

技術的負債に関連する 課題票抽出手法の構築

オープンソースソフトウェア工学研究室

学籍番号：60210032

氏名：木村 祐太

技術的負債：Technical Debt^[1]

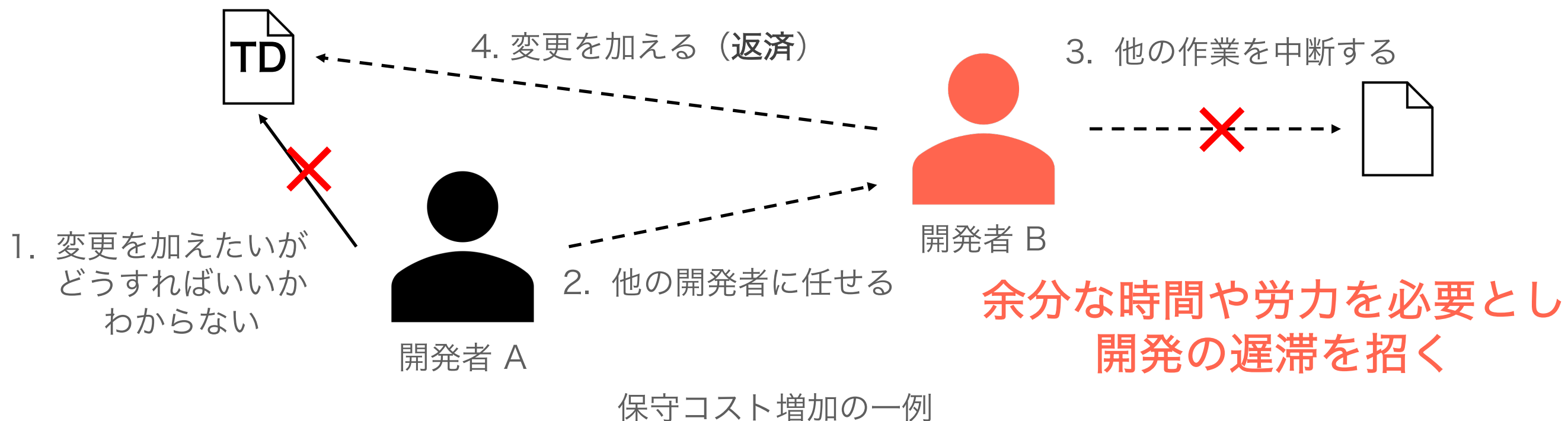
- 短期的には有用，長期的には問題がある実装や設計

例) レガシーコード：かなり前に実装されたコード

コードスメル：深刻な問題を抱えているコードの特徴

- 保守コストの増加を引き起こす^[2]

- 可読性の低いコードや複雑な構造を持つコードなどが原因



[1] Cunningham, W., 1992. The wycash portfolio management system. Proc. Obj. Oriented Programm. Syst. Lang. Appl. 4 (2), pp. 29-30.

[2] C. Seaman, G. Yuepu, N. Zazworka, F. Shull, C. Izurieta, C. Yuanfang, et al., "Using technical debt data in decision making: Potential decision approaches," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 45-48.

TDの影響とマネジメント

- 開発生産性の低下（余分な作業の増加）^[3]
 - 開発の約23～35%が余分な作業^{[4][5]}
 - 既存のTDによって新規にTDが導入される^[4]
 - 開発資源（金銭的，人的，時間的）には限界があるため，蓄積するTDを管理することが必要
- TDマネジメント研究の需要の高まり^[6]
 - TDを明示的に管理する方法を研究
 - TDインスタンス検出ツール
 - 自動優先順位付け

[3] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," Journal of Systems and Software, vol. 86, no. 6, 2013, pp. 1498-1516.

[4] T. Besker, A. Martini and J. Bosch, "Technical Debt Cripples Software Developer Productivity: A Longitudinal Study on Developers' Daily Software Development Work," 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), 2018, pp. 105-114.

