Vol.2021-SE-207 No.12
2021/3/1

深層学習を用いたコードクローン検出器の汎化性能に関する調査

福家 範浩[1]　藤原 裕士[1]　吉田 則裕[2]　崔 恩瀞[3]　井上 克郎[1]

概要：ソフトウェア保守において問題となる要因の 1 つとしてコードクローンがある．字句の違いだけではなく構文的や意味的な違いも含むコードクローンを検出するために，近年，深層学習を用いた検出器が提案されている．深層学習を用いた研究では，検出器の学習と精度計測のために 1 つのデータセットの中から学習データと計測データを取得していることが多い．しかし，実開発環境でコードクローン検出器を用いる場合，学習データセットに含まれていないコードクローンを検出する可能性があるため，既存研究で報告されている精度より低い精度で検出器がコードクローンを検出する可能性がある．そのため，学習データと計測データに異なるデータセットを使った精度計測も必要だと考えられるが，既存研究ではその精度計測は行われていない．そこで，本研究では，コードクローン検出器の汎化性能を明らかにするために，学習データと計測データに異なるデータセットを使う場合のコードクローン検出器の精度を調査した．その結果，学習データと計測データが異なる場合，既存論文で報告されているより精度が低いことを確認した．

Investigating the Generalization Performance of Code Clone Detectors Using Deep Learning

Norihiro FUKE[1]　Yuji FUJIWARA[1]　Norihiro YOSHIDA[2]　Eunjong CHOI[3]　Katsuro INOUE[1]

1. まえがき

コードクローンとは，ソースコード中に存在する同一，または類似した部分を持つコード片を示し，主に開発者が効率よく開発を行うために既存のコード片をコピーアンドペーストすることで生成される．一般的にコードクローンは，ソフトウェアの保守を困難にする要因の 1 つとして挙げられている．あるコード片にバグが存在する場合，一致または類似するコード片にも同様のバグが含まれている可能性が高い．そのため，開発者はバグを見つけた場合，見つけたコード片に対してだけではなく一致または類似するコード片に対してもバグの修正を検討をする必要がある．そのため，ソフトウェアの保守を行う際は，コードクローンがどこに存在するかを把握し，管理する必要がある．しかし，大規模開発となると非常に多くのコード片の組み合わせが存在し，開発者が全てのコードクローンについて手作業で探して管理することは現実的ではない．そこで開発者はコードクローンを管理するためにコードクローン検出器を利用する [1–4]．

コードクローン検出器には，CCFinder [5] や CCFinder の後継機である CCFinderX[*1] などが挙げられる．これらの検出器は検出の前処理として字句解析を行って，字句単位でのコードクローンの検出を行うことが出来る．しかし，ソースコード中で文の挿入や削除など大きな構造の違いがあるコードクローンや，異なる実装で処理が同じであるコードクローンを検出することはこれらの検出器では困

[1] 大阪大学
Osaka University
[2] 名古屋大学
Nagoya University
[3] 京都工芸繊維大学
Kyoto Institute of Technology

[*1] http://www.ccfinder.net/

難である．そこで最近では従来の検出器では検出が困難なコードクローンを検出するために，コード片の特徴量同士から深層学習を用いてコードクローンであるかどうかを分類する CCLearner [6] や ASTNN [7] 等の検出器が提案されている．これら深層学習を用いた検出器では，事前にコードクローンのデータセットを用いて深層学習モデルの学習を行う．実際に開発者がコードクローン検出器を利用する際は，事前に学習したモデルを用いて，学習に用いたデータセットにないソースコードに対して検出を行う．しかし，これらの検出器の性能の調査では 1 つのデータセットの中から学習と検証のデータを取得しているため，現実的な利用に則した汎化性能を調べることが出来ていない．

そこで本稿では，深層学習を用いたコードクローン検出器である CCLearner と ASTNN に，学習と検証で別のデータセットを用いて検出器の汎化性能の調査を行った．

以降，2 章では，本調査に関わるコードクローン検出器の関連研究や精度調査のためのコードクローンデータセットに関して説明する．3 章では，汎化性能を調べるための調査方法について説明を行う．4 章では，本調査で行った実験の結果について説明する．5 章では，まとめと今後の展望について説明する．

