B-006

word2vec によるソースコードとドキュメント間の対応付け
Mapping of Source Code with Document Using word2vec

若竹 慶則 † 小澤 彩果 † 原田 史子 ‡ 島川 博光 †
Wakatake Yoshinori Ozawa Ayaka Harada Fumiko Shimakawa Hiromitsu

1. はじめに

ソフトウェアを開発するコーダの仕様理解不足は, コードから仕様への対応を不明瞭にする [1]. コードが対応する箇所の仕様を見ようとする保守者の負荷は増大する. 本論文は, コードと仕様書での単語の共起確率で, 双方の箇所を対応付ける手法を提案する. 提案手法では, word2vec でコードと仕様書それぞれの単語をベクトル化し, それぞれの箇所をベクトルで表現する. cos 類似度とユークリッド距離でこれらのベクトルの一致度を比較したところ, ユークリッド距離が比較的良い結果を示した. また, これより, Java のクラス名とメソッドからドキュメントの節への対応がついた. これにより, 対応がつかない箇所が自動的に判明するので, 保守者の負担を軽減できる.

2. ソースコードとドキュメントの対応付けのためのアプローチ

2.1 Feature Location の現状

Feature Location は, 機能に対応するコード片を特定するという考えであり, 今までに多くの手法が提案されている. 例えば, Javadoc[2] のようなソースプログラム中にドキュメントを簡潔に記述するためのツールが提供されている. しかし, 開発現場では提案されてきた手法が活用されていない状況であり, Feature Location の方向性や評価方法が議論されている [3].

2.2 word2vec

word2vec は, ニューラルネットワークを用いた単語の分散表現の計算手法である. これは skip-gram モデルと呼ばれる言語モデルによって単語の分散表現を取得する. 具体的には, コーパス中の文脈において特定の単語と共起する単語, すなわち付近に存在する単語の出現確率を最大にするようなパラメータ調整を行うことで, 単語の密なベクトルを生成する [4].

3. word2vec によるソースコードとドキュメントの対応付け

3.1 提案手法

まず, 自然言語処理に不可欠である前処理を, ソースコードとドキュメントに対してする. 前処理をしたそれぞれのデータを基に, 対応付けをする. 本研究では, じゃんけんゲームの仕様を記述した Java 言語をソースコードの学習データとした. また, Javadoc[2] のコメントに自分でコメントを加えたものをドキュメントの学習データとした. 提案手法の概要を図 1 に示す.

†立命館大学
‡(株) コネクトドット

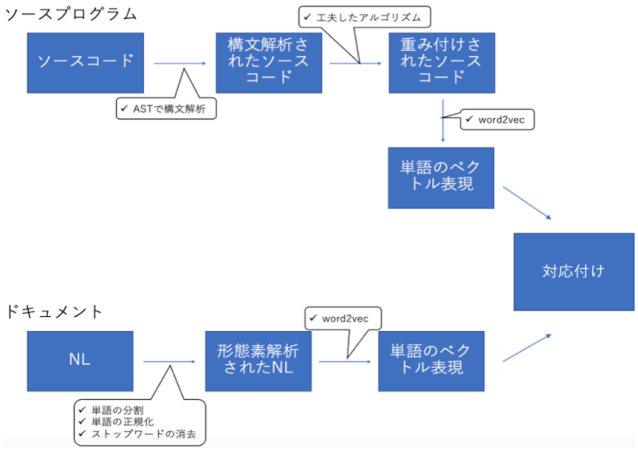

図 1: 提案手法の概要図

3.2 ソースコードに対する前処理

本研究では, JDT で Java のソースコードを抽象構文木に変換し, 構文解析する [5]. 次に, word2vec では出現頻度が低い単語同士は距離が近くなりやすいため, 出現頻度を抑えるためにストップワードの消去する [6]. ストップワードは, 「1 つの関数内に 3 回以下しか出てこない変数」, 「(「string」, 「void」, 「int」, 「double」) などの変数の型名」, 「if 文, switch 文の条件文, 条件式 (ブロック文は残す)」, 「(「+」, 「−」, 「=」, 「*」) などの計算演算子」, 「文脈とはあまり関係のない引数」とする. 次に, 対応付けの際に重要だと考える, クラス名に対して 10, public メソッド・public 変数に対して 5 の重み付けをする. 重み付けは, コピーアンドペーストでする. 最後に, word2vec を用いて単語をベクトルに変換する. 本研究では, 単語を低次元の実数値ベクトルで表す「分散表現」を使用する. ソースコード, ドキュメント共に 59 次元である. window サイズは, 10 とする.

3.3 ドキュメントに対する前処理

本研究では, Janome を使用して文章の単語分割をする [7]. 次に, ストップワードの消去をする. ストップワードは, 「助詞や助動詞などの機能語」, 「括弧や句読点などの記号」, 「数字」とする. 次に, 正規化をし, 文字種の統一をする. アルファベットの大文字を小文字に変換, 半角文字を全角文字に変換, 表記揺れ, 省略語, 口語表現に対する処理をする. 最後に, word2vec を用いて単語をベクトルに変換する.

