Improving Software Defect Assignment Accuracy

With the LSTM and Rule Engine Model

by

Robert Zhu, B.E., M.S., M.A.S.

Submitted in partial fulfillment
of the requirements for the degree of
Doctor of Professional Studies
in Computing

at

Seidenberg School of Computer Science and Information Systems

Pace University

2019

We hereby certify that this dissertation, submitted by Robert Zhu, satisfies the dissertation requirements for the degree of *Doctor of Professional Studies in Computing* and has been approved.

Lixin Tao
Lixin Tao (Oct 6, 2019)

Dr. Lixin Tao
Chairperson of Dissertation Committee

October 4, 2019
Date

Charles Tappert
Charles Tappert (Oct 6, 2019)

Dr. Charles Tappert
Dissertation Committee Member

October 4, 2019
Date

Juan Shan

Dr. Juan Shan
Dissertation Committee Member

October 4, 2019
Date

Seidenberg School of Computer Science and Information Systems
Pace University

Abstract

After a software defect is reported with a title and a text description, a competent developer needs to be assigned to fix it. The accuracy of this assignment has big impact on the quality of the resulting software, and the speed of the debugging process. Traditionally this software defect assignment process is conducted by product managers based on his/her knowledge of the software and the developers, which is not very scalable. In the recent years, this defect assignment problem has been formulated as a problem of (1) feature extraction from the defect title and description, and (2) classification of the resulting feature sets to the developers. Machine learning has been used to automate this software defect assignment problem.

The research improves the existing approaches in automatic defect assignment by (1) improving the feature extraction by NLP and Vector for Words technology, (2) introducing rule-based engine aka expert system to better character the strength of each developer, instead of the traditional characterizing a developer only by the descriptions of the bugs he/she has resolved; (3) combining the two layers model of our model (Layer 1, NLP and Vector for Words and Layer 2, Long Short-Term Memory and Rule-based Engine).

The optimal results are achieved on the CHROME dataset based on our new model of Long Short-Term Memory(LSTM) with Rule-based Engine in comparison with the traditional ML model - Naïve Bayes model.

The proposed neural network model extracts text features on its own, considering not only the word order messages that the word bag model ignores, but also the grammatical and semantic characteristics of the text. Rule-based Engine has absorbed developers' history data, and activity information.

The structure of these two layers network model with Rule-based Engine is relatively simple, i.e. the model is parallel structure, ideal for parallel computing, plus a dedicated hardware processing accelerator GPU makes the model not only high accuracy, but also faster.

The new approach that we introduced in the research shows that it has better accuracy than traditional Naïve Bayes model and pure LSTM model.

The new model can expand and migrate the system to generic bug assignment problems. The model is expandable and migratable.

Acknowledgements

First and foremost, I want to thank my wife, my daughters and parents for putting up with my time spent on this dissertation. I can still hear the complaints from them pointing out another weekend thrown away. More importantly I want to thank Dr. Lixin Tao. His patience and guidance were the light that shone my path. Also, all my friends and family who kept asking me when the day would come, well, it finally comes.

Trademarks

All terms mentioned in this dissertation that are known to be trademarks have been appropriately capitalized. However, the author cannot guarantee the accuracy of this information. A list of these trademarks is given below (It is not exhaustive):

TENSORFLOW is Registered Trademark of Google Inc.

CUDA and NVIDIA® GPU are Registered Trademarks of Nvidia Inc.

CHROME is Registered Trademark of Google Inc.

FIREFOX is Registered Trademark of Mozilla Foundation.

Table of Contents

List of Tables

Table of Figures

Chapter 1 Introduction

As the complexity of software continues to increase, the probability of software defects will increase exponentially. In order to ensure the quality of computer software and enhance the reliability and usability of software, software defects must be assigned accurately.

1.1 Research Background

The goal of software maintenance is to fix defects in the software or to develop new features for the software. Software defects in production software each year can cause billions of dollars in losses. At the same time many software companies have spent huge amount of money on software maintenance and software evolution. Therefore, it is necessary to pay more attention to the research and practice of software defect fix. Later part of the dissertation will use defect and bug interchangeably as in the software industry, they are the same.

Moreover, in large software development projects, developers use bug repositories to manage and perform normal software development. At the heart of the defect tracking system are software artifacts, such as defect reports, source code, and change history.

Those artifacts are important parts of the defect repairing task, because software developers in the project often use these software artifacts to manage and repair software defects.

