# 深層学習を用いたソースコード分類に関する研究

提出先　大阪大学大学院情報科学研究科

提出年月　2022 年 1 月

藤 原　裕 士

# 論文一覧

## 主要論文

1. Yuji Fujiwara, Norihiro Yoshida, Eunjong Choi, Katsuro Inoue: "Code-to-Code Search Based on Deep Neural Network and Code Mutation", Proceedings of the 13th International Workshop on Software Clones (IWSC 2019), pp.1-7, Hangzhou, China, February 2019.（国際会議録）
2. 藤原裕士，崔恩瀞，吉田則裕，井上克郎，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，電子情報通信学会論文誌 D 学生論文特集，Vol.J104-D，No.04，pp.275-284，2021 年 4 月．（学術論文）
3. 藤原裕士，崔恩瀞，吉田則裕，井上克郎，深層学習を用いたソースコード分類手法の比較調査，電子情報通信学会論文誌 D，Vol.J104-D，No.08，pp.622-635，2021 年 8 月．（学術論文）
4. 藤原裕士, 森彰, 井上克郎, 回帰モデルを用いたコードクローン検出手法の提案と汎化性能の評価, 電子情報通信学会論文誌 D, Vol.J104-D, No.09, pp.678-689, 2021 年 9 月．（学術論文）

## 関連論文

1. Yoriyuki Yamagata, Fabien Herve, Yuji Fujiwara and Katsuro Inoue: "Finding repeated strings in code repositories and its applications to code-clone detection", Proceedings of the 28th Asia-Pacific Software Engineering Conference (APSEC 2021)（採録決定）
2. 崔恩瀞，藤原裕士，吉田則裕，水野修: コードクローン検索手法の調査，コンピュータソフトウェア（採録決定）

# 内容梗概

ソフトウェア開発の規模が大きくなるにつれて，ソフトウェア開発の効率化が求められる．ソフトウェア開発を効率よく行うために，開発者はソースコード分類技術を利用することができる．ソースコード分類とは，予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タスクである．ソースコード分類技術を用いることで，開発者は，類似した機能を持つソースコードの検索や，開発中のソフトウェアに必要な機能の検索，開発中のソフトウェアに混入しやすいバグの予測などを行うことができる．このように，ソースコード分類技術を用いることで，ソフトウェアの開発や保守を効率的に行うことができる．

また近年，深層学習を用いて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．既存手法の評価実験では，深層学習を用いることでソースコードの文字列情報や構文情報だけでなくソースコードが実現する機能を捉えられることが示されている．

しかし，既存の深層学習を用いたソースコード分類手法の問題点として，以下の点が挙げられる．

1. 学習データの数はソースコード分類精度に大きな影響を与えるが，深層学習を用いたソースコード分類の既存手法では，学習用データセットを構築する際に，特に説明なくランダムサンプリングが用いられていることが多く，そのデータセットが学習に最適かどうか不明である．
2. 既存手法の評価実験において，学習に使用するニューラルネットワークやソースコード表現の比較検討が不十分であり，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組み合わせが不明である．
3. 深層学習を用いたコードクローン検出では，深層学習モデルに学習させていない 2 つの入力ソースコードがコードクローンかどうかを判定するため，汎化性能が求められるが，既存手法の評価実験では汎化性能の評価が行われていない．

本論文では最初に，1. で挙げた問題点を解決するために，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．この手法では，ソースコード分類を行う深層学習モデルの検証において正答率が相対的に低いクラスに学習データを追加する．この手法を用いることで，学習データを追加したクラスの

正答率が上昇し，低下していた深層学習モデルのソースコード分類精度が向上することを確認した．

次に，2. で挙げた問題点を解決するために，深層学習を用いたソースコード分類手法を 6 種類作成し，それらの手法のソースコード分類精度を比較した．その結果，再帰型ニューラルネットワークにソースコードにトークン列を学習させる手法の分類精度が最も高いことが分かった．

最後に，3. で挙げた問題点を解決するために，依存関係のない 5 種類のデータセットを用いて，深層学習を用いたコードクローン検出手法の汎化性能の評価を行った．また，深層学習を用いたコードクローン検出手法で一般的に利用される 2 値分類モデルの代わりに回帰モデルを利用することで，既存手法の汎化性能が向上することを確認した．

本論文の研究成果を用いることにより，類似した機能を持つソースコードの検索などに応用可能な技術であるソースコード分類技術の精度を高めることが可能になり，ソフトウェア開発や保守をより効率よく行うことができる．

# 謝辞

本研究を遂行するにあたり，日頃より適切なご指導や激励を賜りました，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，心から深く感謝申し上げます．

本論文を執筆するにあたり，適切なご助言とご指導を頂きました，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 増澤 利光 教授，ならびに 楠本 真二 教授に，心から感謝致します．

本研究の遂行ならびに本論文の執筆にあたり，日頃より適切なご助言とご指導を頂きました，名古屋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 吉田 則裕 准教授，ならびに 京都工芸繊維大学情報工学・人間科学系 崔 恩瀞 助教に，心から感謝申し上げます．

本研究を遂行するにあたり，貴重なご助言とご指導を頂きました，産業技術総合研究所 森 彰 氏に，深く感謝申し上げます．

本研究を行うにあたり，様々なご助言ご指導を頂きました，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，春名 修介 特任教授，神田 哲也 助教に，深く御礼申し上げます．

最後に，日々の研究生活にてご助言やご協力頂いた井上研究室の皆様に，厚く御礼申し上げます．

# 目次

# 図目次

# 表目次

# 第 1 章

# はじめに

## 1.1　ソフトウェア再利用と保守

現代社会においてソフトウェアが担う分野は拡大の一途を辿っており，多種多様で大規模なソフトウェアを短期間かつ低コストで開発することに関する要求が高まっている．ソフトウェア工学とは，ソフトウェアの開発や運用，保守に体系的・専門別・定量的なアプローチを適用すること，およびそのアプローチについて研究することである [1,31]．前述の要求を満たすために，ソフトウェア工学の研究は盛んに行われている．

ソフトウェア開発の生産性を向上させるための取り組みのひとつに，ソフトウェア再利用がある．ソフトウェア再利用とは，新たなソフトウェアの開発に既存のソフトウェア部品を利用することである [6,31]．ソフトウェア再利用を行うことで，システムに利用するソースコードの分析や設計，記述，テストなどにかかるコストを削減することができるなどのメリットがある．しかし，ソフトウェアを再利用するには再利用可能なソフトウェアを見つける必要がある．大量の既存ソフトウェアから再利用可能なソフトウェアを手作業で探すのは膨大なコストがかかり，そのコストがソフトウェア再利用によって削減可能なコストを超えてしまうと，ソフトウェア再利用によるメリットが失われる．そのため，再利用可能なソフトウェアを自動で探すための様々な手法が盛んに研究されている．

ソフトウェアの再利用や検索を容易にするための手法のひとつにソースコード分類がある．ソースコード分類とは，ある基準に基づいて，類似した特徴を持つソースコードを同じクラスに分類する手法である．ソースコード分類で用いられる基準は様々であり，現在までに取り組まれているソースコード分類の研究は，記述言語による分類 [32,62]，コンポーネント間の依存関係による分類 [69]，ソフトウェアの機能性による分類 [33]，ファイル単位での機能性による分類 [62]，関数単位での機能性による分類 [12,45,72] などが存在する．ソースコード分類を検索に活用している例として，大規模なオープンソースソフトウェアリポジトリなどが挙げられる．例えば

SourceForge[*1]では，登録されているソースコードが 21 種類の機能カテゴリに分類されてタグ付けが行われており，その他にも，OS やライセンス，自然言語，プログラミング言語，開発状況に関するタグが付けられている．開発者はこのタグを用いることで，開発者が要求する条件に当てはまるソースコードの検索を行うことが可能である．また，特にソフトウェアの機能性に基づくソースコード分類には，以下のような活用方法がある．

- **開発中のソフトウェアに必要な機能の調査**
  開発中のソフトウェアにどのような機能を実装すべきか考える際に，開発中のソフトウェアに類似した機能を持つソースコードを参考にできる [33, 41].
- **開発中のソフトウェアに混入しやすいバグの予測**
  開発中のソフトウェアと同じ機能を持つソフトウェアの多くに共通して含まれる傾向にあるバグについて調査することで，開発中のソフトウェアで発生する可能性のあるバグについて予測することができる [41, 64, 76].
- **機能性に基づいたコードクローン分析**
  同じ機能のクラスに分類されたソースコードは，互いに機能が類似したコードクローンであると見なすことが可能である．そのため，ソースコード分類技術を用いてコードクローン分析を行うことができる．コードクローン分析手法は，類似機能をもつコードクローン検索に利用可能なだけでなく，後述するソフトウェア保守の支援を行うことができる手法のひとつでもある．

以上のように，ソースコード分類はソフトウェア開発工程の効率化に役立つといえる．

ソフトウェアを開発し，ソフトウェアを運用する段階に入ると，ソフトウェア保守を実施する必要がある．ソフトウェア保守とは，ソフトウェアが開発され，運用されはじめてから，その機能と性能を維持するために要求される活動の履行である [31, 47]．ソフトウェア保守は，その目的により以下の 4 つに分類されている [58].

**修正保守**　ソフトウェアの運用が開始した後に発見されたエラーの修正.

**適応保守**　ソフトウェアの外部環境が変化した場合に，その変化にソフトウェアを対応させるためのソフトウェア修正.

**完全化保守**　ソフトウェアの機能拡張や演算効率の改善を伴う修正.

**予防保守**　ソフトウェアの保守性を向上させ，将来のエラー修正や機能拡張などを実行しやすくするためのソフトウェア修正.

ソフトウェアを継続して運用するためにはソフトウェア保守を行う必要があるが，ソフトウェア保守を行う上では様々な問題がある [40]．そのため，ソフトウェア保守作業の支援を行うための手法が盛んに研究されている．

[*1] https://sourceforge.net/

ソフトウェア保守を困難にする要因のひとつとして考えられているのが，コードクローンの存在である [17,23]．コードクローンとは，互いに一致または類似したコード片のことである [52]．例えば，あるコード片に欠陥が存在することを確認した場合，そのコード片のコードクローンにも欠陥が存在すると考えられる．そのため，ソフトウェア保守の担当者は，欠陥が存在するコード片を修正した後，そのコード片のコードクローンを全て確認し，同様の修正を行う必要がある．しかし，大規模ソフトウェアの保守を行う場合，全てのコードクローンを手作業で見つけることは困難である．そのため，ソフトウェアの中から自動で速く正確にコードクローンを検出するための研究が行われている．

近年では，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を行う研究が盛んである [12,21,38,45,46,53,63,72,74]．深層学習を用いることで，ソースコードの構文だけでなくソースコードの意味を捉えたソースコード分類やコードクローン検出を行うことができ，ソフトウェア再利用や保守をより効率的に行うことができると考えられている．以降，本章では，深層学習および深層学習を用いたソースコード分類手法について説明したうえで，既存研究の問題点と本論文の構成について述べる．

## 1.2 深層学習

深層学習の定義には様々なものがある．以下に Deng と Yu が文献 [10] にまとめた深層学習の代表的な定義を示す．

def.1: 教師ありまたは教師なしの特徴抽出と変換およびパターン分析と分類のために，非線形情報処理の多くの層を利用する機械学習技術のクラス．

def.2: データ間の複雑な関係をモデル化するために，複数のレベルの表現を学習するアルゴリズムに基づく機械学習のサブ分野．上位の特徴や概念は下位の特徴で定義され，そのような特徴の階層は深層アーキテクチャと呼ばれる．これらのモデルの多くは教師なしの表現学習に基づいている．

def.3: 特徴や要因，概念のヒエラルキーに対応する，複数のレベルの表現の学習に基づく機械学習のサブ分野で，下位レベルの概念から上位レベルの概念が定義され，同じ下位レベルの概念が多くの上位レベルの概念の定義に役立つ．

def.4: 深層学習とは，機械学習において，異なる抽象度に対応した複数のレベルでの学習を試みるアルゴリズム群のことである．典型的には人工ニューラルネットワークを用いる．これらの学習された統計モデルのレベルは，明確な概念のレベルに対応しており，下位レベルの概念から上位レベルの概念が定義され，同じ下位レベルの概念が多くの上位レベルの概念の定義に役立つ．

def.5: 深層学習は機械学習の研究の新しい分野であり，機械学習を本来の目的のひとつである人工知能に近づける目的で導入されたものである．深層学習とは，画

像，音声，テキストなどのデータの意味を理解するのに役立つ，複数のレベルの表現と抽象化を学習することである．

また，Deng と Yu は，上記の様々な深層学習の定義には 2 つの重要な側面があると述べている．以下にその重要な側面を示す．

(1) 非線形情報処理の複数の層またはステージからなるモデル
(2) より高位で抽象的な層での特徴表現の教師ありまたは教師なしでの学習方法

深層学習による計算機構を表現する際に，深層学習モデルという名詞がしばしば用いられる．一般的に深層学習モデルは，重要な側面 (1) の意味で用いられている．また，def.4 によると，深層学習は典型的にニューラルネットワークを用いるとされており，ニューラルネットワークの場合，1 層以上の隠れ層を利用することで非線形情報の処理を行うことが可能である [9]．そのため，上記の深層学習の定義や重要な側面に基づき，本論文では，深層学習モデルを以下のように定義する．

**定義 1** 隠れ層を 1 層以上持つニューラルネットワークが 1 つ以上組み合わされることで構成された，非線形情報の処理を目的とした計算を行うオブジェクト

深層学習には，教師あり学習と教師なし学習の 2 種類がある [3]．教師あり学習は，ラベル付きの学習データに基づいて深層学習モデルの内部パラメータを調整し，ラベルのない新しいデータに対してラベルを予測する手法である．そのため，入力値とそれに対応する出力値があらかじめ分かっている学習用データセットを用いて深層学習モデルの学習を行う．このとき，深層学習モデルに入力する変数を説明変数，深層学習モデルから出力される変数を目的変数という．また，教師あり学習モデルには，分類モデルと回帰モデルの 2 種類がある．分類モデルは，入力されたデータがどのクラスに属するかを予測する深層学習モデルであり，その中でも分類先のクラスが 2 通りである分類モデルを，本論文では 2 値分類モデルと呼ぶ．回帰モデルは，データが入力されたときに，そのデータに対応する実数を予測する深層学習モデルである．次節以降で述べる，深層学習を用いてソースコード分類を行う既存手法の多くは，教師あり学習の分類モデルを用いている．

教師なし学習は，入力されたデータセット自体の関係性に基づいて学習を行い，データのパターンを識別する手法である．そのため教師あり学習とは異なり，目的変数が不要である．

また，学習済みの深層学習モデルにおいて，学習用データセットと依存関係のない独立したデータセットに対する性能を汎化性能という．一般的に，深層学習モデルの汎化性能は，依存関係があるデータセットで評価した場合の性能と比べて劣化することが知られている [11]．また，1.4 節でも述べるが，深層学習を用いたコードクローン検出の既存研究では，汎化性能を評価していないという問題点がある．

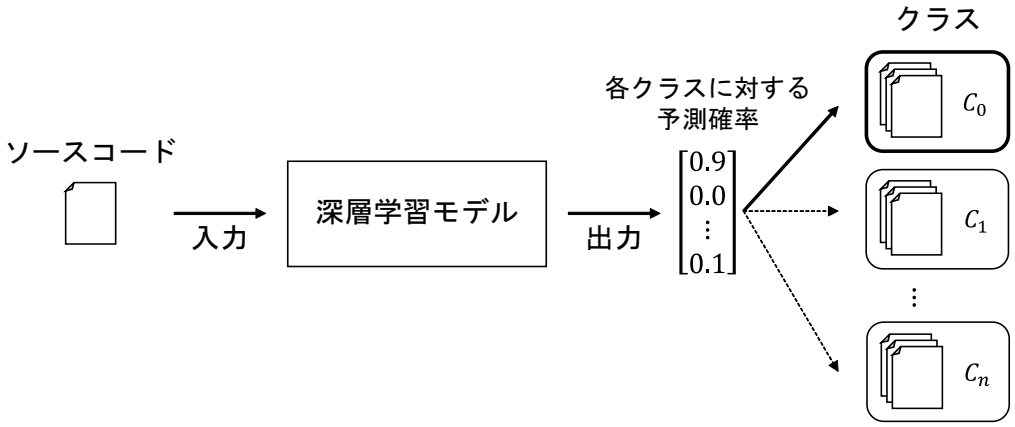


図 1.1: 深層学習を用いたソースコード分類の概要

## 1.3 深層学習を用いたソースコード分類

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 [12, 21, 38, 45, 46, 53, 63, 72, 74]．これらの手法では，深層学習モデルを用いてソースコードの構文や意味の類似性を捉え，入力ソースコードを自動で分類する．

Mou ら [45] は提案手法の評価実験で，教育プログラミングのオープンジャッジデータセット（The dataset of a pedagogical programming open judge system，以下，OJ データセット）を使用してソースコード分類精度の評価を行った．OJ データセットは 104 の異なるプログラミングの設問に対する回答ソースコードから構成されている．このデータセットを用いたソースコード分類の評価実験では，回答ソースコードの 8 割を使って深層学習モデルの学習と検証を行い，残りの 2 割のソースコードがどのプログラミングの設問に対する回答かを深層学習モデルに予測させるという分類問題を解かせることで，提案手法のソースコード分類精度を評価した．このソースコード分類問題ではプログラミングの設問を予測しているが，例えば代わりにソースコードの機能タグを予測させることで，1.1 節で述べたソースコードの機能性による分類を行うといった応用が可能であると考えられる．

そこで本論文では上記の問題を一般化し，本論文で扱うソースコード分類問題を以下のように定義する．

**定義 2** 既存ソースコードを，類似した特徴を持つもの毎にあらかじめ分割して作成したクラス $C_0 \ldots C_n$ に対して，入力として与えられたソースコードを，入力ソースコードに類似した特徴を持つ既存ソースコードが属するクラス $C_i (0 \leq i \leq n)$ に自動で分類する

深層学習を用いたソースコード分類の概要を図 1.1 に示す．分類したいソースコー

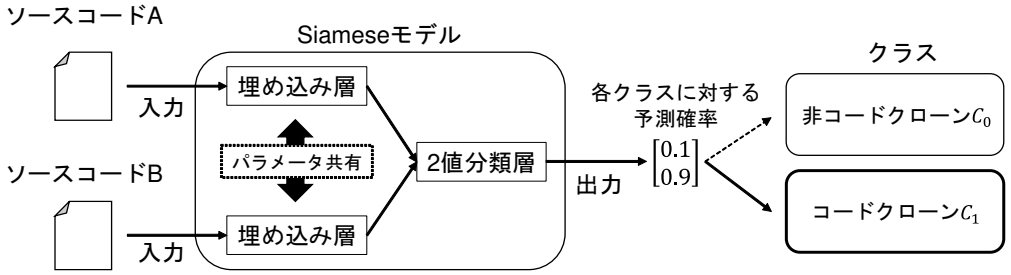


図 1.2: 深層学習を用いたコードクローン検出の概要

ドを深層学習モデルに入力すると，深層学習モデルは各クラスに対する予測確率を出力し，予測確率が最も高いクラスにソースコードを分類する．図 1.1 の例ではクラス $C_0$ に対する予測確率が 0.9 と最も高いため，入力ソースコードはクラス $C_0$ に分類される．このソースコード分類に用いられる深層学習モデルの説明変数は入力ソースコードであり，目的変数は各クラスに対する予測確率である．そのため，この深層学習モデルの学習では，クラス $C_i(0 \leq i \leq n)$ に属している学習用のソースコードを入力したときのクラス $C_i$ に対する予測確率が最も高くなるように，深層学習モデルの内部パラメータが調整される．このように学習が進むと，クラス $C_i(0 \leq i \leq n)$ に属している学習用のソースコードと類似した特徴を持つソースコードを入力したときに，深層学習モデルは入力ソースコードをクラス $C_i$ に分類できるようになる．

次に，ソースコード分類の定義に基づいた深層学習によるコードクローン検出の概要を図 1.2 に示す．ここで概要を説明するコードクローン検出手法は 1.1 節で説明したコードクローン分析とは異なり，2 つのコード片がコードクローンかどうかを 2 値分類モデルを使って判定する手法である．そのため，ここで概要を説明するコードクローン検出手法は，深層学習を用いたソースコード分類のスペシャルケースである．深層学習を用いたコードクローン検出手法は，分類先のクラスが非コードクローンのクラス $C_0$ とコードクローンのクラス $C_1$ の 2 つであり，ソースコードのペアを入力したときに，そのペアがコードクローンでなければクラス $C_0$ に分類し，そのペアがコードクローンであればクラス $C_1$ に分類する．図 1.2 の深層学習モデルは Siamese モデル [7] と呼ばれており，深層学習を用いたコードクローン検出に取り組んだ既存手法で多く利用されている [21, 46, 53, 72]．Siamese モデルは，内部パラメータを共有する 2 つの埋め込み層と，1 つの 2 値分類層から構成される．埋め込み層は，ソースコードが入力されるとソースコードの特徴ベクトルを出力するニューラルネットワークの層である．2 つの埋め込み層は共通のパラメータを持つため，同じソースコードが入力された場合に同じベクトルを生成する．2 値分類層は，2 つの特徴ベクトルが入力されると非コードクローンのクラス $C_0$ とコードクローンのクラス $C_1$ に対する予測確率を出力するニューラルネットワークの層である．図 1.2 の例では，クラス $C_0$ に対する予測確率が 0.1 なのに対し，クラス $C_1$ に対する予測確率が 0.9 と高いため，Siamese モ

デルは入力されたソースコード A，B をコードクローンであると判定したことになる．

## 1.4　既存手法の問題点

深層学習を用いてソースコード分類を行う既存手法の問題点として，以下の 3 つが考えられる．

**問題点 1 深層学習を用いたソースコード分類の既存研究では，データの分布に関する記述が少なく，特に説明なくランダムサンプリングが用いられていることが多い．**
学習データの数は，ソースコード分類精度に大きな影響を与える．しかし，既存研究では，ランダムサンプリング以外の手法についての言及が無い．そのため，ソースコード分類における不均衡データの効率的な学習において，ランダムサンプリングが最適かどうか不明である．また，ランダムサンプリングは静的な手法である．本論文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，深層学習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．一般的に深層学習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．

**問題点 2 ニューラルネットワークやソースコード表現の比較検討が不十分である．**
Saini ら [53] は評価実験で，順伝播型ニューラルネットワーク [57] を用いた提案手法と，深層学習を用いない既存手法の精度を比較している．しかし，深層学習を用いる他の手法との精度比較が行われていない．また，Zhang ら [72] は，ソースコードの抽象構文木を学習する手法として ASTNN を提案し，評価実験の比較対象として，長・短期記憶再帰型ニューラルネットワーク [19] にトークン列を学習させる手法や，グラフニューラルネットワーク [39] に制御フローグラフを学習させる手法を選択している．しかし，長・短期記憶再帰型ニューラルネットワークに抽象構文木の深さ優先探索順列を学習させる手法や，グラフニューラルネットワークに抽象構文木を学習させる手法など，他にも考えられるニューラルネットワークとソースコード表現の組み合わせが存在するが，結果は記載されていない．以上のような例があるため，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組み合わせについての調査を進める必要があると考えられる．

**問題点 3 深層学習を用いたコードクローン検出手法において，深層学習モデルの汎化性能を評価していない場合が多い．**
深層学習を用いたソースコード分類やソースコード検索のような，学習させたソースコードと入力ソースコードの類似性を判定するタスクとは異なり，深層学習を用いたコードクローン検出では，深層学習モデルに学習させていない 2 つの入力ソースコードがコードクローンかどうかを判定する．そのため，コードクローン検出を目的とし

た深層学習モデルには汎化性能が求められる．しかし，既存研究の評価実験では，同一データセットを分割することによって学習用データセットと評価用データセットを構築していることが多い．この場合，学習用データセットと評価用データセットの間に依存関係があるため，評価実験におけるコードクローン検出精度が良い場合でも，深層学習モデルがデータセットに特化している可能性を否定できず，深層学習モデルの汎化性能についての評価が不十分であると考えられる．

本論文では，上記の 3 つの問題点を解決する．最初に，問題点 1 を解決するために，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．この手法は，深層学習モデルの学習結果を用いた動的な学習用データセットの改善を通して，より高い精度の深層学習モデルの作成を目的とする．次に，問題点 2 を解決するために，深層学習を用いたソースコード分類手法の比較調査を実施した．この調査は，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組み合わせを明らかにすることを目的とする．最後に，問題点 3 を解決するために，学習用データセットと評価用データセットをそれぞれ別のデータセットから作成し，深層学習を用いたコードクローン検出手法の評価実験を行った．このような評価実験を行うことで，深層学習モデルの汎化性能をより正確に評価することができる．また，この評価実験を行うと同時に，深層学習モデルの汎化性能を向上させるための手法の提案を行った．

## 1.5　本論文の構成

以降，2 章では，本論文の背景について述べる．

3 章では，深層学習を用いたソースコード分類手法の導入として，順伝播型ニューラルネットワークとコードミューテーションを用いたコードクローン検索を行う手法について説明する．この手法は，深層学習を用いたソースコード分類手法をコードクローン検索に適用した手法であり，コードミューテーションを用いるため，元の学習データ数が少ない場合に学習データを増やすことができる．評価実験では，3 種類のオープンソースソフトウェアから構文的コードクローンや意味的コードクローンを収集して作成したベンチマークを利用し，検索精度を評価した．その結果，3 章で説明するコードクローン検索手法は，構文的な類似度が高いコードクローンの検索精度は高いが，構文的な類似度が低いコードクローンの検索精度は低いことが分かった．さらに，ミューテーションの影響に関する評価実験を行った結果，学習に用いるソースコードの数がコードクローンの検索精度に大きく影響することが分かった．

4 章では，3 章で判明した “学習に用いるソースコードの数がソースコード分類精度に与える影響” に着目し，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．この手法では，深層学習モデルの学習を行った

後，学習済み深層学習モデルの評価用データセットに対するソースコード分類精度の検証を行い，正答率が相対的に低いクラスに対し，そのクラスのソースコードに対してミューテーションを適用して作成したコードクローンを追加する．評価実験では，オープンソースソフトウェアのソースコードを用いて作成した意味的コードクローンデータセットに対して提案手法を適用した．その結果，コードクローンを追加したクラスの正答率が上昇し，低下していた深層学習モデルのソースコード分類精度が向上することを確認した．4 章で提案する手法によって，1.4 節で述べた問題点 1 を解決した．

5 章では，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組み合わせについて調査するために，大規模コードクローンベンチマーク BigCloneBench を用いて，深層学習を用いたソースコード分類手法の分類精度を比較した．その結果，再帰型ニューラルネットワークにソースコードのトークン列を学習させる手法の分類精度が最も高いことが分かった．また，深層学習を用いたソースコード分類手法と深層学習を用いないソースコード分類手法のソースコード分類精度を比較した．その結果，深層学習を用いた手法のほうが分類精度が高いことが分かり，深層学習はソースコード分類に有効であることが分かった．4 章で行った比較実験によって，1.4 節で述べた問題点 2 を解決した．

6 章では，回帰モデルを用いたコードクローン検出手法を提案した．この手法では，コードクローンの学習方法を改善するために，2 値分類モデルではなく回帰モデルを利用し，教師あり学習に使用する目的変数をコード片の類似度に変更する．このように学習させるコードクローンの重みを類似度に基づいて変更することで，コードクローンの類似度情報を利用した学習を行う．また，評価実験では，意味的コードクローンを含む 5 種類のデータセットを用いて，学習データに対する深層学習モデルの挙動を確認し，深層学習モデルの汎化性能を評価した．具体的には，学習用データセットと評価用データセットを同一データセットから作成するのではなく，1 種類のデータセットを学習用データセット，残りの 4 種類を評価用データセットとして扱った．そして，4 種類の深層学習モデルについて，2 値分類モデルから回帰モデルに変更し，変更前モデルと変更後モデルの汎化性能を比較した．その結果，4 つの内 3 つの深層学習モデルの汎化性能が向上することが分かった．6 章で提案した手法と実施した評価実験によって，1.4 節で述べた問題点 3 を解決した．

最後に 7 章で，本論文のまとめと，今後，本論文で述べた研究を発展させる場合に考えられる研究方針について述べる．

# 第 2 章

# 背景

## 2.1 コードクローン

コードクローンとは，互いに一致または類似したコード片のことである．1.1 節で述べたように，コードクローンの存在はソフトウェアの保守を困難にする要因であると言われている [17]．コードクローンの主な発生要因として，既存ソースコードのコピーアンドペーストによる再利用，定型処理（イディオム）による発生，ツールによるソースコード自動生成，偶然の一致による発生などが挙げられる [4]．Roy らはコード片の違いの程度に基づき，コードクローンを以下の 4 タイプに分類している [52].

