特別研究報告書

複合サービス推薦のための多段階協調フィルタリング

指導教員　　石田 亨 教授

京都大学工学部情報学科

佐々木 健人

平成26年1月31日

複合サービス推薦のための多段階協調フィルタリング

佐々木 健人

内容梗概

サービスコンピューティングとはサービスをシステムの構成要素としたソフトウェア開発に関する研究領域であり, 研究とビジネス両方から注目を浴びている分野の一つである. サービスは原子サービスと複合サービスから成り, 原子サービスは単体で実行可能なサービス, 複合サービスは具体的に実行する構成サービスを指定することで複数のサービスを組み合わせて作成することが出来るサービスのことである. サービスコンピューティングにおいて, どのサービスを実際に利用するかの選択は非常に重要な課題となっている. 言語サービスを共有可能にするインターネット上の多言語サービス基盤である言語グリッドにおいては, 特に複合サービスの組み合わせに関してより良い品質を示す可能性がある代替サービスが存在するにもかかわらず, その存在がユーザに伝わっていない. これまで, サービス品質 (Quality of Service, QoS) を評価値として用いた協調フィルタリングによるサービス推薦の研究が盛んに行われてきた. しかし, ユーザに依存する QoS 値を得ることが出来ない場合は多く, また従来研究では原子サービスのみを対象とした推薦は行われているが, 複合サービスにおける呼び出し関係を考慮した推薦に関する研究は行われていない.

そこで, 本研究ではユーザのサービスの呼び出し回数を評価値とした協調フィルタリングを用いた多段階協調フィルタリングを提案する. 多段階協調フィルタリングにおいては, 複合サービスが推薦された際には, 複合サービスへバインディングする構成サービスも共に推薦する. 具体的には, 複合サービスに特有のネスト構造に着目し, 実行可能な原子サービスがユーザに推薦されるまで繰り返し協調フィルタリングを適用することで複合サービスを推薦する. 本手法の実現にあたり, 取り組むべき課題は以下の 2 つである.

推薦対象ユーザの複合サービスを通じた評価値の予測

複合サービスの推薦対象ユーザの複合サービスの用い方を予測するためには, そのユーザがその複合サービスを用いてどのようなサービスを用いるかという情報が必要である. しかし, 協調フィルタリングを用いて複合サービスが推薦されたユーザには, その複合サービスに関する利用情報が存在しないので, この情報を予測する必要がある.

制約を満たす複合サービスの作成

複合サービスの構成サービスを決定する際には, サービスとユーザ双方によって生じる制約を考慮しなければならない. 言語サービスがコンポーネントとして存在する言語グリッドにおいては, 翻訳, 形態素解析などのサービスタイプ, そしてユーザとサービスの対応言語が制約となる.

一つ目の課題に対しては, ユーザからのサービスへの直接の呼び出し回数の情報を用いて協調フィルタリングを適用した際, 複合サービスが推薦されれば, 複合サービスを通じたサービスの呼び出し情報から評価値行列を作成し再度協調フィルタリングを行う. 推薦対象ユーザがその複合サービスを過去に利用したことがなければ, 類似ユーザを推薦対象とみなし評価値を推定して, 類似度を用いた加重平均により推薦対象ユーザの評価値を推定する. 二つ目の課題に対しては, サービスタイプと対応言語の制約を満たすように複合サービスにバインドするサービスを選択し, 推薦対象ユーザへ推薦する複合サービスを作成する.

提案手法を実装し, 言語グリッドにおける複合サービスの推薦システムを構築した. このシステムのアーキテクチャは, 推薦の実行を担う推薦制御部, 実際の言語グリッドのログデータを解析して推薦システムに適した形にデータを整形するデータ整形部, 複合サービスを通じた推薦対象ユーザのサービスに対する評価値を予測し求める評価値予測部, 最後に予測された評価値をもとに推薦する複合サービスを作成する複合サービス作成部からなる. このシステムを用いた評価実験を行い評価値予測精度と推薦実行可能率の観点から評価を行い提案手法の有効性を検証した．本研究の貢献は以下の通りである.

推薦対象ユーザの複合サービスを通じた評価値の予測手法

協調フィルタリングを多段階に適用することにより複合サービスの呼び出しの文脈を考慮した評価値の予測を行った．その結果，文脈を考慮しない多段階の協調フィルタリングの約 2 倍の評価値予測精度 (MAE 値) を得た.

サービスの制約を考慮した複合サービス作成手法

サービスの制約を考慮し，サービスの集合として複合サービス推薦を行った．その結果，組み合わせ方ごとに区別した複合サービスを通常の協調フィルタリングを用いて推薦する手法より複合サービスの推薦実行可能性が向上するわけではないという結果を得た.

Multi-stage Collaborative Filtering for Composite Service Recommendation

Taketo Sasaki

Abstract

Services computing, which is a research fields in the software development, is attracting much attention in both research and business communities. Service consists of atomic service, which can be run alone and composite service, which can be made by combining several services. In services computing, selecting the service from available services with the equivalent function is an important topic. In the Language Grid, a multilingual service infrastructure which enable sharing language services, users cannot know the existence of alternative services that will show better quality. To date, service recommendation by collaborative filtering using Quality of Service (QoS) as the numerical value of service evaluation has been an active area of research. However, it is common that evaluation value from users are not available. In addition, existing methods deal with only atomic services.

In this research, we propose a service recommendation method by multi-stage collaborative filtering using the number of service invocation as evaluation value. In multi-stage collaborative filtering, composite services are recommended with their component services to bind. In particular, this is realized by applying collaborative filtering repeatedly until runnable atomic services are recommended focusing on nested structure in composite service. To realize this method, we face follow two problems.

Prediction of composite service usage information about the recommendation target user

In order to predict how the user who is recommended an composite service use the composite service, information about which services the user invokes through the composite service is needed. However, there is no composite service usage information about the user. So this information need to be predicted.

Making composite service considering constraints

Supported languages and service types are constraints which have to be

considered when service recommendation in the Language Grid. In the Language Grid, service types such as translation or morphological analysis, and supported language can be constraints.

For first problem, if composite service is recommended by collaborative filtering using direct invocation information about user, then make evaluation value matrix through composite service and apply collaborative filtering again. If target user have not used this composite service, proposed method predicts evaluation value by collaborative filtering regarding similar user as target user, and calculating similarity weighted average. For second problem, proposed method select service to bind to composite service considering service type and supported language and make composite service to recommend.

we implemented proposed method and built composite service recommendation system in Language Grid. This system consists of the four components: the recommendation controller component, the data parser component, the evaluation value predictor component and composite service maker component. we performed evaluation using this system and discussed about problems in composite service recommendation. We evaluated proposed method in terms of prediction accuracy of evaluation value and execution possibility of composite service recommendation to inspect efficiency of proposed method. Contributions of this research are as follows:

Prediction of evaluation value through composite service

By applying multi-stage collaborative filtering, we predicted the evaluation value of recommendation target user considering composite service invocation contexts. As a result, we got about 2 times prediction accuracy(MAE value) compared with multi-stage collaborative filtering not considering composite service invocation contexts.

Composite service creation considering constraints in service

We recommended composite service as a set of services considering constraints in services. Then we got a result that proposed method not always improve execution possibility of composite service recommendation compared with collaborative filtering consider each composite service composition as different service.

複合サービス推薦のための多段階協調フィルタリング

目次

第1章　はじめに

近年，Web サービスにおいて，サービスコンピューティング [1] という概念が急速に発達している．サービスコンピューティングとは，迅速かつ柔軟なサービス開発を目標として開発された技術である．具体例として，機械翻訳サービスや専門用語対訳辞書サービスがコンポーネントとして存在する場合を考える．それぞれのサービスが独立して存在する場合，それぞれのサービスは別々に使われるのみである．しかし，これらが共通のインタフェースを持つ場合，機械翻訳と対訳辞書サービスを組み合わせることが可能となる．そのため，一から開発を行うことなく辞書連携機械翻訳サービスを実現し，ドメインに特化したサービスを提供することができる．

言語グリッドプロジェクト [2] は，サービスコンピューティングの先駆的な研究を行うための基盤を提供している．機械翻訳，専門用語対訳辞書，形態素解析などのサービスがコンポーネントとして存在し，これらの多様な言語資源を組み合わせることによってユーザの環境に応じた複合サービスを迅速に作成することが可能となっている．

サービスコンピューティングにおいて，ユーザの目的に沿った有用なサービスを推薦することは潜在的に存在するより良い品質を示すサービスの発見のために重要である．ユーザ自身の利用情報に基づいてこのようなサービス推薦が自動的に行われれば，ユーザのサービス発見・構築コストの削減が期待できる．これまでの，サービス推薦システムにおいて広く用いられてきたサービスの評価指標は，客観的なサービスの評価値を表す Qualoty of Service(QoS)[3] であった．機械翻訳をはじめとする言語サービスにおいては，翻訳精度を数値化し，QoS として用いている．QoS をユーザのサービスに対する評価値であるとみなし，協調フィルタリングによってユーザにとって有用であろうサービスを推薦する試みがなされてきた．協調フィルタリングとは，他の類似するユーザやアイテムから情報を収集することで現在のユーザの評価値を自動的に予測する手法のことである [4].

しかし，ウェブサービスには原子サービスと原子サービスを組み合わせて一つの新たなサービスとする複合サービスが存在する．QoS 値が高い場合でも，複合サービスを推薦されれば，そのままでは実行不可能であるので推薦されたサービスがユーザにとって有益であるかどうかが判断できない．先行研究は多々

あれど複合サービスの構成サービスにまで目を向けた推薦を行っている先行研究は存在しない．複合サービスにバインド可能な組み合わせを全て別のものとみなせば，構成サービスを考慮した推薦が実現できるが，組み合わせの数が膨大になると現実的でないことは容易に想像できる．言語グリッドにおいては，日英間翻訳に限った場合でも，5 つの機械翻訳サービスが存在する．さらに，対訳辞書サービスは無数に存在するために辞書連携翻訳の組み合わせは無数に存在しうる．しかも，オープン言語グリッドと LanguageMashup の出現によりサービス数の増加が見込まれる現在となっては一層現実的ではない．また，複合サービスを通じたサービスの呼び出し回数に大きな偏りが存在し同機能の代替サービス候補の存在にユーザが気付く事が出来ていないという問題も存在する．

本研究では，ユーザからの明示的な評価値が得られない場合に，ユーザのサービスの利用履歴，すなわちサービスの呼び出し回数を評価値として用いた協調フィルタリングによるサービス推薦の枠組みを考える．サービスの呼び出し回数は，ログデータさえ存在すれば容易に収集することが可能である．また，ユーザの利用方法を考慮したサービスの組み合わせを複合サービスという形で推薦することも考える．他のユーザの組み合わせに関する知識を他のユーザに伝える事が目的である．

呼び出し回数を評価値として用いた協調フィルタリングによるサービス推薦の概要を示す．まず，ユーザのサービスの利用履歴から各々のユーザがどのサービスを何度呼び出したかという情報を持つ行列を作成する．この行列を入力としてユーザベースの協調フィルタリングを用いて類似ユーザの利用情報に基づいてサービスを推薦する．

しかし，協調フィルタリングを単純にサービス推薦に適用するだけでは不十分である．その理由として，Web サービスに特有の複合サービスの存在が挙げられる．複合サービスそのものを推薦することに意味はなくむしろ，複合サービスを推薦する際は，その構成サービスをどのように組み合わせて複合サービスを用いるのかを推薦すべきである．これに対して，実行可能な複合サービスを一つのアイテムとして協調フィルタリングを適用することが考えられるがユーザ間で共通に呼び出される複合サービスの数が少なくなるため，原子サービスの利用分布によってのみ類似ユーザが推薦されてしまい，適切な複合サービスが推薦できないという問題が存在する．

本研究では，上記の問題を解決するために，複合サービスの呼び出し関係に

より生じる呼び出し文脈を考慮し，ユーザのサービスの呼び出し回数をサービスに対する評価値として用いた協調フィルタリングを多段階に適用することにより，複合サービス推薦の際に構成サービスの組み合わせとしての複合サービスの推薦を実現するというアプローチをとった．多段階に協調フィルタリングを適用する際の，ユーザの利用履歴に基づいた欠落情報の補完手法とサービスの特性を制約として用いた推薦複合サービスの具象化の手法を提案する．複合サービス具象化とは，複合サービスにバインディングする構成サービスを実際に決定することを指す．本稿の残りは以下のような構成となっている．第 2 章でサービス推薦に関する関連研究を紹介し，第 3 章で多段階協調フィルタリングを用いた複合サービス推薦，その際に必要になる情報をユーザの利用情報に基づいて補完する手法，サービスの特性を制約とした複合サービスの具象化手法について説明する．第 4 章で提案手法を用いたシステムについての説明を行った後に，第 5 章でこの手法の評価を行い第 6 章で本稿をまとめる．

