Evaluation of Collaborative Filtering and Content Based Filtering Recommenders in jTutors (an online intelligent tutoring system)

AN ABSTRACT OF THE THESIS OF

Bharatwaj Appasamy for the degree of Master of Science in Computer Science

presented on March 14, 2014.

Title: Evaluation of Collaborative Filtering and Content Based Filtering Recommenders in jTutors (an online intelligent tutoring system).

Abstract approved:

_____

Christopher P. Scaffidi

Software engineers often need help with discovering and learning how to use APIs. For example, software engineers who are starting to learn Java, and they want to implement a certain feature in a program, they might want to reuse existing APIs in order to save time versus rewriting it themselves from scratch. The most widely-used method for discovering APIs is to search for APIs. Unfortunately, the search results typically take users to a collection of code, documentation, and examples, none of which is collected together in a form that is optimized for making a given API understandable. In our research group's previous work, we created the jTutors system,

which bundles the materials related to an API together into an interactive tutor that has been shown to help people learn an API faster. Yet jTutors has not had a facility to help programmers learn a *sequence* of tutors, for the common situation where software engineers want to learn a combination of APIs.

This thesis describes a new system that provides tutor recommendations to discover APIs in a sequence that users consider to be relevant and well-timed to their needs. The system integrates two internal algorithms to determine, for a given user at a given moment in time, what API tutors to recommend for that user's consideration. One of the algorithms is a content-based recommender (CBR), which builds a graph reflecting the relationships between APIs (as reflected in how they call or refer to one another) to determine the most commonly used APIs with respect to a given API and also the order in which it makes sense to learn the APIs. The other algorithm is a standard collaborative filtering (CF) algorithm, which identifies which users tend to give similar ratings as one another, and then uses the ratings of one person to recommend a sequence of tutors for other people. In an empirical study, 25 novice programmers used the system, learned from recommended tutors, and gave ratings to the tutors. The study tracked which of the internal algorithms gave each recommendation, so that we could determine which algorithm tended to give tutors that got higher ratings. The study showed that ratings of CBR-based recommendations were significantly higher than ratings of CF-based recommendations. Further analysis of study data suggested that the reason for this difference is that CBR's recommendations were more relevant and provided at points in time when the study

participants felt more ready to learn a given API. These results are important because they show how to more effectively teach a sequence of APIs.

Evaluation of Collaborative Filtering and Content Based Filtering Recommenders in jTutors (an online intelligent tutoring system).

by

Bharatwaj Appasamy

A THESIS

submitted to

Oregon State University

In partial fulfillment of

the requirements for the

degree of

Master of Science

Presented March 14, 2014

Commencement June 2014

Master of Science thesis of Bharatwaj Appasamy presented on March 14, 2014

APPROVED:

Major Professor, representing Computer Science

Director of the School of Electrical Engineering and Computer Science

---

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Bharatwaj Appasamy, Author

ACKNOWLEDGEMENTS

At the outset, I would like to thank my advisor, Dr. Christopher Scaffidi for his support and guidance. In the course of working with him, I have been able to learn a lot about the conduct of research in computer science, about how to approach problems and providing reasonable and testable solutions to problems. This experience of working with Chris has showed me what an immensely patient and thorough person he is and I regard those attributes as things I would want to emulate for success in my professional life.

I would also like to thank Dr. Alex Groce, Prof. Margaret Burnett and Dr. Carlos Jensen whose class work was exciting, interesting and challenging. Their courses helped me in learning some of the most important concepts in Software engineering and HCI.

I'd like to thank my fellow EECS students for interesting experiences and discussions that we have had over the years. I'd like to thank my research group, Chris Chambers, Sheela Surisetty, Vasanth Krishnamoorthy and Balaji Athreya for their support and involvement in my research work. I should also say thanks to Dr. Bella Bose and Nicole Thompson and other members of the EECS office who have provided me with necessary logistical guidance. Finally, I wish to thank my wife, Abhinaya and our parents for their unwavering support and unconditional love.

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF FIGURES

Figure Page

1 Introduction

Software engineering in practice today places a high focus on re-use of code, and APIs are components that implement specific functionalities to aid programmers in implementing specific features to a system without them having to do all the work from scratch. An application programming interface (API) is a particular set of rules (code) and specifications that software programs can follow to communicate with each other. It serves as an interface between different software programs and facilitates their interaction, similar to the way the user interface facilitates interaction between humans and computers. With the advent of online code repositories and the explosion of community coding and development, languages like Java, C, C++ have seen a great many number of APIs being published to help programmers implement functionality within a software system.

For example, one recent Java version has nearly 4000 classes/interfaces in its API library [1]. In addition to the standard Java library, there are thousands of other third party libraries published by both proprietary companies and the Open Source software communities [1].

Whereas an experienced Java programmer might be able to intuitively identify APIs to use for implementing features to a software system, the sheer number of APIs and components available makes it impossible for a programmer to be informed about all the APIs. In this context, the problem is magnified for novice programmers who are just beginning to learn programming, as they have to contend with not only the

number of APIs that are available for use (which has been identified as a problem in component re-use by the ACM [6]) but also the problem of choice since there are more than a few APIs available to implement similar functionality.

Whenever a programmer has to learn to use an API to complete a specific programming task, they need to be able to quickly identify what API to use and also understand the methods contained in them to implement requisite functionality [9]. Learning APIs on demand is different from other forms of learning in which the aim is to increase the general knowledge of the learner to prepare for future use [1]. In contrast, when the need for learning to use an unknown API arises for a programmer, their objective is to increase understanding just enough to implement required functionality to accomplish the pressing task before the programmer. Therefore, the process of learning to use a specific API on demand is highly personalized to the specific needs and existing knowledge of the programmer, and is tightly contextualized in the task and environment of the programmer.

