トピックモデリングに基づく開発者検索手法の構築へ向けて

福井 克法
和歌山大学 システム工学部
fukui.katsunori@g.wakayama-u.jp

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

川辺 義勝
株式会社 SRA
kawabe@sra.co.jp

要旨

効率的なソフトウェア開発には適切な人材配置が重要となるため，ソフトウェア開発組織は各開発者がこれまで携わった業務に関する経験をできる限り詳細に把握し管理できることが望ましい．しかしながら，組織の規模がある程度大きくなると在籍するすべての開発者の業務経験を人為的に管理することは現実的に困難な問題となる．本研究の目的は，ソフトウェア開発の全工程を対象として開発者を広く検索できる手法を構築し効率的な人材配置を支援することである．特に本論文では，メーリングリストアーカイブへ潜在的ディリクレ配分法 *(LDA: Latent Dirichlet Allocation)* を適用した結果に基づいて，検索語と開発者の関係性を定量的に計測することにより開発者を検索する手法を提案する．*4* つの *OSS* プロジェクトのメーリングリストアーカイブを対象として提案手法の評価実験を行った結果，プロジェクトによってばらつきが見られるものの，提案手法により *61%*〜*100%*の精度で開発者を検索できることが分かった．

1. はじめに

効率的なソフトウェア開発には適切な人材配置が重要となるため，ソフトウェア開発組織は各開発者[1]がこれまで携わった業務に関する経験をできる限り詳細に把握し管理できることが望ましい．しかしながら，組織の規模がある程度大きくなると在籍するすべての開発者の業務経験を人為的に管理することは現実的に困難な問題となる．そのため，構成管理システムに記録された開発者の過去の活動履歴に基づいて開発者を検索する手法が研究されている．例えば，Mockus らは，ソースファイルの変更履歴に基づいてソフトウェアシステムのモジュール毎に開発者の専門性を計測する手法を提案しており，問い合わせ先として適任の開発者を検索できるよう支援している [1]．同様に，Robbes らは，ソースファイルの変更履歴に加えて変更時期（直近の変更であれば専門性を高くする）を考慮した開発者検索手法を提案している [2]．

先行研究は，多数の開発者が在籍する大規模組織における開発者検索手法としてはある程度の有用性を期待できるものの，構成管理システムに記録される活動と紐付く開発者，すなわち，製造工程や保守工程において構成管理の対象となるファイルを変更したことのある開発者のみしか検索の対象にできないという点で適用範囲に限界がある．実際の開発組織においては，要件定義や基本・詳細設計などの上流工程を主に担当する場合もあり，業務の中でソースファイルの変更に直接的に関与しない開発者も少なくない．したがって先行研究は，大規模な開発組織内に在籍する開発者を広く検索する手法としては実用性の観点で問題があると言える．

本研究の目的は，ソフトウェア開発の全工程を対象として開発者を広く検索できる手法を構築し効率的な人材配置を支援することを最終的な目的とする．特に本論文では，メーリングリストアーカイブへ潜在的ディリクレ配分法 (LDA: Latent Dirichlet Allocation) を適用した結果に基づいて，検索語と開発者の関係性を定量的に計測することにより開発者を検索する手法を提案する．

本論文の構成は以下の通りである．続く 2 章では，提案手法の詳細について記述する．3 章では，提案手法を評価するための実験について述べる．4 章で評価実験の結果を示し，5 章において結果を考察する．6 章で関連研究を紹介し，最後に 7 章において，まとめと今後の課題について述べ本論文を結ぶ．

[1]本研究における開発者とは，ソフトウェア開発工程に関与するすべての実務者を指す．

2. 提案手法

2.1. 概要

本研究の目的は，ソフトウェア開発の全工程を対象として開発者を広く検索できる手法を構築し効率的な人材配置を支援することである．特に本論文では，メーリングリストアーカイブへ潜在的ディリクレ配分法 (LDA: Latent Dirichlet Allocation) を適用した結果に基づいて，検索語と開発者の関係性を定量的に計測することにより開発者を検索する手法を提案する．メーリングリストアーカイブを用いる理由は，開発工程の違いや部門の違いを問わずメーリングリスト[2]が開発組織において現状最も広く利用されているコミュニケーションツールであるため，開発者を広く検索するためのデータソースとして適していると考えたためである．

メーリングリストアーカイブを用いた最も単純な開発者検索手法には，メーリングリスト内で各開発者が使用した単語の回数に基づいて検索語毎に開発者を順位付けする方法が考えられる．しかしながら，このような単純な方法では，開発者が特定の投稿で多数同じ単語を使用した場合や，様々な分脈や話題で広く利用される一般的な単語を使用した場合には，開発者の専門性をその単語から想起することが困難になる．検索語となる単語がメーリングリスト内のどのような議論で特徴的に利用されてきたのかについての意味的な分析を踏まえて，検索語と開発者の関連性が定量化される必要がある．そこで本研究では，メーリングリストに存在するトピック（潜在的な意味）を把握する手段として潜在的ディリクレ配分法（LDA : Latent Dirichlet Allocation）[3] を用いることとした．

