Evaluating an Assistant for Creating Bug Report Assignment Recommenders

John Anvik
University of Lethbridge
Lethbridge, Canada
john.anvik@uleth.ca

ABSTRACT

Software development projects receive many change requests each day and each report must be examined to decide how the request will be handled by the project. One decision that is frequently made is to which software developer to assign the change request. Efforts have been made toward semi-automating this decision, with most approaches using machine learning algorithms. However, using machine learning to create an assignment recommender is a complex process that must be tailored to each individual software development project. The Creation Assistant for Easy Assignment (CASEA) tool leverages a project member's knowledge for creating an assignment recommender. This paper presents the results of a user study using CASEA. The user study shows that users with limited project knowledge can quickly create accurate bug report assignment recommenders.

ACM Classification Keywords

H.5.m. Information Interfaces and Presentation: Miscellaneous; I.2.4. Programming Languages and Software: Expert system tools and techniques; I.2.7. Natural Language Processing: Text analysis

Author Keywords

bug report triage; assignment recommendation; machine learning; recommender creation; computer supported work

INTRODUCTION

Large software development projects can receive hundreds of bug reports per day [5, 6]. Each of these bug reports needs to be analyzed and decisions made about how the report will be handled by the project. In cases where a change to the source code is needed, a decision is made about to whom the work will be assigned. This decision process is called *bug triage* and must be done for all incoming reports.

Bug triage takes significant time and resources [12]. Bug report assignment recommenders have been proposed as a method for reducing this overhead. Many researchers have investigated different approaches for assignment recommender creation, with most focusing on the use of machine learning [5, 7, 14, 29, 31].

Conceptually, the creation of an assignment recommender using machine learning is straightforward [3]. However in practice creating an assignment recommender for a specific software development project is challenging. The Creation Assistant for Easy Assignment (CASEA) tool [1] was created to assist software development projects in creating machine learning assignment recommenders tailored to a specific project.

This paper presents the results of a user study using CASEA to create assignment recommenders for a large open source project, and is the first such study. The study found that subjects could quickly create an accurate assignment recommender using the tool, despite the users having no specific knowledge about the software project.

To our best knowledge, CASEA is the first system to address the bug report assignment recommender creation problem, and this paper presents the first study of its use.

This paper proceeds as follows. First, an overview of CASEA is presented. Next, the results from a user study involving subjects creating assignment recommenders for the Eclipse Platform project are presented. The paper then concludes with a discussion of some of the threats to the validity of this work, related work and possible future improvements to make CASEA more practical for software development projects.

BACKGROUND

This section presents background information about bug reports, their life cycles, and machine learning.

Bug Reports

Bug reports, also known as change requests, provide a means for users to communicate software faults or feature requests to software developers. They also provide a means for developers to manage software development tasks. Bug reports contain a variety of information, some of which is categorical and some of which is descriptive. Categorical information includes such items as the report's identification number (i.e. bug id), its resolution status (e.g., NEW or RESOLVED), the component the report is believed to involve, and which developer has been assigned the work. Descriptive information includes the title of the report, the description of the report, and discussions about possible approaches to resolving the report. Finally, a report may contain other information, such as attachments or links to other reports.

Bug report lifecycles

All bug reports have a lifecycle. When a bug report first enters a project's issue tracking system (ITS), it is in a state such as UNCONFIRMED or NEW. The bug report will then move

EICS'16, June 21-24, 2016, Bruxelles, Belgium.

through different states, depending on the project's development process, and arrive at a resolution state, such as FIXED or INVALID. The lifecycle of a bug report can be used to categorize bug reports [5]. Figure 1 shows an example life cycle state graphs from the Bugzilla ITS [20].

Machine learning algorithms

Machine learning is the development of algorithms and techniques that allow computers to learn [18]. Machine learning algorithms fall under three categories: supervised learning, unsupervised learning, and reinforcement learning. Bug report assignment recommenders primarily use supervised learning algorithms, such as Support Vector Machines (SVM)[15], Naive Bayes [24] and ML-KNN [30]. Understanding how a machine learning algorithm creates a recommender requires understanding three concepts: the feature, the instance and the class. A *feature* is a specific piece of information that is used to determine the class, such as a term that appears in one or more of a set of bug reports. An *instance* is a collection of features that have specific values, such as all of the terms in the description of a specific bug report. Finally, a *class* is the collection of instances that all belong to the same category, such as all of the bug reports fixed by a developer. In supervised machine learning, training instances are labeled with their class. A recommender is created from a set of instances and the output of the recommender is a subset of the classes predicted for a new instance.

