Vol.2020-SE-205 No.7
2020/7/2

判定対象の拡大を目的とした3値分類による ソースコード品質の評価手法

松井 智寛[1,a)]　松下 誠[1,b)]　井上 克郎[1,c)]

概要：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はすでに提案されており，そこではプログラミングコンテストサイトから収集したレーティング上位と下位のソースコードを上級者，初級者として 2 値で分類していた．本研究では既存手法に対し，学習アルゴリズムの変更や，使用するソースコード特徴量の追加などにより精度を向上させ，また，既存手法では利用されなかった上位・下位のどちらでもないソースコードを中級者とすることで判定対象の拡大も行う．これらを適用した本手法の評価実験の結果，上級者，中級者，初級者の F 値がそれぞれ 0.758，0.793，0.699 となった．

キーワード：プログラミングコンテスト，レーティング，機械学習

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 [2,3,10]．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．

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 [7] からもうかがえる．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品質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，品質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．これによって，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と考えられる．しかし，ソースコードの品質を判定するには，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，ソースコードの品質の判定を機械的に行うことは難しい．

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が堤によって行われている [9]．堤の研究によって，上級者と初級者が書くソースコードを比較すると，特定の予約語の利用頻度や計測したメトリクスの値に有意差があることが確認された．そこで槇原ら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う方法を提案し，さらに初級者のソースコードに対する修正の指針を提示した [5]．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，槇原らは’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．

本研究ではまず，槇原らの機械学習によるソースコード

[1] 大阪大学
[a)] t-matui@ist.osaka-u.ac.jp
[b)] matusita@ist.osaka-u.ac.jp
[c)] inoue@ist.osaka-u.ac.jp

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，いくつかの変更点を加えた手法を提案する．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．また，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品質を判定できるようになった．

以下，2 節で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 節で提案手法と既存の手法からの差分について説明する．4 節で評価実験の内容と結果，評価実験に用いたデータセットについて説明する．5 節で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 節でまとめと今後の課題について述べる．

2. 背景

本節では研究の背景として，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，先行研究について説明する．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．

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．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[*2] 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 [1,6] が変動する．プログラミングコンテストには Codeforces[*3] 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[*4] 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．

2.2.1 Codeforces

本研究で利用する Codeforces はロシアのプログラミングコンテストサイトであり，2010 年からサービスが提供されている．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 (言語はロシア語・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．Codeforces におけるアクティブユーザー数は 2020 年 5 月 20 日現在，89289 人となっている．

2.2.2 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 [6] 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 [1] 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．

2.3 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は槇原らによってすでに行われている [5]．槇原ら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ており，機械学習の説明変数にはメトリクスと予約語の利用頻度，アルゴリズムには SVM と決定木の 2 つを採用している．

本研究では提案手法の評価の際，槇原らの手法を再現したものと結果を比較する．

3. 提案手法

3.1 目的

2.3 節で説明したように，槇原らによりすでにソースコードの品質の評価は行われている．しかし，槇原らの実験結果では上級者と初級者で分類の精度が違っており，上級者に比べて初級者の精度が悪いことが確認できる．また，槇

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

原らの手法ではプログラミングコンテストの参加者のうち上級者と初級者のみを学習時に利用しており，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ユーザ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

そこで本研究では既存の手法を改善し，3 値分類によって判定可能な対象を広げる手法を提案する．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品質を判定する手法について説明する．また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・初級者の定義を行う．

(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 ソースコードの品質の判定

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品質の判定を行う流れを図 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（STEP1）．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品質であり，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，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．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（STEP2）．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，ソースコードの品質の判定を行う（STEP3）．

以下，3 つの段階それぞれについて詳細を述べる．

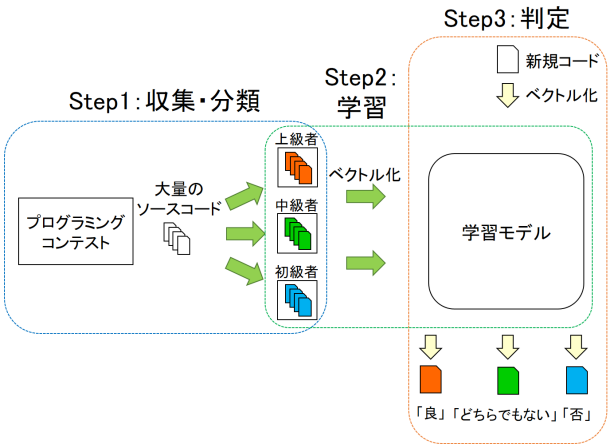

図 1 提案手法の概要

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品質である．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品質である．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 1 に示す．

表 1 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			

これらの予約語は堤の調査 [9] 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．対象の 22 種類のメトリクスを表 2 と表 3 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[*5]を用いて計測できるものであり，表 2 に含まれる 11 種類のメトリクスは堤の調査により，n_statement は初級者に比べて上級者の値が大きくなる傾向があるといったように，上級者，または初級者のソース

[*5] http://www.campwoodsw.com/sourcemonitor.html

コードで値が他方より大きくなる傾向があることがわかっている．表 3 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．これらを合わせて，ソースコード特徴量として 51 次元のベクトルを作成した．ソースコードの品質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．ソースコードの品質は’skill’ という変数名にしている．図 2 にソースコードの品質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，’skill’ 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．