[5] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt: When and by whom will it be paid?," in IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, pp. 13-23.

[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Journal of Systems and Software, vol. 101, 2015, pp. 193-220.

TDマネジメント研究の課題^[6]

定義されているTDタイプ

Code, Design	UI
Tests	Performance
Infrastructure	Documentation
Code Style	Build
Security	Requirement

1. 静的解析ツールでソースコードに関連するTDは自動特定できるが、それ以外特定できない
 - 特定可能：コード，テスト
 - 特定不可能：アーキテクチャ，インフラ，など
2. さまざまなTDが存在するが，各TD間の関係が調査されていない

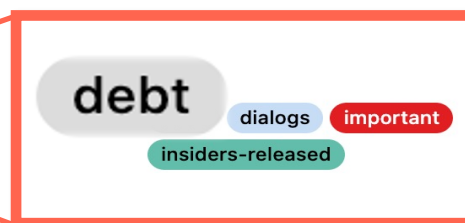
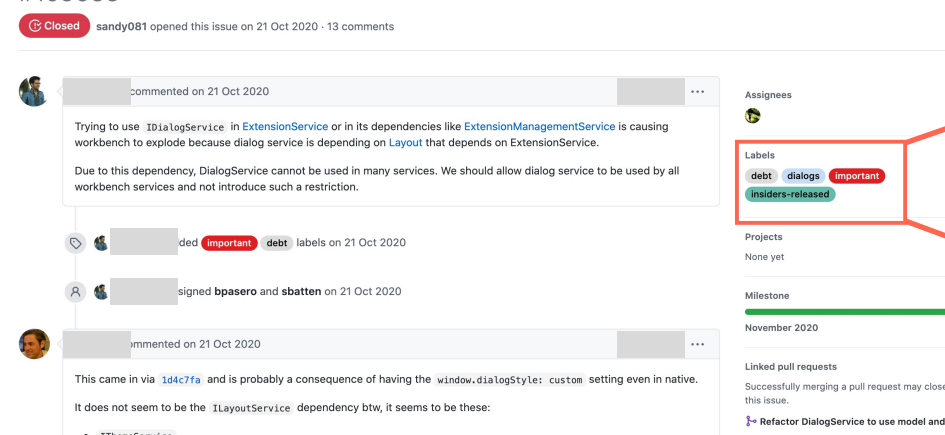
[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Journal of Systems and Software, vol. 101, 2015, pp. 193-220.

Self-Admitted TD

- 何らかの方法で開発者によって明示されるTD
 - SATD-Code ^[7]
 - ソースコード中にソースコードコメントを用いて明示されるSATD
 - 従来のSATD研究の対象
 - 例) *// TODO: - This method is too complex, lets break it up.*
 - SATD-Issue ^[8]
 - 課題票 (Issue) にTDラベルを割り当てて明示されるSATD

Cannot use IDialogService in some services due to custom dialog service dependencies
#109056

引用 : <https://github.com/microsoft/vscode/issues/109056>



“debt”というTDラベル
が割り当てられている

[7] Potdar, A., Shihab, E., An exploratory study on self-admitted technical debt. In: Proceedings of the 30th International Conference on Software Maintenance and Evolution. IEEE, 2014, pp. 91–100.

[8] 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 '20). 2020, pp. 137–146.

Issue

- GitHub * (ソーシャルコーディング環境) に実装されている機能
- 開発されている製品の不具合や機能改善など報告を管理している

ステータス

Open : 未解決
Closed : 解決済

Cannot use IDialogService in some services due to custom dialog service dependencies

#109056

Closed

sandy081 opened this issue on

報告内容
(Markdownで記述)



commented on 21 Oct 2020

Trying to use `IDialogService` in `ExtensionService` or in its dependencies like `ExtensionManagementService` is causing workbench to explode because dialog service is depending on `Layout` that depends on `ExtensionService`.

