PREDICTING BUG SEVERITY IN OPEN-SOURCE SOFTWARE SYSTEMS USING SCALABLE MACHINE LEARNING TECHNIQUES

By

IMRAN

Submitted in Partial Fulfillment of the Requirements

For the Degree of

Master of Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

May, 2016

PREDICTING BUG SEVERITY IN OPEN-SOURCE SOFTWARE SYSTEMS USING SCALABLE MACHINE LEARNING TECHNIQUES

Imran

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

Imran, Student Date

Approvals:

Dr. Alina Lazar, Thesis Advisor Date

Dr. Bonita Sharif, Committee Member Date

Dr. John Sullins, Committee Member Date

Sal Sanders, Associate Dean of Graduate Studies Date

DEDICATION

It is with my deepest gratitude and warmest affection that I dedicate this thesis to my parents, professors and friends who have encouraged me.

ABSTRACT

As software systems become larger and more complicated, the task of detecting and fixing bugs to improve the software performance is getting more tedious and inefficient. Automated processes that detect and report bugs quickly and with high accuracy are needed. In this thesis, we describe an approach, which is fast and performs the bug classification task with comparatively better accuracy then previously reported research. Here, we used the machine learning methods, specifically an online algorithm for bug classification. This approach involves the use of text mining algorithm for feature extraction. Then the data is used to train classifier models using an online machine learning classification algorithm for optimized performance. The above steps are done twice, once for a binary model and once with a multi-class model. The multi-class model predicts as many as seven bug severity levels with the aim of prioritizing the bug assignment process. After analyzing all four datasets collected from open source software system, we can predict good with accuracy (72%-98%) if the data is balanced and has sufficient size of training set.

ACKNOWLEDGEMENTS

I would like to thank Dr. Alina Lazar for giving me this opportunity to work on this challenging research project of predicting bug severity in open-source software systems using scalable machine learning techniques and helping me to complete this thesis successfully. She has provided me tremendous encouragement and knowledge of how to approach a challenging problem by thinking beyond the box. I could not have had a better advisor for my master.

I thank my committee members Dr. John R Sullins and Dr. Bonita Sharif for their support and guidelines.

I thank my family for their love and encouragement. I thank all the members who directly or indirectly helped me to complete my research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Software improvement involves considerable amount of resources. The task of bug tracking is by and large a manual one and affects the cost by billions of dollars each year of the software industry. Moreover, the task is resource intensive, time consuming and error prone, which result in high maintenance costs of the software and leads to decreased productivity. After a bug is submitted, a human triager manually assigns a priority and a severity each bug reported by a user or a programmer. Priority can be low, medium and high while severity can be critical, major, moderate, minor, and cosmetic. The task is much more prone to faults and involves the working of expert triagers. This further pushes the maintenance cost of the software [1]. Therefore, depending on the complexity of the software, a bug tracking system like Bugzilla is required by users and developers to automatically track and classify bugs. Bug tracking systems help in continuously monitoring, reported bugs. It helps developers and users in storing and tracking errors and aids them in understanding problems caused by these errors [2]. The data we are using during our research comes from open software systems like Mozilla and Eclipse. Open source software systems are different from the ones developed and own by any company privately. In open source softwares, developers implement software systems which are publicly accessible at no cost [3].

Machine leaning techniques can provide just the right kind of algorithms to automate the task of bug prediction and classification. To further enhance the efficiency

of a bug classifier, feature extraction techniques could be employed to feed refined information to the classifier. This can be achieved with text mining algorithms that can extract important keywords from different bug reports. Text mining have previously been used for picking up high importance words from the bug reports, and these words along with the original bug report can be effectively used further to improve the overall bug classification accuracy of a classifier. In this research, we are aiming to investigate the bug classification accuracy of an online machine learning classifiers. We further use a keyword extraction algorithm for extracting important tags and then couple them with the original bug report to observe whether we can obtain marked improvement in the prediction accuracy of the classifier.

Next, we explore the accuracy of online machine learning algorithm multi-class model for classifying bugs into seven different levels of severity: trivial, normal, minor, major, enhancement, critical and blocker. We extend our work to include keyword extraction algorithm for tag extraction to analyze its effects on the prediction accuracy of multi-class classifier. Even though our approach of automatic categorization does not achieve perfect accuracy, it shows noticeable improvement in the prediction accuracy and hence promises improved performance of the software maintenance tasks.

CHAPTER 2

DATASET USED

The datasets used for this research were extracted from open source software systems such as Eclipse, Gnome, KDE and Mozall. Bug reports from the Bugzilla websites of the four systems were colleted using web scraping. [4, 5].

Each of the four datasets that we are using in this research have more then 100,000 records and 13 features. The features present in the raw datasets are bug id, product, description, bug severity, dup id, short description, priority, version, component, delta_ts, bug status, creation_ts and resolution. The most important feature to predict bug severity is the long description. The bug severity can take values in one of the following categories: trivial, normal, minor, major, enhancement, critical and blocker. For the binary classification task only two categories the bug severities were divided into two categories: severe and non-severe. The trivial, normal, and minor were considered under non-severe category and major, enhancement, critical, and blocker were considered under severe category. The two categories are denoted by zero and one. Non-severe is replaced by zero and severe is replaced by one in the data.

Table 1, it shows the number of instances in each of the four datasets Eclipse, Gnome, KDE and Mozilla used in our research. Each dataset has more than 279,000 instances. The dataset with the highest number of instances is Mozilla with 768,335 instances.

Table 1. Number of Instances in each Dataset

Dataset	Number of Instances
Eclipse	361,006
Gnome	327,574
KDE	279,843
Mozilla	768,335

Table 2 shows the number of instances in two categories: non-severe and severe. The number of non-severe instances is higher in each dataset as compare to the number of severe instances. Percentage of severe instances in Eclipse, Gnome, KDE and Mozilla are as follow 15.07%, 43.09%, 48.91% and 19.30%.

Table 2. Number of non-severe and severe instances in Binary datasets

Dataset	Non-severe	Severe
Eclipse	306,571	54,435
Gnome	186,452	141,152
KDE	142,955	136,888
Mozilla	619,980	148,355

Table 3 reports all the seven-bug severities used for multi-class classification. This table shows the number of trivial, normal, minor, major, enhancement, critical and blocker number of instances in each dataset.

Table 3. Number of Trivial, Normal, Minor, Major, Enhancement, Critical and Blocker in multiclass datasets

Dataset	Trivial	Normal	Minor	Major	Enhancement	Critical	Blocker
Eclipse	4,815	242,819	13,438	35,367	45,499	12,845	6,223
Gnome	5,337	137,818	16,584	17,968	26,683	116,986	6,198
KDE	0	138,087	4,868	3,727	0	1,449	0
Mozilla	16,134	523,540	36,617	71,084	43,689	65,478	11,793

CHAPTER 3

RELATED WORK

A number of cutting edge researches have been carried out in the field of bug classification. The focus has mainly been on assigning priority and severity.