第 2 章　サービス推薦

本章では，既存の協調フィルタリングを用いたサービス推薦手法について説明する．大別して，内容ベースの協調フィルタリングを用いたサービス推薦手法とユーザの利用情報や評価に基づく協調ベースの協調フィルタリングを用いたサービス選択手法が存在するので，それらを順に説明する．

2.1　内容に基づく推薦

内容ベースの手法において，ユーザが推薦されるものはユーザが過去に好んだものに類似するものである．[5] は，サービスのユーザが自身の要求を意味文書を用いて提供するというスキームを提案した．また，サービス提供者が自身の QoS 情報とサービスのプロファイルを登録するということも提案した．提案されたスキームは，意味文書中のエラーによって正確に動作しなくなるので，意味的パラメータは検証されていなければならない．

内容によるサービスマッチングの為のスキームは，[6] において提案されており，計算資源の特徴が推薦システムにおけるアイテムの特徴に変換される．さらに過去の記録を資源の特徴とした充足度合いを測定するシステムを作成した．[5] においては，意味合致アルゴリズムがユーザの要求とすべての利用可能な

サービスの照合の為に用いられた．これが推薦エンジンに搭載され，協調フィルタリングとユーザの評価による推薦を並び替えフィルタリングする．

木構造を解析する方法は，合致するサービスを取得する際に用いることが出来る．サービスは，自身に関連付けられたタグに基づいて選択することも可能である．タグとユーザからの入力を照合することでサービスを選択することが可能である [7]．この手法においては，タグの質が直接的に推薦の精度に影響する．

2.2 協調推薦

協調ベースの手法において，ユーザが推薦されるものはネットワーク上の類似ユーザから収集された情報に基づいて決定される．

[6, 8] においては，推薦システムにおいて協調フィルタリングのアルゴリズムを用いることが提案された．改良されたアイテムベースの協調フィルタリングのアルゴリズムが [6] において提案されている．このアルゴリズムにおいては，複雑で様々なものが存在するサービスの記述をフィルタリングし分類するためにサービスのマッチングプロセスにユーザの経験を導入することによってサービスのマッチングプロセスを最適化することが可能である. これにより以下のことが可能となる．

- 新たなサービスを推薦したり, サービス間の関係を調べること
- ユーザの行動からのフィードバック情報を学習することで素早くサービスを照合し, 発見すること

内容ベースのアルゴリズムは，ユーザは毎回異なる特徴を入力する可能性が存在するので，サービス推薦においては好ましくない．ユーザに対する推薦は，現在の入力に関連がなければならない．[9] では，Weighted Slope One (WSO)[10] と Artificial Immune System (AIS) という二つの異なる協調フィルタリングのアルゴリズムが比較されている．また，推薦の効果と精度を改善するために行列の因数分解手法が [11] において提案されている．

[12] においては，ユーザベースの協調フィルタリングとアイテムベースの協調フィルタリングのハイブリッド方式を用いた協調フィルタリングによるサービス推薦手法が提案されている．

第3章　多段階協調フィルタリング

3.1　アルゴリズム概要

本研究では，協調フィルタリングアルゴリズムを用いて対象ユーザへの複合サービス推薦を構成サービスの集合として推薦することで，実行可能な状態での複合サービスの推薦を実現する.

アイテムの集合を協調フィルタリングを用いて推薦する既存手法として [13] が存在する．[13] においては，各々のユーザが予算をもち，各々のアイテムがコストとユーザからの評価値をもつという環境においてコストに収まる範囲で評価値の合計が最大となるように推薦集合を作成するというものである．具体的な問題として，以下の二つの例が存在する.

旅行の行き先リストの推薦

複数の観光地とそれぞれに対応するコストが存在する場合に，ユーザの評価値 (満足度) を最大にして，かつ予算内で巡る事ができる観光スポットの選定を行う

twitter のフォローすべきユーザリストの推薦

ユーザが一日に処理できる情報量 (読む事ができる記事の量) が設定されていて，各々のユーザに一日に配信する情報量とユーザからの評価値が設定されている環境において，ユーザの評価値が最大になり，かつユーザが処理できる情報量を配信するユーザリストを作成する

本研究において，既存のアイテムの集合の推薦 [13] と異なる点は，集合の内部に集合が存在する，すなわち集合に階層構造が存在する可能性が存在する点である．これは，複合サービスに複合サービスがバインド可能であるということから起こりうる事象である.

問題の具体例を説明する．そのために, まず言語グリッドについて説明する. 言語グリッドは, 辞書や機械翻訳などの言語資源を言語サービスとして登録し, 共有可能にするインターネット上の多言語サービス基盤である. ここでは, ユーザは言語サービスを組み合わせて利用することが可能である. 言語グリッドには, インタフェースの標準化された機械翻訳, 辞書, 形態素解析といったサービスタイプが存在する. 複合サービスには, 以下のようなサービスが存在する.

- TranslationWithDictionary

  このサービスは, 機械翻訳, 辞書, 形態素解析を組み合わせて実行可能となる

サービスである. 機械翻訳の結果に指定した辞書の登録語を反映することが可能.

- TwoHopTranslation
  このサービスは, 機械翻訳サービスを二つ組み合わせて実行可能となるサービスである. これらの翻訳サービスが連続して翻訳を実行する.
- BackTranslation
  このサービスは, 機械翻訳サービスを二つ組み合わせて実行可能となるサービスである. これらの翻訳サービスが連続して翻訳を実行する. 二つ目の翻訳サービスは, 一つ目の翻訳サービスの翻訳結果を入力文として翻訳を実行する.
- BillingualDictionaryCrossSearch
  このサービスは，複数の対訳サービスを組み合わせて実行可能となるサービスである．複数の対訳辞書サービスが同時に呼び出される.

サービス合成は既存のサービスを組み合わせることで新たなサービスを作成することを可能にする．サービスは原子サービスと複合サービスの 2 種類から成る．原子サービスは，その実行のために他のサービスへ依存しない．具体的には，機械翻訳サービスは翻訳結果をユーザに提示するのに他のサービスに依存しない．対して複合サービスは，その実行のために他のサービス (構成サービスと呼ぶ) に依存する．具体的には，機械翻訳サービスに対訳辞書を組み合わせて用いる事ができる辞書連携翻訳サービスにおいては，サービスの実行のために機械翻訳サービスと形態素解析サービスと対訳辞書サービスが必要となる.

ここで，問題となるのがユーザに複合サービスのサービス構成が固定化されていることである．例えば，ユーザの農業辞書連携翻訳サービスの構成が固定化されていることである (図 1)．現在のサービスの構成よりも良い品質を示すサービスの構成が存在するにもかかわらず，ユーザがそれに気付くことが出来ていない.

これを他のユーザの利用履歴に基づいて，図 1 の中日翻訳と日英翻訳を組み合わせた TwoHop 翻訳よりも Toshiba の中英翻訳を，日本農業辞書単体よりも日本農業辞書と米辞書の辞書横断検索を推薦することを目的とする (図 2).

ユーザには，Alice，Bob，Carol，David の 4 人が存在する状況を考える．表 1 は，この 4 人のサービスに対する評価値行列である．この表は，本稿において手法がどのように働くのかを示すために用いられる.

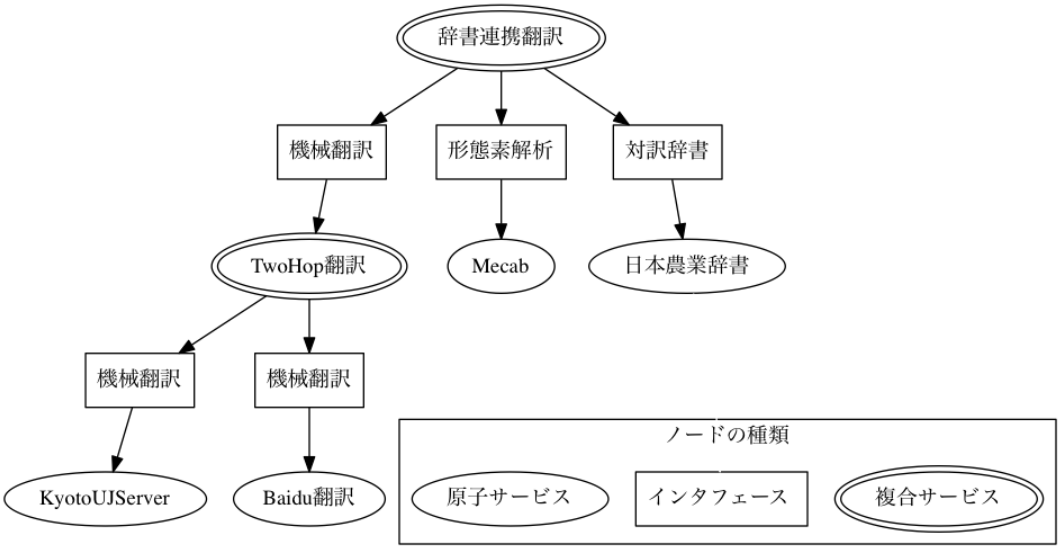

図 1: 推薦前の辞書連携翻訳のコールツリー

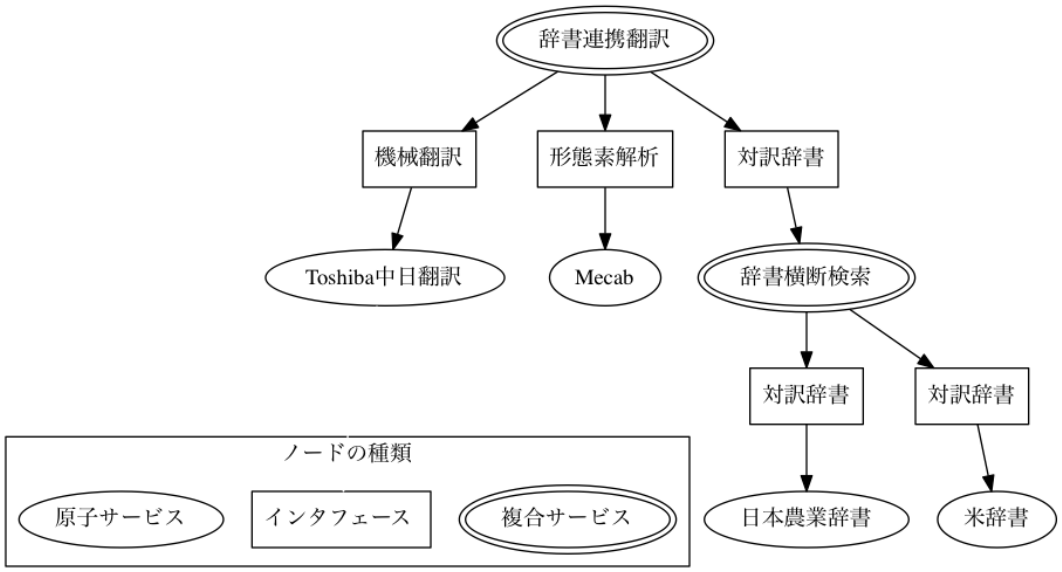

図 2: 推薦後の辞書連携翻訳のコールツリー

アルゴリズムの全体像を図 3 に示す．多段階協調フィルタリングにおいては，まず協調フィルタリングを用いて類似度計算を行い，類似ユーザを選択する．類似ユーザの評価値情報に基づいて推薦対象ユーザの評価値を予測し推薦するサービスを決定する．ここで推薦されたサービスが複合サービスでなければ処理を終了し，複合サービスが推薦されればその複合サービスを通じたユーザの評価値行列を作成する．表 1 の例を用いると，Alice に Alice の類似ユーザであ

表 1: ユーザからの直接サービス評価値行列

	KyotoU Jserver	Mecab	…	TranslationWithDictionary	TwoHop Translation
Alice	100	50		-	-
Bob	-	200		100	100
Carol	500	250		100	-
David	200	100		50	-