**タイプ 1** 空白やタブの有無，コーディングスタイル，コメントの有無などの違いを除き完全に一致するコードクローン

**タイプ 2** タイプ 1 の違いに加え，識別子名，変数の型などが異なるコードクローン

**タイプ 3** タイプ 2 の違いに加え，文の挿入や削除，変更などが行われているコードクローン

**タイプ 4** 類似した処理を実行するが，構文上の実装が異なるコードクローン

さらに Roy らは，コード片の構文的な類似度を定義し，コードクローンのタイプ分類を構文的な類似度に基づいて細分化した [60]．2 つのコード片の識別子名とリテラル値を正規化した後に抽出した行の集合を $l1$,$l2$ とし，$l1$,$l2$ 内で共通している行の集合を $l1 \cap l2$，集合 $l$ の要素数を $|l|$ としたとき，Roy らが定義した構文的な類似度（$Similarity$）は以下のとおりである．

$$Similarity(l1, l2) = \frac{2 * |l1 \cap l2|}{|l1| + |l2|}$$

タイプ 1，2 のコードクローンは，正規化を行ったあとの文字列が等しいため，構文的な類似度は 1.0 である．また，タイプ 3 は構文的に類似したコードクローンで，タイプ 4 は構文的に類似していないコードクローンだが，Roy らはタイプ 3 の構文的類似

度の最小値を定義することによるタイプ 3 とタイプ 4 の線引きは難しいと考えた．そのため彼らは，タイプ 3 とタイプ 4 のコードクローンを．“構文的な類似度が 0.7 以上のタイプ 3 クローン”，“構文的な類似度が 0.5 以上 0.7 未満のタイプ 3 クローン”，“構文的な類似度が 0.5 未満のタイプ 3 クローンおよびタイプ 4 クローン” の 3 つのカテゴリに分類した．

本論文では，上記の細分化された分類に基づき，構文的コードクローンを “タイプ 1，2 および構文的な類似度が 0.5 以上のタイプ 3 クローン”，意味的コードクローンを “構文的な類似度が 0.5 未満のタイプ 3 およびタイプ 4 クローン” と定義する．

### 2.1.1 コードクローン検出ツール

コードクローンに対する保守作業を支援するためのツールのひとつにコードクローン検出ツールがある．コードクローン検出ツールは，ソフトウェア中に含まれるコードクローンを自動で検出するためのツールである．本節では，代表的なコードクローン検出ツールとして CCFinder [29] の説明を行う．CCFinder は，トークン単位のコードクローンを検出するツールである．CCFinder の特徴を以下に示す．

- 識別子名や関数名などのの正規化を行うことで，それらの名前が異なるコードクローンを検出することができる．つまり CCFinder はタイプ 2 までのコードクローン検出が可能である．
- メモリ使用量と実行時間の観点で効率が良く，大規模ソフトウェアにも問題なく適用可能である．1,000 万行のソースコードに対して，Pentium3(650MHz) と 1GB のメモリを搭載したコンピュータを使用して 68 分でコードクローン検出が可能である．
- 対応言語は Cobol，C/C++，C#，Java，Visial Basic である[*1]．また，ツールの中でプログラミング言語に依存する部分が，字句解析器と識別子名の正規化，構造体の識別に関する部分であり，それらの言語依存部分を取り換えることで他のプログラミング言語にも対応可能である．

CCFinder によるコードクローン検出の手順は，以下の 4 つのステップから成り立っている．

1. ソースファイルを字句解析し，トークン列に変換する．検出対象が複数ファイルの場合は，各ファイルのトークン列を連結することで，ひとつのトークン列を生成する．
2. 実用的に意味のないトークンを取り除く処理や，型・変数・識別子名を同一トークンに置き換える処理を行う．

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

3. 2. の処理を行ったトークン列から，指定した長さ以上の等価な部分文字列の組をクローンペアとして検出する．
4. 検出されたクローンペアが存在する位置（元のソースファイルの行番号）を出力する．

CCFinder 以外にも様々なコードクローン検出ツールが提案されており，タイプ 2 だけでなくタイプ 3 までのコードクローン検出を目的としたツールも多く提案されている [28,38,50,54]．また近年では，ソースコードの意味を捉えて類似度の低いタイプ 3 クローンやタイプ 4 クローンの検出を行うために，深層学習を用いるコードクローン検出手法が多く提案されている [21,38,53,63,72,74]．

### 2.1.2 コードクローンデータセット

Roy らは，コードクローン検出ツールを評価するための大規模なデータセットとして，BigCloneBench（以降，BCB）[60] を作成した．このデータセットは，様々なオープンソースソフトウェアに存在するコードクローンを収集したものであり，タイプ 1 からタイプ 4 まで，様々な類似度のコードクローンが含まれている．BCB は現在に至るまで様々なコードクローン検出ツールの評価に利用されている．また，深層学習を用いてコードクローン検出を行う研究では，BCB は深層学習モデルの学習用データセットとして利用されることもある．

また，近年では，ソースコード記述以外の情報を利用して意味的コードクローンデータセットを構築することがある．Al-omari ら [2] は大規模 Q&A サイト StackOverflow[*2] に注目した．同じ質問に寄せられた正しく動作する回答ソースコードは互いに意味的コードクローンであるという仮定の下，意味的コードクローンベンチマーク SemanticCloneBenchmark を提案した．Zhao と Huang [74] は，オープンジャッジシステムにおいて，ある設問に対して提出されたコード片はその設問を解決するための機能を持つことに注目した．そして，同じ設問に提出されたコード片のペアは意味的コードクローンであり，互いに異なる設問に提出されたコード片のペアを非コードクローンであるという仮定の下でデータセットを作成し，DeepSim の評価実験に利用した．Kamp ら [30] は，コード片を説明するコメントである JavaDoc の文章類似度が高いコード片のペアは意味的コードクローンであるという仮定の下，意味的コードクローンデータセット SeSaMe を提案した．

### 2.1.3 コードクローン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 [27,51]．一般的に，ミューテーションはテ

[*2] https://stackoverflow.com/

ストケースの評価に用いられている [27]．Roy と Cordy は，ミューテーションを利用してコードクローンを作成することで，コードクローン検出ツールの精度を評価する手法を提案している [51]．この手法では，コードクローン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，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 2.1 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 2.1 (a) の 6 行目の文を削除することで，タイプ 3 コードクローン（図 2.1 (b)）が作成された．

