

ソースコード特有の近傍単語の影響を考慮した Word2Vec を用いた類似コード片推薦手法

内山 武尊^{1,a)} 新美 礼彦^{1,b)}

概要：開発者はソースコードを修正するときに修正箇所と類似したコードを検索し、必要に応じて修正を行う。その時、単語を検索クエリとして使うと多くの検索結果が得られてしまい、その中から修正が必要なソースコードを判断する手間が発生してしまう。そこで本研究では、近傍の単語が類似している単語同士を類似語として紐づけることができる Word2Vec を用いて、ソースコード片を検索クエリとして類似コード片推薦を行うシステムを考える。開発者は修正を行なったコード片を入力として用いることで修正を行なったコード片と類似したコード片が推薦されるため、検索クエリを考える必要なく同様の修正を行うことが出来る。しかし、Word2Vec は自然言語処理技術であるため、ソースコードと自然言語文章の差である予約語や演算子、標準ライブラリ関数の存在や語彙の少なさの影響を考慮する必要がある。本研究では、Word2Vec による類似コード片推薦システムに対して、予約語や演算子、標準ライブラリ関数を取り除く処理と Word2Vec の学習基準である近傍単語の個数を増やすことでソースコード特有の影響を考慮した手法を提案する。評価実験として 2 つのオープンソースソフトウェアのプロジェクトに対して提案手法を用いてどれだけの類似したコード片が推薦可能か調査した。その結果、対象プロジェクトにおいて提案手法の有効性を確認した。

Retrieving Code Fragments Using Word2Vec Considering the Influence of Nearby Words Peculiar to Source Code

1. はじめに

ソフトウェア保守の場において、開発者はソースコードを修正したときに、修正箇所と類似したコードを検索し、必要に応じて修正を行う。一般的には grep コマンドやエディタのキーワード検索機能を用いて、目的の修正箇所を特定する。しかし、検索クエリとしてソースコード中の出現頻度が高い単語（予約語や演算子など）を用いてしまうと多くの検索結果が得られてしまい、そこから修正が必要なコードだと判断するため、検索クエリを考えることは開発者にとって大きな手間となっている [1]。著者らはこれまでに Word2Vec[2] を用いて、検索したい類似コード片に含まれている単語を入力として類似コード片の推薦を行なった [3]。Word2Vec では近傍の単語が類似している単語同

士は同じ文脈で出現するため、類似した単語であるという仮説のもと単語を表すベクトルのコサイン類似度は高くなる。よって、この特徴を応用し変数名などが僅かに違う類似コード片でも推薦可能であると仮説を立て日本語 IME かんな [4] に対して実験を行った。その結果、入力として与える単語によって推薦される正解データ数に大きな差が生まれた。つまり、入力として与える単語を考える手間が発生しまい、検索クエリを考える手間を軽減出来なかった。そこで、本研究では修正箇所のコード片を入力として、類似したコード片を推薦することで検索クエリを考える手間を軽減することを目指す。しかし、Word2Vec は特定の単語をベクトル化するときに近傍の単語を基に学習を行う。そのため、予約語や演算子、標準ライブラリ関数などが多く出現するソースコードでは、近傍単語として頻出単語が多く含まれているため上手く学習出来ない可能性がある。そこで、予備実験として予約語や演算子、標準ライブラリ関数などの頻出単語が Word2Vec の学習に与える影響について調査する。調査方法として予約語と演算子、標準

¹ 公立はこだて未来大学
116-2, Kameda Nakanochi, Hakodate-shi, Hokkaido, 041-0803, Japan

^{a)} g2116006@fun.ac.jp

^{b)} niimi@fun.ac.jp

ライブラリ関数をそれぞれ取り除いたソースコードを用いる方法と、単語を取り除かずに Word2Vec で学習する際の近傍単語数を増やすことで出現頻度が少ない単語を学習に含む方法で類似コード片推薦を行い、それぞれの方法でどれだけの類似コード片が推薦出来るか調査する。また、予備実験の結果を踏まえ、近傍単語の与える影響を考慮したモデルを別のプロジェクトへ適用実験を行った。実験対象として、日本語 IME かんなど日本語入力システム Mozc に対してコードクローン検出ツールで見つかったコードクローンを正解データとして実験を行った。

