情報処理学会論文誌 Vol.0 No.0 1–9 (??? 1959)

深層学習を用いたソースコード分類のための学習用データセット改善手法の提案

藤原 裕士[1,a)] 崔 恩瀞[2] 吉田 則裕[3] 井上 克郎[1]

受付日 xxxx年0月xx日, 採録日 xxxx年0月xx日

概要：ソフトウェア開発の効率化のために，開発者は過去に開発されたソースコードを頻繁に再利用する．そのとき，類似ソースコード分類手法を用いることで，開発者は再利用対象のソースコードを見つけることができる．ソフトウェアの開発支援のために，様々なソースコード分類手法が提案されており, 特に近年では深層学習を用いる手法が多く提案されている．深層学習モデルの精度はデータセットに大きく左右される．しかし，既存の深層学習を用いた手法では，学習用データセットを構築した後に，データセットの改善を行わない．そこで，本研究は深層学習を用いたソースコード分類のための学習用データセット改善手法を提案する．本手法では，まず，学習用データセットを用いて訓練を行い，類似ソースコードの分類器モデルを作成する．その後，作成されたモデルを検証しその結果に基づいて，学習用データセットを再構築する．このモデルの訓練とデータセットの再構築を繰り返すことで，より高い精度の類似ソースコード分類器モデルの作成が期待できる．評価実験では，提案手法によりデータセットを改善することによって，モデルの分類精度が 0.75 から 1.00 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

An Approach to Improving a Training Dataset for Source Code Classification Based on Deep Learning

Yuji Fujiwara[1,a)] Eunjong Choi[2] Norihiro Yoshida[3] Katsuro Inoue[1]

Received: xx xx, xxxx, Accepted: xx xx, xxxx

Abstract: To develop software more effectively, developers frequently reuse existing source code. They can find relevant source code using a source code classification approach. Many approaches for classifying similar source code have been proposed for supporting software development. In particular, many approaches using deep learning have been proposed in recent decades. The accuracy of the deep learning model highly relies on the training dataset. However, previous approaches using deep learning do not improve the training dataset after construction. To mitigate this problem, this study proposes an approach to improving a training dataset for source code classification using deep learning. At first, a classifier model for classifying similar source code is constructed after a dataset of similar source code is trained. Next, the created model is verified, and then, the training data set is reconstructed based on the results of the verification. Finally, the model is created repeating training the model and reconstructing the dataset. In the experiment, we confirmed that the accuracy of the source code classification was improved from 0.75 to 1.00 by applying the proposed approach.

Keywords: source code classification, abstract syntax tree, graph convolution networks, training dataset

[1] 大阪大学
Osaka University, Suita, Osaka, 565-0871, Japan
[2] 京都工芸繊維大学
Kyoto Institute of Techinology, Kyoto, 606-8585, Japan
[3] 名古屋大学
Nagoya University, Nagoya, Aichi, 464-8601, Japan
[a)] y-fujiwr@ist.osaka-u.ac.jp

1. まえがき

ソフトウェア開発の効率化のために，開発者は過去に開発されたソースコードを再利用する．ソースコードの再利用にあたって，与えられたソースコードと類似しているソースコードを識別する手法を使うことで開発者は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．ソフトウェアの開発支援のために，様々なソースコード分類手法が提案されてきた．特に近年では，ASTNN[1] など，深層学習を用いて，ソースコードの意味や機能の情報を考慮した分散ベクトル表現を生成することを目的とした手法が提案され，機能別のソースコード分類などのソースコード解析タスクにおいて高い精度を実現したことが報告されている．

一般的に，深層学習モデルの訓練結果は，学習用データセットの内容に大きく影響されるため，学習用データセットの構成には注意を払う必要がある．深層学習を用いた分類器モデルの訓練に悪影響を及ぼす問題の 1 つに，不均衡データの問題が存在する．この問題は，分類カテゴリ間における学習データ数の不均衡が原因で，ある特定の分類カテゴリに関して学習が進まなくなることである．この問題への対処法としては，各カテゴリから同じ数のデータをランダムサンプリングする手法含め，様々な手法が提案されており，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 [2][3][4].

