卒　業　研　究　報　告　書

令和3年度

研究題目

プログラミング授業のソースコード提出履歴を用いた理解の有無の予測

指導教員　　上　野　秀　剛　　准教授

氏　　名　　西 城 戸 星 龍

令和4年2月21日　提出

奈 良 工 業 高 等 専 門 学 校　　情 報 工 学 科

プログラミング授業のソースコード提出履歴を用いた理解の有無の予測

上野研究室　西城戸星龍

プログラミン講義では，少数の教員が多数の受講者を対象に講義を行うため，教員の負担が大きい．また，演習中に各受講者全員の作業状況を常に監視できない．これまでの研究ではプログラミングの自動採点システムである Online Judge System (OJS) を用いることで教員の負担が軽減され，受講者の作業状況の識別が可能であることが分かっている．しかし，自動採点されるため無作為に修正する受講者の存在が報告されているが，無作為な修正をする受講者だけを識別することは難しい．そこで無作為修正者を含めた理解が不十分な受講者を演習時間内に予測できると教員の負担が軽減できより効果的な指導ができると考えた．そのため，本研究では講義時間内に受講者の完答までにかかる時間の予測を目的とした．実験では，先行研究で提案されていたメトリクスのみのデータと提案メトリクスを追加したデータの 2 つについて機械学習を用いて完答までにかかる時間の予測した．また，各メトリクスと完答までにかかる時間の相関係数を求め，その有意性を検定した．結果は，機械学習に使用するメトリクスを追加する前と追加した後で，課題開始からの時間により予測精度の優劣が変化した．そのため，課題開始からの時間によりメトリクスを変更することで予測精度が向上することが示唆された．また，修正行の偏りと完答までの時間や訂正回数，平均修正行数，0 点での提出回数，提出全体での 0 点の割合，が有意に弱い正の相関があることが分かった．今後の課題として，予測精度の向上が挙げられる．そのためには学習に使用するデータセットの変更した場合での実験や，課題を開始してからの時間によって使用するメトリクスや重要度の変更が挙げられる．

目 次

1 序論

教育現場におけるプログラミング教育では座学と演習を組み合わせた授業形式が一般的である．授業では少数の教員が多数の受講者を対象に講義を行うため，演習中に各受講者全員の作業状況を常に監視できない．そのため、各受講者の理解状況や作業の様子を把握することが難しい．その結果，理解が不十分な受講者を教員が直接指導することが難しい．

教育効果を高めるために講義の受講者を対象とした状況識別が研究されている [1–7]．その中でも，プログラミング講義を対象にスナップショットを用いた研究が複数報告されている [8–15]．スナップショットはある時点でのプログラムの情報であり，ソースコード，日時，エラー情報などが記録される．先行研究では受講者のスナップショットから算出されるメトリクス値をもとに無作為修正者を識別しようと試みた [23]．しかし，指導が必要な受講者は無作為修正者だけではないことや，無作為修正をする前に指導できないことから，無作為修正者だけを識別するだけでは実際の授業時に活用することは難しい．そのため，無作為修正者を含めた理解が不十分な受講者を講義時間内に予測できれば効率的に指導できると考えた．また，課題を解くために必要な知識を十分に理解できている受講者に比べ，理解が不十分な受講者は課題を開始してから完答するまでにかかる時間が長いと考えられる．そのため，本研究では各受講者のスナップショットから算出されるメトリクス値をもとに各受講者が課題を完答するまでにかかる時間を予測する．

プログラミングの学習環境の 1 つとして，Online Judge System (OJS) がある．OJS は課題に対応したソースコードを学生が作成し提出すると，コンパイル･実行し，その実行結果を元にソースコードを採点する．プログラミング講義に OJS を用いることで，受講者は演習課題で作成したソースコードを OJS に提出し即座に採点結果を得ることができる．従来のプログラミング講義では受講者はソースコードを 1 度提出するだけで，評価もすぐにはわからない．そのため，受講者が提出したソースコードが演習で課された要求を満たしていないとき，受講者は誤答であることに気づきにくい．一方で，OJS による講義では，提出したソースコードの評価がすぐにわかる．そのため，従来の講義形態と比べ受講者がすぐにフィードバックを受けられることから，再提出しやすくなっている．そのため，誤答に対して原因を考察し修正する受講者や，修正方法が分からず放置してしまう受講者など複数の振る舞いが考えられる．そこからプログラミングの理解が不十分な受講者の提出履歴の特徴を抽出することで課題の完答までに時間のかかる受講者の予測に役立てられると考えられる．本研究では，OJS を用いたプログラミング講義を分析対象とする．受講者は OJS にソースコードを随時提出するため，提出ごとのすべてのソースコードがスナップショットとして記録される．

そのため，スナップショットから各受講者がどのような編集をしたのか履歴をとることができる．OJSから得られるソースコードの編集履歴から各受講者の行動や理解状況を把握できれば，将来的に演習中に対象を絞った指導をすることが可能となる．

本研究では理解が不十分な受講者の予測を試みる．理解が不十分な受講者とは完答できなかったり完答までに極端に時間がかかったりする受講者を指す．理解が十分かどうかを正確に判定することは難しいため，本研究では完答までにかかる時間に絞り予測を試みる．

他人から正解のコードをもらうなどして，課題で求められているプログラミング能力の理解が十分でないまま演習を完答してしまうとそれまでに培っているはずのプログラミング能力を前提とした演習についていけない可能性がある．演習中に理解が十分でない受講者を予測することで，教員が直接指導して課題を通して理解して欲しいプログラミング能力の理解を促すことができ，より多くの受講者が理解に辿り着くことができると考えられる．

2 関連研究

2.1 受講者の状況識別

講義の教育効果を高めるために受講者を対象とした状況識別が研究されている．Yoshihashi らは高等教育機関における授業の教育効果を改善することを目的に，畳み込みニューラルネットワーク (CNN: Convolutional Neural Networks) を用いて聴講者の受講状態を識別するシステムを提案している [1]. システムはカメラで教室前方から後方に向かって撮影された授業映像から聴講者を検出する．検出された各聴講者に対して，検出された各聴講者の状態を CNN を用いて推定し,「集中」,「非集中」,「聴講者不在」の 3 種類の状態ラベルを割り当てる．Baradwaj らは，受講者の講義出席状況，試験の点数，課題の提出状況などから，その受講者の学期末時点の成績を決定木学習を用いて予測している [2,3].

また，学習管理システム (LMS: Learning Management System) などの授業支援システムを導入した講義で得られたデータに対してデータマイニングの手法を用いた「学習分析」(LA: Learning Analytics) や Educational Data Mining (EDM) に関する研究が報告されている [4–7]. 加藤は LA や EDM の利点を以下のように述べている [6,7].