2. 背景

2.1 コードクローン

コードクローンとは，ソースコード中の一致または類似するコード片を指す．コードクローンは違いによって大きく 4 つのタイプに分類される．

タイプ 1（以降 T1） 空白やタブ・改行・コメント以外は一致

タイプ 2（以降 T2） タイプ 1 に加え，変数名や関数名などユーザ定義・変数の型の違い

タイプ 3（以降 T3） タイプ 2 に加え，文の挿入や削除変更などの違い

タイプ 4（以降 T4） タイプ 3 に加え，同一の処理を行うが構文上の実装の違い

T3 コードクローンは構文的な類似度によって，90%以上は Very Strong Type 3（以降 VST3），70%から 90%は Strongly Type 3（以降 ST3），50%から 70%は Moderately Type 3（以降 MT3），50%未満は Weakly Type 3（以降 WT3）のようにさらに細分化される．

2.2 コードクローン検出器

CCFinder 等の従来のルールベースでの検出器の場合，T1 と T2 のコードクローンを検出することは出来るが，T3 と T4 のコードクローンに対する検出精度が低いことが知られている．そこで近年では，コード片の特徴量からコードクローンか否かの分類に深層学習を取り入れた検出器が提案されている．

深層学習を用いたコードクローン検出器に CCLearner [6] と ASTNN [7] がある．CCLearner はトークンベースでの検出器であり，コード片を分割して得たトークンをあらかじめ決められた 8 つのカテゴリに分類し，それらの頻度をコード片の特徴量として深層学習モデルに入力することで，入力したコード片対はコードクローン対か否かを分類する．CCLearner では，T1 から ST3 のコードクローンの検出精度が従来の検出器よりも高くなった．ASTNN は AST（抽象構文木）の構造をベースにした検出器であり，コード片から生成した AST と Word2Vec を用いてベクトル化したノードを用いてコード片の特徴ベクトルを求め，それらを深層学習モデルに入力することで，入力したコード片対はコードクローン対か否かを分類する．

深層学習を用いたコードクローン検出器は，使用する前に検出器に使われている深層学習モデルの学習を行う．検出器の精度を調べるためには，モデルの学習を行う用とモデルの学習を見る検証用とモデルの最終的な精度計測用のデータが必要となる．CCLearner では 1 つのデータセットをソースコードのフォルダ単位で学習と精度計測用のデータに分けている．ASTNN では 1 つのデータセットの 6 割を学習，2 割を検証，2 割を精度計測用に分けている．

2.3 検出精度の評価のためのデータセット

コードクローン検出器を評価するためには，事前にコードクローンの分類などをタグ付けしたデータセットを用いる．

最も用いられるコードクローン検出器評価用のデータセットの 1 つに BigCloneBench [8]（以降 BCB）が挙げられる．BCB は Svajlenko らによって作成された Java ソースコードの大規模データセットで，プロジェクト間リポジトリの大規模データである IJaDataset 2.0 [9] にあるソースコードをもとに作成されたものである．IJaDataset 2.0 の中から特定の機能のソースコードをマイニングし，それらに手動でクローンと非クローンのラベル付けをすることでコードクローン検出器のベンチマークとした．BCB には，T1・T2・T3・T4 のタイプ分類のタグやコード片同士の類似度もつけられており，クローンタイプごとの検出精度も求めることが出来るようになっている．

Google Code Jam に提出されたソースコードを利用してコードクローン検出器を評価する研究が行われている [10–12]．Zhao と Huang は, DeepSim の評価用に Google Code Jam に提出されたデータセットを公開している [12]．このデータセットを以降，GCJ と呼ぶ．このデータセットは，同じ設問に対して提出された解答ソースコードは，実装の上では異なるが行う処理は同じである T4 のコードクローンという仮定のもとに作成されたものである．12 個の設問から 1669 個のプロジェクトを取得し，内訳はコードクローン対は約 275,000，非コードクローン対は約 1,116,000

となっている．

2.4 汎化性能

深層学習では，特定のデータに対してのみ精度が良いのではなく，どのデータに対しても精度が公平に良いものが優れているとされている．例えば顔認識の問題では，人種間での検出精度に差があり，その差を埋める公平に検出を行うための研究がされている [13]．コードクローン検出器に対しても同様のことが考えられ，特定の機能のソースコードに対して性能が良いものよりも，どのソースコードに対しても性能が良いものが優れていると言える．本調査では，特定のソースコードにのみ性能が良いのではなく，どのソースコードに対して性能が良いことを汎化性能が高いとする．