Due to this dependency, `DialogService` cannot be used in many services. We should allow dialog service to be used by all workbench services and not introduce such a restriction.



added `important` `debt` labels on 21 Oct 2020



assigned `bpasero` and `sbatten` on 21 Oct 2020



commented on 21 Oct 2020

This came in via `1d4c7fa` and is probably a consequence of having the `window.dialogStyle: custom` setting even in native.

It does not seem to be the `ILayoutService` dependency btw, it seems to be these:

- `IThemeService`

Assignees



Labels

`debt` `dialogs` `important`
`insiders-released`

Projects

None yet

Milestone

November 2020

Linked pull requests

Successfully merging a pull request may close this issue.

[Refactor DialogService to use model and c...](#)

内容に関連するラベル

引用 : <https://github.com/microsoft/vscode/issues/109056>

* <https://github.com>

SATD-Issueの特徴

TDマネジメント研究の課題^[6]

定義されているTDタイプ

Code, Design	UI
Tests	Performance
Infrastructure	Documentation
Code Style	Build
Security	Requirement

1. 静的解析ツールでソースコードに関連するTDは自動特定できるが、それ以外特定できない
 - 特定可能：コード、テスト
 - 特定不可能：アーキテクチャ、インフラ、など
2. さまざまなTDが存在するが、各TD間の関係が調査されていない

SATD-Issueで扱われるTDタイプ^[8]

Design	UI
Tests	Performance
Infrastructure	Documentation
Code Style	Build
Security	Requirement

 : SATD-Codeで扱われている

- これまで自動特定できなかったTDタイプをカバー
 - SATD-Codeでも多数のTDタイプがカバーされているが、カバーしきれないTDタイプも存在する
 - 自動特定の足掛かりとなる可能性

[8] 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 '20). 2020, pp. 137–146.

SATD-Issue研究の問題点

問題点

- ・ 慣習的に扱われていないためデータが少ない



仮説

- ・ 実務者にTDと認識されていないが、TDに関連するIssueが存在するのではないか？
 - － 企業開発者60人のうち「体系的にTDの管理をおこなっている」と回答したのは10人（**約17%**）^[9]

[9] Justus Bogner, Jonas Fritsch, Stefan Wagner, and Alfred Zimmermann. Limiting technical debt with maintainability assurance: an industry survey on used techniques and differences with service- and microservice-based systems. In Proceedings of the 2018 International Conference on Technical Debt (TechDebt '18), 2018, pp. 125-133.

本研究の目的とアプローチ

目的

- TD-Issueを自動抽出すること
 - ※ TD-Issue：開発者が明示していないTDに関連するIssue

アプローチ

- TD-Issueを抽出する分類モデルの構築
 1. ランダムフォレストを用いた分類モデルの構築
 - ・ 特徴量の組合わせを変えて複数モデルを構築
 2. ケーススタディで分類モデルの評価
 - ・ 分類精度評価
 - ・ 効果的な特徴量 ← 今後の予定
 - ・ 他プロジェクトを利用した一般性の検証 ← 今後の予定

データセット

- SATD-Issueを慣習的に扱っている4プロジェクトをGitHubから抽出
 - VSCode, Influxdb, Saleor, Server
- データ数が最も多かったプロジェクトをモデル構築に利用
 - 課題内容が確定しているClosed Issue（解決済み）を対象

プロジェクト	SATD-Issue [件]（正例）	通常のIssue [件]（負例）
vscode	1,612	102,301

- 残りの3プロジェクトは、今後の一般性の検証に用いる予定

入力する特徴量の検討

- Closed Issueから取得できる特徴量を抽出
 - TD-Issue分類に関する研究が存在しないため、有効な特徴量が不明
- Issue発見から解決までの工程における構成要素に着目



Issue発見から解決までの工程

特徴量の抽出 (1/4)