(1) 学習者の傾向と行動パターンの解読
学習者の傾向と行動パターンの解読により，脱落の恐れがあるグループを発見し，より良い教授法を構築し，在籍率向上が期待できる．

(2) 理解度不足の学習内容と行き詰まり原因の推定
理解度不足の学習内容をシステムが把握することにより，学習者に最良の学習方針を提案することが期待できる．

(3) 到達学力の推定
到達学力を推定することで，学習者の進捗状況に対してリアルタイムに反応して教材への取り組みを深められる教育ソフトウェアや適応学習環境の設計に役立つ情報が得られる可能性がある．

以上の研究報告から，講義の受講者の状態や行動を記録することで，講義の雰囲気や受講者の集中度合い，取り組み状況を自動的に推定できることが分かる．これにより，指導すべき受講者を自動で識別できるため，教員は効率よく指導できることが期待される．

2.2 プログラミング講義における状況識別

プログラミング講義を対象に受講者の状況識別を行った研究が複数報告されている．Ihantola らは，プログラマがプログラムの実装時に行なったコーディング，コンパイル，デバッグ，テストなどによって作成される成果物，およびそれに対応す

るメトリクスを通してプログラミング行動が計測できると述べている [8]. 本研究で識別に用いるスナップショットも, プログラムのコーディングで得られる成果物であり, プログラミング行動を計測できると考えられる.

Fujiwara らはプログラミング演習時に取得した初学者のソースコード編集履歴を分析し, その傾向を明らかにした [9]. 分析では (1) プログラミング熟練者の目視による定性的分析と, (2) ソースコードのトークンレベルでの Levenstein 距離の計測に基づく定量的分析を実施した. 分析 (1) の結果, 初学者が誤りに陥っている際に本来修正すべき箇所ではなく, 課題内容とは関係の無い箇所を編集するといった行動パターンが観測された. 分析 (2) の結果, 演習において理解困難な状態に陥っている場合, 距離の変動を確認することで初学者が誤りに陥っているか検出できる可能性があることがわかった.

また, 藤原らは受講者のソースコード編集履歴を分析し, どのような箇所で行き詰まっていたのか特定する手法を提案した [10]. 提案手法は, ある時点における演習課題のソースコードを, 演習課題完了時のソースコードにするために必要な作業を推定残作業量として定量化する. 推定残作業量の変化を元に, 受講者が作成しているソースコードが正解に近づいていない, 行き詰まり区間を検出し, その区間のスナップショットからどのような箇所で行き詰まっていたのかを特定する.

槇原らは探索的プログラミングと呼ばれる, プログラムの実現方法が定かではないときに学習者が修正・コンパイル・実行を繰り返すプログラミング行動に着目した手法を提案している [11]. 手法は学習者がプログラムのどこで探索しているかを検出し, 課題に対する取り組みや行き詰まりをリアルタイムに特定する.

藤原らはプログラミング学習者の学習状況から測定されるソースコードの行数, コンパイル回数, プログラムの実行回数などの時系列データから学習者の特徴を定量化する手法を提案した [12]. 複数の時系列データをグラフで可視化し, 関連性のあるところに対してデータ値の増減, データ値の変化量, データ値の上限, データ値の下限, データの時間微分値の 5 つの定性的な表現に当てはめ, 特定の学習状況に現れるデータ値の推移の特徴を数式として定量化する. 分析の結果, 一定時間でのソースコードの行数の増減やプログラムの実行回数の増加などから学習者の行き詰まりを定量化できることがわかった.

Jadud らは初学者向けプログラミング講義を対象に, エラーの発生と修正の傾向を識別する手法を提案した [13]. 手法は発生したエラー数と修正から求められる値である Error Quotient (EQ) を用いる. 演習の点数と講義の成績のそれぞれと EQ に相関関係があるか検証した結果, 両方に負の相関があることが示された. また, EQ において機械学習を用いて受講者を識別している [14].

Blikstein は自由解答 (Open-ended) 形式によるプログラミング課題に対するスナップショットからコンパイル回数やソースコードのサイズの増減, コンパイルの成

功・失敗の回数などを算出し，受講者の状況や振る舞いを分析した [15].

以上の研究は，いずれもプログラミング受講者が作成するソースコードのスナップショットを分析することで，演習中の受講者の状況を把握している．本研究は，OJS を用いたプログラミング講義を対象とし，理解が不十分な学生に限定した機械学習を用いた予測を試みる．理解が不十分な学生は完答までにかかる時間が長いと考えられる．そのため，解答時間を予測することで理解が不十分な受講者の予測ができると考えられる．また，解答時間に着目し，予測している点でこれらの研究と異なる．本研究はプログラミング学習者の中で理解が不十分なため完答までに時間がかかる受講者を予測するために，スナップショットから得られるメトリクスを用いて完答までにかかる時間の予測を試みる．

2.3 OJS に関する研究

本研究は OJS を用いたプログラミング講義を対象としている．OJS は Automatic Assessment (AA) Tool, AA System とも呼ばれており，これまでにプログラミング教育現場での活用 [16], 不正コピーを検出する機能を追加した OJS の実装 [17], LMS とのプラグイン連携に関する調査 [18] , 再提出回数を制限することによる受講者の行動の変化の検証 [19,20] など多岐に渡って研究されている [16–22]. 本節では，プログラミング教育現場での OJS の活用による影響について述べる．

2.3.1 プログラミング教育現場での OJS の活用メリット

OJS は自動でプログラムを採点するため，手動での採点（手動でプログラムを実行し，目視で実行結果を確認する方法）に比べ，より短い時間で，かつ人手がより少ない状態で採点できる．

岩本らはプログラミング講義受講者の演習課題に対するソースコードの不正コピーを検出する OJS を実装した [17]. 著者らは，コードクローン検出手法を用いて，ソースコードにおける不正コピーを検出している．岩本らのシステムによって，ソースコードの不正コピーを常習的に行う受講者を特定できるようになり，演習課題に対して手をつけられない，全くやる気の無い受講者に対する支援が可能となった．

以上の研究報告から，プログラミング教育現場に OJS を導入することで，教員の負担を軽減する効果や特定の性質を持つ受講者を識別できると期待されている．

2.3.2 プログラミング教育現場での OJS の活用のデメリット

OJS による受講者への採点結果のフィードバックによって，受講者にプログラミング演習中に誤答に対する原因を考える機会を与えられる一方で，1 章で述べた

無作為修正（誤答の原因を考察せず, ソースコードを修正する）を繰り返す受講者が発生することが考えられる点が指摘されている [21,22]. Ben-Ari は誤答の原因を考察せず, 多くの回数をかけてソースコードを修正する行為を bricolage と定義しており, 開発の上で非効率であると述べている [22].

