Vol.2022-SE-210 No.12
2022/3/11

ソースコードのグラフ表現を利用した深層学習による コーディングの専門性の判定手法

松井 智寛[1,a)]　松下 誠[1,b)]　井上 克郎[1,c)]

概要：ソフトウェア開発者のコーディングの専門性を客観的に判定するため，ソースコードを利用した機械学習・深層学習による判定が行われている．しかし，この種の手法は予約語の利用頻度とメトリクス，もしくは出現する単語とその順序関係のみを学習モデルへ入力しており，ソースコードの特徴的な構造に関する情報や変数などの意味的な情報をほとんど利用していない．そこで本研究では，ソースコードを構造に関する情報や意味的な情報を表現できるグラフへ変換し，グラフを学習することができる深層学習を利用して能力の判定を行う手法を提案する．提案手法を用いて実験を行った結果，既存手法に比べて高い精度で判定を行えることがわかった．

キーワード：コーディング，専門性，深層学習，プログラミングコンテスト

1. まえがき

ソフトウェア開発において，企業は必要なコーディングの専門性を持つ開発者を採用し，開発プロジェクトに配置することが必要になっている．たとえば Facebook や Google，Amazon といった技術系の企業では，採用の場面でコーディングの課題を実施することで応募者の専門性を測っている [11]．しかし，このような場面で人によるソースコードの評価を行うと非常に大きなコストがかかってしまう．そのため，コストがかからない方法で開発者の専門性を判定する必要がある．

開発経験が長いほど専門性が高い，といった評価が行われることがある [21]．しかし，開発者の経験と専門知識に関する調査 [2] では一貫した結果が得られておらず，開発経験の長さを利用して専門性を予測すると採用の決定を間違えてしまう可能性があり，仮に採用決定を間違えてしまった場合のコストはその従業員の年俸の 10 倍になるとも言われている [3]．

開発者の専門性はソースコードの質に現れるとされている [2]．そのため，ソースコードから開発者の専門性を予測できると考えられる．これまでにソースコード自体から開発者の専門性を予測する方法として，機械学習や深層学習を利用したモデル [9], [13], [14], [15], [16] や，Zipf の法則のパラメータ推定を利用したもの [19] がある．しかし，機械学習・深層学習を利用するものは，ソースコードの特徴的な構造に関する情報や変数などの意味的な情報を利用していない．またパラメータ推定を利用するものは，推定に必要なソースコードの量が多いといった欠点がある．

そこで本研究ではまず，ソースコードの構造に関する情報や変数の意味的な情報をグラフとして表現する．またグラフを学習することができる RGCN[20] を利用してソースコードから開発者の専門性を予測する手法を提案する．

手法の評価には，既存研究 [13], [14], [15], [16] で用いられたプログラミングコンテストのデータセットと判定手法を利用する．コンテストの参加者は成績によって順位がつけられ，レーティングが変動する．問題を解いて実装する能力，すなわちコーディングの専門性が高い開発者が良い順位を取り，レーティングが高くなると考えられるため，レーティングが高い開発者から上級者・中級者・初級者と分類を行う．この 3 値分類による学習・評価を既存手法と提案手法で行い，結果を比較したところ，提案手法は既存手法に比べて精度が高い予測を行えることが確認できた．

以下，2 節では，本研究の背景として，プログラミングコンテストと機械学習・深層学習を用いた専門性の判定手法，ソースコードのグラフ表現について説明する．3 節では提案手法について説明する．4 節では評価実験の内容と結果について説明し，考察を行う．最後に 5 節ではまとめと今後の課題について述べる．

[1] 大阪大学大学院情報科学研究科
[a)] t-matui@ist.osaka-u.ac.jp
[b)] matusita@ist.osaka-u.ac.jp
[c)] inoue@ist.osaka-u.ac.jp

2. 背景

評価対象のデータセットとして利用するプログラムコンテストとその採点に利用されるオンラインジャッジシステム，既存の機械学習・深層学習を用いた判定手法，ソースコードのグラフ表現について説明する．プログラミングコンテストについてはさまざまなものが存在するが，本研究では問題の要求を満たすプログラムを時間内に作成することを競うコンテストを対象とする．

2.1 プログラミングコンテスト

プログラミングコンテストはプログラミングの能力や技術を競い合うコンテストのことである．オンラインジャッジシステムが採用されており，同じ問題に対して，同じ時間内に複数の参加者がソースコードを記述し，提出する．コンテストが終了すると，正解問題数や回答時間に応じて参加者の順位が決定し，レーティング [6], [18] が変動する．プログラミングコンテストには Codeforces[*1]のように個人でプログラムを作成して回答するもの以外にも，ACM ICPC[*2]のように複数の参加者がチームを組んで回答するものも存在する．本研究では，個人で参加するプログラミングコンテストである Codeforces を対象とする．