3.1 Assigning severity levels automatically

The first research in the field of bug classification was done by Marcus and Menzis [4]. They built an automated method, which helps the software engineer to assign different severity levels to the reported bug reports by user and developer. They called this method SEVERIS (Severity issue assesment). SEVERIS is a combination of machine learning (Naïve Bayes) and text mining techniques applied to bug reports. Their approach is able to classify the output into five fine-grained severity levels with accuracy that very from 65% to 98%.

3.2 Assigning severity levels using text mining algorithm

Lamkanfi et al. [7] extended the work of Marcus and Menzis They worked to improve the performance of prediction. They were the first to predict the severity of bug reports from various projects of open source repositories such as Mozilla, Eclipse and Gnome. They basically coarse grained the five of the six severity levels of Bugzilla into two severe and non-severe and one is omitted. The severe group includes blocker, critical and major while minor and trivial are grouped under non severe. They found in their research that by giving sufficient training data, we could predict severity of a reported bug with good accuracy. Accuracy in terms of precision and recall varies from

65% - 75% and recall varies from 70% - 85%. Lamkanfi further expand his work by comparing machine learning algorithms (Naïve Bayes, Naïve Bayes multinomial, K-Nearest Neighbor and Support Vector Machines) to predict the severity of bug reports [1]. InFor the Eclipse and Gnome open source systems, Lamkanfi showed that the Naïve Bayes multinomial algorithm performs better than the other approachs. Their reported accuracy varies from 48% - 93%

3.3 Severity prediction using nearest neighbor classification

Apart from the above mentioned works, Tian et al. [5] presented an approach to predict fine grained bug severity prediction using nearest neighbor classification. The method they used automatically detects and analyzes bug reports, which had been reported in past days with severity labels, and with the help of these reports suggests severity labels to new reported bug reports. They used duplicate bug reports with relative information and features to determine similarity between both reports. This similarity in information helps in assigning the severity labels accurately and quickly.

3.4 Suggesting priority level by considering multifactor

Other work of Tian et al. [6] presented an automated approach with the help of machine learning in suggesting a priority level on the basis of information in bug reports. They consider multifactor temporal, textual, author, related reports, product and severity, as potential factors, which affect the priority level of reported bug reports. They use these factors as features to train a model with the help of a classification algorithm (thresholding and linear regression), which can perform well in ordinal class labels and

imbalanced data. They conducted their experiment on more than 100,000 reports collected from Eclipse. This experiment shows an improvement of 58.61% in terms of average F-measure by outperforming baseline approach. Tian et al. [7] extended their previous work by using extracted features to train a discriminative model via a new classification algorithm (linear regression) and their framework named DRONE. The new work provides a way to handle ordinal class labels and imbalanced data. They managed to improve their work on 100,000 bug reports from Eclipse in terms of F-measure by 209%, which outperform baseline approach.

3.5 Detection of bug report using textual similarity features

In addition to this, Lazar et al. [8] presented an approach, in which they implement an improved method of detecting duplicate bug reports using the textual similarity features and binary classification. They used a total of 25 textual features, then run their classification method to categorize pairs of bugs into two types: duplicate and non-duplicate. They used new texual features, derived based on text similarity measures, and trained several binary classification models. After training the models, they tested their work on bug reports collected from Eclipse, OpenOffice and Mozilla to analyze the effectiveness of the improved method. They also compared it with the current state-of-the-art and highlighted the similarities and differences. They were able to achieve an improvement of 6.32% in duplicate bug report detection even without considering context-based features.

CHAPTER 4

METHOD

Under this section we describe the method step-by-step. Firstly, we discuss the tools and algorithms. Next we describe how the features were generated. Then, we explain how we divided the dataset into training and testing. In the end, we give a brief idea of the performance measures used to analyze and compare the experiments.

4.1 Machine learning

Machine Learning is a technique of driving intelligence from the data. Machine learning models keep evolving in order to make data driven predictions. Online Machine learning is a more sophisticated technique as it allows models to dynamically update themselves based on the real time data. In this research, we use one of the online machine learning package named Vowpal Wabbit (VW) [9]. This online machine-learning algorithm can handle different kind of problems and also different loss functions. These algorithms can make several iterations of data in less time. On the first run, it saves the data in a cache file, which can be quickly retrieved for making several of the following passes while training a model, hence making the entire training process very fast. It further speed up its process by two methods, i.e. it uses a single floating algorithm and employs parallel processing using two threads for training a model (loading data from the compressed files and updating the model).

4.2 Stochastic gradient descent (SGD)

The VW, SGD implementation is one of the fastest online gradient optimization algorithm through which we can optimize processing large datasets. In other words, the algorithm minimizes the loss function or the objective function [10]. SGD doesn't compute the gradient exactly as other algorithms do. However, it randomly picks examples to estimate the gradient in all iteration. SGD does not need to remember its previous iteration example, because of its randomly drawn example nature. Bottou applied SGD to different variables by enforcing its positivity to help obtain sparser solutions [11].

4.3 Keyword extraction

If you are working with text, keyword extraction or tag extraction is an important topic. Keyword extraction can be very usefull for index or glossary creation, summarizing and word cloud creation. For this task, we are using the keyword extraction library named RAKE [12]. This library is a keyword extraction algorithm implemented in python. Keywords are of primary importance when we are discussing about texts. Through keywords we can build our understanding of the topic expressed in the document. Usually, the keyword extraction algorithm is a three-step algorithm. It includes candidate selection, properties calculation and scoring, and keywords selection. In candidate selection, we extract all those words and phrases, which can be keywords. Property calculator calculates a property score for each keyword candidate. To select the final keyword, each candidate is scored by combining properties into a formula or by machine learning technique. The keyword extraction algorithm helps in automatically

getting the most significant words and phrases from the documents, which are under analysis.

4.4 Generating features

We downloaded four different datasets of bug reports from open source repositories. Each dataset Includes the following thirteen features: bug_id, product, description, bug_severity, dup_id, short_desc, priority, version, component, delta_ts, bug_status, creation_ts and resolution. In both, binary and multi-class, classifications we generated four new features named as tag1, tag2, tag3 and tag4. We are using the keyword extraction algorithm to generate aforementioned four features.

4.5 Building the training and testing datasets

We are using a script written in python to divide the datasets into training and test sets. We divide the data into a train and test datasets in the ratio 9:1. The datasets are then converted from the cvs format into the online machine learning format. The data is then scaled down into the [0,1] interval. We assigned the value zero to non-severe bugs and value one to severe in case of binary classification.

4.6 Measures

Usually, in most research about classification methods, accuracy is considered as the main measure to express performance. There are other measures, which can be used in addition to accuracy, especially when we are talking about highly imbalanced data. In order to obtain the holistic performance view, we give equal weightage to the following performance measures: precision, recall, f-measure and ROC curve. We use Perf [13] to

calculate performance measures in binary classification. It is a tool that calculates performance measures. We also used the scikit-learn kit [14] for multi-class classification. These tools are customized for calculating machine learning performance metrics. They can help in calculating different performance metrics for binary classification and multi-class classification problems. Perf and scikit-learn include different measures, out of which we used accuracy, precision, recall, F-score and area under the ROC curve. Following are the definitions and formula of the precision, recall and F-score measures:

Precision (P): The percentage bugs report correctly predicted either severe or non-severe. Then we consider precision for each severe and non-severe category separately. It is defined it:

$$\text{Precision} = \frac{\text{tp}}{\text{tp} + \text{fp}} \quad (1)$$

Where tp stands for true positive and fp stands for false positives. Moreover, in our case true positive is correctly classified instances and false positive is incorrectly classified instances.

