Vol.2016-DPS-167 No.5
Vol.2016-MBL-79 No.5
Vol.2016-ITS-65 No.5
2016/5/26

ソーシャルコーディングにおける ユーザの行動に着目した有益提案の抽出

江見 圭祐[1]　乃村 能成[1]　谷口 秀夫[1]

概要：ソフトウェア開発において，ソーシャルコーディングと呼ばれる手法が広がりつつある．ソーシャルコーディングで進められるプロジェクトには，様々なソースコードの変更が提案される．しかし，これら提案には，有益な提案ばかりではなく，無益な提案も数多く存在する．そこで本稿では，提案の良し悪しを自動で判別する手法とその評価について述べる．具体的には，提案作成者の行動を学習し，提案の有益さを予測する．そして，算出された提案の有益さの精度を評価することで提案手法の有用性を示す．

キーワード：ソーシャルコーディング，仕事効率化

1. はじめに

ソフトウェア開発において，ソーシャルコーディングと呼ばれる手法が広がりつつある [1]．ソーシャルコーディングとは，ソーシャルネットワークの考え方をソフトウェア開発に適用した試みである．

ソーシャルコーディングでは，文章や写真ではなく，ソースコードを使って他ユーザとの交流を促進する．各ユーザは，ソーシャルネットワークシステム（SNS）で自分の写真アルバムを公開するのと同様にソフトウェアプロジェクトのソースコードを公開し，それに対して「イイネ」が付けられる．ソースコードに対するレビューや改善案を掲示板に書き込むことで，プロジェクトに関する議論を行う．また，これらのやりとりが公開されていることによって，誰がどのソフトウェアプロジェクトに興味があり，どの程度のソースコード量をどのプロジェクトに提供しているのか可視化され，ユーザのコーディングスキルやプロジェクトでの貢献度，興味のある分野が分かる．このようなユーザ間のやりとりを通して，ユーザの評価や評判が形成され，ユーザ間の交流と協調に繋がる [2]．

こうしたソーシャルコーディングの手法を取り入れた代表的なサービスとして GitHub[1] があり，2016 年 4 月現在で 1,400 万人のユーザと 3,500 万のソフトウェアプロジェクトを擁するまでとなっている [3]．GitHub は，ソースコードレビュー，バージョン管理，チケット管理，およびパッチ提示といった作業を支援する Web システムを提供している．そして，これらの作業履歴は記録され，可視化されている．

ソーシャルコーディングで開発が進められるプロジェクトでは，プロジェクトに対する理解度の低いユーザが，プロジェクトの方針と合わないソースコードの変更を提案することがある．提案にまつわるやりとりは，全ユーザに公開されているため，プロジェクトオーナーといえども理由もなく提案を拒否できない．たとえば，技術的に取り込むべきではないソースコードの変更の提案であっても，プロジェクトオーナーはソースコードのレビューやテストを行い，提案したユーザにソースコードの変更を取り入れない理由を説明する．このような対応による手間は，プロジェクトの円滑な進行を妨げるだけでなく，いわゆる「ソーシャル疲れ」という現象 [4] の原因となる．

無益提案の対応にかかる時間と労力をできるだけ抑えて有益提案の対応に割くために，プロジェクトオーナーは有益提案から優先的に議論したいという要求がある．しかし，提案の良し悪しは提案を吟味した後でなければ判断できない．このため，提案を吟味する前に提案の良し悪しを測れる尺度が必要である．そこで，本稿では議論が終了した過去の提案から提案の性質を学習し，提案の有益さを算出して，有益な提案を抽出する手法を提案する．まず，ソーシャルコーディングにおけるソフトウェア開発の流れについて述べ，ソーシャルコーディングにおける問題とその対処を明らかにする．次に，無益提案の割合の調査について述べ，ソーシャルコーディングにおける問題がどの程度発生するか確かめる．さらに，提案の有益さとユーザの行動

[1] 岡山大学大学院自然科学研究科
Graduate School of Natural Science and Technology, Okayama University

Vol.2016-DPS-167 No.5
Vol.2016-MBL-79 No.5
Vol.2016-ITS-65 No.5
2016/5/26

の関係を調査する．そして，有益提案を抽出する手法について述べる．最後に，提案手法を用いた有益提案の抽出の評価を行い，提案手法の有用性を示す．

2. ソーシャルコーディングにおけるソフトウェア開発

2.1 GitHub におけるソーシャルコーディング

まず，代表的なソーシャルコーディングサービスである GitHub における開発の流れを説明して，ソーシャルコーディングの特徴を説明する．詳細は文献 [5] に示されているため，ここでは概略を述べる．GitHub のユーザは，自身がプロジェクトオーナーとなってソフトウェアプロジェクトを開設できる．プロジェクトは他ユーザにも公開されており，プロジェクトオーナーだけでなく，他ユーザも開発に関わることができる．他ユーザは，公開されているプロジェクトから自身のプロジェクトを派生させ，自由にプロジェクトを改変できる．そして，他ユーザは自身が派生させたプロジェクトの改変を派生元プロジェクトにパッチとして提供できる．このような開発の流れは，ソーシャルコーディングの特徴的な開発形態である．以上の開発の流れを容易にするために，プロジェクトには以下の機能が提供される．