2. 関連研究

Marcus ら [5] は潜在的意味インデキシング (Latent Semantic Indexing) を使ったコード片検索方法を提案している。この手法では、ソースコードを自然文章の文字列として扱うことで構文解析を行わずに分析が出来るため、ソースコードの言語に依存しない利点がある。しかし、単語を文字列として扱っているため、変数名の違いによる影響を受けやすい特徴がある。

Yoshida ら [1] は自然文章中の語の類似性を計測するための Jensen-Shannon divergence 手法をコード片の単語に用いて同義語を特定している。その同義語と構文の情報を用いることで単語が異なっても類似コード片検索を行うことができる。この手法では、Jensen-Shannon divergence 手法を用いて同義語を特定することで一部が少し異なるような類似コード片でも検索が行える。しかし、この手法では、検索対象のコード片が類似しているかを閾値によって判別しているが、プロジェクトによって最適な閾値が異なるためプロジェクトごとに最適な閾値の調査が必要である。本研究では、頻出単語を取り除く処理や Word2Vec で学習する近傍の単語を増やすことで複数のプロジェクトで同じパラメタを用いて類似コード片を推薦が行えるか調査する。また、Kamiya ら [6] はユーザ定義の変数名や定数を予め決められた記号に置換したうえで、ソースコードをトークン列として扱うことで一部の変数名が異なるような類似コード片の検索をしている。この手法では、入力コード片のトークン列と検索したいコード片のトークン列が連続して一致しなければ検索することが出来ない。本研究では、コード片のベクトルを Word2Vec で生成される多次元空間のベクトルの総和で求める。よって、コード片中の単語の順序に影響されないため、トークン列の連続が一致しないコード片でも検索することができる。

宇田川 [7] は文書内での単語に対して重み付けを行う TF-IDF (Term Frequency-Inverse Document Frequency) 法を用いてコード片の特徴量を抽出しベクトル空間モデルで類似コード片検索を行なった。しかし、TF-IDF 法を用いて類似度を計算した場合には、単語が一致しない場合には類

似度が低くなってしまう。本研究では、単語をベクトルとして扱うため単語が一致しない場合でも類似語であればコード片同士の類似度が高くなる。そのため、Word2Vec で生成されるモデルに対して入力として単語を与え、その総和を求めることでコード片の特徴量を抽出している。

著者ら [3] はこれまでにソースコードに対して Word2Vec で学習を行い、ソースコード中の単語から類義語を特定し類似コード片を検索する手法を提案した。この手法では自然文章に対して Word2Vec で学習を行うと周辺の単語が類似している単語同士はコサイン類似度が高くなることを利用し類義語を特定している。類義語を特定した後に類義語が含まれているコード片を推薦することで検索クエリを考える手間を軽減した類似コード片検索を行う研究である。しかし、入力として与える単語によって結果に大きな違いが生まれてしまうことから検索クエリを考える手間を軽減できなかった。本研究では、入力としてコード片を与え類似コード片を推薦することで検索クエリを考える手間を軽減する。

本研究では、評価実験として類似したコード片グループの 1 つのコード片を入力として他のコード片が推薦出来たか調査するため、正解データとして類似したコード片グループを作成する必要がある。しかし、類似したコード片というのは主観的な意見が入るため類似したコード片を用意するのは難しい。類似したコード片を検出する方法の一つとしてコードクローン検出ツールが挙げられる [6] [8]。コードクローンとは、主にコピー&ペーストによって発生し、ソースコード上に存在する同一、または類似したコード片である [9]。そこで、ソースコード中の類似コード片を検出する手法の一つとしてコードクローン検出ツール CCFinderX [10] を用いることで類似コード片グループを作成する。

