Design of information tree for support related queries: Axis Communications AB

Exploratory research study in debug suggestions at Axis Communications, Lund

Santosh Bharadwaj Rangavajjula

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
This thesis is submitted to the Department of Computer Science & Engineering at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Computer Science. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Santosh Bharadwaj Rangavajjula
E-mail: sara16@student.bth.se

Industry advisors:
Patrik Iselind (Software Developer)
Christian Petersson (Software Developer)
Axis Communications AB, Lund

University advisor:
Prof. Dr. Julia Sidorova
Dept. Computer Science & Engineering

Dept. Computer Science & Engineering
Blekinge Institute of Technology
SE-371 79 Karlskrona, Sweden
Internet : www.bth.se/didd
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57
Context. In today’s world we have access to so much of data than anytime in the past with more and more data coming from smart phones, sensors networks and business processes. But, most of this data is meaningless, if its not properly formatted and utilized. Traditionally, in service support teams, issues raised by customers are processed locally, made reports and sent over in the support line for resolution. The resolution of the issue then depends on the expertise of the technicians or developers and their experience in handling similar issues which limits the size, speed and scale of the problems that can be resolved. One solution to this problem is to make relevant information tailored to the issue under investigation to be easily available.

Objectives. The focus of the thesis is to improve turn around time of customer queries using recommendations and evaluate by defining metrics in comparison to existing workflow. As Artificial Intelligence applications can have a broad spectrum, we confine the scope with a relevance in software service and Issue Tracking Systems. Software support is a complicated process as it involves various stakeholders with conflicting interests. During the course of this literary work, we are primary interested in evaluating different AI solutions specifically in the customer support space customize and compare them.

Methods The following thesis work has been carried out by making controlled experiments using different datasets and Machine learning models.

Results We classified Axis data and BugZilla (eclipse) using Decision Trees, K Nearest Neighbors, Neural Networks, Naive Bayes and evaluated them using precision, recall rate and F-score. K Nearest Neighbors was having precision 0.11, recall rate 0.11, Decision Trees had precision 0.11, recall rate 0.11, Neural Networks had precision 0.13, recall rate 0.11 and Naive Bayes had precision 0.05, recall rate 0.11. Result shows too many false positives and true negatives for being able to recommend.

Conclusions In this Thesis work, we have gone through 33 research articles and synthesized them. Existing systems in place and the current state of the art is described. A debug suggestion tool was developed in python with sklearn. Experiments with different Machine Learning models are run on the tool and highest 0.13 (precision), 0.10 (f-score), 0.11 (recall) are observed with MLP Classification Neural Network.

Keywords: predictive analytics, machine learning, issue tracking systems, customer support, after sales
Acknowledgements

Firstly, I’d like to thank my supervisor Dr. Julia for helping me from time to time and our Director- International Relations V. Gurudutt for his much needed guidance and counseling.

I’m deeply indebted to my grand mother, father, mother and sister for their constant motivation and support.

Patrik and Christian, It’s been an exciting time with you all so far. I don’t have enough words to express my gratitude towards efforts made by each one of you to incubate us into Axis. Over the past few months, I felt like I’ve learnt more than any other point of time in my career, not just professionally, but also personally. A great saga needs expertise of all kinds and we gained a lot of motivation in embedded software development, testing and more importantly, the support services through the thesis program, in our quest to constantly break new ground. I look forward to collaborate more with you for times to come and thank all unicorns out there as we leap into the completion.

A big thanks to Henric Stenhoff - a very supportive line manager who gave me similar assignments to correlate with my thesis- Niklas Johansson, Per Swillus, Olof Persson, Michael West, Marcus at Ericsson Göteborg for motivating me.

Thanks to Dan Matthews, Gökan Kurt of IFS Göteborg who gave me the first opportunities in Artificial Intelligence and Machine learning world.

I’d also like to take the opportunity to thank my thesis partner and dear friend Li Kang for his support and positive spirit he shared with us.

Finally, a big thanks to my friends Shravya, Shiva and Karthik for sharing all the happy and sorrow situations.

Above all, I would like to personally thank you for choosing to read through my thesis work and hope you will find it worthy of time spent.
# List of Figures

2.1 Issue Tracking System ........................................... 5  
2.2 Line of organization at Axis ..................................... 6  
2.3 An example bug report ............................................ 7  
2.4 Mind Map .......................................................... 10  

4.1 Systematic review process ....................................... 17  
4.2 Usecase View ....................................................... 19  
4.3 State Diagram ...................................................... 20  
4.4 Experimental work flow .......................................... 21  
4.5 Sample Data from Eclipse dataset ............................. 22  
4.6 Student’s t-test results .......................................... 23  
4.7 Analysis of Metrics used ......................................... 24  

5.1 Architecture of pipeline .......................................... 26  
5.2 Work-flow of verification and prediction process ............. 27  
5.3 Example Decision Tree ........................................... 28  
5.4 Example KNN Classification ...................................... 29  
5.5 Example Multi layer Neural Network ........................... 30  

7.1 Analysis of results ................................................. 38
### List of Tables

<table>
<thead>
<tr>
<th>Table Number</th>
<th>Table Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Quality Criteria</td>
<td>15</td>
</tr>
<tr>
<td>4.1</td>
<td>Machine Learning models analysis</td>
<td>24</td>
</tr>
<tr>
<td>4.2</td>
<td>Metrics Used in similar research</td>
<td>24</td>
</tr>
<tr>
<td>6.1</td>
<td>Results for Decision Trees</td>
<td>31</td>
</tr>
<tr>
<td>6.2</td>
<td>Results for K nearest Neighbors</td>
<td>32</td>
</tr>
<tr>
<td>6.3</td>
<td>Results for Naive Bayes</td>
<td>33</td>
</tr>
<tr>
<td>6.4</td>
<td>Results for Neural Networks</td>
<td>35</td>
</tr>
<tr>
<td>6.5</td>
<td>T-test for Neural Networks</td>
<td>36</td>
</tr>
<tr>
<td>6.6</td>
<td>T-test for Naive Bayes</td>
<td>36</td>
</tr>
<tr>
<td>6.7</td>
<td>T-test for K Nearest Neighbors</td>
<td>36</td>
</tr>
<tr>
<td>6.8</td>
<td>T-test for Decision Trees</td>
<td>37</td>
</tr>
<tr>
<td>8.1</td>
<td>Summary of the relevant work</td>
<td>51</td>
</tr>
</tbody>
</table>
Abstract

1 Introduction
   1.1 Context ................................ 1
   1.2 Scope ................................ 1
   1.3 Problems with existing systems .......... 2
   1.4 Thesis Contribution ....................... 2
   1.5 Thesis Organization ....................... 3

2 Background
   2.1 Background ................................ 4
      2.1.1 Thesis Description ..................... 4
   2.2 Motivation for thesis ...................... 4
   2.3 Current Architecture and tool Stack ........ 4
      2.3.1 Issue Repositories ..................... 5
      2.3.2 Support Organization at Axis ............ 6
      2.3.3 Bug Report ........................... 6
   2.4 Proposed work ............................ 7
      2.4.1 Challenges ........................... 8
      2.4.2 Advantages to Axis ..................... 8
   2.5 Aim and Objectives ....................... 8
      2.5.1 Aim ................................ 8
      2.5.2 Objectives .......................... 8
      2.5.3 Problem Statement ...................... 9
      2.5.4 Analysis ............................ 9
      2.5.5 Mitigation of Bias ...................... 10

3 Related Work .............................. 11
   3.1 Relevant Work .............................. 11
      3.1.1 Research Gap .......................... 14
      3.1.2 Quality Criteria ....................... 15
# Method

4.1 Methodology

- Systematic Mapping study

4.2 Research Execution

- Experimental design
- Data Collection and Extraction
- Experimental Setup
- Machine learning
- Validation
- Description of metrics

## Proposed Model

5.1 Our Approach

5.2 Pipeline

5.3 Machine Learning Models

- Software Specifications
- Hardware Requirements
- Decision Trees
- K Nearest Neighbors
- Naive Bayes
- Neural Networks

## Results

6.1 Research Findings

- Decision Trees
- K Nearest Neighbors
- Naive Bayes
- Neural Networks
- Repeatability
- Statistical Analysis
- Threats to Validity

## Analysis

7.0.1 Discussion

## Conclusions and Future Work

8.1 Conclusion

8.2 Future Work

## References

8.3 Summary of related work
Chapter 1

Introduction

1.1 Context

The world today is at the brink of a data revolution. With more smart devices capable of extracting data from the environment getting connected to the Internet. There is now massive amounts of data that can be understood and put to use. On the other hand, Artificial Intelligence, which was mostly a research interest, is now a new buzzword in today's Information Technology (IT) industry and that is due to improved flexibility of algorithms, efficiency and accuracy of the predictions made. Applying Artificial Intelligence (AI) principles to tailor the debug data can improve the customer service by making the support line thinner and faster. According to Guo et al. [1] Mozilla received 170 bug reports per day in 2009, with its database holding more than 500,000 bugs. This data, if carefully extracted and understood, can help to create a baseline to take key business decisions, for example prioritizing resolution of bugs or assigning the bugs to developers [2]. Axis Communications receives more than 70,000 issue reports per year, a few of them could be duplicate or identical in nature, Kang [3] formulated a bug triaging algorithm to identify duplicate issue reports which reduces support time by recommending resolved issues. It might so turn out, that issue report has no duplicates, suggesting trouble shooting information (debug data) like crash reports can make the support line even more faster and efficient.

1.2 Scope

The focus of this thesis is to find an efficient prediction algorithm in the context of suggesting debug information for support issues and evaluate it by defining metrics for a reference architecture. As the support data can have a broad spectrum, we confine it data pertaining to software issues from internal Axis Customer Support Tool (CST). Issue Tracking is a complicated process as it involves various stakeholders with conflicting interests. A complicated and diverse arrangement of the teams is yet another constraint for the support text written in different languages and assumptions, most of which are not written down. We wish to exclude such issues (not written in English and vague descriptions). During the
course of this thesis project, we are primarily interested in gathering data from CST, cleaning data to fit to the context, tuning parameters to find the optimal recommendations, describe and evaluate prediction models. Li implemented a CST [3] with bug triaging in Django which can identify duplicate issue reports but doesn’t take any actions in case when no relevance is found. During the course of research, the missing functionality in recommending diagnosis data is developed and tested for effectiveness. The existing CST could give out irrelevant similarity especially in case of low duplicate percentage, which can be troublesome from a reliability perspective due to outliers in the input dataset.

