Vol.2017-SE-197 No.5
2017/11/9

ソフトウェア開発において併用されるライブラリ機能の推薦

桂川 大輝[1,a)]　伊原 彰紀[1,b)]　ラウラ ガイコビナ クラ[1,c)]　松本 健一[1,d)]

概要：高品質なソフトウェアシステムを効率的に開発するために多くのプロジェクトがライブラリを利活用しており，複数のライブラリを併用しているシステムも少なくない．本論文では，システムを効率的に実現するために併用されるライブラリの機能を明らかにする．さらに，プロジェクトが開発するシステムが利用中のライブラリに基づき，当該システムで併用することが可能なライブラリの機能を推薦するモデルを提案する．本論文で構築したモデルにおいて，3 件の機能を推薦した場合，66%の精度であることが分かった．

1. はじめに

ソフトウェアシステム（以下，システム）を効率的に実装するために，ライブラリの利用は必要不可欠な手段となっている [4,6]．ライブラリは，汎用性の高い複数のプログラムを再利用可能な形でまとめたファイルであり，ライブラリを利用することでシステムの一部の機能を容易に実装することができる．例えば，Web アプリケーション開発で利用されるライブラリには，プログラム実行時のログ出力，HTML 解析，SSH（セキュアシェル），暗号化の機能を実現する GWT [*1]，Spring [*2]，Hibernate [*3]がある．

今日では，多種多用なライブラリが多数開発され，また，それらの多くは GitHub [*4]などで公開されている．膨大にあるライブラリの中から，システム開発プロジェクトが，実装を効率化するライブラリを容易に発見するために，ライブラリ検索サービスが公開されている．例えば，Java 言語のライブラリであれば Maven Repository [*5]，JavaScript 言語であれば，npm [*6]がある．これらライブラリ検索サービスでは，ライブラリを機能別に分類し，目的に応じたライブラリの検索を容易にする機能を備えている．システム開発プロジェクトは，これら検索サービスを活用することでシステムの実装を効率化するライブラリを特定している．

システム開発プロジェクトでは，ライブラリ検索サービスから複数のライブラリを見つけ出し，併用することも少なくない．しかし，ライブラリ検索サービスには類似する機能を有するライブラリが多数登録されており，その中から併用するライブラリの組み合わせは膨大に存在する．本論文では，システム開発において頻繁に併用されるライブラリの機能の組み合わせを明らかにする（4 章）．また，システムが利用中のライブラリに基づき，アソシエーションルールマイニング手法を用いて，当該システムが併用することが可能なライブラリの機能を推薦するモデルを提案する（5 章）．本論文では，Java プログラミング管理ツール MAVEN [*7]を利用する 8,1420 件のシステムが利用するライブラリを対象に実験を行う．

以降，2 章ではライブラリ推薦の関連研究，3 章では分析，実験を行うための準備について述べる．4 章は併用されるライブラリの機能の分析について，5 章では併用可能なライブラリの機能を推薦するモデルの実験とその評価を述べ，6 章では本論文における制約を述べる．最後に 7 章では本論文でのまとめと今後の展望について述べる．

2. ライブラリ推薦手法

ソフトウェアシステムの効率的な実装を実現するために，開発プロジェクトが利用可能なライブラリを推薦する研究が進められている．Thung ら [9] は，システムが利用す

[1] 奈良先端科学技術大学院大学
Nara Institute of Science and Technology
[a)] katsuragawa.daiki.ka7@is.naist.jp
[b)] akinori-i@is.naist.jp
[c)] raula-k@is.naist.jp
[d)] matumoto@is.naist.jp

[*1] GWT: http://www.gwtproject.org
[*2] Spring: https://netbeans.org/kb/docs/web/quickstart-webapps-spring_ja.html
[*3] Hibernate: http://www.techscore.com/tech/Java/Others/Hibernate/index/
[*4] GitHub: https://github.com/
[*5] Maven Repository: https://mvnrepository.com/
[*6] npm: https://www.npmjs.com/
[*7] Maven: https://maven.apache.org/

