深層学習を用いたソースコード分類のための動的な学習用データセット改善手法の提案

藤原　裕士[†a)]　崔　恩瀞[††]　吉田　則裕[†††]　井上　克郎[†]

A Dynamic Improvement of a Training Dataset for Source Code Classification Using Deep Learning

Yuji FUJIWARA[†a)], Eunjong CHOI[††], Norihiro YOSHIDA[†††], and Katsuro INOUE[†]

あらまし　近年，深層学習を用いてソースコード分類を行う種々の手法が提案されている．深層学習を用いた手法の分類精度は学習用データセットに大きく影響される．そのため，学習用データセットの構築方法を改善することで，より高い精度のモデルを作成できる可能性がある．そこで本研究では，深層学習を用いたソースコード分類のための動的な学習用データセット改善手法を提案する．提案手法では，まず，学習用データセットを用いてソースコード分類モデルの学習と検証を行う．次に，検証結果に基づいて，学習用データセットを再構築する．この学習と再構築を繰り返し，学習用データセットの改善を行うことで高精度のモデルを作成する．評価実験では，提案手法を用いてソースコード分類モデルを学習し，三つのベースライン手法と分類精度を比較した．その結果，提案手法を用いて学習したモデルの分類精度が最も高いことがわかった．また，提案手法により，モデルの分類精度が 0.64 から 0.96 まで向上することを確認した．

キーワード　ソースコード分類，抽象構文木，グラフ畳み込みネットワーク，学習用データセット

1. まえがき

ソフトウェア開発を効率良く行うために，開発者は既存のソースコードを頻繁に再利用する [1], [2]．ソースコード分類手法は，あらかじめ用意されたクラスに基づいて，入力として与えられたソースコードがどのクラスに属する既存ソースコードと類似しているかを自動で識別する手法である．開発者は，このソースコード分類手法を用いることで，再利用対象のソースコードを素早く特定することができる．現在まで様々なソースコード分類手法が提案されてきた [3]～[7]．特に近年では，TBCNN [6] など，深層学習を用いてソースコード分類を行う手法が提案され，高い分類精度を示した．

一般的に，深層学習モデルの分類精度は学習用データセットに大きく影響される．深層学習を用いてソースコード分類を行う既存研究 [6], [7] では，学習時間や要求メモリを考慮し，ランダムにデータを抽出してクラス間のデータ数を揃える手法であるランダムサンプリングが用いられている．

しかし，ランダムサンプリングは静的な手法（モデルの学習前に 1 度だけ学習用データセットを修正する手法）である．一般的にモデルの学習結果を予測することは困難なため，静的な手法は分類精度の面で非効率的である可能性がある．そのため，モデルの学習結果を用いて動的に何度も学習用データセットの改善を行うことで，より高い精度のモデルを作成できる可能性がある．

そこで本研究では，深層学習を用いたソースコード分類のための動的な学習用データセット改善手法を提案する．提案手法では，実際に深層学習モデルの学習を行った後，モデルの検証結果に基づいて学習が正確に進んでいないクラスに類似ソースコードを追加する．

† 大阪大学，吹田市
Osaka University, 1 Yamadagaoka, Suita-shi, Osaka, 565–0871 Japan
†† 京都工芸繊維大学，京都市
Kyoto Institute of Technology, Hashikami-cho, Matsugasaki, Sakyo-ku, Kyoto-shi, Kyoto, 606–8585 Japan
††† 名古屋大学，名古屋市
Nagoya University, Furo-cho, Chikusa-ku, Nagoya-shi, Aichi, 464–8601 Japan
a) E-mail: y-fujiwr@ist.osaka-u.ac.jp
DOI:10.14923/transinfj.2020PDP0005

これにより，そのクラスの学習が正確に進み，低下していたモデルの分類精度が向上する．

評価実験では，オープンソースソフトウェアに含まれる 20 種類の類似メソッドから，三つのベースライン手法と提案手法をそれぞれ用いて学習用データセットを構築し，ソースコード分類モデルの学習を行い，分類精度の比較を行った．その結果，各クラス間のメソッド数または抽象構文木（Abstract Syntax Tree，以降 AST）のノード数を揃えて学習させたベースライン手法と比べて提案手法は高い精度でメソッド分類ができることを確認した．また，提案手法を用いてモデルの学習と類似ソースコードの追加を繰り返すことで，分類精度が向上することを確認した．

以降，**2.** で本研究の背景について述べる．**3.** で本研究で提案する手法について述べる．**4.** で本研究の評価実験について述べる．**5.** で妥当性の脅威について述べる．**6.** で関連研究について述べる．最後に，**7.** でまとめと今後の課題について述べる．

2. 背　　景

2.1 ソースコード分類

本研究におけるソースコード分類手法とは，既存ソースコードが分類された n 個の クラス $C_1 \ldots C_n$ に対して，入力として与えられたソースコードを，類似ソースコードが含まれるクラス $C_i (1 \leq i \leq n)$ に自動で分類する手法である．この手法を用いることでソフトウェアを効率的に開発することができる．例えば，ソースコードを自動で機能ごとに分類できる場合，大規模なソフトウェアリポジトリに新たに登録されたソースコードに対して機能に関するタグを自動で付与することができる．このように，ソースコード分類手法を用いることで，開発者にとって必要な機能をもったソースコードの検索や，既存ソースコードの再利用がより簡単になり，ソフトウェア開発の生産性の向上が期待できる．

ソースコード分類に関する研究では，現在まで，記述言語による分類 [3]，コンポーネント間の依存関係による分類 [4]，プログラムの意味（機能性）による分類など，多様な手法が提案されている．また，プログラムの意味によるソースコード分類は様々な粒度で取り組まれており，ソフトウェア単位の分類手法 [5] や，メソッド単位の分類手法 [7] などが存在する．

特に近年，深層学習を用いることで高い精度でソースコード分類を行う手法が提案されている [6], [7].

