2 段階 LDA を用いたインクリメンタルなソフトウェアリポジトリの自動分類手法

井上 雅翔† 新井 啓介‡ 山名 早人§,¶

† 早稲田大学基幹理工学部 〒169-8555 東京都新宿区大久保 3-4-1
‡ 早稲田大学大学院基幹理工学研究科 〒169-8555 東京都新宿区大久保 3-4-1
§ 早稲田大学理工学術院 〒169-8555 東京都新宿区大久保 3-4-1
¶ 国立情報学研究所 〒101-8430 東京都千代田区一ツ橋 2-1-2
E-mail: {m.inoue, k-arai, yamana}@yama.info.waseda.ac.jp

あらまし 近年，ソフトウェアはその規模が膨大となっており，ソフトウェアの再利用がますます重要となってきている．ソフトウェアの再利用を効率的に行うためには，ソフトウェアのソースコード群であるプロジェクトを分類するソフトウェアリポジトリの構築が欠かせない．既存手法によってソフトウェアリポジトリを自動分類できるが，既存の分類結果に対するプロジェクト追加時の再分類に対応していない．そこで本論文では，2 段階 LDA を用いたソフトウェアリポジトリの自動分類手法を提案する．既存手法の問題に対し，提案手法では局所的な再分類を行うことができる．200 のプロジェクトについての実験の結果，既存の分類結果に対するプロジェクトの追加に対して，提案手法は既存手法の最大約 1213 倍の速度で再分類できた．

キーワード ソフトウェアリポジトリ，テキストマイニング，自動分類，再利用

An Automatic Clustering Method with 2 Steps LDA for Incremental Software Repositories

Masatobu INOUE† Keisuke ARAI‡ and Hayato YAMANA§,¶

† Faculty of Science and Engineering, Waseda University 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555 Japan
‡ Graduate School of Fundamental Science and Engineering, Waseda University 3–4–1 Okubo, Shinjuku-ku, Tokyo, 169–8555, Japan
§ Science and Engineering, Waseda University 3–4–1 Okubo, Shinjuku-ku, Tokyo, 169–8555, Japan
¶ National Institute of Informatics 2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, 101-8430, Japan
E-mail: {m.inoue, k-arai, yamana}@yama.info.waseda.ac.jp

1. はじめに

近年のソフトウェア制作規模の拡大による効率的なソフトウェア開発の要求に対し，ソフトウェアの再利用は重要な役割を果たしている．ソフトウェアの再利用ではソフトウェアライブラリを利用する方法が取られてきたが，ソフトウェアライブラリの数は少なく，再利用できるものが見つかりにくいという問題がある．そこで，既存のソースコード群の中から再利用に適したものを検索する方法が提案されている[1]．既存のソースコード群とは，開発者が過去に作成したソースコードや SourceForge[2]をはじめとするソフトウェアリポジトリに登録されているソースコードを指す．既存のソースコードの数はソフトウェアライブラリの数よりも多いため，再利用できるものがよりみつかりやすい．

ソースコードを再利用するには，開発するソフトウェアの要求仕様にあったソースコードを検索する必要がある．既存のソースコード群から目的の機能を持つものを検索するために，ソースコードを全文検索する方法が用いられる．しかし，全文検索では，目的とするソースコードを検索することは困難である．これは，ソースコード内に必ずしも当該コードの機能が記述されているわけではないためである．この問題に対し，ソフトウェアリポジトリ内でのカテゴリを用いた検索が有効である．ソフトウェアリポジトリ内でのカテゴリとは，プロジェクト単位での機能による分類である．プロジェクトとは，1 つのソフトウェアを構成する，ソースコードの集まりである．ソフトウェアリポジト

リ内でのカテゴリを利用する利点は以下の 2 つである．1 つ目に，ソフトウェアリポジトリは機能に基づいて分類しているため，機能に対する検索がしやすい．2 つ目に，1 つのソースコード単体では満たせず複数のソースコードによって満たされる機能を，プロジェクトを用いて検索することができる．

SourceForge[2]には，2010 年 12 月現在，約 27 万のプロジェクトが登録されており，そのプロジェクト数は膨大である．膨大な数のプロジェクトを人手によって分類することは困難であり，自動的に分類する手法が提案されている．分類にあたっては，各プロジェクトの機能を把握することが不可欠となる．しかし，ソースコードには機能を示す語が少ない．そこで，ソースコードの語群における潜在的な意味を解析し機能を把握する手法がよく用いられる．

Tian らの手法[4]では，Latent Dirichlet Allocation (LDA) [7]を用いてソフトウェアリポジトリを自動的に分類している．LDA により，語群における潜在的な話題を解析し，機能に基づいた自動分類を実現している．しかし，これらの既存の手法では，一旦できた分類に対して，プロジェクトを追加する場合，すべてのプロジェクトを再分類する必要がある．つまり，頻繁にプロジェクトが追加される，インクリメンタルなソフトウェアリポジトリにおいては，分類効率が悪い．

