CASTR: A WEB-BASED TOOL FOR CREATING BUG REPORT ASSIGNMENT RECOMMENDERS

DISHA DEVAIYA
Bachelor of Science, Saurashtra University, 2006
Master of Computer Application, Veer Narmad South Gujarat University, 2009

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

CASTR: A WEB-BASED TOOL FOR CREATING BUG REPORT ASSIGNMENT RECOMMENDERS

DISHA DEVAIYA

Date of Defence: June 21, 2019

Dr. John Anvik Thesis Supervisor	Assistant Professor	Ph.D.
Dr. Yllias Chali Thesis Examination Committee Member	Professor	Ph.D.
Dr. Wendy Osborn Thesis Examination Committee Member	Associate Professor	Ph.D.
Dr. Howard Cheng Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

In memory of my loving father, my mother for her unconditional love and my parents-in-law for helping me become my best self.

To Aashka, a little girl who is eagerly waiting to be a part of my graduation ceremony.

Abstract

Large software development projects receive a large number of bug reports every day. Bug triage is a process where issues are screened and prioritized. Bug triage takes significant time and resources. For reducing the workload of project members, researchers have proposed using assignment recommenders.

As the creation of bug report assignment recommenders is complex, we propose a web-based tool called the *Creation Assistant for Supporting Triage Recommenders* (CASTR) to assist the project members with the creation of assignment recommenders. CASTR assists a user in labeling and filtering the bug reports used for creating a project-specific assignment recommender.

As the field study results present, recommenders can be created with good accuracy using CASTR such as 50-95% for Top-1 recommendations, 20-80% for Top-3 recommendations and 10-70% for Top-5 recommendations. Most participants (60%) found CASTR easy to use and were very likely to recommend CASTR for creating an assignment recommender.

Acknowledgments

I would like to express my deepest gratitude to my supervisor Dr. John Anvik, who made this work possible through constant support and encouragement. His friendly guidance and expert advice have been invaluable throughout all stages of the work. I cannot thank him enough for his patience and contribution towards correcting my writing.

I would like to thank my M.Sc. supervisory committee members Dr. Yllias Chali, and Dr. Wendy Osborn for their time and effort spent on thesis committee meetings. They provided very valuable feedback and helpful suggestions to my research.

I must thank Natural Sciences and Engineering Research Council (NSERC) of Canada Discovery Grant for providing me funding for the research work. I am also thankful to Nexix Inc for allowing me to use office resources for my thesis writing. I would also like to thank the experts who were involved in the field study for this research project. Without their participation and input, the field study could not have been successfully conducted.

Special thanks are due to my husband Bhavin, for his continuous support and understanding. My thanks are extended to my family members Mital, Vishal, and Mitsu for their constant source of inspiration.

Contents

Contents vi

List of Tables viii

List of Figures x

1 Introduction 1

1.1 Outline . . . 4

2 Background 5

2.1 Recommendation Systems in Software Engineering . . . 5

2.2 Bug Reports . . . 6

2.3 Bug Report Triaging Workflow . . . 7

2.4 Machine Learning Algorithms . . . 9

2.4.1 Support Vector Machines . . . 10

2.4.2 Naive Bayes . . . 11

2.4.3 C4.5 Decision Tree . . . 13

2.4.4 Rules . . . 15

2.4.5 Expectation-maximization . . . 16

2.5 RESTful APIs . . . 17

3 Related Work 18

3.1 Assisting With Bug Report Triage . . . 18

3.1.1 CASEA: Creation Assistant Supporting Triage Recommenders . . . 18

3.1.2 Porchlight / TeamBugs . . . 20

3.2 Assisted Machine Learning Recommender Creation . . . 20

4 An Approach to Assisting with Triage Recommender Creation 22

4.1 An Overview of Bug Report Assignment Recommender Creation . . . 22

4.2 CASTR . . . 23

4.2.1 Download Bug Reports . . . 24

4.2.2 Triage Recommender Configuration . . . 26

4.3 CASTR: Web Service . . . 33

4.3.1 An approach to handle data imbalances . . . 33

4.4 Summary . . . 37

5 Evaluation **38**
5.1 An Analytic Evaluation of the Recommenders . . . 39
5.1.1 Analytic Evaluation Procedure . . . 39
5.1.2 Assignment Recommender Evaluation . . . 41
5.2 A Field Study of CASTR . . . 44
5.2.1 Field Study Procedures . . . 45
5.2.2 Dataset Information . . . 47
5.2.3 Prior Knowledge and Experience . . . 48
5.2.4 Quantitative Results . . . 48
5.2.5 Discussion of Quantitative Results and Observations . . . 53
5.3 Qualitative Results . . . 54
5.4 Threats to Validity . . . 55
5.4.1 Internal Validity . . . 55
5.4.2 External Validity . . . 56
5.4.3 Construct Validity . . . 57
5.5 Summary . . . 57

6 Conclusion and Future Work **59**
6.1 Conclusion . . . 59
6.2 Future Work . . . 60

References **62**

A Assignment Recommender Evaluation Results **65**
A.1 LibreOffice . . . 65
A.2 Firefox . . . 67
A.3 Plasmashell . . . 69

B Field Study Outcome **71**
B.1 Initial Survey Results . . . 71
B.2 CASTR Usage Feedback . . . 73

C Quantitative Results **74**
C.1 Heuristics Configurations . . . 74
C.2 Precision and Recall . . . 79

D Ethics Certificate **80**

List of Tables

4.1 Projects with different date range and number of reports used for tuning recommenders. 25
4.2 Example of dataset distribution based on resolution group. 27
4.3 Examples of valid data sources. 28
4.4 Examples of project heuristics used for labeling reports. 29
4.5 Precision and Recall of a recommenders when using different machine learning algorithm. 30
4.6 Weka classes to train the classifier. 33
4.7 Example of instances created when sequential over-sampling is applied on the dataset. 36
4.8 Example of instances used for training the recommender. 37

5.1 Bug reports used in evaluation of assignment recommender created by hand. 40
5.2 Training and testing set sizes for evaluating recommenders. 41
5.3 Example of sample recommendations generated by CASTR. 42
5.4 Recommenders evaluation result from Confusion Matrix. 44
5.5 Questionnaire used to get background information of participants. 45
5.6 Questionnaire used for the post usage interview. 46
5.7 Developer wise bug report assignment from Plasmashell project. 47
5.8 Best assignment recommenders created by participants. 50
5.9 Project heuristics used for creation of best assignment recommenders. . . . 52
5.10 Heuristics used for labeling bug reports by most of the participant. 53

A.1 Evaluation results of assignment recommender created using LibreOffice dataset. 65
A.2 Evaluation results of assignment recommender created using Firefox dataset. 67
A.3 Evaluation results of assignment recommender created using Plasmashell dataset. 69

B.1 Participants demographic information and technical background details. . . 72
B.2 Participant's feedback on CASTR tool. 73

C.1 Heuristics configurations used in assignment recommender creation by User1. 75
C.2 Heuristics configurations used in assignment recommender creation by User2. 75
C.3 Heuristics configurations used in assignment recommender creation by User3. 75
C.4 Heuristics configurations used in assignment recommender creation by User4. 76
C.5 Heuristics configurations used in assignment recommender creation by User5. 76
C.6 Heuristics configurations used in assignment recommender creation by User6. 76
C.7 Heuristics configurations used in assignment recommender creation by User7. 77

C.8 Heuristics configurations used in assignment recommender creation by User8. 77
C.9 Heuristics configurations used in assignment recommender creation by User9. 77
C.10 Heuristics configurations used in assignment recommender creation by User10. 78
C.11 Repetitive Heuristics configurations used in assignment recommender creation. 78

List of Figures

1.1 Recommender creation process. 2

2.1 Default lifecycle of a Bugzilla bug report. 7
2.2 Bug report triage workflow of KDE open source project. 8
2.3 Hyperplane in 2-dimensional space. 10
2.4 Naive Bayes classification 12
2.5 Example of a Decision Tree. 14
2.6 CASTR application structure. 17

4.1 CASTR login screen. 23
4.2 CASTR dashboard screen. 24
4.3 CASTR downloading bug reports screen. 25
4.4 CASTR configuration tab. 26
4.5 CASTR analysis tab. 31
4.6 CASTR confusion matrix tab. 32
4.7 Example of imbalanced data. 34

5.1 Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Plasmashell dataset. Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None 43
5.2 Average precision and recall for top 10 recommendations. 49
5.3 Top 1, 3 and 5 precision and recall from all the recommendations created. . 50
5.4 Average recommender creation time in seconds. 54

A.1 Top 1, 3 and 5 precision for the recommendations created using different types of sampling techniques on LibreOffice dataset. Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None 66
A.2 Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Firefox dataset. Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None 68
A.3 Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Plasmashell dataset. Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None 70

C.1 Top 1, 3 and 5 precision and recall from all the recommendations created by participants. 79

Chapter 1

Introduction

In the era of agile software development, there are many things to handle collaboratively in a software development project, such as the development of a new feature, improvements to existing functionality or providing the resolution to an issue. Most software development projects use a software tool to track all changes made during the software development life cycle. This kind of tool is known as an *Issue Tracking System* or *Bug Tracking System*. Anvik *et* al. [2] have shown that the use of an issue tracking system makes a software project more manageable, especially when team members are geographically distributed. Issue tracking systems are used to help simplify these processes; they are dedicated platforms which enable teams to track projects from kick-off to delivery, or from identifying a bug to its resolution.

A large software development project receives a large number of bug reports every day, and the number of bug reports that need to be managed can become overwhelming [2]. When a new bug report is introduced into a software development project, it goes through different phases of the bug report life cycle. Each report must be examined to decide how the request will be handled by the project members. This decision process is called *bug report triage* and must be done for all incoming reports [1, 2]. In situations where a bug report needs a developer's consideration, a decision needs to be made about to whom the work will be assigned. An assignment decision can be made using bug report history, similar bug reports fixed by the developer, or other relevant parameters such as impacted component, Severity and Priority.

Bug triage takes significant time and resources [12]. Also, it is a tedious job to assign bug reports manually to the individual developers based on the view of the developer's ability or their bug fixing history. For reducing the workload of project members, bug report assignment recommenders have been proposed [2, 5, 17, 18, 30]. The creation of such recommenders for a specific software development project is a complex process as project members have to perform many steps such as data preparation, selection of machine learning algorithm and recommender creation.

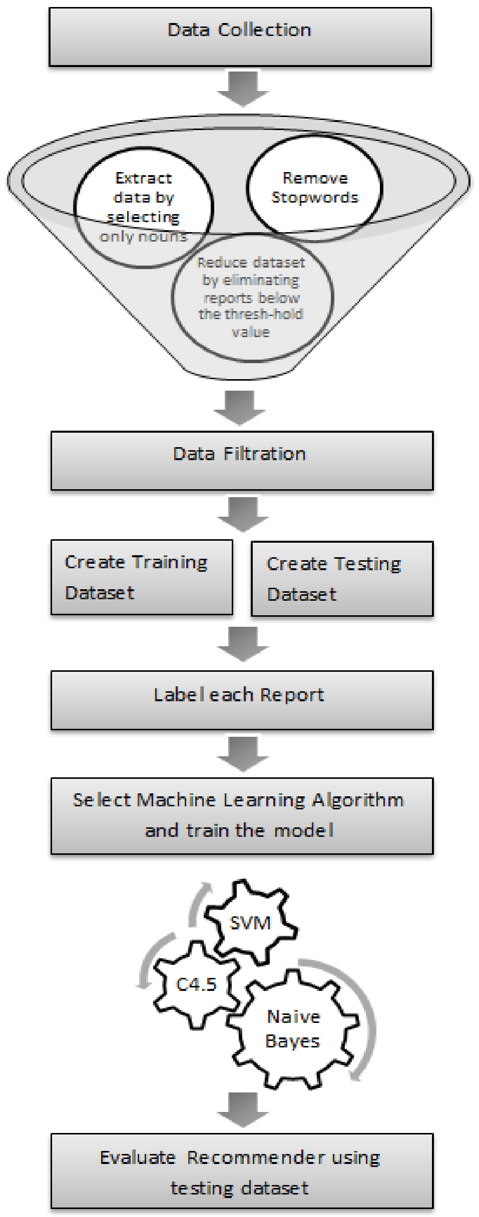

Data Collection
Extract data by selecting only nouns
Remove Stopwords
Reduce dataset by eliminating reports below the thresh-hold value
Data Filtration
Create Training Dataset
Create Testing Dataset
Label each Report
Select Machine Learning Algorithm and train the model
SVM
C4.5
Naive Bayes
Evaluate Recommender using testing dataset

Figure 1.1: Recommender creation process.

To begin with the recommender creation process (See Figure 1.1), the first step is for the project members to extract data from the set of bug reports. Next the data needs to be processed such as selecting only nouns from the bug reports description, removing stop words, setting a minimum threshold value to eliminate bug reports of developers who have fixed a small number of reports and are not appropriate for recommendation. After applying a filter to the dataset, the project members must label each bug report. For example, bug reports marked as Fixed will be labeled with the name of the last developer to attach a patch so that at the time of classification the machine learning algorithm will classify the bug report based on label. Labeling is a required stage of data preprocessing in the recommender creation with supervised learning algorithms. Labelers must be extremely attentive because each mistake or inaccuracy negatively affects a dataset's quality and the overall performance of a predictive model. After labeling, another key decision a project member has to make is the choice of a machine learning algorithm. In the last step, project members create the recommender and determine if the recommender works well. The creation of such recommenders for a specific software development project is challenging. Creating an assignment recommender can take many hours to days or weeks for someone who is unfamiliar with the process [1] given in Figure 1.1. For example, project members need to know machine learning algorithms, how the reports are labelled with developer names affects the set of names recommended, and the choice of valid names affects which reports and how many of them will be used for creating the recommender.

This dissertation presents a web-based tool called the *Creation Assistant for Supporting Triage Recommenders (CASTR)* to assist the project members with the creation of assignment recommenders. This thesis makes the following contributions:

1. We propose a platform-independent web-based tool that provides the following:
    (a) Allows a project member to analyze the dataset using a graphical representation.
    (b) Assists a project member in configuring project-specific parameters when creating a recommender.
    (c) Allows a project member to create an assignment recommender with the machine learning algorithm chosen by them.
    (d) Allows a project members to select a technique to handle the class-data imbalance issue.
    (e) Allows a project members to compare the last five recommenders created.
2. We implemented three different algorithms to handle class data imbalance issue: oversampling using SMOTE (Synthetic Minority Over-sampling Technique), undersampling using clusters and manual oversampling.

3. We compare the processing time of recommenders created by CASTR with recommenders tuned by hand.

4. We show, through analytical evaluations that recommenders can be created with good accuracy using machine learning algorithms.

5. We presents a field study showing how CASTR is used by professional developers and works in practice.

Using CASTR, we expect that a project member will be able to use their project knowledge to create an assignment recommender in a short period of time. Our main goal in creating CASTR is to understand the use of recommender creation systems for bug report assignment in practice.

CASTR is a platform-independent multi-tier web application with separate presentation and application layers. CASTR presents developer activity profiles and bug report state distributions, as well as the choice of four machine learning algorithms (SVM, Naive Bayes, C4.5, and Rule-based) for tuning the creation of the recommender.

1.1 Outline

The remainder of this thesis proceeds as follows. In the next chapter we provide information on bug reports, the lifecycle of a bug report, machine learning algorithms and RESTful APIs. The third chapter presents information on similar tools and related work done in the past. In the fourth chapter, we provide a more detailed background of our approach to assist with triage recommender creation and we explain about CASTR's functionality.

We explain our evaluation technique for CASTR in chapter five with presenting analytical and field study results. We then discuss various questions raised based on the field study result before making our concluding remarks and discussing future directions of study.

Chapter 2

Background

This section presents background information about bug reports, their life cycles, machine learning and recommendation systems in software engineering.

2.1 Recommendation Systems in Software Engineering

Recommendation systems in software engineering (RSSEs), are software applications that help to make a software engineering decision by providing a relevant suggestion on how to proceed. Recommendation systems are frequently used to derive contextualized recommendations for a variety of tasks. Given the wide variety of structured artifacts generated while generating software, recommendation systems have the opportunity to help developers complete their tasks.

Dumitru *et* al. [14] present that the field of recommender systems has been studied extensively, but mostly within the context of e-commerce systems, where numerous algorithms have been developed to model user preferences and create predictions. These algorithms vary greatly, depending on the type of data they use as input to create the recommendations. For example, some use content information about the items [24], or collaborative data of other user ratings [29], or knowledge rules of the domain [10], or hybrid approaches [11].

RSSEs have emerged to assist software developers in various activities from reusing code to writing effective bug reports [27]. In bug report triage, recommender systems can discover related or duplicate issue reports, help to find a suitable developer to assign to a bug report [3] and recommend source code to use for implementing features in a project.

2.2 Bug Reports

In software development, a bug is an error, a flaw, or a failure that produces an incorrect or unexpected result. Basically, a software bug is something where the software is not working as expected. A bug report, also known as a change request or issue report, is a document which contains a description of the behaviour caused by a fault in the software. A bug report is an efficient form of communication between users and members of the software project. Bug reports contain a variety of information such as a bug report identifier, a summary of the problem (often as a title), a description of how to reproduce the problem, a report creation timestamp, and the report's current status. A bug report may contain attachments or links to other related reports.

All bug reports have a lifecycle. It starts when a description of a bug is entered into the project's issue tracking system and ends when a bug report is closed. The bug report life cycle has many states. In general, bug reports follow the states given in Figure 2.1, but projects commonly add or remove states to fit their specific development process.

1. New: When a bug enters into the tracking system.

2. Assigned: The report has been assigned to a developer.

3. Resolved: There are many different ways that a report can be resolved, such as the developer making changes to the code (FIXED), identifying that the problem has already been reported (DUPLICATE), the request will not be addressed (WONTFIX), the described problem cannot be reproduced (WORKSFORME), or the problem is outside the scope of the project (INVALID).

4. Verified: This state indicates an independent verification of the fix by someone other than the developer. If a bug still exists then it will be reopened otherwise it will be closed.

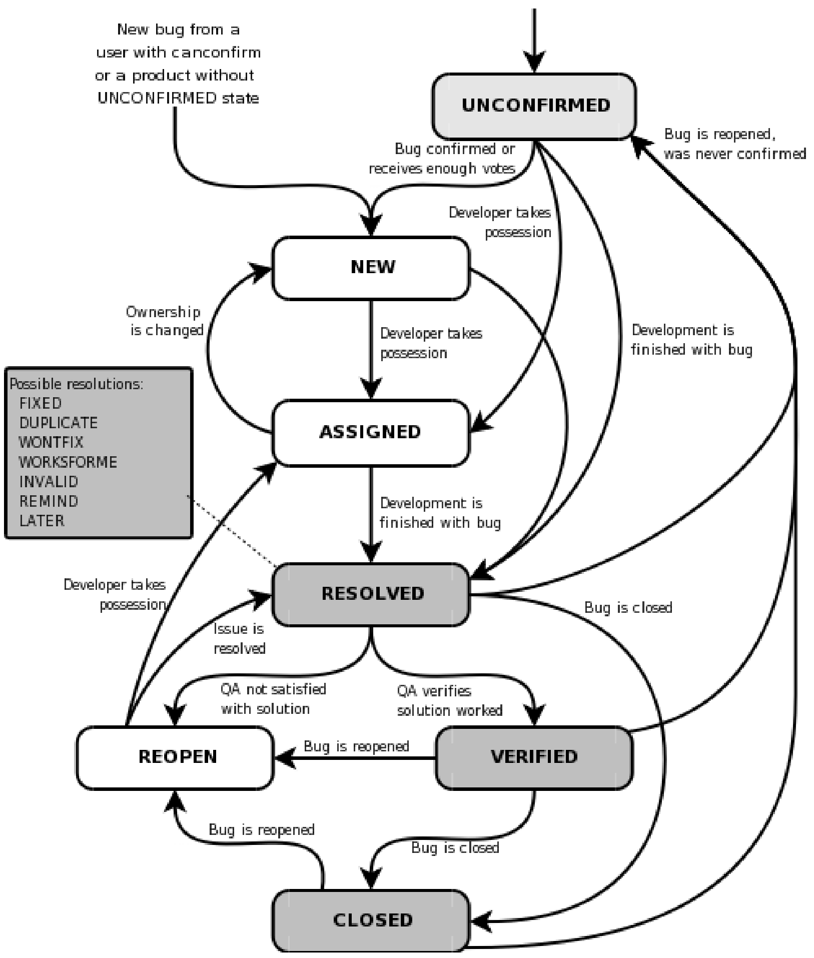

New bug from a user with canconfirm or a product without UNCONFIRMED state
UNCONFIRMED
Bug is reopened, was never confirmed
Bug confirmed or receives enough votes
Developer takes possession
NEW
Ownership is changed
Developer takes possession
Development is finished with bug
Possible resolutions:
FIXED
DUPLICATE
WONTFIX
WORKSFORME
INVALID
REMIND
LATER
ASSIGNED
Development is finished with bug
Developer takes possession
RESOLVED
Bug is closed
Issue is resolved
QA not satisfied with solution
QA verifies solution worked
REOPEN
Bug is reopened
VERIFIED
Bug is reopened
Bug is closed
CLOSED

Figure 2.1: Default lifecycle of a Bugzilla bug report.

