修士学位論文

題目

競技プログラミングデータセットにおける独立性が機械学習モデルに与える影響調査

指導教員
肥後 芳樹 教授

報告者
服部 文志

令和 6 年 2 月 1 日

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 ソフトウェア工学講座

令和 5 年度 修士学位論文

競技プログラミングデータセットにおける独立性が機械学習モデルに与える影響調査

服部 文志

内容梗概

近年，機械学習を用いたアプローチが様々な分野で増えている．特にソースコード解析の分野では，バグの修復やコードの補完，メソッドの命名等多くのタスクで機械学習モデルが利用されている．機械学習は大量のデータの中から共通する規則やパターンを学習することで未知のデータに対する予測を行うため，使用するデータの質や扱い方がモデルの精度に直結している．データの質や扱い方には様々な要因があり，それらを正しく理解して適切に扱うことがモデルの正確な精度を測定する上で重要となる．実際に，データセットに重複が含まれることや，データセットの分割を不適切な方法で行うことでモデルの精度が本来の性能よりも上昇してしまうことが報告されている．

このような精度上昇の問題を抱えているモデルとして，本研究ではコーディング能力判定モデルに着目する．このモデルは，ソフトウェア開発者のコーディング能力を客観的に判定することを目的として提案されたモデルであり，競技プログラミングのソースコードとレーティングを学習に利用している．競技プログラミングのデータセットには，同一のユーザが提出したソースコードや，同一の問題に提出されたソースコードが複数含まれている．また同一のユーザが同一の問題に提出したコード，つまり編集量が少なくほぼ重複しているコードも含まれている．モデル提案時の精度評価では，モデルの学習のための訓練セットとモデルの評価のためのテストセットとして，同一のデータセットをランダムに分割したものを利用している．そのため訓練・テスト間で重複したコードや，同一のユーザが提出した関連性を持つコードが含まれており，これが原因となってモデル精度が本来の性能よりも高くなっている可能性がある．

そこで本研究では，コーディング能力判定モデルを対象に重複の排除や適切な分割により精度が減少することを確かめる．重複の有無やデータセットの分割方法を変えた複数のパターンのデータセットでモデルの学習と評価を行い，その精度を比較することでそれぞれの要因によってモデルの精度がどの程度減少するかを調査する．

調査の結果，モデルの精度は重複を排除することで約 10%減少すること，分割方法を適切にすることで約 20%減少すること，重複と分割方法の両方を考慮した場合には約 30%減少

することを確認した．つまりモデル提案時の精度評価ではデータセットの扱い方が不適切であったために，モデルの精度が本来の精度よりも 50%近く上昇していたことがわかった．

主な用語

データセット
深層学習
汎化性能

目 次

1　はじめに

機械学習とは，大量のデータを学習することでその中から共通する規則やパターンを見つけ出し，未知のデータへ予測や推論を行う手法である [2]．ソースコード解析分野においても機械学習を用いたアプローチは急増しており，コードの補完やバグの修復，メソッドの命名など様々なタスクで利用されている．機械学習において，データセットの品質は機械学習モデルの性能に直結している．データセットの品質には，データの規模や多様性，データセットの前処理や分割など様々な要因が影響している．そのため，データ収集時やデータセット利用時にはこれらの要因を正しく理解して適切にデータセットを扱わなければ，モデルの精度を見誤ることになってしまう．

Allamanis[1] はデータセット内に重複，つまりファイルレベルで酷似したクローンが存在することで，モデルの精度が見かけ上で最大 100%上昇することを確認している．Maleki らの調査 [10] では医療分野の画像認識モデルにおいて，訓練セット・テストセット間で同一の患者のデータを含むように分割した場合，同一の患者のデータを含まない場合に比べて精度が高くなることがわかっている．これらの精度上昇は，データセット内の各データが他のデータと独立しておらず，関連性をもっていることが原因である．本研究ではこのような性質をデータセットの非独立性と呼ぶ．

Allamanis による調査では，GitHub[1]から収集したデータセットのみが調査対象であった．しかし，ソースコード解析タスクで利用されるデータセットの収集源は，GitHub 以外にも StackOverflow[2]や競技プログラミングサイトなどがある．特に競技プログラミングのデータセットは豊富なメタデータを持ち，様々な言語で書かれたソースコードを含むため，ソースコードの生成や翻訳，類似性判定等のタスクで利用されている．競技プログラミングのデータセットには，同一のユーザが提出したソースコードや同一の問題に提出されたソースコードが含まれている．また，同一のユーザが同一の問題に提出したソースコードも含まれており，これらのソースコードは基本的に編集量が少ないため，非常に酷似している．そのため，競技プログラミングのデータセットは非独立性な性質を持っているが，その影響については明らかになっていない．

そこで本研究では，競技プログラミングデータセットにおける非独立性の影響を調査する．特に，本調査ではコーディング能力判定モデルを対象として調査を行う．コーディング能力判定モデルは，松井 [11] によって提案されたモデルであり，ソースコードを入力することでそのソースコードを書いた開発者のコーディング能力を，上級者，中級者，初級者のいずれかで判定する．このモデルは競技プログラミングのソースコードとレーティングを学習に利