Recall (R): The percentage of all bug reports with a severity severe and non-severe that is correctly predicted as severe or non-severe called recall. It can be formally defined as:

$$\text{Recall} = \frac{\text{tp}}{\text{tp} + \text{fn}} \quad (2)$$

Here fn stands for false negative. In our case false negative is incorrectly classified instances.

F-measure (F): The weighted harmonic mean of precision and recall is called F-measure. The harmonic mean is a conservative average. Usually the balanced F-measure is used. It is defined as follows:

$$\text{F-measure} = 2^{*}\frac{\text{presision} * \text{recall}}{\text{presision} + \text{recall}}$$

For positive real β

$$\text{f-measure} = \frac{(\beta^2+1)\text{precision.recall}}{\beta^2\text{precision}+\text{recall}} \quad (3)$$

CHAPTER 5

EXPERIMENT

We divide this work into two steps. First we predict the bug severity without tagsor keywords. Second we predict bug severity with tags.

5.1 Binary Classification

We perform our first experiment on binary classification. In binary classification, we calculate accuracy, precision, recall and plot ROC curve for the two different cases. First the initial dataset is used and second the tags are added to the dataset.

5.1.1 Binary classification without tag data

In this research, we follow an approach, which states that the words used in any bug report are significant and informative enough to describe it. Tailoring the above approach to this research, we can classify a bug, based on its short and long description, as severe or non-severe.

Data was downloed from from different open source systems, namely: Eclipse, GNOME, KDE and Mozilla. Thirteen data features were selected from the bug reports. Datasets in csv format were converted into an online machine-learning format using a conversion script written in python. The bug severity field contains seven different severity levels. The script also changes the seven severity levels, in the bug severity field, into two categories, severe and non-severe. *Trivial, normal, minor* are grouped under the non-severe category and *major, enhancement, critical, blocker* are assigned to the severe category [7]. We use another script written in python to divide the dataset into training

and test datasets using a ratio of 9:1. The train dataset is then used to train the online machine learning binary classification model. The classifcation model is then deployed to make predictions on the test dataset. Prediction outputs along with the original bug severity values are stored in an output text file. Performance metrics are calculated using the perf tool on the output text file.

Performance measures, including precision, recall and F-measures are plotted using Numpy and Matplotlib python libraries. Area under the curve (ROC) compares the rate of true positives with the rate of false positives and indicates how good a classifier performance is.

The overallAccuracy obtained is better than the accuracy reported by previous works in the same field. Table 4, shows that accuracy is varying from 82% to 98% for the binary classification model. Highest accuracy is obtained for the GNOME dataset (98.025%), while the lowest accuracy is 82.951% for the KDE dataset.

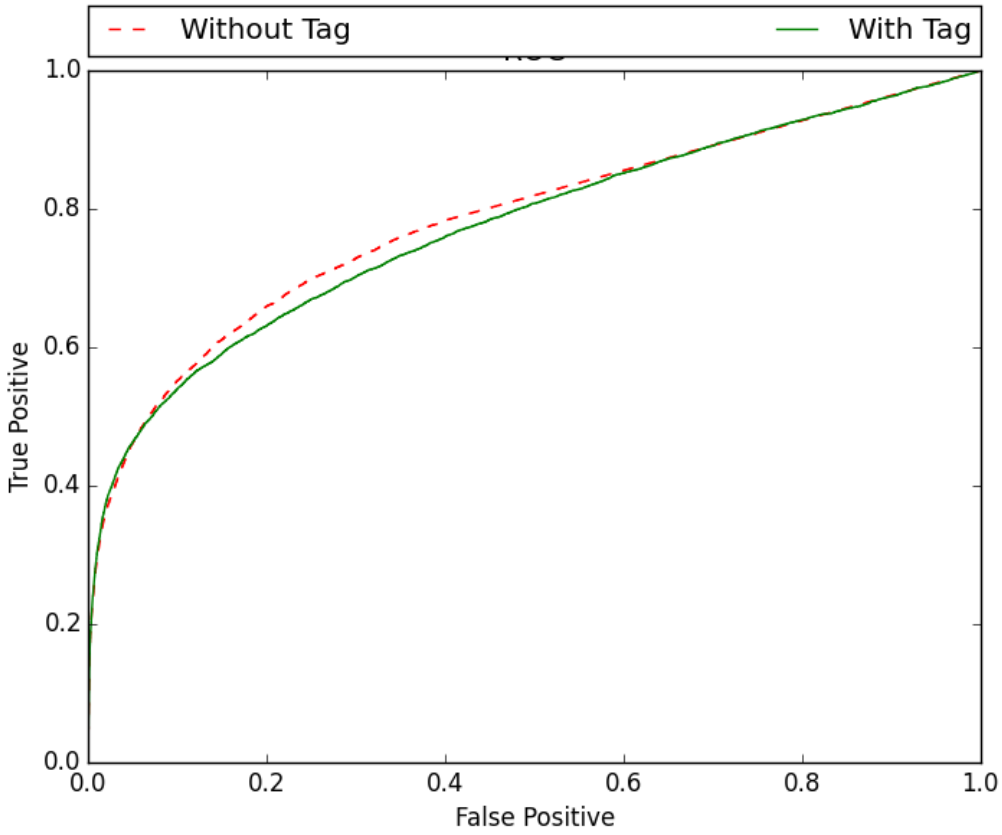


Figure 1. ROC curve for binary classification without tag and with tag for the Moziilla dataset

In Figure 1, Figure 2, Figure 3 and Figure 4, we can see two lines, one in green and another one in red. The green line represents the graph related to the result for the dataset with tags while red is describing the results for the dataset that did not include tags. This graph is plotted between true positive and false negative. True positive is correctly classified instances while falsely positive represent incorrectly classified instances. If the curve is closer to the left border and the top border, the result is more

accurate. If the curve is closer to 45-degree diagonal on the graph, the result are considered less accurate.

In Figure 1 we can see that the results for the dataset without tags are closer to the left border and the top border, which means, the results results are sligthly better as compared to the dataset with tag.

