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

横井 一輝[1,a)]　崔 恩瀞[2]　吉田 則裕[3]　井上 克郎[1]

概要：コードクローンを自動検出する手法の 1 つとして，コード片をベクトル空間に写像し，ベクトル空間上で距離が近いコード片をコードクローンとして検出する手法がある．既存研究において，TF-IDF を用いたコード片のベクトル表現に対し，コサイン類似度を用いてクローン検出を行う手法が提案されている．TF-IDF 以外に次元圧縮を行う LSI や機械学習を用いてベクトル生成を行う Doc2Vec などコード片をベクトル化する手法が多く存在する．また距離尺度としてコサイン類似度以外にも様々な尺度が存在するが，これらの手法がコードクローンの検出にどのように影響を与えるのかは明らかになっていない．そこで本研究では，コードクローン検出の再現率や検出時間を用いて異なるコード片のベクトル表現手法と距離尺度がコードクローン検出に与える影響について調査した．本調査では，コードクローン検出ツールの評価に使用される大規模ベンチマーク BigCloneBench に含まれるコードクローン集合を調査対象とする．本調査の結果，それぞれのベクトル空間上でコードクローンが持つ特徴が明らかになり，この調査結果に基づいて新たなコードクローン検出手法の考察を行う．

キーワード：コードクローン，ソフトウェア保守，情報検索技術

1. まえがき

ソフトウェアの保守性を低下させる要因の 1 つとして，コードクローンが指摘されている [2]．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のことであり，一般的に，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と言われている．この問題を解決するためにコードクローンをソースコードから見つけて適切に管理する必要があるが，ソースコードの規模が大きくなるとソースコード中に含まれるコードクローンも膨大な量となり，手作業でコードクローンを見つけることは困難となる．この問題を解決するために，コードクローンを自動的に検出するための様々なコードクローン検出手法が提案されている [8], [16]．

コードクローン検出手法の 1 つとして，ベクトル表現に基づくクローン検出手法がある．この手法ではコード片をベクトル空間に写像し，ベクトル空間上で距離が近いコード片をコードクローンとして検出する．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．またクラスタリング手法を用いることによりコードクローンを高速度で検出することができる．ベクトル表現に基づくクローン検出の 1 つとして，山中らの関数クローン検出法 [25] がある．関数クローン検出法は TF-IDF[1] を利用することで，意味的に処理が類似した関数単位のコードクローンを検出する．そしてコサイン類似度を用いてベクトル間の類似度を計算することによって，関数クローンの検出を行う．

しかし，関数クローン検出法は TF-IDF を用いることにより，ベクトルが高次元となる点や，多義語や同義語といった単語間の意味を考慮しない問題点が存在する．ベクトルが高次元になる場合，類似度計算に時間を要したりメモリ使用量が増大したりと，計算量が大きくなる問題につながる．さらに，単語間の意味を考慮しない場合，意味的に処理が類似したコードクローンの検出漏れを起こす可能性がある．

関数クローン検出法で使用されている TF-IDF 以外にもコード片をベクトル表現に変換する手法は多くあり，また，ベクトル表現の距離尺度としてコサイン類似度以外にも様々な距離尺度がある．しかし，これらがコードクローン

[1] 大阪大学
Osaka University
[2] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[3] 名古屋大学
Nagoya University
[a)] k-yokoi@ist.osaka-u.ac.jp

© 2018 Information Processing Society of Japan

の検出にどのように影響を与えるのかは明らかになっていない．そこで本研究では，クローン検出で用いられる様々なベクトル表現や距離尺度の特徴を調査し，クローン検出により適した手法の考察をするために，1) ベクトル表現によって再現率は変化するか．2) コードクローンのタイプごとに，各ベクトル表現の類似度はどう異なるか．3) true positive および false positive の類似度は，ベクトル表現によりどのように変化するか．4) ベクトル表現や距離尺度の選択は，検出速度に影響を与えるか．という 4 つのリサーチクエスチョンを設定した．リサーチクエスチョンの実施はコードクローン検出の再現率や検出時間を用いて異なるコード片のベクトル表現手法と距離尺度がコードクローン検出に与える影響について調査した．本調査の対象として，大規模コードクローン集合 BigCloneBench[23] を用いた．BigCloneBench とは，実在の約 2 万 5 千の Java プロジェクトから関数単位のコードクローンを収集したコードクローン集合であり，クローン検出手法の再現率評価に利用される．