Learning about an API is not easy. A recent study reported that many API methods are “inadequately documented”, forcing the programmers to search online for usage samples and discussions of API methods to aid them in understanding the working of API methods in order for them to implement it successfully [4,23]. Finding a good, easy to understand example for API method usage is often a time consuming and tedious process for programmers [11].

Our research group has previously developed a system called jTutors to deliver learning materials for API usage, called Intelligent API Tutors, in a comprehensible form for novice programmers, by bundling common usage patterns, documentation, queries and remarks of other programmers from coding communities like Stack Overflow, Dzone etc [5,15]. jTutors achieves this by combining code samples for API methods along with descriptions of what they achieve and providing them to the user in a coherent way. The users are also provided with small quizzes as they step through the tutor, in the form of blanks in the code where they are required to call the correct method from the API being presented to implement a short functionality within the code snippet. Students also have functionality to look at code examples (automatically retrieved from online open source repositories by jTutors) that demonstrate how to use APIs. Empirical studies showed that this was in fact an effective way for novice programmers to pick up on usage of API methods without making them feel daunted about the abundance of materials they would have to go through by simply doing an online search for said API methods [5,15].

However, jTutors to date has lacked a way to provide novice programmers with API tutor recommendations in order to enable them to discover more APIs for implementing functionality. Novice programmers often are not aware of what public APIs are available in general. For novice programmers to be confident about using a language as extensible as Java, it is required that they have an understanding of what APIs are available and have an opportunity to understand these API methods in an order that makes them comfortable about learning different APIs. Novice

programmers may also want to have an understanding of the most commonly used APIs for implementing mundane features in a software system like date processing, database connectivity, string processing, using files, IO streams, network connectivity etc.

Along with providing learning materials for API methods bundled together as API tutors, it is therefore essential for the system to provide these API tutors in an order that the programmers feel comfortable in so that their understanding and learning of different APIs can be enhanced. It is also necessary that a system like jTutors should provide API tutor recommendations so that novice programmers are given an opportunity for discovering APIs for use in implementing software engineering tasks.

In order to address the limitations of our research group's work to date, this thesis investigates the use of two algorithms to provide such API tutor recommendations to novice programmers using the jTutors system. Of the two algorithms, one of them is a collaborative filtering based approach that provides recommendations of API tutors based on likeness of the current user to other users of the system in their ratings of API tutors that the current user has completed. The second algorithm is a content based filtering algorithm that uses structures within the source code of samples used in a given tutor to determine which of the other API tutors are related to the given API tutor which the user is more likely to find useful for their needs.

An empirical study was conducted to evaluate the effectiveness of the recommendations that were provided by the algorithms in terms of the ratings given by the participants at each stage when the recommendation was made and subsequently completed by the participant. The results of the study were analyzed through standard non-parametric statistical tools. These analyzes revealed several strengths of the content-based recommendation algorithm over the collaborative filtering algorithm.

The results of this study could be useful in understanding what best ways to teach novice programmers a sequence of APIs. It could be used by course instructors to deliver teaching materials, by software organizations to train new hires on proprietary APIs, and to support novice programmers in a distance learning scenario.

This document is organized as follows: Section 2 covers a literature review of existing methodologies for providing API recommendations, understanding API usage patterns, delivering said recommendations to programmers and their effectiveness. Section 3 details the working of the prototype used for conducting the study and the mechanisms of the two recommender algorithms being evaluated. Section 4 outlines the experimental study and evaluation methods. Section 5 documents the results of the empirical study and also evaluates possible explanations for the results obtained, including threats to validity. Lastly, Section 6 concludes by stating contributions of this study and also provides a discussion on future research opportunities.

2. Background and Related Work

2.1 Existing jTutors System

The current jTutors system in place works by foraging for API usage samples, documentation and method descriptions from the internet and packaging all of those learning materials as a single unit which include descriptive texts of the method calls, showing actual code samples, annotations on presented code fragments along with quizzes in stages in order to help learning of API methods [5,15]. The tutors are created by experts (this could be teachers/instructors/TAs of a course, or any expert with knowledge of the API in question) who organize the tutor in a manner that is comprehensible to a novice programmer by placing these extracted samples and documentation in an order that aims to improve the understanding of the programmer who wishes to take the tutor to learn a particular API.

2.1.1 Process of creating an API tutor

When an expert wishes to create an API tutor, he/she clicks on the “create tutor” link in the website and enters a query in a search box. This query can be an abstract software engineering task (for e.g. “connect to a database”) or it can also be a specific class or API name (for e.g. Log4j). The jTutors system takes this query text and uses popular third party search engines (currently Google’s Custom Search API and Bing API) to get a list of code samples relevant to the search text. A web scraper module implemented within jTutors looks at the Top-N results from the search and catalogs all of the source code samples present within the <pre> or <code> tags in the

result pages. These code samples may be partial or complete code samples. The samples are then processed through a Java Parser provided by the Eclipse framework to develop compilation units which can be used to generate an Abstract Syntax Tree (AST). The ASTs provide information about method calls, imports, object invocations etc, which are then indexed by the class, method and type terms within the jTutors database. Relevant code samples are picked using the calculated TF-IDF co-efficient of these terms indexed and are shown to the expert in the tutor creation module.

The expert can then pick the most relevant samples from among the snippets provided and add descriptions, re-order snippets and also designate specific method calls as blanks intended as a quiz for the user to complete. The expert also has an option to enter code snippets themselves if there are not many useful samples retrieved through the search process (as can be the case in obscure or less widely used APIs). After going through all of the snippets, the expert is asked to enter a description of the tutor now created and give it a title. Once the expert completes this step, API tutor creation is complete and the new API tutor created is added to the list of available tutors.

2.1.2 Finding and taking API tutors

Users who wish to understand API usage for any given API can log in to the jTutors system and access saved tutorials by searching for their specific topic of interest or by looking at the list of tutors and selecting one that they want to take. When a saved tutorial is clicked in the list of tutorials, the user is taken to the start page of the tutor or the description page.

