# 解答ソースコードを用いたプログラミング演習問題に対するタグ付け手法の提案

川渕 皓太[1,a)]　松下 誠[1]　吉田 則裕[2]　井上 克朗[3]

**概要**：IT 産業の発展に伴いプログラミング学習サイトの数が増加しており，タグ付けされた問題を解くことでユーザのプログラミング能力の判定を行うものも存在する．既存のプログラミング演習問題をこの能力の判定に用いる場合，問題のタグ付けに労力がかかることが問題となる．そこで本研究では，タグ付与済みのプログラミング演習問題を Doc2Vec で学習し，タグが付与されていない演習問題に対してタグを自動で付与する手法を提案する．提案手法を用いて実験を行った結果，タグによって推測精度の差があることがわかった．また，推測するタグをアルゴリズム情報に関するタグのみに絞ることで，精度が向上することを確認した．

**キーワード**：プログラミングコンテスト，Doc2Vec，抽象構文木，タグ

## 1. まえがき

IT 産業の発展に伴って，プログラミング学習サイトの数が増加している [1]．あるプログラミング学習サイトでは，タグ付けされた問題を解くことでプログラミングの能力判定を行うことができる．例えば，文字列操作のタグが付与された問題を解くことができるが，動的計画法のタグが付与された問題を解けないユーザは，文字列操作が得意で動的計画法が不得意であると判定できる．このような能力判定により，ユーザは自身の苦手な分野を把握することができ，その分野を重点的に練習することで能力を向上させることが期待できる．

また，このサイトの別のサービスとして，教員が作成した問題とテストケースを学生向けに公開し，学生が解くことができるものがある．プログラミング学習サイトの運営会社としては，教員が作成した問題も能力判定に利用することで，能力判定に利用する問題数を増やしたい．しかし，教員が作った問題に社員が 1 つ 1 つタグを付与すると，労力がかかりすぎる．また，問題を作成した教員が自身の尺度でタグ付けすると，タグを付与する基準にばらつきが出てしまい，そのような問題を能力判定に使用した場合に能力判定の精度が落ちてしまうことが考えられる．そのため現在は能力判定に関する問題は社員が作成し，タグを付与している．

そこで本研究では，問題の解答から自動的に問題のタグを付与する手法を提案し，ツールを作成した．このツールが自動的に問題のタグ付けを行うことで，タグ付けの基準は統一されたものとなる．またツールを用いることで，教員が作成した問題に自動でタグ付けが行え，その問題を能力判定に使用することができるようになり，社員が作成した問題においてもタグを付与する労力が軽減される．

提案手法の手順は次の通りである．まず，あらかじめタグが付与されている問題とその解答の集合をデータセットとする．データセットの各問題の解答から抽象構文木 (以下，AST) を作成し，それを後行順探索し，到達したノードを順にリスト化する．次に，作成したリストを Doc2Vec で学習し，分散表現を獲得する．タグ付けを行いたい問題の解答に対しても同様に AST を作成，リスト化して，学習済みモデルから分散表現を獲得する．その分散表現と，データセットから作成した分散表現を比較してコサイン類似度を算出し，コサイン類似度が高いデータセット内の解答を特定する．最後に，特定した解答が提出された問題に付与されているタグから，タグ付けを行いたい問題に対するタグを決定する．

以下，2 節では本研究の背景として，プログラミングコンテストサイト，関連研究と課題について説明する．3 節では提案手法について説明する．4 節では評価実験の手法と結果，用いたデータセットを説明し，実験の結果についての考察を行う．最後に，5 節ではまとめと今後の課題に

[1] 大阪大学
[2] 立命館大学
[3] 南山大学
[a)] k-kawabt@ist.osaka-u.ac.jp

ついて述べる．

## 2. 背景

本節ではプログラミングコンテスト，関連研究とその課題について説明する．

### 2.1 プログラミングコンテスト

プログラミングコンテストとはプログラミングの能力や技術を競い合うコンテストのことである [2]．オンラインジャッジシステム [3] が採用されており，参加者はオンラインで参加する．例えば，AtCoder[*1]，AIZU ONLINE JUDGE[*2](以下，AOJ)，Codeforces[*3]などがある．

本研究では AtCoder を対象とするため，以下 AtCoder について説明する．

#### 2.1.1 AtCoder で定期的に行われるコンテスト

AtCoder では，定期的に行われるコンテストが主に 3 種類あり，易しい方から順に AtCoder Beginner Contest (ABC), AtCoder Regular Contest (ARC), AtCoder Grand Contest (AGC) と名付けられている．

#### 2.1.2 AtCoder Tags

AtCoder Tags[*4]とは，ユーザの投票によって AtCoder の問題にタグ付けをする Web アプリである．タグとは，問題に関する情報や解くために必要な知識などに関するキーワードであり，例えば AtCoder Tags での大分類タグは表 1 に示す通りである．