しかし，深層学習を用いたソースコード分類を行う既存研究では，データセットの構築手順に詳しく言及されることは少なく，学習用データセットの構築手法に関する工夫はなされていない．多くの場合では，学習に必要な時間やメモリを考慮し，可能な範囲内で適当と思われる数だけランダムにデータを抽出して学習させている．そのため，適用するモデルや問題に対して，用意したデータセットが最適化されているかどうか不明であり，データセットの構成次第でさらに良い精度のモデルを作成することが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タスク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．本手法では，まず，学習用ソースコードを構文解析することで抽象構文木（以下 AST）を生成する．次に，その AST を学習データとしてグラフ畳み込みネットワークを訓練することで，ソースコードの分類器モデルを作成する．次に，作成したモデルを用いて学習用ソースコードを正しく分類できるかどうかを検証し，正しく分類できなかった学習用ソースコードに対してミューテーションを行い，類似ソースコードを作成し，学習用データセットに追加する．以降，モデルの訓練と類似ソースコードの追加を，正しく分類できなかった学習用ソースコードの数が減らなくなるまで繰り返す．

評価実験では，オープンソースソフトウェアに存在する 20 種類のメソッドを，提案手法で構築した学習用データセットを用いて学習させ，メソッドの分類器を作成し，分類精度を調査した．その結果，各分類カテゴリ間のメソッド数や AST のノード数を揃えて学習させた場合と比べて，高い精度でメソッドの分類が可能であることを確認した．

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

2. 背景

本章では，本研究の背景として，ソースコード分類，グラフ畳み込みネットワーク，ミューテーション，学習用データセットにおける不均衡データの問題について述べる．

本研究では，3 章以降で，深層学習を用いたソースコード分類（2.1 節）のための学習用データセット改善手法について説明する．この手法で用いる深層学習モデルは，ソースコードの AST を学習データとして扱うことを目的とした，グラフ畳み込みネットワーク（2.2 節）である．

また，深層学習では，モデルの頑強性を高めるために，元の学習データを少し編集して新たな学習データを増やすことがある．例えば画像認識では，画像を少し回転させたり，平行移動させることで新たな画像を作成することがある．本研究においてもモデルの頑強性向上を目的として，Roy らが提案したミューテーション（2.3 節）を用いて，学習用ソースコードを増やす．また，深層学習を用いたソースコード分類では，分類カテゴリ間の不均衡データ問題（2.4 節）に対処する必要がある．本研究では，ミューテーションによって増やした学習用ソースコードを適当な数だけ学習用データセットに追加することで，不均衡データの問題に対処する．

2.1 ソースコード分類

ソースコード分類はソフトウェアの効率的な開発に有効である [5]．例えば，ソースコードを自動で機能別に分類することができれば，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ようになる．これによって，ユーザーにとって必要な機能を持ったソースコードの検索や，ソフトウェア開発における既存ソースコードの再利用をより簡単に行うことができるようになり，ソフトウェア開発の生産性向上が期待できる．

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（以下 GCN）[6] とは，グ

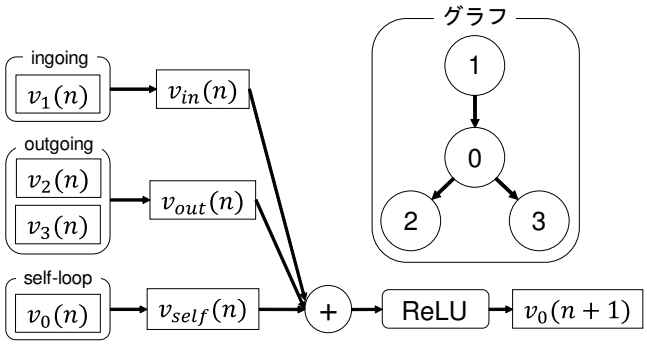

図 1 GCN の畳み込み層の例
Fig. 1 An Example of the Convolution Layer of a GCN

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り，化合物の推定などに用いられる．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，エッジ毎の中間ベクトルを全て足し合わせたベクトルを ReLU のような活性化関数（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．

深層学習モデルには，学習の過程で自動で調整される内部パラメータの他に，開発者によって手動で設定されるハイパーパラメータが存在し，ハイパーパラメータは，深層学習モデルの訓練の結果や効率に大きく影響する．GCN のハイパーパラメータには，隠れ層のユニット数，ノードベクトルの次元数，畳み込み層の数，学習率，重み減衰，Dropout が存在する．隠れ層のユニット数とは，図 1 における中間ベクトル $v_{in,out,self}$ の次元数であり，ネットワークの表現能力や汎化性能に影響する．ノードベクトルの次元数とは，図 1 における $v_{0,1,2,3}$ の次元数である．畳み込み層の数とは，図 1 の計算を実行する回数である．学習率は，ネットワークのパラメータを 1 度の調整でどの程度変化させるかを決定する値で，大きいとネットワークのパラメータが収束しなくなり，小さいとパラメータが収束するまでに時間がかかるようになる．重み減衰とは，ネットワークのパラメータの値域にどの程度制限をかけるかを決定する値で，過学習の対策に用いられる．Dropout とは，モデルの訓練中にニューラルネットワークのユニットを無効化する確率で，過学習の対策に用いられる．