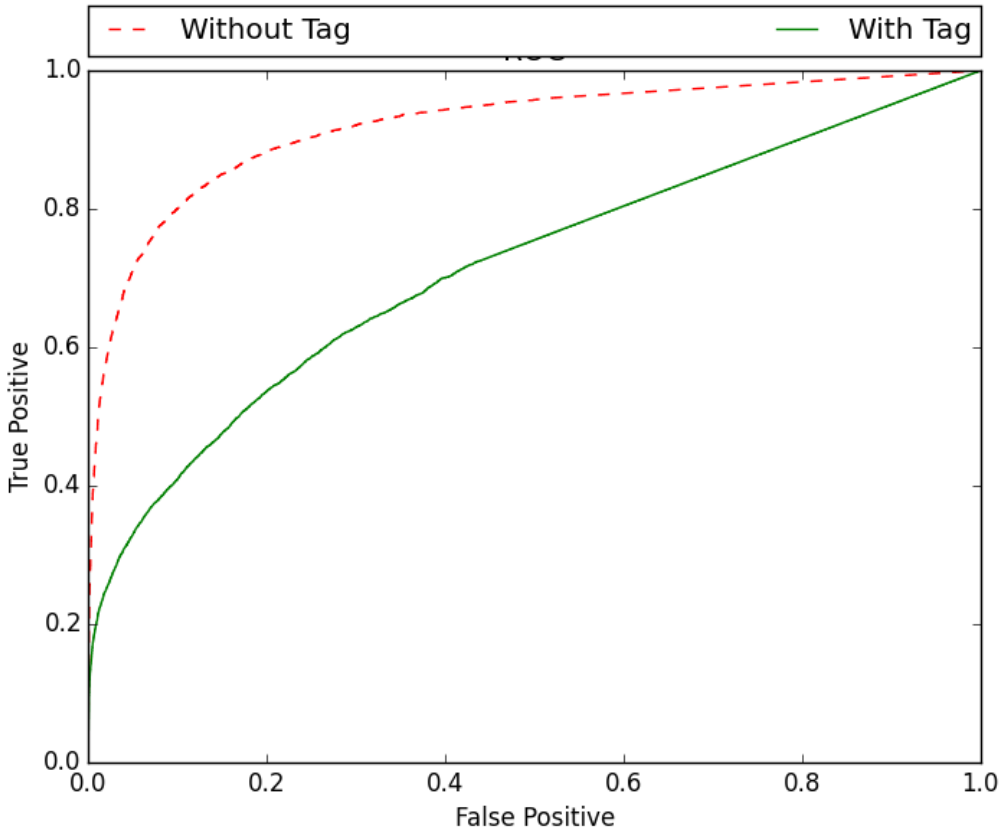


Figure 2. ROC curve for binary classification without tag and with tag for the Eclipse dataset

Figure 2 shows that we have significantly better result for without tag data instead of with tag data.

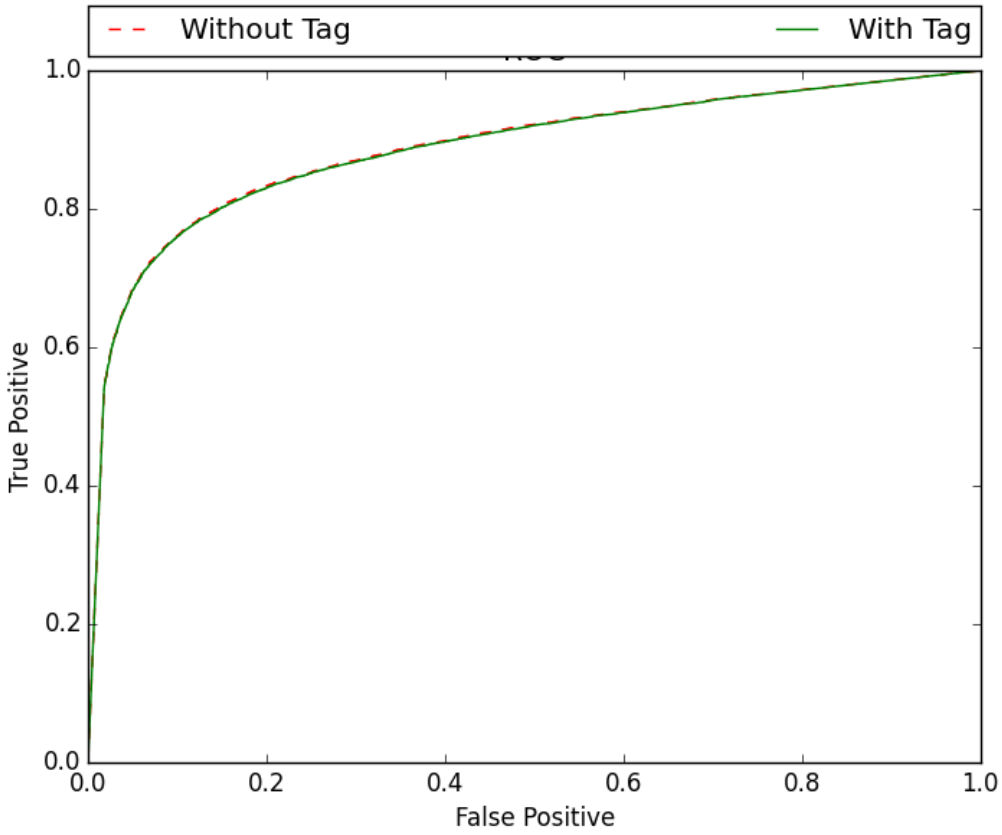


Figure 3. ROC curve for binary classification without tag and with tag for the KDE dataset

Figure 3 has a much better result in terms of the data, which has the tag. If we compare it to previous figures, we can find that the data with tags, has the same curve as the data without tag.

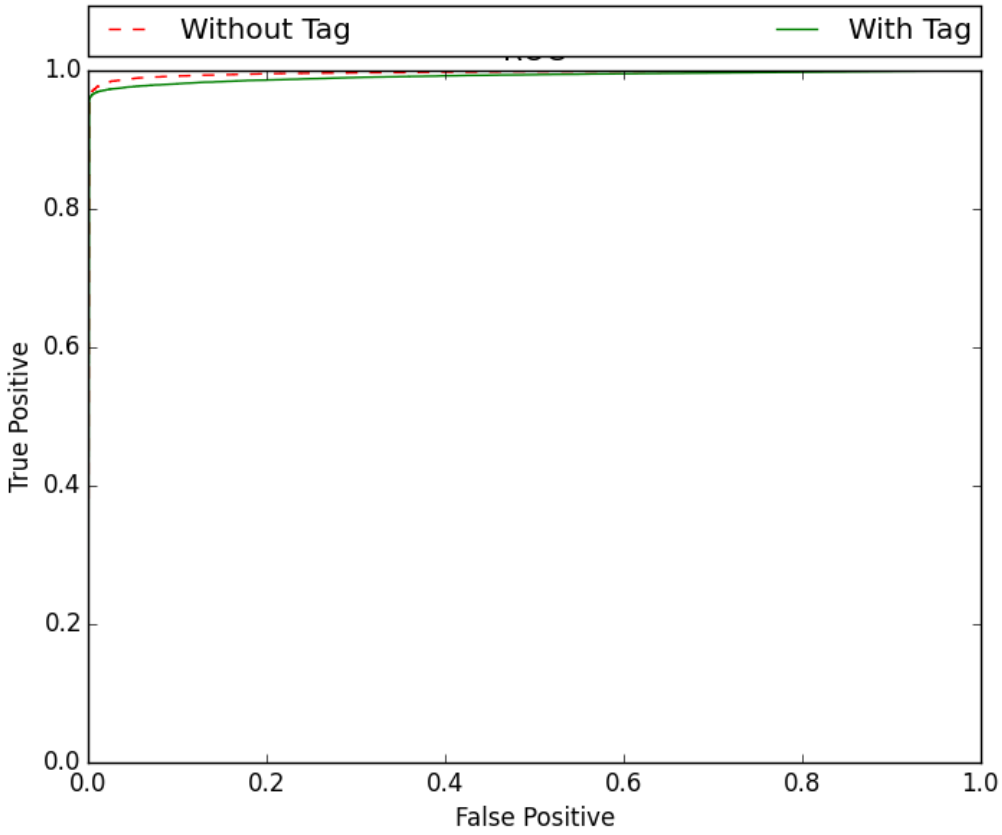


Figure 4. ROC curve for binary classification without tag and with tag for GNOME dataset

Figure 4 shows the data for GNOME for both tag and without tag. We can see from the graphthat these are the best result we have obtained among all four datasets for binary classification. The curve follows closer to the left border and the upper border which means is the most accurate among all four datasets.