ユーザは投票画面で AtCoder の問題 ID と大分類，小分類のタグを入力して投票することができ，これまで投票されたタグの割合を確認することができる．例えば，ある問題に対して 3 人が “Mathematics” に投票し，1 人が “String” に投票した場合，その問題は “Mathematics” が 75%，“String” が 25%という結果となる．

### 2.2 関連研究

新濱らの研究 [5] では，Codeforces の問題文同士の類似度を，問題文から算出する手法を提案し，その手法を評価する実験を行っている．

新濱らは Word2Vec[6] を用いる手法と Doc2Vec[7] を用いる 2 つの手法を提案し，その手法を評価する実験を行った．Doc2Vec とは任意の文書の分散表現を獲得する手法であり，学習済みのモデルから未知の文書の分散表現を獲得することができるものである．その結果，Doc2Vec を用いた手法の方が精度が高いことを確認した．また，アルゴリズム情報を基に付与されたタグ同士の問題文は類似度が高いことを推測した．

[*1] https://atcoder.jp/
[*2] https://judge.u-aizu.ac.jp/onlinejudge/
[*3] https://codeforces.com/
[*4] https://atcoder-tags.herokuapp.com/

表 1 AtCoder Tags での大分類タグとその説明 [4]

| タグ名 | 説明 |
|---|---|
| Ad-Hoc | どのカテゴリにも属さない特有の性質を用いる問題 |
| April-Fool | エイプリルフールコンテストなどの特殊なコンテストの問題 |
| Construct | 条件を満たすものを実装する問題 |
| Data-Structure | データ構造に関する問題 |
| Dynamic-Programming | 動的計画法に関する問題 |
| Easy | カテゴリ分類できないほど簡単な問題 |
| Flow-Algorithms | ネットワークフローに関する問題 |
| Game | ゲームに関する問題 |
| Graph | グラフ理論に関する問題 |
| Greedy-Methods | 貪欲法に関する問題 |
| Interactive | 出力の後に入力が与えられる問題 |
| Marathon | 中長期に渡って得点を競い合う問題 |
| Mathematics | 数学に関する問題 |
| Other | 問題以外のページに付与されるタグ |
| Searching | 探索アルゴリズムに関する問題 |
| String | 文字列に関する問題 |
| Technique | 累積和などのテクニックに関する問題 |

### 2.3 課題

新濱らは実験結果から，同じアルゴリズム情報に関するタグが付与されている問題同士の類似度は高くなる可能性があるとしたが，タグが不明な問題に対してタグを付与する実験は行っていない．また，問題文よりも問題に対する解答の方が問題を解くために使われるアルゴリズムの情報などが顕著に出ると考えられるが，問題の解答のソースコードの類似度と問題に付与されているタグの類似度に関する研究は行われていない．

## 3. 提案手法

新濱らは問題同士の類似度を問題文により算出していたが，我々は問題よりも解答の方が問題の特徴が顕著に出ると考えた．そこで本研究では，問題の類似度を解答のソースコードから算出し，その結果からタグ付けを行う．

提案手法によるソースコードへのタグ付け方法を図 1 に示す．タグ付けの流れは大きく 5 つのステップに分けることができる．まず STEP1 として，あらかじめタグが付与された問題の解答のソースコードから AST を作成し，その AST を後行順に探索し，到達したノードを順にリスト化する．STEP2 として，リスト化したものを Doc2Vec で学習を行い，分散表現を獲得する．STEP3 として，タグ付けを行いたい問題の解答において同様に AST を作成，リスト化し，学習モデルから分散表現を獲得する．STEP4 として，STEP2 で獲得した分散表現と STEP3 で獲得した分散表現のコサイン類似度を算出し，コサイン類似度上位 20 位のソースコードを特定する．STEP5 として，STEP4

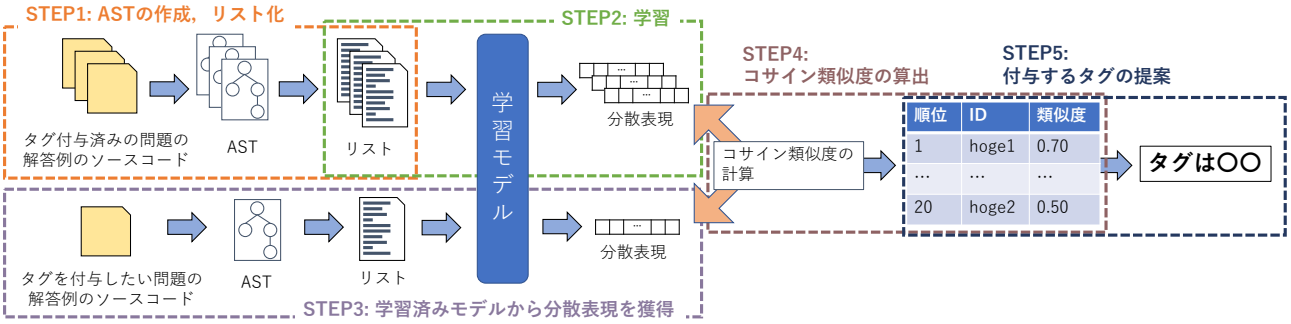


**図 1** タグ付けの流れ

で特定したソースコードが提出された問題に付与しているタグから，問題に付与するタグを推測する．5 つのステップそれぞれの詳細を以下で述べる．

### STEP1: AST の作成，リスト化

あらかじめタグが付与されている問題の解答から AST を作成する．AST の作成には Python ライブラリ Clang python bindings[*5]を使用した．

