PyBugs: Dataset for Python Bug Prediction

Chhandita Chavan[1], R Sai Shanta[2] Pranjal Mendhekar[2], Eashaan Rao[2] and Prof. Sandeep Patil[2]

[1]International Institute of Information Technology, Pune, India
chavanchhandita@gmail.com

Abstract

Background: Software testing accounts for substantial investments in software development life cycle. It is an iterative process of bug tracking, bug detection and bug prediction for better quality of source codes. There are various ways in which bugs can be predicted from codes. In our project, we consider the structural features of python source codes.
Findings: There are various techniques for static code analysis using bug patterns and software code metrics for object-oriented languages like Java and C++. Although, marginal for languages like python which has seen it's growth in usage since the introduction of data science.
Focus: Our project focuses on creating a dataset comprising of code metrics for a less-researched language: Python. The project uses open source static analysis tools Bandit and Radon for generation of code metrics.

Keywords: Python, Software Metrics, Bandit, Radon, Dataset, Bug Prediction, Software Bugs

Introduction

Software testing is a major aspect of software development life cycle. It is an iterative process of bug detection and bug removal to build an efficient and a bug free end-product. Bug detection and prediction has been a topic of interest for the researchers since decades. Moreover, bug prediction using the structure of source codes has gained much popularity compared to other techniques of bug prediction.

The structure of source codes reveals a lot about the contextual relationship between the components (modules, classes, functions, objects) of the code. Furthermore, abstract syntax trees are a classical way of representing the dependencies and the hierarchy between these components. Although, ASTs have huge importance in bug detection and prediction of bugs, it is also essential to know the parameters which contribute to the generation of bugs in the source codes.

A popular paper written by chidamber and kemerer[5] describes how a set of metrics which they named CK metrics could be useful for bug detection and prediction for java projects. The CK metrics suite contains metrics which predict bug on the basis of parameters such as coupling between objects in a code, depth of inheritance in codes, weighted methods per class etc. Other papers written on bug predictions took into consideration static analysis[3][4] tools and bug patterns[3] which helped in detecting common bug occurrences based on patterns discovered in codes leading to generation of bugs.

Although, the research in code metrics[10][8] and static analysis tools changed the world of software testing, it was only of greater use for programming languages which had higher use of classes, interfaces and objects like java[13]. For programming languages like python, different metrics were introduced for detection of bugs in source codes. These metrics namely lines of codes, halstead metrics[7], McCabe cyclomatic complexity[2] and Maintainibility index parameters were apt for bug detection from structure of python codes. Although, the research has been tremendous and fruitful for java source codes, other budding languages such as python which is currently heavily used for data science have been given marginal importance.

For the same reason and to contribute to such a flourishing open source community, our project emphasizes on bug detection in python source codes.

Related Work

A great amount of work has been done to study the source code metrics which could be used for bug prediction. Some code metrics have been proposed to evaluate the complexity of the source code. There is a significant correlation between size oriented metrics and bugs in the code [10].

The proneness of software bugs depends on measures such as DIT (Depth of Inheritance Tree), WMC (Weighted Methods per Class), CBO (Coupling Between Objects), LoC (Line of Code), etc. Various qualitative and quantitative research methods are combined to propose the dependence of these metrics [6]. Chidamber and Kemerer (CK) metrics were designed from characteristic features of object oriented language such as inheritance, cohesion, and coupling. It is difficult to find a tool that calculates all metrics from the

(CK) metrics suite [5].

Various other software complexity metrics, such as halstead complexity metric and cyclomatic complexity metric have been proposed. The comparison and relationship of various software metrics such as lines of code, halstead metrics, and cyclomatic complexity has been studied [6].
Halstead proposed numerous size metrics such as program vocabulary, length, volume, difficulty, effort and time [7].
McCabe proposed the cyclomatic metric to represent complexity of software products [2]. Cyclomatic metric is computed by the number of nodes, arcs and connected components in control ow graphs of source code. The use of various software metrics has been done in software defect prediction system. One such example is done using NASA metrics data repository giving high accuracy [8]. Bug data for Eclipse projects and open office project are available which can be used for bug prediction [11]. Bug prediction in Java codes has also been proposed which uses several bug patterns [3].

