特別研究報告

題目

判定対象の拡大を目的とした3値分類による ソースコードの良さの判定手法

指導教員

井上 克郎 教授

報告者

松井 智寛

令和 2 年 2 月 10 日

大阪大学 基礎工学部 情報科学科

平成 31 年度 特別研究報告

判定対象の拡大を目的とした 3 値分類によるソースコードの良さの判定手法

松井 智寛

内容梗概

プログラミングコンテストでは参加者が問題に対する回答ソースコードを作成して優劣を競っており，現在プログラミングの初級者から上級者まで世界中から多くの人が利用している．参加者はコンテストの成績によって順位がつけられ，レーティングが変動する．

ソースコードの良さを知ることができると，プログラミングの学習を支援することができる．例えば，あるソースコードが良くないと判断されると，そのソースコードには改善の余地があることがわかる．他にも，良くないソースコードを修正した後に良いソースコードと判定されるようになった場合は，行った修正が正しいものであったことがわかる．しかし，ソースコードの良さには様々な基準があり，また，人によって考え方も異なるため判定することは難しい．

そこで槇原はソースコードの特徴量を用いて，機械学習により良否を判定する手法を提案した．その手法はまず，プログラミングコンテストで提出された大量のソースコードに対し，字句解析やメトリクスの収集を行い，ソースコード特徴量を取得した．その後，対象のソースコードのうち，上級者，初級者が書いたものをそれぞれ「良」,「否」の 2 値に分類し，ソースコード特徴量と良否を機械学習により学習モデルを作成した．しかし，そのモデルはあまり精度が高くなく，また，用いたソースコードが上級者と初級者のもののみであったため，中級者の判定ができないものであった．

そこで本研究では，まず良否の判定を改善した後に 3 値分類により判定対象を拡大する手法を提案する．良否の判定はメトリクスの追加，ランダムフォレストの採用，上級者・初級者の定義の変更により結果を改善することができた．そして，3 値分類は全体のソースコードを上級者，中級者，初級者が書いたものの 3 値に分類することによって行った．モデルは良否の判定の改善したモデルを 3 値分類に拡張したものを用いた．その結果，F 値は上級者で 0.758，中級者で 0.793，初級者で 0.699 となった．

主な用語

プログラミングコンテスト
レーティング
機械学習

目 次

1 まえがき **4**

2 背景 **6**

2.1 オンラインジャッジシステム 6
2.2 プログラミングコンテスト 6
2.2.1 ルールと流れ 7
2.2.2 開催規模や参加者 7
2.2.3 レーティングシステム 7
2.3 機械学習と先行研究 8

3 提案手法 **11**

3.1 目的 11
3.2 上級者・中級者・初級者の定義 11
3.3 ソースコードの良さの判定の手順 11
3.4 既存の手法との差分 17
3.4.1 上級者・初級者の定義の変更 17
3.4.2 メトリクスの追加 17
3.4.3 学習アルゴリズムの変更 18
3.4.4 中級者の追加 18

4 手法の評価 **19**

4.1 データセット 19
4.1.1 ソースコードデータ 23
4.1.2 提出履歴データ 23
4.1.3 統計情報 23
4.2 精度の改善の評価実験 24
4.2.1 実験内容 24
4.2.2 結果・比較 25
4.3 判定対象の拡大の評価実験 26
4.3.1 実験内容 26
4.3.2 結果・比較 27

5　考察 **28**
5.1　既存手法からの 4 つの差分それぞれによる効果 28
5.2　問題に対するソースコードの正誤に基づいた学習対象の選別 30

6　まとめ **34**

謝辞 **35**

参考文献 **36**

1　まえがき

ソースコードの編集は，ソフトウェアを開発する上での必須作業であり，ソフトウェア工学ではソースコードの編集作業に関する研究は多く行われている [5,6,9]．ソフトウェアの開発では，ソースコードの実装，テスト，修正を繰り返し行う．そのため，ソースコードの編集に対する労力は大きくなり，特にプログラミングに慣れていない初級者は，慣れている上級者と比べると編集にかかる時間的コストが大きくなってしまう．また，初級者は行ったソースコード編集が適切であるか判断することも難しい．そのため，初級者でも行った編集が適切であるかどうか，判断できるようになることが望ましい．

近年，プログラミングは注目を浴びている．このことは日本で 2020 年から小学校でプログラミングが必修化されることや，プログラミングコンテストの参加者の増加 [8] からもうかがえる．こういったこともあり，今後プログラミングを行う人の数はますます増加することが予想される．そこでソースコードの編集作業にかける労力を減らすことは今後の課題になると考えられる．

ソースコードが与えられたとき，そのソースコード自体の良さを判定できるとソースコードの編集作業の助けになると考える．例えば，あるソースコードが良くないと判定されると，そのソースコードは改善の余地があることがわかる．また，その良くないソースコードを修正した後に良いソースコードと判定されるようになった場合は行った修正が正しいものであったことがわかる．また，良さの判定基準があると，良いソースコードと良くないソースコードの違いが明確になるため，良くないソースコードに対してどう修正したらよいかといった指針を与えることができるようになる．すると，良くないソースコードを書くことが多いと考えられる初級者のソースコードの編集作業を支援することも可能になる．しかし，ソースコードの良さを判定するには可読性や実行時間など，さまざまな基準が存在し，人によってどの基準を重要視するかといった考え方も異なる．そのため，ソースコードの良さの判定を行うことは難しい．

プログラミングコンテストにおける上級者と初級者の違いの分析は堤によって行われている [1]．その研究では上級者と初級者が書くソースコードは，特定の予約語の利用頻度や計測したメトリクスの値に差があることが確認された．そこで槇原は堤の研究結果を利用し，予約語の利用頻度やメトリクスの値を使って機械学習により定量的かつ自動的にソースコードの良否の判定を行い，さらに悪いソースコードに対して修正の指針を提示した [3]．しかし，その良否の判定の精度はあまり高くなかった．また，槇原は”上級者”が書いた”良”のソースコードと”初級者”が書いた”否”のソースコードのみを判定対象としており，どちらでもない”中級者”が書いたソースコードを扱っていなかった．そのため，仮に”中級者”が書いた”良”でも”否”でもないソースコードを学習モデルに入力した場合，”良”か”否”のどちらかで

出力されるため，中級者が書いたソースコードに対して正しく判断することができない．

そこで，本研究ではまず槇原の機械学習によるソースコードの良否判定をもとにして，学習アルゴリズムの変更や機械学習に利用するメトリクスの追加，判定対象として中級者の追加など，さまざまな変更点を加えた提案手法を試し，既存の手法と比較を行うことにより判定精度の向上と判定対象の拡大を確認する．そして，それぞれの変更点が与える影響について考察する．そこではそれぞれの変更点が判定精度の向上，判定対象の拡大のいずれかに良い影響を与えていることがわかった．これにより，既存の手法の精度が改善されただけでなく，既存の手法では判定することができなかったソースコードに対して良さを判定できるようになった．

以下，2 章では，本研究の背景として，プログラミングコンテストと機械学習について説明する．3 章では，提案手法と既存の手法からの差分について説明する．4 章では，評価実験の内容と結果，そして評価実験に用いたデータセットについて説明する．5 章では提案手法に用いられた 4 つの差分と，用いられなかった差分について考察を行う．最後に 6 章では，まとめと今後の課題について述べる．

