技術的負債に関する課題票の単語分散表現を用いたテキスト分類

田口 舞奈
和歌山大学システム工学部
taguchi.maina@g.wakayama-u.jp

木村 祐太
和歌山大学システム工学研究科
kimura.yuta@g.wakayama-u.jp

大平 雅雄
和歌山大学システム工学部
masao@wakayama-u.ac.jp

要旨

技術的負債は，納期や予算などの制約により実装や設計の品質を妥協した結果，後に発生する追加のコストを指す．近年のオープンソースソフトウェア（*OSS*）開発では，課題管理システムを用いて技術的負債を管理するプロジェクトが増えてきているものの，不具合報告などと区別されることなく管理されることがいまだ一般的である．技術的負債と不具合とでは，原因および影響範囲の調査や修正に要するコスト，優先順位付けや担当者の割り当て方法など，様々な点で異なる場合があるため，それぞれを適切に管理するためには課題票が登録された時点で区別されていることが望ましい．

本研究の目的は，課題管理システムに登録された時点で課題票の内容が技術的負債に関するものかどうかを判別するための分類モデルを構築することである．課題票が登録された時点で入手可能な情報として，課題票のタイトルと本文に含まれるテキストデータがある．本論文では，これらテキストデータを *5* 種類（*BoW, TF*IDF, Word2Vec-Wiki, Word2Vec-SO, GloVe-Wiki*）の単語分散表現に変換し，*3* 種類の分類器（ランダムフォレスト，ロジスティック回帰，サポートベクターマシン）を用いて分類モデルを構築することで，それぞれの分類性能を定量的に示すとともに，単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにする．実験の結果から，作成した分類器を目的に応じて使い分けることが有効であるとわかった．

1　はじめに

ソフトウェア開発では，納期や予算などの制約などにより，設計や実装の品質を犠牲にする場合がある．最適ではない設計や実装をおこなった結果，後に保守作業の効率を低下させ余分のコストを発生させる．Cunningham [1] は，このような事象のことを，会計用語を交えて技術的負債（Technical Debt: TD）と表現した．

技術的負債は，放置する時間が長引くほど保守コストが増大してしまうため早期の発見と除去が重要であり，開発者・研究者双方の立場から高い関心を集めている [2]．これまで，技術的負債の存在を開発者がソースコードコメントとして表明する Self-Admitted Technical Debt (SATD) に関する研究 [3–13] が盛んに行われてきたが，特に近年では，課題管理システムを用いて課題票として管理される技術的負債の研究 [14, 15] が注目されつつある．

課題票として管理される技術的負債には，課題管理システムに登録される時点で技術的負債であることが判明している場合と，課題票として登録された後の開発者同士の議論を通じて技術的負債であることが判明する場合とがある．前者は，開発者がすでに技術的負債であることを認識しているため，SATD と同様のものとして管理することが可能である．一方後者は，課題票の登録時点では，不具合報告や改善要求などと区別しにくいため，開発者が技術的負債であると認識するまでに時間を要することがある．さらに，技術的負債と不具合とでは，原因および影響範囲の調査や修正に要するコスト，優先順位付けや担当者の割り当て方法など，様々な点で異なる

場合があるため，それぞれを適切に管理するためには課題票が登録された時点で区別されていることが望ましい．

そこで本論文では，課題管理システムに登録された時点で課題票の内容が技術的負債に関するものかどうかを判別するための分類モデルを構築する．課題票が登録された時点で入手可能なテキストデータを 5 種類（BoW, TF-IDF, Word2Vec-SO-corpus, Word2Vec-Wiki-corpus, Glove-Wiki-corpus）の単語分散表現に変換し，3 種類の分類器（ランダムフォレスト，ロジスティック回帰，サポートベクターマシン）を用いて分類モデルを構築することで，それぞれの分類性能を定量的に示すとともに，単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにする．

2 動機

本章では，本研究の動機を説明するために，課題票として管理される技術的負債の研究意義について述べた後，課題票として管理される技術的負債に関する研究を紹介し，最後に先行研究 [15] と本研究の位置付けの違いを明確にする．

2.1 課題票として管理される技術的負債の研究意義

技術的負債は開発者・研究者双方の立場から注目が集まっており，技術的負債の検出ツールなどの開発が盛んに行われている [2]．これまで開発されたツールは技術的負債の可能性があるものに関して大量の検出結果が得られる．しかし，偽陽性や開発者が重視しない結果も多数含まれるため，積極的なツールの利用には至っていない [16]．そのため，ソースコードコメント中に表明した開発者が返済すべきであると認識している技術的負債 (SATD) に関する研究 [3–13] が盛んに行われてきた．