1.3 Problems with existing systems

Tracking issues is an important part of delivering quality software, which takes more than 45 percent of the total cost [4]. According to Zhang and Sun [5] redundancy and noise affects the quality of bug reports. While redundancy slows down the resolution by making developers re-resolve issues, noise misleads developers. With more smart phones and mobiles joining the Internet [6], the world today is at the brink of a data revolution. With one-third of the world being on social media[7], there is a high chance of a customer raising an issue on the social media networks which can be considered as potential cases in the Issue Tracking Systems(ITS).

Currently Knowledge Bases (KBs) are being used to refer to commonly occurring issues by creating a knowledge article. With the volume of issues being created today, converging to the knowledge article in relevance to the issue entered is a difficult task. Adding to it is a problem of abstraction and KBs are already proven to be efficient in handling incomplete information especially when it changes dynamically [8].

1.4 Thesis Contribution

During the course of the thesis, we gather and evaluate a multitude of research work, understand and explain the state of the art of the existing system. We then describe, present our case and research questions. Research methods used during the literary work, reports and comparisons are presented in the methods section. We then conclude the research study by evaluating the effectiveness of different solutions to suggest debug information by defining metrics and rest our case with a conclusion.

According to Xuan et al., [4] bug resolution is impacted by low quality and large scale. While the large scale is almost unavoidable in real-time issue tracking systems, the former is a resultant of redundancy and noise [5]. While Li [3] improved the turn around time for resolving customer’s queries, we focus on
reducing the noise by recommending the debug data that is needed to resolve the issue.

1.5 Thesis Organization

The following thesis document is organized as follows. Section 1 introduces the thesis work and set a context for the thesis. In Section 2, we present the use case, motivation for our thesis and in section 3, we evaluate the existing literature with a systematic literature study. In section 4, we describe the research methods used during the thesis work and present them in the order of importance. In section 5, we present our proposed model, approach and machine learning models we used. In section 6, we discuss the results obtained by carrying out the research method and in section 7, the results are analyzed. We summarize, conclude the research work in section 8, discuss limitations and describe the future work.
Chapter 2

Background

2.1 Background

2.1.1 Thesis Description

The proposed thesis work was carried out at Axis communications AB, which is now a part of Canon Inc. and specializes in the field of surveillance systems, is a designer and manufacturer of CCTV security cameras. With about 2600 employees, Axis Communications is a leader in highly reliable surveillance systems with low cost of ownership.

2.2 Motivation for thesis

At Axis Communications, Issue Tracking Systems are typically first point of contact for the customers and is also used by various levels in the organization for internal problem reporting as depicted in Figure 2.1. The issues are reported to an internal help desk ITS either by end customer, distributors, internal employees or in partner network when they find a problem either by phone, email, chart or the help desk system on the official website. On further examination of the issue, it is further forwarded as a ticket in their internal customer support issue tracking system which is a separate deployment. If the customer query is identified as a bug it is reported in a Jira - a bug tracking system. This brings inconsistency of formats and language into the system.

2.3 Current Architecture and tool Stack

Organization for support services at Axis Communications, Lund is flat i.e, the line of control or hierarchy is minimal with dedicated teams for development and support. Developers, on checking in the code into gerrit, a git system, a CI/CD build process starts with internal Jenkins deployment. A local deployment of JIRA - an issue tracking system by Atlassian along with in house ITS is in place. The internal ITS gets more than 70000 issue tickets every year, most of
them raised by customers and a few reported internally. Axis has its warehouses, production units, development centers and local support spread across the globe which share internal software deployments like JIRA and other internal Issue Tracking Systems. This also brings in challenges with issues reported in different languages.

2.3.1 Issue Repositories

To effectively manage incidents and customer queries in a software system, software companies generally store their reports in a database or repository. Depending on the nature of the reports, companies may choose to have either have one or more centralized or distributed issue repositories to keep track of the issue progress. At Axis communications, customers report their problems through a front end help desk application. The application is then processed manually by a support technician called first in line of support as shown in figure 2.2, who goes through a Knowledge Base, categories the problem and raises a ticket through front end clients which then gets stored in the database. While some issue can be resolved by support technicians, many require a more detailed review and understanding which is done through escalation.

When an issue is escalated, it gets further forwarded into Issue tracking System and notifies the next in line technician, who then go through the reports using front end kanban boards in ITS like JIRA, BugZilla, YouTrack and assign tasks to resolve the issue. Typically, issue resolved at a second in line are of nature needing basic business knowledge like product family, newest offerings, discounts and capacity of production. Yet, there still remain lot of issue needing a more profound understanding. The last line of support handles only 5 percent of entire problems reported and needs expert resolving them. These issues most often are the nature of software bugs, faults, crashes mostly resolved by senior software developers by
making changes in the source. At Axis communications these reports are handled separately.

2.3.2 Support Organization at Axis

At Axis communications, the service support is a layered organization with a few layers having a hierarchy of expert groups local to the division 2.2. In this research we propose to design a knowledge based expert system which takes the input data from different levels of the support system and recommends similar cases or debug information when the case gets inserted at customer care or First in Line of Support (FLS).

![Figure 2.2: Line of organization at Axis](image)

2.3.3 Bug Report

A bug report is an object of a bug repository [9] comprising of bud details like reporting date, reporter, assignee, severity, short description, details to reproduce, category, firmware, product, version, labels, expected time of completion and bug id.

In some ways, verification of bug reports or JIRA ticket is also a classification problem as a general layout of a bug report consists of following textual elements:

- What is the problem.
- Steps to reproduce.
- Expected result.
- Actual result.
- Reporting and assignment to the appropriate team member. (Eg: Forward to abcd)

Therefore, a bug report can be said to have textual, numerical, categorical and meta information. During the process of completion, bug number are numerical information, description, labels, details to reproduce are textual
information, while category, firmware, product, version, reporter, assignee, severity are categorical information. Some bug reports have crash dumps and screen shots attached to them which are not textual or numerical or categorical in nature but are integration to resolution of the bug.

2.4 Proposed work

A considerable amount of research work in text similarity has been finished with vector space models [10], yet the vector space models have been turned out to be ineffective with large documents. Another issue with vector space models is that they do not stress on the semantic connotation of the similitude and in this way it isn’t much useful.

As indicated by Wang et el. [11], in his research on text similarity, it can be ascertained that relevance in text classification can be computed by modeling them into topics and training the models with a dataset. This should be possible to utilize regression models by finding the relationship between the text with probabilities or text classification models along with K-nearest neighbors [12] or Decision Trees. We propose to utilize learning based techniques to classify text semantically in our research.
2.4.1 Challenges

The field of image recognition in machine learning is a more mature field than text classification according to Kang [3]. A difference between classifying images and issue reports is text which is slightly more difficult to handle for its abstractness and contextual usage. Generally, in image recognition and classification, a matrix of 0’s and 1’s with intensities between 0 to 255 for black and white spots of the image is extracted and then flattened matrices or Vectors of different images are compared to identify similarities and differences. In case of a colorful image, three matrices are used for Red, Green, Blue respectively.

Textual classification in that sense is complicated in relation to images as vectors which are already abstract are now bulky and meaningless (as a lot of words in a typical sentence relate to each other) without context as we can see in figure 4.5.

2.4.2 Advantages to Axis

Advantages of this research to Axis Communications includes but not limited to faster issue resolution, analyzing issue reports to calculate similarity between issue reports, recommend similar resolved bug report solutions for reading for faster resolutions and recommend the much required debug information forehand to reduce customer annoyance.

2.5 Aim and Objectives

2.5.1 Aim

The research focus on reducing the average turn around time by equipping the support line with an intelligent service support system to the existing ITS. The primary aim of broader research is finding text similarity functionality in service cases for detecting duplicate cases in real time, specifically in the context of ITS. A secondary aim is to find a recommendation feature in the ITS in case no similarity is found to recommend debug information that needs to be asked to the customer for troubleshooting.

2.5.2 Objectives

- Statistical analysis of patterns in the internal ITS data set, calculating correlations and extracting features.

- Data collection from internal Axis employees and partners about the functioning of the existing system and find methods to interact and experiment.
Chapter 2. Background

- Understand and implement the standard principles in global software product maintenance.

- Finding ways to automate filtering, categorization, and redirection of a significant portion of correspondence in the ITS.

- Investigating different models to calculate similarity, specific to customer service in ITS and rank the suggestions in the order of similarity.

- Investigating different models in context of debug suggestions and compare them.

2.5.3 Problem Statement

A fast and efficient customer service is the key to retain existing customers [13] and adding more layers in the workflow only makes the process slower with the issues moving back and forth between the teams. In order to make the support process faster, the support chain has to become thinner, which essentially is a difficult scenario in case of big companies. Alternatively, the support chain can be made more intelligent with Decision Support Systems (DSS) and recommendation systems. One way of achieving it using Decision Trees. With currently used Knowledge Bases (KBs) for software support it is a tedious process to create knowledge articles and especially search through them given the volume, velocity and variety of the issues raised. The current problem with the ITS is also a big data problem with a volume of more than seventy five thousand cases created each year, having a lot of variations depending partly on creator of the ticket (Variety). With this volume also come many problems as data is traditionally stored in a data warehouse which are inflexible and expensive.

2.5.4 Analysis

Raising customer expectations about issue resolutions require reduction of turn around time in resolving issues by efficiently managing the raised tickets and reducing wastage of resources. According to Xuan et al.[9] redundancy and noise are two major problems that lead to wastage of time, thereby slowing down the process. Considering the multi level stage issue resolution at Axis communications, data extraction across different levels and issue tracking systems combined with a machine learning model to identify duplicates and also propose maintenance information needed to resolve the issues in case its not a duplicate will reduce redundancy and noise in the system, making it efficient and faster.
2.5.5 Mitigation of Bias

Bias is an unavoidable problem in research. Although it may not be possible to eliminate bias completely, it can be reduced. One way to contain bias is by considering different viewpoints of various stakeholders in the project. Figure 2.4 shows a mind map containing use cases, stakeholders and activities of the underlying research. We also employed the snowball sampling technique and did a backward snowball sample with our keywords. Thirdly, we discussed extensively with our supervisors at Axis Communications to understand the problem.

