Learning Classifier Systems as Recommender Systems

Sooraj K Babu and Sebastian von Mammen

Games Engineering, Julius-Maximilians University, Würzburg, Germany
{sooraj.kandathil-babu, sebastian.von.mammen}@uni-wuerzburg.de

Abstract

A multitude of software domains, ranging from e-commerce services to personal digital assistants, rely on recommender systems (RS) to enhance their performance and customer experience. There are several established approaches to RS, including algorithms, namely, collaborative filtering, content-based filtering, and various hybrid systems that use different verticals of data, such as the user's preferences, the metadata of items to be suggested, etc., to generate recommendations. Contrary to those, we adopt a classifier system to benefit from its rule-based knowledge representation and interwoven learning mechanisms. In this paper, we present our approach and elaborate on its potential use to flexibly and effectively guide a user through the application of an authoring platform.

Introduction

Assisting novices in using a complex software system, be it a simple interactive web application or a sophisticated game engine, can be a challenging task (Oshita et al., 2019). The conventional and widespread methods to address user assistance in a complex system are either text-based, such as software documentation in the form of paper manuals or interactive websites (Mirel, 1998), or audio-visual assistance either directly through the platform or online forums and social media such as YouTube (Van der Meij and Van Der Meij, 2014). In some cases, there are intelligent systems that can identify user states and react accordingly (Kao, 2020). Most of the time, users of a software system, specifically novices, will have to spend a sensible amount of time finding a solution to their problems. It might even be challenging to understand the usage situation in some cases. Mainly, in the case of complex software such as an authoring platform, there is a high likelihood that the type of functionalities it offers may confuse a user. We propose using an RS as an intelligent user guidance mechanism to address this. An RS, by utility, is an essential tool that makes any user not feel overwhelmed with information overload (Selimi and Nuci, 2020). In a typical case, an RS considers a set of data points to make recommendations. Depending on the RS approach, this can range from user preferences to the knowledge of any item to be recommended. In this work, the RS captures the user requirements, classifies them, and delivers appropriate recommendations back to the users to fulfill their needs. This task can be performed by a classifier system, which gives specific recommendations based on the classification of a user's given situation (Woźniak et al., 2014). Assuming that users should be given different recommendations based on their backgrounds, goals, etc., and that both the users and the software might change over time in terms of behaviours and usage, a learning classifier system (LCS) could adapt and improve its recommendations based on experience over time (Urbanowicz and Browne, 2017). Means to direct a user's attention are captured by the term 'guidance' (Badjio and Poulet, 2005). In the context of this work, we limit the scope of according intelligent guidance in an authoring platform to the tasks of (1) familiarisation (overall workflow, functionalities, and UIs) and (2) usage of a software system (e.g., various modeling, composition processing, or analytics tasks). Apart from classifying the information we need out of a vast amount of data, an RS is a generic solution to receive expert feedback (Ricci et al., 2010). However, an RS is hardly utilized to assist a user with contextual recommendations to familiarize them with complex software or to help them use it thoroughly. To fill this gap, we explore primarily two research questions through this work: RQ1) How to guide a user through different steps in using a complex software system by employing an RS? RQ2) How can an LCS be adapted to develop a practical and generic rule-based RS for user guidance? The following sections briefly describe the related work on RS approaches and user assistance, the methodology we have adapted, our currently implemented concept and its integration into an authoring platform, along with the evaluation plan.

Related Work

The logic behind widely used RS approaches can be conceptually associated with rule-based systems. For example, algorithms such as collaborative filtering and content-based filtering classify the items to be recommended either by evaluating user interactions (Koren et al., 2022) or by assessing heterogeneous information about the items themselves (Aggarwal, 2016). Meaning, *If* a user requirement falls under any of these classifications, *then* specific recommendations are delivered to the users from these classifications. Such conditional statement, as explained above, are typically hard-coded part of the respective RS algorithms and only different classifications of user data result in differently configured recommendations. In contrast, some approaches rely on rule representations to make recommendations. Padmanabhan et al. (2011), for instance, rely on a decision list rule learner to recommend TV shows to the user. Abel et al. (2008) designed a rule-based system to sort and recommend relevant threads from lengthy online discussion forums. Sharma and Kaur (2015) ask the user for inputs, convert them into rules, and use contextual information to provide location-based services to the user. All these works and most of the research we found during the literature review were either based on established RS algorithms or hybrid methods. They have proven to be effective in their domains. However, to the best of our knowledge, LCS had not been considered for implementing RS before. At the same time, the applied nature of LCS brings various advantages to RS, most importantly the flexibility to adjust to changes in user data, a given application context, and available recommendations over time. As we will detail in this paper, LCS also allows us to represent different RS approaches simultaneously, which may reduce the maintenance costs of an according software framework, increase its applicability, or even support cross-fertilization of different RS concepts.