特に近年では，課題管理システムを用いて課題票として管理される技術的負債の研究 [14, 15] が注目されつつある．SATD は開発者個々人がソースコードコメント中で表明しているものであり必ずしもプロジェクトとして管理対象となる保証はないが，課題票として管理される技術的負債はプロジェクト全体としての関心事であると捉えられるため，相対的に SATD よりも重要性が高いものである可能性が高い．実際，課題票として管理される技術的負債のうち SATD を対象としたものは 28.7% [14] しかなく，SATD の大多数はプロジェクトとしての管理対象とはなっていない．また，製品のリリース後も 26.3% から 63.5%しか SATD は返済（除去）されない [3] ことからも，SATD の多くはプロジェクトとして重大な関心事ではない可能性がある．そのため今後は，SATD ではなく課題票として管理される技術的負債について研究する価値が高いと言える．特に，課題票として管理される技術的負債を大量に収集しその特徴や返済方法を体系化することは，プロジェクトとして重要度の高い技術的負債の検出するツールの開発や，技術的負債の効率的・効果的な返済（自動化など）に貢献できる可能性がある．

2.2 技術的負債に関する課題票の分類

Xavier らの調査 [14] によると，GitHub にホスティングされたスター数上位 5000 件のプロジェクトのうち，課題票に技術的負債を示すラベルを付与して明示的に管理しているプロジェクトはわずか 23 件のみである．さらに，技術的負債を示すラベルを課題票に 10 件以上付与しているプロジェクトは，23 件のうち 4 件とさらに少なくなる．課題票に付与されたラベルに基づいて技術的負債を抽出することは技術的に容易であるが，特徴分析や体系化をおこなうには絶対数が不足している．ただし，技術的負債を課題票として管理しているプロジェクトがごく少数しか存在しないのではなく，技術的負債に高い関心が集まったのは比較的最近の現象であり，技術的負債の明確かつ具体的な定義は存在しないため，ほとんどのプロジェクトは技術的負債と不具合とを明確に区別していないのが実態である．

木村らは，ラベルを用いて明示的に管理していないが実質的には技術的負債を取り扱っている課題票をより広範囲に収集することを目的として，技術的負債に関する課題票とその他の課題票を分類する手法 [15] を提案している．[15] では，技術的負債を示すラベルが付与された課題票を収集し構築した分類モデルが，Precision 0.995, Recall 0.903 という高い分類性能を示しており，ラベルは付与されていないが実質的には技術的負債を扱っている課題票を今後大量収集できる可能性がある．

2.3 本研究の位置付け

木村らの分類モデル [15] は，課題票として管理される技術的負債の特徴や返済方法を研究するためのデータ収集を支援するために，ラベルは付与されていないが実質的に技術的負債を扱っている課題票とその他の課題票を分類することを目的としている．そのため，分類モデルの構築においては，すでに返済が完了した技術的負債

表 1. クリーニングにより除去したテキストと置換したテキストの対応

クリーニングの対象	除去のための正規表現	置換後のテキスト
Issue の ID	#[0-9]+	issueid
コードスニペット	“‘.*? “‘	code snippet
インラインコード	‘.*? ‘	inline code
画像	<img .*?>|!\[.*?\]\(.*?\)	image url
HTML タグ	<img(/)?.*?>	置換語なし
URL	http(s)?://.*?(\s|$)|\[.*\]\(.*?\)	link url
ファイル名	[^\s]+\.[a-z]+(\s|$)	filename
ディレクトリパス	/?[\.a-z0-9]+/.*?(\s|$)	pathname
日付	0-0-0(T|\s)	date number
時間	0:0:0(Z|\s)	time number
ユーザ名	@[a-z0-9]+	username
関数名	[^\s]+\(.*?\)	functionname
記号	[=\+\-&%#$,:;_*\'\”.\\/—||]+	置換語なし
数字	[0-9]+	0

(close された課題票) から様々な特徴量 (課題票に含まれるテキストの特徴，議論に参加する開発者の特徴，技術的負債を返済するために変更したコードの特徴など) を収集して利用している．

一方，本研究で構築する課題票の分類モデルは，課題票が登録された時点でその内容が技術的負債に関するものかそうでないのかを分類する．技術的負債の早期発見を支援することを意図したものであり，[15] の分類モデルとは目的が異なる．また，分類モデルの構築においては，木村らの分類モデルとは異なり，課題票が登録された時点で取得可能な特徴量 (図 1) のみを用いて分類モデルを構築する必要がある．課題票が登録された時点で取得可能な特徴量としては，課題票のタイトルや本文に含まれるテキストの特徴と課題票の登録者 (報告者) の特徴などが挙げられるが，本論文では分類モデルの基本性能を確認するために課題票に含まれるテキストデータのみを用いて分類モデルを構築する．その他の特徴量を用いた分類モデルの構築と分類性能の検証については今後の課題とする．