(1) ソースコードの共有スペース（repository）
repository は，バージョン管理機能を備えるソースコードとドキュメントの共有スペースである．また，repository は全ユーザが自由にコピー可能で，repository の派生関係はシステムで管理されている．これによって，他ユーザの改変をプロジェクトにパッチとして提供しやすくなっている．repository をコピーし，派生 repository を作成することを fork と呼ぶ．

(2) 全ユーザが自由に書き込める掲示板（issues）
プロジェクトの不具合や新たな機能を報告，議論できる機能として issues がある．issues で作成される報告を issue と呼ぶ．issue は，ソースコードの変更をともなうか否かによって分類できる．issues では，それぞれの issue について議論が行われる．

他ユーザのプロジェクトのソースコードに手を加えたい場合は，以下の手順を踏む．

(1) プロジェクトを fork して，派生プロジェクトを作成する．

(2) 派生プロジェクトのソースコードを改変する．なお，派生プロジェクトは自身のプロジェクトであるため，自由にソースコードを改変できる．

(3) fork 元プロジェクトに改良後のソースコードを取り込んでもらうように要求する．具体的には，fork 元のプロジェクトの issues に pull request を作成する．

fork 元のプロジェクトオーナーは，pull request に示されたソースコードを認めれば，取込み操作（merge）をするだけで当該ソースコードを簡単に取り込める．このように，上記機能の利用により，ユーザ同士はコミュニケーションを取りながらプロジェクトを進行でき，かつその手続きはシステム化されている．

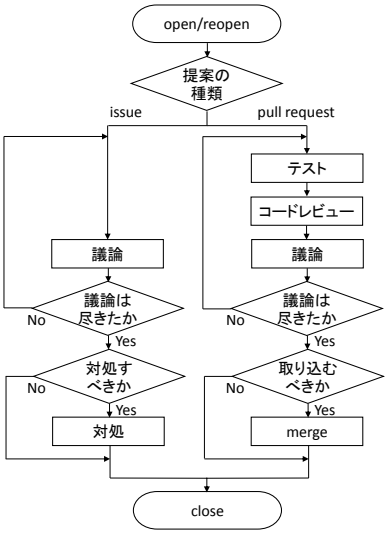

図 1 GitHub で提案が作成されてから対応が完了するまでの流れ図

2.2 GitHub における提案

ソーシャルコーディングでは，プロジェクトへのバグの発見報告や新機能の報告ができる．本稿では，このような報告を提案と呼ぶ．**図 1** を用いて，GitHub で提案が作成されてから対応が完了するまでの流れを説明する．提案を作成する動作を open と呼ぶ．提案には 2 種類あり，GitHub ではソースコードをともなわない提案は issue，ソースコードをともなう提案は pull request を使用する．

まず，issue の流れを説明する．issue が作成されると，プロジェクトオーナーと提案作成者を含む参加者は issue の内容について議論する．議論が尽きると，プロジェクトオーナーは当該 issue を対処するべきか判断する．対処する必要があると判断した場合，プロジェクトオーナーは報告されたバグを変更したり，提案された機能を追加する改変をソースコードに反映し，close という操作をもって提案を終了させる．一方，対処すべきではないと判断した場合は，提案作成者に提案に対処しない理由を説明し，提案を close する．

次に，pull request の流れを説明する．pull request が作成されると，プロジェクトオーナーはまず，変更を反映したソースコードのテストやコードレビューを行い，それを踏まえて議論する．議論の中で，提案作成者は提示したソースコードの変更を修正することもでき，その場合プロジェクトオーナーは再びソースコードのテストやコードレビューを行う．議論が尽きると，プロジェクトオーナーは当該 pull request を取り込むべきか判断する．取り込む必要があると判断した場合，プロジェクトオーナーは取り込

み操作（merge）を行い，プロジェクトにソースコードの変更を反映させ，提案を close する．一方，取り込むべきではないと判断した場合は，提案作成者に提案を取り込まない理由を説明し，提案を close する．

close された提案であっても，議論の余地があるとプロジェクトオーナーに判断された提案は reopen と呼ばれる操作を行うことができ，open された提案と同様の状態にできる．

2.3 ソーシャルコーディングの問題と課題

ソーシャルコーディングでは，すべての開発者が自由に提案を作成できる．GitHub においては，すべてのユーザが提案を作成可能で，全提案はシステム上で平等に扱われる．このため，プロジェクトのコアメンバからだけでなく，世界中の様々なユーザから提案が寄せられ，提案の内容がプロジェクトに反映される．これは，ソーシャルコーディングの利点であるが，欠点にもなる．ソーシャルコーディングでは，世界中の不特定多数の開発者がプロジェクトに参加するため，参加者の中には，プロジェクトへの理解度の低い者も多い．また，プロジェクトの参加者は時間を経て変化するため，プロジェクト開始当初の方針を知らない参加者が増えることもある．このような参加者は，すでに議論された内容や技術的に取り込むべきではない内容の無意味な提案を作成することがある．本稿では，このような提案を無益提案と呼ぶ．無益提案はプロジェクトに取り込まれないばかりか，プロジェクトオーナーや他の開発者の手間を増やす．対して，プロジェクトに取り込まれる提案や，多くの開発者によって活発に議論が交わされる提案を有益提案と呼ぶ．