2.1.1 ルールと流れ

プログラミングコンテストの Codeforces の大まかな流れについて説明する．Codeforces における一般的なコンテストでは，コンテストの開始時間が指定されており，開始時間になるとコンテストの問題が公開され，参加者は問題を解き始める．問題を解く順序や，用いるプログラミング言語は自由であり，得点は解いた問題の難易度や回答にかかった時間，これまでの提出回数によって変わる．また，参加者は制限時間内であれば，同じ問題に対して何度も回答を提出することが可能であり，回答ソースコードの正誤やコンパイル可能性については提出の可否に影響しない．回答ソースコードは，提出された時点でオンラインジャッジシステムによって事前テストの実行が行われ，通過したか否かが参加者に通知される．事前テストは全テストケースを網羅しておらず，事前テストを通過しても回答が正しくない場合もある．コンテスト時間終了後に提出ソースコードに対して最終テストが実施されることによって，参加者の最終的な得点が決定する．また，点数の高い順に順位を決定し，その結果レーティングが変動する．

2.1.2 開催規模や参加者

本研究で利用する Codeforces においては，開催規模や参加者は以下の通りになっている．

- 開催頻度：偏りはあるが，大体週に 1 回以上
- 参加人数：偏りが大きいが，毎コンテスト 10000〜30000 人程度
- 国籍：全世界から参加（使用言語はロシア語・英語）

Codeforces では，過去 6 か月以内に一度でもコンテストに参加したことのあるユーザーをアクティブユーザーと定義しているが，そのアクティブユーザー数は 2022 年 1 月 6 日現在，108310 人となっている．

2.1.3 レーティングシステム

Codeforces は参加者の熟練度をレーティング [18] を用いて表している．コンテストが行われるたびに参加者のレーティングが順位によって変動する．レーティングの計算方式はチェスなどの対戦型の競技で用いられるイロレーティング [6] と呼ばれる方式に似たものとなっている．本研究では，既存の研究 [13], [14], [15], [16] と同様に，このレーティングが高い参加者を専門性が高い参加者として扱う．

2.2 オンラインジャッジシステム

プログラミングコンテストにおいて，その採点に利用されるオンラインジャッジシステムについて説明する．オンラインジャッジシステムにはさまざまな問題が収録されており，その利用者は問題を選択し，回答を提出する．その後，システムは利用者に提出された回答の採点結果を通知する．採点の基準の一例を以下に示す．

- コンパイルできるかどうか
- 不正なメモリアクセスがないかどうか
- 制限時間内にプログラムが終了するか

このような基準の内，満たさない基準があればそのことを通知し，すべての基準を満たせば正解であることを通知する．

こういったオンラインジャッジシステムの一例として，国内では AIZU ONLINE JUDGE[*3]，国外では Topcoder といったものが存在する．

2.3 機械学習・深層学習を用いた専門性の判定手法

ソースコードそのものを用いて機械学習や深層学習によって専門性を判定する手法には，ソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法 [13], [14], [15], [16] やソースコードを単語毎に分割して自然言語のように扱った深層学習による手法 [9] が存在する．

槇原 [13], [14] はプログラミングコンテストに提出されたソースコードを対象とした学習を行った．プログラミングコンテストの上級者と初級者の特定の予約語の利用頻度やメトリクスの値の違いは堤 [22] によって確認されている．そこで槇原はプログラミングコンテストのレーティングをもとに参加者を上級者と初級者に分類し，ソースコードを

[*1] http://codeforces.com/
[*2] https://icpc.baylor.edu/
[*3] http://judge.u-aizu.ac.jp/onlinejudge/

堤により確認された上級者または初級者で利用頻度が高い if や for 等の予約語の利用頻度とソースコードの行数やクラス数等のメトリクスの値から成るベクトルに変換して決定木や SVM で学習を行うことで，開発者の専門性を予測することができるモデルを作成した．

この槇原の判定手法に対し，改良を加えた研究をこれまで行ってきた [15], [16]．ここでは槇原の研究では利用されていなかった，上級者・初級者以外の参加者が中級者として判定の対象に加えられ，さらにメトリクスとしてネストの深さごとの論理行数が追加された．また，学習アルゴリズムもランダムフォレスト（RF）に変更することでモデルの改良が行われた．

Javeed ら [9] は，GitHub に公開されている Java プロジェクトを対象とした学習を行った．ここではまず 14,785 のプロジェクトが収集され，コンパイル可能であった 6,244 のプロジェクトが機密性，信頼性，複雑性，行数，保守性，重複といった観点から expert と novice に分類される．次にソースコードを単語毎に分割し，単語の埋め込みを深層学習のライブラリである Keras[*4]に実装されている Embedding 層を利用して行い，LSTM[7] や CNN[8]，またはその両方を利用した学習を行うことで，開発者の専門性を予測することができるモデルを作成した．