Basic Representations

A $User$ of a software service and the RS that support him/her can be represented as two agents communicating with each other (Afsharchi et al., 2006). As defined by Denzinger and Kidney (2006), an agent Ag can be represented as a quadruple (Sit, Act, Dat, f_{Ag}) where Sit is the set of situations an Ag can be in, Act is the set of actions Ag can perform, Dat is the set of data Ag can rely on, and $f_{Ag} : Sit \times Dat = Act$ yields Ag's action in a given situation, provided certain data. Following this definition, Sit of the $User$ agent would capture all the situations the user may be in, i.e. different stages in different use cases. Act captures the set of inputs the $User$ can provide to navigate, manipulate, select, or control the software in use (Sutcliffe, 2000), Dat consists of all the data the $User$ has about one's own goals and data that could be provided, about the software system, including for instance how it works, previously stored files, the currently displayed information, etc. Likewise, the Sit of the RS agent would consist of all activities $User$ is performing across different use cases. Act contains recommendations to the $User$ agent to navigate, manipulate, select or control the system, and Dat contains all data inferred from user interaction, and knowledge and metadata of the software components and use cases. In summary, the knowledge bases, Sit, Act, and Dat, of both agents contain a vast amount of contextual information from different use cases to serve each other. They would ideally contain information such as the user's intentions of utilizing the system, necessary awareness of the task being carried out, the decisions a user makes, the data extrapolated from user interaction patterns, the knowledge of all the system components, the metadata of all the objects that drives the system, etc.

Implementing the function f_{Ag} of the RS agent specifies which recommendation it can provide to the $User$; and in this case, an LCS does this, which will be detailed in the next section.

Methodology and Concept

XCS is one of the most investigated learning classifier systems (Stein et al., 2020). Compared to the original LCS approach, the primary change in XCS is in the definition of classifier fitness. Fitness is a value that determines the worth of a classifier. As a result, XCS tends to strengthen classifiers that are both accurate in making predictions and maximally general. This change results in the population having more strong classifiers and the system delivering high performance, which is crucial for an RS. In addition, the system representation is compact compared to traditional LCS. Also, the genetic algorithm works on the action set, whereas conventional LCS considers the whole rule population (Wilson, 1999). This makes XCS effective computationally. As with every LCS, an XCS classifier has a condition and an action part. Condition in this context consists of a set of inputs captured from the users (if the user grants respective permissions). The corresponding action part can be contextual depending upon the type of action the user is performing. For example, if the user is familiar with a game engine, the inputs captured (condition) could be the interaction information pattern indicating what the user is planning to do. The corresponding recommendation (action) could be information on the intended task in the form of a tooltip or an appropriate guiding cue. A classifier can be directly translated to the function $(f_{Ag}(RS))$ that provides an agent's action in any situation. Both the classifier and the function $f_{Ag}(RS)$ process similar data to assess a user's state and assist with recommendations (See Fig. 2). An agent's Sit and the Dat it needs to decide what action to take are the same as the attributes of a condition analyzed by a classifier.

Multi-instance Approach: User guidance in a complex system should serve different use cases in a usual scenario. Considering an authoring platform as an example, these use cases could be familiarisation of tools, various composition mechanisms, UI navigation, and live assistance for various tasks the platform could offer. Having an RS with knowledge about the user base and the whole software platform could support users with all the above use cases. However, in an XCS-based RS, a single rule population that contains

