CK-002

プログラミング授業での決定木を用いたドロップアウト原因の可視化
Visualizing Cause of Dropout in Programming Class with Decision Tree

千枝 睦実 † 大枝 真一 ‡
Mutsumi Chieda Shinichi Oeda

1. まえがき

学習者，特に学生からログデータなどを収集することができる Intelligent Tutoring System(ITS) や学習管理システムは，教育の分野で広く使われている．Educational Data Mining(EDM) は，これらの教育システムによって収集された大量の電子データから有用な情報を発見することを目的としている．EDM は，さまざまな教育情報システムのデータを探索するための技術を用いた学際的な研究分野である [1].

本研究では，学校でのプログラミング教育の初期に焦点を当てる．プログラミング教育の初期で学ぶ事項は，各プログラミング言語によらない共通の概念やアルゴリズムなどに関するものであり，学生の理解度や，授業に追従できていない (ドロップアウト) 原因の把握はきわめて重要である．

しかしながら，ほとんどの学校では，教師が一度に 30 人以上の学生を一度に教えるため，教師が各学生の理解度を把握したり，各学生に適した指導を提供することは困難である．学生がスキルを習得したかどうかを確認する方法として，定期的な試験が挙げられる．しかし，このような試験には時間がかかり，採点作業は教師の負担となるため，頻繁には実施できない．そのため，学生の理解度のチェックが遅れてしまうことがある．

この問題を解決するために，ソースコードを自動的に分析して，それらが内包する特徴を抽出することを提案する．本研究では，学生のソースコードの特徴を決定木モデルを用いて可視化することで，ドロップアウトする可能性の高い，危険な学生とそうでない学生を分ける原因を明らかにする．

2. 先行研究

学生の監視と支援は，多くの教育機関で非常に重要だと考えられている．教師が授業に追従できていない学生を早い段階で検出できれば，その学生が授業から脱落する前に対策を講じることができるためである．したがって，教師が重点的に注意を払うべき学生を把握できるように，授業間に脱落する危険性がある学生を予測する必要がある．

プログラミング授業において，ログデータを使って学生の行動を確認できるよう，我々はソースコードの編集履歴を取得できるログ取得システムを開発し，44 人の学生から定期テストでのソースコードを得た．潜在的なドロップアウトを予測するとき，教師あり学習で評価器を作成するのが一般的である [2, 3]．しかし，このようなデータは，データセットのサイズが小さい場合に学習が困難である．また，ログデータの特徴は，複数の演習があるかどうか，またはどれだけ詳細に説明したかなど，授業の内容によっても異なる．同様な環境において，コマンドログや，キーログのみを用いてプログラマのスキル評価を行った実験では，修飾キーの打鍵数と成績との間に正の相関が見られたものの，それ以上の知見は得られなかった [4].

我々は本研究を行うにあたって，ソースコードの解析や評価方法についてすでに類似の研究や手法がないか調査を行った．いくつかの論文や既存のサービスの中で，本研究と関連性が大きいものを述べる．

2.1. データ収集

データ収集の観点で参考にした論文 [5] では，バージョン管理システム git において，ソースコードの指定した行に関する情報を取得する機能である blame を用いて学習に使用するデータを取得していた．この論文では，収集したデータに対して，断片的なソースコードの解析に対応したパーサーを用いて抽象構文木を作成していた．また，学習するデータは，数行程度のコードを 1 サンプルとして扱い，15 人分のデータセットに対して，約 60,000 個の特徴を抽出していた．しかし，パーサーを用いた本研究用の処理環境は，依存関係の問題のために現時点では開発していない．

2.2. 既存のプログラム評価方法

既存の就職・転職活動サービスである paiza [6] や testdome [7] は，就職・転職希望者がプログラミング課題を Web 上で解答すると，その結果および過程が検証され，就職・転職希望者の能力をサービスを利用する企業に保証するサービスである．このサービスでは，プログラマーの能力評価において，「コードの中身自体の評価は行わず，複数のテストケースに対する正答率，コードの記述時間，実行速度，メモリ消費量により評価を行う」としている．ソースコード自体の評価は企業面接時に行うものとしており，このときの判断基準は，企業の視点に依存する．したがって，ソースコード自体の自動的な評価は実用に耐え得るほど信頼性が高くないものと考えられる．