5.1.2 Binary classification with tag data

Here, we follow the same procedure as above except for one change. Four extra keywords, extracted with the help of the keyword extraction library, are added to our raw

dataset, making the total number of field count equal to seventeen. The new fields are named tag1, tag2, tag3 and tag4. The keyword extraction algorithm produces a list of important keyword with weights assigned to each keyword. We use our tag extraction script written in python to select the top four most important keywords.

For binary classification with tag data we have got accuracy ranging between 80% to 97%. As we can see in Table 4, accuracy has decreased in all four cases when we add tags and that the running time is higher as compare to without tag data. In these tag datasets, the lowest accuracy obtainedwas for the Mozilla dataset (80.328%). While we have got 96.648% accuracy as highest one in Gnome dataset.

We are exploring accuracy and time is seconds taken to build a model for each dataset with and without dataset in Table 4. This table shows that if we use the extra tags, time increases to build a model while no significant improvement in accuracy is seen.

Table 4. Accuracy and time stamps for both the data with tag and without tag in binary classification

	Time(s)	**Accuracy (%)**	**Time(s)**	**Accuracy (%)**
Dataset	**No Tag**	**No Tag**	**Tag**	**Tag**
Eclipse	13.062	92.449	21.903	86.111
Gnome	8.823	98.025	21.762	96.648
KDE	14.486	82.951	19.271	82.772
Mozilla	21.953	92.714	19.789	80.328

we are comparing three additional measures for each dataset in both conditions with tag and without tag. Values of precision, recall and F-1 measures are reported as almost same in each dataset in both cases, with or without tag.

Table 5, we are comparing three additional measures for each dataset in both conditions with tag and without tag. Values of precision, recall and F-1 measures are reported as almost same in each dataset in both cases, with or without tag.

Table 5. Values of Precision, Recall and F-1 in binary

	No Tag	Tag	No Tag	Tag	No Tag	Tag
Dataset	**Precision**	**Precision**	**Recall**	**Recall**	**F score**	**F score**
Eclipse	0.647	0.645	0.721	0.720	0.654	0.652
Gnome	0.911	0.900	0.826	0.908	0.866	0.899
KDE	0.687	0.685	0.775	0.775	0.707	0.707
Mozilla	0.861	0.862	0.894	0.894	0.857	0.858

5.2 Multi-Class Classification

After performing our experiment on binary classification, we performed a similar experiment on multi-class classification. In multi-class classification, we calculate accuracy, precision, recall and plot ROC curve again for two different cases first is for without tags datasets and the second is by generating tags datasets.

5.2.1 Multi-class classification on without tag data

The same procedure is followed as that for binary classification without tags except for one major change. During the conversion of the csv file, containing bug reports, all the seven severity levels remain as they are and are assigned values from one to seven where one being least severe and seven being most severe.

In Table 6, we can see that the accuracy is varying from 72% to 91% in multi-classclassification in case of no tag. Highest accuracy, we have got for the GNOME dataset (90.942%), while the lowest accuracy is 72.116% for the Eclipse dataset.

In all four graphs for the multi-class classification below, we used two different lines in two colors, red and green. The red color line shows the curve for the without tag dataset while green lines show the results for the with tag dataset. For better test result curve should be closer to the top upper left corner. The closer is the curve to the left upper curve, better the test result would be. The graph is also showing the values for area for both with tag and without tag datasets.

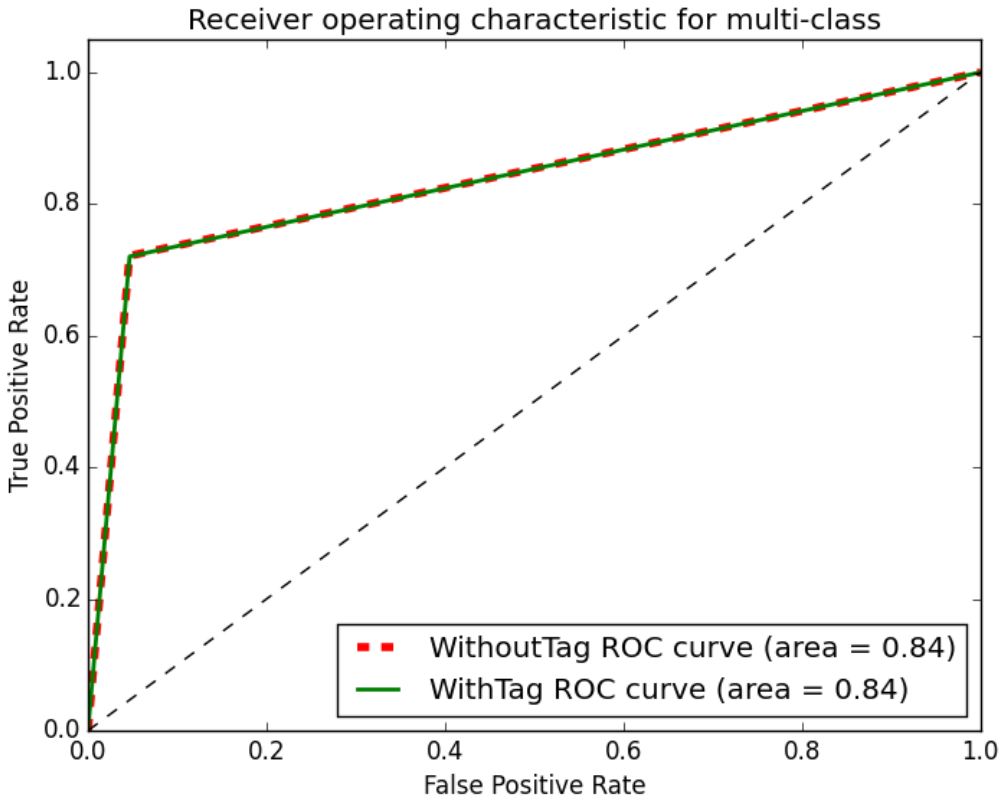


Figure 5. ROC curve for multi-class classification without tag and with tag for Eclipse dataset

Figure 5 shows that without tags and with tag results have same curve. It also shows that both kinds have the same area, which is 0.84.