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

本研究では，類似ソースコードの作成を目的として，ソースコードにミューテーションを適用する．ミューテーションとは，Roy らが提案したコードクローン検出ツール評価手法 [7] で用いられる，ルールに基づいたソースコードの変更のことである．Roy らは，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，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．

```
1   public static  void BubbleSort()
2   {
3       int temp;
4       for (int j = 0; j < num.length - 1; j++) {
5           if (num[j] > num[j + 1]) {
6               temp = num[j];
7               num[j] = num[j + 1];
8               num[j + 1] = temp;
9           }
10      }
11  }
```

(a)

```
1   public static  void BubbleSort()
2   {
3       int temp;
4       for (int j = 0; j < num.length - 1; j++) {
5           if (num[j] > num[j + 1]) {
6               ;
7               num[j] = num[j + 1];
8               num[j + 1] = temp;
9           }
10      }
11  }
```

(b)

図 2 ミューテーションオペレータ mSDL の例
Fig. 2 An Example of the Mutation Operator mSDL

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ー定義名，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ー定義名，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: 1 つ以上の文を挿入する．
mDLs: 1 つ以上の文を削除する．
mMLs: 1 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 1（b））が作成された．

2.4 学習用データセットにおける不均衡データの問題

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訓練に大きな影響を与える．深層学習を用いた分類器モデルの訓練の際に発生する問題の 1 つに，不均衡データの問題がある．この問題は，分類カテゴリ間でデータの数に不均衡が存在すると，ある特定の分類カテゴリに関して学習が進まず，モデルの分類精度に悪い影響を与えることがある，というものである．そのため，この問題に対処するための様々な手法が提案されている [2][3][4]．

不均衡データ問題の対処法は，モデル指向型アプローチとデータ指向型アプローチの大きく 2 つに分かれる．モデル指向型アプローチは，損失関数の計算においてデータ数

が少ないカテゴリの重みを増やすなどの方法で，学習のアルゴリズムを修正する手法である．データ指向型アプローチは，データ数が多いカテゴリのデータ数を減らしたり，データ数が少ないカテゴリに新たなデータを追加したりすることで，データ数の不均衡そのものを無くす手法である．このように，不均衡データ問題には様々な対処法が存在する．

しかし，深層学習を用いてソースコード分類やコードクローン検出に取り組んだ既存研究では，ほとんどの場合，データ指向型アプローチの 1 つで，単純な手法であるランダムサンプリングが用いられており，データセット構築手法の比較は行われていない．そのため，使用する深層学習モデルや解決したい問題に対して，用意したデータセットの構成が適切であるかどうか不明であり，学習用データセットの構成次第でさらに良い精度のモデルを作成することができる可能性がある．

3. 提案手法

本研究では，GCN 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．データセットの構成は深層学習モデルの訓練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．以降，3.1 章では，用語の定義について述べる．3.2 節では，本研究で提案する動的な学習用データセット改善手法及び，本研究において提案手法を適用する対象となる，GCN を用いたソースコード分類手法について述べる．

3.1 用語の定義

類似ソースコードセット: 本研究では，構文的または意味的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と定義する．本研究においては，類似ソースコードセット中のどの 2 つのソースコードを抽出しても，構文的または意味的に類似しているとする．

類似ソースコードセット ID: 本研究におけるソースコード分類では，学習済みの類似ソースコードセットに入力ソースコードを分類するモデルの作成を目的としている．このとき，各類似ソースコードセットに対して固有の数値を割り当て，モデルにその数値を予測させることで，分類器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数値を，本研究では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: 本研究では，データセットに含まれるソースコードのうち，モデルの訓練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: 本研究では，データセットに含まれるソースコードのうち，モデルの検証や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．