2.5 既存研究の問題点

CCLearner や ASTNN 等の既存研究のように学習と計測に同じデータセットを用いた場合，データセットに偏りがあると学習済みのクローン検出器でどのソフトウェアに対しても同じ精度で検出することが出来ない可能性が出てくる．実際，BCB は IJaDataset 2.0 から特定の機能を持つように抽出されたデータセットで，世の中に存在しているソースコードの中では偏っている可能性が高い．特定のコードクローンのみ検出出来る可能性を捨てるためには，別のデータセットに対しての検出精度を調べる必要がある．しかし，CCLearner や ASTNN 等の既存研究ではこの調査が行われていない．

3. 調査の手順

この章では，深層学習を用いたコードクローン検出器である CCLearner と ASTNN の汎化性能の調査手順について説明する．深層学習を用いたコードクローン検出器の最終的な目標は，事前に検出器の学習を行って未知のソースコードに対して適用することである．そのためには，学習済みのコードクローン検出器がどのデータセットに対しても同じ精度が出せるのかどうかが検出器の信頼性につながってくる．そのため，本調査では，現実的な検出器の利用をするための検出器の信頼性の指標の 1 つとして利用されるように，同一のデータセットを使った場合と別のデータセットを使った場合の精度を検証し汎化性能の調査を行う．

3.1 汎化性能調査に用いたデータセット

3.1.1 BCB （BigCloneBench）

2.3 節で説明した BCB は論文で公開されてから何度かデータセットが追加されており，どの検出器も発表時の最新のデータセットを用いて学習と検出器の精度検証を行っている．本調査では，報告された検出器を用いてどのデータセットに対しても検出器は有効かどうかを調べるため，検出器の学習には報告された当時のデータセットを使用することにした．ASTNN は CCLearner よりもあとに提案された検出器のため，BCB 全体のデータ数は CCLearner の時よりも増加している．**表 1** と **表 2** は，CCLearner と ASTNN に用いた BCB のデータの内訳である．

表 1 CCLearner の学習に用いたクローンタイプの内訳 [6]

クローンタイプ	学習データ数	計測データ数
T1	13,802	2,383
T2	3,116	671
VST3	1,210	873
ST3	4,666	5,365
MT3	0	31,413
WT3/T4	0	1,540,513
非コードクローン対	22,794	-

表 2 ASTNN の学習に用いたクローンタイプの内訳

クローンタイプ	学習データ数	計測データ数
T1	1,2092	3,164
T2	9,281	706
VST3/ST3	10,973	3,651
MT3	11,978	4,017
WT3/T4	11,949	3,974
非コードクローン対	12,092	3,994

CCLearner の論文中に，CCLearner はトークンベースの検出器であるため MT3 や WT3/T4 のコードクローンを学習データに用いると検出精度が下がるという報告 [6] があったため構文的な類似度が MT3 以下のコードクローンは本調査でも学習データに含めていない．ASTNN では比較的数の少ない T2 等も検出器に反映されるように，学習と検証と計測に用いるコードクローンの多いものを最大 20,000 に制限して学習を行った．

検出器の評価指標の 1 つである precision は一般的に非コードクローン対の数を使用して求めるが，CCLearner では違った方法で精度を評価しているため非コードクローン対の計測数はハイフンとしている．CCLearner での評価方法は，3.3.1 で説明する．

3.1.2 GCJ （Google Code Jam データセット）

2.3 節で説明した GCJ は，Zhao と Huang がコードクローン検出器の精度評価のために作成したデータセットである．このデータセットは，オープンジャッジシステムの 1 つである Google Code Jam の中から 12 の設問に対する解答ソースコードを取得して作られた．設問ごとの解答数は，**表 3** のようにばらつきがあり，多いものは 477 個，少ないものは 2 個となっている．12 の設問全ての解答ソースコード数は 1663 個，コードクローン対の数は 274,048 個である．非コードクローン対に関しては，別の設問に対する解答ソースコードのコード片同士は非コードクローン対となるため，非常に多くのデータ数となる．しかし，今回