2　背景

この章では本研究の背景として，プログラムコンテストとその採点に利用されるオンラインジャッジシステム，そして機械学習について説明する．プログラミングコンテストについてはさまざまなものが存在するが，本研究では問題の要求を満たすプログラムを時間内に作成することを競うコンテストを扱う．

以下ではまず，オンラインジャッジシステムについて説明する．

2.1　オンラインジャッジシステム

プログラミングコンテストにおいて，その採点に利用されるオンラインジャッジシステムについて説明する．オンラインジャッジシステムにはさまざまな問題が収録されており，その利用者は問題を選択し，回答を提出する．そして，システムは利用者に提出された回答の採点結果を通知する．採点の基準の一例を以下に示す．

- コンパイルできるかどうか
- 不正なメモリアクセスがないかどうか
- 制限時間内にプログラムが終了するか

このような基準の内，満たさない基準があればそのことを通知し，すべての基準を満たせば正解であることを通知する．

こういったオンラインジャッジシステムの一例として，国内では AIZU ONLINE JUDGE[1]，国外では Topcoder といったものが存在する．

2.2　プログラミングコンテスト

プログラミングコンテストはプログラミングの能力や技術を競い合うコンテストのことである．オンラインジャッジシステムが採用されており，同じ問題に対して，同じ時間内に複数の参加者がソースコードを記述し，提出する．コンテストが終了すると，正解問題数や回答時間に応じて参加者の順位が決定し，レーティング [4,7] が変動する．プログラミングコンテストには Codeforces[2]のように個人でプログラムを作成し，回答するもの以外にも，ACM ICPC[3]のように複数の参加者がチームを組んで回答するものも存在するが，本研究では個人で参加するプログラミングコンテストである Codeforces を対象とする．

[1] http://judge.u-aizu.ac.jp/onlinejudge/
[2] http://codeforces.com/
[3] https://icpc.baylor.edu/

2.2.1 ルールと流れ

プログラミングコンテストの Codeforces の大まかな流れについて説明する．Codeforces における一般的なコンテストでは，コンテストの開始時間が指定されており，開始時間になるとコンテストの問題が公開され，参加者は問題を解き始める．問題を解く順序や，用いるプログラミング言語は自由であり，得点は解いた問題の難易度や回答にかかった時間，そしてこれまでの提出回数によって変わる．また，参加者は制限時間内であれば，同じ問題に対して何度も回答を提出することが可能であり，回答ソースコードの正誤やコンパイル可能性については提出の可否に影響しない．回答ソースコードは提出された時点でオンラインジャッジシステムによって事前テストの実行が行われ，通過したか否かが参加者に通知される．事前テストは全テストケースを網羅しておらず，事前テストを通過しても回答が正しくない場合もある．コンテスト時間終了後に提出ソースコードに対して最終テストが実施され，参加者の最終的な得点が決定され，それに伴い順位も決定され，レーティングが変動する．

2.2.2 開催規模や参加者

本研究で利用する Codeforces においては，開催規模や参加者は以下の通りになっている．

- 開催頻度：偏りはあるが，大体週に 1 回以上
- 参加人数：毎コンテスト 7000〜8000 人程度
- 国籍：全世界から参加 (言語はロシア・英語)

また，過去 6 か月以内に一度でもコンテストに参加したことのあるユーザーを Codeforces ではアクティブユーザーと定義しているが，Codeforces におけるアクティブユーザー数は 2020 年 1 月 28 日現在，71578 人となっている．

2.2.3 レーティングシステム

Codeforces は参加者の熟練度をレーティング [7] を用いて表している．コンテストが行われるたびに参加者のレーティングが順位によって変動する．レーティングの計算方式はチェスなどの対戦型の競技で用いられるイロレーティング [4] と呼ばれる方式に似たものとなっている．本研究では，このレーティングが高い参加者を熟練度が高い参加者として扱う．

イロレーティングでは A と B のレーティングをそれぞれ r_A，r_B とすると，A が B に勝利する確率が

$P(r_A, r_B) = \frac{1}{1+10^{\frac{r_B - r_A}{400}}}$

となるようにレーティングが調整される．Codeforces では A が B より高い順位にいる確率となる．

Codeforces でのレーティング変更計算のアルゴリズムを Algorithm 1 に示す．(1) の for 文でおおよそのレーティング変動を計算している．関数 $getSeed(r)$ はレーティング r の順位の期待値を返す関数であり，$seed_i \leftarrow getSeed(r_i)$ とすることで，$seed_i$ はレーティング r_i の参加者の順位の期待値となる．そして $seed_i$ と実際の順位 $rank_i$ の幾何平均を m_i とし，$getSeed(R_i) = m_i$ となるようなレーティング R_i の探索を行う．その後，探索によって得られた R_i と r_i の平均を取った後，全体のレーティングの変動が 0 になるように処理を行ったものを参加者 i の暫定的なレーティングとする．しかしこのままでは上位のレーティングがインフレする恐れがあるため，(2) 以降でレーティングの変動の調整を行っている．こうして最終的なレーティングが決まる．

2.3 機械学習と先行研究

ソースコードの良否の判定を機械学習を用いて行う研究 [3] は槇原によってすでに行われている．そもそも機械学習とはコンピュータがデータを学習し，予測や分類を行う仕組みのことである．そして機械学習は教師あり学習，教師なし学習，そして強化学習の大きく 3 つにわけることができる．教師あり学習は，すでにある問題の特徴を表すデータとその答えを与えて学習を進める．そして新しいデータからその答えを予測することに使われる．教師なし学習は特徴を表すデータのみを入力して学習を進める．入力データの構造を分析することなどに使われる．強化学習はある環境の中で得られる報酬が最大となるようににして学習を進める．具体例に囲碁がある．囲碁では勝つことを報酬として，繰り返し対局することで現在の盤面からの最善手を学習する．槇原はソースコードの特徴から良否という答えを求めるために教師あり学習を用いた．

教師あり学習において，特徴を表すデータを説明変数，その答えを目的変数という．学習には説明変数と目的変数の両方を与える．その後学習を終えたモデルに新しい説明変数を与え，目的変数の予測をさせる．ここで，学習に使ったデータを訓練データ，予測に使ったデータをテストデータという．学習モデルの精度はテストデータから予測した目的変数が実際の目的変数とどれくらい等しいかで判断する．また，目的変数が”性別”といった離散値を予測する問題を分類問題，”身長”といった連続値を予測する問題を回帰問題という．分類問題で使われるアルゴリズムには，サポートベクターマシン (SVM)，決定木，ロジスティック回帰，ナイーブベイズ，ランダムフォレストなどがあり，回帰問題で使われるアルゴリズムには，線形回帰，正則化，SVM，回帰木，ランダムフォレストなどがある．

槇原の研究ではソースコードの良否判定を，”良”と”否”の 2 つの離散値を予測する分類問題として扱った．そして槇原は SVM と決定木の 2 つのアルゴリズムを採用した．

そして槇原により作成されたモデルの評価結果は表 1 とまとめられていた．そこで，私が [3] の 3.2 節のデータセット，3.3 節の上級者・初級者の定義をもとに，槇原が用いたデー

Algorithm 1 Codeforces におけるレーティング変更計算

Data: U：コンテストの全参加者

Data: r_i：参加者 i のコンテスト前のレーティング

Data: $rank_i$：参加者 i のコンテストでの順位

Result: r'_i：参加者 i の変化後のレーティング

function $getSeed(r)$

  return $\Sigma_{i \in U} P(r_i, r) + 1$;

function $getRatingToRank(r)$

  $left \leftarrow 1$;