こうした問題に対し，本論文では，インクリメンタルなソフトウェアリポジトリに対する高速な再分類を実現する手法を提案する．具体的には，ソフトウェアリポジトリを分類するにあたって，Tian らの方法のように 1 回で分類するのではなく，2 段階に分けて分類し，可能な限り 1 段階目の分類が，後に追加されるプロジェクトによって影響を受けない手法を考える．第 1 段階では，分類要素をあらかじめ決められたカテゴリに分類し，一部の識別子のみを用いて高速に分類する．第 2 段階では，分類要素によって決められるカテゴリに分類し，すべての識別子を用いて詳細に分類する．識別子とはソースコード中の型名，関数名，変数名を指す．提案手法では，既存の分類結果に対しプロジェクトが追加されたとき，すべてのプロジェクトを分類する必要がない．

本稿の構成は以下のとおりである．2 節では，関連研究について述べる．3 節では提案手法について説明する．4 節では提案手法の評価実験について述べ，5 節ではまとめを述べる．

2. 関連研究

本節では，本手法に関係する研究を述べる．2.1 では Latent Semantic Analysis (LSA) を用いたソフトウェアリポジトリ分類の研究について述べる．LSA とは，LDA と同じく語群における潜在的な意味を解析する手法の 1 つである．2.2 では Probabilistic LSA (pLSA) を用いたソースコード群を分類する研究について述べる．pLSA も，同様に語群における潜在的な意味を解析する手法の 1 つである． 2.3 では LDA を用いたソフトウェアリポジトリ分類の研究について述べる．

2.1. LSA を用いたソフトウェアリポジトリ分類の研究

川口ら[3]はプロジェクトを LSA にて解析することで，ソフトウェアの機能に基づくソフトウェアリポジトリ分類を行った．

川口らの手法では，まず，プロジェクト中の識別子を抽出，解析することでプロジェクトを多次元配列にマッピングする．得られた多次元配列はソフトウェアの特徴ベクトルと呼ぶ．次に特徴ベクトルを LSA にて次元圧縮することで低次元の特徴ベクトルを取得する．最後に特徴ベクトルを用いることでソフトウェアリポジトリ分類を行う．川口らの手法は 1 つのプロジェクトが複数の分類に属することのできる分類を実現し，分類精度を示す F 値は 0.72 を得ている．

しかし，川口らの手法[3]が用いた LSA では次元圧縮において誤差が発生する問題がある．

2.2. pLSA を用いたソースコード群を分類する研究

Sandhu ら[8]は，ソースコード群を pLSA を用いて解析することで，ソースコード群の分類を行った．

Sandhu らの手法は，まず，ソースコード中の識別子を抽出する．次に，抽出した識別子を pLSA 用いて解析することで，ソースコードを多次元配列にマッピングする．得られた多次元配列であるソースコードの特徴ベクトルからソースコードの分類を行う．

Sandhu らはソースコードの分類を，LSA を用いた分類結果と比較することで，pLSA の方がよい分類精度となることを示した．これは，LSA の次元圧縮による誤差が，pLSA で発生しないためである．しかし，pLSA の計算コストの高さを問題点としてあげている．

2.3. LDA を用いたソフトウェアリポジトリ分類の研究

Tian ら[4]はプロジェクトを LDA にて解析することでソフトウェアの機能に基づく分類を行った．

Tian らの手法は，まず，プロジェクト中の識別子を抽出，LDA を用いて解析することでプロジェクトを多次元配列にマッピングする．得られた多次元配列であるプロジェクトの特徴ベクトルを用いることでソフトウェアリポジトリ分類を行う．

同手法で用いた LDA は，LSA のように次元圧縮による誤差が発生しない． よって，Tian らは川口ら[3]

よりも精度の高い分類を行うことができる．さらに，LDA では pLSA よりも計算コストが低いため，より高速な分類ができる．

3. 提案手法

本節では，本項が提案するソフトウェアリポジトリの分類手法について述べる．具体的には，提案手法の特徴について述べた後，各特徴について詳細に説明をしていく．

3.1. 提案手法の特徴

本節では，提案手法の特徴について述べていく．提案手法の特徴は以下の 2 つである．

1. トピックモデルである LDA を用いたこと
2. LDA を 2 段階で用いたこと

提案手法の 1 つ目の特徴として，2 節で述べた関連研究の中で最も精度の高く計算コストの低い LDA を用いている．しかし，LDA を用いたソフトウェアリポジトリの分類には問題がある．それは，従来の分類方法は，事前に分類対象となるすべてのプロジェクトが存在することを前提としている問題である．つまり，頻繁にプロジェクトが追加される，インクリメンタルなソフトウェアリポジトリに対しては，追加の度に全プロジェクトを再分類しなければならない．

この問題に対応するために，提案手法の 2 つ目の特徴として示したように，LDA を 2 段階で適用することを，新しく提案する．LDA を 2 段階に適用する目的は，既存手法である Tian らの手法[4]と同等の分類精度を保ちながらインクリメンタルなデータに対する再分類速度を向上させることである．