CREATION ASSISTANT FOR EASY ASSIGNMENT

The Creation Assistant for Easy Assignment (CASEA) [1] is a software tool to assist a software project in creating and maintaining bug report assignment recommenders. CASEA guides a project member through the assignment recommender creation process in four steps: Data Collection, Data Preparation, Recommender Training, and Recommender Evaluation. The remainder of this section presents an overview of how CASEA assists with each of these steps.

Data Collection

The first step in recommender creation is to gather the data to be used for creating the recommender. Specifically, bug reports are extracted from the project's issue tracking system (ITS). The project member provides the URL of the project's ITS, a date range for data collection, and an optional maximum limit for number of reports to gather. Reports that have a resolution status of RESOLVED, VERIFIED or CLOSED are gathered chronologically, with every tenth report selected as a testing report to create an unbiased set for evaluation.

Data Preparation

Having collected the data from the project's ITS, the next step is to filter the data to produce the highest quality training set. Two types of filtering are performed: automatic and assisted.

The automatic filtering performs three actions on the textual data. First, terms that are stopwords (i.e. common words such as 'a' and 'the') are removed. Next, stemming is performed to reduce all of the terms to their respective root values so that words such as 'user' and 'users' are treated as the same word, ensuring a common vocabulary between the reports. Finally, punctuation and numeric values are removed, except where the punctuation is important to the term, such as URLs or class names (e.g. "org.eclipse.jdt").

CASEA assists the user with two types of filtering: label filtering and instance filtering. To assist with label filtering, CASEA presents the user with a label frequency graph. For an assignment recommender, this graph presents bug fixing statistics, a type of activity profile [21], for the project developers based on a random sample of all of the bug reports in the training data set. Figure 2 shows the Configuration tab and an example of label filtering. As can be seen in the graph, developer activity follows a Pareto distribution curve with a few developers contributing the bulk of the work, and many other developers making small contributions [11, 16, 19, 25]. CASEA visualizes the project's development activity and allows the user to select a threshold using a slider, such that only a core set of developers are recommended. In Figure 2, a cutoff of 21 has been selected.

Instance filtering is done using project-specific heuristics. The heuristics have two parts: a grouping rule and a label source. The grouping rule is used to categorize the data into groups for which the label source will be used for labeling the instances. For an assignment recommender, the grouping rule is a bug report lifecycle (called "Path Group" in Figure 2) and the label source (i.e. data source in Figure 2) is either a field from the bug report, such as the *assigned-to* field, or other labelling information that can be extracted from the bug report, such as the user that last attached a patch or the developer who marked the report as resolved.

Figure 2 shows an example of instance filtering in CASEA. All of the training reports are used to determine the specific bug report life cycles for the project and the user is presented with a statistical summary of the categories. The figure shows that for the data set for the Eclipse Product project, 30.6% of the reports have a NEW $\rightarrow$ FIXED (NF) lifecycle, followed by 22.6% NEW $\rightarrow$ FIXED $\rightarrow$ VERIFIED (NFV), and 15.6% being NEW (N). Using this information, the user can create heuristics for the most common occurring categories. In this case, "FixedBy" was chosen for the NF category, "Resolver" for the NFV category, and "Reporter" for the N category. The user can also choose the number of heuristics to be applied, to a maximum of ten; six was chosen in Figure 2.

Recommender Training

After filtering the data to create the set of training and evaluation instances for the recommender, the data is then formatted for use with the machine learning algorithm. Once the user has filtered the data and the data is formatted, the recommender is created using a multi-class Support Vector Machines (SVM) algorithm with a Gaussian kernel. SVM is a commonly used algorithm for assignment recommendation [5, 7].

Recommender Evaluation

Once the user starts the recommender creation process, the user is moved to the Analysis tab that presents the recommender evaluation results (Figure 3). The user can then return to the Configuration tab, adjust the values for label and instance filtering, and create a new recommender. This process