2.2 グラフ畳み込みネットワーク

グラフ畳み込みネットワーク（Graph Convolution Network，以降 GCN）[8] とは，グラフの隣接ノードを畳み込んでいくことによってノードやエッジ，グラフ全体の特徴を抽出するニューラルネットワークである．グラフの学習を行う際，深層学習モデルの入力形式に合わせて元のグラフを変形することがある [6] が，GCN ではグラフの変形は不要である．そのため，グラフの構造情報が欠落しないという利点がある．これにより，グラフを変形する必要があるモデルに比べ，グラフに含まれる情報をより正確に利用した学習を行うことができる．GCN の畳み込み層の例を図 1 に示す．図 1 右上のグラフの中央のノード 0 のベクトル表現を計算する手順について説明する．ノード 0 の畳み込み $n+1$ 層目におけるベクトルは，隣接ノードの n 層目におけるベクトルと各エッジ（ingoing，outgoing，self-loop）の重みから中間ベクトルを計算し，エッジごとの中間ベクトルを全て足し合わせたベクトルを ReLU などの活性化関数（ネットワークの出力を補正する関数）に入力することで得られる．このように GCN では，ノード 0 に隣接しているノード 1，2，3 のベクトル表現を加味してノード 0 のベクトル表現が計算される．

2.3 類似ソースコード作成を目的としたミューテーション

ソースコードに対するミューテーションとは，あらかじめ定めておいたルールに基づいてソースコードを変更することである [9], [10]．一般的に，ミューテーションはテストケースの評価に用いられている [9]．Roy らは，ミューテーションを利用して類似ソースコードを作成することで，類似ソースコード検出ツールの精度を評価する手法を提案している [10]．この手法では，類似ソースコードを作成する際のソースコードの変更ルールをミューテーションオペレータと呼び，

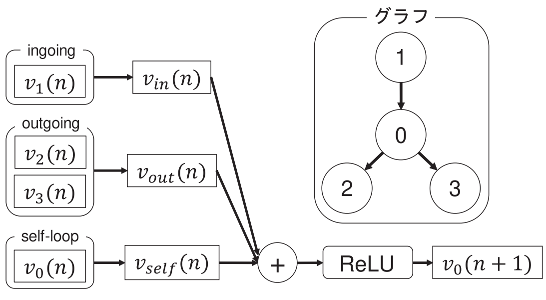

図 1　GCN の畳み込み層の例

以下の 14 種類のミューテーションオペレータを定義している．

mCW 空白の数を変更する．

mCC コメントを変更する．

mCF 改行などのコーディングスタイルを変更する．

mSRI 変数名などのユーザ定義名，変数の型などを規則的に変更する．

mARI 変数名などのユーザ定義名，変数の型などを不規則的に変更する．

mRPE 変数単体の式を別の式に置き換える．

mSIL ある文にわずかな挿入を行う．

mSDL ある文の一部を削除する．

mILs 一つ以上の文を挿入する．

mDLs 一つ以上の文を削除する．

mMLs 一つ以上の文を修正する．

mRDS 宣言文を並べ替える．

mROS 宣言文以外の文を並べ替える．

mCR if 文などの制御構造を別のものに置き換える．

図 2 は，ミューテーションオペレータ mSDL をソースコードに適用した例である．図 2 (a) の 6 行目の文を削除することで，元のソースコードと構文的に類似したソースコード（図 2 (b)）が作成された．本研究では，Roy らが定義したミューテーションオペレータを利用し，類似ソースコードを作成する．

2.4 学習用データセットの改善手法

一般的に，深層学習において学習用データセットの構築方法はモデルの分類精度に大きな影響を与える．そのため，学習用データセットを改善するための手法が提案されている [11]．深層学習を用いた分類モデルの精度を低下させる原因の一つに不均衡データ問題がある．不均衡データ問題とは，クラス間でデータ数に不均衡が存在するため，あるクラスに関して学習が正確に進まず，モデルの分類精度が低下するという問題である．Yan ら [11] は，この不均衡データ問題を解消することで学習用データセットの改善を図っている．

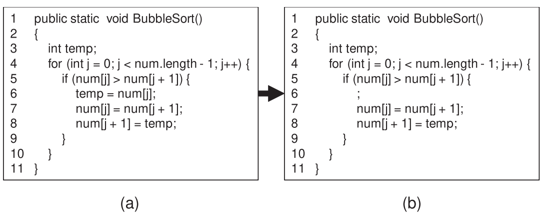

図 2　ミューテーションオペレータ mSDL の例

具体的には，データ数が多いクラスのデータを削除したり，データ数が少ないクラスに新たなデータを追加したりしている．特に，データ数が多いクラスからランダムにデータを削除してクラス間のデータ数を揃える手法はランダムサンプリングと呼ばれる．深層学習を用いてソースコード分類に取り組んだ多くの既存研究 [6], [7] では，このランダムサンプリングが用いられている．