In the maintenance of large software projects, bug reporting in software products is an important tool to help software developers to fix defects. Users, developers, software test engineers, program managers and others can create and fill out a software defect report with what they have found, and log them into bug database once they have discovered some defects in the software during the process of using or developing the software in order to facilitate software engineers to quickly verify and repair defects. In general, a complete defect report should consist of three parts: predefined fields like bug

title, owner fields, status, type of bug, priority, severity, release milestone, area path, resolution, fixes, and history, etc. The second part is the bug description field including repro steps, which is natural language text, and the third party is the related attachments including repro picture, videos, crash dump or trace files, and etc. The bug report's fields can be configurable and system administrators of the bug database can define what fields are needed to satisfy all the software engineering process, as shown in Figure 1 and Figure 2.

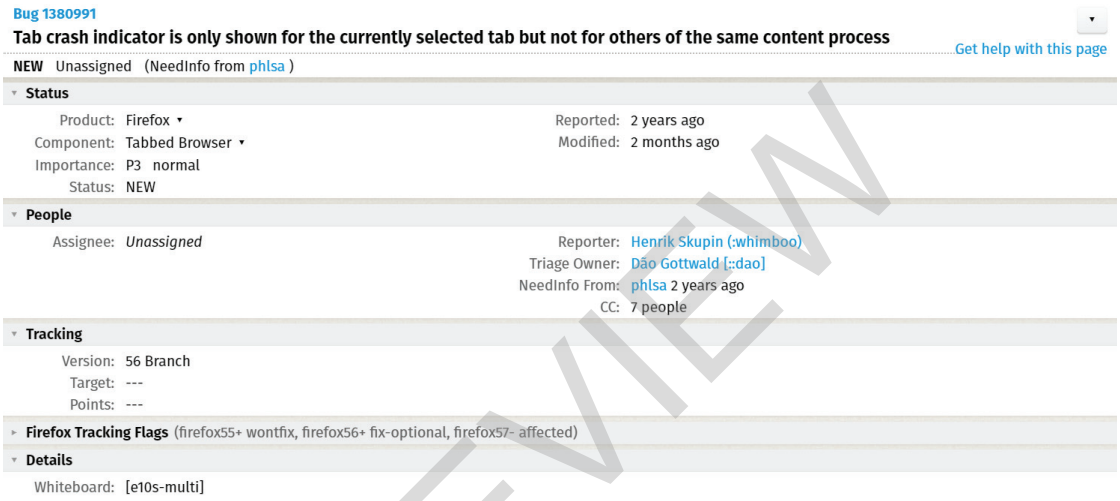

Figure 1 Bug Report Fields - Bug 1380991's fields in Firefox Browser project

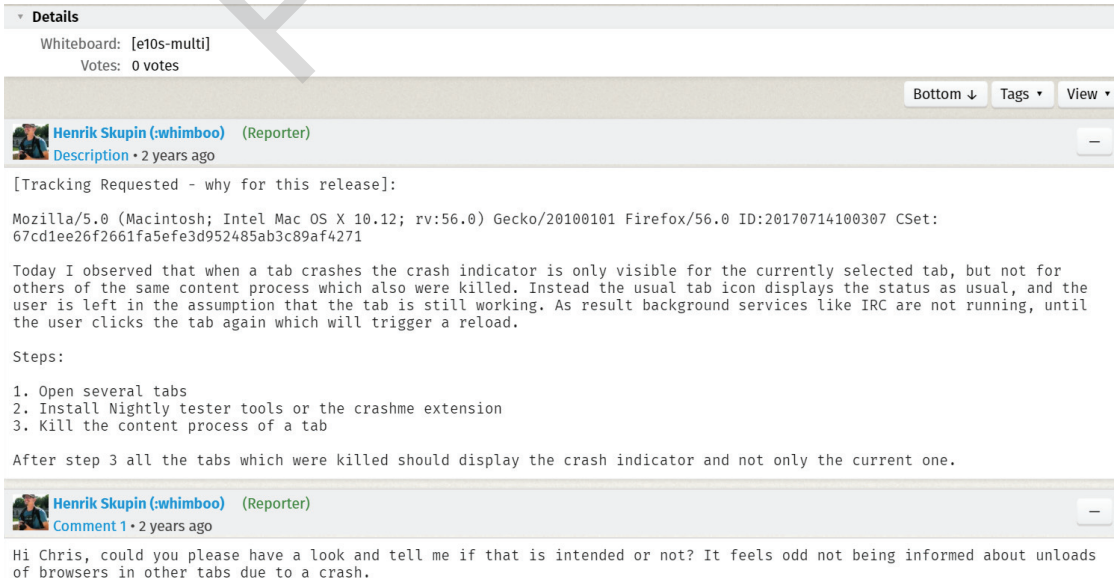

Figure 2 Bug Report Description - Bug 1380991 description text in Firefox Browser project