3.2. LDA によるソフトウェアリポジトリ分類手法

本節では，提案手法が用いている，Tian ら[4]の提案する LDA によるソフトウェアリポジトリ分類手法について述べる．分類は，図 1 のフローチャートの手順に沿って行われる．分類手法について，以下で詳しく述べてく．

3.2.1. LDA の概要

LDA によるソフトウェアリポジトリ分類手法について述べる前に，LDA の概要について述べる．LDA とはトピックモデルの 1 つである．以下において，トピックモデルと，LDA について述べた後，提案手法で用いる意味について述べる．

トピックモデル

トピックモデルを用いることで，語の集合ではなく，話題と呼ばれる文書の潜在的な意味によって，文書を表現できる．LDA による文書の表現は，文書集合を行列$\mathbf{Z} = (z_{ij})_{i,j}$に変換することで行う．行列 **Z** の要素$z_{ij}$は，文書 *i* における話題 *j* の存在する確率を表す．

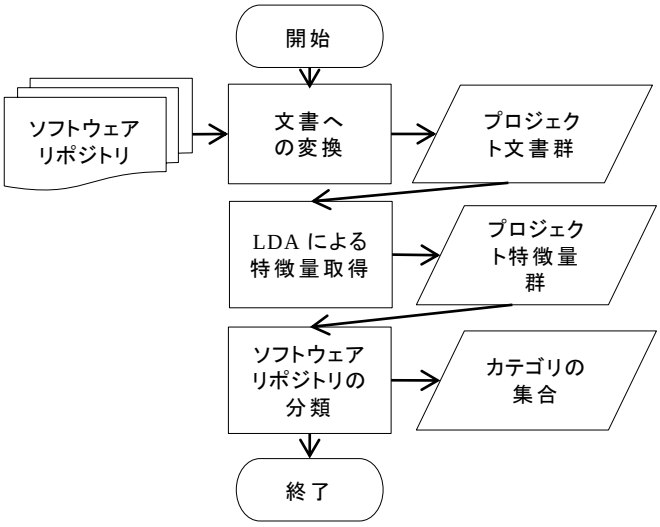

図 1 LDA によるソフトウェアリポジトリ分類手法のフローチャート

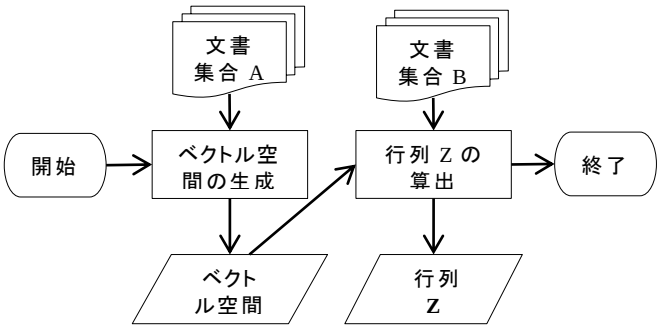

図 2 LDA のフローチャート

LDA

まず，LDA が文書集合を行列 **Z** に変換する流れを，図 2 のフローチャートに示す．次に，LDA における，行列 **Z** の生成方法について述べる．行列 **Z** の生成は，図 2 の「ベクトル空間の生成」の処理において，文書集合 A の特徴に基づいて生成されたベクトル空間に，文書集合をマッピングすることで行われる．このとき，文書集合 A，B の選び方には以下の 2 つがあり，各々で行列 **Z** の意味が異なる．

1. 文書集合 A，B を同じ文書集合にすることで，モデル化する文書集合の特徴を，最も顕著に表す行列 **Z** を生成する．
2. 文書集合 A，B を異なる文書集合にすることで，特定の文書集合 A の特徴に基づく，行列 **Z** を生成する．

LDA を用いる意味

提案手法では，ソフトウェアリポジトリ中のプロジェクトを文書に変換した上で，LDA を適用する．LDA で得られる行列 **Z** で表現された文書の話題とは，文書の潜在的な意味であった．プロジェクトを変換した文

書において，プロジェクトが構成するソフトウェアの潜在的な意味とは，ソフトウェアの機能であるといえる．よって，プロジェクトを文書に変換することで，ソフトウェアの機能を LDA で得られる行列 **Z** で表現できる．

3.2.2. プロジェクトの文書化

図 1 における，「文書への変換」の処理について述べる．この処理では，ソフトウェアリポジトリを LDA に適用させるために，ソフトウェアリポジトリ中のプロジェクトを文書に変換する．プロジェクトの文書への変換は，プロジェクト中のソースコードに現れる識別子を抽出し羅列していくことで行う．ただし，識別子は一般的な語に分解して文書に変換する．例えば，“getName” という名前の関数名があった場合には，“get” と “name” の 2 つの語に分解し文書に変換する．どの識別子を用いるかについては 3.3 にて述べる．

3.2.3. LDA によるプロジェクトの特徴量取得

次に，図 1 における，「特徴量取得」の処理について述べる．この処理では，文書に変換したプロジェクトを集め，LDA を適用し，3.2.1 で述べた行列 **Z** を取得する．本手法では，Collapsed Gibbs Sampling に基づいた LDA である JGibbLDA[9]を用いる．得られた行列 **Z** は各プロジェクトの構成するソフトウェアの機能を表現し，行列 **Z** の各行ベクトルは各プロジェクトを表している．よって，行列 **Z** の各行ベクトルは，各プロジェクトが構成するソフトウェアの機能に基づく特徴量となる．LDA において，3.2.1 で述べた文書集合 A, B の選び方については 3.3 にて説明する．