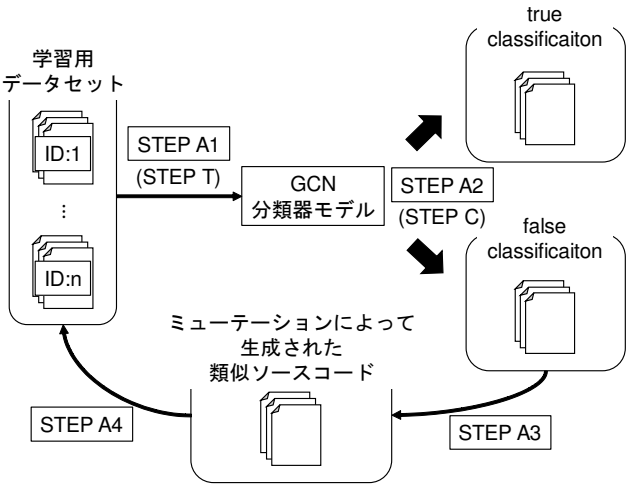

図 3 STEP A の概要
Fig. 3 An Overview of the STEP A

3.2 動的な学習用データセット改善手法

この節では，本研究で提案する動的な学習用データセット改善手法及び，本研究において提案手法を適用する対象となる，GCN を用いたソースコード分類手法について説明する．

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 の概要を図 3 に示す．STEP A では，モデルの訓練と学習データの追加を，モデルの分類精度が十分に高まるまで繰り返す．この手法は既存の不均衡データ問題の対処手法と異なり，深層学習モデルの訓練結果に基づいて動的に学習用データセットを再構築するため，利用する深層学習モデルや解決したい問題に適した学習用データセットを構築することができる．

また，GCN を用いたソースコード分類手法は，STEP T（Training）と STEP C（Classification）の 2 ステップに分かれている．この分類手法は，GCN を利用するので，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，ソースコードをトークンのストリームとみなして学習させる既存手法とは異なり，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．

以降，本節では，STEP A を適用する対象となる GCN を用いたソースコード分類手法の STEP T と STEP C について先に説明し，最後に，提案手法である STEP A について説明する．

3.2.1 GCN による分類器モデルの訓練手順

この節では，STEP T について説明する．この STEP では，説明変数を AST，目的変数を類似ソースコードセット ID とし，教師あり学習によって GCN の訓練を行い，ソースコード分類器のモデルを作成する．STEP T の概要を図 4 に示す．

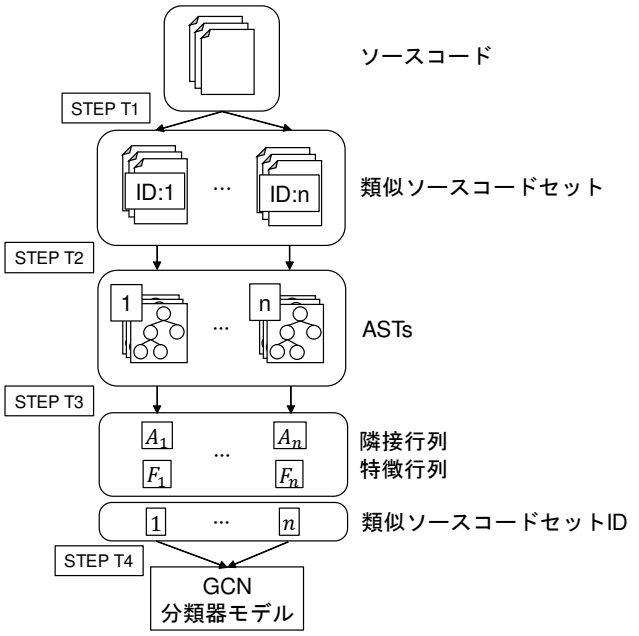

図 4 STEP T の概要
Fig. 4 An Overview of the STEP T

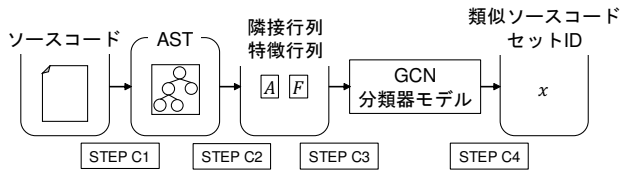

図 5 STEP C の概要
Fig. 5 An Overview of the STEP C

STEP T1. 学習データとなるソースコードをクラスタリングすることで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. 各ソースコードをパースし，AST に変換する．
STEP T3. 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 の訓練を行い，ソースコード分類器モデルを作成する．