2.3. コードレビュー

近年，自動的に収集されるデータ群を，ツールなどを用いて解析する現代的なコードレビューの方法は多く研究されている [8]．しかし，本研究は以下の点において，これまで提案されてきた方法とは異なる．

- ある程度経験を積んだプログラマではなく，プログラミング初学者を対象としている．
- すでにあるプロジェクトに追加されるパッチではなく，一から作られるソースコードを評価する．

†木更津工業高等専門学校 制御･情報システム工学専攻, Advanced Cource of Control and Information Engineering, National Institute of Technology, Kisarazu College
‡木更津工業高等専門学校 情報工学科, Information and Computer Engineering, National Institute of Technology, Kisarazu College

- ソースコードの特徴を抽出する際，構文解析的に行うのではなく，ソースコード全体の特徴を抽象的に捉える．

3. 手法

3.1. 特徴抽出

本研究では，学生のテストの答案となるソースコードを用いて決定木を生成する．収集したデータを決定木に適用するためには，すべての特徴が数値もしくはカテゴリカル変数である必要がある．そのため，収集したソースコードやログを互いに関連させて，いくつかの特徴に変換する．抽出する特徴は，ソースコードの著者を特定する実験で有効だったものの中で，特別なパーサーを用いずにに実装できるものを用いた [5]．抽出した特徴を表 1 に示す．

表 1　抽出した特徴

特徴	図中の表記
自作関数定義の出現頻度	ln_functions
自作関数名の平均長さ	mean_function_name
関数の引数の数	n_function_args
変数名の平均長さ	mean_variable_name
インデント方式 (BSD/kernel)	indentation_style
空行の出現頻度	ln_blank_lines
1 行あたりの文字数	mean_letters_of_line
1 行の文字数のばらつき	std_letters_of_line
インデント文字	tab_space_style
“do” の出現頻度	do
“while” の出現頻度	while
“for” の出現頻度	for
“if” の出現頻度	if
“else if” の出現頻度	elseif
“else” の出現頻度	else
“switch” の出現頻度	switch
自作関数の平均長さ	mean_function_lines
1 行コメントの出現頻度	oneline_comment
複数行コメントの出現頻度	multiline_comment

3.2. 決定木

本研究では，教育の現場で使用されることを想定するため，単なる機械学習による分類・予測だけでなく，機械学習に詳しくない人でもわかりやすく可視化することが不可欠となる．決定木 [9] は，木構造によって表される予測モデルである．予測結果に対する原因が，人が把握しやすい形で出力される長所を持つため，本研究に適している．

決定木を生成する際は，以下の手順に従う．

1. 全データをルートに入れる．
2. ジニ不純度 (式 1) が最小になるようにデータを分割する．

$$I_G = 1 - \sum_{i=1}^{N} p(i|t)^2 \qquad (1)$$

3. ノード内のジニ不純度がある値を下回るか，木がある深さに達するまで，分割された各ノードで手順 1，2 を繰り返す．

決定木は，データを混じりけなく分割するために学習した規則を木構造の形で可視化する．決定木の各ノードは，データ分割の条件，ジニ不純度，属するサンプル数，各クラスのサンプル数，およびそのノードで最も多いサンプルのクラスを示す．

3.3. ランダムフォレスト

ランダムフォレスト [10] は，アンサンブル学習の代表的な手法である．アンサンブル学習とは，決定木などの基本モデルを集約してより優れた予測性モデルを作成する方法である．ランダムフォレストは，汎化性能の高いモデルを作成するためにさまざまな決定木をまとめることで，決定木の過学習しやすい性質を克服する．

