修士学位論文

題目

情報検索技術に基づくベクトル表現と
深層学習を用いたコード片の類似性判定法

指導教員
井上 克郎 教授

報告者
横井 一輝

平成 31 年 2 月 6 日

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 ソフトウェア工学講座

平成 30 年度 修士学位論文

情報検索技術に基づくベクトル表現と深層学習を用いたコード片の類似性判定法

横井 一輝

内容梗概

コード片の類似性判定法は，ソフトウェア工学における重要な基礎技術である．コードクローン検出やコード片検索など，類似したコード片を見つける作業はソフトウェアの開発や保守において重要であり，その際にコード片の類似性判定法が使用される．今までにコード片の構文的な類似性を判定する手法が数多く提案されている．しかし，コード片の処理内容が意味的に類似していることを判定する手法は少ない上に，種々の課題が残っている．

例えば，Zhao らは，制御フローとデータフローの情報を基に深層学習モデルを作成し，コード片の類似性を判定する手法として DeepSim を提案している．DeepSim はコード片の意味的な類似性を高い精度で判定できるが，モデルの学習に時間がかかるという課題がある．また筆者の既存研究では，類似したコード片であるコードクローンを検出するために，情報検索技術に基づくベクトル表現とコサイン類似度を用いたコード片の類似性判定法を提案した．この手法は意味的に類似したコードクローンを高速に検出できる一方で，検出漏れが多いことが指摘されている．

これらの課題を解決するため，本研究では情報検索技術に基づくベクトル表現と深層学習モデルを組み合わせた，より高速かつ高精度なコード片の類似性判定法を提案する．提案手法では，軽量的なソースコード解析を用いた前処理を行い，情報検索技術を用いてベクトル表現に変換する．その後, 深層学習モデルを用いてコード片が類似しているか否かを判定する．これにより，構文的に異なるが処理内容の意味が類似しているコード片を，より高速かつ高精度に判定することが可能になる．

評価実験では，大規模コードクローン集合と競技プログラミングにおいて提出されたコード片に対し，最新手法である DeepSim との比較評価を行った．評価実験の結果，情報検索技術の一種であり，主成分分析による潜在的意味解析を行う LSI（Latent Semantic Indexing）と深層学習モデルを用いた提案手法が，再現率，適合率，F 値において最も高い値となった．また，実行時間も DeepSim と比較し，学習時間においては約 350 倍高速であり，推定時間においては同等であることが確認できた．

主な用語

ソフトウェア保守
コードクローン
情報検索技術
深層学習

目 次

1　まえがき

コード片の類似度判定法は，ソフトウェア工学における重要な基礎技術である．コードクローン検出 [28] やコード片検索 [12, 16] など，類似したコード片を見つける作業はソフトウェアの開発や保守において重要であり，その際にコード片の類似性判定法が使用される．コードクローンとはソースコード中に含まれる互いに一致または類似した部分を持つコード片であり，コードクローン検出ではコード片の類似性判定法を用いてソースコード中の互いにコードクローンになっているコード片を識別する [10]．またコード片検索とは，クエリとして与えられたコード片に類似したコード片を検索する手法であり，類似性判定法を用いて検索する．開発者は主に再利用対象を見つけるために，コード片検索を行う．

今までにコード片の構文的な類似性を判定する手法は数多く提案されている．既存手法の多くは，テキストやトークン，抽象構文木などからコード片の特徴を抽出し，ユークリッド距離やコサイン類似度などの距離尺度を用いてコード片の類似性を判定する．しかし構文的ではなく，コード片の処理内容が意味的に類似していることを判定する手法は少ない上に，種々の課題が残っている．

コード片の類似性判定法として，Zhao らは DeepSim という手法を提案している [40]．DeepSim は，制御フローグラフとデータフローグラフの情報を基に，深層学習モデルを用いて学習することでコード片の類似性を判定する．DeepSim は構文的に異なるが処理内容の意味的に類似したコード片を高い精度で判定する．しかし一方で，DeepSim はモデルの学習に非常に時間がかかるという課題もある．

また，コードクローンの検出手法として，情報検索技術に基づくコードクローン検出法がある [38]．この手法は，情報検索技術の一種である TF-IDF を用いてコード片をベクトル化し，ベクトル空間上で距離が近いコード片を類似していると判定することで，コードクローンとして検出する．ベクトル空間上の距離尺度としてはコサイン類似度を用いる．ベクトル表現をコードクローン検出に用いることで，構文や字句単位で類似していなくてもベクトル空間で距離が近ければコードクローンとして検出することができる．この手法は高速な検出が可能であるが，構文的類似度が低いコードクローンの検出漏れが多い点が指摘されている [37, 39]．

これらの課題を解決するために，本研究では情報検索技術に基づくベクトル表現と深層学習モデルを組み合わせた，より高速かつ高精度なコード片の類似性判定法を提案する．提案手法は，字句解析や識別子分割などの軽量なソースコード解析を用いた前処理を行う．その後，情報検索技術に基づきコード片をベクトル表現に変換する．この手法は，制御フローやデータフローの解析を行う手法に比べて軽量に実行可能である．最後，深層学習モデルを用いてコード片が類似しているか否かを判定する．これにより，構文的に異なるが処理内容の

意味が類似しているコード片を，より高速かつ高精度に類似していると判定することが可能になる．

提案手法の有効性を確認するために，Google Code Jam[1] と BigCloneBench[33] の 2 つのデータセットを用いて評価実験を行った．Google Code Jam は Google が開催している競技プログラミングであり，同一問題に回答し，正解したソースコードは類似コードとみなすことができる．BigCloneBench はコードクローンの大規模ベンチマークであり，600 万以上のクローンペア（互いに処理が類似したコードクローンの対）と 26 万以上の非クローンペアが登録されている．これらのデータセットに対し，提案手法と最新手法の DeepSim の比較を行ったところ，情報検索技術の一種であり，主成分分析による潜在的意味解析を行う LSI（Latent Semantic Indexing）[1] と深層学習モデルを組み合わせた提案手法が，適合率，再現率，F 値においてより高い値となった．また，実行時間を DeepSim と比較して，学習時間においては約 350 倍高速であり，推定時間においては同等であることが確認できた．

[1] https://codingcompetitions.withgoogle.com/codejam/

2　背景

本研究の提案手法は，情報検索技術に基づくベクトル表現と，順伝播型ニューラルネットワークを用いる．本章では，研究背景を説明する．以降，2.1 節では情報検索技術に基づくベクトル表現について，2.2 節では順伝播型ニューラルネットワークについて，2.3 節では提案手法と同様に深層学習を用いたコード片の類似性の判定手法について述べる．

2.1　情報検索技術に基づくベクトル表現

情報検索とは，コンピュータを用いて大量のデータ群から目的に合致した情報を取り出すことであり，自然言語で書かれた文書も情報検索を行う対象となる [1]．情報検索の分野において，ベクトル空間モデルを用いて意味的に類似した文書を検索する手法がある．ベクトル空間モデルを用いる手法は，文書を多次元の特徴ベクトルで表現することにより，文書間の意味的な類似性の判定をベクトル間の類似度計算に帰着させた手法である [18]．このベクトル空間モデルで使用されるベクトル表現を，情報検索技術に基づくベクトル表現と呼ぶ．