3.2.2 GCN 分類器モデルを用いたソースコード分類手順

この節では，STEP C について説明する．この STEP では，分類したいソースコードの AST をソースコード分類器モデルに入力することで，そのソースコードの類似ソースコードセット ID 推測結果が出力され，入力したソースコードは，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に分類される．STEP C の概要を図 5 に示す．
STEP C1. 分類対象のソースコードをパースし，AST 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器モデルに入力する．
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に分類する．

3.2.3 動的な学習用データセット改善アルゴリズム

この節では，STEP A について説明する．STEP A では，3.2.1 節の STEP T と 3.2.2 節の STEP C の手順を利用し，モデルの訓練と学習データの追加を，モデルの分類精度が十分に高まるまで繰り返す．GCN を用いたソースコード分類手法に対する STEP A の概要を図 3 に示す．
STEP A1. STEP T の手順に基づいて，用意したソースコードからなるデータセットを GCN に学習させ，ソースコード分類器のモデルを作成する．
STEP A2. STEP C の手順に基づいて，モデルの訓練に使用した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が存在する場合は，そのソースコードに 2.3 節のミューテーションを適用することで構文的に類似した類似ソースコードを作成する．適用するミューテーションオペレータは，mCW，mCC，mCF 以外の 11 種類とする．
STEP A4. 作成し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 におけるモデルの訓練と類似ソースコードの追加を，間違った ID が出力されたソースコードの数が減少しなくなるまで繰り返す．

このように，モデルの学習結果を踏まえて分類に失敗したデータを増やすことで，データセットの構成を，実現したいタスクと使用するモデルに合わせて改善することができる．

4. 評価実験

この章では，提案したデータセット改善手法の評価実験について説明する．この評価実験では，ベースライン手法として 2 種類のデータ指向型アプローチを取り上げ，提案手法を含む 3 つの手法から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用いて訓練された分類器モデルの分類精度を比較し，提案手法が有効であることを示す．本評価実験において，分類するソースコードの単位はメソッドとする．本評価実験を行うにあたって，GCN

表 1 実験環境
Table 1 Experimental Environment

OS	Ubuntu 16.04.6 LTS
CPU	Intel(R)Xeon(R) CPU E5-2623 v4 2.60GHz
GPU	NVIDIA Tesla P100 16GB 1.30GHz
ライブラリ	Tensorflow 1.13.1[*1]

の実装は Kipf らの実装 [8] を使用した．また，本評価実験を行った環境を表 1 に示す．

4.1 データセット

本評価実験では，OpenSSL 0.9.1〜1.1.0[*2]において，バージョン間で編集が行われたメソッドを利用する．まず，同じプロジェクトの各バージョン間において，ファイルパスを含め同じ名称を持つメソッドは，同じ機能を持つと考えられるので，バージョン間で編集が行われている同一名称を持つメソッドを集めて，類似ソースコードセットを作成し，その類似ソースコードセットからランダムに 20 個だけ選択し，評価用データセットとする．また，選択した各類似ソースコードセットの最も古いバージョンのメソッドに 2.3 節のミューテーションを適用し，作成したメソッド群を学習用データセットに使用する．

4.2 データセット構築手法の設定

この節では，比較する 2 種類のデータ指向型アプローチと提案手法の設定について説明する．

Method-oriented: 2.3 節のミューテーションを適用して作成したメソッドを各類似ソースコードセット ID につき 1000 個ずつ学習用データセットに使用する．

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 7000〜8000 個になるように，2.3 節の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．

提案手法: 2.3 節のミューテーションを適用して作成したメソッドを各類似ソースコードセット ID につき 100 個ずつ学習用データセットに加えた状態から開始し，3.2.3 節 STEP A4 では 100 個のメソッドを新たに追加する．

4.3 ハイパーパラメータ

評価実験を行うにあたって，Method-oriented のデータセットを用いて GCN モデルのハイパーパラメータを調整した．ハイパーパラメータの探索は，グリッドサーチと，Hyperopt[9] で実装されている Tree of Parzen Estimators（TPE）[10] アルゴリズムを併用して行った．その結果，グラフ畳み込み層の数を 4 層，グラフ畳み込みネットワークの隠れ層のユニット数を 256，dropout を 0.2 に設定する場合に最もモデルの分類精度が良くなったため，本評価実験では，このハイパーパラメータ設定を用いて以降の評価実験を行う．また，最適化アルゴリズムは Adam[11] を利用した．

