Vol.2017-CE-142 No.17
Vol.2017-CLE-23 No.17
2017/12/9

アルゴリズム推定に向けたAutoencoderに基づくプログラムのベクトル表現

間嶋 義喜[1,a)]　岡田 拓也[1,b)]　竹内 和広[2,c)]

概要：プログラム教育支援には，分析対象プログラムが採用するアルゴリズムの推定が有益に作用すると考えられる．採用アルゴリズムの推定には，分析プログラムを AST(Abstract Syntax Tree) に表現した上で，その木構造上の推定に寄与する特徴の選定が課題になる．本稿では，プログラムの構造的特徴を表現するために，ソフトウェア工学分野で提案された，プログラムのベクトル化手法の一つである状態ベクトルを拡張する．具体的には，従来手法の AST 表現における非終端記号ノードの出現回数に基づいたベクトル表現を，Autoencoder を用いて別のベクトル空間に変換する．このベクトル化手法の有効性を，プログラムからの目的推定および，プログラムのクラスタリングにより検討する．

キーワード：プログラム構造評価，プログラム類似度，プログラム教育，機械学習

An autoencoder-based vector model for estimating algorithms used in programs

Yoshiki Mashima[1,a)]　Takuya Okada[1,b)]　Kazuhiro Takeuchi[2,c)]

Abstract: An estimation system that analyzes the algorithms that a certain input program employs will support program education. In this paper, we select structural features of the program contributing to the estimation of algorithms. In particular, we extend a vector model that is based on the Abstract Syntax Tree (AST) of the target program to express the structural features of the program. Our proposed method employs autoencoder to convert a vector representation based on the number of occurrences of a nonterminal symbol node in the AST representation to another vector space. As evaluation of the effectiveness of our proposed method, we examine by estimating purposes of programs and clustering them.

Keywords: Evaluation of Program Structure, Similarity Measure between Programs, Program Education, Machine Learning

1. はじめに

未知のプログラムが採用するアルゴリズムや手法を，ソースコードを詳しく読むことなく推定することができれば，プログラム教育や学習を支援する上で，有益な基盤技術になるものと思われる．

本稿では，上記の動機づけの下で，プログラムを分析するために提案された抽象構文木（Abstract Syntax Tree, AST）の節集合から，プログラムの目的に対応して現れるパターンを自動的に抽出することを目指す．ここで，パターンとは，プログラムの熟練者や教育者が経験的に持っていると考えられる，プログラムを分析をする際の構造類型を仮定している．そのようなパターンには，デザインパターン [1] のように，列挙整理されたものが存在することも事実である．しかし，本稿では，そのような経験的な整理をする前段階として，機械学習の手法を取り入れ，ソー

[1] 大阪電気通信大学大学院
Osaka Electro-Communication University
[2] 大阪電気通信大学
Osaka Electro-Communication University
[a)] mi17a004@oecu.jp
[b)] mi16a002@oecu.jp
[c)] takeuchi@osakac.ac.jp

スコードの集合から，プログラム作成の上でよく採用される部分目的に対応する特徴的なパターンを自動的に選定することを検討する．

提案手法では，Neural network による高精度なクラス分類に利用される Autoencoder をプログラムの構造的特徴を検討することに利用する．一般的に，Autoencoder は入力の特徴を自動的に選定することに利用され，入力から Autoencoder を通して入力そのものを再現できるとき，Autoencoder の中間層は対象とする入力の特徴を表現しているものと考えられる．そこで我々は，プログラムを AST で表現した上で，当該プログラムの AST 上の特徴を選定する Autoencoder を作成し，AST によって表現されるプログラム構造における特徴的なパターンを得ることを狙う．具体的には，AST 特徴を選定する Autoencoder は，JDK のソースコードすべてを用いて，Java ソースコードに表出するであろう特徴の一般性を担保する．

次に，得られた Autoencoder が表現する特徴が，未知のプログラムの目的や採用アルゴリズムを知ることに繋がるプログラムの部分目的に対応する特徴として機能するかを検証する．ただし，一般にアルゴリズムを実装する方法は一通りではなく，アルゴリズムと呼ばれる手続きの中に小さなアルゴリズムが含まれる階層構造を持っているため，あるプログラムに内在する特徴をすべて網羅的に注釈付けしたデータを用意し，提案手法の特徴を検討することは困難である．そこで，本稿では，作成目的が明確なソースコードの集合を用いて，Autoencoder が表現する特徴が分類問題にどう寄与するかを検討することにより，提案手法の有効性と今後の研究の発展性を検討する．

