特別研究報告

題目

異なる競技プログラミングのデータセットを用いたソースコード品質判定モデルの汎化性能調査

指導教員
井上 克郎 教授

報告者
服部 文志

令和 4 年 2 月 8 日

大阪大学 基礎工学部 情報科学科

令和 3 年度 特別研究報告

異なる競技プログラミングのデータセットを用いたソースコード品質判定モデルの汎化性能調査

服部 文志

内容梗概

競技プログラミングは，与えられた要件を満たすプログラムを記述する正確さや速さを競う競技であり，プログラミング上級者から初級者まで数多くの人が参加している．Web 上で開催される競技プログラミングにおいては，熟練度を示すための指標としてレーティングが使用されており，コンテストに参加する度に成績によって変動していく．近年，競技プログラミングのレーティングとソースコードの品質の関係を調査する研究が行われている．

槇原はソースコードの特徴量を用いて，機械学習によりソースコードの品質を判定する手法を提案した．松井は槇原の研究を派生させ，判定対象の拡大や判定精度の向上に成功した．しかし，モデルを学習させる訓練と精度を測定するためのテストにおいては，同一の競技プログラミングサイトから収集したソースコードを用いているため，異なる競技プログラミングサイトのコードに対して正確に品質判定を行うことができるかはわかっていない．

そこで本研究では，機械学習を用いたソースコード品質判定モデルを対象として，訓練データと異なるテストデータを用いた場合の予測精度，つまり汎化性能について調査を行った．複数の競技プログラミングサイトからソースコードを収集して新たに 3 つのデータセットを構築し，既存のデータセットと合わせて合計 4 つのデータセットを準備した．各データセットで訓練した 4 つのモデルに対して，テストデータとして同じデータセットを使用した場合の予測精度と，異なるデータセットを使用した場合の予測精度をそれぞれ測定し，比較を行った．その結果，訓練データとテストデータが同じ場合に比べて，異なる場合には予測精度が低くなることがわかった．この結果から，ソースコード品質判定モデルの汎化性能が低いことがわかった．

主な用語

競技プログラミング
機械学習
汎化性能

目 次

1　まえがき

ソフトウェアの開発においてソースコードの編集作業は必須であり，コンピュータサイエンスの研究分野としてソースコードの編集作業に関する研究は数多く行われている．[2, 3, 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ため，ソースコードの編集作業にかかる労力は大きい．特にプログラミングに慣れていない初級者は上級者に比べて編集にかかる時間的コストが大きく，自身が行った編集が適切であるかを判断することも難しいため，それを客観的に判断できるようになることが望ましい．

プログラミングの学習において，ソースコードの品質を判定することは編集作業の支援につながると考える．例えば，あるソースコードの品質が良くないと判定された場合，そのソースコードにはまだ改善の余地が残っていることがわかる．また，そのソースコードを修正してソースコードの品質が良いと判定されるようになった場合には，その修正作業は適切であったと判断することができる．また，ソースコードの品質が判定できることで，品質が良くないソースコードに対して良いソースコードとの差分を提示することができるようになり，プログラミング初学者に対して適切な修正の指針を与えることができるようになる．これにより，初学者はより効率的なプログラミングの学習が可能となる．しかし，ソースコードの品質を判定する基準は，可読性や実行時間など複数存在しており，プログラムの利用目的や作成者によってどの基準を重視するかは異なっている．そのため万人に受け入れられる基準を設けることは難しい．

競技プログラミングにおける初級者と上級者のソースコードの分析が堤 [8] によって行われている．その研究では，上級者と初級者が提出したソースコードを定量的に分析し，使用する予約語の利用頻度やメトリクスの値に差異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を使った機械学習によって定量的かつ自動的にソースコードの品質の良否判定を行い，良くないと判定されたソースコードに対して修正の指針を提示した．[5] また，松井は槇原の研究に加えてメトリクスの追加や学習方法の変更などを行うことにより，判定精度の向上や判定対象の拡大を行った．[6] しかし，槇原と松井はモデルの学習を行うための訓練データと，判定精度を測定するためのテストデータに同一のデータセットを使用していた．そのため，訓練データと異なるデータに対する判定精度，いわゆる汎化性能は明らかになっていない．

そこで本研究では，複数のデータセットを利用してソースコード品質判定モデルの汎化性能の調査を行う．汎化性能の調査を行うために 3 つの競技プログラミングサイトからそれぞれデータセットを構築し，既存のデータセットと合わせた合計 4 つのデータセットを使用した．各データセットを訓練データとして使用した 4 つのモデルを作成して各データセットに対する予測精度を測定し，訓練データと同じデータセットに対する予測精度と異なるデータ

セットに対する予測精度を比較することで汎化性能の調査を行った．また，予測精度に差が生じた原因を，モデルの特徴量重要度やデータセットごとの特徴量の分布の観点から考察した．

2　背景

この章では本研究の背景として，競技プログラミング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述べる．競技プログラミングの種類は多岐にわたるが，本研究ではアルゴリズムやデータ構造に関する問題を時間内に解く形式のものを対象としている．

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

競技プログラミングにおいて，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分野やレベルが異なる様々な問題が収録されており，各問題には問題文，サンプルテストケース，実行時間やメモリ制限等の実行上の制約などが設定されている．ユーザが問題を選択して解答ソースコードを提出することで，システムはそのコードをコンパイルし，事前に用意されているテストケースを用いてコードの正しさやメモリ・実行時間の効率等を自動で判定し，その結果をユーザに通知する．オンラインジャッジシステムの一例としては，国内で代表的な Aizu Online Judge[1]やアメリカの TopCoder[2]が存在する．

2.2　競技プログラミング

競技プログラミングは与えられた要件を満たすプログラムを記述する正確さや速さを競う競技であり，プログラミングコンテストとも呼ばれている．複数の参加者が同一の問題を決められた時間内に解き，オンラインジャッジシステムによって採点が行われ，その採点を元に各ユーザが順位付けされてレーティング [1, 7] が変動する．プログラミングコンテストには ACM ICPC のように複数の参加者がチームを組んで回答する形式のものも存在するが，本研究では個人で参加して解答する形式のプログラミングコンテストを対象とする．

2.2.1　概要

プログラミングコンテストの実際の流れについて，本研究でデータセットとして用いる Codeforces[3]を例として説明する．Codeforces で開催される一般的なコンテストでは，コンテストの開始時間と終了時間が指定されており，コンテストの開始時間と同時に問題が公開され，参加者は終了時間までに解答ソースコードを提出する．参加者が解答ソースコードを

