特別研究報告

題目

Doc2Vec を用いたプログラミング演習問題のタグ付け手法の提案

指導教員
井上 克郎 教授

報告者
川渕 皓太

令和 4 年 2 月 8 日

大阪大学 基礎工学部 情報科学科

令和 3 年度 特別研究報告

Doc2Vec を用いたプログラミング演習問題のタグ付け手法の提案

川渕 皓太

内容梗概

IT 産業の発展に伴い，現在，プログラミング学習サイトの数が増加している．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ユーザのプログラミングに関する能力判定を行うことができる．既存のプログラミング演習問題をこの能力判定に使いたいが，問題のタグ付けに労力がかかることが問題となる．また，能力判定の精度を担保するためには，統一した基準でタグ付けを行う必要がある．

そこで本研究では，既存のプログラミングの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．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ソースコード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．その後，リストを Doc2Vec で学習し，分散表現を獲得する．次に，タグ付けを行いたい問題の解答を同様にリスト化し，学習済みモデルから分散表現を獲得する．得られた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学習用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．

評価実験として，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．実験結果から，タグによって高い精度で付与できるものと付与の精度が低くなってしまうものがあり，精度が高いタグでは macro-F1 が 0.85 と，高い精度で推測できることがわかった．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が確認できた．

主な用語

プログラミングコンテスト
Doc2Vec
抽象構文木
タグ

目 次

1　まえがき

IT 産業の発展に伴い，現在，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難易度の高い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難易度の低い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，ユーザが自身の苦手な分野を把握することができ，能力が向上す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．このサービスは現在，多くの教育の場で利用されている．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．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 と呼ぶ) 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おいても同様に 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 章では本研究の背景として，オンラインジャッジシステム，プログラミングコンテストサイト，Doc2Vec，関連研究と課題について説明する．3 章では提案手法について説明する．4 章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 章ではまとめと今後の課題について述べる．

2　背景

本章では，本研究の背景としてオンラインジャッジシステム，プログラミングコンテスト，Doc2Vec，関連研究とその課題について説明する．

2.1　オンラインジャッジシステム

オンラインジャッジシステムとは，ユーザから提出されたプログラムをコンパイル・実行し，厳格な検証データや検証器を用いて，その正確さと性能を自動評価しフィードバックするサービスである．一般的に C/C++や Java 等のさまざまなプログラミング言語に対応し，24 時間インターネットを通して利用することができる [8]．一般的には web サイト上でプログラムの提出，検証を行うことができる．

AIZU ONLINE JUDGE[1](以下，AOJ と呼ぶ) での “ITP1_1_A: Hello World” の問題[2]の例を挙げる．AOJ では図 1 のように問題が与えられる．図 1 中の右上に赤く示したアイコンをクリックすることで図 2 のようなソースコード提出画面に遷移する．提出画面に問題の解答となるソースコードと，そのソースコードの言語を入力して，図 2 の下部に赤く示した “Submit” ボタンをクリックすることで，記述したソースコードがコンパイル・実行されて検証される．検証結果は図 3 のように表示され，結果から，用意されたテストケースを通過し，問題に正解したことがわかる．

2.2　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 [3]．オンラインジャッジシステムが採用されており，参加者はオンラインで参加する．定められた制限時間内に，同時に複数の参加者が同じ問題を解き，解答ソースコードを提出する．コンテストの終了後，提出したソースコードの正誤や解答時間によって参加者の順位が決定し，レーティングが変動する．例えば，国内では AtCoder[3]，AOJ，海外では Codeforces[4]，Topcoder[5]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．

[1] https://judge.u-aizu.ac.jp/onlinejudge/
[2] https://judge.u-aizu.ac.jp/onlinejudge/description.jsp?id=ITP1_1_A
[3] https://atcoder.jp
[4] https://codeforces.com
[5] https://www.topcoder.com

