メトリクスである [4]．CK メトリクスは全部で 6 種類あり，全てクラスを対象にしたメトリクスである．提案手法では，メソッド抽出が行われたメソッドが属するクラスを対象にして，CK メトリクスを計測する．以下，CK メトリクスのそれぞれの定義について述べる

WMC(Weighted Methods par Class)

WMC は，対象クラス内で定義されているメソッドの複雑度の総和である．WMC を計算するためには，メソッドの複雑度を計算する必要があり，本研究では前述したサイクロマチック数を各メソッドの複雑度として使用した．よって本研究における WMC は，対象クラス内で定義されているメソッドのサイクロマチック数の総和である．

DIT(Depth of Inheritance Tree)

DIT は，クラスの継承関係を木構造で表した時の，その木構造の中での対象クラスの深さである．多重継承が存在する場合は，木構造の根までの深さの最大値を DIT とする．

NOC(Number Of Children)

NOC は，対象クラスの直接の子クラスの数である．

CBO(Coupling Between Object classes)

CBO は，対象クラスが結合しているクラスの数である．あるクラス C_1 が，他のクラス C_2 のメソッドやインスタンス変数を参照しているとき，C_1 が C_2 に結合しているという．

RFC(Response For a Class)

RFC は，対象クラス内で定義されているメソッドとそれらのメソッドから呼び出されるメソッドの和である．

LCOM(Lack of COhesion in Methods)

LCOM は，対象クラス内で定義されているメソッドの組について，インスタンス変数を共有していない組の総数と共有している組の総数の差である．LCOM の計算式を以下に示す．式において，対象クラスを C，$M_1 \cdots M_n$ を C 内で定義されているメソッド，I_i を M_i で使用されている変数の集合，P をインスタンス変数を共有していないメソッドの組の集合 ($P = \{(I_i, I_j) | I_i \cap I_j = \phi\}$)，$Q$ をインスタンス変数を共有しているメソッドの組の集合 ($Q = \{(I_i, I_j) | I_i \cap I_j \neq \phi\}$) とする．

$$LCOM = \textbf{ if } (|P| - |Q|) > 0 \textbf{ then } |P| - |Q| \textbf{ else } 0 \tag{6}$$

- コードクローン

コードクローンとは，互いに一致または類似したコード片のことである．コードクローンがソフトウェア中に多く存在すると，ソフトウェアの保守性に悪影響を与える場合が

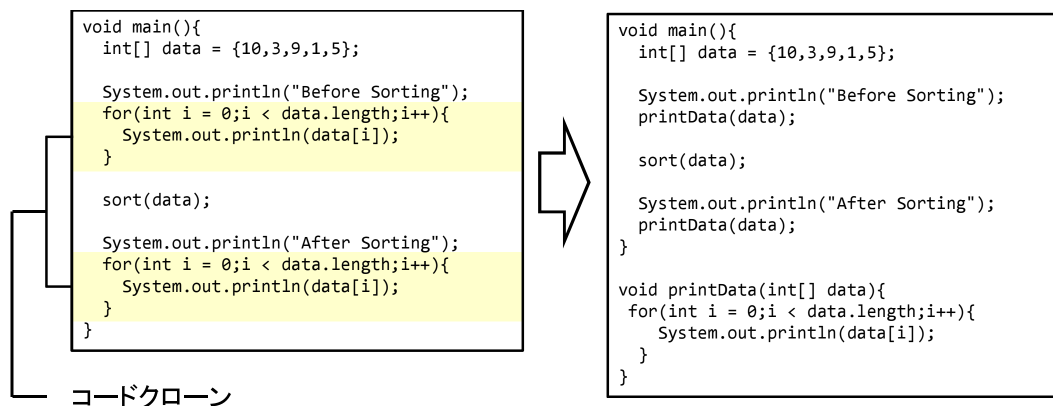


図 11: メソッド抽出によるコードクローンの集約

ある．例えば，あるコード片 A をバグが存在する場合，A とコードクローンとなっているコード片 B についても，バグ修正が必要である可能性がある．もし，コードクローンが多数存在する場合，バグ修正が必要な個所が大量に存在することになり，この作業は大きなコストとなる．また，文献 [6] で Fowler は，コードクローンを "Duplicated Code" として定義しており，リファクタリングを推奨している．

