修 士 論 文

題　目　　リリースを考慮したソフトウェアの
不具合予測精度にSMOTEが与える影響の分析

主任指導教員　　水野 修　教授

指導教員　　崔 恩瀞　助教

京都工芸繊維大学大学院 工芸科学研究科

情報工学専攻

学生番号　　20622016

氏　　名　　里形　洋道

令和4年2月7日提出

様式 2 号

学位論文内容の要旨（和文）

令和 4 年 2 月 7 日

京都工芸繊維大学大学院
工芸科学研究科長　殿

工芸科学研究科　　情報工学専攻
令和 2 年入学
学生番号　　20622016
氏　　名　　里形　洋道　㊞

（主任指導教員　　水野 修　㊞　）

本学学位規則第 4 条に基づき、下記のとおり学位論文内容の要旨を提出いたします。

1.　論文題目

リリースを考慮したソフトウェアの不具合予測精度に SMOTE が与える影響の分析

2.　論文内容の要旨（400 字程度）

ソフトウェアの不具合予測の 1 つの手法として,「変更レベルの不具合予測（以下, JIT DP）」がある．これは，変更管理システムにおけるコミットの単位で不具合が含まれているかを機械学習などを用いて予測する手法である．通常，JIT DP では，十重交差検証などの実験手法を用いるので，コミットがなされた時系列は考慮されない．そのため，我々は先行研究において，ソフトウェアリリースを時系列と仮定して，リリースを考慮した JIT DP を提案した．しかし，リリースを考慮した JIT DP においてはデータが不足する区間が発生するなどの問題があった．

本研究では，リリースを考慮した JIT DP におけるデータ不足の問題を解決するために，Synthetic Minority Over-sampling Technique（以下，SMOTE）と呼ばれるオーバーサンプリング手法を用い，それが予測精度にどのような影響を与えるかの分析を実施した．分析の結果，SMOTE はリリースを考慮した JIT DP での予測精度を向上させることがわかった．さらに，リポジトリの規模に関わらず予測精度を向上させることがわかった．また，大規模なリポジトリと小規模なリポジトリに対する SMOTE の効果量を測定したところ，大規模なリポジトリにおける影響の方が大きいことがわかった．この結果から，SMOTE の適用は，JIT DP において大きな精度向上をもたらすことが示された．

Analyzing the impact of SMOTE on software defect prediction accuracy between software releases

2022 20622016 *SATOGATA Hiromichi*

Abstract

"Just-In-Time defect prediction (JIT DP)" is known as a method of software defect prediction. This is a method of predicting whether a defect is infected in a commit unit of a change management system by using machine learning technique.

Because previous JIT DPs did not consider time series information, we proposed a release-aware JIT DP, which considers the release of software assuming that the software release is in chronological order in the previous research. However, in the release-aware JIT DP, there was a problem because of insufficient amount of data.

In this study, we used an oversampling technique called Synthetic Minority Over-sampling Technique (SMOTE) to solve the problem of data shortage in the release-aware JIT DP. We analyzed how SMOTE affects the prediction accuracy using the data of well-known open souse software. As a result of the analysis, it was found that SMOTE improves the prediction accuracy the release-aware JIT DP. Furthermore, it was found that the prediction accuracy was improved regardless of the size of the repository. We also investigated the effect size of SMOTE on large and small repositories and found that SMOTE has greater impact on large repositories. From this result, it was shown that the application of SMOTE brings a great improvement in accuracy in the release-aware JIT DP method.

目　次

1. 緒言

ソフトウェアを開発する上で，ソフトウェアの品質を高く保つことは広く求められている．その一例として不具合の除去があり，ソフトウェア開発の現場では不具合を取り除くことに多くの労力や時間などのリソースが割かれている．不具合の除去を目的とした技術として不具合予測がある．不具合予測は，ソフトウェアに不具合が含まれているか否かを機械学習などを用いて予測する．不具合予測の中でも，変更管理システムにおけるコミットに重点を置いて不具合予測を行う変更レベルの不具合予測手法（Just-In-Time Deffect Predction，以下 JIT DP）が提案されている [1]．これは，ソフトウェアの変更ごとに不具合が含まれているかを予測する点において，ファイルレベルの不具合予測 [2, 3] などと比べて粒度が細かいものとなっている．具体的には，コミット単位で不具合を含むか否かを予測する．

JIT DP は，ソフトウェアを開発する上での不具合の特徴が変化しないものと仮定している．その仮定より，過去に行われた変更とその不具合の特徴を用いて学習データを作成し，予測モデルを訓練し不具合予測を行っている．しかし，McIntosh らは不具合を予測する上で重要な特徴が，一定期間ごとに異なることを指摘している [4]．我々は，その一定期間に当てはまるものが，ソフトウェア開発の活動の 1 つであるリリースが該当するのではないかと仮定した．そのため，リリースを考慮した JIT DP は，より正確に不具合を予測することができるのではないかと仮説を立てた．我々は過去に研究で，ソフトウェアリリースを考慮した変更レベルの不具合予測の検証という研究を行った [5]．その結果，3 つのことがわかった．1 つ目がリリース毎に訓練データを分けて不具合予測を行った場合と，あるリリース以前のデータを訓練データとして不具合予測を行った場合，後者のモデルの方が予測精度は比較的良かったこと．2 つ目が，リリース毎に訓練データを作成した場合，訓練データに用いる不具合ラベルのデータが不足する場合が発生すること．3 つ目が，修正のタイミングを考慮しさらに訓練データを制限した場合，より顕著に不具合ラベルの訓練データが不足したこと．解決策として，より大きなデータセットを用いることや，オーバーサンプリングなどを用いてデータセットを増強することなどが挙げられた．

そのため，本研究では，オーバーサンプリング手法の一つである Synthetic Minority Over-sampling Technique（以降 SMOTE）を用いることで，リリースを考慮した JIT

DP にどのような影響を与えるか分析することを目的とした．また，リポジトリの規模によりオーバーサンプリングが与える影響を比較分析することも目的とした．SMOTE は近接したデータの内挿をとる形でサンプルを増やす手法である．内挿をとるときに，ランダムな gap を掛け合わせることで，サンプル間に位置するサンプルを生成することができる．SMOTE は単純なオーバーサンプリングと比較して，単純なデータの複製ではなく新しいデータを生み出すため，過学習をしにくいという利点がある．具体的に，まず，コミット数が比較的多いリポジトリ（大規模なデータセット）のコミットを用いて，SMOTE の有無による不具合予測の精度の検証を行った．次に，コミット数が比較的少ないリポジトリ（小規模なデータセット）のコミットを用いて，SMOTE の有無による不具合予測の精度の検証を行った．最後に，大規模なデータセットと小規模なデータセットで SMOTE が与える影響を比較し，考察した．その結果，以下の 3 つが明らかになった．

- SMOTE の適用前後の予測精度を比較した結果，SMOTE を適用した後のデータセットは予測精度が大きく向上した．
- 小規模なデータセットに対しても同様に予測精度に良好な影響を与えた．
- 大規模なデータセットと小規模なデータセットで SMOTE により得られた効果を比較測定した結果，大規模なデータセットの方がより良い影響を受けていた．

以降の本稿の構成を紹介する．第 2 章では，必要となる背景知識について言及する．第 3 章では，研究設問と，分析方法について述べる．第 4 章では，分析結果および考察を示す．第 5 章では，妥当性の脅威について言及し，第 6 章で結言とする．

2. 背景

2.1 不具合予測

不具合予測とは，ソフトウェアの開発情報から機械学習などで不具合予測モデルを構築し，不具合を含む可能性のあるファイルや変更を予測する手法である．不具合予測の粒度としては，ファイルレベル [2, 3] や，変更レベル [1] などが存在している．本研究では，変更レベルの不具合予測である JIT DP を取り扱う．JIT DP とは，ソフトウェアの変更に焦点を当てた不具合予測である．具体的には，コミットした変更内容に含まれる様々な情報を用いて，不具合を予測する．JIT DP はファイルレベルなどの他の不具合予測と比較すると，2 点優れた点がある．1 点目は，コミット単位で不具合を予測するため，コミット直後に不具合が存在しているか予測することができることから，開発者は変更を適用した直後に不具合を含んでいる可能性があるかどうか確認することができる点である．2 点目は，不具合を予測する粒度が細かいため，開発者が不具合を含む箇所を特定するために調べるコードの行数を遥かに抑えることができる傾向にある点である [1].

2.2 リリースを考慮したソフトウェアの不具合予測

既存の JIT DP は，十重交差検証などで過去から現在までのデータを混ぜて予測モデルを構築し，データの時系列は考慮しないことが多い．しかし，McIntosh らによると，システムは時間経るごとに複雑になるはずなので，時間を重ねるごとに専門的な知識がより重要になるなどの変化が生じるはずである [4]．McIntosh らによる検証によると，不具合を予測する上で重要な特徴が一定期間ごとに異なっているという結果が報告されている [4]．McIntosh らの研究では，不具合予測モデルに用いるデータセットを 3ヶ月あるいは 6ヶ月という単位で区切って使用しており，ある一定の期間でデータセットを区切っているという特徴がある．そのため，本研究では予測モデルを構築する上で，データの時系列を考慮することとした．ソフトウェア開発における一定期間ごとに行われる行為としてコミットやリリースなどが存在する．その中でも，リリースは時間的に十分な幅を持ち，先行研究における一定期間に相当する単位であると考えた．また，リリースは時間的特徴だけでなく，性質的

特徴においても先行研究での特徴に当てはまると考えられる．具体的には，リリースは目的によって種類分けを行うことができる点である．例えば，不具合修正を主としたリリースや機能追加が主となるリリースなどが存在している．それらの理由により，不具合予測において重要な特徴が変化すると思われる一定の期間に相当するものがリリースではないかと考えた．そこで，本研究の実験においては，テストデータをリリースごとに区切り，訓練データとしてテストデータより前のデータを用いて不具合予測を行うこととした．

2.3 リリースを考慮した不具合予測における問題点

我々は過去に，ソフトウェアリリースを考慮した変更レベルの不具合予測の検証を行った．実際にリリースを考慮して不具合予測を行った結果，予測するために必要な不具合ラベルのデータが不足しているという問題が発生した [5]．その結果，予測モデルを構築できない区間や，予測結果が良好でない区間が発生した．そのため，オーバーサンプリングなどを用いてサンプル数を増やすことで，予測モデルが構築できない区間を解消したり，予測結果を向上させることにつながるのではないかと考えた．また，小さなプロジェクトよりも大きなプロジェクトで不具合予測を行った方が精度が良くなるか検証することを課題としてあげた．そのため，本研究ではそれらを解消するために，オーバーサンプリングでサンプルを複製した前後での予測結果の違いを比較分析を行う，また，プロジェクトの大小によってオーバーサンプリングの影響に差異があるか分析を行うことにした．

2.4 不均衡なデータ

機械学習を用いた分類問題において，扱うデータセットに付けられたラベルには偏りがあるケースがある．例えば，異常検知や病理診断などにおいては大抵，異常でないラベルや病気ではないラベルのデータが大多数であり，異常であるラベルや病気であるラベルは極端に数が少ないことが多い．このようなデータセットは不均衡データと呼ばれ，機械学習で取り扱う上では注意が必要である．なぜならば，多数派のデータは予測がしやすいが，少数派のデータは予測が難しいからである．単純に多数派のデータが全体の 90 %を占めていた場合，予測器が 90 %程度の確率で

多数派のデータを多数派のデータとして予測することは容易である．しかし，こういったデータにおいては，本来予測したいデータは少数派であることが多いため，その予測結果は無意味なものであることが多い．そのため，データセットに適切な処理を施し，真陽性率を上げることが求められる．実際にデータセットの不均衡を解消することにより，分類精度が向上することが指摘されている [6, 7]．このような不均衡の解消には一般的にオーバーサンプリングやサンダーサンプリング，バギングなどが用いられる．本研究ではオーバーサンプリング手法の一種である SMOTE という手法を用いた．SMOTE の詳細は次節にて説明する．

2.5 Synthetic Minority Over-sampling Technique（SMOTE）

オーバーサンプリング手法の一つであり，Chawla ら [8] によって提案された手法である．SMOTE は不均衡データに対する処理として有用であり，非常に多くの支持を得ている [9]．また，SMOTE は低次元なデータにおいて非常に良好な結果を示したという報告もされている [10]．SMOTE の特徴としてあげられる点は，ランダムオーバーサンプリングと異なり，近傍のデータを選択して内挿を取る形でサンプルを増やす点である．具体的な手順を説明するにあたり，概略図を図 2.1 に示す [11]．まず，複製の起点となるサンプル X1 をランダムに選択する．次に，起点となるサンプル X1 に最も近いサンプル X11 から X15 を k 個選択する．（ここでは k=5）そして，その k 個の X11 から X15 の中からランダムに 1 つ選択する．選択したサンプル（ここでは X11）と起点となったサンプルの差（diff）に 0 から 1 のランダムなギャップ（gap）を掛けた位置に新たなサンプル r1 を生成する．この工程を目的のサンプル数に到達するまで繰り返す．

SMOTE はこれらの工程により，少数派クラスの分布を崩すことなく，単調なオーバーサンプリングによる過学習を抑制することができる．Imblanced learn[12] などのライブラリにより SMOTE は提供されているが，ライブラリでの実装は 2 次元のデータを前提としている．しかし，今回実験に用いたメトリクスは 13 個あるため，13 次元のデータに対応できるようにライブラリを用いず実装した．

SMOTE は近傍のデータをサンプリングに用いるため，近傍を取る必要がある．今回の実装ではユークリッド距離を採用した．また，近傍の数は 5 個とり，その中か

らランダムで1つ選択し，オーバーサンプリングを行った．

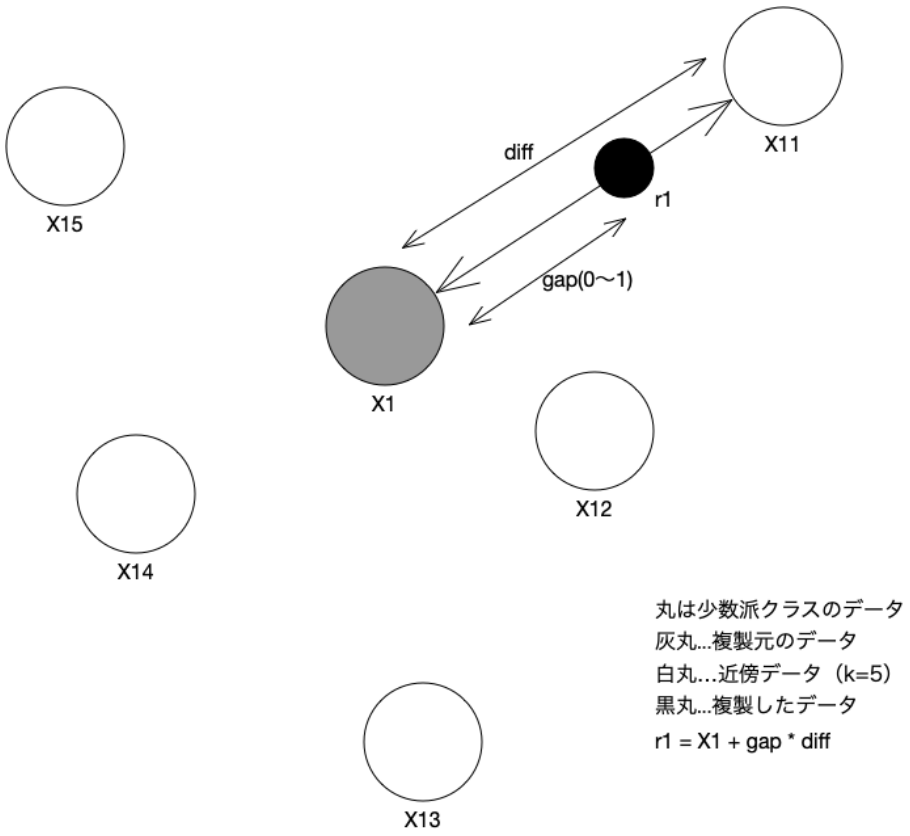

図 2.1　SMOTE の概略図

3. 実験

3.1 研究設問

本研究の目標を達成するために 3 つの研究設問を設定した.