5. Reopen: The report didnot pass verification, or the problem returned in a later release.

6. Closed: Once the report has been verified, the bug report will be closed.

2.3 Bug Report Triaging Workflow

Bug report triage is a process where issues are screened, prioritised and assigned a developer. For a given bug report, identifying an appropriate developer who could potentially fix the bug is the primary task of a bug triaging process. Each project follows their own method to deal with bug triaging.

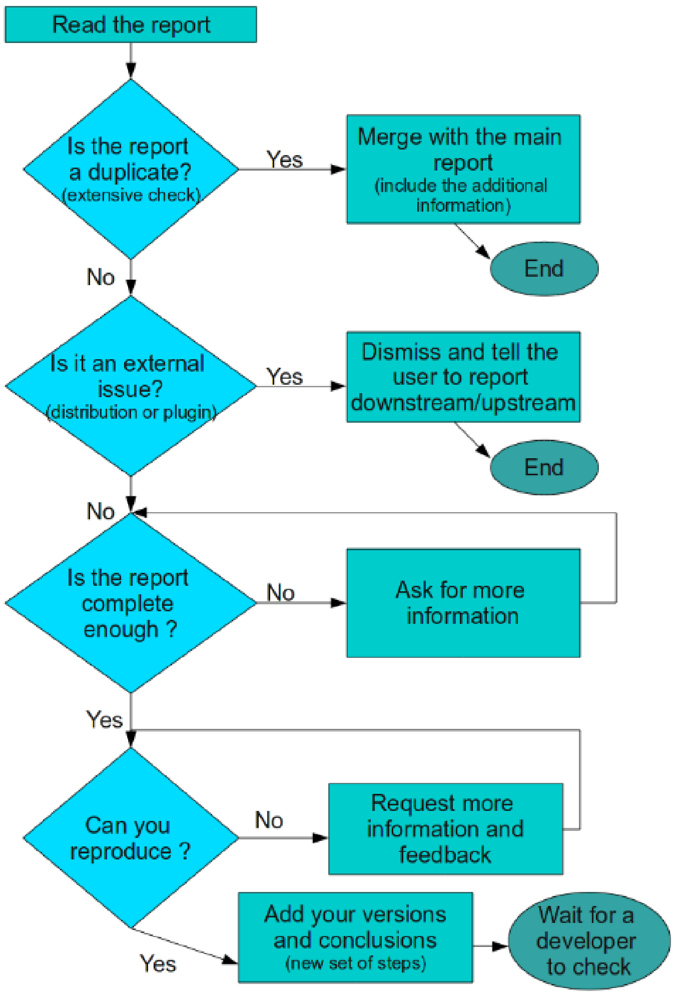

Read the report
Is the report a duplicate? (extensive check)
Yes
Merge with the main report (include the additional information)
End
No
Is it an external issue? (distribution or plugin)
Yes
Dismiss and tell the user to report downstream/upstream
End
No
Is the report complete enough ?
No
Ask for more information
Yes
Can you reproduce ?
No
Request more information and feedback
Yes
Add your versions and conclusions (new set of steps)
Wait for a developer to check

Figure 2.2: Bug report triage workflow of KDE open source project.

Figure 2.2 shows the optimal bug triaging workflow from the KDE project[1]. Before putting any effort into the newly introduced bug report, first the triager should check for an existing report. If triager finds a pre-existing bug report describing the same issue, mark the new bug report as a duplicate of it and merge the bug report with duplicate of bug report. The second step for the triager is to identify bugs caused by external issues. For example, the KDE app may experience an issue, but only when using the proprietary NVIDIA driver. The third step for the triager is to ask for any missing information if all the needed information is not provided. In the fourth step, a triager should use the information provided by the reporter and try to reproduce the bug on their own system. If the triager

[1]https://community.kde.org

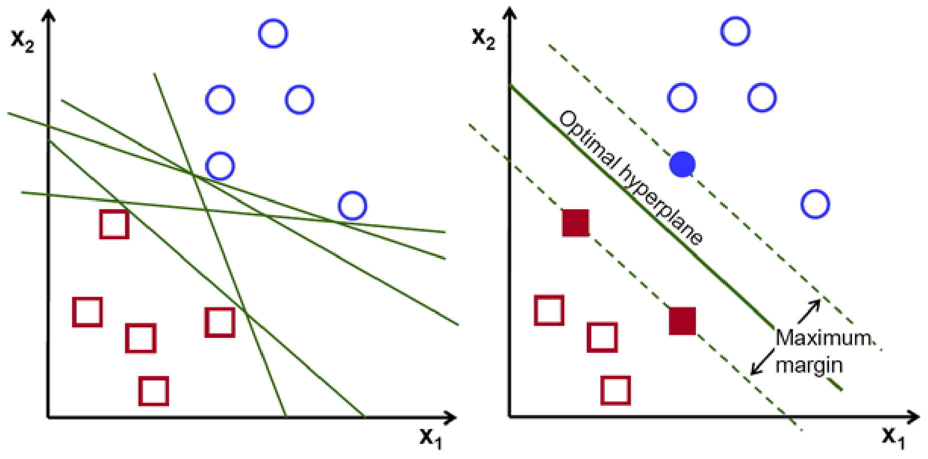
cannot reproduce the bug under any circumstances then they should ask the reporter for feedback or more information. In the last step, triager should provide useful information to the developers. For example, custom input data, environment and circumstances under which developer can reproduce the bug.

2.4 Machine Learning Algorithms

Arthur Samuel [28] in 1959 defined machine learning as the "*field of study that gives computers the ability to learn without being explicitly programmed*." Machine learning explores the study and construction of algorithms to make predictions on input data with the use of statistical analysis. Machine learning algorithms are classified into three different categories: supervised, unsupervised and reinforcement. In this work, we will primarily focus on supervised learning.

In supervised learning, a machine learning algorithm is trained on a training set [7]. The process to learn the general rules is known as training the recommender. The algorithm will apply the same rules derived from the training set to all of the other data. Working with a supervised machine learning algorithm requires an understanding of three concepts: *feature*, *instance*, and *class*. A feature is a property or characteristic which is used to determine the class. Features are extracted from the training data instances such as text, numbers, or nominal values in the description of bug reports. An instance is a group of features that have specific values such as a description of a bug report. A class is a collection of instances which belong to the same category such as bug reports fixed by a specific developer. In supervised learning, each training instance is labeled with their class, which for bug report assignment is the name of the developer that is believed to have fixed the bug. In CASTR, we have used the following machine learning algorithms (SVM, Naive Bayes, C4.5, and Rule-based).

In unsupervised learning, all data is unlabeled and the algorithms learn the inherent structure from the input data. We use one of the most common algorithms Expectation-



(a) Possible hyperplanes (b) Optimal hyperplane

Figure 2.3: Hyperplane in 2-dimensional space.

maximization in unsupervised learning to handle imbalanced data.

2.4.1 Support Vector Machines

A support vector machine (SVM) [25] constructs a hyperplane or set of hyperplanes in a N-dimensional space. A SVM draws a decision boundary as given in the Figure 2.3(b), which is also known as hyperplane which segregates the distinct classes in N-dimensional space.

To illustrate a SVM, we use an example of a linear dataset of bug reports and seek to classify whether a new bug report is to be fixed by developers Kim or James. The developers names are represented by the shapes (Kim (circle) and James (square)). So how do we decide where to draw our decision boundary? We can draw it at many places as given in the Figure 2.3(a). Any of these would be fine, but what would be the best? If we do not have the optimal decision boundary we could classify a bug report. Therefore, we draw an arbitrary separation line and we use intuition to draw it somewhere between the solid blue circle and the solid red square data point for both of the developer classes as given in the Figure 2.3(b). The solid blue circle and solid red square are the extreme data points called *support vectors*. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. The larger the maximum margin,the better the classifier output would be. The algorithm implies that only the support vectors are important whereas other training data points are ignorable.

2.4.2 Naive Bayes

Naive Bayes [20] is a simple technique for constructing a classifier using Bayes' theorem, which is also known as Bayes' law or Bayes' rule. Bayes' theorem describes the probability of an event, based on prior knowledge of the conditions that might be related to the event. The core equation for the Bayes' theorem is:

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)} \tag{2.1}$$

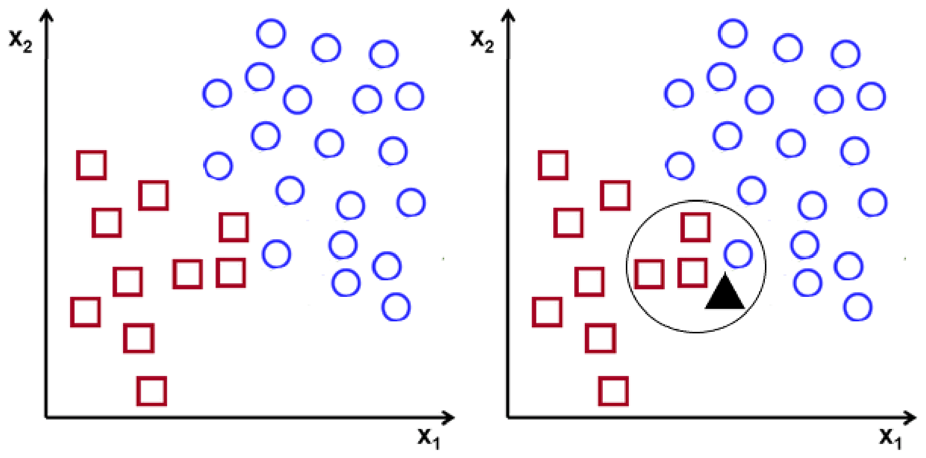
where A and B are events and $P(B) \neq 0$.

1. $P(A \mid B)$ is a conditional probability: the likelihood of event A occurring given that B is true.

2. $P(B \mid A)$ is also a conditional probability: the likelihood of event B occurring given that A is true.

3. $P(A)$ and $P(B)$ are the probabilities of observing A and B independently of each other; this is known as the marginal probability.

To demonstrate the concept of Naive Bayes classification, consider the example of bug reports given in the Figure 2.4(a). As indicated, the report can be classified as either developer Kim or James. Our task is to classify new bug reports based on the report summary text (i.e., decide to which class label the bug report belong). Since there can be many reports fixed by each developers, it is reasonable to believe that a new report is given to the developer Kim who has fixed twice as many reports as James in the past with the similar



(a) Bug reports fixed by developer Kim and James. (b) Classify new bug report belongs to Kim or James?

Figure 2.4: Naive Bayes classification

text given in the report summary. In the Bayesian analysis, this belief is known as the prior probability. We must first find prior probabilities for each class. Prior probabilities are based on previous experience.

$$Prior\ Probability\ for\ Kim = \frac{\#of\ reports\ fixed\ by\ Kim}{\#\ of\ total\ reports} \tag{2.2}$$

$$Prior\ Probability\ for\ James = \frac{\#\ of\ reports\ fixed\ by\ James}{\#\ of\ total\ reports} \tag{2.3}$$

After formulating the prior probability, now we can classify a new report (represented by the solid black triangle shape in the Figure 2.4(b)). Since the bug reports are well clustered, it is reasonable to assume that the more reports fixed by Kim in the vicinity of "X", the more likely that the new report belongs to that particular developer. To measure this likelihood, we draw a circle around X which encompasses a number of points irrespective of their class labels. Then we calculate the number of points in the circle belonging to each class label. From this we calculate the likelihood:

$$Likelihood\ of\ new\ report\ given\ to\ Kim = \frac{\#\ of\ reports\ fixed\ by\ Kim\ in\ the\ vicinity\ of\ X}{\#\ of\ reports\ fixed\ by\ Kim} \tag{2.4}$$

$$Likelihood\ of\ new\ report\ given\ to\ James = \frac{\#\ of\ reports\ fixed\ by\ James\ in\ the\ vicinity\ of\ X}{\#\ of\ reports\ fixed\ by\ James} \tag{2.5}$$

From the illustration above, it is clear that the likelihood of X given Kim ($P(X \mid Kim)$) is smaller than the likelihood of X given James ($P(X \mid James)$), since the circle encompasses 1 report fixed by Kim and 3 reports fixed by James. Although the prior probabilities indicate that X may belongs to Kim, the likelihood indicates otherwise; that the class membership of X is James (given that there are more bug reports fixed by James in the vicinity of X than Kim). In the Bayesian analysis, the final classification is produced by combining both sources of information, i.e., the prior and the likelihood, to form a posterior probability using Bayes' rule. As per the equation below, we classify that the new report belongs to James since its class membership achieves the largest posterior probability.

$$Posterior\ probability\ of\ Kim = Prior\ Probability\ of\ Kim \times Likelihood\ of\ Kim \tag{2.6}$$

$$Posterior\ probability\ of\ James = Prior\ Probability\ of\ James \times Likelihood\ of\ James \tag{2.7}$$

2.4.3 C4.5 Decision Tree

Quinlan [26] presents the classification algorithm C4.5 which generates a decision tree using attributes values of instances for the given dataset. The construction of decision

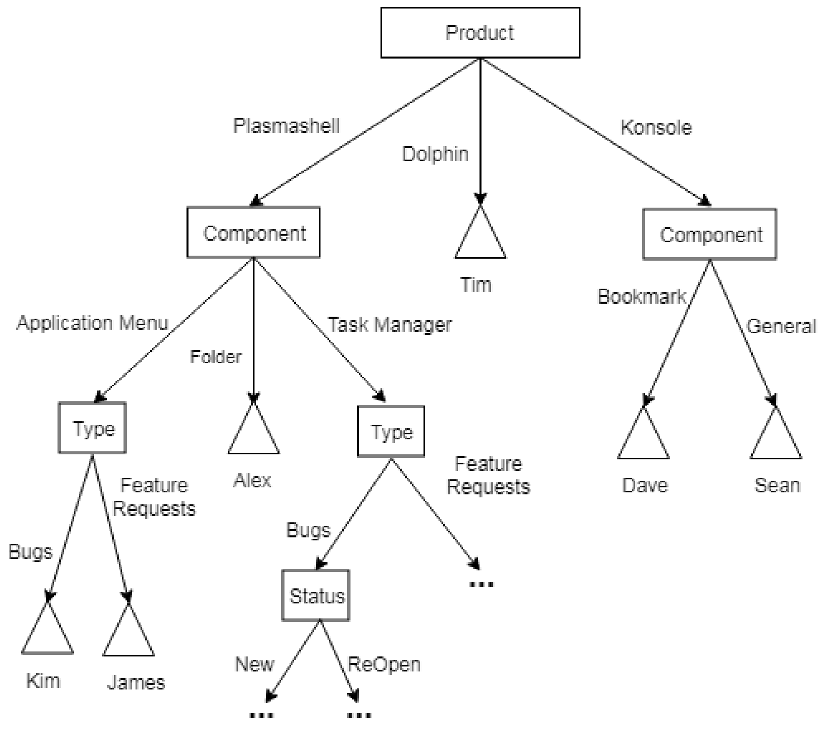


Figure 2.5: Example of a Decision Tree.

tree classifiers does not require any prior knowledge or parameter setting, and therefore is appropriate for exploratory knowledge discovery [16]. Decision trees can handle multidimensional data. Their representation of acquired knowledge in tree form is intuitive and generally easy to understand by humans.

C4.5 is a greedy algorithm in which decision trees are constructed in a top-down recursive manner using the concept of information entropy. The algorithm starts with the root node by selecting an attribute from the given instances. The selection criteria of the attribute depends on the information gain. The attribute with the highest information gain is deemed the best choice. Then it creates branches for each possible attribute values. Further, it splits instances into subsets recursively until all instances reach a leaf node. The leaf nodes represents the classes.

Figure 2.5 shows an example of a top part of a decision tree created for assiging bug reports to a developer using the KDE project dataset. The example shows that the C4.5 algorithm chose the Product attribute as root node of the tree. Depending on the value of the product in the bug report, one of three paths is followed. If the product is Plasmashell, it will again follow one of three paths depending on the value of the component field. If the component is Application Menu then it is checking whether the new bug report is a type of bug or a feature request. If the type is Bugs then the bug report is assigned to Kim otherwise it is assigned to James. If the product type is Dolphin then the report is assigned to Tim. For the component Task Manager, the path is followed by more levels to decide whom to assign the bug report. The leaf nodes are the developer names to be assigned to a bug report.

2.4.4 Rules

In rule-based classifiers, a trained model is represented as a set of IF-THEN rules. The IF part of a rule is known as the rule antecedent or precondition and the THEN part is the rule consequent. In the rule antecedent, the condition consists of one or more attribute tests that are logically ANDed. The rules consequent contains a class prediction. If the condition (i.e., all the attribute tests) in a rule antecedent holds true for a given instance, we say that the rule antecedent is satisfied. For example:

R1: IF product = Plasmashell AND component = Folder THEN assign report = Alex

Rules can be generated, either from a decision tree or directly from the training data using a sequential covering algorithm. In comparison with a decision tree, the IF-THEN rules may be easier for humans to understand, particularly if the decision tree is very large.

To extract rules from a decision tree, one rule is created for each path from the root to a leaf node. Each splitting criterion along a given path is logically ANDed to form the rule antecedent. The leaf node holds the class prediction, forming the rule consequent. A disjunction is implied between each of the extracted rules. Because the rules are extracted directly from the tree, they are mutually exclusive and exhaustive. Mutually exclusive means that we cannot have rule conflicts here because no two will be triggered for the same instance. Exhaustive means there is one rule for each possible attribute value combination, so that this set of rules does not require a default rule. Therefore, the order of the rules does not matter. The following are some of the rules extracted from the example of a decision tree given in Figure 2.5,

R2: IF product = Konsole AND component = Bookmark THEN assign report = Dave

R3: IF product = Dolphin THEN assign report = Tim

R4: IF product = Plasmashell AND component = Application Menu AND type = Bugs THEN assign report = Kim

2.4.5 Expectation-maximization

An expectation-maximization (EM) algorithm is a framework that approaches Maximum Likelihood (ML) or maximum a posteriori estimates of parameters in statistical models. Expectation-maximization algorithms can be used to compute fuzzy clustering and probabilistic model-based clustering. The EM algorithm is an efficient iterative procedure to compute the ML estimate in the presence of missing or hidden data. Each iteration of the EM algorithm consists of two processes: The E-step, and the M-step.

1. In the expectation, or E-step, the missing data are estimated given the observed data and current estimate of the model parameters. This is achieved using the conditional expectation, explaining the choice of terminology.

2. In the M-step, the likelihood function is maximized under the assumption that the missing data are known. The estimate of the missing data from the E-step are used in lieu of the actual missing data.

We use the EM algorithm as the basis of unsupervised learning of clusters. We implemented a method for under-sampling a dataset using the EM algorithm by removing some of the bug reports in the majority class. EM assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters.

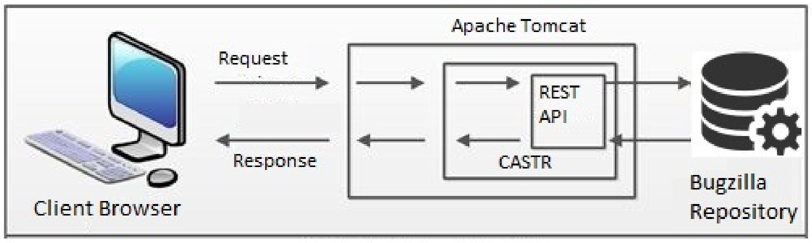


Figure 2.6: CASTR application structure.

2.5 RESTful APIs

REST stands for Representational State Transfer, a term coined by Roy Fielding [15]. It is an architecture style for designing loosely coupled applications over HTTP, that is often used in the development of web services. REST does not enforce any rule regarding how it should be implemented at a lower level rather it puts forth high-level design guidelines and enables users to provide their own implementation. RESTful web services allow the requesting systems to access and manipulate textual representations of web resources by using a uniform and predefined set of stateless operations. By using a stateless protocol and standard operations, RESTful systems aim for fast performance, reliability, and the ability to grow, by re-using components that can be managed and updated without affecting the system as a whole, even while it is running. Web service APIs that adhere to the REST architectural constraints are called RESTful APIs.