At the start of any tutor, an example code snippet of a method in an API is shown along with a description of what that method does. The learner can examine these examples to get an understanding of what the methods do before progressing to other stages in the tutor where they are prompted with blanks among other code fragments which are intended to test the learner's understanding of the API in view. The user can also click on a "get hints" button on the side when presented with a quiz page to get more information about what needs to be filled in the blank if the answer is not immediately obvious to the user. In addition, jTutors provides a search screen so that students can look for code examples that demonstrate how to use APIs. These code examples are retrieved from a cache of industry code examples, which are retrieved automatically by the system from an open source web repository (sourceforge.net) and indexed according to what Java classes they use [5].

2.2 The problem of API tutor discovery

While it is possible for users to peruse the list of available API tutors and also search for API tutors by key words, jTutors has lacked a way to generate automatic tutor recommendations. This might be a hindrance for novice programmers, as they would have to specifically know what API they want to learn and use appropriate keywords to search for the API tutor. Furthermore, there may be more than one tutor for a given topic or more than one API for a given programming task that novice users might find difficult to distinguish exactly which API tutor would suit their needs.

In the case of novice programmers who are casually learning APIs with the intention of increasing their understanding of a breadth of API topics, they would have

to identify a list of APIs that they might want to learn and then make a “lesson plan” (for themselves) consisting of API tutors available in the system and go through all of them in an order that helps them understand the usage of APIs conceptually. From the context of a novice programmer, this is a daunting task as they usually have very little prior knowledge of APIs in use or APIs that are popularly used in specific software engineering tasks.

2.3 Recommenders in the context of E-Learning Systems

Over the years, there has been some research on applying recommender systems to the field of e-learning and intelligent tutoring systems (ITS). Systems that can automatically guide the learner’s activities and intelligently recommend on-line activities or resources based on modeling the learner’s profile in terms of the access history of learners and navigation patterns using association rules mining have been studied [16]. While the topic of providing recommendations have been extensively studied for e-commerce systems, in the context of e-learning systems there are tangible differences in the objectives and techniques for providing recommendations that aim to improve the learning of the user [18, 19].

The majority of this work (cited above) has relied on association mining, which identifies objects frequently used in sequence, in order to recommend these sequences to users. A key problem with simple association rule mining is that just because a student used a certain object does not imply that the object contributed to learning. More sophisticated approaches rely on collaborative filtering-like algorithms, in which an algorithm receives a list of items, a list of people who have used each

item, and a “goodness” measure indicating how useful or appropriate the item was to each person who used it [19]. However, collaborative filtering algorithms have not yet been tested for use with recommending intelligent API tutors to computer science students. As a result, no empirical data exist regarding the effectiveness of this approach in this context.

With the issue of recommenders in e-learning, there is no one hard and fast rule for developing recommendation techniques. An entirely different approach is to use the specific characteristics of the specific content domain to generate content-based recommendations (e.g., as in [21]). In most cases, studies have been done on using the profiles of learners and guiding new learners to content based on the profiles of successful learners or making automatic recommendations based on an instructor’s intended sequence of navigation through course material [16]. Again, however, these content-based recommendation approaches have not yet been used to teach API usage. While the concept of Collaborative filtering can be readily implemented for jTutors considering items to be API tutors that users take, building efficient and effective content based recommenders for API tutors needs to be studied.

3. Approach

This section details the solution developed and used for evaluating the effectiveness of recommendations provided within the jTutors system. The repository was enhanced to track each student's history within the system. Specifically, features were added that each student can create an account, and the repository now tracks the list of tutors that each student uses. In addition, the user interface was enhanced so that the student can rate (on a 5-point scale) each tutor used, in terms of appropriateness for the student's needs and current knowledge at that moment in time. For example, if a student does a search for "send data via network," the search engine might return several tutors, each of which the student can rate after using them.

The repository was then further enhanced so that it can make personalized recommendations of intelligent tutors to each student. Specifically, two algorithms were implemented that each assigns a "recommendation score" to each tutor in the repository based on the appropriateness of each tutor, given the previous ratings given by the student to other tutors. The repository can use either one of these algorithms. That way, the user sees recommendations ordered according to one of the two algorithms based on the student's history up to that moment. Implementing two algorithms, rather than just one, made it possible later to gather data for an empirical study evaluating which of the two algorithms gave higher-rated recommendations.

The subsections below describe these two algorithms in detail, as they are the key technical contribution of this thesis. Section 3.1 outlines the collaborative filtering algorithm used for providing recommendations. Section 3.2 discusses the approach

used for developing a content based recommender for the tutors made available through jTutors. (Section 4.4 discusses how the algorithms' recommendations were integrated into the user interface for our empirical study.)

3.1 Weighted Slope-one predictor for collaborative filtering

The weighted slope one predictor by Lemire et al. [12] lends itself to be a usable candidate for generating collaborative filtering recommendations. The algorithm sufficiently satisfies the following requirements for integrating into the jTutors system for generating recommendations:

1. Updates nearly instantaneously: A new rating added to the system would change the predictions made for the next recommendation almost instantly
2. Query efficiency: Queries to the engine are fast albeit at a little more expense to storage
3. Cold start handling: The algorithm has been shown to be relatively adept at making reasonable recommendations to new users who have fewer ratings into the system to show their preferences

The slope one family of predictors considers both the ratings of other users for the same item and also the history of ratings of the current user in consideration before generating predictions [12]. The weighted slope one predictor specifically also considers the number of ratings observed for a given item and weighs the ratings accordingly thereby being superior to the generic slope one predictor, by mitigating

the effect of one of the drawbacks of the generic algorithm of having different numbers of ratings for items [12]. In the context of jTutors, the item is the tutor that a user takes and rates.