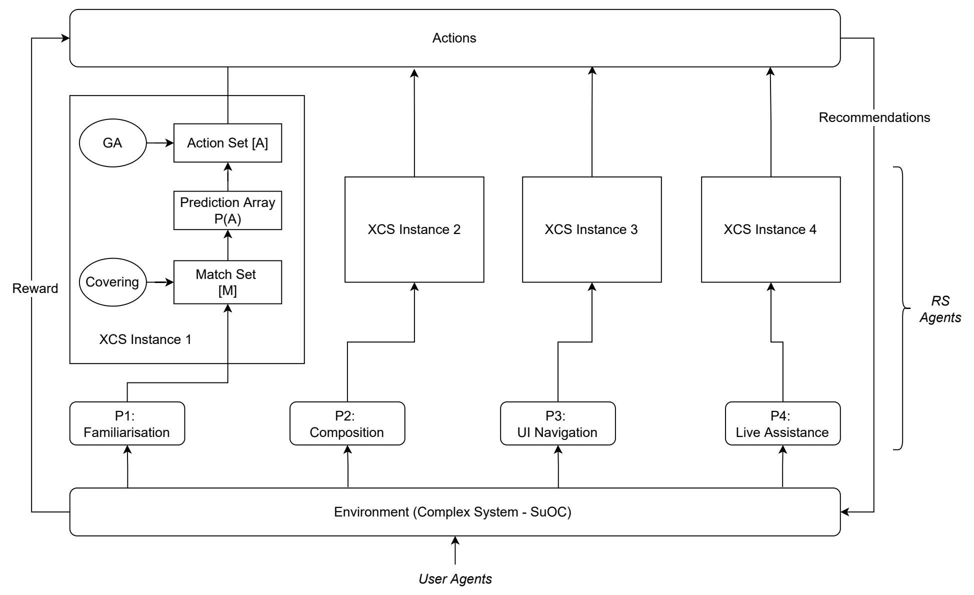

Figure 1: Architecture of the proposed multi-instance XCS system

Agent Representation:
$f_{Ag(RS)}$: *Sit* x *Dat* = *Act*
Sit: {usecase, intended-task}
Dat: {user-profile, user-goal, session-information}
Act: {recommendations}

Classifier Representation:
[condition]:[action]
condition: usecase, current-activity, user-profile, session-information
action: recommendations

Figure 2: Agent/Classifier representation

classifiers to address all these use cases might lead to *over-generalization*. Generalization is a process through which an XCS tries to make classifiers as general as possible through a genetic algorithm (GA) to attain optimal performance (Wagner and Stein, 2022). In each learning iteration, the GA selects a few rules with the highest fitness from the population and mutates them into general rules. A general classifier typically has a few attributes and can match with different situations (Booker et al., 1989). Whereas over-generalization is when the rule population will have the classifiers with more wild characters (#) and fewer attributes, resulting in giving inaccurate actions (Lanzi, 1999). In the case of an XCS-based RS, over-generalization can drastically reduce the performance and accuracy of the RS agent. To solve this, we propose using multiple RS agents, each meant for specific use cases, and each has its own XCS instances with individual rule populations. The proposed architecture of the XCS system is shown in Fig. 1. The $User$ interacts with the *System under Observation and Control (SuOC)*, in this case, the complex software system. They will be monitored by the RS throughout the session. The user activity will be captured in the form of a classifier by corresponding XCS instances. Depending on the Sit of the $User$ agent, and the Dat available, the RS will compare this classifier with the corresponding rule populations [P1-P4]. The XCS instance can select appropriate actions considering the highest prediction value and be given back to the SuOC as recommendations. However, there will be potential challenges in having multiple XCS instances. Such as resolving conflicts by different RS agents, representing rules across different RS agents, coordinating between RS agents for a seamless user experience, etc. These have to be researched over the course of development. As in any XCS system, the user's response to the recommendations is crucial. If the user accepts the recommendation, a positive reward *r* is given back to the XCS to strengthen the rule. If the user is not benefited from the recommendation, the reward will be negative. However, in this case, assuming the user's expertise in the software increases over time, they might ignore the recommendations. Solely modifying the rule's fitness without considering the user's mastery will impact the predictions for users who actually need them. Mapping user expertise and the conditions which deviated a lot could be one way to address this problem.

XCS as a Substitution for Established RS Approaches: Traditional recommendation approaches, such as collaborative filtering (CF) or content-based filtering (CB), use specific data points to formulate recommendations. For example, CF uses similarities between users and their usage/ratings on items to be recommended to generate suggestions (Koren et al., 2022). In comparison, CB uses the metadata of the item along with the user's interests to yield recommendations (Pazzani and Billsus, 2007). An XCS-based RS, as mentioned above, can be used as a substitution for these conventional approaches. For example:

$$[domain, rating, theme, goal, objects] : [template]$$

consider the simple classifier mentioned above with multiple attributes that suggest a 3D environment template to a user. The attributes referring to the user's domain and the attributes referring to ratings given by other users on an item can be examined to generate recommendations similar to CF. Similarly, attributes indicating the theme, the goal of using the environment, and the kind of object the environment contains can be considered to produce results similar to CB. An actual classifier can be further detailed and will contain more contextual attributes. In this way, using an XCS-based RS delivers the functionality of a hybrid RS.