Figure 2.6 shows that how CASTR handles a request using the RESTful API and sends the response back. CASTR's presentation layer makes a request to CASTR web services whenever data processing is required. For downloading the dataset CASTR web service makes a request to Bugzilla REST API.

Chapter 3

Related Work

As the manual assignment of bug reports to a developer is a monotonous job, many researchers have proposed using assignment recommenders to reduce the amount of resources to be invested towards bug triage [1, 2, 5, 17, 18, 30]. All previous work has sought to find specific answers to creating an assignment recommenders however none of these works focus on assisting with the recommender creation process, except work done by Anvik *et* al. [1].

3.1 Assisting With Bug Report Triage

Assisting with recommender creation is an emerging research area in software engineering. There are similar tools to the CASTR, such as CASEA [1], Porchlight [8], and to a lesser extent, Porchlight's predecessor TeamBugs [9].

3.1.1 CASEA: Creation Assistant Supporting Triage Recommenders

Anvik *et* al. [1] implemented the software tool CASEA to assist a software project in creating and maintaining bug assignment recommenders. CASEA is an implementation of the approach proposed by Anvik and Murphy [2] that leverages a project members knowledge to assist in creating and managing bug report assignment recommender configurations. CASEA assists in the creation of project-specific heuristics and the selection of valid developer recommendations. A project-specific heuristic is a generalized rule used to label a bug report with the name of the developer that resolved the problem. CASEA also assists a user in labeling and filtering the bug reports used for creating a project-specific assignment recommender. CASEA guides a project member through the assignment recommender creation process in four steps: Data Collection, Data Processing, Recommender Training, and Evaluation. CASEA provides three different ways to collect data:

- Connect to the project's issue tracking system (ITS) through a web interface URL (e.g. XML-RPC, REST).
- Connecting to the project's database.
- Importing training and validation XML files containing bug reports.

Once the data is imported into CASEA, the system performs two types of filtering: automatic and assisted. The automatic filtering performs three actions on the textual data such as removing stop words, stemming and removal of punctuation and numeric values. CASEA supports two types of assisted filtering: label filtering and instance filtering. A label frequency graph is used for label filtering and project-specific heuristics are used for instance filtering. Once the data processing is done, CASEA performs recommender creation using two common machine learning algorithms: Support Vector Machines and Naive Bayes. CASEA provides analysis data like the time taken to train the recommender, and evaluation results in the form of precision and recall metrics [1].

CASEA was developed as a desktop application in 2013 on the Microsoft platform using the C# programming language. CASEA had a number of limitations including: performance issues, the audience was limited to Windows users, it does not address data imbalances, and it does not distinguish between complex and simple source code changes. Also, CASEA was never field tested.

CASTR is a refinement of CASEA [1]. This thesis presents a web-based tool which is platform independent and addresses some of the limitations of CASEA.

3.1.2 Porchlight / TeamBugs

The Porchlight tool uses tags, attached to individual bug reports by queries expressed in a specialized bug query language, to organize bug reports into sets so developers can explore, work with, and ultimately assign bugs effectively in meaningful groups [8]. This tagging functionality is similar to the grouping of reports in CASTR, which also groups bug reports into categories for specifying and applying labeling heuristics.

The TeamBugs tool was integrated with their project-specific bug tracker to populate the list of bug reports which need to be triaged. The tool shows a list of all the developers who are mapped to the project so that a project member can assign new bug reports to the appropriate developer. The bug report is assigned to a developer by drag-and-droping the bug report on a developer's name, which labels the bug report. This labeling functionality is similar to that of CASTR, which also labels the bug reports with a developers name based on the resolution group.

3.2 Assisted Machine Learning Recommender Creation

There are commercial products that can be viewed as similar to the CASTR tool, such as Skytree Infinity, BigML, Azure ML, Amazon ML and Google Cloud ML.

Skytree Infinity [31] (formally called Skytree Server) provides a platform to make profoundly precise and versatile analytic solutions utilizing machine learning technologies. It is designed to be an extremely scalable platform to enable data scientists and analysts to focus more on building analytic solutions rather than implementing algorithms. It performs tasks such as data prediction, exploration, and transformations. Like CASTR, Skytree Infinity assists in preparing data for use with a machine learning algorithm, visualizing the data and assisting in selecting the most appropriate parameter values when building a model utilizing a machine learning algorithm.

BigML [6] is a simple, highly adaptable machine learning platform that provides cloud-based service to automate prediction data. BigML accepts data in comma-separated values

(CSV) format to prepare a data set for use by a machine learning algorithm. The user can then generate a decision tree model from the data. If desired, a user can return to the data set creation step and deselect different attributes to tune the model. Unlike CASTR, BigML provides no support for data preparation such as stemming and stop word removal - the user is expected to perform such tasks before use. Also, BigML does not provide any visualization of the data beyond a tabular format. The utilization of BigML for making a bug assignment recommender would require a lot of effort with respect to the users, whereas CASTR tries to reduce required efforts.

Microsoft Azure ML [22], Amazon ML [21], and Google Cloud ML [23] are GUI-based integrated development environments for constructing and executing a machine learning workflow. To initialize a model in Azure ML, the system gives numerous alternatives to choose the most appropriate algorithm to classify data. Azure ML additionally supports R language modules and Python scripts. The data model is deployed as a fully managed web service on the cloud that connects to any data anywhere. The user can save and edit the solution anytime. Amazon ML provides visualization tools and wizards that create machine learning models. The user does not need to learn complex algorithms. Amazon ML creates data patterns from existing data and then it makes predictions. Google Cloud ML creates the data model with the TensorFlow framework and performs large scale training on a managed cluster. It also manages batch predictions and supports a huge number of users.

All the commercial machine learning products (Skytree, BigML, Azure, Amazon and Google Cloud) are mainly developed for business solutions rather than focusing on bug report triage recommenders. The main objective of CASTR is to concentrate on bug triage workflows and to integrate the tool with open source projects so that real-world software projects can assess the system. CASTR provides visualizations such as frequency charts and graphs to analyze bug reports.

Chapter 4

An Approach to Assisting with Triage Recommender Creation

This chapter presents an approach to assisting with bug report triage recommender creation that guides the triager in creating a recommender for bug report assignment. We start with an overview of the methodology. Next, we show how CASTR helps a user through the bug report assignment creation process for those who use the Bugzilla[2] issue tracking system. To demonstrate how CASTR is used, we have chosen to use the Plasmashell product from KDE[3], the LibreOffice from the Document Foundation[4] project, and Firefox from the Mozilla[5] project. We chose to use these three open source projects for demonstrating CASTR as these projects have large development communities, many products[6], and have a large set of reports for training an assignment recommender.

4.1 An Overview of Bug Report Assignment Recommender Creation

A similar process to that given in the Figure 1.1 is used to create a recommender that suggests which developer should be assigned the responsibility for resolving a particular bug. However, applying our approch in practice is complex as many related decisions must be made to create the recommender. Specifically, the following questions must be answered

[2]Bugzilla is a web-based general-purpose bugtracker and testing tool. A large number of companies, organizations, and projects use Bugzilla.

[3]KDE is software community which developes an open source softwares. It provides tools and resources that allow collaborative work and can be found at www.kde.org

[4]The Document Foundation promotes open-source document handling software and can be found at www.documentfoundation.org

[5]Mozilla Firefox is a web browser and can be found at www.mozilla.org/en-US/firefox

[6]KDE-748 products, LibreOffice-7 products and Mozilla-167 products

in order to create a bug report assignment recommender:

1. From which project should the bug reports be extracted?
2. How many bug reports should be used to create the recommender?
3. How should the bug reports be labeled?
4. How many and which project heuristics should be used?
5. Which machine learning algorithm should be used to create the recommender?
6. What is the minimum number of bug reports that a developer should have resolved to be recommended?
7. What technique should be used to handle data imbalances?

The next section presents our answer to the list of questions, the first three questions are part of the data collection and remaining questions arise at the time of recommender creation.

4.2 CASTR

In this section, we present how CASTR works in practice. CASTR first asks a user to login into the web application using a username and password. Each unique user will have their own storage space on the server to save the data. Figure 4.1 shows the login interface of CASTR.

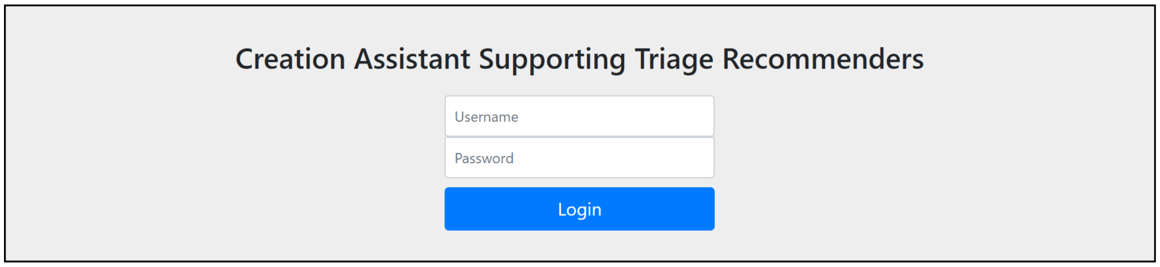
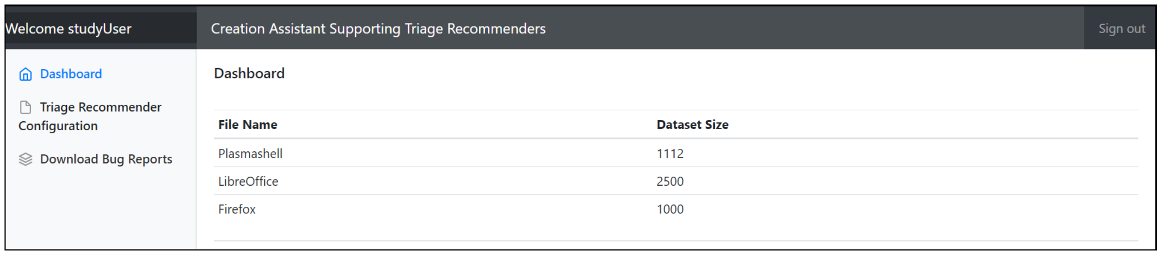


Figure 4.1: CASTR login screen.


Welcome studyUser
Creation Assistant Supporting Triage Recommenders
Sign out
Dashboard
Triage Recommender Configuration
Download Bug Reports
Dashboard

File Name	Dataset Size
Plasmashell	1112
LibreOffice	2500
Firefox	1000

Figure 4.2: CASTR dashboard screen.

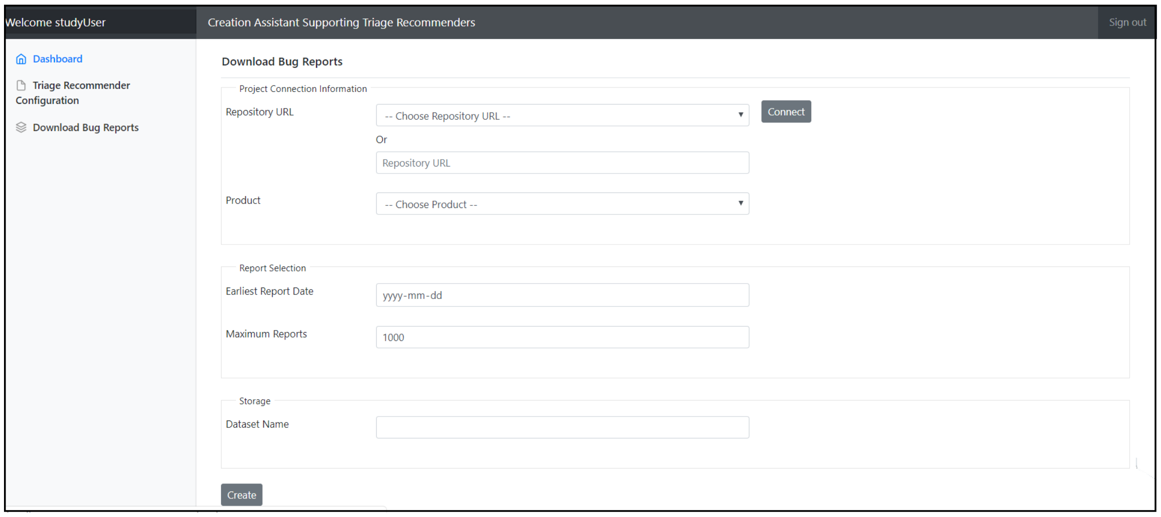
Once the user is logged into CASTR, the dashboard screen will be displayed as given in Figure 4.2. It provides information about previously downloaded datasets and the total number of bug reports in the dataset. It also provides the access link to download a new dataset (see Section 4.2.1) and to upload triage recommender configurations (see Section 4.2.2).

4.2.1 Downloading the Dataset

To answer the question "From which project should the bug reports be extracted", CASTR provides a web interface as given in the Figure 4.3 to download the bug reports from a Bugzilla repository. The system can be extended in the future to support data collection from other issue tracking systems. The user can select a project repository URL from the given list[7][8][9] or can provide a Bugzilla repository URL to download the bug reports. Once the user is connected to the repository's REST end point, all of the products of the project will be filled up in the selection list. In the Bugzilla repository, the "product" is also known as the "project". Also, the user can specify the initial bug report creation date from which to start pulling reports, and the number of reports to gather. When the user clicks on the "Create" button, the CASTR frontend forwards the request to the Bugzilla repository via CASTR's backend services.

Table 4.1 shows the parameters we used for collecting the three datasets from KDE, Document Foundation, and Mozilla. As shown in Figure 4.3, CASTR enables the user to

[7]https://bugs.kde.org/jsonrpc.cgi
[8]https://bugs.documentfoundation.org/jsonrpc.cgi
[9]https://bugzilla.mozilla.org/jsonrpc.cgi


Welcome studyUser
Creation Assistant Supporting Triage Recommenders
Sign out
Dashboard
Triage Recommender Configuration
Download Bug Reports
Connect
Create

Download Bug Reports

Project Connection Information

Repository URL: -- Choose Repository URL --

Or

Repository URL

Product: -- Choose Product --

Report Selection

Earliest Report Date: yyyy-mm-dd

Maximum Reports: 1000

Storage

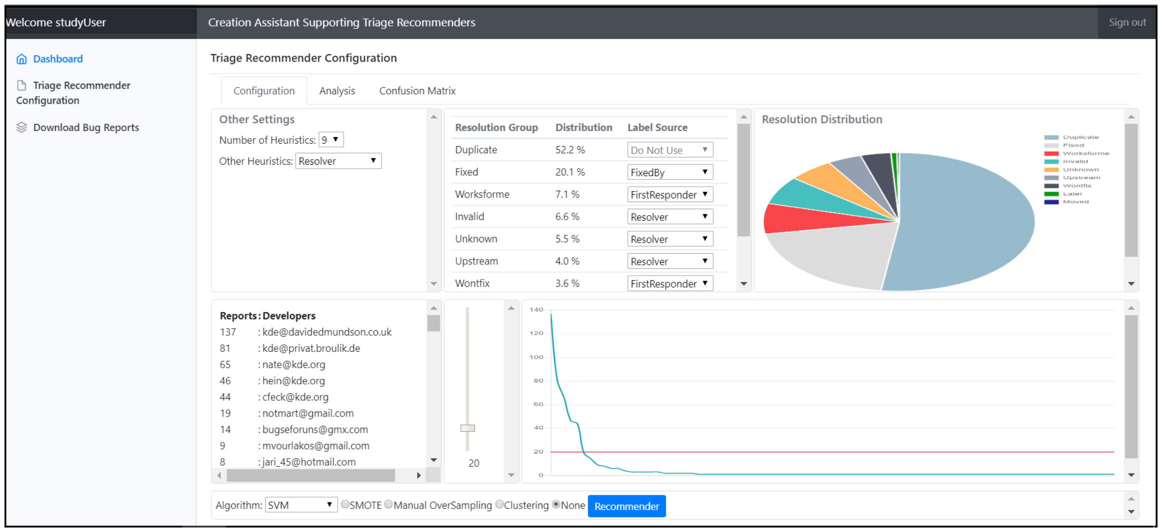
Dataset Name:

Figure 4.3: CASTR downloading bug reports screen.

download the bug reports using two fields: earliest report date and maximum reports. The earliest report date is a report creation date in the past and the maximum reports field is set to the maximum number of reports to be downloaded. For example, we made a request to download a maximum of 1500 reports which were reported on or after Jan 1, 2018 for the Plasmashell project. The size of the dataset for Plashmashell project was only 1112, showing that some projects may not have a pre-chosen fixed quantity reports. Alternatively, a large number of reports can be downloaded by setting the earliest report date to be 3 months earlier. We determined the quantity of the reports based on the selection of the time period and providing a fixed quantity such as six months and 1000 to 5000 reports so that we can train a recommender with a sufficient number of reports.

Table 4.1: Projects with different date range and number of reports used for tuning recommenders.

Projects	Start Date	End Date	# of Bug Reports
Plasmashell	Jan 1, 2018	Feb 23, 2019	1112
LibreOffice	Jun 1, 2018	Feb 23, 2019	2500
Firefox	Jun 1, 2018	Nov 13, 2018	1000



Other Settings

Number of Heuristics: 9

Other Heuristics: Resolver

Resolution Group	Distribution	Label Source
Duplicate	52.2 %	Do Not Use
Fixed	20.1 %	FixedBy
Worksforme	7.1 %	FirstResponder
Invalid	6.6 %	Resolver
Unknown	5.5 %	Resolver
Upstream	4.0 %	Resolver
Wontfix	3.6 %	FirstResponder

Reports	Developers
137	kde@davidedmundson.co.uk
81	kde@privat.broulik.de
65	nate@kde.org
46	hein@kde.org
44	cfeck@kde.org
19	notmart@gmail.com
14	bugseforuns@gmx.com
9	mvourlakos@gmail.com
8	jari_45@hotmail.com

Algorithm: SVM SMOTE Manual OverSampling Clustering [✓ None]

Figure 4.4: CASTR configuration tab.

4.2.2 Triage Recommender Configuration

CASTR shows developer activity profiles, bug report distribution based on status, precision for the top one, three and five developer recommendations and a confusion matrix for the created recommender. In the triage recommender configuration screen there are three tabs: Configuration, Analysis, and Confusion Matrix. Each tab displays the relevant information.

Configuration Tab

Information about the collected dataset will be displayed under the Configuration tab, as shown in the Figure 4.4. Through configuration options, a user will be able to perform data filtration by setting project-specific heuristics for labeling the reports, and a minimum threshold value for which labels to recommend.

As shown in the Figure 4.4, CASTR provides four supervised machine learning algorithms for creating a recommender: SVM, Naive Bayes, C4.5, and Rules. CASTR also supports the handling of data imbalances with three different approaches: oversampling using SMOTE, undersampling using clusters and manual oversampling.

To show which projects heuristics will be the most useful for labeling, CASTR shows resolution-wise the general distribution of the bug reports. Based on the dataset for each project, the number of heuristics will be different as shown in Table 4.2. The number of heuristics will be equal to the number of resolution groups. By changing the number of heuristics, the display of the resolution distribution is updated. For example, if 3 is selected as number of heuristic then only heuritics for the first 3 resolution groups are labeled. All remaining bug reports will be consolidated into the "Other" resolution group and the selected value will be used to label all of the remaining bug reports. For example, if we change the number of heuristics to five for LibreOffice then bug reports for the the last three resolution groups will be grouped into the "Other" resolution group and labeled separately.

Labeling Instances

To train a bug report assignment recommender, we need to provide a set of reports that are labeled with the name of the developer who was either assigned to the report or who resolved it. Labeling is an important stage of data preprocessing in supervised learning. A training algorithm must be shown which target label to recommend. Mapping of the target attribute in a dataset is called labeling.

For an assignment recommender, this step appears to be simple as it seems obvious to use the value of the *assigned-to* field in the report. However, the problem is not that simple because projects tend to use the *status* and *assigned-to* fields of a report differently. For

Table 4.2: Example of dataset distribution based on resolution group.

Plasmashell		**LibreOffice**		**Firefox**	
# of Heuristics: 9		# of Heuristics: 8		# of Heuristics: 7	
Resolution Group	**Distribution (%)**	**Resolution Group**	**Distribution (%)**	**Resolution Group**	**Distribution (%)**
Duplicate	52.2	Fixed	37.6	Fixed	43.3
Fixed	20.1	Duplicate	31.0	Duplicate	22.3
Worksforme	7.1	Worksforme	11.2	Incomplete	15.8
Invalid	6.6	Notabug	8.3	Invalid	7.8
Unknown	5.5	Invalid	4.5	Worksforme	5.7
Upstream	4.0	Wontfix	3.5	Wontfix	4.9
Wontfix	3.6	Unknown	3.1	Unknown	0.2
Later	0.7	Moved	0.7		
Moved	0.3				

example, in all the projects from the three chosen repositories, the value of the *assigned-to* field does not initially refer to a specific developer, but is first assigned to a default email address before the first report is assigned to an actual developer. For reports with a different resolutions, such as duplicate, or reports with a trivial fix, such as changing the access modifier of a method, the *assigned-to* field is often not changed.