コードクローンによる問題に対する 1 つの解決策として，複数のコードクローンを 1 つに集約するという方法がある．この集約を行う際に，メソッド抽出リファクタリングが使用されることがある．図 11 はメソッド抽出によるコードクローンの集約の例である．この例では，データの表示部分のソースコードが完全に一致しており，その部分をメソッド抽出して，メソッド呼び出し文に置換することによって，コードクローンの集約を実現している．このように，メソッド抽出が行われるかどうかにコードクローンの有無が関連している可能性があるため，本研究の特徴量としてコードクローンの有無を使用した．コードクローンの検出には，高速なコードクローン検出ツールである CCFinder[10] を使用した．

3.3 Step 3：メソッド抽出対象の特定のためのモデル構築

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．2.4.2 節で述べたように，Hall らが行った評価では，Wrapper Subset Evaluation 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くなると述べられている [7]．そのため，本研究では Wrapper Subset Evaluation 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．

4　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．本章では，実験方法と実験結果，結果に基づいた考察について述べる．

4.1　実験方法

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 Java ファイル数と，メソッド数は最新リビジョンにおける数である．本節では，実験の手順と評価尺度について述べる．

4.1.1　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．2.3 節で述べたように，藤原らのリファクタリング検出ツールは類似度の閾値が 0.3 のとき，最も検出精度が良いとされているため，本実験では，藤原らのリファクタリング検出ツールを適用して得られた結果のうち類似度が 0.3 以上のものだけを用いる．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．そのため，実験用データセット中のメソッド抽出が行われたメソッドと行われたなかったメソッドの割合は 50%ずつである．

次に，データセット中の全メソッドに対して特徴量の計測を行う．特徴量の計測にあたっては，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[10] 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[8] や統合開発環境 Eclipse の機能を用いて作成した，特徴量を計測するためのツールを用いる．

最後に，データマイニングツールである Weka[2] に実装されている機械学習アルゴリズムを用いて，予測モデルの構築を行う．本実験では，予測モデルは決定木とロジステッィク回帰，ベイジアンネットの 3 種類を用いる．各予測モデルの構築の前には，変数選択アルゴリズムを用いて，有用な特徴量のみを選択する．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た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．

以上の手順を全ての対象ソフトウェアに対して行い，次節で述べる評価尺度を用いて評価する．

4.1.2 評価尺度

本節では，機械学習によって構築したモデルの評価尺度について述べる．

モデルの評価にあたっては，学習を行うためにデータセットと，評価を行うためのデータセットが必要になる．本実験では，2.4.3 節で述べた，k-fold 交差検定を用いて，データセットを分割する．ブロックの分割数である k は，weka のデフォルト設定である 10 を用いた．

次に，モデルの評価尺度について述べる．評価尺度は Precision，Recall，F 値の 3 種類の尺度を用いる．以降それぞれの尺度について説明する．

Precision は，適合率とも呼ばれ，予測結果のうちどの程度が正解であったかを評価する尺度である．本実験において 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，再現率とも呼ばれ，データセット中に存在する正解集合のうち，どの程度が正しく予測されてたかを評価する尺度である．本実験において 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．一般的に，Precision と Recall は，トレードオフの関係にあり，どちらかの評価値が増加するとどちらかが減少する傾向にある．そのため，Precision と Recall を総合的に評価するために，これらの値の調和平均である F 値が用いられる．

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．

$$Precision = \frac{|E \cap C_E|}{|C_E|} \tag{7}$$

$$Recall = \frac{|E \cap C_E|}{|E|} \tag{8}$$

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	Java 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

$$F = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (9)$$

k-fold 交差検定では，ランダムでブロックの分割を行うため，ブロックの分割の結果によって結果に差がでる場合がある．そのため，本実験では k-fold 交差検定を 100 回行い，それぞれ Precision と Recall，F 値を求め，それらの平均値を最終的な評価値とする．