る Carol と David の利用情報に基づいて TranslationWithDictionary が推薦され TranslationWithDictionary を通じたサービスの評価値行列 (表 2) を作成する．この複合サービスを通じたサービスの評価値行列を CIM(CIM:Composite servies Invocation Matrix) と呼ぶことにする．

表 2: 複合サービス TranslationWithDictionary の CIM

	TGEver1	Mecab	…	TranslationWithDictionary	TwoHop Translation
Bob	100	-		50	-
Carol	50	25		-	-
David	25	50		-	-

そして，先ほど選択した類似ユーザに対してこの CIM において協調フィルタリングを適用し複合サービスを通じたサービスに対する評価値を予測する．予測された類似ユーザの評価値を用いて推薦対象ユーザの複合サービスを通じたサービスに対する評価値を予測する．先ほどの例で説明すると，表 2 には Alice に関する情報が存在しないので，表 3 のように Alice の?の部分を予測する．

以上で述べた利用情報を，類似ユーザは複合サービスの利用方法も似ているという考えに基づいて予測する．ここで用いるのは，以下のものである．

類似ユーザの CIM における評価値

推薦対象ユーザ (上記の例での Alice) に類似するユーザ (上記の例での Carol, David) が，対象ユーザに推薦された複合サービスを通じてどのようなサービスを何回呼び出しているかを表すものである．もちろん，類似ユーザであっても利用情報が存在しない場合は存在するので，その際は利用情報の

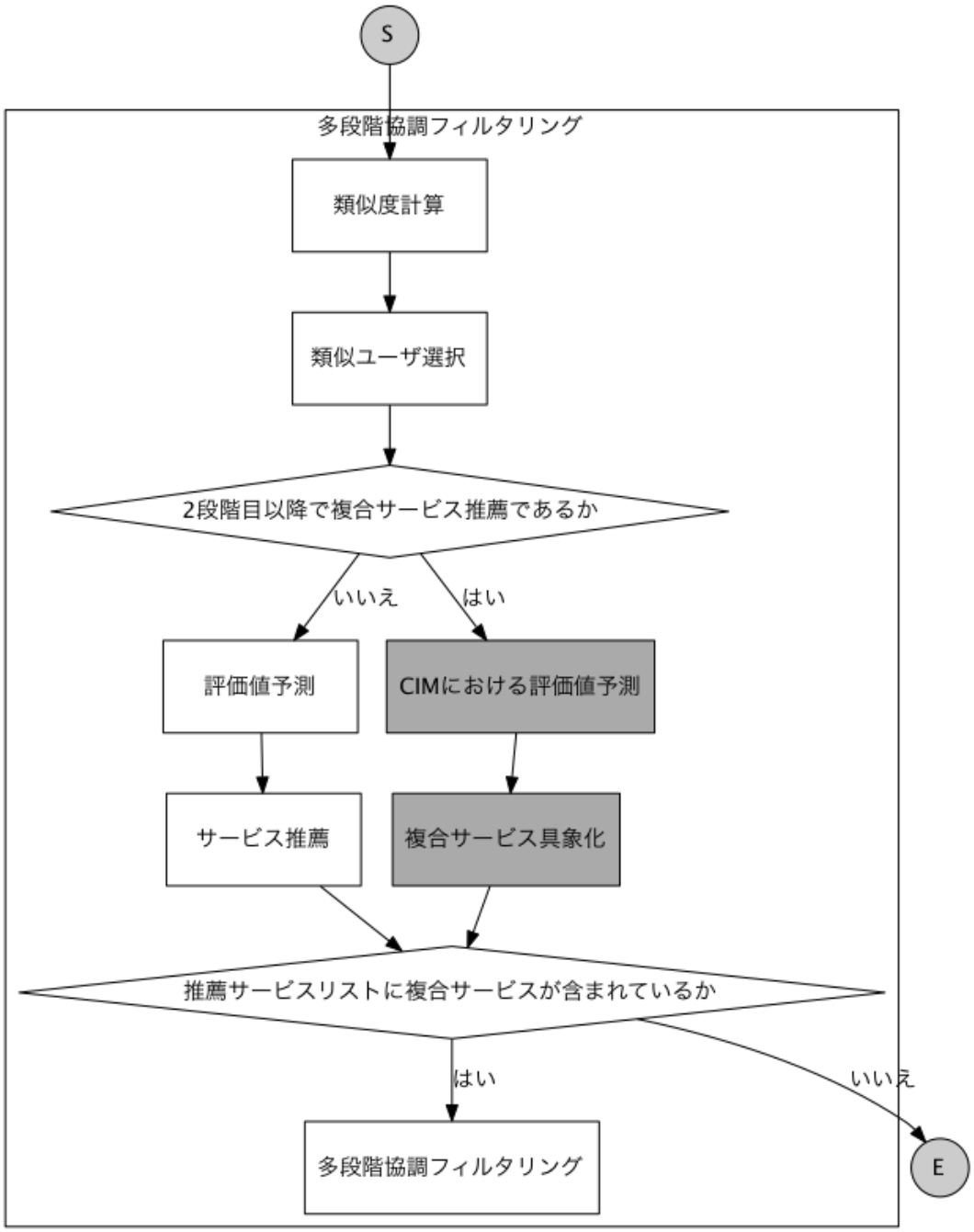

図 3: 多段階協調フィルタリングのフローチャート

存在する類似ユーザについて考える．このそれぞれの類似ユーザについて，複合サービスの CIM 上で協調フィルタリングによってすべてのサービスへの評価値の予測値を算出する．

類似ユーザとの類似度

表 3: 複合サービス TranslationWithDictionary の CIM

	TGEver1	Mecab	⋯	TranslationWithDictionary	TwoHop Translation
Alice	?	?		?	?
Bob	100	-		50	-
Carol	50	25		-	-
David	25	50		-	-

推薦対象ユーザ (上記の例での Alice) とその類似ユーザ (上記の例での Carol, David) のユーザプロファイル間の類似度．類似度の算出は，ピアソンの相関係数により行う．

上記の 2 つを用いて対象ユーザの，推薦された複合サービスの CIM における利用情報を予測する．

ここで予測した評価値を用いてユーザにとって最も評価値が高くなるような複合サービスに対する構成サービスを選択し，それらをバインディングしてユーザに推薦する．この際，複合サービスにバインディング出来るサービスのタイプが決まっているので，それを考慮したサービス集合の作成を行う必要がある．問題の例で言うと，TranslationWithDictionary には，機械翻訳，辞書，形態素解析サービスのみがバインディング出来るのでその制約を満たすように構成サービスを選択する．

以降の節でフローチャート中の網掛けで示した部分の詳細を説明する．

3.2 評価値の予測

まず，3.1 で説明した手法を厳密に議論するために本論で扱う対象を厳密に定式化する．ユーザ集合を U，原子集合サービスを A，複合サービス集合を C，サービス集合を S とし $S = A \cup C$ とする．あるユーザを $u \in U$，ある原子サービスを $a \in A$，ある複合サービスを $c \in C$，あるサービスを $s \in S$ とする．このとき，ユーザ u のサービス s に対する呼び出し回数を $v_u(s)$ と表現する．推薦システムは，この $v_u(s)$ の値が設定されていない際，ユーザ u の過去のサービスの利用履歴や類似ユーザのサービスの利用履歴を用いてその値を予測する．$s \in S$ に対して，s のサービスタイプを $ST(s)$，s の対応言語を $L(s)$ で表す．具体的には，s が形態素解析の Mecab である場合には，$ST(s) = MorphologicalAnalysis$,

$L(s) = Japanese$ となる．サービスの集合 $R \subset S$ が与えられた際，R に対するユーザ u の評価値を $v_u(R) = \sum_{s \in R} v_u(s)$ で定義する．

以上の定式化を基に3.1節のフローチャートを厳密に記述したものをAlgorithm 1に示す．

Algorithm 1 の各部分を順に詳細に説明する．まず，協調フィルタリングの部分について説明する．ユーザ u_a とユーザ u_b の類似度計算の類似度関数には，以下のピアソンの相関係数を用いる．

$$sim(u_a, u_b) = \frac{\sum_{s \in S}(v_{u_a}(s) - \overline{v_{u_a}})(v_{u_b}(s) - \overline{v_{u_b}})}{\sqrt{\sum_{s \in S}(v_{u_a}(s) - \overline{v_{u_a}})^2}\sqrt{\sum_{s \in S}(v_{u_b}(s) - \overline{v_{u_b}})^2}} \tag{1}$$

ピアソンの相関係数は正確な類似度計算を行うことが可能であるが，実際には類似ユーザでなくたまたま少数のサービスに対して類似するような評価値を与えているユーザの類似度を過大に評価してしまう [14]．この問題に対処するために，少数の共通に呼び出されたサービスの影響を減少させるために重みを導入する．改良されたピアソンの相関係数 [12] を以下に示す．

$$sim'(u_a, u_b) = \frac{2 \times |S_{u_a} \cap S_{u_b}|}{|S_{u_a}| + |S_{u_b}|} sim(u_a, u_b) \tag{2}$$

$sim'(u_a, u_b)$ が新たな類似度の値で，$|I_{u_a} \cap I_{u_b}|$ は両方のユーザに共通して呼び出されたサービスの数，そして $|S_{u_a}|$ と $|S_{u_b}|$ はユーザ u_a とユーザ u_b それぞれに呼び出されたサービスの数を表している．この式より共通に呼び出したサービスの数が少ないユーザ間の類似度は低くなる．

評価値行列の値を予測する前に，類似ユーザの集合を特定する必要がある．類似しないユーザは評価値予測精度を下げるので，類似ユーザ選択は正確な予測のために重要な段階である．本手法においては，類似ユーザを正の相関をもつユーザに限定した．

ユーザ・サービス行列における $v_u(s)$ の値を求めるために，ユーザ u の類似ユーザの集合 N は以下の式により求められる．

$$N = \{u_a | Sim'(u_a, u) > 0, u_a \neq u\} \tag{3}$$

ユーザ u のサービス s に対する評価値予測には以下の式を用いる．以下の式の N はユーザ u の類似ユーザの集合である．

$$pred(u, s) = \overline{v_u} + \frac{\sum_{u_n \in N} sim'(u, u_n) \times (v_{u_n}(s) - \overline{v_{u_n}})}{\sum_{u_n \in N} sim'(u, u_n)} \tag{4}$$

Algorithm 1 多段階協調フィルタリング *Multi-Stage Collaborative Filtering* (u, s, N, SIM) **returns** a set of services R

s /* 多段階協調フィルタリングを適用する複合サービス */
R /* 推薦されたサービス集合．$r \in RC$ */
u /* 推薦対象ユーザ */
CIM /* 複合サービスを通じた評価値行列 */
N /* 推薦対象ユーザの類似ユーザの集合 */
SIM /* ユーザと他のユーザの類似度のリスト */
$CSIM$ /* 複合サービスにおけるユーザと他のユーザの類似度のリスト */
CN /* 複合サービスにおけるユーザの近傍ユーザの集合 */
CPV /* 複合サービスにおけるユーザのサービスに対する評価値のリスト */
RV /* ユーザの複合サービスにおける予測評価値 */
$CIM \leftarrow MakeCIM(s)$
for all u_i in N **do**
    $CSIM \leftarrow CalculateSimilarity(CIM, u_i)$ /* 式 6 を用いて類似度を計算する */
    $CN \leftarrow SelectNearestUsers(CSIM)$ /* 式 7 を用いて類似ユーザ集合を作成する */
    $CPV \leftarrow CPV + PredictEvaluationValue(CSIM, CN, CIM) \times SIM(u_i)$ /* 式 8 を用いて評価値を予測する */
end for
$RV \leftarrow PredictCIMValue(SIM, CPV)$ /* 式 9 を用いて評価値を予測する */
Remove s from R
$MakeCompositeService(RV, R)$ /* この中で R が更新される */
if $\exists r_c \in R$ *s.t.* r_c is composite service **then**
    $R \leftarrow R \cup Multi\text{-}stage\ Collaborative\ Filtering(u,$
    $r_c, CN, CSIM)$
end if
return R