しかし，既存の学習用データセット改善手法は静的な手法である．本研究において静的な手法とは，各クラスのデータの重みを均等にすることを目的とし，モデルの学習結果を用いずに学習用データセットを 1 度だけ修正する手法である．一般的にモデルの学習結果を予測することは困難なため，1 度しか学習用データセットを修正しない静的な手法は分類精度の面で非効率的である可能性がある．そのため，モデルの学習結果を用いて動的に何度も学習用データセットの改善を行うことで，より高い精度のモデルを作成できる可能性がある．

3. 提 案 手 法

本研究では，深層学習を用いたソースコード分類のための動的な学習用データセット改善手法を提案する．データセットの構築方法は深層学習モデルの学習結果に大きな影響を与えるため，より適切なデータセットを構築することで，モデルの分類精度の向上が期待できる．

提案手法は **2.4** で説明した既存の学習用データセット改善手法と異なり，ソースコード分類モデルの学習結果に基づいた動的な学習用データセットの再構築が大きな特徴である．一般的に深層学習の学習結果を予測することは困難なため，提案手法では実際に深層学習モデルの学習を行った後，その学習結果に基づいてデータセットを再構築する．

3.1 用語の定義

類似ソースコードセット 構文的に互いに類似しているソースコードの集合を類似ソースコードセット S と定義する．クラス数を n とする場合，本研究で扱うソースコードは類似ソースコードセット $S_0 \dots S_{n-1}$ のいずれか一つにのみ分類される．類似ソースコードセット $S_i (0 \leq i \leq n-1)$ に属するソースコードは互いに類似している．また，$S_j, S_k (0 \leq j \leq n-1, 0 \leq k \leq n-1, j \neq k)$ の二つの類似ソースコードセットから一つずつ抽出した二つのソースコードは互いに類似していない．

類似ソースコードセット ID 本研究では，n 個の各類似ソースコードセットに対して固有のインデックス $0 \ldots n-1$ を割り当て，モデルにそのインデックスを予測させることでソースコード分類モデルを実現している．ここで，各類似ソースコードセットに対して割り当てられる固有のインデックス $0 \ldots n-1$ を類似ソースコードセット ID と定義する．

学習用データセット モデルの学習に使用するソースコード群を学習用データセットと定義する．

評価用データセット モデルの分類精度の評価に使用するソースコード群を評価用データセットと定義する．

3.2 動的な学習用データセット改善手法

まず，本研究で提案する動的な学習用データセット改善手法を STEP A（Adjustment）と定義する．STEP A では，ソースコード分類モデルの学習及び学習用データセットへのソースコードの追加を，モデルの分類精度が向上しなくなるまで繰り返す．このとき，初期状態の学習用データセットは，ランダムサンプリングで構築する．

本研究では，**2.2** で紹介した GCN を用いたソースコード分類手法を利用する．このソースコード分類手法は，AST をそのまま学習データとして利用することができる．そのため，深層学習を用いた既存手法と異なり AST を入力形式の都合によって変化させないので，プログラムの構造情報が欠落しないという利点がある．この分類手法は，モデルの学習を行う STEP T（Training）と学習済みモデルを用いてソースコードの分類を行う STEP C（Classification）の二つの手順で構成されている．

STEP A の概要を図 3 に示す．

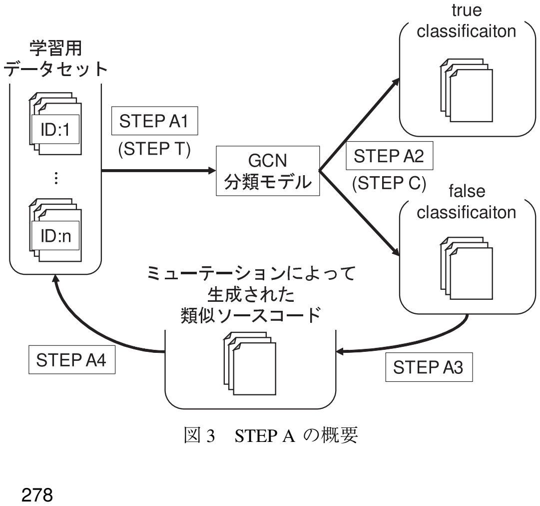

図 3 STEP A の概要

STEP A1 STEP T (**3.2.1**) の手順に基づいて，学習用データセットをソースコード分類モデルに学習させる．

STEP A2 ソースコード分類モデルが学習用データセットを正確に学習できているか検証するため，STEP C (**3.2.2**) の手順に基づいて，学習済みモデルを用いて学習用データセット中のソースコードを分類する．その結果，モデルによって，各ソースコードに対する類似ソースコードセット ID の推測結果が出力されるので，正しい ID が出力されたソースコード (true classification) と，間違った ID が出力されたソースコード (false classification) に分割する．

STEP A3 間違った ID が出力されたソースコードに対して **2.3** のミューテーションを適用し，一定数の類似ソースコードを作成する．本研究では，一つの類似ソースコードを作成する際，AST に変更を加えないオペレータである mCW，mCC，mCF を除いた 11 種類のオペレータのうちいずれか 1 種類をランダムに選択して適用する．