4.2 実験結果と考察

4.2.1 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．表 3 の抽出有と抽出無は，それぞれメソッド抽出が行われたメソッドと，行わなかったメソッドを意味しており，括弧内の数はデータセット中の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数である．

これらのデータセットを用いて，ロジスティック回帰，決定木，ベイジアンネットの 3 種類のモデルの構築を行った．実験方法の節で述べたように，各モデルについて，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 50%，40%，30%，20%，10%と変化させながら，5 回ずつモデルの構築と評価を行った．

Precision と Recall，F 値について，結果を表したグラフを図 12，13，14 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

表 3: メソッド抽出リファクタリングの検出結果

ソフトウェア名	メソッド抽出事例数	データセットのメソッド数 (抽出有：抽出無)
Ant	766	1532(766:766)
ArgoUML	740	1480(740:740)
jEdit	502	1004(502:502)
jFreeChart	90	180(90:90)
Mylyn	490	980(490:490)

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．

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．

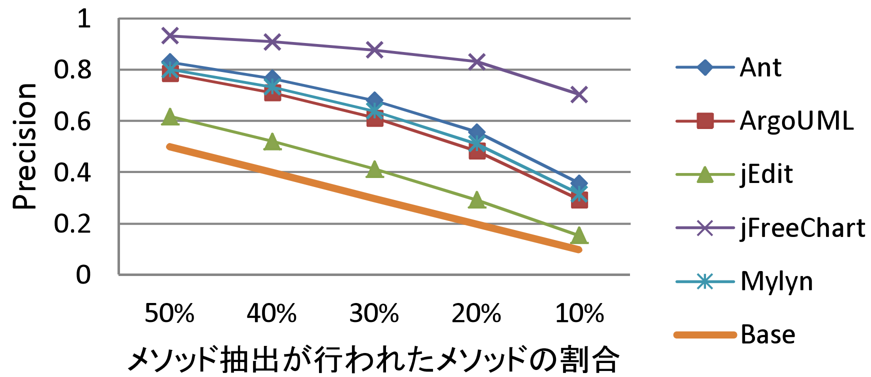
次に，変数選択の結果から，どの特徴量が予測に有用であったかを調べた結果を示す．表 4 は，変数選択によって各特徴量が選択された回数を示している．表 4 では，各ソフトウェアごとの特徴量が選択された回数と，それらの合計値を示しており，合計回数で降順に並び替えている．本手法で変数選択に用いている Wrapper Subset Evaluation は，各モデルごとに変数選択を行う手法である．そのため，各ソフトウェアについて 3 回変数選択を行っており，各ソフトウェアにおける選択回数の最大値は 3，合計の選択回数の最大値は 15 となる．

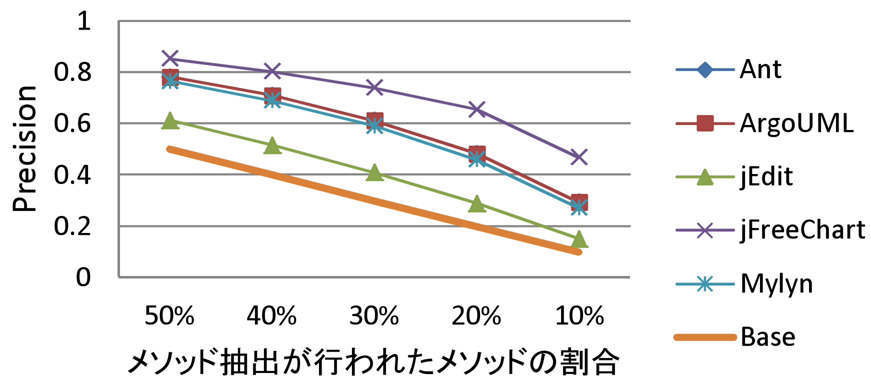
表 4 の結果から，本実験においてはメソッドの文の数である NOS と，スライスベースの凝集度メトリクスである Coverage が最も選択された特徴量であることがわかった．各ソフトウェアにおける結果を比較すると，対象ソフトウェアによって有用な特徴量が異なることがわかる．例えば，ブロック数を表す BLOCK は，Ant と ArgoUML，jFreeChart では全てのモデルにおいて選択されているが，jEdit では 1 度しか選択されていない．