これらの研究報告に対し, 受講者の無作為修正を防ぐための研究がいくつか報告されている. Karavirta らは, ソースコードの修正・提出できる回数を制限して受講者に修正内容を考察させることが無作為修正の抑制に有用かどうかを検証した [19,20]. また, 松永は授業中に紙と鉛筆を取り入れ, プログラムの作成・修正の前に, 論理的に正しいかどうか考えさせる手法を試みた [16]. これらの研究では, プログラミング講義の受講者から得られるデータを細かく採ることができず, 受講者が無作為修正を行っているかどうかを判断することが困難となる. また, 修正内容を考えて修正している中で提出の制限回数に到達してしまったとき, 受講者の考えている時間や努力が無駄となるため, 受講者のモチベーション低下を生んでしまうことが考えられる. そのため, 提出回数やフィードバックの制限を無くすことが必要であると考えられる.

また, 大野は無作為修正者を複数のメトリクスと点数との相関を取ることで識別する手法を試みた [23]. この研究では特定のメトリクスが点数との相関があることを解明した. しかし, 無作為修正者のみを識別することはできていない. また, 識別するだけでなく予測することで更に実際の講義で役に立つと考えた. そこで無作為修正者を含めた完答までに時間のかかる受講者を講義中に予測したいと考え, 本研究では提出履歴から完答までにかかる時間を予測できるかを検証する.

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，ある単元に対して座学の後に演習課題を解く形式である．受講者は演習課題に対してオンラインエディタで作成したソースコードをOJS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，教員があらかじめ作成したテストケースと回答結果が一致するか判定する．テストケースとは作成したプログラムが要件を満たしているかテストするための入力と，入力に対して望まれる出力の組みを表す. テストケースの出力と提出されたプログラムの出力の一致した数に応じて以下の式で点数scoreが計算される．

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

受講者は各テストケースの正誤判定結果，score, コンパイル時のエラー，実行時エラーを得る．受講者はscoreが100点になるまで，ソースコードの修正と提出を繰り返す. 作成されたソースコードは提出ごとにリビジョンとして，提出日時，点数とともに記録される．

3.2 ランダムフォレスト

本研究はランダムフォレストを用いて，受講者の提出履歴から完答までにかかる時間を予測する．ランダムフォレストとはLeo Breimanによって提案されたアンサンブル学習アルゴリズムである[24]. 大数の法則やout-of-bagエラーを利用しているため，予測精度が高い．また，学習に使用した各メトリクスの重要度を算出するため，提案メトリクスの有用性を評価できる．メトリクスとは学習させるときに使用する特徴量のことである．本研究では各提案メトリクスの有用性の評価もするため，ランダムフォレストを利用する．また，ランダムフォレストには回帰と識別の2種類があるが，本研究では予測する値が2値ではない時間長のため，回帰分析を行う.

4 提案手法

提案手法は，受講者が完答するまでの時間を予測し，各メトリクスが予測に有用であるかを検証する．また，演習時間中に直接指導することを将来的な目標としているので，演習時間中に受講者が完答するまでの時間を予測しなければならないそのため，リビジョン 1 の提出から t 分間に提出されたソースコードの変更履歴からメトリクスを計算して予測に用いる．

大野の研究では無作為な修正を繰り返す無作為修正者の識別のため複数のメトリクスを提案し，スコアとの相関を確認した [23]. 無作為修正者はプログラミングに対する理解が十分でないため完答までに時間もかかると考えられる．そのため、本研究では大野研究のメトリクスを利用して予測を試みる．また，各メトリクスについてなぜ本研究の予測の役に立つかをそれぞれ説明する．加えて，メトリクスが多い方が予測精度が向上すると考えた．また，大野研究では受講者の行動の量を表すメトリクスが多いため本研究では行動の質や時間を評価するメトリクスを複数追加する．

4.1 先行研究のメトリクス

先行研究では以下の 4 つのメトリクスを用いた．

Freq (t)：修正行の偏り
DiffLine (s,t)：s 行以下の修正が続いた回数
Revs (t)：t 分間のリビジョン数
AveDiff (t)：1 回あたりの平均修正行数

4.1.1 Freq (t)

Freq (t) は t 分間に修正された行の偏りを表す. 受講者が無作為修正を行う時，エラーの原因と思われる箇所に対して，自分が持っている修正パターンを闇雲に適用する傾向があると考えられる．そのため，たとえば if 文の条件式のような 1 行に対して繰り返し修正を行うと考えられ，一定時間内の修正がわずかな行に集中している可能性がある．先行研究ではソースコードの各行に対する修正回数から分散を求めることで，t 分間に行われた修正がどれだけ特定の行に集中しているかを分析した．

エラーの原因でない箇所を集中して修正した場合，点数の向上につながらない時間が使われる可能性がある．また，エラーの原因が複数行あった場合，修正箇所が偏っていない受講者と比べ修正箇所が偏っている受講者はエラーの原因を全て無くすまでにかかる時間が長くなると考えられる．そのため，本研究の予測に役立つと考えられる．

4.1.2 DiffLine (t)

DiffLine (s, t) は，t 分までの提出の中で s 行以下の修正が続いた回数の割合である．受講者が無作為修正を行う時，エラーが見つかると原因と思われる箇所をわずかに修正し，その正しさを考えることなくコンパイル・実行を繰り返す傾向があると考えられる．そのため，たとえばif 文の条件式だけのような 1 行の修正を繰り返す可能性がある．また，途中で s 行より多い行数を修正したとしても再度 s 行以下の修正を繰り返す可能性がある．そのため，全ての提出の中で s 行以下の修正をした回数合計の割合も算出する．

無作為修正によって DiffLine (s, t) が大きくなる傾向があると考えられる．なお，本研究では先行研究を踏襲し s = 1 とし，1 行以下の修正が続いた回数を用いて計算する．

Freq (t) と同じくエラーの原因でない箇所を集中して修正した場合，点数の向上につながらない時間が使われる．また，エラーの原因が複数行あった場合，複数行を同時に修正する受講者に比べ Freq (t) の値が大きい受講者はエラーの原因を全て無くすまでにかかる時間が長くなると考えられる．そのため，本研究の予測に役立つと考えられる．

4.1.3 Revs (t)

Revs (t) は，t 分間に提出されたリビジョン数である．受講者が無作為修正を行う時，エラー箇所と思われる部分に自分の修正パターンを闇雲に適用する傾向があると考えられる．そのため一定時間内の修正・コンパイル回数が多く，無作為修正によって Revs (t) が大きくなる傾向があると考えられる．

課題を理解しており，短い時間で完答できる場合，闇雲な修正を繰り返し連続で提出することはないと考えられる．つまり，Revs (t) の値が大きい受講者は課題のパスに繋がる効率的な修正をしていないと考えられる．そのため，Revs (t) の値が大きい受講者は完答までにかかる時間が長いと考えられるため，本研究の予測に役立つと考えられる．