The predefined fields of the defect report are primarily metadata describing the defect report. For example, status as "resolved by fix" means that the bug has been resolved. Importance as “P0” means is very high and developers need to drop everything and start to work on the bug now. Assignee is whom the defect shall be assigned to, Triage Owner is the person who is charge of triaging the bug, and Reporter is the person who initially reported the bug.

All bug reports constitute the basic characteristics of the defect. The natural language text part can be divided into three parts: Summary, Description, and Comments. Summary is mainly to make a proper title for the defect content, so that others can browse and view it. Description will detail how to reproduce defects, as well as some basic analysis about defects. Comments are generally free discussions by relevant software developers on current bug, which are either long or short.

In general, these discussions are very helpful in fixing defects. In addition, software developers provide attachments, such as repro videos, crash dumps, pictures, test cases, and others.

As mentioned earlier, the bug fix process has a life cycle. It starts with bug creation, then to bug distribution, to bug fix and final resolution phase and bug close phase. The process is also reflected in the predefined field status of the defect report, as shown in Figure 3. After the reporter submits a new bug report, the bug reviewer, who usually is the program manager or product manager, in Microsoft for instance, will verify the bug. After the bug reviewer has verified the bug is not a duplicate bug, the bug reviewer will change the status of the bug “unconfirmed” to “new defect”. Next, the bug reviewer will assign it to the appropriate software developer for a fix based on the content of the defect report and the relevant developer information. After that, the bug status will be changed to “in progress” state.

If the bug is successfully fixed, the corresponding status in the bug report becomes “Resolved”. Finally, after the software test engineer verifies that the defect was successfully repaired, the corresponding

status in the bug report will become "closed". However, this does not mean the end of the life cycle of the bug.

After that, if it is found that the defect has not been completely repaired, for example a regression has been found, then the corresponding state will be changed to "active" state again. If the bug reviewer verifies the bug is a real bug, the bug will go to "new defect" state and repeat the above steps.