Instead of blindly using the *assigned-to* field, we use project-specific heuristics to label the reports. These heuristics can be derived either from direct knowledge of a project member or by examining the logs of a random sample of reports for the project. However, before we can label the bug report, we need to decide where to get the inforation from the bug report to use as a label. The set of valid labels is determined from the bug report dataset. Based on examining the bug report resolution and activity history for various projects, we determined a set of valid data sources (see Table 4.3) from which to extract labels.

The user can label bug reports using resolution groups. For example, bug reports marked as *Fixed* can be labeled as *FixedBy* so that at the time of classification it will classify the bug report based on this label. To label other bug reports the user needs to select “Other Heuristics” value. In Other Heuristics, the selected value will be labeled for all remaining bug reports. It will impact when the user sets a smaller number of heuristic than the number of resolutions. Based on the heuristic and label source, the developer profiles (developer name and number of bug reports resolved) will be calculated and displayed.

Table 4.4 shows an example of project-specific heuristics. For example, if a report is

Table 4.3: Examples of valid data sources.

Valid Label	**Description**
AssignedTo	The value of the assigned-to field
FixedBy	The last person to mark the bug report with resolution fixed
Resolver	The person who marks the bug report as resolved
Reporter	The person who reports the bug report
FirstResponder	The person that first responds to the bug report in the comments
Attachment	The last person to submit an attachment to the bug report
DoNotUse	Ignores bug report for the classification

Table 4.4: Examples of project heuristics used for labeling reports.

Resolution Group	**Label**		
	Plasmashell	**LibreOffice**	**Firefox**
Fixed	FixedBy	FixedBy	Attachment
Worksforme	FirstResponder	FirstResponder	Resolver
Invalid	Resolver	Resolver	Resolver
Unknown	Resolver	Resolver	Resolver
Upstream	Resolver	-	-
Wontfix	FirstResponder	Resolver	FirstResponder
Later	Resolver	-	-
Moved	Resolver	Resolver	-
Duplicate	DoNotUse	DoNotUse	DoNotUse
Notabug	-	Resolver	-
Incomplete	-	-	FirstResponder

resolved as Fixed, is labeled with whoever has fixed the report. If a report is resolved as Worksforme, is labeled with whoever responded first to the report.

Selecting Valid Labels

The user can set a minimum threshold to eliminate developers that have fixed a small number of bug reports. For example, developers who has resolved less then 10 bug reports according to the project heuristics may not be considered for recommendation. As shown in Figure 4.4, we took 20 as minimum threshold value to recommend the set of developers. The line chart presents developer-wise bug reports and the red line indicates the minimum threshold. CASTR removes from labeling bug reports for developers falling under the lower threshold.

Selection of the Machine Learning Algorithm

There are several machine learning algorithms that can be used to create a triage recommender (see Section 2.4). To determine an appropriate algorithm for recommender creation, we evaluated four different algorithms for the three different projects of Plasmashell, LibreOffice and Firefox. We chose to investigate Naive Bayes [20], Support Vector Machines

[25], C4.5 [26] and Rules [26] as they cover the different categories of supervised machine learning algorithms.

To fully evaluate the effectiveness of a recommender created using the four different algorithms, we examined both precision (*the fraction of relevant instances among the retrieved instances*) and recall (*the fraction of relevant instances that have been retrieved over the total amount of relevant instances*). Improving precision typically reduces recall and vice versa. The precision and recall of the recommenders created using the four algorithms is presented in Table 4.5. For the recommender creation we have used the project-specific heuristics given in Table 4.4 to label the bug reports and we set minimum threshold number as 20 to eliminate developer falling under the lower threshold. The minimum threshold can vary from project to project and based on the size of the developer profiles.

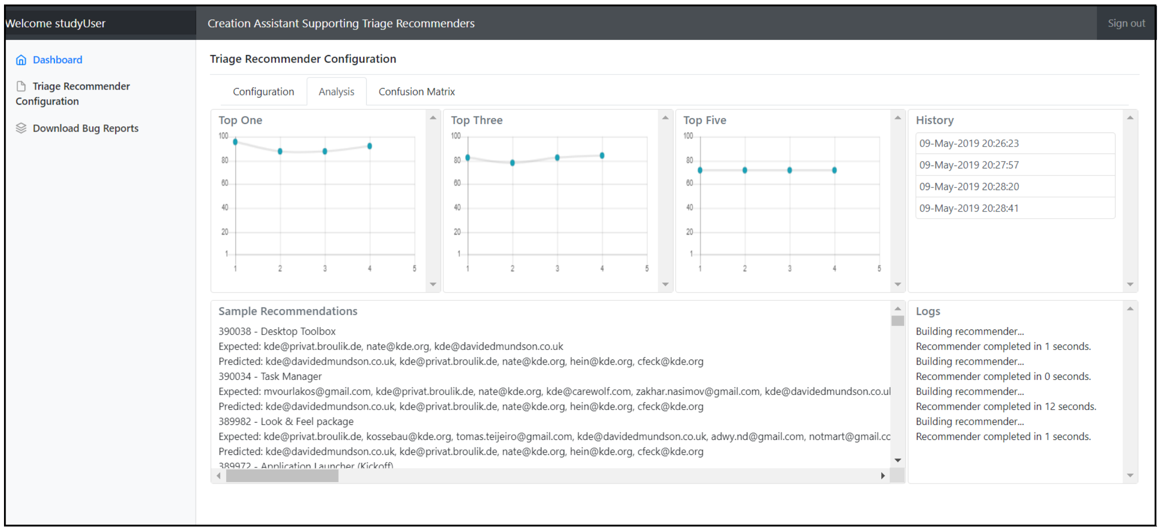
$$Precision = \frac{\#\, of\ correct\ recommendations}{\#\, of\ recommendations\ made} \tag{4.1}$$

$$Recall = \frac{\#\, of\ correct\ recommendations}{\#\, of\ possibly\ relevant\ recommendations} \tag{4.2}$$

For recommender creation, we are interested in a recommender that has high precision as we would prefer the recommender to produce a small list of developers with the right expertise as opposed to a recommender that produces a list containing all developers with only some having the right expertise. From the Table 4.5 we see that the SVM algorithm

Table 4.5: Precision and Recall of a recommenders when using different machine learning algorithm.

Top Predictions	**SVM**						**Naive Bayes**						**C4.5**						**Rules**					
	Plasmashell (P/R)		LibreOffice (P/R)		Firefox (P/R)		Plasmashell (P/R)		LibreOffice (P/R)		Firefox (P/R)		Plasmashell (P/R)		LibreOffice (P/R)		Firefox (P/R)		Plasmashell (P/R)		LibreOffice (P/R)		Firefox (P/R)	
1	96	11	95	2	51	11	89	10	96	2	55	11	89	10	81	2	38	10	92	10	97	2	46	11
2	90	19	94	5	40	14	82	17	94	5	37	12	89	19	82	3	30	11	89	19	94	5	32	12
3	83	26	93	7	34	15	78	25	93	7	35	15	83	26	78	5	34	13	85	27	94	7	35	14
4	79	33	92	9	29	15	76	31	92	9	30	16	76	32	80	7	41	17	71	30	92	9	27	14
5	73	38	91	11	36	19	73	38	90	11	36	19	73	38	78	8	36	19	73	38	91	11	36	19



09-May-2019 20:26:23
09-May-2019 20:27:57
09-May-2019 20:28:20
09-May-2019 20:28:41

Sample Recommendations

```
390038 - Desktop Toolbox
Expected: kde@privat.broulik.de, nate@kde.org, kde@davidedmundson.co.uk
Predicted: kde@davidedmundson.co.uk, kde@privat.broulik.de, nate@kde.org, hein@kde.org, cfeck@kde.org
390034 - Task Manager
Expected: mvourlakos@gmail.com, kde@privat.broulik.de, nate@kde.org, kde@carewolf.com, zakhar.nasimov@gmail.com, kde@davidedmundson.co.ul
Predicted: kde@davidedmundson.co.uk, kde@privat.broulik.de, nate@kde.org, hein@kde.org, cfeck@kde.org
389982 - Look & Feel package
Expected: kde@privat.broulik.de, kossebau@kde.org, tomas.teijeiro@gmail.com, kde@davidedmundson.co.uk, adwy.nd@gmail.com, notmart@gmail.cc
Predicted: kde@davidedmundson.co.uk, kde@privat.broulik.de, nate@kde.org, hein@kde.org, cfeck@kde.org
389972 - Application Launcher (Kickoff)
```

Logs

```
Building recommender...
Recommender completed in 1 seconds.
Building recommender...
Recommender completed in 0 seconds.
Building recommender...
Recommender completed in 12 seconds.
Building recommender...
Recommender completed in 1 seconds.
```

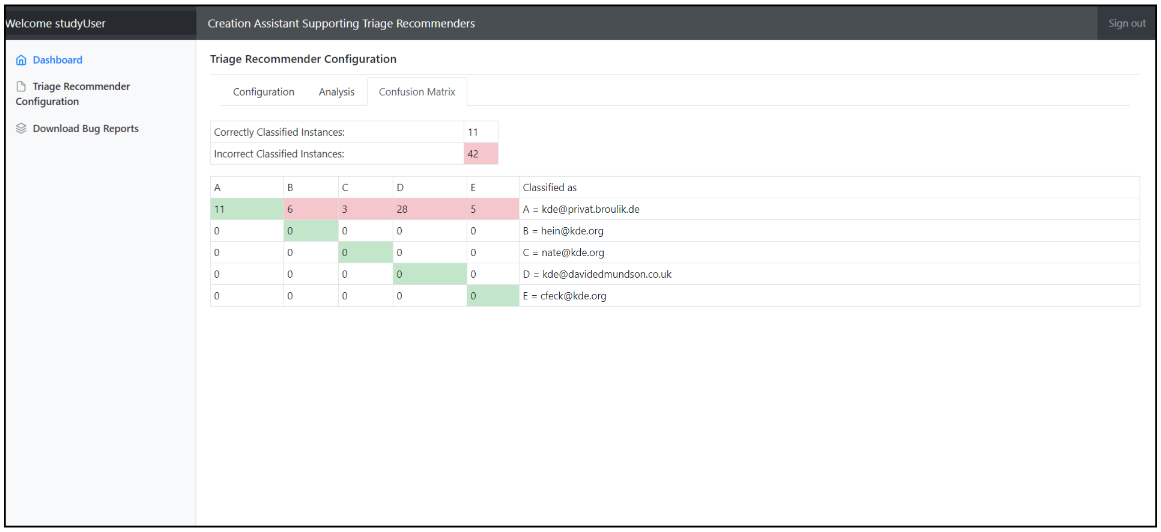
Figure 4.5: CASTR analysis tab.

produced recommender that had the highest precision when making one recommendation for Plasmashell. The Rules algorithm produced a recommender that had the highest precision when making one recommendation for LibreOffice and Firefox. However, when making three or more recommendations, the SVM algorithm generally provides a higher precision. We therefore chose SVM as the algorithm for creating a recommender.

Analysis Tab

Once the user starts the recommender creation process, the user is moved to an Analysis tab that presents progress information, such as the time taken to train the recommender and evaluation results. The user can then return to the Configuration tab, adjust the values for label and instance filtering, and create a new recommender. This process 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.

The Analysis tab displays the precision for the top 1, 3 and 5 recommendations for a testing set. Figure 4.5 shows a chart of the precision of recommender using Plasmashell project specific heuristics (see Table 4.4) and four different algorithms: SVM, Naive Bayes,



Triage Recommender Configuration

Correctly Classified Instances:	11
Incorrect Classified Instances:	42

A	B	C	D	E	Classified as
11	6	3	28	5	A = kde@privat.broulik.de
0	0	0	0	0	B = hein@kde.org
0	0	0	0	0	C = nate@kde.org
0	0	0	0	0	D = kde@davidedmundson.co.uk
0	0	0	0	0	E = cfeck@kde.org

Figure 4.6: CASTR confusion matrix tab.

C4.5 and Rules. The Analysis tab also displays an example of possible labels and predictions for randomly selected bug reports from the testing set (see Figure 4.5 Sample Recommendations). The detailed information on how the testing set is created is given in Section 4.3. By selecting an item from the history grid (see Figure 4.5 History), user is moved to Configuration tab that displays configuration which were used in past. The logs section displays the processing time taken by CASTR to create the assignment recommenders.

Confusion Matrix Tab

The confusion matrix tab displays the performance statistics as a classification confusion matrix. For each class value, it shows the distribution of predicted class values. A confusion matrix is used to describe the performance of a classifier using a testing dataset. It will display the correct/incorrect classified instances from the testing dataset which is used in the evaluation. For example, Figure 4.6 showing kde@davidmundson.co.uk was predicted 28 times when the correct label was kde@private.broulik.de.

4.3 CASTR: Web Service

All requests made by the interface will be processed by the application layer using RESTful web services. In the application layer, we use the Java API of Weka [19], a mature machine learning environment that is successfully used across several domains. When the user clicks on the “Recommender” button given in the Configuration tab (see Figure 4.4), first we perform data filteration in three stages as follows:

1. First we remove bug reports labeled as “Do Not Use” from the selected dataset. After this step we split the original dataset into a training set and a testing set.
2. We set the first 90% of the bug reports chronologically from the dataset as the training set and remaining 10% of the most recent bug reports as the testing set.
3. We eliminate bug reports from the training set resolved by developers that do not meet the minimum threshold criteria.
4. We perform text filtering by selecting only nouns and removing of stop words from the bug report summary and description field.

Once data filtration is done, we train the classifier using the Weka classifiers given in Table 4.6 on the training dataset for the selected machine learning algorithm.

4.3.1 An approach to handle data imbalance

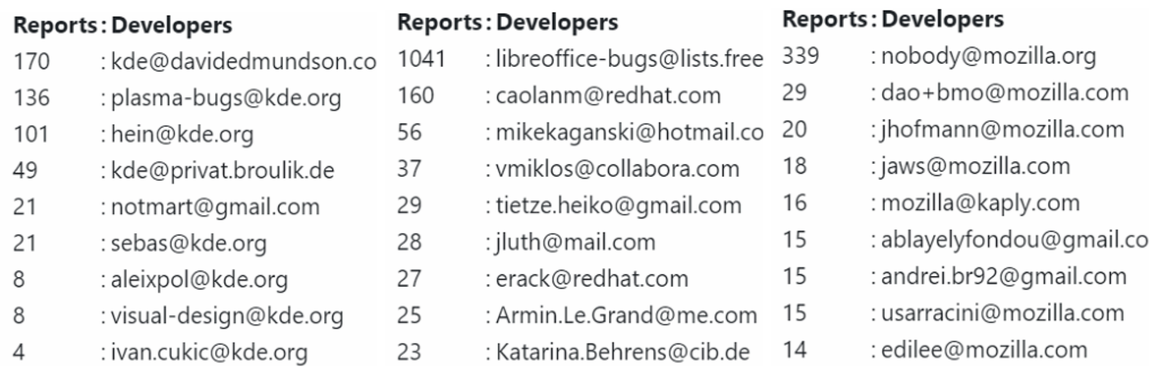
A dataset is imbalanced if the class distribution is not uniform among the classes. An imbalanced dataset will bias the prediction model towards the more common class (see

Table 4.6: Weka classes to train the classifier.

Machine Learning Algorithm	**Weka Classifier**	**Description**
SVM	SMO	Implements sequential minimal optimization algorithm for training a support vector classifier.
Naive Bayes	NaiveBayesMultinomial	Class for building and using a multinomial Naive Bayes classifier.
C4.5	J48	Class for generating a pruned or unpruned C4.5 decision tree.
Rules	ConjunctiveRule	This class implements a single conjunctive rule learner that can predict for numeric and nominal class labels.

Figure 4.6). Oversampling and undersampling are techniques used to adjust the class distribution of a data set. An example of imbalanced data is shown in Figure 4.7 in which the first three developers for Plasmashell have at least twice as many resolved reports as the other six developers. Figure 4.7 shows huge difference between the number of reports resolved by first developer and remaining developers for LibreOffice and Firefox project. One of the potential reasons behind data imbalance can be assigning the new incoming bug reports to a default user of the open source project. Mostly all of the software project's issue tracking system are assigning the new incoming bug report to a default user and then the bug triager will assign the incoming bug report to the correct developer. To handle imbalanced data, CASTR provides three different approaches: oversampling using SMOTE, manual oversampling and undersampling using clusters.

First we experimented with the most common method *Synthetic Minority Over-sampling Technique* (SMOTE) introduced by Chawla *et* al. [13] to resample a dataset. SMOTE is an over-sampling approach in which the minority class is over-sampled by creating "synthetic" examples rather than by over-sampling with replacement. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any/all of the *k* minority class's nearest neighbors. Depending upon the amount of over-sampling required, neighbors from the *k* nearest neighbors are randomly chosen. The main disadvantage with oversampling from our perspective, is



Reports	Developers
170	: kde@davidedmundson.co
136	: plasma-bugs@kde.org
101	: hein@kde.org
49	: kde@privat.broulik.de
21	: notmart@gmail.com
21	: sebas@kde.org
8	: aleixpol@kde.org
8	: visual-design@kde.org
4	: ivan.cukic@kde.org

(a) Plasmashell

Reports	Developers
1041	: libreoffice-bugs@lists.free
160	: caolanm@redhat.com
56	: mikekaganski@hotmail.co
37	: vmiklos@collabora.com
29	: tietze.heiko@gmail.com
28	: jluth@mail.com
27	: erack@redhat.com
25	: Armin.Le.Grand@me.com
23	: Katarina.Behrens@cib.de

(b) LibreOffice

Reports	Developers
339	: nobody@mozilla.org
29	: dao+bmo@mozilla.com
20	: jhofmann@mozilla.com
18	: jaws@mozilla.com
16	: mozilla@kaply.com
15	: ablayelyfondou@gmail.co
15	: andrei.br92@gmail.com
15	: usarracini@mozilla.com
14	: edilee@mozilla.com

(c) Firefox

Figure 4.7: Example of imbalanced data.

that by making exact copies of existing bug reports, overfitting is more likely. A second disadvantage of oversampling is that it increases the number of bug reports in the training set, thus increasing the training time.

Second, we implemented our own approach of sequential over-sampling by adding existing bug reports for each minority class uniformly unlike SMOTE which selects *k* nearest neighbors randomly from the minority class. We considered all of the classes as a minority class except the class name which has highest number of bug reports in the dataset. We took the difference between the majority class's number of bug reports and minority class's number of bug reports and then we increased the training dataset by adding the difference number of bug reports to the minority class sequentially. Table 4.7 shows an example of applying the sequential over-sampling technique. The algorithm for oversampling is:

1. Create a map *devInstances* to store the class (developer name) and list of bug reports resolved by that developer.

2. Find the majority class *topDev* that has resolved the highest number of the bug reports from devInstances.

3. Iterate through each class value except majority class.

4. Find the number of bug reports to be added by calculating the difference between number of bug reports resolved by *topDev* and the current developer.

5. Iterate through the bug reports resolved by the current developer until the number of bug reports to be added is reached.

6. Copy the bug reports and add it to the list of instances for the classification.

Using this approach, we get the class distribution uniform for our dataset. The major disadvantage of this method is that it increases the number of bug reports in the training set by roughly three times (see Table 4.8) which leads to decreases in the performance of recommender creation process.

Table 4.7: Example of instances created when sequential over-sampling is applied on the dataset.

Class Attribute	**# of Bug Reports Resolved**	**# of Bug Reports Added in Sampling**	**# Bug Reports for Classification**
kde@davidedmundson.co	170	0	170
plasma-bugs@kde.org	136	34	170
hein@kde.org	101	69	170
kde@privat.broulik.de	49	121	170

After experimenting with the two over-sampling methods, we implemented a method for under-sampling a dataset using the *Expectation Maximization* (EM) algorithm by removing some of the bug reports in the majority class. EM assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters. In this technique, first we split the training dataset into two groups: the top one majority class with the highest number of bug reports and the remaining classes in the other group. Then we take the difference between the majority class's number of bug reports and the other group's majority class to create the number of clusters. We create clusters from the dataset of top one majority class. From each cluster, we take only a specific number of bug reports for classification. The algorithm for under-sampling is:

1. Split the training set into a *topDev* dataset and a *otherDev* dataset. Store all the bug reports resolved by majority class into *topDev* dataset and remaining all the bug reports to *otherDev* dataset.