4.1.4 AveDiff (t)

AveDiff (t) は，1 回あたりの平均修正行数である．受講者の理解が不十分な時，エラーが見つかると複数個所が原因になっている可能性を考えず，ごく少量の修正を行い，コンパイル・実行を繰り返す傾向があると考えられる．そのため，1 回あたりの修正量は小さくなる可能性がある．

複数行を同時に修正する回数が少ない受講者は AveDiff (t) の値が小さくなる．そのような受講者はエラーの原因が複数行ある場合，同時に複数行修正する受

講者と比べプログラミングが理解できていないと考えられる．そのため，本研究の予測に役立つと考えられる．

4.2 提案メトリクス

本研究では以下の6つのメトリクスを先行研究のメトリクスとともに予測に用いることを提案する．

ErrorCnt (t)：SyntaxError でなくなるまでの提出回数
ErrorTime (t)：SyntaxError でなくなるまでの時間
ErrorPct (t)：t 分間の SyntaxError 率
ZeroCnt (t)：0 点でなくなるまでの提出回数
ZeroTime (t)：0 点でなくなるまでの時間
ZeroPct (t)：t 分間の 0 点率

4.2.1 ErrorCnt (t)

ErrorCnt (t) は演習を開始してから t 分までの間で初めて提出結果が SyntaxError でなくなるまでの提出回数である．受講者が課題を完答するために必要なプログラミングの知識を理解できていない場合，プログラミング言語の文法が理解できていない場合があると考えられる．そのため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，提出結果が SyntaxError でなくなるまでの提出回数は多くなる傾向があると考えられる．

4.2.2 ErrorTime (t)

ErrorTime (t) 演習を開始してから t 分までの間で初めて提出結果が SyntaxError でなくなるまでの時間である．受講者がプログラミング言語の文法が理解できていない場合，提出回数に関わらず完答までの時間はかかると考えられる．そのため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，提出結果が SyntaxError でなくなるまでの時間が長くなる傾向があると考えられる．

4.2.3 ErrorPct (t)

ErrorPct (t) は t 分までの提出の中で提出結果が SyntaxError の割合である．受講者が課題を完答するために必要なプログラミングの知識を理解できていない場合理解できていない時，プログラミング言語の文法が理解できておらず，一度 SyntaxError でなくなっても再度 SyntaxError になる場合があると考えられる．その

ため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，提出結果がSyntaxErrorの割合が増える傾向があると考えられる．

4.2.4 ZeroCnt (t)

ZeroCnt (t) はt分までの提出の中で初めて0点でない提出結果が出るまでの提出回数である．受講者が課題を完答するために必要なプログラミングの知識を理解できていない場合理解できていない程，エラーの原因箇所の正確な特定が遅くなり，エラーの原因箇所以外の訂正を繰り返すことで0点での提出を繰り返すと考えられる．そのため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，0点でなくなるまでの提出回数は多くなる傾向があると考えられる．

4.2.5 ZeroTime (t)

ZeroTime (t) はt分までの提出の中で初めて0点でない提出結果が出るまでの時間である．受講者が課題を完答するために必要なプログラミングの知識を理解できていない場合理解できていない程，提出回数に関わらず完答までの時間はかかると考えられる．また，提出結果が0点でなくなると課題を解決するために必要なプログラミング知識を理解できたと考えられるため，ZrtoTime (t) の値が大きいほど理解が遅いと考えられる．そのため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，提出結果が0点でなくなるまでの時間が長くなる傾向があると考えられる．

4.2.6 ZeroPct (t)

ZeroPct (t) はt分までの提出の中で提出結果が0点である提出の割合である．受講者が課題を完答するために必要なプログラミングの知識を理解できていない程，課題の解決方法が理解できておらず，0点での提出が増えると考えられる．また，一度0点でなくなっても課題を解決するための根本が理解できていない場合提出結果が再度0点になることがあると考えられる．そのため，受講者が課題解決に必要なプログラミングの知識の理解が十分でない場合，理解できている受講者と比べ，提出結果が0点の割合が増える傾向があると考えられる．

4.3 メトリクスの算出手順

各メトリクスの算出手順を図1に示す．また，各手順を以下で説明する．ここで，R_k (k=0,1,2,⋯,N) をk番目に提出されたリビジョンと表す．

(1) t 分間のスナップショット群の抽出

OJS に記録された各ソースコードの提出日時を元に，R_0 から t 分間に提出されたソースコードと，課題 ID と受講者 ID が含まれる提出番号，評価結果 score，Syntax Error か否かを抽出する．

(2) ErrorCnt (t), ErrorTime (t), ErrorPct (t) の算出

R_i で提出結果が SyntaxError となり R_j で提出結果が SyntaxError でなくなった場合，ErrorCnt (t) を $j-i$ として算出する．また，R_i 提出結果が SyntaxError となり R_j で提出結果が SyntaxError でなくなった場合，ErrorTime (t) を R_jの時間 $- R_i$ の時間として算出する．最後に式 4.3.1 を用い ErrorPct (t) を算出する．

$$ErrorPct(t) \quad = \quad \frac{ErrorCnt(t)}{t\text{分間でのリビジョン数}} \tag{4.3.1}$$

(3) ZeroCnt (t), ZeroTime (t), ZeroPct (t) の算出

R_i で提出結果が 0 点となり R_j で提出結果が 0 点でなくなった場合，ZeroCnt (t) を $j-i$ として算出する．また，R_i 提出結果が 0 点となり R_j で提出結果が 0 点でなくなった場合，ZeroTime (t) を R_jの時間 $- R_i$の時間として算出する．最後に式 4.3.2 を用い ZeroPct (t) を算出する．

$$ZeroPct(t) \quad = \quad \frac{ZeroCnt(t)}{t\text{分間でのリビジョン数}} \tag{4.3.2}$$

(4) Revs (t) の算出

提出番号から t 分間の間に同じ受講者が同じ課題に対して提出したソースコードの内，最後のリビジョンを R_j とした時，Revs (t) は j とする．

(5) リビジョン間の修正箇所の検出

手順 (1) で抽出されたソースコードに対して，Ri と Ri+1 間の修正箇所を diff コマンドと-W オプションを用いて検出し，差分テキストファイルとして生成する．差分テキストファイルは Ri を基準として Ri+1 に,「追加」,「削除」,「変更」があれば，その行に各々「＜」,「＞」,「|」の記号が記すことでリビジョン間の差分を表す. 修正がなければ何も記されない．差分ファイルの生成を各受講者の各課題に対して i = 1 ∼ (Revs(t) − 1) まで繰り返す.