STEP A4 ミューテーションにより作成された類似ソースコードに対して，元のソースコードに割り当てられていた類似ソースコードセット ID と同じ ID を付与した後，一定数を学習用データセットに追加する．

STEP A5 以上の STEP A1～A4 におけるモデルの学習と類似ソースコードの追加を，false classification の数が減少しなくなるまで繰り返す．

3.2.1 GCN を用いたソースコード分類モデルの学習手順 (STEP T)

STEP T では教師あり学習によってソースコード分類モデルを作成する．STEP T の概要を図 4 に示す．

STEP T1 学習対象のソースコードから類似ソースコードセットを構築し，各類似ソースコードセットに対して固有の類似ソースコードセット ID を付与する．

STEP T2 各ソースコードの構文解析を行い，AST に変換する．

STEP T3 各 AST を，隣接行列と特徴行列の形式に変換する．

STEP T4 隣接行列と特徴行列を説明変数，類似ソースコードセット ID を目的変数として，教師あり学習による GCN の学習を行い，ソースコード分類モデルを作成する．

3.2.2 学習済みモデルを用いたソースコード分類手順 (STEP C)

STEP C では，STEP T で作成した学習済みのモデルを利用してソースコード分類を行う．STEP C の概要

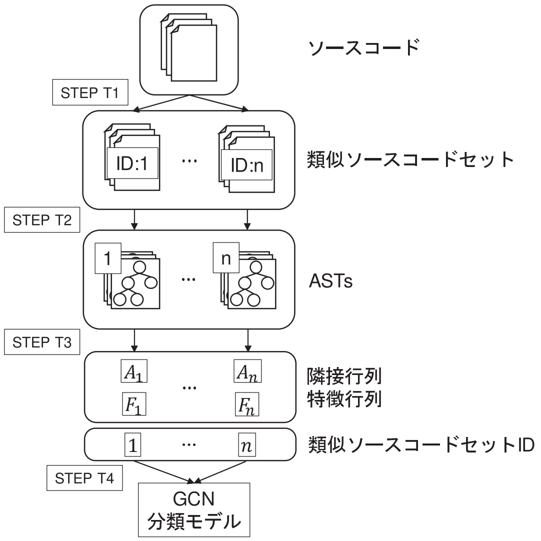

図 4　STEP T の概要

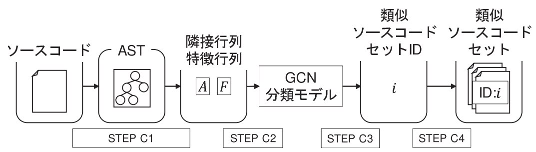

図 5　STEP C の概要

を図 5 に示す．

STEP C1 分類対象のソースコードの構文解析を行い，AST に変換した後，隣接行列と特徴行列の形式に変換する．

STEP C2 隣接行列と特徴行列をソースコード分類モデルに入力する．

STEP C3 分類対象のソースコードに対する類似ソースコードセット ID の推測結果が出力される．

STEP C4 分類対象のソースコードを，出力された類似ソースコードセット ID が示す類似ソースコードセットとして分類する．

4. 評価実験

提案手法が学習用データセットの改善に有効であることを確認するために評価実験を行った．評価実験では，3 種類のベースライン手法と提案手法，計四つの手法をそれぞれ用いて学習用データセットを構築し，構築した各学習用データセットを学習させた各モデルの分類精度を比較した．本評価実験における分類精度は，**4.1** に従って作成される評価用データセットに含まれるメソッドが，モデルによって正しく分類される割合とする．また本評価実験では，分類対象のソースコード単位はメソッドとし，メソッド名や引数は利用せず，メソッド本体の記述を基に分類を行った．評価実験でメソッドを分類対象にした理由は，メソッドは一つの機能のまとまりであるため，再利用対象になりやすいからである．本評価実験では，STEP T2 におけるメソッドの AST 変換は ANTLR(注1) を使用し，その文法ファイルには CPP14.g4(注2) を使用した．また，GCN の実装は Kipf らの実装 [12] を使用した．本評価実験を行った環境を表 1 に示す．

表 1　実験環境

OS	Ubuntu 16.04.6 LTS
CPU	Intel(R)Xeon(R) CPU E5-2623 v4 2.60GHz
GPU	NVIDIA Tesla V100 32GB 1.53GHz
ライブラリ	Tensorflow 1.13.1(注3)

4.1 データセット

評価実験では，オープンソースソフトウェア OpenSSL(注4) のバージョン 0.9.1 から 1.1.1 までの 13 バージョンにおいて，バージョン間で編集が行われた 20 文以上のメソッドを利用した．本実験では，同じプロジェクトの各バージョンにおいて，ファイルパスを含め同じ名称をもつメソッドは同じ機能をもち，異なる名称をもつメソッドは異なる機能をもつという仮定の下で類似ソースコードセットを作成した．まず，学習や評価に使用する類似ソースコードセットを選択した．具体的には，バージョン間で編集が行われている同一名称をもつメソッドを集めた類似ソースコードセットを作成した．ここで，メモリ容量の都合上，作成した類似ソースコードセットをランダムに 20 個を選択して使用した．次に，各類似ソースコードセットから学習用データセットと評価用データセットを作成した方法を図 6 に示す．まず，類似ソースコードセットに含まれる最も古いバージョン以外のメソッドを評価用データセットに追加した．このとき，バージョン間で記述に差分のあるメソッドだけを追加した．この結果，本評価実験における評価用データセットのメソッド数は 166 個となった．次に，類似ソースコードセットに含まれる最も古いバージョンのメソッドにミューテーションを適用することで類似メソッドを一定数作