開発に参加するユーザの数の多いプロジェクトでは，無益提案が増加し，全提案数の半分以上を占めると示されている [5][6]．プロジェクトオーナーはなるべく有益な提案から対応し，提案の内容をプロジェクトに反映させたいという要求がある．そのような状況で以下の問題が発生する．

問題　多数の提案の中から有益なものだけを発見する手間が大きい

これは，提案を吟味しなければ，提案の良し悪しを判断できないためである．

この問題を解決するための課題として以下が挙げられる．

課題　提案を吟味する前に提案の良し悪しを測れる尺度の導入

プロジェクトオーナーは提案のタイトルを見てその提案の良し悪しを判断しようとすることもあるが，提案の良し悪しを測る尺度として十分ではない．提案の良し悪しを測るためにふさわしい尺度を導入することで，膨大な提案から優先的に有益提案に対応できる．

2.4 目的

本稿では，2.3 節で述べた課題の対処として以下を目的とする．

目的　有益提案が持つ性質を学習し，提案作成時点で提案の有益さを算出

これにより，プロジェクトオーナーは提案を吟味する前に提案の良し悪しを判断することができる．算出された有益さを基に高確率で有益だと考えられる提案を抽出することで，優先的に有益提案に対応できる．

ここで，提案のうち，特にソースコードの変更をともなう提案は，プロジェクトオーナーがソースコードのレビューやテストを行う必要があり，プロジェクトオーナーの負担が大きく，提案が無益だった場合のコストが高い．このため，ソースコードの変更をともなう提案から有益提案を抽出することは，ソースコードの変更をともなわない提案から有益提案を抽出することと比べて，プロジェクトオーナーの負担の軽減だけでなく，プロジェクトの円滑な進行を促進すると考える．したがって，本稿では，ソースコードの変更をともなう提案に焦点を絞る，

3. 無益提案の割合の調査

3.1 調査目的

まず，参加するユーザが多いプロジェクトで作成されるソースコードの変更をともなう提案を有益なものと無益なものに分別し，無益提案の割合を調査する．これによって，2.3 節で述べた問題がソースコードの変更をともなう提案でどの程度発生し，プロジェクトオーナーにとって深刻となるのか確認する．以降，本稿ではソースコードの変更をともなう提案のことを特に提案と呼ぶ．ここで，提案の調査は GitHub で公開されるプロジェクトを対象に行う．

3.2 有益提案の条件

GitHub 上の有益な提案の特徴として以下の 2 つが挙げられている [5]．

特徴 1　提案が示すアイデアやソースコードがプロジェクトに取り込まれている

特徴 2　提案に対して活発に議論が行われた

これらの特徴を GitHub から機械的に判別するための条件に置き換えると以下の 4 つとなる．本稿では，これらの条件の内，1 つでも満たす提案を有益提案とし，条件を 1 つも満たさない提案を無益提案とする．

条件 A　提案が merge されている

条件 B　提案が commit から参照されている

条件 C　提案が commit message により close されている

条件 D　提案が close 後に reopen されている

3.3 調査結果

3.2 節で述べた有益提案の定義を基に提案の内訳につい

表 1 無益提案の割合の調査結果（2015 年 9 月から 2016 年 1 月）

プロジェクト名	fork 数	全提案数	無益提案数	有益提案数	無益提案の割合
angular.js	21,264	5,966	3,326	2,640	55.7 %
bootstrap	38,748	5,301	2,622	2,679	49.5 %
jquery	9,828	1,961	1,149	812	58.6 %
rails	11,807	13,510	3,830	9,680	28.3 %

て調査した．調査を行ったプロジェクトは，angular.js，bootstrap，jquery，および rails の 4 つである．これらは，GitHub で進められる中でも fork 数が多い，つまり，参加するユーザの多いプロジェクトである．

これらのプロジェクトについて，fork 数，提案数の内訳，および無益提案の割合を**表 1** に示す．表 1 から，無益提案の割合が 50 %前後のプロジェクトが多いと分かる．このため，ソースコードをともなう提案について，2.3 節で述べた問題は深刻になると考える．

なお，rails の無益提案の割合が他のプロジェクトと比べて低い理由は，提案を作成する際のガイドライン [7] が細かく指定されているためだと考える．たとえば，rails のガイドラインでは，提案にソースコードの変更とともに動作を確認できるテストを添えることを推奨している．

4. 有益提案の抽出手法

4.1 有益提案の抽出

2.4 節で述べた目的を実現するために，提案が持つ性質を学習することで有益提案を予測し，有益な確率の高い順に提案を提示するシステムを提案する．提案手法が有効に機能していれば，整列された提案の上位に有益提案が集まる．プロジェクトオーナーは提示された提案の上位を抽出すれば，限られた時間で効率的に有益提案に対応できる．このため，提案手法では，提案が有益か否かの 2 値分類ではなく，提案の有益さの指標を算出する．

4.2 学習に用いる素性の選定

4.2.1 素性の選定方針

有益提案の学習と抽出のために，どのような要素をパラメータとして扱うか検討し，素性を選定する．素性とは，学習に使用するパラメータのことを指す．

3.2 節で述べたように提案の良し悪しは議論が終了した後に判別できる．提案手法では，議論が終了した提案を教師データとして扱う．この教師データを学習し，新たに作成された提案の有益さを予測する．素性は，教師データのどの要素を学習して提案の有益さを予測するかを決めるものであり，提案作成時に分かる要素を用いる必要がある．つまり，3.2 節で述べた有益提案の条件は，提案の議論が終了した後に分かる特徴であるのに対して，素性は提案の議論が始まる前，提案が作成された時点で分かる特徴となる．