調査の結果，ベクトル表現によって再現率は大きく異なり，次元圧縮を行うことで再現率が上昇することが分かった．しかし，機械学習を行う手法は必ずしも高い再現率が得られるわけではなかった．また距離尺度に関しては，Word Mover's Distance[10] という距離尺度を用いることで，true positive と false positive をよく判別できた．そして計算時間に関しては，ベクトル表現ごとに生成時間は異なり，機械学習を行う手法は低速であった．さらに Word Mover's Distance の類似度計算速度がコサイン類似度と比較して非常に低速であった．この調査結果に基づき，計算コストの異なる複数のベクトル表現と距離尺度を組み合わせたクローン検出手法の提案を行った．

以降，2 章では，コードクローンとその検出について述べる．3 章では，大規模コードクローン集合の特徴調査について述べる．4 章では，調査結果から考えられるコードクローン検出手法について述べる．最後に，5 章でまとめと今後の課題について述べる．

2. コードクローンとその検出

本章では，本研究の背景としてコードクローンおよびコードクローン検出の 1 つであるベクトル表現に基づいたクローン検出手法について述べる．

2.1 コードクローン

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のことであり，一般的に，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と言われている．Roy らはコードクローン間の違いの度合いに基づき以下の 4 つのタイプに分類している [18].

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.

タイプ 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.

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.

タイプ 3 とタイプ 4 のコードクローンを分離することは難しいため，Svajlenko らは構文的な類似度に基づいてタイプ 3 とタイプ 4 のコードクローンをさらに以下の 4 つに分類している [21]．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．

VST3 (Very-Strongly Type-3) 構文的類似度 90%以上 100%未満.

ST3 (Strongly Type-3) 構文的類似度 70%以上 90%未満.

MT3 (Moderately Type-3) 構文的類似度 50%以上 70%未満.

WT3/T4 (Weakly Type-3/Type-4) 構文的類似度 0%以上 50%未満.

2.2 ベクトル表現に基づくコードクローン検出手法

コードクローンをソースコードから見つけて適切に管理する必要があるが，ソースコードの規模が大きくなるとソースコード中に含まれるコードクローンも膨大な量となり，手作業でソースコードからコードクローンを見つけることは困難となる．この問題を解決するために，コードクローンを自動的に検出することを目的とした様々な検出手法が提案されている [8], [16].

コードクローン検出手法の 1 つとして，ベクトル表現に基づくクローン検出手法がある．この手法ではコード片をベクトル空間に写像し，ベクトル空間上で距離が近いコード片をコードクローンとして検出する．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．またクラスタリング手法を用いることによりコードクローンを高速度で検出することができる．ベクトル表現に基づくクローン検出の 1 つとして，山中らの関数クローン検出法 [25] がある．関数クローン検出法は TF-IDF (Term Frequency and Inverse Document Frequency)[1] を利用することにより，意味的に処理が類似した関数単位のコードクローンを検出する手法である．情報検索技術とは，大量のデータ群から目的に合致したものを取り出すことであり，自然言語で書かれた文書の処理などに利用される [1]．そしてベクトル間の類似

© 2018 Information Processing Society of Japan

度を計算することによって，関数クローンの検出を行う．

しかし，関数クローン検出法は TF-IDF を用いることで，ベクトルが高次元になる点や，多義語や同義語といった単語間の意味を考慮しない問題点が存在する．ベクトルが高次元になる場合，類似度計算に時間を要したりメモリ使用量が増大したりと，計算量が大きくなる問題につながる．また，単語間の意味を考慮できない場合，意味的に処理が類似したコードクローンの検出漏れを起こす可能性がある．

3. 大規模コードクローン集合の特徴調査

本章では，本研究で行った大規模コードクローン集合の特徴調査について述べる．最初に調査の目的とリサーチクエスチョンについて述べ，次に調査対象のソースコード，ベクトル，距離尺度について説明する．そしてリサーチクエスチョンの調査方法と回答を述べ，最後に妥当性の脅威を考察する．

3.1 目的とリサーチクエスチョン

関数クローン検出法で使用されている TF-IDF 以外にもコード片をベクトル表現に変換する手法は多くある．また，ベクトル表現の類似性を示す距離尺度として，コサイン類似度以外にも様々な距離尺度が存在するが，これらをクローン検出に適用する観点から横断的に評価した研究はない．そこで本研究では，コードクローン検出において様々なベクトル表現や距離尺度の特徴を調査し，その結果に基づいて新たなコードクローン検出手法の考察を行う．本研究ではコードクローン検出の再現率や検出時間を用いて異なるコード片のベクトル表現手法と距離尺度がコードクローン検出に与える影響について調査する．