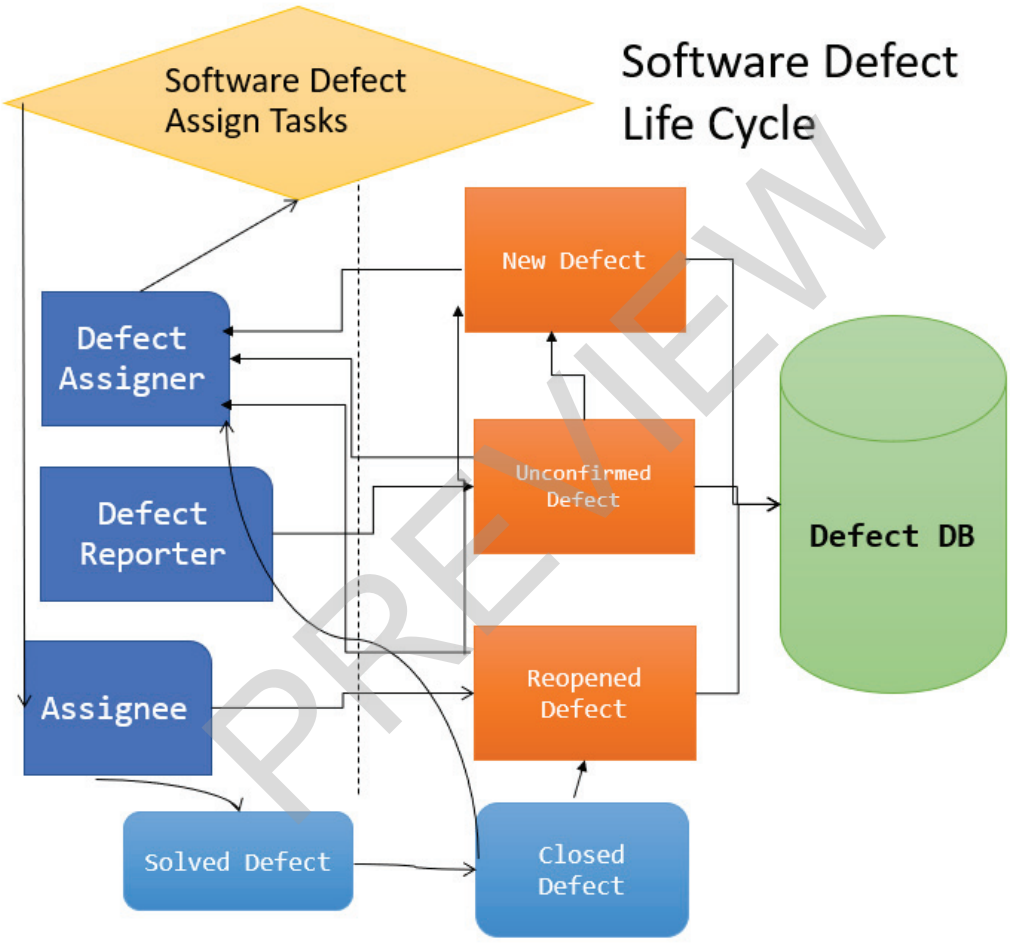

Figure 3 Software defect life cycle

From the perspective of the life cycle of bug management, the bug repair process is roughly divided into three stages: the bug understanding phase, the bug triage and assignment phase, and the bug fixing phase. When the bug report changes from the "unconfirmed" status to the "new defect" status, the bug

reviewer needs to fully understand the content of the bug report. By capturing important information of the bug, the bug reviewer can duplicate the bug to another bug, or simply add a bug hit count number to increase the bug's frequency or file a new bug report.

Some important feature fields (such as Priority, Severity, etc.) are ultimate relevant for future bug fix work, and the corresponding phase is called the bug understanding phase. After the bug is fully be understood, the bug will enter the triage and assignment stage.

In the triage and assignment phase, usually the software development manager, senior software engineers will look into the bug report. They will manually assign the bug to software engineers based on the bug reviewers' understanding of the report and the reviewers' empirical experience and knowledge on relevant developers' expertise. Many times, those tribal knowledges are not accurate and results in assigning the bug to the wrong software engineer and leads to delaying of bug fix.

The software developer who is assigned to the bug will perform the repair work based on the understanding of the bug report and his or her related experience. Then the bug goes into the bug fixing phase.

Finally, the bug fixing phase is further divided into two steps: the first step is to complete the root cause analysis of the bug aka, RCA work. The second step is to come up a solution and create a service pack or patch for the bug and complete the bug fixing life cycle.

1.2 Research Challenges

The dissertation is designed to improve the automatic bug assignment accuracy for large software projects. The improvement of the bug assignment accuracy and efficiency is of great significance to alleviate the burden of the bug assignment engineering staff and improving the quality of the software.

Traditionally managers will assign the defects to their developers and assign the defects to the components according to managers' experience. However, this method of manual assignment not only

consumes a lot of valuable time for managers, moreover, the accuracy of the assignment is bad. Later on, researchers started automatic defect assignment models which mostly gives a general empirical estimation formula through empirical analysis. Such prediction methods often have certain limitations, and their accuracy is not ideal. With the emergence of machine learning and data mining techniques, these theories or methods are gradually used in software defect assignment prediction. The way of defect assignment prediction has also evolved from the early empirical formula estimation to the prediction of defect assignment classic data mining science.

In order to improve the efficiency of defect assignment, it is necessary to fully understand the details of the bug report, understand the competency, capacity, and resources like tools and equipment of the relevant software developers have, and other necessary information like the environment and location he or she is at, for example, if the software engineer is trying to do a root cause analysis on a mobile phone bug with cellular issue in AT&T cellular network, but the engineer has no AT&T cellular signal in his or her location, so the bug reviewer needs to find a proper engineer with the access of the cellular network available to re-produce the bug. Those requirements are the criteria for the bug to be root cause analysis (RCA), and be properly fixed and verified.

From the above requirements, we can imagine we need to have a very experienced engineering manager and rich experienced engineers triage the bugs, while they have huge burden to triage the high volume of incoming bug reports.

Even so, the accuracy of assigning bug reports to the right developers is difficult to guarantee. The bugs are often like hot potatoes being passed around and are never earned a chance to be fixed. The study by Jeong et al. [1] shows that the more re-routing of the bugs, the less chance of the bug will be fixed. Sometimes, the bug has never found a right host to address its problem.