## 2.2 ソースコード分類に用いられる代表的なニューラルネットワーク

### 2.2.1 順伝播型ニューラルネットワーク

順伝播型ニューラルネットワーク（Feedforward Neural Networks, 以降, FNN）[57] とは，ネットワークにループ構造が含まれない標準的なニューラルネットワークであ

```
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 2.1: ミューテーションオペレータ mSDL の例

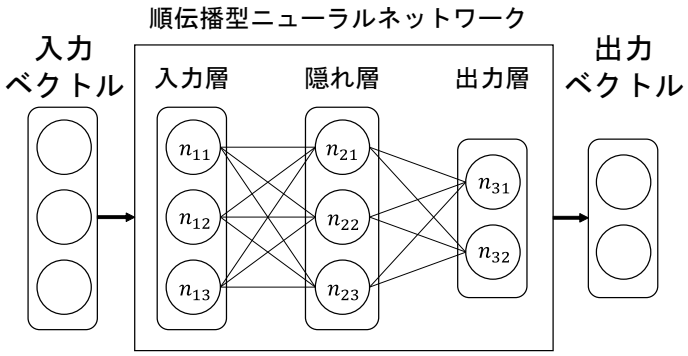


図 2.2: 順伝播型ニューラルネットワークの例

る．最初期は入力層と出力層のみからなる機械学習の手法だったが，隠れ層を増やすことで線形分離不可能な問題を解くことができる深層学習の手法として研究に利用されている．このニューラルネットワークは，ニューロンと呼ばれる，単純なベクトル計算を行うための要素を接続することで構成されている．ソースコードのメトリクスを並べるなどの方法でソースコードをベクトル化することで，FNN をソースコード分類に利用することができる [12, 38, 46, 53, 74]

図 2.2 は 8 個のニューロン $n_{11}$〜$n_{32}$ で構成される，3 層の FNN の例である．FNN の入力と出力はともにベクトルであり，そのベクトルの次元はネットワーク構造に依存する．図 2.2 の例では入力が 3 次元，出力が 2 次元のベクトルである．また，ネットワークの機能は，各ニューロン間の接続に設定されている重みとバイアスと呼ばれる，学習によって調整される値に依存する．FNN の学習では，入力ベクトルと出力ベクトルの組を FNN に与え，入力ベクトルを FNN に与えた際に対応する出力ベクトルが出力されるように FNN の内部パラメータを調整する．これにより，FNN は入力ベクトルと出力ベクトルを対応づけることが可能になる．

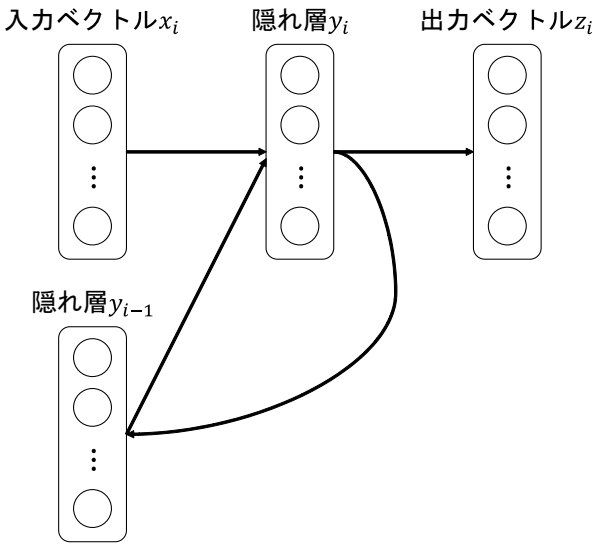


図 2.3: 再帰型ニューラルネットワークの例

### 2.2.2 再帰型ニューラルネットワーク

再帰型ニューラルネットワーク（Recurrent Neural Networks，以降，RNN）[57] とは，ベクトルの系列が入力として与えられ，入力ベクトルの値だけでなく，入力ベクトルの順番にも出力が影響されるニューラルネットワークである．ソースコードはトークン列などの系列で表現することが可能なため，RNN はソースコード分類に用いられている [21, 63, 72].

RNN の例を図 2.3 に示す．この図から分かるように，RNN は FNN と違いネットワークにループ構造が含まれている．RNN では入力ベクトル系列に含まれるベクトルが 1 つずつ順に入力される度に計算が行われる．RNN における $i$ 回目 $(1 \leq i \leq n)$ の計算では，$i$ 番目のベクトル $x_i$ と同時に，$i-1$ 番目のベクトルを入力した後のニューラルネットワークの隠れ層 $y_{i-1}$ が RNN に入力される．この 2 つの入力を用いて RNN の隠れ層 $y_i$ と出力 $z_i$ の計算が行われる．このような手順で計算が行われるため，1〜$i$ 番目のすべての入力ベクトルの値や入力順序が RNN の出力に影響する．

また，代表的な RNN の 1 つに，LSTM（長・短期記憶再帰型ニューラルネットワーク，Long-Short Term Memory recurrent neural network）[19] がある．LSTM は RNN の隠れ層を LSTM block に置き換えることによって，一般的な RNN と比べてさらに長期的な依存関係の学習が可能になったニューラルネットワークである．

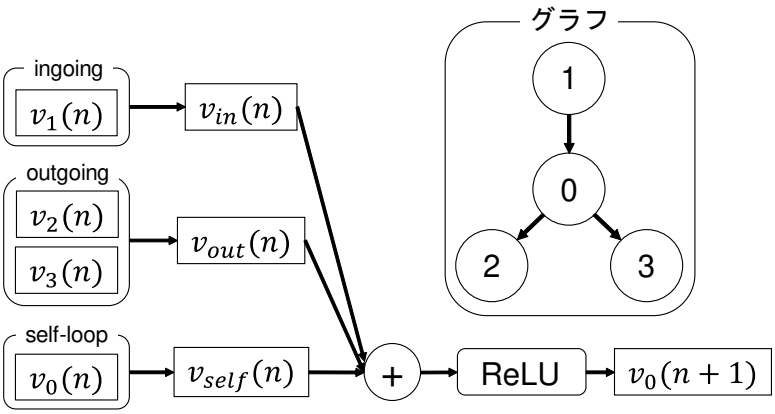


図 2.4: GCN の畳み込み層の例

### 2.2.3 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 Networks，以降，GCN）[56] 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．ソースコードは AST などのグラフで表現することができるため，GCN はソースコードの分類に利用されている [21]．GCN はグラフを学習可能なニューラルネットワークの中でも比較的新しいニューラルネットワークである．GCN が提案される前にまず，GCN の前身となる技術として，グラフニューラルネットワーク（Graph Neural Networks，以降，GNN）[39] が提案された．GNN は，グラフ構造に対して深層学習を行うために開発されたニューラルネットワークである．その後，画像認識の分野で畳み込みを利用したニューラルネットワークが高い精度を示していることから，グラフにも畳み込みを適用することで精度が高くなると考えられ，GCN が提案された．

深層学習を用いてソースコード分類を行う既存研究 [45, 72] では，グラフの学習を行う際，ニューラルネットワークの入力形式に合わせて元のグラフを変形する．しかし，GCN ではグラフの変形は不要であるため，グラフの構造情報が欠落しないという利点がある．従って GCN を用いることで，グラフを変形する必要があるニューラルネットワークに比べ，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 2.4 に示す．図 2.4 は，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している．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他ノードからのエッジ（ingoing，outgoing），ノード 0 にループするエッジ（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 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のベクトル表現およびノード0に隣接しているノード1，2，3のベクトル表現に基づいて，ノード0のベクトル表現が計算される．

# 第 3 章

# 順伝播型ニューラルネットワークとコードミューテーションを用いたコードクローン検索

## 3.1 まえがき

ソフトウェアの開発者は，効率的なソフトウェアの開発を目的として，ソフトウェア再利用を行う場合がある [20,59]．その際，開発者は，自分が持っているコード片より効率的で信頼性の高いコード片を見つけることなどを目的として，あるコード片と同様の機能を持つコード片を検索することがある [35]．他にも，オープンソースソフトウェア (Open Source Software, 以下 OSS) や，Stack Overflow[*1]などの Q&A サイトから，利用可能なコード片を見つけた場合，開発者は元のコード片のライセンスを特定したり，より脆弱性の低いコード片を見つける必要がある．そのため，これまでに，コードクローンを検索するための手法が提案されている [18,35,48,70]．

深層ニューラルネットワーク (Deep Neural Networks, 以下 DNNs) はこれまで，コンピュータビジョン分野での物体検出だけでなく [42,49]，ソフトウェアエンジニアリング分野でのソースコード分類やコードクローン検出に用いられてきた [53,63,65,74]．そして，後に 5 章で示すように，深層学習を用いないソースコード分類手法と比べて，DNNs を用いるソースコード分類手法の精度は優れているため，DNNs は高精度なソースコード分類に有効な技術である．

本章では，FNN とミューテーションを用いたコードブロック単位のコードクローン検索手法を提案する．まず，本章の研究にて，DNNs の一種である FNN を用いて，ソースコードの機能に応じたコード片のラベリングを試みる．FNN によるラベリングが成功すると，そのラベリング結果をコードクローン検索に利用できるようになる．

[*1] https://stackoverflow.com/

また，学習に用いるソースコードのコードクローンを 2.1.3 節で説明したミューテーションによって作成することで，深層学習に利用可能な学習データを増やすことができる．

本検索手法は，学習を行う STEP L (Learning) と検索を行う STEP S (Search) の 2 段階で構成されている．STEP L では，まずソースコードに対してミューテーションを行い，様々なタイプのコードクローンを作成した後，元ソースコードとそのコードクローンからコードブロックを抽出する．次に，抽出した各コードブロックを特徴ベクトルに変換する．最後に，各クローンセットをソースコード分類におけるクラスに設定し，教師あり学習を実行することで FNN モデルの学習を行う．STEP S では，入力コード片から作成した特徴ベクトルを学習済み FNN モデルに入力する．その結果，学習済コードブロックの中に入力コード片のコードクローンが存在する場合は，そのコードクローンが属するクローンセットのクラスに，入力コード片は分類される．そのため，分類先のクローンセットを検索結果として出力することで，入力コード片のコードクローンを検索することができる．評価実験では，3 つのオープンソースソフトウェアに対して本検索手法を適用した．その結果，本検索手法は構文的なコードクローンを高い精度で検索することが確認できた．

以降，3.2 節では，本章で提案するコードクローン検索手法について述べる．3.3 節では，評価実験を行い，提案手法の検索精度について述べる．3.4 節では，本章の提案手法に関する妥当性の脅威について述べる．3.5 節では，本章の関連研究について述べる．最後に 3.6 節で本章のまとめについて述べる．

## 3.2 提案手法

本章では，FNN とソースコードミューテーションを用いたコードブロック単位のコードクローン検索手法を提案する．本検索手法は，深層学習を使用することで，ある 2 つの要素の対応付けを学習用データセットから経験的に取得することができ，入力として与えられたコード片の構文的なコードクローンを容易に対応させることが可能である．また，深層学習を用いることで構文上類似しているが検索できなかったコードブロックを新たに学習させることで次からは正しく判定でき，検索漏れや誤検索を減らすことができる．本検索手法は，FNN にコードブロックを学習させる STEP L (Learning) と，学習済み FNN を用いてコードブロックの検索を行う STEP S (Search) の 2 段階で構成されている．

### 3.2.1 用語の定義

**コードブロック:** 以下の 2 つの条件のいずれかを満たすコード片を，コードブロックと定義する．

条件 1. 関数の“{ }”で囲まれた範囲

条件 2. if，else，for，while，do-while，switch 文の“{ }”で囲まれた範囲

**クローンセット:** その集合内の任意のペアがコードクローンであるコードブロックの最大集合を，クローンセットと定義する．

**ネガティブデータ:** コードクローン検索を行う際，クローンセットを出力するという事象の他に，検索結果なしという事象が存在するが，本検索手法では，FNN を使用したソースコード分類をコードクローン検索に応用しているため，学習の際に，分類における “検索結果なし” クラスを作成するためのベクトルが必要となる．本検索手法では，以下の条件 3 に当てはまるコードブロックから作成した特徴ベクトルをネガティブデータとして定義し，“検索結果なし” クラスを作成するためのベクトルとして学習に使用する．

条件 3. 検索対象のリポジトリ内には存在せず，次に説明するポジティブデータのコードブロックすべてと構文的に類似していないため，“検索結果なし” を意味するクラスに分類されるコードブロック

**ポジティブデータ:** 本検索手法では，以下の条件 4 に当てはまるコードブロックから生成された特徴ベクトルをポジティブデータとして定義する．

条件 4. 検索対象のリポジトリ内に存在し，“検索結果なし” を意味するクラス以外に分類されるコードブロックと，それらのコードブロックに対して 2.1.3 節のミューテーションを適用して作成したコードクローン

### 3.2.2 順伝播型ニューラルネットワークを用いた深層学習

この節では，FNN による深層学習を行う STEP L について説明する．STEP L では，OSS リポジトリから取得したソースコードを基に学習データを作成し，そのデータを用いて FNN モデルを作成する．STEP L は 5 つの手順から構成されている．STEP L の概要を図 3.1 に示す．

**STEP L1** OSS リポジトリ内のソースコードに対して字句解析を行い，コードブロックを抽出した後，我々のグループで開発しているコードクローン検出ツール CCFinder [29] および，横井らのブロッククローン検出ツール CCVolti [68] を使用し，クローンセット $S_i(1 \leq i \leq I)$ を作成する．クローンセット $S_i$ には互いにコードクローン関係にあるコードブロック ${b_i}^{(j)}(1 \leq j \leq J)$ が属する．また，ネガティブデータ (3.2.1 節) 生成用のソースコードを用意し，コードブロック ${b_0}^{(l)}(1 \leq l \leq L)$ を抽出する．

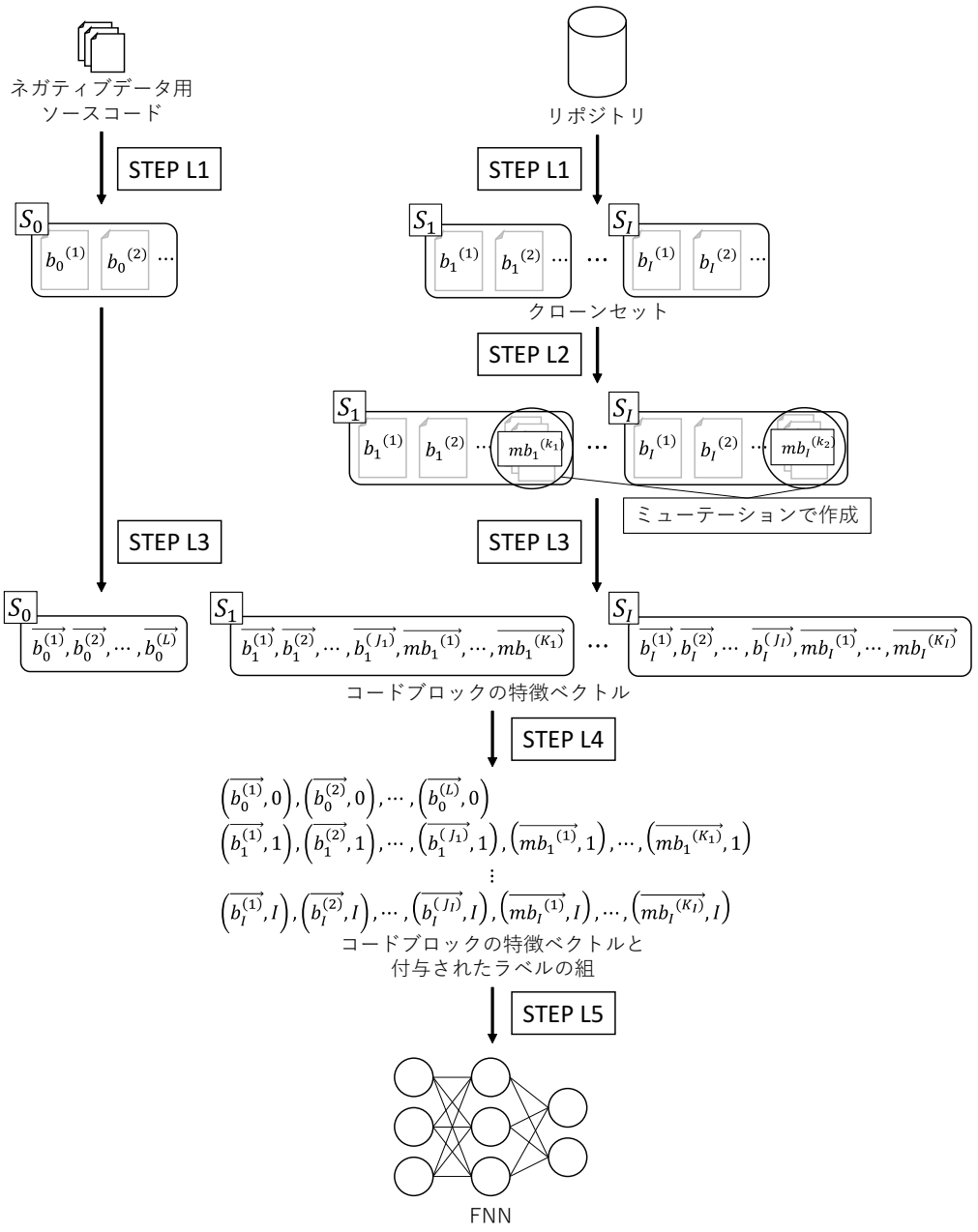


図 3.1: STEP L の概要図

**STEP L2** 2.1.3 節で説明した 13 種類のミューテーションオペレータを利用し, クローンセット $S_i$ に含まれるコードブロック ${b_i}^{(j)}$ のコードクローン ${mb_i}^{(k)}(1 \leq k \leq K)$ を作成する．このとき，${mb_i}^{(k)}(1 \leq k \leq K)$ は，クローンセット $S_i$ に属する．

**STEP L3** コードブロック ${b_i}^{(j)}$，${mb_i}^{(k)}$ を特徴ベクトルに変換する．その際に用いた手法については 3.2.3 節で説明する．

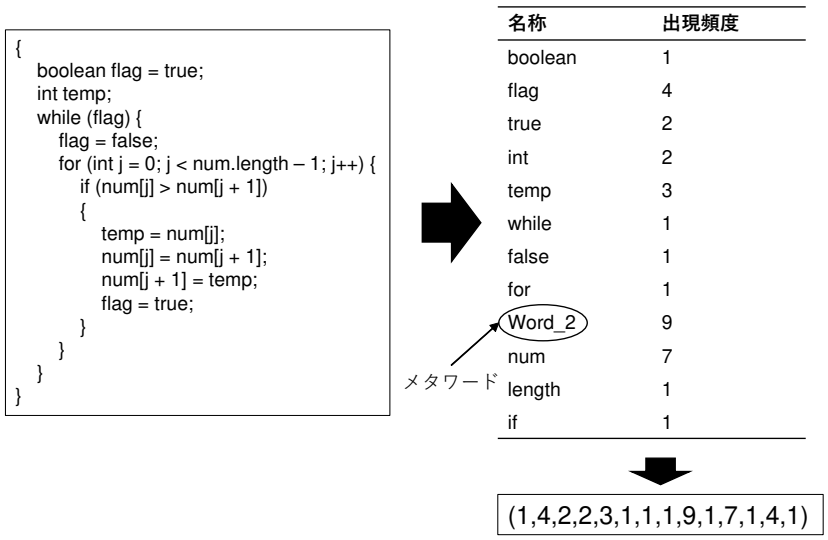


図 3.2: BoW をコード片に適用した例

**STEP L4** クローンセット $S_i$ に属するコードブロックから生成された特徴ベクトルをクラス $C_i$ と定義し，ラベル $i$ を付与する．また，コードブロック集合 ${b_0}^{(l)}$ には，OSS リポジトリ内にコードクローンが存在しないという意味でクラス $C_0$ と定義し，ラベル 0 を付与する．

**STEP L5** 特徴ベクトルと付与したラベルを用いて教師あり学習を行い，FNN モデルを作成する．

### 3.2.3 特徴ベクトルの生成

本章の評価実験では，特徴ベクトルを生成する最も単純な手法である BoW（Bag of Words）と，教師なし学習をベースにした文書のベクトル化手法である Doc2Vec [37] の 2 種類を個別に使用してそれぞれモデルを作成し，検索精度の比較を行った．

**BoW:** 学習データ用コードブロックに現れる各予約語・識別子の数を特徴量として，各コードブロックを特徴ベクトルに変換する．その際，2 字以下の識別子はメタワードとして，まとめて数える．また，検索クエリとして与えられたコード片を特徴ベクトルに変換する際は，学習用データセットに含まれるコードブロックに現れていた各予約語・識別子がそのコード片に現れている数を特徴量としてベクトルに変換する．BoW を用いたベクトル変換の例を図 3.2 に示す．図 3.2 中の Word_2 はメタワードであり，変数 x および y がこれに該当する．

**Doc2Vec:** Doc2Vec は，教師なし学習に基づいて特徴ベクトルを生成する手法であり，自然文書を対象としたタスクにおいて有効性が実証されている [37]．本検索手法

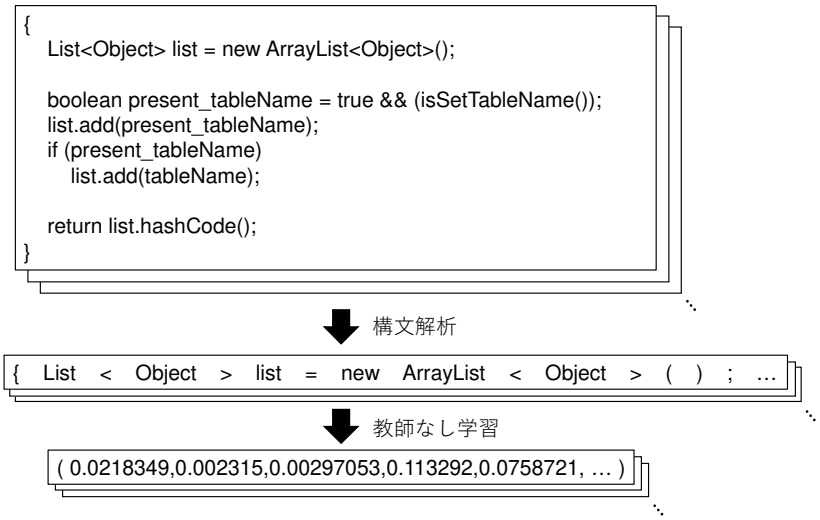


図 3.3: Doc2Vec をコード片に適用した例

では，Doc2Vec を用いてコードブロックをベクトル化するにあたって，ライブラリは gensim[*2]を利用している．Doc2Vec を用いたベクトル変換の例を図 3.3 に示す．まず，ANTLR[*3]を用いて学習用データセットに含まれるコードブロックをトークン単位に分割する．次に，コードブロックのトークンを 1 行にスペースで分かち書きをする．そのような処理を行った文字列を学習データとして与えると，教師なし学習によってトークンの埋め込みベクトル情報を持った Doc2Vec モデルが作成される．その Doc2Vec モデルにコードブロックのトークン列を学習時の入力と同様の形式で入力すると，そのコードブロックの埋め込みベクトルを得ることができる．

### 3.2.4 学習済み順伝播型ニューラルネットワークを用いたコードクローン検索

この節では，コードクローンの検索を行う STEP S について説明する．STEP S では，STEP L で作成した学習済み FNN を使用し，検索を実行する．STEP S は 3 つの手順から構成されている．STEP S の概要を図 3.4 に示す．

**STEP S1** 検索クエリコード片を字句解析し，STEP L3 と同様，3.2.3 節で説明した手法で特徴ベクトルに変換する．

**STEP S2** STEP L5 で作成した学習済み FNN モデルに STEP S1 で作成した特徴ベクトルを入力として与えることで，検索クエリコード片のラベルを予測する．

[*2] https://radimrehurek.com/gensim/
[*3] http://www.antlr.org/

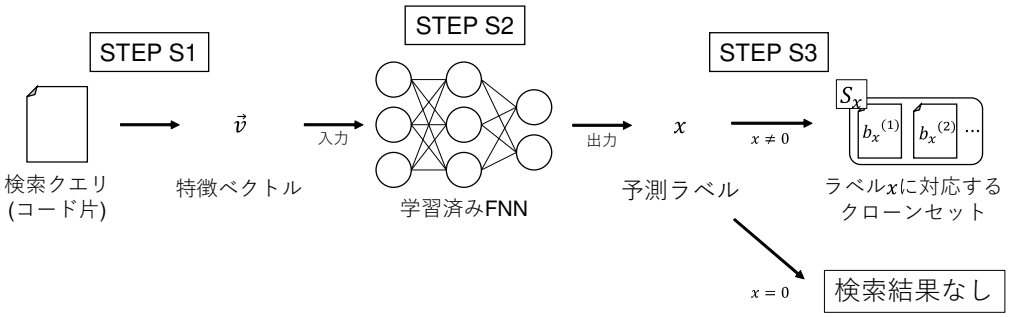


図 3.4: STEP S の概要図

**STEP S3** FNN モデルが予測したラベル $x$ に対応するクローンセット $S_x$ に属するコードブロック ${b_x}^{(j)}(1 \leq j \leq J)$ を検索結果のコードクローンとして出力する．ただし，$x = 0$ の場合は，検索クエリコード片がポジティブデータよりもネガティブデータに類似していることが示唆されているため，検索結果は無しとする．

## 3.3 評価実験

本評価実験では，3 つの OSS から作成したベンチマークを使用し，コードクローンの検索精度を，適合率，再現率，F 値の観点から評価した．対象とした OSS は HBase 2.0[*4]，OpenSSL 0.9.1〜1.1.0[*5]，FreeBSD 11.1.0[*6]である．OpenSSL はアップデート等により生じるバージョン間のコードクローンを評価に使用するため，複数のバージョンを使用している．また，本評価実験では，深層学習フレームワーク Chainer3.4.0[*7]を使用し，FNN モデルを実装した．

### 3.3.1 ハイパーパラメータ

本評価実験では，グリッドサーチによってハイパーパラメータを決定した．グリッドサーチとは，ハイパーパラメータ群の候補を規則的に決め，各ハイパーパラメータの組み合わせを順番に探索し，適切なハイパーパラメータを決定する手法である．その結果，ニューラルネットワークの層の数は 3 層で，隠れ層の次元数は 200 となった．また，埋め込みベクトルは，次元数を増やすほどベクトルの質は良くなり次元数が 300 を超えるとベクトルの質の変化が少なくなる傾向がある [43] ため，本章では Doc2Vec で生成する埋め込みベクトルの次元数を 300 に設定した．本評価実験の

[*4] https://hbase.apache.org/
[*5] https://www.openssl.org/
[*6] https://www.freebsd.org/
[*7] https://chainer.org/

FNN モデルで用いる活性化関数は ReLU，損失関数は CrossEntropy, 最適化アルゴリズムは Adam である．また，FNN モデルの出力層にはソフトマックス関数と閾値 0.95 のステップ関数を使用し，その結果出力が 0 ベクトルになった場合は，特徴ベクトルにラベル 0 を割り当てる．

### 3.3.2 実験手順

この節では，検索精度の評価実験を行う STEP E について説明する．STEP E は 3 つの手順で構成されている．

**STEP E1** 各 OSS を用いて学習用データセットと評価用データセットを作成する．

**STEP E2** 学習用データセットを用いて FNN モデルの学習を行う．

**STEP E3** 学習済み FNN モデルに評価用データセットに含まれる各コードブロックを入力として与え，適合率，再現率，F 値を算出する．

### 3.3.3 データセット作成手順

3.3.2 節の STEP E1 において，本実験で使用した学習用データセットと評価用データセットは以下の 4 つの手順で作成した．

**STEP E1 (1)** 各 OSS からクローンセットを抽出し，100 個以上の要素を持つクローンセットを選択することで，FNN モデルの学習や評価に使用するコードブロックを多数用意する．

**STEP E1 (2)** 選択したクローンセット毎に，コードブロックの 20% に対して，2.1.3 節のミューテーションを適用して新たなコードクローンを生成し，そこから生成した特徴ベクトルを用いて，学習用のポジティブデータを作成する．また，各特徴ベクトルには，3.2.2 節の STEP L4 と同じ方法で，クローンセットに対応したラベルを付与する．

**STEP E1 (3)** STEP E1(2) で作成したポジティブデータと構文的に類似していないコードブロックの特徴ベクトルを，学習用データセットのネガティブデータとして 30000 個追加する．その際に使用したソースコードは，用いられているプログラミング言語の関係上，HBase に関しては BigCloneBench [60] から，OpenSSL に関しては FreeBSD から，FreeBSD に関しては OpenSSL から取得した．

**STEP E1 (4)** 実験対象の OSS から抽出したコードブロックと，STEP E1(2) で使わなかった 80% のコードブロックから生成した特徴ベクトルを評価用データセットとする．評価用データセットに存在する，STEP E(1) で抽出したクローンセットに属す

表 3.1: データセット

| | 学習用データセット | | 評価用データセット | |
|---|---|---|---|---|
| OSS | ポジティブデータ | ネガティブデータ | ポジティブデータ | ネガティブデータ |
| Hbase | 28,822 | 30,000 | 740 | 12,688 |
| OpenSSL | 36,772 | 30,000 | 281 | 99,719 |
| FreeBSD | 27,852 | 30,000 | 747 | 8,177 |

るコードブロックの特徴ベクトルには，3.2.2 節の STEP L4 と同じ方法で，クローンセットに対応したラベルを付与する．

各 OSS から作成したデータセットにおけるポジティブデータとネガティブデータの内訳を表 3.1 に示す．評価用データセットは実験対象の OSS に含まれているすべてのコードブロックであるため，評価用データセットのポジティブデータとネガティブデータの間には偏りがある．また，OpenSSL は複数バージョンを使用しているため，Hbase や FreeBSD に比べ，全体のコードブロック数が多い．また，学習させるコードブロックと検索したいコード片の間に存在する構文的な差異の程度と検索精度の関係を調べるため，学習用データセットにおけるポジティブデータと評価用データセットにおけるポジティブデータの構文的な類似度（2.1 節参照）を，対象の OSS ごとに変更した．具体的には，Hbase からは構文的な類似度が 0.9〜1.0 のコードクローンを，OpenSSL からは構文的な類似度が 0.7〜1.0 のコードクローンを，FreeBSD からは構文的な類似度が 0.1〜0.2 のコードクローンを抽出し，評価実験に利用した．すなわち，Hbase と OpenSSL から抽出したのは構文的コードクローンであり，FreeBSD から抽出したのは意味的コードクローンである．

### 3.3.4 評価指標

本実験では，適合率，再現率，F 値を用いて検索精度の評価を行った．これらの評価指標の説明を以下に示す．

**適合率:** 検索結果に対して本当に正しかった割合を指し，正確性に関する指標として用いられる．本実験では，学習済み FNN モデルが 0 以外のラベルを予測した評価用データセットのデータに対する，ポジティブデータの割合によって適合率を求める．

**再現率:** 正解に対して実際に検出された割合を指し，網羅性に関する指標として用いられる．本実験では，評価用データセットに含まれるポジティブデータ全体のうち，学習済み FNN モデルへ入力すると正解ラベルが予測されるデータの割合によって再現率を求める．

**F 値:** 適合率と再現率の総合的な評価として用いられ，適合率と再現率の調和平均に

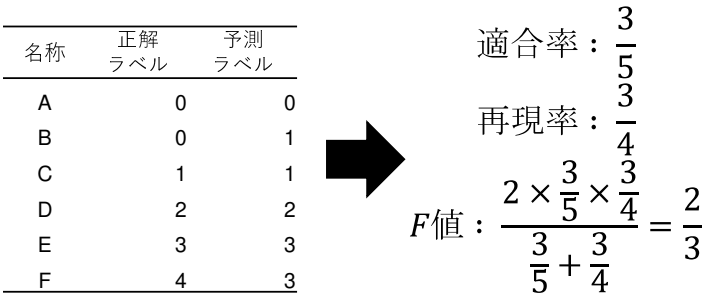


図 3.5: 評価指標の計算例

表 3.2: 検索精度の評価

| | BoW | | | Doc2Vec | | |
|---|---|---|---|---|---|---|
| OSS | 適合率 | 再現率 | F 値 | 適合率 | 再現率 | F 値 |
| Hbase | 0.924 | 1.000 | 0.960 | 0.830 | 1.000 | 0.907 |
| OpenSSL | 0.733 | 1.000 | 0.846 | 0.652 | 1.000 | 0.789 |
| FreeBSD | 0.497 | 0.822 | 0.620 | 0.519 | 0.529 | 0.524 |

よって求める．

評価指標の計算例を図 3.5 に示す．この例では，コードブロック B〜F の 5 つに対して学習済み FNN モデルが 0 以外のラベルを予測結果として出力しているが，そのうち正解ラベルを予測できているのは C〜E の 3 つであるため，適合率は 3/5 となる．また，入力として与えたコードブロックのうち，実際にコードクローンを FNN モデルが学習している（正解ラベルが 0 以外である）のは C〜F の 4 つで，そのうち学習済み FNN モデルが正解ラベルを予測したのは C〜E の 3 つであるため，再現率は 3/4 となる．F 値は，適合率と再現率の調和平均なので，2/3 となる．

### 3.3.5 実験結果

評価実験から算出された適合率，再現率，F 値を表 3.2 に示す．Hbase と OpenSSL から作成したデータセットの再現率は 1.000 であった．この結果から，コードブロックを事前に学習した FNN モデルは，構文的な類似度が高いコードクローンを高い精度で検索可能なことが分かった．

検索に成功したコードブロックの例を図 3.6，3.7 に示す．図 3.6 (a) と図 3.7 (a) のコードブロックは評価用ポジティブデータであり，コードクローン検索における検索クエリである．また，図 3.6 (b) と図 3.7 (b) のコードブロックは学習用ポジティブデータであり，コードクローン検索における検索対象である．学習済み FNN モデルの出力は，図 3.6 (a) と図 3.7 (a) から生成された特徴ベクトルは，図 3.6 (b) と図 3.7

```
List<Object> list = new ArrayList<Object>();

boolean present_message = true && (isSetMessage());
list.add(present_message);
if (present_message)
   list.add(message);

return list.hashCode()
```

(a) 検索クエリ

```
List<Object> list = new ArrayList<Object>();

boolean present_tableName = true && (isSetTableName());
list.add(present_tableName);
if (present_tableName)
   list.add(tableName);

return list.hashCode();
```

(b) 検索結果

図 3.6: 構文が一致しているコードクローンの検索成功例（HBase）

```
{
int num,i;
(省略)
for (;;)
        {
        (省略)
        if (num >= arg->count)
                {
                char **tmp_p;
                int tlen = arg->count + 20;
                tmp_p = (char **)OPENSSL_realloc(arg->data,
                        sizeof(char *)*tlen);
                if (tmp_p == NULL)
                        return 0;
                arg->data  = tmp_p;
                arg->count = tlen;
                for (i = num; i < arg->count; i++)
                        arg->data[i] = NULL;
                }
        (省略)
        }
*argc=num;
*argv=arg->data;
return(1);
}
```

(a) 検索クエリ

```
{
int num,len,i;
(省略)
for (;;)
        {
        (省略)
        if (num >= arg->count)
                {
                arg->count+=20;
                arg->data=(char **)OPENSSL_realloc(arg->data,
                        sizeof(char *)*arg->count);
                if (argc == 0) return(0);
                }
        (省略)
        }
*argc=num;
*argv=arg->data;
return(1);
}
```

(b) 検索結果

図 3.7: 構文が類似しているコードクローンの検索成功例（OpenSSL）

```
ArrayList<Long> timestamps = new ArrayList<>(filterArguments.size());

for (int i = 0; i<filterArguments.size(); i++) {
    long timestamp =
         ParseFilter.convertByteArrayToLong(filterArguments.get(i));
    timestamps.add(timestamp);
   }
return new TimestampsFilter(timestamps);
```

(a) 検索クエリ

```
List<Object> list = new ArrayList<Object>();

boolean present_tableName = true && (isSetTableName());
list.add(present_tableName);
if (present_tableName)
   list.add(tableName);