RQ1 リリースを考慮した JIT DP での不具合予測に，SMOTE が与える影響はどの程度か.

調査方法：リリースを考慮した JITDP での不具合予測において，元のデータセットと SMOTE によりオーバーサンプリングしたデータセットそれぞれの予測精度を算出し，SMOTE による影響を調べる.

RQ2 RQ1 で予測に用いたリポジトリと比較して小規模なリポジトリを対象とした場合，SMOTE が与える影響はどの程度か.

調査方法：RQ1 と比較して小規模なリポジトリでの不具合予測を行った時に. SMOTE が予測精度に与える影響を調べる.

RQ3 リポジトリの大きさの違いによって，与えられた影響の違いはどの程度か.

調査方法：大規模なリポジトリと小規模なリポジトリから得られた不具合予測の精度を用いて効果量を測定し，SMOTE の影響にどの程度の差異があったかを比較検証する.

3.2 分析方法

本実験では，リリースを考慮した JIT DP を対象とする．リリースの考慮の方法としては，図 3.1 のようにリリース 1 以前のコミットを全て 1 とタグ付け，リリース 1 以降リリース 2 以前のコミットを全て 2 とタグ付けといったようにリリースを一つずつずらしてタグ付けをしていく．リリースを分けるにあたって，リリースおよびコミットの時間を判別するものとして，author_date_unix_timestamp という Unix タイムスタンプを用いて判別した．これは，コミットを行った時間である author date を Unix time の形式で記録されたタイムスタンプである．実験の方法を説明するにあたって，不具合予測モデルの訓練に使う区間のデータを Tr，不具合予測のテスト

に使う区間のデータを Te と呼ぶ．Tr は Te の一つ前以前のリリース全てのコミットを含み，Te は 1 つのリリース区間ののコミットだけを用いる．図 3.2 を用いて説明すると，Te がリリース 2 の時は Tr としてリリース 1 のコミットの情報を用いて学習し，リリース 2 のコミットの情報に対して不具合予測を行う．そして，次に Te がリリース 3 の時は Tr としてリリース 2 以前のコミットの情報を用いて学習し，リリース 3 のコミットの情報に対して不具合予測を行う．このようにリリースを 1 つずつずらしていき，不具合予測を行う．

3.3 分析手順

分析手順の概略図を図 3.3 に示す．まず，3.5 節で説明する Commit Guru を使ってリポジトリのコミットごとのメトリクスを取得する．次に，取得したコミットごとのメトリクスを，リリースごとに区切ったタグを割り振る．リリースごとに区切ったメトリクスに対して SMOTE を適用し，SMOTE を適用したデータセットと適用していないデータセットを作成する．そして，それぞれのデータセットに対してロジスティック回帰モデルを用いて，不具合予測を行う．その際，Te をテストデータとし，Tr を学習データとして，1 つずつリリースをずらして不具合予測を行う．不具合予測の結果から AUC，Precision，Recall，F-measure を算出し，結果の考察を行う．

3.4 libraries.io

本研究において実験に用いるリポジトリの選定に用いたサービスである [13]. libraries.io は，5,123,023 個の OSS パッケージのリリースを監視し，プロジェクトの一覧やバージョンの一覧などを csv の形式で公開している．また，パッケージの依存関係などもマッピングしている．本実験においては，コントリビュータ数をプロジェクトの選定に用いた．理由としては，一定数以上のコントリビュータ数をもつリポジトリは十分な数のコミットとリリースが担保されると考えたためである．

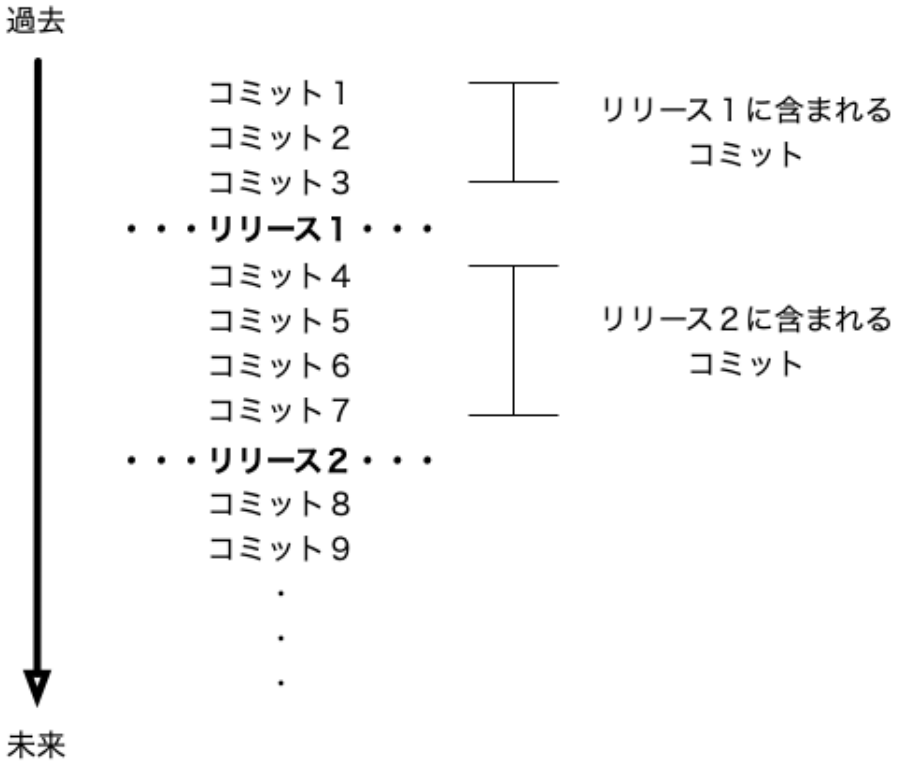

図 3.1　リリースとコミットの関係

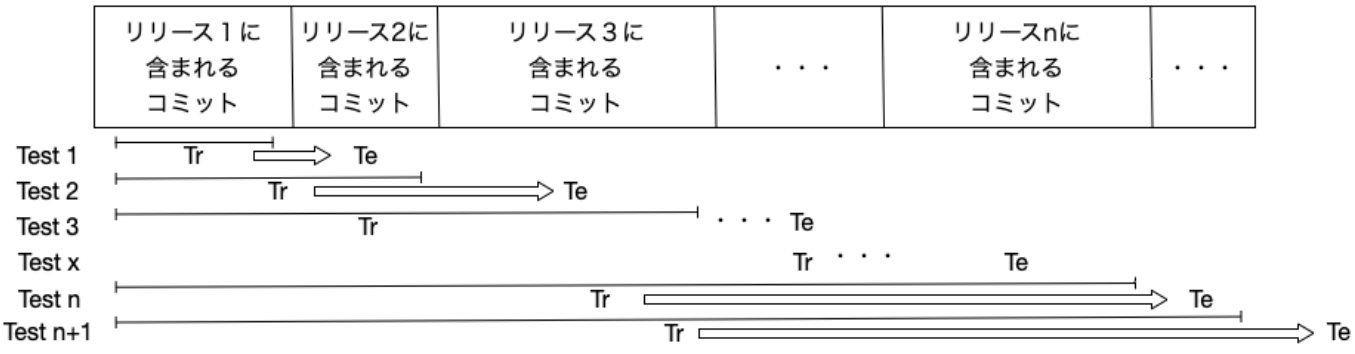

図 3.2　訓練データとテストデータの関係

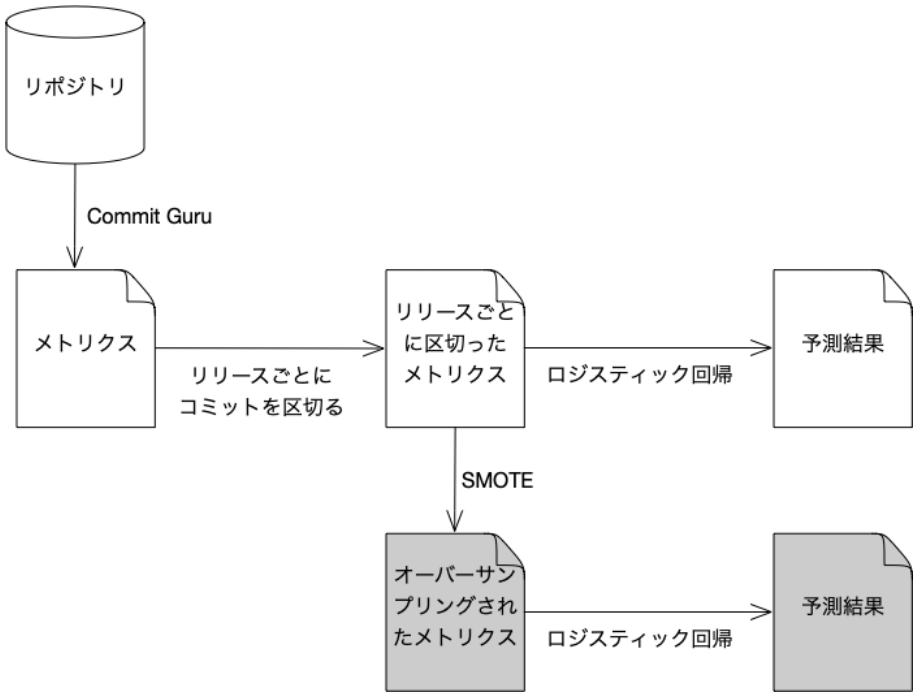
リポジトリ
Commit Guru
メトリクス
リリースごとに
コミットを区切る
リリースごと
に区切った
メトリクス
ロジスティック回帰
予測結果
SMOTE
オーバーサン
プリングされ
たメトリクス
ロジスティック回帰
予測結果

図 3.3　分析手順

3.5 Commit Guru

本研究において実験に用いるメトリクスをリポジトリから抽出するツールである．Commit Guru は，Rosen らによって公開されている [15, 16]．指定した Git のリポジトリから，コミットのコミットされた時間や不具合の有無，不具合を修正したコミットのコミットハッシュなどのメトリクスを計算する．実験に用いるメトリクスは表 3.1 に示す.

3.6 scikit-learn

本研究での機械学習には scikit-learn を用いている．scikit-learn はオープンソースの機械学習ライブラリである [17]．scikit-learn はユーザーコミュニティにより現在も開発が進められている．また，ドキュメントも多く存在していることから，初学者も気軽に用いることができるという利点がある．scikit-learn は分類，回帰，クラスタリングなどのさまざまな機械学習モデルを実装している．本研究ではロジスティック回帰モデルを用いており，sklearn.linear_model.LogisticRegression というクラスで実装されている．ロジスティック回帰モデルは入力されたメトリクスから確率を計算し，閾値をもって 0 か 1 の 2 値に分類する．

3.7 リポジトリの選定方法

libraries.io から取得したコントリビュータ数を用いて上位 100 件のリポジトリを選択した．3 個のリポジトリは削除されていたため，取得に成功したリポジトリは 97 個だった．そのうち，リポジトリ内に.git ファイルのみのリポジトリが存在していたため，以降の選定には 58 個のリポジトリを用いた．ダウンロードしたリポジトリからコミット数を取得し，第一四分位数・第三四分位数を計算した．第三四分位数以上のコミット数を持つ 15 個のリポジトリを大規模なリポジトリ，第一四分位数以下のコミット数を持つ 15 個のリポジトリを小規模なリポジトリとして，実験に用いるリポジトリとして選択した．実験に用いるリポジトリのメトリクスは Commit Guru を用いて取得した．Commit Guru で取得する際，大規模なリポジトリのうち https://github.com/Homebrew/homebrew-cask のリポジトリは異常終了が発生して取

表 3.1　不具合予測に用いたメトリクス一覧

メトリクス名	説明
ns	修正されたサブシステムの数
nd	修正されたディレクトリの数
nf	修正されたファイルの数
entropy	修正されたコードの各ファイル毎の分布
la	変更によって追加されたコードの行数
ld	変更によって削除されたコードの行数
lt	変更される前のファイルのコード行数
ndev	修正に関わった開発者の人数
age	最後の変更から最新の変更までの平均時間
nuc	修正されたファイルに対する変更の数
exp	開発者の経験，開発者の総コミット数
rexp	年齢によって重み付けされた exp
sexp	開発者のサブシステムに対する総コミット数
contains_bug	不具合を含むか（True/False）

得できなかったため，実験対象から除外した．また，選択したリポジトリのうち，tag が存在していないリポジトリおよび，リリースが 1 つだけのものは実験の設定上不適切であると判断したため，除外した．最終的に大規模なリポジトリは 13 個，小規模なリポジトリは 10 個となった．

3.8 データセットのタグ付け

リリースを考慮した不具合予測を行うにあたって，コミットをリリースごとに分割している．分割に用いるタグは，セマンティックバージョニングを参考に選択した．また，プレリリースバージョンにあたる RC や alpha などは除外した．それにあたり，'\d+\.\d+\.\d+' や'v\d+\.\d+\.\d+' といった正規表現に当てはまるものを選択した．

3.9 評価指標

本実験の不具合予測の評価にあたり，AUC（Area Under the Curve），再現率（Recall），適合率（Precision），F-measure，Cohen's d の 5 つの評価値を用いた．各評価値の説明および計算式は以下に示す．

3.9.1 AUC（Area Under the Curve）

AUC とは，ROC 曲線を作成したときに ROC 曲線以下の面積を指す．ROC 曲線とは，推定確率を閾値として変化させたときに，真陽性率と偽陽性率に対応する点を結んだものである．真陽性率を縦軸，偽陽性率を横軸に設定して曲線を描く．偽陽性率が大きくなるように閾値を設定すると真陽性率も共に大きくなるため，常に ROC 曲線は右肩上がりのグラフとなる．偽陽性率が低い段階から真陽性率が高い予測モデルは，AUC の面性が広くなり，良い不具合予測モデルであるとされている．

