卒　業　研　究　報　告　書

令和7年度

研究題目

ASTの差分情報を用いた受講者のfor文に対する苦手度の予測

指導教員　　上　野　秀　剛　　准教授

氏　　名　　谷　垣　歩　輝

令和8年1月27日　提出

奈良工業高等専門学校　　情報工学科

ASTの差分情報を用いた for文に対する苦手度の予測

上野研究室　谷垣歩輝

プログラミング教育においては，Online Judge System（OJS）を用いた自動採点型の演習が広く利用されているが，従来のOJSは提出プログラムの出力の正否のみを評価対象とするため，受講者がどの構文を理解していないかを把握することが困難である．その結果，受講者は自身の苦手構文を自覚しにくく，教員も個々の理解状況に応じた指導を行いにくいという課題がある．本研究ではOJSに蓄積された受講者の提出履歴を対象として，最終版ソースコードと提出過程のソースコード間の抽象構文木（AST）の差分情報から，for文に対する苦手度を予測する手法を提案する．AST差分をfor文内外の構造に基づいて60通りに場合分けし，各場合について誤修正数および平均修正量という2種類のメトリクスを算出することで，受講者の修正過程を定量化する．提案手法の評価の際は，得られた特徴量と主観評価に基づく苦手度指標を用いてデータセットを構築し，順序ロジスティック回帰およびOrdinal Random Forestにより苦手度の予測を行う．実験の結果，提案したメトリクスを用いることで，受講者のfor文に対する苦手度を一定の精度で推定できることを確認した．本手法により，受講者は自身の苦手構文を客観的に把握でき，教員は個々の受講者に応じた指導支援を行うことが可能になると期待される．

目次

1 はじめに

プログラミング講義の多くでは，座学の後にその内容を用いたプログラミング課題が学生に課される．講義や課題の提示に用いられるシステムの1つにOnline Judge System（OJS）が存在する．OJSは教員がプログラミング課題を設定すると，受講者が課題に取り組み，作成したソースコードをOJS上に提出する．提出されたソースコードは自動でコンパイル・実行され，教員が設定したテストケースによって採点される．テストケースと実行結果が一致するかにより正誤判定を行い，採点結果やコンパイル・実行時のエラーがフィードバックとして受講者に提示される．受講者は提出とソースコードの修正を採点結果が満点になるまで繰り返すことで，講義内容の定着を目指す．

これまでにOJSの設計開発に関する研究[1, 2]が行われている．Wenjuらはテストケースを利用した採点では評価できないプログラムの類似性や可読性を評価して教員に送信するOJSのフレームワークを提案している[1]．Huiらはプログラムの関数など一部のみの提出許可やポインタ，配列，再帰などの抽象概念理解を評価する問題設計が可能なシステムを提案している[2]．しかし，いずれのシステムも従来のOJSと同様に受講者のソースコード中の誤りの種類を識別できず，受講者の苦手とする文法についてのフィードバックを生成できない．そのため，受講者は自身がどの文法を理解していないかを把握できず，if文やfor文をはじめとした数ある文法の中から，自身の理解できていない文法を自力で探らなければならない．苦手文法を分析するには，その人のプログラミングの能力を客観的に評価する必要がある．これにはプログラミング文法に関する高度な理解が必要となるが，文法理解が進んでいない学生には難しい．また，教員も受講者の文法理解状況を把握できず，個人向けの苦手克服用の課題の設定が難しい．

本研究は受講者がOJS上に提出したソースコードの差分情報から，受講者の特定の構文への未理解の度合いを予測する手法を提案する．OJS上には1)100点（満点）と採点される最終版のソースコードと，2)100点に満たない提出過程のソースコードの2種類が存在する．最終版と提出過程のソースコードを抽象構文木に変換して差分情報を抽出することで，構文単位で修正箇所を特定し，受講者がどの構文で躓いているかの予測が可能となる．提案手法は差分情報を元に各構文要素に対する理解の低さを表す苦手度を算出し，OJSの評価結果と組み合わせたデータセットを用いた機械学習による予測を行う．提案手法によって構文単位の未理解の度合いが予測できれば，受講者は自身が苦手な文法を把握し，その文法に焦点を当てた効率的な復習や学習計画の立案が可能となる．また，教員は個々の受講者に対して苦手文法の練習問題を提供するなどの支援が可能となり，個人の苦手度に応じた学習環境の構築が可能となる．

提案手法の評価として，奈良高専3年生18名を対象として実施した理解度アン

ケートの結果から苦手度を予測し，提案手法による予測結果と比較を行う．本研究の制約として，提案手法が対象とする構文要素をfor文とする．for文は逐次・分岐・反復というプログラムの基本3要素をすべて含む構文である．そのため，3要素すべてに対する苦手度の影響を考慮した結果が得られると期待される．分析では以下の3つのRQを設定し，提案手法の有効性を評価する．

RQ1 初回提出時の間違い数と修正できなかった間違い数からfor文に対する苦手度を予測できるか？

RQ2 間違い数と提出回数からfor文に対する苦手度を予測できるか？

RQ3 上記の組み合わせからfor文に対する苦手度を予測できるか？

以下，2章では関連研究を述べ，本研究の立ち位置を説明する．3章では本研究に使用する理論と前提知識について説明する．4章では提案手法について詳しく説明する．5章で提案手法を実際のOJS上のソースコードに適用し，苦手度の予測を行い，6章で．結果と考察を述べる．7章で本研究のまとめを述べる．

2 関連研究

2.1 OJSを対象とした学習データ分析

これまでに，OJSを対象とした学習データ分析に関する研究は複数行われている[3, 4, 5]．青木らはOJSにおけるフィードバックの動的生成のため，OJS上の提出ソースコード間の差分構文木を自動的に抽出するツールを提案している[3]．奈良高専本科の講義における課題の提出履歴を用いた複数の分析例と共に，差分構文木から得られた情報を利用することでさらなる学習支援の実現ができる可能性について述べている．しかし，その具体的な手法については提案されていない．本研究では青木らの提案した手法を用いて差分構文木を抽出し，苦手度の予測に利用する．

槇原らは学習者が次に解くと成長につながりやすい課題を自動推薦するモデルを提案し，ユーザの解答履歴から求めた問題間の関係が，プログラミング学習に有効か調査した[4]．OJS上のある課題を終えたのちに，別の課題に取り組む流れを遷移として課題の遷移確率モデルを作成し，可視化した．その結果，遷移確率の高い課題群など，プログラミング教育における自学自習の際に利用可能な情報が得られた．鄭らは初心者のプログラミング技術の習得支援のため，部分点を導入したシステムを提案している[5]．抽象構文木を用いた正解プログラムとの構造比較による静的解析と，プログラム実行時の動作過程を正解プログラムの動作と比較する動的解析を行っている．その結果，正解プログラムと提出プログラムの差を表す編集距離が得られたが，その編集距離を質的評価にどう繋ぎ，部分点として評価するかについて今後の課題としている．以上2つの研究は本研究と同じくOJSから得られるデータを用いた研究だが，取り組むべき課題の導出[4]や部分点の導入[5]を目的としており，OJSのデータから受講者の苦手度を予測することを目的とした研究は行われていない．

2.2 差分分析による評価

差分を用いたソースコードの分析手法はソフトウェア工学の分野では複数提案されている[6, 7, 8, 9]．

福元らはテンプレートをベースとしたソースコードの自動修正手法を複数行にわたる変更パターンへ対応する手法として，同時変更(共変更)を考慮したソースコードの変更パターン抽出手法を提案している[6]．抽象構文木の差分から変更箇所を検出し，オープンソースソフトウェアの1つであるneo4jプロジェクトを対象に変更パターンを計測した結果，単一行変更パターンに加え，共変更パターンの存在を明らかにした．Hoanらはソフトウェア進化の過程においてコード変更がどの程度繰り返されるかを大規模に分析している[7]．抽象構文木で表現した連続するリビジョン間の差分を分析し，繰り返されるコード変更を機械学習の特