(6) DiffLine (1, t), AveDiff (t) の算出

手順 (5) で生成した差分テキストファイルに対し，wc コマンドを用いて各リビジョン間の修正行数を算出する．この結果と式 4.3.4 を用いて，DiffLine (1, t) を算出する．

$$DiffLine(s,t) \quad = \quad \frac{s\text{行以下の修正が続いた回数の最大値}}{Revs(t)-1} \tag{4.3.3}$$

同様に，各リビジョン間の修正行数と Revs (t) から，各受講者に対する AveDiff (t) を算出する．

$$AveDiff(t) \quad = \quad \frac{t\text{分間の修正行数の合計}}{Revs(t)-1} \tag{4.3.4}$$

(7) Freq(t) の算出

手順 (5) で生成した差分テキストファイルを基に，ソースコードの各行に対して ID を割り当て，R_i と R_{i+1} 間の各行に対する修正箇所と R_{i+1} と R_{i+2} 間の各行に対する修正箇所をそれぞれ対応づける (図 2)．また，途中のリビジョンから追加された行の場合は新たに ID を割り当てる．これを i = 1 ∼ (Revs (t) − 1) まで繰り返し，全リビジョンにおいて修正した行 ID とその修正回数を求める．行 ID の集合 ID = { $id_1, id_2, \cdots, id_k, \cdots, id_N$ } に対して，各 ID に対応する行の修正回数を用いて分散を求め，Freq(t) として算出する．

$$Freq(t) \quad = \quad \frac{1}{N}\sum_{k=1}^{N}(\frac{c_k}{s}-\mu)^2 \tag{4.3.5}$$

ここで，c_k は id_k の修正回数を表す．s,μ はそれぞれ次式を用いて算出する．

$$s \quad = \quad \sum_{k=1}^{N} c_k \tag{4.3.6}$$

$$\mu \quad = \quad \frac{1}{N}\sum_{k=1}^{N}\frac{c_k}{s} \tag{4.3.7}$$

4.4 予測方法

算出された各メトリクスを説明変数，完答までにかかった時間を目的変数としてランダムフォレストを用い機械学習を行う．また，目的変数が 2 値でない時間のため，回帰分析を行う．先行研究で提案されていたメトリクスのみで機械学習した結果と，本研究の提案メトリクスを加えて機械学習した結果を比較し，予測精度の差や各メトリクスが予測に有用であるかを検証する．

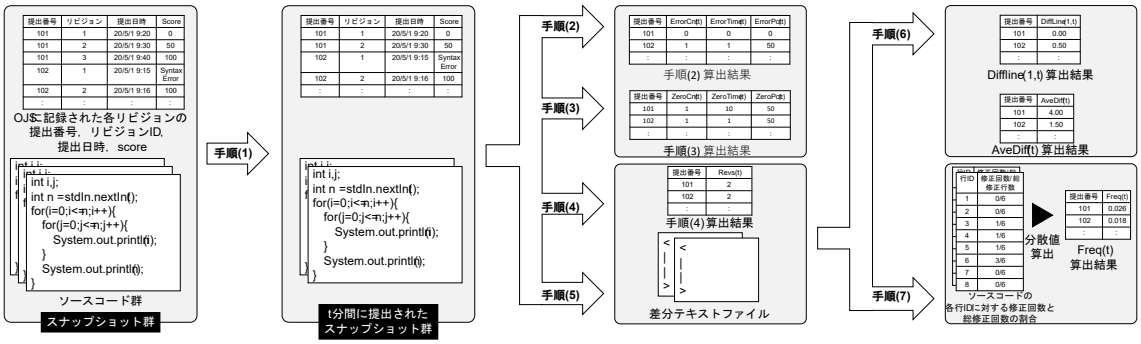

図 1 各メトリクスの算出手順

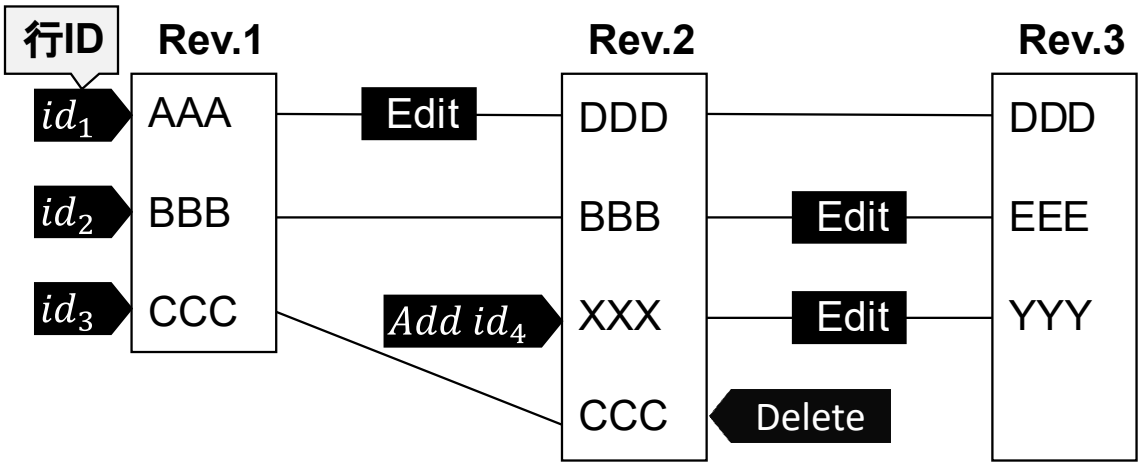

図 2 ID の割り当て

5 実験

5.1 データセット

奈良工業高等専門学校の2年生が受講する授業であるプログラミングIで収集されたデータを分析対象とする．Learning Management System (LMS) の1つであるMoodleと連携してOJSの機能を提供するCodeRunnerを用いて演習の出題と回収，評価を行う．CodeRunnerは学生が提出したソースコードを課題ID, 受講者ID, リビジョン番号，提出日時，scoreと共に記録する．なお，初回提出で完答した場合，完答までにかかる時間が0分となるため除外する．本研究では，2017年7月21日～9月1日に収集された計2,529回（11演習，述べ364人）の提出ソースコードのデータを対象に実験を行う．本研究の最終的な目的は授業時間中に理解が不十分な受講者を予測し，教員の指導のサポートをすることである．また，本研究では夏季休暇期間の課題の提出履歴を分析対象とした．なぜなら，授業時間内とは違い近くに他の受講者や教師がいないため，対面での講義よりも人に相談するハードル高いと考えられる．そのため，課題を完答するために必要なプログラミングの知識を理解できていない受講者は自力で理解しないといけないため，対面での講義より理解できるまで長い時間を要すると考えられる．また，他人から答えを教えられることも少ないと考えられるため，理解できていない受講者の特徴がよく表れると考えた．これらの理由から夏季休暇期間の課題の提出履歴を分でき対象とした．