（注1）：https://www.antlr.org/
（注2）：https://github.com/antlr/grammars-v4/tree/master/cpp
（注3）：https://www.tensorflow.org/
（注4）：https://www.openssl.org/

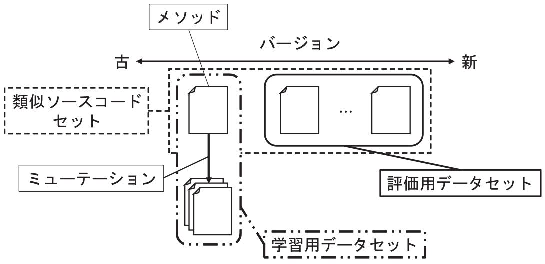

図 6　学習用・評価用データセット作成方法の概要

表 2　各手法で構築したデータセットの詳細

手法	学習用	評価用	割合
Method-oriented-50	1000	166	6:1
Method-oriented-500	10000	166	60:1
Node-oriented	6461	166	39:1
提案手法	1360	166	8:1

表 3　類似メソッド分類の精度

手法	分類精度
Method-oriented-50	0.64
Method-oriented-500	0.81
Node-oriented	0.90
提案手法	0.96

成し，学習用データセットに追加した．ここで，一つの類似メソッドを作成する際，AST に変更を加えないミューテーションオペレータである mCW，mCC，mCF を除いた 11 種類のオペレータのうちいずれか 1 種類をランダムに選択して適用する．また，作成及び追加する類似メソッドの数は **4. 2** のデータセット構築手法に依存する．これを選択した 20 個の類似ソースコードセットに対して適用し，学習用データセットと評価用データセットを作成した．以上のようにデータセットを作成することで，構文的に一致しているメソッドが学習用データセットと評価用データセットの両方に含まれることを避けた．これにより，未学習のメソッドに対する深層学習モデルの分類精度を評価することができる．

しかし，ミューテーションが適用されて作成されたメソッドは元のメソッドと異なる類似ソースコードセットに属する可能性がある．この問題を解決するために，本論文の第一著者は 20 種類の類似ソースコードセットを全て目視で確認し，異なる類似ソースコードセットに属するメソッド間には，実現機能に大きな差異があることを確認した．また，ミューテーションオペレータを適用する際，変更する行数の割合をメソッド全行数の 5% 以下に制限した．以上の目視確認と変更行数割合の制限により，ミューテーションによって機能がわずかに変わったメソッドを作成した場合でも元のメソッドと類似しているため，元のメソッドと同じ類似ソースコードセットに含まれる．

4. 2　データセット構築手法

3 種類のベースライン手法と提案手法の詳細は以下のとおりである．

Method-oriented-n　ミューテーションを適用して作成されたメソッドを各類似ソースコードセット ID につき n 個ずつ学習用データセットに使用する．本評価実験では $n = 50, 500$ の 2 通りについて実験を行う．また，20 種類の類似ソースコードセットを使用するため，Method-oriented-n における学習用データセットのメソッド数は $n * 20$ 個である．

Node-oriented　AST ノードの総数が各類似ソースコードセットにつき約 15000 個になるように，ミューテーションを適用して作成したメソッドを学習用データセットに使用する．この結果，Node-oriented における学習用データセットのメソッド数は 6461 個となった．

提案手法　Method-oriented-50 の状態から学習を開始し，**3. 2** STEP A4 では 10 個のメソッドを新たに追加する．したがって，Method-oriented-50 は提案手法の初期状態である．また，提案手法を用いて学習用データセットを改善した結果，最終的な学習用データセットのメソッド数は 1360 個となった．

また，各手法におけるデータセットの詳細を表 2 に示す．ここで，“学習用” は学習用データセットに含まれるメソッド数，“評価用” は評価用データセットに含まれるメソッド数，“割合” は，“学習用” と “評価用” の比である．

4. 3　実 験 手 順

本評価実験は以下の手順で行う．

(1) **4. 2** で説明した四つの手法に基づいて，**4. 1** のとおりに選択した類似ソースコードセット 20 個から，学習用データセットを構築する．

(2) 手順（1）で構築した 4 種類の学習用データセットをそれぞれ用いて学習を行い，四つのソースコード分類モデルを作成する．

(3) 評価用データセットを用いて，各ソースコード分類モデルの分類精度を評価する．

4. 4　実 験 結 果

各データセット構築手法の分類精度を表 3 に示す．この表から分かるように，提案手法で構築したデータ

セットを学習させたモデルの分類精度が最も高い．次に Node-oriented で構築したデータセットを学習させたモデルの分類精度が高い．Method-oriented-500 で構築したデータセットを学習させたモデルの分類精度は 3 番目に高く，Method-oriented-50 で構築したデータセットを学習させたモデルの分類精度が最も低いことが分かった．