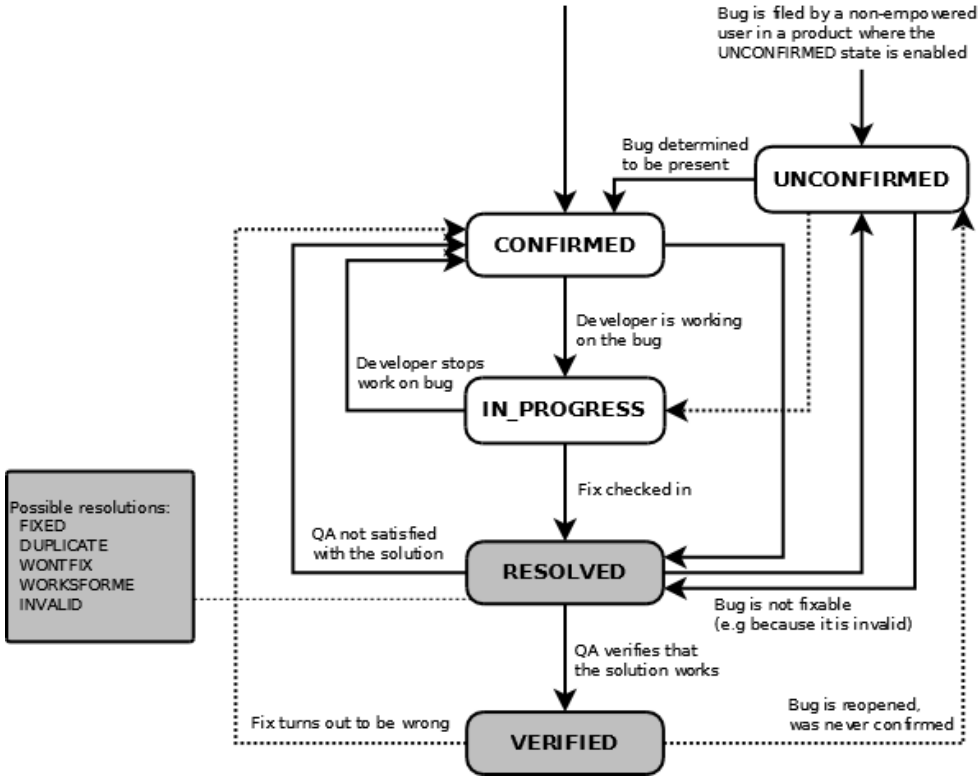

Figure 1. Bug report life cycle state diagram from the Bugzilla ITS.

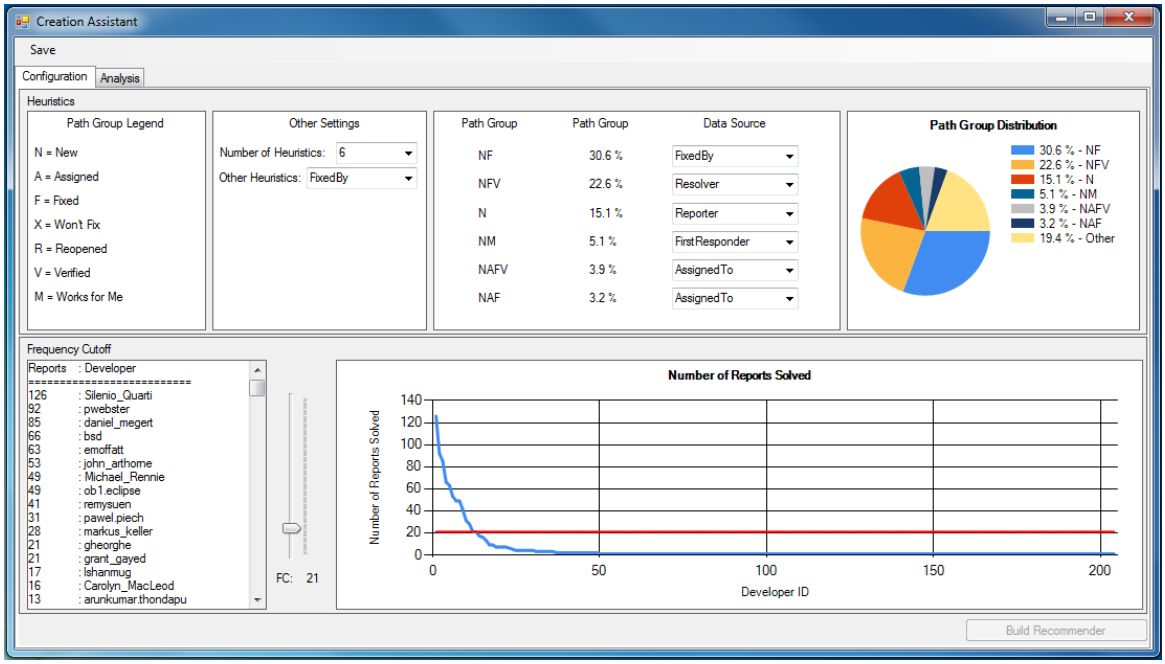

Figure 2. The Creation Assistant for Easy Assignment (Configuration tab).

continues until the user is either satisfied with the created recommender, or the user has determined that an assignment recommender cannot be created with a high enough accuracy to benefit the project. At any time the user can save the recommender configuration and return at a later date.

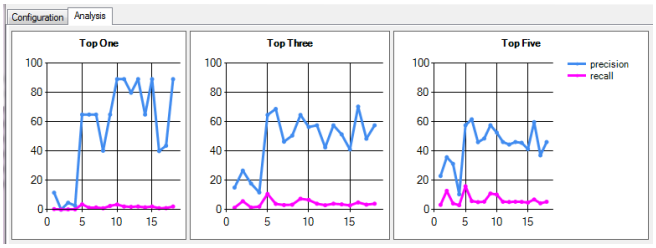

Figure 3. Recommender evaluation in CASEA (Analysis Tab).