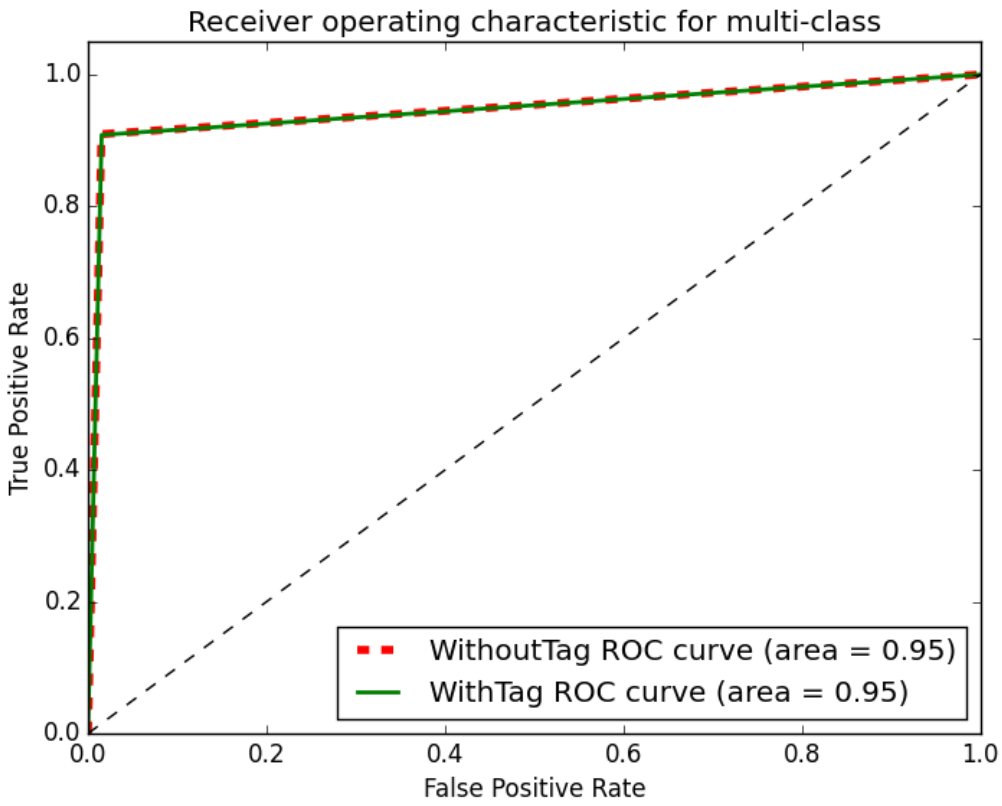


Figure 6. ROC curve for multi-class classification without tag and with tag for the GNOME dataset

Figure 6 shows that without tags and with tag have better curves than the Eclipse dataset curve. It is closer to the upper left corner. It also shows that both kinds have same and much better area (0.95), for both with tag and without tag data.

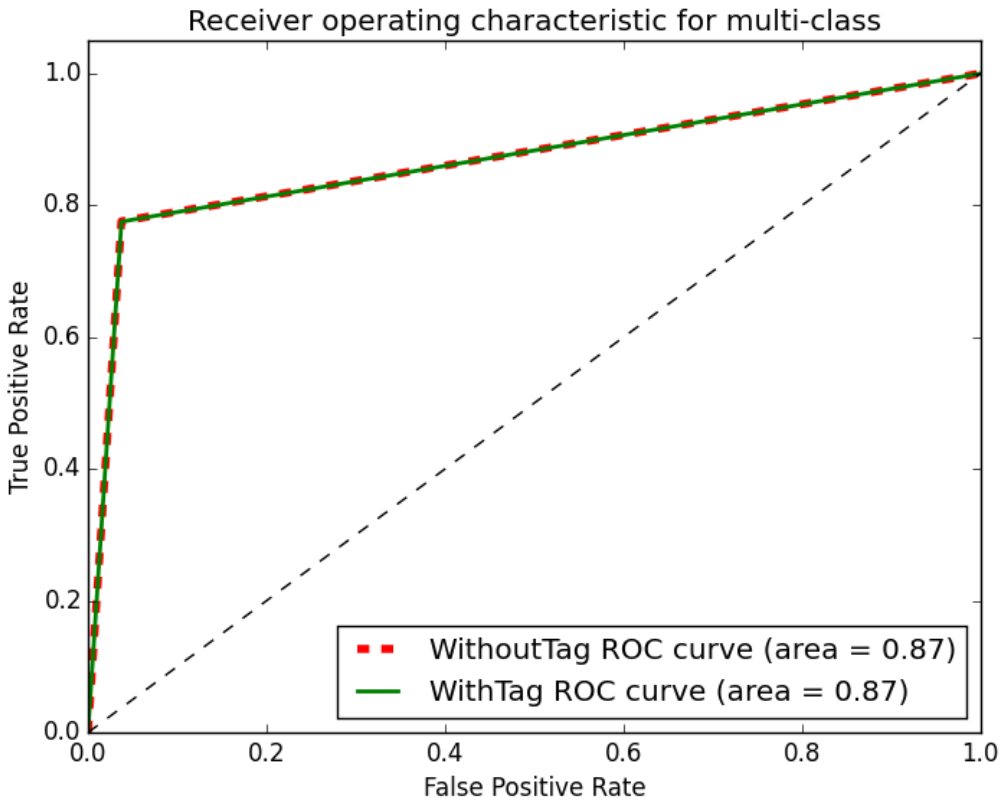


Figure 7. ROC curve for multi-class classification without tag and with tag for KDE dataset

Figure 7 shows that without tags and with tag have better curves than the Eclipse dataset curve, but lesser accurate to gnome curve. It has less area as compared to GNOME dataset graph in both cases whereas; the area under the ROC curve in both cases is 0.87.