3. 提案手法

本節では、Word2Vec を用いてコード片を入力として類似コード片の推薦を行う方法について説明する。Word2Vec では、関連する単語同士は類似した文章中出现しやすいという仮定を基に、単語の意味をベクトル表現化したモデルを生成する。具体的には、特定の単語の周辺に出現する単語の出現確率によりベクトル表現化を行う。提案手法の概要を図 1 に示す。本研究はコード片の特徴量を抽出する処理と検索対象のコード片を作成する処理、類似コード片の推薦の 3 つの処理で構成されている。特徴量を抽出する処理はコード片からコード片の意味を表現したベクトルを生成するために、検索対象コード片の作成と類似コード片の推薦の 2 つの処理で用いる。検索対象のコード片を作成する処理は推薦高速化のために、最初に一度だけ実行し処理結果をデータベースに蓄積する。その後、類似コード片

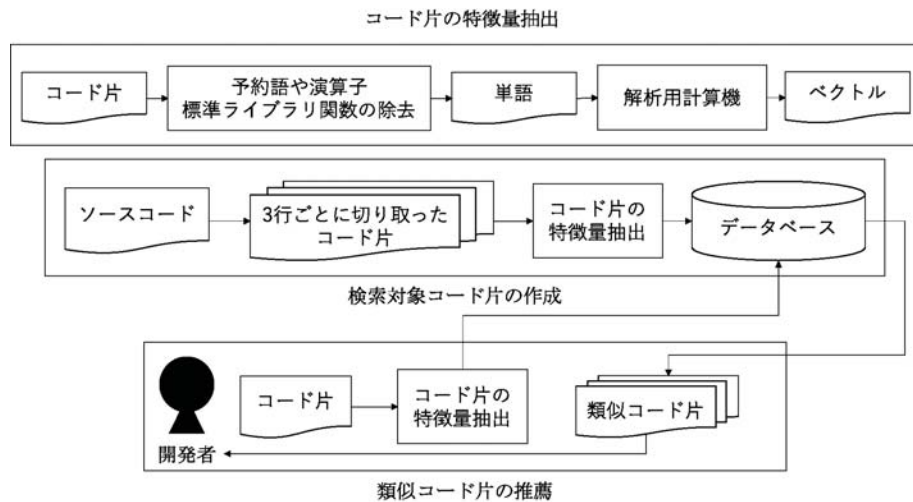


図 1 システム概要図

```
buf += HEADER_SIZE; Request.type5.context = S2TOS(buf);
buf += SIZEOFSHORT; Request.type5.size = S2TOS(buf);
buf += SIZEOFSHORT; Request.type5.mode = L4TOL(buf);
ir_debug( Dmsg( 10, "req->context = %d\n", Request.type5.context ));
ir_debug( Dmsg( 10, "req->size = %d\n", Request.type5.size ));
```

↓

```
buf += HEADER_SIZE; Request.type5.context = S2TOS( buf,
buf += SIZEOFSHORT; Request.type5.size = S2TOS( buf,
buf += SIZEOFSHORT; Request.type5.mode = L4TOL( buf,
ir_debug( Dmsg( 10, req ->, context, =, %, d, n, Request.type5.context,
ir_debug( Dmsg( 10, req ->, size, =, %, d, n, Request.type5.size
```

図 2 単語抽出の例

を検索する際には既に蓄積されているデータを用いて推薦を行う。

3.1 コード片の特徴量抽出

本研究では以下の手順でコード片の特徴量を抽出する。

- (1) コード片の正規化し単語を抽出
- (2) 得られた単語の中から予約語や演算子，標準ライブラリ関数を取り除く
- (3) 残った単語を Word2Vec でベクトル化
- (4) 単語のベクトルの総和を算出