また，全て学習用データとして使用すると学習結果の検証が出来なくなるため，データセットを指定した割合でランダムに分割する．本研究では学習用データ9割テスト用データ1割で分割し，完答までの全ての提出履歴で学習させた．また，1回の学習では分割されるパターンによっては学習結果に極端な偏りが生まれてしまう可能性があるため，実験結果では100回学習させその結果の平均を示す．

5.2 実験方法

提案したメトリクスの有用性を評価するために，各メトリクスをランダムフォレストを用い学習させる．

本研究では，演習の時間内に受講者が課題を完答するまでの時間を予測することを目的としている．そのため，課題を開始してからの時間tについて完答までにかかる時間を予測し，各メトリクスが完答までにかかる時間の予測のために有用であるか検証することが望ましい．そこで，本研究では対象講義1コマ分の半分の時間である45分をtの最大とし，5分刻みのテスト用データを学習モデルに適用し比較する．また，実際の現場で使用するのは$t = 20$程度まで十分だと考えられる．しかし本研究ではメトリクスの評価のため，$t = 45$までの予測精度

表 1 予測値の MSE

t	MSE_train	MSE_test	add_MSE_train	add_MSE_test
5	36271539	36650190	32736445	37774450.49
10	33158189	25458130	27283677	31705529.24
15	38525595	34683519	32592960	38890147.6
20	37510131	38754220	33637773	39143287.91
25	35456512	28256936	32881369	27138398.4
30	35761203	35027698	33960850	24616830.81
35	36957177	34998367	34283785	32929179.27
40	37125711	38481722	34643896	29169390.7
45	36206817	31910295	33236667	30176148.81
full	12544666	12936057	12849138	11634037.93

を求める．また，実際の演習中では完答までにかかる時間がより長い受講者から指導することが望ましいと考えたため，検証は予測結果の細かな差が表れる平均二乗誤差 (MSE) を主として評価する．

5.3 結果

最初に先行研究で提案されていたメトリクスだけで学習し，各時間での MSE を求めた．また，各メトリクスが予測にどの程度重要であったかを求めた．MSE を求めた結果を表 1 に示す.

1 列目は課題を開始してからの時間を表している．t が 5 の行は課題を開始してから 5 分までの提出履歴を元に算出した各メトリクスを学習モデルに適用した予測結果である．また，full の行は課題を開始してから完答までの全ての提出履歴を元に算出した各メトリクスを学習モデルに適用した予測結果である．MSE_train の列は学習用データに含まれている提出番号の t 分までの提出履歴を元に先行研究で提案されていたメトリクスのみを算出したデータで予測した結果の MSE を表している．また，MSE_test の列はテスト用データに含まれている提出番号の t 分までの提出履歴を元に先行研究で提案されていたメトリクスのみを算出したデータで予測した結果の MSE を表している．また，4,5 列目は 2,3 列目に本研究で提案したメトリクスを追加したデータで予測した結果の MSE を表している．提出番号には受講者 ID と課題番号の情報が含まれており，どの受講者のどの課題に対する提出履歴かを識別することができる．各メトリクスの重要度を表 2 に示す.

1 列目は先行研究で提案されていたメトリクスを示しており，2 列目は 1 行目のメトリクスだけで学習をした際予測にどれだけ重要かを表す重要度を示している．3 列目は先行研究で提案されていたメトリクスと本研究で提案したメトリ

表 2 各メトリクスの重要度

metrics	重要度	add_metrics	重要度
Freq (t)	0.3314	Freq (t)	0.2283
DiffLine (1,t)	0.1558	DiffLine (1,t)	0.1035
DiffLine_prime (1,t)	0.1379	DiffLine_prime (1,t)	0.1110
Revs (t)	0.1433	Revs (t)	0.0913
AveDiff (t)	0.2316	AveDiff (t)	0.1550
		ErrorCnt (t)	0.1001
		ErrorPct (t)	0.1137
		ZeroCnt (t)	0.0384
		ZeroPct (t)	0.0588

表 3 各メトリクスと完答時間の相関

	相関係数	p 値
Freq(t)	-0.153	0.028∗
DiffLine(t)	0.102	0.145
DiffLine_prime(t)	-0.043	0.538
Revs(t)	0.183	0.008∗∗
AveDiff(t)	0.036	0.607
ErrorCnt(t)	0.156	0.025∗
ErrorPct(t)	0.002	0.980
ZeroCnt(t)	0.160	0.021∗
ZeroPct(t)	0.177	0.011∗

クスを示しており, 4 列目は 3 列目のメトリクスで学習をした際の重要度を示している.

また, 全体の 98% 以上が 0 点でなくなってから 1 分以内に完答していたため, ZeroTime (t) が予測したい値とほぼ同値を取っていた. そのため, 完答する前の完答までの時間の予測には適していなかった. ErrorTime (t) についても同様だった. そのため以降は ZeroTime (t) と ErrorTime (t) 以外の各メトリクスについて結果を説明する. 表 1 より t が 5 ∼ 20 の時の MSE_test と add_MSE_test を比較すると, 全ての t で MSE_test の方が値が小さくなっていることが分かる. MSE は予測値が実際の値からどれだけ離れているかを示しているので, 20 分までの提出履歴を対象として予測させた場合は先行研究のメトリクスだけの方が予測精度が高いことが分かる. 逆に, t が 25 ∼ full の時の MSE_test と add_MSE_test を比較すると, 全ての t で add_MSE_test の方が値が小さくなっていることが分かる. そのため, 30 分以降の提出履歴を対象として予測させた場合は提案メトリクスを追加した時の方が予測精度が高いことが分かる. また, ランダムフォレストの重要度とは, 各メトリクスが予測にどれだけ寄与しているかを割合で示しているものなので, 表 2 より追加した 4 つのメトリクスが予測に使用されていることが分かる.

各メトリクスと完答までにかかった時間の相関, および, 両側 t 検定の結果を表 3 に示す.

Freq (t) と Revs (t) や ErrorCnt (t) , ZeroCnt (t) , ZeroPct (t) はいずれも弱い正の相関

表 4 1000 分以上のデータを除外した時の MSE

t	MSE_train	MSE_test	add_MSE_train	add_MSE_test
5	13638	14035	13885	16275
10	12985	13576	14299	13100
15	14224	14070	16227	16275
20	14127	15240	17636	17443
25	14103	15080	18092	12535
30	14723	14367	18103	21239
35	15892	14969	20056	18969
40	15818	14262	19973	18444
45	15810	14644	19429	23345
full	3458	4014	3292	3471

表 5 1000 分以上のデータを除外した時の各メトリクスの重要度

metrics	重要度	add_metrics	重要度
Freq(t)	0.5639	Freq(t)	0.5276
DiffLine(t)	0.0496	DiffLine(t)	0.0340
DiffLine_prime(t)	0.1029	DiffLine_prime(t)	0.0718
Revs(t)	0.0968	Revs(t)	0.0469
AveDiff(t)	0.1868	AveDiff(t)	0.1386
		ErrorCnt(t)	0.0330
		ErrorPct(t)	0.0700
		ZeroCnt(t)	0.0304
		ZeroPct(t)	0.0479

で，有意な相関が見られた ($p < 0.05$).