3.2.4. ソフトウェアリポジトリの分類

次に，図 1 における，「ソフトウェアリポジトリの分類」の処理について述べる．ソフトウェアリポジトリ中のプロジェクトの分類は，3.2.3 によって得られたプロジェクトの特徴量に基づいて決定される．具体的には，各プロジェクトを示す行列 **Z** の行ベクトルについては，閾値より大きい列の値を探し，この列に対応するカテゴリにプロジェクトを所属させる．例えば，あるプロジェクトの特徴量の中でも 1,4,5 次元の値が閾値より大きければ，1,4,5 番目のカテゴリに分類する．本手法では，1 つのプロジェクトは複数のカテゴリに所属できる．また，LDA を用いたプロジェクトの特徴量は，機能に基づくもとのとなるため，機能に基づくカテゴリに分類することができる．

3.3. 2 段階分類手法

インクリメンタルなソフトウェアリポジトリに対応するために，提案手法では図 3 に示されるフローチャートに従い，2 段階による分類を行う．

分類は，ソフトウェアリポジトリを図 3 の「分類第 1 段階」の処理にて分類し，作られた各分類のプロジェクト群を，図 3 の「分類第 2 段階」の処理にて再度分類する．図 3 の「分類第 2 段階」の処理で生成した各カテゴリを併合し，提案手法が作る最終的なカテゴリ群を生成する．なお，第 1 段階では，分類要素をあらかじめ決められたカテゴリに分類し，分類を高速に行うことを目標とした．第 2 段階では，分類要素によって決められるカテゴリに要素を分類し，分類は Tian ら[4]と同じ分類精度になることを目標とした．以下で各段階の分類手法について述べる．

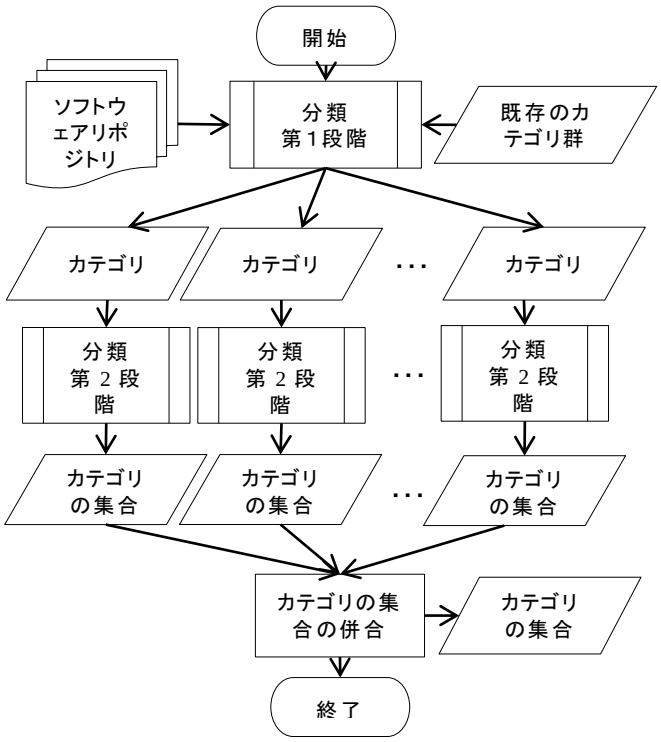

図 3 提案手法によるソフトウェアリポジトリ分類手法のフローチャート

3.3.1. 第 1 段階

図 3 の「分類第 1 段階」の処理では，分類するカテゴリをあらかじめ作ることで，プロジェクト追加時の再分類を高速化する．まず，3.2.1 で述べた手法で，分類するプロジェクトを文書に変換する．ただし，Tian らの手法と異なり，抽出する識別子はソースコード内で宣言された関数名に限定する．理由は以下の 2 つである．

1. LDA は取り扱う識別子の数が多くなると計算コストが大きくなるため，宣言された関数名のみを用いて計算コストを小さくする．
2. 宣言された関数名はプロジェクトの機能を端的に表しているため，無視してしまうと機能とは関係の無い分類ができる．

次に，3.2.3 で述べた方法でプロジェクトの特徴量を取得する．ここで，3.2.1 で述べた文書集合 A，B の選

び方は，異なる文書集合を選ぶこととする．異なる文書集合を選ぶことで，行列 **Z** は特定の文書集合の特徴に基づいて得ることができる．特定の機能に基づく行列 **Z** からは，特定の機能に基づくソフトウェアリポジトリのカテゴリが生成されるため，あらかじめ決められたカテゴリにプロジェクトを分類できる．図 2 における「ベクトル空間」をあらかじめ生成しておけるため，より高速に分類が可能となる．続く 3.2.4 における閾値によって，各々のプロジェクトがあらかじめ決められたどのカテゴリに含まれるかを決める．