表 3 Google Code Jam データセットの設問ごと解答コード数

設問	解答コード数
1	477
2	88
3	242
4	38
5	2
6	434
7	27
8	245
9	68
10	18
11	20
12	4
計	1663

の調査では，非コードクローン対の数をコードクローン対と同じ 274,048 個ランダムに選び，精度の検証を行った．なお，CCLearner と ASTNN ともに同じデータセットを用いて検出器の精度の検証を行った．

3.2 評価指標

本調査では，GCJ での評価の指標に precision，recall，f1，AUC を用いた．本調査での，true positive や true negative，false positive，false negative について以下で説明する．

true positive コードクローンのうち，検出されたコード片

true negative 非コードクローンのうち，検出されなかったコード片

false positive 非コードクローンのうち，誤って検出されたコード片

false negative コードクローンのうち，検出されなかったコード片

precision（適合率）は，true positive と false positive のうち，true positive の割合である．recall（再現率）は，true positive と false negative のうち，true positive の割合である．precision または recall のどちらかだけが極端に数値が高い場合，ほとんどのコード片対にたいしコードクローンまたは非コードクローンと判断している可能性がある．それらを解消するための指標の 1 つに f 値があり，以下の式 (1) で表される precision と recall の調和平均は f1 と呼ばれる指標である．調和平均では，どちらかの値だけが極端に大きい場合にも低い結果となる．

$$f1 = 2\frac{precision \cdot recall}{precision + recall} \quad (1)$$

検出器の性能の比較を行うには，主にこの指標を用いることが多い．

AUC とは，ROC 曲線と呼ばれる曲線と軸との間に出来た面積の値である．ROC 曲線とは，false positive rate を横軸にとり，true positive rate を縦軸にとり，閾値を変化させながらプロットしていくことで出来る曲線のことである．曲線と軸との間の面積が大きく 1 に近い場合，その検出器は分解能が高いものであると言える．本調査では，AUC の値を precision と recall の値の変化と閾値の関係を見るために使用した．

3.3 学習と汎化性能の計測手順

本調査で行った実験の手順を**図 1** に示す．

3.3.1 STEP1: BigCloneBench での学習

CCLearner と ASTNN ともに論文と同様のデータセットを用いて学習を行った．CCLearner では，BCB の中で最もソースファイルの多いフォルダ内のデータを学習用データにし，ASTNN では抽出済みの BCB のデータをランダムに分割して学習用データにし，検出器の学習を行った．

3.3.2 STEP2: BigCloneBench での計測

CCLearner の評価は検出したコードクローン対を, 実際に人が確認を行って精度の評価を行っている．論文と全く同じ環境で評価することが出来ないため, 本調査では CCLearner については BCB での計測結果は，CCLearner の論文で報告されている値を使用することにした．CCLearner の論文での評価の方法の詳細を説明する．recall の値は他の検出器の評価方法と同様で, 全ての既知のコードクローンのうち検出出来ているコードクローンの割合である．precision の値は他の検出器の評価方法とは異なっている．CCLearner では約 10 万のコードクローンを検出しており，その中から 95 %の精度で評価を行うために検出されたコードクローン対から 385 個を抽出して評価を行っている．抽出された 385 個のコードクローン対に対し，実際に調べて確認を行うことで母集団での精度を ±5% で評価することが出来る．f1 の値は，上記の方法で求めた precision と T1-ST3 での recall を用いて算出している．

ASTNN では，コードクローン対と非コードクローン対の双方のタグが付けられたコード片対に対して計測を行っている．そのため，評価はそれらの全ての検出結果に対して評価が行われている．ASTNN で用いられている BCB のデータセットは，コードクローンのタイプごとの最大数を 20,000 にしていたため，もとのコードクローンのタイプの分布に合わせて重みを付けて評価を行った．

3.3.3 STEP3: Google Code Jam データセットでの精度評価

CCLearner では，ソースコードの入ったフォルダ内にあるコードクローン対として検出して提示する．本調査では，ASTNN と同じ条件でコードクローン検出の精度を評価するために，選択したコード片同士の比較を行った．また，提案されている検出器では，以下のようなものは深層学習モデルに通す前にコードクローン対の候補から外していた．