微量とすることは，自動プログラム修復や保守支援ツールにおいて有望であると結論づけている．以上2つの研究[6, 7]では，いずれも変更箇所の検出を目的としてASTの差分情報が用いられており，構文単位での分析を可能としていた．本研究でも同様に，ASTの差分情報を用いて変更箇所の検出，構文単位での分析を行うことで，受講者のある構文に対する苦手度の予測を行う．

入山らは後方互換性が破壊されるソースコードの変更を，抽象構文木の差分を利用して検出する手法を提案した[8]．8件のオープンソースソフトウェアを対象とした実験の結果，既存手法よりも多くのAPI変更を検出し，後方互換性の判別精度の向上も確認された．市井らはリファクタリング前後でプログラムの振舞いが保持されていることを静的解析で検証する手法の提案を目的とし，抽象構文木に基づくモデルを用いてプログラム構造の差分を検出している[9]．実装したツールの適用実験の結果，ある組込み製品にて実施されたリファクタリングのうち，56%を正しく判定できた．以上2つの研究[8, 9]においても，コードの変更箇所の検出にASTの差分情報が用いられており，リファクタリング前後のような類似性の高いコードもASTの差分から検出できる可能性が示されている．類似性の高いコードは，単なる文字列としての比較は難しく，意味のない空白や改行なども検出してしまう問題が存在する．本研究が対象とするコードは受講者の課題修正履歴であり，それらコード間には強い類似性が見られる場合があるが，ASTを用いることでこの問題は解決されると考えられる．

3 準備

3.1 OJS を用いたプログラミング講義

本研究が対象とするプログラミング講義は，教員がある単元に対し講義資料に基づいた座学と，単元に関連する内容の課題をOJS上に提示する形式で行われる．受講者は座学を受けた後に各自で課題に取り組み，ソースコードを任意のエディタで作成してOJS上に提出する．OJSは提出されたソースコードをコンパイル・実行し，事前に教員が用意した入力をプログラムに与える．入力に対する出力結果が教員が設定した出力と一致するか判定する．事前に用意された入力と，それに対応する正解となる出力の組をテストケースと呼ぶ．OJSは受講者が提出したプログラムに対して複数のテストケースを適用し，一致した割合に基づいて*score*を計算する．

$$score = \frac{\text{テストケース正解数}}{\text{テストケース数}} \times 100 \tag{3.1.1}$$

*score*の計算が終わると，受講者にフィードバックとして各テストケースの正誤，*score*，コンパイルエラーの有無，実行時エラーの有無が表示される．受講者はフィードバックをもとにソースコードを修正し，再提出するという動作を*score*が100になるまで繰り返す．*score*が100となるソースコードを提出することで課題が完了となり，それまでに提出したすべてのソースコードが提出日時，*score*とともにリビジョンとして記録される．本研究では，リビジョンとして記録された提出過程のソースコードから，受講者の苦手度を予測する．

3.2 抽象構文木(AST)

抽象構文木(AST: Abstract Syntax Tree)とは，ソースコードの構文情報を木構造で表現したものである．図1(a)にJavaソースコード，(b)にソースコードに対応するASTを示す．ASTは順序木であり，各ノードがプログラムの構文上の1つの要素に対応する．親ノードと枝で結ばれた子ノードは，親ノードの詳細情報を表す．表1に図1に存在する各ノードの意味とそれが表す構文情報の対応表を示す．例えば，図1(b)の赤下線で示した頂点”SingleName:b”は，図1(b)の4行目1文字目の要素に対応しており，ラベルSimple Nameが単純名（例の場合は変数名）を示し,変数名がbであることを表す．また，図1(b)の破線で示したノードは，図1(a)のfor文全体に対応しており，ラベルForStatementの構文情報が4つの子ノードVariableDeclarationExpression, InfixExpression, PostfixExpression, Blockが示す4要素からなることを示している．

本研究では，受講者が最終版のソースコードを作成するまでに修正した構文を抽出するために，ASTの差分を用いる．図2にある受講者の(a)最終版の直前のソースコードと(b)最終版のソースコードを示す．これらのソースコードは，1か

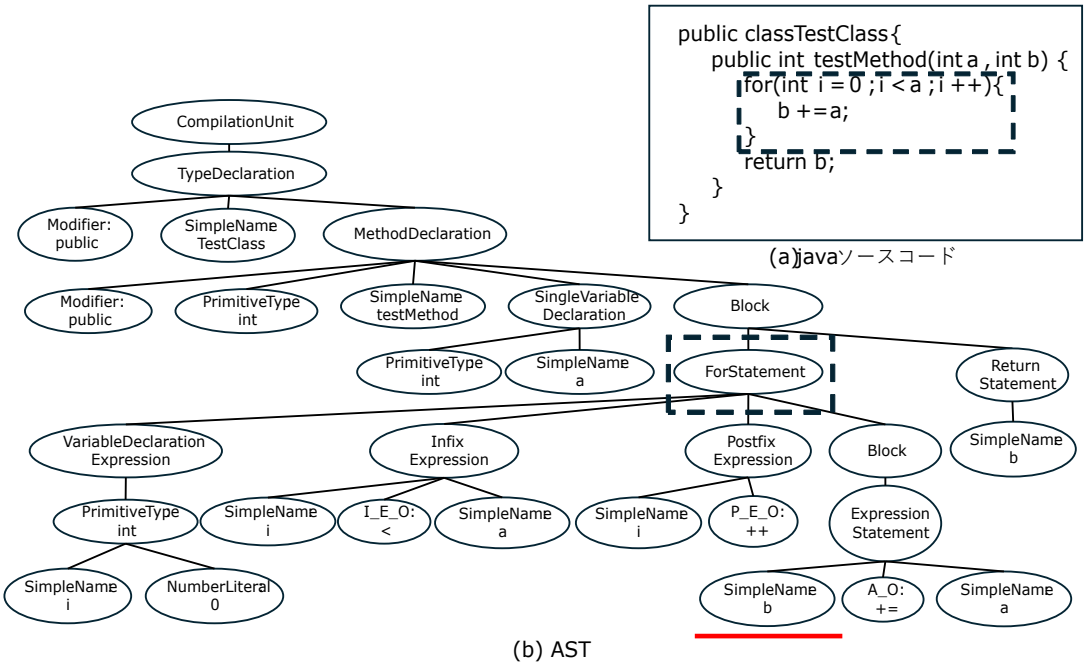


図 1 ソースコードと AST

```
public static void main ( String [ ] args ) {
   int b = 0;
   for ( int i = 1 ; i < 10 ;  i = 1 ) {
      b += i ;
   }
   System. out . println (b) ;
}
```

(a) 最終版の直前のソースコード

```
public static void main ( String [ ] args ) {
   int b = 0 ;
   for ( int i = 1 ; i <= 10;   i ++) {
      b +=i ;
   }
   System. out . println (b) ;
}
```

(b) 最終版のソースコード

図 2 1 から 10 の総和を出力する Java プログラム

ら 10 の総和を出力するプログラムを作成する課題に対応する．赤下線が 2 つのソースコードの差分を表している．(a) は繰り返し回数を間違えており，1 から 9 の総和が出力される．最終版の (b) では繰返しの継続条件式が修正されており，正しく 1 から 10 の総和が出力される．図 3 に 2 つのソースコードの赤下線部を AST で表現したものを示す．赤色のノードは (a) と (b) の異なる部分を示しており，演算子が変更されていることがわかる．また，親ノードを辿ると変更箇所が for 文に属していることがわかる．この赤色のノードのような最終版とそれ以外のソースコード間の違いを本論文では差分と呼ぶ．

AST の差分は受講者の修正箇所を構文単位で表し，受講者が「どこでつまずき，どこを試行錯誤したか」を直接的に反映する行動ログと言える．そのため，ある構文に関する差分が多い受講者は，その構文に内在する制約や意味を十分