また，表 1 より MSE の値があまりにも大きくなっていることが分かる．これは本研究で使用したデータの問題と考えられる．

本研究では講義時間内のデータではなく夏季休暇期間の課題の提出履歴を対象としたため，296 回の提出履歴の内，課題開始から完答までの時間が 1000 分を超える提出履歴が 24 回，最長で 49577 分の提出履歴があった．受講者が課題の完答までにかかる時間の予測をし，課題の完答までに時間のかかる受講者の特徴を調べたいため解答時間が極端に長い提出履歴を一概に除外してしまうと本来取りたいデータが取れなくなってしまう可能性があるため実験では除外しなかった．しかし，これら 1000 分以上のデータが予測精度を低下させる原因になったと考えられるため，これらのデータを除外して同じ条件で実験した．その結果得られた MSE を表 4 に，各メトリクスの重要度を表 5 に，相関係数を表 6 に示す.

表 4 より予測精度が大幅に向上したことが分かる．また，表 1 とは違い，提案メトリクスを追加しても特定の t を境に，予測精度の優劣が逆転することはなかった．表 5 より，表 2 と比べ提案メトリクスの重要度は下がっていることが分かるが，全体の 18 % 程度は予測に影響を与えていることが分かった．表 3 と表 6 より，

表 6 1000 分以上のデータを除外した時の各メトリクスと完答時間の相関

メトリクス	相関係数	p 値
Freq(t)	-0.161	0.028*
DiffLine(t)	0.018	0.803
DiffLine_prime(t)	0.011	0.881
Revs(t)	0.194	0.008**
AveDiff(t)	0.297	0.000**
ErrorCnt(t)	0.086	0.241
ErrorPct(t)	0.142	0.053
ZeroCnt(t)	0.188	0.010**
ZeroPct(t)	0.172	0.018*

AveDiff(t) は有意な相関が見られるようになり, ErrorCnt(t) は有意な相関が見られなくなった.

5.4 考察

講義時間内の提出履歴を対象とした場合は講義時間内に完答できない人がいるため時間の予測という観点からは正確な学習結果を得られない可能性があるが, 極端に長い時間がかかるデータはなくなるため, 本研究とは違う結果が出る可能性があると考えられる.

また, 結果で述べた通り t によって既存メトリクスのみで予測した場合と, 提案メトリクスを追加して予測した場合, 予測精度の優劣が変化した. この原因について考察する.

ErrorPct (t) に関しては高いほどプログラミング言語が理解できていないため完答までの時間がかかると予想した. しかし, 1 回目の提出で SyntaxError を起こして 2 回目の提出で完答した人と, 1 回目の提出だけで SyntaxError を起こして, 10 回目の提出で完答した人を比べる. これは前者の方が課題の解決に必要なプログラミングの知識を理解していると考えられるが, 前者の方がメトリクスの値が高くなってしまう. このような場合では, 想定していた値を取らないため, ErrorPct (t) の改善か他のメトリクスで補う必要がある. また, 表 6 より p 値が 0.053 と 0.05 より大きい値になっていることから完答までにかかる時間の予測には役に立つかは分からなかった.

ZeroPct (t) に関しては高いほど問題の解決方法が理解できていないため, 完答までの時間がかかると予想した. 結果として表 5 より重要度の大きさは 5 番目だったため他のメトリクスと比べ中程度に重要であることが示唆され, 表 6 より p 値と相関係数からは予測に有用であることが示唆された. また, OJS の正誤判定のアルゴリズムの仕様上, 解答のプログラムの処理自体はあっているが, 出力部分でスペルミスや空白の誤挿入などをしているため指定されたフォーマット通りの出力となっていないケースが見られた. こういったケースでは課題の解決に必要

なプログラミングの知識を理解しているが，0点を短時間で連続して提出していたためZeroPct (t) が想定よりも大きな値をとることがあった．しかし，こういったケースは全体で7回ほどしかなかったため，有意な相関となったと考えられる．またさらなる改善のために，課題の内容や受講者のグループによっては上記のような振る舞いをする受講者が増える可能性があるので，上記のような振る舞いを識別するか，OJSの正誤判定システムを改善するとより良い精度が得られると考えられる．

ErrorCnt (t) に関しては高いほどプログラミング言語が理解できていないため完答までの時間がかかると予想した．結果として表5より重要度の大きさは8番目だったため，他のメトリクスより予測に重要でないことが示唆され，表6よりp値からも予測に有用でないことが示唆された．

ZeroCnt (t) に関しては高いほど課題の解決に必要なプログラミングの知識の理解が十分でないため完答までの時間がかかると予想した．結果として表5より重要度が最も小さかったため，他のメトリクスより予測に重要でないことが示唆されたが，表3よりp値と相関係数からは予測に有用であることが示唆された．

今回の実験で使用するメトリクスを変更することによって，各時間において予測精度が異なることが分かった．そのため，各時間において完答までの時間の予測に向いているメトリクスを特定することで今後予測精度が向上していくと考えられる．

6 結論

本研究では，理解が不十分なため完答までに時間がかかる受講者の予測を目的に，受講者が提出したソースコードのスナップショットから受講者が完答までにかかる時間を予測する手法を提案し，その有用性を検証した．提案手法は，先行研究で提案されていたメトリクスに，予測に役立つと考えられるメトリクスを加え，ランダムフォレスト用いて機械学習を行い受講者が完答までにかかる時間を予測することである．実験では夏季休暇中課題の提出履歴を学習データとし，各提出時間において完答時間を予測した．実験の結果，各時間によって既存メトリクスのみで予測した場合と，提案メトリクスを追加して予測した場合，予測精度の優劣が変化した．そのため，時間に合わせて使用する学習モデルを変えることで各受講者の完答までにかかる時間を予測し，指導することで教育効果を高められると考えられる．また，OJS に各受講者の完答までにかかる時間を動的に予測・表示する機能を追加することで，支援が不要な受講者の作業を妨げることなく完答までに時間のかかる受講者に対して優先的な支援が可能になる．

今後の課題として予測精度の向上が挙げられる．本研究では夏季休暇中課題を学習対象のデータとしたため，完答までに極端に時間のかかる受講者が複数存在した．そこで 1000 分以上の提出履歴データを除外することで予測精度が向上した．データを加工することで予測精度が向上したため，今後講義時間内の提出履歴や，更に短い時間の提出履歴に絞るなど別のデータを学習対象とすることで精度の向上が期待される．また，提案メトリクスを追加することで予測精度が変化したことから，受講者の特性を表す新たなメトリクスを追加することで予測精度を向上させることができると考えられる．特に，Freq (t) と Revs (t) や AveDiff (t) , ZeroCnt (t) , ZeroPct (t) には受講者が課題を完答するまでにかかる時間と有意な弱い正の相関が見られた．これはこれら 5 つのメトリクスが完答までにかかる時間の予測に有用であることを示唆している．そのためこれらのメトリクスを中心とし新たに学習モデルを作ることで更に精度の良い予測ができる可能性がある．また，学習に使用するメトリクスの差異により時間によって精度の優劣が逆転した．このことから各時間帯において学習に使用するメトリクスや各メトリクスの重要度を変えることで予測精度が向上すると考えられる．

謝辞

本論文の執筆及び本研究を進めるにあたり, 多くの方々に御指導, ご協力を賜りました. ここに謝意を添えて御名前を記させていただきます.