Figure 2.4: Mind Map
Chapter 3

Related Work

3.1 Relevant Work

In this section, we present a concise presentation on the existing related work and existing state of the art that pertains to our research in recommendation of debug information in Issue Tracking Systems.

One of the most accorded work that sets a base line is done by Edmison and Edwards [14], who developed a customised driver to be used with GZoltar (spectrum based fault localisation) test automation software for ranking to each row of the program written in Java for evaluating student assignments for faster feedback to students to lessen their frustrating moments. The tool is then embedded into their proprietary web application - WEB-CAT that gives a score between 0 to 1 for each line of the code predicting probability of faults. They believe that student written JUnit tests written pre-development as a part of Test Driven Development (TDD) methodology are not properly written by students especially for boundary cases. Their data set consisted of 212 programs of assignment for minesweeper game and queue of which 135 had bugs and achieve a 73 percent precision.

One of the objectives of our research is to recommend maintenance information to a fault which requires identifying the fault, this article [14] sets a baseline for fault localization.

Vetro et el., in their research [15] analyse the tool for finding software defects called FindBugs in the context of university projects. They found that only 2 out of 15 groups of issues can be considered as reliable predictors of actual defects by analysing 508 issues of 85 versions of lab assignments. Although, they do not touch upon prediction tools and techniques, they introduce the state of the art in bug analysis and evaluate tools used in industry.

Xia et al.[16], in their research propose an imbalanced multi-label classifier to reassign bug tickets. They base their study on changes in details meta or non textual information like os, platform, assignee change on comparison to re-assignment. They also argue that multi-label classifier with text, meta, mixed classifiers used with ml-compose for new bugs will improve prediction on assignee. They use four open source projects from Mozilla, Eclipse OpenOffice Netbeans to
collect 190558 bug tickets or issues with the help of Bugzilla - an Issue Tracking System, used by the community of those projects. Although the aim of their [16] research is slightly different from ours which is re assignment of tickets, the data sets selection and non textual classification are of interest to our research.

Limsettho et al.[17], use a slightly different topic modeling approach to categorise bug reports by using Hierarchical Dirichlet Process (HDP) and project bug reports into topic vector space. While most of the other reviewed literary work in topic modeling is focused on Latent Dirichlet Allocation $LDA$ [18], this is perhaps the only research article we came across that used HDP. The highest F-Measure of 0.787 is achieved with logistic regression.

Sujon et al. [19] use Term Frequency- Inverse Document Frequency (TF-IDF) to project the code into vector space. Their research is focused on finding performance bugs. They argue that performance bugs are hardest to find and lead to poor client experience, low throughput, slow system down and waste a lot of computation time and resources. Although, they don’t impact the functioning of the system. Their framework scans through the reported performance bugs from an Issue Tracking systems and doing a static code analysis of the underlying code. They propose a more profiling and reasoning strategy that takes into consideration bug types, bug sources unlike a traditional simple static code analysis method. They claim that their system is trained based on symptoms using a naive bays classifier and also does the fault localization which gives them 75 percent accuracy.

Pingclasi et el.,[18] propose a topic modeling technique based on Latent Dirichlet Allocation to classify bug reports with a rule based system. Authors [18] classify the report as a bug if it had a NullPointerException, had descriptions about code changes, pertained to run time or memory issues. Their feature set included title, description, discussion and uses open source projects HTTPClient, Jackrabbit, Lucene. The machine learning models used in the research were decision trees (specifically, Alternating Decision Tree), naive bays and logistic regression over 7000 issue reports.

Phan and Nguyen [20] instead of using sequences of programs to detect bugs, convert it into assembly and use them as feature set with abstract syntax trees instead. Their dataset was obtained from codechef, a programming contest website containing programs in C, C++,Python of four different tasks. They use multiview convolution neural networks with four layers and a fully connected layer in a mapping and reducing fashion with pool, merge layers. Described learning rate is 0.1 and size of the tokens is 30 and claim that their Abstract Syntax Tree Convolutional Neural Network - ASCNN enhances F1 score by 10.94 percent.

Singh and Verma [21] in their research, analyse the most popular Object Oriented Chidamber and Kemerer CK metrics and investigate relationship between software metrics and defects. Data extraction: They use Metric data and bug data from SourceForge platform for different versions of iText and scan the logs they also get the source code. Then they see how far the class with error is in
Chapter 3. Related Work

the inheritance tree and how many children it has and how many relations it has to other classes. Finally fault prediction is done using decision trees and bayes classifier with WEKA platform. Performance in their research was measured using Accuracy, Precision, recall, f measure with 10 fold cross validation. Their research findings explain that Naive Bays is better than J48 decision trees.

Theisen et al., [22] describe the Risk based attack surface approximation (RASA) technique to scan the core dumps, look at the stack trace and connect it to individual files. The authors describe methodology that will calculate risk in core dumps generated randomly by finding vulnerabilities and relate them to individual classes in code. They also attempt to scan the bug reports and connect them to the the core dumps. Although, there is no prediction or recommendation described in the research, the vulnerability finding methodology is interesting to our thesis work. In a few bug reports we came across at Axis Communications, there were attachments of core dumps resulting from a crash in the camera software. Apart from the methodology, their background work relates to articles focused on localising vulnerabilities on scanning the bug reports. The authors claim that coverage through entry point of application by scanning APIs is not sufficient and map source code of the projects to vulnerabilities instead to get a broader understanding with better coverage. A Mozilla Firefox dataset with 50000 records is used for this purpose.

Otoom et al., [23], In their research on severity prediction of software bugs, worked towards bug severity prediction using bug tickets from open source projects Bugzilla, Eclipse and Gnome. Instead of contextual similarity, they classify text with feature extraction, tokenisation (to remove stop words) and applying Naive Bayes (NB), RBF Networks, Functional Trees (FT), Random Trees (RT), Random Forests (RF) and AdaBoost algorithms. They separated their dataset into 70-30 percent for training, testing and do a 10 fold cross validation. The results from their experiments with 59 features and 163 bug reports show that Ada Boost, by combining weak classifiers improved the accuracy of prediction by 0.0 to 4.9 percent.

Kaushik et al. have performed a comparative study for duplicate bug detection [24] which compares between information retrieval models. According to the author the information retrieval models can either be topic based or word based.

Köpcke et al. [25] use the FEVER (Framework for EValuating Entity Resolution) for measuring quality match and run time efficiency which is interesting as it fits into our research and can be used in our experiments.

Xiao et al. [26] use similarity joins to group the similar words together and generate suggestions for more words given some. We can use this when making one of our prototypes and tag our data set for recommending new case give one.

Christen et al. [27] discuss FEBRL (Freely Extensible Biomedical Record Linkage) that provides a support vector machine (SVM) implementation for learning suitable matcher combinations

Xuan et al. [9], claims that 45 percent of cost is fixing bugs and use text
classification (Naive Bayes) for automated assignment of bugs using instance selection and feature selection over 60000 issue reports. They argue that speed of resolution of bug reports is impacted by two factors namely large scale and low quality. While scale is difficult to control in large corporations, low quality, according to them is a result of noise (improper descriptions) and redundancy. In their research, they claim to have built a binary classifier that predicts the order of instance selection, feature selection. They compare their results with Support Vector Machines, K-Nearest Neighbors, Naive Bayes and compare time, cost of manual assignment of reports to fully automated. Metrics used were F-measure and precision.

Lamkanfi and Demeyer [28], work on the problem of erroneous bug reports to predict their reassignment. They argue that reporters being unaware of technicalities of the underlying software tend to improperly report bugs which adds to manual work of bug triager. They analyse the frequency of change of product component field in the bug report and investigate the viability of a classifier. The features they use are component, reporter, operating system, severity and summary. Metrics used are precision, recall rate with 10 fold cross validation. Experimentation tool used is WEKA for Naive Bayes classifier to predict reassignment frequency of product category. Four projects from Eclipse and Mozilla are considered for dataset creation from BugZilla Issue Tracking System. A precision of 0.58-0.93 and recall rate between 0.54-0.82 was observed.

Gou et al. [29], focus on trends in number of software bugs and predict the number of bugs of a future medical imaging software by using static analysis and complexity metrics like change rate of software code. They extract the source code for the project, no of bugs discovered in the code historically in different versions, do the static analysis and get complexity metrics. Then correlation analysis between static analysis metrics and bug numbers is carried out with IBM SPSS statistics software to predict the bug number of future release. First five versions of historical data is is used as training set and 6th or newest version is used for test. Relative error of 1.426 % was observed.

In their research, Seo and Kim [30], explain that 48 percent of crashes in Firefox project are recurring ones due to incomplete fixes and propose a method to predict recurring crashes by extracting stack traces and comparing them to the bug fix locations. A tree based structure is used to balance speed and precision and a precision of 0.57 with F-measure of 0.53 is claimed to be recorded when at expansion level 4 with 1159 stack traces. 292 stack traces were claimed to have been observed to be recurring.

### 3.1.1 Research Gap

On reviewing our research articles we observed the lack of emphasis of recommending maintenance information in the context of issue resolution and software support. Although a lot of research had been done in terms of machine learn-
ing models fitted towards re-assigning issue reports [16], identifying and ranking faults [14], finding software defects [15], categorize bug reports [17], we have not found any article focused towards our objective which gives us an opportunity to advance our research.

3.1.2 Quality Criteria

In order to streamline the quality of research, we assessed the research articles in terms of their relation to our objectives with the help of following questions and summarize them in table 3.1.

- QC1: Does research article describe datasets used?
- QC2: Are machine learning algorithms used detailed in the article?
- QC3: Is measurement of the techniques used discussed in the article?
- QC4: Are limitations stated clearly in the research article?