[1] https://github.com/
[2] https://stackoverflow.com/

用している．松井によるモデルの精度調査では，重複の除去やデータの関連性を意識した分割は行っていないため，報告されたモデルの精度は非独立性によって本来の性能以上に高くなっている可能性がある．このことを実証するために，本調査では重複の排除やデータの関連性を意識した分割によって非独立性を排除することで，モデルの精度が減少することを確かめた．その結果，重複を排除することでモデルの精度は約 10%減少し，データの関連性を意識した分割によってモデルの精度は約 20%減少した．また，重複と分割の両方を考慮して非独立性を排除することで，モデルの精度は約 30%減少した．つまり当初報告された精度は，データセットの非独立性によって本来の性能より約 50%上昇していたことがわかった．

以降 2 節では，本研究の背景として，機械学習モデルの精度上昇について説明する．3 節では調査対象となるモデルについて説明する．4 節では評価実験の内容を説明する．5 節では実験結果とその考察を述べる．6 節では本研究の妥当性の脅威について述べる．7 節では競技プログラミングのデータセットを用いた関連研究について紹介する．最後に 8 節ではまとめと今後の課題について述べる．

2　背景

本節では，データセットを用いて機械学習モデルを評価する際，その性能が高く評価されてしまう 2 つの原因，データの重複による影響とデータセットの分割方法による影響について述べる．

2.1　重複による影響

Allamanis により，データセット内の重複が機械学習モデルの精度を最大で 100%上昇させることがわかっている [1]．重複とは，ファイルレベルでほぼ酷似したソースコードのことを指す．ソフトウェア開発者は他のソースコードを部分的または全体的にコピーすることが多いため，このような重複が発生してしまう．Allamanis の調査では，github から収集した複数のメジャーなデータセットで重複が存在しており，それにより様々なタスクで精度の上昇が起きていることが報告されている．例えば，Java-Large データセット [3] は全体の 20.2%，Concode[14] データセットは 68.7%のファイルが重複したファイルになっており，Java-Large データセットを利用したメソッド命名タスクでは，重複を除くことで f 値が 12.3%減少することがわかっている．

2.1.1　重複の定義と種類

データセットがモデルの学習を行うための訓練セットと，モデルの精度を測定するためのテストセットに別れている時，以下の 3 種類の重複が発生する．

1. **訓練重複** 訓練セット内で重複しているファイル
2. **テスト重複** テストセット内で重複しているファイル
3. **交差重複** 訓練セットとテストセットの両方に現れるファイル

これを図 1 に示す．

重複による問題の核は，訓練セットとテストセットに同一のファイルが現れること，つまり交差重複の存在にある．機械学習モデルを学習させる目的は，モデルを用いて新しいコードや見たことのないコードに対して何らかの推測や見識を提供することである．そのためには，機械学習モデルが未知のソースコードによく汎化すること，つまりモデルのユースケースで観察されるようなデータの真の分布を忠実にモデル化することを必要とする．機械学習モデルが真のデータ分布に汎化するためには，その分布から独立に抽出されたデータで学習する必要がある．しかし重複の存在はそれに違反することとなり，モデルが汎化したように

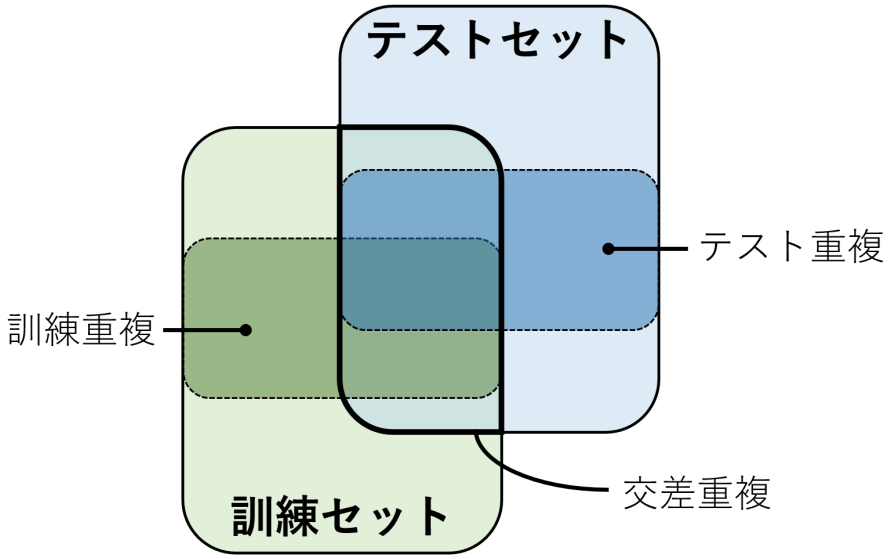

図 1: 重複の種類．[1] の図 1 を日本語に訳して転載.

見せかけて実際には重複を単に記憶しただけであるため，ユーザが本来観測するユースケース以上の精度となってしまう.

