Fuzzy and Probabilistic Rule-Based Approaches to Identify Fault Prone Files

Piyush Kumar Korlepara, The University of Western Ontario

Supervisor: Kontogiannis, Kostas, The University of Western Ontario
A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science
© Piyush Kumar Korlepara 2021

Follow this and additional works at: https://ir.lib.uwo.ca/etd

Part of the Software Engineering Commons

Recommended Citation
https://ir.lib.uwo.ca/etd/7941

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact wlsadmin@uwo.ca.
Abstract

Most software fault proneness prediction techniques utilize machine learning models which act as black boxes when performing predictions. Software developers cannot obtain any insights as to why such trained models reached their conclusions when applied to new data. This leads to a reduced confidence in accepting the prediction results while applying the model in complex systems. In this thesis, we propose two rule-based and programming language-agnostic fault proneness prediction techniques. The first technique utilizes fuzzy reasoning, while the second utilizes Markov Logic Networks. The rules operate on facts that are produced by harvesting and postprocessing raw data extracted from the GitHub records of the system that is being analyzed. Furthermore, files in each GitHub record are reconciled using bug resolution reports from corresponding Bugzilla repositories. The reconciliation process is used for tagging purposes so that the number of false positives in the raw data can be reduced. To better organize the extracted data, we group GitHub commits to form what we refer to as segments. Reasoning about fault proneness of a file is then considered at the level of a segment (i.e. whether the file will exhibit a failure in the time frame of the next segment – e.g. the 10 next commits).

In this thesis we have identified twenty generic rules, and we propose two processes to customize these rules for each system. The first process aims to select a subset of these twenty rules that perform the best (i.e. maximize prediction recall and prediction accuracy) for given a project. The selection is performed on a subset of the project’s historical data that serves as a training set and then these rules are applied to the rest of the system (current data). The second process aims to identify new rules by examining areas of opportunity to maximize prediction recall and accuracy. We have evaluated the proposed approach by applying six different strategies to answer four research questions related to which technique is best, whether there is a common set of rules that performs equally well in all projects, whether a rule set performing well in one project can be used in another, and whether customizing the rules is better for performance compared to generic ones.

We conclude the thesis by providing pointers for future research and how rule-based systems can be used in the field of Fault Prediction.
Summary for Lay Audience

In this thesis, we use two rule-based techniques to identify fault-prone files in large software systems. The vast majority of the literature in this field use Machine Learning for prediction. However, the drawback of Machine Learning approaches is that they do not provide explanations to the users on why and how a prediction result is reached. In the proposed approach, we allow for expert knowledge to be encoded in the form of If-Then rules. We utilize and compare two reasoning approaches, one based on Fuzzy Reasoning and other on Markov Logic Networks. We introduce twenty generic rules that can be optimized for each project in order to yield maximal recall. The results indicate that the rule-based approaches provide comparable results with Machine Learning approaches with the added benefit of being able to provide explanations and a high degree of customizability.
Acknowledgements

The completion of this thesis would not have been possible without the constant mentorship and expertise of my advisor Dr. Kostas Kontogiannis. Your keen insight and knowledge into the subject matter steered me through this research. Special thanks to my colleagues Marios Grigoriou and Ria, whose support allowed my studies to go to the extra mile. I’m sorry for all the extra work I imposed on you! I would also like to thank Chris Brealey of IBM Toronto Lab, and Alberto Giammaria of IBM Austin TX. Lab for their invaluable comments and technical discussions over the past year. I am indebted to IBM Center for Advanced Studies and NSERC for providing financial support for my thesis.

My biggest thanks and appreciation go to my parents for all the support they have shown me through this research. It hasn’t been easy being away from you, but you have always been there to listen to me and help me when I needed you the most. Finally, I would like to thank my committee for their time and effort evaluating my thesis and providing valuable feedback.
# Table of Contents

Abstract.......................................................................................................................................... ii  
Summary for Lay Audience........................................................................................................ iii  
Acknowledgements ...................................................................................................................... iv  
List of Figures ............................................................................................................................. viii  
List of Tables ................................................................................................................................ ix  

Chapter 1: Introduction ............................................................................................................... 1  
1.1 Introduction ........................................................................................................................... 1  
1.2 Motivation ............................................................................................................................. 3  
1.3 Thesis Contributions ............................................................................................................. 4  
1.4 Research Questions ............................................................................................................... 5  
1.5 Thesis Outline ....................................................................................................................... 6  

Chapter 2: Background ................................................................................................................ 7  
2.1 Threshold Moving ................................................................................................................. 7  
2.2 GitHub ................................................................................................................................... 8  
2.3 Bugzilla ................................................................................................................................. 9  
2.4 Fuzzy Logic ............................................................................................................................ 10  
2.4.1 Fuzzy Sets .................................................................................................................... 10  
2.4.2 Membership Functions ................................................................................................. 10  
2.4.3 Linguistic Variables ..................................................................................................... 12  
2.4.4 Fuzzy Set Operations ................................................................................................... 13  
2.4.5 Fuzzy Conditional Rules .............................................................................................. 14  
2.4.6 Fuzzy Inference ............................................................................................................ 15  
2.4.7 Defuzzification ............................................................................................................. 15  
2.4.8 Fuzzy Inference and Defuzzification Example ............................................................ 16  
2.5 Markov Logic Networks ..................................................................................................... 17  
2.5.1 Markov Networks ........................................................................................................ 17  
2.5.2 First-Order Logic .......................................................................................................... 19  
2.5.3 Markov Logic Networks .............................................................................................. 20  
2.5.4 Inference ...................................................................................................................... 22  
2.5.5 Weight Learning .......................................................................................................... 23
# Chapter 3: Related Work

3.1 Machine Learning ................................................................. 25
3.2 Fuzzy Logic ................................................................. 27
3.3 Markov Logic Networks ......................................................... 29
3.4 Bug Prediction .................................................................... 30
   3.4.1 Package Level ....................................................... 31
   3.4.2 Class Level .......................................................... 32
   3.4.3 File Level ............................................................. 34
   3.4.4 Method Level ....................................................... 38
   3.4.5 Other ..................................................................... 40
   3.4.6 Comparison of Results ........................................... 41
   3.4.7 Recent Work ........................................................ 42
   3.4.8 Research Gap ........................................................ 45

# Chapter 4: Data Modelling .......................................................... 46

4.1 Repository Data Extraction ................................................ 46
   4.1.1 GitHub Model ...................................................... 46
   4.1.2 Bugzilla Model ...................................................... 46
   4.1.3 GitHub-Bugzilla Data Reconciliation ......................... 48
   4.1.4 Data Format .......................................................... 49
   4.1.5 Project Selection ..................................................... 50
4.2 Segments ........................................................................... 51
4.3 Metrics ............................................................................. 53
4.4 Threshold Calculations ....................................................... 55
4.5 Rule Repository ................................................................. 57
4.6 Customized Rules ............................................................... 59

# Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base ........................................ 61

5.1 Fuzzy Based Fault Proneness Identification ...................... 61
   5.1.1 System ..................................................................... 61
   5.1.2 Intermediate Data Storage Format ......................... 63
   5.1.3 Fuzzy Control Language ......................................... 64
   5.1.4 Membership Functions Block ................................. 65
   5.1.5 Rule Block ............................................................. 67
5.1.6 Rule Set Optimization

5.2 Markov Logic Networks Based Fault Proneness Identification

5.2.1 System

5.2.2 MLN Predicates for Fault-Proneness Prediction

5.2.3 MLN Rules

5.2.4 Data Facts (Fact Base)

5.2.5 Weight Learning

5.2.6 Discussion

Chapter 6: Experiments and Results

6.1 Strategies

6.2 Strategy-1: Best Subset of Rules

6.2.1 Discussion

6.3 Strategy-2: Expanded Rules

6.3.1 Discussion

6.4 Strategy-3: Common Rules

6.4.1 Discussion

6.5 Strategy-4: Rules Over Time

6.5.1 Discussion

6.6 Strategy-5: Reusing Trained Subset of Rules

6.6.1 Discussion

6.7 Strategy-6: Custom Rules

6.7.1 Discussion

6.8 Comparison with Machine Learning Techniques

Chapter 7: Conclusion and Future Work

7.1 Conclusion

7.2 Thesis Findings

7.3 Threats to validity

7.4 Future Work

Chapter 8: References

Appendix A: Deep Level Results of Reusing Trained Subset of Rules

Appendix B: Expanded Rules for Each Project

Appendix C: Customized Rules for Each Project
List of Figures

Figure 1: Example ROC Curve and Optimum Threshold .............................................................. 8
Figure 2: Membership Functions for the variable “tip” ............................................................... 11
Figure 3: Types of Membership Functions ............................................................................... 12
Figure 4: Membership Functions for the variables Service and Food .................................. 13
Figure 5a: Union of two fuzzy sets ........................................................................................... 14
Figure 6: Fuzzy Inference Example .......................................................................................... 16
Figure 7: Example Markov Network .......................................................................................... 18
Figure 8: Example Grounded Markov Network ....................................................................... 22
Figure 9: Data Model for Raw GitHub Data [108] ................................................................. 47
Figure 10: Timeline for Reconciled Data [108] ...................................................................... 48
Figure 11: Data Extraction and Reconciliation Overview ....................................................... 49
Figure 12: Illustrative Example for Segment Widths ............................................................... 53
Figure 13: First screenshot of output given by the custom segment analyzing program ........ 59
Figure 14: Second screenshot of output given by the custom segment analyzing program ...... 60
Figure 15: Overall System Design for Fuzzy Logic Model ....................................................... 62
Figure 16: Overall System Design for MLN Model ................................................................. 71
List of Tables

Table 1: Joint Probability Distribution for Overworking and Stress .......................................................... 18
Table 2: Example MLN Rules ...................................................................................................................... 22
Table 3: Projects Examined and their last five years' profile ..................................................................... 51
Table 4: JSON structure for a segment ....................................................................................................... 64
Table 5: Variable Block in FCL File .......................................................................................................... 65
Table 6: Membership Function Block in FCL File ....................................................................................... 66
Table 7: Rule Block in FCL File .................................................................................................................. 68
Table 8: Best Performing Rules for kdelibs ................................................................................................. 69
Table 9: Rules and their Frequency of Use .................................................................................................. 69
Table 10: Example db file ............................................................................................................................ 76
Table 11: Example Output MLN File .......................................................................................................... 78
Table 12: Fuzzy Logic Results using rule subset .......................................................................................... 83
Table 13: MLN Results using trained rule subset ....................................................................................... 84
Table 14: Fuzzy Logic Results using Expanded Rules ............................................................................... 86
Table 15: MLN Results for Expanded Rules ............................................................................................... 87
Table 16: Fuzzy Logic Results for Common Rules ..................................................................................... 89
Table 17: MLN Results for Common Rules ................................................................................................. 90
Table 18: Fuzzy Logic Results over time .................................................................................................... 92
Table 19: MLN Results Over Time .............................................................................................................. 92
Table 20: Fuzzy Logic Results for using trained rules for other projects .................................................... 94
Table 21: MLN Results for using trained rules on other projects ............................................................... 95
Table 22: Fuzzy Logic Results for Custom Rules ...................................................................................... 97
Table 23: MLN Results for Custom Rules .................................................................................................. 98
Table 24: Comparison of Rule-Based and Machine Learning .................................................................... 101
Table 25: Strategy that gives the best results for each project .................................................................... 104
Chapter 1: Introduction

1.1 Introduction

Software maintenance and evolution is one of the costliest phases of the software lifecycle. Introducing changes to a complex software system not only requires careful planning and understanding of the dependencies between the different components, but also requires significant testing in order to ensure that the changes did not introduce any faults or touch upon error prone modules. The whole problem of keeping a system healthy when maintained and evolved becomes even more complex in the context of *continuous software engineering*, where new features are introduced very frequently and in super-short release cycles. The approach proposed by the software engineering community is to provide infrastructure to DevOps engineers to introduce testing and risk analysis as early as possible in the software lifecycle, an approach that is referred to as *shift-left*. In this respect, we need tools and techniques to warn the developers about fault-prone modules when changes are introduced in a software system. These tools and techniques must be capable of effectively predicting the fault proneness of a file or a module that is touched by a change so that faults can be caught earlier in the life cycle of the project. However, it is not enough for a prediction technique to just perform well. Over the past years, the major trends towards developing such prediction techniques rely on machine learning and features that can be extracted from the source code. However, these machine learning models act as black-boxes and do not provide any explanation to the developers as to why a prediction was reached. This lack of transparency leads to a reduced acceptance of fault proneness prediction algorithms. Also, most techniques that rely on features extracted from the source code require the use of parsers and scanners and are language dependent. These techniques are bound to a specific language and do not work well with large projects that have different subsystems written in multiple languages. In this thesis, we propose two language-agnostic models which allow for domain expert knowledge to be considered during the prediction process. The first model is based on Fuzzy reasoning, while the second model is based on Markov Logic Networks. More specifically, we investigate whether domain knowledge, denoted by experts, can be used in a rule-based framework to predict fault prone files and identify the type of data that contributes the most to fault proneness of a file, and
Chapter 1: Introduction

how this rule-based framework compares to the models that use classical machine learning. Using rule-based models allows us to, explicitly represent knowledge in the form of If-Then rules, help reasoning in human-understandable terms, take information from human experts, and translate them to rules used in inferencing. It also enables us to customize the rules based on the domain knowledge of different projects given that a rule that works very well for one project might not work well for a different one.

Therefore, the main focus of this thesis is to create rule-based fault prone prediction models, which identify fault-prone files in large software systems and can help software testers in streamlining the testing process by giving an initial estimate of how to execute and prioritize testing in large systems. Using rule-based models enables us to create customizable and transparent systems with respect to the explanations that can be given to the developers as to how and why predictions were reached. Furthermore, the proposed system is programming language agnostic as it utilises process data harvested form GitHub repositories and not from the system’s source code. For each GitHub record we extract information about the files committed in the record (e.g. co-committed files, lines of code changed) and we populate a data model that yield a collection of facts about the specific GitHub commit record. These facts constitute the fact-base which along with the rules form the knowledge base (KB) of the proposed system. In this thesis, we propose two rule-based techniques. The first applies fuzzy reasoning which requires fuzzy membership functions to be selected for the fuzzy linguistic variables. The second technique applies Markov Logic Networks and Lifted Belief Propagation which requires the rules to be trained by assigning weights to each rule. In this respect, we select a part of the system for training purposes and the rest of the system for testing purposes. It has to be noted that the training phase is not a machine learning phase as in the case of Fuzzy reasoning, it is used to select a subset of rules and not to create a trained ML model, while in the case of Markov Logic Networks the purpose of the training is to assign weights to rules based on their applicability in the training set.

At any given point, the output of the system is collection of files that are considered as fault-prone based on their historical profile and the expert rules. Such a prediction model can be the first line of defense in a pipeline by giving insights for automating continuous DevOps.
1.2 Motivation

Software faults cost a lot of money and can be introduced even by small changes in a software system. It is no surprise that companies invest a lot of resources in software defect detection by hiring more testers and enforcing specific code review practices. Kaner et al. report that most companies maintain a 1-1 ratio of developers and testers, including very large firms like Microsoft [1]. Despite this, many bugs creep up in the final product which can lead to devastating consequences for the stakeholders. Telang et al. estimate that the cost of faulty software is almost $60 billion a year [2].

It is estimated that quality assurance, defect tracking and testing consume about 40-70% of the total development time [3] and it is still insufficient to catch all the defects. Bugs that are caught in the operational phase are exponentially more expensive to fix. According to Boehm’s curve [4], fixing a bug in production can cost up to 250 times more than fixing it in development. That’s why it has become very important to catch most of the bugs in the development and testing phase. By providing predictions on which files are more fault-prone than others, we can focus on testing these files to make testing more efficient and effective.

There have been a lot of techniques and tools to predict the occurrence of bugs. Most studies in the literature have focussed on finding buggy files using Machine Learning with a combination of source code and repository metrics. However, the problem is that the inherent black-box nature of Machine Learning Algorithms make them very unattractive for widespread industrial use. The most commonly used methods are finding bugs by analyzing source code and using machine learning algorithms to predict bugs [5]. The Rule-Based Models discussed in the thesis provide full comprehensibility of the knowledge available to the model while also making it completely customizable by the domain expert. Additionally, Kalliamvakou et al. studied the quality of data mined from GitHub in [6] and found that mined data gives solid information on the descriptive characteristics of the system, which is why we only use repository metrics in our model to make it completely language agnostic.
1.3 Thesis Contributions

This thesis focuses on predicting whether a file may be fault-prone in the near future by using rule-based models which utilize domain expert knowledge and the past history of the file. In this thesis we propose the use of two different rule-based inferencing techniques and we experiment with six different application strategies on eighteen open source projects to answer four key research questions that help with measuring the effectiveness of such rule-based systems. The thesis contributions are as follows:

1. Design a fully language-agnostic fault prediction model that has a high recall and is much more transparent than the standard machine learning algorithms.
2. Experiment with fuzzy and probabilistic rule-based reasoning models to highlight the advantages and disadvantages of using either of them to predict fault-prone files.
3. Investigate whether adding customizable domain knowledge of a system by an expert is effective for classifying fault-prone segments.
4. Investigate the overall advantages and effectiveness of rule-based reasoning systems in predicting file-level fault proneness.
5. Evaluate the approach in order to answer research questions related to the applicability and performance of the approach in different open-source systems.

The tools and elements that constitute this thesis are:

- A custom-made client-side GitHub and Bugzilla extractor tool to mine repository data of a project.
- A Java program to convert the repository data into a meaningful process metrics format that can serve as inputs to the Fuzzy Logic Model and the Markov Logic Networks Model.
- A custom training program that chooses the best subset of Fuzzy Rules from a set of rules that best define the dataset under observation.
- A custom segment user-interface driven analysis program (Section 4.6) that color codes segment metrics based on the threshold levels, and assists experts formulated customized rules for a given project.
• A Python program that can convert the results file of an MLN model into reports that give meaningful performance metrics about the classification.
• Analysis of the various strategies and discussions on the results of using each of them.

1.4 Research Questions

We seek to find answers and evidence to the following research questions:

RQ1: *Is there a common set of rules that can be applied to all projects for predicting fault-prone segments?*

If a set of common rules, that are chosen based on some factors, can successfully be applied to a majority of projects, that would mean that there is no need to perform costly training of rules and it is incredibly helpful on a system that is still young and doesn’t have enough past data to train the rules.

RQ2: *If a collection of rules works in one system, can it be used in other systems?*

This is a typical research question found in the literature for machine learning algorithms. The idea is that if an optimized set of rules performs very well for one system, then it might have similar performance on other projects too. While the inherent idea is similar to RQ1, the difference is in the way common rules are selected. RQ1 uses rule frequency to select the most used rules as the common rule subset, while RQ2 uses the rule subset of a project selected by a training function.

RQ3: *Does the trained set of rules apply equally well over time or does its performance deteriorate?*

We experiment by dividing the testing set into two parts and apply the same rules on both parts. The ideal case would be that the performance remains the same or increases over time.

RQ4: *What is the average increase or decrease of using expanded rules on top of the chosen rule subset?*

We supplement the trained rules returned by the program by adding rules that are customized to each project. We then compare the performance metrics of the expanded rules to the optimized rule subset.
1.5 Thesis Outline

The rest of the thesis is organized as follows. Chapter 2 contains some background information that is relevant to the thesis and provides related work by authors on the field of Machine Learning, Fuzzy Logic, Markov Logic Networks, and Bug Prediction. Chapter 3 covers the data extraction techniques used to get raw data, the calculation of segments, and metrics. It also gives the list of rules used in the rule repository along with information about a program that helps customize rules. In Chapter 4, we discuss the Fuzzy Logic Model framework while also providing background information about the calculations that go on in the inference. We also give examples and formats of the various input files that the model needs. Chapter 5 contains similar information as the previous chapter but in the context of Markov Logic Networks. We discuss in detail how the system works and how the weights of the rules are trained. In Chapter 6, we state all the strategies that we used to run the model and state the results we obtained from both models using each strategy. Chapter 7 contains the discussions, and findings related to the results declared in the previous chapter. We then provide some pointers on where the research can be headed in the future and conclude the thesis.
Chapter 2: Background

2.1 Threshold Moving

Most classification algorithms or techniques predict probabilities which are then mapped to class labels. In the case of binary classification, the default decision threshold is generally assumed to be 0.5. A probability higher than 0.5 would be classified as the positive class label and a probability less than 0.5 is mapped to the negative class label. However, there are certain cases where the default decision threshold is not efficient or optimal.

- If the class distribution in the dataset is severely imbalanced or skewed
- If a misclassification of one class label is more expensive than the misclassification of the other one

Optimal Threshold Selection or Threshold Moving is a technique by which the default decision threshold is tuned to accommodate the imbalance of the data [7]. There are typically two ways of optimizing the threshold, using ROC curves or using precision-recall curves. We chose to use ROC curves since they’re more resilient to class imbalance and that therefore only that technique is explained in this section.

An ROC curve is a performance metric that measures the results of the classifier at different thresholds. It is plotted by taking the True Positive Rate (TPR) on the Y-Axis and the False Positive Rate (FPR) on the X-Axis, as seen in the example given in Figure-1. The rates are calculated against the minority (positive) class.

\[
TPR \text{ or Sensitivity} = \frac{True \ Positives}{True \ Positives + False \ Negatives} \quad (2.1)
\]

\[
Specificity = \frac{True \ Negatives}{True \ Negatives + False \ Positives} \quad (2.2)
\]

\[
FPR = 1 - Specificity = 1 - \frac{False \ Positives}{True \ Negatives + False \ Positives} \quad (2.3)
\]
The threshold that is closest to the top-left plot is considered the optimal threshold. The most popular way to calculate that point is by selecting the threshold that gives the highest Geometric Mean (G-Mean) between the Sensitivity and the Specificity of the classifier.

$$Geometric\ Mean = \sqrt{Sensitivity \times Specificity}$$ \hspace{1cm} (2.4)

An optimized G-Mean will try to balance the values of the two metrics. The large black dot in the graph in Figure-1 shows the optimized threshold for the example ROC curve.

In the context of this thesis, a false negative is far more expensive to the company than a false positive. Also, as can be observed from Table-3, almost all the projects selected in the experiments have a high level of imbalance. That’s why threshold moving is used in the MLN model. The Fuzzy Model doesn’t need to use this because it uses the max activation method and only the rules that are activated contribute towards the final probability (Section 2.4.6).

![Figure 1: Example ROC Curve and Optimum Threshold](image)

2.2 GitHub

GitHub is a web-based collaborative platform for software development. It has social features like the ability to follow members, rate projects, receive notifications for specific projects, and communicate freely and publicly. It allows the creation of public repositories which can be forked by others to contribute back to the project, or to use for their personal code. A fork is a copy or
clone of the repository. Contributions to the code are made using pull requests, which allow the original creator to view and approve the changes that are requested to the main branch.

All these activities are publicly published along with any content and discussion related to issues, pull requests, and commits. Users can also star certain projects, which is a way to indicate that the project is useful or interesting to them. Stars are one of the ways to distinguish open source projects that provide the same functionality and a project with a higher number of stars is generally more trustworthy and active. Researchers also have access to a project’s metadata available through the API provided by GitHub [developer.github.com] [8].

GitHub is built on top of Git, which is a decentralized version control system. It stores and tracks the complete history of the project and all the revisions that were done to it. It is open-source, free, and fast, which is why it is the most popular version control system for software development.

2.3 Bugzilla

Bugzilla is an open-source bug-tracking system and was built by Mozilla in 1998. The first public release was written in Perl, it continues to receive improvements and is actively developed by contributors. It is used by thousands of companies around the globe and is one of the most trusted defect-tracking systems [9].

It allows teams of developers to keep track of bugs, issues, or change requests more effectively than the native issue tracker available on GitHub. It is a cloud-based system, which means that all the tasks related to identifying and solving issues can be done through a web browser.
2.4 Fuzzy Logic

Fuzzy Logic was introduced in 1967 as a way of allowing computers to understand more human logic. Whereas most traditional methods of logic have standard evaluations like Yes/No or True/False, Fuzzy Logic allows us to define intermediate values between them. Coffee can for example be defined as very hot or very cold unlike the typical discrete value of its temperature. This allows us to form rules which are very human-like in nature. The fuzzy rules are thus written descriptively, and approximate reasoning is applied to them.

Some concepts and terminology related to Fuzzy Logic and Fuzzy Inference are discussed below.

2.4.1 Fuzzy Sets

In a crisp set, an element either belongs to the set or doesn’t. Conversely, Fuzzy sets have elements that are partially in the set. Each element in the set will contain a degree of membership which expresses how much the element belongs to the set. Unlike crisp or classical sets, fuzzy sets do not have a fixed boundary. Its boundary is defined by the membership functions which give a value between 0 and 1 for each element, 0 meaning that the element doesn’t belong to the set at all, and 1 meaning that the element completely belongs to the set. As an example, we can create a model to classify whether the height of a building is high or low. The lower range of low is easy to calculate at 0 meters. However, the upper range can be arbitrary. Assuming the upper range to be 150 meters, a building that is 149 meters would be low while a building that is 150.2 meters would not be. It would be more natural to relax the separation between low and high. This can be done using fuzzy sets by introducing more flexible rules like fairly low instead of using the classical Yes/No decision approach.

2.4.2 Membership Functions

The membership function of an element is a graphical representation of how much the element belongs in the set. Classical Sets can be written as $A = \{x \mid x \in X \text{ and } x \text{ is } \Delta \}$ which means that $A$ is a set of elements $x$ such that it belongs to the universe of discourse and has a certain property. Similarly, a fuzzy set would be written as $A_{\text{Fuzzy}} = \{(x, \mu_A(x)) \mid x \in X, \mu_A(x) \in [0,1] \}$ where $\mu_A(x)$ is the membership function of $x$ in $A$. $\mu_A(x)$ quantifies the degree to which $x$ belongs to $A$. 
Figure 2: Membership Functions for the variable “tip”

Figure-2 shows general membership functions for a variable Tip. The x-axis represents the value of the variable tip, while the y axis represents the degree of membership $\mu(x)$ of the input variable for the sets cheap, average and generous. If the value of Tip is 5, then its degree of membership to the set cheap will be 1. A value of 7 means that Tip only half belongs to cheap. A value of 8 would mean that its degree of membership with cheap will be 0.25 and the set average will also have a value of 0.25.

There are generally three types of membership functions as defined by [24]:

1) Triangular Membership Functions:

   It is represented using three parameters $a,b,c$ as:

   $$triangle(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{b-a}{c-x}, & a \leq x \leq b \\ \frac{c-b}{c-x}, & b \leq x \leq c \\ 0, & x \leq c \end{cases}$$

   An example of a triangular membership function can be seen in Figure 3a).

2) Trapezoid Membership Functions:

   It is represented using four parameters $a,b,c,d$ as:

   $$trapezoid(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a \leq x \leq b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & d \leq c \end{cases}$$

   An example of a trapezoid membership function can be seen in Figure 3b).
3) Gaussian Membership Functions:
It is represented using two parameters $c$ and $\sigma$ where $c$ is the center of the membership function and $\sigma$ is the membership function’s width.

\[
gaussian(x; c, \sigma) = e^{\frac{1}{2} \left( \frac{x-c}{\sigma} \right)^2}
\]  

(2.7)

An example of a gaussian membership function can be seen in Figure 3c).

![Gaussian Membership Function](image)

**Figure 3: Types of Membership Functions**

2.4.3 Linguistic Variables

Linguistic variables are an essential and necessary part of fuzzy set theory. Fuzzy Inference mainly deals with reasoning on vague and unclear terms in a language [25]. These terms are called Linguistic Variables. An example of it would be in the sentence “The humidity in the room is very low”. Here, the linguistic variable would be “Humidity” and the value of it would be “Very Low”.

The universe of discourse for a linguistic variable would be a set of all the discrete values that it can hold. For example, the variable *service* in Figure-4 can have a range of values from 0 to 9 with fuzzy subsets *excellent, poor,* and *good*. The variable *food* also has values ranging from 0 to 9 with the subsets being *rancid* and *delicious*. 
2.4.4 Fuzzy Set Operations

Fuzzy set operations are similar in nature to those of classical sets. Three operations are used: Union, Intersection, and Complement. If we have two fuzzy sets $A$ and $B$, an element $x$, and the universe of discourse $\Delta$, then the operations will have the following relations:

1) Fuzzy OR/ Union:

$$\mu_{A \cup B}(x) = \mu_A \vee \mu_B \quad \forall \ x \in \Delta$$

Here, the operation $\vee$ refers to the max operation and can be graphically observed in Figure 5a).

2) Fuzzy AND/ Intersection:

$$\mu_{A \cap B}(x) = \mu_A \wedge \mu_B \quad \forall \ x \in \Delta$$

Here, the operations $\wedge$ refer to the min operation and can be graphically observed in Figure 5b).

3) Fuzzy NOT/ Complement:

$$\mu_{\overline{A}}(x) = 1 - \mu_A(x) \quad \forall \ x \in \Delta$$

The operation can be graphically observed in Figure 5c).
2.4.5 Fuzzy Conditional Rules

A fuzzy rule is written in the form of a conditional statement as

**IF** X is A  
**THEN** Y is B

Where both X and Y are linguistic variables with A and B as one of their fuzzy subsets. These rules are used to relate fuzzy sets and are executed partially based on the discrete value of the linguistic variable. “X is A” is called the antecedent while “Y is B” is called the consequent. If the antecedent is true to some degree, then the consequent will also be true to the same degree. Some examples of fuzzy rules using linguistic variables and membership functions defined in Figure-2 and Figure-4 are:

- IF service IS excellent AND food IS delicious THEN tip IS high
- IF service IS good AND food IS delicious THEN tip IS avg
2.4.6 Fuzzy Inference

Fuzzy Inference, also known as approximate reasoning, is the process by which the output of a fuzzy process is determined using a predefined set of rules and input vectors. There are mainly two types of inference techniques for Fuzzy Logic: Mamdani-type and Sugeno-type inference [25]. The first part of both techniques is the same. The crisp values of the input variables are converted into membership values based on the appropriate fuzzy subsets. The differences in the techniques appear in the second part, representing the output as a single value.

In Mamdani inference, the output for each rule will be a fuzzy set. All the resultant consequent fuzzy sets are combined to get an output distribution which is defuzzified to get the crisp value of the output variable. In Sugeno’s method, there are no membership functions for the output variables. Instead, the output variable of each rule is represented as a polynomial \( z = px + qy + r \) for a rule with two linguistic variables in the antecedent; where \( p, q, r \) are constants. The resultant values for each rule are weighed and a crisp output value is returned. The problem with Sugeno’s inference is that it is very inefficient to find the values of the constants for each rule. That’s why this thesis focuses on Mamdani’s Inference.

2.4.7 Defuzzification

Defuzzification refers to the process of converting an aggregated fuzzy set into a crisp value for the output variable. Many defuzzification methods exist in literature including Center of Gravity, Mean of Maxima, and Center of Sums. The most popular however is COG (Center of Gravity) and is the one that is used in the thesis. In COG, the membership function distribution which is used to represent the aggregated fuzzy set is divided into several sub-areas. The centroid or center of gravity of each sub-area is calculated and the summation according to the below formula is used to find the discrete or crisp value of the output variable.

\[
x^* = \frac{\sum_{i=1}^{n} x_i \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)}
\]

(2.8)

Where \( x^* \) is the defuzzified value of the output variable, \( x_i \) is a sample element, \( \mu_i \) is the membership function, and \( n \) is the number of elements in the sample set.
2.4.8 Fuzzy Inference and Defuzzification Example

Figure 6: Fuzzy Inference Example

Figure-6 shows an illustrative example of how Fuzzy Inference works and was modified from [10]. The membership functions shown in Figure-2 and Figure-4 are used for the variables service, food, and tip. We use the two rules described in Section 2.4.5 to perform inference. The input values given for service and food are 7 and 8 respectively.

The antecedent of the first rule requires the service to be excellent and the food to be delicious. A value of 7 would mean that service has a degree of membership of about 0.4 with the fuzzy set excellent. Since food has a value of 8, it has a degree of membership of around 0.45 with the set delicious. However, since the operation we are using is and, we take the least value of the two and cut the membership function of the set generous for the output variable tip.

The second rule only has one variable in its antecedent. A value of 7 gives service a degree of membership of 0.8 with the set good. This means that the output variable tip also has a degree of membership of 0.8 with the set average. Once the degree of membership for the output variable is calculated for all the rules, it is aggregated as shown in Figure-6. The final defuzzified value is calculated by taking the center of gravity of the aggregated fuzzy set. In this case, the inference value for tip is 15.35.
2.5 Markov Logic Networks

In general, the techniques in Machine Learning and AI can be classified into two approaches: The Logic-based Approach and Statistical Approach. On the side of the logical approach, there are techniques like First Order Logic, Satisfiability testing, Classical Planning, Inductive Logic Programming, etc. These approaches tend to focus more on the complexity of data. Statistical techniques on the other hand focus on handling uncertainty in data. This includes techniques and models like Neural Networks, Graphical Models, Markov Chain Monte Carlo (MCMC), Bayesian Networks, etc. These two approaches are quite different in execution but are complementary in design. Neither approaches by itself are enough to solve the type of real-world applications that we find in AI. Domingos et al. try to combine the two approaches in Markov Logic Networks so that the model can handle both uncertainty and intricacy.

There have been previous attempts in unifying the two methods. In 1986, Genesereth et al. [11] proposed a framework to utilize the probability of logical sentences using propositional sentences and first-order logic. They introduce Propositional Logic, in which they followed the assumption of probability theory that there is a consistent probability distribution for all states that represent the current knowledge base. If all the probabilities of the formulas are either zero or one, then the probability logic reduces to classical logic. Wellman et al. propose an approach called Knowledge-Based Model Construction [12], in which decision models are converted to a knowledge representation. Taskar et al. introduce Relational Markov Networks in [13] which construct undirected probabilistic models for relational domains. Muggleton et al. introduce stochastic logic programs which give a structured definition to probabilistic distributions over some logic formulae [14].

The key advantages of Markov Logic Networks over the previous models are that it is much simpler in implementation and it comes with a set of learning and inference algorithms that are available through the open-source software Alchemy [15].

2.5.1 Markov Networks

Markov Networks, like Bayesian Networks, is a model of the joint distribution of a set of variables $X = (X_1, X_2, ..., X_n) \in \mathcal{X}$. Each node in the network represents a variable and an undirected edge
between two nodes exist only if there is a direct dependency between them. Graph separation in the network gives information about which variables are independent of each other. The parameters for the joint distribution are given by potential functions \( \phi \), which are defined for each clique of a graph. Clique refers to a completely connected subgraph in the network. The potential function for a clique can hold any non-negative real value. The probability of a state of a function can be given by taking the product of all the normalized potential functions for it.

\[
T(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}})
\]

Table 1: Joint Probability Distribution for Overworking and Stress

<table>
<thead>
<tr>
<th>Overworking</th>
<th>Stress</th>
<th>( \phi(O, S) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>True</td>
<td>3.6</td>
</tr>
<tr>
<td>True</td>
<td>False</td>
<td>1.3</td>
</tr>
<tr>
<td>False</td>
<td>True</td>
<td>3.6</td>
</tr>
<tr>
<td>False</td>
<td>False</td>
<td>3.6</td>
</tr>
</tbody>
</table>

Figure 7: Example Markov Network

The joint probability distribution of a Markov Network is [16]:

\[
P(X = x) = \frac{1}{Z} \prod_k \phi_k(x_{\{k\}})
\]

Where \( Z \) is the partition function with value \( Z = \sum_{x \in X} \prod_k \phi_k(x_{\{k\}}) \), and \( x_{\{k\}} \) is the state of the \( k \)th clique. However, since the size of the potential functions increases exponentially with the number of variables, it's not very convenient for large models. Therefore, Markov Networks are more...
commonly represented as log-linear forms with the potential function being replaced by a weighted sum of features.

\[ P(X = x) = \frac{1}{Z} e^{\sum_j w_j f_j(x)} \]  

(2.10)

Here, a feature is defined for every state of the clique and the weight is the log of its clique potential. The Joint Probability Distribution described in Table-1 can be represented as a log-linear form by using the function

\[ f_1(\text{Overworking}, \text{Stress}) = \begin{cases} 
1 & \text{if } \neg\text{Overworking} \lor \text{Stress} \\
0 & \text{otherwise}
\end{cases} \]

With \( w_1 \) being 1.28.

2.5.2 First-Order Logic

In this section, we will define all the common terms and symbols related to first-order logic (FOL) which will be used in subsequent sections. Formulae in FOL are made up of logical connectives like quantifiers, conjunction, disjunction, etc. The symbols in FOL are generally classified into logical symbols and non-logical symbols. The more common logical symbols are conjunction \((X \land Y)\), which is true only if both \(X\) and \(Y\) are true; disjunction \((X \lor Y)\), which is only true only if either \(X\) or \(Y\) is true; negation \((\neg X)\), which is true only if \(X\) is false; implication \((X \rightarrow Y)\), which is false only if \(X\) is false and \(Y\) is true; biconditional \((X \leftrightarrow Y)\), which is true only if \(X\) and \(Y\) have the same truth values; universal quantification \((\forall x P(x))\), which is true only if \(P(x)\) is true for every object \(x\) in the domain; existential quantification \((\exists x P(x))\), which is true only if \(P(x)\) is true for at least one object \(x\) in the domain.

The non-logical symbols are predicates, functions, constants, and variables. A predicate/relation takes one or more arguments and is either true or false. They are relations between elements of the domain (e.g., \(\text{publishedBy}(book, publisher)\)). Functions also take one or more arguments, but unlike predicates, they return a value. They are used to map a group of objects to an object (e.g., \(\text{directorOf}\)). A constant can be thought of as a 0nary operation that takes zero inputs and gives an output. They represent objects in a domain and are generally written in lowercase. Variables are also a 0nary operation, but they can only range over the objects of the corresponding type. They are generally denoted by a string that starts with an uppercase letter.
The *grounding* of a predicate or a formula is the result of replacing all the variables with constants. A grounded variable is a Boolean variable that is true if the predicate or formula correctly describes the constants. A *world* is an assignment of truth values to all the grounded predicates. A formula is said to be *satisfiable* if there exists a world where it is true. A *conjunctive normal form* (CNF; also known as clausal form) is a conjunction of one or more clauses, where a clause is the disjunction of one or more literals.

As an example, consider a Teacher-Student database that has three predicates and two formulae. $\text{Teacher}(X)$ means that $X$ is a teacher, $\text{Student}(Y)$ means $Y$ is a student, and $\text{tutors}(X, Y)$ means that $X$ tutors $Y$. $X$ and $Y$ here are variables that range over the objects in the type $\text{Person}$. The two formulae that can be used in conjunction with the given predicates are:

$$\text{Teacher}(X) \Rightarrow \neg\text{Student}(X)$$

$$\text{tutors}(X, Y) \Rightarrow \text{Student}(Y) \land \text{Teacher}(X)$$

The first rule states that if $X$ is a teacher, then $X$ cannot be a student. The second rule states that if $X$ tutors $Y$, then $X$ is a teacher, and $Y$ is a student.

### 2.5.3 Markov Logic Networks

A Logical Knowledge Base (KB) can be represented as a set of hard constraints on all the possible worlds. This means that if one of the worlds violates just one grounding of one formula, then the world becomes impossible. This is the reason for the brittleness of first-order logic or logic in general. It doesn’t separate a world that violates one formula from a world that violates all of them.

In Markov Logic Networks (MLN), the formulae are converted to soft constraints so that when a world violates a formula, instead of it becoming impossible, it just becomes less probable.

MLN associates each formula with a weight that represents how strong of a constraint it is. If the formula is known to be quite accurate, then it can be given a higher weight, which in turn will give out a bigger penalty to the world that violates it. So, the more formulae it satisfies and the higher its weight, the more probable is the world. This leads to the equation:

$$P(\text{world}) \propto e^{\sum \text{weights of the formulae that the world satisfies}} \tag{2.11}$$
Domingos et al. defined MLN in [16] as:

**Definition 2.5.3.1.** [16] A Markov Logic Network is a set of pairs \((F_k, w_k)\) where \(F_k\) is a formula defined in first-order logic and \(w_k\) is its corresponding weight. Along with a set of constants, the MLN defines a Markov Network as follows:

1. There is a node for every grounding of every predicate defined in the MLN.
2. There is a feature for every grounding of every formula in the MLN. The feature has a value of 1 if the grounding is true, and 0 otherwise. The feature will have the weight \(w_k\) for formula \(F_k\).