<table>
<thead>
<tr>
<th>Article</th>
<th>QC1</th>
<th>QC2</th>
<th>QC3</th>
<th>QC4</th>
</tr>
</thead>
<tbody>
<tr>
<td>[14]</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[15]</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[16]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[17]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[19]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[18]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[20]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[21]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[22]</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>[23]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[24]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[9]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[28]</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
</tr>
<tr>
<td>[1]</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>[30]</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 3.1: Quality Criteria
4.1 Research Methods

This chapter covers the different research methods used and explain procedure, implementation and measurements.

4.1.1 Systematic Mapping study

Review Method

Pilot Study: The initiation of the research is started with a motivation to improve the speed of existing issue tracking system using recommendation, prediction algorithms and keywords have been identified from the title as "Decision Trees" AND "Information trees" AND "Service support systems" which returned 94 results for a basic search on IEEE but the articles were completely irrelevant to the domain of our research. Therefore, we refined our search string to "Prediction" AND "Software Support", an initial database search on IEEE returned 1500 results, which returned a few useful results but also many that are irrelevant to the focus of this research. Therefore, search string had been further modified to ((Prediction OR Recommendation OR Suggestion) AND (bug OR issue OR support ticket) AND (debug information OR stack trace OR crash dumps) AND (methods OR techniques) ) for narrowing the nature of results to area of research and a basic search on IEEE Xplore returned 52 results of which 40 are found to be relevant. Later, four research articles are snowball sampled from a research article. A pictorial representation of our systematic study is detailed in the figure 4.1.

Database selection: Our initial basic search stated with using IEEE Xplore which is a recognized choice in the field of our research and later ACM was also found to give good search results. Scopus, which is a citation database is also referenced for our research.

Research Questions

• RQ1: What are the different factors that slow down the processes in a tech-
Chapter 4. Method

Figure 4.1: Systematic review process

Motivation: There might be a lot of processes that slow down the entire support line, in terms of people, projects or processes. This research question is framed for acquiring knowledge on studying the current state of the art in handling customer queries efficiently through an investigative approach. With the help of this research question, we wish to focus our study on performance of different software systems and project management models to synthesize the impact of bottlenecks. As our research is primarily focused on speedup, this research question will also help us to get a bigger picture, explore and find gaps in the current state-of-the-art in terms of time, cost and efficiency.

- RQ2: Which components of issue tracking systems can be improved or changed to offer faster services?
  Motivation: The research aims to find an efficient solution for improving the turn around time for customer query, it is therefore interesting to identify the bottlenecks in the system and analyze which of those can be improved with current research.

- RQ3: Can the existing issue tracking systems be made faster by automating
processes with recommendation and prediction algorithms?

Motivation: Automating some routine tasks can improve the turn around time but may not be viable solution for all cases. We are therefore, curious to know if Artificial Intelligence (AI) concepts can help make support chain faster and if its the only efficient approach. This research question, will help set the focus on our research to find appropriate metrics to evaluate our results.

Included and Excluded Study

In order to control the quality of the articles reviewed, We have considered reviewing the research articles that follow the following criteria:

Inclusion Criteria:

- Articles with full text.
- Articles in English.
- Journal articles and conference proceedings.
- Articles that focus on recommendation systems.
- Articles focused on issue/ bug tracking systems.
- Articles describing data acquisition and processing.
- Articles describing algorithms for predictive analytics.
- Comparative research articles focused on evaluation of prediction algorithms.

Exclusion Criteria:

- Articles that do not focus on recommendation / prediction.
- Articles that were not peer reviewed.
- Articles published prior to 2009.
- Research articles not relating to computer science.
- Articles that do not have full text.
Chapter 4. Method

4.2 Research Execution

On going through a multitude of previously evaluated work and examining research methods used in other studies we referred to we decided to conduct experiments as our primary research method. This is because, experiments give more control over executing the study which is important to study facts in isolation. Adding to that, most research articles we reviewed point that way. As a secondary study, we carry out interviews within Axis Communications to verify the results from experiments mostly kept internal for their sensitivity.

4.2.1 Experimental design

To carry out our experiments with different machine learning algorithms, we needed a tool to build and test our solution. Doing our literature review, it became evident that chosen domain still is new as most of research articles found are conference proceedings and very few journal articles. It was also clear that of a readily available tool does not exist, so we decided to build our own. Figure 4.2 shows the use case view and Figure 4.3 shows the state chart for the system.

![Use Case View](image)

We queried the CST rest api and compiled the data into a csv file using python JSON, csv libraries. To cross-validate our results we also acquired issue tracking data from Ericsson - Mobile Baseband Modules department (having signed non disclosures). As we will not be able to present the above data sets, we have also written some example scripts of the tool that can be used with open source Eclipse dataset. Then, we have written python scripts to extract data using pandas library from csv file and converted categorical variables to numbers with numpy. We then cleaned the data and removed columns that have low correlation coefficients. We then divided our input data into 70-30 as suggested by Sujon et al. [19] to be used with Decision Trees, K Nearest Neighbors, Random Forest and Neural Networks. Finally we test our solution and calculate Precision, recall and accuracy. Figure 4.4 explains the working process.
4.2.2 Data Collection and Extraction

In order to carry out our experiments, data is essential. As our objective is to suggest debug information, we start by collecting data from bug reports at Axis Communications. We query a rest API and get back a JSON response consisting of fields in bug report. We then convert them into python dictionary, extract the relevant fields and store them as a CSV file. The database has the following fields: Product, version, firmware, tags, last updated, assignee, reported by, customer details, severity, comments, priority. From Eclipse dataset, we extracted Bug ID, Product, Component, Assignee, Status, Resolution, Summary, Changed, Assignee Real Name, Classification, Flags, Hardware, Keywords, Number of Comments, Opened, OS, Priority, QA Contact, QA Contact Real Name, Reporter, Reporter Real Name, Severity, Summary, Tags, Target Milestone, URL, Version, Votes, Whiteboard and Alias.

Our thesis is targeted towards debug suggestions dovetailed to Axis CST but in order to explain our experiments, we have chosen to also use Eclipse historical bug dataset acquired from kaggle for it was was used in similar research [23, 24, 28].

4.2.3 Experimental Setup

In this subsection, we discuss the experimental setup which is done manually. Microsoft Excel is used for modifying the dataset.
Duplicate Detection and other external systems

Kang [3] in his research works describes the system to identify and rank duplicates for issues in CST deployment of Axis communications which is a web application based on the Django framework, this can be used to find duplicates and requests with queries for a similar type of issue can be returned with maintenance or debug information along with a "INFO" tag can be returned in case no similarity is found in the system. However, the debug recommendation system can run in isolation regardless of the Django web application by using a dataset in CSV format and running it locally.

Data cleaning and preparation

Data preparation is an important step in carrying out machine learning experiments. According to Lu et al. [31], selecting the right kind of data and correlating it to the objective of the experiment is more important than optimization and tuning. As the dataset used in our research is a combination of several datasets, it is needed to harmonize the data for example the time stamps which were in different formats, serial numbers, similar components, hardware and OS tagged differently had to be searched and modified. As we manually triaged the maintenance or debug information, it had to be manually added as a column in the dataset.
Extracting feature vectors

During our extraction process, we start with a descriptive statistic approach i.e., identifying relevance of the collected data and problem under investigation, i.e., debug prediction, compare the mean of random sample and eliminate those columns that are irrelevant, e.g., the number of watchers in Eclipse dataset. Then we manually analyzed a few samples of data set in random and eliminate fields that might not contribute to predictions e.g., we eliminated the category column as we found 'Eclipse' in 112 of 150 samples and 'Other' in 38 samples which doesn’t give any insight into the problem. Then we run the Ordinary Least Squares (OLS) regression which also calculates the t-values on all potential features i.e., ['Product', 'Component', 'Hardware', 'OS', 'Classification', 'Status', 'Resolution', 'Severity', 'Number of Comments', 'Votes'] and compute p-values using statsmodel API in python. Figure 4.6 shows the coefficient, standard error, t values and \( p > |t| \). In t-test, generally, \( t = \frac{\text{Difference between means}}{\text{variance}} \) is calculated. The hypothesis can be formulated as follows:

- H0: Selected feature doesn’t influence change in debug information
- H1: Selected feature influences change in debug information

For a significance i.e p-value less than 0.05, we reject the null hypothesis and accept the alternate hypothesis Thus, relating for the statistical results, we have considered Product, status, severity as our feature vectors as they influence change in debug information.
4.2.4 Machine learning

As we would like to understand and tune the algorithms in a cause and effect relation, machine learning algorithms are chosen independent of feature vectors. In line with the state-of-the-art, for our experimentation, we have chosen K Nearest Neighbors, Decision Trees, Naive Bays, Neural Networks, Random Forests for their frequent usage in similar research as reflected in table 4.1. Topic modeling is excluded from our research as we use non textual or meta information.

Evaluating machine learning

During the literature review, we found that precision, reliability, f-measure, recall, accuracy, relative error are the metrics that can be used for evaluating their studies. Analyzing them further, in table shows that Precision, recall rate and F-Measure are the most used as reflected in table 4.2. In order to evaluate the machine learning model, we divided the dataset into training and testing parts and train the model with the training part of the dataset. We then hide the result vector in the testing part of the dataset and run the model. In case the predicted result matches the actual result, it is a true positive otherwise, it could be called...
Chapter 4. Method

<table>
<thead>
<tr>
<th>Machine learning Algorithm</th>
<th>Research study</th>
<th>No of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>K Nearest Neighbors</td>
<td>[16, 32]</td>
<td>2</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>[17, 21, 23, 9, 18]</td>
<td>5</td>
</tr>
<tr>
<td>Logistic Regression</td>
<td>[17]</td>
<td>1</td>
</tr>
<tr>
<td>X means</td>
<td>[17]</td>
<td>1</td>
</tr>
<tr>
<td>Naïve Bays</td>
<td>[19, 18, 21, 23, 9]</td>
<td>5</td>
</tr>
<tr>
<td>Expected Maximization</td>
<td>[17]</td>
<td>1</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>[23, 20]</td>
<td>2</td>
</tr>
<tr>
<td>Random Forests</td>
<td>[9, 23]</td>
<td>2</td>
</tr>
<tr>
<td>RBF Networks</td>
<td>[23]</td>
<td>1</td>
</tr>
<tr>
<td>Topic Modeling</td>
<td>[17, 18, 24]</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 4.1: Machine Learning models analysis

...a false negative...