本節では，学習に使用する素性として，どのような要素がふさわしいかを調査し，調査結果より提案手法で採用する素性を選定する．

4.2.2 ユーザの行動から得られる要素

GitHub では，どのユーザがいつ，どのような提案を作成したか，また，ユーザがどのようなファイルを変更したかといった行動が記録されている．このようにシステムに蓄積されたユーザの行動に着目することの有為性は既存研究で示されている．たとえば，ユーザがあるファイルに行ったバグフィックスの回数とバグフィックスを行った時間を基に相対的なスコアを算出することで，バグを含むファイルを予測するバグ予測アルゴリズムが提案されている [8]．これは，ソースコードの静的解析というアプローチだけではなく，ユーザの行動からもそのファイルの特性を得られることを示す．そこで，本稿ではシステムに記録されているユーザの行動に着目し，ユーザの行動から得られる要素と提案の有益さとの関係を調査する．

ここで，「提案を誰が作成したか」や「提案がいつ作成されたか」といった要素は，提案者が「誰なのか」といったことによって左右される要素であるといえる．プログラミングが人に強く依存する行為である．とりわけ，人は SNS における重要な要素であり，ソフトウェアプロジェクトにおける人間関係を強く反映している要素でもある．実際，ユーザが誰なのかという要素は，そのソフトウェアプロジェクトでの地位や貢献度と直結している [9]．このことから，提案者が「誰なのか」という要素は，提案の良し悪しと密接に関わると考える．そこで，本稿では以下の 2 つの要素を挙げ，素性に採用する．

(要素 1) 提案作成者

提案作成者は，提案内容の考案者である場合が多い．提案作成者は，プロジェクトに対する理解の低いユーザである場合もあるし，プロジェクト開発当初から携わる理解の深いユーザである場合もある．提案作成者のプロジェクトの理解度によって，提案の有益さは変化する．

(要素 2) 提案が提出された時間帯

提案が提出される時間帯の傾向はプロジェクトによって異なる．たとえば，ユーザが企業の業務として提案を作成することが多いプロジェクトもあれば，有志の開発者によって開発が進むプロジェクトもある．ユーザによってプロジェクトに参加する時間帯は異なり，理解度の深いユーザが多く参加する時間帯もプロジェ

クトによって偏りがある．

一方，本提案手法では，そのソフトウェアプロジェクトが何であるかに依存した素性も必要であると考える．すなわち，プロジェクトのファイルの構成や開発の進度（フェーズ）といった特性である．これは，repository 内におけるファイルの変更履歴に強く現れていると考えられる．ソーシャルコーディングにおける人を重視した要素だけでなくこれら 2 つの要素を加味することで，ソフトウェアプロジェクトに新規に加わった者による有益提案を見逃さないようにする利点もあると考える．そこで，本稿では以下の 2 点も要素として考える．

(要素 3) 提案で変更されたファイル
提案作成者が提案で変更したファイルによって，その提案が有益か否かを判断できると考える．たとえば，複雑な処理を行うソースコードは，バグを含みやすい．つまり，有益提案で変更される可能性が高い．

(要素 4) 提案で変更されたファイルの最終変更日時から提案作成日時までの日数
提案作成者が提案で変更したファイルを提案以前にいつユーザが変更したかによって，その提案が有益か否かを判断できると考える．たとえば，1 日前に変更されたファイルは，直前の変更で新たに発生したバグやタイプミスを含む可能性がある．一方，1 年間変更されていないファイルには，1 日前に変更されたファイルに比べて，バグやタイプミスは少ないと考えられる．

4.2.3 調査対象の要素とプロジェクト

4.2.2 項で述べた（要素 3）と（要素 4）は，（要素 1）と（要素 2）と比べて，提案の有益さとの関係が明らかでない．このため，次項では（要素 3）と（要素 4）が提案の有益さとどのような関係をもつか調査し，それぞれの要素が学習に用いる素性としてふさわしいか検討する．なお，調査には表 1 で示したプロジェクトの内，最も fork 数の多い bootstrap を用いる．

4.2.4 提案で変更されたファイルと提案の有益さの関係

まず，（要素 3）と提案の有益さの関係の調査として，有益提案で変更されたファイルと無益提案で変更されたファイルを集計し，ファイルごとに有益提案で変更された割合を算出した．ファイルごとの有益提案で変更された割合とは，あるファイルを変更する提案が複数存在するときにその内の何割が有益提案であるかを表す．たとえば，お互いに異なる提案 A，B，C，および D があり，すべての提案がプロジェクトに存在するファイル f の変更を含んでいるとする．これらの提案はすべてファイル f を変更しているが，提案の良し悪しは異なる．ここで，提案 A は有益提案，提案 B，C，および D は無益提案だったとすると，ファイル f の有益提案で変更された割合は 4 分の 1 となる．一般的には，あるファイルが有益提案で変更された回数を N_{useful}，無益提案で変更された回数を $N_{useless}$ とすると有益提案で変更された割合は（1）式で算出できる．