In this paper, a bug dataset creation is proposed for Python programming language. Much work has not been done on Python and so we have created a dataset for Python bug prediction system.

Dataset Development

This dataset is specifically built for prediction of bugs in python source codes. Python files in the dataset have been scraped from top python Github repositories consisting of 100 python projects in total. These projects comprise of 12,318 python files which have been included in the dataset. For each python file in the dataset, corresponding python metrics have been defined with the help of two python static analysis tools namely **Bandit** and **Radon**.
Following are some research questions which our project satisfies:

RQ1: Why use this dataset?

This dataset as specified above focuses majorly on python files. Although bug prediction has been extensively carried out for programming languages such as JAVA and C++, there is a negligible research on languages such as python. Python is a simple and interesting language which has seen increase in its usage since machine learning has become the buzz word of the 21st century. Thus, this dataset defines itself as a benchmark for bug prediction research in python.

RQ2: Which code metrics to use for python source codes?

CK metrics has acted as a breakthrough research for java projects. Similarly, for languages like python where the inter-dependency between classes and modules is less compared to java, code metrics such as raw metrics (LOC, LLOC, SLOC), halstead metrics[7], cyclomatic complexity[2] and maintainability index prove efficient for bug prediction. The above mentioned metrics are calculated using a static analyzer for python known as Radon.
Lines of code help in determining the complexity of a code. The greater the lines of code, the greater there is a chance of occurrence of bugs.
Halstead metrics on the other hand identify measurable properties in a source code analogous to those of matter (volume, density, mass, pressure) and thus cannot be categorized fully as complexity metrics.
The halstead metrics comprise of the following parameters: *Program vocabulary, Program length, Calculated program length, Volume, Difficulty, Effort, Time required to program, Number of delivered bugs*
The next metric i.e. cyclomatic complexity is a software metric, used to indicate the complexity of a program. It is a quantitative measure of the number of linearly independent paths through a program's source code.
Maintainability Index is a software metric which measures how maintainable (easy to support and change) the source code is. The maintainability index is calculated as a factored formula consisting of Lines Of Code, Cyclomatic Complexity and Halstead volume.

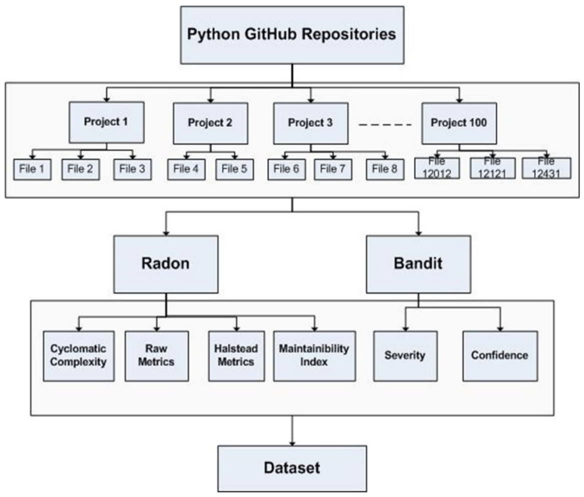


Figure 1: Dataset Development Process

RQ3: Finding bug probability and severity?

Software bugs severity assessment is essential for developers to prioritize bug reports[1]. Our proposed model of dataset aims to be used for bug prediction. Bug prediction using this model gives rise to bug severity and bug probability or confidence as its target variables.
Here, the calculation of severity and confidence is done

using a tool called Bandit. Bandit is an open-source static analyzer for performing security analysis. Bandit utilizes the ast module from the Python standard library and calculates the target variables namely severity and confidence from python codes.
There are four attributes of each severity and confidence, namely undefined, low, medium and high which signifies number of files under each of these attributes.