提案手法は主に，(1) メーリングリストアーカイブに LDA を適用しトピックを抽出する処理，(2) メーリングリストへの各投稿と検索語との関係性を定量化するための処理，(3) 検索語との関係性が高い投稿を行った開発者を順位付けするための処理から成る．以降では，それぞれの処理を詳しく説明する．

2.2. LDA によるトピック抽出

本研究では，メーリングリスト全体に含まれるトピックと各投稿を構成するトピックを把握するために LDA を用いる．図 1 は，メーリングリストアーカイブへ LDA を適用する一連の流れを示したものである．本節では，LDA のアルゴリズムを簡潔に説明するとともに，メーリングリストアーカイブを LDA に適用する際の前処理について説明する．

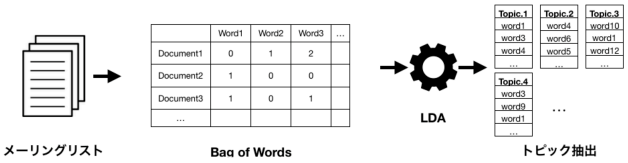

図 1. メーリングリストアーカイブへの LDA の適用

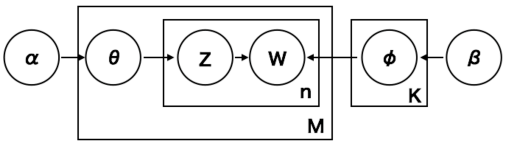

図 2. LDA のグラフィカルモデル

2.2.1. 潜在的ディリクレ配分法（LDA ： Latent Dirichlet Allocation）

LDA は文章の生成過程を確率的にモデル化するトピックモデルの 1 つである．トピックモデルとは，文章集合に含まれる潜在的なトピック（潜在的な意味）を推定するモデルである．LDA を用いることで，文章集合に含まれる複数のトピックを推定できるだけではなく，各トピックを構成する単語のトピック内での生起確率である**単語分布**や，1 つの文書に含まれるトピックの構成割合である**トピック分布**を計算することができる．

図 2 に，潜在的ディリクレ配分法のグラフィカルモデルを示す．LDA では 1 つの文書は複数のトピックから構成されており，各トピックで生起する単語の生起確率に応じて文書が生成されるという考えに基づいたモデルであるため，各文書はトピックの構成比であるトピック分布 θ を持っており，θ に応じて生起する単語のトピック Z が決定する．トピック Z の単語分布より生起する単語 W が決定する．n は 1 つの文書上の単語の出現回数を表し，M は文書数，K はトピック数を表す．α は θ を決定するための，β は ϕ を決定するためのハイパーパラメータである．

[2]個人間のやり取りを含むメールは本研究では対象外とする．

2.2.2. メーリングリストアーカイブへの LDA の適用

メーリングリストアーカイブに LDA を適用するためには，メーリングリストアーカイブから抽出したテキストデータ（本文）を数値で扱う必要がある．本研究では，図 1 左部分に示したように，Bag-of-Words を用いてテキストデータをベクトル化する．具体的には，メーリングリストへの各投稿に含まれるテキストデータ（本文）を Document1，Document2 のように行として，投稿中に出現する単語を Word1，Word2 のように列とし，各投稿に含まれる単語の頻度を行列で表したものが本研究で扱う Bag-of-Words ベクトルである．本研究では，30% 以上の文書に出現する一般的な単語（頻繁に使われる特徴的ではない単語）を除外してから Bag-of-Words ベクトルを作成している．また，文書全体で 2 回以下しか出現しないような単語も専門性を表す際のノイズとなる可能性が高いためあらかじめ除去した．

2.3. 検索語と各投稿との関係性の定量化

本研究では，開発者の人材配置を意図して大規模な開発組織に所属する開発者を検索する際には，開発者の業務経験や技術スキルを表す専門的な用語が検索語として用いられると想定している．本研究ではまず，LDA を適用した際に算出されるトピック分布 θ および単語分布 ϕ を用いて検索語と各投稿の関係を定量化し，2.4 節で述べる開発者のスコアリングのための前処理を行う．

まず，LDA の適用により抽出されるトピックから検索語と関連性の高い上位 5 つのトピックを抽出する．図 3 に示すように，具体的には各トピックの単語分布から検索語の生起確率が高い順に 5 つのトピックを抽出する．この処理によって，検索語とは一致しなくても検索語を含むトピックに含まれる単語を用いた開発者を検索する手がかりとなる．すなわち，検索語を含むトピックを何らかの専門性を表す文脈であると仮定し，その文脈に多く現れる開発者を検索語に関連性の深いエキスパートと考え検索結果の上位に位置付けるための前処理である．

次に，検索語と関連性の高い上位 5 つのトピックが各投稿にどの程度含まれているかをトピック分布（図 4）から算出し，検索語と各投稿との関係性を $DocScore_{w_i,d_j}$ として定量化する．