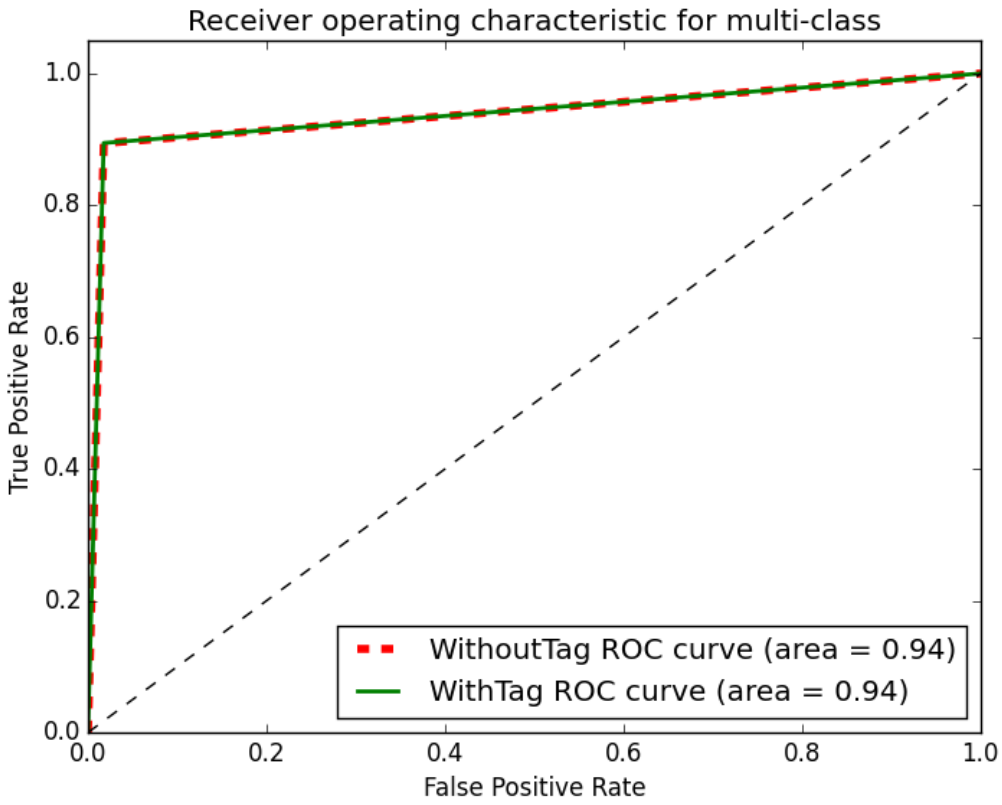


Figure 8. ROC curve for multi-class classification without tag and without tag for the Mozilla Dataset

Figure 8 shows the Mozilla dataset has the second best curve after the GNOME dataset. It has also better area under ROC curve. If we compare it with the GNOME in terms of area, it is only short by 0.01.

5.2.2 Multi-class classification on with tag data

The same procedure is followed as that for binary classification with tags except for one major change. During the conversion of the csv file, containing bug reports, all the seven severity levels remain as they are and are assigned values from one to seven where one being least severe and seven being most severe.

Accuracy improves by slightly less than 1 percent in some dataset. We can see that in Table 6 the accuracy is varying from 72% to91 % in multi-class classification in case of tag. We obtained the highest accuracy in the GNOME dataset, which is 90.826%, while the lowest accuracy is 72.024% in Eclipse dataset.

Table 6 describes the accuracy and running time for all four datasets. It is also comparing the accuracy and time taken to build the model between with tag and without tag dataset. There is not much improvement in time as well as accuracy, however we can see a slight improvement for the Mozilla dataset.

Table 6. Accuracy with or without tag in multiclass

	Time(s)	**Accuracy (%)**	**Time(s)**	**Accuracy (%)**
Dataset	**No Tag**	**No Tag**	**Tag**	**Tag**
Eclipse	3.834	72.116	4.404	72.024
Gnome	6.723	90.942	10.717	90.826
KDE	2.105	77.571	3.102	77.513
Mozilla	7.774	89.409	7.774	89.447

Table 7 is explaining all three measures for all four datasets in both cases with tag and without tag. In both cases, if we analyse precision in tag and no tag, we find that apart from GNOME dataset all other datasets have a better result in the no tag case. For

recall, GNOME and KDE have better result in the tag case. F-1 measure has better result for GNOME dataset in the tag case.

Table 7. values of precision, recall and F-1 in multiclass

	No Tag	Tag	No Tag	Tag	No Tag	Tag
Dataset	**Precision**	**Precision**	**Recall**	**Recall**	**F score**	**F score**
Eclipse	0.51969	0.41402	0.21838	0.16085	0.30754	0.23169
Gnome	0.91101	0.98547	0.82689	0.97135	0.86691	0.97836
KDE	0.90423	0.86935	0.84312	0.86935	0.87261	0.82166
Mozilla	0.65876	0.58921	0.44732	0.42019	0.53283	0.4598

In Table 8, we compare our results with results previously reported by Lamkanfi. We compare the highest value of precision and recall in any case of Lamkanfi research with our highest value of precision and recall in any case. We can see from the table 8 apart from Eclipse dataset (which is also very close to Lamkanfi work) we have good values of precision and recall in both GNOME and Mozilla datasets.

Table 8. Comparison of Precision and Recall between our result and Lamkanfi result

	Lamkanfi		Our Result	
Datasets	**Precision**	**Recall**	**Precision**	**Recall**
Eclipse	0.713	0.738	0.647	0.721
Gnome	0.828	0.842	0.985	0.971
Mozilla	0.752	0.785	0.862	0.894

In Table 9, we compare the area under curve values with Lamkanfi [15] work in any case of the dataset. We can see that for Eclipse, GNOME and Mozilla datasets our results are better than the result of Lamkanfi.

Table 9. Comparison of Area under curve between Lamkanfi and our result

	Lamkanfi	Our Result
Datasets	**AUC**	**AUC**
Eclipse	0.775	0.84
Gnome	0.869	0.95
Mozilla	0.813	0.94

In Table 10, we compared our work for F-1 measure with Tian et al. [5], Valdivia et al. [16] and baseline model in Eclipse and Mozilla open source software system. We can see that in the Tian's research F-1 score is 65.10%, in Valdivia is 15.40% and in baseline model F score is 5.30% that are less than our result 65.20% in Eclipse dataset. On the other hand, in the Mozilla dataset, our result of the F-1 score (85.80%) is much better than the Tian (56.00%), Valdivia (42.10%) and baseline model (20.05%).

Table 10. Comparison of F score between Tian, Valdivia, Baseline model and our result in Eclipse and Mozilla

	Tian	Valdivia	Baselinemodel	Our work
Dataset	**F score**	**F score**	**Fscore**	**F score**
Eclipse	65.10%	15.40%	5.30%	65.20%
Mozilla	56.00%	42.10%	20.05%	85.80%

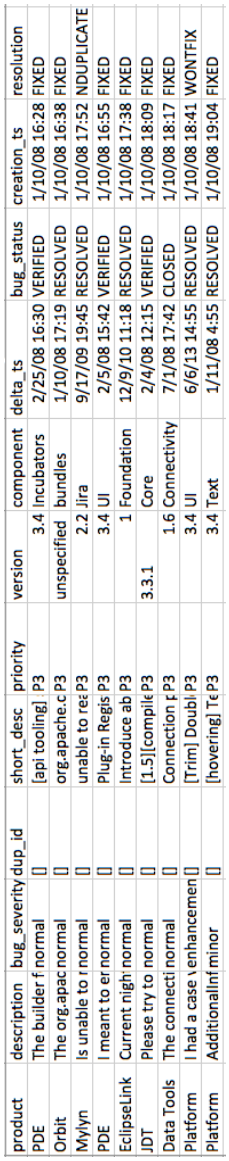
CHAPTER 6

CONCLUSION AND FUTURE WORK

In this thesis, we present an approach that combines feature extraction and machine learning to predict the severity level of a bug report. We are using keyword extraction text-mining algorithm for extracting keywords. With the help of the keyword extraction algorithm, we extracted important keywords from description feature and used four different tags in each binary and multi-class classification case. Then. we trained model on 90% of refined data using machine learning algorithms. After training the model, we tested this model on the 10% refined dataset. This obtained refined dataset, better performace and achieved higher classification accuracy. We validated our approach on bug reports from Eclipse, Mozilla, GNOME and KDE. The average accuracy of our method is greater than 90% and performs better than many other present time machine learning approaches for bug classification. We also compare precision, recall, F-1 score and ROC curves with other research we claim to have improved the performance for each dataset (precision varies from 0.4 up to 0.98, recall varies from 0.16 up to 0.97, F-1 score varies from 0.23 to 0.97 and the highest value of the ROC curve is 0.95). In future, we plan to include a combination of text mining algorithms to see if we can improve our accuracy further. We would also like to test our refined data on new machine learning algorithms to find if significant speed improvements could be achieved.