<table>
<thead>
<tr>
<th>Metric Used</th>
<th>Research study</th>
<th>No of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>[14, 17, 19, 21, 9, 28, 30]</td>
<td>7</td>
</tr>
<tr>
<td>F-Measure</td>
<td>[16, 18, 19, 20, 21, 9, 30, 17]</td>
<td>8</td>
</tr>
<tr>
<td>Accuracy</td>
<td>[19, 21, 23]</td>
<td>3</td>
</tr>
<tr>
<td>Recall Rate</td>
<td>[16, 17, 19, 21, 24, 28]</td>
<td>6</td>
</tr>
<tr>
<td>Relative error</td>
<td>[29]</td>
<td>1</td>
</tr>
<tr>
<td>Reliability</td>
<td>[15]</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.2: Metrics Used in similar research

Figure 4.7: Analysis of Metrics used
4.2.5 Validation

From our multitude of evaluated research, 70-30 train-test split rule is by far the most popular choice for generalizing the results [23, 21, 19, 17, 18, 20]. During our experiments we divide the dataset into training and test using train_test_split of sklearn.model_selection package and randomize the insertion of states using random time series.

4.2.6 Description of metrics

The chosen metrics for our experiments are Precision, recall rate and F-Measure for their usage in similar research 4.2.

According to Xia et al.[16] Precision, Recall and F-Measure can be defined as follows:

\[
\text{Precision} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Positives}}
\]

\[
\text{Recall rate} = \frac{\text{Number of True Positives}}{\text{Number of True Positives} + \text{Number of False Negatives}}
\]

\[
F - \text{Measure} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}
\]
Chapter 5

Proposed Model

5.1 Our Approach

We tried to solve the problem of getting early insight into debug information needed from customer by using historical data, manually triaging it to include debug information and using it later to train our model. In this section, we will discuss the details of the machine learning models used along with a bird’s eye view of the overall functioning of the system.

5.2 Pipeline

As stated earlier in the background section, the current research is a part of a broader objective to achieve speedup to the existing work flow and Kang’s [3] work was focused on duplicate detection while the current research aims at dovetailing debug information prediction to the framework for its believed to make the chain faster. However, predicting debug information is not just a speedup problem but a recommendation problem that aims to reduce customer frustration.

![Figure 5.1: Architecture of pipeline](image)
5.3 Machine Learning Models

In our experiments, machine learning is a very important process as it decides the quality of debug recommendations, so we decided to experiment with several candidate models in the order of most used in reviewed work. In this section, we elaborate on the system requirements and machine learning algorithms used in our studies.

5.3.1 Software Specifications

Python version 3.6.4 is used to write our scripts for experiments, which uses sklearn package for machine learning, csv package for extracting data, pandas for processing data, numpy for storing multi dimensional arrays, json for loading data from rest end points and converting to python dictionaries, statsmodel api for feature selection, scipy to generate statistical reports and jupyter is used to write our scripts in a notebook for testing purposes.

5.3.2 Hardware Requirements

Experiments were carried out on Intel core i5 2.7GHz processor, Iris graphics 6100 graphic card, 8GB DDR3 Random Access Memory and repeated on Intel core i7 3.4 GHz processor with 12GB DDR3, Nvidia graphic card. The Performance difference observed was negligible.

5.3.3 Decision Trees

The Decision Tree is a popular classification algorithm. This model on fitting the training set resembles a flow chart. Each step of a decision tree acts as a condition checker based on input characteristics and on series of checks, classifies the target
into the most appropriate category based on these checks. The advantage with a decision tree is the ability to visualize the tree using graphviz library in python.

![Figure 5.3: Example Decision Tree](image)

**Pseudocode 1 Decision Trees**

1: **Inputs:**
   
   $FeatureSet = f_i = (x_{1i}, x_{2i}, x_{3i}, ...)$
   
   $DebugSet = d_i = (d_1, d_2, ...)$
   
2: $features\_train \leftarrow train\_test\_split(f_i, 70, random)$
3: $features\_test \leftarrow train\_test\_split(f_i, 30, random)$
4: $DT \leftarrow DecisionTreeClassifier()$
5: $DT = DT.fit(features\_train, debug\_train)$
6: $Predictions \leftarrow DT.predict(features\_test)$

### 5.3.4 K Nearest Neighbors

In K Nearest Neighbors algorithm, the feature vectors are represented or plotted in the feature space, given a new vector, the classification depends on the average distance from the represented vectors.

### 5.3.5 Naive Bayes

Naive Bayes classifier is based on Bayes theorem. On training the model, conditional probability of the occurrence of target variable given the feature is computed. Baye’s theorem can be explained as follows:

$$P(A|B) = \frac{P(A)P(B|A)}{P(A)P(B|A) + P(A)P(B|\bar{A})}$$
**Pseudocode 2** \( K \) Nearest Neighbors

1: **Inputs:**
   
   \[ \text{FeatureSet} = f_i = (x_{1i}, x_{2i}, x_{3i}, ...) \]
   \[ \text{DebugSet} = d_i = (d_1, d_2, ...) \]

2: \[ \text{features} \_ \text{train} \leftarrow \text{train} \_ \text{test} \_ \text{split}(f, 70, \text{random}) \]

3: \[ \text{features} \_ \text{test} \leftarrow \text{train} \_ \text{test} \_ \text{split}(f, 30, \text{random}) \]

4: \[ \text{KNN} \leftarrow \text{KNeighboursClassifier}(5) \]

5: \[ \text{KNN} = \text{KNN.fit(features} \_ \text{train, debug} \_ \text{train}) \]

6: \[ \text{Predictions} \leftarrow \text{KNN.predict(features} \_ \text{test}) \]

where,

\( P \) is the probability, \( P(A) \) is probability of \( A \),

\( A \) and \( B \) are occurrences,

\( P(A|B) \) is the conditional probability of \( A \) given \( B \).

**Pseudocode 3** Naive Bayes

1: **Inputs:**
   
   \[ \text{FeatureSet} = f_i = (x_{1i}, x_{2i}, x_{3i}, ...) \]
   \[ \text{DebugSet} = d_i = (d_1, d_2, ...) \]

2: \[ \text{features} \_ \text{train} \leftarrow \text{train} \_ \text{test} \_ \text{split}(f, 70, \text{random}) \]

3: \[ \text{features} \_ \text{test} \leftarrow \text{train} \_ \text{test} \_ \text{split}(f, 30, \text{random}) \]

4: \[ \text{NB} \leftarrow \text{GaussianNB}() \]

5: \[ \text{NB} = \text{NB.fit(features} \_ \text{train, debug} \_ \text{train}) \]

6: \[ \text{Predictions} \leftarrow \text{NB.predict(features} \_ \text{test}) \]

5.3.6 **Neural Networks**

A neural network is an arrangement of nodes or neurons and unidirectional transitions associated with weights and biases. In a typical feed forward neural network, the neurons are arranged in a hierarchical manner starting with the input layer, hidden layers and an output layer. Hidden layers, provide neural network with most of the computational power. An activation function like sigmoid or relu (\( f(p) = \text{max}(0, p) \))
Chapter 5. Proposed Model

) is used to introduce non linearity and acts as a threshold for classification. In the figure 5.5 W refers to wight, O refers to the node, X refers to input and T refers to output.

![Figure 5.5: Example Multi layer Neural Network](image)

**Pseudocode 4** Neural Network MLP Regressor

1: **Inputs:**
   
   $FeatureSet = f_i = (x_1, x_2, x_3, ..)$
   
   $DebugSet = d_i = (d_1, d_2, ...)$

2: $features\_train \leftarrow train\_test\_split(f_i, 70, random)$

3: $features\_test \leftarrow train\_test\_split(f_i, 30, random)$

4: $NN \leftarrow MLPRegressor(hidden\_layer = (100, 100, ), activation = 'relu')$

5: $NN = NN.fit(features\_train, debug\_train)$

6: $Predictions \leftarrow NN.predict(features\_test)$
Chapter 6

Results

6.1 Research Findings

In this section, we present the experimental results with different machine learning algorithms and hold a discuss about them in analysis section. In order to understand prediction across all the labels, we calculate the average of all metrics as suggested by Xia et al. [16].

6.1.1 Decision Trees

On carrying out our experiments with Decision Trees algorithm with Eclipse dataset, we observe the following results presented in table 6.1.

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</td>
<td>0.09</td>
<td>0.10</td>
</tr>
<tr>
<td>1</td>
<td>0.11</td>
<td>0.15</td>
<td>0.13</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.18</td>
<td>0.13</td>
</tr>
<tr>
<td>3</td>
<td>0.11</td>
<td>0.24</td>
<td>0.15</td>
</tr>
<tr>
<td>4</td>
<td>0.10</td>
<td>0.11</td>
<td>0.11</td>
</tr>
<tr>
<td>5</td>
<td>0.15</td>
<td>0.03</td>
<td>0.05</td>
</tr>
<tr>
<td>6</td>
<td>0.16</td>
<td>0.08</td>
<td>0.10</td>
</tr>
<tr>
<td>7</td>
<td>0.13</td>
<td>0.02</td>
<td>0.04</td>
</tr>
<tr>
<td>8</td>
<td>0.11</td>
<td>0.09</td>
<td>0.10</td>
</tr>
</tbody>
</table>

Table 6.1: Results for Decision Trees

\[
AveragePrecision = \frac{\sum_{i=1}^{n} Precision_i}{n}
\]

\[
= \frac{0.10 + 0.11 + 0.10 + 0.11 + 0.10 + 0.15 + 0.16 + 0.13 + 0.11}{9} = 0.11
\]
Chapter 6. Results

AverageRecallrate = \frac{\sum_{i=1}^{n} Recall_i}{n} = \frac{0.09 + 0.15 + 0.18 + 0.24 + 0.11 + 0.03 + 0.08 + 0.02 + 0.09}{9} = 0.11

AverageF - Measure = \frac{\sum_{i=1}^{n} F\_Measure_i}{n} = \frac{0.10 + 0.13 + 0.13 + 0.15 + 0.11 + 0.05 + 0.10 + 0.04 + 0.10}{9} = 0.11

6.1.2 K Nearest Neighbors

On carrying out our experiments with K Nearest Neighbors algorithm with Eclipse dataset, we observe the following results presented in table 6.2.