3.9.2 再現率（Recall）

再現率（Recall）とは，実際に正であったもののうち，正であると正しく予測できたものの割合を指す．本実験では，実際に不具合を含むコミットのうち，不具合を

含むコミットであると予測できたものの割合をさす．表 3.2 の凡例にのっとると，式 3.1 のように定義される．

$$Recall = \frac{TP}{TP + FN} \tag{3.1}$$

不具合予測において Recall の値が高くなることは，存在している不具合を取りこぼさずに検出できた割合を指すため，重要な指標であるといえる．しかし，全ての正解データに対して正であると予測した場合 Recall は 1 となるように，純粋に高ければ良いわけではなく，後述の Precision とバランスを取ることが重要である．

3.9.3 適合率 (Precision)

適合率（Precision）とは，正と予測したものが実際に真値が正であったものの割合を指す．本実験では，不具合を含むコミットであると予測したものが，実際に不具合を含むコミットであった割合を指す．表 3.2 の凡例にのっとると，式 3.2 のように定義される．

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

不具合を含まないコミットを不具合であるコミットと予測するケースが増えると，Precision の値は低くなる．Precision の値が低い予測結果を用いると，不具合を発見するために必要なコストが多くなってしまう．また，Precision は前述の Recall とトレードオフな関係にある．

3.9.4 F-measure

F-measure とは，Precision と Recall の調和平均である．Precision と Recall はトレードオフの関係にあるため，両指標を考慮した評価をするために F-measure を用いる．F-measure が高い値を出しているほど，Precision と Recall がバランスよく高いことになり，予測精度が高いといえる．Precision と Recall の値がバランスを欠き，どちらかの値が著しく低い場合，F-measure も同様に低い値を示すようになる．F-measure は Precision と Recall の値を用いて式 3.3 のように定義される．

$$F - measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{3.3}$$

F-measure には Precision もしくは Recall を重視した算出方法があるが，本研究では両者の値を重視したため，どちらの算出方法も用いなかった．

3.9.5 Cohen's d

Cohen's d は効果量の差の大きさを表す指標の一つである．具体的には，2 つのグループの平均の差がどの程度あるかを表す．グループ全体の標準偏差 s，グループごとのサンプル数 n_1，n_2，グループごとの標準偏差 s_1，s_2，グループごとの平均 $\bar{x}_1$，$\bar{x}_2$ を用いて式 3.4，3.5 のように定義される．

$$d = \frac{|\bar{x}_1 - \bar{x}_2|}{s} \tag{3.4}$$

$$s = \sqrt{\frac{n_1 s_1^2 + n_2 s_2^2}{n_1 + n_2}} \tag{3.5}$$

Cohen's d は一般的に 0.2 で小程度，0.5 で中程度，0.8 で大程度の差があるとされている．

表 3.2　評価の凡例

		実測	
		不具合を含む	不具合を含まない
予測	不具合を含む	True Positive（TP）	False Positive（FP）
	不具合を含まない	False Negative（FN）	True Negative（TN）

4. 分析結果

4.1 結果

4.1.1 RQ1）リリースを考慮した JIT DP での不具合予測に，SMOTE が与える影響はどの程度か

大規模なデータセットに対して SMOTE を適用した場合と適用しなかった場合における予測精度の結果を図 4.1 から図 4.26 に示す．いずれのデータにおいても，SMOTE 適用前の予測結果よりも，SMOTE 適用後のデータの方が優れた予測精度を出している．具体的に表 4.1 を見てみると，特に Recall や F-measure の値は中央値が大きく増加しており，全体的に予測精度が向上したと考えられる．AUC と Precision においても，中央値が 0.8 を超えており，十分に予測精度が向上したと考えられる．また，全体として四分位範囲が狭くなっていることから，平均的に精度が良くなっているのではないかと考えられる．

RQ1 への回答は，SMOTE を適用することによって予測精度に良い影響を与えることがわかった．また，最も改善される値は Recall であった．

4.1.2 RQ2）RQ1 で予測に用いたリポジトリと比較して小規模なリポジトリを対象にした場合，SMOTE が与える影響はどの程度か．

小規模なデータセットに対して SMOTE を適用した場合と適用しなかった場合おける予測精度の結果を図 4.27 から図 4.46 に示す．いずれのデータにおいても，SMOTE 適用前の予測結果よりも，SMOTE 適用後のデータの方が優れた予測精度を出している．具体的に表 4.2 を見てみると，全体的に中央値が増加しており，全体的に予測精度が向上したと考えられる．特に Recall の値は大きく増加していた．また，Precision，Recall，F-measure の四分位範囲が狭くなっていることから，平均的に精度が良くなっているのではないかと考えられる．ただし，AUC の四分位範囲は適用前と比較して大きくなっている．これは，第三四分位数付近の増加と比較して第一四分位数付近の増加が若干少ないため，四分位範囲に開きが出てしまっている．そのため，予測精度のばらつきが少し大きくなってしまったといえる．