ソースコードでは演算子前後や括弧の前後においてスペース文字の有無など同じ内容の処理でも開発者によって書き方が異なる場合が多く存在する。本研究では，僅かな違いがあっても同じ内容の処理であれば類似コード片として推薦を行うことを目的とする。そこで，ピリオドや括弧，セミコロンを取り除くことで正規化を行なった。図 2 はソースコードを前処理によって正規化し単語を抽出する例を示している。図 2 では，抽出された単語をカンマ (,) 区切りで表示している。また，本研究では関数名として用いられている識別子と変数名として用いられている識別子を区別するために関数名の識別子の最後尾に「(」を追加する。図 2 では「S2TOS」という関数名に対して「S2TOS(」として抽出する。変数名として用いられている識別子には

何も追加せずに区別する。次に，抽出された単語から予約語や演算子，標準ライブラリ関数などの頻出単語を取り除く。本研究では，ISO/IEC 9899:1999 で規定されている単語を参考に予約語と演算子として処理を行う。プログラミング言語では予約語や演算子，標準ライブラリ関数を用いるため記述が簡潔になる特徴がある。そのため，プログラミング言語は自然文章と比べると語彙の偏りが見られる傾向にある。Word2Vec では近傍の単語を基に学習するため，予約語や演算子，標準ライブラリ関数など頻出単語が出現頻度の少ない単語の学習に影響を与えている可能性がある。よって予約語や演算子，標準ライブラリ関数を取り除くことで取り除く前と Word2Vec の学習の違いがあるか比較する。しかし，予約語や演算子，標準ライブラリ関数はコード片の特徴として重要であり，あくまでここでは影響を調べるために取り除く処理を行う。取り除く処理を行い，残った単語は Word2Vec から生成されたモデルに入力として与えることで多次元空間のベクトルを得る。モデルを生成する際に Word2Vec で学習する近傍の単語数を増やすことで頻出単語以外の単語を近傍に含む。本研究では，ソースコード中において周辺の単語が類似していれば処理内容が類似していると仮定し，単語を文字列ではなくベクトルとして扱うことで変数名が異なるコード片でも推薦を行う。コード片を文章として考えた場合に，文章は単語の集合であるため，文章は単語ベクトルの総和で表すことができる。よって，単語ごとに得られたベクトルの総和を算出することでコード片の特徴ベクトルを算出する。

3.2 検索対象コード片の作成

検索対象コード片の作成では，ソースコードに対して決められた行数で切り出しを行い，切り出されたコード片に対してコード片の特徴量抽出を行う。その後，抽出された

特徴量はファイルのパスや行番号と共にデータベースで蓄積される。本研究では、実装を簡略化するために特定の行から前後 1 行の合計 3 行の定数で切り出しを行う。予備実験における対象の正解データがおおよそ 3 行であることから切り出しの基準を 3 行としている。しかし、行単位で切り出した場合に、開発者によって行単位の単語数が大きく異なっている場合に情報量が違いが出てしまうことを考慮出来ていない。また、3 行で切り出した場合にフィッチューニングになってしまうが、ノイズが含まれないコード片で学習した場合に類似コード片を推薦することができると確認するために 3 行で切り出しを行った。図 3 に 5 行のソースコードを 3 つのコード片に切り出す例を示す。図 3 のような場合①と②、③の 3 つのコード片に切り出される。切り出したコード片はコード片の特徴量抽出によりベクトル化された後にソースコードの元ファイルのパスとコード片の開始行番号と共にデータベースへ蓄積される。ソースコードの元ファイルとコード片の開始行番号は推薦をするときに使用する。

```

①
② buf += HEADER_SIZE; Request.type5.context = S2TOS(buf);
③ buf += SIZEOFSHORT; Request.type5.size = S2TOS(buf);
   buf += SIZEOFSHORT; Request.type5.mode = L4TOL(buf);
   ir_debug(Dmsg(10, "req->context = %d\n", Request.type5.context));
   ir_debug(Dmsg(10, "req->size = %d\n", Request.type5.size));
    
```