return list.hashCode()
```

(b) 検索結果

図 3.8: 検索に失敗した例（HBase）

(b) から生成された特徴ベクトルと類似していることを示していた．図 3.6 と図 3.7 の太字は，それぞれの検索クエリコードと検索結果の違いを示している．

検索に失敗した例を図 3.8 に示す．これらのコードブロックは構文的に類似していないが，上下のコードブロックがコードクローンであると FNN モデルは判定している．太字で示した部分は，FNN モデルがこれら 2 つのコード片がコードクローンであると判定した原因だと考えられる文である．この結果についての考察は，3.3.6 節で述べる．

### 3.3.6 考察

HBase，OpenSSL を用いた実験では，再現率が 1.000 という結果になった．この結果は，検索クエリのコードブロックと高い類似性を持つコードブロックが検索対象のコードブロックに含まれている場合は，正しい検索結果を返すということを示している．また，再現率と比較して，適合率が少し低いという結果になった．適合率が低い原因として，FNN モデルが局所的な特徴に基づいてラベルを予測する傾向が影響していると考えられる．例えば，List<Object> list = new Arraylist<Object>(); という文が含まれるコードブロックが OSS リポジトリに存在し，そのコードブロックを学習データに用いた場合，学習済み FNN モデルに，List<Object> list = new Arraylist<Object>(); などの List 型のオブジェクトを作成する文を含むコードブロックを与えると，これら 2 つのコードブロックは，全体的には構文的に類似していなくても，コードクローンであるとモデルが誤判定しやすくなる．本実験では，図 3.8 の太字になっている文が原因で，図 3.8 の (a)，(b) はコードクローンであると学習済み FNN モデルは判定したと考えられる．

FreeBSD を用いた実験では，HBase や OpenSSL に比べると検索精度が悪かった．これは，FreeBSD から作成したデータセットに含まれるコードクローンの構文的な類似度が，他のデータセットに比べて低かったためだと考えられる．よって，本提案手法では，構文的な類似度の低い意味的コードクローンの検索は困難であることが分かった．

### 3.3.7 ミューテーションの影響に関する評価実験

本節では，ミューテーションを行うことによる検索結果への影響を調べるための評価実験を行う．

提案手法で用いている FNN モデルは，出力層でソフトマックス関数とステップ関数を使ってラベルの予測を行ったが，この評価実験では，FNN モデルに特徴ベクトルを入力したときにソフトマックス関数が出力する，各ラベルに対する予測確率について分析する．具体的には，学習用データセットに使用するポジティブデータの数と，学習済み FNN モデルがコードクローンを入力として与えられたときに正解ラベルに対する予測確率の関係を調べることで，ミューテーションの効果を評価している．

#### ミューテーションの影響に関する評価実験の手順

この節では，ミューテーションの評価を行う STEP M (evaluation of Mutation) について説明する．STEP M は以下の 4 つの手順で構成されている．

**STEP M1** OpenSSL の過去 200 種類以上のバージョン間でコードクローン $f_n(n =$

$1, 2, \cdots, N$) が存在する OpenSSL の最新バージョンの関数 $f$ を選択する．それらの関数に対して 2.1.3 節のミューテーションを適用することでコードクローンを生成し，OpenSSL のバージョン 1.0.0 以前のソースコードを学習させた Doc2Vec のモデルを用いて特徴ベクトルに変換する．そして，それらの特徴ベクトルをクラス $C_1$ と定義し，ラベル 1 を付与する．

**STEP M2** STEP M1 で作成した特徴ベクトルの中から $a$ 個だけポジティブデータとして学習用データセットに加え，FreeBSD から抽出したコードブロックからランダムに 30000 個を選択して STEP M1 と同じ Doc2vec モデルを用いて特徴ベクトルに変換する．そして，それらの特徴ベクトルをクラス $C_0$ と定義してラベル 0 を付与し，ネガティブデータとして学習用データセットに加える．そして，$a + 30000$ 個のコードブロックからなる学習用データセットを用いて，FNN モデル $M_a$ の学習を行う．$a$ の値は，$a = 1, 100, 1000, 2000, 3000, 4000, 5000, 7000, 10000$ と変化させ，合計 9 種類の FNN モデルの学習を行う．

**STEP M3** STEP M1 で選択した関数 $f$ の過去バージョンに存在するコードクローン $f_n(n = 1, 2, \cdots, N)$ を，STEPM1 と同じ Doc2vec モデルを用いて特徴ベクトル $\vec{f_n}(n = 1, 2, \cdots, N)$ に変換し，9 種類の学習済み FNN モデル $M_1, M_{100}, \cdots, M_{10000}$ それぞれに入力として与える．

**STEP M4** 学習済み FNN モデルが出力したベクトルから，STEP M2 で学習させたポジティブデータ群と STEP M3 で入力した特徴ベクトルがクローンセットに属している (ラベルが 1 である) と FNN モデル $M_n$ が予測する確率 $P_{M_a}(\vec{f_n}, 1)$ を求め，ポジティブデータ数 $a$ と確率 $P_{M_a}(\vec{f_n}, 1)$ の関係を調べる．

### ミューテーションの影響に関する評価実験の結果

ミューテーションの影響に関する評価実験の結果を表 3.3 に示す．また，表 3.3 を折れ線グラフで表現したものを図 3.9 に示す．表 3.3 中の $average_a$ と $min_a$ は以下の式で与えられる．$average_a$ は学習済み FNN モデル $M_a$ が特徴ベクトル $\vec{f_n}(n = 1, 2, \cdots, N)$ に対して算出するラベル予測確率の平均値であり，$min_a$ はラベル予測確率の最小値である．

$$average_a = \frac{1}{N} \sum_{n=1}^{N} P_{M_a}(\vec{f_n}) \tag{3.1}$$

$$min_a = \min_{n=1,2,\cdots,N} P_{M_a}(\vec{f_n}) \tag{3.2}$$

表 3.3 や図 3.9 から分かるように，学習データ数 $a$ が大きいほど $average_a$ と $min_a$ は増加するという結果になった．

表 3.3: ポジティブデータ数と予測確率

| ポジティブデータ | $average_a$ | $min_a$ |
|---:|---:|---:|
| 1 | 0.000 | 0.000 |
| 100 | 0.000 | 0.000 |
| 1000 | 0.000 | 0.000 |
| 2000 | 0.973 | 0.173 |
| 3000 | 0.992 | 0.675 |
| 4000 | 0.989 | 0.731 |
| 5000 | 0.998 | 0.871 |
| 7000 | 0.999 | 0.955 |
| 10000 | 0.999 | 0.978 |

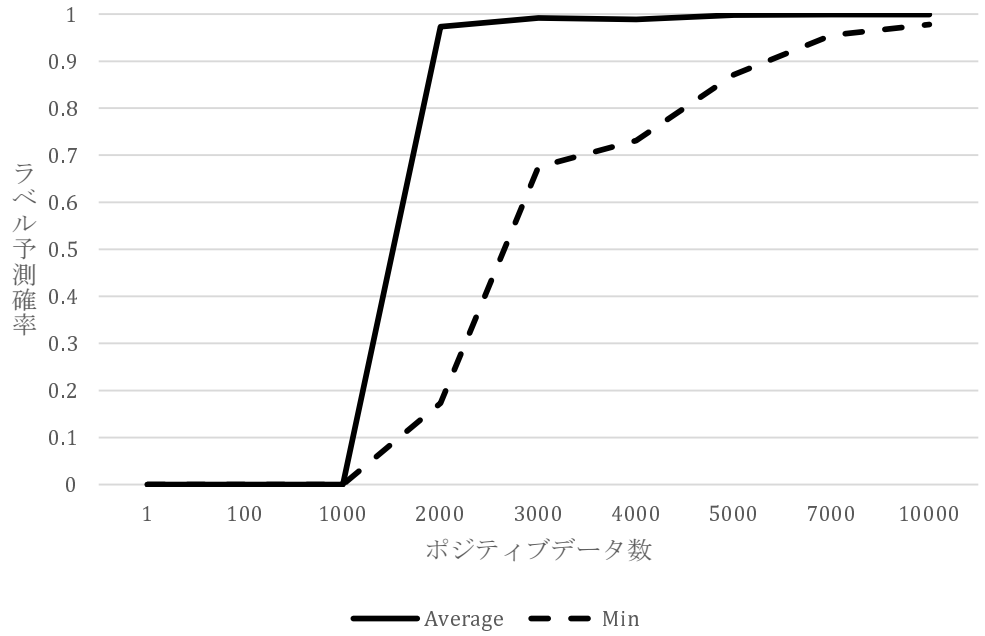


図 3.9: ポジティブデータ数と予測確率の関係

### ミューテーションの影響についての考察

ソースコードに対してミューテーションを行って学習データを増やす手法は，深層学習における入門である，MNIST などの画像分類の問題から着想を得ている．画像分類の問題では，学習データを増やすために，元の画像に対して拡大，縮小，移動などの操作を行い，元の画像と少し異なる類似画像を新たに作成して学習データに加えることがある．この発想をソースコードに適用し，元のソースコードと少し異なるコー

ドクローンを生成するために，ミューテーションを利用した．

本節の実験においては，ポジティブデータを 2000 個以上使って学習を行うと，学習済み FNN モデルは，学習に使用したポジティブデータと過去バージョンに存在するコードクローンのほとんどを 99% 以上の確率で類似していると判定し，ポジティブデータを 7000 個以上使って学習を行うと，過去バージョンに存在する全てのコードクローンに対して，90% 以上の確率で類似していると判定するようになった．この実験から，学習に用いるコードブロックの数は検索精度に大きく影響することが分かった．また，ミューテーションを用いて学習データを増やすことは，検索精度の向上に有効であることが分かった．

## 3.4 妥当性の脅威

提案手法における妥当性の脅威として，3.2.2 節にて CCFinder や CCVolti が出力したコードクローンを学習に用いているため，実際はコードクローンではないものをコードクローンと一緒に学習させる可能性がある点が挙げられる．この点に関して，本章の評価実験では OSS ごとにコードクローンの類似度を変更しており（3.3.3 節参照），その際にコードクローンの構文的な類似度を測定している．そのため本章の評価実験では問題ないと考えられるが，他のデータセットに対して適用する場合は注意が必要である．

## 3.5 関連研究

ソースコード検索とコードクローン検出は関連性のあるタスクである．コード片を検索クエリとして入力するソースコード検索は，入力コード片のコードクローンが検索対象のプロジェクトに存在するかどうかを調べている．そのため，Ichi Tracker [24] などのソースコード検索エンジンは，コードクローン検出ツールを用いてソースコード検索を行っている．

神谷らが開発した CCFinder [29] は，トークン解析を行い，ユーザー定義名を特殊文字に変換した後，ソースコードをトークン列に変換し，閾値以上の長さで一致したトークン列をコードクローンとして検出している．

我々の研究グループでは，コンポーネントやファイル単位での再利用を特定する研究を行っている [25, 26, 34]．これらの研究では，コンポーネントに含まれるクラスのシグネチャ（クラス名，メソッドのシグネチャ，フィールド名など）やコンポーネントに含まれるファイル間の Jaccard 係数，ファイル間の最長共通部分列を利用してコンポーネント単位およびファイル単位の再利用が可能なソースコードを検出を行う．本章の研究では，深層学習を用いて，コンポーネントやファイルよりも細粒度であるコード片の検索を行っている．

White ら [65] は，RNN とオートエンコーダを使用して抽象構文木のベクトル化を行い，各構文木ベクトルの l2 ノルムを算出することによって，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深層学習モデルである，RNN Encoder-Decoder モデルを使用し，自然言語をクエリとして与えることで API の使用順序例を生成する，“API Learning”を提案している．これらの研究と異なり，本章の研究は，深層学習を用いたコードクローンの検索を目的としている．また，本章の研究は RNN に比べて単純であり，動作が軽量な構造のニューラルネットワークである FNN を使用している．

## 3.6 まとめ

本章では，FNN を用いたコードクローン検索手法を提案した．コードクローンの学習では，検索対象ソースコードのコードブロックと，それに対してミューテーションを適用して生成したコードクローンを用いて学習用データセットを作成し，そのデータセットを用いて FNN モデルの学習を行う．コードクローンの検索では，検索クエリとして入力されたコード片の特徴ベクトルを学習済み FNN モデルに入力すると，FNN モデルがラベル予測を行い，そのラベルに対応するコードクローンが出力される．評価実験では，3 つの OSS に対してコードクローン検索を行った．その結果，提案手法は構文的に類似したコードクローンを高い精度で検索できることが分かった．

# 第 4 章

# 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

## 4.1　まえがき

3.3.7 節で示したように，学習に用いるソースコードの数は，深層学習を用いたソースコード分類の精度に大きく影響することが分かった．また，ソースコードの分類に限らず，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深層学習モデルの分類精度に大きな影響を与えると考えられているため，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 [67]．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 1 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，深層学習モデルの分類精度が低下するという問題である．Yan ら [67] 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．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 [12, 21, 45, 46, 53, 72] 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．本章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，深層学習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．一般的に深層学習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，深層学習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，より高い分類精度の深層学習モデルを作成できる可能性がある．

そこで本章では，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，学習済み深層学習モデルの評価用データセットに対するソースコード分類精度の検証を行い，正答率が相対的に低いクラスに対し，そのクラスのソースコードに対してミューテーションを適用して作成したコードクローンを追加する．このようにすることで，コードクローンを追加したクラスの正答率が上昇し，低下していた深層学習モデルのソースコード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 種類のクローンセットから，3 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の分類ができることを確認した．また，提案手法を用いて深層学習モデルの学習とコードクローン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，4.2 節で本章の提案手法について述べる．4.3 節で本章の評価実験について述べる．4.4 節で本章の提案手法に関する妥当性の脅威について述べる．4.5 節で関連研究について述べる．最後に，4.6 節で本章のまとめについて述べる．

## 4.2 提案手法

本章では，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，深層学習モデルの分類精度の向上が期待できる．

提案手法は 4.1 節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，深層学習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．一般的に深層学習モデル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．

### 4.2.1 用語の定義

**クローンセット** 互いにコードクローンであるソースコードの集合をクローンセット $S$ と定義する．クラス数を $n$ とする場合，本章で扱うソースコードはクローンセット $S_0 \ldots S_{n-1}$ のいずれか 1 つにのみ分類される．つまり，クローンセット $S_i (0 \leq i \leq n-1)$ に属するソースコードは互いにコードクローンであり，$S_j$ , $S_k (0 \leq j \leq n-1,\ 0 \leq k \leq n-1,\ j \neq k)$ の 2 つのクローンセットから 1 つずつ抽

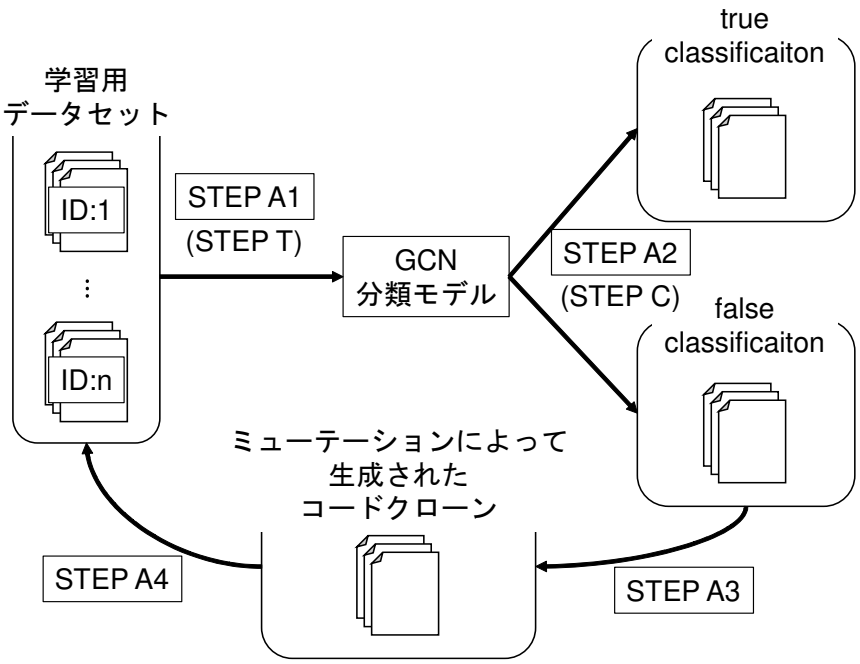


図 4.1: STEP A の概要

出した 2 つのソースコードはコードクローンではない.

**クローンセット ID** 本章では，$n$ 個の各クローン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，深層学習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，各クローンセットに対して割り当てられる固有のインデックス $0 \ldots n-1$ をクローンセット ID と定義する.

**学習用データセット** 深層学習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** 深層学習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.

### 4.2.2 動的な学習用データセット改善手法

まず, 本章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，深層学習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.

本章では，3.1 節で用いた FNN とは異なり，2.2.3 節で説明した GCN を用いたソースコード分類手法に提案手法の適用を試みる．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，深層学習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 2 つの手順で構成されている．

STEP A の概要を図 4.1 に示す．

**STEP A1** STEP T (4.2.2 節) 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 (4.2.2 節) 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，深層学習モデルによって，各ソースコードに対応するクローンセット ID の予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 2.1.3 節のミューテーションを適用し，一定数のコードクローンを作成する．本章では，1 つのコードクローンを作成する際，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コードクローンに対して，元のソースコードに割り当てられていたクローンセット 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 における深層学習モデルの学習とコードクローンの追加を，false classification の数が減少しなくなるまで繰り返す．

### 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 4.2 に示す．

**STEP T1** 学習対象のソースコードからクローンセットを構築し，各クローンセットに対して固有のクローンセット 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，クローンセット ID を目的変数として，

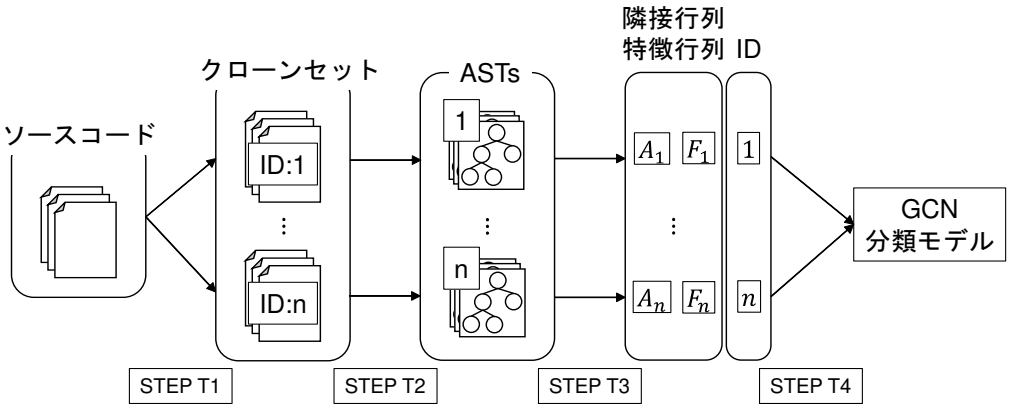


図 4.2: STEP T の概要

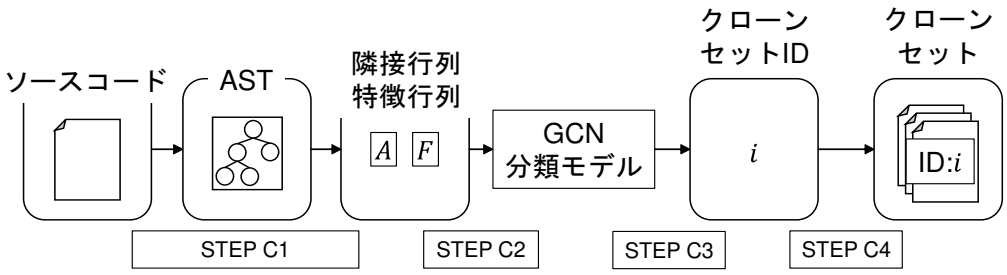


図 4.3: STEP C の概要

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．

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 の概要を図 4.3 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クローンセット ID の予測結果が出力される．

**STEP C4** 分類対象のソースコードを，出力されたクローンセット ID が示すクローンセットとして分類する．

表 4.1: 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[*3] |

## 4.3 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 4 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深層学習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，4.3.1 節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，深層学習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 1 つの機能のまとまりであり，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[*1]を使用し，その文法ファイルには CPP14.g4[*2] を使用した．また，GCN の実装は Kipf らの実装 [36] を使用し，ハイパーパラメータは 3.3.1 節と同様にグリッドサーチで決定した．その結果，GCN の畳み込み層は 4 層，GCN の隠れ層のノード数は 128 となった．また，AST ノードのベクトル化は 3.2.3 節で説明した BoW を用いて行った．ただし 3.2.3 節とは異なり，本章では識別子や予約語だけでなく全ての AST ノードを特徴量として扱い，ベクトル化の単位はコードブロック単位ではなく AST ノード単位である．すなわち AST ノードのベクトル化を行うと，その AST ノードに対応する要素が 1 で他の要素が 0 であるベクトル（One-hot ベクトル）が生成される．また，本評価実験を行った環境を表 4.1 に示す．

### 4.3.1 データセット

本評価実験では，オープンソースソフトウェア OpenSSL[*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．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

[*1] https://www.antlr.org/
[*2] https://github.com/antlr/grammars-v4/tree/master/cpp
[*3] https://www.tensorflow.org/
[*4] https://www.openssl.org/

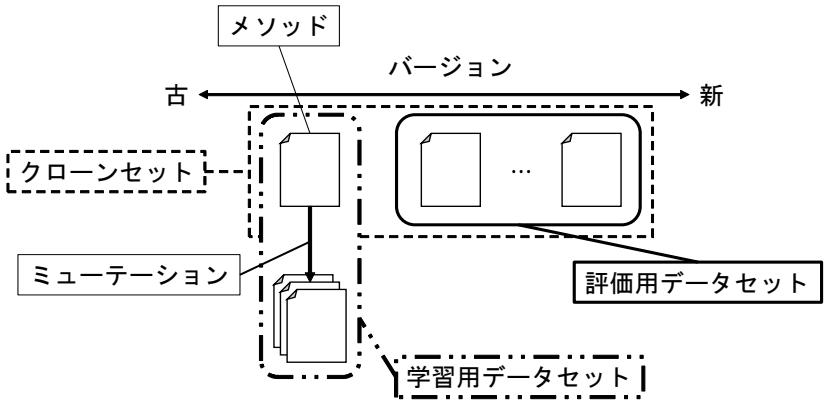


図 4.4: 学習用・評価用データセット作成方法の概要

同じ名称を持つメソッドは同じ機能を持ち，異なる名称を持つメソッドは異なる機能を持つという仮定の下で，意味的コードクローンセットを作成した．ここで，メモリ容量の都合上，作成したクローンセットをランダムに 20 個を選択して評価実験に使用した．

次に，各クローン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 4.4 に示す．まず，クローン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，クローン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コードクローンを一定数作成し，学習用データセットに追加した．ここで，1 つのコードクローンを作成する際，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，作成および追加するコードクローンの数は 4.3.2 節のデータセット構築手法に依存する．これを選択した 20 個のクローン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クローン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，20 種類のクローンセットを全て目視で確認し，異なるクローンセットに属するメソッド間には、実現機能に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

表 4.2: 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 |

ソッドを作成した場合でも元のメソッドと構文的に類似しているため，元のメソッドと同じクローンセットに含まれる．

### 4.3.2 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通りである．

**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クローンセットにつき $n$ 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．このとき，20 種類のクローン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 ノードの総数が各クローンセットにつき約 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．このとき，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，4.2.2 節 STEP A4 では 10 個のメソッドを新たに追加する．従って，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 4.2 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．

### 4.3.3 実験手順

本評価実験は以下の手順で行う．

**(1)** 4.3.2 節で説明した 4 つの手法に基づいて，4.3.1 節の通りに選択したクローンセット 20 個から，学習用データセットを構築する．

表 4.3: ソースコード分類の精度

| 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 |

**(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，4 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．

### 4.3.4 実験結果

各データセット構築手法の分類精度を表 4.3 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，4.2.2 節の STEP A5 によってモデルの学習とコードクローンの追加が繰り返されたときの分類精度の変化を図 4.5 に示す．この図より，STEP A5 によって動的にコードクローン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．

### 4.3.5 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 3 つと提案手法の計 4 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 4.3 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ソースコード分類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クローンセットが存在した．これは 4.1 節で説明した不均衡データ問題の特徴であり，ベースライン手法では不均衡データ問題を解決できなかった．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クローンセットは存在しなかった．これにより，提案手法を用いることで不均衡データ問題を解決でき，分類精度を向上させられることが明らかになった．

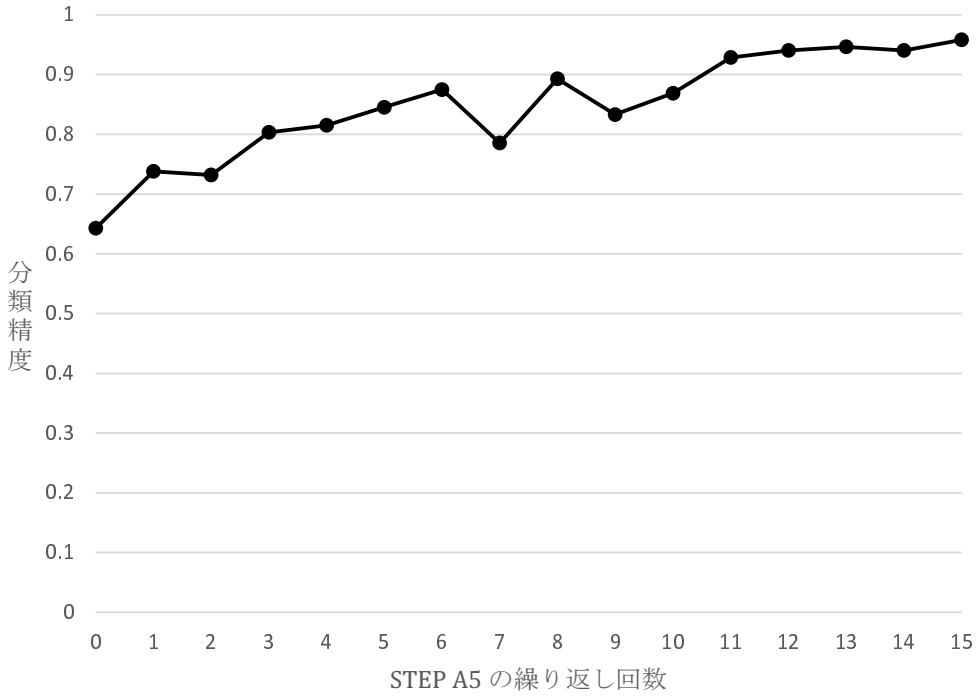


図 4.5: STEP A5 の繰り返し回数と分類精度

また，図 4.5 から分かるように，ソースコード分類モデルの学習結果に基づいたコードクローン追加 (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 1 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 4.5 において，分類精度が低下する場合がある．これは，STEP A4 においてコードクローンを学習用データセットに追加した際に不均衡データ問題による精度低下が再発したためだと考えられる．しかし，その後も STEP A5 を繰り返すことでソースコード分類精度は向上しているため，この不均衡データ問題は一時的なものであり，特に問題はないと考えられる．

次に，本評価実験では学習用データセットのソースコードの個数を単調に増加させたが，逆に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 4.5 の通り，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，正しく分類できたクローンセットの方が多いことを確認した．そのため，ソースコードの個数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 より低い場合にはソースコードの個

表 4.4: 提案手法による改善後のデータセットの詳細

| 統計量 | 値 |
|---|---|
| クローンセットの個数 | 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 |

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 4.4 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 のノード数に各クローンセット間の不均衡が存在する．しかし，本章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を実行できることが分かった．

## 4.4 妥当性の脅威

本章の提案手法に関する妥当性の脅威として，4 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．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．

2 つめは，評価実験で用いたクローンセットが 20 個と，比較的少ない点である．実験に使用するクローン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さらに繰り返し回数が必要になると考えられる．4.2.2 節の STEP A を繰り返す毎に，図 4.5 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，用いるクローン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．