An MLN is a template for grounded Markov Networks. Its distribution can be written as:

\[
P(X = x) = \frac{1}{Z} e^{\sum_k w_k n_k(k)}
\]  

[16](2.12)

Where \(n_k(x)\) is the number of true groundings that the formula \(F_k\) has in \(x\). The MLN does not represent just one distribution of the Markov Network. It represents a family of ground distributions depending on the domain that it is applied. A Markov Network with a lot of constants is exponentially larger than one with fewer constants. However, they can both be said to be repetitions of the same template. MLN defines the template that is repeated for each set of constants. To perform inference, the MLN is grounded to its corresponding Markov Network. To reduce the size of the potential Markov Network, MLN uses typed constants and variables. The variables are only grounded to the constants of the same type.

An MLN can be converted to first-order logic by giving every formula infinite weight, thus converting the soft constraints to hard constraints. Every satisfiable world in first-order logic will also be satisfiable in an MLN with positive weights. However, these worlds will form the peaks in the distribution and are smoother than the distribution given by FOL. Finally, MLN allows for contradictory knowledge, unlike FOL. If a KB in FOL has a contradiction, then all the formulas will follow from it, which makes it very difficult to build large KBs.

For the example defined in 5.2.2, the FOL rules can be converted to MLN rules by adding weights to them.
Chapter 2: Background

<table>
<thead>
<tr>
<th>Weight</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.743</td>
<td>( Teacher(X) \Rightarrow \neg Student(X) )</td>
</tr>
<tr>
<td>1.12</td>
<td>( tutors(X,Y) \Rightarrow Student(Y) \land Teacher(X) )</td>
</tr>
</tbody>
</table>

Table 2: Example MLN Rules

The MLN rules can be used as a template for grounded Markov Networks. Assume that in the example above, there are two constants Mary and John. The grounded Markov Network for the constants is shown in Figure-8.

![Figure 8: Example Grounded Markov Network](image)

2.5.4 Inference

A basic task in inferencing is to find the probability of a formula given some evidence. This is also called marginal and conditional probability. MLN can answer these kinds of queries using equation 5.5. To get the probability of a formula \( F_1 \) given that \( F_2 \) holds, if \( F_1 \) and \( F_2 \) are formulas in FOL, \( C \) is a set of constants and \( L \) is an MLN, we can use the equation:

\[
P(F_1|F_2, L, C) = P(F_1|F_2, M_{L,C}) = \frac{P(F_1 \land F_2|M_{L,C})}{P(F_2|M_{L,C})} = \frac{\sum_{x \in \chi_{F_1} \cap \chi_{F_2}} P(X = x|M_{L,C})}{\sum_{x \in \chi_{F_2}} P(X = x|M_{L,C})}
\]

[16](2.13)
where $\chi_{F_i}$ are the worlds in which $F_i$ holds. Calculating the result of the equation will only be feasible for small domains. However, since probabilistic inference is an \#p-complete problem and logical inference is an \#np-problem, it is inevitable that the inference here is not very efficient.

The default inference algorithm used in Alchemy, however, is Lifted Belief Propagation. The problem with most inference algorithms is that they are purely probabilistic in nature. They create a large ground network and perform inference on it. First-Order Logic, on the other hand, allows inference to be performed without materializing all the atoms in the domain. This is called Lifted Inference and it is a lot more efficient than proportionalized inference. Domingos et al. built upon the work done by Jaimovich et al. [17] to bring lifted inference to probabilistic logic in Lifted Belief Propagation [18].

The atoms and the clauses in the network are grouped together into indistinguishable sets. A SuperNode is a set of ground atoms that send and receive the same messages, while a SuperFeature is a set of ground clauses that send and receive the same messages. By grouping the atoms and clauses, the total number of connections can be exponentially reduced, which leads to a faster inference time. The SuperNodes and SuperFeatures constitute a lifted network, and belief propagation is run on it with two changes:

1. The message sent from a SuperNode $x$ to SuperFeature $f$ becomes a product of the nodes within it.
   \[
   \mu_{x\rightarrow f}(x) = \prod_{h \in h_n(x)\{f\}} \mu_{h\rightarrow x}(x) \quad \text{[18](2.14)}
   \]

2. The marginal value of each SuperNode is
   \[
   \prod_{h \in h_{nb}(x)} \mu_{h\rightarrow x}^{n(h,x)}(x) \quad \text{[18](2.15)}
   \]

2.5.5 Weight Learning

The weights of formulas in an MLN can be learned automatically using a training database. Domingos et al. introduce two types of weight learning algorithms: generative and discriminative
learning. The default algorithm used in Alchemy is discriminative learning and is the one used in the thesis. Therefore, only that algorithm will be discussed in the current section.

The idea behind discriminative learning is that if the system knows in advance which variables are going to be queried \( y \) and which variables are going to be evidence, then the conditional likelihood of \( y \) given \( x \) can be calculated by:

\[
P(y|x) = \frac{1}{Z_x} e^{\Sigma_{i \in F_y} w_i n_i(x,y)}
\]

or

\[
P(y|x) = \frac{1}{Z_x} e^{\Sigma_{j \in G_y} w_j g_j(x,y)}
\] \hspace{1cm} \text{(2.16)}

where \( F_y \) is the set of clauses with at least grounding involving a query atom, \( G_y \) is the set of ground clauses in \( M_{L,C} \) involving query atoms, \( n_i(x,y) \) is the number of groundings of the \( i \)th clause that involves the query atoms and \( g_j(x,y) \) is equal to 1 if the \( j \)th clause is true and 0 otherwise.

The likelihood as a function of weights is a concave function, so it only has a single local optimum. Gradient descent is used to find the maximum likelihood and its equation is

\[
\frac{\partial}{\partial w_i} log P_w(y|x) = n_i(x,y) - E_w[n_i(x,y)]
\] \hspace{1cm} \text{(2.17)}

where \( E_w \) is the number of true groundings predicted by the MLN. If the predicted number is less than the actual number, then the weight of the formula needs to up and vice versa. Once the predictions and the counts are equal, the maximum likelihood point is reached. One of the problems with this approach is that every step requires inference, but since we only use the query and evidence variables, some of the efforts are mitigated. To make the algorithm more efficient, Domingos et al. propose using approximations instead of full conditional inference. The count for \( E_w \) can be approximated by the counts \( n_i(x, y^*_w) \) in the MAP state \( y^*_w \).
Chapter 3: Related Work

3.1 Machine Learning

Machine Learning helps build models that automatically learn through experience and time. The research and usage of Machine Learning and Artificial Intelligence has progressed dramatically over the last few decades. It has evolved from being a laboratory curiosity to one of the most widely used industrial technologies in fields like Image Recognition, Speech Recognition, Medical Diagnosis, Classification, etc. [19] [20]. Depending on how the learning process is structured, it can be divided into three categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning [21]. In Supervised Learning, we predict a target variable based on a set of independent predictors. Unsupervised Learning is commonly used to find clusters or patterns in a large dataset; while Reinforcement Learning is a way of training computer agents by providing a reward and punishment without specifying how the task needs to be completed [22].

The most common techniques used for classifying datasets in Supervised Learning are Regression, Decision Tree, Linear Support Vector Machines, Random Forest, KNN, and Logistic Regression.

**Decision Trees:** These trees classify instances of data based on the values their feature vectors hold. Each node in the decision tree corresponds to a feature, while each branch corresponds to a value that the feature can hold. Each leaf node represents a class that the instance can be classified into [23]. Chen et al. use Decision Trees to diagnose failures in network systems by ranking the paths by their correlation with failure and merging nodes that are subsumed by their successor nodes [24]. Al-Radaideh et. al. use the CRISP framework for mining students’ data in university courses and decision trees to evaluate the main attributes that contribute to their performance in a course [25].

**Regression:** Linear Regression is a technique borrowed from Statistics that is used to find the relationship between input variables and an output variable [26]. It has been around for more than 200 years and therefore has been studied very extensively. Feigelson et. al. study the use of ordinary least squares to calculate cosmic distance scale in the field of Astronomy [27]. Naseem et. al. use linear regression in the field of Facial Recognition by transforming 2-dimensional
pictures into lower-dimensional vectors using downscaling, which then serve as the basis for linear regression [28].

**Linear Support Vector Machines:** Unlike most other learning techniques which try to reduce the error on the training set, SVM tries to reduce the risk of error on the data. It works very well on data that is less structured [29] [30]. It tries to draw a line, or a hyperplane between two classes of data by maximizing the margin between them. The data points nearest to the boundary are called support vectors and they contain everything required to define the classifier. Jack et. al. use SVMs for fault localization in modern rotating machinery by using vibration data taken from two sources and genetic algorithms to select relevant features in [31]. They compare the results with Neural Networks and find that SVMs have a very high success rate in training data but do not generalize very well.

**Random Forest:** It is a collection of decision trees where each tree is independent of the other. Each tree will classify the instance, and the class which is selected the most is chosen to be the right classification. It has a much higher success rate than a Decision Tree because the other trees overwhelm the errors that an individual tree makes [32]. Pal et. al. compared Random Forest and SVM to classify and identify land cover types. They found that the Random Forest has similar accuracy to SVM whilst using fewer parameters than it [33]. Ahmad et. al. used RF in [34] to identify the occurrence of Lymph disease based on symptoms fed to the model. They initially used several techniques including genetic algorithms and Sequential Forward Floating Search to reduce the dimensionality of the dataset. Then the reduced feature sets were given to the model to classify for the disease. The model received an impressive accuracy of 92%, which is a significant improvement over previous work on this classification.

**k-Nearest Neighbors:** This algorithm relies on the simple assumption that similar things appear close to each other. For every data point, it checks the k closest neighbors and chooses the most common class among them for it. The value of k is normally the value that gives the least error. Zhang et. al. uses KNN in [35] for multi-label classification on Yeast gene data. Once the classes of the neighbors are identified, the maximum a priori algorithm is used to select the classes that the sample should have. The proposed technique was shown to be comparable to existing multi-label classification algorithms. Samanthula et. al. uses KNN to classify encrypted data that resides in third-party cloud systems [36]. To ensure user’s privacy, the data is not decrypted at any stage.
of the classification process. The proposed protocol was not very computationally efficient, but some improvements were suggested to decrease the computation time.

**Logistic Regression:** Unlike Linear Regression, which is mainly used when the dependent variable is continuous in nature, logistic regression can classify categorical response variables. It gives the probability of a sample belonging to a certain class [37]. Subasi et. al. used Logistic Regression and Artificial Neural Networks to predict the onset of epilepsy using a patient’s EEG signals [38]. Since an EEG is non-stationary, they used wavelet transformation to represent the signals as features for the models. Logistic Regression was found to give a 90.3 specificity and an 89.2 sensitivity. Cheng et. al. attempts to use Logistic Regression for both feature selection and classification of remote sensing data [39]. Unlike most conventional algorithms for feature selection, LR can be used with fewer assumptions and for features that are not linearly related to the classes. They compared the results with work done by Habema et. al in the same domain using Stepwise Linear Discriminant Analysis [40] and found that LR can be used to get comparative accuracy results while using fewer features.

### 3.2 Fuzzy Logic

Fuzzy Logic is generally considered to be a branch of Artificial Intelligence in which the Intelligence is supplied by Humans in the form of rules and criteria. Instead of the typical Boolean classification, fuzzy logic was one of the first models to support a *degree of membership* for variables which allows for more flexible and robust classification. Another advantage is that the rules in Fuzzy Logic are very readable, so they can easily be adapted and modified by domain experts. It incorporates rules that are in the form of simple IF-ELSE statements instead of adapting a complex model mathematically. It has been used in a plethora of applications in fields including stock trading, viticulture, oceanography, facial recognition, knowledge-based systems, medical diagnosis, robotics, among others [41].

Nedeljkovic et al. use fuzzy logic in image classification to classify a given image into different land cover masses [42]. They use fuzzy logic over other supervised learning techniques to reduce the bias that is inherently a part of most supervised learning techniques. The membership functions were defined based on the results of the supervised classification and the rules were created using MATLAB Fuzzy Logic Toolbox.
Optical properties are derived in Ocean color remote sensing using bio-optical algorithms. However, there is a problem where different algorithms need to be used for different types of water. Using a hard classification to decide which algorithm needs to be used often results in a patchwork quilt effect where there are sudden discontinuities between continuous image data. Moore et al. attempt to solve the problem by classifying pixels in ocean color satellite images into partial membership functions for different water types and possibly blending the algorithms together using fuzzy logic [43].

Das et al. use a fuzzy-logic-based fault classification system in [44] to identify faults in a transmission line. The faults identified can be caused under different fault resistances, loading levels, and inception angles. Fuzzy Logic is used over the traditional approaches using Artificial Neural Network or Fuzzy-Neural Networks because it doesn’t need expensive training nor expert knowledge. The model proposed has over 97% accuracy for ten different types of faults under various operating conditions and can be considered a vast improvement over other existing models.

Nilashi et al. propose a Fuzzy Logic classification approach in the field of medical diagnosis to predict the occurrence of breast cancer [45]. They first use Expectation Maximization to cluster the data into similar groups. Principal Component Analysis is then used on the clusters to prevent the multi-collinearity issue. Classification and Regression trees are used to generate dynamic fuzzy rules which can be in a knowledge system to classify the data. The model outperformed existing classification models with an accuracy of over 0.93 in all the datasets that it was tested on.

Marzano et al. classify hydrometeors, which have applications in detecting storms, rain-clouds microphysics investigation, and tuning precipitation rate algorithms, using fuzzy logic and C-band polarimetric radar data [46]. Most current techniques at the time focused on classifying hydrometeors using S-band dual-polarized radars. The authors use a supervised model-based approach in classifying the data. The fuzzy-logic-based model is supervised by a scattering model.

Petropoulosa et al. propose a fuzzy logic-based classification model in [47] that classifies whether the wine observed is high quality or not. The traditional methods use data from lab testing which are difficult to get and hard to implement in real-world applications. Therefore, the authors use multiple simple grape parameters, which can be easily calculated, to classify the wines. The results of the model were compared with sensory data from wine experts to test the performance of the model. The fuzzy system was modeled by oenologist experts and ranked according to the
importance of certain parameters in the literature. The model gave results similar to the ones given by the experts with the inclusion of a few exceptions.

3.3 Markov Logic Networks

A Markov Logic Network can be thought of as a set of first-order logic formulae and a weight associated with each formula. A formula in MLN is a relational rule providing relationships between different predicates. Unlike traditional First Order Logic, a world doesn’t get unsatisfiable because of the violation of a rule. Instead, the world is treated as being less likely to occur at a magnitude depending on the weight of the violated rule. Weights are learned from a training dataset either generatively or discriminatively. Approximate inference can be performed on the set of rules with the learned weights and a set of evidence literals.

He et al. use MLN in the field of text sentiment analysis to overcome some of the shortcomings that supervised learning faces [48]. Supervised Machine Learning models have a strong domain dependency which leads to lower precision when the model’s knowledge in the domain is insufficient. They create a cross-domain classification model which can transfer known knowledge from other domains into one model. Random Sampling was performed to test the model on 400 bad comments and 200 good comments. They concluded that the multi-task transfer learning method outperforms a single-task learning method and has good noise resistance.

Crane et al. test the efficiency and performance of Markov Logic Networks for Collective Classification (CC) [49]. CC is the process of jointly assigning labels to a collection of interconnected nodes. They test the model on four standard datasets and perform an n-fold cross-validation study against three traditional CC algorithms: Gibbs Sampling, Cautious ICA, and ICA [50] [51] [52]. They observed that while MLNs performed well for complex data, they generally failed to give accurate predictions on data with simple patterns. They theorize that the problems with MLNs may have more to do with learning than inference.

Dierkes et al. predict the potential churners for a mobile phone company in [53]. Churners refer to customers or subscribers who leave the service during a specific time period. The influence of Word of Mouth (WOM) in the decision of a customer to leave the company is measured using MLNs. WOM is modeled using a customer’s anonymized call records and uses the intensity of the
communication as an observable variable. The MLN model based on WOM was found to have better predictive accuracy and sensitivity than logistic regression on the same data.

Semantic Role Labelling normally consists of a three-stage pipeline: finding predicates and their senses, identifying possible arguments, and assigning argument labels to the candidates. Riedel et al. combine these three stages into a joint SRL system and perform all the steps collectively [54]. MLN is used to build the system because it can integrate the dependencies between the results of different stages in the pipeline and the availability of its open-source implementation. The system achieves a semantic F1 score of 76% but they show that the results can be improved by building an MLN that is similar to the traditional bottom-up SRL model.

Mihalkova et al. use MLN to suggest a potential collaborator to a user working on an online project [55]. They discuss the problem statement in the context of Wikipedia editors where editors are said to be collaborators if they contribute to the same article. If a user is working on several new projects, then for each project, a set of collaborators are suggested based on their relational data. The sets for projects and users can be represented as annotated nodes in a graph G with labeled edges signifying different relationships. To handle the problem of scalability, the graph is broken into pieces with each piece being centered around one user. The relational features are represented as MLN formulae and the weights are learned. The model was tested on data from the Wikipedia encyclopedia.

3.4 Bug Prediction

A bug prediction model typically classifies fault-prone modules in the following levels of granularity:

*Package Level:* This is the lowest level of granularity that a model can classify. In this, a package that is used in a project is classified as being buggy or not. Most modern studies have moved away from this level but there are still a few case studies for models that classify packages.

*Class Level:* It is one of the most common modules used in bug prediction algorithms that operate exclusively on object-oriented languages. In it, classes are classified as either being fault-prone or not.
Chapter 3: Related Work

File Level: This is the level of granularity found in this thesis and is also the level that most modern bug prediction algorithms consider. The model classifies an entire file as being buggy or clean.

Method Level: It is the highest level of granularity that has been studied and experimented on. Instead of an entire file being the target for classification, only a method or a function in the file is classified. The availability of metrics is a major challenge in this domain but most studies that have worked on it have found ways to circumvent these problems.

Related Work on each level is discussed below.

3.4.1 Package Level

Nagappan et al. combine the use of past failure history from bug databases along with code complexity metrics to predict faulty Windows components (in this case individual binaries) in [56]. Five major commercial Microsoft projects were mined to get the past faulty entities. Principal Component Analysis was used to select the best combination of metrics which were fed into multiple regression models along with the past defect data. They also experimented with using the metrics collected from failure data of one project to predict defects in another project, including projects with no past failure history. The $R^2$ values of the predictor ranged from 0.416 to 0.882 for the five projects. They also concluded that using metrics collected from one project in a different project works only if the two projects are cross-dependent.

Schroter et al. test the hypothesis that some problem domains are more fault-prone than others [57]. They assume that the problem domain of a project is defined by the set of components that are used. Since the components to be used are decided upon in the design phase, managers can allocate appropriate resources to projects which are likely to have more defects in the future. For every project, they collected data on post defect failures, the files that are associated with the failures, and the import statements in the files. Four Machine Learning Models: Linear Regression, Ridge Regression, Regression Trees, and Support Vector Machines (SVM) were used to predict the faulty components. 52 Eclipse plugins were tested to prove the validity of the hypothesis and SVM gave the best predictive power with an average recall of package-level prediction of 0.74.
While they predict which (problem) domains are more fault-prone, they do not give reasons on what makes those domains difficult to maintain and left it to future work.

3.4.2 Class Level

Singh et al. introduce two new object-oriented metrics that would help in identifying classes and classes that have a bad smell [58]. The first metric, Public Factor, is a score between 0 and 1 that measures how visible the methods and attributes of a class are to other classes. The second metric, the Encapsulation Factor, is a measure that provides the relationship between cohesion and data visibility of a class. They used a binary regression model on Mozilla data to calculate how well the new metrics perform compared to other metrics like coupling, polymorphism, inheritance, etc. They found that the metrics work well to predict faulty classes and increase the AUC for the categorical model to predict smelly classes by 1% to 30%.

Huang et al. use Multi Instance Learning (MIL) in [59] to solve the problem of the assumption that most supervised techniques employ i.e. the logical levels of the training set have to be the same for the training set and the validation set. The problem with this is that the quantity of the available training metrics can be quite limited for high-level modules. Also, the software metrics for the modules at a high level can be scarce and, by treating a high-level unit as a basic learning unit, some relational information would be ignored. MIL uses a bags and instances approach with classes being bags and its attributes and methods being instances. Four MIL algorithms were used to create the model (Statistical Kernel, Set Kernel, Citation KNN, MI EM-DD). A mission control project from NASA was used to validate the model, the results showed that the MIL approach outperformed the existing supervised learning techniques in accuracy, Type-I, and Type-II error ratios.

Pandey et al. use Fuzzy Inferencing to predict buggy modules in [60]. They use a decision tree, which has test metrics as nodes and its class in the leaf node, to extract the rules which are fed to a Fuzzy Inference System. They have five linguistic categories for each software metric: very low, low, medium, high, and very high. They categorize the modules as faulty and non-faulty and rank the faulty modules based on their degree of fault proneness. The model was tested on KC2 project data from NASA and they got a classifier accuracy of 87.37%, which is better than the referenced models.
Marcus et al. in [61] introduce a new metric for Class Cohesion which they name as Conceptual Cohesion of Classes (C3). C3 analyzes textual information like comments and identifiers in a class to build a cohesion measure of the class with respect to the rest of the system. They use Latent Semantic Indexing to analyze the text information from the code. To measure the performance of the metric, they combined C3 with 10 other structural and conceptual cohesion metrics and used multivariate logistic regression with 45 combinations of the metrics. C3 appeared in the top five performing models with an average precision of 64.5. It also had a high degree of completeness and correctness when used with other metrics.

Puranik et al. sought to make a model that does not use too many metrics but still performs better than comparable complex bug prediction models [62]. They opt to use Marginal R square values, to measure the error proneness index, instead of the classical regression method because regression tends to find the best approximation for the training, and it doesn't always translate well to the validation set. Simple Linear Regression is used on each metric separately with the bug count being the dependent variable. The top n metrics are selected based on their coefficient of determination. Then linear multiple regression is used n times to compute the marginal R square values of each metric. They test the model on an Eclipse JDT core and found that their model outperforms comparable models in Precision and F1 score.

Singh et al. propose a Clustering Based Classification (CBC) model that splits the data using entropy and uses binary discretization for preprocessing the dataset [63]. Static Code attributes including LOC, McCabe complexity, and other Halsted attributes are used to predict the faulty modules. The model was tested on seven projects from the NASA MDP repository. They find that the time efficiency of the CBC model is much better compared to other defect prediction models. It also achieves a probability of detection of 83% with a 40% false alarm rate.

Sandhu et al. propose a decision tree-based classifier model which uses k means clustering technique for preprocessing [64]. Requirement metrics, code metrics, and a combination of them are used to evaluate whether metrics collected early in the life cycle can help increase the accuracy of defect prediction models. The defect dataset is taken from the MDP repository from NASA. The data is first separated based on whether it’s defective or not. Then K Means Clustering is applied to it which divides the dataset into various clusters. C4.5 Decision Tree algorithm is used to predict the faulty modules and results are fed into a confusion matrix. They get a class recall of
100% and a class precision of 100% which means that all the faulty modules were predicted correctly. However, since the size of the dataset tested was comparatively smaller due to the limited availability of requirement metrics, it remains to be seen how the model scales up to more complex datasets.

Palomba et al. [65] introduce a new metric to calculate the intensity of code smell. A code smell detector, JCodeOdor, is used to detect and classify smelly code in a class. The intensity index, a measure with a value between 1 and 10, is an approximation of the extent of design flaws in the specified source code. This metric is used in addition to other simple predictors like structural metrics, entropy, etc. in a Simple Logistic probability model. The results indicate that the addition of the metric increases the F1 measure by 7-21%. When the code smell metrics were combined with product metrics and process metrics, the model outperformed all the other referenced models suggesting that this metric is very useful when used in combination with other existing metrics in literature.

Jin et al. propose a defect prediction model in [66] using Artificial Neural Networks (ANN). ANN is chosen because it is better at being able to model non-linear function relationships. To reduce the dimensionality of the metrics, Quantum Particle Swarm Optimization (QPSO) is chosen, primarily because it only has one control parameter and is thus easier to control. The output of QPSO is given to a multi-layer ANN to get a hybrid model for predicting buggy classes. The model was tested on four NASA projects and compared with 25 other machine learning models. AUC was calculated for the datasets and QPSO+ANN outperformed all the models in three datasets and was only marginally lower than Least Square SVM on the fourth one.

3.4.3 File Level

Moeyersoms et al. attempt to solve the issue that most rule-based classification systems face i.e. the trade-off between comprehensibility and performance [5]. In most cases, models having high comprehensibility will have better model acceptance but have to sacrifice predictive performance. They use ALPA to find a ruleset that mimics the performance of a complex black-box model that classifies using Random Forest and Support Vector Regression. They find that the results of the model using the extracted rules give results comparable to complex models whilst being more comprehensible.
Kamei et al. in [67] predict the occurrence of fault-prone modules by using a hybrid approach between rule-based classification and machine learning. They select rules based on their "interestingness" by using a variety of factors like support, confidence, and lift and then use them to classify modules. If a module does not satisfy any of the rules, then they use logistic regression to classify it instead. They used a module dataset from Eclipse to test their approach and found that using lift to categorize rules increased the F1 score by about 0.163 compared to other fault proneness models.

Ostrand et al. [68] test the idea of using information about individual developers to predict the existence of faults in future releases by the same developer. The metric is appended to an existing model based on Negative Binomial Regression which uses the metrics: file size, number of releases that the file has been a part of, the changes that the file has participated in, the number of faults found previously, and the programming language. They predict the number of faults that each file can have and take the 20% of worst performing files as the faulty files. They find that these files account for 75% of the total faults. However, their results indicated that adding the metric about individual developers only improves the results by about 1% and wasn't a very useful measure to predict the fault proneness of a file.

Wang et al. [69] study the relevance of semantic features in code snippets and how they could be an important metric when predicting faulty files. They theorize that since two code snippets can have identical statistical metrics while having different semantic information, any model built on semantic tokens will have a better defect catch rate. A Deep Belief Network (DBN) was used to represent the semantic information of source code into features for the fault prediction model. For the file level, the Abstract Syntax Tree of the source code was converted into DBN features and for the change level, the features were generated using tokens extracted from code changes. They tested the model on 10 open source projects and found that the file level defect prediction improves the F1 score of existing models by an average of 13.3% while code level defect prediction, which was tested on 1 million changes from six open-source projects and four commercial projects, was improved by an average of 5.1%.

Similar to [68], Jiang et al. attempt to use individual developer's coding styles and history to improve a defect prediction model in [70]. However, unlike [68], Jiang et al. build separate prediction models for each developer. They call this Personalized Change Classification (PCC)
and they pick the model which gives the highest confidence measure for each change. The classifiers used for this are the same as traditional classifiers: Alternating Decision Tree [71], Naive Bayes [72], and Logistic Regression [37]. They also combine PCC and Traditional Change Classification (CC) to create PCC+. Here, they use weighted PCC which uses a training set that is divided into two halves. One half is a collection of changes of an individual developer, and the other half are changes by the rest of the developers. In PCC+, they pick the prediction which has the highest confidence measure among PCC, CC, and weighted PCC. They tested the models on six open-source projects and found that both PCC and PCC+ marginally improve the F1 score by 0.06-0.08 and drastically improve the cost-effectiveness of the prediction model by 19 to 155 bugs.

Bettenburg et al. in [73] build upon the research conducted by Menzies et al. in [74] on the benefits of classifying faulty files by using local models over global models using linear regression. For the global model, they divide the dataset into 90% training and 10% testing and then use linear regression to build the defect prediction model. However, for the local model, the training data is divided into clusters based on their properties. They use a state-of-the-art clustering algorithm MCLUST to partition the dataset. Then they create individual linear regression models for each cluster in the training dataset. For every input in the testing set, they choose the cluster that is most similar to it and use that cluster's model to classify it. They tested both models on four different datasets from the PROMISE repository and found that using local models leads to an increased fit, which is beneficial for data modeling. However, the improvements in data prediction are very low and there is a higher error variance for it compared to global models. For practical applications, they recommend using a hybrid model like MARS [75] which combines global and local models and gives consistent results.

Giger et al. in [76] use Fine-Grained Source Code Changes (SCC) [77] as a replacement for the more popular metric, Code Churn. SCC gives semantic data about the code that was changed, and they theorize that it is a much better indicator for fault proneness than Lines Modified (LM). SCC is calculated by taking the difference of the abstract tree of the two versions of the file. They found that the Spearman Rank Correlation between SCC and the number of bugs to be 0.77, so there is a very strong correlation between them. Additionally, they established that a Logistic Regression Model built on SCC over eclipse datasets would predict faulty files over non-faulty files with an average probability of 90%.
Lee et al. in [78] attempt to use developer behavioral metrics on top of source code metrics (CM) and repository metrics (RM) to improve traditional defect prediction models. They use a plugin for eclipse called Mylyn to record each developer's tasks like selecting or editing files. Mylyn stores all information in an XML format and attaches it to the bug report. The authors analyze this data to create 56 Micro Interaction Metrics (MIM) like time spent on a task, the number of files edited more than once in a week, the number of tasks worked on, average time interval between editing, etc. They use linear regression with a combination of MIM, CM, and RM to classify defective files. The model was tested on Eclipse projects and it was found that using MIM increases the F1 score by 0.23 and reduces Mean Square Error by 0.05. They conclude that the developer's interactions with the files, albeit hard to get, can be a very useful metric for defect classification.

Kim et al. in [79] try to classify code changes in a file as buggy and non-buggy so that developers get immediate feedback after a commit. A new technique called Change Classification is introduced which uses Support Vector Machine as the classifier and they get repository data for their dataset from the Software Configuration System of 12 open source projects. Bug-introducing data is used instead of bug-fixing data, which they theorize is more accurate for training. The features are extracted from the source code using a Bag-of-Words model so that the source code is treated as a text file, which enables the model to be compatible with multiple programming languages. The results show a high accuracy with an average of 78% with recall ranging from 43% to 93%.

Pan et al. argue that traditional code metrics [80] like lines of code, number of functions, cyclomatic complexity, etc. only capture information about source code on a coarse-grained scale. They introduce 13 new metrics based on program slices of C++ functions. Since the metrics are calculated on the scale of a program slice, it gives fine-grained properties of the code. The Program Slicing metrics at a function level include the number of slices, the number of vertices in the function's PDG, the number of edges in the function's PDG, the function's edges to vertices ratio among others. A 10-fold cross-validation method with Bayesian Network Classifier is used to classify buggy functions and files. Apache HTTP project and Latex2rtf project are used to test the model and it was observed that the program slicing metrics outperform traditional code metrics in
accuracy, precision, and recall. The authors state that while the performance of the metrics is very good, it is very time-consuming to compute.

3.4.4 Method Level

Elish et al. in [81] use Support Vector Machines, instead of the classical binary prediction models, to identify buggy functions or methods. They use four mission-critical projects from NASA to test the model and compare the results to eight statistical and machine learning models. They find that SVM has a better F1 score than 5 of the models. It has lower precision than most models but has a higher recall than all of them. They state that the high recall has a lot of implications in software testing as it reduces the risk of a fault not getting detected.

Kim et al. in [82] use the idea of cache in software bug prediction. They theorize that if a file introduces a fault, then it will continue to create faults and the related entities will also introduce faults shortly. They create a cache of entities called FixCache that automatically updates its list every time a fix is committed. It identifies the change that was updated and gets the entities that surrounded the file that originally introduced the bug. Every time a file or method that was present in the cache was buggy, they increased the hit rate of the cache. By using a 10% cache, they found a 73-95% hit rate for file-level entities and a 46-72% hit rate for method-level entities.

Mizuno et al. introduce the idea of using a spam-based classification approach to defect prediction in [83]. Since the spam filtering techniques are quite advanced due to a need for detecting spam emails, these techniques can directly be used by taking the source code as a text file and classifying it as either faulty or non-faulty. The spam filtering software used is CRM114 and they use three classification techniques built into it: Sparse Binary Polynomial Hash Markov model (SBPH), Orthogonal Sparse Bigrams Markov model (OSB), and Simple Bayesian model. The key advantage of using this technique is that there is no need for costly extraction and definition of software metrics which most mathematical models require. Instead, the model can directly be trained on a project and used in real-time with minimal effort. Two open-source projects were selected to test the model and SBPH got the best accuracy with an average of 77.5% along with a decent recall and precision. OSB has better recall but worse accuracy than SBPH. Meanwhile, BAYES has the best recall among the three, but it gives very bad accuracy and precision, so it is not considered optimal.
Menzies et al. [84] argue that the number of static metrics used by a model does not matter, but the way it uses them is important. The authors use a Naive Bayes data miner to classify the data and compare it with other known works [85] [86] [87] [88] [89]. Because Bayesian methods poll various Gaussian approximations of the distributions, they state that it is the best baseline to compare various metrics with no bias or misdirection towards the data. They achieve an average probability of detection (pd) of 71% and an average probability of false alarm (pf) of 25% using only static metrics, which bolsters their theory that the selection of a classifier model is of far greater importance than the metrics that will be used to classify the data. They also disproved the previous pessimism [90] [91] against the relevance of static code metrics for bug prediction.

Tosun et al. in [92] implement a defect prediction model for a Turkish Telecommunications Company to streamline their development process. A Naive Bayes classifier is used with a collection of repository metrics and code metrics. They initially got a pd of 88% and a pf of 28%. Further improvements were made in the form of extra relational data and adjusting the threshold of the Bayes model. After the improvements, the pd was enhanced to 86% and pf to 14%. The authors discuss the challenges of putting a theoretical academic defect prediction model into practice and state that more effort should be put into understanding the type of data and the requirements of the company than the selection of a classifier. They recommend the use of a simple classifier like Naive Bayes, which can be fine-tuned based on the needs of the organization, over more complex models.

Hata et al. used fine-grained module history data to predict the existence of faulty methods or functions [93]. One of the challenges they faced is that almost all version control data is at the file level or package level. So, they proposed a fine-grained version control system that is built on top of git called "Historage" that can provide suitable metrics for method-level data. They used Random Forest Classifier to test the performance of their model compared to other file-level and package level models. They tested the model on eight open-source projects and found that model-based prediction is better at reducing effort-based evaluation. They also found that past data for methods did not correlate well with future predictions, but code metrics had positive co-relations.

Giger et al. in [94] use four classifiers: Random Forest, SVM, Bayesian Network, and J48 Decision Tree to classify methods of a file as defective or not defective. They take method-level code metrics like the ratio of comments to source code, McCabe's complexity of the method, number of
executable statements, etc. They also get method-level repository metrics by taking file-level metrics from a version control system and taking the abstract tree difference between the two versions of a file to track the changes made to the method. The repository metrics used include the number of authors, statements added, churn, number of declarations among others. The model was tested on 21 open source JAVA projects and Random Forest performed consistently well with a recall of 0.85 and AUC of 0.95. The combination of repository metrics and code metrics gave the best results compared to using them individually for all four classifiers. The authors conclude that given the performance of the model combined with the reduction in effort needed to fix the bug compared to file level prediction, their model performs better in an industrial setting.

Antal et al. propose function-level JavaScript-based defect prediction models [95] which use static and hybrid code metrics. Since JavaScript is a dynamic language, static code metrics do not capture the full extent of its functionality. Therefore, they introduce hybrid code analysis-based metrics which are calculated based on the call graph of the JavaScript functions using ESLint. Using the results of the call graph, two new metrics were calculated: Hybrid Number of Incoming Invocations and Hybrid Number of Outgoing Invocations. Nine Machine Learning classifiers including Linear Regression and SVM were used to test the new metrics. It was found that the new metrics consistently improved the performance of the classifiers by 2-10% in all categories. Random Forest Classification achieved the highest accuracy and precision while KNN got the best recall with balanced results in precision and accuracy.

3.4.5 Other

Tan et al. [96] propose a bug classification technique that finds buggy program clusters, instead of the traditional method of finding buggy methods, classes, or files. The programs are clustered based on their code metrics like function calls and variables references, and information metrics like comments and names of identifiers. They use linear and logistic regression to classify the clusters as buggy and find an increase of 30% in accuracy for linear regression compared to other class-based prediction models. Logistic Regression gives even better results improving recall by 67.6% and precision by 17.8%. 
3.4.6 Comparison of Results

Most of the techniques discussed above use Machine Learning to predict and identify faulty modules. There are some exceptions to this, however. Pandey et. al. use a Fuzzy Inference System [60] with rules extracted from decision trees to predict faulty classes and achieved an accuracy of 87%. Moeyersoms et. al. in [5] use ALPAC to generate rules that mimic the performance of a Random Forest model. The model got high accuracy of 88% on average but the recall was quite low at 20%. Kamei et. al. in [67] used a hybrid approach between rules-based classification and machine learning to classify error-prone files. They tried various approaches with the highest recall being 87% and the highest F1 score being 43%.

The models which predict faulty packages gave some of the highest values performance metrics because of the low granularity. Nagappan et. al. used Principal Component Analysis on Microsoft Projects [56] and got an $R^2$ value ranging from 0.416 to 0.882. Schroter et. al. predicted faulty components at design time using problem domain knowledge and their best approach got a recall of 79% with a precision of 60% [57].

Class level fault prediction is one of the most common levels of granularity observed in the literature. Huang et. al. [59] classify classes from a Multi-Instance Learning perspective by using a bags and instances approach. The model reached an accuracy of 91% but got a relatively bad Type-1 error ratio of 67%. Palomba et. al. in [65] introduced a new metric that can calculate the intensity of code smell for a class. JCodeOdor to classify the faulty modules and an overall recall of 84% with a precision of 76% was obtained.

File-level fault prediction is the level where most modern bug prediction techniques reside. Wang et. al. used a Deep Belief Network [69] on semantic data to classify buggy files. They got an average recall of 56% with a precision of 37%. Lee et. al. [78] introduce new developer behavior measuring metrics and use them on top of repository and source metrics to get an average F1 score of 49%. Kim et. al. classify code changes [79] as buggy and non-buggy using SVM on repository data. The average buggy recall achieved by the model was 59% with an average accuracy of 78%.

Method Level Bug Prediction is a relatively new area and not a lot of work has been done on this level of granularity. Elish et. al. use SVMs on mission-critical projects from NASA to classify fault prone methods or functions [81]. The model outperformed most statistical models and
obtained an average recall of 99% with an average accuracy of 90%. Mizuno et. al. use a Spam Based approach to filter buggy Java methods by treating the code as a text file and using advanced Spam filtering techniques on it [83]. They evaluated the technique using three different state of the art filtering approaches and the average recall obtained for the three was 82% with an accuracy of 72%.

The above results show that there is still a lot of research to be done in this field of Software Engineering. Almost every technique in the literature is inherently black box in nature, with a few exceptions like [60] [5]. However, even the techniques that are partially transparent give very complex explanations that are not understandable to the typical developer. They also don’t allow for any kind of customization by a domain expert. Our research aims to fill these gaps by presenting a technique that performs on par with more complex algorithms while providing comprehensible explanations and customizability.

3.4.7 Recent Work

In this section, we discuss the recent research that has been going on in the field of Bug or Fault Prediction. We chose the year 2017 as a benchmark and only chose papers that were published on after that year in this section.

Kumar et al. [97] discuss the scope of using Learning Classifier Techniques (LCS), of Genetic Algorithms, in the field of fault prediction. They use Zeroth level Classifier Systems (ZCS) and Accuracy based Classifier (XCS) Systems, both of which are popular and well studied LCS techniques. To get the data, they demonstrate the use of binary level encoding to form binary strings of data instances, which can be used as input to the prediction model. Genetic Algorithms are used here to create new rule systems from existing rules using a credit and a reward system. The proposed methodology is feasible to create a tool which can aid in predicting faulty modules in the early stages of software development.

Nucci et al. proposes two new developer metrics [98] which measure how focussed the developers are during their code commits. They leverage research from [99] which conclude that focused developers are less likely to introduce faulty changes than an unfocused one. Their first metric measures how structurally far the code components modified by a developer are, and the second metric measures how scattered the implemented responsibilities of the modified code components


Chapter 3: Related Work

are. The authors use two bug predictor models on 26 open-source systems to test their performance against four baseline prediction algorithms used in the industry. They found that, on average, the new predictor techniques improve the F1 score by 20%.