しかし，これらのモデルによる予測は，その入力としてソースコードを用いているにも関わらず，ソースコードにおいて特徴的である構造に関する情報や変数の意味的な情報を全く利用しておらず，情報が大きく損なわれていると考える．このような情報を利用することができれば，より精度の高い予測を行うことができると考えた．

2.4 ソースコードのグラフ表現

Allamanis ら [1] はソースコードをグラフとして表現する手法を提案した．ノードにはソースコードを構文木に変換した際の文法とトークンを利用する．エッジとして，構文木から得られた親子関係を表す Child エッジとトークンの順序関係を表す NextToken エッジを用いる．また，変数が最後に利用されうる場所へ LastUse エッジ，最後に書き込まれうる場所へ LastWrite エッジ，最後に出現した場所へ LastLexicalUse エッジを用いる．その他，メソッド中の return トークンに関する ReturnsTo エッジや仮引数に関する FormalArgName エッジ，if 文に関する GuardedBy，GuardedByNegation エッジなどが定義されている．最後に，それらのエッジの両端を逆にしたものを別のラベルとして加える．

このグラフ表現により，従来使われることが多かった自然言語の手法では扱うことができなかったソースコードの特徴的な構文的構造や意味的構造を扱うことができるようになった．このグラフ表現とグラフを対象とした深層学習のグラフニューラルネットワークの 1 つである GGNN[12] を利用して，変数の利用箇所の中から 1 つを空けたソースコードを提示した際にどの変数が使われるべきかを予測する VARMISUSE タスクにおいて，従来の手法よりも優れた結果が得られることがわかった．このタスクの精度を高めるにはソースコードの構文に関する情報だけでなく，変数などの意味的な情報も必要である．このタスクで優れた結果が得られたことから，このグラフ表現と深層学習を用いるとそれらの情報を上手く学習できることがわかる．

また，このソースコードのグラフ表現は変数の予測以外にも，ソースコード中の脆弱性の特定 [24] やバグの検知・修正 [4] といった分野でも優れた結果が得られている．

3. 提案手法

3.1 目的

2.3 節で示したように，既存の予約語の利用頻度とメトリクスを利用した機械学習による手法 [13], [14], [15], [16] やソースコードを単語ごとに分割して自然言語のように扱った深層学習による手法 [9] は，ソースコードの構造に関する情報や変数の意味的な情報がほとんど利用されていない．そのため，本研究では 2.4 節で示したようなソースコードのグラフ表現とグラフを対象とした深層学習を利用することで，開発者の専門性を判定する手法を提案する．以降の節では，この手法におけるグラフ表現について説明した後，コーディングの専門性の判定の手順について説明する．

3.2 ソースコードのグラフ表現

2.4 節で説明した，Allamanis らが [1] 定義したグラフを利用する深層学習は，ソースコードの特徴的な構造に関する情報や変数の意味的な情報を上手く学習できることが分かっている．また，本研究でのコーディングの専門性の判定対象は，プログラミングコンテストの問題に対する回答のような比較的単純なものを想定している．しかし Allamanis らが定義したグラフはオープンソースのプロジェクトのような複雑なものを対象としており，エッジの種類が多く複雑なものとなっている．そこで，本研究では Allamanis らが定義したグラフの内，コーディングの専門性の判定に十分だと考えられたものを利用する．まず，以下のものをノードとして利用する．

- ソースコード中のトークン
- 構文木から得られる文法を表す節

ノード間を接続するエッジは以下のものを利用する．

- 構文木の親子関係を表す Child エッジ
- トークンの順序関係を表す NextToken エッジ
- 同一変数の利用情報を表す LastLexicalUse エッジ

こうして得られたノードの中には，”assignExpression”（代

[*4] https://keras.io/ja/

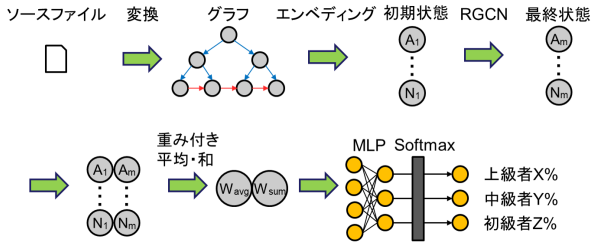


図 1: コーディングの専門性の判定の手順

入文を表すノード）のように複数の単語の組合せとなっているものがある．このような単語の特徴をグラフニューラルネットワークで効果的に学習するため，”assignExpression”に対して”assign”と”Expression”のようにノード名を分割したノードを追加する．分割前のノードから分割後の各ノードには，UsesSubtoken エッジを接続する．このノードの分割および UsesSubtoken エッジの作成には，Microsoft が公開している VARMISUSE タスクの実装 [17] を利用した．