[1] https://judge.u-aizu.ac.jp/onlinejudge/
[2] https://www.topcoder.com/
[3] https://codeforces.com/

提出すると即座にオンラインジャッジシステムによって事前テストが行われ，その結果が参加者に通知される．テストを通過しなかった場合でも，参加者は時間内であれば解答を何度でも再提出できるが，誤答する度に減点が科される．コンテスト終了後に問題の難易度や解答にかかった時間，誤答回数によって最終的な得点とそれに伴う順位が決定され，レーティングが変動する．

本研究で利用する Codeforces の基本情報は以下のようになっている．

- 開催頻度：偏りはあるがおおよそ週に 1 回以上
- 参加人数：毎コンテスト 10000〜15000 人程度
- 国籍：全世界から参加（言語はロシア語・英語）

また直近 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して定義しているが，Codeforces におけるアクティブユーザー数は 2022 年 1 月 28 日現在 59353 人となっている．

2.2.2 レーティングシステム

Codeforces ではユーザの熟練度を表す値としてレーティングを用いている．レーティングのシステムは，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に似た計算方式を採用しており，コンテストごとにユーザの順位に応じてレーティングが更新される．本研究ではこのレーティングが高いユーザをプログラミングの習熟度が高いユーザとして扱う．

イロレーティングではユーザ A とユーザ 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 においては，これは A が B の得点を上回る確率と解釈できる．

Codeforces のレーティング変更計算アルゴリズムを Algorithm 1 に示す．(1) の for 文ではおおよそのレーティングの変動を計算している．$seed_i$ はレーティング r_i の参加者の順位の期待値を表し，m_i を $seed_i$ と実際の順位 $rank_i$ の幾何平均とする．ここで，$getSeed(R_i) = m_i$ となるようなレーティング R_i の探索を行い，この R_i と r_i の平均をユーザ i の暫定的なレーティングとする．また，上位ユーザのレーティングのインフレを避けるために (2) の調整を行っている．最終的に (1) と (2) によって計算されたレーティングの変動を r_i に加えたものがユーザ i のレーティングとなる．

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

$d_i \leftarrow d_i + inc$;

$r'_i + d_i$;

2.3 先行研究

ソースコードの品質判定を機械学習を用いて行う研究が槇原 [5] と松井 [6] によって行われている．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．機械学習は大きく以下の 3 種類に分類することができる．

教師あり学習 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行う．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 特徴を表すデータのみを入力して学習を行う．入力データの構造分析などに使われることが多い．

強化学習 ある環境の中で得られる報酬が最大となるように学習を行う．具体例としては囲碁 AI が挙げられ，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．

槇原と松井はソースコードの品質を正解とする教師あり学習を用いて，ソースコードの品質判定を行った．

教師あり学習において，特徴を表すデータを説明変数，その正解となるデータを目的変数と呼び，これらを用いて学習を行う．その後学習を終えたモデルに新しい説明変数を与え，目的変数を予測させる．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本来の目的変数とどの程度一致しているかで判断する．教師あり学習は説明変数の性質によって大きく以下の 2 つにわけることができる．

分類問題 説明変数とし”性別”のような離散値を扱う．主な学習アルゴリズムとして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る．

回帰問題 説明変数として”身長”のような連続値を扱う．主な学習アルゴリズムとして線形回帰，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品質判定を”良”と”否”の 2 つの離散値を予測する分類問題として扱い，SVM と決定木を用いて学習を行った．一方松井は，判定対象の拡大のためにソースコードの品質判定を”良”，”否”，”どちらでもない”の 3 つの離散値を予測する分類問題として扱い，ランダムフォレストを用いて学習を行った．

松井により作成されたモデルの評価結果は表 1 のようにまとめられていた．

表 1: 松井の手法の結果

	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

3 調査方法

3.1 目的

2.3 節で示したように，松井が作成したモデルでは一定の精度でソースコードの品質を予測できることがわかっている．松井が行った研究では，モデルの予測精度の評価のために 10 分割交差検証を用いた．10 分割交差検証の手順を以下に示す．

1. あらかじめ用意したデータセットを，各グループの数が等しくなるように 10 個のグループに分割する
2. 10 個のグループのうち，9 個のデータを用いてモデルの訓練を行い，残った 1 つのデータでモデルのテストを行って予測精度を測定する
3. 手順 2. を，テストで用いるデータが毎回異なるように 10 回繰り返し行い，各モデルの予測精度の平均値をとる

この方法はモデルの訓練とテストに同じデータセットのデータを使用しているため，訓練とテストに異なるデータセットのデータを使用した場合の予測精度については評価できていない．そのため，本研究では訓練データとテストデータを別のデータセットから作成してモデルの訓練とテストを行うことで，松井が作成したソースコード品質判定モデルの汎化性能の調査を行う．以降の章では訓練とテストに用いる各データセットの詳細を述べる．

3.2 使用するデータセット

今回の調査では，次にあげる 4 つのデータセットを用いる．

Codeforces16

このデータセットはロシア最大級の競技プログラミングサイトである Codeforces からソースコードを収集したものであり，堤 [8] によって作成された．このデータセットは，ユーザが問題への解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データ

表 2: Codeforces16 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

ベースの 2 種類から構成される．提出履歴データベースの内容は表 3 で，データセットの統計情報は表 2 で示している．本研究ではこのデータセットとは別に，Codeforces から収集したデータセットを使用するため，両者を区別するため，このデータセットを以降，Codeforces16 と呼ぶ．

以下では提出履歴データベースに含まれるテーブルのうち，本研究で利用した Participant テーブルと File テーブルの詳細を述べる．

Participant 2016/5/19〜2016/11/15 の期間中に Codeforces で開催されたコンテストに 1 回以上参加したユーザの情報を収集したテーブルである．各ユーザの情報は，表 3 に示す通りユーザ名，最大レーティング，レーティングの 3 種類の項目で構成されている．Codeforces は一意のユーザ ID を提供していないため，本データセットにおいてはユーザ名を ID としている．また，レーティングはコンテストに参加する度に変動するが，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．このテーブルは，ソースコードデータの総数と同じ 1,644,636 行のデータを含む．各提出履歴に付与される一意の提出 ID と，提出対象である問題の ID から対応するソースコードページの URL を構築することができ，url カラムに格納されている．