図 3 3 行切り出しの例

3.3 類似コード片の推薦

類似コード片の推薦では、開発者から入力として与えられたコード片に対してコード片のベクトルを求めるためにコード片の特徴量抽出を行う。この時、Word2Vec で学習した際に語彙に存在しない単語はベクトルを取得することが出来ないためコード片ベクトルを求める計算から無視される。その後、データベースに蓄積されたベクトルの中から入力コード片のベクトルと類似度が高いコード片を探索する。この時、ベクトル同士の類似度計算には Word2Vec で類似度の高い単語を求める際に使用しているコサイン類似度を用いてデータベースからコサイン類似度が高い順でソートし、どのような推薦の変化があるのか調査するために上位 100 個のコード片を出力した。

4. 予備実験

コード片を入力として、類似コード片を推薦することが可能なのか調査するために、日本語 IME かなのリポジトリを対象として予備実験を行なった。日本語 IME かなではバージョン 3.6 と 3.6p1 の間で、セキュリティホールが原因で 21 箇所の修正が行われている。そこで、21 箇所の修正箇所をそれぞれを修正開始行から 3 行のコード片として切り出して正解データとする。正解データをそれぞれ入力として与え、Word2Vec を用いてコード片を推薦した

```

buf += len * SIZEOFSHORT;
Request.type12.dicname = (char *)buf;
ir_debug(Dmsg(10, "req->context = %d\n", Request.type12.context));
    
```

図 4 再現率が低かったコード片の例

```

buf += HEADER_SIZE; Request.type4.context = S2TOS(buf);
buf += SIZEOFSHORT; Request.type4.begin = S2TOS(buf);
buf += SIZEOFSHORT; Request.type4.end = S2TOS(buf);
    
```

図 5 再現率が高かったコード片の例

時に予約語や演算子、標準ライブラリ関数や Word2Vec の学習する近傍の単語数が影響を与えているか調査を行なった。実験の結果を表 1 に示す。表 1 では番号は入力として与えた正解データを 1 から 21 番まで順に付けた数字を表す。また、再現率は 21 個の正解データの内どれだけの正解データを推薦したかを表す。本研究では、必ず 100 個のコード片が推薦されるため、適合率では評価を行わず再現率でのみ評価を行う。

表 1 21 個の修正箇所類似コード片
推薦を行なった場合の再現率

番号	再現率
1	0.143
2	0.143
3	0.857
4	0.857
5	0.857
6	0.857
7	0.857
8	0.857
9	0.857
10	0.143
11	0.857
12	0.857
13	0.143
14	0.810
15	0.524
16	0.857
17	0.857
18	0.857
19	0.857
20	0.857
21	0.857

表 1 の結果から 1 と 2, 10, 13 番目の 4 つのコード片で特に再現率が低くなる結果となった。再現率が低くなったコード片の一例を図 4 に、高くなったコード片の一例を図 5 に示す。再現率が高くなったコード片は図 5 の下線部が異なるという傾向が見られた。よって、変数名と数字を用いて区別を行なっているコード片や呼び出しているメンバ変数が異なる傾向が見られた。それに対して再現率が低くなったコード片は図 4 のように演算子や図 5 では用いられていないユーザ定義の関数などが多く含まれているという

傾向が見られた。そこで、予約語や標準ライブラリ関数、演算子などプログラミング言語に多く含まれる要素をそれぞれを取り除き推薦したコード片について影響を調査した。実験結果を表 2 に示す。番号は入力として与えた正解データを 1 から 21 番まで順に付けた数字を表す。取り除く前を Canna とし予約語を取り除いた結果を Reserved, C 言語の標準ライブラリ関数として stdio.h と stdlib.h に記述されている関数を取り除いた結果を Function, 演算子を取り除いた結果を Operator, 演算子や予約語, 標準ライブラリ関数を取り除かずに, Word2Vec で近傍 64 個の単語を対象に学習したモデルを用いた結果を Window64 とした。Window64 以外は近傍 8 個の単語を対象に学習を行った。