In order to solve the above problems, the researcher began to propose various methods and models for automatically assigning bugs, and they hoped to use automated methods to recommend bugs to the

appropriate software engineers, thereby reducing the work pressure of the bug reviewers and software engineering managers and solving the low efficiency and inaccuracy of bug assignment problems.

Undoubtedly, the existing models have achieved some reasonable and limited results on the automatic assignment of defects. It can also reduce the probability of multiple re-routing of bug assignments. However, some of the existing methods do not make full use of text information (such as word classification based on the word bag model), and some require a large amount of manual feature selection work (such as C4.5 based text classification method). Therefore, there is still a long way to go to achieve efficient, accurate and smart automatic bug assignments.

Firstly, the existing models are usually impossible to effectively deal with many mixed and irregular natural language texts in bug reports.

Secondly, the existing methods are often based on the word bag model and its transformation of words usually cannot effectively deal with word order and the models are too sparse in terms of word hashing.

Thirdly, the current traditional approach usually requires a large number of human interfered feature extraction or labor-intensive work on feature selections to achieve their bug assignment efficiency, which adds the burden of bug assignment engineering process.

Recently thanks to the rapid development of artificial intelligence especially in the area of deep learning neural network, the field of natural language processing has undergone tremendous breakthroughs. Deep learning has achieved remarkable results in many aspects of natural language processing (such as text modeling, text categorization, machine translation, etc.) and has gone beyond traditional methods.

Therefore, with the idea of deep neural network (DNN) in combination of rule-based engine research, the dissertation introduces the successful experience of deep learning in the field of text classification into the field of automatic software bug assignment to improve many shortcomings of the previous automatic bug assignment models. The rule-based engine on top of DNN further tunes the models.

With the hybrid of advanced deep learning neural networks with rule-based engine technology, the research actively explores how to make full use of the hybrid model to address the above shortcomings in order to achieve the high accuracy and high efficiency of smart automatic software bug assignments.

The key is to incorporate and rate the right candidate developers for bugs through rule-based engine. After applying DNN model at the first layer, the research uses the rule-based engine in the second layer of the hybrid model to evaluate the developer's experience and rate the developers. Then the hybrid model gives out its recommended developer to be chosen based on rule-engine ranker to further modify the DNN's preliminary suggested developer data.

1.3 Research Problem Statement

The research is aimed at improving the existing approaches in automatic defect assignment by (1) improving the feature extraction by NLP and Vector for Words technology, (2) introducing rule-based engine aka expert system to better character the strength of each developer, instead of the traditional characterizing a developer only by the descriptions of the bugs he/she has resolved; (3) combining two layers model (Layer 1, NLP and Vector for Words and Layer 2, LSTM and Rule-based Engine).

1.4 Research Methodology

How to improve the accuracy of automatic bug assignment process? Firstly, good training data must be collected. The researcher believes that the bug report can be used as the effective bug assignment basis for the bug assignment. It can be used as the training set. The text information in the bug report can provide insightful knowledge for bug assignment. The bug assignment prediction model learns these inside knowledge through training, the accuracy of bug assignment can be effectively improved.

How can we learn this insightful information that is crucial to the assignment of defects? The question raises a challenge for us to find deep learning model with rule-based engine of the defect text for the bug assignment model. The current work is still far from meeting this challenge. The research has already clarified this point in Section 1.2.

Therefore, this dissertation aims at the hybrid model of Layer 1: natural language processing (NLP) and Layer 2: deep learning neural network model, and rule engine method. The deep learning neural network is currently widely used in the field of natural language processing. It is expected to learn from the well-developed text processing ideas and model methods. The combination of deep learning neural network with rule-based engine is the new methodology to try it on bug assignment area. The hybrid approach is introduced into the field of automatic bug assignment problem space and is developed as a more accurate model to fit the real engineering bug practice. The hybrid model includes the layer 1 – vector for words, and layer 2 – LSTM and rule-based engine. The new model is shown in Fig 4.