- 報告者
 - Issueを発見・報告する開発者

(仮説) 開発経験が豊富な開発者ほどTDを認識している

SATD-Code研究 [7]で, SATDコメントはベテラン開発者のほうが導入させる傾向にある

→ 報告者のこれまでの開発経験

- 経験年数, Issue報告数, コミット数 (変更追加回数)
- 特徴量次元数: 3次元

特徴量の抽出 (2/4)



- テキスト
 - Issueに記述されるテキスト

(仮説) 報告されるIssue内容が異なる

→ Issue内容を記述した自然言語 (英語)

- Issueタイトル, 本文
- 特徴量次元数: 46989次元

特徴量の抽出 (3/4)



- プロセス
 - Issueの報告から解決までにおこなわれる活動情報

(仮説) 通常のIssueに比べて解決にコストがかかる

SATD-Issueの方が解決されるまでに時間がかかる [8]

→ Issue解決に至るまでの活動に伴う情報

- 本文の文字数, 開発者数, 解決までにかかる時間, など
- 特徴量次元数: 7次元

[8] 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 '20). 2020, pp. 137–146.

特徴量の抽出 (4/4)



- ソースコード
 - Issue解決のためにおこなわれるソースコードの変更

(仮説) 主にリファクタリングがおこなわれる

既存のTD検出ツールはソースコードベースでおこなわれる [6]

→ 変更によるソースコードメトリクスの変化

- コード行数の増減, サイクロマティック複雑度の増減, など
- 特徴量次元数: 18次元

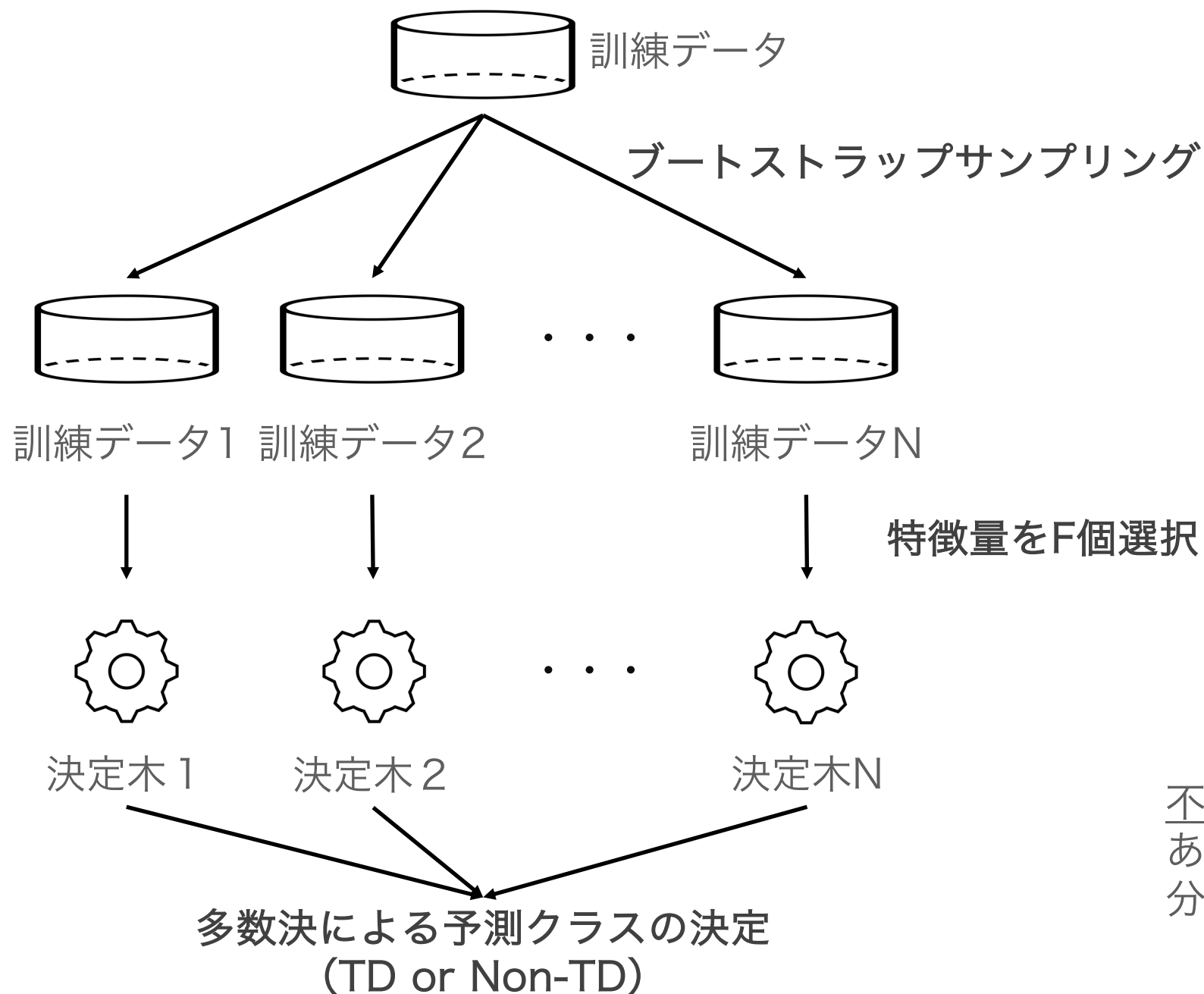
[6] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," Journal of Systems and Software, vol. 101, 2015, pp. 193-220.