  $right \leftarrow 8000$;

  while $right - left > 1$ **do**

    $mid \leftarrow \frac{left+right}{2}$;

    if $getSeed(mid) < r$ **then**

      $right \leftarrow mid$;

    else

      $left \leftarrow mid$;

  return $left$;

$sum \leftarrow 0$; // (1)

for $i \in U$ **do**

  $seed_i \leftarrow getSeed(r_i)$;

  $m_i \leftarrow \sqrt{seed_i * rank_i}$;

  $R_i \leftarrow getRatingToRank(m_i)$;

  $d_i \leftarrow \frac{R_i - r_i}{2}$;

  $sum \leftarrow sum + d_i$;

for $i \in U$ **do** // レーティング変動の合計を 0 にする

  $d_i \leftarrow d_i - \left(\frac{sum}{|U|} + 1\right)$;

// (2) 上位のレーティングのインフレを抑える処理

$topU \leftarrow U$ の上位 $min(|U|, 4\sqrt{|U|})$ 人;

$sum \leftarrow \Sigma_{i \in topU} d_i$

$inc \leftarrow min(max(-\frac{sum}{|topU|}, 10), 0)$;

for $i \in U$ **do** // 最終的なレーティングの決定

  $d_i \leftarrow d_i + inc$;

  $r'_i + d_i$;

タセットのソースコードを上級者・初級者に分類した．そして 5 章の評価実験と同様にソースコード特徴量を抽出しベクトル化を行い，学習データを 9 割，テストデータを 1 割として機械学習による良否の判定手法を再現し，評価を行った．その結果，求めた精度は表 2 のようになった．表 1 と表 2 を比べると表 2 の再現実験の方が精度が悪くなっていることがわかる．実験の精度の比較の際には再現実験の結果と比較を行う．

表 1: 槇原の手法の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.918	0.906	0.912	0.940	0.900	0.920
初級者	0.850	0.869	0.860	0.846	0.906	0.875

表 2: 再現実験の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.860	0.854	0.857	0.842	0.854	0.848
初級者	0.767	0.777	0.771	0.756	0.739	0.748

3　提案手法

3.1　目的

2.3 節で示したように，槇原による機械学習を用いたソースコードの良否の判定手法 [3] の再現実験を行うと槇原の手法は精度が悪くなってしまい，この結果はあまり良いと言うことができない．また，上級者に比べて初級者の精度が悪いことも確認できる．他にも，既存の手法では上級者と初級者によって提出されたソースコードしか学習しておらず，学習モデルに新しいソースコードを与えても良否でしか判定されない．プログラミングコンテストで提出されたソースコードには上級者でも初級者でもない人から提出されたソースコードも存在するが，既存の手法では学習に使われないため，判定できない．

以上より，既存の手法には以下の課題が考えられる．

- 分類の精度がよくない
- 判定可能な対象が狭い

そのため，私は既存の手法を改善し，そして 3 値分類により判定可能な対象を広げる手法を提案する．そこで，以降の節では上級者・中級者・初級者の定義を行った後，3 値分類によりソースコードの良さを判定する手法について説明する．そして既存の手法との差分についても述べる．

3.2　上級者・中級者・初級者の定義

以下の手順でプログラミングコンテストに提出されたソースコードを分割し，上級者・中級者・初級者の定義を行った．

1. ソースコードを解答者のレーティングの降順にソートする
2. ソートしたソースコードをファイル数が等しくなるように 4 分割する
3. 4 分割されたもののうち，最もレーティングが高いグループを上級者，最もレーティングが低いグループを初級者のソースコードとする
4. 上級者，初級者のどちらでもないグループを中級者のソースコードとする

3.3　ソースコードの良さの判定の手順

ソースコード特徴量を利用した機械学習を用いた 3 値分類によるソースコードの良さの判定までの流れを図 1 に示す．判定を行うまでの流れは大きく 3 つの段階に分けることが

できる．まず，プログラミングコンテストにおいて提出されたソースコードを収集し，レーティングによって上級者，中級者，初級者のソースコードに分類する．次に，分類したソースコードに対して予約語の利用頻度やメトリクスの値などのソースコード特徴量を取得し，機械学習の説明変数とする．目的変数はソースコードの良さであり，上級者のソースコードを”良い”ソースコード，初級者のソースコードを”良くない”ソースコード，そして中級者のソースコードを”どちらでもない”ソースコードと定義する．そして説明変数と目的変数を 1 つのベクトルにし，機械学習を行う．最後に，学習に用いられていない新規のソースコードのソースコード特徴量をベクトル化して入力し，ソースコードの良さの判定を行う．

3 つの段階それぞれについて，以下で詳細を述べる．

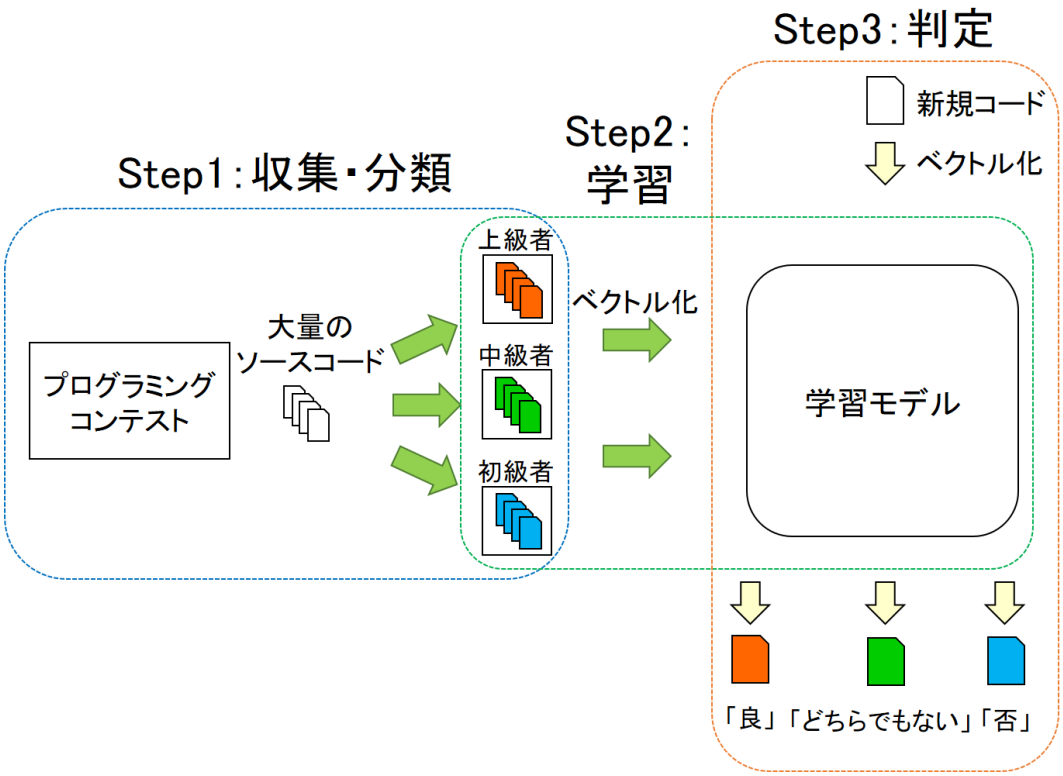

図 1: ソースコードの良さの判定の流れ

STEP1：収集・分類

ソースコードはプログラミングコンテストから収集する．収集したソースコードを 3.2 節で述べた定義に従って，上級者，中級者，初級者の 3 つに分類する．分類の対象はソースコードのうち，”GNU C++”，”GNU C++11”，”GNU C++14”で記述されたソースコードとする．