CASEA uses the metrics of precision and recall to evaluate a created recommender. It presents results for the top recommendation (top-1), the top 3 recommendations (top-3), and the top 5 recommendations (top-5). Figure 3 shows an example of the evaluation results for a recommender after eighteen trials. It shows that the first four configurations did not create very accurate recommenders as the activity threshold was too low, but when the threshold was raised, a more accurate recommender was produced. After about five more trials, a good heuristic configuration was determined that produced an assignment recommender with a high top-1 accuracy and reasonable top-3 and top-5 accuracies. Further experimentation was done with the heuristics with varying results, before determining that the configuration from trial #10 was the best configuration.

USER STUDY OF CASEA

A small user study was conducted to assess the potential for CASEA to assist software projects in creating assignment recommenders. Specifically, the study sought to answer qualitative questions such as "What aspects of the recommender creation process and the CASEA interface do users find helpful? challenging? or confusing?". This study was similar in intent to Stumpf et al [28], who conducted a user study to determine how users interact with a machine learning system.

The study was conducted using a sample of eight computer science graduate and undergraduate students. This study population was selected under the assumption that using a group with no specific project knowledge would provide a lower-bound for future in-field user studies.

User Study Setup

The user study was conducted in the following manner. First, subjects were asked to complete a prior knowledge and experience survey. Specifically, subjects were asked about their prior knowledge and experience in two areas: technical experience and technical knowledge. For technical experience, subjects were asked about their level of experience with issue tracking systems, open source projects and software testing. To assess prior technical knowledge, subjects were asked about their familiarity with bug reports, machine learning algorithms, classifiers or recommender systems, user interface design principles, and data mining. A Likert scale (Very Experienced/Familar, Some Experience/Familarity, Little Experience/Familarity, Heard Of, No Experience/Familarity) was used for the self-reporting of their experience level.

After completing the prior experience and knowledge survey, subjects were asked to create an assignment recommender for the Eclipse Platform project within fifteen minutes.

Once the subjects expressed that they were done using CASEA, a debriefing interview was conducted. The subjects were asked to explain their approach to creating an assignment recommender using CASEA and what they recommended as improvements to the tool, as well as any other comments about their experience with CASEA.

Prior Knowledge and Experience

Figure 4 shows a summary of the responses from subjects regarding their prior experience. As shown, most of the subjects had prior experience with issue tracking systems, with four reporting some experience and four reporting little experience. Overall testing experience was a bit less, with six subjects reporting little experience, two reporting "heard of" and one reporting no experience. Subjects had the least overall experience with contributing to open source projects, with two reporting litte experience, three reporting "heard of" and four reporting no experience. [1]

For technical knowledge, Figure 5 shows a summary of the responses. Most of the subjects reported either being very familar (2 subjects) or having some familiarity (6 subjects) with user interface design principles from either recently or currently taking an undergraduate course about this topic. Half of the subjects reported familiarity with machine learning algorithms, and slightly more (5 subjects) reported familiarity with classifiers and recommender systems. The knowledge in these areas came from either taking an undergraduate course in computational intelligence or from other course projects. Subjects reported the least familiarity with the bug report lifecycle and data mining.

Quantitative Results

Table 1 shows the quantitative results from the eight subjects. The first column identifies the subjects. The next two columns present both the number of trials a subject conducted before creating their most accurate assignment recommender, and the total number of trials that the subject conducted in creating an assignment recommender using CASEA. The next three columns show the Top-1, Top-3 and Top-5 precision and recall values for the best Eclipse Platform assignment recommender created by the subject. The last three rows of Table 1 show a summary of the results, presenting the maximum, minimum, and median values for the columns.

Table 2 shows the threshold and heuristic configurations for the best assignment recommender created by each of the eight subjects. The first two columns list the top ten path groups for the data set and how much of the data set is covered by the path group. As shown by the table, 77% of the data set is covered by the first five path groups, and most of the subjects specified heuristics for six or fewer path groups. Also, with

[1] Not all subjects provided answers to all of the questions.