3.3.2. 第 2 段階

図 3 の「分類第 2 段階」の処理では，Tian ら[4]の手法をそのまま用いる．具体的には，3.2 で述べた手法でプロジェクト群を分類する．ただし，3.2.1 においては，すべての型名・関数名・変数名を用いる．また，3.2.3 における，文書集合 A，B の選び方は，同じ文書集合を選ぶこととする．これにより，分類するプロジェクトの特徴を最も顕著に表すカテゴリに分類することができる．

3.4. プロジェクト追加時の再分類方法

提案手法では，既存の分類結果に対して新たにプロジェクトが追加された場合でもすべてのプロジェクトを再分類することなく，局所的な再分類を可能としている．プロジェクト追加時の局所的な再分類は，3.3 で述べた手法と同様に 2 段階によりを行う．プロジェクト追加時の再分類の様子を図 4 に示す．以下では，プロジェクト追加時の再分類の手順を説明する．

3.4.1. 第 1 段階

分類第 1 段階では，追加するプロジェクトのみについて 3.3.1 で示した手順を適用する．これにより，追加するプロジェクトの分類先となるカテゴリを決めることができる．このカテゴリは追加前の分類における，第 1 段階でつくられるカテゴリと同じで，あらかじめ決められたカテゴリを指す．

3.4.2. 第 2 段階

分類第 1 段階にて決定される，追加するプロジェクトのカテゴリに対してのみ，第 2 段階の分類を行う．分類は，当該カテゴリのプロジェクト群に対して，3.3.2 で示した手順を適用することで行う．当該カテゴリは追加するプロジェクトによって分類が変化してしまう可能性があるため，分類に属するすべてのプロジェクトを再分類する．当該カテゴリ以外は，プロジェクト追加前後において全く変わらないため，追加前の分類結果をそのまま使うことが可能である．

4. 評価実験

本手法の目的は，既存手法である Tian らの手法[4]と同等の分類精度を保ちながら，インクリメンタルなソフトウェアリポジトリに対する再分類速度を向上させることである．本節で述べる実験の結果より，本手法の目的が達成されているかを評価する．

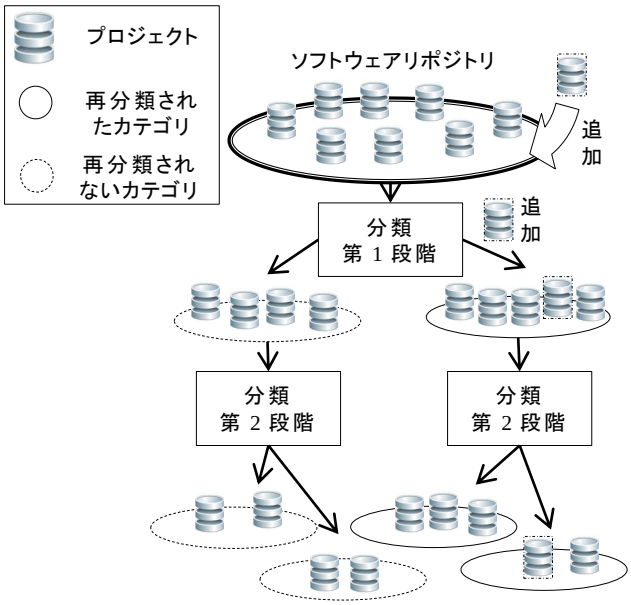

図 4 プロジェクト追加時の再分類の様子

4.1. 実験方法

本節では以下の 3 つの比較実験を行う．

1. 提案手法と既存手法[4]との比較実験
2. 分類第 1 段階の作るカテゴリの数による比較実験
3. プロジェクト追加順による比較実験

提案手法と既存手法[4]との比較実験では，提案手法の目的が達成されていることを示す．分類第 1 段階の作るカテゴリの数による比較実験では，本手法における分類第 1 段階の作るカテゴリの数による，再分類速度と分類精度への影響を示す．プロジェクト追加順による比較実験では，提案手法において，既存の分類結果に対するプロジェクトの追加順を変化させたときの，再分類速度と分類精度への影響を示す．

本節で行う比較実験は，以下の方法により行う．まず，実験では，評価用データセットとして，SourceForge[2] よりランダムに取得した 200 セットの Java 言語を用いたプロジェクト用いる．実験は，200 セットの評価用データセットを分類することで行う．ただし，200 セットのうち 100 セットを分類した後，残った 100 セットを 1 セットずつ追加し，追加の度に再分類を行う．また， i 番目のプロジェクトを追加したときの再分類とは，$100+i$ セットのプロジェクトに対して分類しなおすことを指す．なお，実験では表 1 に示す性能を持つコンピュータを使用するものとする．

表 1　実験に用いたコンピュータの性能

項目	性能
CPU	Intel Core i7 930 @ 2.8GHz
メモリ	DDR3-SDRAM 12GB
ハードディスク	2TB Serial ATA 7200 rpm RAID0
OS	Windows 7 Professional Professional 64bit
コンパイラ	javac 1.6.0_20