今回のサービス推薦においては，上記の評価値予測で得られた評価値の最も高い 1 つのサービスのみを推薦している．この推薦されたサービスが複合サービスであれば，この複合サービスの CIM を作成し CIM における協調フィルタリングへと移る．ユーザ u の複合サービス c を通じたサービス s の評価値を $v_{u,c}(s)$ と表現する．ここで注意していただきたのが，複合サービスの CIM において協調フィルタリングを適用するのは，1 回目の推薦の対象ユーザの類似ユーザであるという点である．複合サービス c の CIM におけるユーザ u_a とユーザ u_b の間の類似度関数は以下のようになる．

$$sim(u_a, u_b, c) = \frac{\sum_{s \in S}(v_{u_a,c}(s) - \overline{v_{u_a,c}})(v_{u_b,c}(s) - \overline{v_{u_b,c}})}{\sqrt{\sum_{s \in S}(v_{u_a,c}(s) - \overline{v_{u_a,c}})^2}\sqrt{\sum_{s \in S}(v_{u_b,c}(s) - \overline{v_{u_b,c}})^2}} \tag{5}$$

この式についても，式 2 と同様にユーザから呼び出されたサービスの数を考慮して重みを付ける．

$$sim'(u_a, u_b, c) = \frac{2 \times |S_{u_a,c} \cap S_{u_b,c}|}{|S_{u_a,c}| + |S_{u_b,c}|} sim(u_a, u_b, c) \tag{6}$$

下記の式を用いて複合サービス c の CIM におけるユーザ u の類似ユーザ集合を作成する．

$$N_c = \{u_a | Sim'(u_a, u, c) > 0, u_a \neq u\} \tag{7}$$

また下記の式を用いて類似ユーザの CIM におけるサービスの評価値を予測する．下記の式の N_c は，複合サービス c の CIM における類似ユーザ集合を表している．

$$pred(u, c, s) = \overline{v_u} + \frac{\sum_{u_n \in N_c} sim'(u, u_n, c) \times (v_{u_n,c}(s) - \overline{v_{u_n,c}})}{\sum_{u_n \in N_c} sim'(u, u_n, c)} \tag{8}$$

上記までの過程で求まった推薦対象ユーザの類似ユーザの複合サービス c の CIM における評価値と類似度を用いて以下のように推薦対象ユーザの複合サービス c におけるサービスへの評価値を予測する．上記の式と異なる点は，類似ユーザの集合が 1 度目の協調フィルタリングにおける類似ユーザ集合となっている点である．

$$pred(u, c, s) = \overline{v_u} + \frac{\sum_{u_n \in N} sim'(u, u_n) \times (v_{u_n,c}(s) - \overline{v_{u_n,c}})}{\sum_{u_n \in N} sim'(u, u_n)} \tag{9}$$

3.3 複合サービスの具象化

3.2 節では，前段階の協調フィルタリングにおける類似ユーザの情報をもとに複合サービスをもちいたことのない推薦対象ユーザの複合サービスを通じたサービスの評価値を予測する方法について説明した．本研究の目標は，利用情報に基づいて複合サービスが推薦された際にその複合サービスにさらに利用情報に基づいてサービスをバインドし，実行可能な状態で推薦することであるので，予測した評価値を用いて複合サービスを具象化する必要がある．推薦するアイテムの集合を作成する例には [13] がある．これは，各々のアイテムがコストを持ち，各々のユーザが予算を持つ状況において評価値を最大にするようにアイテムの集合を作成する問題をナップサック問題として定式化することで解決している．

本研究では，推薦アイテム集合，すなわち複合サービス推薦の際の構成サービス群を選択する際に言語グリッドの特有の制約を考慮しなければならない．言語グリッドに特有の性質には以下のものが挙げられる．

複合サービスにバインド可能なサービスタイプ

複合サービスは，抽象的なワークフロー，すなわち実行手順を示しているだけで各段階でどのサービスを実行するかを指定する必要がある．しかし，言語グリッドにおいては複合サービスにバインドできるサービスタイプが設定されている．例えば，辞書連携翻訳サービスにバインドするサービスは，機械翻訳サービス，対訳辞書サービス，形態素解析サービスの三つである．各々のサービスタイプに合致するサービスのみバインドすることが可能である．

ユーザの利用言語とサービスの対応言語

言語グリッドは，翻訳や辞書を中心とした言語という領域の問題解決を可能にするグリッドシステムである．言語グリッドのユーザは，各々母国語を持ち多くの場合，母国語とその他の言語の間での翻訳などを言語グリッドを用いて行う．対して，サービスにも扱う事のできる対応言語が存在する．ユーザの利用言語に対応したサービスのみを選択し推薦集合に加える必要が存在する．

問題の具体例を以下に示す．第一段階の協調フィルタリングを用いて，推薦対象ユーザに辞書連携翻訳サービスが推薦された場合を考える．辞書連携翻訳

にバインド可能なサービスタイプは以下の三つである．

- 機械翻訳
- 対訳辞書
- 形態素解析

また，辞書連携翻訳サービスの CIM において推薦対象ユーザの類似ユーザに対して協調フィルタリングを適用し，それを基に推薦対象ユーザの辞書連携翻訳を通じたサービスへの評価値を予測し，評価値の降順に並べた結果，以下のような結果が得られたとする．以下の，箇条書きの括弧内はそのサービスのサービスタイプを示している．

1. TGEver1(BilingualDictionaryWithLongestMatchSearch)
2. JServer(Translation)
3. Mecab(MorphologicalAnalysis)
4. BilingualDictionaryWithLongestMatchCrossSearch (BilingualDictionaryWithLongestMatchSearch)
5. StanfordPosTagger(MorphologicalAnalysis)
6. ICTCLAS(MorphologicalAnalysis)
7. …

このとき，単に評価値の高いものを複数選択するだけでは推薦する複合サービスの作成は出来ない．上記の制約を考慮して推薦するサービスの集合としての複合サービスを作成する必要がある．

上記の制約を考慮した集合の作成手順を自然言語で以下に示す．まず，多段階協調フィルタリングによって予測された推薦対象ユーザの推薦された複合サービスにおけるサービスを評価値の降順に並べる．評価値の高いサービスから順に ST 関数と L 関数を用いてサービスタイプと対応言語を調べ，複合サービスにバインド可能でユーザの利用言語に合致していれば推薦集合に加える．これを複合サービスの全ての複合サービスがバインドされるまで繰り返す．これを厳密に記述したものを Algorithm 2 に示す．

第4章　推薦システムのアーキテクチャ

第 3 章で説明したアルゴリズムに基づく，サービス推薦システムの実際の構成を説明する．このシステムは，大きく分けて四つのコンポーネントからなる．

Algorithm 2 複合サービス具象化アルゴリズム $MakeCompositeService(RV, R)$

```
1: r_c /* 推薦された複合サービス */
2: RV /* ユーザの複合サービスにおける予測評価値のマップ．キーはサービスで値は予測評価値である．また，要素は呼び出し回数順にソートされている． */
3: R /* 推薦されたサービス集合．t ∈ R */
4: ST() /* サービスタイプを返す関数 */
5: L() /* 対応言語を返す関数 */
6: while All component services are not binded to r_c do
7:     t ← GetNextKey(RV)
8:     if ST(t) can be binded to r_c AND L(t) is compatible with user then
9:         if ST(t) of r_c is empty then
10:            R ← t
11:        end if
12:    end if
13: end while
```

一つ目は推薦の実行を担う推薦制御部，二つ目は実際の言語グリッドのログデータを解析して推薦システムに適した形にデータを整形するデータ整形部，三つ目は複合サービスの CIM における推薦対象ユーザの評価値を予測し求める評価値予測部，最後に予測された CIM における評価値をもとに推薦する複合サービスを作成する複合サービス作成部である．協調フィルタリングの基本的な計算部分や一段階目の協調フィルタリングの実装には，Lenskit ライブラリ [15] を用いた．システムの全体の構成図を図 4 に示す．

ここでの，ユーザインタフェースとはユーザが結果を確認するためのインタフェースを指す．推薦は，ユーザの呼び出し履歴のログデータを用いて行われるので今回の推薦システムにおいてユーザとシステムの直接的なインタラクションは存在しない．以下で，推薦制御部，データ整形部，評価値予測部，複合サービス作成部それぞれの詳細を述べる．

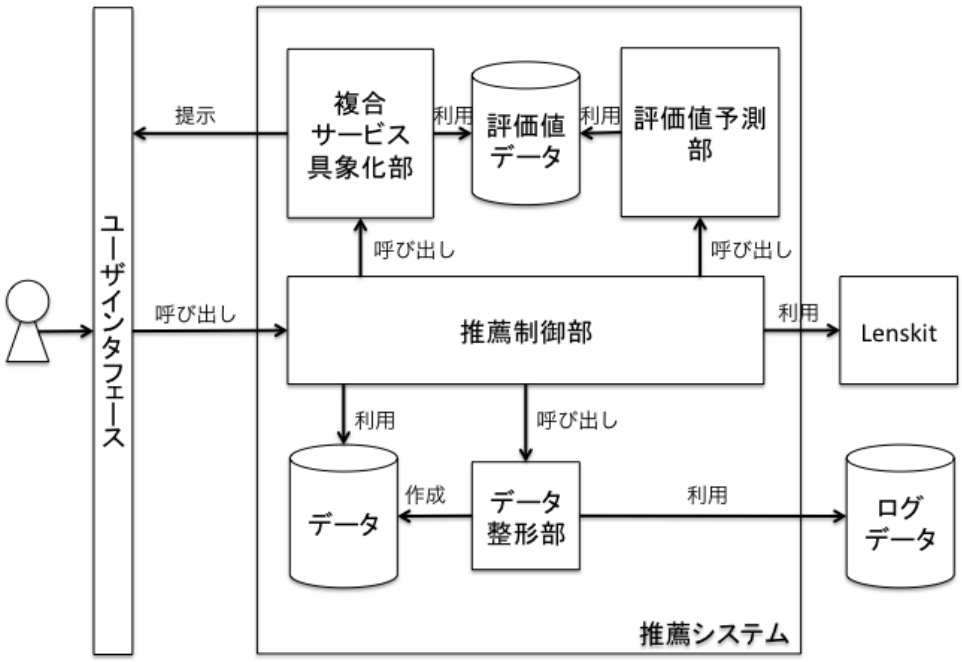

図 4: 推薦システムのアーキテクチャ

4.1 推薦制御部

推薦制御部は，3.1 節で説明した多段階協調フィルタリングを用いて，ユーザのサービス呼び出し履歴から推薦結果を作成するというように推薦の実行全体を制御する.

推薦制御部は，データ整形部によって作成されたユーザ，サービス，複合サービスのデータを読み込み Lenskit ライブラリのコンポーネントを用いて第 1 段階目の推薦を実行し，推薦結果データを作成する．それを基に各ユーザに推薦されたサービスが複合サービスであるかを確かめ次の段階の協調フィルタリングの対象ユーザのリストを作成する．このユーザが存在しなければ処理を終了し，存在すればこれを用いて各ユーザごとの類似ユーザのリストを作成する．各推薦対象ユーザごとの類似ユーザに対して Leskit ライブラリのコンポーネントを用いて，次の段階の協調フィルタリングを行い類似ユーザの複合サービスにおける評価値データを作成する．そのデータと類似ユーザとの類似度データを評価値予測部に渡し，推薦対象ユーザの複合サービスにおける評価値データを受

け取る．この受け取った結果を複合サービスにおける評価値データに追記する．続いてそのデータを複合サービス具象化部に渡し，推薦する複合サービスの構成サービスのデータを受け取る．これを複合サービスの推薦対象ユーザごとに行う．このユーザごとの推薦結果データを用いて再び複合サービスが含まれているかを確認し含まれていれば上記の処理をもう一度繰り返し，含まれていなければ処理を終了する．

4.2 データ整形部

データ整形部は，推薦実行の前処理として実行される言語グリッドの実際のログデータを基に推薦エンジンに適した形にデータを整形するモジュールである．

言語グリッドのログデータは，ユーザ ID，タイムスタンプ，サービス ID，複合サービスのバインディング情報の 4 つ組が 1 行で 1 回の呼び出しを表現している．このデータから協調フィルタリングに用いるための評価値行列の形式，すなわち，ユーザ ID，サービス ID，呼び出し回数という以下のような形式に変換する必要が生じる．

```
"UserName","LsdDb","2712"
"UserName","TranslationWithDictionary","309"
"UserName","JServer","301"
"UserName","Mecab","309"
```