The Dataset

The dataset consists of the following parameters:
Input Parameters: *File Name, File Path, LOC(Lines of Code), LLOC(Logical Lines of Code), SLOC(Source Lines of Code), Halstead Volume, CC(Cyclomatic Complexity), h(Program Vocabulary), N(Program Length), Calculated Program Length*
Output Parameters: *Severity and Confidence*

The files in the dataset can be classified into buggy and non-buggy files on the basis of severity and confidence. We have considered non-zero values of severity and confidence as buggy files and non-buggy otherwise.
The dataset consists of 10,007 non-buggy files and 2,311 buggy files as shown below.

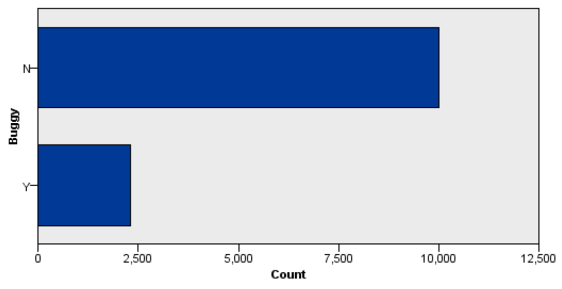
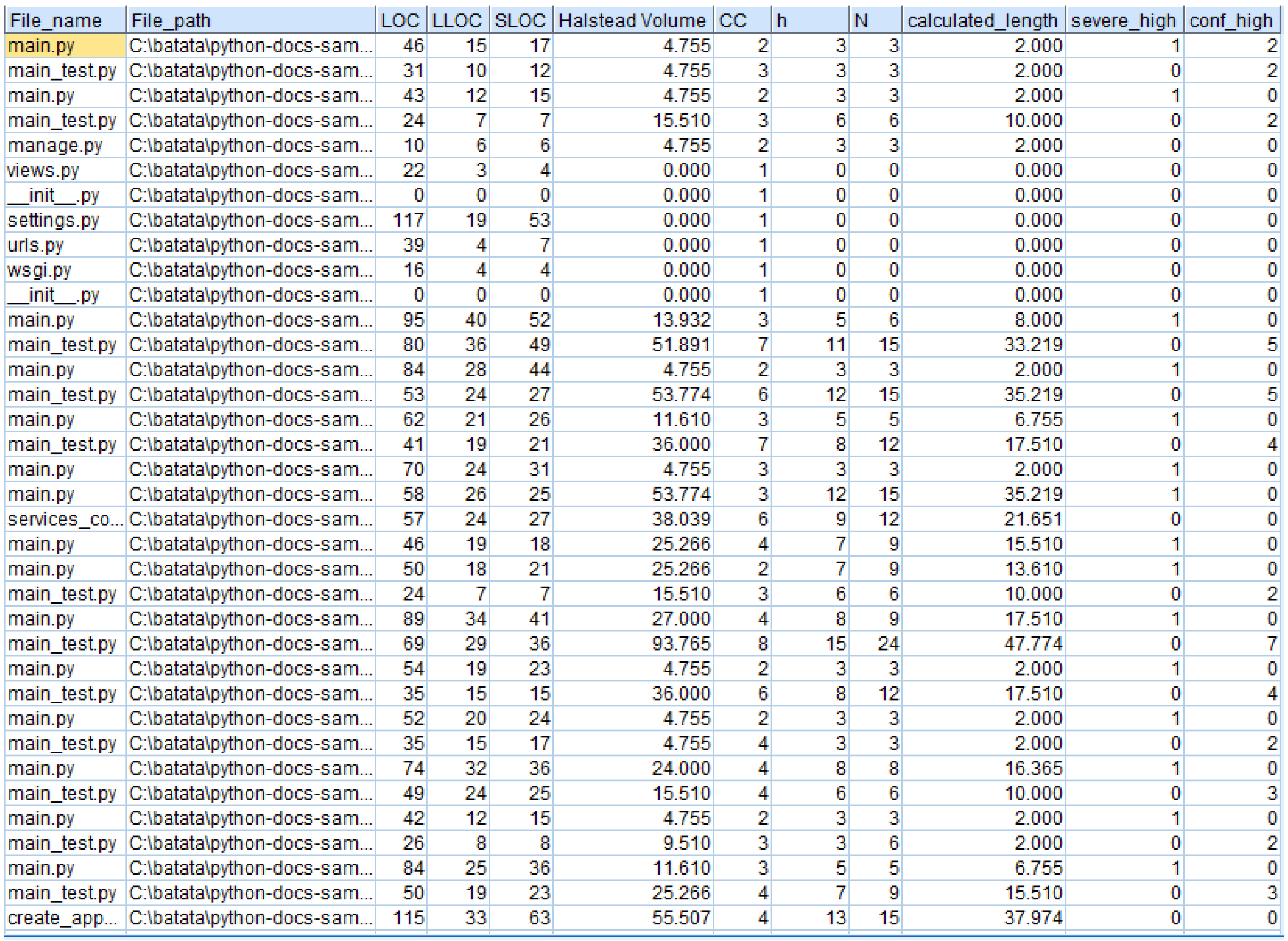