2.2 分割方法による影響

機械学習において，モデルの学習で用いる訓練データとモデルの評価で用いるテストデータは，一般的に 1 つのデータセットをランダムに分割したものが利用される．しかし，データセット中の各データが他のデータと共通の性質を持つ場合，それらが訓練データとテストデータに分割されることで精度の上昇につながってしまう．実際に，医療分野の画像診断モデルにおいて，同一の患者のデータが訓練データとテストデータの両方に含まれる場合と，どちらか一方のみに含まれる場合では，前者のほうが精度が約 40%高くなることが確認されている [10]．このように，訓練データとテストデータの両方に同一のグループのデータが含まれることを避けるようにデータセットを分割する手法は Group K-Fold と呼ばれ，訓練セットとテストセットの独立性を守ることができる.

2.3 データセットの非独立性

先述した 2 つの要因のように，データセット内の各データが独立ではなく，他のデータと共通の属性や特性を持っていることを，本研究ではデータセットの非独立性と呼ぶ．このような性質は，機械学習における重要な仮定に違反している．それは,「各データはすべて同じ確率分布から得られ，互いに独立である (i.i.d)」というものである [1]．これは不合理な仮定

ではなく，機械学習の研究と実践において広く使用されている．

3　調査対象モデル

本節では，今回実験の対象となるコーディング能力判定モデル，評価モデルを対象とした性能指標の 1 つである汎化性能と，これまで行った調査について述べる．

3.1　コーディング能力判定モデル

ソースコードからコーディング能力を判定する手法には，ソースコードのメトリクス等を利用する槇原の手法や [9]，ソースコードのトークン列から Long Short-Term Memory（LSTM）を用いて判定する手法 [6]，ソースコードのグラフ表現を利用する松井の手法などがある [11]．この中でも特に，松井の手法はソースコードの構文的・意味的情報を利用しているため，高い精度で判定を行うことができる．しかしこのモデルは，学習のための訓練セットと評価のためのテストセットで，同一のデータセットを分割したものを利用しており，異なるデータセットのデータに対する判定精度が不明であった．そのため，このモデルを対象とした汎化性能の調査をこれまで行ってきた．以降，本研究ではこのモデルを対象モデルと呼ぶ．

対象モデルは，ソースコードを入力として受け取り，「上級者」，「中級者」，「初級者」のいずれかの判定結果を出力する．このような判定を可能にするために，モデルの学習データとして競技プログラミングのソースコードとレーティングを利用している．競技プログラミングとは，与えられた要件を満たすプログラムを記述する速さや正確さを競う競技である．レーティングとは，競技プログラミングにおける熟練度をあらわす指標である．レーティングが高い上位 20%のソースコードを上級者，レーティングが低い下位 20%のソースコードを初級者，ちょうど中間の 20%のソースコードを中級者として定義する．このモデルの学習の流れを図 2 に示し，以下に各手順の概要を述べる．

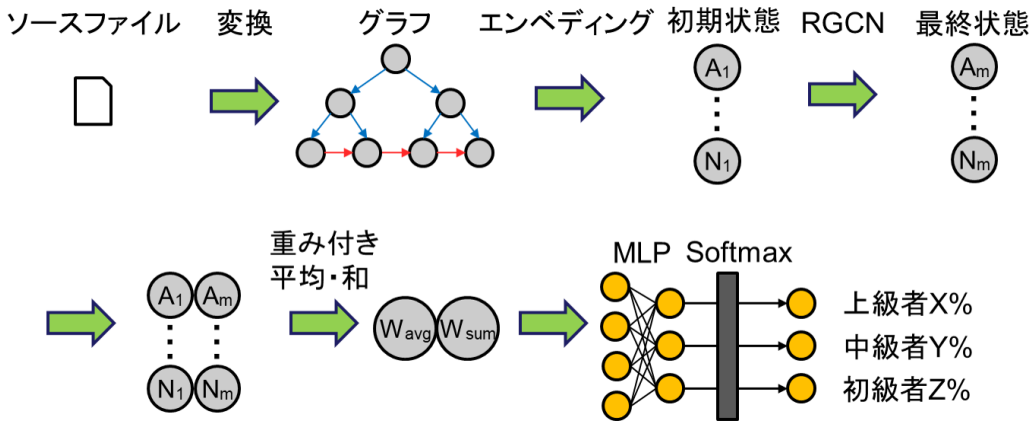

図 2: コーディング能力の判定手順

- **ソースコードをグラフに変換する．**構文木を構築し，各トークンの順序関係や同一変数の利用情報を表すエッジを追加する．
- **エンベディングにより初期状態を取得する．**各ノードを表す文字列に対して Character-level-CNN[18] を適用することでノードの初期状態を得る．
- **グラフの状態を更新し，最終状態を得る．**各ノードが隣接ノードから情報を集約してノードの状態を更新する，という工程を繰り返すことで最終状態が得られる．対象モデルでは，RGCN[13] というグラフネットワークを扱うためのディープラーニングアーキテクチャを採用している．
- **初期状態と最終状態から判定結果を出力する．**各ノードの初期状態と最終状態のベクトルを連結させた後，全ノードの重み付き和と重み付き平均を算出して連結させる．このベクトルを出力数 3 の MLP に入力し，得られる出力をさらに Softmax 層に入力することで，上級者・中級者・初級者である確率の合計値が 100%となるようにする．