表 2 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の割合
percent_comments	全体の物理行数に占めるコメントの割合

表 3 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

asm	break	case	・・・	n_lines	skill
0	0	0		31	1
0	0	0		55	1
0	0	0		129	0
0	0	0		106	0
0	2	0		43	-1
0	4	0		50	-1

図 2 目的変数’skill’ を追加したベクトル

STEP3：判定

判定を行いたいソースコードに対し，STEP2 と同様の手順で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．

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品質の判定手法と槇原らによるソースコードの品質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 1 に示した予約語 29 種類の利用頻度と表 2 と表 3 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 3 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. 評価

本節では，2 つの実験を通じて 3 節で述べた提案手法の評価を行う．最初に 2 つの実験で利用したデータセットについて説明する．次に，中級者の追加を除いた 3 つの変更点を加えることで，精度が向上されるかどうかを確認するために行った実験について説明する．最後に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ために行った実験について説明する．

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ら [5] も用いており，堤 [9] 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[*6]．

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．データセットの統計情報は表 4 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．提出されたソースコードの言語別提出数の割合を表 5 に示す．ソースコードのうち，90%は C++によって記述されている．表 4 の参加者数は，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．

表 4 データセットの統計情報

収集期間	ファイル数	参加者数
2016/5/19〜2016/11/15	1,644,636	14,520

コンテスト数	問題数	DB サイズ
739	3,218	357MB

[*6] https://sites.google.com/site/miningprogcodeforces/

表 5 提出ソースコードの言語分布

言語	C++	Java	C	Python	その他
割合	90.00%	4.15%	2.08%	1.92%	1.86%

4.1.1 ソースコード

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

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．

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 6 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 7 のような分布であった．

表 6 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 7 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

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．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．

4.2.2 結果・比較

4.2.1 節で説明した実験の結果を表 8 に，既存の手法を再現した結果を表 9 に示す．

表 8 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

表 9 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

これらを比較すると，上級者は再現率ではほとんど違いがないが，適合率では 3 つの差分を加えた手法の方が高く，F 値も向上している．一方，初級者は差分を加えたことで，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 8 の上級者・初級者の F 値を見ると，ほとんど差がない．表 9 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．

4.3 判定対象の拡大の評価実験

本節では 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 10 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．これらのソースコードを用いて 4.2.1 節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 が与えられ，3 値で分類される．

比較のために，既存手法に対して本手法と同様に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 11 のようになった．また，既存の手法ではメトリクスを表 2 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．学習アルゴリズムは決定木と SVM を採用する．

表 10 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

表 11 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 節で説明したもののうち，提案手法による結果を表 12 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 13 に示す．表 12 と表 13 を比較すると，提案手法では上級者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．

表 12 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 13 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

5. 考察

提案手法において，既存手法に加えた差分は以下の 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

この章では，これらの差分を 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．

5.1 上級者・初級者の定義の変更

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 6 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 7 であった．このとき，差分を加えたときの結果を表 14 に示す．

表 14 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 9 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，既存の手法にも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 6 と表 7 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．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い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．

5.2 メトリクスの追加

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 1 と表 2 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 3 を追加した 51 種類となる．このとき得られた結果を表 15 に示す．表 9 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 15 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

5.3 学習アルゴリズムの変更

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 16 に示す．表 9 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 15 の上がり幅より大きい．

表 16 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手法の結果が表 8 となる．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ており，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．また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 8 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．

5.4 中級者の追加

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 節で行ったものと同じであるため，結果は表 13 となる．表 9 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と考えられる．ここで表 11 のレーティングの分布を見ると，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品質の判定では，判定対象のソースコードの品質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，ソースコードの品質を知ることに適していると考えられる．

6. まとめ

本研究では，機械学習によりソースコードの品質を判定する手法を改善し，3 値分類を用いたソースコードの品質を判定する手法を提案した．本手法では，従来手法に対してメトリクスの追加，ランダムフォレストの採用，上級者・初級者の定義を変更することにより，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，上級者ではないソースコードを対象として，レーティングを向上させるための修正案を提示することである．槇原ら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 [5]．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．槇原ら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．

参考文献

[1] Elo, A. E.: The rating of chessplayers, past and present. Arco Pub. (1978).
[2] Goues, C. L., Dewey-Vogt, M., Forrest, S., and Weimer, W.: 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).
[3] Hanam, Q., Brito, F.S.d.M., and Mesbah, A.: Discovering bug patterns in javascript. *ACM SIGCSE Bulletin*, pp. 144–156 (2016).
[4] 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).
[5] 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).
[6] Mirzayanov, M.: Codeforces rating system. `http://codeforces.com/blog/entry/102` (2010).
[7] Mirzayanov, M.: Codeforces: Results of 2018. `https://codeforces.com/blog/entry/65026` (2019).
[8] 堤祥吾，吉田則裕，崔恩瀞，井上克郎：プログラミングコンテスト参加者を対象とした編集作業の特徴調査，情報処理学会研究報告，Vol. 2017-SE-197, No. 6, pp. 1-8 (2017).
[9] 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).
[10] Zhong, H. and Su, Z.: An empirical study on real bug fixes. In *Proc. of ICSE 2015*, pp. 913–923 (2015).