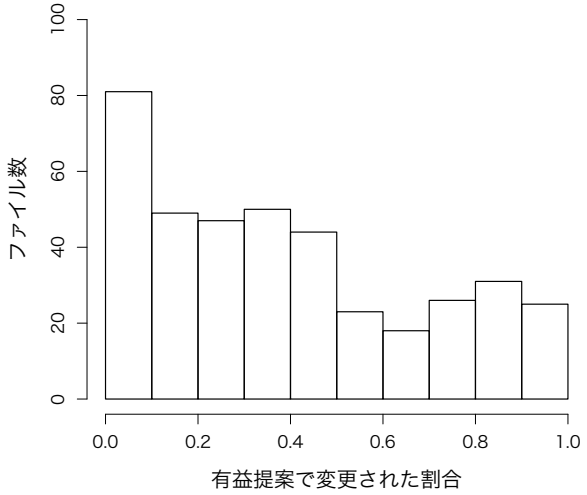

図 2 有益提案で変更された割合ごとのファイル数のヒストグラム

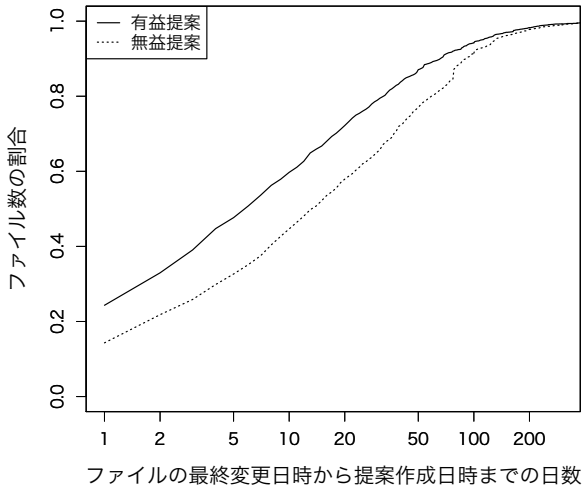

図 3 有益提案と無益提案ごとの変更したファイルの変更日時から提案作成日時までの日数とファイル数の割合の累積グラフ

$$\frac{N_{useful}}{N_{useful} + N_{useless}} \tag{1}$$

ここで，提案で変更される回数が少ないファイルは，有益提案で変更された割合が極端な値になってしまうため，10 個以上の提案で変更されたファイルを集計する．**図 2** に有益提案で変更された割合ごとのファイル数のヒストグラムを示す．図 2 は，横軸が有益提案で変更された割合，縦軸がファイル数を表す．図 2 から，有益提案で変更された割合が 50 %以下のファイルの総数が多いことがわかる．特に有益提案で変更された割合が 10 %以下のファイルの総数は 80 個を超えている．これは，無益提案では特定のファイルを変更することが多いことを示している．このため，（要素 3）は提案の有益さと相関があり，素性にふさわしいと考える．

4.2.5 提案で変更されたファイルの最終変更日時から提案作成日時までの日数と提案の有益さの関係

次に，（要素 4）と提案の有益さの関係の調査として，変

更したファイルの最終変更日時から提案作成日時までの日数によって，有益提案と無益提案の割合に差があるかを調査する．横軸にファイルの最終変更日時から提案作成日時までの日数，縦軸にファイル数の割合を示す累積グラフを**図 3** に示す．図 3 では，実線が有益提案の推移，点線が無益提案の推移を表す．図 3 から，有益提案で変更したファイルは最終変更日時から提案作成日時までの日数が短いものが多いことが分かる．一方，無益提案は有益提案と比べて，最終変更日時から提案作成日時までの日数が長いファイルを多く変更していると分かる．たとえば，bootstrap では最終変更日時から提案作成日時までの日数が 9 日未満のとき，有益提案と無益提案の割合の差が最も大きくなる．具体的には，有益提案は約 56.3 %であるのに対し，無益提案は約 40.4 %である．このため，（要素 4）は提案の有益さと相関があり，素性にふさわしいと考える．

4.2.6 提案手法で用いる素性

4.2.2 項で述べた内容から学習に用いる素性を以下の 4 つとする．

(素性 1) 提案作成者
(素性 2) 提案が提出された時間帯
(素性 3) 提案で変更されたファイル
(素性 4) 提案で変更されたファイルの最終変更日時から提案作成日時までの日数

上記の素性は 4.2.2 項で述べた（要素 1）から（要素 4）と対応している．ここで，（素性 2）と（素性 4）は時間，または日数を表す連続値であるが，素性として扱いやすくするために，離散化して利用する．（素性 2）は 1 日を 4 つに離散化し，UTC 時間で大まかに分類する．（素性 4）は 4.2.5 項での調査から，9 日を閾値として離散化する．

4.3 学習に用いるアルゴリズム

本稿では，提案手法で用いるアルゴリズムとして，単純ベイズ分類器を利用する．単純ベイズ分類器を利用する理由は，以下の 2 つである．

(1) 次元数の多い素性に適する
提案手法は，素性として提案で変更されたファイルを利用するため，ファイルの種類数だけ次元数が増加する．素性の次元数が増加すると，指数関数的に学習に必要なデータ集合が増大する問題が発生する．単純ベイズ分類器は，素性間に独立性を仮定するため，この問題による影響を緩和できる．

(2) クラスごとの確率モデルを生成できる
単純ベイズ分類器では，各クラスごとに事後確率を生成する．ここで，提案手法におけるクラスとは，提案が有益か無益かである．提案を有益としたときの事後確率は提案の有益さとみなせる．このため，提案の有益さで提案の整列を行う提案手法に適する．