るライブラリ名を入力とし，アソシエーションルールマイニング手法と協調フィルタリングによって，他のシステムが併用するライブラリを推薦する手法を提案している．また，Ouni ら [7] は，多目的最適化を用いた検索ベースのアルゴリズム NSGA-II によって併用するライブラリを推薦する手法を提案している．Zhu ら [8] は，70 を超える様々な推薦アルゴリズムを実装したツール LibRec [*8]によってライブラリを推薦する手法を比較評価している．

一方で，本論文では，システムが利用するライブラリの機能，及び，その併用が可能なライブラリの機能について分析する．また，併用することが可能なライブラリの機能を推薦する手法を提案する．ライブラリの機能とは，システムがライブラリを利用する目的，用途を指す．例えば，ソフトウェアテストの自動実行プログラムの作成などを支援する Testing Framework（JUnit [*9]など），プログラムの動作ログの記録などを支援する Logging Framework（Log4j [*10]，SLF4J [*11]など）がある．システム開発において，これらの異なる機能を併用することで，様々な機能を効率的に実装することができる．

従来研究で提案されたライブラリ推薦手法は，機能に関わらず併用可能なライブラリを推薦する手法であり，ライブラリの機能，種類を把握していない開発者にとって，推薦されるライブラリの用途を理解するには時間を要する．また，類似する機能を有するライブラリが多数存在するため [3, 10]，自身が開発するシステムに適切なライブラリの組み合わせを理解することは容易でない．本論文では，併用されるライブラリの機能および，その組み合わせを明らかにするために 2 つの Research Question (RQ) に回答する．

RQ1: ソフトウェアシステムは，どの機能を有するライブラリを併用しているのか？

RQ2: ソフトウェアシステムが利用中のライブラリから，併用可能なライブラリの機能をどの程度の精度で推薦できるのか？

3. 実験準備

3.1 分析対象データセット

本論文では，システムで利用されているライブラリを特定するために，Java プログラミング管理ツール MAVEN を利用しているシステムを分析対象とする．MAVEN は，システムのビルド，テスト，成果物などを管理するツールであり，システムを開発するプロジェクトが依存するライブラリ情報を Project Object Model（以下，POM）ファイルで一括管理する機能を提供している．POM ファイルにはシステムが利用しているライブラリ名，バージョン番号が記述されており，本論文では POM ファイルを用いて，システムが利用するライブラリを分析する．

[手順 1: MAVEN を利用するシステムの特定] 本論文では，ライブラリを特定する必要があるため，GitHub に公開されるシステムの中で MAVEN を利用しているシステム，つまり POM ファイルを有するシステムを対象とする

[手順 2: システムが利用するライブラリの特定] 本論文で対象とするシステムが持つ POM ファイルを解析し，各システムが利用するライブラリ名を取得する．

本論文では，手順 1，手順 2 を満たしたデータとして，Kula ら [5] が抽出した GitHub に公開されるシステムの中で MAVEN を利用しているシステム及び，それらが利用するライブラリを対象とする [*12]．上記データはシステム毎のライブラリの導入，最後に利用が見られた日付が示されている．これらを用いて，システム毎に最終的に利用しているライブラリを特定する．また，本論文では併用するライブラリを対象とするため，Thung ら [9] と同様に 10 件以上のライブラリを利用しているシステムを対象とする．Kula らが抽出したシステムで，これらを満たすシステムは 8,142 件であった．

[手順 3: システムが利用するライブラリの機能の特定] 各システムが利用するライブラリの機能を特定するために，Maven Repository でライブラリ分類のために利用される 150 の機能（Maven Repository では Category）の情報を用いる [*13]．Maven Repository では，ライブラリを 150 機能に分類し，各機能には最大 150 のライブラリが登録されている．Maven Repository に登録される利用数の多いライブラリの機能は，Testing Frameworks，Logging Frameworks，Language Runtime などがある [*14]．しかし，Maven Repository で公開されている一部のライブラリや，利用数の少ないライブラリは機能に分類されていない．実際に，手順 2 によって特定したライブラリ 38,884 件のうち 37,306 件は，Maven Repository で機能別に分類されていない，または，Maven Repository に登録されていないため，「その他の機能」として取り扱う．本論文が対象とするシステム 8,142 件において，利用頻度の高い機能を表 1 に示す．また，利用頻度の高いライブラリにおいて機能に分類されている割合を表 2 に示す．利用頻度の高い上位 100 件のライブラリの中で 94%がライブラリの機能に分類されているため（6%は「その他の機能」に分類されている），