Yokoi らは情報検索技術に基づく様々なベクトル表現をソースコードに対して適用し，コードクローン検出においてそれぞれのベクトル表現の特徴を調査した [39]．この調査では，調査対象として表 1 に示す 7 つのベクトル表現を使用した．この調査で使用されたベクトル表現のうち，BoW (Bag-of-Words)[9] は出現する単語集合であり，伝統的かつ単純な手法として従来より利用されている．BoW に対して単語の重要度を付与した手法として，TF-IDF (Term Frequency and Inverse Document Frequency)[1] がある．TF-IDF は文書頻度（単語を含む文書数）の逆数を基に単語を重み付けする．この重み付けにより，多数の文書に共通して出現する単語には小さい重みが，特定の文書に偏って出現する単語には大きい重みが付与される．BoW や TF-IDF はベクトルの次元数が出現単語の総数と等しい．したがって，文書の数が増えるに従いベクトルが高次元化し，空間計算量や時間計算量が増大する課題が発生する．この課題を解決するため，高次元のベクトルを低次元空間に射影する次元圧縮が行われる．LSI は主成分分析によって潜在的意味解析を行い，次元圧縮を行う．さらに潜在的意味解析を行うことにより，文書中に同じ単語を含んでいない場合も，意味的に近い文書を類似していると判定することができる．しかし LSI の圧縮後の各次元の値は，主成分分析を行った計算結果に過ぎず値の意味が見えづらい．そこで LDA (Latent Dirichlet Allocation)[4] はこの課題を解決するために，LSI にベイズ学習を取り入れ，文書が各トピックに属する確率分布を生成する．そのため LDA はトピックモデルとも呼ばれる．また，機械学習を用いて単語の分散表現を得る手法として Word2Vec[22, 23] がある．Word2Vec は単語の意味や類似性を高精度に表現できる単語ベクトルである．そして Word2Vec のベクトル空間モデルを文書ベクトルへと拡張した手法として，Doc2Vec[20] が提案され，自然言語文書を対象とし

表 1: 情報検索技術に基づくベクトル表現

	次元圧縮	機械学習	その他特徴
BoW			出現する単語集合
TF-IDF			文書頻度の逆数で重み付け
LSI	✓		主成分分析により潜在的意味解析
LDA	✓		ベイズ学習により潜在ディリクレ配分
Doc2Vec	✓	✓	機械学習による文書ベクトル
WV-avg	✓	✓	単語ベクトル Word2Vec の平均ベクトル
FT-avg	✓	✓	単語ベクトル FastText の平均ベクトル

表 2: ベクトル表現ごとの再現率

クローンタイプ	BoW	TF-IDF	LSI	LDA	Doc2Vec	WV-avg	FT-avg
T1	0.99	0.99	0.99	0.99	0.99	0.99	0.99
T2	0.84	0.82	0.92	0.85	0.91	0.95	0.94
VST3	0.90	0.82	0.91	0.95	0.83	0.97	0.93
ST3	0.45	0.37	0.61	0.61	0.46	0.84	0.79
MT3	0.06	0.03	0.09	0.23	0.04	0.55	0.43
WT3/T4	0.00	0.00	0.00	0.02	0.00	0.08	0.05

た機械学習において有効性が確認されている [20]．また文書ベクトルとして，文書に出現する全単語のベクトルの平均を用いる手法がある．単語ベクトル Word2Vec の平均ベクトルを WV-avg (Word2Vec average)，Word2Vec から派生した単語ベクトル FastText[5] の平均ベクトルを FT-avg (FastText average) という．

Yokoi らはこれらのベクトル表現を用いて，大規模コードクローンベンチマーク BigCloneBench[33] に対してコードクローン検出を行い，その再現率を比較することで，コード片の意味を良く表すベクトル表現を調査した．この調査ではベクトル表現のコサイン類似度が 0.9 以上になるコード片のペアをコードクローンとした．また，Svajlenko らが提案したコードクローンの分類方法 [33]（4.1 節にて詳細に後述）を採用し，コードクローンの編集度合いに応じて 6 つのタイプに分類した．

BigCloneBench を用いた再現率の評価結果を表 2 に示す．この表から，次元圧縮や機械学習を行うことで，コードクローン検出の再現率が上昇し，その中でも特に機械学習を行う WV-avg や FT-avg の再現率が高いことがわかった．

また，各ベクトル表現の計算時間の比較も行った．その結果を表 3 に示す．この表から，

表 3: ベクトル表現ごとの計算時間（秒）

ベクトル表現	BoW	TF-IDF	LSI	LDA	Doc2Vec	WV-avg	FT-avg
ベクトル生成時間（秒）	5.1	10.0	9.7	60.3	44.7	42.7	196.1
類似度計算時間（秒）	5.5	5.1	1.1	1.1	1.6	1.1	1.1

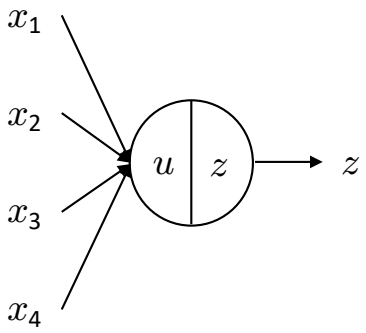

図 1: 4 つの入力を受け取るユニット
（参考文献 [24] より図を引用）

ベクトルの生成速度においては単純な手法である BoW が最も速いが，次元圧縮を行う LSI は TF-IDF と同等の速度であり，他の次元圧縮や機械学習を行うベクトル表現と比較してより高速な手法であることが確認できた．また機械学習を行う手法の中では，WV-avg が高速であった．類似度の計算時間においては，次元圧縮を行った手法は計算時間が短縮しており，計算時間の観点からは総体的に LSI の有用性が見られた．

2.2 順伝播型ニューラルネットワーク

多層パーセプトロンとも呼ばれる順伝播型ニューラルネットワークは，最も基本的かつよく使われているニューラルネットワークであり，情報が入力側から出力側に一方向にのみ伝播する [13, 24]．ネットワークの各層は，ニューロンと呼ばれる複数のユニットで構成されており，各ユニットは図 1 のように複数の入力を受け取り 1 つの出力を計算する．各入力 x に対して重み w を掛け，バイアス b を足したものが，ユニットの総入力である．ユニットが受け取る総入力 u は以下の式で表される．

$$u = \sum_{i=1}^{I} w_i x_i + b$$

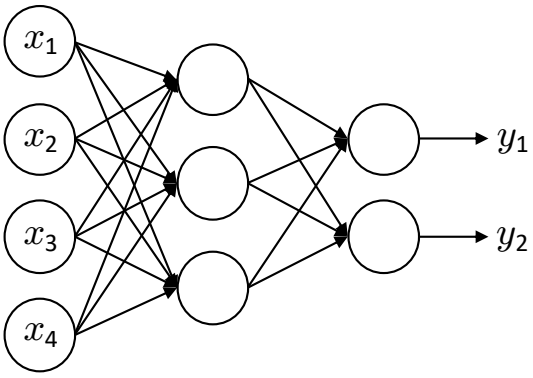

図 2: 3 つのユニットを中間層に持つ多層ネットワーク
（参考文献 [24] より図を引用）

この総入力 u に対して，活性化関数と呼ばれる関数 f の出力がユニットの出力である．ユニットの出力 z は，以下の式で表される．

$$z = f(u)$$

活性化関数とは，入力の総和をどのように活性化（発火）させるかを決定する役割があり，一般的に単調増加する非線形関数が使われる．古くからよく使われる活性化関数として，ロジスティック関数（あるいはシグモイド関数とも呼ばれる）がある．ロジスティック関数は (0,1) を値域として持ち，入力の絶対値が大きくなると出力が飽和し一定値となる特徴を持つ．ロジスティック関数は以下の式で表される．

$$f(u) = \frac{1}{1 + e^{-u}} \tag{1}$$

他にも活性化関数として様々な種類が存在するが，近年は ReLU[8] が最もよく用いられる．ReLU は単純で計算量が小さく，精度を低下させることなく学習が速く進むという特徴を持つ．ReLU は以下の式で表される．

$$f(u) = \max(0, u) \tag{2}$$

上記のユニットが複数集まって 1 つの層を構成し，さらに複数の層を重ねることで図 2 に示すような多層構造のネットワークになる．ネットワークは入力層，中間層（あるいは隠れ層とも呼ばれる），出力層からなる構造をしており，2 層以上の中間層で構成されるネットワークで機械学習することを，一般的に深層学習という．

順伝播ニューラルネットワークが表現する関数は，ネットワークの重み w を調整することで変化する．ネットワークが表す関数の推定値と学習データの正解値の近さを測る尺度を損

失関数という．問題の種別に応じて損失関数は変わるが，入力に応じて 2 種類に区別する二値分類では以下の損失関数 E が用いられる．ここでは，d_n は正解値，y_n は推定値とする．

$$E(w) = -\sum_{n=1}^{N}\{d_n \log y_n + (1-d_n)\log(1-y_n)\} \tag{3}$$

上記の損失関数に対し最小値を与える重み w を求めて最適化することで，ニューラルネットワークの学習を行う．損失関数を最適化するために，降下勾配法を用いて反復計算を行い，w を更新する．降下勾配法は以下の式で重み w を更新する．

$$w^{(t+1)} = w^{(t)} - \alpha\frac{\partial E(w)}{\partial w}$$

ここで α は w の更新量の大きさを定める定数であり，学習係数と呼ばれる．学習の初期ほど大きな学習係数を選び，学習の進捗とともに学習係数を小さくする方法が，一般的によく用いられる [17]．