表1ノードと構文情報の対応表

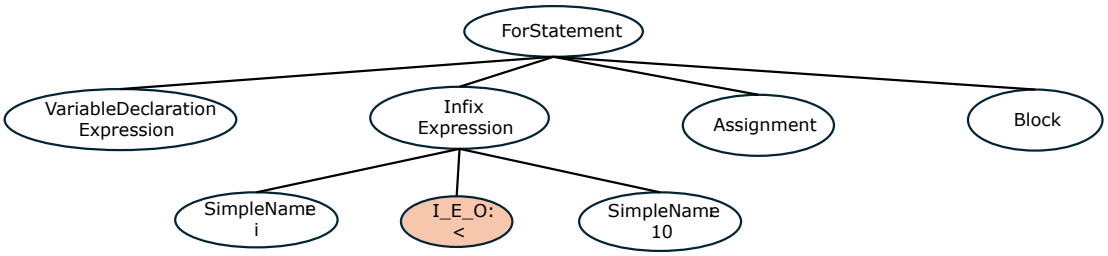
構文情報	概要
CompilationUnit	対象ソースコード全体
TypeDeclaration	型宣言
Modifier	修飾子
MethodDeclaration	method 宣言
PrimitiveType	基本データ型
SingleVariableDeclaration	変数
Block	method や Statement の範囲
ForStatement	for 文
ReturnStatement	return 文
VariableDeclarationExpression	変数宣言式
ExpressionStatement	式文
InfixExpression	中置演算式
PostfixExpression	後置演算式
ASSIGNMENT_OPERATOR	代入演算子
I_E_O(Infix_Expression_Operator)	中置演算子
P_E_O(Postfix_Expression_Operator)	後置演算子
SimpleName	単純名

に理解できていないために，正しい表現をスムーズに生成できなかったと考えられる．例えば「ForStatement」以下に差分が多数見られた場合は，for文の未理解に起因した間違いを多く生み出している可能性があるとわかる．したがって，OJS上のソースコードからfor文に関する差分を抽出すれば，受講者のfor文への未理解に起因するミスの修正過程を分析することが可能となる．

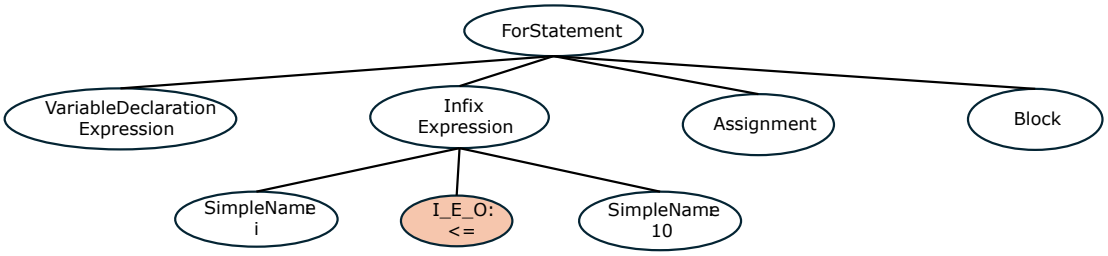
3.3 Leave-One-Out クロスバリデーション

Leave-One-Out（LOO）クロスバリデーションは，限られたデータ数においてモデルの汎化性能を評価するための交差検証手法の1つである．データセットに含まれるサンプル数をNとすると，各サンプルを一度ずつテストデータとして用い，残りのN－1個のサンプルを訓練データとしてモデルを学習する．この操作を全サンプルについて繰り返し，得られた評価結果の平均をモデル性能の指標とする．データセットが小規模である場合，単純なhold-out法による評価は訓練データ数の減少により学習の不安定化や評価結果の偏りを招く可能性がある．これに対し，LOOは各テストデータに対してN－1サンプルと可能な限り多くの訓練データを用いることができるため，小規模データセットにおいてもモデルがデータ分布の特徴を十分に反映した学習を行うことが可能である．

本研究では評価実験で用いるデータセットが小規模であるため，LOOを用い



(a) 最終版の直前のAST



(b) 最終版のAST

図 3 AST の差分例

ることで最大限のデータ量で学習を行い，学習データ不足による性能劣化を抑える．

3.4 順序ロジスティック回帰

順序ロジスティック回帰は，目的変数が順序尺度を持つカテゴリ変数である場合に用いられる回帰モデルである．そのため，大小関係は定義されるが，隣接カテゴリ間の間隔が等しいとは限らないデータを扱う場合に有効な回帰モデルである．

本研究の評価実験では，被験者の主観的な苦手度を4段階の離散的順序尺度で評価する．そのため，機械学習の際はその順序性を考慮する必要があり，機械学習アルゴリズムの1つとして順序ロジスティック回帰を採用する．

3.5 Ordinal Random Forest(ORF)

Ordinal Random Forest（ORF）は，順序尺度を持つカテゴリ変数を目的変数とする分類問題に対応するために拡張された Random Forest 手法である．通常の Random Forest はクラス間の順序関係を考慮せず，各クラスを独立なカテゴリとして扱う．これに対し，ORF はクラス間の順序情報を学習過程に組み込むことで，順序付き分類に適した予測を行う．

目的変数がK段階の順序カテゴリからなる場合，ORF では「クラスがK以下であるか否か」というK − 1個の二値問題を定義する．各二値問題に対して Random

Forestを学習し，得られた確率出力を統合することで，最終的なクラス予測を行う．また，拡張元のRandom Forestの特徴としてスパースデータにも比較的安定して適用できることが挙げられる．

評価実験で用いる被験者の主観的な苦手度は4段階の離散的順序尺度であり，強いスパース性が存在する．そのため，スパース性に強く，かつ順序尺度をもつカテゴリ変数を目的変数にできるORFを機械学習アルゴリズムの1つとして採用する．

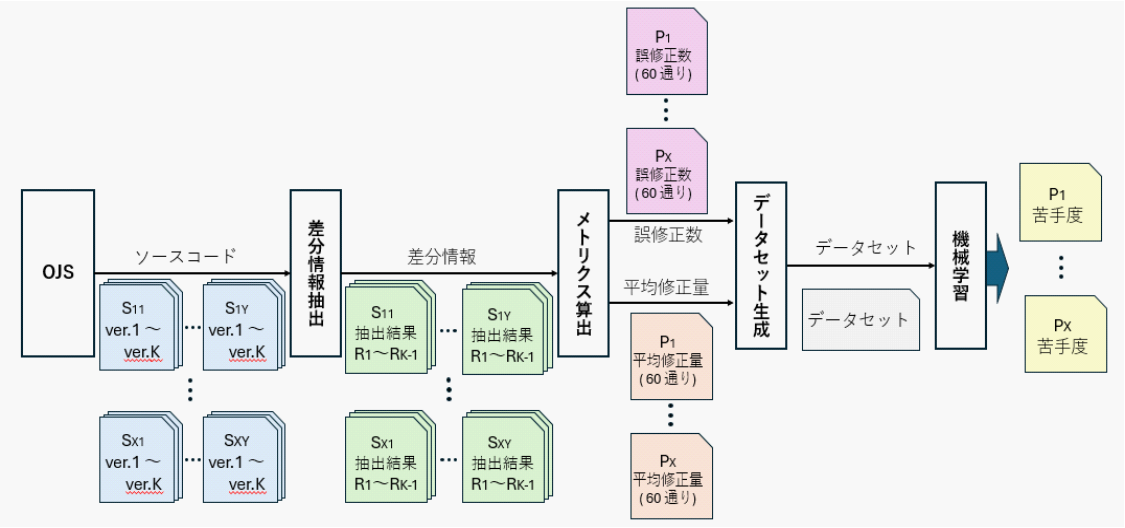


図 4 苦手度予測プロセス

4 提案手法

4.1 概要

提案手法はOJSに記録されたソースコードの提出履歴から受講者のfor文に関する苦手度を予測する．青木らの手法[3]により最終版とそれ以前のソースコード間のAST差分を取得し,「コード中の間違いに早く気づけたか」と「効率よくコード修正できたか」の2種類のメトリクスを算出する．算出したメトリクスから生成したデータセットを用いた機械学習で受講者の苦手度の予測を行うことを提案する．