決定木は，非常に不規則なパターンを学び，その木が学習した特徴量に過学習する傾向がある．すなわち，低いバイアスが，非常に高い分散を持っている状態である．したがって，ランダムフォレストは，分散を減らすことを目的として，同じトレーニングセットの異なる部分で訓練された複数の比較的深い決定木を平均化する方法である．この方法により，最終的なモデルのパフォーマンスを大いに高めることができる．

本研究では，ランダムフォレストを適用する過程で生成された複数の決定木から，ドロップアウトの危険がある学生とそうでない学生とを分けた原因を可視化する．

4. 計算機実験

4.1. 実験条件

決定木に対して前節の方法で前処理されたデータを学習させ，学生の評定に影響する隠れた要因を可視化する．また，生成された決定木モデルに，評定が未知のソースコードの評点を予測する分類タスクをさせることで，モデルの正当性を確認する．

前処理された CSV 形式の入力に対して，評定の予測値と，その過程で生成された決定木による分類モデルを出力する学習・予測プログラムを作成した．このとき，収集された 44 人のソースコードのうち，ランダムな 35 人分を訓練データに，残った 9 人分を精度評価用データとした．予測結果は，5 点の評定において，3 点以上を「安全」，2 点以下を「危険」とした 2 種類の評定カテゴリとして出力する．ランダムフォレストに与えるハイパーパラメータは，あらかじめ 100 回の訓練・予測の試行で最適化されたものを使用した．

本実験で特徴を抽出するソースコードは，木更津工業高等専門学校情報工学科 2 年,「プログラミング基礎 I」の 2019 年 6 月に行われた定期テストにおいて，試験時間中に収集されたものを使用する．課題の内容は，C 言語で以下のプログラムを実装することである．

1. Body Mass Index(BMI) を求める関数を持つプログラム．
2. 1 以上 1000 以下のすべての数の和を求めるプログ

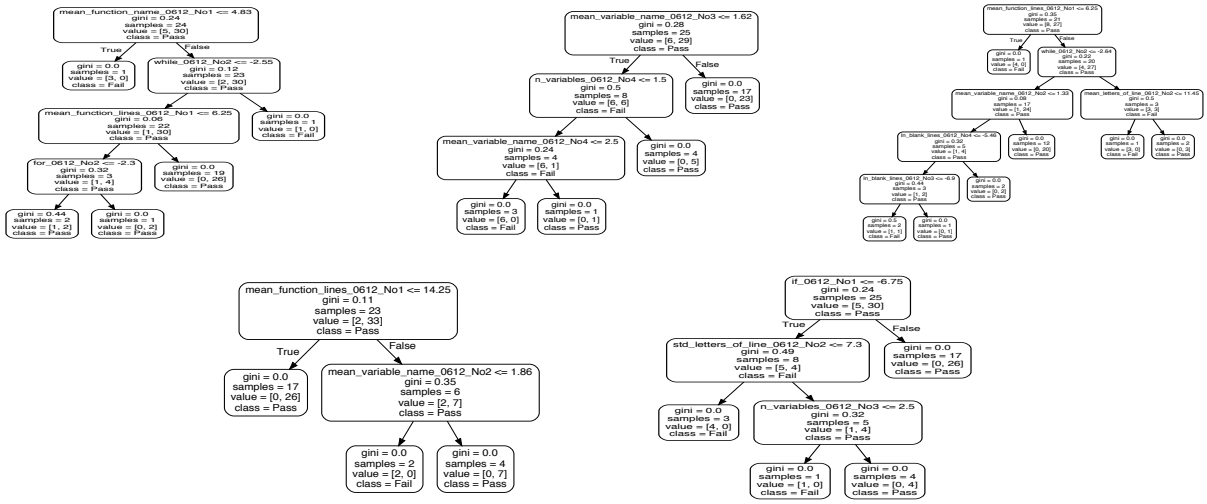

図 1　生成された 5 本の決定木

ラム (小問 1) と，ある条件下でのみ和を求めるプログラム (小問 2).