[*1] https://www.tensorflow.org/
[*2] https://www.openssl.org/

表 2 分類精度
Table 2 Classification Accuracy

手法	分類精度
Method-oriented	0.92
Node-oriented	0.94
提案手法	1.00

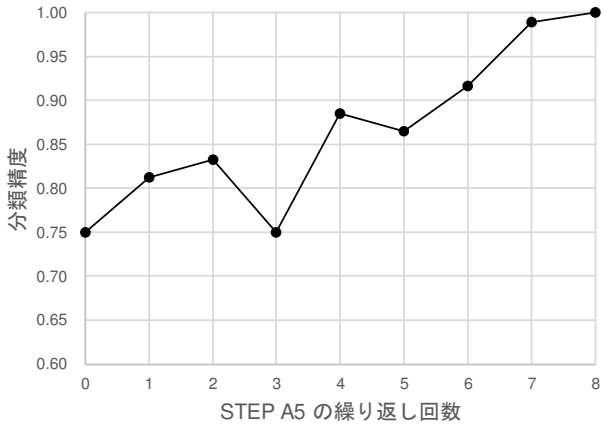

図 6 繰り返し回数と分類精度
Fig. 6 The Number of Repetitions and Classification Accuracy

4.4 実験手順

本評価実験は以下の手順で行う．

(1) 4.2 節で説明した 3 つの手法に基づいて，4.1 節で説明した類似ソースコードセット 20 個からなるデータセットから，学習用データセットを構築する．
(2) (1) で構築した 3 種類の学習用データセットを用いて訓練を行い，3 つのソースコード分類器モデルを作成する．
(3) 評価用データセットを用いて，各ソースコード分類器モデルの分類精度を評価する．

4.5 実験結果

各データセット構築手法の分類精度を表 2 に示す．提案手法で構築したデータセットによって訓練されたモデルの分類精度が最も高く，次に Node-oriented で構築したデータセットによって訓練されたモデルの分類精度が高く，Method-oriented で構築したデータセットによって訓練されたモデルの分類精度が最も低いという結果になった．

次に，3.2.3 節 STEP A5 によってモデルの訓練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 6 に示す．STEP A5 によって動的に学習データの追加を行った結果，最終的な分類精度は 1.0 に到達した．

表 3 提案手法による改善後のデータセットの詳細
Table 3 Details of the Improved Dataset by the Proposed Approach

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	400
メソッド数の最小値	100
セット合計ノード数の最大値	39656
セット合計ノード数の最小値	5402

また，提案手法によって改善された後のデータセットのメソッド数と AST ノード数の詳細を表 3 に示す．一般的に，深層学習モデルを用いた分類タスクにおいて，データセットの不均衡は，モデルの学習に悪い影響を与えると考えられている．しかし，本研究における評価実験によって，提案手法を用いて構築した，メソッド数と AST ノード数に各類似ソースコードセット間の不均衡があるデータセットを用いて訓練されたモデルが最も高い分類精度を実現できることを確認した．このように，使用する深層学習モデルと解決したい問題によっては，単純にデータセットの不均衡を解消して構築したデータセットよりも適切なデータセット構成が存在する場合があることが分かった．

4.6 考察

評価実験では，単純な方法で不均衡データ問題に対処したベースライン手法 2 つと提案手法の計 3 つの手法で構築したデータセットを用いて訓練された GCN 分類器モデルの分類精度を比較した．その結果，提案手法のデータセットによって訓練されたモデルの分類精度が最も高くなった．これは，データセットの構成に関しても，ハイパーパラメータのように調整を行うことで，モデルの性能を良化させる余地があることを示している．表 3 に示すように，提案手法で構築したデータセットは，メソッド数と AST ノード数に各類似ソースコードセット間の不均衡がある．特に，ID が 18 の類似ソースコードセットに含まれるメソッドは，他の類似ソースコードセットのメソッドよりもノード数が少なく，ソースコードが短いにも関わらず，1 度もデータ追加は発生しなかった．また，3.2.3 節 STEP A5 においてモデルの訓練と類似ソースコードの追加を繰り返しているとき，ID が 7，14 の類似ソースコードセットには，交互に類似ソースコードが追加されていった．このことから，類似ソースコードセット ID：18 のメソッドのように，GCN によって捉えられやすい特徴を持つメソッドや，類似ソースコードセット ID：7，14 のメソッドのように，GCN によって混同されやすい特徴を持つメソッドが存在することがわかる．このように，提案手法を用いることで，深層学習モデルと学習データの特徴に合わせたデータセットを構築することができる．また，3.2.3 節 STEP A5 の進行状況に注目することで，各分類カテゴリ間の関係性についての情報を得ることができる．