4.2.2 考察

本節では 4.2.1 節で示した結果をもとに，考察と妥当性への脅威について述べる．

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，





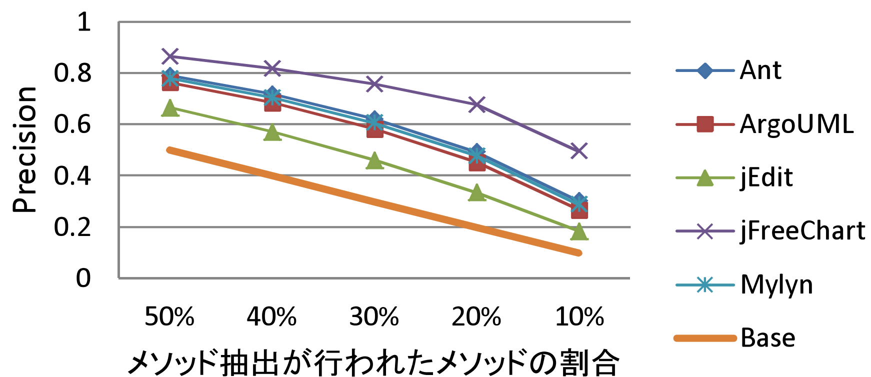
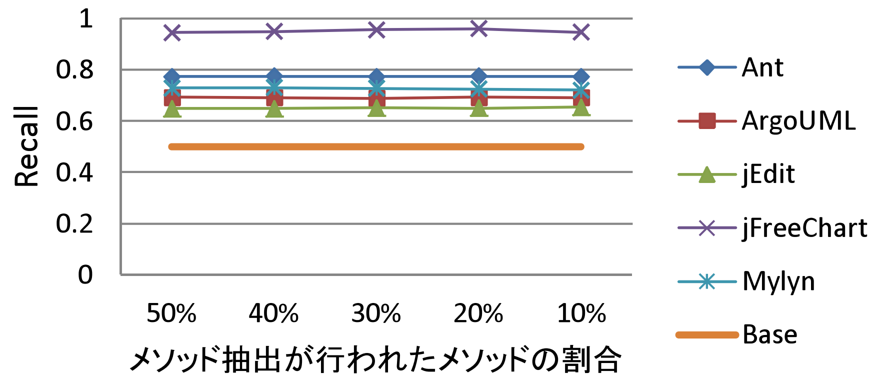
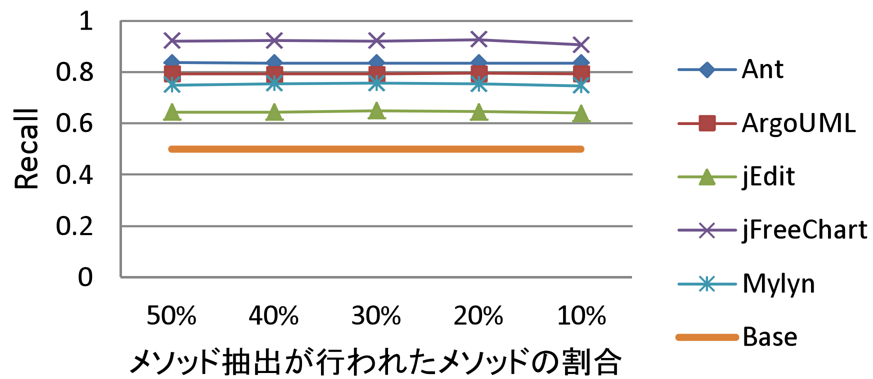