3.2 モデルの汎化性能調査

汎化性能とは，機械学習モデルが未知のデータに対して正しく予測できる性能のことである．機械学習における学習とは，訓練セットに対する損失値ができるだけ小さくなるようにモデルのパラメータを設定することである．しかし，訓練セットに対して上手く予測が行えるようなパラメータを学習できたとしても，未知のデータに対して同様に正確に予測を行うことができるとは限らない．例えば，天気を予測する機械学習モデルが過去の天気を 100% 当てることができたとしても，未来の天気に対する予測が完璧にできるかどうかはわからない．そのため，汎化性能は機械学習モデルの信頼性や実用性を確認するための重要な指標となっている[3]．

筆者らは先行研究において，対象モデルの汎化性能を調査するために，出自の異なる 4 つのデータセットを利用し，相互にモデルの学習と評価を行ってそれぞれの精度を比較した [4]．その結果を図 3 に示す．

この結果から，学習と評価で同じデータセットを利用した場合の精度は高いが，異なるデータセットを利用した場合の精度は低くなる，すなわち対象モデルの汎化性能が低いことが確認できた．このように精度に大きく差が出た原因は，データセットの非独立性にあると考えられる．3.3 項で後述しているように，競技プログラミングデータセットは非独立な性質を持っている．しかしこの先行研究においては，学習と評価で同じデータセットを利用する際，データセット内に重複が含まれた状態で訓練セットとテストセットをランダムに分割

[3] https://www.varista.ai/knowledge/generalization-ability/

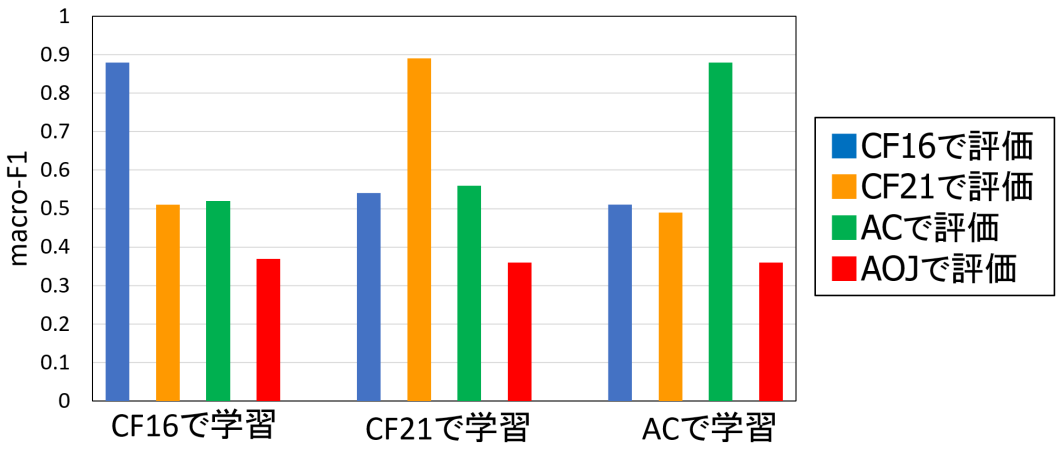

図 3: コーディング能力判定モデルの汎化性能

している．つまり訓練セットとテストセットが独立になっておらず，非独立性によってモデルの精度が高くなりすぎてしまったと考えられる．

3.3 競技プログラミングデータセットの非独立性

対象モデルの学習や評価で利用されているデータセットは競技プログラミングから収集したものであり，非独立な性質を持っている．それは，同一のユーザが提出したソースコードが多く存在していることと，同一の問題に提出されたソースコードが多く存在していることである．同一のユーザが提出したソースコードは，構文的特徴や識別子名などの特徴が似たものになっていると考えられる．同一の問題に提出されたソースコードは，使用されるアルゴリズムやデータ構造などの特徴が似たものになっていると考えられる．さらに，データセットには同一のユーザが同一の問題に提出したソースコードも複数存在している．これは，提出したソースコードが間違っていた場合に，ソースコードを修正して再提出をすることが可能になっているからである．このようなソースコードは修正量が少なく，酷似した状態にある．そこで本研究では，同一のユーザが同一の問題に提出したソースコードが複数存在した場合，それらを互いに「重複したソースコード」とみなす．重複したソースコードの例を図 4 に示す．この例では編集前後で 1 行のみの追加が行われているが，修正量が少なく酷似していることがわかる．

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string S; cin >> S;
    string ans;
    for(int i = S.size() - 1; i >= 0; --i){
        if(S[i] != '.') ans = S[i] + ans;
    }
cout << ans << endl;
}
```

(a) 編集前の不正解のソースコード