4.2.　評価方法

4.1 の実験結果の評価には，再分類速度と分類精度の 2 つの評価指標を用いる．本項では，2 つの評価指標について詳しく述べていく．

4.2.1. 再分類速度

再分類速度とは，インクリメンタルなソフトウェアリポジトリに対する再分類を行う速度のことである．再分類速度を評価するために，再分類を行う時間である再分類時間と，既存手法[4]の再分類速度に対する提案手法の再分類速度比 (*SpeedUpRatio*) を評価指標として用いる．再分類速度比は，以下の式の値を用いる．

$$SpeedUpRatio_i = \frac{T_{2i}}{T_{1i}} \ (i > 0) \qquad (1)$$

なお，$SpeedUpRatio_i$ は i 番目のプロジェクトを追加したときの再分類速度比，T_{1i} は提案手法による i 番目のプロジェクトを追加したときの再分類時間，T_{2i} は既存手法[4]による i 番目のプロジェクトを追加したときの再分類時間，すなわち i 番目のプロジェクトを追加し再度全体を分類するのにかかった時間とする．

4.2.2. 分類精度

分類精度とは，ソフトウェアリポジトリ分類の正しさのことである．ソフトウェアリポジトリの正しい分類結果である正解カテゴリは，評価用データセットの SourceForge[2]中における各プロジェクトの所属カテゴリとする．このカテゴリは，プロジェクトのすべての機能を知る開発者が指定するため正しいとされる．分類精度の評価には，精度 (*Precision*) と網羅率 (*Recall*) を評価指標として用いる．精度とは，生成されたカテゴリ群がどの程度正しいかを評価する指標である．また，網羅率とは，生成しうるソフトウェアリポジトリのカテゴリ群をどの程度網羅しているかを評価する指標である．なお，生成しうるソフトウェアリポジトリのカテゴリ群とは，ソフトウェアリポジトリ中のプロジェクトの機能をすべて把握している人が作ることのできる，すべてのカテゴリのことを指す．精度と網羅率には，以下の (式 2) と (式 3) の値を用いる．

$$Precision = \frac{\sum_{i=1}^{C} \max_{j \in S} \frac{n_{ij}}{N_i}}{C} \qquad (2)$$

$$Recall = \frac{\sum_{i=1}^{C} \max_{j \in S} \frac{\acute{n}_{ij}}{\acute{N}_j}}{C} \qquad (3)$$

なお，C は作られるカテゴリの数，S は正解カテゴリの通し番号の集合を示す．また，N_i はカテゴリ i に分類されたプロジェクトの数，n_{ij} はカテゴリ i に分類された正解カテゴリ j 中のプロジェクトの数を示す．最後に，$\acute{N}_j$ は正解のカテゴリ j 中のすべてのプロジェクトの数，$\acute{n}_{ij}$ はカテゴリ i に分類された正解カテゴリ j 中のプロジェクトの数を示す．ここで，分類結果と正解カテゴリにおいて，要素が 1 つ以下のカテゴリは除外するものとする．