3 つめは，評価実験における提案手法の初期状態に Node-oriented を採用していない点である．深層学習を用いてソースコード分類に取り組んだ既存研究 [12,21,45,46,

53,72] で用いられているランダムサンプリングは，本章における Method-oriented である．このように既存研究では Method-oriented がよく用いられているため，本提案手法でも，初期状態として Method-oriented を採用した．提案手法の初期状態を Node-oriented に設定した場合，ソースコード分類精度が向上するようにメソッドの追加が繰り返され，初期状態を Method-oriented に設定した場合のソースコード分類精度に近づくと考えられるが，今後検証する必要がある．

4 つめは，本章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コードクローンを深層学習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 章において評価実験を行った．この評価実験ではまず，本章の提案手法で 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[*5] と LibreSSL[*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持つメソッドを評価用データセットに追加した．さらに OpenSSL と無関係なソフトウェアの 1 つである Apache httpd[*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持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，本章の評価実験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．本章の提案手法に対する評価実験の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．

## 4.5 関連研究

### 4.5.1 不均衡データの効率的な学習

4.1 節で説明した通り，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深層学習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 [67] 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深層学習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 [66] は，ブートストラップ法を畳み込みニューラル

[*5] https://boringssl.googlesource.com/
[*6] https://www.libressl.org/
[*7] https://httpd.apache.org/

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深層学習モデルの学習手法を提案している．Chen と Shyu [8] 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 2 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．本章の提案手法は，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．

### 4.5.2 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 [45] 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 [72] 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）[61] 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 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，本章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．

## 4.6 まとめ

本章では，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ソースコード分類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，作成されたコードクローン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 4 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ソースコード分類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，提案手法を用いてソースコード分類モデルの学習とコードクローン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．

# 第 5 章

# 深層学習を用いたソースコード分類手法の比較調査

## 5.1　まえがき

ソフトウェア開発の生産性を向上させるため，開発者はソフトウェアの再利用を行う [20, 59]．ソフトウェアの再利用や検索を支援するための手法のひとつにソースコード分類があり，現在まで，多くのソースコード分類手法が提案されてきた [12, 21, 33, 38, 45, 46, 53, 63, 72, 74]．特に近年では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 [12, 21, 38, 45, 46, 53, 63, 72, 74]．これらの既存手法は様々なニューラルネットワークにソースコード表現を学習させることで実現されている．しかし，ニューラルネットワークやソースコード表現がどのようにソースコード分類の精度に影響し，どのニューラルネットワークやソースコード表現を利用することで高精度なソースコード分類の実現ができるかは明らかになっていない．また，深層学習を用いた既存のソースコード分類モデルは複雑な構造であるため，どのニューラルネットワークやソースコード表現が高精度なソースコード分類に有効かを把握することが難しい．

そこで本章では，高精度なソースコード分類に有効なニューラルネットワークとソースコード表現の組み合わせについて調査するために，深層学習を用いたソースコード分類手法の精度を比較する．本章では，比較調査を行うにあたってリサーチクエスチョン（以降，RQ）を設定した．

RQ　高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組み合わせは何か

この RQ に回答するためにまず，既存のソースコード分類手法でよく用いられるニューラルネットワークである，順伝播型ニューラルネットワーク，再帰型ニューラルネットワーク，グラフ畳み込みネットワークの 3 つを選択した．次に，選択したニュー

ラルネットワークにソースコードのトークン列または抽象構文木（Abstract Syntax Tree，以降，AST）を学習させ，計 6 種類のソースコード分類手法の精度を比較した．

その結果，再帰型ニューラルネットワークにソースコードのトークン列を学習させた手法の分類精度が最も高いことが分かった．

以上の調査によって，深層学習を用いたソースコード分類手法の中で最も分類精度が高い手法が明らかになった．しかし，深層学習を利用せずに高精度なソースコード分類が実行できる可能性がある．この前提を確認するために，深層学習を用いるソースコード分類手法と深層学習を用いないソースコード分類手法の精度の比較を行った．その結果，深層学習を用いる手法のほうが分類精度が高く，深層学習はソースコード分類に有効であることが明らかになった．

本章の貢献は以下の点である．

- 既存研究で広く利用されている 3 種類のニューラルネットワークに対してソースコードのトークン列または AST を学習させ，計 6 種類のソースコード分類手法の精度比較を行うことで，どのソースコード分類手法の精度が高いかを調査した．その結果，再帰型ニューラルネットワークにソースコードのトークン列を学習させる手法の分類精度が最も高いことが分かった．
- 深層学習を用いたソースコード分類手法と用いない手法の精度比較を行った．その結果，深層学習を用いたソースコード分類手法のほうが精度が高く，深層学習はソースコード分類に有効であることが分かった．

以降，5.2 節では，深層学習を用いたソースコード分類手法の比較調査の説明と結果の考察を行い，高精度なソースコード分類を実現することができるニューラルネットワークとソースコード表現の組み合わせについて述べる．5.3 節では，深層学習を用いる手法と用いない手法の分類精度の比較を行い，深層学習を用いる手法が高い精度を示すか確認する．5.4 節では，本章で行った比較調査に関する妥当性の脅威について述べる．5.5 節では，関連研究として，深層学習を用いてソースコード解析を行う既存研究について述べる．最後に，5.6 節で本章のまとめについて述べる．

## 5.2 深層学習を用いたソースコード分類手法の精度比較調査

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 [12, 21, 38, 45, 46, 53, 63, 72, 74]．これらの手法における深層学習モデルは複雑な構造をしており，様々なニューラルネットワークやソースコード表現を利用する．しかし，これらのニューラルネットワークやソースコード表現がどの程度ソースコード分類の精度に影響を及ぼすかは明らかになっていない．そこで本章では，既存研究で広く利用されているニューラルネットワークとソースコード表現を用いたソースコード分類

手法の精度の比較を行う．この比較を行うことで，高い精度でソースコード分類ができるニューラルネットワークとソースコード表現の組み合わせについて調査する．本章では以上の調査を行うにあたって，5.1 節で述べた RQ を設定した．

### 5.2.1 ベンチマーク

本章では, ソースコード分類のベンチマークとして, BigCloneBench（以下, BCB）[60] を用いた．BCB とは，Java で開発されたオープンソースソフトウェアからメソッド単位でソースコードを収集したベンチマークである．BCB の開発者らによって収集されたメソッドは，そのメソッドが実現する機能に基づき，43 種類の機能クラスに分類されている．そのため，BCB はソースコード分類手法の評価に用いることが可能である．BCB は開発者らが約 6 万個のメソッドを目視確認して作成した信頼性の高い大規模ベンチマークであるため，本章ではこのベンチマークを利用する．

### 5.2.2 比較対象のソースコード分類手法

2.2 節では，ソースコード分類手法に広く利用されている 3 つのニューラルネットワークについて説明した．本章では，それぞれのニューラルネットワークに対して 2 つのソースコード表現（ソースコードのトークン列または AST）の各々を学習させ，計 6 種類のソースコード分類手法の精度を比較する．そのため，本節ではその 6 種類のソースコード分類手法について説明する．

ソースコード表現には，トークン列と AST の他に制御フローグラフやデータフローグラフなどがある．この 2 つのソースコード表現を生成するためにはソースコードのコンパイルが必要である．しかし，5.2.1 節で説明したように BCB はオープンソースソフトウェアからメソッド単位でソースコードを収集したベンチマークであるため，BCB に含まれている個々のソースコードをコンパイルすることが困難である．そのため，本章の比較調査ではコンパイルが必要なソースコード表現である制御フローグラフやデータフローグラフを利用しない．

本章の比較調査で利用した深層学習フレームワークは PyTorch1.5.0[*1]，各ニューラルネットワークで用いる活性化関数は ReLU，損失関数は CrossEntropy, 最適化アルゴリズムは Adam である．各手法で用いるニューラルネットワークの隠れ層の層数とノード数は，3.3.1 節や 4.3 節と同様にグリッドサーチによって適切な値を決定した．また，埋め込みベクトルの次元数は，3.3.1 節と同様に 300 に設定した．

[*1] https://pytorch.org/

FNN+Token

本手法では，2.2.1 節で説明した FNN に，Doc2Vec [37] により生成したトークン列のベクトルを学習させる．Doc2Vec は教師なし学習によって文書のベクトルを生成する手法である．ソースコードのトークン列はトークンの並び順に意味があるため，単語の前後関係を踏まえ，系列全体のベクトル化を行うことが可能な手法を利用する必要がある．よって本手法では Doc2Vec を利用してメソッドをベクトル化する．

具体的にはまず，学習用データセット（5.2.3 節参照）に含まれるメソッドを，javalang[*2]を用いてトークン列にする．次に，そのトークン列を文書，トークンを単語として扱い，Doc2Vec を用いてメソッドをベクトル化する．Doc2Vec ベクトルの次元数は 300，FNN の隠れ層は 4 層，隠れ層のノード数は 128 とした．

FNN+AST

本手法は，学習させるソースコード表現以外の設定は上記の FNN+Token と同じである．本手法では，2.2.1 節で説明した FNN に，Doc2Vec により生成した AST のベクトルを学習させる．具体的にはまず，学習用データセット（5.2.3 節参照）に含まれるメソッドを，Eclipse JDT[*3]の ASTParser を用いて AST に変換する．次に，AST ノードを深さ優先探索順に並べた列を文書，AST ノードを単語として扱い，Doc2Vec を用いてメソッドをベクトル化する．

LSTM+Token

本手法では，メソッドのトークン列を学習データとすることで，2.2.2 節で説明した LSTM にトークンの順序関係を学習させる．メソッドのトークン列は javalang を用いて生成する．LSTM の埋め込み層の次元数は 300，LSTM の隠れ層のノード数は 128 である．

LSTM+AST

本手法では，学習させるソースコード表現以外の設定は上記の LSTM+Token と同じである．本手法は，学習させるソースコード表現を AST ノードの深さ優先探索順列にすることで，LSTM に AST ノードの順序関係を学習させる．AST ノードの深さ優先探索順列は Eclipse JDT の ASTParser を用いて生成する．

GCN+Token

本手法では，2.2.3 節で説明した GCN にメソッドのトークン列を学習させる．GCN でトークン列を学習させるためにはトークン列をグラフで表現する必要があるため，

[*2] https://github.com/c2nes/javalang
[*3] https://www.eclipse.org/jdt/

今回はトークン列に含まれる各トークンをノードとして扱い，前後のトークンをエッジで繋いだ 1 直線のグラフを扱う．GCN の実装には PyTorch Geometric[*4]を利用する．本手法におけるトークン列のグラフは無向グラフとみなし，エッジの重みは全て同じとする．ノードのベクトル化は Word2Vec [44] を用いて行う．具体的には，トークン列を文書，トークンを単語として扱い，Word2Vec を適用して各トークンのベクトルを生成し，各トークンに対応するノードに，生成したベクトルを割り当てる．メソッドのトークン列は javalang を用いて生成する．Word2Vec で生成するノードベクトルの次元数は 300，GCN の畳み込み層は 4 層，GCN の隠れ層のノード数は 128 とする．

GCN+AST

本手法では，学習させるソースコード表現以外の設定は上記の GCN+Token と同じである．本手法は，学習させるソースコード表現をメソッドの AST とする．これにより GCN は，ある AST ノードの周辺に出現するノードの特徴を学習することができる．本手法における AST は無向グラフとみなし，エッジの重みは全て同じとする．AST ノードのベクトル化は Word2Vec [44] を用いて行う．具体的には，AST ノードを深さ優先探索順に並べ，各ノードを単語とみなし，Word2Vec を適用する．また，メソッドの AST は Eclipse JDT の ASTParser を用いて生成する．

ソースコードをグラフで表現する方法は AST の他に制御フローグラフやデータフローグラフがある．しかし，制御フローグラフやデータフローグラフを生成するにはソースコードのコンパイルが必要であり，5.2.1 節で説明した BCB にはコンパイルが困難なソースコードが含まれる．そのため本手法では，生成にソースコードのコンパイルが不要な AST を対象にする．

### 5.2.3 調査方法

本章の比較調査では，ソースコード分類手法の評価尺度として Top-k を用いる．本章の比較調査における Top-k とは，各ソースコード分類手法が，評価用データセットに含まれる各メソッドに対して機能クラス毎の分類確率を計算し，その確率が高いクラス順にランキングにしたとき，正解クラスが k 位以内に含まれている割合である．ここで正解クラスとは，各メソッドが持つ機能に基づいて，5.2.1 節で説明した BCB によって定められた機能クラスである．

Top-k の算出手順を図 5.1 に示す．Top-k の算出は以下の 5 つの手順に従って行う．

**(1-1)** 5.2.1 節で説明した BCB のメソッドを機能クラス毎に分割し，各機能クラスに固有の ID を割り当てる．

[*4] https://github.com/rusty1s/pytorch_geometric

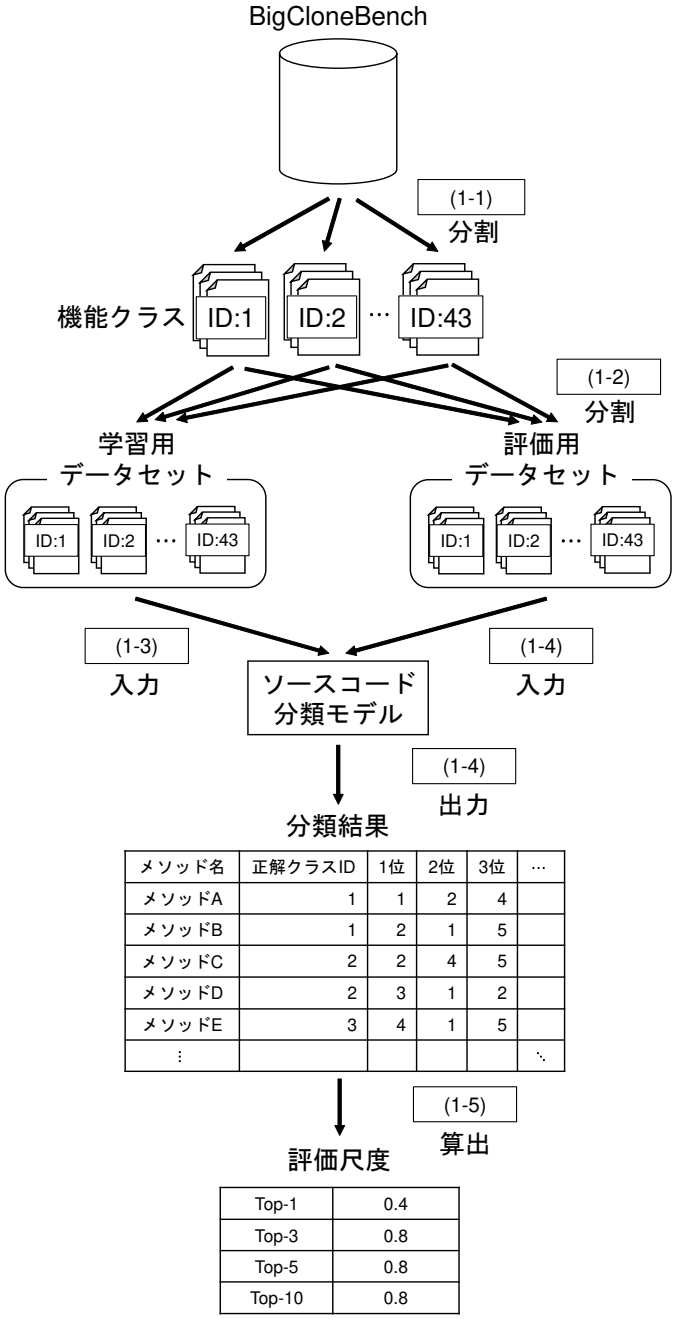


図 5.1: Top-k 算出方法の概要

**(1-2)** 各機能クラスのメソッドを 8:2 の割合でランダムに分割し，8 割を学習用データセット，2 割を評価用データセットとする．

**(1-3)** 学習用データセットのメソッドからソースコード表現を生成し，ニューラルネットワークに学習させる．

表 5.1: 各分類手法の分類精度

| 分類手法 | 深層学習 | Top-1 | Top-3 | Top-5 | Top-10 |
|---|---|---|---|---|---|
| FNN+Token | ○ | 0.575 | 0.766 | 0.830 | 0.911 |
| FNN+AST | ○ | 0.644 | 0.803 | 0.853 | 0.922 |
| LSTM+Token | ○ | **0.943** | **0.980** | **0.985** | 0.991 |
| LSTM+AST | ○ | 0.939 | 0.977 | 0.981 | 0.991 |
| GCN+Token | ○ | 0.772 | 0.927 | 0.967 | 0.989 |
| GCN+AST | ○ | 0.803 | 0.948 | 0.972 | **0.993** |
| FaCoY | × | 0.840 | 0.940 | 0.950 | 0.958 |
| Siamese | × | 0.848 | 0.897 | 0.908 | 0.925 |

**(1-4)** 学習用データセットのメソッドを学習したニューラルネットワークを用いて，評価用データセットに含まれる各メソッドの分類を行う．

**(1-5)** 分類結果から Top-k を算出する．

本章の比較調査では，5.2.2 節で説明した 6 種類のソースコード分類手法に対して，以上の手順に従って Top-1，Top-3，Top-5，Top-10 をそれぞれ算出し比較する．ただし，(1-2) では各機能クラスのメソッドの分割をランダムに行うため，分割の試行ごとに分類精度が変わる可能性がある．本章の比較調査では，6 種類の分類手法で同じデータセットを用いて学習と評価を行うために，共通のランダムシード値を用いて各機能クラスの分割を行う．また，各機能クラスに属するメソッドの数は同じではないため，学習用データセットのメソッドを全て使うと 4.1 節で述べた不均衡データ問題により深層学習モデルの分類精度が低下する．そのため，4 章で提案した手法を応用して各機能クラスのメソッド数を調整した．具体的には最初に各機能クラスからランダムにメソッドを 50 個選択して学習を行い，相対的に正答率が低い機能クラスは次の学習で使用するメソッドを 10 個ずつ増やしていった．学習に使用するメソッドが足りなくなった場合は，コードミューテーションを用いてメソッドを増やし，学習に使用した．以上の手順で，高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組み合わせについて調査する．

### 5.2.4 調査結果と考察

各ソースコード分類手法の精度を表 5.1 に示す．この表では Top-k で最も高い数値を太字で表している．表 5.1 から分かるように，各分類手法の中では，LSTM+Token が Top-1，Top-3，Top-5 で分類精度が最も高く，LSTM+AST がそれに匹敵する分類精度だった．Top-10 に関しては GCN+AST の分類精度が最も高く，LSTM+Token

や LSTM+AST がそれに匹敵する結果となった．以上の結果から RQ の回答は，「LSTM とトークン列の組み合わせが最も高精度なソースコード分類を実現できる組み合わせであり，LSTM と AST の深さ優先探索順列の組み合わせや GCN と AST の組み合わせも比較的高い精度のソースコード分類を実現することができる」となった．

また，分類精度は，学習させるソースコード表現よりも用いるニューラルネットワークに大きく影響されることが分かった．LSTM を用いた手法は，学習させるソースコード表現に関わらず分類精度が平均的に高かった．GCN を用いた手法は，Top-1 が LSTM を用いた手法に大きく劣り，Top-3，Top-5 は LSTM を用いた手法と比べるとやや劣るが，Top-10 は LSTM を用いた手法と同じくらいの分類精度であり，FNN を用いた手法と比べると平均的に分類精度は高かった．FNN を用いた手法は，他のニューラルネットワークを用いた手法と比べると分類精度は低かった．よって，LSTM が最も高精度なソースコード分類を実現可能なニューラルネットワークであることが分かった．このような結果になった理由のひとつとして考えられるのが，トークンやノードの順序関係である．LSTM は 2.2.2 節で説明した通り，トークンやノードの入力順序が出力に影響する．しかし，FNN や GCN はソースコード表現を深層学習モデルへの入力形式に変換するときに，トークンやノードの順序関係が失われて隣接情報だけになる．よって，LSTM だけがトークンやノードの順序関係を学習することができたため，LSTM のソースコード分類精度が最も高かったと考えられる．

次に，ニューラルネットワークとソースコード表現の組み合わせ方法について考察する．まず，FNN を用いた手法では，FNN+Token よりも FNN+AST のほうが分類精度が高かった．このような結果になった原因は Doc2Vec にあると考えられる．Doc2Vec は $n$-gram の考え方をベースにした教師なし学習アルゴリズムで文書をベクトル化する．よって，トークン列の学習を行う際，括弧やセミコロンなど，ソースコードに頻繁に出現するトークンが Doc2Vec の学習のノイズになり，生成されるソースコードベクトルがソースコードの特徴をうまく表現できていなかったと考えられる．その一方で，AST の深さ優先探索順列には，学習のノイズになり得るノードなどの要素がトークン列より少ないため，トークン列に比べてソースコードの特徴を表現する良いソースコードベクトルを生成できたと考えられる．

次に，LSTM を用いた手法では，わずかに LSTM+Token のほうが分類精度が高かった．LSTM+AST において，AST を LSTM に学習させるためには AST を何らかの系列に変換する必要があるため，今回は AST の深さ優先探索順列を学習させている．しかし，AST の深さ優先探索順列からソースコードを完全に復元することはできないため，元のソースコードの情報はわずかに欠落しており，その分だけ分類精度が低下したと考えられる．その一方で，LSTM+Token ではトークン列を学習するため，空白やインデントなどのフォーマット情報を除き，ソースコードをそのまま学習可能である．また，LSTM は Doc2Vec と異なり，トークンの長期的な依存関係を学習することができる．そのため，LSTM の学習は，括弧やセミコロンなどの頻出トー

クンにあまり影響されないと考えられる．よって，AST よりもトークン列のほうが LSTM との相性が良いと考えられる．

GCN を用いた手法では，GCN+Token よりも GCN+AST のほうが分類精度が高かった．トークン列から生成したグラフには分岐がないため，グラフを学習可能なニューラルネットワークの利点を生かすことができていない．また，Doc2Vec の場合と同様に，頻出するトークンが学習ノイズになっている可能性が考えられる．その一方で，AST はグラフ形式の表現であるため，変形せずに GCN に学習させることができ，学習ノイズになり得るノードがトークン列に比べて少ない．よって，トークン列よりも AST のほうが GCN との相性が良いと考えられる．

## 5.3　深層学習を用いない分類手法との比較

5.2 節では深層学習を用いる分類手法の精度比較を行い，LSTM を用いる手法が最も精度が高いことが分かった．しかし，深層学習を利用せずに高精度なソースコード分類ができる可能性がある．この前提を確認するために，本節では，5.2.2 節で説明した深層学習を用いるソースコード分類手法と，深層学習を用いないソースコード分類手法の中で代表的な手法の精度比較を行う．また，ソースコード分類に深層学習を用いる利点について考察する．

### 5.3.1　深層学習を用いない分類手法

本節では，深層学習を用いる手法との比較対象として選択した，深層学習を用いないソースコード分類手法について説明する．これらの手法は，構文的に類似したソースコードだけでなく意味的に類似したソースコードを検索できる最新手法であり，実装が公開されている[*5][*6]ため，比較対象として選択した．

#### FaCoY

FaCoY [35] は，深層学習を用いない意味的コードクローン検索手法である．この手法は，開発者向け Q&A サイト StackOverflow[*7]を用いて意味的コードクローンを検索する手法である．具体的に FaCoY はまず，入力として与えられたコード片のコードクローンが含まれる回答文を，ソースコードの類似度が高い順に $n_s$ 個だけ StackOverflow から検索する．このとき，ソースコードの類似度は，2 つのソースコードを TF-IDF（Term Frequency - Inverse Document Frequency）[55] でベクトル化したときの 2 つのベクトル間のコサイン類似度を用いて計算する．次に，検索された

[*5] https://github.com/FalconLK/facoy
[*6] https://github.com/UCL-CREST/Siamese
[*7] https://stackoverflow.com/

回答文に対応する質問文をクエリとし，再び StackOverflow から回答文の検索を行い，検索結果の先頭から $n_q$ 個の回答文に含まれるソースコードを合計 $n_c$ 個まで検索結果として出力する．このような手順で，FaCoY は入力として与えたコード片の意味的コードクローンを検索する．この手法のキーアイデアは，“Q&A サイトにおいて類似した質問文に対応する形で出現するソースコードは意味的に類似している” であり，このアイデアによって構文的なコードクローンだけでなく意味的なコードクローンも検索することができる．この手法では，入力コード片と検索されたソースコードに対して TF-IDF を適用してベクトル化し，2 つのベクトル間のコサイン類似度を算出し大きい順に並べることで，検索されたソースコードのランキングを作成する．

FaCoY のコード検索性能に影響するハイパーパラメータには，前段落で述べた $n_s$，$n_q$，$n_c$ の 3 つがある．本章の比較調査では，$n_s = 3$，$n_q = 3$，$n_c = 100$ で比較を行う．この設定は，文献 [35] における FaCoY の BCB に対する適用実験で用いられたものである．

#### Siamese

Siamese [48] は，深層学習を用いない意味的コードクローン検索手法である．この手法は，$n$-gram を用いてコードクローンを検索する手法であり，BCB を用いた評価実験で，意味的に類似しているが構文的な類似度が低いコードクローンも高い精度で検索できることが確認されている．この手法ではまず，メソッドを 4 種類の異なる配列（トークン列，$n$-gram の配列，識別子・文字列・型を正規化した後の $n$-gram の配列，括弧・セミコロン以外を正規化した後の $n$-gram の配列）で表現する．次に，検索対象メソッドと入力メソッドの組において，それぞれの表現方法に対してトークンや $n$-gram の出現頻度を考慮したスコア $\theta$（0〜100%）を算出する．最後に，閾値 $T$ よりもスコア $\theta$ が高いメソッドの組をコードクローンとして出力する．

Siamese のコード検索性能に影響するハイパーパラメータには，$n$-gram を用いる際の $n$ の値と前段落で述べた閾値 $T$ の 2 つがある．本章の比較調査では Siamese のデフォルト値である $n = 4$，$T = 10\%$ で設定し，比較を行う．

### 5.3.2 比較方法

5.3 節では，深層学習を用いない意味的コードクローン検索手法をソースコード分類に適用し，5.2 節で用いた 6 種類のソースコード分類手法と精度を比較する．5.3 節で利用するベンチマーク及び評価尺度は，5.2 節の比較調査と同様である．

本節で選択した 2 つの意味的コードクローン検索手法において，検索クエリのソースコードと検索結果として出力されたソースコードは意味的に類似している．また，1.3 節で説明したソースコード分類の定義では，意味的コードクローンは同じクラスに分類される．そのため本節では，検索クエリのソースコードが，検索結果のソース

コードが属するクラスに分類されるとみなすことで，意味的コードクローン検索手法をソースコード分類に適用する．また，検索結果のソースコードは，検索クエリとの類似度順にランキング化されて出力される．そこで本節では，ランキングに含まれる検索結果のソースコードを，そのソースコードが属するクラスに置き換えることで，意味的コードクローン検索手法によって出力されたランキングを基に，5.2.3 節で説明した Top-k を算出するためのランキングを作成する．

選択した意味的コードクローン検索手法をソースコード分類に適用し，5.2.3 節で説明した Top-k の算出に必要なランキングの作成手順を図 5.2 に示す．ランキング作成は以下の 4 つの手順に従って行う．ここで，以下の手順に出現する機能クラス ID は 5.2.3 節の手順 (1-1) で BCB によって定められた機能クラスに対して割り当てた ID であり，学習用データセット，評価用データセットは 5.2.3 節の手順 (1-2) で作成したデータセットである．また，図 5.2 の正解クラス ID は，評価用データセットの各メソッドが持つ機能に基づいて，5.2.1 節で説明した BCB によって定められた機能クラスの ID である．