図4にOJS上に提出されたソースコードから提案手法による受講者の苦手度を予測する際のプロセスを示す．長方形が処理，矢印がデータフローを表し，色付きの図形は受け渡されるソースコードやデータを表す．提案手法は受講者P_i(i = 0, 1, ..., X)が課題T_j(j = 0, 1, ..., Y)に対してOJS上に提出したソースコードS_{ij}を対象とする．ソースコードはそれぞれver.1から最終版(ver.K)までが存在し，すべてOJS上に記録されている．「差分情報抽出」はa)最終版Kと，b)最終版を除くすべてのコード1～(K-1)の間でASTによる差分情報を得る．ここで，あるソースコードS_{ij}から得られた差分情報を抽出結果$R_v(v = 1, 2, ..., K-1)$と呼ぶ．次に「メトリクス算出」は抽出結果R_vを元に2種類の60次元からなるメトリクスを算出する．「データセット生成」は2種類のメトリクス（計120次元）を次元圧縮したデータセットを作成し,「機械学習」によって苦手度の予測を行う．なお，次元圧縮には広く使われているPCAを用い，累積寄与率が0.8以上となる主成分のみを採用する．また，機械学習アルゴリズムには順序ロジスティック回帰とORFを用いる．4.2節で差分情報抽出，4.3節でメトリクス算出について詳細を説明する．

```
R 1
CompilationUnit/TypeDeclaration/MethodDeclaration/Block/ForStatement/VariableDeclarationExpression/VariableDeclarationFragment/NumberLiteral
diff word: 0

R 2
CompilationUnit/TypeDeclaration/MethodDeclaration/Block/ForStatement/VariableDeclarationExpression/VariableDeclarationFragment/NumberLiteral
diff word: 0
CompilationUnit/TypeDeclaration/MethodDeclaration/Block/IfStatement/InfixExpression/INFIX_EXPRESSION_OPERATOR
diff word: <
```

図 5 抽出情報の例

```
1   public class Example {
2     public static void main(String[] args) {
3       int sum = 0;
4       for (int i = 1; i < 5; i++) {
5         sum += i;
6       }
7       if (sum<10) {
8         System.out.println("hoge");
9       }
10    }
11  }
```

（a）初回提出時のコード

```
1   public class Example {
2     public static void main(String[] args) {
3       int sum = 0;
4       for (int i = 0; i < 5; i++) {
5         sum += i;
6       }
7       if (sum<10) {
8         System.out.println("hoge");
9       }
10    }
11  }
```

（b）最終提出時のコード

図 6 初回提出時と最終提出時のソースコード

4.2 差分情報抽出

差分情報は学習者がどの部分をどのように修正したかを時系列で表しており，単なる間違いの数ではなく「ミスの発生から修正までの過程」を定量的に分析できる．青木らの手法 [3] で抽出結果 R_1〜R_{K-1} を得る．図 5 に抽出結果 R_1 と R_2 の出力例を，図 6 に対応する (a) 初回提出時のコードと (b) 最終提出時のコードを示す．R は最終版との差分抽出の対象とした提出バージョンを示す．例えば，R 1 は 1 回目の提出と最終版との差分である．抽出結果は複数の 1) 構造情報と，2) 原文情報の組からなる．原文情報は 2 つのソースコード間に存在する差分のうち，図 3 のような AST の葉ノードに対応する，修正後の単語を表す．構造情報は修正された葉ノードの親ノードから根ノードまでのすべてのノードの構文情報を表す．また，ある構造情報と原文情報のペアを部分情報と呼ぶ．図 5 の R 1 は初回提出時と最終提出時の抽出結果 R_1 で，1 組の部分情報からなる．CompilationUnit から始まる行は構造情報，diff word から始まる行は原文情報を示す．この部分情報は初回提出時と最終提出時の差が for 文の変数宣言式内の単語 1 箇所で，その単語は最終提出版で 0 になることを示す．図 6(a)4 行目の $i = 1$ と，(b)4 行目の $i = 0$ がこの差に対応する．例の R 2 は 1 度の修正で 2 箇所の修正がされており，対応する部分情報も 2 つ存在する．

提案手法は for 文に対する苦手度を求めるため，差分から for 文の理解に関連すると考えられる部分情報のみを取り出す．差分情報には for 文の理解と関係の無

い修正も含まれている．例えば，図5のR2は「ForStatement」を含まないため本研究の分析対象外と考えられる．また，for文以外の要素の影響が混在している可能性がある．例えば，次の例を考える．

1. ForStatement/Block/IfStatement/ExpressionStatement/Assignment/NumberLiteral
   diff word: 0

例の構造情報から,「ForStatement」の「Block」内に「IfStatement」が存在することから，この部分情報はfor文内にあるif文内の要素が変更されたことを示している．複数の文法を使用した際に生じた間違いは，どの文法への未理解から生じたものであるか判別が難しい．例のような部分情報の出現はfor文のみならずif文に対する苦手度の高さが原因である可能性があり，for文の苦手度への関連性がfor文単体に対する部分情報と異なる可能性がある．そこで，提案手法では差分抽出の工程で抽出結果を条件A,B,Cに基づいて場合分けする（図7）．

- 条件A：for文が何重か
  2重，3重ループのようにfor文の重なる数で受講者の苦手度にあたえる影響の強さが異なる可能性がある．構造情報に含まれるforstatementの数（1つだけ，2つだけ，3つ以上）で分類する．

- 条件B：for文以外の構文を含むか
  他の構文内に記述されたfor文でミスをしている場合，他の構文の苦手度が大きく関わっている可能性がある．構造情報に含まれるもっとも末端のfor文が，他の構文のBlockに含まれるかどうか（含む，含まない）で分類する．

- 条件C：for文内のどの部分か
  for文を構成する1)変数宣言部，2)条件分岐部，3)インクリメント部，4)Block内は役割が大きく異なり，苦手度に与える影響が一定でない可能性がある．構造情報に含まれるforstatement以下の構文要素（VariableDeclarationExpression, InfixExpression, PostfixExpression, Block）で分類する．

なお，条件$A \sim C$の条件が苦手度に影響しない場合を想定し，条件$A \sim C$すべてにおいて「区別をしない」を設定する．そのため，条件Aが4種類，条件Bが3種類，条件Cが5種類の全パターンで合計60通りの方法で差分をカウントする．

4.3 メトリクス算出

出力された差分情報を元に，1)誤修正数，および，2)平均修正量の2種類のメトリクスを算出する．ある構文が苦手な受講者は，その構文中に存在する間違いやミスに気づきにくい．また，効率よくコード修正を行えなかったり，OJSの採点結果をこまめに確認したりするため，コードの修正量に対して提出回数が多く

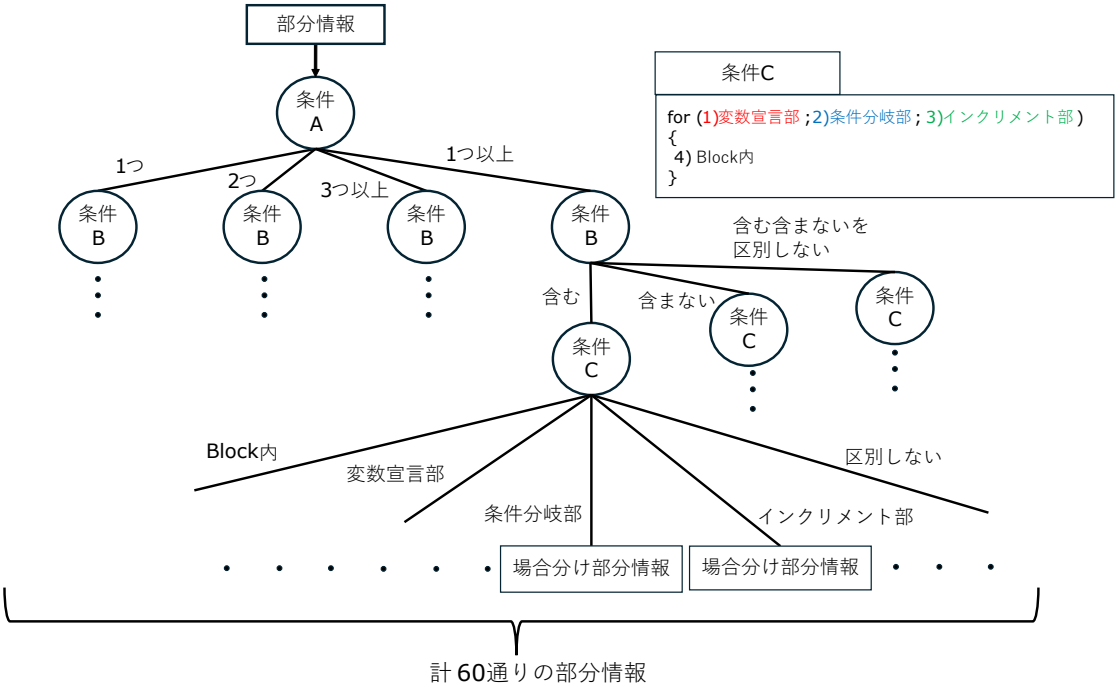


図 7 部分情報の場合分け

なる．提案手法はfor文が苦手な受講者の特徴を，60通りの差分情報それぞれから算出した2種類のメトリクスで表現する．

4.3.1 誤修正数

誤修正数は受講者が本来修正すべき場所を放置した回数を示す．受講者が提出したソースコードに誤りがあり，その誤りを修正せずに再度提出した場合，その誤りが発生した構文に対する受講者の苦手度が高い可能性がある．本研究では誤修正数を次の式で定義する．