2. Find the number of clusters by calculating the difference between bug reports resolved by *topDev* and the second top developer (majority class from *otherDev* dataset).

3. Creates the clusters from *topDev* dataset.

4. Find the number of instances to be fetched from each clusters using the number of bug reports resolved by the second top developer *secondDev/No. of Clusters*.

5. Iterate through each clusters and add the required instances to *otherDev* dataset.

6. Use *otherDev* as the dataset for training the recommender.

The main disadvantage with under-sampling is it may remove some of the majority class instances which are more representative. In other words, it may leads to discarding useful information.

Table 4.8 shows the number of instances used for training the recommender using each technique after applying the filtering given in Section 4.3. The SMOTE and manual over-sampling techniques increased the instances by three times and the clustering technique reduced the number of instances.

4.4 Summary

This chapter presented an approach to assist the project members with the creation of assignment recommenders.

First, we presented information to guide a project member through the user interface towards the recommender creation questions, such as how to extract bug report reports, how to label the bug reports, and what heuristics configurations they should use. CASTR allows a user to download bug reports for any project from the Bugzilla issue tracking system. We showed how CASTR assists a project member with project-specific heuristics configurations for labeling the reports. We also presented precision and recall values of the assignment recommenders created using all of the four types of algorithms and how to utilize the information provided in the Analysis tab and the Confusion Matrix tab.

Second, we explained the role of the CASTR web service and an approach to handle data imbalance with different techniques and their impact on the dataset.

Table 4.8: Example of instances used for training the recommender.

Projects	**SMOTE**	**Manual Oversampling**	**Clustering**	**None**
Plasmashell	829	833	274	354
LibreOffice	6235	6264	1220	1265
Firefox	1231	1248	449	472

Chapter 5

Evaluation

This chapter presents an analytical method for evaluation and a user study of the CASTR tool. Our hypothesis is that the use of CASTR will help reduce the complexity and time needed for creating a bug report assignment recommender. To investigate this hypothesis, we formulate three research questions.

RQ1: **Does CASTR create assignment recommenders that make accurate recommendations?** If CASTR creates assignment recommenders that make accurate recommendations, then a triager need not to examine the report as deeply as they would without the recommendations. Using such recommenders change the triager's role from making decisions relying on their own knowledge, experience, and intuition or that which they can gain from existing tools, to confirming decisions made by the recommender. This shift changes, and hopefully, reduces the cognitive load on the triager.

RQ2: **Do bug report assignment recommenders created by CASTR perform similar to the recommenders tuned by hand?** If CASTR creates assignment recommenders that provide similar precision than the recommenders tuned by hand (manual process to create assignment recommemders), then a project member can save the time in terms of efforts made by human resources to tune the recommenders manually.

RQ3: **Can human triagers make effective use of information recommended by CASTR?** If CASTR creates assignment recommenders that assist human triagers then human triager can save time while assigning bug reports to appropriate developer manually. Some of the human resources consumed by the triage process can be then directed elsewhere in the project.

To investigate RQ1 and RQ2, we conducted an analytical evaluation of an assignment recommender using our approach for three open source projects (Section 4.2.1). We investigated RQ3 through a field study, in which ten users (human triagers, software developers, software testers and project manager) used CASTR.

5.1 An Analytic Evaluation of the Recommenders

To validate our hypothesis that CASTR can produce accurate assignment recommenders, we analytically evaluated CASTR through laboratory experiments using data from three open source projects: Plasmashell, LibreOffice and Firefox. These projects were chosen as they vary in code size, small to large numbers of contributors, and good bug submission frequencies. We evaluated assignment recommenders created by CASTR to determine that assignment recommender can be created with a high enough accuracy to benefit the project. This section begins with a description of our evaluation methodology. The remainder of this section presents the results of our analytic evaluation of the created assignment recommenders.

5.1.1 Analytic Evaluation Procedure

First we evaluated an assignment recommender created by hand. We performed an evaluation using the bug reports with the ids given in Table 5.1 from the Plasmashell project. Initially, we followed the similar process given in Figure 2.2. After identifying the valid bug reports we have to determine the potential developer who can fix the bug report. In order to find the potential developer, first we performed data filtering by removing stop words, before searching for similar bugs (if reported) using the bug report summary. For each bug report we found more than 500 similar bug reports. It was impossible to find the

Table 5.1: Bug reports used in evaluation of assignment recommender created by hand.

Bug Report Id	**Component**	**Bug Report Summary**
390038	Desktop Toolbox	Desktop toolbox should be in top-right corner to keep it from intruding on folder view icons
390034	Task Manager	Switching applications via mouse may result in a drag and drop of a desktop file into the switched application
389815	Device Notifier	Possibility of code execution when opening volume which label contains “ or $() from notifications panel

developer who has resolved nearly similar bug reports from such an extremely large set of bug reports. In the end, we obtained the list of developers who have resolved bug reports within the same component as the list of potential developers that could fix the bug. Again, we failed to identify the developer as there are many developers who contribute to the Plasmashell project. The process of creating assignment recommender by hand, certainly requires project knowledge to recommend the most appropriate developer who can fix the bug report.

To evaluate assignment recommenders created by CASTR, we trained the recommender using 90% of the bug reports from the dataset and remaining 10% of the bug reports we set in the testing set. We use precision and recall to measure the performance of the assignment recommenders created using CASTR. Precision measures how often the approach makes a relevant recommendation for a report (Equation 4.1). Recall measures how many of the recommendations that are relevant are actually recommended (Equation 4.2).

The key piece of information in computing both precision and recall is the set of appropriate recommendations. Table 5.2 shows the number of bug reports used while evaluating CASTR for three different projects. The column “After filtering dataset size” shows the number of bug reports remaining in the dataset after removing bug reports labeled as *DoNotUse*. If we apply other filters such as setting the minimum threshold value then those bug reports will be eliminated from the training set.

As we determined the set of valid labels from the bug report dataset (see Table 4.3), we added a *Do Not Use* label to not include a set of bug reports for the classication. Mainly

Table 5.2: Training and testing set sizes for evaluating recommenders.

Projects	Original Dataset Size	After Filtering Dataset Size	# of Bug Reports in Training Set	# of Bug Reports in Testing Set
Plasmashell	1112	532	479	53
LibreOffice	2500	1725	1553	172
Firefox	1000	777	699	78

we aimed to use *Do Not Use* label for the set of bug reports which are marked as *Duplicate* because, we should have the developer's name who has resolved the bug report which is a duplicate of the bug reports and CASTR creates an assignment recommender on a pre-downloaded dataset. CASTR does not access the bug reports at run time. Initially, we considered downloading the duplicate of the bug reports at the time of downloading the actual dataset but it increased the size of the dataset drastically. As given in Table 4.2, the dataset of Plasmashell project contains 52% of bug reports with the resolution group *Duplicate*, while LibreOffice and Firefox project contains 31% and 22% of this resolution group respectively. We can also set the label *Do Not Use* for the set of bug reports which are in resolution groups: Later, Moved and Unknown. It is very unlikely for CASTR to recommend possible developer to fix the bug report from the resolution group which has lower than 1% of the bug reports.

5.1.2 Assignment Recommender Evaluation

Once the recommender is created, we evaluate the performance of the recommender using precision, recall and a confusion matrix. Before we evaluate the recommender, we prepare the set of developers for each different component to find possibly relevant recommendations. We need to know for each report in the test set which developers on the project might be a possible recommendation to resolve the report. Table 5.3 shows an example of sample recommendations created by CASTR using the SVM algorithm and project specific heutistics given in Table 4.4.

Figure A.3 presents the recommender evaluation results for each type of algorithm using

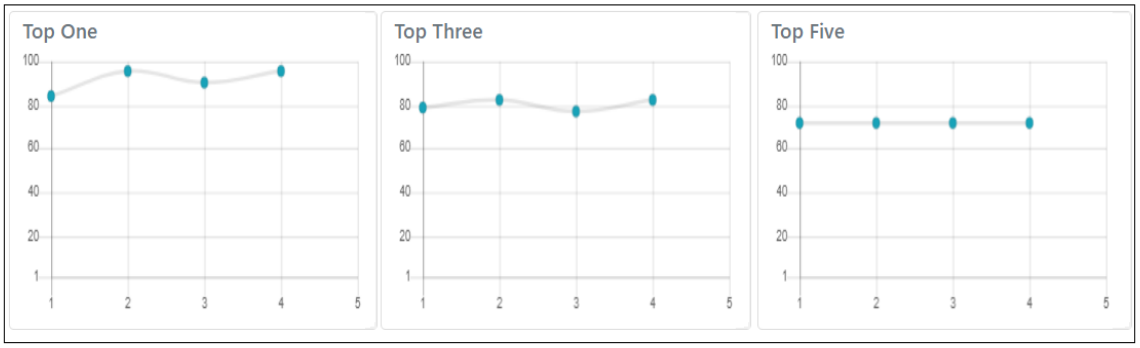
Table 5.3: Example of sample recommendations generated by CASTR.

Bug Report Id	Component	Expected	Predicted
390038	Desktop Toolbox	kde@privat.broulik.de nate@kde.org kde@davidedmundson.co.uk	kde@davidedmundson.co.uk kde@privat.broulik.de hein@kde.org nate@kde.org cfeck@kde.org
390034	Task Manager	mvourlakos@gmail.com kde@privat.broulik.de nate@kde.org kde@carewolf.com zakhar.nasimov@gmail.com kde@davidedmundson.co.uk bugseforuns@gmx.com chadjoan@gmail.com cfeck@kde.org omarplummer@imergetechnologies.com	kde@davidedmundson.co.uk kde@privat.broulik.de hein@kde.org nate@kde.org cfeck@kde.org
389815	Device Notifier	kde@davidedmundson.co.uk kde@privat.broulik.de hein@kde.org nate@kde.org cfeck@kde.org	kde@privat.broulik.de nate@kde.org kde@davidedmundson.co.uk hein@kde.org cfeck@kde.org

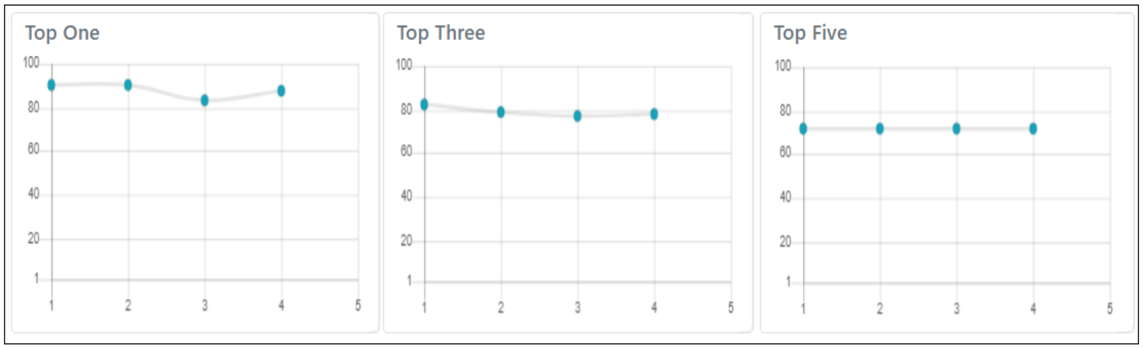
the Plasmashell project dataset. To create the assignment recommender we used project-specific heuristics given in Table 4.4 with a minimum threshold as 20 and four different type of sampling techniques, one at a time. Each chart shows 4 data points which represents the precision values of the recommender created using three different sampling techniques : SMOTE, Manual Oversampling, Clustering and the last point presents the precision value of recommender created without applying any sampling techniques (See Figure 4.4 option *None*).

Figure A.3 shows the assignment recommenders created using the algorithm SVM, Naive Bayes, C4.5 and Rules with data imbalance technique (SMOTE, Manual Oversampling, Clustering and None). Note that option *None* provides a higher precision than SMOTE and the clustering technique. For the assignment recommenders created using the Rules algorithm, they perform better with an option *None* than clustering and manual oversampling and SMOTE performs almost similar.

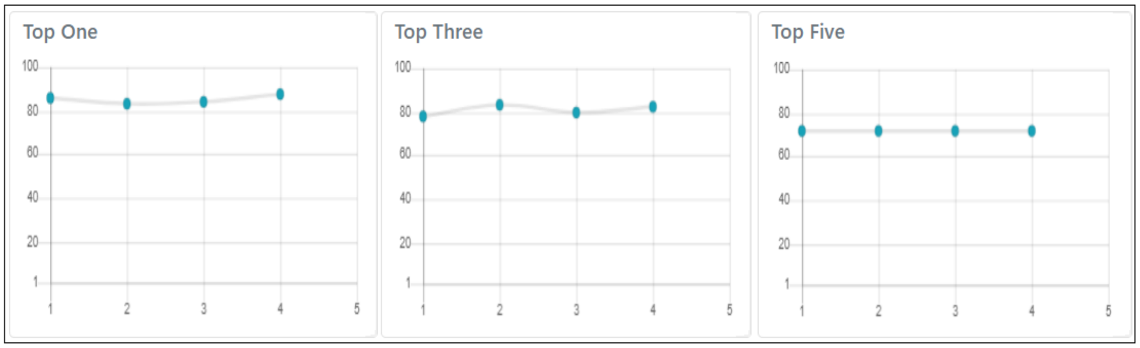
Furthermore, we also evaluated the assignment recommenders using a confusion matrix



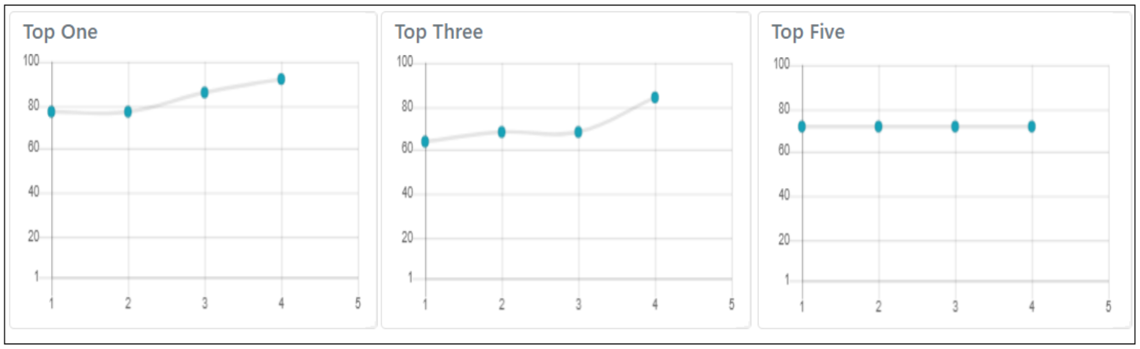
(a) SVM



(b) Naive Bayes



(c) C4.5



(d) Rules

Figure 5.1: Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Plasmashell dataset.
Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None

Table 5.4: Recommenders evaluation result from Confusion Matrix.

Technique	SVM		Naive Bayes		C4.5		Rules	
	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect	Correct	Incorrect
SMOTE	14	39	9	44	0	53	0	53
Manual Oversampling	13	40	9	44	10	43	0	53
Clustering	28	25	22	31	22	31	47	6
None	11	42	9	44	7	46	0	53

method. Table 5.4 presents the evauation results using confusion matrix on the Plasmashell testing dataset. The Correct and Incorrect columns show the number of testing bug reports predicted correctly and incorrectly.

Table 5.4 presents the opposite results of precision and recall. One of the possible reason behind the variation in results may be because we prepared the set of developers for each different component to find possibly relevant recommendations. We consider component-wise the developer list in the precision and recall calculation which results in getting higher precision for the assignment recommender. Appendix A shows detailed results from recommender evaluation for all three projects.

5.2 A Field Study of CASTR

Given the analytical evaluation results we decided to conduct the field study with the data sampling fixed at *None* as the other three approaches (SMOTE, Manual Oversampling and Clustering) were not making a progressive impact on the results. Most often the results were similar or with lower precision. Inaddition, all three approaches were resizing the dataset so we chose not to use any sampling techniques.

To gain insight into whether a human can make effective use of the information presented by CASTR, we conducted a field study with experienced software developers, project managers, bug triagers and graduate students. The study contained ten (10) participants: 3 project managers, 5 application developers and 2 computer science graduate students. The following sections presents the field study setup, prior knowledge and experience of partic-

Table 5.5: Questionnaire used to get background information of participants.

Demographic Questions	
1.	In which of the following age ranges do you fall? [18-25, 26-39, 40-59, 60 or older, Prefer Not to Answer]
2.	To which gender identity do you most identify? [Female, Male, Transgender Female, Transgender Male, Gender Variant/Non-Conforming, Not Listed, Prefer Not to Answer]
3.	What is your nationality?
4.	What is the highest degree or level of school you have completed? [Elementary school, High School, Trade/technical/vocational training, Associate degree (2-year), Bachelors degree (4-year), Graduate (Masters or Doctorate), Not Listed]
5.	What is your job function? [Application Developer, QA/Testing, Program Director, System Integrator, Application Architect, Project Manager, Business Analyst, Database Administrator, Student, Other]
Technical Background	
6.	How many years of experience do you have with programming? [0-3, 4-6, 7-10, 11-15, $>$15]
7.	What level of experience do you have with triaging bug reports? [Beginner, Developing, Competent, Advanced, Expert]
8.	How frequently do you log a bug in the Issue tracking system? [Never, Rarely, Occasionally, Often, Always]
9.	What level of experience do you have with using machine learning algorithms? [Beginner, Developing, Competent, Advanced, Expert]

ipants and results of the participants using CASTR to create assignment recommenders for a large open source project.

5.2.1 Field Study Procedures

The field study was conducted in the following manner. First we asked participants to complete an initial survey that collected demographic information and technical background details. Table 5.5 shows the list of questions asked in the initial survey. The questions for demographic information are only for general analysis to breakdown the overall survey response data into meaningful groups of respondents. For example, we found that Indian participants took the survey more than the participants with other nationalities. Participants with age between 26 to 39 were more interested in the field study. Most of the participants has completed minimum Graduate level schooling, and most of them belong to job function Application Developer.

Table 5.6: Questionnaire used for the post usage interview.

1.	Did you find CASTR easy to use? [Extremely, Very, Moderately, Slightly, Not at all]
2.	How likely are you to recommend CASTR for creating a recommender for bug report assignment? [Extremely, Very, Moderately, Slightly, Not at all]
3.	How likely do you think assignment recommenders created using CASTR will reduce the time it takes to triage bug reports? [Extremely, Very, Moderately, Slightly, Not at all]
4.	Do you have any suggestions for improvements to the CASTR user interface?
5.	Do you have any suggestions for improvements in the workflow of recommender creation?
6.	Do you want to receive a notification of the study result?
7.	Do you want to be contacted by the researcher(s) for follow-up questions?
8.	Do you want to try your own dataset for triaging?

For technical background, participants were asked about their level of experience with triaging bug reports, how frequently they use an issue tracking systems, familiarity with machine learning algorithms and years of experience with programming. After completing the initial survey, we asked participants to create an assignment recommender using the different settings provided by CASTR. We provided each participant a unique study id to access CASTR, a dataset of the Plasmashell project (see Table 4.1) and an user manual of CASTR.

We asked participants to submit tool usage feedback after they completed the creation of an assignment recommender. The participants were asked to provide suggestions for any improvements to the CASTR user interface or workflow of recommender creation or any other comments about their experience with CASTR. Table 5.6 shows the list of questions asked about the performance of CASTR. Appendix B shows the answers of questions asked in the initial survey and post tool usage interview.

Table 5.7: Developer wise bug report assignment from Plasmashell project.

# of Bug Reports Assigned	After Removing Duplicate Bug Reports	Developer Id
496	170	kde@davidedmundson.co.uk
263	136	plasma-bugs@kde.org
178	101	hein@kde.org
72	49	kde@privat.broulik.de
33	21	sebas@kde.org
28	21	notmart@gmail.com
12	8	aleixpol@kde.org
9	8	visual-design@kde.org
5	4	ivan.cukic@kde.org
5	4	kossebau@kde.org
2	1	kwin-bugs-null@kde.org
2	2	vladzzag@gmail.com
1	1	alex19930329@gmail.com
1	1	amantia@kde.org
1	1	arsenarsentmc@outlook.com
1	1	bhush94@gmail.com
1	1	faure@kde.org
1	1	mvourlakos@gmail.com
1	1	scott@spharvey.me

5.2.2 Dataset Information

The field study was conducted using the Plasmashell dataset similar to which we used for our analytical experiment. The Plasmashell dataset contained 1112 bug reports (see Table 4.1). The resolution-wise dataset distribution is given in Table 4.2. Table 5.7 presents developer-wise the number of bug report assignments. The first column shows the number of bug reports assigned to a particular developer initially. The second column shows the number of bug reports assigned to a developer after removing bug reports marked as Duplicate resolution. The last column identifies unique developers who are assigned to the bug report.