以上のことから，提案手法では単純ベイズ分類器を利用して提案の有益さを算出する．

5. 抽出結果の評価

5.1 評価

提案手法の評価は以下の 2 つである．

(評価 1) 提案の順位付けの精度
(評価 2) プロジェクトオーナーに提示できる実際の有益提案数

まず（評価 1）は，提案の順位付けの精度である．提案手法では，議論が終了した提案を学習し，議論が終了していない提案の有益さの指標を算出する．算出した有益さから，提案を有益さの降順で整列し，プロジェクトオーナーに提示する．このとき，有益提案がどれ程上位に集中しているかを評価することで，提案の順位付けの精度を評価できる．具体的な評価方法として，（評価方法 1）を以下に示す．

(評価方法 1) 提案を無秩序に並べた場合と提案手法から得られる有益さで提案を整列した場合の順位付けの精度を比較する

次に（評価 2）は抽出できる実際の有益提案数である．プロジェクトオーナーが提案手法を用いて，有益さの高い提案の上位から対応する状況を想定する．このとき，プロジェクトオーナーが対応する提案数を 1 日に対応できる提案数とする．この提案数を抽出した場合に抽出した提案に含まれる有益提案数を評価する．具体的な評価方法として，（評価方法 2）を以下に示す．

(評価方法 2) 無作為に提案を抽出した場合と提案手法を用いて提案を抽出した場合の実際に抽出できる有益提案数を比較する

5.2 評価環境

本稿では，評価を行うプロジェクトとして，表 1 で調査したプロジェクトを選定する．表 1 に示した各プロジェクトの提案から，無作為に選んだ 9 割の提案を学習データとし，学習データを除いた 1 割をテストデータとする．

5.3 評価方法

5.3.1 評価 1: 提案の順位付けの精度

（評価 1）では，提案の順位付けの評価指標として，平均適合率（Average Precision: AP）を使用する．平均適合率とは，テストデータを上位から走査し，有益提案が現れた時点での適合率の平均値である．平均適合率は，提案の順位付けを再現率と適合率の観点から総合的に評価する指標である．平均適合率が高いほど，有益提案が上位に集中していることを示す．テストデータ中に含まれる有益提案の総数を N，整列された提案の上位から i 番目の有益提案時点での適合率を P_i とすると，平均適合率 AP は，(2) 式で求められる．

表 2 プロジェクトごとの 1 日に処理された提案数の最小値，中央値，および最大値

プロジェクト名	最小値	中央値	最大値
angular.js	1	3	64
bootstrap	1	3	45
jquery	1	1	25
rails	1	7	47

$$AP = \frac{1}{N}\sum_{i=1}^{N} P_i \tag{2}$$

また，テストデータによる評価結果の偏りを緩和させるために，ランダム関数のシード値を変えながら評価を 10,000 回試行し，平均適合率の平均値を求める．この平均適合率の平均値を無秩序に並べた場合と提案手法から得られる有益さで提案を整列した場合で比較する．

5.3.2 評価 2: プロジェクトオーナーに提示できる実際の有益提案数

（評価 2）では，対応していない提案の中からプロジェクトオーナーが上位から k 個までの提案を処理したときに，どの程度の有益提案に対応できるかを評価する．

まず，評価を行うプロジェクトについて，プロジェクトオーナーが提案を処理する際，どの程度の提案数を 1 日にまとめて処理するかを調査する．プロジェクトごとの 1 日に処理された提案数の最小値，中央値，および最大値を**表 2** に示す．表 2 から，どのプロジェクトにおいても 1 日に処理される提案数の最小値と中央値の差が小さい．このことから，1 日に処理される提案数は少ないことが分かる．また，1 日に集中して提案を処理する場合も多くて 64 個の提案を処理する程度である．このため，k の取る値の範囲は 1 から 70 までとする．

k の取り得る値について，上位 k 個の提案を無作為に抽出した場合と提案手法を用いて抽出した場合の適合率 ρ_k の推移を求める．適合率とは，抽出した提案に含まれる有益提案数の割合である．ここで，適合率 ρ_k は，テストデータをランダムに 10,000 回選定した各適合率の平均値である．ρ_k より，上位 k 個の提案を抽出したときに実際に抽出できる有益提案数の期待値は $k\rho_k$ である．無作為に提案を抽出した場合と提案手法を用いて提案を抽出した場合の ρ_k の推移と $k\rho_k$ を比較する．

5.4 評価結果

5.4.1 評価 1: 提案の順位付けの精度

5.3.1 項で述べた評価方法にしたがって，表 1 に示したプロジェクトについて平均適合率を求める．プロジェクトごとの提案手法を用いない場合と提案手法を用いる場合の平均適合率を**図 4** に示す．図 4 から，すべてのプロジェクトについて，提案手法を用いた平均適合率は提案手法を用いない場合と比べて向上していることが分かる．