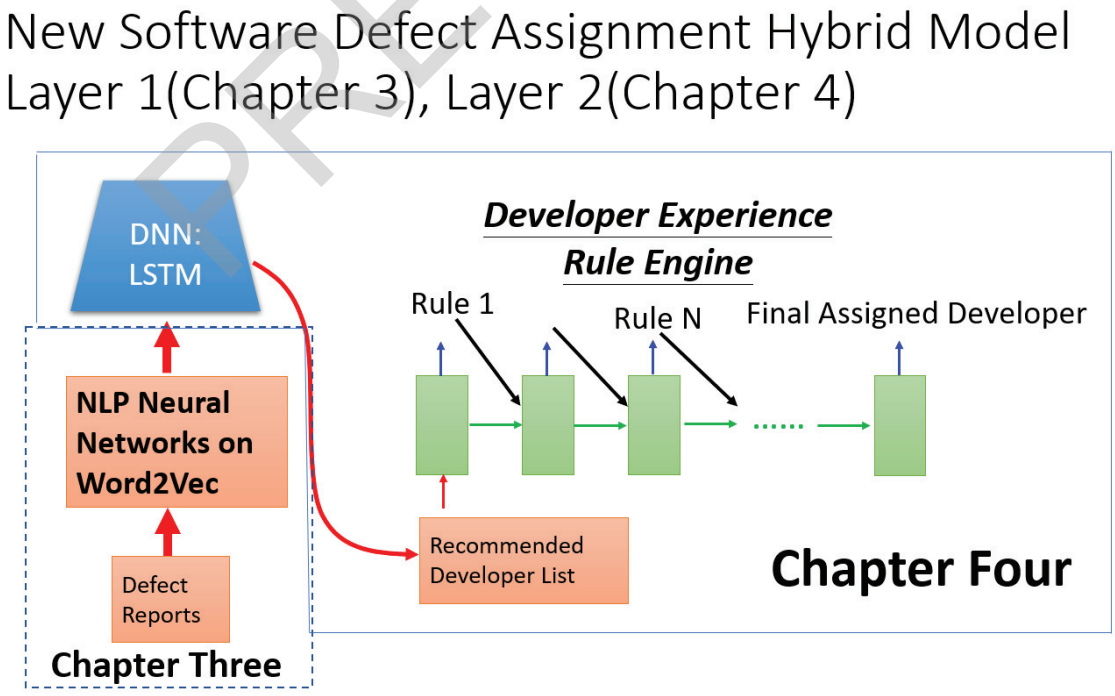

Figure 4 New Software Defect Assignment Hybrid Model

Researchers found that deep neural networks such as FFN, CNN, RNN, especially long-short-time memory (LSTM) proposed by Dyer et al. [6][7] LSTM has strong advantages in the field of natural language processing, especially in text classification.

Firstly, it has the natural advantage of dealing with local features selection. Moreover, this advantage is more prominent in the field of natural language processing, a lot of work [2][3][4][5] indicates that texts are based on deep learning models based on RNN, and it variant LSTM.

When classifying, the optimal effect can be achieved at that time. It can be also seen that DNN has a good ability to capture local text features. Moreover, LSTM operation can take full advantage of the text information, for example, the regional word order can be learned. LSTM can also have the ability to mine grammatical and semantic information, which cannot be achieved by the traditional word bag model. LSTM can automatically capture text features without any manual involvement, which can automatically assign the bugs to right owners.

Secondly, the hybrid model of NLP, LSTM with rule engine runs reasonable fast by using modern GPU, and the neuron weight and bias computation can not only reduce a large number of network model parameters, but also make the model have parameter sharing characteristics, which helps to quickly train the model.

Finally, there is also a special operation accelerator GPU to provide parallelized acceleration for neuron computation, SoftMax operations and cross entropy calculations in DNN in order to have the entire model built quickly and easily.

After reading a large number of references and researches, the dissertation is aimed at designing a deep learning model based on natural language processing (NLP), LSTM neural network and rule-based engine.

In the research, a dependency syntax analysis model to create word representations using shallow neural network [61] is created to get the word vectors. It reconstructs linguistic contexts of words. It is named as Word2vec and used to produce word embeddings. Word2vec uses the text corpus as input values to produce a vector space in serval hundred dimensions. Each individual word in the corpus is assigned a specific vector in the space. Word vectors are positioned in the vector space, so the words share common contexts in the corpus are located in close to each other in the vector space.