STEP2：学習

機械学習の説明変数はソースコード特徴量，目的変数はソースコードの良さである．まず準備として，対象となったすべてのソースコードに対して特徴ベクトルを作成する．学習させる値は，ソースコードごとのソースコード特徴量とソースコードの良さである．ソースコード特徴量は予約語の利用頻度とメトリクスの値を使用する．対象の 29 種類の予約語を表 3 に示す．これらの予約語は堤の調査 [1] により，上級者，または初級者による利用頻度が高いことがわかっている．そして対象の 22 種類のメトリクスを表 4 と表 5 に示す．これらは Web 上に公開されているソースコードメトリクス計測ツールである SourceMonitor[4]を用いて計測できるものであり，表 4 に含まれる 11 種類のメトリクスは堤の調査により上級者，または初級者のソースコードでは値が大きくなることがわかっている．表 5 に含まれる 11 種類のメトリクスは堤による調査の対象とはならなかったが，説明変数に利用したメトリクスとなっている．statements_at_block_level_3〜7 については表示を省略している．これらを合わせ，ベクトルにするとソースコード特徴量として 51 次元のベクトルを作成することができる．図 2 にソースコード特徴量のベクトルの一部を掲載する．そしてソースコードの良さは上級者が書いたソースコードを”良い”ソースコード，初級者が書いたソースコードを”良くない”ソースコード，中級者が書いたソースコードを”どちらでもない”ソースコードと定義し，それぞれ値として 1，-1，0 を与える．これを先ほど作成したソースコード特徴量のベクトルに追加し，52 次元のベクトルを作成する．ソースコードの良さは”skill”という変数名にしている．図 3 にソースコードの良さを追加したベクトルの一部を掲載する．

学習の対象となる全てのソースコードに対して図 2 のようなベクトル化を行い，ソースコード特徴量を説明変数，良否を目的変数として機械学習を行う．使用したアルゴリズムはランダムフォレストであり，python の機械学習ライブラリである scikit-learn を用いてプログラムを記述した．

STEP3：判定

良否の判定を行いたいソースコードに対し，STEP2 と同様の手順で図 2 のようなソースコード特徴量による 51 次元のベクトルを作成する．学習を行ったモデルに対し，ベクトルを入力すると”良い”ソースコード，”良くない”ソースコード，”どちらでもない”ソースコードのいずれかが出力される．

[4] http://www.campwoodsw.com/sourcemonitor.html

表 3: 利用頻度が高い予約語

asm	break	case	catch
class	continue	decltype	do
else	enum	extern	for
friend	goto	if	namespace
operator	private	public	return
struct	switch	template	try
typedef	typeid	typename	using
while			

表 4: 堤の調査の対象となったメトリクス

メトリクス	説明
avg_complexity	各関数の循環的複雑度の平均値
max_complexity	各関数の循環的複雑度の最大値
avg_depth	各関数のネスト深さの平均値
max_depth	各関数のネスト深さの最大値
methods_per_class	クラス当たりのメソッド数
n_classes	クラス数
n_func	関数の数
n_lines	物理行数
n_statements	セミコロンで区切られた論理行数
percent_branch_statements	全体の論理行数に占める分岐文 (if，else，for，while，goto，break，continue，switch，case，default，return を用いた文) の割合
percent_comments	全体の物理行数に占めるコメントの割合

asm	break	case	catch	class	・・・
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	2	0	0	0	
0	4	0	0	0	
0	0	0	0	0	

statements_at_ block_level5	statements_at_ block_level6	statements_at_ block_level7	statements_at_ block_level8	statements_at_ block_level9
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
2	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

図 2: 説明変数のベクトル

asm	break	case	catch	class	・・・
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	0	0	0	0	
0	2	0	0	0	
0	4	0	0	0	
0	0	0	0	0	

statements_at_ block_level5	statements_at_ block_level6	statements_at_ block_level7	statements_at_ block_level8	statements_at_ block_level9	skill
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1
0	0	0	0	0	1
2	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	-1
0	0	0	0	0	-1
0	0	0	0	0	-1
0	0	0	0	0	-1

図 3: 目的変数”skill”を追加したベクトル

表 5: ソースコード特徴量として用いるメトリクス

メトリクス	説明
avg_statements_per_method	各メソッドの論理行数の平均値
statements_at_block_level_0	深さ 0 の論理行数
statements_at_block_level_1	深さ 1 の論理行数
statements_at_block_level_2	深さ 2 の論理行数
⋮	⋮
statements_at_block_level_8	深さ 8 の論理行数
statements_at_block_level_9	深さ 9 以上の論理行数

3.4 既存の手法との差分

この節では 3.3 節で述べた本研究で提案するソースコードの良さの判定手法と槇原によるソースコードの良否の判定手法との差分を説明する．

3.4.1 上級者・初級者の定義の変更

提案手法と既存手法では上級者・初級者の定義に違いがある．提案手法では 3.2 節で述べたように定義を行ったが，既存手法では以下のように定義が行われている．

1. 参加者をレーティングの降順にソートする
2. ソートした参加者を人数が等しくなるように 4 分割する
3. 4 分割されたもののうち，最もレーティングが高いグループを上級者，最もレーティングが低いグループを初級者とする．

このため，提案手法では上級者・初級者に分類されるファイル数は近い値となるが，人数が異なることがある．一方で既存手法では上級者・初級者に分類される人数は近い値となるが，ファイル数が異なることがある．

3.4.2 メトリクスの追加

提案手法と既存手法では機械学習に用いるソースコード特徴量のベクトルの内容にも違いがある．提案手法で作成したソースコード特徴量のベクトルは 3.3 節の STEP2 の学習で説明したように，表 3 に示した予約語 29 種類の利用頻度と表 4 と表 5 に示した 22 種類のメトリクスの値からなっていた．そのうち，表 5 のメトリクスは既存手法ではソースコード特徴量のベクトルに採用されていなかった．そのため，既存手法ではソースコード特徴量のベク

トルによる説明変数は 40 次元であったのに対し，メトリクスの追加により，提案手法では説明変数が 51 次元となっている．

3.4.3　学習アルゴリズムの変更

採用した学習アルゴリズムにも違いがある．提案手法ではランダムフォレストを採用したが，既存手法では決定木と SVM が採用されていた．

3.4.4　中級者の追加

既存手法はプログラミングコンテスト参加者のうち，上級者，初級者が書くソースコードをそれぞれ”良”，”否”として 2 値で分類を行っていた．その際，上級者，初級者のどちらでもない参加者は考慮されず，判定の対象となっていなかった．提案手法では 3.2 で説明したように上級者，初級者のどちらでもない参加者を中級者と定義することで判定対象を拡大した．

4　手法の評価

この章では提案手法の評価のために，2 つの実験を行う．そこで，まず実験に利用したデータセットについて説明する．そして 1 つ目の実験では既存の手法からの差分のうち，中級者の追加を行わずに残り 3 つの差分で良否の判定を行ったものと既存の手法を比較することで精度が向上されるかどうかを確認する．そして 2 つ目の実験では全ての差分を加えた提案手法による精度を確認し，既存の手法に中級者の追加を行ったものと比較することで，判定対象を拡大しても精度が向上されるかどうかを確認する．