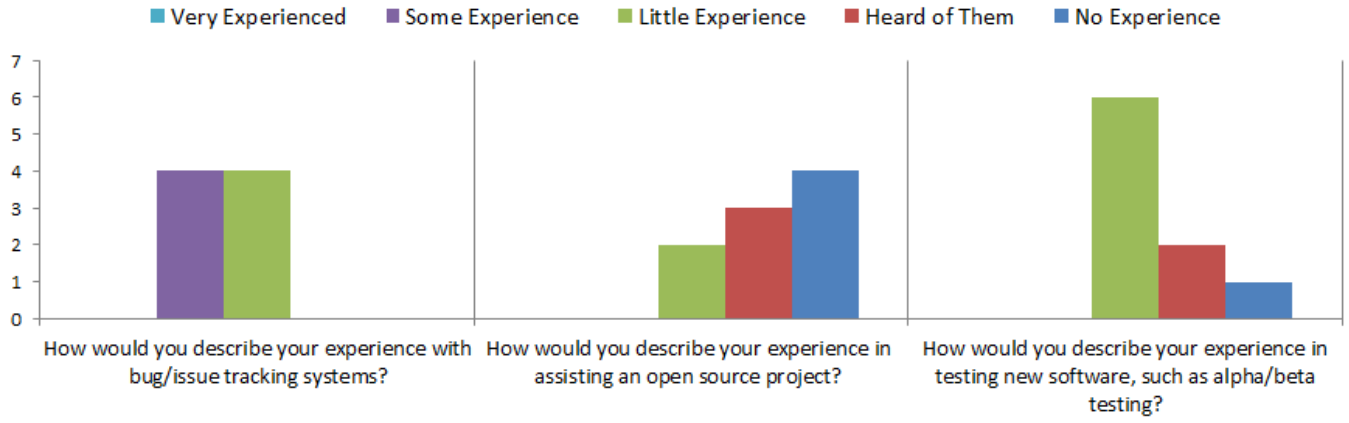

Figure 4. Responses by subjects regarding prior technical experience.

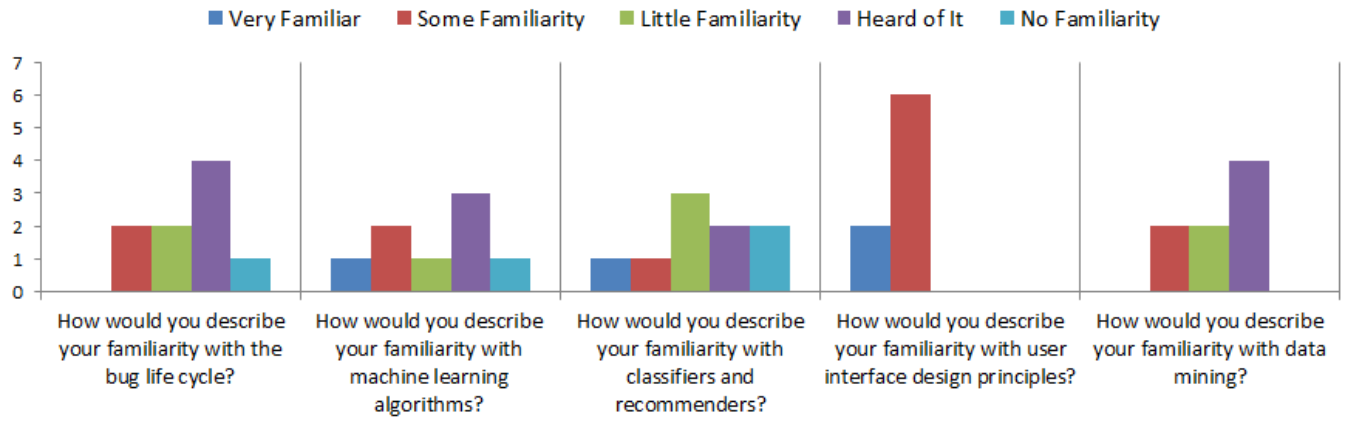

Figure 5. Responses by subjects regarding prior technical knowledge.

one exception, the "Assigned" data source was used for the remaining 11%-27% not covered by the specified heuristics. Half of the subjects chose values less than 10 for the threshold and the others used values greater than 20.

The results show that the subjects were usually able to create a reasonably accurate assignment recommender in 10 trials or less. The two most accurate recommenders (created by Subjects #7 and #8) had the same configuration (i.e. threshold and heuristic values), as shown in Table 2.

Qualitative Results and Observations

Based on observations during the study and responses from the debriefing interview, subjects were found to employ two strategies for assignment recommender creation using CASEA. Some subjects were found to be very experimental in their approach, making many changes before creating a new recommender. Other users were more methodical, making small changes and testing the results. Figure 6 shows a categorization of the different types of changes (heuristic change, threshold change or both) made by each subject. As expected, subjects changed the heuristic configurations the most, and most subjects only changed the threshold three times or fewer.