3.3 開発者のコーディングの専門性の判定の手順

開発者のコーディングの専門性の判定の手順を**図 1** に示す．この手順は，ソースコードからグラフへ変換する部分と深層学習を利用してグラフから専門性を判定する部分に分けられる．以降の節では，それぞれの部分について説明する．

3.3.1 ソースコードのグラフへの変換

ANTLR[*5]と Understand[*6]を利用して，以下の手順でソースコードからグラフへの変換を行った．

1. ANTLR でソースコードを構文木に変換する
2. 構文木中の各ノードをグラフのノードとする
3. 親ノードから子ノードに接続されるエッジを Child エッジとする
4. 前のトークンから後ろのトークン NextToken エッジを接続する
5. 冗長なノードを削除する
6. Understand を利用して LastLexicalUse エッジを追加する
7. ノード名を分割して新しいノードと UsesSubtoken エッジを追加する

5. で示した冗長なノードの削除について説明する．**図 2** に，ソースコード中の代入文 n=1 を構文木に変換したものの一部を示している．代入文を表すノード”assignment-Expression”から左辺，=，右辺を表すノードに分かれている．この例では代入文の右辺に 1 という定数が入っているが，代入文の右辺には定数以外にも変数や関数など様々なものが入る可能性がある．そういったものの中から右辺が

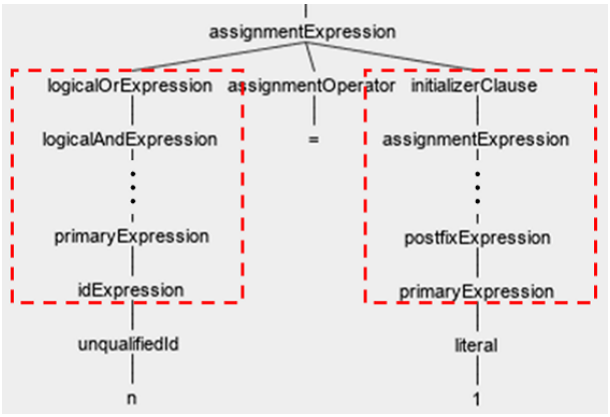


図 2: 代入文 n=1 を ANTLR で構文木に変換したもの

図 3: 代入文 n=1 の構文木から冗長なノードを削除したもの

定数であることを確定するまでの処理が，赤い点線で囲まれた部分となっている．左辺も同様に，変数であることを確定するまでの処理が赤い点線で囲まれた部分で行われている．この部分はノード数が非常に多くなっているため図 2 では省略しているが，本来赤い点線内には左辺で 16 個，右辺で 18 個のノードが入っており，構文木の深さが深くなっている．グラフニューラルネットワークでは，グラフの各ノードはエッジが隣接するノードから情報を集めて状態を更新していくため，この構文木のまま学習してしまうと，例えば右辺の 1 というトークンが代入文の中にあるという情報を得るまでに非常に長い時間がかかってしまう．そのため，赤い点線で囲っている部分を冗長なノードとして削除することで効率よく学習したい．

そこで，以下の 2 つの条件を同時に満たすノードを冗長なノードと定義し，削除する．この際，対象の親ノードから接続される Child エッジはその子ノードへ接続する．

- 子ノードが文法を表すノードであるもの
- 子ノードが 1 つしかないもの

この条件に従い図 2 の冗長なノードを削除したものが**図 3** である．構文木の深さは改善されており，学習の際にノードは効率よく情報を集めることができるようになる．また，4.1 節で説明するデータセットに対して冗長なノードの削除を行った結果，分割したノードの追加を行う前に存在するノードの約 73%を削減することができた．

次に，新しいノードの追加と UsesSubtoken エッジについて説明する．キャメルケースまたはスネークケースに従ってノード名を分割して得られた単語のノードを新たに作成する．また分割する前のノードから分割後の各ノードへ UsesSubtoken エッジを追加する．ただし，分割前の

*5 https://www.antlr.org/
*6 https://www.scitools.com/

ノードが予約語である場合，これらの処理は行わない．

こうしてソースコードをグラフに変換する様子を図 **4** に示す．図 **4a** はノードの分割と UsesSubtoken エッジ以外の例を，図 **4b** はノードの分割と UsesSubtoken エッジの例を示している．