さらに，複合サービス呼び出し情報は，以下のような JSON 形式で格納されている．

```
[
  {// バインディングするサービスの情報
      "children": [],
      "faultCode": null,
      "faultString": null,
      "invocationName": "MorphologicalAnalysisPL",
      "responseTimeMillis": 133,
      "serviceCopyright": "",
      "serviceId": "Mecab",
      "serviceLicense": "http://mecab.sourceforge.net/",
      "serviceName": "MeCab"
```

```
  },
  ...
]
```

そのため，これをパースしてサービス ID を取り出しその呼び出し回数をユーザごとにカウントし，複合サービスが推薦された際の第 2 段階目以降の協調フィルタリングに用いるための評価値行列の形に変換する．このデータを複合サービスごとに作成する．

4.3 評価値予測部

評価値予測部は，3.2 節で説明した評価値予測手法を用いて，複合サービスの評価値情報から推薦対象ユーザの類似ユーザの評価値予測を行った後に，類似ユーザとの類似度を用いて推薦対象ユーザの複合サービス通じた呼び出し回数の予測値を算出する．

評価値予測部は，推薦対象ユーザの類似ユーザの複合サービスにおける評価値データと類似ユーザとの類似度データを受け取り，推薦対象ユーザの複合サービスにおける評価値を予測する．具体的には，各データを用いて数式 9 の値を算出する．算出された値は各ユーザごとにベクトル形式のデータに格納される．

4.4 複合サービス具象化部

複合サービス具象化部は，3.3 節で説明した複合サービス作成手法を用いて，ユーザに推薦された複合サービスにバインディングする複合サービスを決定する．

複合サービス具象化部は，評価値予測部で作成されたユーザごとの評価値ベクトルを受け取り，その要素を評価値の降順にソートする．

さらに，複合サービスにバインディングできるサービスタイプのデータを用いて，複合サービスの制約を満たしたユーザにとって評価値が最大となるように構成サービスを決定し，構成サービス集合としての複合サービスをリスト形式で作成し，これを推薦制御部に返す．

第 5 章　評価

第 4 章で説明したシステムを用いて第 3 章 で説明した多段階協調フィルタリングを実世界のデータセットに適用する．実行には言語グリッドの 3 年分の実

際のログデータを用いる．本章では，まず言語グリッドについて説明し，評価手法について目的とともに説明する．その後, 言語グリッドのログデータに推薦アルゴリズムを適用した結果を示し，その結果について考察する．

5.1 言語グリッド

言語グリッドとは,「辞書や機械翻訳などの言語資源を言語サービスとして登録し，共有可能にするインターネット上の多言語サービス基盤」[2] を指す．サービスコンピューティングにおいては，ラッピングされた言語サービスがそれぞれ Web サービスのコンポーネントとなっており，実際の Web ツールとしての利用法を公開している．これらのコンポーネントを組み合わせることにより,「農業領域に特化した日英翻訳サービス」などの複合サービスを負担を抑えて作成することができる．

5.2 評価手法

提案手法を用いた推薦システムを以下に示す観点から評価する．なお，評価には言語グリッドの実際のログデータ 3 年分を用いる．

評価は以下のそれぞれの手法に対して，評価値予測精度と複合サービスの推薦実行可能性の観点からの評価を行う．

1. 提案手法
   これまでの章を用いて説明してきた多段階協調フィルタリングを用いた推薦システムを評価対象として用いる．
2. 比較手法 1
   提案手法と同様に多段階に協調フィルタリングを適用するが，毎回の入力行列を同様のものにする．この行列はどの複合サービスを通じてサービスを呼び出したかを区別せずにユーザがどのサービスを何回呼び出したかという情報のみを用いて作成される．これによって複合サービス呼び出しにおける文脈を考慮しない多段階協調フィルタリングとの比較を行うことができ，呼び出しにおける文脈を考慮することの効用が検証できる．
3. 比較手法 2
   評価値行列を作成する際に同様の複合サービスでも構成サービスが異なるものを別のものとみなして評価値行列を作成する．例えば，農業用語辞書サービスと Google 翻訳と Mecab 形態素解析サービスを組み合わせて作成

した辞書連携翻訳と，医療用語辞書サービスとGoogle翻訳とMecab形態素解析サービスを組み合わせて作成した辞書連携翻訳を別のものとみなして評価値行列を作成するということである．その行列を用いて推薦を実行する．これによって抽象ワークフローとしての複合サービスの利用分布を考慮しない手法との比較を行う事ができ，複合サービスを抽象ワークフローとして扱い，推薦を多段階に適用することの効用が検証できる．

上記で説明した手法それぞれに対して，以下に示す指標を用いて評価値予測精度と複合サービスの推薦実行可能性の測定を行う．

1. 評価値予測精度

   推薦システムの評価値予測精度を平均絶対誤差 (*mean absolute error; MAE*) を用いて評価する．この計算手法を以下に示す．

$$MAE = \frac{\sum_{u \in U} \sum_{i \in testset_u} |rec(u,i) - r_{u,i}|}{\sum_{u \in U} |testset_u|} \tag{10}$$

   MAEは評価を行った全てのユーザ $u \in U$ とテストデータセット $(testset_u)$ 内の全てのアイテムに対して，計算された推薦スコア $(rec(u,i))$ と実際の評価値 $(r_{u,i})$ の間の偏差の平均を計算する．すなわち，この値が小さいほど精度の高い評価値予測が行えていることになる．提案手法と比較手法1の値を比較することにより，複合サービスにおける呼び出し関係を考慮することの有効性を示す．

2. 複合サービスの推薦実行可能性

   複合サービスの推薦実行可能性を推薦リストにおける複合サービスの被覆率を用いて測定する．被覆率は，ユーザの母集団のどのくらいに対して推薦を実行できるかという能力を異なるシステム間で比較する指標の1つである．これは，推薦システムがどのくらい疎な入力行列にまで推薦を実行できるかを分析する際に非常に有用な指標である．ユーザに対する複合サービスの被覆率は以下の式を用いて算出できる．

$$Ccov = \frac{\sum_{u \in U} \rho_u}{|U|} \tag{11}$$

$$\rho_u = \frac{|recset_u \cap C|}{|recset_u|} \tag{12}$$

   これはユーザの推薦リストにおける複合サービスの占有率を測定するもの

である．この指標を用いて提案手法を用いたシステムと比較手法 2 の比較を行い，複合サービス推薦の際に，階層ごとに協調フィルタリングを適用することの有効性を示す．

5.3 結果

5.2 節で説明した評価手法を用いて提案手法の評価を行った結果を以下に示す．

1. 提案手法と比較手法の評価値予測精度
   提案手法を用いた推薦システムの評価値予測精度を平均絶対誤差 (*mean absolute error; MAE*) を用いて評価した結果を以下の表 4 に示す．

表 4: MAE を用いた提案手法の評価結果

	2011 年	2012 年	2013 年	平均
提案手法	1158	5003	14206	6789
比較手法 1	2044	5965	23256	15761
比較手法 2	851	2709	7606	3722

2. 提案手法と比較手法の複合サービスの被覆率の比較
   提案手法を用いた推薦システムと組み合わせ方の異なる複合サービスを別のサービスとみなして単純に協調フィルタリングを適用する比較手法を用いた推薦システムに対して被覆率を算出した結果を以下の表 5 に示す．

表 5: 被覆率を用いた提案手法と比較手法の比較結果

	2011 年	2012 年	2013 年	平均
提案手法	0.148	0.0	0.230	0.126
比較手法 1	0.0	0.0	0.0	0.0
比較手法 2	0.56	0.0	0.0	0.187

5.4 考察

本節では，5.3 節で提示した結果を受けて提案手法の評価値予測精度とサービス推薦の実行可能性に関して考察を与えた後，推薦結果に対して定性的な評価

を与える．まず，評価結果に対する考察に関して述べる．

1. 評価値予測精度

   表 4 の結果より，提案手法を用いた推薦システムの方が比較手法 1 より評価値予測精度が高いことが見て取れる．これは，協調フィルタリングを各段階ごとに多段階に適用することで複合サービスにおける一種の呼び出し文脈を考慮できるようになったことが原因であると考えられる．呼び出し文脈を考慮することで，その文脈においてより適切な組み合わせを推薦することが可能となったために評価値の予測精度が向上したと考えられる．

2. 推薦実行可能性

   表 5 の結果より，提案手法を用いた推薦システムの方が被覆率が低い事が見て取れる．これは，提案手法を用いたほうが複合サービスを含めたサービスの推薦実行可能性が低いことを示している．これは呼び出し文脈を導入することで，組み合わせが異なっていても組み合わせ手順にユーザ間で共通項を持つことができ，原子サービスと抽象ワークフローの利用履歴に基づいてより正確な類似ユーザを判定することが可能になる．しかし，やはり類似ユーザが複合サービスを用いていなければ複合サービスは推薦されないために以上の結果になったと考えられる．

続いて，推薦結果に対する定性的な評価を与える．以上までで評価値の予測精度と推薦実行可能性に関して定量的な評価を行い，それについて考察を与えたが推薦されたサービスが本当にユーザにとって有用なものであるかという点に関しても考慮しなければならない．

複合サービスはドメイン非依存の抽象ワークフローであるが，原子サービスをバインディングすることによってドメイン依存となる．このことから推薦結果の適切性をユーザの利用ドメインと推薦されたサービスのドメインを比較することによって測ることができるが，今回の推薦の実行結果ではデータ量の不足も原因の 1 つと考えられるが適切な推薦がなされているとは言いがたい．

同じドメインのサービスを利用しているユーザが存在すれば，そのユーザの利用情報に含まれるドメインを特徴付けるサービスの呼び出し回数によって自分のドメインに適用できるサービスの発見が可能となる．しかし，そのようなユーザが存在せず一般的に用いられている機械翻訳などによって類似ユーザが決定された場合は，自分が利用していないだけで一般的に用いられているサービスが推薦され，ユーザにとって有用でない推薦結果となってしまう．これが

上記の結果の原因であると考える．

以上のことから，以下の結論が導ける．提案手法は，複合サービス推薦において評価値予測精度の観点からは有用である．また，呼び出し文脈を考慮したより正確な類似度を与える事ができるが複合サービスの推薦実行可能性を向上するわけではない．さらに，ユーザにとって有用なサービスを推薦できているかどうかという点に関しては改善の余地が存在する．

第6章　おわりに

サービスコンピューティングが今後さらに発展していくために，ユーザへのサービスの候補の提示や複合サービスの構成サービスを提示する手法は必要不可欠である．今後，サービスや計算資源の増加が見込まれる状況において，代替サービス候補の存在や出現を知る方法がなければ，サービスの質の向上の機会を損失してしまう可能性がある．

本研究では，従来の厳密なQoS評価値の代わりに，サービスの呼び出し回数を評価値として用いた協調フィルタリングによるサービス推薦手法を提案した．そして，サービス推薦における複合サービスの存在と複合サービスを作成する際の制約の両者に対応する手法として呼び出し回数を評価値として用いた多段階協調フィルタリングが有効に作用することを示した．

ユーザのサービスに対する評価値として，ユーザからの明示的な評価が必要なQoS値でなくサービスの呼び出し回数を用いることでユーザにサービスの評価の負担を強いる事なくサービス推薦を実現することができる．通常のユーザベースの協調フィルタリングを適用しサービスを推薦した．その結果，複合サービスが推薦された場合には，その複合サービスを通じた呼び出し情報を抽出し，それと類似ユーザを用いて複合サービスを通じたサービスの呼び出し回数の予測を行った．そこで得られた評価値を用いて，サービスの制約を考慮した推薦複合サービスの作成を行った．提案手法を用いた推薦システムを用いて評価値予測精度と推薦実行可能率の観点から評価を行い，提案手法の有効性を検証した．

本研究の貢献は以下の2点である．

推薦対象ユーザの複合サービスを通じた評価値の予測手法

協調フィルタリングを多段階に適用することにより複合サービスの呼び出しの文脈を考慮した評価値の予測を行った．その結果，文脈を考慮しない

多段階の協調フィルタリングの約 2 倍の評価値予測精度 (MAE 値) を得た.

サービスの制約を考慮した複合サービス作成手法

サービスの制約を考慮し，サービスの集合として複合サービス推薦を行った．その結果，組み合わせ方ごとに区別した複合サービスを通常の協調フィルタリングを用いて推薦する手法より複合サービスの推薦実行可能性が向上するわけではないという結果を得た.

