International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2014): 5.611

A Survey on Extended MI technique for Edit Recommendation using Hybrid History Mining and Relevance Feedback

Shradha P. Patil[1], B. Padmavathi[2]

[1]P. G. Student, Department of Computer Engineering, GHRCEM, Wagholi, Pune, India

[2]Professor, Department of Computer Engineering, GHRCEM, Wagholi, Pune, India

Abstract: ***Now a day's use of recommendation systems while developing software increasing in order to speed up the process of software development by software developers. Accurate recommendations leads to successful, faster, efficient development, but inaccurate recommendations can lead to inappropriate, missed deadline software development. To guide programmers, researchers have developed history-based recommendation systems following two approaches either by mining view history or by mining edit history. However these methods failed to achieve the accuracy, flexibility and early recommendations. These problems are overcome by recently presented method called MI which is recommendation system extending ROSE. But the limitation of MI is that no end user satisfaction is taken into the considerations, and hence there is always scope for improvement in accuracy. In this system we are presenting EMI (Extended MI) technique in which we are improving the accuracy by relevance feedback method, in which log of feedbacks should be maintained and based on end users feedbacks, proposed system can refine and regenerate more accurate recommendations next time for same query with less time.***

Keywords: Association rules, Context formation, Data mining, Mining programmer interaction histories, ROSE

1. Introduction

The main goal of using recommendation systems is to gain the productivity of developer by recommending files to edit. Association rules mining is done for such in software revision histories. On the other hand, mining coarse-grained rules using only edit histories produces recommendations with low accuracy, and can only generate recommendations after a developer edits a file. The existing methods presented are falls in two categories such as view history based mining or edit history based mining. But due to the less accuracy, more time for generating recommendations, there is always research problem in this domain whether which history is better to mine. To overcome these limitations, recently MI technique is presented to produce recommendations from both view and edit histories. This method practically showing better accuracy, flexibility and speed, but there is no end user satisfaction achieved with this method. This becomes research challenge in this domain. So we have proposed relevance feedback method to achieve end user satisfaction. Using detailed edit and view histories to recommend files to edit produces the following advantages:

a) **Accurate recommendations**: The approach that considers viewed file provides more accurate recommendation over approach that considers only edited files.
b) **Early recommendations:** Programmer can recognize files to edit early, even though he has not edited a single file before
c) **Flexible recommendations:** When recommendations generate based on viewed files the recommendations change in response to programmer's exploration paths.

2. Related Work

R. Robbes and M. Lanza, *"Characterizing and understanding development Sessions,"*-To reconstruct development sessions we have implemented an evolution monitoring prototype which records all semantic change performed by the user so, to understand and characterize the development sessions we used the fine-grained information as they were carried out on two object-oriented systems[1].

M. Kim and D. Notkin, *"Discovering and representing systematic code changes,"* - We proposed a tool - Logical Structural Diff that assumes systematic structural differences as logic rules. Logical Structural Diff form systematic change pattern by grouping code changes regardless of their distribution [2].

R. Agrawal, T. Imielinski and A. N. Swami, *"Mining association rules between sets of items in large databases,"* presents an efficient algorithms which develops all significant association rules between items in the database as well as to solve the problem of mining a cluster of basket data type transactions for association rules[4].

T. Zimmermann, P. Weissgerber, S. Diehl and A. Zeller, *"Mining version histories to guide code changes,"* - To guide programmers along with related changes we apply data mining to version histories. We are using the set of existing changes which suggest and predict the similar further changes, item coupling that is invisible by program analysis and prohibit errors that occurs due to incomplete changes. After first change, our ROSE technique can correctly predict 26% of new files to be changed [5].

A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, - *"Predicting source code changes by mining change history,"* - To help a developer we have described new approach of mining revision history for identifying appropriate source code for a change task. We are applying our approach to two open-source systems, Eclipse and Mozilla, that provide useful recommendations and then evaluating the results based on the predictions and likely interest to a developer. Moreover to provide evidence for our hypothesis, we have developed a set of interestingness criteria for assessing the utility of recommendations that can be used in qualitative analyses of source code recommendations [6].

3. Proposed Work

To recommend files to edit by using the records of viewed files, MI mines programmer interaction histories that are history database. As shown in Fig. 1, MI mines interaction traces and finds association rules using the current context. To improve the accuracy of recommendation we are using relevance feedback method, in which log of feedbacks should be maintained based on end users feedbacks. If the end users are satisfied with recommendation generated by MI , it sends that to the programmer else proposed system can refine and regenerate more accurate recommendations next time for same query with less time. The fundamental part of the recommendation system is the *context*. For recommendation system, context characterizes the information about the user, their work and environment (e.g., viewed files), that are present at the time of recommendation and is used as a query at the time of further recommendation. To form the context we are using context formation algorithm.

3.1 MI (Mining Programmer interaction Histories)

MI extending ROSE which is an approach that mines software revision histories. We have revised ROSE to mine programmer interaction histories. The revised ROSE forms a context using only edited files by mining the association rules from edited files in programmer interaction histories. To recommend files to edit we have used this version of ROSE to enclose viewed files, we have proposed new approach MI, for mining association rules in programmer interaction histories (MI)[3]. MI forming a hybrid context of viewed and edited files by mining the association rules from both viewed and edited files. This MI approach can use several methods to form a context from edited and viewed files.