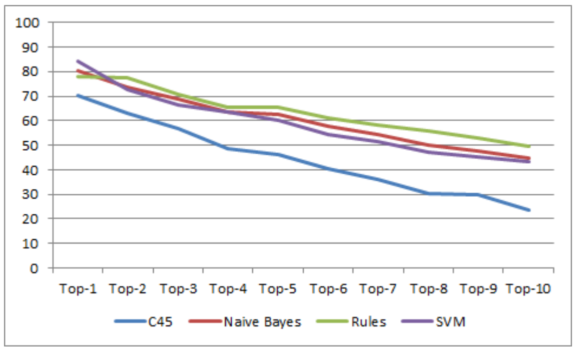
5.2.3 Prior Knowledge and Experience

Most of the participants had completed a minimum Bachelor-level degree with five participants having completed a Masters degree. From all the 10 participants, 60% had prior experience with issue tracking systems whereas 40% of the participants used issue tracking system occasionally or rarely. Overall bug triaging experience was a lower, with four participants reported as beginner, three reported as developing, two reported as competent and one reported advanced. Possible reason for the less advanced level or expert level involvement in bug triaging is that most of the participants were part of a large software development project team where responsibilities are limited within the modules or feature development. Participants that reported the least familiarity with the machine learning algorithms included five which were beginner, four which were developing interest and one which was advanced level. Three participants had a good amount of experience with contributing to open source projects such as Mozilla and KDE. We measured "good amount of experience" based on years of the contribution to an open source project and having commit access on the source code repository.

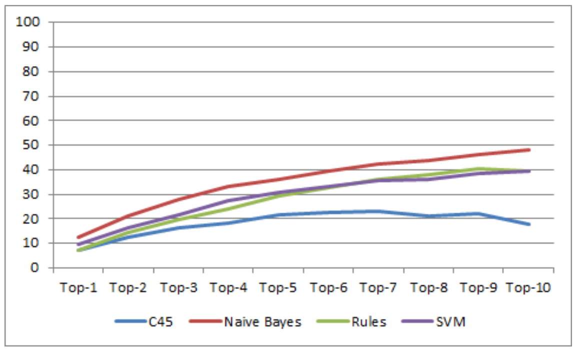
5.2.4 Quantitative Results

In the field study, a total of 71 recommenders were created by the participants using different heuristic configurations provided by CASTR. Figure 5.2 shows the average precision and recall for the top 10 recommendations created using the four different type of machine learning algorithms. Figure 5.2 also shows improving precision typically reduces recall. While precision refers to the percentage of recommendations which are correct, recall refers to the percentage of total relevant recommendations classified correctly by an algorithm.

Figure 5.2(a) shows that assignment recommenders created using the SVM algorithm produced a higher precision with 86%, Naive Bayes with 80%, Rules with 78% and C4.5 with 70%. Figure 5.2(b) shows that assignment recommenders created using Naive Bayes



(a) Precision



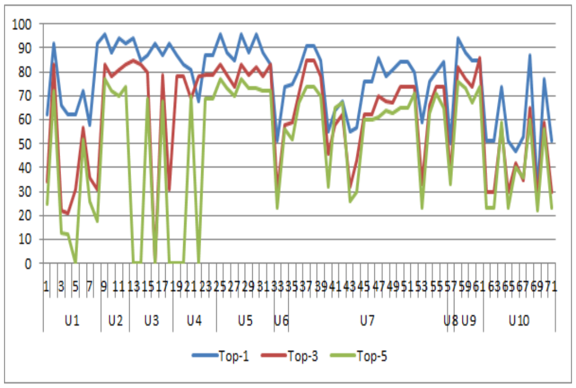
(b) Recall

Figure 5.2: Average precision and recall for top 10 recommendations.

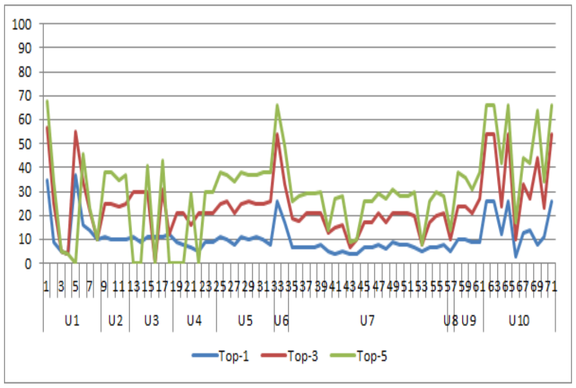
and SVM machine learning algorithm increased recall gradually with an increasing number of top recommendations whereas assignment recommenders created using C4.5 and Rules algorithm decreased in recall for the last 3 recommendations (Top-8, Top-9 and Top-10) recommendations respectively.

Figure 5.3 shows the precision and recall with trend lines for top 1, 3 and 5 recommendations for each assignment recommender created by all ten participants. The chart shows that participants were able to get acceptable precision between 50% to 95% throughout for Top-1 recommendation. Precision for Top-2 recommendations was between 20% to 80% and precision for Top-5 recommendations was a maximum of 70% and a minimum of 10%. The chart shows 0% precision when there was no recommendations for Top-5 due to the selection of developer's profile such as less than 5 developers left to consider for the recommendation. The charts for the top 1, 3 and 5 recommendations with precision and recall for all the assignment recommender created by each participants is given in Appendix C.2

Table 5.8 shows the quantitative results from the ten participants. The first column identifies the unique participant. The second column presents the machine learning alogrithm selected by the participants for creating their best assignment recommender. The next two columns present the minimum and maximum threshold that the participants selected before creating their most accurate assignment recommender using CASTR. The minimum threshold value is set to remove bug reports from labeling for developers that fall under the



(a) Precision



(b) Recall

Figure 5.3: Top 1, 3 and 5 precision and recall from all the recommendations created.

lower threshold. Half of the participants chose values greater than or equal to 10 for the minimum threshold and the remaining participants used values less than 10. CASTR will consider the maximum threshold as the maximum number of bug reports labeled with an individual developer. The value of the maximum threshold will be calculated based on the selected heuristics. The next three columns show the Top-1, Top-3 and Top-5 recommendations with precision and recall values for the best assignment recommender created by the participants. Most of the assignment recommenders were created using the SVM machine learning algorithm to get a higher precision.

Table 5.9 shows the heuristic configurations used for the creation of the best assignment recommender (see Table 5.8). The first column identifies the unique participant. Next column presents the number of heuristics to create assignment recommender. Based on

Table 5.8: Best assignment recommenders created by participants.

Identifier	Algorithm	Trials to Best	Threshold		Top 1 (%)		Top 3 (%)		Top 5 (%)	
			Min	Max	Precision	Recall	Precision	Recall	Precision	Recall
User1	SVM	2	5	108	92	9	83	25	72	34
User2	SVM	1	10	130	96	11	83	25	77	38
User3	SVM	1	26	112	94	11	85	30	-	-
User4	Naive Bayes	5	49	73	87	9	78	21	-	-
User5	SVM	1	10	130	96	11	83	25	77	38
User6	SVM	2	1	141	74	17	58	33	56	49
User7	SVM	4	1	61	91	7	85	21	74	29
User8	Naive Bayes	1	1	117	50	5	37	10	33	14
User9	SVM	1	10	115	94	10	82	24	76	38
User10	SVM	7	1	102	87	14	65	27	60	42

the number of heuristics, CASTR decides that how many heuristics need to be used while creating the recommender. If a participant select a smaller number than the maximum available resolution groups then CASTR uses the value of “Other heuristics” set in column three for labeling all the remaining bug reports (see Section 4.2.2). The remaining nine columns present the heuristic selected by participants for each resolution group to label the bug reports. Table 5.9 shows that participants did not consider “Other Heuristics” for labeling the bug reports.

Table 5.9: Project heuristics used for creation of best assignment recommenders.

Identifier	# of Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
User1	9	AssignedTo	Do Not Use	FirstResponder	Resolver	FirstResponder	Reporter	Resolver	FixedBy	FixedBy	AssignedTo
User2	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
User3	9	AssignedTo	Do Not Use	FixedBy	Resolver	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
User4	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
User5	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
User6	3	AssignedTo	Do Not Use	Resolver	AssignedTo						Do Not Use
User7	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
User8	4	AssignedTo	Do Not Use	Reporter	AssignedTo	AssignedTo					Resolver
User9	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
User10	6	Attachment	Do Not Use	Resolver	Resolver	AssignedTo	AssignedTo	FirstResponder			

Table 5.10: Heuristics used for labeling bug reports by most of the participant.

Resolution Group	Label
Duplicate	DoNotUse
Fixed	FixedBy
Worksforme	Resolver
Invalid	Resolver
Unknown	Resolver
Upstream	Resolver
Wontfix	FirstResponder
Later	DoNotUse
Moved	DoNotUse

Table 5.10 shows the label selected for each resolution group by most of the participants. The heuristic configurations used for all the created recommenders is given in the Appendix C.1. As shown in Table 5.9, most participants chose to use all of the heuristics. In this scenario other heuristic label will be considered importent in the recommender creation process.

As a part of the recommender evaluation, CASTR provides information about how long the tool takes to create a recommender. Figure 5.4 presents the average time (in seconds) for recommender creation using different types of machine learning algorithms. Figure 5.4 shows that assignment recommenders created using the C4.5 and SVM algorithms took more processing time than the assignment recommender created using the Naive Bayes and Rules algorithms. One possible reason for the increased processing time is both the participant *User1* and *User7* have set the minimum threshold value as 1 which leads to an increase the processing time as CASTR will consider all the possible developers for classification.

5.2.5 Discussion of Quantitative Results and Observations

Based on observations while analyzing the field study result, participants were found to employ two strategies for assignment recommender creation using CASTR. Some participants were found to be very experimental in their approach, making many changes before

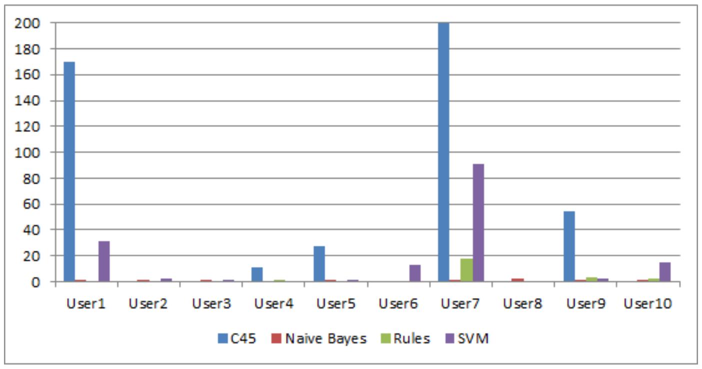


Figure 5.4: Average recommender creation time in seconds.

creating a new recommender. Other users were more methodical, making small changes and testing the results. Most of the time participants changed the heuristic configurations not the minimum threshold. Out of 71 recommenders, 42 were created with the minimum threshold value as 1, 27 recommenders were created with threshold more than or equal to 10 and the remaining 2 recommenders were created with thresholds of 3 and 5. The maximum number of recommenders were created using the SVM machine learning algorithm recorded as 32, using Naive Bayes recorded 19 times and 10 recommenders were created using each of C4.5 and Rules algorithm.

5.3 Qualitative Results

Although participants were provided with a video presentation of CASTR and a brief tutorial of the recommender creation process at the beginning of the field study, participants encountered a number of problems related to understanding concepts and specifically with labeling bug reports and setting minimum threshold value. Most of the participants were not clear on how to select the appropriate label for bug report resolution and which machine learning algorithm to use. Also, the meaning of the precision and recall metrics was not initially well understood by participants. However, once their meaning was understood, participants felt that they made more intelligent choices about the configuration.

As mentioned in Section 5.2.3, most of the participants did not have specific knowledge about the project, such as who formed the core group of developers. This led to some participants 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 CASTR, as they would have knowledge about the core development team. For example, *User3* has bug triaging experience with Mozilla project and therefore created assignment recommenders with a higher precision (See Appendix C.2).

The field study results shows that 60% of the participants found CASTR easy to use whereas the remaining participants found CASTR moderately and slightly easy to use. We received positive response about recommending CASTR for creating a recommender for bug report assignment with 50% of the participants responded as very and extremely and the remaining participants responded moderately. The important question we asked to the participants was whether they believed that the assignment recommenders created using CASTR would reduce the time to triage bug reports and the response we received is 2 participants found it extremely likely, 5 participants found it very likely, and 3 participants found it moderately likely.

5.4 Threats to Validity

We designed our experiment to minimize the threats to validity, but still a number of decisions that might influence our results had to be made. We discuss the main validity threats to our study with respect to internal validity, external validity, and construct validity of this work.

5.4.1 Internal Validity

There is a possibility of error in the creation of the data set used for the evaluation. Although, we examined a random number of bug reports to verify the data collection procedure was correct, there may have been some bug reports that contained incorrect data.

Currently in our approach, we use the bug report summary and description for the classification. While the experimental results show that these two unstructured text data are necessary, there could be other added information required to create a more accurate assignment recommender.

We use the default configurations provided by WEKA for all of the individual classifiers studied. While most classifiers are highly configurable, we did not set any external parameters. The default configurations for some classifiers might be favorable for bug report assignment and others might underperform.

During the training and testing of a classifier, we assumed only one developer as the rightful owner of a bug report. However, based on patterns and history of bug reports that are solved, there could be more than one active developer in the project who could potentially address the bug.

To address the data imbalance issue, CASTR enables three implementation technique and none of them worked well. It may be possible that other technique of oversampling or undersampling work better with recommender creation.

5.4.2 External Validity

External validity reflect the generalizability of our results. In this work, participants with no to little project-specific knowledge were used to evaluate the usability of CASTR. Therefore, these results would not generalize to those with project-specific knowledge, but could be considered as a lower-bound for such a group.

Another possible threat to the external validity of the result is that the participants automatically accepted the top recommendation without considering if the recommendation was appropriate. This blind acceptance of recommendations could have led to the high accuracy results.

The main threat to the external validity of the result is that field study was conducted using a dataset of bug report from single project. The results may not extend to other projects.

5.4.3 Construct Validity

Threats to construct validity refers to the suitability of the evaluation measures. We used a method to determine the set of developers that could have fixed a bug report and to calculate precision and recall that 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 CASTR show the relative differences between different configurations, so even if the precision and recall values are over or under their true value, CASTR still provides meaningful information to the user.

5.5 Summary

We evaluated CASTR to answer three research questions.

RQ1: **Does CASTR create assignment recommenders that make accurate recommendations?** The results of the field study provides evidence that the accuracy of the assignment recommenders is sufficiently high to be considered for assisting project members to create assignment recommender using CASTR. The ten participants contributed in the field study were positive about the value of the recommendations. Unfortunately, the small set of participants and the relatively small number of assignments recommender created by some of participants made it difficult to determine if CASTR creates assignment recommenders that make accurate recommendations.

RQ2: **Do bug report assignment recommenders created by CASTR perform similar to the recommenders tuned by hand?** The process of assignment recommender creation by hand is extremely time consuming. The person who is creating recommender must have project specific knowledge in order to identify the correct developer who can fix the bug report. The recommender created by CASTR works much better in terms of saving the time, predicted accurate developer and project knowledge is not necessary to create assignment recommender.

RQ3: **Can human triagers make effective use of information recommended by CASTR?**

The results given in Table 5.8 shows that most of the participants were able to create the assignment recommender with the best accuracy within 5 trials. Most participants did not have project knowledge. The field study result shows that human triagers can make effective use of the information recommended by CASTR.

Regarding improvements to the CASTR user interface, most of the participants suggested to improve user interface with graphs providing more details. A few participants found the CASTR interface difficult to understand and one participant suggested to make the CASTR interface more fancy with no scrollbars and more user friendly.

We also asked participants about any improvements in the workflow of recommender creation and response was neutral. One participant found that CASTR would be more useful if there was a wizard for creating a recommender and guidance on what machine learning approach to use. Other participants found that configuration page can be more descriptive and a wizard-based flow could have worked better than what CASTR offers. One participant also suggested improvements in the sample recommendations section with providing more detail and make each recommendation entry separated by lines or some space so that it will be easy to read and understand.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The main goal of bug triage is to evaluate, prioritize and assign the resolution of bug reports. A project member validates the bug reports according to the severities of defects. If any changes are required then a decision needs to be made about to whom the work will be assigned. For a given bug report, identifying an appropriate developer who could potentially fix the bug is the primary task of a bug triaging process. This process become overwhelming when the project receives a large number of bug reports everyday.

[2, 5, 17, 18, 30] proposed bug report assignment recommenders as a method for reducing the workload of a project member. Furthermore, the process of bug report assignment recommender is a complex process as project members have to perform many steps such as data filtration, labeling, selection of machine learning algorithm and create the recommender. After creation of bug report assignment recommender, a project member needs to determine if the recommender works well.

This dissertation presents an approach to assist a project member with the creation of assignment recommenders to streamline the development process. The work described in this dissertation makes following contributions to the field of software engineering.

First, we implemented a web based tool CASTR that allows a project member to analyze the dataset with graphical representation. Additionally, CASTR assists project members in configuring project-specific parameters when creating a recommender using four different types of machine learning algorithm. In addition, CASTR allows project members to compare the last five recommenders created using different configurations.

Second, we show, through analytical evaluations, that recommenders can be created with good accuracy using CASTR as compare to recommenders tuned by hand. The analytical evaluation was conducted using the bug reports dataset from three different open source projects and showed that recommenders with good accuracy could be created using four different type of machine learning approach.

Lastly, we conducted the empirical evaluation using a field study with ten participants from the different technical background and showed that the recommenders worked well in practice. The field study results reveal that human triagers can make effective use of information presented by CASTR.

6.2 Future Work

Although the results we obtained have shown that a CASTR assists the project members with the creation of assignment recommenders based on feedback and the results of the user study, a number of future improvements were identified for CASTR:

- Extending CASTR to collaborate with the other issue tracking systems (JIRA, Redmine etc.,) in addition to the Bugzilla repository.
- Additional work is also needed to create assignment recommenders using bug reports marked Duplicate resolution. In this thesis work, we do not include bug reports marked as Duplicate while creating recommendations.
- The CASTR user interface needs enhancement in the charts with more explanation, better visualization, more descriptive configuration, analysis and confusion matrix page.
- CASTR stores the created assignment recommenders on the server. However, the UI only shows the five most recently created recommenders.

- CASTR does not distinguish between complex and simple source code changes.
- Extending CASTR to support other types of triage recommenders.

References

[1] J. Anvik, M. Brooks, H. Burton, and J. Canada. Assisting software projects with bug report assignment recommender creation. *In Proceedings of the 26th International Conference on Software Engineering and Knowledge Engineering*, pages 470–473, 2014.

[2] J. Anvik and G. C. Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Trans. on SE and Methodology*, 2011.

[3] John Anvik, Lyndon Hiew, and Gail C. Murphy. Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370, New York, NY, USA, 2006. ACM.

[4] John Anvik and Gail C. Murphy. Determining implementation expertise from bug reports. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 2–, Washington, DC, USA, 2007. IEEE Computer Society.

[5] Pamela Bhattacharya and Iulian Neamtiu. Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging. In *Proceedings of the 2010 IEEE International Conference on Software Maintenance*, ICSM '10, pages 1–10, Washington, DC, USA, 2010. IEEE Computer Society.

[6] BigML. Bigml is machine learning for everyone. `http://bigml.com`, Aug 24, 2017.

[7] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[8] G. Bortis, Hoek, and van der A. Porchlight: A tag-based approach to bug triaging. *In Proceedings of the 2013 International Conference on Software Engineering*, page 342351, 2013.

[9] G. Bortis and van der A. Teambugs: A collaborative bug tracking tool. *In Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 69–71, 2011.

[10] Robin Burke. Knowledge-based recommender systems. In *ENCYCLOPEDIA OF LIBRARY AND INFORMATION SYSTEMS*, page 2000. Marcel Dekker, 2000.

[11] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, November 2002.

[12] Y Cavalcanti, P Neto, I Machad, E de Almeida, and S de Lemos Meira. Towards understanding software change request assignment: a survey with practitioners. *17th Inter. Conf. on Eval. and Assess. in SE*, 2013.

[13] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. Smote: Synthetic minority over-sampling technique. *J. Artif. Int. Res.*, 16(1):321–357, June 2002.

[14] Horatiu Dumitru, Marek Gibiec, Negar Hariri, Jane Cleland-Huang, Bamshad Mobasher, Carlos Castro-Herrera, and Mehdi Mirakhorli. On-demand feature recommendations derived from mining public product descriptions. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 181–190, New York, NY, USA, 2011. ACM.