しかし，ニューラルネットワークの層が多い場合，計算量が大きくなり勾配計算が困難になる．この課題を解決するために，誤差逆伝播法 [29] が用いられる．誤差逆伝播法は以下の 3 つのステップからなる．

1. 現時点での重みを用いて出力を計算する（順伝播）
2. 出力層から入力層の方向へ，各層での誤差を計算する（逆伝播）
3. 各層で計算した誤差を用いて勾配を計算し，重み w を更新する

2.3 深層学習を用いたコード片類似性の判定法

深層学習を用いたコード片の類似性判定法として，Zhao らは DeepSim[40] を提案している．DeepSim の概要を図 3 に示す．この図からわかるように，DeepSim は前半の意味表現生成過程と後半の類似性判定過程に分かれる．前半の意味表現生成過程では，制御フローグラフとデータフローグラフを解析し，特徴行列に変換することでコード片の意味表現を生成する．後半では，ニューラルネットワークの一種である自己符号化器 [11] を用いて，特

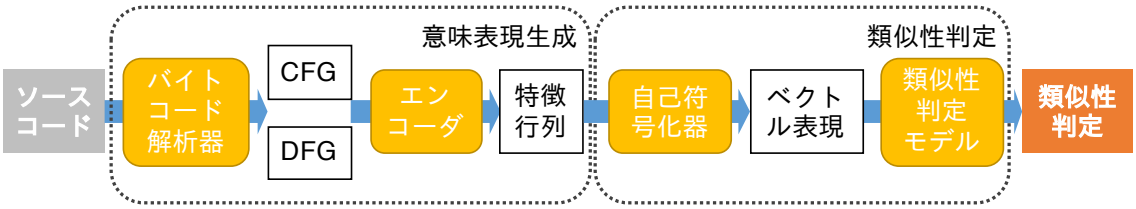

図 3: DeepSim の概要

徴行列の情報を最もよく表すベクトル表現に変換する．その後，ベクトル表現を類似性判定モデルに入力し，2 つのコード片が類似しているか否かの二値分類を行う．DeepSim は深層学習ベースのモデルを構築することで，構文的に異なるが意味的に類似したコード片を，高い精度で類似していると判定することができる．しかし，DeepSim はモデルの学習に非常に時間がかかるという課題点があげられる．実験では学習が終了するまでに，Intel Core i7 4.0GHz 4 コア CPU，NVIDIA GeForce GTX 1080 GPU という実験環境で約 3.7 時間かかっている [40].

深層学習を用いたコードクローン検出ツールとして，Saini らは Oreo[30] を提案している．Oreo は深層学習，情報検索技術，およびソフトウェアメトリクスを組み合わせたコード片の類似性判定法を用いることで，構文的に異なるが意味的に類似したコードクローンを検出する．Oreo は深層学習の手法として，Siamese アーキテクチャニューラルネットワーク [2] を用いている．Siamese アーキテクチャは，2 つの対象の類似性を比較する問題に適したニューラルネットワークであり，指紋の画像データを照合する際などに用いられる [2]．Oreo は事前に学習を行い，学習済みのモデルを用いることで，クローン検出の段階では学習を行わずにコードクローンの検出が行える．事前学習の段階では，クローンペア（互いに処理が類似したコードクローンの対）と非クローンペアのソフトウェアメトリクスを入力値として学習する．クローンペアと非クローンペアの情報は，GitHub から取得したプロジェクトに対し，コードクローン検出ツール SourcererCC[31] のクローン検出結果を用いることで作成する．Oreo は既存のコードクローン検出法と比較して，構文上の類似性が低いが意味的に類似しているコードクローンも高い精度で検出が可能である．

また深層学習を用いたコード片検索法として，藤原らは順伝播型ニューラルネットワークを用いて類似コード片を検索する手法 [7] を提案している．藤原らの手法は，最初にコードクローン検出ツール CCFinder[15] および CCVolti[38] を用いて類似コードブロックセットを作成し，さらに，ミューテーションオペレータを適用して類似コードブロックの増量も行う．次に BoW[9] や Doc2Vec[20] を用いてコード片を特徴ベクトルに変換する．その後，同一の類似コードブロックセットに属するコード片に対して同一のラベルを付与し，順伝播型ニューラルネットワークにより多クラス分類問題として学習する．ここでコードブロックとは，関数などの “{ }” で囲まれた範囲のコード片のことである．また，構文的に一致，または差異はあるが一致部分も存在するコードブロックの同値類を，類似コードブロックセットという．コード片検索の段階では，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し，学習済みモデルにて推定されたラベルと同一のラベルを持つ類似コードブロックセットを検索結果として出力する．藤原らの手法を 3 つのオープンソースソフトウェアに適用した結果，構文的に差異のある類似コード片を高い精度で検索できることが明らかになった．

3　提案手法

2.1 節で述べた情報検索技術に基づくベクトル表現を用いて，コードクローン検出を行う手法 [38] があるが，この手法は高速な検出が可能である一方で，構文的類似度が低いコードクローンの検出漏れが多い点が課題として指摘されている [37, 39]．また，2.3 節で述べた深層学習を用いたコード片の類似性判定法の中でも，DeepSim は既存手法と比較して高い精度が得られた一方で，学習に時間がかかるという課題が明らかになった．

そこで本研究では，情報検索技術に基づくベクトル表現と深層学習を組み合わせることで，コード片の類似性を判定する手法を提案する．DeepSim と提案手法の相違点は，DeepSim は制御フローグラフとデータフローグラフから求めた特徴を自己符号化器に入力してベクトル表現を得るのに対し，提案手法はコード片の単語列から情報検索技術に基づきベクトル表現を得る点である．提案手法は主に以下の 3 つのステップで実行される．提案手法の概要を図 4 に示す．

STEP 1 ソースコード解析を用いた前処理を行い，コード片を単語列に変換する

STEP 2 情報検索技術に基づき単語列をベクトル表現に変換する

STEP 3 ベクトル表現を類似性判定モデルに入力し，深層学習によりコード片の類似性を判定する

以降，3.1 節では，ソースコード解析を用いた前処理について，3.2 節では，情報検索技術に基づくベクトル化について 3.3 節では，深層学習を用いた類似性判定モデルについて述べる．

3.1　STEP 1：ソースコード解析を用いた前処理

提案手法では，入力コード片をベクトル表現に変換するために，前処理を行なう．コード片は文字の羅列であり構造化されていないため，そのままでは解析することが困難である．そのため，前処理としてコード片を解析可能なデータの型に変換することは，精度向上のために重要な手段である．前処理は自然言語で書かれた文書をベクトル化する際にも一般的に

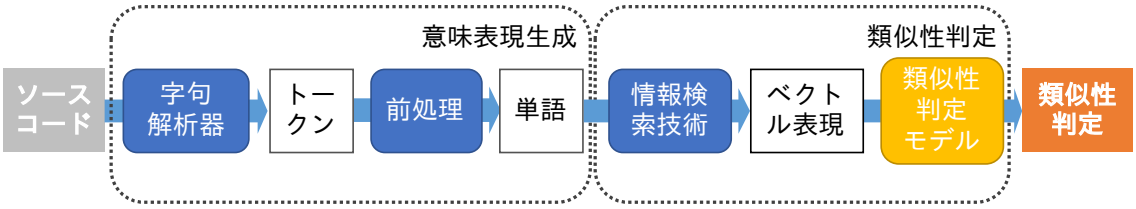

図 4: 提案手法の概要

行われる．提案手法ではソースコードを解析可能なデータに変換するため，以下の前処理を行う．

- コメントの除去
- コード片をトークン単位に分割
- 予約語と識別子以外を除去
- 識別子名をキャメルケースやスネークケースを基に分割
- 分割後の識別子をすべて小文字に正規化