テキストの前処理

- テキストデータは数値にする必要があるため、以下のステップで変換処理を行う
 1. ノイズとなるテキストの除去
 - ・ インラインコード, Markdownタグ, など
 2. 1-gram (1単語) に分割
 3. ストップワードの除去
 4. 単語の正規化
 - ・ 小文字統一, 語幹抽出
 5. 1つのIssueにのみ出現する単語の除去
 6. 単語のベクトル化
 - ・ Bag of Wordsを利用

学習アルゴリズム

• ランダムフォレスト^[10]



ハイパーパラメータの設定^[11]

- 決定木の本数: $N = 200$
- 特徴量の個数: $F = \sqrt{M}$
 M は特徴量の全個数

重み付け

- ジニ不純度に対してデータ件数の逆数を重み付け

不純度

あるノードにおいてどのくらいデータが分類できなかったのかを表す指標

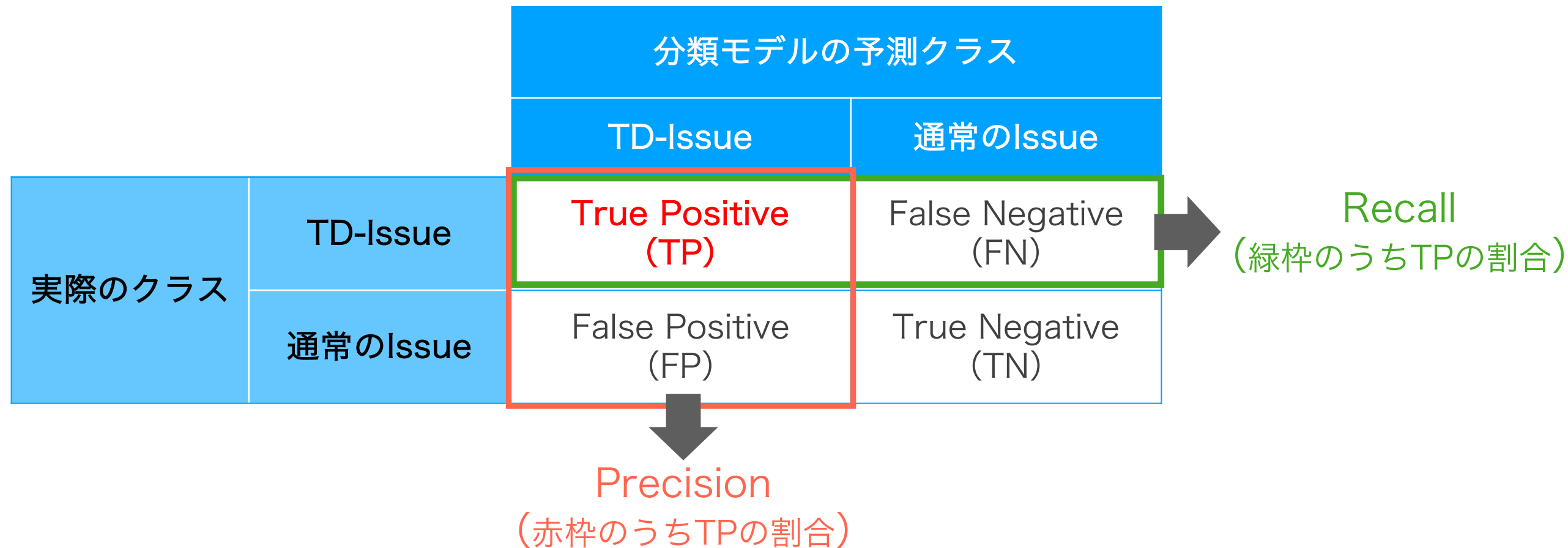
[10] Leo Breiman. 2001. Random Forests. Mach. Learn. 45, 1, October 1, 2001, pp. 5–32.