2. プログラム特徴のベクトル表現

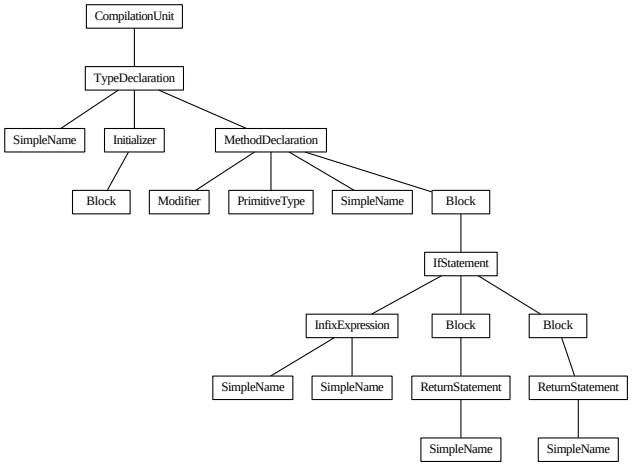
2.1 抽象構文木

プログラムは一般的に, 文脈自由文法 (Context Free Grammar) と呼ばれる文法によって規定される. この文法に則って, プログラムの特徴を表現する手法に抽象構文木 (Abstract Syntax Tree,AST) がある. 抽象構文木は, プログラム中に出現するトークンと呼ばれる識別子同士の関係性を木構造で表現する. 抽象構文木では, 文脈自由文法で定められた非終端記号が中間ノードに対応する. 図 1 に, 抽象構文木の一例を示す. 図 1(a) は, 最大値を計算するプログラムである. この図 1(a) から抽象構文木を作成すると, 図 1(b) のようになる.

プログラムを，それが記述されたプログラム言語を規定する文法に従って構文解析し，それをグラフ構造で表現する方法には多様性があり，その表現の利用目的により表現方法を適切に設計する必要がある．本来ならば，分析プログラムの固有のメソッド名の max や，変数名の x や y は終端記号であるので，AST の木構造の葉として出現するが，本稿の目的はプログラム構造におけるパターンの発見

```
class Max(){
    public int max(){
        if (x >= y) {
            return x;
        } else {
            return y;
        }
    }
}
```

(a) プログラム



(b) 抽象構文木

図 1: 抽象構文木の一例

にあるため，図 1(b) のように，終端記号は木から削除し，非終端記号同士の関係を表した木構造を扱う．

将来的には，終端記号も汎化して木構造に表現したパターンの発見が課題となるが，識別子の個別性を扱うためには，それらの識別子に関する参照情報や呼び出し情報も考慮に入れる必要がある．そのような場合，プログラムの構造を表現をするためには木構造ではなく，一般的なグラフ構造を扱う必要があり，問題の複雑性が高まる．そのため，それらの課題を解決する前段階として，今回は終端記号を除いた木構造によるパターンの発見を検討することにした．

2.2 状態ベクトル法

ソフトウェア開発において，プログラムのバージョンアップに伴うソースコードの変更作業は，人的コストがかかるだけでなく，変更箇所を見逃すなどのヒューマンエラーが起きやすい．そのため，過去の開発履歴やプログラムの修正箇所の情報などを利用して，プログラムの修正箇所をプログラマに事前に知らせる研究や，バグの発見，修正を自動で行う研究が進められている [2],[3],[4].

状態ベクトルとは，肥後らによって提案された，そのようなプログラムの開発リポジトリに含まれるプログラムの要素に着目した手法である [5],[6]．肥後らは状態ベクト