```cpp
#include<iostream>
#include<string>
using namespace std;
int main(){
    string S; cin >> S;
    string ans;
    for(int i = S.size() - 1; i >= 0; --i){
        if(S[i] != '.') ans = S[i] + ans;
        else break;
    }
cout << ans << endl;
}
```

(b) 編集後の正解のソースコード

図 4: 重複したソースコードの例

4　調査内容

本研究では，前節で説明した競技プログラミングデータセットの非独立性による精度への影響を，対象モデルを用いて実証的に調査する．そのために，データセット内の重複の有無や訓練セット・テストセットの分割方法を変えて精度を測定・比較する．

4.1　使用するデータセット

本研究では，CF16，CF21，AC の 3 つのデータセットを使用する．各データセットの概要を表 1 に示す．

表 1: 使用する各データセットの概要

データセット名	CF16	CF21	AC
収集元ドメイン	Codeforces	Codeforces	AtCoder
ファイル数	1,644,636	1,752,427	1,459,964
収集期間	2016/5/19〜2016/11/15	2021/12/1〜2021/12/31	2021/1/1〜2021/6/1

CF16 と CF21 は，ロシア最大級の競技プログラミングサイトである Codeforces[4]から収集したデータセットであるが，それぞれ収集期間が異なっている．一方，AC は日本最大級の競技プログラミングサイトである AtCoder[5]から収集したデータセットである．CF16 は堤 [16] によって作成されたデータセットであり，対象モデル提案時の学習や評価で使用されたものである．CF21 と AC は筆者らが汎化性能調査のために先行研究にて作成したデータセットである．

これらのデータセットの重複度合いを表 2〜4 に示す．重複度とは，同一のユーザが同一の問題に提出したファイル数を表す．各データセットの総ファイル数は表 1 より少なくなっているが，これは対象モデルの学習と評価で利用する上級者・中級者・初級者のファイルのみを対象としているためである．

これらの表から，どのデータセットにおいても重複が多数存在していることがわかる．各データセットに注目すると，AC は重複度 1 のファイルが 58%を占めており，重複しているソースコードはデータセット全体の半分以下である．一方，CF16 と CF21 は重複度が 2 以上のファイルの割合がともに 70%を超えており，非常に多くの重複ファイルが存在している．さらに，CF16 と CF21 は重複度が 5 以上の割合がともに 30%前後あり，非常に深刻な重複の問題を抱えている．

[4] https://codeforces.com/
[5] https://atcoder.jp/

表 2: CF16 の重複統計

重複度	1	2	3	4	5 以上
グループ数	193,667	69,905	34,222	18,996	31,130
ファイル数	193,667	139,810	102,666	75,984	231,092
割合	26%	19%	14%	10%	31%

表 3: CF21 の重複統計

重複度	1	2	3	4	5 以上
グループ数	244,051	80,458	38,224	20,642	32,431
ファイル数	244,051	160,916	114,672	82,568	232,895
割合	29%	19%	14%	10%	28%

表 4: AC の重複統計

重複度	1	2	3	4	5 以上
グループ数	344,856	51,694	17,188	7,487	8,880
ファイル数	344,856	103,388	51,564	29,488	59,837
割合	58%	18%	9%	5%	10%

4.2 評価指標

本研究では，評価指標として macro-F1 を使用する．macro-F1 は各クラスにおける F 値の平均をとったものであり，本研究では以下の式で求めることができる．

$$macro\text{-}F1 = \frac{F1_{high} + F1_{mid} + F1_{low}}{3}$$

ここで，式中の $F1_{high}$ は上級者の F 値，$F1_{mid}$ は中級者の F 値，$F1_{low}$ は初級者の F 値を表している．

4.3 重複と分割方法の組み合わせ

ここでは，重複や分割方法に関する用語について述べる．まず，重複に関するテストの用語を定める．これらの用語は，Allamanis の用語定義を参考にして定義したものである．

- **完全非重複テスト** 訓練セットとテストセットの両方から重複を取り除いて行うテスト．ここで,「重複を取り除く」とは，同一のユーザが同一の問題に提出したソースコードのうち，最新のもののみを残して残りをすべて除去することである．
- **訓練重複テスト** 訓練セットに重複があり， テストセットからは重複を取り除いて行うテスト．
- **交差重複テスト** 訓練セットの重複と交差重複を含み，テスト内の重複は含まないテストセットを使用して行うテスト．
- **完全重複テスト** すべての重複を含む状態で行うテスト．つまり，データセットに全く加工をしていない元のままで学習やテストが行われる．重複を意識せずに行われるテストは完全重複テストとなる．

次にデータセットの分割方法に関する用語を紹介する．本研究では 3 種類の分割方法を比較する．

- **ランダム分割** データセット中のデータをランダムに訓練・検証・テストセットに分割する方法．ここで，分割比率はいずれの手法においても訓練: 検証: テスト=8:1:1 になるように分割する．
- **ユーザ分割** 同一のユーザが提出したソースコードが，訓練・検証・テストセットのいずれかの集合にのみ含まれるように分割する手法．つまり，同一のユーザのソースコードが訓練セットとテストセットの両方に現れることはない．

- **問題分割** 同一の問題に提出されたソースコードが，訓練・検証・テストセットのいずれかの集合にのみ含まれるように分割する手法.

これらの重複と分割方法の組み合わせを図 5 に示す.

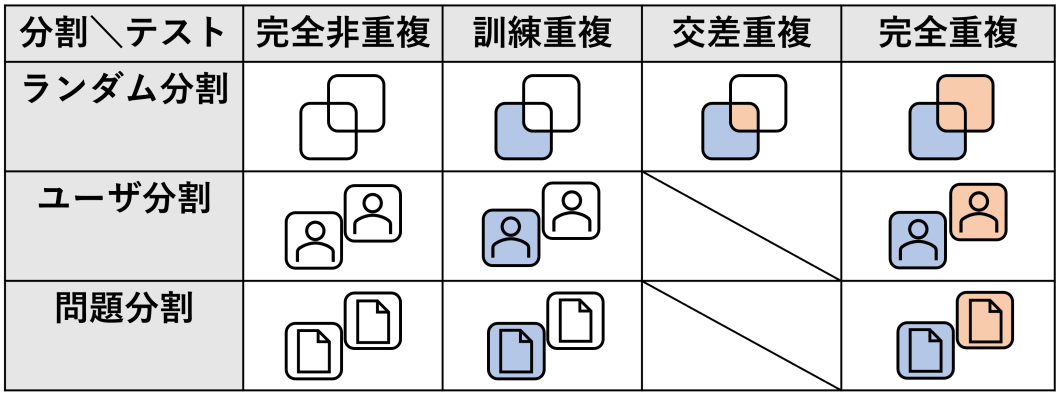

図 5: 重複と分割方法の組み合わせ