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 1 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．本研究ではこのデータセットのソースコードのうち，C++で記述されたソースコードのみをモデルの訓練データとして用いる．

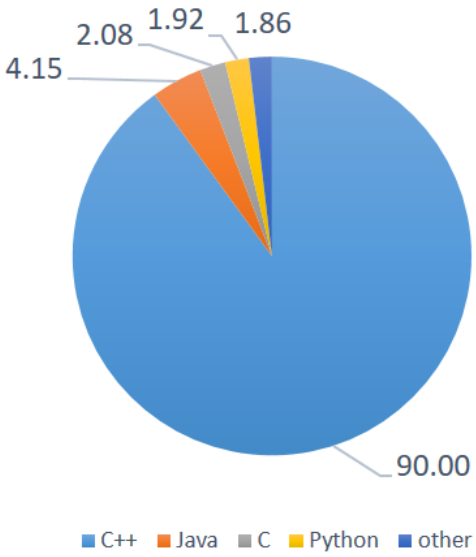

図 1: 提出ソースコードの言語分布

表 3: 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

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
	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
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-	problem_id	PK	対応する *problem_id*	String

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
Submission-Statistics				
	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
	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	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
			トークンベースの編集距離	
	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

Codeforces21

Codeforces のソースコードは Codeforces16 データセットでも使用しているが，Codeforces16 は 2016 年に作成されたものである．2021 年現在は，PC やプログラミング学習の普及によって 2016 年に比べてユーザ数の増加やそれに伴うレーティング分布の変化などが考えられるため，新たにソースコードを収集してデータセットを構築した．データセットの構成は，訓練データセットと同じく表 3 のようになっている．新たに構築したデータセットの統計情報を表 4 に示す．Codeforces16 データセットと区別をするために，このデータセットを以降，Codeforces21 と呼ぶことにする．

表 4: Codeforces21 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
2021/12/1〜2021/12/31	1,752,427	59,353	1547	7490	432MB

AtCoder

AtCoder は日本最大級の競技プログラミングサイトであり，基本情報は以下のようになっている．

- 開催頻度：週に 1 回以上

- 参加人数：毎コンテスト 8000 人程度
- 国籍：全世界から参加（言語は日本語・英語）

AtCoder におけるレーティングシステムは Codeforces と同様，イロレーティングを採用している．ただし Codeforces のレーティングシステムが 1500 からスタートして増減するのに対して，AtCoder は 0 からスタートし，0 以上の値でレーティングが増減していく．

AtCoder のデータセットは，AtCoder に提出された解答ソースコードと，各ソースコードのメタデータをまとめた CSV 形式の提出情報ファイルから構成されている．提出情報ファイルは，AtCoder Problems[4]で公開されているものを使用している．AtCoder Problems は，AtCoder に提出されたソースコードをクロールして管理しているウェブアプリであり，AtCoder の公式ではなく有志により作成されている．提出情報ファイルの構成を表 5 に示す．提出情報ファイルの構成要素のうち，レーティング情報についてはもともと含まれておらず，新たに自分で追加したものとなっている．各ソースコードのユーザ ID から提出者のユーザページへアクセスし，スクレイピングによってレーティング情報を取得した．このレーティングの値は 2021/6/1 時点のものになっている．

解答ソースコードについては，提出情報ファイルを基にスクレイピングで収集を行った．収集したソースコードは，提出情報ファイルに含まれるソースコードのうち，2021/1/1〜2021/6/1 の期間に提出され，C++で記述されたソースコードである．ソースコードは，提出情報ファイルに含まれる提出 ID とコンテスト ID を基にソースコードページにアクセスし，スクレイピングにより収集した．本データセットには，1,459,964 個のソースコードが含まれている．

Aizu Online Judge

Aizu Online Judge（以降，AOJ）は競技プログラミング問題のオンラインジャッジシステムサイトである．Codeforces や AtCoder とは異なり AOJ 上ではコンテストを開催していないものの，データ構造やアルゴリズムを学ぶための演習問題から，JOI（日本情報オリンピック）や ICPC（国際大学対抗プログラミングコンテスト）の過去問題まで，幅広い問題が用意されている．

AOJ では Codeforces や AtCoder とは異なる独自のレーティングシステムを採用している．AOJ において問題 P に正解した際に得られるポイントは，$X/(Y+min(LIMIT, N_P))$ である．ここで，N_P は問題 P に正解したユーザ数であり，X，Y，$LIMIT$ はそれぞれ 2022/2/5 現在 $X = 70$，$Y = 1$，$LIMIT = 400$ となっている．これらの値はユーザ数や問題数，解答数に応じて変更される場合があるが，2022/2/5 現在の最終更新日は 2016/8/28 となってい

[4] https://kenkoooo.com/atcoder/

表 5: AtCoder データセットの提出情報ファイルの構成

名前	型	単位	説明
提出 ID	int	none	ソースコードに一意に付与される ID
提出日時	int	UNIX 秒	ソースコードが提出された時間
問題 ID	string	none	ソースコードが提出された問題の ID
コンテスト ID	string	none	ソースコードが提出されたコンテストの ID
ユーザ ID	string	none	ソースコードを提出したユーザのユーザ名
言語	string	none	ソースコードが記述されたプログラミング言語
得点	int	none	その解答により得られた点数
コード長	int	バイト	ソースコードの長さ
提出結果	string	none	ソースコードの提出結果
実行時間	int	ミリ秒	ソースコードの実行にかかる時間
レーティング	int	none	提出したユーザのレーティング

る．あるユーザが正解したすべての問題において，上記の式で得られるポイントを加算したものがそのユーザのレーティングとなる．この式からわかるように，正解したユーザが少ない問題，つまり難しい問題に正解した際は大幅にレーティングが増加するが，正解したユーザが多い問題，つまり簡単な問題に正解した際はわずかしかレーティングが増えないシステムとなっている．

AOJ のデータセットは AtCoder のデータセットと同様に，AOJ に提出された解答ソースコードと，各ソースコードのメタデータをまとめた CSV 形式の提出情報ファイルから構成されている．提出情報ファイルと解答ソースコードは AOJ のホームページで公開されているものを使用している．これらのデータは研究，または教育目的での利用のみ可能となっている．提出情報ファイルの構成を表 6 に示す．提出情報ファイルの構成要素のうち，レーティング情報についてはもともと含まれておらず，新たに自分で追加したものとなっている．各ソースコードのユーザ ID を基に提出者のユーザページへアクセスし，スクレイピングによってレーティング情報を取得した．このレーティングの値は 2021/12/20 時点のものになっている．