[*8] LibRec: https://www.librec.net/
[*9] JUnit: http://junit.org/junit5/
[*10] Log4j: https://logging.apache.org/log4j/2.x/
[*11] SLF4J: https://www.slf4j.org/
[*12] Impact-of-Security-Advisories-on-Library-Migrations by raux: https://raux.github.io/Impact-of-Security-Advisories-on-Library-Migrations/
[*13] 2017 年 5 月 26 日取得
[*14] Maven Repository Poplular Categories: https://mvnrepository.com/open-source

表 1 分析対象システムにおけるライブラリの機能の利用頻度

ランク	ライブラリの機能	利用頻度	被利用数
1	Testing Frameworks	52%	4,198
2	Logging Frameworks	48%	3,940
3	Java Specifications	42%	3,437
4	Core Utilities	39%	3,181
5	Logging Bridges	29%	2,348
6	Dependency Injection	27%	2,162
7	JSON Libraries	26%	2,154
8	Mocking	25%	2,048
9	I/O Utilities	21%	1,729
10	XML Processing	21%	1,698
11	Web Frameworks	18%	1,497
12	HTTP Clients	17%	1,404
13	Bytecode Libraries	16%	1,285
14	Embedded SQL Databases	15%	1,253
15	Object/Relational Mapping	14%	1,166
16	Collections	14%	1,102
17	Aspect Oriented	13%	1,062
18	Base64 Libraries	13%	1,031
19	Transaction APIs/Managers	12%	984
20	Validation Frameworks	11%	902
…	…	…	…
150	Enterprise Service Bus	0%	5

表 2 分析対象システムにおいて利用頻度の高いライブラリが機能に分類されている割合

利用頻度の高いライブラリ (TOP N)	分類されている割合
1-100	94%
1-200	85%
1-300	75%
1-400	69%
1-500	61%
1-600	57%
1-700	53%
1-800	50%
1-900	48%
1-1000	46%

Maven Repository が用いる機能分類で，本論文が対象とする利用頻度が高いライブラリの機能を特定することができた．

3.2 機能実装のために併用されるライブラリ

RQ1，RQ2 に回答するにあたり，システムが併用するライブラリ数，及び，機能数を分析する．システムでは，同じ機能に分類されるライブラリを併用することもあるため，システムが併用するライブラリ数とライブラリの機能数は一致するとは限らない．図 1，図 2 は，それぞれライブラリ，及び，ライブラリの機能を併用するシステムの分布を示し，横軸は併用しているライブラリ数，または，ライブラリの機能数を示す．それぞれの図中の縦軸は中央値

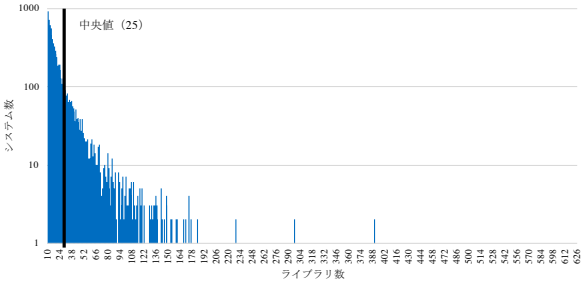

図 1 併用されるライブラリ数別のシステム数の分布

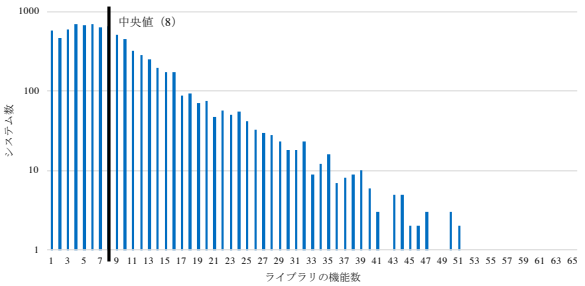

図 2 併用されるライブラリの機能数別のシステム数の分布

を示す．システムは 25 件（中央値）のライブラリ，8 件（中央値）の機能を利用している．分析対象システムの中で，7,185 件のシステムが 2 件以上の機能を利用し，特に，多くのシステムは 3 種類から 5 種類の機能実装のためにライブラリを利用している．