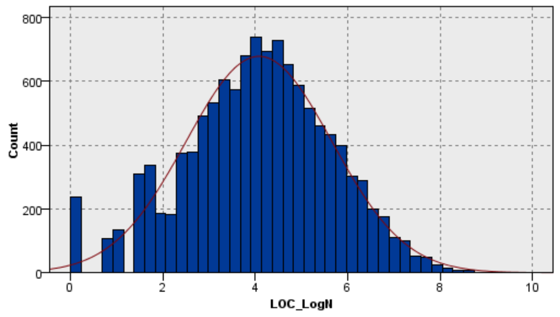
Figure 2: Distribution of Buggy files



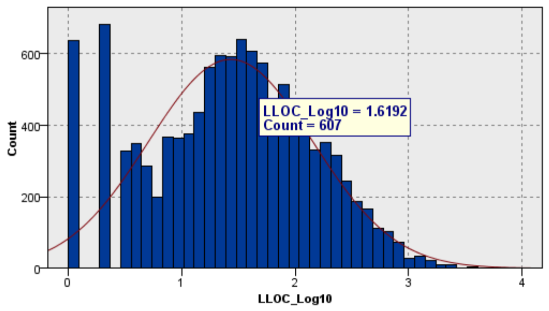
File_name	File_path	LOC	LLOC	SLOC	Halstead Volume	CC	h	N	calculated_length	severe_high	conf_high
main.py	C:\batata\python-docs-sam...	46	15	17	4.755	2	3	3	2.000	1	2
main_test.py	C:\batata\python-docs-sam...	31	10	12	4.755	3	3	3	2.000	0	2
main.py	C:\batata\python-docs-sam...	43	12	15	4.755	2	3	3	2.000	1	0
main_test.py	C:\batata\python-docs-sam...	24	7	7	15.510	3	6	6	10.000	0	2
manage.py	C:\batata\python-docs-sam...	10	6	6	4.755	2	3	3	2.000	0	0
views.py	C:\batata\python-docs-sam...	22	3	4	0.000	1	0	0	0.000	0	0
__init__.py	C:\batata\python-docs-sam...	0	0	0	0.000	1	0	0	0.000	0	0
settings.py	C:\batata\python-docs-sam...	117	19	53	0.000	1	0	0	0.000	0	0
urls.py	C:\batata\python-docs-sam...	39	4	7	0.000	1	0	0	0.000	0	0
wsgi.py	C:\batata\python-docs-sam...	16	4	4	0.000	1	0	0	0.000	0	0
__init__.py	C:\batata\python-docs-sam...	0	0	0	0.000	1	0	0	0.000	0	0
main.py	C:\batata\python-docs-sam...	95	40	52	13.932	3	5	6	8.000	1	0
main_test.py	C:\batata\python-docs-sam...	80	36	49	51.891	7	11	15	33.219	0	5
main.py	C:\batata\python-docs-sam...	84	28	44	4.755	2	3	3	2.000	1	0
main_test.py	C:\batata\python-docs-sam...	53	24	27	53.774	6	12	15	35.219	0	5
main.py	C:\batata\python-docs-sam...	62	21	26	11.610	3	5	5	6.755	1	0
main_test.py	C:\batata\python-docs-sam...	41	19	21	36.000	7	8	12	17.510	0	4
main.py	C:\batata\python-docs-sam...	70	24	31	4.755	3	3	3	2.000	1	0
main.py	C:\batata\python-docs-sam...	58	26	25	53.774	3	12	15	35.219	1	0
services_co...	C:\batata\python-docs-sam...	57	24	27	38.039	6	9	12	21.651	0	0
main.py	C:\batata\python-docs-sam...	46	19	18	25.266	4	7	9	15.510	1	0
main.py	C:\batata\python-docs-sam...	50	18	21	25.266	2	7	9	13.610	1	0
main_test.py	C:\batata\python-docs-sam...	24	7	7	15.510	3	6	6	10.000	0	2
main.py	C:\batata\python-docs-sam...	89	34	41	27.000	4	8	9	17.510	1	0
main_test.py	C:\batata\python-docs-sam...	69	29	36	93.765	8	15	24	47.774	0	7
main.py	C:\batata\python-docs-sam...	54	19	23	4.755	2	3	3	2.000	1	0
main_test.py	C:\batata\python-docs-sam...	35	15	15	36.000	6	8	12	17.510	0	4
main.py	C:\batata\python-docs-sam...	52	20	24	4.755	2	3	3	2.000	1	0
main_test.py	C:\batata\python-docs-sam...	35	15	17	4.755	4	3	3	2.000	0	2
main.py	C:\batata\python-docs-sam...	74	32	36	24.000	4	8	8	16.365	1	0
main_test.py	C:\batata\python-docs-sam...	49	24	25	15.510	4	6	6	10.000	0	3
main.py	C:\batata\python-docs-sam...	42	12	15	4.755	2	3	3	2.000	1	0
main_test.py	C:\batata\python-docs-sam...	26	8	8	9.510	3	3	6	2.000	0	2
main.py	C:\batata\python-docs-sam...	84	25	36	11.610	3	5	5	6.755	1	0
main_test.py	C:\batata\python-docs-sam...	50	19	23	25.266	4	7	9	15.510	0	3
create_app...	C:\batata\python-docs-sam...	115	33	63	55.507	4	13	15	37.974	0	0