ソースコードは 2020/12/5〜2021/12/5 に提出されたソースコードのうち，C++で記述され，なおかつ提出したユーザのレーティングが 0 でないもののみを抽出している．このデータセットには 48962 個のソースコードが含まれている．

表 6: AOJ データセットの提出情報ファイルの構成

名前	型	単位	説明
提出 ID	int	none	ソースコードに一意に付与される ID
ユーザ ID	string	none	ソースコードを提出したユーザのユーザ名
問題 ID	string	none	ソースコードが提出された問題の ID
言語	string	none	ソースコードが記述されたプログラミング言語
正解率	string	none	通過したテスト数 / 全テストケース数
提出結果	string	none	ソースコードの提出結果
実行時間	int	1/100 秒	ソースコードの実行にかかる時間
メモリ使用量	int	キロバイト	ソースコードの実行時に使用するメモリ量
コード長	int	バイト	ソースコードの長さ
提出日時	int	UNIX 秒	ソースコードが提出された時間
判定日時	int	UNIX 秒	ソースコードの正誤が判定された時間
レーティング	int	none	提出したユーザのレーティング

3.3 評価対象モデル

本研究では，松井が作成した 3 値分類によるソースコード品質判定モデル [6] を評価対象としている．このモデルの判定手順を図 2 に示す．判定を行うまでの流れは大きく 3 段階に分けることができる．初めに，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 3 クラスに分類する．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，これを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”ソースコードとして定義する．そして説明変数と目的変数をベクトル化して機械学習を行う．こうして学習したモデルに新規のソースコードの特徴量をベクトル化して入力することで，ソースコードの品質を判定することができる．

以下で 3 つの段階それぞれについて詳細を述べる．

STEP1：収集・分類

ソースコードは競技プログラミングサイトから収集する．収集したソースコードを以下の手順に従って，上級者，中級者，初級者の 3 クラスに分類する．分類の対象は収集したソースコードのうち，”GNU C++”，”GNU C++11”，”GNUC++14”で記述されたソースコードとする．

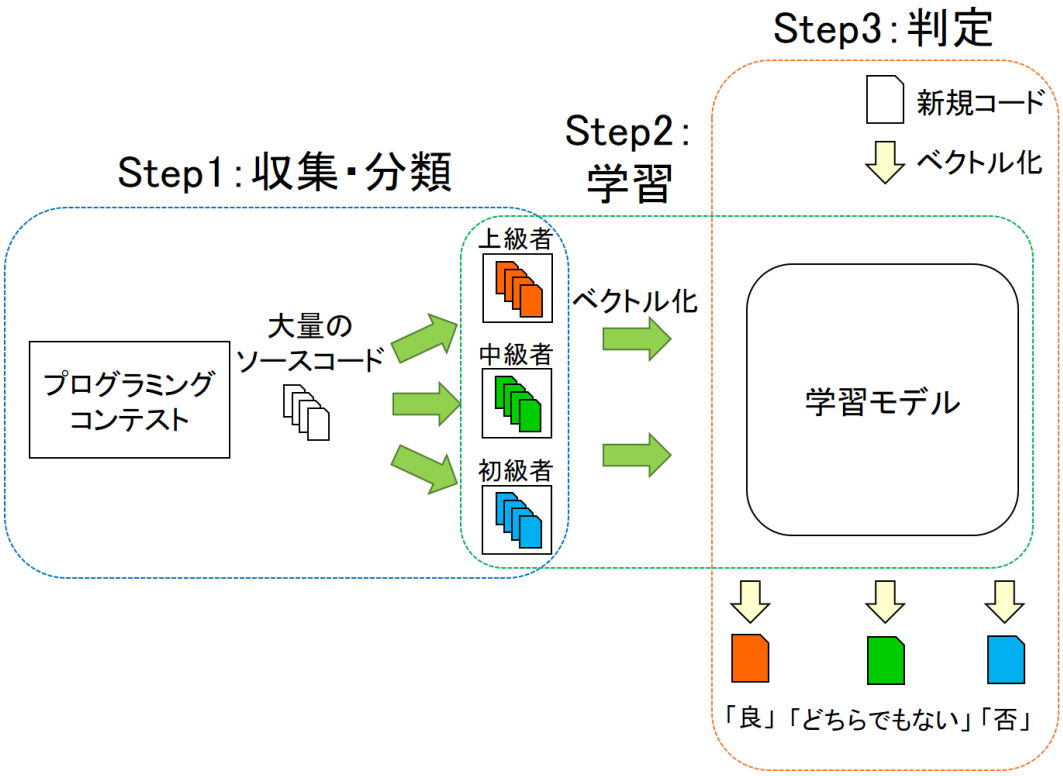

図 2: ソースコードの品質の判定手順

1. ソースコードを提出者のレーティングの降順に並べ替える
2. 並べ替え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 分割したグループのうち，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，その中間となる 2 つのグループを中級者のソースコードとする

STEP2：学習

学習を行う前に，学習に使用する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．学習に用いる値は，各ソースコード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用いる．使用する 29 種類の予約語を表 7 に，使用する 22 種類のメトリクスを 8 と表 9 に示す．これらの予約語は堤の調査 [8] により，上級者または初級者の利用頻度が高いことがわかっている．また，メトリクスについては 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 8 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において値が大きくなることがわかっている．表 9 に含まれる 11 種類のメトリクスは，堤の調査対象とはならなかったが説明変数として利用したメトリクスである．statements_at_block_level_3〜7 については表示を省略している．これらを統合してベクトルにすることで，ソースコード特徴量である 51 次元のベクトルを作成することができる．図 3 にソースコード特徴量のベクトルの一部を示す．そして，このソースコード特徴量ベクトルに，上級者が書いた”良い”ソースコードには 1，初級者が書いた”良くない”ソースコードには-1，中級者が書いた”どちらでもない”ソースコードには 0 をそれぞれ追加し，52 次元のベクトルを作成する．図 4 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．ソースコードの品質は”skill”という変数名で表している．最後にソースコード特徴量を説明変数，品質を目的変数として機械学習を行う．モデルの学習に使用する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 を用いる．

表 7: 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			

STEP3：判定

品質を判定したいソースコードに対し，STEP2 と同様の手順で図 3 のようなソースコード特徴量から成る 51 次元のベクトルを作成する．学習済みのモデルに対してベクトルを入力すると，”良い”，”良くない”，”どちらでもない”のいずれかの判定結果が出力される．入力したソースコードが上級者のものである場合は，判定結果が”良い”であれば正しく判定が行えているということになる．

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

表 8: 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

表 9: 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	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 3: 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 4: 目的変数”skill”を追加したベクトル

3.4 汎化性能の測定

3.4.1 評価指標

本研究では，モデルの汎化性能の評価指標として f 値を用いる．本研究での true positive(TP)，true negetive(TN)，false positeve(FP)，false negative(FN) について上級者の f 値を計測する場合の例を表 10 に示す.