図 1: AOJ における問題画面

図 2: AOJ におけるソースコード提出画面

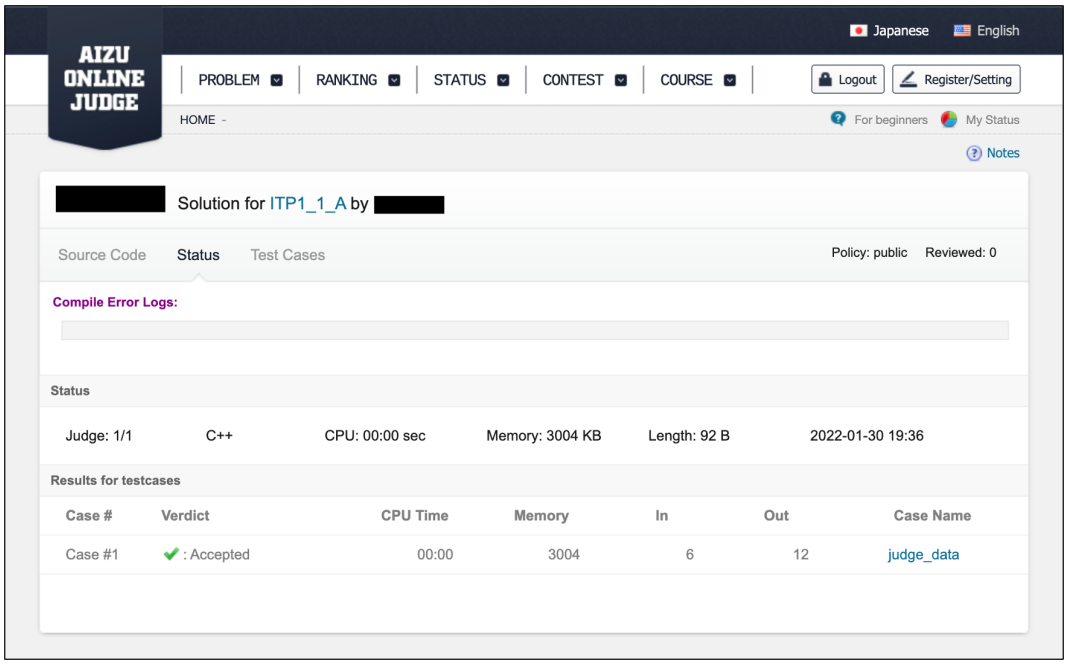

図 3: AOJ における提出したソースコードの検証結果画面

2.2.1 AtCoder

本節では，AtCoder におけるコンテストの流れ，コンテストの分類，ユーザ数について述べる．

まず，AtCoder におけるコンテストの流れについて説明する．AtCoder における一般的なコンテストの流れは以下の通りである．

1. 開始時間までに参加希望者は参加登録を行う．
2. 開始時間になると参加者は問題の閲覧と回答の提出が可能になる．
3. 問題を解き，作成した解答のソースコードをオンラインで提出すると，提出した時点でオンラインジャッジシステムによる正誤判定が行われる．与えられたテストケース全てに正解すると問題に応じた点数が加算され，不正解の場合は誤答の回数が 1 加算される．
4. 終了時間になると，その時点までに獲得した点数の合計で順位が決まり，順位に応じてレーティングが変動する．点数が同じ場合は 最終解答時間 + (誤答の回数) × 5 分が早いほど順位が上となる．

コンテスト中，問題を解く順番は指定されておらず，使用するプログラミング言語も自由に選択することができる．また，終了したコンテストの問題を閲覧，解答の提出を行うことも可能であるため，ユーザはいつでも学習を行うことができる．

次に，AtCoder のコンテストの分類について説明する．定期的に行われるコンテストは主に以下の 3 種類がある．

- AtCoder Beginner Contest(ABC): 初心者，初級者向けコンテスト
- AtCoder Regular Contest(ARC): 中級者，上級者向けコンテスト
- AtCoder Grand Contest(AGC): 上級者向けコンテスト

次に，AtCoder のユーザ数について説明する．AtCoder では 2 年以内にレーティングが変動するコンテストに参加したユーザをアクティブユーザとして定義しており，2022 年 1 月 11 日現在，アクティブユーザは 92754 人となっている．

2.2.2 AtCoder Tags

AtCoder Tags[6]とは，AtCoder の問題にタグ付けをすることを目標にした Web アプリである．2022 年 1 月 11 日現在，全問題のうち約 74%の問題がタグ付けされている．ユーザ

[6]https://atcoder-tags.herokuapp.com

は投票画面で AtCoder の問題 ID と大分類，小分類のタグを入力することで投票することができ，この投票数で問題のタグが決定される．大分類は 18 種類，小分類は 104 種類のタグがあり，本研究では大分類タグを対象とした．大分類の内訳を表 1 に示す．また，1 つの問題が複数のタグを持つこともある．例えば，あるユーザ A が，問題 ID が hoge200 の問題の大分類が “String” であると投票し，別のユーザ B，ユーザ C，ユーザ D がその問題の大分類が “Mathematics” であると投票した場合，問題 ID が hoge200 の問題の大分類タグは “Mathematics”，“String” となり，各タグが全体に占める割合は “Mathematics” が 75%，“String” が 25%となる．

表 1: AtCoder Tags での大分類タグとその説明

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 Doc2Vec

Doc2Vec とは，任意の文書の分散表現を獲得する手法であり，PV-DM(Distributed Memory Model of Paragraph Vectors)，PV-DBoW(Distributed Bag of Words version of Paragraph Vector) の 2 つのアルゴリズムを総称したものである [4]．PV-DM は，以下の手順を

繰り返して文書の分散表現を獲得する．

1. 文書 ID と，文書の中から一部サンプリングした単語を結合して入力とする．
2. 入力から次に出現する単語を予測する．
3. 中間層，出力層の重みを更新する．