4.1　データセット

ここでは評価実験に用いたデータセットについて述べる．このデータセットは槇原 [3] も用いており，堤 [1] によって作成されたものである．また，このデータセットはオンライン上で公開されている[5]．

データセットは，参加者が問題への回答として提出したソースコードと，言語やタイムスタンプ等の提出履歴情報データベースの 2 種類からなる．提出履歴情報データベースの内容は表 6 で，データセットの統計情報は表 7 で示している．また，本データは 2016/5/19〜2016/11/15 の期間に収集されており，ソースコードのファイル数は 1,644,636 である．プログラミングコンテストにおける提出は 1 つのソースファイルにまとめられるため，これは提出数と一致している．参加者数は，2016/5/19〜2016/11/15 の期間に 1 度以上 Codeforces のコンテストに参加したユーザーの総数である．また，各コンテストには複数の問題が含まれるため，コンテストの数と比較して問題数が多くなっている．

[5] https://sites.google.com/site/miningprogcodeforces/

表 6: データセットの内訳

テーブル名	カラム名	キー	説明	データ型
Participant	user_name	PK	Codeforces の参加者名	String
	rating		参加者の現在のレーティング	Integer
	max_rating		2016/11/15 までの最大到達レーティング	Integer
Participant-Submission	user_name	PK FK	*Participant* テーブルにおける *user_name*	String
	files		データセット内における *user_name* の提出数	Integer
File	file_name	PK	データセット上でのソースファイル名	String
	submission_id		Codeforces における提出 ID	Integer
	lang		ソースファイルのプログラミング言語	String
	user_name	FK	ソースファイルの提出者	String
	verdict		提出の正誤	String
	timestamp		提出時刻	Date/Time
	competition_id	FK	*Competition* テーブルにおける *competition_id*	Integer
	problem_id		この提出に対応する *problem_id*	String
	url		Codeforces におけるこのソースファイルの URL	String
	during_competition		この提出がコンテスト時間内に提出されたかどうか	Boolean
Competition	competition_id	PK	コンテスト ID	Integer
	name		コンテスト名	String
	start_time		コンテスト開始時刻	Date/Time
	duration_time		コンテスト時間	Integer
	participants		コンテスト参加者数	Integer

次ページへ続く

前ページからの続き

テーブル名	カラム名	キー	説明	データ型
Problem	problem_id	PK	問題 ID	String
	competition_id	FK	この問題が掲載されたコンテストの *competition_id*	Integer
	prob_index		Index of the problem in the competition	String
	points		コンテストにおけるこの問題の得点	Integer
Acceptance	problem_id	PK FK	対応する問題の *problem_id*	String
	submission_in_sample		データセット上におけるこの問題に対する提出数	Integer
	solved_in_sample		データセット上におけるこの問題に対する正答数	Integer
	submission		この問題に対する全提出数	Integer
	solved		この問題に対する全正答数	Integer
	acceptance_rate		*solved/submissions*	Double
	lastmodied		データの最終更新日	Date/Time
Problem-Submission-Statistics	problem_id	PK	対応する *problem_id*	String
	filesize_max		この問題に対する提出ファイルサイズの最大値	Integer
	filesize_min		この問題に対する提出ファイルサイズの最小値	Integer
	filesize_mean		この問題に対する提出ファイルサイズの平均値	Double
	filesize_median		この問題に対する提出ファイルサイズの中央値	Integer
	filesize_variance		この問題に対する提出ファイルサイズの分散	Double
	filesize_max_competition		*filesize_max* のうちコンテスト中に提出されたもの	Integer

次ページへ続く

前ページからの続き

テーブル名	カラム名	キー	説明	データ型
	filesize_min_competition		*filesize_min* のうちコンテスト中に提出されたもの	Integer
	filesize_mean_competition		*filesize_mean* のうちコンテスト中に提出されたもの	Double
	filesize_median_competition		*filesize_median* のうちコンテスト中に提出されたもの	Integer
	filesize_variance_competition		*filesize_variance* のうちコンテスト中に提出されたもの	Double
Submission-Distance	file_name	PK	対応するソースファイル名	String
	proble_id		この提出に対応する *problem_id*	String
	next_file		*file_name* の次の提出	String
	submission_index		同じ問題に対してこの提出が何番目の提出か	Integer
	levenshtein_distance		*file_name* と *next_file* とのトークンベースの編集距離	Integer
	add_node		*file_name* から *next_file* にかけての追加ノード数	Integer
	delete_node		*file_name* から *next_file* にかけての削除ノード数	Integer
	update_node		*file_name* から *next_file* にかけての更新ノード数	Integer
	move_node		*file_name* から *next_file* にかけての移動ノード数	Integer
	node_sum		追加， 削除， 更新， 移動ノード数の合計	Integer

以上

表 7: データセットの統計情報

収集期間	ファイル数	参加者数	コンテスト数	問題数	DB サイズ
2016/5/19〜2016/11/15	1,644,636	14,520	739	3,218	357MB

4.1.1 ソースコードデータ

ソースコードは，プログラミングコンテストの参加者が問題に対する回答として提出したものであり，コンパイル環境が用意されている任意の言語を提出することができる．また，回答ソースコードが正答であるか，コンパイル可能かどうかは提出の可否に影響しないため，文法的に不完全なソースコードが含まれる場合もある．

4.1.2 提出履歴データ

提出履歴情報には，参加者のレーティング情報や，どの参加者がどの問題にいつ提出したか，提出が正解であったか等の情報が含まれている．データベースの構成は表 6 に示す通りであり，以下では本研究で利用したテーブルである，Participant と File の詳細について述べる．

- **Participant** このテーブルには，2016/5/19〜2016/11/15 の期間中 1 度以上 Codeforces で開催されたプログラミングコンテストに参加したユーザーの情報を含む．各参加者情報には表 6 に示す通り 3 種類の項目が含まれる．Codeforces は一意のユーザー ID を提供しておらず，本データセットにおいてはユーザー名を ID としてある．また，レーティングはコンテストごと変化するが，本データに含まれるレーティングは 2016/11/15 時点のものである．
- **File** 2016/5/19〜2016/11/15 の期間中に Codeforces に対して提出されたソースコードの情報を収集したテーブルである．このテーブルには，ソースコードデータの総数と同じ 1,644,636 のデータを含む．各提出履歴に与えられた一意の提出 ID と提出対象である問題の ID をもとに対応するソースコードの URL を構築することができ，url カラムに格納されている．

4.1.3 統計情報

本データセットにおける，ソースコードの言語別提出数の割合を図 4 に示す．提出されたソースコードのうち，90%は C++によって記述されている．

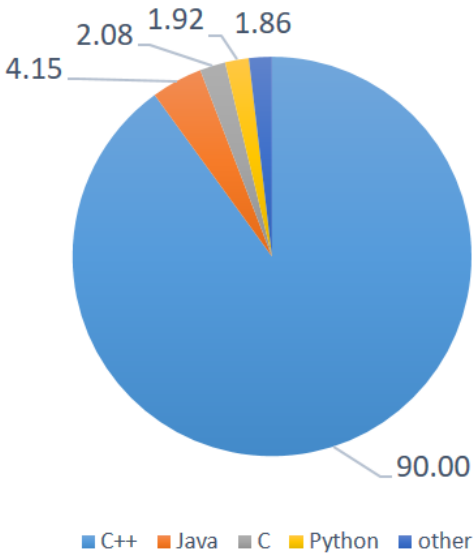

図 4: 提出ソースコードの言語分布

4.2 精度の改善の評価実験

この節では既存手法に以下の 3 つの差分を加えたものにより，4.1 節で説明したデータセットに対してソースコードの良否の判定を行い，精度を求め，既存の手法の結果と比較する．