本研究では調査するにあたって以下の 4 つのリサーチクエスチョンを立てた．

RQ1 ベクトル表現によってコードクローン検出の再現率は変化するか．

RQ2 コードクローンのタイプにより，各ベクトル表現間の類似度がどのように変化するか．

RQ3 true positive および false positive の類似度は，ベクトル表現によりどのように変化するか．

RQ4 ベクトル表現や距離尺度の選択は，検出速度に影響を与えるか．

タイプ 1 や 2 を高い再現率で検出できる既存のコードクローン検出手法は多いが，タイプ 3 や 4 を高い再現率で検出できる手法はまだない [21]．そこで，あらゆるタイプで高い再現率で検出できるベクトル表現を調査するために RQ1 を設定した．また，BigCloneBench は人間の手作業によってコードクローンのタイプを分類されている．これらを機械的に分類できるベクトル表現や距離尺度を調査するために RQ2 を設定した．BigCloneBench に含まれている true positive と false positive を分類できるベクトル表現や距離尺度を調査するために RQ3 を設定した．最後に，近年コードクローン検出は大規模なプロジェクトに対しても行われており，スケーラビリティの観点がクローン検出手法の評価において重要であるため，RQ4 を設定した．

3.2 対象ソースコード

3.1 節で設定した 4 つのリサーチクエスチョンに答えるため，大規模コードクローン集合 BigCloneBench を調査した [23]．BigCloneBench とは，実在の約 2 万 5 千の Java プロジェクトから関数単位のコードクローンを収集したコードクローン集合であり，クローン検出手法の再現率評価に利用される．BigCloneBench は実プロジェクトから手作業でコードクローンを収集した大規模なベンチマークであり，近年のクローン検出の評価手段の 1 つとしてよく用いられる [20], [21]．

本研究では調査を行う前段階として，ソースコードに対して次の前処理を行う．最初にソースコードを字句解析し，トークン単位での分割とコメント除去を行う．その後，識別子名はキャメルケースやスネークケースによって分割し，さらに小文字変換の正規化を行う [6]．最後に，予約語と識別子以外のトークンを除去し，関数単位でベクトル表現に変換する．なお，関数の抽出は ANTLR v4[*1] を用いて行う．

3.3 対象ベクトルと距離尺度

本研究ではコード片のベクトル表現として，関数単位のコード片を 1 つの文書と捉えベクトル化する手法を考える．この文書をベクトル化する手法は，情報検索技術の分野で用いられる手法である．本研究で調査に用いたベクトル表現は以下の通りである．

BoW (Bag-of-Words)[7] 出現する単語の多重集合

TF-IDF 単語に出現頻度に基づき重要度を付与

LSI (Latent Semantic Indexing)[1] 主成分分析により次元圧縮し潜在的意味解析

LDA (Latent Dirichlet Allocation)[4] LSI をベイズ化

Word2Vec [14], [15] 教師なし機械学習による単語ベクトル

FastText [5] Word2Vec の派生

Doc2Vec [11] 教師なし機械学習による文書ベクトル

BoW とは，文書中に出現する単語の多重集合として表すことで文書をベクトル化する．BoW は複数の文書を比較したり類似性を測定したりするために用いられ，文書をベクトル化する単純かつ伝統的なベクトル表現である．そして BoW に対して単語の重要度を付与したベクトル表現とし

[*1] http://www.antlr.org/

© 2018 Information Processing Society of Japan

て TF-IDF がある．TF-IDF は単語の逆文書頻度に基づいて重要度を計算する．これらのベクトル表現は全文書中の単語の語彙数がベクトルの次元となるため，非常に高次元なベクトル表現となりやすい．そこで，潜在的意味解析を行うことでベクトルを次元圧縮し，より低次元で文書を表現するベクトル化手法が存在する．潜在的意味解析として主成分分析を行うベクトル表現として LSI がある．さらに LSI をベイズ化したベクトル表現として LDA がある．また，ニューラルネットワークを用いて単語の意味を表現可能なベクトルを生成し，文書中に含まれる単語ベクトルの平均を文書ベクトルとする手法が存在する．ニューラルネットワークを用いた単語ベクトルの 1 つとして，Word2Vec と FastText がある．本論文では，Word2Vec の平均ベクトルを WV-avg，FastText の平均ベクトルを FT-avg と表記する．さらに Word2Vec を拡張し，任意の長さの文書をニューラルネットワークによりベクトル化する手法として Doc2Vec が存在する．