The weighted slope one prediction scheme for a user *u* for tutor *j* is given by:

$$P^{wS1}(u)_j = \frac{\sum_{i \in S(u) - \{j\}} (dev_{j,i} + u_i) c_{j,i}}{\sum_{i \in S(u) - \{j\}} c_{j,i}}$$

Where

$$\text{dev}_{j,i} = \sum_{u \in S_{j,i}(\chi)} \frac{u_j - u_i}{card(S_{j,i}(\chi))}$$

u_j, u_i are ratings for any two items j,i

$u \in S_{j,i}(\chi)$ is the set of user ratings for user *u*

χ is the training set of all user ratings

card(s) is the cardinality of a set *S*

and $c_{j,i} = card(S_{j,i}(\chi))$

Considering three tutors A, B, C, with the tutor pair A and C having 100 ratings versus the tutor pair B, C having 25 ratings, and a user U_1 who has rated tutors A and B. With the weighted slope one predictor, the user's rating for tutor A is used as a far better indicator for tutor C than the user's rating for tutor B.

$P^{wS1}(u)$ is the vector of predicted ratings for all tutors in the set. Once this vector is calculated from the algorithm, the predictions for all tutors that the user has already taken are removed from the set (in order to not provide a tutor that the user has already completed as a recommendation). The tutor with the highest rating prediction is provided to the user as a recommendation of the weighted slope one predictor.

3.2 Content-based tutor recommendations

The second algorithm makes recommendations by analyzing the relationships among the classes in the APIs. One can reasonably justify that there are always a combination of classes that are used together to make certain features work. For example, in projects that use Swing and AWT, if the component JButton (javax.swing.JButton) is used, then the button needs to be bound to a method that implements some function every time the button is clicked. ActionEvent (java.awt.event.ActionEvent) and ActionListener (java.awt.event.ActionListener) are two API classes in AWT events that provides for a mechanism to invoke specific actions when the conditions are met, like clicking a button in this case. So it is reasonable to say that a novice programmer learning to use JButton would also want to subsequently learn using ActionEvent in order to implement functionality with the button.

The idea behind the algorithm is to recommend classes that are related to classes that the student has already learned about. The algorithm works in two phases.

Phase 1 is the miner/network builder. This phase builds an index consisting of a weighted, undirected graph. There is one node in the graph for each Java class, and there is an edge between two classes if they reference each other. Our prior work on jTutors has shown how to automatically scrape open source projects from a web-based repository (sourceforge.net) [5], which are used to provide code examples to students who are using the tutors in the repository. In the case of Phase 1, we configured jTutors to retrieve 36 Open Source projects. For each Java class defined or referenced in these projects, jTutors now generates a node in the undirected graph. It builds a compilation unit using the AST java parser from eclipse JDT for the class. It retrieves the list of class imports for each compilation unit. Using a method visitor, finally, it adds an edge to the graph for each method call discovered. If such a link already exists, its weight is increased by one.

Phase 2 is the querying, which relies on a spreading activation model over the weighted undirected graph, which assigns an activation a_x to every class x. Recall that all tutors have a list of classes used in them. Given the set S_u of tutors that student u has already taken and rated higher than the student's average rating, jTutors retrieves the set of classes C_u referenced by the tutors in S_u. For each class c in C_u, jTutors calculates the link coefficient $LC_{c,d,u}$ to each other adjacent node and adds this $LC_{c,d,u}$ to a_d. The link co-efficient on edge (c,d) to another class d is computed as:

$LC_{c,d,u} = Rating_{u,c} * (\text{weight of edge } (c,d))/(\text{weight of all links } c)$.

For any class *c* in tutors that the user has already rated,

$Rating_{u,c}$ = average of user's ratings for those tutors in which *c* appears.

Thus, a_d ends up equaling the sum of all link coefficients over all links from classes that the student has already learned about in past tutors. The recommender picks the tutor with the highest score and propagates that to the UI.

4. Study design and Evaluation

This section seeks to provide details of the experimental study conducted for the evaluation of the two recommenders developed for content discovery within the jTutors systems. The design of the experiment, participation selection, overall methodology of the study, expectations of the participants and survey questions provided to the participants are discussed in detail.

4.1 Research Questions

The main objective of this research work is to investigate the effectiveness of a content based recommender and a collaborative filtering recommender in the context of the intelligent tutoring system jTutors.

It is not clear in the case of an e-learning system such as jTutors if a content based recommender may work better or worse than a collaborative filtering algorithm for recommendations. On the one hand, it makes sense to model a learning process for a user based on how other successful learners in the system have progressed, however, it also means that the system would need have enough prior successful learners to avoid the cold-start and sparsity problems that collaborative filtering recommender systems face whereas a content based recommender could provide clear deterministic recommendations to overcome these issues. In view of these points, this research seeks to ask the following questions:

RQ1: For the same set of tutors, does the content based recommender tend to give recommendations that fetch higher ratings than the collaborative filtering algorithm?

RQ2: For the same set of tutors, does the content based recommender provide recommendations that are perceived to be more relevant than the collaborative filtering algorithm's recommendations at a point in time when the recommendations are given?

RQ3: For the same set of tutors, does the content based recommender provide recommendations at a time that is more appropriate for the user than the recommendations of the collaborative filtering algorithm?

RQ4: For the same set of tutors, does the content based recommender provide recommendations that the users perceive as novel when compared to the collaborative filtering algorithm's recommendations?

4.2 Participants and Study recruitment

Since the jTutors system is intended to be used by novice programmers who are beginning to learn programming using Java and its APIs, the objective for recruitment of participants for the study was to have a set of users who were versed in programming in general but only were novice Java programmers. Recruitment was done by sending an email to instructors of the programming courses under OSU's computer science department to forward the email to their students with the recruitment text which pointed to a website where interested students could read the

consent form, and indicate a time slot which would comfortable for them to visit the laboratory for signing up for the study. A total of 25 participants were sanctioned to be recruited for the study by the IRB. All of the participants were CS majors.