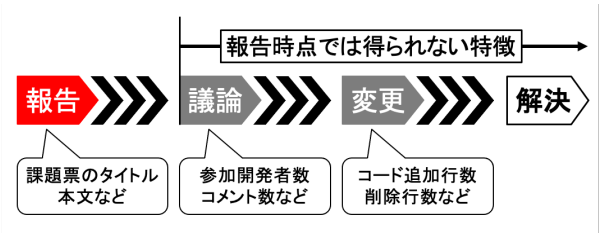

図 1. 課題が解決するまでの過程と得られる情報

3 分類モデルの構築

本章では，課題票が登録された時点でその内容が技術的負債であるかどうかを判別するための分類モデルの構築手順 (図 2) について説明する．

3.1 前処理

モデル構築に用いるテキストデータは自然言語で記述されているため，そのままでは計算機で取り扱うことができない．そのため，テキストデータを前処理 (正規表現によるクリーニング，トークン化とレンマ化) した上でベクトル化 (単語分散表現) する必要がある．以下では，前処理の手順に従ってそれぞれの内容を概説する．

3.1.1 クリーニング

課題票には，テキストデータを分散表現で扱う際にノイズとなるデータが含まれているため，正規表現によりノイズを除去し特定の単語の置換を行う．表 1 に，除去したノイズと置換した単語を示す．また，全てのテキストを小文字に変換し，大文字と小文字で単語を区別しないようにする．

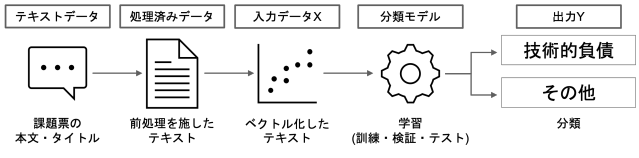

図 2. 分類モデル作成の手順

3.1.2 トークン化

クリーニングを行なったテキストデータを，単語ごとに分割することでトークン化する．トークン化には，spaCy 3.4.4[1]を利用する．トークン化に使用する辞書は，spaCy が公開している en_core_web_lg とする．トークン化した際に，それぞれのトークンは品詞ごとに分類され，品詞を記した POS（Part-of-speech）タグが付与される．POS タグが句読点（PUNCT），記号（SYM），空白（SPACE），その他（X）のトークンはノイズとなるため除去する．さらに，辞書に登録されていない単語（Out-of-Vocabulary: OOV）であるトークンは「oov」に置換する．また，英文では「it」や「the」のような単語が頻出する．ストップワードと呼ばれるこれらの頻出単語はノイズとなるため除去する．

3.1.3 レンマ化

英文では，動詞の活用や名詞の単数形と複数形などにおいて単語が変化するため，テキストデータのトークン化のあとは spaCy を用いて辞書の見出し語に統一するレンマ化を行う．レンマ化は，トークン化と同様に spaCy を用いて en_core_web_lg モデルを使用して変換する．

3.2 ベクトル化

上述の手順でノイズを除去したテキストデータを，計算機で扱える形式にするためにベクトル化を行う．具体的には，課題票に出現した単語のベクトルの平均値を求める．ベクトル化にあたっては，以下の 5 つの手法を採用した．

bag of words: BoW

BoW は，全テキストデータに含まれる単語の種数を次元数とし，文書（本研究では課題票）ごとに含まれる単語の数を要素とした単純な分散表現である．

TF*IDF

TF はある文書におけるある単語の出現頻度，IDF は全文書中のある単語を含む文書の珍しさ，つまり逆文書頻度を表す．BoW に TF と IDF を掛け合わせた重み付けをおこなうことで，文書ごとに異なる単語の重要度をより表現できる．

Word2Vec

Word2Vec [17] は，単語分散表現の代表的な手法とされており，2 層のニューラルネットワークからなるシンプルな構造を持つ．中心語から周囲語を確率的に予測する skip-gram モデルと，周囲語から中心語を確率的に予測する continuous BoW（CBOW）モデルの 2 種類の構造が実装されている．本論文では，頻度の低い単語に対して有用でありより精度が高いとされる skip-gram モデルを利用する．BoW や TF*IDF と大きく異なる点として，コーパスと呼ばれる自然言語の使用方法を構造化して大規模に集積したものを利用して，単語の数よりもはるかに低い次元のベクトル空間で単語の意味を表現できることが挙げられる．

Global Vector: Glove

GloVe [18] も Word2Vec と同様な単語分散表現の手法の一つである．Word2Vec では単語ごとに周囲語や中心語を予測していたが，GloVe ではコーパスから集積された単語のペアの共起確率を線形回帰により予測する．BoW や TF*IDF のような単語の出現頻度に基づいた手法と，Word2Vec のような確率的に単語を予測する手法を折衷した手法であるため，学習が早く大きなコーパスに対応でき，小さなコーパスや小さなベクトルでも優れた性能を示すとされている．