When the parameters in the dataset are plotted (graphs transformed to parameter*log10) following are the statistics obtained:

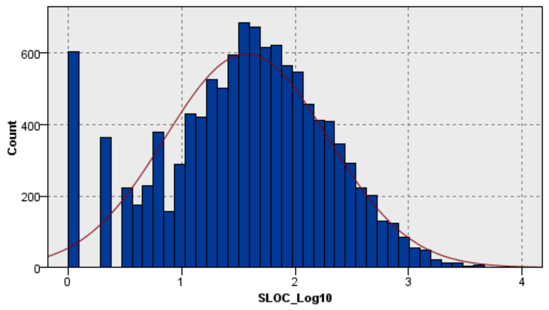
LOC ranged from 0 to 7,827 with a mean value of 171.827 and standard deviation 373.798



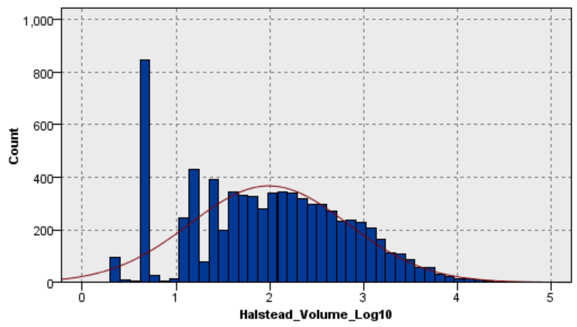
LLOC ranged from 0 to 4,060 with a mean value of 85.580 and standard deviation 194.899