5. 妥当性の脅威

本研究の妥当性の脅威として，2 点を挙げることができる．

1 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．しかし，提案手法を用いる目的は不均衡データ問題の軽減であるため，ソースコード分類タスク全般に発生しうる問題に取り組んでいるといえる．そのため，提案手法を適用する対象となるソースコード分類手法の詳細についての議論は主眼ではないと考えられるが，分類精度や 3.2.3 節 STEP A5 の繰り返し回数に影響する可能性は存在するので，今後検証する必要がある．

2 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した場合，データの不均衡が大きくなる．そのため，3.2.3 節 STEP A 開始時の分類精度は低くなり，分類精度の収束までにさらに繰り返し回数が必要になると考えられる．しかし，3.2.3 節 STEP A を繰り返す毎に分類精度は向上すると思われるので，提案手法の有効性は損なわれないと考えられるが，今後，実験を行うことで検証する必要がある．

6. 関連研究

6.1 深層学習の最適化

効率の良いモデルの訓練や，精度の良いモデルを得るためには，利用できるリソースを最大限に活用し，最適な深層学習モデルの訓練を行うことが必要である．このとき，学習用データセットのカテゴリ間にデータ数の不均衡が存在する場合，モデルの訓練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 [12] は，オーバーサンプリングやダウンサンプリングを用いてデータを操作することで，不均衡データに対応した分類モデルの訓練手法を提案している．この手法は，データ指向型アプローチに該当する．また，Yan ら [3] 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，不均衡データに対応した分類モデルの訓練手法についても提案している．Chen と Shyu[2] 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．これらは，モデル指向型アプローチに該当する．本研究の提案手法は，データ指向型アプローチに該当し，データセットを動的に構築する点で，以上の手法とは異なる．

また，深層学習モデルのハイパーパラメータ調整は，不均衡データへの対処と同様に，訓練の結果や効率に大きく影響する．そのため，モデルのハイパーパラメータ探索を

効率的に行う手法や，ハイパーパラメータ調整を自動化する手法が提案されている．

Bergstra と Bengio[13] は，ハイパーパラメータの最適化手法の 1 つとして，ランダムサーチという手法を提案している．ハイパーパラメータを離散的に変化させて探索するグリッドサーチと比べて，多くの種類のハイパーパラメータの値を評価できる利点がある．また，Bergstra ら [9] は，自動的にハイパーパラメータを調整するためのライブラリである Hyperopt[9] を開発している．このライブラリでは，決定木を用いたアルゴリズムである Tree of Parzen Estimators（TPE）[10] などのハイパーパラメータ探索手法を簡単に利用し，自動でハイパーパラメータを調整することができる．

6.2 深層学習によるソースコード解析

近年，ソースコード分類やコードクローン検出などのソースコード解析に深層学習を応用する研究が多く発表されている．

White ら [14] は，再帰型ニューラルネットワークとオートエンコーダを使用して AST のベクトル化を行い，各 AST ベクトルの l2 ノルムを計測することによって，コードクローン検出を行う手法を提案している．Gu ら [15] は，機械翻訳にも使用されているディープラーニングモデルである，RNN Encoder-Decoder モデルを使用し，自然言語のクエリから API の使用順序例を生成する “API Learning” を提案している．Zhao と Huang[16] は，ソースコードの制御フローグラフとデータフローグラフを深層学習を用いてベクトルに変換し，機能的な類似性を測定する DeepSim という手法を提案し，この手法をコードクローン検出に適用している．以上の通り，深層学習を用いて様々なソースコード解析に関する研究が行われている．

また，深層学習をソースコード分類に応用した研究も存在する．Zhang ら [1] 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）[17] 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によって AST をベクトル化する 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とコードクローン検出に適用している．

本研究で利用したソースコード分類手法は，以上の既存手法と異なり，Bag of Words や word2vec[18] などの，主に自然言語に対して適用される技術を利用していないことが特徴の 1 つである．

7. まとめと今後の課題

本研究では，深層学習を用いたソースコード分類タスク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．この手法を用いると，効率よく不均衡データの問題に対処することができる．提案手法では，ソースコード分類器モデルを作成した後，学習用ソースコードを用いてそのモデルの分類精度を検証し，正しく分類できなかった学習用ソースコードに対してミューテーションを行い，作成した類似ソースコードを学習用データセットに追加する．このように，検証結果に基づいて動的に学習用データセットを再構築することで，使用する深層学習モデルや解決したい問題に対してデータセットを最適化することができきる．