4.3 Study Initiation and jTutors Tutorial

On the day the participants visited the laboratory, they were provided with the consent forms and given a walkthrough of the consent forms, perceived risks involved in taking part in the study (including data privacy concerns) and participation requirements for the study. Each participant was informed that they were required to take and complete as many API tutors as they could up to a maximum of 20 API tutors within a period of two weeks from the day they signed the consent forms. The participants also had the option of withdrawing from the study at anytime.

The participants were provided with a link to access the study website from their personal computers and were assigned a randomly generated user id (of the form userxxxxxx – where xxxxxx was a randomly generated 6 digit number) for access to the study website. The participants were allowed to pick their own passwords for access to the website when they were logging in to the system for the first time. The user id that was generated for each user was logged separately in order track the behavior of each participant within the jTutors system and to keep a record of how many API tutors each participant completed for the purpose of compensation for participating in the study.

Once each participant was initiated with access to the system, they were given a tutorial API tutor to familiarize them with the jTutors interface. The participants were informed that for the purpose of the study, completion of an API tutor meant that they had to finish the entire API tutor followed by the survey questionnaire at the end of each API tutor.

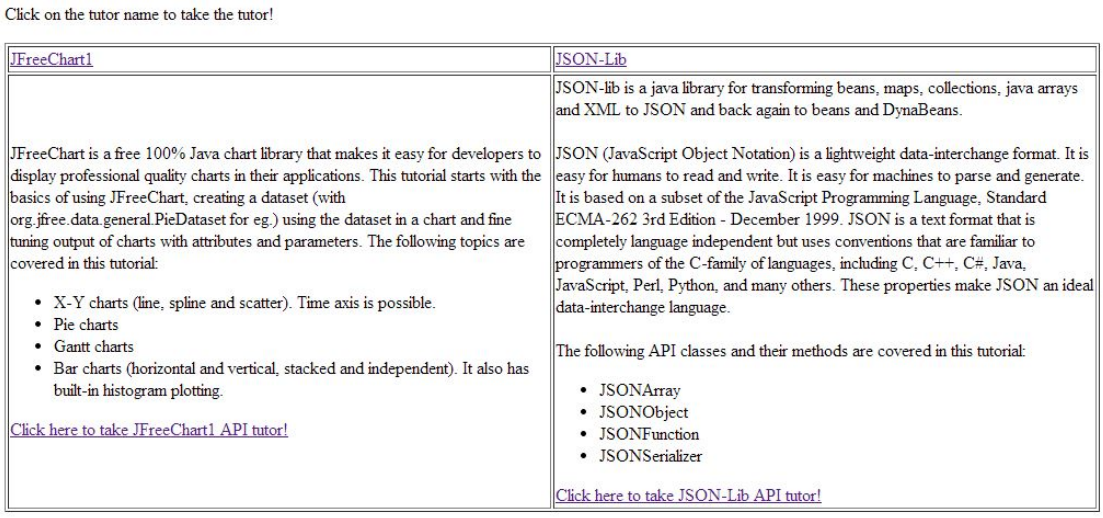
4.4 Integration of recommendations into the user interface

After completing the tutorial for understanding the jTutors interface, the participants were shown two API tutors alongside each other (in two columns, one on the right and the other on the left of the screen) each of which was generated as a recommendation from one of the two algorithms being evaluated. The participant was shown no indication of which algorithm recommended which API tutor and the order in which the recommendations were shown on the split screen was randomized each time the page was generated. For each API tutor shown to the user, the title of the tutor was displayed along with a brief description (from the initial page of the API tutor) of the contents of the API or tutor.

The participants were never shown the ratings of the API tutors that other users had assigned to the same if they had taken it. The participants could pick either of the two API tutors and complete it, after which they would shown two other recommendations similarly until each assigned participant had completed 20 API tutors or until such time that the participant chose to withdraw from participation in the study. Any time in the middle of taking an API tutor, the participants had the

option to save and quit and later resume at the same place or they could choose not to complete the API tutor mid way and were shown two new recommendations instead.

At each stage when recommendations were shown to the participants, the information regarding what recommendations were made by which algorithm, which recommendation was picked by the participant for completion and survey answers at the end of the API tutor were logged. In addition to this, the jTutors systems also logged other meta-information such as the time taken for completion of each API tutor and number of attempts for blanks within each stage of the tutor along with incorrect answer attempts.


Click on the tutor name to take the tutor!

JFreeChart1	JSON-Lib
JFreeChart is a free 100% Java chart library that makes it easy for developers to display professional quality charts in their applications. This tutorial starts with the basics of using JFreeChart, creating a dataset (with org.jfree.data.general.PieDataset for eg.) using the dataset in a chart and fine tuning output of charts with attributes and parameters. The following topics are covered in this tutorial: • X-Y charts (line, spline and scatter). Time axis is possible. • Pie charts • Gantt charts • Bar charts (horizontal and vertical, stacked and independent). It also has built-in histogram plotting. Click here to take JFreeChart1 API tutor!	JSON-lib is a java library for transforming beans, maps, collections, java arrays and XML to JSON and back again to beans and DynaBeans. JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language. The following API classes and their methods are covered in this tutorial: • JSONArray • JSONObject • JSONFunction • JSONSerializer Click here to take JSON-Lib API tutor!

Figure 1: Screenshot showing top recommendation made by the CF and CBR each displayed to a participant

4.5. Survey Questions

After each API tutor was completed, the participants were asked to rate the API tutor that they had just taken on a 5-star rating scale and were also shown a set of

3 survey question statements that they had to answer on a scale of 1 to 5 depending on how much they agreed.