3.3.2 グラフを対象とした深層学習

この節では，グラフを対象に深層学習を行って上級者・中級者・初級者の判定を行う部分について説明する．この部分は大きく 3 つの段階に分けることができる．まず，入力されたグラフのノードをベクトルに変換し，初期状態を得る．次に，ノードの状態とエッジからグラフニューラルネットワークによりノードの状態を更新していく．最後に，ノードの状態から上級者・中級者・初級者の確率を出すことで専門性の判定を行う．

3 つの段階それぞれについて，以下で詳細を述べる．

エンベディング

グラフニューラルネットワークを利用するためには，まずノードをベクトルへ変換する必要がある．本手法ではソースコード中のトークンと構文木から得られる文法を表す節，それらを分割したものをノードとして扱っている．そこで，ノードを表す文字列に対して Character-level CNN[23] を適用することでノードの初期状態を取得する．

グラフニューラルネットワーク

グラフニューラルネットワークはグラフ構造を学習し各ノードの状態を更新していくことで，ノードの予測やエッジの予測，グラフレベルの予測などに利用することができる．本手法はソースコードをグラフに変換し，そのグラフに対してグラフレベルの予測を行う．予測の際，各ノードはエッジが接続している隣接ノードから情報を集約し，集めた情報をもとにノードの状態を更新する．この工程は繰り返し行われる．この集約・更新の方法の違いにより，様々な種類のグラフニューラルネットワークが存在する．本手法では Microsoft が公開している複数のグラフニューラルネットワーク [17] を検証し，今回の用途では最も精度が高かった RGCN[20] を採用した．RGCN は，画像に対する深層学習に使われる畳み込みと呼ばれる技術をグラフニューラルネットワークに応用した GCN[5], [10] を，さらにラベル付きのエッジを持つグラフに適用できるように拡張したものとなっている．

本手法では RGCN によりノードの情報を集約する際，終端のノードが始端のノードの情報を集めるだけでなく，始端のノードも終端のノードの情報を集めることができるようにする．これは VARMISUSE タスク [1] でも採用されており，情報の伝達をより速くし，モデルの表現力を向上させることがわかっている．

専門性の判定

RGCN によって得られたノードの最終状態から専門性の判定を行う．

まず，各ノードの初期状態と最終状態のベクトルを連結させる．次に全ノードの重み付き和と重み付き平均を算出し，連結させる．この連結したベクトルを出力数 3 の MLP に入力し，得られる出力をされに Softmax 層に入力すると，3 つの値の合計値が 100%となるように上級者・中級者・初級者である確率が出力される．ここで例えば上級者の割合が最も高くなった場合，このソースコードを書いた開発者は上級者であると判定する．

この実装は，Microsoft が公開しているグラフの回帰タスクの実装[*7]を本手法で扱う 3 値分類に拡張したものとなっている．

4. 手法の評価

実験を行って提案手法を評価する．最初に，実験に利用したデータセットと比較対象のベースラインについて説明する．その後提案手法とベースライン手法で開発者のコーディングの専門性の判定を行い，結果の比較と考察を行う．

4.1 データセット

ここでは評価実験に用いたデータセットについて述べる．このデータセットは既存のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法 [13], [14], [15], [16] で用いられているものであり，堤 [22] によって作成された．このデータセットはオンライン上で公開されている[*8]．

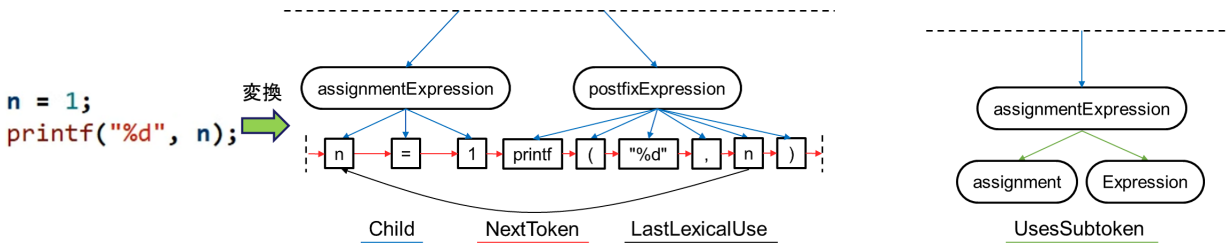
データセットは，参加者が問題への回答として提出したソースコードと，言語やタイムスタンプ等の提出履歴情報データベースの 2 種類からなる．データセットの統計情報は表 1 で示している．また，本データは 2016/5/19〜2016/11/15 の期間に収集されており，ソースコードのファイル数は 1,644,636 である．プログラミングコンテストにおける提出は 1 つのソースファイルにまとめられるため，これは提出数と一致している．提出されたソースコードの言語別提出数の割合を表 2 に示す．ソースコードのうち，90%は C++によって記述されている．表 1 の参加者数は，2016/5/19〜2016/11/15 の期間に 1 度以上 Codeforces のコンテストに参加したユーザーの総数である．また，各コンテストには複数の問題が含まれるため，コンテストの数と比較して問題数が多くなっている．