Jayanthi et. al. [100] implement a Neural Network based fault prediction classifier. They use Data Mining techniques because they are better suited for imbalanced data and they employ Principal Component Analysis for feature and dimensionality reduction. They find that the reduced dataset is more efficient in performance that a standard dataset. The model is used to predict defects in four projects in the PROMISE repository and they received an impressive AUC of 97%, which is better than other conventional techniques. They theorize that applying double pre-processing on the data using instance filtering will allow for better performance metrics.

Most techniques and models proposed to identify fault prone modules are supervised in nature. However, their performance depends on the availability of fault data history. Boucher et. al. propose an unsupervised fault proneness prediction model, HySOM, in [101] to circumvent the problem of data availability. They adapt the HySOM model to predict faulty classes in an object-oriented setting to improve the overall performance of the model. The model is further compared with other supervised approaches like Naïve Bayes Network, ANN, and Random Forest. The thresholds for the source code metrics used in the study were calculated using ROC curves and Alves Rankings. The proposed model had better performance than supervised techniques in the same setting and authors propose to extend the model using reusable elements from other techniques in the literature.

Sureka et. al. propose a technique [102] to identify bugs which occur due to the software aging. Software metrics are extracted from tera-PROMISE repository and four different feature selection techniques (OneR, Information Gain, Gain Ratio, RELEIF and Symmetric Uncertainty) are tested on it to reduce dimensionality. SMOTE method is then used on it to negate class imbalance. Extreme Learning Machines with three different kernels are used to model and predict the data. Their performance was compared using Wilcoxon signed-rank test and the results indicate that the ELM with linear kernel function outperform ELM with polynomial and RBF kernel.

Strüder et. al. study the performance of bug prediction techniques on a software feature level [103]. Features are a primary unit of abstraction in software product lines and play a major role in Agile
Methodology. This study was one of the first works done in this level therefore, the major hurdle was in obtaining the relevant datasets to train and test the models. Preprocessor macros were used to annotate source code with features. Feature references from files were then extracted using pattern matching and a feature was annotated as faulty if one of the files modified within it was found to be defective. A total of fourteen process and source code metrics were extracted to train seven different popular supervised classifiers. The authors observed precision and recall values of around 85% and the robustness was improved when a more diverse metric set with richer feature information was used.

Singh et. al. study the use of cross project data for rule-based learners [104]. They chose classifiers like JRip, CART, C4.5, OneR, and NNge partly because they don’t need a lot of training data to perform well. In total, they applied nine classifiers on five projects in different domains to test their execution. C4.5 gave the best results when the data used was in-project, but DTNB outperformed all other classifiers for cross-project datasets with an ROC of 69%.

Moustafa et. al. perform a comprehensive study on different ensemble classifiers and different sets of software metrics and evaluate them on datasets of different sizes [105]. The ensemble classifiers are a combination of single classifiers like Naïve Bayes, Decision Trees, SVM, Logistic Regression, and Random Forest. The results of a classifier are decided based on four different weighted majority voting techniques and their performance is compared with base classifiers. They use process metrics, source code metrics, and a combination of both to test the models. The results indicate that the cascading ensemble classifiers that use process metrics outperform all other models and classifiers that test on imbalanced data. The authors recommend using over-sampling or under-sampling to deal with the class imbalance problem.

Unlike typical binary classification models found in the literature, Hong et. al. attempt to classify fault prone modules on a severity index [106], so that developers can focus more on the bugs that will have a bigger impact on the software system. To accomplish this, they used three custom severity-based metrics: Maximum Severity Level of a module, Module Severity of a module, and Module Severity Density of a module. The classification algorithms used in the study were Naïve Bayes, Random Forest and NNge, which were tested on three NASA projects with severity data
present in them. They find that the new metrics outperform previously studied models, and the multi-layer perceptron neural network showed the best results among the classifiers.

3.4.8 Research Gap

In the previous section, we have shown the ongoing research that is going on in the field of Bug Prediction. None of the studies mentioned above focus on the model acceptance of the techniques in the industry. The primary goal of all the models is to have better performance, but at the cost of transparency and customizability. [104] is the only recent study we have found that uses a semi-transparent rule-based system. However, even they use automatically generated rules, which do not scale very well and do not give developers a lot of feedback or control. In contrast, the models and techniques described in this thesis, not only give developers, or projects experts, explanations as to why the model finds certain files or modules as faulty but gives them complete control to customize the model based on their needs and knowledge. We provide custom tools that leverage thresholds of metrics to ease the creation of new rules and also offer rule training programs that can be used to examine the validity of certain custom rules. We believe that putting developers in control of the bug prediction technique is the best way to increase the model acceptance of fault proneness identification algorithms in the industry since it can be tailored to a company’s individual needs.
Chapter 4: Data Modelling

4.1 Repository Data Extraction

In this thesis, we have gathered data from 18 open-source systems that vary in size and complexity. Table-3 lists the profiles of each project. The reason we selected these 18 open source systems is that they have both GitHub and Bugzilla repositories. We use the two repositories to reconcile fault data between them. More specifically the Bugzilla records are used to indicate the resolution of a fault and indicate the culprit file or files. Then the reconciliation process identifies the GitHub record that commits the bug-fixing changes. The data from GitHub and Bugzilla are extracted using specialized readers which populate two corresponding data models. This Chapter discusses the extraction and modeling process in detail.

4.1.1 GitHub Model

The process of acquiring GitHub data is based on a two-step procedure. We first connect and extract data from GitHub using a custom-made client-side extractor. Then all the acquired information is put together into one data model which is defined in Figure-9. We have used Python for the data extraction and fusion steps. The data model is populated by first taking the initial repository and then cycling through the commits iteratively while adding relative data to the model. By adding the information found on the commits of the master branch sequentially, we maintain the original structure of the repository. Various pre-processing tasks are then conducted after the model is populated with the data. All the files that do not contribute to a defect, for example, non-configuration and non-compilable files, are removed. To only keep the substantial code-changing commits, all the extracted commits are cleaned up using a simple heuristic. This step removes any commit that is annotated as a refactoring commit and all files that are eventually removed from the system, are retroactively removed from all of their respective commits.

4.1.2 Bugzilla Model

Bugzilla data, like GitHub data, was acquired using a custom-built extractor. The data was collected from Bugzilla repositories, which follow the Bugzilla Data Schema defined in [107]. The
primary entity in the schema is the bug report, which has the following fields we are interested in: bug_id, bug_status, creation_ts (date on which it was created), resolution, short_desc (a description of the bug), product, reporter, and comments which is a list of all submitted comments or attachments for the bug. From the bug report object, we extract the data and comments for all the reports which have status Resolved or Fixed. The comments for these reports are analyzed to get the list of modified files. This information is stored in the report either as plaintext submissions or as changelog attachments that originate from the versioning control software used by the project. We use this list of modified files to narrow down the commit which introduced the bug using the reconciliation process, discussed in the next section.

Chapter 4: Data Modelling

Figure 9: Data Model for Raw GitHub Data [108]
4.1.3 GitHub-Bugzilla Data Reconciliation

A fault fixing commit can have other files committed which are unrelated to the fault. Therefore, it is important to find out exactly which files were affected. We use the data that was collected from GitHub and Bugzilla to identify in a buggy GitHub commit, which file or files were faulty. It is part of work conducted in a related project [108] and the reconciliation process takes place after all the data is extracted and the data models for GitHub and Bugzilla are populated. In the first step, all the bug resolutions are arranged chronologically using their resolution date and grouped by bug ID. In the second step, we go through all commits available in the GitHub model iteratively till we reach one of the dates available in the Bugzilla resolution data. The search space is then limited to the window around the date of resolution. Finally, for each commit in that window, the maximal intersection between the files in the commit and the modified files in the bug report was calculated and the commit which showed the largest intersection while remaining closest to the Bugzilla resolution date had those particular files marked as faulty. We now show the reconciliation process using an example.

Let $B_k$ be a Bugzilla report with a timestamp $t_k$ along with the set of modified files $E_k = \{F^1_k, F^2_k, ..., F^j_k\}$ where $F^j_k$ is the $j$th file committed on commit $k$. We then identify the set of commits $cid_1, cid_2, ..., cid_m$ that fit in our timestamp window $[t_k - x, t_k + x]$. In our experiments, we set the value of $x$ to one month. If $S_i = \{F^1_i, F^2_i, ..., F^l_i\}$ is the set of files committed in commit $cid_i$, then we choose the $cid_n \in \{cid_1, cid_2, ..., cid_m\}$ committed at the time $t_n$ where $cid_n$ is the commit for which the intersection of $E_k$ and $S_n$ is maximal. If two commits have the same maximal intersection, we choose the $t_n$ that is the closest to $t_k$ as shown in Figure-10.

![Figure 10: Timeline for Reconciled Data][108]
4.1.4 Data Format

Figure-11 shows the entire process of collecting and reconciling GitHub and Bugzilla data. After the models are populated and the reconciliation is done, we store the resultant data in a CSV file with each file in a commit having a separate row with the following columns: id, branch, message, parent_ids, author, authored_at, commit_additions, commit_deletions, changed_files, is_bug_linked, is_fix_related, is_bug_fixing, is_merge_commit, is_refactoring, file_path, previous_file_path, file_additions, file_deletions, file_id, fractal_value, distinct_authors_to_now.

We will now briefly explain every column.

**id:** The commit ID that the file was committed in  
**branch:** Specifies whether the commit was committed on the master branch or regular branch  
**message:** The commit message given by the developer when committing the file  
**parent_ids:** The commit ID of the previous commit  
**author:** The name of the developer that committed the file  
**authored:** The date and time of the commit
commit_additions: Total number of lines added in the commit
commit_deletions: Total number of lines deleted in the commit
changed_files: Total number of files modified in the commit
is_bug_fixing: A Boolean value specifying whether the commit is a bug fixing commit
is_merge_commit: A Boolean value specifying whether the commit is a merging commit
is_refactoring: A Boolean value specifying whether the commit is a refactoring commit
file_path: The absolute path of the file in the project
previous_file_path: The absolute path of the file before the commit
file_additions: Number of lines added in the file committed
file_deletions: Number of files deleted in the file committed
file_id: The ID of the file that was committed
fractal_value: The percentage of modified lines for this file over other files in the commit
distinct_authors_to_now: Number of distinct authors that have contributed to the file

4.1.5 Project Selection

The experiments were performed on 18 open-source projects whose profiles are listed in Table-3. The projects were chosen based on various factors and conditions that include:

- The project needs to have both Bugzilla and GitHub repositories.
- The project has to have a high level of activity and participation from the developers.
- The repository is not a personal project and is used extensively in software applications.
- The project has to use Github exclusively and should not use any other versioning systems.

For each project chosen, we have extracted the last five years of historical data. This was done because software systems, especially open source, undergo a lot of changes quite quickly. The old commit data would no longer be relevant now because the processes, developers, and activity change over time. By taking the latest data, we ensure that the results we get and observe are valid and will continue to be valid for quite some time.
Table 3: Projects Examined and their last five years' profile

<table>
<thead>
<tr>
<th>Project</th>
<th>Total Commits</th>
<th>Total Segments</th>
<th>Ratio of Clean vs Buggy Segments in Testing Dataset</th>
<th>Testing Data in Years</th>
<th>Testing Data Percentage*</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>1922</td>
<td>1286</td>
<td>16</td>
<td>1</td>
<td>19%</td>
</tr>
<tr>
<td>ark</td>
<td>1772</td>
<td>1200</td>
<td>3.7</td>
<td>2</td>
<td>21%</td>
</tr>
<tr>
<td>elisa</td>
<td>1879</td>
<td>1571</td>
<td>1.4</td>
<td>1</td>
<td>27%</td>
</tr>
<tr>
<td>gwenview</td>
<td>4185</td>
<td>1466</td>
<td>2.8</td>
<td>3</td>
<td>23%</td>
</tr>
<tr>
<td>juk</td>
<td>3017</td>
<td>1230</td>
<td>15.7</td>
<td>3</td>
<td>19%</td>
</tr>
<tr>
<td>k3b</td>
<td>934</td>
<td>4982</td>
<td>74.4</td>
<td>2</td>
<td>23%</td>
</tr>
<tr>
<td>kate</td>
<td>1854</td>
<td>1450</td>
<td>17</td>
<td>1</td>
<td>25%</td>
</tr>
<tr>
<td>kdelibs</td>
<td>247</td>
<td>325</td>
<td>8.5</td>
<td>2</td>
<td>46%</td>
</tr>
<tr>
<td>kio-extras</td>
<td>710</td>
<td>855</td>
<td>10.4</td>
<td>2</td>
<td>48%</td>
</tr>
<tr>
<td>kmix</td>
<td>2252</td>
<td>1370</td>
<td>18.8</td>
<td>2</td>
<td>16%</td>
</tr>
<tr>
<td>kompare</td>
<td>1056</td>
<td>481</td>
<td>42</td>
<td>2</td>
<td>20%</td>
</tr>
<tr>
<td>konsole</td>
<td>1466</td>
<td>1450</td>
<td>4.3</td>
<td>1</td>
<td>27%</td>
</tr>
<tr>
<td>konversation</td>
<td>563</td>
<td>444</td>
<td>5.8</td>
<td>1</td>
<td>20%</td>
</tr>
<tr>
<td>ktorrent</td>
<td>534</td>
<td>977</td>
<td>18.2</td>
<td>2</td>
<td>29%</td>
</tr>
<tr>
<td>lokalize</td>
<td>1668</td>
<td>1766</td>
<td>3.4</td>
<td>2</td>
<td>17%</td>
</tr>
<tr>
<td>plasma</td>
<td>2898</td>
<td>2004</td>
<td>6.4</td>
<td>2</td>
<td>18%</td>
</tr>
<tr>
<td>solid</td>
<td>577</td>
<td>2413</td>
<td>44</td>
<td>2</td>
<td>30%</td>
</tr>
<tr>
<td>systemsettings</td>
<td>2094</td>
<td>509</td>
<td>10</td>
<td>2</td>
<td>25%</td>
</tr>
</tbody>
</table>

*- The test data was chosen based on the number of years and availability of commits. So, the percentages of the testing set are not constant for all projects. They correspond to the number of years worth of commits. The selection of the testing data set is discussed in more detail in Chapter 6.

4.2 Segments

We observe trend values of our metrics over segments where each segment is a collection of ordered commits. The concept of segments was introduced in the work done by Ria et. al. in [109].
In this thesis, the use of segments is three-fold. It is used to calculate the metrics of the commits, find the trends of the metrics with respect to other segments, and predict the fault-proneness of a file in a segment. If we predict a file $f_{id_i}$ to be faulty in segment $sid_j$, then we say that one or more code changes in $f_{id_i}$ that were committed as part of a commit in segment $sid_j$ is faulty.

If we consider an ordered list of commits for a project as $C = \{cid_1, cid_2, \ldots, cid_n\}$, then we can partition this list into an ordered list of segments $S = \{sid_1, sid_2, \ldots, sid_m\}$ where each segment $sid_i$ is a collection of commits $\{cid_{k}, cid_{k+1}, \ldots, cid_{k+l-1}\}$. In this case, we say that the width of the segment is $l$. The width of a segment can either be static or variable. Having static segment width would mean that every file would have the same number of commits in a segment. We opt to have a variable segment width because different files are committed at different frequencies and having the same width for all files will result in some files having too many commits in a segment and other files having no commits in a segment.

We calculate segment width by taking the mean average of the difference of successive commits for a file. Let’s assume file $A$ is committed $m$ times over $n$ commits where $m \leq n$. Let the commits where $A$ was committed be $\{cid_{A1}, cid_{A2}, \ldots, cid_{Am}\}$. The function $diff(cid_{1}, cid_{2})$ calculates the number of hours that elapsed between the two commits and then normalizes it so that the value does not exceed 24. We use the function to calculate the difference in commits for each pair of consecutive commits of $A$. If $diff(cid_{1}, cid_{2})$ is represented as $diff_{1,2}$, then we calculate segment width for file $A$ using

$$swidth_A = \frac{\sum_{i=1}^{m-1} diff_{Ai,Ai+1}}{m} = \text{avg}(diff_{A1,A2}, diff_{A2,A3}, \ldots, diff_{Am-1,Am}) \quad (4.1)$$

Since files are committed with different frequencies, the value of $swidth$ changes with each file.

Figure–12 shows an illustrative example of how different files can have segments of different widths. File $F_x$ has three segments in the given range of commits while File $F_y$ has two segments in the same range of commits. This is because $F_x$ was committed more frequently and hence has more segments than $F_y$. The width of each segment is calculated based on Equation 3.1 above. For
example, the segment width of a file which is not committed frequently would be wider (i.e. will contain more commits) than the segment width of a file committed less frequently.

Figure 12: Illustrative Example for Segment Widths

4.3 Metrics

Commit Frequency: This metric measures the number of times a file was committed in a segment. We found this metric to be an extremely effective investigative tool because a file that was committed frequently in a short span of time tended to be fault-prone than others. As a standalone metric, its accuracy was low, but it worked well when combined with other metrics.

Merge Frequency: It represents the number of branch merges that occurred in the segment. If there are multiple merges in a short window, it could be a sign of an unstable application or a project that introduced multiple changes in quick succession. Both factors help identify fault-prone files.
**File Churn:** This metric measures the total number of lines modified to a file in a commit. It includes the lines added and deleted. A file that has a high file churn is typically fault-prone more frequently than a file with a low file churn.

**Failure Intensity:** This metric is used to measure the fault proneness of the file in segments prior to the current one. It is the ratio of the number of times the file was fault-prone to the total number of times the file was committed. In most cases, the failure intensity of a file is zero. We are interested in the cases where the file was fault-prone in the previous segment or the previous two segments. We theorize that if a file was fault-prone in the previous segments, then it is also fault-prone in the current segment.

**Fractal Value:** It is a measure of how much different developers contributed towards a file and was adapted from the work done in [110]. Its value ranges from (0,1] and a value of 1 would signify that the file only has one developer. We consider a file having a higher Fractal Value to be more consistent and cleaner than a file that has been modified and committed by multiple developers. This value is updated with every commit that the file participates in.

**Distinct Authors:** This metric calculates the number of authors that modified the file in the given segment. The results indicated that if a file was modified by a lot of developers, it is a sign that the file is/will be unstable. This metric was typically used in conjunction with commit frequency to find fault-prone segments.

**Overall Strength:** This metric was introduced as part of the work done by Ria et. al. in [109]. It is a score that is given to every file in a commit based on its dependency with the other files in the commit and the decay of the dependencies of the file with every other file committed with it before. The dependency score of one file with respect to another file is called Binary Strength and it is calculated using the formula:

\[
\text{Binary Strength} = CT(A, B) + LR(A, cid) + LR(B, cid) - OC(A, B) \quad (4.2)
\]

where:

\(CT(A, B)\) is the total number of times files A and B have been co-committed divided by the sum of the number of times File A and File B have been committed together throughout the history of the project.
$LR(A, cid)$ is the number of modified lines of file $A$ in commit $cid$ divided by the total lines modified in the commit $cid$, without counting the number of modified lines in files $A$ and $B$.

$OC(A, B)$ is the total number of commits where file $A$ is committed with other files (except $B$), between two subsequent co-commits of files $A$ and $B$ divided by the number of times File $A$ is committed so far.

The Binary Strength value between two files $A$ and $B$ is decayed every time $A$ is committed without $B$. Overall Strength is calculated by adding the binary strength of file $A$ with all other files $\{B_1, B_2, ..., B_k\}$ in commit $cid$ along with the decayed Binary Strengths of file $A$ with all files $\{C_1, C_2, ..., C_n\}$ that were committed with $A$ in the previous commits $cid_p$ but not co-committed with $A$ in the current commit.

\[
Overall\_Strength = \sum_{i=1}^{k} Binary\_Strength(A, B_i, cid) + \sum_{j=1}^{n} Binary\_Strength(A, C_j, cid_p, j) \quad (4.3)
\]

4.4 Threshold Calculations

Every metric described in the previous section must first be divided into categories. This is done for two reasons: to satisfy the requirement of Fuzzy Logic to have Linguistic Variables, and because it is easier for the expert to write rules when the variables are divided into classes. In this thesis, we divide the metrics into three threshold levels: low, avg and high. The boundaries for the thresholds are flexible for Fuzzy Logic and rigid for Markov Logic Networks (MLN). We use membership functions to define the boundaries in Fuzzy Logic and create different predicates for different categories in MLN.

The thresholds are calculated by comparing the given metric of all files in the segment with the file under observation. The lower boundary of low is the lowest metric value of any file in the segment and the upper boundary is equal to the value of avg. The lower boundary of avg is calculated by summing up the values of the metric of all files in the segment, except the current file, and dividing it with the number of files.

\[
avg = \frac{\sum_{i=1}^{n} metric(File_i)}{n} \quad (4.4)
\]
Where \( n \) is the number of files, excluding the current file in the segment.

The upper limit of \( \text{avg} \) is equal to the lower limit of \( \text{high} \) and is calculated by adding the \( \text{avg} \) calculated using eq. 3.4 with 2 times the standard deviation of values of the metric of all files in the segment, except the current file.

\[
\text{Standard Deviation}(\sigma) = \sqrt{\frac{\sum (x_i - \text{avg})^2}{N}} \tag{4.5}
\]

Where \( x_i \) is the \( i^{th} \) file’s metric value and \( N \) is the number of files.

\[\text{high} = \text{avg} + 2 * \sigma \tag{4.6}\]

The upper limit of \( \text{high} \) is dynamic and is equal to the highest metric value of any file in the segment.

Consider an example where the segment of File A consists of 4 commits which include 7 files. The values of the overall strength of each file in the segment are as follows: 3, 2, 1, 2, 10, 1, and 2.

Here, the lower boundary of \( \text{avg} \) is equal to \( \frac{3+2+1+2+10+1+2}{7} = \frac{21}{7} = 3 \)

Standard deviation is equal to \( \sqrt{\frac{(3-3)^2+(2-3)^2+(1-3)^2+(2-3)^2+(10-3)^2+(1-3)^2+(2-3)^2}{7}} = \sqrt{\frac{60}{7}} = 2.92 \)

We get the lower boundary of \( \text{high} \) by calculating \( 3 + 2 * 2.92 = 8.8 \)

The ranges for \( \text{low}, \text{avg} \) and \( \text{high} \) for the overall strength of File A are:

\[
\text{low} \in [1,3) \\
\text{avg} \in [3,8.8) \\
\text{high} \in [8.8,10)
\]
4.5 Rule Repository

This model relies on expert knowledge to create tailor-made rules for each project. While we do provide the framework to do that, we also create a boilerplate set of rules that can be a starting point to create a robust fault prediction model. This section is used to state and explain each rule that was used to test the model.

**Rule 1:** If the file was fault-prone in the previous two segments, is committed frequently in the current segment, and has a high overall strength; then the probability of the file being fault-prone in the current segment is high.

**Rule 2:** If a file having high overall strength is committed frequently and is also merged a lot in the current segment; then the probability of the file being fault-prone in the current segment is high.

**Rule 3:** If the file has a lot of modifications and is merged frequently in the current segment; then the probability of the file being fault-prone in the current segment is high.

**Rule 4:** If the file is heavily modified in the current segment and was known to be fault-prone in the previous segment, then the probability of the file being fault-prone in the current segment is high.

**Rule 5:** If a file that was fault-prone in the previous segment and is committed frequently, has an above-average file churn and overall strength; then the probability of the file being fault-prone in the current segment is high.

**Rule 6:** If a file having average file churn was found to be fault-prone in the previous segment, then the probability of the file being fault-prone in the current segment is average.

**Rule 7:** If a file is committed very frequently and has an average file churn, then the probability of the file being fault-prone in the current segment is average.

**Rule 8:** If the file was fault-prone in the previous segment and participated in a high number of merges in the current segment, then the probability of the file being fault-prone in the current segment is average.
Rule 9: If a file having average overall strength has an average number of commits, then the probability of the file being fault-prone in the current segment is average.

Rule 10: If a file having a high number of commits in the segments also has a high file churn, then the probability of the file being fault-prone in the current segment is high.

Rule 11: If none of the metrics that define the segment are high, then the probability of the file being fault-prone in the current segment is low.

Rule 12: If a file having a high number of commits in the segment also has an average file churn and a high or average fractal value, then the probability of the file being fault-prone in the current segment is high.

Rule 13: If a file was ever fault-prone in the previous three segments, then the probability of the file being fault-prone in the current segment is average.

Rule 14: If a file having average or high overall strength is committed frequently with a low or average file churn, then the probability of the file being fault-prone in the current segment is average.

Rule 15: If a file having high overall strength also participated in merges frequently in the segment, then the probability of the file being fault-prone in the current segment is high.

Rule 16: If the file participated in an average number of merges in the segment and has an average or high overall strength, then the probability of the file being fault-prone in the current segment is average.

Rule 17: If the overall strength of the file in the segment is high or average and the file churn of the file is also high or average, then the probability of the file being fault-prone in the current segment is average.

Rule 18: If the overall strength of a file is average and its fractal value in the segment is average or high, then the probability of the file being fault-prone in the current segment is average.

Rule 19: If the number of commits the file has participated in the segment is average and its fractal value is also average, then the probability of the file being fault-prone in the current segment is average.
Rule 20: If the number of distinct authors that the file has in the segment is high and it was committed frequently, then the probability of the file being fault-prone in the current segment is high.

4.6 Customized Rules

Adding custom rules for a project is a very heuristic process that largely depends on the expert’s understanding of the project. To simplify the procedure of finding the rules, a custom utility program was used that displays color-coded values of each metric of a segment. Visualizing different thresholds with different colors makes it much easier to find recurring patterns in the data.

First, inference is performed, and the segments are put into one of the four files: True Positives, True Negatives, FalsePositives, and False Negatives by the program. Each section has segments and its color-coded metric values as shown in Figure-13 and Figure-14.

Depending on the metric of interest, the corresponding file can be observed. For example, if the recall is needed to be increased, the file False Negatives would be the important file to monitor and if precision needs to be improved, then False Positives would need to be observed. By finding patterns in the data, rules can be introduced, and the inference is performed iteratively until a satisfactory set of rules is obtained.

<table>
<thead>
<tr>
<th>SID</th>
<th>FrequencyCommits</th>
<th>OverallSu</th>
<th>FrequencyMerges</th>
<th>FileChum</th>
<th>Fractal</th>
<th>DistinctAuthors</th>
<th>FailureIntensity1stSegment</th>
<th>FailureIntensity2ndSegment</th>
<th>FailureIntensity3rdSegment</th>
</tr>
</thead>
<tbody>
<tr>
<td>SID3287</td>
<td>0.685223</td>
<td>0.51</td>
<td>0.82528177</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3288</td>
<td>0.912991</td>
<td>0.15</td>
<td>0.8326542</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3289</td>
<td>0.910323</td>
<td>0.13</td>
<td>0.85467614</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3290</td>
<td>0.502699</td>
<td>0.48</td>
<td>0.96204416</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3291</td>
<td>0.90141</td>
<td>0.32</td>
<td>0.96432866</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3292</td>
<td>0.881242</td>
<td>0.21</td>
<td>0.1117661</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3293</td>
<td>0.699184</td>
<td>0.14</td>
<td>0.12073621</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3294</td>
<td>0.516814</td>
<td>0.10</td>
<td>0.1535202</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3295</td>
<td>0.369675</td>
<td>0.03</td>
<td>0.13861669</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3296</td>
<td>0.449878</td>
<td>0.34</td>
<td>0.15507457</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3297</td>
<td>0.890175</td>
<td>0.12</td>
<td>0.14111448</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3298</td>
<td>0.901324</td>
<td>0.2</td>
<td>0.1306533</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SID3299</td>
<td>0.703634</td>
<td>0.45</td>
<td>0.17041546</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 13: First screenshot of output given by the custom segment analyzing program

Figure-13 contains example data for the file FalseNegatives with 16 segments in it (SID287-SID1996). An expert can use the interface to start forming rules for the data to increase the recall of the example project. The first relation we can observe is between FrequencyMerges and...
Overall Strength. There are three fault-prone segments where both overall strength and frequency of merges are high (SID400, SID 991, SID1996). So, the first rule we add is:

**Rule 1:** *If both overall strength and frequency of merges are high, then the segment is fault-prone.*

The next relation we can observe is for the metrics FrequencyCommits, Fractal, and DistinctAuthors. We can introduce the following rule:

**Rule 2:** *If the frequency of commits of a segment is either high or avg and the segment has high file churn and distinct authors, then the segment is fault-prone.*

This rule would affect SID378, SID385, and SID400. The last rule we can add is:

**Rule 3:** *If the frequency of commits of a file in a segment is average, and the segment has an average file churn with the number of distinct authors being avg or high, then the segment is fault-prone.*

The above rule would affect SID983 and SID991. After the above rules are applied, the file FalseNegatives would like Figure-14.

**Figure 14:** Second screenshot of output given by the custom segment analyzing program

At this point, any rule we add will either only affect one segment or introduce too many false positives. So, we stop adding custom rules and use the three rules that were created to run the model on the testing set.
Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base

5.1 Fuzzy Based Fault Proneness Identification

This section outlines the fault proneness prediction system that uses Fuzzy Logic Inference. The first section in this chapter provides the overall system framework of the Fuzzy Logic model. We discuss the terminology and techniques that are used in Fuzzy Logic to provide inference and end the chapter by giving an overview of all the data and their formats that are used in the model.

5.1.1 System

The overall framework of the model is depicted in Figure-15. The process begins by first collecting the raw GitHub commit data from the selected projects using a custom client-side extractor. Raw bug records were also collected from Bugzilla using another custom-made extractor. The commits from GitHub were reconciled using Bugzilla data to get more accurate tagging for fault-prone commits. This GitHub-Bugzilla reconciliation process is discussed in detail in Section 4.1.

After the commit data are reconciled and properly formatted, according to the structure in Section 4.1.4, they are split into “training” and “testing” datasets\(^1\). We then combine commits to get segments for each file. Every file has a different segment width and its calculation is provided in Section 4.2. The metrics defined in Section 4.3 are calculated per segment for each file, after which their threshold levels are found by comparing their values with the metric values of all the other files in the segment. The values of metrics for each segment and their thresholds are stored in the form of a JSON file whose structure is explained in Section 5.1.2. Separate JSON files are created for training and testing datasets. The training JSON file is combined with an FCL file (Section 5.1.3), which contains all the rules present in the Rule Repository given in Section 4.5, and given to a Training Program (Section 5.1.6) which gives the best rule subset for the particular project.

\(^1\) We use quotes in “training” and “testing” datasets as the concept of these datasets differs from the classic use in Machine Learning.
The output FCL file given by the training module is sent to the Fuzzy Inference System along with some expanded rules customized to each project and the testing JSON file. The Fuzzy Inference finds the probability that the given segment is fault-prone using the rules and thresholds and outputs the results in a CSV format. A custom program is used to read this CSV file and report the performance metrics to generate the final results.
5.1.2 Intermediate Data Storage Format

Since the data fed to the *jfuzzylogic* controller is at the granularity of a segment, we have to store the metrics and the thresholds for each segment separately. In this thesis, JSON is used as the data format of choice to store the segment data. The main reason for choosing it is that JSON is more readable than XML and can handle nested and dynamic structures better than CSV. Key/Value pairs work better for the data we have than a traditional tree structure that XML provides. Also, JSON is faster to parse, easier to code and has a map structure.

Table-4 shows the JSON structure for the metrics in one segment. This structure is repeated for all segments and is stored in a single file. Every metric has the keys *avg*, *max*, and *threshold*. The metrics *fileChurn* and *fractalValue* are calculated for each commit of the file in the segment and the other metrics are calculated over the segment. *overallStrength* is the only metric that can hold negative values, which is why it has a *min* element. The JSON file is parsed iteratively for each segment and membership functions for each linguistic variable are updated dynamically based on the values of *min*, *avg*, *threshold*, and *max*.

```json
{
   "failureIntensity2ndSegment": {
      "avg": 0.6416667,
      "max": 1.0,
      "threshold": 1.0,
      "value": 0.0
   },
   "frequencyOfCommits": {
      "avg": 1.8571428,
      "max": 8.0,
      "threshold": 4.614117,
      "value": 1.0
   },
   "overallStrength": {
      "avg": 2.3313756,
      "min": 0.0,
      "max": 12.713992,
      "threshold": 7.3237686,
      "value": 18.738716
   },
   "fractalValue": {
      "avg": 0.5514831,
      "max": 1.0,
      "threshold": 1.0
   },
   "frequencyOfMerges": 5.0,
   "segmentID": "SID999",
   "failureIntensity3rdSegment": {
      "avg": 0.0,
      "max": 1.0,
      "threshold": 0.0
   }
}
```
5.1.3 Fuzzy Control Language

Fuzzy Control Language (FCL) is a standard format to represent Fuzzy Logic published by the International Electrotechnical Commission (IEC). The framework we use, *jfuzzylogic*, uses this specification to define the fuzzy rules and linguistic variables. Table 5 shows an example initial part of the FCL file we use in our experiments. The values of parameters for each linguistic variable were defined dynamically and changed for each segment. The values were updated by reading the JSON file described in the next section.

The VAR_INPUT and the VAR_OUTPUT blocks are the declaration sections of the file where the input linguistic variables and the output linguistic variable is defined. In our case, we define eight input variables for the various metrics and *isSegmentBuggy* as the output variable. They are all declared as real numbers.
VAR_INPUT
  distinctAuthors : REAL;
  failureIntensity1stSegment : REAL;
  failureIntensity2ndSegment : REAL;
  failureIntensity3rdSegment : REAL;
  fileChurn : REAL;
  fractalValue : REAL;
  frequencyOfCommits : REAL;
  frequencyOfMerges : REAL;
  overallStrength : REAL;
END_VAR

VAR_OUTPUT
  isSegmentBuggy : REAL;
END_VAR

Table 5: Variable Block in FCL File

5.1.4 Membership Functions Block

The keyword FUZZIFY is used to define the membership functions for each linguistic variable. In this thesis, we use three levels of granularity: low, avg and high for each linguistic input variable. We opt to use triangular and trapezoidal membership functions over gaussian curves because we wanted the degrees of membership of low and high to be the highest at the two extremes. This is the section that is updated for each segment.

DEFUZZIFY is used to define the membership functions of the output variable. We use the following three levels of granularity for it: buggy, maybe, and notBuggy. It also allows us to define the defuzzification method and default value. As discussed before, we use Center of Gravity (COG) as the defuzzification method and the default value is zero.

Sample membership functions for a segment are shown in Table-6.

FUZZIFY distinctAuthors
  TERM avg :=  (1.35, 0.0) (1.36, 1.0) (2.06, 1.0) (2.75, 0.0) ;
  TERM high :=  (2.06, 0.0) (2.75, 1.0) (4.0, 1.0) ;
  TERM low :=  (0.0, 1.0) (1.36, 0.0) ;
END_FUZZIFY

FUZZIFY failureIntensity1stSegment
  TERM avg :=  (0.46, 0.0) (0.47, 1.0) (0.73, 1.0) (1.0, 0.0) ;
  TERM high :=  (0.73, 0.0) (1.0, 1.0) (1.0, 1.0) ;
  TERM low :=  (0.0, 1.0) (0.47, 0.0) ;
END_FUZZIFY
Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base

FUZZIFY failureIntensity2ndSegment
  TERM avg := (0.63, 0.0) (0.64, 1.0) (0.82, 1.0) (1.0, 0.0);
  TERM high := (0.82, 0.0) (1.0, 1.0) (1.0, 1.0);
  TERM low := (0.0, 1.0) (0.64, 0.0);
END_FUZZIFY

FUZZIFY failureIntensity3rdSegment
  TERM avg := (0.0, 0.0) (0.0, 1.0) (0.0, 1.0) (0.0, 0.0);
  TERM high := (0.0, 0.0) (0.0, 1.0) (1.0, 1.0);
  TERM low := (0.0, 1.0) (0.0, 0.0);
END_FUZZIFY

FUZZIFY fileChurn
  TERM avg := (25.68, 0.0) (25.94, 1.0) (135.25, 1.0) (244.57, 0.0);
  TERM high := (135.25, 0.0) (244.57, 1.0) (779.0, 1.0);
  TERM low := (0.0, 1.0) (25.94, 0.0);
END_FUZZIFY

FUZZIFY fractalValue
  TERM avg := (0.54, 0.0) (0.55, 1.0) (0.77, 1.0) (1.0, 0.0);
  TERM high := (0.77, 0.0) (1.0, 1.0) (1.0, 1.0);
  TERM low := (0.0, 1.0) (0.55, 0.0);
END_FUZZIFY

FUZZIFY frequencyOfCommits
  TERM avg := (1.83, 0.0) (1.85, 1.0) (3.23, 1.0) (4.61, 0.0);
  TERM high := (3.23, 0.0) (4.61, 1.0) (8.0, 1.0);
  TERM low := (0.0, 1.0) (1.85, 0.0);
END_FUZZIFY

FUZZIFY frequencyOfMerges
  TERM avg := (3.97, 0.0) (4.01, 1.0) (6.66, 1.0) (9.31, 0.0);
  TERM high := (6.66, 0.0) (9.31, 1.0) (11.0, 1.0);
  TERM low := (0.0, 1.0) (4.01, 0.0);
END_FUZZIFY

FUZZIFY overallStrength
  TERM avg := (2.30, 0.0) (2.33, 1.0) (4.82, 1.0) (7.32, 0.0);
  TERM high := (4.82, 0.0) (7.32, 1.0) (12.71, 1.0);
  TERM low := (0.0, 1.0) (2.33, 0.0);
END_FUZZIFY

DEFUZZIFY isSegmentBuggy
  TERM buggy := (70.0, 0.0) (100.0, 1.0);
  TERM maybe := (40.0, 0.0) (55.0, 1.0) (70.0, 0.0);
  TERM notBuggy := (0.0, 1.0) (40.0, 0.0);
  METHOD := COG;
  DEFAULT := 0.0;
  RANGE := (0.0 .. 100.0);
END_DEFUZZIFY

Table 6: Membership Function Block in FCL File
5.1.5 Rule Block

The RULE BLOCK section allows us to define the method for AND and OR, activation method, accumulation method, and the rules. FCL follows a pair-wise declaration for AND and OR. Since we have defined the MIN method for AND, MAX is automatically applied to OR to conform to De Morgan’s laws. The activation method defines the degree of activation that each rule follows. In our case, since we use the MIN method, the minimum of all degrees of the truth of the antecedent is considered. The accumulation method defines the aggregation of the fuzzy sets obtained by each rule. Because we use the MAX method, it takes the maximum value among the membership functions of each rule. The rules are written in the form of an IF-ELSE statement as discussed in section 2.4.5.

Table -7 shows all the Fuzzy Rules used in the FCL file given as input for the model. The rules from 3.5 are reformatted in the syntax accepted by jfuzzylogic.

RULEBLOCK No1

    ACT : MIN;
    ACCU : MAX;
    AND : MIN;

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND
    (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND
    overallStrength IS high THEN isSegmentBuggy IS buggy;

RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND
    overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;

RULE 3: IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;

RULE 4: IF fileChurn IS high AND failureIntensity1stSegment IS high THEN
    isSegmentBuggy IS buggy;

RULE 5: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND
    failureIntensity1stSegment IS high AND (overallStrength IS high OR
    overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN
    isSegmentBuggy IS buggy;

RULE 6: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN
    isSegmentBuggy IS maybe;

RULE 7: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;

RULE 8: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high
    THEN isSegmentBuggy IS maybe;
Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base

Rule 9: IF overallStrength is avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;

Rule 10: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;

Rule 11: IF NOT(frequencyOfCommits IS high OR frequencyOfMerges IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR distinctAuthors IS high) THEN isSegmentBuggy IS notBuggy;

Rule 12: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;

Rule 13: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;

Rule 14: IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;

Rule 15: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;

Rule 16: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;

Rule 17: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;

Rule 18: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;

Rule 19: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

Rule 20: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;

END_RULEBLOCK

Table 7: Rule Block in FCL File

5.1.6 Rule Set Optimization

In fuzzy logic, each linguistic variable can have different membership functions (denoted by their terms) and the rules are a combination of the variables and the terms. Unlike Markov Logic Networks, Fuzzy Logic does not have a training framework to test the efficiency of the rules. To circumvent that, we created a custom Java-based Rule Selector which uses brute force to select the best set of rules that give the highest accuracy and recall for the training set. The input to the selector is the JSON file containing the training dataset and a set of rules from the Rule Repository.
described in Section 4.5. Twenty rules were defined out of which the selector selects the best performing rules for a given project.