表 10: 上級者の f 値を求める際の混同行列

		実際のソースコード品質		
		上級者	中級者	初級者
モデルの予測結果	上級者	TP	FP	FP
	中級者	FN	TN	TN
	初級者	FN	TN	TN

適合率（precision）は，TP と FP の合計に対する TP の割合である．この指標は予測結果の誤りの少なさを表している．再現率（recall）は TP と FN の合計に対する TP の割合である．この指標は予測結果の見逃しの少なさを表している．これらの 2 つの指標はトレードオフの関係になっており，例えばすべてのソースコードに対して”良い”と予測した場合，適合率は 1 となるが，再現率は低い値となってしまう．そのため，これら 2 つの評価指標を総合的に判断するための評価指標として f 値を用いる．f 値は適合率と再現率の調和平均であり，以下の式で表される.

$$f = \frac{2 \times precision \times recall}{precision + recall}$$

3.4.2 計測手順

ソースコード品質判定モデルの汎化性能は以下の手順で調査する.

1. 訓練データとテストデータで同じデータセットを使用した場合の予測精度を測定する，ここでは 3.1 節で示した 10 分割交差検証によって精度を測定する.

2. 訓練データとテストデータで異なるデータセットを使用した場合の精度を測定する．各データセットを訓練データとして使用する 4 つのモデルを作成し，各モデルにおいて訓練データと異なる 3 つのデータセットに対する予測精度を測定する.

3. 各データセットにおいて 1. と 2. の予測精度を比較する．1. の精度に比べて 2. の精度が低い場合にはこのモデルの汎化性能は低いといえる．反対に，1. と 2. の精度が同程度であれば，このモデルの汎化性能は高いといえる.

4　調査結果

Codeforces16 で訓練したモデルの各データセットに対する予測精度を図 5 に示す．この結果では，Codeforces16 に対する予測精度は上級者で 0.79，中級者で 0.82，初級者で 0.72 であるのに対して，他のデータセットに対する予測精度は Codeforces16 に対する予測精度と比べて，上級者と初級者では約 0.3 低く，中級者では約 0.2 低くなっている．このことから訓練データと同じデータセットに対する予測精度は高いが，訓練データと異なるデータセットに対する予測精度は低いことがわかる．また，各データセットで訓練したモデルに対する予測精度を図 6 と表 11 に示す．表中の CF16 は Codeforces16 のデータセット，CF21 は Codeforces21 のデータセット，AC は AtCoder のデータセット，AOJ は AOJ のデータセットを表している．この図と表から，Codeforces16 以外のデータセットにおいても，訓練データと同じデータセットに対する予測精度は高いが，訓練データと異なるデータセットに対する予測精度は低いことがわかる．

上記の調査結果から，ソースコード品質判定モデルの汎化性能は低いことが分かった．また，訓練データと同じデータセットに対する予測精度はどのデータセットでも高いことから，ソースコードの品質判定を行う際には，判定したいソースコードと同じデータセットで学習したモデルを使用したほうが良いことが分かった．