ベクトルの類似度を測定する距離尺度はコサイン類似度と WMD (Word Mover's Distance)[10] の 2 つの尺度を用いた．コサイン類似度とはベクトル同士の成す角の近さに基づき計算する距離尺度で，ベクトル空間モデルにおいて一般的に用いられる類似度計算手法である．WMD は単語ベクトルを用いて文書間の距離を計算する手法であり，2 つの分布間の距離尺度として画像検索などで用いられる Earth Mover's Distance[19] を文書間距離に適用した手法である．

なお，本研究において各種ベクトルに設定するパラメータは，実装に用いた gensim[17][*2]のデフォルト値を基に決定した．

3.4 リサーチクエスチョンの調査方法

大規模コードクローン集合の特徴調査では，3.1 節で立てた 4 つのリサーチクエスチョンに答えるために，以下の調査を行う．

RQ1 に答えるためにベクトル表現ごとの再現率を比較することで，コードクローンをより網羅的に検出できるベクトル表現の調査を行う．本調査では，BoW，TF-IDF，LSI，LDA，Doc2Vec，Word2Vec の平均（WV-avg），FastText の平均（FT-avg）の各ベクトル表現における検出された各タイプのコードクローンの再現率を評価した．検出対象として BigCloneBench を使用し，再現率の評価には BigCloneBench を用いた再現率測定フレームワークである，BigCloneEval[22][*3]を使用した．本調査では，コサイン類似度 0.9 を閾値として，閾値以上をコードクローンとして検出した．閾値を 0.9 とした理由は既存研究において検出精度の観点から優れた値であったからである [25]．なお，本調査では WMD を用いた検出は実行に 3 か月程度かかる見込みとなったため行っていない．

RQ2 に答えるためにベクトル表現や距離尺度による，BigCloneBench のコードクローンのタイプごとの各ベクトル表現の類似度の分布を調査した．本調査では，コサイン類似度と WMD の 2 つの距離尺度を用いた．コサイン類似度の分布調査では，BoW，Doc2Vec，WV-avg，FT-avg の 4 つのベクトル表現を対象に行った．また WMD の分布調査では，Word2Vec と FastText の 2 つのベクトル表現を対象に行った．WMD はコード片中に含まれる単語ベクトルの分布を基に距離計算を行うため，コサイン類似度のように文書ベクトルは用いない．なお，TF-IDF，LSI，LDA は，BigCloneBench が大規模コードクローン集合であることから非常に高次元なベクトル空間となり，スケーラビリティの問題が発生したため行っていない．

RQ3 に答えるために，BigCloneBench に含まれる true positive と，false positive の類似度の分布を調査した．調査方法は RQ2 と同様である．

RQ4 に答えるために，ベクトル計算速度が高速なベクトル表現を調査した．BigCloneBench からランダムにソースコードを選択し，約 1MLOC のデータセットを作成した．このデータセットの作成方法は，クローン検出のスケーラビリティ評価で一般的に用いられる手法と同様である．作成したデータセットでは，関数の数が 25440，語彙数が 24319 である．作成したデータセットをもとに，ベクトルの生成時間（機械学習を行う手法では学習時間も含む）と，ある 1 つの関数とその他全関数の類似度計算時間を測定した．

3.5 リサーチクエスチョンに対する回答

3.5.1 RQ1: ベクトル表現によって再現率は変化するか

各ベクトル表現による再現率に関する調査の結果を**表 1** に示す．全てのベクトル表現においてタイプ 1 のコードクローンに関しては，一様に再現率 0.99 で検出した．関数の抽出に使用した ANTLR v4 による構文解析の失敗により，再現率 1.0 にはならなかったという点に注意されたい．タイプ 2 に関しては，LSI，Doc2Vec，WV-avg，FT-avg が再現率 0.9 以上となった．この結果より，次元圧縮や機械学習を行うことで，識別子名の変更などがあるコードクローンでも検出可能となることが分かった．タイプ 3，4 に関してはベクトル表現ごとに比較を行う．TF-IDF に関しては，BoW と比較して再現率が低くなった．単語の重要度を考慮する TF-IDF は，予約語より識別子の重要度が高くなる傾向にあり，コードクローンは識別子名の変更が行われる場合が多いため，BoW よりコサイン類似度が低くなったと考えられる．また，LSI や LDA といった次元圧縮を行った手法は BoW や TF-IDF より再現率が高くなった．LSI と LDA で比較を行った場合，タイプ 2 において

[*2] https://radimrehurek.com/gensim/
[*3] https://jeffsvajlenko.weebly.com/bigcloneeval.html

