Vol.2022-SE-210 No.26
2022/3/12

問題解決過程を考慮した不具合報告の要約生成手法の提案

作島　大智[1,a)]　大平　雅雄[1,b)]　辻　裕真[1,c)]　東　裕之輔[1,2,d)]　西脇　一尊[2,e)]

概要：不具合報告を利用する開発者を支援するために，不具合報告の要約生成に関する研究が行われている．既存手法では，日々大量に報告される不具合の内容把握を効率化することを目的としており，自身の環境で起きている問題を解決するために必要な情報の入手を目的とする要約は提案されていない．そこで本研究は，句構造の特徴を用いて問題解決過程文を考慮した要約生成手法を提案する．評価の結果，句構造の特徴を考慮すると問題解決過程文を含む要約を作成できる割合が全体的に向上しており，特に問題解決過程文の 3 つの要素を含む要約の割合は 25%向上している．また，問題解決過程文を抽出するために利用した句構造の特徴が不具合の内容把握に対して大きな悪影響を与えることはなく，提案手法が作成する要約は不具合の内容把握に適していることがわかった．

Daichi Sakushima[1,a)]　Masao Ohira[1,b)]　Yuma Tsuji[1,c)]　Yunosuke Higashi[1,2,d)]
Kazutaka Nishiwaki[2,e)]

1. はじめに

オープンソースソフトウェアプロジェクトでは日々多くの不具合報告が行われている．不具合報告はバグリポジトリ（Bugzilla 等）や課題追跡システム（Github の Issue Tracker 等）で管理されており，プロジェクトの長期的な活動を維持するために重要な資源である．そのため，プロジェクトを管理するメンテナやプロジェクトの製品を使用するユーザなど，多くの利害関係者が不具合報告を利用している．しかし，開発者は不具合報告を理解するために，非常に大きな時間的コストを費やしている．例えば，開発者がわずか 10 件の過去の不具合報告を参照するだけで，平均して 600 件近くの文章を読む必要がある [6]．そのため，不具合報告を利用する開発者の支援を目的とする研究が複数行われている．

不具合報告を利用する開発者の支援を目的とする研究の一環として，不具合報告の要約生成に関する研究が行われており，様々な手法が提案されている．例えば，不具合報告に出現する頻出単語に基づく生成手法 [4] や，オートエンコーダモデルを用いて抽出した頻出単語に依存しないテキスト特徴量とコメントの妥当性に基づく生成手法 [5] などがある．これらの手法では，日々大量に報告される不具合の内容把握を効率化することを目的として，メンテナを支援している．しかし，自身の環境で起きている問題を解決するために必要な情報の入手を目的とする個々の開発者

[1] 和歌山大学
Wakayama University
[2] 株式会社日本総合研究所
The Japan Research Institute, Limited
[a)] s210045@wakayama-u.ac.jp
[b)] masao@wakayama-u.ac.jp
[c)] s226148@wakayama-u.ac.jp
[d)] s199005@wakayama-u.ac.jp
[e)] nishiwaki.kazutaka@jri.co.jp

やユーザの支援は行われていない．そこで本研究は，個々の開発者やユーザの支援を目的とする要約生成手法の提案を行う．

2. 不具合報告の要約生成手法

2.1 不具合報告

オープンソースソフトウェア（OSS）プロジェクトは，ソフトウェアの不具合を記録するために不具合報告を管理している．不具合報告はソフトウェアの不具合に直面した開発者によって OSS プロジェクトに提出される．不具合報告には，不具合に関する情報や．不具合解決に向けた議論，現在の不具合修正の進捗状況を示す Status 等が記載されている．本研究では，不具合報告の議論が行われたテキストデータ，すなわち報告者のテキストデータおよび参加者のテキストデータを対象に要約生成を行う．

2.2 要約生成手法の概要

要約生成には抽出型要約と抽象型要約が存在する．また教師あり機械学習，教師なし機械学習のアプローチが存在する．本研究では，教師なし学習を用いた抽出型要約生成手法を採用する．要約の種類，アプローチの採用理由を以下で説明する．

- 抽出型要約を採用する理由
  不具合報告には，不具合が生じた関数名やソフトウェアのバージョンなど，不具合が生じている環境情報が多数含まれている．環境情報が少しでも異なる場合，要約された不具合の意味が大きく変わる恐れがある．抽象型要約の場合，不具合報告に記載されている環境情報と異なる環境情報を出力する可能性がある．そのため，本研究では環境情報をそのまま抜き出すことができる抽出型要約を採用する．
- 教師なし学習のアプローチを採用する理由
  不具合報告の要約において，教師あり学習を行うための十分な学習データは存在せず，開発者が手作業で収集する必要がある．この作業は膨大なコストが必要なため，現実的ではない．そのため，本研究では機械的に学習データを収集できる教師なし学習のアプローチを採用する．