RQ2 への回答は，大規模なリポジトリと比較すると，小規模なリポジトリに対す

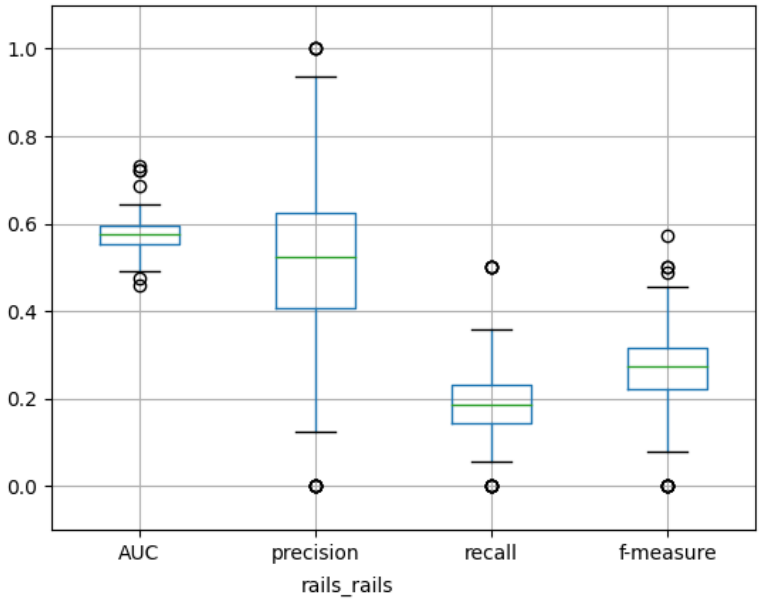

図 4.1　rails/rails のデータの予測結果

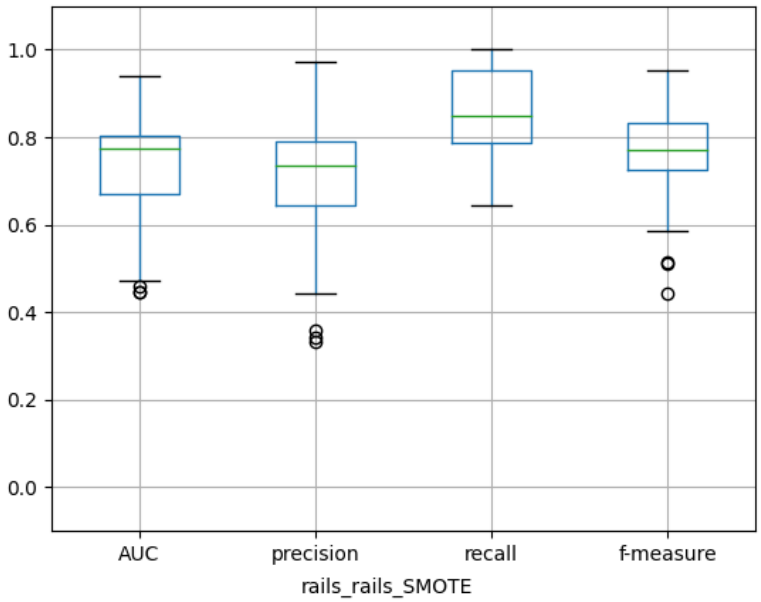

図 4.2　rails/rails に SMOTE を適用したデータの予測結果

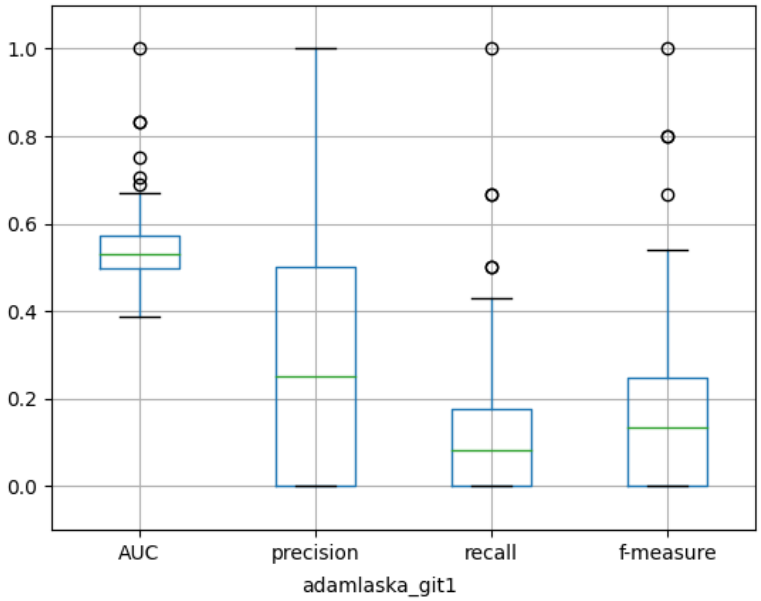

図 4.3　adamlaska/git1 のデータの予測結果

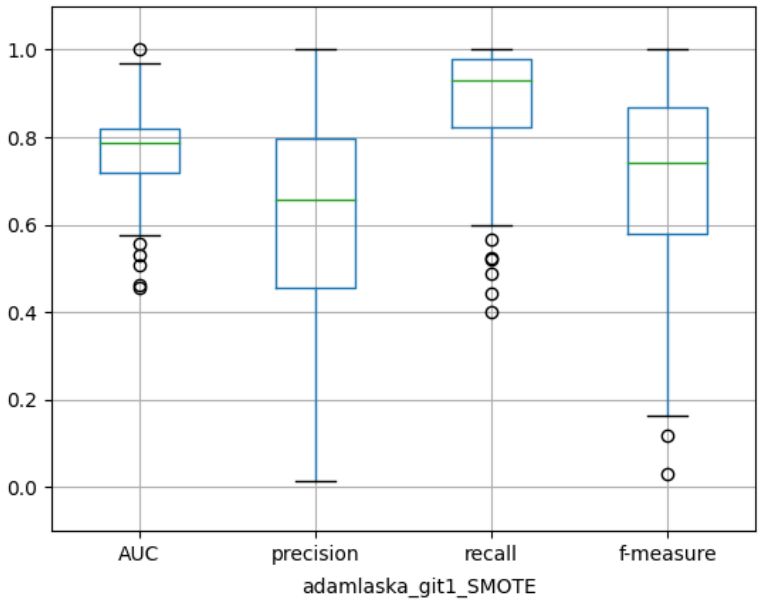

図 4.4　adamlaska/git1 に SMOTE を適用したデータの予測結果

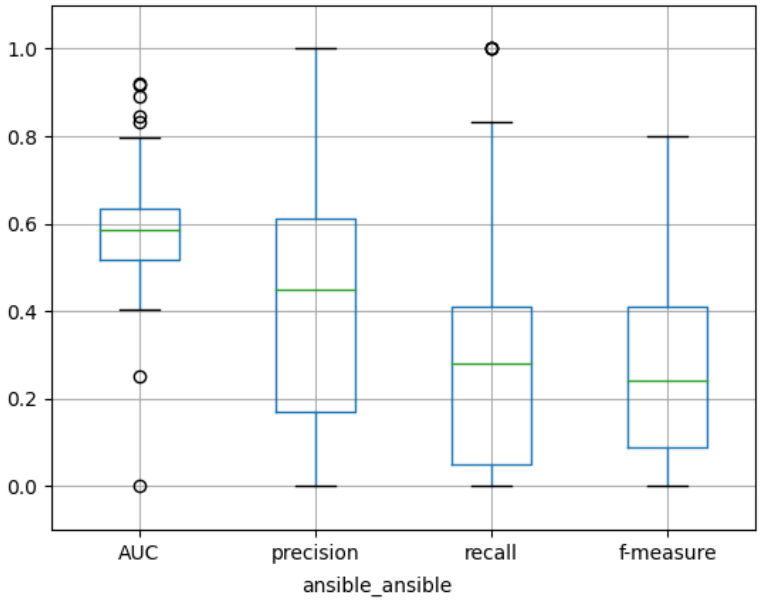

図 4.5　ansible/ansible のデータの予測結果

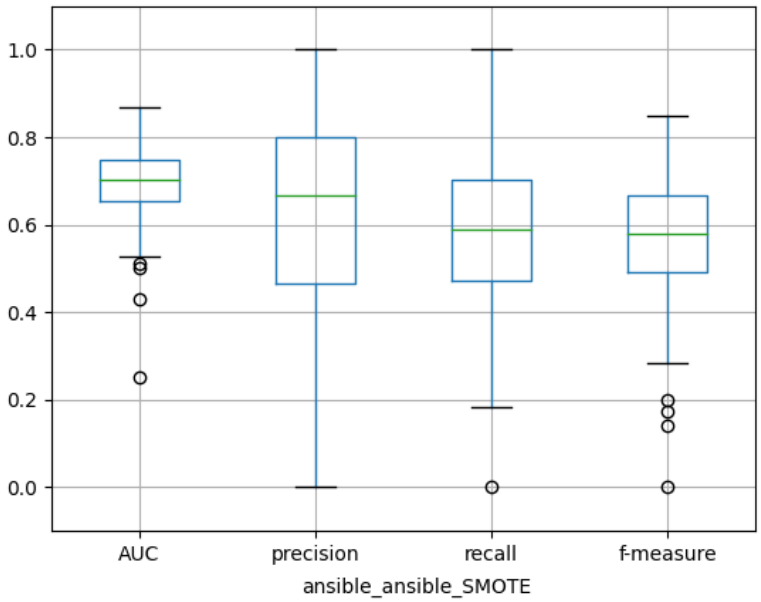

図 4.6　ansible/ansible に SMOTE を適用したデータの予測結果

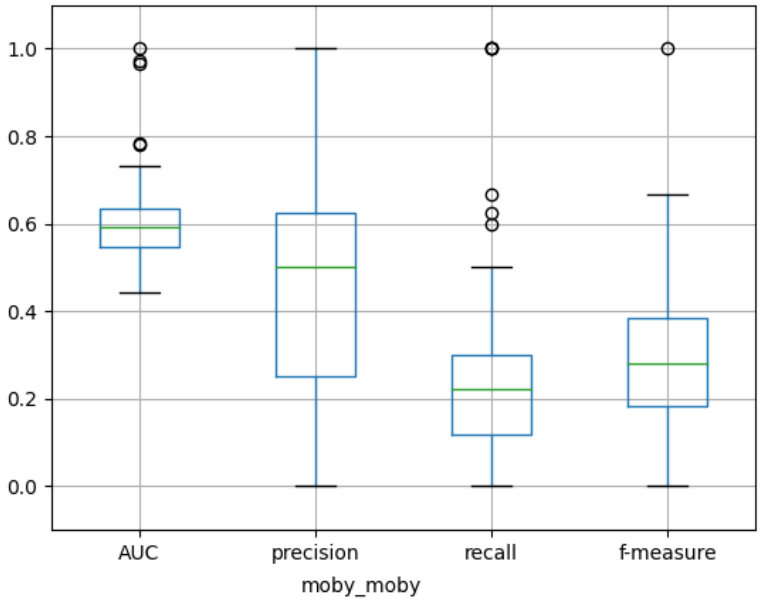

図 4.7　moby/moby のデータの予測結果

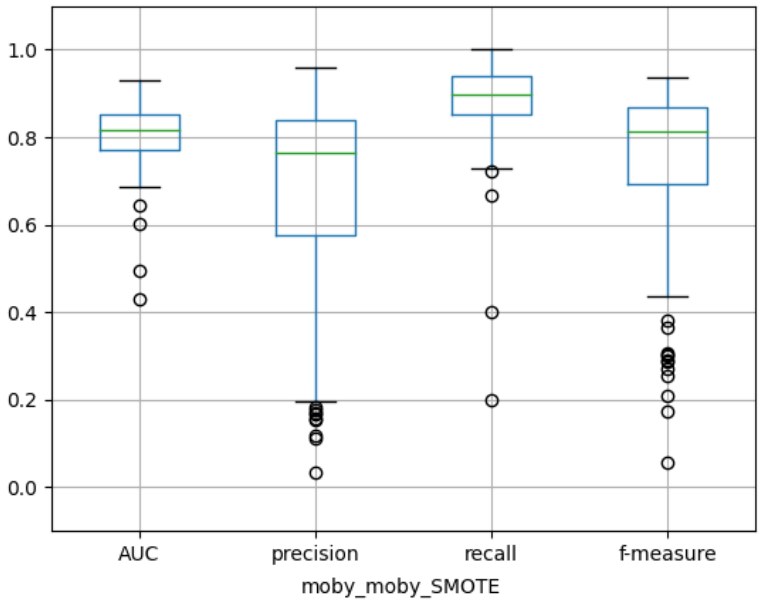

図 4.8　moby/moby に SMOTE を適用したデータの予測結果

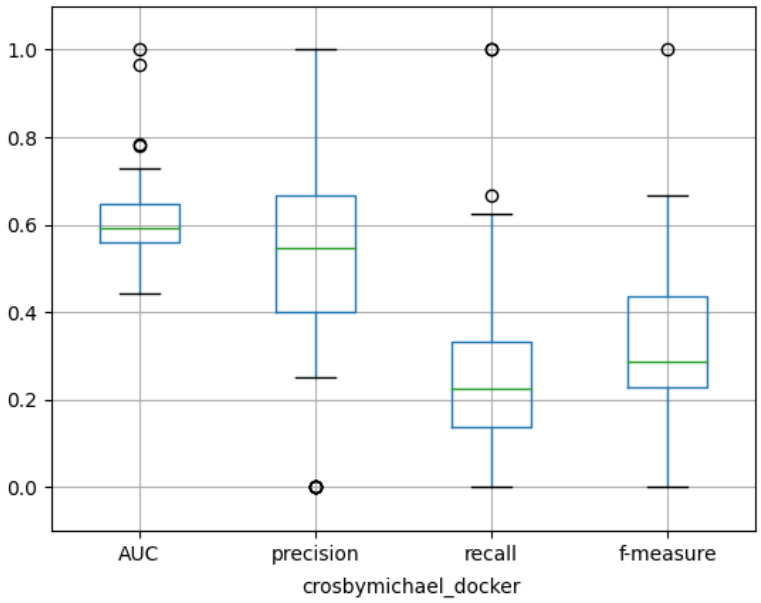

図 4.9　crosbymichael/docker のデータの予測結果

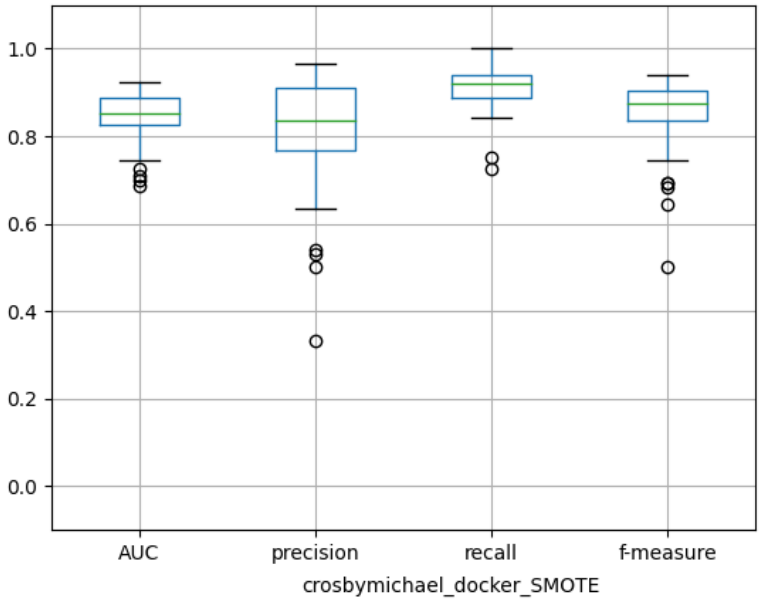

図 4.10　crosbymichael/docker に SMOTE を適用したデータの予測結果

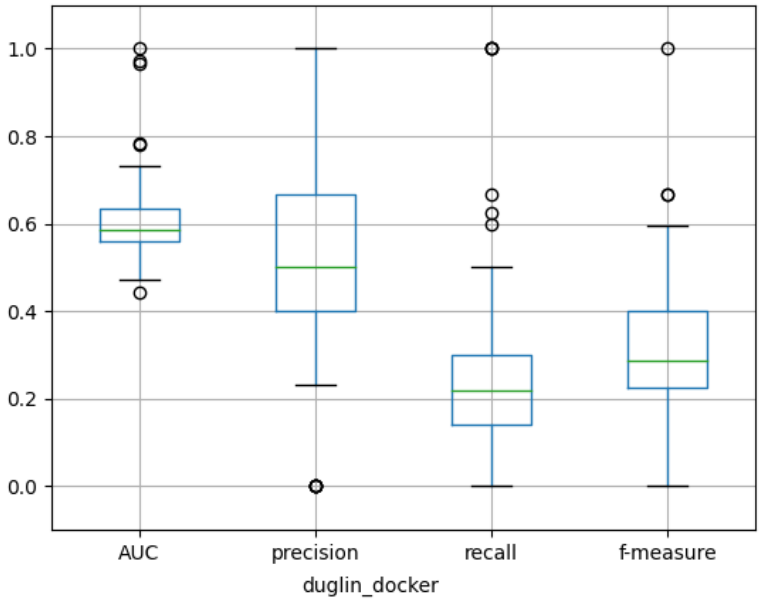

図 4.11　duglin/docker のデータの予測結果

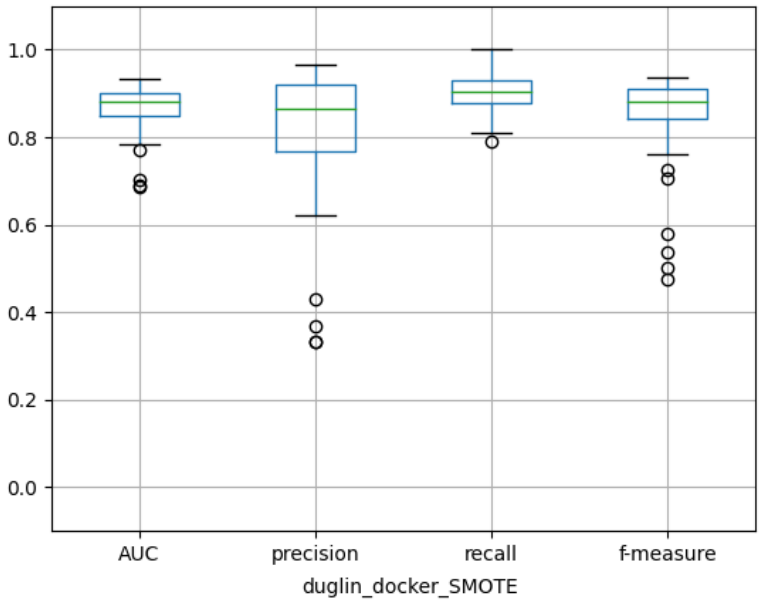

図 4.12　duglin/docker に SMOTE を適用したデータの予測結果

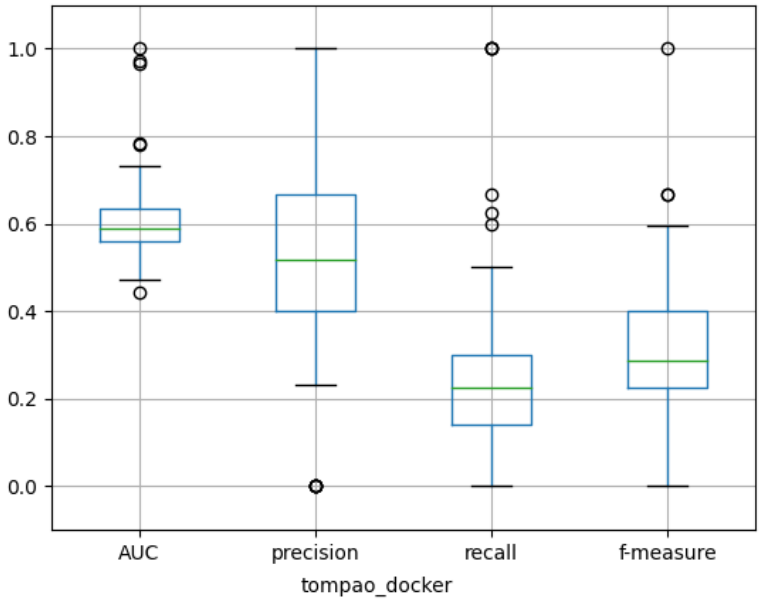

図 4.13　tompao/docker のデータの予測結果

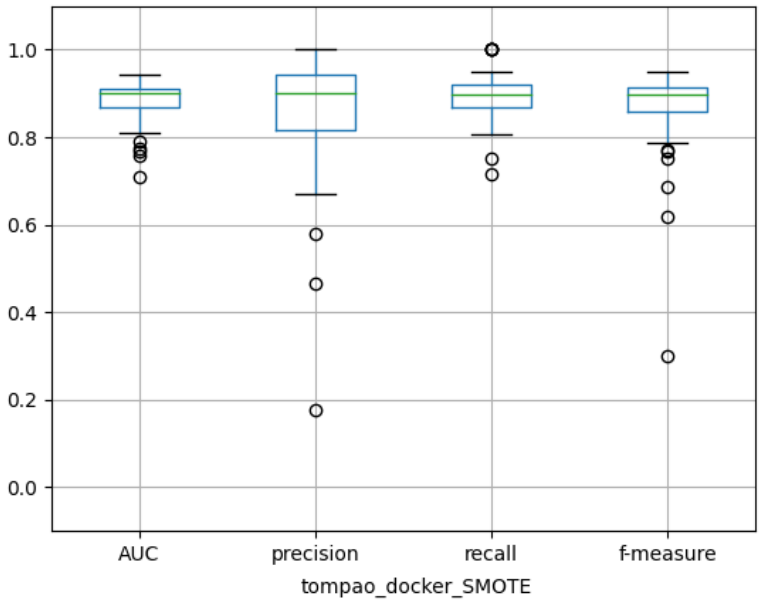

図 4.14　tompao/docker に SMOTE を適用したデータの予測結果

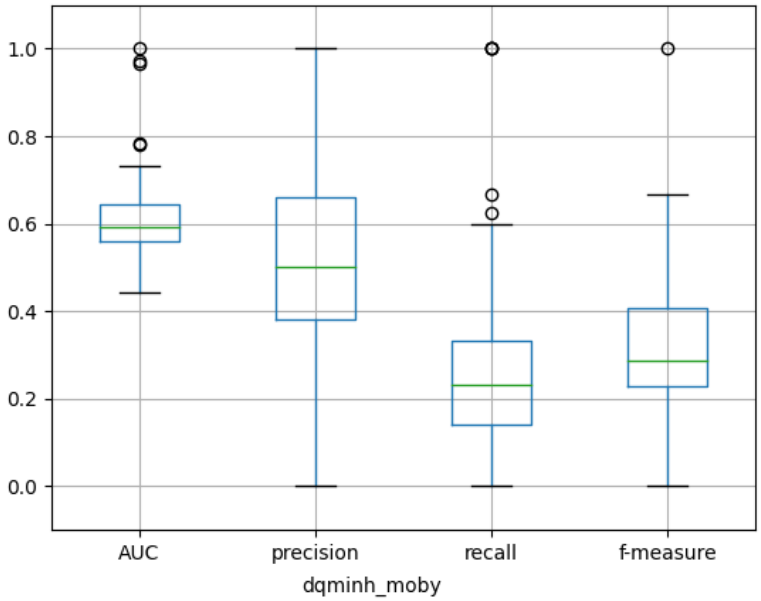

図 4.15　dqminh/moby のデータの予測結果

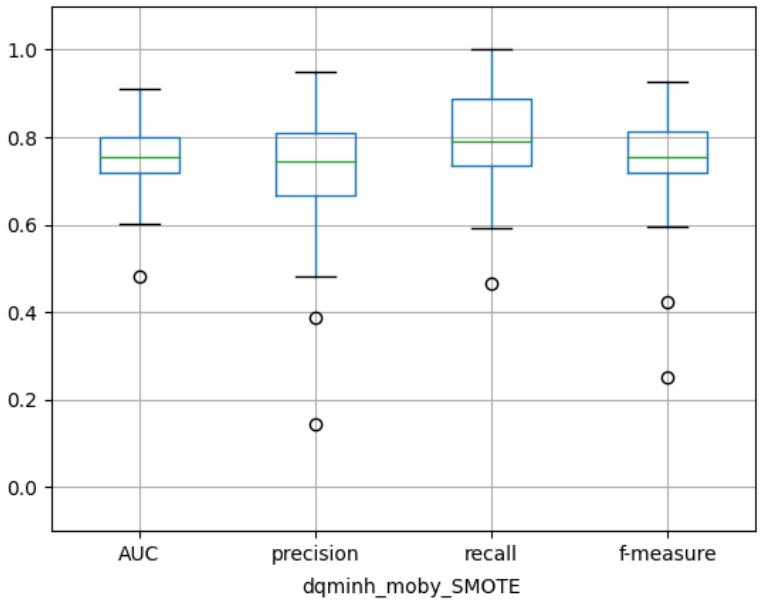