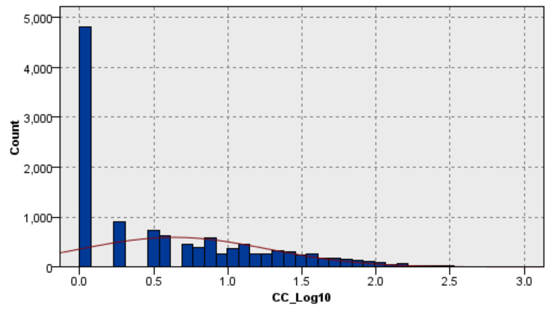
SLOC ranged from 0 to 4,614 with a mean value of 111.525 and standard deviation 246.143



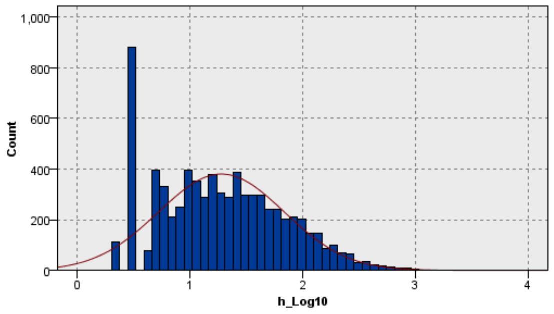
Halstead Volume ranged from 0.000 to 24591.903 with a mean value of 321.012 and standard deviation 1165.113



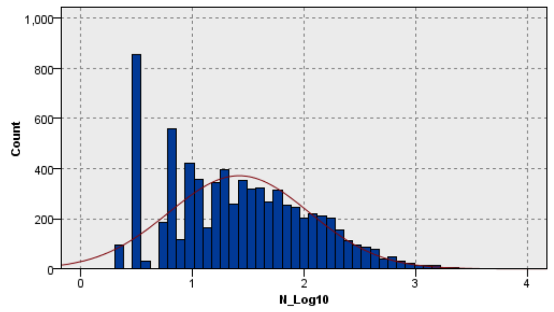
CC ranged from 1 to 962 with a mean value of 15.516 and standard deviation 41.827



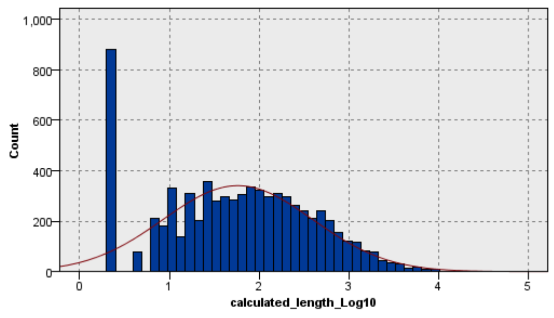
h ranged from 0 to 1265 with a mean value of 27.203 and standard deviation 68.899



N ranged from 0 to 2422 with a mean value of 46.309 and standard deviation 132.213



Calculated Length ranged from 0.000 to 12899.513 with a mean value of 169.631 and standard deviation 587.286



Result

We used IBM SPSS modeler to carry out the experimentations on our dataset due to its easy drag and drop feature in the user interface. The modeler gave us various results which are discussed below.

RQ4: Which input parameters are more dependent on the output parameters?

Considering severity and confidence as output parameters (dependent variables), there are 14 input variables (independent variables) out of which six variables namely cyclomatic complexity, LOC, LLOC, SLOC, halstead volume and maintainability index are highly correlated to the target variables.

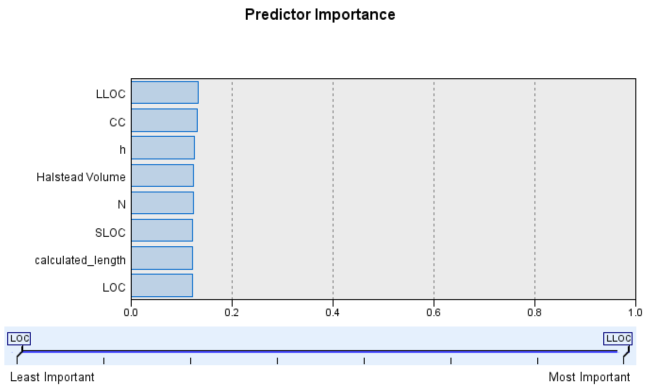
RQ 5: Which model satisfies the underlying data?