- 上級者・初級者の定義の変更
- メトリクスの追加
- 学習アルゴリズムの変更

4.2.1 実験内容

データセットのソースコードを上級者・初級者に分類したとき，それぞれのレーティングの分布は表 8 のようになった．初級者のレーティング範囲は-39〜1310，上級者は 1711〜3367 となった．既存の手法で分類したときは表 9 のような分布であった．そしてこれらのソースコードから 3.3 節の Step2:学習で述べたように，合計 51 種類のソースコード特徴量を抽出し，ベクトル化を行う．その後，ベクトルに目的変数を加える．この実験では中級者を含まないため上級者の”良い”ソースコードに 1 が，初級者の”良くない”ソースコードに-1 が与えられ，2 値で分類される．そして学習アルゴリズムにはランダムフォレストを用い，学習データを全体の 9 割，テストデータを残りの 1 割として 10 分割交差検証により，学習・評価を行う．評価値には適合率，再現率，F 値を採用した．適合率，再現率，F 値について以下で説明する．

- 適合率：予測結果に含まれる正解の割合
- 再現率：実際の正解のうち，予測された者の割合
- F 値：適合率と再現率の総合的な評価を表す指標であり，適合率と再現率の調和平均

表 8: 上級者・初級者のレーティングの統計情報

		初級者	上級者
レーティング	平均	1180.22	1944.35
	分散	13170.37	46307.75
	最小値	-39	1711
	中央値	1211	1902
	最大値	1310	3367
人数		3899	2212
ファイル数		353,346	352,557

表 9: 既存手法における上級者・初級者のレーティングの統計情報

		初級者	上級者
レーティング	平均	1171.12	1824.824
	分散	12909.16	51321.37
	最小値	-39	1573
	中央値	1202	1764
	最大値	1299	3367
人数		3634	3622
ファイル数		332,863	544,290

4.2.2 結果・比較

4.2.1 項で説明した実験の結果を表 10 に示す．既存の手法の結果を示している表 2 と比較を行う．すると上級者は再現率ではほとんど変わっていないが，適合率が上がったことにより，F 値も向上している．一方，初級者は適合率，再現率，F 値のすべてが大きく上がっている．また，表 10 の上級者・初級者の F 値を見ると，ほとんど差がない．表 2 で見られた初級者の方が精度が悪くなっているといったものも解消されている．

表 10: 3 つの差分を加えた手法の結果

	ランダムフォレスト		
	適合率	再現率	F 値
上級者	0.912	0.852	0.881
初級者	0.861	0.918	0.889

4.3 判定対象の拡大の評価実験

この節では 3 章で説明した提案手法により，4.1 節で説明したデータセットに対してソースコードの良さの判定を行い，精度を求める．比較のために，既存手法に対して中級者の追加を行った 3 値分類によるソースコードの良さの判定も行い，精度を求める．

4.3.1 実験内容

提案手法によりデータセットのソースコードを上級者・中級者・初級者に分類したとき，それぞれのレーティングの分布は表 11 のようになった．初級者のレーティング範囲は-39〜1310，中級者は 1311〜1710，上級者は 1711〜3367 となった．そしてこれらのソースコードを用いて 4.2.1 項で述べたようにベクトルの作成，学習，評価を行う．この実験では中級者を含むため，中級者のソースコードには目的変数として 0 があたえられ，3 値で分類される．

表 11: 上級者・中級者・初級者のレーティングの統計情報

		初級者	中級者	上級者
レーティング	平均	1180.22	1459.66	1944.35
	分散	13170.37	10153.27	46307.75
	最小値	-39	1311	1711
	中央値	1211	1434	1902
	最大値	1310	1710	3367
人数		3899	8409	2212
ファイル数		353,346	701,871	352,557

そして比較のために，既存手法に対して中級者の追加を行った手法でも精度を求める．これによりデータセットのソースコードを上級者・中級者・初級者に分類したとき，それぞれのレーティングの分布は表 12 のようになった．また，既存の手法ではメトリクスを表 4 のもののみを採用していたため，ソースコード特徴量は 40 種類となる．そして学習アルゴリズムは決定木と SVM を採用する．

表 12: 既存の手法を拡張した際のレーティングの統計情報

		初級者	中級者	上級者
レーティング	平均	1171.12	1419.55	1824.824
	分散	12909.16	4577.94	51321.37
	最小値	-39	1300	1573
	中央値	1202	1408	1764
	最大値	1299	1572	3367
人数		3634	7264	3622
ファイル数		332,863	530,621	544,290

4.3.2 結果・比較

4.3.1 項で説明したもののうち，提案手法による結果を表 13 に，既存手法に中級者を追加した 3 値分類による結果を表 14 に示す． 表 13 と表 14 を比較すると，提案手法では上級者

表 13: 提案手法による 3 値分類の結果

	ランダムフォレスト		
	適合率	再現率	F 値
上級者	0.847	0.686	0.758
中級者	0.755	0.836	0.793
初級者	0.708	0.691	0.699

表 14: 既存手法に中級者を追加した 3 値分類の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.757	0.741	0.749	0.716	0.706	0.711
中級者	0.676	0.676	0.676	0.558	0.641	0.597
初級者	0.613	0.635	0.624	0.580	0.456	0.511

で再現率が落ちているが，それ以外の指標はすべて上がっており，全体としては既存の手法を拡張したものより，提案手法の方が高い精度で分類できていることがわかる．また，F 値は上級者・中級者・初級者のすべてで上がっているが特に中級者・初級者で大きく上がっていることが確認できる．

5　考察

この章では，既存手法から提案手法に加えた 4 つの差分のそれぞれの効果について確認し，考察する．また，精度が上がらなかったために，提案手法には採用されなかった差分についても説明し，なぜ精度が上がらなかったのかを考察する．

5.1　既存手法からの 4 つの差分それぞれによる効果

まず，差分は以下の 4 つであった．

- 上級者・初級者の定義の変更
- メトリクスの追加
- 学習アルゴリズムの変更
- 中級者の追加

この節では，差分を 1 つずつ加えた手法による学習の評価を行うことで，それぞれの差分の効果を確認し，考察する．

最初に上級者・初級者の定義の変更のみを加えた手法で学習・評価を行う．この方法で分類を行ったとき，レーティングの分布は表 8 であった．また，既存の手法による分類を行ったときのレーティングの分布は表 9 であった．このとき，差分を加えたときの結果を表 15 に示す．

表 15: 上級者・初級者の定義を変更した手法の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.838	0.834	0.836	0.848	0.799	0.823
初級者	0.836	0.840	0.838	0.810	0.856	0.832

表 2 に比べると，上級者では SVM の適合率はほとんど変わっていないがそれ以外で評価値が落ちていることがわかる．一方，初級者ではすべての評価値が上がっていることがわかる．上級者の精度の下がり幅に比べ，初級者の精度の上がり幅の方が大きく，全体としては精度が上がったと考えられる．また，既存の手法に存在した，上級者と初級者の精度の差も F 値を見ると改善されていることがわかる．これにより，上級者・初級者の定義の変更が精度の改善，特に上級者・初級者の精度の差の改善に大きく寄与していることがわかる．また，表 8 と表 9 を比べると，上級者のファイル数と上級者のレーティングの最小値に大きな