**(2-1)** 評価用データセットに含まれる各メソッドを検索クエリに設定し，学習用データセットに対してコードクローン検索を実行する．

**(2-2)** 検索結果が出力される．この時 FaCoY の場合は，検索結果のソースコードがランキング形式で出力される．Siamese の場合は，検索クエリのコードクローンがスコアと共に出力されるので，出力されたコードクローンを類似度スコアが高い順に並べることでランキングを作成する．

**(2-3)** 手順 (2-2) で得られたランキングに含まれる各ソースコードを，そのソースコードが属する機能クラスの ID に置き換える．

**(2-4)** 5.2.3 節の手順 (1-4) で作成される分類結果では，同じ機能クラス ID は 2 つ以上重複して存在しない．従って，Top-k の算出における条件を揃えるために，ランキングで重複した機能クラス ID のうち最も高い順位のもの以外を除外することで，再度ランキングを作成する．

以上の手順で作成したランキングを用いて Top-k を計算し，深層学習を用いる手法とソースコード分類精度を比較する．

### 5.3.3 比較結果と考察

深層学習を用いる分類手法と用いない分類手法の分類精度を表 5.1 に示す．この表の下 2 行は深層学習を用いないソースコード分類手法を示している．また，この表では Top-k で最も高い数値を太字で表している．


| メソッド名 | 正解クラスID | 1位 | 2位 | 3位 | … |
|---|---|---|---|---|---|
| メソッドA | 1 | メソッドa | メソッドb | メソッドc | |
| メソッドB | 1 | メソッドa | メソッドc | メソッドb | |
| メソッドC | 2 | メソッドc | メソッドd | メソッドe | |
| メソッドD | 2 | メソッドc | メソッドd | メソッドb | |
| メソッドE | 3 | メソッドe | メソッドf | メソッドg | |
| ⋮ | | | | | ⋱ |

| メソッド名 | 正解クラスID | 1位 | 2位 | 3位 | … |
|---|---|---|---|---|---|
| メソッドA | 1 | 1 | 1 | 2 | |
| メソッドB | 1 | 1 | 2 | 3 | |
| メソッドC | 2 | 2 | 3 | 2 | |
| メソッドD | 2 | 2 | 3 | 1 | |
| メソッドE | 3 | 2 | 1 | 3 | |
| ⋮ | | | | | ⋱ |

| メソッド名 | 正解クラスID | 1位 | 2位 | 3位 | … |
|---|---|---|---|---|---|
| メソッドA | 1 | 1 | 2 | … | |
| メソッドB | 1 | 1 | 2 | 3 | |
| メソッドC | 2 | 2 | 3 | … | |
| メソッドD | 2 | 2 | 3 | 1 | |
| メソッドE | 3 | 2 | 1 | 3 | |
| ⋮ | | | | | ⋱ |

図 5.2: 意味的コードクローン検索手法を用いてランキングを作成する手順

表 5.1 から分かるように，Top-1，Top-3，Top-5，Top-10 の全てで LSTM を用いる手法の精度は深層学習を用いない分類手法を上回った．LSTM を用いた手法のソースコード分類精度が深層学習を用いない手法の分類精度より高かった理由として，LSTM で学習可能な情報であるトークンや AST ノードの順序関係とその長期的な依

存関係が BCB のソースコードを分類するのに重要だったことが考えられる．しかし，深層学習を用いない手法では，トークンやノードの関連性や長期的な依存関係を捉えた分類を行うことは困難である．Siamese は $n$-gram を用いているため，トークンの短期的な依存関係を捉えることはできるが，長期的な依存関係を捉えることはできない． そのため，分類精度が低くなったと考えられる．また，FaCoY はソースコードの類似性を捉えるために Q&A サイトにおける質問と回答を用いている．類似した回答文に含まれているソースコードに類似性があることは確かだが，この類似性が BCB のソースコードを分類するのに必要な類似性とは異なっていたために分類精度が低くなった可能性がある．

以上のように，LSTM を用いた分類手法は，本節で選択した深層学習を用いない既存手法よりも高精度なソースコード分類を行うことができることが分かった．

## 5.4 妥当性の脅威

本章の比較調査に関する妥当性の脅威として 4 点を挙げることができる．

1 つ目は，比較対象に設定したニューラルネットワークの種類が少ない点である．本章では広く利用されている 3 つのニューラルネットワークを調査対象としたが，ソースコード分類に適用可能なニューラルネットワークは他にも様々なものが考えられるため，今後検証していく必要がある．

2 つ目は，ソースコード分類手法の精度比較調査を行う際に，1 つのベンチマークを使用したため，調査結果が使用したベンチマークに強く依存している可能性がある．しかし，本章の比較調査で使用した BCB は約 6 万のメソッドを開発者らが手作業で確認することによって作成された非常に大規模なベンチマークであり，様々なソフトウェアのソースコードが含まれている．そのため，BCB を用いることでソースコード分類の一般的な評価を行うことができると考えられるが，今後は他のベンチマークを用いた比較調査を実施し，本調査と同様の傾向が得られるかを調査する必要がある．

3 つ目は，今回の調査結果は様々な要因によって変わりうる点である．ニューラルネットワークの学習結果に影響する要因として，隠れ層の数やノード数などのハイパーパラメータ，活性化関数，損失関数，最適化アルゴリズム，データセットの分割方法が挙げられる．また，ソースコードをニューラルネットワークに入力するために実施する前処理の手法も，結果に影響する要因であると考えられる．FNN を使用した手法におけるメソッドのベクトル化手法や，GCN を使用した手法におけるノードのベクトル化手法が結果に影響すると考えられる．また，トークン列を学習する手法については，変数名を正規化することによって結果が変わると考えられる．AST を学習する手法については，使用する構文解析器によって出力される AST が異なるため，結果が変わると考えられる．本章の比較調査では，比較する分類手法の間で以上の要因を揃えているため，比較結果に一定の意味はあると考えられる．しかし，あくまでも本

章の比較調査の結果は 5.2 節及び 5.3 節で説明した設定の下での結果であるため，今後は，本章では調査しなかったパラメータ設定や前処理手法についても調査する必要がある．

4 つ目は，サポートベクターマシン（SVM）やランダムフォレスト（RF）など，機械学習の手法との比較を行っていないため，機械学習の手法が LSTM より優れた分類精度になる可能性がある点である．この点に関しては，既存研究 [45, 72] にて既に深層学習を用いたソースコード分類手法と機械学習の手法のソースコード分類精度が比較されていたため，本章の比較調査で新たに比較を行うことはせず，機械学習を行わずに意味的コードクローンを検索可能な最新手法である FaCoY と Siamese との比較を行った．ただし，既存研究 [45, 72] で用いられたデータセットは BCB ではない．また，比較対象として選択されたのは SVM のみである．そのため，本章の比較調査で行ったソースコード分類を行った場合，SVM や RF 等の機械学習の手法のほうが優れた分類精度になる可能性があるため，今後調査する必要がある．

## 5.5 関連研究

近年，深層学習を用いてソースコードの解析を行う研究が発表されている．

まず我々は，過去に FNN を用いてコードクローンの検索を行う手法を提案した [12]．この研究では，FNN を用いたソースコード検索において，BoW と Doc2Vec の 2 つのソースコードベクトル化手法のどちらを用いた場合に精度が優れているかを比較しており，BoW が高い精度を示している．しかし，BCB はソースコード数が非常に多いデータセットであるため，本章の比較調査で BoW を採用した場合，BoW で生成したベクトルは次元数が膨大になり，学習に非常に時間がかかる．そのため，本章の比較調査では Doc2Vec を採用した．また，Saini ら [53] や Nafi [46] らはソースコードメトリクスを用いてソースコードをベクトル化し，FNN を用いてコードクローン検出を行っている．Saini ら [53] は，深層学習を用いない既存手法と FNN を用いた提案手法の精度を比較し，提案手法のコードクローン検出精度が優れていることを示している．また，Nafi ら [46] は，生成したベクトルのコサイン類似度を計算する手法と FNN を用いた手法を比較し，FNN を用いた手法のほうがコードクローン検出精度が優れていることを示している．本章の比較調査では，Saini ら [53] や Nafi ら [46] の研究のように深層学習を用いない既存手法との精度比較を行ったのに加え，ニューラルネットワークを利用した手法の精度比較を行った．

また，RNN を用いた研究が発表されている．Zhou ら [75] は深層学習を用いたソースコード補完手法を提案している．この手法では学習するソースコード表現としてクラス名・メソッド名・トークン列を利用し，トークン列の解析には RNN を用いている．White ら [65] は深層学習を用いたコードクローン検出手法を提案している．この手法ではトークンのベクトル化において RNN を利用し，このベクトルを用いて AST

のノードをベクトル化することで，AST の類似性を判定している．これら 2 つの手法は様々なソースコード表現を学習することで精度向上を目的としているのに対し，本章の比較調査では LSTM で学習するソースコード表現を 1 種類に限定することで，ソースコード表現と分類精度の関連性調査を目的としている，

また，GCN 以外の方法で AST を学習に用いた研究が発表されている．Mou ら [45] 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，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構文木の特徴を学習する TBCNN という手法を提案し，この手法をソースコード分類に適用している．Mou ら [45] は，SVM，DNN（本章の比較調査における FNN），RNN と TBCNN の分類精度を比較し，TBCNN のソースコード分類精度が優れていることを示している．また，TBCNN を除き，最適なベクトル化手法を選択した場合は FNN，SVM，RNN の順に分類精度が高いという結果も示している．ここで比較対象に選択されている RNN は木構造に対して用いる再帰型オートエンコーダであり，LSTM とは異なる．それに対して本章の比較調査では，精度の高かった FNN に加え，Mou ら [45] が比較対象に選択していないニューラルネットワークである LSTM と GCN を比較対象として選択し，トークン列と AST ノードの深さ優先探索順列を学習させた．その結果，LSTM にトークン列を学習させる手法が最も分類精度が高いという結果になった．Zhang ら [72] 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）[61] 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 ASTNN という手法を提案し，この手法をソースコード分類に適用している．Zhang ら [72] は，LSTM などのニューラルネットワークを用いた様々な手法や SVM と ASTNN のソースコード分類精度を比較している．ニューラルネットワークを単体で用いた手法や機械学習の手法については，トークン列に対する LSTM，SVM，制御フローグラフに対する GNN の順で分類精度が高いことを示している．GNN は GCN の前身となる，グラフを学習可能なニューラルネットワークである．また，彼ら [72] の研究では，LSTM に対するトークン列や，GNN に対する制御フローグラフやデータフローグラフなど，著者らによって適当だと考えられたニューラルネットワークとソースコード表現の組み合わせが比較対象に選択されている．それに対して本章の比較調査では，LSTM にグラフ由来のソースコード表現を学習させたり，GCN にトークン列由来のソースコード表現を学習させるなど，既存研究で比較対象に選択されていないニューラルネットワークとソースコード表現の組み合わせに対しても比較調査を行った．その結果，LSTM とトークン列の組み合わせなど，一般に既存研究で比較対象に選択されることの多い組み合わせは，選択されていない組み合わせに比べて分類精度が高いことが分かった．

また，GCN を用いた研究が発表されている．Hua ら [21] は，トークン列に LSTM を，AST に Tree-LSTM を，制御フローグラフに GCN を適用し，これら 3 つの

ニューラルネットワークを組み合わせることで，高い精度でコードクローン検出を行う FCCA という手法を提案した．そして評価実験では，深層学習を用いない既存手法や深層学習を用いた既存手法と比較して，FCCA のコードクローン検出精度が優れていることを示している．また，FCCA は LSTM，Tree-LSTM，GCN の 3 つのニューラルネットワークを組み合わせて利用しているが，これらの 3 つのニューラルネットワークを個別に用いた場合の精度比較も行っている．この結果，LSTM，Tree-LSTM，GCN の順にコードクローン検出精度が高いことが明らかになっている．Hua らの研究によって，トークン列と LSTM の組み合わせのコードクローン検出精度が高いことや，制御フローグラフと GCN の組み合わせの精度が比較的低いことが明らかになっているため，本章の比較調査では，彼らの研究の比較結果で最も高精度なトークン列と LSTM の組み合わせと，比較対象に選択されていない組み合わせを比較するため，トークン列，AST と FNN，LSTM，GCN を総当たりで組み合わせてソースコード分類精度の比較を行った．

以上のように，2.2 節で説明した 3 種類のニューラルネットワークの比較及び本章で比較対象に選択したソースコード表現の比較について関連研究で示されているのは，Hua ら [21] によって LSTM を用いた手法のほうが GCN を用いた手法よりもコードクローン検出精度が高いという結果のみである．そのため，本章では 2.2 節で説明した 3 種類のニューラルネットワークに 2 種類のソースコード表現を組み合わせた 6 種類のソースコード分類手法の精度を比較した．また，Saini ら [53] と Nafi ら [46] の研究では深層学習を用いない既存手法よりも FNN を用いた手法のほうがコードクローン検出精度が高いという結果が示されていたが，本章の調査結果によって，深層学習を用いない分類手法のほうが FNN を用いた手法よりも精度が高い場合があることが分かった．

## 5.6 まとめ

本章では，高精度なソースコード分類を実現できるニューラルネットワークとソースコード表現の組み合わせについて調査するため，深層学習を用いた 6 種類の手法をソースコード分類に適用し，ベンチマークに BCB を，評価尺度に Top-k を用いてソースコード分類精度を比較した．その結果，LSTM にトークン列を学習させる手法が最も高精度なソースコード分類を実現でき，LSTM に AST の深さ優先探索順列を学習させる手法や GCN に AST を学習させる手法も，比較的高精度なソースコード分類を実現できた．それに対し，FNN を用いた手法の分類精度はあまり高くなかった．また，深層学習を用いる手法と深層学習を用いない手法のソースコード分類精度を比較した．その結果，LSTM を用いてソースコードの構造情報を学習する深層学習の手法は，本章の比較調査で選択した深層学習を用いない手法よりも高精度なソースコード分類を実現できることが分かった．

# 第 6 章

# 回帰モデルを用いたコードクローン検出手法の提案と汎化性能の評価

## 6.1 まえがき

近年，大規模なソフトウェアを効率的に開発することが求められる中，意味的コードクローン検出技術は，ソフトウェアの開発や保守において重要な役割を果たす．意味的コードクローン検出技術を用いて，過去に開発されたソフトウェアの中から開発者が求める機能を持ったソースコードを効率的に検索し再利用することで，既存ソフトウェアを効率的に利用したソフトウェア開発を行うことができる．また，ソフトウェアの保守性を低下させる原因となるコードクローンを検出し集約することで，ソフトウェアの保守性を向上させることができる．

現在まで，深層学習を用いた意味的コードクローン検出手法が提案されている [38, 53, 63, 65, 72, 74]．例えば Zhang ら [72] は，独自に開発した深層学習モデルである ASTNN を用いて意味的コードクローン検出に取り組んでいる．評価実験では，オープンジャッジのソースコードを用いたデータセットと大規模コードクローンベンチマーク BigCloneBench（以下，BCB） [60] に対して ASTNN を用いたコードクローン検出を実行し，オープンジャッジデータセットに対しては 0.955，BCB に対しては 0.938 の F 値を記録している．

このように近年，深層学習を用いた意味的コードクローン検出手法が多く提案されている．しかし，既存手法には 2 つの問題点がある．1 つ目は，コードクローンの学習方法である．コードクローンの類似度は様々であり，構文的に類似していて簡単に識別できるコードクローンがある一方，構文的な類似度が低く，人によってコードクローンかどうか意見が別れるコードクローンも存在する．しかし，多くの既存手法では 2 値分類モデルが用いられているため，与えられたコード片のペアは，コードクロー

ンか非コードクローンかの 2 択で表現される．このように，明らかなコードクローンも人によって意見が別れるコードクローンも全て同じ重みで表現されるため，コードクローンの類似度情報が欠落した状態で学習を行っている．そのため，コードクローン検出精度が低下している可能性がある．

2 つ目は，学習用データセットと評価用データセットの間に依存関係があるため，深層学習モデルの汎化性能が正しく評価できていない点である．データセット間に依存関係がある状態とは，同じデータセットを分割するなどの方法で学習用データセットと評価用データセットを構築することにより，データの特徴がデータセット間で酷似している状態のことである．学習用データセットと評価用データセットの間に依存関係があると，深層学習モデルが過学習を起こしていた場合でも，評価用データセットに対する予測精度が高くなる．そのため，深層学習モデルの汎化性能を正しく評価することができない．例えば ASTNN [72] の場合，BCB に含まれるコードクローンと非コードクローンをそれぞれ 3:1:1 の割合に分割し，学習用データセット・検証データセット・評価用データセットとすることで，適合率・再現率・F 値はそれぞれ 0.998，0.884，0.938 を記録している．しかし，筆者が BCB のコードクローンを学習した ASTNN モデルを再現し，DeepSim [74] の評価実験に用いられたオープンジャッジデータセットを用いて評価したところ，適合率・再現率・F 値はそれぞれ 0.521，0.314，0.392 を記録した．このように，学習用データセットとの間に依存関係がある評価用データセットを用いた場合には検出精度が高くても，依存関係のない評価用データセットを用いると検出精度が大きく低下する場合がある．既存研究の評価実験では，同一データセットを分割して学習用データセットと評価用データセットを作成していることが多い．この手順で作成した学習用データセットと評価用データセットの間には依存関係があるため，評価用データセットに対するコードクローン検出が良い場合でも，深層学習モデルがデータセットに特化している可能性を否定できず，深層学習モデルの汎化性能についての評価は不十分である．

上記の 2 つの問題を解決するために，本章では，回帰モデルを用いたコードクローン検出手法を提案し，その汎化性能について評価する．この手法では，コードクローンの学習方法を改善するために，2 値分類モデルではなく回帰モデルを利用し，教師あり学習に使用する目的変数をコード片の類似度に変更する．このように学習させるコードクローンの重みを類似度に基づいて変更することで，コードクローンの類似度情報を利用した学習を行う．

評価実験では，未学習データに対する深層学習モデルの挙動を確認し汎化性能を評価するために，学習用データセットと評価用データセットを同一データセットから作成するのではなく，互いに依存関係のない 5 つのデータセットから作成した．そして，ASTNN などの 4 つの手法で用いられている深層学習モデルを 2 値分類モデルから回帰モデルに変更し，汎化性能を比較した．その結果，4 つの内 3 つの深層学習モデルの汎化性能が向上することが分かった．例えば ASTNN の場合，学習用データセット

として BCB のコードクローンを利用し，評価用データセットとしてオープンジャッジのソースコードを利用することによってデータセット間の依存関係を取り除くと，F 値は 0.938 から 0.392 に低下した．しかし，2 値分類モデルから回帰モデルに変更することで F 値は 0.392 から 0.694 に向上した．

本章の貢献は以下の 2 点である．

- 2 値分類モデルから回帰モデルに変更し教師あり学習の目的変数をコード片の類似度にすることで，学習におけるコードクローン毎の重みを設定した．その結果，4 つの深層学習モデルのうち 3 つで汎化性能が向上することを確認した．
- 未学習データに対する深層学習モデルの挙動を確認し，汎化性能を評価するために，学習用データセットと評価用データセットを同一データセットから作成するのではなく，依存関係のない 5 つのデータセットから作成し，評価実験を行った．

以降，6.2 節では，コードクローンの類似度を学習に反映するための提案手法について説明する．6.3 節では，深層学習モデルの汎化性能をより正確に評価するための実験について説明する．6.4 節では，本章に関する妥当性の脅威について述べる．6.5 節では，深層学習を用いた意味的コードクローン検出の関連研究を紹介する．最後に，6.6 節で本章のまとめについて述べる．

## 6.2 提案手法

本章では，回帰モデルを用いた意味的コードクローン検出手法を提案する．多くの既存の意味的コードクローン検出手法では，入力された 2 つのコード片がコードクローンか判定するために 2 値分類モデルを用いている．それに対して提案手法では，2 値分類モデルではなく回帰モデルを利用し，目的変数を抽象構文木（Abstract Syntax Tree，以降，AST）の類似度に変更することで，コードクローンの類似度を学習に反映する．このようにすることで，学習させるコードクローンの重みに類似度を反映することができる．

### 6.2.1 回帰モデルの適用

多くの既存の意味的コードクローン検出手法では 2 値分類モデルが用いられている．このモデルは，コードクローンのクラス $C_0$ とコードクローンのクラス $C_1$ に対する予測確率を出力し，確率が高いほうに入力された 2 つのコード片を分類することで，入力された 2 つのコード片がコードクローンかどうかを判定する．2 値分類モデルでは，コードクローンを全て同じクラス $C_1$ に分類することから，全てのコードクローンは同じ重みで表現される．しかし，コードクローンには類似度の違いがある．多くの

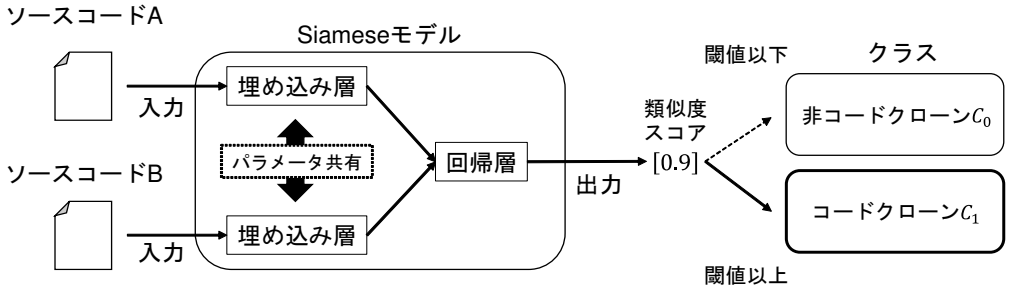


図 6.1: 提案モデル

人がコードクローンだと判断できるコードクローンがある一方，人によってコードクローンかどうか意見が分かれるコードクローンもある．それらのコードクローンを全て同じ重みで扱い，同じクラスに分類するため，コードクローンの類似度情報が欠落し，深層学習モデルのコードクローン検出性能を低下させている可能性が考えられる．

そのため，提案手法では 2 値分類モデルの代わりに回帰モデルを用いる．2 値分類モデルの目的変数は整数であるのに対し，回帰モデルの目的変数は実数である．そのため，コードクローンの類似度を目的変数に設定し，コードクローンの類似度情報を欠落させることなく学習に利用することができる．

一般的な意味的コードクローン検出モデルを 2 値分類モデルから回帰モデルに変更した例を図 6.1 に示す．図 6.1 の提案モデルでは，2 値分類層の代わりに回帰層を用いる．具体的には，学習に使用する損失関数を the Binary Cross Entropy から the Mean Squared Error に変更する．そのため，深層学習モデルの出力は整数ではなく実数となる．そして，入力されたコード片のペアがコードクローンかどうかを，深層学習モデルが出力する類似度スコアに基づいて判定するために，閾値 T を設定する．深層学習モデルが出力した類似度スコアが閾値 T より大きい場合はコードクローン，小さい場合は非コードクローンであると判定する．

このように，提案手法は損失関数の変更を行うという単純な手法であるため，既存の深層学習モデルに対して容易に適用できる手法である．

### 6.2.2 目的変数の変更

回帰モデルを用いて教師あり学習を行う場合，目的変数には実数を設定する必要がある．提案手法では，コードクローンの場合は以下で説明する 2 つのコード片の AST 類似度を目的変数に設定し，非コードクローンの場合は 0.0 を目的変数に設定する．この点が，単に AST 類似度が高いペアをコードクローンだと判定する手法との違いである．提案手法の構成をとることによって，“意味的コードクローンではない” と学習データ内でラベリングされているペアは，たとえ AST 類似度の値が大きくても，0 が出力されるように深層学習モデルの学習が進行する．これにより，意味的コードク

ローンに対するモデルの出力スコアは，非コードクローンに対する出力スコアよりも高くなるため，意味的コードクローンを検出することができる．

提案手法では，AST 類似度を算出するツールを Zhang と Shasha が提案したアルゴリズム [73] を元に実装した．Zhang と Shasha のアルゴリズムは，順不同の木のペア $(A, B)$ が与えられた時，最小編集コストで木 $A$ を木 $B$ に変換する編集シーケンスを作成するためのものである．編集シーケンスは，ノードのラベル変更，木へのノード挿入，ノード削除の 3 つのノード編集操作の列である．また，編集コストは，各ノード編集操作のコストの合計で定義され，通常，各操作のコストは 1 である．Zhang と Shasha のアルゴリズムは最小編集コストでの編集シーケンスの検出は保証されているが，時間計算量はノード数の 2 乗に比例するため，大きな木に対しては時間効率が悪くなる．

そこで提案手法では，Hashimoto と Mori によって提案された編集コスト近似手法 [16] を用いて効率的に木の編集シーケンスを計算できるようにした．また，ノードの移動を新たに編集操作として認めることで，ソースコードの AST に特化した編集シーケンスを作成するツールを実装した．そして，このツールで得られた編集シーケンスを用いて AST 類似度を定義した．具体的には，ある AST のペア $(A, B)$ の間の編集シーケンスを $s$，$A, B$ のノード数を $x, y$，編集シーケンス $s$ によって新たに挿入および削除がされておらず，何も編集操作が適用されていないか，ラベル変更やノード移動の操作のみが適用されているノードの数を $z$ とした時，ペア $(A, B)$ の類似度 $S$ を次のように定義した．

$$S(A, B) = \frac{2z}{x + y}$$

また，この AST 類似度 $S$ を使用して，各コード片のペアに対する目的変数の値 $V$ を次のように定義した．ここで，コード片 $a, b$ の AST は $A, B$ である．

$$V(a, b) = \begin{cases} S(A, B) & (a, b \text{ がコードクローン}) \\ 0 & (a, b \text{ が非コードクローン}) \end{cases}$$

そして，コード片の組 $(a, b)$ を説明変数，$V(a, b)$ を目的変数とした教師あり学習を行うことで，意味的コードクローン検出モデルを作成する．

## 6.3 評価実験

深層学習を用いた意味的コードクローン検出において，2 値分類モデルから回帰モデルに変更することがコードクローン検出精度や深層学習モデルの汎化性能にどのような影響を与えるか確認するために，本節では評価実験を行う．ここで，実験対象モデルの 1 つである ASTNN [72] のコードクローン検出粒度がメソッド単位であるため，本評価実験のコードクローンの粒度はメソッド単位とする．本評価実験では，深

表 6.1: 実験環境

| | |
|---|---|
| OS | macOS Catalina 10.15.7 |
| CPU | 8-Core Intel Xeon W 3.2GHz |
| メモリ | 128GB DDR4-2666 |
| 深層学習ライブラリ | PyTorch[*1] |

層学習モデルの汎化性能を正確に評価するため，6.1 節で説明した依存関係を持たないように学習用データセットと評価用データセットを構築した．また，本評価実験を行った環境を表 6.1 節に示す．

### 6.3.1 評価尺度

#### 適合率・再現率・F 値

まず適合率とは，コードクローン検出ツールがコードクローンであると判定したコード片ペアのうち，評価用データセット中に存在するコードクローンの割合である．次に再現率とは，評価用データセット中に存在するコードクローンのうち，コードクローン検出ツールがコードクローンだと判定することができた割合である．最後に F 値とは，適合率と再現率の調和平均である．これら 3 つの評価尺度は一般的にコードクローン検出ツールの評価尺度として用いられているため，本評価実験でもこれら 3 つの評価尺度を用いる．

#### AUC

AUC（Area Under the Curve） [5] とは，受信者操作特性（Receiver Operating Characteristic, 以降，ROC）曲線の下の面積であり，分類モデルの評価尺度として用いられる．ここで ROC 曲線とは，“1-再現率” を横軸，“適合率” を縦軸に設定し，閾値を連続的に変化させて “1-再現率” と “適合率” の交点をプロットしていき，プロットされた点を結んだ曲線である．ROC 曲線を引く際に閾値を変化させる必要があるので，2 値分類モデルでは，コードクローンクラスに対する予測確率を，回帰モデルにおける類似度スコアのように使用して AUC を算出する．AUC の値域は 0 から 1 の実数であり，1 に近いほど，適切な閾値を設定した場合の F 値が高く，分類性能が高いことを表す．コードクローン分野において広く用いられる評価尺度である適合率・再現率・F 値に加えて，分類モデルの評価尺度として一般的な AUC を用いることによって，より多角的にコードクローン検出モデルを評価することができる．

[*1] https://www.pytorch.org/

表 6.2: データセットの内訳

| | コードクローン | 非コードクローン |
|---|---|---|
| BCB | 77508 | 24633 |
| GCJ | 274048 | 274048 |
| SCB | 861 | 861 |
| SeSaMe | 114 | 114 |
| CSN | 300 | 300 |