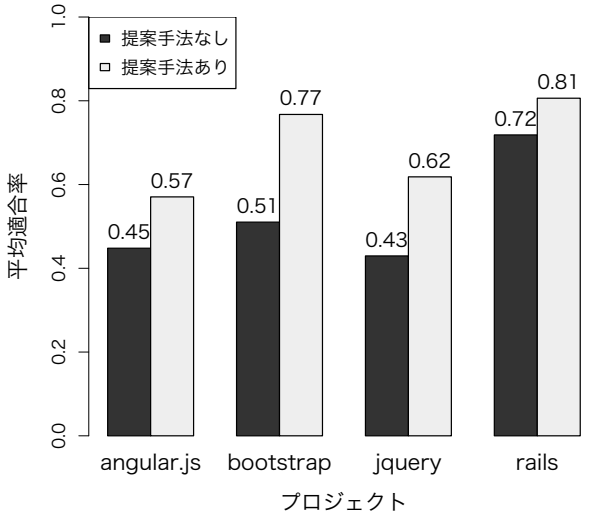

図 4 提案手法を用いない場合と用いた場合の平均適合率

最も平均適合率が向上しているプロジェクトは bootstrap である．bootstrap の提案手法を用いない場合の平均適合率が 0.51 であるのに対し，提案手法を用いた場合の平均適合率は 0.77 であり，0.26 向上している．これは，提案手法で使用する素性を決定する際にプロジェクトの特徴の調査を行ったのが bootstrap であったため，学習の精度が高くなったと考える．しかし，他のプロジェクトにおいても平均適合率は向上しており，bootstrap の特徴の調査から選定した素性は，他のプロジェクトの学習においても有効であるといえる．一方，最も平均適合率が向上しなかったプロジェクトは rails である．rails の平均適合率の向上率が低い原因は，他のプロジェクトに比べて rails の有益提案の割合が高く，提案手法を用いない場合の平均適合率が高いためだと考える．rails のように，有益提案の割合が高いプロジェクトでは，提案手法を用いることによる効果が出にくい．しかし，効果の出にくい rails の場合も，提案手法を用いた場合の平均適合率は 0.81 であり，提案手法を用いない場合の平均適合率である 0.72 から 0.09 向上している．以上のことから，提案手法を用いた提案の順位付けは有効に機能しているといえる．

5.4.2 評価 2: プロジェクトオーナーに提示できる実際の有益提案数

5.3.2 項で述べた評価方法にしたがって，表 1 に示したプロジェクトに関して，k 個の提案を抽出したときの ρ_k の推移と $k\rho_k$ を求める．

まず，k の取り得る値について，上位 k 個の提案を無作為に抽出した場合と提案手法を用いて抽出した場合の適合率 ρ_k の推移を**図 5** に示す．図 5 は，横軸が抽出する提案数，縦軸が適合率である．図 5 から，すべてのプロジェクトについて，提案手法を用いて提案を抽出した場合の適合率の推移は，無作為に提案を抽出した場合の適合率の推移を上回っていることが分かる．このため，提案手法は有効に機能しているといえる．

Vol.2016-DPS-167 No.5
Vol.2016-MBL-79 No.5
Vol.2016-ITS-65 No.5
2016/5/26

表 3 プロジェクトごとの提案手法を用いず提案を抽出した場合と用いて抽出した場合の実際に抽出できる有益提案数の期待値

		実際に抽出できる有益提案数の期待値	
プロジェクト名	抽出する提案数	提案手法を用いない場合	提案手法を用いる場合
angular.js	64	28.3	41.2
bootstrap	45	22.8	39.4
jquery	25	10.4	17.7
rails	47	33.8	38.1

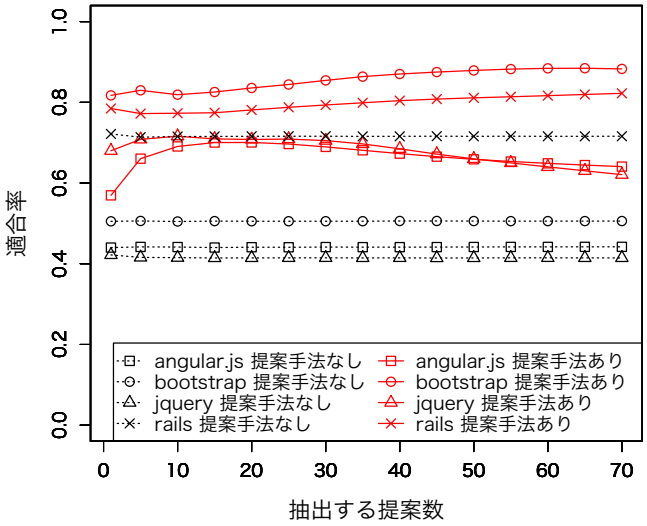

図 5 上位 k 個の提案を提案手法を用いず抽出した場合と用いて抽出した場合の適合率の推移

図 5 の提案手法を用いた場合の適合率の推移をプロジェクトごとに比較すると，以下の 2 つが分かる．

(1) 抽出する提案数が少ないとき，angular.js と bootstrap，および rails の適合率が低い

適合率の定義から，提案手法で算出される有益さが理想的であれば，適合率は抽出する提案数が少ない場合に高くなる．つまり，適合率の推移は横ばい，もしくは緩やかに減少する．しかし，angular.js と bootstrap，および rails は，抽出する提案数が 10 個以降の適合率と比べて，1 個から 10 個までの適合率は低くなっている．これは，いくつかの無益提案の有益さが不当に高く算出されているためだと考える．いくつかの無益提案の有益さが不当に高く算出される原因については，今後のさらなる調査を要する．