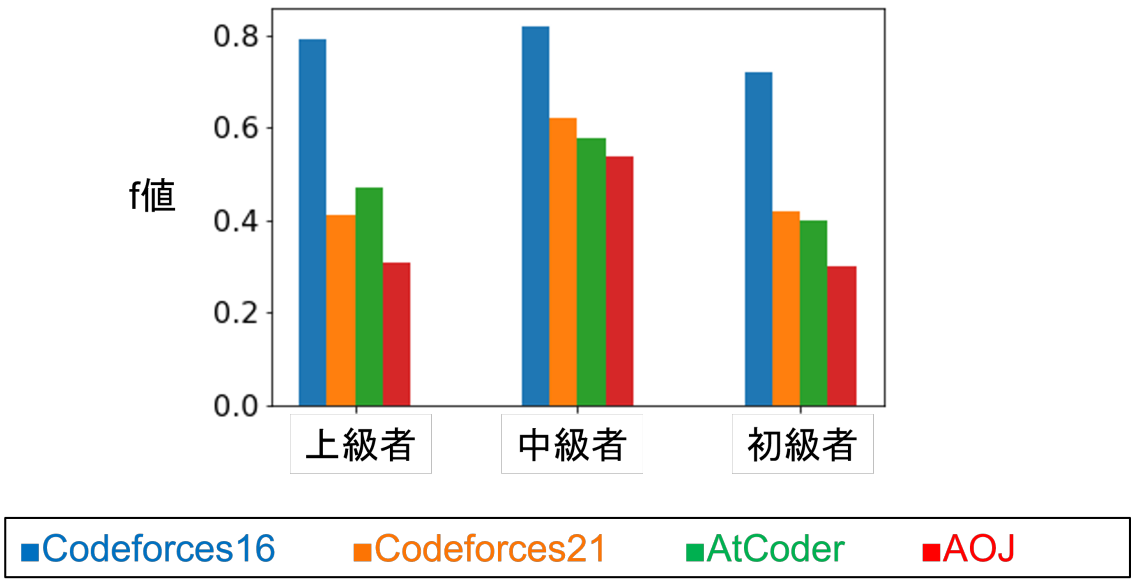

図 5: Codeforces16 で訓練したモデルの予測精度

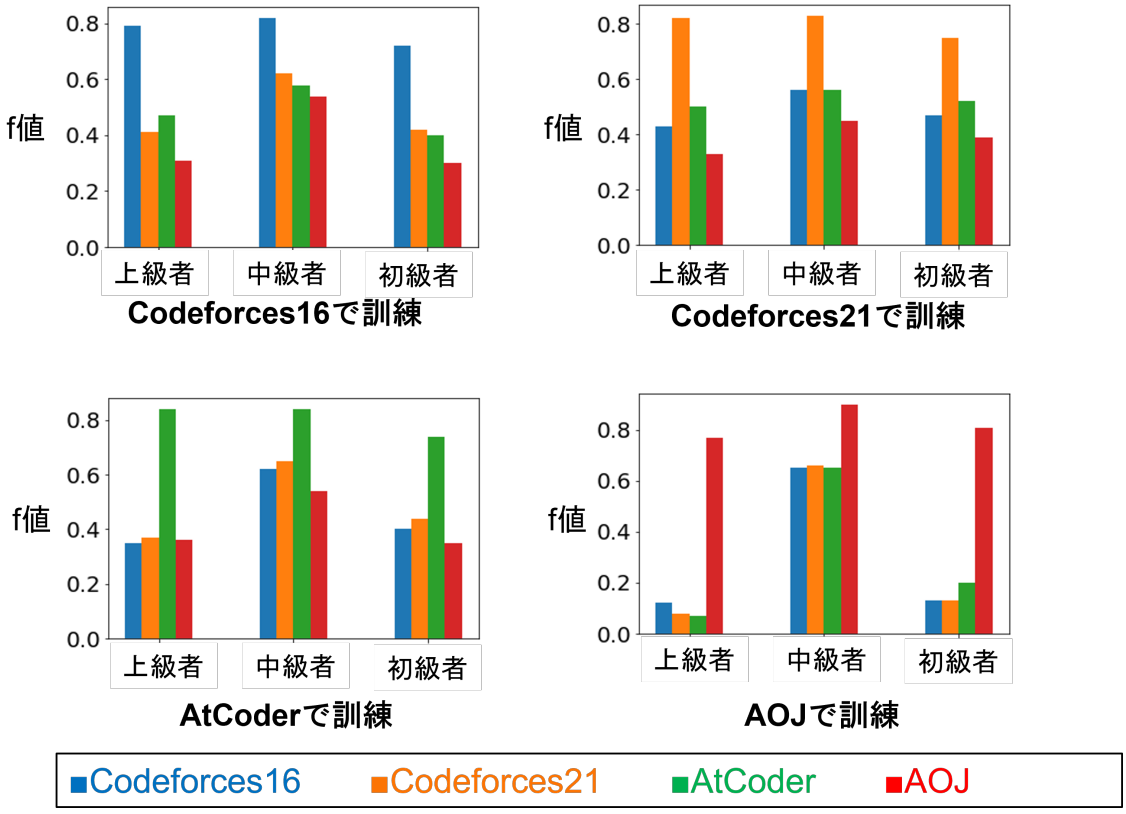

図 6: 各データセットで訓練したモデルの予測精度

表 11: 各データセットにおける予測精度

訓練データ		テストデータ			
		CF16	CF21	AC	AOJ
CF16	上級者	0.79	0.41	0.47	0.31
	中級者	0.82	0.62	0.58	0.54
	初級者	0.72	0.42	0.40	0.30
CF21	上級者	0.43	0.82	0.50	0.33
	中級者	0.56	0.83	0.56	0.45
	初級者	0.47	0.75	0.52	0.39
AC	上級者	0.35	0.37	0.84	0.36
	中級者	0.62	0.65	0.84	0.54
	初級者	0.40	0.44	0.74	0.35
AOJ	上級者	0.12	0.08	0.07	0.77
	中級者	0.65	0.66	0.65	0.90
	初級者	0.13	0.13	0.20	0.81

5 考察

4 章の調査結果から，松井が作成したソースコード品質判定モデルの汎化性能は低いことがわかった．この章では，訓練とテストで異なるデータセットを使用した際に予測精度が下がってしまった原因を考察する．

5.1 特徴量重要度に関する考察

異なるデータセットを使用した際に予測精度が下がってしまった原因の一つとして，データセットごとに予測で重視されるべき特徴量が異なっていることが考えられる．例えば，Codeforces16 で上級者と初級者の間でソースコードの行数が大幅に異なっている場合，モデルは行数の特徴量を重視してソースコードの品質予測を行うことになる．しかし，Codeforces21 でソースコードの行数に上級者・初級者間の差があまりない場合，行数を重視して予測を行うモデルでは正確に予測ができないことが考えられる．そこで，追加調査として各データセットを訓練データとして使用した際のモデルの特徴量重要度の比較を行う．以降の節では，特徴量重要の説明のためにランダムフォレストのアルゴリズムを説明し，その後特徴量重要度の詳細について述べる．

5.1.1 ランダムフォレスト

本研究においてモデルの学習に使用するアルゴリズムであるランダムフォレストは，少しずつ異なる決定木を多数構築し，それらの予測結果の多数決をとることで予測を行う手法である．決定木は木構造の分類モデルであり，各ノードに分類条件が割り当てられ，その分類条件に従ってノードを移動していくことで最終的な予測結果が得られる．決定木の例として，身長と体重から性別を予測するモデルを図 7 に示す.

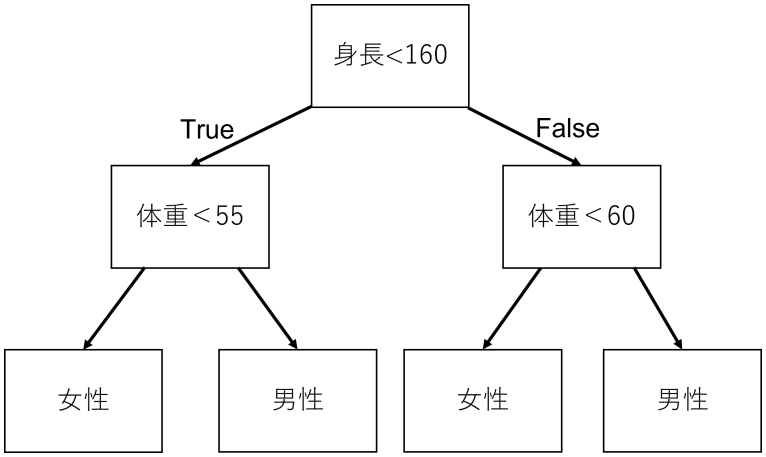

図 7: 身長と体重から性別を予測する決定木の例

決定木単体では過学習を起こしやすいが，ランダムフォレストはブーストラップにより複数の決定木モデルを並列的に学習させるため，このデメリットを回避することができる．以下にランダムフォレストによる品質予測の手順を示す.