2.3 既存手法

2.3.1 DeepSum [4]

Xiaochen らは，不具合報告特有の 3 つの特徴に着目し，従来手法よりも精度の高い要約生成手法を提案している．

- 評価文と引用文
- 不具合報告文の優先順位
- 定義済みフィールドと関連づけられている環境情報

2.3.2 BugSum [5]

教師なし学習を用いたアプローチでは，不具合報告の頻出単語に基づいて要約を作成する方法が主流である．Haoran らは頻出単語に依存したアプローチは冗長な要約を作成する傾向にあることを問題視している．そこで，頻出単語に依存しないテキスト特徴量とコメントの妥当性に基づいて要約を作成することで要約の冗長性を緩和している．

2.4 本研究の立場

要約とは，文章の要点を短くまとめたものである．本研究の提案手法と既存手法では，不具合報告の要約に含めるべき要点は異なる．既存手法では，プロジェクトの運営に関わる開発者が不具合の内容把握を効率的に行うための要約を作成している．そのため要約に含めるべき要点は，不具合の概要を理解する上で必要な文である．それに対して，提案手法では，自身の環境で起きている問題を解決したい開発者が必要とする情報を含む要約の作成を目指す．そのため要約に含めるべき要点は，不具合の要因や解決方法に関する文である．以上のように，既存手法と提案手法では，不具合報告の要約に含めるべき要点は異なる．

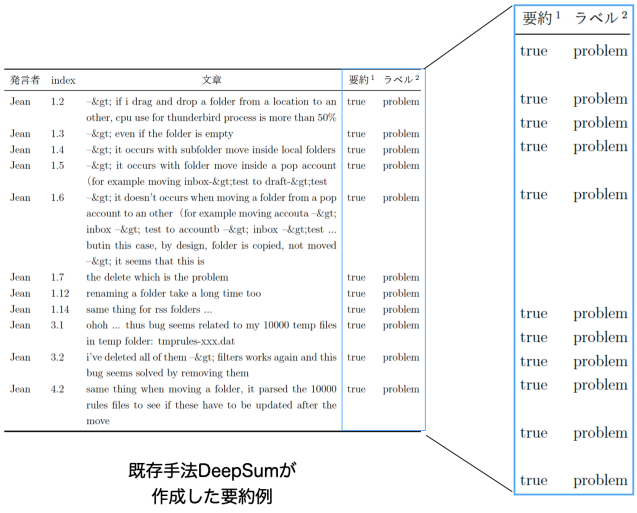
- 既存手法
  日々大量に報告される不具合の内容把握を効率化することを目的としている．貢献対象は，プロジェクトを運営するプロジェクトマネージャやメンテナである．
- 提案手法
  過去に解決された大量の不具合報告から，自身の環境で起きている問題を解決するために必要な情報の入手を効率化することを目的としている．貢献対象は，プロジェクトの製品を利用する個々の開発者やユーザである．

2.5 既存手法の課題

2.4 節で述べたように，提案手法と既存手法では，不具合報告の要約に含めるべき要点は異なる．本研究では，既存手法は提案手法の目的を達成することができないと考えている．そのように考える根拠を既存手法である DeepSum が作成した要約例を用いて説明する．図 1 は評価データである SDS [8] のうち，正解集合に problem，suggestion，fix のラベルが存在する不具合報告を用いた際の要約例である．要約の列は，対象文が不具合の内容把握に必要な文であるかどうかを示している．ラベルの列は，対象文がどのような事柄に関わる文であるかを示している．

この例では，既存手法が作成した要約に含まれる文は全て不具合の内容把握に必要な文であり，既存手法の抽出精度が高いことを示している．しかし，既存手法が抽出した文のラベルはすべて problem であり，suggestion や fix 等のラベルを持つ文が含まれていない．これは要約には報告者の抱える問題のみが含まれており，不具合が生じた要因や解決方法が含まれていないことを意味する．そのため，この例では過去の不具合情報に基づいて自身の開発環境下

で生じている問題を解決しようとする開発者に必要な情報が十分に含まれていないと考えられる．



発言者	index	文章	要約[1]	ラベル[2]
Jean	1.2	–> if i drag and drop a folder from a location to another, cpu use for thunderbird process is more than 50%	true	problem
Jean	1.3	–> even if the folder is empty	true	problem
Jean	1.4	–> it occurs with subfolder move inside local folders	true	problem
Jean	1.5	–> it occurs with folder move inside a pop account (for example moving inbox->test to draft->test	true	problem
Jean	1.6	–> it doesn't occurs when moving a folder from a pop account to an other (for example moving accouta –> inbox –> test to accountb –> inbox –>test ... butin this case, by design, folder is copied, not moved –> it seems that this is	true	problem
Jean	1.7	the delete which is the problem	true	problem
Jean	1.12	renaming a folder take a long time too	true	problem
Jean	1.14	same thing for rss folders ...	true	problem
Jean	3.1	ohoh ... thus bug seems related to my 10000 temp files in temp folder: tmprules-xxx.dat	true	problem
Jean	3.2	i've deleted all of them –> filters works again and this bug seems solved by removing them	true	problem
Jean	4.2	same thing when moving a folder, it parsed the 10000 rules files to see if these have to be updated after the move	true	problem