### 6.3.2 データセット

既存の意味的コードクローン検出に関する研究では，1 つのデータセットを分割して学習用データセットと評価用データセットを作成し，深層学習モデルの学習と評価を行っている．しかし，この方法はデータセット間に 6.1 節で説明した依存関係があるため，深層学習モデルの汎化性能を正しく評価できていない．そのため，本評価実験では，学習用データセットと評価用データセットをそれぞれ別のデータセットから作成し学習と評価を行うことで，既存研究よりも正確に深層学習モデルの汎化性能を評価する．具体的には，2.1.2 節で説明した 5 つのデータセットの内，目視確認が行われ，コードクローンに対するラベリングの正確さが保証されており，学習に十分な数のコードクローンを含むデータセットである BCB を用いて学習用データセットを作成し，BCB 以外の 4 つのデータセットを用いて評価用データセットを作成することで依存関係を排除した．以下に各データセットの詳細を述べる．

#### 学習用データセット

本評価実験では，手作業で正確にラベリングされているコードクローンが最も多いデータセットが BCB であることから，BCB を学習用データセットとして利用する．最初に ASTNN に対して，ASTNN の開発者らが公開している BCB のデータセットを学習させた．しかし，コードクローンの数に比べて非コードクローンの数が大幅に少なかったため，ASTNN は非コードクローンの特徴を学習できず，ほぼすべてのペアをコードクローンと判定するようになった．そのため，次に 5000 個の非コードクローンを学習用データセットに追加して学習を行った．この非コードクローンは BCB に存在する 2 つのメソッドを組み合わせて作成した．その結果，ASTNN は非コードクローンの特徴を学習することができ，大部分のペアをコードクローンと判定する現象は発生しなかった．また，非コードクローンをさらに追加し，コードクローンと非コードクローンを同じ数にして学習を行った．その結果，ASTNN はコードクローン特徴をうまく学習することができず，大部分のペアを非コードクローンと判定した．

そのため，本評価実験では，ASTNN の開発者らが公開している BCB のデータセットに含まれるコードクローン及び非コードクローンと，新たに作成した 5000 個の非コードクローンを学習用データセットとして利用する．学習用データセットの内訳を表 6.2 に示す．

評価用データセット

本評価実験では，GoogleCodeJam データセット，SemanticCloneBench，SeSaMe，CodeSearchNet データセットの計 4 つの意味的コードクローンベンチマークを評価用データセットとして利用した．各評価用データセットはユニークな視点から意味的コードクローンを定義している．また，構文的な類似度が高いことは，これらの評価用データセットにおいてはコードクローンであることの必要条件ではないことから，BCB のコードクローンとは性質が異なる．そのため，本評価実験の学習用データセットと評価用データセットの間に依存関係は存在しないと考えられる．各評価用データセットの内訳を表 6.2 に示す．

また，短いコード片は一般的にコードクローン検出ツールの評価に利用しない．短さの基準は様々であり，コードの行数の観点からは 5 行 [35,74] や 10 行 [14]，トークン数の観点からは 50 トークン [35,53] などがよく用いられる．コード片の行数は書き方に依存するため，本評価実験では 50 トークンを短さの閾値として採用し，50 トークン以下のメソッドは評価用データセットから取り除いた．

**GoogleCodeJam データセット**（以降，GCJ）[74] は，Zhao と Huang によって，オープンジャッジシステムに提出された回答ソースコードを用いて構築されたデータセットである．このデータセットは “ある同じ設問に対して提出されたメソッドはその設問を解決するための機能を持つため，互いに意味的コードクローンである” という仮定の下構築されている．また，非コードクローンは “互いに異なる設問に対して提出されたメソッド” という仮定の下で収集されている．このとき，コードクローンに比べて非コードクローンの数が非常に多くなる．そのため，本評価実験では非コードクローンの数をコードクローンと同数になるように調整した．また，オープンジャッジの特性上，全てのメソッドに入力を受け付けるためのコード片や設問に対する解答を出力するためのコード片が含まれる．これらのコード片は本評価実験では意図していないコードクローンと見なし，データセットに利用する回答ソースコードからこれらのコード片を削除した．

**SemanticCloneBench**（以降，SCB）[2] は，Al-omari らによって提案された意味的コードクローンのベンチマークであり，Stack Overflow[*2]のコード片を用いて構築されている．このベンチマークは，“同じ質問に対する回答コード片は互いに意味的コードクローンである” という仮定の下構築されている．このベンチマークには 1000

[*2] https://stackoverflow.com/

個の意味的コードクローンが含まれていたが，そのうち 139 個に Javalang[*3]で構文解析できないメソッドが含まれていたため，残りの 861 個を利用した．さらに本評価実験では，861 個の非コードクローンを作成した．具体的には，SCB に含まれるメソッドからランダムに 2 つ選択し，それらがコードクローンとして SCB に登録されていなければ，それらの 2 つのメソッドを非コードクローンとしてデータセットに追加した．

**SeSaMe** [30] は Kamp らによって提案された意味的コードクローンのデータセットで，オープンソースソフトウェアのメソッドが用いられている．このデータセットは，“JavaDoc の文章が類似しているメソッドは意味的コードクローンである” という仮定に基づいて構築されており，900 個の意味的コードクローンが含まれている．しかし，このデータセットには，API を呼び出すだけのメソッド等，短いメソッドが多く含まれていた．また，JavaDoc の文章全体の意味は異なっていても同じ単語が文章に使われているために，実際にはコードクローンではないがコードクローンとしてデータセットに含まれているペアが存在していた．そのようなペアを取り除くため，本評価実験では SeSaMe データセットを目視確認した．その結果，114 個のメソッドペアが実際のコードクローンであることを確認した．そして，SCB と同様の方法で 114 個の非コードクローンを作成した．

**CodeSearchNet** （以降，CSN） [22] は，コード片に特化した検索エンジンである．本評価実験では，“CSN でコード検索を行うとき，同じ検索クエリで検索可能なコード片は意味的コードクローンである” という仮定の下，485100 個のコードクローン候補ペアを収集した．その中から短いメソッドを含むペアを削除したあと，コードクローン候補ペアを手動で確認し，コードクローンのラベリングを行なった．ただし，485100 個全てを目視確認することは困難であるため，コードクローンであると判断したペアが 300 個現れるまでランダムな順で目視確認を行い，見つけた 300 個のペアをコードクローンとした．そして，SCB・SeSaMe と同様に 300 個の非コードクローンを作成した．このとき，300 個の非コードクローンペアの中に 485100 個のコードクローン候補ペアは含まれないようにした．

### 6.3.3 評価対象モデル

本節では評価対象として選択した 4 つの深層学習モデルについて説明する．本評価実験で用いるデータセットにはコンパイルが難しいソースコードが含まれるため，本評価実験ではソースコードを入力する際にコンパイルを必要としない深層学習モデルを選択した．

[*3] https://github.com/c2nes/javalang

### ASTNN

ASTNN [72] 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）[61] にステートメントベクトルの深さ優先探索順列を入力することでコード片をベクトル化し，Siamese モデルを用いて意味的コードクローン検出を行う深層学習モデルである．著者らによって深層学習モデルのソースコードやデータセットが公開されており，コードクローン検出対象のコード片のコンパイルが不要なため，ASTNN を本評価実験に用いる．ASTNN のハイパーパラメータは，全てデフォルト値である．

### LSTM・Bi-LSTM

Long Short-Term Memory（以降，LSTM）[19] は時系列データに対応可能な深層学習モデルの 1 つである．LSTM は前から順に系列を読み込むのに対し，Bi-directional Long Short-Term Memory（以降，Bi-LSTM）は前からの読み込みと同時に後ろからも系列を読み込む．ソースコードはトークン列などの系列データとして表現可能なため，LSTM と Bi-LSTM はコードクローン検出手法に利用されている [63, 72]．よって，これらの深層学習モデルを，本評価実験における Siamese モデルの埋め込み層に用いる．また本評価実験では，目的変数に使用する AST 類似度との親和性を考え，トークン列ではなく，AST ノードの深さ優先探索順列（先行順）を説明変数に使用する．本手法では最初に，ASTNN の構文解析器を利用してソースコードの構文解析を行い，AST ノードの深さ優先探索順列（先行順）を作成する．2 番目に，作成した順列の順番に従って AST ノードを 1 つずつ LSTM または Bi-LSTM に入力する．その結果，ソースコードの埋め込みベクトルが出力される．その埋め込みベクトルを用いて，Siamese モデルによる意味的コードクローン検出を行う．LSTM・Bi-LSTM のハイパーパラメータは 5.2.2 節の LSTM+AST と同様の値（埋め込み層の次元数は 300，LSTM の隠れ層のノード数は 128）に設定し，2 値分類層および回帰層のハイパーパラメータは ASTNN と同じ値に設定した．

### FNN

Feedforward Neural Networks（以降，FNN）[57] は，ネットワークにループ構造が含まれない標準的なニューラルネットワークである．ソースコードメトリクスを並べるなどの方法でソースコードのベクトル化を行うことで，FNN をコードクローン検出に利用することができる [38, 46, 53, 74]．本評価実験では，ソースコードのコンパイルが不要な既存手法の 1 つである CLCDSA [46] を選択し，CLCDSA で利用されているソースコードメトリクスと FNN モデルを再現して実験を行う．FNN のハイパーパラメータは，全て CLCDSA のデフォルト値である．

### 6.3.4　回帰モデルの閾値

6.2.1 節で説明した通り，回帰モデルの出力は実数になるため，閾値を設定する必要がある．そのため事前実験として，BCB のソースコードを学習させた ASTNN の回帰モデルを BCB のソースコードで評価した．その結果，閾値が 0.1 の時に最も F 値が高くなった．そのため，本評価実験における回帰モデルでは閾値を 0.1 に設定した．

### 6.3.5　実験結果

実験結果を表 6.3 に示す．この表から分かるように，2 値分類モデルから回帰モデルに変更すると，ASTNN，LSTM，Bi-LSTM の場合は全てのデータセットに対して F 値と AUC が共に高くなった．ASTNN の場合は平均で F 値が 0.341，AUC が 0.253 だけ上昇し，LSTM の場合は平均で F 値が 0.120，AUC が 0.157 だけ上昇し，Bi-LSTM の場合は平均で F 値が 0.169，AUC が 0.116 だけ上昇した．FNN の場合は GCJ に対しては F 値と AUC が高くなったが，GCJ 以外のデータセットに対しては F 値と AUC が低くなった．平均すると F 値が 0.033，AUC が 0.050 だけ低下した．以上のように，2 値分類モデルから回帰モデルに変更することによって，4 つ中 3 つの深層学習モデルのコードクローン検出精度が向上することが分かった．

### 6.3.6　考察

評価実験では 2 値分類モデルと回帰モデルのコードクローン検出精度比較を行った．その結果，FNN 以外の深層学習モデルにおいてコードクローン検出精度が向上することが確認できた．FNN で精度が向上しなかったのは，入力としてモデルに与えたソースコード構造に関する情報が，他のモデルと比べて少ないためだと考えられる．FNN 以外のモデルは AST ノードの深さ優先探索順列を入力として与えられ，AST ノードの順序関係を学習しているのに対し，FNN にはソースコードメトリクスのベクトルを入力として与えており，ソースコードの構造を学習していない．そのため，回帰モデルで類似度スコアを予測することが難しく，精度がほとんど変わらなかったと考えられる．

次に，回帰モデルを用いることによって学習に利用できるようになった AST 類似度の影響を調べるために，評価データにおけるコードクローンと非コードクローンの AST 類似度と，回帰モデルに変更した ASTNN が出力した類似度スコアの分布を図 6.2 に示す．各図の横軸はコード片ペアの AST 類似度（AST Similarity）であり，縦軸は ASTNN が出力した類似度スコア（score）である．そして，コード片ペア毎の AST 類似度と ASTNN による類似度スコアを黒点でプロットし，回帰直線を青線で引いている．ただし GCJ のコード片ペアは非常に数が多いため，コードクローンと非

表 6.3: 実験結果

| 深層学習モデル | データセット | 分類層 | 適合率 | 再現率 | F 値 | AUC |
|---|---|---|---|---|---|---|
| ASTNN | GCJ | 2 値分類 | 0.521 | 0.314 | 0.392 | 0.513 |
| | | 回帰 | 0.575 | 0.877 | 0.694 | 0.724 |
| | SCB | 2 値分類 | 0.669 | 0.285 | 0.399 | 0.667 |
| | | 回帰 | 0.706 | 0.820 | 0.759 | 0.832 |
| | SeSaMe | 2 値分類 | 0.974 | 0.325 | 0.487 | 0.746 |
| | | 回帰 | 0.791 | 0.763 | 0.777 | 0.844 |
| | CSN | 2 値分類 | 0.843 | 0.197 | 0.319 | 0.711 |
| | | 回帰 | 0.745 | 0.643 | 0.691 | 0.785 |
| LSTM | GCJ | 2 値分類 | 0.506 | 0.936 | 0.657 | 0.511 |
| | | 回帰 | 0.517 | 0.979 | 0.676 | 0.743 |
| | SCB | 2 値分類 | 0.659 | 0.659 | 0.659 | 0.718 |
| | | 回帰 | 0.597 | 0.914 | 0.722 | 0.842 |
| | SeSaMe | 2 値分類 | 0.682 | 0.509 | 0.583 | 0.689 |
| | | 回帰 | 0.662 | 0.807 | 0.727 | 0.804 |
| | CSN | 2 値分類 | 0.661 | 0.383 | 0.485 | 0.631 |
| | | 回帰 | 0.671 | 0.823 | 0.740 | 0.788 |
| Bi-LSTM | GCJ | 2 値分類 | 0.521 | 0.687 | 0.593 | 0.537 |
| | | 回帰 | 0.508 | 0.986 | 0.671 | 0.671 |
| | SCB | 2 値分類 | 0.644 | 0.582 | 0.611 | 0.672 |
| | | 回帰 | 0.606 | 0.904 | 0.726 | 0.810 |
| | SeSaMe | 2 値分類 | 0.741 | 0.351 | 0.476 | 0.637 |
| | | 回帰 | 0.622 | 0.807 | 0.702 | 0.753 |
| | CSN | 2 値分類 | 0.689 | 0.280 | 0.398 | 0.628 |
| | | 回帰 | 0.629 | 0.683 | 0.655 | 0.703 |
| FNN | GCJ | 2 値分類 | 0.562 | 0.922 | 0.698 | 0.717 |
| | | 回帰 | 0.690 | 0.606 | 0.645 | 0.725 |
| | SCB | 2 値分類 | 0.545 | 0.931 | 0.685 | 0.757 |
| | | 回帰 | 0.575 | 0.804 | 0.670 | 0.687 |
| | SeSaMe | 2 値分類 | 0.528 | 0.895 | 0.664 | 0.715 |
| | | 回帰 | 0.556 | 0.789 | 0.652 | 0.634 |
| | CSN | 2 値分類 | 0.519 | 0.837 | 0.640 | 0.587 |
| | | 回帰 | 0.498 | 0.723 | 0.590 | 0.529 |

コードクローンをそれぞれ 1000 ペアだけランダムに抽出しプロットしている．この図を見ると，AST 類似度と類似度スコアに相関があり，ASTNN は入力されたコード片ペアの AST 類似度に近い値を予測していることが分かる．また，図 6.2 から分かるように，多くのペアの AST 類似度は 0.0 から 0.4 の範囲に含まれている．そして，こ

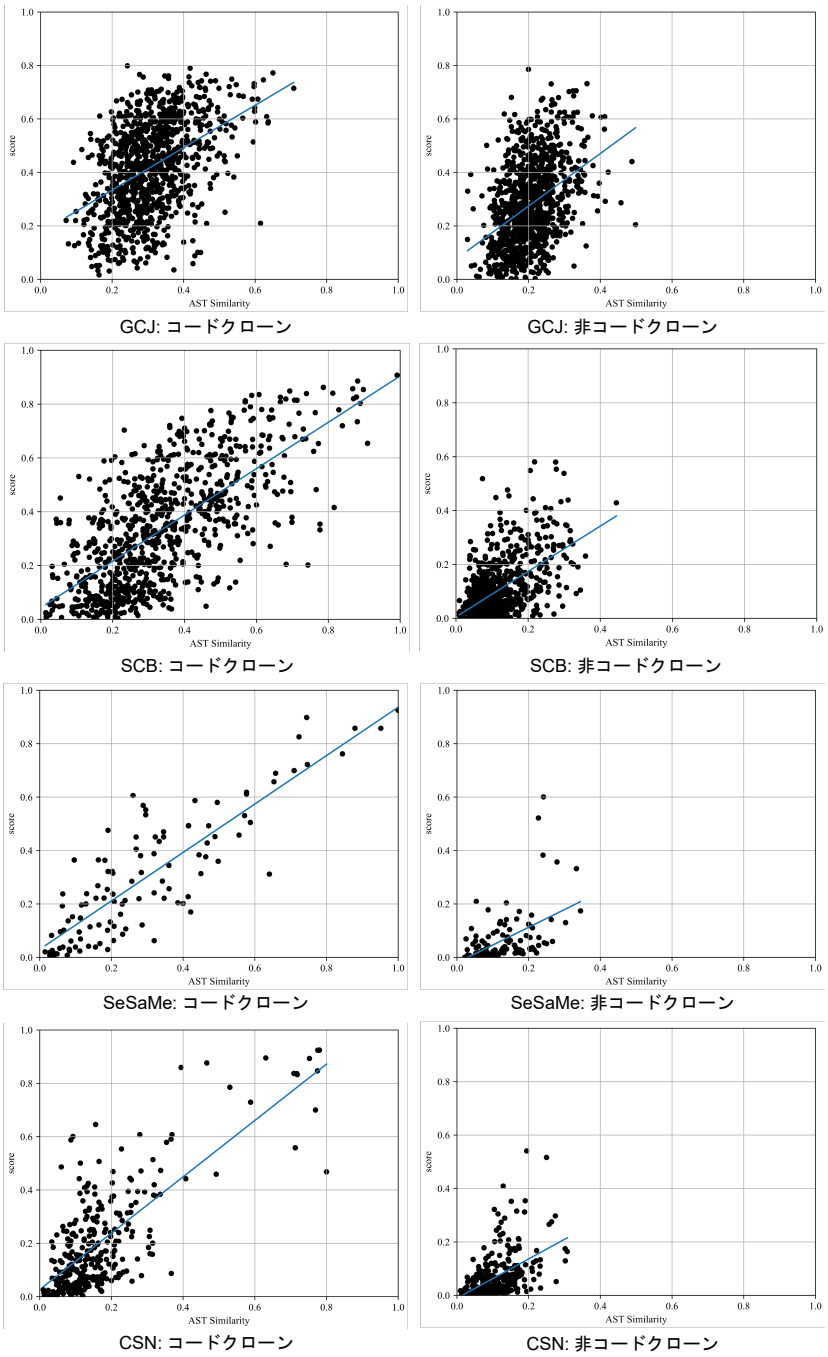


図 6.2: AST 類似度・予測スコアの散布図と回帰直線

の範囲では全てのデータセットにおいてコードクローンの回帰直線が非コードクローンの回帰直線の上部に描かれている．これは，同じ AST 類似度のコードクローンと非コードクローンをそれぞれ入力した場合に，回帰モデルは非コードクローンよりもコードクローンに対して高い類似度スコアを出力する傾向があることを示している．従ってこの結果から，本評価実験における回帰モデルは AST の類似度だけでなく意

表 6.4: AST 類似度が 0.5 以上のコードクローンを取り除いた場合の実験結果

| 深層学習モデル | データセット | 分類層 | 適合率 | 再現率 | F 値 | AUC |
|---|---|---|---|---|---|---|
| ASTNN | SCB | 2 値分類 | 0.582 | 0.231 | 0.331 | 0.634 |
| | | 回帰 | 0.649 | 0.782 | 0.709 | 0.802 |
| | SeSaMe | 2 値分類 | 0.957 | 0.234 | 0.376 | 0.700 |
| | | 回帰 | 0.744 | 0.713 | 0.728 | 0.811 |
| LSTM | SCB | 2 値分類 | 0.598 | 0.606 | 0.602 | 0.685 |
| | | 回帰 | 0.547 | 0.896 | 0.679 | 0.814 |
| | SeSaMe | 2 値分類 | 0.585 | 0.404 | 0.478 | 0.626 |
| | | 回帰 | 0.605 | 0.766 | 0.676 | 0.762 |
| Bi-LSTM | SCB | 2 値分類 | 0.575 | 0.521 | 0.547 | 0.634 |
| | | 回帰 | 0.556 | 0.883 | 0.683 | 0.780 |
| | SeSaMe | 2 値分類 | 0.622 | 0.245 | 0.351 | 0.572 |
| | | 回帰 | 0.562 | 0.766 | 0.649 | 0.702 |
| FNN | SCB | 2 値分類 | 0.492 | 0.918 | 0.640 | 0.732 |
| | | 回帰 | 0.519 | 0.769 | 0.619 | 0.656 |
| | SeSaMe | 2 値分類 | 0.474 | 0.872 | 0.614 | 0.661 |
| | | 回帰 | 0.493 | 0.745 | 0.593 | 0.570 |

味的コードクローンの特徴を類似度スコアに反映する能力があると考えられる．

また，図 6.2 から，SCB と SeSaMe データセットには，AST 類似度が高いコードクローンが比較的多く含まれていることが分かった．具体的には，AST 類似度が 0.5 以上のコードクローンが，SCB に 146 個，SeSaMe データセットに 20 個含まれていた．本章では，意味的コードクローン検出手法，すなわち構文的な類似度が低いコードクローンを検出可能な手法に注目しているため，AST 類似度が低いコードクローンに対する検出精度を調べることが重要である．そのため，SCB と SeSaMe データセットから，AST 類似度が 0.5 以上のコードクローンを取り除き，AST 類似度が低いコードクローン検出精度の評価実験を行った．その実験結果を表 6.4 に示す．表 6.4 から分かる通り，FNN 以外の場合で，提案手法である回帰モデルの方が 2 値分類モデルよりも F 値と AUC は高かった．この結果から，検出対象を AST 類似度の低いコードクローンに限定した場合でも，提案手法を適用することで FNN 以外の場合に深層学習モデルのコードクローン検出精度が向上することが確認できた．

また本評価実験では，多くの既存研究での評価実験と同様に，評価用データセットのコードクローン数と非コードクローン数を揃えた実験を行った．しかし，実際のソースコードに対してコードクローン検出を行う場合，メソッドのペアがコードクローンかどうかを総当たりで調査する必要がある．そのため，評価用データセットのコードクローンを構成するメソッドを利用し，メソッドのペアを総当たりで調査するための現実に即した評価用データセットを作成した．その内訳を表 6.5 に示す．表 6.5 の “

表 6.5: 現実に即した評価用データセットの内訳

| | メソッド数 | コードクローン | 非コードクローン |
|---|---|---|---|
| GCJ | 1663 | 274048 | 1107905 |
| SCB | 1722 | 861 | 1480920 |
| SeSaMe | 194 | 114 | 18607 |
| CSN | 542 | 697 | 145914 |

メソッド数” は，評価用データセットのコードクローンを構成するメソッドの種類数に等しい．SCB では，コードクローンを構成するメソッドに重複がないため，メソッド数はコードクローン数の 2 倍である．その他のデータセットでは，1 つのメソッドが複数のコードクローンを構成するため，コードクローン数の 2 倍よりもメソッド数が少ない．また，CSN は，542 個のメソッドの全てのペアを作成した時，評価用データセットでラベリングされていた 300 個のコードクローンの他に 6.3.2 節で説明したコードクローン候補が約 2000 個存在した．そのため，コードクローン候補の目視確認を行った．その結果，全体のコードクローン数は 697 個になった．

現実に即したデータセットでの実験結果を表 6.6 に示す．適合率が 0.001 を下回る場合があったため，表 6.6 の数値は小数第 4 位まで表記している．6.3.2 のデータセットと比べて非コードクローン数が大幅に増えたため，全ての場合で適合率と F 値が低下した．CSN 以外のデータセットは含まれているコードクローンに変化がないため，再現率は変わらなかった．一方，CSN データセットに新たに追加されたコードクローンを高精度で検出することができたため，CSN データセットに対する再現率は上昇した．また，SeSaMe に対する ASTNN の実験結果など，表 6.3 において 2 値分類モデルの適合率が高いデータセットと分類層の組み合わせの一部で，2 値分類モデルと回帰モデルの F 値の優劣が逆転した．しかし，AUC は表 6.3 と同様に FNN 以外の場合で回帰モデルの方が高かった．AUC が高いという結果から，閾値（6.3.4 節参照）を評価用データセット毎に適切に設定すれば，FNN 以外の場合では回帰モデルの方が高精度なコードクローン検出を行うことができる．実際に，SeSaMe に対する ASTNN の実験結果に関して，F 値が最も高くなる閾値について調査した．その結果，2 値分類モデルは閾値が 0.8 の時に F 値が 0.0952 で最も高くなったのに対し，回帰モデルは閾値が 0.45 の時に F 値が 0.2625 で最も高くなり，回帰モデルの F 値が 2 値分類モデルの F 値を上回ったことを確認した．ただし，本評価実験は意味的コードクローン検出モデルの汎化性能の評価を目的としているため，6.3.4 節のように学習用データセットの検証結果に基づいて閾値を決定する必要があり，評価用データセット毎に閾値を変えることはできない．よって，評価用データセット毎に適切な閾値が変わることなく一定になるように学習を行うことが，提案手法の課題の 1 つであることが分かった．

表 6.6: 現実に即したデータセットでの実験結果

| 深層学習モデル | データセット | 分類層 | 適合率 | 再現率 | F 値 | AUC |
|---|---|---|---|---|---|---|
| ASTNN | GCJ | 2 値分類 | 0.1370 | 0.3141 | 0.1908 | 0.3494 |
| | | 回帰 | 0.2498 | 0.8771 | 0.3889 | 0.7228 |
| | SCB | 2 値分類 | 0.0013 | 0.2846 | 0.0026 | 0.6787 |
| | | 回帰 | 0.0014 | 0.8200 | 0.0028 | 0.8285 |
| | SeSaMe | 2 値分類 | 0.0247 | 0.3246 | 0.0458 | 0.6834 |
| | | 回帰 | 0.0148 | 0.7632 | 0.0290 | 0.7988 |
| | CSN | 2 値分類 | 0.0272 | 0.3257 | 0.0503 | 0.7843 |
| | | 回帰 | 0.0133 | 0.7604 | 0.0262 | 0.8263 |
| LSTM | GCJ | 2 値分類 | 0.2019 | 0.9356 | 0.3321 | 0.5067 |
| | | 回帰 | 0.2092 | 0.9785 | 0.3448 | 0.7412 |
| | SCB | 2 値分類 | 0.0012 | 0.6585 | 0.0024 | 0.7255 |
| | | 回帰 | 0.0009 | 0.9141 | 0.0017 | 0.8436 |
| | SeSaMe | 2 値分類 | 0.0119 | 0.5088 | 0.0233 | 0.6683 |
| | | 回帰 | 0.0097 | 0.8070 | 0.0192 | 0.7758 |
| | CSN | 2 値分類 | 0.0154 | 0.5538 | 0.0299 | 0.7638 |
| | | 回帰 | 0.0087 | 0.9082 | 0.0173 | 0.8602 |
| Bi-LSTM | GCJ | 2 値分類 | 0.2145 | 0.6874 | 0.3270 | 0.5459 |
| | | 回帰 | 0.2037 | 0.9856 | 0.3377 | 0.6943 |
| | SCB | 2 値分類 | 0.0011 | 0.5819 | 0.0022 | 0.6789 |
| | | 回帰 | 0.0010 | 0.9036 | 0.0019 | 0.8220 |
| | SeSaMe | 2 値分類 | 0.0117 | 0.3509 | 0.0226 | 0.6254 |
| | | 回帰 | 0.0095 | 0.8070 | 0.0187 | 0.7640 |
| | CSN | 2 値分類 | 0.0175 | 0.4534 | 0.0336 | 0.7387 |
| | | 回帰 | 0.0105 | 0.8077 | 0.0208 | 0.8197 |
| FNN | GCJ | 2 値分類 | 0.2407 | 0.9221 | 0.3817 | 0.7172 |
| | | 回帰 | 0.3547 | 0.6056 | 0.4474 | 0.7256 |
| | SCB | 2 値分類 | 0.0007 | 0.9314 | 0.0014 | 0.7457 |
| | | 回帰 | 0.0008 | 0.8037 | 0.0015 | 0.6825 |
| | SeSaMe | 2 値分類 | 0.0066 | 0.8947 | 0.0132 | 0.7101 |
| | | 回帰 | 0.0073 | 0.7894 | 0.0145 | 0.6536 |
| | CSN | 2 値分類 | 0.0053 | 0.8451 | 0.0106 | 0.6404 |
| | | 回帰 | 0.0056 | 0.7475 | 0.0110 | 0.6353 |

## 6.4 妥当性の脅威

妥当性の脅威として，本章を通して学習に用いたデータセットが BCB のみである点が挙げられる．6.3.2 節で説明した通り，BCB に含まれているコードクローンは手

作業で正確にラベリングされているため，本章では BCB を学習用データセットに選択している．しかし，汎化性能が低い原因として，BCB のコードクローンが学習に適していない可能性が考えられるため，BCB とは異なるデータセットを学習に用いるなどの方法で検証する必要がある．

## 6.5 関連研究

現在まで，深層学習を用いた意味的コードクローン検出に関する研究が多く存在する．DeepSim [74] は，制御/データフローグラフの情報を用いてコード片をベクトル化し，それを順伝播型ニューラルネットワークを用いて解析し，意味的コードクローン検出を行う．CDLH [63] は，AST に対して tree-LSTM を適用することで抽象構文木を用いたコードクローン検出を行う．Oreo [53] や CLCDSA [46] は，Java のメソッドをソースコードメトリクスのベクトルで表現し，それを Siamese モデルを用いて解析することで，意味的コードクローン検出を行う．TBCCD [71] は，トークン情報をノードに付与した AST をベクトル化し，Siamese モデルを用いて解析を行うことで意味的コードクローン検出を行う．FCCA [21] は，1 つのソースコードから 3 つのソースコード表現（トークン列，AST，制御フローグラフ）を生成し，3 つの深層学習モデル（LSTM，Tree-LSTM，グラフ畳み込みネットワーク）を使ってそれぞれのソースコード表現を学習することで，意味的コードクローン検出を行う．この手法は，一般的な既存手法で用いられている 2 値分類モデルではなく，非コードクローン・タイプ 1 クローン・タイプ 2 クローン・タイプ 3 クローン・タイプ 4 クローンの 5 通りで分類を行う 5 値分類モデルであることが特徴の 1 つである．以上の研究では，同じデータセットから学習データと評価データを作成しているため，学習データと評価データの間に依存関係が生じており，過学習が発生している可能性がある．一方，本評価実験では BCB から学習データを作成し，BCB 以外のデータセットから評価データを作成しているため，学習データと評価データの間に依存関係は生じておらず，深層学習モデルの汎化性能をより正確に評価することができる．

## 6.6 まとめ

本章では，回帰モデルを用いたコードクローン検出手法を提案し，その汎化性能について評価した．提案手法では，深層学習によるコードクローン検出の既存手法で一般的に用いられている 2 値分類モデルの代わりに回帰モデルを用いることで，コードクローンの AST 類似度を学習に利用した．評価実験では，学習と評価に異なるデータセットを用いて依存関係を取り除き，適合率・再現率・F 値・AUC を用いて深層学習モデルのコードクローン検出精度を評価した．その結果，回帰モデルを用いると 4 つ中 3 つの深層学習モデルでコードクローン検出精度が向上することを確認した．