また，PV-DBoW は以下の手順を繰り返して文書の分散表現を獲得する．

1. ある文書から任意の数の単語をサンプリングする．
2. 文書 ID を入力として，サンプリングした単語を出力するように中間層，出力層の重みを更新する．

PV-DBoW の方がシンプルでメモリの使用量が少なく，効率的に計算可能であるが，単語の語順を考慮しないため，PV-DM の方が精度面では優れていると報告されている [5]．また，Doc2Vec では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．

2.4 新濱らの研究

新濱らの研究 [9] では，Codeforces の問題文同士の類似度を算出する手法を提案している．本節では，新濱らが提案した手法，手法を評価する実験と結果について述べる．

まず手法について説明する．新濱らは 2 つの手法で Codeforces の問題同士の類似度を算出した．まず 1 つ目の手法として，新濱らは Word2Vec を用いた手法を提案した．Word2Vec とは，深層学習によって単語の分散表現を獲得する手法である [6]．Word2Vec を用いた手法の手順は以下の通りである．

1. Wikipedia から収集した英単語に対してクリーニング処理を施したコーパスのである text8 を使用して Word2Vec のモデルを作成する．作成したモデルの次元数は 300 で，学習時のウィンドウサイズは 5，訓練回数は 10 回，5 回未満しか登場しない単語は棄却し，その他のパラメータに関してはデフォルト値を使用した．
2. データセットに含まれる単語の正規化を行い，数値，不等号，演算子などの記号を削除する．また，自然言語処理のツールキットである NLTK[7]を用いてストップワードを除去した．
3. 作成したモデルから問題文に含まれる全単語の分散表現を獲得する．Word2Vec は未知の単語の分散表現を獲得できないため，既知の単語の分散表現のみを獲得した．

[7] https://www.nltk.org

4. Simple Word-Embedding-based Models(以下，SWEM) という手法を使用して，獲得した分散表現を基に文章全体の分散表現を獲得する．SWEM とは，単語の分散表現から文章の分散表現を獲得する手法であり，新濱らは SWEM で提案されている手法のうち SWEM-aver という文章に含まれる単語の分散表現に対して average pooring を行う手法を使用した．

5. 問題文の組み合わせに対してコサイン類似度を算出することで問題文間の類似度を算出する．

次に 2 つ目の手法として，新濱らは Doc2Vec を用いた手法を提案した．Doc2Vec を用いた手法の手順は以下の通りである．

1. データセットから次元数 64 のモデルを作成する．手法は PV-DM を使用し，エポック数は 20，その他のパラメータの値はデフォルト値を使用した．

2. 作成したモデルから問題文の分散表現を獲得し，問題文間のコサイン類似度を算出する．

次に，実験と結果について述べる．提案した手法を評価する実験として，新濱らは 3 つの実験を行った．

まず 1 つ目の実験として，ランダムな問題の組み合わせに対しての類似度を 2 つの手法で算出した．まずランダムに抽出した 50 問のデータに対して 2 つの手法で問題文間の類似度を算出し，比較した．また，5 つの問題文の組み合わせに対して 2 つの手法により算出した類似度を比較した．結果として，Word2Vec を用いた手法の方が類似度が高くなる傾向があることがわかった．これは，問題文に共通する単語が多く出現することに起因すると考えられる．また，Word2Vec のモデル作成において，wikipedia から取得したデータセットを使用したために，プログラミング課題特有の単語に対して適切な分散表現を獲得できなかったと考えられる．また，Doc2Vec を用いた手法が Word2Vec を用いた手法と比較して，算出した類似度が低くなる理由として，Doc2Vec は数字や記号を考慮して類似度を算出したため，より正確に類似度を算出した可能性が考えられる．

2 つ目に，タグごとに抽出した問題の類似度を 2 つの手法で算出した．Codeforces の各問題にはタグがついており，このタグは 1 問あたり複数付与されている場合がある．新濱らはタグごとにランダムで 50 問ずつ抽出したデータに対して，問題文間の類似度を 2 つの手法で算出した．対象とするタグは付与された回数が多い “implemention”，“math”，“greedy”，“dp”，“data structures” とした．結果として，手法の差異に関係なく，“implemention”，“math” はデータのばらつきが大きく，“greedy” と “dp” はデータのばらつきが小さいことがわかった．“implemention”，“math” は問題のカテゴリを表すタグであるのに対して，“greedy” と

“dp” はアルゴリズム情報を表していることから，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可能性が考えられる．

3 つ目に，それぞれの手法で類似度が高い問題の組み合わせ 5 組と低い問題の組み合わせ 5 組に対して，類似しているかどうかを 5 段階で評価する被験者実験を行った．評価基準は 2 つの問題が似たような知識で解けるかとした．被験者はプログラミング経験が 3 年以上ある 20 代学生 10 名で，実際のオンラインジャッジシステム利用環境を想定して入力条件や出力条件を配布した．また，同様の理由から，実験中にブラウザ検索を可能とした．結果として，算出した類似度が高い組み合わせは，算出した類似度が低い組み合わせと比較して，被験者実験による類似度が高いことがわかった．また，Doc2Vec を用いた手法では特にその傾向が見られたことから，Doc2Vec を用いた手法の方が精度が高いと考えられる．