[15] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000. AAI9980887.

[16] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[17] Gaeul Jeong, Sunghun Kim, and Thomas Zimmermann. Improving bug triage with bug tossing graphs. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 111–120, New York, NY, USA, 2009. ACM.

[18] Sunghun Kim and E. James Whitehead, Jr. How long did it take to fix bugs? In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 173–174, New York, NY, USA, 2006. ACM.

[19] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann, and Witten IH. The weka data mining software: an update. *SIGKDD Explor News*, pages 10–18, 2011.

[20] Andrew Mccallum and Kamal Nigam. A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on 'Learning for Text Categorization'*, 1998.

[21] Amazon ML. Amazon machine learning. `https://aws.amazon.com/machine-learning/`, Aug 24, 2017.

[22] Azure ML. Microsoft azure machine learning studio. `https://studio.azureml.net/`, Aug 24, 2017.

[23] Google Cloud ML. Cloud machine learning engine. `https://cloud.google.com/ml-engine/`, Aug 24, 2017.

[24] Michael J. Pazzani and Daniel Billsus. The adaptive web. chapter Content-based Recommendation Systems, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.

[25] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Schoelkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.

[26] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[27] M.P. Robillard, W. Maalej, R.J. Walker, and T. Zimmermann. *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, 2014.

[28] A. L. Samuel. Some studies in machine learning using the game of checkers. *IBM J. Res. Dev.*, 3(3):210–229, July 1959.

[29] J. Ben Schafer, Dan Frankowski, Jon Herlocker, and Shilad Sen. The adaptive web. chapter Collaborative Filtering Recommender Systems, pages 291–324. Springer-Verlag, Berlin, Heidelberg, 2007.

[30] Ramin Shokripour, John Anvik, Zarinah M. Kasirun, and Sima Zamani. Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 2–11, Piscataway, NJ, USA, 2013. IEEE Press.

[31] Skytree Inc. Skytree server. `http://www.skytree.net`, Aug 24, 2017.

Appendix A

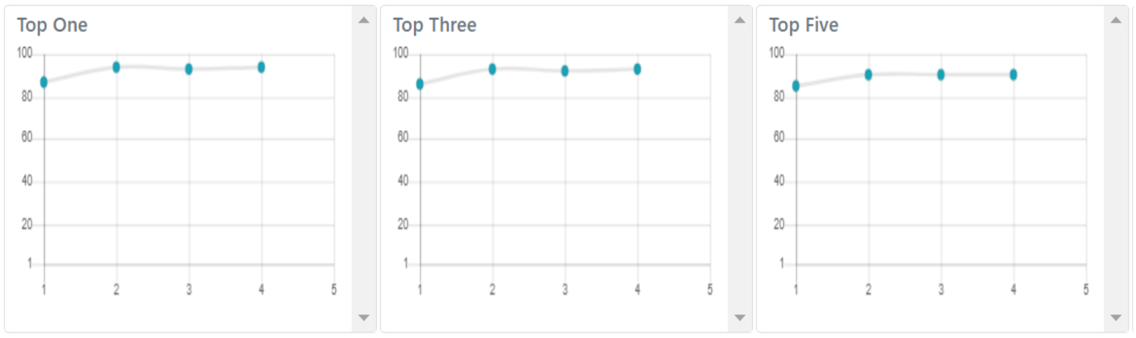
Assignment Recommender Evaluation Results

This appendix presents the assignment recommender evaluation results for the three different projects from Section 5.1.2.

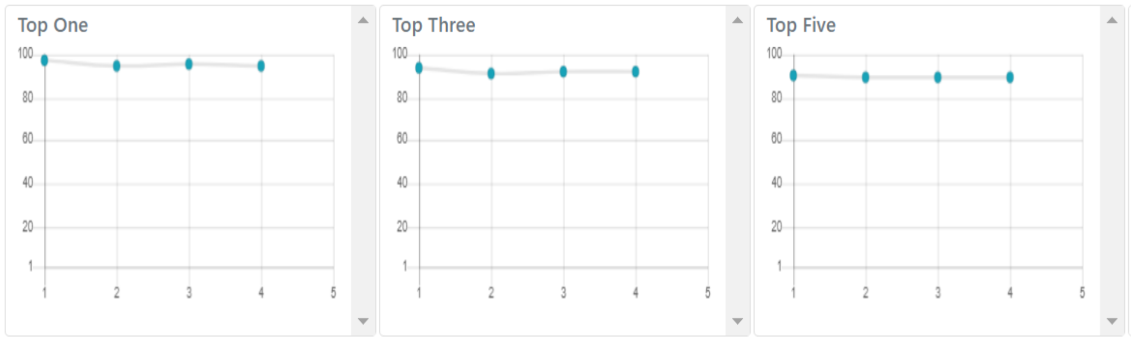
A.1 LibreOffice

Table A.1: Evaluation results of assignment recommender created using LibreOffice dataset.

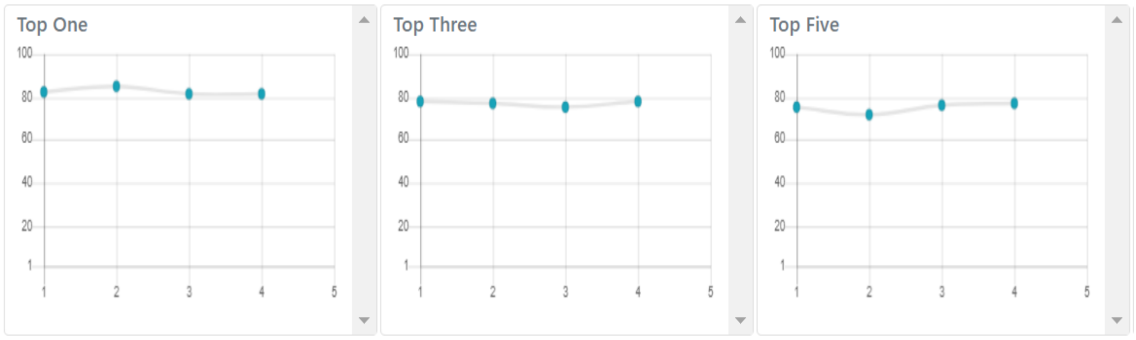
Algorithm	**Sampling Technique**	**Precision**			**Recall**			**Confusion Matrix**	
		Top 1	**Top 3**	**Top 5**	**Top 1**	**Top 3**	**Top 5**	**Correct Instance**	**Incorrect Instance**
SVM	SMOTE	88	87	86	2	6	10	36	136
	MANUAL	94	93	91	2	7	11	93	79
	CLUSTERING	94	93	90	2	7	11	86	86
	NONE	95	93	91	2	7	11	97	75
Naive Bayes	SMOTE	97	95	91	2	7	11	103	69
	MANUAL	95	92	90	2	7	11	70	102
	CLUSTERING	97	93	90	2	7	11	68	104
	NONE	96	93	90	2	7	11	82	90
C45	SMOTE	83	78	76	2	5	8	0	172
	MANUAL	85	78	73	2	5	7	23	149
	CLUSTERING	82	76	76	2	4	8	22	150
	NONE	81	78	78	2	5	8	55	117
Rules	SMOTE	74	58	63	1	3	6	0	172
	MANUAL	31	36	51	1	2	5	0	172
	CLUSTERING	97	93	90	2	7	11	154	18
	NONE	97	94	91	2	7	11	161	11



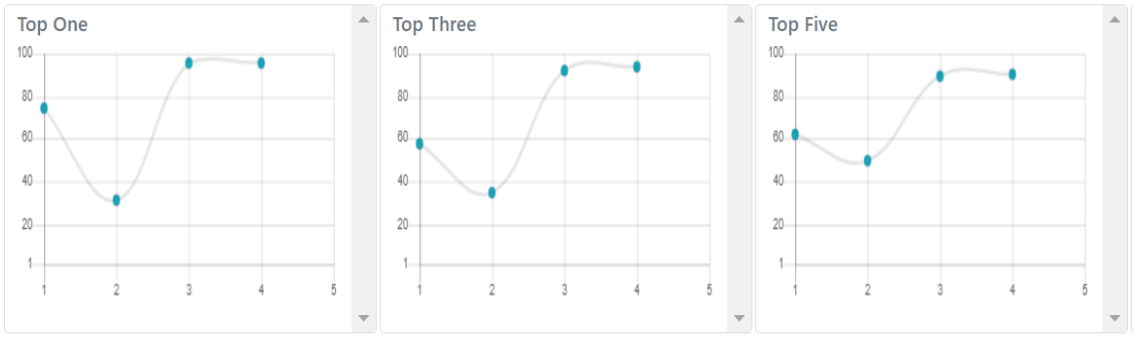
(a) SVM



(b) Naive Bayes



(c) C4.5



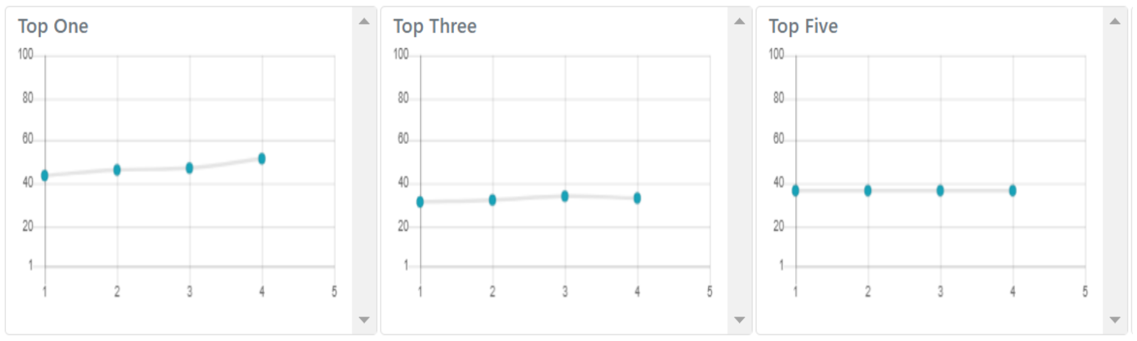
(d) Rules

Figure A.1: Top 1, 3 and 5 precision for the recommendations created using different types of sampling techniques on LibreOffice dataset.
Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None

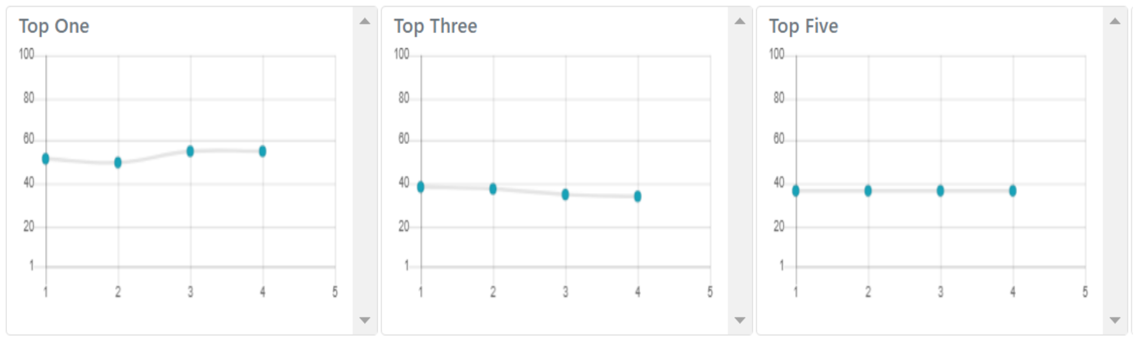
A.2 Firefox

Table A.2: Evaluation results of assignment recommender created using Firefox dataset.

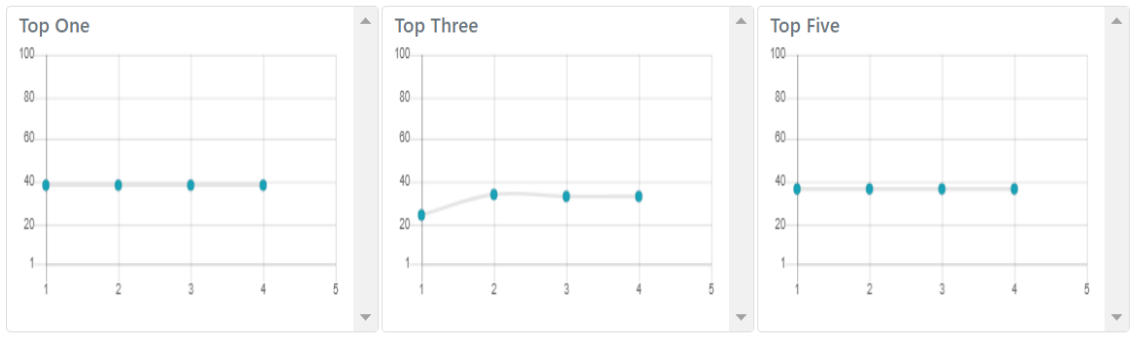
Algorithm	**Sampling Technique**	**Precision**			**Recall**			**Confusion Matrix**	
		Top 1	**Top 3**	**Top 5**	**Top 1**	**Top 3**	**Top 5**	**Correct Instance**	**Incorrect Instance**
SVM	SMOTE	43	31	36	10	15	19	15	63
	MANUAL	46	33	36	10	15	19	1	77
	CLUSTERING	47	35	36	11	15	19	3	75
	NONE	51	34	36	11	15	19	2	76
Naive Bayes	SMOTE	51	39	36	11	16	19	5	73
	MANUAL	50	38	36	11	16	19	6	72
	CLUSTERING	55	36	36	11	15	19	6	72
	NONE	55	35	36	11	15	19	5	73
C45	SMOTE	38	25	36	10	14	19	0	78
	MANUAL	38	34	36	9	14	19	6	72
	CLUSTERING	38	33	36	10	13	19	7	71
	NONE	38	34	36	10	13	19	5	73
Rules	SMOTE	34	28	36	9	12	19	78	0
	MANUAL	36	37	36	9	15	19	0	78
	CLUSTERING	46	35	36	11	14	19	0	78
	NONE	46	35	36	11	14	19	0	78



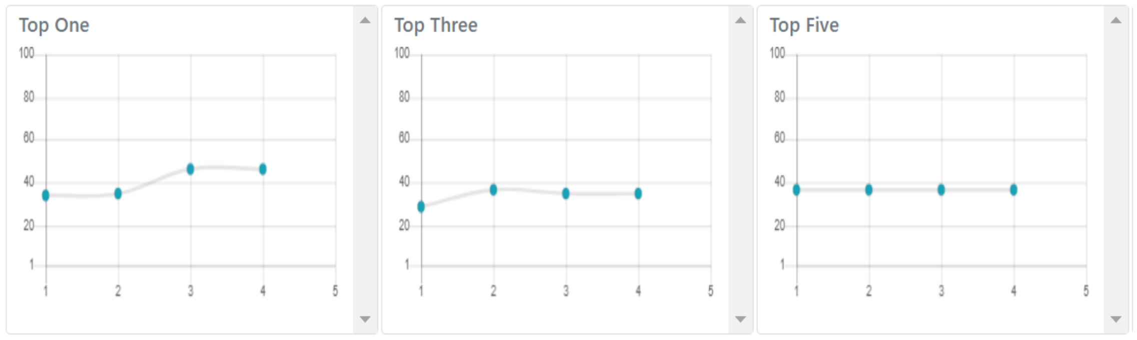
(a) SVM



(b) Naive Bayes



(c) C4.5



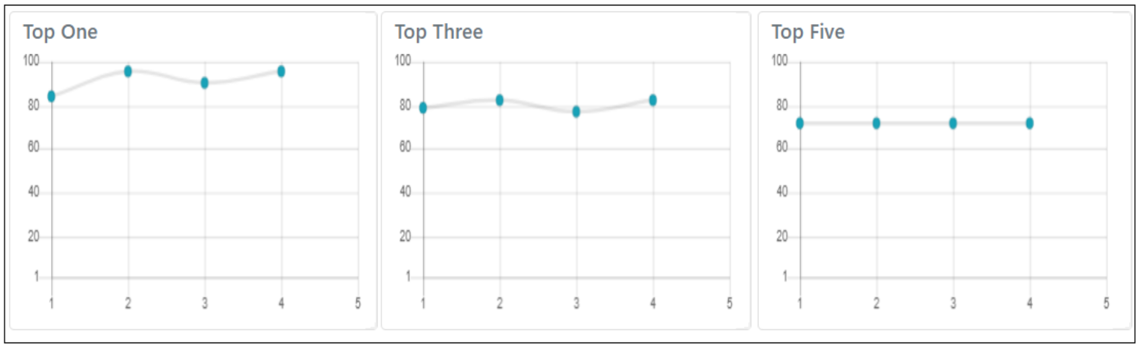
(d) Rules

Figure A.2: Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Firefox dataset.
Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None

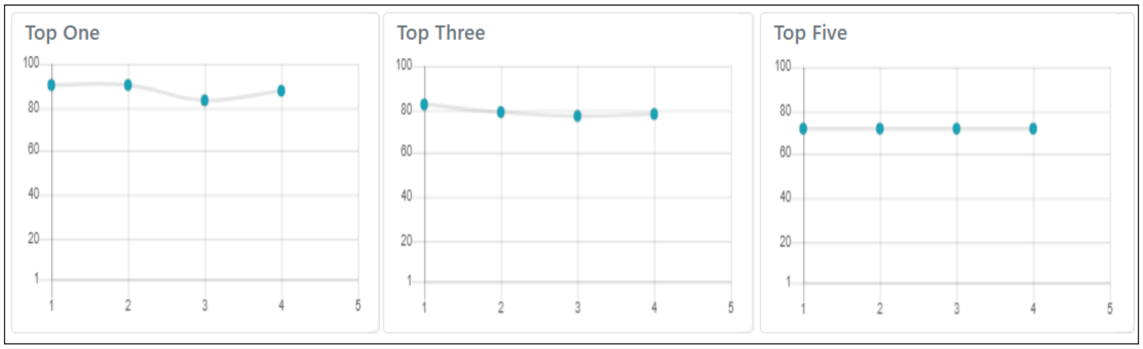
A.3 Plasmashell

Table A.3: Evaluation results of assignment recommender created using Plasmashell dataset.

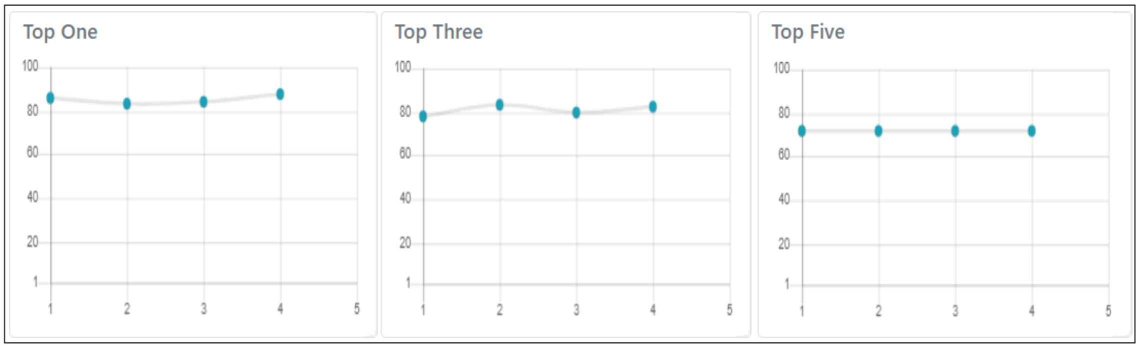
Algorithm	**Sampling Technique**	**Precision**			**Recall**			**Confusion Matrix**	
		Top 1	**Top 3**	**Top 5**	**Top 1**	**Top 3**	**Top 5**	**Correct Instance**	**Incorrect Instance**
SVM	SMOTE	85	80	73	8	25	38	14	39
	MANUAL	96	82	73	11	25	38	13	40
	CLUSTERING	91	78	73	10	23	38	28	25
	NONE	96	83	73	11	26	38	11	42
Naive Bayes	SMOTE	91	83	73	10	26	38	9	44
	MANUAL	91	80	73	10	26	38	9	44
	CLUSTERING	83	77	73	9	25	38	22	31
	NONE	89	78	73	10	25	38	9	44
C45	SMOTE	87	79	73	10	25	38	0	53
	MANUAL	83	83	73	8	26	38	10	43
	CLUSTERING	85	81	73	9	26	38	22	31
	NONE	89	83	73	10	26	38	7	46
Rules	SMOTE	77	64	73	8	20	38	0	53
	MANUAL	77	69	73	8	21	38	0	53
	CLUSTERING	87	68	73	9	21	38	47	6
	NONE	92	85	73	10	27	38	0	53



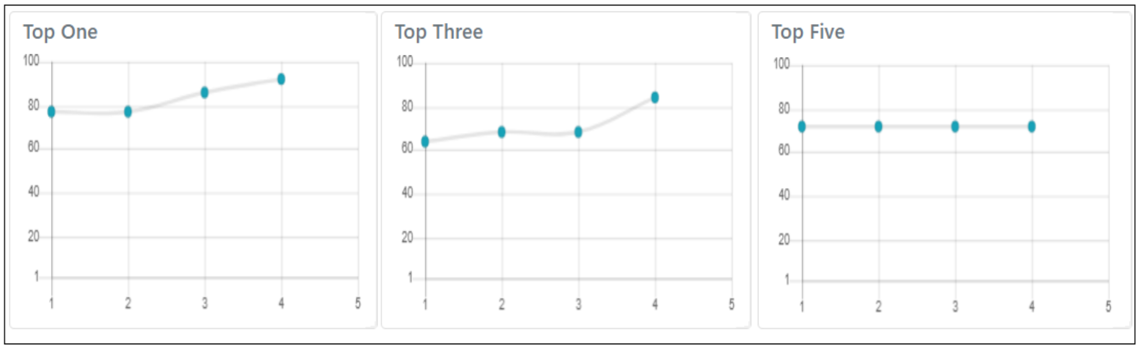
(a) SVM



(b) Naive Bayes



(c) C4.5



(d) Rules

Figure A.3: Top 1, 3 and 5 precision for the recommendations created using 4 different types of sampling techniques on Plasmashell dataset.
Data points: 1. SMOTE 2. Manual Oversampling 3. Clusters 4. None

Appendix B

Field Study Outcome

This appendix presents the results of questions that were asked to participants while filling initial survey and post tool usage survey after they completed the creation of an assignment recommender from Section 5.2.1.