(2) 抽出した提案数が多くなるにつれて，jquery の適合率が大きく減少する

抽出した提案数が多くなるにつれて，jquery の適合率は，他のプロジェクトと比べ，減少傾向にある．これは，jquery のテストデータに含まれる有益提案の総数が他のプロジェクトよりも少ないためだと考える．jquery のテストデータは全提案数約 1900 個の 1 割の約 190 個であり最も少ない．さらに，有益提案の事前確率は約 40 %であるため，テストデータに含まれる有益提案数は約 76 個になる．このため，抽出する提案数が多くなると，他のプロジェクトと比べて無益提案が含まれる確率が高くなり，適合率が低下しやすい．これは，ある意味当然の結果である．

次に，1 日に集中して提案を処理する際，無作為に提案を抽出した場合と提案手法を用いて抽出した場合の実際に対応できる有益提案数を比較する．ここで，プロジェクトごとの 1 日に集中して提案を処理する際の抽出する提案数は，表 2 に示した 1 日で処理された提案数の最大値を用いる．**表 3** に提案手法を用いず提案を抽出した場合と提案手法を用いて提案を抽出した場合の実際に抽出できる有益提案数の期待値をプロジェクトごとに示す．表 3 から，期待値の増加が最も大きいプロジェクトは bootstrap で，約 17 個の増加が期待できる．期待値の増加が最も小さいプロジェクトである rails に関しても，約 4 個の増加が期待できる．以上のことから，提案手法は提案の抽出時に，有益提案をより多く抽出できるといえる．

6. おわりに

本稿では，ソーシャルコーディングで作成される提案の良し悪しを自動で算出し，有益な提案を抽出する手法とその評価について述べた．まず，ソーシャルコーディングで作成される提案の中から有益なものだけを発見する手間が大きいことを問題として挙げた．具体的には，ソーシャルコーディングサービスである GitHub を対象として，参加するユーザが多いプロジェクト内で無益提案が占める割合を調査した．その結果，fork 数の多い 4 つのプロジェクトでは，無益提案の割合が約 30 %から約 50 %を占めていることが分かった．さらに，ソーシャルコーディングにおけるユーザの行動に着目し，ユーザの行動と提案の有益さの関係を示した．そして，調査結果から，有益な提案の学習のための素性を決定し，単純ベイズ分類器を用いた提案の抽出手法を示した．最後に，提案の順位付けの精度とプロジェクトオーナーに提示できる実際の有益提案数を評価した．評価の結果，提案手法を用いた場合の平均適合率は，提案手法を用いない場合と比べて最高で 0.26，最低でも 0.09 向上した．また，実際に抽出できる有益提案数の期待値は最大で 22.8 個から 39.4 個，最小でも 33.8 個から 38.1 個に増加した．以上のことから，提案手法の有用性を示した．

本稿で述べた提案手法では，提案で変更したファイルを素性として学習を行う．これは，単語をファイル名とみな

したときの文書分類問題と等しい．このとき，一般的な文書分類問題と比較して，提案で変更するファイル数は極端に少ない．このため，学習精度の向上を妨げていると考えられる．そこで，今後は有益提案が持つ特徴を再調査し，学習に使用する素性を検討する．また，提案の学習で素性として用いる変更したファイルの最終変更日時から提案作成日時までの日数は，閾値を 9 日として離散化する．しかし，プロジェクトごとに最適な閾値は異なると考えられる．そこで，今後はプロジェクトごとに最適な閾値を設定する手法について検討する．

参考文献

[1] GitHub, Inc.: GitHub, GitHub, Inc. (online), available from ⟨https://github.com/⟩ (accessed 2016-04-19).
[2] Dabbish, L., Stuart, C., Tsay, J. and Herbsleb, J.: Social coding in GitHub: transparency and collaboration in an open software repository, *ACM*, pp. 1277–1286 (2012).
[3] GitHub, Inc.: Press, GitHub, Inc. (online), available from ⟨https://github.com/about/press⟩ (accessed 2016-04-19).
[4] Yamakami, T.: Towards understanding SNS fatigue: exploration of social experience in the Virtual World, *Proc. 2012 7th International Conference on Computing and Convergence Technology (ICCCT 2012)*, pp. 203–207 (2012).
[5] 檀上未来，乃村能成，谷口秀夫：ソーシャルコーディングにおける有益な提案の抽出について，情報処理学会研究報告，Vol. 2013-DPS-157, No. 4, pp. 1–8 (2013).
[6] Rahman, M. M. and Roy, C. K.: An insight into the pull request of GitHub, *ACM*, pp. 372–381 (2014).
[7] Hansson, D. H.: Contributing to Ruby on Rails — Ruby on Rails Guides, Ruby on Rails Developer Team（オンライン），入手先 ⟨http://guides.rubyonrails.org/contributing_to_ruby_on_rails.html⟩（参照 2016-04-19）.
[8] Lewis, C., Lin, Z., Sadowski, C., Zhu, X., Ou, R. and Whitehead Jr., E. J.: Does Bug Prediction Support Human Developers? Findings From a Google Case Study., *Proc. ICSE '13*, pp. 372–381 (2013).
[9] Marlow, J. and Dabbish, L.: Activity Traces and Signals in Software Developer Recruitment and Hiring, *ACM*, pp. 145–156 (2013).