2.5 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まだ行われていない．

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，問題文よりも解答の方が問題の特徴が顕著に出ると考えたため，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．

まず，提案手法の流れについて説明する．ソースコードのタグ付けの流れを図 4 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．そして 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 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．

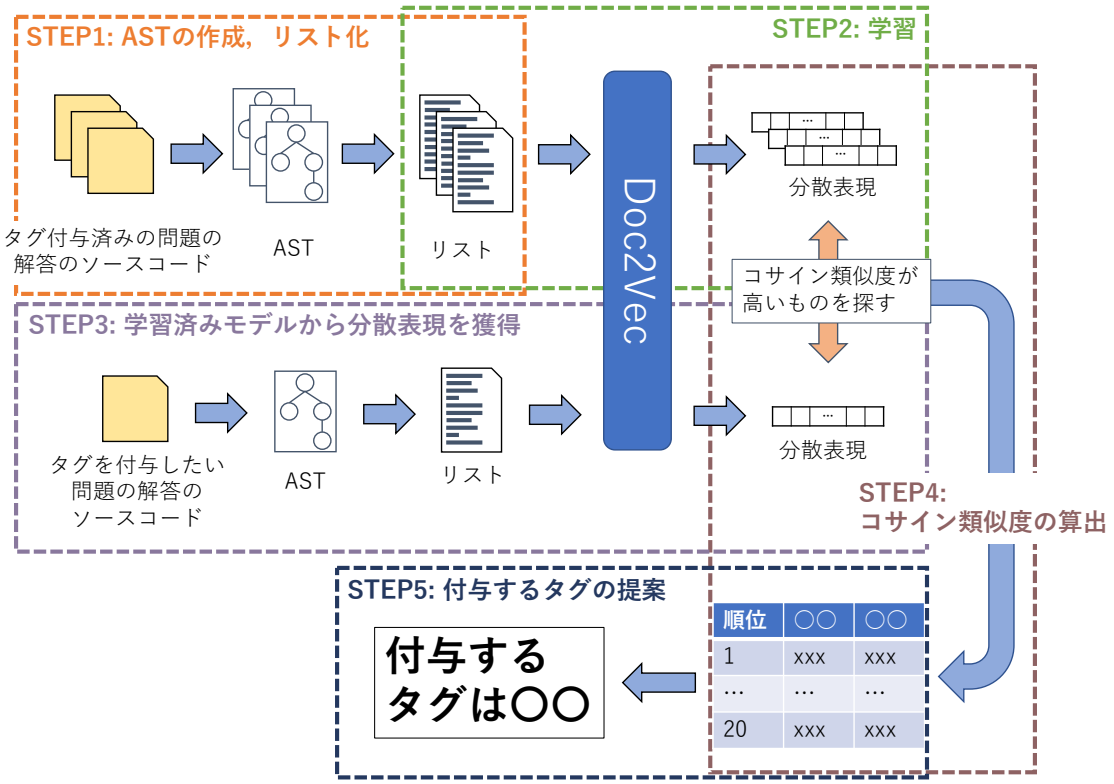

図 4: タグ付けの流れ

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複数の解答から AST を作成する．AST の作成には python ライブラリ Clang python bindings 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を参照している部分はリストから除外する．
2. 使用していない関数をリストから除外する．
3. ノード名が “UNEXPOSED_EXPR” であるノードを除外する．
4. 変数宣言部を除外する．
5. 外部ライブラリから使用した関数は具体的な関数名を表示する．
6. 変数参照部は変数の型名を表示する．

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なっている」という情報だけでなく,「どの型の変数の操作を行なっているか」の情報を与えた．

使用していない関数の特定方法を説明する．まず，関数の呼び出し関係を表す木の集合である “call_tree” を作成する．例として，関数 main 内で関数 A を呼び出し，関数 A 内で関数 B と関数 C を呼び出し，関数 C 内で関数 D と関数 C を呼び出し，関数 E 内で関数 A と関数 F を呼び出した時の “call_tree” は図 5 のようになる．そして，“call_tree” を関数 main からたどり，呼び出された関数と呼び出されていない関数を特定する．図 5 の “call_tree” においてこの処理を行い，呼び出された関数を赤色，呼び出されていない関数を青色で示すと図 6 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．使用したアルゴリズムは PV-DM で，python の機械学習ライブラリの gensim を用いて学習を行った．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．