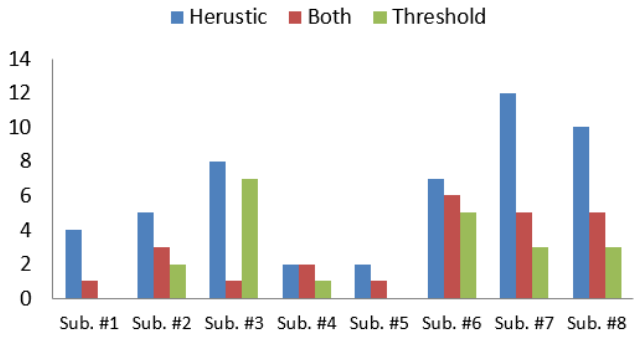

Figure 6. Types of changes made by subjects.

Identifier	Trials to Best	Max Trials	Top 1 (%)		Top 3 (%)		Top 5 (%)	
			Precision	Recall	Precision	Recall	Precision	Recall
Subject #1	5	5	68.63	1.25	54.98	3.01	66.27	6.04
Subject #2	10	10	39.95	1.14	41.26	3.52	45.1	6.42
Subject #3	5	16	68.63	2.08	46.49	4.22	35.25	5.33
Subject #4	3	5	37.99	2.32	27.53	5.04	22.99	7.02
Subject #5	2	3	81.62	1.49	62.66	3.43	51.62	4.7
Subject #6	16	18	68.87	4.07	34.97	6.2	34.66	10.24
Subject #7	11	20	89.22	3.48	56.45	6.6	52.7	10.27
Subject #8	10	19	89.22	3.48	56.45	6.6	52.7	10.27
Max	16	20	89.22	4.07	62.66	6.6	66.27	10.27
Min	2	3	37.99	1.14	27.53	3.01	22.99	4.7
Median	7.5	13	68.75	2.2	50.735	4.63	48.36	6.72

Table 1. Best Eclipse Platform assignment recommenders created by subjects.

	Covers	Sub. #1	Sub. #2	Sub. #3	Sub. #4	Sub. #5	Sub. #6	Sub. #7	Subject #8
NF	30.6%	FirstResp.	Resolver	Assigned	FixedBy	FirstResp.	FixedBy	FixedBy	FixedBy
NFV	22.6%	Resolver	Resolver	Assigned	Assigned	FixedBy	Assigned	Resolver	Resolver
N	15.1%	Reporter	Assigned	FixedBy	Assigned	FirstResp.	Assigned	Assigned	Assigned
NM	5.1%	FixedBy	FirstResp.	FirstResp.	Assigned	Assigned	Assigned	FirstResp.	FirstResp.
NAFV	3.9%	FixedBy	Resolver	FirstResp.		Assigned	Assigned	Assigned	Assigned
NAF	3.2%		Resolver	Reporter				Assigned	Assigned
NC	3.0%		FirstResp.	Reporter					
NX	2.5%		FirstResp.	Assigned					
NFRFV	1.8%		FixedBy						
NA	1.1%		Assigned						
Other		Assigned	Assigned	Assigned	Assigned	Assigned	FixedBy	Assigned	Assigned
Activity Threshold		47	31	3	5	5	5	22	22

Table 2. Best assignment recommender configurations of subjects.

One subject commented that the best strategy was to make small incremental changes, and that CASEA made it easy to employ this strategy. Another subject observed that creating an assignment recommender using CASEA was similar to trying to get a high score in a game, where the score was the precision and recall values.

As part of the recommender evaluation, CASEA provides information about how long it takes to create a recommender. This led some subjects to work towards an incorrect goal of minimizing the recommender creation time.

Although subjects were provided with a brief tutorial of CASEA and a high level explanation of the recommender creation process at the beginning of the study session, subjects encountered a number of problems related to understanding terminology or concepts. Specifically, the term “Path Group" was used to describe the categorization of bug reports, and subjects found this term unintuitive. This led to some initial confusion about the options in the heuristic configuration panel. Also, the meaning of the precision and recall metrics was not initially well understood by subjects. However, once their meaning was understood, subjects felt that they made more intelligent choices about the configuration.

As was mentioned, the subjects did not have specific knowledge about the project, such as who formed the core group of developers. This led to some subjects choosing a low activity cutoff so as to not exclude developers, and resulted in recommenders that were not accurate and took longer to create. This behavior would not be expected from an actual project member using CASEA, as they would have knowledge about the core development team.

THREATS TO VALIDITY

This section highlights some of the threats to the internal validity, external validity, and construct validity of this work.

Threats to the internal validity of this work relate to the potential sources of error in the evaluation of CASEA. A potential source of error is with the creation of the data set used for the evaluation. Although a random sample of bug reports was examined to establish that the data collection procedure was correct, there may have been some bug reports that contained incorrect data.