まず，各検索語 w_i に対して生起確率の高い上位 5 つのトピック T_{w_i} として特定する．

図 3. トピック特定

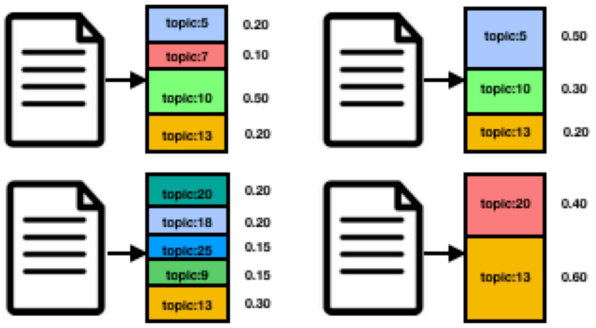

図 4. トピック分布の抽出

$$T_{w_i} = (t_1, t_2, t_3, t_4, t_5) \tag{1}$$

これらの 5 つのトピックを利用して式（2）よりに各投稿ごとに $DocScore_{w_i,d_j}$ を算出する．

$$DocScore_{w_i,d_j} = \sum_{k=1}^{5} \theta_{t_k,w_i} \times \phi_{t_k,d_j} \tag{2}$$

ここで，θ_{t_k,w_i} は検索語 w_i のトピック t_k における検索語の生起確率を表し，ϕ_{t_k,d_j} は投稿 d_j のトピック t_k のトピック分布の比率を表す．θ_{t_k,w_i} は，検索語 w_i の生起確率が高いトピックほど検索語 w_i に関する内容を含むトピックであると考え $DocScore_{w_i,d_j}$ の値が高くなる．また，ϕ_{t_k,d_j} についても，検索語と関連性の高いトピック t_k の比率が高いほどトピック t_k の内容を投稿 d_j が多く含むと考え $DocScore_{w_i,d_j}$ の値が高くなる．

2.4. 開発者の順位付け

次に，2.3 節で求めた $DocScore_{w_i,d_j}$ を利用して検索語と開発者の関係性を $AuthorScore_{dev_n}$ として算出し

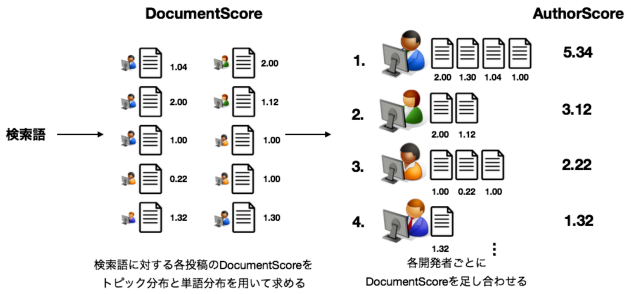

図 5. 開発者の順位付け

開発者を順位付けする（図 5）.

$$AuthorScore_{dev_n} = \sum_{m=1}^{NumOfMail} DocScore_{w_i,d_m} \quad (3)$$

$NumOfMail$ は各開発者が投稿した件数を表しており，$AuthorScore_{dev_n}$ は各開発者の各投稿の $DocScore$ を足し合わせることで算出する．$AuthorScore$ を開発者間で比較して，$AuthorScore_{dev_n}$ が高い開発者ほど検索語に関連したトピックを含む投稿を多く行っていると考えられる，すなわち，検索語と関連性の高い開発者と考えられるため，$AuthorScore$ の高い順に検索結果として出力する．

3. 評価実験

本章では，提案手法を評価することを目的に行った評価実験について述べる．

3.1. 目的

本実験の目的は，検索語に対してふさわしい開発者を本手法により検索できているかを確かめることである．実験の対象として複数のプロジェクトが混在しており様々な工程の開発者が使用している開発組織のメーリングリストアーカイブを用いることが望ましい．しかし，企業などの開発組織から大規模なデータを入手することが困難なため，本実験では 4 つの OSS プロジェクトのメーリングリストアーカイブを合成し，複数プロジェクトが混在したような合成データを作成する．各プロジェクトのドメイン用語を検索語とした時，対象としたドメイン用語を用いるプロジェクトに所属する開発者を検索できるかどうか調べることで提案手法を評価する．

表 1. 評価実験に使用するデータセット

プロジェクト	期間	件数	開発者数
Apache Hive	2008/11/11~2010/10/07	1,881	28
Apache Pig	2007/10/31~2010/09/28	1,662	44
Apache ZooKeeper	2007/10/31~2010/09/28	1,506	23
Apache Chukwa	2009/03/10~2013/10/08	877	25
合計		4,209	118

3.2. 対象データセット

本実験では，Apache Hadoop プロジェクトのサブプロジェクトである Apache Chukwa，Apache ZooKeeper，Apache Hive，Apache Pig のメーリングリストアーカイブを評価対象とする．本手法は大規模組織での人材配置に役立てることをの最終的な目標としているため，1 つの組織を再現するためにプロジェクトの内容が離れすぎていないプロジェクトを選択した．また，正解集合を作成しやすいように，Hadoop のサブプロジェクトであるがサブプロジェクト間で開発者の重複が少なく，データが取得しやすいという理由で上記 4 プロジェクトを選択した．