今後，実世界において実際に多段階協調フィルタリングによる複合サービスの構成サービスを考慮したサービス推薦手法を適用させることを考えた場合，呼び出し回数以外の評価値を用いた場合の推薦結果との比較が必要となる．呼び出し回数を評価値として用いた場合，ユーザの嗜好や目的はある程度呼び出し回数に現れるが，やはりユーザによる明示的なサービス評価比べると精度が落ちることが予想されるため，どれくらいユーザの嗜好・目的に沿ったサービス推薦を行うことが出来ているのかを調査する必要がある．また，この研究の拡張として，ユーザの評価値の正規化手法や複合サービスのコールツリーを考慮した類似度計算手法が考えられる．呼び出し回数を評価値として用いる利点として，ユーザに明示的な評価を強制しない点が挙げられるので，正規化による推薦精度の向上は重要である．また，複合サービスの呼び出し方を考慮した類似度計算が可能になればより正確な推薦の実行が可能となるであろう.

謝辞

本研究を行うにあたり，熱心なご指導，ご助言を賜りました石田亨教授に深謝申しあげます．また，日頃より時間を惜しまず研究内容に関して様々なご助言とご協力をいただきました村上陽平特定研究員に深謝申しあげます．最後に，普段からお世話になっている石田・松原研究室の皆様，特に研究にご協力いただいた Trang Mai Xuan さん，Kemas Muslim Lhaksmana さんを中心に言語グリッドプロジェクト運営チームメンバーの皆様に心より感謝いたします.

参考文献

[1] Zhang, L.-J., Zhang, J. and Cai, H.: *Services computing*, Springer (2007).

[2] Ishida, T.: *The language grid*, Springer (2011).

[3] Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J. and

Chang, H.: QoS-Aware Middleware for Web Services Composition, *IEEE Trans. Softw. Eng.*, Vol. 30, No. 5, pp. 311–327 (2004).

[4] Herlocker, J. L., Konstan, J. A., Borchers, A. and Riedl, J.: An Algorithmic Framework for Performing Collaborative Filtering, *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, New York, NY, USA, ACM, pp. 230–237 (1999).

[5] Manikrao, U. S. and Prabhakar, T. V.: Dynamic Selection of Web Services with Recommendation System, *Proceedings of the International Conference on Next Generation Web Services Practices*, NWESP '05, Washington, DC, USA, IEEE Computer Society, pp. 117–121 (2005).

[6] Lin, Y., Luo, S. and Han, B.: A novel grid resource matching scheme, *2010 Sixth International Conference on Natural Computation (ICNC)*, Vol. 7, pp. 3743–3747 (2010).

[7] Nanopoulos, A.: Item Recommendation in Collaborative Tagging Systems, *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, Vol. 41, No. 4, pp. 760–771 (2011).

[8] You, Z., Sun, Y., Chen, Y., Zhang, Y. and Zhu, Y.: The Intelligent Recommendation System Based on Amended Rating Matrix in TTP, *The Sixth World Congress on Intelligent Control and Automation, 2006. WCICA 2006.*, Vol. 1, pp. 4302–4306 (2006).

[9] Sobecki, J. and Piwowar, K.: Comparison of Different Recommendation Methods for an e-Commerce Application, *First Asian Conference on Intelligent Information and Database Systems, 2009. ACIIDS 2009.*, pp. 127–131 (2009).

[10] Tuan, C.-C., Hung, C.-F. and Tseng, K.-W.: A Relational Compound Collaborative Filtering Recommendation System, *Broadband and Wireless Computing, Communication and Applications (BWCCA), 2011 International Conference on*, pp. 411–415 (2011).

[11] Adomavicius, G. and Kwon, Y.: Improving Aggregate Recommendation Diversity Using Ranking-Based Techniques, *IEEE Trans. Knowl. Data Eng.*, Vol. 24, No. 5, pp. 896–911 (2012).

[12] Zheng, Z., Ma, H., Lyu, M. R. and King, I.: QoS-Aware Web Service Recommendation by Collaborative Filtering, *IEEE Trans. Serv. Comput.*, Vol. 4, No. 2, pp. 140–152 (2011).

[13] Xie, M., Lakshmanan, L. V. and Wood, P. T.: Breaking out of the Box of Recommendations: From Items to Packages, *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, New York, NY, USA, ACM, pp. 151–158 (2010).

[14] McLaughlin, M. R. and Herlocker, J. L.: A Collaborative Filtering Algorithm and Evaluation Metric That Accurately Model the User Experience, *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, New York, NY, USA, ACM, pp. 329–336 (2004).

[15] Ekstrand, M. D., Ludwig, M., Konstan, J. A. and Riedl, J. T.: Rethinking the Recommender Research Ecosystem: Reproducibility, Openness, and LensKit, *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, New York, NY, USA, ACM, pp. 133–140 (2011).

付録

A.1 推薦システムのソースコード

ソースコード 1: アプリケーションの起動部

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import java.util.Arrays;
import java.util.Iterator;
import java.util.List;

public class App {
        public static void main(String[] args) {

                List<String> source_list = Arrays.asList("
                  accesslog_5c_2011.csv",
                                "accesslog_5c_2012.csv", "
                                  accesslog_5c_2013.csv");

                Iterator<String> source_itr = source_list.iterator();
                while (source_itr.hasNext()) {
                        String source = source_itr.next();

                        RecController controller = new RecController(
                          source);
                        controller.start();
                }
        }
}
```

ソースコード 2: 推薦を実行するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import it.unimi.dsi.fastutil.longs.LongArrayList;
import it.unimi.dsi.fastutil.longs.LongListIterator;
import it.unimi.dsi.fastutil.longs.LongSet;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.grouplens.lenskit.data.dao.ItemDAO;
import org.grouplens.lenskit.data.dao.UserDAO;
import org.grouplens.lenskit.data.dao.UserEventDAO;
import org.grouplens.lenskit.data.event.Event;
import org.grouplens.lenskit.data.history.
  RatingVectorUserHistorySummarizer;
import org.grouplens.lenskit.data.history.UserHistory;
import org.grouplens.lenskit.data.history.UserHistorySummarizer;
import org.grouplens.lenskit.scored.ScoredId;
import org.grouplens.lenskit.vectors.MutableSparseVector;
import org.grouplens.lenskit.vectors.SparseVector;
import org.grouplens.lenskit.vectors.VectorEntry;
import au.com.bytecode.opencsv.CSVWriter;

/**
 * 推薦の状態を示すもの.
 *
 * @author taketo957
 *
 */
enum Status {
        INIT, ATOM, COMP
}

/**
 * アプリケーション全体の動作を制御する
 *
 * @author taketo957
 *
 */
public class RecController {
        private String MATRIX_FILENAME;
        private LenskitComponentProvider lcp;
        private String FILENAME;

        /**
         * コンストラクタ
         *
```

```
 * @param filename
 */
public RecController(String filename) {
        this.FILENAME = filename;
        this.MATRIX_FILENAME = "data/user_service_matrix
          /" + filename;
        new HashMap<Integer, List<ScoredId>>();
        lcp = new LenskitComponentProvider(
          MATRIX_FILENAME);
}

/**
 * システムの動作を開始するメソッド
 */
public void start() {
        UserListHolder userlist = new UserListHolder(FILENAME
          );
        ServiceListHolder servicelist = new ServiceListHolder(
          FILENAME);
        ServiceTypeHolder service_type_list = new
          ServiceTypeHolder();
        BindingInfoHolder binding_info = new BindingInfoHolder();
        CompositeServiceListHolder composite_service_list = new
          CompositeServiceListHolder(
                        FILENAME);
        int depth = 0;

        List<String> recommended_services;

        for (int i = 0; i < userlist.size(); i++) {
                Status s = Status.INIT;
                recommended_services = new ArrayList<String>();
                List<String> service_path = new ArrayList<String
                  >();

                multiStageCollaborativeFiltering((long) i,
                  recommended_services,
                                lcp, userlist, servicelist,
                                  composite_service_list,
                                service_path, s, service_type_list,
                                  binding_info, depth);
        }
}
```

```
@SuppressWarnings("resource")
private void multiStageCollaborativeFiltering(long i,
                List<String> recommended_services,
                  LenskitComponentProvider lcp2,
                UserListHolder userlist, ServiceListHolder servicelist
                  ,
                CompositeServiceListHolder composite_service_list,
                List<String> service_path, Status s,
                ServiceTypeHolder service_type_list,
                BindingInfoHolder binding_info, int depth) {

        CSVWriter writer = null;
        try {
                writer = new CSVWriter(new FileWriter("data/
                  rec_result_mscf/stage"
                                + depth + "_for_user" + i + "_"
                                  + FILENAME));
        } catch (IOException e) {
                e.printStackTrace();
        }

        DAOInterface pre_daoi = lcp2.getDao();
        NeighborhoodProvider pre_np = new NeighborhoodProvider
          (pre_daoi);
        Map<Long, Double> pre_neighbors = pre_np.
          findNeighborUsers((int) i);


        if (s == Status.COMP) {
                lcp2 = renewDataSource(service_path);
        } else if (s == Status.INIT) {

        } else {
                return;
        }

        DAOInterface daoi = lcp2.getDao();
        ItemDAO idao = daoi.getIdao();
        UserDAO udao = daoi.getUdao();
        UserEventDAO uedao = daoi.getUedao();

        LongSet items = idao.getItemIds();
```

```
MissingValuePredictor mvp = new MissingValuePredictor(
  pre_daoi, daoi);

SparseVector pred_vec = null;

if (matrixHasUser(i, udao)) {
        NeighborhoodProvider np = new
          NeighborhoodProvider(daoi);
        Map<Long, Double> neighbors = np.
          findNeighborUsers((int) i);

        writeOutSimData(i, userlist, depth, neighbors);

        UserHistory<Event> measured_history = uedao.
          getEventsForUser(i);
        UserHistorySummarizer measured_summarizer =
          new RatingVectorUserHistorySummarizer();
        SparseVector measured_items =
          measured_summarizer
                        .summarize(measured_history);
        LongSet measured_items_keys = measured_items.
          keySet();

        pred_vec = mvp.predictMissingValues(i, neighbors,
          items);

        MutableSparseVector tmp_vec = pred_vec.
          mutableCopy();
        MutableSparseVector eval_vec = tmp_vec
                        .withDomain(measured_items_keys);
        tmp_vec.set(measured_items);

        pred_vec = tmp_vec;

        writeOutPredVector(i, userlist, servicelist, depth,
          eval_vec);

} else {
        pred_vec = mvp.predictMissingCIMValues(i,
          pre_neighbors);
        renewTables(i, service_path, pred_vec);
        lcp2 = reloadDataSource(service_path);
```

```
}


if (s == Status.INIT) {
        LongArrayList sorted_items = pred_vec.keysByValue
          (true);
        if (sorted_items.size() == 0) {
                return;
        }
        long rec_service = sorted_items.getLong(0);
        String rec_service_name = servicelist.getServiceName
          (Long
                        .toString(rec_service));
        if (Double.isNaN(pred_vec.get(rec_service))) {
                return;
        }

        recommended_services.add(rec_service_name);

        writer.writeNext(new String[] {
                        userlist.getUserName(Long.toString
                          (i)),
                        servicelist.getServiceName(Long.
                          toString(rec_service)),
                        Double.toString(pred_vec.get(
                          rec_service)) });
        try {
                writer.flush();
        } catch (IOException e) {
                e.printStackTrace();
        }

} else if (s == Status.COMP) {
        String comp_service = service_path.get(service_path.
          size() - 1);

        List<String> component_service_types =
          binding_info
                        .findComponetServiceTypes(
                          comp_service);

        Map<String, String> binded_services = new
          HashMap<String, String>();
```

```
Iterator<String> type_itr =
  component_service_types.iterator();

while (type_itr.hasNext()) {
        String type = type_itr.next();
        binded_services.put(type, "");
}

LongArrayList sorted_items = pred_vec.keysByValue
  (true);

LongListIterator pred_vec_itr = sorted_items.iterator
  ();

while (pred_vec_itr.hasNext()) {
        long service = pred_vec_itr.nextLong();
        String service_name = servicelist.
          getServiceName(Long
                        .toString(service));
        String service_type = service_type_list
                        .findServiceType(
                          service_name);

        if (binded_services.containsKey(service_type)
                        && binded_services.get(
                          service_type).equals(""))
                          {
                binded_services.put(service_type,
                  service_name);
                recommended_services.add(
                  service_name);
                writer.writeNext(new String[] {
                                userlist.
                                  getUserName(
                                  Long.toString(i)),
                                servicelist.
                                  getServiceName(
                                  Long.toString(
                                  service)),
                                Double.toString(
                                  pred_vec.get(
                                  service)) });
```

```
                                        try {
                                                writer.flush();
                                        } catch (IOException e) {
                                                e.printStackTrace();
                                        }
                                }

                        }

                        recommended_services.remove(comp_service);

                } else {
                }

                if (recommeded_services_contains_composite_services(
                                recommended_services,
                                  composite_service_list, service_path)) {
                        s = Status.COMP;
                } else {
                        s = Status.ATOM;
                }

                depth++;

                multiStageCollaborativeFiltering(i, recommended_services,
                  lcp2,
                                userlist, servicelist, composite_service_list,
                                  service_path, s,
                                service_type_list, binding_info, depth);

                return;

}

private void writeOutSimData(long i, UserListHolder userlist, int
  depth,
                Map<Long, Double> neighbors) {
        CSVWriter sim_writer = null;

        try {
                sim_writer = new CSVWriter(new FileWriter("
                  data/sim_data/user"
```

```
                                + i + "_depth" + depth + "_" +
                                  FILENAME));
                } catch (IOException e2) {
                        e2.printStackTrace();
                }

                for (Entry<Long, Double> entry : neighbors.entrySet()) {
                        sim_writer.writeNext(new String[] {
                                        userlist.getUserName(Long.toString
                                          (i)),
                                        userlist.getUserName(Long.toString
                                          (entry.getKey())),
                                        Double.toString(entry.getValue())
                                          });
                        try {
                                sim_writer.flush();
                        } catch (IOException e1) {
                                e1.printStackTrace();
                        }
                }
        }

        private void writeOutPredVector(long i, UserListHolder userlist,
                        ServiceListHolder servicelist, int depth,
                        MutableSparseVector eval_vec) {
                CSVWriter vec_writer = null;
                try {
                        vec_writer = new CSVWriter(new FileWriter(
                                        "data/vec_data/pred_stage" +
                                          depth + "_for_user" + i
                                                        + "_" +
                                                          FILENAME));
                } catch (IOException e) {
                        e.printStackTrace();
                }

                for (VectorEntry e : eval_vec) {
                        vec_writer.writeNext(new String[] {
                                        userlist.getUserName(Long.toString
                                          (i)),
                                        servicelist.getServiceName(Long.
                                          toString(e.getKey())),
                                        Double.toString(e.getValue()) });
```

```
                try {
                        vec_writer.flush();
                } catch (IOException e1) {
                        e1.printStackTrace();
                }
        }
}

private LenskitComponentProvider renewDataSource(List<String>
   service_path) {
        LenskitComponentProvider lcp2;
        String tmp_service_path = "";

        for (int j = 0; j < service_path.size(); j++) {
                String service = service_path.get(j);
                tmp_service_path = tmp_service_path + "2" +
                  service;
        }

        tmp_service_path = tmp_service_path.substring(1,
                        tmp_service_path.length());

        lcp2 = new LenskitComponentProvider("data/tmp/" +
          tmp_service_path
                        + "_" + FILENAME);
        return lcp2;
}

private LenskitComponentProvider reloadDataSource(List<String>
   service_path) {
        LenskitComponentProvider lcp2;
        String tmp_service_path = "";

        for (int j = 0; j < service_path.size(); j++) {
                String service = service_path.get(j);
                tmp_service_path = tmp_service_path + "2" +
                  service;
        }

        tmp_service_path = tmp_service_path.substring(1,
                        tmp_service_path.length());
```

```
        lcp2 = new LenskitComponentProvider("data/tmp/" +
          tmp_service_path
                        + "_" + FILENAME);
        return lcp2;
}

private void renewTables(long i, List<String> service_path,
                SparseVector pred_vec) {

        String tmp_service_path = "";

        for (int j = 0; j < service_path.size(); j++) {
                String service = service_path.get(j);
                tmp_service_path = tmp_service_path + "2" +
                  service;
        }

        tmp_service_path = tmp_service_path.substring(1,
                        tmp_service_path.length());

        File csv = new File("data/tmp/" + tmp_service_path + "_
          " + FILENAME);
        BufferedWriter bw = null;
        try {
                bw = new BufferedWriter(new FileWriter(csv, true
                  ));
        } catch (IOException e) {
                e.printStackTrace();
        }

        for (VectorEntry e : pred_vec) {
                try {
                        bw.write(i + "," + (int) e.getKey() + ","
                          + (int) e.getValue());
                        bw.newLine();
                } catch (IOException e1) {
                        e1.printStackTrace();
                }
        }
        try {
                bw.close();
        } catch (IOException e1) {
                e1.printStackTrace();
```

```
        }
}

/**
 * 指定したユーザが評価値行列に存在するか確認する.
 *
 * @param i
 * @param udao
 * @return
 */
private boolean matrixHasUser(long i, UserDAO udao) {
        LongSet users = udao.getUserIds();
        if (users.contains(i)) {
                return true;
        }
        return false;
}

/**
 * 多段階協調フィルタリングの終了条件のチェック
 *
 * @param recommended_services
 * @param composite_service_list
 * @param service_path
 * @return
 */
private boolean recommeded_services_contains_composite_services(
                List<String> recommended_services,
                CompositeServiceListHolder composite_service_list,
                List<String> service_path) {

        boolean result = false;
        Iterator<String> itr = recommended_services.iterator();

        while (itr.hasNext()) {
                String service_name = itr.next();
                if (composite_service_list.isCompositeService(
                  service_name)) {
                        result = true;
                        service_path.add(service_name);
                }
        }
        return result;
```

```
        }
}
```