また，**3.2** の STEP A5 によってモデルの学習と類似ソースコードの追加が繰り返されたときの分類精度の変化を図 7 に示す．この図より，STEP A5 によって動的に類似ソースコードの追加を行った結果，ソースコード分類モデルの分類精度は 0.64 から 0.96 まで向上することが確認できた．

4.5 考　　察

評価実験では，学習用データセットを改善するためのベースライン手法三つと提案手法の計四つの手法で構築した学習用データセットを学習させた各ソースコード分類モデルの分類精度を比較した．その結果，表 3 から分かるように，提案手法の学習用データセットを学習させたモデルの分類精度が最も高いことが分かった．分類結果を確認したところ，ベースライン手法では，評価用データセットに含まれるメソッドが全く分類されない類似ソースコードセットが存在した．その反面，提案手法では評価用データセットに含まれるメソッドが全く分類されない類似ソースコードセットは存在しなかった．このように，提案手法を用いることで分類精度を向上させられることが明らかになった．

また，図 7 から分かるように，モデルの学習結果に基づいた類似ソースコード追加 (STEP A5) を繰り返すことで，分類精度を向上させることができた．この結果からも，学習用データセット改善手法の一つとして提案手法が有効であることが明らかになった．また，図 7 において，分類精度が低下する場合がある．これは，STEP A4 において類似ソースコードを学習用データセットに追加する際，追加前よりも追加後の方が理想的な学習データ数の比率から離れてしまったことによるものだと考えられる．

次に，本評価実験では学習用データセットの類似ソースコードの個数を単調に増加させたが，逆に類似ソースコードの個数を減少させる手法も考えられる．ただし，本評価実験においては効率的ではない．まず図 7 のとおり，本評価実験では初期状態の分類精度が 0.5 を超えており，20 クラス分類モデルとしてある程度高い精度である．そのため，学習用データセットを大胆に変更する必要性は低く，微調整を行うべき段階であると考えられる．また，分類結果について調べた結果，正しく分類できた類似ソースコードセットの方が多いことを確認した．そのため，類似ソースコードを増加させる手法のほうが学習用データセットの修正量が少なく済み，学習用データセットの微調整に適している．そのため，分類精度が 0.5 以上の場合は類似ソースコードの個数を増加させる手法が効率的だと考えられる．反対に分類精度が 0.5 より低い場合には類似ソースコードの個数を減少させる手法も視野に入れるべきである．

また，提案手法によって改善された後の学習用データセットのメソッド数と AST ノード数の詳細を表 4 に示す．この表から分かるように，提案手法を用いて構築した学習用データセットにはメソッド数と AST のノード数に各類似ソースコードセット間の不均衡が存在する．しかし，本研究における評価実験では，提案手法を用いて構築したデータセットを学習させたモデルの分類精度が最も高かった．このように，ベースライン手法のようにクラス間のデータ数を揃えることで構築したデータセットより，最終的にメソッド数やノード数が不均衡だったとしても，ソースコード分類モデルの学習結果に基づいて動的に再構築した学習用データセットの方がより正確な深層学習モデルの学習

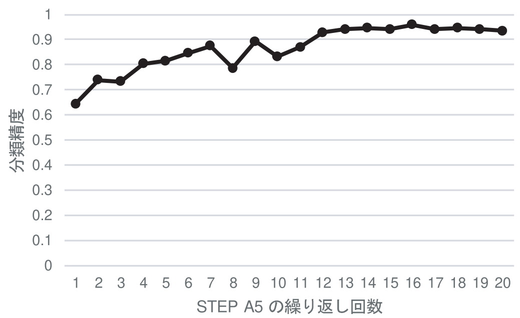

図 7　繰り返し回数と分類精度

表 4　提案手法による改善後のデータセットの詳細

統計量	値
類似ソースコードセットの個数	20
メソッド数の最大値	110
メソッド数の最小値	50
セット合計ノード数の最大値	20717
セット合計ノード数の最小値	4462

を実行できることが分かった．

5. 妥当性の脅威

本研究の妥当性の脅威として，3 点を挙げることができる．

一つめは，評価実験において，提案手法を適用した対象が “GCN を用いたソースコード分類手法” のみである点である．この点に関しては，提案手法を用いる目的が “深層学習を用いたソースコード分類の際に必要となる学習用データセットの改善” であるため，提案手法を適用する対象となるソースコード分類手法についての議論は主眼ではないと考えられる．しかし，他のソースコード分類モデルに対して提案手法を適用した場合に同様の効果が得られることは不明なため，今後検証する必要がある．

二つめは，評価実験で用いた類似ソースコードセットが 20 個と，比較的少ない点である．実験に使用する類似ソースコードセットの個数を増やす場合，データの不均衡が大きくなり，分類精度が向上するまでに更に繰り返し回数が必要になると考えられる．**3. 2** の STEP A を繰り返すごとに，図 7 のように分類精度は向上すると予想されるため，提案手法の有効性は損なわれないと考えられるが，今後，用いる類似ソースコードセットの数を増加させた評価実験を行うことで検証する必要がある．