The best subset of rules is selected by iterating through every combination of rules and selecting the rules that give the highest value of the metric. Here, the metric used to compare the performance of rules is \(\text{Accuracy} + \text{Recall}\). If we are selecting the best rule subset for the project \textit{gwenview} from the rule repository in Section 4.5, then the program iterates through 1.2 million combinations of rules to find the combination that gives the maximum sum of accuracy and recall for the training set. Table-8 shows the combinations of rules that give the best value for the metric in \textit{kdelibs}. We elect to only choose the best combination, but the user has the ability to select any combination from the top 5 list.

<table>
<thead>
<tr>
<th>Rule Combination</th>
<th>Combined Accuracy and Recall Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,2,3,9,10,19</td>
<td>1.377</td>
</tr>
<tr>
<td>2,4,6,7,9,10,12,15,19</td>
<td>1.368</td>
</tr>
<tr>
<td>2,3,6,8,9,10,13,15,19</td>
<td>1.359</td>
</tr>
<tr>
<td>4,5,6,7,9,10,12,13,19</td>
<td>1.35</td>
</tr>
<tr>
<td>1,3,4,5,6,8,12,14,15,19</td>
<td>1.342</td>
</tr>
</tbody>
</table>

\textbf{Table 8: Best Performing Rules for kdelibs}

This training module run for all projects and the frequency that each rule used is tabulated in Table-9. We use this table as a reference to select the most frequently used rules that can be part of the common rule dataset as discussed in Section 6.4.

<table>
<thead>
<tr>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rule 3</th>
<th>Rule 4</th>
<th>Rule 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Rule 6</td>
<td>Rule 7</td>
<td>Rule 8</td>
<td>Rule 9</td>
<td>Rule 10</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>10</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>Rule 11</td>
<td>Rule 12</td>
<td>Rule 13</td>
<td>Rule 14</td>
<td>Rule 15</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>15</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>Rule 16</td>
<td>Rule 17</td>
<td>Rule 18</td>
<td>Rule 19</td>
<td>Rule 20</td>
</tr>
<tr>
<td>9</td>
<td>13</td>
<td>9</td>
<td>10</td>
<td>14</td>
</tr>
</tbody>
</table>

\textbf{Table 9: Rules and their Frequency of Use}
5.2 Markov Logic Networks Based Fault Proneness Identification

In this section, we discuss the second rule-based fault proneness prediction system we propose, the one that uses Markov Logic Networks. We first present the overall process, and then we outline the Markov Logic Networks inferencing fundamentals. We then present the predicates and the raw data and fact extraction process. The chapter proceeds by presenting and discussing the rules and the training process which allocates weights to each rule for each training data set from each project. There is a separate training phase and different weights allocated for each rule in each different project. We conclude the chapter by explaining and giving an example of input and output data that is given to this type of reasoning model.

5.2.1 System

Figure-16 depicts the overall system framework of the Markov Logic Network Inference Model. The initial steps remain the same as the ones for the Fuzzy Logic Inference Model. The first step is the selection of projects and extraction which are the same as the one discussed in Section 5.1.1. The raw data that are extracted from GitHub and Bugzilla, as outlined in Sections 4.1.1 and 4.1.2, go through a reconciliation process where the bug records from Bugzilla are matched with the commits from GitHub to accurately identify bug fixing commits. After the data are reconciled and are stored in the format specified in Section 4.1.4, are split into training and testing sets as discussed in the previous chapters.

The second step of the model starts with aggregating the commits into segments. The segment width varies for every file and is dependent on the frequency of the commits of a file. The overall process by which segments are calculated and created is explained in Section 4.2. After the segments are created, the metrics discussed in Section 4.3 are calculated on individual segments. The metric values for the given file in a segment is compared with the metric values of all the other file that have commits in the segments. Based on these values, the threshold levels are calculated using the process given in Section 4.4.

Once the metrics and their thresholds are calculated, the DB files are created for both training and testing sets. The structure and format of the DB file are given in Section 5.3.4 below. The training DB file will serve as an input to the training module along with an MLN file which outlines all the
predicates and rules that will be used for inference. We use a discriminative training process to train the weights of all the rules as it will be discussed in Section 2.5.5. The training function outputs an MLN file with all the rules converted to Conjunctive Normal Form (CNF) and with their corresponding trained weights (see Section 5.2.5). This output MLN file is further enriched by adding custom expanded rules to better describe the training dataset.

The testing DB file and the output MLN file serve as inputs to the MLN inference system as discussed in Section 2.5.4. The inference system outputs a results file that has a list of segments and their associated probabilities of being fault-prone. These probabilities are given to a custom validation program that utilizes a custom threshold, discussed in 2.1.1, to classify segments as either buggy or not buggy. This classification is then validated, and performance results are given as output.
Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base

5.2.2 MLN Predicates for Fault-Proneness Prediction

The first part of the MLN file contains all the predicate declarations. These predicates are then used in rules to describe relations between them. The predicates that were used in the model are presented below.

\texttt{lowFileChurn}(sid, fid): True if the maximum file churn of the file \textit{fid} in segment \textit{sid} is lower than the average file churns of every other file that was committed in the segment.

\texttt{avgFileChurn}(sid, fid): True if the maximum file churn of the file \textit{fid} in segment \textit{sid} is higher than or equal to the average file churns of every other file that was committed in the segment and is lower than the sum of average and two times the standard deviation of the file churns of every other file in the segment.

\texttt{highFileChurn}(sid, fid): True if the maximum file churn of the file \textit{fid} in segment \textit{sid} is higher than the sum of the average file churns of every other file that was committed in the segment and two times its standard deviation.

\texttt{lowFrequencyOfCommits}(sid, fid): True if the number of times the file \textit{fid} was committed in the segment \textit{sid} is lower than the average frequency of commits of every other file that was committed in the segment.

\texttt{avgFrequencyOfCommits}(sid, fid): True if the number of times the file \textit{fid} was committed in the segment \textit{sid} is higher than or equal to the frequency of commits of every other file that was committed in the segment and is lower than the sum of the average and two times the standard deviation of the frequency of commits of every other file in the segment.

\texttt{highFrequencyOfCommits}(sid, fid): True if the number of times the file \textit{fid} was committed in the segment \textit{sid} is higher than the sum of the average frequency of commits of every other file that was committed in the segment and two times its standard deviation.

\texttt{lowOverallStrength}(sid, fid): True if the maximum overall strength of the file \textit{fid} in segment \textit{sid} is lower than the average overall strengths of every other file that was committed in the segment.

\texttt{avgOverallStrength}(sid, fid): True if the maximum overall strength of the file \textit{fid} in segment \textit{sid} is higher than or equal to the average overall strengths of every other file that was committed in the segment and is lower than the sum of average and two times the standard deviation of the overall strengths of every other file in the segment.

\texttt{highOverallStrength}(sid, fid): True if the maximum overall strength of the file \textit{fid} in segment \textit{sid} is higher than the sum of the average overall strengths of every other file that was committed in the segment and two times its standard deviation.
lowFrequencyOfMerges(sid, fid): True if the number of times the file fid was merged in the segment sid is lower than the average frequency of merges of every other file that was committed in the segment.

avgFrequencyOfMerges(sid, fid): True if the number of times the file fid was merged in the segment sid is higher than or equal to the frequency of merges of every other file that was committed in the segment and is lower than the sum of the average and two times the standard deviation of the frequency of merges of every other file in the segment.

highFrequencyOfMerges(sid, fid): True if the number of times the file fid was merged in the segment sid is higher than the sum of the average frequency of merges of every other file that was committed in the segment and two times its standard deviation.

lowFailureIntensity(sid, fid): True if the failure intensity of the file fid in segment sid is lower than the average failure intensities of every other file that was committed in the segment.

avgFailureIntensity(sid, fid): True if the failure intensity of the file fid in segment sid is higher than or equal to the average failure intensities of every other file that was committed in the segment and is lower than the sum of average and two times the standard deviation of the failure intensities of every other file in the segment.

highFailureIntensity(sid, fid): True if the failure intensity of the file fid in segment sid is higher than the sum of the average failure intensities of every other file that was committed in the segment and two times its standard deviation.

lowFractalValue(sid, fid): True if the maximum fractal value of the file fid in segment sid is lower than the average fractal values of every other file that was committed in the segment.

avgFractalValue(sid, fid): True if the maximum fractal value of the file fid in segment sid is higher than or equal to the average fractal values of every other file that was committed in the segment and is lower than the sum of average and two times the standard deviation of the fractal values of every other file in the segment.

highFractalValue(sid, fid): True if the maximum fractal value of the file fid in segment sid is higher than the sum of the average fractal values of every other file that was committed in the segment and two times its standard deviation.

lowDistinctAuthors(sid, fid): True if the distinct authors of the file fid in the segment sid is lower than the average distinct authors of every other file that was committed in the segment.

avgDistinctAuthors(sid, fid): True if the distinct authors of the file in the segment sid is higher than or equal to the distinct authors of every other file that was committed in the segment and is
lower than the sum of average and two times the standard deviation of the distinct authors of every other file in the segment.

**highDistinctAuthors(sid, fid)**: True if the distinct authors of the file *fid* in the segment *sid* is higher than the sum of the average distinct authors of every other file that was committed in the segment and two times its standard deviation.

**isBuggy(sid, fid)**: True if the corresponding file *fid* of the segment *sid* participated in a bug fixing commit in the segment.

### 5.2.3 MLN Rules

We now represent the rules initially presented in Section 4.5 in the form MLN and *Alchemy* supports. Note that the symbol ^ is used to denote the conjunction operator, the symbol V is used to denote disjunction, and the symbol => is used to denote implication. While the best rules to be applied (in order to maximize recall and accuracy) differ for every project and strategy, the predicates remain the same in all systems

//Rule 1
(highFrequencyOfCommits(s,f) v avgFrequencyOfCommits(s,f)) ^ highFailureIntensity(s,f) ^ highOverallStrength(s,f) => isBuggy(s,f)

//Rule 2
(highFrequencyOfCommits(s,f) v avgFrequencyOfCommits(s,f)) ^ highOverallStrength(s,f) ^ highFrequencyOfMerges(s,f) => isBuggy(s,f)

//Rule 3
highFrequencyOfMerges(s,f) ^ highFileChurn(s,f) => isBuggy(s,f)

//Rule 4
highFileChurn(s,f) ^ highFailureIntensity(s,f) => isBuggy(s,f)

//Rule 5
(highFrequencyOfCommits(s,f) v avgFrequencyOfCommits(s,f)) ^ highFailureIntensity(s,f) ^ highOverallStrength(s,f) v avgOverallStrength(s,f) ^ (highFileChurn(s,f) v avgFileChurn(s,f)) => isBuggy(s,f)

//Rule 6
avgFileChurn(s,f) ^ highFailureIntensity(s,f) => isBuggy(s,f)
//Rule 7
highFrequencyOfCommits(s,f)^avgFileChurn(s,f)=>isBuggy(s,f)

//Rule 8
highFailureIntensity(s,f) ^ highFrequencyOfMerges(s,f) => isBuggy(s,f)

//Rule 9
avgOverallStrength(s,f) ^ avgFrequencyOfCommits(s,f) => isBuggy(s,f)

//Rule 10
highFrequencyOfCommits(s,f) ^ highFileChurn(s,f) => isBuggy(s,f)

//Rule 11
!(highFailureIntensity(s,f) v highFrequencyOfMerges(s,f) v highOverallStrength(s,f) v
highFrequencyOfCommits(s,f) v highFileChurn(s,f) v highDistinctAuthors(s,f))=>!isBuggy(s,f)

//Rule 12
avgFileChurn(s,f) ^ (avgFractalValue(s,f) v highFractalValue(s,f)) ^
highFrequencyOfCommits(s,f) => isBuggy(s,f)

//Rule 13
highFailureIntensity(s,f) => isBuggy(s,f)

//Rule 14
(lowFileChurn(s,f) v avgFileChurn(s,f)) ^ (highOverallStrength(s,f) v avgOverallStrength(s,f)) ^
(highFrequencyOfCommits(s,f) v avgFrequencyOfCommits(s,f)) => isBuggy(s)

//Rule 15
highFrequencyOfMerges(s,f) ^ highOverallStrength(s,f) => isBuggy(s,f)

//Rule 16
avgFrequencyOfMerges(s,f) ^ (highOverallStrength(s,f) v avgOverallStrength(s,f)) =>
isBuggy(s,f)

//Rule 17
(highOverallStrength(s,f) v avgOverallStrength(s,f)) ^ (avgFileChurn(s,f) v highFileChurn(s,f))
=> isBuggy(s,f)

//Rule 18
avgOverallStrength(s,f) ^ (avgFractalValue(s,f) v highFractalValue(s,f)) => isBuggy(s,f)

//Rule 19
avgFrequencyOfCommits(s,f) \land\ avgFractalValue(s,f) \implies \text{isBuggy}(s,f)

//Rule 20

highDistinctAuthors(s,f) \land (highFrequencyOfCommits(s,f) \lor \text{avgFrequencyOfCommits}(s,f)) \implies \text{isBuggy}(s,f)

### 5.2.4 Data Facts (Fact Base)

The db file that constitutes the training fact base contains all the true groundings of the predicates declared in the mln file. We use a closed world assumption i.e. any ground atoms not defined in the database file assumed to be False. Table-10 shows a snippet of the db file of one of the projects we tested in the thesis.

<table>
<thead>
<tr>
<th>Groundings</th>
</tr>
</thead>
<tbody>
<tr>
<td>lowFrequencyOfCommits(SID2,fid3)</td>
</tr>
<tr>
<td>lowFrequencyOfCommits(SID4,fid21)</td>
</tr>
<tr>
<td>avgFrequencyOfCommits(SID27,fid7)</td>
</tr>
<tr>
<td>avgFrequencyOfCommits(SID41,fid3)</td>
</tr>
<tr>
<td>avgFrequencyOfCommits(SID42,fid2)</td>
</tr>
<tr>
<td>highFrequencyOfCommits(SID39,fid44)</td>
</tr>
<tr>
<td>highFrequencyOfCommits(SID40,fid1)</td>
</tr>
<tr>
<td>lowFileChurn(SID4,fid21)</td>
</tr>
<tr>
<td>lowFileChurn(SID7,fid11)</td>
</tr>
<tr>
<td>avgFileChurn(SID18,fid7)</td>
</tr>
<tr>
<td>highFileChurn(SID8,fid18)</td>
</tr>
<tr>
<td>lowFrequencyOfMerges(SID2,fid3)</td>
</tr>
<tr>
<td>lowFrequencyOfMerges(SID3,fid2)</td>
</tr>
<tr>
<td>avgFrequencyOfMerges(SID9,fid18)</td>
</tr>
<tr>
<td>highFrequencyOfMerges(SID24,fid21)</td>
</tr>
<tr>
<td>highFrequencyOfMerges(SID29,fid44)</td>
</tr>
<tr>
<td>lowOverallStrength(SID2,fid3)</td>
</tr>
<tr>
<td>avgOverallStrength(SID3,fid17)</td>
</tr>
<tr>
<td>highOverallStrength(SID5,fid17)</td>
</tr>
<tr>
<td>lowFailureIntensity(SID3,fid13)</td>
</tr>
<tr>
<td>avgFailureIntensity(SID1685,fid7)</td>
</tr>
<tr>
<td>highFailureIntensity(SID602,fid2)</td>
</tr>
<tr>
<td>lowFractalValue(SID157,fid3)</td>
</tr>
<tr>
<td>lowFractalValue(SID158,fid44)</td>
</tr>
<tr>
<td>avgFractalValue(SID2,fid3)</td>
</tr>
<tr>
<td>highFractalValue(SID19,fid1)</td>
</tr>
<tr>
<td>lowDistinctAuthors(SID3,fid11)</td>
</tr>
<tr>
<td>lowDistinctAuthors(SID4,fid21)</td>
</tr>
<tr>
<td>avgDistinctAuthors(SID531,fid44)</td>
</tr>
<tr>
<td>highDistinctAuthors(SID48,fid13)</td>
</tr>
<tr>
<td>\text{isBuggy}(SID965,fid2)</td>
</tr>
<tr>
<td>\text{isBuggy}(SID5826,fid11)</td>
</tr>
</tbody>
</table>

**Table 10: Example db file**
5.2.5 Weight Learning

The training function of MLN described in Section 2.5.5 performs two tasks. It converts all the rules into its Conjunctive Normal Form and gives each subset of the rule an associated weight based on its importance. The individual non-evidence predicates are also given an associated weight so that their probability doesn’t default to 0.5. These weights are stored in an output MLN file, which is one of the inputs that the MLN Inference module needs. Table-11 shows an example output file for an MLN file with three random rules selected from the rule repository.

```
// predicate declarations
lowFailureIntensity(sid, fid)
avgFailureIntensity(sid, fid)
avgFileChurn(sid, fid)
highFractalValue(sid, fid)
highOverallStrength(sid, fid)
lowFrequencyOfMerges(sid, fid)
avgOverallStrength(sid, fid)
avgFrequencyOfMerges(sid, fid)
highDistinctAuthors(sid, fid)
lowFractalValue(sid, fid)
avgDistinctAuthors(sid, fid)
avgFractalValue(sid, fid)
highFrequencyOfCommits(sid, fid)
highFailureIntensity(sid, fid)
lowFileChurn(sid, fid)
lowOverallStrength(sid, fid)
lowFrequencyOfCommits(sid, fid)
highFileChurn(sid, fid)
isBuggy(sid, fid)
lowDistinctAuthors(sid, fid)
avgFrequencyOfCommits(sid, fid)
highFrequencyOfMerges(sid, fid)

// -0.335496  (highFrequencyOfCommits(s, f) v avgFrequencyOfCommits(s, f)) ^
highFailureIntensity(s, f) ^ highOverallStrength(s, f) => isBuggy(s, f)
-0.335496  !highFrequencyOfCommits(a1, f1) v !highOverallStrength(a1, f1) v
!highFailureIntensity(a1, f1) v isBuggy(a1, f1)
0       !avgFrequencyOfCommits(a1, f1) v !highOverallStrength(a1, f1) v
!highFailureIntensity(a1, f1) v isBuggy(a1, f1)
```
Chapter 5: Fuzzy Logic and Markov Logic Network Knowledge Base

// -0.905691  (highFrequencyOfCommits(s, f) v avgFrequencyOfCommits(s, f)) ^
highFailureIntensity(s, f) ^ (highOverallStrength(s, f) v avgOverallStrength(s, f)) ^
(highFileChurn(s, f) v avgFileChurn(s, f)) => isBuggy(s, f)
0.271075  !highFileChurn(al, f1) v !highFrequencyOfCommits(al, f1) v
!highOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
-1.4407  !highFileChurn(al, f1) v !avgFrequencyOfCommits(al, f1) v
!highOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
-0.443598  !highFileChurn(al, f1) v !highFrequencyOfCommits(al, f1) v
!avgOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
-0.62258  !highFileChurn(al, f1) v !avgFrequencyOfCommits(al, f1) v
!avgOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
0.0495548  !avgFileChurn(al, f1) v !highFrequencyOfCommits(al, f1) v
!highOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
0.378934  !avgFileChurn(al, f1) v !avgFrequencyOfCommits(al, f1) v
!avgOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
0.941439  !avgFileChurn(al, f1) v !highFrequencyOfCommits(al, f1) v
!avgOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)
-0.0398144  !avgFileChurn(al, f1) v !avgFrequencyOfCommits(al, f1) v
!avgOverallStrength(al, f1) v !highFailureIntensity(al, f1) v isBuggy(al, f1)

// 0.248469  !(highFailureIntensity(s, f) v highFrequencyOfMerges(s, f) v
highOverallStrength(s, f) v highFrequencyOfCommits(s, f) v highFileChurn(s)) =>
!isBuggy(s, f)
0.248469  highFileChurn(al, f1) v highFrequencyOfCommits(al, f1) v
highOverallStrength(al, f1) v highFrequencyOfMerges(al, f1) v highFailureIntensity(al, f1) v !isBuggy(al, f1)

Table 11: Example Output MLN File

5.2.6 Discussion

Markov Logic Networks (MLNs) presents a common ground between Bayesian inferencing and Machine Learning. More specifically, rules in MLNs present the expert knowledge for a given domain, facts represent the training data (during the training phase), and the actual data for which the reasoning will be applied on (during the testing or execution phase), while the training process allocates weights to each rule. The higher the weight, the higher the importance of the rule given the specific training data set. The rules are trained on a subset of data represented as facts. Once the rules are trained, they can be applied to the actual data. The reasoning takes into account the weights and based on the actual data; deductions hold with a certain level of confidence (probability).
In the context of this thesis, the expert knowledge is encoded in the form of rules, and facts in the form of ground predicates. We have identified twenty rules which can be customized per project. The facts are generated by a populator module that processes data stored in the data model and generates facts according to its encoded logic. For example, if the frequency a file \( f_1 \) is committed in a segment \( s_1 \), is two standard deviations above the mean value of the frequency the file is committed overall then, the predicate \( highFrequencyOfCommits(a_1, f_1) \) is added to the fact base as True. In the next chapter, we discuss in detail the obtained results by applying both Fuzzy reasoning and MLN reasoning.
Chapter 6: Experiments and Results

We have implemented six different strategies to predict fault-prone segments using the two rule-based techniques. The strategies remain the same for both Fuzzy Logic and Markov Logic Networks, but the implementation is different for each of them. The differences are explained in detail in the following sections.

The data is split into training and testing datasets with the training set being the last 1-3 years of the total historical data as tabulated in Table-3. Because the testing dataset is measured in years, the percentage of testing data over the total data is slightly different for different projects. For example, the testing dataset of akregator is 19% of the total data while elisa has a testing dataset of 27% even though both the testing sets are for the last year of the project. This is due to the different frequencies of commits observed in different projects. Once most of the development is done on a project, the number of commits can decrease because it gets more stable.

We extracted the testing set by first taking the last year of data, and then adding another year of data if the number of commits in the last year were not enough to have meaningful results. It is a bit hard to define when we can say that the data gives meaningful results, but overall, we wanted the testing set to be 20% of the complete data and to have at least 10% of the total faulty segments. Gwenview and Juk have three years' worth of data in the testing set because the frequency of commits in the last three years declined for them. Kdelibs and kio-extras have an approximately 55-45 split between training and testing due to the overall frequency of commits available being low. Additionally, in the tables shown in the below sections, the averages for precision and F1 score are not displayed because of the wide variation in their values brought upon by the imbalanced data.

6.1 Strategies

In this section, we explain each strategy in detail.

**Strategy-1:** The purpose of this strategy is to identify whether there exists a collection of rules that can be applied to historical data of the project and this set can apply equally well in the testing data (and in any future data). More specifically, for each project $P_i$ we identify a subset $R_i$ from the twenty rules $R_1, R_2, ... , R_{20}$ from the rule repository (See Section 4.5) which gives the best results
when applied on a subset of the process data (i.e. the training data) extracted from GitHub. The definition of best results is outlined in Section 6.2. Once such a best rule subset is selected, then we apply this rule subset to the remaining process data (i.e. testing data) and we evaluate whether these rules perform equally well on the testing dataset.

**Strategy-2:** This strategy examines whether adding customized rules for a project on top of the optimized rule subset selected in Strategy-1 will help the model perform better on the testing data. More formally, if a project $P_i$ has an optimal rule subset $R_i$ as identified in Strategy-1, then in Strategy-2 we will add on this set additional rules $R_{k}, R_{k+1}, \ldots, R_{k+n}$ that enhance accuracy and recall in the training set, and then perform inference on the testing data using this expanded ruleset. More specifically, after Strategy-1 selects the rules that perform the best on the training dataset, we use the heuristic technique outlined in Section 4.6 to further create custom rules that can be added to the existing ruleset to increase the recall in the training dataset. Once this expanded ruleset is compiled, we apply it to the testing data and collect the results.

**Strategy-3:** This strategy evaluates whether frequently applied rules on the training data across all projects, can form a common set of rules that may consistently yield good results when applied to the testing data in all projects. More specifically, by using Strategy-1, for each project $P_i$ we have obtained the best performing ruleset $R_i$. We then select all rules $r_k$ in $R_1, R_2, \ldots, R_{20}$ which have been used collectively more than ten times across all eighteen projects. Let us call this set $R_{frequent}$. We then evaluate whether the ruleset $R_{frequent}$ can yield good results when applied to the testing data across all projects $P_1, P_2, \ldots, P_{18}$.

**Strategy-4:** In this strategy, the objective is to assess whether the rules obtained in Strategy-2 for each project perform equally well over time in the testing set. More specifically, we split the testing set into two equal parts. We then evaluate whether the rules obtained in Strategy-2 perform equally well in each of these parts. For example, if project $P_i$ has an applicable optimized ruleset $R_i$, we divide the testing set of $P_i$ into $Test_1$ and $Test_2$ and apply $R_i$ on both the sets. We select the ruleset using Strategy-2 because it provides the best balance between accuracy and recall.

**Strategy-5:** This strategy evaluates whether a ruleset that performs well in one project can be used equally well in other projects. More specifically, after we obtain a ruleset $R_i$ that yields the best results for project $P_i$ (i.e. using Strategy 1), we apply this ruleset $R_i$ to all other projects $P_k$ where
\[ k \neq i. \] We then obtain the average performance scores by applying each such rule set \( R_i \) to all other projects \( P_k \) \((k \neq i)\).

**Strategy-6:** This strategy studies the feasibility of creating customized rules for each project without starting from an optimized ruleset. More specifically, for every project \( P_i \), we create a custom ruleset \( R_i \) using the technique outlined in Section 4.6 yielding rules \( R_n, R_{n+1}, \ldots, R_{(n+m)} \) that provide the best recall in the training dataset \( Train_i \) of \( P_i \). We then use the ruleset \( R_i \) on the testing set \( Test_i \) of \( P_i \).

### 6.2 Strategy-1: Best Subset of Rules

**Approach:** The purpose of this strategy is to identify whether there exists a collection of rules that can be applied to historical data of the project and applies equally well in the testing data (and in any future data). More specifically, for each project \( P_i \) we identify a subset \( R_i \) from the twenty rules \( R_1, R_2, \ldots, R_{20} \) from the rule repository (See Section 4.5) which gives the best results when applied on a subset of the process data (i.e. the training data) extracted from GitHub. The definition of best results is outlined in Section 6.2. Once such a best rule subset is selected, then we apply this rule subset to the remaining process data (i.e. testing data) and we evaluate whether these rules perform equally well on the testing dataset.

**Fuzzy Logic-**

The training algorithm prescribed in Section 5.1.6 is used to select the best subset of rules that describe the training dataset of a project. A brute force algorithm is used that runs every possible combination of the twenty rules and the combination that gives the highest sum of accuracy and recall is chosen as the best subset. After the training is completed and the rules are picked, we perform inference on the testing dataset. The results are tabulated in Table-12.

**Markov Logic Networks-**

The MLN model uses a built-in training module to give weights to the rules. There are two training algorithms available to use: discriminative and generative. The current model uses discriminative learning and its technique is outlined in Section 2.5.5. The initial MLN file given to the model will contain all the rules in the rule repository. After the training, all the rules that have negative weights are removed. A rule with negative weight implies negative correlation and while they can still be
used for inference, the results obtained after removing the negative rules were much better. During the training stage, the model converts each rule to its Conjunctive Normal Form (CNF) and trains each partial rule separately. Therefore, unlike Fuzzy Logic which has the entire rule either included or excluded in the final subset, MLNs can have partial rules included in the final output file. Table-13 shows the results obtained by the MLN by using both negative rules and positive rules; and using positive rules only.

Table 12: Fuzzy Logic Results using rule subset

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Negative Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.62</td>
<td>0.50</td>
<td>0.63</td>
<td>0.08</td>
<td>0.13</td>
</tr>
<tr>
<td>ark</td>
<td>0.51</td>
<td>0.80</td>
<td>0.43</td>
<td>0.25</td>
<td>0.38</td>
</tr>
<tr>
<td>elisa</td>
<td>0.62</td>
<td>0.56</td>
<td>0.32</td>
<td>0.38</td>
<td>0.45</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.62</td>
<td>0.76</td>
<td>0.57</td>
<td>0.39</td>
<td>0.51</td>
</tr>
<tr>
<td>juk</td>
<td>0.56</td>
<td>0.67</td>
<td>0.55</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>k3b</td>
<td>0.74</td>
<td>0.64</td>
<td>0.75</td>
<td>0.03</td>
<td>0.06</td>
</tr>
<tr>
<td>kate</td>
<td>0.51</td>
<td>0.55</td>
<td>0.51</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.8</td>
<td>0.42</td>
<td>0.85</td>
<td>0.25</td>
<td>0.31</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.7</td>
<td>0.66</td>
<td>0.75</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>kmix</td>
<td>0.52</td>
<td>1.00</td>
<td>0.49</td>
<td>0.10</td>
<td>0.17</td>
</tr>
<tr>
<td>kompare</td>
<td>0.6</td>
<td>1.00</td>
<td>0.55</td>
<td>0.05</td>
<td>0.10</td>
</tr>
<tr>
<td>konsole</td>
<td>0.52</td>
<td>0.80</td>
<td>0.45</td>
<td>0.25</td>
<td>0.38</td>
</tr>
<tr>
<td>konversation</td>
<td>0.47</td>
<td>0.67</td>
<td>0.43</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.82</td>
<td>1.00</td>
<td>0.81</td>
<td>0.23</td>
<td>0.37</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.53</td>
<td>0.60</td>
<td>0.32</td>
<td>0.20</td>
<td>0.30</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.63</td>
<td>0.69</td>
<td>0.63</td>
<td>0.23</td>
<td>0.34</td>
</tr>
<tr>
<td>solid</td>
<td>0.9</td>
<td>0.43</td>
<td>0.91</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.87</td>
<td>0.20</td>
<td>0.94</td>
<td>0.25</td>
<td>0.22</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.64</strong></td>
<td><strong>0.66</strong></td>
<td><strong>0.60</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 13: MLN Results using trained rule subset

<table>
<thead>
<tr>
<th>Project</th>
<th>All Rules</th>
<th></th>
<th></th>
<th>Only Positive Rules</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Accuracy</td>
<td>Recall</td>
<td>Negative</td>
<td>Precision</td>
<td>F1 Score</td>
</tr>
<tr>
<td>akregator</td>
<td>0.7</td>
<td>1</td>
<td>0.85</td>
<td>1</td>
<td>0.84</td>
<td>0.29</td>
<td>0.45</td>
</tr>
<tr>
<td>ark</td>
<td>0.8</td>
<td>0.5</td>
<td>0.76</td>
<td>0.6</td>
<td>0.81</td>
<td>0.46</td>
<td>0.52</td>
</tr>
<tr>
<td>elisa</td>
<td>0.63</td>
<td>0.625</td>
<td>0.62</td>
<td>0.65</td>
<td>0.61</td>
<td>0.39</td>
<td>0.49</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.68</td>
<td>0.38</td>
<td>0.68</td>
<td>0.55</td>
<td>0.73</td>
<td>0.42</td>
<td>0.48</td>
</tr>
<tr>
<td>juk</td>
<td>0.74</td>
<td>0.34</td>
<td>0.78</td>
<td>1</td>
<td>0.77</td>
<td>0.22</td>
<td>0.36</td>
</tr>
<tr>
<td>k3b</td>
<td>0.55</td>
<td>0.64</td>
<td>0.7</td>
<td>0.79</td>
<td>0.7</td>
<td>0.03</td>
<td>0.06</td>
</tr>
<tr>
<td>kate</td>
<td>0.65</td>
<td>0.27</td>
<td>0.76</td>
<td>0.36</td>
<td>0.79</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.51</td>
<td>0.5</td>
<td>0.72</td>
<td>0.58</td>
<td>0.73</td>
<td>0.2</td>
<td>0.30</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.61</td>
<td>0.56</td>
<td>0.81</td>
<td>0.55</td>
<td>0.83</td>
<td>0.24</td>
<td>0.33</td>
</tr>
<tr>
<td>kmix</td>
<td>0.8</td>
<td>0.5</td>
<td>0.72</td>
<td>1</td>
<td>0.71</td>
<td>0.15</td>
<td>0.26</td>
</tr>
<tr>
<td>kompare</td>
<td>0.47</td>
<td>1</td>
<td>0.395</td>
<td>1</td>
<td>0.38</td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>konsole</td>
<td>0.73</td>
<td>0.71</td>
<td>0.71</td>
<td>0.66</td>
<td>0.72</td>
<td>0.36</td>
<td>0.47</td>
</tr>
<tr>
<td>konversation</td>
<td>0.7</td>
<td>0.5</td>
<td>0.8</td>
<td>0.67</td>
<td>0.83</td>
<td>0.4</td>
<td>0.50</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.83</td>
<td>1</td>
<td>0.85</td>
<td>1</td>
<td>0.85</td>
<td>0.26</td>
<td>0.41</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.7</td>
<td>0.48</td>
<td>0.72</td>
<td>0.48</td>
<td>0.76</td>
<td>0.3</td>
<td>0.37</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.8</td>
<td>0.46</td>
<td>0.72</td>
<td>0.77</td>
<td>0.71</td>
<td>0.3</td>
<td>0.43</td>
</tr>
<tr>
<td>solid</td>
<td>0.85</td>
<td>0.57</td>
<td>0.86</td>
<td>0.64</td>
<td>0.87</td>
<td>0.1</td>
<td>0.17</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.51</td>
<td>0.8</td>
<td>0.58</td>
<td>0.4</td>
<td>0.6</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.68</strong></td>
<td><strong>0.60</strong></td>
<td><strong>0.72</strong></td>
<td><strong>0.71</strong></td>
<td><strong>0.74</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.2.1 Discussion

For this strategy, we have applied the 20 generic rules on a subset of each project’s data in order to obtain a subset of the 20 rules which yield the best recall and accuracy scores for this specific project the rules are applied on. We then use this subset of rules on the testing data. Fuzzy Logic
Chapter 6: Experiments and Results

gives an average accuracy of 64%, recall of 66%, and negative recall of 60%. SystemSettings, kdelibs, and solid gave sub-par results with a recall of 20%, 42%, and 43% respectively. However, all three projects gave high accuracy which leads us to the conclusion that the training module described in Section 5.1.6 prioritized improving accuracy over recall.

We notice that removing rules with negative weights in MLNs gives better overall results. There was an increase of 4% in accuracy, 11% in recall, and 2% in negative recall, which suggests that this approach of removing negative rules can be continued for other strategies too. Overall, MLNs outperformed Fuzzy Logic with an average increase of 8% in accuracy, 5% in recall, and 14% in negative recall. The projects that gave lower recall in Fuzzy Logic all gave better recall in MLN but had a steep decrease in accuracy. However, the projects gwenview, juk, and kate had a decrease in recall compared to Fuzzy Logic. One of the reasons for the disparity in performance metrics is the fact that MLN breaks each rule in parts when converting it to CNF form and gives individual weights to each part. So, it supports the inclusion of parts of a rule in the final subset, unlike Fuzzy Logic.

6.3 Strategy-2: Expanded Rules

*Approach:* This strategy examines whether adding customized rules for a project on top of the optimized rule subset selected in Strategy-1 will help the model perform better on the testing data. More formally, if a project $P_i$ has an optimal rule subset $R_i$, we will add additional rules $R_k, R_{k+1}, ..., R_{k+n}$ that better describe the training set on it and perform inference on a subset of process data (i.e. testing data) using the expanded rules to report the performance metrics on it. More specifically, after Strategy-1 selects the rules that perform the best on the training dataset, we use the heuristic technique outlined in Section 4.6 to further create custom rules that can be added to the existing ruleset to increase the recall in the training dataset. Once this expanded ruleset is compiled, we apply it to the testing data and collect the results.

**Fuzzy Logic**-

The Fuzzy Logic model uses Mamdani’s Inference to predict fault-prone segments. In this approach, the rules selected in Strategy-1 are used and inference is performed on the training set of the selected project. Then, the custom utility program described in Section 4.6 is used to find
all the False Negative and False Positive segments. The files are analyzed, and custom rules are added to the existing subset to better describe the training dataset. After the rules are finalized, inference is performed using the expanded rules on the testing dataset. The results of performing the inference are shown in Table-14. The rules that were used for each project are outlined in Appendix B1.

Table 14: Fuzzy Logic Results using Expanded Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Negative Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.47</td>
<td>1</td>
<td>0.44</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>ark</td>
<td>0.47</td>
<td>0.8</td>
<td>0.38</td>
<td>0.26</td>
<td>0.51</td>
</tr>
<tr>
<td>elisa</td>
<td>0.60</td>
<td>0.65</td>
<td>0.59</td>
<td>0.37</td>
<td>0.44</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.59</td>
<td>0.83</td>
<td>0.5</td>
<td>0.37</td>
<td>0.52</td>
</tr>
<tr>
<td>juk</td>
<td>0.46</td>
<td>0.67</td>
<td>0.45</td>
<td>0.07</td>
<td>0.14</td>
</tr>
<tr>
<td>k3b</td>
<td>0.72</td>
<td>0.79</td>
<td>0.72</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>kate</td>
<td>0.35</td>
<td>0.55</td>
<td>0.33</td>
<td>0.05</td>
<td>0.09</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.77</td>
<td>0.5</td>
<td>0.8</td>
<td>0.23</td>
<td>0.46</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.70</td>
<td>0.66</td>
<td>0.7</td>
<td>0.18</td>
<td>0.35</td>
</tr>
<tr>
<td>kmix</td>
<td>0.52</td>
<td>1</td>
<td>0.49</td>
<td>0.1</td>
<td>0.19</td>
</tr>
<tr>
<td>kompare</td>
<td>0.60</td>
<td>1</td>
<td>0.55</td>
<td>0.05</td>
<td>0.1</td>
</tr>
<tr>
<td>konsole</td>
<td>0.52</td>
<td>0.8</td>
<td>0.45</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td>konversation</td>
<td>0.41</td>
<td>0.83</td>
<td>0.34</td>
<td>0.18</td>
<td>0.36</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.82</td>
<td>1</td>
<td>0.81</td>
<td>0.23</td>
<td>0.45</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.45</td>
<td>0.63</td>
<td>0.41</td>
<td>0.18</td>
<td>0.36</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.64</td>
<td>0.77</td>
<td>0.61</td>
<td>0.24</td>
<td>0.47</td>
</tr>
<tr>
<td>solid</td>
<td>0.87</td>
<td>0.43</td>
<td>0.88</td>
<td>0.07</td>
<td>0.15</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.87</td>
<td>0.2</td>
<td>0.94</td>
<td>0.25</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.6</strong></td>
<td><strong>0.73</strong></td>
<td><strong>0.58</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Markov Logic Networks -

The same rules that were added in the Fuzzy Logic model were added to the MLN model. The built-in discriminative training was first used on the rules from the rule repository to filter out the rules with negative weights. Then, the new custom rules were added to the MLN file for each project and the weights were trained again. Inference is then performed on the testing dataset and the results are tabulated in Table-15. We declare all the rules used for a project in Appendix B2.

### Table 15: MLN Results for Expanded Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Neg. Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.85</td>
<td>1</td>
<td>0.84</td>
<td>0.29</td>
<td>0.45</td>
</tr>
<tr>
<td>ark</td>
<td>0.72</td>
<td>0.6</td>
<td>0.76</td>
<td>0.4</td>
<td>0.48</td>
</tr>
<tr>
<td>elisa</td>
<td>0.57</td>
<td>0.71</td>
<td>0.51</td>
<td>0.35</td>
<td>0.47</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.61</td>
<td>0.71</td>
<td>0.58</td>
<td>0.38</td>
<td>0.50</td>
</tr>
<tr>
<td>juk</td>
<td>0.78</td>
<td>1</td>
<td>0.77</td>
<td>0.22</td>
<td>0.36</td>
</tr>
<tr>
<td>k3b</td>
<td>0.91</td>
<td>0.71</td>
<td>0.91</td>
<td>0.1</td>
<td>0.18</td>
</tr>
<tr>
<td>kate</td>
<td>0.76</td>
<td>0.36</td>
<td>0.78</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.53</td>
<td>0.67</td>
<td>0.52</td>
<td>0.14</td>
<td>0.23</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.74</td>
<td>0.59</td>
<td>0.75</td>
<td>0.19</td>
<td>0.29</td>
</tr>
<tr>
<td>kmix</td>
<td>0.72</td>
<td>1</td>
<td>0.71</td>
<td>0.15</td>
<td>0.26</td>
</tr>
<tr>
<td>kompare</td>
<td>0.4</td>
<td>1</td>
<td>0.38</td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>konsole</td>
<td>0.70</td>
<td>0.69</td>
<td>0.7</td>
<td>0.35</td>
<td>0.46</td>
</tr>
<tr>
<td>konversation</td>
<td>0.66</td>
<td>0.67</td>
<td>0.66</td>
<td>0.25</td>
<td>0.36</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.85</td>
<td>1</td>
<td>0.85</td>
<td>0.26</td>
<td>0.41</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.62</td>
<td>0.59</td>
<td>0.63</td>
<td>0.25</td>
<td>0.35</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.69</td>
<td>0.77</td>
<td>0.67</td>
<td>0.27</td>
<td>0.40</td>
</tr>
<tr>
<td>solid</td>
<td>0.76</td>
<td>0.64</td>
<td>0.78</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.36</td>
<td>0.6</td>
<td>0.34</td>
<td>0.08</td>
<td>0.14</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>0.68</td>
<td>0.74</td>
<td>0.67</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
6.3.1 Discussion

This strategy can be used to answer RQ4, which is “What is the average increase or decrease of using expanded rules on top of the chosen rule subset?”. Fuzzy Logic saw an average decrease in Accuracy of 7% with the highest decrease seen in *kate* and an average decrease of 2% in Negative Recall. On the flip side, however, the average recall increased by 12% which shows that adding expanded rules had a net positive effect. The biggest increases in recall were observed for the projects akregator, konversation, k3b, and plasma-nm.