図 1　DeepSum が作成した要約例（SDS#23）

2.6　本研究の動機と着眼点

本研究は，個々の開発者やユーザが抱える問題を解決するために，過去に報告・解決された不具合の情報を効率的に入手することを目的とする．上記の目的を達成することのできる要約には，以下の 3 つの文（以降，問題解決過程文）を含むと考える．

- 報告者が抱える問題（問題文）
- 不具合が生じた要因（要因文）
- 不具合の解決方法（解決文）

本研究では，問題解決過程を考慮した要約生成を行うために，文の句構造に着目して問題解決過程文の特徴を捉える．句構造に着目する理由は，問題解決過程文は通常の文と異なる文構造を持っていると考えられるためである．例えば，報告者は不具合が生じることに疑問を持っているため，問題文は疑問詞に関する特徴が他の文に比べて多くなることが推測できる．また，頻出単語に依存しない抽象的な特徴であり，作成した要約の冗長性を緩和することができると考えられるためである．予備調査として，文の構造的特徴を反映している構文 POS タグの分布を明らかにし，問題解決過程文とその他の文で出現する句構造に違いがあるのかを確認する．Github の Issue Tracker で管理されている docker-library/python リポジトリから著者を含めた 2 人の大学院生が目視で収集した問題解決過程文（表 1）を分析対象に予備調査を行なった結果，要因文や解決文では形容詞句タグや副詞句タグ，前置詞句タグの出現確率が他と比べて少し大きく，出現割合に違いがあることを確認した．

3.　本研究の目的

2.4 節で述べたとおり，本研究と既存手法では作成する要約の目的が異なる．既存手法では，日々大量に報告される不具合の内容把握を効率化することを目的としている．それに対し，提案手法は過去に解決された大量の不具合報告から，自身の環境で起きている問題を解決するために必要な情報の入手を効率化することを目的としている．本研究では問題を解決するために必要な情報が記載されていると考えられる問題解決過程文を含む要約を作成することを目指す．

表 1　抽出した文数

	問題文	要因文	解決文	その他
文数	82	46	66	464

4.　本研究のアプローチ

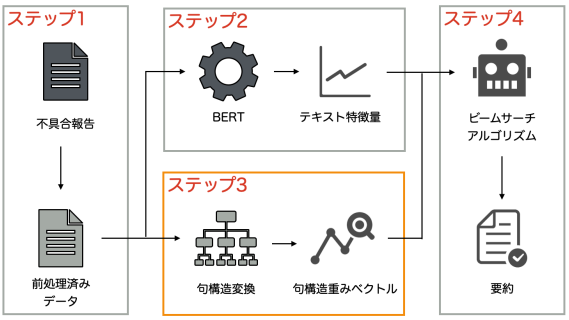


図 2　提案手法の全体図

本章では，本研究の提案手法である「問題解決過程を考慮した要約生成手法」について説明する．提案手法は図 2 のように，4 つのステップで構成される．まず，前処理として不具合報告文をクリーニングし，単語に分割する．次に，事前学習済みモデルを用いて不具合報告のテキスト特徴量を抽出する．そして，句構造の特徴を用いて問題解決過程文の特徴量を捉え，ベクトルに変換する．最後に，テキスト特徴量と問題解決過程文の特徴量を捉えた句構造重みベクトルに基づいて用いて要約を作成する．提案手法は既存手法である BugSum をベースとしており，第 3 ステップが提案手法の重要なステップである．それぞれのステップについて以降で詳しく説明する．

4.1　不具合報告の前処理

実社会のデータとしての不具合報告には，非常に多くのノイズが含まれている．そのため，ノイズを除去するために不具合報告のテキストデータに対して前処理を行う必要がある．以降で前処理で行なった工程を順に説明する．

まず，不具合報告文のクリーニングを行う．不具合報告の文中に含まれる引用符や箇条書きの記号，$や&等の特殊記号など，要約のノイズとなる文字を正規表現を用いて除去する．また，URL やコードスニペットも記号と同様に要約のノイズとなってしまうため，正規表現を用いて表 2

のように統一する．次に，“Bug” と “bug” のような表記揺れを解消するために文字を全て小文字に変換する．

表 2　統一表

対象	統一文字列
URL	url
コードスニペット	codesnipet
モジュール名や関数名	modulename
ファイル名やパス	filename
バージョン番号	versionid