BoW と TF*IDF は scikit-learn 1.2.0[2]を用いて実装する．また，Word2Vec, GloVe については，Wikipedia から取集しレンマ化した文書データをコーパスとした学習済みモデル[3]を利用する．それぞれの学習済みモデルを本論文では，Word2Vec-Wiki, GloVe-Wiki と呼ぶこととする．また，Word2Vec に関しては，プログラミング技術に関する Q&A サイトである Stack Overflow から収集しレンマ化した文書データをコーパスとした学習済みモデル[4]（Word2Vec-SO と呼称する）も利用する．

3.3 分類器

分類器の作成にあたっては，代表的な教師あり分類アルゴリズムであるサポートベクターマシン，ロジスティック回帰，ランダムフォレストの 3 つの手法を利用する．いずれの手法も scikit-learn 1.2.0 を用いて実装する．

ランダムフォレスト

ランダムフォレストは，単体では分類性能の低い弱

[1] https://spacy.io/

[2] https://scikit-learn.org/stable/

[3] http://vectors.nlpl.eu/repository/

[4] https://zenodo.org/record/1199620#.Y7wiF-zP23L

表 2. データセットの概要

オーナー / リポジトリ名	TD ラベル	一般（件数）	TD（件数）
influxdata / influxdb	kind/tech-debt	8,514	120
saleor / saleor	technical debt	2,230	108
nextcloud / server	technical debts	7,255	83
microsoft / vscode	debt	79,829	2,044
合計	-	97,828	2,355

分類器を複数組み合わせることで高性能の分類器を作る，アンサンブル学習の一つである [19]．本論文では，学習時のハイパーパラメータは，決定木の本数 k を 200，決定木の分岐に使用する特徴の個数を全特徴の個数の平方根，ランダムシードを 42 に設定する．その他のハイパーパラメータは scikit-learn のデフォルト値を利用する．

ロジスティック回帰

ロジスティック回帰は，説明変数を集めた入力データ X から確率的にクラスに分類する統計的パターン認識手法の一つである [19]．本論文では，学習時のハイパーパラメータは，正則化項を L2 ノルム，反復回数を 2000，ランダムシードを 42 に設定する．その他のハイパーパラメータは scikit-learn のデフォルト値を利用する．

サポートベクターマシン

サポートベクターマシンは，入力データ X の各要素との距離，つまりマージンが最大となるようにデータ X を分割する m 次元の超平面を求めることで分類するパターン認識のための機械学習アルゴリズムである [19]．本論文では，学習時のハイパーパラメータは，正則化項を L2 ノルム，反復回数を 2000，ランダムシードを 42 に設定する．その他のハイパーパラメータは scikit-learn のデフォルト値を利用する．

4 分類性能評価実験

本章では，5 種類の単語分散表現と 3 種類の分類器を用いて構築した合計 15 種類の分類モデルの性能を比較するためにおこなった評価実験について述べる．

4.1 実験の目的

本実験の目的は，5 種類（BoW, TF-IDF, Word2Vec-Wiki, Word2Vec-SO, GloVe-Wiki）の単語分散表現と 3 種類の分類器（ランダムフォレスト，ロジスティック回帰，サポートベクターマシン）を用いて構築した合計 15 種類の分類モデルそれぞれの分類性能を定量的に示すとともに，単語分散表現が技術的負債に関連する課題票の分類に与える影響を明らかにすることである．本実験の対象とする課題票に含まれる文書は，多くが自然言語で記述される．そのため，先行研究でも用いられた BoW や TF*IDF のような単語の出現頻度に依存した単純な単語分散表現よりも，Word2Vec や GloVe のような単語同士の関係などを表現できる複雑な単語分散表現手法を用いた分類器がより高い性能を示すと予想する．

4.2 データセット

Xavier らの研究 [14] と同様に，GitHub 上のスター数が多い上位 5000 件のプロジェクトのうち，技術的負債を課題票として習慣的に管理している influxdb，saleor，server，vscode の 4 つのプロジェクトからデータを収集する．(1) GraphQL[5]を用いて 2022 年 4 月 26 日までに close された課題票をすべて収集した後，(2) 報告回数が 1 回のみの開発者による課題票を削除[6]し，(3) 残りの課題票をデータセットとした．表 2 に本実験で用いるデータセットの概要を示す．課題票が技術的負債（TD）に関するのなのか，その他一般に不具合等に関するものなのかについての判断は，プロジェクト毎に用意されているラベルに基づいて分類した．