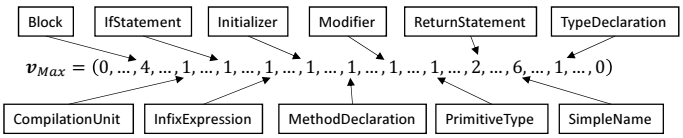


図 2: 状態ベクトルの例

ルを用いて，プログラム中の修正が必要な箇所の予測を，メソッド単位で試みている．それにより，変更すべき箇所の見逃しや，変更すべきでない箇所の特定に役立てようとしている．プログラム要素とは，IF 文や FOR 文などの制御構文，識別子名，変数宣言などのプログラム中に出現する要素一つ一つのことを指している．肥後らは，プログラム要素の特定に抽象構文木を使用し，抽象構文木の中間ノードの一つ一つをプログラム要素としている．抽象構文木は，木構造上の部分木構造を適切に計量することにより，プログラムの持つ構造的特徴を数値的に表すことが可能になるという特徴を持つ．肥後らの研究は，このような抽象構文木の特徴に着眼したものである．具体的には，肥後らは，抽象構文木の構築に JDT(Java Development Tools) で提供されている EclipseJDTParser[7] を用いている．EclipseJDTParser は Java を対象として，92 種類の中間ノードを用いて抽象構文木を構築する機能を提供している．これらの中間ノードのうち，コメントに関する中間ノードは 3 種類ある．今回は，自然言語で書かれたコメントの内容はそのままでは扱えないため，コメントに関する中間ノードを除いた 89 種類の中間ノードを状態ベクトルの次元とした．状態ベクトルの各次元の値は，出現するそれぞれのプログラム要素の出現回数となっている．

図 2 に，プログラムから生成される状態ベクトルの一例を示す．図 1(b) から図 1(a) のプログラム中には，89 種類のプログラム要素のうち，11 種類のプログラム要素が出現していることがわかる．したがって，89 次元の要素のうち，11 次元の要素は 1 以上の値を持ち，残りの 78 次元の要素の値は 0 となっている．なお，図 2 では状態ベクトルの先頭と末尾を除き，値が 0 である要素の記述を省略している．

2.3 他のベクトル化手法

Mou らは, プログラムを抽象構文木に基づくベクトル表現に変換することによって, プログラム分析のために，機械学習の活用を試みている [8]. 具体的には, 本当と同様に，プログラムを抽象構文木で使用される非終端記号に絞って表現する．次にその表をを利用して，プログラム群を表現するベクトル空間を得る．このベクトル空間は分散表現と呼ばれ，その獲得には Neural network を利用している．

分散表現は，自然言語処理の分野で提案された，単語や文を, ベクトル空間上に配置可能なベクトルとして表現する手法である．その際，ベクトルの要素の値は実数値であり，ベクトル間の関係が密になるように表現する．Mou らの研究は，この分散表現の獲得を，プログラム構造の表現に応用したものであり，抽象構文木のノード間の関係調査やプログラムの分類調査を行っている. この方向性は，本稿の提案手法と共通する．

分散表現の研究は，自然言語処理の分野で，言語の様々な構成単位に対して検討されている．最も一般的に利用されているものが，Word2Vec と呼ばれる単語を分散表現に変換するツールである [9],[10]. 単語以外の単位については，例えば，山本らは Doc2Vec と呼ばれる，文書を単位とした分散表現を得る技術の研究を行っている [11]. また，この単語の分散表現の獲得技術を文に拡張した研究もある [12]．このように, 自然言語処理の研究では，言語の様々な単位ついて，その要素の意味を数値的に扱うことが検討されている．

本研究は，プログラム構造に適した分散表現を得るために適切な Autoencoder を設計する位置づけにある．

3. Sparse Autoencoder

Autoencoder は入力をそのまま教師値に持つ Neural network のアーキテクチャであり，データのみから学習を行う教師なし学習である．すなわち，ネットワークの出力を $\boldsymbol{y}$，教師値 (入力) を $\boldsymbol{d}$ としたとき，(1) 式で定義する損失関数 $\boldsymbol{E}(\boldsymbol{w})$ を最小化する．

$$\boldsymbol{E}(\boldsymbol{w}) = \sum_k (y_k - d_k)^2 \tag{1}$$

学習済みの Autoencoder の中間層は特徴抽出器として働く．しかし，適切な制約を加えていない Autoencoder による特徴抽出は，意図しない特徴を学習する可能性があり，ある一定のノイズに対して非常に弱いものとなることがある [13].