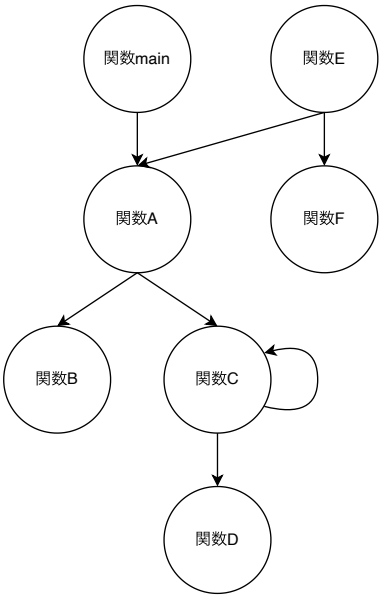

図 5: “call_tree” の例

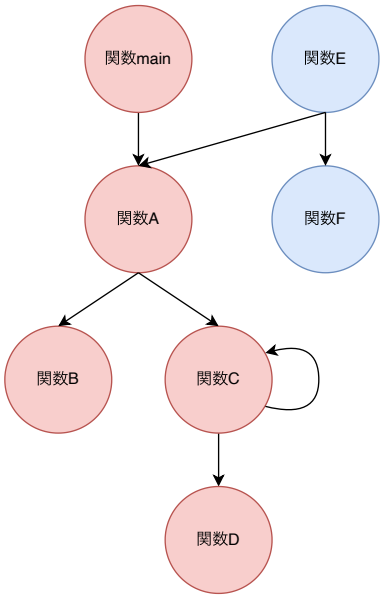

図 6: 呼び出された関数を特定した例

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 と同様の処理を行なってリスト化する．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

STEP3 で獲得した分散表現と，STEP2 で獲得した全ての分散表現を比較し，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は以下の手順で行う．