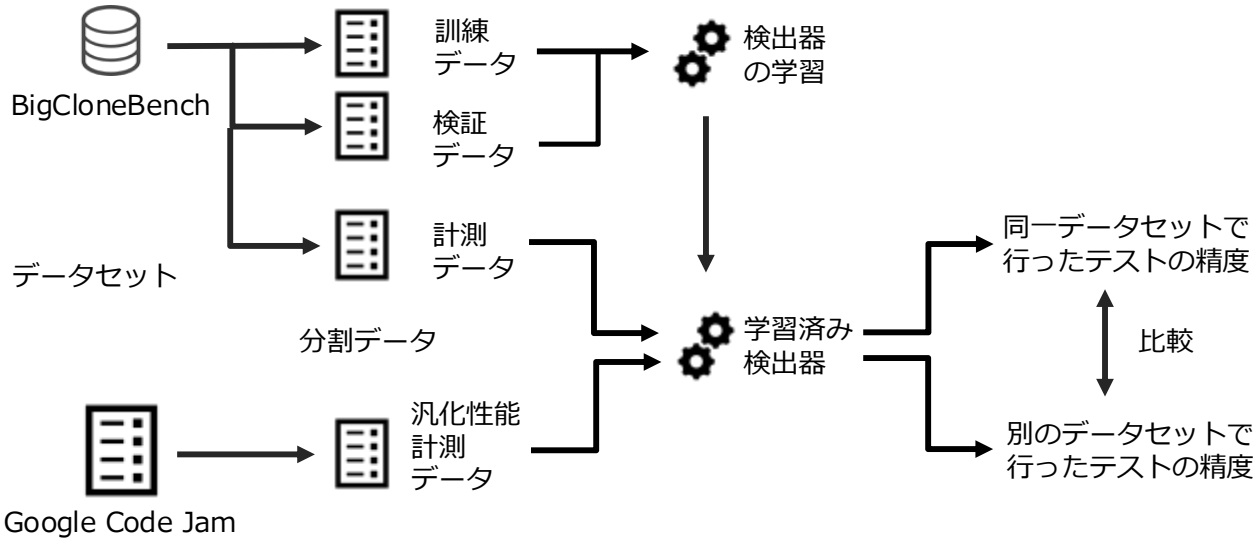

図 1 汎化性能調査で行った実験手順

- メソッド内の行が 6 行以内
- メソッド A とメソッド B の対の場合，A と B のどちらかが他方の 3 倍以上の行数
- 8 つの類似度ベクトルの計算を行い，平均が 0.5 以下となったもの

しかし，本調査では AUC の値も求めたいため，それらに対しても precision と recall と f1 の計算を行うこととした．

ASTNN では，BCB と同様に計測を行い，全てのコードクローン対に対し公平に計測を行ったため，重みを付けずに計算された precision と recall と f1 をそのまま結果とした．

4. 調査実験

この章では本調査で行った BCB での性能の検証と GCJ での性能の検証の実験の結果を説明する．

4.1 BCB での性能の検証結果

BCB で学習した検出器の性能を，BCB を使って計測した．**表 4** には precision，**表 5** には recall，**表 6** には f1，**表 7** には AUC の値（AUC については ASTNN のみ）を示す．ただし，CCLearner の値には論文で報告されているものを使用した．ASTNN では実際に検証を行い，ASTNN の論文で報告されている値に近い性能が出ていることを確認した．

4.2 GCJ での精度の計測結果

BCB で学習を行い BCB で精度の検証も行った検出器で，GCJ で精度の検証を行った．**表 8** は，GCJ での計測結果である．**図 2** は CCLearner での ROC 曲線，**図 3** は ASTNN での ROC 曲線である．

CCLearner と ASTNN ともに，正確にコードクローンの検出を行えているとは言い難い結果となった．

4.2.1 CCLearner の GCJ での精度計測結果

CCLearner は BCB での T4 のコードクローンの検出精度が悪く，今回の GCJ のような T4 のコードクローンを主とした別のデータセットに対しても検出の精度が悪いこ