3.3. データセットの作成

実験のデータセットとして使用する対象プロジェクトの開発者向けメーリングリストのメールデータを mbox 形式で取得する．取得した mbox ファイルは月ごとにすべての投稿が 1 つのファイルに連結して保存されている．mbox ファイルから実験に必要なデータ（送信者情報，メッセージ ID，本文など）を取得することができる．ただし，開発者メーリングリストには，開発者同士のやりとり以外にプロジェクトで利用されているツール (Bugzilla, JIRA, Jenkins 等) から通知される投稿も含まれてるため，それらの投稿はあらかじめ除去した．また，本文に含まれる URL や開発者の署名は LDA を利用してトピック抽出する際のノイズとなるため，正規表現を利用してできる限り除去した．

上記の処理を施した後の本評価実験で使用したデータセットの内訳を表 1 に示す．メーリングリストへの投稿は総計 4,209 件，開発者（投稿者）は延べ 118 人であった．この内，複数のプロジェクトに参加している開発者が 2 人存在する．取得したこれら 4 つのサブプロジェクトの mbox ファイルを 1 つの mbox ファイルにまとめることによって，複数プロジェクトが混在した合成データ

表 2. tf-idf 値の高い上位 10 単語

Hive		Pig		Zookeeper		Chukwa	
単語	tf-idf 値	単語	tf-idf 値	単語	tf-idf 値	単語	tf-idf 値
hiveopensourc	29494.33	physicallay	6189.56	nioservercnxn	964.15	adaptor	818.90
srcpart	7406.33	pig	5193.71	zookeep	906.55	chukwa	683.94
metaexcept	5130.52	mapreducelay	4413.45	nioservercxn	842.17	hicc	420.35
jobconf	4256.35	cocoon	2088.30	todd	810.10	demux	398.50
thrift	3680.84	gld	1757.78	clientcnxn	756.99	collector	296.34
srcbucket	3274.15	inactiveaccount	1718.07	socket	662.31	seqfilewrit	281.23
protect	2116.89	piggybank	1497.06	peer	543.22	agent	267.72
numreducetask	2010.47	eq	1425.57	antlib	504.34	row	249.19
processnam	1842.93	acct	1389.78	hunt	488.12	depart	243.32
libjar	1746.07	foreach	1334.48	elect	483.75	scienc	243.26

を作成する．

3.4. 評価実験用の検索語の抽出

評価実験において，提案手法が検索語にふわさしい開発者を提示することができるかどうかを調べるために，実験で用いる検索語をあらかじめ用意しておく必要がある．評価実験で用いる検索語は，各プロジェクトのドメインの特徴を表すような用語（ドメイン用語）が望ましい．本論文におけるドメイン用語とは，特定の開発のみで使用される用語，または，特定の開発を表すような用語を指す．

評価実験では，文書集合の単語の重要度を評価する手法である tf-idf を用いて各プロジェクトの特徴語を抽出し検索語を決定する．また，プロジェクト内で頻繁に利用されている語もプロジェクトの特徴を表しているのではないかと考え頻出語を抽出して検索語とする．頻出語は tf 値を用いて抽出することが一般的であるが，tf 値を用いて抽出すると tf-idf 値を用いて抽出した場合と重複することが多いため，本研究では df 値を用いて抽出した．以降では，評価実験用の検索語を抽出するために行った処理について説明する．

3.4.1. 汎用語の削除

まず，図 6 のように，すべてのプロジェクトで共通して利用される汎用語を削除する．汎用語は，各プロジェクトから特徴語と頻出語を抽出する際のノイズになると考えられる．例えば，投稿メッセージの冒頭でのあいさつ（Hi や Dear など）や，実験で用いたプロジェクトは hadoop のサブプロジェクトであるため hadoop などの用語も全てのサブプロジェクトのメーリングリストで出現する汎用語に含まれる．

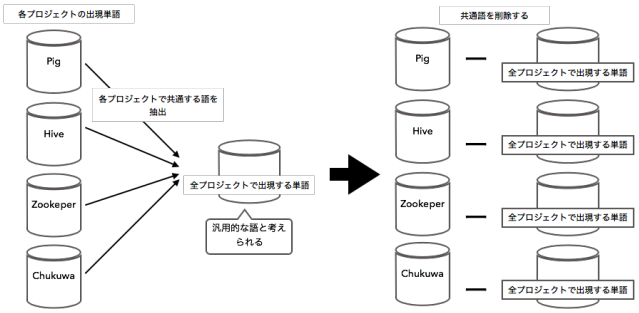

図 6. 汎用語の削除

3.4.2. 特徴語の抽出

文章中の単語の重要度を評価する手法の 1 つに tf-idf がある．tf-idf は文章 d における単語 t の出現頻度 $\mathrm{tf}(t,d)$(Term Frequency) と，単語 t が出現する文書数の全文書数に対する割合である $\mathrm{df}(t)$ の逆数の対数をとった逆文書頻度 $\mathrm{idf}(t)$(Inverse Document Frequency) に基づいて，式（7）で算出することができる．

$$\mathrm{tf}(t,d) = \frac{|\{t \in T_d\}|}{|T_d|} \tag{4}$$

$$\mathrm{df}(t) = \frac{|\{d \in D : t \in T_d\}|}{|D|} \tag{5}$$

$$\mathrm{idf}(t) = \log_2 \frac{1}{\mathrm{df}(t)} \tag{6}$$

$$\mathrm{tfidf}(t,d) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t) \tag{7}$$