4. RQ1: ソフトウェアシステムは，どの機能を有するライブラリを併用しているのか？

4.1 分析手法

RQ1 では，システムが併用するライブラリの機能を明らかにするために，アソシエーションルールマイニング手法 [1] を用いる．アソシエーションルールは，ある事象の発生に対して，別の事象が発生する，同時性や関係性が強い事象の組み合わせ，強い事象間の関係を示す．また，アソシエーションルールマイニングとは，多数の事象の組み合わせから，アソシエーションルールを抽出する技術を指す．アソシエーションルールは条件部（A）と結論部（B）によって以下のように表される．

$$\{A\} \Rightarrow \{B\}$$

アソシエーションルールは支持度，信頼度，LIFT 値の 3 つの指標値で評価する．

- 支持度は，全データの中でルールが出現する割合
- 信頼度は，条件部の項目が出現する割合の中で条件部（A）と結論部（B）が同時に出現する割合
- LIFT 値は，条件部と結論部を共に満たすデータの割合と全事象の中で結論部を満たす割合を比較した倍率

また，アソシエーションルールマイニングでは，多数のルールが出力されるため，apriori アルゴリズムを用いて，

指標値に基づき一定の条件を満たすルールを選定する．

本論文では，Python 言語における apriori アルゴリズムが実装されているライブラリ `Orange` [2] を用いてアソシエーションルールマイニング手法を実装する．

4.2 結果

表 3 は，分析対象としたシステムにおいて併用されることが多いライブラリの機能の組み合わせをアソシエーションルールマイニング手法によって出力した結果を示す．表 3 は，支持度によりソートしているため，条件部と結論部に現れる機能が併用される頻度が高い組み合わせを示す．

知見 1 — 利用頻度が高い機能 Testing Frameworks, Logging Frameworks を併用するシステムが多い．表 3 より，Testing Frameworks，Logging Frameworks という組み合わせの支持度が最も高く，22%のシステムで利用されていることが明らかになった．Testing Frameworks と Logging Frameworks はそれぞれがそれぞれが利用頻度が高い機能であり，独立して，汎用的なライブラリの機能であるため，並列利用が多いと考えられる．ただし，これらルール 1，2 の信頼度はそれぞれ 0.60，0.48 と約半数であり，必ずしも併用されておらず，依存関係が高いわけではないと考えられる．

知見 2 — Logging Bridges の機能を有するシステムは，高い頻度で Logging Frameworks の機能を併用する．表 3 に示すルール 5 より，Logging Bridges を利用する 80%システムはの Logging Frameworks を利用する．一方で，ルール 6 より，同じ組み合わせではあるが Logging Frameworks を利用するシステムで，Logging Bridges を利用するシステムは 43%しかいない．これらの結果は，併用の頻度は高いものの，Logging Frameworks を利用するからといって Logging Bridges は必ずしも利用されず，また，Logging Bridges を利用するにあたって Logging Frameworks を併用する可能性が高いことを示している．Logging Frameworks の一部はログ出力のインターフェースのみを提供するものであるのに対して，Logging Bridges は，Logging Frameworks に対する実装のプロキシ（バインダー）であることで併用されている考えられる（例えば，{SLF4J, slf4j-log4j12, log4j} など）．一方で，Logging Frameworks には，ログ出力のインターフェースに加え，実装もされているものもあるため，Logging Frameworks のみを利用することが考えられる（例えば，{SLF4J, logback} など）[*15]．