The three questionnaire statements shown to the participants were:

1. This tutor helped me discover APIs that were relevant to my needs.
2. I felt ready to take this tutor.
3. I already knew about the APIs in this tutor.

The first question was intended to determine the relevance of the current API tutor recommended to the user with respect to the user's own history and also to determine if the recommendation was perceived as within the interests of the user taking the API tutor. The second question was meant to determine if the API tutor was recommended at a point in time when the user felt comfortable in taking the tutor and also with respect to any other API tutors that might have to be taken before the user is able to comprehend the contents of the currently recommended tutor. The third question was meant to determine the novelty value of the recommendation made in terms of helping the user discover API tutors that they might not have otherwise discovered themselves.

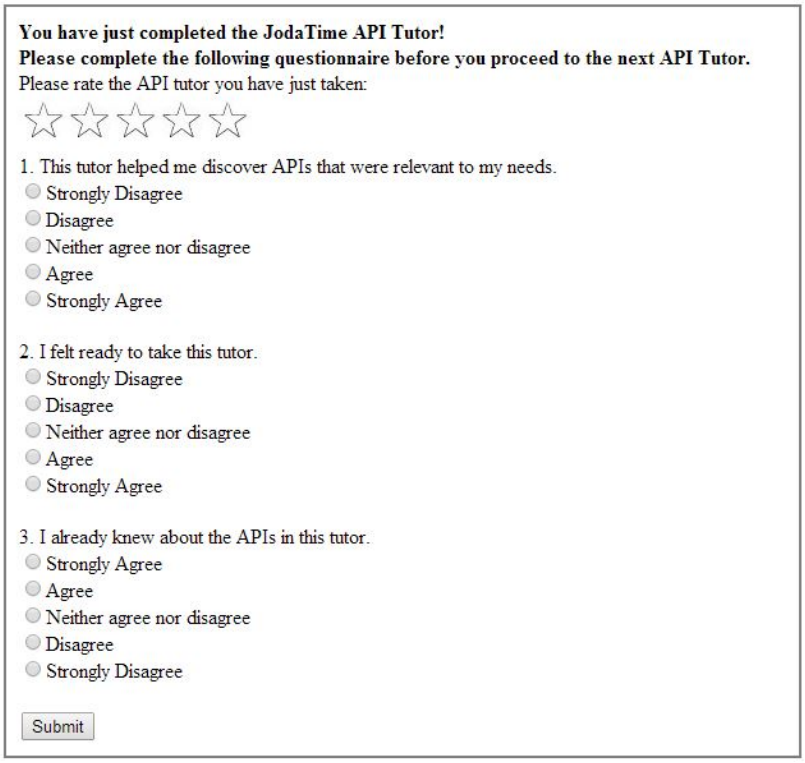


Figure 2: Screenshot showing the tutor rating bar and survey questionnaire after a participant completes an API tutor

4.5 Data analysis

Recall that, for a given tutor i, each might recommend i at certain points in time to certain users, while the algorithm would recommend i at other points in time to other users. The repository recorded the rating $r_{u,i}$ given by each user u to each tutor i, regardless of which algorithm recommended i at that moment. The study was aimed at determining which algorithm was better at recommending i at appropriate moments. Therefore, to analyze these data, a value CF_i was computed for each tutor i as the mean of $r_{u,i}$ over all situations when i was recommended by the collaborative filtering algorithm; likewise, a value CB_i was computed as the mean of $r_{u,i}$ over situations when i was recommended by the content-based algorithm. This made it possible to compare

the two algorithms while controlling for inter-tutor differences. Specifically, the pairs (CF_i, CB_i) were compared using the non-parametric Wilcoxon two-tail rank sum test (recognizing that the data would not necessarily be normally distributed). Analogously, we compared the two algorithms based on each of the three survey questions (looking for a $p < 0.05$ cut-off).

4.6 API Tutor Topics

The goal of the study was to evaluate the effectiveness of API tutor recommendations to users and hence it was important to have a wide selection of API topics that cover many of the common scenarios in which programmers would want to use third party APIs for implementing features. A total of 20 third party API topics that were most commonly re-used was identified [13,14] and a total of 26 API tutors to cover the most important features and methods over these 20 libraries were created. The following 20 libraries were covered:

- Apache Commons Lang
- Commons Lang 2
- Jsoup
- GWT
- Google Guava SDK
- JFreeCharts
- Apache DBUtils
- Standard Widget Toolkit

- OpenNLP
- WALA
- GSON
- JSON-lib
- JDBC
- Log4j
- JodaTime
- Swing Components
- Swing Events
- Eclipse RCP
- Java.util.regex
- Java.sql
- Commons Math

5. Experimental Results

A total of 25 participants were recruited for the experimental study conducted over a period of 3 weeks in two phases. In the first phase 13 participants were recruited and initiated into the study. In the second phase, a total of 12 participants were recruited. A total of 481 ratings and survey responses (after completing API tutors) were obtained during the course of the study.

5.1 Results

RQ1: For the same set of tutors, does the content based recommender tend to give recommendations that fetch higher ratings than the collaborative filtering algorithm?

There was strong evidence (p=0.0006) for a significant difference between ratings given to recommendations made through the content based recommender and the recommendations made by collaborative filtering algorithm for the same tutors. This indicates that when the recommendations were made by the content-based recommender, the participants tended to give higher ratings to the tutor than when the recommendation was made by the collaborative filtering algorithm.

RQ2: For the same set of tutors, does the content based recommender provide recommendations that are perceived to be more relevant than the collaborative filtering algorithm's recommendations at a point in time when the recommendations are given?

There was strong evidence (p=0.002) for a significant difference between the scores assigned to the *relevance* question in the survey between the recommendations made by the content based recommender and the recommendations made by the collaborative filtering recommender for the same tutors. This shows that the users tended to find the recommendations of the content-based recommender more relevant to their needs than the recommendations provided by the collaborative filtering algorithm.