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.10</td>
<td>0.23</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>0.12</td>
<td>0.28</td>
<td>0.17</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.15</td>
<td>0.12</td>
</tr>
<tr>
<td>3</td>
<td>0.09</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>4</td>
<td>0.06</td>
<td>0.02</td>
<td>0.03</td>
</tr>
<tr>
<td>5</td>
<td>0.16</td>
<td>0.06</td>
<td>0.03</td>
</tr>
<tr>
<td>6</td>
<td>0.10</td>
<td>0.11</td>
<td>0.09</td>
</tr>
<tr>
<td>7</td>
<td>0.10</td>
<td>0.08</td>
<td>0.11</td>
</tr>
<tr>
<td>8</td>
<td>0.14</td>
<td>0.02</td>
<td>0.03</td>
</tr>
</tbody>
</table>

Table 6.2: Results for K nearest Neighbors

AveragePrecision = \frac{\sum_{i=1}^{n} Precision_i}{n} = \frac{0.11 + 0.12 + 0.10 + 0.09 + 0.06 + 0.16 + 0.10 + 0.10 + 0.14}{9} = 0.11
Chapter 6. Results

\[
\text{Average Recall rate} = \frac{\sum_{i=1}^{n} \text{Recall}_i}{n}
\]

\[
= \frac{0.23 + 0.28 + 0.15 + 0.02 + 0.02 + 0.06 + 0.11 + 0.08 + 0.02}{9}
\]

\[
= 0.11
\]

\[
\text{Average F-Measure} = \frac{\sum_{i=1}^{n} F\_\text{Measure}_i}{n}
\]

\[
= \frac{0.15 + 0.17 + 0.12 + 0.03 + 0.03 + 0.09 + 0.11 + 0.09 + 0.03}{9}
\]

\[
= 0.09
\]

6.1.3 Naive Bayes

On carrying out our experiments with Naive Bayes algorithm with Eclipse dataset, we observe the following results presented in Figure ??.

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.07</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>1</td>
<td>0.13</td>
<td>0.08</td>
<td>0.10</td>
</tr>
<tr>
<td>2</td>
<td>0.11</td>
<td>0.18</td>
<td>0.14</td>
</tr>
<tr>
<td>3</td>
<td>0.11</td>
<td>0.69</td>
<td>0.19</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>0.03</td>
<td>0.00</td>
<td>0.01</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>0.05</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

Table 6.3: Results for Naive Bayes
Chapter 6. Results

Average Precision = \frac{\sum_{i=1}^{n} \text{Precision}_i}{n}

= \frac{0.07 + 0.13 + 0.11 + 0.11 + 0.00 + 0.03 + 0.00 + 0.00 + 0.05}{9}

= 0.05

Average Recall rate = \frac{\sum_{i=1}^{n} \text{Recall}_i}{n}

= \frac{0.01 + 0.08 + 0.18 + 0.69 + 0.00 + 0.00 + 0.00 + 0.00 + 0.01}{9}

= 0.11

Average F-Measure = \frac{\sum_{i=1}^{n} F \_ Measure_i}{n}

= \frac{0.01 + 0.10 + 0.14 + 0.19 + 0.00 + 0.01 + 0.00 + 0.00 + 0.01}{9}

= 0.05

6.1.4 Neural Networks

On carrying out our experiments with Neural Networks Multi Layer Perceptron (MLP) Classification with Eclipse dataset, we observe the following results presented in table 6.4. An MLP network has several layers of nodes and uses back propagation technique for training.

Average Precision = \frac{\sum_{i=1}^{n} \text{Precision}_i}{n}

= \frac{0.11 + 0.11 + 0.10 + 0.08 + 0.11 + 0.33 + 0.13 + 0.14 + 0.08}{9}

= 0.13

Average Recall rate = \frac{\sum_{i=1}^{n} \text{Recall}_i}{n}
Table 6.4: Results for Neural Networks

<table>
<thead>
<tr>
<th>Serial No.</th>
<th>Precision</th>
<th>Recall</th>
<th>F1 Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.11</td>
<td>0.15</td>
<td>0.13</td>
</tr>
<tr>
<td>1</td>
<td>0.11</td>
<td>0.20</td>
<td>0.14</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>3</td>
<td>0.08</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>4</td>
<td>0.11</td>
<td>0.14</td>
<td>0.12</td>
</tr>
<tr>
<td>5</td>
<td>0.33</td>
<td>0.01</td>
<td>0.02</td>
</tr>
<tr>
<td>6</td>
<td>0.13</td>
<td>0.07</td>
<td>0.09</td>
</tr>
<tr>
<td>7</td>
<td>0.14</td>
<td>0.09</td>
<td>0.11</td>
</tr>
<tr>
<td>8</td>
<td>0.08</td>
<td>0.16</td>
<td>0.11</td>
</tr>
</tbody>
</table>

\[
\text{Average F-Measure} = \frac{\sum_{i=1}^{n} F\_\text{Measure}_i}{n}
\]

\[
= \frac{0.15 + 0.20 + 0.08 + 0.06 + 0.14 + 0.01 + 0.07 + 0.09 + 0.16}{9}
= 0.11
\]

\[
= \frac{0.13 + 0.14 + 0.09 + 0.06 + 0.12 + 0.02 + 0.09 + 0.11 + 0.11}{9}
= 0.10
\]

6.1.5 Repeatability

In order to make our research study repeatable, the tool we used calculates precision, recall and f1-score for multiple observations. In our case, it was nine observations.
Chapter 6. Results

6.1.6 Statistical Analysis

In order to compare between the models, we use the two sided t-test with sample size of 9 and the parameter applied on is precision.

Null Hypothesis $H_0$: Sample one is accepted to as a better model
Alternate Hypothesis $H_1$: Sample one is rejected as a better model and sample two is accepted as a better model.

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
<th>T-statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neural Networks</td>
<td>K Nearest Neighbors</td>
<td>0.8532427039505345</td>
<td>0.4061174531035727</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>Decision Trees</td>
<td>0.49978312729406943</td>
<td>0.6240308878164376</td>
</tr>
<tr>
<td>Neural Networks</td>
<td>Naive Bayes</td>
<td>2.4805544223673723</td>
<td>0.024617798228843447</td>
</tr>
</tbody>
</table>

Table 6.5: T-test for Neural Networks

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
<th>T-statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Bayes</td>
<td>K Nearest Neighbors</td>
<td>-2.689592710466524</td>
<td>0.016111656409681034</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Decision Trees</td>
<td>-3.3514918826157714</td>
<td>0.004054514848094956</td>
</tr>
<tr>
<td>Naive Bayes</td>
<td>Neural Networks</td>
<td>-2.4805544223673723</td>
<td>0.024617798228843447</td>
</tr>
</tbody>
</table>

Table 6.6: T-test for Naive Bayes

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
<th>T-statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K Nearest Neighbors</td>
<td>Naive Bayes</td>
<td>2.689592710466524</td>
<td>0.016111656409681034</td>
</tr>
<tr>
<td>K Nearest Neighbors</td>
<td>Decision Trees</td>
<td>-0.817337896536993</td>
<td>0.4257394147188108</td>
</tr>
<tr>
<td>K Nearest Neighbors</td>
<td>Neural Networks</td>
<td>-0.8532427039505345</td>
<td>0.4061174531035727</td>
</tr>
</tbody>
</table>

Table 6.7: T-test for K Nearest Neighbors
### Chapter 6. Results

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
<th>T-statistic</th>
<th>P-Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decision Trees</td>
<td>Naive Bayes</td>
<td>3.3514918826157714</td>
<td>0.004054514848094956</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>K Nearest Neighbors</td>
<td>0.8173378965356993</td>
<td>0.4257394147188108</td>
</tr>
<tr>
<td>Decision Trees</td>
<td>Neural Networks</td>
<td>-0.49978312729406943</td>
<td>0.6240308878164376</td>
</tr>
</tbody>
</table>

Table 6.8: T-test for Decision Trees

#### 6.1.7 Threats to Validity

**Future Invalidation**

Most of the articles we refereed to and which in turn influenced our research work are conference articles and not many journal articles. Abandoning the methods using in these articles in the future is a risk we are willing to take.

**Generalization**

As our research work is very focused, generalization of these results may not hold true in all cases. Application of our candidate machine learning models in a more well formed data set with more accurate debug information might give better results.

**Internal Validity**

Performance of debug recommendation engine has direct implications from triaging the issue reports manually. Bug triaging needs expertise and reports can be easily mislabeled. In our research, we refereed to comments in ITS and labeled the debug information accordingly.

**External Validity**

Our experimentation model is designed to the specific application of debug information suggestion at Axis Communications and so, the results may not be generalized to other problems with Issue Tracking Systems at Axis. However, we experiment with BugZilla i.e. Eclipse dataset but with modifications including manual triaging of samples.
In this chapter, we will elaborate on the results from our experiments and attempt to answer our research questions.

On conducting experiments and analyzing the results we can observe that precision remained nearly same for trials with K nearest neighbors, Naive Bayes, Decision Trees and Neural Networks i.e., 0.11, F-Measure was slightly higher for Neural networks at 0.13 and by far makes a good candidate for debug predictions. Naive Bays on the other side had the lowest F-Measure 0.05 and Recall 0.05. Analyzing the results, it is clearly evident that machine learning isn’t a good fit for debug recommendation given the existing feature set i.e, product, Severity and Status which had lower p-values and target vector i.e, debug information extracted by manually triaging issue reports and analyzing comments. One reason for poor results might relate to poor quality data set for issue reports might have been triaged incorrectly as we are new to the nature of issues ana-
lyzed and suggestions in comments might not be a good indicator. However, it is recommended to collect debug information by experts by including a mandatory input field in Issue Tracking System and re run the experiment after 70,000 issue reports (Current yearly statistic).