次に，クリーニング後の不具合報告文を言語的に意味のある最小単位であるトークンに分割する．そして，分割したトークンを辞書における単語の見出語（レンマ）で統一する．例えば，“women” を “woman”，“databases” を “database” で統一する．トークンのレンマ化を行うことで，トークンが正規化され，トークンの品詞を正確に捉えることができる．

最後に，ノイズおよび語彙数を削減するため，文章において重要ではない単語であるストップワードを除去する．本研究では，公開されているストップワード[*1]と spaCy [2] の en_core_web_lg に登録されているストップワードに該当するトークンを対象とする．

4.2　テキスト特徴量抽出

既存手法ではエンコーダ・デコーダモデルを構築し，不具合報告文のテキスト特徴量を抽出している．提案手法は既存手法である BugSum をベースとしているため，既存手法が使用しているエンコーダ・デコーダモデルを本研究でも使用することが妥当であると考えられる．しかしながら，モデルを再現するために必要な学習データは公開されていないため，本研究では豊富な単語埋め込みの表現が可能である事前学習された言語モデルを使用する．本研究では，事前学習された言語モデルの 1 つである BERT を採用する．実装時には bert-as-service[*2]を使用し，大文字小文字を区別しない BERT-Base モデルを採用している．不具合報告の各文を入力とし，各文に対応するテキスト特徴量を 728 次元の高次元ベクトルとして抽出する．

4.3　句構造重みベクトル

本節では，問題解決過程文の特徴を捉える句構造重みベクトルの構築方法について説明する．まず，予備調査で使用した問題解決過程文データ（表 1）を用いて，問題解決過程文で頻出するルールを抽出する．そして，抽出したルールに各文がどの程度該当するかに基づいて重みを算出し，重みベクトルを構築する．以降で，詳しく説明する．

[*1] https://www.ranks.nl/stopwords
[*2] https://github.com/hanxiao/bert-as-service

4.3.1　頻出ルールの抽出

まず，問題解決過程文がどのような構造をしているのかを分析するために，構文解析を行う．主な構文解析には，依存関係解析と構成要素解析の 2 種類がある．依存関係解析では，文中の単語間の関係（主オブジェクトや述語などのマーク付け）に焦点を当てている．構成要素解析では，PCFG（Probabilistic Context-Free Grammar）を使用して構文解析木を構築することに焦点を当てている．本研究では文全体の構造に着目するため，構成要素解析を行う．本研究では，自然言語処理ツールである Stanford CoreNLP [7] を使用する．

次に，構成要素解析を行なったデータを分析可能なデータに整形する．句構造は文中の単語や句，節などのまとまりであり，関係性を示す構文 POS（Part-of-speech）タグと，単語の品詞を表す品詞 POS タグで構成されている．これらの POS を図 3 のように深さ優先で探索し，一次元配列データへと整列させる（以降，整列データと呼称）．

整列データを作成する際に，句構造のどの POS タグを使用するのかを考慮する必要がある．全ての POS タグを使用して整列データを構築する，もしくはより抽象的な特徴表現にするために宣言句タグ及び葉ノードタグを含めずに整列データを構築するなど，複数の整列方法がある．しかし，どの整列方法が最も問題解決過程文の特徴を捉えられるのかは明らかではない．そのため，本研究では全ての組み合わせで整列データを作成する．全ての組み合わせで整列データを作成する場合の具体例を図 4 に示す．また本研究ではそれぞれの整列データを表 3 のように Type で呼称する．

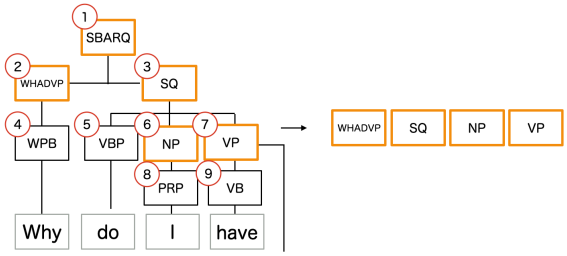


図 3　整列データの作成方法
（宣言句タグおよび葉ノードタグを含めない場合）

表 3　整列データの種類

整列データ	葉ノードタグ	宣言句タグ
Type1	○	○
Type2	○	×
Type3	×	○
Type4	×	×

最後に，整列データから問題解決過程文の特徴を捉える POS の組み合わせ（以降，頻出ルールと呼称）を抽出する．

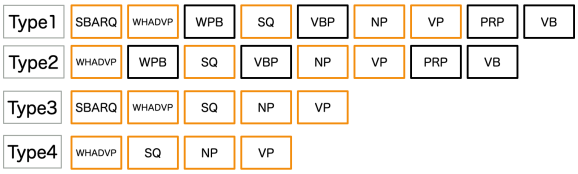


図 4 図 3 から整列データを作成した場合の具体例

本研究ではアソシエーション・ルール・マイニングを行い，頻出ルールを抽出する．アソシエーション・ルール・マイニングで抽出されるルールには評価指標が存在する．抽出されるルールは非常に膨大であり，有効なルールの抽出が困難であるため，評価指標に基づいてルールのフィルタリングを行う．一般的な目安として，リフト値が 1 より大きいルールは有効であるとされている．そのため，本研究ではリフト値が 1 以上のルールのみを対象とし，アソシエーション・ルール・マイニングで抽出した頻出ルールを用いてベクトル構築を行う．