4.3 実験手順

以下では，本評価実験の手順について説明する．

[5] https://graphql.org
[6] 報告自体に不慣れなため報告方法や内容に不備があることが多くノイズとなるため

4.3.1 データの正規化

3.1 節で説明した前処理を行なったデータセットについて，ロジスティック回帰やサポートベクターマシンのような線形分類モデルでは，外れ値の影響を受けることで訓練データに過剰に適合してしまう過学習が起こる可能性がある．そのため，本実験では分類モデルへの入力となるデータを，最大値が 1，最小値が 0 になるように正規化することで，過学習を防ぐ．正規化には次の式（1）を用いる．

$$X' = \frac{X - x_{min}}{x_{max} - x_{min}} \tag{1}$$

4.3.2 データセットの分割

正規化されたデータセットを入力とし，3 章の手順で作成する分類モデルの訓練と評価を行う．

4.3.3 層化 k 分割交差検証

訓練と評価を行う交差検証について，まず学習データを正例と負例が同じ比率になるよう k 等分する．本実験では，データセットの分割に scikit learn 1.2.0 を利用し，交差検証のための分割数はデフォルト値である $k = 5$ とした．次に，学習データを 5 等分したうちの 1 つを評価データ，残りを訓練データとするモデルの訓練と評価を行う．等分されたデータが，1 度ずつ評価データとなるように，訓練と評価を $k = 5$ 回繰り返す．これらの操作を層化 5 分割交差検証と呼びこれを 20 回おこない，訓練と評価をそれぞれ合計 100 回実行する．このような手順をとるのは，訓練データと評価データに分割した際のデータの偏りにより，不正確な結果となること，つまり過学習を防ぐためである．

4.4 評価指標

分類の予測結果と実際の正誤の関係について，技術的負債に関する課題票を正しく予測した場合を真陽性（True Positive: TP），技術的負債に関する課題票をその他の課題票であると誤って予測した場合を偽陰性（False Negative: FN），その他の課題票を技術的負債に関する課題票であると誤って予測した場合を偽陽性（False Positive: FP），その他の課題票を正しく予測した場合を真陰性（True Negative: TN）とし，分類性能を Accuracy（2），Precision（3），Recall（4），F1 値（5）を用いて評価する．それぞれの評価指標は，以下の式で求められる．

$$Accuracy = \frac{TP + TN}{TP + FP + FN + FP} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{5}$$

これらの指標が取りうる値の範囲は 0 から 1 である．4.3.3 節で示した手順により 100 回実行して得られた指標を平均し，有効数字 3 桁で示した値を最終的な評価値とする．

4.5 実験結果

表 3 に実験結果を示す．4 つの評価指標において最も高い値はそれぞれ太字で強調している．

表 3 より，作成した分類モデルのうち Accuracy が最も高かったのは，TF*IDF でベクトル化したロジスティック回帰で，0.978 を示した．ただし，Accuracy に関してはいずれの分類モデルもほぼ同等の高い値を示しておりモデル間で大差はない．

Precision が最も高い値を示したのは TF*IDF でベクトル化したランダムフォレストモデルで，0.896 であった．次いで高い値を示したのは BoW でベクトル化したランダムフォレストモデルで，0.880 であった．これら 2 種類の組み合わせについては他の組み合わせによる分類モデルよりも良好な結果を示していると言える．

Recall および F1 値に関しては，TF*IDF を用いたサポートベクターマシンモデルで，Recall で 0.240，F1 値で 0.341 とそれぞれ最も高い値を示した．ただし，0.240 という Recall の値は決して高くなく TP の見逃しが多数発生していることを示唆している．また，0.341 という F1 値も実用性の観点からは十分ではない．

本実験では，単語分散表現が分類モデルの性能に与える影響を調べるために 5 種類のベクトル化手法を用いたが，我々の当初の予想に反して BoW や TF*IDF を用いたシンプルな分類モデルの方が，より複雑なベクトル化手法である Word2Vec や GloVe を用いた分類モデルよりも性能が高かった．5 章では，追加調査を実施し得られた結果をより良く理解するための考察をおこなう．

表 3. 性能評価実験の結果

機械学習アルゴリズム	ベクトル化手法	Accuracy	Precision	Recall	F1
ランダムフォレスト	BoW	0.978	0.880	0.059	0.111
	TF*IDF	0.978	**0.896**	0.059	0.110
	Word2Vec-SO	0.977	0.768	0.018	0.035
	Word2Vec-Wiki	0.977	0.663	0.020	0.038
	GloVe-Wiki	0.977	0.639	0.002	0.038
ロジスティック回帰	BoW	0.977	0.662	0.014	0.028
	TF*IDF	**0.979**	0.736	0.156	0.257
	Word2Vec-SO	0.977	0.518	0.113	0.185
	Word2Vec-Wiki	0.977	0.360	0.005	0.010
	GloVe-Wiki	0.977	0.495	0.071	0.124
サポートベクターマシン	BoW	0.977	0.520	0.092	0.156
	TF*IDF	0.978	0.589	**0.240**	**0.341**
	Word2Vec-SO	0.977	0.652	0.019	0.037
	Word2Vec-Wiki	0.977	0.563	0.006	0.013
	GloVe-Wiki	0.977	0.624	0.020	0.039