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．

例として，表 2，表 3 に対して STEP5 の処理を行うと，以下の計算が順に行われる．

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全体のうち tag が占める割合
        result[tag] ← result[tag] + sim × ratio
    end for
end for
```

1. $result$[“String”]$+= 0.6 \times 0.5$，$result$[“Game”]$+= 0.6 \times 0.5$
2. $result$[“Graph”]$+= 0.5 \times 1.0$
3. $result$[“String”]$+= 0.4 \times 1.0$

よって，辞書型配列 result は表 4 のようになり，“String”，“Graph”，“Game” の順で付与するタグを提案する．

表 2: コサイン類似度と問題番号の例

類似度順位	問題番号	コサイン類似度
1	abc300_a	0.6
2	abc301_b	0.5
3	abc299_a	0.4

表 3: 問題番号とタグの組み合わせの例

問題番号	タグと割合
abc299_a	String(100%)
abc300_a	String(50%), Game(50%)
abc301_b	Graph(100%)

表 4: 提案するタグの例

提案順位	キー	要素
1	Strings	0.7
2	Graph	0.5
3	Game	0.3

4 評価実験

提案手法が推測したタグの精度と，個々のタグ毎の推測精度を確認するために評価実験を行った．

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

本章では，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 の全問題の 2021 年 10 月 29 日時点での AtCoder Tags で付与されているタグを収集した．この時，付与されている全てのタグを収集し，各タグが全体に占める割合も収集した．この収集したデータにより，AtCoder の問題の問題 ID からその問題のタグを取得することができる．

4.1.2 基本学習用データセット

各実験では，本節で説明する基本学習用データセットを加工した学習用データセットを使用する．基本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[7] 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．以下，CodeNet の説明と基本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 の統計情報を表 5，CodeNet の言語分布を図 7，CodeNet に含まれるソースコードのステータスの分類を図 8 に示す．また，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，このメタデータから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取得できる．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 5: CodeNet の統計情報

問題数	ソースコード数	言語数	コード元
4,053	13,916,868	55	AtCoder，AOJ

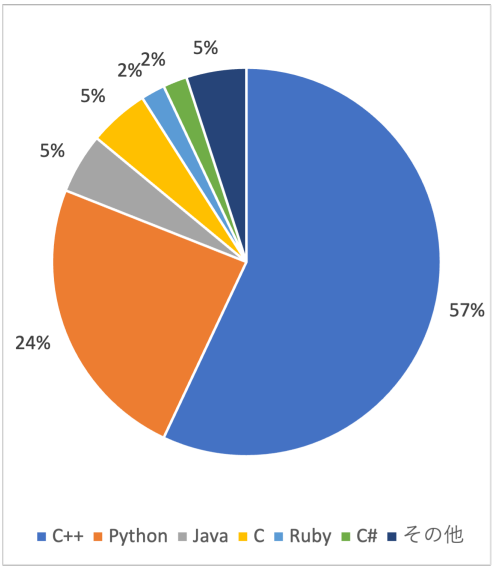

図 7: CodeNet の言語分布

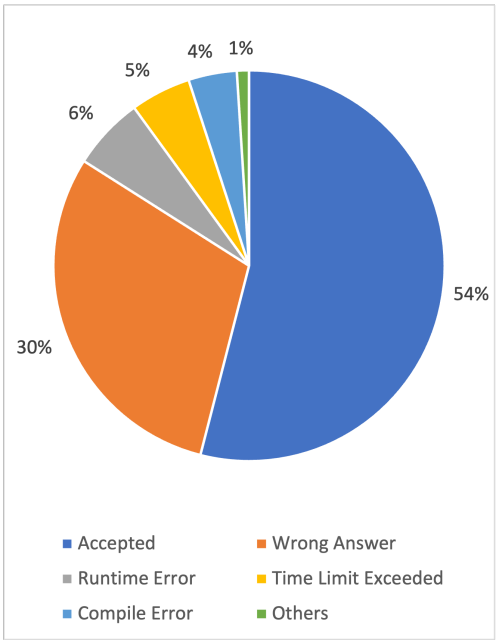

図 8: CodeNet のステータス分類

基本学習用データセットの作成について説明する．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

次に，“bits/stdc++.h” という GCC のみでしかインクルードできないヘッダファイルをインクルードしているソースコードを Clang Python Bindings に読み込ませるとエラーになるため，そのようなソースコードにおいては，該当部分を削除し，同等の機能を持つ複数のヘッダファイルをインクルードするように変更した．

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，図7に示すように，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．

このようにして作成した基本学習用データセットを各実験で加工して使用した．

4.1.3 基本評価用データセット

各実験では，基本評価用データセットを加工した評価用データセットを使用する．基本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

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

この時，基本学習用データセットと同様に，収集したソースコードが “bits/stdc++.h” をインクルードしている場合には，同等の機能を持つ複数のヘッダファイルをインクルードするように変更した．また，収集時にソースコードが提出されている問題の問題 ID を取得し，この問題 ID と AtCoder Tags から取得したデータを使用することで，問題に付与されているタグを取得することができるようにした．

このようにして作成した基本評価用データセットを各実験で加工して使用した．

4.2 評価実験 1: タグ全体に対する推測精度

評価実験 1 として基本学習用データセット，基本評価用データセットの問題のうち，特殊なタグが付与されている問題を除いたすべての問題で学習，評価を行った．特殊なタグとは，

具体的には “Interactive”，“April-Fool”，“Marathon”，“Other” のことであり，これらのタグを全てのタグから除いた 14 個のタグの一覧を表 6 に示す．

以下，本節では学習用データセットの作成，評価用データセットの作成，実験方法，結果と考察について述べる．

表 6: 評価実験 1 で使用した問題のタグ一覧

Ad-Hoc	Construct	Data-Structure	Dynamic-Programming
Easy	Flow-Algorithms	Game	Geometry
Graph	Greedy-Methods	Mathematics	Searching
String	Technique		

4.2.1 学習用データセット

基本学習用データセットのうち，AtCoder Tags でタグが 1 つ以上付与されている問題で，かつ付与されているタグが表 6 に示すタグのみで構成されている問題を選出する．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降順に並べたものを表 7 に示す．合計して 100%を超えるのは 1 つの問題にタグが複数付与されている場合があるためである．

4.2.2 評価用データセット

学習用データセットと同様に，AtCoder Tags でタグが 1 つ以上付与されている問題で，かつ付与されているタグが表 6 に示すタグのみで構成されている問題を選出する．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降順に並べたものを表 8 に示す．合計して 100%を超えるのは学習用データセットの場合と同様に，1 つの問題にタグが複数付与されている場合があるためである．

4.2.3 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，各問題の AtCoder Tags で付与されているタグを取得し，これを正解タグと呼ぶ．正解タグは複数存在する場合もある．