1. 元データからランダムにデータをブーストラップでサンプリングし，N グループ分のデータセットを作る．ブーストラップは元データから一部のデータを復元抽出によりサンプリングする方法であり，復元抽出では同じデータが複数回選択される場合もある．
2. Nグループそれぞれで決定木を構築する．
3. 各決定木で予測を行う．
4. 各決定木の予測結果の多数決を取り，最終的なモデルの予測結果とする．

5.1.2 特徴量重要度

特徴量重要度とは，ランダムフォレストにおいて，その特徴量による分割がデータの分類にどれだけ寄与しているかを測る指標である．特徴量重要度は以下に示すジニ不純度を基に計算される．

$$G(k) = \sum_{i=1}^{n} p(i) \times (1 - p(i))$$

各変数の定義は以下の通りである

$$\begin{aligned} G(k) &:= \text{あるノード } k \text{ における不純度} \\ n &:= \text{ターゲットラベルの数} \\ p(i) &:= \text{あるノード } k \text{ におけるターゲットラベル } i \text{ の頻度} \end{aligned}$$

ジニ不純度はノードごとにターゲットがどれくらい分類できていないかを測る指標であり，上記の定義からあるノードにおいて完全にターゲットが分類される場合にはジニ不純度は 0 になる．このジニ不純度を用いて，以下の式で特徴量重要度は定義される．

$$I(j) = \sum_{i=1}^{n \in F(j)} (N_{parent}(i) \times G_{parent}(i)) - (N_{left\_child}(i) \times G_{left\_child}(i) + N_{right\_child}(i) \times G_{right\_child}(i))$$

各変数の定義は以下のとおりである．

$$\begin{aligned} I(j) &:= \text{ある特徴量 } j \text{ における重要度} \\ F(j) &:= \text{ある特徴量 } j \text{ が分割対象となるノードの集合} \\ N_{parent}(i) &:= \text{あるノード } i \text{ におけるサンプル数} \\ N_{left\_child}(i) &:= \text{あるノード } i \text{ の子ノードのうち左側のノードのサンプル数} \\ N_{right\_child}(i) &:= \text{あるノード } i \text{ の子ノードのうち右側のノードのサンプル数} \\ G_{parent}(i) &:= \text{あるノード } i \text{ におけるジニ不純度} \\ G_{left\_child}(i) &:= \text{あるノード } i \text{ の子ノードのうち左側のノードにおけるジニ不純度} \\ G_{right\_child}(i) &:= \text{あるノード } i \text{ の子ノードのうち右側のノードにおけるジニ不純度} \end{aligned}$$

上記の式から特徴量重要度は，ある特徴量で分割することでどれだけジニ不純度を下げられるかを表す値であると解釈することができる．特徴量重要度は 0〜1 の値をとる相対的な値となっており，全特徴量の重要度の和は 1 となる．

5.1.3 特徴量重要度の測定結果

各データセットを訓練データとして使用した場合の各モデルの特徴量重要度のうち，全モデルにおいて 0.01 以上の値となった 20 個の特徴量を抽出し，20 個の中での順位をまとめた

ものを表 12 に示す．表中の CF16 は Codeforces16 のデータセット，CF21 は Codeforces21 のデータセット，AC は AtCoder のデータセット，AOJ は AOJ のデータセットを表している．

表 12: 各特徴量の順位

特徴量	特徴量の順位			
	CF16	CF21	AC	AOJ
Lines	1	1	1	1
Average Block Depth	2	4	4	2
Statements at block level 0	3	3	2	4
Statements	4	2	3	3
Percent Branch Statements	5	5	6	5
Statements at block level 1	6	6	5	6
Statements at block level 2	7	7	8	7
Percent Lines with Comments	8	8	7	9
Average Complexity*	9	9	10	8
Statements at block level 3	10	11	15	11
Maximum Complexity*	11	13	13	13
return	12	10	9	12
for	13	12	11	10
if	14	14	12	14
else	15	18	17	16
typedef	16	19	16	18
Statements at block level 4	17	17	19	17
Functions	18	15	14	15
while	19	16	18	19
Maximum Block Depth	20	20	20	20

この表から，各データセットにおいて予測の際に重視される特徴量は一致している傾向があることがわかる．このことから，各データセットにおいて上級者・初級者間で差が出る特徴量はおおむね一致しているということが言える．そのため考察の最初で述べた，各データセットにおいて差が出る特徴量が異なるために予測が正確に行われていないという考えは誤りであったことがわかった．

5.2　特徴量の分布に関する考察

予測が正確に行われない別の原因として，各データセットごとに上級者，中級者，初級者それぞれでの各特徴量の分布が異なっていることが考えられる．ランダムフォレストで用いる決定木では各ノードに分類に用いる特徴量と閾値が決定されるが，この閾値は訓練データをうまく分類できるような値が設定されるため，データの分布が異なるデータセットでは予測が正確に行われないと考えられる．

そこで実際に特徴量の分布をグラフ化して比較を行った．最も特徴量重要度が高い Lines の，Codeforces16 データセットと Codeforces21 データセットにおける上級者・中級者・初級者それぞれの分布をヴァイオリン図で表したものを図 8 に示す．各グループにおける上位 5%のデータは外れ値として除外している．