評価実験では，グラフ畳み込みネットワークとオープンソースソフトウェアを使用し，本手法を含む 3 つのデータセット構築手法で構築したデータセットを用いてソースコード分類器モデルを訓練し，分類精度を比較した．その結果，本手法を用いて構築したデータセットで訓練したモデルは，最も高い精度でソースコードを分類できることが確認できた．

今後の課題として以下の点を挙げることができる．

- モデル指向型アプローチと本手法を比較する必要がある．
- 学習させる類似ソースコードセットの数を増やすことで，より大規模なデータセットに対する本手法の有用性を評価する必要がある．
- 既存のソースコード分類手法に対して本手法を適用し，有効性を評価する必要がある．
- ソースコード分類以外のタスクに対して本手法を適用することで，一般的な深層学習タスクに対する本手法の有効性を評価する必要がある．

参考文献

[1] Zhang, J., Wang, X., Zhang, H., Sun, H., Wang, K. and Liu, X.: A Novel Neural Source Code Representation based on Abstrace Syntax Tree, *Proc. of ICSE 2019*, pp. 783–794 (2019).

[2] Chen, C. and Shyu, M.-L.: Clustering-based binary-class classification for imbalanced data sets, *Proc. of IRI 2011*, pp. 384–389 (2011).

[3] Yan, Y., Chen, M., Shyu, M.-L. and Chen, S.-C.: Deep learning for imbalanced multimedia data classification, *Proc. of ISM 2015*, pp. 483–488 (2015).

[4] Chawla, N. V., Japkowicz, N. and Kotcz, A.: Editorial: Special Issue on Learning from Imbalanced Data Sets, *SIGKDD Explor. Newsl.*, Vol. 6, No. 1, pp. 1–6 (2004).

[5] Kawaguchi, S., Garg, P. K., Matsushita, M. and Inoue, K.: Mudablue: An automatic categorization system for open source repositories, *Journal of Systems and Software*, Vol. 79, No. 7, pp. 939–953 (2006).

[6] Schlichtkrull, M., Kipf, T. N., Bloem, P., Van Den Berg, R., Titov, I. and Welling, M.: Modeling relational data with graph convolutional networks, *Proc. of ESWC 2018*, pp. 593–607 (2018).

[7] Roy, C. K. and Cordy, J. R.: A mutation/injection-based automatic framework for evaluating code clone detection tools, *Proc. of ICSTW 2009*, pp. 157–166 (2009).

[8] Kipf, T. N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks, *Proc. of ICLR 2017* (2017).

[9] Bergstra, J., Yamins, D. and Cox, D. D.: Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms, *Proc. of SCIPY 2013*, pp. 13–20 (2013).
[10] Bergstra, J. S., Bardenet, R., Bengio, Y. and Kégl, B.: Algorithms for hyper-parameter optimization, *Proc. of NIPS 2011*, pp. 2546–2554 (2011).
[11] Kingma, D. P. and Ba, J.: Adam: A method for stochastic optimization, *Proc. of ICLR 2015* (2015).
[12] Yan, Y., Liu, Y., Shyu, M.-L. and Chen, M.: Utilizing concept correlations for effective imbalanced data classification, *Proc. of IRI 2014*, pp. 561–568 (2014).
[13] Bergstra, J. and Bengio, Y.: Random search for hyper-parameter optimization, *Journal of Machine Learning Research*, Vol. 13, No. Feb, pp. 281–305 (2012).
[14] White, M., Tufano, M., Vendome, C. and Poshyvanyk, D.: Deep learning code fragments for code clone detection, *Proc. of ASE 2016*, pp. 87–98 (2016).
[15] Gu, X., Zhang, H., Zhang, D. and Kim, S.: Deep API learning, *Proc. of FSE 2016*, pp. 631–642 (2016).
[16] Zhao, G. and Huang, J.: DeepSim: deep learning code functional similarity, *Proc. of FSE 2018*, pp. 141–151 (2018).
[17] Tang, D., Qin, B. and Liu, T.: Document modeling with gated recurrent neural network for sentiment classification, *Proc. of EMNLP 2015*, pp. 1422–1432 (2015).
[18] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J.: Distributed representations of words and phrases and their compositionality, *Proc. of NIPS 2013*, pp. 3111–3119 (2013).