次に，作成した AST を後行順に探索し，到達したノードを順にリスト化した．この時，以下のルールを適用した．

(1) ライブラリなどの外部ファイル内での処理はリストから除外する．
(2) 呼び出されていない関数を特定しリストから除外する．
(3) ノード名が “UNEXPOSED_EXPR” であるノード[*6]を除外する．
(4) 変数宣言部を除外する．
(5) 外部ライブラリの関数を呼び出しているノードを具体的な関数名に変更する．
(6) 変数参照部は参照されている変数の型名に変更する．

ここでルール 1〜4 を適用した理由は，問題に直接関係ない冗長な箇所を除外するためである．また，ルール 5 の適用により，「外部ライブラリからある関数を参照している」という情報だけでなく，「どの関数を参照しているか」の情報を与えた．また，ルール 6 の適用により，「ある変数の操作を行っている」という情報だけでなく，「どの型の変数の操作を行っているか」の情報を与えた．

AST のリスト化の例を図 2 に示す．図 2 の上部に示す AST を先述したルールを適用してリスト化を行うと，図 2 下部のようになる．

### STEP2: 学習

リスト化したノードを Doc2Vec で学習させ，分散表現を獲得する．Doc2Vec は PV-DM，PV-DBoW の 2 つのアルゴリズムがあるが，PV-DM の方が精度面では優れている

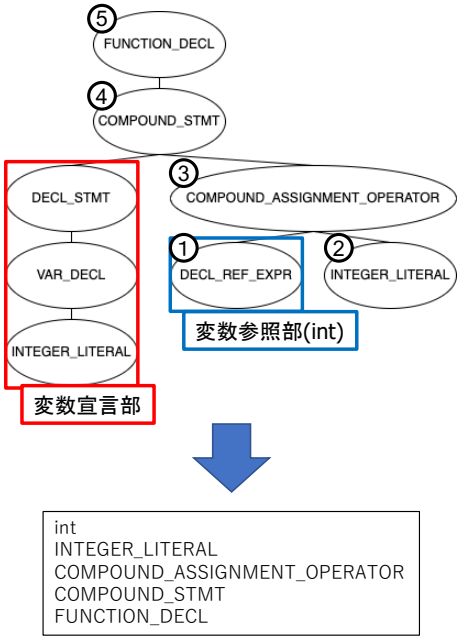


**図 2** AST のリスト化の例

と報告されているため [8]，ここでは PV-DM を採用した．作成したモデルは次元数 300 とし，ウィンドウサイズ 5，出現回数が 5 回以下の単語は棄却，エポック数は 600 として学習を行った．

### STEP3: 学習済みモデルから分散表現を獲得

タグ付けを行いたい問題の解答のソースコードを STEP1 と同様の処理を行ってリスト化する．そして，STEP2 で学習済みのモデルを用いて分散表現を獲得する．

### STEP4: コサイン類似度の算出

STEP2 で獲得した全ての分散表現と STEP3 で獲得した分散表現とを比較し，コサイン類似度を算出する．そして，コサイン類似度上位 20 位までのソースコードが提出された問題を特定する．

### STEP5: 付与するタグの提案

付与するタグの提案を以下の手順で行う．

*5 https://pypi.org/project/clang/
*6 その時点では何のノードか確定できないリテラルが出現したときのノード

**Algorithm 1** タグの付与処理

```
result ← dict()                          ▷ result を辞書型配列で初期化
for all tag ← 全てのタグ do
    result[tag] ← 0
end for
for all code ← コサイン類似度上位 20 位までのソースコード do
    sim ← code のコサイン類似度
    problem ← code が提出されている問題
    for all tag ← problem に付与されているタグ do
        ratio ← 付与されているタグのうち tag が占める割合
        result[tag] ← result[tag] + sim × ratio
    end for
end for
```

(1) STEP4 で特定した問題のタグを取得する．
(2) Algorithm1 に示す処理を行う．
(3) 辞書型配列 *result* の要素の数値により降順にソートし，順位が上のタグから順に付与することを提案する．

## 4. 評価実験

提案手法が推測したタグの精度を確認するために，評価実験を行った．また，問題文を用いた学習と比べて解答のソースコードを用いた学習の方が精度が高いかどうかを確かめる実験を行った．