4.3.2 ベクトル化

抽出した頻出ルールに不具合報告の各文がどの程度該当するかに基づいて重みを算出し，重みベクトルを構築する．具体的な手順を図 5 を用いて説明する．この例では，不具合報告が s_1，s_2，s_3，s_4 の 4 つの文から構成されているものとする．各文が頻出ルールに該当するか，環境情報（関数名もしくはバージョン情報）を含むかの 2 点を考慮して重みを付与する．まず，頻出ルールに該当するかの判定を行う．s_1 には SBAR が存在しないので頻出ルールに該当しない．s_2 は VP，SBAR が共に存在しているが，頻出ルールは出現する順序を考慮するため，頻出ルールに該当しない．s_3，s_4 は VP，SBAR が共に存在し，頻出ルールと同じ順序でタグが出現しているため，頻出ルールに該当する．次に，頻出ルールに該当した文（s_3，s_4）が環境情報を含むかを確認する．環境情報を含む文は問題を解決するために必要な情報である可能性が高いと考えられる．そのため，本研究では環境情報を含む文は含まない文よりも相対的に重みを大きくしている．以上の手順を全てのルールを対象に行い，最終的な重みを用いて重みベクトルを構築する．この例においてルール j のみを用いた場合の重みベクトルは（0，0，0.05，0.10）となる．

本研究では，各文の重みに基づいて作成したベクトルを句構造重みベクトル $Wscore$ と呼称する．不具合報告の文 s_i の重みを（1）式，句構造重みベクトルの重みを（2）式で表すことができる．

$$Weight_i = \alpha\beta N_i \tag{1}$$

$$Wscore_i = 1 + Weight_i \tag{2}$$

α はルール 1 つ分の重みとして 0.10，β は環境情報を含む場合 1，含まない場合 0.5 としている．また N_i は，文 s_i に該当するルールの総数を示す．ルール 1 つ分の重みが大きい場合，少しのルールに該当する文でも大きな重みが付与され，要約生成の際にテキスト特徴量が反映されないことが考えられる．そのため本研究では，テキスト特徴量と頻出ルールを用いた重みの両方を用いて要約を作成するために重みの数値を小さく設定している．

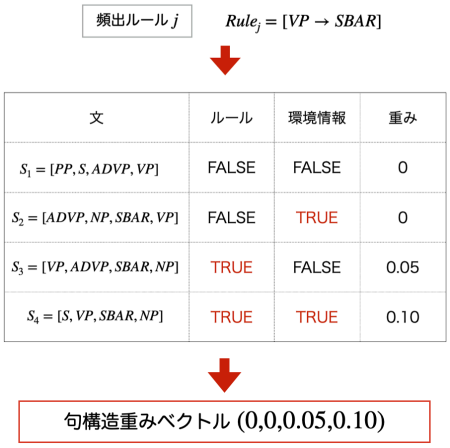


図 5 頻出ルールに該当するか否かをベクトルで表現する手順例

4.4 要約生成

本節では要約生成について説明する．まず，不具合報告に含まれる文章から要約候補文のフィルタリングを行う．その後，テキスト特徴量と句構造重みベクトルに基づいて要約を作成する．

4.4.1 要約候補文のフィルタリング

要約に含めるには不適切な文章が不具合報告には含まれているため，不具合報告の中から要約の対象外となる文を除去する．対象となる文について以下で説明する．

- ソフトウェア言語文
  不具合報告では，報告者が不具合を報告する際，ソフトウェア言語文（ソースコードやエラー文等）をテキストデータとしてコピー&ペーストすることがある．このような不具合報告では，ソフトウェア言語文を前処理で行った正規表現で除去することができない．そのため，要約を作成する際に，ソフトウェア言語文を要約の候補文から削除する．
  本研究では，言語判定モデルである FastText [3] を用いて，ソフトウェア言語文を判定する．FastText は，入力文に最も類似している言語の類似度を出力する．入力文が自然言語でない場合，どの言語にも類似しないという結果になるため，出力結果は小さくなる．そのため，類似度が小さい文をノイズ文であると判定することができる．本研究では閾値を 0.30 とする．この閾値は，明らかなソフトウェア言語文を除去し，文中にソースコードの一部を含む文は除去しない値となっている．