本ステップでは最初に入力コード片に対して字句解析を行い，コメントの除去とトークン単位への分割を行う．開発者がコード片の処理内容をコメントに記述する場合もあるが，処理に直接的な影響を与えないためコメントは解析の対象外として除去する．また，改行や空白の有無やフォーマットの差異を排除するため，コード片の字句解析を行いトークン単位への分割を行う．字句解析には，構文解析器生成系 ANTLR[2]が生成した字句解析器を用いる．ANTLR 用に複数言語の構文定義記述がウェブサイト[3]で公開されており，これを用いることで容易に複数言語の解析に対応できる．本研究では Java を解析対象の言語としている．

次に，字句解析により得られた情報により，予約語と識別子以外を除去する．セミコロンなどコード片の処理内容の意味に影響を与えない記号を除去することにより，より正確にコード片の意味を表すことができる．また提案手法の特徴として，演算子の除去も行う．演算子を除去することにより，加算と減算が反転しているようなコード片も類似していると判定することができる．

最後に，識別子をプログラミング言語の命名規則を基に分割し，さらに小文字変換の正規化を行う．空白や句読点を使用して単語を区切る自然言語とは異なり，識別子に空白を含めることはできない．したがって識別子を分割する一般的な方法として，プログラミング言語の命名規則に従う方法がある．例えば Java の開発者は，単語が大文字または非アルファベット文字で区切られているキャメルケースをよく使用する．他言語では，アンダーバーなどで単語を区切るスネークケースが使用されることもある．さらに小文字変換により，大文字と小文字の表記ゆれがある単語も区別なく同一の単語として扱えるようになる．これらの識別子分割や正規化は，自然言語処理によりソースコードを解析する際に有用な手法として用いられる [6]．

[2]https://www.antlr.org/
[3]https://github.com/antlr/grammars-v4/

表 4: ベクトル表現とパラメータ

ベクトル表現	パラメータ
LSI	トピック数:200
LDA	トピック数:100
PV-DBoW	ベクトルサイズ:300，ウィンドウサイズ:15，エポック数:20
PV-DM	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数:20
WV-avg	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数:20

3.2 STEP 2：情報検索技術に基づくベクトル化

本研究では情報検索技術に基づくベクトル表現として，2.1 節で述べた 7 つのベクトル表現の内，LSI，LDA，Doc2Vec，WV-avg の 4 つのベクトル表現を研究対象として用いる．これらのベクトル表現は，既存研究においてコード片の類似性判定法として用いられている [37, 39]．BoW と TF-IDF は 2.1 節でも述べたように文書の数が増えるに従いベクトルの次元数も増加する特徴がある．これにより，類似性判定モデルに入力するベクトルの次元数が固定されないという点を理由に，BoW と TF-IDF を本研究の対象から除外した．また 2.1 節の調査結果より，FT-avg は WV-avg より再現率が低いかつ計算速度も遅いことが判明したため，FT-avg も本研究の対象から除外した．ただし，Doc2Vec の実装として PV-DBoW (Distributed Bag of Words version of Paragraph Vector) と PV-DM (Distributed Memory version of Paragraph Vector) の 2 つの異なるアルゴリズムが提案されている [20] ため，本研究ではこの 2 つのアルゴリズムを別個のベクトル表現として扱う．したがって，LSI，LDA，PV-DBoW，PV-DM，WV-avg の計 5 つのベクトル表現を研究対象とする．また Word2Vec の実装としても複数のアルゴリズムが提案されているが，本研究では SGNS(skip-gram algorithm with negative sampling)[23] を用いることとする．

本研究で対象とするベクトル表現と，そのベクトル表現に与える主なパラメータを表 4 に示す．この表で示すパラメータは，実装に用いた Python ライブラリ gensim[25][4]のデフォルト値を参考に決定した．

3.3 STEP 3：深層学習を用いた類似性判定モデル

与えられた 2 つのコード片の類似性を判定する方法として，コサイン類似度やユークリッド距離のような距離尺度を用いる手法がある．この手法は既存の手法でもよく用いられてきた [14, 36, 38]．しかし，このような距離尺度には閾値をヒューリスティックに決める必要があり，最適な閾値を機会学習により求めることができないという課題点がある．

[4]https://radimrehurek.com/gensim/

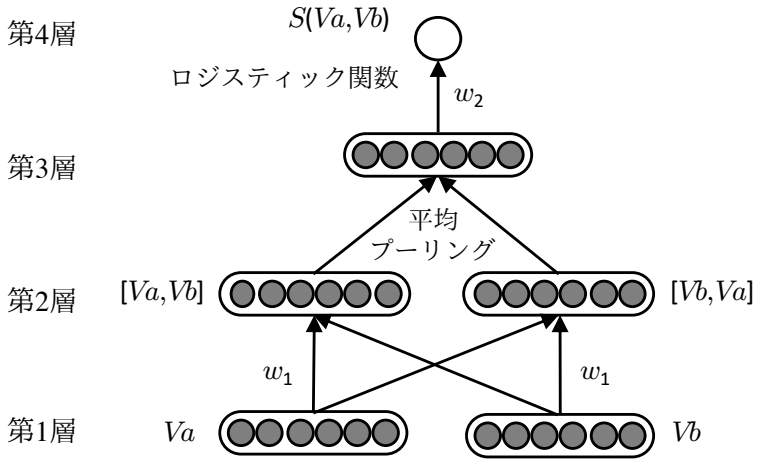

図 5: コード片類似性判定モデルのアーキテクチャ

そこで本研究では深層学習モデルを用いることで，コード片の類似性を判定する手法を提案する．提案手法の類似性判定モデルでは，ニューラルネットワークによる二値分類問題に置き換えることにより，類似性判定を行う．ニューラルネットワークとして様々なアーキテクチャが存在するが，提案手法では順伝播型ニューラルネットワークを基に，コード片類似性判定モデルを設計した．このコード片類似性判定モデルのアーキテクチャを図 5 に示す．この図から分かるように，このネットワークには 2 つのコード片から得られたベクトル表現 V_a と V_b を入力する．その後第 2 層で，入力した 2 つのベクトルを $[V_a, V_b]$ と $[V_b, V_a]$ となるように異なる順序で連結する．このとき，2 つの連結ベクトルは重み w_1 を共有して計算される．第 3 層は平均プーリング層である．2 つの層からなる第 2 層の出力値の平均値を第 3 層への入力として計算する．最後は出力層としてロジスティック関数（2.2 節の式 1 参照）を用いてコード片の類似性を出力する．中間層の活性化関数としては ReLU（2.2 節の式 2 参照）を用いる．損失関数としては 2.2 節の式 3 を用いてネットワーク全体を最適化する．これにより，構文が異なるコード片の類似性を学習することができる．このニューラルネットワークアーキテクチャを実現するために，本研究では TensorFlow[5] と Keras[6] を用いて実装した．

2 つの異なる順序でベクトルを連結し平均プーリングを行うことは，類似性を判定するネットワークにおいて重要な操作である．類似性判定関数 S において，$S(V_a, V_b) = S(V_b, V_a)$ の対称性が成立することが必要である．しかし，$[V_a, V_b]$ または $[V_b, V_a]$ の片方の順序のみ使用する場合，(V_a, V_b) と (V_b, V_a) の両方の入力を学習する必要性が生じ，ネットワークの学習や推定に約 2 倍時間がかかることにつながる．

[5] https://www.tensorflow.org/
[6] https://keras.io/

4　評価実験

本章では，本研究で提案したコード片の類似性判定法の評価実験について述べる．評価実験では精度と実行時間における提案手法の有用性を評価するため，再現率，適合率，F 値，計算時間の観点から，既存手法の比較を行った．以降，4.1 節では評価実験に用いる対象のデータセットについて，4.2 節では評価実験で用いる比較対象とその実装について，4.3 節では Google Code Jam を用いた精度評価について，4.4 節では Google Code Jam を用いた実行時間評価について，4.5 節では Big Clone Bench を用いた精度評価について述べる．次に，4.6 節では意味表現の生成時間評価について述べ，最後に 4.7 節では評価実験の結果に対して考察を行う．