図 4.16　dqminh/moby に SMOTE を適用したデータの予測結果

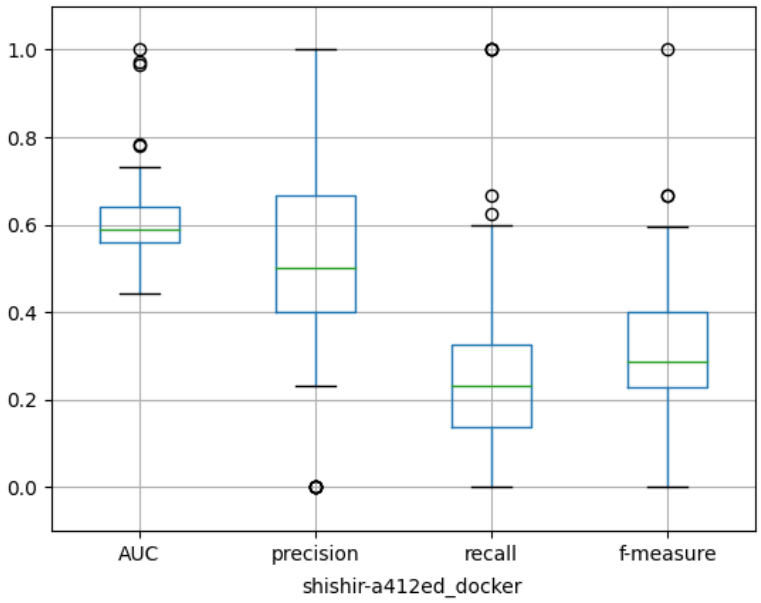

図 4.17　shishir-a412ed/docker のデータの予測結果

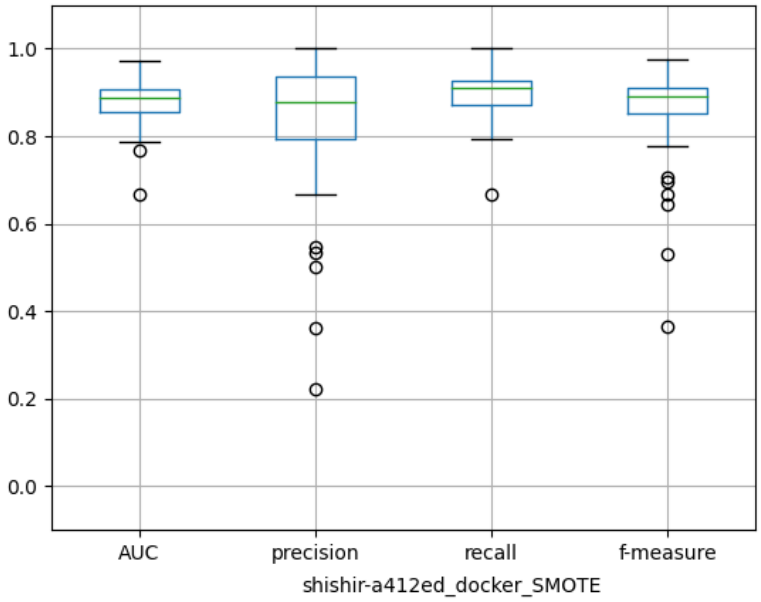

図 4.18　shishir-a412ed/docker に SMOTE を適用したデータの予測結果

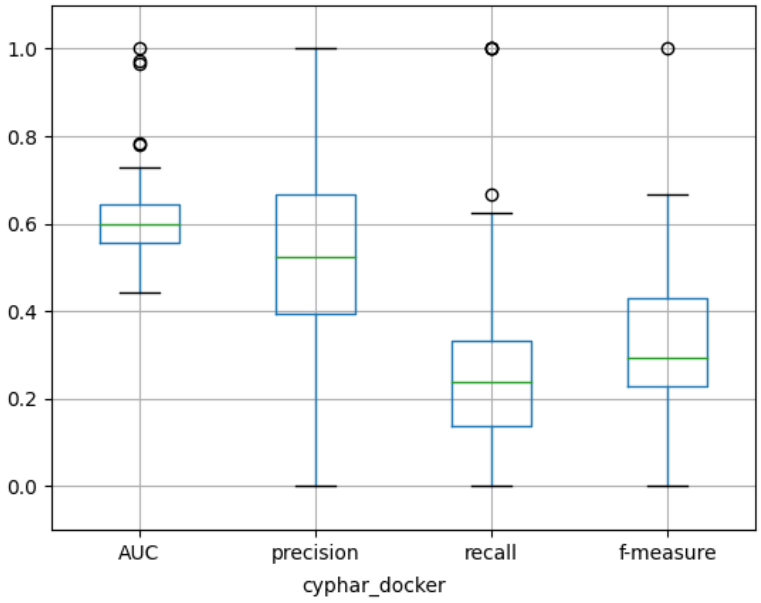

図 4.19　cyphar/docker のデータの予測結果

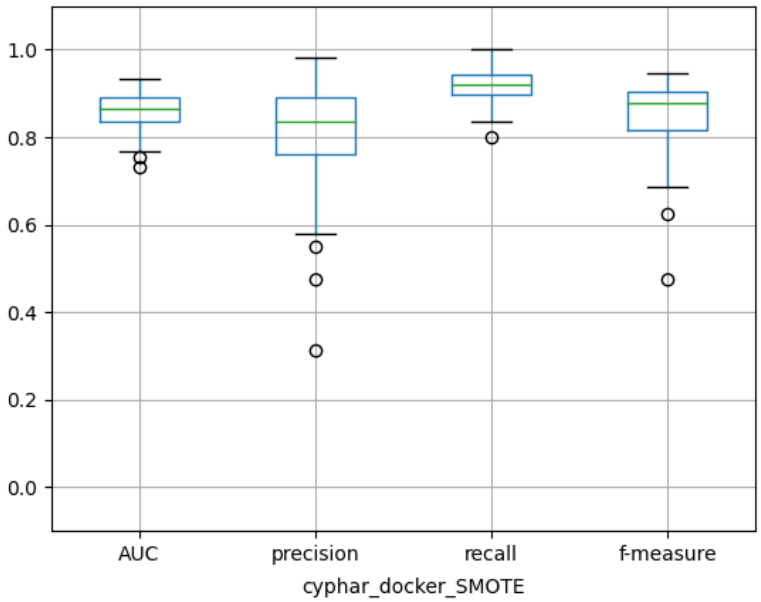

図 4.20　cyphar/docker に SMOTE を適用したデータの予測結果

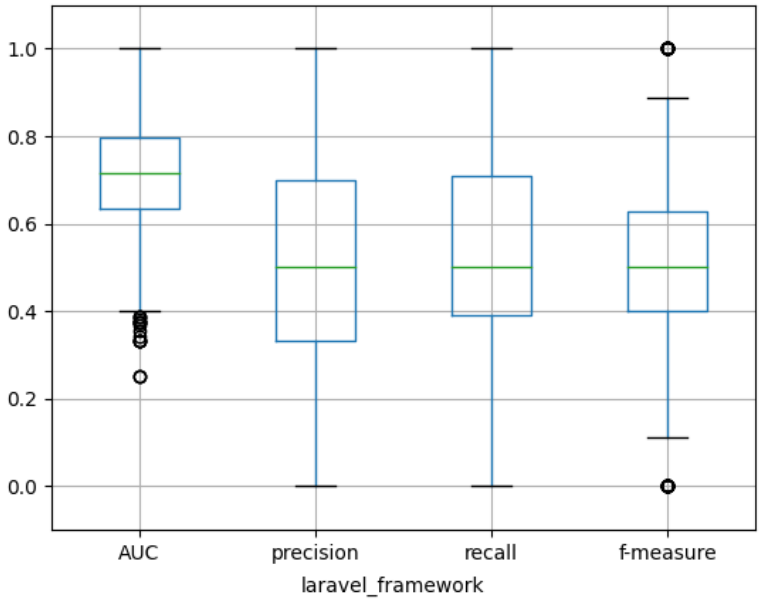

図 4.21　laravel/framework のデータの予測結果

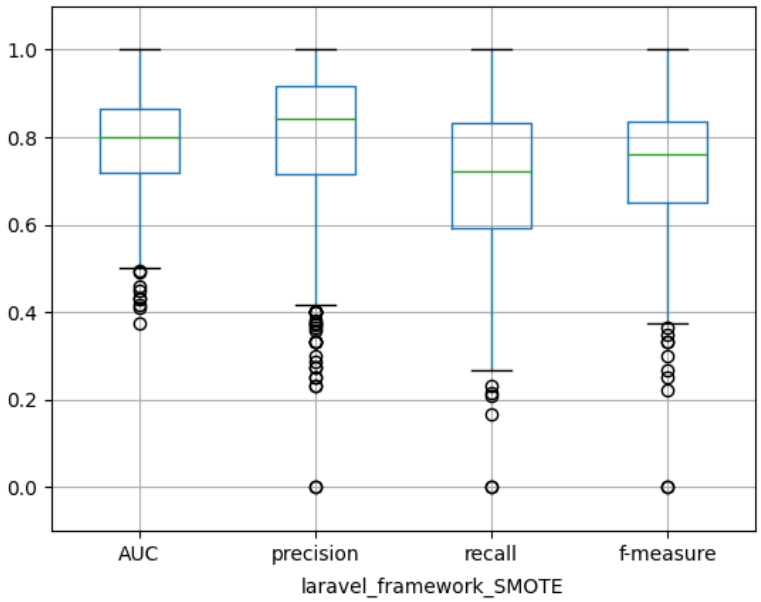

図 4.22　laravel/framework に SMOTE を適用したデータの予測結果

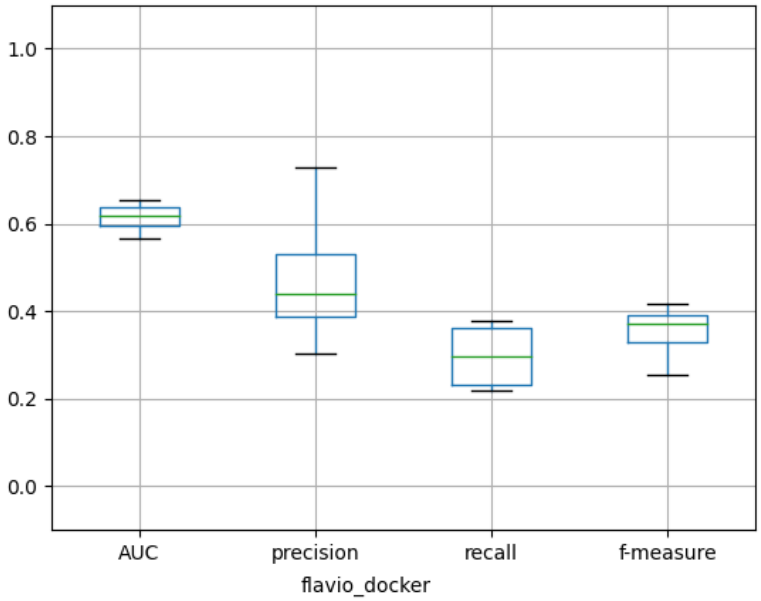

図 4.23　flavio/docker のデータの予測結果

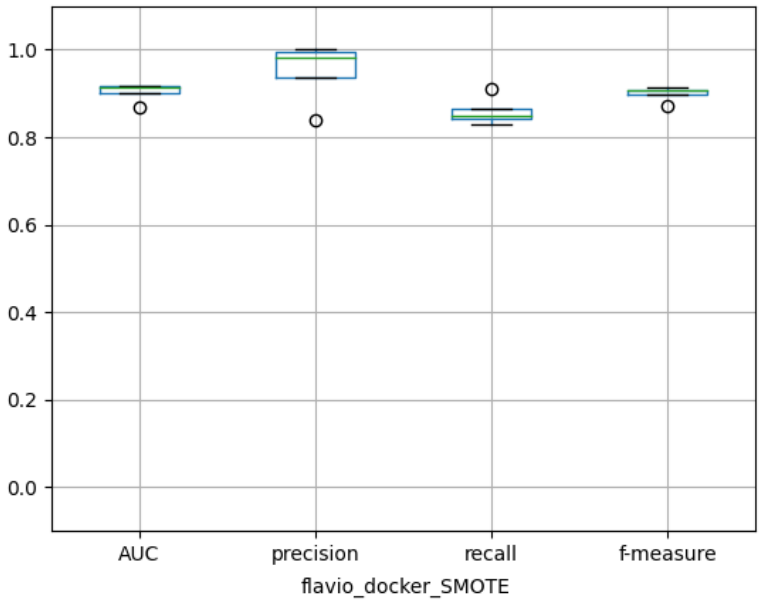

図 4.24　flavio/docker に SMOTE を適用したデータの予測結果

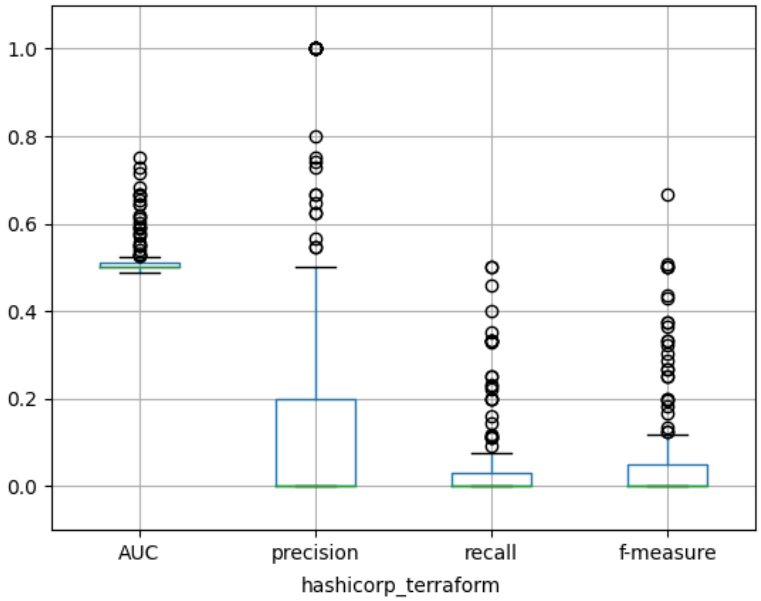

図 4.25　hashicorp/terraform のデータの予測結果

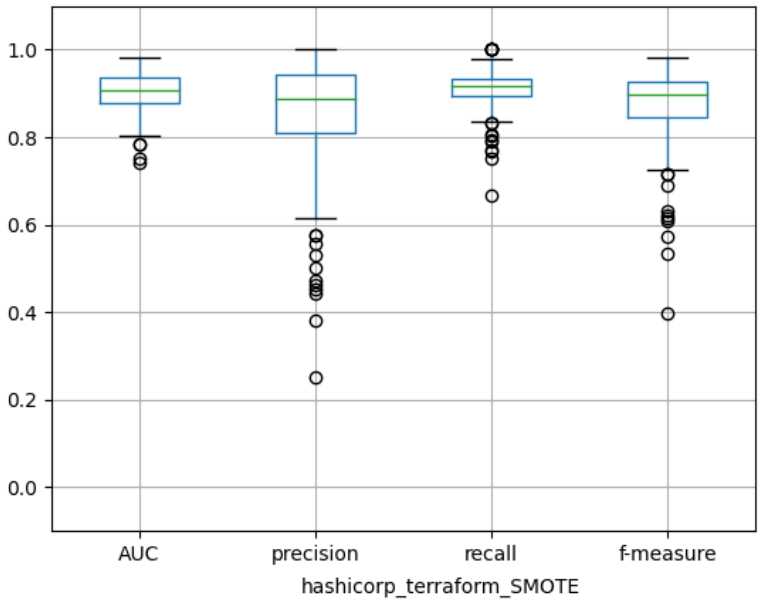

図 4.26　hashicorp/terraform に SMOTE を適用したデータの予測結果

表 4.1　大規模なデータセットでの SMOTE 適用前後の予測結果の値

		SMOTE 適用前	SMOTE 適用後
AUC	第 1 四分位数	0.5535	0.7897
	中央値	0.5895	0.8338
	第 3 四分位数	0.6322	0.8642
	四分位範囲	0.0787	0.0746
Precision	第 1 四分位数	0.3019	0.7062
	中央値	0.4423	0.8152
	第 3 四分位数	0.5988	0.8841
	四分位範囲	0.2969	0.1779
Recall	第 1 四分位数	0.1359	0.8001
	中央値	0.2256	0.8533
	第 3 四分位数	0.3192	0.9034
	四分位範囲	0.1833	0.1033
F1 値	第 1 四分位数	0.1987	0.7538
	中央値	0.2711	0.8190
	第 3 四分位数	0.3772	0.8666
	四分位範囲	0.1784	0.1127