ここで，T_d は文章 d で出現する単語の集合を，D は全文書の集合を表している．

本実験では各投稿 1 通を 1 つの文書として td-idf を適用し，各プロジェクト毎に出現単語の tf-idf 値を算出した．出現単語の tf-idf 値を目視で確認して，プロジェクトの特徴を表すと考えられる上位 10 単語を各プロジェクトの特徴語として抽出した．表 2 に抽出した各プロジェクト毎の特徴語（if-idf 値の高い単語）を降順に示す．

3.4.3. 頻出語の抽出

各プロジェクトの頻出語は式（5）の $df(t)$ の値の高い上位 10 単語を用いる．$tf(t,d)$ は単語の使用頻度である

表 3. df 値の高い上位 10 単語

Hive		Pig		Zookeeper		Chukwa	
単語	df 値	単語	df 値	単語	df 値	単語	df 値
hiveopensourc	7311.00	pig	18039.00	zookeep	2152.00	chukwa	1560.00
hive	4844.00	physicallay	1194.00	hunt	404.00	adaptor	347.00
srcpart	2041.00	mapreducelay	830.00	nioservercnxn	242.00	hicc	168.00
thrift	1357.00	schema	569.00	clientcnxn	239.00	demux	168.00
metaexcept	1231.00	foreach	539.00	todd	216.00	collector	123.00
jobconf	1102.00	gate	516.00	elect	201.00	scienc	115.00
srcbucket	836.00	cocoon	495.00	socket	198.00	depart	114.00
protect	590.00	piggybank	452.00	nioservercxn	193.00	jiaqi	113.00
numreducetask	456.00	gld	407.00	lock	165.00	agent	107.00
metastor	448.00	javacc	365.00	peer	148.00	seqfilewrit	79.00

ため特定の文書に偏って同じ単語が使用されていても高い値が示されるのに対して，$df(t)$ は単語 t が出現する文章が多いほど高い値を示すため，プロジェクト内で幅広く使われている単語も各プロジェクトの特徴を表すのではないかと考えた．表 3 に各プロジェクト毎の頻出語（df 値の高い単語）を降順に示す．

3.5. 実験手順

評価実験は手順で行った．

手順 1：LDA モデルの作成 3.3 節で述べた 4 つのプロジェクトを組み合わせた合成データに対して LDA を適用する．LDA ではトピック数を任意の数に設定できる．設定したトピック数が検索精度に与える影響を調べるため，本評価実験ではトピック数をそれぞれ 100，500，1000 の 3 つのモデルを作成した．各モデルに対して各投稿のトピック分布およびトピック毎の単語分布を計算する．

手順 2：特徴語・頻出語を用いた開発者の検索 3.4 節で述べた各プロジェクトの特徴語および頻出語を検索語として用いて提案手法を適用する．2.4 節で述べた AuthorScore の高い上位 10 名の開発者を検索結果とする．

手順 3：検索結果から正答数と正答率を算出 手順 2 の検索で得られた開発者がどの程度正しく検索できているか検証するために，検索して得られた開発者が特徴語および頻出語（検索語）が出現するプロジェクトに参加していれば正解，参加していなければ不正解としてトピック数とプロジェクト毎に正答数と正答率を算出する．

4. 実験結果

表 4 に，特徴語（tf-idf 値の高い上位 10 単語）を検索語とした場合の平均正答率をプロジェクト毎に示す．また，表 5 に，頻出語（df 値の高い上位 10 単語）を検索語とした場合の平均正答率をプロジェクト毎に示す．表 4 および表 5 ではそれぞれ，設定したトピック数別（100, 500, 1000）で平均正答率を示している．ここでの正答率とは，表 2 および表 3 の検索語を用いて検索を行なった結果得られる上位 10 名の開発者が，検索語が属するサブプロジェクトに何名参加していたかを表す．例えば，hiveopensource を検索語とした場合に Apache Hive に参加する開発者が 10 名中何名検索結果として提示されていたかで正答率を求める．したがって，ここでの平均正答率とは，検索語 10 単語を用いて検索を行った場合の正答率の平均値を表す．

表 4. 特徴語（tf-idf 値の高い上位 10 単語）を検索語とした場合のプロジェクト毎の平均正答率

プロジェクト	トピック数		
	100	500	1000
Apache Hive	81%	61%	62%
Apache Pig	91%	100%	99%
Apache ZooKeeper	61%	70%	73%
Apache Chukwa	71%	67%	78%

表 5. 頻出語（df 値の高い上位 10 単語）を検索語とした場合のプロジェクト毎の平均正答率

プロジェクト	トピック数		
	100	500	1000
Apache Hive	78%	65%	66%
Apache Pig	86%	99%	91%
Apache ZooKeeper	66%	75%	75%
Apache Chukwa	71%	69%	82%