4.1　評価対象データセット

本研究では提案手法を Google Code Jam と BigCloneBench[33] の 2 つのデータセットを用いて評価実験を行った．

Google Code Jam

本研究の最初の実験では，DeepSim の実験 [40] と同様に Google Code Jam のデータセットを用いて提案手法の評価を行う．Google Code Jam とは，Google[7]が実施している競技プログラミングであり，提出されたソースコードは Google によって正しい挙動をするか確認される．これにより，異なるプログラマが提出したソースコードも同一の動作を行うことが Google によって保証されており，これらは処理内容が類似したソースコードとみなすことができる．表 5 は本研究で実験対象の Google Code Jam のデータセットの情報を示す．この表に示すように，本研究では 12 種の問題から集めた 1,669 個の提出コードを用いて実験した．本研究で用いた 12 種の問題は，すべて異なる問題であることが確認されている [40]．したがって，同一問題に対して提出されたソースコードは類似コード，異なる問題に対して提出されたソースコードは非類似コードとみなすことができる．なお，競技プログラミングに提出されたソースコードの特徴として，ソースコードによって関数分割の粒度に差がある．1 つの関数にまとめて処理を記述したソースコードもあれば，複数の関数に分割して記述したソースコードもある．したがって，本研究では Google Code Jam に提出されたソースコードを main 関数にインライン展開し，展開後の main 関数を実験の対象として抽出した．

[7] https://about.google/

表 5: Google Code Jam データセット

問題数	12
提出コード数	1,669
LOC	98,117
トークン数	445,371
語彙数	4,803
類似コードペア	275,570
非類似コードペア	1,116,376

BigCloneBench

本研究の 2 番目の実験では，コードクローン検出ツールの評価実験で頻繁に用いられる BigCloneBench[33] を用いて評価を行う．BigCloneBench とは，600 万以上のクローンペアと約 26 万の非クローンペアを含む大規模なコードクローンベンチマークである．BigCloneBench には関数単位のコードクローンが登録されており，コードクローンの編集度合いに応じて手作業でタイプに分別されている．コードクローンのタイプの分類として Roy らが提案した 4 つの分類 [28] が一般的に用いられるが，この分類法には定量的にコードクローンを分類できないという課題点がある．そこで Svajlenko らは定量的に分類可能な分類法 [34] を提案し，BigCloneBench にて採用している．その 6 つの分類を以下に示す．ここではタイプ 1 とタイプ 2 の正規化後に，行またはトークンが一致する最小比率のうち，小さい方を構文的類似度として用いる．

T1 (Type 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

T2 (Type 2) タイプ 1 の違いに加えて，変数名などのユーザー定義名，変数の型などが異なるコードクローン

VST3 (Very-Strongly Type-3) タイプ 2 の違いに加えて，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（構文的類似度 90%以上 100%未満）

ST3 (Strongly Type-3) タイプ 2 の違いに加えて，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（構文的類似度 70%以上 90%未満）

MT3 (Moderately Type-3) タイプ 2 の違いに加えて，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（構文的類似度 50%以上 70%未満）

WT3/T4 (Weakly Type-3/Type-4) タイプ 2 の違いに加えて，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（構文的類似度 0%以上 50%未満）

本研究では DeepSim の評価実験と条件を揃えるため，クローンペアまたは非クローンペアとタグ付けされていないコード片と，5 行未満のコード片を対象外とした．行数の小さいコード片はコードクローンとして保守対象となりにくく，一般的にコードクローンの評価実験において対象外にされることが多い．したがって，約 550 万のクローンペアと約 20 万の非クローンペアからなる，約 5 万のコード片のデータセットを作成した．このデータセットの内，ほとんどが WT3/T4 であった．

4.2 比較評価対象

本研究では，3.2 節で説明した 5 類（LSI，LDA，PV-DBoW，PV-DM，WV-avg）のベクトル表現を用いた提案手法との比較評価対象として，最新手法の DeepSim を選択した．提案手法で採用するベクトル表現の候補は複数あり，その中でより有用なベクトル表現を得るために，情報検索技術に基づくベクトル表現として 5 種類選択して比較評価対象とした．さらに提案手法以外にも，コード片の類似性判定手法の最新手法である DeepSim を比較評価対象として加えた．

情報検索技術に基づくベクトル表現

提案手法では，情報検索技術に基づくベクトル表現を図 5 の深層学習モデルに入力することで，コード片の類似性を学習・推定する．3.2 節でも述べたように複数のベクトル表現を提案手法で採用する候補としており，その中で精度と実行時間の観点からより有用なベクトル表現を調べるため，5 種類のベクトル表現それぞれを選択した手法を評価対象とする．本論文では，深層学習モデルに入力した 5 種のベクトル表現を区別するために，以下の表記を用いる．NN はニューラルネット（Neural Network）の略である．

- LSI+NN (Latent Semantic Indexing + Neural Network)
- LDA+NN (Latent Dirichlet Allocation + Neural Network)
- PV-DBoW+NN (Distributed Bag of Words version of Paragraph Vector + Neural Network)
- PV-DM+NN (Distributed Memory of Words version of Paragraph Vector + Neural Network)
- WV-avg+NN (Word2Vec average vector + Neural Network)

表 6: 手法ごとのハイパーパラメータ設定

手法	パラメータ
DeepSim	レイヤーサイズ:8-6，(128x6-256-64)-128-32，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003
	ドロップアウト:0.75
LSI+NN	トピック数:200
	レイヤーサイズ:200-100，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003
LDA+NN	トピック数:100
	レイヤーサイズ:100-100，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003
PV-DBoW+NN	ベクトルサイズ:300，ウィンドウサイズ:15，エポック数 (PV-DBoW):20
	レイヤーサイズ:300-100，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003
PV-DM+NN	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数 (PV-DM):20
	レイヤーサイズ:300-100，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003
WV-avg+NN	ベクトルサイズ:300，ウィンドウサイズ:5，エポック数 (Word2Vec):20
	レイヤーサイズ:300-100，エポック数:4
	初期学習率:0.001，L2 正則化の λ :0.00003

DeepSim

また評価実験の比較対象として，提案手法と同様に深層学習モデルを用いたコード片の類似性判定法である DeepSim を用いた．DeepSim は最新のコード片類似性判定法であり，他の既存手法と比較して高い精度で意味的に類似したコード片の類似性を判定できる [40]．深層学習モデルで用いたハイパーパラメータを表 6 に示す．提案手法は，DeepSim のハイパーパラメータを参考にし，さらに提案手法の入力ベクトルのサイズも考慮した上で決定した．

4.3 Google Code Jam を用いた精度評価

Google Code Jam を用いた精度評価では，再現率，適合率，F 値を評価指標として DeepSim と提案手法を比較した．再現率とは，正解集合に対して実際に正しく正解と推定された割合を指し，網羅性に関する指標として用いられる．また適合率とは，正解として推定された集合に対して真に正しい割合を指し，正確性を示す割合である．一般的に再現率と適合率はト

表 7: 精度評価で用いる特異度

推定＼正解	類似している	類似していない
類似している	真陽性	偽陽性
類似していない	偽陰性	真陰性

レードオフの関係にあり，一方の値が高くなるともう一方の値が低くなる傾向にある．したがって再現率と適合率の総合的な評価指標が必要となる．F 値とは，再現率と適合率の調和平均として求められ，再現率と適合率の総合的な指標として用いられる．これらの指標は，以下の式で表される．

$$\text{再現率} = \frac{\text{真陽性}}{\text{真陽性} + \text{偽陰性}}$$

$$\text{適合率} = \frac{\text{真陽性}}{\text{真陽性} + \text{偽陽性}}$$

$$F\ \text{値} = \frac{2 \times \text{再現率} \times \text{適合率}}{\text{再現率} + \text{適合率}}$$

表 7 の本実験の精度評価で用いる特異度が示すように，真陽性とは，正解データとモデルの推定データが両者とも “類似している” で一致している場合を意味する．また，正解データは “類似している” が推定データは “類似していない” 場合は偽陰性，反対に正解データは “類似していない” が推定データは “類似している” 場合は偽陽性となる．正解データとモデルの推定データが両者とも “類似していない” で一致している場合は真陰性となる．