まず，あらかじめタグが付与されている問題のタグを，その問題の解答を用いて提案手法で推測する．次に，推測したタグと事前に付与されているタグを比較することによって提案手法の精度を調べる．また，タグ毎に推測精度を調べ，推測精度の高いタグと低いタグを特定した．この実験を問題文から学習したモデルでも行い，ソースコードを用いた場合との比較を行った．

本研究では，AtCoder に提出されたソースコードと AtCoder Tags で付与されている大分類タグを使用して，提案手法の評価実験を行った．

本節では，実験に用いたデータセットの作成方法，実験の詳しい手法，結果と考察について説明する．

### 4.1 データセット

#### 4.1.1 AtCoder Tags 上で付与されているタグの収集

AtCoder の全問題に対して 2021 年 10 月 29 日時点での AtCoder Tags で投票されている大分類タグとその割合を収集して，問題 ID と大分類タグの割合との対応を表すテーブルを作成した．

#### 4.1.2 学習用データセット

学習用データセットは，AtCoder と AOJ に提出されているソースコードから構成された IBM の CodeNet[9] のうち，AtCoder に提出されたソースコードの一部を加工して作成した．CodeNet には CodeNet 内の全てのソースコードに関するメタデータが含まれており，このメタデータにはソースコードを記述している言語，問題に正解したかどうか，どのコンテストのどの問題 ID の問題に提出されたかなどの情報が含まれる．問題 ID が取得できることから，AtCoder Tags から収集したデータと組み合わせて問題に付与されているタグも取得することができる．

**表 2** 各データセットにおいて各タグが全体に占める割合

| タグ | 学習用 | 評価用 |
|---|---|---|
| Ad-Hoc | 13.5% | 7.0% |
| Construct | 7.1% | 4.7% |
| Data-Structure | 6.8% | 4.7% |
| Dynamic-Programming | 14.9% | 14.6% |
| Easy | 26.9% | 19.2% |
| Flow-Algorithms | 1.6% | 1.0% |
| Game | 2.7% | 1.9% |
| Geometry | 1.1% | 1.4% |
| Graph | 9.0% | 4.7% |
| Greedy-Methods | 11.2% | 8.0% |
| Mathematics | 19.1% | 28.6% |
| Searching | 17.5% | 10.8% |
| String | 6.0% | 2.8% |
| Technique | 10.5% | 11.7% |

学習用データセットの作成方法について説明する．まず，CodeNet に含まれるソースコードのうち以下の条件に合う問題を選んだ．

- AtCoder に出題された問題
- ABC，ARC または AGC に分類されるコンテストに出題された問題
- 特殊なタグ (“Interactive”, “April-Fool”, “Marathon”, “Other”) が付与されていない問題
- タグが 1 つ以上付与されている問題

次に，Clang ではインクルードできないヘッダファイルを同等の機能をもつ複数のヘッダファイルに置き換えた．

最後に，以下の条件に合うソースコードを各問題から最大 200 個ランダムに選出した．

- 問題に正解しているソースコード
- C++で書かれたソースコード
- Clang Python Bindings に読み込ませた場合にエラーが発生しないソースコード

言語として C++を選んだ理由としては，C++のソースコードが CodeNet に含まれるソースコードの中で一番割合が高かったためである．

選出した問題数は 566，付与されているタグの個数は平均で 1.47，ソースコードの数は 112,442 となった．また，各タグが付与されている問題のソースコード数がソースコード数全体に占める割合を表 2 に示す．

#### 4.1.3 評価用データセット

評価用データセットの作成方法について説明する．

まず，2021 年 6 月 28 日時点までに出題された AtCoder の全問題のうち，以下の条件に合う問題を選出する．

- CodeNet に含まれない問題
- ABC，ARC または AGC に分類されるコンテストに出題された問題
- 特殊なタグ (“Interactive”, “April-Fool”, “Marathon”, “Other”) が付与されていない問題
- タグが 1 つ以上付与されている問題

次に，選出した各問題に 2021 年 6 月 28 日時点までに提出されたソースコードのうち，以下の条件に合うソースコードを 10 個ずつランダムに収集した．

- 正解のソースコード
- C++で書かれたソースコード
- Clang Python Bindings に読み込ませた場合にエラーが発生しないソースコード

最後に，学習用データセットと同様に，Clang ではインクルードできないヘッダファイルを同等の機能をもつ複数のヘッダファイルに置き換えた．

選出した問題数は 213，付与されているタグの個数は平均で 1.21 であり，ソースコードの数は 2130 となった．また，各タグが付与されている問題のソースコード数がソースコード数全体に占める割合を表 2 に示す．