図 12: 評価結果 Precision





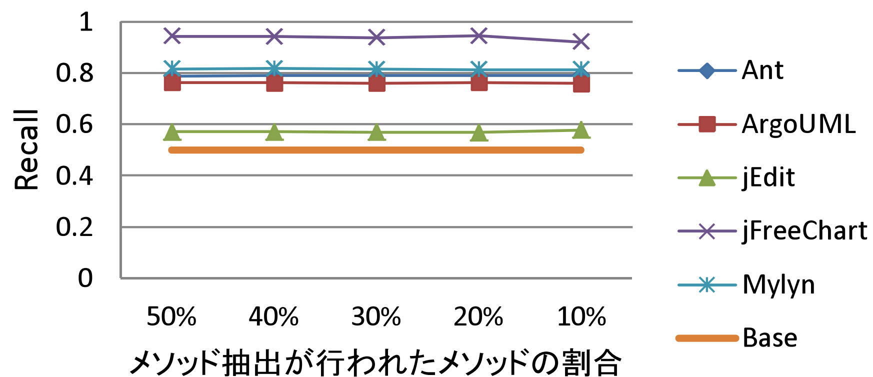
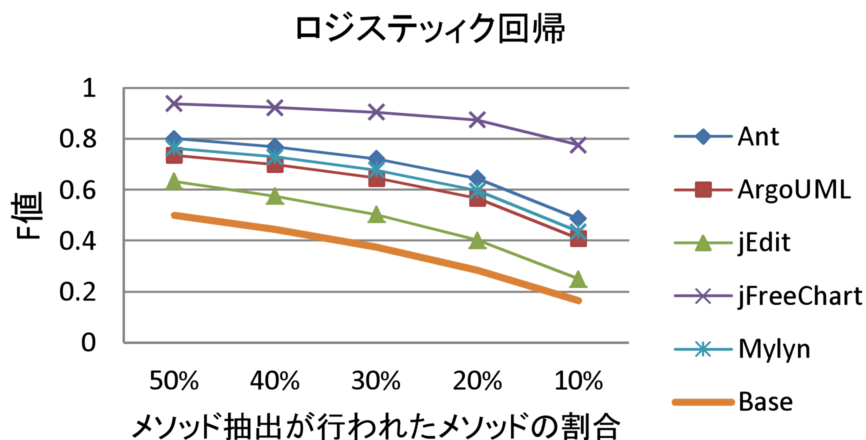
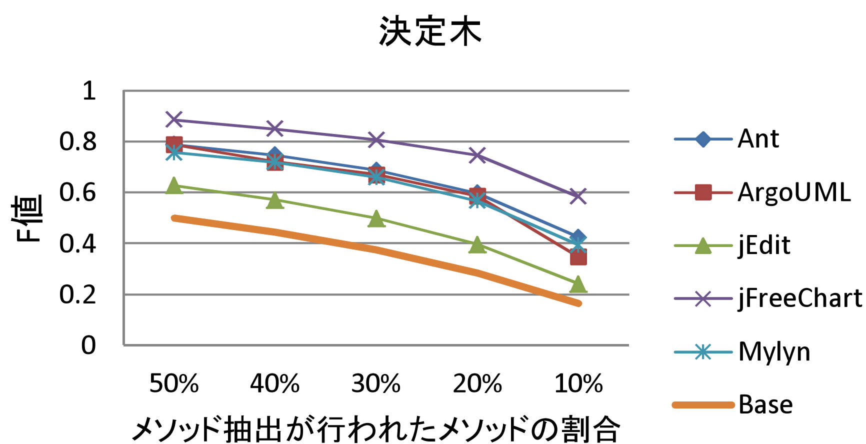


図 13: 評価結果 Recall





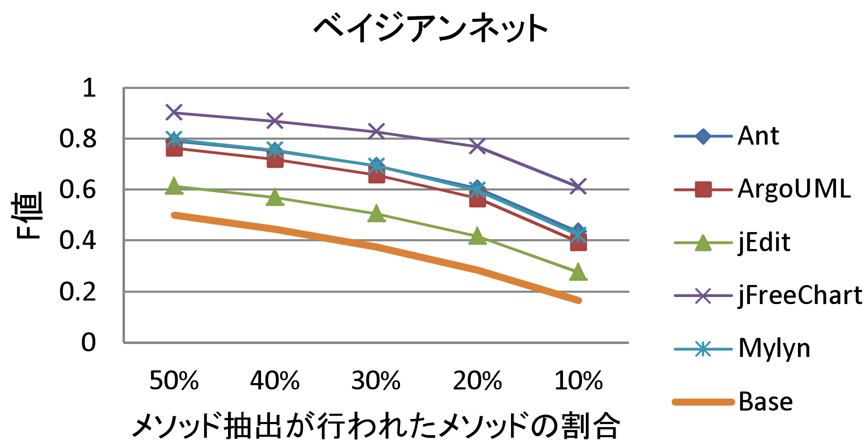


図 14: 評価結果 F 値

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 12，13，14 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