表 4 BCB で検証したときの precision

	CCLearner [6]	ASTNN
T1	-	99
T2	-	98
VST3/ST3	-	99
MT3	-	99
WT3/T4	-	99
全体	93	99

表 5 BCB で検証したときの recall

	CCLearner [6]	ASTNN
T1	100	100
T2	98	100
VST3	98	93[*1]
ST3	89	-
MT3	28	91
WT3/T4	1	88
全体	-	93

表 6 BCB で検証したときの f1

	CCLearner [6]	ASTNN
T1	-	99
T2	-	99
VST3/ST3	-	96
MT3	-	95
WT3/T4	-	93
全体	93[*2]	96

[*1] VST3 と ST3 を合わせた結果
[*2] Recall には T1-ST3 の値を使用

表 7　BCB で検証したときの AUC（ASTNN のみ）

	ASTNN
T1	100
T2	99
VST3/ST3	96
MT3	95
WT3/T4	94
全体	96

表 8　Google Code Jam データセットでの検証

	precision	recall	f1	AUC
CCLearner	94	5	11	67
ASTNN	50	99	66	39

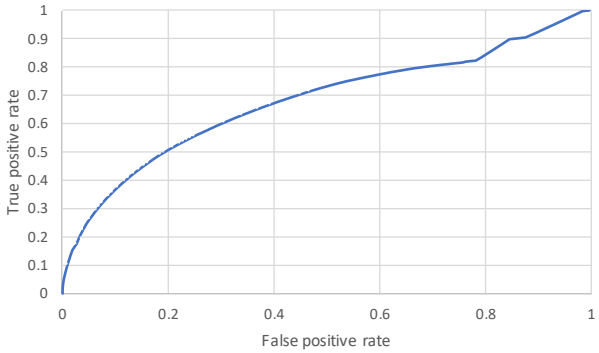

図 2　CCLearner で BCB を学習 GCJ で検証した時の ROC 曲線

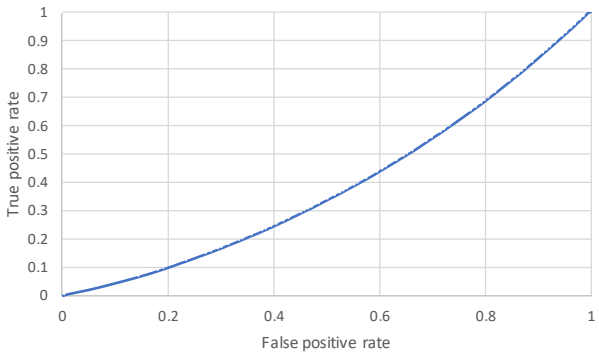

図 3　ASTNN で BCB を学習 GCJ で検証した時の ROC 曲線

とが分かった．precision の値は 94 と高い結果であったが，false negative は非常に多く recall の値は 5 と非常に低い結果となった．AUC の値も 67 と低いため，コードクローンと判断するときの閾値を下げたとしても大幅な精度の向上が見込めないことが分かった．

4.2.2　ASTNN の CCLearner での精度計測結果

ASTNN は BCB での T4 のコードクローンの検出精度が良いが，GCJ に対してはコードクローンの検出精度が悪くなった．precision の値は 50 で BCB での T4 の検証結果よりも悪い結果となった．recall の値は 99 で BCB での検証結果よりも高い精度が出たが，f1 の値を見ると 66 と BCB での T4 の結果よりも大幅に低く，精度が悪くなったことが分かった．precision と recall の値から true positive と false positive が多く，ほとんどのコード片対に対してコードクローン対と判定しているということが分かった．AUC の値も 39 と低いため，コードクローンと判断するときの閾値を上げたとしても大幅な精度の向上が見込めないことが分かった．

4.3　実験結果考察

BCB と GCJ での実験結果より，コードクローンが検出出来るかどうかは検出を行うソースコードによると分かった．BCB は SQL や ZIP 等ソフトウェアに関するソースコードであり，GCJ はオンラインジャッジシステムへの解答ソースコードであるため，使われている語句や構造などに大きな違いがあった可能性がある．検出を行うソースコードによって検出できるかどうかが決まるならば，検出されたコードクローンの精度はどの程度なのか知るためには手作業でコードクローン識別をしなければならない．これは現実の利用方法に則していない．深層学習を用いた検出器においては，本調査で行ったように学習と計測で別のデータセットを使用して汎用的な精度も調べるべきであることが分かった．