$$\text{誤修正数} = \text{初回提出時の間違いの数} + \text{修正後の同じ間違いの数} \quad (4.3.1)$$

初回提出時の間違いの数は，受講者がある課題に対して最初に提出したソースコードと最終版の間に存在する構造情報の数を示し，初回提出版に存在する間違いが多いほど増加する．この値は初回提出時にどれほど間違えていたかで決まり，受講者のコードがOJSの採点を受けていない状態で正解のコードからどれほど離れていたかを示す．修正後の同じ間違いの数は，連続する2つのバージョンのソースコードに同じ間違いが存在した回数を示し，修正されないまま提出された間違いが多いほど増加する．この値は受講者の「コード中の誤りに気付く力」と相関し，受講者がOJSの採点結果を参考にしながら自身のコードをどれほ

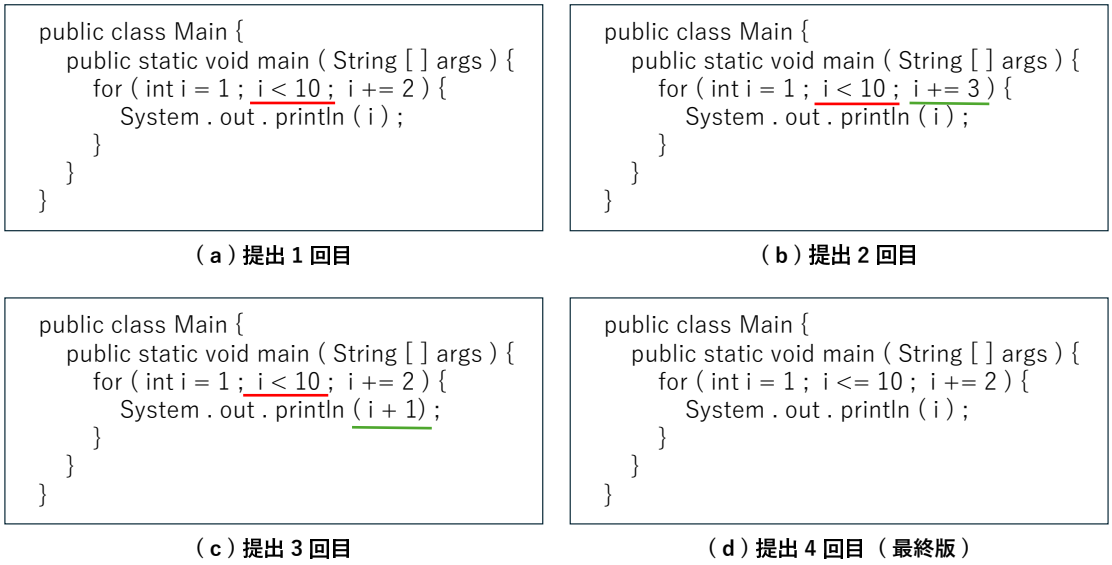


図 8 提出ソースコードの具体例

ど正しく分析できていたかを示す. 誤修正数が大きい受講者はOJSの助力無しでの作成コードの正確さとコード分析能力が低く, 対象構文に対する苦手度が高いと考えられる.

図8に誤修正を繰り返している提出の例を示す. 図は「1から10のうち奇数を出力するプログラム課題」に対して受講者が提出した4バージョンのソースコードである. 赤下線が誤りを示し, 正答である $i <= 10$ に4回目の提出で到達している. 緑下線は受講者が誤修正を行ったことで, 増加した誤りを示す. 図8(a)の提出1回目には誤りが1つあり, この誤りは提出2, 3回目でも修正されずに残っている. この例の場合, 誤修正数は1(初回提出時の間違い数)+2(見逃した回数)の3となる. なお, 図8の緑下線については, いずれも次の提出にて修正されている. これは間違いを発見し, 修正できていることを示すため誤修正数の計算には使われない.

4.3.2 平均修正量

平均修正量は受講者がどれほど効率よくデバッグを行ったか示す. プログラミングの際, 1回の提出あたりの修正量の大小は, 受講者の構文苦手度に関係すると考えられる. 多くのミスをしたとしても, それらすべてを少ない提出数で解決できた場合, 受講者は苦手度が低い可能性がある. 本研究では平均修正量を次の式で定義する.

$$\text{平均修正量} = \frac{\text{間違いの数(重複なし)}}{\text{提出回数}} \tag{4.3.2}$$

間違いの数（重複なし）は，受講者が提出したコード内に存在する間違いの数であり，間違いが多いほど大きくなる．また，連続する提出情報に同一の間違いが見られた場合については，コード修正の効率を測るうえで，重複は平均修正量を無意味に増加させてしまうことから除外し，1つの間違いとしてカウントする．この値は受講者の提出したソースコードに多くの誤りがあるほど増加するため，受講者がどれほど多くの間違いを生み出したか示す．また，提出回数は受講者が最終版を完成させるまでに要した提出回数を示し，多くの提出を行うほどに増加する．この値は受講者が間違いをすべて修正するために，何回の提出及びOJSの採点結果を受け取ったか示し，提出回数で間違いの数（重複なし）を除することで，1度の提出あたりのデバッグの量を求めることができる．以上から平均修正量は受講者の「デバッグの効率性」を示し，平均修正量が小さい受講者ほど，デバッグの効率が悪く苦手度が高い可能性があることを示す．

図8の例では赤下線3つ，緑下線2つの計5つの間違いがある．また，提出回数は4回である．このとき，赤下線の間違いは重複しているために最初の1回のみをカウントするため，平均修正量は$\frac{1+2（間違いの数）}{4（提出回数）}$の0.75となる．

5 実験

ソースコード, 理解度アンケート結果の研究への使用に書面による同意を得た奈良高専3年生18名を対象者として, 提案手法の評価実験を行う.

5.1 対象ソースコード

提案手法の評価実験で差分抽出に使用するソースコードについて述べる. 対象者の奈良高専3年生がJavaを用いたプログラミングの講義「プログラミングII」にて提出したソースコードを収集し, 以下の条件すべてに当てはまる課題の提出履歴から得られたソースコードを提案手法に使用する.

1 対象となる奈良高専3年生全員がfor文を使用していること

2 対象となる奈良高専3年生の全員が最終版までの提出を行っていること

3 上記条件にあてはまるソースコードの中で, 各理解度アンケートの実施日時に最も近い時期に提出された3課題分のソースコード

1について, for文の有無による結果への影響を排除することができる. 2について, 提案手法は受講者が最終版を提出している前提であるため, 最終版を提出していない受講者がいる課題は提案手法に用いることができない. 3について, 後述する理解度アンケートの結果と対応させるため, 課題実施時期が理解度アンケートの直近である必要がある. また, 1つの理解度アンケートに対し, 3課題を対応させることで1課題の難易度による影響をやわらげることができる.