表 1: データセットの統計情報

収集期間	ファイル数	参加者数
2016/5/19〜2016/11/15	1,644,636	14,520

コンテスト数	問題数	DB サイズ
739	3,218	357MB

[*7] https://github.com/microsoft/tf2-gnn
[*8] https://sites.google.com/site/miningprogcodeforces/



(a) 代入文と printf 文を変換する例．ただしノードの分割は省略している．　**(b)** 代入文を表すノードを分割する例

図 4: グラフ中のエッジの例

表 2: 提出ソースコードの言語分布

言語	C++	Java	C	Python	その他
割合	90.00%	4.15%	2.08%	1.92%	1.86%

4.1.1 ソースコード

ソースコードは，プログラミングコンテストの参加者が問題に対する回答として提出したものであり，コンパイル環境が用意されている任意の言語を提出することができる．また，回答ソースコードが正答であるか，コンパイル可能かどうかは提出の可否に影響しないため，文法的に不完全なソースコードが含まれる場合もある．

4.1.2 提出履歴

提出履歴情報には，参加者のレーティング情報や，どの参加者がどの問題にいつ提出したか，提出が正解であったか等の情報が含まれている．

4.2 ベースライン

実験には，既存のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法と，ソースコードを単語毎に分割して自然言語のように扱った深層学習による手法 [9] を比較対象として利用する．

ランダムフォレスト (RF)

メトリクスを利用した機械学習を用いる手法として，2.3 節で説明したものの中から，ここでは槇原の手法 [13], [14] に改良を加えたもの [15], [16] を利用する．

LSTM

ソースコードを自然言語のように扱った深層学習による手法では，始めにソースコードを単語ごとに分割し，それぞれの単語をベクトルに変換する．その後ソースコードを単語列のベクトルとして深層学習モデルに入力して判定を行う．既存研究では CCN，LSTM，CCN と LSTM を組み合わせたモデルで実験を行っていたが，本研究のベースラインにはその実験で最も精度が高かった LSTM によるモデルを利用する．また，2 値分類による判定が行われていたが，本研究では上級者・中級者・初級者の 3 値分類による判定を行うため，出力層を変更する．そこで GitHub 上で公開されている実装[*9]を確認したところ，以下の 2 つの問題点が存在した．

- 単語の分割をスペースとタブ，改行，コンマの有無で判断している
- コメントアウトされている部分も単語をベクトルに変換して学習する

まず単語の分割が不適切であることについて説明する．単語の分割を上で示したように行うと，例えば printf(”a b”); のようなソースコードの内の 1 行は「printf(”a」という部分と「b”);」という部分の 2 つを単語として分けてしまう．しかし，ソースコードの最小単位であるトークンで分割すると printf，(，”a b”，)，; という 5 つのトークンに分かれるため，この分割方法は適切であるとは言えない．そのため，実験を行う際にはソースコードをトークンで分割して学習を行うように変更する．

次に，コメントアウトされている部分の学習が不適切であることについて説明する．まず，この手法では単語をベクトルへ変換した後，全ての単語を深層学習へ入力するのではなく，ハイパーパラメータで指定された 251 個目以降の単語は入力しない．また，ソースコードの始めにはコピーライト等がコメントとして記述されることもある．実際に本研究で利用するデータセット内に**図 5** に示すようなソースコードも存在した．このようなソースコードをコメントを含めて学習してしまうと，ソースコードの処理を行う部分に焦点を当てた学習ができなくなる可能性があり，コメントアウトの部分を含めた学習は適切であるとは言えない．そのため，実験を行う際にはコメントアウトされている部分を無視するように変更する．

4.3 実験内容

提案手法とベースライン手法を用いて，開発者のコーディングの専門性の判定を行う．

まず，データセットのソースコードを上級者・中級者・初級者のコードに分類する．ここでは既存の手法 [15], [16] と同様に最も提出数の多かった C++のソースコードを提出者のレーティングでソートし，上位 25%を上級者，下位

[*9] https://github.com/Akhtar-Munir/Developer-level-detect

```
// Copyright (C) 2016 Sayutin Dmitry.
//
// This program is free software; you can redistribute it and/or
// modify it under the terms of the GNU General Public License as
// published by the Free Software Foundation; version 3

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program; If not, see <http://www.gnu.org/licenses/>.
```