5.　まとめ

本調査では，学習済みのコードクローン検出器が，どのデータセットに対しても論文で報告されている精度が出るか汎化性能の調査を行った．CCLearner と ASTNN の 2 種類の深層学習を用いたコードクローン検出器に対し，BCB で学習，GCJ で精度の計測を行い，学習とは違うデータセットに対して検出の精度を計測した場合，学習と同じデータセットに対する検出の精度に比べて f1 の値が低いことが分かった．また，計測した AUC の値の低さから，検出精度の差が検出の時の閾値の差では無いことも確認した．

今後の課題としては，にデータセットの種類を増やすことが挙げられる．現在，検出の精度を測るためのデータセットが少なく，特定のデータセット間での汎化性能の検証しか行えていない．BCB は OSS の特定の機能を持ったソースコード，GCJ はオンラインジャッジシステムの同じ設問に提出されたソースコードからデータセットが作られており，それ以外のソースコードに対して検出器が有効かどうか判断することが出来ない．今後，新たなデータセットを作成することにより，検出器自体の問題だけではなく，学習データが検出器の精度に対して影響するのかどうかも調べることが可能となる．

謝辞　本研究は JSPS 科研費 18H04094, JP19K20240, JP20K11745 の助成を受けたものです．

参考文献

[1] 肥後芳樹，神谷年洋，楠本真二，井上克郎：コードクローンを対象としたリファクタリング支援環境，電子情報通信学会論文誌. D-I， Vol. 88, No. 2, pp. 186–195 (2005).
[2] Baker, B. S.: Finding Clones with Dup: Analysis of an Experiment, *IEEE Transactions on Software Engineering*, Vol. 33, No. 9, pp. 608–621 (online), DOI: 10.1109/TSE.2007.70720 (2007).
[3] Baxter, I. D., Yahin, A., Moura, L., Sant'Anna, M. and Bier, L.: Clone detection using abstract syntax trees, *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)*, pp. 368–377 (online), DOI: 10.1109/ICSM.1998.738528 (1998).
[4] 肥後芳樹，楠本真二，井上克郎：コードクローン検出とその関連技術，電子情報通信学会論文誌. D， Vol. 91, No. 6, pp. 1465–1481 (2008).
[5] Kamiya, T., Kusumoto, S. and Inoue, K.: CCFinder: a multilinguistic token-based code clone detection system for large scale source code, *IEEE Transactions on Software Engineering*, Vol. 28, No. 7, pp. 654–670 (online), DOI: 10.1109/TSE.2002.1019480 (2002).
[6] Li, L., Feng, H., Zhuang, W., Meng, N. and Ryder, B.: CCLearner: A Deep Learning-Based Clone Detection Approach, *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 249–260 (online), DOI: 10.1109/ICSME.2017.46 (2017).
[7] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K. and Liu, X.: A Novel Neural Source Code Representation Based on Abstract Syntax Tree, *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 783–794 (online), DOI: 10.1109/ICSE.2019.00086 (2019).
[8] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K. and Mia, M. M.: Towards a big data curated benchmark of inter-project code clones, *Proc. ICSME 2014*, Victoria, BC, Canada, pp. 476–480 (2014).
[9] Ambient Software Evoluton Group: BigCloneBench, https://github.com/clonebench/BigCloneBench.
[10] Fang-Hsiang Su, Bell, J., Kaiser, G. and Sethumadhavan, S.: Identifying functionally similar code in complex codebases, *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pp. 1–10 (online), DOI: 10.1109/ICPC.2016.7503720 (2016).
[11] Wu, Y., Zou, D., Dou, S., Yang, S., Yang, W., Cheng, F., Liang, H. and Jin, H.: SCDetector: Software Functional Clone Detection Based on Semantic Tokens Analysis, *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, ASE '20, New York, NY, USA, Association for Computing Machinery, p. 821–833 (online), DOI: 10.1145/3324884.3416562 (2020).
[12] Zhao, G. and Huang, J.: DeepSim: Deep Learning Code Functional Similarity, New York, NY, USA, Association for Computing Machinery (2018).
[13] Wang, M. and Deng, W.: Mitigating Bias in Face Recognition Using Skewness-Aware Reinforcement Learning, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).