一方で，Autoencoder の中間層に制約を加えるアーキテクチャに Sparse Autoencoder がある [14]．Sparse Autoencoder は，入力を与えた際の中間層の出力値が 0 に近づくよう，損失関数 $\boldsymbol{E}(\boldsymbol{w})$ を (2) 式で定義する．

$$\boldsymbol{E}(\boldsymbol{w}) = \sum_k (y_k - d_k)^2 + \beta \sum_j KL\left(\rho || \hat{\rho}_j\right) \tag{2}$$

ここで，β は正則化をどの程度重要視するかを決定するパラメータであり，これを変化させることで正則化の強さを調整することができる．$KL\left(\rho||\hat{\rho}_j\right)$ は ρ と $\hat{\rho}_j$ の距離関数であり，(3) 式で与えられる．

$$KL\left(\rho||\hat{\rho}_j\right) = \rho \log\left(\frac{\rho}{\hat{\rho}_j}\right) + (1-\rho)\log\left(\frac{1-\rho}{1-\hat{\rho}_j}\right) \tag{3}$$

また，$\hat{\rho}_j$ は入力が与えられたときの中間層の j 番目素子の出力の平均で，(4) 式で表される．

$$\hat{\rho}_j = \frac{1}{N}\sum_n^N h_j(x_n) \tag{4}$$

パラメータの更新式は，時刻 t のパラメータを $\boldsymbol{w}^t$ として (5) 式で定義する．

$$w_{ji}^{t+1} = w_{ji}^{t} - \epsilon \delta_j^l z_i^{l-1} \tag{5}$$

ここで，ϵ は学習係数，z_i^{l-1} は 1 つ前の層の i 番目素子からの入力を表し，δ_j^{l-1} は出力層，中間層でそれぞれ (6) 式と (7) 式で与えられる．

$$\delta_j^{out} = y_j - d_j \tag{6}$$

$$\delta_j^{hidden} = \sum_k \delta_j^{out} w_{kj} + \beta \left(-\frac{\rho}{\hat{\rho}_j} + \frac{1-\rho}{1-\hat{\rho}_j} \right) \tag{7}$$

(7) 式の右辺第 2 項が制約により加わった項であり，この項により中間層の素子が多くの入力に反応する限り，中間層の重みは 0 に向かって抑えられる形となり，ほとんどの入力に対しては反応せず，一部の入力に対してのみ反応する Sparse なモデルが構築できる．

これにより，中間層は限られたごくわずかな基底によって入力を表現しなければならないため，自動的に入力の構成単位を反映した基底を学習することができる．

4. 提案手法

本提案手法は，抽象構文木で表現されたプログラムから Autoencoder を用いてベクトル表現を取得する．ここで，抽象構文木は木構造であるため，直接的に Autoencoder へ入力することはできない．そのため，肥後らによって提案された状態ベクトルを用いて，抽象構文木をベクトル化した後，これを Autoencoder へ入力し，その次元圧縮による特徴抽出を通して中間表現を得る．便宜上，肥後らの提案する状態ベクトルを unigram ベクトルと呼ぶ．

我々は更に，unigram ベクトルでは表現できない各プログラム要素間の関係性を取得するため，状態ベクトルを拡張し，任意の 2 つの中間ノードを接続した中間ノードの組み合わせのパターンをベクトルの次元として利用する．このベクトルを bigram ベクトルと呼ぶ．bigram ベクトルも unigram ベクトルと同様に Autoencoder へ入力し，中間表現を得る．

Autoencoder を用いて得られた中間表現は，入力の再現は達成されているが，その特徴を正しく表現しているとは限らない．そのため，中間表現を評価するために，この中間表現を用いて訓練した Neural network を用いてクラス分類を行い，プログラムが正しく分類されるかを判定する．図 3 に，提案手法の概要図を示す．

5. 実験

5.1 使用データ

本実験で使用したデータは，JDK のソースコードとして Java SE Development Kit 8 に含まれる Java SE 標準ライブラリのソースコード [15] と，Aizu Online Judge[16],[17] で出題される各問題に対する利用者が提出した解答コード群である．これらのデータの詳細を表 1 に示す．Files はファイル総数を示し，Average，Median，Mode はそれぞれ各問題に対する解答コードのプログラム行数の平均値，中央値，最頻値を示す．Total LOC は総プログラム行数を示す．