るSMOTEの影響は低くなる．ただし，SMOTE適用後の予測結果は適用前と比較して改善が見られる．また，大規模なリポジトリと同様にRecallが最も改善されている．

4.1.3 RQ3）リポジトリの大きさの違いによって，与えられた影響の違いはどの程度か．

SMOTEを適用した結果，大規模なデータセットと小規模なデータセットそれぞれにどの程度の影響を与えたか分析するために効果量の測定を行った．今回はCohen's dを用いて，効果量を測定することにした．Cohen's dは，2つの標本間の平均値の差を比較する際に用いられる統計的な指標の一つである．平均値にどれだけの差があるかを示し，2標本の平均の差を標準偏差で割って求める．結果は表4.3に示す．効果量を測定したところ，4つ全てのデータにおいてSMOTEは小規模なデータセットより大規模なデータセットに対してより良い影響を与えることがわかった．ただし，Cohen's dは一般に0.8程度あれば大きな効果があったと見なされることが多く，小規模なデータセットにおいても十分な効果があったと考えられる[18]．

RQ3への回答は，効果測定の結果，大規模なデータセットは小規模なデータセットと比較してより大きなの効果を得られたことがわかった．特にRecallの差が最も大きく，Precisionの差は最も小さかった．

4.2 考察

4.2.1 SMOTEの影響

RQ1とRQ2の結果より，SMOTEは不具合予測の予測精度に良い影響を与えることがわかった．4つの測定値の中ではRecallが特に良くなることもわかった．Recallは実際に正例であるものを正であると予測できた割合を示しているものである．この値が著しく良くなったということは，不具合を取りこぼしてしまう可能性を大きく減らせたと捉えることができるため，SMOTEは不具合の予測に良好な影響を与えると考えられる．また，Recallの増加に伴ってPrecisionが減少することもなかった．RecallとPrecisionはトレードオフな関係になり，どちらかの増加に伴ってどちらかの値が減少することがあるが，どちらの値も減少させることなく共に増加した