表 3 で示すルールから高い頻度で併用されるライブラリの機能が明らかになり，これを用いてライブラリの機能が推薦できると考えられる．

続く RQ2 では，アソシエーションルールマイニング手法で得られたルールを用いて，併用可能なライブラリの機能を推薦するモデルを構築する．表 4 は，信頼度でソートした機能の組み合わせを示す．ルールに現れる機能の組み合わせには，知見 2 で示すように，支持度は低いが信頼度が高いルール（例えば {Logging Frameworks}⇒{Logging Bridges}）が存在し，単純に利用数の多い機能だけでは推薦できないことがわかる．本論文では，RQ2 において，アソシエーションルールマイニング手法で得られたルールの信頼度に基づき，併用可能なライブラリの機能を推薦するモデルを構築する．

5. RQ2: ソフトウェアシステムが利用中のライブラリから，併用可能なライブラリの機能をどの程度の精度で推薦できるのか？

5.1 概要

本論文では，システムが利用中のライブラリから併用可能なライブラリの機能を推薦するモデルを提案する．本論文では，アソシエーションルールマイニングを用いて併用される頻度の高いライブラリの機能を優先的に推薦するために，信頼度を推薦における点数（スコア）として用いる．

図 3 は，提案手法の概略図を示す．はじめに Kula らが抽出したシステムから，トレーニングデータセット，及び，テストデータセットを選定する（1）．次に，Maven Repositry と FunctionIdentifier を用いて各データセットが併用しているライブラリの機能を特定する（2）．そして，RuleGenerator に，特定されたトレーニングデータセットのライブラリの機能を入力して，アソシエーションルールを抽出する（4）．RankGenerator に，特定されたテストデータのライブラリの機能と，RuleGenerator が抽出したルールを入力として併用される可能性が高い機能とそのスコアを推薦リスト（Recommendation List）として出力する（4）．以上の流れでシステムが利用中のライブラリから併用可能なライブラリの機能を推薦する．続いて，FunctionIdentifier，RuleGenerator，RankGenerator について説明する．

[FunctionIdentifier] FunctionIdentifier は，Maven Repositry で利用される Category を機能名として用いてライブラリの機能を出力する．

[RuleGenerator] RuleGenerator は，トレーニングデータセットからアソシエーションルールマイニングを用いて，ライブラリの機能のルールを出力する．Algorithm 1 は RuleGenerator の具体的なアルゴリズムを示す．RuleGenerator の入力は，トレーニングデータセットにおけるシステムが利用するライブラリの機能であり，アソシエーションルールマイニングにおける最小支持度（minsup），最小信頼度（minconf）をルールの選定のために設定する．出力は，アソシエーションルールマイニングによって出力されるルール群である．はじめに，アソシエーションルールマイニングに用いるライブラリの機能の利用パターン

[*15] SLF4J: `https://www.slf4j.org/`

表 3 アソシエーションルールの出力結果（支持度の高い上位 10 件）

ルール	条件部		結論部	支持度	信頼度	LIFT 値
1	{Logging Frameworks}	⇒	{Testing Frameworks}	0.23	0.60	1.28
2	{Testing Frameworks}	⇒	{Logging Frameworks}	0.23	0.48	1.28
3	{Core Utilities}	⇒	{Testing Frameworks}	0.18	0.62	1.32
4	{Testing Frameworks}	⇒	{Core Utilities}	0.18	0.38	1.32
5	{Logging Bridges}	⇒	{Logging Frameworks}	0.16	0.80	2.13
6	{Logging Frameworks}	⇒	{Logging Bridges}	0.16	0.43	2.13
7	{Java Specifications}	⇒	{Testing Frameworks}	0.16	0.48	1.02
8	{Testing Frameworks}	⇒	{Java Specifications}	0.16	0.34	1.02
9	{Mocking}	⇒	{Testing Frameworks}	0.15	0.79	1.70
10	{Testing Frameworks}	⇒	{Mocking}	0.15	0.33	1.70