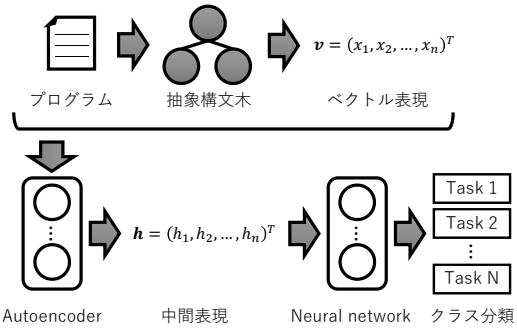


図 3: 提案手法の概要

表 1: 実験利用ソースコード

Task	Files	Average	Median	Mode	Total LOC
1	300	23.03	20	15	6,909
2	300	16.08	13	11	4,823
3	300	19.22	18	15	5,766
4	300	25.31	24	26	7,592
5	300	23.42	20	20	7,026
6	300	23.35	21	18	7,005
7	300	30.27	28	28	9,080
8	300	25.95	23	21	7,784
9	300	24.92	22	21	7,464
JDK	7,704	131.82	35	5	1,015,523

表 2: クラス分類に利用した問題内容

Task	Problem
1	数値列の上位 3 件を出力する
2	矩形の面積と周囲を計算する
3	2 つの整数の大小関係を表示する
4	3 つの整数を昇順に出力する
5	円が矩形の内側に配置されているかどうかを判断する
6	2 つの整数を昇順に出力する
7	演算記号が入った式を計算する
8	数値列から最小値，最大値，合計値を出力する
9	記号で矩形を描画する

5.2 実験設定

本実験では，プログラムからベクトル表現を作成するために EclipseJDTParser を利用する．生成されるベクトル表現は，unigram ベクトルと bigram ベクトルである．unigram ベクトルの次元数は，EclipseJDTParser で定義されている中間ノードの数を利用した 89 次元である．bigram ベクトルの次元数は，中間ノードの数の組み合わせパター

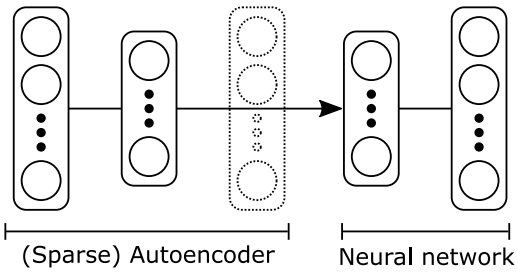


図 4: 実験に使用したネットワークの構成

表 3: 解答コードの分類結果

手法	精度
random guess	11.1%
AE+unigram	82.2%
AE+bigram	90.4%

表 4: 各モデルに使用したハイパーパラメータ

	中間層のユニット数	β	ρ
AE+unigram	30	0.01	0.5
AE+bigram	100	0.01	0.1

ンの数から，一つでも出現するパターンを利用する．すなわち，出現する中間ノードの組み合わせパターンである 644 次元を用いる．状態ベクトルは，出現頻度の数え上げであり，部分木パターンを直接的に表現してはいない．よって，bigram ベクトルから部分木パターンを抽出するため，Autoencoder(AE) を用いて特徴抽出を行う．

実験で使用するネットワークの構成を図 4 に示す．図 4 のネットワークの訓練は，Autoencoder 部分と Neural network 部分で独立して行う．まず，Autoencoder 部分を訓練した後，訓練済みの Autoencoder へ入力が与えられたときの中間層出力を用いて，Neural network 部分を訓練する．本実験では，提案手法によるベクトル化の一般性を担保するため，比較的大きなデータサイズを持つ JDK のソースコードを用いて Autoencoder 部分の訓練を行った．その後，得られた中間層表現がプログラムの特徴を表現できているかを確かめるため，解答コードを用いて分類器を訓練し，テストを行った．ここで，JDK のソースコードはテストに用いないため，全体を訓練に使用し，解答コードは各問題につき全 300 サンプルのうち，270 サンプルを訓練に，残り 30 サンプルをテストに用いた．

6. 結果および考察

Sparse Autoencoder のハイパーパラメータに適した値は自明ではないため，本実験ではグリッドサーチを用いてハイパーパラメータを探索した．それらのネットワークを用い，解答コードの分類を行ったときの結果を表 3 に示す．また，AE+unigram および AE+bigram の訓練時に設定したハイパーパラメータを表 4 に示す．ここで精度は，全てのサンプルの数を N，そのうちから正しいクラスに分類されたサンプルの数を n_c として，(8) 式を用いて計算した．