5.2 訓練用データセットの生成

提案手法の評価実験で生成するデータセットについて述べる. 受講者の文法への苦手度と相関する指標を苦手度指標と呼ぶ. 機械学習を行う際, 訓練用のデータには受講者の苦手度を示すラベルを付加する必要があるため, 評価実験の際は「データセット生成」の工程にて2種類のメトリクス(計120次元)を次元圧縮したデータセットに苦手度指標をラベルとして付与する. また, 第1章にて設定した3つのRQを確かめるため, 訓練用データセットは次の3種類のものを生成する.

1 誤修正数データセット(誤修正数をPCAしたものにラベルを付加したもの)

2 平均修正量データセット(平均修正量をPCAしたものにラベルを付加したもの)

3 総合データセット(誤修正数と平均修正量をPCAしたものにラベルを付加したもの)

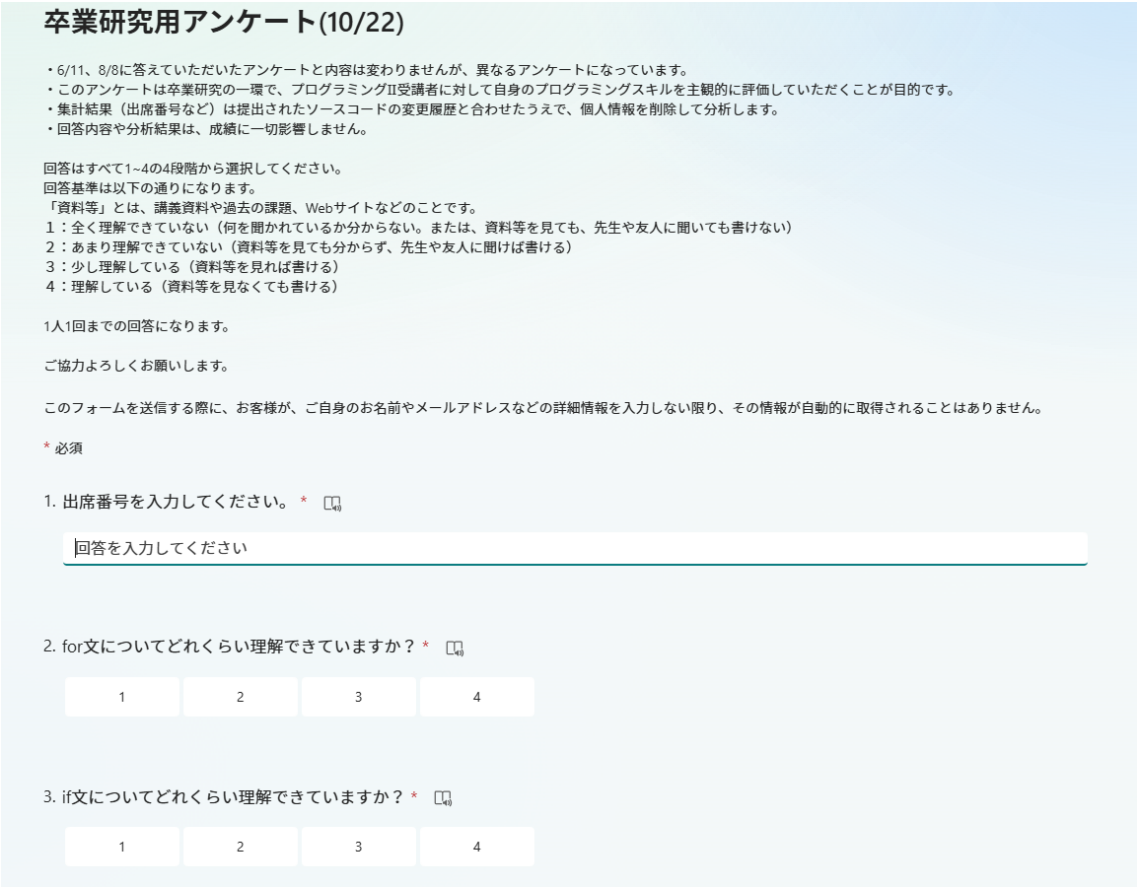


図 9 理解度アンケート画面

各データセットはそれぞれRQ1, RQ2RQ3に対応している.

本研究では苦手度指標として理解度アンケートの結果を用いる. 理解度アンケートは受講者の主観的な構文苦手度の評価が得られる. 理解度アンケートの結果が悪い学生ほど, その評価を下す原因となる間違いがソースコードに多く表れている可能性が高く, 結果として苦手度が高くなると考える. そこで, 評価実験では理解度アンケートへの回答をデータセットのラベルとして使用する.

対象となる奈良高専3年生に向けて, Microsoft Formsによる理解度アンケートを行う. アンケートの内容はいくつかのプログラミング文法についての理解度を4段階で答えるものである. なお, 理解度とは苦手度と反対の意味を持つ指標である. 例えば, ある受講者の理解度アンケート結果が1であった場合, その受講者の苦手度は4であり, 理解度アンケート結果が2である場合は苦手度は3, 理解度アンケート結果が3である場合は苦手度は2, 理解度アンケート結果が4である場合は苦手度は1となる. 本評価実験では, 受講者の理解度アンケート結果を予測することで評価を行う. 図9へ実際のアンケート画面の一部を示す.

図9のような問を「for文」,「if文」,「スタティックメソッド」,「インスタンスメソッド」,「コンストラクタ」,「クラス」について同様に用意し, 対象者が回答する. な

お，評価実験ではfor文に対する回答結果のみを用いる．

理解度アンケートは計3回実施し，実施日時は2025年の6月11日，8月8日，10月22日に実施する．しかし，8月8日実施の第2回アンケートについては，5.1節にて述べた条件にあてはまるソースコードが用意できなかったため，今回は6月11日の第1回，10月22日の第3回アンケートの結果のみを使用する．

以上により，6月11日の第1回，10月22日の第3回アンケート結果を得るが，これらのアンケートは実施時期が離れており，その間に受講者の苦手度は変化している可能性がある．そのため，これらの時期の違うアンケートに回答した受講者は区別する．例えば，同一の受講者が2つのアンケートを受けた場合は，2人分のアンケート結果を得たことと同義とする．結果として本研究では30名分のアンケート結果が得られた．

以上のことから，本研究の評価実験では30人分のデータを持つデータセットを3種類用いる．

5.3 機械学習アルゴリズム

5.1節，5.2節にて用意したソースコードとアンケート結果を用いた機械学習を行う際のアルゴリズムについて，評価実験では順序ロジスティック回帰とORFを使用する．

順序ロジスティック回帰は目的変数が順序尺度を持つカテゴリ変数である場合に有効であるため，提案手法の機械学習のアルゴリズムとして採用する．また，本研究の提案手法は60通りの場合分けを行った際，データにスパース性が発生したため，ORFも採用する．ORFも同様に目的変数が順序尺度を持つカテゴリ変数である場合に有効であることに加え，スパース性への強さを持つため本研究ではこれら2つのアルゴリズムを使用し，比較を行う.

5.4 評価方法

機械学習結果を評価するにあたり，3種類のデータセットによる予測結果それぞれに混同表と的中率，平均絶対誤差(MAE)を出力する．図10へ混同表の例を示す. 横軸が予測結果，縦軸が実際のアンケート結果を示し，交点の数値はアンケート結果に対しその予測値を出力した回数を示す. 例ではアンケート結果が4であったにも関わらず，機械学習で3と予測された回数は8回となる．色付きのマスは予測結果と実際のアンケート結果が一致するマスを表し，色付きのマスの数値の合計が大きいほど，予測の精度は良いと評価できる．

機械学習がどれほどの確率でアンケート結果を予測したか示す的中率は以下の式で算出する．

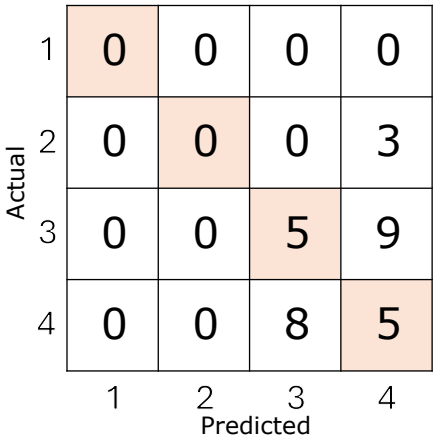


図 10 機械学習結果の混同表例

$$\text{的中率} = \frac{\text{一致回数}}{\text{データ数}} \tag{5.4.1}$$

一致回数は予測結果と実際のアンケート結果が一致した回数, データ数はデータセット内の訓練用データの数を示す. 的中率が高いほど, 機械学習の精度は良いと評価できる.