以下ではサブプロジェクト毎に検索精度を議論する．

4.1. Apache Hive プロジェクトの実験結果

特徴語を用いた検索結果

すべての検索語の正答率の平均では，トピック数を 100 に設定した時に正答率が 81%で最も高い値を示し，トピック数を 500 に設定した時の正答率が 61%で最も低い値を示した．トピック数を 1000 に設定した時の正答率は 62%であった．

頻出語を用いた検索結果

すべての検索語の正答率の平均では，トピック数を 100 に設定した時に正答率が 78%で最も高い値を示し，トピック数を 1000 に設定した時の正答率が 65%で最も低い値を示した．トピック数を 500 に設定した時の正答率は 66%を示した．

4.2. Apache Pig プロジェクトの実験結果

特徴語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 500 に設定した時の正答率が 100%で最も高い値を示し，トピック数を 100 に設定した時の正答率が 91%で最も低い値を示した．トピック数を 1000 に設定した時の正答率は 99%であった．

頻出語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 500 に設定した時の正答率が 99%で最も高い値を示し，トピック数を 100 に設定した時の正答率が 86%で最も低い値を示した．トピック数を 1000 に設定した時の正答率は 91%であった．

4.3. Apache ZooKeeper プロジェクトの実験結果

特徴語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 1000 に設定した時の正答率が 73%で最も高い値を示し，トピック数を 100 に設定した時の正答率が 61%で最も低い値を示した．トピック数を 500 に設定した時の正答率は 70%であった．

頻出語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 1000 と 500 に設定した時の正答率が 75%で最も高い値を示し，トピック数を 100 に設定した時の正答率が 66%で最も低い値を示した．

4.4. Apache Chukwa プロジェクトの実験結果

特徴語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 1000 に設定した時の正答率が 78%で最も高い値を示し，トピック数を 500 に設定した時の正答率が 67%で最も低い値を示した．トピック数を 100 に設定した時の正答率は 69%であった．

頻出語を用いた検索結果

すべての検索語の正答率の平均は，トピック数を 1000 に設定した時の正答率が 82%で最も高い値を示し，トピック数 500 に設定した時の正答率が 69%で最も低い値を示した．トピック数 500 に設定した時の正答率は 71%であった．

4.5. 実験結果のまとめ

各プロジェクトとトピック数の設定によって正答率に差はあったが，正答率は 61%〜100%であった．Apache Pig の特徴語を用いてトピック数を 500 に設定した時の正答率が 100%で最も高く，Apache Hive の特徴語を用いてトピック数を 500 に設定した場合の正答率が 61%で最も低かった．Apache Hive ではトピック数を 100 に設定した時，Apache Pig ではトピック数を 500 に設定した時，Apache ZooKeeper と Apache Chukwa ではトピック数を 1000 に設定した時が最も正答率が高かった．

5. 考察

本章では，評価実験の結果を踏まえ，提案手法の有用性と今後の課題について考察する．

5.1. 正答率と提案手法の有用性

実験の結果，Apache Pig の検索語を用いて実験を行った場合が最も正答率が高い結果となった．表 1 で示したように Apache Pig の開発者数は 44 人と最も多く，他の 3 サブプロジェクトは多くとも 28 人である．これらの結果を鑑みると，多くの開発者が議論に参加する活発なメーリングリストであったため，Apache Pig の開発者の経験やスキル，役割などに沿って LDA によるトピック抽出が他のサブプロジェクトに比べて上手く行えたのではないかと考える．結果として，検索語から得られる上位 10 人の開発者が，検索語毎に比較的分散して出力された結果，他のサブプロジェクトよりも高い平均正答率を示したのではないかと考えている．

一方で，特徴語を検索語とした場合の Apahce Zookeeper は，61%（トピック数）と最も平均正答率が低かった．トピック数を増やすことで平均正答率が 70%

表 6. 正答率が半分以下の検索語

検索語						
thrift	libjar	javacc	socket	antlib	row	scienc
hicc	peer	numreducetask	srcbucket	seqfilewrit	jobconf	

∼ 73%に向上するものの，平均正答率には改善の余地があると言える．また，Apache Hive のようにトピック数を増やすと平均正答率が低下するプロジェクトも存在するため，トピック数を増やすことが平均正答率の向上につながるとも一概には言えない．Apahce Zookeeper の開発者は 4 つのサブプロジェクトの中では最も少ない 23 人であり，メーリングリストへの参加者数が正答率に影響を与えている可能性も否定できない．これらのことから，プロジェクトの開発者数，投稿数，トピック数からは平均正答率との一貫した因果関係を読み取ることはできないため，検索精度の向上へ向けて今後は各開発者の投稿内容をより詳細に分析する必要がある．

評価実験の結果全体としては，すべてのプロジェクトにおいても平均正答率は 6 割以上を示しており提案手法は一定の有用性を示したと考えている．

5.2. 検索語と正答率

実験の結果，検索語によって正答率の違いがあることが分かった．各プロジェクトで使用した検索語のうちトピック数 100，500，1000 のいずれかに設定した際に正答率が半分以下であった検索語を表 6 に示す．