図中の各記号は訓練セットとテストセットのベン図として解釈できる．左下の四角形が訓練セットの集合，右上の四角形がテストセットの集合を表しており，網掛けは重複を含むことを，空白は重複が排除されていることを意味している．また，各集合内の人型記号や紙型記号はそれぞれ，ユーザ分割と問題分割を意味している．本研究では，この図で示した各パターンでの対象モデルの精度を測定し比較することで，データセットの非独立性に起因する各要因の影響を調査する．

5　調査結果

各データセットを対象モデルの学習・評価で利用した際の各条件下でのモデルの精度を表5〜表7に示す．

表 5: CF16における各条件下での精度．表中の値はmacro-F1．

分割＼重複	完全非重複	訓練重複	交差重複	完全重複
ランダム分割	0.75	0.76	0.87	0.88
ユーザ分割	0.59	0.59	-	0.61
問題分割	0.73	0.73	-	0.74

表 6: CF21における各条件下での精度．表中の値はmacro-F1．

分割＼重複	完全非重複	訓練重複	交差重複	完全重複
ランダム分割	0.82	0.82	0.88	0.88
ユーザ分割	0.66	0.66	-	0.67
問題分割	0.82	0.82	-	0.82

表 7: ACにおける各条件下での精度．表中の値はmacro-F1．

分割＼重複	完全非重複	訓練重複	交差重複	完全重複
ランダム分割	0.82	0.82	0.87	0.87
ユーザ分割	0.61	0.62	-	0.62
問題分割	0.82	0.82	-	0.83

完全非重複テスト（）の結果と，訓練重複テスト（）の結果から，訓練セットにおける重複の有無はモデルの精度にはあまり影響を与えていないことがわかる．また，交差重複テスト（）と完全重複テスト（）の結果から，テストセットにおける重複の有無もモデルの精度にあまり影響を与えていないことがわかる．一方，訓練重複テスト（）と交差重複テスト（）の結果から，交差重複の存在によってモデルの精度が高くなっていることがわかる．また，分割方法に着目すると，ランダム分割はユーザ分割や問題分割よりも精度が高くなっていることがわかる．

5.1　交差重複の影響

各データセットにおける交差重複によるモデルの精度の変化率を表8に示す．

表 8: 交差重複による精度の変化率．表中の値は macro-F1.

データセット			Δ ( , )
CF16	0.87	0.76	**-12.6%**
CF21	0.88	0.82	**-6.8%**
AC	0.87	0.82	**-5.7%**

この結果から，いずれのデータセットにおいても重複によってモデルの精度が高くなっており，重複を排除することでその精度が低下することが確認できた．変化の一番小さかった AC においても精度が 5%以上低くなっており，重複による影響が顕著であることがわかる．特に CF16 は精度が最も低くなっており，交差重複を排除することによってモデルの精度が 10%以上も低くなっている．CF21 は重複統計において CF16 と同程度の重複割合であったが，精度の減少率は CF16 の半分程度であり，重複率と重複の排除による精度の低下は必ずしも比例するとは限らないことがわかる．

5.2 ユーザ分割の影響

各データセットにおけるユーザ分割によるモデルの精度の変化率を表 9 に示す．

表 9: ユーザ分割による精度の変化率．表中の値は macro-F1.

データセット			Δ ( , )
CF16	0.76	0.59	**-22.4%**
CF21	0.82	0.66	**-19.5%**
AC	0.82	0.62	**-24.4%**

いずれのデータセットにおいても，ユーザ分割をすることでランダム分割より精度が約 20%程度減少している．つまり，同一のユーザが提出したソースコードが訓練セットとテストセットの両方に存在していたことで，モデルの精度が高くなっていたことがわかる．このことから，対象モデルは未知のユーザが書いたソースコードに対しては，コーディング能力を正確に判定しにくいといえる．

5.3 問題分割の影響

各データセットにおける問題分割によるモデルの精度の変化率を表 10 に示す．

CF16 では問題分割によって精度の低下が起きているが，CF21 と AC では問題分割による精度の低下は起きていない．この表の値は macro-F1 の小数第 3 位を四捨五入した値になっ

表 10: 問題分割による精度の変化率．表中の値は macro-F1.

データセット			Δ ( , )
CF16	0.76	0.73	**-3.9%**
CF21	0.82	0.82	**0.0%**
AC	0.82	0.82	**0.0%**

ており，実際にはわずかに精度の変化はあったものの，ほぼ誤差とみなすことができる程度のものであった．この結果から CF21 と AC で学習したモデルは，未知の問題に対して提出されたソースコードに対しても正確にコーディング能力を判定できるといえる．

5.4 結果の考察

本研究の結果から，以下の 3 つのことがわかった．

1. 競技プログラミングのデータセットにおいても，交差重複の存在によってモデルの精度が上がること
2. 同一のユーザが提出したソースコードが訓練・テストセットの両方に存在することで，モデルの精度が大幅に上がること
3. 同一の問題に提出されたソースコードが訓練・テストセットの両方に存在することによる影響は，データセットによって異なること

これらの結果から，対象モデルの性能はデータセットの非独立性によって高くなっていることがわかった．特に重複とユーザ分割による影響が大きいことが確認できた．重複とユーザ分割の両方を考慮した精度の変化率を表 11 に示す．

表 11: 重複とユーザ分割の両方による精度の変化率．表中の値は macro-F1.

データセット			Δ ( , )
CF16	0.87	0.59	**-32.2%**
CF21	0.88	0.66	**-25.0%**
AC	0.87	0.62	**-28.7%**

この表から，重複を除いてデータセットを適切に分割することで，モデルの精度が約 30% 低下することが確認できた．逆に言えば，先行研究で報告されていた対象モデルの精度は，本来の性能よりも 50%近く高くなっていたことになる．このようなモデルの精度の変化は，他のモデルやデータセットでも発生している可能性がある．

(1) の結果については，競技プログラミングのデータセットを使用する多くのタスクに適用されると考えられる．どのようなモデルであっても，訓練セットと重複した（同一のユーザが同一の問題に提出した）ソースコードが実際のユースケースにおいて入力されることはほとんどないと考えられる．交差重複の存在は，モデルの汎化性能ではなくモデルの記憶力を評価することにつながってしまう．そのため，モデルの真の性能を確かめるうえで交差重複を除くことは重要である．