そして，すべてのソースコードにおいて以下の評価値を算出する．

表 7: 評価実験 1 の学習用データセットにおいて各タグが全体に占める割合

タグ	割合
Easy	26.9%
Mathematics	19.1%
Searching	17.5%
Dynamic-Programming	14.9%
Ad-Hoc	13.5%
Greedy-Methods	11.2%
Technique	10.5%
Graph	9.0%
Construct	7.1%
Data-Structure	6.8%
String	6.0%
Game	2.7%
Flow-Algorithms	1.6%
Geometry	1.1%

表 8: 評価実験 1 の評価用データセットにおいて各タグが全体に占める割合

タグ	割合
Mathematics	28.6%
Easy	19.2%
Dynamic-Programming	14.6%
Technique	11.7%
Searching	10.8%
Greedy-Methods	8.0%
Ad-Hoc	7.0%
Graph	4.7%
Construct	4.7%
Data-Structure	4.7%
String	2.8%
Game	1.9%
Geometry	1.4%
Flow-Algorithms	1.0%

Mean Reciprocal Rank

Mean Reciprocal Rank[2](以下，MRR と呼ぶ) 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．

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

例えば，推測タグの 1 位が “Game”，2 位が “String” であり，正解タグが “String” と “Graph” である場合，$MRR = \frac{1}{2}$ となる．適切なタグを推測タグの上位に提案しているとき，MRR は 1 に近くなる．

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．

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 $count$，正解タグの個数を sum とする．この時の網羅率を以下のように定義して，n を変化させて算出した．

$$\text{網羅率} = \frac{count}{sum}$$

Random

ランダムに n 個タグをあげて，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．

各タグ毎の macro-precision

macro-precison の算出方法を説明する．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．

あるタグに注目する．そのタグが 1 位タグであり，かつ正解タグにも含まれている数を TP とおく．また，そのタグが 1 位タグであり，かつ正解タグに含まれていない場合を FP とおく．この時．このタグの macro-precision は以下のようになる．

$$macro\text{-}precision = \frac{TP}{TP + FP}$$

この指標は，あるタグを 1 位タグとして提案した場合に，そのタグが正解タグに含まれている確率を示す．

各タグ毎の macro-recall

macro-recall の算出方法を説明する．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．

あるタグに注目する．そのタグが正解タグに含まれていて，かつそのタグが 1 位タグである数を TP とおく．また，そのタグが正解タグに含まれていて，かつそのタグが 1 位タグでない数を FN とおく，この時，このタグの macro-recall は以下のようになる．

$$macro\text{-}recall = \frac{TP}{TP + FN}$$

この指標は，あるタグが正解タグに含まれている場合に，そのタグが 1 位タグである確率を示す．

各タグ毎の macro-F1

あるタグにおける macro-F1 はそのタグの macro-precision と macro-recall の調和平均となる．つまり，以下のようになる．

$$macro\text{-}F1 = \frac{2 \times macro\text{-}precision \times macro\text{-}recall}{macro\text{-}precision + macro\text{-}recall}$$

4.2.4 結果

結果は 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 9 のようになった．この n とは，何個のタグを推測タグとして提案するかを表す．また，各タグ毎の macro-precision，macro-recall，macro-F1 とその平均を macro-F1 の降順に表 9 に示す．

図 9 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 9 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．

4.2.5 考察

表 9 から，“Geometry”，“Flow-Algorithms” の macro-F1 が 0.6 以上と高く，この 2 つのタグの推測精度が高いことがわかる．それに対して，“Game”，“Ad-Hoc” の macro-F1 が 0.1 以下と低く，この 2 つのタグの推測精度が低いことがわかる．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．

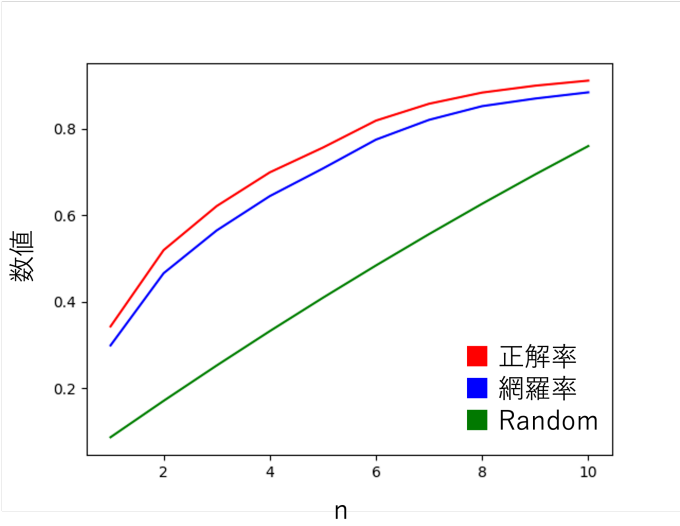

図 9: 評価実験 1 で算出した正解率，網羅率，Random

表 9: 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

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タグ 1 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タグ 2 と呼ぶ．あるタグに注目した時の頻出タグ 1 と頻出タグ 2，また，それぞれの出現確率を調べた結果を表 10 に示す．

表 10: 評価実験 1 における出現しやすいタグとその確率

タグ	頻出タグ 1	頻出タグ 2
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%)

結果から，多数のタグにおいて，頻出タグ 1 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，頻出タグ 2 として “Easy” が挙げられていることがわかる．また，表 8 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”，表 7 に示す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 として基本学習用データセット，基本評価用データセットの問題のうち，図 11 に示すアルゴリズム情報に関するタグが付与されている問題で学習，評価を行った．

以下，本節では学習用データセットの作成，評価用データセットの作成，実験方法，結果と考察について述べる．

4.3.1 学習用データセット

基本学習用データセットのうち，AtCoder Tags でタグが 1 つ以上付与されている問題で，かつ付与されているタグが表 11 に示す 5 個のタグのみで構成されている問題を選出する．選出した問題数は 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降順に並べたものを表 12 に示す．合計して 100%を超えるのは 1 つの問題にタグが複数付与されている場合があるためである．