指導教員である上野秀剛准教授には御多忙の中, 研究に対する助言, 論文及び発表資料の添削などについて多くの助力を頂きました. 心より感謝申し上げます.

内田真司教授には, 査読やプログラミングIの講義データの利用許可を頂きました. 心より感謝申し上げます.

本間啓道准教授には, 研究に対して御指摘と御助言をいただきました. 心より感謝申し上げます. ありがとうございました.

参考文献

[1] 島田大樹, 彌冨仁:畳み込みニューラルネットワークを使った授業映像中の聴講者の状態推定システムの構築と特徴量獲得に関する検討, 知能と情報, Vol.29, No.1, pp.517–526 (2017).

[2] B. Baradwaj and S. Pal: Mining Educational Data to Analyze Students ' Performance, International Journal of Advanced Computer Science and Applications, Vol.2, No.6, pp.63–69 (2011).

[3] S. Pal: Analysis and Mining of Educational Data for Predicting the Performance of Students, International Journal of Electronics Communication and Computer Engineering, Vol.4, pp.1560–1565 (2013).

[4] N. Tselios, A. Stoica, M. Maragoudakis, N. Avouris and V. Komis: Enhancing user support in open problem solving environments through Bayesian Network inference techniques, Educational Technology & Society, Vol.9, No.4, pp.150–165 (2006).

[5] 植野真臣:e ラーニングにおけるデータマイニング, 日本教育工学会論文誌, Vol.31, No.3, pp.271–283 (2007).

[6] 加藤利康:授業支援システムにおける学習分析の展開, 研究報告コンピュータと教育, Vol.2014, No.23, pp.1–7 (2014).

[7] 加藤利康, 石川孝:プログラミング演習のための授業支援システムにおける学習状況把握機能の実現, 情報処理学会論文誌, Vol.55, No.8, pp.1918–1930 (2014).

[8] P. Ihantola, A. Vihavainen, A. Ahadi, M. Butler, J. Börstler, S. H. Edwards, E.Isohanni, A. Korhonen, A. Petersen, K. Rivers, M. Á. Rubio, J. Sheard, B. Skupas, J. Spacco, C. Szabo, D. Toll: Educational data mining and learning analytics in programming: Literature review and case studies, Proceedings of the 2015 ITiCSE on Working Group Reports, pp.41–63 (2015).

[9] K. Fujiwara, K. Fushida, H. Tamada, H. Igaki and N. Yoshida: Why Novice Programmers Fall into a Pitfall?: Coding Pattern Analysis in Programming Exercise, Fourth International Workshop on Empirical Software Engineering in Practice (IWESEP 2012), pp.46–51 (2012).

[10] 藤原賢二, 上村恭平, 井垣宏, 吉田則裕, 伏田享平, 玉田春昭, 楠本真二, 飯田元:スナップショットを用いたプログラミング演習における行き詰まり箇所の特定, コンピュータソフトウェア, Vol.35, No.1, pp.3–13 (2018).

[11] 槇原絵里奈, 井垣宏, 吉田則裕, 藤原賢二, 飯田元:プログラミング演習における探索的プログラミング行動の自動検出手法の提案, コンピュータソフトウェア, Vol.35, No.1, pp.110–116 (2018).

[12] 藤原理也, 田口浩, 島田幸廣, 高田秀志, 島川博光:ストリームデータによる学習者のプログラミング状況把握, 第 18 回データ工学ワークショップ, pp.1–6 (2007).

[13] M. Jadud: Methods and tools for exploring novice compilation behaviour, ICER2006 - Proceedings of the 2nd International Computing Education Research Workshop, pp.73–84 (2006).

[14] A. Ahadi, R. Lister, H. Haapala and A. Vihavainen: Exploring Machine Learning Methods to Automatically Identify Students in Need of Assistance, Proceedings of the Eleventh Annual International Conference on International Computing Education Research, ICER '15, pp.121–130 (2015).

[15] P. Blikstein: Using Learning Analytics to Assess Students' Behavior in Openended Programming Tasks, Proceedings of the 1st International Conference on Learning Analytics and Knowledge, pp.110–116 (2011).

[16] 松永賢次:導入プログラミング教育におけるオンラインジャッジシステムの活用の試み, 情報科学研究, Vol.31, pp.25–41 (2010).

[17] 岩本舞, 中村真人, 小島俊輔, 中嶋卓雄:不正コピー検出手法を備えたオンラインジャッジシステムの開発, 情報処理学会論文誌教育とコンピュータ, Vol.1, No.4, pp.38–47 (2015).

[18] 古谷勇樹, 林真史, 山本隆弘, 長尾和彦:オンラインジャッジシステムと連携可能な Moodle プラグインの実装と比較, 情報科学技術フォーラム講演論文集, Vol.14, No.3, pp. 89–94 (2015).

[19] V. Karavirta, A. Korhonen and L. Malmi: On the use of resubmissions in automatic assessment systems, Computer science education, Vol.16, No.3, pp.229–240 (2006).

[20] V. Karavirta, A. Korhonen and L. Malmi: Different learners need different resubmission policies in automatic assessment systems, Proceedings of the 5th Annual Finnish/Baltic Sea Conference on Computer Science Education, pp.95–102 (2005).

[21] P. Ihantola, T. Ahoniemi, V. Karavirta and O. Seppälä: Review of Recent Systems for Automatic Assessment of Programming Assignments, Proceedings of the 10th Koli Calling International Conference on Computing Education Research, pp.86–93 (2010).

[22] M. Ben-Ari: Constructivism in computer science education, Journal of Computers in Mathematics and Science Teaching, Vol.20, No.1, pp.45–73 (2001).

[23] 大野優, 上野秀剛, 内田眞司:ソースコードのスナップショットに基づいた無作為修正者の検出, 信学技報教育工学研究会, Vol.118, No.214, pp.53-58, September 2018.

[24] L. Breiman: Random Forests, Machine Learning Vol.45, pp.5–32 (2001).