- 重複文
  不具合報告には，他のコメントに返信を行うためにそのコメントを引用して自身のコメントを作成することがある．引用を含む文は他の文と重複しており，要約生成の際にノイズになるため除去する．本研究では，既存手法と同様に文の初めに「>」がある場合もしくは内容の 9 割が一致している文を重複文として除去する．

4.4.2 要約の作成方法

提案手法が作成した要約の特徴量は不具合報告全文の特徴量と可能な限り近い必要がある．そのため本研究では，不具合報告全文の特徴量と提案手法が選択している文からなる集合（選択文セット）の特徴量の差が最も小さくなる場合の選択文セットが最も要約に適していると考える．

$$WSF = \sum_{i=1}^{n} SF_i * Wscore_i \tag{3}$$

$$\tilde{WSF} = \sum_{i \in Chosen}^{n} SF_i * Wscore_i \tag{4}$$

$$MSE = \frac{\sum_{i=1}^{d}(y_i - \tilde{y_i})^2}{d} \tag{5}$$

$$\delta = |MSE(WSF, \tilde{WSF})| \tag{6}$$

SF_i は BERT を用いて抽出した文 S_i のテキスト特徴量，$Wscore_i$ は文 S_i の重みであり，WSF は不具合報告全文の特徴量，$\tilde{WSF}$ は選択文セットの特徴量を示している．また，d は特徴ベクトルの次元であり，y_i は WSF の各要素，$\tilde{y_i}$ は $\tilde{WSF}$ の各要素を示している．

要約の作成方法の具体的な手順について説明する．まず，(3) 式を用いて不具合報告全文の特徴量を，(4) 式を用いて選択文セットの特徴量を抽出する．次に，不具合報告全文の特徴量と選択文セットの特徴量の平均二乗誤差を (5) 式を用いて算出する．誤差 δ は小さいほど，選択文セットには不具合報告全文の特徴量がより多く含まれていることを示している．そのため，選択文セットごとに算出した平均二乗誤差の中から最も誤差が小さい選択文セットを要約とする．提案手法は，単語量制限 θ の下で誤差を最小化する文の最適な組み合わせを見つけることを目的としている．

5. ケーススタディ

提案手法の評価には Summary Dataset（SDS）[8] と Authorship Dataset（ADS）[1] という公開されている 2 つの評価データを使用する．評価データには各不具合報告の要約および要約作成に役立った文が含まれている．3 人のうち 2 人が要約作成に役立ったと感じた文を，GSS（Gold Standard Set）と呼ぶ [8]．また，ケーススタディで評価する提案手法が使用しているルールの内訳を表 4 に示す．本研究では，その他の文で出現するルールは使用ルールの対象外とし，問題解決過程文でのみ出現するルールを使用している．

表 4 リフト値 1 以上，支持度 0.30 以上のルール数

	問題文固有	要因文固有	解決文固有
Type1	4	32	3936
Type2	2	72	3094
Type3	0	2	32
Type4	0	0	34

リサーチクエスチョンを以下に示す．

- RQ1:句構造の特徴を考慮すると，不具合解決に必要な情報の収集を目的とする要約を作成できるようになるのか
- RQ2:句構造の特徴を考慮することは，不具合の内容把握を目的とする要約に影響を与えるのか

6. 評価

6.1 RQ1

6.1.1 評価指標

提案手法が作成する要約が本研究の目的を達成できているかを問題解決過程文を含むかで評価する．具体的には GSS に該当する問題解決過程文のうちどれか 1 つを含む要約，どれか 2 つを含む要約，3 つ全てを含む要約の数および割合で評価する．問題解決過程文の 3 つ要素全てを対象とした時の評価例を図 6 に示す．左のベン図では今回対象としている問題文，要約文，解決文を含む要約を作成することができている．しかし中央のベン図では，要約に GSS に該当する要因文を含まないため，対象文を含む要約を作成することはできていないと判定する．また，右のベン図のようにそもそも GSS 内に対象文が存在しない不具合報告は評価対象外としている．

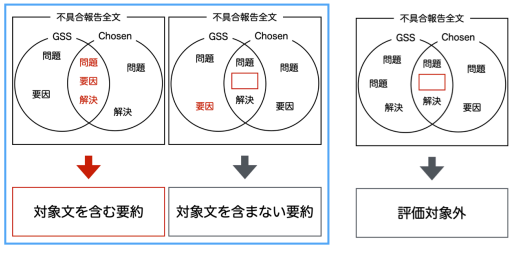


図 6 対象文を含む要約と含まない要約

そのため RQ1 では，対象文を含む要約の数 $Count_{target}$ とその割合 $Rate_{target}$ を評価指標とする．Num は対象文において評価対象となる不具合報告の数を示す．

$$Rate_{target} = \frac{Count_{target}}{Num} \tag{7}$$

6.1.2 評価結果

問題解決過程文の全ての組み合わせを対象に，提案手法がどの程度問題解決過程文を含む要約を作成できているのかを評価する．ADS では文ごとの細かいラベルが存在しないため，RQ1 では SDS のみを評価データとして使用する．