三つめは，本研究で用いた分類手法では，ミューテーションによって作成された類似メソッドをモデルの学習に使用するため，過学習が発生している可能性があるという点である．この点について，筆者の文献 [13] の **5.** において評価実験を行った．この評価実験ではまず，本研究の提案手法で OpenSSL を用いた学習用データセットを作成した．次に，OpenSSL の派生ソフトウェアである BoringSSL(注5) と LibreSSL(注6) から，学習用データセットのメソッドと同じメソッド名をもつメソッドを評価用データセットに追加した．更に OpenSSL と無関係なソフトウェアの一つである Apache httpd(注7) から，学習用データセットのメソッドと類似したメソッド名をもつメソッドを評価用データセットに追加した．このように学習用データセットに利用するソフトウェアと評価用データセットに利用するソフトウェアを異なるものにすることで過学習の影響について調査することがこの評価実験の目的である．この評価実験の結果，本研究で用いた分類手法に該当する手法（文献 [13] 表 4 の “BoW・識別子名正規化”）の分類精度は 0.79 を記録した．本研究の提案手法の評価実験結果と比べて分類精度は 0.17 だけ低下しているものの，OpenSSL と異なるソフトウェアから収集した類似ソースコードに対する分類精度としては十分に高く，過学習の影響は小さいと考えられる．

6. 関 連 研 究

6.1 不均衡データの効率的な学習

2. 4 で説明したとおり，学習用データセットのクラス間にデータ数の不均衡が存在するとモデルの学習結果や効率に悪影響を及ぼす可能性がある．そのため，不均衡データの効率的な学習に多くの研究者が取り組んでいる．

Yan ら [11] は，オーバサンプリングやダウンサンプリングを用いて学習用データセットを構築することで，不均衡データに対応した分類モデルの学習手法を提案している．この手法は学習用データセットを静的に改善することで不均衡データ問題に対処している．また，Yan ら [14] は，ブートストラップ法を畳み込みニューラルネットワーク（CNNs）に組み込むことで，マルチメディアデータセットの不均衡データ問題に対処した分類モデルの学習手法を提案している．Chen と Shyu [15] は，k 平均法を用いて，不均衡データに対応した分類手法を提案している．これら二つの手法は学習アルゴリズムを静的に修正することで不均衡データ問題に対処している．本研究の提案手法は，学習用データセットを動的に改善する点でこれらの既存手法とは異なる．

6.2 深層学習によるソースコード分類

近年，深層学習を用いてソースコード分類を行う研究が発表されている．

Mou ら [6] は，AST を 2 分木に変形した後，AST ノードのベクトル表現を教師なし学習で作成し，ツリーベースの畳み込みカーネルを AST 全体に対してスライドすることで AST 全体の特徴を捉える TBCNN というモデルを提案し，このモデルをソースコード分類に適用している．Zhang ら [7] は，ソースコードの AST をステートメントレベルに分割してそれぞれベクトル化してから Bi-directional Gated Recurrent Unit（Bi-GRU）[16] にステートメントベクトルのストリームを入力することでソースコードをベクトル化する

（注5）：https://boringssl.googlesource.com/
（注6）：https://www.libressl.org/
（注7）：https://httpd.apache.org/

ASTNN という深層学習モデルを提案し，このモデルをソースコード分類に適用している．

以上の既存手法はモデルの入力形式に合わせて AST を変形するが，本研究で利用したソースコード分類手法は，GCN を使うことにより，AST の構造をそのまま学習できることが特徴である．

7. むすび

本研究では，深層学習を用いたソースコード分類のための動的な学習用データセット改善手法を提案した．提案手法では，ソースコード分類モデルの学習を行った後，学習用データセットを用いてモデルの分類精度を検証する．そして，正しく分類できなかった学習用データセットのソースコードに対してミューテーションを行い，作成された類似ソースコードを学習用データセットに追加する．

評価実験では，オープンソースソフトウェアを対象に，ベースライン手法と提案手法の計四つのデータセット構築手法で構築した各学習用データセットを用いてソースコード分類モデルの学習を行い，分類精度を比較した．その結果，提案手法を用いて構築したデータセットで学習したモデルが最も高い精度でソースコードを分類することを確認した．また，提案手法を用いてモデルの学習と類似ソースコードの追加を繰り返すことで，分類精度が向上することを確認した．

今後の課題として以下の点を挙げることができる．

- 不均衡データ問題を解決するための既存手法と提案手法の分類精度を比較する．
- 本研究の GCN を用いたソースコード分類手法とは異なる，既存のソースコード分類手法に対して提案手法を適用し，提案手法の有効性を評価する．
- 学習させる類似ソースコードセットの数を増加させた，より大規模なデータセットに対する提案手法の有用性を評価する．
- 評価実験の結果が OpenSSL に特化している可能性がある．そのため，他のソフトウェアに対して評価実験を行い，提案手法の汎用性を評価する．
- 提案手法の初期状態が分類精度に与える影響について調査する．
- STEP A4 におけるソースコードの追加個数が，学習用データセット改善の過程や最終的な分類精度にどのような影響を与えるか調査する．

謝辞 本研究は JSPS 科研費 JP19K20240，JP18H04094 の助成を受けた．

文献

[1] R. Hoffmann, J. Fogarty, and D.S. Weld, “Assieme: Finding and leveraging implicit references in a web search interface for programmers,” Proc. UIST 2007, pp.13–22, New York, NY, USA, Oct. 2007. DOI:10.1145/1294211.1294216

[2] K.T. Stolee, S. Elbaum, and D. Dobos, “Solving the search for source code,” ACM Trans. Softw. Eng. Methodol., vol.23, no.3, pp.26:1–26:45, June 2014. DOI:10.1145/2581377

[3] G. Kavita and F. Romano, “C# or java? typescript or javascript? machine learning based classification of programming languages,” https://github.co/2Jif7Sg, 2019.