Threats to external validity relate to the generalizability of the results to other projects or user groups. In this work, subjects with no project-specific knowledge were used to evaluate the usability of CASEA. Therefore, these results would not generalize to those with project-specific knowledge, but could be considered as a lower-bound for such a group.

Threats to construct validity refers to the suitability of the evaluation measures. The method used to determine the set of developers that could have fixed a bug report, used for calculating precision and recall, is known to overestimate the group [4]. This results in precision values that are overvalued, and recall values that are undervalued. However, the evaluation results in CASEA show the relative differences between different configurations, so even if the precision and recall values are over or under their true value, CASEA still provides meaningful information to the user.

RELATED WORK

This section presents related work in the areas of assisting with triage, assisting with recommender creation, and explaining machine learning.

Assisting with Bug Report Triage

Like CASEA, Porchlight [9] and it's predecessor TeamBugs [10] seek to provide a tool to assist project triagers in making their tasks more efficient. Porchlight allows a triager to group similar bug reports together using tags, and then apply a triage decision to the group. This tagging is similar to the path groups in CASEA, which also groups bug reports into categories for specifying and applying labelling heuristics.

Assisting with Recommender Creation

SkyTree Infinity [26] and BigML [8] both provide means for guiding a user through the creation of machine-learning recommenders. However, using Skytree Infinity still requires advanced knowledge of machine learning and statistics [13]. BigML provides no support for data preparation or visualization, and creates recommenders using decision-trees, which was shown to be ineffective for the bug report assignment problem [5].

Explaining Machine Learning

One avenue toward making the use of recommender systems practical is to assist in their creation and evaluation. This is the approach taken by CASEA. An alternative approach to making their use practical is by explaining their results.

Poulin et al. [23] developed ExplainD, a framework for explaining decisions made by classifiers that use additive evidence, such as Naive Bayes. The framework was used in a bioinformatics web-based system called Proteome Analyst.

Strumbelj and Konoenko [27] presented a method for explaining classifier predictions that used coalitional game theory. The method used a sampling-based approach to reduce the computational complexity of explaining the contributions of individual feature values. Their approach was applied to explaining the results of various machine learning algorithms, including Naive Bayes and SVM.

Kulesza et al. [17] created an end user debugging approach for intelligent assistants, such as bug report assignment recommenders. The system allowed the user to ask 'why' questions about predictions and then change the answers to debug current and future predictions.

Basilio Noris developed a visualization tool for machine learning called MLDemos [22]. MLDemos assists in understanding how different machine learning algorithms function. It also demonstrates how the parameters of the algorithms affect and modify the results in classification problems.

CONCLUSION

This paper presented the results of a pilot study of CASEA, a tool to assist in the creation of bug report assignment recommenders. CASEA assists a user in labelling and filtering the bug reports used for creating a project-specific assignment recommender, as well as providing feedback on the effectiveness of the configured assignment recommender. The study found that users with little to no project-specific knowledge were able to quickly create effective assignment recommenders for the Eclipse Platform project.

Based on feedback and the results of the user study, a number of future improvements were idenitified for CASEA, including having CASEA first attempt to tune a recommender automatically and then have the user tweak the configuration, extending CASEA to assist with the creation of other triage recommenders, supporting other machine learning algorithms, and providing other evaluation metrics, such as F1. An improved version of CASEA, called the Creation Assistant Supporting Triage Recommenders (CASTR) [2] was created to incorporate these changes in preparation for a field study with project developers.

REFERENCES

1. J. Anvik, M. Brooks, H. Burton, and J. Canada. 2014. Assisting Software Projects with Bug Report Assignment Recommender Creation. In *Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*. 470–473.
2. J. Anvik, M. Brooks, H. Burton, J. Canada, and T. Henders. 2016. CASTR - Creation Assistant Supporting Triage Recommenders. (July 25, 2016). https://bitbucket.org/bugtriage/castr
3. J. Anvik, L. Hiew, and G. C. Murphy. 2006. Who should fix this bug?. In *Proceedings of the 28th International Conference on Software Engineering*. 361–370.
4. J. Anvik and G. C. Murphy. 2007. Determining Implementation Expertise from Bug Reports. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*. 2–9.
5. J. Anvik and G. C. Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology* 20, 3 (Aug. 2011), 10:1–10:35.
6. S. Banitaan and M. Alenezi. 2013. TRAM: An approach for assigning bug reports using their Metadata. In *2013 Third International Conference on Communications and Information Technology*. 215–219.
7. P. Bhattacharya, I. Neamtiu, and C. R. Shelton. 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. *Journal of Systems and Software* 85, 10 (Oct. 2012), 2275–2292.