Prototype Development

A virtual reality (VR) authoring platform (von Mammen et al., 2019) is chosen to test the XCS-based RS. Out of the multiple interlinked components an authoring platform typically have, we have considered the 3D environment editor (referred to as editor from now on) as the initial use case to implement the RS. The editor allows users to import 3D assets from online libraries and compose a VR environment by drag and drop facility. User assistance with an editor like this can be broadly classified into four categories. 1) Personalisation: allowing the users to resume work from where they stopped, provide usage statistics, and keep track of user preferences, 2) Platform Navigation: providing context-aware support in the form of cues, 3) Composition assistance: providing assets such as 3D models, automate/auto-complete the design, recommend templates, and provide cue assistance, and 4) Context-aware live support: alerts based on user activities. Platform personalization and recommendations for UI navigation are generated by processing a particular user's preferences and experience on the system. The details of the current session, such as the active/passive time, several IDs, etc., are also considered to narrow down the rules to construct accurate recommendations. The session data, the metadata and the theme of the 3D environment under composition, and the index of the assets used, will be considered to provide context-aware composition assistance, such as auto-completion of the environment and 3D model suggestions. In addition, the user's interaction pattern aligned with the session information and explicitly mentioned requests/requirements through the help window will be used to provide live assistance. Out of the four categories of assistance listed above, composition assistance is in the prototype stage, and the rest are under development. A graph database is adapted for keeping track of information considering the dynamicity of the data generated over time. As a result, the performance of the RS will remain constant even if the size of the graph multiplies over time. In addition, the graph schema is flexible enough to accommodate random changes that might arise as the development progresses and the understanding of the problem space grows (Robinson et al., 2015). In the context of an RS to assist a user, this is essential as we cannot foresee what kind of interaction a user can have with the system. As a part of the platform development, we conducted a contextual inquiry among 102 healthcare professionals who could be the platform's potential users. The study was conducted online, and the section of the study concerning user assistance asked the users about their expectations about RS. In addition, a mockup video of the authoring platform was shown to the users to understand all use cases of the RS better. The response from the users aligned with our proposed application ideas. A few of the notable requirements from the study were 1) the subsequent steps have to be indicated clearly while using the authoring platform, 2) to get pre-designed 3D environments upon providing a requirement as a keyword, 3) the system should indicate when there is a possibility of cybersickness while using the VR application (for further results: (Halbig et al., 2022)) All of these explicit requests are achievable using the proposed rule-based system. Consider the example rule (abstract representation):

$$[domain, usecase, profile, activity, status] : [R]$$

A rule consisting of the above attributes, the use case, user's experience, ongoing activity, and the status of that activity are a set of attributes that can be considered to suggest the subsequent steps, existing templates, and to generate alerts along with other recommendations the situation demands. During and after completing the development, user studies and interviews will be conducted to measure the performance of RS, the impact on user experience, and the effectiveness of user guidance it provides while using the platform.

Discussion and Future Work

During the prototype development, the XCS-based RS gave recommendations to the user with feature suggestions. For example, the user was asked to provide names of 3D models they would require to create a virtual environment, and the RS would suggest additional assets that align with the user's requirement. However, this observation is based on the ongoing development in a limited space of a 3D editor, with only a few classifiers in the rule population. A few questions that need to be explored through the next course of development are how to address the cold-start issue that might arise once the RS is implemented across multiple use cases in addition to a 3D editor. and how to maintain and improve the system's performance when the scope of the software platform and user base increases. In addition, currently, the system's front end acts as the SuOC, and it will remain the primary method to infer requirements from the user. However, the scope of using an Intelligent User Interface, similar to the famous Microsoft Clippy (Tymchuk, 2015), as an additional input mechanism to capture user input to increase the UX efficiency has to be explored.

Acknowledgements

This research was partially funded by the German Federal Ministry of Education and Research (BMBF) as part of the research initiative for Human-Technology Interaction "Digital Platforms: Interactive Assistance Systems for Humans" in the project VIA-VR (project number 16SV8444).