本研究では，10 分割交差検証を用いて学習とテストを行い，学習モデルの精度を推定する．10 分割交差検証では，データセットを 10 分割し，そのうちの 1 つをテスト用データ，残りの 9 つを学習用データとして検証を行う．この手順をテスト用データと訓練用データの組み合わせを変えて 10 回繰り返す．こうして得られた結果を平均して精度の推定を行う．なお本実験は，DeepSim の論文 [40] と同様の条件設定で行っている．したがって，DeepSim の再現率，適合率，F 値の値は論文より引用した．

Google Code Jam を用いた精度評価の結果を表 8 に示す．この表から分かるように，再現率に関しては 5 つの提案手法の中では LSI+NN が 92.8%と最も高く，さらに DeepSim の 81.6%よりも高い値が得られた．適合率に関しても，5 つの提案手法の中では LSI+NN が 96.0%と最も高く，こちらも DeepSim の 71.3%より高い値が得られた．また，LSI+NN が提案手法の中で F 値が 0.94 と最も高く，さらに DeepSim の 0.76 よりも上回った．

LDA+NN は再現率 54.1%，適合率 61.0%と両者とも最も低い値となった．LDA はベイズ統計に基づく確率的トピック生成モデルによって，コード片の潜在表現を求めるが，プロ

表 8: Google Code Jam を用いた精度評価

手法	再現率	適合率	F 値
DeepSim[40]	0.81	0.71	0.76
LSI+NN	**0.93**	**0.96**	**0.94**
LDA+NN	0.54	0.61	0.55
PV-DBoW+NN	0.88	0.86	0.86
PV-DM+NN	0.89	0.68	0.75
WV-avg+NN	0.91	0.90	0.88

グラミング言語は自然言語と比べてトピック数が少ないため，LDA の生成モデルがソースコードに対してうまく働かない可能性が理由として考えられる．

PV-DBoW+NN と WV-avg+NN は LSI+NN には及ばないが，DeepSim と比較すると再現率と適合率ともに高い値となっている．特に WV-avg+NN は LSI+NN と僅差であり，WV-avg+NN も充分に有用な手法である．

PV-DM+NN の再現率は 89.4%と PV-DBoW の再現率 88.3%とほぼ同等であるが，適合率が 68.2%と低く，DeepSim の適合率 71.3%より低い値となった．この理由として，一般的に PV-DM は PV-DBoW よりエポック数を増やす場合が多いが，今回はベクトル生成のエポック数を 20 に揃えたため，コード片の潜在表現が充分に求められなかった可能性がある．

全体を通してみると，LSI+NN，PV-DBoW+NN，WV-avg+NN の 3 つの手法が DeepSim と比較して，再現率と適合率ともに高い値が得られた．さらに，DeepSim の再現率と比較すると適合率が低くなっているのに対し，これら 3 つの手法は再現率と適合率が同程度の値を示しており，DeepSim より正確性の高い手法であることが確認できた．

4.4 Google Code Jam を用いた実行時間評価

本研究では精度の評価に加えて，各工程の実行時間を測定して評価を行った．本実験では，Google Code Jam のデータセット全体の 9 割を学習し，1 割を推定するために所要した時間を測定した．実験環境は，Intel Xeon E5 2.7GHz 4 コア CPU，NVIDIA Quadro 5000 GPU，32GB メモリのデスクトップパソコンで実験を行った．Google Code Jam を用いた実行時間の評価結果を表 9 に示す．また，情報検索技術に基づくベクトル化に所要する時間を明らかにするために，表 9 における学習時間の内訳を表 10 に示す．

学習時間に関しては，DeepSim が 70,503 秒と，約 20 時間かかることが分かった．DeepSim の学習時間の 13,525 秒 [40] と比較すると本実験では約 5.2 倍の時間が所要しているが，これは実験環境の CPU の性能差や使用用途の違いが理由として挙げられる．5 つの提案手法

表 9: Google Code Jam を用いた実行時間評価（秒）

手法	学習時間	推定時間
DeepSim	70,503 秒	27 秒
LSI+NN	**210 秒**	**21 秒**
LDA+NN	228 秒	**21 秒**
PV-DBoW+NN	224 秒	25 秒
PV-DM+NN	533 秒	25 秒
WV-avg+NN	256 秒	25 秒

表 10: Google Code Jam を用いた学習時間の内訳（秒）

手法	ベクトル化	類似性判定
LSI+NN	**0.6 秒**	209 秒
LDA+NN	27 秒	**201 秒**
PV-DBoW+NN	8 秒	216 秒
PV-DM+NN	318 秒	215 秒
WV-avg+NN	40 秒	216 秒

の中では LSI+NN が最も高速で 210 秒で学習が完了しており，DeepSim と比較して約 350 倍も高速に学習することができることが示せた．

また表 10 より，情報検索技術に基づくベクトル化にかかる時間が学習時間の差に大きく寄与することが分かった．表 6 が示すように，PV-DBoW+NN，PV-DM+NN，WV-avg+NN のベクトル生成のためのエポック数は全て 20 に固定し，ベクトル化にかかる時間を公平に比較したが，ベクトル化に所要する時間は大きく異なる結果となった．

推定時間に関しては，どの手法も同程度であり，提案手法が DeepSim より若干短時間で推定が完了した．なお，なお，5 つの提案手法の推定時間には，コード片をベクトル化する時間も含んでいる．DeepSim の特徴として推定が短時間で完了する点が挙げられていたが [40]，提案手法はより高速に推定が可能であることが示せた．

4.5 BigCloneBench を用いた精度評価

BigCloneBench を用いた精度評価では，再現率，適合率，F 値を評価指標として DeepSim と提案手法を比較した．再現率，適合率，F 値に関しては 4.3 節にて述べた指標と同様のものを使用し，10 分割交差検証により精度の評価を行う．なお本実験は，DeepSim の論文 [40] と同様の条件設定で行っている．したがって，DeepSim の再現率，適合率，F 値の値は論文

表 11: BigCloneBench を用いた精度評価

手法	再現率	適合率	F 値
DeepSim[40]	0.97	0.98	0.98
LSI+NN	0.999	0.995	0.997
LDA+NN	0.999	0.995	0.997
PV-DBoW+NN	0.999	0.999	0.999
PV-DM+NN	0.999	0.998	0.998
WV-avg+NN	0.999	0.998	0.998

より引用した．なお，DeepSim の値は有効桁数 2 桁までしか掲載されていないため，提案手法とは表中の有効桁数が異なる．

BigCloneBench を用いた精度評価の結果を表 11 に示す．再現率に関しては，5 つの提案手法のすべてが 99.9%と高く，さらに DeepSim の 97%と比較して高い値が得られた．適合率に関しても，5 つの提案手法の中では PV-DBoW が 99.9%と最も高く，残りの 4 つの手法に関しても 99.5%以上の値が得られ，こちらも DeepSim の 98%と比較して高い値が得られた．また，すべての提案手法の F 値が DeepSim の値を上回った．これらの結果から DeepSim も十分に高い精度となるが，提案手法はさらに高い精度で類似性の判定が可能であることが確認できた．

4.6 意味表現の生成時間評価

本研究ではさらに，意味表現生成過程の実行時間を測定し，評価した．図 3 が示すように，DeepSim は意味表現生成過程において，ソースコードをバイトコード解析にかけて制御フローグラフ（CFG）とデータフローグラフ（DFG）を生成し，その後エンコーダを用いて特徴行列を生成する．一方で，図 4 のように提案手法の意味表現生成過程では，字句解析によりトークンに分割した後，前処理によって単語列を生成する．このように DeepSim と提案手法では意味表現の生成方法が異なる．生成方法の違いによる生成時間の差を明らかにするため，表 12 に示す 6 つのオープンソースソフトウェアプロジェクトを用いて意味表現の生成時間を測定し，比較する．なお，提案手法はソースコードを解析対象とするのに対し，DeepSim はコンパイルされたバイトコードを解析対象とする．したがって，ソースコードと単一のバイトコードが得られるプロジェクトを本実験の対象として選択する必要がある．MvnRepository[8]では，ソースコードとバイトコードの両方が公開されているため，MvnRepository において利用者の多いプロジェクトの中から表 6 に示した 6 つのプロジェ