図 5: コピーライトとライセンスが記述されている例

25%を初級者，残りを中級者のコードとしている．この時，上級者・中級者・初級者のレーティングの分布は表 3 のようになった．

表 3: 上級者・中級者・初級者のレーティングの統計情報

		初級者	中級者	上級者
レーティング	平均	1180.22	1459.66	1944.35
	分散	13170.37	10153.27	46307.75
	最小値	-39	1311	1711
	中央値	1211	1434	1902
	最大値	1310	1710	3367
人数		3899	8409	2212
ファイル数		353,346	701,871	352,557

提案手法とベースライン手法それぞれでこれらのソースコードに対して学習・評価を行う．提案手法及び深層学習を利用した手法 [9] は，学習データを全体の 8 割，評価データを全体の 1 割，テストデータを残りの 1 割して分割したときのテストデータの評価値を求める．機械学習による手法 [15], [16] は，学習データを全体の 9 割，テストデータを残りの 1 割として 10 分割交差検証を行い評価値を求める．評価値として，accuracy と F 値の 2 つを採用した．提案手法については，ソースコードの構造や変数の意味情報を表すエッジの有効性を評価するために，NextToken エッジの制限や LastLexicalUse エッジの制限，NextToken エッジと LastLexicalUse エッジのどちらも制限したグラフを用いて学習を行った評価もあわせて行う．

利用するソースコードに対して，提案手法のグラフ変換を行った際に得られた構文的な情報と，変数の意味的な情報を表す Child エッジ，NextToken エッジ，LastLexicalUse エッジの 1 ファイル当たりの数を表 4 に示す．

表 4: 各エッジの 1 ファイル当たりのエッジの数

エッジの種類	エッジ数
Child エッジ	656
NextToken エッジ	347
LastLexicalUse エッジ	60

表 5: 提案手法とベースライン手法の結果

モデル	上級者	中級者	初級者	accuracy
提案手法	0.891	0.881	0.835	0.871
RF	0.758	0.793	0.699	0.762
LSTM	0.770	0.768	0.697	0.750

表 6: 提案手法のエッジを制限した学習の結果

制限したエッジ	上級者	中級者	初級者	accuracy
NextToken	0.882	0.874	0.827	0.863
LastLexicalUse	0.888	0.876	0.829	0.866
上記 2 種類	0.868	0.860	0.804	0.847

4.4 実験結果

4.3 節で説明した実験の結果を表 5 と表 6 に示す．

4.5 結果の比較・考察

まず表 5 から，提案手法は上級者の F 値が 0.891，中級者の F 値が 0.881，初級者の F 値が 0.835，accuracy が 0.871 となっており，全ての評価値でベースライン手法を上回っている．そのため，ソースコードの構文的情報と変数の意味的な情報によりコーディングの専門性を効果的に学習できたことがわかる．

また，表 6 の提案手法から NextToken エッジと LastLexicalUse エッジを制限した学習は，表 5 のベースライン手法に比べて各評価値が高く，大きく精度が向上している．このことから，構文木から得られる構文の情報は非常に有用だと考えられる．

そしてこの 2 つのエッジの両方を制限した学習は，どちらか片方を制限した学習に比べて全ての評価値が低くなっている．NextToken エッジを制限した学習と LastLexicalUse エッジを制限した学習の評価値は多少の差はあるが，両方を制限した学習に比べ各評価値は同程度向上している．一方で，表 4 に示したように，1 ファイル数当たりのエッジ数は LastLexicalUse エッジは NextToken エッジの 1/5 以下となっている．従って，LastLexicalUse エッジは少ない数で NextToken エッジと同等の精度向上があると考えられ，また変数の意味的情報はコーディングの専門性の判定において非常に有用だと考えられる．

5. まとめ

本研究では，ソースコードの構文的情報や変数の意味的情報をグラフとして表現し，グラフを対象とする深層学習の 1 つである RGCN を用いてコーディングの専門性を判定する手法を提案した．プログラミングコンテストのデータセットを利用して提案手法の評価を行ったところ，提案手法は従来のソースコード中の予約語の利用頻度やソースコードから計測できるメトリクスを利用した機械学習による手法や，ソースコードを単語毎に分割して自然言語のよ

うに扱った深層学習による手法と比べて良い結果が得られた．また，グラフのエッジを制限した学習の評価も行うことで，エッジの有用性も確認した．

今後の課題として，汎化性能を確認することが考えられる．本研究ではデータセットにプログラミングコンテストを利用した．そのため，この手法がプログラミングコンテスト以外のドメインにおいて有用かどうかは判明していない．プログラミングコンテストではラベル付けに客観的な指標であるレーティングを利用することができた．しかし，他のドメインでレーティングのような客観的なコーディングの専門性の指標を得ることは難しいのではないかと考えられる．データセットを作成することができれば，汎用性の確認だけでなく，汎用性の高い手法の作成も行うことができるようになると考えられる．