表 4. 重要度の高い上位 5 件の単語

ベクトル化	重要度の高い単語				
BoW	oov	unique	inline	link	debt
TF*IDF	oov	link	unique	code	debt

oov：spaCy の辞書外の単語であることを示す
unique：固有名詞であることを示す
inline：プログラムコードであることを示す
link：リンクであることを示す

5 考察

本章では，分類モデルの性能評価実験で得られた結果について考察する．

5.1 分類に寄与する単語の調査

表 3 より，Precision が最も高い値を示したのは TF*IDF でベクトル化したランダムフォレストモデルで，0.896 であった．次いで高い値を示したのは BoW でベクトル化したランダムフォレストモデルで，0.880 であった．このような結果が得られた理由を知るために，ランダムフォレストモデルで得られる単語の重要度 $M(T)$ について，上位の単語 5 つを調査した．調査結果を表 4 に示す．重要度の高さは，ランダムフォレストに基づく分類モデルを評価する際に合計 100 回学習した過程で算出された結果から集計した表 4 より，BoW と TF*IDF で共に，前処理のレンマ化ができない辞書外の単語（oov）や固有名詞（unique），プログラムコード（inline），リンク（link），技術的負債を示す単語「debt」などの重要度が高いことがわかった．これらの単語のうち，特に辞書外の単語や固有名詞については，複雑なベクトル化手法を使用した場合，置き換えた後の単語の特徴をうまく捉えたベクトル化ができなかった可能性が考えられる．

5.2 不均衡データへの考慮

表 3 から結果を俯瞰すると，Accuracy と Precision が高く，Recall が低い分類モデルを作成できた．これは，分類モデルが技術的負債であると予測した課題票については正しく分類できたが，実際には予測の取りこぼしが多かったことを示している．Precision が高く Recall が低い理由としては，正例と負例のデータの偏りが考えられる．表 2 のとおり，技術的負債以外を扱う課題票が技術的負債を扱う課題票の約 40 倍以上あり，正例が非常に少ない．これにより，技術的負債を扱っている課題票の特徴を掴めず，技術的負債を扱っている課題票の多くをその他の課題票であると予測している可能性がある．

そこで，それぞれの分類モデルを学習させる際に，ハイパーパラメータの class_weight を balanced に設定することで分類するクラスの偏りに応じて重み付けを行い，再度実験を実施した．表 5 に実験結果示す．

表 5 より，作成した分類モデルのうち，Accuracy が最も高かったのは，BoW と TF*IDF でベクトル化したランダムフォレストモデルで，同率の最高値の 0.978 を示した．Precision で最も高い値を示したのは，TF*IDF でベクトル化したランダムフォレストモデルで，値は 0.857 であった．さらに，Word2Vec-SO でベクトル化し

表 5. 正例データに重み付けした場合の結果

機械学習アルゴリズム	ベクトル化手法	Accuracy	Precision	Recall	F1
ランダムフォレスト	BoW	**0.978**	0.855	0.064	0.118
	TF*IDF	**0.978**	**0.857**	0.050	0.095
	Word2Vec-SO	0.977	0.749	0.029	0.056
	Word2Vec-Wiki	0.977	0.798	0.022	0.042
	GloVe-Wiki	0.977	0.786	0.020	0.039
ロジスティック回帰	BoW	0.827	0.094	0.738	0.166
	TF*IDF	0.916	0.017	0.672	**0.272**
	Word2Vec-SO	0.834	0.101	**0.776**	0.179
	Word2Vec-Wiki	0.821	0.094	0.768	0.167
	GloVe-Wiki	0.831	0.099	0.766	0.175
サポートベクターマシン	BoW	0.872	0.112	0.647	0.190
	TF*IDF	0.928	0.172	0.546	0.261
	Word2Vec-SO	0.906	0.155	0.669	0.252
	Word2Vec-Wiki	0.834	0.100	0.770	0.178
	GloVe-Wiki	0.837	0.101	0.759	0.179

表 6. 重要度の高い上位 5 件の単語（重み付け後）

ベクトル化	重要度の高い単語				
BoW	image	version	unique	vscode	os
TF*IDF	image	version	link	unique	os

image：画像であることを示す

たロジスティック回帰モデルは，Recall で最高値 0.776 を示した．最後に，TF*IDF を用いたロジスティック回帰モデルは，最も高い F1 値として 0.272 を示した．クラスに重み付けする前と結果を比較すると，分類モデルにより増減幅の差はあるものの，ほとんどの分類モデルで Precision が低くなり，また多くの分類モデルで Recall が高くなった．また，F1 値の最高値は，クラスに重み付けする前の方が高かった．

クラスに重み付けすることで Recall が Precision をよりも大きく上回るようになったロジスティック回帰とサポートベクターマシンに基づく分類モデルは，正例を比較的取りこぼさずに予測できるようになったが，偽陽性を多く含むようになったことを示している．一方，ランダムフォレストを用いた分類モデルでは，クラスに重み付けしても結果がほとんど変わらなかった．