(2) の結果については，コーディング能力判定という今回の調査で対象としたタスクに固有のものであると考えられる．本研究では，コーディング能力の基準として競技プログラミングのレーティングを使用しており，レーティングはソースコードに依存するものではなく，そのソースコードを書いたユーザに依存するものである．そのため，同じユーザが書いたソースコードを訓練セット内で学習しているか否かはモデルの精度に大きく影響を与えると考えられる．他のタスクにおいては，ソースコードを書くユーザに依存するような指標を予測したい場合には，ユーザ分割は非常に有効であると考えられる．

(3) の結果については，対象とするタスクやデータセットによって変化すると考えられる．コーディング能力の判定においては，同一のアルゴリズムやデータ構造を学習しているかどうかはモデルの精度にあまり影響を与えないことがわかった．しかし，カテゴリ分類のようにソースコードの機能を学習させるようなタスクにおいては，同じ機能をもつソースコードを学習しているかどうかという点は非常に重要である．そのため，問題分割によって同一の問題に提出されたソースコードが訓練・テストセットの両方に出現しないようにすることで，そのモデルの性能を正確に判断することができる．

6　妥当性の脅威

本研究では，モデルの学習や評価において交差検証を行っていない．交差検証とは，データセット中のデータを k 個に分割してそのうちの 1 つをテストセットに，残りの k-1 個を訓練セットとしてモデルの学習と評価を行う．そして，これを k 個のデータすべてが 1 回ずつテストセットとなるように k 回学習と評価を行い，それらの精度の平均を取る手法である．本研究では，重複の有無や分割方法による複数の組合せパターンでの学習を 3 つのデータセットそれぞれで行っているため，モデルの学習に非常に時間がかかってしまい，1 つの組み合わせパターンに対して複数回の学習が必要となる交差検証を行うことができなかった．そのため，1 つの組み合わせパターンに対しては，訓練・テストセットは 1 通りの分割でしか精度を測定できていない．訓練・テストセットを同じ条件で異なる分割にした場合，本研究で紹介した結果とは精度が異なる可能性がありえる．

7　関連研究

競技プログラミングから収集したデータセットは，本研究で使用したもの以外にも多くあり，様々なタスクで利用されている．その代表的な例を以下に示す．

- **GoogleCodeJam データセット [19]** GoogleCodeJam[6]は，Google により毎年開催される世界最大規模の競技プログラミングのコンテストである．このデータセットは，Zhao らによって作成されたもので，12 の問題から 1,669 のソースコードを集めている．Zhao らはこのデータセットを利用して，ソースコードの類似性判定を行う深層学習モデルを作成した．

- **POJ-104[8]** POJ-104 は教育的なオンラインジャッジシステムから収集したデータセットで，104 個の問題に対してそれぞれ 500 個の解答ソースコードを含んでおり，計 52,000 個のソースコードから構成されている．このデータセットは，コード分類 [15] やコードの類似性判定 [17] の研究に利用されている．

- **APPS[5]** APPS データセットは，Codeforces や Kattis[7]など，様々な競技プログラミングサイトから収集した 10,000 の問題と 232,421 個のソースコードで構成されている．このデータセットは，GPT 等の大規模言語モデルにおけるソースコード生成の正確性を評価ことを目的として，Hendrycks らによって作成された．そのため，各問題に対して解答の正誤を判定するためのテストケースも含まれており，生成されたプログラムの正確性を厳密に評価することができる．

- **CodeNet[12]** CodeNet は IBM によって作られたデータセットであり，日本の競技プログラミングサイトである AtCoder と Aizu Online Judge[8]からソースコードを収集している．このデータセットは 4,053 個の問題から 13,916,868 個のソースコードを集めており，非常に大規模なデータセットになっている．CodeNet は Li らによって作成されたコード生成モデルである AlphaCode[7] の学習データの一部として利用されている．

これらのデータセットは，いずれも同一のユーザによって提出されたソースコードや，同一の問題に提出されたソースコードを含んでいる．また，同一のユーザによって同一の問題に提出されたソースコードも含んでいる．つまり非独立な性質を持っているため，これらのデータセットを利用する際には対象となるタスクに合わせて，適切に非独立性を排除する必

[6] https://codingcompetitions.withgoogle.com/codejam
[7] https://open.kattis.com/
[8] https://judge.u-aizu.ac.jp/onlinejudge/

要があると考えられる．特に CodeNet は他のデータセットと比較して非常に規模が大きく，非独立性による影響も大きくなると考えられる．

8　おわりに

本研究では，競技プログラミングデータセットの非独立性がコーディング能力判定モデルの精度に与える影響を調査した．その結果，非独立性によってモデルの精度が過剰に高くなっており，非独立性を排除することでモデルの精度が約 30%減少することが確認できた．非独立性の中でも，特に同一のユーザが提出したソースコードが訓練・テストセットの両方に含まれることで，モデルの精度が大きく上がることがわかった．

今後の課題は，他のデータセットやモデルにおけるデータセットの非独立性の影響を調べることである．競技プログラミングから収集したデータセットは，7 節で紹介したもの以外にも多く存在しており，様々な研究で利用されている．それらのデータセットやタスクにおいて非独立性の影響を調査することで，本研究で得られた結果の一般性が明らか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 肥後芳樹 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．