違いがあることがわかる．そのため，既存の手法はファイル数が上級者の方が大きくなってしまったことにより，上級者に偏った学習になってしまって初級者の精度が落ちてしまったことが考えられる．また，上級者の分布が初級者に近くなってしまったことにより，全体として精度が落ちてしまったことも考えられる．

次にメトリクスの追加のみを行った手法で学習・評価を行う．既存の手法で目的変数が表 3 と表 4 を合わせた 40 種類であるのに対し，この手法では説明変数が表 5 を追加した 51 種類となる．このとき得られた結果を表 16 に示す．表 2 と比較すると，すべての指標で少しずつ精度が上がっていることがわかる．

表 16: 説明変数のメトリクスを追加した手法の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.872	0.863	0.868	0.868	0.877	0.873
初級者	0.781	0.794	0.787	0.796	0.783	0.790

次に学習アルゴリズムの変更のみを加えた手法で学習・評価を行う．既存の手法で学習アルゴリズムが決定木と SVM であるのに対し，この手法では学習アルゴリズムにランダムフォレストを採用している．このとき得られた結果を表 17 に示す．表 2 と比較すると，すべての指標で精度が上がっていることがわかる．この上がり幅はメトリクスを追加した表 16 の上がり幅より大きい．

表 17: ランダムフォレストを用いた手法の結果

	ランダムフォレスト		
	適合率	再現率	F 値
上級者	0.900	0.880	0.890
初級者	0.812	0.841	0.826

これまでに見た 3 つの差分をすべて加えたものが表 10 の結果であった．この結果は既存の手法より上級者・初級者の精度の差が改善され，全体的な精度も向上していた．3 つの差分をそれぞれ加えた結果から考えると，上級者・初級者の精度の差は上級者・初級者の定義の変更によって改善されたと考えられる．次に全体的な精度の向上は 3 つの差分すべてによって得られた結果だと考えられる．特に学習アルゴリズムの変更のみを加えた手法による精度の向上は他の差分を加えたものより大きかった．そのため，表 10 の精度の向上も他の差分に比べて学習アルゴリズムの変更が大きく寄与したと考えられる．

最後に中級者の追加のみを行った手法の結果を見る．これは 4.3.1 項で行ったものと同じ

であるため，結果は表 14 となる．表 2 と比較すると，上級者・初級者のどちらも精度が落ちていることがわかる．この精度の低下は中級者を追加したことによるものである．ここで表 12 のレーティングの分布をみると，初級者のレーティングの分布が-39〜1299，中級者が 1300〜1572，上級者が 1573〜3367 であった．全てのソースコードを判定対象にしたため，初級者の最大値と中級者の最小値，中級者の最大値と上級者の最小値の差は 1 となっている．このため，3 値分類では 1280 などのレーティングを持つ，比較的中級者に近い初級者が中級者と判断されるといったことによる間違いで精度が落ちてしまったと考えられる．既存の手法のテストデータにはテストデータの作成の手順のため，あらかじめ上級者・初級者のどちらかであるとわかっているもののみが入る．しかし，実際のソースコードの良さの判定では，判定対象のソースコードの良さがわかっていないことが多いと考えられる．そのため，中級者を追加することにより判定対象を拡大した手法は，上級者・初級者の精度は下がっているが，ソースコードの良さを知ることに適していると考えられる．

5.2 問題に対するソースコードの正誤に基づいた学習対象の選別

この節では精度の改善のために考案した手法であるが，改善されることがなかったため，提案手法で用いなかった差分について説明する．

既存手法では，上級者・初級者のすべてのソースコードを対象に学習を行っていた．プログラミングコンテストは解答ソースコードの正誤に関わらずに提出することができるため，正しくないソースコードも学習に使われていた．コンテストが行われている間に正しくないソースコードを提出した解答者が自分のミスに気付いた場合，その箇所に訂正を行い再び提出することが多い．こういった訂正前と訂正後のソースコードは似ていることが多いため，既存手法では同じようなソースコードを多く学習していると考えられる．これが精度の低下につながるのではないかと考えた．

そこで，正答であるソースコードのみを選別して学習に使うという手法を考案した．具体的な手順は以下のとおりである．

1. データセットから正答以外のソースコードを除外する
2. ソースコードを解答者のレーティングの降順にソートする
3. レーティングの上位 25%，下位 25%をそれぞれ上級者，初級者のソースコードとして学習を行い，評価する

これらの流れを図 5 に示す．

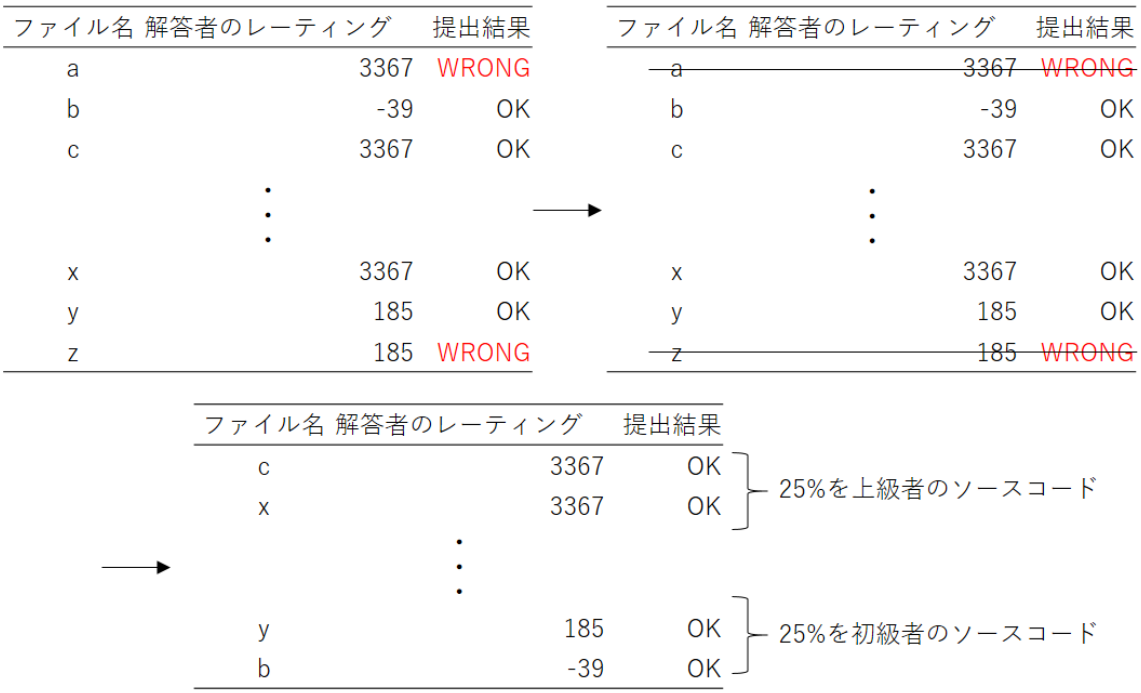

図 5: 正答のソースコードを選別する流れ

この結果，上級者・初級者のレーティングの統計情報は表 18 のようになった．そしてこれ

表 18: 正答のみを用いた場合の上級者・初級者のレーティングの統計情報

		初級者	上級者
レーティング	平均	1189.24	1961.43
	分散	13448.12	45731.89
	最小値	-39	1732
	中央値	1219	1915
	最大値	1322	3367
人数		4175	2055
ファイル数		140,058	140,173