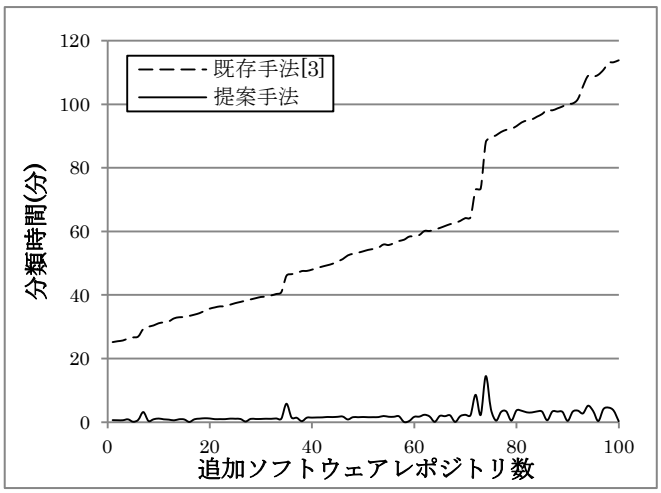

図 5　提案手法と既存手法[4]との比較実験における再分類時間結果

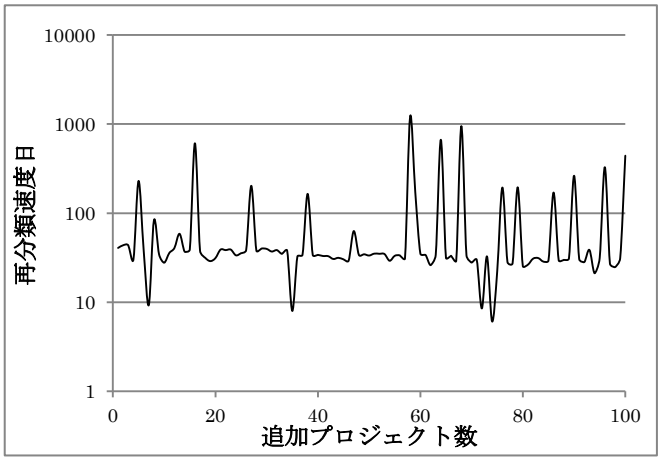

図 6　提案手法と既存手法[4]との比較実験における再分類速度比結果

4.3.　評価結果

本節では，4.1 で述べた 3 つの実験ごとに，評価結果を述べる．

4.3.1. 提案手法と既存手法[4]との比較実験

提案手法と既存手法[4]との比較実験では，提案手法と既存手法[4]の各々を用いて実験を行った．評価結果は，再分類速度と分類精度について述べていく．

再分類速度

図 5 は，実験結果に対する再分類時間を示したグラフである．グラフの横軸は追加したプロジェクトの数で，縦軸はプロジェクト追加時の再分類にかかる時間を表す．図 6 は図 5 の結果から算出された，実験結果における再分類速度比を示したグラフである．グラフ

表 2　提案手法と既存手法[4]との比較実験における初期分類時間結果

	提案手法	既存手法[4]
初期分類時間(分)	25.4	25.3

の横軸は追加したプロジェクトの数で，縦軸はプロジェクト追加時の既存手法[4]に対する再分類速度比(式 1) である．表 2 は，実験結果における初期分類時間を示している．初期分類時間とは，分類する 200 セットのプロジェクトの中で，最初に分類する 100 セットの分類にかかる時間である．

図 6 より，実験では提案手法は既存手法である既存手法[4]に対して最大で約 1213 倍，最小でも約 6 倍速度向上を達成できたことがわかる．追加プロジェクト数に対して速度比が大きく変動しているが，これは追加するプロジェクトによって追加される分類の数が大きく変化するためである．また，表 2 より，一度にソフトウェアリポジトリを分類するときは，既存手法[4]とほぼ同じ時間で，分類ができることがわかる．

分類精度

表 3 は，実験結果における分類精度を示している．

表 3　提案手法と既存手法[4]との比較実験における分類精度結果

	提案手法	既存手法[4]
Precision	0.48	0.53
Recall	0.38	0.41

表 3 により，提案手法は既存手法[4]とほぼ同等の精度を持つことがわかる．しかし，第 2 段階ではほぼ同じ分類を行っているのに，提案手法の精度と網羅率の両方が，既存手法[4]より減少している．これは第 1 段階の分類で誤って正しいカテゴリを壊してしまったためと考えられる．

4.3.2. 分類第 1 段階の作るカテゴリの数による比較実験

分類第 1 段階の作るカテゴリの数による比較実験では，分類第 1 段階の作るカテゴリの数を，10,50,100,500,1000 と変化させて実験を行った．評価結果は，再分類時間と分類精度について述べる．

再分類時間

表 4 は，実験結果における再分類時間の各代表値を示した表である．表 5 の各項目の意味は表 2 の通りである．

表 4 により，以下の 2 つの実験結果を得られた．

1. Average の項目により，分類第 1 段階の作るカテゴリ数を多くすると，再分類時間が大きくなる．
2. Correl の項目により，分類第 1 段階の作るカテゴリ数を少なくすると，分類結果に追加するプロジェクト数の増加に伴う，再分類時間の増加が大きくなる

表 4　分類第 1 段階の作るカテゴリ数による比較実験の再分類時間の代表値

カテゴリ数	Average (分)	Correl
10	0.74	0.81
50	0.84	0.45
100	1.89	0.45
500	4.31	0.28
1000	5.80	0.18

表 5　実験結果における再分類時間の代表値の意味

項目名	意味
カテゴリ数	分類第 1 段階の作るカテゴリの数
Average	100 セットのプロジェクト追加における再分類時間の平均
Correl	100 セットのプロジェクト追加における，再分類時間と追加したプロジェクトのセット数との相関係数

1 つ目の結果は，分類第 1 段階の再分類時間が大きくなることによるものである．2 つ目の結果は，分類第 1 段階の作るカテゴリの数を多くすることで，既存の分類結果に対するプロジェクト追加時に再分類するカテゴリをより細かく決められることによるものである．これにより，追加するプロジェクトに関係のない再分類を少なくできるため，より高速に再分類が可能となる．

分類精度

表 6 は，実験結果の分類精度を示した表である．なお，表 6 のラベルのカテゴリ数とは，提案手法の分類第 1 段階の作るカテゴリの数のことである．

表 6　分類第 1 段階の作るカテゴリ数による比較実験の分類精度

カテゴリ数	10	50	100	500	1000
Precision	0.46	0.49	0.48	0.49	0.51
Recall	0.42	0.40	0.39	0.41	0.41

表 6 により，分類第 1 段階の作るカテゴリ数によらず，表 3 の既存手法[4]とほぼ同じ精度で分類できることがわかる．また，提案手法の分類第 1 段階の作るカテゴリの数が多くなると，Precision が高くなる傾向があることがわかる．これは，提案手法では，最終的に既存手法[4]と同じ方法で分類するためである．また，分類第 1 段階の作るカテゴリ数を多くすると，Precision が高くなるのは，分類第 1 段階をより詳細に行うことで，最終的に，より正確な分類結果となるためである．