### 4.2 評価実験 1: タグ全体に対する推測精度

評価実験 1 として学習用データセット，評価用データセットのすべての問題で学習，評価を行った．

#### 4.2.1 実験方法

まず，学習用データセットで学習を行う．次に，評価用データセットのソースコードに対して，先述した手法で付与するタグを推測し，これを推測タグと呼ぶ．また，各問題において AtCoder Tags で 1 票でも投票された全てのタグを正解タグと呼ぶ．

そして，すべてのソースコードにおいて Mean Reciprocal Rank(MRR)，正解率，網羅率，Random，タグ毎の macro-precision，タグ毎の macro-recall，タグ毎の macro-F1 を算出する．以下，これらの評価値について説明する．

**Mean Reciprocal Rank**

Mean Reciprocal Rank[10](以下，MRR) の算出方法を説明する．まず，推測タグを上位から順番に見て，そのタグが最初に正解タグに含まれていた場合の順位を $rank$ とする．この時，MRR は以下のようになる．

$$MRR = \frac{1}{rank}$$

ただし，推測タグが正解タグに含まれていない場合，MRR は 0 とする．

**正解率**

推測タグの上位 $n$ 番目までのタグのうち，1 つでも正解タグに含まれていれば推測は正解，含まれていなければ不正解とする．この時の正解率を，$n$ を変化させて算出した．

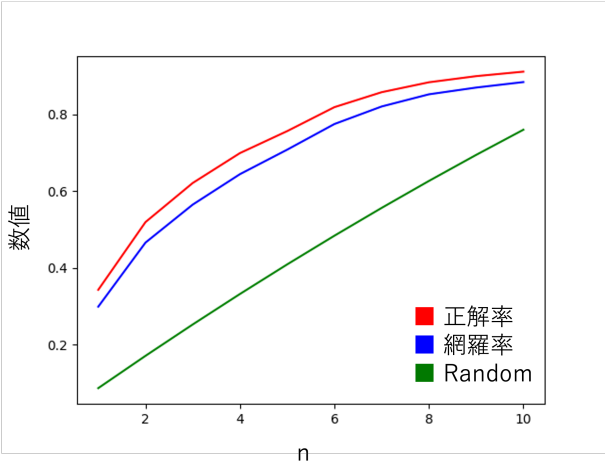


図 3　評価実験 1 で算出した正解率，網羅率，Random

**網羅率**

推測タグの上位 $n$ 番目までのタグのうち，正解タグに含まれるものをカウントしたものを $count_n$，正解タグの個数を $sum_n$ とする．この時の網羅率を 網羅率$_n$ と呼び，以下のように定義する．

$$\text{網羅率}_n = \frac{count_n}{sum_n}$$

これを n を変化させて算出した．

**Random**

ランダムに n 個タグを選び，1 つでも正解タグに含まれている確率を Random と定義し，n を変化させて算出した．

**タグ毎の macro-precision**

この指標の算出においては，推測タグの 1 位のものだけに注目し，このタグを 1 位タグと呼ぶ．あるタグを 1 位タグと予測した場合に，実際にそのタグが 1 位タグである割合を macro-precision とし，これをタグ毎に算出した．

**タグ毎の macro-recall**

この指標の算出においては，macro-precision の算出時と同様に 1 位タグに注目する．あるタグが正解タグに含まれる予測のうち，そのタグが 1 位タグとして予測された割合を macro-recall とし，これをタグ毎に算出した．

**タグ毎の macro-F1**

あるタグにおける macro-precision と macro-recall の調和平均をそのタグの macro-F1 とする．

#### 4.2.2 結果

MRR の平均が 0.52 となり，n を 1〜10 まで変化させた場合の正解率，網羅率，Random は図 3 のようになった．また，タグ毎の macro-precision，macro-recall，macro-F1 とその平均を macro-F1 の降順に表 3 に示す．

図 3 における正解率と Random の比較から，提案手法の方が精度が高いことがわかる．このことから，同じタグがついた問題の解答には共通する特徴が少なからずあることがわかる．また，表 3 から，推測精度が高いタグと低いタグがあることがわかる．

表 3　評価実験 1 で算出した
macro-precision，macro-recall，macro-F1 とその平均

| タグ | macro-precision | macro-recall | macro-F1 |
|---|---|---|---|
| Flow-Algorithms | 0.85 | 0.85 | 0.85 |
| Geometry | 1.00 | 0.50 | 0.67 |
| Mathematics | 0.53 | 0.46 | 0.49 |
| Dynamic-Programming | 0.40 | 0.33 | 0.36 |
| Easy | 0.40 | 0.33 | 0.36 |
| Technique | 0.27 | 0.25 | 0.25 |
| Graph | 0.27 | 0.23 | 0.25 |
| Data-Structure | 0.23 | 0.19 | 0.21 |
| Greedy-Methods | 0.18 | 0.14 | 0.16 |
| Searching | 0.16 | 0.13 | 0.15 |
| Construct | 0.16 | 0.13 | 0.14 |
| String | 0.15 | 0.10 | 0.12 |
| Ad-Hoc | 0.06 | 0.05 | 0.06 |
| Game | 0.04 | 0.03 | 0.04 |
| 平均 | 0.33 | 0.27 | 0.29 |