予測結果が実際のアンケート結果とどれほどずれたか示すMAE（平均絶対誤差）は以下の式で算出する.

$$MAE = \frac{|\text{実際のアンケート結果} - \text{予測結果}|}{\text{データ数}} \tag{5.4.2}$$

MAEが小さいほど, 予測の精度が良いと評価できる. 順序ロジスティック回帰とORFそれぞれについて, 3種類のデータセットを用いた機械学習を行うため, 評価実験では結果として6つの混同表とMAEを求める. 得た混同表, 的中率とMAEについて, 順序ロジスティック回帰を用いた場合とORFを用いた場合の比較を行う. また, 3種類のデータセットの比較も行い, 使用するメトリクス評価も行う.

表 2 機械学習結果の MAE

	誤修正数のみ		平均修正量のみ		総合	
	的中率	MAE	的中率	MAE	的中率	MAE
順序ロジスティック回帰	0.300	0.767	0.300	0.767	0.300	0.800
ORF	0.167	0.900	0.733	0.300	0.567	0.467

6 結果と考察

表2へ各アルゴリズムで異なるデータセットを予測したときの的中率とMAEを示す. もっとも的中率が高く, かつMAEが低かったのは平均修正量データセットからORFにより学習した場合であり, 的中率0.733, MAE 0.300と非常に高い精度だった. 各データセット内での結果を比較すると, 誤修正数データセットに対しては順序ロジスティック回帰で的中率, MAEが良い. 平均修正量データセットと総合データセットに対してはORFが良い結果を示した.

図11に全組み合わせの混同表を示す. 順序ロジスティック回帰を用いた場合(a)〜(c), いずれのデータセットを使用した場合でも混同表が酷似している. 予測が一致した回数はいずれも9回であり, 残りの21回の分布も似ている. 理解度アンケートでは4と回答した学生がもっとも多く（30人中19人）予測結果が4と出力される可能性が高いと思われるが, 実際の予測結果は3が多く, 4は1度も予測されなかった. ORFを用いた場合(d)〜(f), データセットごとに違いが見られ, 4が多く予測されている.

アルゴリズムによる違いが発生した原因として, スパース性への強さの差が考えられる. 順序ロジスティック回帰とORFはいずれも目的変数が順序尺度を持つカテゴリ変数である場合に用いられる回帰モデルであるが, スパース性への強さが大きく異なる. 順序ロジスティック回帰はスパース性に弱く, 一方でORFはスパース性に強いRandomForestの拡張である. また, 提案手法では4.2節で述べた通り, 60通りの場合分けを行った. この際, データにスパース性が発生し, 順序ロジスティック回帰による予測を妨げたと考えられる. 図12に実験で用いた誤修正数の一部を示す. 各行がある受講者の誤修正数を, 各列が場合分けを示している. 算出された誤修正数は多くの0が存在するスパースなデータであり, このようなスパース性は平均修正量にも見られた. スパース性の強いデータを用いた機械学習は, 回帰式を作成する際に重みが0に近い値になりやすい. そのため本実験では, 順序ロジスティック回帰はどのデータセットを用いても, 回帰式内でのメトリクスの重みが0に近くなったと考えられる. したがって, 順序ロジスティック回帰は本研究の提案手法には不向きであると考えられ, 提案手法はORFを用いた場合のほうが精度が高いと考えられる.

次に, ORFの結果について比較すると, もっとも精度が高かったのは平均修正量

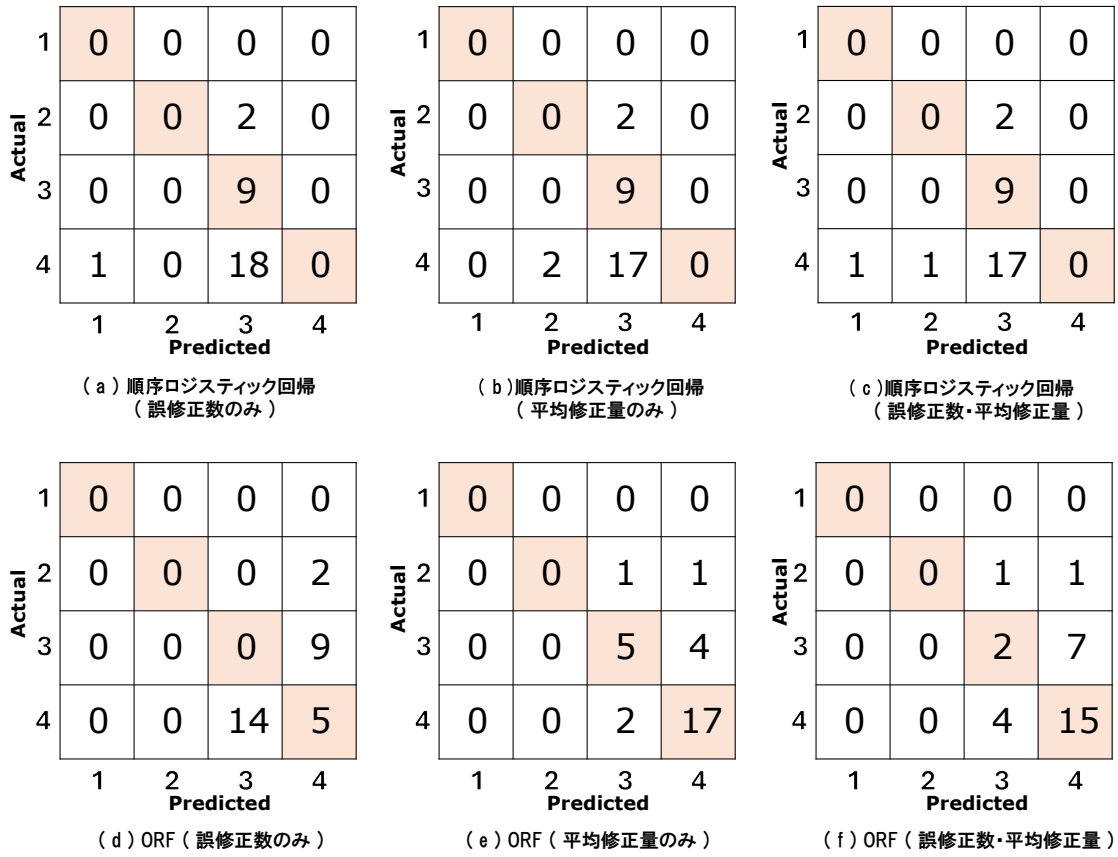


図 11 機械学習結果の混同表

データセットを用いた場合の的中率 0.733, MAE 0.300 であった. 一方, 誤修正数データセットを用いた場合は的中率 0.167, MAE 0.900 と大きく劣り, 総合データセットを用いた場合でも的中率 0.567, MAE 0.467 と平均修正量データセットを用いた場合より精度が悪化した.

以上の結果に基づいて, RQ に回答する.

RQ1: 初回提出時の間違い数と修正できなかった間違い数から for 文に対する苦手度を予測できるか?

RQ1 への回答: 誤修正数データセットによる予測精度が低く, 初回提出時の間違い数と修正できなかった間違い数からの for 文に対する苦手度の予測は難しい.

RQ2: 間違い数と提出回数から for 文に対する苦手度を予測できるか?

RQ2 への回答: 平均修正量データセットの予測精度が高く, 間違い数と提出回数から for 文に対する苦手度は有効である.