8. BigML. 2016. BigML is Machine Learning for everyone. (Mar 10, 2016). https://bigml.com
9. G. Bortis, Hoek, and van der A. 2013. PorchLight: A Tag-based Approach to Bug Triaging. In *Proceedings of the 2013 International Conference on Software Engineering*. 342–351.
10. G. Bortis and A. van der Hoek. 2011. Teambugs: A Collaborative Bug Tracking Tool. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*. 69–71.
11. G. Canfora and L. Cerulo. 2006. Supporting Change Request Assignment in Open Source Development. In *Proceedings of the 2006 ACM Symposium on Applied Computing (SAC '06)*. 1767–1772.
12. Y. C. Cavalcanti, P. A. da Mota Silveira Neto, I. Machado, T. F. Vale, E. S. de Almeida, and S. R. de Lemos Meira. 2014. Challenges and opportunities for software change request repositories: a systematic mapping study. *Journal of Software: Evolution and Process* 26, 7 (2014), 620–653.
13. S. Charrington. 2012. Three New Tools Bring Machine Learning Insights to the Masses. (Feb 27, 2012). http://readwrite.com/2012/02/27/three-new-tools-bring-machine
14. D. Cubranic and G. C. Murphy. 2004. Automatic Bug Triage Using Text Categorization. In *Proceedings of the 16th International Conference of Software Engineering and Knowledge Engineering*. 92–97.
15. T. Joachims. 1998. Text Categorization with Suport Vector Machines: Learning with Many Relevant Features. In *Proceedings of the 10th European Conference on Machine Learning*. 137–142.
16. S. Koch and G. Schneider. 2002. Effort, Cooperation and Coordination in an Open Source Software Project: GNOME. *Information Systems Journal* 12, 1 (2002), 27–42.
17. T. Kulesza, S. Stumpf, W. Wong, M. M. Burnett, S. Perona, A. Ko, and I. Oberst. 2011. Why-oriented End-user Debugging of Naive Bayes Text Classification. *Transactions on Interactive Intelligent Systems* 1, 1 (Oct. 2011), 2:1–2:31.
18. Tom Mitchell. 1997. *Machine Learning*. McGraw Hill.
19. A. Mockus, R. T. Fielding, and J. D. Herbsleb. 2002. Two Case Studies of Open Source Software Development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology* 11, 3 (July 2002), 309–346.
20. Mozilla Corporation. 2014. Bugzilla. (Nov 24, 2014). http://www.bugzilla.org/
21. H. Naguib, N. Narayan, B. Brügge, and D. Helal. 2013. Bug Report Assignee Recommendation Using Activity Profiles. In *Proceedings of the 10th Working Conference on Mining Software Repositories (MSR '13)*. 22–30.
22. B. Noris. 2016. ML Demos. (Mar 10, 2016). http://mldemos.epfl.ch/
23. B. Poulin, R. Eisner, D. Szafron, P. Lu, R. Greiner, D. S. Wishart, A. Fyshe, B. Pearcy, C. MacDonell, and J. Anvik. 2006. Visual Explanation of Evidence in Additive Classifiers. In *Proceedings of the 18th Conference on Innovative Applications of Artificial Intelligence - Volume 2 (IAAI'06)*. 1822–1829.
24. Jason D. M. Rennie, Lawrence Shih, Jaime Teevan, and David R. Karger. 2003. Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In *In Proceedings of the Twentieth International Conference on Machine Learning*. 616–623.
25. G. Robles, S. Koch, J. M. González-Barahona, and J. Carlos. 2004. Remote Analysis and Measurement of Libre Software Systems By Means of the CVSAnalY tool. In *In Proceedings of the 2nd ICSE Workshop on Remote Analysis and Measurement of Software Systems (RAMSS)*. 51–55.
26. Skytree Inc. 2016. Skytree Server. (Mar 10, 2016). http://www.skytree.net
27. E. Strumbelj and I. Kononenko. 2010. An Efficient Explanation of Individual Classifications Using Game Theory. *The Journal of Machine Learning Research* 11 (March 2010), 1–18.
28. S. Stumpf, V. Rajaram, L. Li, W. Wong, M. Burnett, T. Dietterich, Erin S., and J. Herlocker. 2009. Interacting Meaningfully with Machine Learning Systems: Three Experiments. *International Journal of Human-Computer Studies* 67, 8 (Aug. 2009), 639–662.
29. X. Xia, D. Lo, X. Wang, and B. Zhou. 2013. Accurate developer recommendation for bug resolution. In *In Proceedings of the 20th Working Conference on Reverse Engineering*. 72–81.
30. Min-Ling Zhang and Zhi-Hua Zhou. 2007. ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition* 40, 7 (2007), 2038 – 2048.
31. T. Zhang and B. Lee. 2013. A Hybrid Bug Triage Algorithm for Developer Recommendation. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. 1088–1094.