表 2 頻出単語の影響を調査した結果

	Canna	Reserved	Function	Operator	Window64
1	0.14286	0.28571	0.14286	0.14286	0.61905
2	0.14286	0.95238	0.90476	0.90476	0.95238
3	0.85714	0.90476	0.85714	0.80952	0.90476
4	0.85714	0.90476	0.85714	0.80952	0.90476
5	0.85714	0.90476	0.90476	0.85714	0.90476
6	0.85714	0.90476	0.85714	0.85714	0.90476
7	0.85714	0.90476	0.90476	0.80952	0.90476
8	0.85714	0.90476	0.85714	0.85714	0.90476
9	0.85714	0.85714	0.85714	0.85714	0.90476
10	0.14286	0.76190	0.71429	0.57143	0.80952
11	0.85714	0.90476	0.85714	0.80952	0.95238
12	0.85714	0.95238	0.85714	0.80952	0.95238
13	0.14286	0.23810	0.14286	0.19048	0.66667
14	0.80952	0.95238	0.90476	0.90476	0.95238
15	0.52381	0.33333	0.38095	0.85714	0.42857
16	0.85714	0.90476	0.90476	0.80952	0.90476
17	0.85714	0.85714	0.85714	0.80952	0.85714
18	0.85714	0.90476	0.85714	0.80952	0.85714
19	0.85714	1.00000	0.90476	0.80952	0.90476
20	0.85714	0.85714	0.80952	0.80952	0.85714
21	0.85714	0.85714	0.85714	0.80952	0.90476

頻出単語の影響を調査した結果、予約語を取り除いた場合には、取り除く前と比べると再現率が低くなった 4 つのコード片において再現率の向上が見られた。また、19 番目のコード片では 21 個のコード片すべてを推薦することが出来た。しかし、15 番目のコード片では取り除く前よりも低い再現率になった。標準ライブラリ関数を除いた場合には、多くのコード片で再現率の向上が見られ、最も低かった 1 番目のコード片でも半分以上のコード片を推薦することが出来た。演算子を除いた場合には、予約語を取り除いたときに取り除く前より再現率が低くなった 15 番目のコード片において最も高い再現率となった。Word2Vec で近傍 64 個の単語を対象に学習したモデルを用いた場合には、多くのコード片で再現率の向上が見られた標準ライブラリ関数を取り除いた場合に再現率が低くなった 1 番目

表 3 総単語数と検出されたコードクローン数

	Canna	Mozc
総単語数	315,804	1,059,131
コードクローン数	733	3,850
取り除いた予約語数	33,794	48,281

のコード片で最も高い再現率となった。

5. 実験

予約語を取り除いた場合と Word2Vec で学習する際の近傍の単語数を増やした場合でどれだけの類似コード片が推薦出来るか調査を行なった。実験対象として日本語 IME かん々と Google が開発した日本語 IME の Mozc の 2 つに対して実験を行った。Mozc を調査対象として選んだ理由は、GitHub で多くのスターが付けられていることとかなんと同じく日本語 IME のプロジェクトであるためである。正解データとして類似コード片検出方法の 1 つであるコードクローン検出ツール CCFinderX で検出されたコードクローンをを用いた。コード片同士が類似しているかどうかは主観的な基準が入ってしまうため、客観的な類似したコード片の正解データを作成するための基準としてコードクローンをを用いた。コードクローン検出ツールで検出されたコードクローンは 2 つ以上のコード片で構成されている。かん々と Mozc のそれぞれの総単語数と検出されたコードクローン数を表 3 に示す。検出されたコードクローンそれぞれに対して、1 つのコード片を入力としてコードクローンとして検出された他のコード片をどれだけ推薦出来るか調査を行った。推薦出来たかどうかは推薦されたコード片の開始行番号がコードクローンの開始行番号が一致しているかで判定している。調査結果を表 4 と表 5 に示す。