© 2018 Information Processing Society of Japan

表 1　ベクトル表現ごとの再現率

	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

表 2　ベクトル表現ごとの計算時間

	BoW	TF-IDF	LSI	LDA	Doc2Vec	WV-avg	FT-avg	Word2Vec	FastText
距離尺度	コサイン類似度							WMD	
生成時間	5.1 s	10.0 s	9.7 s	60.3 s	44.7 s	42.7 s	196.1 s	29.5 s	187.7 s
類似計算時間	5.5 s	5.1 s	1.1 s	1.1 s	1.6 s	1.1 s	1.1 s	497.5 s	538.1 s

は LSI の再現率が高いが，MT3 においては LDA の再現率が高い．機械学習により文書ベクトルを生成する Doc2Vec は BoW や TF-IDF と同等の再現率で，LSI や LDA より全てのタイプにおいて再現率が低くなった．Doc2Vec はタイプ 2 の検出においては高い再現率が得られたが，VST3，ST3 などのタイプ 3 のコードクローンは多く検出できないことが分かった．機械学習により生成した単語ベクトルの平均を求める WV-avg，FT-avg では再現率が最も高い結果となったが，単純に検出数が多いから再現率が高いだけの可能性が否定できない．

今後の課題として，コードクローン検出の再現率だけでなく，検出の正確性を示す適合率の調査も行う必要がある．

次元圧縮を行うことで BoW より再現率は上昇し，特にタイプ 3 のコードクローンを高い再現率で検出できる．機械学習を用いて生成した文書ベクトルは BoW と同等またはそれ以下の再現率となる．機械学習を用いて生成した単語ベクトルの平均を用いた場合は最も再現率が高くなる．

3.5.2　RQ2: コードクローンのタイプにより，各ベクトル表現間の類似度がどのように変化するか

コードクローンのタイプごとの各ベクトル表現の類似度に関する調査結果を箱ひげ図で**図 1** と**図 2** に示す．図 1 からは，コサイン類似度はどのベクトル表現であっても，MT3 と WT3/T4 が分離していないことが分かった（図 1 の赤丸で囲んである部分）．それに対し，図 2 では WMD ではコサイン類似度に比べてタイプごとに分離していることが分かった．

コサイン類似度はどのベクトル表現においてもコードクローンのタイプごとに分離しないが，WMD ではコサイン類似度に比べてよく分離している．

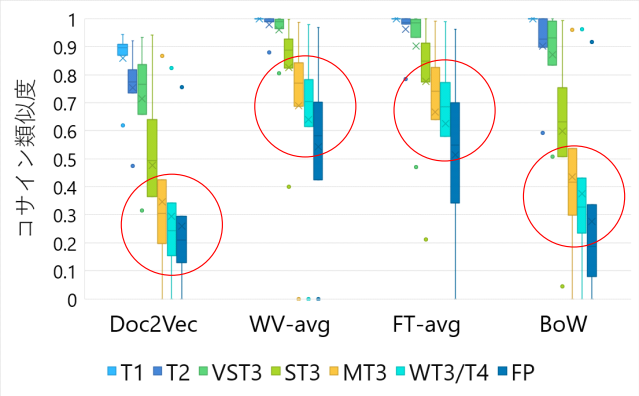

図 1　コサイン類似度の分布

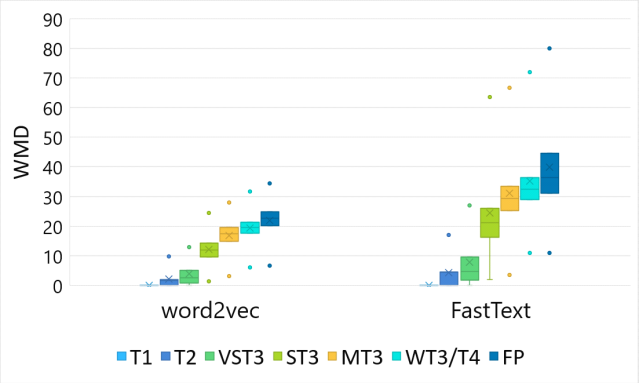

図 2　WMD の分布

3.5.3　RQ3: true positive および false positive の類似度は，ベクトル表現によりどのように変化するか

true positive と false positive において，各ベクトル表現の類似度に関する調査結果を箱ひげ図で図 1 と図 2 に示す．これらの図において FP は false positive であり，BigCloneBench に含まれている誤検出されたコードクローンを示している．この結果から，コサイン類似度はどのベクトル表現であっても，WT3/T4 と false positive が分離しておらず，true positive と false positive が判別できていないことが分かった（図 1 の赤丸で囲んである部分）．その反面，WMD ではコサイン類似度に比べて true positive