RQ3: For the same set of tutors, does the content based recommender provide recommendations at a time that is more appropriate for the user than the recommendations of the collaborative filtering algorithm?

There was strong evidence (p=0.03) for a significant difference between the scores assigned to the *appropriateness* question between the recommendations made by the content based recommender and the collaborative filtering recommender for the same tutors. This shows that the users felt more comfortable that the content-based recommender provided recommendations at a time when the user was more comfortable to take a tutor.

RQ4: For the same set of tutors, does the content based recommender provide recommendations that the users perceive as novel when compared to the collaborative filtering algorithm's recommendations?

There was not strong evidence (p=0.07) for a significant difference between the scores assigned to the *novelty* question in the survey between the recommendations

made by the content based recommender and the collaborative filtering algorithm for the same tutor. This indicates that the evidence is inconclusive on whether the users perceived the recommendations made by the content-based recommender to be something formerly unknown as compared to the recommendations made by collaborative filtering algorithm at points in time when the recommendations were made, for the same set of tutors.

The figure below summarizes the scores for the algorithms on each survey question and for the overall rating, averaged over all recommendations made by the corresponding algorithm. The collaborative-based recommender clearly outperformed the collaborative filtering algorithm.

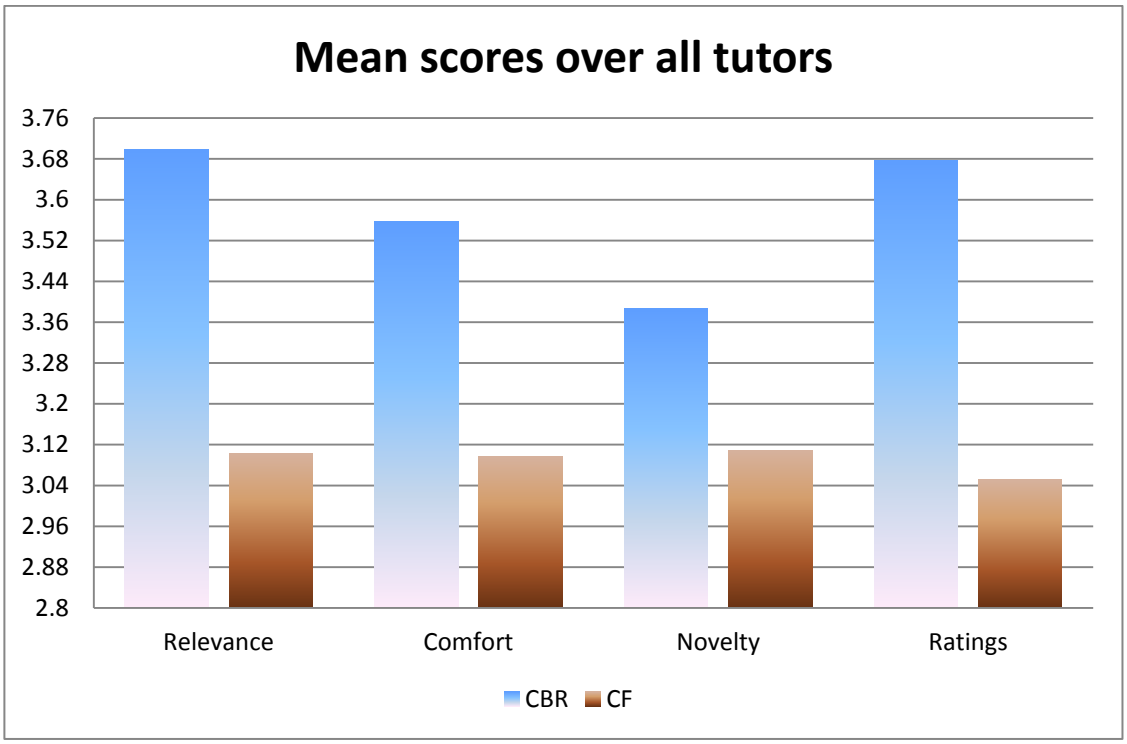


Figure 3: Means scores for survey questions over all tutors

5.2 Discussion

Results from this research work did indicate that the users perceived recommendations made by the CBR to be more relevant to their needs based on their history and that they were made at a time when the users felt they were ready. These findings can also be explained to an extent by the looking at the order in which users had taken tutors through the course of the study.

It is notable that the CBR made recommendations for API tutors that are logically next in the series as opposed to the CF algorithm which made recommendations across topic lines. For example, whenever the CBR recommended tutor 4 (GWT 1) it was followed by CBR recommending tutor 5 (GWT 2) or vice versa. The same observation is also present with tutors 10 and 11 (SWT 1 and 2), tutors 24, 25 & 26 (Commons Math) and tutors 1 & 2 (Commons Lang). The users who had taken any one of the tutors in the series followed up by picking the recommendation of the other tutors in the series when it was recommended by the CBR. This shows an example of a scenario where the CBR makes recommendations within the same topic.

In some cases, the CBR also tended to recommend API tutors that are off topic but logically related, as in the case of tutor recommendations for Apache DBUtils when the user had complete the JDBC tutor, or recommendations for GSON or JSON-Lib after JDBC or DBUtils. In this case, although they are different libraries, in many use cases, source files had combinations of methods where JDBC was used to interface with a database and then the result set returned was manipulated using

GSON or JSON-lib to send data to a client in JSON format. This example also helps to explain CBR recommendations that were perceived as relevant to what the users had taken previously, although not within the same topic, but logically related through their function.

With respect to the *comfort* (appropriateness) question in the survey, the amount of retries needed before users got answers right to quiz questions could be an indicator of how valid their comfort score was. On average, users who gave a comfort rating of 4 or 5 tended to answer the quizzes correctly in the first attempt or with 1 retry at the most, whereas recommendations that received a comfort score of 1 – 3 tended to have users who needed 2 or more retries before arriving at the correct answer to a blank.