MLN also saw a similar trend. It had an average drop of 6% in Accuracy, 8% in Negative Recall, and an increase of 7% in recall. It is clear that the choice of using expanded rules almost always comes with a decrease in Accuracy. The primary reason behind this is that most rules added on top of the existing rules were positive class identifying rules which increases the number of False Positives. We experimented with adding more negative class rules, but they almost always reduced Recall significantly, which is why they were used sparsely. In the case of fault-prone file detection, the impact of False Negatives is far worse for the system than False Positives. Since the decrease in accuracy isn’t drastic, we can conclude that this Strategy gives superior performance over Strategy-1.

6.4 Strategy-3: Common Rules

*Approach:* This strategy evaluates whether frequently applied rules on the training data across all projects, can form a common set of rules that may consistently yield good results when applied to the testing data in all projects. More specifically, by using Strategy-1, for each project $P_i$ we have obtained the best performing ruleset $R_i$. We then select all rules $r_k$ in $R_1, R_2, \ldots, R_{20}$ which have been used collectively more than ten times across all eighteen projects. Let us call this set $R_{\text{frequent}}$. We then evaluate whether the ruleset $R_{\text{frequent}}$ can yield good results when applied to the testing data across all projects $P_1, P_2, \ldots, P_{18}$.

*Fuzzy Logic:*

The rules from the Rule Repository that were used most frequently in Strategy-1 were selected and run for the testing datasets of all projects. The results reported after performing inference using the common rules are reported in Table-16.
Markov Logic Networks:

The common rules that were selected were all given equal positive weights and the same MLN output file was used to run inference on all projects. The weights were not trained for each project because this Strategy was primarily to observe the results that are reported when no training is performed on either model. The results are organized in Table-17.

Table 16: Fuzzy Logic Results for Common Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Negative Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.53</td>
<td>1</td>
<td>0.5</td>
<td>0.11</td>
<td>0.2</td>
</tr>
<tr>
<td>ark</td>
<td>0.47</td>
<td>0.8</td>
<td>0.38</td>
<td>0.26</td>
<td>0.39</td>
</tr>
<tr>
<td>elisa</td>
<td>0.45</td>
<td>0.73</td>
<td>0.35</td>
<td>0.3</td>
<td>0.42</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.55</td>
<td>0.79</td>
<td>0.47</td>
<td>0.34</td>
<td>0.48</td>
</tr>
<tr>
<td>juk</td>
<td>0.44</td>
<td>1</td>
<td>0.4</td>
<td>0.1</td>
<td>0.18</td>
</tr>
<tr>
<td>k3b</td>
<td>0.58</td>
<td>0.79</td>
<td>0.57</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>kate</td>
<td>0.33</td>
<td>0.73</td>
<td>0.3</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.75</td>
<td>0.58</td>
<td>0.76</td>
<td>0.23</td>
<td>0.33</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.63</td>
<td>0.82</td>
<td>0.61</td>
<td>0.17</td>
<td>0.28</td>
</tr>
<tr>
<td>kmix</td>
<td>0.32</td>
<td>1</td>
<td>0.28</td>
<td>0.07</td>
<td>0.13</td>
</tr>
<tr>
<td>kompare</td>
<td>0.47</td>
<td>1</td>
<td>0.45</td>
<td>0.04</td>
<td>0.08</td>
</tr>
<tr>
<td>konsole</td>
<td>0.52</td>
<td>0.8</td>
<td>0.45</td>
<td>0.25</td>
<td>0.38</td>
</tr>
<tr>
<td>konversation</td>
<td>0.41</td>
<td>0.83</td>
<td>0.34</td>
<td>0.18</td>
<td>0.29</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.54</td>
<td>1</td>
<td>0.52</td>
<td>0.1</td>
<td>0.19</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.47</td>
<td>0.59</td>
<td>0.44</td>
<td>0.18</td>
<td>0.27</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.27</td>
<td>0.85</td>
<td>0.18</td>
<td>0.14</td>
<td>0.24</td>
</tr>
<tr>
<td>solid</td>
<td>0.70</td>
<td>0.64</td>
<td>0.7</td>
<td>0.05</td>
<td>0.09</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.42</td>
<td>0.6</td>
<td>0.4</td>
<td>0.09</td>
<td>0.16</td>
</tr>
<tr>
<td>Average</td>
<td>0.49</td>
<td>0.81</td>
<td>0.45</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 17: MLN Results for Common Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Neg Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.91</td>
<td>1</td>
<td>0.9</td>
<td>0.4</td>
<td>0.57</td>
</tr>
<tr>
<td>ark</td>
<td>0.62</td>
<td>0.8</td>
<td>0.57</td>
<td>0.34</td>
<td>0.47</td>
</tr>
<tr>
<td>elisa</td>
<td>0.63</td>
<td>0.56</td>
<td>0.65</td>
<td>0.38</td>
<td>0.45</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.71</td>
<td>0.62</td>
<td>0.74</td>
<td>0.46</td>
<td>0.53</td>
</tr>
<tr>
<td>juk</td>
<td>0.68</td>
<td>1</td>
<td>0.66</td>
<td>0.16</td>
<td>0.27</td>
</tr>
<tr>
<td>k3b</td>
<td>0.85</td>
<td>0.71</td>
<td>0.85</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>kate</td>
<td>0.77</td>
<td>0.45</td>
<td>0.79</td>
<td>0.11</td>
<td>0.18</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.79</td>
<td>0.67</td>
<td>0.8</td>
<td>0.29</td>
<td>0.4</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.73</td>
<td>0.7</td>
<td>0.73</td>
<td>0.31</td>
<td>0.31</td>
</tr>
<tr>
<td>kmix</td>
<td>0.82</td>
<td>0.75</td>
<td>0.83</td>
<td>0.19</td>
<td>0.3</td>
</tr>
<tr>
<td>kompare</td>
<td>0.58</td>
<td>1</td>
<td>0.57</td>
<td>0.05</td>
<td>0.1</td>
</tr>
<tr>
<td>konsole</td>
<td>0.83</td>
<td>0.51</td>
<td>0.9</td>
<td>0.56</td>
<td>0.53</td>
</tr>
<tr>
<td>konversation</td>
<td>0.66</td>
<td>0.67</td>
<td>0.66</td>
<td>0.25</td>
<td>0.36</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.78</td>
<td>1</td>
<td>0.77</td>
<td>0.19</td>
<td>0.32</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.77</td>
<td>0.48</td>
<td>0.83</td>
<td>0.37</td>
<td>0.42</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.72</td>
<td>0.54</td>
<td>0.75</td>
<td>0.25</td>
<td>0.34</td>
</tr>
<tr>
<td>solid</td>
<td>0.88</td>
<td>0.5</td>
<td>0.89</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.75</td>
<td>0.4</td>
<td>0.78</td>
<td>0.15</td>
<td>0.22</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.75</strong></td>
<td><strong>0.69</strong></td>
<td><strong>0.76</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.4.1 Discussion

This strategy evaluates whether frequently applied rules on the training data across all projects can form a common set of rules that may consistently yield good results when applied to the testing data in all projects. We use the results from this strategy to provide answers to RQ1, “Is there a common set of rules that can be applied to all projects for predicting fault-prone segments?”. In Fuzzy Logic, we choose the rules that were used the most frequently and give them as input to all projects. The results are tabulated in Table-16, and it can be noticed that
using common rules is good for recall but bad for accuracy and negative recall. The average accuracy dropped by 15% when compared to Strategy-1 and recall increased by 15%. Plasma-nm, kmix, and kate saw the worst drops in accuracy while kio-extras and solid had a better balance between recall and accuracy compared to previous strategies.

MLN had more stable results than Fuzzy Logic with an average accuracy of 75% and recall of 69%. The difference in the rules selected, along with the advanced inference used in MLN is theorized to be the cause of the varied results. It had the highest accuracy among previous strategies but suffered from having a relatively lower recall. Because this approach does not require any training for both Fuzzy Logic and MLN, it is quite an attractive method of prediction on projects that are still new and do not have enough historical repository data. There is still more research needed on what rules would better fit in the common rules database, but we believe these results are a step in the right direction.

6.5 Strategy-4: Rules Over Time

Approach: In this strategy, the objective is to assess whether the rules obtained in Strategy-2 for each project perform equally well over time in the testing set. More specifically, we split the testing set into two equal parts. We then evaluate whether the rules obtained in Strategy-2 perform equally well in each of these parts. For example, if project $P_i$ has an applicable optimized ruleset $R_i$, we divide the testing set of $P_i$ into $Test_1$ and $Test_2$ and apply $R_i$ on both the sets. We select the ruleset using Strategy-2 because it provides the best balance between accuracy and recall.

Fuzzy Logic:

The projects whose testing sets could be divided into two without either set being too sparse were chosen and their testing dataset divided. Because Strategy-2 (Expanded Rules) gave the best results, inference was performed using the expanded rules on both sets to measure how well the rules hold over time. The projects that were chosen and the comparison between the two sets are shown in Table-18.

Markov Logic Networks:

The same projects that were chosen for Fuzzy Logic were also chosen in MLNs to observe how well the rules hold over time. The best strategy that worked for MLN was also Strategy-2
(Expanded Rules), so the same rules were used for inference. The results are tabulated in Table 19.

**Table 18: Fuzzy Logic Results over time**

<table>
<thead>
<tr>
<th>Project</th>
<th>First Half</th>
<th></th>
<th></th>
<th></th>
<th>Second Half</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Neg. Recall</td>
<td>Precision</td>
<td>Accuracy</td>
<td>Recall</td>
<td>Neg. Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>ark</td>
<td>0.47</td>
<td>0.86</td>
<td>0.39</td>
<td>0.24</td>
<td>0.4</td>
<td>1</td>
<td>0.25</td>
<td>0.25</td>
</tr>
<tr>
<td>elisa</td>
<td>0.66</td>
<td>0.59</td>
<td>0.68</td>
<td>0.3</td>
<td>0.55</td>
<td>0.6</td>
<td>0.5</td>
<td>0.48</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.65</td>
<td>0.6</td>
<td>0.67</td>
<td>0.47</td>
<td>0.6</td>
<td>1</td>
<td>0.53</td>
<td>0.27</td>
</tr>
<tr>
<td>k3b</td>
<td>0.63</td>
<td>0.5</td>
<td>0.63</td>
<td>0.01</td>
<td>0.63</td>
<td>0.9</td>
<td>0.62</td>
<td>0.07</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.73</td>
<td>0.6</td>
<td>0.74</td>
<td>0.12</td>
<td>0.63</td>
<td>0.76</td>
<td>0.61</td>
<td>0.22</td>
</tr>
<tr>
<td>konsole</td>
<td>0.54</td>
<td>0.85</td>
<td>0.46</td>
<td>0.27</td>
<td>0.45</td>
<td>0.57</td>
<td>0.42</td>
<td>0.17</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.47</td>
<td>0.57</td>
<td>0.44</td>
<td>0.21</td>
<td>0.44</td>
<td>0.67</td>
<td>0.41</td>
<td>0.1</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.71</td>
<td>1</td>
<td>0.68</td>
<td>0.24</td>
<td>0.52</td>
<td>0.71</td>
<td>0.45</td>
<td>0.29</td>
</tr>
<tr>
<td>solid</td>
<td>0.84</td>
<td>0.37</td>
<td>0.87</td>
<td>0.11</td>
<td>0.9</td>
<td>1</td>
<td>0.9</td>
<td>0.04</td>
</tr>
</tbody>
</table>

**Table 19: MLN Results Over Time**

<table>
<thead>
<tr>
<th>Project</th>
<th>First Half</th>
<th></th>
<th></th>
<th></th>
<th>Second Half</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Recall</td>
<td>Neg. Recall</td>
<td>Precision</td>
<td>Accuracy</td>
<td>Recall</td>
<td>Neg. Recall</td>
<td>Precision</td>
</tr>
<tr>
<td>ark</td>
<td>0.84</td>
<td>0.57</td>
<td>0.9</td>
<td>0.57</td>
<td>0.8</td>
<td>1</td>
<td>0.75</td>
<td>0.5</td>
</tr>
<tr>
<td>elisa</td>
<td>0.66</td>
<td>0.55</td>
<td>0.69</td>
<td>0.29</td>
<td>0.58</td>
<td>0.52</td>
<td>0.63</td>
<td>0.52</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.61</td>
<td>0.79</td>
<td>0.52</td>
<td>0.45</td>
<td>0.72</td>
<td>0.67</td>
<td>0.73</td>
<td>0.3</td>
</tr>
<tr>
<td>k3b</td>
<td>0.8</td>
<td>0.5</td>
<td>0.81</td>
<td>0.2</td>
<td>0.77</td>
<td>0.9</td>
<td>0.77</td>
<td>0.11</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.87</td>
<td>0.6</td>
<td>0.89</td>
<td>0.24</td>
<td>0.6</td>
<td>0.8</td>
<td>0.57</td>
<td>0.21</td>
</tr>
<tr>
<td>konsole</td>
<td>0.77</td>
<td>0.63</td>
<td>0.81</td>
<td>0.44</td>
<td>0.68</td>
<td>0.57</td>
<td>0.7</td>
<td>0.29</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.65</td>
<td>0.57</td>
<td>0.67</td>
<td>0.32</td>
<td>0.66</td>
<td>0.67</td>
<td>0.66</td>
<td>0.17</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.8</td>
<td>1</td>
<td>0.78</td>
<td>0.32</td>
<td>0.66</td>
<td>0.57</td>
<td>0.68</td>
<td>0.36</td>
</tr>
<tr>
<td>solid</td>
<td>0.63</td>
<td>0.91</td>
<td>0.62</td>
<td>0.1</td>
<td>0.98</td>
<td>1</td>
<td>0.98</td>
<td>0.9</td>
</tr>
</tbody>
</table>
6.5.1 Discussion

This strategy evaluates whether a ruleset performs consistently well across time in a project. The results from this strategy can be used to answer RQ3, which is “Does the trained set of rules apply equally well over time or does its performance deteriorate?”. Table-18 displays the results for Fuzzy Logic Strategy-1 over two periods of time to compare the results of the model in the two time periods. Almost all the projects give relatively the same results or slightly increase over time. The two exceptions to this being konsole and plasma-nm. The accuracy decreases by 9% and the recall decreases by 28% for konsole while plasma-nm saw a drop of 19% in accuracy and 29% in recall. MLN, whose results are displayed in Table-19, stays relatively stable for both time periods with only plasma having a decrease of 14% in accuracy and 43% in recall. This shows that, despite some exceptions, the model continues to give good results over time without a steep decrease in the validity of the rules selected.

6.6 Strategy-5: Reusing Trained Subset of Rules

*Approach:* This strategy evaluates whether a ruleset that performs well in one project can be used equally well in other projects. More specifically, after we obtain a ruleset $R_i$ that yields the best results for project $P_i$ (i.e. using Strategy 1), we apply this ruleset $R_i$ to all other projects $P_k$ where $k \neq i$. We then obtain the average performance scores by applying each such rule set $R_i$ to all other projects $P_k$ ($k \neq i$).

*Fuzzy Logic:*

For every project that had good results in Strategy-1, the rules were directly applied for every other project. We assume the project performed well if its recall is above 0.6. In the interest of space, we only present the average metrics that were obtained by using each project’s rules in Table-20. The detailed results are given in Appendix A1.
Markov Logic Networks:

The output MLN files of the projects that gave decent rules were directly given as input to the other projects’ inference system. The results shown in Table-21 display the average values of each metric when using that project’s rules. Deep-level results for each project are outlined in Appendix A2.

Table 20: Fuzzy Logic Results for using trained rules for other projects

<table>
<thead>
<tr>
<th>Project</th>
<th>Avg Accuracy</th>
<th>Avg Recall</th>
<th>Avg Neg. Recall</th>
<th>Avg Precision</th>
<th>Avg F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ark</td>
<td>0.54</td>
<td>0.77</td>
<td>0.52</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.71</td>
<td>0.59</td>
<td>0.73</td>
<td>0.21</td>
<td>0.31</td>
</tr>
<tr>
<td>juk</td>
<td>0.64</td>
<td>0.60</td>
<td>0.64</td>
<td>0.18</td>
<td>0.27</td>
</tr>
<tr>
<td>k3b</td>
<td>0.62</td>
<td>0.66</td>
<td>0.62</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.56</td>
<td>0.78</td>
<td>0.53</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>kmix</td>
<td>0.56</td>
<td>0.79</td>
<td>0.53</td>
<td>0.16</td>
<td>0.27</td>
</tr>
<tr>
<td>kompare</td>
<td>0.56</td>
<td>0.70</td>
<td>0.54</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>konsole</td>
<td>0.49</td>
<td>0.81</td>
<td>0.45</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>konversation</td>
<td>0.56</td>
<td>0.77</td>
<td>0.53</td>
<td>0.17</td>
<td>0.28</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.54</td>
<td>0.72</td>
<td>0.52</td>
<td>0.15</td>
<td>0.24</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.66</td>
<td>0.70</td>
<td>0.66</td>
<td>0.19</td>
<td>0.29</td>
</tr>
</tbody>
</table>
Table 21: MLN Results for using trained rules on other projects

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.69</td>
<td>0.71</td>
<td>0.70</td>
<td>0.24</td>
<td>0.36</td>
</tr>
<tr>
<td>ark</td>
<td>0.69</td>
<td>0.66</td>
<td>0.71</td>
<td>0.24</td>
<td>0.35</td>
</tr>
<tr>
<td>elisa</td>
<td>0.72</td>
<td>0.63</td>
<td>0.74</td>
<td>0.24</td>
<td>0.35</td>
</tr>
<tr>
<td>juk</td>
<td>0.70</td>
<td>0.70</td>
<td>0.71</td>
<td>0.23</td>
<td>0.35</td>
</tr>
<tr>
<td>k3b</td>
<td>0.66</td>
<td>0.73</td>
<td>0.65</td>
<td>0.20</td>
<td>0.32</td>
</tr>
<tr>
<td>kmix</td>
<td>0.70</td>
<td>0.71</td>
<td>0.71</td>
<td>0.23</td>
<td>0.34</td>
</tr>
<tr>
<td>kompare</td>
<td>0.69</td>
<td>0.69</td>
<td>0.70</td>
<td>0.22</td>
<td>0.33</td>
</tr>
<tr>
<td>konsole</td>
<td>0.72</td>
<td>0.70</td>
<td>0.73</td>
<td>0.23</td>
<td>0.35</td>
</tr>
<tr>
<td>konversation</td>
<td>0.69</td>
<td>0.64</td>
<td>0.70</td>
<td>0.22</td>
<td>0.33</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.74</td>
<td>0.69</td>
<td>0.74</td>
<td>0.27</td>
<td>0.39</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.68</td>
<td>0.75</td>
<td>0.68</td>
<td>0.23</td>
<td>0.35</td>
</tr>
<tr>
<td>solid</td>
<td>0.70</td>
<td>0.73</td>
<td>0.71</td>
<td>0.23</td>
<td>0.35</td>
</tr>
</tbody>
</table>

6.6.1 Discussion

This strategy evaluated whether a ruleset that performed well in one project can be used equally well in other projects. We used the rules that performed well for a project as input for all other projects and tabulated the results. We consider a ruleset to perform well if it gives a recall of 60% or above. The results from this strategy can be used to answer RQ2, which is “If a collection of rules works in one system, can it be used in other systems?”. Table-20 shows the average performance results of all projects using Fuzzy Logic. Almost every project’s rules gave acceptable results with the best being that of kmix with an average accuracy of 56% and an average recall of 79%. The rules of gwenview gave the highest accuracy but had the lowest average recall of all. We hypothesize that the reason behind this behavior is the inclusion of negative class rules in the rule subset of gwenview.

MLN’s results, which are shown in Table-21, show a similar picture. Elisa’s output MLN file gave the least recall, while plasma’s output MLN file gave the best recall. Plasma-nm’s rules
also gave a good balance between Accuracy and Recall giving better results than using Common Rules in Strategy-3. Overall, it is hard to know what makes one project’s rules perform better than the other but it is clear that using this approach gives acceptable results in most cases.

6.7 Strategy-6: Custom Rules

Approach: This strategy studies the feasibility of creating customized rules for each project without starting from an optimized ruleset. More specifically, for every project $P_i$, we create a custom ruleset $R_i$ using the technique outlined in Section 4.6 yielding rules $R_n, R_{n+1}, \ldots, R_{(n+m)}$ that provide the best recall in the training dataset $Train_i$ of $P_i$. We then use the ruleset $R_i$ on the testing set $Test_i$ of $P_i$.

Fuzzy Logic:

For each project, rules are written from scratch. First, inference was performed on the training set with zero rules applied. The result of this step is that all the fault-prone segments are considered as False Negatives and all clean segments will be in the True Negatives file. The custom utility program described in Section 4.6 is used to inspect the thresholds of the metrics and rules are iteratively written and applied. The results for this approach using Fuzzy Logic are presented in Table-22.
Table 22: Fuzzy Logic Results for Custom Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Negative Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.47</td>
<td>1</td>
<td>0.44</td>
<td>0.1</td>
<td>0.18</td>
</tr>
<tr>
<td>ark</td>
<td>0.53</td>
<td>1</td>
<td>0.41</td>
<td>0.31</td>
<td>0.48</td>
</tr>
<tr>
<td>elisa</td>
<td>0.58</td>
<td>0.67</td>
<td>0.55</td>
<td>0.36</td>
<td>0.47</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.55</td>
<td>0.8</td>
<td>0.45</td>
<td>0.34</td>
<td>0.48</td>
</tr>
<tr>
<td>juk</td>
<td>0.74</td>
<td>1</td>
<td>0.72</td>
<td>0.19</td>
<td>0.32</td>
</tr>
<tr>
<td>k3b</td>
<td>0.87</td>
<td>0.64</td>
<td>0.88</td>
<td>0.07</td>
<td>0.12</td>
</tr>
<tr>
<td>kate</td>
<td>0.45</td>
<td>0.55</td>
<td>0.44</td>
<td>0.05</td>
<td>0.1</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.82</td>
<td>0.17</td>
<td>0.89</td>
<td>0.15</td>
<td>0.16</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.82</td>
<td>0.46</td>
<td>0.85</td>
<td>0.23</td>
<td>0.3</td>
</tr>
<tr>
<td>kmix</td>
<td>0.65</td>
<td>1</td>
<td>0.63</td>
<td>0.13</td>
<td>0.22</td>
</tr>
<tr>
<td>konpare</td>
<td>0.63</td>
<td>1</td>
<td>0.62</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>konsole</td>
<td>0.61</td>
<td>0.63</td>
<td>0.61</td>
<td>0.27</td>
<td>0.38</td>
</tr>
<tr>
<td>konversation</td>
<td>0.54</td>
<td>0.83</td>
<td>0.38</td>
<td>0.15</td>
<td>0.26</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.81</td>
<td>1</td>
<td>0.8</td>
<td>0.22</td>
<td>0.36</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.44</td>
<td>0.63</td>
<td>0.39</td>
<td>0.18</td>
<td>0.27</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.43</td>
<td>0.85</td>
<td>0.36</td>
<td>0.17</td>
<td>0.29</td>
</tr>
<tr>
<td>solid</td>
<td>0.89</td>
<td>0.43</td>
<td>0.9</td>
<td>0.08</td>
<td>0.14</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.60</td>
<td>0.4</td>
<td>0.62</td>
<td>0.1</td>
<td>0.15</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.66</strong></td>
<td><strong>0.73</strong></td>
<td><strong>0.61</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Markov Logic Networks:

The same rules as the ones used in Fuzzy Logic are applied in the MLN model. After training the weights, the rules with negative weights were removed and inference was applied on the testing dataset using the output MLN file. The results for this approach are tabulated in Table-23.
Table 23: MLN Results for Custom Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Neg. Recall</th>
<th>Precision</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>ak regator</td>
<td>0.89</td>
<td>1</td>
<td>0.92</td>
<td>0.37</td>
<td>0.54</td>
</tr>
<tr>
<td>ark</td>
<td>0.49</td>
<td>0.7</td>
<td>0.43</td>
<td>0.25</td>
<td>0.37</td>
</tr>
<tr>
<td>elisa</td>
<td>0.69</td>
<td>0.52</td>
<td>0.76</td>
<td>0.45</td>
<td>0.48</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.67</td>
<td>0.62</td>
<td>0.68</td>
<td>0.4</td>
<td>0.49</td>
</tr>
<tr>
<td>juk</td>
<td>0.76</td>
<td>0.67</td>
<td>0.77</td>
<td>0.15</td>
<td>0.25</td>
</tr>
<tr>
<td>k3b</td>
<td>0.81</td>
<td>0.72</td>
<td>0.8</td>
<td>0.05</td>
<td>0.09</td>
</tr>
<tr>
<td>kate</td>
<td>0.41</td>
<td>0.64</td>
<td>0.4</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.72</td>
<td>0.58</td>
<td>0.74</td>
<td>0.2</td>
<td>0.30</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.75</td>
<td>0.48</td>
<td>0.78</td>
<td>0.17</td>
<td>0.25</td>
</tr>
<tr>
<td>kmix</td>
<td>0.59</td>
<td>1</td>
<td>0.57</td>
<td>0.11</td>
<td>0.20</td>
</tr>
<tr>
<td>kompare</td>
<td>0.63</td>
<td>1</td>
<td>0.62</td>
<td>0.06</td>
<td>0.11</td>
</tr>
<tr>
<td>konsole</td>
<td>0.74</td>
<td>0.63</td>
<td>0.77</td>
<td>0.39</td>
<td>0.48</td>
</tr>
<tr>
<td>konversation</td>
<td>0.64</td>
<td>0.67</td>
<td>0.63</td>
<td>0.24</td>
<td>0.35</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.94</td>
<td>0.8</td>
<td>0.95</td>
<td>0.45</td>
<td>0.58</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.62</td>
<td>0.6</td>
<td>0.63</td>
<td>0.25</td>
<td>0.35</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.75</td>
<td>0.31</td>
<td>0.82</td>
<td>0.21</td>
<td>0.25</td>
</tr>
<tr>
<td>solid</td>
<td>0.82</td>
<td>0.57</td>
<td>0.83</td>
<td>0.07</td>
<td>0.12</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.58</td>
<td>1</td>
<td>0.54</td>
<td>0.18</td>
<td>0.31</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>0.7</strong></td>
<td><strong>0.69</strong></td>
<td><strong>0.71</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6.7.1 Discussion

In this strategy, we devised custom rules for each project without relying on the training module included with the models. The technique described in Section 4.6 was used to create the rules of a specific project. Table-22 displays the results of using custom rules with Fuzzy Logic. On average, it performed better than Strategy-1, Strategy-2, and Strategy-3 with an average accuracy of 66%, recall of 73%, and negative recall of 61%. The steepest increase in recall was observed in the projects *juk, kate, kio-extras, solid, and systemsettings*. 
Chapter 6: Experiments and Results

The results of MLN, displayed in Table-23, were comparable to those of Fuzzy Logic with an increase of 4% in accuracy, 10% in negative recall, and a 4% decrease in recall. Plasma-nm saw the biggest drop in recall while akregator, lokalize, and systemsettings all saw their recall improve using this strategy. The average recall observed was lower than that of Strategy-1 and Strategy-2, which suggests that MLNs perform better when the inbuilt weight training module is used to assign weights to rules.

6.8 Comparison with Machine Learning Techniques

In this section, we compare the results that can be obtained using a pure black-box machine learning approach, with the rule-based approach. The comparison is based on a recent empirical study conducted by Grigoriou et. al. in [111] by using different Machine Learning Techniques on process metrics of the same projects as we used for the experiments in this thesis. This is not a pure one-to-one comparison, as the Machine Learning approach and the rule-based approaches are based on different technologies and the way the training is performed. However, we feel that such a comparison provides a point of reference of how the best rule-based approaches compare with the best of Machine Learning ones, on the same projects. It is noted that there are some key differences in the approach used in [111] and the one used in this thesis. Firstly, we have used the latest five years of historical data for each project while the authors have classified fault-prone files in the entire lifecycle of the projects. Secondly, we have used segments to measure the trends of metrics in intervals of time with the use of segments, while Grigoriou et. al. predicted fault-prone files on the commits directly. Thirdly and possibly most importantly, files were annotated as faulty by parsing the GitHub message and looking for terms that indicate that this commit is a bug fixing commit as opposed to a clean one. This approach, as discussed by the authors, has the potential to add a high number of false positives which possibly skewed the results. Table-24 shows the results of the authors using Logistic Regression on the eighteen open-source projects using the following features: Commit Frequency, Total Churn, Total Co-Commit Size, and Total Distinct Authors.

While it cannot be a one-to-one comparison because of the differences listed above, it is still useful to see the potential of the rule-based techniques in this thesis compared to other classic Machine Learning (ML) techniques (Logistic Regression in this case). The first obvious difference can be found in the Precision and F1 score. The ML technique got an average of 0.73 in both the metrics,
which is in stark contrast to the performance metrics of our approaches. One of the main reasons behind this is the granularity of the rules. We divide every metric into three categories Low, Avg, and High, while ML has the advantage of using very precise training that can differentiate the behavior of two elements with very similar metric values. The promising observation is that recall achieved by the models in this thesis is very close to the recall attained by ML. MLN got an average recall of 0.74 in Strategy-2 (Table-15) and Fuzzy Logic got an average recall of 0.73 in Strategy-6 (Table-22). The accuracy and precision can be improved in three ways: introducing more granular rules, use Rule Learning, and use a filtering and ranking mechanism in the CI/CD pipeline to remove obvious false positives. Overall, in our belief, the benefit that is obtained in terms of transparency, explainability, and customizability outweigh the lower precision scores which can be potentially mitigated using expert knowledge, ranking, or filtering. For example, once a high recall/low precision list of fault-prone files is provided by the rule-based system, then the developers can use three different techniques to identify the true positives and increase precision as follows: a) use their expert knowledge of the system to sieve the list and focus on the suspicious files based on their expertise on the system; b) use a ranking mechanism to sort the results in order to move to the top of the list the true positives. Such a ranking mechanism can be based on the past interactions between known buggy or error-prone files and the file under investigation, intense call relations that can be extracted by logs, and high volume data dependencies that can be extracted by observing data transfers in call parameters or use of common resources or databases, and; c) use a filtering mechanism that can discard from the initial list of results, files that are known to be healthy or have a very low probability to be buggy (e.g. they have minimal interaction with other files and their commits involve a very low churn).

Keeping in mind that our primary goal in customizing the rules was to maximize recall, so that we make sure we obtain the highest possible number of true positives, the results of comparing the best rule-based approach for each system\(^2\) with the corresponding ML approach (see Table 24), indicate that the rule-based approaches provide a comparable level of recall as the ML approach. The added benefit of using the rule-based approach is that developers being able to trace the logic as to why and how a prediction was reached. For the systems where a higher recall is reported using the ML approach (e.g. elisa, k3b) we hypothesize that this is due to the fact that the ML

\(^2\) Please see Table 25 for the best approach of each project.
approach does not use a GitHub-Bugzilla reconciled data set and therefore the ratio of the buggy to non-buggy files is higher which leads not only to higher recall but also to higher precision, compared to the rule-based approach.

Table 24: Comparison of Rule-Based and Machine Learning

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>0.89 (0.86)</td>
<td>1 (0.82)</td>
<td>0.37 (0.81)</td>
<td>0.54 (0.81)</td>
</tr>
<tr>
<td>ark</td>
<td>0.53 (0.86)</td>
<td>1 (0.77)</td>
<td>0.31 (0.88)</td>
<td>0.48 (0.82)</td>
</tr>
<tr>
<td>elisa</td>
<td>0.57 (0.87)</td>
<td>0.71 (0.82)</td>
<td>0.35 (0.92)</td>
<td>0.47 (0.86)</td>
</tr>
<tr>
<td>gwenview</td>
<td>0.59 (0.86)</td>
<td>0.83 (0.68)</td>
<td>0.37 (0.55)</td>
<td>0.52 (0.59)</td>
</tr>
<tr>
<td>juk</td>
<td>0.8 (0.92)</td>
<td>1 (0.92)</td>
<td>0.23 (0.98)</td>
<td>0.37 (0.95)</td>
</tr>
<tr>
<td>k3b</td>
<td>0.91 (0.98)</td>
<td>0.71 (0.98)</td>
<td>0.1 (1)</td>
<td>0.18 (0.99)</td>
</tr>
<tr>
<td>kate</td>
<td>0.64 (0.89)</td>
<td>0.64 (0.81)</td>
<td>0.1 (0.64)</td>
<td>0.17 (0.71)</td>
</tr>
<tr>
<td>kdelibs</td>
<td>0.75 (0.9)</td>
<td>0.58 (0.75)</td>
<td>0.23 (0.47)</td>
<td>0.33 (0.58)</td>
</tr>
<tr>
<td>kio-extras</td>
<td>0.63 (0.82)</td>
<td>0.82 (0.69)</td>
<td>0.17 (0.64)</td>
<td>0.28 (0.66)</td>
</tr>
<tr>
<td>kmix</td>
<td>0.72 (0.86)</td>
<td>1 (0.88)</td>
<td>0.15 (0.74)</td>
<td>0.26 (0.8)</td>
</tr>
<tr>
<td>kompare</td>
<td>0.4 (0.74)</td>
<td>1 (0.5)</td>
<td>0.04 (0.65)</td>
<td>0.08 (0.52)</td>
</tr>
<tr>
<td>konsole</td>
<td>0.52 (0.83)</td>
<td>0.8 (0.69)</td>
<td>0.25 (0.73)</td>
<td>0.38 (0.7)</td>
</tr>
<tr>
<td>ktorrent</td>
<td>0.85 (0.89)</td>
<td>1 (0.91)</td>
<td>0.26 (0.93)</td>
<td>0.41 (0.92)</td>
</tr>
<tr>
<td>konversation</td>
<td>0.8 (0.81)</td>
<td>0.67 (0.73)</td>
<td>0.4 (0.74)</td>
<td>0.5 (0.73)</td>
</tr>
<tr>
<td>lokalize</td>
<td>0.62 (0.77)</td>
<td>0.6 (0.75)</td>
<td>0.25 (0.79)</td>
<td>0.35 (0.77)</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>0.72 (0.89)</td>
<td>0.77 (0.86)</td>
<td>0.3 (0.77)</td>
<td>0.43 (0.81)</td>
</tr>
<tr>
<td>solid</td>
<td>0.86 (0.93)</td>
<td>0.64 (0.42)</td>
<td>0.1 (0.26)</td>
<td>0.17 (0.31)</td>
</tr>
<tr>
<td>systemsettings</td>
<td>0.58 (0.86)</td>
<td>1 (0.73)</td>
<td>0.18 (0.71)</td>
<td>0.31 (0.71)</td>
</tr>
<tr>
<td>Average</td>
<td><strong>0.69 (0.86)</strong></td>
<td><strong>0.82 (0.76)</strong></td>
<td><strong>0.23 (0.73)</strong></td>
<td><strong>0.35 (0.73)</strong></td>
</tr>
</tbody>
</table>

3 Numbers in parentheses indicate the corresponding ML results.
The lower scores in F1 are explained by the lower Precision scores which consequently drive F1 scores to be lower. Similarly, the lower Accuracy scores are explained by the nature of the rule-based approach which uses a gross classification of the data namely, Low, Avg, and High (e.g. \texttt{avgFrequencyOfCommits(s,f)}, \texttt{highFrequencyOfCommits(s,f)}, \texttt{“frequencyOfCommits IS high”}, \texttt{“frequencyOfCommits IS avg”}), compared to the real or integer metric values used in the Machine Learning approach. The problem can be possibly mitigated by adding more detailed predicates and rules.
Chapter 7: Conclusion and Future Work

7.1 Conclusion

In this thesis, we present a fault-proneness prediction technique that incorporates domain expert knowledge in the form of inference rules, and we report results by applying two different inferencing models using six different strategies, on nineteen open-source systems.

The first inferencing model is based on Fuzzy logic which employs different membership functions for each metric. Rules and metrics were written using the Fuzzy Control Language Standard and inference was performed by keeping Center of Gravity as the defuzzification method. The second inferencing model is based on Markov Logic Networks (MLNs). The model uses predicates and rules that are similar to First-Order Logic, but each rule has an associated weight based on its importance. Lifted Belief Propagation was used to perform inference on the testing dataset after the weights were trained using Discriminative Learning.

We report results by using six different strategies on both the Fuzzy logic and the MLN models, and we compare and contrast their performance. Our results indicate that on average, MLN-based inferencing performs better than Fuzzy Logic inferencing. We believe that there are four reasons explaining this behavior. The first reason is that in MLNs, every rule contributes towards the final probability whereas Fuzzy Logic only considers the activated rules to calculate the final score (Section 2.4.6). The second reason is that MLNs have an associated weight with every rule. This sets important rules apart from less important rules and makes sure that the final probability is calculated based on all factors. The third reason is that MLNs convert every rule into its CNF form and give separate weights to each subset of the rule, unlike Fuzzy Logic where the entire rule is either included or excluded. The final and probably the most important reason is that MLNs have a very robust and efficient training algorithm to train the weights of the rule, while Fuzzy Logic uses a Brute-Force method of training which is not very scalable.

Table-25 provides an overview of which strategies gave the best results for each individual project. It is not exactly clear cut how to differentiate results from different strategies, but we put the most importance on recall and accuracy. We consider a strategy that gives balanced values
Table 25: Strategy that gives the best results for each project

<table>
<thead>
<tr>
<th>Project</th>
<th>The strategy that gave the best results</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>MLN-custom</td>
</tr>
<tr>
<td>ark</td>
<td>Fuzzy-custom</td>
</tr>
<tr>
<td>elisa</td>
<td>MLN-expanded</td>
</tr>
<tr>
<td>gwenview</td>
<td>Fuzzy-expanded</td>
</tr>
<tr>
<td>juk</td>
<td>MLN-common</td>
</tr>
<tr>
<td>k3b</td>
<td>MLN-expanded</td>
</tr>
<tr>
<td>kate</td>
<td>MLN-common</td>
</tr>
<tr>
<td>kdelibs</td>
<td>Fuzzy-common/MLN-common</td>
</tr>
<tr>
<td>kio-extras</td>
<td>Fuzzy-common/MLN-common</td>
</tr>
<tr>
<td>kmix</td>
<td>MLN-best subset</td>
</tr>
<tr>
<td>kompare</td>
<td>MLN-common</td>
</tr>
<tr>
<td>konsole</td>
<td>Fuzzy-common/MLN-expanded</td>
</tr>
<tr>
<td>konversation</td>
<td>Fuzzy-custom/MLN-best subset</td>
</tr>
<tr>
<td>ktorrent</td>
<td>MLN-best subset/ MLN-common</td>
</tr>
<tr>
<td>lokalize</td>
<td>MLN-custom</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>MLN-best subset</td>
</tr>
<tr>
<td>solid</td>
<td>MLN-best subset</td>
</tr>
<tr>
<td>systemsettings</td>
<td>MLN-common/MLN-custom</td>
</tr>
</tbody>
</table>

accuracy and recall better than one where they are lopsided. One thing that is observed from the table is that there is no one size fits all when it comes to the approach to create a rule-based model for fault prediction. Strategy-3 of having common rules had the best results most often but it has the caveat of giving very substandard results for others. Overall, the choice of which strategy to use would have to be decided on a case-to-case basis.