[8]https://mvnrepository.com/

表 12: 意味表現の生成時間評価に用いた対象プロジェクト

プロジェクト	バージョン	ファイル数	LOC
ANTLR	4.7.2	233	96,043
Apache Ant	1.10.5	787	92,219
Apache Commons Lang	3.8.1	153	27,646
Apache Log4j	1.2.17	213	21,050
JUnit	4.12	195	9,317
Guava	27.0.1	573	86,542

表 13: 意味表現の生成時間（秒）

プロジェクト	DeepSim	提案手法
ANTLR	メモリ不足	13 秒
Apache Ant	42 秒	12 秒
Apache Commons Lang	20 秒	6 秒
Apache Log4j	15 秒	4 秒
JUnit	9 秒	4 秒
Guava	29 秒	12 秒

クトを選択した．実験環境は，Intel Xeon E5 2.7GHz 4 コア CPU，32GB メモリのデスクトップパソコンで実験を行い，5 回測定した時間の平均値を求めた．

本実験の結果を表 13 に示す．この表より提案手法の方が 2 分の 1 から 3 分の 1 程度の実行時間で意味表現を生成できることが分かった．さらに DeepSim による ANTLR の解析時には，Java 仮想マシンのヒープ領域不足により意味表現の生成を完了できないことが判明した．なお，Java 仮想マシンの最大ヒープサイズを初期値の 8GB から 16GB まで増やして再度実行したが結果は変わらなかった．この評価実験より，DeepSim が行う制御フローグラフやデータフローグラフの解析と比較して，提案手法による字句解析や識別子分割などの前処理の方が高速に実行可能であることが確認できた．

4.7 考察

情報検索技術に基づくベクトル表現の比較

本研究では，Google Code Jam を用いて 4.3 節では精度評価を行い，4.4 節では実行時間評価を行った．情報検索技術に基づくベクトル表現を用いた 5 種類の提案手法の中では，4.3 節の表 8 より LSI+NN が再現率，適合率，F 値ともに最も高く，次いで WV-avg+NN も高

かった．ベクトル化に所要する時間に関しては，4.4 節の表 10 より LSI+NN が 0.6 秒と非常に短時間であった．2 番手の PV-DBoW+NN も 8 秒であり，残りの手法は 20 秒以上であることから，LSI+NN がベクトル化の観点から非常に高速であることがわかる．これらにより，5 種の提案手法の内，LSI+NN が最も有用性の高い手法であることが確認できた．

DeepSim と提案手法の比較

また，最新手法である DeepSim と比較しても，適合率，再現率，F 値の精度の観点と，学習や推定に所要する実行時間の観点から，提案手法の LSI+NN が有用であることが分かった．また LSI+NN に加えて，PV-DBoW+NN と WV-avg+NN も DeepSim より精度と実行時間の観点から，有用性が高いことが分かった．提案手法が高い精度が得られた理由として，人間はコードを書く際にソースコードに処理の意味を持たせることが挙げられる．人間は後からソースコードを読んだときに，その処理内容を理解できるようにソースコードを記述する．特にソフトウェアの保守性を高めるために，他人が読んでも理解しやすいソースコードを記述することが求められ，特にオープンソースソフトウェアの開発において重要視されている．一方で，開発後の保守は行わず，提出の速度が求められる競技プログラミングにおいて，ソースコードの理解のしやすさは二の次にされることが多く，また同一問題に提出されたソースコードも開発者によって異なる識別子名をつけられる可能性が高い．しかし Google Code Jam を用いた評価実験でも，提案手法が高い精度を出すという興味深い結果が得られた．本研究の実験においては，厳格なコーディング規則に則って開発を行わなくても，人間が書く以上そのソースコードに処理内容の意味が与えられ，情報検索技術に基づくベクトル表現がその処理内容の意味を充分に表現できることが分かった．

DeepSim は制御フローグラフとデータフローグラフを解析した特徴行列から，自己符号化器を用いてコード片のベクトル表現を求める．一方で，提案手法はコード片中の出現単語から情報検索技術に基づくベクトル表現を求め，それを類似性判定モデルに入力する．提案手法は自己符号化器を用いるのではなく，情報検索技術に基づいてベクトル化することにより，高速な学習が可能になった．特に，情報検索技術の一種である LSI にて用いられる主成分分析は，行列計算に置き換えることで高速に計算可能なアルゴリズムが提案されており，ネットワークの学習のために反復計算が必要な自己符号化器と比較して，高速に学習を行うことができる．

本研究の成果として，制御フローグラフとデータフローグラフの解析結果の特徴を自己符号化器に入力して得られるコード片のベクトル表現より，情報検索技術に基づくコード片のベクトル表現の方が，高速にかつ処理内容の意味をよく表現できることが分かった．

提案手法の拡張性

本研究の評価では交差検証を行うことにより，モデルが過学習していないことを確認しているが，学習とテストを同じデータセット内にて行っている．学習したデータセット以外でも高い精度が得られる，より汎化性能が高いモデルの作成も課題として挙げられる．実際に提案手法を学習とテストでデータセットを変更し評価実験を行った．BigCloneBench のデータセットで学習し，Google Code Jam のデータセットでテストした結果，再現率 0.95，適合率 0.21，F 値 0.34 となった．また，Google Code Jam のデータセットで学習し，BigCloneBench のデータセットでテストした結果，再現率 0.63，適合率 0.97，F 値 0.76 となった．この結果は，4 章の同一データセット内の評価結果と比較すると F 値が低くなっており，提案手法は学習したデータセット以外では高い精度が得られていないことが分かる．これは，データセット内のコード片には偏りが存在することが理由として挙げられる．実際に，BigCloneBench に登録されているコード片の機能は 10 種類しかなく [33]，また Google Code Jam のデータセットには 12 種類の問題しか含まれていない（表 5 参照）．しかしあらゆる種類の機能を持つコード片のデータセットの作成は現実的に困難であるため，学習したデータセット以外でも高い精度が得られる，より汎化性能が高いモデルの作成は課題である．

さらに提案手法は，入力した 2 つのコード片ペアと類似しているか否かのタグ情報を学習する教師あり学習モデルである．しかし実際には，コード片が類似しているか否かのタグ付けは手作業で行う必要性があり，容易な作業ではない．またコードクローン検出が行われる目的とは，検出対象内に類似したコード片が存在するか否かを自動的に検出することであり，コードクローン検出のためにコード片が類似しているか否かのタグ付けを行うことは，手段と目的が入れ替わる行為である．本研究で評価実験として用いたように，競技プログラミングの同一問題への回答を類似コードとして仮定したり，既存のコードクローン検出ツールの検出結果を基にしたりした，簡易的なタグ付けの結果を教師あり学習する手法は存在するが，学習済みでないデータセット内からコード片の類似性を自分で学習する教師なし学習モデルの作成も課題として挙げられる．

5 関連研究

類似したコード片を自動で検出する手法としてコードクローン検出法がある．コードクローン検出として，字句単位の検出手法 [15, 21, 31] や，テキストベースの検出手法 [26, 27]，抽象構文木（ソースコードの構文構造を木構造で表したグラフ）を用いた検出手法 [3, 14] などが存在する．

また筆者は情報検索技術に基づくベクトル表現を用いてコード片をベクトル空間上に射影し，ベクトル空間上で近接しているコード片をコードクローンとして検出する手法 [38] を提案した．この手法は CCVolti[9]という名のコードクローン検出ツールとして実装されている．CCVolti は情報検索技術の一種である TF-IDF を用いてベクトル化を行っており，既存のコードクローン検出ツールと比較して構文的に異なるが処理内容が類似しているコードクローンをより多く検出できることを示している．さらに，TF-IDF 以外の次元圧縮や機械学習を行うベクトル表現を用いた場合，より多くのコードクローンが検出できる可能性が示唆された [39]．

Jiang らは，DECKARD という抽象構文木に基づくコードクローン検出法を提案している [14]．この手法は，事前に定義された規則に基づきソースコードを抽象構文木に変換し，コード片の特徴ベクトルを生成する．そして特徴ベクトルをクラスタリングすることにより，コードクローンを検出する．しかし，DECKARD は他のコードクローン検出手法と比較し，再現率が低く，スケーラビリティが低いことが指摘されている [34, 38]．またコード片の類似性判定法の最新手法である DeepSim と比較して，DeepSim の方が再現率と適合率ともに高いことが明らかになっている [40]．