表 4 かんんで推薦したコードクローン数

	Canna	Reserved	Window64
推薦したコードクローン数	468	461	473

表 5 Mozc で推薦したコードクローン数

	Mozc	Reserved	Window64
推薦したコードクローン数	2	0	1,864

かんんでは、検出された 733 個のコードクローンに対して、頻出単語を取り除く処理をしない場合では 468 個のコードクローンを類似コード片として推薦することが出来た。しかし、予約語を取り除いてから学習した場合には推薦出来るコードクローンの数が僅かに減少してしまった。一方で Word2Vec で学習する単語を近傍 8 個から近傍 64 個にした場合は推薦できるコードクローンの数が僅かに増加した。Mozc では、検出された 3,850 個のコードクローンに対して、頻出単語を取り除く処理をしない場合では、類似コード片として推薦すること出来たのは 2 個のみであっ

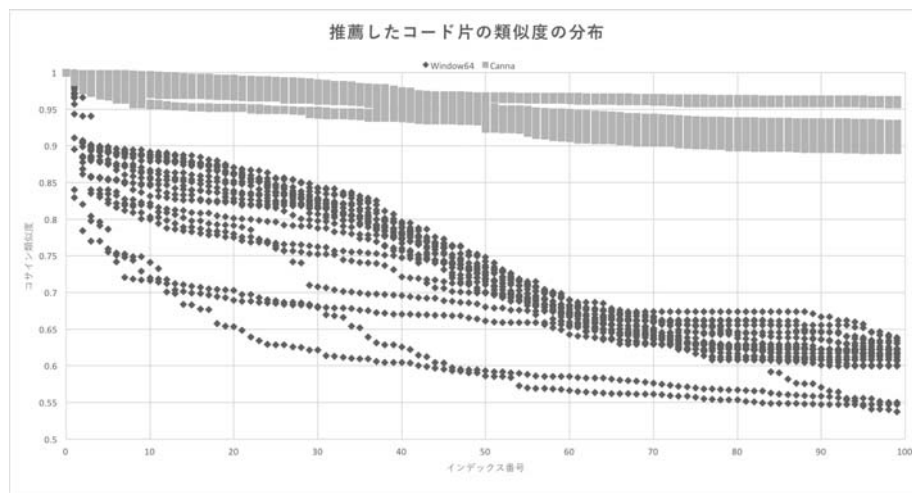


図 6 推薦したコード片の類似度の分布

た。また、予約語を取り除いてから学習した場合にはコードクローンは推薦出来なかった。一方で Word2Vec で学習する単語を近傍 8 個から近傍 64 個にした場合は大きく数が増加した。

6. 考察

Word2Vec では近傍の単語を基に学習を行うため予約語や演算子、標準ライブラリ関数を取り除くことで再現率の向上が見られた。しかし、演算子を取り除いた場合以外 15 番目のコード片で再現率の減少が見られた。15 番目のコード片を図 7 に示す。図 7 を見ると 15 番目のコードは図 5 の再現率が高くなったコード片に比べると演算子が多く含まれている。よって、演算子が近傍の単語に含まれる確率が高く上手く特徴を確認できていなかったため、取り除いた場合に 15 番目のコード片では改善され再現率が高くなったのである。予備実験と実験結果から近傍の単語では多くの語彙が含まれた方が良く特徴を学習出来るため、近傍の単語数を 8 個から 64 個で学習した場合が良い結果となった。本研究では、コード片の入力の時に 100 個の類似コード片候補が推薦される。入力コード片と出力された 100 個の類似コード片のコサイン類似度の値を分布図で表したものを図 6 に示す。Word2Vec で学習する近傍の単語数を 64 個にしたものを Window64、予約語や演算子などの頻出単語を取り除かずに近傍の単語数を 8 個にしたものを Canna とした。縦軸はコサイン類似度を表しており、横軸は類似度でソートされた 100 個のコード片中の何番目に推薦されたかを表している。その結果、処理なしで推薦された場合の類似コード片推薦では、入力したコード片と類似コード片の類似度は全て 0.9 から 1 までに分布している。これより、多くのコード片ベクトルが似たようなベクトルになっていることがわかる。よって出現頻度の低い単語を学習するときに頻出単語が周辺に多く出現し、似たようなベクトル