The approaches presented in this thesis yield high recall scores which are comparable to the best Machine Learning techniques presented in the related literature. However, the obtained results suffer from low precision, an issue that can be possibly mitigated by using expert knowledge, ranking, and filtering as discussed in Section 6.8. We believe that one of the reasons the rule-based techniques yield low precision scores is because, in their current form, the Fuzzy linguistic
variables and the MLN predicates are not very granular as they utilize only three distinct categories namely *low*, *avg* and *high*, and therefore the rules and the overall data representation become less granular in nature than of the Machine Learning techniques that use real value or integer metrics. This leads to the relatively lower precision as reported in the results presented in Chapter 6. Nevertheless, the average recall stayed relatively high for all strategies with the best results being observed in Strategy-3 of Fuzzy Logic and the worst results in Strategy-1 of MLNs. The best balance of recall and accuracy was observed in Strategy-6 of Fuzzy Logic and Strategy-2 of MLNs. Furthermore, the proposed system allows for project-custom rule sets to be selected so that accuracy and recall can be maximized.

### 7.2 Thesis Findings

- We have designed and implemented a fault prediction technique that is solely based on process metrics that can be obtained from source code repositories and hence not depend on the underlying implementation language of the system being analyzed. The system is geared towards maximizing recall values so it can be used as an effective fault prediction method in large systems. We envision a post-processing filtering/ranking step in a CI/CD pipeline where these high-recall lists of potentially fault-prone files can be used to flag the fault-prone files of the system.

- We have investigated and compiled a collection of domain-specific rules which can be used by software developers and testers to identify fault-prone files. More specifically, we have identified twenty generic rules which take into account the past and present behavior of a file and its interactions with other files to deduce whether the file will exhibit a faulty behavior in the near future or not. Furthermore, rule-based approaches can be customizable in the sense that new rules which fit a particular project can be added by developers and testers. This is not possible in classic machine learning approaches where when a model is trained, it remains immutable. Finally, and in addition to the above, the thesis proposes an optimization algorithm which can present the best subset of rules for a project from a rule repository.
• We have conducted a series of experiments in this thesis to compare the Fuzzy rule-based approach with the MLN probabilistic approach. Our results indicate that the MLNs give consistently better accuracy than the Fuzzy Logic model. The recall however is similar for both models with Fuzzy Logic giving slightly better results. MLNs also have the advantage of having a robust weight learning framework and can divide each rule into subsets based on its CNF form to better optimize a ruleset for a project.

• In this thesis, we have evaluated the performance of rule-based systems for fault proneness prediction compared to classic black-box machine learning approaches. Our results indicate that the recall scores using rule-based systems are similar to the ones obtained by machine learning approaches while the precision is much lower using rule-based systems than machine learning ones. However, the added benefit of rule-based approaches is that they allow for explanations to be given as to how the system reached its conclusion. The drawback of the low precision values can be mitigated by using a post-processing filtering/ranking mechanism.

• In this thesis, we aimed to answer four major research questions. Our findings are:

  RQ1: “Is there a common set of rules that can be applied to all projects for predicting fault-prone segments?”. Our experiments on making a common set of rules using rule frequency as a metric is viable, especially in MLNs. Fuzzy Logic showed very high recall but had low accuracy. The results vary depending on which rule-set selection technique is used to compile the common rule-set.

  RQ2: “Does the trained set of rules apply equally well over time or does its performance deteriorate?”. We applied the rules on two halves of the testing data divided over time. We observed that rules on most projects have either the same level of performance or perform better over time. There were a couple of exceptions for both models, but for the most part, the rules perform equally well over the lifetime of the project.

  RQ3: “If a collection of rules works in one system, can it be used in other systems?”. We chose rules from projects that work well and applied them to all other projects. The results are extremely promising for both Fuzzy and MLNs and quite consistent.
Chapter 7: Conclusion and Future Work

RQ4: “What is the average increase or decrease of using expanded rules on top of the chosen rule subset?” We measured the difference in the performance of expanded rules over the optimized rule subset and found that adding extra rules on top of the optimized ruleset is good for recall but comes with a small decrease in accuracy. The primary reason behind this behavior is because we have mostly added positive class identifying rules on top of the subset.

Concluding, our results indicate that there is a lot of potential in using expert rules for a fault prediction system. While the accuracy and precision were lower when compared to models using Machine Learning, we were able to maintain a relatively high level of recall. The advantage of full transparency and control could be an attractive alternative to the traditional models, and we believe there is a lot of future research that can be done in this area. We also include the future work that can be done on this domain using this thesis as the springboard.

7.3 Threats to validity

1. The projects that were chosen were all predominantly written in C++. It is necessary to compare the performance of the models with projects written in other languages. It is difficult to find projects which have both GitHub and Bugzilla repositories, but it would be interesting to use the model on high-volume industrial projects that may use other repositories such as Jira, Jenkins, and Slack.

2. There is a chance that the reconciliation process introduced some false positives or false negatives. The current process relies on Bugzilla records and simple heuristics to tag a file in a commit as buggy or not buggy, which is not always accurate.

3. The metrics and segments were calculated for the last five years’ worth of commits of each project to get the latest results, but there is a chance that older projects have stabilized by this point, thus resulting in fewer faulty files and segments, which may skew our data.

4. The current framework randomly selects a starting commit for each file and calculates segment width for 36 consecutive segments for time optimization. So, there is a chance that some files that had faulty commits did not get included in a segment.

5. The default algorithms were chosen for inference in both MLN and Fuzzy Logic, but there is a possibility that using a different algorithm could potentially yield better results.

6. The rules are heuristic by their nature so, the results could vary depending on the rules.
Chapter 7: Conclusion and Future Work

7. The thresholds of the metrics were intentionally kept less granular to ease the writing of the custom rules. But, using more specific levels of thresholds could result in better accuracy and recall.

7.4 Future Work

1. The training algorithm to select rules for Fuzzy Logic can be updated to be more robust and scalable. While no such algorithm exists in the current literature, there is a lot of work in Rule Learning where the rules are automatically derived from a dataset of training examples [112] [113] [114]. Rule learning can be combined with an increase in granularity of threshold levels to give more fine-tuned results.

2. The current implementation only uses Repository Metrics to have a language-agnostic nature. However, some source code metrics related to comments and lines of code can be added as metrics while still keeping the model language agnostic. There is also some ongoing work where the source code is treated as a bag-of-words model and useful information extracted [79] [83]. Also easy to extract source code dependencies between files can be extracted by using simple scanners or log analyzers. In this respect, files that are dependent on each other (e.g. via calls or data references) can be clustered in fault-prone groups. The addition of these metrics and dependencies can be beneficial to the prediction model.

3. The current model was only tested on C++ systems due to a lack of availability of large projects which use Bugzilla and GitHub. It will be interesting to experiment with systems in other languages to compare the performance.

4. Another possible direction of future work is to expand the input dataset by including data extracted from Jira, Slack, and Jenkins. In this respect, we can create more rules by considering new predicates (i.e. new features), compiling thus a more detailed rule-set and knowledge base.
Chapter 8: References


Appendix A: Deep Level Results of Reusing Trained Subset of Rules

Note- Only for projects where the best rule subset yielded a recall of 60% or above

A1: Fuzzy Based Fault Proneness Identification (Metrics in Percentage)

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>73</td>
<td>100</td>
<td>18</td>
<td>72</td>
</tr>
<tr>
<td>elisa</td>
<td>51</td>
<td>67</td>
<td>32</td>
<td>46</td>
</tr>
<tr>
<td>gwenview</td>
<td>58</td>
<td>74</td>
<td>35</td>
<td>52</td>
</tr>
<tr>
<td>juk</td>
<td>58</td>
<td>100</td>
<td>12.5</td>
<td>55</td>
</tr>
<tr>
<td>k3b</td>
<td>63</td>
<td>79</td>
<td>3</td>
<td>63</td>
</tr>
<tr>
<td>kate</td>
<td>43</td>
<td>55</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>kdelibs</td>
<td>75</td>
<td>58</td>
<td>23</td>
<td>77</td>
</tr>
<tr>
<td>kio-extras</td>
<td>70</td>
<td>66</td>
<td>18</td>
<td>71</td>
</tr>
<tr>
<td>kmix</td>
<td>34</td>
<td>100</td>
<td>7</td>
<td>30</td>
</tr>
<tr>
<td>kompare</td>
<td>46</td>
<td>100</td>
<td>5</td>
<td>45</td>
</tr>
<tr>
<td>konsole</td>
<td>63</td>
<td>69</td>
<td>30</td>
<td>62</td>
</tr>
<tr>
<td>konversation</td>
<td>46</td>
<td>83</td>
<td>19</td>
<td>40</td>
</tr>
<tr>
<td>ktorrent</td>
<td>55</td>
<td>100</td>
<td>11</td>
<td>53</td>
</tr>
<tr>
<td>lokalize</td>
<td>50</td>
<td>60</td>
<td>19</td>
<td>47</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>33</td>
<td>85</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>solid</td>
<td>79</td>
<td>65</td>
<td>7</td>
<td>80</td>
</tr>
<tr>
<td>systemsettings</td>
<td>51</td>
<td>60</td>
<td>11</td>
<td>50</td>
</tr>
</tbody>
</table>
## Appendix A: Deep Level Results of Reusing Trained Subset of Rules

### Optimized Rule Subset of gwenview

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>44</td>
<td>50</td>
<td>6</td>
<td>44</td>
</tr>
<tr>
<td>ark</td>
<td>62</td>
<td>70</td>
<td>32</td>
<td>60</td>
</tr>
<tr>
<td>elisa</td>
<td>62</td>
<td>52</td>
<td>37</td>
<td>66</td>
</tr>
<tr>
<td>juk</td>
<td>84</td>
<td>100</td>
<td>28</td>
<td>83</td>
</tr>
<tr>
<td>k3b</td>
<td>80</td>
<td>58</td>
<td>4</td>
<td>80</td>
</tr>
<tr>
<td>kate</td>
<td>78</td>
<td>37</td>
<td>10</td>
<td>80</td>
</tr>
<tr>
<td>kdelibs</td>
<td>81</td>
<td>25</td>
<td>19</td>
<td>87</td>
</tr>
<tr>
<td>kio-extras</td>
<td>83</td>
<td>52</td>
<td>27</td>
<td>86</td>
</tr>
<tr>
<td>kmix</td>
<td>61</td>
<td>75</td>
<td>9</td>
<td>60</td>
</tr>
<tr>
<td>kompare</td>
<td>54</td>
<td>100</td>
<td>5</td>
<td>52</td>
</tr>
<tr>
<td>konsole</td>
<td>69</td>
<td>49</td>
<td>30</td>
<td>74</td>
</tr>
<tr>
<td>konversation</td>
<td>68</td>
<td>50</td>
<td>23</td>
<td>72</td>
</tr>
<tr>
<td>ktorrent</td>
<td>92</td>
<td>60</td>
<td>33</td>
<td>94</td>
</tr>
<tr>
<td>lokalize</td>
<td>65</td>
<td>30</td>
<td>16</td>
<td>73</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>43</td>
<td>69</td>
<td>15</td>
<td>39</td>
</tr>
<tr>
<td>solid</td>
<td>89</td>
<td>43</td>
<td>9</td>
<td>90</td>
</tr>
<tr>
<td>systemsettings</td>
<td>80</td>
<td>60</td>
<td>25</td>
<td>82</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of juk

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>77</td>
<td>100</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>ark</td>
<td>66</td>
<td>70</td>
<td>35</td>
<td>65</td>
</tr>
<tr>
<td>elisa</td>
<td>54</td>
<td>52</td>
<td>31</td>
<td>54</td>
</tr>
<tr>
<td>gwenview</td>
<td>65</td>
<td>58</td>
<td>38</td>
<td>67</td>
</tr>
<tr>
<td>k3b</td>
<td>76</td>
<td>65</td>
<td>4</td>
<td>77</td>
</tr>
<tr>
<td>kate</td>
<td>50</td>
<td>64</td>
<td>7</td>
<td>49</td>
</tr>
<tr>
<td>kdelibs</td>
<td>78</td>
<td>50</td>
<td>24</td>
<td>82</td>
</tr>
<tr>
<td>kio-extras</td>
<td>71</td>
<td>71</td>
<td>19</td>
<td>71</td>
</tr>
<tr>
<td>kmix</td>
<td>61</td>
<td>75</td>
<td>10</td>
<td>60</td>
</tr>
<tr>
<td>kompare</td>
<td>63</td>
<td>0</td>
<td>0</td>
<td>65</td>
</tr>
<tr>
<td>konsole</td>
<td>60</td>
<td>80</td>
<td>29</td>
<td>55</td>
</tr>
<tr>
<td>konversation</td>
<td>59</td>
<td>67</td>
<td>21</td>
<td>57</td>
</tr>
<tr>
<td>ktorrent</td>
<td>71</td>
<td>60</td>
<td>11</td>
<td>72</td>
</tr>
<tr>
<td>lokalize</td>
<td>55</td>
<td>60</td>
<td>21</td>
<td>53</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>57</td>
<td>62</td>
<td>19</td>
<td>57</td>
</tr>
<tr>
<td>solid</td>
<td>74</td>
<td>57</td>
<td>5</td>
<td>75</td>
</tr>
<tr>
<td>systemsettings</td>
<td>68</td>
<td>60</td>
<td>16</td>
<td>70</td>
</tr>
</tbody>
</table>
## Appendix A: Deep Level Results of Reusing Trained Subset of Rules

### Optimized Rule Subset of k3b

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>74</td>
<td>100</td>
<td>18</td>
<td>72</td>
</tr>
<tr>
<td>ark</td>
<td>53</td>
<td>80</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>elisa</td>
<td>59</td>
<td>59</td>
<td>35</td>
<td>59</td>
</tr>
<tr>
<td>gwenview</td>
<td>59</td>
<td>65</td>
<td>34</td>
<td>56</td>
</tr>
<tr>
<td>juk</td>
<td>68</td>
<td>67</td>
<td>12</td>
<td>68</td>
</tr>
<tr>
<td>kate</td>
<td>46</td>
<td>55</td>
<td>6</td>
<td>46</td>
</tr>
<tr>
<td>kdelibs</td>
<td>81</td>
<td>25</td>
<td>19</td>
<td>87</td>
</tr>
<tr>
<td>kio-extras</td>
<td>75</td>
<td>57</td>
<td>20</td>
<td>77</td>
</tr>
<tr>
<td>kmix</td>
<td>47</td>
<td>100</td>
<td>9</td>
<td>44</td>
</tr>
<tr>
<td>kompare</td>
<td>52</td>
<td>100</td>
<td>5</td>
<td>50</td>
</tr>
<tr>
<td>konsole</td>
<td>66</td>
<td>57</td>
<td>29</td>
<td>68</td>
</tr>
<tr>
<td>konversation</td>
<td>51</td>
<td>67</td>
<td>18</td>
<td>49</td>
</tr>
<tr>
<td>ktorrent</td>
<td>81</td>
<td>80</td>
<td>19</td>
<td>82</td>
</tr>
<tr>
<td>localize</td>
<td>56</td>
<td>44</td>
<td>18</td>
<td>60</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>40</td>
<td>77</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>solid</td>
<td>81</td>
<td>57</td>
<td>7</td>
<td>81</td>
</tr>
<tr>
<td>systemsettings</td>
<td>55</td>
<td>40</td>
<td>8</td>
<td>56</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of kio-extras

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>71</td>
<td>100</td>
<td>17</td>
<td>69</td>
</tr>
<tr>
<td>ark</td>
<td>51</td>
<td>80</td>
<td>28</td>
<td>43</td>
</tr>
<tr>
<td>elisa</td>
<td>53</td>
<td>67</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>gwenview</td>
<td>56</td>
<td>74</td>
<td>34</td>
<td>50</td>
</tr>
<tr>
<td>juk</td>
<td>58</td>
<td>100</td>
<td>13</td>
<td>55</td>
</tr>
<tr>
<td>k3b</td>
<td>64</td>
<td>79</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>kate</td>
<td>42</td>
<td>55</td>
<td>5</td>
<td>41</td>
</tr>
<tr>
<td>kdelibs</td>
<td>75</td>
<td>58</td>
<td>23</td>
<td>78</td>
</tr>
<tr>
<td>kmix</td>
<td>36</td>
<td>100</td>
<td>8</td>
<td>32</td>
</tr>
<tr>
<td>kompare</td>
<td>49</td>
<td>100</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>konsole</td>
<td>64</td>
<td>69</td>
<td>30</td>
<td>63</td>
</tr>
<tr>
<td>konversation</td>
<td>46</td>
<td>84</td>
<td>19</td>
<td>40</td>
</tr>
<tr>
<td>ktorrent</td>
<td>55</td>
<td>100</td>
<td>11</td>
<td>53</td>
</tr>
<tr>
<td>localize</td>
<td>50</td>
<td>60</td>
<td>19</td>
<td>48</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>33</td>
<td>85</td>
<td>15</td>
<td>24</td>
</tr>
<tr>
<td>solid</td>
<td>80</td>
<td>64</td>
<td>7</td>
<td>80</td>
</tr>
<tr>
<td>systemsettings</td>
<td>51</td>
<td>60</td>
<td>11</td>
<td>50</td>
</tr>
</tbody>
</table>
### Appendix A: Deep Level Results of Reusing Trained Subset of Rules

#### Optimized Rule Subset of kmix

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>59</td>
<td>100</td>
<td>13</td>
<td>56</td>
</tr>
<tr>
<td>ark</td>
<td>51</td>
<td>70</td>
<td>26</td>
<td>46</td>
</tr>
<tr>
<td>elisa</td>
<td>44</td>
<td>65</td>
<td>28</td>
<td>36</td>
</tr>
<tr>
<td>gwenview</td>
<td>60</td>
<td>72</td>
<td>37</td>
<td>57</td>
</tr>
<tr>
<td>juk</td>
<td>48</td>
<td>100</td>
<td>11</td>
<td>45</td>
</tr>
<tr>
<td>k3b</td>
<td>64</td>
<td>72</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>kate</td>
<td>41</td>
<td>73</td>
<td>7</td>
<td>39</td>
</tr>
<tr>
<td>kdelibs</td>
<td>77</td>
<td>59</td>
<td>24</td>
<td>79</td>
</tr>
<tr>
<td>kio-extras</td>
<td>66</td>
<td>75</td>
<td>18</td>
<td>65</td>
</tr>
<tr>
<td>kmpare</td>
<td>54</td>
<td>100</td>
<td>5</td>
<td>52</td>
</tr>
<tr>
<td>konsole</td>
<td>55</td>
<td>80</td>
<td>27</td>
<td>49</td>
</tr>
<tr>
<td>konversation</td>
<td>44</td>
<td>84</td>
<td>19</td>
<td>37</td>
</tr>
<tr>
<td>ktorrent</td>
<td>64</td>
<td>100</td>
<td>13</td>
<td>62</td>
</tr>
<tr>
<td>lokalize</td>
<td>53</td>
<td>60</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>53</td>
<td>85</td>
<td>20</td>
<td>48</td>
</tr>
<tr>
<td>solid</td>
<td>72</td>
<td>65</td>
<td>5</td>
<td>72</td>
</tr>
<tr>
<td>systemsettings</td>
<td>45</td>
<td>60</td>
<td>10</td>
<td>44</td>
</tr>
</tbody>
</table>

#### Optimized Rule Subset of kompare

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>65</td>
<td>100</td>
<td>14</td>
<td>63</td>
</tr>
<tr>
<td>ark</td>
<td>49</td>
<td>80</td>
<td>27</td>
<td>41</td>
</tr>
<tr>
<td>elisa</td>
<td>53</td>
<td>71</td>
<td>33</td>
<td>47</td>
</tr>
<tr>
<td>gwenview</td>
<td>59</td>
<td>74</td>
<td>36</td>
<td>53</td>
</tr>
<tr>
<td>juk</td>
<td>48</td>
<td>100</td>
<td>11</td>
<td>45</td>
</tr>
<tr>
<td>k3b</td>
<td>69</td>
<td>72</td>
<td>3</td>
<td>69</td>
</tr>
<tr>
<td>kate</td>
<td>38</td>
<td>73</td>
<td>7</td>
<td>36</td>
</tr>
<tr>
<td>kdelibs</td>
<td>79</td>
<td>34</td>
<td>20</td>
<td>84</td>
</tr>
<tr>
<td>kio-extras</td>
<td>70</td>
<td>73</td>
<td>19</td>
<td>70</td>
</tr>
<tr>
<td>kmix</td>
<td>37</td>
<td>75</td>
<td>6</td>
<td>35</td>
</tr>
<tr>
<td>konsole</td>
<td>57</td>
<td>74</td>
<td>27</td>
<td>53</td>
</tr>
<tr>
<td>konversation</td>
<td>49</td>
<td>50</td>
<td>14</td>
<td>49</td>
</tr>
<tr>
<td>ktorrent</td>
<td>69</td>
<td>80</td>
<td>12</td>
<td>68</td>
</tr>
<tr>
<td>lokalize</td>
<td>52</td>
<td>52</td>
<td>18</td>
<td>52</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>27</td>
<td>69</td>
<td>12</td>
<td>20</td>
</tr>
<tr>
<td>solid</td>
<td>74</td>
<td>50</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>systemsettings</td>
<td>62</td>
<td>40</td>
<td>10</td>
<td>64</td>
</tr>
</tbody>
</table>
## Appendix A: Deep Level Results of Reusing Trained Subset of Rules

### Optimized Rule Subset of console

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>53</td>
<td>100</td>
<td>11</td>
<td>50</td>
</tr>
<tr>
<td>ark</td>
<td>47</td>
<td>80</td>
<td>26</td>
<td>38</td>
</tr>
<tr>
<td>elisa</td>
<td>45</td>
<td>73</td>
<td>30</td>
<td>34</td>
</tr>
<tr>
<td>gwenview</td>
<td>55</td>
<td>79</td>
<td>34</td>
<td>47</td>
</tr>
<tr>
<td>juk</td>
<td>44</td>
<td>100</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>k3b</td>
<td>57</td>
<td>79</td>
<td>3</td>
<td>57</td>
</tr>
<tr>
<td>kate</td>
<td>33</td>
<td>73</td>
<td>6</td>
<td>30</td>
</tr>
<tr>
<td>kdelibs</td>
<td>75</td>
<td>58</td>
<td>23</td>
<td>77</td>
</tr>
<tr>
<td>kio-extras</td>
<td>63</td>
<td>82</td>
<td>17</td>
<td>61</td>
</tr>
<tr>
<td>kmix</td>
<td>32</td>
<td>100</td>
<td>7</td>
<td>28</td>
</tr>
<tr>
<td>kompare</td>
<td>47</td>
<td>100</td>
<td>4</td>
<td>45</td>
</tr>
<tr>
<td>konversation</td>
<td>41</td>
<td>83</td>
<td>18</td>
<td>34</td>
</tr>
<tr>
<td>ktorrent</td>
<td>53</td>
<td>100</td>
<td>10</td>
<td>51</td>
</tr>
<tr>
<td>lokalize</td>
<td>47</td>
<td>60</td>
<td>18</td>
<td>44</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>27</td>
<td>85</td>
<td>14</td>
<td>18</td>
</tr>
<tr>
<td>solid</td>
<td>70</td>
<td>65</td>
<td>5</td>
<td>70</td>
</tr>
<tr>
<td>systemsettings</td>
<td>42</td>
<td>60</td>
<td>9</td>
<td>40</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of konversation

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>62</td>
<td>100</td>
<td>13</td>
<td>59</td>
</tr>
<tr>
<td>ark</td>
<td>51</td>
<td>70</td>
<td>30</td>
<td>46</td>
</tr>
<tr>
<td>elisa</td>
<td>44</td>
<td>65</td>
<td>27</td>
<td>35</td>
</tr>
<tr>
<td>gwenview</td>
<td>63</td>
<td>72</td>
<td>39</td>
<td>60</td>
</tr>
<tr>
<td>juk</td>
<td>48</td>
<td>100</td>
<td>11</td>
<td>45</td>
</tr>
<tr>
<td>k3b</td>
<td>64</td>
<td>72</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>kate</td>
<td>42</td>
<td>73</td>
<td>7</td>
<td>40</td>
</tr>
<tr>
<td>kdelibs</td>
<td>76</td>
<td>58</td>
<td>24</td>
<td>79</td>
</tr>
<tr>
<td>kio-extras</td>
<td>66</td>
<td>75</td>
<td>17</td>
<td>65</td>
</tr>
<tr>
<td>kmix</td>
<td>52</td>
<td>100</td>
<td>10</td>
<td>49</td>
</tr>
<tr>
<td>kompare</td>
<td>54</td>
<td>100</td>
<td>5</td>
<td>52</td>
</tr>
<tr>
<td>konsole</td>
<td>55</td>
<td>80</td>
<td>27</td>
<td>49</td>
</tr>
<tr>
<td>ktorrent</td>
<td>64</td>
<td>100</td>
<td>13</td>
<td>62</td>
</tr>
<tr>
<td>lokalize</td>
<td>53</td>
<td>60</td>
<td>20</td>
<td>52</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>53</td>
<td>85</td>
<td>21</td>
<td>48</td>
</tr>
<tr>
<td>solid</td>
<td>72</td>
<td>64</td>
<td>5</td>
<td>73</td>
</tr>
<tr>
<td>systemsettings</td>
<td>45</td>
<td>60</td>
<td>10</td>
<td>44</td>
</tr>
</tbody>
</table>
## Appendix A: Deep Level Results of Reusing Trained Subset of Rules

### Optimized Rule Subset of ktorrent

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>44</td>
<td>100</td>
<td>10</td>
<td>40</td>
</tr>
<tr>
<td>ark</td>
<td>49</td>
<td>100</td>
<td>30</td>
<td>35</td>
</tr>
<tr>
<td>elisa</td>
<td>53</td>
<td>59</td>
<td>31</td>
<td>51</td>
</tr>
<tr>
<td>gwenview</td>
<td>58</td>
<td>81</td>
<td>36</td>
<td>50</td>
</tr>
<tr>
<td>juk</td>
<td>50</td>
<td>100</td>
<td>11</td>
<td>47</td>
</tr>
<tr>
<td>k3b</td>
<td>64</td>
<td>65</td>
<td>3</td>
<td>64</td>
</tr>
<tr>
<td>kate</td>
<td>36</td>
<td>73</td>
<td>6</td>
<td>34</td>
</tr>
<tr>
<td>kdelibs</td>
<td>78</td>
<td>17</td>
<td>12</td>
<td>85</td>
</tr>
<tr>
<td>kio-extras</td>
<td>68</td>
<td>77</td>
<td>18</td>
<td>67</td>
</tr>
<tr>
<td>kmix</td>
<td>40</td>
<td>100</td>
<td>8</td>
<td>37</td>
</tr>
<tr>
<td>kompare</td>
<td>49</td>
<td>100</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>kde</td>
<td>58</td>
<td>69</td>
<td>26</td>
<td>55</td>
</tr>
<tr>
<td>konversation</td>
<td>49</td>
<td>50</td>
<td>14</td>
<td>49</td>
</tr>
<tr>
<td>lokalize</td>
<td>56</td>
<td>45</td>
<td>18</td>
<td>59</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>33</td>
<td>77</td>
<td>14</td>
<td>27</td>
</tr>
<tr>
<td>solid</td>
<td>71</td>
<td>57</td>
<td>4</td>
<td>72</td>
</tr>
<tr>
<td>systemsettings</td>
<td>46</td>
<td>20</td>
<td>4</td>
<td>48</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of plasma-nm

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>76</td>
<td>100</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>ark</td>
<td>55</td>
<td>60</td>
<td>26</td>
<td>54</td>
</tr>
<tr>
<td>elisa</td>
<td>57</td>
<td>65</td>
<td>35</td>
<td>54</td>
</tr>
<tr>
<td>gwenview</td>
<td>65</td>
<td>74</td>
<td>40</td>
<td>61</td>
</tr>
<tr>
<td>juk</td>
<td>62</td>
<td>100</td>
<td>14</td>
<td>60</td>
</tr>
<tr>
<td>k3b</td>
<td>71</td>
<td>72</td>
<td>3</td>
<td>71</td>
</tr>
<tr>
<td>kate</td>
<td>65</td>
<td>55</td>
<td>9</td>
<td>66</td>
</tr>
<tr>
<td>kde</td>
<td>77</td>
<td>42</td>
<td>21</td>
<td>81</td>
</tr>
<tr>
<td>kio-extras</td>
<td>76</td>
<td>55</td>
<td>19</td>
<td>78</td>
</tr>
<tr>
<td>kmix</td>
<td>55</td>
<td>100</td>
<td>10</td>
<td>52</td>
</tr>
<tr>
<td>kompare</td>
<td>54</td>
<td>100</td>
<td>5</td>
<td>52</td>
</tr>
<tr>
<td>kde</td>
<td>67</td>
<td>60</td>
<td>31</td>
<td>69</td>
</tr>
<tr>
<td>konversation</td>
<td>59</td>
<td>50</td>
<td>18</td>
<td>60</td>
</tr>
<tr>
<td>ktorrent</td>
<td>81</td>
<td>100</td>
<td>22</td>
<td>80</td>
</tr>
<tr>
<td>lokalize</td>
<td>62</td>
<td>52</td>
<td>23</td>
<td>64</td>
</tr>
<tr>
<td>solid</td>
<td>80</td>
<td>57</td>
<td>6</td>
<td>80</td>
</tr>
<tr>
<td>systemsettings</td>
<td>58</td>
<td>40</td>
<td>9</td>
<td>60</td>
</tr>
</tbody>
</table>
A2: MLN Based Fault Proneness Identification (Metrics in Percentage)

<table>
<thead>
<tr>
<th>Project</th>
<th>Optimized Rule Subset of</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>akregator</td>
<td>94</td>
<td>100</td>
<td>50</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>elisa</td>
<td>64</td>
<td>42</td>
<td>36</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>gwenview</td>
<td>65</td>
<td>81</td>
<td>41</td>
<td>59</td>
</tr>
<tr>
<td></td>
<td>juk</td>
<td>81</td>
<td>33</td>
<td>33</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>k3b</td>
<td>94</td>
<td>100</td>
<td>50</td>
<td>94</td>
</tr>
<tr>
<td></td>
<td>kate</td>
<td>74</td>
<td>64</td>
<td>3</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>kdelibs</td>
<td>87</td>
<td>55</td>
<td>22</td>
<td>89</td>
</tr>
<tr>
<td></td>
<td>kio-extras</td>
<td>72</td>
<td>58</td>
<td>21</td>
<td>74</td>
</tr>
<tr>
<td></td>
<td>kmix</td>
<td>66</td>
<td>57</td>
<td>14</td>
<td>67</td>
</tr>
<tr>
<td></td>
<td>kompare</td>
<td>84</td>
<td>75</td>
<td>20</td>
<td>84</td>
</tr>
<tr>
<td></td>
<td>konsole</td>
<td>9</td>
<td>100</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>konversation</td>
<td>70</td>
<td>66</td>
<td>34</td>
<td>71</td>
</tr>
<tr>
<td></td>
<td>ktorrent</td>
<td>62</td>
<td>100</td>
<td>12</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>lokalize</td>
<td>73</td>
<td>48</td>
<td>31</td>
<td>78</td>
</tr>
<tr>
<td></td>
<td>plasma-nm</td>
<td>74</td>
<td>69</td>
<td>30</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>solid</td>
<td>73</td>
<td>71</td>
<td>6</td>
<td>73</td>
</tr>
<tr>
<td></td>
<td>systemsettings</td>
<td>53</td>
<td>40</td>
<td>8</td>
<td>54</td>
</tr>
</tbody>
</table>
### Appendix A: Deep Level Results of Reusing Trained Subset of Rules

#### Optimized Rule Subset of elisa

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>82</td>
<td>50</td>
<td>17</td>
<td>84</td>
</tr>
<tr>
<td>ark</td>
<td>72</td>
<td>70</td>
<td>41</td>
<td>73</td>
</tr>
<tr>
<td>gwenview</td>
<td>71</td>
<td>60</td>
<td>46</td>
<td>75</td>
</tr>
<tr>
<td>juk</td>
<td>83</td>
<td>50</td>
<td>43</td>
<td>89</td>
</tr>
<tr>
<td>k3b</td>
<td>78</td>
<td>67</td>
<td>17</td>
<td>79</td>
</tr>
<tr>
<td>kate</td>
<td>68</td>
<td>71</td>
<td>3</td>
<td>68</td>
</tr>
<tr>
<td>kdelibs</td>
<td>70</td>
<td>45</td>
<td>9</td>
<td>72</td>
</tr>
<tr>
<td>kio-extras</td>
<td>60</td>
<td>83</td>
<td>19</td>
<td>57</td>
</tr>
<tr>
<td>kmix</td>
<td>72</td>
<td>66</td>
<td>19</td>
<td>72</td>
</tr>
<tr>
<td>kompare</td>
<td>71</td>
<td>100</td>
<td>15</td>
<td>69</td>
</tr>
<tr>
<td>kio</td>
<td>60</td>
<td>83</td>
<td>19</td>
<td>57</td>
</tr>
<tr>
<td>konsole</td>
<td>47</td>
<td>100</td>
<td>4</td>
<td>45</td>
</tr>
<tr>
<td>konversation</td>
<td>72</td>
<td>60</td>
<td>36</td>
<td>75</td>
</tr>
<tr>
<td>ktorrent</td>
<td>90</td>
<td>60</td>
<td>27</td>
<td>91</td>
</tr>
<tr>
<td>lokalize</td>
<td>64</td>
<td>52</td>
<td>24</td>
<td>66</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>85</td>
<td>39</td>
<td>45</td>
<td>93</td>
</tr>
<tr>
<td>solid</td>
<td>88</td>
<td>43</td>
<td>8</td>
<td>89</td>
</tr>
<tr>
<td>systemsettings</td>
<td>65</td>
<td>60</td>
<td>15</td>
<td>66</td>
</tr>
</tbody>
</table>

#### Optimized Rule Subset of juk

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>97</td>
<td>100</td>
<td>67</td>
<td>97</td>
</tr>
<tr>
<td>ark</td>
<td>77</td>
<td>70</td>
<td>47</td>
<td>78</td>
</tr>
<tr>
<td>elisa</td>
<td>66</td>
<td>50</td>
<td>40</td>
<td>72</td>
</tr>
<tr>
<td>gwenview</td>
<td>71</td>
<td>59</td>
<td>45</td>
<td>75</td>
</tr>
<tr>
<td>k3b</td>
<td>78</td>
<td>100</td>
<td>21</td>
<td>77</td>
</tr>
<tr>
<td>kate</td>
<td>81</td>
<td>64</td>
<td>5</td>
<td>82</td>
</tr>
<tr>
<td>kdelibs</td>
<td>74</td>
<td>55</td>
<td>11</td>
<td>75</td>
</tr>
<tr>
<td>kio-extras</td>
<td>83</td>
<td>50</td>
<td>32</td>
<td>87</td>
</tr>
<tr>
<td>kmix</td>
<td>80</td>
<td>55</td>
<td>23</td>
<td>82</td>
</tr>
<tr>
<td>kompare</td>
<td>63</td>
<td>100</td>
<td>12</td>
<td>61</td>
</tr>
<tr>
<td>konsole</td>
<td>37</td>
<td>100</td>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>konversation</td>
<td>75</td>
<td>66</td>
<td>40</td>
<td>77</td>
</tr>
<tr>
<td>ktorrent</td>
<td>68</td>
<td>60</td>
<td>9</td>
<td>68</td>
</tr>
<tr>
<td>lokalize</td>
<td>70</td>
<td>56</td>
<td>29</td>
<td>73</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>64</td>
<td>100</td>
<td>27</td>
<td>58</td>
</tr>
<tr>
<td>solid</td>
<td>89</td>
<td>50</td>
<td>10</td>
<td>90</td>
</tr>
<tr>
<td>systemsettings</td>
<td>49</td>
<td>100</td>
<td>15</td>
<td>44</td>
</tr>
</tbody>
</table>
Appendix A: Deep Level Results of Reusing Trained Subset of Rules

### Optimized Rule Subset of k3b

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>74</td>
<td>100</td>
<td>18</td>
<td>72</td>
</tr>
<tr>
<td>ark</td>
<td>53</td>
<td>80</td>
<td>29</td>
<td>46</td>
</tr>
<tr>
<td>elisa</td>
<td>62</td>
<td>63</td>
<td>38</td>
<td>62</td>
</tr>
<tr>
<td>gwenview</td>
<td>60</td>
<td>74</td>
<td>37</td>
<td>55</td>
</tr>
<tr>
<td>juk</td>
<td>78</td>
<td>67</td>
<td>36</td>
<td>80</td>
</tr>
<tr>
<td>kate</td>
<td>70</td>
<td>79</td>
<td>3</td>
<td>70</td>
</tr>
<tr>
<td>kdelibs</td>
<td>48</td>
<td>36</td>
<td>4</td>
<td>48</td>
</tr>
<tr>
<td>kio-extras</td>
<td>61</td>
<td>58</td>
<td>15</td>
<td>61</td>
</tr>
<tr>
<td>kmix</td>
<td>71</td>
<td>68</td>
<td>19</td>
<td>71</td>
</tr>
<tr>
<td>kompare</td>
<td>48</td>
<td>100</td>
<td>9</td>
<td>45</td>
</tr>
<tr>
<td>konsole</td>
<td>70</td>
<td>100</td>
<td>7</td>
<td>69</td>
</tr>
<tr>
<td>konversation</td>
<td>69</td>
<td>57</td>
<td>32</td>
<td>72</td>
</tr>
<tr>
<td>ktorrent</td>
<td>84</td>
<td>100</td>
<td>25</td>
<td>84</td>
</tr>
<tr>
<td>lokalize</td>
<td>48</td>
<td>67</td>
<td>20</td>
<td>45</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>67</td>
<td>62</td>
<td>23</td>
<td>67</td>
</tr>
<tr>
<td>solid</td>
<td>81</td>
<td>50</td>
<td>6</td>
<td>82</td>
</tr>
<tr>
<td>systemsettings</td>
<td>58</td>
<td>60</td>
<td>13</td>
<td>58</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of kmix

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>76</td>
<td>100</td>
<td>20</td>
<td>75</td>
</tr>
<tr>
<td>ark</td>
<td>79</td>
<td>50</td>
<td>50</td>
<td>86</td>
</tr>
<tr>
<td>elisa</td>
<td>66</td>
<td>48</td>
<td>40</td>
<td>72</td>
</tr>
<tr>
<td>gwenview</td>
<td>60</td>
<td>64</td>
<td>35</td>
<td>58</td>
</tr>
<tr>
<td>juk</td>
<td>56</td>
<td>67</td>
<td>20</td>
<td>54</td>
</tr>
<tr>
<td>k3b</td>
<td>78</td>
<td>100</td>
<td>21</td>
<td>77</td>
</tr>
<tr>
<td>kate</td>
<td>59</td>
<td>86</td>
<td>3</td>
<td>58</td>
</tr>
<tr>
<td>kdelibs</td>
<td>77</td>
<td>45</td>
<td>11</td>
<td>79</td>
</tr>
<tr>
<td>kio-extras</td>
<td>75</td>
<td>50</td>
<td>21</td>
<td>77</td>
</tr>
<tr>
<td>kompare</td>
<td>72</td>
<td>100</td>
<td>15</td>
<td>71</td>
</tr>
<tr>
<td>konsole</td>
<td>67</td>
<td>100</td>
<td>7</td>
<td>67</td>
</tr>
<tr>
<td>konversation</td>
<td>78</td>
<td>54</td>
<td>43</td>
<td>84</td>
</tr>
<tr>
<td>ktorrent</td>
<td>89</td>
<td>80</td>
<td>29</td>
<td>89</td>
</tr>
<tr>
<td>lokalize</td>
<td>66</td>
<td>48</td>
<td>25</td>
<td>70</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>77</td>
<td>77</td>
<td>34</td>
<td>77</td>
</tr>
<tr>
<td>solid</td>
<td>84</td>
<td>71</td>
<td>10</td>
<td>85</td>
</tr>
<tr>
<td>systemsettings</td>
<td>47</td>
<td>60</td>
<td>10</td>
<td>46</td>
</tr>
</tbody>
</table>
## Appendix A: Deep Level Results of Reusing Trained Subset of Rules