4.3.3. プロジェクト追加順による比較実験

プロジェクト追加順による比較実験では，100 セットのプロジェクトの追加順を変化させて，提案手法に対する実験を 10 回行った．評価結果は，再分類時間と分類精度について述べていく．

再分類時間

表 7 は，実験結果における再分類時間の代表値を示した表である．なお，Max はプロジェクト追加時の最大分類時間，Min はプロジェクト追加時の最小分類時間を表し，Ave.は表 5 に示す通りである．

表 7　プロジェクトの追加順による比較実験の再分類時間の代表値

	Max(分)	Min(分)	Ave.(分)
1 回目	11.41	0.06	1.94
2 回目	12.10	0.03	1.73
3 回目	12.48	0.03	1.88
4 回目	11.45	0.10	1.79
5 回目	12.92	0.09	2.22
6 回目	13.14	0.05	1.06
7 回目	11.29	0.03	1.46
8 回目	12.22	0.03	1.29
9 回目	10.58	0.08	2.39
10 回目	10.63	0.02	2.07

表 7 により，提案手法では，既存の分類結果に対するプロジェクトの追加順によって，再分類時間が大きく変化しないことがわかる．

分類精度

表 8 は，実験結果の分類精度を示した表である．

表 8　プロジェクトの追加順による比較実験の分類精度

	Precision	Recall
1 回目	0.48	0.40
2 回目	0.48	0.40
3 回目	0.48	0.39
4 回目	0.47	0.41
5 回目	0.48	0.39
6 回目	0.49	0.39
7 回目	0.49	0.39
8 回目	0.50	0.39
9 回目	0.49	0.40
10 回目	0.49	0.39

表 8 により，提案手法では，提案手法では，既存の分類結果に対するプロジェクトの追加順によって，分類精度は大きく変化しないことがわかる．

5.　まとめ

本稿では，2 段階における LDA を用いたソフトウェアリポジトリ分類手法について説明し実験を行った．結果として，既存手法と比べて同等の精度と網羅率を保ちつつ，200 プロジェクトにおけるインクリメンタルなソフトウェアリポジトリにおける再分類速度を最大約 1213 倍向上させることに成功した．

今後の展望としては，機能に基づくソースコード検索システムを構築したい．提案手法が生成する機能ごとのカテゴリを用いることで，プロジェクト中のソースコードを，機能に基づいて検索できる．検索システムを実現するにあたり，以下の 2 つの課題がある．

1. カテゴリ名の生成
2. 分類精度と網羅率の向上

まず，1 つ目の課題について述べる．ソースコードを検索するために，提案手法の作るカテゴリを検索するが，検索にはカテゴリの名前が必要となる．よって，プロジェクトの分類と同時に，カテゴリ名を生成する必要がある．次に，2 つ目の課題について述べる．検索システムでは，高い検索精度が求められる．よって，提案手法を用いるには，提案手法の分類精度が高くなければならない．つまり，提案手法の分類精度と網羅率を向上させる必要がある．

参　考　文　献

[1] S. Thummalapenta and T. Xie, “Parseweb: a programmer assistant for reusing open source code on the web”, Proc. of the twenty-second IEEE/ACM international conference on Automated software engineering, pp.204-213, 2007.

[2] SourceForge.net. http://sourceforge.net/

[3] S. Kawaguchi, P.K. Garg, M. Matsushita and K. Inoue, “MUDAblue: An automatic categorization system for open source repositories”, Proc. of the 11th Asia-Pacific Software Engineering Conference, pp.184–193, 2004.

[4] K. Tian, M. Revelle and D. Poshyvanyk, “Using latent Dirichlet allocation for automatic categorization of software”, Proc. of the Mining Software Repositories, pp.163–166, 2009.

[5] S. Madan and S. Batra, “Discrete Characterization of Domain Using Semantic Clustering”, J. of advances in information thechnology, Vol.1, No.3, pp.127-132, 2010.

[6] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer and R. Harshman, “Indexing by Latent Semantic Analysis”, J. of American Society for Information Science, Vol.41, No.6, pp.391-407, 1990.

[7] D.M. Blei, A.Y. Ng and M.I. Jordan, “Latent Dirichlet Allocation”, J. of Machine Learning Research, Vol.3, pp.993-1022, 2003.

[8] P.S. Sandhu, J. Singh and H. Singh, “Aproaches for Categorization of Reusable Software Components”, J. of Computer Science, Vol.3, No.5, pp 266-273, 2007.

[9] JGibbLDA, http://jgibblda.sourceforge.net/

[10] T.K. Landauer, P.W. Foltz and D. Laham, “An introduction to latent semantic analysis”, Discourse Processes, Vol.25, pp.259-284, 1998.

[11] R.C. Veras, S.R.L. Meira, A.L.I. Oliveira, B.J.M. Melo, "Comparative Study of Clustering Techniques for the Organization of Software Repositories", Proc. of the 19th IEEE International Conference on Tools with Artificial Intelligence, Vol.1, pp.210-214, 2007.