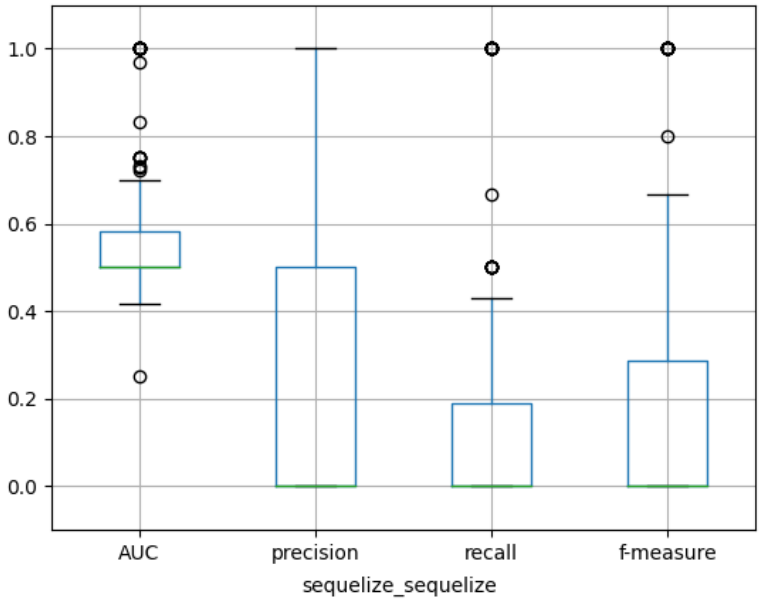

図 4.27　sequelize/sequelize のデータの予測結果

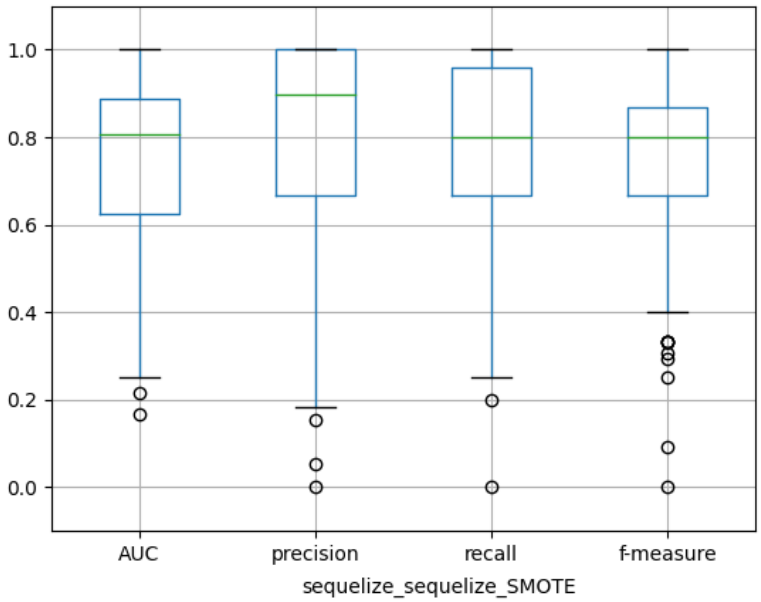

図 4.28　sequelize/sequelize に SMOTE を適用したデータの予測結果

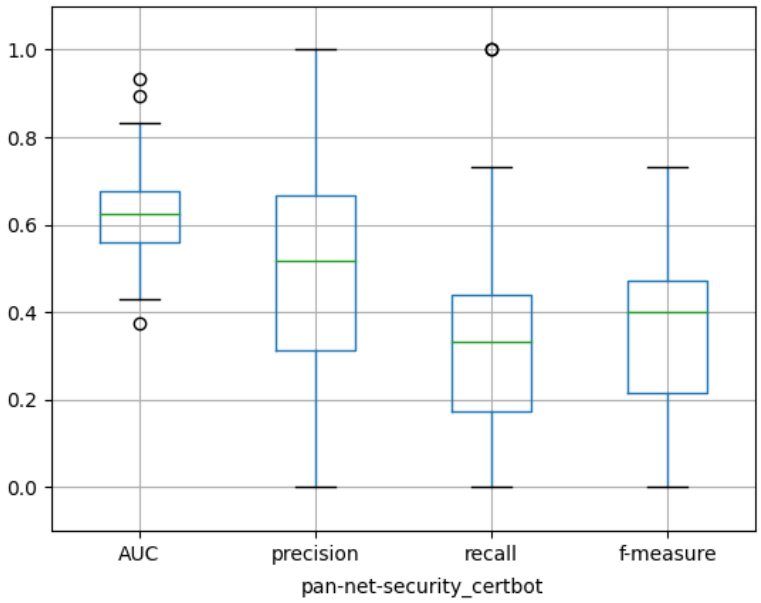

図 4.29　pan-net-security/certbot のデータの予測結果

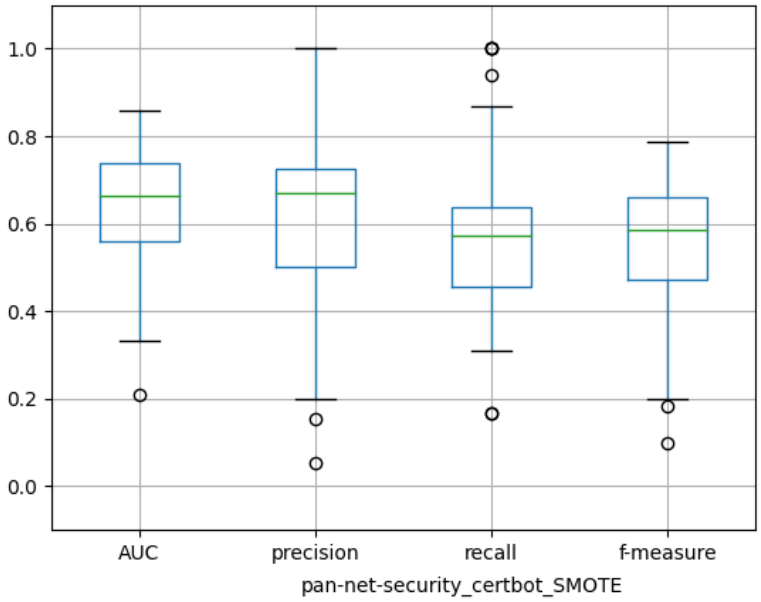

図 4.30　pan-net-security/certbot に SMOTE を適用したデータの予測結果

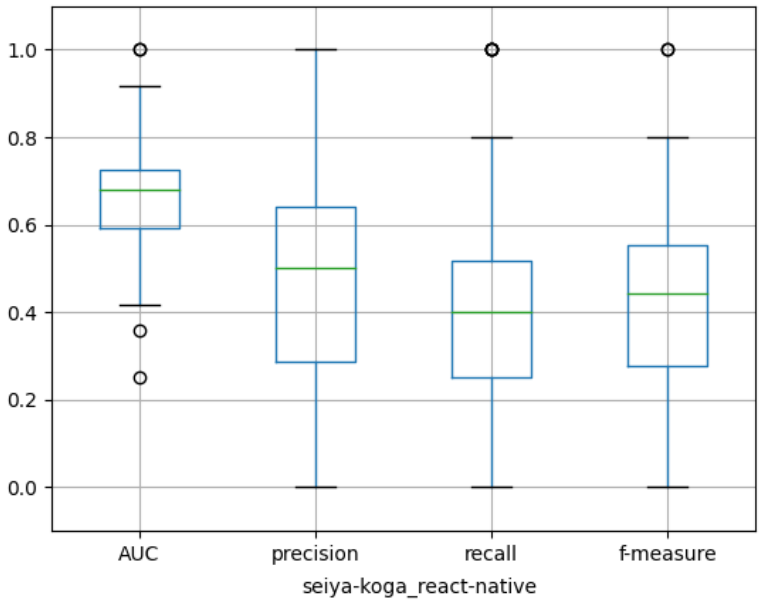

図 4.31　seiya-koga/react-native のデータの予測結果

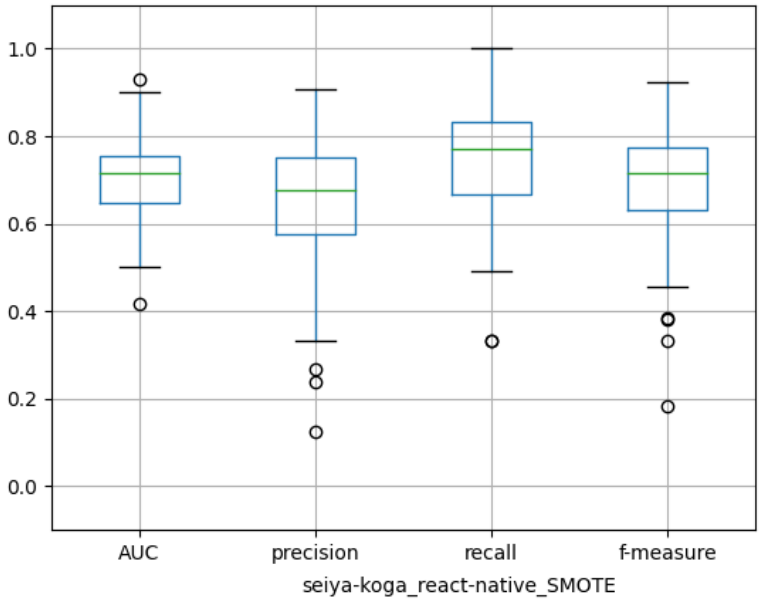

図 4.32　seiya-koga/react-native に SMOTE を適用したデータの予測結果

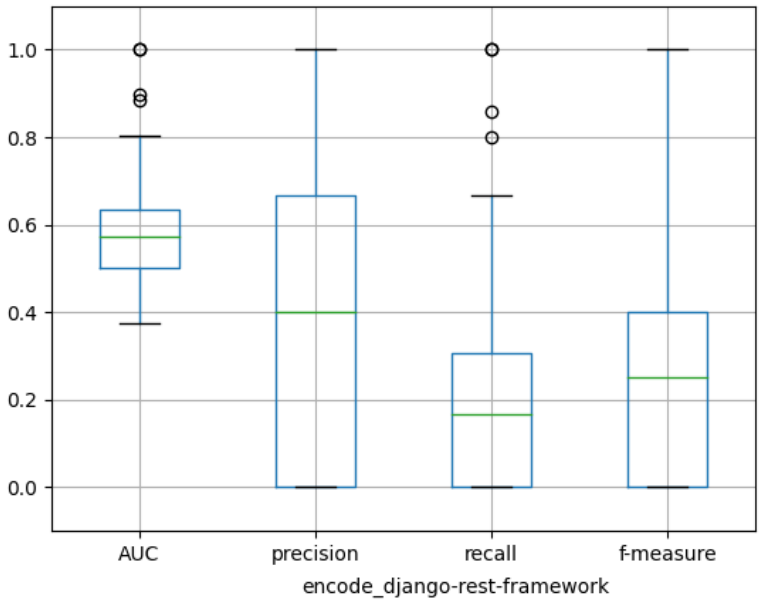

図 4.33　encode/django-rest-framework のデータの予測結果

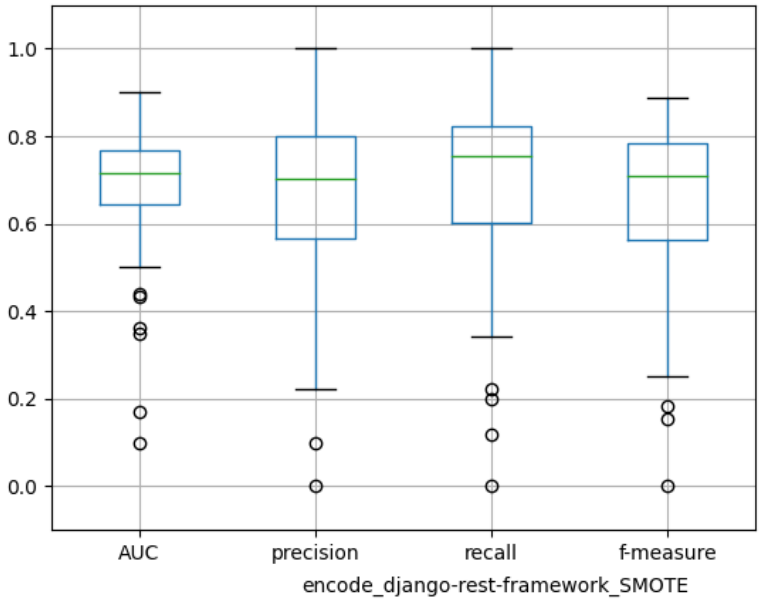

図 4.34　encode/django-rest-framework に SMOTE を適用したデータの予測結果

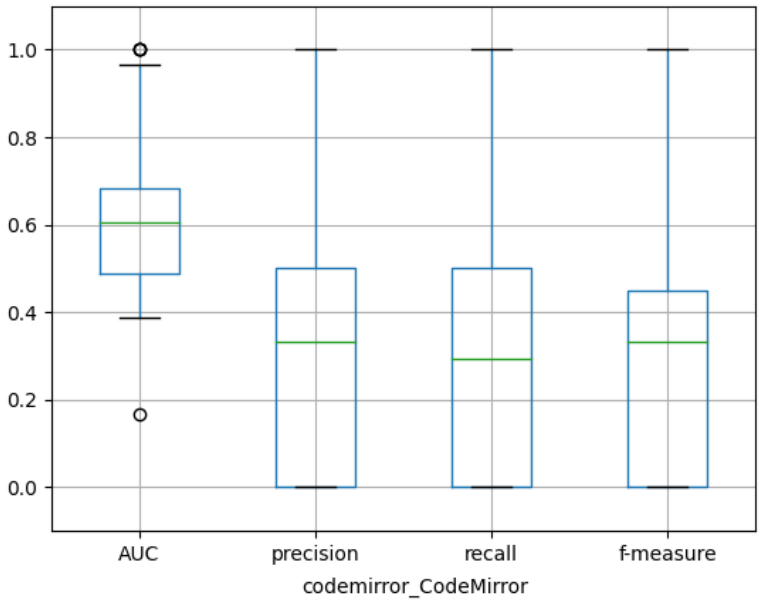

図 4.35　codemirror/CodeMirror のデータの予測結果

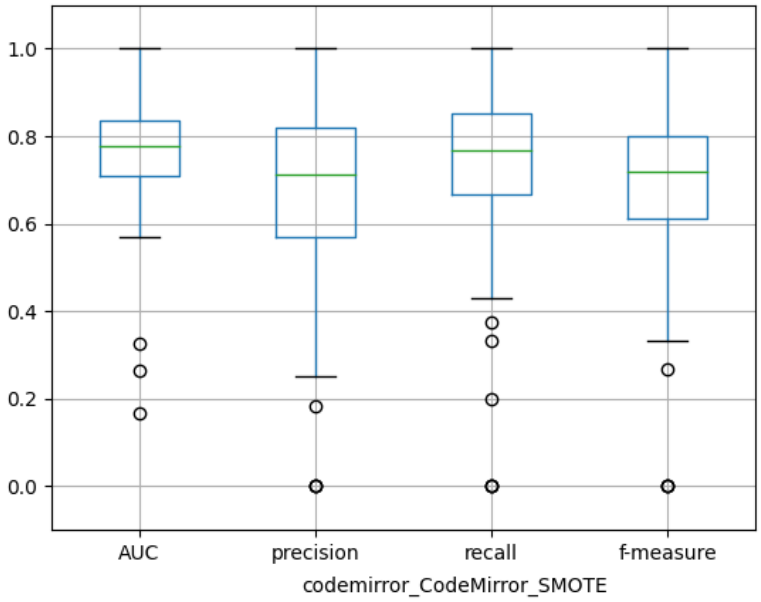

図 4.36　codemirror/CodeMirror に SMOTE を適用したデータの予測結果

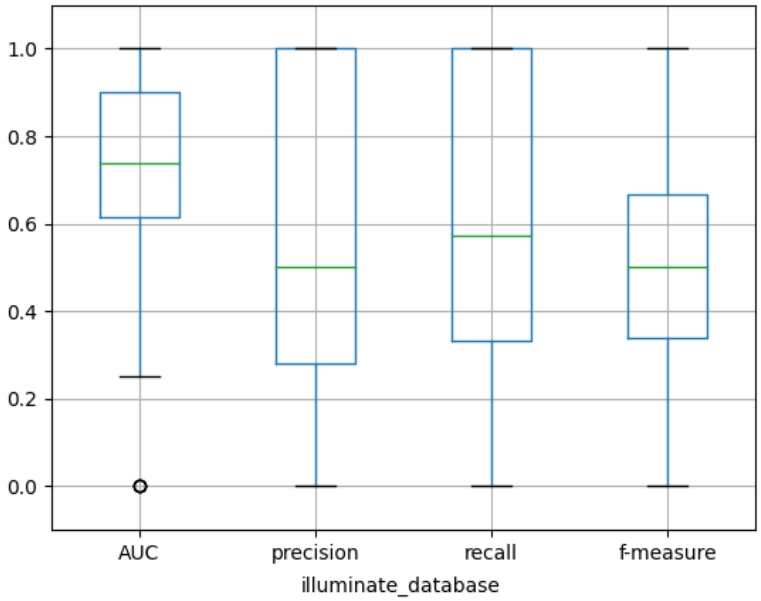

図 4.37　illuminate/database のデータの予測結果

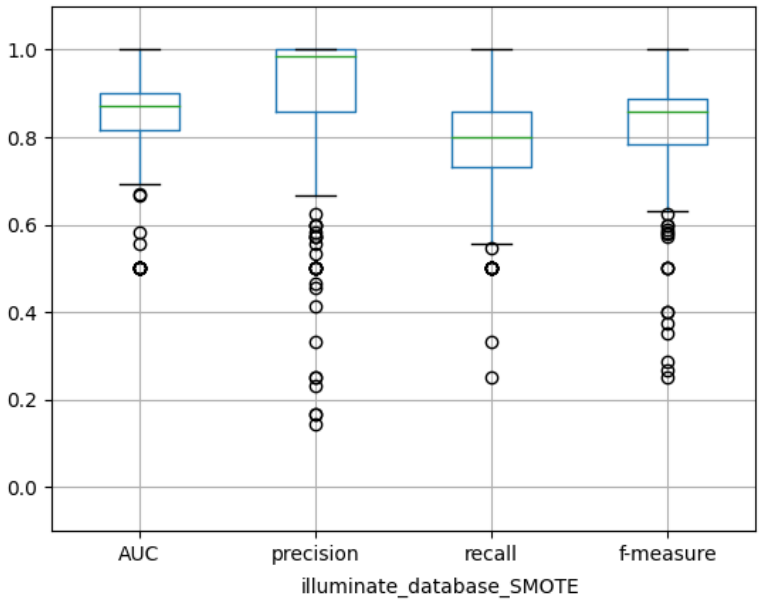

図 4.38　illuminate/database に SMOTE を適用したデータの予測結果

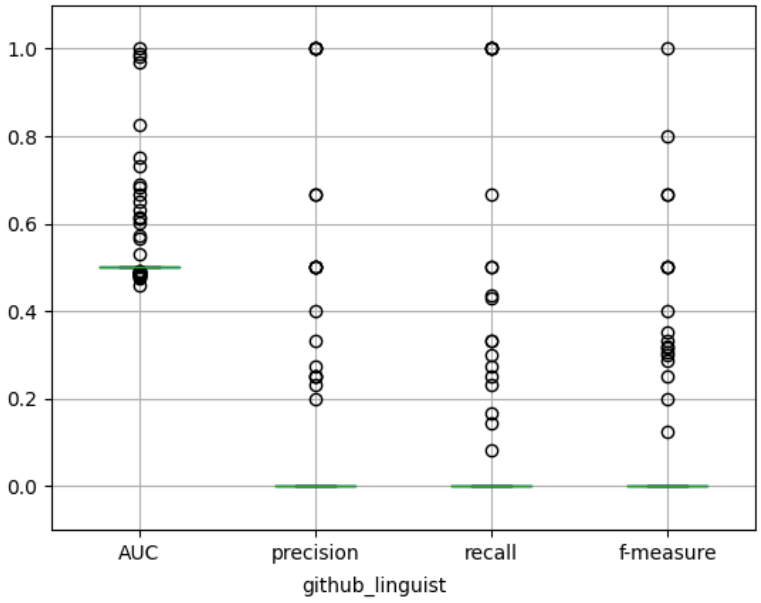

図 4.39　github/linguist のデータの予測結果

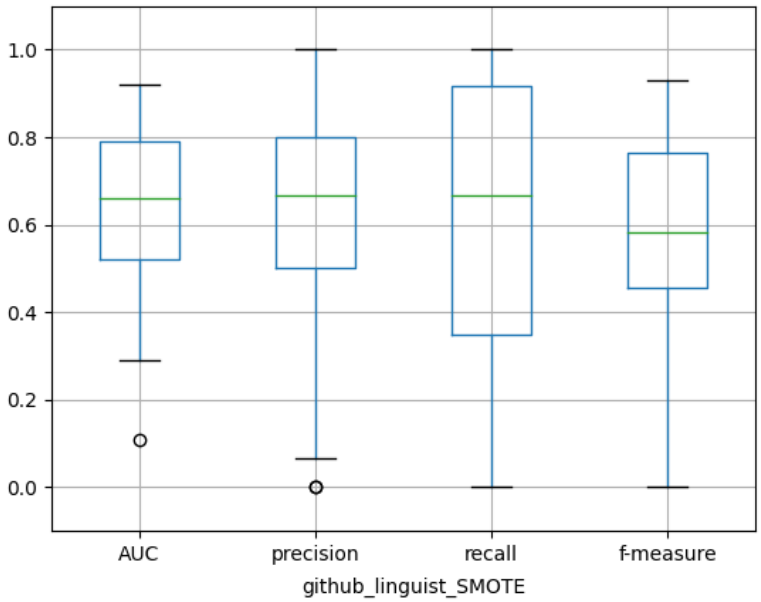

図 4.40　github/linguist に SMOTE を適用したデータの予測結果

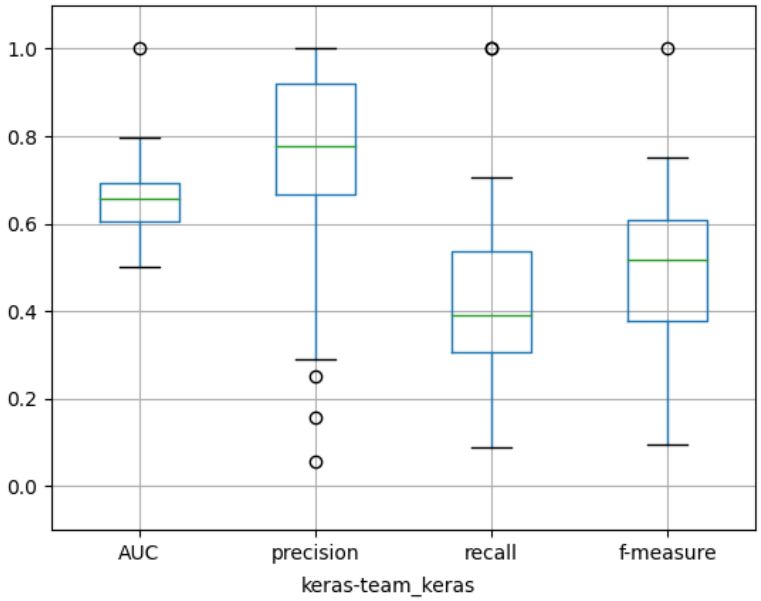

図 4.41　keras-team/keras のデータの予測結果

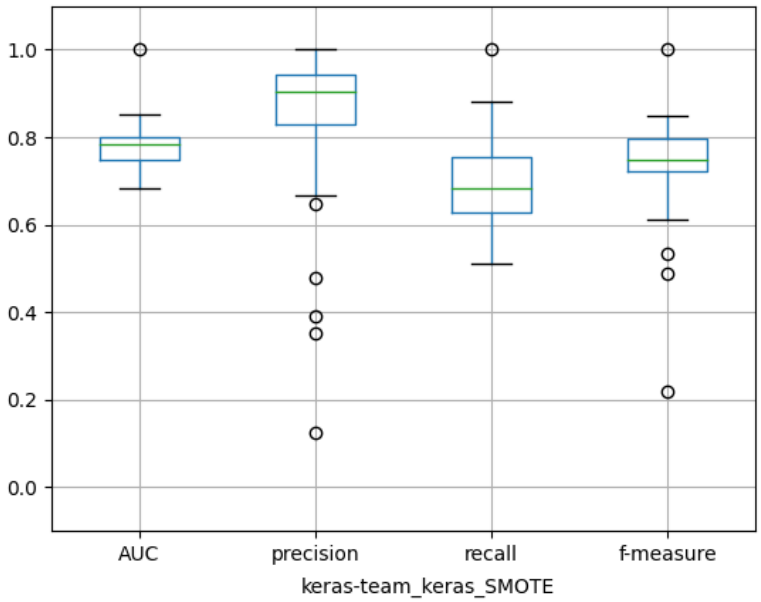

図 4.42　keras-team/keras に SMOTE を適用したデータの予測結果

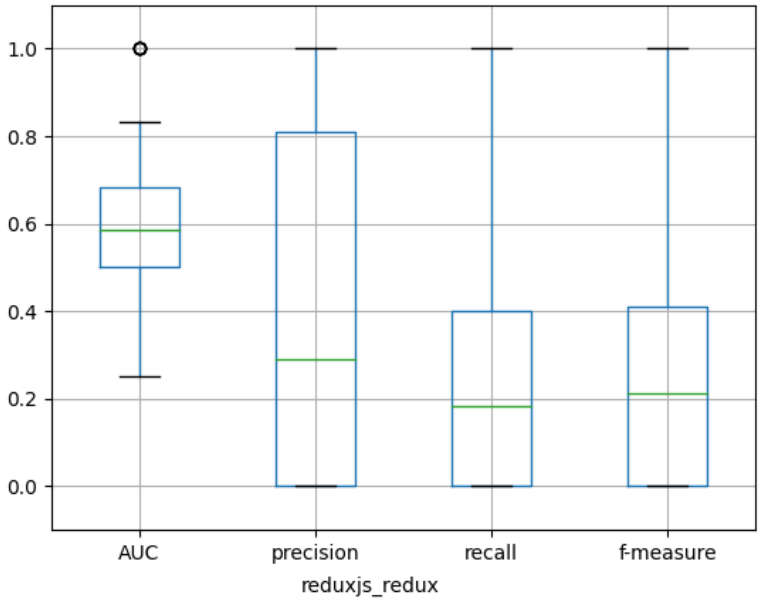

図 4.43　reduxjs/redux のデータの予測結果

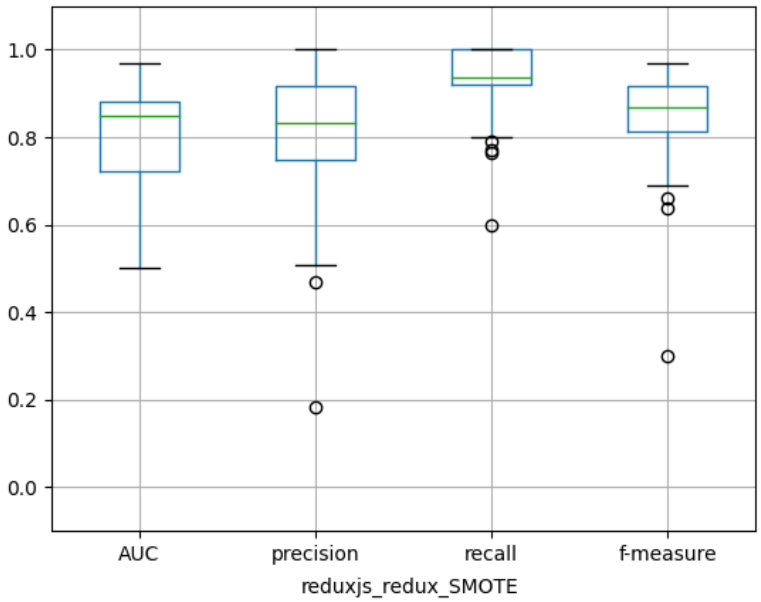

図 4.44　reduxjs/redux に SMOTE を適用したデータの予測結果

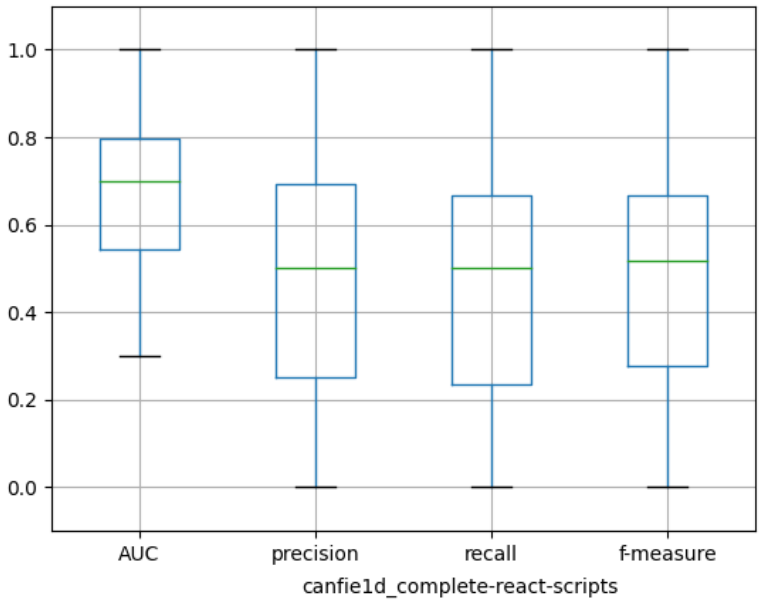

図 4.45　canfie1d/complete_react_scripts のデータの予測結果

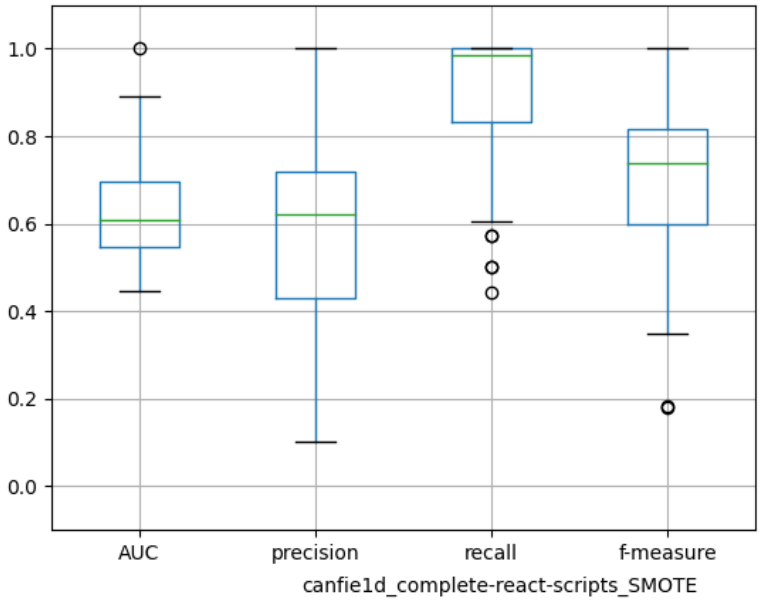

図 4.46　canfie1d/complete_react_scripts に SMOTE を適用したデータの予測結果

表 4.2　小規模なデータセットでの SMOTE 適用前後の予測結果の値

		SMOTE 適用前	SMOTE 適用後
AUC	第 1 四分位数	0.5402	0.6535
	中央値	0.6164	0.7447
	第 3 四分位数	0.6873	0.8056
	四分位範囲	0.1470	0.1521
Precision	第 1 四分位数	0.1797	0.6242
	中央値	0.3820	0.7447
	第 3 四分位数	0.6395	0.8056
	四分位範囲	0.4598	0.1521
Recall	第 1 四分位数	0.1297	0.6522
	中央値	0.2837	0.7737
	第 3 四分位数	0.4558	0.8632
	四分位範囲	0.3261	0.2110
F1 値	第 1 四分位数	0.1489	0.6320
	中央値	0.3173	0.7323
	第 3 四分位数	0.4514	0.8069
	四分位範囲	0.3024	0.1749

表 4.3　中央値の SMOTE の効果量の測定結果（Cohen's d）

	大規模なデータ	小規模なデータ
AUC	4.3591	1.6232
Precision	3.0491	2.1127
Recall	6.0922	3.1792
F1 値	5.4247	2.8085

ことは，全体的な予測精度が向上したことの裏付けと考えられる．今回の実験においては，SMOTE によるオーバーサンプリングの比率を，少数派クラスと多数派クラスの数が同じ数になるように設定した．そのため，実験の設定としてはオーバーサンプリングにかける計算量が比較的多いことが考えられる．したがって，SMOTE による少数派クラスのオーバーサンプリングではサンプル数の倍率を 200%や 300%などに設定して，オーバーサンプリングの比率を減らして予測精度に対する影響にどの程度変化があるか検証してみるべきであると考える．

4.2.2 リポジトリの大小による差異

RQ3 の結果より，大規模なリポジトリは小規模なリポジトリと比較して SMOTE からより良好な影響を得られることがわかった．今回の実験の設定として，SMOTE によるオーバーサンプリングは少数派クラスと多数派クラスの数が同じになるように設定した．そのため，小規模なプロジェクトと比較すると，大規模なプロジェクトの方がオーバーサンプリングによって生成されたサンプルの数が多いと推察される．そのため，より少数派クラスのサンプルが多い大規模なプロジェクトの方が SMOTE が有利に働いたと考えられる．プロジェクトの大小に関わらず，少数派クラスと多数派クラスの割合を揃えてオーバーサンプリングすることによって，プロジェクトの大小や種類が予測精度に影響するかさらに追究することができると考えられる．

5. 妥当性への脅威

5.1 内的妥当性

本実験に用いたスクリプトやプログラムに意図しない不具合が含まれている可能性がある．スクリプトの検証は行ったが，実験結果の妥当性の脅威となりうる．また，最もコミット数が多かった Homebrew プロジェクトに対する検証も，今回は行えていない．大きすぎるリポジトリに対する SMOTE の影響の変化を検証できていないので，今後，Commit Guru や実行環境の改善によって実際に取得し，影響の有無を確認したい．

5.2 外的妥当性

本実験で用いたリポジトリは，コントリビュータ数で上位 100 件を選択したものであり，結果を全てのリポジトリに一般化できない．今後，範囲を拡大して他のリポジトリにおいても SMOTE を用いることによって，不具合予測の結果に良い影響を与えることができるか検証したい．また，本実験で用いたデータセットは全てオープンソースソフトウェアから収集した．そのため，商用に開発されたソフトウェアとオープンソースソフトウェアでは性質が異なる可能性が考えられる．よって今後の課題として，商用に開発されたソフトウェアに対して同じ実験を行い，同様の効果が得られるか分析することは必要であると考えられる．

5.3 構成概念妥当性

RQ1 及び RQ2 で用いた評価値である，Precision，Recall，F-measure は，実験対象となるデータの偏りの影響を受けることがある．そのため，実験結果がこれに左右される可能性があることを考慮する必要がある．Tanithamthavorn らによると，閾値を使用しない評価指標を使用するべきであり，AUC はこの指摘を満たしている評価指標である [19]．しかし，一方で AUC のみを用いて不具合予測の結果の評価とすることは，結果にバイアスを混入させる恐れがある．そのため本実験では他の評価指標である Precision，Recall，F-measure も用いている．

本実験では，不具合予測にロジスティック回帰モデルを用いている．ロジスティック回帰モデルを選択した理由としては，既存手法においても用いられているためである [16]．しかし，ロジスティック回帰モデル以外にも機械学習手法によって不具合を予測する方法は存在している．そのため，その他の手法を用いた場合に予測結果にどのような変化が現れるか分析することは，今後の課題としてあげられる．