正答率が半分以下であった検索語のうち thrift, libjar, javacc, socket, antlib, row, science, peer については，ソフトウェア開発に関するプロジェクトでは頻繁に利用される単語であるため，各プロジェクトの正答率が半分以上であった検索語と比較すると各プロジェクトの特徴を表すことができず正答率が低かったと考えられる．

thrift は，Apache Thrift プロジェクトのことを指していると考えられる．Apache Thrift は RPC（リモートプロシジャーリコール）フレームワークのプロジェクトである．RPC はネットワークに接続された他の PC 上でプログラムを呼び出すためのプロトコルである．

libjar は，Java 言語において外部ライブラリをインポートする際に lib フォルダに jar ファイルを置くため，外部ライブリのインポートの一連の流れに関する内容がデータに含まれていたため出現した単語であると考えられる．

javacc は，Java Compiler Compiler の略であり Java 言語向けの構文解析器生成ツールである．

socket は，ネットワーク用語の一つであり通信プロトコルを利用する際に通信の出入り口を Socket のことを指していると考えられる．このようにこれらの単語は必ずしも各サブプロジェクトの開発に特化した内容で用いられる用語ではなく，サブプロジェクト内で比較的頻繁に利用される語であった．

さらに，numreducetask の正答率が低かった原因として，複数のプロジェクトで使用されていたため各プロジェクトの特徴を表すことができなかったと考えられる．chukwa プロジェクトでは tf-idf 値が 13.55，pig プロジェクトでは tf-idf 値が 52.56 で出現していた．3.4.1 節で述べたように，本研究では汎用語をあらかじめ除去して合成データを作成しているが，本評価実験では 4 つのプロジェクトで共通して出現する単語のみを除去している．2 つあるいは 3 つのプロジェクトで共通して出現する単語も汎用語として除去することで平均正答率の向上につながる可能性がある．しかしながら，汎用語として多くの用語を除去すれば，該当するトピック・開発者が紐付きにくくなる可能性もあるため，汎用語の削除の効用については実験条件を追加し比較する必要がある．

また，正答率が半分以下であった残りの hicc，srcbucket，seqfilewrit については，検索語として抽出したプロジェクトでのみ使用される語であったが正答率は半分以下であり，正答率の低さを説明することが現時点では困難なため，今後はより詳細に各検索語の利用される文脈を投稿内容から理解する必要がある．

5.3. トピック数

実験の結果から，トピック数の変化による正答率の変化はプロジェクトによって異なることが分かった今回の実験では，トピック数による正答率の大きな違いは見られなかったが正答率を上げるためにはデータに対して最適なトピック数を設定する必要があると考えられる．本

表 7. Apache Hive の特徴語：srcbucket

topic:100：正答率 (80%)					topic:500：正答率 (50%)				
#74	#97	#88	#75	#9	#225	#143	#2	#210	#244
ok	ok	org	junit	junit	timefram	junit	junit	org	junit
junit	org	type	info	org	timefram	tabl	data	type	info
exec	junit	hive	exec	info	maintain	ok	tabl	ok	org
tabl	data	problem	hiveopensourc	hive	hdf	data	ok	partit	data
info	tabl	java	usr	tabl	hive	partit	info	srcpart	ok
data	java	tabl	file	exec	zookeep	srcpart	srcpart	tabl	tabl
org	hive	data	org	ok	think	hr	partit	java	srcpart
partit	queri	class	build	file	client	usr	org	class	usr
hr	partit	refer	job	hiveopensourc	project	hiveopensourc	hr	hr	partit
build	srcpart	ok	data	usr	award	queri	exec	hive	hr

節では，特定のプロジェクトでのみ使用されているが，正答率が半分以下を示した hicc について設定したトピックのうち正答率が最も高かったトピック数と低かったトピック数で検索語が含まれるトピックとして提案手法により特定できた上位 5 つの構成語を比較することで，正答率とトピック数の違いを考察する．表 7 は，検索語として srcbucket を用いた時に特定できたトピックの主要な構成語を表している．正答率の最も高かったトピック数 100 の時と，低かったトピック数 500 の時に特定できたトピックの構成語を比較すると 3.4.3 節で特定できた Apache Hive の特徴語と頻出語がトピックの構成語として含まれている数は一緒であったが，正答率が高かったトピック数 100 の時の方が，hiveopensourc などの tf-idf 値がより高い語が多く含まれていた．正答率の高かったトピック数と低かったトピック数で特定できたトピックの構成語を確認することで正答率の高かったトピック数の方が，各プロジェクトの特徴語と頻出語が多く含まれることや，より tf-idf 値が高い語が含まれる場合が多いことが分かった．そのため，正答率が高い場合は正答率が低い場合よりも特定したトピックの内容が各プロジェクトに関連するトピックであったため正答率が高くなったと考えられる．

5.4. ソフトウェア開発組織への応用へ向けて

本実験では，OSS プロジェクトのメーリングリストを用いることで評価実験を行なった．しかし，OSS プロジェクトと企業などのソフトウェア開発組織ではコミュニケーションの取り方が異なると考えられる．OSS プロジェクトでは，メールでのコミュニケーションが中心であり議論の内容等がメールアーカイブに残りやすいが，ソフトウェア開発組織では会議や会話でのコミュニケーションが中心となるためメールアーカイブに開発に関する議論などが残りにくい．そのため，本手法をソフトウェア開発組織に応用するにあたり，メールアーカイブのみでなく会議の議事録や仕様書を利用することが有効であると考える．また，本実験では tf-idf 値と df 値を用いることで機械的に各プロジェクトの開発者を検索するための語を決定したが，ソフトウェア開発組織において利用する際の検索語は異なると考えられる．