© 2018 Information Processing Society of Japan

と false positive が分離していることが分かった．

コサイン類似度はどのベクトル表現においても true positive と false positive が分離しないが，WMD ではコサイン類似度に比べてよく分離している．

3.5.4 RQ4: ベクトル表現や距離尺度の選択は，検出速度に影響を与えるか

各ベクトル表現のベクトル生成時間と類似度計算時間に関する調査結果を**表 2** に示す．この表から，ベクトルの生成時間は BoW が最速であり，TF-IDF や LSI も十分に高速であることが分かった．それに対し，機械学習を行う手法は低速であり，特に FastText は非常に低速であることが分かった．Word2Vec は機械学習を行う中では高速な部類であることも確認できた．

次元圧縮を行う手法では，LSI が高速であるのに対し，LDA は低速であった．機械学習を行う手法では WV-avg が最も速く，FT-avg は WV-avg と比較して再現率が同等であるにも関わらず（表 1）低速であることが分かった．

類似度計算時間に関しては，次元圧縮を行う手法は高速になるという結果が得られた．その反面，WMD を用いた手法はコサイン類似度に比べて非常に低速であることが分かった．特に本調査での類似計算時間の測定方法は，ある 1 関数と他の全関数との類似計算，つまり $O(n)$ の計算時間を測定している．しかし，実際のクローン検出は全関数と全関数との類似計算，つまり $O(n^2)$ の計算を行う必要性があり，何かしらの高速化の手段を用いない限り，WMD を用いたクローン検出は実用的でないことが分かった．

ベクトルの生成時間は BoW が最速であり，TF-IDF や LSI も高速である．その反面機械学習を行う手法は総じて低速である．類似度計算時間は次元圧縮を行ったベクトルのコサイン類似度計算は高速である．その反面 WMD の計算は非常に低速である．

3.6 妥当性の脅威

本調査の妥当性の脅威としては，まず本調査の結果が，BigCloneBench に依存した結果であることが挙げられる．BigCloneBench は実在するプロジェクトから収集したコードクローンの大規模集合であるため，ある程度一般性のある結果が得られたと考えている．しかし，単一のプロジェクトなど小規模な検出対象に対しては異なる結果が得られる可能性がある．また，BigCloneBench は Java プロジェクトのコードクローン集合であることから，Java に依存した調査結果である可能性がある．プログラミング言語によってソースコードの記述方法が異なるため他言語に本調査を適用して結果を調査することは今後の課題である．さらに，本調査ではベクトル計算のパラメータ調整を行わずデフォルト値を採用したが，適切なパラメータ調整を行うことで結果が変化する可能性はある．しかし，ベクトル表現の素直な特性を調査するため，あえてパラメータ調整を行わずデフォルト値を採用した．

4. 調査結果から考えられるコードクローン検出手法

本章では 3.5 節で説明したリサーチクエスチョンに対する回答に基づいて新たなコードクローン検出手法の提案を行う．RQ2，RQ3 の答えより WMD がコードクローンと false positive を判別することができていたが，RQ4 の答えより WMD は類似度計算速度が非常に遅い手法であることが分かった．したがって WMD をそのままクローン検出に適用することは現実的でない．よって，WMD の計算を行う前に，計算コストの低い処理により計算対象を削減し，コードクローンを検出する手法を提案する．提案手法は以下の 6 つのステップで実行する．提案手法の概要を**図 3** に示す．

STEP 1 BoW で軽量ベクトル作成．
STEP 2 LSH クラスタリングを用いてクローン候補作成．
STEP 3 Word2Vec で重量ベクトル作成．
STEP 4 クローン候補に対しフィルタリング．
STEP 5 コサイン類似度計算を行いクローン検出．
STEP 6 STEP 5 で検出されなかったクローン候補に対して WMD 計算を行いクローン検出．

軽量ベクトルとは，計算コストが低いベクトル表現を表している．RQ4 の答えより BoW や TF-IDF，LSI が軽量ベクトルとして適用可能である．その中でも RQ1 に対する答えでも十分に高い再現率が得られ，また単純かつ伝統的な手法でもある BoW を軽量ベクトルとして提案手法では用いる．BoW が LSI より再現率は低いが，提案手法では重量ベクトルとしてもう 1 つベクトル表現を作成するため，BoW の方が適していると考えた．

ベクトル表現を用いたクローン検出法の特徴であるクラスタリングによる検出の高速化を提案手法でも行う．提案手法ではクラスタリングを行うことでコードクローンになり得る候補，クローン候補を作成する．クラスタリングとして様々な手法が考えられるが，既存研究では LSH がクローン検出に適用されている [9], [25]．LSH は高次元ベクトル空間において類似ベクトルを高速に探索する手法であり，BoW が高次元なベクトルになるという問題にも対応できる [24]．