クラスに重み付けすることにより重要な単語が変わる可能性があるため，単語の重要度を調査した結果を表 6 に示す．表 6 に示す通り，クラスに重み付けしたことにより重要な単語が変化した．表 4 と表 6 を比較すると，クラスに重み付けすることで辞書外の単語やプログラムコードなどの重要度よりも，画像や「version」などの既存の単語の重要度が高くなるという変化があった．

以上の結果から，BoW や TF*IDF 以外の分散表現を用いた分類モデルの性能を向上させるためには，辞書外の単語について前処理の改善を行う必要があることがわかった．また，学習データにおける正例・負例を均衡させるためにクラスに重みを付けた分類モデルは，Recall を改善できることがわかった．そのため，はじめに作成した Precision が高い値を示す分類モデルは，量は期待できないが，技術的負債を取り扱う課題票を正しく収集する際に利用できる．また，追加調査で作成した Recall が高い値を示す分類モデルは，正しさは保証できないが，技術的負債を取り扱う課題票の取りこぼしがないように収集するときに利用できると言える．

5.3 妥当性

5.3.1 内的妥当性

本論文では，「debt」等ラベルが付与された課題票を技術的負債を管理するための課題票として定義した．技術的負債を管理するためのその他のラベルが利用されている可能性があり，実験結果に影響している可能性がある．また，本論文ではレンマ化において spaCy を利用しているが，辞書外に特徴量となる単語（OOV など）が多数あることが分かったため，辞書外の単語の置き換えについて再考する必要がある．また，より新しい単語分散表現手法の RNN や CNN，そして最新の単語分散表現手法である Transformer やそれをベースとした BERT や GPT などについても比較することで，より広く分類性能への影響を分析できるため，今後の課題としたい．

表 7. 本実験で利用した Issue とそのプロジェクトの情報

オーナー / リポジトリ名	主な開発言語	アプリケーションの概要
influxdata / influxdb	Go	時系列データベース
saleor / saleor	Python	EC サイト構築のプラットフォーム
nextcloud / server	PHP	オンラインストレージ
microsoft / vscode	JavaScript	ソースコードエディタ

5.3.2 外的妥当性

技術的負債を課題票として扱うプロジェクトが未だ少ないことから，本実験が一般化できるかについては不明である．しかし，表 7 に示す通り，本論文のデータセットとした 4 つのプロジェクトはそれぞれ異なる言語と目的で開発されている．そのため，さまざまな分野のプロジェクトにおいて本論文で作成した分類モデルをある程度適用させることができると考える．今後の課題として，本実験のデータセットには含まれなかった分野のプロジェクトについてデータを収集し検証する必要があると考える．

6 関連研究

6.1 ソースコードコメントに表明される技術的負債

Potdar ら [3] は，開発者が技術的負債を意識的に管理する方法として，ソースコード中にコメントで表明する技術的負債（SATD: Self-Admitted Technical Debt）の存在を明らかにした．ソースコード中に記述されるコメントから技術的負債を特定できることから，近年 SATD の検出に関する研究が盛んに行われている．Potdar ら [3] は，SATD-Comment に含まれる，技術的負債を示す 62 種類のコメントパターンを特定した．また，2.4%から 31.0%のソースコードファイルに SATD が含まれること，主に経験豊富な開発者が活動全体で技術的負債を導入すること，製品の公開までの期間やコードの複雑さと技術的負債の間に強い相関は存在しないこと，そして製品の公開後も 26.3%から 63.5%しか技術的負債が除去されないことについても明らかにした．その後，ソースコードコメントに基づいて技術的負債を検出する手法 [4–7] が研究された．さらに，単語分散表現などの自然言語処理を用いてさらに検出精度を高める研究 [8–13] なども進められている．

6.2 課題票として管理される技術的負債

技術的負債は，課題票として管理される事例があることが Xarvier らの研究 [14] によって明らかになった．木村らの先行研究 [15] では，課題票として管理される技術的負債の特徴分析に基づいて技術的負債に関する課題票を分類する手法を構築している．本研究は，課題票として管理される技術的負債の早期検出を支援するために，課題票に含まれるテキストのみを入力とした分類モデルを作成し，その性能について比較評価した．

7 おわりに

本論文では，課題管理システムに登録された課題票からタイトルと本文のテキストデータを収集し，5 種類の単語分散表現と 3 種類の機械学習アルゴリズムを用いて，技術的負債として管理すべき課題票を分類するモデルを作成した．性能比較実験の結果，BoW や TF*IDF といったシンプルな単語分散表現を用いた分類モデルが高い Precision を示すことがわかった．また，学習データに含まれる正例・負例を均衡させるように学習させた場合，ロジスティック回帰モデルとサポートベクタマシンモデルを用いた分類モデルで Recall が大きく改善することを確認した．

BoW や TF*IDF 以外の単語分散表現については，本実験の設定では特徴を掴めなかった単語への前処理の工夫を行うほか，文脈を把握し未知語へのより柔軟な対応が可能である最新の単語分散表現を加えることで，更なる分類モデルの性能向上を目指す．さらに今後は，設計に関する技術的負債やテストに関する技術的負債など，技術的負債の種類ごとに分類するモデルを構築する予定である．

謝辞

本研究の一部は，文部科学省科学研究補助金（基盤（C）：22K11974）による助成を受けた．

参考文献

[1] Ward Cunningham. The wycash portfolio management system. *SIGPLAN OOPS Mess.*, Vol. 4, No. 2, pp. 29–30, 1992.