7.0.1 Discussion

In this subsection, we correlate the results obtained and attempt to answer our research questions.

- **RQ1: What are the different factors that slow down the processes in a technical support line?**
  As a result of our literature review, we found that redundancy and noise are two important factors that slow down the issue resolution [9]. Furthermore, we also observed that assigning reports to an incorrect developer [28] leads to delays resolutions.

- **RQ2: Which components of Issue tracking systems can be improved or changed to offer faster services?**
  During my thesis project, on observing different Issue Tracking Systems assignment of issues, duplicate detection and debug suggestions are key parts of ITS that can be improved for faster services. It was observed by Kang [3] that it is possible to achieve speedup with duplicate detection. However, for debug recommendations it is observed from the current research that debug information data needs to be systematically collected for system to be improved.

- **RQ3: Can the existing Issue tracking systems be made faster by automating processes with recommendation and prediction algorithms?**
  From the experimental results, Figure 7.1, the highest precision achieved is 0.11 or 11 percent which is insufficient for being able to correct recommend debug information. Recommending incorrect debug information might result in loosing developer confidence. We suspect insufficient and poor quality data set as we might have manually triaged issue reports incorrectly being new to the nature of issue reports. Also the size our our data set is 7000 which is small in relation to the size of data set used in reviewed literature. Therefore, it doesn’t speed up the process now, but, that can change as a result of improved data collection.
Chapter 8

Conclusions and Future Work

8.1 Conclusion

During the course of the current thesis work, we started by understanding the research problem by searching the research databases and conducted a systematic Mapping Study. During the systematic review, we analyzed 33 research articles between 2002 - 2017 and summarized them in the related work section. We found redundancy and noise are the two mains concerns [9] need to be addressed to speedup the service support system, in addition to assignment of bug reports to right developers [28]. We have then formulated our research questions, proposed an approach with machine learning models decision trees, K Nearest Neighbors, Naive Bayes and Neural Networks combined with manual triaging for debug information, conducted experiments, extracted metrics and evaluated our solution using F-Measure, precision, recall on 7,000 issue reports. Using this approach, the precision of debug information found is 0.11, while the highest F-measure found was 0.13 using Neural Network which is not sufficient for being able to give recommendations as a service. We therefore suggest debug information included as a mandatory field in the Issue Tracking System, so that expert developers can tag every new issue report with debug data and re run our experiment.

8.2 Future Work

Following up on the current research work, we would like to know the results after re running our experiments after 70,000 new issue reports with debug information. During the literature review, we understood that automated reassignment of bug reports is a potential research area needing work [28]. It will be interesting to see results for similar approaches to classify assignees and classify core dumps generated on software crash.
References


References


References


References


### 8.3 Summary of related work

<table>
<thead>
<tr>
<th>Article</th>
<th>Contributions</th>
<th>Dataset used</th>
<th>Metrics used</th>
<th>Methodology</th>
<th>Research details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edmision and Edward [14]</td>
<td>Customized driver for GZOLTAR, WEBCAT for faster feedback to student programmers to reduce frustrating moments.</td>
<td>212 assignment programs for minesweeper and queue. 135 of them have bugs.</td>
<td>Experiments, quantitative data analytics.</td>
<td>Precision</td>
<td>SPECTRUM based fault localization. GZOLTAR for static code analysis and ranks every row between 0,1.</td>
</tr>
<tr>
<td>Vetro et al. [15]</td>
<td>Analyse software defect finding tool called FindBugs in the context of university projects</td>
<td>508 issues of 85 versions of lab assignments.</td>
<td>Experiments, quantitative data analytics.</td>
<td>2 of 15 groups can be considered as reliable. Metric used is Reliability.</td>
<td>FindBugs, static code analysis.</td>
</tr>
<tr>
<td>Authors</td>
<td>Methodology</td>
<td>Dataset Description</td>
<td>Evaluation Metrics</td>
<td>Machine Learning Methods</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------------</td>
<td>-----------------------------</td>
<td>------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>
### References

<table>
<thead>
<tr>
<th>Sujon et al. [19]</th>
<th>Find performance bugs with term frequency and Inverse document frequency</th>
<th>Bug reports and source code. Maybe internal reports and code.</th>
<th>Experiment</th>
<th>Precision 85 percent, recall 81-93%, 75%</th>
<th>Scan through the reported performance bugs from an Issue Tracking systems and doing a static code analysis of the underlying code. Naïve Bays classifier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pingclasi et al.</td>
<td>Classify bug reports with a rule based system</td>
<td>Leucene, Jackrabbit, HTTPclient. 7000 issue reports</td>
<td>Experiment</td>
<td>F-Measure 0.66-0.76 for HTTPclient</td>
<td>LDA. Latent Dirichlet Allocation. Decision trees, naïve bays , logistic regression.</td>
</tr>
<tr>
<td>Phan and Nguyen [20]</td>
<td>CNN on assembly code to predict software defects</td>
<td>Codechef, c,c++, python code for 4 different tasks</td>
<td>Experiment</td>
<td>F-Measure improvement by 10.94 percent</td>
<td>Instead of using sequences of programs to detect bugs, convert it into assembly and use them as feature set with abstract syntax trees instead. Multi view convolutional neural networks</td>
</tr>
</tbody>
</table>
### References

<table>
<thead>
<tr>
<th>Singh and Verma[21]</th>
<th>Investigate relationship between software metrics and defects</th>
<th>Metric data and bug data from SourceForge platform for different versions of iText.</th>
<th>Experiment</th>
<th>Accuracy, Precision, recall, f measure. Naive Bayes is better than decision trees.</th>
<th>Analyse the most popular Object Oriented Chidamber and Kemerer CK metrics. Fault prediction is done using J48 decision trees and bayes classifier with WEKA platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theisen et al. [22]</td>
<td>Risk Based Attack Surface Approximation (RASA) technique to scan the core dumps</td>
<td>Mozilla Firefox dataset with 50000 records</td>
<td>Experiment</td>
<td>NA</td>
<td>calculate risk in core dumps generated randomly by finding vulnerabilities and relate them to individual classes in code. They also attempt to scan the bug reports and connect them to the core dumps.</td>
</tr>
</tbody>
</table>
### References

<table>
<thead>
<tr>
<th>Author(s)</th>
<th>Method</th>
<th>Data Description</th>
<th>Experiment</th>
<th>Accuracy</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Otoom et al. [23]</td>
<td>severity prediction of software bugs</td>
<td>59 features and 163 bug reports from open source projects Bugzilla, Eclipse and Gnome.</td>
<td>Experiment</td>
<td>Accuracy</td>
<td>classify text with feature extraction, tokenisation (to remove stop words) and applying Naive Bayes (NB), RBF Networks, Functional Trees (FT), Random Trees (RT), Random Forests (RF) and AdaBoost algorithms</td>
</tr>
<tr>
<td>Kaushik et al. [24]</td>
<td>comparative study for duplicate bug detection</td>
<td>4330 bug reports from Eclipse’s Platform project and 9474 bug reports from Mozilla</td>
<td>Experiment</td>
<td>Recall Rate</td>
<td>word-based models, in particular a Log-Entropy based weighting scheme, compare to topic based ones such as LSI, LDA and Random Projections</td>
</tr>
<tr>
<td>Xuan et al. [9]</td>
<td>Text classification (Naive Bayes) for automated assignment of bugs</td>
<td>60000 issue reports</td>
<td>Experiment</td>
<td>F-measure and precision</td>
<td>binary classifier that predicts the order of instance selection, feature selection</td>
</tr>
</tbody>
</table>
References

<table>
<thead>
<tr>
<th>Lamkanfi and Demeyer [28]</th>
<th>Classify erroneous bug reports to predict their reassignment</th>
<th>4 projects from Eclipse and Mozilla</th>
<th>Experiment, simulation with WEKA</th>
<th>Analyse the frequency of change of product component field in the bug report and investigate the viability of a classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gou et al. [29]</td>
<td>predict the number of bugs of a future medical imaging software</td>
<td>Internal historical data about code changes, complexity</td>
<td>Experiments</td>
<td>Relative error of 1.426</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>static analysis and complexity metrics like change rate of software code with IBM SPSS Statistics</td>
</tr>
<tr>
<td>Seo and Kim [30]</td>
<td>method to predict recurring crashes</td>
<td>1159 stack traces</td>
<td>Experiments</td>
<td>precision of 0.57 with F-measure of 0.53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Extracting stack traces and comparing them to the bug fix locations</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Table 8.1: Summary of the relevant work</td>
</tr>
</tbody>
</table>

Python Scripts

Feature Selection

```python
import pandas as pd
import csv
import random
import numpy as np
import statsmodels.api as sm
from scipy import stats

input_data = pd.read_csv('data.csv')
dump_information = ['Crash Dump', 'Kernal Library', 'execution']
```

relevent_details = ['Product', 'Component', 'Hardware', 'OS', 'Classification', 'Status', 'Resolution', 'Severity', 'Number_of_Comments', 'Votes']

relevent = input_data[relevent_details]
relevent = relevent[relevent.Hardware != "All"]
relevent = relevent[relevent.OS != "All"]

relevent['Product_numerical'] = pd.factorize(relevent['Product'])[0]
relevent['Component_numerical'] = pd.factorize(relevent['Component'])[0]
relevent['Hardware_numerical'] = pd.factorize(relevent['Hardware'])[0]
relevent['OS_numerical'] = pd.factorize(relevent['OS'])[0]
relevent['Classification_numerical'] = pd.factorize(relevent['Classification'])[0]
relevent['Status_numerical'] = pd.factorize(relevent['Status'])[0]
relevent['Resolution_numerical'] = pd.factorize(relevent['Resolution'])[0]
relevent['Severity_numerical'] = pd.factorize(relevent['Severity'])[0]
relevent['Debug_numerical'] = pd.factorize(relevent['Debug_Information'])[0]

used_features = ['Product_numerical', 'Component_numerical', 'Hardware_numerical', 'OS_numerical', 'Classification_numerical', 'Status_numerical', 'Resolution_numerical', 'Severity_numerical', 'Number_of_Comments', 'Votes']