重量ベクトルとは，計算コストが大きいベクトル表現を表している．表 2 より Word2Vec が重量ベクトルの中でも高速であるため提案手法では用いる．Word2Vec を用いた場合，文書中の単語ベクトルの平均である WV-avg を用いてクローン検出を行うことも可能であり，さらに WMD によるクローン検出も行うことも可能である．

© 2018 Information Processing Society of Japan

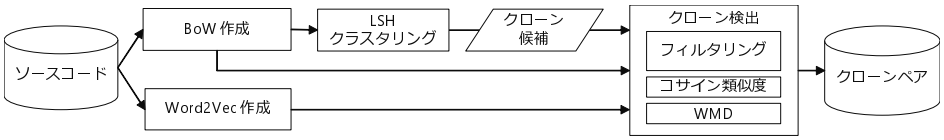

図 3 提案手法の概要

クローン検出の処理を行う際は誤検出を防ぐためにクローン候補のフィルタリングを行う．フィルタリングとしては，コード片の行数比や行数差などを用いる．フィルタリング処理後，コサイン類似度の計算コストの低い類似度計算を行いクローン検出を行う．そして，コサイン類似度で検出できなかったクローン候補に対し，WMD を用いてクローン検出を行う．

つまり，提案手法では WMD の計算を行う前にいくつかの処理を経ることで，WMD を計算する対象を減らし，クローン検出にかかる時間の削減を図る．

5. まとめと今後の課題

本研究では，コードクローン検出の再現率や検出時間を用いて異なるコード片のベクトル表現手法と距離尺度がコードクローン検出に与える影響を調査するために 4 つのリサーチクエスチョンを設定した．また，リサーチクエスチョンに答えるために大規模コードクローン集合 BigCloneBench を対象に，ベクトル表現ごとの再現率調査，ベクトル表現と距離尺度ごとの分布調査，ベクトルの生成時間と類似度計算時間の調査を行った．調査の結果，ベクトル表現によって再現率は大きく異なり，次元圧縮を行うことで再現率が上昇することが分かった．しかし，機械学習を行う手法は必ずしも高い再現率が得られるわけではなかった．また距離尺度に関しては，WMD を用いることでコードクローンと false positive をよく判別できた．そして計算時間に関しては，ベクトル表現ごとに生成時間は異なり，機械学習を行う手法は低速であった．さらに WMD の類似度計算速度がコサイン類似度と比較して非常に低速であった．この調査結果に基づき，計算コストの異なる複数のベクトル表現と距離尺度を組み合わせたクローン検出手法の提案を行った．

今後の課題として，以下の点が挙げられる．

- 提案手法の実装
- 提案手法の適合率，再現率，スケーラビリティの評価
- 前処理の拡張（単語の正規化（‘str’ → ‘string’ など）[3]）
- 他のベクトル表現の調査（NTSG（Neural Tensor Skip-Gram model）[12]， SCDV（Sparse Composite Document Vector）[13]）
- 他の距離尺度の調査（ユークリッド距離，マンハッタン距離，Jaccard 係数，レーベンシュタイン編集距離）
- 他言語への本研究の適用

本論文では手法の提案のみ行い，提案手法の実装および評価実験は行っていない．これら今後の課題として挙げられる．評価実験では，本研究と同様の再現率評価の他にも，検出の正確性を示す適合率やスケーラビリティを評価する必要性がある．

また，提案手法の拡張の 1 つとして，前処理の拡張が挙げられる．識別子は ‘string’ という単語を ‘str’ と表記するように，省略して記述する場合が多くある．そこで，ソースコードの前処理の段階で省略された単語の正規化を行う手法が考えられる [3]．さらに今回調査したベクトル表現の他にも，NTSG[12]，SCDV[13] などの多くのベクトル表現が提案されている．特に SCDV は Word2Vec で用いている単語の分散表現だけでなく，文書のトピックも考慮したベクトル表現となっている [13]．本研究の調査対象をさらに増やし，よりクローン検出に適したベクトル表現の調査を行いたい．また，距離尺度として本研究ではコサイン類似度と WMD を調査対象としたが，他にもユークリッド距離，マンハッタン距離，Jaccard 係数，レーベンシュタイン距離が文章の距離尺度として用いられる．これらも調査対象に加えて評価を行いたい．さらに，本研究の対象のプログラミング言語は Java のみであるため，他の言語への本研究の適用も行いたい．