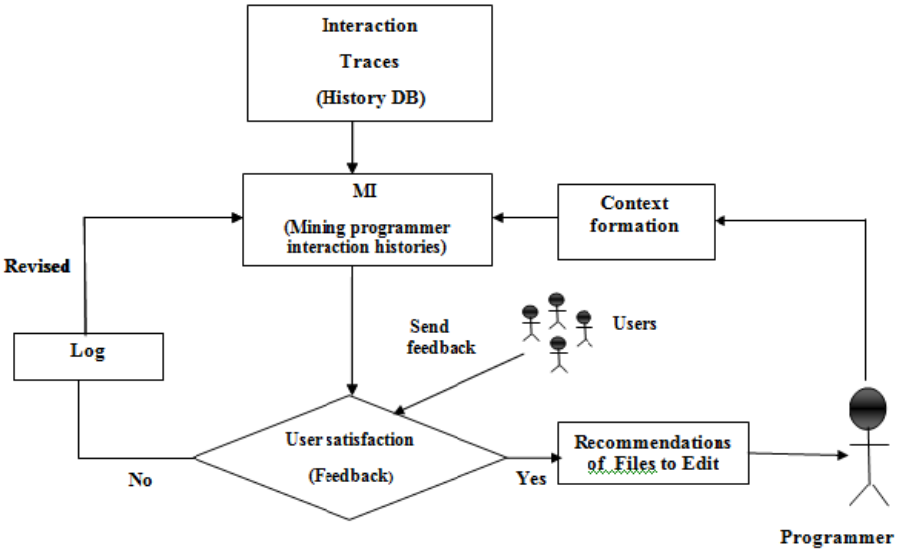

Figure 1: Architecture of the proposed recommendation system EMI

The limitation of MI is that no end user satisfaction is taken into the considerations, and hence there is always scope for improvement in accuracy. So we have proposed extended MI technique in which the end user can send the feedback. The log of feedback is maintained. We are applying filtering technique on log record so that it can refine and regenerate more accurate recommendation next time for the same query in less time.

3.2 Interaction Traces

An interaction trace is information consists of the records that define a programmer's actions (i.e. views and edits) and files on which the actions were taken.

An interaction trace Tk is transformed into a pair of sets $Tk = (Vk, Ek)$, where Vk is the set of viewed files in interaction trace Tk.
$Vk = \{v1, ...,vn\}$ and Ek is the set of edited files in interaction trace Tk .

The collected information of interaction traces can be expressed as $History\ DB = \{Tk | 1 \leq k \leq i\text{-}1\}$.

3.3 Context Formation

Conceptually, a context specifies information which can be used to describe the situation of a current user. In a recommendation system, a context forms a query, which generates a recommendation. In EMI, a context is formed from current actions of a programmer's. When the current programmer is carrying out a task m, a context is created

from the last files edited and viewed by the current programmer from each time-point in *Tm*. If the current programmer keep on viewing and editing files, the context get change. The context *C* can be defined as *(Vc, Ec)*, where *Vc* is a set of the last viewed *v* files of a current programmer.

Vc = {v1, ..., vn}, and *Ec* is a set of the last edited files by programmer. *Ec = {e1, ..., en}* at each timepoint.

4. Scope of Work

Main goal of this system is to present improved method for recommendation systems using explicit log based relevance feedback scheme.

- To present literature review of different techniques of recommendation systems.
- To present limitations of existing techniques.
- To present proposed algorithms and framework.
- To present practical analysis and performance evaluation

5. Conclusion

In this work, we have investigated how the use of view information collected from programmer interaction histories, can help provide detailed context to programmer to get more accurate, earlier and more flexible edit recommendations. To evaluate this, we proposed new approach MI, which extends previous approach ROSE, by moreover considering the records of viewed files. Additionally we have presented extended MI technique to improve the accuracy by relevance feedback method which maintains the log of feedback's by the end users.

References

[1] R. Robbes and M. Lanza, “Characterizing and understanding development Sessions,” *Proc. 15th IEEE International Conf. on Program Comprehension (ICPC '07),* IEEE Computer Society, Washington, DC, USA, 2007, pp. 155-166.

[2] M. Kim and D. Notkin, “Discovering and representing systematic code changes,” *Proc. 31st International Conf. on Software Engineering (ICSE '09),* IEEE Computer Society, Washington, DC, USA, 2009, pp. 309-319.

[3] Seonah Lee, Sungwon Kang, Sunghun Kim, and Matt Staats “The Impact of View Histories on Edit Recommendations”, IEEE Transactions on Software Engineering,(Volume:41 , Issue: 3),March 1 2015

[4] R. Agrawal, T. Imielinski and A. N. Swami, “Mining association rules between sets of items in large databases,” - *Proc. ACM* D.C., May 26-28, 1993, ACM Press, pp. 20–216.

[5] T. Zimmermann, P. Weissgerber, S. Diehl and A. Zeller, “Mining version histories to guide code changes,” *IEEE Transactions on Software Engineering*, 31(6), 2005, pp. 429–445.

[6] A. T. T. Ying, G. C. Murphy, R. Ng and M. C. Chu-Carroll, “Predicting source code changes by mining change history,” *IEEE Transactions on Software Engineering*, 30, 2004, pp. 574–586.