ソースコード 3: サービスごとのバインディング情報を保持するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import au.com.bytecode.opencsv.CSVReader;

public class BindingInfoHolder {
        /**
         * コンストラクタ
         */
        public BindingInfoHolder() {

        }

        public List<String> findComponetServiceTypes(String service) {
                CSVReader reader = makeReader();
                String[] nextline;
                List<String> result = new ArrayList<String>();
                try {
                        while ((nextline = reader.readNext()) != null) {
                                String service_name = nextline[0];
                                if (!service_name.equals(service)) {
                                        continue;
                                }
                                int size = nextline.length;
                                List<String> service_types = new
                                  ArrayList<String>(size - 1);
                                for (int i = 0; i < size - 1; i++) {
                                        service_types.add(i, nextline[i + 1]);
                                }
                                result = service_types;
                        }
                } catch (IOException e) {
                        // TODO Auto-generated catch block
```

```
                e.printStackTrace();
            }
            return result;
        }

        private CSVReader makeReader() {
            CSVReader reader = null;
            try {
                reader = new CSVReader(new FileReader(
                        "data/binding_info/
                          binding_info.csv"));
            } catch (FileNotFoundException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
            return reader;
        }
}
```

ソースコード 4: 複合サービス一覧を保持するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import au.com.bytecode.opencsv.CSVReader;

/**
 * 複合サービスの一覧を保持するクラス
 *
 * @author taketo957
 *
 */
public class CompositeServiceListHolder {
        private List<String> servicelist;
        private String FILENAME;

        /**
         * コンストラクタ ファイルを読み込んでリストの作成を行う.
         */
```

```
public CompositeServiceListHolder(String filename) {
        FILENAME = "data/composite_service_list/" +
          filename;
        CSVReader reader = null;
        servicelist = new ArrayList<String>();
        try {
                reader = new CSVReader(new FileReader(
                  FILENAME));
        } catch (FileNotFoundException e) {
                System.err.println(e);
        }
        try {
                makeServiceList(reader);
        } catch (IOException e) {
                e.printStackTrace();
        }
}

/**
 * 引数で指定したサービスが複合サービスかどうかを確かめる
 *
 * @param service_name
 * @return
 */
public boolean isCompositeService(String service_name) {
        if (servicelist.contains(service_name)) {
                return true;
        }
        return false;
}

/**
 * リーダから情報を読み込んでサービスリストを作成する
 *
 * @param reader
 * @throws IOException
 */
private void makeServiceList(CSVReader reader) throws
  IOException {
        String[] nextline;
        while ((nextline = reader.readNext()) != null) {
                String service_name = nextline[0];
                servicelist.add(service_name);
```

```
                }
        }
}
```

ソースコード 5: データファイルとのインタフェースを提供するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import java.io.File;
import org.grouplens.lenskit.data.dao.EventDAO;
import org.grouplens.lenskit.data.dao.ItemDAO;
import org.grouplens.lenskit.data.dao.ItemEventDAO;
import org.grouplens.lenskit.data.dao.PrefetchingItemDAO;
import org.grouplens.lenskit.data.dao.PrefetchingItemEventDAO;
import org.grouplens.lenskit.data.dao.PrefetchingUserDAO;
import org.grouplens.lenskit.data.dao.PrefetchingUserEventDAO;
import org.grouplens.lenskit.data.dao.SimpleFileRatingDAO;
import org.grouplens.lenskit.data.dao.UserDAO;
import org.grouplens.lenskit.data.dao.UserEventDAO;

/**
 * データとのインタフェース役
 *
 * @author taketo957
 *
 */
public class DAOInterface {
        private EventDAO dataSource;
        private ItemDAO idao;
        private UserDAO udao;
        private UserEventDAO uedao;
        private ItemEventDAO iedao;

        /**
         * デフォルトコンストラクタ
         */
        public DAOInterface() {

        }

        /**
         * コンストラクタ
         *
```

```
 * @param filename
 */
public DAOInterface(String filename) {
        dataSource = new SimpleFileRatingDAO(new File(filename
          ), ",");
        udao = new PrefetchingUserDAO(dataSource);
        idao = new PrefetchingItemDAO(dataSource);
        uedao = new PrefetchingUserEventDAO(dataSource);
        iedao = new PrefetchingItemEventDAO(dataSource);
}

public EventDAO getDataSource() {
        return dataSource;
}

public void setDataSource(EventDAO dataSource) {
        this.dataSource = dataSource;
}

public UserEventDAO getUedao() {
        return uedao;
}

public void setUedao(UserEventDAO uedao) {
        this.uedao = uedao;
}

public ItemEventDAO getIedao() {
        return iedao;
}

public void setIedao(ItemEventDAO iedao) {
        this.iedao = iedao;
}

public ItemDAO getIdao() {
        return idao;
}

public void setIdao(ItemDAO idao) {
        this.idao = idao;
}
```

```
        public UserDAO getUdao() {
                return udao;
        }

        public void setUdao(UserDAO udao) {
                this.udao = udao;
        }

}
```

ソースコード 6: 変更した類似度計算式を実装したクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import java.io.Serializable;

import javax.inject.Inject;

import org.grouplens.lenskit.core.Shareable;
import org.grouplens.lenskit.vectors.MutableSparseVector;
import org.grouplens.lenskit.vectors.SparseVector;
import org.grouplens.lenskit.vectors.similarity.PearsonCorrelation;
import org.grouplens.lenskit.vectors.similarity.SimilarityDamping;
import org.grouplens.lenskit.vectors.similarity.VectorSimilarity;

@Shareable
public class EnhancedPearsonCorrelation implements VectorSimilarity,
                Serializable {
        private static final long serialVersionUID = 1L;

        private final double shrinkage;

        public EnhancedPearsonCorrelation() {
                this(0);
        }

        @Inject
        public EnhancedPearsonCorrelation(@SimilarityDamping double s)
           {
                shrinkage = s;
        }

        @Override
```

```
        public double similarity(SparseVector vec1, SparseVector vec2) {
                double num1 = vec1.size();
                double num2 = vec2.size();
                MutableSparseVector mvec1 = vec1.mutableCopy();
                MutableSparseVector mvec2 = vec2.mutableCopy();
                double coinvoked = mvec1.countCommonKeys(mvec2);

                PearsonCorrelation pcc = new PearsonCorrelation();

                return ((2 * Math.abs(coinvoked)) / (Math.abs(num1) +
                  Math.abs(num2)))
                                * pcc.similarity(vec1, vec2);
        }

        @Override
        public boolean isSparse() {
                return true;
        }

        @Override
        public boolean isSymmetric() {
                return true;
        }

        @Override
        public String toString() {
                return String.format("EnhancedPearson[d=%s]", shrinkage
                  );
        }

}
```