$$\textbf{Accuracy} = \frac{n_c}{N} \tag{8}$$

また，訓練した各ネットワークのハイパーパラメータと精度の関係を AE+unigram，AE+bigram についてそれぞれ図 5，図 6 に示す．

これらのネットワークが，ソースコードをどのようなベクトル空間へ配置しているのかを調査するため，訓練済みの Autoencoder の中間層出力を PCA を用いて 2 次元平面上にプロットする．300 サンプルあるプログラムを全て表示した場合，その集合関係を把握しにくいため，今回は各問ごとに中間層出力値を平均してから使用した．このときの結果を，AE+unigram，AE+bigram についてそれぞれ図 7，図 8 に示す． 図 7 より，AE+unigram によりベクトル化されたプログラムは，大別して Task1 と Task9 のグループ，Task2 と Task5 のグループ，Task4 と Task6，Task3 および Task7 のグループ，そして Task8 に分けられる．Task1 と Task9 は，目的を達成するために 2 重の FOR 文が使用されており，それ以外の Task では 2 重の FOR 文はほとんど使用されていない．また，Task1 と Task9 以外の Task では IF 文がよく使用されており，そのことから

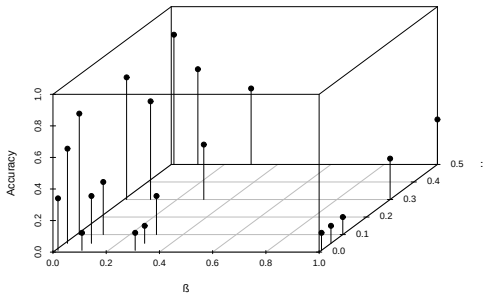


図 5: AE+unigram を用いたときのハイパーパラメータと精度の関係

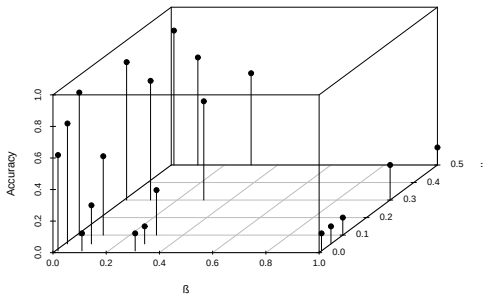


図 6: AE+bigram を用いたときのハイパーパラメータと精度の関係

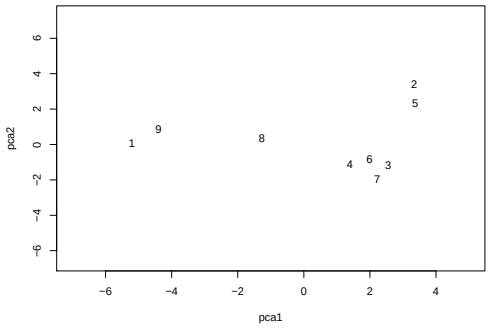


図 7: AE+unigram を用いてベクトル化した Task プログラムの分布

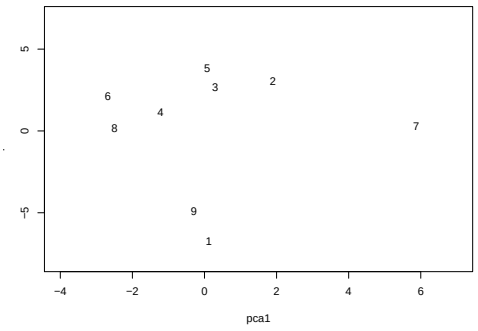


図 8: AE+bigram を用いてベクトル化した Task プログラムの分布

第一主成分軸の負の方向は 2 重の FOR 文を表し，正の方向は IF 文を表していることが示唆される．ここで，Task8 は IF 文と FOR 文の両方を使用するため，第一主成分が 0 付近になっていると考えられる．第二主成分軸は，Task2，Task5，Task9 といった比較的多くの変数を必要とするプログラムが正の方向へ固まっており，変数の使用量を表していると考えられる．

同様に，図 8 より，AE+bigram によりベクトル化されたプログラムは，大別して Task1 と Task9 のグループ，Task4，Task6，Task8 のグループ，Task2，Task3，Task5 のグループ，そして Task7 に分けられる．Task7 はその他の Task と違い，文字に対して IF 文を用いているため，これが特徴として強く影響したと考えられる．また，AE+unigram で近い位置に配置されていた Task1 と Task9 は AE+bigram でも同様に近い位置へと配置されている．