#### 4.2.3　考察

表 3 から，“Geometry”，“Flow-Algorithms” の macro-F1 が 0.6 以上と高く，“Game”，“Ad-Hoc” の macro-F1 が 0.1 以下と低いことがわかる．このことから，“Geometry”，“Flow-Algorithms” のタグが付与されている問題は解答のソースコードに特徴が出やすく，反対に “Game”，“Ad-Hoc” のタグが付与されている問題は解答のソースコードに特徴が出にくいことが考えられる．また，全体的に，macro-precision の方が macro-recall よりも高くなっている理由としては，タグが複数付与されている問題があるためであると考えられる．

詳しく考察するために，あるタグが推測されている時にどのタグが正解に出現しやすいか，また，あるタグが正解タグに含まれる場合にどのタグが推測されやすいかを調べる．あるタグが n=1 の時の推測タグに含まれる時，一番正解タグに出現しやすいタグを頻出正解タグと呼ぶ．また，あるタグが正解タグに含まれるとき，n=1 の時の推測タグに一番含まれるタグを頻出推測タグと呼ぶ．あるタグに注目した時の頻出正解タグと頻出推測タグ，また，それぞれの出現確率を調べた結果を表 4 に示す．

結果から，多数のタグにおいて頻出正解タグとして “Mathematics” が挙げられていることがわかる．また，多数のタグにおいて頻出推測タグとして “Easy” が挙げられていることがわかる．また，表 2 に示す評価用データセットにおける各タグが占める割合 1 位が “Mathematics”，学習用データセットにおける各タグが占める割合の 1 位が “Easy” であることから，データセットにおける各タグが占める割合に依存していると考えられる．これは，学習モデルがタグごとの特徴を精度良く学習していないためと考えられる．理由として，“Easy”，“Ad-Hoc”，“Technique” などの，アルゴリズム情報に関係のない抽象的なタグが含まれているためと考えられる．そのため，そのようなタグを除くことで推測精度が上がると考えられる．

### 4.3　評価実験 2: タグを絞った場合の推測精度

評価実験 2 として評価実験 1 で使用した学習用データセット，評価用データセットの問題のうち，アルゴリズム情報に関するタグが付与されている問題のみで学習，評価を行った．具体的には “Dynamic-Programming”，“Flow-Algorithms”，“Greedy-Methods”，“Mathematics”，“Searching” の 5 つのタグが付与されている問題を用いた．

#### 4.3.1　学習用データセット

評価実験 1 で使用した学習用データセットのうち，付与されているタグがアルゴリズム情報に関するタグのみで構成されている問題を用いる．問題数は 172，付与されているタグの個数は平均で 1.22，ソースコードの数は 33,910 となった．また，各タグが付与されている問題のソースコード数がソースコード数全体に占める割合を表 5 に示す．

#### 4.3.2　評価用データセット

学習用データセットと同様に，評価実験 1 で使用した評価用データセットのうち，付与されているタグがアルゴリズム情報に関するタグのみで構成されている問題を用いる．問題の数は 96 問，付与されているタグの個数は平均で 1.17 であり，ソースコードの数は 960 となった．また，各タグが付与されている問題のソースコード数がソースコード数全体に占める割合を表 5 に示す．

#### 4.3.3　実験方法

評価実験 1 と同様に，まず学習用データセットで学習を行う．次に，評価用データセットのソースコードに対して，評価実験 1 で述べた評価値をすべて算出する．

#### 4.3.4　結果

MRR の平均が 0.69 となり，n を 1〜5 まで変化させた場合の正解率，網羅率，Random は図 4 のようになった．また，タグ毎の macro-precision，macro-recall，macro-F1 とその平均を macro-F1 の降順に表 6 に示す．

結果から，評価実験 1 の結果と比較して全体的に精度が上がったことがわかる．また，表 6 から，タグを絞った場合でも，推測精度の高いタグと低いタグがあることが確認できる．

#### 4.3.5　考察

評価実験 1 の結果と比較して精度が上がった理由として，アルゴリズム情報に関するタグが付与された問題の解答を使うことで，解答の特徴が顕著になりタグを付与する精度が上がったと考えられるが，単にタグの種類が減ったためであるとも考えられるため，以下，詳しく考察を行う．

表 6 から，“Flow-Algorithms” の macro-F1 が 0.80 と高く，“Greedy-Methods” の macro-F1 が 0.23 と低いことが分かる．評価実験 1 と同様に，頻出正解タグ，頻出推測タ