**Optimized Rule Subset of kompare**

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>82</td>
<td>100</td>
<td>25</td>
<td>81</td>
</tr>
<tr>
<td>ark</td>
<td>62</td>
<td>90</td>
<td>35</td>
<td>54</td>
</tr>
<tr>
<td>elisa</td>
<td>64</td>
<td>48</td>
<td>38</td>
<td>70</td>
</tr>
<tr>
<td>gwenview</td>
<td>63</td>
<td>57</td>
<td>36</td>
<td>65</td>
</tr>
<tr>
<td>juk</td>
<td>73</td>
<td>50</td>
<td>27</td>
<td>77</td>
</tr>
<tr>
<td>k3b</td>
<td>70</td>
<td>100</td>
<td>17</td>
<td>68</td>
</tr>
<tr>
<td>kate</td>
<td>54</td>
<td>86</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>kdelibs</td>
<td>69</td>
<td>45</td>
<td>8</td>
<td>70</td>
</tr>
<tr>
<td>kio-extras</td>
<td>75</td>
<td>42</td>
<td>19</td>
<td>79</td>
</tr>
<tr>
<td>kmix</td>
<td>78</td>
<td>57</td>
<td>21</td>
<td>80</td>
</tr>
<tr>
<td>konsole</td>
<td>40</td>
<td>100</td>
<td>4</td>
<td>38</td>
</tr>
<tr>
<td>konversation</td>
<td>74</td>
<td>46</td>
<td>35</td>
<td>81</td>
</tr>
<tr>
<td>ktorrent</td>
<td>69</td>
<td>100</td>
<td>14</td>
<td>67</td>
</tr>
<tr>
<td>lokalize</td>
<td>61</td>
<td>56</td>
<td>23</td>
<td>62</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>75</td>
<td>62</td>
<td>30</td>
<td>77</td>
</tr>
<tr>
<td>solid</td>
<td>60</td>
<td>79</td>
<td>4</td>
<td>59</td>
</tr>
<tr>
<td>systemsettings</td>
<td>89</td>
<td>20</td>
<td>33</td>
<td>96</td>
</tr>
</tbody>
</table>

**Optimized Rule Subset of konsole**

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>88</td>
<td>100</td>
<td>33</td>
<td>88</td>
</tr>
<tr>
<td>ark</td>
<td>70</td>
<td>60</td>
<td>38</td>
<td>73</td>
</tr>
<tr>
<td>elisa</td>
<td>71</td>
<td>46</td>
<td>48</td>
<td>81</td>
</tr>
<tr>
<td>gwenview</td>
<td>65</td>
<td>67</td>
<td>39</td>
<td>64</td>
</tr>
<tr>
<td>juk</td>
<td>63</td>
<td>67</td>
<td>24</td>
<td>63</td>
</tr>
<tr>
<td>k3b</td>
<td>80</td>
<td>100</td>
<td>23</td>
<td>79</td>
</tr>
<tr>
<td>kate</td>
<td>66</td>
<td>79</td>
<td>3</td>
<td>66</td>
</tr>
<tr>
<td>kdelibs</td>
<td>69</td>
<td>45</td>
<td>8</td>
<td>70</td>
</tr>
<tr>
<td>kio-extras</td>
<td>66</td>
<td>67</td>
<td>19</td>
<td>66</td>
</tr>
<tr>
<td>kmix</td>
<td>76</td>
<td>64</td>
<td>21</td>
<td>77</td>
</tr>
<tr>
<td>kompare</td>
<td>82</td>
<td>75</td>
<td>19</td>
<td>83</td>
</tr>
<tr>
<td>konversation</td>
<td>71</td>
<td>66</td>
<td>35</td>
<td>72</td>
</tr>
<tr>
<td>ktorrent</td>
<td>78</td>
<td>80</td>
<td>17</td>
<td>78</td>
</tr>
<tr>
<td>lokalize</td>
<td>76</td>
<td>48</td>
<td>35</td>
<td>82</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>73</td>
<td>62</td>
<td>28</td>
<td>75</td>
</tr>
<tr>
<td>solid</td>
<td>90</td>
<td>50</td>
<td>11</td>
<td>91</td>
</tr>
<tr>
<td>systemsettings</td>
<td>40</td>
<td>80</td>
<td>11</td>
<td>36</td>
</tr>
</tbody>
</table>
### Optimized Rule Subset of konversation

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>71</td>
<td>50</td>
<td>10</td>
<td>72</td>
</tr>
<tr>
<td>ark</td>
<td>74</td>
<td>60</td>
<td>43</td>
<td>78</td>
</tr>
<tr>
<td>elisa</td>
<td>61</td>
<td>67</td>
<td>38</td>
<td>58</td>
</tr>
<tr>
<td>gwenview</td>
<td>71</td>
<td>59</td>
<td>46</td>
<td>76</td>
</tr>
<tr>
<td>juk</td>
<td>80</td>
<td>67</td>
<td>40</td>
<td>83</td>
</tr>
<tr>
<td>k3b</td>
<td>86</td>
<td>67</td>
<td>25</td>
<td>87</td>
</tr>
<tr>
<td>kate</td>
<td>88</td>
<td>43</td>
<td>5</td>
<td>89</td>
</tr>
<tr>
<td>kdelibs</td>
<td>63</td>
<td>54</td>
<td>8</td>
<td>63</td>
</tr>
<tr>
<td>kio-extras</td>
<td>64</td>
<td>67</td>
<td>18</td>
<td>64</td>
</tr>
<tr>
<td>kmix</td>
<td>62</td>
<td>73</td>
<td>15</td>
<td>60</td>
</tr>
<tr>
<td>kompare</td>
<td>68</td>
<td>75</td>
<td>11</td>
<td>68</td>
</tr>
<tr>
<td>konsole</td>
<td>44</td>
<td>100</td>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>ktorrent</td>
<td>78</td>
<td>100</td>
<td>19</td>
<td>77</td>
</tr>
<tr>
<td>lokalize</td>
<td>69</td>
<td>44</td>
<td>26</td>
<td>74</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>60</td>
<td>62</td>
<td>20</td>
<td>60</td>
</tr>
<tr>
<td>solid</td>
<td>92</td>
<td>43</td>
<td>13</td>
<td>93</td>
</tr>
<tr>
<td>systemsettings</td>
<td>42</td>
<td>60</td>
<td>9</td>
<td>40</td>
</tr>
</tbody>
</table>

### Optimized Rule Subset of ktorrent

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>68</td>
<td>100</td>
<td>15</td>
<td>66</td>
</tr>
<tr>
<td>ark</td>
<td>64</td>
<td>90</td>
<td>36</td>
<td>57</td>
</tr>
<tr>
<td>elisa</td>
<td>58</td>
<td>54</td>
<td>34</td>
<td>60</td>
</tr>
<tr>
<td>gwenview</td>
<td>66</td>
<td>57</td>
<td>39</td>
<td>69</td>
</tr>
<tr>
<td>juk</td>
<td>61</td>
<td>50</td>
<td>19</td>
<td>63</td>
</tr>
<tr>
<td>k3b</td>
<td>90</td>
<td>100</td>
<td>38</td>
<td>89</td>
</tr>
<tr>
<td>kate</td>
<td>77</td>
<td>71</td>
<td>4</td>
<td>77</td>
</tr>
<tr>
<td>kdelibs</td>
<td>71</td>
<td>45</td>
<td>9</td>
<td>72</td>
</tr>
<tr>
<td>kio-extras</td>
<td>82</td>
<td>50</td>
<td>29</td>
<td>85</td>
</tr>
<tr>
<td>kmix</td>
<td>71</td>
<td>68</td>
<td>19</td>
<td>71</td>
</tr>
<tr>
<td>kompare</td>
<td>95</td>
<td>50</td>
<td>50</td>
<td>97</td>
</tr>
<tr>
<td>konsole</td>
<td>98</td>
<td>100</td>
<td>50</td>
<td>98</td>
</tr>
<tr>
<td>konversation</td>
<td>63</td>
<td>74</td>
<td>30</td>
<td>61</td>
</tr>
<tr>
<td>lokalize</td>
<td>57</td>
<td>59</td>
<td>22</td>
<td>57</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>77</td>
<td>54</td>
<td>30</td>
<td>81</td>
</tr>
<tr>
<td>solid</td>
<td>95</td>
<td>43</td>
<td>19</td>
<td>96</td>
</tr>
<tr>
<td>systemsettings</td>
<td>51</td>
<td>80</td>
<td>13</td>
<td>48</td>
</tr>
</tbody>
</table>
Appendix A: Deep Level Results of Reusing Trained Subset of Rules

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>82</td>
<td>100</td>
<td>25</td>
<td>81</td>
</tr>
<tr>
<td>ark</td>
<td>66</td>
<td>90</td>
<td>38</td>
<td>59</td>
</tr>
<tr>
<td>elisa</td>
<td>69</td>
<td>50</td>
<td>44</td>
<td>76</td>
</tr>
<tr>
<td>gwenview</td>
<td>70</td>
<td>62</td>
<td>44</td>
<td>72</td>
</tr>
<tr>
<td>juk</td>
<td>71</td>
<td>67</td>
<td>29</td>
<td>71</td>
</tr>
<tr>
<td>k3b</td>
<td>84</td>
<td>100</td>
<td>27</td>
<td>83</td>
</tr>
<tr>
<td>kate</td>
<td>75</td>
<td>79</td>
<td>4</td>
<td>75</td>
</tr>
<tr>
<td>kdelibs</td>
<td>77</td>
<td>55</td>
<td>13</td>
<td>78</td>
</tr>
<tr>
<td>kio-extras</td>
<td>66</td>
<td>58</td>
<td>17</td>
<td>67</td>
</tr>
<tr>
<td>kmix</td>
<td>63</td>
<td>68</td>
<td>15</td>
<td>63</td>
</tr>
<tr>
<td>kompare</td>
<td>66</td>
<td>100</td>
<td>13</td>
<td>64</td>
</tr>
<tr>
<td>konsole</td>
<td>14</td>
<td>100</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>konversation</td>
<td>75</td>
<td>54</td>
<td>39</td>
<td>80</td>
</tr>
<tr>
<td>ktorrent</td>
<td>84</td>
<td>100</td>
<td>25</td>
<td>84</td>
</tr>
<tr>
<td>lokalize</td>
<td>64</td>
<td>56</td>
<td>25</td>
<td>66</td>
</tr>
<tr>
<td>solid</td>
<td>69</td>
<td>71</td>
<td>5</td>
<td>69</td>
</tr>
<tr>
<td>systemsettings</td>
<td>53</td>
<td>60</td>
<td>11</td>
<td>52</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project</th>
<th>Accuracy</th>
<th>Recall</th>
<th>Precision</th>
<th>Neg. Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>akregator</td>
<td>91</td>
<td>100</td>
<td>40</td>
<td>91</td>
</tr>
<tr>
<td>ark</td>
<td>60</td>
<td>100</td>
<td>34</td>
<td>49</td>
</tr>
<tr>
<td>elisa</td>
<td>56</td>
<td>54</td>
<td>32</td>
<td>57</td>
</tr>
<tr>
<td>gwenview</td>
<td>71</td>
<td>45</td>
<td>44</td>
<td>80</td>
</tr>
<tr>
<td>juk</td>
<td>61</td>
<td>50</td>
<td>19</td>
<td>63</td>
</tr>
<tr>
<td>k3b</td>
<td>88</td>
<td>100</td>
<td>33</td>
<td>87</td>
</tr>
<tr>
<td>kate</td>
<td>69</td>
<td>79</td>
<td>3</td>
<td>69</td>
</tr>
<tr>
<td>kdelibs</td>
<td>45</td>
<td>82</td>
<td>8</td>
<td>42</td>
</tr>
<tr>
<td>kio-extras</td>
<td>71</td>
<td>75</td>
<td>23</td>
<td>71</td>
</tr>
<tr>
<td>kmix</td>
<td>68</td>
<td>71</td>
<td>18</td>
<td>68</td>
</tr>
<tr>
<td>kompare</td>
<td>86</td>
<td>75</td>
<td>23</td>
<td>87</td>
</tr>
<tr>
<td>konsole</td>
<td>86</td>
<td>100</td>
<td>14</td>
<td>86</td>
</tr>
<tr>
<td>konversation</td>
<td>70</td>
<td>71</td>
<td>35</td>
<td>69</td>
</tr>
<tr>
<td>ktorrent</td>
<td>67</td>
<td>80</td>
<td>11</td>
<td>66</td>
</tr>
<tr>
<td>lokalize</td>
<td>68</td>
<td>52</td>
<td>27</td>
<td>71</td>
</tr>
<tr>
<td>plasma-nm</td>
<td>81</td>
<td>31</td>
<td>31</td>
<td>89</td>
</tr>
<tr>
<td>systemsettings</td>
<td>42</td>
<td>80</td>
<td>11</td>
<td>38</td>
</tr>
</tbody>
</table>
Appendix B: Expanded Rules for Each Project

Note: The customized expanded rules for each project are shown in bold.

B1: Fuzzy Based Fault Proneness Identification

Akregator:

RULE 1: IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 3: if failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF NOT(frequencyOfCommits IS high OR frequencyOfMerges IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR fractalValue IS high OR distinctAuthors IS high) THEN isSegmentBuggy IS notBuggy;
RULE 6: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 8: IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 9: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 10: IF distinctAuthors IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 11: IF overallStrength IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 12: IF overallStrength IS avg AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 13: IF frequencyOfCommits IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 14: IF fractalValue IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;

Ark:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 5: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 7: if overallStrength is avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 8: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 9: IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 10 : IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 11 : IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 12 : IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 13 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 14 : IF frequencyOfCommits IS avg AND fileChurn IS avg AND distinctAuthors IS avg THEN isSegmentBuggy IS maybe;
RULE 15 : IF frequencyOfMerges IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;

Elisa:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3 : if failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 4 : IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 5 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 7 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 8 : IF distinctAuthors IS high AND fractalValue IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 9 : IF frequencyOfMerges IS high AND (overallStrength IS avg OR overallStrength IS high) THEN isSegmentBuggy IS maybe;
RULE 10 : IF overallStrength IS high AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS maybe;
RULE 11 : IF (frequencyOfCommits IS low AND overallStrength IS low AND frequencyOfMerges IS low AND fileChurn IS low AND distinctAuthors IS low) THEN isSegmentBuggy IS notBuggy;
RULE 12 : IF frequencyOfCommits IS avg AND overallStrength IS low AND frequencyOfMerges IS low AND fractalValue IS low AND distinctAuthors IS low THEN isSegmentBuggy IS notBuggy;

Gwenview:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4 : IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 5 : IF NOT(frequencyOfCommits IS high OR frequencyOfMerges IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR distinctAuthors IS high OR fractalValue IS high) THEN isSegmentBuggy IS notBuggy;
RULE 6 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS buggy;
RULE 7 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 8: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 9: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 10: IF overallStrength IS high OR overallStrength IS avg AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 11: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 12: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 13: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 14: IF frequencyOfCommits IS avg AND overallStrength IS avg AND fileChurn IS avg THEN isSegmentBuggy IS buggy;
RULE 15: IF frequencyOfMerges IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 16: IF fractalValue IS high AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;

Juk:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 5: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 7: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 8: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 9: IF fileChurn IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 10: IF overallStrength IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

K3b:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 4: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 6: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 7: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
Appendix B: Expanded Rules for Each Project

RULE 8: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 9: IF frequencyOfMerges IS avg AND (fileChurn IS high OR fileChurn IS avg) AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10: IF frequencyOfMerges IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

Kate:

RULE 1: IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF overallStrength is avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 8: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 9: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10: IF overallStrength IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 11: IF frequencyOfMerges IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 12: IF distinctAuthors IS high AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;

Kdelibs:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 8: IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

Kio-extras:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 5: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
Appendix B: Expanded Rules for Each Project

RULE 6 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 8 : IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 9 : IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 11 : IF frequencyOfMerges IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;

Kmix:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 4 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 5 : IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6 : IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : if failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 8 : if overallStrength is avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 9 : IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 10 : IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS maybe;
RULE 11 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 12 : IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 13 : IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 14 : IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

Kompare:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4 : IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 5 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
Appendix B: Expanded Rules for Each Project

RULE 8: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 9: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;

Konsole:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 6: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 8: IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 9: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 10: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 11: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 12: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 13: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 14: IF frequencyOfCommits IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 15: IF frequencyOfMerges IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

Konversation:

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 3: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF overallStrength IS avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 6: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 8: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 9: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 10: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
Appendix B: Expanded Rules for Each Project

RULE 11: IF frequencyOfCommits IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 12: IF overallStrength IS high AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;

Ktorrent:
RULE 1: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 2: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 3: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 4: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 5: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 6: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;

Lokalize:
RULE 1: IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS maybe;
RULE 3: IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 6: IF overallStrength IS avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 8: IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 9: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 10: IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 11: IF overallStrength IS avg AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
RULE 12: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 13: IF distinctAuthors IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS buggy;
RULE 14: IF overallStrength IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 15: IF frequencyOfCommits IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 16: IF frequencyOfCommits IS avg AND fractalValue IS high THEN isSegmentBuggy IS maybe;
RULE 17: IF frequencyOfCommits IS avg AND overallStrength IS high THEN isSegmentBuggy IS maybe;
RULE 18: IF overallStrength IS avg AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
Plasma-nm:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF fileChurn IS high AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4 : IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 5 : if failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 6 : if overallStrength is avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF fileChurn IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 8 : IF frequencyOfCommits IS avg AND (fractalValue IS avg OR fractalValue IS high) AND frequencyOfCommits IS high THEN isSegmentBuggy IS maybe;
RULE 9 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 10 : IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 11 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 12 : IF frequencyOfCommits IS avg AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;

Solid:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity2ndSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF fileChurn IS avg AND failureIntensity1stSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4 : IF NOT(frequencyOfCommits IS high OR frequencyOfMerges IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR distinctAuthors IS high) THEN isSegmentBuggy IS notBuggy;
RULE 5 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6 : IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 7 : IF (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 8 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 9 : IF frequencyOfCommits IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10 : IF frequencyOfCommits IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;

138
Appendix B: Expanded Rules for Each Project

Systemsettings:

RULE 1 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (failureIntensity1stSegment IS high OR failureIntensity1stSegment IS high) AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfMerges IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND failureIntensity1stSegment IS high AND (overallStrength IS high OR overallStrength IS avg) AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 4 : if failureIntensity1stSegment IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS maybe;
RULE 5 : IF NOT(frequencyOfCommits IS high OR frequencyOfCommits IS high OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR fractalValue IS high) THEN isSegmentBuggy IS notBuggy;
RULE 6 : IF (fileChurn IS low OR fileChurn IS avg) AND (overallStrength IS high OR overallStrength IS avg) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 8 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 9 : IF fractalValue IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;

B2: MLN Based Fault Proneness Identification (With Weights)

Akregator:

// -1.24626  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.515069  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.28011  highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
1.28011  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.0796974  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.0796974  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.94953  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
1.94953  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.309298  !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v highFrequencyOfCommits(s) v highFileChurn(s)) => !isBuggy(s)
0.309298  highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v highFrequencyOfMerges(a1) v highFailureIntensity(a1) v !isBuggy(a1)

// 1.2437  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) => isBuggy(s)
1.2437  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.72634  highFailureIntensity(s) => isBuggy(s)
1.72634  !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 4.35292 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
2.52422 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.87528 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.613725 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.397618 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.669066 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.561561 highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.561561 !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// -0.180339 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.235131 !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// -3.34474 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
0.152046 !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// -0.616703 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.195893 !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.435887 avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.435887 !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.3856 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) =>
isBuggy(s)
0.926581 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.459016 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

Ark:

// 1.54102 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0.3794 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
1.16162 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 3.24304 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
1.21182 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
2.03122 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.471984 highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
0.471984 !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.479985 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.479985 !highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -0.291873 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.245589 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

0.116701 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.36665 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
1.05249 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.0659994 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.7895 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
1.7895 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 1.84423 highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
1.84423 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.424177 !highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.424177 !highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v highFrequencyOfMerges(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// -0.127884 avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) => isBuggy(s)
0 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.15928 highFailureIntensity(s) => isBuggy(s)
1.15928 !highFailureIntensity(a1) v isBuggy(a1)

// 0.807241 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.28845 !lowFileChurn(a1) v !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
1.43949 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.655609 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.221032 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.662935 !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.288708 !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.968302 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.977481 !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.557264 avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.557264 !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.78757 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.780029 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
1.00754 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.0737715 avgFrequencyOfCommits(s) ^ avgFileChurn(s) ^ avgDistinctAuthors(s) => isBuggy(s)
0.0737715 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgDistinctAuthors(a1) v isBuggy(a1)
Elisa:

// 2.2532  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
2.21275  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.040448  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -0.0626866  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
0.444838  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.61096  highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
1.61096  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.908708  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.486251  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0.482224  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0.412613  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
1.4594  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1)
v isBuggy(a1)

// 0.100956  avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.100956  !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.379435  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.379435  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.455362  highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.455362  !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.66012  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.66012  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.165014  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
0.165014  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 1.02197  !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v
highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
1.02197  highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v
highFrequencyOfMerges(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// 1.63025  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
2.22681  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.833377  highFailureIntensity(s) => isBuggy(s)
0.833377  !highFailureIntensity(a1) v isBuggy(a1)

142
Appendix B: Expanded Rules for Each Project

// 5.62338 \((\text{lowFileChurn}(s) \lor \text{avgFileChurn}(s)) \land (\text{highOverallStrength}(s) \lor \text{avgOverallStrength}(s)) \land (\text{highFrequencyOfCommits}(s) \lor \text{avgFrequencyOfCommits}(s)) \Rightarrow \text{isBuggy}(s)\)
0.076375 !\text{lowFileChurn}(a1) \lor !\text{highFrequencyOfCommits}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)
1.01882 \text{avgFileChurn}(a1) \lor !\text{highFrequencyOfCommits}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)
1.62627 !\text{lowFileChurn}(a1) \lor \text{highFrequencyOfCommits}(a1) \lor !\text{avgOverallStrength}(a1) \lor \text{isBuggy}(a1)
0.697938 \text{avgFileChurn}(a1) \lor !\text{highFrequencyOfCommits}(a1) \lor !\text{avgOverallStrength}(a1) \lor \text{isBuggy}(a1)
1.12763 !\text{lowFileChurn}(a1) \lor \text{avgFrequencyOfCommits}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)
0.363501 !\text{lowFileChurn}(a1) \lor !\text{highFrequencyOfCommits}(a1) \lor !\text{avgOverallStrength}(a1) \lor \text{isBuggy}(a1)
1.39818 \text{avgFileChurn}(a1) \lor !\text{avgFrequencyOfCommits}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)

// 0.127707 \text{highFrequencyOfMerges}(s) \land \text{highOverallStrength}(s) \Rightarrow \text{isBuggy}(s)
0.127707 !\text{highOverallStrength}(a1) \lor !\text{highFrequencyOfMerges}(a1) \lor \text{isBuggy}(a1)

// -0.675535 (\text{highOverallStrength}(s) \lor \text{avgOverallStrength}(s)) \land (\text{avgFileChurn}(s) \lor \text{highFileChurn}(s)) \Rightarrow \text{isBuggy}(s)
0.668858 !\text{highFileChurn}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)

// -0.0949396 \text{avgOverallStrength}(s) \land (\text{avgFractalValue}(s) \lor \text{highFractalValue}(s)) \Rightarrow \text{isBuggy}(s)
0.496101 !\text{avgOverallStrength}(a1) \lor !\text{highFractalValue}(a1) \lor \text{isBuggy}(a1)

// 0.198563 \text{avgFrequencyOfCommits}(s) \land \text{avgFractalValue}(s) \Rightarrow \text{isBuggy}(s)
0.198563 !\text{avgFrequencyOfCommits}(a1) \lor !\text{avgFractalValue}(a1) \lor \text{isBuggy}(a1)

// 0.333071 \text{highDistinctAuthors}(s) \land (\text{highFrequencyOfCommits}(s) \lor \text{avgFrequencyOfCommits}(s)) \Rightarrow \text{isBuggy}(s)
0.128884 !\text{highFrequencyOfCommits}(a1) \lor !\text{highDistinctAuthors}(a1) \lor \text{isBuggy}(a1)
0.204487 !\text{avgFrequencyOfCommits}(a1) \lor !\text{highDistinctAuthors}(a1) \lor \text{isBuggy}(a1)

// 2.95958 \text{highDistinctAuthors}(s) \land \text{highFractalValue}(s) \land \text{highOverallStrength}(s) \Rightarrow \text{isBuggy}(s)
2.95958 !\text{highFractalValue}(a1) \lor !\text{highDistinctAuthors}(a1) \lor \text{isBuggy}(a1)

// 0.523969 \text{highFrequencyOfMerges}(s) \land (\text{highOverallStrength}(s) \lor \text{highOverallStrength}(s)) \Rightarrow \text{isBuggy}(s)
0.396262 !\text{highOverallStrength}(a1) \lor !\text{highFrequencyOfMerges}(a1) \lor \text{isBuggy}(a1)
0.127707 !\text{highOverallStrength}(a1) \lor !\text{highFrequencyOfMerges}(a1) \lor \text{isBuggy}(a1)

// 0.382603 \text{highOverallStrength}(s) \land (\text{highFileChurn}(s) \lor \text{avgFileChurn}(s)) \Rightarrow \text{isBuggy}(s)
0.668858 !\text{highFileChurn}(a1) \lor !\text{highOverallStrength}(a1) \lor \text{isBuggy}(a1)

// 0.0651108 \text{lowFrequencyOfCommits}(s) \land \text{lowOverallStrength}(s) \land \text{lowFrequencyOfMerges}(s) \land \text{lowFileChurn}(s) \land \text{lowDistinctAuthors}(s) \Rightarrow \text{isBuggy}(s)
0.0651108 !\text{lowFileChurn}(a1) \lor !\text{lowFrequencyOfCommits}(a1) \lor !\text{lowOverallStrength}(a1) \lor !\text{lowFrequencyOfMerges}(a1) \lor !\text{lowDistinctAuthors}(a1) \lor \text{isBuggy}(a1)

Gwenview:

// 1.03399 (\text{highFrequencyOfCommits}(s) \lor \text{avgFrequencyOfCommits}(s)) \land \text{highFailureIntensity}(s) \land \text{highOverallStrength}(s) \Rightarrow \text{isBuggy}(s)
1.71804 !\text{highFrequencyOfCommits}(a1) \lor !\text{highOverallStrength}(a1) \lor !\text{highFailureIntensity}(a1) \lor \text{isBuggy}(a1)

// 0.90729 \text{highFrequencyOfMerges}(s) \land \text{highFileChurn}(s) \Rightarrow \text{isBuggy}(s)
0.90729 !\text{highFileChurn}(a1) \lor !\text{highFrequencyOfMerges}(a1) \lor \text{isBuggy}(a1)
Appendix B: Expanded Rules for Each Project

// 0.575218 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.575218 !highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.250804 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.28553 |highFileChurn(a1) v !highFrequencyOfCommits(a1) v highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0 |highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
1.2072 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0.728161 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)

// 0.103448 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.103448 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 1.00164 highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
1.00164 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.792915 !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v
highFrequencyOfCommits(s) v avgFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.792915 highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v
highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v !avgFileChurn(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)

// 0.302557 highFailureIntensity(s) => isBuggy(s)
0.302557 !highFailureIntensity(a1) v isBuggy(a1)

// 0.460899 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.138282 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgFrequencyOfCommits(a1) v !isBuggy(a1)
0.432994 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFrequencyOfCommits(a1) v !isBuggy(a1)
0.808311 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgFrequencyOfCommits(a1) v !isBuggy(a1)
0.342952 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFrequencyOfCommits(a1) v !isBuggy(a1)
0.639058 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFrequencyOfCommits(a1) v !isBuggy(a1)

// -0.614495 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.0622599 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 0.144705 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
0.139622 !avgFileChurn(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
0.347548 !highFileChurn(a1) v !highOverallStrength(a1) v !isBuggy(a1)

// 0.136111 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.212283 !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 0.639058 avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFileChurn(s) => isBuggy(s)
0.639058 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
// 0.395231 highFrequencyOfMerges(s) ^ avgFractalValue(s) => isBuggy(s)
0.395231 !highFrequencyOfMerges(a1) v !avgFractalValue(a1) v isBuggy(a1)

Juk:

// -0.280428 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
0 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.408379 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.408379 !highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -4.81344 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.978803 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0.502426 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0.170926 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)

// 0.506087 avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.506087 !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.34513 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
1.34513 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0 !highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0 !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.188029 avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.188029 !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.782636 avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
0.782636 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.32775 highFailureIntensity(s) => isBuggy(s)
1.32775 !highFailureIntensity(a1) v isBuggy(a1)

// 2.27077 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.958591 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.571288 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.16965 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.170504 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.963883 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

0.485399  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.268012  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.564951  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.24254   !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.703823  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.119856  !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
0.583967  !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.840181  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.840181  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.66957   highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.07652   !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.593049  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

K3b:

// 3.90631  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^ highOverallStrength(s) => isBuggy(s)
1.08974   !highFrequencyOfCommits(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
2.81658   !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 2.79232  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
2.97179   !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.01418  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
            (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0       !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
                 isBuggy(a1)
0       !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
                 isBuggy(a1)
0       !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
                 isBuggy(a1)
0.974649  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
                 isBuggy(a1)
1.27711   !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
                 isBuggy(a1)

// 0.252551  avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.252551  !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.29345   highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.29345   !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 2.15741  highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
2.15741   !highFailureOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 0.591382 !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.591382 highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v highFrequencyOfMerges(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// -11.4076 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.666419 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !isBuggy(a1)

// 5.13203 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
1.612 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v !isBuggy(a1)
3.52003 !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v !isBuggy(a1)

// 2.01239 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.82856 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)
0.183831 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)

// 2.58517 avgFrequencyOfMerges(s) ^ (highFileChurn(s) v avgFileChurn(s)) ^ avgFractalValue(s) => isBuggy(s)
0.830588 !highFileChurn(a1) v !avgFrequencyOfMerges(a1) v !avgFractalValue(a1) v !isBuggy(a1)
1.75458 !avgFileChurn(a1) v !avgFrequencyOfMerges(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 1.72064 highFrequencyOfMerges(s) ^ avgFractalValue(s) => isBuggy(s)
1.72064 !highFrequencyOfMerges(a1) v !avgFractalValue(a1) v !isBuggy(a1)

Kate:

// -0.451549 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^ highOverallStrength(s) => isBuggy(s)
0.816189 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)

// 0 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v !isBuggy(a1)
0 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v !isBuggy(a1)

// 1.03715 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
1.03715 !highFileChurn(a1) v !highFailureIntensity(a1) v !isBuggy(a1)

// 0.736469 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.816189 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)
0 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)
0 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)
0.237092 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)
Appendix B: Expanded Rules for Each Project

0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.459733  avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.459733  !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.0468572  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.0468572  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.887014  !highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s) => !isBuggy(s)
0.887014  highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v highFailureIntensity(a1) v isBuggy(a1)

// 0.851052  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) => isBuggy(s)
0.851052  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.58386  highFailureIntensity(s) => isBuggy(s)
0.58386  !highFailureIntensity(a1) v isBuggy(a1)

// 3.4144  (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
2.07328  !lowFileChurn(a1) v !highOverallStrength(a1) v !avgOverallStrength(a1) v !highFrequencyOfCommits(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)
2.08476  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
0.00775494  !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
0.935303  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgOverallStrength(a1) v !isBuggy(a1)

// 0.593907  highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.593907  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 2.53422  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
1.37582  !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
1.1584  !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 3.42054  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
1.02224  !avgFileChurn(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
0.805689  !highFileChurn(a1) v !highOverallStrength(a1) v !isBuggy(a1)
1.89295  !highFileChurn(a1) v !avgOverallStrength(a1) v !isBuggy(a1)

// 0.366645  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.342547  !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)
0.0240975  !avgOverallStrength(a1) v !highFractalValue(a1) v !isBuggy(a1)

// 0.436796  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.436796  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v !isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 1.2919 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.941263 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.350633 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 2.11404 highOverallStrength(s) ^ highFractalValue(s) => isBuggy(s)
2.11404 !highOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.594113 avgFrequencyOfMerges(s) ^ avgFileChurn(s) => isBuggy(s)
0.594113 !avgFileChurn(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 0.941263 highDistinctAuthors(s) ^ highFrequencyOfCommits(s) => isBuggy(s)
0.941263 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

Kdelibs:

// 0.465065 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0.465065 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -0.263349 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
0 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.511909 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.511909 !highFailureIntensity(a1) v !highFileChurn(a1) v isBuggy(a1)

// 0.941196 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s) ^ (highFileChurn(s) v avgFileChurn(s))) => isBuggy(s)
0.465065 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0 !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0.476131 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)

// 0.838531 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.838531 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.220232 highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.220232 !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 0.415208  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
0.415208  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 1.79473  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
1.95258  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.42843  highFailureIntensity(s) => isBuggy(s)
1.42843  !highFailureIntensity(a1) v isBuggy(a1)

// -1.16392  (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0       !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0       !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.694326  highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.694326  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.99523  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.623641  !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
1.37159  !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 1.45336  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
1.43251  !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.247755  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
0       !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.42025  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
1.50613  !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 6.172   highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
2.78923  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
3.38277  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// -0.085882  avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.085882  !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

Kio-extras:

// 0       (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0       !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0       !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.286238  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
0       !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

0.286238  \text{!avgFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

// 0  \text{(highFrequencyOfCommits(s)} \lor \text{avgFrequencyOfCommits(s))} \lor \text{!highFailureIntensity(s)} \lor
\text{(highOverallStrength(s)} \lor \text{avgOverallStrength(s))} \lor \text{!highFileChurn(s)} \lor \text{avgFileChurn(s))} \Rightarrow \text{isBuggy(s)}
0  \text{!highFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}
0  \text{!highFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!avgOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}
0  \text{!highFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}
0  \text{!avgFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}
0  \text{!avgFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}
0  \text{!avgFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!avgOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}

// 2.08234  \text{avgFileChurn(s)} \lor \text{!highFailureIntensity(s)} \Rightarrow \text{isBuggy(s)}
2.08234  \text{!avgFileChurn(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}

// 0.709059 \text{highFailureIntensity(s)} \lor \text{!highFrequencyOfMerges(s)} \Rightarrow \text{isBuggy(s)}
0.709059  \text{!highFrequencyOfMerges(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}

// 0.252581 \text{highFrequencyOfCommits(s)} \lor \text{!highFileChurn(s)} \Rightarrow \text{isBuggy(s)}
0.252581  \text{!highFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{isBuggy(a1)}

// 1.22129  \text{!highFailureIntensity(s)} \lor \text{!highFrequencyOfMerges(s)} \lor \text{!highOverallStrength(s)} \lor \text{!highFrequencyOfCommits(s)} \lor \text{!highFileChurn(s)} \lor \text{!highDistinctAuthors(s)} \Rightarrow \text{!isBuggy(s)}
1.22129  \text{!highFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{!highFailureIntensity(a1)} \lor \text{!highDistinctAuthors(a1)} \lor \text{isBuggy(a1)}

// 1.27715  \text{avgFileChurn(s)} \lor \text{!avgFractalValue(s)} \lor \text{!highFractalValue(s)} \lor \text{!highFrequencyOfCommits(s)} \Rightarrow \text{isBuggy(s)}
1.27715  \text{!avgFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!avgFractalValue(a1)} \lor \text{isBuggy(a1)}
0  \text{!avgFileChurn(a1)} \lor \text{!highFrequencyOfCommits(a1)} \lor \text{!avgFractalValue(a1)} \lor \text{isBuggy(a1)}

// 0.499177 \text{!highFailureIntensity(s)} \Rightarrow \text{isBuggy(s)}
0.499177  \text{!highFailureIntensity(a1)} \lor \text{isBuggy(a1)}

// -3.17845 \text{(lowFileChurn(s)} \lor \text{avgFileChurn(s))} \lor \text{!highOverallStrength(s)} \lor \text{avgOverallStrength(s))} \lor \text{!highFrequencyOfCommits(s)} \lor \text{avgFrequencyOfCommits(s))} \Rightarrow \text{isBuggy(s)}
1.05547  \text{!lowFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{isBuggy(a1)}
0.318215  \text{!avgFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!highOverallStrength(a1)} \lor \text{isBuggy(a1)}
0.226882  \text{!lowFileChurn(a1)} \lor \text{!avgFrequencyOfCommits(a1)} \lor \text{!avgOverallStrength(a1)} \lor \text{isBuggy(a1)}

// 0.60068 \text{avgFrequencyOfMerges(s)} \lor \text{!highOverallStrength(s)} \lor \text{avgOverallStrength(s))} \Rightarrow \text{isBuggy(s)}
0.60068  \text{!highOverallStrength(a1)} \lor \text{!avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}
0.086846  \text{!avgOverallStrength(a1)} \lor \text{!avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}
0.513834  \text{!avgOverallStrength(a1)} \lor \text{!avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}
Appendix B: Expanded Rules for Each Project

// 0.79774  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.373272 !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
2.14486  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.15016  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.15016  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.48915  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.03379  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.455359  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.714031  highFrequencyOfMerges(s) ^ highFractalValue(s) => isBuggy(s)
0.714031  !highFrequencyOfMerges(a1) v !highFractalValue(a1) v isBuggy(a1)

Kmix:

// 0.671213  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0.145114  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.526099  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.16259  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
1.16259  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.291033  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.291033  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.21253  highFrequencyOfCommits(s) ^ highFractalValue(s) => isBuggy(s)
1.21253  !highFractalValue(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.72903  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
0.72903  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.08621  highFailureIntensity(s) => isBuggy(s)
1.08621  !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// -2.06218 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.148439 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.363438 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// -0.352026 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.175207 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 2.52852 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.177173 !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.689303 !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.314351 !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.34769 !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.137814 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.163131 !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.369395 avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.369395 !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.166889 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.173943 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

Kompare:

// 0 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0 !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -0.481142 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
0.147966 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.670625 highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
0.670625 !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.45536 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
1.45536 !highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

0       !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.973662 avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.973662 !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 2.68713 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
2.68713 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.615984 highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.615984 !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.741161 avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.741161 !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.842341 avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) => isBuggy(s)
0.842341 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0       !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.4795 highFailureIntensity(s) => isBuggy(s)
1.4795 !highFailureIntensity(a1) v isBuggy(a1)

// 2.06935 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.56366 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.27756 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.396539 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.677477 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.358387 highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.358387 !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.09763 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.165858 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
0.931773 !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// -0.532101 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
0.803853 !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.0421309 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.0421309 !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
0       !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.643609 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
Appendix B: Expanded Rules for Each Project

0.264825  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.378784  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

Konsol

// 0.566497  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
  highOverallStrength(s) => isBuggy(s)
1.16753  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
  highFrequencyOfMerges(s) => isBuggy(s)
0  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
0  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.990857  highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
0.990857  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 4.35198  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
  (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
1.43448  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
  !highFailureIntensity(a1) v isBuggy(a1)
0.577444  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v
  !highFailureIntensity(a1) v isBuggy(a1)
0  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
  isBuggy(a1)
1.08252  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
  isBuggy(a1)
0.819918  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
  !highFailureIntensity(a1) v isBuggy(a1)
0.548062  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
  !highFailureIntensity(a1) v isBuggy(a1)

// 0.206648  avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.206648  !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.162902  highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.162902  !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.37489  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
1.37489  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.71629  !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v
  highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.71629  highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v
  highFrequencyOfMerges(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// 3.17286  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
  isBuggy(s)
0.321217  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
2.85164  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 0.30349 highFailureIntensity(s) => isBuggy(s)
0.30349 !highFailureIntensity(a1) v isBuggy(a1)

// -0.304915 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.667004 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
2.15706 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.56567 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.314556 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.706707 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.93568 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 1.89135 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.640411 !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)
1.25094 !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.09732 avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.09732 !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// -0.456689 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.172586 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.40066 highFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.40066 !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.227302 avgFrequencyOfMerges(s) ^ avgFractalValue(s) => isBuggy(s)
0.227302 !avgFrequencyOfMerges(a1) v !avgFractalValue(a1) v isBuggy(a1)

Konversation:

// -0.74931 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0.74931 !highOverallStrength(a1) v !highFailureIntensity(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 2.09137 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
2.09137 !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.468615 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.468615 !highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -1.44405 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
-1.44405 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.468615 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.468615 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