表 11: 評価実験 2 で使用した問題のタグ一覧

Dynamic-Programming	Flow-Algorithms	Greedy-Methods
Mathematics	Searching	

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，AtCoder Tags でタグが 1 つ以上付与されている問題で，かつ付与されているタグが表 11 に示すタグのみで構成されている問題を選出する．選出した問題の数は 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降順に並べたものを表 13 に示す．合計して 100%を超えるのは学習用データセットの場合と同様に 1 つの問題にタグが複数付与されている場合があるためである．

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．

表 12: 評価実験 2 の学習用データセットにおいて各タグが全体に占める割合

タグ	割合
Dynamic-Programming	37.6%
Mathematics	35.7%
Searching	28.5%
Greedy-Methods	17.1%
Flow-Algorithms	3.5%

表 13: 評価実験 2 の評価用データセットにおいて各タグが全体に占める割合

タグ	割合
Mathematics	56.2%
Searching	24.0%
Dynamic-Programming	22.9%
Greedy-Methods	11.5%
Flow-Algorithms	2.1%

4.3.4 結果

結果は 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 10 のようになった．この n とは，何個のタグを推測タグとして提案するかを表す．また，各タグ毎の macro-precision，macro-recall，macro-F1 とその平均を macro-F1 の降順に表 14 に示す．

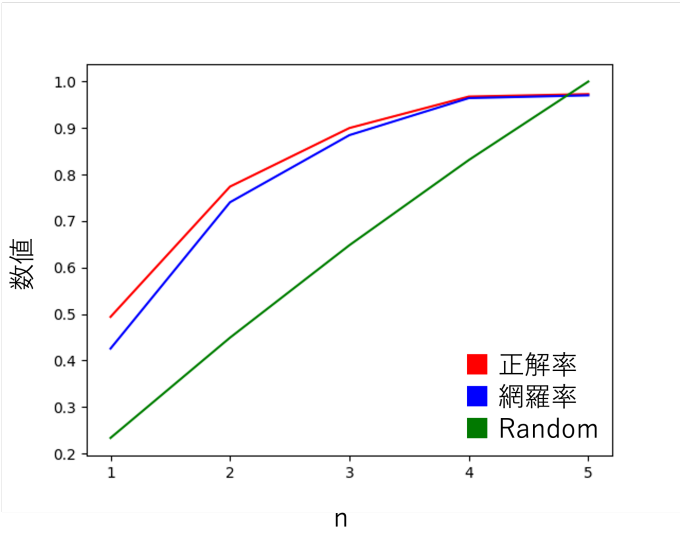

図 10: 評価実験 2 で算出した正解率，網羅率，Random

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 14 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．

表 14: 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

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 14 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，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タグ 1 と呼び，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タグ 2 と呼ぶ．あるタグに注目した時の頻出タグ 1 と頻出タグ 2，また，それぞれの出現確率を調べた．結果を表 15 に示す．

表 15: 評価実験 2 における出現しやすいタグとその確率

タグ	頻出タグ 1	頻出タグ 2
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%)

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 12 と表 13 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．

また，新濱らの研究 [9] 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文は類似度が高くなることが確認されているため，ソースコード情報だけでなく問題文などの情報も学習に使用することで，推測の精度が向上すると考えられる．

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．

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．

今後の課題として，以下の 2 つが挙げられる．1 つ目は，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 [9] 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，AtCoder の問題だけでなく，Codeforces の問題などにも提案手法が有効かどうか調べることである．Codeforces では各問題にタグが付与されているため，提案手法を適用することができる．この調査により提案手法の汎化性能を調べることができると考えられる．

謝辞

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 井上 克郎 教授には，研究活動に対して多くの貴重な御助言や御指導を賜りました. 心より深く感謝申し上げます.

大阪大学大学院情報科学研究科コンピュータサイエンス専攻 松下 誠 准教授には，研究の着想から研究活動の直接の御指導，論文の執筆に至るまで，あらゆる場面で多くの御指導を賜りました. 松下 誠 准教授の適切な御指導により，本論文を完成させることができました. 心より深く感謝申し上げます.

名古屋大学大学院情報学研究科附属組込みシステム研究センター 吉田 則裕 准教授には，研究の着想から研究に関する御指導，論文の執筆に至るまで，多くの御指導を賜りました. 吉田 准教授の適切な御指導及び御助言を頂いたことにより，本論文を完成させることができました. 心より深く感謝いたします.

大阪大学大学院情報科学研究科 神田 哲也 助教には，研究室での発表や評価内容について，大変貴重な御意見・御助言を賜りました. 多くの御助言を頂いた神田助教に心より深く感謝いたします.

最後に，様々な御指導・御助言を頂いた大阪大学大学院情報科学研究科コンピュータサイエンス専攻井上研究室の皆様に，心より深く感謝いたします.

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] Nick Craswell. *Mean Reciprocal Rank*, pp. 1703–1703. Springer US, Boston, MA, 2009.

[3] Ebtekar and Paul Liu. An elo-like system for massive multiplayer competitions. *arXiv preprint arXiv:2101.00400*, 2021.

[4] 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.

[5] Andrew M. Dai, Christopher Olah, and Quoc V. Le. Document embedding with paragraph vectors. *arXiv preprint arXiv:1507.07998*, 2015.

[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 - Volume 2*, NIPS'13, pp. 3111–3119, Red Hook, NY, USA, 2013. Curran Associates Inc.

[7] 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.

[8] 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.

[9] 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.