参考文献

[1] Allamanis, M., Brockschmidt, M. and Khademi, M.: Learning to Represent Programs with Graphs, *International Conference on Learning Representations* (2018).
[2] Baltes, S. and Diehl, S.: Towards a Theory of Software Development Expertise, *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Association for Computing Machinery, pp. 187–200 (2018).
[3] Bressler, M. S.: Building the winning organization through high-impact hiring, *Journal of Management and Marketing Research*, Vol. 15 (2014).
[4] Dinella, E., Dai, H., Li, Z., Naik, M., Song, L. and Wang, K.: HOPPITY: LEARNING GRAPH TRANSFORMATIONS TO DETECT AND FIX BUGS IN PROGRAMS, *International Conference on Learning Representations* (2020).
[5] Duvenaud, D., Maclaurin, D., Aguilera-Iparraguirre, J., Gómez-Bombarelli, R., Hirzel, T., Aspuru-Guzik, A. and Adams, R. P.: Convolutional Networks on Graphs for Learning Molecular Fingerprints, *Advances in Neural Information Processing Systems*, Vol. 2, pp. 2224–2232 (2015).
[6] Elo, A. E.: *The Rating of Chessplayers, Past and Present*, Arco Pub. (1978).
[7] Hochreiter, S. and Schmidhuber, J.: Long Short-Term Memory, *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780 (1997).
[8] Jacovi, A., Sar Shalom, O. and Goldberg, Y.: Understanding Convolutional Neural Networks for Text Classification, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, Association for Computational Linguistics, pp. 56–65 (2018).
[9] Javeed, F., Siddique, A., Munir, A., Shehzad, B. and Lali, M. I.: Discovering software developer's coding expertise through deep learning, *IET Software*, Vol. 14, No. 3, pp. 213–220 (2020).
[10] Kipf, T. N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks, *International Conference on Learning Representations* (2017).
[11] Laakmann McDowell, G.: *Cracking the coding interview: 189 programming questions and solutions*, CareerCup (2015).
[12] Li, Y., Tarlow, D., Brockschmidt, M. and Zemeli, R.: Gated Graph Sequence Neural Networks, *International Conference on Learning Representations* (2016).
[13] 槇原啓介：ソースコード特徴量を用いた機械学習によるソースコードの良否の判定，大阪大学基礎工学部情報科学科卒業論文 (2019).
[14] 槇原啓介， 松下誠，井上克郎：ソースコード特徴量を用いた機械学習によるソースコード品質の評価手法，電子情報通信学会技術研究報告，Vol. 119, No. 113, pp. 105–110 (2019).
[15] 松井智寛：判定対象の拡大を目的とした 3 値分類によるソースコードの良さの判定手法，大阪大学基礎工学部情報科学科卒業論文 (2020).
[16] 松井智寛， 松下誠，井上克郎：判定対象の拡大を目的とした 3 値分類によるソースコード品質の評価手法，情報処理学会研究報告，Vol. 2020-SE-205, No. 7, pp. 1–8 (2020).
[17] Microsoft: tf-gnn-samples, `https://github.com/microsoft/tf-gnn-samples` (2019).
[18] Mirzayanov, M.: Codeforces Rating System, `http://codeforces.com/blog/entry/102` (2010).
[19] Moradi Dakhel, A., C. Desmarais, M. and Khomh, F.: Assessing Developer Expertise from the Statistical Distribution of Programming Syntax Patterns, *Proceedings of the Evaluation and Assessment in Software Engineering*, Association for Computing Machinery, p. 90–99 (2021).
[20] Schlichtkrull, M., Kipf, T. N., Bloem, P., vanÂ den Berg, R., Titov, I. and Welling, M.: Modeling Relational Data with Graph Convolutional Networks, *European Semantic Web Conference*, Springer International Publishing, pp. 593–607 (2018).
[21] Siegmund, J., Kästner, C., Liebig, J., Apel, S. and Hanenberg, S.: Measuring and Modeling Programming Experience, *Empirical Softw. Engg.*, Vol. 19, No. 5, pp. 1299–1334 (2014).
[22] 堤祥吾：プログラミングコンテスト初級者・上級者におけるソースコード特徴量の比較，大阪大学大学院情報科学研究科修士論文 (2018).
[23] Zhang, X., Zhao, J. and LeCun, Y.: Character-Level Convolutional Networks for Text Classification, *Advances in Neural Information Processing Systems*, Vol. 1, pp. 649–657 (2015).
[24] Zhou, Y., Liu, S., Siow, J., Du, X. and Liu, Y.: Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks, *Advances in Neural Information Processing Systems*, pp. 10197–10207 (2019).