Depending upon the behavior of the data, corresponding model has to be chosen to find an apt model which will be able to train the data most accurately.

Models according to overall accuracy (Highest to lowest) Training the underlying dataset with 10 different models in IBM SPSS modeler, the top contenders in terms of overall accuracy are C5 decision tree, CART, CHAID, Quest 1, Multi-layer perceptron networks and logistic regression.

RQ 6: Which software metrics have more importance?

The following figure shows the importance of the processed metrics in the prediction of software bugs. The result shown is for C5 1 model which has the highest accuracy of all the models.It can be concluded that LLOC has highest significance while LOC has the least significance for C5 1 model.



Model	No. of Input Parameters provided	No. of Input parameters used	Input parameters used	Max Profit	Area Under Curve	Overall Accuracy (Percentage)
C5 1	8	8	NIL	1797.347	0.834	84.226
C & R Tree	8	8	NIL	1110.593	0.777	83.122
CHAID 1	8	7	h (Program Vocabulary)	1000.700	0.829	82.871
Quest 1	8	8	NIL	972.941	0.641	82.854
Multi-Layer Perceptron	8	8	NIL	995.0	0.808	82.854
Logistic Regression	8	8	NIL	935.0	0.799	82.814
Bayesion Network	8	8	NIL	540.0	0.538	82.156
Linear Discriminant Analysis	8	8	NIL	935.0	0.802	78.6
Decision Tree	8	3	LOC, LLOC, SLOC, Calculated length, Halstead volume	636.302	0.764	70.068

References

[1] Ahmed Fawzi Otoom, Doaa Al-Shdaifat, Maen Hammad, Emad E. Abdallah, *Severity Prediction of Software Bugs*, 7th International Conference on Information and Communication Systems (ICICS), IEEE, 2016, pp. 92-95.

[2] Thomas J McCabe, *A Complexity Measure*, IEEE Transactions On Software Engineering, IEEE, Washington, Oct 1976, pp. 308-320.

[3] David Hovemeyer, William Pugh,FindingBugs is easy, OOPSLA, ACM Journal, 2004, pp.24-28.

[4] Mahdi Nasrullah Al-Ameen, Md.Monjurul Hasan, Asheq Hamid, Making Findbugs More Powerful,IEEE, 2011, pp. 705-708.

[5] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. IEEE Trans. Softw. Eng., 1994, pp. 476493.

[6] Sheng Yu, Shijie Zhou, A Survey on Metric of Software Complexity, IEEE, 2000.

[7] M. H. Halstead. Elements of Software Science (Operating and Programming Systems Series),Elsevier Science Inc., New York, USA, 1977.

[8] S.M. Fakhrahmad, A.Sami , Effective Estimation of Modules Metrics in Software Defect Prediction, Proceedings of the World Congress on Engineering, Vol I, London, U.K., 2009.

[9] Mohamad Mahdi Askari and Vahid Khatibi Bardsiri, Software Defect Prediction using a High Performance Neural Network, International Journal of Software Engineering and Its Applications Vol. 8, 2014, pp. 177-188

[10] Varuna Gupta, Dr. N. Ganeshan , Dr. Tarun K. Singhal, Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicators, International Journal of Advanced Computer Science and Applications, Vol. 6, No.2, 2015, pp. 60-65.

[11] Meera Sharma, Punam Bedi, K.K. Chaturvedi, V.B. Singh, Predicting priority of bug prediction using ML techniques and cross project validation, IEEE, 2012, pp.539-545.

[12] Ruchika Malhotra and Ankita Jain, Fault Prediction Using Statistical and Machine Learning Methods for Improving Software Quality, Journal of Information Processing Systems, Vol.8, No.2, 2012, pp. 241-262.

[13] Nick Rutar, Christian B. Almazan, Jeffrey S. Foster, *A Comparison of Bug Finding Tools for Java*, published in the Proceedings of the ISSRE '04, 15th International Symposium on Software Reliability Engineering, 2004.

[14]Radon, `Available from: https://radon.readthedocs.org/`

[15]Bandit, `Available from: https://pypi.python.org/pypi/bandit`