References

Abel, F., Bittencourt, I. I., Henze, N., Krause, D., and Vassileva, J. (2008). A rule-based recommender system for online discussion forums. In *International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 12–21. Springer.

Afsharchi, M., Far, B. H., and Denzinger, J. (2006). Ontology-guided learning to improve communication between groups of agents. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 923–930.

Aggarwal, C. C. (2016). Content-based recommender systems. In *Recommender systems*, pages 139–166. Springer.

Badjio, E. F. and Poulet, F. (2005). User guidance: from theory to practice, the case of visual data mining. In *17th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'05)*, pages 2–pp. IEEE.

Booker, L. B., Goldberg, D. E., and Holland, J. H. (1989). Classifier systems and genetic algorithms. *Artificial intelligence*, 40(1-3):235–282.

Denzinger, J. and Kidney, J. (2006). Evaluating different genetic operators in the testing for unwanted emergent behavior using evolutionary learning of behavior. In *2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology*, pages 23–29.

Halbig, A., Babu, S. K., Gatter, S., Latoschik, M. E., Brukamp, K., and von Mammen, S. (2022). Opportunities and challenges of virtual reality in healthcare – a domain experts inquiry. *Frontiers in Virtual Reality*, 3.

Kao, D. (2020). Exploring help facilities in game-making software. In *International Conference on the Foundations of Digital Games*, pages 1–14.

Koren, Y., Rendle, S., and Bell, R. (2022). Advances in collaborative filtering. *Recommender systems handbook*, pages 91–142.

Lanzi, P. L. (1999). An analysis of generalization in the xcs classifier system. *Evolutionary computation*, 7(2):125–149.

Mirel, B. (1998). "applied constructivism" for user documentation: Alternatives to conventional task orientation. *Journal of business and technical communication*, 12(1):7–49.

Oshita, M., Kaida, K., and Matsumoto, S. (2019). Improving user experience of c programming language learning system for novices. In *2019 8th International Congress on Advanced Applied Informatics (IIAI-AAI)*, pages 306–309. IEEE.

Padmanabhan, V., Seemala, S. K., and Bhukya, W. N. (2011). A rule based approach to group recommender systems. In *International Workshop on Multi-disciplinary Trends in Artificial Intelligence*, pages 26–37. Springer.

Pazzani, M. J. and Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer.

Ricci, F., Rokach, L., Shapira, B., and Kantor, P. B. (2010). Recommender systems handbook.

Robinson, I., Webber, J., and Eifrem, E. (2015). *Graph databases: new opportunities for connected data*. " O'Reilly Media, Inc.".

Selimi, D. and Nuci, K. P. (2020). The use of recommender systems in web technology and an in-depth analysis of cold state problem. *arXiv preprint arXiv:2009.04780*.

Sharma, S. and Kaur, D. (2015). Location based context aware recommender system through user defined rules. In *International Conference on Computing, Communication Automation*, pages 257–261.

Stein, A., Maier, R., Rosenbauer, L., and Hähner, J. (2020). Xcs classifier system with experience replay. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 404–413.

Sutcliffe, A. (2000). On the effective use and reuse of hci knowledge. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(2):197–221.

Tymchuk, Y. (2015). What if clippy would criticize your code? In *BENEVOL'15: Proceedings of the 14th edition of the Belgian-Netherlands software evoLution seminar*.

Urbanowicz, R. J. and Browne, W. N. (2017). *Introduction to learning classifier systems*. Springer.

Van der Meij, H. and Van Der Meij, J. (2014). A comparison of paper-based and video tutorials for software learning. *Computers & education*, 78:150–159.

von Mammen, S., Müller, A., Latoschik, M. E., Botsch, M., Brukamp, K., Schröder, C., and Wacker, M. (2019). Via vr: A technology platform for virtual adventures for healthcare and well-being. pages 1–2.

Wagner, A. R. and Stein, A. (2022). Mechanisms to alleviate overgeneralization in xcs for continuous-valued input spaces. *SN Computer Science*, 3(2):1–23.

Wilson, S. W. (1999). State of xcs classifier system research. In *International workshop on learning classifier systems*, pages 63–81. Springer.

Woźniak, M., Grana, M., and Corchado, E. (2014). A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 16:3–17.