6. 関連研究

本研究で提案したような開発者検索手法および推薦手法は広く研究されている．本節では，本研究に関する関連研究としてコードレビューや不具合修正などに着目した開発者推薦に関する研究を紹介する．

Rahman ら [4] は，開発者推薦において特にコードレビューを行うレビューアーについて着目した．コードレビューはテストやコーディングの初期段階で行われ，あらかじめバグとなる箇所を適切な開発者が目視でコードを確認しバグを発見することで開発の全体的なコストを削減することができることが知られている．レビュー待ちのプルリクエストに対して，過去に開発者がレビューしたプルリクエストとの類似度を測ることで推薦手法を提案している．

Alan ら [5] は，不具合修正を担当する開発者の推薦手法を提案した．開発者の推薦を行う際に，バージョン管理システムのデータと不具合管理システムのデータに着目した．双方のデータを用いた際の推薦手法の結果の比較を行い．双方のデータを用いた手法の利点について

述べた．より正確に目的の開発者を推薦したい場合は，バージョン管理システムをデータとして用い，正確さよりも関連する開発者を多く推薦したい場合は，不具合管理システムのデータを用いることが良いことを示した．

Bayati ら [6] は，特に情報セキュリティに関する分野に精通した開発者を推薦するための手法を提案した．ソフトウェアのセキュリティバグの深刻度について述べ，情報セキュリティに関する知識を持った開発者を推薦する手法を提案した．大規模 QA サイトである StackOverflow のデータから投稿に付与されたタグと，質問への回答数，信頼度スコアを利用することで開発者をスコアリングし開発者を推薦する手法を提案した．

これらの研究では，特定のタスク（レビュー，不具合修正など）における開発者の検索・推薦を支援することを目的としており，汎用性の高い開発者検索を目指す本研究とは立場が異なるが，検索結果の評価方法や検索精度の議論については本研究の参考になる．

7. まとめと今後の課題

本研究ではソフトウェア開発におけるコミュニケーションデータであるメーリングリストでの開発者のやりとりに着目し，大規模ソフトウェア開発組織における適切な人材配置を支援することを最終目的にして開発者検索手法を提案した．具体的には，メーリングリストに対してトピックモデルを適用し，メーリングリストからトピックと各トピックの単語分布，各投稿のトピック分布をそれぞれ算出し，検索語と各投稿との関係性の定量化するとともに検索語と開発者の順位付けして開発者を検索するための手法を提案した．

OSS プロジェクト 4 つのメーリングリストを用いて複数プロジェクトが混在した合成データを作成し，各プロジェクトの tf-idf 値の高い特徴語と df 値の高い頻出語を検索語として用いて評価実験を行なった．プロジェクトによって正答数にばらつきはあったが，すべてのプロジェクトで各プロジェクトの検索語を用いて検索を行なった結果，正答率は 61%〜100%であった．本研究における開発者の検索を行う際には，各プロジェクトでのみ使われるドメイン用語を検索語に用いると正答率が上がることが分かった．

今回はメーリングリストとメーリングリストに含まれる各投稿の内容把握のためにトピックモデルである LDA を用いた．しかし，LDA は事前にトピック数を設定する必要があり良い結果を得るためには最適なトピック数を推定する必要がある．今後は LDA 以外のアルゴリズムを用いてメーリングリストと各投稿から特徴量を抽出し，正答率を改善する予定である．

謝辞

本研究の成果は株式会社 SRA と国立大学法人和歌山大学との共同研究の一部である．また，本研究の一部は，文部科学省科学研究補助金（基盤 (A): 17H00731，基盤 (C): 18K11243）による助成を受けた．

参考文献

[1] Mockus, A. and Herbsleb, J. D.: Expertise Browser: A Quantitative Approach to Identifying Expertise, *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, pp. 503–512 (2002).

[2] Robbes, R. and Röthlisberger, D.: Using Developer Interaction Data to Compare Expertise Metrics, *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pp. 297–300 (2013).

[3] Blei, D. M., Ng, A. Y. and Jordan, M. I.: Latent Dirichlet Allocation, *J. Mach. Learn. Res.*, Vol. 3, pp. 993–1022 (2003).

[4] Rahman, M. M., Roy, C. K. and Collins, J. A.: CoRReCT: Code Reviewer Recommendation in GitHub Based on Cross-project and Technology Experience, *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pp. 222–231 (2016).

[5] Anvik, J. and Murphy, G. C.: Determining Implementation Expertise from Bug Reports, *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pp. 2– (2007).

[6] Bayati, S.: Security Expert Recommender in Software Engineering, *Proceedings of the 38th International Conference on Software Engineering Companion*, ICSE '16, pp. 719–721 (2016).