ルになっていると考える。Word2Vec で学習する近傍の単語数を 8 個から 64 個にした場合は 0.5 から 1 までに分布しており、文字列としてほぼ完全に一致していなくても推薦出来るため、変数名などの違いがある場合でも類似コード片として推薦することが出来る。

```
buf += len;  
Request.type13.yomi = (Ushort *)buf;  
len = ((int)Request.type13.dataLen - len - sizeofSHORT * 4) / sizeofSHORT;
```

図 7 15 番目の正解コード片

7. まとめ

近傍の単語が類似している単語同士のベクトルはコサイン類似度が高くなる Word2Vec の特徴を用いて類似コード片推薦を行なった。しかし、ソースコードを学習データとして学習する時に頻出単語が近傍単語に与える影響について考慮出来ていない問題があった。そこで本研究では、予約語や演算子、標準ライブラリ関数を取り除く処理と近傍単語の個数を増やすことで頻出単語が近傍単語に与える影響について調査した。評価実験としてオープンソースソフトウェアであるかんたと Mozc の 2 つに対して、コードクローン検出ツールで検出されたコードクローンを正解データとして実験を行なった。その結果、予約語を取り除いた場合には僅かに再現率の減少が見られたが、近傍の単語数を増やした場合には全体的に再現率の向上が見られた。よって、頻出単語を取り除くよりも近傍の単語数を増やす方が Word2Vec を用いて類似コード片推薦を行う上で有効であることが分かった。今後は、近傍の単語数を考慮し推薦されたコード片とコードクローン検出ツールで検出された類似コード片との違いについて調査し、提案手法でのみ推薦可能なコード片について調査したいと考えている。

参考文献

- [1] Yoshida, N., Hattori, T., Inoue, K.: Finding Similar Defects Using Synonymous Identifier Retrieval, *In Proceedings of the 4th International Workshop on Software Clones (IWSC '10)*, pp. 49-56, 2010.
- [2] Tomas, M., Kai, C., Greg, C., Jeffrey, D.: Efficient Estimation of Word Representations in Vector Space, *arXiv preprint arXiv:1301.3781*, 2013.
- [3] 内山武尊, 新美綾彦: Word2Vec を用いた類義語によるコード片検索の検討, ソフトウェア工学の基礎 23 日本ソフトウェア科学会FOSE 2016 (レクチャーノート/ソフトウェア学), pp. 257-258, 2016.
- [4] NEC: Canna Home Page, NEC (online), available from <http://www.nec.co.jp/canna/> (accessed 2017-5-19)
- [5] Marcus, A., Maletic, J.: Identification of high-level concept clones in source code, *Proc. 16th International Conference on Automated Software Engineering*, pp. 107-114, 2001.
- [6] Kamiya, T., Kusumoto, S., Inoue, K.: CCFinder: A multilingual token-based code clone detection system for large scale source code. *IEEE Trans. Softw. Eng.*, pp. 654-670, 2002.
- [7] 宇田川佳久: 制御構造とメソッド名のシーケンスに基づくソースコードの類似検索手法, 電子情報通信学会論文誌. D, 情報・システム J96-D(10), pp. 2182-2191, 2013.
- [8] Ueda, Y., Kamiya T., Kusumoto, S., Inoue, K.: Gemini: Maintenance Support Environment Based on Code Clone Analysis, *8th International Symposium on Software Metrics*, pp. 67-76, 2002.
- [9] 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 D Vol.91-D No.6, pp. 1465-1481, 2008.
- [10] 神谷 年洋: CCFinder ホームページ 跡地, CCFinder ホームページ (online), available from <http://www.ccfinder.net/index-j.html> (accessed 2017-5-19)