APPENDIX

A: Dataset used in CSV format



product	description	bug_severity	dup_id	short_desc	priority	version	component	delta_ts	bug_status	creation_ts	resolution
PDE	The builder f	normal	[]	[api tooling]	P3	3.4	Incubators	2/25/08 16:30	VERIFIED	1/10/08 16:28	FIXED
Orbit	The org.apac	normal	[]	org.apache.c	P3	unspecified	bundles	1/10/08 17:19	RESOLVED	1/10/08 16:38	FIXED
Mylyn	Is unable to r	normal	[]	unable to rec	P3	2.2	Jira	9/17/09 19:45	RESOLVED	1/10/08 17:52	NDUPLICATE
PDE	I meant to er	normal	[]	Plug-in Regis	P3	3.4	UI	2/5/08 15:42	VERIFIED	1/10/08 16:55	FIXED
EclipseLink	Current nigh	normal	[]	Introduce ab	P3	1	Foundation	12/9/10 11:18	RESOLVED	1/10/08 17:38	FIXED
JDT	Please try to	normal	[]	[1.5][compile	P3	3.3.1	Core	2/4/08 12:15	VERIFIED	1/10/08 18:09	FIXED
Data Tools	The connecti	normal	[]	Connection p	P3	1.6	Connectivity	7/1/08 17:42	CLOSED	1/10/08 18:17	FIXED
Platform	I had a case v	enhancemen	[]	[Trim] Doubl	P3	3.4	UI	6/6/13 14:55	RESOLVED	1/10/08 18:41	WONTFIX
Platform	Additionalinf	minor	[]	[hovering] Te	P3	3.4	Text	1/11/08 4:55	RESOLVED	1/10/08 19:04	FIXED

B: Converted data in VW format

2 1.0 214966|xsnum digit:0 istart:1 textblock:1 url:0 lines:3 question:0 period:3 finalthanks:0 initcap:3 exclam:0 nonword:5 sent:3 codeblock:0 |yslen code:0 lasttext:439 title:60 text:439 lastcode:0 firsttext:439 firstcode:0 |zsratio esent:0.0 ftext:1.0 qsent:0.0 psent:1.0 fcode:0 ftc:0 tc:0 |wsmean text:439.0 code:0 sent:145.333333333 |short_descwords search for problems api selectively tooling usage more |bodywords and this all reference process into up will api references determine in speed any if information from for to support there extracted add usage prerequisite resolved then allow problems component elementsconditions appears those noreference a restrictions search i builder modified us specific collect previoulsy components were the first

2 1.0 214967|xsnum digit:2 istart:0 textblock:1 url:0 lines:1 question:0 period:1 finalthanks:0 initcap:1 exclam:0 nonword:3 sent:1 codeblock:0 |yslen code:0 lasttext:93 title:42 text:93 lastcode:0 firsttext:93 firstcode:0 |zsratio esent:0.0 ftext:1.0 qsent:0.0 psent:1.0 fcode:0 ftc:0 tc:0 |wsmean text:93.0 code:0 sent:93.0 |short_descwords abouthtml orgapachecommonspool missing |bodywords 13 builds missing abouthtml is orgapachecommonspool bundle an version in the latest

2 1.0 214971|xsnum digit:0 istart:0 textblock:1 url:0 lines:1 question:0 period:0 finalthanks:0 initcap:1 exclam:0 nonword:1 sent:1 codeblock:0 |yslen code:0 lasttext:82 title:55 text:82 lastcode:0 firsttext:82 firstcode:0 |zsratio esent:0.0 ftext:1.0 qsent:0.0 psent:0.0 fcode:0 ftc:0 tc:0 |wsmean text:82.0 code:0 sent:82.0 |short_descwords tasks no edit task to unable reassign permissions if |bodywords a tasks reassing to is actions field unable through in reassign the panel

REFERENCES

[1] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing Mining Algorithms for Predicting the Severity of a Reported Bug," in *2011 15th European Conference on Software Maintenance and Reengineering (CSMR)*, 2011, pp. 249–258.

[2] N. Serrano and I. Ciordia, "Bugzilla, ITracker, and other bug trackers," *IEEE Softw.*, vol. 22, no. 2, pp. 11–13, Mar. 2005.

[3] "Open Source Software and the 'Private-Collective' Innovation Model: Issues for Organization Science," *Organ. Sci.*, vol. 14, no. 2, pp. 209–223, Apr. 2003.

[4] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," in *IEEE International Conference on Software Maintenance, 2008. ICSM 2008*, 2008, pp. 346–355.

[5] Y. Tian, D. Lo, and C. Sun, "Information Retrieval Based Nearest Neighbor Classification for Fine-Grained Bug Severity Prediction," in *2012 19th Working Conference on Reverse Engineering (WCRE)*, 2012, pp. 215–224.

[6] Y. Tian, D. Lo, and C. Sun, "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis," in *2013 IEEE International Conference on Software Maintenance*, Los Alamitos, CA, USA, 2013, vol. 0, pp. 200–209.

[7] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empir. Softw. Eng.*, vol. 20, no. 5, pp. 1354–1383, Aug. 2014.

[8] A. Lazar, S. Ritchey, and B. Sharif, "Improving the Accuracy of Duplicate Bug Report Detection Using Textual Similarity Measures," in *Proceedings of the 11th*

Working Conference on Mining Software Repositories, New York, NY, USA, 2014, pp. 308–311.

[9] "JohnLangford/vowpal_wabbit," *GitHub*. [Online]. Available: https://github.com/JohnLangford/vowpal_wabbit. [Accessed: 16-Feb-2016].

[10] "Stochastic gradient descent," *Wikipedia, the free encyclopedia*. 03-Feb-2016.

[11] L. Bottou, "Stochastic Gradient Descent Tricks," in *Neural Networks: Tricks of the Trade*, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Springer Berlin Heidelberg, 2012, pp. 421–436.

[12] "zelandiya/RAKE-tutorial," *GitHub*. [Online]. Available: https://github.com/zelandiya/RAKE-tutorial. [Accessed: 02-May-2016].

[13] "KDD Cup 2004 - Download PERF Software." [Online]. Available: http://osmot.cs.cornell.edu/kddcup/software.html. [Accessed: 02-May-2016].

[14] "scikit-learn: machine learning in Python — scikit-learn 0.17.1 documentation." [Online]. Available: http://scikit-learn.org/stable/index.html. [Accessed: 02-May-2016].

[15] A. Lamkanfi, S. Demeyer, E. Giger, and B. Goethals, "Predicting the severity of a reported bug," in *2010 7th IEEE Working Conference on Mining Software Repositories (MSR)*, 2010, pp. 1–10.

[16] H. Valdivia Garcia and E. Shihab, "Characterizing and Predicting Blocking Bugs in Open Source Projects," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, New York, NY, USA, 2014, pp. 72–81.