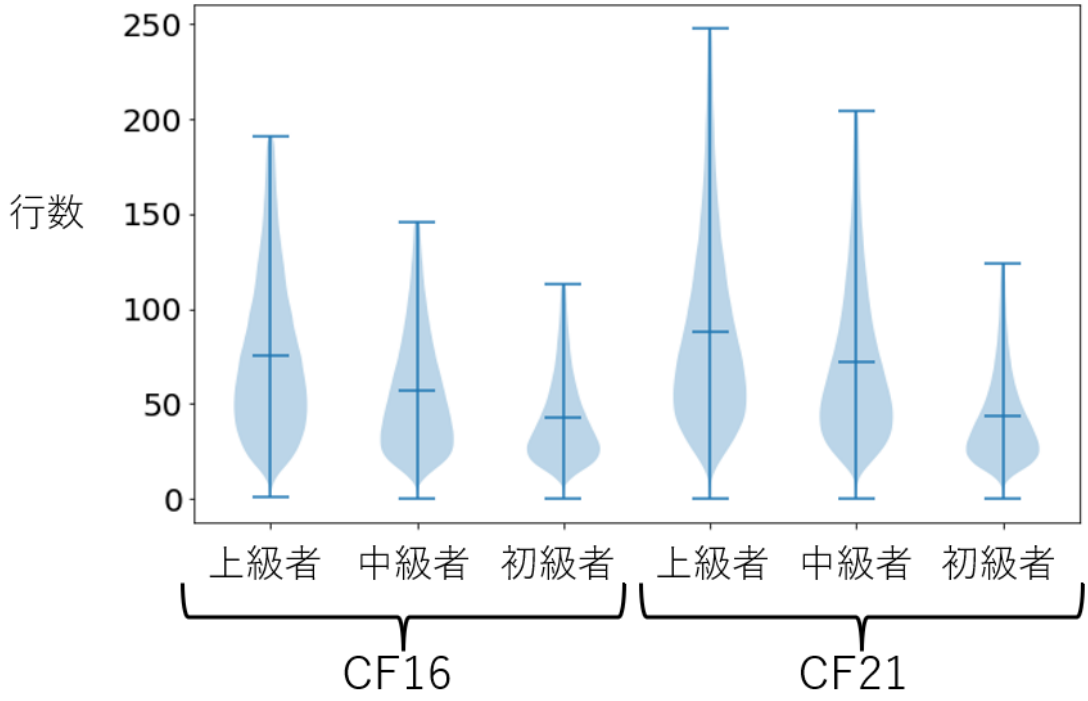

図 8: Lines の各グループにおける分布. 左から順に Codeforces16 の上級者，中級者，初級者，Codeforces21 の上級者，中級者，初級者の分布を表す．

この図において，Codeforces16 で訓練したモデルが分類を行う閾値の 1 つとして図 9 に点線で示した 150 という値が考えられる．この分類を行うことにより，Lines が 150 よりも大きいグループには上級者しか含まれなくなるため，このグループに対するこれ以上の分類は不要となる．しかし，この閾値を Codeforces21 に同様に適用した場合は 16%の中級者のデータも含まれるため，この中級者のデータに対する予測結果は間違ったものになってしま

う．他の特徴量についても分布が完全に一致しているものはないため，訓練データでは完全に分類できていても，異なるデータセットのデータに対しては十分に分類しきれず，予測結果が間違ったものになってしまうと考えられる．

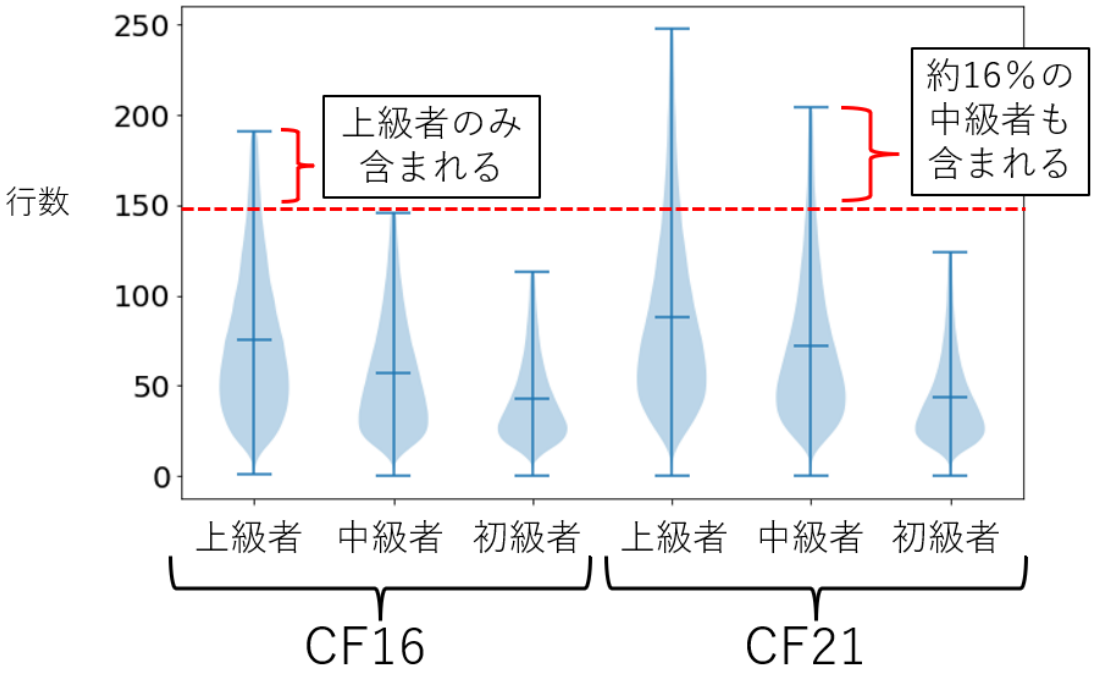

図 9: 150 を閾値とした場合の分類イメージ

6　まとめ

本研究では異なる競技プログラミングのデータセットを用いて，ソースコード品質判定モデルの汎化性能の調査を行った．精度測定に用いるためのデータセットを新たに構築し，それらを用いてモデルの予測精度を測定することにより，訓練データとテストデータが異なる場合，予測精度は低くなることが分かった．またこの結果からソースコード品質判定モデルの汎化性能は低いということが分かった．さらに，特徴量重要度の測定やデータセットに含まれるデータの分布の比較により，予測精度が低下してしまう原因の考察を行った．

今後の課題としては，汎化性能の高いモデルの構築が挙げられる．本研究では汎化性能の調査と原因の考察しか行っていないため，今回得られた知見を活かして学習アルゴリズムの変更やデータセットの構成を変更することにより，汎化性能が高いモデルを構築することが可能になると考えられる．

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，研究の方針について直接の御指導及び御助言をして頂きました．本論文の完成は松下准教授のおかげであると，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 松井智寛 氏には，ツールを提供していただくなど，様々なご支援を賜りました．心より深く感謝いたします．

最後に，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は，本論文の執筆にあたって様々な場面で支えていただきました．皆様のおかげで本論文を完成させることができました．心より深く感謝しております．

参考文献

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

[2] 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.

[3] Quinn Hanam, Fernando S. de M. Brito, and Ali Mesbah. 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] 松井智寛．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，大阪大学基礎工学部情報科学科卒業論文，2020.

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

[8] 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.

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