らのソースコードで学習・評価をしたときの結果を表 19 に示す．この手法は上級者と初級者のソースコードの数が同じであるため，表 15 と比較を行う．この手法では決定木，SVM の両方で，上級者・初級者のすべての評価指標において値が下がってしまった．表 8 と表 18 でレーティングの統計情報を比較すると，ファイル数は大きく変わっているが，上級者・初級者のそれぞれの最大値・最小値・平均・分散などの値はあまり変わらない．そのため，レーティングの分布の違いによる精度の低下は考えられない．ここで，結果の原因として以下の

表 19: 正答のソースコードのみを用いた手法の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.732	0.728	0.730	0.786	0.770	0.778
初級者	0.727	0.730	0.729	0.776	0.792	0.784

ことが考えられた．

- 学習させるソースコードの数が少なく，学習不足になってしまった
- 正答以外のソースコードを学習させることに意味がある

これらの原因が正しいかどうかを調べるために加えて 2 つの実験を行った．

1. 上級者・初級者のソースコードを正答のソースコードのそれぞれレーティング上位・下位 33%にすることで学習に使うソースコードを増やす
2. 上級者･初級者のソースコードを正答以外も含むすべてのソースコードのそれぞれレーティング上位・下位 25%として，そこからそれぞれ約 140,000 個を抽出し，学習する

1 つ目の実験における上級者・初級者のレーティングの統計情報を表 20 に示す．そしてこの実験の結果は表 21 のようになった．また，2 つ目の実験の結果は表 22 のようになった．

表 20: 正答のみを用い，上級者・初級者を全体の 33%とした場合のレーティングの統計情報

		初級者	上級者
レーティング	平均	1238.18	1891.26
	分散	15446.20	48281.44
	最小値	-39	1650
	中央値	1270	1833
	最大値	1380	3367
人数		5893	2768
ファイル数		185,907	184,918

まず表 19 と表 21 を比較すると，上級者・初級者を正答ソースコードの 25%とした方が精度が良いことがわかる．次に表 19 と表 22 を比較すると，追加実験 2 の方が精度がよく，この結果は表 15 とほとんど同じ結果となった．以上から，表 19 の結果は学習不足によるものではなく，正答のソースコードのみを学習させたことによるものであると考えられる．また，

表 21: 追加実験 1 の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.698	0.692	0.695	0.756	0.742	0.749
初級者	0.696	0.703	0.700	0.748	0.763	0.755

表 22: 追加実験 2 の結果

	決定木			SVM		
	適合率	再現率	F 値	適合率	再現率	F 値
上級者	0.843	0.838	0.840	0.858	0.805	0.831
初級者	0.838	0.842	0.840	0.812	0.864	0.837

表 18 と表 20 を比較すると，上級者と初級者のレーティングの差は表 20 の方が小さい．追加実験 1 は，レーティングの差が小さくなったことにより結果が悪くなったと考えられる．

今回，正答であるソースコードのみを選別して学習に使う手法は，プログラミングコンテストの参加者が正しくないソースコードを訂正して提出することが多いことから考えられたものであった．しかし正しくないソースコードもレーティングを決める大きな要因になる．また，訂正する参加者も多い一方で，プログラミングコンテストには時間制限もあることから訂正せずに終わる参加者や，訂正して提出したとしても，そのソースコードが正しくなく，結果時間内に正しいソースコードを提出できない参加者もいる．そうした参加者のソースコードの良否をただしく判断するには，正答以外のソースコードも学習に使うことも必要であると考えられる．そのため，正答であるソースコードのみを選別した学習に使う手法で結果が悪くなってしまった原因は，正答以外のソースコードを学習させることに意味があるからだと考えられる．

6　まとめ

本研究では，機械学習によりソースコードの良否を判定する手法を改善し，また，3 値分類をすることでソースコードの良さを判定する手法を提案した．良否の判定では，メトリクスの追加，ランダムフォレストの採用，そして上級者・初級者の定義の変更により，精度の向上が確認できた．また，3 値分類では良否の判定を改善した手法を利用することで，既存の手法で 3 値分類を行った結果よりも良い結果が得られ，良否の判定では判定することができなかったソースコードも判定することができるようになった．

今後の課題として，以下の 2 つが挙げられる．

1 つ目は修正案の提示することである．槇原 [3] は良否の判定において，”否”と判定されたソースコードに対して修正案を提示していた．しかし，今回行った 3 値分類では修正案の提示を行っていない．そこで，3 値分類でも修正案を提示することが考えられる．槇原は上級者で利用される割合が高い語を修正案に提示していた．3 値分類の場合では初級者・中級者に対して上級者で利用される割合が高い語を提示することも考えられるが，初級者に対しては中級者で利用される割合が高い語を提示することも考えられる．

2 つ目は精度の向上のために，よりよい学習モデルを作成することである．3 値分類により判定の対象は拡大されたが，2 値分類よりも精度が悪くなってしまった．そのため，学習モデルを改良する必要がある．今回の手法は予約語の利用頻度とメトリクスの値を説明変数としていた．しかし，これはソースコードの構造の情報が大きく損なわれてしまう．ソースコードの持っている情報を損なうことなく学習することができれば，精度が向上すると考えられる．

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，研究活動において貴重な御指導及び御助言を賜りました．心より深く感謝いたします．

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 松下誠 准教授には，研究の方針から本論文の執筆に至るまで，直接の御指導及び御助言を賜りました．心より深く感謝いたします．

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 神田哲也 助教には，研究活動において適切な御助言を賜りました．心より深く感謝いたします．

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 槇原啓介 氏には，ツールを提供していただくなど，様々な御支援を賜りました．心より深く感謝いたします．

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 藤原裕士 氏には，研究活動において適切な御助言を賜りました．心より深く感謝いたします．

最後に，その他様々な御指導及び御助言を頂いた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の皆様に心より深く感謝いたします．

参考文献

[1] 堤祥吾．プログラミングコンテスト初級者・上級者間におけるソースコード特徴量の比較，大阪大学大学院情報科学研究科修士論文，2018．

[2] 槇原啓介．ソースコード特徴量を用いた機械学習によるソースコードの良否の判定，大阪大学基礎工学部情報科学科卒業論文，2019．

[3] 槇原啓介，松下誠，井上克郎．ソースコード特徴量を用いた機械学習によるソースコード品質の評価手法．電子情報通信学会技術研究報告，Vol. 119, No. 113, pp. 105-110, 2019．

[4] Arpad E Elo. The rating of chessplayers, past and present. Arco Pub., 1978.

[5] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In *Proc. of ICSE 2012*, pp. 3–13, 2012.

[6] Quinn Hanam, Fernando S. de M. Brito, and Ali Mesbah. Discovering bug patterns in javascript. *ACM SIGCSE Bulletin*, pp. 144–156, 2016.

[7] Mikhail Mirzayanov. Codeforces rating system. `http://codeforces.com/blog/entry/102`, 2010.

[8] Mikhail Mirzayanov. Codeforces: Results of 2018. `https://codeforces.com/blog/entry/65026`, 2019.

[9] Hao Zhong and Zhendong Su. An empirical study on real bug fixes. In *Proc. of ICSE 2015*, pp. 913–923, 2015.