3.4 ソースコードとドキュメントの対応付け

ソースコードとドキュメントをそれぞれベクトル化したものに対して, 対応付けをする. 本研究では, ソースコードのベクトルとドキュメントのベクトル間の類似度を, cos 類似度, ユークリッド距離でそれぞれ計算した場合の精度を比較する. また, ユークリッド距離に関して, 重み付けを行なった場合と行なっていない場合を比

較する．実験はそれぞれ 10 回ずつとし，その平均を結果として示す．対象とする単語は，ソースコードの（start, judge），ドキュメントの（開始, 判定）とする．ソースコードとドキュメントから（開始, start），（判定, judge）は類似度が高く，（開始, judge），（判定, start）は類似度が低いという結果であれば，学習の効果があり対応付けの精度も良いとする．結果を表 1, 表 2, 表 3 に示す．

表 1: cos 類似度

(ソースコード, ドキュメント)	cos 類似度
(開始, start)	0.02
(判定, judge)	-0.13
(開始, judge)	0.03
(判定, start)	-0.06

表 2: ユークリッド距離

(ソースコード, ドキュメント)	ユークリッド距離
(開始, start)	0.05
(判定, judge)	0.03
(開始, judge)	0.13
(判定, start)	0.35

表 3: ユークリッド距離 (重み付けなし)

(ソースコード, ドキュメント)	ユークリッド距離
(開始, start)	0.04
(判定, judge)	0.03
(開始, judge)	0.21
(判定, start)	0.42

3.5 評価

結果を見ると，cos 類似度の場合，全ての場合で値が 0（$\theta = \pi/2$）前後であり，良い結果を示したとは言えない．ユークリッド距離の場合，多少なりとも差が出ており，Java のクラス名とメソッドから仕様書の節への対応としては比較的良い結果を示していると言える．cos 類似度に関しては，次のようなことが考えられる．例えば，分かりやすく 4 次元で考えた場合，One hot 表現で，開始（1, 1, 0, 0），start（0, 0, 1, 1）のように表されるとすると，ベクトルの向きはそれぞれ，開始↑，start →となり，cos 類似度を出すと当然 90° 前後となる．このようなことが N 次元になった場合も起こり，今回このような結果が出てきたのではないかと考える．また，日本語と英語では，ベクトルの軸がそもそも異なるからといった理由も考えられる．そのため，cos 類似度による算出は，適していないと考える．ユークリッド距離に関しては，非常に小さな差が出ているだけで，全ての単語群において類似度が非常に高いという結果が出ており，一見しただけでは，この結果が類似度の精度として正しいのかということが分からない．cos 類似度と比較すると良い結果を示していると言えるが，ソースコードとドキュメントの対応付けの精度としては改善すべき点が非常に多くあると考える．また，ユークリッド距離において，重み付けあり，重み付けなしの場合を行なったが，本研究の重み付けの方法では，重み付けをしたことで精度が向上したとは言えなかった．以上より，日本語と英語といった異なる言語間のベクトルの類似度算出において，ユークリッド距離を用いることは有用であるとは考えるが，改善すべき点・工夫すべき点は大いにある．

4. おわりに

本研究では，word2vec を用いてソースコードとドキュメントをベクトル表現し，類似度が高くなるべき単語群，なるべきでない単語群を対象とし，cos 類似度，ユークリッド距離を用いた単語間の対応付けの精度を図った．結果としては，ユークリッド距離を用いる手法が，比較的良い結果を実現させ，Java のクラス名とメソッドからドキュメントの節への対応がついた．しかし，対応付けの精度としては改善すべき点が非常に多くあると考える．

対応付けの精度の向上に関しては，最適なパラメータの発見と重み付け方法の見直しを挙げる．重み付け方法の見直しに関しては，文章内，文章全体での単語の出現頻度による重み付けを行う tf-idf 値を用い，word2vec と結合させることで，精度が上がるのではないかと考える．tf-idf 値と word2vec を単体で用いるのではなく，結合させることで，全く異なる性質のベクトルを学習し，両方の特徴量から学習することができ，類似度の精度が上がるのではないかと考える．

参考文献

[1] Engineer.club. http://www.globus.org/toolkit/, 2018.

[2] 後藤英斗, 大久保弘崇, 粕谷英人, 山本晋一郎ほか. 文脈に基づいたソースプログラムとドキュメント間の識別子対応付け手法. 情報処理学会研究報告ソフトウェア工学 (SE), Vol. 2005, No. 29 (2004-SE-147), pp. 41–48, 2005.

[3] 岡田譲二, 鹿島悠, 坂田祐司. Feature location 技術の開発現場での活用に向けて. ウィンターワークショップ 2013・イン・那須 論文集, 第 2013 巻, pp. 41–42, jan 2013.

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

[5] org.eclipse.jdt.astview-ast view — the eclipse foundation. https://www.eclipse.org/jdt/ui/astview/, 2020.

[6] 単純な単語のベクトル表現. https://qiita.com/yuku_t/items/483b56be83a3a5423b09, 2015.

[7] Janome. https://mocobeta.github.io/janome/, 2015.