3. 題意に沿って，2 次元配列からのランダム選択を行うプログラム (小問 1) と，前問のプログラムを拡張する課題 (小問 2).

4. 題意に沿って，条件分岐を用いた簡単な関数を持つプログラム.

収集したソースコード群からデータを抽出する処理を行い，作成した訓練用データ，およびテスト用データを用いて計算機実験を行う.

4.2. 評価指標

出力された予測結果がテスト用データをどれほど正確に予測しているかを示す最終的な指標には，F 値を用いる．2 値カテゴリを予測するモデルの精度を評価する方法に，Precision と Recall がある．前者は誤検知のしにくさ，後者は見逃しのしにくさを評価するが，どちらか片方のみでは全てのデータに対して同じ評定を予測したモデルが良好なモデルとされてしまう．したがって，これらの調和をとった F 値を指標とする.

Precision，Recall，および F 値は，次のように表される．式中の TP は真だと予測して真だったサンプル数，FP は真だと予測して偽だったサンプル数，FN は偽と予測して真だったサンプル数を表す.

$$Precision = \frac{TP}{TP + FP} \quad (2)$$

$$Recall = \frac{TP}{TP + FN} \quad (3)$$

$$F - measure = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (4)$$

F 値は，0 以上 1 以下の値をとり，予測結果がテスト用データと完全に一致していた場合に 1 となり，予測が外れるほど値は小さくなる.

4.3. 実験結果

実験の結果，決定木は 5 本生成された．生成された決定木群を図 1 に示す．実験時に出力された予測結果の精度は，F 値で 1.000 であり，真値と予測値はすべて一致していた．この時の真値と予測値の比較を図 2 に示す．また，訓練データと精度評価用データの組み合わせを変えて 10 回実験した際の F 値の平均は，0.867 であった．予測における各特徴の重要度を図 3 に示す.

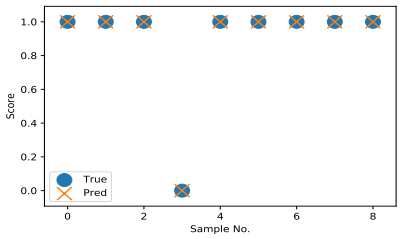

図 2　真値と予測値の比較

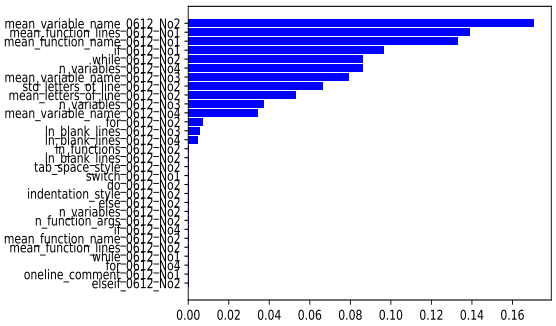

図 3　特徴重要度

5. 考察

生成された決定木と特徴重要度を読むことで，学生が誤答をした抽象的な原因を探る．図 3 を見ると，変数名の平均長さ，自作関数の平均長さ，自作関数名の平均長さなどの特徴が大きく決定木の生成に寄与したことがわかる．そこで，それらの特徴が分割条件となっている木の一つを図 4 に示す．

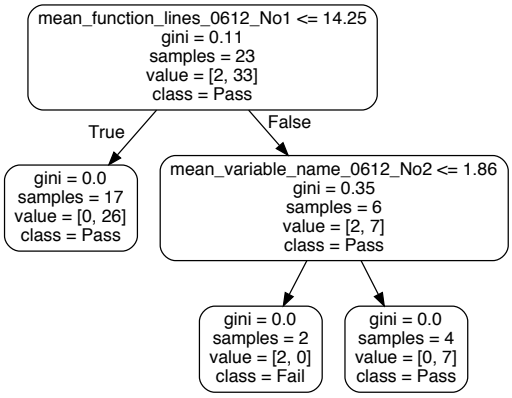

図 4　生成された決定木の一本 (拡大)