[11] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

ケーススタディ

- 分類モデル（全 10モデル）
 - テキスト
 - テキスト + 1（報告者, プロセス, ソースコード）
 - テキスト + 2（報告者, プロセス, ソースコード）
 - テキスト + 報告者 + プロセス + ソースコード
- 方法
 - 層化10分割交差検証を10回試行（計 100回）
 - Precision, Recallの平均値を算出

ケーススタディ：評価指標



$Precision = \frac{TP}{TP+FP}$... TD-Issueと分類して、実際にTD-Issueであった割合

$Recall = \frac{TP}{TP+FN}$... 実際のTD-Issueのうち、分類でTD-Issueを特定した割合

ケーススタディ：結果

T：テキスト， R：報告者， P：プロセス， C：ソースコード

モデル	Precision	Recall
T	0.758	0.086
T+R	0.872	0.095
T+P	0.875	0.088
T+C	0.807	0.092
T+R+P	0.916	0.110
T+R+C	0.852	0.107
T+P+C	0.869	0.094
T+R+P+C	0.908	0.120

- テキストのみと比較すると全体的に向上している
- Precisionは高いが， 全体的にRecallが低い

考察：Recallが低い要因

- ベースとなる，テキストのみの分類モデルのRecallが低い

モデル	Precision	Recall
T	0.758	0.086

- TD-Issueを表す単語の特徴量が少ない
→ バグのような原因がはっきりしているIssueに比べて，TD-Issueは曖昧に記述される or 記述が少ない？

- 単語の特徴量に対する重み付け
- ノイズとなる単語の特定と削除
- タイトル，本文のどちらかに限定して利用

- Recallが低い要因について調査が必要

今後の予定 (1/2)

- モデルの精度 (Recall) 改善
 - データレベル
 - Recallを下げる要因の調査
 - アンダーサンプリングやSMOTEなどを利用した不均衡データへの対処
 - モデルレベル
 - ハイパーパラメータの調整
 - 分類閾値の調整
- 他プロジェクトを利用した一般性の検証
 - Precision, Recallによる定量的評価
 - False PositiveとなるIssueの定性的分析
 - False Positive : TD-Issueと分類された通常のIssue

今後の予定 (2/2)

- 学会発表
 - FOSE 2021 (7月初旬 締切)
 - SATD-Issueの分析に関する調査で論文投稿予定
 - ICSE 2022 (8月下旬 締切)
 - TD-Issueの自動抽出手法に関する調査で論文投稿予定