表 4 アソシエーションルールの出力結果（信頼度の高い上位 10 件）

ルール	条件部		結論部	支持度	信頼度	LIFT 値
1	{Dependency Injection, Logging Bridges}	⇒	{Logging Frameworks}	0.06	0.89	2.37
2	{Mail Clients}	⇒	{Java Specifications}	0.06	0.89	2.68
3	{Logging Bridges, Testing Frameworks}	⇒	{Logging Frameworks}	0.11	0.88	2.34
4	{Mocking, Logging Frameworks}	⇒	{Testing Frameworks}	0.09	0.88	1.87
5	{Mocking, Core Utilities}	⇒	{Testing Frameworks}	0.08	0.87	1.87
6	{Mocking, Java Specifications}	⇒	{Testing Frameworks}	0.06	0.85	1.81
7	{Dependency Injection, Core Utilities, Logging Frameworks}	⇒	{Testing Frameworks}	0.06	0.83	1.78
8	{Validation Frameworks}	⇒	{Java Specifications}	0.06	0.83	2.49
9	{Web Frameworks, Logging Frameworks}	⇒	{Java Specifications}	0.06	0.81	2.45
10	{Logging Bridges}	⇒	{Logging Frameworks}	0.16	0.80	2.13

Algorithm 1 RuleGenerator

Input: $Systems$ = set of systems with libraries combined use by each of them
$minsup$ = minimum support threshold
$minconf$ = minimum confidence threshold
Output: $Rules$ = association rules
1: **Method:**
2: **Let** $baskets = \{\}$
3: **for all** $System \in Systems$ **do**
4: **for all** $LibraryFunction \in System$ **do**
5: **add** $LibraryFunction$ to $baskets$
6: **end for**
7: **end for**
8: **Let** $Rules = \{\}$
9: $Rules = AssociationRulesSparseInducer(baskets, minsup, minconf)$
10: **return** $Rules$

を作成するために，トレーニングデータセットが利用するライブラリの機能を basket に加える（2〜7 行目）．次に，basket，minsup，minconf をアソシエーションルールに入力し，最小支持度，最小信頼度の条件を満たすすべての Rules を RuleGenerator の出力とする（9〜10 行目）．以上より，トレーニングデータセットからアソシエーションルールマイニングを用いてルールを出力する．

[**RankGenerator**] RankGenerator は，RuleGenerator によって出力されたルールとテストデータセットにおけるシステムが利用するライブラリの機能を入力として受け取り，結論部にあたるライブラリの機能とスコアを推薦リストとして出力する．Algorithm 2 は RankGenerator の具体的なアルゴリズムを示す．はじめに，推薦に利用できるルールを特定するために，ルールの条件部となるライブラリの機能が，テストデータにおけるシステムが利用するライブラリの機能の部分集合であるかを判定する（8 行目）．判定が真である場合，ルールの結論部は併用される可能性が高いライブラリの機能であるため，そのライブラリの機能と信頼度をを推薦リストに加える．もし，ライブラリの機能がすでに推薦リストに含まれ，信頼度が推薦リストにあるスコアを上回る場合は，他のルールによって推薦されるべきだと判定されるため，値を更新する（9〜17 行目）．以上をルールの数だけ繰り返し，推薦リストのライブラリの機能とスコアを更新する（6〜19 行目）．最後に，推薦リストを RankGenerator の出力とする（20 行目）．以上より，テストデータセットにおけるシステムに対応したライブラリの機能の推薦リストを出力する．

5.2 評価方法

本論文では，提案手法を評価するために，テストデータの各システムが利用するライブラリをランダムに 2 分割し，一方を現在のライブラリとして推薦モデルへの入力，他方の機能を今後導入するであろう正解の集合とする．そ