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
CYC	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

ともなって低下していることがわかる．Precision の値が低いことは，モデルによってメソッド抽出対象であると判断されたメソッドに，実際には抽出が行われなったメソッドが多く含まれていたことを示している．リファクタリング検出ツールの結果から判断すると，実際のソフトウェア開発においては，メソッド抽出が行われるメソッドは 10%より小さいと考えれる．そのため，実際に開発者の支援を行うためには，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．Overlap は，メソッド内の全ての出力変数に関連した文の数と，ある 1 つの出力変数に関連した文の数の比の平均を表している．Overlap は各出力変数に関する処理の重複を表しており，その値が低い場合は，出力変数ごとに別々のメソッドに分割するためにメソッド抽出が行われると考えられるが，本実験では有用ではない特徴量という結果となった．この理由として考えられるのは，Overlap の値が 0 か 1 のメソッドが非常の多いという点である．Overlap の値は，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．そのため，値が分布が極端になりやすく，予測に有用ではなかったのではないかと考えられる．CASE は，同じ分岐処理である IF が多く選択されているにも関わらず，選択された回数が少なかった．これについては，ほとんどのメソッドにで CASE の値が 0 であったこと，すなわち case 文が使われていなかったことが原因であると考えらえる．オブジェクト指向における switch 文と case 文を用いた分岐について，Folwer はコードの不吉な臭いの 1 つとしており，ポリモーフィズムを用いて置換することを推奨している [6]．そのため，今回対象とした Java 言語で記述されたソフトウェア中には，case 文がほとんど存在しなかったと考えらえる．

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．

4.3 妥当性への脅威

本節では，本実験における妥当性への脅威について述べる．

まず，使用したリファクタリング検出ツールについてである．本手法では，藤原らのリファクタリング検出ツールが出力した結果のうち類似度の閾値が 0.3 以上のものについて，メソッド抽出が行われたと判断して利用している．そのため，藤原らのツールの出力結果に誤検出が多く含まれている場合や，開発履歴から検出ができなかったメソッド抽出事例が多く存在すると，実験結果に悪影響を与える可能性がある．ただし，藤原らがツールの検出結果を目視で確認した結果，Presicion が 0.96，Recall が 0.86 と，他のリファクタリング検出ツールと比べて高い精度であることが示されており [26]，藤原らのツールの検出結果による本実験への大きな影響はないと考えられる．

次に，実験対象についてである．本研究では，実験対象として 5 つのオープンソースのソフトウェアを使用したが，今回の実験結果が対象ソフトウェアに限定されたものである可能性がある．特に，本実験で最も良い結果となった jFreeChart については，他の対象ソフトウェアに比べてリビジョン数が少ない，検出されたメソッド抽出事例数も少なかった．そのため，今後様々な規模のソフトウェアに対して実験を行い，結果がどのように変化するか確認する必要がある．

最後に，本研究で使用した特徴量について述べる．本研究では，メソッド抽出が行われるかどうかを判別するための特徴量として，メソッドとクラスを対象にした特徴量を 21 種類用いた．今回使用した特徴量以外にも，メソッド抽出の対象であるかの予測に有用なものが存在する可能性があるため．そのような特徴量を特定することができれば，より予測の精度を向上することができると考えられる．ただし，本研究では，リファクタリングに関連すると思われるサイズや複雑度，凝集度といった特徴量をなるべく多く選び実験に用いた．また，実験結果では，全ての結果がランダムで分類を行うベースラインより優れていたため，特徴量による学習の効果があったことを示している．

5　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，開発履歴からメソッド抽出が行われた事例を収集し，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た．そして，計測した特徴量を用いて，メソッドがリファクタリングの対象であるかを判別するための予測モデルを構築した．予測モデルは，ロジスティック回帰と決定木，ベイジアンネットの 3 種類を使用した．