White らは，RtvNN という機械学習の一種である再帰型ニューラルネットワーク [32] を用いるコード片の類似性の判定法を提案した [36]．RtvNN はトークン列と抽象構文木の情報から再帰型ニューラルネットワークを用いてベクトル表現を求め，コード片の類似性の判定にはベクトル表現のユークリッド距離を用いる．また，Wei らは CDLH という機械学習の一種である LSTM(Long Short-Term Memory) を用いたコード片の類似性の判定法を提案している [35]．CDLH は字句と構文の情報から抽象構文木ベースの LSTM を用いてハッシュコードを求め，高速に類似性を学習する．コード片の類似性の判定にはハッシュコードのハミング距離を用いる．したがって RtvNN と CDLH は，コード片の類似性を判定するために機械学習モデルを使用しない点で提案手法とは異なる．さらに，この 2 つの手法と DeepSim を BigCloneBench に対象として精度評価を比較した結果，DeepSim の方が再現率と適合率ともに高いことが明らかになっている [40]．再帰型ニューラルネットワークや LSTM といった手法は自然言語処理の分野でよく用いられる手法である．自然言語処理を基に機械学習し

[9]https://github.com/k-yokoi/CCVolti

たベクトル表現より，DeepSim のようにソースコード解析を行い制御フローグラフやデータフローグラフを基に機械学習したベクトル表現の方が，より高い精度が得られことを示している [40]．再帰型ニューラルネットワークや LSTM といった手法は自然言語処理の分野でよく用いられる．自然言語処理を基に機械学習したベクトル表現より，DeepSim のようにソースコード解析を行い制御フローグラフやデータフローグラフを基に機械学習したベクトル表現の方が，より高い精度が得られることを DeepSim の著者らは示した [40]．本研究では，簡易的なソースコード解析と，自然言語処理でも用いられるような情報検索技術に基づくベクトル化手法を組み合わせることで，さらに高精度なコード片の類似性判定法を実現した．

6　まとめと今後の課題

本研究では，情報検索技術に基づくベクトル表現と深層学習モデルを組み合わせることで，コード片の処理内容の意味的な類似性を判定する手法を新たに提案した．情報検索技術に基づくベクトル化手法によりコード片をベクトル表現に変換し，2 つのコード片対が類似しているか否かの二値分類を行う深層学習モデルに入力する．

評価実験では，5 種のベクトル表現の中での評価と，最新手法である DeepSim との比較を行った．評価方法としては，競技プログラミング Google Code Jam と，大規模コードクローンベンチマーク BigCloneBench をデータセットとして用いて，再現率，適合率，F 値の観点から評価した．また Google Code Jam のデータセットを対象に，ベクトル化，学習，推定に所要する時間を比較した．

評価実験の結果，情報検索技術に基づくベクトル表現として LSI を用いる手法が，最も高速かつ高精度であることが分かった．また最新手法である DeepSim と比較して，再現率，適合率，F 値ともに高く，さらに学習時間に関しても約 350 倍高速であり，提案手法の有用性が高いことが確認できた．

今後の課題として，以下の 2 点が挙げられる．

- 学習したデータセット以外でも高い精度が得られる，より汎化性能が高いモデルの作成
- 学習済みでないデータセット内からコード片の類似性を自分で学習する教師なし学習モデルの作成

謝辞

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 教授には，研究において大変貴重な御指導及び御助言を賜りました．井上 教授の御指導及び御助言のおかげで本論文を完成させることができました．井上 教授に心より深く感謝いたします．

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 准教授には，研究の各段階において多くの御助言を賜りました．多くの御指導及び御助言を頂いた松下 准教授に心より深く感謝いたします．

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 神田 哲也 助教には，研究において多くの御助言を賜り，また息抜き用の差し入れを頂きました．多くの御助言及び差し入れを頂いた神田 助教に心より深く感謝いたします．

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 / 情報システム学専攻 吉田 則裕 准教授には，研究に関する直接の御指導及び御助言を賜りました．常に適切な御指導及び御助言を頂いたことにより，本論文を完成することができました．吉田 准教授に心より深く感謝いたします．

奈良先端科学技術大学院大学先端科学技術研究科情報科学領域 崔 恩瀞 助教には，研究に関する直接の御指導及び御助言を賜りました．常に様々な御指導及び御助言を頂いたことにより，本論文を完成することができました．崔 助教に心より深く感謝いたします．

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 春名 修介 特任教授には，研究において御指導及び御助言を賜りました．春名 特任教授に心より深く感謝いたします．

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 藤原 裕士 氏には，研究に関する相談に乗っていただき，特に深層学習に関する貴重な知見を共有していただきました．藤原 氏に心より深く感謝いたします．

最後に，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研究室の皆様には，多くの御指導及び御助言を頂き私を支えてくださりました．有意義な研究室生活を送りながら本論文を完成させることができたことは井上研究室の皆様のおかげであると，心より深く感謝いたします．

参考文献

[1] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern information retrieval: The concepts and technology behind search*. Addison-Wesley, 2011.

[2] Pierre Baldi and Yves Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, Vol. 5, pp. 402–418, 1993.

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

[4] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, Vol. 3, No. Jan, pp. 993–1022, 2003.

[5] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, Vol. 5, pp. 135–146, 2017.

[6] Eric Enslen, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *Proc. of MSR*, pp. 71–80, 2009.

[7] 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.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, Vol. 15, pp. 315–323, 2011.

[9] Zellig S. Harris. Distributional structure. *WORDs*, Vol. 10, No. 2-3, pp. 146–162, 1954.

[10] 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.

[11] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, Vol. 313, No. 5786, pp. 504–507, 2006.

[12] Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *Proc. of ICSME*, pp. 117–125, 2005.

[13] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, Vol. 2, No. 5, pp. 359 – 366, 1989.

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

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

[16] Iman Keivanloo, Juergen Rilling, and Ying Zou. Spotting working code examples. In *Proc of ICSE*, pp. 664–675, 2014.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[18] 北研二, 津田和彦, 獅々堀正幹. 情報検索アルゴリズム. 共立出版, 2002.

[19] Matt J. Kusner, Yu Sun, Nicholas I. Kolkin, and Kilian Q. Weinberger. From word embeddings to document distances. In *Proc. of ICML*, Vol. 37, pp. 957–966, 2015.

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

[21] Zhenmin Li, Shan Lu, Suvda Myagmar, and Yuanyuan Zhou. Cp-miner: finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on Software Engineering*, Vol. 32, No. 3, pp. 176–192, 2006.

[22] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proc. of ICLR*, 2013.

[23] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. *Proc. of NIPS*, 2013.

[24] 岡谷貴之. 深層学習. 機械学習プロフェッショナルシリーズ. 講談社, 2015.

[25] Radim Řehůřek and Petr Sojka. Software Framework for Topic Modelling with Large Corpora. In *Proc. of LREC* , pp. 45–50, May 2010.

[26] Chanchal K. Roy and James R. Cordy. An empirical study of function clones in open source software. In *Proc. of WCRE*, pp. 81–90, 2008.

[27] 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*, pp. 172–181, 2008.

[28] 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.

[29] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pp. 696–699. 1988.

[30] 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*, pp. 354–365, 2018.

[31] 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*, pp. 1157–1168, 2016.

[32] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y. Ng, and Christopher D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *Proc. of the 28th International Conference on International Conference on Machine Learning*, pp. 129–136, 2011.

[33] 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*, pp. 476–480, 2014.

[34] Jeffrey Svajlenko and Chanchal K. Roy. Evaluating clone detection tools with bigclonebench. In *Proc. of ICSME*, pp. 131–140, 2015.

[35] 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*, pp. 3034–3040, 2017.

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

[37] 横井一輝, 崔恩瀞, 吉田則裕, 井上克郎. コード片のベクトル表現に基づく大規模コードクローン集合の特徴調査. 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. 192–199, 2018.

[38] 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.

[39] Kazuki Yokoi, Eunjong Choi, Norihiro Yoshida, and Katsuro Inoue. Investigating vector-based detection of code clones using bigclonebench. In *Proc. of APSEC*, pp. 699–700, 2018.

[40] Gang Zhao and Jeff Huang. Deepsim: Deep learning code functional similarity. In *Proc. of ESEC/FSE*, pp. 141–151, 2018.