謝辞 本研究は JSPS 科研費 JP25220003, JP18H04094, JP16K16034 の助成を受けた．

参考文献

[1] Baeza-Yates, R. and Ribeiro-Neto, B.: *Modern information retrieval: The concepts and technology behind search*, Addison-Wesley (2011).
[2] Baxter, I. D., Yahin, A., Moura, L., Anna, M. S. and Bier, L.: Clone detection using abstract syntax trees, *Proc. of ICSM*, pp. 368–377 (1998).
[3] Binkley, D., Lawrie, D. and Uehlinger, C.: Vocabulary normalization improves IR-based concept location, *Proc. of ICSM*, pp. 588–591 (2012).
[4] Blei, D. M., Ng, A. Y. and Jordan, M. I.: Latent dirichlet allocation, *Journal of machine Learning research*, Vol. 3, No. Jan, pp. 993–1022 (2003).
[5] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T.: Enriching Word Vectors with Subword Information, *Transactions of the Association for Computational Linguistics*, Vol. 5, pp. 135–146 (2017).
[6] Enslen, E., Hill, E., Pollock, L. and Vijay-Shanker, K.: Mining source code to automatically split identifiers for software analysis, *Proc. of MSR*, pp. 71–80 (2009).
[7] Harris, Z. S.: Distributional Structure, *WORDs*, Vol. 10, No. 2-3, pp. 146–162 (1954).

© 2018 Information Processing Society of Japan

[8] 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).
[9] Jiang, L., Misherghi, G., Su, Z. and Glondu, S.: DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones, *Proc. of ICSE*, pp. 96–105 (2007).
[10] Kusner, M. J., Sun, Y., Kolkin, N. I. and Weinberger, K. Q.: From Word Embeddings to Document Distances, *Proc. of ICML*, Vol. 37, pp. 957–966 (2015).
[11] Le, Q. and Mikolov, T.: Distributed Representations of Sentences and Documents, *Proc. of ICML*, pp. 1188–1196 (2014).
[12] Liu, P., Qiu, X. and Huang, X.: Learning Context-sensitive Word Embeddings with Neural Tensor Skip-gram Model, *Proc. of IJCAI*, pp. 1284–1290 (2015).
[13] Mekala, D., Gupta, V., Paranjape, B. and Karnick, H.: SCDV: Sparse Composite Document Vectors using soft clustering over distributional representations, *Proc. of EMNLP*, pp. 659–669 (2017).
[14] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient Estimation of Word Representations in Vector Space, *Proc. of ICLR* (2013).
[15] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. and Dean, J.: Distributed Representations of Words and Phrases and their Compositionality, *Proc. of NIPS* (2013).
[16] Rattan, D., Bhatia, R. and Singh, M.: Software clone detection: A systematic review, *Information and Software Technology*, Vol. 55, No. 7, pp. 1165–1199 (2013).
[17] Řehůřek, R. and Sojka, P.: Software Framework for Topic Modelling with Large Corpora, *Proc. of LREC* , pp. 45–50 (2010).
[18] Roy, C. K., Cordy, J. R. and Koschke, R.: 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).
[19] Rubner, Y., Tomasi, C. and Guibas, L. J.: The Earth Mover's Distance As a Metric for Image Retrieval, *Int. J. Comput. Vision*, Vol. 40, No. 2, pp. 99–121 (2000).
[20] Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K. and Lopes, C. V.: SourcererCC: Scaling code clone detection to big-code, *Proc. of ICSE*, pp. 1157–1168 (2016).
[21] Svajlenko, J. and Roy, C. K.: Evaluating clone detection tools with BigCloneBench, *Proc. of ICSME*, pp. 131–140 (2015).
[22] Svajlenko, J. and Roy, C. K.: BigCloneEval: A Clone Detection Tool Evaluation Framework with BigCloneBench, *Proc. of ICSME*, pp. 596–600 (2016).
[23] Svajlenko, J., Islam, J. F., Keivanloo, I., Roy, C. K. and Mia, M. M.: Towards a Big Data Curated Benchmark of Inter-project Code Clones, *Proc. of ICSME*, pp. 476–480 (2014).
[24] 徳井翔梧，吉田則裕，崔　恩瀞，井上克郎：局所性鋭敏型ハッシュを用いたコードクローン検出のためのパラメータ決定手法，電子情報通信学会技術研究報告，Vol. 117, No. 477, pp. 57–62 (2018).
[25] 山中裕樹，崔　恩瀞，吉田則裕，井上克郎：情報検索技術に基づく高速な関数クローン検出，情報処理学会論文誌，Vol. 55, No. 10, pp. 2245–2255 (2014).

© 2018 Information Processing Society of Japan