[4] R. Yokomori, N. Yoshida, M. Noro, and K. Inoue, “Use-relationship based classification for software components,” Proc. QuASoQ 2018, pp.59–66, Nara, Japan, Dec. 2018.

[5] S. Kawaguchi, P.K. Garg, M. Matsushita, and K. Inoue, “Mudablue: An automatic categorization system for open source repositories,” J. Systems and Software, vol.79, no.7, pp.939–953, 2006. DOI:10.1016/j.jss.2005.06.044

[6] L. Mou, G. Li, L. Zhang, T. Wang, and Z. Jin, “Convolutional neural networks over tree structures for programming language processing,” Proc. AAAI 2016, pp.1287–1293, Phoenix, Arizona, USA, Feb. 2016. DOI:10.5555/3015812.3016002

[7] J. Zhang, X. Wang, H. Zhang, H. Sun, K. Wang, and X. Liu, “A novel neural source code representation based on abstrace syntax tree,” Proc. ICSE 2019, pp.783–794, Montréal, QC, Canada, May 2019. DOI:10.1109/ICSE.2019.00086

[8] M. Schlichtkrull, T.N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” Proc. ESWC 2018, pp.593–607, Heraklion, Crete, Greece, June 2018. DOI:10.1007/978-3-319-93417-4_38

[9] Y. Jia and M. Harman, “An analysis and survey of the development of mutation testing,” IEEE Trans. Software Engineering, vol.37, no.5, pp.649–678, Sept. 2010. DOI:10.1109/TSE.2010.62

[10] C.K. Roy and J.R. Cordy, “A mutation/injection-based automatic framework for evaluating code clone detection tools,” Proc. ICSTW 2009, pp.157–166, Denver, CO, USA, April 2009. DOI:10.1109/ICSTW.2009.18

[11] Y. Yan, Y. Liu, M.-L. Shyu, and M. Chen, “Utilizing concept correlations for effective imbalanced data classification,” Proc. IRI 2014, pp.561–568, Redwood City, CA, USA, Aug. 2014. DOI:10.1109/IRI.2014.7051939

[12] T.N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” Proc. ICLR 2017, Palais des Congrés Neptune, Toulon, France, April 2017.

[13] 藤原裕士，“抽象構文木とグラフ畳み込みネットワークを用いた類似ソースコード検索，” 大阪大学大学院情報科学研究科修士学位論文，Feb. 2020. http://sel.ist.osaka-u.ac.jp/lab-db/Mthesis/contents.ja/150.html

[14] Y. Yan, M. Chen, M.-L. Shyu, and S.-C. Chen, “Deep learning for imbalanced multimedia data classification,” Proc. ISM 2015, pp.483–488, Miami, FL, USA, Dec. 2015. DOI:10.1109/ISM.2015.126

[15] C. Chen and M.-L. Shyu, “Clustering-based binary-class classification for imbalanced data sets,” Proc. IRI 2011, pp.384–389, Las Vegas, NV, USA, Aug. 2011. DOI:10.1109/IRI.2011.6009578

[16] D. Tang, B. Qin, and T. Liu, “Document modeling with gated recurrent neural network for sentiment classification,” Proc. EMNLP 2015, pp.1422–1432, Lisbon, Portugal, Sept. 2015. DOI:10.18653/v1/D15-1167

（2020 年 5 月 18 日受付，8 月 27 日再受付，
12 月 1 日早期公開）

藤原　裕士

平 29 大阪大学基礎工学部情報科学科卒，令元大阪大学大学院情報科学研究科博士前期課程了．深層学習を用いたコードクローン分析手法に関する研究に従事．

崔　　恩瀞　（正員）

平 27 大阪大学大学院情報科学研究科博士後期課程了．同年同大学大学院国際公共政策研究科助教．平 28 奈良先端科学技術大学院大学情報科学研究科助教．平 30 より同大学先端科学技術研究科助教（改組による）．平 30 より京都工芸繊維大学情報工学・人間科学系助教．博士（情報科学）．コードクローン管理やリファクタリング支援手法に関する研究に従事．

吉田　則裕　（正員）

平 21 大阪大学大学院情報科学研究科博士後期課程了．同年日本学術振興会特別研究員（PD）．平 22 奈良先端科学技術大学院大学情報科学研究科助教．平 26 名古屋大学大学院情報科学研究科附属組込みシステム研究センター准教授．平 29 より同大学大学院情報学研究科附属組込みシステム研究センター准教授（改組による）．博士（情報科学）．コードクローン分析手法やリファクタリング支援手法に関する研究に従事．

井上　克郎　（正員：フェロー）

昭 59 大阪大学大学院基礎工学研究科博士後期課程了（工学博士）．同年大阪大学基礎工学部情報工学科助手．昭 59～61 ハワイ大学マノア校コンピュータサイエンス学科助教授．平 3 大阪大学基礎工学部助教授．平 7 同学部教授．平 14 より大阪大学大学院情報科学研究科教授．ソフトウェア工学，特にコードクローンやコード検索などのプログラム分析や再利用技術の研究に従事．