It should also be noted that the average ratings for the CF algorithm's recommendations had a tendency to be elevated as more ratings were added to the system. This could be due to the fact that the CBR made some initial recommendations that were well received by the users which in turn influenced the CF algorithm's recommendations as the study progressed. In this regard, it can be said that the CBR can be used as a way to alleviate some of the cold start issues that the CF algorithm has, although this hypothesis would need to be investigated further through future studies.

6. Conclusion

Our study showed that ratings of content-based recommendations were significantly higher than ratings of collaborative-filtering-based recommendations. Further analysis of study data suggested that the reason for this difference is that content-based recommendations were more relevant and provided at points in time when the study participants felt more ready to learn a given API.

These results are important because they suggest a process to more effectively teach a sequence of APIs. The results could be applied in several ways. First, they could be used by professors to determine how to effectively cover APIs in a course. Second, a company that needs to teach new employees about corporate APIs could follow the content-based recommendations algorithm to bring the new employees up to speed. Third, the underlying jTutors system could be used to cover other languages and a broad range of APIs, then deployed to the world at large in order to help people learn programming more easily via the internet. Fourth, a company such as Amazon that has a large collection of public-facing APIs could implement a system similar to jTutors to provide better training materials for programmers that the company wants to entice into using those APIs.

Apart from addressing API discovery issues, the content based recommender could also be used as a tool for providing information about what API tutors could be added to the jTutors system. It would be possible to determine any gaps in tutor coverage by looking at APIs/classes that are frequently used together (mined from the open source projects) but currently do not have any API tutors in the system. Further

studies in this regard may be conducted to explore the validity of using mined informaton to enable tutor creators (users with privileged accounts, like a TA or an instructor for instance) to enhance the system by adding more content or improving existing content.

7. Bibliography

[1] Ye, Yunwen, et al. (2007) Searching the library and asking the peers: learning to use Java APIs on demand. *Proceedings of the 5th international symposium on Principles and practice of programming in Java*.

[2] Begel, A, and Simon, B. (2008b) Struggles of new college graduates in their first software development job. *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*.

[3] Brandt, J, Guo., P, Lewenstein, J., Dontcheva, M., and Klemmer, S. (2009) Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. *Proceedings of the 27th International Conference on Human Factors in Computing Systems*.

[4] Robillard, M, and Deline, R. (2010) A field study of API learning obstacles. *Empirical Software Engineering*, 1-30.

[5] Dahotre, A., Krishnamoorthy, V., Corley, M and Scaffidi, C, (2011) Using intelligent tutors to enhance student learning of application programming interfaces. *ACM Journal of Computing Sciences in Colleges, ACM Consortium for Computing Sciences in Colleges, 27,* 1, 195-201.

[6] What is Java technology and why do I need it? http://www.java.com/en/download/faq/whatis_java.xml (retrieved on May 24, 2012)

[7] Davor C. Cubranic and G.C. Murphy, (2003) Hipikat: Recommending Pertinent Software Development Artifacts, in *Proceedings of the 25th International Conference on Software Engineering (ICSE03)*. 408-418.

[8] Lange, B.M. and T.G. Moher. (1989) Some Strategies of Reuse in an Object-oriented Programming Environment, in *Proceedings of Human Factors in Computing Systems*.

[9] Mandelin, D., et al., (2005) Jungloid Mining: Helping to Navigate the API Jungle, in *Proceedings of 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation*. 48-61.

[10] Mili, A., et al., (1999) Toward an Engineering Discipline of Software Reuse. *IEEE Software 22-31.*

[11] Raymond, E.S., (2004) *The Art of UNIX Programming*. Boston, MA: Addison-Wesley

[12] Daniel Lemire, Anna Maclachlan, (2005) Slope One Predictors for Online Rating-Based Collaborative Filtering, In *SIAM Data Mining* (SDM'05).

[13] Stylos, J, and Myers, B. (2006) Mica: A web-search tool for finding API components and examples. *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, 195-202.

[14] The Most Widely Used Third Party Java Libraries by X Wang : http://www.programcreek.com/2011/08/the-most-widely-used-java-apis/ (last retrieved on 11/24/2013)

[15] Krishnamoorthy, V., Appasamy, B., and Scaffidi, C. (2013). Using intelligent tutors to teach students how APIs are used for software engineering in practice. *IEEE Transactions on Education, 56*, 3, 355-363.

[16] Zaíane, Osmar R. (2002) Building a recommender agent for e-learning systems. *Computers in Education.*

[17] Khribi, Mohamed Koutheaïr, Mohamed Jemni, and Olfa Nasraoui. (2008) Automatic recommendations for e-learning personalization based on web usage mining techniques and information retrieval. *Eighth IEEE International Conference on Advanced Learning Technologies*.

[18] Romero, Cristóbal, and Sebastian Ventura. (2007) Educational data mining: A survey from 1995 to 2005. *Expert Systems with Applications*, 33.1, 135-146.

[19] Baker, Ryan SJD, and Kalina Yacef. (2009) The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining*, 1.1, 3-17.

[20] Zhuhadar, Leyla, et al. (2009) Multi-model ontology-based hybrid recommender system in e-learning domain. *Proceedings of the 2009 IEEE/WIC/ACM*

International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 03.

[21] Meteren, R. V., & Someren, M. V. (2000). Using Content-Based Filtering for Recommendation. *MLnet / ECML2000 Workshop*, May, Barcelona, Spain.

[22] Pahl, C., & Donnellan, C. (2003). Data mining technology for the evaluation of web-based teaching and learning systems. In *Proceedings of the congress e-learning*, Montreal, Canada

[23] Robillard, Martin P. (2009) What makes APIs hard to learn? Answers from developers. *Software, IEEE* 26.6, 27-34.