The accuracy is improved after adding Rule-based Engine on top of pure LSTM, for example, a bug with the real issue_title and description:

{

"id" : 8942,

"issue_id" : 68953,

"issue_title" : "Use after free in PepperPluginDelegateImpl::GetTextInputType on Mac OS",

"reported_time" : "2008-08-31 02:47:11",

"owner" : "",

"description": "\n[89950,1056861440:11:24:19.994000] Fatal error in file /Users/glider/src/chrome-commit/src/ppapi/native_client/src/shared/ppapi_proxy/ppb_rpc_client.cc, line 293: !(ppapi_proxy::PPBCoreInterface()->IsMainThrea\r\nd())\r\n[89950,1056861440:11:24: …EOF is received instead of response. Probably, the other side (usually, **nacl** module or **browser plugin**) **crashed**.\r\n[89947,2953392128:15:24:20.063623]…, freed by thread T0 here:\r\n #0 0x7365 in AsanThread::Init() (in Chromium Helper) + 229\r\n #1 0x964d15f9 in **free** (in libSystem.B.dylib) + 261\r\n, previously allocated by **thread** T0 here:\r\n … [89950,1056861440:11:24:19.995000] ReleaseResourceMultipleTimes: **PPAPI** calls are not supported off the main thread\r\nLOG_FATAL **abort** exit\r\n … "

}

Pure LSTM model assigns the bug's owner field with email alias: scherkus@chromium.org, a wrong developer. The reason of why we know it is wrong is because the pre-collected data has the right answers(email aliases) in the owner field in the training set. LSTM is a supervised learning, and the training model gives a confusion matrix, and it shows that record has the wrong owner prediction scherkus@chromium.org. But after the Rule-based engine, since there are many words including **nacl**, **browser, plugin**, **crashed**, **free**, **thread, PPAPI** and **abort** in the bug's description are matching the attributes aka keywords list of developer kinaba@chromium.org. According to a preconfigured threshold value on overlapping of the bug's title, description keywords with the developer's attributed keywords, the Rule-based engine on top of LSTM correctly assigns it to kinaba@chromium.org, a correct developer. With more records running through Rule-based engine, the accuracy of confusion matrix becomes higher and higher for the hybrid model.

The dissertation adds LSTM and rule-based engine with expert system and fuzzy logic basic idea to design a new bug assignment model. LSTM uses deep neural networks to perform all encoding on such transition states as stack, cache, history transfer sequence state, and current dependent subtree collection state. By this way, it makes full use of historical transfer information. More fine-grained modeling of words, stacks, caches, transfer sequences, etc. It has more flexible control of parameters tuning in a multitasking learning framework.

With a little more complexity by adding additional memory gates storages and introducing Rule-based engine, the hybrid of Layer 1 and Layer 2 model proves that it has better accuracy and generalization in the experiment to compare with traditional ones.

1.5 Dissertation Roadmap

The research is divided into six chapters, the content of each chapter is arranged as follows:

Chapter 1 Introduction The research introduces the background of the field of software bug fix engineering system and process. It states the main problem space of this dissertation is addressing, the automatic defects assignment related work, and briefly introduces the main researches on how deep learning neural networks and Rule-based engine aka expert system can solve the automatic defect assignment challenges.

Chapter 2 Software defect assignment traditional theory and approaches The traditional defect assignment models divides the current research work into four categories:

1) Model Based on Fuzzy Logic

2) Model based on Traditional Machine-learning

3) Expert System Model

4) Tossing-graph model

5) Social-network model

6) Topic-model.

Chapter 3 New Software Defect Assignment Model – Layer 1: NLP and Vector for Words The chapter introduces the methods of automatic software defect assignment based on text classification aka natural language processing. The method of defect is assignment fundamentally based on the text classification and how to create word vector dictionary. Generally, the defect report is the main feature to be used for the text classification and the developer acts as a label, and then the defect assignment problem is converted into the text classification problem.

Chapter 4 New Software Defect Assignment Model – Layer 2: LSTM and Rule-based Engine Inspired by LSTM, decision trees, and expert systems theories on software defect assignment, the chapter decided to apply LSTM, rule engine approach to the automatic of defect assignment research areas. The combination of text classification training on LSTM with developer productivity rule architecture is discussed for bug assignment project.

Chapter 5 Data Collection and Experiments Result Many comparative studies are presented. Through experiments, it is found that the hybrid model not only has a significant effect on text classification, but also has a higher accuracy rate than the traditional learning method for the problem of automatic defect assignment.

Chapter 6 Summary and Future Work This paper summarizes the work of the dissertation, explains the main contribution of the research in the field of defect assignment field, and future looks to the next step (like BERT, GPT 2 system) based on the current research topics.