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肥後研究室の皆様には，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] Miltiadis Allamanis. The adverse effects of code duplication in machine learning models of code. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 143–153, 2019.

[2] Miltiadis Allamanis, Earl T. Barr, Premkumar Devanbu, and Charles Sutton. A survey of machine learning for big code and naturalness. *ACM Comput. Surv.*, Vol. 51, No. 4, jul 2018.

[3] Uri Alon, Shaked Brody, Omer Levy, and Eran Yahav. code2seq: Generating sequences from structured representations of code. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[4] 服部文志, 松下誠, 井上克郎. 深層学習を用いたコーディング能力判定モデルの汎化性能調査. 情報処理学会研究報告, Vol. 2022-SE-211, No. 15, pp. 1–7, 2022.

[5] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge competence with APPS. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, 1997.

[7] Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. Competition-level code generation with alphacode. *Science*, Vol. 378, No. 6624, pp. 1092–1097, 2022.

[8] Lu Zhang Tao Wang Lili Mou, Ge Li and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 1287–1293, 2016.

[9] 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.

[10] Farhad Maleki, Katie Ovens, Rajiv Gupta, Caroline Reinhold, Alan Spatz, and Reza Forghani. Generalizability of machine learning models: quantitative evaluation of three methodological pitfalls. *Radiology: Artificial Intelligence*, Vol. 5, No. 1, p. e220028, 2022.

[11] 松井智寛, 松下誠, 井上克郎. ソースコードのグラフ表現を利用した深層学習によるコーディングの専門性の判定手法. 情報処理学会研究報告, Vol. 2022-SE-210, No. 12, pp. 1–8, 2022.

[12] Ruchir Puri, David S Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladimir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, et al. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.

[13] Michael Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne vanÂ den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *European Semantic Web Conference*, pp. 593–607. Springer International Publishing, 2018.

[14] Alvin Cheung Srinivasan Iyer, Ioannis Konstas and Luke Zettlemoyer. Mapping language to code in programmatic context. In *In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1643–1652, 2018.

[15] Alice Shoshana Jakobovits Tal Ben-Nun and Torsten Hoefler. Neural code comprehension: A learnable representation of code semantics. In *Advances in Neural Information Processing Systems 31*, pp. 3588–3600, 2018.

[16] 堤祥吾. プログラミングコンテスト初級者・上級者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.

[17] Fangke Ye, Shengtian Zhou, Anand Venkat, Ryan Marcus, Nesime Tatbul, Jesmin Jahan Tithi, Paul Petersen, Timothy G. Mattson, Tim Kraska, Pradeep Dubey, Vivek Sarkar, and Justin Gottschlich. MISIM: an end-to-end neural code similarity system. *CoRR*, Vol. abs/2006.05265, , 2020.

[18] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems*, Vol. 1,

pp. 649–657, 2015.

[19] G. Zhao and J.Huang. Deepsim: Deep learning code functional similarity. In *Proc. FSE 2018*, pp. 141–151, 2018.