ソースコード 7: ライブラリのコンポーネントを提供するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import org.grouplens.lenskit.ItemRecommender;
import org.grouplens.lenskit.ItemScorer;
import org.grouplens.lenskit.RatingPredictor;
import org.grouplens.lenskit.Recommender;
import org.grouplens.lenskit.RecommenderBuildException;
import org.grouplens.lenskit.core.LenskitConfiguration;
import org.grouplens.lenskit.core.LenskitRecommender;
```

```
import org.grouplens.lenskit.data.dao.EventDAO;
import org.grouplens.lenskit.knn.NeighborhoodSize;
import org.grouplens.lenskit.knn.user.UserUserItemScorer;
import org.grouplens.lenskit.transform.normalize.
  DefaultUserVectorNormalizer;
import org.grouplens.lenskit.transform.normalize.UserVectorNormalizer;
import org.grouplens.lenskit.vectors.similarity.PearsonCorrelation;
import org.grouplens.lenskit.vectors.similarity.VectorSimilarity;

public class LenskitComponentProvider {
        private DAOInterface dao;
        private LenskitConfiguration config;

        /**
         * コンストラクタ
         *
         * @param filename
         */
        @SuppressWarnings("unchecked")
        public LenskitComponentProvider(String filename) {
                dao = new DAOInterface(filename);
                config = new LenskitConfiguration();
                config.bind(EventDAO.class).to(dao.getDataSource());
                config.bind(ItemScorer.class).to(UserUserItemScorer.class);
                config.bind(VectorSimilarity.class).to(PearsonCorrelation.
                  class);
                config.bind(UserVectorNormalizer.class).to(
                                DefaultUserVectorNormalizer.class);
                config.set(NeighborhoodSize.class).to(5);
        }

        /**
         * アイテムレコメンダーを作成する
         *
         * @return
         */
        public ItemRecommender createItemRecommender() {
                Recommender rec = null;
                try {
                        rec = LenskitRecommender.build(config);
                } catch (RecommenderBuildException e) {
                        e.printStackTrace();
                }
```

```
        ItemRecommender irec = rec.getItemRecommender();
        return irec;
}

/**
 * アイテムスコアラーを作成する
 *
 * @return
 */
public ItemScorer createItemScorer() {
        Recommender rec = null;
        try {
                rec = LenskitRecommender.build(config);
        } catch (RecommenderBuildException e) {
                e.printStackTrace();
        }
        ItemScorer scorer = rec.getItemScorer();
        return scorer;
}

/**
 * 評価値予測器を作成する
 *
 * @return
 */
public RatingPredictor createRatingPredictor() {
        Recommender rec = null;
        try {
                rec = LenskitRecommender.build(config);
        } catch (RecommenderBuildException e) {
                e.printStackTrace();
        }
        RatingPredictor rp = rec.getRatingPredictor();
        return rp;
}

public DAOInterface getDao() {
        return dao;
}

public void setDao(DAOInterface dao) {
        this.dao = dao;
}
```

```
}
```

ソースコード 8: 評価値予測を行うクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import it.unimi.dsi.fastutil.longs.LongIterator;
import it.unimi.dsi.fastutil.longs.LongSet;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.grouplens.lenskit.data.dao.ItemDAO;
import org.grouplens.lenskit.data.dao.UserDAO;
import org.grouplens.lenskit.data.dao.UserEventDAO;
import org.grouplens.lenskit.data.event.Event;
import org.grouplens.lenskit.data.history.
  RatingVectorUserHistorySummarizer;
import org.grouplens.lenskit.data.history.UserHistory;
import org.grouplens.lenskit.data.history.UserHistorySummarizer;
import org.grouplens.lenskit.vectors.MutableSparseVector;
import org.grouplens.lenskit.vectors.SparseVector;

/**
 * 評価値予測を行うクラス
 *
 * @author taketo957
 *
 */
public class MissingValuePredictor {
        DAOInterface pre_daoi;
        DAOInterface daoi;

        /**
         * コンストラクタ. 予測に用いるデータソースを指定する.
         *
```

```
 * @param pre_daoi
 * @param daoi
 */
public MissingValuePredictor(DAOInterface pre_daoi, DAOInterface
   daoi) {
        this.pre_daoi = pre_daoi;
        this.daoi = daoi;
}

/**
 * 類似ユーザの情報に基づいて評価値を予測するメソッド.
 *
 * @param user_id
 * @param neighbors2sim
 * @param items
 * @return
 */
public SparseVector predictMissingValues(long user_id,
                Map<Long, Double> neighbors2sim, LongSet items)
                   {
        SparseVector pred_vec;
        UserEventDAO uedao = daoi.getUedao();

        double sum_of_sim = 0.0;
        double sum_of_dev = 0.0;
        double mean = calcEvalationAverageValue(user_id, uedao);
        Map<Long, Double> neighbors2mean = new HashMap<
          Long, Double>();

        for (Entry<Long, Double> e : neighbors2sim.entrySet()) {
                long neighbor_user_id = e.getKey();
                double sim_user_mean =
                  calcEvalationAverageValue(neighbor_user_id,
                                  uedao);
                neighbors2mean.put(neighbor_user_id,
                  sim_user_mean);
        }

        LongIterator li = items.iterator();
        MutableSparseVector result_vector = MutableSparseVector.
          create(items,
                          0.0);
        while (li.hasNext()) {
```

```
                long item = li.nextLong();
                for (Entry<Long, Double> e : neighbors2sim.
                  entrySet()) {
                        long neighbor_user_id = e.getKey();
                        UserHistory<Event> neighbor_user_events
                          = uedao
                                        .getEventsForUser(
                                          neighbor_user_id);
                        UserHistorySummarizer summarizer = new
                          RatingVectorUserHistorySummarizer();
                        SparseVector vector = summarizer
                                        .summarize(
                                          neighbor_user_events);
                        if (!vector.containsKey(item)) {
                                continue;
                        }
                        sum_of_dev += e.getValue()
                                        * (vector.get(item) -
                                          neighbors2mean
                                                        .get(
                                                          neighbor_user_id
                                                          ));
                        sum_of_sim += e.getValue();
                }
                double result = mean + (sum_of_dev / sum_of_sim);
                result_vector.set(item, result);
        }
        pred_vec = result_vector;
        return pred_vec;

}

public SparseVector predictMissingCIMValues(long user_id,
                Map<Long, Double> pre_neighbors2sim) {
        SparseVector pred_vec;
        List<Map.Entry<Long, Double>> entries = sortNeighbors(
          pre_neighbors2sim);
        UserEventDAO pre_uedao = pre_daoi.getUedao();
        double mean = calcEvalationAverageValue(user_id,
          pre_uedao);

        double sum_of_sim = 0.0;
```

```
ItemDAO idao = daoi.getIdao();
UserDAO udao = daoi.getUdao();
UserEventDAO uedao = daoi.getUedao();

LongSet items = idao.getItemIds();

MutableSparseVector result_vec = MutableSparseVector.
  create(items, 0.0);

Iterator<Entry<Long, Double>> neighbor_itr = entries.
  iterator();
while (neighbor_itr.hasNext()) {
        Entry<Long, Double> neighbor = neighbor_itr.next
          ();
        long user = neighbor.getKey();
        double sim = neighbor.getValue();

        if (!udao.getUserIds().contains(user)) {
                continue;
        }
        if (sim <= 0.0) {
                continue;
        }

        NeighborhoodProvider np = new
          NeighborhoodProvider(daoi);
        Map<Long, Double> neighbors2sim = np.
          findNeighborUsers((int) user);
        Map<Long, Double> neighbors2mean = new
          HashMap<Long, Double>();

        for (Entry<Long, Double> e : neighbors2sim.
          entrySet()) {
                long neighbor_user_id = e.getKey();
                double sim_user_mean =
                  calcEvalationAverageValue(
                                  neighbor_user_id, uedao);
                neighbors2mean.put(neighbor_user_id,
                  sim_user_mean);
        }

        sum_of_sim += sim;
```

```
                SparseVector vec = predictMissingValues(user,
                  neighbors2sim, items);

                MutableSparseVector mvec = vec.mutableCopy();
                mvec.multiply(sim);
                result_vec.add(mvec);
        }

        result_vec.multiply(1 / sum_of_sim);
        result_vec.add(mean);
        pred_vec = result_vec;
        return pred_vec;
}

private double calcEvalationAverageValue(long i, UserEventDAO
  pre_uedao) {
        UserHistory<Event> history = pre_uedao.getEventsForUser(
          i);
        UserHistorySummarizer summary = new
          RatingVectorUserHistorySummarizer();
        SparseVector user_summary_vec = summary.summarize(
          history);
        double mean = user_summary_vec.mean();
        return mean;
}

private List<Map.Entry<Long, Double>> sortNeighbors(
                Map<Long, Double> neighbors) {
        List<Map.Entry<Long, Double>> entries = new ArrayList
          <Map.Entry<Long, Double>>(
                        neighbors.entrySet());
        Collections.sort(entries, new Comparator<Map.Entry<Long
          , Double>>() {

                @Override
                public int compare(Entry<Long, Double> entry1,
                                Entry<Long, Double> entry2) {
                        return ((Double) entry2.getValue()).
                          compareTo((Double) entry1
                                        .getValue());
                }
        });
        return entries;
```

```
        }

}
```

ソースコード 9: 類似ユーザの集合を作成するクラス

```
package jp.ac.kyoto_u.i.soc.ai.graduation_thesis;

import it.unimi.dsi.fastutil.longs.LongIterator;
import it.unimi.dsi.fastutil.longs.LongSet;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.grouplens.lenskit.data.dao.UserDAO;
import org.grouplens.lenskit.data.dao.UserEventDAO;
import org.grouplens.lenskit.data.event.Event;
import org.grouplens.lenskit.data.history.
  RatingVectorUserHistorySummarizer;
import org.grouplens.lenskit.data.history.UserHistory;
import org.grouplens.lenskit.data.history.UserHistorySummarizer;
import org.grouplens.lenskit.knn.user.UserSimilarity;
import org.grouplens.lenskit.knn.user.UserVectorSimilarity;
import org.grouplens.lenskit.vectors.SparseVector;

/**
 * 近傍ユーザ発見に関連することを行うクラス
 *
 * @author taketo957
 *
 */
public class NeighborhoodProvider {
        private DAOInterface daoi;
        private UserEventDAO uedao;
        private UserSimilarity usim;

        /**
         * コンストラクタ
```

```
 *
 * @param daoi
 */
public NeighborhoodProvider(DAOInterface daoi) {
        this.daoi = daoi;
        this.uedao = daoi.getUedao();
        this.usim = new UserVectorSimilarity(new
          EnhancedPearsonCorrelation());
}

/**
 * 指定した
   ID のユーザと類似するユーザとその類似度のマップを作成する
 *
 * @param user_id
 * @return
 */
public Map<Long, Double> findNeighborUsers(int user_id) {
        UserDAO udao = daoi.getUdao();
        LongSet user_ids = udao.getUserIds();
        if (!user_ids.contains(user_id)) {
                return null;
        }
        SparseVector vec_for_target = makeSparseVector(user_id);
        LongIterator itr = user_ids.iterator();
        Map<Long, Double> user2sim = new HashMap<Long,
          Double>();
        while (itr.hasNext()) {
                long id = itr.next();
                SparseVector vec_for_pair = makeSparseVector(id);
                double similarity = usim.similarity(user_id,
                  vec_for_target, id,
                                  vec_for_pair);
                user2sim.put(id, similarity);
        }
        List<Map.Entry<Long, Double>> entries = new ArrayList
          <Map.Entry<Long, Double>>(
                        user2sim.entrySet());
        Collections.sort(entries, new Comparator<Map.Entry<Long
          , Double>>() {
                @Override
                public int compare(Entry<Long, Double> entry1,
                                  Entry<Long, Double> entry2) {
```

```
                        return ((Double) entry2.getValue()).
                          compareTo((Double) entry1
                                  .getValue());
                    }
                });

                Map<Long, Double> result = new HashMap<Long, Double
                  >();
                for (Entry<Long, Double> s : entries) {
                    if (s.getValue().isNaN() || s.getValue() == 1.0
                                || s.getValue() <= 0.0) {
                        continue;
                    }
                    result.put(s.getKey(), s.getValue());
                }
                return result;
        }

        private SparseVector makeSparseVector(long user_id) {
                UserHistory<Event> history = uedao.getEventsForUser(
                  user_id);
                UserHistorySummarizer sum = new
                  RatingVectorUserHistorySummarizer();
                SparseVector vec_for_target = sum.summarize(history);
                return vec_for_target;
        }

}
```