句構造の特徴を考慮する場合と考慮しない場合の比較結果を表 5，表 6 に示す．Text は不具合報告のテキスト特徴量のみを用いた場合，Type は不具合報告のテキスト特徴量と句構造の特徴を用いた場合を示している．対象文は，問題解決過程文の各組み合わせを示している．Num は対象文において評価対象となる不具合報告の数を示す．Type は表 3 の整列データの種類を示しており，全ての整列データを用いた際の結果を示している．句構造を考慮することで結果が向上したセルを灰色で示し，対象文ごとに最も結果が良かった数値を黒字で示す．

評価の結果，句構造の特徴を考慮するとどの組み合わせでも要約に問題解決過程文を含む割合が向上していることが明らかになった．特に Type1 の句構造を用いた場合，問題文，要因文，解決文の全てを同時に要約に含む割合が 25%向上しており，不具合解決に必要な情報の収集を目的とする要約の作成に貢献できることがわかる．以上の結果より，句構造の特徴を用いると問題解決過程文の特徴を捉えることができることが示唆される．

RQ1 の回答

句構造の特徴量を考慮する提案手法は不具合解決に必要な情報の収集を目的とする要約作成に貢献できる．

6.2 RQ2

6.2.1 評価指標

RQ2 では再現率，適合率，F 値の 3 つの評価指標を用いて，提案手法を評価する．

$$Recall = \frac{|Chosen \cap GSS|}{|GSS|} \tag{8}$$

$$Precision = \frac{|Chosen \cap GSS|}{|Chosen|} \tag{9}$$

$$F1 = \frac{2 * Recall * Precision}{Recall * Precision} \tag{10}$$

6.2.2 評価結果

句構造の特徴を考慮する場合と考慮しない場合を対象に GSS の抽出精度を比較する．GSS の抽出精度の比較結果を表 7 に示す．先ほどの表と同様に，Text は不具合報告のテキスト特徴量のみを用いた場合，Type は不具合報告のテキスト特徴量と句構造の特徴を用いた場合を示している．

表 7 RQ2 の結果

	SDS			ADS		
	Recall	Precision	F	Recall	Precision	F
Text	0.35	0.45	0.39	0.44	0.40	0.39
Type1	0.39	0.42	0.40	0.43	0.38	0.40
Type2	0.38	0.43	0.40	0.43	0.39	0.41
Type3	0.38	0.44	0.41	0.45	0.38	0.41
Type4	0.36	0.42	0.39	0.44	0.39	0.41
BugSum[1]	0.41	0.63	0.49	0.42	0.61	0.49

[1] BugSum の値は論文 [5] から引用している．

表より，GSS の抽出精度は F 値で見ると 0〜2%と小幅ながら向上していることがわかる．つまり，句構造を考慮

表 5 対象文を含む要約の数（括弧内の数値は向上率を示す）

対象文	Num	Text	Type1	Type2	Type3	Type4
問題	35	33	**35（6%）**	**35（6%）**	**35（6%）**	**35（6%）**
要因	33	26	**29（9%）**	27（3%）	**29（9%）**	28（6%）
解決	31	18	**20（7%）**	**20（7%）**	18（0%）	17（-3%）
問題+要因	32	23	**28（16%）**	27（12%）	**28（16%）**	27（12%）
問題+解決	30	16	**20（14%）**	19（10%）	18（7%）	17（4%）
要因+解決	29	11	**16（17%）**	15（14%）	14（10%）	12（3%）
問題+要因+解決	28	9	**16（25%）**	15（22%）	14（18%）	12（11%）

表 6 対象文を含む要約の割合

対象文	Num	Text	Type1	Type2	Type3	Type4
問題	35	0.94	**1.00**	**1.00**	**1.00**	**1.00**
要因	33	0.79	**0.88**	0.82	**0.88**	0.85
解決	31	0.58	**0.65**	**0.65**	0.58	0.55
問題+要因	32	0.72	**0.88**	0.84	**0.88**	0.84
問題+解決	30	0.53	**0.67**	0.63	0.60	0.57
要因+解決	29	0.38	**0.55**	0.52	0.48	0.41
問題+要因+解決	28	0.32	**0.57**	0.54	0.50	0.43

する場合としない場合では GSS の抽出精度は大きく変化していないことを示している．また，内容把握を目的とする既存手法である BugSum と比べて精度が大きく劣化しているわけではない．そのため，問題解決過程文を抽出するために利用した句構造の特徴が，不具合の内容把握に対して大きな悪影響を与えてはいないことがわかる．