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，全てのモデルにおいて，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となった．同様に，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も，ベースラインを上回る結果となった．また，変数選択アルゴリズムの適用結果をもとに，メソッド抽出対象を特定するために有用な特徴量を調査した．本研究の結果からは，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，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によって改善することができると考えられる．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については，もし異なるソフトウェア間で学習と評価を行っても予測性能に変化がないのであれば，1 度学習して構築したモデルを，他のソフトウェアでも利用できることになる．このように，実験によってモデルの一般性を示すことができれば，ソフトウェア開発において提案手法がより有効であることを示すことができる．

謝辞

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 井上 克郎 教授には，研究科長の職務でご多忙の中，本研究のご指導に多くの時間を割いて頂きました．井上 教授に適切なご指導やご助言を頂いたことで，本論文を完成させることができました．また，博士前期課程在学中に，国際会議での発表を初め多くのことに挑戦する機会を頂きました．たくさんの貴重な経験させて頂くことで，研究者としての素養や専門知識を得ることができただけではなく，多くの面で成長することができました．井上 教授のもとで多くのことを学ばせて頂いたこと，3 年間の研究生活を送らせて頂き本論文を執筆できたことに厚く御礼申し上げます．

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 松下 誠 准教授には，本研究の各段階において多くのご助言を頂きました．特に，研究の中間段階においては，本研究の方向性について多くのご意見を頂き，それらをご参考にさせて頂くことで本研究を遂行することができました．多くのご指導を頂きました 松下 准教授に心から御礼申し上げます．

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 石尾 隆 助教には，本研究の方向性や問題点などに関して，多くのご意見を頂きました．また，本研究についてだけではなく，専門知識に関して様々なご教授を頂き，ソフトウェア工学に止まらず多くのことを学ばさせて頂きました．本論文ならびに研究生活において，常に多くのご意見を賜りました石尾 助教に心から御礼申し上げます．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 吉田 則裕 助教には，私が研究室に配属されてから 3 年間の間，研究に関する直接のご指導を賜りました．常に学生のことを考え，ご指導下さった 吉田 助教のもとで研究をさせて頂いたことで，多くのことを考え，学び，成長することができました．ソフトウェア工学に関する知識がなかった私が，研究テーマを自ら考え，本論文を執筆できたことは 吉田 助教のご指導によるものであると確信しております．吉田 助教のご指導のもとで本論文を完成させることができたことに，心から御礼申し上げます．

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 崔 恩瀞 氏には，研究内容ならびに研究室生活において多くの場面で支えて頂きました．また，新日鉄住金ソリューションズ 井岡 正和 氏には，在学中に多くのご相談に乗って頂きました．両氏とともに，研究室生活を送ることができたことに，心から感謝致します．

奈良先端科学技術大学院大学情報科学研究科ソフトウェア設計学講座 藤原 賢二 氏には，本研究のためにツールを提供して頂きました．また，ツールの使用方法だけでなく，研究内容についてもご相談に乗って頂きました．藤原 賢二 氏の多くの有益なご助言に，深く感謝致します．

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井上研究室，ならびに楠本研究室の皆様には，公私ともに支えて頂きました．私が 3 年間の充実した研究室生活を送ることができたのは，皆様のおかげです．特に，同期生の友人方には，研究が上手くいかず苦しい時を初め，様々な場面で支えて頂きました．このように周囲の方々に恵まれた環境で，研究室生活を過ごすことができたことに，心から感謝致します．

最後に，今日まで 24 年間支えてくださった両親と，8 年もの間共に連れ添い支えてくれた彼女に深く感謝致します．

参考文献

[1] JDeodorant. http://www.jdeodorant.com/.

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

[3] A. Z. Broder. On the resemblance and containment of documents. In *Proc. of SEQUENCES*, pp. 21–29, 1997.

[4] S. R. Chidamber and C.F. Kemerer. A metrics suite for object oriented design. *IEEE Trans. Softw. Eng.*, Vol. 20, No. 6, pp. 476–493, 1994.

[5] E. V. Emden and L. Moonen. Java quality assurance by detecting code smells. In *Proc. of WCRE*, pp. 97–106, 2002.