156
Appendix B: Expanded Rules for Each Project

0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.53227 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0  highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0  !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.0957727 avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.0957727 !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.815797 !(highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.815797 highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// 1.91874 avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) => isBuggy(s)
1.91874 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// -1.76583 (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.670391 !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.0359601 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.606846 !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// -1.27514 avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.0630257 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 0.278192 (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
1.34865 !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.368704 !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.783179 avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.783179 !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.807704 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.807704 !highFrequencyOfCommits(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)
0  !highDistinctAuthors(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.95248 highFrequencyOfCommits(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.95248 !highFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 2.69877 highOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
2.69877 !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
0.990231 !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
1.70854 !highOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

Ktorrent:

// 0  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.99199  highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
1.99199  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.99199  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)

// 1.82397  highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
1.82397  !highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 1.74308  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
1.74308  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.43885  highFailureIntensity(s) => isBuggy(s)
1.43885  !highFailureIntensity(a1) v isBuggy(a1)

// 0.338458  (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.74308  !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 2.84356  highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
2.84356  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 6.00906  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
2.68919  !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
3.31987  !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 4.37761  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
Appendix B: Expanded Rules for Each Project

1.92904  !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)  
0.0105727 :avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)  
2.53648 :highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)  

// 3.36009  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)  
0.546872 :avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)  
2.81322 :avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)  

// 1.53985  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)  
1.53985  :avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)  

// -0.157582  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)  
0  :avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)  

Lokalize:  

// 0.0830474  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^ 
  highOverallStrength(s) => isBuggy(s)  
0.299688  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  

// 0.324465  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^  
  highFrequencyOfMerges(s) => isBuggy(s)  
0  :avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)  
0.324465  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)  

// -0.839587  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^  
  (highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)  
0.083209  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  
0.335324  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  
1.80721  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  

// 1.20665  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)  
1.20665  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)  

// 0.334483  avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)  
0.334483  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)  

// 1.11522  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)  
1.11522  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)  

// -0.526814  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>  
  isBuggy(s)  
0.602533  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)  

// 1.44905  highFailureIntensity(s) => isBuggy(s)  
1.44905  !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// -2.7428  (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.471167  !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.59456  !lowFileChurn(a1) v highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.324523  !lowFileChurn(a1) v avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.347288  highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.347288  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// -0.146304  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.0312916  !avgFrequencyOfMerges(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.07763  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.503382  !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.348063  !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.483972  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// -0.101138  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.221483  !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.242563  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.242563  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.114167  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) =>
isBuggy(s)
0.201756  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.201756  highDistinctAuthors(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.201756  !highDistinctAuthors(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 0.129499  highOverallStrength(s) ^ highFractalValue(s) => isBuggy(s)
0.129499  !highOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.948491  avgFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.948491  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 0.85703  avgFrequencyOfCommits(s) ^ highFractalValue(s) => isBuggy(s)
0.85703  !avgFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.0312916  avgOverallStrength(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.0312916  !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

Plasma-nm:

// 0.926552  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0.347734  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)
0.578818  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.0611747  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
Appendix B: Expanded Rules for Each Project

1.457  \!
highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.40209 highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
1.40209  \!
highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.368666 highFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.368666  \!
highFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// -2.20939 (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0  \!
highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0.77287  \!
highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0.347734  \!
avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)
0.532035  \!
avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v
!highFailureIntensity(a1) v isBuggy(a1)

// 0.564204 highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.564204  \!
avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.659032 highFailureIntensity(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.659032  \!
highFrequencyOfMerges(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.857671 highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
0.857671  \!
highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.179599 \!
highFailureIntensity(s) v highFrequencyOfMerges(s) v highOverallStrength(s) v
highFrequencyOfCommits(s) v highFileChurn(s) v highDistinctAuthors(s)) => !isBuggy(s)
0.179599  \!
highFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v
highFrequencyOfMerges(a1) v highFailureIntensity(a1) v highDistinctAuthors(a1) v !isBuggy(a1)

// 0.287726 avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
0.287726  \!
avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)
0  \!
avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)

// 1.0708 highFailureIntensity(s) => isBuggy(s)
1.0708  \!
highFailureIntensity(a1) v isBuggy(a1)

// 4.91573 (lowFileChurn(s v avgFileChurn(s)) ^ (highOverallStrength(s v avgOverallStrength(s))) ^
(highFrequencyOfCommits(s v avgFrequencyOfCommits(s)) => isBuggy(s)
1.17422  \!
lowFileChurn(a1) v highFrequencyOfCommits(a1) v highOverallStrength(a1) v isBuggy(a1)
2.48746  \!
avgFileChurn(a1) v highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.640843  \!
lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.895427  \!
lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.408023  \!
avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.470804  \!
lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.311065 highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
0.311065  \!
highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 1.9083  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
1.08185  !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.132095  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.929221  !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 0.383252  avgOverallStrength(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)
0.397501  !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.219545  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.219545  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.782752  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.500918  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.281834  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.47951  avgFrequencyOfCommits(s) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.523193  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)
0.956316  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

Solid:

// -0.0976424  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)

// 0.105505  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highOverallStrength(s) ^
highFrequencyOfMerges(s) => isBuggy(s)
1.24902  !highFrequencyOfMerges(a1) v !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.768832  highFrequencyOfMerges(s) ^ highFileChurn(s) => isBuggy(s)
0.768832  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// -0.097428  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
(highOverallStrength(s) v avgOverallStrength(s)) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)
0  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v
isBuggy(a1)

// 0.053933  avgFileChurn(s) ^ highFailureIntensity(s) => isBuggy(s)
0.053933  !avgFileChurn(a1) v !highFailureIntensity(a1) v isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 1.11444  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
1.11444  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.354  highFrequencyOfCommits(s) ^ highFileChurn(s) => isBuggy(s)
0.354  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// -1.5213  avgFileChurn(s) ^ (avgFractalValue(s) v highFractalValue(s)) ^ highFrequencyOfCommits(s) =>
isBuggy(s)
0.0157551  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.22598  highFailureIntensity(s) => isBuggy(s)
1.22598  !highFailureIntensity(a1) v isBuggy(a1)

// 0.262748  (lowFileChurn(s) v avgFileChurn(s)) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^
(highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.922605  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.898698  !lowFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
1.37582  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
0.755795  !lowFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.17884  highFrequencyOfMerges(s) ^ highOverallStrength(s) => isBuggy(s)
1.17884  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 2.47299  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
1.42401  !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
1.04898  !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 0.0249433  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
1.3687  !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.0709752  !avgFileChurn(a1) v !avgOverallStrength(a1) v !isBuggy(a1)

// 1.03407  avgFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.03407  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 4.39221  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) =>
isBuggy(s)
2.334  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
2.05821  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)

// 0.958479  highFrequencyOfCommits(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.958479  !highFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

Systemsettings:

// -0.192764  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highFailureIntensity(s) ^
highOverallStrength(s) => isBuggy(s)
0  !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v !highFailureIntensity(a1) v !isBuggy(a1)
Appendix B: Expanded Rules for Each Project

// 1.39875 \text{highFrequencyOfCommits(s)} \lor \text{avgFrequencyOfCommits(s)} \lor \text{highOverallStrength(s)}\lor 
\text{highFrequencyOfMerges(s)} \Rightarrow \text{isBuggy(s)}

0.649018 \!\text{highFrequencyOfCommits(a1)} \lor \!\text{highOverallStrength(a1)} \lor \!\text{highFrequencyOfMerges(a1)} \lor 
\text{isBuggy(a1)}

0.749735 \!\text{avgFrequencyOfCommits(a1)} \lor \!\text{highOverallStrength(a1)} \lor \!\text{highFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

// 3.59335 \text{highFrequencyOfMerges(s)} \lor \text{highFileChurn(s)} \Rightarrow \text{isBuggy(s)}

3.59335 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

// 0 \text{highFrequencyOfCommits(s)} \lor \text{avgFrequencyOfCommits(s)} \lor \text{highFailureIntensity(s)}\lor 
\text{highOverallStrength(s)} \lor \text{avgOverallStrength(s)} \lor \text{highFileChurn(s)} \lor \text{avgFileChurn(s)} \Rightarrow \text{isBuggy(s)}

0 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{highOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{avgOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{avgOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{avgOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{avgFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{highOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{avgFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{avgOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

0 \!\text{avgFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{avgOverallStrength(a1)} \lor \!\text{highFailureIntensity(a1)} \lor 
\text{isBuggy(a1)}

// 0.128286 \text{highFailureIntensity(s)} \lor \text{highFrequencyOfMerges(s)} \Rightarrow \text{isBuggy(s)}

0.128286 \!\text{highFrequencyOfMerges(a1)} \lor \!\text{highFailureIntensity(a1)} \lor \text{isBuggy(a1)}

// 1.78774 \text{highFrequencyOfCommits(s)} \lor \text{highFileChurn(s)} \Rightarrow \text{isBuggy(s)}

1.78774 \!\text{highFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \text{isBuggy(a1)}

// -0.170059 \text{avgFileChurn(s)} \lor \text{avgFractalValue(s)} \lor \text{highFractalValue(s)} \lor \text{highFrequencyOfCommits(s)} \Rightarrow \text{isBuggy(s)}

-0.170059 \!\text{avgFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{highFractalValue(a1)} \lor \text{isBuggy(a1)}

// -0.393653 \text{lowFileChurn(s)} \lor \text{avgFileChurn(s)} \lor \text{highOverallStrength(s)} \lor \text{avgOverallStrength(s)} \lor 
\text{highFrequencyOfCommits(s)} \lor \text{avgFrequencyOfCommits(s)} \Rightarrow \text{isBuggy(s)}

1.39194 \!\text{lowFileChurn(a1)} \lor \!\text{highFrequencyOfCommits(a1)} \lor \!\text{highOverallStrength(a1)} \lor \text{isBuggy(a1)}

// 3.64713 \text{highFrequencyOfMerges(s)} \lor \text{highOverallStrength(s)} \Rightarrow \text{isBuggy(s)}

3.64713 \!\text{highOverallStrength(a1)} \lor \!\text{highFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

// 8.24606 \text{avgFrequencyOfMerges(s)} \lor \text{highOverallStrength(s)} \lor \text{avgOverallStrength(s)} \Rightarrow \text{isBuggy(s)}

8.24606 \!\text{highOverallStrength(a1)} \lor \!\text{avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

3.78481 \!\text{highOverallStrength(a1)} \lor \!\text{avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}

4.46125 \!\text{avgOverallStrength(a1)} \lor \!\text{avgFrequencyOfMerges(a1)} \lor \text{isBuggy(a1)}
Appendix B: Expanded Rules for Each Project

// -1.93904  (highOverallStrength(s) v avgOverallStrength(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.411204  !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 1.73346  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
1.73346  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 2.26979  highFractalValue(s) ^ avgFileChurn(s) => isBuggy(s)
2.26979  !avgFileChurn(a1) v !highFractalValue(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

C1: Fuzzy Based Fault Proneness Identification

**Akregator:**

RULE 1: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND distinctAuthors IS high THEN isSegmentBuggy IS maybe;
RULE 2: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS high AND overallStrength IS high AND fileChurn IS avg THEN isSegmentBuggy IS buggy;
RULE 4: IF (overallStrength IS high OR overallStrength IS avg) AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 5: IF frequencyOfCommits IS high AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfCommits IS avg AND overallStrength IS avg AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF frequencyOfCommits IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 8: IF frequencyOfCommits IS avg AND frequencyOfMerges IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 9: IF fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

**Ark:**

RULE 1: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 2: IF distinctAuthors IS high AND (frequencyOfCommits IS avg OR frequencyOfCommits IS high) THEN isSegmentBuggy IS buggy;
RULE 3: IF frequencyOfCommits IS avg AND (overallStrength IS high OR overallStrength IS avg) AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF frequencyOfCommits IS avg AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF frequencyOfMerges IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfMerges IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF fractalValue IS avg AND distinctAuthors IS avg THEN isSegmentBuggy IS maybe;
RULE 8: IF frequencyOfCommits IS avg AND fileChurn IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 9: IF frequencyOfMerges IS avg AND overallStrength IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 10: IF (frequencyOfCommits IS avg OR frequencyOfCommits IS high) AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 11: IF overallStrength IS avg AND (frequencyOfMerges IS avg OR frequencyOfMerges IS high) AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 12: IF frequencyOfCommits IS avg AND frequencyOfMerges IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
Appendix C: Customized Rules for Each Project

**Elisa:**

RULE 1 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfCommits IS avg AND (overallStrength IS avg OR overallStrength IS high) THEN isSegmentBuggy IS maybe;
RULE 3 : IF distinctAuthors IS high AND (frequencyOfCommits IS avg OR frequencyOfCommits IS high) THEN isSegmentBuggy IS buggy;
RULE 4 : IF frequencyOfCommits IS high AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS buggy;
RULE 5 : IF frequencyOfMerges IS high AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;
RULE 6 : IF overallStrength IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 7 : IF (fileChurn IS avg OR fileChurn IS high) AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 8 : IF overallStrength IS high AND (fractalValue IS high OR fractalValue IS avg) THEN isSegmentBuggy IS maybe;
RULE 9 : IF (frequencyOfCommits IS low OR frequencyOfCommits IS avg) AND fractalValue IS avg AND distinctAuthors IS avg THEN isSegmentBuggy IS notBuggy;
RULE 10 : IF frequencyOfCommits IS low AND overallStrength IS low AND frequencyOfMerges IS low AND fileChurn IS low AND fractalValue IS low AND distinctAuthors IS low THEN isSegmentBuggy IS notBuggy;

**Gwenview:**

RULE 1 : IF distinctAuthors IS high AND (frequencyOfCommits IS avg OR frequencyOfCommits IS high) THEN isSegmentBuggy IS buggy;
RULE 2 : IF overallStrength IS high AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4 : IF overallStrength IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 5 : IF overallStrength IS high AND frequencyOfCommits IS high AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS maybe;
RULE 6 : IF distinctAuthors IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfCommits IS avg AND overallStrength IS avg AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 8 : IF overallStrength IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 9 : IF frequencyOfMerges IS high AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS buggy;
RULE 10 : IF frequencyOfCommits IS avg AND overallStrength IS high THEN isSegmentBuggy IS maybe;
RULE 11 : IF frequencyOfCommits IS high AND overallStrength IS avg AND fileChurn IS high THEN isSegmentBuggy IS maybe;
RULE 12 : IF frequencyOfCommits IS high AND (fileChurn IS high OR fileChurn IS avg) THEN isSegmentBuggy IS maybe;
RULE 13 : IF frequencyOfMerges IS high AND (fractalValue IS avg OR fractalValue IS high) THEN isSegmentBuggy IS maybe;
Appendix C: Customized Rules for Each Project

**Juk:**

RULE 1: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2: IF (failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high) AND (overallStrength IS high OR overallStrength IS avg OR fractalValue IS avg) THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF fileChurn IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF fileChurn IS high AND (fractalValue IS high OR fractalValue IS avg) THEN isSegmentBuggy IS buggy;
RULE 6: IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF frequencyOfCommits IS avg AND overallStrength IS high THEN isSegmentBuggy IS maybe;

**K3b:**

RULE 1: IF distinctAuthors IS high AND frequencyOfCommits IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF frequencyOfMerges IS high AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS buggy;
RULE 3: IF frequencyOfMerges IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF frequencyOfMerges IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfMerges IS avg AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS maybe;

**Kate:**

RULE 1: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2: IF frequencyOfMerges IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS avg AND overallStrength IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF (failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high) AND overallStrength IS high AND fractalValue IS avg THEN isSegmentBuggy IS buggy;
RULE 5: IF distinctAuthors IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 6: IF frequencyOfCommits IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 7: IF fileChurn IS high AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS buggy;
RULE 8: IF fileChurn IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 9: IF overallStrength IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 10: IF frequencyOfCommits IS avg AND frequencyOfMerges IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 11: IF (overallStrength IS avg OR overallStrength IS high) AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
Appendix C: Customized Rules for Each Project

**Kdelibs:**

RULE 1: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2: IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

**Kio-extras:**

RULE 1: IF frequencyOfCommits IS high AND overallStrength IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 2: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 3: IF frequencyOfMerges IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF overallStrength IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfMerges IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 8: IF overallStrength IS high AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS buggy;

**Kmix:**

RULE 1: IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2: IF (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND (fileChurn IS avg OR fileChurn IS high) THEN isSegmentBuggy IS buggy;
RULE 3: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 4: IF frequencyOfCommits IS avg AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 5: IF overallStrength IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6: IF fileChurn IS high AND (overallStrength IS high OR overallStrength IS avg) THEN isSegmentBuggy IS buggy;

**Kompare:**

RULE 1: IF frequencyOfCommits IS high AND overallStrength IS high AND fileChurn IS avg THEN isSegmentBuggy IS buggy;
RULE 2: IF (overallStrength IS high OR overallStrength IS avg) AND fileChurn IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 3: IF frequencyOfCommits IS high AND distinctAuthors IS high THEN isSegmentBuggy IS buggy;
RULE 4: IF overallStrength IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 5: IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6: IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
Appendix C: Customized Rules for Each Project

**Konsole:**

RULE 1 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 2 : IF frequencyOfCommits IS high AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 3 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 4 : IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 5 : IF fileChurn IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 6 : IF ( overallStrength IS high) AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfCommits IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 8 : IF overallStrength IS avg AND fileChurn IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 9 : IF frequencyOfCommits IS avg AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10 : IF fileChurn IS avg AND overallStrength IS avg THEN isSegmentBuggy IS maybe;

**Konversation:**

RULE 1 : IF overallStrength IS avg AND frequencyOfMerges IS high THEN isSegmentBuggy IS buggy;
RULE 2 : IF overallStrength IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 3 : IF frequencyOfCommits IS high AND (fileChurn IS avg OR frequencyOfMerges IS avg) THEN isSegmentBuggy IS maybe;
RULE 4 : IF frequencyOfCommits IS high AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 5 : IF overallStrength IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6 : IF frequencyOfMerges IS high AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF fileChurn IS high AND overallStrength IS avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 8 : IF overallStrength IS high AND fractalValue IS high THEN isSegmentBuggy IS buggy;
RULE 9 : IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 10 : IF fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 11 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;

**Ktorrent:**

RULE 1 : IF fileChurn IS high AND overallStrength IS avg THEN isSegmentBuggy IS buggy;
RULE 2 : IF fractalValue IS high AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 3 : IF frequencyOfCommits IS avg AND overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 4 : IF overallStrength IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 5 : IF frequencyOfCommits IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6 : IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
Appendix C: Customized Rules for Each Project

**Lokalize:**

RULE 1 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS maybe;
RULE 3 : IF frequencyOfCommits IS avg AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 4 : IF frequencyOfCommits IS high AND overallStrength IS high THEN isSegmentBuggy IS buggy;
RULE 5 : IF frequencyOfMerges IS avg AND overallStrength IS avg THEN isSegmentBuggy IS maybe;
RULE 6 : IF overallStrength IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF frequencyOfCommits IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 8 : IF fileChurn IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS maybe;
RULE 9 : IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 10 : IF overallStrength IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 11 : IF overallStrength IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 12 : IF fileChurn IS avg AND frequencyOfCommits IS avg THEN isSegmentBuggy IS maybe;
RULE 13 : IF frequencyOfCommits IS avg AND overallStrength IS high AND frequencyOfMerges IS avg THEN isSegmentBuggy IS maybe;
RULE 14 : IF overallStrength IS high AND frequencyOfMerges IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 15 : IF frequencyOfCommits IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 16 : IF (overallStrength IS high OR overallStrength IS avg) AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 17 : IF frequencyOfCommits IS avg AND fractalValue IS high THEN isSegmentBuggy IS maybe;

**Plasma-nm:**

RULE 1 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfCommits IS high AND fileChurn IS high THEN isSegmentBuggy IS buggy;
RULE 3 : IF failureIntensity1stSegment IS high OR failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high THEN isSegmentBuggy IS buggy;
RULE 4 : IF (frequencyOfCommits IS high OR fileChurn IS high) AND overallStrength IS high THEN isSegmentBuggy IS maybe;
RULE 5 : IF overallStrength IS avg AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 6 : IF frequencyOfCommits IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 7 : IF fileChurn IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 8 : IF overallStrength IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

**Solid:**

RULE 1 : IF distinctAuthors IS high AND (frequencyOfCommits IS high OR frequencyOfCommits IS avg) AND fractalValue IS avg THEN isSegmentBuggy IS buggy;
RULE 2 : IF frequencyOfCommits IS high AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 3 : IF frequencyOfCommits IS high AND fileChurn IS high AND distinctAuthors IS high THEN isSegmentBuggy IS buggy;
Appendix C: Customized Rules for Each Project

RULE 4: IF NOT(frequencyOfCommits IS high OR frequencyOfCommits IS avg OR frequencyOfMerges IS high
OR fileChurn IS high OR overallStrength IS high OR failureIntensity1stSegment IS high OR
failureIntensity2ndSegment IS high OR failureIntensity3rdSegment IS high OR distinctAuthors IS high) THEN
isSegmentBuggy IS notBuggy;
RULE 5: IF frequencyOfCommits IS avg AND overallStrength IS avg AND fractalValue IS avg THEN
isSegmentBuggy IS maybe;
RULE 6: IF frequencyOfCommits IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;
RULE 7: IF overallStrength IS high AND fileChurn IS avg AND fractalValue IS avg THEN isSegmentBuggy IS
maybe;

Systemsettings:

RULE 1: IF frequencyOfCommits IS high AND fileChurn IS high AND distinctAuthors IS high THEN
isSegmentBuggy IS buggy;
RULE 2: IF frequencyOfCommits IS high AND overallStrength IS high AND fractalValue IS avg THEN
isSegmentBuggy IS buggy;
RULE 3: IF overallStrength IS avg AND fileChurn IS avg THEN isSegmentBuggy IS maybe;
RULE 4: IF fileChurn IS avg AND fractalValue IS high THEN isSegmentBuggy IS maybe;
RULE 5: IF overallStrength IS high AND fractalValue IS avg THEN isSegmentBuggy IS maybe;

C2: MLN Based Fault Proneness Identification (With Weights)

Akregator:

// 1.55903  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ highDistinctAuthors(s) => isBuggy(s)
0.896887  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.662146  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.65203  highFailureIntensity(s) => isBuggy(s)
1.65203  !highFailureIntensity(a1) v isBuggy(a1)

// 1.33505  highFrequencyOfCommits(s) ^ highOverallStrength(s) ^ avgFileChurn(s) => isBuggy(s)
1.33505  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.860326  (highOverallStrength(s) v avgOverallStrength(s)) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.55709  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
0.303236  !avgOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 2.14751  highFrequencyOfCommits(s) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.844072  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)
1.30344  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 0.599916  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.599916  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 2.67868  highFrequencyOfCommits(s) ^ highFractalValue(s) => isBuggy(s)
2.67868  !highFractalValue(a1) v isBuggy(a1)

// 2.10118  avgFrequencyOfCommits(s) ^ avgFrequencyOfMerges(s) ^ avgFileChurn(s) => isBuggy(s)
2.10118  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

// 0.0104376  avgFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
0.0104376  !avgFileChurn(a1) v !avgFractalValue(a1) v isBuggy(a1)

Ark:

// 1.24951  highFailureIntensity(s) => isBuggy(s)
1.24951  !highFailureIntensity(a1) v isBuggy(a1)

// 3.40324  highDistinctAuthors(s) ^ (avgFrequencyOfCommits(s) v highFrequencyOfCommits(s)) => isBuggy(s)
1.45057  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
1.95266  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.45554  avgFrequencyOfCommits(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) ^ avgFileChurn(s) => isBuggy(s)
1.67664  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFileChurn(a1) v isBuggy(a1)

// 0.0211412  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.0211412  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.436281  avgFractalValue(s) ^ avgDistinctAuthors(s) => isBuggy(s)
0.436281  !avgFractalValue(a1) v !avgDistinctAuthors(a1) v isBuggy(a1)

// 0.238611  avgFrequencyOfCommits(s) ^ highFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
0.238611  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.31569  (avgFrequencyOfCommits(s) v highFrequencyOfCommits(s)) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.34718  !avgFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
0.968514  !highFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

Elisa:

// 1.48504  highFailureIntensity(s) => isBuggy(s)
1.48504  !highFailureIntensity(a1) v isBuggy(a1)

// 1.29995  avgFrequencyOfCommits(s) ^ (avgOverallStrength(s) v highOverallStrength(s)) => isBuggy(s)
0.756174  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.543781  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.4377  highDistinctAuthors(s) ^ (avgFrequencyOfCommits(s) v highFrequencyOfCommits(s)) => isBuggy(s)
0.669545  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.768156  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.9891  highFrequencyOfCommits(s) ^ (avgOverallStrength(s) v highOverallStrength(s)) => isBuggy(s)
1.08763  !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highOverallStrength(a1) v isBuggy(a1)
0.901476  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.48439  highFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
0.689081  !highOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
0.795312  !avgOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

// 1.42846 highOverallStrength(s) ^ highFileChurn(s) => isBuggy(s)
1.42846 !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.550237 (avgFileChurn(s) v highFileChurn(s)) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s))
=> isBuggy(s)
0.858409 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)
0.0879561 !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)
0.0412019 !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 2.71326 highOverallStrength(s) ^ (highFractalValue(s) v avgFractalValue(s)) => isBuggy(s)
2.13131 !highOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)
0.581955 !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.40638 (lowFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ avgFractalValue(s) ^
avgDistinctAuthors(s) => !isBuggy(s)
1.28462 !lowFrequencyOfCommits(a1) v !avgFractalValue(a1) v !avgDistinctAuthors(a1) v !isBuggy(a1)
0.121762 !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v !avgDistinctAuthors(a1) v !isBuggy(a1)

// 0.338696 lowFrequencyOfCommits(s) ^ lowOverallStrength(s) ^ lowFrequencyOfMerges(s) ^ lowFileChurn(s) ^
lowFractalValue(s) ^ lowDistinctAuthors(s) => !isBuggy(s)
0.338696 !lowFileChurn(a1) v !lowFrequencyOfCommits(a1) v !lowOverallStrength(a1) v !lowFrequencyOfMerges(a1) v !lowFractalValue(a1) v !lowDistinctAuthors(a1) v !isBuggy(a1)

Gwenview:

// 0.713572 highFailureIntensity(s) => isBuggy(s)
0.713572 !highFailureIntensity(a1) v isBuggy(a1)

// 0.471085 highOverallStrength(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.471085 !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 1.89571 highOverallStrength(s) ^ highFrequencyOfCommits(s) ^ (avgFileChurn(s) v highFileChurn(s)) =>
isBuggy(s)
0.532959 !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.36275 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.341041 highDistinctAuthors(s) ^ avgOverallStrength(s) => isBuggy(s)
0.341041 !avgOverallStrength(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.564555 avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.564555 !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.8456 highFrequencyOfMerges(s) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)
0.680705 !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)
1.16489 !avgFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.627577 avgFrequencyOfCommits(s) ^ highOverallStrength(s) => isBuggy(s)
0.627577 !avgFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 0.578838 highFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ highFileChurn(s) => isBuggy(s)
0.578838 !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

// 1.41745  highFrequencyOfCommits(s) ^ (highFileChurn(s) v avgFileChurn(s)) => isBuggy(s)  
0.71096  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)  
0.706494  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)  

// 0.790796  highFrequencyOfMerges(s) ^ (avgFractalValue(s) v highFractalValue(s)) => isBuggy(s)  
0.442605  !highFrequencyOfMerges(a1) v !avgFractalValue(a1) v isBuggy(a1)  
0.348191  !highFrequencyOfMerges(a1) v !highFractalValue(a1) v isBuggy(a1)  

// 0.790796  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)  
1.16138  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)  
0.597441  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)  

// 1.75882  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)  
1.75882  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)  

// 2.12871  highFailureIntensity(s) ^ (highOverallStrength(s) v avgOverallStrength(s) v avgFractalValue(s)) => isBuggy(s)  
0.3926  !highOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  
1.19885  !avgOverallStrength(a1) v !highFailureIntensity(a1) v isBuggy(a1)  
0.53726  !highFailureIntensity(a1) v !avgFractalValue(a1) v isBuggy(a1)  

// 1.73296  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)  
1.73296  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)  

// 2.12871  highFrequencyOfMerges(s) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)  
1.10837  !avgFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)  
0.281839  !highFileChurn(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)  

// 1.73296  highFrequencyOfMerges(s) ^ avgOverallStrength(s) => isBuggy(s)  
1.68811  !avgOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)  

// 4.96074  avgFrequencyOfMerges(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)  
1.86264  !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

Kdelibs:

// 5.43012 highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
2.9008 !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
2.52932 !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.897546 avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.897546 !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

Kio-extras:

// 1.66876  highFrequencyOfCommits(s) ^ highOverallStrength(s) ^ highFileChurn(s) => isBuggy(s)
1.66876  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 2.42853  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.96256  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
1.46588  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.89607  highFrequencyOfMerges(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
1.89607  !avgFrequencyOfCommits(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.881511  highOverallStrength(s) ^ avgFrequencyOfMerges(s) => isBuggy(s)
0.881511  !avgFrequencyOfMerges(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.10731  highFailureIntensity(s) => isBuggy(s)
1.10731  !highFailureIntensity(a1) v isBuggy(a1)

// 0.48069  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.48069  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 2.67701  highOverallStrength(s) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
0.843256  !avgFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)
1.83375  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

Kmix:

// 0.140587  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.0803921  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
0.0601944  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 3.45825  (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^ (avgFileChurn(s) v highFileChurn(s)) => isBuggy(s)
1.30342  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)
0.583957  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)
1.10542  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)
0.465442  !highFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 0.892979  highFailureIntensity(s) => isBuggy(s)
0.892979  !highFailureIntensity(a1) v isBuggy(a1)

// 0.568955  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.568955  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.633422  avgOverallStrength(s) ^ avgFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
0.633422  !avgFileChurn(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.03329  highFileChurn(s) ^ (highOverallStrength(s) v avgOverallStrength(s)) => isBuggy(s)
1.16275  !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
Appendix C: Customized Rules for Each Project

Kompare:

// 2.68455  highFrequencyOfCommits(s) ^ highOverallStrength(s) ^ avgFileChurn(s) => isBuggy(s)
2.68455  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !isBuggy(a1)

// 1.6377  (highOverallStrength(s) v avgOverallStrength(s)) ^ highFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
1.6377  !highFileChurn(a1) v !highOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 0.105811  highFrequencyOfCommits(s) ^ highDistinctAuthors(s) => isBuggy(s)
0.105811  !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)

// 1.18433  highOverallStrength(s) ^ highFractalValue(s) => isBuggy(s)
1.18433  !highOverallStrength(a1) v !highFractalValue(a1) v !isBuggy(a1)

// 0.491809  avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.491809  !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 1.18593  highFailureIntensity(s) => isBuggy(s)
1.18593  !highFailureIntensity(a1) v !isBuggy(a1)

Konsole:

// 1.04862  highFailureIntensity(s) => isBuggy(s)
1.04862  !highFailureIntensity(a1) v !isBuggy(a1)

// 0.764977  highFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.764977  !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 0.337818  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) => isBuggy(s)
0.337818  !avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v !isBuggy(a1)

// 1.84814  highFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.84814  !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 0.175253  avgOverallStrength(s) ^ highFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
0.175253  !highFileChurn(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 1.03491  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
1.03491  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v !isBuggy(a1)

// 0.224735  avgFileChurn(s) ^ avgOverallStrength(s) => isBuggy(s)
0.224735  !avgFileChurn(a1) v !avgOverallStrength(a1) v !isBuggy(a1)
Appendix C: Customized Rules for Each Project

Konversation:

// 0.522607  avgOverallStrength(s) ^ highFrequencyOfMerges(s) => isBuggy(s)
0.522607  !avgOverallStrength(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 1.18474  highOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
1.18474  !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.1726  highFrequencyOfCommits(s) ^ (avgFileChurn(s) v avgFrequencyOfMerges(s)) => isBuggy(s)
0.435965  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)
0.736636  !highFrequencyOfCommits(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)

// 1.04522  highFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
1.04522  !highFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.52059  avgOverallStrength(s) ^ avgFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
1.52059  !avgFileChurn(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.955925  highFrequencyOfMerges(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.955925  !avgFrequencyOfCommits(a1) v !highFrequencyOfMerges(a1) v isBuggy(a1)

// 0.136334  highFileChurn(s) ^ avgOverallStrength(s) ^ avgFrequencyOfCommits(s) => isBuggy(s)
0.136334  !highFileChurn(a1) v !avgOverallStrength(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 2.00396  highOverallStrength(s) ^ highFractalValue(s) => isBuggy(s)
2.00396  !highOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 0.536447  avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.536447  !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.0531928  highFailureIntensity(s) => isBuggy(s)
0.0531928  !highFailureIntensity(a1) v isBuggy(a1)

Ktorrent:

// 2.61944  highFileChurn(s) ^ avgOverallStrength(s) => isBuggy(s)
2.61944  !highFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)

// 3.17645  highFractalValue(s) ^ avgOverallStrength(s) => isBuggy(s)
3.17645  !avgOverallStrength(a1) v !highFractalValue(a1) v isBuggy(a1)

// 4.40551  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
4.40551  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 1.71701  highOverallStrength(s) ^ highFileChurn(s) => isBuggy(s)
1.71701  !highFileChurn(a1) v !highOverallStrength(a1) v isBuggy(a1)

// 1.26847  highFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
1.26847  !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.991156  avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
Appendix C: Customized Rules for Each Project

0.991156 \texttt{!avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)}

\textbf{Lokalize:}

\begin{itemize}
  \item // 0.118716 \texttt{highDistinctAuthors(s) \land (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) \rightarrow isBuggy(s)}
  \item // 0.0894595 \texttt{!highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)}
  \item // 0.0292568 \texttt{avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)}
  \item // 0.82748 \texttt{highFailureIntensity(s) \Rightarrow isBuggy(s)}
  \item 0.82748 \texttt{!highFailureIntensity(a1) v isBuggy(a1)}
  \item // 0.0438699 \texttt{avgFrequencyOfCommits(s) \land avgOverallStrength(s) \Rightarrow isBuggy(s)}
  \item 0.0438699 \texttt{!avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v isBuggy(a1)}
  \item // 0.147104 \texttt{avgFrequencyOfMerges(s) \land avgOverallStrength(s) \Rightarrow isBuggy(s)}
  \item 0.147104 \texttt{!avgOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)}
  \item // 1.00145 \texttt{avgOverallStrength(s) \land avgFileChurn(s) \land avgFractalValue(s) \Rightarrow isBuggy(s)}
  \item 1.00145 \texttt{!avgFileChurn(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)}
  \item // 0.809681 \texttt{highFileChurn(s) \land (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) \Rightarrow isBuggy(s)}
  \item 0.691508 \texttt{!highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)}
  \item 0.118172 \texttt{!highFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)}
  \item // 0.826952 \texttt{highFrequencyOfCommits(s) \land avgFileChurn(s) \Rightarrow isBuggy(s)}
  \item 0.826952 \texttt{!avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)}
  \item // 0.109677 \texttt{avgOverallStrength(s) \land avgFractalValue(s) \Rightarrow isBuggy(s)}
  \item 0.109677 \texttt{!avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)}
  \item // 0.506278 \texttt{avgFileChurn(s) \land avgFrequencyOfCommits(s) \Rightarrow isBuggy(s)}
  \item 0.506278 \texttt{!avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)}
  \item // 0.286433 \texttt{highOverallStrength(s) \land avgFrequencyOfMerges(s) \land avgFileChurn(s) \Rightarrow isBuggy(s)}
  \item 0.286433 \texttt{!avgFileChurn(a1) v !highOverallStrength(a1) v !avgFrequencyOfMerges(a1) v isBuggy(a1)}
  \item // 0.35378 \texttt{avgFrequencyOfCommits(s) \land avgFractalValue(s) \Rightarrow isBuggy(s)}
  \item 0.35378 \texttt{!avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)}
  \item // 0.861458 \texttt{avgFrequencyOfCommits(s) \land highFractalValue(s) \Rightarrow isBuggy(s)}
  \item 0.861458 \texttt{!avgFrequencyOfCommits(a1) v !highFractalValue(a1) v isBuggy(a1)}
\end{itemize}

\textbf{Plasma-nm:}

\begin{itemize}
  \item // 1.54023 \texttt{highDistinctAuthors(s) \land (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) \Rightarrow isBuggy(s)}
  \item 0.822446 \texttt{!highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)}
  \item 0.717781 \texttt{!avgFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)}
  \item // 0.348835 \texttt{highFrequencyOfCommits(s) \land highFileChurn(s) \Rightarrow isBuggy(s)}
  \item 0.348835 \texttt{!highFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)}
\end{itemize}
Appendix C: Customized Rules for Each Project

// 1.24154  highFailureIntensity(s) => isBuggy(s)
1.24154  !highFailureIntensity(a1) v isBuggy(a1)

// 0.968666  (highFrequencyOfCommits(s) v highFileChurn(s)) ^ highOverallStrength(s) => isBuggy(s)
1.49925  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v isBuggy(a1)
// 1.49842  avgOverallStrength(s) ^ avgFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
1.49842  !avgFileChurn(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.133181  avgFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
0.133181  !avgFileChurn(a1) v !avgFrequencyOfCommits(a1) v isBuggy(a1)

// 1.011111  highFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
1.011111  !highFileChurn(a1) v !avgFractalValue(a1) v isBuggy(a1)
// 0.291316  highOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.291316  !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

Solid:

// 3.83044  highDistinctAuthors(s) ^ (highFrequencyOfCommits(s) v avgFrequencyOfCommits(s)) ^
avgFractalValue(s) => isBuggy(s)
1.6898  !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v !highDistinctAuthors(a1) v isBuggy(a1)
2.14064  !avgFrequencyOfCommits(a1) v !avgFractalValue(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.60546  highFrequencyOfCommits(s) ^ avgFileChurn(s) => isBuggy(s)
1.60546  !avgFileChurn(a1) v !highFrequencyOfCommits(a1) v isBuggy(a1)

// 1.60788  highFrequencyOfCommits(s) ^ highFileChurn(s) ^ highDistinctAuthors(s) => isBuggy(s)
1.60788  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 0.645536  avgFrequencyOfCommits(s) ^ avgOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
0.645536  !avgFrequencyOfCommits(a1) v !avgOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.223698  highFrequencyOfCommits(s) ^ avgFractalValue(s) => isBuggy(s)
0.223698  !highFrequencyOfCommits(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 2.00205  highOverallStrength(s) ^ avgFileChurn(s) ^ avgFractalValue(s) => isBuggy(s)
2.00205  !avgFileChurn(a1) v !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

SystemSettings:

// 2.96813  highFrequencyOfCommits(s) ^ highFileChurn(s) ^ highDistinctAuthors(s) => isBuggy(s)
2.96813  !highFileChurn(a1) v !highFrequencyOfCommits(a1) v !highDistinctAuthors(a1) v isBuggy(a1)

// 1.28955  highFrequencyOfCommits(s) ^ highOverallStrength(s) ^ avgFractalValue(s) => isBuggy(s)
1.28955  !highFrequencyOfCommits(a1) v !highOverallStrength(a1) v !avgFractalValue(a1) v isBuggy(a1)

// 0.617782  avgOverallStrength(s) ^ avgFileChurn(s) => isBuggy(s)
0.617782  !avgFileChurn(a1) v !avgOverallStrength(a1) v isBuggy(a1)
// 2.3975  avgFileChurn(s) ^ highFractalValue(s) => isBuggy(s)
Appendix C: Customized Rules for Each Project

\[ 2.3975 \ \neg \text{avgFileChurn}(a1) \lor \neg \text{highFractalValue}(a1) \lor \text{isBuggy}(a1) \]

// 0.426634 \ \text{highOverallStrength}(s) \land \text{avgFractalValue}(s) \Rightarrow \text{isBuggy}(s)

\[ 0.426634 \ \neg \text{highOverallStrength}(a1) \lor \neg \text{avgFractalValue}(a1) \lor \text{isBuggy}(a1) \]
Curriculum Vitae

Name: Piyush Kumar Korlepara

Post-Secondary: University of Western Ontario London, ON 2019-2021 M.Sc

GITAM University, Hyderabad, Telangana, India, 2014 - 2018 B.Tech.

Related Experience:

Full Stack Developer
Initlive Inc.
2021 - current

Teaching Assistant
The University of Western Ontario
2020 - 2021

Software Engineer I
NCR Corporation
2018-2019