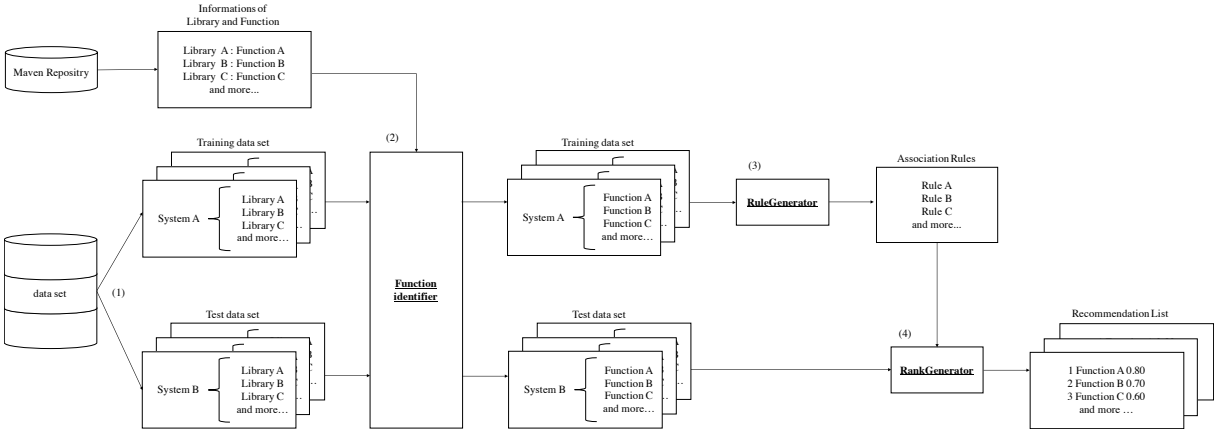

図 3 ライブラリの機能推薦手法の概略図

Algorithm 2 RankGenerator

Input: $CurLibrariesFunction$ = libraries function that the system combined use
$Rules$ = association rules
Output: $RecommendationList$ = libraries function and score

Method:
Let $RecommendationList = \{\}$
for all $Rule \in Rules$ **do**
  if $CurLibrariesFunction \supseteq Rule.PreCondition$ **then**
    for all $A \in Rule.PostCondition$ **do**
      if $A \notin CurLibrariesFunctions$ **then**
        add A and $A.Conf$ to $RecommendationList$
      else
        if $RecommendationList[A] < A.Conf$ **then**
          add $RecommendationList[A] = A.Conf$
        end if
      end if
    end for
  end if
end for
return $RecommendationList$

して，推薦モデルが，併用される可能性の高いライブラリの機能を推薦することにより，開発者が現在利用しているライブラリに対して他の併用されるライブラリの機能を発見するのに役に立つ状況を模倣する．推薦モデルは，システムが利用中のライブラリから併用可能なライブラリの機能を推薦するため，本論文では少なくとも 2 つ以上のライブラリの機能を利用しているシステム 7,185 件を対象とする．また，推薦モデルでは「その他の機能」と分類されたライブラリは推薦の対象外とする．

実際は，推薦されたライブラリの機能のスコアの上位から順に導入を検討することが考えられる．従って，評価において，推薦数（Recommendation Size）を定義し，提案手法により出力される推薦リストからスコアが高い推薦数分のライブラリの機能を選出する例えば，推薦数を 3 と設定した場合，推薦リストからスコアが高い上位 3 つのライブラリの機能を選出する．

本論文では，提案手法を評価するために，評価指標として推薦精度を定義する．選出されたライブラリの機能に対して少なくとも正解が 1 つ存在するシステムの割合を推薦精度として算出する．

$$\text{推薦精度} = \frac{\text{システム数（正解が一つ以上存在）}}{\text{全システム数}}$$

本論文では，提案手法の推薦精度を測定するために 10 分割交差検定を行う．はじめに，データセットにおけるシステムをランダムに 10 つの同じサイズのセットに分割する．1 回の施行で 1 つのセットをテストデータセット，他 9 つのセットをトレーニングデータセットとし，推薦精度を算出する．以上の施行を 10 回繰り返し，推薦精度の平均を評価する．

今回，極力多くのルールを出力するために，最小支持度（minsup）を 0.05，結果的に高い信頼度のものが推薦リストに加えられるため，最小信頼度（minconf）を 0.4 と設定した．

システムで利用されているライブラリの機能，及び，推薦数によっては，推薦リストに十分な数のライブラリの機能が存在しない可能性があるため，推薦リストとして出力されるライブラリの機能の数が推薦数を満たさない場合，不足分はランダムに選出する．

5.3 評価結果

本論文では，5.1 節の条件を満たす 7,185 件のシステムに対して 10 分割交差検定により実験を実施した（minsup=0.05，minconf=0.4）．図 4 は，推薦数による推薦精度を示す．図 4 より，ライブラリの機能の推薦数が 1 のとき 41%，3 のとき 66%，5 のとき 75%である．また，変化量が徐々に減少していることから，推薦リストの上位に正解となるライブラリの機能が出現している．従って，提案手法において，アソシエーションルールをの信頼度を用いたライブラリの機能の推薦が有用であることが示唆された．