[6] M. Fowler. *Refactoring: Improving the Design of Existing Code.* Addison Wesley, 1999.

[7] M. A. Hall and G. Holmes. Benchmarking attribute selection techniques for discrete class data mining. *IEEE Trans. on Knowl. and Data Eng.*, Vol. 15, No. 6, pp. 1437–1447, 2003.

[8] Y. Higo, A. Saitoh, G. Yamada, T. Miyake, S. Kusumoto, and K. Inoue. A pluggable tool for measuring software metrics from source code. In *Proc. of IWSM-MENSURA*, pp. 3–12, 2011.

[9] K. Hotta, Y. Higo, and S. Kusumoto. Identifying, tailoring, and suggesting form template method refactoring opportunities with program dependence graph. In *Proc. of CSMR*, pp. 53–62, 3 2012.

[10] T. Kamiya, S. Kusumoto, and K. Inoue. 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.

[11] T. Lee, J. Nam, D. Han, S. Kim, and I. P. Hoh. Micro interaction metrics for defect prediction. In *Proc. of ESEC/FSE*, pp. 311–321, 2011.

[12] Y. Y. Lee, N. Chen, and R. E. Johnson. Drag-and-drop refactoring: intuitive and efficient program transformation. In *Proc. of ICSE*, pp. 23–32, 2013.

[13] T. J. McCabe. A complexity measure. *IEEE Trans. Softw. Eng.*, Vol. 2, No. 4, pp. 308–320, 1976.

[14] E. Murphy-Hill and A. P. Black. Breaking the barriers to successful refactoring: observations and tools for extract method. In *Proc. of ICSE*, pp. 421–430, 2008.

[15] E. Murphy-Hill, C. Parnin, and A. P. Black. How we refactor, and how we know it. *IEEE Trans. on Softw. Eng.*, Vol. 38, No. 1, pp. 5–18, 2011.

[16] N. Nagappan, A. Zeller, T. Zimmermann, K. Herzig, and B. Murphy. Change bursts as defect predictors. In *Proc. of ISSRE*, pp. 309–318, 2010.

[17] L. M. Ott and J. J. Thuss. Slice based metrics for estimating cohesion. In *Proc. of METRICS*, pp. 71–81, 1993.

[18] K. Prete, N. Rachatasumrit, and M. Kim. Catalogue of template refactoring rules. Technical Report UTAUSTINECE-TR-041610, The University of Texas at Austin, 2010.

[19] K. Prete, N. Rachatasumrit, N. Sudan, and Miryung Kim. Template-based reconstruction of complex refactorings. In *Proc. of ICSM*, pp. 1–10, 2010.

[20] N. Rachatasumrit and M. Kim. An empirical investigation into the impact of refactoring on regression testing. In *Proc. of ICSM*, pp. 357–366, 2012.

[21] W. P. Stevens, G. J. Myers, and L. L. Constatine. Structured design. *IBM Syst. J.*, Vol. 13, No. 2, pp. 115–139, 1974.

[22] N. Tsantalis and A. Chatzigeorgiou. Identification of extract method refactoring opportunities for the decomposition of methods. *J. of Syst. and Soft.*, Vol. 84, No. 10, pp. 1757–1782, 2011.

[23] M. Vakilian, N. Chen, S. Negara, B. A. Rajkumar, B. P. Bailey, and R. E. Johnson. Use, disuse, and misuse of automated refactorings. In *Proc. of ICSE*, pp. 233–243, 2012.

[24] M. Weiser. Program slicing. In *Proc. of ICSE*, pp. 439–449, 1981.

[25] Z. Xing and E. Stroulia. Refactoring detection based on umldiff change-facts queries. In *Proc. of WCRE*, pp. 263–274, 2006.

[26] 藤原賢二, 吉田則裕, 飯田元. 構文情報を付加したリポジトリによるメソッド抽出リファクタリングの検出. 信学技報, Vol. 113, No. 24, pp. 19–24, 2013.

[27] 肥後芳樹, 楠本真二, 井上克郎. コードクローン検出とその関連技術. 電子情報通信学会論文誌, Vol. J91-D, No. 6, pp. 1465–1481, 2008.