[2] Zengyang Li, Paris Avgeriou, and Peng Liang. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, Vol. 101, No. C, pp. 193–220, 2015.

[3] Aniket Potdar and Emad Shihab. An exploratory study on self-admitted technical debt. In *Proceedings of the International Conference on Software Maintenance and Evolution*, ICSME, pp. 91–100, October 2014.

[4] Xiaoxue Ren, Zhenchang Xing, Xin Xia, David Lo, Xinyu Wang, and John Grundy. Neural network-based detection of self-admitted technical debt: From performance to explainability. *ACM Transactions on Software Engineering and Methodology*, Vol. 28, No. 3, 2019.

[5] Zhe Yu, Fahmid Morshed Fahid, Huy Tu, and Tim Menzies. Identifying self-admitted technical debts with jitterbug: A two-step approach. *IEEE Transactions on Software Engineering*, Vol. 48, No. 5, pp. 1676–1691, 2022.

[6] Irene Sala, Antonela Tommasel, and Francesca Arcelli Fontana. Debthunter: A machine learning-based approach for detecting self-admitted technical debt. In *Proceedings of the Evaluation and Assessment in Software Engineering*, EASE, pp. 278–283, June 2021.

[7] Murali Sridharan, Mika Mäntylä, Maëlick Claes, and Leevi Rantala. Soccminer: A source code-comments and comment-context miner. In *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR, pp. 242–246, May 2022.

[8] Jernej Flisar and Vili Podgorelec. Enhanced feature selection using word embeddings for self-admitted technical debt identification. In *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA, pp. 230–233, August 2018.

[9] Supatsara Wattanakriengkrai, Rungroj Maipradit, Hideki Hata, Morakot Choetkiertikul, Thanwadee Sunetnanta, and Kenichi Matsumoto. Identifying design and requirement self-admitted technical debt using n-gram idf. In *Proceedings of the 9th International Workshop on Empirical Software Engineering in Practice*, IWESEP, pp. 7–12, December 2018.

[10] Jernej Flisar and Vili Podgorelec. Identification of self-admitted technical debt using enhanced feature selection based on word embedding. *IEEE Access*, Vol. 7, pp. 106475–106494, 2019.

[11] Fiorella Zampetti, Alexander Serebrenik, and Massimiliano Di Penta. Automatically learning patterns for self-admitted technical debt removal. In *Proceedings of the 27th International Conference on Software Analysis, Evolution and Reengineering*, SANER, pp. 355–366, February 2020.

[12] Leevi Rantala. Towards better technical debt detection with nlp and machine learning methods. In *Proceedings of the 2nd International Conference on Software Engineering: Companion Proceedings*, ICSE, pp. 242–245, October 2020.

[13] Amleto Di Salle, Alessandra Rota, Phuong T. Nguyen, Davide Di Ruscio, Francesca Arcelli Fontana, and Irene Sala. Pilot: Synergy between text processing and neural networks to detect self-admitted technical debt. In *Proceedings of the International Conference on Technical Debt*, TechDebt, pp. 41–45, May 2022.

[14] Laerte Xavier, Fabio Ferreira, Rodrigo Brito, and Marco Tulio Valente. Beyond the Code: Mining Self-Admitted Technical Debt in Issue Tracker Systems. In *Proceedings of the 17th International Conference on Mining Software Repositories*, MSR, pp. 137–146, June 2020.

[15] 木村祐太, 大平雅雄. 技術的負債に関連する課題票分類手法の構築. 情報処理学会論文誌, Vol. 64, No. 1, pp. 2–12, 2023.

[16] Emerson Murphy-Hill, Chris Parnin, and Andrew P. Black. How we refactor, and how we know it. *IEEE Transactions on Software Engineering*, Vol. 38, No. 1, pp. 5–18, 2012.

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

[18] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, 2014.

[19] Christopher M. Bishop, 元田浩, 栗田多喜夫, 樋口知之, 松本裕治, 村田昇. パターン認識と機械学習：ベイズ理論による統計的予測. 丸善出版, 2012.

[20] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information, 2016.

[21] Pei Wang, Jinqiu Yang, Lin Tan, Robert Kroeger, and J. David Morgenthaler. Generating precise dependencies for large software. In *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD, pp. 47–50, May 2013.