X = relevent[used_features].values
y = relevent['Debug_numerical'].values
X2 = sm.add_constant(X)
est = sm.OLS(y, X2)
est2 = est.fit()
print(est2.summary())

References

Decision Trees

import pandas as pd
import csv
import random
import numpy as np
import statsmodels.api as sm
from scipy import stats
import time
from sklearn import tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

input_data = pd.read_csv('data.csv')
debug_information = ['Crash Dump', 'Kernel Library execution',
                     'All sent messages', 'All responses',
                     'Configuration Parameters',
                     'Operation Language', 'Wireshark trace', 'Stack trace',
                     'Error code', 'Screenshot']

relevent_details = ['Product', 'Component', 'Hardware', 'OS', 'Classification', 'Status',
                    'Resolution', 'Severity', 'Number of Comments', 'Votes']

relevent = input_data[relevent_details]
relevent = relevent[relevent.Hardware != "All"]
relevent = relevent[relevent.OS != "All"]

relevent['Product_numerical'] = pd.factorize(relevent['Product'])[0]
relevent['Component_numerical'] = pd.factorize(relevent['Component'])[0]
relevent['Hardware_numerical'] = pd.factorize(relevent['Hardware'])[0]
Hardware'] [0]
relevent ['OS_numerical'] = pd.factorize(relevent ['OS']) [0]
relevent ['Classification_numerical'] = pd.factorize(
relevent ['Classification']) [0]
relevent ['Status_numerical'] = pd.factorize(relevent ['
Status']) [0]
relevent ['Resolution_numerical'] = pd.factorize(relevent ['
Resolution']) [0]
relevent ['Severity_numerical'] = pd.factorize(relevent ['
Severity']) [0]
relevent ['Debug_numerical'] = pd.factorize(relevent ['
Debug_Information']) [0]

used_features = ['Product_numerical', 'Component_numerical'
, 'Hardware_numerical',
'OS_numerical', 'Classification_numerical'
, 'Status_numerical',
'Resolution_numerical', '
Severity_numerical', 'Number_of_
Comments', 'Votes']

used_features = ['Product_numerical', 'Component_numerical'
, 'Hardware_numerical', 'OS_numerical']

features_train, features_test = train_test_split(relevent,
test_size=0.3, random_state=int(time.time()))

cclf = tree.DecisionTreeClassifier()
cclf = clf.fit(features_train[used_features].values,
features_train['Debug_numerical'].values)
predictions_test = clf.predict(features_test[used_features])

# print(debug_information[clf.predict([[product_types['JDT'
', component_types['UI'], hardware_types['PC'], ops_types
'other ']]]) [0]])
# print(predictions_test)
predictions_list = predictions_test.tolist()
print(classification_report(features_test['Debug_numerical'
], predictions_list))
#for prediction in predictions_test:
K Nearest Neighbors

```python
import pandas as pd
import csv
import random
import numpy as np
import statsmodels.api as sm
from scipy import stats
import time
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

input_data = pd.read_csv('data.csv')
default_information = ['Crash Dump', 'Kernel Library',
                      'Execution',
                      'All sent messages', 'All responses',
                      'Configuration Parameters',
                      'Operation Language', 'Wireshark trace',
                      'Stack trace', 'Error code',
                      'Screenshot']

relevant_details = ['Product', 'Component', 'Hardware', 'OS',
                     'Classification', 'Status',
                     'Resolution', 'Severity', 'Number of Comments', 'Votes']

relevant = input_data[relevant_details]
relevant = relevant[relevant.Hardware != "All"]
relevant = relevant[relevant.OS != "All"]

relevant ["Product Numerical"] = pd.factorize(relevant ["Product "])[0]
relevant ["Component Numerical"] = pd.factorize(relevant ["Component "])[0]
relevant ["Hardware Numerical"] = pd.factorize(relevant ["Hardware "])[0]
relevant ["OS Numerical"] = pd.factorize(relevant ["OS "])[0]
relevant ["Classification Numerical"] = pd.factorize(relevant ["Classification "])[0]
```
relevent['Status_numerical'] = pd.factorize(relevent['Status'])[0]
relevent['Resolution_numerical'] = pd.factorize(relevent['Resolution'])[0]
relevent['Severity_numerical'] = pd.factorize(relevent['Severity'])[0]
relevent['Debug_numerical'] = pd.factorize(relevent['Debug_Information'])[0]

used_features = ['Product_numerical', 'Component_numerical',
                 'Hardware_numerical',
                 'OS_numerical', 'Classification_numerical',
                 'Status_numerical',
                 'Resolution_numerical',
                 'Severity_numerical', 'Number_of_Commments', 'Votes']

used_features = ['Product_numerical', 'Component_numerical',
                 'Hardware_numerical', 'OS_numerical']

features_train, features_test = train_test_split(relevent,
                                                   test_size=0.3, random_state=int(time.time()))

clf = KNeighborsClassifier(n_neighbors=5)
clf = clf.fit(features_train[used_features].values,
              features_train['Debug_numerical'].values)

predictions_test = clf.predict(features_test[used_features])
predictions_list = predictions_test.tolist()
print(classification_report(features_test['Debug_numerical'], predictions_list))

print(features_test[used_features].values[0])
print(clf.predict([[features_test[used_features]].values[0]]))

Naive Bayes

import pandas as pd
import csv
import random
References

```python
import numpy as np
import statsmodels.api as sm
from scipy import stats
import time
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.naive_bayes import GaussianNB
import warnings
import sklearn.exceptions

input_data = pd.read_csv('data.csv')
debug_information = ['Crash Dump', 'Kernel Library execution',
                     'All sent messages', 'All responses',
                     'Configuration Parameters',
                     'Operation Language', 'Wireshark trace', 'Stack trace', 'Error code',
                     'Screenshot']

relevent_details = ['Product', 'Component', 'Hardware', 'OS',
                     'Classification', 'Status',
                     'Resolution', 'Severity', 'Number of Comments', 'Votes']

relevent = input_data[relevent_details]
relevent = relevent[relevent.Hardware != "All"]
relevent = relevent[relevent.OS != "All"]

relevent['Product_numerical'] = pd.factorize(relevent['Product'])[0]
relevent['Component_numerical'] = pd.factorize(relevent['Component'])[0]
relevent['Hardware_numerical'] = pd.factorize(relevent['Hardware'])[0]
relevent['OS_numerical'] = pd.factorize(relevent['OS'])[0]
relevent['Classification_numerical'] = pd.factorize(relevent['Classification'])[0]
relevent['Status_numerical'] = pd.factorize(relevent['Status'])[0]
relevent['Resolution_numerical'] = pd.factorize(relevent['Resolution'])[0]
relevent['Severity_numerical'] = pd.factorize(relevent['Severity'])[0]
```
References

```python
relevent['Debug_numerical'] = pd.factorize(relevent['Debug_Information'])[0]

used_features = ['Product_numerical', 'Component_numerical', 'Hardware_numerical', 'OS_numerical', 'Classification_numerical', 'Status_numerical', 'Resolution_numerical', 'Severity_numerical', 'Number_of_Comments', 'Votes']

features_train, features_test = train_test_split(relevent, test_size=0.3, random_state=int(time.time()))

clf = GaussianNB()
clf = clf.fit(features_train[used_features].values, features_train['Debug_numerical'].values)
predictions_test = clf.predict(features_test[used_features])
predictions_list = predictions_test.tolist()
# warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)
print(classification_report(features_test['Debug_numerical'], predictions_list))
```

Neural Network

```python
import pandas as pd
import csv
import random
import numpy as np
import statsmodels.api as sm
from scipy import stats
time
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,
```
confusion_matrix

import warnings
import sklearn.exceptions

input_data = pd.read_csv('data.csv')

data = [ 'Crash Dump', 'Kernel Library execution',
         'All sent messages', 'All responses',
         'Configuration Parameters',
         'Operation Language', 'Wireshark trace', 'Stack trace', 'Error code',
         'Screenshot']

relevent_details = [ 'Product', 'Component', 'Hardware', 'OS',
                    'Classification', 'Status',
                    'Resolution', 'Severity', 'Number of Comments', 'Votes']

relevent = input_data[relevent_details]
relevent = relevent[relevent.Hardware != "All"]
relevent = relevent[relevent.OS != "All"]

relevent[ 'Product_numerical'] = pd.factorize(relevent[ 'Product'])[0]
relevent[ 'Component_numerical'] = pd.factorize(relevent[ 'Component'])[0]
relevent[ 'Hardware_numerical'] = pd.factorize(relevent[ 'Hardware'])[0]
relevent[ 'OS_numerical'] = pd.factorize(relevent[ 'OS'])[0]
relevent[ 'Classification_numerical'] = pd.factorize(relevent[ 'Classification'])[0]
relevent[ 'Status_numerical'] = pd.factorize(relevent[ 'Status'])[0]
relevent[ 'Resolution_numerical'] = pd.factorize(relevent[ 'Resolution'])[0]
relevent[ 'Severity_numerical'] = pd.factorize(relevent[ 'Severity'])[0]
relevent[ 'Debug_numerical'] = pd.factorize(relevent[ 'Debug Information'])[0]

used_features = [ 'Product_numerical', 'Component_numerical',
                 'Hardware_numerical',
                 'OS_numerical', 'Classification_numerical']
References

used_features = ['Product_numerical', 'Component_numerical', 'Hardware_numerical', 'OS_numerical']

features_train, features_test = train_test_split(relevent, test_size=0.3, random_state=int(time.time()))

clf = MLPClassifier(hidden_layer_sizes=(13,13,13), max_iter=500)

clf = clf.fit(features_train[used_features].values, features_train['Debug_numerical'].values)

predictions_test = clf.predict(features_test[used_features])

#predictions_test = predictions_test.round(decimals=0, out=None)
predictions_list = predictions_test.tolist()

#warnings.filterwarnings("ignore", category=sklearn.exceptions.UndefinedMetricWarning)
print(classification_report(features_test['Debug_numerical'], predictions_list))