それらの特徴を図 4 と照らし合わせると，図 4 の上から 2 段目右のノードにある，「課題 2 における変数名の平均長さ ≤ 1.86」の条件によって，「危険」の評定を受けた学生とそうでない学生が分けられていることがわかる (分割前は危険な学生，通常の学生がそれぞれ 2 人，7 人いたところ，分割後は成績に危険のある学生，そうでない学生でそれぞれ 2 人，0 人のノードと 0 人，7 人のノードに分けられている)．

以上のことから，課題 2 では `i` や `x` などの意味を持たない 1 文字の変数名を用いた学生の多くが正答に失敗したことが読み取れる．そこで，誤答した学生に対して変数に意味のある名前をつけるなどのアドバイスを与えることが効果的だと考えられる．

同様に，課題 1 における自作関数の平均長さについても図 4 と照らし合わせると，図 4 に「自作関数の長さ $\leq$14.25」と示されている．この条件によって成績の良い学生が多く分割されたことから，関数を短くまとめることができた学生が正答しやすかったことが読み取れる．そのため，誤答した学生に対して関数を短くまとめられるよう，コーディングする前に設計をするなどのアドバイスが考えられる．

なお，これらの木を生成した実験での予測は，すべて真値と一致していたが，サンプル数が 44 人分の非常に小規模な実験であったため，新たなデータを確保し，規模を大きくした上で，改めて決定木モデル群の妥当性を検証する必要がある．

6. まとめ

本研究では，決定木分類を用いて学生のスキルの予測と学生の評定を分けた要因の可視化を行うことを提案した．今後は，データ収集や特徴抽出の段階を重点的に改善する．そこで，具体的な今後の課題として，4 つのことが挙げられる．

- 課題学習における，学生がつまづいている要素の可視化．
- 授業からのソースコード収集．
- ログデータ解析による特徴の抽出．
- 時系列でのコードの変化の可視化．

謝辞

本研究は，JSPS 科研費 19H01728 の助成を受けたものです．

参考文献

[1] Toon Calders, Mykola Pechenizkiy, “Introduction to The Special Section on Educational DataMining”, SIGKDD, Vol.13, Issue.2, pp.3-5, 2011.

[2] Gerben Dekker, Mykola Pechenizkiy, Jan Vleeshouwers, “Predicting students drop out: a case study”, Proceedings of the 2nd International Conference on Educational Data Mining (EDM2009), pp.41-50, 2009.

[3] DiyiYang, Tanmay Sinha, David Adamson, Carolyn Penstein Rose, “Turn on, tune in, dropout: anticipating student dropouts in massive open online courses”, Proceedings of the 2013 NIPS Data-driven Education Workshop. Vol. 11, 2013.

[4] 橋本 玄基, 清野 真理子, 大枝 真一, “プログラマのスキル評価のためのログデータ解析”, 情報処理学会第 78 回全国大会 (2016).

[5] E. Dauber, A. Caliskan, R. Harang, R. Greenstadt, “Git Blame Who?: Stylistic Authorship Attribution of Small, In- complete Source Code Fragments”, Cornell University Library, arXiv:1701.05681, Jan 2017.

[6] paiza, `https://paiza.jp`

[7] testdome, `https://testdome.com`

[8] Deepik Badampudi, Ricardo Britto, Michael Unterkalmsteiner, “Modern code reviews - Preliminary results of a systematic mapping study”, EASE ’19 Proceedings of the Evaluation and Assessment on Software Engineering, pp.340-345, April 2019.

[9] J. R. Quinlan, “Induction of Decision Trees”, Machine Learning, Vol. 1, Issue 1, pp.81-106 (1986).

[10] Tin Kam Ho, “Random decision forests”, ICDAR ’95 Proceedings of the Third International Conference on Document Analysis and Recognition, Vol. 1 (1995).