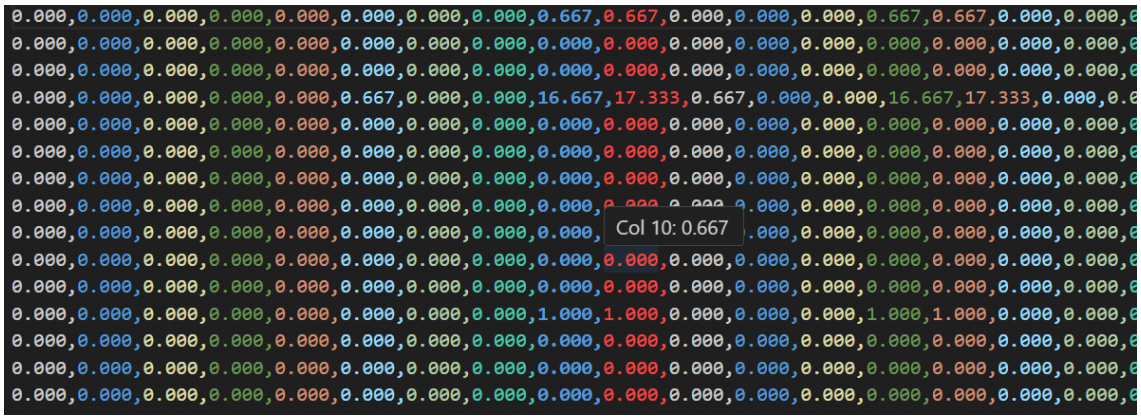


図 12 スパース性を持った誤修正数データ

RQ3: 上記の組み合わせからfor文に対する苦手度を予測できるか?

RQ3への回答: 総合データセットの予測精度が, 平均修正量データセットの予測精度より低いことから, 初回提出時の間違い数と修正できなかった間違い数を用いる必要性は薄い.

7 おわりに

本研究はOJSの個別フィードバック生成を目的に，受講者の特定の構文への苦手度を予測する手法を提案した．提案手法は学生が1つの課題に対して提出した一連のソースコードから構文単位の差分を抽出し，各構文に対する誤修正数と平均修正量を求める．誤修正数は受講者のソースコード内の初回提出時の間違い数と修正できなかった間違い数から計算した苦手度と相関すると考えられるメトリクスであり，平均修正量が間違い数と提出回数から計算した苦手度を相関すると考えられるメトリクスである．抽出差分をfor文への苦手度に影響を与える要因で場合分けし，60次元の構文要素に対して求めた合計120次元を特徴量として主観的な理解度をラベルとした機械学習を行い，精度を評価した．評価の際には主観的な苦手度とそれに対する予測結果が示された混同表と，主観的な苦手度と予測結果が一致した割合を示す的中率と，その平均絶対誤差(MAE)を評価する．評価の結果，平均修正量データセットを用いてORFで学習した場合に的中率とMAEの両面において他の条件を大きく上回った．また，アルゴリズムを比較すると，順序ロジスティック回帰では使用するデータセットに依らず予測結果がほぼ同一となり，訓練用データのスパース性の強さと順序ロジスティック回帰との相性の悪さが示され，提案手法にはORFのようなスパース性に強いモデルの使用が有効であると考えられる．次に，ORFを用いた場合のメトリクスを比較すると，最も精度が高かったのは平均修正量データセットを用いた場合であり，誤修正数を用いた場合が低くなった．また，誤修正数と平均修正量の併用もかえって精度を下げる結果となった．

本研究が提案した手法によってOJS上に提出された課題から受講者のfor文に対する苦手度が予測できる．苦手度を受講者に提示することで自身が苦手な文法を把握し，その文法に焦点を当てた効率的な復習や学習計画の立案が可能となる．また，教員はfor文を苦手とする学生に対して補助課題等を与えることで個人の苦手度に応じた学習環境の構築が可能となる．例えば，ある受講者のfor文の苦手度がOJS上に表示される機能が存在した場合，受講者は自身の苦手度が高ければfor文を重点的に学習し，苦手度が低ければfor文の勉強の優先順位を下げ，他の構文の勉強時間に充てることが可能となる．また，教員は受講者のfor文の苦手度が低いことが確認できた場合にfor文に関する追加課題や補助教材を該当する受講者に配布することができる．また，受講者のfor文の苦手度が全体的に低い場合は，講義内容をよりfor文に寄せるなどの対応をすることが可能となる．

本論文では提案手法についてfor文に限定した評価を行ったが，if文など他の文法に対する苦手度の予測にも応用できる．if文など他の文法苦手度予測に応用することで，学習者はより学習すべき文法を把握しやすくなる．また，他の文法の苦手度と比較しての相対的な評価も行えるようになる．相対的評価が可能にな

れば，受講者が優先して苦手を克服する必要のある文法を示すことができ，より詳細な受講者の学習補助が可能となる．加えて，他の文法の苦手度予測結果と組み合わせることで，「for 文中の if」や「while 文中のメソッド呼び出し」など，より限定的な状況の苦手度の予測が可能となると考えられる．複数文法の組み合わせに対する苦手度が予測できれば，受講者や教員はより対策問題を作りやすくなり，学習効率向上につながると考えられる．

本研究の今後の展望として，以下が挙げられる．提案手法の精度を向上させる方法として，本論文では評価対象の機械学習アルゴリズムとして順序ロジスティック回帰と ORF を用いたが，さらにスパース性に強いアルゴリズムを用いることで予測精度の向上が期待できる．また，本研究では for 文が使用された状況ごとに場合分けを行い，計 120 次元の説明変数を PCA により圧縮したが，この場合分けが適切であるかどうかは十分検証できていない．適切でない場合分けは機械学習データのスパース性を高める要因となるため，各場合分けがどれほど有効であったかの評価実験を行う必要がある．本研究の発展として，本論文の評価実験では受講者の苦手度評価指標としてアンケート結果を用いたが，アンケートは主観的な評価であるため，客観的な評価に基づいた指標とは異なる可能性がある．提案手法を試験結果や成績点などの客観的な評価指標を用いるよう拡張することは興味深い発展の 1 つである．本研究の応用として，if 文などの他の文法への提案手法の適用が挙げられる．提案手法は逐次・分岐・反復の 3 要素を持つ for 文において有効性が確認されたため，他の文法への適用が可能であると考えられる．ただし，場合分けについてはその文法に適当であるものに変更する必要がある．

謝辞

本研究を行うにあたり，被験者実験にご協力いただいた方々や，ご指導いただきました上野秀剛准教授，並びに本論文の査読をしていただいた岡村真吾教授に，この場を借りて心からお礼を申し上げます.

参考文献

[1] Wenju Zhou, Yigong Pan, Yinghua Zhou, Guangzhong Sun, "The framework of a new online judge system for programming education," Proceedings of ACM Turing Celebration Conference (TURC2018), pp.9–14, (2018)

[2] Hui Sun, Bofang Li, Min Jiao, "YOJ: An Online Judge System Designed for Programming Courses," Proceedings of the 9th International Conference on Computer Science Education (ICCSE2014), pp.812–816, (2014)

[3] 青木晃汰, 上野秀剛, "差分構文木を用いたプログラミング授業受講者のコーディング特徴の自動抽出", ソフトウェアエンジニアリングシンポジウム 2024, (2024)

[4] 槇原絵里奈, 池田太郎, 小野景子, 新濱遼大, "オンラインジャッジシステムにおける解答履歴を利用した問題の関係性調査", 情報処理学会論文誌, Vol.63, No.3, pp.742–751 (2022)

[5] 鄭佳健, 寺田実, "初心者のプログラムを正誤と質で評価するシステム", 第 62 回プログラミング・シンポジウム予稿集, pp.75-82 (2021)

[6] 福元春輝, 伊原彰紀, "共変更されるソースコード修正パターンの抽出", 2021 年度情報処理学会関西支部支部大会講演論文集, (2021)

[7] Hoan A. Nguyen, Anh T. Nguyen, Tung T. Nguyen, Tien N. Nguyen, Hridesh Rajan, "A study of repetitiveness of code changes in software evolution," 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp.180–190 (2013) pp.833–844 (2017)

[8] 入山優, 肥後芳樹, 楠本真二,"抽象構文木を利用した API の後方互換性が破壊される変更の検出", ソフトウェアエンジニアリングシンポジウム 2021 論文集, pp.209-218 (2021)

[9] 市井誠, 小川秀人, "モデル変換を用いたリファクタリング検証手法", コンピュータソフトウェア, vol.32, No.3, pp.70-76 (2015)