RQ2 の回答

問題解決過程文を抽出するために利用した句構造の特徴が不具合の内容把握に対して大きな悪影響を与えることはなく，提案手法が作成する要約は不具合の内容把握に適していることがわかった．

7. 考察

句構造重みベクトルの重みを計算する際，環境情報を含む文の重みを相対的に大きくしている．その結果，問題解決過程文の抽出精度を向上することができている．しかし，句構造ではなく，文に環境情報を含んでいるかどうかが精度向上に寄与している可能性がある．そこで，単純に環境情報を含む文に環境情報の個数を重みとして付与する場合（Text+envinfo）と提案手法を比較し，句構造が問題解決過程の抽出精度向上に寄与しているかを明らかにする．それぞれの評価結果を表 8 に示す．

表 8 Text+envinfo と提案手法の比較（括弧内は向上率を示す）

対象文	Num	Text	Text+envinfo	提案手法
問題	35	33	32（-3%）	**35（6%）**
要因	33	26	25（-3%）	**29（9%）**
解決	31	18	**22（13%）**	20（7%）
問題+要因	32	23	22（-3%）	**28（16%）**
問題+解決	30	16	19（10%）	**20（14%）**
要因+解決	29	11	15（14%）	**16（17%）**
問題+要因+解決	28	9	13（14%）	**16（25%）**

表より，環境情報に基づいて重みを付与する場合，問題解決過程文の抽出精度が向上している．特に解決文を抽出できるようになっていることがわかる．そのため，環境情報を含む文の重みを相対的に大きくする句構造重みベクトルの構築方法は問題解決過程文の特徴を捉えるうえで適していると考えられる．また，提案手法の方が環境情報のみを用いる場合より問題解決過程文を抽出できていることがわかる．そのため，単純に環境情報のみではなく，構造的特徴を用いる本研究のアプローチは問題解決過程文の特徴を捉えることができていると考えられる．

.

8. まとめ

本研究は，過去に解決された大量の不具合報告から自身の環境で起きている問題を解決するために必要な情報の入手を効率化することを目的として，問題解決過程文を考慮した要約生成手法を提案している．本研究ではケーススタディとして，既存手法と同様の評価データを用いて提案手法を評価している．評価の結果，整列データ Type1 を用いる場合，問題解決過程文を含む要約を作成できる割合が 6〜25%向上することが明らかになった．特に問題文，要因文，解決文の全てを含む要約を作成できる割合が 25%向上しており，提案手法は本研究の目的を達成することができていると言える．また，句構造の特徴を考慮すると GSS の抽出精度は F 値で見ると 0〜2%と小幅ながら向上しており，問題解決過程文を抽出するために利用した句構造の特徴が不具合の内容把握に対して大きな悪影響を与えていないことがわかった．そのため，提案手法が作成する要約は不具合の内容把握に適していると言える．

参考文献

[1] He, J., Ma, H., Nazar, N. and Ren, Z.: Mining authorship characteristics in bug repositories, *Sciece China. Information Sciences*, Vol. 60, pp. 1–16 (online), DOI: 10.1007/s11432-014-0372-y (2017).
[2] Honnibal, M., Montani, I., Van Landeghem, S. and Boyd, A.: spaCy: Industrial-strength Natural Language Processing in Python, (online), DOI: 10.5281/zenodo.1212303 (2020).
[3] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H. and Mikolov, T.: FastText.zip: Compressing text classification models, *arXiv preprint arXiv:1612.03651* (2016).
[4] Li, X., Jiang, H., Liu, D., Ren, Z. and Li, G.: Unsupervised Deep Bug Report Summarization, *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, New York, NY, USA, Association for Computing Machinery, pp. 144–155 (online), DOI: 10.1145/3196321.3196326 (2018).
[5] Liu, H., Yu, Y., Li, S., Guo, Y., Wang, D. and Mao, X.: BugSum: Deep Context Understanding for Bug Report Summarization, *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, New York, NY, USA, Association for Computing Machinery, pp. 94–105 (online), DOI: 10.1145/3387904.3389272 (2020).
[6] Mani, S., Catherine, R., Sinha, V. S. and Dubey, A.: AUSUM: Approach for Unsupervised Bug Report Summarization, *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, New York, NY, USA, Association for Computing Machinery, (online), DOI: 10.1145/2393596.2393607 (2012).
[7] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S. and McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit., *ACL (System Demonstrations)*, The Association for Computer Linguistics, pp. 55–60 (online), available from ⟨http://dblp.uni-trier.de/db/conf/acl/acl2014-d.html#ManningSBFBM14⟩ (2014).
[8] Rastkar, S., Murphy, G. C. and Murray, G.: Automatic Summarization of Bug Reports, *IEEE Transactions on Software Engineering*, Vol. 40, No. 4, pp. 366–380 (online), DOI: 10.1109/TSE.2013.2297712 (2014).