# 第 7 章

# むすび

## 7.1　まとめ

本論文では，深層学習を用いたソースコード分類の既存手法が抱える問題点に着目し，その解決を試みた．

3 章では，深層学習を用いたソースコード分類手法の導入として，順伝播型ニューラルネットワークとコードミューテーションを用いたコードクローン検索を行う手法について説明した．この手法は，学習に用いるソースコードに対してミューテーションを適用し，コードクローンを作成することで，元の学習データ数が少ない場合でも，深層学習を行うにあたって十分なデータ数を確保することができる．評価実験により，この手法は構文的に類似したソースコードを高い精度で検索できることが分かった．さらに，ミューテーションの影響に関する評価実験を行った結果，学習に用いるソースコードの数がコードクローンの検索精度に大きく影響することが分かった．

4 章では，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ことで，深層学習を用いたソースコード分類の既存研究で特に説明なくランダムサンプリングが用いられているという問題点に取り組んだ．この章では，深層学習モデルの学習を行った後，学習済み深層学習モデルの評価用データセットに対するソースコード分類精度の検証を行い，正答率が相対的に低いクラスに対し，そのクラスのソースコードに対してミューテーションを適用して作成したコードクローンを追加する手法を提案した．また，評価実験を行い，提案手法によってコードクローンを追加したクラスの正答率が上昇し，低下していた深層学習モデルのソースコード分類精度が向上することを確認した．

5 章では，深層学習を用いたソースコード分類手法の分類精度に関する比較調査を行うことで，ニューラルネットワークやソースコード表現の比較検討が不十分であるという問題点に取り組んだ．この章では，3 種類のニューラルネットワークと 2 種類のソースコード表現を組み合わせた計 6 種類のソースコード分類手法の精度の比較調査を行った．その結果，再帰型ニューラルネットワークにソースコードのトークン

列を学習させる手法の分類精度が最も高いことが分かった．また，深層学習を用いたソースコード分類手法と深層学習を用いないソースコード分類手法のソースコード分類精度を比較した．その結果，深層学習を用いた手法のほうが，深層学習を用いないソースコード分類手法よりも分類精度が高いことが分かり，深層学習はソースコード分類に有効であることが分かった．

6 章では，回帰モデルを用いたコードクローン検出手法の提案と複数のプロジェクトを用いた評価実験を通して，深層学習を用いたコードクローン検出手法の汎化性能が評価されていないことが多いという問題点に取り組んだ．この章では，コードクローンの学習方法を改善するために，2 値分類モデルではなく回帰モデルを利用し，教師あり学習に使用する目的変数をコード片の類似度に変更する手法を提案した．また，評価実験では，学習データに対する深層学習モデルの挙動を確認し，深層学習モデルの汎化性能を評価するために，学習用データセットと評価用データセットを同一プロジェクトから作成するのではなく，依存関係のない 5 種類のプロジェクトを用いて作成した．そして，4 種類の深層学習モデルについて，2 値分類モデルから回帰モデルに変更し，変更前モデルと変更後モデルの汎化性能を比較した．その結果，4 つの内 3 つの深層学習モデルの汎化性能が向上した．

## 7.2　今後の研究方針

今後，深層学習分野での研究が進むと，新たなネットワーク構造や学習アルゴリズムを持つニューラルネットワークが開発されると考えられる．そのため，今後の研究方針として，本論文で扱ったニューラルネットワークの代わりに新たなニューラルネットワークを用いてソースコード分類精度の調査を行い，新たなニューラルネットワークのソースコード分類に対する有効性を調べることが挙げられる．本調査の発展として，深層学習を用いたソースコード分類のガイドラインを作りたいと考えている．ガイドラインを作り上げていくことで，深層学習を用いたソースコード分類についての知識や，実際のソフトウェア開発に深層学習を用いたソースコード分類技術をどう適用するかにあたっての知見を共有すると，深層学習を用いたソースコード分類技術の発展速度向上が期待でき，ソースコード再利用のさらなる効率化，ひいてはソフトウェア開発の生産性向上に繋がると考えられる．

また，6 章の評価実験で，2 値分類モデルから回帰モデルに変更することで深層学習モデルの汎化性能が向上する場合があることを示した．しかし，回帰モデルのコードクローン検出精度は，データセットの依存関係を取り除く前のコードクローン検出精度には及んでおらず，汎化性能にはまだ向上の余地があると考えられる．そのため，今後の研究方針として，上記で述べた新たなニューラルネットワークを利用するほか，学習させるソースコード表現を変更したり，様々なニューラルネットワークを組み合わせるなどの方法で，さらに汎化性能の高い意味的コードクローン検出モデルを開発

したいと考えている．例えば 6.1 節で述べたように，Zhang ら [72] の提案した深層学習モデルである ASTNN を用いた意味的コードクローン検出手法は，依存関係のあるデータセットに対しては 0.938 という非常に高い F 値を記録している．残念ながら同節で述べたように ASTNN の汎化性能は低いという結果になっているが，汎化性能を得ることができれば意味的コードクローンを非常に高い精度で検出することができるようになるポテンシャルを秘めている．高精度な意味的コードクローン検出を実現できると，さらにコードクローンに対する保守作業を効率的に行うことができるようになるほか，ソフトウェア開発者が必要としている機能を持ったソースコードをより効率的に検索できるようになると考えられる．

# 参考文献


[1] IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[2] F. Al-Omari, C. K. Roy, and T. Chen. SemanticCloneBench: A semantic code clone benchmark using crowd-source knowledge. In *Proc. of IWSC 2020*, pp. 57–63, 2020.

[3] Mohammad Samy Baladram, Atsushi Koike, and Kazunori D Yamada. Introduction to supervised machine learning for data science. *Interdisciplinary Information Sciences*, Vol. 26, No. 1, pp. 87–121, 2020.

[4] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant Anna, and Lorraine Bier. Clone detection using abstract syntax trees. In *Proc. of ICSM 1998*, pp. 368–377, 1998.

[5] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern Recognit.*, Vol. 30, No. 7, pp. 1145–1159, 1997.

[6] C Braun. Nato standard for the development of reusable software components. *NATO Communications and Information Systems Agency*, 1992.

[7] Jane Bromley, James W. Bentz, Leon Bottou, Isabelle Guyon, Yann Lecun, Eduard Sackinger, and Roopak Shah. Signature verification using a 'siamese' time delay neural network. *Int. J. Pattern Recognit. Artif. Intell.*, Vol. 7, No. 4, pp. 669–688, 1993.

[8] Chao Chen and Mei-Ling Shyu. Clustering-based binary-class classification for imbalanced data sets. In *Proc. of IRI 2011*, pp. 384–389, 2011.

[9] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems*, Vol. 2, pp. 303–314, 1989.

[10] Li Deng and Dong Yu. Deep learning: Methods and applications. *Found. Trends Signal Process.*, Vol. 7, No. 3–4, pp. 197–387, 2014.

[11] Vahid Farrahi, Maisa Niemelä, Petra Tjurin, Maarit Kangas, Raija Korpelainen, and Timo Jämsä. Evaluating and enhancing the generalization performance of machine learning models for physical activity intensity predic-

tion from raw acceleration data. *IEEE J. Biomedical and Health Informatics*, Vol. 24, No. 1, pp. 27–38, 2020.

[12] 藤原裕士, 崔恩瀞, 吉田則裕, 井上克郎. 順伝播型ニューラルネットワークを用いた類似コードブロック検索の試み. ソフトウェアエンジニアリングシンポジウム 2018 論文集, pp. 24–33, 2018.

[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, 2020.

[14] Y. Gao, Z. Wang, S. Liu, L. Yang, W. Sang, and Y. Cai. TECCD: A tree embedding approach for code clone detection. In *Proc. of ICSME 2019*, pp. 145–156, 2019.

[15] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. Deep api learning. In *Proc. of FSE 2016*, pp. 631–642, 2016.

[16] Masatomo Hashimoto and Akira Mori. Diff/TS: A tool for fine-grained structural change analysis. In *Proc. of WCRE 2008*, pp. 279–288, 2008.

[17] 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, Vol. J91-D, No. 6, pp. 1465–1481, 2008.

[18] Yoshiki Higo, Yasushi Ueda, Shinji Kusumoto, and Katsuro Inoue. Simultaneous modification support based on code clone analysis. In *Proc. of APSEC 2007*, pp. 262–269, 2007.

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

[20] Raphael Hoffmann, James Fogarty, and Daniel S. Weld. Assieme: Finding and leveraging implicit references in a web search interface for programmers. In *Proc. of UIST 2007*, pp. 13–22, 2007.

[21] W. Hua, Y. Sui, Y. Wan, G. Liu, and G. Xu. FCCA: Hybrid code representation for functional clone detection using attention networks. *IEEE Trans. Reliability*, Vol. 70, No. 1, pp. 304–318, 2021.

[22] Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. CodeSearchNet Challenge: Evaluating the state of semantic code search, 2019.

[23] Katsuro Inoue and Chanchal K. Roy. *Code Clone Analysis: Research, Tools, and Practices*. Springer, 2021.

[24] Katsuro Inoue, Yusuke Sasaki, Pei Xia, and Yuki Manabe. Where does this code come from and where does it go?-integrated code history tracker for open source systems. In *Proc. of ICSE 2012*, pp. 331–341, 2012.

[25] Takashi Ishio, Raula Gaikovina Kula, Tetsuya Kanda, Daniel M. Germán, and Katsuro Inoue. Software ingredients: detection of third-party component

reuse in java software release. In *Proc. of MSR 2016*, pp. 339–350, 2016.

[26] Takashi Ishio, Yusuke Sakaguchi, Kaoru Ito, and Katsuro Inoue. Source file set search for clone-and-own reuse analysis. In *Proc. of MSR 2017*, pp. 257–268, 2017.

[27] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE Trans. Software Eng.*, Vol. 37, No. 5, pp. 649–678, 2010.

[28] Lingxiao Jiang, Ghassan Misherghi, Zhendong Su, and Stephane Glondu. DECKARD: Scalable and accurate tree-based detection of code clones. In *Proc. of ICSE 2007*, pp. 96–105, 2007.

[29] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, Vol. 28, No. 7, pp. 654–670, 2002.

[30] M. Kamp, P. Kreutzer, and M. Philippsen. SeSaMe: A data set of semantically similar java methods. In *Proc. of MSR 2019*, pp. 529–533, 2019.

[31] 片山卓也, 土居範久, 鳥居宏次. ソフトウェア工学大事典. 朝倉書店, 1998.

[32] Ganesan Kavita and Foti Romano. C# or java? typescript or javascript? machine learning based classification of programming languages. `https://github.co/2Jif7Sg`, 2019.

[33] Shinji Kawaguchi, Pankaj K Garg, Makoto Matsushita, and Katsuro Inoue. Mudablue: An automatic categorization system for open source repositories. *J. Syst. Softw.*, Vol. 79, No. 7, pp. 939–953, 2006.

[34] Naohiro Kawamitsu, Takashi Ishio, Tetsuya Kanda, Raula Gaikovina Kula, Coen De Roover, and Katsuro Inoue. Identifying source code reuse across repositories using lcs-based source code similarity. In *Proc. of SCAM 2014*, pp. 305–314, 2014.

[35] Kisub Kim, Dongsun Kim, Tegawendé F. Bissyandé, Eunjong Choi, Li Li, Jacques Klein, and Yves Le Traon. FaCoY: A code-to-code search engine. In *Proc. of ICSE 2018*, pp. 946–957, 2018.

[36] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR 2017*, 2017.

[37] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proc. of ICML 2014*, pp. 1188–1196, 2014.

[38] L. Li, H. Feng, W. Zhuang, N. Meng, and B. Ryder. CCLearner: A deep learning-based clone detection approach. In *Proc. of ICSME 2017*, pp. 249–260, 2017.

[39] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard S. Zemel. Gated

graph sequence neural networks. Proc. of ICLR 2016 Poster Presentations, 2016.

[40] Bennet P Lientz. Issues in software maintenance. *ACM Computing Surveys (CSUR)*, Vol. 15, No. 3, pp. 271–278, 1983.

[41] Mario Linares-Vásquez, Collin Mcmillan, Denys Poshyvanyk, and Mark Grechanik. On using machine learning to automatically classify software applications into domain categories. *Empir. Softw. Eng.*, Vol. 19, No. 3, p. 582–618, 2014.

[42] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Proc. of ECCV 2016*, pp. 21–37. Springer International Publishing, 2016.

[43] Oren Melamud, David McClosky, Siddharth Patwardhan, and Mohit Bansal. The role of context types and dimensionality in learning word embeddings. In *Proc. of NAACL 2016*, pp. 1030–1040. Association for Computational Linguistics, 2016.

[44] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Proc. of NIPS 2013*, pp. 3111–3119, 2013.

[45] Lili Mou, Ge Li, Lu Zhang, Tao Wang, and Zhi Jin. Convolutional neural networks over tree structures for programming language processing. In *Proc. of AAAI 2016*, pp. 1287–1293, 2016.

[46] K. W. Nafi, T. S. Kar, B. Roy, C. K. Roy, and K. A. Schneider. CLCDSA: Cross language code clone detection using syntactical features and api documentation. In *Proc. of ASE 2019*, pp. 1026–1037, 2019.

[47] Federal Information on Processing Standards. Guideline on software maintenance. 1984.

[48] Chaiyong Ragkhitwetsagul and Jens Krinke. Siamese: scalable and incremental code clone search via multiple code representations. *Empir. Softw. Eng.*, pp. 2236–2284, 2019.

[49] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proc. of CVPR*, pp. 779–788, 2016.

[50] Chanchal K. Roy and James R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proc. of ICPC 2008*, pp. 172–181, 2008.

[51] Chanchal K Roy and James R Cordy. A mutation/injection-based automatic

framework for evaluating code clone detection tools. In *Proc. of ICSTW 2009*, pp. 157–166, 2009.

[52] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, Vol. 74, No. 7, pp. 470–495, 2009.

[53] Vaibhav Saini, Farima Farmahinifarahani, Yadong Lu, Pierre Baldi, and Cristina V Lopes. Oreo: Detection of clones in the twilight zone. In *Proc. of ESEC/FSE 2018*, pp. 354–365, 2018.

[54] Hitesh Sajnani, Vaibhav Saini, Jeffrey Svajlenko, Chanchal K. Roy, and Cristina V. Lopes. SourcererCC: Scaling code clone detection to big-code. In *Proc. of ICSE 2016*, pp. 1157–1168, 2016.

[55] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., 1986.

[56] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne Van Den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In *Proc. of ESWC 2018*, pp. 593–607, 2018.

[57] Jurgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, Vol. 61, pp. 85 – 117, 2015.

[58] Yogesh Singh and Bindu Goel. A step towards software preventive maintenance. *SIGSOFT Softw. Eng. Notes*, Vol. 32, No. 4, p. 10–es, 2007.

[59] Kathryn T. Stolee, Sebastian Elbaum, and Daniel Dobos. Solving the search for source code. *ACM Trans. Softw. Eng. Methodol*, Vol. 23, No. 3, pp. 26:1–26:45, 2014.

[60] Jeffrey Svajlenko, Judith F Islam, Iman Keivanloo, Chanchal K Roy, and Mohammad Mamun Mia. Towards a big data curated benchmark of inter-project code clones. In *Proc. of ICSME 2014*, pp. 476–480, 2014.

[61] Duyu Tang, Bing Qin, and Ting Liu. Document modeling with gated recurrent neural network for sentiment classification. In *Proc. of EMNLP 2015*, pp. 1422–1432, 2015.

[62] Secil Ugurel, Robert Krovetz, and C. Lee Giles. What's the code? automatic classification of source code archives. In *Proc. of KDD 2002*, KDD '02, p. 632–638. Association for Computing Machinery, 2002.

[63] Huihui Wei and Ming Li. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code. In *Proc. of IJCAI 2017*, pp. 3034–3040, 2017.

[64] Cathrin Weiss, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *Proc. of MSR 2007*, pp. 1–1, 2007.

[65] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk. Deep learning code fragments for code clone detection. In *Proc. of ASE 2016*, pp. 87–98, 2016.

[66] Yilin Yan, Min Chen, Mei-Ling Shyu, and Shu-Ching Chen. Deep learning for imbalanced multimedia data classification. In *Proc. of ISM 2015*, pp. 483–488, 2015.

[67] Yilin Yan, Yang Liu, Mei-Ling Shyu, and Min Chen. Utilizing concept correlations for effective imbalanced data classification. In *Proc. of IRI 2014*, pp. 561–568, 2014.

[68] 横井一輝, 崔恩瀞, 吉田則裕, 井上克郎. 情報検索技術に基づく細粒度ブロッククローン検出. コンピュータ ソフトウェア, Vol. 35, No. 4, pp. 16–36, 2018.

[69] Reishi Yokomori, Norihiro Yoshida, Masami Noro, and Katsuro Inoue. Use-relationship based classification for software components. In *Proc. of QuASoQ 2018*, pp. 59–66, 2018.

[70] Norihiro Yoshida, Takeshi Hattori, and Katsuro Inoue. Finding similar defects using synonymous identifier retrieval. In *Proc. of IWSC 2010*, pp. 49–56, 2010.

[71] Hao Yu, Wing Lam, Long Chen, Ge Li, Tao Xie, and Qianxiang Wang. Neural detection of semantic code clones via tree-based convolution. In *Proc. of ICPC 2019*, pp. 70–80, 2019.

[72] Jian Zhang, Xu Wang, Hongyu Zhang, Hailong Sun, Kaixuan Wang, and Xudong Liu. A novel neural source code representation based on abstrace syntax tree. In *Proc. of ICSE 2019*, pp. 783–794, 2019.

[73] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, Vol. 18, No. 6, pp. 1245–1262, 1989.

[74] Gang Zhao and Jeff Huang. DeepSim: deep learning code functional similarity. In *Proc. of FSE 2018*, pp. 141–151, 2018.

[75] S. Zhou, H. Zhong, and B. Shen. SLAMPA: Recommending code snippets with statistical language model. In *Proc. of APSEC 2018*, pp. 79–88, 2018.

[76] Thomas Zimmermann, Nachiappan Nagappan, Harald C. Gall, Emanuel Giger, and Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. In *Proc. of FSE 2009*, pp. 91–100, 2009.