表 4　評価実験 1 における出現しやすいタグとその確率

| タグ | 頻出正解タグ | 頻出正解タグの出現確率 | 頻出推測タグ | 頻出推測タグの出現確率 |
|---|---|---|---|---|
| Ad-Hoc | Mathematics | 17.5% | Easy | 40.1% |
| Construct | Greedy-Methods | 17.6% | Easy | 39.0% |
| Data-Structure | Technique | 20.8% | Easy | 22.0% |
| Dynamic-Programming | Dynamic-Programming | 32.7% | Mathematics | 23.2% |
| Easy | Easy | 32.6% | Easy | 72.9% |
| Flow-Algorithms | Flow-Algorithms | 84.6% | Flow-Algorithms | 55.0% |
| Game | Mathematics | 41.9% | Easy | 30.0% |
| Geometry | Mathematics | 50.0% | Easy | 63.3% |
| Graph | Graph | 23.4% | Graph | 26.0% |
| Greedy-Methods | Technique | 22.0% | Easy | 25.9% |
| Mathematics | Mathematics | 45.7% | Mathematics | 35.7% |
| Searching | Mathematics | 16.1% | Easy | 30.4% |
| String | Greedy-Methods | 18.6% | Easy | 65.0% |
| Technique | Mathematics | 34.0% | Easy | 22.8% |

表 5　評価実験 2 の各データセットにおいて各タグが全体に占める割合

| タグ | 学習用 | 評価用 |
|---|---|---|
| Dynamic-Programming | 37.6% | 22.9% |
| Flow-Algorithms | 3.5% | 2.1% |
| Greedy-Methods | 17.1% | 11.5% |
| Mathematics | 35.7% | 56.2% |
| Searching | 28.5% | 24.0% |

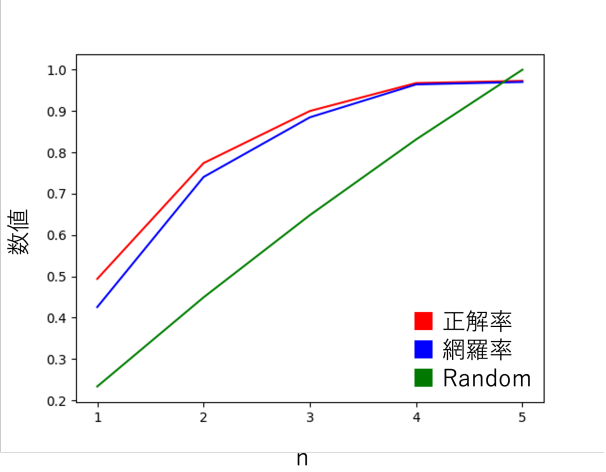


図 4　評価実験 2 で算出した正解率，網羅率，Random

表 6　評価実験 2 で算出した macro-precision，macro-recall，macro-F1 とその平均

| タグ | macro-precision | macro-recall | macro-F1 |
|---|---|---|---|
| Flow-Algorithms | 0.80 | 0.80 | 0.80 |
| Mathematics | 0.73 | 0.63 | 0.68 |
| Dynamic-Programming | 0.36 | 0.30 | 0.33 |
| Searching | 0.34 | 0.29 | 0.31 |
| Greedy-Methods | 0.25 | 0.21 | 0.23 |
| 平均 | 0.50 | 0.45 | 0.47 |

グを定義し，あるタグに注目したときの頻出正解タグ，頻出推測タグの出現確率を調べた．結果を表 7 に示す．

結果から，“Greedy-Methods” が推測タグに含まれる場合に一番正解タグに出現しやすいタグが “Searching” で，かつ “Greedy-Methods” が正解タグに含まれる場合に一番推測タグに出現しやすいタグが “Searching” であることから，“Greedy-Methods” が付与されている問題のソースコードと，“Searching” が付与されている問題のソースコードは共通する特徴を持つために分類が難しく，このことから “Greedy-Methods” の macro-F1 が低くなったと考えられる．また，評価実験 1 と比較して，表 5 に示すデータセットにおける各タグが占める割合に依存している度合いが低いことから，アルゴリズム情報に関するタグが付与された問題の解答を使うことで学習の精度が上がったと考えられる．

### 4.4　評価実験 3: 問題文から学習したモデルとの比較

3 節で我々が仮定した，問題よりも解答を用いて学習を行った方が精度の高い予測を行えることを確認するために，問題文から学習したモデルと解答のソースコードから学習したモデルの精度を比較した．

#### 4.4.1　実験方法

評価実験 1 で用いたデータセットをデータセット 1，評価実験 2 で用いたデータセットをデータセット 2 と呼ぶ．まず各データセットに含まれる問題の英語で書かれた問題文を Python ライブラリ nltk を用いて単語で分割し，ストップワード，記号を除去したリストを作成した．そして，それらを先述した実験方法と同様の方法で実験し，macro-Recall の平均，macro-Precision の平均，macro-F1 の平均をそれぞれ算出した．評価実験 3 では AST を作成せず，作成したリストをそのまま学習に利用した．