これらの結果から，提案手法によりベクトル化されたプログラムは，Task1 と Task9 のような，その目的が近いプログラム同士で近い位置へ配置されており，提案手法がプログラムの部分木パターンを表現する能力があることが示唆された．しかし，各主成分の検討や，中間層表現の各次元と入力次元の関係性の調査など，詳細な結果の解析は行えていないため，今後の課題として残る．

7. おわりに

本稿では，プログラム教育支援を目的に、プログラムの AST 表現から，プログラムの熟練者や教育者が経験的に持っていると考えられる，プログラムを分析する際のパターン抽出を検討した．提案手法は，従来手法の AST 表現における非終端記号ノードの出現回数に基づいたベクトル表現を，Autoencoder を用いて特徴選択されたベクトル空間から得ることである．また，このベクトル化手法の有効性を，プログラムからの目的推定および，プログラムのクラスタリングにより検討した．その結果，プログラム集合に属するプログラムが持つ九種類の目的のうち，一つを推定する実験において精度で 90%を達成することを確認した．さらに提案のベクトル化手法により，プログラム集合における，それぞれのプログラムが持つ特徴的な部分構造を反映して，それらを適切に分類できる可能性を主成分分析によって示した．以上の検討により，提案手法がプログラムの部分目的に対応する特徴的なプログラム構造を表現するベクトル化として適切であることを示した．

今後の課題としては，ベクトル化の基準化となる AST 表現をより高度化するために，今回の特徴ベクトル化に対応する部分木パターンを整理することを考えている．

謝辞 本研究は JSPS 科研費（基盤（C）課題番号 15K01100）の助成を受けたものです．

参考文献

[1] E.Gamma, R.Helm, R. a. J.: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Professional (1994).
[2] Le Goues, C., Nguyen, T., Forrest, S. and Weimer, W.: Genprog: A generic method for automatic software repair, *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 54–72 (2012).
[3] Nguyen, H. D. T., Qi, D., Roychoudhury, A. and Chandra, S.: Semfix: Program repair via semantic analysis, *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, pp. 772–781 (2013).
[4] Mechtaev, S., Yi, J. and Roychoudhury, A.: Directfix: Looking for simple program repairs, *Proceedings of the 37th International Conference on Software Engineering-Volume 1*, IEEE Press, pp. 448–458 (2015).
[5] Murakami, H., Hotta, K., Higo, Y. and Kusumoto, S.: Predicting Next Changes at the Fine-Grained Level, *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*, Vol. 1, IEEE, pp. 119–126 (2014).
[6] 村上寛明，堀田圭佑，肥後芳樹，楠本真二：ソースコードの自動進化に向けて，電子情報通信学会技術研究報告. MSS, システム数理と応用， Vol. 113, No. 421, pp. 107–112 (2014).
[7] EclipseJDT: EclipseJDT, http://www.eclipse.org/jdt.
[8] Mou, L., Li, G., Liu, Y., Peng, H., Jin, Z., Xu, Y. and Zhang, L.: Building Program Vector Representations for Deep Learning, *arXiv preprint arXiv:1409.3358* (2014).
[9] Mikolov, T., Chen, K., Corrado, G. and Dean, J.: Efficient estimation of word representations in vector space,

arXiv preprint arXiv:1301.3781 (2013).
[10] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S. and Dean, J.: Distributed representations of words and phrases and their compositionality, *Advances in neural information processing systems*, pp. 3111–3119 (2013).
[11] 山本哲男：分散表現ベクトルを用いたソースコードの検索及び分類の検討，電子情報通信学会技術研究報告，Vol. 116, pp. 67–72 (2017).
[12] Le, Q. and Mikolov, T.: Distributed representations of sentences and documents, *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1188–1196 (2014).
[13] Goodfellow, I. J., Shlens, J. and Szegedy, C.: Explaining and harnessing adversarial examples, *arXiv preprint arXiv:1412.6572* (2014).
[14] Ng, A.: Sparse autoencoder, *CS294A Lecture notes*, Vol. 72, No. 2011, pp. 1–19 (2011).
[15] Oracle: Java SE Development Kit 8, https://www.oracle.com/.
[16] 有隆渡部：Aizu Online Judge, http://judge.u-aizu.ac.jp/onlinejudge/.
[17] 有隆渡部：オンラインジャッジの開発と運用 -Aizu Online Judge-，情報処理，Vol. 56, No. 10, pp. 998–1005 (2015).