本実験におけるリリースの分割は，git の author date を用いて行った．具体的には，コミットとリリースの author date を比較し，あるリリースの author date よりもコミットの author date が過去のものであり，その前のリリースよりも未来のものであった場合，そのリリースに含まれるコミットとして分類した．しかし，この方法ではブランチを考慮せずに分割していることになり，厳密にはリリースごとに分けられていない可能性がある．ブランチとは，履歴の流れを分岐して記録していくものであり，ブランチごとに別のコミットやリリースが記録される．つまり，ブランチごとに時系列やコミット，リリースが存在していることになる．そのため，それらを 1 つにまとめて扱うことは，時系列や不具合の混同などにつながる可能性がある．

本研究の実験データへのラベル付け及びメトリクスの計算には，Commit Guru を用いている．Commit Guru はソースコードがおよび Web アプリケーションが公開されていることから，実験の再現性が高く，Commit Guru の採用は妥当であると考える．

6. 結言

本研究では，リリースを考慮したJIT DPにSMOTEが与える影響について分析した．SMOTEを適用した結果，リポジトリの規模に関わらず予測精度を向上させることができた．また，SMOTEによる効果量を比較すると，小規模なリポジトリと比べて大規模なリポジトリの方がより強い影響を受けていた．

しかし，本研究には，まだ多くの改善の余地がある．具体的には次の３つが挙げられる．

1. Homebrewプロジェクトのような，非常に大規模なコミット数を持ったリポジトリを使った実験の分析を行う．
2. 商用のソフトウェアに対してリリースを考慮したJIT DPとSMOTEを適用した場合の分析を行う．
3. ロジスティック回帰モデル以外の機械学習手法を用いた場合のSMOTEの影響の差異を分析する．

謝辞

本研究を行うにあたり，研究課題の設定や研究に対する姿勢，本報告書の作成に至るまで，全ての面で丁寧なご指導を頂きました，本学情報工学・人間科学系水野修教授，崔恩瀞助教に厚く御礼申し上げます．本報告書執筆にあたり貴重な助言を多数頂きました，九州大学システム情報科学研究院　近藤将成助教，岡山大学大学院自然科学研究科　西浦生成特任助教，サイボウズ株式会社　國領正真さん，本学情報工学専攻ソフトウェア工学研究室の皆さん，学生生活を通じて著者の支えとなった家族や友人に深く感謝致します．

参考文献

[1] M. Tan, L. Tan, S. Dara, and C. Mayeux, “Online Defect Prediction for Imbalanced Data,” Proceedings of the 2015 International Conference on Software Engineering (ICSE), pp.99–108, IEEE, 2015.

[2] Q. Li, B. Li, and J. Liu, “Predicting Developers’ Contribution with Developer Networks and Social Network Analysis,” Journal of Computational Information Systems, vol.7, no.15, pp.5553–5560, 2011.

[3] N. Bettenburg, M. Nagappan, and A.E. Hassan, “Think Locally, Act Globally: Improving Defect and Effort Prediction Models,” Proceeding of the 2012 IEEE International Working Conference on Mining Software Repositories (MSR), pp.60–69, IEEE, 2012.

[4] S. McIntosh and Y. Kamei, “Are Fix-Inducing Changes a Moving Target? A Longitudinal Case Study of Just-In-Time Defect Prediction,” IEEE Transactions on Software Engineering, vol.44, no.5, pp.412–428, 2018.

[5] 里形，“ソフトウェアリリースを考慮した 変更レベルの不具合予測の検証,” 卒業研究報告，京都工芸繊維大学情報工学課程，2020.

[6] K.E. Bennin, J.W. Keung, and A. Monden, On the Relative Value of Data Resampling Approaches for Software Defect Prediction, Empirical Software Engineering, 2019.

[7] L. Zhou, R. Li, S. Zhang, and H. Wang, “Imbalanced Data Processing Model for Software Defect Prediction,” Wireless Personal Communications, vol.102, no.2, pp.937–950, 2018.

[8] V.C. Nitesh, W.B. Kevin, O.H. Lawrence, and W.P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” Journal of Artificial Intelligence Research, vol.37, pp.321–357, 2002.

[9] A. Fernández, S. García, F. Herrera, and N.V. Chawla, “SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary,” Journal of Artificial Intelligence Research, vol.61, pp.863–905, 2018.

[10] R. Blagus and L. Lusa, “SMOTE for high-dimensional class-imbalanced data,” BMC Bioinformatics, vol.14, pp.1–16, 2013.

[11] michio，解説編: オーバーサンプリング手法解説 (SMOTE, ADASYN, Borderline-SMOTE, Safe-level SMOTE)，（オンライン），入手先〈https://qiita.com/eigs/items/8ae0970afe188a1124d1〉（参照 2022-02-05）.

[12] T. imbalanced-learndevelopers, Imbalanced-learn Documentation, （オンライン）, 入手先〈https://imbalanced-learn.org/stable/#〉（参照 2022-02-05）.

[13] I. Tidelift, Libraries.io, （オンライン）, 入手先〈https://libraries.io〉（参照 2022-02-05）.

[14] asmsuechan, yokoponzoo, コーエンの d を python で求める。, （オンライン）, 入手先〈https://kagglenote.com/misc/cohens-d/〉（参照 2022-02-05）.

[15] C. Rosen, B. Grawi, and E. Shihab, “Commit Guru: Analytics and Risk Prediction of Software Commits,” Proceedingsof the 2015 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE), pp.966–969, 2015.

[16] Y. Kamei, E. Shihab, B. Adams, A.E. Hassan, A. Mockus, A. Sinha, and N. Ubayashi, “A Large-Scale Empirical Study of Just-In-Time Quality Assurance,” IEEE Transactions on Software Engineering, vol.39, no.6, pp.757–773, 2013.

[17] scikit-learndevelopers, scikit-learn, （オンライン）, 入手先〈https://scikit-learn.org/stable/index.html〉（参照 2022-02-05）.

[18] y. asmsuechan，コーエンの d を python で求める。，

（オンライン）.

[19] C. Tantithamthavorn and A.E. Hassan, “An Experience Report on Defect Modelling in Practice: Pitfalls and Challenges,” Proceedings of International Conference on Software Engineering (ICSE), pp.286–295, ICSE, 2018.