B.1 Initial Survey Results

Table B.1: Participants demographic information and technical background details.

Age Ranges	**Gender**	**Nationality**	**Highest Degree**	**Job Function**	**Years of Experience**	**Level of Experience With Triaging Bug Reports**	**Frequency of Logging a Bug in Issue Tracking System**	**Level of Experience With Using Machine Learning Algorithms**
26 - 39	Male	Indian	Graduate (Masters or Doctorate)	Application Developer	7-10	Beginner	Often	Beginner
26 - 39	Male	Indian	Graduate (Masters or Doctorate)	Project Manager	4-6	Developing	Always	Developing
18 - 25	Male	Indian	Bachelors degree (4-year)	Application Developer	0-3	Developing	Often	Developing
26 - 39	Male	Canadian	Bachelors degree (4-year)	Application Developer	4-6	Advanced	Always	Advanced
26 - 39	Male	Indian	Graduate (Masters or Doctorate)	Project Manager	7-10	Beginner	Always	Beginner
18 - 25	Female	India	Bachelors degree (4-year)	Student	0-3	Beginner	Occasionally	Beginner
26 - 39	Male	Indian	Bachelors degree (4-year)	Application Developer	7-10	Competent	Occasionally	Beginner
26 - 39	Male	Sri Lankan	Graduate (Masters or Doctorate)	Student	4-6	Beginner	Rarely	Developing
40 - 59	Prefer Not to Answer	USA	Graduate (Masters or Doctorate)	Project Manager	>15	Advanced	Always	Developing
18 - 25	Male	Indian	High School	Student	4-6	Developing	Occasionally	Beginner

B.2 CASTR Usage Feedback

Table B.2: Participant's feedback on CASTR tool.

CASTR Easy to Use?	Recommend CASTR?	Reduce the Time?	Improvements to the CASTR User Interface?	Improvements in the Workflow of Recommender Creation?
Very	Moderately	Moderately		
Very	Extremely	Extremely	The working of CASTR is really good. Still we can improve the Line Graph by adding little more detail to it so that it will be easily understandable. Other than that I tried it with all different Algorithms and worked perfectly for me as I expected the output to be.	Everything in Recommender looked perfect but we can still improve it by making some changes to Sample Recommendations. I think it should me more detailed and each entry should be separated by lines or some space so that it will be easy to read and understand.
Very	Very	Very	Looks pretty good.	No
Extremely	Extremely	Extremely	The whole tool is amazing in terms of user interface. A slight improvement could be done on the login page perhaps.	Nope. It is one of the best I have come across till now.
Very	Moderately	Very	User interface is good.	
Moderately	Moderately	Very	Yes, UI is not so impressive. One has to read user manual to operate./play with the tool. Too many scrollbars in single page. Graphs axis should have labels to understand what it is for. Current graph does not convey any anything. One has to think and figure out himself that what exact it visualized.	Configuration page can be more descriptive and wizard based flow could have been better then what we have now (everything in single page).
Extremely	Very	Very	I wonder whether a triagger needs to access previous recommendations he/she made. In that case, do you think it will be beneficial to add a save model/configuration option?	
Moderately	Moderately	Moderately	more feedback on choices made	It would be more useful if that was a wizard for creating a recommender. Also guidance on what ML approach to use.
Slightly	Moderately	Moderately	Current User interface is very difficult to understand by self, what each component is actually meant to. It can be improved by providing more details of the usage for each of the element.	
Moderately	Very	Very	None	None

Appendix C

Quantitative Results

This appendix presents the quantitative results of the creation of an assignment recommender from Section 5.2.4.

C.1 Heuristics Configurations

Table C.1: Heuristics configurations used in assignment recommender creation by User1.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
Naive Bayes	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	5 / 108	9	AssignedTo	Do Not Use	FirstResponder	Resolver	FirstResponder	Reporter	Resolver	FixedBy	FixedBy	AssignedTo
SVM	1 / 93	9	AssignedTo	Do Not Use	FixedBy	Resolver	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	1 / 61	9	AssignedTo	Do Not Use	FixedBy	FixedBy	Reporter	Attachment	Resolver	AssignedTo	AssignedTo	AssignedTo
Naive Bayes	26 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
Naive Bayes	1 / 141	9	AssignedTo	Do Not Use	Resolver	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
C45	1 / 141	9	AssignedTo	Do Not Use	Resolver	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	1 / 124	1	Resolver	Do Not Use								

Table C.2: Heuristics configurations used in assignment recommender creation by User2.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Naive Bayes	15 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
SVM	15 / 137	5	Resolver	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver				
SVM	3 / 137	5	Resolver	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver				

Table C.3: Heuristics configurations used in assignment recommender creation by User3.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	26 / 112	9	AssignedTo	Do Not Use	FixedBy	Resolver	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
Naive Bayes	26 / 112	9	AssignedTo	Do Not Use	FixedBy	Resolver	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
Naive Bayes	1 / 127	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
Rules	74 / 127	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
Naive Bayes	1 / 152	9	AssignedTo	Do Not Use	Resolver	FirstResponder	FirstResponder	FirstResponder	AssignedTo	Resolver	Resolver	FirstResponder
SVM	64 / 146	9	AssignedTo	Do Not Use	Resolver	FirstResponder	FirstResponder	Resolver	Resolver	Resolver	Resolver	FirstResponder

Table C.4: Heuristics configurations used in assignment recommender creation by User4.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
Naive Bayes	49 / 73	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
C45	49 / 73	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
C45	21 / 73	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
Rules	30 / 73	9	FixedBy	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
Rules	19 / 56	5	FixedBy	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder				
Rules	19 / 56	5	FixedBy	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder				

Table C.5: Heuristics configurations used in assignment recommender creation by User5.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Naive Bayes	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
C45	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
SVM	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Naive Bayes	10 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
SVM	20 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Naive Bayes	20 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
C45	20 / 130	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use

Table C.6: Heuristics configurations used in assignment recommender creation by User6.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	1 / 141	3	AssignedTo	Do Not Use	Resolver	AssignedTo						

Table C.7: Heuristics configurations used in assignment recommender creation by User7.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
Rules	1 / 59	5	AssignedTo	Do Not Use	FixedBy	FixedBy	Reporter	Attachment				
SVM	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use				
SVM	1 / 61	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
SVM	1 / 61	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
Naive Bayes	1 / 61	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
C45	1 / 61	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
Rules	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Attachment				
Rules	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use				
C45	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use				
C45	1 / 83	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder				
SVM	1 / 89	4	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter					
SVM	1 / 89	4	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter					
SVM	1 / 84	6	AssignedTo	Do Not Use	Resolver	AssignedTo	Reporter	FirstResponder	Attachment			
SVM	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Attachment				
Naive Bayes	1 / 64	5	AssignedTo	Do Not Use	FixedBy	Resolver	Attachment	Reporter				
Naive Bayes	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
Naive Bayes	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
SVM	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
C45	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
Rules	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
SVM	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
Naive Bayes	1 / 61	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			

Table C.8: Heuristics configurations used in assignment recommender creation by User8.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
Naive Bayes	1 / 117	4	AssignedTo	Do Not Use	Reporter	AssignedTo	AssignedTo					

Table C.9: Heuristics configurations used in assignment recommender creation by User9.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	10 / 115	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Naive Bayes	10 / 115	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
C45	10 / 115	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
Rules	10 / 115	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use

Table C.10: Heuristics configurations used in assignment recommender creation by User10.

Algorithm	Min./Max. Threshold	# Heuristics	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
SVM	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	19 / 120	3	AssignedTo	Do Not Use	Resolver	Resolver						
SVM	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
Naive Bayes	1 / 31	4	FixedBy	Do Not Use	Reporter	FixedBy	AssignedTo					
SVM	1 / 136	6	Attachment	Do Not Use	AssignedTo	Resolver	AssignedTo	AssignedTo	AssignedTo			
SVM	1 / 102	6	Attachment	Do Not Use	Resolver	Resolver	AssignedTo	AssignedTo	FirstResponder			
Rules	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
SVM	1 / 116	8	AssignedTo	Do Not Use	Resolver	AssignedTo	AssignedTo	Reporter	AssignedTo	AssignedTo	AssignedTo	
SVM	1 / 170	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo

Table C.11: Repetitive Heuristics configurations used in assignment recommender creation.

# of Times Configuration Used	# of Heuristics Selected	Other Heuristics	Duplicate	Fixed	Worksforme	Invalid	Unknown	Upstream	Wontfix	Later	Moved
10	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
8	9	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
7	6	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Do Not Use			
4	9	AssignedTo	Do Not Use	FixedBy	Resolver	Resolver	Resolver	Resolver	FirstResponder	Do Not Use	Do Not Use
4	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use	Attachment	FirstResponder	AssignedTo	Reporter
3	9	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
3	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Do Not Use				
2	9	AssignedTo	Do Not Use	FixedBy	Resolver	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
2	9	AssignedTo	Do Not Use	FixedBy	FirstResponder	FirstResponder	FirstResponder	FixedBy	Resolver	FirstResponder	FirstResponder
2	9	AssignedTo	Do Not Use	Resolver	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo	AssignedTo
2	5	Resolver	Do Not Use	FixedBy	FirstResponder	Resolver	Resolver				
2	5	FixedBy	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder				
2	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	Attachment				
2	4	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter					
1	9	AssignedTo	Do Not Use	Resolver	FirstResponder	FirstResponder	FirstResponder	AssignedTo	Resolver	Resolver	FirstResponder
1	9	AssignedTo	Do Not Use	FirstResponder	Resolver	FirstResponder	Reporter	Resolver	FixedBy	FixedBy	AssignedTo
1	9	FixedBy	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder	Resolver	Attachment	FirstResponder	Resolver
1	9	AssignedTo	Do Not Use	Resolver	FirstResponder	FirstResponder	Resolver	Resolver	Resolver	Resolver	FirstResponder
1	9	AssignedTo	Do Not Use	FixedBy	Resolver	AssignedTo	Do Not Use	AssignedTo	AssignedTo	AssignedTo	AssignedTo
1	9	AssignedTo	Do Not Use	FixedBy	FixedBy	Reporter	Attachment	Resolver	AssignedTo	AssignedTo	AssignedTo
1	8	AssignedTo	Do Not Use	Resolver	AssignedTo	AssignedTo	Reporter	AssignedTo	AssignedTo	AssignedTo	
1	6	Attachment	Do Not Use	Resolver	Resolver	AssignedTo	AssignedTo	FirstResponder			
1	6	Attachment	Do Not Use	AssignedTo	Resolver	AssignedTo	AssignedTo	AssignedTo			
1	6	AssignedTo	Do Not Use	Resolver	AssignedTo	Reporter	FirstResponder	Attachment			
1	5	AssignedTo	Do Not Use	FixedBy	Resolver	Reporter	FirstResponder				
1	5	AssignedTo	Do Not Use	FixedBy	Resolver	Attachment	Reporter				
1	5	AssignedTo	Do Not Use	FixedBy	FixedBy	Reporter	Attachment				
1	4	FixedBy	Do Not Use	Reporter	FixedBy	AssignedTo					Reporter
1	4	AssignedTo	Do Not Use	Reporter	AssignedTo	AssignedTo					
1	3	AssignedTo	Do Not Use	Resolver	AssignedTo						
1	3	AssignedTo	Do Not Use	Resolver	Resolver						
1	1	Resolver	Do Not Use								

C.2 Precision and Recall

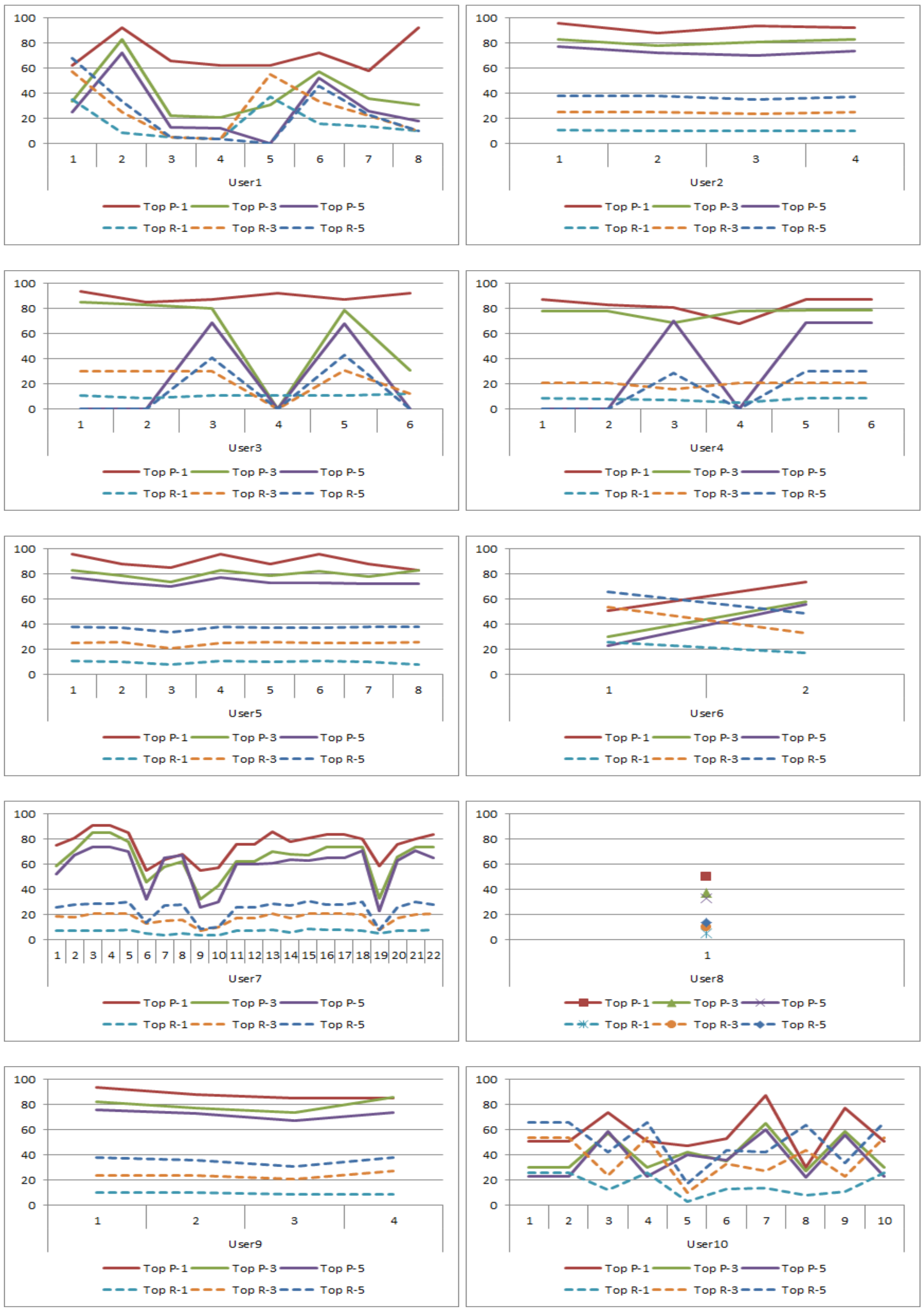
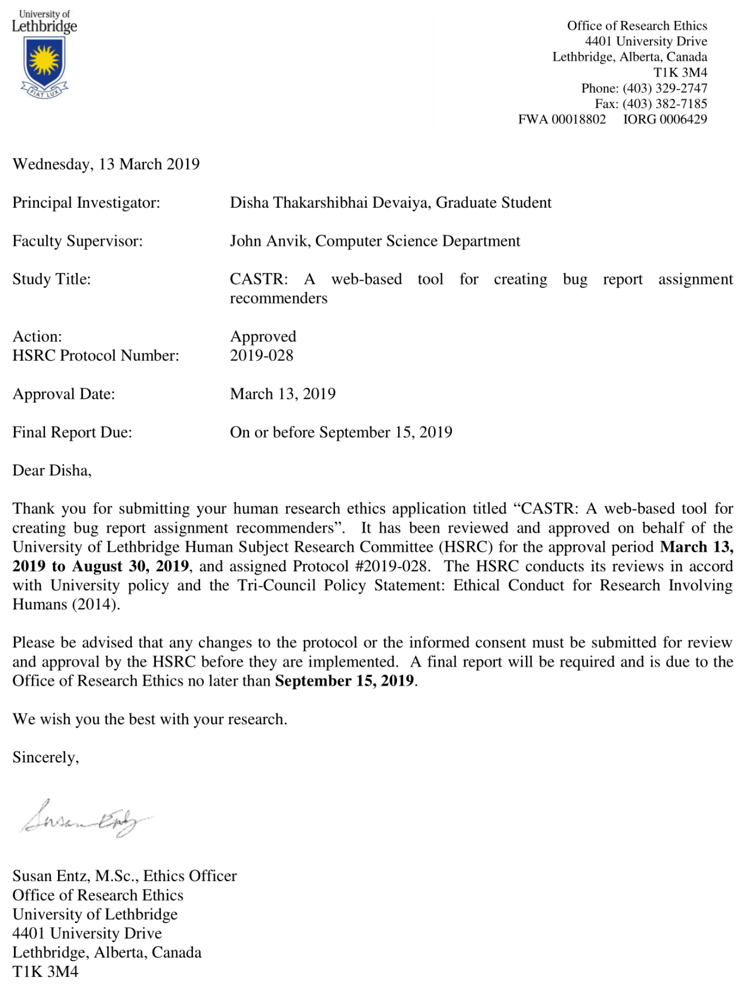


Figure C.1: Top 1, 3 and 5 precision and recall from all the recommendations created by participants.

Appendix D

Ethics Certificate



Office of Research Ethics
4401 University Drive
Lethbridge, Alberta, Canada
T1K 3M4
Phone: (403) 329-2747
Fax: (403) 382-7185
FWA 00018802 IORG 0006429

Wednesday, 13 March 2019

Principal Investigator:	Disha Thakarshibhai Devaiya, Graduate Student
Faculty Supervisor:	John Anvik, Computer Science Department
Study Title:	CASTR: A web-based tool for creating bug report assignment recommenders
Action:	Approved
HSRC Protocol Number:	2019-028
Approval Date:	March 13, 2019
Final Report Due:	On or before September 15, 2019

Dear Disha,

Thank you for submitting your human research ethics application titled "CASTR: A web-based tool for creating bug report assignment recommenders". It has been reviewed and approved on behalf of the University of Lethbridge Human Subject Research Committee (HSRC) for the approval period **March 13, 2019 to August 30, 2019**, and assigned Protocol #2019-028. The HSRC conducts its reviews in accord with University policy and the Tri-Council Policy Statement: Ethical Conduct for Research Involving Humans (2014).

Please be advised that any changes to the protocol or the informed consent must be submitted for review and approval by the HSRC before they are implemented. A final report will be required and is due to the Office of Research Ethics no later than **September 15, 2019**.

We wish you the best with your research.

Sincerely,

[illegible]

Susan Entz, M.Sc., Ethics Officer
Office of Research Ethics
University of Lethbridge
4401 University Drive
Lethbridge, Alberta, Canada
T1K 3M4