#### 4.4.2　結果

実験結果を表 8 に示す．結果と表 3，表 6 を比較すると，

表 7　評価実験 2 における出現しやすいタグとその確率

| タグ | 頻出正解タグ | 頻出正解タグの出現確率 | 頻出推測タグ | 頻出推測タグの出現確率 |
|---|---|---|---|---|
| Dynamic-Programming | Mathematics | 45.5% | Dynamic-Programming | 43.2% |
| Flow-Algorithms | Flow-Algorithms | 80.0% | Flow-Algorithms | 60.0% |
| Greedy-Methods | Searching | 39.3% | Searching | 30.9% |
| Mathematics | Mathematics | 63.2% | Mathematics | 48.9% |
| Searching | Mathematics | 37.3% | Searching | 29.6% |

表 8　評価実験 3 の結果

| 評価指標 | データセット 1 | データセット 2 |
|---|---|---|
| macro-Recall の平均 | 0.22 | 0.26 |
| macro-Precision の平均 | 0.31 | 0.30 |
| macro-F1 の平均 | 0.32 | 0.37 |

データセット 1 における macro-F1 以外はソースコードを学習に使用したモデルの方が高いことが確認できる．

#### 4.4.3　考察

データセット 1 においてソースコードを学習に使用したモデルの方が macro-F1 が低かった理由として，ソースコードを学習に使用したモデルにおいてもタグごとの特徴を精度よく学習しておらず，大きな差が出なかったと考えられる．また，データセット 2 においてはソースコードを使用したモデルの方が全ての指標において明らかに高いことが確認できるため，アルゴリズム情報に関するタグを付与する精度はソースコードを用いたモデルの方が精度が良いことがわかる．

## 5.　まとめ

本研究では，Doc2Vec とプログラミング演習問題の解答を用いて，プログラミング演習問題にタグを付与する手法を提案した．この手法の精度を評価するための実験として，あらかじめタグが付与されている問題に対して，提案手法を用いて付与したタグと，あらかじめ付与されているタグを比較する実験を行った．また，問題文を用いた学習よりもソースコードを用いた学習の方が精度が高いことを確かめる実験を行った．

評価実験の結果として，高い精度で付与できるタグと，付与の精度が低くなってしまうタグが存在することが確認でき，高い精度で付与できるタグにおいては，14 個のタグのうち 1 つのタグを付与する時，macro-F1 の値が 0.85 の精度で付与できることが確認できた．また，付与するタグをアルゴリズム情報に関するタグに絞ることで，タグを付与する精度を高めることができた．また，アルゴリズム情報に関するタグを付与する精度は問題文を学習したモデルよりもソースコードを学習したモデルの方が精度がいいことが確認できた．

今後の課題として，問題文や問題の入力例・出力例などの，解答のソースコード以外の情報を学習に使用することである．新濱らの研究 [5] から，同一のアルゴリズム情報に関するタグが付与されている問題文は類似度が高くなることが確認されているため，解答のソースコードと問題文を学習に使用することで，付与するタグの精度が上がると考えられる．2 つ目は，Doc2Vec 以外の学習方法でも同様の実験を行い，より精度の良い学習方法を模索することである．

**謝辞**　本研究は JSPS 科研費 18H04094 の助成を受けたものです．

### 参考文献

[1] 【2021 年版】おすすめのプログラミング学習サイト 20 選！各サイトを徹底比較 - テックキャンプブログ. https://tech-camp.in/note/technology/87876/
[2] Ebtekar and Paul Liu. An elo-like system for massive multiplayer competitions, *arXiv preprint arXiv:2101.00400*, 2021.
[3] 渡部有隆. 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.
[4] CATEGORY - AtCoder Tags https://atcoder-tags.herokuapp.com/explain
[5] 新濱遼大, 槇原絵里奈, 小野景子, 幾島直哉, 山川蒼平. オンラインジャッジシステムにおける問題文の類似度調査. 情報処理学会研究報告, Vol.2021-SE-209, No.9, pp.1–7, 2021 年 11 月.
[6] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems-Volume2*, NIPS'13, pp.3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.
[7] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In Eric P. Xing and Tony Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, Vol.32 of *Proceedings of Machine Learning Research*, pp.1188–1196, Bejing, China, 22–24 Jun 2014. PMLR.
[8] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.
[9] Ruchir Puri, David Kung, Geert Janssen, Wei Zhang, Giacomo Domeniconi, Vladmir Zolotov, Julian Dolby, Jie Chen, Mihir Choudhury, Lindsey Decker, Veronika Thost, Luca Buratti, Saurabh Pujar, Shyam Ramji, Ulrich Finkler, Susan Malaika, and Frederick Reiss. Codenet: A large-scale ai for code dataset for learning a diversity of coding tasks. *arXiv preprint arXiv:2105.12655*, 2021.
[10] Nick Craswell. *Mean Reciprocal Rank*, pp.1703–1703. Springer US, Boston, MA, 2009.