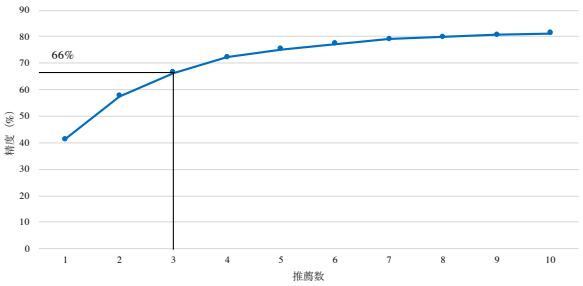

図 4 推薦数の変更が推薦精度に与える影響

6. 制約

内的妥当性： 今回対象としたライブラリは，Maven Repository で公開されてるライブラリかつ，Category が明記されているものに限定した．しかし，実際は，Maven Repository で Category が明記されていないライブラリや，そもそも Maven Repository で公開されていないライブラリが存在する．それらの存在を今後考慮する必要がある．

外的妥当性： 本論文では，分析及び実験対象を MAVEN を利用する Java 言語のプロジェクトのみとした．しかし，それ以外の言語，プロジェクトに対しても，本実験と同様の傾向が得られるかを調査する必要があり，今後の課題とする．

7. おわりに

本論文では，効率的なシステム開発を実現するために併用されるライブラリの機能を分析した．さらに，システムが利用中のライブラリに基づき，当該システムが併用することが可能なライブラリの機能を推薦するモデルを提案し，その精度を検証した．分析結果として，頻繁に 併用されるライブラリの機能の組み合わせを明らかにした．本論文で構築したモデルにおいて，3 件の機能を推薦した場合，66%の精度であることが分かった．今後は，本論文で構築した推薦モデルを元に，ライブラリを推薦し，その実用性を検討する．

謝辞

本論文の一部は，文部科学省科学研究補助費（基盤 A: 17H00731，若手 B: 課題番号 16K16037）による助成を受けた．

参考文献

[1] Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases, *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pp. 487–499 (1994).

[2] Demšar, J., Curk, T., Erjavec, A., Črt Gorup, Hočevar, T., Milutinovič, M., Možina, M., Polajnar, M., Toplak, M., Starič, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M. and Zupan, B.: Orange: Data Mining Toolbox in Python, *Journal of Machine Learning Research*, Vol. 14, pp. 2349–2353 (2013).

[3] Kabinna, S., Bezemer, C. P., Shang, W. and Hassan, A. E.: Logging Library Migrations: A Case Study for the Apache Software Foundation Projects, *Proceedings of the 13th International Conference on Mining Software Repositories (MSR'16)*, pp. 154–164 (2016).

[4] Krueger, C. W.: Software Reuse, *ACM Computing Surveys (CSUR)*, Vol. 24, No. 2, pp. 131–183 (1992).

[5] Kula, R. G., German, D. M., Ouni, A., Ishio, T. and Inoue, K.: Do developers update their library dependencies?, *Empirical Software Engineering*, p. 134 (2017).

[6] Lim, W. C.: Effects of Reuse on Quality, Productivity, and Economics, *IEEE Software*, Vol. 11, No. 5, pp. 23–30 (1994).

[7] Ouni, A., Kula, R. G., Kessentini, M., Ishio, T., German, D. M. and Inoue, K.: Search-based software library recommendation using multi-objective optimization, *Information and Software Technology*, Vol. 83, pp. 55 – 75 (2017).

[8] Sun, Z., Guo, G. and Zhang, J.: Exploiting implicit item relationships for recommender systems, *Proceedings of the International Conference on User Modeling, Adaptation, and Personalization*, pp. 252–264 (2015).

[9] Thung, F., Lo, D. and Lawall, J.: Automated library recommendation, *Proceedings of the 20th Working Conference on Reverse Engineering (WCRE'13)*, pp. 182–191 (2013).

[10] Zerouali, A. and Mens, T.: Analyzing the evolution of testing library usage in open source Java projects, *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering (SANER'17)*, pp. 417–421 (2017).